POLITECNICO
MILANO 1863

Travlendar+ project Jerkenhag, Kverne, Rebner

Implementation and Testing document

Deliverable:	Implementation and Testing document
Title:	Implementation and Testing document
Authors:	Jerkenhag, Kverne, Rebner
Version:	1.0
Date:	December 29, 2017
Download page:	https://github.com/Rebnersaurus/JerkenhagKverneRebner
Copyright:	Copyright 2017, Jerkenhag, Kverne, Rebner All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Revision History

Version 1.0, date: 29.12.17

1.2 Purpose and Scope

The purpose of this document is to describe in detail the implementation and corresponding testing for the first prototype of Travlendar+. The prototype will include the basic features that we found the most essential from the previous RASD and DD documents. The following sections will include:

- Essential explanation of definitions and abbreviations.
- A discussion regarding the requirements stated in the RASD document about which features that are implemented in the prototype.
- An explanation about the adopted development frameworks and a discussion about the advantages and disadvantages.
- A description of the source code.
- Detailed explanation about the subsequence testing and the corresponding test cases.
- The installation instructions for Travlendar+ along with a description of the tools required for testing.

1.3 List of Definitions and Abbreviations

- Event - A time block on a day in the calendar with two geolocations, the address of the event and the origin address to get to the event.
- Journey - The complete route to take to arrive at an event.
- Path - One piece of the journey, i.e. in a journey you might first have to take the bus to the train station and then the train to arrive at your meeting. In this case the journey would consist of 2 paths.
- Preferences - Concerns the importance the user places on the following points: Economy, Environment, Dryness and Speed.
- Relevance - When Travlendar+ generates possible journeys for an event they will be ordered by relevance with the most relevant option on top.
- Economy - How important price is in the user for the relevance of a journey.
- Environment - How important carbon footprint of different MoT is for the user for the relevance of a journey.
- Weather Sensitive MoT - A MoT that is an outdoor activity and therefore affected by change in weather, like walking or taking the bike.
- Wetness - How much rain the user will be exposed to for a specific journey.
- Dryness - How unwilling the user is to travel by a weather sensitive MoT if there is a bad weather forecast.

- Flexible Break - Interval of the day that should be kept open for a break of a given duration at some point in this interval.
- S<Transport> - Refers to the shared version of a certain transport mood. (SCar = Shared Car)
- Selected journey - Refers to the journey Travlendar+ suggests or the user manually selects as active for the transportation to an event.
- Speed - The total travel time for a journey.

Acronyms:

- RASD - Requirement Analysis and Specification Document
- DD - Design Document
- DBMS - Database Management System
- MoT - Method of Transportation
- FR - Functional requirements
- NFR - Non Functional requirements

1.4 List of Reference Documents

- Implementation and Testing Assignment document
- Google Directions, API - https://developers.google.com/maps/documentation/directions/
- Weather API - https://openweathermap.org/api
- BikeMi - https://www.bikemi.com/en/stations-map.aspx
  Sidenote: For storing the information about BikeMI pick-up and drop-off locations in Milano.

2 Requirements

2.1 Requirements traceability

2.1.1 Functional Requirements

In this section we will explain how the implementation corresponds to the RASD documents functional requirements. The table will show two columns of Goal ID, FR ID, Testing ID, and then if the FR is implemented or not, and lastly further comments if the requirement is not implemented.

Figure 1: Requirement Figure

Story ID	Goal ID	Functional requirement	Test ID	Comments
FR1	G1	**Calendar requirements** FR1.1 User can access the calendar through a login page. ✓ FR1.2 The calendar should be able to give warnings. ✓ FR1.3 The calendar should show the entire day, week, month and year. ✓ FR1.4 The user should be able to click on different appointment to acquire further details about their journey.✓	I1.1 I3.1 I3.3 I6.2	None
FR2	G2 & G3	**Customize transportation preferences** FR2.1 Support of walking, biking (own, or shared), bus, train, tram, taxis, driving (own or shared) ✓ FR2.2 A user can deselect transportation options ✓ FR2.3 A user can specify maximum walking distance ✓ FR2.4 A user can specify timeslots for using public transportation (from the beginning of the travel). ✓ FR2.5 A user can select the possibility to specify the amount of passengers. ✓ FR2.6 A user should specify if walking or biking in rain is acceptable. ✓ FR2.7 A user should have the possibility to select an environmentally friendly travel, as specified in FR3.1. ✓ FR2.8 The application should give the user the possibility to prioritize their transport favorite, as specified in FR3. ✓ FR2.9 A user can in default settings create their own starting point for calculation of journey. ✓	I4.1 I4.2 I4.3 I4.4 I4.5	For the prototype, walking, biking, driving and public transportation will be supported. However, for the first implementation the different means of transit will not be filtered in the journey. In the event of a user not being able to move by buss, the system will fetch a route with the other means of transit as preferred. If however the route returned is still composed with one path being by buss the whole route will be discarded. Environmental friendly option: the amount of carbon footprint.

FR3	G3	**Prioritize their transportation favorites** FR3.1 The application will, based on carbon footprint, provide the most environmentally friendly traveling option. ✓ FR3.2 The application should get info from the travel companies to calculate the ticket price. ✓ FR3.3 The application should get information from a weather service to have knowledge about optimal travel according to the weather. ✓ FR3.4 If the user does not want to prioritize the standard setting will be to find the quickest travel. ✓	I7.1	The ticket price is based on a standard fee, like metro ticket is 1.50 . The quickest journey: the shortest travel in time.
FR4	G4	**Creating and deleting new appointments in the calendar** FR4.1 The user can create new events. ✓ FR4.1.1 If meetings collide see FR6. ✓ FR4.1.2 If the traveling time between meetings is too short, a warning message should occur. ✓ FR4.1.3 If meetings are created at locations that are unreachable see FR9. ✓ FR4.1.4 The user can specify, when an event is created, the address to travel from and too. ✓ FR4.1.5 The user can specify the event time and duration ✓ FR4.1.6 The user can click generate path and the application will suggest a journey, see FR7. ✓ FR4.2 If a meeting require more traveling time than expected, the app should notify the user. FR4.3 The user should be able to delete events. ✓	I5.1 I5.2 I5.3 I5.4 I5.5 I6.1	FR4.2 is not in the prototype since it is not essential for the application use. It is however relevant for a future improvement.
FR5	G5	**Creating breaks** FR5.1 The user should be able to specify that a break must be possible every day in a specified time slot. ✓ FR5.1.1 The application will always make room for a break in the timeslot. ✓ FR5.1.2 The user should be able to specify the length of the break. ✓ FR5.2 The number of breaks can be specified by the user. ✓	I4.4	None

