UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Science and Technology
Department of Computer Science

StudentLink

A Generic Calendar Platform

—

Ruben Alexander Andreassen

INF-3990 Master’s Thesis in Computer Science, May 2015

Abstract

Large organizations often use many software applications. In some cases, two or more of these applications contains information that is very similar or the same. In this thesis we are going to look at calendar information. UiT - The Arctic University of Norway is a large organization which uses many applications that contains calendar information. Fronter, Syllabus and Microsoft Exchange all covers different needs of the organization, but it is challenging to get a fast and easy overview of calendars from several of the systems at the same time. By creating a generic calendar platform that communicates with these systems, we get a new combined calendar system that contains every calendar within the organization. This platform implements many interfaces to suit the needs of an end user or end user application. The result is that the end user gets everything in one place, without the need to access several different portals and manually compare calendars. The generic calendar platform is also easily extensible, meaning that it could reach outside the organization.

Table of Contents

List of Figures

List of Tables

Glossary

University	When talking about the university, we are always referring to UiT - The Arctic University of Norway.
Room	A teaching room, meeting room, or any other physical room defined as a resource at the university. It is possible to allocate such a room; therefore it has a calendar that is relevant to this paper. Private offices etc. are not a part of this definition.
Room list	A group of rooms. A room list can contain a group of rooms that is physically located at the same institute or building, but does not need to.
Web Service	When referring to the Web Service, we mean in most cases the generic calendar platform that we have developed in this thesis.

1 Introduction

Large organizations often use many types of software systems such as a variety of applications, databases and Web Services to accommodate the different needs of the organization. When using different systems to solve different tasks, it is not uncommon that some of the systems contains similar types of information. Some users in the organization are super users and use specific systems on a daily basis, and other users only use the different systems occasionally. In some, cases users want to retrieve information from several applications that are similar. The last case can be time consuming if the users need to hop in and out of several applications and web sites to get an overview of the information they are looking for. The task can become even more difficult if the user only have little or no training in using the software. There can also be situations where the user has a right to know about the information, but don't have direct access to the software containing that information. This can be the case if the software don't have advanced access management. This would mean that the user would get access to view more information than intended, and that is often not a good solution. Another alternative would be to hand out printouts of the relevant information. We are going to look at a solution that eliminates the need of using several different systems when the main goal is information retrieval of similar types of information.

A common way to retrieve similar information from different software systems is to create middleware. When we are referring to middleware me mean the creation of a software that take similar types of information as input, and outputs the same information in a combined format. The input format can have different data structure and semantics. The middleware solves the interoperability problem between the different software systems and the end user or end user software. This raises a lot of challenges for the developers of the middleware in terms of handling different API's, communication protocols, interoperability and semantics, but the reward for the user can be significant.

One architecture of middleware is Web Services. With Web Services we mean a method of communication between two electronic devices over the World Wide Web (WWW). In this case between several types of software and the end user software. A Web Service is highly flexible because it supports many types of communication protocols over the network.

In this thesis, we will create a generic calendar platform in the form of a Web Service. The Web Service will be able to retrieve calendar information from several different types of software, and deliver the retrieved information to the user. The user will be able to access the Web Service directly through a web page, or other developers can develop 3rd party applications and retrieve information from the API that the Web Service offers. In addition, since we are dealing with calendar information, the Web Service will provide a function to share public calendar information with other already established calendar systems such as Google Calendar and Outlook Web Access.

We will only focus on information retrieval. This means that the Web Service will only be able to retrieve information from the external software that the Web Service is communicating with. The Web Service will not be able to add or modify any information. This design issue was made due to the limited time we have to develop and write this thesis. However, the Web Service could be extended to allow such operations. Access rights could be an issue when extending the Web Service in that direction.

1.1 Problem statement

We shall develop a generic calendar platform that interacts with several external systems using different protocols and data formats. The focus will be on data retrieval and combining the retrieved data. The generic calendar platform shall not be able to add or update any data back to the external systems.

The generic calendar platform must have one connector for each external system that the platform is going to communicate with and retrieve information from. It shall be simple to expand the platform with new connectors that can communicate with other calendar systems that we have not implemented in this thesis.

The generic calendar platform must implement middleware to be able to combine the retrieved data into one internal representation. The middleware must understand the semantics of the data from each external system. This makes it easy to add more connectors to integrate even more external systems in the future. The connector only need to implement a method to parse the retrieved data into the internal representation, then the platform will take care of the rest.

We shall also implement one or more interfaces that delivers the combined data in one or more formats to the end user or end user software.

1.2 Method

The method we use in this thesis is software prototyping. This means that we develop a prototype of the generic calendar platform that is functional. The platform may not be complete, but good enough to get a feel for how the finished product would look like, and some of the functionality the finished product could have.

There are two dimensions when developing a prototype. There is the horizontal prototype which focuses on the user interaction rather than low-level functionality, and there is the vertical prototype which is a more complete elaboration of a single subsystem (Nielsen, 1993). We use the latter type of prototype. We will implement all the components that is needed from information retrieval to the delivery of the combined information.

There are many variants when dealing with software prototyping, but all of these are mostly based on two major types; Throwaway prototyping and Evolutionary prototyping (Davis, 1992). Throwaway prototyping is when a prototype is made that will eventually be discarded. The prototype is not intended to become a part of the final software. Evolutionary prototyping on the other hand, is when a robust prototype is made in a structural manner that eventually the prototype becomes the finished product. In this thesis we use evolutionary prototyping. This makes it possible to add features and make changes that are not planned during the design phase. We also focus on one part of the system at the time. Interaction between the generic calendar platform and a specific external system can be one part, or combining information retrieved from the external systems can be one part. The prototype will therefore evolve into a more and more finished product during the development process.

We will also develop a GUI that we can use in the testing and evaluation of the prototype. By setting up a use case we can determine if the generic calendar platform function and performs as intended.

1.3 Summary

In this thesis, we develop a generic calendar platform that is able to retrieve data from four different external systems. The systems are Active Directory (AD), Microsoft Exchange, Syllabus and Fronter. All systems except AD contains calendar information, and this is the data we want to combine into one combined calendar on our platform.

To do this, we need to develop connectors that communicates with each system we want to retrieve information from. These connectors need to be developed according to each API specification. This means that we need to have in depth knowledge about each system. Documentation is our friend here. We need to use several different protocols such as HTTP, LDAP and TNS. Also, there are several different data formats when communicating with the different systems such as SOAP, JSON and iCalendar.

When the communication is in place and we are able to retrieve data, we need to find out the semantics of the data. Since we are dealing with calendar data, the data contains more or less the same information such as subject, description, location, date and time. But all the systems do not use the same description for the same thing. For example, location in Exchange and room in Syllabus are different terms of the same thing.

The last phase is to display the result to the user. We develop a graphical user interface in form of a web page that has all of the functionality of the platform. In addition to this, we implement a REST-based API for other developers to use. We will also implement a way to share public calendar information so that any user can easily add calendar data in well-known calendar systems such as Google Calendar and Outlook Web App (OWA).

The testing shows that our platform is a useful tool for users that are using two or more of the external systems that we are implementing. Instead of having to access several different systems and check several calendars, we have created a single portal that is easy and time saving to use. The user experience is also greatly improved compared to using several of the external systems at the same time.

1.4 Outline

Chapter 2 covers all the background information needed to understand the Design and Implementation phase. Since we are integrating external systems, it is important to understand how they work and how to connect to them.

Chapter 3 describes the design phase. We make some choices on how we think is the best way to implement the external systems, how to combine the data retrieved, and how to deliver the data to the end user.

Chapter 4 gives a very detailed overview of how we implemented each part of the platform. The chapter covers much of the hurdles we met when integrating with external systems.

Chapter 5 shows the performance of the generic calendar platform and compares the platform to the university's already existing solutions.

Chapter 6 evaluates our findings and looks outside the organization.

Chapter 7 concludes the thesis and suggests further work.

2 Background

In this thesis we are creating a generic calendar platform in the form of a Web Service that retrieves calendar information from several external systems, combine the data where possible and deliver the combined data to the end user or end user application. We therefore need to give an introduction to each of the external system that we are going to retrieve information from. In addition, we need to cover what a Web Service is and the common types of interfaces that a Web Service can use.

2.1 Service Oriented Architecture (SOA)

When creating a Web Service, Service Oriented Architecture (SOA) comes into mind. SOA means that the interface of the software is independent of any hardware, operating system or any underlying software. The SOA makes it possible for services to exchange information over the network without any human interference (Microsoft, Service-Oriented Architecture (SOA), 2015).

This becomes very important when we create software that is going to communicate with other 3rd party software. If the 3rd party software we are trying to communicate with don't have clean interface that we can communicate with, it can be very difficult to interact with. This also applies to our own software. We want to deliver the retrieved information to others, and it needs to be clean and simple.

2.2 Web Service

A Web Service is a method to enable two electronic devices to communicate with each other over the World Wide Web (WWW) (Booth, et al., 2004). The Web Service is most likely to use underlying components, and creates an abstraction layer between the application code and the application client. This is often referred to as middleware. The Web Service are loosely coupled with the underlying components.

Common internet protocols, such as HTTP, makes sense to use when the main goal is to make information available to a variety of users and applications. A Web Service is highly flexible and can be written in a variety of languages. PHP, Java, C#, Ruby and Python to mention some. It makes sense to develop a Web Service over a desktop or mobile application for the task we have at hand. In the end, our Web Service can deliver information to any application that other developers might want to create.

2.2.1 World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF)

The W3C is an organization that develops protocols and guidelines to ensure the long-term growth of the Web. They are an important driving force when it comes to creating open standards and design principles (W3C, 2014).

The mission of the IETF is to make the Internet work better by producing high quality, relevant technical documents that influence the way people design, use, and manage the Internet (Internet Engineering Task Force, 2015).

We mention them here because it is impossible to make anything on the web without noticing the W3C and IETF.

2.2.2 Simple Object Access Protocol (SOAP)

SOAP is a packaging structure for sending XML documents over a variety of different transport protocols such as HTTP, TCP, UDP and SMTP (Gudgin, et al., 2007). However, the most common transport protocol to use when working with SOAP is HTTP. SOAP can be used for remote procedure call (RPC) client-server applications, but can also be used for one-to-many applications, such as messaging systems.

When developing an API to a Web Service, SOAP is a common protocol to use when exchanging data. In this thesis, we will not implement a SOAP based API for the 3rd party applications that wants to use the Web Service. This is because SOAP is very complex and would be overkill for our lightweight Web Service. However, one of the systems we are integrating with uses SOAP to exchange data. This means that we need to implement a SOAP client to retrieve information from that system.

2.2.3 Representational State Transfer (REST)

REST is an architecture that relies on the transport protocol HTTP (Fielding & Taylor, 2002) (Fielding R. T., 2000). In comparison to SOAP, REST can only be used by client-server applications. Web Service API's that uses REST are known as RESTful. REST uses HTTP operations (request methods), and there are four methods that is mostly used. The notably methods are described in Table 2-1.

Since our Web Service will only be dealing with information retrieval we only need to implement the GET and POST method. We will use the POST method for authentication because we don't want the user's credentials to be listed in the URL.

RESTful API HTTP methods				
Resource	**GET**	**PUT**	**POST**	**DELETE**
Collection URI (http:// example.com/ resources)	**List** the URIs and perhaps other details of the collection's members	**Replace** the entire collection with another collection	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation	**Delete** the entire entry
Element URI (http:// example.com/ resources/ item17)	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type	**Replace** the addressed member of the collection, or it if doesn't exist, **create** it	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it	**Delete** the addressed member of the collection

Table 2-1 Description of the most common REST methods

2.2.4 Enterprise Mashup

"Mashups solve the quintessential information sharing problem: accessing and combining data from disparate internal and external data sources in ways that where not pre-imagined." (Crupi & Warner, 2009).

The previous quotation is a good explanation of a mashup, and one could imagine that our solution is exactly this, a mashup. We combine information from both internal (Syllabus, Exchange) and external (Fronter) sources in a way that none of the sources could have pre-imagined. The end result could be a mashup in the form of a web page or other 3rd party application.

But this statement has left out one important detail. A mashup is created from modular components that the end user can assemble and reassemble as desired to serve current needs. The Web Service that we have created could not be assembled easily without special expertise in the field of programming. We can therefore not consider the end result as a mashup of any kind.

2.3 Calendar sharing

There are several ways to share a calendar. The most advanced way is to create a CalDAV server. CalDAV is an extension of WebDAV that allows clients to access and manage a calendar on a remote server (Daboo, Desruisseaux, & Dusseault, Calendaring Extensions to WebDAV (CalDAV), 2007). When a calendar is shared using CalDAV, any user that has access can do DELETE operations to delete a single event, or PUT operations to change an event. And many other operations that we do not mention here. The way that this is done is that there is one URL for the whole calendar, and one child URL for each event. Similar to a REST API. There exist many extensions to CalDAV, among them a scheduling extension to plan meetings (Daboo & Desruisseaux, Scheduling Extensions to CalDAV, 2012).

If the goal is to just share a calendar, a common approach is to simply make iCalendar files accessible over HTTP. When this is done, one URL represents a full calendar with all events. There are two ways of doing this. The user could export a calendar from one system, getting an iCalendar file, and then import that file into another system. The user would need to do the same process each time the exported calendar changes.

The second approach is to share an iCalendar file by URL. Then, the user can provide the shared URL into a calendar system that supports to add calendars by URL. The system that is importing the calendar would manage the changes of the iCalendar file. Note that this is only a one-way synchronization compared to the two-way that a CalDAV server would support.

A side note to basic calendar sharing is the Webcal URL scheme (IANA, 2012). This is an experimental scheme that is used to tell an application, usually a web browser, that the URL should be handled by a different application. This would be the same as taking HTTP URL and add it manually in the program that is set up to handle the Webcal URLs.

2.4 Single sign-on

Single sign-on (SSO) is a technique for access control of multiple related, but independent, systems (Huntington, 2015). When using SSO, a user logs in once to gain access to all the systems. There are several ways of implementing SSO, from the usage of LDAP to cookies (Cugley, 2007). If cookies is used, all the sites must be on the same domain.

However, this type of SSO would not help in our case. Assume that all the systems we are implementing supports the same type of SSO. If the user logs on to our Web Service, he will automatically be authenticated on the other systems. But this only means that the user can browse from our system to another system without the need to authenticate one more time. In our case, it is the Web Service that retrieves information from the external systems, meaning that the Web Service is acting on behalf of the user. This means that the Web Server needs to authenticate to the external systems on the users behalf. There are many papers of multiparty authentication, and the common understanding of these is that multiparty authentication is complex to implement (IBM Corporation & Microsoft Corporation, 2002).

We have decided on a much simpler scheme since this thesis does not have authentication as the main focus. The user must send his credentials to the Web Service on each request. Then the Web Service can use these credentials to authenticate against the external systems. The credentials would only exist on the Web Service during the execution of a script, and never cached. The Web Service could only be compromised between when the Web Service receives the credentials and before the script has finished execution. We leave it up to the 3rd party applications that uses the interfaces of the Web Service to handle the user's credentials securely.

2.5 HTTP Status Codes

RFC 7231 contains information about HTTP Semantics and Content, including status codes (Fielding & Reschke, 2014). Some status codes are well known as we see them more or less every day. Example of these are 200 OK, 404 Not Found and in some cases 500 Internal Server Error. We are not going to describe these in detail, but we are going to look at the 302 Found status code that is relevant to understand when looking at the Fronter login flow in Appendix A.

2.5.1 302 Found

The 302 Found status code indicates that the target resource resides temporarily under a different URI. Since the redirection might be altered on occasion, the client ought to continue to use the effective request URI for future requests.

The server SHOULD generate a Location header field in the response containing a URI reference for the different URI. The user agent MAY use the Location field value for automatic redirection. The server's response payload usually contains a short hypertext note with a hyperlink to the different URI(s).

Note: For historical reasons, a user agent MAY change the request method from POST to GET for the subsequent request. If this behavior is undesired, the 307 (Temporary Redirect) status code can be used instead (Fielding & Reschke, 2014).

2.6 Microsoft Active Directory (AD)

The Microsoft Active Directory (AD) is a special-purpose database. The data structure is hierarchical and it is replicated and extensible. The directory is designed to handle a large number of read and search operations and a significantly smaller number of changes and updates. Since the design of the AD do not handle many changes and updates, it is not suitable for dynamic data. Examples of data stored in the directory are user profiles, computer configuration data and other resource objects (Microsoft, So What Is Active Directory?, 2014).

Term	**Description**
GUID	Globally Unique Identifier
RDN	Relative Distinguished Name
CN	Common Name
OU	Organizational Unit Name
DC	Domain Component
DN	Distinguished Name
O	Organization

Table 2-2 AD object naming

When working with Active Directory it is important to understand the object naming within the directory. We have listed the terms that are useful to understand the meaning of in Table 2-2. The objects in the AD are located with a hierarchical path, where the full path of the object is the distinguished name (DN) (Microsoft, Object Naming, 2014).

To illustrate this we can look at Figure 2-1. The CN *Andreassen Ruben Alexander* lies within the CN *Recipients* that lies within the OU *Exchange Administrative Group* that lies within the O *UIT*. When we write this out in reverse order, it makes up the DN. In most examples the O would be replaced with *DC=UIT, DC=NO*, but the organization is free to organize their own structure with any keyword suitable for the organization.

```
O=UIT
|->       OU=Exchange Administrative Group
|-->              CN=Recipients
|--->                     CN=Andreassen Ruben Alexander

DN: CN=Andreassen Ruben Alexander,CN=Recipients,OU=Exchange Administrative Group,O=UIT
```

Figure 2-1 Example of an object path in AD

2.6.1 Lightweight Directory Access Protocol (LDAP)

The Lightweight Directory Access Protocol (LDAP) is an application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network. The current version of LDAP is version 3 (LDAPv3), and the default Transmission Control Protocol (TCP) port is 389. RFC 4510 contains the LDAP technical specification (Zeilenga E. K., 2006).

When working with a secure connection, LDAPv3 has a Transport Layer Security (TLS) extension. The older LDAP version (LDAPv2) uses the URL scheme LDAPS on port 636. The usage of LDAPS was never standardized and LDAPS is deprecated along with LDAPv2 (Zeilenga K. , 2003).

A client starts an LDAP session by connecting to a server that supports LDAP. The client then sends an operation request to the server, and the server sends responses in return. The client can send many requests without waiting for the responses, and the server can return the responses in any order.

LDAP requests can include StartTLS, Bind and Search. The StartTLS request set up a secure connection. Bind authenticates the client and specify the LDAP version for the connection. Search searches the directory for entries and returns these entries if any matches the search criteria.

In theory, LDAP supports a persistent connection. A persistent connection means that a client can authenticate once, and then use the same connection until the connection times out or is manually closed. This eliminates the need to authenticate for each request. However, the LDAP implementation of the programming platform that we have chosen does not support a persistent LDAP connection.

2.7 Microsoft Exchange

Microsoft Exchange (MXS) is a server software that specializes on email, calendar and contacts (Microsoft, Exchange Server 2013, 2015). The email, calendar and contacts can easily be accessed through a desktop software like Outlook, online with Outlook Web App, or on any mobile phone. This makes the Exchange software very powerful and easy to use for end users. The Exchange software also contains email distribution groups and other resource objects such as meeting rooms.

2.7.1 Microsoft Exchange Web Services (EWS)

Exchange Web Services (EWS) provides the functionality to enable client applications to communicate with the Exchange server. EWS provides access to much of the same data that is made available through Outlook. SOAP provides the messaging framework for messages sent between the client application and the Exchange server. The SOAP messages are sent by HTTP (Microsoft, Exchange Web Services (EWS) in Exchange 2010, 2010).

In other words, the EWS is the API that we need to use when communicating with the university's Exchange server. The documentation of EWS is very large and detailed, and it has a big community of experts in forums. The programming platform we chose did not have a build in library to communicate with EWS, but we found an open source project on GitHub that we could use as a starting point. The library where not complete so we ended up adding the functionality that we needed and contributed the code back to the community through GitHub (Andreassen, 2014).

2.8 Syllabus

Scientia is the company that develops Syllabus. In this thesis, most of the information about Syllabus is unknown. According to Scientia's website:

"Syllabus Plus Enterprise Foundation is our core scheduling and timetabling solution. It provides a suite of core applications which help organizations to improve their efficiency by managing the planning and scheduling processes involved in the production of a timetable or schedule." (Scientia, Timetable Scheduling, 2014).

Of this, we can understand that Syllabus is a term of some software. What we do know on the technical side is that Syllabus runs on top of an Oracle database. The version of Syllabus that the university use does not have an API. This means that we need to retrieve the data directly from the database. It is therefore not important how Syllabus works, as long as we are able to understand the structure of the database.

This is of course only true when we focus on information retrieval. If we would be able to add or modify data in the database we would need to understand how Syllabus would do the exact same operation to avoid breaking the database.

2.8.1 Oracle

An Oracle database is a collection of data treated as a unit. The purpose of the database is to store and retrieve related information (Oracle, 2015).

Since the version of Syllabus that the university uses do not have an API that other developers can use, we can take advantage of the database. The database can be accessed directly by anyone and any software that has access and supports the right protocols. In addition, our contact at the IT department provided us with documentation of the database (Scientia, All SDB Schema Tables, 2003). One of the relevant tables in this thesis are documented as shown in Table 2-3.

As Table 2-3 illustrates, there is not much description available for what the table and columns contain. What we do know is that the tables we use in this thesis are views. A view is a SQL statement that is stored in the database to make a logical table.

SDVC_COURSE			
No description available.			
Column	**Type**	**Description**	**From Table**
ID	char(32)	No description available.	SDO_COURSE
NAME	varchar(255)	No description available.	
HOST_KEY	varchar(255)	No description available.	
DESCRIPTION	varchar(255)	No description available.	
DEPARTMENT	char(32)	No description available. Foreign key to SDO_DEPARTMENT.	
LINK_SIZE	integer	No description available.	SDP_COUSE
PLANNED_SIZE	integer	No description available.	
DICT	char(32)	No description available.	SDP_OBJ_WITH_DICT
USER_TEXT_1	varchar(long)	No description available.	SDP_USER_TEXT
USER_TEXT_2	varchar(256)	No description available.	
USER_TEXT_3	varchar(256)	No description available.	
USER_TEXT_4	varchar(256)	No description available.	
USER_TEXT_5	varchar(256)	No description available.	

Table 2-3 Example of table documentation of Syllabus (Scientia, All SDB Schema Tables, 2003)

2.8.2 Syllabus database setup at the university

Because of security reasons the university copy information from the production database to another identical database that 3rd party applications can use. This is because they don't want any other software than Syllabus to work on the production database. The copy process takes place every night, so the data is quite up to date. As we know of, our project and the in house web solution for viewing timetables developed at the university (timeplan.uit.no) uses this database for accessing Syllabus information.

2.9 Fronter

Fronter is a learning platform that includes a number of tools to collaborate and learn through the Internet (Fronter, Fronter is a virtual learning environment, 2015). The tool that we are most interested in is the calendar. All users in Fronter have their own personal calendar. The calendar is available in their personal toolbar, and can be shared with other users. We will use the OpenAPI to access calendar events. Although Fronter have an API available, we will see that the API is not suitable for our needs because of the authentication method used to access the API.

2.9.1 OpenApi

The OpenApi v1 is REST-based. The API uses the HTTP protocol and simple requests and responses. No complex SOAP calls. Because of this, it should be possible to use the API from most programming languages, even JavaScript.

2.9.1.1 Request format

The structure of the calls is best explained with examples, as shown in Figure 2-2.

```
api/v1.php/rooms
api/v1.php/rooms/12345
api/v1.php/rooms/12345/folders
api/v1.php/rooms/12345/folders/1234
api/v1.php/rooms/12345/folders/1234/elements
api/v1.php/rooms/12345/folders/1234/elements/1234
```

Figure 2-2 Fronter rooms URL structure (Fronter, OpenAPIv1, 2014)

When the last parameter is numeric, we will get a detailed response. It contains only information on one element. When the last parameter is a keyword, it is a collection, and the response is a list of elements.

Because the API is REST-based, the methods behave differently depending on how we call them. We can do GET, POST, PUT, DELETE or OPTIONS request.

GET requests will get info on the element requested. PUT requests will update information on the element requested. POST requests will create new elements beneath the requested element. DELETE requests will delete the element requested, and OPTION is a request to get information about a method. The latter will return what calls the method supports (Fielding & Taylor, 2002).

2.9.1.2 Response format

The OpenApi supports several response formats, and the output is always UTF-8 encoded. To change output format, we add the parameter *alphanum_format* to the request and specify the output format as shown in Figure 2-3. The valid output formats is listed in Table 2-4.

```
api/v1.php/rooms?alphanum_format=json
```

Figure 2-3 Fronter room, how to change format of the response (Fronter, OpenAPIv1, 2014)

Format	**Description**
XML	The XML format used is WDDX. WDDX is an open standard for data exchange between systems, using XML.
JSON	JavaScript Object Notation. Refer to RFC 4627 for more details.
dump	PHP's print_r output only used for debugging.
serialize	PHP's serialize function. Use it if you need it.
HTML	Only for debugging. More readable by humans, and partially hyperlinked.
raw	Raw iCalendar format.

Table 2-4 Fronter response types (Fronter, OpenAPIv1, 2014)

2.9.1.3 Response limits

Each response is limited to maximum 100 records. We can use the parameters *int_limitstart* and *int_limitcount* to adjust the response as shown in Figure 2-4. If the call returns 100 records we must use several request to make sure we retrieve all the records.

```
//gets the first 100 records
api/v1.php/rooms?int_limitstart=0&int_limitcount=100

//gets the next 50 starting at 100
api/v1.php/rooms?int_limitstart=100&int_limitcount=50
```

Figure 2-4 Fronter response limits example (Fronter, OpenAPIv1, 2014)

Note! During the testing of the Web Service we discovered that the documentation on the response limits are not accurate. We could not manage to limit the results, and we retrieved over 100 elements in one request. The Fronter documentation is therefore wrong or outdated on this matter.

2.9.2 OAuth

Fronter uses OAuth to authenticate access to the OpenApi functions. OAuth is an open protocol to allow secure API authorization in a simple and standard method from desktop and web applications. Fronter uses OAuth 1.0[1] (Hammer-Lahav, 2010). All 3rd party applications must implement OAuth 1.0 to get access to information inside Fronter through the OpenApi. Fronter supports the HMAC-SHA1 OAuth signature method. HMAC is a mechanism for message authentication using cryptographic hash functions (Krawczyk, Bellare, & Canetti, 1997).

2.9.2.1 OAuth in Fronter

When a 3rd party application want's to call the OpenApi in Fronter, they first get a request-token from Fronter. Then the user gets a GUI from Fronter asking him to log in[2]. If the user is already logged in, the user will be asked if he want to grant access to the 3rd party application. When granted access, the user is redirected back to the 3rd party application. The request-token is now enabled. The 3rd party application then exchanges the request-token for an access-token. Now the 3rd party application can do OpenApi calls with the access-token until the token expires, the default is 2 hours. This workflow is illustrated in Figure 2-5. If the 3rd party application wants to continue accessing the OpenApi after the token has expired, it needs to do the authentication possess all over again.

To get access to OAuth and OpenApi, 3rd party applications must get a consumer-key and consumer-secret from Fronter. The consumer-key and consumer-secret will be tied to one Fronter building. A 3rd party can get multiple keys if required. The consumer-key should have a name that identifies the 3rd party in some way. We have managed to get a pair of consumer-key and –secret to use in this thesis.

[1] A newer version, OAuth 2.0 obsoletes OAuth 1.0 (Hardt, 2012).
[2] When the 3rd party app gets the request-token, and the user must grant access, this GUI will not redirect the user to the login page. If the user is not logged in, they must open their normal Fronter login page and then login. After that, return to the OAuth access page and refresh. Fronter is doing this for security reasons. Fronter wants to protect the users from any phishing attempts.

When we have the consumer-key and –secret, we can authenticate our Web Service using the OAuth endpoints as shown in Table 2-5.

Description	**Endpoint**
Obtain a request token	https://fronter.com/customername/api/oauth.php/requesttoken
Authorize the request token	https://fronter.com/customername/api/oauth.php/authorize
Upgrade to an access token	https://fronter.com/customername/api/oauth.php/accesstoken

Table 2-5 Fronter OAuth endpoints (Fronter, Oauth In Fronter, 2014)

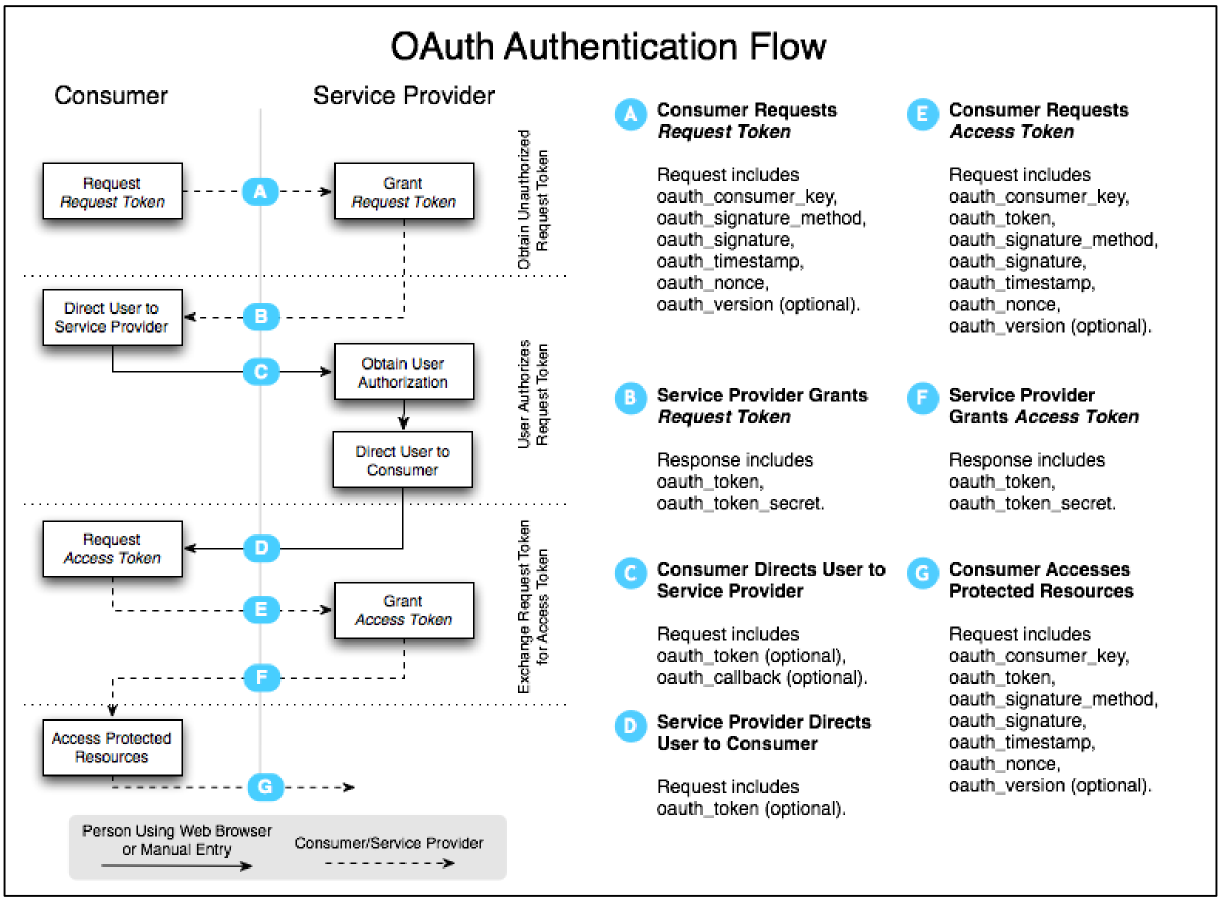

OAuth Authentication Flow
Consumer
Service Provider
Request Request Token
A
Grant Request Token
Obtain Unauthorized Request Token
Direct User to Service Provider
B
C
Obtain User Authorization
Direct User to Consumer
User Authorizes Request Token
Request Access Token
D
E
Grant Access Token
F
Exchange Request Token for Access Token
Access Protected Resources
G
Person Using Web Browser or Manual Entry
Consumer/Service Provider
A Consumer Requests Request Token
Request includes oauth_consumer_key, oauth_signature_method, oauth_signature, oauth_timestamp, oauth_nonce, oauth_version (optional).
B Service Provider Grants Request Token
Response includes oauth_token, oauth_token_secret.
C Consumer Directs User to Service Provider
Request includes oauth_token (optional), oauth_callback (optional).
D Service Provider Directs User to Consumer
Request includes oauth_token (optional).
E Consumer Requests Access Token
Request includes oauth_consumer_key, oauth_token, oauth_signature_method, oauth_signature, oauth_timestamp, oauth_nonce, oauth_version (optional).
F Service Provider Grants Access Token
Response includes oauth_token, oauth_token_secret.
G Consumer Accesses Protected Resources
Request includes oauth_consumer_key, oauth_token, oauth_signature_method, oauth_signature, oauth_timestamp, oauth_nonce, oauth_version (optional).

Figure 2-5 OAuth authentication flow (Fronter, Oauth In Fronter, 2014)

2.10 Summary

A Web Service needs to have a Service Oriented Architecture to make the Web Service loosely coupled from any external system. Many protocols are used, but HTTP stands out as the most important. In addition to that, there are several methods to choose from when passing data to or from the Web Service such as SOAP and REST.

Each of the external systems that we are interacting with uses different types of interfaces and communication protocols, which means that our Web Service needs to be highly flexible and extensible.

Taking all this into account, we will design our Web Service accordingly in the next chapter.

3 Design

In this chapter we will describe a real scenario that we will use in the design and implementation of our Web Service. Then we will look at an overview of the design, the system architecture and all of the components that make up the Web Service.

3.1 Case

Large organizations often use software from many different vendors. Sometimes, software from different vendors contain similar types of information that could naturally be combined into one new system. In this thesis we will concentrate on systems that contains calendar information and how we can combine these calendars on a generic calendar platform.

One of the main issues within the university's organization is that it is hard for one person to plan a meeting when the participants and rooms have calendars in totally different systems. If a participant is a student he has a personal calendar in Exchange but also a timetable in Syllabus. If the room is a meeting room it has a calendar in Exchange, but if the room is a teaching room or auditorium it has a calendar in Syllabus. The person planning the meeting, maybe a teacher, has a calendar in Exchange and the teaching plan in Syllabus, and can also have a private calendar in Fronter.

As we can imagine, it can be challenging to accurate plan a meeting without the organizer having to have an extensive dialog between every participant. We believe that if we make a calendar system that integrates all the necessary calendars into one new calendar system, an organizer can much more accurately plan a meeting between several participants on the first attempt or at least at an earlier stage than would be possible without such a system.

