Aalto University
School of Science

T-111.4360

Design of WWW Services

Final Phase

Topic

Study Schedule Planner

18 May 2014

Group 10

[redacted]

Table of Contents

1 Introduction and Design

1.1 Introduction

Planning of study schedules is a slow multi-step process that requires lots of time and patience from university students and academic staff. At Aalto University, the students' typical schedule planning process starts by searching the course information in the WebOodi and Noppa web services. After finding some suitable courses, the users then manually add the course information to their personal calendars. If the user notices that some courses have overlapping or in some other way problematic schedules, they then delete the overlapping courses from their calendars and go back to WebOodi and Noppa to check if some other useful courses are available. After finding some suitable courses, they will then add the new courses to their calendars again and check if the new courses overlap with some existing ones.

In addition to the previously mentioned user group (students), the teachers may also need a similar schedule planning features when they want to add their own lecture dates and times to their calendars.

We want to simplify the previously described process by providing a web service that allows the users to search course schedule information and view the schedules of all the selected courses in a calendar view on the same page. The users can easily see if the course schedules overlap in the calendar view. When the user is satisfied with the created schedule, he/she can then export the events to his/her personal calendar (Google Calendar, Apple iCal or some other calendar application that supports the popular iCalendar file format).

One similar competing study schedule planning service, www.lukkarit.fi, already exists. The main problem with the existing lukkarit.fi service is that it is a Chrome web browser plugin, which means that it can be only used with one web browser. This is a major problem because the Chrome web browser is not installed on the university IT center's computers, which means that the students and academic staff cannot use the typical classroom computers to design their schedules. We want to develop a web service that works with all the major web browsers (also the ones that can be used with the university classroom computers).

1.2 User Group Analysis

As the introduction section mentioned, the users of the service are all Aalto university students and university personnel. Thus two groups have been chosen as the main groups: **students** and **lecturers**.

Student users are concerned about their *own courses* and want to schedule them with a minimal overlapping. Typically users from this group own an external personal calendar application (such as iCal and Google Calendar), which is mainly used for study planning.

Lecturers are concerned about the *courses that they teach* and *adding the course events to their personal calendars easily.*

1.3 Scenarios

The following diagram shows the most important use case scenarios of the service. The arrows indicate which user groups are involved in certain scenarios. Detailed usage scenarios are listed below. The number of each item indicates its priority level.

Figure 1. The main usage scenarios and user groups.

1a. Lotta, 2th Year Student

"I have signed into courses 'Ohjelmoinnin perusteet Y' and 'Business Process Design and Implementation P'. I want to know the lecture dates and export them into my iCal."

1b. Mikko, 5th Year CSE student

"I want to plan my next period so that I can work 4 days per week and do the school stuff (lectures/exercises) within one day. I also want to get my planned exercises groups and lectures to my Google Calendar"

- Current period is II
- Doesn't remember which courses have teaching atm
- Wants to select exercises/lectures for specific week day and deadlines regardless of day
- Has already HOPS with remaining courses
    - "Information retrieval" III
    - "Web services P" I-II
    - "T-106.5600" - II
    - "T-76.5115" - II-IV

2. Minna, CSE Student

"I want to see where (in which lecture room) my today's lectures are?"

3. Maria, CSE Student

"I created my calendar few days ago. Now I'm going to drop one of my courses. I want to modify my existing calendar so that I can drop that course and import the new version to my Google Cal."

- Courses already selected:
    - Design of WWW Services
    - Web Software Development
    - Information Retrieval
    - User Interface Construction
- Course to drop:
    - Information Retrieval

4. Heikki, Former CSE Student

"I discussed with my colleagues about my former studies. I used this application to plan my calendars. I'd like to review my old schedules by using this service."

- Has created calendars for years 2011-2012, one for each period
- Wants to review his previous studies from III/2012

1.4 Main Features

Based on the scenarios, we identified the following main features that must be implemented to the service.

- Searching courses *(students, lecturers)*
- Adding courses to user's own courses *(students, lecturers)*
- Viewing course schedules in calendar view *(students, lectures)*
- Viewing a list of old courses *(students)*
- Exporting events to an external calendar *(students, lecturers)*

- Checking if course events overlap with other courses *(students)*

1.5 Site Structure

The site consists of several parts, which are shown in the diagram below. The login / info page, the terms of service and privacy policy page are publicly available for everyone and are meant to be used by all the users of the service, whereas the calendar planning functionality and settings require login.

All the pages have some common parts that are added to the pages dynamically, such as the navigation bar and the footer. The calendar planner page is the most dynamic page; almost all the contents are generated dynamically. Likewise, the settings page needs to check the user account information from the database to know which social media accounts are connected to the user account and which are not connected. The login/info page and the legal pages (terms of service and privacy policy) contain mostly static content.

The **login / info page** contains allows the user to log in to the service. The page also contains material about how the service can be used (site tour that showcases the most important features of the web service). *(students, lecturers)*

The service contains a page that shows **terms of service** and **privacy policy**. This page is needed because a service that collects information about its users is legally required to have a privacy policy and a terms of service in Finland. *(students, lecturers)*

The **calendar planner** page is implemented as a single page application, in which the content is refreshed dynamically. The page contains the following functionality:

- **Course search**, which allows the user the search for courses. *(students: searching for courses, lecturers: searching for courses)*
- **Course selector**, which contains checkboxes for selecting which course events (exercise groups, lectures) should be shown in the calendar. *(students, lecturers)*
- **Calendar view**, which allows the user to view course events (lectures, exercise sessions, assignment deadlines, midterm exams and final exams) in calendar. *(students, lecturers)*
- **Calendar exporting** feature, which allows the user to export the calendar in iCalendar format to Google Calendar, Apple iCal or some other similar application. *(students, lecturers)*
- It is also possible to view a list of **old courses**. *(students)*

Settings page contains settings for choosing which social media accounts the user wants to use to login to the service.

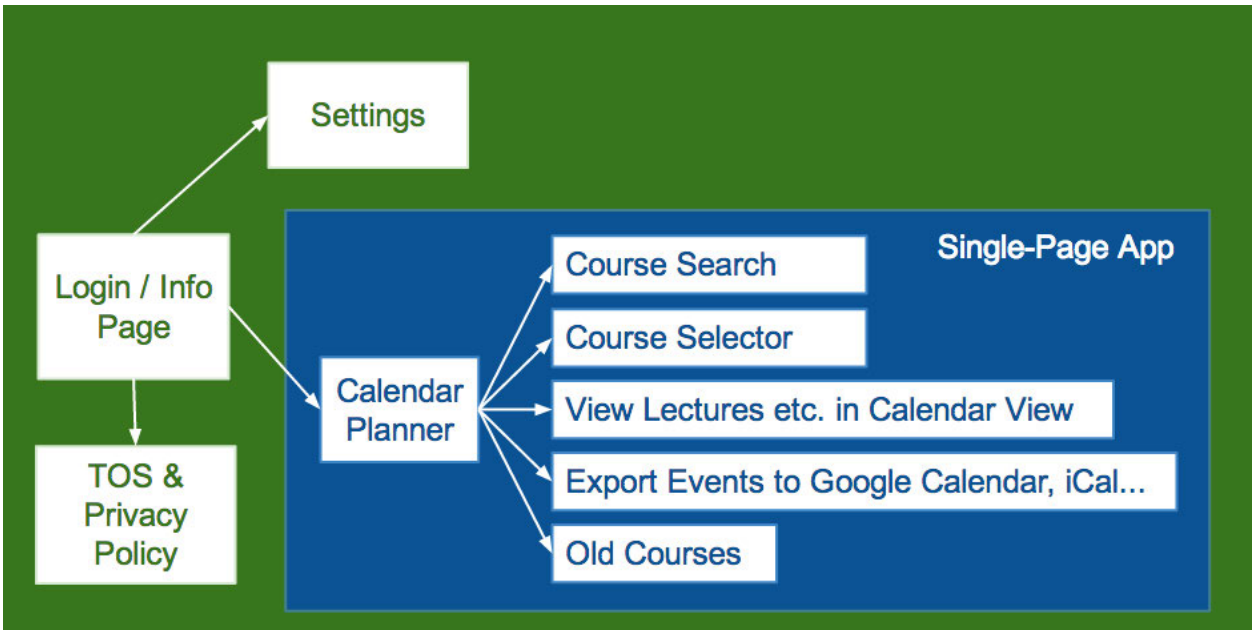


Figure 2. Site structure.

URLs for each page:

Login/info	/
Calendar planner	app/
Settings	settings/
Privacy policy	legal/privacy-policy/
Terms of service	legal/terms-of-service

1.6 Reflection

The implemented application follows the design plan quite well. Some features were dropped due to lack of time: information about deadlines, study modules and course material (lecture slides etc.) were not developed.

2 Content

2.1 Content Production

Login / Register / Info page

An introductory page showcasing functionality offered by the application. This consists of images that tell about the web service.

