PRD - Product Requirements Document

Team 1

Contents

Revision history

Version	Date	Reason
1.0	2021-09-15	First draft
1.1	2021-09-15	Fixed typos from team feedback
1.11	2021-09-20	Added references to SDP and PH book
1.12	2021-09-20	Added clients product description to introduction and system user section
1.13	2021-09-20	Added description of admin under Terminology Section
1.14	2021-09-21	Linked requirements to user from pre-sprint 1
1.15	2021-09-21	Added more categories of non-functional requirements
1.16	2021-09-21	Added technological requirements
1.17	2021-09-21	Linked all requirements to user stories
1.18	2021-09-21	Completely revised route definitions and matching
1.19	2021-09-21	Added new UML use case diagram and new figure caption
1.20	2021-09-21	Fixed requirement dependencies
1.21	2021-09-21	Updated Appendix B based on discussion with team

1 Referenced documents

References

[1] "Programvaruutveckling för Stora System Projekthandledning 2021", Chapter 9, Institutionen för Datavetenskap Lunds Tekniska Högskola, Lunds Universitet, 26 August 2021

[2] "SDP - Software Development Plan, Team 1 - ETSN05", Alexander Ekman and Linnea Johnsson, url=`https://www.overleaf.com/read/bvbkjmdrkpdx`, accessed October 20, 2021

[3] "Ride sharing Benefits, Brooklyn National Laboratory", 2012, url=`https://www.bnl.gov/rideshare/benefits.asp`, accessed 2021-09-14

2 Introduction

This Product Requirements Document (PRD) details the requirements to develop a web-app based carpool service called ETSN05_PG1 Carpool Webb-app (ECW). The purpose of this PRD is to provide the system characteristics, the epics and user stories, as well as the requirements of the system. This document is intended as a basis for the contract between the development team and the client, it will also serve as a reference for the development team during the development process.

As stated in the clients product description[1], ECW will be a web-application for a carpool service. Users will set an origin, destination, time and date, for a car trip and be notified which car to go with. If the user is a driver, they should be able to go into the system, declare origin, destination, data and time, and be notified of any passengers joining the carpool. The system should find a reasonable distribution of passengers into the avaliable rides. In the simplest scenario, the system finds riders and drivers with the same origin, destination and time. In a more advanced scenario, the system can create rides where riders have different destinations along the same route. For example, one driver can drive between Malmö and Helsingborg with two passengers, where one rides along the whole route, and the other only rides along between Malmö and Landskrona. The system doesn't necessarily feature a map, but can utilize a list of addresses or cities from which users choose origin and destination. Users of the system need to register before using it. They will also declare if they are riders, drivers, or both. There should also be functionality for an administrative user, which can for example remove registered users. The system should also be able to handle abnormal behaviour from users without crashing or freezing. Multiple users should be able to use the system in paralell. There ares some topics left for the development team to work out, for example what cities are available, how a destination is defined to be "en rute", how riders and drivers are matched, and how they are notified.

The detailed plan of the developemnt can be found in the Software Development Plan (SDP) produced for the development of ECW. The document includes the process model, the project's phases and their corresponding time-estimates. The SDP also includes a description of the project's organisation, time plan, configuration handling, and quality review, as well as a a risk analysis of the project[2].

3 Background and product goals

3.1 Purpose

A lot of the cars on our roads today are only transporting single individuals, which is problematic when aiming for a more sustainable future [3]. The goal of ECW is to give non-car owners and car owners the ability to easily match with each other and carpool together to a common destination. ECW will provide its users with a more economic and sustainable mode of transportation.

3.2 System Users

The system user is either a non-car-owner who would like to carpool somewhere (rider), a car owner who is willing to bring someone along a planned route (driver), or someone who sometimes is a driver and sometimes a rider.