FR6	G6	**Colliding events** FR6.1 If two meetings collide, a specific warning message will occur. ✓ FR6.2 If a meeting and a break collide, a specific warning message will occur. ✓ FR6.3 The application will give the user the possibility to change the appointment, finding a suitable space in the calendar. ✓	I5.4	Meetings collide: when the user tries to save an event that collides with a pre-existing one.
FR7	G7	**An accurate path description** FR7.1 The application will based on the client's preferences use Google Directions API's to calculate shortest travel. ✓ FR7.2 The application should give the client several options regarding different transportation. ✓	I5.2 I5.5	The application will only provide several traveling options if there are alternatives available.
FR8	G8	**Purchase tickets for public transportation** FR8.1 The application should allow clients to buy public transportation tickets when a bus, train or tram will be taken. ✓ FR8.2 The client will have the possibility to add information about having a day/week/season pass. ✓ FR8.3 The application should also provide information about the nearest car or bake of a vehicle sharing system. ✓ FR8.4 The application should use the client?s payment information to order tickets from the different public transportation companies.	I8.1 I9.1	FR8.1 the prototype will only support ATM FR8.3 the prototype will only support BikeMi service in Milano FR8.4 The prototype does not support any secure payment method, however this is an implementation in the future.
FR9	G9	**Unreachable destination** FR9.1 If a destination is unreachable a warning message will occur. ✓ FR9.1.1 If the destination is unreachable because of strike, the application should provide alternatives. FR9.2 If the destination is not found, a warning message should occur. ✓	I5.3	During the event creation, Travlendar+ will notify the user if the destination is unreachable. FR9.1.1 is not implemented because of lack of good sources of strike data.
FR10	G10	**Creating profile** FR10.1 The client should have the possibility to create a new profile to access the system. ✓ FR10.2 The client should be able to log in with their Facebook or Google account. FR10.3 The client should be asked to leave payment information, so the application can buy tickets. FR10.3.1 The client should be able to proceed using the app without having to leave payment information.	I2.1 I3.1	We did not prioritize 10.1, 10.2, for the prototype, since the features are not essential. Also since FR8.4 was not implemented, FR10.3 could not be fulfilled.

2.1.2 Non-Functional Requirements

The system has implemented the stated security requirements, to ensure security for all users. Also we have confirmed the maintainability and portability by developing a web application that can be used from every device with an up to date browser. See section 6.2 regarding testing.

3 Adopted development frameworks

3.1 Programming Languages

We have designed Travlendar+ as a web based application which is hosted by a web server containing a database. The server can be reached independently and since all information is stored on the database it can always be reached from any hardware with a web browser since the data is stored remotely.

The implementation uses a variety of programming languages. For the layout of the webpage HTML code is used accompanied with CSS for design and aesthetics and JavaScript for interactive functionalities. For all interaction with the database PHP is used and within the database SQL is utilized for query commands. The web development technique Ajax has been used in association with JavaScript for interacting with the PHP code.

The RDBMS MySQL has been used for administering the database of Travlendar+. We have then used PHPMyAdmin for an easy administration of MySQL over the database. PHPMyAdmin has been very convenient since it is a web based tool and therefore accessible from any computer, which has been useful for the project team when working together. The drawback of this is of course that there is no possibility to work or interact with the MySQL database without an internet connection.

3.2 Middleware

We have been utilizing the service HostGator for providing the web server which hosts Travlendar+. The choice behind this was largely based on the fact that one of the project group members had an available server on the service that we could use for this project. For future development an option would be to set up a physical server in order to leverage a faster and more reliable access, also we would be able to use more functionalities like sending emails to the user. However, due to budget constraints HostGator was a good option for the first prototype.

3.3 API

We have used the API's Google Directions and OpenWeatherMap. We have also used information from BikeMi station map for mapping out pick-up and drop-off locations for the shared bike system. However in the implementation we did not find any good sources of Strike data, as we proposed in the DD document. As a result we have not implemented the Strike API table in the database and our prototype does not factor strike when presenting journeys to the user.

4 Structure of source code

The source code for Travlendar+ is located in different environments. Since the whole system consists of the web application and database in the back end different programming language are utilized in the different locations. The image below gives a simple overview over how the source code and the programming languages utilized are distributed across the system.

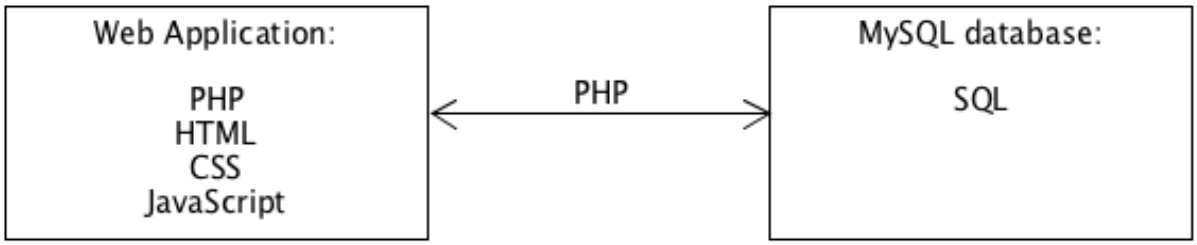

Figure 2: Location of code

The files containing the code for the web application Travlendar+ is located in one folder where it is split up in different files depending on their purpose. Here the code is stored in a number of PHP files which each corresponds to a page or a specific functionality of Travlendar+. Generally these files contains different programming languages namely PHP, HTML, JavaScript and CSS. The HTML code is used for layout if the PHP file refers to a specific page on the application and the CSS code for the esthetics. JavaScript is used for interactive functionalities in collaboration with the PHP code. The PHP code establishes the connection to the database and is associated with the functionalities of the page regarding the interaction with the database. A separate CSS file contains the overarching information about the design and esthetics for the web page and its components. Most of these files deals with some functionality of the web application, the only exception to this is the databaseconnection.php file which only establishes the connection between the web application and the database. This file is stored in a sub folder which is called 'dbc'. The icons that are used in the application are located in a subfolder called 'icons'. These icons are free for use (credit to https://www.flaticon.com) and are placeholder icons for the prototype. They are intended to be replaced in future development by icons created by the project group. Within the MySQL database SQL code is utilized which is contained within triggers for the different tables.