User groups: students, lecturers.
Reference material: we will use the calendar planner page as reference material when creating the site tour and help content.
Producer of the material: we. In a real web service, we would also use the help of a usability expert, a graphic designer and an art director.
Copyrights: we.
Modification: we. In a real web service, we would also use the help of a usability expert, a graphic designer and an art director.
Time dimension and dynamics: The help content etc. must be checked if any changes are made to the calendar planning functionality.
Languages: The page was only implemented in English due to lack of time. If we happened to have more time, we would have translated the web app to Finnish and Swedish.

Calendar Planner Page

Help tooltips will aid the user depending on what functionality the user is using. The tooltips appear when the user places the mouse over an item that may need some clarification; for example, when the user places the mouse over the name of the calendar application that they want to use for opening the exported calendar, a help tooltip is shown for describing how the user can add the calendar feed to Google Calendar or Apple iCal.

The course information is retrieved from Noppa API.

User groups: students, lecturers. More detailed information about how each user group uses the calendar planner is given in section 1.5.
Reference material: Noppa API for the course information, Aalto study guides for the study modules.
Producer of the material: course information is produced by Aalto University employees. We create the help content.

Copyrights: Aalto University owns the rights to the course information (the content is licenced with the Noppa API's open data license). We reserve rights to the user history content and help content.
Modification: Aalto University employees modify the content that is provided via the Noppa API. We have to input the information about the study modules because the Noppa API does not provide that information.
Time dimension and dynamics: the content is mostly useful when the courses are arranged and before the courses are arranged, but some users may also want to view a history of their previously taken courses.
Languages: The page was only implemented in English due to lack of time. If we happened to have more time, we would have translated the web app to Finnish and Swedish.

Terms of Use and Privacy Policy Page

The terms of use and privacy policy statements. When creating this page, we used the Finnish legislation as the main reference.

User groups: students, lecturers.
Reference material: Finnish legislation. (www.finlex.fi)
Producer of the material: we; in a real service it would be a lawyer.
Copyrights: we.
Modification: we, in a real service the content would be modified by a lawyer.
Time dimension and dynamics: The content may need to be modified if some changes are made to the calendar planning functionality or to the Noppa API open data license. Changes to the Finnish legislation may also affect the validity of the privacy policy and the terms of service.
Languages: The page was only implemented in English due to lack of time. If we happened to have more time, we would have translated the web app to Finnish and Swedish.

Settings Page

User groups: students, lecturers.
Reference material: Python Social Auth documentation.
Producer of the material: we.
Copyrights: we.
Modification: we.
Time dimension and dynamics: The content may need to be modified if Facebook, Twitter or Google change their social login features.

Languages: The page was only implemented in English due to lack of time. If we happened to have more time, we would have translated the web app to Finnish and Swedish.

2.2 Content Update Process

We will create the help content ourselves. The help content and tutorials may need to be updated after some parts of the calendar planner single-page application are modified. If the service happened to be a real commercial service, the help content would be written by a usability specialist and the tutorials would be designed by a usability specialist, a graphic designer and an art director. The required tools for the content updates include image and video editing programs and programming tools that support the Django templating language.

The terms of service and the privacy policy may need to be updated if the Finnish legislation changes. In this demo application, we would just update all the legal content ourselves, but if this happened to be a real web service, we should probably ask a lawyer to review all the legal content before it is published. The required tools for the content updates include programming tools that support the Django templating language. If the content happened to be written by a lawyer, he/she would most likely write the updated legal content with a word processor and the web site maintainers would convert the content to Django templates.

The course information comes from the Noppa API and that content is updated by people who maintain the Noppa API. To enable the users to search courses based on in which study module they belong, we will need to input the information about study modules to the database because the Noppa API does not provide information about study modules. The information about study modules should be updated annually after the new study guides are published on the university website. The required tools for updating the study module information include tools that support adding data to the database. To get the latest information from the Noppa API, we just need to make normal HTTP GET requests to retrieve the latest information from the API.

We will update the settings page ourselves. The page may need to be updated if Facebook, Twitter or Google change their social login features.

2.3 Reflection

The login/info page showcases the functionality of the application in the form of text and screenshots. We did not have enough time to complete short videos but the text and the screenshots wound up being informative enough.

The terms of service and privacy policy documents were somewhat easy to create as intended in the design phase, as there were drafts to be found for such purposes. Further development would need to be done with a lawyer. The documents are reasonably small and divided into subtopics so that reading the documents is possible for the target users. Overall we think that the content satisfies our initial plans well.

3 User Interface and Interaction

3.1 Interaction

When new user enters the service, he/she must use a social media account to login. Because the majority of the users are students, it is highly probable that the user has a social media account, thus the SSO speeds up the registration process. If this web app was an official Aalto University web service, we would use the Aalto Shibboleth login instead of social media login functionality.

In an earlier version of the service we forced the user to create a new user account with traditional username/password authentication. However, we realized that it was a bad idea because it was required that the user would memorize a username/password combination. A typical user of this web service uses the service only five times during per year (once before each teaching period), which means that many users will forget their usernames and passwords because they use the service so rarely. Some users might also use weak and easy-to-guess password, which would weaken the security of the service. Therefore, we decided that it is best to allow the users to use their existing social media accounts or Aalto accounts, which they use more often.

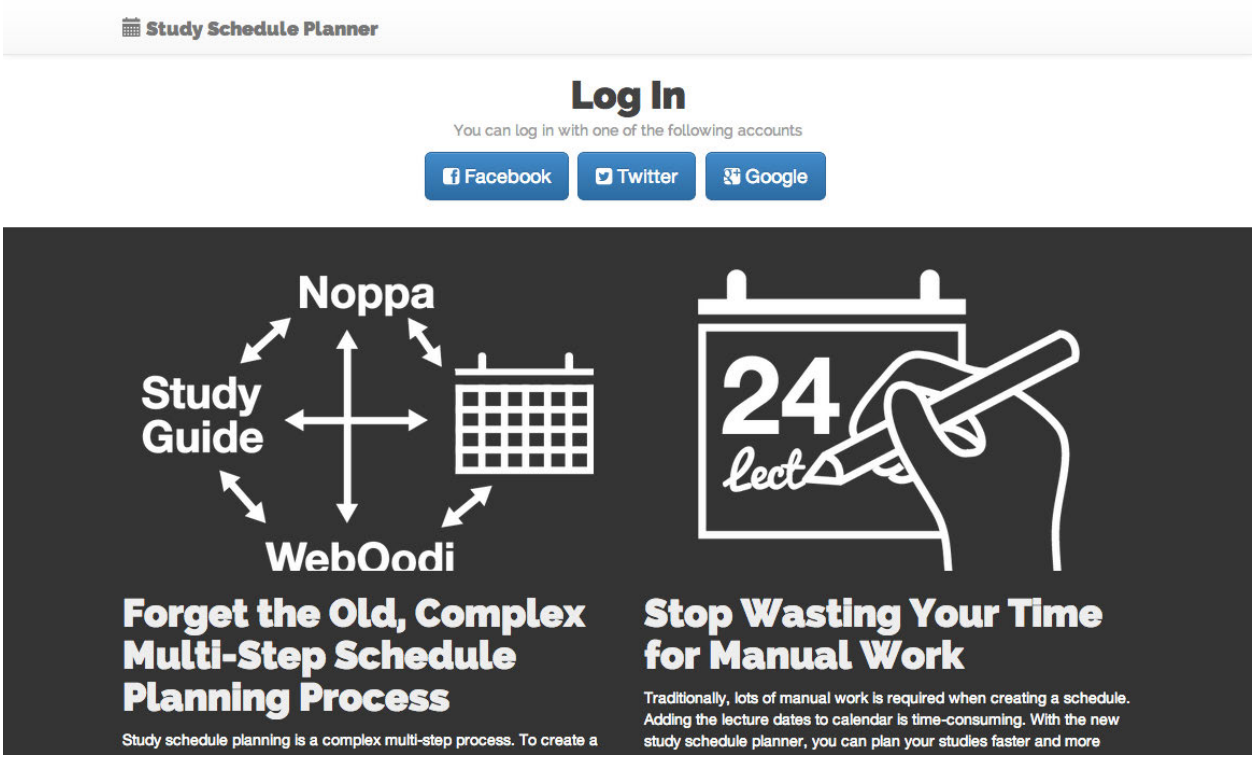


Figure 3. The top of the login/info page contains buttons for signing in with social media accounts and information about why the traditional study schedule planning process is difficult and complicated.

