

Bachelor MKI
„Distributed Systems“,
WS 2018/2019
Prof. Dr. Natividad Martínez Madrid

- Project Documentation -

LotusGarden

by:

Alina Elena Aldea-Ionescu (310194),
Philipp Schiweck (752778),
Matthias Schüle in (751450),
Joffrey Schneider (762380)

4. Semester

Contact addresses:

alina_elena.eldea-ionescu@student.reutlingen-university.de,
philipp_marius.schiweck@student.reutlingen-university.de,
matthias_ulrich.schuelein@student.reutlingen-university.de,
joffrey.schneider@student.reutlingen-university.de

Submitted on:
21.01.2019

Table of Contents

1 Introduction	3
2 Goals	3
3 Requirements	3
3.1. Functional Requirements	3
3.1.1. Must Have Requirements	3
3.1.2 Optional Requirements	6
3.2. Non-Functional Requirements	6
3.2.1. Must Have Requirements	6
4 Conceptual model	6
4.1 App Component Structure	6
4.2 User Interface	8
5 Design Decisions & Implementation	8
5.1 Used Technologies	8
5.2 User Interface Design	9
5.3 Functionalities and general Implementation	10
5.3.1 Add reservation	10
5.3.2 Display available tables	11
5.3.3 Confirmation by email	12
5.3.4 Reservations for the restaurant	12
6 Results	14
6.1 Evaluation of Requirements	14
6.2 Future Improvements	15
7 Conclusion	16
8 Distribution of Tasks	16
9 Sources	17
10 Declaration on oath	18

1 Introduction

The spread of technology in everyday life has shifted the paradigm in the way we manage and plan our daily activities. Consumer preferences continue to evolve, and everyone is seeking convenience when it comes to eating out and making a reservation in a restaurant. This is why we decided to create a web page with an integrated online booking system in order to enable clients to book a table in an easy, fast and comfortable manner.

This paper will walk you through a conceptual model, the creation process of the said application, technologies used and concludes with future improvements.

2 Goals

The aim of our project is to create a web application for a restaurant, with an online booking system for the reservations. The chief goal of this app is to make reservations more comfortable and less complicated for both sides.

It should allow the user to enter the website, look for a desired date and time and visually choose a table available at that particular moment. If the user finds a suitable table according to the needs, s/he can proceed with the reservation by giving few personal details (like name, phone number and email) and finalize the reservation. An email will then be sent to the customer with a confirmation. If the customer wants to cancel the reservation later, s/he can do so by simply clicking on the link provided in the confirmation email.

Even though there are similar applications and systems on the market which serve roughly the same purpose, our main goal was to deepen our understanding in technologies described later in this paper.

3 Requirements

This chapter proposes to present the functional and non-functional requirements of the application, together with the level of importance, where 5 means the highest priority.

3.1. Functional Requirements

3.1.1. Must Have Requirements

Requirement	Date and Time selection in form
Number	1

Description	A date and time can be selected at which the user wants to reserve a table.
Rationale	The user wants to reserve the table at a certain date and time.
Success criteria	The correct date and time is selected.
Level of importance	5 High, this is a basic requirement.

Requirement	Tables with different amount of seats, depending on table
Number	2
Description	The displayed tables should reflect the amount of seats available at said table.
Rationale	The user can directly see how many seats are at the table.
Success criteria	The tables are being displayed correctly.
Level of importance	5 High, this is a basic requirement.

Requirement	Table visualization for user, showing free and already reserved tables
Number	3
Description	The tables immediately show the user which tables are free and which are not. Tables that fulfill the users seat requirements are highlighted.
Rationale	The user can easily choose which table he wants to reserve.
Success criteria	Already reserved tables are displayed a certain way, free tables are displayed differently.
Level of importance	5 High, this is a basic requirement.

Requirement	Selectable tables
Number	4
Description	The user should be able to select the desired table by clicking on it and selecting it.
Rationale	Easy navigation, simple to use.
Success criteria	Tables are clickable, clicked table is selected and highlighted.
Level of importance	5 High, this is a basic requirement.

Requirement	Form to enter name, email and phone number of user
Number	5
Description	The user has to enter Name, Email and phone number in order to be identified when making reservations.
Rationale	The restaurant has to know who reserved the table.
Success criteria	Name, Email and phone number can be entered in the form.
Level of	5 High, this is a basic requirement.

importance	
------------	--

Requirement	Backend server handling reservations
Number	6
Description	The backend server sends the already reserved tables to the frontend, and stores new reservations in the database.
Rationale	-
Success criteria	Tables are correctly displayed to the user and reservations are correctly stored.
Level of importance	5 High, this is a basic requirement.

