# Wheretrip.com

Bachelor Thesis

R. Bergström
M. Simidžioski
G Spek

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

# WHERETRIP.COM

## BACHELOR THESIS

by

**R. Bergström**
**M. Simidžioski**
**G Spek**

in partial fulfillment of the requirements for the degree of

**Bachelor of Science**
in Computer Science

at the Delft University of Technology,

| | | |
|---|---|---|
| Project Duration: | November 13, 2017 - February 1, 2018 | |
| Supervisor: | dr. ir. O. Visser | |
| Thesis committee: | dr. ir. O. Visser, | TU Delft |
| | Asst. Prof. H. Wang, | TU Delft |
| | S. van der Helm, | Wheretrip |

*This thesis is confidential and only the latest version can be made public after the 1st of February, 2018*



TU Delft Delft University of Technology

# PREFACE

This is the report for the completion of our (everyone except Rasmus who is an exchange student) Bachelor degrees in Computer Science. It documents a ten week project for the startup company Wheretrip.

The project involved a full rewrite of the code base of the company, with many improvements for extensibility and modularity. Though it failed to meet the goals set forth at its outset, we are of the opinion that we have drastically improved the quality of the program.

We would like to thank our supervisors at Wheretrip, Sando van der Helm and Wikaas Dihalu for believing in us and giving us this opportunity. We would also like to mention Rohan Sobha and Daryll Lobman, interns at Wheretrip who gave ample of their time to bring us up to speed with the state of the project.

Thanks also to TU Delft and to our supervisor, dr.ir. Otto Visser.

*R. Bergström*
*M. Simidžioski*
*G Spek*
*Delft, February 2018*

# Contents

# 1

# INTRODUCTION

## 1.1. THE COMPANY

Wheretrip, describing themselves as,

"A Dutch based start-up for travelers who like to explore new places, offering countless destinations within your budget. Being an official YES!Delft student start-up we have created our platform with the help of the best in-house experts available.

With our diverse team of experienced travelers we have carefully selected the best destinations for every type of trip: Whether you are a party animal, surfer or both, we have got you covered! By combining our wanderlust with our smart algorithms, we offer you trips which truly fit your theme and budget" [1].

The company wanted to improve their (unpublished) website, and having had good experiences with BEP earlier, they decided to reach out to the Bachelor Program again. They wanted to propose two major improvements in how things are done, namely: implementing an integrated booking system and personalizing the current search engine. A more detailed description of these two components can be found in the following section.

## 1.2. PROBLEM DESCRIPTION

### 1.2.1. INTEGRATED BOOKING

In order to make revenue out of trips made through Wheretrip, the company thought of monetization options. One of the main ideas is asking a commission on the total price of the trip. To be able to charge this commission it is required that an integrated booking system is implemented within the platform itself. In the current situation the customer is directed off the platform and is responsible to book the trip themselves. This is not ideal since there is no confirmation whether the customer actually booked a trip through Wheretrip. This booking system would streamline this experience for customers on the platform, since they would not have to pay at both the transport provider as well as the accommodation provider.

### 1.2.2. PERSONALIZATION

Wheretrip wants to offer their customers the best possible experience, and their target clients are mainly students and other travelers with small budgets. Thus, they created a system that revolves around the budget of their customers. Having realized, however, that money is not the only parameter relevant to people, as there are other that should be accounted for in the trip selection process. As the next step on their mission, they wanted to develop a set of other relevant parameters for the customer to specify. The system should then provide them with the best possible trip suggestions, based on their selection.

## 1.3. OVERVIEW

This report outlines the requirements first brought by the company and the research performed to determine their feasibility. It then covers the revisions made to said requirements following the research and the resultant plan of approach, and how this plan was carried out.

# 2

# Initial Requirements

## 2.1. Features

The company was not satisfied with the current options that their search engine provided to the users and felt that there could be an upgrade to the features that were available. To satisfy the wishes of their users, they made a survey with question about the current state of the website and asked the users to give feedback and propose improvements for the current state of the search engine. The results of this survey showed that indeed the users wished for more options to be added to the search engines in order to make their search experience better.

The first feature was to have an option searching flexible dates on which they could take their trip. To add this feature, several options are possible. The first option that the user can select a departure date and get also results for +/- 3 or more days. Another option is to let the user select a specific date and then show a monthly calendar with prices for all dates of that month. This way the user will have a clear overview and can find the best option easier. Yet another option is to let the user select that he/she has no preferred date option, but instead only select a destination and show results of the cheapest flights to that destination at anytime.

The second feature that was proposed by the users was to make it possible to choose a specific accommodation type instead of getting results of all available accommodations. The user will have a choice menu where he/she can select the desired accommodation type and based on that get filtered results for that specific type.

The third feature that was proposed by the users was to include general information and an picture of the selected destination. This will mean that after a user has chosen a destination, in the results for that destination, an information section will be included that will contain general information about local transportation, activities, language, etc. and a picture.

The fourth feature that was proposed by the users was to include an option to specify the means of transportation to a travel destination. The user will have a choice menu that shows different transportation types (train, bus, airplane) and accordingly results will be shown.

The fifth proposed feature was to also include weather information. When a user selects a date and destination for the trip, a weather map is shown that gives an overview of the weather conditions at the time of the trip and if possible additional information about clouds, wind, pressure and air pollution.

The last feature that the users proposed was to get more results for their search. This will mean that different travel agencies and accommodation/transportation providers need to be added to the current search options.

## 2.2. Integrated Booking

The company also wanted to extend the current product with an integrated booking system. This extension would make it possible for the users to book a trip at the company's website instead of being redirected to other flight and accommodation providers websites. In order to implement such an integrated booking system, the website needed to be PCI-compliant, otherwise credit-card data could not be processed. Since WhereTrip was not PCI-compliant, the first step was to make it PCI-compliant. After this the integrated booking system could be implemented. The company wanted the booking system to include several features.

The first and most important feature was that the booking system would allow a user to book both the flight and the accommodation directly on their website. This feature would make the booking process faster and easier because the user would not have to be redirected twice (once to the flight website and once to the accommodation website) in order to book a desired trip.

The other features included entering and processing payment information on the website, entering and processing the passenger booking information and luggage on the website, adding a commission to the overall price of the booked trip, enabling the processing of credit card data on the website and sending a confirmation email to the user when the booking is finished.

# 3

# RESEARCH

In the summer, another BEP group performed a similar project to this one for Wheretrip. In this section the background information for the feasibility study is given, that is crucial to understand the project and the plan of approach. It also includes a description of the state of the project was when we found it and a more detailed elaboration on the findings of the research phase.

## 3.1. INITIAL FEATURES

This section briefly discusses which features had already been implemented at the beginning of the project. Mainly, these concern selection and information features. These will be discussed in that order.

When opening wheretrip.com, the user was offered the selection of a theme (sun, surfing, winter, party or city) as well as specifics for the trip. These were implemented using a static list of cities, pre-sorted into the specific theme categories. Factors like seasons were not taken into account, meaning that you could be sent for a winter semester to Sweden in the summer, or to a beach when the temperature was around 10°C.

No choice of a destination location was offered, just that of a departure location. The website would then offer flights from the nearest airport. Other options included date range, price range and the number of passengers. Finally, once a destination had been selected, the option of changing the selected accommodation at the destination to a different one was provided.

With regards to the information features, as soon as the search was completed, the user could see a map displaying the location of and a grid displaying the information about viable trip options. This information included trip price, destination city and country, an image and times for arrival and departure. When having selected a specific trip, the user was also supplied the price of the flight and the flight and the accommodation, along with information about the destination and links for booking.

## 3.2. THE STATE OF THE CODEBASE

### 3.2.1. STATE OF THE BACK END

After the initial talk with WhereTrip it became apparent that most of the server code was written by the developers of the previous BEP assignment. According to WhereTrip and the report the code-base was written with scalability in mind. They talked about the project as though it would involve mainly adding features and extending the previous work. It was not until the end of the first week that we finally got access to the code base.

It immediately became apparent that if the project had been scalable when the previous BEP group had delivered it to Wheretrip, it wasn't now. New developers had been tasked with adding features to the codebase, and the structure and design of the previous BEP group were gone or quickly disappearing. This had happened due to a focus on rapidly adding more features to the system, at the expense of good coding practices.

The part of the server that was written as part of the bachelor end project was developed for a legacy front-end, which has been replaced recently as mentioned below. Unfortunately, the previous project did not ship a fully functional application, which meant that the developers had to merge both the front-end and back-end

together. Here decoupling was not prioritized, leading to a setup, in which the back end was serving the front end files directly, instead of exposing the back-end logic through certain endpoints.

### 3.2.2. STATE OF THE FRONT END

The front end had been built in React, but was nowhere near the standards that the description by the company would let one believe. It was designed such that each page was its own individual application (instead of being a much more maintainable and extensible single-page application) and served by the *same* express server that was running the back end. There had been absolutely no tests implemented for the front end. To keep track of state that was to be shared between the various pages, Redux [2] was used.

Each page being its own application means that adding a component requires you to manually add its HTTP-requests to the server, which is *not scalable*. In Wheretrip's case, it had also resulted in the same code being implemented multiple times at different places, which had made the code *difficult to maintain*. Unless given its proper attention, these circumstances can lead to the code falling into a state of non-maintenance, as developers tend to avoid such code.

Another consequence is that the server has to read in a completely new page multiple times as the user navigates the website. This results in a quite *uneven navigation experience* compared to using react-router to make the site into a single-page application.

Also, because each page is totally decoupled, neither React, nor Redux has a way of carrying state between the various pages. All information must thus be produced by the server at each step. This results in a huge amount of boilerplate code, which has to be rewritten each time a new page or container is to be added.

The individual components, containers and reducers worked well and were reasonably well thought out, but the separation of concerns was unclear. Furthermore, the structure was not designed to be the view and the controller of a restful back end model, instead it was rendered and controlled by the server.

The difference in programming paradigm and lack of centralized app state meant that all components would have to be rewritten at some level in order to facilitate a common app state and decoupling from the server and achieving a much greater level of extensibility.

## 3.3. PCI-DSS COMPLIANCE

The *Payment Card Industry Data Security Standard*, also known as the PCI DSS or PCI for short, is a regulation that aims to protect consumers credit card information from theft or disclosure. Any organization that stores, processes and transmits any sort of credit card information needs to comply with the PCI DSS [3].

### 3.3.1. WHEN DO YOU NEED TO BE COMPLIANT?

If your business accepts payment, then you are required to be PCI compliant at various levels, as determined by your transaction volume. PCI compliance levels generally apply differently to different cardholders brands like American Express and Visa. These levels are determined by the number of credit card transaction, processed per year.

For example, the following table shows the amount of Visa transactions needed per year for a merchant to fit each of the levels,

| Level | Transactions/year |
|---|---|
| Level 1 | More than 6 000 000 |
| Level 2 | 1 000 000 - 6 000 000 |
| Level 3 | 20 000 - 1 000 000 |
| Level 4 | Less than 20 000 |

Figure 3.1: Amount of Visa transactions in each of the categories [4]

For each level of merchant compliance there are specific requirements, such as either an on site assessment by an actual PCI-QSA (level 1) or self-assessing via the Self-Assessment Questionnaires (SAQ) for levels 2 to 4 [3].

### 3.3.2. HOW DO YOU BECOME PCI COMPLIANT?

The following measures are among those included in the PCI Data Security Standard,

- Building and maintaining a secure network

- Protect Card holder Data
- Maintain a Vulnerability Management Program.
- Implement Strong Access Control Measures
- Maintain an Information Security Policy

For a small business there are many solutions to make compliance easier to achieve. These so called payment-providers offer a portal at which the credit-card details are entered and so no actual credit-card data has to flow to the business' servers itself. However this only limits the scope of PCI Compliance, the business still needs to do the SAQ and maintain the strong access control measures.

### 3.3.3. PCI COMPLIANCE AND WHERETRIP

After careful evaluation it was determined that it, granted it would be a lot of work, might be possible to make Wheretrip PCI-compliant within the time frame available. This would involve stopping all other possible projects and focusing solely on making the site adhere to the PCI Data Security Standard. This was suggested to Wheretrip management, and rejected on the basis that other concerns were seen as more urgent.

This was then brought back to the BEP supervisor, with whom it was decided that, albeit beyond feasibility to have the system entirely within the scope of the assignment, great steps ought still to be made in that direction, to simplify the process for whomever would later endeavor to make an integrated booking system a reality.

Thus, a complete refactoring of the front end and most of the back end was recommended, as well as looking over hosting and deployment, to comply with the PCI DSS.

## 3.4. FEATURES

Aspiring to fulfill the goal of the Wheretrip platform, it would be pertinent to provide a personalized offer to the prospective travelers to find the most suitable options of destination within a given budget. To make this happen as efficient with regards to time, it was decided to make use of some currently available resources and API's that provide the basic functionalities needed for Wheretrip. Currently, Wheretrip employs two APIs that work together to make sure the customer can book a trip, the Kiwi API for booking flights and the HostelWorld API to find accommodations.