5 Subsequence Testing

The following sections aims to visualize and describe how the implementation steps and the corresponding components were built out. For this we have created a series of images depicting the involved components and their relation for each step.

5.1 Entry Criteria

The integration of Travlendar+ should begin immediately after finishing the DD. It is implied that the RASD and DD is revised and finalized to secure necessary information to all stakeholders about the scope of the project, the settled requirements and design choices. Travlendar+ will be tested periodically at certain milestones during the integration, as the different components of the applications are functional and ready to be tested.

The milestones that are needed for the specific testing includes a fully functional DBMS, in order to ensure secure storage and proper interaction between the main components. Other milestones are the development of the different pages in the application and their interaction with the webserver and database. Also the usage of the APIs such as Google Maps API, needs to reach a certain level of implementation before we can fully test the system. The minimum percentages of completion of the different components before testing is 80 percent.

5.2 Elements to be Integrated

The elements that needs to be integrated for Travlendar+ can be divided in 3 categories: front-end, back-end and external APIs.

- There is only one front-end component which is the web application.
- The web application will be accessible from different means of hardware through a web browser.
- The back-end components consists of a web server hosting the web application as well as a database storing all the information accessible through the application.
- The external APIs are the following; Google Maps API for calculating the journey to be taken, OpenWeatherMap API for providing updates regarding the weather, API for public transportations home pages or communication pages for information about potential strikes that may impact a journey for an event, BikeMi API for including a shared bike option for the prototype.

5.3 Integration Testing Strategy

Our integration testing strategy consists of first creating the shell for the web application which includes building out the different web pages that it will consist of. After this a database will be set-up for storing all the information about the different components. Once this is completed we are aiming to build out the functionalities one after one and test them as the implementation moves along.

The most essential functionality for Travlendar+ is the creation of a journey, however this requires most of the other requirements of Travlendar+ to already be in place in order for it to work correctly. The integration testing will therefore follow a bottom-up strategy, starting with the functionalities that can be tested isolated and finally proceeding to the functionalities dependent on others at the end of the testing phase.

5.4 Sequence of Component/Function integration

This section will describe the integration of the different components and detail their interaction with each other at each implementation step. There are certain dependencies among the components, that will be indicated with an arrow in the following section.

ID	Description	Preconditions
I1	Create a shell of the website without any function.	None
I2	Create all the tables in the database.	Working database
I3	Make the user able to create an account and login.	I2
I4	Make the user able to change its settings.	I3
I5	Make the user able to create an event	I3
I6	Make the user able manage an event	I5
I7	Create a functional weather table.	I2, weather data
I8	Create a functional bike table.	I2, bike data
I9	Create a payment system for ticket purchase	I6

Table 2: Implementation

5.5 Software Integration sequence

The implementation of the overall shell of the application requires the following pages: Create event, Settings and Calendar. The main component that these pages depend on is the web server.

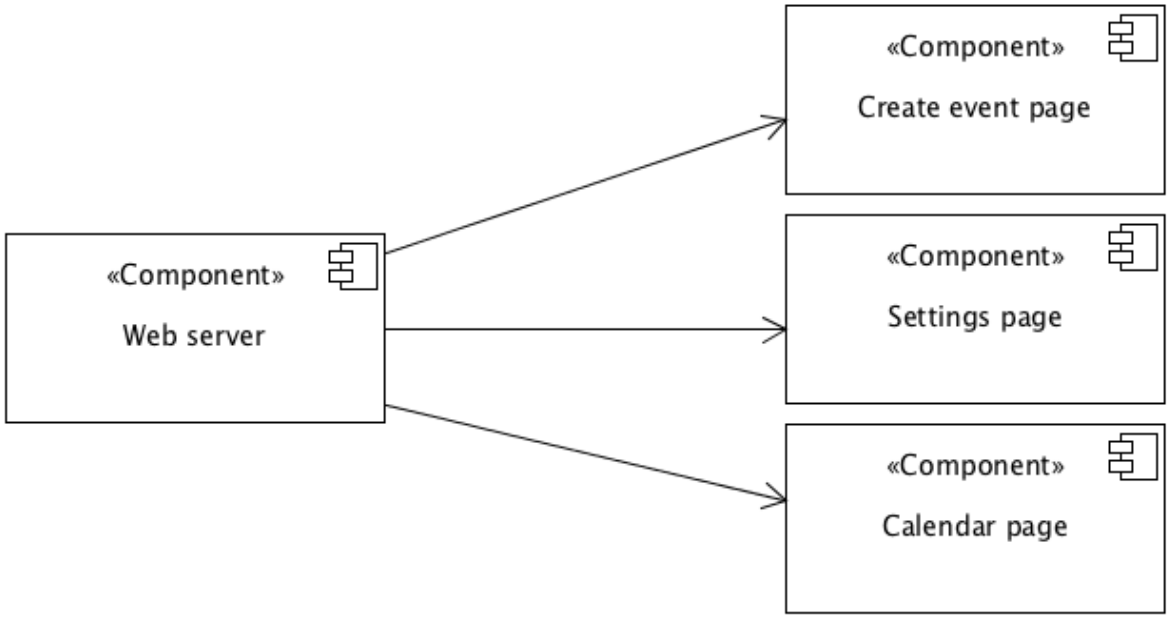

Figure 3: Integration I1

The next step is to implement the database with all the tables corresponding to the data from the different pages in the web server. Here we need a dependency both ways, as the interaction will go back and forth between the two. The web server will update the database with new data when a change occurs,

and the database will send requested data to the web server when it is called for.