Requirement	Database storing reservations
Number	7
Description	The database stores reservations with the respective table and customer details.
Rationale	-
Success criteria	Reservations are stored correctly.
Level of importance	5 High, this is a basic requirement.

Requirement	One form to reserve a table, user does not have to go through multiple pages
Number	8
Description	The form is not complicated, everything is visible on the same page.
Rationale	-
Success criteria	-
Level of importance	4 high, this is a basic requirement

Requirement	Email confirmation upon reservation
Number	9
Description	The user receives an email of confirmation after finishing the reservation, containing the date and time, together with a token that can be inputted later on the website in order to cancel the reservation.
Rationale	The customer is assured that the confirmation was placed successfully. Customers also have the possibility to call off a reservation if they cannot honour it.
Success criteria	An email is sent correctly to the user.
Level of importance	5 High, this is a basic requirement.

3.1.2 Optional Requirements

Requirement	Login for Restaurant, displaying reserved tables
Number	10
Description	Seperate page for restaurant owner, displaying the reservations for a day selected on the page. Reservations will be displayed in a timeline showing every table.
Rationale	Owner needs to see which table is reserved.
Success criteria	-
Level of importance	5 High, this is a basic requirement.

3.2. Non-Functional Requirements

3.2.1. Must Have Requirements

- Easy-to-use interface.
- One form to reserve a table, user does not have to go through multiple pages.
- Good UX: responsive form, regardless of screen size (phone, tablet, desktop).

4 Conceptual model

4.1 App Component Structure

The Application is implemented with a REST API written with Express, that communicates between the Frontend, the webpage of the restaurant, and the backend, which in our case is a MongoDB database managed with Mongoose. The Predefined routes are as follows:

Description	Method	URL
Adds a new reservation to the database and sends an E-Mail to the customer	POST	/add
Cancels the reservation with the specific token	GET	/cancel/:token
Returns the tables that are available at a specific time	POST	/tables
Method for the restaurant to get all reservations for a specific day	POST	/restaurant/all_reservations

With these routes it's possible to communicate and send data between the frontend and the backend.