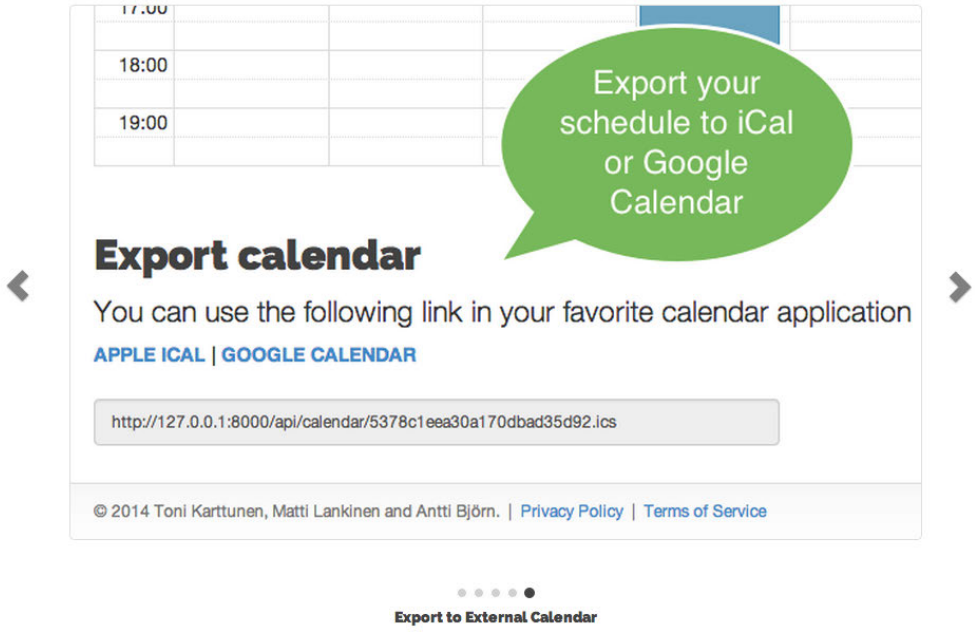


Figure 4. The bottom of the info/login page contains help content in an image carousel, which explains how the users can user use the service to plan their studies.

When the user has logged in, the scheduling view is displayed. In our opinion, service is ill-designed if it requires multiple pages for the calendar planning process. Thus, the scheduler is a single-page app that contains all necessary components for study planning:

1. User searches the course he/she wants to add to his/her calendar
2. User selects which exercises/lectures he/she wants to take from courses (calendar is updated in real time when selections are being made)
3. User exports/syncs the calendar into his/her personal calendar application

Because the calendar is updated immediately after course selections, the same view can be used for both teachers and students.

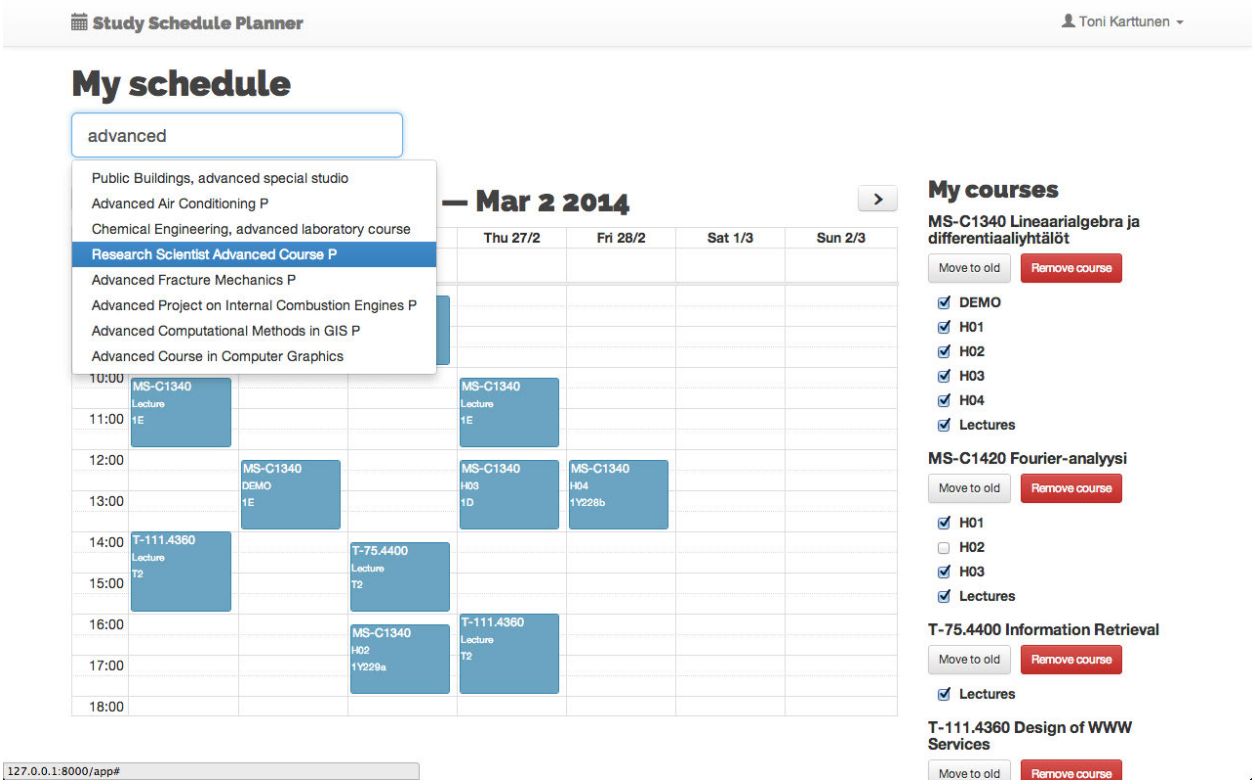


Figure 4. Calendar planner (single-page application).

There might also be cases when the users want to view their history (old courses). This can be done by clicking the “My Old courses” button, which opens the dialog listing all old courses for the user.

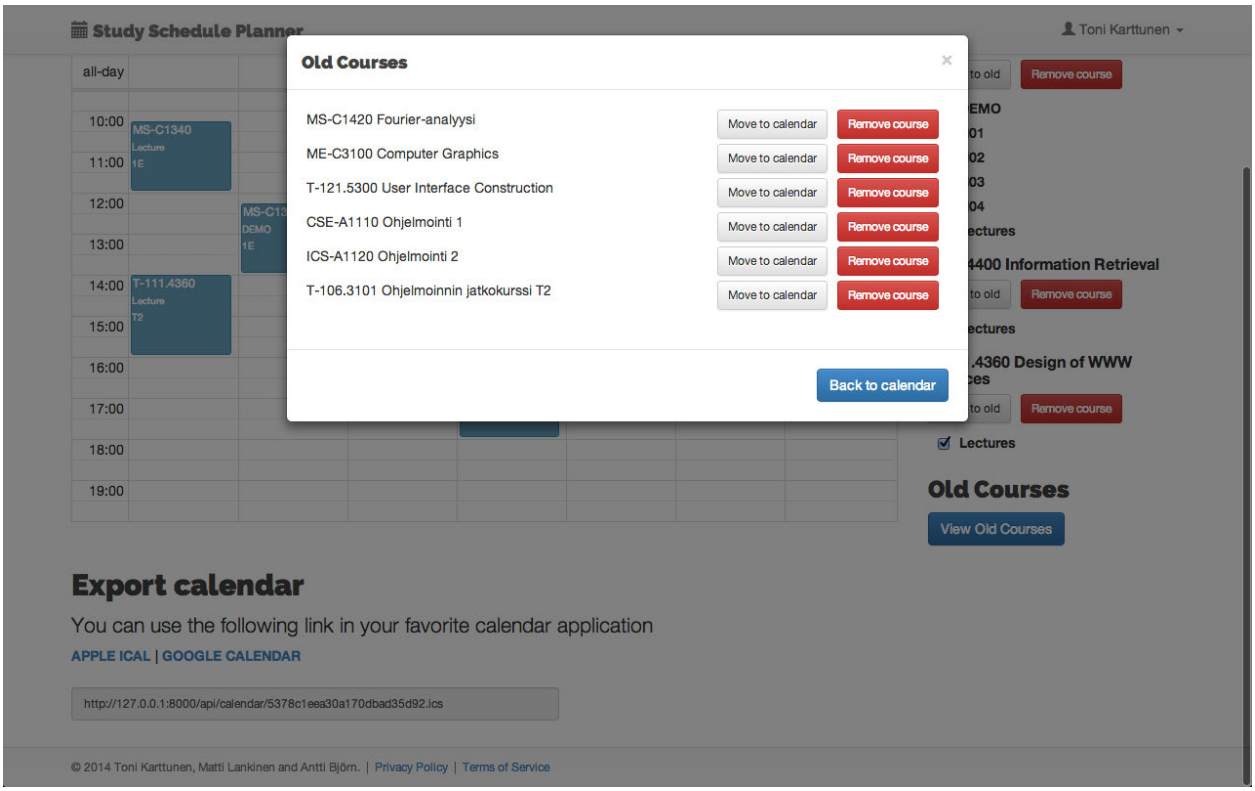


Figure 5. A list of old courses is displayed as an overlay on top of the calendar planner. The user can use the *Move to calendar* button to move old courses back to the calendar and the *Remove course* button to delete the course from his/her courses forever.

3.2 User Interface

We used Twitter Bootstrap (http://getbootstrap.com) for our visual style and graphical elements. We also created some own custom CSS rules with the LESS stylesheet language for those parts of the user interface when the Bootstrap user interface controls did not suit our needs. Twitter Bootstrap is a modern and widely used front-end framework with good support. Because the service is targeted at a wide range of university students and teachers who most likely have varying tastes for the visual style of web services, we used a neutral and simple visual style for the service (mostly white and black with some accent colors, e.g. blue, red and green). We used mostly sans-serif fonts (Raleway, Helvetica, Arial etc.), which are easy to read on computer displays. If this web service happened to be an official university-sponsored study schedule planning service, we should follow the university's visual style guidelines.

