Final Document: Kyytipalvelu

T-111.4360: Design of WWW services

Phase: Final Pha e
Group number: 16

[redacted]

1. Introduction and design

1.1 Introduction

Have you ever looked at cars on the road and wondered: "So many empty seats in every car going to the same direction. Why couldn't they use the same vehicle"? A car driving from town to town with all seats taken is a rarity. Usually there are from one to four vacant seats that could be used by other people that have the same trip.

Our project for this course is a service that helps drivers get passengers for their vacant seats and passengers to find cars with vacant seats. The main functionalities include authentication, offering a route, searching for vacant seats on a route and reserving a seat. Our routing functionality utilizes an API offered by Google maps. User profiles include preferences for trips and there is a referral system for tackling trust issues.

Similar services exist, such as kyydit.net in Finland. However, sharing rides in Finland is not as popular as it is in some other European countries, and e.g. France has quite a bit more advanced services for sharing rides (e.g. http://www.covoiturage.fr/), including preferences of different characteristics of the trip, such as chatting or pets, and also the possibility to ask for a fee for the trip. In other European countries the pay for a ride has often settled to a little bit under the price a train or other public transport would charge for that trip. In Finland, where trips by train are notoriously expensive - and often late, a good service for sharing rides is needed.

This documents sections 1. and 2. are based on the design document which was done earlier on the course. In those sections new areas are marked with blue text as suggested by the course staff.

1.2 User group analysis

The main user group for our service includes young (from 18 to 35 year old) Finns. The group is further divided to students, students who own a car, regular travelers, and users on holiday, and users interested in public events such as music festivals.

Below you will find justifications for each of our user-group choices.

- **Young people**
  - Youngsters are our main user group since they travel more and are generally more prone to new experiences, such as getting to a car with a stranger. There are also more youngsters without a car and thus in need of a lift than there are middle-aged and elderly people.

- **Finns**
    - The service includes Finnish cities and cities close by.
- **Students**
    - Students rather often move away from their hometown to study and thus take trips to their hometown more often than an average person.
- **Students who own a car**
    - Students who own a car form a separate user group, since it is highly relevant to have seat providers, so the service has to appeal to this group.
- **Regular travelers**
    - Travelers who travel regularly due to reasons such as relationships or work might require different aspects of the service than one-time users.
- **People on vacation**
    - People on vacation might want to take detours to see the scenery or stay in a city for a few days and continue with a different driver.
- **People interested in festivals**
    - Our service will be most valuable during events targeted at young people such as music festivals due to numerous possible users sharing a destination.

Other possible relevant user properties include whether people have a car, a holiday, a work trip, pets, a specific taste of music or a long trip or want to chat during the trip, take the scenery route, or are motivated to use the service in order to save money, nature or to have company on a trip.

1.3 Scenarios and use cases

1.3.1 User Scenarios

In this chapter user scenarios for two personas are presented.

Mikko
Autumn semester is almost over and Mikko starts to plan his trip to visit his parents in Tampere for christmas. His last exam is 20.12 and he wants to leave as soon as possible, but the trains are rather expensive for him. He opens our page and searches for a ride, filling in the date and destination of the trip. He is taken to a page with offers of three rides, one on 21st at 11:00 am, second on 22nd at 3:00 pm and third on 27th of December at 5:00 pm.

He checks information on the first two trips, as the third one would be too late. The driver of the first offer is talkative, but he has a dog on board. As Mikko is allergic to dogs, he chooses the ride that leaves 22.12. He clicks "order" and is prompted to register. He registers with his email account~~, fills in some personal characteristics that he wants the driver to see~~ and books the ride, suggesting that he would be picked up at 3:00pm from Pasila's railway station.

After a few days Mikko receives an email that the driver has confirmed his ride. He has saved quite a bit by not taking the train, and possibly he'll have a fun chat on the trip.

Teemu
Teemu is planning a trip to the Provinssirock festival in Seinäjoki 27.6.-28.6. He opens our page and wants to offer a ride for 3 festive people in his car. Teemu wants to be able to pick the people his riding with, so that the trip to the festival will be part of the party. First he ~~registers~~ logs in with his Google account and leaves information that he is 18-years-old student from Oulu, he has a nice Toyota car and he loves to party.

Then he fills in a form for sharing a ride. Teemu wants to travel from Oulu to Seinäjoki at 27.6. starting at 08 a.m. In an open text message for potential passengers he types "a trip to the festival to party!" He sends and confirms the form and is directed to his account page. He wants to ride back from Seinäjoki at 29.6. starting at 11 a.m. He fills, sends and confirms another form for his second ride offer. He gets directed to his account page, where he can review his open offers. He closes the application and starts waiting for responses.

After a few days Teemu gets ~~a message~~ messages from two people that they ~~that two people~~ have booked a ride with him. ~~He can now see the profiles of these people. and decide whether to confirm the ride or not.~~ They seem like fun travelling companions, so Teemu ~~confirms the rides and~~ is happily planning further his upcoming festival experience.

1.3.2 Use cases

Our service has two major use cases: booking a ride and offering a ride, which are presented in this chapter. To give a better picture of our service, a use case diagram for booking a ride is provided in the picture 1.

Use case: Book a ride
Actors: User and the service
Brief: The user searches for a ride and reserves a single seat from the service.
Steps:

1. The user selects the search for a ride option
2. The user has to fill a route information form
   - Start and end locations of the route are required
   - Time and date ~~are optional~~
   - ~~(Possibility to choose start and end locations from a interactive map)~~
   - Additional result filtering is optional
3. The user sends the form to the service
4. The service validates the form and searches for matching ride offers

    - If the form is missing required information the user has to provide the missing information before advancing
    - If there are no matches the user has to try again with different route information
5. The service shows a list of matching ride offers
    - There must be available seats for a ride offer to shown on the list
6. The user browses the ride offers and selects one
7. The service shows the full information of the ride offer
8. The user has to login before the seat reservation option will be available
    - If the user is not a member of the service a sign in flow is activated
9. The user confirms the seat reservation
10. The service saves the seat reservation and sends a notification to the user and to the owner
    - ~~(The owner has an option to reject the seat reservation)~~
11. The user logs out

Use case: Offer a ride
Actors: User and the service
Brief: The user fills a route information form and sends it for the service. The service will process the form and saves it for listing.
Steps:

1. The user has to login before the ride offer option will be available
    - If the user is not a member of the service a sign in flow is activated
2. The user selects the offering option
3. The user has to fill a ride information form
    - Start and end locations of the route are required
    - Time and date are required (Possibility to utilize a pop-up calendar)
    - ~~(Possibility to choose start and end locations from a interactive map)~~
4. The user sends the form to the service
5. The service validates the form
    - If the form is missing required information the user has to provide the missing information before advancing
6. The user has to confirm the ride information
7. The service saves the ride information and sends a notification to the user
8. The user logs out

1.4 Main features

The main features of our service include authentication, offering a ride, searching for a ride and user profiles. This chapter elaborates further on each of these features.

- Authentication
    - Registration with name and email.
    - Log in, sign out

    - Session handling
    - Google login
- Offer a ride
    - Offering a ride includes filling the following information:
        - Point of departure
        - Destination
        - Free space
        - Time and date
        - A text field for open comments
- Search for a ride
    - Searching for a ride includes filling the following information:
        - Point of departure
        - Destination
        - Contact message and additional information for the offerer
        - Preference of filtering of search results
- User profiles
    - The user page includes personal information and booked and offered rides.

1.5 Structure of the site

The site is designed to work as a single-page application where the user stays on a single page. The content of the page is changed dynamically according to the current state of the application. In this final documentation a page may refer to a widget inside a page that can be changed dynamically.

Overall structure of the site is represented in the site structure diagram and the functionality of the each individual page are described below:

1.5.1 Landing page

Landing page is the front page of the site containing links to different parts of the service. The page itself has big links to both searching for a ride and offering a ride. The site has a header which has all necessary links for navigation: home page (By clicking the title), offer a ride and search for a ride. Account page link is shown if a user is logged in, otherwise a login page link is shown instead. The user can register from the login page.

1.5.2 Account page

Account page will be available after the user has registered and logged in to the site. On the account page the user can view his personal information. Also all the user's current and past reservations and offers are listed on the page.

1.5.3 Login page

On the login page the user can login to the service. After logging in the user is redirected to the account page and placing reservations or leaving new offers is made available. The login credentials are stored; there will be no need to login before the credentials expire again. If the user doesn't have an account there is an option to register for the service. This will redirect the user to the register subpage, which contains a form for registration.

The user must provide his user name, email address and user password on the register subpage in order to create an account. After the provided information is validated on the server the user can confirm the account creation. This redirects the user to the account page.

The user can choose to log in and register with his google account.

1.5.4 Search page

The search functionality is divided in multiple subpages where the user can view and reserve rides. On the search page the user must provide either the source or the destination address or both of the addresses in order to search for matching ride offers. The user can optionally choose to filter the results by distance, time, free seats or seats in total. The search results will be listed below the search parameters and the results may be divided on multiple pages.

Selecting a ride from the list dynamically shows more information on the ride. On the same page there is a button for reserving a seat from the ride. This will be active only when the user is logged in and at least one seat is available. After reserving the seat there is notification that the reservation has succeeded.

1.5.5 Offer page

On the offer page the user can leave ride offers for others. On this page the user must provide the source and destination addresses of the ride, departure date, the number of available seats and optionally some text to be seen by the possible future passenger.

When all the required information is provided and validated on the server the user can advance to the confirmation subpage where the user can review and confirm the offer. If the user is not logged in then the user must login before confirming the offer. After successfully leaving the offer there is a notification.