3.2 Overview

The Web Service has to be able to communicate with several different types of systems. To make matters worse, none of the systems we are implementing use the same way of exchanging data. This means that the Web Service must be extremely flexible. To handle this, we create many different components that can be considered as interfaces and connectors between the Web Service and the 3rd party systems that is not a part of the Web Service.

The workflow of the Web Service has three steps. First, we must retrieve information from the external systems. Then, we need to combine the information so it looks like the information would have come from one single system. The last step is to present the retrieved combined information to the end user or other end user application.

The Web Service is stateless. We only store temporarily files, cookies and access tokens. This means that the Web Service need to authenticate the user using the service on every request against the external systems. The reason for this is that some of the external systems does not support persistent connections. Without a persistent connection, the connection is lost at the end of every sequence of requests. This makes it impossible for the Web Service to handle following requests without knowing the username and password of the user. If the Web Service would know the credentials, it means that it could be compromised to give up that information. To make the security on the Web Service as good as possible, we leave it up to the 3rd party systems that uses the Web Service to handle the user's credentials in a secure fashion and send them along to the Web Service on every request. The Web Service uses HTTPS with a valid certificate to ensure that the traffic is encrypted.

3.3 System architecture

Figure 3-1 gives a graphical overview of the whole system. How the dataflow is between our Web Service and the external systems that we retrieve data from, and how the end user can interact with the Web Service through a web browser or other 3rd party system.

The systems we retrieve data from is on the bottom of the figure. These are the external systems and are mainly located at the university, but they could be located anywhere. Fronter is located outside the university.

In the middle of the figure is the Web Service. This is where all the heavy work is done. The Web Service has connectors that communicate with the external systems, middleware that combine the data being retrieved, and interfaces that deliver the finished product to the end user. We also have temporary storage.

At the top of the figure we have the different 3rd party systems that may use the interfaces provided by the Web Service.

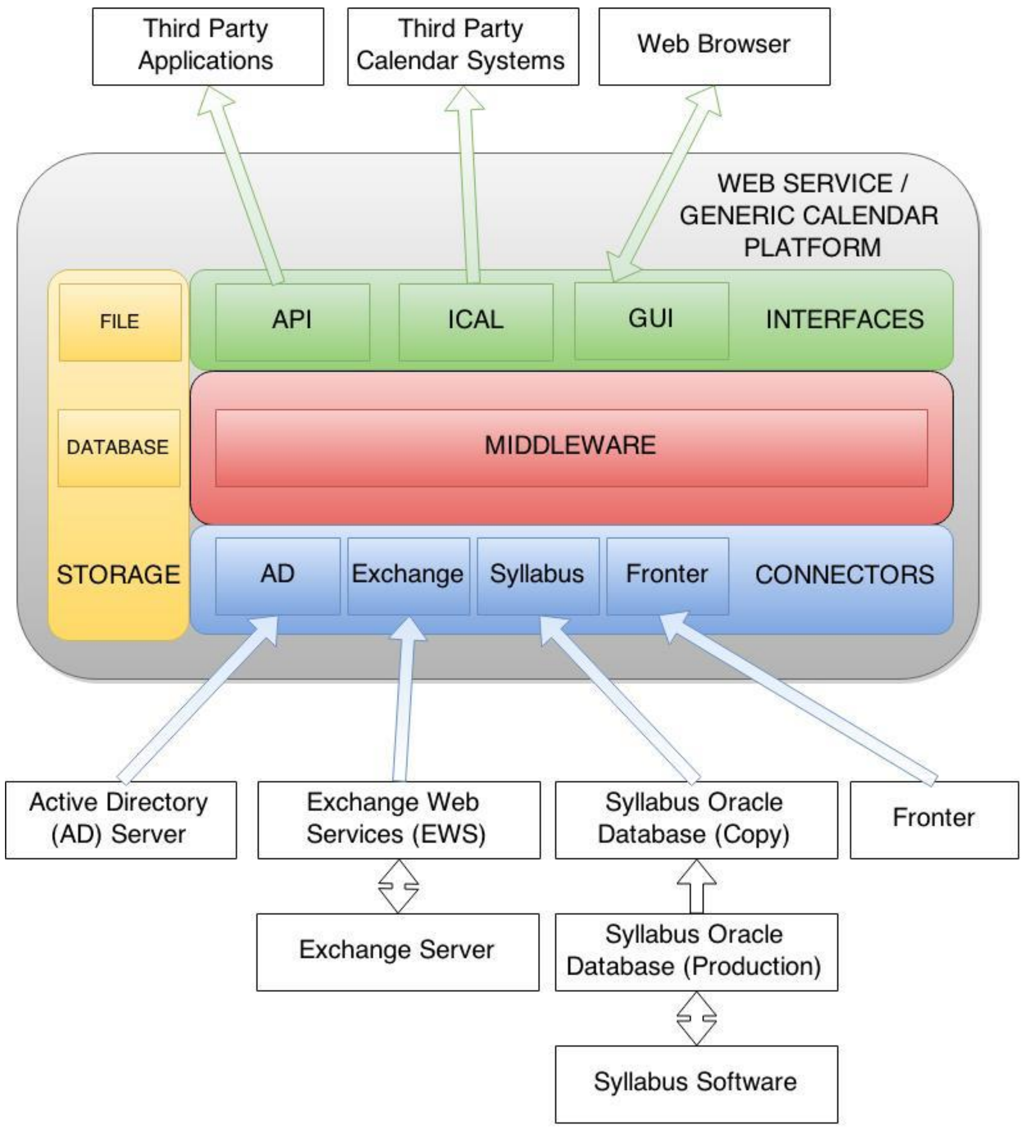

Third Party Applications
Third Party Calendar Systems
Web Browser
WEB SERVICE / GENERIC CALENDAR PLATFORM
FILE
API
ICAL
GUI
INTERFACES
DATABASE
MIDDLEWARE
STORAGE
AD
Exchange
Syllabus
Fronter
CONNECTORS
Active Directory (AD) Server
Exchange Web Services (EWS)
Syllabus Oracle Database (Copy)
Fronter
Exchange Server
Syllabus Oracle Database (Production)
Syllabus Software

Figure 3-1 System overview

3.4 Components

To make maintenance easy and flexible, the Web Service is built up of many components. We have one component for each of the external system that the Web Service retrieves information from. These are the connectors shown in Figure 3-2. The middleware itself is where the data is being processed and combined if necessary. The components on the top are the interfaces that makes the retrieved data available to the end user or other end user system. All of the components can use the temporary storage as needed shown to the left in the figure.

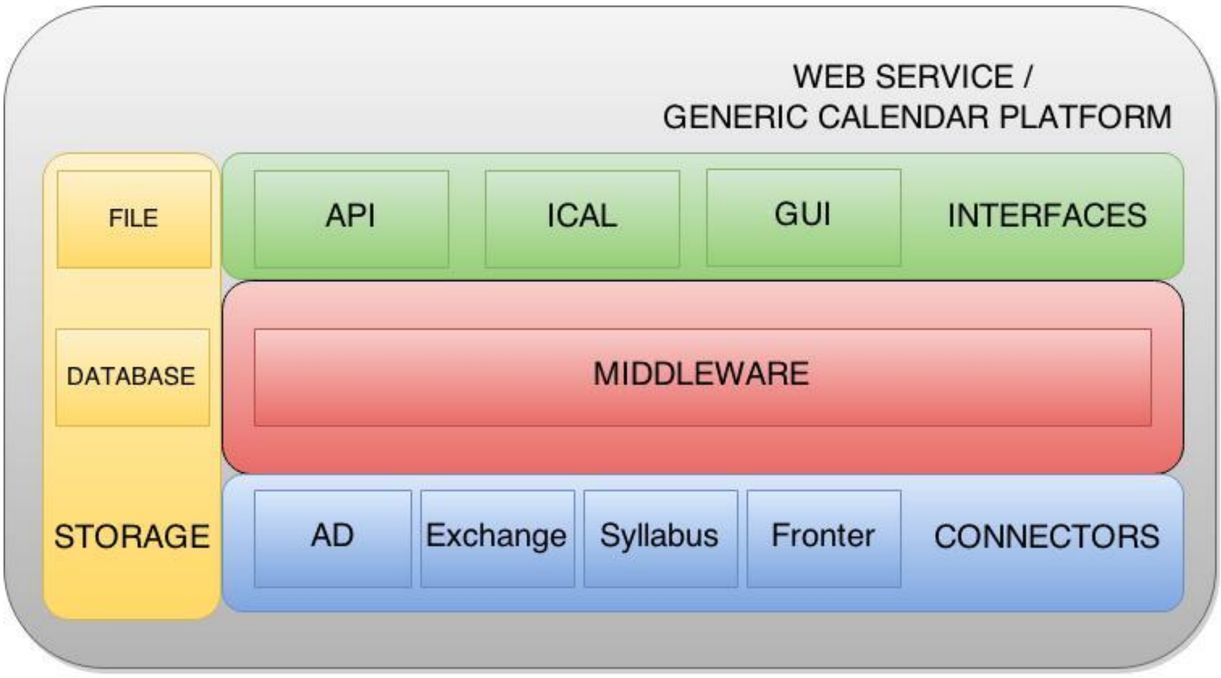


Figure 3-2 System architecture

3.4.1 Connectors

The connectors communicates and retrieves information from external systems that are not part of the Web Service. We will describe these components and provide information on how they communicate with the external systems and which information they retrieve from the external systems.

3.4.1.1 Active Directory Connector (ADC)

The ADC retrieves information from the Microsoft Active Directory (AD) at the university. All communication is done with plain LDAP since the AD does not support encrypted LDAP. Although the AD does not support encrypted LDAP at the time of the development process, the Web Service can easily be configured to use encrypted LDAP.

There is not only information retrieval that is done against the AD, we also use the AD for authentication. When we try to bind a username and password against the AD, it can tell us whether the credentials are valid or not. All of the external systems we implement, except Syllabus, authenticates against the university's AD. This is very important as it means that the user only need to use one username and password to access our Web Service, or each external system individually. The alternative would be to ask the user for several different credentials when using the Web Service so that the Web Service would be able to connect to all of the external systems. This would be very inconvenient for the end user.

Information about the logged in user

The AD contains much information about any given user. In fact over 70 attributes. We are not interested in all of them, but the ones listed in Table 3-1 stands out as useful.

LDAP attribute	Description
displayName	The display name for an object. This is usually the combination of the user's first name, middle initial, and last name.
givenName	Contains the given name (first name) of the user.
sn	This attribute contains the family or last name for a user.
mail	The list of email addresses for a contact.
samAccountName	This attribute is used to store the SAM account names that correspond to the DNS host names in ms-DS-Additional-Dns-Host-Name.
memberOf	The distinguished name of the groups to which this object belongs.

Table 3-1 LDAP attributes used by the AD connector (Microsoft, All Attributes, 2015)

One important design feature we came up with was the idea that when a user wants to see his or hers calendar in our Web Service, the timetable of the courses and student sets that the user is taking would show up in the calendar automatically. The university has a system that contains this information, code name BAS[3]. This system knows everything about a student. For example which courses and student sets the user has taken, are currently taking and are going to take. Unfortunately we do not get access to this system because of security reasons, and therefore we had to find other ways to retrieve that information.

One of the things we noticed was that the *memberOf* attribute that contains information about all of the groups a user is a member of, had groups that looked very similar to course codes. By traversing the groups in this attribute it is possible to find out which courses a user is taking at the current moment. It is not possible to find out which courses a user has already taken, or is going to take. The reason for this is that the groups a user is a member of change each semester.

A user can also be a member of groups that represents student sets. However, we have decided to not implement the linking between the student set groups in AD and the student sets in Syllabus. The main reason for this is the lack of documentation from our contacts at the university on how to parse the student set groups to be able to match them with the correct student sets in Syllabus. It seems that this parsing cannot be done, and the only way is to change the AD group to contain more information, or retrieve the information directly from BAS which we do not have access to.

Searching for other users

The Web Service have the ability to merge several calendars, or to view other users' calendars. To be able to merge other students or staffs calendars we first need to find them. The AD is the best way to search for users since this is one of the main purpose of the directory.

[3] The university has several systems with user information. Some of the systems we know about but don't have access to, are the following systems: FS, PAGA, BAS and System-X (Orakelet, Støttesystem for oppretting av IT-brukerkonto, 2015).

We first thought that the best way to search for a user was to use the name attributes described in Table 3-1. After a while we discovered that it was difficult to search for some users, especially if they had middle names. We were getting zero results when we knew that we should get at least one or more results. Then we discovered the *anr* attribute that is an efficient search algorithm that allows objects to be found without complex search filters (Microsoft, Ambiguous Name Resolution for LDAP in Windows 2000, 2007). This meant that we discarded our attempt to make a search filter based on the name attributes and used the *anr* attribute instead.

When a user is found by his or hers name, we retrieve the same information about that user as we do for the logged in user as described Table 3-1.

Room names, e-mail addresses and searching

In this thesis we want to retrieve calendar information from meeting rooms, classrooms and auditoriums. All of the meeting rooms are in Exchange, and the other rooms are in Syllabus. Although all of the meeting rooms are in Exchange, they are also represented as resources in AD since there is a close binding between Microsoft Active Directory and Microsoft Exchange (Microsoft, Access to Active Directory, 2013).

We must retrieve the calendar of a room from Exchange. But if we have the identifier or the name of a room and only want information about that room we have two alternatives. Our first instinct was to retrieve the room name from the AD instead of Exchange. The reason for this is that we believe that the overhead of SOAP messages would be significant to the small amount of information that we need to retrieve. This was later documented in Appendix D, proving that our instinct was right. We therefore decided to retrieve room information from AD trough LDAP, which has little overhead. Part of the room information is the name of the room and email address. The email address can be used as an identifier to retrieve the room's calendar information from the Exchange server through the EWS interface.

Since the AD is specially designed to handle searches and performing them as fast as possible, it makes more sense to look up the rooms in AD instead of Exchange. When searching for a room we search on the *displayName* attribute. We did not experience any difference in the search results when using the *anr* attribute as explained earlier. The reason for this is that only the *displayName* attribute is relevant in a room object in AD. The other attributes that are used when doing an ANR search contains information that would never be matched. We know this because we have inspected some room resources in AD and compared the attributes of the rooms to the attributes used in an ANR search. Examples of the attributes used in a calendar search can be found in the Implementation chapter.

3.4.1.2 Exchange Connector (XC)

The Exchange Connector uses SOAP requests to communicate with the university's Microsoft Exchange Server (MXS) through the Exchange Web Services (EWS) interface.

When we communicate with the Exchange through EWS we need to send three requests to the EWS interface as shown in Figure 3-3. The reason for this is that the detailed description about a calendar event is not returned by the first request. The Exchange server has many types of identifiers. The events that we receive has a unique identifier in a format called *HexEntryId*. But to get the detailed description of the events we need to have an identifier called *EwsId*. We therefore need to convert all the identifiers of the events from *HexEntryId* to *EwsId* in a second request, and then get the detailed description of all the events in a third request using the new identifiers. If we only wanted the date, time, subject and location we could manage with only one request. We have not implemented a future to turn this functionality on and off as needed, but it would be a good optimization for further work to implement such a feature. The details of the requests are listed in Table 3-2.

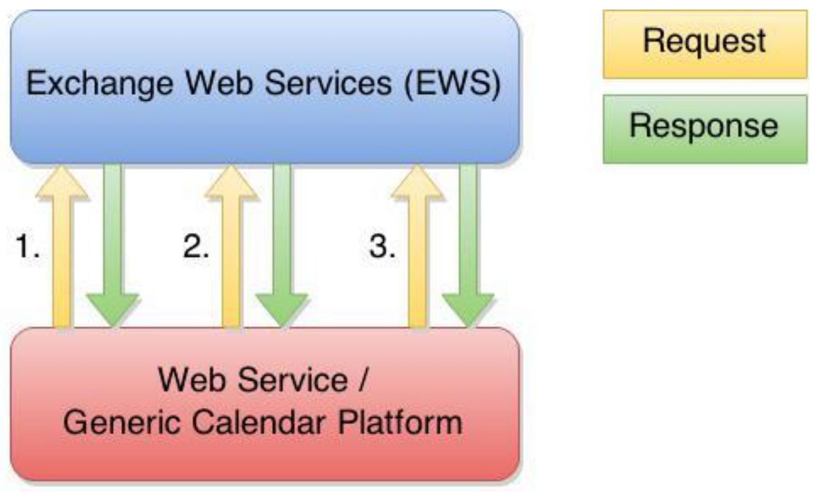

Exchange Web Services (EWS)
Request
Response
1.
2.
3.
Web Service / Generic Calendar Platform

Figure 3-3 EWS SOAP Request/Response

EWS SOAP Requests overview		
Request	**Operation**	**Description**
1.	GetUserAvailability	Provides information about the availability of a set of users, rooms and resources within a specified time period.
2.	ConvertId	Converts item and folder identifiers between different formats. In our case we convert between *HexEntryId*[4] and *EwsId*[5].
3.	GetItem (calendar item)	Returns all details about calendar events using the identifiers from the second request.

Table 3-2 EWS SOAP Request explanation

Note! There is a limit to query 100 mailboxes in each request when communicating with the EWS using the *GetUserAvailability* operation (Microsoft, GetUserAvailability Operation, 2011). This means that for each additional 100 mailboxes we want calendar events from we need to double the number of requests being made.

[4] The hexadecimal representation of the PR_ENTRYID property. The PR_ENTRYID identifies an object such as a CalendarEvent object in our case.

[5] The EWS identifier format is used in Exchange 2007 SP1 and later versions of Exchange Server.

Calendar information

There are to main types of calendar information we can retrieve from the Exchange server. These types are detailed calendar information and busy-free calendar information.

Detailed calendar information can be obtained when the logged in user has extended access to a specific calendar. This is the case for the personal calendar of the user, and any calendar events that the user has created on a meeting room, or any calendars that are shared with the user. Detailed calendar information includes the date, time, length, subject, description and location of all events.

Since most users don't share their calendar with everyone else, the logged in user can only view what is called busy-free calendar information about other users and meeting rooms. This is basically only information about whether a calendar has events or not, meaning that we get the date and time of all events. We don't get to see the subject, description or location of the events without the right kind of permissions.

Room List (RL) information and searching

Both Syllabus and Exchange has the possibility to group several rooms into room lists (RL). This makes sense, as one could make a RL of all the rooms in a given building or a given floor, and it would be easy to find all the rooms at that building or floor based on the RL. RL information does not exist in the AD as resources in the same way as rooms, so the RL's needs to be retrieved from the Exchange through the EWS interface.

RL information includes the name and the e-mail address of the RL, and names and e-mail addresses of all the rooms included in the RL. However, there is a limitation in the EWS interface that means that we first need to retrieve all the RL's, and then make one request for each RL to retrieve all the rooms in that specific RL. This means that if we want to retrieve ten RL with all of the associated rooms, we need to make eleven requests. It would have been convenient if we could do this in one request because of the overhead in SOAP requests and the fact that less requests is better in this setting, but that is not possible. We design this function so that it is possible to retrieve all the RL's with or without the rooms to save requests when the actual rooms are not relevant.

The EWS interface does not have any search operation on the RL's. We therefore need to retrieve all the RL's and do the searching on that result set on the Web Service. We would recommend to redesign this in the future if the EWS interface would implement support to search for among RL's. Or even better, if the AD would contain RL information in the future, the searching could be done through LDAP.

3.4.1.3 Syllabus Connector (SC)

The Syllabus Connector (SC) communicates directly with a copy of the Syllabus database (SDB). The version of Syllabus that the university is using don't have any interfaces for 3rd party applications. The database is running on an Oracle database server. The SC uses SQL with Oracle syntax to retrieve data from the database.

The user does not need to authenticate against Syllabus to retrieve information as with the other systems. We use a separate database user for this. The database user has read only rights for an extra layer of security against manipulating the queries. We are very aware of SQL injection and use only prepared statements and proper escaping of any input data generated by the user[6].

[6] This does not mean that one should consider stored data as safe. Always escape data properly for the system that the data is being inserted into.

Searching
The SDB contains information about courses, student sets, rooms, room lists, and staff. All this information is searchable by writing Oracle SQL statements. The search term is applied to different tables and columns based on what the user is searching for. If the user is searching for a course, the search term is applied to the course name and course code.

Sometimes, instead of having a lot of courses making up a semester, student sets (SS) are created. A SS is a predefined study program that contains all of the lectures for that given study program. We have implemented searching on the name and code of SS's.

As mentioned earlier, both Exchange and Syllabus contains room calendars at the university. Rooms that are used for teaching, colloquium and other activities except meetings are located in the SDB. We have implemented searching on the room code and room name.

Like on the Exchange server, Syllabus also has the ability to group rooms into room lists (RL). We have implemented searching on RL code and RL name.

Detailed calendar information
A course, student set or room in the SDB can have one or more activities. Each activity contains at least one event, but usual a series of events. By gathering all activities we can calculate the events date and time, and get the subject and description. Each activity can also be linked to one or more rooms. This makes it possible to find the location of events for courses and SS's.

Room List (RL) information
Like on the Exchange, the rooms in SDB can also grouped into several RL's. But unlike the EWS interface, we have the ability to make a query that retrieves all of the room lists and all of the rooms in one result set by joining several tables.

However, we have not done this and make one request to retrieve all the RL's and one additional request for each RL to retrieve all the rooms in that list. The reason that we do this is that there is almost no overhead when making a SQL request to the Oracle database server. We also have a persistent connection that means that we only need to connect and authenticate once, and can then make as many requests as needed. If we would to do this operation in one request we need to do a little bit more processing on the Web Service. To be absolutely sure that we have made a good choice here, one could compare the execution time on both methods. We have not done this since we only have made a proof of concept, and everything should be double checked if this project would go into production.

RL information includes the code and the name of the RL, and codes and names of all the rooms included in the RL if needed. We have also made it possible to retrieve all the RL's without including the rooms if that information is irrelevant for the request.

Staff information
In the same way that the AD contains which courses a student is taking, the SDB contains information on which courses a staff (teacher etc.) is teaching. Unlike the information in AD that is real time, SDB contains information on previous courses, the current courses, and the courses that a staff is planned to teach in the future if such a plan exist.

3.4.1.4 Fronter Connector (FC)
The Fronter Connector (FC) communicates with Fronter through the Fronter OpenApi, which is REST-based. OAuth is used to gain access to the OpenApi functions.

Getting access to the API
Since the OpenApi requires OAuth authentication, there are a few steps that needs to be done to gain access to the API. The first thing we need is a consumer key. This is provided by Fronter and is unique to the university's organization. The consumer key can be used to gain a request token. The request token can be used to gain an access token. To achieve this we send a request to Fronter with the request token. Then the user has to log in to Fronter and by doing so granting access to an access token. The Web Service can then use this access token for a limited period of time, the default is 2 hours, to send requests to the OpenApi.

This would not be very user friendly if other applications implementing our REST API would need to redirect the user to the Fronter login page every two hours. To avoid this, we have made a scheme simulating a browser to automate the whole process.

Private calendar information
This is the users own private calendar in Fronter. We can retrieve full calendar information from this calendar. This includes the date, time, subject, description and location of all events.

Room calendar information
The room calendar in Fronter can be considered a course calendar since it is not a physical room, but a virtual room containing information about a given course, study group etc. When a user is added to a room in Fronter, meaning a virtual group and not a room according to our definition, the user can retrieve full calendar information from this calendar. This includes the date, time, subject, description and location of all events.

3.4.1.5 Adding new connectors
Since the Web Service is a generic calendar platform, it is easy to expand the platform with more connectors. The connector's main purpose is to implement the interface of an external calendar system.

It is best if the user uses the same credentials[7] to access the external system that is going to be implemented as on the other external systems that are already implemented. If this is true, the connector can retrieve the user's credentials from a static object on the Web Service. If not, the connector must retrieve the credentials itself.

When the credential issue is solved, the connector must implement the authentication method of the external system. This can be HTTP Basic Authentication, OAuth or any other authentication type that the external system uses to protect its interface.

The connector must be able to retrieve calendar information from the interface of the external system based on an identifier and a date interval. The connector can implement searching to retrieve identifiers and other information from the external system.

The connector is usually a separate class on the platform. The only modification to the main code that is needed, is to register the connector's calendar functionality in the main calendar script. If the connector offers searching possibilities it must register these in the main search script.

[7] We are not encouraging users to use the same credentials on several services. A user should always use different credentials on every service he is using to minimize the damage if one service is compromised.

3.4.2 Middleware

The data retrieved from the connectors will be in different format and have different semantics, but the content itself will be more or less the same. We therefore need to make a common representation of the data, and parse the data from the connectors to our own internal format.

When we have all the data in one format, the middleware can hand it over to the interfaces ready for delivery to the end user. Since we have programming interfaces, we need to deliver the data in a format that other developers can understand. We therefore use the RFC 5545 specification to make an iCalendar format (Desruisseaux, 2009). The data is outputted as raw iCalendar from the ICAL interface, and wrapped in JSON from the API.

3.4.3 Interfaces

The interfaces make the information gathered by the connectors available to the end user. There are several different interfaces, making the information available in a variety of ways to the end user or a 3rd party application. In this section, we will describe these interfaces and provide information on how they make the information available to the end user or a 3rd party application.

3.4.3.1 Graphical User Interface (GUI)

The GUI makes the information retrieved by the connectors directly available to the end user through a web browser. This is a web page that any user with a user account at the university can access and explore the functionality of the Web Service.

Searching for a user’s calendar

This function makes it possible to search for any user (student or staff) that exist in the university’s AD. When the user you are searching for is found, it is possible to click on that user to view the user’s calendar. This is a combination of several different calendars from several of the external systems.

- **Exchange:** The user’s Busy-Free calendar.
- **Syllabus:**
    - If the user is a student, all the courses that the student is currently taking is shown. The information about which courses a student is taking comes from parsing the groups that the student is a member of in AD.[8]
    - If the user is a staff (teacher etc.), all the courses the staff is teaching are shown.
- **Fronter:** We cannot retrieve any information about other users than the authenticated user.

The calendar information that we are able to retrieve on the logged in user is slightly different and is explained in My Calendar (MC).

Other search functions

The ability to search for things is important. We want the user to be able to search for courses, student sets, rooms and room lists.

To search for a course, the user needs to input three or more letters to make a list of current matches appear. When the course that the user are searching for is found, it is possible to click on that course to view the calendar events of the course. The events of a course is typically when and where the course has lectures and colloquiums. Everybody has full access to view the detailed information about these events. The same procedure applies to student sets since student sets and courses are very similar in the SDB.

[8] The same could be done with student sets. See Information about the logged in user in chapter 3.4.1.1 about why we did not implement this.

When searching for a room by room code or name, the search function will search in both AD and SDB. The results are then combined into one list and displayed to the user. When the user clicks on a room, the room's calendar is displayed. We do not display which system that the room calendar comes from in the GUI. Our initial thought was that the user does not need to know this information, but we have later changed our mind. It would not hurt to display which external system the calendar comes from. The user could benefit from this information if the calendar would need to be edited.

It is also possible to search for any room list (RL) by code or name. The search function also combine similar RL's. When the RL that the user want is found, the user can click on that RL to display all the rooms in that RL. The list of all the rooms in the RL is also combined from both MXS and SDB. When clicking on a room the room's calendar is shown.

Display all rooms

This function makes it possible to browse through a list of all the rooms. It can be handy if the user don't know the name of the room that he is looking for. The list is a combination of all the rooms from AD and SDB. It is possible to click on a room and view the room's calendar. The details of the events will depend on which external system the room exist in. If the room exist in AD the calendar is stored in MXS and the user can only view the Busy-Free information about the events, unless the user have access to view detailed calendar information. If the room exist in SDB, the user can view detailed calendar information about all the events in the room's calendar.

When browsing through the list it is possible to find two entries of the same room. This is not a bug, but it means that the room exist in both external systems[9]. Since the Web Service don't combine rooms, the user will have to view both calendars to get the full overview. We could try to combine these rooms into one, but we decided not to do this for one main reason. If the rooms does not have the exact same name in both systems it would be difficult to programmatically figure out that two rooms are the same without any other common identifier. It would therefore be necessary to allow some slack when comparing room names, which could lead to rooms that are not exactly the same being combined into one calendar. It would be very hard to get the accuracy right without any manual processing.[10]

Display all Room Lists (RL)

This function makes it possible to browse through all the RL's, find out how many rooms are in each RL and browse through all the rooms in each RL. Since the RLs exist in two systems, MXS and SDB, the Web Service combines RL's with similar names. This means that a RL can contain rooms from both MXS and SDB. The criteria for combining two RL's is that the names of the RL's need to be more than 95% similar. This means that we could end up with two RL's that are not combined together even if a human can see that they are very similar, because the test shows 95% or less similarity. When clicking on a RL the functions displays all the rooms in that RL, and when clicking a room the user will get the room's calendar.

[9] One example we found was the room MV.110 F.220 Møterom.

[10] We later learned that the university was in a process to move all meeting rooms from Syllabus to Exchange. The same room should not exist in both systems at the same time, but we did not delete this section since the issue can be relevant in the future in a porting process or when adding more external systems.

My Calendar (MC)

The MC function displays the logged in users calendar. This is a combination of several different calendars from all the external systems, including:

- **Exchange:** The user's personal detailed calendar information.
- **Syllabus:**
  - If the user is a student, all the courses the student is currently taking is shown. The information about which courses a student is taking comes from parsing the groups that the student is a member of in AD. [11]
  - If the user is a staff (teacher etc.), all the courses that the staff is teaching are shown.
- **Fronter:**
  - The user's private calendar.
  - All the room calendars (virtual groups).

In addition to this, the MC function lets the user add other users, courses, rooms, student sets and the user can even upload iCalendar formatted files. To understand why this is a useful function, consider the following scenario:

A student is planning a meeting with his tutor. The student has a few options. He could phone the tutor and try to find a date and time that they could meet. They could also e-mail back and forth to find a date and time that they both are free. Or, the student could use the MC function of the Web Service to see the students own calendar, add the tutors calendar and add a few meeting rooms. The student can now easily see when they both are free, and a meeting room is available. In one combined calendar.

We did not implement a reverse calendar, meaning that the Web Service could suggest which dates and times that the student and tutor can meet. We consider such a function to be suitable for further development. The main reason for this is that our main goal is to retrieve all relevant calendar information and get an oversight of that information. A reverse calendar functionality would be a natural extension to this, but an extension that would have been more nice-to-have than relevant in this thesis.

3.4.3.2 ICAL Interface (ICAL)

The ICAL interface is used to access iCalendar files. This interface does not implement any authentication, therefore only public calendar information is available through this interface. This means that the same information displayed on the university's public web based timetable (http://timeplan.uit.no/) is available through this interface.

Any 3rd party calendar system can access these iCalendar files by using a URL. Depending on the URL that is being used, an iCalendar file is generated dynamically by the Web Service with the events of the courses that are specified in the URL.

If we would want to offer any user specific calendar information we would need to implement some sort of authentication scheme or make a unique hash or key that are provided in the URL. If we would implement any authentication it would limit the number of 3rd party applications that could import the calendar feed. For example, Outlook Web App (OWA) and Google Calendar only supports to add 3rd party calendars by URL without any authentication. If we would implement a unique hash that are provided in the URL that hash could be sniffed by anyone, making private information public

[11] The same could be done with student sets. See Information about the logged in user in chapter 3.4.1.1 about why we did not implement this.

to anyone that knows the hash. Another problem is that many of the external systems require authentication on every request. This would mean that we would need to cache all the calendar events on the Web Service, or store the user's credentials. Neither of the options seems acceptable. We therefore chose to only implement basic functionality for demonstration purposes.

3.4.3.3 Application Programming Interface (API)

To make the Web Service complete, we have made a REST API with JSON responses. By using this API other 3rd party applications can make combined calendars on any device that has access to the internet, for example a calendar application on an Android device.

REST API's are easy to use compared to SOAP, and JSON responses are much more modern than XML. The Web Service also offers authentication of user accounts against the university's AD. This means that student programmers that are not interested in calendar information can use the API to authenticate users instead of creating their own user registration and authentication. The advantage of this is that other student programmers don't need to learn how to use the LDAP protocol, if LDAP is even supported in the chosen programming language. This means that users at the university can use the same credentials that they use elsewhere at the university.

All resources available in the API is listed in Table 3-3. The full documentation is available in Appendix C. An API testing tool is also available along with the documentation in the Web Service GUI.

Resource	**Description**
GET api/v1/combine/calendar	Get calendar items from all specified calendars. The calendars can be from any user, room, course or student set.
POST api/v1/user/auth	Authenticates a user's credentials against Active Directory at the university.
GET api/v1/user/search	Search for a specific user.
GET api/v1/user/info/:id	Get information on a specific user.
GET api/v1/user/course/:semester/:id	Get all the courses a specific user is taking or teaching.
GET api/v1/user/calendar/:id	Get calendar items from a specific user's calendar.
GET api/v1/roomlist/all	Get a list of all room list's available.
GET api/v1/roomlist/search	Search for a specific room list.
GET api/v1/roomlist/room/:type/:id	Get a list of all rooms in a given room list.
GET api/v1/room/all	Get a list of all rooms available.
GET api/v1/room/search	Search for a specific room.
GET api/v1/room/calendar/:type/:id	Get calendar items from a specific room resource.
GET api/v1/course/search	Search for a specific course.
GET api/v1/course/calendar/:id	Get calendar items from a specific course.
GET api/v1/studentset/search	Search for a specific student set.
GET api/v1/studentset/calendar/:id	Get calendar items from a specific student set.

Table 3-3 Resources available in the Web Service API

3.4.4 Storage

The Web Service uses two types of storage. Both storage usages are only for temporarily data since the Web Service is stateless. The file storage is used to store sessions retrieved from the EWS, iCalendar files when the user uses the function to upload calendars to the system, and to store Fronter calendars when retrieved from Fronter.

The latter can seem a bit strange since the calendars that are retrieved from Fronter are in raw iCalendar format and can be parsed directly. The case is that we already had a library to parse iCalendar files, but the library only supported to pass on file names and not to pass iCalendar formatted strings directly. Instead of modifying the library we made a workaround to store the retrieved iCalendar string into a file, and then pass the file to the iCalendar parser library. An optimization could clearly be done to avoid this. We could use the file to cache the results from Fronter, but we did not do this either since we wanted real-time data on every request.

We also use a local MySQL database. We store the access tokens used to access the Fronter OpenApi in the database. Since the access tokens are valid for two hours it makes sense to store them instead of authenticating against Fronter more times than absolutely necessary. The complexity of authenticating against Fronter also matters in this choice since the authentication can take 1-2 seconds.