Figure 4: Integration I2

Furthermore we have the login and logout sequence. To have a fully functional system the different subcomponents from the web server needs to be implemented. The subcomponents and their interaction is shown in the following figure.

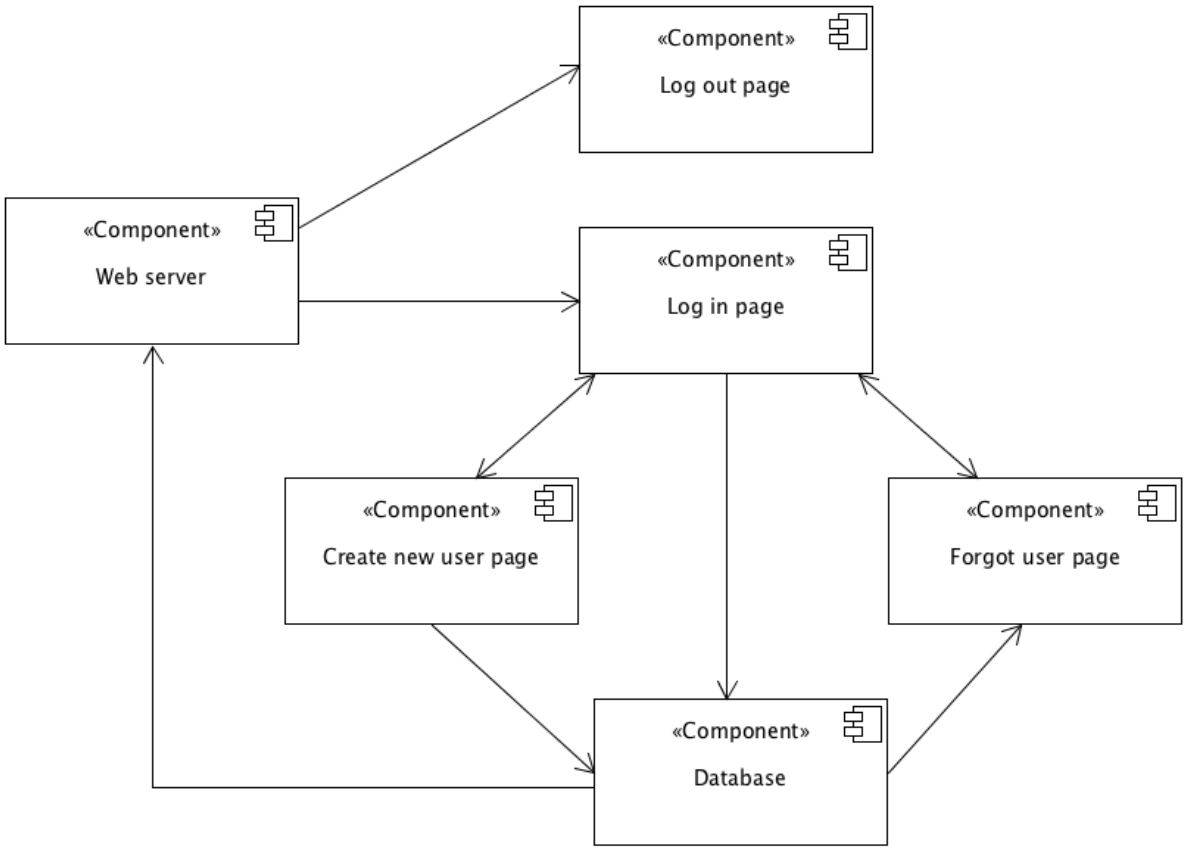

Figure 5: Integration I3

Next we need to implement the settings page, and its interaction with the webserver and database.

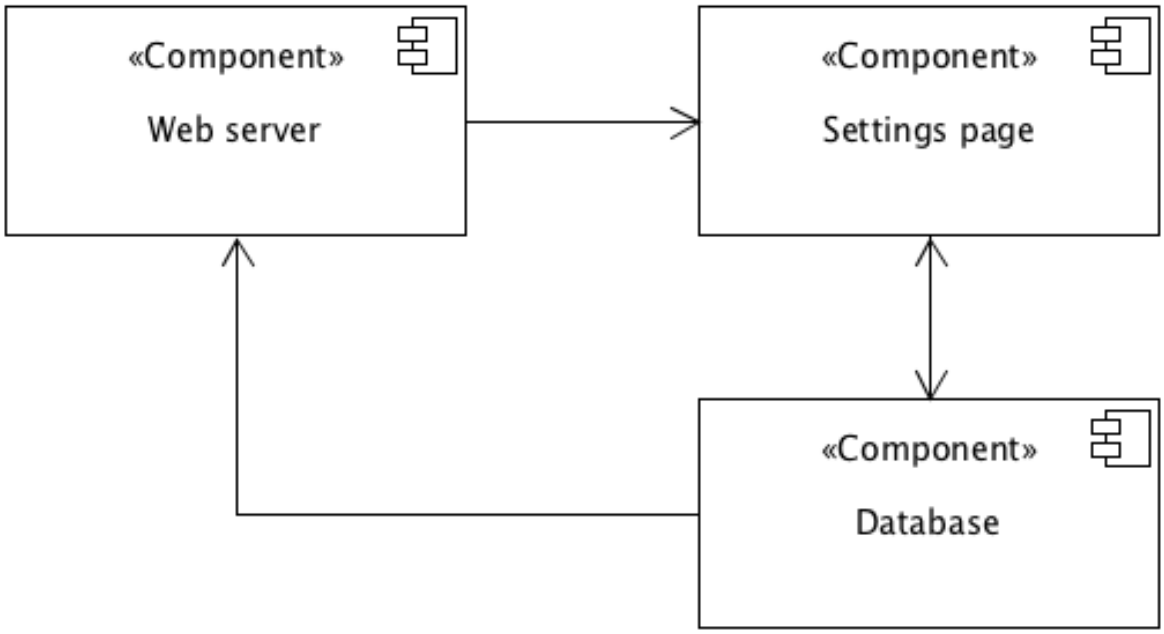

Figure 6: Integration I4

Now we are ready to start implementing: creating a new event. Here we have a primary component: Create event page that interacts with the web server and database.