There are some shortcomings with the current APIs. For example, HostelWorld offers only hostels, no hotels, bed & breakfast etc. Another problem that was encountered in one of the API's was that the system did not always provide the best (cheapest) result. Therefore, it was decided to conduct research regarding alternative flight and accommodation APIs. Research was also conducted about other possible features, such as choosing the means of transportation of the trip, providing general information and images of the selected destination, adding a calendar view with the cheapest price for a selected destination and providing information about the current weather conditions.

In order to derive to a good choice of APIs for the data needed to implement the various features, comparison tables were constructed for each feature where the functionalities of the found alternatives are listed. The next section provides information about existing personalization systems and conducted literature in this field.

### 3.4.1. RELEVANCE

Travel planning is a multi-faceted decision process consisting of choosing a destination and grouping together tourism products and services (attractions, accommodations, and activities) closely related to the destination[5]. The search process is one of the crucial steps of finding the desired result.

One of the main reasons that companies use personalized search is to improve the choices of the customers and also attract more customers to buy their products. This is also true for the tourism industry. A great amount of research is conducted on the choices that are made when choosing a travel destination. Some individuals prefer an all-inclusive package where everything is regulated for them beforehand while others are more flexible and search for all components of the trip separately.

To be able to design a search tool that helps the customer to find a desired travel destination several components need to be taken into consideration, namely: the travel period, the desired activities, landscape and weather of the destination, the budget of the customer and the transportation and accommodation. Jansen and Pooch argue that based on the considerations of the customer, whether the customer has already

decided for a specific travel destination or he/she is looking for different options and recommendations, the keywords entered into the search engine will be significantly different[6]. A customer that has not decided for a specific destination would not type in the destination name as the keyword in the engine but instead would want to get results based on other variables.

The current search engine does not allow a personalized search query. Tailoring and personalizing the search engine of the Wheretrip website to fit the needs of the customer is a step that needs to be taken to upgrade the current situation of the website. There are several proposed models: Choice set models[7, 8], general travel models[9], decision net models[10], and multi-destination travel models[11]. One such model is the NutKing Recommendation system. The system consists of three main parts[5]:

1. Acquisition of travel preferences: in this part the collaborative features are collected. The customer needs to provide information about individual preferences such as the travel period, the means of transportation, the budget, the types of accommodation, the desired activities and previous knowledge of the travel destination.

2. Search for travel product: in the second part the content features are defined. Now the customer has to fine-tune the query and define the requirements for the travel destination. This includes the landscape or nature characteristic of the location (nearby beach, mountains, desert, city center) and the type of activities (exhibition, sport activities, etc.).

3. Choice: the third part returns the results of the query. Based on the previously acquired information, the results are sorted using similarity ranking criteria and presented to the customer.

In order to design a personalized search for the Wheretrip website, different API's were necessary. The company had several requirements and preferences that needed to be fulfilled when choosing an API of which the most important was that the API is free of charge. The results found after exploration and analysis of the various APIs are found in the next section.

### 3.4.2. APIs

In this section the APIs that were found to personalize the search engine are presented and evaluated. For each category, one alternative was chosen that suited the needs of the website at this moment. There are of course limitations, and there may be better alternatives to the chosen ones. For each alternative a brief explanation is given for why it is chosen. A detailed evaluation of all alternative APIs can be found in the Appendix B.

1 Accommodation
SkyScanner was chosen for the purpose of adding different types of accommodations to the filtering options of the search engine. The main reason why SkyScanner was chosen over the other alternatives is the availability and integration of the API. The main contender, Expedia, Inc. has a wider choice of accommodation types, the company did not allow their API to be used by partners unless the partners are PCI compliant.

2 Date Flexibility
SkyScanner was chosen for the purpose of upgrading the current search engine with the option of selecting flexible dates and also creating a calendar that will show the cheapest prices of a selected trip per month. The main reason that SkyScanner was chosen was because it provided the desired date flexibility options and also a monthly calendar with the cheapest tickets for a specified destination. Again, Dohop could also be an option because it provides a lot of filtering options, but since they now require travel payouts to be implemented this alternative will not be used.

3 Travel Types
GoEuro was chosen for the purpose of upgrading the current search options which are only focused on flights, with different means of travel such as a train and a bus. There were not many alternatives available where an option existed to select the means of transportation to the travel destination. Booking.com and GoEuro were the two best alternatives for this category. They also provide very similar filtering options, the choice of GoEuro over Booking.com was made because Booking.com did not allow their API to be used by partners unless the partners are PCI compliant.

4 Weather
Open Weather Map was chosen for the purpose of integrating a weather map that will show the information of the weather conditions at a selected destination in a specific time frame. Many other alternatives existed for this category. One of the preferences of the company was that the chosen APIs are free. Open Weather Map has similar options to Weather Underground and Dark Sky, but the main reason that it was chosen was the amount of API calls that can be made with a free account. The limit of API calls that Weather Underground allowed was 500 per day, the limit of API calls that Dark Sky allowed was 1000 per day and the limit of Open Weather Map was 60 per minute which is significantly more than the others.

5 Travel Information
Teleport Public was chosen for the purpose of retrieving additional information about a selected destination in the search engine. The other comparable alternatives were TripAdvisor and BaseTrip. Although TripAdvisor and BaseTrip also provided the necessary general information, BaseTrip had no free API. The Teleport Public API had no hard limit on the number of calls and is easy to integrate into the current system.

6 Images
Teleport Public was also chosen for the purpose of providing an image of the destination additional to the general information. The other alternatives that also provided very nice pictures of different cities and destinations were Pixabay, GettyImages and Unsplash. Unsplash had a requirement to tell them how their API is used in the website in order to allow the full API call amount. Otherwise the limit of the API calls was set to 50 calls per hour. GettyImages had a requirement that their content could only be used for non-commercial purposes and Pixabay had a requirement to add a link to their website and also did not allow hotlinking of the pictures. Because of these mentioned reasons, Teleport Public was chosen over the other alternatives. Also another benefit is that Teleport Public can be used for both providing general information and a picture.

## 3.5. AREAS OF FOCUS

### 3.5.1. FEATURES

Because there were different features that the company wanted to be added to their existing product, these were divided into several categories. The first category was Accommodation where the main focus lied in creating an option to select different types of accommodations (hostel, bed & breakfast, etc). The second category was Travel Types where the focus was to add different means of transport to the search option menu. The third category was Date Flexibility where the focus was to add an option to get results for days or even a month with cheapest available tickets additional to the specified date. The fourth category was Travel Information where the focus was to add general information and a picture for each destination. The fifth category was weather where the focus was to add information and predictions about the weather conditions at a specified destination.

### 3.5.2. FRONT END

The main issues of the front end where that it had no tests at all and was not scalable. The first step into re-factoring the front end was to find a suitable framework which allows both extensibility and modularity. The other factors that needed to be taken into account included testing the logic, components and browser of the product and making the product scalable.

### 3.5.3. BACK END

The back end was re-factored by a previous BEP group that collaborated with the same company. But currently, the back end was also in a not so good state anymore and needed some improvements. In order to re-factor the back end a number of factors needed to be taken into account including the scalability of the code, creating a decent database model, caching often-used data, decoupling the front and back end and testing the logic and API.

### 3.5.4. HOSTING

The product was hosted on a private server and the web server wasn't even correctly configured to serve the application on `wheretrip.com` instead port 3005 needed to be added. Furthermore deployment had

to be done manually through *FTP* and there was no continuous integration implemented to validate that the application is still working with every change of the code base. As the company would like to have the integrated booking system implemented in the future, effort will be put in migrating the application to a more secure means of hosting together with continuous integration and deployment.

# 4

# PLAN OF APPROACH

As discussed in the previous chapter, it was deemed unfeasible to make the website PCI-compliant and extend it with an integrated booking system within the given time frame. The proposal was made to Wheretrip to make the system PCI-compliant, meaning that the integrated booking system and personalization features would not be implemented. This was presented to and discussed with the company, which decided that the personalization of was their highest priority at the moment.

Furthermore, as also mentioned in the previous chapter, all parts of the project, both server and client, were in dire need of refactoring. Thus, in order to achieve any measure of quality result, it was determined to make a full rewrite of the project.

This chapter lays out the plan of approach as it was finally adopted, discussing requirements, time plan and criteria for success.

## 4.1. REQUIREMENTS

When discussing the goals applicable to an endeavor, the MoSCoW method can be used to establish common agreement among the stakeholders for which importance is placed on each requirement. MoSCoW is actually an acronym, defining four levels of priority, Must Have, Should Have, Could Have and Won't Have. The requirements of this project were organized according to the MosCoW principle. A complete list of the amended requirements can be found in appendix A.

The success criteria for this assignment were narrowed down to the following parts.

- Meet all of the Must Have requirements.
- Meet 80% of the Should Have requirements.
- Make the underlying structure production ready.
- Pass the project on in a way, such that the programmers at Wheretrip can continue to implement it.

### 4.1.1. MUST HAVE

To make the application scalable, the code of both the front end and back end has to be rewritten to adhere to sound software practices. Specifically, the client and server need too be decoupled, the back end must be re-imagined with a proper database model, accounting for the prospect of adding additional transport and accommodation providers, the entire code base needs appropriate testing and hosting and continuous integration pipelines need to be established.

In order to comply with Wheretrip's desires, the most requested features according to their recent user surveys must also be introduced. Common points made by customers included the option of additional date flexibility, such that a duration could be selected from a larger date range, the options of different types of accommodations (hotels, airbnb etc.) and adding other travel providers to the search. Other Must Have features included a weather forecast for the given destination and an interesting trip page with information about the destination, such as possible activities at the location.

The technical must haves are especially crucial, because without them the rest of the requirements could not be implemented.

### 4.1.2. SHOULD HAVE

Continuous deployment should be implemented. This requires setting up a production environment, because the current implementation is not secure.

The experience of booking a trip is made more enjoyable by the existence of interesting information about the destination. Thus the Should Have requirements included pulling in user reviews to display on the trip information page, providing a map displaying the location of the accommodation within the city, outlining the public transport at the destination, supplying all the flight information regarding the trip and offering pictures of the destination and accommodation.

A wider range of options also improve user experience, and thus regional destination search and the option of going by car should also be implemented.

### 4.1.3. COULD HAVE

It could be possible to deliver the number of seats available on a certain flight, and additional selection could include the departure airport, showing a calendar with prices and searching based on events like hiking or festivals.

## 4.2. TIME PLAN

This section provides an outline for the time devoted to the project, as it was established in the end of week 3 (see Figure 4.1).

The first two weeks of the project were mostly spent on conducting research about different parts and requirements of the project. Since the state of the product that was handed over to us was not good (not working) a lot of time needed to be put in fixing and rewriting a huge part of the code. This step was crucial because without rewriting the code and getting it to a state where it works, no other requirements and features could be added to the product.

After these steps the results of the research were presented to the company. The requirement to implement an integrated booking system into the current product was not the main focus of the company at this moment and instead they instructed us to add more features to their search engine and refactor the front end.

- **Week 1:** Visit company and start research (13-19 Nov)
- **Week 2:** Conduct research and fix the code (20-26 Nov)
- **Week 3:** Finalizing the research report and fixing code (27 Nov -3 Dec)
- **Week 4:** Design & Start with implementation (4 - 10 Dec)
- **Week 5:** Implementation (11-17 Dec)
- **Week 6:** Finalize implementation of technical Must Haves & submit code to SIG (18-24 Dec)
- **Christmas Holiday:** (25 Dec - 7 Jan)
- **Week 7:** Implementation & Implement SIG feedback (8-14 Jan)
- **Week 8:** Add extra features (should haves, could haves) (15-21 Jan)
- **Week 9:** submit code to SIG (22-28 Jan)
- **Week 10:** Finalize product, finish report and make presentation (29 Jan -2 Feb)

Figure 4.1: Overview of the time planning

The idea was to achieve a testable and working product in the end of week 6, before the holidays, and to implement the features after the holidays, while evaluating and adapting based upon the received feedback from SIG.

# 5

# DESIGN

In this chapter the conceptual design of the overall application is discussed, followed by a in-depth look at the designs of both front and back end. After establishing this design, the technologies used to achieve this are discussed.

## 5.1. CONCEPTUAL DESIGN

The main goal of Wheretrip is to provide a scalable web solution that allows its customers to find the best trips on their platform. To fulfill this goal the application needs to be scalable. This problem of scalability is generally defined by how many costumers the application can handle.

Conceptually all applications consist of a *n-tier* architecture[12]. This *n* defines how many separated programs work together to provide the necessary functionality required by the application as a whole. A single tiered application is the traditional program that installs on the users device and has no external communication. For Wheretrip this is not a suitable solution since it would require them to release a new version of the application with the latest trips basically every day to have up-to-date information available.