3.5 Summary

In this chapter we have seen how we have designed the Web Service and the choices we have taken. There is a lot to implement. We need to develop all the connectors to be able to retrieve data from each external system. Then we need to develop the middleware that needs to know the semantics of all the data, and create a new internal format to be able to deliver combined information to the interfaces. The interfaces will send the information to the end user or other end user software.

We have a lot to do, and in the next chapter we will get down to the dirty bit and actually do some programming.

4 Implementation

This chapter describes the implementation of each component in detail. The Web Service has a configuration file located in the root folder. If any of the external systems changes it is easy to edit the configuration file so that the Web Service don't fail. In the configuration file it is also possible to enable and disable each connector except the AD connector. The Web Service will continue to work if any of the connectors fail, except for the AD connector. The AD connector is used for authentication and must therefore always be available.

The Web Service is made up of many classes, but there is one class that acts as a controller class for the implementation of all the calendar systems. If someone would like to implement other external systems on the platform, it is necessary to implement a function in that class to populate the internal calendar object.

4.1 Connectors

First we are going to see how the Web Service implements the connectors. These components are the backbone of the Web Service, and stands for all of the information retrieval from the different external systems.

4.1.1 Active Directory Connector (ADC)

The ADC takes care of all communication between the Web Service and the university's AD. This includes authentication, searching for users and rooms and detailed information about these users and rooms. The AD itself does not contain any calendar information, but it contains objects that can be linked to Exchange through email addresses as identifiers.

We communicate with the AD through LDAP queries. The queries can be powerful, but confusing because of the syntax. We will not go into detail about how to write these queries, but it is explained in the TechNet article (Mueller, Active Directory: LDAP Syntax Filters, 2014) and we have some example figures in this chapter that we have commented for readability.

4.1.1.1 LDAP setup

To make the ADC able to communicate to the university's AD, LDAP needs to be set up correctly. The configuration unique to the university's set up can be viewed in Table 4-1.

Parameter	**Value**
Account suffix	@uit.no
Base DN	DC=ad,DC=uit,DC=no
Hostname	ad.uit.no

Table 4-1 LDAP setup

The default port of LDAP is 389. If the Web Service would need to communicate with a different AD, the values could be changed in the configuration file.

When writing this, the university's AD does not support encrypted communication, but we have implemented configuration parameters to turn encrypted communication on if the university would support this in the future. There are two types of cryptographic protocols that could be used when working with LDAP, the difference is described in Table 4-2.

We did not implement LDAPv2. There are several reasons not to implement this encryption method. RFC 3494 recommends that the LDAPv2 specification is moved to Historic status and gives two examples of the reason to do this (Zeilenga K. , 2003). The short draft is that some implementations of LDAPv2 does not comply with the specification and uses different syntaxes and semantics to those who follow the specification. Because of this, we decided to only implement LDAPv3 for security which creates an encrypted channel instead of an encrypted connection. LDAPv3 also comply with the specifications and is the recommended way to go.

	LDAPv2 (deprecated)	**LDAPv3**
Port	636	389
Encryption	Secure Sockets Layer (SSL)	Transport Level Security (TLS)
Hostname	ldaps://ad.example.com[12]	ad.example.com
PHP documentation	(The PHP Group, LDAP Functions (ldap_connect), 2014)	(The PHP Group, LDAP Functions (ldap_start_tls), 2014)
RFC Reference	RFC 2559 (Boeyen, Howes, & Richard, 1999)	RFC 3377 (Hodges & Morgan, 2002)

Table 4-2 Comparison of encrypted LDAPv2 and LDAPv3

4.1.1.2 Users

A user object in AD has many attributes. We use only a handful of these. The attributes we use in the Web Service is described in Table 4-3.

There are three types of name attributes. When the Web Service determines a user's full name, the preferred method is combining *givenName* and *sn* to get the users full name. The reason for this is that the common practice in the university's AD is that the last name comes first in the *displayName* attribute. Since we want to display the users full name as *firstName-lastName* instead of *lastName-firstName* we combine the *givenName* and *sn* attributes. However, if for some reason *givenName* or *sn* would be set to NULL, the Web Service falls back on *displayName* as we assume that this attribute will always be present. Our testing shows that it is possible to create a new user object without specifying *givenName* or *sn*, but it is not possible to create a new user object without specifying the *displayName* attribute.

Attribute	**Description**
displayName	The display name for an object. This is usually the combination of the user's first name, middle initial, and last name.
givenName	Contains the given name (first name) of the user.
sn	This attribute contains the family or last name for a user.
mail	The list of email addresses for a contact.
samAccountName	This attribute is used to store the SAM account names that correspond to the DNS host names in ms-DS-Additional-Dns-Host-Name[13].
memberOf	The distinguished name of the groups to which this object belongs.

Table 4-3 LDAP Attributes in User Object (Microsoft, All Attributes, 2015)

[12] This would make the hostname a URL and is only supported by OpenLDAP 2.x.x and up.

[13] The attribute is used to store the additional DNS host name of a computer object. This attribute is used at the time a DC is renamed (Microsoft, All Attributes, 2015).

Although the Web Service only use a few attributes in the AD, all fields in the AD is returned when using the Web Service' API to get user information about a specific user. This is because 3rd party applications may have different needs than the Web Service itself.

When searching for a given user the query string is parsed using the pseudo code in Figure 4-1.

```
//Assume a search like this query
String q = "Ruben Alexander Andreassen"
//Convert the string into an array
Array qA = explode (WHITESPACE, q)

//If the query has more than two words
if (qA.length > 2) {
        //Assume that the last word is the last name
        //and extract it
        String lastName = qA[(qA.length-1)]
        //Remove the last word from the array
        unset(qA[(qA.length-1)])
        //Build a new query with the last name and the
        //rest of the words from the array
        q = lastName+" "+implode(WHITESPACE, qA)
        //The last name is now at the beginning
}
//Insert the query into search filter
```

Figure 4-1 Pseudo code for doing a user search

As we can see from the pseudo code, we are assuming that the name in the query is formatted as *firstName middleName lastName*. If the query has more than two words, we move the last name to the beginning of the query. After formatting the query, we use the query in a LDAP search filter shown in Figure 4-2. The reason we move the last name to the beginning if the query has more than two words is because our initial testing revealed that it was difficult to search for persons with middle names. The user would do a search, but the search result was not as expected. After implementing the pseudo code in Figure 4-1 the search results matched the users' expectations.

```
(& //AND operator
        (objectCategory=person)
        (objectClass=user)
        (anr=query)
        (mail=*)
        (! //NOT operator
                (userAccountControl:1.2.840.113556.1.4.803:=2)
        )
)
```

Figure 4-2 LDAP search filter for doing a user search

In Figure 4-2 we tell the AD that five attributes need to match in the returned result. The first two attributes tells the AD that we are looking for a user account, then we use the *anr* attribute to search for the name. The mail attribute is set to a wildcard meaning that it need to be present. The last part of the filter is a bit more complex.

Some attributes on AD objects are composed of bitwise flags. The syntax of the LDAP Matching Rule is *attributename:ruleOID:=value* (Microsoft, How to query Active Directory by using a bitwise filter, 2014). In our search query for users, we don't want disabled accounts. This can be students that are finished studying, or staff that has left the university. We do this by checking that if the attribute *userAccountControl* has its value set to *2*, we do not want these objects. The cryptic number *1.2.840.113556.1.4.803* is the *AND* operator.

ANR stands for Ambiguous Name Resolution, and is an efficient search algorithm in the AD (Mueller, Active Directory: Ambiguous Name Resolution, 2014). Several attributes are used in this algorithm depending on which schema version the AD uses. Using Active Directory Explorer we have determined that the university uses the latest schema version which is 69 at the time of writing. Some of the attributes included in the ANR search is *displayName*, *givenName* and *sn*, which makes sense as we would search in these attributes if the *anr* attribute was not available.

To get a more deep understand why we need to move the last name to the beginning of the query if it contains more than two words, we need to look at the search filter that the *anr* attribute is converted into. We look at an example using AD schema version 13 for simplicity. The attributes and the values we use in the ANR example is listed in Figure 4-3. These are the attributes that the ANR search algorithm will be using. By comparing the values of these attributes to the ANR filter examples we can see where and why we get a match on the search.

```
displayName=Andreassen Ruben Alexander
givenName=Ruben Alexander
legacyExchangeDN=/o=UiT/ou=EAG/cn=R/cn=Andreassen Ruben Alexander (ran033)
name=Andreassen Ruben Alexander (ran033)
physicalDeliveryOfficeName=Institutt for informatikk
proxyAddresses=x500:/o=EL/ou=EAG/cn=R/cn=-Andreassen;SMTP:ran033@post.uit.no
sAMAccountName=ran033
sn=Andreassen
```

Figure 4-3 Attributes and values used in an ANR search. Note! Some of the values may be shortened for readability.

The query *Ruben* is converted to *anr=Ruben* that is converted into the filter shown in Figure 4-4. As we can see this filter will match on *givenName* from Figure 4-3.

```
anr=Ruben is converted into the following filter:
(|
        (displayName=ruben*)
        (givenName=ruben*) //Match
        (legacyExchangeDN=ruben)
        (physicalDeliveryOfficeName=ruben*)
        (proxyAddresses=ruben*)
        (name=ruben*)
        (sAMAccountName=ruben*)
        (sn=ruben*)
)
```

Figure 4-4 ANR filter example

When searching for only one word the *anr* attribute converts to a relatively simple filter. Let's look at what happens if we are searching for two words. The query *Ruben Alexander* or *Ruben Andreassen* are converted into *anr=Ruben Alexander* or *anr=Ruben Andreassen*. These attributes are converted into the filters shown in Figure 4-5.

anr=Ruben Alexander is converted into the following filter:

```
(| //OR operator
        (| //part1
                (displayName=ruben alexander*)
                (givenName=ruben alexander*) //Match
                (legacyExchangeDN=ruben alexander)
                (physicalDeliveryOfficeName=ruben alexander*)
                (proxyAddresses=ruben alexander*)
                (name=ruben alexander*)
                (sAMAccountName=ruben alexander*)
                (sn=ruben alexander*)
        )
        (& //part2
                (givenName=ruben*) //Match
                (sn=alexander*)
        )
        (& //part3
                (givenName=alexander*)
                (sn=ruben*)
        ) ) //END FILTER
```

anr=Ruben Andreassen is converted into the following filter:

```
(| //OR operator
        (| //part1
                (displayName=ruben andreassen*)
                (givenName=ruben andreassen*)
                (legacyExchangeDN=ruben andreassen)
                (physicalDeliveryOfficeName=ruben andreassen*)
                (proxyAddresses=ruben andreassen*)
                (name=ruben andreassen*)
                (sAMAccountName=ruben andreassen*)
                (sn=ruben andreassen*)
        )
        (& //part2
                (givenName=ruben*) //Match
                (sn= andreassen*) //Match
        )
        (& //part3
                (givenName= andreassen*)
                (sn=ruben*)
        ) ) //END FILTER
```

Figure 4-5 ANR filter example

As we can see the query *Ruben Alexander* gives us a match in part one and a partial match in part two, but both attributes in part two are required since the AND operator is used. This means that the partial match in part two is not taken into account in the result. *Ruben Andreassen* gives us a full match in part two of the filter.

The query *Ruben Alexander Andreassen* is converted into *anr=Andreassen Ruben Alexander* that gives us the filter shown in Figure 4-6.

As we can see this query gives us a two matches in part one of the filter, and also a full match in part three of the filter. The original query that where *Ruben Alexander Andreassen* would not have given us any matches (Microsoft Corporation, 2001).

anr=Andreassen Ruben Alexander is converted into the following filter:

```
(| //OR operator
        (| //part1
                (displayName=andreassen ruben alexander*) //Match
                (givenName=andreassen ruben alexander*)
                (legacyExchangeDN=andreassen ruben alexander)
                (physicalDeliveryOfficeName=andreassen ruben alexander*)
                (proxyAddresses=andreassen ruben alexander*)
                (name=andreassen ruben alexander*) //Match
                (sAMAccountName= andreassen ruben alexander *)
                (sn= andreassen ruben alexander*)
        )
        (& //part2
                (givenName= andreassen*)
                (sn=ruben alexander*)
        )
        (& //part3
                (givenName=ruben alexander*) //Match
                (sn=andreassen*) //Match
        )
)
```

Figure 4-6 ANR filter example

4.1.1.3 Courses

Information about which courses any given student is taking at the current moment is based on which groups a user is a member of in the AD. The *memberOf* attribute contains all the groups that a user is linked to. Each course is represented by a group, formatted as shown in Figure 4-7.

```
emner.<course name>.<year>.<semester>.<course version>.<term>.student
```

Figure 4-7 AD Course Group Format

The first and the last word is fixed. Course name and year are self-explaining. Semester can have two values, the Norwegian words for spring and fall (*vår* and *høst*). Each course can have several different versions in Syllabus, and the course version field reflect which version of the course the student is taking. The term field is information about which term of the student's study program the student is taking the course.

```
emner.inf-3320.2014.vår.1.6.student
```

Figure 4-8 AD Course Group Example

Figure 4-8 shows an example of a group in the AD. A user assigned to this group would mean a student taking version one of the course INF-3320 (Middleware), in the year 2014, in the spring semester, at the students sixth term (typically the last term in a Bachelor).

By finding all the groups that a user is a member of that starts with *emner* and ends with *student*, and parsing them to retrieve the course name and version, we are able to link that information to the courses in SDB and retrieve the course calendar. SDB stores the courses names as <course name>-<course version>. The example in Figure 4-8 would be parsed to INF-3320-1.

At first the group name in the AD did not contain the course version, this was added in the middle of this thesis. The code was then changed to use this information to improve the Web Service. In the first version of the code, where we did not know the course version, we had to search in Syllabus for the course code without the course version. This would give us all the course versions and we would use the latest version as a match. We would then have made the conclusion that this was good enough since we don't have information about the previous courses taken by students, and we would have to assume that the students are taking the latest version of the course. But since the version number was added, we can with 100% accuracy link the information in the AD with SDB.

4.1.1.4 Rooms

Although the room's calendar exist in MXS, all the rooms are also resources in the AD. It makes sense to search for rooms in the AD instead of MXS for two reasons; LDAP is more lightweight than SOAP, and AD is designed for a large number of search operations (Microsoft, So What Is Active Directory?, 2014).

Attribute	Description
displayName	The name of the room
mail	The email address of the room

Table 4-4 Attributes used in the Room Object

To filter the room resources from all the other resources in the AD we make sure the attribute *msExchRecipientTypeDetails* is set to *16*. A value of *16* indicates that the resource is a room mailbox (Bailey, 2013). The LDAP filter that is used in a room search is displayed in Figure 4-9.

```
(&
        (displayname=query*)
        (msExchRecipientTypeDetails=16)
)
```

Figure 4-9 LDAP search filter for a room search

Like the user object, the room object has a lot of attributes. We only use the two listed in Table 4-4. Since we don't care about the order of the names that make up a room name, we only need to retrieve the *displayName* attribute when handling rooms and not the *givenName* and *sn* attributes. The *mail* attribute is used as an identifier to link the room's resource in the AD to the room's calendar in MXS.

4.1.2 Exchange Connector (XC)

The Exchange connector communicates with the EWS API. This is done through SOAP messages which is essentially XML messages sent over the Internet. To implement this from scratch would be very time consuming, so we have taken an open source project as a base and added the missing functionality that we needed.

4.1.2.1 Exchange setup

To make the Web Service able to communicate to the university's MXS through the EWS interface we need to configure a few parameters. The configuration that is unique to the university's setup can be viewed in Table 4-5.

Parameter	**Value**
Hostname	mail.uit.no
Host Version	Exchange2010[14]

Table 4-5 Exchange setup

4.1.2.2 Detailed and Busy-Free calendar

The *GetUserAvailability* operation provides detailed information about the availability of a set of users, rooms and other resources within a specified time period. The Web Service use the email address as an identifier for the user or room that we want to retrieve calendar information from.

The XML response contains a *FreeBusyResponseArray* element, where each mailbox appears in a unique *FreeBusyResponse* element. The *FreeBusyResponse* element contains a *FreeBusyView* element, and in that element we find the *CalendarEventArray* element. This element contains one *CalenarEvent* element for each event. There are three attributes making up an event: *StartTime*, *EndTime* and *BusyType*. The *BusyType* element can contain any of the values listed in Table 4-6.

Value
Free
Tentative
Busy
OOF (Out of Office)
NoData

Table 4-6 Possible values of the BusyType element (Microsoft, BusyType, 2009)

By using this information, we can populate a calendar with information about when any user or room is busy or free. We know that a user or room is free when one of the following is true; there is an event where the *BusyType* element is set to "Free", or there is not any events at the given date and time

In addition to this, the *CalendarEvent* element contains a *CalendarEventDetails* element if the user has permissions to view this information. From the *CalendarEventDetails* element we can get detailed information about the logged in user's calendar events, and any other users or rooms that the logged in user has permissions to view detailed event information from. This permission is usually given when a user shares a calendar or event with other users. The *CalendarEventDetails* element contains several attributes, but we only use the ones listed in Table 4-7.

Attribute	**Description**
ID	Represents the entry ID of the calendar item.
Subject	Represents the subject of a calendar item.
Location	Represents the location field of the calendar item.

Table 4-7 Elements used in the CalendarEventDetails element (Microsoft, GetUserAvailability Operation, 2011)

[14] During the development of this thesis the university's IT department upgraded from Exchange 2010 to Exchange 2013. This does not affect the Web Service because none of the introduced operations in Exchange 2013 would have been used by the Web Service, and none of the operations the Web Service uses where deprecated (Microsoft, EWS operations in Exchange, 2014).

The description attribute of the event does not exist in the *CalendarEventDetails* element. The process of getting the description is a little bit more complicated. The ID attribute from the *CalendarEventDetails* element is a hexadecimal representation of the PR_ENTRYID property (Microsoft, IdFormat enumeration, 2014). To use this ID to get the description of the event we need to convert it to a EWS identifier format that is used in Exchange 2007 Service Pack 1 (SP1) and later versions of Exchange Server. The conversion is done using the *ConvertId* operation (Microsoft, ConvertId operation, 2014).

Figure 4-10 shows a simple request to convert an identifier (ID) from the hexadecimal representation (HexEntryId) to the EWS identifier format (EwsId) for an item in a user mailbox. If everything goes as expected we get a response with the converted id as shown in Figure 4-11. If an error occurs, we get a response explaining what went wrong like the one shown in Figure 4-12.

When we have the ID in the correct format, we can use the *GetItem* operation to retrieve calendar items from the Exchange store (Microsoft, GetItem operation, 2012) (Microsoft, GetItem operation (calendar item), 2013). We can get several events in one request by adding multiple IDs. The response contains all the elements of the *CalendarItem* element, but we are interested in the description that lies inside the *Body* element. The *Body* element can contain HTML or plain text, we use the plain text in the Web Service because the iCalendar specification only allows plain text to be used in the description of an event object (Desruisseaux, 2009).

Figure 3-3 illustrates that even if we wanted to get calendar information from twenty accounts, only three requests are sent between the Web Service and the EWS interface. If we could manage without the information in the *CalendarItem* element, which is the description in our case, only one request would have been sent. This becomes more important if we want calendar information from over 100 mailboxes, since we can only retrieve up to 100 mailboxes in each request.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
                    xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types">
<soap:Header>
  <t:RequestServerVersion Version="Exchange2010"/>
    </soap:Header>
  <soap:Body>
    <ConvertId xmlns=http://schemas.microsoft.com/exchange/services/2006/messages
                  xmlns:t=http://schemas.microsoft.com/exchange/services/2006/types
                  DestinationFormat="EwsId">
      <SourceIds>
        <t:AlternateId Format="HexEntryId" Id="AAMkAGZhN2IxYTA0LWNiNzItN="
                         Mailbox="user1@example.com"/>
      </SourceIds>
    </ConvertId>
  </soap:Body>
</soap:Envelope>
```

Figure 4-10 ConvertId operation request example (Microsoft, ConvertId operation, 2014)

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas..." xmlns:xsi="...w3..."  xmlns:xsd="...w3...">
  <soap:Header>
    <t:ServerVersionInfo MajorVersion="8" MinorVersion="1" MajorBuildNumber="191"
                    MinorBuildNumber="0" Version="Exchange2010" xmlns:t="http://schemas..." />
  </soap:Header>
  <soap:Body>
    <ConvertIdResponse xmlns="http://schemas...">
      <ResponseMessages>
        <ConvertIdResponseMessage ResponseClass="Success">
          <ResponseCode>NoError</ResponseCode>
          <AlternateId xsi:type="t:AlternateIdType" Format="EwsId" Id="RgAAAAAS2%2"
                        Mailbox="user@example.com" />
        </ConvertIdResponseMessage>
      </ResponseMessages>
    </ConvertIdResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 4-11 ConvertId operation response example (Microsoft, ConvertId operation, 2014)

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
                    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <ServerVersionInfo MajorVersion="8" MinorVersion="1"
                          MajorBuildNumber="206" MinorBuildNumber="0"
                          Version="Exchange2010"
                          xmlns="http://schemas.microsoft.com/exchange/services/2006/types" />
  </soap:Header>
  <soap:Body>
    <ConvertIdResponse
                        xmlns="http://schemas.microsoft.com/exchange/services/2006/messages">
      <ResponseMessages>
        <ConvertIdResponseMessage ResponseClass="Error">
          <MessageText>Id is malformed.</MessageText>
          <ResponseCode>ErrorInvalidIdMalformed</ResponseCode>
          <DescriptiveLinkKey>0</DescriptiveLinkKey>
        </ConvertIdResponseMessage>
      </ResponseMessages>
    </ConvertIdResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 4-12 ConvertId operation error response example (Microsoft, ConvertId operation, 2014)

4.1.2.3 Room List

To retrieve a list of all room lists from the MXS, we use the *GetRoomLists* operation in the EWS interface (Microsoft, GetRoomLists Operation, 2011). The *GetRoomLists* operation will return all the room lists available within the Exchange organization.

The response is in XML and the element containing room lists is named *RoomLists*. The *RoomLists* element contains child elements called *Address*. There is one *Address* element for each room list. The *Address* elements child elements contain information about a specific room list. There are four attributes that make up the room list information, but we only use the *Name* and *EmailAddress* attribute. These attributes contains the information that the names of the attributes implies.

The Web Service implements a method to search for a particular room list, but the EWS don't have any functions to do this. We therefore have to retrieve all room lists from the Exchange organization, parse through them and only return the ones that match the search query.

4.1.2.4 Room

We have previously discussed that we use the AD to search for particular rooms, but when we want to retrieve all the rooms in a particular room list, we have to use the EWS interface. This is because the AD don't contain information about which room list a particular room is a member of. Since the university uses room lists as logical groups of rooms at a physical location, they could use the LDAP attribute *physicalLocationObject*[15] to indicate that a room is within a particular room list. But they have to do this consistently, or else it would not work very well. Our testing shows that in some cases, the university uses this attribute to describe the location even more specific than the room list would as shown in Table 4-8. Instead, we use the *GetRooms* operation to get the rooms within a specified room list (Microsoft, GetRooms Operation, 2011).

The XML response contains a *Rooms* element, where the child elements *Room* contains information about each room. Each *Room* element contains an *Id* element with the room information. There are four attributes that make up the room information, but we only use the *Name* and *EmailAddress* attributes.

Room	**Room List**	**physicalLocationObject**
MH L8.121 Møterom	MH-Bygget	MH bygget plan 8
REALF A042 Møterom	Realfagsbygget	Realfagsbygget
Teknologibygget 1.027 Kollokvierom	Teknologibygget	Teknologibygget, 1etg

Table 4-8 Examples of values in the physicalLocationObject attribute compared to the Room List that a specific Room is associated with

[15] Used to map a device (for example, a printer, computer, and so on) to a physical location. (Microsoft, All Attributes, 2015)

4.1.3 Syllabus Connector (SC)

The university runs an older version of Syllabus that don't have any API that 3rd party applications can use to retrieve information. The solution is therefore to retrieve information directly from the Syllabus database (SDB). This is an Oracle database and the Web Service uses SQL statements to retrieve information.

In addition to this, the database is very poorly documented (Scientia, All SDB Schema Tables, 2003). We got some documentation on the table and column names, but that's it. We therefore had close contact with the university's IT department to understand how the information in the database should be interpreted, and we got some example queries to get started.

The calculation of events is complicated, but necessary to describe in detail because it affects how the Web Service performs.

4.1.3.1 SDB setup

To make the Web Service able to communicate with the university's SDB we need to configure a few parameters. The configuration that is unique to the university's setup can be viewed in Table 4-9.

Parameter	**Value**	**Description**
Database Username	Not Public	
Database Password	Not Public	
Database Connection String	MISC	The connection string is defined in a configuration file named tnsnames.ora. The content of the file can be viewed in Figure 4-13.
Database Start Year	Previous year	We can only search for data as far back as this value.

Table 4-9 Syllabus Database Setup

```
MISC =
 (DESCRIPTION =
  (LOAD_BALANCE = yes)
  (ADDRESS = (PROTOCOL = TCP)(HOST = nille-vip.uit.no) (PORT = 1771))
  (ADDRESS = (PROTOCOL = TCP)(HOST = helene-vip.uit.no)(PORT = 1771))
  (CONNECT_DATA =
   (SERVICE_NAME = MISC)
  )
 )
```

Figure 4-13 Content of the database connection configuration file (tnsnames.ora)

4.1.3.2 Courses, rooms, room lists and staff

The SDB contains the timetable of many things. We will look at courses, rooms and staff. The way that this is built up is that there is one table with all the courses, one table with all the rooms and one table with all the staff at the university. These are the tables we use as base tables to do searches against.

Then we have the activity table, which contains all the activities. Each activity represents a series of events. Each course, room and staff can be linked to zero or more activities. To achieve a many-to-many relation like this, the database has linking tables between each base table and the activity table (Caffrey, 2011).

The staff table contains email addresses of the staff, and these email addresses can be linked to the AD. This enable us to do the user search in the AD, and then check if the user is a staff at the university by searching for the users email address in the SDB. If the user is a staff, we can get all the courses the user is teaching. This is much easier than if the user is a student, since we then have to parse the groups in the AD and then link the parsed groups to the courses in the SDB.

If we retrieve the activities of a course or staff, we can also check the linking between the activities and the rooms to get the location. If we want to get the activities of a room, that room is naturally the location.

Syllabus also group rooms into room lists in the same way as the Exchange Server. The only difference here is the semantics. What is called a room list in Exchange, is called a zone in Syllabus.

4.1.3.3 Activity

An activity in the SDB is like a repeated event. Unlike other systems where the system calculates the repeated event into several events, we have to do this calculation ourselves on the Web Service. This calculation is quite complicated at first sight. Let's look at the following columns and values in one activity from the activity table *uit_activity* shown in Table 4-10.

Name	**Value**
scheduled_periods	((1 3 4 5 6 7 8 9 13 14 15 16 17) (205))
duration	3
week_pattern	0101111111000111110000000000000000000000000000000000

Table 4-10 Selected columns from the uit_activity-table

This means that the activity is in week 1, 3, 4, 5, 6, 7, 8, 9, 13, 14, 15, 16 and 17 from *scheduled_periods*. The corresponding value in *week_pattern* is also set to one. The activity starts in the period 205 (from *scheduled_periods*) and the *duration* is set to 3, meaning that the event is 3 periods long. This means that the first event of the activity is in week 1, and starts at period 205, and ends at period 208. We need some more information to determine the date and time.

From a support table *uit_institute_setup* in SDB we can get the information shown in Table 4-11.

Name	**Value**	**Description**
start_time_of_day	25200	When the day starts for all activities in the database
end_time_of_day	79200	When the day ends for all activities in the database
periods_per_day	60	How many periods of 15 minutes between the start and end time of the day

Table 4-11 Selected columns from the uit_institute_setup-table

The number from *start_time_of_day* and *end_time_of_day* is the number of seconds from midnight (00:00:00). If we calculate the actual start and end time from these seconds, we get the times shown in Table 4-12. We can use these times, or we can use only the seconds to calculate how long each period is using the value from *periods_per_day* as shown in Figure 4-14.

Value	**Time (calculated)**
25200	07:00
79200	22:00

Table 4-12 Seconds convertet to time

(((79200 seconds – 25200 seconds)/60 seconds)/60 periods per day) = 15 minutes

((22 hours – 7 hours)/60 periods per day) = 15 minutes

Figure 4-14 Calculation of the length of each period

Each week starts on a Monday (Monday and not Saturday from the documentation), and each day starts at 07:00 and ends at 22:00. In our example, the activity starts at period 205 and has a duration of 3 periods. Table 4-13 shows how we use that information to calculate the day, start time and end time of the activity.

From the calculation we now know that the first event of the activity starts in week one on a Thursday at 13:15 and ends at 14:00. We still need to find the actual date. This is simply done by calculating which date a Thursday was on in the given week of the given year. From *week_pattern* we can see that the activity is repeated in week 3, 4, 5 and so on. We calculate the dates of all the Thursdays of the weeks in the week pattern to get all events in the activity.

Attribute	**Calculation**	**Result**	**Description**
Day	(205 periods/60 periods per day)	Day 3	The value here represents the day of the week. 0 = Monday 1 = Tuesday 2 = Wednesday **3 = Thursday** 4 = Friday 5 = Saturday 6 = Sunday
Start Time	(205 periods%60 periods per day)	25 periods	Since the day starts at 07:00 we need to add this to the time we get. This means that 6 hours and 15 minutes are actually meaning 13:15 when we add 7 hours.
	(25 periods*15 minutes per period)	375 minutes	
	375 minutes converted to hours and minutes	6 hours and 15 minutes	
	(6 hours and 15 minutes) + 7 hours	13:15	
End Time	(13:15 + (3 periods * 15 minutes per period))	14:00	We need to add the duration of the activity to get the end time.

Table 4-13 How to calculate the day, start and end time of an activity

While testing the Web Service we discovered that the Syllabus connector was quite slow when dealing with a large number of activities. By optimizing the code we got a whole 50% increase in performance when dealing with a large number of activities. The main problem was that the first version of the code calculated all events of an activity by default. This lead to unnecessary processing when an activity did not have events in the weeks that the date interval limited. We then modified the code to check if the activity has events in the weeks that we want events from first, and then calculate the events if there is any in the given weeks. This check is based on the *week_pattern* attribute.

4.1.4 Fronter Connector (FC)

The Fronter connector handles requests between the Web Service and the OpenApi that Fronter provides. The requests are GET requests sent over HTTP, and the responses are in iCalendar format.

The most challenging with the FC is the authentication needed to use the OpenApi. Fronter does not support automatic authentication and relies on user interaction to authenticate. We did not want to implement the authentication this way because that would limit the user experience on our service. We therefore needed to do some reverse engineering to solve this problem.

4.1.4.1 Fronter setup

To make the Web Service able to communicate with Fronter we need to configure a few parameters. The configuration that is unique to the Fronter setup can be viewed in Table 4-14.

Parameter	**Value**
Consumer key	Not Public
Consumer secret	Not Public
Consumer name	uit
OAuth host	https://fronter.com/uit
Request token URL	https://fronter.com/uit/api/oauth.php/requesttoken
Authorize URL	https://fronter.com/uit/api/oauth.php/authorize
Access token URL	https://fronter.com/uit/api/oauth.php/accesstoken

Table 4-14 Fronter setup

4.1.4.2 Automated OAuth authorization

To access the OpenApi functions in Fronter, the 3rd party application needs to authenticate through OAuth. From the Fronter documentation:

“When a 3rd party application want's to call the OpenApi in Fronter, they first get a request-token from Fronter. Then the user gets a GUI from Fronter asking him to log in. If the user is already logged in, the user will be asked if he want to grant access to the 3rd party application. When granting access, the user is redirected back to the 3rd party application. The request-token is now enabled. The 3rd party app exchanges the request-token for an access-token. Now the 3rd party app can do OpenApi calls with the access-token, until the token expires (default 2 hours).

When the 3rd party app gets the request-token, and the user must grant access, this GUI will NOT redirect the user to the login page. If the user is not logged in, they must open their normal Fronter login page and then login. After that, return to the OAuth access page and refresh.

Fronter is doing it this way for security reasons. We want to protect the users from any phishing attempts.” (Fronter, Oauth In Fronter, 2014).

This would have been acceptable if our Web Service only had the GUI. But since we want our Web Service to make the information available to other 3rd party applications, this procedure would be highly unpractical. In addition, we know from experience that normal users want to login as few times as possible.

With this motivation we analyzed the login process that Fronter uses. The findings of how the Fronter login process works is described in Appendix A.

To automate this process we created a class with functions that simulate a browsers behavior. By doing this, the web servers at the other end will think that our Web Service is a user browsing the web trying to log in to Fronter. The PHP library used to simulate browser behavior is cURL and is a default library embedded in the PHP installation (The PHP Group, Client URL Library, 2015).

When looking at the login process in Appendix A, it becomes clear that we need to interfere with the step two, six, seven and eight. Step two and eight because cURL don't have a JavaScript engine, and step six and seven because they require user input.

We have control over the user input, because the organization (org input field) is fixed, and the username and password (feidename and password input fields) are provided by the user authenticating with the API or through the GUI.

One big drawback in doing this, are the possibilities that the flow of the login process can change. If the login flow would to change, we would need to change the code automating the login process. This actually happened during the course of this thesis.

Step two in the Fronter login process was removed at some point during the development. This caused the automatic login to fail. We therefore needed to reverse engineer the whole process again to find out why the automatic authentication failed. After some debugging, it was clear that one of the JavaScript redirect pages was removed. We then needed to modify the code to accommodate for this, and everything worked again, until there is another change in the login process.

The experience this gives us is that everything can be done, but the use of these kinds of hacks need close monitoring and is highly time consuming to maintain. But the user experience is greatly improved.

4.1.4.3 User calendar

Each user has its own private calendar in Fronter. To retrieve this calendar information we use the iCal method in the Fronter OpenApi (Fronter, OpenAPIv1, 2014). The iCal method takes the three parameters described in Table 4-15.

Figure 4-15 shows an example API call to the iCal method. This will output all calendar events in August 2014 from the user's private calendar in Fronter as shown in Figure 4-16. The output format is specified as raw, which means that the output is the same as it would be in an iCalendar file. We then need to parse this result to retrieve the calendar events.

Parameter	**Description**
alphanum_format=raw	When specifying the value raw Fronter will output raw iCalendar data instead of data wrapped in JSON or XML. This is the recommended setting.
date_from	Displays calendar starting from the specified date. The format is Y-m-d. The default value is 30 days back from the current date of the request if not specified.
date_to	Displays calendar **up to the specified date**[16]. The format is Y-m-d. The default value is 90 days forward from the current date of the request if not specified.

Table 4-15 Fronter OpenApi iCal method parameters (Fronter, OpenAPIv1, 2014)

```
https://fronter.com/uit/api/v1.php/ical;private?date_from=2014-08-01&date_to=2014-09-01&alphanum_format=raw
```

Figure 4-15 Fronter iCal method private calendar call example

[16] This does not include the specified date. If we want the date 2014-08-31, we need to specify 2014-09-01.