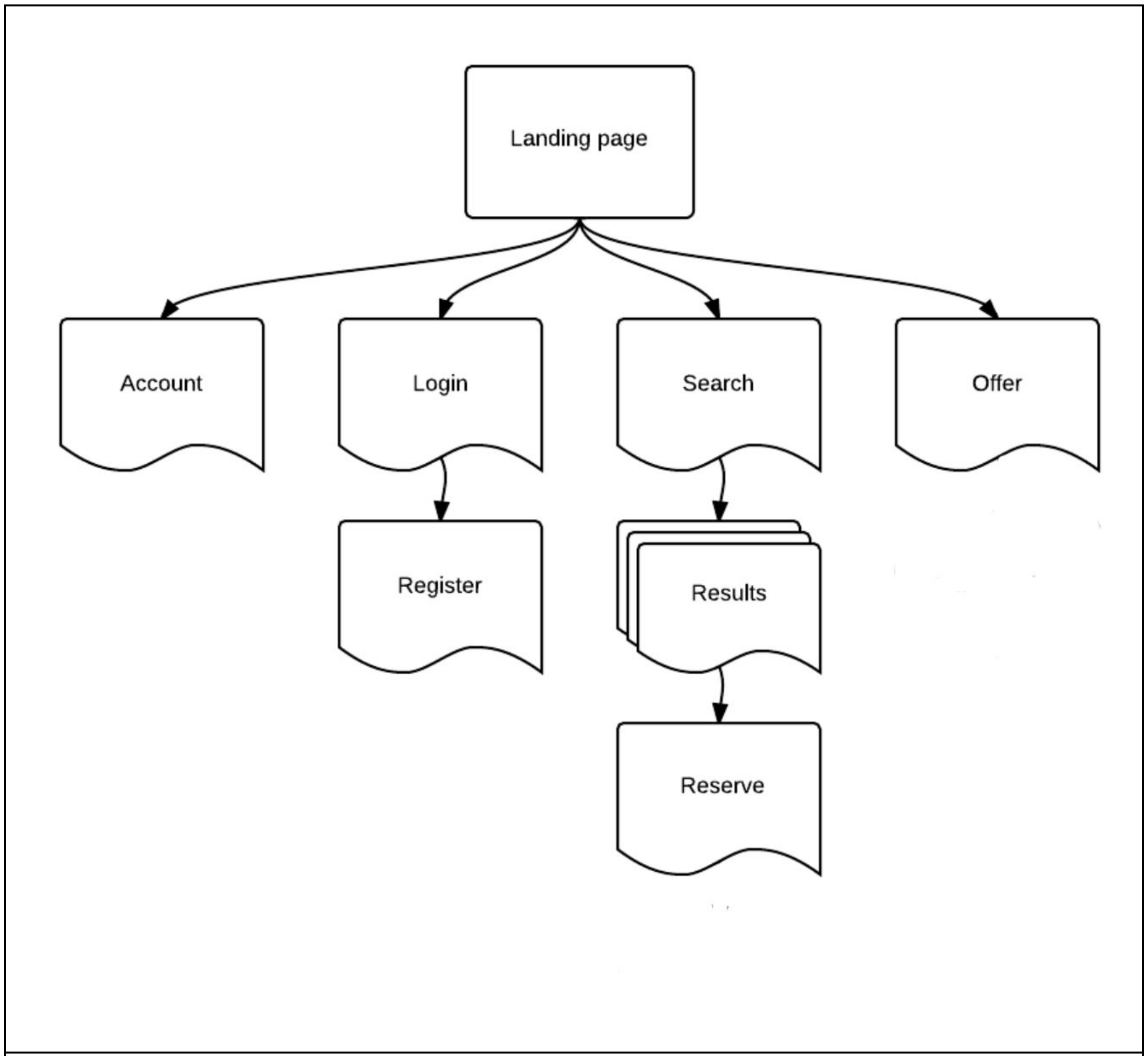


Diagram 1: A site structure diagram

1.6 Reflection

User groups, scenarios, main functionalities or the structure of the site didn't change that much. Some features weren't realized, mainly to keep the page simple and because of lack of time. For example, we didn't do quick search button for the landing page, because it was recommended by the assistant that the landing page should either have the big links to search/offer or the quick search, preferably not both.

There is no ability to change user information, because we decided we don't need the actual name of users, and we shouldn't ask for it if we don't need it. That left us with username, email address and password. The email address is used to log in, so we think it isn’t necessary to be able to change any of them. A possibility to change the password should be made possible in the future.

Additional offer information is shown dynamically by expanding the offer element and showing the information and the order button. This is a lot smoother than showing an additional page. We added Google login to make it easier to use our application.

Complicated communication between the one offering the ride and the searcher was left fully to their responsibility, we only provide the email address. This communication wasn’t of high priority even if it was in our user stories, as this way the application itself is rather light and users can use it more flexibly.

2. Content

2.1 Content production and update

Our web service depends heavily on the user-generated content. Most of the content is generated when the users leave their ride offers or registers for the service. This means that there will be almost no content when service is launched.

To solve the lack of the content in the launch we might need to create content by our self for the launch. This can be done by recruiting people to leave ride offers before the service is started and maybe even paying them in advance for leaving ride offers. People can be encouraged to leave offers through social media campaigning on Twitter and Facebook and in the launch by offering a price to a random user who has offered a ride. We could use Twitter tags to encourage users to post about their experiences using the service, like posting a picture with the person you’re riding with and tagging it with a certain tag. Creating a feeling of community through social media for the users would be useful for the service.
There also exists a closed group of 24000 users in facebook called kimppakyyti, which could be used to reach our potential users.

After launching the service the content production should rapidly increase by the number of active user the services has. Especially user that are offering rides are important for content production.

Besides the content created by the users there are geographical content created by third party APIs. Geographical data is needed for validating the source and destination addresses. Also the search algorithm may need the geographical coordinates of the addresses and route coordinates of the path from the source address to the destination address. The geographical data is fetched on the client side using third party APIs and the data is then sent to the server where it is stored in the database.

There is a notion of terms of service and contact information in the footer of the site.

Offer page

Content	Geographical data	User content
Reference material	None	None
Producer of the content	Google Maps API	User
Copyrights of the producer	https://developers.google.com/maps/terms	
Modifications	None	None
Time dimension	Infinite	When account is removed
Languages	English/Finnish	Finnish

Search page

Content	Geographical data
Reference material	None
Producer of the content	Google Maps API
Copyrights of the producer	https://developers.google.com/maps/terms
Modifications	None
Time dimension	Session duration
Languages	English/Finnish

2.1 Reflection

We didn't have much content planned during the design phase, because the content is mostly generated by users. However, it's important to think how to get users to create content for the site and get them interested in it. Social media campaigning could be a good way to interact with our main user group, young and active people. Creating interaction between the users outside the site can also bring new users to the site.

3. User Interface and interaction

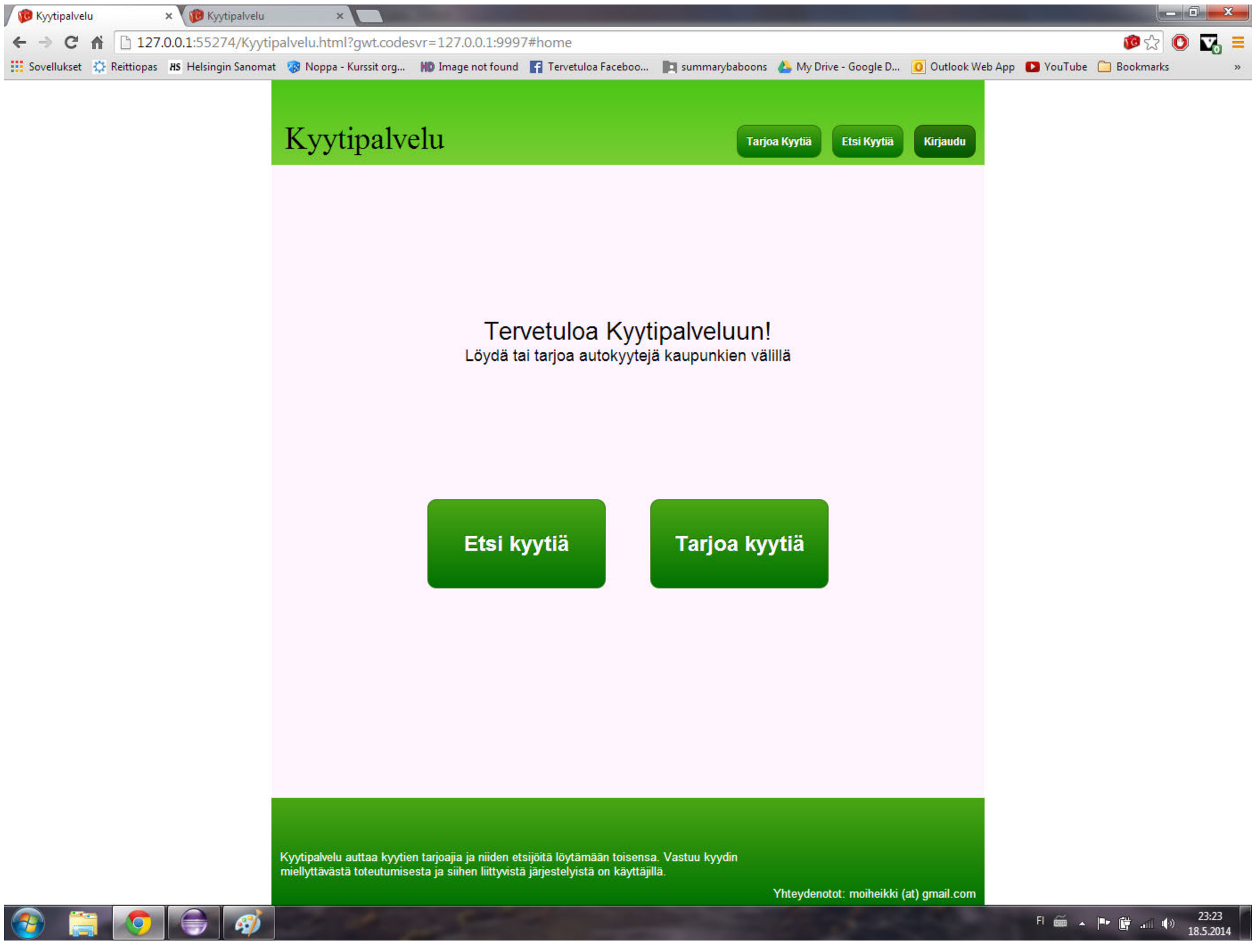
3.1 User Interface

In the feedback of our design document you requested more justification for design decisions in the layout. The user interface was designed to be as simple as possible to take care of the core tasks: offering a ride and searching for one. That’s why it consists of a header with navigation, a footer and very simple content elements. Even the amount of header buttons was minimized by putting registration flow inside the login page instead of it’s own button in the navigation bar, and homepage link works by clicking the title. The home page does not have a quick-search option in order to enhance usability by not having many ways to reach the same goal, as suggested by the course assistant.

As the application works through generating javascript, we decided to use as much css styles as possible instead of images to optimize performance issues.

Green color was chosen to remind people of the nature-friendliness of sharing rides. White background was chosen for simplicity and lightness. Gradient was added to look a bit more modern. All necessary tasks were put into containers so that the user intuitively knows they belong together.