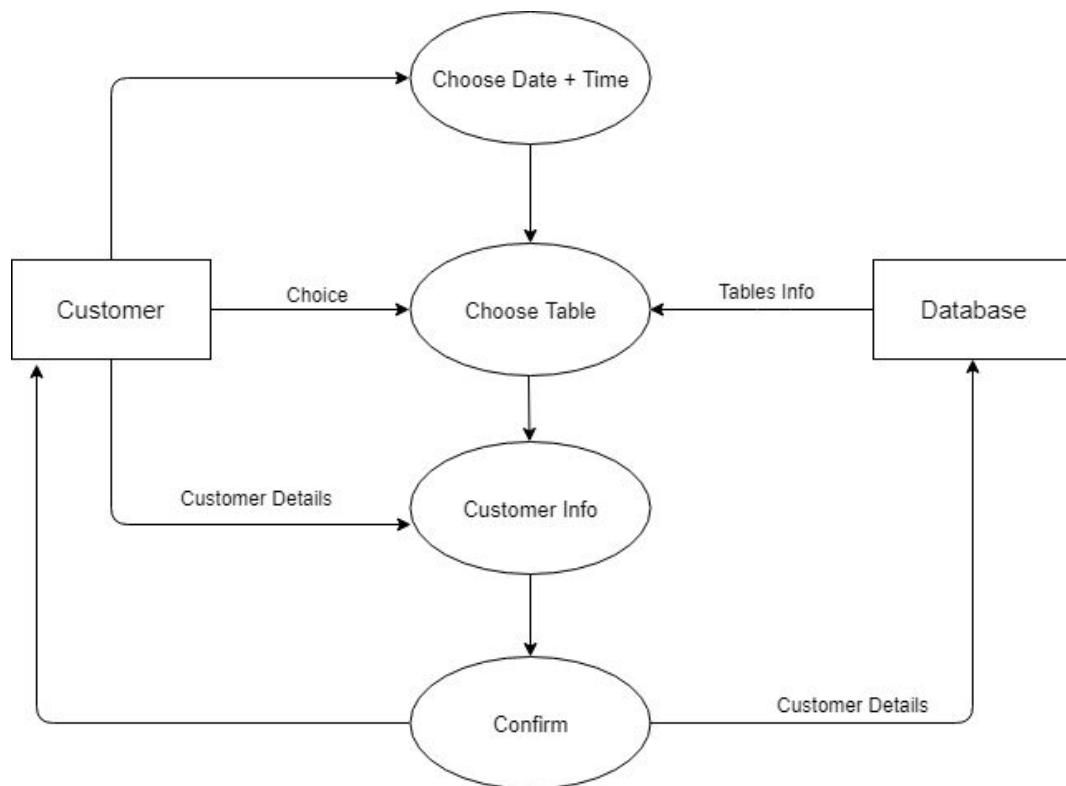


Figure 1. General working flow

The general working flow is shown in Figure 1. The first thing that a user gets is an HTML page where the date and the time for the reservation is chosen. After that the client communicates with the server via the “/tables” route. The frontend then shows the available tables to the user. When clicking on an available table the user can fill in his details and then click on Reserve, which sends the data of the reservation via the “/add” route to the server where it gets stored in the database. The details needed are:

- Name
- Mail
- Phone Number

The backend also generates a random unique token for every reservation. This token is send via mail to the user in a clickable link, so the user can cancel his reservation independently from the restaurant.

Cancel Process

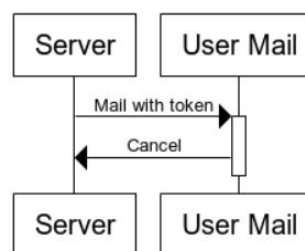
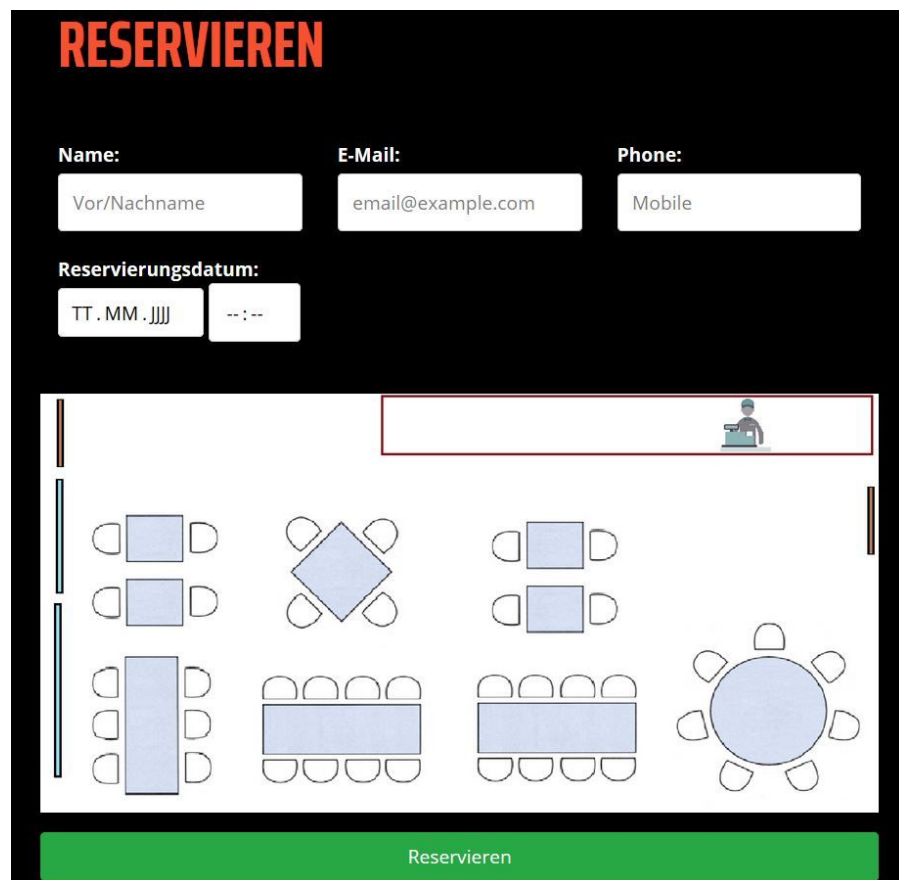


Figure 2. Cancel Process

4.2 User Interface

The user interface should be simple to understand and easy to use. It needs to show the tables of the restaurant and all the input fields that are necessary to get all informations of the customers. The input fields for the date and the time should be restricted so the user only can enter the correct data format. As a main functionality the interface needs to show the user the tables that are available for reservation at a specific time.