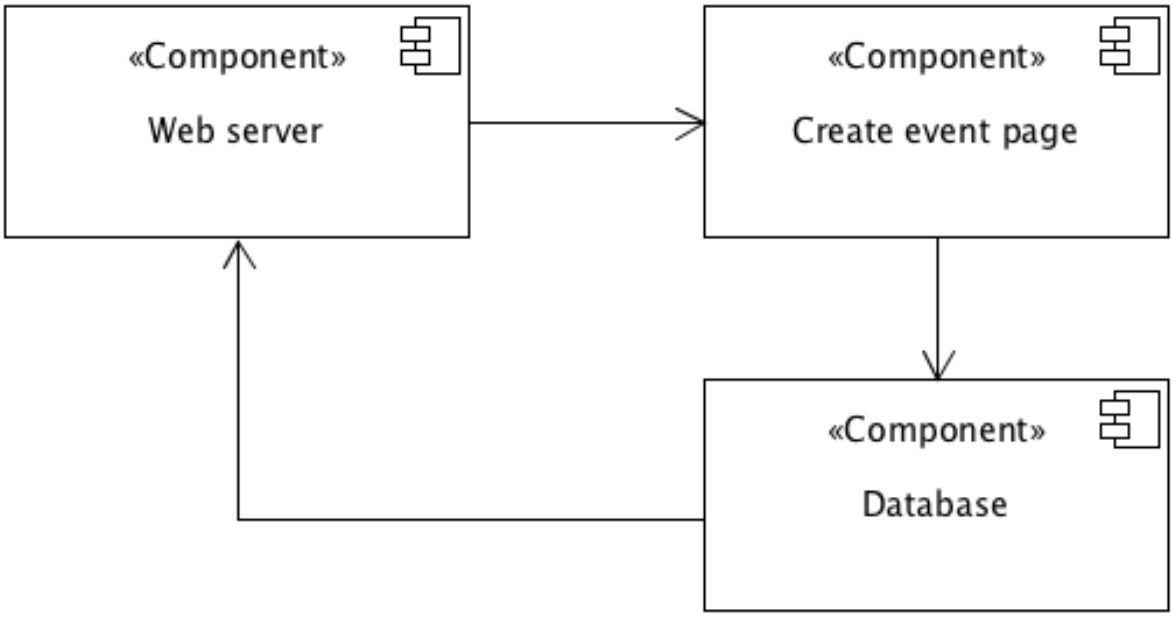

Figure 7: Integration I5

The next step is to make the create event page interact with the calendar page. When an event is created by a user iit should appear in the calendar for that user with the correct information.

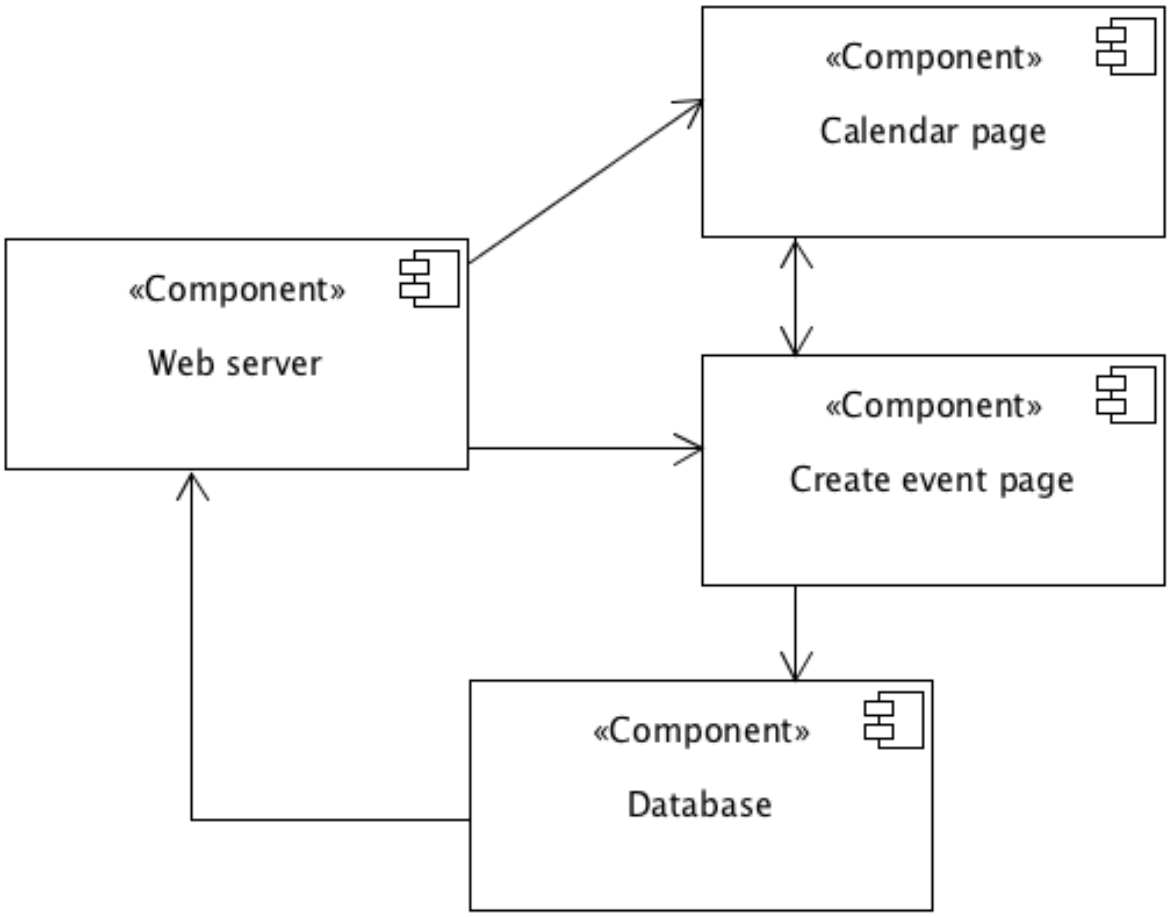

Figure 8: Integration I6

This image depicts the implementation of the weather data API. It should on a hourly basis keep track on the weather forecast and alarm the journey planning if it affected by bad weather.

