



# What trip are you looking for?

M. Meuleman,  
C. M. Koster, and  
Y. E. S. S. Arkesteijn

Final report  
Bachelor Project







# What trip are you looking for?

by

M. Meuleman,  
C. M. Koster, and  
Y. E. S. S. Arkesteijn

to obtain the degree of Bachelor of Science  
at the Delft University of Technology,

Student names:	M. Meuleman	
	C. M. Koster	
	Y. E. S. S. Arkesteijn	
Project duration:	April 24, 2017 – July 7, 2017	
Bachelor Project committee:	Prof. dr. ir. D. H. J. Epema,	TU Delft, coach
	dr. ir. O. Visser	TU Delft, coördinator
	 ,	 , client

*This report is confidential and only the latest version can be made public after the 26th of July, 2017.*

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.



# Preface

This report is written for the completion of our Bachelor degrees in Computer Science. It describes the whole process of this ten-week project. The project has been finished in assignment for the start-up company [REDACTED]. This company combines budget flights and accommodations to provide affordable trips to customers who do not yet know where they want to go to. During these ten weeks, we have rewritten and restructured all the logic to make this application work. During this process we have had high regards to maintainability and extendability.

We would first like to thank Marcel Pennings from the company Yex, for providing our initial Bachelor Project. Unfortunately, this project was unfeasible to both Marcel's and our regret. We would also like to thank prof. dr. ir. M. Loog, who was our TU coach for our project at Yex.

We would like to thank all the people who have and who still are helping us in our project, especially dr. ir. O.W. Visser for his help with finding a new project and a new TU coach after our first project fell short. We also want to thank dr. ir. C.C.S. Liem because she offered us a project on such short notice, despite the large number of groups she already had agreed to supervise.

We also would like to thank [REDACTED], [REDACTED] and [REDACTED] for guiding us inside the company and providing us with the support and tools we needed to get the job done. We want to thank [REDACTED] for providing us with his technical expertise about the former system and server support and we want to thank [REDACTED] for supporting us with several issues, technical expertise and insight. Lastly, we want to thank Ruben van Gaalen for freeing up some of his time and creating a new front-end design for the company.

Note: To protect the company, the name of the company is not made public. This is also the case for the members of the company.

*M. Meuleman,  
C. M. Koster, and  
Y. E. S. S. Arkesteijn  
Delft, June 2017*



# Summary

This is the final report for the Bachelor Project. The project is finished in assignment for the start-up [REDACTED]. This company offers the cheapest trips by combining the cheapest flights with the cheapest accommodation. Their web application was built as a proof of concept and therefore put together as quickly as possible. The resulting system is not maintainable and not extendable, something the company needs since they are looking to extend their amount of partners to be able to provide more interesting trips for their customers.

The assignment, therefore, was to rewrite the code in a maintainable and extendable way. First, the requirements were made to keep clear what needed to be done and what did not. Also, the success criteria were set to make sure we could determine at the end of this project whether the project had succeeded. These requirements had to be researched on their feasibility. When this was done a design was made with extendability and maintainability in mind. This design was implemented and of course, some problems were encountered during this implementation. These problems had to be solved and reconsidered, which also result in some changes to the design. About halfway through the project, the first feedback from the Software Improvement Group was received. This feedback indicated that the code was above average maintainable, but needed some minor changes. These changes were implemented just as the last features and refactors were finalised, which ultimately resulted in a maintainable and extendable platform, a happy client and of course this report.





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Requirements</b>	<b>3</b>
2.1	Determining the requirements . . . . .	3
2.2	The Required features (Must haves) . . . . .	3
2.3	Features the project should have . . . . .	5
2.4	Expansions and future idea's (Could haves) . . . . .	5
2.5	Not relevant to the client (Won't haves) . . . . .	5
<b>3</b>	<b>Feasibility</b>	<b>7</b>
3.1	Feasibility of creating an integrated booking system . . . . .	7
3.2	Required features . . . . .	8
3.3	Features it should have . . . . .	8
3.4	Features it could have. . . . .	8
3.5	Features it won't have . . . . .	8
<b>4</b>	<b>Research</b>	<b>9</b>
4.1	Course of Action . . . . .	9
4.2	PCI Certification . . . . .	9
4.3	API requests. . . . .	11
4.3.1	Kiwi . . . . .	11
4.3.2	HostelWorld . . . . .	11
4.4	Back-end Technologies . . . . .	11
4.4.1	Node.js. . . . .	11
4.4.2	Socket.io . . . . .	11
4.4.3	Express. . . . .	12
4.5	Front-end Technologies . . . . .	12
4.6	Programming Environment. . . . .	12
4.7	Code Improvement . . . . .	12
4.7.1	Functional Suitability . . . . .	13
4.7.2	Performance Efficiency . . . . .	13
4.7.3	Compatibility . . . . .	13
4.7.4	Usability . . . . .	14
4.7.5	Reliability . . . . .	14
4.7.6	Security . . . . .	14
4.7.7	Portability . . . . .	14
4.7.8	Maintainability . . . . .	15
4.8	Schema validation . . . . .	15
4.9	Ethics . . . . .	16
<b>5</b>	<b>Design</b>	<b>17</b>
5.1	Trip . . . . .	17
5.2	Transport . . . . .	18
5.3	Accommodation . . . . .	19
5.4	Themes . . . . .	20
<b>6</b>	<b>Implementation</b>	<b>21</b>
6.1	Back-end Implementation . . . . .	21
6.1.1	Transport . . . . .	21
6.1.2	Accommodation . . . . .	22
6.1.3	Trip . . . . .	22

---

6.1.4	Theme . . . . .	22
6.1.5	City . . . . .	22
6.1.6	Logger . . . . .	22
6.1.7	JSON validation . . . . .	23
6.1.8	Adapter . . . . .	23
6.2	Front-end Implementation . . . . .	23
<b>7</b>	<b>Software Improvement Group</b>	<b>25</b>
7.1	Unit Size . . . . .	25
7.2	Unit Interfacing . . . . .	26
7.3	Final feedback . . . . .	26
<b>8</b>	<b>Evaluation</b>	<b>27</b>
8.1	Requirements . . . . .	27
8.2	Testing . . . . .	28
8.3	Success Criteria . . . . .	28
<b>9</b>	<b>Discussion and Recommendations</b>	<b>29</b>
9.1	Process Evaluation . . . . .	29
9.2	Recommendations for the back-end . . . . .	29
9.3	Recommendations for the front-end . . . . .	30
9.4	General Recommendations . . . . .	30
<b>10</b>	<b>Conclusion</b>	<b>31</b>
	<b>Appendices</b>	<b>33</b>
<b>A</b>	<b>Framework Research</b>	<b>35</b>
<b>B</b>	<b>Research IDE</b>	<b>39</b>
<b>C</b>	<b>Research Static Analysis Tools (SAT)</b>	<b>41</b>
<b>D</b>	<b>UML</b>	<b>43</b>
<b>E</b>	<b>Original Project Description</b>	<b>45</b>
<b>F</b>	<b>Coverage</b>	<b>47</b>
	<b>Bibliography</b>	<b>51</b>



# Introduction

This project was started with the company Yex, who had an interesting project available. Unfortunately, this project seemed unfeasible. Therefore, to regret of both parties, the project had to be cancelled and this team had to switch to a different company. This new company is the start-up ██████████.

██████████ is a start-up company with an on-line platform which started last year. The goal of the platform is to offer cost-effective options for travelling. The most budget-friendly flights and hostels are combined to create these cost-effective trips. This platform is mainly focused on customers who do not yet know where they want to go to, but want to make a low-budget trip. These customers can select a theme of their trip; if they want to go on a surfing trip, for example, all surf destinations are selected.

The company was founded by a five-person team and has been working since May 2016 on a feasibility study. In September that same year, another person joined the team as the senior programmer. Currently, the first five people are still active as strategists, whose main tasks include reviewing and revising the vision of the company and performing user tests by talking to travellers. Two people are currently active as part-time tech specialists, one of them is the lead developer at the company and one of them an intern who was recently acquired by the company.

At this moment, the current system was only built with a conceptual phase in mind and finished as quickly as possible. This concept was used to test the product with different user groups and now that ██████████ deems this concept feasible, a reliable version of the system has to be created, whilst keeping maintainability and extendability in mind. This is a necessary step in the development of the company. If they want to grow and produce a product that is viable and adaptable in the future, the quickly-made concept has to be restructured and rewritten. This will also help future developers that will work with ██████████ since they can focus on building features instead of having to deal with code that is unstructured and polluted. In the end, this is a necessary investment, in order to make the product more future-resistant.

In the next chapter, the requirements for this application will be stated. This is needed for a good overview of what has to be done to make the application as the client wants it. Chapter three will then question whether this assignment is feasible. During and after this feasibility study, more research has to be done, which is discussed in chapter four. When everything is researched, chapter five discusses the initial design of the system. This design was made to pay extra attention to the maintainability and extendability. This design has to be implemented and possible issues have to be taken care of. The problems which were encountered and the solutions for those are discussed in chapter six. After seven weeks the feedback of the Software Improvement Group (SIG) was received. Chapter seven states that feedback and discusses what actions were taken to improve the system based on this feedback. Eventually, the product has to be evaluated which is done in chapter eight. In chapter nine, there are some recommendations for the client and a discussion about the process. Finally, chapter ten will conclude this report.



# 2

## Requirements

A first essential step before implementing the application is to be clear on what the requirements for the application need to be. In this way, it will become apparent what we need to implement, what we won't implement and what will be the most efficient order to implement these requirements. Also, this will help to determine what the design will look like and to make the implementation maintainable and extendable. All of this will be discussed in chapters 5 and 6.

### 2.1. Determining the requirements

These requirements are determined and divided to fit the MoSCoW method [13]. The main reason this method is used is the priority it gives to some requirements over others. In this way, the important requirements are distinguished from the less important requirements, which could be seen as optional.

To evaluate the system afterwards and determine whether the implementation is a success, success criteria should be determined. This is done with the help of the requirements in the next section. All the “must have” requirements need to be implemented, because they are the core functions of the systems. Without those requirements, the system would not work as it is supposed to. From the “should have” requirements should at least 80% be implemented. Without those requirements, the system works, but it would still miss important features. The “could have” requirements indicate optional features, so these will only be implemented if there is time left. They will not be present in the success criteria. Finally, the won't requirements will not be implemented, because they are simply out of the project scope.

There are some more success criteria, all of them are listed here:

- All “must have” requirements have to be implemented.
- 80% of the “should have” requirement have to be implemented.
- The system has to be able to operate in a real-time environment (it should be able to go live).
- The wishes of the Client should be satisfied as much as possible.