Based on the clients product description [1], ECW users will register an account as either driver, rider or both. During registration, they will provide some contact details, and drivers will provide details of their car. These details will be possible to edit later. A rider can log in and request a starting city, a destination city, and a time and date they would like to go. Likewise, a driver can submit a starting city, a destination and a time that they have planned to leave, and if a drivers submission matches with one or several riders requests a match is made. Contact details are then shared between the users for easy communication between them. All users are also here given the option to decline the offer. For future versions it is planned for drivers to be able to enter multiple cities where they are willing to stop, and in such a way be able to support more short distance riders.

As specified in Section 2 and the clients product description[1], ECW will support an administrator as a third user type. This admin user should have the ability to add registered users. In order to help with functionality testing during the product development, this administrator will also have the ability to create "fake" users and edit their profile details.

4 Terminology

ECW = The name of the system
Driver = A car owner who is willing to bring someone along a planned route
Rider = A non-car-owner who would like to carpool somewhere
Admin/Administrator = A user of the system with special privileges sremoving users.
Drive instance = The objects created when a driver has declared that they will be driving a certain route a certain date and time
Ride request = The objects created when a rider has declared that they would like to ride along a certain route a certain date and time

5 System Use Case Diagram

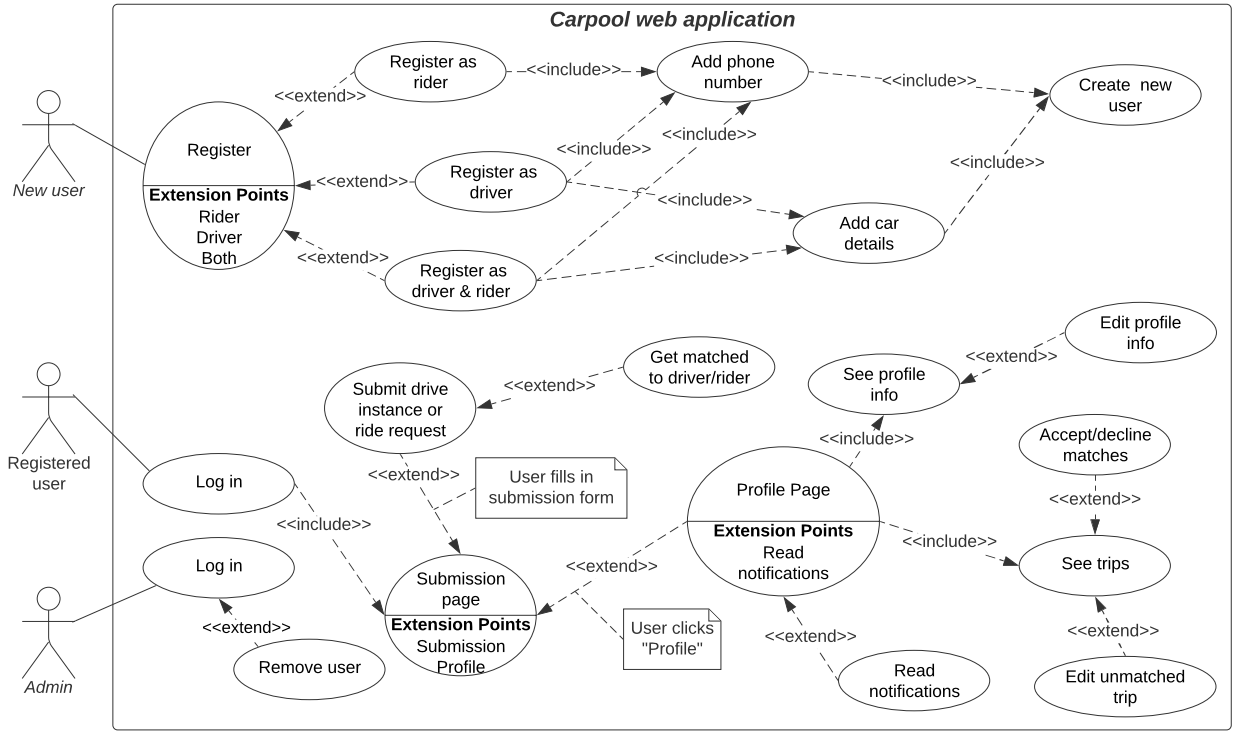

Figure 1: The system use case diagram for ECW. New users register as either riders, drivers, or both and can after registration log in to access the service. When logged in, the user is shown a submission form as a main page, shown in Figure 4. The user submits a an origin, destination, and time. If another rider/driver has a similar request a match is made and the users are notified. From the submission page, users can access their profile page, which is visualized in Figure 5. Here they can edit profile details, read notifications, and edit/accept/decline trips and matches.

6 Epics, User stories, and Issues

The entries are labeled as X.Y.Z where this represents a unique identifier as Epic.Story.Issue.

1. User registration and authentication
   - 1.1 As a user, I can register for the service as rider, drivers, or both
     - 1.1.1. All users register full name, email, and phone number
     - 1.1.2. If a user is a driver, they register car brand, model, color, and licence plate
     - 1.1.3. Design registration window
     - 1.1.4. Add a registration button in the login window