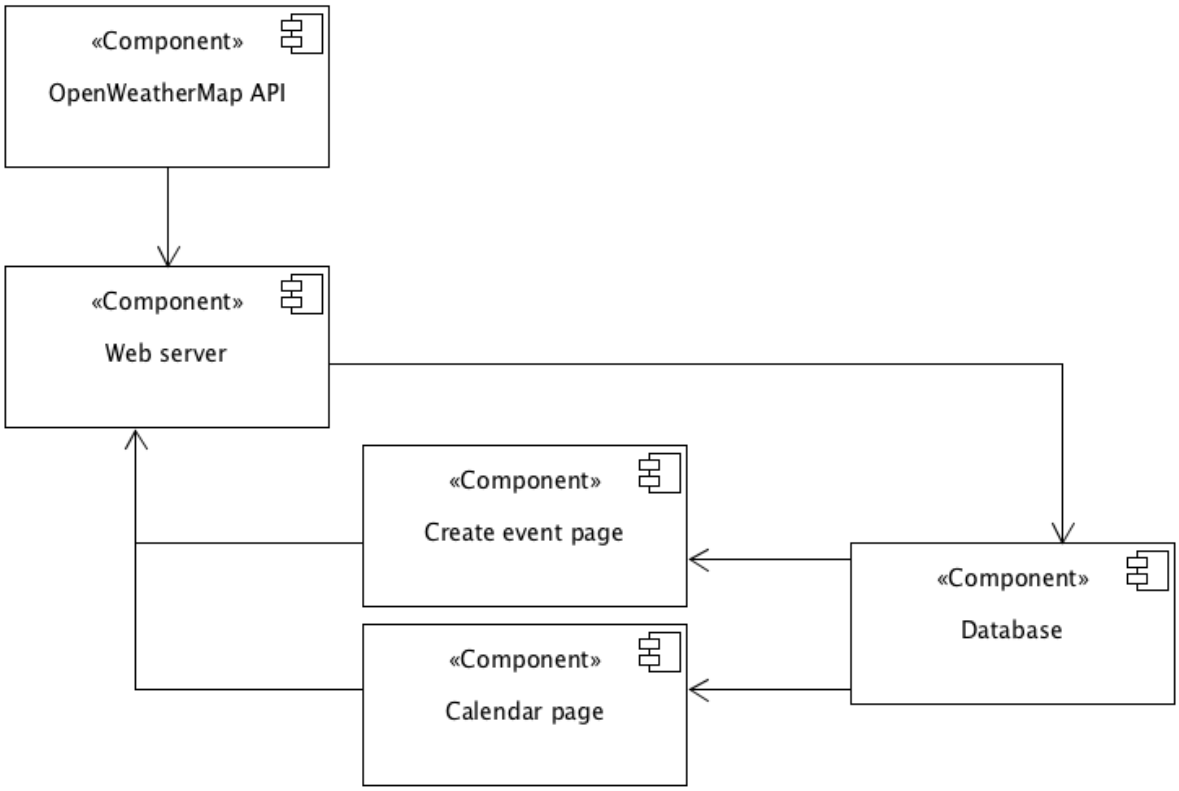

Figure 9: Integration I7

The implementation of the API BikeMi for providing a shared bike service in the Milano region for the prototype.

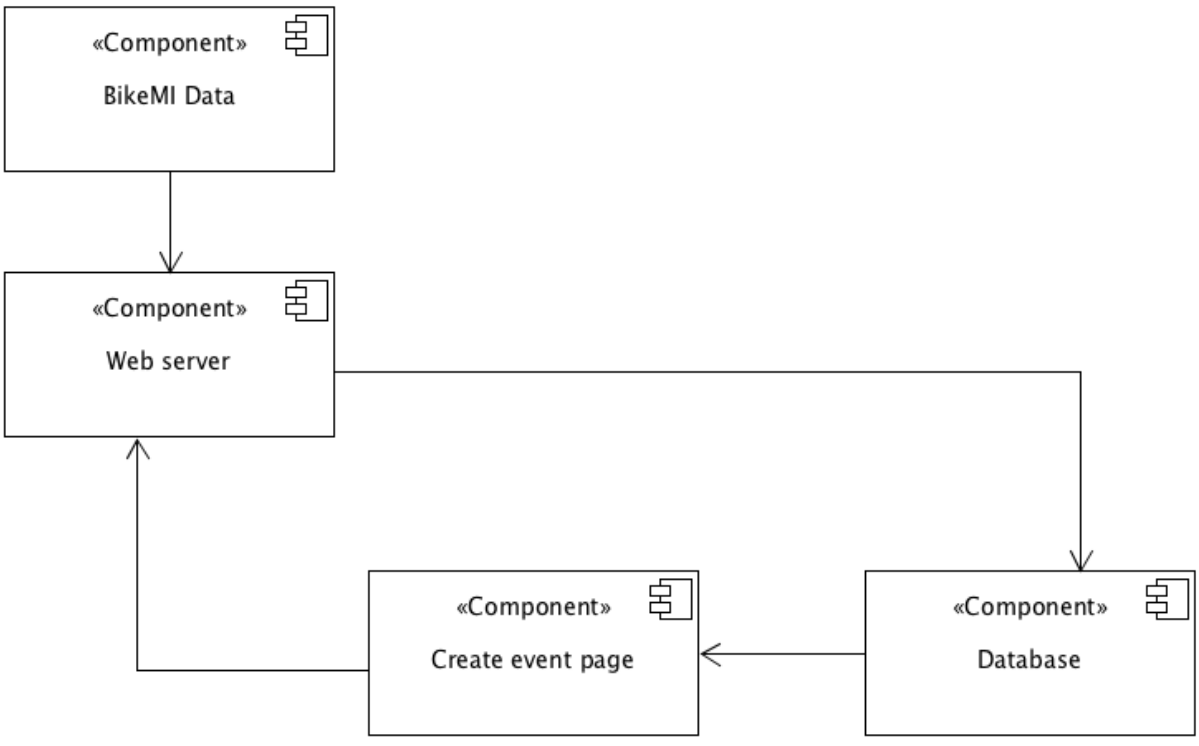

Figure 10: Integration I8

For the final implementation step, I9, no new components needs to be integrated. The ticket purchase functionality will be contained within the calendar page in a created event.