In picture 1 you can see the home page. The middle of the page has buttons for the core functions of the service: offering a ride and searching for a ride. The header has a navigation bar that also includes a login button. Clicking the title would result in the user getting to this page. If the user clicks “kirjaudu”, he’s taken to login page, which is shown in picture 2.



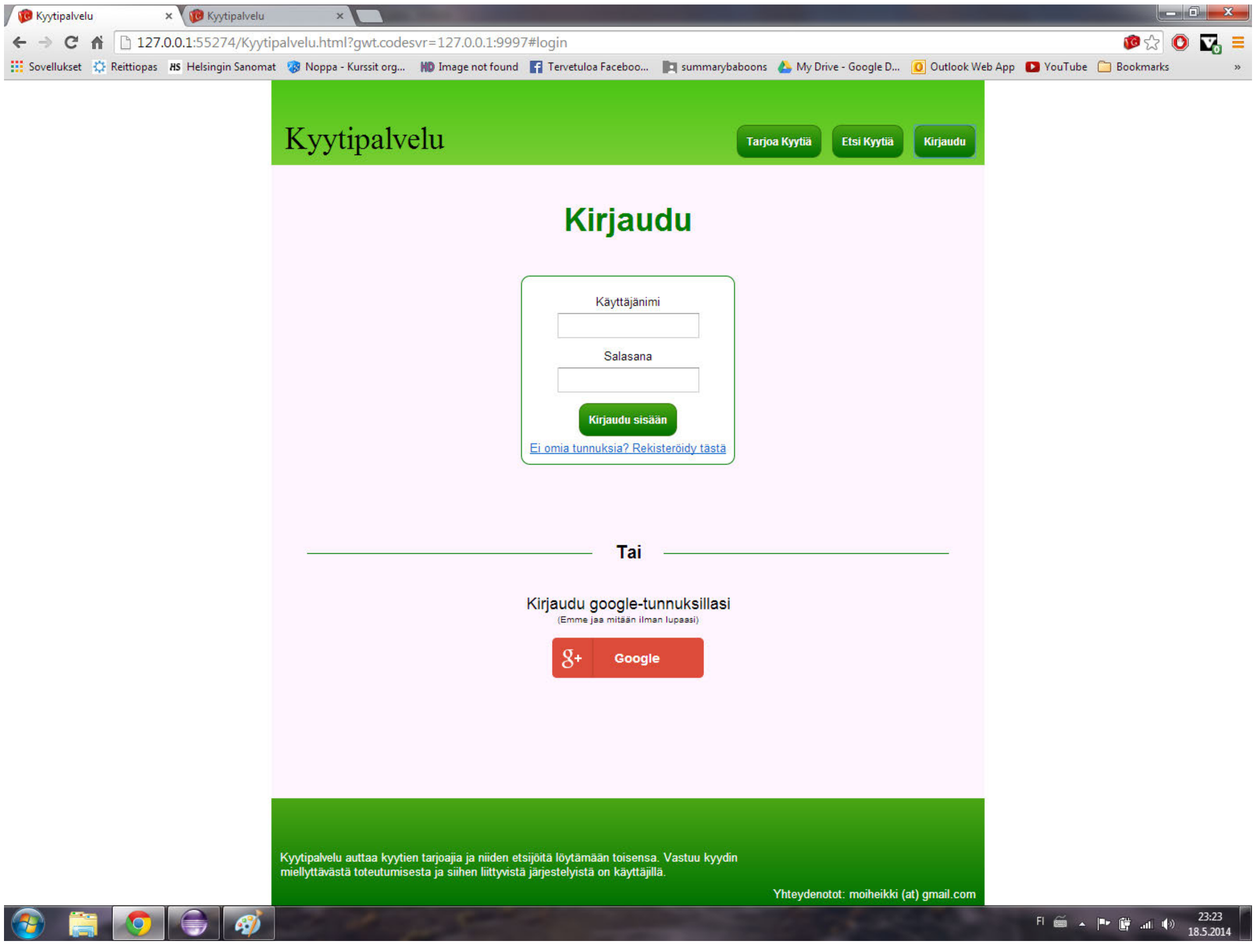
Tervetuloa Kyytipalveluun!
Löydä tai tarjoa autokyytejä kaupunkien välillä

Kyytipalvelu auttaa kyytien tarjoajia ja niiden etsijöitä löytämään toisensa. Vastuu kyydin miellyttävästä toteutumisesta ja siihen liittyvistä järjestelyistä on käyttäjillä.

Yhteydenotot: moiheikki (at) gmail.com

Picture 1: Homepage

In picture 2 you can see how the content area of the page changed into login page. The user is required to add his username and password in order to log in. There’s also a link to register subpage. The user can additionally choose to log in with his google account for which there is an option. The register subpage is similar to this one in layout except without the google button area, and thus a picture of it is not needed. After adding user information and clicking the login-button the user is taken to the main page. From there he can choose to search for a ride.

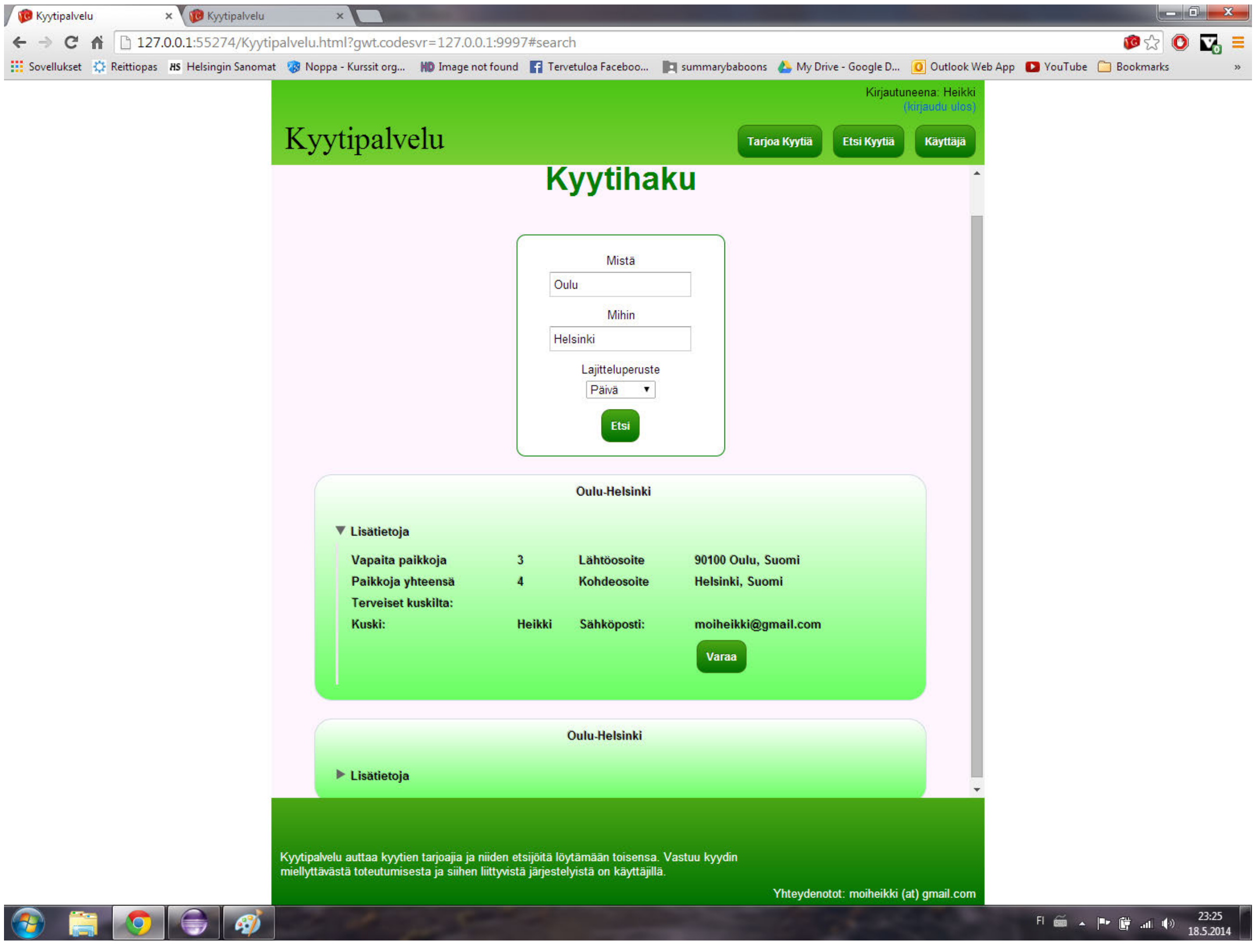

Kyytipalvelu
127.0.0.1:55274/Kyytipalvelu.html?gwt.codesvr=127.0.0.1:9997#login
Kyytipalvelu
Tarjoa Kyytiä
Etsi Kyytiä
Kirjaudu
Kirjaudu
Käyttäjänimi
Salasana
Kirjaudu sisään
Ei omia tunnuksia? Rekisteröidy tästä
Tai
Kirjaudu google-tunnuksillasi
(Emme jaa mitään ilman lupaasi)
Google

Kyytipalvelu auttaa kyytien tarjoajia ja niiden etsijöitä löytämään toisensa. Vastuu kyydin miellyttävästä toteutumisesta ja siihen liittyvistä järjestelyistä on käyttäjillä.

Yhteydenotot: moiheikki (at) gmail.com

Picture 2: login page

In picture 3 you can see how the header has changed after the user has logged in. It now has the username of the user and the possibility to log out or enter the account page. You also see the search page in the content area. Only the upper container is seen at first, and as the user filled in information “Helsinki” and “Oulu” and clicked search, the website dynamically added the search results under the search container. Each is shown only as small search results at first, but by clicking “lisätietoja” the user is able to see all necessary information about the ride, including a reserve button. By clicking that button he can reserve a seat in that ride - after being prompted whether or not he’s sure of his choice.