The scalability increases when an additional tier is added in the form of a server that provides the application with the business logic. This server can then serve the trip data to the first tier which is only concerned with displaying this data to the costumer. However this two-tiered architecture still does not solve the problem of having the latest data available automatically. Therefor a solution using only two-tiers is still not suitable for Wheretrip.

A separate data source tier is added to the previous solution to create a truly scalable solution. In this case the business logic in the second tier fetches the required data from external data sources. This solution already meets Wheretrip's data requirements, but it makes the application severely dependent on the performance of these external data providers. To partially remedy this dependency a local database is added to store all the data that is more permanent of nature (e.g. destinations and accommodations).

This three, or one could say four, tier architecture is chosen as the preferred solution for Wheretrip. The architecture can be seen in 5.1. The first tier is known as the front end. This tier sends the *HTML* documents with styling and *Javascript* logic to correctly display the data received from the second tier. The back end manages all business logic and is the connecting element between the data sources and the front end. The external data sources are drawn as to be in separate "clouds" to indicate that there is no control over their functioning.

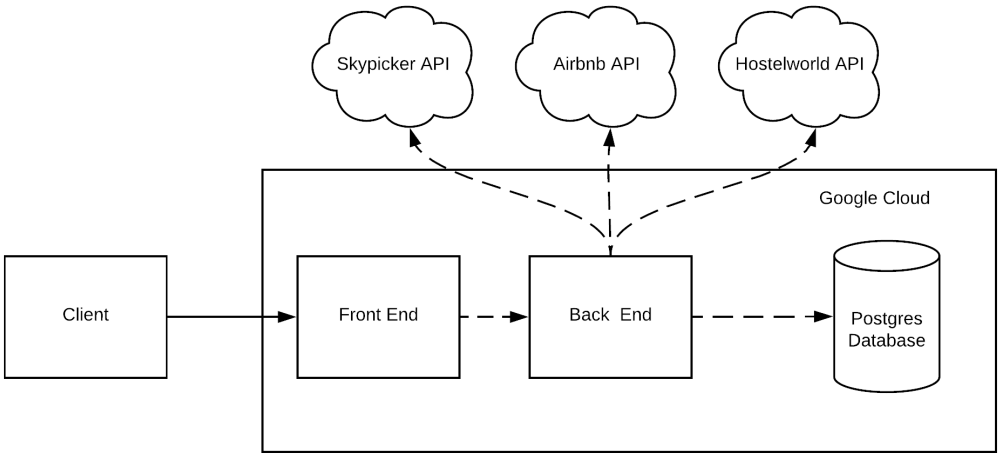


Figure 5.1: The architecture of the system.

### 5.1.1. FRONT END

The role of the front end is to be the view and control layers (referring to the Model View Controller programming pattern [13]) for the back end. This means that it requests infromation from the back end and changes its displayed output based on the information obtained. Not every change is worthy of a call to the back end however, so to a certain degree it needs to maintain a separate state model as well.

Thus it needs to be its own application, depending on the server for most of its data, but deployed separately. Because of the changing nature of user interfaces, its state model needs to always be synchronized (preferably data flow is unidirectional, from the parents to the children), and the division of concerns becomes of extreme importance.

It should be designed in multiple layers (see Figure 5.2). The top layer should be what the user sees, and that layer should only be responsible for rendering logic. The next layer is the logic responsible for management of state, and finally at the bottom layer a clear api for the back end.

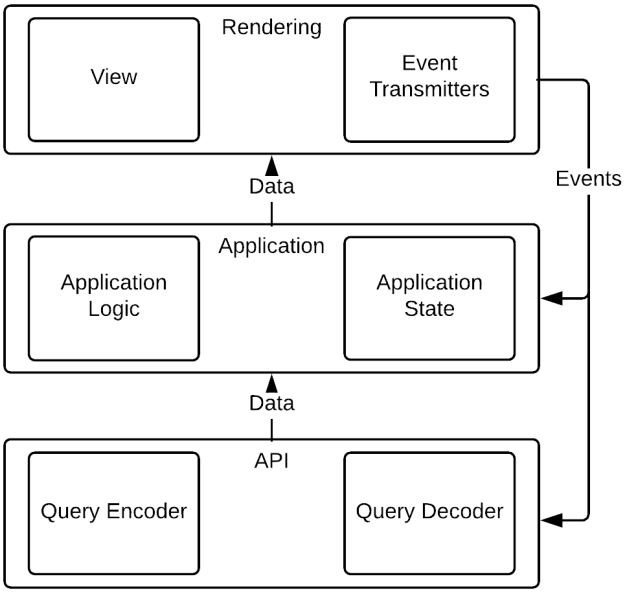


Figure 5.2: Overview of the front end layout

Preferably, as the programmers at Wheretrip are relatively inexperienced, it is imperative that the model be easy to understand and to work with. The code should be descriptive and logging should be understandable. Furthermore, it needs to be hugely modular, as this facilitates experimentation and learning.

## 5.2. BACK END

The main goal of the back end is to compose data required for the front end and make this available to be consumed by said front end. The design of the back end is based on RESTful web services [14], such a web service allows any HTTP Client to send CRUD (Create, Read, Update, Delete) operations to specific URIs (Uniform Resource Identificator) on the web service. For example if a client sends the following request:

```
GET /destinations/1/accommodations
```

Upon receiving such a request the back end will return all accommodations for that destination. Analogously it is possible to retrieve other data by querying the back end on a different URI.

The back end is comprised of *Controllers*, *Repositories* and *Entities*. The controllers handle the routing of the request and call, if necessary, a specific repository to fulfill this request. The repositories are responsible for combining and retrieving data from different sources. Finally the application will return one of the constructed value classes, here called entities. The figure below shows these components working together to handle a *GET* request to the */accommodations* endpoint.

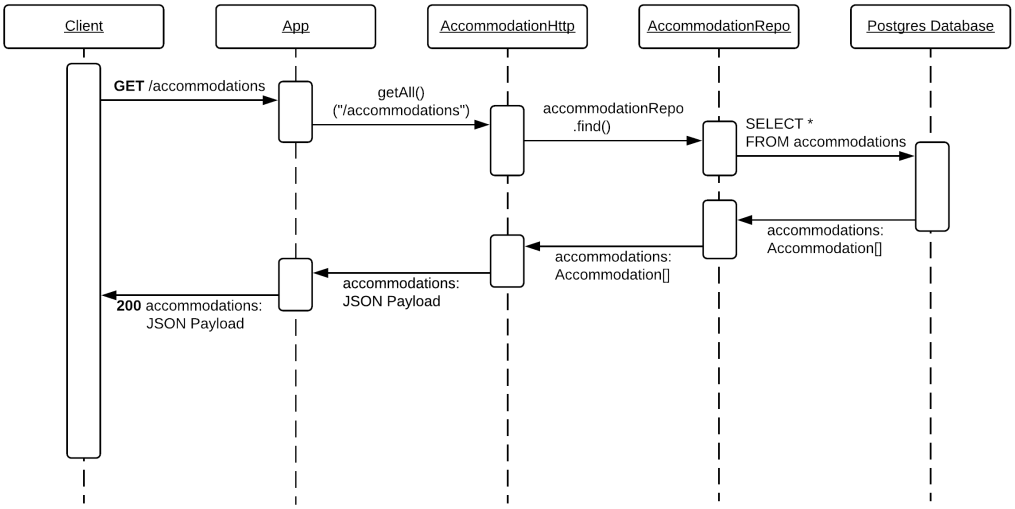


Figure 5.3: Overview of the different URIs and its composition

Whenever a HTTP request is received by the back end it will evaluate if there is a route registered with the request URI in any of the controllers, in case the route is not found a **404 Not Found** is returned. When a route is found the attached function is executed, this in turn calls the function from the accompanying repository to fulfill this request. Finally the repository executes a query on its resource, in this case a database, and creates an entity to return to the client. This entity is returned back through the chain of functions until it is returned as an HTTP Response 200 with the accommodations encoded as the payload in JSON format.

Throughout the back end design, the SOLID design principles were used where applicable. This acronym represents five design principles commonly referred to when mentioning maintainable, flexible and above all understandable code.

- **S**ingle responsibility principle: Every class or module should only be responsible for fulfilling the functionality of a single part of the system.
- **O**pen/closed principle: An entity, such as a class is defined as open when it is still reasonable to change its sourcecode. However, if an entity has many other entities which depend on it, changing its code will likely break implementations in those other entities. Therefor when a entity is defined as closed, it should only be extended instead of modifying the base entity.

- **L**iskov's substitution principle: Whenever a class has a dependency on another class A, it should be possible to substitute class A with class B which is a subclass of A without breaking the contract set in the parent.

- **I**nterface segregation principle: No entity should have to depend on methods which the entity does not use. This principle basically explains that interfaces should be kept as small as possible.

- **D**ependency inversion principle: A dependency should be on abstractions not implementation. Furthermore it states that high level modules should not depend on low level modules, instead it should be dependent upon abstractions which abstract those low level modules.

Ultimately when designing a software system it is important that the different modules that make up the system are loosely-coupled, using the aforementioned principles will help in achieving this goal. In addition to these principles, dependency injection[15] is used to further ensure loose coupling of the modules. In the following section the components of the back end will be discussed and how these components follow the design principles.

### 5.2.1. CONTROLLER

As mentioned before the main task of a controller is routing the request to the correct repository, beside that it also performs serialization to JSON and input validation. These controllers closely follow the design principles since they have the single responsibility of routing, they only depend on those repositories required to fulfill their function and they do not contain any low level logic such as database manipulation.

### 5.2.2. REPOSITORY

The repositories fulfill the function of retrieving data from a datasource, this can be from a database or an other web service, such as the flight API. This allows the repository to be responsible of returning one single type of data. Also it enforces the fourth principle by only exposing the minimum amount of functions to a controller. In addition to the previously mentioned functionality, a repository can also retrieve data from one or more other repositories. This allows for more complex operations to be implemented whilst adhering to the single responsibility principle.

### 5.2.3. ENTITY

Apart from being a simple value class, the entities also provide the interface between the data exchanged with the database as these entities provide an Object Relational Mapping with the relation database. As the database is constructed from the definitions of these entities, it is guaranteed that data returned from the database is according to the specifications within the application.

## 5.3. CHOICE OF TECHNOLOGIES

As the conceptual design of the application is now known, the technologies needed to implement the design are discussed. An overview of the considered technologies can be found in appendix C. Given that the application consists of multiple layers that interface through common protocols the technologies between the front end and back end can be widely different. However using the same technologies has its advantages; it allows for code reuse between the two systems, eliminate the need for learning multiple programming languages and it makes it easier to share knowledge within the team. Since JavaScript[16] is the defacto language for front end applications and the emergence of server-side JavaScript frameworks, such as NodeJS[17], this language was also chosen for the back end code.

Instead of using plain JavaScript, TypeScript[18] was chosen. TypeScript is a superset of JavaScript, so it compiles to plain, clean JavaScript code. It adds types to JavaScript, which enforces static type checking. Static type checking is a good development practice as it shows programming mistakes as you are typing. Furthermore it has support for the latest JavaScript features, which allows the application to be written in the state-of-the-art and be future-proof.

Since the same programming language is used, also the same testing framework can be used. For unit-testing the Mocha[19] testing framework was chosen. Mocha has support for the browser as well as NodeJS, making it an ideal choice for Wheretrip. However it does not have built-in assertions so Chai[20] was chosen to accompany Mocha. Integration testing is different between front end and back end as the interface for the

front end is a browser, whereas the back end provides an API. For the front end WebdriverIO[21] is chosen and the back end will use Supertest[22].

As mentioned in the conceptual design, the application requires a database to reduce the dependency on external data sources. For this purpose PostgreSQL[23] was chosen. PostgreSQL is highly performant and it supports expensive functions to be pre-calculated in the database it self such as fuzzy search indexing (used for an auto-complete functionality).

In order to work on the application at the same time, a version control system is required. Git[24] is the goto solution for any kind of collaborative assignment and has replaced older systems such as SVN[25]. The project is split over three git repositories for the back end, front end and shared code between the two. It is hosted on gitlab.com[26], which besides a git repository also provides features for code review and continuous-integration. All the repositories are setup to build the code and run the tests as soon as code is pushed to the repository. This way only code which successfully passes these steps can make its way onto the main branch.

Lastly Google Cloud was chosen as hosting provider for the application. It has native support for PostgreSQL databases and it allows applications to be run in containers on Kubernetes[27]. Kubernetes is a containerized application manager, that means it monitors and automatically scales applications based on their load. Wheretrip has acquired $2500,- in credit for the platform, which makes it best option between the different hosting providers.

## 5.4. FRONT END

The first decision, as previously discussed, was to discontinue server side rendering and instead rely on a dedicated server running the front end. That design decision was deemed necessary in order to ensure extensibility and quick rendering. It was determined to use React, a JavaScript library [28], as the front end framework, due to its component-based model, its reactive view model, its popularity (meaning there is a large community and extensive documentation around it), its integration with the NodeJS back end and the way it fit perfectly into our design model.