### 2.2. The Required features (Must haves)

- The user must be able to choose a theme when going to the web application.
- The user must be able to change the chosen theme.
- There must be six themes from which the user can choose:
  - Sun theme
  - Surf theme
  - Winter theme
  - Party theme

- City theme
  - Best and Cheapest theme (all cities)
- The user must be able to set:
  - the departure date.
  - the arrival date.
  - the place from where the trip must start.
  - the maximum price the user is willing to spend.
  - the minimum price the user is willing to spend.
  - the amount of passengers going on the trip.
- After loading the trips, the user must be able to see all the trips in a summary.
- In this trip summary the following items should be visible:
  - the total price of the trip.
  - the destination city and country of the trip.
  - the price of the flight.
  - the price of the accommodation.
  - the departure time and date of the trip.
  - the arrival time and date of the trip.
- The user must be able to click on a trip summary.
- When a user clicks on a trip, all the information of the trip must be shown.
- All the trip information must contain information about the flight.
- All the flight information must contain:
  - Departure flight date
  - Duration of the flights
  - Time of departing from the departure airport
  - Time of arriving at the arrival airport
  - The different flight segments with their airports and times
  - Return flight date
  - Price
- All the trip information must contain information about the accommodation.
- All the accommodation information must contain:
  - Price
  - Image of the accommodation
  - Rating
  - Name of the accommodation
- The user must be able to book the flight via [Kiwi.com](http://Kiwi.com).
- The user must be able to book the accommodation via [HostelWorld.com](http://HostelWorld.com).
- The server must filter the accommodations:
  - on availability
  - on price combined with the transport
- The system must be more extendable.
- The system must be maintainable according to the Software Improvement Group.

### 2.3. Features the project should have

- A trip summary should have an image related to the destination city of the trip.
- The user should be able to see the total amount of trips which are available for the given parameters.
- The accommodation information should show the distance to the airport.
- The trip information should show a map with the destination pointed out.
- The server should send the data in a JSON format to the client.
- The server should validate the data it gets from:
  - SkyPicker
  - HostelWorld
  - The client
- The client should validate the data it gets from the server.

### 2.4. Expansions and future idea's (Could haves)

- The user could see a loading bar which indicates how many trips already have been loaded.
- The user could be able to set the range for an airport in comparison to the place where the trip starts.
- The user could be able to set the preferred destination.
- The start location could be computed through GPS data.
- The user could get special deals which are trips for a different period.
- The trip summary could contain the logo of the hostel provider.
- The server could filter the accommodations on a rating.
- The application could get a new design

### 2.5. Not relevant to the client (Won't haves)

- A log-in system for users.
- Include non-airport cities.
- Create a booking system within ██████████





# 3

## Feasibility

Now that the requirements are known, the next step is to find out whether those requirements are feasible. When we have agreed on a set of requirements that is feasible, a design can be made. This will be discussed in chapter 5.

During the first week of the project at [REDACTED], the project consisted of a different assignment. This assignment was deemed unfeasible after the initial research; this chapter's first section describes this assignment. After this, the chapter goes through the different requirement priorities of the current assignment. All the research which was done is discussed in chapter 4.

### 3.1. Feasibility of creating an integrated booking system

The current assignment was not the original assignment. The original assignment was concerned with [REDACTED] booking system. At the moment their customers have to pay two times; once for the flight and once for the accommodation. These two booking processes are both on a separate external web application (one for each partner of [REDACTED]). This is far from ideal for the customers and can even be a big risk. When the flight is booked and the customer waits a while before booking the accommodation, the accommodation might be gone and the customer is left with nothing. Please see appendix E for the original project description.

The assignment was to combine these two booking processes on the web application of [REDACTED]. This would make the whole process for the customer a lot more convenient. However, this project was deemed unfeasible after researching the possibilities with the given APIs. Kiwi's API (used from SkyPicker) was fairly permissive and it became soon apparent that this would not give us any problems. For the API of HostelWorld it was a lot harder to find out what the possibilities were however. At first, the internet could not tell us anything about it. A logical next step was to contact the company. This was much harder than expected. After a lot of calls and emails, there was still only a minimal form of contact. At the end of the feasibility study, they responded that it is not possible in the standard API, to get the booking system on [REDACTED]. They had to customise this.

Before HostelWorld would let us use a customised API, they made a few requirements of the platform. The first one, concerning market share, was that we would need to prove [REDACTED] had already successfully helped 150 users. This would maybe be doable if the company could take the time and manpower needed for this kind of advertisement, although it might take a while. The second requirement, however, was less feasible. HostelWorld requires their partners to have a PCI-compliant platform. After a little bit of research into this topic, this requirement, and therefore the entire API partnership, was deemed unfeasible by both [REDACTED] and us. Because of the minimal contact, the expectation was that this would also take much effort and time. So, unfortunately, from this study it was concluded that it is not feasible to solve this problem in the amount of time there is for this project. See chapter 4 for this research, especially the part about PCI certification.

During this research, there was already noticed that the code behind the web application was not at all maintainable and not extendable, since the code was built with a conceptual phase in mind. So after

finding the outcome of the first feasibility study, and after hearing from the company that they were looking at expanding to more partners (and therefore more APIs), we decided to jump to the current assignment on rewriting the platform in a maintainable and extendable manner.

### 3.2. Required features

The most important features to check for feasibility are of course the required features (must have). The biggest part of the feasibility study here is to find out whether all the data needed from the APIs are available. The data needed for this assignment can be found in the requirements in chapter 2. After doing this research, this assignment did seem feasible and the whole assignment started to take shape. All of the research can be found in chapter 4 as well, especially the part about the APIs (section 4.3).

The booking options at the partners' platforms are also a requirement for [REDACTED]. The customers will need to be redirected to the proper platforms when booking a flight and a hostel. For the course of this project, these platforms will be the Kiwi and HostelWorld platforms, since they are the only current partners of [REDACTED]. For these redirects, the proper URLs are needed. Luckily for us, this is already handled in the former system and therefore is feasible to implement in the new system.

### 3.3. Features it should have

The main part of these requirements is the validation of the data. All data is sent and received in JSON format, a standard in web development. The data that is received at the server, from both the APIs as well as the client, have to be validated to make sure we receive the correct data types. Without this validation, the system might still do what it is supposed to do, but we cannot be sure. When a wrong data format is received by the application things could go wrong and a wrong output might be produced. This is the reason validation is the most important secondary feature; besides that the system will work because of the required features, this feature makes sure the system works properly. The research on the JSON validation can be found in section 4.8.

### 3.4. Features it could have

These requirements are not necessary but might prove to be of added value. So if some of these requirements are unfeasible, it will not be a big problem to take them out of the backlog. A number of the requirements in this category are already implemented in the current system, which would suggest they are also feasible in the implementation of the new system. Some of them require more work, such as the new front-end design. However, with the current technologies, this would definitely be a feasible feature.

### 3.5. Features it won't have

The requirements that are not relevant for this assignment of course also do not need a feasibility study, because they will not be implemented. During the rest of this report, these features also will not be mentioned again. This section is only included for completeness.

# 4

## Research

This chapter includes all the research which has been done. Most of this is done in the first two weeks of the project, but some of the research was done later in the project. Examples of this are the front-end technologies React and LessCSS. In those first two weeks, a course of action was defined which is also discussed in this chapter.

### 4.1. Course of Action

After the feasibility study, a plan of action was created to outline the big picture and planning of the project. At first, the whole system has to be rewritten as a whole to improve the code. This probably will take a big part of the available time into account. At the moment, there are only a few files with functions and both the files and functions are huge. To make this process as smooth as possible, first, there has to be made clear how the code is going to change. In other words, how the functions are divided into multiple functions, how files are divided into classes, but also how redundant code will be removed. This all has to be done without changing the functionality of the code.

After a few weeks, the code has to be checked by the Software Improvement Group (SIG). This is a really nice check to get information how the code can be improved even more. At the moment, the code which is now available would probably fail all criteria provided by SIG.

When this code is improved, it will be much easier to add new features to the system. When there is still time at the end, we would like to discuss with ██████████ what the options are concerning this. An example for this would be changing the booking process. As discussed in the previous section, it will not be possible to create only one payment out of the two payments, but we can simplify the process for the user. This can be done, for example, through a wizard which shows how to get through the process. Also, when the code is improved, the platform will be smaller, because there is much less redundant code, which means it is easier to maintain and extend.

This global planning has been put into table 4.2. The first week of the Bachelor Project phase was unfortunately lost to another project that was cancelled. In week 6 a SIG deadline is scheduled. Our goal is to finish as much code as possible before then (without losing the quality of code) to be able to check as much code as possible. The week after this has been reserved for implementing SIG feedback and the possible extension for the parts we could not finish yet. The weeks after this are not planned yet. This is very much dependent on the company and the team, we have to come together at this point and discuss next steps. An example of this was already mentioned briefly during one of our meetings: a wizard that guides users through the booking process might be a good alternative for the unfeasible booking process. This is just one of the possibilities and these weeks will be filled in after the first SIG deadline.

### 4.2. PCI Certification

As stated in chapter 3, the current assignment was not the original assignment. To make the booking of the HostelWorld accommodations, ██████████ would need a PCI Certification. This section explains what this PCI Certification is and why it is not a possibility for ██████████ at this moment.

Week	Tasks
1	Visit previous project, break with previous project, find new project
2	Start at ██████████, feasibility study, designing contract, research report
3	Familiarizing with codebase back-end, setup code stack, finalize research report, architecture design back-end
4	Implement back-end, tweak code stack (findBugs, coveralls, headless chrome for karma)
5	Study front-end codebase, research front-end stack, design front-end architecture (start front-end implementation)
6	Front-end implementation SIG DEADLINE (Thursday)
7	Implement SIG feedback, extension front-end implementation
8	(extension) possibility for extra features
9	possibility for extra features
10	SIG DEADLINE (Monday), possibility for extra features, finalize report
11	Finalize project, presentation

Table 4.1: Course of Action.

PCI-Certification requires companies to oblige to the Payment Card Industry Data Security Standard (or PCI DSS). This certification is required of all companies that handle credit card information and credit card transactions. The main idea behind this certification is that credit card information is highly susceptible to theft and misuse and companies should be sufficiently careful in the way they handle this information. The practical side behind this is a little bit less colourful. The criteria required of the companies are very steep and financially challenging.

This standard has six objectives. The first objective is about the security of the network. A secure network requires a firewall which will not cause inconvenience for the customers. Also, the customer must have a secure authentication. This means that the password must not be any default and the customer must be able to alter this password frequently.

The customers will give information about themselves and their credit cards. The second objective is about the protection of this data. It does not matter where this data is stored, it has to be protected. This data sometimes has to be transmitted from the client to the server and vice versa. When this data is transmitted, it must also be secure. The way this is done is to encrypt the data. There are many ways to do this. [2]

There are always hackers who try to get to the data of the customers. The system should be protected against those hackers, which is the third objective. This can be done through multiple software, such as anti-virus and anti spy-ware software. But this also concerns the bugs in the system. If the system contains a bug which allows a hacker to get to the personal data, the previously mentioned software will not matter.

The fourth objective is about controlling the access. The access of the personal data should be controlled and restricted. Every user in the system should also have a unique identification number which is confidential. With this number, the access can be controlled.

The software for prevention of malicious software, discussed in the third objective needs to be updated once in a while. Therefore the fifth objective is that the network must be monitored all the time. This also has to ensure that other security measures work how they are supposed to work. Finally, the sixth objective states that the security policy must be defined. This might change over time, so it also has to be maintained and followed by every party. [14]

The consequences for start-ups and other small companies like ██████████ are very big. The financial support needed for the certification cannot be brought up by the company and also the time and resources needed to transform the current system are not possible to bring up in the short amount of time reserved for this project.

## 4.3. API requests

This section discusses the two APIs which are used within ██████████. The first API which will be discussed is the one of skypicker.com which is used by Kiwi. The second API is the API for eXternal Services (XS). This API is used by HostelWorld. Both these APIs are currently necessary to book a trip on the platform.