The calendar element is in a centric part in our user interface so (regardless of the user group) it must be extremely easy-to-use and similar to the known implementations (such as Google Calendar / iCal). Thus, we used FullCalendar (http://arshaw.com/fullcalendar).

Because the students and lecturers usually use desktop or laptop computers to design their schedules, we did not use responsive design for the calendar planning page. Implementing the calendar view for mobile devices' small screens would have been challenging. However, we used mobile-friendly responsive design for every other part of the web service (login/info page, settings and legal content); some users may visit the web service with their mobile device if they see an advertisement of the service somewhere at the university campus or hear about it from their friends.

Screenshots of Each Page

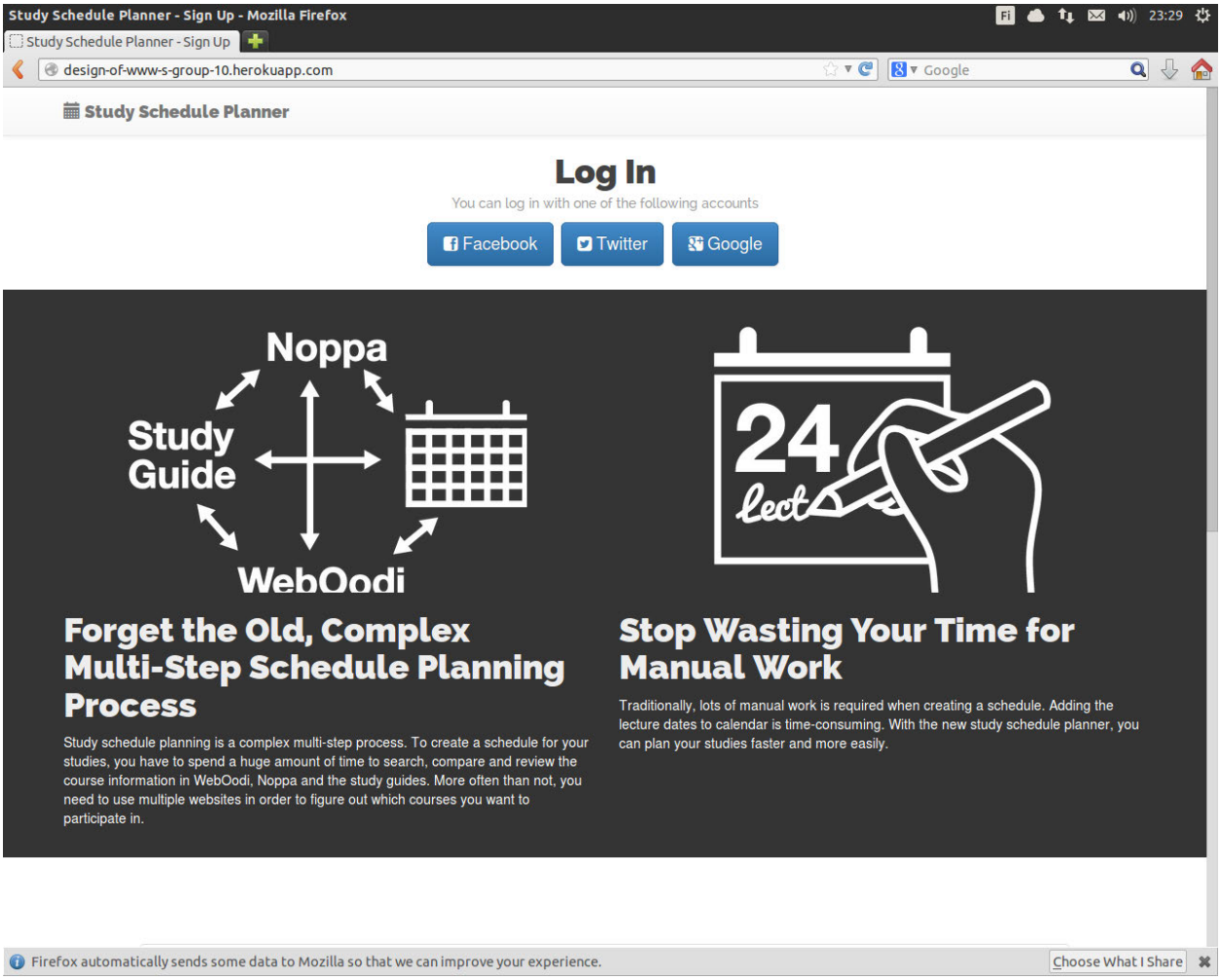


Figure 6. Login/info page in Mozilla Firefox on Linux.

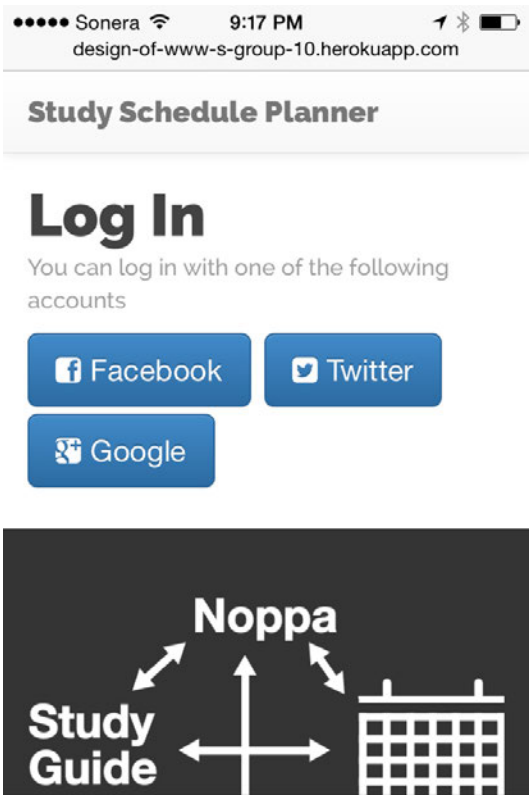


Figure 7. Login/info page on iPhone (iOS 7).

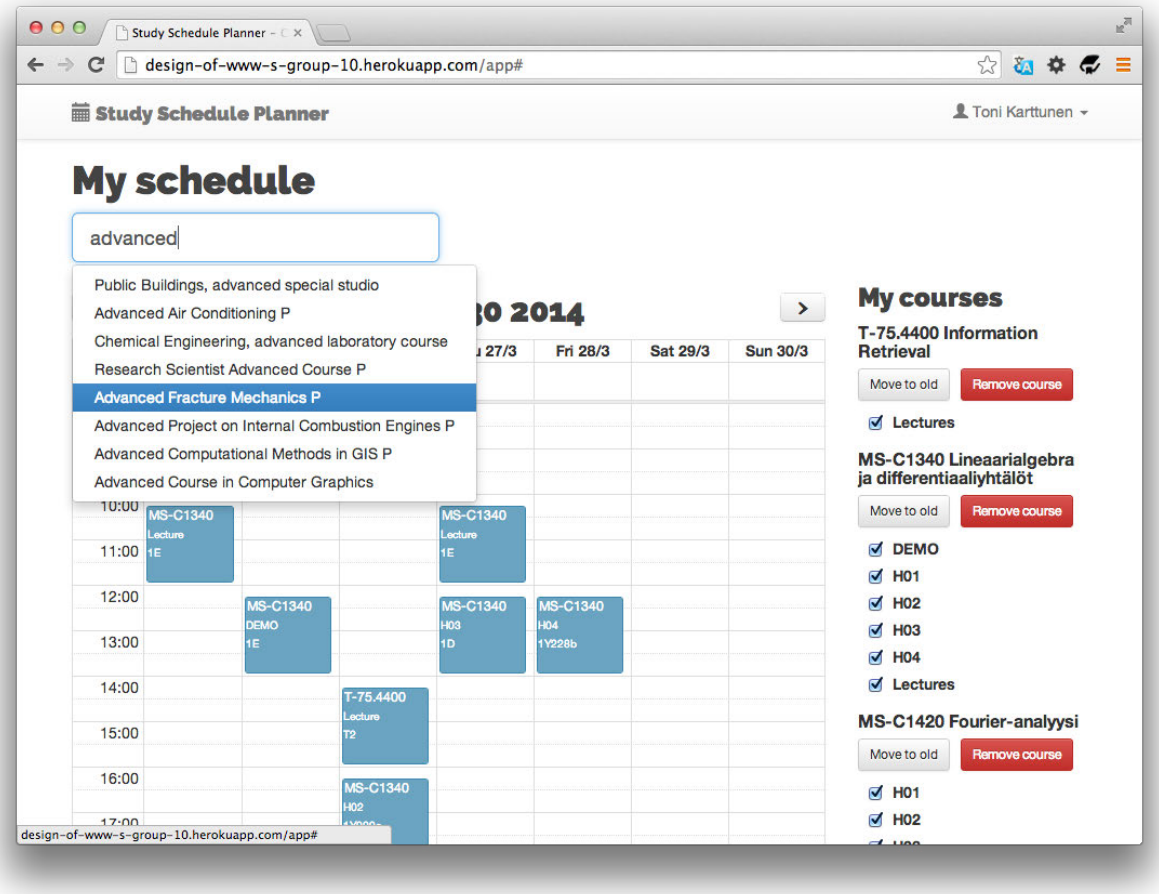


Figure 8. Calendar planning page in Google Chrome (OS X Mavericks).