```
BEGIN:VCALENDAR
VERSION:2.0
METHOD:PUBLISH
PRODID:-//Fronter/Calendar ICS export//EN
BEGIN:VEVENT
CREATED:20140816T080000
LAST-MODIFIED:20140816T080000
DTSTAMP:20140816T080000
UID:uit-96890
SUMMARY:test
DESCRIPTION: Owner: Ruben Alexander Andreassen\n
ORGANIZER;CN=Ruben Alexander Andreassen:MAILTO:ran033@post.uit.no
URL;VALUE=URI:http://fronter.com/uit
 /calendar/index.phtml?bm=01&bd=16&by=2015&showroomcalendars=1
DTSTART:20140816T070000Z
DTEND:20140816T080000Z
LOCATION:Private room
END:VEVENT
END:VCALENDAR
```

Figure 4-16 Fronter iCal method private calendar response example

4.1.4.4 Room calendar

A room in Fronter is not the same as a room in MXS or SDB. The rooms from the latter systems are physical rooms, but the rooms in Fronter are virtual representation of courses, student sets, student programs and other groups. A room in Fronter is therefore not mixed with the rooms from MXS and SDB, but threated as a normal calendar connected to the user that is a member of the room.

The method for retrieving calendar information from a room is the same as for the user calendar with one minor change. We specify *room* instead of *private* in the API call. An example of such a call can be viewed in Figure 4-17.

```
https://fronter.com/uit/api/v1.php/ical;room?date_from=2014-08-01&date_to=2014-09-
01&alphanum_format=raw
```

Figure 4-17 Fronter iCal method room calendar call example

This will output all calendar events in August 2014 from all the rooms that a user is a member of in Fronter. The output would be similar to the one shown in Figure 4-16.

4.1.5 Adding new connectors

The first thing the new connector need to implement is the authentication method of the interface of the external system. If the user uses the same credentials on the new external system that is going to be implemented, the credentials can be retrieved from a static object that the Web Service offers. If not, the connector must retrieve the credentials itself.

When the authentication is in place, the connector must implement a method to retrieve calendar information from a specific calendar at the external system. This method must take the identifier of the calendar that is going to be retrieved and a date interval as input.

When all this functionality is in place, the connector must implement a function that uses the connector's class in the main calendar class of the Web Service. This function must receive an identifier and use the class' date interval to retrieve calendar information from the connector. Then the function must parse the calendar information and create an event object for each event. These event objects must be added to the calendar class' calendar list. The function can create a new calendar in the list or use an existing calendar in the list to combine the connector's calendar to another calendar. The Web Service will then hand the calendar list to the interfaces that delivers the calendars to the end user or end user system.

If the connector implements searching, almost the same procedure is needed. The Web Service has a search class that the connector can register its searching possibilities. The search result can be independent or combined with the other search functions such as rooms or room lists.

4.2 Middleware

To handle all the different data formats and semantics from the connectors we need to convert the data into an internal format. We do this by creating an event class that each calendar event from all the systems are converted into.

This class contains the key fields of an event. This includes date, time, subject, location and description of the event. Location and description is optional.

When we have all the data in our own representation of a calendar event, we can format the data correctly and hand it over to the interfaces. This can be in an iCalendar format, JSON representation or an object depending on which interface that wants to retrieve the data.

4.3 Interfaces

The interfaces takes all the information from the middleware and make the information available to the user or other 3rd party system.

4.3.1 Graphical User Interface (GUI)

The GUI is a web page that any student or staff at the university can use. First, the user has to authenticate through a login page, and then the user can use all of the functionality that the Web Service offers. The documentation of the API can also be found in the GUI if the user wants to develop a 3rd party application that uses the functionality that the Web Service offers.

The URL of the GUI is https://studentlink.ifi.uit.no/. We also have a certificate issued by the university. During this thesis, the Heartbleed bug was known. When this bug was known we patched our web server right away and the certificate was changed to make sure that the Web Service could not be compromised.

The biggest challenge with the GUI is that the Web Service needs to authenticate to some of the external systems on every request. Since the user only has to authenticate when he logs in, the Web Service need to store the username and password in a safe manner. Since the Web Service needs to use the password, the password cannot be hashed. The username and password should never be stored at the Web Service in clear text, or any way that is reversible, since the Web Service could be compromised.

Since we only want to forward the credentials and do not intend to store it, but also don't want the user having to re-enter the credentials many times while using the GUI, we came up with the following solution:

1. Encrypt the username and password using 256 bit AES encryption.
2. Store the encrypted username and password in the cookie. The user's browser needs to have cookies enabled to do this since the cookies are stored in the browser.
3. Store the encryption keys in the session store at the Web Service.

With each request the user's browser will send the cookie. The Web Service can then use the keys in the session to decrypt the data stored in the cookie. Someone exploiting the Web Service and gaining access to the session store will be able to steal encryption keys, but they need the user's cookie to get the encrypted username and password. If we would to store the encrypted credentials in the session, someone gaining access to the session store could try to brute force the data. Gaining access to the keys are useless without the cookie. The only problem we can see is if someone with root access is reading the Web Service process' memory between the time that the Web Service receives the cookie and the script exiting from execution. SquirrelMail uses the same scheme as we have described here (SquirrelMail, 2014).

An alternative method would be to implement HTTP Basic Access Authentication (Franks, et al., 1999). With this method, the user would be prompted for the credentials when he first visit the GUI. The browser would then cache the credentials for a period of time, and send them to the Web Service on each request. No cookies or sessions are needed. One of the reasons that we do not want to use this method for the GUI is that there would be no way for a user to log out. It would be very unsafe if the user is using a public computer and has no possibilities to log out safely. However, we use this authentication method in the API. Then the application using the API could handle log outs.

We also implement the EU regulation that states that all websites in Europa must inform the users that cookies is being used and what they are being used for (European Commission, 2014) (Samferdselsdepartementet, 2013). The full notice can be viewed in Appendix B.

4.3.2 ICAL Interface (ICAL)

The ICAL interface creates an iCalendar file on the fly. Much like the iCal method in the Fronter OpenApi. Many 3rd party systems can use this file to integrate the calendar info from our Web Service in their own service. Examples of these kinds of systems are listed in Table 4-16.

Calendar service
Google Calendar
Outlook Calendar
ICalSync2 app for Android

Table 4-16 Example of existing calendar services

In these systems the user only need to enter the URL to the shared calendar, and the system will display calendar the calendar information. An example URL is shown in Figure 4-18.