Each component is responsible for its own rendering based on its state and the properties that are passed to it. Thus, when a change occurs on the page, only the components that are affected by the change re-render, the rest remain as they were before. The *component state* is information that are only pertinent to the component itself, where properties is a union between properties passed down from parent components and the *application state*.

For application state management, Redux was chosen. It is an extension of Flux [29] is a library built by Facebook, designed to ensure a unidirectional data flow throughout software applications. It stores the whole state of your app as an object tree inside a single *store*. This state can only be changed by emitting an *action*, a JavaScript object describing what happened. To specify their effect, one writes *reducers*, pure functions that specify how the actions transform the state tree.

Whenever one component needs to communicate with another component (unless that component is an immediate child), it is accomplished by dispatching an action to the Redux store. This action is received by reducers, changing the application state, which flows down to the affected components, causing them to rerender. The advantage of such a unidirectional data flow is that a change can be trusted to have absolutely no side effects. This simplifies the management of separate components all responsible for their own tasks. That in turn leads to great performance increases, as it is expensive to reload a whole web page each time a change occurs.

Before the project started, the front end was already built in React, but it was not responsible for its own routing. Instead, each page inside the application was compiled to a separate application, delegating to the back end the responsibility of performing server-side rendering. This approach works with smaller scale applications, but when the goal is extensibility and limberness, it has worse performance. The original website had already been built in React, but there was very little to reuse from it, because it had been built for server-side rendering instead of internally handling routing.

In the jargon of React-Redux (the adapter responsible for communication between React and Redux), elements are sub-categorized into components and containers. According to our layer model (see Section 5.1.1), a *component* is in the Rendering Layer, meaning it only is responsible for rendering according to app and component state, and for dispatching actions. A *container* resides in the Application Layer, wrapping a component and mapping Redux state and functions dispatching actions to the properties available on the component. Thus, containers know about Redux, components do not.

A Redux store is the union of multiple reducers, each responsible for a piece of the application state. The creation of reducers thus allows subgrouping of state within the application. Redux allows for the creation of reducers as children of the total application state. Each reducer is thus responsible for handling its own substate of the component state. Thus actions were created for api calls, one for each endpoint, with the actual api calls abstracted away into dedicated repositories, and for navigation and seleciton on the front end itself.

At the same time as this project was underway a master student in industrial design was beginning his master thesis work involving the design of the website and its user interface. As it was already known that the design of the user interface would come later, it became a priority to ensure that the design would be as modular and malleable as possible. This would enable the company to add the designs on top of our work without much effort.

In order to simplify the work of the engineers that will later have to implement the design coming from that project, we chose to implement styled components into the projects. Styled Components is an attempt by one of the leading innovators in the React community to integrate styling more easily into React. It builds upon the premise that components are a better abstraction than CSS class names, and thus removes the binding between them. Using Styled Components, one simply creates a component with the desired styling, no more class-names required.

Because of our separation of concerns between function and user experience, those decisions can easily be changed later as desired by the designers. For this reason, the decision was made quickly, seeing the rise in popularity of the library. For the most part, it has made in-line styling much easier.

## 5.5. BACK END

As previously discussed the Node.js JavaScript runtime is used. Node.js, also abbreviated to Node, is designed specifically for building scalable network applications. Instead of a traditional concurrency model, Node is single threaded so that programmers do not need to worry about dead-locking their application. However having only a single thread is detrimental for being scalable. Node solves this by managing almost all I/O with asynchronous functions which never blocks the code from executing, instead executing a so-called callback function as soon as the I/O operation is finished.

The Node.js ecosystem is built around the node package manager[30] which is commonly refered to as npm. This open-source ecosystem has over half a million packages available which range in functionality from checking if an array is sorted to frameworks such as Express. By using these packages it is possible to implement a lot of functionality in relatively little time, since you can depend on tried and tested software and build the application on top of it. In the following section different packages are discussed which helped to implement the design as discussed in the previous section.

Express is the web framework used in the application. It abstracts away Node's HTTP I/O, basically hiding the difficult parts of the native implementation and provides the user with a easy to use API. In the back end Express is mostly responsible for routing the different requests to the required functionality.

In order to implement the different routes of the application in a readable and maintainable way, the Routing-Controllers[31] framework is added on top of Express. With this framework all functionality for one URI can be put together in a single responsible class that then is registered to handle all requests to that specific URI. A visual representation of how routing-controllers is used within the application can be seen in 6.10 in the next chapter.

For the access to the database the TypeORM[32] package is used. TypeORM is an Object-Relational mapping framework, which means it "maps" the result from a SQL query to the PostgresQL database to a TypeScript class. TypeORM enforces compatibility with the database by defining entities in TypeScript classes, these entities then create the schema for the database. Also it provides with functionality to manage this database through repositories, these create interfaces for manipulating SQL tables within TypeScript.

To tie these frameworks together the TypeDI[33] Dependency Injection package is used. Both Routing-Controllers as well as TypeORM has support for the inversion of control (IOC) container provided by this package. This allows the same container to be used for all the different injectable classes and thus providing the glue for the application to tie the different frameworks together. By only coupling classes through dependency injection keeps these classes loosely-coupled fulfilling one of the major design goals.

# 6

# IMPLEMENTATION

This chapter will discuss the implementation of the system into more detail. Using snippets from the codebase the thought process behind the code will be explained. In general the code was written with testability in mind, furthermore effort was put to make it human readable from just the code and comments are seen as a code smell indicating that the code itself is not descriptive enough in the naming of the functions. Whenever functionality was needed in multiple places that functionality was abstracted into a separate module to be imported wherever it was needed.

## 6.1. FRONT END

### 6.1.1. ACTIONS

There was a decision to be made with regards to the actions, how were they to be typed? In regular Redux, built for JavaScript, they are pure objects, but using TypeScript we wanted additional type safety. It was decided to write the actions as classes, and to introduce a middleware that compiles those classes into objects.

This led to the following general template for creating a basic action,

```
export const MY_ACTION="MY_ACTION";
export type ACTIONS = MY_ACTION // | ANOTHER_ACTION
class MyAction {
  private readonly type = MY_ACTION;
  constructor(public payload: ActionParams);
}
```

Figure 6.1: Example of a class-based action

Using Redux, An action can be dispatched in the following manner,

```
store.dispatch(myAction(myParams));
```

Figure 6.2: Example of dispatching an action

assuming that myAction is an action creator, store is the Redux store wrapping the application and myParams have already been declared. Synchronous action dispatch is very straightforward,

```
export const myAction(myParams: ActionParams) {
  return new MyAction(myParams);
}
```

Figure 6.3: Example of a synchronous action creator

Often, asynchronous action dispatch is preferred. This can be accomplished by installing a thunk middleware, allowing Redux to accept functions as actions. This enables the construction of asynchronous action creators, like in the following example,

```
export const fetchItems() {
  return (dispatch) => {
    dispatch(requestItems());
    fetch("http://www.example.com/")
      .then(
        (response) => response.json(),
        (error) => { throw error; },
      ).then(
        (json) => dispatch(receiveItems(json)),
        (error) => dispatch(invalidateRequest(error.message)),
      );
  }
}
```

Figure 6.4: Example of an asynchronous action creator

where requestItems(), invalidateRequest(message) and receiveItems(items) are synchronous action creators.

The fetches themselves were then abstracted into dedicated repositories, giving us final action creators like the following, taken from the final code,

```
export function fetchAccommodations(tripId: number) {
  return (dispatch) => {
    dispatch(requestAllAccommodations());
    return accommodationRepo.get(tripId)
      .then(
        (json) => dispatch(receiveAllAccommodations(json)),
        (error) => dispatch(invalidateRequest(error.message)),
      );
  };
}
```

Figure 6.5: Example of actual asynchronous action creator

### 6.1.2. REPOSITORIES

A separate repository was created for each endpoint, this is the one for the accommodation endpoint.

```
import { Service } from "typedi";
import { IQuerySchema, URLEncoder } from "wheretrip-shared";
@Service()
export class AccommodationRepo {
  private readonly encoder: URLEncoder = new URLEncoder(SERVER + "accommodation");
  public get(tripId: number): Promise<object> {
    const url = this.encoder.add("tripId", tripId).getURL();
    return fetch(url)
      .then(
        (response) => response.json(),
        (error) => {throw error; },
      );
  }
}
```

Figure 6.6: Example of api repository

This kind of repositories were first designed for the back end but determined useful also as the binding link between the client and the server. Thus, the discussion concerning them can be found in the design of the server.

### 6.1.3. REDUCERS

A reducer is a pure function that takes a previous state and an action, returning a new state of the two. Each reducer contains its own state, such that its state can be accessed as a property on the app state.

```
export default function(state = initialState, action) {
  switch (action.type) {
    case REQUEST_ALL_ACCOMMODATIONS:
      return { ...state,
        requests: state.requests + 1,
        accommodations: undefined,
        message: undefined,
        receivedAt: undefined };
    case RECEIVE_ALL_ACCOMMODATIONS:
      return { ...state,
        receivedAt: action.payload.receivedAt,
        accommodations: action.payload.accommodations };
    case INVALIDATE_REQUEST:
      return { ...state,
        message : action.payload};
    default:
      return state;
  }
}
```

Figure 6.7: Example of an actual reducer

### 6.1.4. TESTING

The tests for the front end were written using enzyme, a framework designed by airbnb for testing React applications. It allows shallow rendering (mounting the component is also possible) for react components. Chai was used for assertions. Below is a sample of a test, checking that the static page delivers the right page depending on the parameters passed.

```
describe ("StaticPage", () => {

  it("Should contain a back button", () => {
    const wrapper = shallow(<StaticPage contents="About"></StaticPage>);
    const children = wrapper.getElement().props.children;
    expect(children).to.deep.include(<BackButton/>);
  });

  it("Should have AboutComponent if contents=About", () => {
    const wrapper = shallow(<StaticPage contents="About"></StaticPage>);
    const main = wrapper.getElement();
    const StaticPart = main.props.children[1];
    expect(StaticPart.props.children).to.deep.equal(<AboutComponent/>);
  });

  it("Should have FAQComponent if contents=FAQ", () => {
    const wrapper = shallow(<StaticPage contents="FAQ"></StaticPage>);
    const main = wrapper.getElement();
    const StaticPart = main.props.children[1];
    expect(StaticPart.props.children).to.deep.include(<FAQComponent/>);
  });

  it("Should have TermsComponent if contents=Terms", () => {
    const wrapper = shallow(<StaticPage contents="Terms"></StaticPage>);
    const main = wrapper.getElement();
    const StaticPart = main.props.children[1];
    expect(StaticPart.props.children).to.deep.include(<TermsComponent/>);
  });

  it("Should have ContactComponent if contents=Contact", () => {
    const wrapper = shallow(<StaticPage contents="Contact"></StaticPage>);
    const main = wrapper.getElement();
    const StaticPart = main.props.children[1];
    expect(StaticPart.props.children).to.deep.include(<ContactComponent/>);
  });

});
```

Figure 6.8: Example of component testing with Enzyme

## 6.2. BACK END

Now that the design and the technologies for the back end are known, the system still needs to be implemented. Specifically the destinations resource will be inspected and how the technologies are used in the system.

```
public static async start() {
  routingUseContainer(Container);
  ormUseContainer(Container);

  await createConnection({
    ...Config.database,
    entities: [ __dirname + "/**/*.entity{.ts,.js}"],
  });

  const app = await createExpressServer({
    ...Config.server,
    controllers: [__dirname + "/**/*Http{.ts,.js}"],
  });

  await app.listen(Config.server.port);
}
```

Figure 6.9: App initialization

At the entry point of the back end code both Routing-Controllers as well as TypeORM are setup to use the TypeDI container. Then the connection with the database is created based on the entities in the system. Each entity has the `.entity.ts` extension so that they can be loaded without specifically importing the entities. Afterwards the Express server is created using all the controllers which are imported based on the `Http.ts` extension.

### 6.2.1. CONTROLLERS

Each base route in the application has a controller implementing the available functionality. In figure 6.10 a part of the Destination controller is shown.

```
@JsonController("/destinations")
export class DestinationHttp {
  constructor(private readonly destinationRepo: DestinationRepo) {}

  @Get("/:id([0-9]+)")
  public getOne(@Param("id") id: number): Promise<Destination | undefined> {
    return this.destinationRepo.findOne(id);
  }

  @Get("/search")
  public search(@QueryParam("name") name: string): Promise<Destination[]> {
    return this.destinationRepo.search(name);
  }
}
```

Figure 6.10: DestinationHttp with two routes

The `JsonController` decorator registers this class under the `/destinations` route and puts the class in the container. Each class in the container will automatically inject other classes that appear in the constructor of that class. In this case the `DestinationRepo` is injected. Each method implements one of the routes in the `/destinations` resource. The `getOne` function is registered as callback to all requests on `/destinations/id` where id is a number. With the `@Param` decorator the parameter "id" is injected as the argument of the method. The destination repository will then be called upon to find one Destination entity with a specific id.