Figure 9. Calendar planning page on iPhone (iOS 7).

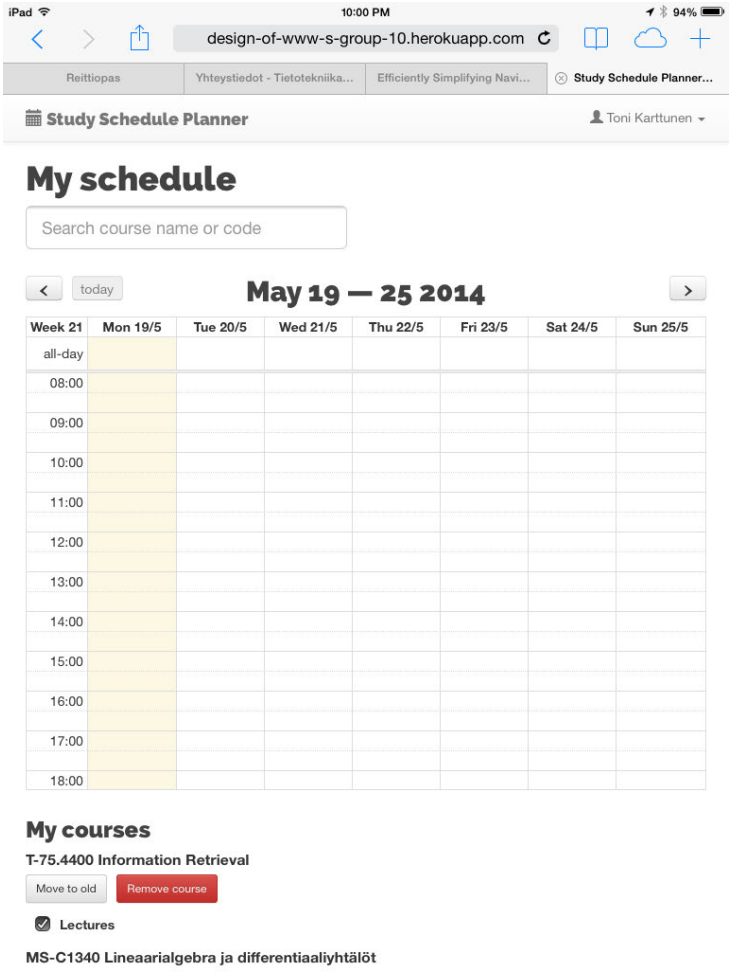


Figure 10. Calendar planning page on iPad (iOS 7).

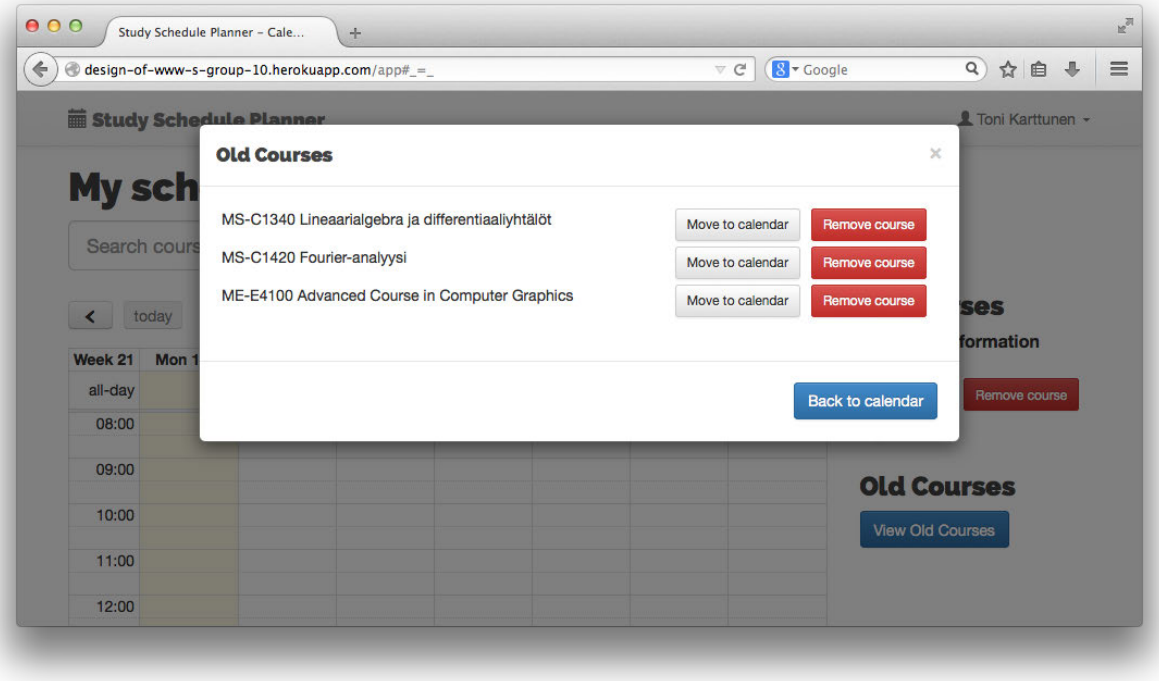


Figure 11. Old courses view in Firefox (OS X Mavericks).

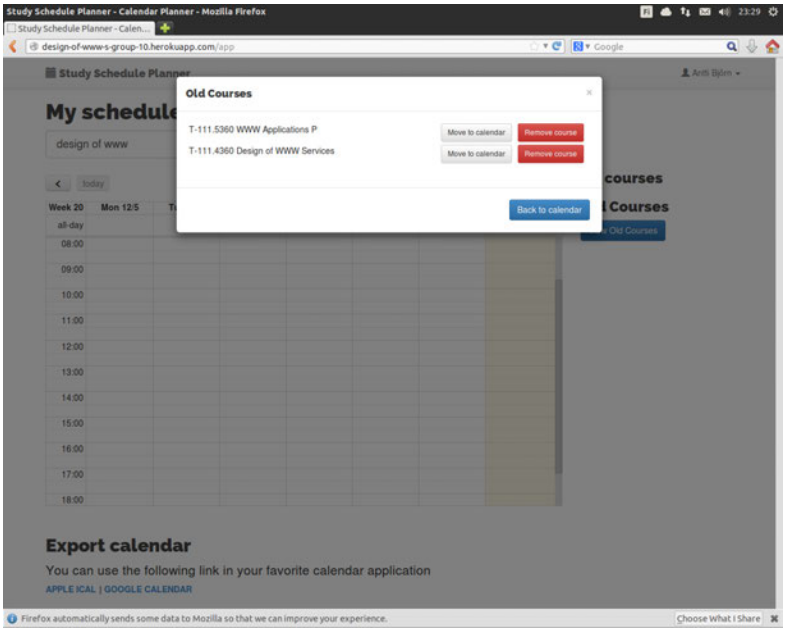


Figure 12. Old courses view in Firefox (Linux).

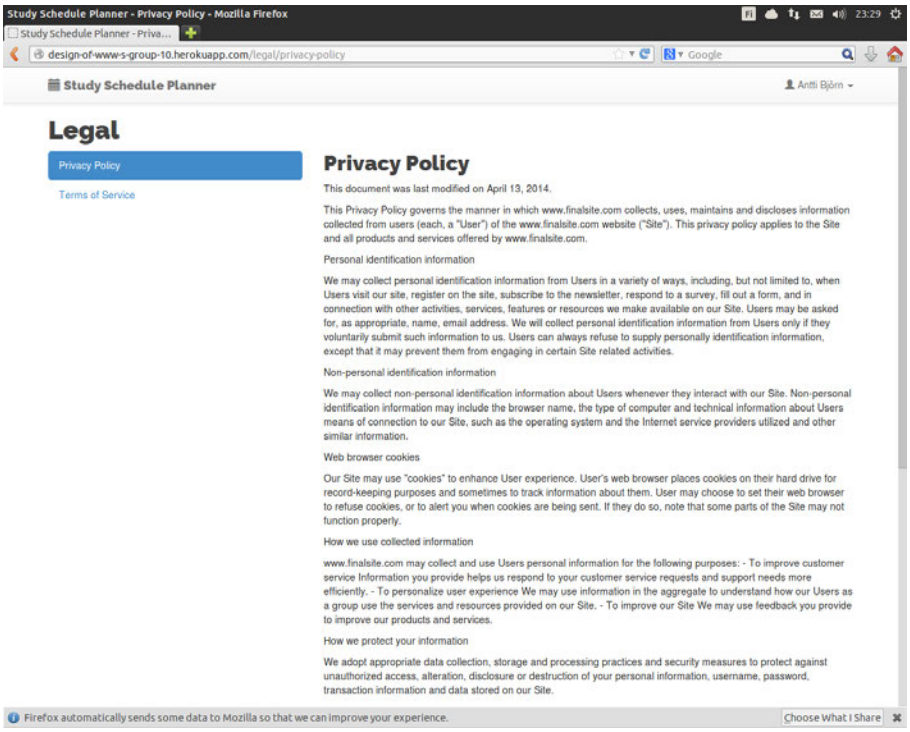


Figure 13. Privacy policy page in Firefox (Linux).