The image shows a web form titled "RESERVIEREN" in large orange letters. Below the title, there are three input fields for "Name:", "E-Mail:", and "Phone:". The "Name:" field has a placeholder "Vor/Nachname". The "E-Mail:" field has a placeholder "email@example.com". The "Phone:" field has a placeholder "Mobile". Below these, there is a "Reservierungsdatum:" section with two input fields: "TT.MM.JJJJ" and "--:--". Below the form, there is a large white rectangular area containing a diagram of a restaurant floor plan. The floor plan shows various table shapes: two 2-person square tables, one 4-person diamond table, two 2-person square tables, one 4-person rectangular table, one 6-person rectangular table, one 8-person rectangular table, and one 8-person round table. Each table is surrounded by small circles representing chairs. A small icon of a person at a laptop is visible in the top right corner of the floor plan area. At the bottom of the form, there is a green button labeled "Reservieren".

Figure 3. Prototype of the online reservation system

5 Design Decisions & Implementation

5.1 Used Technologies

HTML:

"Hypertext markup language is a programming language used to create web pages, and is rendered by a web browser. [1]"

Bootstrap:

"Bootstrap is an open source toolkit for developing with HTML, CSS, and JS".[2]"

Javascript:

“JavaScript (JS) is a lightweight, interpreted or JIT compiled programming language with first-class functions.” [3]

NodeJS:

“As an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications.” [4]

MongoDB:

“MongoDB is a cross-platform document-oriented database program.” [5]

Mongoose:

“Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB” [6]

npm:

The **node package manager** “[...] is the world’s largest software registry. Open source developers from every continent use npm to share and borrow packages, and many organizations use npm to manage private development as well.” [7]

5.2 User Interface Design

We tried to design the interface as friendly and quick as possible. When you enter the reservation section of the page, you can select your desired table within just minimum 2 clicks.

For the time selection, we created a horizontal element containing only the possible time slots. This means you can only choose from hours within the opening time of the restaurant. For simplicity, a time slot can only begin in quarter hour steps.

For the table selection, you have a top-view of the restaurant table layout, so you can see at one glance which table suits your needs best. The tables are PNG images, each one placed inside a div. The widths and heights of these divs are set as percentages, so that if you resize the browser, the tables will be resized accordingly without changing the layout or aspect ratio.

After choosing a time slot, the database will be asked about existing reservations around that time. After the answer of the database, the tables become active. This means that they get listeners for mouseenter, mouseleave, and click. They also get titles, according to the availability of that table, so that it will be displayed when you hover the mouse over a table.

A big challenge here was to synchronize the reservations that come from the database with the tables that are shown on the screen without making mistakes with the data structure. It was also important to make sure that it isn't possible to cheat the UI and submit a reservation to an already booked table by for example changing the time after a table was already successfully chosen.

5.3 Functionalities and general Implementation

5.3.1 Add reservation

The most challenging aspects that occurred when posting a reservation to the database lied in the way time and time zones work, and how MongoDB handles them. Because Germany is GMT+1 (local time), everytime a reservation was made, the MongoDB saves it in Zulu time (also known as UTC), so would store it with one hour less. To overcome this, a "Z" concatenated to the date before sending it to the server solved the problem, and all the dates could be eventually stored correctly to appear as current local time.

Moreover, in order to avoid wastage, the customer can only choose a table if the number of persons selected is more than half of the table's capacity.

The data that is being stored in the database has the format from Figure 4. If the selected table is green, it is assumed that the customer spends 2 hours in the restaurant and in this case $end_dateTime = start_dateTime + 2$. If the selected table is yellow, it means that the customer chooses to spend between 1 and 2 hours, depending on the availability shown. In both cases, $start_dateTime$ and $end_dateTime$ are both sent from front end to the server. The date time picker is also configured to display only the current and future days.

```
const reservationSchema = new mongoose.Schema({
  name: String,
  email: String,
  phone_number: Number,
  start_dateTime: Date,
  end_dateTime: Date,
  table_id: Number,
  number_of_people: Number,
  token: String
});
```

Figure 4.. Mongoose Schema

5.3.2 Display available tables



A table can have 4 different states:

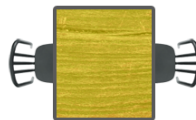
1. *Free*

The table is available at your chosen time for at least a full 120 minutes.



2. *Free for less than 120 minutes*

You're able to book the table, but you have to keep in mind that your reservation is directly followed by another one. The time available will be displayed when hovering over this table.



3. *Free but the number of seats isn't right*

This is the case if you either don't meet the minimum amount of seats to get a bigger table, or you want more seats than the table has to offer. You get a hint when you hover over such a table.



4. *Not available*

There is already another reservation for this table at your chosen time slot.