6 Test Case Description

6.1 Functional requirements testing

Figure 11: Story I1

Test ID	I1.1
Components	Web server, Login page, Settings page, Calendar page
Description	The user should be able to correctly navigate in the application. The pages reference accurately to any connected page.
Precondition	None
Input specification	Navigate to the correct folder by writing the correct URL
Output specification	The user is being directed to the correct page.
Assumption	A supported web browser is being used.

Figure 12: Story I2

Test ID	I2.1
Components	Database, Web server
Description	Add user
Precondition	None
Input specification	We created 1000 users and made sure they were correctly stored in the database.
Output specification	The information should be stored in the database.
Assumption	A supported web browser is being used.

Figure 13: Story I3

Test ID	I3.1
Components	Log in page, Web Server, Database
Description	The user should be able to create a new user profile to start using the application.
Precondition	The user is not a registered user.
Input specification	• Navigate to the login site • Click Create new user • Write your preferred username and password • Click Save
Output specification	Successfully created new user, stored in database, and redirected to login page. If the username is in use, an error should occur and the user should be able to choose another username and try again.
Assumption	A supported web browser is being used.

Test ID	I3.2
Components	Log out page,Web Server, Database
Description	The user should be able out log out of the application by clicking the Logout button
Precondition	The user is logged in.
Input specification	• Navigate to the logout folder • Click Logout
Output specification	The user should be successfully logged out of the application.
Assumption	A supported web browser is being used.

Test ID	I3.3
Components	Login page, Web Server, Database
Description	The system has a functional login page, that redirects the user to the calendar.
Precondition	The user is already registered and not logged in.
Input specification	• Navigate to the login site. • Write the correct username and password in the box • Click Login
Output specification	The user is successfully logged in and sent to the calendar page
Assumption	A supported web browser is being used.

Figure 14: Story I4

Test ID	I4.1
Components	Settings, Web Server
Description	The user should be able to change settings and personalize the traveling preferences.
Precondition	Settings page is available and functional.
Input specification	• Navigate to the Settings page. • Activate preferred traveling means. • Click Save
Output specification	The new settings should be successfully stored in the database, in it's own table.
Assumption	A supported web browser is being used.

Test ID	I4.2
Components	Settings, Web Server
Description	The user should be able to choose their preferred max walk distance.
Precondition	Settings page is available and functional.
Input specification	• Navigate to the Settings page. • Determine max walk distance • Click Save
Output specification	The new settings should be successfully stored in the database, in it's own table.
Assumption	A supported web browser is being used.

Test ID	I4.3
Components	Settings, Web Server
Description	The user should be able to determine max biking distance.
Precondition	Settings page is available and functional.
Input specification	• Navigate to the Settings page. • Determine max bike distance • Click Save
Output specification	The new settings should be successfully stored in the database, in it's own table.
Assumption	A supported web browser is being used.

Test ID	I4.4
Components	Settings, Web Server
Description	The user should be able to add breaks of different length and time perspective during the day.
Precondition	Settings page is available and functional.
Input specification	• Navigate to the Settings page. • Choose add break • Determine the length of the break and in what time period of the day the break is supposed to happen. • Click Save (under the break setting)
Output specification	The new settings should be successfully stored in the database, in it's own table.
Assumption	A supported web browser is being used.

Test ID	I4.5
Components	Settings, Web Server
Description	The user should be able to change settings by determining how they prefer to travel. The user can choose between the importance of economical journey, environmental friendly and nice weather.
Precondition	Settings page is available and functional.
Input specification	• Navigate to the Settings page. • Determine the importance among the three preferences. • Click Save
Output specification	The new settings should be successfully stored in the database, in it's own table. If no specifications are made the application will choose the shortest journey in time.
Assumption	A supported web browser is being used.

Figure 15: Story I5

Test ID	I5.1
Components	Create new event, Database, Web server
Description	The user should be able to create events and specify length, time and address.
Precondition	The user is logged in, and has selected preferred way(s) of travel
Input specification	• Navigate to the calendar page • Click Add new event • Specify the details of the event (time, duration, address) • Click Save
Output specification	The new event should appear as an event in the calendar with no journey. And the event should be stored in the database as a new table.
Assumption	A supported web browser is being used.

Test ID	I5.2
Components	Calendar, Google API, create new event, Database, Web server
Description	The user should be able to create a complete functional event with journey to and from the event.
Precondition	The user is logged in, and has activated transportation preferences.
Input specification	• Navigate to the calendar page • Click Add new event • Specify description, time and duration on the event. • Click Generate path • Click Save
Output specification	The new event should be created in the Calendar and stored in the database. If there is a collision a warning message should occur, and the user should have the possibility of changing the details of the event.
Assumption	A supported web browser is being used.

Test ID	I5.3
Components	Web server, add event, calendar, database
Description	A user should be able to create a new event and get a warning message if the event destination is unreachable in transportation.
Precondition	The user is logged in
Input specification	• Navigate to the calendar page • Click Add new event • Specify description, time and duration on the event. • Click Generate path • Click Save • See warning message 'No path available'.
Output specification	The new event should be created in the Calendar with a valid destination.
Assumption	A supported web browser is being used.