127.0.0.1:55274/Kyytipalvelu.html?gwt.codesvr=127.0.0.1:9997#search

Kirjautuneena: Heikki
(kirjaudu ulos)

Kyytipalvelu

Tarjoa Kyytiä · Etsi Kyytiä · Käyttäjä

Kyytihaku

Mistä: Oulu

Mihin: Helsinki

Lajitteluperuste: Päivä

Etsi

Oulu-Helsinki

▼ Lisätietoja

Vapaita paikkoja	3	Lähtöosoite	90100 Oulu, Suomi
Paikkoja yhteensä	4	Kohdeosoite	Helsinki, Suomi
Terveiset kuskilta:			
Kuski:	Heikki	Sähköposti:	moiheikki@gmail.com

Varaa

Oulu-Helsinki

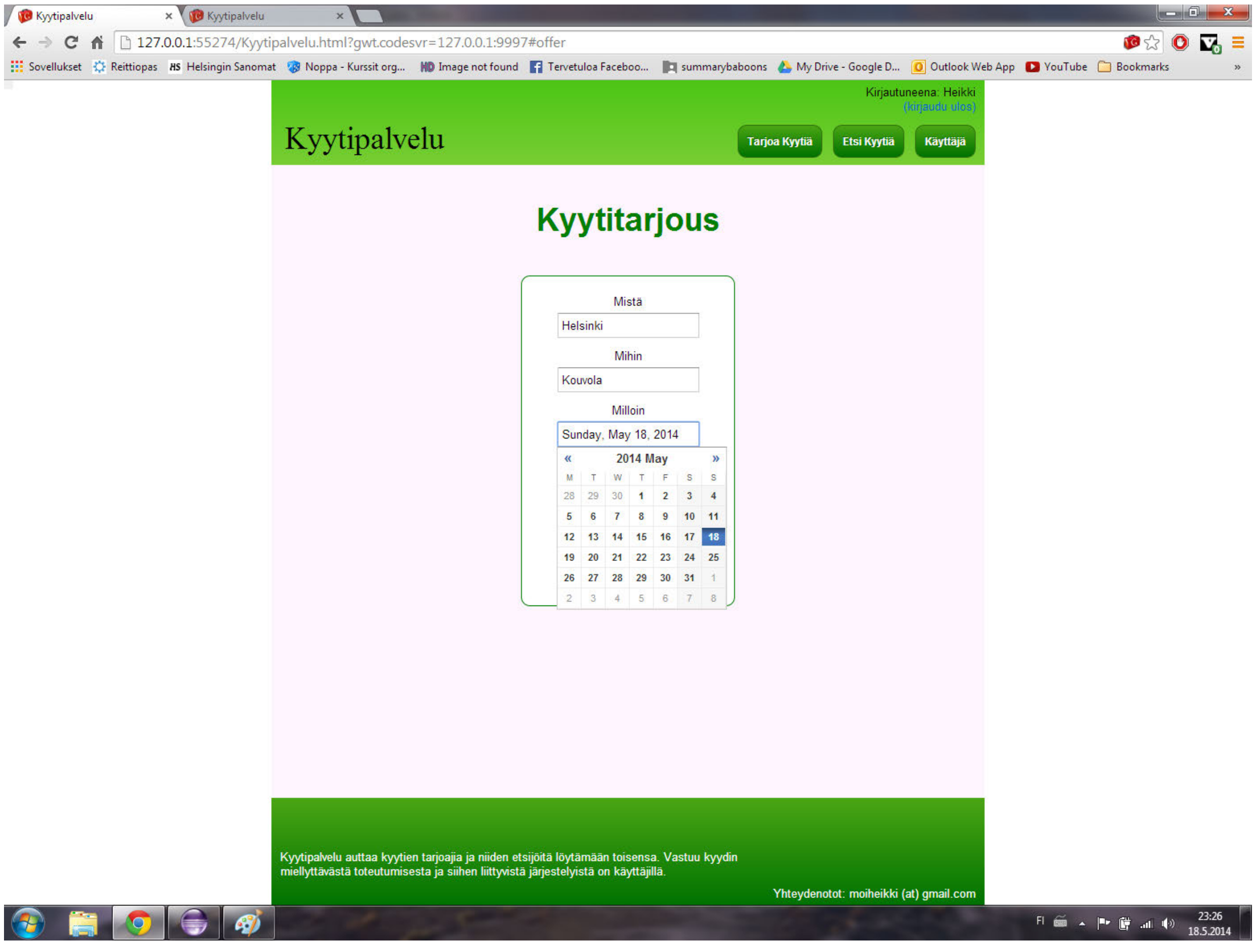
▶ Lisätietoja

Kyytipalvelu auttaa kyytien tarjoajia ja niiden etsijöitä löytämään toisensa. Vastuu kyydin miellyttävästä toteutumisesta ja siihen liittyvistä järjestelyistä on käyttäjillä.

Yhteydenotot: moiheikki (at) gmail.com

Picture 3: search page

In Picture 4 you can see how the user chooses to also offer a few rides of his own. He clicks “tarjoa kyytiä” on the navigation bar. He fills in the necessary information and uses the pop-up calendar for choosing a date. He also fills in a text message implying that he’s talkative.


127.0.0.1:55274/Kyytipalvelu.html?gwt.codesvr=127.0.0.1:9997#offer
Kyytipalvelu
Tarjoa Kyytiä
Etsi Kyytiä
Käyttäjä

Kirjautuneena: Heikki
(kirjaudu ulos)

Kyytitarjous

Mistä: Helsinki

Mihin: Kouvola

Milloin: Sunday, May 18, 2014

« 2014 May »

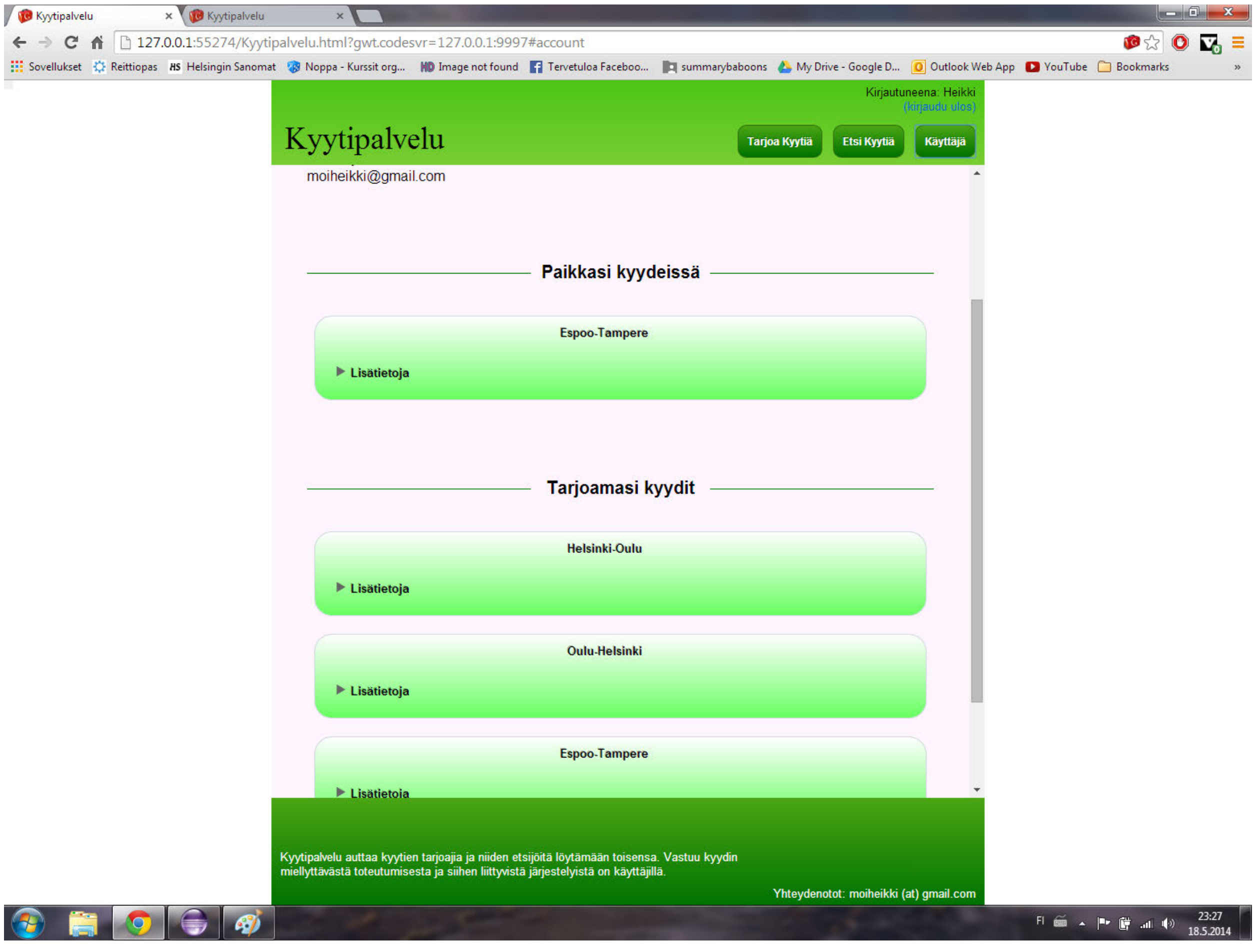
M	T	W	T	F	S	S
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Kyytipalvelu auttaa kyytien tarjoajia ja niiden etsijöitä löytämään toisensa. Vastuu kyydin miellyttävästä toteutumisesta ja siihen liittyvistä järjestelyistä on käyttäjillä.

Yhteydenotot: moiheikki (at) gmail.com

Picture 4: offer page

In picture 5 you will see the user’s own account page. It includes his basic information (username and email address) and all rides he has either reserved a seat in or has offered. These ride elements can also be expanded to show more information as in the search results, but naturally there’s no reserve button.