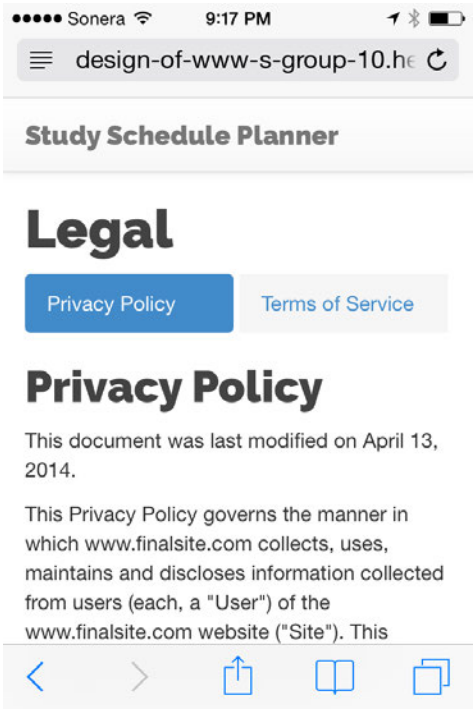


Figure 13. Privacy policy page on iPhone (iOS 7).

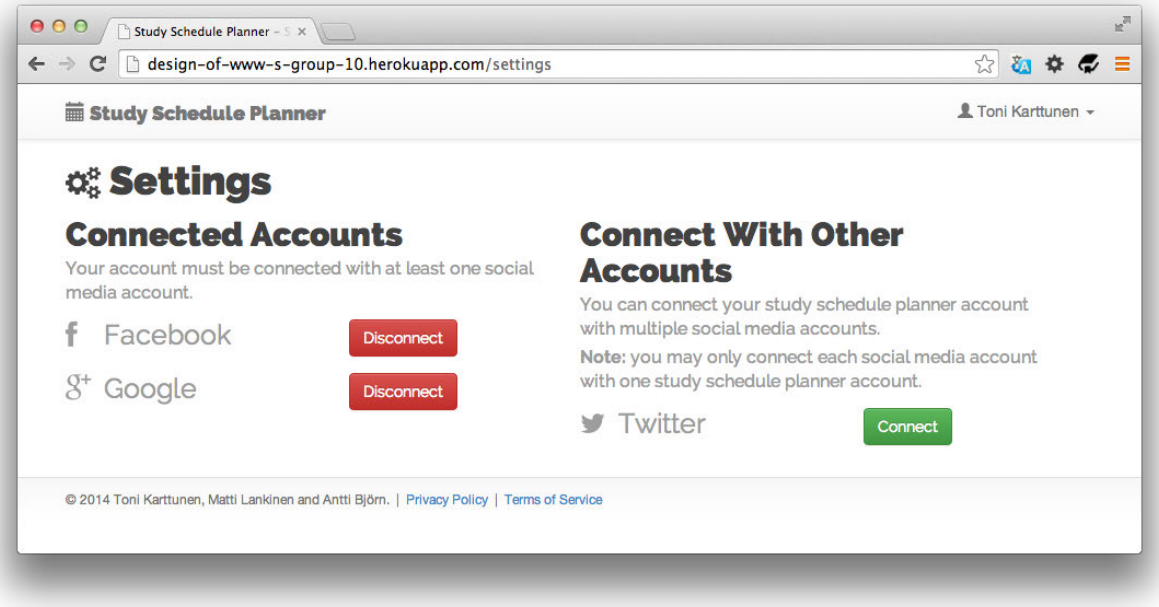


Figure 14. Settings page in Google Chrome (OS X Mavericks).

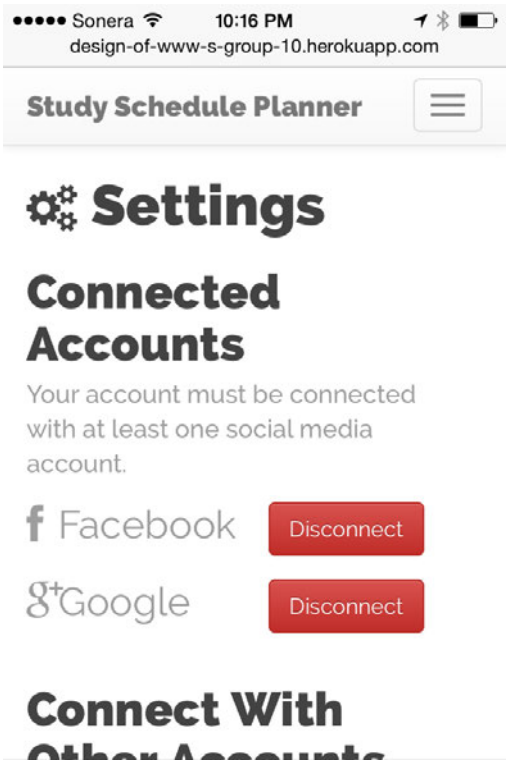


Figure 15. Settings page on iPhone (iOS 7).

3.3 Reflection

The login/info page looks different from the original design because we decided to remove the new user account creation (with username and password authentication) feature. The final design is better because it is simpler than the original design. In the mockups, we planned to have some video tutorials on the login/info page but we did not have time to create videos. However, we believe that the screenshots that we used on the login/info page explain how to use the service very well. It should be noted, too, that the calendar planning single-page application is not overly complex to use and therefore it is not necessary to create a huge amount of help content.

The initial plan for the calendar planning page contained information about course material (lecture slides etc.), study modules and deadlines. We did not have enough time to implement those features. Otherwise, the calendar planning page is very similar when compared with the early mockups.

The old courses view look exactly the same in the final version as in the early mockups.

The final version of the legal content pages (privacy policy and terms of service) looks very similar to the early drafts of those pages.

The final version of the settings page looks completely different from earlier versions because we decided to remove the basic username/password authentication. Therefore, the settings page does not contain an option for changing the password. We believe that the final design is better because it is much simpler from end user’s point of view.

We think that the final version of the web service works well for both main user groups. It can be challenging to design a user interface that suits well for two user groups (e.g. Noppa has different user interface for students and lecturers), but we succeeded quite well in designing a user interface that can be used by two user groups.

4 Architecture and Technologies

4.1 Architecture

Our service is a standard HTML5 application that can be run with all modern web browsers with JavaScript enabled (Chrome, Firefox, IE 9+). We are using “thin server, thick client” design pattern. This means that our backend is just a data storage with lightweight JSON REST API and the actual logic is done with JavaScript in the client. Backend contains also integration to Noppa API and Google calendar. The back end uses Model-View-Template design pattern, which is very similar to the popular Model-View-Controller design pattern (essentially, Django’s views correspond to the controllers in traditional MVC and Django’s templates correspond to the views in traditional MVC).

The following figure displays the high-level architecture of our service:

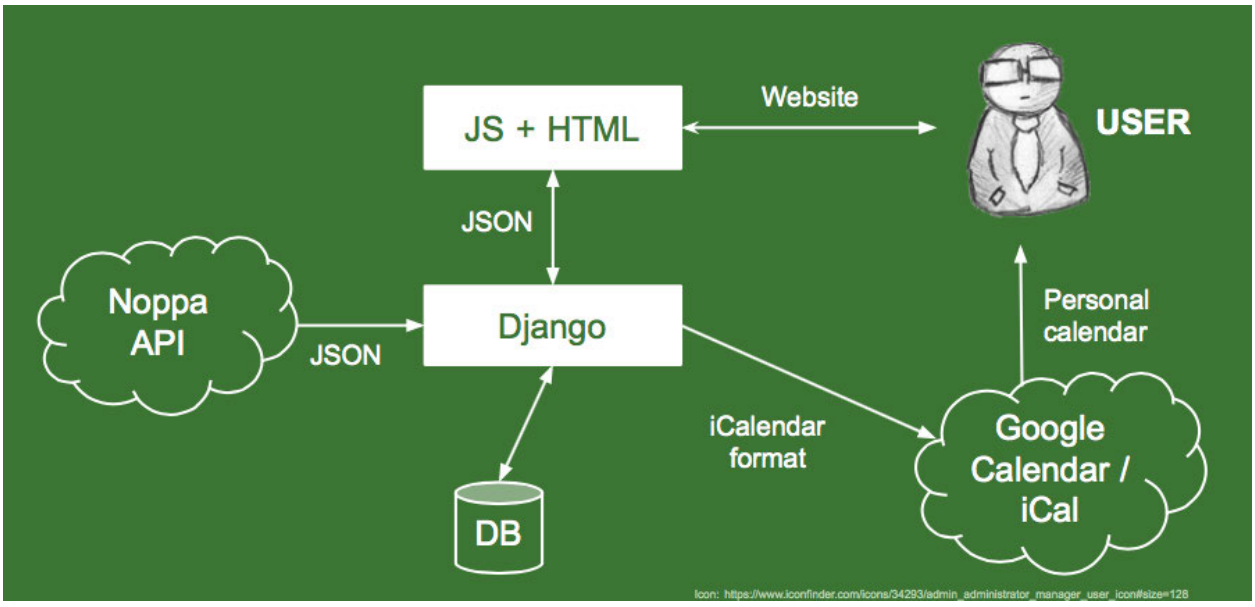


Figure 16. High-level view of the web application architecture.