### 4.3.1. Kiwi

Kiwi is a website where you can book flights. Currently, Kiwi only sells economy class tickets. It uses the skypicker.com API, which was also the previous name of Kiwi. This API is called from ██████████ to get the information of the flights in order to give the best recommendations. This information can contain information as the destination of a flight and departure and arrival times. Skypicker has a nice feature with this. Normally when calling a flight API like this, you are required to provide the IATA code for the airport or the ISO country code. The skypicker.com API is not restricted to this since it is also possible to give a kind of description as a destination. This would be something like “party destination” or “surf destination”. [11]

### 4.3.2. HostelWorld

Another API used in ██████████ is the XS API which is used with HostelWorld. This API is used to book a hostel. ██████████ first has to check which hostels are available. These hostels are then shown to the client. After this, ██████████ retrieves the information about these available hostels which will also be shown to the client.

When a client wants to book a hostel, ██████████ has to make a “property booking request”. This request will reserve that hostel, for the specified dates, for thirty minutes. ██████████ will receive a Booking session ID, called a BSID. This BSID guarantees that only the client with the correct BSID is able to book that hostel in that thirty minutes.

Finally, the hostels have a predetermined minimum and maximal duration. If a client wants to stay longer or shorter he or she has to choose another hostel. This is called the seasonal night’s rules.

## 4.4. Back-end Technologies

██████████ already uses some technologies and environments. This section will discuss these currently used technologies in the back-end of the system.

### 4.4.1. Node.js

██████████ uses the environment node.js (also called node). Node is a JavaScript environment on the server side. It is a very powerful and expressive, but at the same time lightweight and efficient because of its event-driven and non-blocking I/O model.

The main difference of node in comparison with other environments which are used a lot these days is the lack of multithreading. Node does not rely on multithreading, which can have an impact on the performance, depending on how it is used. The performance has been optimised in another way, however, since non-blocking I/O is used as a core principle. This means processes are not waiting while I/O processes are being run, which generally takes most of the time.

There is also some use of libraries in node in the application of ██████████. These libraries add some functionality to node and can be installed via the Node Package Manager (NPM). Some important libraries will be discussed, which are socket.io and Express. [9]

### 4.4.2. Socket.io

Socket.io is a library for node which is dealing with WebSockets. These WebSockets take care of the connection between the server and the browser of the client. The server can answer a message from the client event-driven, which is much more efficient than when you would do this without WebSockets. Without WebSockets the client would have to poll the server constantly, waiting for a reply, which is very inefficient. [3]

The WebSocket protocol exists of two parts. The first is the handshake between the client and the server, to confirm that they are going to exchange data which is done in the second part. This handshake is not only done at the beginning but also at the end, so that the server and client know the data transfer is completed.[5]

However, socket.io is more than only WebSockets. WebSockets are not always supported in every browser. In this case, socket.io has the possibility to use a number of other protocols for data transportation. This gives

rise to the big advantage of socket.io: an easy-to-program API that is supported by a lot of browsers. This relieves a lot of the burden of a developer. [12]

#### 4.4.3. Express

Express is a framework developed for node. There are a lot of useful features in this framework that makes development a whole lot easier. It can help with issues like routeing static and dynamic content, connecting other frameworks to the server (like databases) and basically provides an MVC architecture on the server.

As for socket.io, Express also makes it a lot easier for the developer. Some processes are already defined in Express. In this way, the developer does not have to reproduce this function and only have to call the function in Express. Express can only be used in combination with the HTTP module. This module deals with the HTTP responses and requests. [7, 12]

### 4.5. Front-end Technologies

On the client side, ██████████ makes use of Javascript in combination with some HTML and CSS. The HTML and CSS only make the static content of the page. JavaScript is used to create the dynamic content on the page. The data rendered at the server side (like trip data) can be dynamically added to the platform with JavaScript.

JavaScript is already embedded in a lot of web systems, so a lot of documentation can be found, which makes it a reliable technology. It is a language in which could be programmed in a number of ways, where function-oriented programming the most common choice is. Syntactically, this language is similar to languages such as Java and C++. A difference with Java, for example, is that JavaScript is an untyped language. This means that values are not bound to a specific type, which can have advantages as well as disadvantages [6].

For the new front-end design, there were several options. One possibility was to go with the HTML/ CSS/ JavaScript stack as in the old front-end. But with the new front-end technologies, like ReactJS, proving their worth we decided to switch to a combination of ReactJS and LessCSS. ReactJS is a new JavaScript library that allows a programmer to build component-based applications and web pages. It is a full JavaScript library, but is extended with the JSX markup, which sort of resembles XML/HTML. However, in the HTML/CSS/JavaScript stack, static content and dynamic logic were strictly separated, with ReactJS these boundaries disappear. JavaScript can be and is encouraged to be used within the JSX markup language, which assures that pieces of JSX and JavaScript that belong together, also stay together. This gives a much clearer overview of the system. For styling LessCSS was chosen as an extension to classic CSS. There are a few CSS preprocessors out there, LessCSS is one of them, others are SASS and Stylus. The reason we chose for LessCSS was the simple fact that one of us was already familiar with it. Researching the other options indicated that the choice did not really matter, each one of them had a few pros and a few cons, so we went with the familiar one.

### 4.6. Programming Environment

All of the current programming on the application is done in Node.js, without any tools to ensure code quality. To make sure that the product achieves the highest standard in code quality. Luckily two of our team members already have experience with developing quality code on the JavaScript stack. So here are our research results from TI2806 - Context Project. See appendices A, B and C.

### 4.7. Code Improvement

As stated before the code has to be improved a lot. This chapter will discuss the methodologies and standards which will introduce during this project to makes sure the system is programmed in a good way. The program has to work correctly for business success and so it is important to create a high-quality product. After the code has been altered, the code quality has to be measured. The question now is how to measure this quality. In the history, a lot of people tried to find standards for this. A few models are:

- McCall Model
- Boehm Model
- FURPS Model
- Dromey Model

- BBN Model
- Star Model

All these models have disadvantages which are the reason why these models did not become a standard for measuring the software quality [1]. Instead, the standard for measuring software quality is the ISO model. First ISO 9126 was used, but in 2011 the standard became ISO 25010. In figure 4.1 you can see a representation of this model. The quality can be divided into eight categories, but these categories are still hard to measure. So these categories are divided into a total of 31 subcategories.



Figure 4.1: Software product quality

#### 4.7.1. Functional Suitability

Every system has functionalities which are needed. The Function Suitability represents in what degree the system meets these functionalities. As seen in the image above this category has three subcategories:

- Functional Completeness: How many of the needed functionalities are covered.
- Functional Correctness: In how many cases does the system provide the correct results with the pre-specified degree of precision.
- Functional Appropriateness: In what extent does the functions makes it easier to

#### 4.7.2. Performance Efficiency

The performance of a system is also an important category. This category is about this performance relative to the resources which are used by the system. This category has the following three subcategories:

- Time Behaviour: Every functionality is allowed to process a predetermined amount of time. This subcategory states in what degree the real processing times are met with these predetermined times.
- Resource Utilization: This is the same as the previous subcategory, but now with the amount and type of the resources used by the system.
- Capacity: This is also the same as the previous subcategories, but this subcategory is about the maximum limits of the system.

#### 4.7.3. Compatibility

For systems, it is often important to exchange data with other systems. Or components have to exchange data with other components. Additionally, both systems or components have to perform their functions while sharing the same environment. In what degree the system is able to this, is the subject of this category. This category has two subcategories:

- Co-Existence: In what extent the system can perform its functions which are needed while sharing an environment with another system or multiple systems. This sharing has to be done without a damaging impact on one of the other systems.
- Interoperability: In what extent, the systems can exchange information and use information from other systems.

#### 4.7.4. Usability

The system has to be used by users eventually. This category is about in what extent these users can use this system to achieve their goals. This has to be achieved with efficiency, satisfaction and effectiveness. This category has the following six categories:

- **Appropriateness recognizability:** A user has specific needs. This subcategory is about in what extent the user can recognise if the system will fulfil these needs or not.
- **Learnability:** How far can the user use the system and learn from it to use the system effective, efficient, without risks and satisfied.
- **Operability:** In what extent the system contains attributes which make it easy to control and operate.
- **User Error Protection:** Errors are not uncommon in a system. But in how far is the user protected for making these errors.
- **User Interface Aesthetics:** In what extent the user interface pleases and satisfies the user for interacting with this system.
- **Accessibility:** People have a wide range of capabilities and characteristics. In what extent can the system be used by different people to achieve their goal.

#### 4.7.5. Reliability

In what extent performs the system it functions in a given amount of time under given conditions. This category has four subcategories:

- **Maturity:** In what extends meets the system its needs to be reliable given the normal circumstances.
- **Availability:** In what extent the system is accessible and operational when it is needed by a user.
- **Fault tolerance:** A system can have some faults in it. In what extent does the system perform as it is needed despite those faults.
- **Recoverability:** When a failure or interruption occurs, it is needed that the data and state of the system are recovered. This subcategory is about in what extent the system is able to do this.

#### 4.7.6. Security

Security is a subject which is very important. Systems have to protect their data and information so that other parties cannot access this without an appropriate level of authorization. This category is about the extent in which the system is able to do this with the help of the following five subcategories:

- **Confidentiality:** In what extent only the parties which have the correct level of authorization are allowed to access the data.
- **Integrity:** In what extent parties with not the correct level of authorization are prevented from accessing the data.
- **Non-Repudiation:** In what extent the system allows to prove that certain events have taken place so that, in the future, the events cannot be denied.
- **Accountability:** In what extent the origin of the action can be traced.
- **Authenticity:** In what extent there can be proved that the identity is correct.

#### 4.7.7. Portability

In what extent it is possible to transfer the system from one environment to another environment. This category has three subcategories:

- **Adaptability:** In what extent the system is able to adapt to a different environment.
- **Installability:** In what extent the system is able to be successfully installed and uninstalled in an environment.
- **Replaceability:** In what extent the system can replace another system in the same environment and for the same purpose.



	Volume	Duplication	Unit-size	Unit-complexity	Unit-interfacing	Module-coupling	Component-balance	Component-independence
Analysability	•	•	•					•
Modifiability		•		•		•		
Testability	•			•				•
Modularity						•	•	•
Reusability			•		•			

Figure 4.2: Maintainability factors

### 4.7.8. Maintainability

A system, of course, has to be modified after a while. Reasons for this could be improving, correcting or adapting to changes in its environment or requirements. This category is about in what extent the system is able to do that. This category has five subcategories:

- **Modularity:** In what extent the components are discrete. In other words, when you change one component it has to have minimal impact on the other components.
- **Reusability:** In what extent a part of the system can be used more than once.
- **Analysability:** When systems need changes, it has to be assessed what the impact will be, what parts has to be changed and what failures might occur. This subcategory is about in which extent the system is able to do this.
- **Modifiability:** In what extent the can be changed without creating new defect or decreasing the quality of the product in an effective and efficient way.
- **Testability:** In what extent the system can be tested to determine whether the test criteria (which also have to be established in an effective and efficient way) have been met.

[10]

However, there are some disadvantages of this model. How would you measure all the characteristics? In what unit do you have to measure these characteristics? And when is something good or wrong? The Software Improvement Group (SIG) has answered these questions for one category: Maintainability. They have chosen maintainability because this category has a strong influence on the other categories. If the maintainability of a system can be empirically improved, there is a stronger guarantee over the other categories.