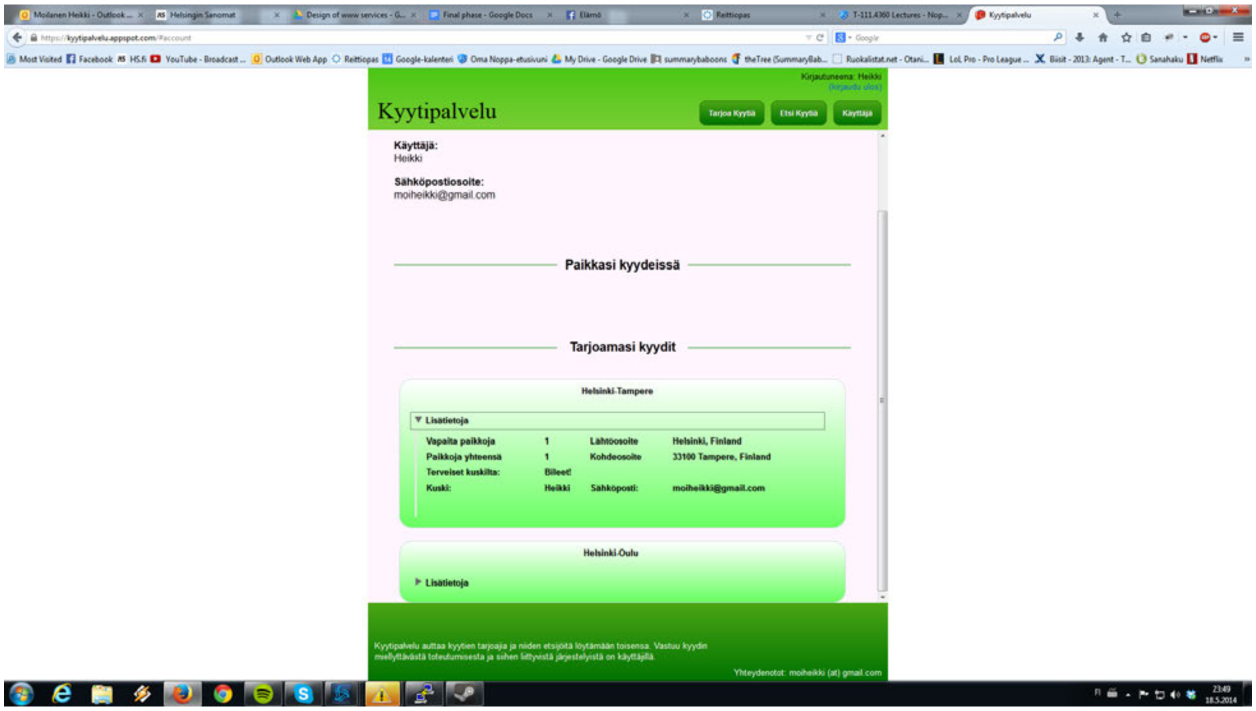


Kyytipalvelu auttaa kyytien tarjoajia ja niiden etsijöitä löytämään toisensa. Vastuu kyydin miellyttävästä toteutumisesta ja siihen liittyvistä järjestelyistä on käyttäjillä.

Yhteydenotot: moiheikki (at) gmail.com

Picture 5: Account page

As the course requested screenshots from multiple browsers, the next one (Picture 6) is the same page from mozilla firefox.


Kirjautuneena: Heikki
(kirjaudu ulos)

Kyytipalvelu

Tarjoa Kyytiä | Etsi Kyytiä | Käyttäjä

Käyttäjä:
Heikki

Sähköpostiosoite:
moiheikki@gmail.com

Paikkasi kyydeissä

Tarjoamasi kyydit

Helsinki-Tampere

▼ **Lisätietoja**

Vapaita paikkoja	1	Lähtöosoite	Helsinki, Finland
Paikkoja yhteensä	1	Kohdeosoite	33100 Tampere, Finland
Terveiset kuskilta:	Bileet!		
Kuski:	Heikki	Sähköposti:	moiheikki@gmail.com

Helsinki-Oulu

▶ **Lisätietoja**

Kyytipalvelu auttaa kyytien tarjoajia ja niiden etsijöitä löytämään toisensa. Vastuu kyydin miellyttävästä toteutumisesta ja siihen liittyvistä järjestelyistä on käyttäjillä.

Yhteydenotot: moiheikki (at) gmail.com

Picture 6: Account page in firefox

3.2 Reflection

Our original designs were boundary objects for discussion: We used them to discuss the colors and in the spirit of “this element here is one-colored, but in reality it’ll have a gradient to make it look better”. The positioning of differents elements stayed and the colors changed to look more modern. We held on to our idea of a very simplistic design.

4. Architecture and technologies

4.1 Architecture

The resulting architecture of the services matches almost exactly our design architecture.The service is designed to work as a single-page application where the document content is dynamically changed via JavaScript. As suggested in the framework’s documentation we used model-view-presenter pattern for the service architecture. An example of the service architecture is illustrated in the diagram below. All the views are be built using the framework’s built-in UI editor. The models will be stored in the database provided by Google App Engine. We are also using a third party library to simplify the store and save process of the models. The data models are shared between the client and server code. The presenters are responsible for handling any UI events and communicating with the application controller and the server. Communication between the client and the server is done using the framework’s built-in RPC services which also does the automatic serialization of the data models. Contrary to the original server-side architecture we ended up to using our own custom solution for the user authentication and authorization. We were first looking for third party system for user management but we were unable to find suitable library for your technology stack. For Google Maps API integration we are using a third party module for Google Web Toolkit that provides us clean and simple API to access all the latest Google Maps API features.

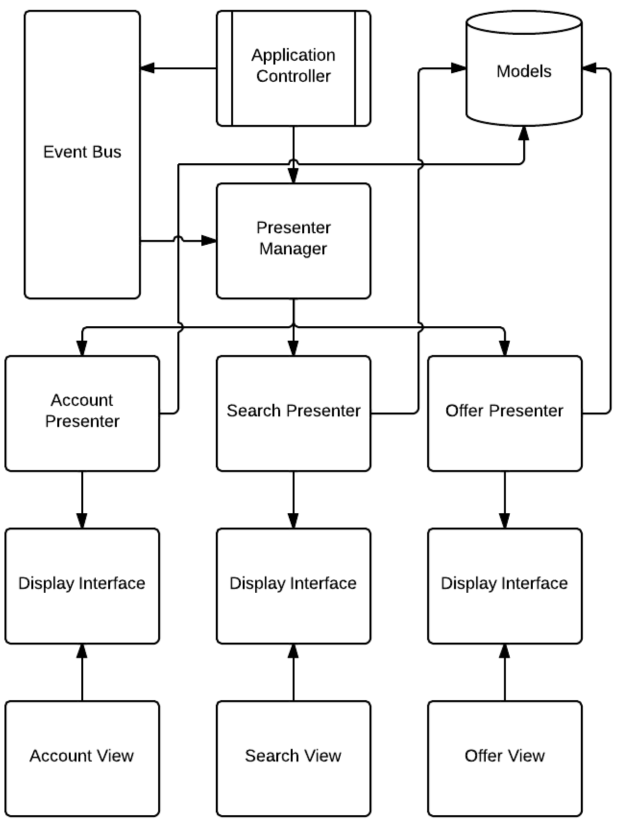


Diagram2: An illustration of the client architecture

Using the architecture described above provided us very robust and easily modifiable architecture with functionalities needed for our service. Using the frameworks integrated UI editor and simultaneous programming of the server and client code allowed us rapidly to develop our service and achieve our goals. Also using a single programming language for both client and server development made the application and server-client communication debugging much more easier and also allowed us to easily shared client and server side tasks.

4.2 Database

The database of our service is implemented using Google App Engine's DataStore API which provides us a schemaless NoSQL datastore. This means that the entities of same kind stored in the datastore can have different properties. Also we can create all the database tables in the code during runtime without needing to execute any external scripts. This allowed us to easily modify our database structure without needing to do any changes to entities previously stored in the database. The queries that can be executed are more restrictive in this system because of the way how the entities are indexed.

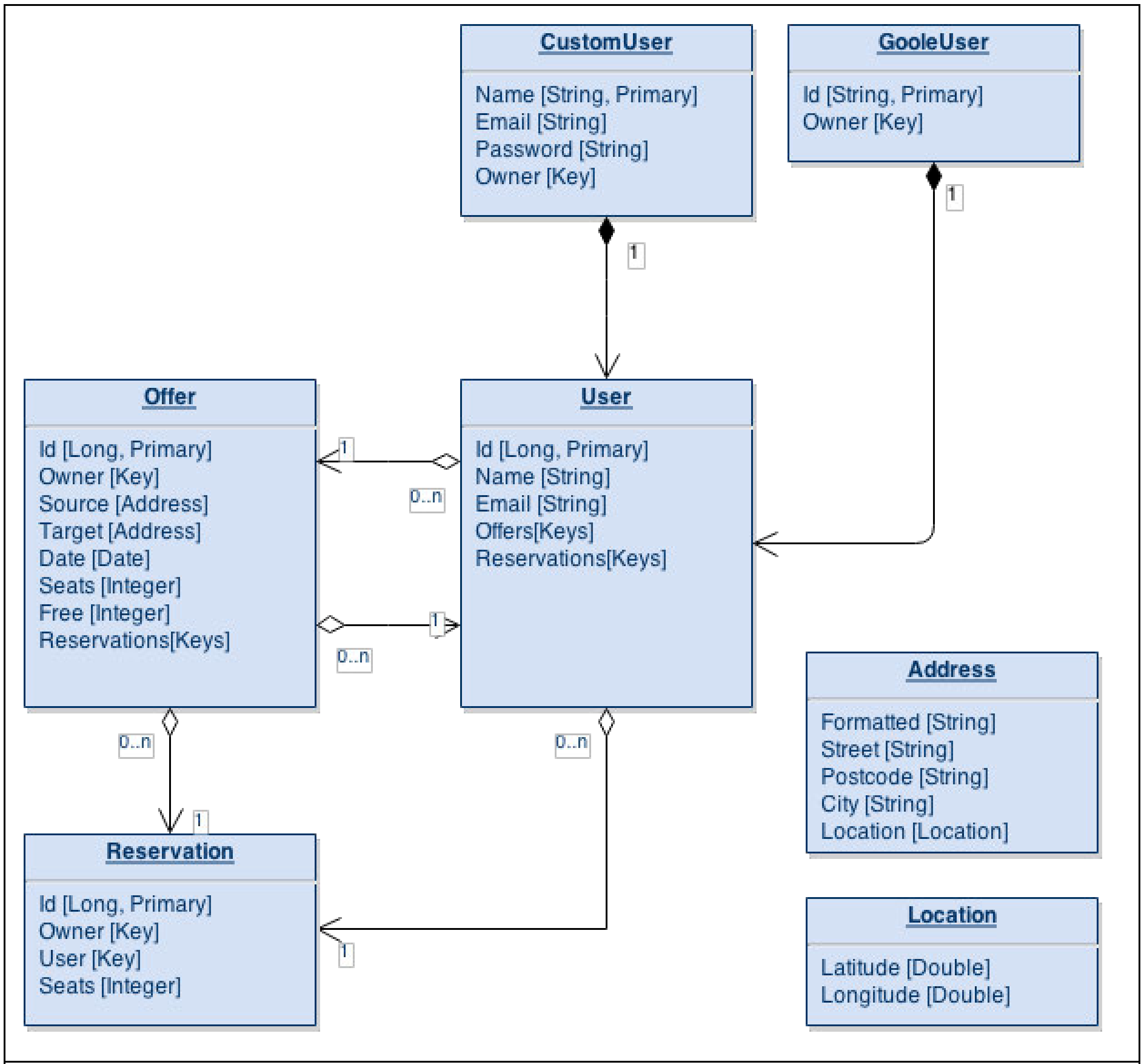


Diagram3: Database structure of the service

The structure of our service's database is a very compact. The service has five different kind of entities and the most important ones are the User and Offer entities. User entity stores all the user information which is includes unique identifier, email address, user's nickname and the offers and reservations made by the user. The login information is stored in a separate entity.