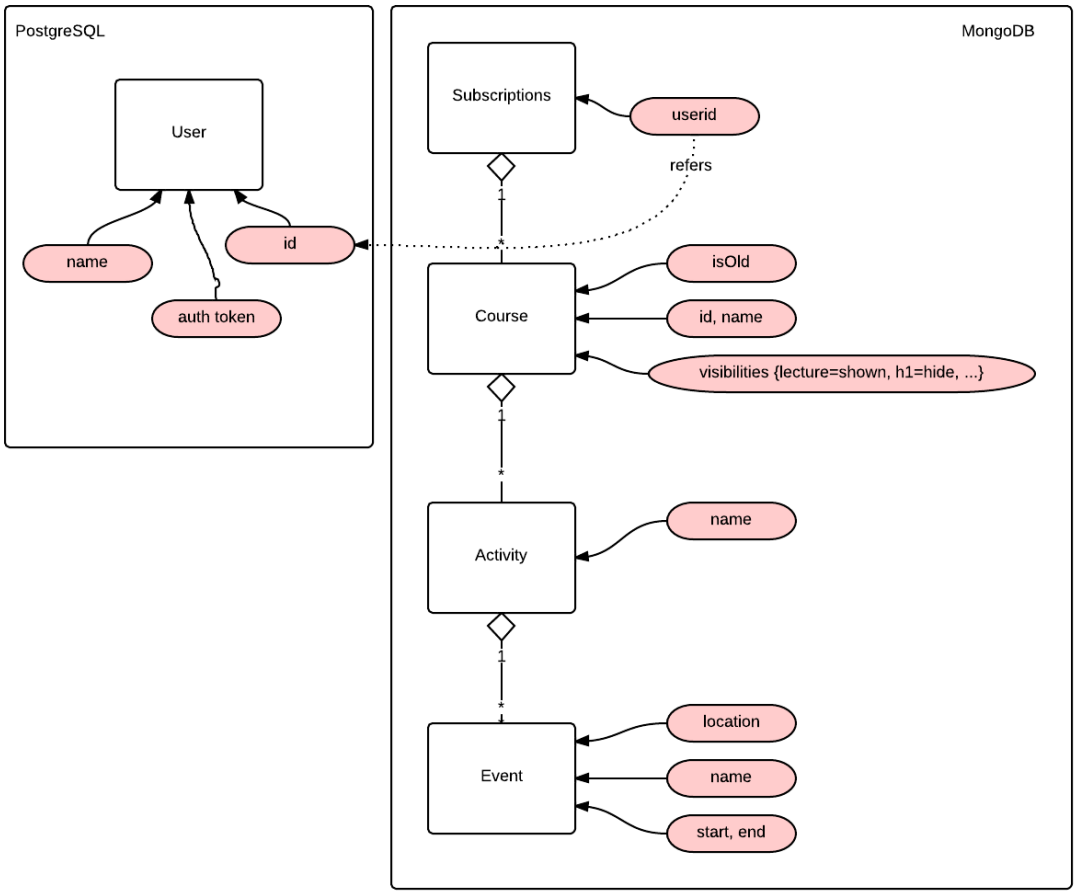
We use Django for our backend. For database engine, we are using PostgreSQL for user account data storing and MongoDB for calendar data storing.

4.2 Database

For database engine, we are using PostgreSQL for user account data storing and MongoDB for calendar data storing. The reason for choosing two databases is that

Django authentication support PostgreSQL natively. However, the course data is much easier and efficient to store as a mongo record.

The following diagram demonstrates the database structure of our application. The PostgreSQL tables are simplified (for our use case only) because they are generated automatically by Python Social Auth library.



In general, each user has one “subscription” record in the Mongo database. This subscription can contain many courses (either marked as “old” or “new”) and these courses can contain several activities (lectures, exercises…). Each activity can contain from *0* to *n* events (with start/end times and optional location). When user requests events or courses, they are fetched from this record. Because we use Mongo’s aggregation framework for the whole operation, it is very fast.

4.3 Performance

We do not believe that the service will face any difficult performance issues because the user base is relatively small (only the Aalto University students and faculty) and the responsibilities of the backend are small. Some usage spikes may occur at the near beginning of each teaching period when many users are planning their schedules. The operations in the client does not contain much calculation so tablets and even mobile phones can handle the service.

Our target was the client load-time under 1 sec (+ network latency) which was successfully achieved. We didn't do any automated performance tests but browser debugging tools show that the backend operations took normally under 200ms to process. Good performance comes mainly from MongoDB native aggregation framework usage: almost all data processing is done inside the database engine and results are only converted to the JSON in the application server.

Because we are using the Heroku's free node as our "production" server, its performance is poor compared to our development environments. Heroku also puts the free nodes to hibernate after certain period of inactivity: when requesting the service first time, it might take up to 5 minutes for the service to start. However, this could be fixed easily by buying the proper backend nodes.

Our main concern is the availability of the Noppa API, because as active users of the Noppa web service we know that it often has maintenance breaks that often cause some downtime and other problems. The Google calendar integration is done by using generated iCal feeds which are polled by Google, so the downtime in Google services doesn't affect the usage of our service.

4.4 Security

We try to mitigate the threat of security risks by building the service from widely used and well tested open source components, such as the Django framework and the Apache web server. If this web service was developed as a commercial project or as an official university study schedule planning tool, the connection should use SSL encryption but it's not possible in the scope of this course.

Our data model ensures that there is no sensitive information about users - we don't store any password but use OAuth and OpenID instead: currently supported providers are Facebook, Google and Twitter. If this application came to the real use, we'd also integrate Shibboleth for Aalto University students.

4.5 HTML & CSS

The HTML of the website follows the guidelines of HTML5 standard. There are few errors coming from `<form>` tags because validators can't detect the JavaScript usage and custom functionality we built for form inputs. There are also some errors generated by FullCalendar plugin. Most of the HTML is generated dynamically based on user data.

Because we are using Twitter Bootstrap as our style base, the CSS is not valid: the Bootstrap itself contains several hundreds of errors[1]. However, we've tried to keep our own CSS as clean as possible. We use LESS stylesheet language which only compiles to CSS if the stylesheet`s syntax is correct.

4.6 Browser Dependencies

We are not using any browser dependent parts in our service. All front-end code is plain vanilla JavaScript, CSS and HTML. JavaScript must be turned on in order to use the service (all modern browsers support JavaScript by default nowadays).

The login/info page, the settings page, the privacy policy page and the terms of service page are optimized for mobile devices. However, the user experience of the calendar page is not very good on mobile devices due to high amount of information combined with small device's screen.

4.7 Technologies and Installation

Backend Software Requirements Summary

Table 1. Backend software requirements.

Name	Version	Usage	Usefulness
Python	2.7.5	Backend language	Pretty good for quick software development
Django	1.6.x	Web framework	A little bit heavyweight for simple REST API, could be replaced with e.g. Flask
icalendar	3.6.2	iCal feed export	Very useful for iCal feed generation, saved many hours of work

[1] https://github.com/twbs/bootstrap/issues/6398

Python social auth	0.1.23	OAuth and OpenID provider integration	Very useful and easy to use. Also secure compared to own implementations.
MongoDB	2.4.8	Course data storage	Easy-to-use and high performance. Can be used for enterprise applications too.
SQLite	3.x.x	Development user data	Very useful for development
PostgreSQL	9.x.x	Production user data	Not very useful: hard to install and administrate. Could be replaced with MySQL. We chose this database because Heroku provides free hosting for PostgreSQL and Django ORM supports it well.

Some libraries that are mentioned in table 1 depend on other pieces of software. All the dependencies are listed in the requirements.txt file. All the application dependencies can be downloaded with pip (https://pypi.python.org/pypi/pip) by using the project requirements.txt file:

```
$ pip install -r requirements.txt
```

Databases must be installed by using OS dependent installation tools (Linux: apt-get, OSX: homebrew or macports, Windows: installers from websites).

Front-End Software Components Summary

Table 2. Front-end software requirements.