In image 4.2 you can see that the subcategories are related to some system properties. These properties can be measured.

Not all things of the above characteristic are going to be measured due to time reasons. Some characteristics will not be efficient due to little profit returned from a lot of time needed to measure this.

## 4.8. Schema validation

The data of the API's are received in a JSON format, which is shorthand for JavaScript Object Notation. This format is very popular and, as stated before, it is used for the exchange of data between clients and servers. JSON is readable and very light-weight. It has a close resemblance with JavaScript, therefore the name. But it is not dependent on JavaScript, other programming languages can use JSON as well [15].

To validate the JSON arriving at the server, JSON schemas can be used. In such a schema, you can specify which fields a JSON object must have and of what type they must be. There are six types available in the JSON

schemas: Number, String, Array, Object, Null and Boolean. Next, to these types a field in the JSON object can also have a description, a default value or an array with fixed values that field must have (an enumeration). These are all optional specifications for the fields.

In addition, you can combine schemas. There are four keywords for this: allOf, anyOf, oneOf, not. For example, oneOf indicates that exactly one of the schemas must be valid.

A lot more is possible with these schemas like optional fields, regular expressions etc. See [4] for the documentation of JSON schemas.

## 4.9. Ethics

There are not that obvious ethical issues in this project. All the payment data is given on an external site. However, there is one small ethical issue left: Trust.

In section 4.2 the PCI certification was discussed, but even such a PCI certificate does not always give enough certainty for the customer. The customer must have trust in the application before he/she is willing to give personal information [8]. Possible reasons for the customer to not trust the web application might be a lack of trust in the security of given data, lack of trust in the distribution of given data or lack of trust in the product result (in this case the process of booking a trip). The design of the web application will also be an important aspect for this trust. An application with a bad design will create distrust with a client.

Trust has a strong relationship with a number of risks there are for doing a particular online purchase. There are two kinds of risks: the system dependent risks and the transactional risks. The system dependent risk is the risk that is related to the infrastructure. The transactional risks are the risks that are concerned with the transaction parties and processes. A consumer's trust is more correlated with the transactional risks and less with the system dependent risks, so an online merchant should focus on preventing transactional risks rather than preventing system dependent risks [16]. Consumers who are already familiar with making online payments are easier to convince because they know the process of payment. But still, these people also need to trust the merchant before they make a purchase.

Table 4.1 shows that risks are dependent on a multitude of things such as district, age and experience. The higher the ratio, the higher the observed risk is dependent on that factor. This can help the merchant to focus on the risks which are more important for the target group. If for example, economic risks are not really seen as a risk in the district he is selling his product, the merchant can focus on functional risks.

All these things are not only applicable to our first assignment, the creation of a booking system. These are still applicable if these functionalities are hosted on the partner's sites. Before a customer arrives at a Partners platform, he or she still needs to go through the entire flow in the platform. If the customer gives up trust in the platform, he or she will not even have the chance to arrive at the partner's platform. This means a loss of customers for [REDACTED], which is, of course, bad for business. The owners also want a log-in system for a better user experience. A customer should be able to give its preferences and information once, after that it should be remembered by the system. Before a customer is giving this information, the customer needs to trust the platform is secured by the merchant and will not give away his or her private information.

	Gender	District	Age	Experience
Economic risk	0.109	0.855	0.305	0.008
Functional risk	0.486	0.000	0.519	0.044
Security risk	0.798	0.008	0.182	0.365
Time risk	0.568	0.383	0.002	0.002
Privacy risk	0.229	0.000	0.061	0.344
Social risk	0.423	0.720	0.261	0.038
Service risk	0.411	0.000	0.077	0.939
Psychological risk	0.469	0.041	0.177	0.099
Total risk	0.833	0.000	0.439	0.000

Table 4.2: Significance of difference between classified groups by control variables.

# 5

## Design

The goal of this project is to rewrite the application in such a way that is maintainable, extendable and scalable for future growth. To accomplish this goal, the design had to be thought through in advance. This chapter will explain the design choices that were made to accomplish these objectives.

### 5.1. Trip

When you see the web application, the main part is the result of the customer's query; a list of trips that satisfy the customers search parameters, which can be changed and filtered through the options menu. This is also the only part where data is required from the server and thus the only part that is really interesting to design when rebuilding the server. The first logical step would be to represent those trips by a `Trip` object on the server.

So it is important to find out what a `Trip` consists of. In this case a `Trip` consists of some way to get there, some way to get back, and multiple ways to stay at the place the customer is going. This is the general case in the program but in the old version of the software these options were not bound by interfaces. Instead they were implemented only in `SkyPicker` flights and `HostelWorld` accommodations, instead of keeping a general structure.

To ensure that the program is extendable there was decided to formalise the composition of `Transport` and `Accommodation` into `Trips` by interfacing into `TransportOption` and `AccommodationOption`, as is shown in listing 1. This is just a rough sketch, a detailed version can be found in appendix D.

```
1 class Trip {
2     private accommodationOptions: AccommodationOptions;
3     private transport: TransportOption;
4     ...
5     constructor(accOptions: AccommodationOptions, transpOption: TransportOption);
6     public getAccommodationOptions(): AccommodationOptions;
7     public getTransport(): TransportOption;
8     ...
9 }
```

Listing 1: The `Trip` class

Those `Trips` have to be provided by a `TripProvider`. This `TripProvider` consists of a `TransportProvider` and an `AccommodationProvider`. The meaning of those two providers will be explained later in this chapter. As can be seen, to create a trip we have to get `TransportOptions` and `AccommodationOptions`, so let us examine how these sections of the system work.

## 5.2. Transport

In the former version of the system, the transport was only provided by SkyPicker with little room for extensibility. To improve this the logical choice is to increase abstraction in the system by programming against interfaces. The following interfaces will be used: the `TransportProvider` is the way into the transport module, it provides an interface to the constructor of a `TransportProvider` and can create the actual `TransportOptions`. This interface only contains getters to get the search parameters from the user, plus a `Promise` of `TransportOptions`. `TransportOption` is the interface that enforces the data that we need to display a journey, so a destination when we leave, when we arrive and also when we return. This interface was built with the data types that SkyPicker returns in mind, to make sure that the API will fit optimally in the system. A `TransportOption` may consist of multiple parts because of flight transfers and return flights. All these interfaces are currently only implemented by the SkyPicker implementation but can be easily extended by implementing them for a different partner that provides transport. All of these implementation will be bundled in a `TransportProviderConstructors` list, to make it easier to instantiate them all. Some of these interfaces have later been split because of the SIG feedback as can be seen in 7.2 with their exact final implementation shown in Appendix D. The Transport structure can be also be found in figure 5.1.

```
1 interface TransportProvider {
2     getTheme(): Theme;
3     getMinPrice(): number;
4     getMaxPrice(): number;
5     getAirportCodeFrom(): string;
6     getPassengers(): number;
7     getTransport(): Promise<TransportOptions>;
8 }

1 interface TransportOption {
2     getId(): string;
3     getCityFrom(): string;
4     getCityTo(): string;
5     getCountryFrom(): string;
6     getCountryTo(): string;
7     getAirportCodeFrom(): string;
8     getAirportCodeTo(): string;
9     getMapIdTo(): string;
10    getPrice(): number;
11    ...
12    getTransportOptionParts(): TransportOptionParts;
13 }

1 interface TransportPart {
2     getCityFrom(): string;
3     getCityTo(): string;
4     getAirportCodeFrom(): string;
5     getAirportCodeTo(): string;
6     getLatitudeFrom(): number;
7     getLatitudeTo(): number;
8     getLongitudeFrom(): number;
9     getLongitudeTo(): number;
10    getDepartureTime(): Moment;
11    getArrivalTime(): Moment;
12    getReturnBoolean(): boolean;
13 }
```

Listing 2: The original version of the important interfaces in Transport

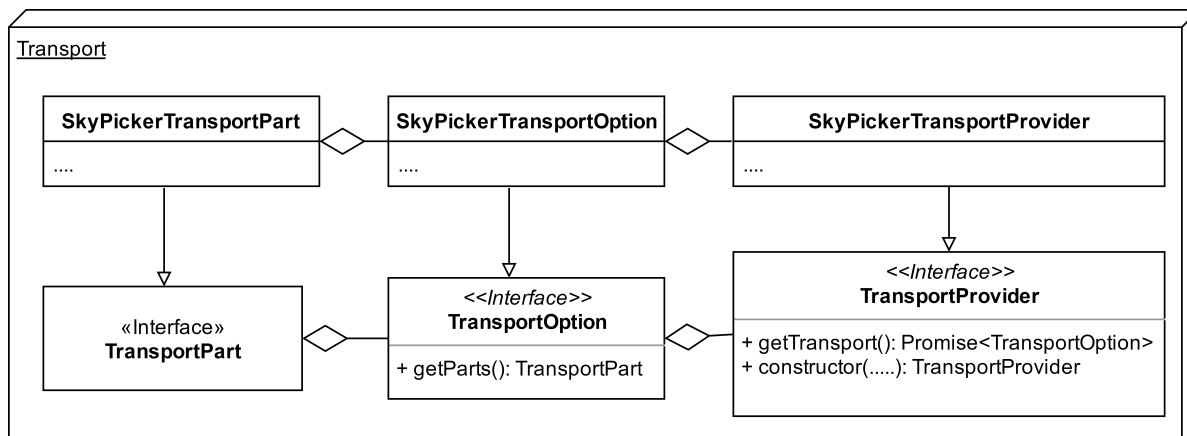


Figure 5.1: A rough overview of the Transport section

To get `TransportOptions`, at least one `TransportProvider` out of the `TransportProviderConstructor` list needs to be instantiated, in our case this means building a `SkyPickerTransportProvider` after this is done correctly we can request `TransportOptions`, these are returned as a promise because you need to wait for the Provider to provide the actual data. When the `TransportOption` is received an `AccommodationProvider` can be instantiated. To see how this works, let us take a close look at the Accommodation section of our system.

### 5.3. Accommodation

The structure of the Accommodation section is very similar to the structure of Transport because this makes this section easily extendable. Interfaces are used again to provide this extendability and `HostelWorld`-specific classes are currently implementing these interfaces, just like `SkyPicker`-specific classes are implementing the Transport interfaces. One of the noticeable exceptions is that the `AccommodationProvider` needs a `TransportOption` to be created, this `TransportOption` contains almost all the important information needed to fetch the possible accommodations. Also note that the way the `AccommodationProvider` provides us with information is through `getTrip()`, which promises us a completely instantiated `Trip` object. In the initial design this was not the case and `AccommodationProvider` would simply return `AccommodationOptions`, but after careful consideration, it was decided this was way easier to implement and would reduce the amount of logic needed in the system. The accommodation section design consisted of the interfaces shown in listing 3. All these interfaces are implemented only in corresponding `HostelWorld` classes.

```

1 interface AccomodationOption {
2     getName(): string;
3     getPrice(): number;
4     getRating(): number;
5     getImageURL(): string;
6     getLatitude(): number;
7     getLongitude(): number;
8     getId(): number;
9     getRoomTypes(): RoomType[];
10 }

1 interface AccommodationProvider {
2     getTransportOption(): TransportOption;
3     getAmountOfPersons(): number;
4     getTrip(): Promise<Trip>;
5 }
  
```

Listing 3: The original version of the important interfaces in Accommodation

## 5.4. Themes

As you may have noticed, `TransportProvider` makes use of a `Theme`. These Themes are used by `TransportProvider` to restrict the resulting trips and destinations, by giving a set of cities that apply to that specific theme. In the design, there is one interface for all the themes. This way it is easy to add, change or remove a theme in the future. These themes are given to the transport provider, which will then search for `TransportOptions` to the cities in the given theme. The cities also become an object. A city will have a country and an airport code. All the cities are combined in one big object, so that they can easily be found with the city name as their key. This is modelled by making a theme provide a list of cities for which it is needed to search `TransportOptions`. Theme implementations adhere to the following interface (listing 4).

```
1 interface Theme {  
2     getCityCodes(): string[];  
3 }
```

Listing 4: Theme interface

# 6

## Implementation

After creating the design discussed in chapter 5, it had to be implemented. No matter how hard and long the design is being thought over, there are always issues that arise during implementation that need to be addressed by changing parts of the design. This chapter will discuss the implementation phase during which the design was implemented, as well as the arisen issues and their solutions. It will be divided into the back-end implementation and the front-end implementation.