### 6.2.2. REPOSITORIES

For each repository there is a accompanying interface that describes the methods available on the repository. In figure 6.11 the interface is shown for the destination repository.

```
export interface IDestinationRepo extends IRepo<Destination> {
  search(name: string): Promise<Destination[]>;
}
```

Figure 6.11: the interface for DestinationRepo

This interface extends on the generic repository interface. the generic interface shows which functions are implemented by default from extending the TypeORM repository class.

```
export interface IRepo<T> {
  find(): Promise<T[]>;
  findOne(id: number): Promise<T | undefined>;
  save(object: T): Promise<T>;
  update(id: number, object: T): Promise<UpdateResult>;
  delete(id: number): Promise<DeleteResult>;
}
```

Figure 6.12: The interface for the generic repository functions

By extending the TypeORM repository class the specific repositories already have a lot of functionality available. The implemented classes can then focus on extending behavior instead of having the same code in each class. Figure 6.13 shows the implementation of the `DestinationRepo`, here the class only has to implement specific functions such as the search functionality.

```
@EntityRepository(Destination)
export class DestinationRepo extends Repository<Destination> implements IDestinationRepo {

  public search(name: string): Promise<Destination[]> {
    return this.createQueryBuilder()
    .addSelect("name <-> :name", "similarity")
    .orderBy("similarity", "ASC")
    .setParameter("name", name)
    .limit(10)
    .getMany();
  }
}
```

Figure 6.13: DestinationRepo with the search function shown

Next to implementing the default repository functions, extending the `Repository` class also provides access to the `QueryBuilder`. This builder allows for SQL queries to be implemented in a specific domain specific language (DSL) as shown above in the search function. This exposes advanced querying possibilities on the database while keeping the benefits of static typing on the returned query. The `getMany` function automatically parses the result to Destination entities as TypeORM knows how the result should map to the specific entity.

### 6.2.3. ENTITIES

Each entity is decorated with the `Entity` decorator, this tells TypeORM that a database table should exist for the class. In figure 6.14 the implementation for the `Country` entity is shown.

```
@Entity()
export class Country implements ICountry {

    @PrimaryColumn()
    public id: string;

    @Column()
    public name: string;

    @Column()
    public slug: string;

    @Column()
    public code: string;

    @OneToMany(
        (type) => Destination,
        (destination: Destination) => destination.country,
    )
    public destinations: Destination[];
}
```

Figure 6.14: Country entity

As seen in the figure the id field of the class is decorated with the `PrimaryCOlumn` decorator. This indicates that the id column should be the primary key for the Country table. The `Column` decorator marks a field to have a column in the table. Also complex relations can be modeled in the class with the `OneToMany` decorator. This tells TypeORM that for each country many destinations can exist. This does not create a list of destinations in the country table instead it enforces a foreign key on the CountryId column in the destination table.

### 6.2.4. TESTING

Since the ideal way to test a single piece of functionality is by controlling the input and respective output of all dependencies of the unit under test. In this section `DestinationHttp` is the unit under test. As seen in the previous sections, the controller has a dependency on `DestinationRepo`, and if the behavior of this repository can't be changed a testing database would be required in order to test the functionality of the controller. Luckily the repository can be replaced with a fake implementation of all functions returning exactly what is expected to test the functionality of the unit under test. Below shows such a fake implementation called the `DestinationRepoStub` this "stub" implementation replaces the original functionality of requesting data from the database with a simple function return the required data.

```
export class DestinationRepoStub implements IDestinationRepo {

    constructor(private readonly destinations: Destination[]) {}

    public find(): Promise<Destination[]> {
        return Promise.resolve(this.destinations);
    }

    public findOne(id: number): Promise<Destination | undefined> {
        return Promise.resolve(this.destinations[id]);
    }

    public search(name: string): Promise<Destination[]> {
        return Promise.resolve(this.destinations.filter((value) => value.name.includes(name)));
    }
}
```

Figure 6.15: DestinationRepo Stub implementation

For example the search function now searches through a given list of destinations to find the one that includes a part of the search query in it's name.

Each file be it a controller, entity, repository or utility function has an accompanying `.spec.ts` file containing the tests for the specific functionality that the module implements.

```
before(() => {
  Container.set(DestinationRepo,
    new DestinationRepoStub(testDestinationArray, testDestinationArray)
  );
  destinationHttp = new DestinationHttp(
    Container.get(DestinationRepo)
  );
});

describe("DestinationHttp Spec:", () => {

    it("GetOne should return the queried destination", () => {
        return destinationHttp.getOne(4).should.eventually.deep.equal(testDestinationArray[4]);
    });

    it("GetAll should return all destinations", () => {
        return destinationHttp.getAll().should.eventually.deep.equal(testDestinationArray);
    });

    it("Search should return Aarhus when searching for Aar", () => {
      return destinationHttp.search("Aar").should
        .eventually.deep.equal([testDestinationArray.find((destination) => {
          return destination.name === "Aarhus";
      })]);
    });
});
```

Figure 6.16: Testing of `DestinationHttp`

In this figure the tests for the `DestinationHttp` are shown. First of all the fake implementation is inserted to replace the DestinationRepo entry in the IOC container. Then the controller is created and the fake repository is injected through the constructor. Now by calling a specific function of the controller the return

value is known through the fake implementation. Thus it can be tested if it actually returns the value that should be returned by the controller. In the same fashion each unit is tested.

# 7

# Software Improvement Group

The code that we were working on for the Wheretrip website was send to the software improvement group (SIG) at the end of week 8. SIG examines the code and determines whether the code is well-structured and whether it is maintainable. SIG is supposed to evaluate the code twice. The first time is the initial code of the group is send to be evaluated and the second time the improved code with the generated feedback from the first evaluation is send. This chapter will only discuss the first evaluation of the code since there was no time to send the code to a second evaluation. The state of the code that was send to SIG is very similar to the state of the final deliverable. Code for both the front end and back end was submitted to SIG and the group received feedback at the end of week 9 mostly about the state of the back end. The group received 3.5 stars for their submitted code. This means that the state of the code is above average. Since we did not manage to get 5 stars, there is still room for improvement on both the front and back end. The main issues were the arguments in the constructors and the length of functions in several classes.

## 7.1. Feedback

### 7.1.1. Unit Size

The unit size issue referred to the length of some of the classes mostly in the back end. One example of this was the DestinationRepo class which included a function getMinAccommodationPricePerDestination that retrieved the minimal price for a desetination from the database. Here the main problem was that the query that was implemented to get the data from the database was very long and afterwards it was also mapped to get the desired output format. The exact funcion can be seen below.

```
public getMinAccommodationPricePerDestination(conversions: Currency[]): Promise<Destination[]> {
  const query = this.query(`
  SELECT DISTINCT ON ("destination"."id") "destination"."id" AS "destination_id",
            "destination"."string_id" AS "destination_string_id",
            "destination"."name" AS "destination_name",
            "destination"."slug" AS "destination_slug",
            "destination"."code" AS "destination_code",
            "destination"."latitude" AS "destination_latitude",
            "destination"."longitude" AS "destination_longitude",
            "destination"."type" AS "destination_type",
            "di"."id" AS "di_id",
            "di"."image" AS "di_image",
            "di"."description" AS "di_description",
            "di"."timezone" AS "di_timezone",
            "a"."id" AS "a_id",
            "a"."name" AS "a_name",
            "a"."latitude" AS "a_latitude",
            "a"."longitude" AS "a_longitude",
            "a"."type" AS "a_type",
```

```
        "destination"."countryId" AS "destination_countryId",
            (LEAST(
              CAST(ai."minPrice"->>'EUR' AS FLOAT),
              CAST(ai."minPrice"->>'USD' AS FLOAT)*$1,
              CAST(ai."minPrice"->>'AUD' AS FLOAT)*$2,
              CAST(ai."minPrice"->>'GBP' AS FLOAT)*$3
            )) AS "minPrice"
      FROM "destination" "destination"
        INNER JOIN "accommodation" "a" ON Destination.id = a."destinationId"
        INNER JOIN "accommodation_information" "ai" ON "ai"."id" = "a"."id"
        INNER JOIN "destination_info" "di" ON di.id = Destination."informationId"
      WHERE ai."minPrice"->>'EUR' NOTNULL OR
            ai."minPrice"->>'USD' NOTNULL OR
            ai."minPrice"->>'AUD' NOTNULL OR
            ai."minPrice"->>'GBP' NOTNULL
      ORDER BY "destination".id ASC, "minPrice" ASC`,
          (conversions.find((value) => value.name === "USD") as Currency).rate,
          (conversions.find((value) => value.name === "AUD") as Currency).rate,
          (conversions.find((value) => value.name === "GBP") as Currency).rate,
       ]);

         return query.then((result: any[]) => {
            return Promise.resolve(result.map((row: any) => {
                const destination: Destination = plainToClass(Destination, {
                    id: row.destination_id,
                    string_id: row.destination_string_id,
                    name: row.destination_name,
                    slug: row.destination_slug,
                    code: row.destination_code,
                    latitude: row.destination_latitude,
                    longitude: row.destination_longitude,
                    type: row.destination_type,
                    information: plainToClass(DestinationInfo, {
                        id: row.di_id,
                        image: row.di_image,
                        description: row.di_description,
                        timezone: row.di_timezone,
                      }),
                     accommodations: [plainToClass(Accommodation, {
                            id: row.a_id,
                            name: row.a_name,
                            latitude: row.a_latitude,
                            longitude: row.a_longitude,
                            type: row.a_type,
                            images: [],
                      }),
                    ],
                    countryId: row.destination_countryId,
                    minPrice: row.minPrice,
                 });

                return destination;
             }));
         });
      }
```

According to SIG, this function is difficult to understand and test. The solution is to split the function into

smaller function which will also contain smaller components. With doing this, it will be easier to understand and test the different components of the functions and the function will again be maintainable. Specifically, the getAccommodationPerDestination function has two main responsibilities namely making a query to retreive the results and mapping the results of that query to a desired format. Since these responsibilities are very likely to change over time according to the current needs of the website, it is best to implement them in a way that they will be extendable and maintainable in the long term. This can be achieved by splitting these two responsibilities into two different functions.

### 7.1.2. UNIT INTERFACING

The unit interfacing issue referred to the number of parameters that were included in the constructors of several classes. The main problem here was that some of the constructors contain too many parameters and because of this an absence of abstraction occurs in these classes. The large number of parameters can also lead to confusion when calling a particular function and it also can add to the complexity of a function. The two constructors that were pointed out by SIG are the `QuerySchema` constructor which consists of 9 parameters and the `SkyPickerParams` constructor which consists of 13 parameters. These constructors contain many parameters because the data that is used in these classes is retrieved from the API's that are used. An example of the `SkypickerParams` constructor can be seen below.

```
constructor(flyFrom: string,
        dateFrom: Moment,
        dateTo: Moment,
        daysInDestinationFrom?: number,
        daysInDestinationTo?: number,
        to?: string,
        returnFrom?: Moment,
        returnTo?: Moment,
        typeFlight: flightType = "round",
        passengers = 1,
        price_from = 1,
        price_to = 1000,
        single = true,
    )
```

This constructor contains several different from and to parameters namely, dateFrom, dateTo, from, to, returnFrom, returnTo, price from and price to. A solution to solve this problem and reduce the number of parameters in the constructor is to create a seperate class for the date and a separate class for the price. This would make the datastructure simpler and the cosntructor easier to understand. The group chose not to create seperate classes for the price and date parameters because this would not lead to any additional functionalities and there was no significant loss of abstraction.

# 8

# EVALUATION

## 8.1. DELIVERABLES

Refer to section A for the requirements used by this evaluation. The summary of this section is that most of the goals related to technology were achieved, whereas, due to time constraints, most of the goals related to features were not.

### 8.1.1. MUST HAVES

The initial goal of the testing requirement was to have the front and back end entirely covered. The front end needed to include encompassing unit, integration and end-to-end tests. At the time of this writing most of the front end has unit and integration tests, and it seems probable that full coverage will be reached before the end of the project. The back end has full code coverage with regards to unit tests, but misses API validation tests. During development an API tester was used to check the endpoints, but no automated tests were written. The requirement to enable continuous integration for the project has been accomplished. The project repository was divided into three, client, server and shared. Each has a master and a development branch, with enabled continuous integration pipelines to check that only code that builds and passes the existing tests can be merged into them. Direct pushes have been disabled. The requirement to decouple the front end and back end was also accomplished. The front and the back end are able to run separately as set out in the requirement. The files that are used for coordination between front and back end (interfaces, query schemas and the URLEncoder)resides in a third repository named shared. The requirements to re-factor the front end and back end have also been accomplished. The front end has been rebuilt form scratch and is much easier to work with and to extend. Work still remains to make use of this potential, but the foundation is there. The back end has also been rewritten from scratch, and is now maintainable and extensible. The requirements that were set for extending the website with features were only partly accomplished. The requirements that were accomplished included adding a possibility to search for a wider time frame to get the cheapest trip and also adding a possibility to get different accommodation types. Integration with airbnb was added to the back end, but not implemented in the front end. Just as with travel providers, many flight providers require PCI compliance. The requirements to add different travel providers and travel types were not accomplished. Research was made into various travel providers but no additional functionality was added in order to contact them. Many travel providers require PCI compliance, and as that was beyond the time we had, our options were cut short in that regard. The requirements to include more detailed information about the chosen destination and the weather at that destination were not accomplished.