     - 1.1.5. Add functionality to the registration button
     - 1.1.6. As a registered user, I want to be able to log in and out
     - 1.1.7. Users log in using email
2. Matching
   - 2.1 As a driver/rider, I select two cities and a time
     - 2.1.1. If a user is registered as "both", they have to specify if riding or driving
     - 2.1.2. Add a drop-down button of available cities (frontend)
     - 2.1.3. Add functionality to the drop-down of available cities (backend)
     - 2.1.4. Add a "Create Drive Instance"/"Create Ride Request" button for drivers/riders respectively (frontend)
     - 2.1.5. Design how to store the drive instances and ride requests
     - 2.1.6. If a request is created by a user it is added to the requests list
   - 2.2 As a driver I can also add cities on my route
3. Matchmaking system
   - 3.1 As a user, if two users have matching routes, we want to be paired up
     - 3.1.1. Matches are made when there exists a driving instance, and there exists a rider request for a part of that route.
4. After successful match
   - 4.1 As a user, when a match is made, I want to be notified about the match and know how to recognize/contact each other
     - 4.1.1. Implement notifications in BASE
     - 4.1.2. Send notifications to specific user
     - 4.1.3. Send notifications to users from match making system
     - 4.1.4. Include useful information in notifications (car brand, model, color, driver/rider phone number)
   - 4.2 As a driver, I want to be able to decline a match
   - 4.3 As a rider, I want to be able to decline a match
5. Administration
   - 5.1 As administrator I can remove users
     - 5.1.1. Allow administrator to remove users

7 Project Requirements

This section contains all the technological, functional and non-functional requirements of the system. Requirements associated to a user story ID marked with an asterisk (*) means that this requirement has been assigned the best matching Sprint 1 user story, but there will be a new and more relevant user story created for this requirement in sprint 2.

7.1 Technological requirements

7.1.1 Back-end

7.1.1.1 Java and Jetty

ID T8:
User Story ID: 1.1.0*
Priority:HIGH
Description: The system will utilize a Java backend with Jetty and Jersey. Jetty provides a small footprint HTTP server and servlet container, whilst Jersey is an abstraction layer on op of Jetty for implementing REST servers.
Rational: This combination is chosen because this is a common setup and is the most familiar to the developers in the team.
Dependency: none

7.1.1.2 BASE server

ID: T4
User Story ID: 1.1.0*
Priority: HIGH
Description: The core of the system requires a server running the BASE system, provided as a part of the ETSN05 course at Lund University, faculty of engineering
Rational: The BASE system provides the core front-end and back-end functionality for the web-app. This includes user registration, requests, admin user etc.
Dependency: 7.1.1.1