GoogleUser and CustomUser entities hold the login information. GoogleUser entity is used when the user sign-in using Google Account and CustomUser entity is used in other cases. User's password is encrypted and random salted when it is stored in the CustomUser entity.

Offer represents a trip that the user can leave to the services and from which other user can reserve seats. Offer entity contain source and target address which stores the street, postcode, city and location of the address. These values are filled by the client using the Google Maps API. The street and postcode are not required and may be empty in some cases. The location of the address is most important information because it is used by the search algorithm. The other properties can be used on the client side to sort the results.

The Offer entity also contains the date of the trip, number of seats, number of free seats and the keys to the reservations made by the other users. Reservation entity holds the information about the reservation including number of seats reserved and the user who made the reservation.

4.3 Performance

On the client side the performance of the service depends heavily on the browser's JavaScript performance because all the UIs are created dynamically. The JavaScript code generated by the GWT framework is highly optimized and browser independent. We are also using asynchronous functions to keep our UIs responsive when doing heavy work (for example loading other UI's) and communicating with the server or third party libraries. We have tried to avoid any extra traffic caused by images with minimizing their usage and relying on using plaing css-styles.

The search algorithm used to find offers is our biggest bottleneck. To improve the performance we have divided some of the task between client and server. The client does all the address parsing and location finding using the Google Maps API. The server only needs to compare the location coordinates to the coordinates stored in the database. Because of the automatic caching and synchronization of the Google App Engine the database queries should be really fast even when the queries are reasonable large. We didn't have enough time to further optimize the queries but in the future version we could use geohashing to encode our geographic coordinates. This should greatly increase the speed of the search queries because we could filter out much more results when doing the query to the database. Currently most of the filtering is done after the query.

4.4 Security

For user authentication and authorization we planned to use a third party library that could handle securely all the user registration and sign in flows for us. But unfortunately we couldn't find a suitable working solution for our service because of the technologies and dependencies we have. We implemented our own user management system that stores the user password using BCrypt and random salting. This should ensure that the passwords shouldn't be easily crackable if passwords are stolen.

To fight against other type of attacks we have followed strictly Google Web Toolkit's security guidelines. Instead of directly modifying the html code we rely on the component abstraction provided by the GWT framework. GWT framework is very secure against the XSS vulnerabilities because of the abstraction. Also all communication between the client and server is secured using HTTPS communications protocol. Users are always redirected to use HTTPS over insecure HTTP communication protocol. Unfortunately we didn't have enough time to implement GWT framework's built-in XSRF token service which would have protected us against cross-site request forgery attacks.

4.5 HTML & CSS

Google Web Toolkit mostly creates the html by itself - in some specific cases we wrote some html ourselves. CSS were written by hand and can be found in Kyytipalvelu.css. We decided to include them in a single file instead of nesting multiple files.The HTML Google Web Toolkit creates is defined through an XML-file and leads to well formed and valid HTML. The validation was tested with w3schools HTML validator. CSS can be found in Kyytipalvelu.css. We decided to include them in a single file instead of nesting multiple files.

4.6 Browser dependencies

All the browser compatibility issues are handled by the Google Web Toolkit. This means that we didn't need use any time to work on ensuring that the service works on all browser. The only dependency is that the browser must have JavaScript enabled otherwise the service will not work at all. All other features are created using JavaScript supported by all the browsers. The css-styles should work at least in the tested browser. The services has been mostly tested with Google Chrome and Mozilla Firefox. Also some smoke testing have been done with Internet Explorer (version 6 or lower of the Internet Explorer is not supported anymore by the Google Web Toolkit).

4.7 Technologies and Installation

The main technologies used in the development were Google Web Toolkit (GWT) framework and Google App Engine (GAE) service. These two technologies are the core of your service infrastructure. The GWT framework handles both client side and server side logic. Also with the GWT's UI editor we were able to implement UI's rapidly. The GAE services provide us the databases for storing our data and the hosting for the service. We have also used couple of third party plugin-ins with GWT framework that includes Google Maps API integration and simplified data storing to GAE services.

4.7.1 Technologies and libraries

Name	Version	Link

Google Web Toolkit	2.5.1	http://www.gwtproject.org

The Google Web Toolkit (GWT) provided us the infrastructure to build the service easily with Java programming language without needing to code single line of JavaScript or any other language. Also both the client and the server side programming were done in a same project workspace. This greatly reduced any communication errors between the server and client. Also debugging of the service was really easy because were able to simultaneously debug both the client and the server with the debugger provided by the Java framework.

Name	Version	Link
Google App Engine	1.9.1	https://developers.google.com/appengine/

The Google App Engine (GAE) provided us the database to store our data and also the hosting services for our service. Because of the GAE's schemaless datastore it was easy for us to change the database structure by our needs during the development. Also the GAE integrates seamlessly to the GWT framework this meant that we didn't need to spent time on setting up or deploying our service. Everything was done by a single click in the IDE. Hosting services provided us the security and stability to run our service. All communication is done over secured connection and the server and database status can be monitored from the GAE's administrative pages.

Name	Version	Link
GWT-Maps-V3-Api	3.10.0 Alpha 7	https://github.com/branflake2267/GWT-Maps-V3-Api
Gwt-ajaxloader (required)	1.1.0	https://code.google.com/p/gwt-google-apis/

With the GWT maps API we were able to query locational information from the Google Maps API using simple Java callbacks on the client-side. The implementation took only few lines of code. Unfortunately we didn't have enough time to use any interactive maps and we only used it to check the addresses get from the user input and fetching the geographical coordinates of the addresses.

Name	Version	Link
Objectify	5.0	https://code.google.com/p/objectify-appengine/
Objectify-GWT	1.0	https://code.google.com/p/objectify-appengine/

Objectify library is data access API for the Google App Engine’s DataStore API which simplifies loading, saving and querying when using GAE’s data storing services. It hides the lower level API of the DataStore and provides us simpler API to use. Using Java’s annotations and generics we were able to reduce the complexity of saving and loading information in our service and with GWT support we could use the object model also on the client side. This meant that we didn’t need to create duplicate models for the client-side.

4.7.2 Running and deploying the service

The Eclipse IDE is required to run or deploy the service. You will also need the Google Web Toolkit plug-in for the Eclipse IDE. Google App Engine libraries should come with the GWT framework but after the installation you may need to update the existing libraries to the latest versions using Eclipse IDE’s update functionality. The latest version of the GWT framework (2.6.0) has an issue that prevents the users to open the UI editor, because of that we used 2.5.1 version of the GWT framework. After installing the frameworks and importing the project files the service should runnable and deployable from the Eclipse IDE. Below are step-by-step instructions to run and deploy the service.

1. Install Google Web Toolkit Plugin For Eclipse IDE using the instructions from http://www.gwtproject.org/download.html
2. Select “Google Plugin for Eclipse”, “GWT Designer for GPE” and “SDKs” to install
3. Update Google App Engine libraries by checking for new updates from the Eclipse IDE
4. Import the “Kyytipalvelu” project as an existing project to the Eclipse IDE
5. Check that the Google Web Toolkit and Google App Engine are included in the project’s build path (also check that all the third party libraries are included)
6. Select “Kyytipalvelu.gwt.xml” located in “src/fi/aalto/kyytipalvelu/” using the file explorer
7. Choose run or debug from the selection menu to start the service
8. To deploy the service you have to press the GDT pulldown icon (blue google icon) from the Eclipse IDE
9. Selecting the “Deploy to App Engine…” will start the deployment wizard

5. Dynamic functionality

As an example of the site’s dynamic functionality we have decided to show how the site’s search system works. The flow of the system is described below and it is also presented as an diagram in the picture below.

The idea behind the search system is that the user can search for ride offers that matches to the given search parameters. The search parameters that the user can input are source and target addresses. Both target and source addresses are not required but at least one them must be given. Before sending the search parameters to the server two things are done. First the target and source addresses processed with Google Maps API. This process will resolve the real address and location of the address inputted by the user. The Google Maps API can find us the missing information if the user didn't provide all the needed information. For example if the input address is missing city name, the API can guess the correct city from the address provided. It also provides us clean and formatted address of the input. The most important functionality of the API is the fetching of the geographical coordinates of address. For any address it can resolve the geographical coordinates and return them using latitude and longitude values. This is crucial for the search algorithm used by server. If the API can't resolve the address inputted by the user the process returns an empty address which the client must handle properly. If at least one of the addresses inputted by the user is resolved the client will send search request to the server. Request is asynchronous and it doesn't block the UI but the search button is disabled while waiting for the response. This to avoid spamming of the same search request. If none of the addresses were resolved the client will show a message stating that the addresses couldn't be resolved.

When the server gets the search request it will first validate the search query parameters by checking the target and source addresses and throwing an error if they are not valid. The client will catch the exception in the server response and acts correctly. After validation the server will begin the search for nearby offers. Due to the restrictions of the Google App Engine we first must make a query that returns all offers that are after the date get from the search parameters. After that the query results are processed in a loop where the source and target address of the offer are compared to the search request's source and target address. This done by using an algorithm that can calculate distance between two geographical coordinates and return the distance in meters. Also the offer must have enough seats to be accepted. This is repeated until all offers returned by the query are compared or the maximum limit of search results is reached. Currently the maximum limit and distance are constants determined by the server. The server will return a response containing an array of offers that matched to the search request parameters. The array may be empty if no matching offer are found.

After receiving the response from the server the client empties and fills the results container with the results from the response. If no offers were found a message stating that the user should try again with different search parameters is shown. Items in the results container are filtered by the currently active filtering mode which the user can change any time. The default option for sorting is by distance from closest to furthest. Each item shows information of the offer. This includes formatted address of the source and target addresses, number of seats, number of available seats and information about the owner of the offer.