### 8.1.2. SHOULD HAVES

Most of the requirements were not accomplished. These requirements included: adding user reviews to the website, adding an option to choose the location of the accommodation at the chosen destination, enabling continuous deployment, including an option to search for a region instead of a specific place, providing more information about a selected flight and also providing pictures to the destination description. Effort was put into setting up hosting at Google Cloud, but due to time constraints (coupled with the delays caused by the issues mentioned in the communication section), this was never pursued. The location selection was pulled

into the back end instead of relying on external APIs. Implementing regional lookup would be much easier now, even though it was not implemented during the project.

### 8.1.3. COULD HAVES

These requirements were not accomplished. The requirements included: making the website PCI-compliant, including an option to see the seats availability, including an option to choose a departure airport, creating a calendar that shows the cheapest price for a trip, creating an option to select a destination based on hiking trails and festivals. One of the main requirements for PCI compliance is a secure hosting environment. As there was not enough time for that, this goal was deemed beyond reach.

# 9

# Discussion & Conclusion

## 9.1. Timeline

In this section a timeline of the project development is presented. For each week an explanation and evaluation of the tasks and progress is discussed.

### 9.1.1. Weeks 1-3

The first week of the project three members of the group got together to meet each other and to arrange an appointment with the company Wheretrip in order to get information about their assignment. At the start of week two a meeting with the company was arranged and an additional member joined the group. The company explained the assignment and the expected results and also briefly explained the state of the code-base at that moment.

The initial assignment that was given to the group was made up of two parts. Part one was to add the functionality for users to be able to book flight and accommodation on the company's website instead of being redirected to the actual websites that provided them. Part two was to extend the website with additional features. This task was mainly focused on providing a more personalized and tailored experience for the user when searching for a travel destination. Some of the discussed extensions where adding an option to search for different means of transportation, different types of accommodation and information and pictures of the chosen destination.

Upon discussing the assignment, the company gave the group access to their code-base. After seeing that the code did not compile, the rest of week two was mainly spent on trying to get the code to a compiling state and researching the possibility of making the website PCI compliant. By the end of week two a meeting with the mentor took place. During week three it became evident that it would take a lot of time to implement the integrated booking system and make the website PCI-compliant. These findings were discussed with the mentor and the company and eventually a decision was made to work on the personalization of the website. The research phase about the personalized features and API's started in week three and was finished by the end of week three. Parallel to this also the refactoring of the front and back-end started.

This was needed because a non-working product was provided as a starting point, with the assignment to extend it and add more personalization features to it. Since this was not possible with the code base in that state, extra work was required to get it to a state where it would be possible to fulfill the assignment. The research report was finished and handed in at the end of week three.

### 9.1.2. Weeks 4

The implementation was initialized by installing the testing framework. First, the idea was to use Jasmine to test the front end, in keeping with the back end. When it was discovered that a significant rewrite of the back end was required, however, which would render the existing tests useless, the matter was reevaluated and it was decided to use Mocha instead.

This led to coding on the front end having to start with a delay, and without testing frameworks ready to go, which was not at all ideal.

The brunt of this period was used to dissect the old front end, and to transcribe the few golden nuggets still contain therein into code applicable to the new front end. Most of the front end implementation became a

total rewrite however. The first step was installing React Router. This was a direct consequence of the decision to be able to do on page routing, instead of using server-side rendering.

### 9.1.3. WEEKS 5-6

In the middle of week five, the first endpoint of the new back end came online, and the work on the front end was divided into two main areas of focus. One was navigation and search, or in other words, the actions firing from the rendering layer to the application layer (see Section 5.1.1), the other to enable data fetching from the back end, including the actions causing those fetches to occur. One group member was assigned to each of these areas.

Towards the end of week six, good progress had made with the fetches, data requests had been facilitated, and information was pouring in from the back end. The search and navigation features as well as the fetches had been implemented before the holidays began. They hadn't been synchronized yet, meaning that the navigation and search were implemented on mock data, but the functionality was there.

The supervisor never came back with the information about code submission to SIG before the holidays, so that request for feedback could never be carried out. In total there was a delay of about two weeks on the front end at this point.

### 9.1.4. WEEKS 7-8

In the beginning of week seven, the navigation and search were synchronized with the fetches. The fetches were then abstracted into dedicated repositories. The shared schemas were finally made available, and the types throughout the front end were updated to match them. Additionally, the URLEncoder was put in the shared repository as well, facilitating querying based on the data supplied by the user. The navigation features were extended with the selection of tripdestinations, as well as with the ability to navigate back and forth between the various screens, retaining the state while doing so.

The success criteria were revised, as the ones specified at the end of week three would no longer be attainable. These were the new success critria,

- Meet all of the Must Have requirements.
- Pass the project on in a way, such that the programmers at Wheretrip can continue to implement it.

The tests for the front end were still missing, thus the SIG hand-in was made with tests still lacking for the front end pages. The front end was now in the state that it was supposed to have been in at the end of week 6.

### 9.1.5. WEEKS 9-10

Week nine was spent writing the report, and in week ten the goal is to implement the final touches, including making sure the whole front end is tested, implementing search completion and educating both the wheretrip programmers and the designer on how to work with React and Redux.

## 9.2. RECOMMENDATIONS

As the project did not really go according to plan there are missing features that still need to be implemented. Apart from those features, there are some observations on the process during the project and in the start-up as a whole.

### 9.2.1. FRONT END

The front end has during the project gone from being a frail collection of applicaitons each rendered separately by the back end into a fully fledged front end application. If used correctly it will remain extendable, useful, and able to implement any design desired. The following recommendations are meant for the programmers that will be working on the front end in the future.

The first recommendation would be to read Section 5.1.1 of this report and adhere to those design decisions. Of utmost importance is to retain the division of concern between the application and the rendering layer. This ensures that the design never has to be limited by the logic, and vice versa.

The second recommendation would be to design components/containers with reusability in mind. This means that if there are multiple places where the same component is being used, it should be pulled out into a general component/container library, from where it can be imported from anywhere in the system. This

makes the system much easier to maintain and to test, as there is no repeating of code and only one place where errors can occur.

### 9.2.2. BACK END

Although the back end has been rewritten into clean, loosely coupled classes, not all requirements were implemented due to time constraints. The back end was written with Skypicker and Hostelworld as the main provider and currently a Skypicker Datum object is returned as part of the trip object this should be refactored to a more general flight/travel object. This would enable extension to other flight providers and even to more travel providers in the long term. In the same way the accommodation entity is fully focused on the Hostelworld data type.

It would be good to abstract this to a more generic accommodation, so that it can be used for hotels as well as B&Bs from different providers.

When developing back end code try to follow the conventions set by the current code base, that means: Keep domain logic in repositories and let each repository fulfill only one functionality. This will keep the code extensible and maintainable. The thought process behind this was explained in section 5.2. And these concepts should be followed when extending the code base.

Finally a good improvement for the reliability of the back-end is implementing automated integration tests, these are now done manually, but this leaves the possibility for a bug to be introduced by a software update.

### 9.2.3. HOSTING

Since there were severe delays in the project, hosting was one of the could-haves that still needs to be implemented. Even though effort was put into setting up the services on Google cloud, the group did not succeed in hosting the application as of writing this document. The database is already set-up on Google cloud what is left is deploying the front end and back end in separate containers on the cloud service. To expose the application to the internet setting up a reverse-proxy on Google cloud needs to be researched as this allows SSL to be implemented on the Wheretrip domain.

### 9.2.4. PROCESS

Throughout the duration of the project the group missed a common technology vision within the company. As there is no dedicated technical founder in the company, the owners make these technological decisions based on opinions from students that in most cases also do not posses the level of knowledge required. The company relies on first and second year Computer Science students that work part-time, as interns, on this project. In addition to these interns most if not all of the development choices have been made by Bachelor End Project groups, both in the fourth quarter of the 2016-2017 Academic year, as well as this group. Unfortunately these projects only last for two months and then leave the company together with all the knowledge about the application. In order to improve knowledge within the company, we would advice on attracting an experienced developer which would be able to create the companies technological vision.

## 9.3. CONCLUSION

For the fulfillment of the project the group re-build a travel website application. This project was conducted for the company Wheretrip. The original state of the product was not maintainable nor extendable. To improve their product the code behind the website was entirely rebuild. This mainly included rebuilding the front end and the back end and extensively testing both front and back end. The data that is retrieved is also validated to prevent invalid formatting which could lead to different errors. Unfortunately, most of the personalization features were not included in the final product. Because the whole product was rebuild and also as a consequence of several problems the group encountered during their work on the product, not all requirements were achieved.

The final product that the company will receive is both extendable and maintainable as opposed to the initial state that the product was found in, namely not working at all. The product can now be very easily extended and improved. If future engineers follow the recommendations that were given in the previous section they can adequately improve the product instead of hacking things together.

In conclusion, the group is partly satisfied with the final product, despite that they did not manage to implement all desired features they did manage to create a stable, well designed foundation of the website application.

# A

# REQUIREMENTS

## A.1. PERSONALIZATION

### MUST HAVE

- **Date Flexibility:** A user must be able to search for a wider time frame instead of pre-specified dates to get the cheapest trip.
- **Travel providers:** A user must get results from different travel providers to get more choice.
- **Weather:** A user must be able to see the weather forecast for a certain destination.
- **Destination Info:** A user must be able to see information about the destination, such as possible activities.
- **Accommodation Types:** A user must get results from different types of accommodations, so that there is more choice of accommodations. Different types that should be available:
  - Hotels
  - Bed & Breakfast's
- **Travel Types:** A user must be able to choose between the following means of transport:
  - Plane
  - Train
  - Bus

### SHOULD HAVE

- **User Reviews:** A user should be able to see user reviews about a certain destination.
- **Accommodation Location:** A user should be able to see the accommodation location in respect to the arrival location (e.g. airport, train station).
- **Regions:** A user should be able to specify a region where the trip should go to, such that only results into that region, within budget are shown.
- **Public Transport:** A user should be able to see the available public transport providers for a certain destination.
- **Flight Information:** A user should be able to see the flight information for a certain trip
- **Get Pictures from API:** An administrator should be able to choose different pictures for trip destination results.

- **Additional Travel Types:** A user should additionally be able to choose between the following means of transport:
  - Car
  - Car-share

### COULD HAVE

- **Seats Available:** A user could be able to see the remaining seats available for a trip.
- **Choice of Airports:** A user could be able to specify the choice of departure airports.
- **Calendar with Prices:** A user could see an overview of trip prices for a certain month.
- **Hiking:** A user could search for trips for certain recommended hiking trails
- **Festivals:** A user could search for festivals and get available trips that go there at the required time.
- **Specify Accommodation location:** A user could specify the location of the accommodation in the trip destination (e.g. city center, suburbs, near airport)

### WON'T HAVE

- **Virtual Travel Agent:** A user will not be able to talk to a virtual travel agent to assist in finding the trips.

## A.2. TECHNICAL

### MUST HAVE

- **Front-End Tests:** The front-end must have tests.
  - It must have unit-tests
  - It must have integration-tests
  - It must have end-to-end tests
- **Back-End Tests** The back-end must have tests.
  - It must have unit-tests
  - It must have API validation tests
- **Continuous Integration:** Continuous integration must be set up for the application to validate that the application works correctly.
- **Decoupling:** The front-end and back-end code must be able to run separately, being loosely coupled through the API.
- **Refactoring Front-End:** The front-end must be refactored to be maintainable and extensible.
- **Refactoring Back-End:** The back-end must be refactored to be maintainable and extensible.
  - **Database Improvements:**
  - **Caching:**

### SHOULD HAVE

- **Continuous Deployment:** The application should have its latest working version automatically deployed to a live environment.

### COULD HAVE

- **PCI-DSS Compliance:** The application could be PCI-DSS Compliant.
- **Containers:** The application could be hosted using containers.

### WON'T HAVE

- **Integrated Booking System:** The application will not have an integrated booking system, since it is not feasible in the current state of the application.