### 6.1. Back-end Implementation

Unlike the design, this section will start with the `Transport` and `Accommodation` implementations and only after that the two will be combined in a `Trip` object. The reason this choice is made is the fact that we are dealing with the two APIs, for each module one. During implementation, we wanted to be sure the connection to the APIs was working properly and we wanted to be assured that the data we were getting from the APIs was in the format we expected. In hindsight, this was a good choice because we quickly found out the documentation of the APIs was more than once lacking necessary information.

In the implementation, there was chosen to adhere to a set of principles in the hope that this would make development faster and less error prone. The first principle is only setting at construction. This makes it so that our classes don't have state (with the exception of `URLEncoder`) and also prevents the possibility of race conditions and other concurrency-related problems. Secondly, we try to implement if-statements mainly through polymorphism. This makes sure that we don't have classes or objects which have multiple functionalities and should actually be multiple classes. In the beginning it was quite hard for our entire team to adhere to these principles but once you get used to it gives a lot of certainties, you never have to guess in what state an object is, nor whether the object might behave differently in different situations, so in hindsight these principles worked very well during development.

The section will first start with a discussion about the implementation of the transport and accommodation modules. After this, the trip classes are discussed to show how these transport and accommodations are merged into a single object. The different themes and the cities that belong to them are discussed after that, followed by a few sections concerning some additional implementation details of this project.

#### 6.1.1. Transport

The first things which are being implemented are the `TransportProvider` and `TransportOption` classes. Since the entire interfaces were already designed previously, the main challenge is implementing a SkyPicker specific `TransportProvider` and `TransportOption`, which were given by us the very logical names `SkyPickerTransportProvider` and `SkyPickerTransportOption`. The SkyPicker API connection in the `SkyPickerTransportProvider` is handled through a helper class, called `SkyPickerURLtoTransports`. This helper class is created to split the multiple responsibilities of the `SkyPickerTransportProvider` over multiple classes, to avoid cluttering in the code. This helper class gets data from the SkyPicker API and uses that to instantiate a `SkyPickerTransportOption` object. This object exists of `SkyPickerTransportParts`, which are used to indicate the different flights that are contained in one `SkyPickerTransportOption`. Examples

of these different flights are the departure and return flight, or the multiple flights that are required due to transfers. Because of the interface structure, multiple implementations of `Transport` can be created in the future. All these implementations are stored in the list `TransportProviderConstructors` so that all of them can easily be instantiated in a batch process.

### 6.1.2. Accommodation

After finishing with the `Transport` section, the `Accommodation` module is created. As with the `Transport` interfaces, the interfaces in this section already exist from the design phase, so the challenge is to create the `HostelWorld` specific implementations. This is structured somewhat similar to the `Transport` module; a `HostelWorldAccommodationProvider` is created that provides `HostelWorldAccommodationOptions`. The `HostelWorldURLtoAccommodationOptions` makes sure to separate the API logic from the provider. There are a few differences however. The initial idea was to get `TransportOptions` from a `TransportProvider` and `AccommodationOptions` from a `AccommodationProvider` and later merge these together in a `Trip` object. However, since the `AccommodationProvider` already needs the data in `TransportOption` for a number of reasons, it becomes apparent it is much easier for the `AccommodationProvider` to return the already merged `Trips`, instead of only the `AccommodationOptions`. For the `AccommodationProviders` it also holds they are bundled in the `AccommodationProviderConstructors` list, to make instantiating all of them easier.

### 6.1.3. Trip

Then the `TripProvider` and corresponding `Trip` classes are created. The `TripProvider` contains providers itself, one for `Transport` and one for `Accommodation`. These will provide the separate `Options` for a `Trip`, which are combined in a `Trip` object. The `TripProvider` has a few other tasks that concern `Trips` in their totality, mainly these tasks come down to filtering functionalities e.g. the total price of a trip has to be in between the minimum and maximum price set by the customer. The `Trip` object contains a single `TransportOption`, combined with multiple `AccommodationOptions` that fit within this `TransportOption`. Furthermore, there is some more auxiliary data in the `Trip` object, mainly containing data about the destination `City`.

### 6.1.4. Theme

As mentioned before, the trips `offers` to its customers are mainly divided into themes, such as a sun theme, a winter theme, a city theme etc. These themes of course need to be represented in the back-end, for which a `Theme` module is created. This module is divided into a single `Theme` interface, which is being implemented by multiple `Theme` classes, one for each theme `offers` its customers. To indicate which destinations belong to which theme, a list of `Cities` per `Theme` is kept in the back-end. If a customer selects a specific theme, only the cities that belong to that specific theme are considered for the resulting `Trips`.

### 6.1.5. City

All the `City` data is currently stored in a static file. The reason for choosing for a static file instead of a database is mainly a legacy one. The company was already used to doing it this way, all data about the cities were already present in a static file. The access to the data in the file is very restricted; only a minimal amount of classes are actively accessing this data. This makes sure the architecture around the data is very modular, which enables the company to eventually switch to a database if desired. We have decided to keep the structure like it is, for now, to not enforce a new technology on the company.

### 6.1.6. Logger

Getting feedback from the system is always required. This way the developers can figure out what is going on inside and what has to change. Especially when being dependent on third parties, the APIs from `SkyPicker` and `HostelWorld` in this case, it is very important to be able to check what is going on where at any point in time. These APIs might not always guarantee the same result, which can make the system behave in unpredictable ways. In front-end implementations, a lot of this feedback would be visual, but in back-end development, we don't have this luxury. Therefore the `winston` `Logger` is used in this project. It is an external package that could be easily obtained from open-source package management systems. This package was used both to display real-time output and to write output to log files so that the content could be studied later.



### 6.1.7. JSON validation

One of the “should have” features as described in section 2.3 is concerned with JSON validation. Since the system is dependent on third-party APIs, it is really useful to have certain assurances about the data you are working with. Since the data the system receives is JSON format, a JSON validator would be of great help. In such a validator, JSON data is put to the test by a JSON schema, a structure that indicates what fields can be expected and of what type these fields may be. The first possibility to finding these schemas is to find them at the third parties. Unfortunately, these parties don't provide JSON schemas, so we created these schemas ourselves. There is a slight problem with this. Since we do not know what is going on in the system at these third parties, we cannot make any guarantees about the correctness of these schemas. What we do know is that the schemas work in the majority of the cases. Also, since the data provided by the third parties do not really change in terms of structure and typing, we are very confident these schemas provide sufficient guarantees concerning the JSON data.

### 6.1.8. Adapter

The first idea was to completely rebuild the front-end, but unfortunately, this was deemed unfeasible during the project (please refer to section 6.2 for more about this). Therefore the old front-end had to be made compatible with the new back-end. The data structure the new back-end provided and the data structure the old front-end expected are not entirely the same, so some middleware has to be created to overcome this gap. For this purpose, the Adapter module is created. Because interfaces allow for extendability, and extendability is one of the core values in this project, of course, the first step is to create an Adapter interface. Now for each version of the front-end, a new class can implement this interface. Currently, there are two such implementations, one is the `IdentityAdapter`, which simply returns the data it gets as input. This will most likely be used for the soon-to-come new front-end. The second one is the `LegacyFrontEndAdapter`, which is used as the middleware between the new back-end and old front-end. This implementation of the Adapter interface transforms data that is returned from the server (more specifically, from `TripProvider`) into data that is accepted into the old front-end. If [REDACTED] later decides the data that is returned from the server is not in the proper format, after all, a new adapter can simply be implemented to overcome this issue.

## 6.2. Front-end Implementation

[REDACTED] was working to get a new design for its web application. The current design was not optimal in terms of user interface and user experience. The initial plan was to rewrite the front-end with this new design in mind. This way, the entire platform could be restructured and all components would be optimally working together. Unfortunately, however, the design was not completely finished on time. This would mean the developer team would simply have to wait for the new design and see an entire week be wasted. Because that is a completely wasteful option, there is decided to leave the front end as it currently is and make sure the current front-end and the new back-end can function together for now. The adapter discussed in section 6.1.8 was used for this purpose.

During the remainder of this project, a first take on the new design was finished and so the [REDACTED] programmers are working on the new design in parallel to this project. Although this does not necessarily influence this project, steps have been taken to take this upcoming change in mind, especially with the design of the adapter.



# 7

## Software Improvement Group

At the end of the sixth week of this project, the code base was sent to the Software Improvement Group (SIG). The code base at that time consisted of the big majority of features that can be found in the final product. A few things were missing at that time. Examples of this are checks for single-day trips, where trips with the same departure date as arrival date were correctly handled and the integration of the front-end code in the server. The state of the front-end when this code was submitted to SIG was minimal, just a bare minimum was implemented in React to account for the new front-end that was supposed to come after. Only after the code was submitted to SIG, we decided to go a different direction with the front-end and create an adapter class to bind the old front-end to the new server (see section 6.2).

"The code of this systems scores 3.5 stars on our maintainability model, which indicates the code is above average maintainable. The highest score wasn't acquired due to lower scores in the Unit Size and Unit Interfacing categories. In the Unit Size category, the percentage of the code of above-average length is studied. Splitting these methods into smaller pieces makes each part easier to understand, easier to test and because of that easier to maintain.... In the Unit Interfacing category, the percentage of code with an above-average number of parameters is studied. In general, an above-average number of parameters indicates a lack of abstraction. Besides that, a large amount of parameters leads to confusion in method calls and to an above-average size of methods more often than not. ..."

---

D. Bijlsma, *Software Improvement Group*

As can be seen in the quote from SIG, the feedback they provided the week after was overall quite positive and they scored the project 3.5 stars, which means it scored "above average on maintainability". There were a few issues that were addressed by SIG as the reason the code did not score the most optimal score. These issues were categorised into two categories: Unit Size and Unit Interfacing. These issues are discussed in the sections below, together with the solutions implemented to fix these issues.

### 7.1. Unit Size

The problem addressed by SIG as "Unit Size" was mainly focused at the length some classes were taking on. Overall we tried to keep our classes and functions as short as possible by giving them a single responsibility and splitting them up if necessary. The issue lied in the classes that contain the different Themes that are used to provide trips to users based on one of the six themes. These themes all contained data as a list of Cities that belong to the themes, besides the logic needed in these classes. Data and logic got entangled in these classes, which is a bad practice.

The solution is simple: split the data and the logic into separate classes. The class lists now have their own files in which for each list of cities a constant is exported containing these cities. These constants are then used in the classes containing the logic.