To reserve a seat from the offer the user must press the reserve button in the drop down menu of the item. This triggers the reservation process where the user must first confirm the reservation via popup window. After that a new reservation request is sent to the server. The server will check that the user is logged in and throws an exception if it fails. Next the server will check that the offer has enough seats available for the reservation. If the check fails the reservation is discarded and an exception is thrown. If succeeded the reservation is saved in the database and added to the offer. The offer is also updated and saved in the database. The server will return a response containing the created reservation. The client will show either success or failure message according to the returned response state. Also the offer where the reservation was made is updated.

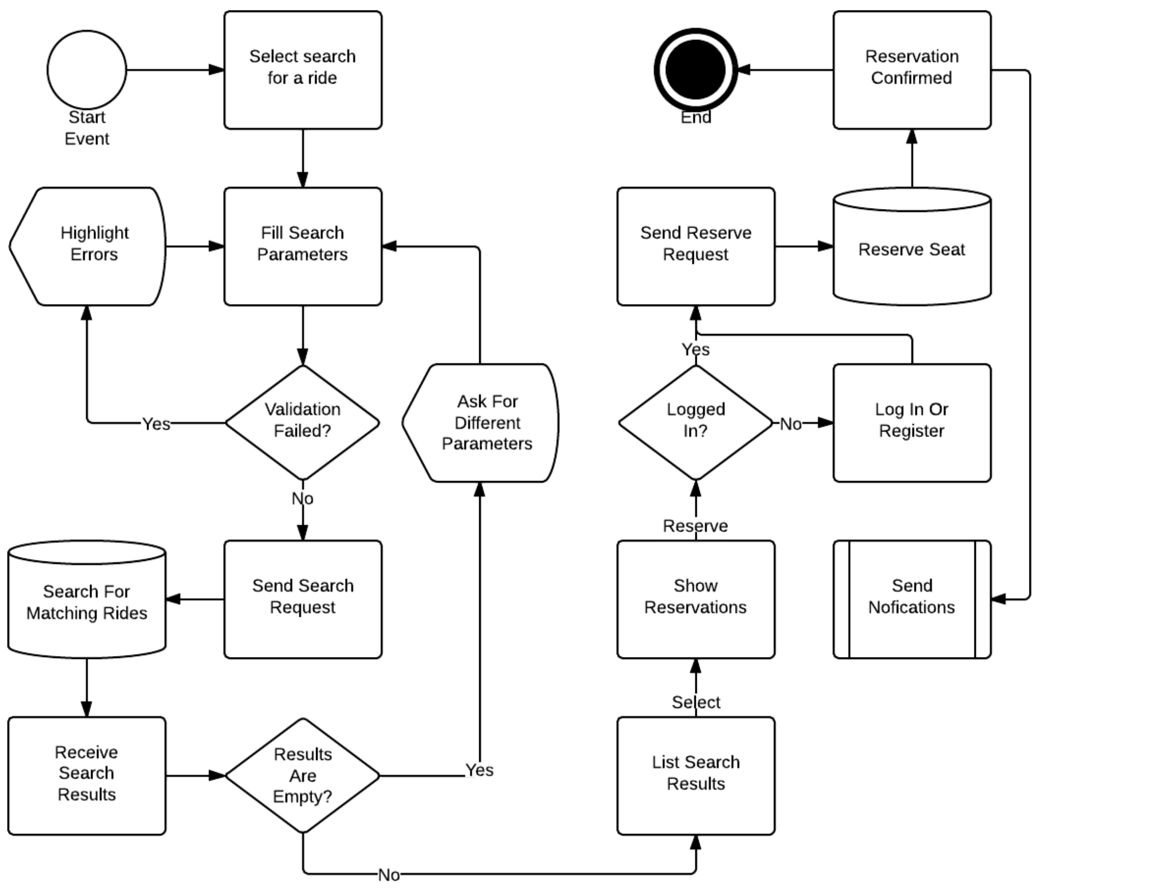


Diagram 4: A flow diagram of the dynamic search functionality

5.1 Reflection

Our final implementation follows our original design very closely. There are some features that were dropped because of the time limits and conflicts with the technologies used. Most importantly our architecture stayed untouched throughout the development. This means that we didn’t need to do any large modifications to our project in order to make some of the features work. Also the performance estimations should be quite accurate but this cannot be confirmed until performance analysis is done. To improve security of the service we were thinking of implementing GWT framework’s built-in XSRF token service but we had to drop it because we run of time. Reason for this was the unexpected problems finding a suitable user authentication and authorization library for the project. This caused us to waste lot of time on task that was considered trivial and quick and it also delayed implementations of the other features. This could have been avoided by paying more attention to the tasks that were thought to be "not-so-important" and trivial to implement. Browser dependencies are handled entirely by the Google Web Toolkit and they have stayed same through the development process. For the dynamic functionality we managed to implement almost all the features that were described in the original design documentation. The route comparison stage from the search algorithm and email notifications were dropped from the final version due to the time limitations. We decided to implement algorithm that only compares start and finish coordinates instead of comparing full path. This was because complexity of algorithm would have grown exponentially and it would have required considerably more research and time to implement.

In overall our original design was very well planned and realistic. We didn't need to make any large changes to the service design after starting the development. Also the use of the more unknown Google Web Toolkit and Google App Engine provided us lot of valuable experience working with platforms and services that are closely tied together.

6. Design of the project

6.1 Work breakdown structure

Login page

Name	Type	Description	Priority	Estimate	Used	Assigned to
Layout design	Design	Create a sketch of the page layout.	2	2h	4h	[redacted]
Layout implementation	Client	Implement the page layout using the editor.	2	2h	3h	[redacted]
Page logic	Client	Request authentication information from the server and redirect the user to the front page. Allow users to sign in for a new account.	2	4h	3h	[redacted]
Google login	Client & Server	Enable the user to log in with his Google account	2	10h	10h	[redacted]
Fetch account information	Server	Authenticate the user information and fetch the account information from the database.	2	6h	10h	[redacted]
Testing	Client/Server	Test client and server side functionalities.	3	1h	1h	[redacted]

Sign in page

Name	Type	Description	Priority	Estimate	Used	Assigned to
Layout design	Design	Create a sketch of the page layout.	2	2h	2h	[redacted]

Layout implementation	Client	Implement the page layout using the editor.	2	2h	2h	[redacted]
Page logic	Client	Validate the account information with the server and create a new account. Redirect the user to the account page.	2	4h	5h	[redacted]
Store account information	Server	Validate the account information and store it to the database.	2	8h	20h	[redacted]
Testing	Client/Server	Test client and server side functionalities.	3	1h	1	[redacted]

Account page

Name	Type	Description	Priority	Estimate	Used	Assigned to
Layout design	Design	Create a sketch of the page layout.	2	2h	2h	[redacted]
Layout implementation	Client	Implement the page layout using the editor.	2	2h	2h	[redacted]
Page logic	Client	Fetch account information from the server.	2	4h	2h/1h	[redacted]
Show offers	Client/Server	Fetch and show all the offers where the user is the owner of.	3	6h	1h	[redacted]
Show reservations	Client/Server	Fetch and show all the reservations of the user.	3	6h	1h	[redacted]
Modify account information	Client/Server	Allow users to change account information including password and email address. Update the account information in the database.	3	4h	0h	[redacted]
Testing	Client/Server	Test client and server side functionalities.	3	1h	1h	[redacted]

Search page

Name	Type	Description	Priority	Estimate	Used	Assigned to
Layout design	Design	Create a sketch of the page layout.	1	2h	2h	[redacted]
Layout implementation	Client	Implement the page layout using the editor.	1	2h	4h	[redacted]
Page logic	Client	Send the search request to the server and present the search results. Show only fixed amount of offers.	1	4h	8h	[redacted]
Multi-page search results	Client	Split the search results on multiple pages in case of large amount of items.	3	4h	0h	[redacted]
Google maps integration	Client	Allow users to select route information from a interactive map.	2	8h	0h	[redacted]

Filters for the search results	Client/Server	Allow users to filter search results using distance, date, price etc.	4	4h	3h	[redacted]
Search for closest routes	Server	Create an algorithm that can find the best matching routes using the route information.	1	4h	4h	[redacted]
Fetch offer information	Server	Fetch offers from the database.	1	2h	2h	[redacted]
Testing	Client/Server	Test client and server side functionalities.	3	1h	1h	[redacted]

Offer information page

Name	Type	Description	Priority	Estimate	Used	Assigned to
Layout design	Design	Create a sketch of the page layout.	1	2h	1h	[redacted]
Layout implementation	Client	Implement the page layout using the editor.	1	2h	2h	[redacted]
Page logic	Client	Send a seat reservation to the server and redirect to the confirmation page.	1	4h	5h	[redacted]
Reserve a seat	Server	Check for open seats and reserve a seat from the offer.	2	4h	4h	[redacted]
Send notification to the owner	Server	Send an email to the offer owner when a seat is reserved.	4	4h	0h	[redacted]
Allow owner to reject/view seat reservations	Client/Server	Add extra step for the reservation flow which allows the owner to reject a reservation.	4	12h	0h	[redacted]
Testing	Client/Server	Test client and server side functionalities.	3	1h	1h	[redacted]

Landing page

Name	Type	Description	Priority	Estimate	Used	Assigned to
Layout design	Design	Create a sketch of the page layout, decide which page it is based on.	3	2h	2h	[redacted]
Layout implementation	Client	Implement the page layout using the editor.	3	4h	1h	[redacted]

Offer page

Name	Type	Description	Priority	Estimate	Used	Assigned to
Layout design	Design	Create a sketch of the page layout.	1	2h	2h	[redacted]
Layout implementation	Client	Implement the page layout	1	2h	1h	[redacted]

Page logic	Client	Send offer information to the server.	1	4h	2h	[redacted]
Validate offer information	Client/Server		2	2h	1h	[redacted]
Google maps and directions integration	Client	Allow user to select route information from a interactive map. Use directions API to calculate route waypoints.	2	8h	0h	[redacted]
Store route information	Server	Store the route waypoints to the database. Waypoints are used by the search algorithm.	2	4h	2h	[redacted]
Store offer information	Server	Store the offer information to the database.	1	2h	2h	[redacted]
Testing	Client/Server	Test client and server side functionalities.	3	1h	2h	[redacted]

Offer/Reserve Confirmation page

Name	**Type**	**Description**	**Priority**	**Estimate**	**Used**	**Assigned to**
Layout design	Design	Create a sketch of the page layout.	3	2h	0h	[redacted]
Layout implementation	Client	Implement the page layout using the editor.	1	2h	1h	[redacted]
Page logic	Client	Show the offer or the reservation information.	1	4h	1h	[redacted]
Testing	Client/Server	Test client and server side functionalities.	3	1h	0,5h	[redacted]