```
https://studentlink.ifi.uit.no/indexICAL.ics?course=INF-2700-1
```

Figure 4-18 ICAL example

We don't implement any authentication on this component for one reason; most of the systems that can use this function don't support any authentication. This means that we can only allow public information to be accessed this way. The only public information we have is the calendar information of courses and student sets from Syllabus. This information is already available through the university's online timetable portal. We could provide private information, but then we would need to implement a unique URL for each user that could not accidently be guessed. And the user would need to be informed of the risks, like that the information could be sniffed or accessed by other users if the URL would go astray. Google Calendar does this.

Also, since our Web Server retrieves calendar information from external systems that require authentication on every request, the user's credentials would need to be stored at the Web Service. We take security very serious, and will not have anything to do with storing passwords that can be decrypted.

We implement the VCALENDAR header properties described in Table 4-17. In addition to the standard header properties we implement a set of non-standard properties shown in Table 4-18. The non-standard properties has an "X-"-prefix according to section 3.8.8.2 of the RFC 5545 specification (Desruisseaux, 2009).

It is not without risk to implement non-standard properties. Although the X-WR-TIMEZONE is commonly used by known services such as Google Calendar and Apple's iCal application, and the property don't give any error messages in various iCalendar validators, the events time can be faulty processed by systems that strictly implement the RFC 5545. The correct way to implement a time zone is to add a VTIMEZONE component after the header, before the VEVENT's (Desruisseaux, 2009). And then use the TZID property along with the DTSTART and DTEND properties. We decided to implement the non-standard property since it seems to be widely supported.

The VEVENT has several different attributes. We implement the VEVENT properties described in Table 4-19 and one non-standard property described in Table 4-20.

Property	**Purpose**
PRODID	Product identifier. This property specifies the identifier for the product that created the iCalendar object.
VERSION	This property specifies the identifier corresponding to the highest version number or the minimum and maximum range of the iCalendar specification that is required in order to interpret the iCalendar object.
CALSCALE	This property defines the calendar scale used for the calendar information specified in the iCalendar object. The default value is "GREGORIAN".
METHOD	This property defines the iCalendar object method associated with the calendar object.

Table 4-17 Header properties in the iCalendar implementation (Desruisseaux, 2009)

Property	**Purpose**
X-WR-CALNAME	The display name of the calendar.
X-WR-TIMEZONE	The time zone of the calendar.
X-WR-CALDESC	A description of the calendar

Table 4-18 Non-standard header properties in the iCalendar implementation

Property	Purpose
DTSTART	This property specifies when the calendar component begins.
DTEND	This property specifies the date and time that a calendar component ends.
UID	This property defines the persistent, globally unique identifier for the calendar component.
DESCRIPTION	This property provides a more complete description of the calendar component than that provided by the "SUMMARY" property.
LOCATION	This property defines the intended venue for the activity defined by a calendar component.
STATUS	This property defines the overall status or confirmation for the calendar component.
SUMMARY	This property defines a short summary or subject for the calendar component.
TRANSP	This property defines whether or not an event is transparent to busy time searches.
DTSTAMP	In the case of an iCalendar object that specifies a "METHOD" property, this property specifies the date and time that the instance of the iCalendar object was created. In the case of an iCalendar object that doesn't specify a "METHOD" property, this property specifies the date and time that the information associated with the calendar component was last revised in the calendar store.
CREATED	This property specifies the date and time that the calendar information was created by the calendar user agent in the calendar store.
LAST-MODIFIED	This property specifies the date and time that the information associated with the calendar component was last revised in the calendar store.
SEQUENSE	This property defines the revision sequence number of the calendar component within a sequence of revisions.

Table 4-19 VEVENT properties in the iCalendar implementation (Desruisseaux, 2009)

Property	Purpose
X-MICROSOFT-CDO-BUSYSTATUS	Specifies the BUSY status of an appointment.

Table 4-20 Non-standard VEVENT properties in the iCalendar implementation (Microsoft, 2.1.3.1.1.20.31 Property: X-MICROSOFT-CDO-BUSYSTATUS, 2015)

4.3.3 Application Programming Interface (API)

The Web Service implements a modified REST-based API. This means that the API is RESTful implemented, using standard HTTP protocol and simple responses in JSON format. The most used methods in a REST-based API is described in Table 2-1.

To use the API, authentication is required. We implement HTTP Basic Access authentication (BA) because it is easy to implement and use standard HTTP headers (Franks, et al., 1999). We consider BA to be sufficient since the Web Service use HTTPS. Without HTTPS BA is not safe since it only use Base64 to encode the username and password. A better authentication scheme would be to use OAuth 2.0 (Hardt, 2012). Amazon has also created a custom authentication scheme to their S3 solution (Amazon, 2006). Since the main focus of this thesis is to combine calendar information and not on security, we consider it further work to change the authentication method of the API.

The Web Service only implements the GET and POST method. This choice was easy to make since we don't allow any changes to the data, only read operations.

4.4 Storage

Although the Web Service does not contain any state, we need to temporarily store some data. One of the most obvious reasons to use the file storage is for the function that lets the user upload iCalendar files in the GUI. We store the files in a temp folder and the filename is stored in the session. When the session expires, the file cannot be accessed because the Web Service don't know the filename without the session.

When the Web Service retrieves data from Fronter, we get data in a raw iCalendar format. We could parse this data directly and display it to the user since the data is retrieved in each request and not cached, but the iCalendar parser library that we use does not support this. We therefore need to do a workaround where we store the iCalendar data to a file, and then use the library to parse the file. Although we have done it this way, one should of course find another library or modify the existing library to avoid doing this workaround. This would be a slight performance optimization.

The last usage of file storage we have is less obvious than the previous explained reasons. Some of the external systems uses cookies that the Web Service needs to keep track of. The EWS interface expects a cookie with the name *exchangecookie* to be set, or else we get the HTTP Error: 401 Unauthorized response. The importance of this is poorly documented, but we found a very important sentence in an Exchange development blog stating that "*If you create a custom client, make sure that the **exchangecookie** cookie persists for each mailbox you are accessing.*" (Mainer, 2011).

The other cookies we need to keep track of is created when we emulate a browser to access the OpenApi in Fronter. This process require several redirects, and cookies are used in this process to keep track of how far in the login process the browser[17] has reached.

One optimization that we have not done is to clear the temp folder periodically to free disk space. This can be done on a daily, weekly or monthly basis based on how fast the temp folder grows.

The MySQL database is a part of a library that we are using to handle authentication with the Fronter OpenApi. The library implements an OAuth consumer client that connects to the Fronter server. The Web Service needs to get one access token for each user that is using the Web Service, and the database keeps track of this mapping, and other information related to the OAuth authentication process.

4.5 Summary

In this chapter we have learned how to actually connect the connectors to the external systems. There are many details that needs to be taken into account when communicating with the external systems, and how to interpret the data retrieved. The most important thing is to understand how to use the API of the external system and the limitations of the API.

The middleware takes it all together and deliver the data to the interfaces. The interfaces delivers the data to the end user. We have created interfaces for a variety of usages, but the GUI is where we really see what has been created.

In the next chapter we will do some testing and see how the Web Service compares to the university's existing solutions.

[17] Browser meaning the simulated browser behaviour that the Web Service uses to authenticate against Fronter

5 Testing

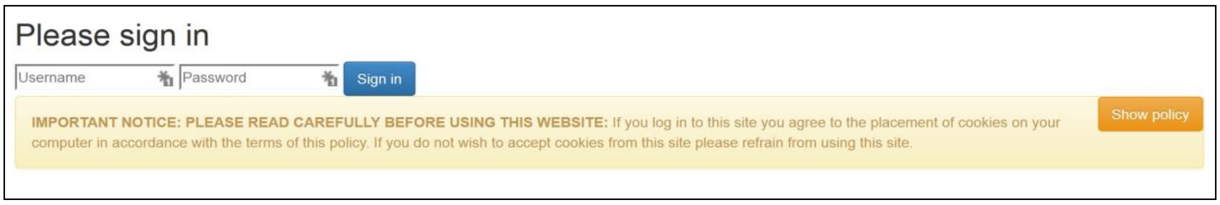
In this chapter we will explain how we tested the Web Service. We will focus on the user experience through the GUI and mention how other 3rd party systems can use the Web Service. We will also compare our solution with the university's existing solutions, and see how the Web Service performs.

5.1 Graphical User Interface (GUI)

5.1.1 Login

The GUI is accessible through the URL https://studentlink.ifi.uit.no/. When the user first access this URL the login page appears as shown in Figure 5-1. The page is simple, asking for a username and password. In addition to this, we have a notice explaining how the web site uses cookies as required by the EU Cookie Directive (European Commission, 2014).

The cookie notice explains what cookies are, and what our web site stores in the cookie. The whole cookie notice can be read in Appendix B.

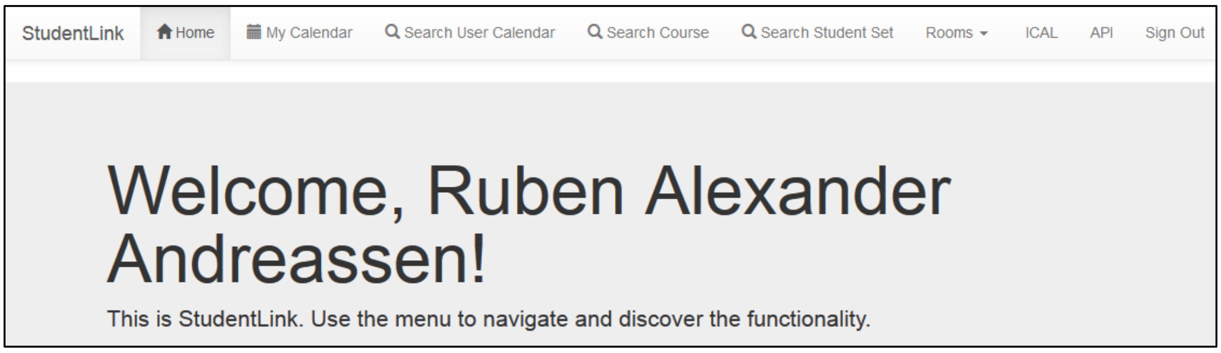


IMPORTANT NOTICE: PLEASE READ CAREFULLY BEFORE USING THIS WEBSITE: If you log in to this site you agree to the placement of cookies on your computer in accordance with the terms of this policy. If you do not wish to accept cookies from this site please refrain from using this site.

Figure 5-1 Login page

5.1.2 Home

When the user has successfully logged in we display a short greeting as shown in Figure 5-2. When the home page is successfully loaded, we know that the Web Service is able to communicate with the university's AD.



Welcome, Ruben Alexander Andreassen!

This is StudentLink. Use the menu to navigate and discover the functionality.

Figure 5-2 Home page

5.1.3 My Calendar

There is a menu on the top of the web page that the user can use to navigate. Instead of visiting the web sites in Figure 5-3, the user just clicks on My Calendar and the Web Service displays the page in Figure 5-4.

https://fronter.com/uit/

http://timeplan.uit.no/

https://mail.uit.no/

Figure 5-3 Web sites with calendar information

In other words, My Calendar displays calendar information from all the calendar systems that the Web Service implements. We can already see that we only had to log in once instead of twice (timeplan.uit.no does not require the user to log in), and we did not have to search for all the courses we are taking at timeplan.uit.no.

The page loads quickly[18], except for the first time. The first time the page loads the Web Service needs to authenticate with Fronter as explained in chapter 4.1.4.2 Automated OAuth authorization. This can take between one and three seconds.

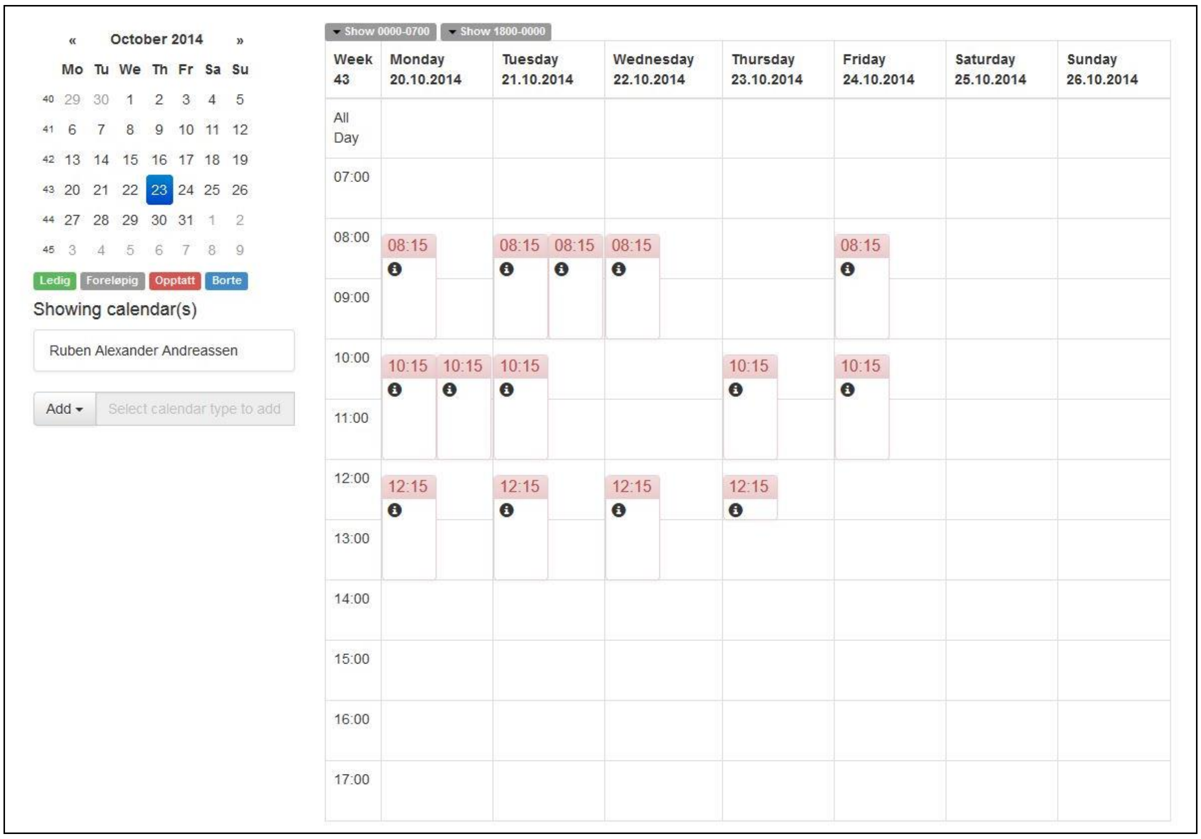
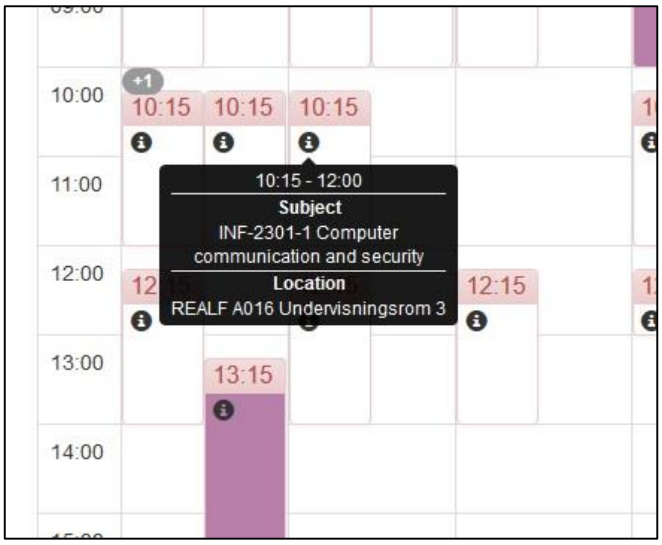


Figure 5-4 My Calendar

[18] The page average between one and two seconds



10:15 - 12:00
Subject
INF-2301-1 Computer communication and security
Location
REALF A016 Undervisningsrom 3

Figure 5-5 My Calendar detailed event view

The main calendar only shows basic information about when the event starts and how long the event is planned to be. If we hover the cursor over the black info icon we get information about the events subject, and if available, description and location as shown in Figure 5-5.

The My Calendar is already useful, but we have taken it one step further. It is possible to bring even more information into the calendar through the *Add* function displayed in Figure 5-6.

The *Add* function makes it possible to search and add calendar information about other users, courses, rooms or student sets. Which information we get from adding the different types are explained in more detail in the following sub chapters. It is also possible to upload iCalendar files generated by other calendar systems such as Google Calendar. In retrospect, we could have implemented a function to add calendars by URL. This would require some state so that the user don't have to do this every time he access the GUI.

This makes My Calendar into a powerful planning tool where the user can plan a meeting with another person at a specific meeting room as shown in Figure 5-7 and Figure 5-8. When all the calendars are added we only need to find a date and time where there is no events to figure out when both persons can meet at the specific meeting room. Each calendar we add will get a different color so it is easy to tell them apart.

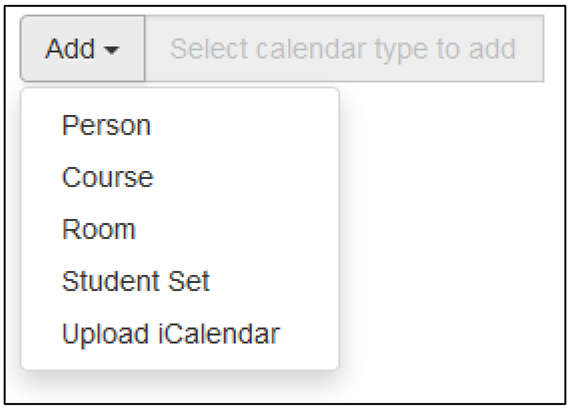


Figure 5-6 My Calendar add calendars

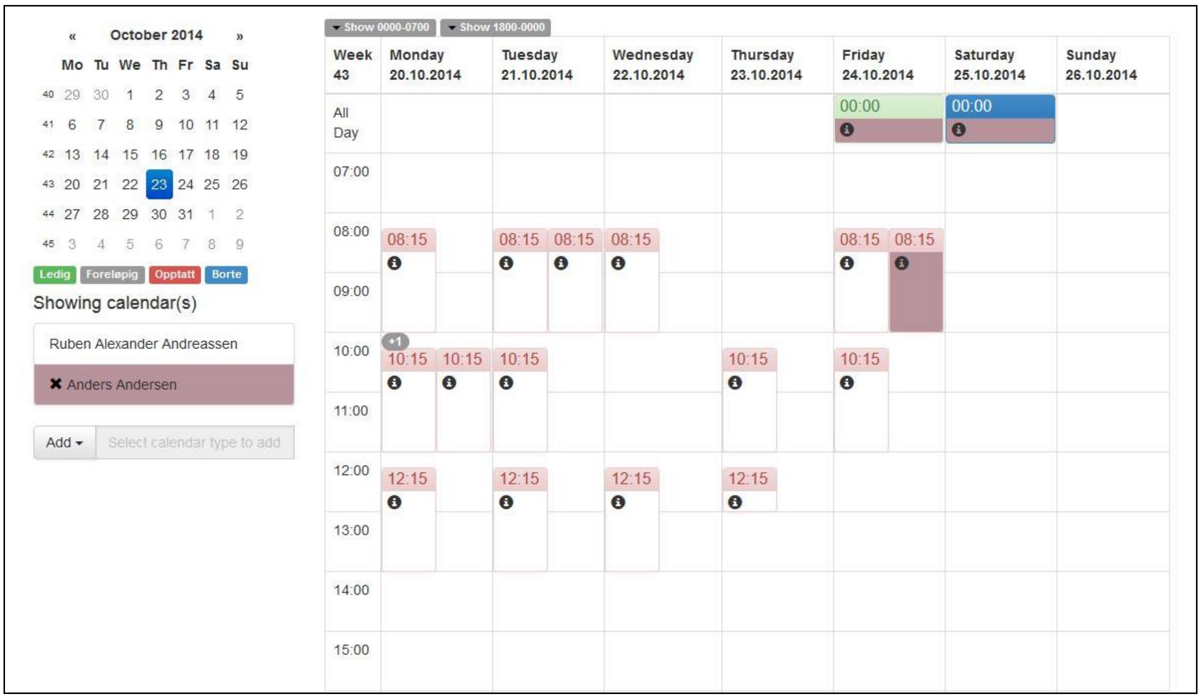


Figure 5-7 My Calendar merged view

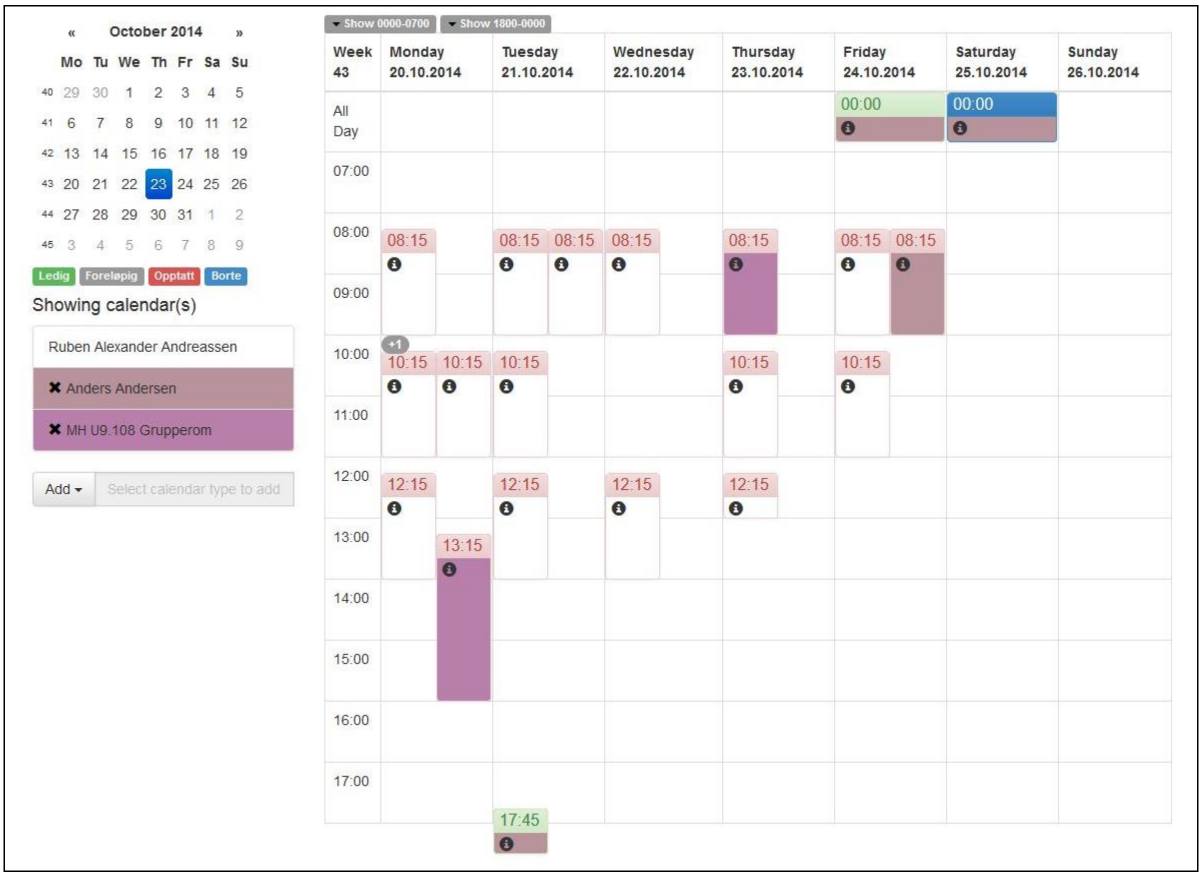


Figure 5-8 My Calendar merged view

We only allow two events to display at the same time and date in the main calendar view. If there are more than two simultaneous events a gray circle with a +x is displayed roughly where there are simultaneous events. When we click on the circle all events that day is displayed. The x is the number of simultaneous events not shown in the main view.

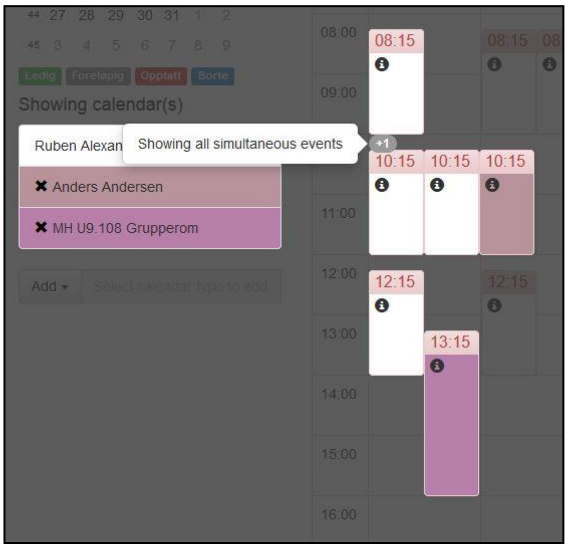


Figure 5-9 My Calendar several simultanious events

5.1.4 Search User Calendar

This page makes it possible to search for any user in the AD and display all public calendar information from that users calendars. This means that we can view whether the user is busy or free in Exchange, and which courses the user is taking or teaching. We cannot get any information from the users Fronter calendar since that calendar is private and not public through the OpenApi.

We do not make it visible in the web page which system the calendar events origin from, but the information is there and is available for anyone using the API. This was a choice we made because we wanted it to be hidden for the end user that the information comes from different systems. In retrospect we have changed our mind. It would be a point to display which system the calendar events origin from. Then the user would know where to make changes if he has access to do so in the external system.

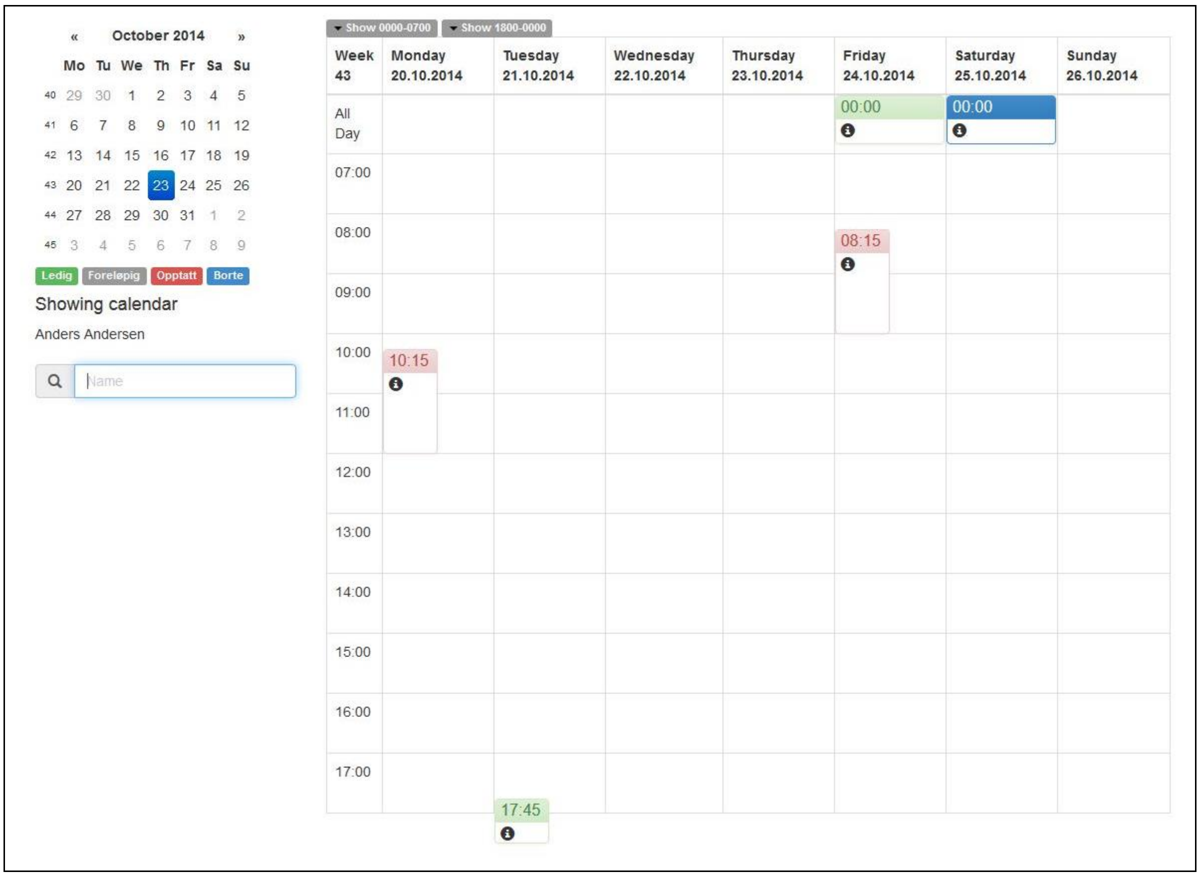


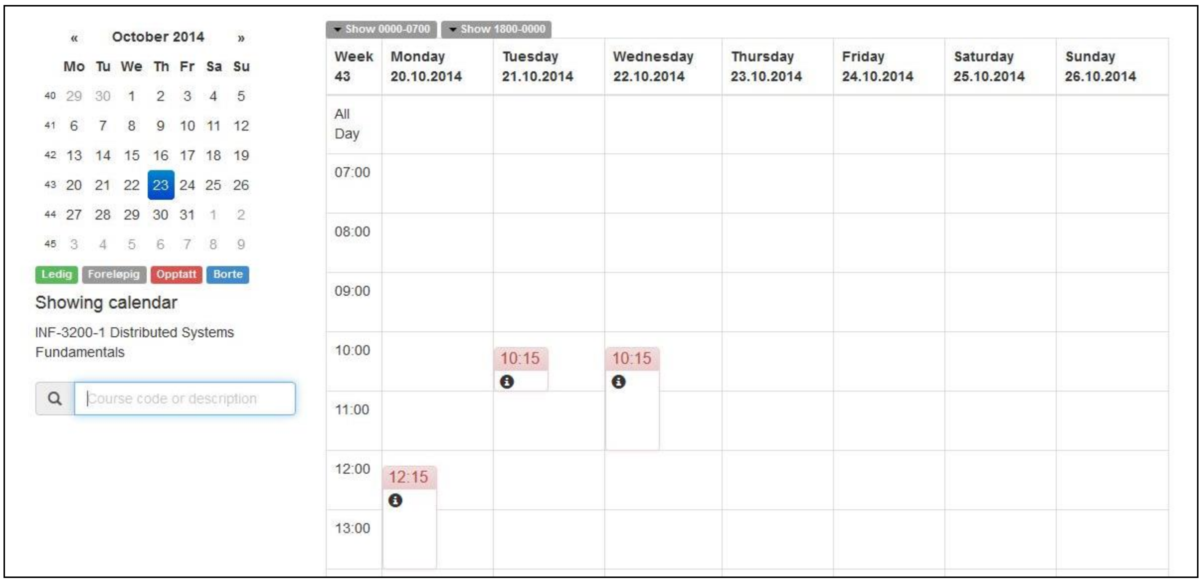
Figure 5-10 Search User Calendar

5.1.5 Search Course and Student Set

The pages to search for courses and student sets are very similar. If the user want to retrieve the timetable for a specific course, he head over to the Search Course function. The user can then start to enter the course code or name of the course. When three letters are hit, the Web Service start to output the search result. The user can then click on a course to display the timetable as shown in Figure 5-11.

Figure 5-12 shows the timetable for a student set after a student set is search for, found and selected. The user can search on the name or code of a student set.

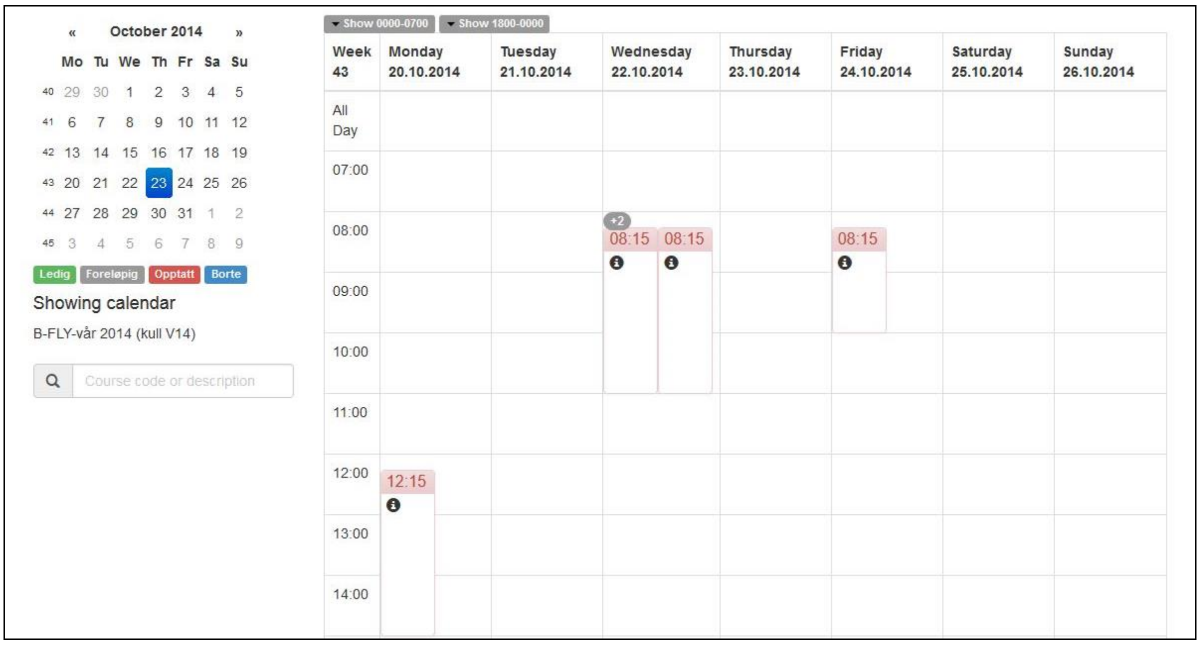
Both of these functions are very similar to the functionality found at the university's timetable portal. We only show them here because these functions are a byproduct of the My Calendar function.



Showing calendar

INF-3200-1 Distributed Systems Fundamentals

Figure 5-11 Search Course



Showing calendar

B-FLY-vår 2014 (kull V14)

Figure 5-12 Search Student Set

5.1.6 Rooms

The search course and search student set functions did not exactly bring anything new to the table, but where more a byproduct of the functionality provided by the Web Service. This is not the case with the room functions. Since rooms exist in both Exchange and Syllabus, we need to merge the results to give the user an impression that he is dealing with only one system.

Figure 5-13 shows the different room functions. There is possible to view all rooms and room lists, or search for them if the room or room list name is known.

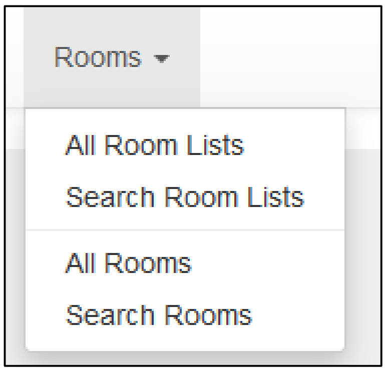


Figure 5-13 Rooms menu

5.1.6.1 All Room Lists

The first menu under the Rooms tab lets the user browse through all the room lists in Exchange and Syllabus. The university has grouped the rooms by physical locations. For example is each building one room list. Almost all physical locations has both meeting rooms and classrooms. All the meeting rooms are in Exchange, and all the classrooms are in Syllabus. Both Exchange and Syllabus has room lists, and the name of these room lists can be exactly the same or very similar.

In Figure 5-14 we have clicked on the room list *Breivika*. This room list actually exist in both Exchange and Syllabus, but the Web Service combines these room lists into one new room list with the same name. When we click on the room lists all the rooms in the new room list are shown. The user does not known that the calendar of the rooms displayed can either be in Exchange or Syllabus. But the user don't need to know this information[19], the user only needs to know that all the rooms at the location *Breivika* is displayed without visiting more than one system.

We click on the room *BRELIA L-133 Ark* and the calendar from Syllabus is shown. If we would to click on the next room in the list, *BRELIA L121 Møterom*, the calendar from Exchange would be displayed.

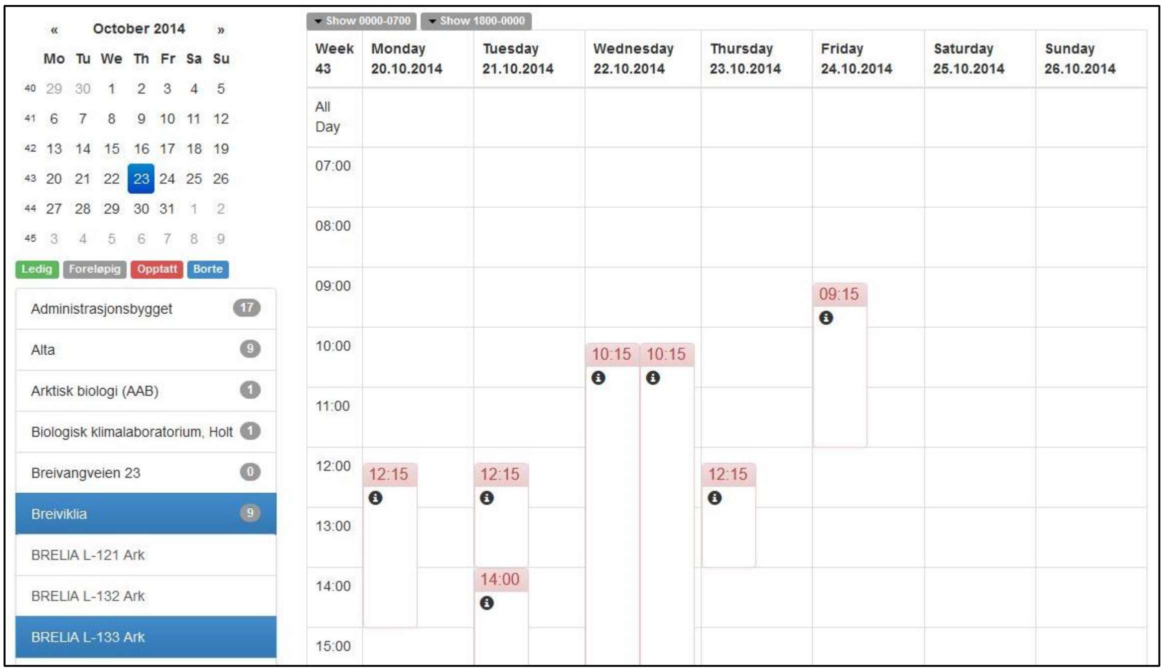
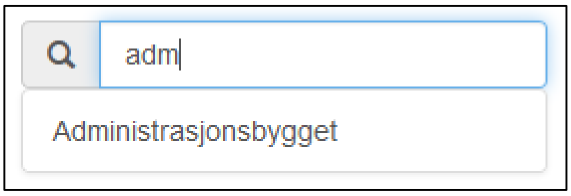


Figure 5-14 Room lists and rooms

[19] In retrospect, maybe the user need this information if the user would like to edit the event.

5.1.6.2 Search Room Lists

The room list *Administrasjonsbygget* also exist in both Exchange and Syllabus. If we would to search for this room list we only need to enter a partial string as shown in Figure 5-15. As we can see we only get one result. The Web Service does a search against Exchange and Syllabus for *adm*, and both systems return one result. Then the Web Service compare[20] the results, and if the room list names are similar the Web Service merge the results into one new list.


adm

Administrasjonsbygget

Figure 5-15 Room List Search (Combined)

5.1.6.3 All Rooms

If the user don't know the name of the room he is looking for, and don't know which room list the room lies within, it is possible to browse through all the rooms in Exchange and Syllabus. Figure 5-16 shows this function.

The list of all the rooms are mixed, some rooms are from Exchange and some rooms are from Syllabus. The complete list from both the systems are sorted by name.

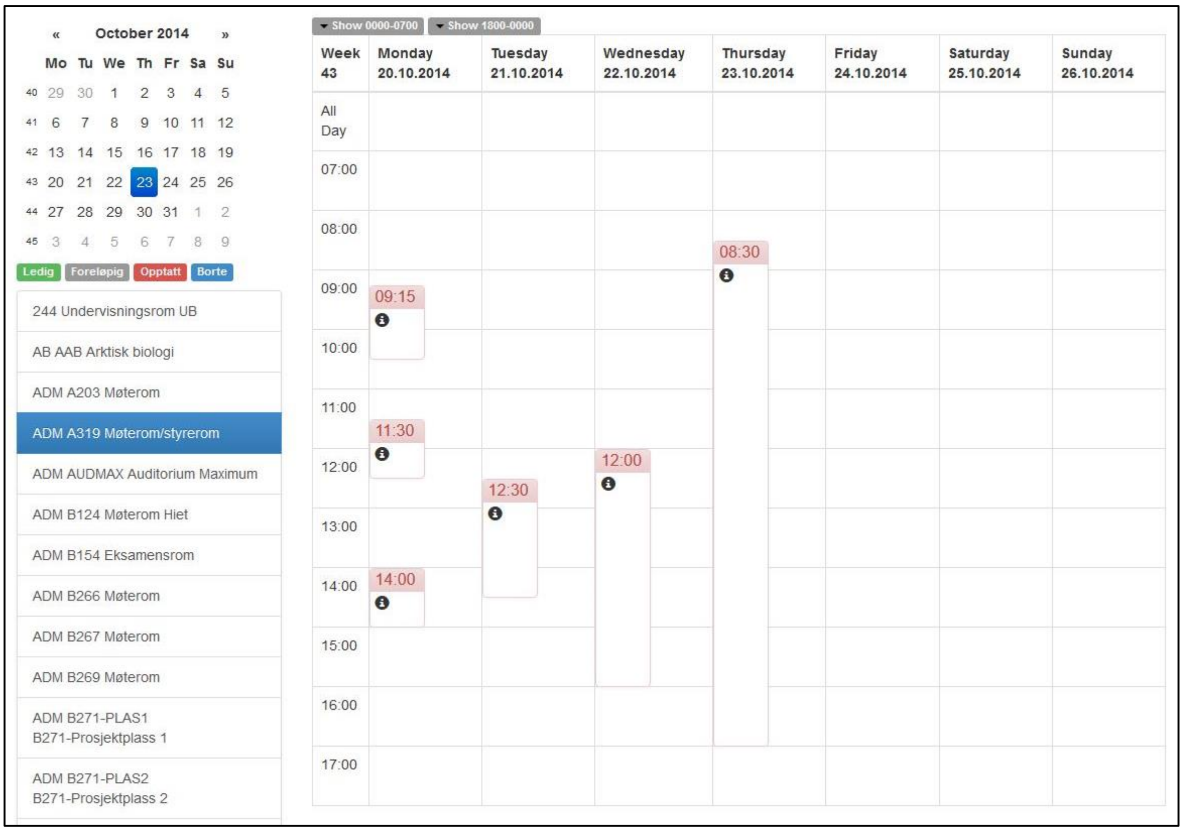


Figure 5-16 All rooms

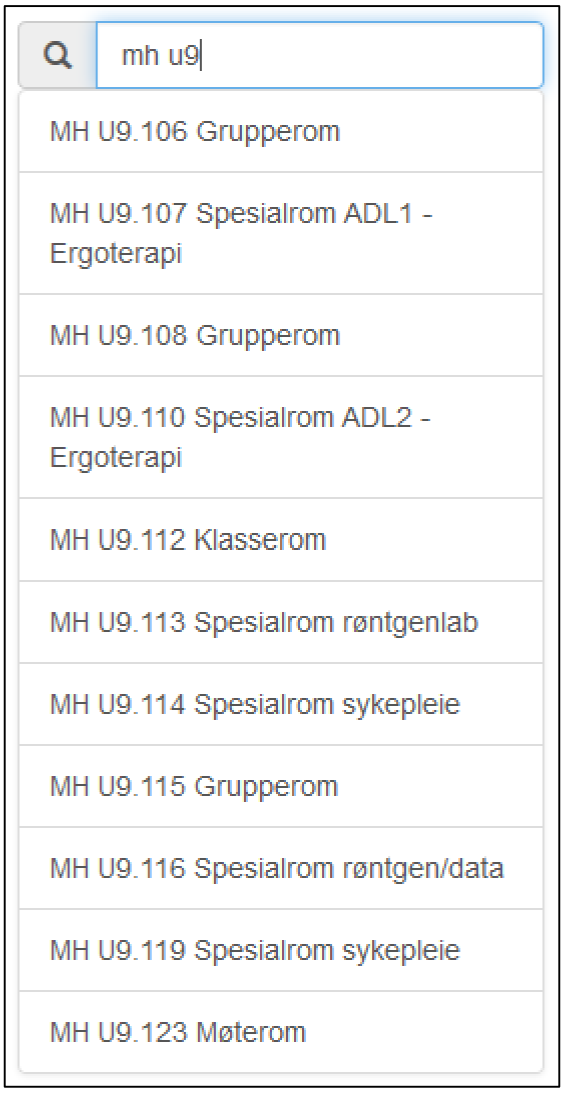
[20] We use PHPs similar_text function to compare strings. If the similarity is more than 95% we assume that the room list from both systems are the same (The PHP Group, similar_text, 2014).

5.1.6.4 Search Rooms

If the user know the name of the room he is looking for, it is possible to search for the room directly. By entering a partial search string as shown in Figure 5-17, all the rooms that matches the string are shown.

This search result of rooms are also a merged result from Exchange and Syllabus. The user don’t need to know which system the room calendar origins from, but the information is available to anyone that uses the API.

As the system developer, we can easily see that any room with a name containing *Møterom* origins from Exchange, and any room name containing *Grupperom* origins from Syllabus.


mh u9

MH U9.106 Grupperom

MH U9.107 Spesialrom ADL1 - Ergoterapi

MH U9.108 Grupperom

MH U9.110 Spesialrom ADL2 - Ergoterapi

MH U9.112 Klasserom

MH U9.113 Spesialrom røntgenlab

MH U9.114 Spesialrom sykepleie

MH U9.115 Grupperom

MH U9.116 Spesialrom røntgen/data

MH U9.119 Spesialrom sykepleie

MH U9.123 Møterom

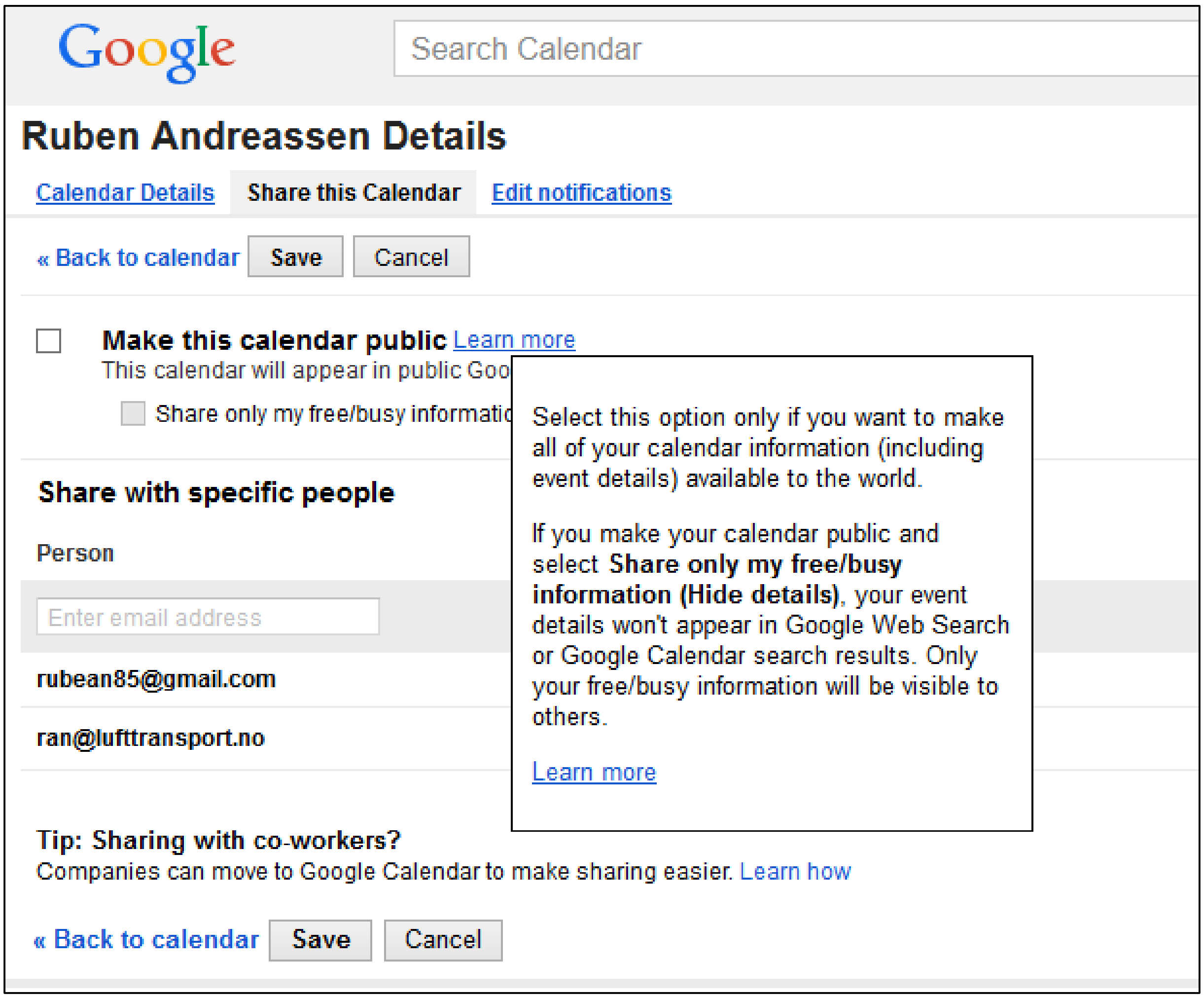
Figure 5-17 Room Search (Combined)

5.2 ICAL

The ICAL menu in the GUI explains how the Web Service creates a shared calendar. The iCalendar file is created when requested. In this thesis we have limited the file to only contain calendar events from one week before the current date, and four weeks from the current date. Since we only want to demonstrate this functionality we don’t see the need to provide more information than that. Further work could for example add the possibility to specify a date span in the URL, or return all known events.

We have decided to only provide calendar information from courses, since this information is already public through the university’s online timetable portal. Most online calendar systems, like Google Calendar, don’t support authentication when adding a calendar by URL. This means that there would not be a good idea to have private calendar information available through this function. If one would need to share ones calendar through the ICAL function, only busy/free status should be shared and not details about the events. If you have a Google Calendar and want to share it, Google makes it pretty clear that you should “Select this option only if you want to make all of your calendar information (including event details) available to the world” as seen in Figure 5-18.

When we started this thesis, this functionality was new and useful for students at the university. Only the people in our circle of friends knew about this, and during the development process there where students that used this functionality to get the timetable of their courses on their mobile phones. But during this thesis the university’s online timetable got this same functionality. Now that the university’s official timetable have this in place, our implementation is obsolete and probably need no further development if anyone where to take the project further.



Search Calendar

Ruben Andreassen Details

Calendar Details | Share this Calendar | Edit notifications

« Back to calendar | Save | Cancel

☐ **Make this calendar public** Learn more

This calendar will appear in public Goo

☐ Share only my free/busy informatio

Select this option only if you want to make all of your calendar information (including event details) available to the world.

If you make your calendar public and select **Share only my free/busy information (Hide details)**, your event details won't appear in Google Web Search or Google Calendar search results. Only your free/busy information will be visible to others.

Learn more

Share with specific people

Person

Enter email address

rubean85@gmail.com

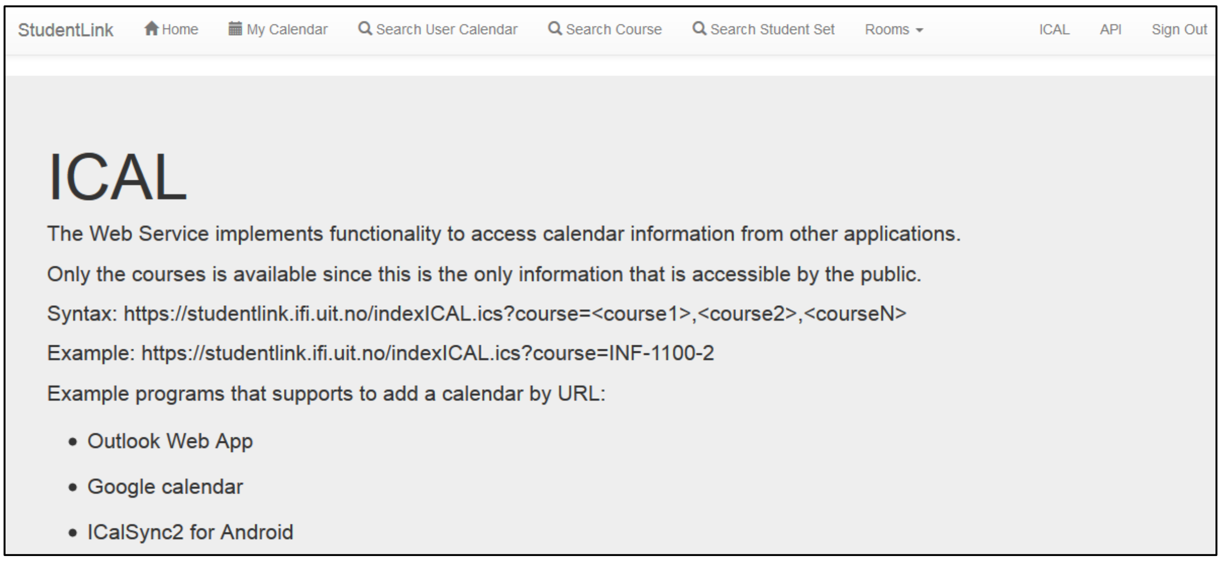
ran@lufttransport.no

Tip: Sharing with co-workers?

Companies can move to Google Calendar to make sharing easier. Learn how

« Back to calendar | Save | Cancel

Figure 5-18 Google Calendar public calendar warning



ICAL

The Web Service implements functionality to access calendar information from other applications.

Only the courses is available since this is the only information that is accessible by the public.

Syntax: https://studentlink.ifi.uit.no/indexICAL.ics?course=<course1>,<course2>,<courseN>

Example: https://studentlink.ifi.uit.no/indexICAL.ics?course=INF-1100-2

Example programs that supports to add a calendar by URL:

- Outlook Web App
- Google calendar
- ICalSync2 for Android

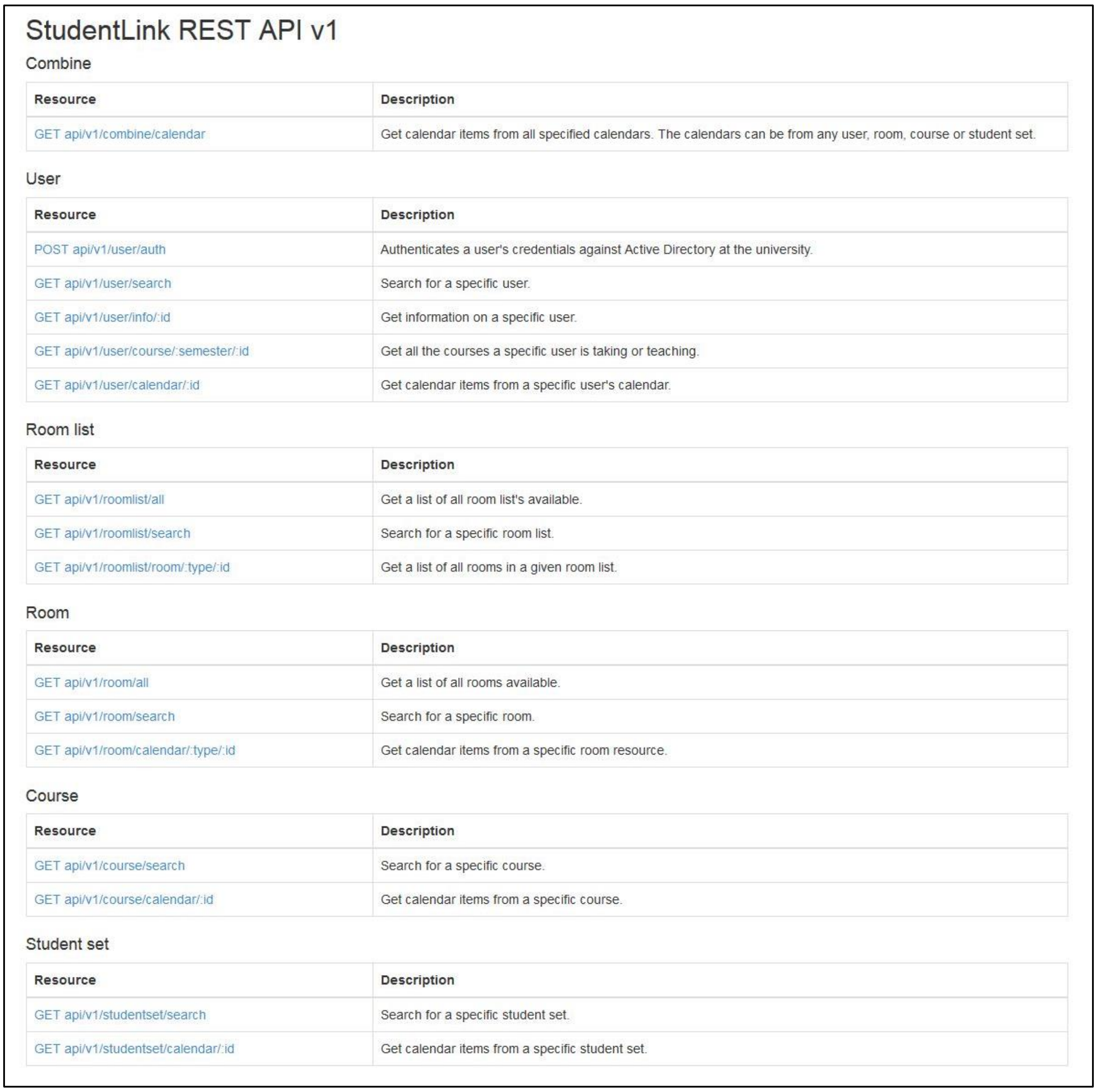
Figure 5-19 Information about the ICAL service

5.3 API

As we have mentioned before, the GUI is only to test the functionality that the Web Service provides. The idea is that other 3rd party systems can use the API to make customized applications on other devices than the web, like mobile phones.

The API requires authentication. We have implemented HTTP Basic Authentication, and anyone registered in the university's AD has access to use the API. There are other authentication protocols out there, like OAuth, but these protocols are more complex to implement and not the main focus in this thesis.

We have documented all the functions in the API as shown in Figure 5-20. On the first page we can see all the resources available in the API and a short description. A full description of the API can be viewed in Appendix C.



StudentLink REST API v1

Combine

Resource	Description
GET api/v1/combine/calendar	Get calendar items from all specified calendars. The calendars can be from any user, room, course or student set.

User

Resource	Description
POST api/v1/user/auth	Authenticates a user's credentials against Active Directory at the university.
GET api/v1/user/search	Search for a specific user.
GET api/v1/user/info/:id	Get information on a specific user.
GET api/v1/user/course/:semester/:id	Get all the courses a specific user is taking or teaching.
GET api/v1/user/calendar/:id	Get calendar items from a specific user's calendar.

Room list

Resource	Description
GET api/v1/roomlist/all	Get a list of all room list's available.
GET api/v1/roomlist/search	Search for a specific room list.
GET api/v1/roomlist/room/:type/:id	Get a list of all rooms in a given room list.

Room

Resource	Description
GET api/v1/room/all	Get a list of all rooms available.
GET api/v1/room/search	Search for a specific room.
GET api/v1/room/calendar/:type/:id	Get calendar items from a specific room resource.

Course

Resource	Description
GET api/v1/course/search	Search for a specific course.
GET api/v1/course/calendar/:id	Get calendar items from a specific course.

Student set

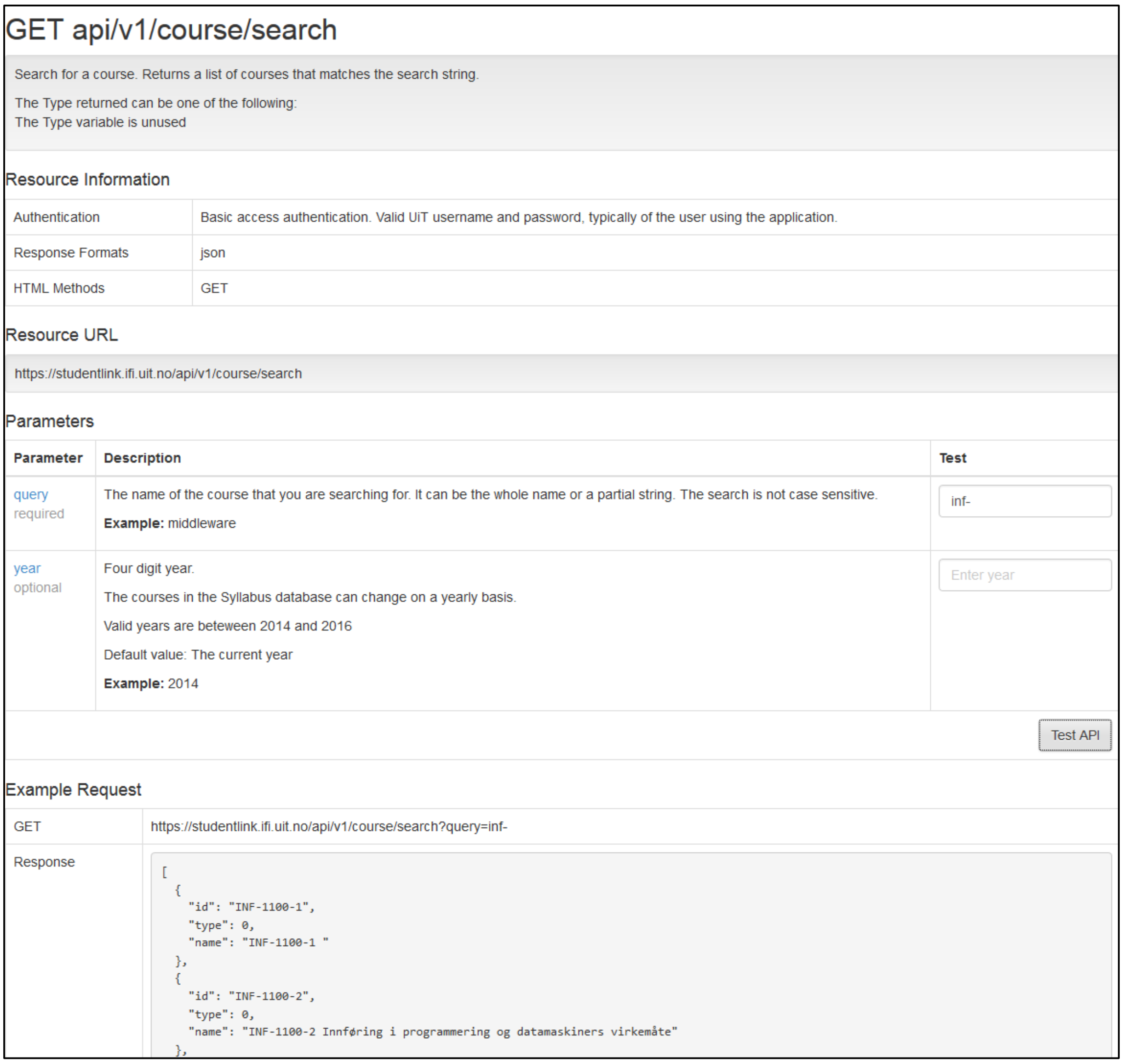
Resource	Description
GET api/v1/studentset/search	Search for a specific student set.
GET api/v1/studentset/calendar/:id	Get calendar items from a specific student set.

Figure 5-20 StudentLink REST API v1

If we click on one of the resources, we get more detailed information as shown in Figure 5-21. The page explains all the details necessary to use the resource.

We feel that it is outside the scope of this thesis to develop an application that uses the API. Instead we include a tool to test the functionality of the API directly in the GUI. In Figure 5-22 we take one of the search results from Figure 5-21 and retrieve the calendar information of a particular course.

This works by filling in the input fields in the *Test* column to the right in the *Parameters* section. When the required input fields are filled, the user can click the *Test API* button. The GUI then sends an AJAX request to the Web Service, using the resource URL. The first time this is done, the browser will ask for the user's credentials since the API requires HTTP Basic Authentication. It does not matter to the API that the user is already authenticated in the GUI. The *Response* field in the *Example Request* section is updated with the response from the API.


GET api/v1/course/search

Search for a course. Returns a list of courses that matches the search string.

The Type returned can be one of the following:
The Type variable is unused

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/course/search

Parameters

Parameter	Description	Test
query required	The name of the course that you are searching for. It can be the whole name or a partial string. The search is not case sensitive. **Example:** middleware	inf-
year optional	Four digit year. The courses in the Syllabus database can change on a yearly basis. Valid years are beteween 2014 and 2016 Default value: The current year **Example:** 2014	Enter year

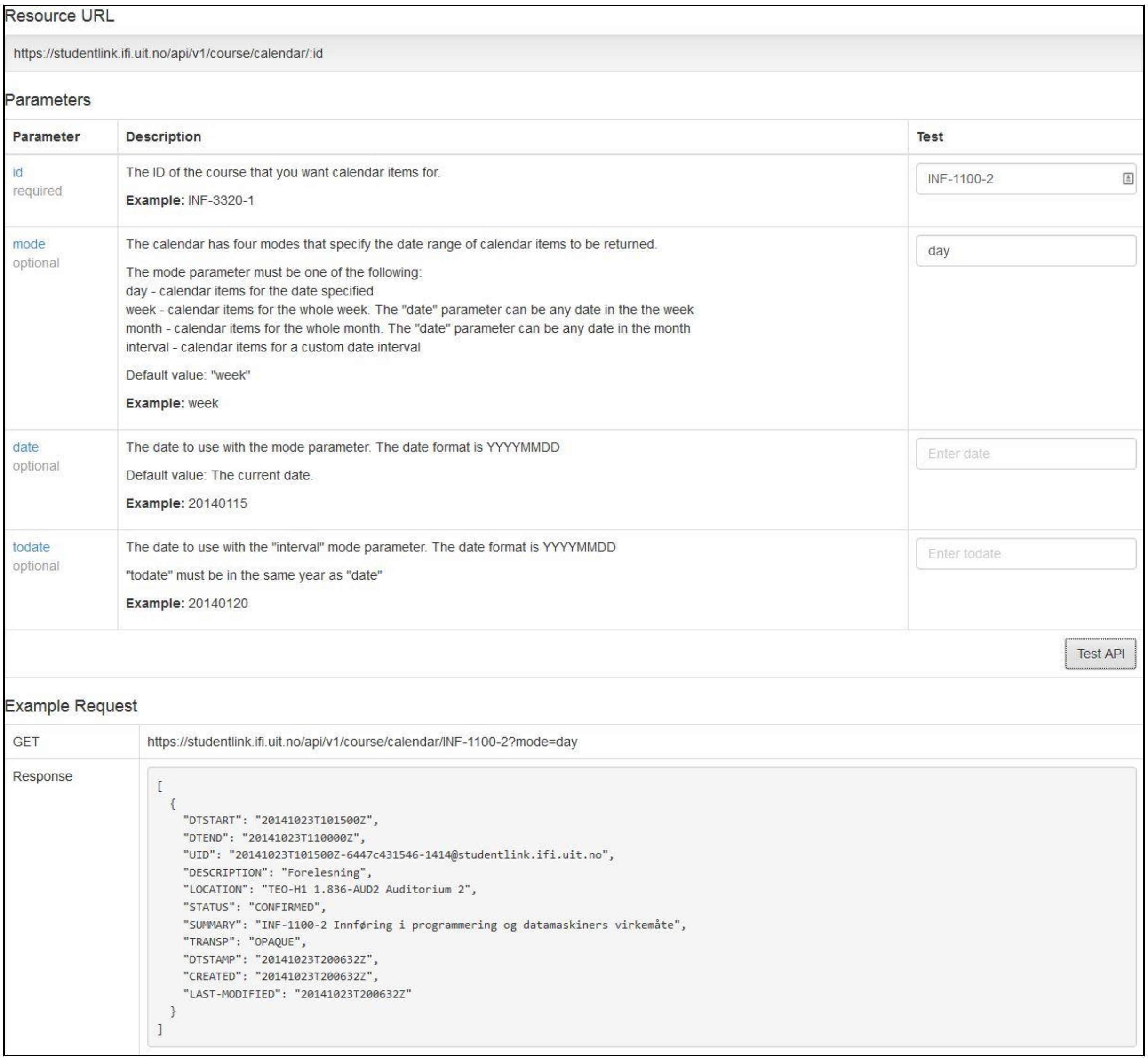
Example Request

GET https://studentlink.ifi.uit.no/api/v1/course/search?query=inf-

Response