## 7.2. Unit Interfacing

The second problem addressed by SIG falls in the category “Unit Interfacing”. This issue indicates that some constructors of classes require too many arguments. This would indicate a lack of abstraction as indicated in the SIG feedback. The main reason for this is the type of data we are handling in our server-side code. There is a lot of data-fields coming directly from the APIs and these were used to build objects that represented this data one-to-one, which resulted in a vast amount of arguments in constructors.

One might argue that this means there is no significant loss in abstraction, where SIG claims it normally does indicate a lack of abstraction. But we chose to refactor these parts in the code anyway, also because it increases the readability and therefore maintainability of the code. To give you an example, please refer back to the “TransportPart” interface in listing 2. The interface indicates that a vast number of arguments are required. During refactoring, this interface was split into multiple sub-interfaces, which were all included in the “TransportPart” interface. The new situation can be seen below in listing 5 as well as in appendix D.

```
1 interface TransportPart {
2     getArrival(): TransportPartInfoInterface;
3     getDeparture(): TransportPartInfoInterface;
4     getReturnBoolean(): boolean;
5 }

1 interface TransportPartInfoInterface {
2     getAirportCode(): string;
3     getCity(): string;
4     getPosition(): PositionInterface;
5     getTime(): Moment;
6 }

1 interface PositionInterface {
2     getLatitude(): number;
3     getLongitude(): number;
4 }
```

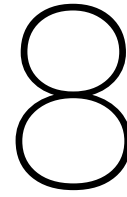
Listing 5: The important interfaces in Transport

During the refactoring of these interfaces, the focus lied on making sure no constructors had more than four parameters, as indicated by SIG, while still making sure the division of the fields over the different interfaces made sense, so grouping elements together in a logical fashion. Only for the `TransportOption` class this was not possible. The current amount of parameters in the constructor is still five because in the team’s opinion this number cannot be reduced without making sure the division of parameters is done in a logical fashion.

## 7.3. Final feedback

The SIG submission is due on the same date as this report will be handed in. Therefore we, unfortunately, don’t know yet what this section will contain. Our wish is that the feedback provided by SIG for this upcoming section will go towards the 5 stars since we have processed all of their feedback in the last few weeks.

Of course, after receiving we will deliberate with [REDACTED] whether anything should be done with the feedback. Factors that will influence this deliberation will be estimated time for implementation and usefulness for the company, among others.



# Evaluation

After implementing and finishing this project, the product will have to be evaluated, in order to determine whether this project was a success and if there are issues to follow up on. In chapter 2 all the requirements and the success criteria were stated. This chapter will discuss whether or not these requirements and success criteria (sections 8.1 and 8.3 respectively) have been met. Also, test code has been developed to support our claims on the functionality of the code, this is evaluated in section 8.2.

## 8.1. Requirements

For most of the requirements, it becomes quickly apparent whether they are completed, especially since we have a user interface on which most of these requirements needed to be present. This section will go shortly over the different categories of requirements as prescribed in chapter 2.

All required features, or “must haves” have been properly implemented in the system. The options menu in the front-end gives the customer the ability to select search parameters, which are sent back to the server to restrict the search results. All these results are sent to the client, shown in a summarising overview, which will redirect the user to a more detailed screen with all the required information for that trip, just as the requirements prescribe it. On this overview, two buttons are present that allow the user to go to the platforms of SkyPicker and HostelWorld to book the flight and hostel. Checking whether all of this information is correct is done through testing, as is explained in section 8.2. The extendability and maintainability requirements that the system must have are a bit harder to evaluate. The SIG feedback was really valuable to check whether the system is maintainable. Since SIG claims to judge maintainability, this feedback is used as the indicator of maintainability. According to the first feedback, the system is above average maintainable which indicates that this requirement has been successfully implemented. All SIG feedback can be found in chapter 7. Unfortunately, there are no clear measures of extendability of the system available for this project, so there is no clear yes or no on the success of this requirement. However, with the way the system is implemented, we strongly believe we have implemented a system that is as extendable as possible. The clear structured interfaces provide a guideline through the system and one-to-one implementations of these interfaces can easily be made to extend the system.

For most of the features the system should have, it is also fairly easy to see they have been successfully implemented. The majority is included in the data that is sent from the server to the front-end and then displayed there. Furthermore, all data is sent in JSON format as indicated in the requirements. All the data the server receives, both from the APIs and the client, are validated with a JSON validator, see section 6.1.7. Since the client has not been implemented by us but was reused from the company (as explained in section 6.2), the requirement which states the client should validate the data it gets from the server has been removed.

Even some of the “could haves” have been implemented in the system. Most of these items that are related to the user interface have been implemented in the old front-end [REDACTED] used and have therefore been reused in the current system. Some of these items have not made it in this project, especially the new design of the application was not implemented in this version of the system.

## 8.2. Testing

For as much of the server code base as possible, test code was developed. These tests help to assure the server is correctly functioning. Of course, we realise that no system can be perfect, so despite the fact that we have a stunning 98% statement coverage, there might be some bugs left. In figure 8.1 the different types of coverage can be seen. Statement coverage was used as a primary indicator of test coverage since it has more semantic meaning than line coverage. Function coverage is not that applicable since we are using TypeScript, an object-oriented superset of JavaScript. The low branch coverage is caused by the transpiling process from TypeScript to JavaScript. This process creates extra branches in the transpiled JavaScript which are very hard to test in the TypeScript code. Therefore the main focus laid on statement coverage, although other types were improved from time to time as well. Please refer to appendix F for a complete overview.

```

===== Coverage summary =====
Statements   : 97.89% ( 743/759 )
Branches    : 70.41% ( 69/98 )
Functions   : 96.72% ( 265/274 )
Lines       : 98.9% ( 720/728 )
=====

```

Figure 8.1: Test Coverage

## 8.3. Success Criteria

To evaluate whether the project was a success, success criteria were defined in chapter 2. The success criteria which were defined are:

- All “must have” requirements have to be implemented.
- 80% of the “should have” requirement have to be implemented.
- The system has to be able to operate in a real-time environment (it should be able to go live).
- The wishes of the Client should be satisfied as much as possible.

The first two criteria have clearly succeeded according to section 8.1. All required features have been implemented. Besides that, 6 out of the 7 “should have” features have been implemented. The seventh of these features is the client-side JSON validation. The reason this wasn’t implemented is the fact the system is reusing the old front-end, so it was not possible to include this. Also with the current view of the company, the old front-end will be replaced as soon as possible, so it was deemed unnecessary to implement this feature. The third criterion has been successfully made. Before writing this report, the server has been successfully tested on multiple machines locally. However this is not yet a complete simulation of a real-time environment, we are very confident it will work without any problems. The old server has already been proven to work in that environment and for this new server, all environments and tools have been kept the same. This would indicate that the system also could go live. After the final SIG feedback has been processed, the system will be implemented live during the handoff of this project to the company. Finally, the last criterion is up to the client:

“The team has been extremely valuable to us. In the starting phase, they were able to think from our point of view. To eventually come up, and implement a back-end system that makes [REDACTED] future proof. Eventually, they guided us through the new system, so that we were able to work with it right away.”

The success of all those criteria is for us reason enough to conclude that this project has been brought to a successful end.

# 9

## Discussion and Recommendations

The back-end is written in a maintainable and extendable way, but now the project is finished, the product has to be handed over to the company. This chapter gives recommendations for the company. It is divided into three categories: back-end recommendations, front-end recommendations and general recommendations. Before that, this chapter will evaluate the process during the project.

### 9.1. Process Evaluation

To make future projects better for ourselves, it is a good thing to evaluate the process during the project. This is a good thing to get a better understanding which things should be handled differently and which things were going well as they were.

This project is organised with the SCRUM methodology and used one-week sprints. The use of this methodology was very good for the process. This allowed the company to gain better insights into what was done and what was going to happen. Another good thing about SCRUM is the flexibility. Sometimes you run into problems and SCRUM allows you to add tasks to fix those problems.

A tool which helped this process was Waffle, an online SCRUM-board. This tool allows you to put all the tasks online available for everyone within the team. Waffle also helps to divide the tasks and can show how far someone is with a task, for example, whether a task has already been started or whether it is already open for review.

At least once every week, there was a meeting with the Client. This was good for them because they could see that something was actually happening and it was also good for us to keep the check if we are on the right track. It was a very good and enlightening experience to get to know how the process is within a start-up. Their programmer was kept closely in the process every step, so when this project is finished, he can take over the system without too many problems.

Also, there were some things in the process which could have gone better. At the beginning, there was made a planning for the project as a whole. This planning took the midterm SIG deadline into account so that we could gather as much feedback as possible. This planning was made very optimistic, because of that deadline. We did expect to have some open features at that deadline, so we also planned for an extension, both to implement the feedback from SIG and the remaining features and improvements. Although it is a drive for you to keep the pressure and try to deliver a lot in a few weeks, it is a bigger demotivator if it turns out it can not be done in time. So being a little optimistic is fine, but it has to stay realistic.

### 9.2. Recommendations for the back-end

Although the back-end is now written in a maintainable and extendable way, there are still some recommendations left. Later on, new features will most likely be added, such as a booking wizard or the original booking system assignment. To keep the code maintainable, first, a design should be made for this new feature by trying to think of all scenarios that could occur. Also, the key features of maintainability and extendability should be kept in mind.

Once the design is ready this feature should be divided into multiple small parts. This keeps a nice overview

of what has to be done. A tool such Waffle is good to use to keep track of what has been done and is still left to do.

During the project, there were some issues with the API calls to HostelWorld. After only three times making an API call, the limit was exceeded. This is not much and when the company really get customers, this will not be enough. Many customers will get no results. A good solution for this would be the use of caching. When they cache the results they only have to make an API call once in a while.

Finally, the cities that are part of a theme could be better specified. A lot of cities are missing in the themes. To be able to offer the customer better trips, this should be taken into account.

### **9.3. Recommendations for the front-end**

For the front-end, there are some recommendations as well. In this project, the front end is not rewritten in a maintainable and extendable way. So a recommendation would be to rewrite the front-end. There are a lot of remarks on the front end in its current state. A lot of processes which are not programmed in an efficient way. But also the whole set up of all the methods through the different files is not optimal. A better structure needs to happen if the Client wants the entire system to be maintainable. The best way to tackle this problem is to make a design before implementing it. This offers great insights in how things are going to be connected. When you encounter a problem, you can change the design, instead of having to rewrite entire chunks of code. And when you start writing code, make sure to keep it clearly divided into small pieces. In our opinion, ReactJS is a good start on this point. Since it is a component-based JavaScript extension, it will help a great deal with keeping the unit size small.

### **9.4. General Recommendations**

Finally, there are also some recommendations for the company itself. It would be a good idea to hire a designer. This is not only for the new design they are currently making but also for new features in the future. These new features have to be put into the existing design. Make sure to keep talking to people about the design and the feel of the system. Users are the key component in every web application, so make it as simple as possible for them.

Another recommendation would be to do more with marketing. This includes using Facebook more active, spread flyers for example on the university and maybe start advertising on another media. This kind of publicity can quickly generate more customers, which is, of course, good for business.



# 10

## Conclusion

The project is done for the start-up [REDACTED]. Their web application was hacked together and so it was not maintainable nor extendable. This is not ideal for the future vision the company has; they want to expand by including multiple partners in their product. Our task was to improve this maintainability and extendability by rewriting the code behind the web application. The resulting back-end code is maintainable and extendable.