# B

# APIs

## B.1. Accommodation Types

### Expedia, Inc.[34]

+ filter accommodation types: hotels, apartments, hostels/backpacker accommodations, bed breakfast , private vacation homes
+ filter by popular locations
+ filter by neighborhood
+ filter by distance from transportation
+ filter by price per night
+ filter by property class: 1, 2, 3, 4, 5 stars
+ filter by amenities: high speed Internet, air conditioning, swimming pool, free breakfast, free airport transportation
+ EAN (Expedia Affiliate Network) that provides different API's and developing solutions
+ customizable API
+ brand content as your own
+ EAN database available

### HotelsCombined[35]

+ filter accommodation types: hotels, apartments, hostels, motels, bed breakfast
+ filter by ratings
+ filter by price per night
+ filter by property class: 1, 2, 3, 4, 5 stars
+ flexible dates search
+ joining the HotelsCombined Affiliate Program is Free
+ customize the affiliate links via customization features: modify search boxes, modify links, modify branding templates
+ setting up the HotelsCombined affiliate links: copy and paste the generated HTML code into your web page
+ multiple integration options available: Text Links, Deep Links, Banners, Search Boxes

GoSeek[36]

+ filter accommodation types: hotels, motels, downtown, beach, extended stay
+ filter by property class: 1, 2, 3, 4, 5 stars
+ filter by savings type: coupons, mobile, AAA member, senior citizen, member rate, military rate, government rate, student rate
+ filter by amenities: indoor pool, outdoor pool, restaurant, fitness facilities, business center, Internet, parking, airport shuttle, childcare, kitchen, wheelchair accessible, pets allowed, spa services, casino, golf course
+ filter by freebies: free breakfast, free Internet, free parking, free airport shuttle, free child care
- no available api

Kayak[37]

+ filter accommodation types: apart-hotels, hotels, Inn, rental, hostels, bed breakfast
+ filter by price per night
+ filter by hotel name
+ filter by neighborhood
+ filter by locations from city center
+ filter by freebies: free breakfast, free Internet, free parking, free airport shuttle, free cancellation, all-inclusive
+ filter by review score
+ filter by property class: 1, 2, 3, 4, 5 stars
+ filter by amenities: pet friendly, pool, Wi-Fi, spa, rooftop, parking, air-conditioned, fitness, Internet, airport shuttle, adults only, non-smoking rooms, restaurant, casino, golf, disability-friendly, 24 hour front desk
+ filter by ambience: beach, romantic, family, luxury, spa/wellness, airport, trendy, budget, boutique, business, Eco-friendly
+ filter by booking providers: Booking.com, ebookers.com, Expedia.com, Hotelopia.com, Hotels.com, HRS,com, lastminute.com, TripAdvisor.com, roomsXXL.com, Prestigia, TravelRepublic, Sembo, Tablet, Preferred Hotel Group, The Leading Hotel

SkyScanner[38]

+ filter by accommodation type: hotel, hostel, town house, private home, guest house, apart-hotel, bed and breakfast
+ filter by hotel chain
+ filter by meal plan: half board, Breakfast included, Room only
+ filter by cancellation policy: free cancellation, refundable, non refundable
+ filter by property class: 1, 2, 3, 4, 5 stars
+ filter by district
+ filter by total price
+ filter by facilities
+ SkyScanner has an API that is free to use after registration of the company.

## B.2. DATE FLEXIBILITY

SKYSCANNER[38]

+ option to choose flexible dates
+ cheapest month calendar and chart
+ multi city option
+ filter by direct flights only
+ flexible destination (everywhere)
+ filter by cabin class
+ required api key request (free to use)

KAYAK[37]

+ flexible dates +/- one month
+ flexible week option
+ flexible month option
+ I am a student option
+ multi city option
+ explore option

DOHOP[39]

- return cheapest tickets grouped by month
- return tickets from a city for every day of the month
- return cheapest non-stop tickets for selected route
- return price calendar with prices for dates closest to the ones specified
- return prices for alternative routes
- required request of api key

FLY[40]

+ fare calendar available for chosen destination
+ today's cheapest flights option
- no api available

STA TRAVEL[41]

+ flexible dates +/- 3 days
+ filter by preferred airline companies
+ filter by departure/arrival time: any time, morning, afternoon, evening
+ multi city flights option
+ information about local transportation: rail travel, bus passes, camper-vans, car hire
- no flexible date month calendar

KIWI[42]

+ multi city flight option
+ filter by price range
+ filter by stops: any, nonstop (direct flights), up to 1 stop, up to 2 stops
+ flexible date options: cheapest flight monthly calendar, cheapest flight anytime

## B.3. TRAVEL TYPES

BOOKING[43]

+ multi-city option
  Flights
+ flexible dates +/- 3 days
+ I am a student option
+ filter by trip length
+ filter by stopover duration length
+ filter by departure/return airport
+ filter by type of booking providers: Aegean Airlines, Aer Lingus, Air Canada, Air France, Bravofly, British Airways, BudgetAir, CityJet, CrystalTravel, eDreams, Etihad Airways, Finnair, Gotogate, Iberia, Kiwi, KLM, lastminute.com, Lufthansa, Opodo, SAS, StudentUniverse, SWISS, Travelgenio, Tripsta, United
+ filter by alliance: oneworld, SkyTeam, Star Alliance
+ filter by cabin: student, economy, premium economy, business, mixed
+ filter by flight quality
+ filter by type of aircraft: narrow-body jet, regional jet, turboprop plane, wide-body jet
+ filter by price range
+ filter by booking providers
+ filter by departure times
+ filter by number of stops: no stop, 1 stop, 2 stops option to filter by travel types: flights, trains, buses
  Buses
+ filter by duration
+ filter by bus station
+ filter by bus companies
+ filter by price range
+ filter by number of changes
+ filter by departure/arrival time
  Trains
+ filter by duration
+ filter by departure/arrival time
+ filter by train stations
+ filter by railways: Thalys, Deutsche Bahn, Eurobahn, ABELLIO Rail, NS International, etc...
+ filter by number of changes: direct, 1 change, 2 changes

COMBITRIP[44]

+ option to choose flexible dates
+ filter by airports
+ filter by departure/arrival time
+ filter by stopovers: direct, 1 stop
+ sort by: price, duration, number of transfers, distance to airport
+ filter by travel type: airplane, train, bus, coach, ride share, car hire
- no api available

GOEURO[45]

+ filter by travel types: airplane, train, bus
  Airplane
+ filter by airports
+ filter by departure/arrival times
+ filter by duration
+ filter by price range
+ filter by airlines: Aegean Airlines, Aer Lingus, Air Europa, Air France, Alitalia, British Airways, Czech Airlines, Eurowings, Flybe, Iberia, KLM, Lufthansa, Norwegian Air Shuttle, Sas Airways, Swiss, TAP Portugal, Transavia Airlines, Vueling filter by number of changes: direct, 1 change, 2 changes
  Bus
+ filter by bus station
+ filter by departure/arrival times
+ filter by duration
+ filter by price range
+ filter by bus companies: Flixbus, RegioJet, Ouibus
  Train
+ filter by train station
+ filter by departure/arrival times
+ filter by duration
+ filter by price range
+ filter by railways: Thalys, NS International, Deutsche Bahn
+ filter by number of changes: direct, 1 change, 2 changes

FROMATOB[46]

+ filter by travel types: airplane, train, bus, ride share
+ filter by departure/arrival times
+ filter by duration
+ filter by number of changes: direct, 1 change, 2 changes
+ sort results by: price , duration, number of changes, departure/arrival times
+ available api (required request api key)
+ easy integration via deep link on your website

## B.4. Weather

### Weather Underground[47]

+ hour -by- hour 10-day forecast
+ yesterday weather summary
+ current tropical storms
+ dynamic satellite images
+ tides and currents
+ astronomy
- limit is 500 calls per day
- no historical weather data

### Dark Sky[48]

+ day-by-day forecast for the next week
+ minute-by-minute forecast for the next hour
+ observed weather condition on a date in the past
+ 1000 API calls for free each day (The counter resets everyday at midnight UTC)
- over 1000 API calls, account will be blocked
- required to display the message “Powered by Dark Sky”

### Weatherbit[49]

+ day-to-day 16-days forecast
+ 5 days forecast updates per 3 hours
+ limit is 75 calls per hour
+ one month historical data

### Open Weather Map[50]

+ Access current weather data for any location including over 200,000 cities
+ Current weather is frequently updated based on global models and data from more than 40,000 weather stations
+ 5 day forecast is available at any location or city
+ 5 day forecast includes weather data every 3 hours
+ Weather maps include precipitation, clouds, pressure, temperature, wind, and more
+ Connect our weather maps to your mobile applications and websites
+ air pollution data
- 60 calls per minute

APIXU WEATHER[51]

+ Real-time weather
+ 10 day weather forecast
+ Historical weather forecast
+ Astronomy
+ Time zone
+ Location data
+ Scalable

## B.5. TRAVEL INFORMATION

NUMBEO[52]

+ information about: cost of living, property prices, crime, health care, pollution, traffic, quality of life, travel
- no free api available

TRIPADVISOR[53]

- information about things to do
- general information about the destination
- reviews about the destination
- weather monthly calendar of travel destination
- filter by neighborhood
- filter by type of attraction: museum, tour, monumental, food/drink, nature and parks, night life, outdoor activities, shopping, transport, water sports, boat trips, recreation and games, concerts and shows, aquarium and zoo, travel help resources, workshops, spa and wellness, casino, other events
- TripAdvisor provides free, up-to-date rating content to select travel websites and apps through its Content API.
- The API can provide dynamic access to TripAdvisor content

BASETRIP[54]

+ basic information about travel destination: General info, Sockets and plugs, Weather
+ Getting Around: Car
+ Health and Safety: Crime, Vaccinations, Emergency Numbers
+ info about communications:Internet, Mobile phone, Dial code
+ Other information: Alcohol and Drugs, Embassy
+ info about money: Currency, Cost of living, Credit and debit cards, ATMs, Tipping
+ required request api key (Individual free if queries <= 1000 per month)
- Startup 249$ if queries <= 10000 per month

TELEPORT PUBLIC[55]

+ Searching for Cities by Name
+ Basic Information About a City
+ Local Reviews
+ Life Quality for Urban Areas: Housing, Cost of Living, Startups, Venture Capital, Travel Connectivity, Commute, Business Freedom, Safety, Healthcare. Education, Environmental Quality, Economy, Taxation, Internet Access, Leisure and Culture, Tolerance, Outdoors
+ Photos for Urban Areas
+ Embedding Linked Resources
+ Searching for Nearest Cities
+ currently no hard limit on the rate of requests issued against our APIs, but we ask that you contact us before starting to send thousands of requests per day.
+ anything you can see on our Teleport Cities Pages is available to use.

## B.6. IMAGES

PIXABAY[56]

+ free images
+ over 1 million available images
+ keyword search option
+ filter by image type: photography, illustration, vector
+ filter images per category: backgrounds/ textures, architecture/buildings, beauty/fashion, companies/financial, computer/communication, animals, emotions, food/drink, health/medical, industry/crafts, people, music, nature/landscapes, education, travel/vacation, religion, sports, cities/monuments, transportation/traffic, science/technology
+ filter by size
+ filter by color: transparent, black/white, other colors
+ filter by orientation : horizontal, vertical
+ 5000 API requests per hour
+ simple to set up
+ permanent hotlinking of images in not allowed (if the intention is to use the image download it to your server first)
+ link to Pixabay is required

TELEPORT PUBLIC[55]

+ Searching for Cities by Name
+ Basic Information About a City
+ Local Reviews
+ Life Quality for Urban Areas: Housing, Cost of Living, Startups, Venture Capital, Travel Connectivity, Commute, Business Freedom, Safety, Healthcare. Education, Environmental Quality, Economy, Taxation, Internet Access, Leisure and Culture, Tolerance, Outdoors
+ Photos for Urban Areas

+ Embedding Linked Resources
+ Searching for Nearest Cities
+ currently no hard limit on the rate of requests issued against our APIs, but we ask that you contact us before starting to send thousands of requests per day.
+ anything you can see on our Teleport Cities Pages is available to use.

GETTYIMAGES[57]

+ interactive tool to integrate api in existing website
+ search by image
+ filter by image type: photography, illustration, vector
+ filter by orientation: vertical ,horizontal, square, panoramic horizontal
+ filter by image resolution
+ filter by number of people
+ filter by age of people
+ filter by composition
+ filter by ethnicity
+ filter by image style
+ filter by color
+ filter by photographers
+ over 84 million images available
- images can be used only for noncommercial purposes
- downloading of images not available

UNSPLASH[58]

+ free developer api: 454.4M requests per month, 363.1K free photos
+ search photos by keyword
+ initially development mode website will be limited to 50 request per hour
- requirement to upload screenshot of website where Unsplash is used to get approval for the full rate limit amount
- available libraries: PHP, Ruby and Javascript
+ hotlinking of images allowed

# C

# Choice of Technologies

## C.1. Frontend Framework

### C.1.1. Requirements

The goal was to find a framework which does not make any assumptions about the rest of the software stack. Furthermore a system was wanted that would encourage modularity and extensibility.