Test ID	I5.4
Components	Web server, add event, calendar, database
Description	A user should be able to create a new event and get a warning triangle if the event collides with another event (both meetings and breaks are seen as events).
Precondition	The user must be logged in, and already have another event created at the specific timeslot.
Input specification	• Navigate to the calendar page • Click Add new event • Specify description, time and duration on the event. • Click Generate path • Click Save • Get a warning 'Change time or route'.
Output specification	The event will be created with no collision in the calendar. If the user decides not to change the time slot, a little warning triangle will appear next to the event.
Assumption	A supported web browser is being used.

Test ID	I5.5
Components	Calendar, Add event, Database, Web Server, Google API
Description	A user that calculates a journey should get the most optimal travel based on their preferences and time.
Precondition	The user must be logged in.
Input specification	• Navigate to the calendar page • Click Add new event • Type the desired fields • Click Generate path • Click Save
Output specification	The journey that is shown is based on the users preferences regarding travel. This journey should be the most optimal travel.
Assumption	A supported web browser is being used.

Figure 16: Story I6

Test ID	I6.1
Components	Calendar, Database, Web server
Description	The user should be able to delete events from the calendar.
Precondition	The user is logged in, and has created an event.
Input specification	• Navigate to the calendar page • Click on the event you want to delete • Click Delete
Output specification	The event should be deleted from the calendar and from the database.
Assumption	A supported web browser is being used.

Test ID	I6.2
Components	Calendar, Database, Web server
Description	The user should be able to click on an upcoming event to see detailed information about traveling journey.
Precondition	The user must be logged in and have previously created an event.
Input specification	• Navigate to the calendar page • Click on any upcoming event.
Output specification	The calendar should show the event with the description of an event.
Assumption	A supported web browser is being used.

Figure 17: Story I7

Test ID	I7.1
Components	Weather table API
Description	The application should extract information about the weather so the user can have a dry journey.
Precondition	The API has the correct information.
Input specification	The database pulls information from the API.
Output specification	Manually check if the information stored in the database,
Assumption	A supported web browser is being used.

Figure 18: Story I8

Test ID	I8.1
Components	BikeMi Data, Webserver,
Description	The application is able to find the closest pick-up location, that has available bikes.
Precondition	The BikeMi Data has the correct information.
Input specification	The event gets the closest pick-up and drop-off.
Output specification	The closest location is given. Manually check if the route works.
Assumption	A supported web browser is being used.

Figure 19: Story I9

Test ID	I9.1
Components	Calendar, Database, Web Server
Description	A user that needs public transportation should have the possibility to purchase tickets in the application.
Precondition	The user must be logged in, and created an event that requires public transportation.
Input specification	• Navigate to Calendar • Click on the event that requires transportation • Click buy ticket
Output specification	The user is redirected with a link to the transportation company's public ticket-purchase online.
Assumption	A supported browser is being used. The transportation company has a possibility to buy tickets online.

6.2 Non - Functional requirements testing

For the first prototype we have not prioritized to test the non-functional requirements even though we have taken measures to satisfy as many as possible during the implementation. Please refer to the RASD document to see the specific NFR requirement.

7 Installation instructions

Since Travlendar+ is a web application it is intended to be supported on any web browser. For testing the initial prototype we suggest the user to utilize a regular computer as opposed to a smart phone since the prototype needs to be hosted locally.

For installation and testing of the prototype the user needs to host an apache server and a MySQL server on their computer. Softwares we recommend for this is XAMPP which have versions available for Windows, Linux and OS X. However, the project group had some issues with XAMPP for OS X and used the MAMP software instead which is created more specifically for OS X systems. Both of these softwares are available for free for download online.

With one of these softwares installed, the user will then need to be granted access to the JerkenhagKverneRebner Travlendar+ files and should be added to the htdocs map in the XAMPP/MAMP file directory. These files will be granted directly by the project group to any user interested in testing Travlendar+. They can also be found in the project groups GitHub repository, instructions and link below.

7.1 General instructions

Once you have started the apache server and the MySQL server type, in your web browser, 'localhost/-Travlendar' to get to the web application. In order to get the application to interact with the database go to 'localhost/phpmyadmin' and create a new database from the file in the project.

7.2 Specific instructions for server hosting application

7.2.1 XAMPP instructions

When XAMPP has been installed on the users computer and the Travlendar+ files has been acquired from the project group and added to the htdocs the application can be tested. In XAMPP select 'Start' in the 'General' tab. Once the status is active and the IP address is shown navigate to the 'Services' tab and start to initialize the Apache and MySQL server. From here, go through the steps of general instructions above to reach Travlendar+.

Once the user has arrived at this page they can create a new user profile and log in to the application. From here the different functionalities implemented in the prototype can be tested by following the test cases we have detailed. To reach the back-end database go to 'localhost/phpmyadmin'. Here the user can view the tables of the database for Travlendar+ as well as their associated SQL triggers.

7.2.2 MAMP instructions

Once MAMP is running on the users computer, the user should select 'Start Servers' in order to initialize the apache server and the MySQL server. Then select 'Open Webstart page', this will bring the user to the MAMP webstart page where the user can navigate both to the user interface of Travlendar+ as well as the MySQL database in the back-end. To get to the application select 'My Website' which will take the user to the localhost page where Travlendar+ is located. Once the user has arrived at this page they can create a new user profile and log in to the application. From here the different functionalities implemented in the prototype can be tested by following the test cases we have detailed.

To get to the SQL database in order to view the back-end information stored in the tables, select 'phpMyAdmin'. Once within phpMyAdmin the user can open the 'travlendar' tree and view the tables and their contents.

7.3 Tools Required

- JerkenhagKverneRebner Travlendar+ files - Link: https://github.com/Rebnersaurus/JerkenhagKverneRebner Sidenote: Download the folder 'Implementation' which contains all the files for the web application and for the database. Once the folder has been downloaded please insert it into the 'htdocs' folder within your server hosting application.
- XAMPP. Link: https://www.apachefriends.org/download.html

  Alternatively for OS X
- MAMP. Link: https://www.mamp.info/en/downloads/ Sidenote: On this page, both MAMP and MAMP PRO are available. For testing Travlendar+ the paid version MAMP PRO is not required.