```
[
  {
    "id": "INF-1100-1",
    "type": 0,
    "name": "INF-1100-1 "
  },
  {
    "id": "INF-1100-2",
    "type": 0,
    "name": "INF-1100-2 Innføring i programmering og datamaskiners virkemåte"
  },
```

Figure 5-21 GET method resource URL and example request



Resource URL

https://studentlink.ifi.uit.no/api/v1/course/calendar/:id

Parameters

Parameter	Description	Test
id required	The ID of the course that you want calendar items for. **Example:** INF-3320-1	INF-1100-2
mode optional	The calendar has four modes that specify the date range of calendar items to be returned. The mode parameter must be one of the following: day - calendar items for the date specified week - calendar items for the whole week. The "date" parameter can be any date in the the week month - calendar items for the whole month. The "date" parameter can be any date in the month interval - calendar items for a custom date interval Default value: "week" **Example:** week	day
date optional	The date to use with the mode parameter. The date format is YYYYMMDD Default value: The current date. **Example:** 20140115	Enter date
todate optional	The date to use with the "interval" mode parameter. The date format is YYYYMMDD "todate" must be in the same year as "date" **Example:** 20140120	Enter todate

Example Request

GET	https://studentlink.ifi.uit.no/api/v1/course/calendar/INF-1100-2?mode=day
Response	