The system is also well tested, which gives a high certainty that the system is doing what it is supposed to do. In addition, the incoming data is always validated. This validation assures the rest of the system that the data format is always correct. This makes sure that there will be no unexpected errors due to lack of data or a different form of data. This means the project has been a success both in the main goal of maintainability and extendability, but also in the safety of the program.

Now the company is able to expand their product by including new partners much easier. Even when they use new companies for their data, it is not that much work anymore, in comparison with the old back-end. They have gained a lot of insights because of our work.

This Bachelor Project was not the most challenging for our skill-set, for two of the team members already had experience in the web development field. For the other one, it was very educational though. The reason we were not able to find a more challenging assignment was the last minute switch that was necessary because of our initial project, unfortunately, was not feasible. But nevertheless, we believe we have gotten the most out of this project as we could. The two with the experience could focus on bringing more depth and even more quality in the system, while the one with less experience has gained a lot of new knowledge and experience in this field. For all of us, it was a good experience working with a start-up. A common thing in Bachelor Projects is that you have to create features inside an already existing system. This project gave us the opportunity to rebuild an entire system instead, which gave the opportunity to redesign everything. A difficult aspect was that the system already existed, so you did not want to stick too much to that old system and start from scratch. The old code might give you bad ideas for an implementation.

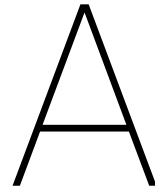
We as a group are very happy with the final result, the code is small, maintainable, extendable, well tested and provides a stable basis for the future of [REDACTED]. The engineers at [REDACTED] know enough about the code to be able to adequately develop further on it. Communication, as well as the division of labour in our group, have gone well and all of this has led to a satisfactory final product.

In short, [REDACTED] can now deploy their web application more efficient and can ask their client: "What trip are you looking for?".



# **Appendices**





# Framework Research

This document contains the results of the Frameworks research. While researching, the results of other researches (that were performed within the same sprint) were taken into account.

The result of the IDE research was the most important one for this research. Since we're using WebStorm, any framework that we use should be easily integrated with the IDE.

## A.1. Javascript Framework Suggestions

This section lists the most important options for JavaScript framework suggestions which are considered for usage during this project. Following these options, a recommendation will be made in section A.2

### A.1.1. jQuery – 'Write less, do more.'

<https://jquery.com/>

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

- + Easily used by millions of people
- + Just JavaScript, only complicated actions (like AJAX calls or attaching handlers) have been abstracted
- + We already know how to use it

### A.1.2. AngularJS – 'Superheroic JavaScript MVW Framework'

<https://angularjs.org/>

HTML is great for declaring static documents, but it falters when we try to use it for declaring dynamic views in web-applications. AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.

- + Easy to use
- Is an entire MVC framework, while we only need a controller framework

### A.1.3. React.js – 'A JavaScript library for building user interfaces'

<http://facebook.github.io/react>

React abstracts away the DOM from you, giving a simpler programming model and better performance. Since React makes no assumptions about the rest of your technology stack, it's easy to try it out on a small feature in an existing project.

- + Virtual DOM
- + Performance
- Is a bit overkill since we do not focus on designing a UI

## A.2. Javascript Framework Recommendation

jQuery, as our project, does not focus on designing a UI and we just need a framework that makes common actions like AJAX calls easier to perform. Another great advantage is that we have all used it before in the course Web- and Database AngularJS is more powerful than jQuery, but there is no need for us to transform the entire DOM while we have only very little HTML. The same goes for React.js.

## A.3. Type Annotation Framework Suggestions

This section lists the most important options for JavaScript type annotation framework suggestions which are considered for usage during this project. Following these options, a recommendation will be made in section A.4

### A.3.1. Flow – ‘A static type checker for JavaScript’

<http://flowtype.org/>

Flow is a static type checker, designed to quickly find errors in JavaScript applications. Typed JavaScript code with Flow annotations easily transforms down to regular JavaScript, so it runs anywhere.

- + Relies heavily on type inference
- + Because of that, easily mix dynamic code with static code
- + Can catch errors related to null
- Does not work on Windows

### A.3.2. TypeScript – ‘JavaScript that scales’

<https://www.typescriptlang.org/>

TypeScript is a typed superscript of JavaScript that compiles to plain JavaScript.

- + Types are optional, but can be used for static checking
- + Support for type-checking virtually any JavaScript library on the web
- Building options are not very versatile (only compile to the same directory or link everything in one file)

### A.3.3. Babel TypeCheck

<https://github.com/codemix/babel-plugin-typecheck>

This is a Babel plugin for static and runtime type checking using flow type annotations. This plugin converts javascript with type annotations to plain javascript with run-time type checks.

- + Run-time support for types
- + Supports primitive types, like int8 or uint64 or the like
- Less support for integration

## A.4. Type Annotation Framework Recommendation

TypeScript, as this framework has the best support in WebStorm. Flow seems just a little bit better than TypeScript, but as all of our developers work with Windows, this is unfortunately not an option.

## A.5. Test Runner Framework Suggestions

This section lists the most important options for JavaScript test runner framework suggestions which are considered for usage during this project. Following these options, a recommendation will be made in section A.6

### A.5.1. Karma – ‘A simple tool that allows executing JavaScript in multiple real browsers.’

<https://github.com/karma-runner/karma>

Karma is not a testing framework, nor an assertion library. Karma just launches an HTTP server and generates the test runner HTML file you probably already know from your favourite testing framework.

- + Supports any testing framework
- + Supports running on any browser (though we only use Chrome)
- + Runs the test on every save

### A.5.2. Mocha – ‘simple, flexible, fun’

<https://mochajs.org/>

Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting while mapping uncaught exceptions to the correct test cases.

- + Includes a testing framework for unit and end-to-end testing
- + Requires little configuration; automatically looks for tests inside a test/ folder
- Little integration in WebStorm

### A.5.3. Wallaby.js – ‘wallaby.js runs your code as you write it’

<https://wallabyjs.com/>

Wallaby.js is an intelligent and super fast test runner for JavaScript that continuously runs your tests. It reports code coverage and other results directly to your code editor, immediately as you change your code. The tool provides a huge productivity boost whether you are doing TDD/BDD or using any other approach.

- + Insanely fast, because it only executes tests affected by your code changes and runs your tests in parallel. + Used by many huge companies
- Insanely expensive (\$100 for a personal licence)

## A.6. Test Runner Framework Recommendation

Karma, as we decided we want a good integration in WebStorm. Mocha provides a lot of extra functionality for testing, but we can also provide these with other frameworks. Wallaby has a really good reputation, the only thing that makes it unusable to us students is the expensive licence.

## A.7. JavaScript Unit Test Framework Suggestions

This section lists the most important options for JavaScript unit test framework suggestions which are considered for usage during this project. Following these options, a recommendation will be made in section A.8

### A.7.1. QUnit – ‘A JavaScript Unit Testing Framework’

<http://qunitjs.com/>

QUnit is a powerful, easy-to-use JavaScript unit testing framework. It’s used by the jQuery, jQuery UI and jQuery Mobile projects and is capable of testing any generic JavaScript code, including itself!

- + Based on the CommonJS Unit Testing standard
- + Simple
- A bit too simple
- Hardly anybody uses this, i.e. little support online

### A.7.2. Jasmine – ‘A JavaScript Testing Framework’

<http://jasmine.github.io/>

Jasmine is a behavior-driven development framework for testing your JavaScript code. Its syntax looks a lot like the Ruby on Rails testing framework and is behaviour-driven.

- + It does not depend on any other JavaScript frameworks
- + It does not require a DOM
- + Behaviour-driven, but can also be used for test-driven (which are nearly the same thing)
- + Most people use this, i.e. more support online
- Less support for virtual server calls (but it is possible)

### A.7.3. Mocha – ‘simple, flexible, fun’

<https://mochajs.org/>

Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting while mapping uncaught exceptions to the correct test cases.

- + Works on unit level and on browser level
- + Support for ‘before’ and ‘after’ functions, like in JUnit for Java

- Little integration in WebStorm

## A.8. JavaScript Unit Test Framework Recommendation

Jasmine, as we really want the WebStorm integration. QUnit is too unused and too simple for our taste, it doesn't meet our needs. Jasmine and Mocha are mostly alike in functionality, but as we already disliked Mocha as test runner for the little integration in WebStorm, Jasmine wins this one.

## A.9. JavaScript End-to-End Test Framework Suggestions

This section lists the most important options for JavaScript end-to-end test framework suggestions which are considered for usage during this project. Following these options, a recommendation will be made in section A.10

### A.9.1. Protractor – ‘end to end testing for AngularJS’

<http://angular.github.io/protractor>

Protractor is an end-to-end test framework for AngularJS applications. Protractor runs tests against your application running in a real browser, interacting with it as a user would.

- + Easy configuration
- + Automatically waits/sleeps during tests
- Heavily depends on the AngularJS framework, while we are not using that.
- Uses a Selenium server, which is an additional dependency

### A.9.2. Nightwatch.js – ‘Browser automated testing is done easily.’

<http://nightwatchjs.org/>

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 Selenium WebDriver API to perform commands and assertions on DOM elements.

- + Clean syntax
- + Easy to use by providing CSS selectors and BDD
- Uses a Selenium server, which is an additional dependency

### A.9.3. WebDriverIO – ‘Selenium 2.0 bindings for NodeJS’

<http://webdriver.io/>

WebdriverIO lets you control a browser or a mobile application with just a few lines of code. Your test code will look simple, concise and easy to read. The integrated test runner allows you to write asynchronous commands in a synchronous way so that you don't need to care about how to propagate a Promise to avoid racing conditions.

- + Very easy to set up and use
- + Can work with the most used Unit Test frameworks, including Jasmine
- ± Works with Travis, though a bit cumbersome
- Uses a Selenium server, which is an additional dependency

## A.10. JavaScript End-to-End Test Framework Recommendation

WebDriverIO. Protractor looked really nice at first, except we don't use AngularJS. Nightwatch and WebDriverIO both look very promising, but they also share a pitfall, which is having to use an additional dependency, a Selenium server. The nice thing about WebDriverIO is that it can be used with our current Unit test framework, only this time the behaviour driven design part can be used, and this is the reason why we chose it in the end.



# B

## Research IDE

This document contains the results of the IDE (Integrated Development Environment) research. While researching, the results of other researches (that were performed within the same sprint) were taken into account. The result of the framework research was the most important one for this research. Since we're using TypeScript, it would be nice to have this as integration (autocomplete, compile, run, debug) into the IDE. Also, the test runner (Karma) should be easily runnable from the IDE without much effort.

### **B.1. IDE Suggestions**

This section lists the most important options for IDE suggestions which are considered for usage during this project. Following these options, a recommendation will be made in section B.2

#### **B.1.1. Atom – ‘A hackable text editor for the 21st Century’**

<https://atom.io/>

Atom is a text editor that's modern, approachable, yet hackable to the core—a tool you can customise to do anything but also use productively without ever touching a config file.

- + Highly customizable
- + Clear interface
- No debugger or test runner

#### **B.1.2. WebStorm – ‘The smartest JavaScript IDE’**

<https://www.jetbrains.com/webstorm>

Lightweight yet powerful IDE, perfectly equipped for complex client-side development and server-side development with Node.js, also available as general version called IntelliJ.