7.1.1.3 SQL Trip database

ID: T5
User Story ID: 2.1.0
Priority: HIGH
Description: This project requires a server to host the submitted ride requests and drive instances in an SQL database so that these can be matched.
Rationale: In order to run a matching algorithm
Dependency: 7.1.1.2

7.1.1.4 Trip matching algorithm

ID: T6
User Story ID: 3.1.0
Priority: HIGH
Description: This project requires a server to run the matching algorithm between submitted ride requests and drive instances so that these can be matched.
Rationale: In order to match riders and drivers
Dependency: 7.1.1.3

7.1.2 Version control and unit testing

7.1.2.1 Version control and issue tracking

ID: T7
User Story ID: none

Priority: HIGH
Description: During development, the development team requires the use of GitLab for both version control and issue tracking
Rational: Version control and issue tracking is essential to the development and Lund University has an educational licence
Dependency: none

7.1.2.2 Unit testing
ID: T9
User Story ID: none
Priority: HIGH
Description: For Java and JavaScript unit testing the team will use JUnit and Jasmine respectively
Rational: These are common tools already familiar to the development team
Dependency: none

7.1.3 Front-end

7.1.3.1 HTML, CSS
ID: T1
User Story ID: 1.1.0*
Priority: HIGH
Description: This project requires a server to host HTML and CSS code
Rationale: In order to display contents to the users of the web-app
Dependency 7.1.1.2:

7.1.3.2 JavaScript
ID: T2
User Story ID: 1.1.0*
Priority: HIGH
Description: This project requires a server to host JavaScript code
Rational: JavaScript is necessary for the users interaction with the web-app and for web-app to dynamically change the contents shown to the user according to their actions
Dependency: 7.1.1.2

7.1.3.3 Bootstrap
ID: T3
User Story ID: 1.1.0*
Priority: HIGH
Description: This project requires a server with support for bootstarp
Rational: This makes the CSS and JavaScript development simpler, and gives the web-app a more modern look
Dependency: 7.1.1.2

7.2 Functional requirements

This section contains the requirements which specify the functions of the system.

7.2.1 User registration and authentication

7.2.1.1 User registration
ID: F1
User Story ID: 1.1.0

Priority: HIGH
Description: A prospective user should be able to access the website and be guided to registering a new account, as shown in Figure 2. To register for the service, all users must provide full name, email, password, and phone number as shown in Figure 3.
Rationale: In order for new members to register for the service
Dependency: 7.1.1.2

7.2.1.2 Driver registration
ID: F2
User Story ID: 1.1.0
Priority: HIGH
Description: A prospective user registering as a driver, or a rider and a driver must also register car brand, model, color, and licence plate number. This will be in addition to what is shown in Figure 3
Rationale: So that the driver is easily identifiable when picking up riders
Dependency: 7.2.1.1

7.2.1.3 User log-in
ID: F3
User Story ID: 1.1.0
Priority: HIGH
Description: A registered use should be able to log in using their email and password as shown in Figure 2.
Rationale: In order to submit a drive instance or a ride request
Dependency: 7.2.1.1

7.2.1.4 User log-out
ID: F4
User Story ID: 1.1.0
Priority: HIGH
Description: A logged in user should be able to log out from the current session, as shown in Figure 4.
Rationale: In order to close a session
Dependency: 7.2.1.3

7.2.2 Submitting drive instance or ride request

7.2.2.1 Drive instance submission
ID: F5
User Story ID: 2.1.0
Priority: HIGH
Description: A logged in driver should be able to submit a drive instance by selecting an origin, a destination, and a time and date for departure as shown in Figure 4.
Rationale: In order to show that a car is available on this route at this time
Dependency: 7.2.1.3

7.2.2.2 Ride request submission
ID: F6
User Story ID: 2.1.0
Priority: HIGH
Description: A logged in rider should be able to submit a ride request by selecting an origin, a destination, and a time and date for departure as shown in Figure 4.
Rationale: In order to show that someone wants a ride on this route at this time
Dependency: 7.2.1.3