### C.1.2. Options

#### React

+ Built with modularity and extensibility in mind.
+ Doesn't make any assumptions about the stack.
+ The earlier front end was built in React, so we will be able to reuse components from the previous versions.
- Have to get used to JSX.

#### Angular

+ Built with modularity and extensibility in mind.
- The earlier front end was built in React, so everything would have to be rewritten from scratch.
- Has been losing popularity recently.

#### Vue

+ Doesn't make any assumptions about the rest of the stack.
+ Built with modularity and extensibility in mind.
- The earlier front end was built in React, so everything would have to be rewritten from scratch.

#### jQuery

jQuery is a fast, small, and feature-rich JavaScript library.

+ Doesn't make any assumptions about the rest of the stack.
- Not built with modularity and extensibility in mind.

### C.1.3. Conclusion

We chose to use React, because one of our developers was well versed with it and because the previous front end was built in React and we thus would be able to reuse certain components.

## C.2. TESTING FRAMEWORKS

There are three layers, on which the front end requires testing,

- Logic (Unit Tests)
- Component (Integration Tests)
- Browser (End-to-End Tests)

This section concerns the frameworks available in which to build these tests, plus the test runner to run them. It's easy to become opinionated when discussing these matters, as they relate directly to work flow. To resolve this, the main frameworks were listed with their pros and cons.

### C.2.1. UNIT TESTING

#### MOCHA + CHAI

"Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing simple and fun." [19]

+ Runs in the browser
+ Fast [59]
+ Used extensively in the community
- Jasmine was already used in the back end
- Relies on Chai [20] for assertions

#### JEST + CHAI

"Complete and easy to set-up JavaScript testing solution. Works out of the box for any React project."

+ Many features
+ Easy to use
+ Works "out of the box"
+ Is growing
- Slightly slower [59]
- Jasmine was already used in the back end
- Relies on Chai for assertions

#### JASMINE

"Jasmine is a behavior-driven development framework for testing JavaScript code." [60]

+ Was used in the tests of the back end.
+ Fast [59]
+ Easy to use
+ Does not require a DOM
+ Works independently of other frameworks.
- Doesn't support e2e testing, but we can use WebdriverIO for that.

### C.2.2. INTEGRATION TESTING FRONT END

Because we are developing in React, it is the most sensible to perform the integration tests on component level. Instead of having to test the entire app, we can test individual components in isolation. Because of its virtual DOM, React even allows us to perform actions on a component without the browser environment.

For this purpose, Airbnb developed Enzyme [61], which is currently without real competition in this niche of front end testing.

#### ENZYME

+ Built specifically to test react components
+ More isolated integration tests
+ Only framework of its kind
+ Works well with any test runner
+ Allows us to perform integration tests without the browser
- Doesn't work outside of React

### C.2.3. END-TO-END BROWSER TESTING

#### NIGHTWATCH

"Nightwatch.js is an easy to use Node.js based End-to-End (E2E) testing solution for browser based apps and websites. It uses the powerful W3C WebDriver API to perform commands and assertions on DOM elements." [62]

+ Easy to use
+ Works out of the box
- Not very extendable
- Less Discussed on Stack Overflow

#### WEBDRIVERIO

"WebdriverIO lets you control a browser or a mobile application with just a few lines of code." [21]

+ More extendable
+ Very commonly used
+ More discussed on Stack Overflow
- Takes some more effort to configure

### C.2.4. INTEGRATION TESTING BACK END

#### SUPERTEST

"Super-agent driven library for testing node.js HTTP servers using a fluent API"

+ Easy to use syntax
+ Widely used
- No longer maintained

#### CHAI HTTP

"HTTP integration testing with Chai assertions."

+ Works with Chai, using the same syntax
+ Recommended by Chai

### C.2.5. CONCLUSION
It was decided to go with Mocha + Chai for the unit tests, as it was needed to rewrite both the front end and back end there was no need to stay with the previously used framework Jasmine. Enzyme was the obvious choice for integration tests for React and WebdriverIO was the framework of choice for the end-to-end tests in the browser. For the back-end it was decided to use Chai HTTP for the integration testing as it integrates perfectly with Chai's assertions.

## C.3. DATABASE
### C.3.1. POSTGRESQL
"PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness."[23]

+ Wide industry adoption
+ Fast queries and indexing
+ Covered in Curriculum
- Requires mapping between Javascript objects and relational data

### C.3.2. MONGODB
"MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need"[63]

+ Natively build on JSON documents.
+ Great integration with JavaScript.
- Not covered in curriculum

### C.3.3. CONCLUSION
PostgreSQL was chosen as our database, mainly because the knowledge already in the group to work with relational databases.

## C.4. HOSTING
There are 3 large companies all offering cloud hosting services, namely Microsoft Azure, Google Cloud and Amazon EC2. Since Wheretrip was awarded with a $2500 credit for Google Cloud it was decided to use their service.

# INFOSHEET

## WHERETRIP.COM

**Company:** Wheretrip
**Presentation date:** 6th of February 2018

Wheretrip, describing themselves as,

"A Dutch based start-up for travelers who like to explore new places, offering countless destinations within your budget. Being an official YES!Delft student start-up we have created our platform with the help of the best in-house experts available.

With our diverse team of experienced travelers we have carefully selected the best destinations for every type of trip: Whether you are a party animal, surfer or both, we have got you covered! By combining our wanderlust with our smart algorithms, we offer you trips which truly fit your theme and budget"

The company wanted to improve their website, and having had good experiences with BEP earlier, they decided to reach out to the Bachelor Program again. They proposed two major improvements in how things are done, namely: implementing an integrated booking system and personalizing the current search engine.

It would prove a great challenge to implement this functionality while maintaining the high code quality needed to make Wheretrip in a scalable application. After researching the possibility of an integrated booking system it was decided to be unfeasible to implement all functionality. Therefor it was decided to focus on the personalization part of the assignment while completely rewriting the poorly written front end, as well as cleaning up the back end. Due to internal problems in the group the outcome of the project was not completely successful. Nevertheless there are significant improvements made to the application and it provides a solid foundation for Wheretrip to grow on.

## TEAM

The team consists of three computer science students:
Rasmus Bergström,
Maria Simidžioski,
Gert Spek

## CLIENT

**Name:** Sando van der Helm
**Affiliation:** Founder at Wheretrip

## COACH

**Name:** dr. ir. O.W. Visser
**Affiliation:** Distributed Systems Group of the Faculty of Engineering, Mathematics and Computer Science at the Delft University of Technology

## CONTACT

Rasmus Bergström, jrasmusbm@gmail.com
Maria Simidžioski, mariasimidzioski@hotmail.com
Gert Spek, gert_spek@hotmail.com

The final report for this project can be found at `https://repository.tudelft.nl/`

# BIBLIOGRAPHY


[1] *Wheretrip about,* Retrieved from wheretrip.com (2017).

[2] *Read me | redux,* Retrieved from https://redux.js.org/ (2018).

[3] *Pci merchant levels 1 - 4.* www.pcipolicyportal.com/what-is-pci/merchants/ (2018).

[4] *Visa security compliance program updates.* https://usa.visa.com/support/small-business/security-compliance.html (2018).

[5] F. Ricci and F. Del Missier, *Supporting travel decision making through personalized recommendation.* in *Designing personalized user experiences in eCommerce* (Springer, 2004) pp. 231–251.

[6] B. J. Jansen and U. Pooch, *A review of web searching studies and a framework for future research.* Journal of the Association for Information science and Technology **52**, 235 (2001).

[7] J. L. Crompton and P. K. Ankomah, *Choice set propositions in destination decisions.* Annals of Tourism Research **20**, 461 (1993).

[8] S. Um and J. L. Crompton, *Attitude determinants in tourism destination choice.* Annals of tourism research **17**, 432 (1990).

[9] A. G. Woodside and S. Lysonski, *A general model of traveler destination choice.* Journal of travel Research **27**, 8 (1989).

[10] D. R. Fesenmaier and J.-M. Jeng, *Assessing structure in the pleasure trip planning process.* Tourism analysis **5**, 13 (2000).

[11] C.-C. Lue, J. L. Crompton, and D. R. Fesenmaier, *Conceptualization of multi-destination pleasure trips.* Annals of tourism research **20**, 289 (1993).

[12] W. W. Eckerson, *Three tier client/server architecture: Achieving scalability, performance, and efficiency in client server applications.* Open Information Systems **10** (1995).

[13] S. Burbeck, *Applications programming in smalltalk-80(tm): How to use model-view-controller (mvc).* (1987).

[14] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures.* (University of California, Irvine Doctoral dissertation, 2000).

[15] *Inversion of control containers and the dependency injection pattern - forms of dependency injection.* Retrieved from Martinfowler.com (2018).

[16] *Netscape and sun announce javascript.* Retrieved from Web Archive (1995).

[17] *Node.js.* Retrieved from nodejs.org (2018).

[18] *Typescript.* Retrieved from typescriptlang.org (2018).

[19] *Mocha - the fun, simple, flexible javascript test framework.* Retrieved from https://mochajs.org/ (2018).

[20] *Chai assertion library.* Retrieved from https://chaijs.com/ (2018).

[21] *Webdriverio - webdriver bindings for node.js.* Retrieved from https://webdriver.io/ (2018).

[22] *Supertest.* Retrieved from github.com (2018).

[23] *Postgresql.* Retrieved from postgresql.org (2018).

[24] *Git scm.* Retrieved from git-scm.com (2018).

[25] *Apache subversion.* Retrieved from subversion.apache.org (2018).

[26] *Gitlab.* Retrieved from gitlab.com (2018).

[27] *Typescript.* Retrieved from kubernets.io (2018).

[28] *React.* Retrieved from https://reactjs.org/ (2018).

[29] *Flux | application architecture for building user interfaces.* Retrieved from https://facebook.github.io/flux/ (2018).

[30] *Node package manager.* Retrieved from npmjs.com (2018).

[31] *Routing-controllers.* Retrieved from github.com (2018).

[32] *Typeorm.* Retrieved from typeorm.io (2018).

[33] *Typedi.* Retrieved from github.com (2018).

[34] *Expedia inc.* Retrieved from expediainc.com (2018).

[35] *Hotels combined.* Retrieved from hotelscombined.com (2018).

[36] *Go seek.* Retrieved from goseek.com (2018).

[37] *Kayak.* Retrieved from https://www.kayak.co.uk/ (2018).

[38] *Skyscanner.* Retrieved from https://www.skyscanner.nl/ (2018).

[39] *Dohop.* Retrieved from https://www.dohop.nl/ (2018).

[40] *Fly.* Retrieved from https://www.fly.com/ (2018).

[41] *Sta travel.* Retrieved from https://www.statravel.co.uk/ (2018).

[42] *Kiwi.* Retrieved from https://www.kiwi.com/ (2018).

[43] *Booking.* Retrieved from https://www.booking.com/ (2018).

[44] *Combitrip - international journey planner flights, trains, buses, rideshare and more.* Retrieved from https://www.combitrip.com/ (2018).

[45] *Goeuro.* Retrieved from https://www.goeuro.nl/ (2018).

[46] *Fromatob.* Retrieved from https://www.fromatob.com/ (2018).

[47] *Weather underground.* Retrieved from https://www.wunderground.com/ (2018).

[48] *Dark sky.* Retrieved from https://www.darksky.net/ (2018).

[49] *Weatherbit.* Retrieved from https://www.weatherbit.com/ (2018).

[50] OpenWeatherMap.org, *Open weather map.* Retrieved from https://www.openweathermap.com/ (2018).

[51] *World weather,* Retrieved from https://www.apixu.com/ (2018).

[52] *Numbeo.* Retrieved from https://www.numbeo.com/ (2018).

[53] *Tripadvisor.* Retrieved from https://www.tripadvisor.nl/ (2018).

[54] *Base trip.* Retrieved from https://www.thebasetrip.com/ (2018).

[55] *Teleoprt public.* Retrieved from https://teleport.org/ (2018).

[56] *Pixabay.* Retrieved from https://www.pixabay.com/ (2018).

[57] *Gettyimages.* Retrieved from https://www.gettyimages.nl/ (2018).

[58] *Unsplash.* Retrieved from https://www.unsplash.com/ (2018).

[59] V. Potapov, *Javascript test-runners benchmark.* Retrieved from https://medium.com/dailyjs/javascript-test-runners-benchmark-3a78d4117b4 (2017).

[60] *Jasmine.* Retrieved from https://jasmine.github.io/ (2018).

[61] *Enzyme.* Retrieved from https://github.com/airbnb/enzyme (2017).

[62] *Nightwatch.js.* Retrieved from https://nightwatchjs.org/ (2018).

[63] *Mongodb.* Retrieved from mongodb.com (2018).