- + TypeScript integration
- + Test runner Karma integration
- + Actually integrates with virtually everything
- High PC resource utilization

#### **B.1.3. Sublime Text**

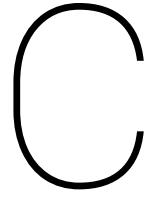
<http://www.sublimetext.com/>

Sublime Text is a sophisticated text editor for code, markup and prose. You'll love the slick user interface, extraordinary features and amazing performance.

- + Works very smoothly
- + Multi-position editing
- No debugger or test runner

## **B.2. IDE Recommendation**

WebStorm, as our project, will use many frameworks (as specified in the other research), and an integration of all these frameworks into the IDE will significantly ease the development process. Even though it has a high resource utilisation, this will only cause a small slowdown compared to the speed-up by all the tools that WebStorm has to offer.



# Research Static Analysis Tools (SAT)

This document contains the results of the SAT research. While researching, the results of other researches (that were performed within the same sprint) were taken into account. The following results were the most important ones for this research:

- For the framework research: we are using TypeScript, so it would be nice to have static analysis tools that can be used for it.
- For the IDE research: we're using WebStorm, so the ability of integration in WebStorm is a must.

## C.1. Suggestions (“major” tools for static analysis)

This section lists the most important options for “major” static analysis tools which are considered for usage during this project. Following these options, a recommendation will be made in section C.2

### C.1.1. TSLint - ‘An extensible linter for the TypeScript language’

(integration with Webstorm)

<https://palantir.github.io/tslint/>, <https://github.com/palantir/tslint>

Description: TSLint is a SAT that enforces a set of rules during software development. These rules are there mainly for assuring a consistent style throughout the development. Therefore TSLint is quite comparable with the Eclipse plug-in ‘Checkstyle’, which is used throughout software development in Java for assuring a consistent programming style. For the full set of rules, please check the link above.

+ Specially designed for TypeScript

### C.1.2. JSHint – ‘A Javascript code quality tool’

(integration with Webstorm)

<http://jshint.com/>, <https://github.com/jshint/jshint>

Description: JSHint is a very well-known SAT for Javascript software development. JSHint finds bugs and potential problems via a set of rules (just like TSLint, but keep in mind that TSLint is there for TypeScript and JSHint is there for JavaScript in general). JSHint is seen as the ‘more flexible’ SAT version of JSLint, containing a new load of rules and removed antagonistic rules (rules that can conflict with each other).

+ Larger set of rules and it's used a lot

- Maybe ‘too’ general for the project, when compared with TSLint

### C.1.3. JSLint – ‘The Javascript code quality tool’

(integration with Webstorm)

<http://jshint.com/>, <https://github.com/douglascrockford/JS�int>

Description: JSLint is also a very well-known SAT for Javascript software development. It's quite similar to JSHint, but more or less the ‘predecessor’ (containing antagonistic rules).

- Compared to JSHint, JSLint might not be the ‘best’ SAT to use

### **C.1.4. ESLint – ‘The pluggable linting utility for Javascript and JSX’**

(integration with Webstorm)

<http://eslint.org/> , <https://github.com/eslint/eslint>

Description: ESLint is the even more ‘flexible’ version of JSHint and provides relatively even more options for its rules. Besides, ESLint is a relatively new SAT.

- + Compared to JSLint, ESLint might be a ‘better’ SAT to use (together with JSHint)
- + Compared to JSHint, even more options for the rules, for more flexibility

## **C.2. Recommendation ‘major’ tools**

TSLint, as our project will be developed in TypeScript, it is useful to use a SAT that focuses its metrics for this language. The alternatives that are proposed above are used for development in Javascript in general and thus could have rules that might not be useful for the language we’re using.

## **C.3. “minor” tool suggestions for static analysis**

This section lists the most important options for “minor” static analysis tools which are considered for usage during this project. Following these options, a recommendation will be made in section C.4

### **C.3.1. Dependo**

<https://kenneth.io/blog/2013/04/01/visualize-your-javascript-dependencies-with-dependo/>,  
<https://github.com/auchenberg/dependo>

Description: Dependo visualizes internal dependencies by a directed graph.

- Dependo is still in progress, according to the notes
- + It’s simple
- + Produces a graph that’s interactive

### **C.3.2. MaDGe (Module Dependency Graph)**

<https://github.com/pahen/madge>

Description: Very similar to dependo; it also visualises dependencies. This is useful for finding circular dependencies within the code.

- + Compared to Dependo, it a more advanced and documented tool for the visualisation.
- It requires another tool: GraphViz

## **C.4. Recommendation ‘minor’ tools**

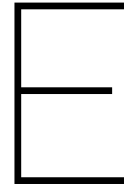
MaDGe, as the tool, is relatively more advanced and we can use the visualisation throughout the project for taking internal dependencies into account.

# D

## UML

Due to confidentiality, the content of this appendix is removed.





# Original Project Description

This original project description was taken from BEPsys and is included as an appendix for completeness considering our initial project at [REDACTED]

For our website, we need to build a booking system where our users can book and pay for a flight ticket and hostel on the same page. At the moment we are redirecting our customers to the website of our flight ticket partner and to our Hostel partner to finish the booking on their website.

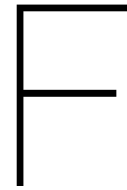
We want to handle the booking and payment on our own website. Our partners allow us to do this through special API links which we can incorporate in our website with our own layout. We need a group of students who would like to analyse and solve this problem. The booking process must be safe, secure and easy for our customers.

---

[REDACTED], BEPsys







# Coverage

File ▲		Statements ⇅	
Accommodation/HostelWorld		100%	40/40
Accommodation/HostelWorld/Option		100%	53/53
Accommodation/HostelWorld/Provider		100%	63/63
Accommodation/Option		100%	32/32
Accommodation/Provider		100%	4/4
Adapter		100%	40/40
City		100%	32/32
Exception		78.95%	60/76
Logs		100%	10/10
Misc		100%	39/39
Theme		100%	66/66
Theme/Cities		100%	15/15
Transport/Option		100%	32/32
Transport/Option/Part		100%	2/2
Transport/Provider		100%	24/24
Transport/SkyPicker/Option		100%	61/61
Transport/SkyPicker/Option/Part		100%	38/38
Transport/SkyPicker/Provider		100%	72/72
Trip		100%	56/56
resources		100%	4/4

Figure F.1: Statement Coverage per Module

Note that the statement coverage of Exception cannot be brought higher than 78.95%, because the Typescript code is transpiled to JavaScript and some extra safety measures are needed for exceptions in JavaScript which are not reachable from Typescript.



# Infosheet

## What trip are you looking for?

*Company:* ██████████

*Presentation date:* July 5th 2017

*Company description:* ██████████ is a start-up offering complete budget trips. It combines budget flights with the cheapest accommodations. The target group consists of people who don't know their destination yet but they often know what kind of trip they like so we offer trips per theme such as sun, surf, city, party, and ski.

*Problem description:* The application was built with just a conceptual phase in mind and so was finished as quickly as possible. To make sure that ██████████ can expand its product and services, the code needs to be built with an eye on maintainability and extendability. Our goal was to rewrite the server of the application in such a way these aspects are properly satisfied.

*Challenges:* The main challenge is managing the maintainability and extendability with all of the features working as in the existing concept.

*Research:* The possibilities within the APIs which were used needed to be researched, as were the ways we could make this code extendable and maintainable, how to validate the responses, how to communicate with the client and which tools to use.

*Process:* Our first step was to design the application with maintainability and extendability in mind after this we needed to make sure everything worked so some minor changes had to be made along the way.

*Product:* The final product is maintainable and verifies every response it gets, it has 97.7% statement coverage through our test suite. Also, the SIG feedback was really valuable.

*Outlook:* ██████████ will continue with this server. we have discussed and explained a lot of our choices to the main programmer so that he is aware of how the system works. At the moment the old client side is used but this can also be more efficient in combination with our server which is what we have recommended them.

## Project Team

Our project team consists of 3 computer science students: Martin Koster, with interests in testing and software engineering who mainly worked on the Research, Testing and the back-end development with little prior experience. Mathias Meuleman with interests in web and server-side development who worked on the continuous integration stack, the front-end and schema validation who has experience with web development through his work with many start-ups and Youri Arkesteijn who's interested in developing maintainable code with main contributions in the software architecture, back-end development and integration and extendibility management who has 7 years of experience working with web technologies for multiple companies.

## Client - ██████████

██████████ is one of the founders of ██████████ and is personally invested in making ██████████ a great platform. His background is in project management. ██████████

## TU Coach - Distributed Systems

D.H.J. Epema is a full professor in the Distributed Systems group of the TU Delft. and besides that our coach and safety net in case we ran into problems in our project. D.H.J.Epema@tudelft.nl

## Contact

*Martin Koster*, trombomart@gmail.com

*Mathias Meuleman*, meuleman.mathias@gmail.com

*Youri Arkesteijn*, youri.essa@gmail.com

The final report for this project can be found at: <http://repository.tudelft.nl>



# Bibliography

- [1] Behshid Behkamal, Mohsen Kahani, and Mohammad Kazem Akbari. Customizing iso 9126 quality model for evaluation of b2b applications. *Information and software technology*, 51(3):599–609, 2009.
- [2] Contel Bradford. 5 common encryption algorithms and the unbreakables of the future, 2014. URL <https://www.storagecraft.com/blog/author/contel-bradford/page/25/>.
- [3] Mozilla Network Developer. Websockets, 2017. URL <https://developer.mozilla.org/nl/docs/WebSockets>.
- [4] Michael Droettboom et al. Understanding json schema. Available on: <http://spacetelescope.github.io/understanding-jsonschema/UnderstandingJSONSchema.pdf> (accessed on 14 April 2014), 2015.
- [5] Ian Fette. The websocket protocol. 2011.
- [6] David Flanagan. *JavaScript: the definitive guide*. " O'Reilly Media, Inc.", 2006.
- [7] Evan Hahn. Understanding express.js, 2014. URL <https://evanhahn.com/understanding-express/>.
- [8] Donna L Hoffman, Thomas P Novak, and Marcos Peralta. Building consumer trust online. *Communications of the ACM*, 42(4):80–85, 1999.
- [9] Madan Prabhu D. Hota, A.K. Node js. 2014.
- [10] ISO. Iso/iec 25010. URL <http://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&limitstart=0>.
- [11] Josef Kepesi. Skypicker.com api. URL <http://docs.skypickerpublicapi.apiary.io/#reference>.
- [12] Alex MacCaw. *JavaScript Web Applications*. " O'Reilly Media, Inc.", 2011.
- [13] Dynamic Systems Development Method. Dsdm atern handbook, 2008. URL <https://www.agilebusiness.org/content/moscow-prioritisation-0>.
- [14] Margaret Rouse. Pci dss (payment card industry data security standard), 2009. URL <http://searchfinancialsecurity.techtarget.com/definition/PCI-DSS-Payment-Card-Industry-Data-Security-Standard>.
- [15] Sai Srinivas Sriparasa. *JavaScript and JSON Essentials*. Packt Publishing Ltd, 2013.
- [16] Qing Yang, Chuan Pang, Liu Liu, David C Yen, and J Michael Tarn. Exploring consumer perceived risk and trust for online payments: An empirical study in china's younger generation. *Computers in Human Behavior*, 50:9–24, 2015.