```
[
  {
    "DTSTART": "20141023T101500Z",
    "DTEND": "20141023T110000Z",
    "UID": "20141023T101500Z-6447c431546-1414@studentlink.ifi.uit.no",
    "DESCRIPTION": "Forelesning",
    "LOCATION": "TEO-H1 1.836-AUD2 Auditorium 2",
    "STATUS": "CONFIRMED",
    "SUMMARY": "INF-1100-2 Innføring i programmering og datamaskiners virkemåte",
    "TRANSP": "OPAQUE",
    "DTSTAMP": "20141023T200632Z",
    "CREATED": "20141023T200632Z",
    "LAST-MODIFIED": "20141023T200632Z"
  }
]
```

Figure 5-22 GET method resource URL and example request

5.4 Use cases

In the early stage of the Web Service the best use case where the ICAL service. A few students that we knew used this function to get their course timetable on their mobile phones. During this thesis, the official timetable of the university got this exact same functionality. We must therefore demonstrate a different use case, unique to our solution.

We will look at the following scenario: we are planning a meeting with our student advisor. We want a one hour long meeting to take place between 08:00 and 16:00, on one of the days Monday to Thursday, in week 13, in the year 2015. This approximate time will be our guideline when planning the meeting. To make the task a little more complex, we are going to plan the meeting in a building called *Teknologibygget*. We want to use either a meeting room or a group room if possible.

To illustrate why our Web Service would be the preferred way to plan such a meeting, we are going to show the workflow of the planning with and without our solution.

5.4.1 Planning a meeting in the conventional way

First, we open a browser and head to our personal Google Calendar. The next calendar we want to check is in Fronter. We head over to Fronter in a new browser window, log in and find the calendar. After that we head over to the university's timetable portal. Here we need to search for the courses that we are taking. Finally we head over to the Outlook Web App (OWA) portal to view our Exchange calendar. We now have four browser windows open as illustrated in Figure 5-23.

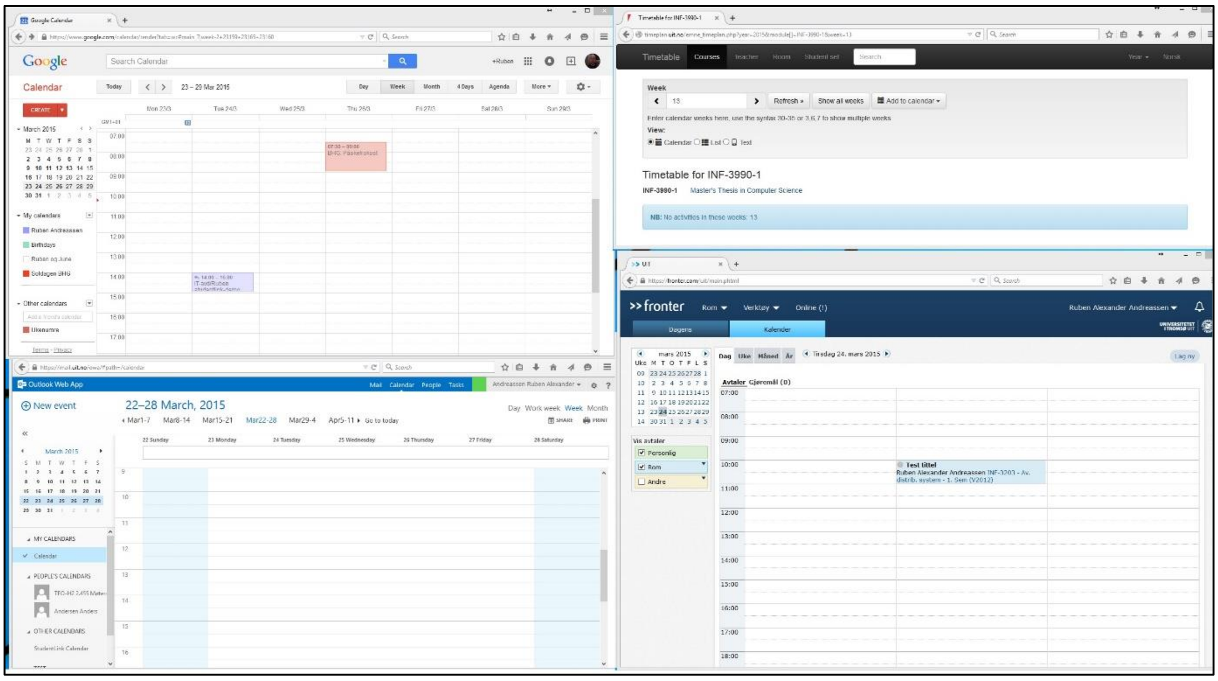


Figure 5-23 A full overview of our personal calendars with information from the following calendar systems: Google Calendar (private), Syllabus (university), Fronter (university) and Microsoft Exchange (university).

We now want to check the student advisors calendar. The student advisor also has an Exchange calendar, and since we are within the same organization we can add him directly in OWA (bottom left in Figure 5-23). This will give us the view shown in Figure 5-24. If we knew which courses our student advisor is teaching, we could add these courses in the university's timetable portal (top right in Figure 5-23), but we don't have that information at hand.

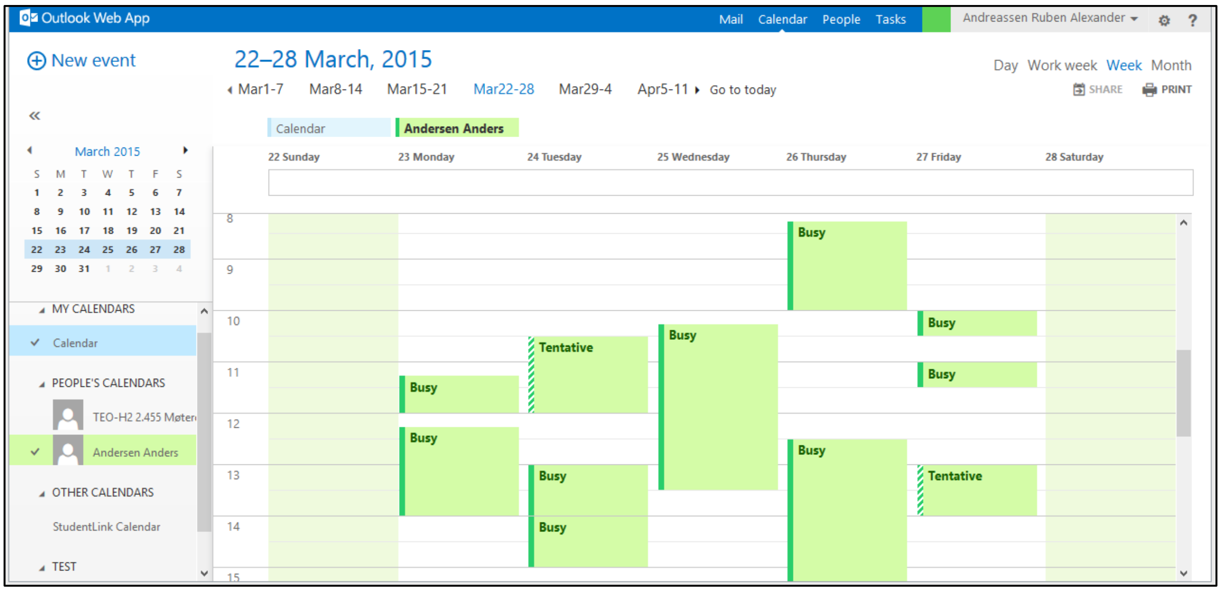


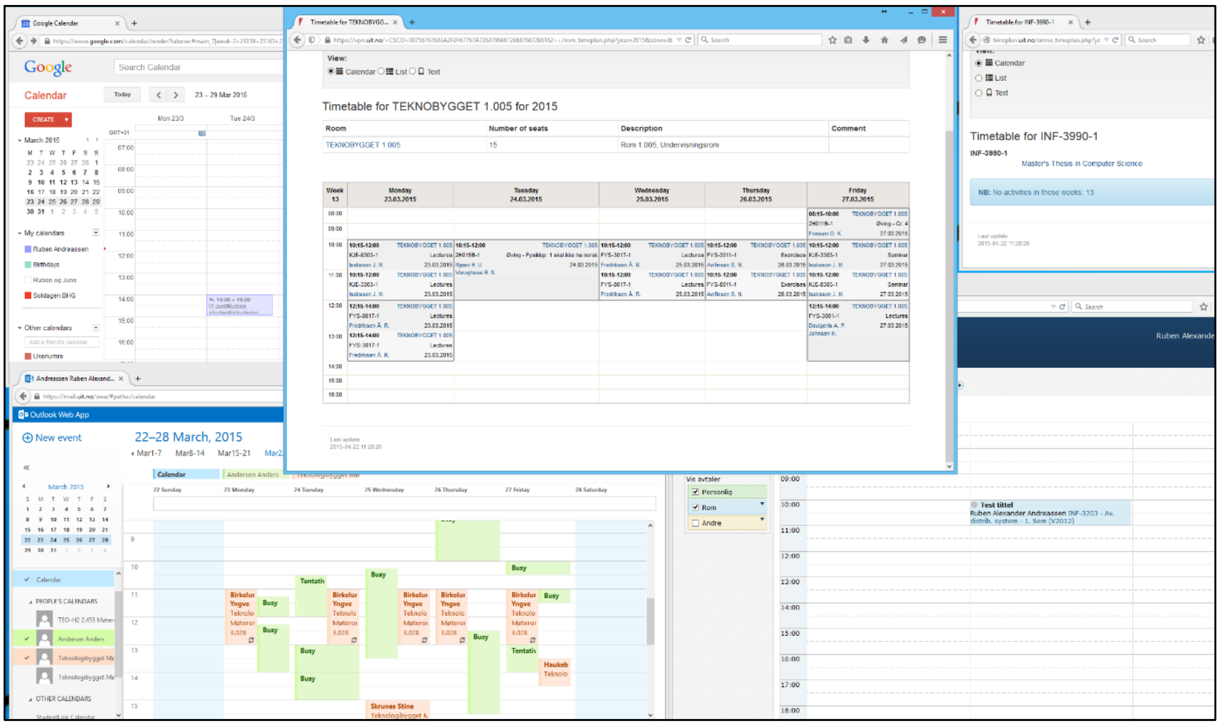
Figure 5-24 Two calendars showing in Outlook Web App

The next step is to find a meeting room or a group room. The meeting rooms at the university is registered in Exchange, so we can search and add them directly in OWA in the same way as we found the student advisor. The group rooms are in Syllabus, so we must use the university's timetable portal to find the calendar of the group room.

When we try to access the room schedule, we get an error message stating that we need to be within the university's network to access the room schedule. Since we are not at the university when planning this meeting, we need to access the university's VPN portal as an extra step on the way.

After we find the schedule for the room we have a complete overview of all calendars, in five browser windows, as illustrated in Figure 5-25. The only calendar we are missing is the courses that the student advisor is teaching, and any private calendars that he might have.

We are now ready to plan a meeting with our student advisor. In this particular example we can discard the calendars with the fewest events so that the task becomes a little bit easier, but we still had to go through a lot of steps to find that out.


View:
(● Calendar) ○ List ○ Text

Timetable for TEKNOBYGGET 1.005 for 2015

Room	Number of seats	Description	Comment
TEKNOBYGGET 1.005	15	Rom 1.005, Undervisningsrom	

Week 13	Monday 23.03.2015	Tuesday 24.03.2015	Wednesday 25.03.2015	Thursday 26.03.2015	Friday 27.03.2015
08:00					08:15-10:00 TEKNOBYGGET 1.005 2H011B-1 Øving - Gr. 4 Fossum O. K. 27.03.2015
09:00					
10:00	10:15-12:00 TEKNOBYGGET 1.005 KJE-8303-1 Lectures Isaksson J. M. 23.03.2015	10:15-12:00 TEKNOBYGGET 1.005 2H015B-1 Øving - Fysikkgr. 1 skal ikke ha norsk Njøen H. U. Varughese B. S. 24.03.2015	10:15-12:00 TEKNOBYGGET 1.005 FYS-3017-1 Lectures Fredriksen Å. R. 25.03.2015	10:15-12:00 TEKNOBYGGET 1.005 FYS-3011-1 Exercises Anfinsen S. N. 26.03.2015	10:15-12:00 TEKNOBYGGET 1.005 KJE-3303-1 Seminar Isaksson J. M. 27.03.2015
11:00	10:15-12:00 TEKNOBYGGET 1.005 KJE-3303-1 Lectures Isaksson J. M. 23.03.2015		10:15-12:00 TEKNOBYGGET 1.005 FYS-8017-1 Lectures Fredriksen Å. R. 25.03.2015	10:15-12:00 TEKNOBYGGET 1.005 FYS-8011-1 Exercises Anfinsen S. N. 26.03.2015	10:15-12:00 TEKNOBYGGET 1.005 KJE-8303-1 Seminar Isaksson J. M. 27.03.2015
12:00	12:15-14:00 TEKNOBYGGET 1.005 FYS-8017-1 Lectures Fredriksen Å. R. 23.03.2015				12:15-14:00 TEKNOBYGGET 1.005 FYS-3001-1 Lectures Doulgeris A. P. Johnsen H. 27.03.2015
13:00	12:15-14:00 TEKNOBYGGET 1.005 FYS-3017-1 Lectures Fredriksen Å. R. 23.03.2015				
14:00					
15:00					
16:00					

Last update
2015-04-22 11:28:26

Timetable for INF-3990-1
INF-3990-1
Master's Thesis in Computer Science
NB: No activities in these weeks: 13

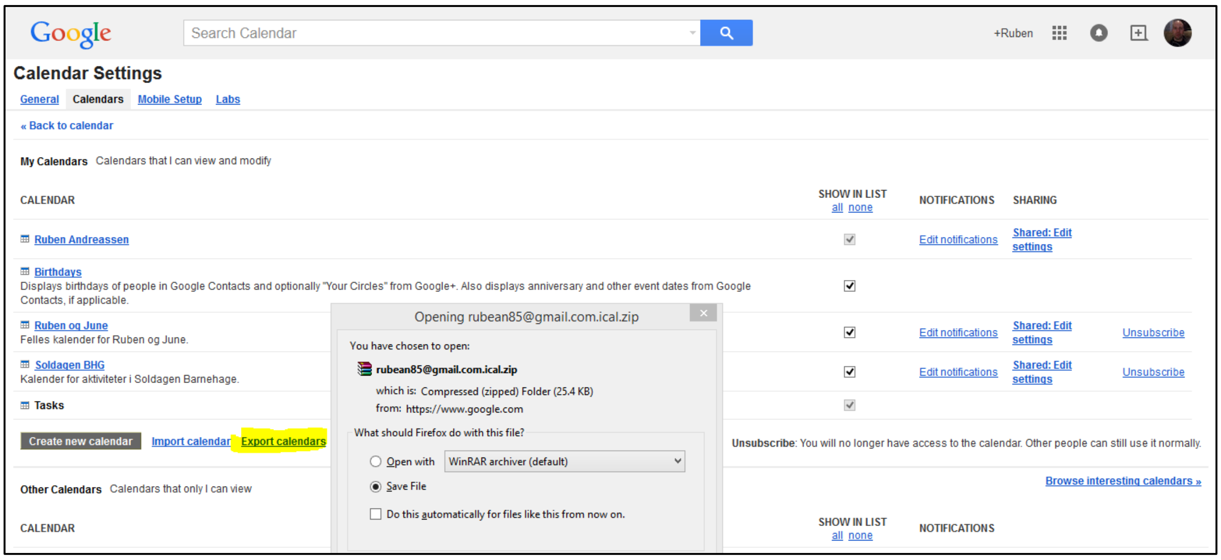
Test tittel
Ruben Alexander Andreassen INF-3203 - Av. distrib. system - 1. Sem (V2012)

Figure 5-25 A full overview of all the calendars needed to plan a meeting as accurately as possible on the first attempt

Side note! 23. February 2015 the university launched a new function that imports courses from Syllabus to the student and teacher calendars in Exchange (Orakelet, TimeIT - Undervisningsplan på telefon og kalender, 2015). This will eliminate the step to use the university's timetable portal to check our courses calendar (upper right window in Figure 5-25), and the student advisor's calendar in Exchange will contain the courses that the student advisor is teaching. Before the 23. February 2015 we did not know which courses the student advisor is teaching as pointed out earlier. We still need to use the university's timetable portal to check the calendars of the rooms that are in Syllabus.

5.4.2 Planning a meeting using the Web Service

First, we head over to the Google Calendar and export the private calendars that we have as shown in Figure 5-26. Then we unzip the downloaded file as shown in Figure 5-27, and upload the iCalendar files to the Web Service using the *Add* function in My Calendar in the GUI as shown in Figure 5-28.



Google Search Calendar +Ruben

Calendar Settings

General Calendars Mobile Setup Labs

« Back to calendar

My Calendars Calendars that I can view and modify

CALENDAR	SHOW IN LIST all none	NOTIFICATIONS	SHARING	
Ruben Andreassen	✓	Edit notifications	Shared: Edit settings	
Birthdays Displays birthdays of people in Google Contacts and optionally "Your Circles" from Google+. Also displays anniversary and other event dates from Google Contacts, if applicable.	✓			
Ruben og June Felles kalender for Ruben og June.	✓	Edit notifications	Shared: Edit settings	Unsubscribe
Soldagen BHG Kalender for aktiviteter i Soldagen Barnehage.	✓	Edit notifications	Shared: Edit settings	Unsubscribe
Tasks	✓			

Create new calendar Import calendar Export calendars

Unsubscribe: You will no longer have access to the calendar. Other people can still use it normally.

Other Calendars Calendars that only I can view

Browse interesting calendars »

CALENDAR SHOW IN LIST all none NOTIFICATIONS

Opening rubean85@gmail.com.ical.zip

You have chosen to open:

rubean85@gmail.com.ical.zip

which is: Compressed (zipped) Folder (25.4 KB)

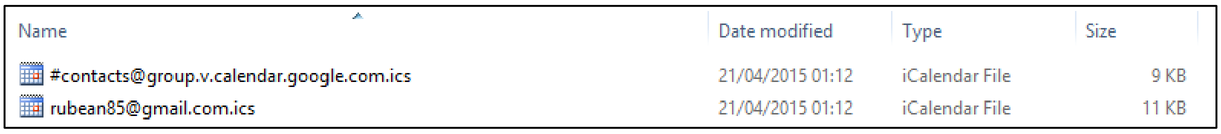
from: https://www.google.com

What should Firefox do with this file?

Open with WinRAR archiver (default) [✓ Save File]

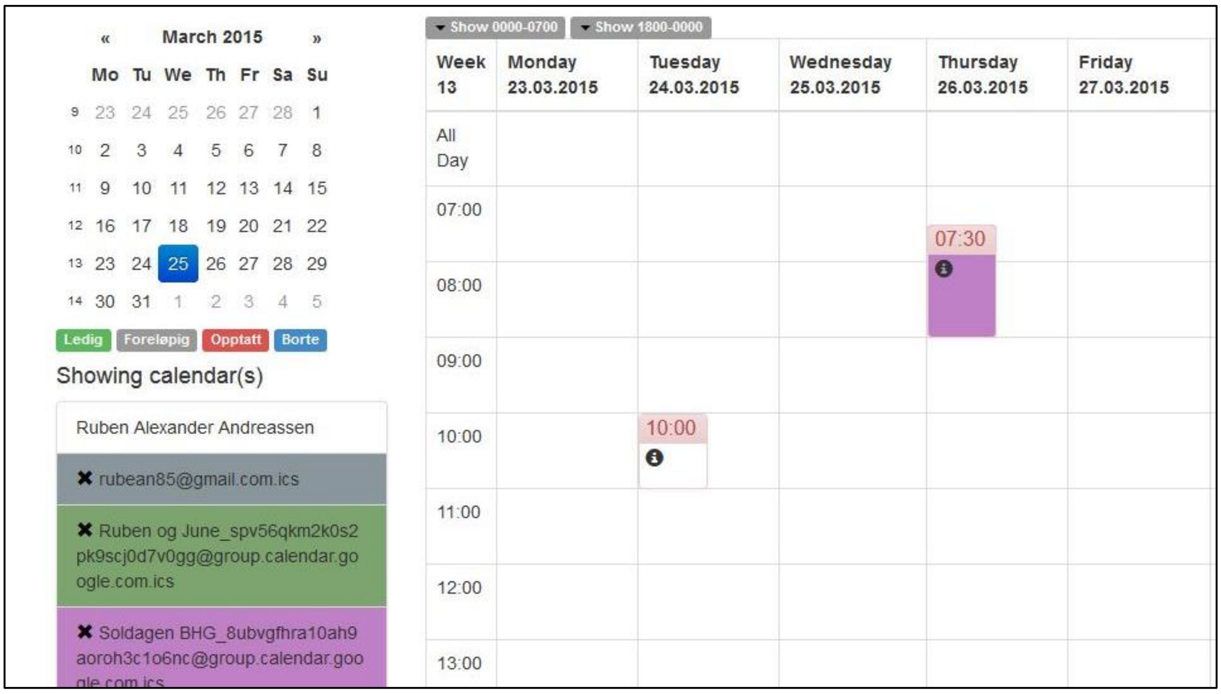
Do this automatically for files like this from now on.

Figure 5-26 Using the «Export calendars» function in Google Calendar lets us download a ZIP file with one iCalendar file for each calendar that we have



Name	Date modified	Type	Size
#contacts@group.v.calendar.google.com.ics	21/04/2015 01:12	iCalendar File	9 KB
rubean85@gmail.com.ics	21/04/2015 01:12	iCalendar File	11 KB

Figure 5-27 When we unzip the downloaded file we find four iCalendar files, three of them are calendars and one contains contact information



Showing calendar(s)

Ruben Alexander Andreassen

✖ rubean85@gmail.com.ics

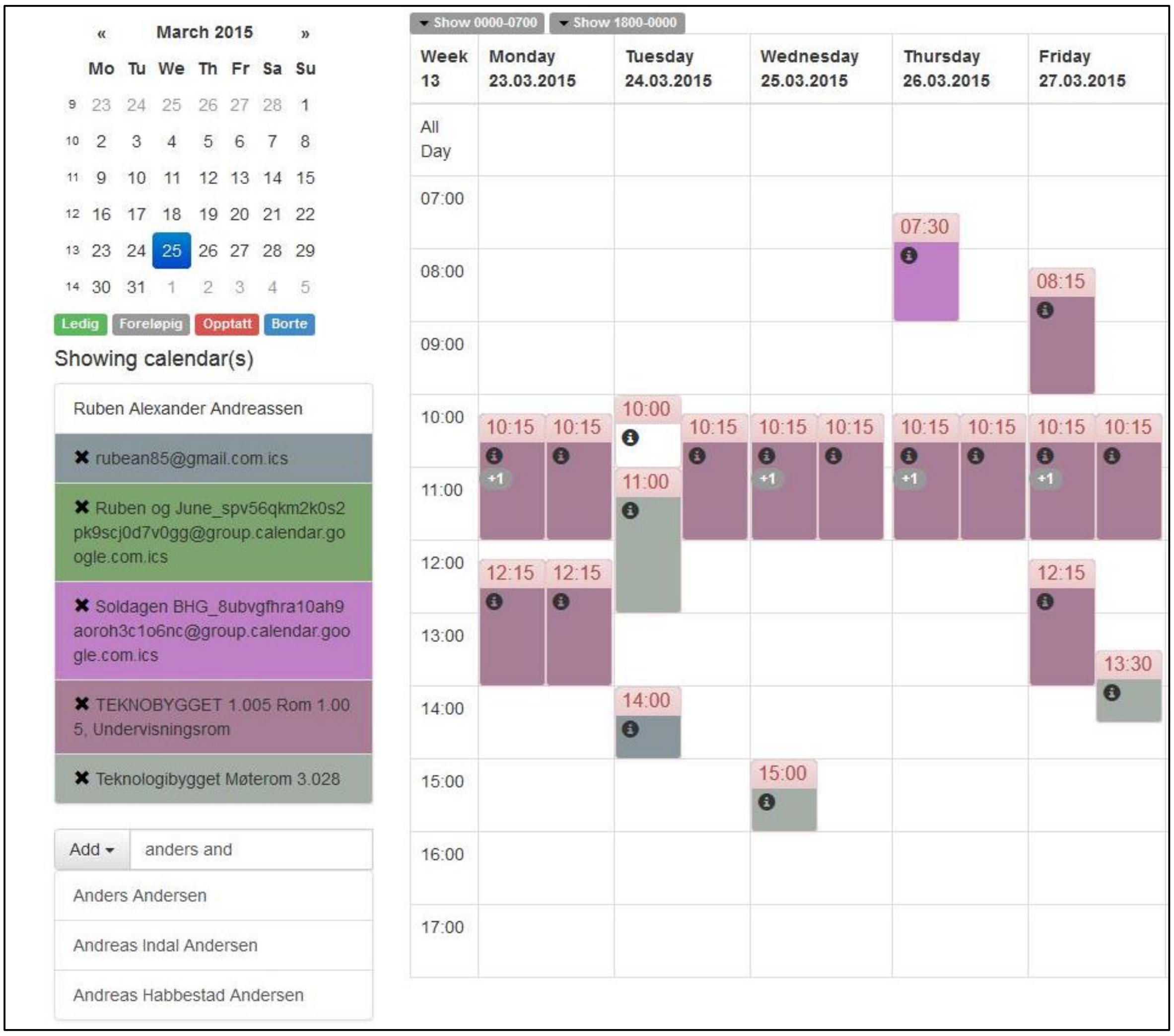
✖ Ruben og June_spv56qkm2k0s2pk9scj0d7v0gg@group.calendar.google.com.ics

✖ Soldagen BHG_8ubvgfhra10ah9aoroh3c1o6nc@group.calendar.google.com.ics

Figure 5-28 My Calendar shows a full overview of our personal calendars with information from the following calendar systems: Google Calendar (private), Syllabus (university), Fronter (university) and Microsoft Exchange (university).

Now we have a complete overview of what we have planned in week 13 of 2015. The next thing we need to do is to add our student advisor and the rooms that we want. We simply use the *Add* function to search for the student advisor and the relevant rooms and add them to the calendar as shown in Figure 5-29. We now have a total of 38 calendar events from 10 calendars from four different calendar systems, in one view.

We are now ready to plan the meeting. Since we have decided that it is further work to have the Web Service suggest dates and times where all calendars are free, we have done this manually as shown in Figure 5-29 and Table 5-1. We still consider this much easier than the alternative described when planning a meeting the conventional way.



« March 2015 »

	Mo	Tu	We	Th	Fr	Sa	Su
9	23	24	25	26	27	28	1
10	2	3	4	5	6	7	8
11	9	10	11	12	13	14	15
12	16	17	18	19	20	21	22
13	23	24	25	26	27	28	29
14	30	31	1	2	3	4	5

Ledig Foreløpig Opptatt Borte

Showing calendar(s)

Ruben Alexander Andreassen

✖ rubean85@gmail.com.ics

✖ Ruben og June_spv56qkm2k0s2pk9scj0d7v0gg@group.calendar.google.com.ics

✖ Soldagen BHG_8ubvgfhra10ah9aoroh3c1o6nc@group.calendar.google.com.ics

✖ TEKNOBYGGET 1.005 Rom 1.005, Undervisningsrom

✖ Teknologibygget Møterom 3.028

Add ▾ anders and

Anders Andersen

Andreas Indal Andersen

Andreas Habbestad Andersen

▾ Show 0000-0700 ▾ Show 1800-0000

Week 13	Monday 23.03.2015	Tuesday 24.03.2015	Wednesday 25.03.2015	Thursday 26.03.2015	Friday 27.03.2015
All Day					
07:00				07:30	
08:00					08:15
09:00					
10:00	10:15 +1, 10:15	10:00, 10:15	10:15 +1, 10:15	10:15 +1, 10:15	10:15 +1, 10:15
11:00		11:00			
12:00	12:15, 12:15				12:15
13:00					13:30
14:00		14:00			
15:00			15:00		
16:00					
17:00					

Figure 5-29 When we add our student advisor and the rooms we have a full overview of our personal calendars with information from the following calendar systems: Google Calendar (private), Syllabus (university), Fronter (university) and Microsoft Exchange (university), the student advisors calendars from the following calendar systems: Syllabus (university) and Microsoft Exchange (university), and the room calendars from the following calendar systems: Syllabus (university) and Microsoft Exchange (university).

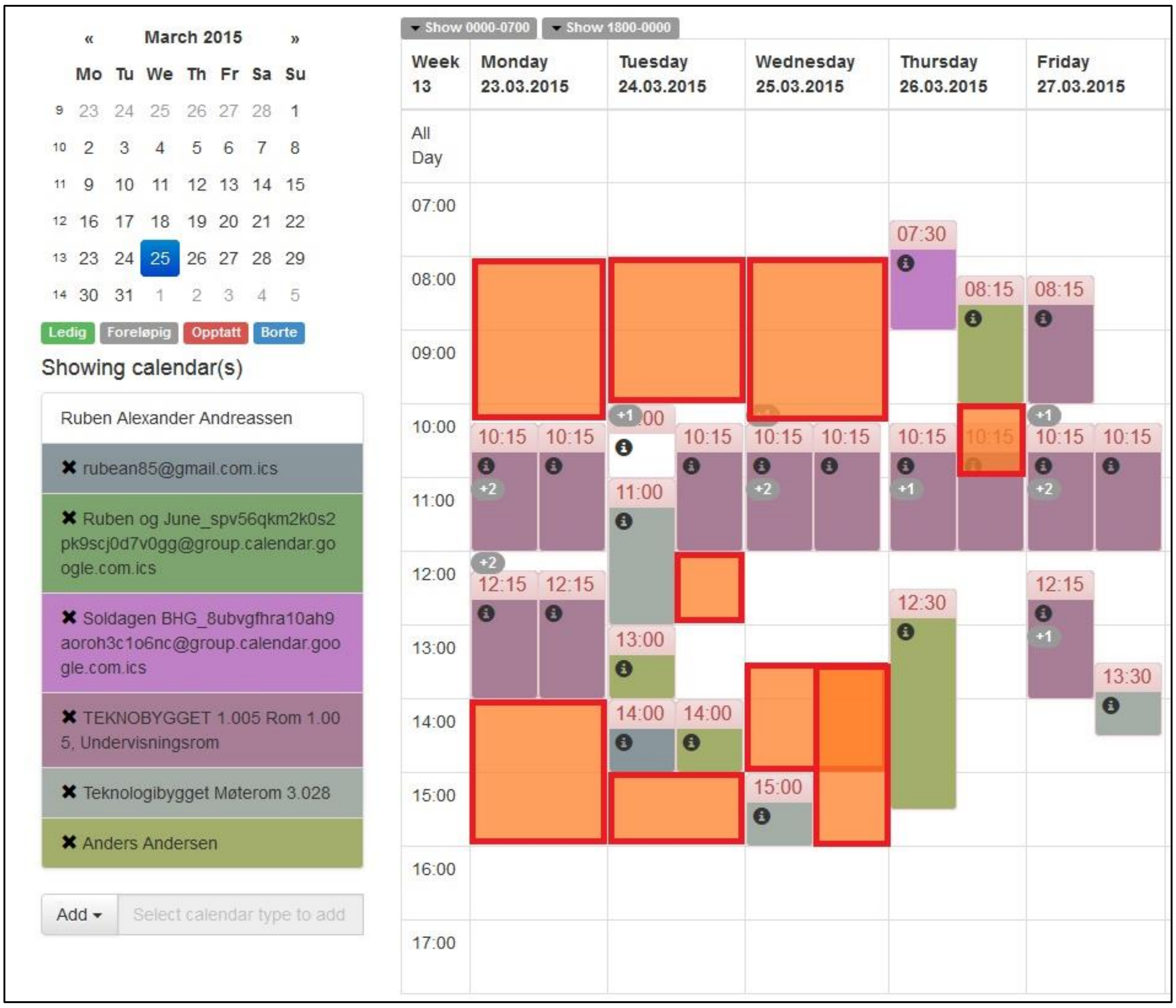

« March 2015 »
Mo Tu We Th Fr Sa Su
Ledig
Foreløpig
Opptatt
Borte
Showing calendar(s)
Ruben Alexander Andreassen
rubean85@gmail.com.ics
Ruben og June_spv56qkm2k0s2pk9scj0d7v0gg@group.calendar.google.com.ics
Soldagen BHG_8ubvgfhra10ah9aoroh3c1o6nc@group.calendar.google.com.ics
TEKNOBYGGET 1.005 Rom 1.005, Undervisningsrom
Teknologibygget Møterom 3.028
Anders Andersen
Add
Select calendar type to add
Show 0000-0700
Show 1800-0000
Week 13
Monday 23.03.2015
Tuesday 24.03.2015
Wednesday 25.03.2015
Thursday 26.03.2015
Friday 27.03.2015
All Day

Figure 5-30 Free periods that are longer than one our between Monday and Thursday are marked manually

Day	**Time**	**Description**
Monday	08:00 to 10:15	No planned events in any of the calendars.
Monday	14:00 to 16:00	No planned events in any of the calendars.
Tuesday	08:00 to 10:00	No planned events in any of the calendars.
Tuesday	12:00 to 13:00	The room 1.005 is free.
Tuesday	15:00 to 16:00	No planned events in any of the calendars.
Wednesday	08:00 to 10:15	No planned events in any of the calendars.
Wednesday	13:30 to 15:00	No planned events in any of the calendars.
Wednesday	13:30 to 16:00	The room 1.005 is free.
Thursday	10:00 to 11:00	The room 3.028 is free. The +X icons indicate that there are simultaneous events that are not shown in the view since there is a limitation on only two simultaneous events at the same date and time. If we click on the +1 on Thursday we get a full overview of that day as shown in Figure 5-31.

Table 5-1 A detailed list of the free periods with description

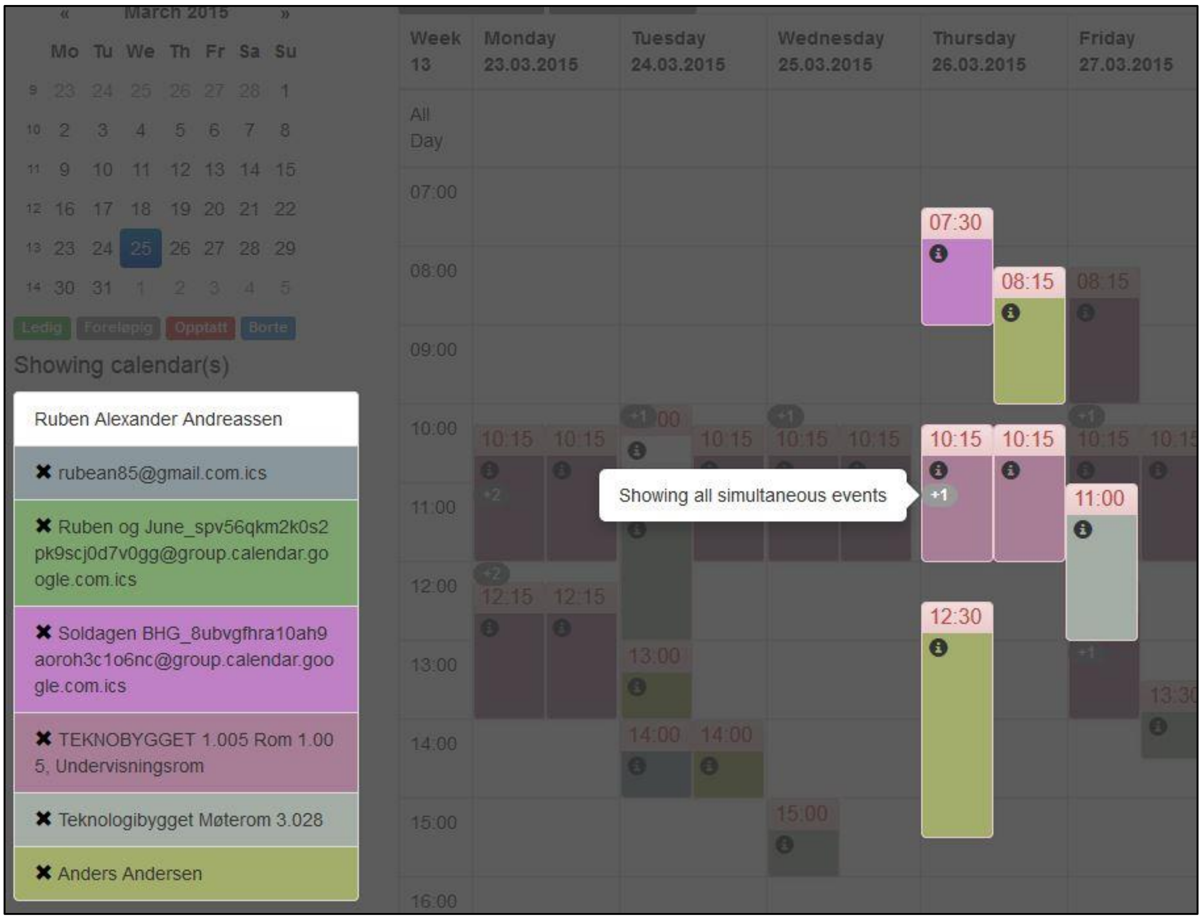

March 2015
Mo Tu We Th Fr Sa Su
Ledig
Foreløpig
Opptatt
Borte
Showing calendar(s)
Ruben Alexander Andreassen
rubean85@gmail.com.ics
Ruben og June_spv56qkm2k0s2pk9scj0d7v0gg@group.calendar.google.com.ics
Soldagen BHG_8ubvgfhra10ah9aoroh3c1o6nc@group.calendar.google.com.ics
TEKNOBYGGET 1.005 Rom 1.005, Undervisningsrom
Teknologibygget Møterom 3.028
Anders Andersen
Week 13
Monday 23.03.2015
Tuesday 24.03.2015
Wednesday 25.03.2015
Thursday 26.03.2015
Friday 27.03.2015
All Day
Showing all simultaneous events

Figure 5-31 All simultaneous events on Thursday

5.5 Performance

The need to scale this Web Service would be limited since we have designed the solution to handle problems within a specific organization. In addition to this, it would be natural to limit the number of calendars viewed at the same time and also the date interval viewed at the same time. It would be natural to view a calendar for one week or one month, maybe one year. This would mean that the number of elements retrieved from each connector would be very limited. We have produced some test results to give a pointer on how the Web Service performs.

The testing is done through the GUI, and the time measured is for the execution of the script on the Web Service, eliminating the time it takes to load the HTML to the client since this would not apply to the other interfaces.

We use the My Calendar function and test how long it takes to retrieve a number of calendar elements from each connector, except the AD connector that does not contain any calendar information.

Without any calendar elements the My Calendar loads in 3713.46 milliseconds the first time and 1008.28 milliseconds the second time. The first time the page loads, the Web Service authenticates with Fronter. This authentication is complex and take some time as shown in Figure 2-5. Since the authentication against Fronter is valid for two hours, each additional request will be faster and therefore we will use that time as the initial loading time.

In Table 5-2 we have listed the result of our testing. We have tested how long it takes to retrieve approximately 10, 100 and 500 elements from each connector, and how long it takes to retrieve the same number of elements from each connector at the same time.

As we would expect, the performance vary on which connector is being used. This is a result of several different things such as the performance of the external system, the amount of data being transferred over the network and how much processing of the data we need to do on the Web Service.

The Fronter connector retrieves raw iCalendar data from Fronter. The Web Service only need to parse this data right into the internal representation without any other processing. This makes the Fronter connector the fastest when dealing with both a few and a large number of calendar elements. Since the data structure is raw iCalendar format, there is now unnecessary overhead being transferred between Fronter and the Web Service. The only optimization to be done here is to fix the file storage workaround as discussed earlier.

The data transfer and data retrieval is fast in the Syllabus connector, but we need to do extensive processing on the data being retrieved to calculate the events from the activities. This means that the more activities we retrieve from Syllabus, the more activities we must calculate the events from. This makes the Syllabus connector slowest when dealing with a lot of elements. We have optimized the code from the first draft, but since the bottleneck is on the Web Service more optimization of the code should be done. The calculation of the events could be done in parallel threads to speed up the process.

The slowest connector when dealing with few elements is the Exchange connector. One of the reasons for this is because SOAP messages have a large overhead. As we see the Exchange connector is the second fastest when dealing with 100 and 500 elements. There is not much to do when it comes to the overhead of SOAP messages, but optimization on how the requests are made could be done. We could manage with only one request instead of three if we don't need the description of the calendar events. Since the GetUserAvailability operation has a limit on 100 mailboxes, the reduction of one third of the request becomes significant for each additional 100 mailboxes we want calendar events from.

Number of elements from Connector (number of calendars)				**Average loading time (MS)**[21]
Syllabus	**Exchange**	**Fronter**	**Total**	
0	0	0	0	1008.281302
13 (1)	0	0	13 (1)	1089.42349
107 (57)	0	0	107 (57)	2190.917087
505 (175)	0	0	505 (175)	4962.859011
0	13 (1)	0	13 (1)	1328.198409
0	109 (20)	0	109 (20)	1649.468899
0	505 (100)	0	505 (100)	2735.23736
0	0	13 (1)	13 (1)	1039.432645
0	0	105 (1)	105 (1)	1477.718782
0	0	471 (1)	471 (1)	2604.579759
13 (1)	13 (1)	13 (1)	39 (3)	1416.869068
107 (57)	113 (20)	105 (1)	325 (78)	2826.924992
484 (175)	502 (100)	473 (1)	1459 (276)	7719.504499

[21] We refresh the page 10 times with CTRL+R to remove cache and use the average time.

Table 5-2 Execution time on the Web Service with a different number of elements from different connectors

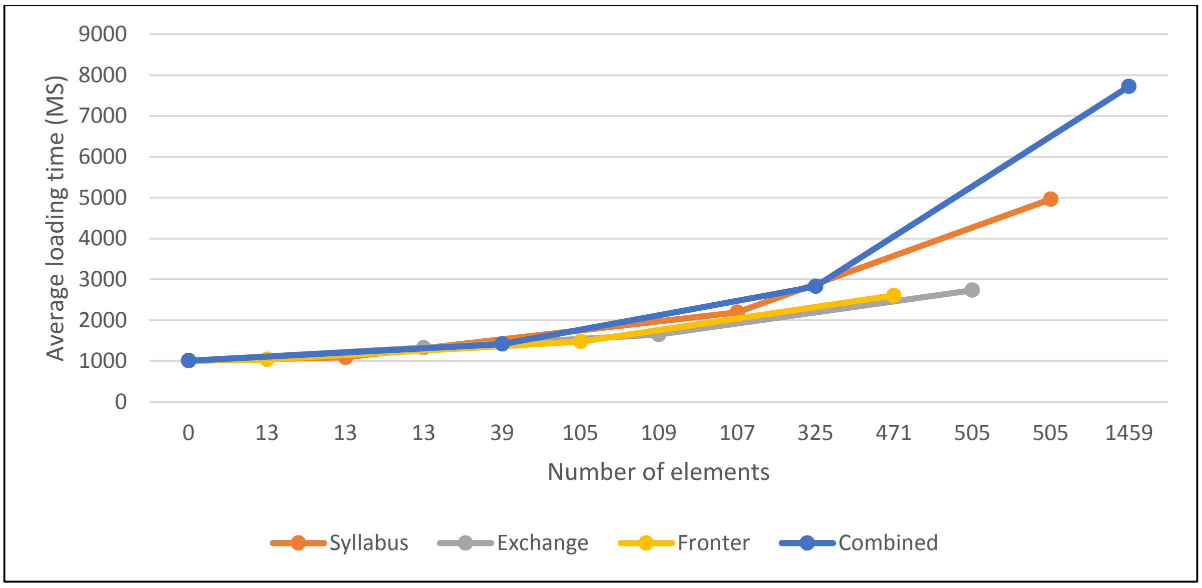


Figure 5-32 Graph illustrating the average loading time with a different number of elements retrieved from the connectors

In addition to the performance we have measured, the time it takes to transfer the data to an end user application trough the ICAL or API must be applied. We consider this time to be out of our control and insignificant for the performance of the Web Service.

When it comes to the user experience we consider one to two seconds loading time to be acceptable. There are many optimization techniques that could be applied to achieve such a performance from our Web Service. In the GUI we could implement AJAX requests to load some elements after the page has loaded. This is a widely used technique when developing web sites.

Applications that uses the API could also only load the calendar elements for one day, one week or one month first, and then load the rest of the year while the user is browsing the current day, week or month. When using the ICAL interface the same technique could be done, limit the requested calendar size by date.

5.6 Summary

In this chapter we demonstrate what the Web Service bring to the table in contrast to using each individual external system. Although the Web Service in the current version has some limitations, only viewing the information without the possibility of updating the external systems, we see that it is a useful planning tool.

We have also seen how the Web Service performs and scale. Most of the performance is dependent on the external systems, except for Syllabus that requires heavy computations on the Web Service.

6 Evaluation

In this chapter we evaluate our generic calendar platform in several different ways. From the technical side to the actual prototype and further extensions.

6.1 Technical

When developing a generic platform, one thing becomes very clear; the programming platform you choose needs to be very flexible. In this thesis we interact with four different external systems, and none of the systems offer the same way of communicating with their respective APIs. If they even have an API. This means that we needed to implement four connectors for communication. It is also essential that the platform chosen supports the underlying communication protocols that is needed. In our case HTTP, LDAP and TNS[22].

In addition to the support of the actual protocols, it speeds up the development if the platform has lots of libraries to use when programming. Or, if the libraries are not a part of the core platform, the community around the platform has developed Open Source projects for everyone to use. The platform we choose had native support for HTTP, extensions for LDAP and TNS, and Open Source projects that implemented connectors to communicate with AD, EWS and to successfully authenticate through OAuth.

With all the technical hurdles in place we were all set to develop the actual software. This meant that we needed to read much of the technical documentation for each system to understand the request and result sets. In some cases, as with Syllabus, no documentation where available. Here we needed to rely entirely on other peoples experience and expertise of the system. This can be both good and bad. On one hand, we can ask about exactly what we want to do and get the right answer the first time. But on the other hand, it becomes very difficult to do complex operations if the source of information becomes unavailable for different reasons. We prefer to have good and detailed written documentation at hand, or a large community around the systems.

Most of the data structures where different. This meant that we had to create a new internal data structure and map all of the structures from the external systems into our new structure. Semantics comes into play here. The attribute *Room* in one system does not automatically mean the same as the attribute *Room* in another system.

6.2 Contribution

When starting this journey, we looked at the problem that occurs when an organization has many different software's that contain the same type of information, and we feel that we have solved the problem to some degree. The platform we have developed works excellent when the main goal is to get an oversight of several different calendars. If the user wants to do more complex operations, like booking a meeting room or creating calendar events, he has to log in to the respective system to do so. The platform could be extended to do more complex operations, but the goal should not be to implement all of the futures that the origin systems contains.

[22] TNS stands for Transparent Network Substrate and forms the basis for Oracle networking products. In our case, an Oracle database.

The university has come a long way at the course of this thesis. Although some people at the university's IT department knew what we were working on, they continued to further develop their own solutions. One example is that the course timetable from the university's timetable portal can now be integrated with external calendar systems. This is exactly the functionality we were first to implement with the ICAL interface.

Another similarity is that the user's and teachers course timetables is exported to Exchange. This is also somewhat up our ally, but we are combining the calendars in a new system instead of converting into one system as the university is doing. There are pros and cons on both approaches, and we believe a combination is the best way.

We did also observe that when we started many meeting rooms was in Syllabus, and the Exchange did only contain one room list. Now, every meeting room is in Exchange, and there are many room lists also. Since we combine room lists and hide the location of the rooms, this transfer is not noticeable for the users of our platform.

We believe that the generic calendar platform has great potential for further development. We also believe that some of our findings are relevant to the university since they seem to move in the same direction.

6.3 Outside the organization

When looking outside the organization, we find even more calendar systems. It would be impossible to make one complete system, but it would be possible to make our platform even more complete by implementing some state. For example, it could be possible to allow users to store shared calendars by URL's from other calendar services. Then users could get information from their Google Calendar etc. integrated on our platform. This would be better than the other way around, since our platform depends on external systems that requires authentication on every request.

By extending the database, we could allow users to store searches or combined calendars. Then the user could create a custom calendar that consists of other calendars as their default calendar. Or otherwise easy to access, without having to search for the same calendars every time the user access the system.

6.4 Other projects

A native Android application, or any mobile application for that matter, could use the API that the platform offers. Then, a user could log in to the Android application and get the same functionality that the GUI offers. The Android application would then be in charge of handling the user's credentials.

When writing a native mobile application it is possible to access things that the web browser cannot access, like the near field communication (NFC) chip on the device if this exist. The NFC chip could be used against NFC tags on outside meeting rooms etc. The NFC tags would contain the ID of the room, and the application can use that ID to get the rooms' calendar from the platform using the API.

Other protocols for calendar sharing than the ones we have covered is also available. Z-Push is an open source ActiveSync project (Z-Push Technology, 2015). By implementing Z-Push as an additional interface of the platform, any calendar could be synchronized to any native mobile phone calendar or other calendar system that supports the Active Sync protocol.

A Z-Push fork on GitHub called PHP-Push-2 is a modified version of Z-Push (dupondje, 2013). This library would be a perfect match for the programming language we have chosen to develop the generic calendar platform in.

7 Conclusion

In this chapter, we are going to make a conclusion and suggest further work.

7.1 Conclusion

We thought that yet another calendar system would solve an organizations problem with having many different types of software that contains calendar information. And that this would be a trivial task. But, as we discovered, not every system wants to share information with other systems. They simply lack the API to do this. And even if you find a way to communicate with all the systems, the data structure is most likely not the same. Sometimes, not even close.

It turns out that creating a new system is helpful. The generic calendar platform that we have created has been used by real people during this thesis, and we have received requests from other departments of the university asking about the possibilities to expand the system to create new events, not just view them. With a bit of reverse engineering, almost everything can be done. The integration with Syllabus and Fronter was not straight forward, but possible.

7.2 Further work

We feel that we have touched most of the areas when making a new calendar platform. But the components we have created could always be improved. There are four particular things we feel that should be improved if someone would to take the on the project.

As mentioned earlier, we use HTTP Basic Authentication for anyone that would want to use the API of the platform. The main weakness of this protocol is that it passes the username and password in the header on every request, and the credentials are only base64 encoded. This makes SSL a must have, but in a man-in-the-middle SSL exploit the credentials of the user are left wide open. A better authentication protocol should be used, preferably with a token management capability to limit access to secured resources. If we had more time, we would look into OAuth 2.0.

From a user point of view, it would be nice to be able to add and edit events instead of only viewing them. In our system the user can get a full view of all the calendars from every system, but if the user wants to book a meeting, or add an event, he still would have to access the specific system. This would require a substantial amount of work.

The integration of student sets are not complete. All the student sets are groups in the AD in the same way as the courses, and every student that takes a student set is a member of that group. But we could not find a way to link the AD group with the student set in Syllabus in the same way that we did with courses. The IT department at the university is also in doubt that this is possible. There seems to be missing some information in the AD group to get a good identifier. A solution to this will be to change the group in AD, or integrate the platform with a system that has more complete information on which student set a student is taking. As mentioned earlier, the university has systems with this information.

Another thing that we did not get the time to implement is a reverse calendar. In our system, you can easily see if people are busy or free, but it would be nice to have the system suggest dates and times when all the calendars you add are available to have the same meeting. For example, we want a 2 hour meeting in week 7, in the building X with a number of people. System, please find a date and time.

And of course, one could always integrate even more systems by developing more connectors.

Bibliography

Amazon. (2006, March 1). *Signing and Authenticating REST Requests*. Retrieved from AWS Documentation: http://docs.aws.amazon.com/AmazonS3/latest/dev/RESTAuthentication.html

Andreassen, R. (2014, August 26). *Files Changed*. Retrieved from GitHub: https://github.com/jamesiarmes/php-ews/pull/208/files

Bailey, J. (2013, September 11). *O365: Exchange and AD - How msExchRecipientDisplayType and msExchangeRecipientTypeDetails Relate to Your On-Premises.* Retrieved from TechNet Blogs: http://blogs.technet.com/b/johnbai/archive/2013/09/11/o365-msexchangerecipienttypedetails.aspx

Boeyen, S., Howes, T., & Richard, P. (1999, April). *Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2*. Retrieved from Internet Engineering Task Force: http://tools.ietf.org/html/rfc2559

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004, February 11). *Web Services Architecture*. Retrieved from W3C: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest

Caffrey, M. (2011, November). *Modeling and Accessing Relational Data*. Retrieved from Oracle: http://www.oracle.com/technetwork/issue-archive/2011/11-nov/o61sql-512018.html

Crupi, J., & Warner, C. (2009, January 11). *SOA World Magazine*. Retrieved from Enterprise Mashups: The New Face of Your SOA: http://soa.sys-con.com/node/719917

Cugley, D. (2007, August 8). *OpenID versus Single-Sign-On Server*. Retrieved from Damian Cugley's Alleged Articles: http://alleged.org.uk/pdc/2007/08/13.html

Daboo, C., & Desruisseaux, B. (2012, June). *Scheduling Extensions to CalDAV*. Retrieved from Internet Engineering Task Force: http://tools.ietf.org/html/rfc6638

Daboo, C., Desruisseaux, B., & Dusseault, L. (2007, March). *Calendaring Extensions to WebDAV (CalDAV)*. Retrieved from Internet Engineering Task Force: http://tools.ietf.org/html/rfc4791

Davis, A. M. (1992). Operational Prototyping: A New Development Approach. *IEEE*, 70-78.

Desruisseaux, B. (2009, September). *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*. Retrieved from The Internet Engineering Task Force: https://tools.ietf.org/html/rfc5545

dupondje. (2013, October 5). *PHP-Push-2*. Retrieved from GitHub: https://github.com/dupondje/PHP-Push-2

European Commission. (2014, June 27). *Cookies.* Retrieved from European Commission, Information Providers Guide: http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Irvine, California, United States of America: University of California.

Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)*, 115-150.

Fielding, R., & Reschke, J. (2014, June). *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. Retrieved from Internet Engineering Task Force (IETF): https://tools.ietf.org/html/rfc7231

Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., & Stewart, L. (1999, June). *HTTP Authentication: Basic and Digest Access Authentication*. Retrieved from Internet Engineering Task Force: http://tools.ietf.org/html/rfc2617

Fronter. (2014, January). *Oauth In Fronter.* Retrieved from Fronter Wiki Oauth: http://wiki.fronter.net/wiki/index.php/Oauth_in_fronter

Fronter. (2014, January). *OpenAPIv1.* Retrieved from Fronter Wiki OpenAPIv1: http://wiki.fronter.net/wiki/index.php/OpenAPIv1

Fronter. (2015, January). *Fronter is a virtual learning environment*. Retrieved from Fronter: http://com.fronter.info/virtual-learning-environment-lms/

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., & Lafon, Y. (2007, April 27). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. Retrieved from W3C: http://www.w3.org/TR/soap12/

Hammer-Lahav, E. E. (2010, April). *The OAuth 1.0 Protocol*. Retrieved from Internet Engineering Task Force (IETF): http://tools.ietf.org/html/rfc5849

Hardt, D. (2012, October). *The OAuth 2.0 Authorization Framework*. Retrieved from Internet Engineering Task Force (IETF): http://tools.ietf.org/html/rfc6749

Hodges, J., & Morgan, R. (2002, September). *Lightweight Directory Access Protocol (v3): Technical Specification*. Retrieved from Internet Engineering Task Force: http://tools.ietf.org/html/rfc3377

Huntington, G. (2015, January). *SSO and LDAP Authentication*. Retrieved from Authentication World: http://www.authenticationworld.com/Single-Sign-On-Authentication/SSOandLDAP.html

IANA. (2012, September 23). *Resource Identifier (RI) Scheme name: webcal* . Retrieved from Internet Assigned Numbers Authority: http://www.iana.org/assignments/uri-schemes/prov/webcal

IBM Corporation, & Microsoft Corporation. (2002, April 7). *Security in a Web Services World: A Proposed Architecture and Roadmap*. Retrieved from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/ms977312.aspx

Internet Engineering Task Force. (2015, May). *The Internet Engineering Task Force (IETF®)*. Retrieved from Internet Engineering Task Force: https://www.ietf.org/

Krawczyk, H., Bellare, M., & Canetti, R. (1997, February). *HMAC: Keyed-Hashing for Message Authentication*. Retrieved from Internet Engineering Task Force: http://www.ietf.org/rfc/rfc2104.txt

Mainer, M. (2011, July 20). *Client access server affinity and network load balancing considerations for programmatic access to Exchange Online*. Retrieved from Exchange dev blog: http://blogs.msdn.com/b/exchangedev/archive/2011/07/20/client-access-server-affinity-and-network-load-balancing-considerations-for-programmatic-access-to-exchange-online.aspx

Microsoft. (2007, March 2). *Ambiguous Name Resolution for LDAP in Windows 2000*. Retrieved from Microsoft: http://support.microsoft.com/en-us/kb/243299

Microsoft. (2009, March 16). *BusyType.* Retrieved from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/aa579531%28v=exchg.140%29.aspx

Microsoft. (2010, September 21). *Exchange Web Services (EWS) in Exchange 2010*. Retrieved from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/dd877045%28v=exchg.140%29.aspx

Microsoft. (2011, September 26). *GetRoomLists Operation.* Retrieved from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/dd899416%28v=exchg.140%29.aspx

Microsoft. (2011, September 26). *GetRooms Operation.* Retrieved from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/dd899415%28v=exchg.140%29.aspx

Microsoft. (2011, September 14). *GetUserAvailability Operation.* Retrieved from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/aa564001%28v=exchg.140%29.aspx

Microsoft. (2012, November 28). *GetItem operation.* Retrieved from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/aa565934%28v=exchg.150%29.aspx

Microsoft. (2013, Mai 17). *Access to Active Directory*. Retrieved from Microsoft Exchange: https://technet.microsoft.com/en-us/library/aa998561%28v=exchg.150%29.aspx

Microsoft. (2013, July 1). *GetItem operation (calendar item).* Retrieved from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/aa564509%28v=exchg.150%29.aspx

Microsoft. (2014, April 3). *ConvertId operation*. Retrieved from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/bb799665%28v=exchg.150%29.aspx

Microsoft. (2014, June 16). *EWS operations in Exchange.* Retrieved from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/bb409286%28v=exchg.150%29.aspx

Microsoft. (2014, June 19). *How to query Active Directory by using a bitwise filter.* Retrieved from Microsoft Support: http://support.microsoft.com/kb/269181

Microsoft. (2014, January). *IdFormat enumeration.* Retrieved from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/microsoft.exchange.webservices.data.idformat%28v=exchg.80%29.aspx

Microsoft. (2014, October). *Object Naming*. Retrieved from Microsoft TechNet: http://technet.microsoft.com/en-us/library/cc977992.aspx

Microsoft. (2014, October). *So What Is Active Directory?* Retrieved from Windows Dev Center - Desktop: http://msdn.microsoft.com/en-us/library/windows/desktop/aa746492%28v=vs.85%29.aspx

Microsoft. (2015, January). *2.1.3.1.1.20.31 Property: X-MICROSOFT-CDO-BUSYSTATUS*. Retrieved from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/ee219533%28v=exchg.80%29.aspx

Microsoft. (2015, January). *All Attributes*. Retrieved from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/ms675090%28v=vs.85%29.aspx

Microsoft. (2015, January). *Exchange Server 2013*. Retrieved from Microsoft Office: https://products.office.com/en-us/exchange/microsoft-exchange-server-2013

Microsoft. (2015, January). *Service-Oriented Architecture (SOA)*. Retrieved from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/bb977471.aspx

Microsoft Corporation. (2001, November). *Creating More Efficient Microsoft Active Directory-Enabled Applications.* Retrieved from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/ms808539.aspx#efficientadapps_topic01e

Mueller, R. (2014, January 20). *Active Directory: Ambiguous Name Resolution.* Retrieved from Microsoft TechNet: http://social.technet.microsoft.com/wiki/contents/articles/22653.active-directory-ambiguous-name-resolution.aspx

Mueller, R. (2014, July 10). *Active Directory: LDAP Syntax Filters*. Retrieved from Microsoft Tech Net: http://social.technet.microsoft.com/wiki/contents/articles/5392.active-directory-ldap-syntax-filters.aspx

Nielsen, J. (1993). *Usability Engineering.* Boston: Academic Press.

Oracle. (2015, January). *Introduction to the Oracle Database*. Retrieved from Oracle: http://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm

Orakelet. (2015, January). *Støttesystem for oppretting av IT-brukerkonto*. Retrieved from UiT: http://uit.no/om/orakelet/frag?p_document_id=318999

Orakelet. (2015, February 23). *TimeIT - Undervisningsplan på telefon og kalender*. Retrieved from UiT Norges Arktiske Universitet: http://uit.no/om/orakelet/frag?p_document_id=406126

Samferdselsdepartementet. (2013, June 14). *Lov om elektronisk kommunikasjon (ekomloven).* Retrieved from Lovdata: http://lovdata.no/dokument/NL/lov/2003-07-04-83/KAPITTEL_2#%C2%A72-7b

Scientia. (2003, October 21). *All SDB Schema Tables.* Retrieved from SDB Technical Handbook: SDBSchema.html

Scientia. (2014, October). *Timetable Scheduling*. Retrieved from Scientia: http://www.scientia.com/en-GB/Solutions/Timetable-Scheduling

SquirrelMail. (2014, January). *This code provides various string manipulation functions that are used by the rest of the SquirrelMail code.* Retrieved from SquirrelMail: http://squirrelmail.org/docs/devel-code/squirrelmail/_functions---strings.php.html#functionOneTimePadCreate

The PHP Group. (2014, January). *LDAP Functions (ldap_connect).* Retrieved from PHP: http://php.net/manual/en/function.ldap-connect.php

The PHP Group. (2014, January). *LDAP Functions (ldap_start_tls).* Retrieved from PHP: http://php.net/manual/en/function.ldap-start-tls.php

The PHP Group. (2014, January). *similar_text*. Retrieved from PHP: http://php.net/manual/en/function.similar-text.php

The PHP Group. (2015, January). *Client URL Library*. Retrieved from PHP: http://php.net/manual/en/book.curl.php

W3C. (2014, January). *W3C Mission*. Retrieved from W3C: http://www.w3.org/Consortium/mission#openstand

Zeilenga, E. K. (2006, June). *Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*. Retrieved from The Internet Engineering Task Force (IETF): http://tools.ietf.org/html/rfc4510

Zeilenga, K. (2003, March). *Lightweight Directory Access Protocol version 2 (LDAPv2) to Historic Status*. Retrieved from The Internet Engineering Task Force (IETF): http://tools.ietf.org/html/rfc3494

Z-Push Technology. (2015, May). Retrieved from Z-Push: http://z-push.org/

Bibliography

Appendix A Fronter login flow

Step	**Method**	**HTTP status code**	**Type**	**URL**
1	GET	302	Redirect to: https://fronte...main.phtml	https://fronter.com/uit/
Function in code This is the fronterLoginStep01() function in the uitFronter class.				
2	GET	200	text/html	https://fronte...main.phtml
Description This is a HTML page with a FORM. The default behavior of this page is that a JavaScript function in the HTML body onload method submits the form when the page is loaded. All the input values are server generated and do not need any user interference.				
Form information Action: https://fronter.com/shibboleth/feide/saml//sp20/idpdisco.php Method: GET Input name (value): entityID (https://fronter.com/shibboleth/feide) return (https://fronter...main.phtml) returnIDParam (entityID) idp_https://idp.feide.no (Select)				
Function in code This is the fronterLoginStep02() function in the uitFronter class.				
3	GET	302	Redirect to: https://fronte...feide.no	https://fronte...ntinue
4	GET	302	Redirect to: https://idp.fei...main.phtml	https://fronte...feide.no
5	GET	302	Redirect to: https://idp.fei...main.phtml	https://idp.fei...main.phtml
6	GET	200	text/html	https://idp.fei...main.phtml
Description This is a HTML page with a FORM. The default behavior of this page is to let the user choose affiliation and submit the form. Most of the input values are server generated. The user only needs to select the right organization (org). Note that the Action of this form is “?”. This means that all the parameters in the URL that were used when retrieving the page will be gone from the URL when the user submits the form.				
Form information Action: ? Method: GET Input name (value): **org (uit.no)** asLen (239)				

authState (_354e3b729279486a6a37f763ca323391a87187553f:https://idp.feide....)				
Function in code This is the fronterLoginStep03() function in the uitFronter class.				
7	GET	200	text/html	https://idp.fei...
Description This is a HTML page with a FORM. The default behavior of this page is to let the user enter username and password and submit the form. Most of the input values are server generated. The user only needs to enter username and password.				
Form information Action: (?asLen=239&amp;AuthState...) Method: POST Input name (value): **feidename (<fill with username>)** **password (<fill with password>)** asLen (239) authState (_354e3b729279486a6a37f763ca323391a87187553f:https://idp.feide.no...) org (uit.no) inside_iframe (0)				
Function in code This is the fronterLoginStep04() function in the uitFronter class.				
8	POST	200	Text/html	https://idp.fei...main.phtml
Description This is a HTML page with a FORM. The default behavior of this page is that a JavaScript function in the HTML body onload method submits the form when the page is loaded. All the input values are server generated and do not need any user interference.				
Form information Action: https://fronter.com/shibboleth/feide/saml/sp20/AssertionConsumerService.php Method: POST Input name (value): SAMLResponse (PHNhbWxwOlJlc3… wOlJlc3BvbnNlPg==) RelayState (https://fronter.com/uit/main.phtml)				
Function in code This is the fronterLoginStep05() function in the uitFronter class.				
9	POST	302	Redirect to: https://fronter...main.phtml	https://fronter...ice.php
10	GET	200	text/html	https://fronter...main.phtml
Description When we get this far we are authenticated (logged in to Fronter). This is a page that will redirect to https://fronter.com/uit/main.phtml, but we don't need to go there.				

Appendix B GUI Cookies notice

IMPORTANT NOTICE: PLEASE READ CAREFULLY BEFORE USING THIS WEBSITE: If you log in to this site you agree to the placement of cookies on your computer in accordance with the terms of this policy. If you do not wish to accept cookies from this site please refrain from using this site.

1. What are Cookies?

A cookie is a text-only string of information that a website transfers to the cookie file of the browser on your computer's hard disk so that the website can recognize you when you revisit and remember certain information about you. This can include which pages you have visited, choices you have made from menus, any specific information you have entered into forms and the time and date of your visit.

2. Types of Cookies

There are two main types of cookies:

Session cookies: these are temporary cookies that expire at the end of a browser session; that is, when you leave the site. Session cookies allow the website to recognize you as you navigate between pages during a single browser session and allow you to use the website most efficiently. For example, session cookies enable a website to remember that a user has placed items in an online shopping basket.

Persistent cookies: in contrast to session cookies, persistent cookies are stored on your equipment between browsing sessions until expiry or deletion. They therefore enable the website to "recognize" you on your return, remember your preferences, and tailor services to you.

In addition to session cookies and persistent cookies, there may be other cookies which are set by the website which you have chosen to visit, such as this website, in order to provide us or third parties with information.

3. Our use of Cookies

We currently use, and may use in the future, the following types of cookies on this website.

We use session cookies to help us maintain security and verify your details whilst you use the website as you navigate from page to page, which enables you to avoid having to re-enter your details each time you enter a new page.

We are not using any persistent cookies.

We are not using any third party cookies.

You can read detailed information about the cookies we use in the table below.

Cookie Name	Details	More Information
PHPSESSID	This cookie contains the ID of your session on the web server. This is a way to preserve certain data across subsequent requests to our site.	

username	Your username encrypted with 256-bit AES. The cookie expire after 12 hours or when you log out.	Our site relies on information from third party services such as Microsoft Active Directory, Microsoft Exchange, Syllabus and Fronter. Some of these services requires authentication on every request. We encrypt your credentials and store them in cookies so you don't have to enter them on every page.
password	Your password encrypted with 256-bit AES. The cookie expire after 12 hours or when you log out.	

4. Refusing Cookies on this Site

Most browsers are initially set to accept cookies. However, you have the ability to disable cookies if you wish, generally through changing your internet software browsing settings. It may also be possible to configure your browser settings to enable acceptance of specific cookies or to notify you each time a new cookie is about to be stored on your computer enabling you to decide whether to accept or reject the cookie. To manage your use of cookies there are various resources available to you, for example the “Help” section on your browser may assist you. You can also disable or delete the stored data used by technology similar to cookies, such as Local Shared Objects or Flash cookies, by managing your browser’s “add-on settings” or visiting the website of its manufacturer. As our cookies are required for the website to work properly we recommend that you leave cookies enabled. Otherwise, if cookies are disabled, it will be prevented from using this site altogether.

Appendix C REST API v1 documentation

Combine

Resource	Description
GET api/v1/combine/calendar	Get calendar items from all specified calendars. The calendars can be from any user, room, course or student set.

User

Resource	Description
POST api/v1/user/auth	Authenticates a user's credentials against Active Directory at the university.
GET api/v1/user/search	Search for a specific user.
GET api/v1/user/info/:id	Get information on a specific user.
GET api/v1/user/course/:semester/:id	Get all the courses a specific user is taking or teaching.
GET api/v1/user/calendar/:id	Get calendar items from a specific user's calendar.

Room list

Resource	Description
GET api/v1/roomlist/all	Get a list of all room list's available.
GET api/v1/roomlist/search	Search for a specific room list.
GET api/v1/roomlist/room/:type/:id	Get a list of all rooms in a given room list.

Room

Resource	Description
GET api/v1/room/all	Get a list of all rooms available.
GET api/v1/room/search	Search for a specific room.
GET api/v1/room/calendar/:type/:id	Get calendar items from a specific room resource.

Course

Resource	Description
GET api/v1/course/search	Search for a specific course.
GET api/v1/course/calendar/:id	Get calendar items from a specific course.

Student set

Resource	Description
GET api/v1/studentset/search	Search for a specific student set.
GET api/v1/studentset/calendar/:id	Get calendar items from a specific student set.

GET api/v1/combine/calendar

Returns calendar items from all specified calendars. The calendars can be from any user, room, course or student set.

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/combine/calendar

Parameters

Parameter	Description
users optinal	The IDs of the users that you want calendar items for separated by semicolon. **Example:** ath025@post.uit.no **Example:** ath025@post.uit.no;ran033@post.uit.no Refer to the GET api/v1/user/calendar documentation for more info.
rooms optinal	The IDs and types of the room that you want calendar items for. The ID and type are separated with colon, and the IDs and types are separated by semicolon. **Example:** MHU9.115:2 **Example:** MHU9.115:2;MH.U9.123.Moterom@asp.uit.no:1 Refer to the GET api/v1/room/calendar documentation for more info.
courses optinal	The IDs of the courses that you want calendar items for separated by semicolon. **Example:** INF-3320-1 **Example:** INF-3320-1;INF-1101-2 Refer to the GET api/v1/course/calendar documentation for more info.

studentsets optinal	The IDs of the student sets that you want calendar items for separated by semicolon. **Example:** IMAT-FYMA13-ANB1-2 **Example:** IMAT-FYMA13-ANB1-2;B-FYSIOTER-06-ANB1-1 Refer to the GET api/v1/studentset/calendar documentation for more info.
mode optional	The calendar has four modes that specify the date range of calendar items to be returned. The mode parameter must be one of the following: day - calendar items for the date specified week - calendar items for the whole week. The "date" parameter can be any date in the week month - calendar items for the whole month. The "date" parameter can be any date in the month interval - calendar items for a custom date interval Default value: "week" **Example:** week
date optional	The date to use with the mode parameter. The date format is YYYYMMDD Default value: The current date **Example:** 20140115
todate optional	The date to use with the "interval" mode parameter. The date format is YYYYMMDD "todate" must be in the same year as "date" **Example:** 20140120

POST api/v1/user/auth

Authenticates a user’s credentials against Active Directory at UiT. Returns true or false.

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	POST

Resource URL

https://studentlink.ifi.uit.no/api/v1/user/auth

Parameters

Parameter	Description
user required	The username for the user that’s going to be authenticated. Use only the username, not the e-mail address. **Example:** ran033
pass required	The password for the user that’s going to be authenticated.

GET api/v1/user/search

Search for a user. Returns a list of users that matches the search string.

The Type returned can be one of the following:
The Type variable is unused

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/user/search

Parameters

Parameter	Description
query required	The name of the user that you are searching for. It can be the whole name or a partial string. The search is not case sensitive. **Example:** thorsen

GET api/v1/user/info/:id

Returns information on a specific user.

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/user/info/:id

Parameters

Parameter	**Description**
id optional	The ID of the user that you want information about. If this parameter is not set, info of the user that is authenticating are returned. **Example:** ath025@post.uit.no

GET api/v1/user/course/:semester/:id

Returns all the courses a specific user is taking or teaching. This information is current and cannot be viewed back or forth in time.

If the user is a teacher, the courses the teacher is responsible for is returned. This information can change on a yearly basis.

The Type returned can be one of the following:
1 = The user is taking the course as a student
2 = The user is teaching the course

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/user/course/:semester/:id

Parameters

Parameter	**Description**
semester required	Which semester. Valid semesters are: 1 = Fall semester 2 = Spring semester **Example:** 1
id optional	The ID of the user that you want information about. If this parameter is not set, info of the user that is authenticating are returned. **Example:** ath025@post.uit.no
year optional	Four digit year. If no year is present, the current year is used. This parameter has only effect if the user is a teacher. Valid years are between 2014 and 2016 **Example:** 2013

GET api/v1/user/calendar/:id

Returns calendar items from a specific user's calendar.

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/user/calendar/:id

Parameters

Parameter	**Description**
id optinal	The ID of the user that you want calendar items for. If this parameter is not set, calendar items for the user that is authenticating are returned. **Example:** ath025@post.uit.no
mode optional	The calendar has four modes that specify the date range of calendar items to be returned. The mode parameter must be one of the following: day - calendar items for the date specified week - calendar items for the whole week. The "date" parameter can be any date in the week month - calendar items for the whole month. The "date" parameter can be any date in the month interval - calendar items for a custom date interval Default value: "week" **Example:**: week
date optional	The date to use with the mode parameter. The date format is YYYYMMDD Default value: The current date **Example:** 20140115

Parameter	**Description**
todate optional	The date to use with the "interval" mode parameter. The date format is YYYYMMDD "todate" must be in the same year as "date" **Example:** 20140120

GET api/v1/roomlist/all

Returns a list of all room lists available.

The web service combines room lists from UiT Exchange and UiT Syllabus database. Valid username and password is required to access the Exchange server.

The Type returned can be one of the following:
1 = Room list from Exchange
2 = Room list from Syllabus
3 = Web Service room list

Note! The Web Service Type is a combination of two or more room lists of the Type Exchange or Syllabus combined by the Web Service if the room list names have a similarity of more than 95% according to the PHP function similar text.

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/roomlist/all

Parameters

Parameter	**Description**
year optional	Four digit year. The room lists in the Syllabus database can change on a yearly basis. Valid years are between 2014 and 2016 Default value: The current year. **Example:** 2014

GET api/v1/roomlist/search

Search for a specific room list. Returns a list of room list's that matches the search string.

The Type returned can be one of the following:
1 = Room list from Exchange
2 = Room list from Syllabus
3 = Web Service room list

Note! The Web Service Type is a combination of two or more room lists of the Type Exchange or Syllabus combined by the Web Service if the room list names have a similarity of more than 95% according to the PHP function similar text.

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/roomlist/search

Parameters

Parameter	Description
query required	The name of the room list that you are searching for. It can be the whole name or a partial string. The search is not case sensitive. **Example:** adm
year optional	Four digit year. The room lists in the Syllabus database can change on a yearly basis. Valid years are between 2014 and 2016 Default value: The current year **Example:** 2014

GET api/v1/roomlist/room/:type/:id

Returns a list of all the rooms in a given room list.

The Type returned can be one of the following:
1 = Room from Exchange
2 = Room from Syllabus

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/roomlist/room/:type/:id

Parameters

Parameter	Description
type required	The Type of the room list The Type can be any of the following: 1 = Exchange room list 2 = Syllabus room list 3 = Web Service room list. **Note!** The Web Service Type is a combination of two or more room lists of the Type Exchange or Syllabus combined by the Web Service if the room list names have a similarity of more than 95% according to the PHP function similar text. With the Web Service Type, the ID parameter will be a combination of several IDs and Types, with the format: <ID>:<Type>;<ID>:<Type> **Example:** 3
id required	The ID of the room list **Example:** KRV.33:2;MV.110:2;#SPLUSA6F58C:2
year optional	Four digit year. The room lists and Rooms in the Syllabus database can change on a yearly basis.

Parameter	Description
	Valid years are between 2014 and 2016 Default value: The current year. **Example:** 2013

GET api/v1/room/all

Returns a list of all rooms available.

The web service combines rooms from UiT Exchange and UiT Syllabus database. Valid username and password is required to access the Exchange server.

The Type returned can be one of the following:
1 = Room from Exchange
2 = Room from Syllabus

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/room/all

Parameters

Parameter	**Description**
year optional	Four digit year. The rooms in the Syllabus database can change on a yearly basis. Valid years are between 2014 and 2016 Default value: The current year. **Example:** 2014

GET api/v1/room/search

Search for a room. Returns a list of rooms that matches the search string.

The Type returned can be one of the following:
1 = Room from Exchange
2 = Room from Syllabus

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/room/search

Parameters

Parameter	Description
query required	The name of the room that you are searching for. It can be the whole name or a partial string. The search is not case sensitive. **Example:** mh u9.11
year optional	Four digit year. The rooms in the Syllabus database can change on a yearly basis. Valid years are between 2014 and 2016 Default value: The current year. **Example:** 2014

GET api/v1/room/calendar/:type/:id

Returns calendar items from a specific room resource.

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/room/calendar/:type/:id

Parameters

Parameter	**Description**
type required	The type of the room that you want calendar items for. The Type can be any of the following: 1 = Exchange Room 2 = Syllabus Room **Example:** 2
id required	The ID of the room that you want calendar items for. **Example:** MHU9.115
mode optional	The calendar has four modes that specify the date range of calendar items to be returned. The mode parameter must be one of the following: day - calendar items for the date specified week - calendar items for the whole week. The "date" parameter can be any date in the week month - calendar items for the whole month. The "date" parameter can be any date in the month interval - calendar items for a custom date interval Default value: "week" **Example:** week
date optional	The date to use with the mode parameter. The date format is YYYYMMDD

Parameter	Description
	Default value: The current day **Example:** 20140115
todate optional	The date to use with the "interval" mode parameter. The date format is YYYYMMDD "todate" must be in the same year as "date" **Example:** 20140120

GET api/v1/course/search

Search for a course. Returns a list of courses that matches the search string.

The Type returned can be one of the following:
The Type variable is unused

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/course/search

Parameters

Parameter	**Description**
query required	The name of the course that you are searching for. It can be the whole name or a partial string. The search is not case sensitive. **Example:** middleware
year optional	Four digit year. The courses in the Syllabus database can change on a yearly basis. Valid years are between 2014 and 2016 Default value: The current year **Example:** 2014

GET api/v1/course/calendar/:id

Returns calendar items from a specific course.

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/course/calendar/:id

Parameters

Parameter	**Description**
id required	The ID of the course that you want calendar items for. **Example:** INF-3320-1
mode optional	The calendar has four modes that specify the date range of calendar items to be returned. The mode parameter must be one of the following: day - calendar items for the date specified week - calendar items for the whole week. The "date" parameter can be any date in the week month - calendar items for the whole month. The "date" parameter can be any date in the month interval - calendar items for a custom date interval Default value: "week" **Example:** week
date optional	The date to use with the mode parameter. The date format is YYYYMMDD Default value: The current date. **Example:** 20140115
todate optional	The date to use with the "interval" mode parameter. The date format is YYYYMMDD

Parameter	Description
	"todate" must be in the same year as "date" **Example:** 20140120

GET api/v1/studentset/search

Search for a student set. Returns a list of student sets that matches the search string.

The Type returned can be one of the following:
The Type variable is unused

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/studentset/search

Parameters

Parameter	**Description**
query required	The name of the student set that you are searching for. It can be the whole name or a partial string. The search is not case sensitive. **Example:** biologi
year optional	Four digit year. The student sets in the Syllabus database can change on a yearly basis. Valid years are between 2014 and 2016 Default value: The current year **Example:** 2014

GET api/v1/studentset/calendar/:id

Returns calendar items from a specific student set.

Resource Information

Authentication	Basic access authentication. Valid UiT username and password, typically of the user using the application.
Response Formats	json
HTML Methods	GET

Resource URL

https://studentlink.ifi.uit.no/api/v1/studentset/calendar/:id

Parameters

Parameter	Description
id required	The ID of the student set that you want calendar items for. **Example:** IMAT-FYMA13-ANB1-2
mode optional	The calendar has four modes that specify the date range of calendar items to be returned. The mode parameter must be one of the following: day - calendar items for the date specified week - calendar items for the whole week. The "date" parameter can be any date in the week month - calendar items for the whole month. The "date" parameter can be any date in the month interval - calendar items for a custom date interval Default value: "week" **Example:** week
date optional	The date to use with the mode parameter. The date format is YYYYMMDD Default value: The current day **Example:** 20140115
todate optional	The date to use with the "interval" mode parameter. The date format is YYYYMMDD

Parameter	Description
	"todate" must be in the same year as "date" **Example:** 20140120

Appendix D Why we use LDAP to retrieve room names instead of EWS

The EWS interface uses SOAP messages to receive and send information to a client. In this appendix we are going to illustrate why we choose to use LDAP over EWS when we want to retrieve a room name based on the email address of the room.

When we want to use the EWS interface we need to use the ResolveNames Operation. In this example we send the following request:

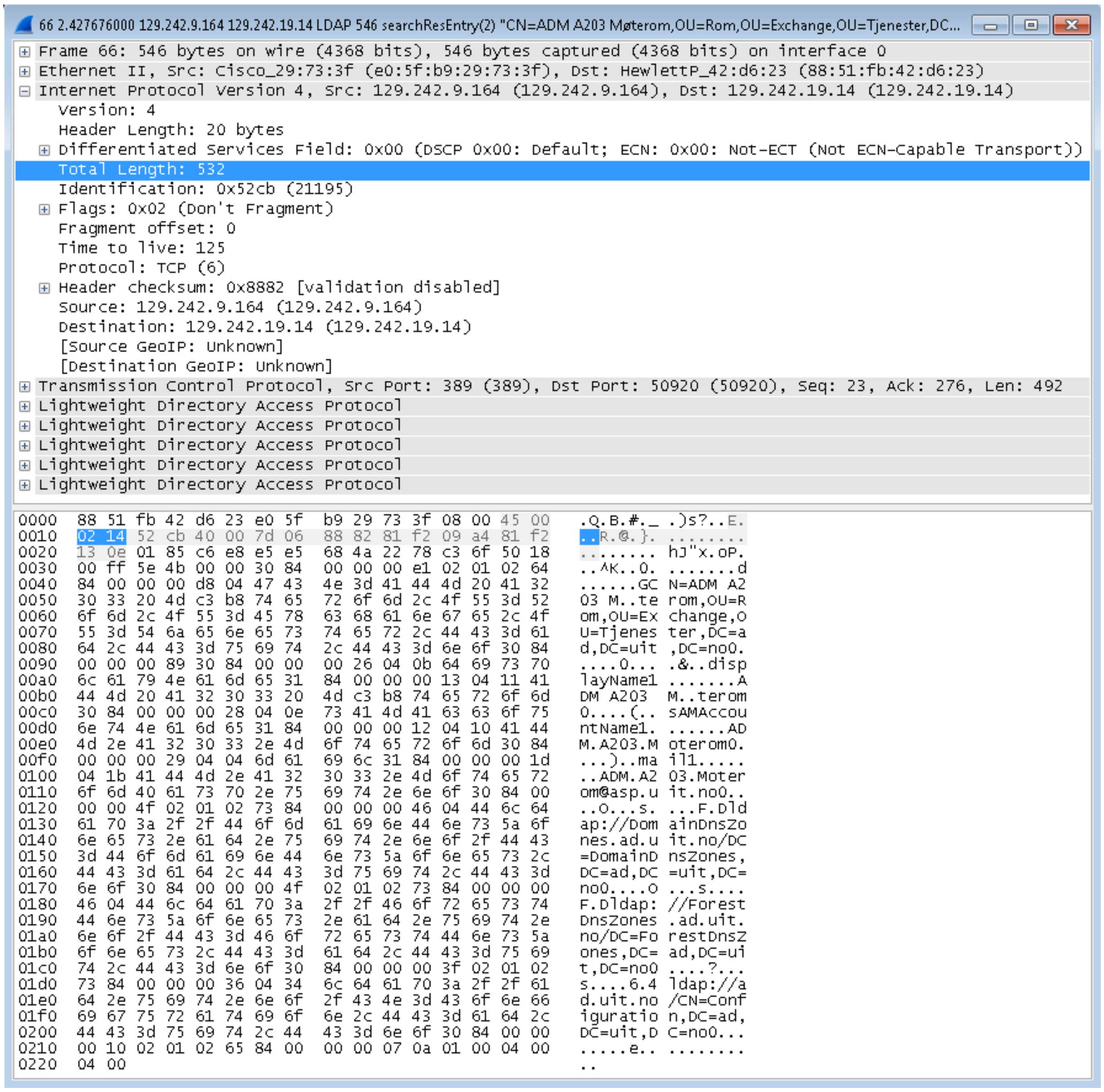
```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
             xmlns:xsd=http://www.w3.org/2001/XMLSchema
             xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
             xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types">
  <soap:Body>
    <ResolveNames xmlns=http://schemas.microsoft.com/exchange/services/2006/messages
               xmlns:t=http://schemas.microsoft.com/exchange/services/2006/types
               ReturnFullContactData="false">
      <UnresolvedEntry>ADM.A203.Moterom@asp.uit.no</UnresolvedEntry>
    </ResolveNames>
  </soap:Body>
</soap:Envelope>
```

This gives us the following reponse:

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <t:ServerVersionInfo MajorVersion="8" MinorVersion="0" MajorBuildNumber="685"
MinorBuildNumber="8"
                 xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types" />
  </soap:Header>
  <soap:Body>
    <ResolveNamesResponse
xmlns:m="http://schemas.microsoft.com/exchange/services/2006/messages"
                  xmlns:t=http://schemas.microsoft.com/exchange/services/2006/types
                 xmlns="http://schemas.microsoft.com/exchange/services/2006/messages">
      <m:ResponseMessages>
        <m:ResolveNamesResponseMessage ResponseClass="Success">
          <m:ResponseCode>NoError</m:ResponseCode>
          <m:ResolutionSet TotalItemsInView="1" IncludesLastItemInRange="true">
            <t:Resolution>
              <t:Mailbox>
                <t:Name>ADM A203 Møterom</t:Name>
                <t:EmailAddress>ADM.A203.Moterom@asp.uit.no</t:EmailAddress>
                <t:RoutingType>SMTP</t:RoutingType>
                <t:MailboxType>Mailbox</t:MailboxType>
              </t:Mailbox>
            </t:Resolution>
          </m:ResolutionSet>
        </m:ResolveNamesResponseMessage>
      </m:ResponseMessages>
    </ResolveNamesResponse>
  </soap:Body>
</soap:Envelope>
```