Main layout

Name	**Type**	**Description**	**Priority**	**Estimate**	**Used**	**Assigned to**
Site layout design	Design	Create a sketch of the site layout.	1	8h	8h	[redacted]
Site Layout implementation	Client	Implement the site layout using the editor.	1	8h	7h	[redacted]
Css design	Design	Plan and Create the site wide stylesheet (colors, styles etc).	2	8h	5h	[redacted]
Header and footer design	Design	Create a sketch of the footer and the header.	3	4h	3h	[redacted]
Header and footer implementation	Client	Implement the page layout using the editor.	3	4h	3h	[redacted]
Navigation	Client	Implement basic site navigation and page history.	1	8h	8h	[redacted]
Sketches	Design	Rough sketches of the site structure and layout.	1	4h	5h	[redacted]
Testing	Client	Test for layout errors.	3	1h	0,5h	[redacted]

Server setup

Name	Type	Description	Priority	Estimate	Used	Assigned to
Design the database structure	Design/Server	Determine the relations and create a document describing the database structure.	1	2h	3h	[redacted]
Create databases	Server	Create the databases using the Google Apps Engine API.	1	4h	2h	[redacted]
Set up the server	Server	Create server setup scripts.	1	4h	0h	[redacted]i
Deploy the server	Server	Deploy the server to the Google Apps Engine.	2	2h	1h	[redacted]
Admin page	Server	A page where the server settings can be change on the fly. Also possibility to view and modify stored data.	4	4h	0h	[redacted]
Familiarize working with the framework	Desing/Client/Server	Read the framework documentation and tutorials.	1	6h	7h	[redacted]
Testing	Server	Create server-side unit test cases.	3	1h	0h	[redacted]

Design Documentation

Name	Type	Description	Priority	Estimate	Used	Assigned to
Layout	Design	Cover page, contents page, styles, headers and footers	1	2h	10h	[redacted]
Design demo	Documentation	Design demo required by the course	1	2h	4h	[redacted]
Illustration	Documentation	llustrations for the design document	1	3h	3h	[redacted]
Description of our work	Documentation	Introduction, user group analysis and the structure of the site	1	2h	6h	[redacted]
Description of content	Documentation	Definition of each page and the production process	1	2h	4h	[redacted]
Description of the layout and the user interface	Documentation	See design document instruction from noppa	1	2h	4h	[redacted]
Description of dynamic functionality	Documentation	See design document instruction from noppa	1	2h	4h	[redacted]
Description of project design	Documentation	Time & areas of responsibility	1	2h	4h	[redacted]

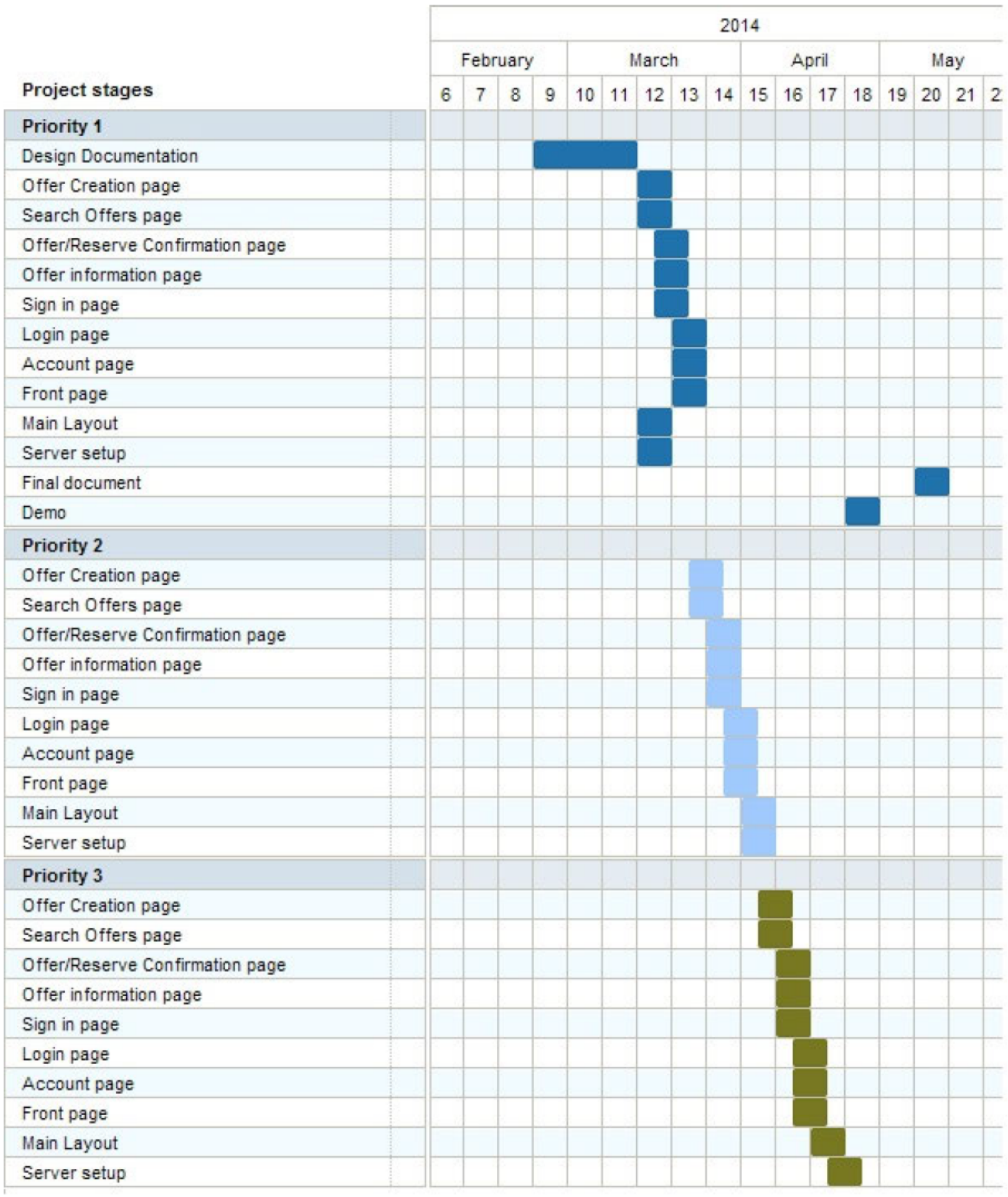
Demo

Name	Type	Description	Priority	Estimate	Used	Assigned to
Demo meeting 1-3.4	Documentation	Preparation for the meeting and the meeting itself	1	3h	4h	[redacted]
Demo Gala 28-30.4	Documentation	Preparation for the gala and the gala itself	1	3h	0h	[redacted]

Final Document

Name	Type	Description	Priority	Estimate	Used	Assigned to
Layout	Documentation	Cover page, contents page, styles, headers and footers	1	1h	1h	[redacted]
Illustration	Documentation	llustrations of different sides of the product	1	1h	2h	[redacted]
Description of our work	Documentation	Introduction, user group analysis and the structure of the site	1	1h	3h	[redacted]
Description of content	Documentation	Definition of each page and the production process	1	1h	3h	[redacted]
Description of the layout and the user interface	Documentation	See design document instruction from noppa	1	1h	2h	[redacted]
Description of technologies and libraries	Documentation	See design document instruction from noppa	1	2h	2h	[redacted]
Description of dynamic functionality	Documentation	See design document instruction from noppa	1	1h	2h	[redacted]
Description of project design	Documentation	Time & areas of responsibility	1	1h	2h	[redacted]
Retrospection	Documentation	What went well during the project, what did not	1	6h	6h	[redacted]

6.2 Time schedule



The original Gantt chart

Our original Gantt chart planning can be found above. In the end the tasks were distributed a bit more towards the end of the course, not as evenly as planned. We still worked with the project during the whole spring, so even if the time distribution didn’t go exactly as planned, we did our best with the time we had.

6.3 Use of time

The use of time on each tasks is depicted in the work breakdown structure above.

The approximated amount of time used by different team members:

- [redacted]: 95 hours
- [redacted]: 102 hours
- [redacted]: 85 hours

6.4 Reflection

The design phase of the project went smoothly with all team members having sufficient amounts of time for the project. The implementation phase proved more difficult as all members were rather busy with other courses and/or work. Thus the weightload of the implementation phase got heavily weighted to the beginning and to the end of the project, midway being a rather difficult timespan.

The prioritizing we initially made correlated with the features and was well done.   The estimations of time were done roughly and weren’t all exact. The accuracy of the estimations depended a lot on in which order you made the tasks, since you learned more from each one. The authentication took more time than we estimated.

7. Final reflection

We have made a good service that does roughly what we planned it would. The extra features prioritized low were not realized. In the future the service could be developed further by realizing the features we planned originally. Still it wouldn’t be necessary to show the Google map for the user, since they can choose their route quicker by typing.

Our service would be useful for our target groups. At the moment there is a Facebook-page in Finland that is used to organize shared rides. This service would be very useful for the people using the Facebook page. For an actual application to be launched we should take our service further and offer a bit more, opportunities for communication and a possibility to name a price for the trip for example.

Our project succeeded mostly according to our original plans. Most of the problems came from trying to find enough time to realize the project during the spring when all of our group members were very busy. The authentication was surprisingly hard also.

The design and initial project planning were successful and the priorities set were mostly accurate. We followed the priorities for most of the features, except when we noticed that the lower priority features are easy to realize.

We wouldn't do much differently. As mentioned earlier the main difficulty with the project was finding enough time, but we did our best in the current situational environment.

We agreed on a weekly meeting on thursdays. However due to everyone's schedules we ended up meeting on maybe 3 thursdays with all three of us. There were problems related to scheduling that we think wouldn't happen in actual work life.
We're most proud in the project of the technical side of it, with the well-defined architecture and asynchronous requests.

We all learned different things during the course. The architecture and asynchronous requests and the model-view-presenter -pattern were new to some. Good designs and prioritization also helped us to get through schedule problems.

7.1 Extra bonus points

The architecture of our application - Model-view-presenter and asynchronous requests - is well defined, modular and enables a smooth basis for future development. Also after no external libraries suited our authentication needs the authentication we made ourselves works rather nicely. We think these are issues we are proud of, they can be read more about in the technology section.