7.2.2.3 Dual user submission
ID: F7
User Story ID: 2.1.0
Priority: MEDIUM
Description: A logged in user registered as both rider and driver should be able to submit either a ride request or diver instance by specifying if they are driving or riding as shown in Figure 4.
Rationale: In order for driver and rider users to specify which type of user they are for this submission
Dependency: 7.2.1.3

7.2.3 User profiles

7.2.3.1 Profile details
ID: F8
User Story ID: 1.1.0*
Priority: MEDIUM
Description: A logged in user should be able to see and edit some of their profile details as shown in Figure 5.
Rationale: In order for users to update their details in case they change
Dependency: 7.2.1.3

7.2.3.2 Upcoming travel
ID: F9
User Story ID: 4.1.0*
Priority: HIGH
Description: A logged in user who is matched with a rider/driver should see the upcoming routes and details of that route on their profile page as shown in Figure 5 and Figure 6.
Rationale: So that users can be reminded of the details of upcoming routes
Dependency: 7.3.4.1

7.2.3.3 Necessary contact details
ID: F10
User Story ID: 4.1.0*
Priority: HIGH
Description: Upcoming routes should display contact details of that route on the users profile page as shown in Figure 6.
Rationale: So that the driver and rider can find each other
Dependency: 7.3.4.1

7.2.3.4 Upcoming travel
ID: F11
User Story ID: 4.2.0 & 4.3.0
Priority: MEDIUM
Description: A logged in user who is matched with a rider/driver needs to accept or decline matches with riders/drivers as shown in Figure 5.
Rationale: In case the situation changes or the driver thinks that too many extra stops have been added
Dependency: 7.2.3.4

7.2.4 User notification

7.2.4.1 Match notification
ID: F12
User Story ID: 4.1.0

Priority: MEDIUM
Description: When a user is matched with a driver/rider they should be notified in the web app as shown in Figure 4 and Figure 7.
Rationale: So that users don't miss successful matches
Dependency: 7.3.4.1

7.2.5 Web-app administration

7.2.5.1 Admin log in
ID: F13
User Story ID: 5.1.0*
Priority: HIGH
Description: As administrator of the system I should be able to log in and obtain extra privileges as administrator.
Rationale: So that users abusing the system or who want to be removed can be removed
Dependency: 7.2.1.3

7.2.5.2 Removing users
ID: F14
User Story ID: 5.1.0
Priority: HIGH
Description: A logged in administrator should be able to remove users.
Rationale: So that users abusing the system or who want to be removed can be removed
Dependency: 7.2.1.1

7.3 Non-functional requirements

This section contains descriptions of requirements on the user interaction and system performance.

7.3.1 Performance requirements

7.3.1.1 Entering correct username and password
ID: Per1
User Story ID: 1.1.0*
Priority: MEDIUM
Description: A user who enters a correct username and password, once pressing the "log in" button, should not wait longer than 2 seconds to see the submission page shown in Figure 4
Rationale: Users should not wait to long to use the service
Dependency: 7.2.1.3

7.3.1.2 Entering incorrect username and password
ID: Per2
User Story ID: 1.1.0*
Priority: MEDIUM
Description: A user who enters an incorrect username and password, once pressing the "log in" button, should not wait longer than 2 seconds to be prompted for another attempt 4
Rationale: Users should not wait to long to use the service
Dependency: 7.2.1.3

7.3.1.3 Notification of new match
ID: Per3
User Story ID: 4.1.0*
Priority: LOW
Description: After a driver/rider is matched, their profile pages should update accordingly within 10 seconds
Rationale: Avoiding clashes in user actions
Dependency: 7.2.4.1

7.3.2 Reliability requirements

7.3.2.1 User input
ID: R1
User Story ID: none*
Priority: MEDIUM
Description: The webb-app should not crash or freeze due to user input
Rational: So that users can reliably use the system
Dependency: none