Name	**Version**	**Usage**	**Usefulness**
jQuery	1.10.2	DOM manipulation	Mandatory
Lodash	2.5.1	Utilities, and collection manipulation	Not mandatory but very useful and provides nice functional programming interface. Could be replaced with underscore
require.js	2.1.11	Dependency management	Not mandatory but helped a lot to maintain the codebase

			and to split features into smaller modules
FullCalendar	1.6	Calendar rendering	Very useful and easy-to-use. Saved many hours
moment.js	2.x.x	Datetime handling	Very useful
Twitter Boostrap	3.x.x	Base styles	Not mandatory but very useful. Provides a wide set of UI components and nice markup.
Handlebars	1.3	Template management	Not used at all, could be useful for bigger projects but not in this one
LESS	1.5.0	Style sheet generation	Modern CSS, this should be used for every project. Could be replaced with SASS

All frontend components can be found from project sources under `static/vendor` folder.

Running the Web Server

After installing all the required code libraries, frameworks and database, you can run the web server on your computer by running the following commands in Terminal:

```
# Create the SQL database
$ python manage.py syncdb

# Start MongoDB
$ mongod

# Run the application
$ python manage.py runserver
```

To open the web application, navigate to http://127.0.0.1:8000 with a web browser.

Accessing an Online Version of the Web Service

You can access an online version of the web service at http://design-of-www-s-group-10.herokuapp.com/.

4.8 Dynamic Functionality

The most important “dynamic functionality” in our service is the user’s calendar view, which is be unique for all users, according to the selections the user makes. The following sequence explains the user actions and results when creating unique calendar.

1. User signs in to the service first time. In this time, there is no previous content for the user in our database so the calendar remains empty. The first time usage is identified and tutorial is displayed to the user.
2. User starts to search courses. This triggers an JSONP call to Noppa API with the typed keywords. Noppa API returns courses matching the keyword, the user interface displays the matches to that user can select the wanted course by clicking it with mouse.
3. When user clicks the course, it will be added to “user’s courses” list. Technically this means that the client tells backend that user has selected a course. Backend saves this information and fetches the course data from Noppa API. The client then asks the content from backend. The fetched course data is rendered in the calendar view by using JavaScript
4. When user closes the browser, the course data still remains in our server so that when the user signs in next time, his/her data can be restored from our backend and the calendar view can be displayed based on the previous selections.

Each course data is stored per user and only when user has requested the course.

4.9 Reflection

We are very satisfied to the results. Everything went as planned and there were no drawbacks in the implementation phase. We used git as our version control: all features were developed in separate branches and merged to the master via pull requests. This ensured that the codebase remained clean and coherent.

The only thing we would done different would be using only single database engine. Because there was so little time for this project, we had to make compromises in order to get all things done - resulting the usage of two database engines.

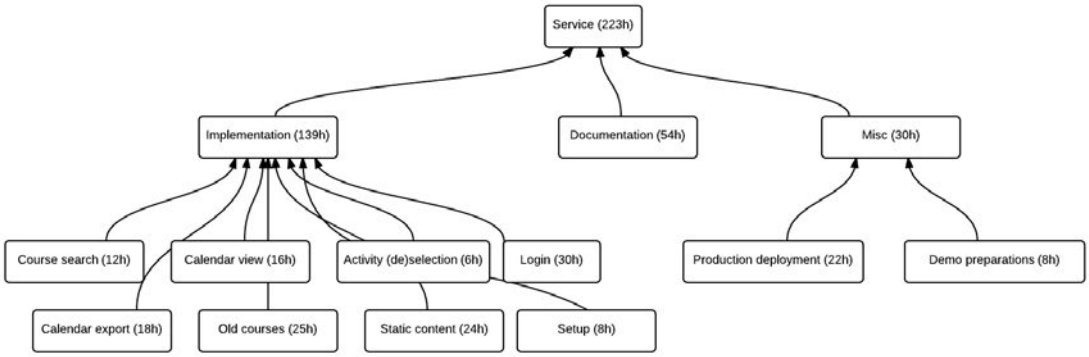
5 Design of the Project

5.1 Work Breakdown Structure

The WBS of our service contains three main task: implementation, documentation and misc. The major workload comes from the implementation task which is divided into smaller subtasks. Each subtask is an independent feature. There is dependencies between subtasks but they are not shown in this graph. The dependencies are displayed in time schedule where those tasks are put into the timeline (dependencies first).

Documentation task contains the time for course documentation writing and preparation. Misc contains sub-tasks that are neither implementation nor documentation but that must be done before the course is done.

We didn't want to assign specific tasks to specific team members. However, we have given areas of responsibilities: [redacted] = backend, [redacted] = frontend, [redacted] = documentation. This doesn't mean that everyone takes only tasks for his responsibility area but just ensures that the things get done before the deadline.



5.2 Time Schedule

The following diagram displays the planned time schedule. The tasks are named identically with our WBS chart.

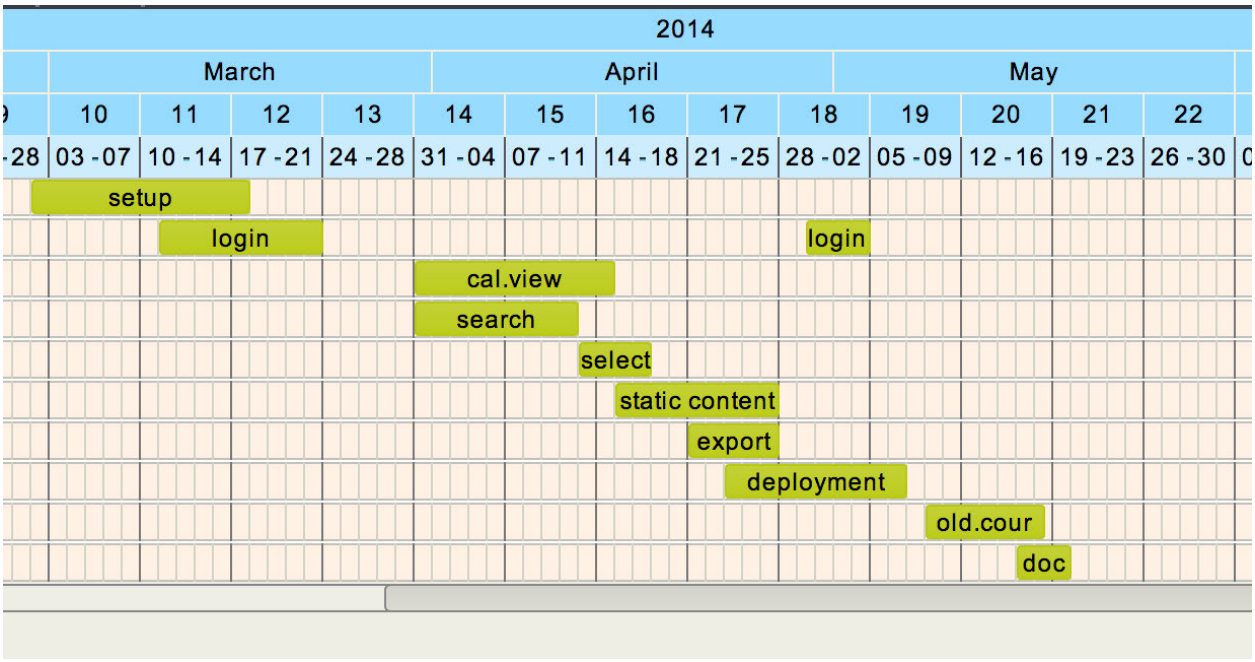


Figure 6. Gantt chart.

5.3 Use of Time

We used Google spreadsheet for our time logging. The hours spent were the following:

Table 1. Time consumption in hours.

[redacted]	115
[redacted]	126
[redacted]	118

It was possible to produce the planned features and documentation in a timely manner. However, because multiple persons were usually involved to same tasks, we didn't record the time per task per person. The overall spent times per tasks can be found from the updated WBS chart.

5.4 Reflection

The work breakdown structure was good in retrospect, the tasks were atomic enough so that dividing them didn't bring extra challenges. The time schedule did lag behind a bit during the late development but not much.

6 Final Reflection

In our opinion, the design process went smoothly and we are happy with the end results. We believe that it was good that our early drafts of the web service were already very well planned and therefore the implementation was quite straightforward. Some initially planned features had to be dropped due to lack of time during the implementation phase.

We think there could be a real userbase for this service and currently our implementation can satisfy their needs. The only thing we would do differently in the implementation phase would be to to use a single database engine as discussed in section 4.9.

What makes us most proud is the simplicity and elegance in the user interface. We spent a lot of time planning the user experience and the simple workflow. We believe that the user interface is so simple that most users can use it without any guidance (although we created some help content to make it clear how the web service can be used and why the service is useful for the user groups.

We did not have significant project management issues as Github was a good platform for development. Our development process required that all new features had to be merged to the master branch by using pull requests. This ensured that the code was always reviewed by at least one person, thus making the codebase coherent and clean.

The most important thing that we learned during the course was to understand the role of different user groups and how to design user interfaces that work well for multiple user groups. We also gained some experience in documenting the planning process of a user interface from early mock ups and user stories to a finalized user interface.