This response is 3324 bytes in total, and the header alone is 1542 bytes. This means that the header is almost larger than the body of 1782 bytes. And that is only because we set the ReturnFullContactData attribute to false because we can use the Name attribute to retrieve the rooms name. If we would set the ReturnFullContactData attribute to true we would get even more information, including the DisplayName attribute. But for the rooms we assume that the Name and DisplauName attributes are the same.

When we use the LDAP to search for the exact same room, the response is 532 bytes and the header is 20 bytes. This means that the body is 512 bytes. Much less overhead than the SOAP request.



66 2.427676000 129.242.9.164 129.242.19.14 LDAP 546 searchResEntry(2) "CN=ADM A203 Møterom,OU=Rom,OU=Exchange,OU=Tjenester,DC...

```
⊞ Frame 66: 546 bytes on wire (4368 bits), 546 bytes captured (4368 bits) on interface 0
⊞ Ethernet II, Src: Cisco_29:73:3f (e0:5f:b9:29:73:3f), Dst: HewlettP_42:d6:23 (88:51:fb:42:d6:23)
⊟ Internet Protocol Version 4, Src: 129.242.9.164 (129.242.9.164), Dst: 129.242.19.14 (129.242.19.14)
    Version: 4
    Header Length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 532
    Identification: 0x52cb (21195)
  ⊞ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 125
    Protocol: TCP (6)
  ⊞ Header checksum: 0x8882 [validation disabled]
    Source: 129.242.9.164 (129.242.9.164)
    Destination: 129.242.19.14 (129.242.19.14)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
⊞ Transmission Control Protocol, Src Port: 389 (389), Dst Port: 50920 (50920), Seq: 23, Ack: 276, Len: 492
⊞ Lightweight Directory Access Protocol
⊞ Lightweight Directory Access Protocol
⊞ Lightweight Directory Access Protocol
⊞ Lightweight Directory Access Protocol
⊞ Lightweight Directory Access Protocol
```

```
0000  88 51 fb 42 d6 23 e0 5f  b9 29 73 3f 08 00 45 00   .Q.B.#._ .)s?..E.
0010  02 14 52 cb 40 00 7d 06  88 82 81 f2 09 a4 81 f2   ..R.@.}. ........
0020  13 0e 01 85 c6 e8 e5 e5  68 4a 22 78 c3 6f 50 18   ........ hJ"x.oP.
0030  00 ff 5e 4b 00 00 30 84  00 00 00 e1 02 01 02 64   ..^K..0. .......d
0040  84 00 00 00 d8 04 47 43  4e 3d 41 44 4d 20 41 32   ......GC N=ADM A2
0050  30 33 20 4d c3 b8 74 65  72 6f 6d 2c 4f 55 3d 52   03 M..te rom,OU=R
0060  6f 6d 2c 4f 55 3d 45 78  63 68 61 6e 67 65 2c 4f   om,OU=Ex change,O
0070  55 3d 54 6a 65 6e 65 73  74 65 72 2c 44 43 3d 61   U=Tjenes ter,DC=a
0080  64 2c 44 43 3d 75 69 74  2c 44 43 3d 6e 6f 30 84   d,DC=uit ,DC=no0.
0090  00 00 00 89 30 84 00 00  00 26 04 0b 64 69 73 70   ....0... .&..disp
00a0  6c 61 79 4e 61 6d 65 31  84 00 00 00 13 04 11 41   layName1 .......A
00b0  44 4d 20 41 32 30 33 20  4d c3 b8 74 65 72 6f 6d   DM A203  M..terom
00c0  30 84 00 00 00 28 04 0e  73 41 4d 41 63 63 6f 75   0....(.. sAMAccou
00d0  6e 74 4e 61 6d 65 31 84  00 00 00 12 04 10 41 44   ntName1. ......AD
00e0  4d 2e 41 32 30 33 2e 4d  6f 74 65 72 6f 6d 30 84   M.A203.M oterom0.
00f0  00 00 00 29 04 04 6d 61  69 6c 31 84 00 00 00 1d   ...)..ma il1.....
0100  04 1b 41 44 4d 2e 41 32  30 33 2e 4d 6f 74 65 72   ..ADM.A2 03.Moter
0110  6f 6d 40 61 73 70 2e 75  69 74 2e 6e 6f 30 84 00   om@asp.u it.no0..
0120  00 00 4f 02 01 02 73 84  00 00 00 46 04 44 6c 64   ..O...s. ...F.Dld
0130  61 70 3a 2f 2f 44 6f 6d  61 69 6e 44 6e 73 5a 6f   ap://Dom ainDnsZo
0140  6e 65 73 2e 61 64 2e 75  69 74 2e 6e 6f 2f 44 43   nes.ad.u it.no/DC
0150  3d 44 6f 6d 61 69 6e 44  6e 73 5a 6f 6e 65 73 2c   =DomainD nsZones,
0160  44 43 3d 61 64 2c 44 43  3d 75 69 74 2c 44 43 3d   DC=ad,DC =uit,DC=
0170  6e 6f 30 84 00 00 00 4f  02 01 02 73 84 00 00 00   no0....O ...s....
0180  46 04 44 6c 64 61 70 3a  2f 2f 46 6f 72 65 73 74   F.Dldap: //Forest
0190  44 6e 73 5a 6f 6e 65 73  2e 61 64 2e 75 69 74 2e   DnsZones .ad.uit.
01a0  6e 6f 2f 44 43 3d 46 6f  72 65 73 74 44 6e 73 5a   no/DC=Fo restDnsZ
01b0  6f 6e 65 73 2c 44 43 3d  61 64 2c 44 43 3d 75 69   ones,DC= ad,DC=ui
01c0  74 2c 44 43 3d 6e 6f 30  84 00 00 00 3f 02 01 02   t,DC=no0 ....?...
01d0  73 84 00 00 00 36 04 34  6c 64 61 70 3a 2f 2f 61   s....6.4 ldap://a
01e0  64 2e 75 69 74 2e 6e 6f  2f 43 4e 3d 43 6f 6e 66   d.uit.no /CN=Conf
01f0  69 67 75 72 61 74 69 6f  6e 2c 44 43 3d 61 64 2c   iguratio n,DC=ad,
0200  44 43 3d 75 69 74 2c 44  43 3d 6e 6f 30 84 00 00   DC=uit,D C=no0...
0210  00 10 02 01 02 65 84 00  00 00 07 0a 01 00 04 00   .....e.. ........
0220  04 00                                              ..
```

Note! We use the PHP curl_getinfo function to get the SOAP information since we use a SOAP library in PHP that uses curl. To get the LDAP information we use Wireshark since the PHP LDAP library don't have any functions to get the response size. We could note use Wireshark for the SOAP requests since they are encrypted.

Appendix D Why we use LDAP to retrieve room names instead of EWS

Appendix E Open Source Libraries

The generic calendar platform is programmed in PHP. Most of the functionality we use are a part of the PHP platform. However, we use some open source libraries to speed up the development process. The libraries and information about them is listed in this appendix.

adLDAP

License	GNU
URL	http://adldap.sourceforge.net/
Language	PHP

Description
adLDAP is a PHP class that provides LDAP authentication and integration with Active Directory.

AES_Encryption

License	Free to use and modify.
URL	http://www.coderelic.com/2011/10/aes-256-encryption-with-php/
Language	PHP

Description
This class allows you to easily encrypt and decrypt text in AES format. The class automatically determines whether you need 128, 192, or 256 bits based on your key size. It handles multiple padding formats.

ics-parser

License	MIT
URL	http://code.google.com/p/ics-parser/
Language	PHP

Description
This PHP-Class should only read a iCal-File (*.ics), parse it and give an array with its content.

oauth-php

License	MIT
URL	https://code.google.com/p/oauth-php/
Language	PHP

Description
A PHP library for OAuth 1.0a consumers and servers. Complete with an extensible OAuth store, including a full working implementation of MySQL/MySQLi, Postgresql, PDO and Oracle stores.

php-ews

License	Free to use and modify.
URL	http://adldap.sourceforge.net/
Language	PHP

Description

The PHP Exchange Web Services library (php-ews) is intended to make communication with Microsoft Exchange servers using Exchange Web Services easier. It handles the NTLM authentication required to use the SOAP services and provides an object-oriented interface to the complex types required to form a request

jQuery

License	MIT
URL	https://jquery.com/
Language	JavaScript

Description

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

Epoch

License	Free to use and modify.
URL	http://www.epoch-calendar.com/support/getting_iso_week.html
Language	JavaScript

Description

While the JavaScript Date object is one of the most complete of all the default date-handling routines in programming languages, one of the features it is sorely lacking is an algorithm for computing a date's week number.

Bootstrap

License	Free to use and modify.
URL	http://getbootstrap.com/
Language	JavaScript/CSS/HTML

Description

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

Bootstrap-datepicker

License	Apache License
URL	https://github.com/eternicode/bootstrap-datepicker
Language	JavaScript/CSS/HTML

Description

Add datepicker picker to field or to any other element.