7.3.3 Memory requirements

7.3.3.1 Memory usage
ID: M1
User Story ID: none*
Priority: MEDIUM
Description: Running the web-app should never require more than 512 MB of memory from the users browser
Rational: We want users with low end devices to be able to use the service
Dependency: none

7.3.4 Usability requirements

7.3.4.1 Browser Back and forward button
ID: U1
User Story ID: 4.1.0*
Priority: LOW
Description: A user using the back or forward button in their browser should be directed as expected. For example, a user on the submission screen pressing "Profile" as seen in Figure 4 should be taken to the profile page seen in Figure 5. If the user presses "back" in their browser, they should be taken to the submission page. If the users presses forward after this, they are taken to their profile page.
Rational: Having dysfunctional and unsupported navigation gives web-applications a very bad reputation and makes users frustrated
Dependency:

7.3.4.2 Multiple users
ID: U2
User Story ID: none*
Priority: HIGH
Description: Multiple users should be able to be logged in and use the system features in parallel
Rational: This greatly increases the usability and efficiency of the system
Dependency: none

7.3.4.3 User browser and operating system

ID: U3
User Story ID: none*
Priority: LOW
Description: The system should work the same for users using Microsoft Edge, Google Chrome, Mozilla Firefox, or Safari web browsers. It should also be independent from their operating system
Rational: If we discriminate one browser or operating system we risk missing out of 20% of the user base
Dependency: none

7.3.4.4 Mobile support

ID: U4
User Story ID: none*
Priority: LOW
Description: Users should be able to use the web-app from their mobile devices without suffereing from decreased usability compared to using a PC
Rational: Most users will be on the go and require a good mobile user support
Dependency: none

7.3.5 Portability requirements

7.3.5.1 Remote/local host

ID: Por1
User Story ID: none
Priority: HIGH
Description: The system should be able to run in full on a local or remote host
Rational: Enabling the system to run on a local host will make the distributed development simpler
Dependency: none

7.3.5.2 Host operating system

ID: Por2
User Story ID: none*
Priority: HIGH
Description: The system should support a host server of either Windows, Mac, or Linux
Rational: Enabling the system server to be hosted on multiple OS makes it easier for development and distribution to client
Dependency: none

8 Appendix A: Prototypes

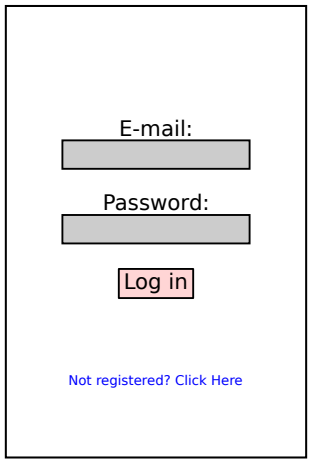

Figure 2: Log-in screen shown to unauthenticated user. The window also guides unregistered users to register

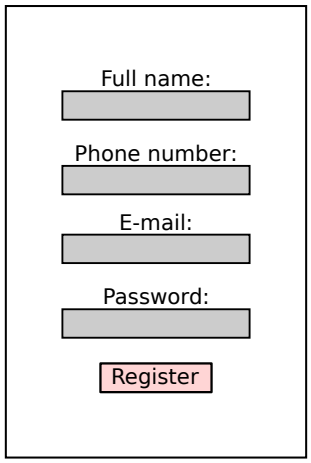

Figure 3: Screen shown to users registering for the first time. A distinction will be made if they register as drivers, as specified in paragraph 7.2.1.2

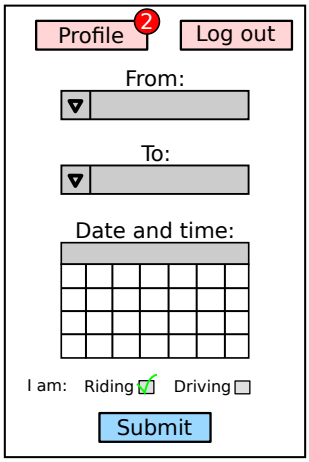

Figure 4: Submission screen shown after successful log-in. Here riders can submit ride requests, and drivers submit drive instances

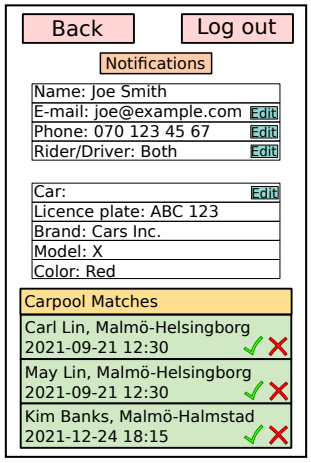

Figure 5: Profile screen showing users their current details and which of these are editable. It also shows upcoming trips and the choice to accept/decline them

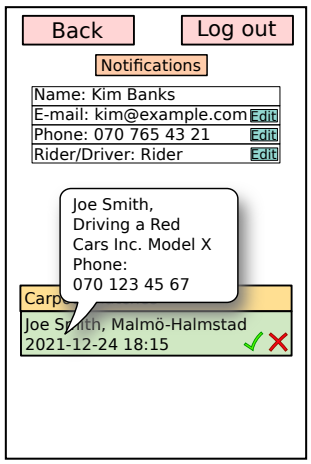

Figure 6: Demonstration of how each upcoming trip can display more information

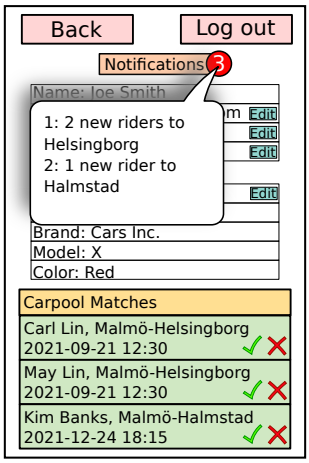

Figure 7: Demonstration of how users will be notified about new events

9 Appendix B: Available routes, and route matching

The available origins and destinations that users can chose will be based on the lists found in "areas.csv" and "districts.csv". These files exist in the same respository as this PRD. "areas.csv" contain the major cities/counties in Skåne and their coordinates. "district.csv" contain the smaller city/county districts and their coordinates.

A driver will pick an origin and destination available in these files, and submit a drive instance together with a value for how much longer the trip is allowed to be with an added carpool rider (tolerance). A rider will pick an origin and destination available in these files and submit a ride request. For the drive instances and ride requests that have matching origins, they are tested for a match. A match is made if the riders destination doesn't deviate more than the tolerance from the drivers original route.

In order to deliver a working prototype early, and to deliver value to our client often, the initial implementation of the route matching will be simplified. In the initial implementation the original driver route will be estimated as a straight line between origin and destination on a flat plane. The deviation induced by a rider will be estimated as a perpendicular line from the original route to the rider destination. This could be done in spherical coordinates, but since an actual driving route is far from a straight line (or arc), spherical coordinates is not our main concern.

There is a major flaw with this straight line approach. For example, a straight line between Malmö and Kristianstad would put the small town of Önneköp right on that line with little to no deviation. But in reality, Önneköp is a 20 minute detour on this route. On the other side of the spectrum, if you draw a straight line between Malmö and Helsingborg, most of that line is across water, and the small town of Fjelie is a 10 km deviation from that line. But in reality, Fjelie is only a 4 min detour on this route, which is a very reasonable detour for a carpool.

We envision the matching to be delivered in three phases. In the first phase we will only support routes between town centers. Drivers and users will match if they have submitted the same origin and destination town centers. In the second phase, we will support routes between town centers and different city districts. In the third phase, we will also match drivers and riders where the rider only joins a part of the original rout, i.e the rider does not share the exact same origin and destination as the driver.