5.3.3 Confirmation by email

After the reservation has been posted to the database, an email is sent to the customer, using Nodemailer module of NodeJS. The email contains the relevant information of the reservation, as depicted below.

In the email received, the customer also has the possibility to cancel a reservation by simply pressing on the link included.

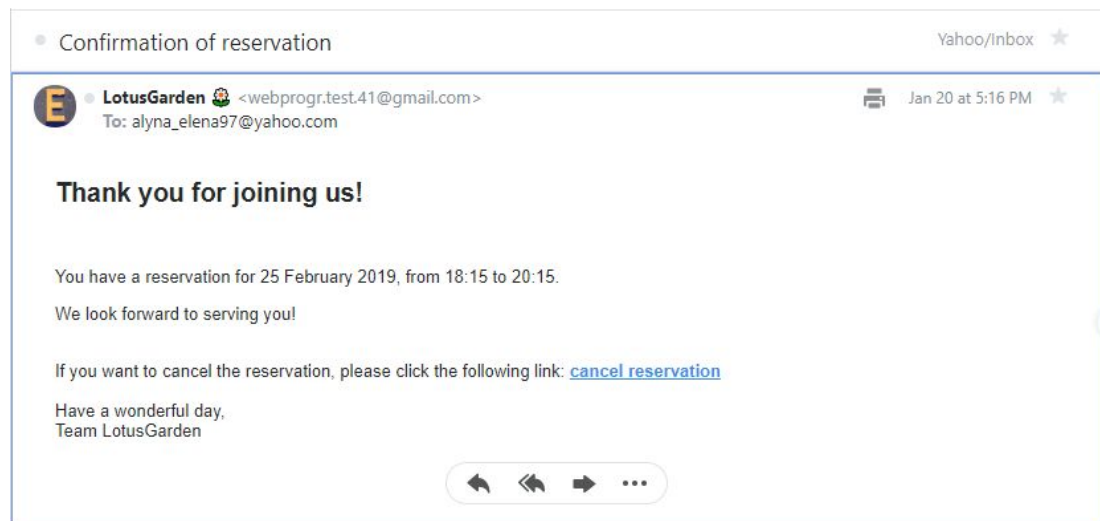


Figure 5. Email confirmation

5.3.4 Reservations for the restaurant

The restaurant has to have the ability to check for the current reservations in order to be able to communicate with the guest in case there is a conflict, as well as to be able to plan ahead in terms of how full the restaurant will be. The solution for this has to be hosted on the same server as the website, so the restaurant can easily access the reservations for the day/the next day without needing to jump through hoops, like accessing a database themselves.

This is why we implemented a separate site for the restaurant to access, where the current reservations for that day can be viewed, sorted by table and ordered by time.

The restaurant owners can access this via the domain of the website /getReservations (for example, localhost:3000/getReservations).

Usually, this should be hidden and only accessible by logging in, but we felt that setting up a login server would go too far for this project.



Figure 6. Restaurant reservation checker

When first accessing `/getReservations`, the restaurant employee is asked to enter a date, for which the reservations should be checked.

When a date is entered and the “Get Reservations” button is pressed, a POST request is sent to the backend. The backend then queries the database for every reservation on the day that was entered.

This data is then sent as JSON to the frontend. Upon receiving this data from the backend, the frontend then sorts the reservations by `table_id` into an array for each of the nine different tables.

After this, the arrays are then used to generate HTML tables that display the reservations, one HTML table per table in the restaurant. In these HTML tables, the data is sorted by time, displaying the earlier reservations first.

The data displayed in the tables is:

- The name of the person placing the reservation
- The number of people
- The start time of the reservation
- The end time of the reservation (standard 2h after the beginning, if the table is available that long, otherwise the end time is the next time the table is reserved)
- The phone number of the person that placed the reservation

This data is set to the innerHTML of the table display div of the `/getReservations` document.

20.01.2019

Table 2

Name	Number of people	Start Time	End Time	Phone
Philipp Schiweck	2	16:30	18:30	12345678

Table 3

Name	Number of people	Start Time	End Time	Phone
Philipp Schiweck	3	13:30	15:30	1234567
Max Mustermann	3	14:30	15:45	123456
Matthias	4	15:45	17:45	123456789
Alina Elena Aldea-Ionescu	3	18:15	20:15	730998047

Table 5

Name	Number of people	Start Time	End Time	Phone
Matthias	1	17:00	19:00	123456789

Table 6

Name	Number of people	Start Time	End Time	Phone
Max Mustermann	3	13:15	15:15	12345678

Table 9

Name	Number of people	Start Time	End Time	Phone
Test McDude	4	13:00	15:00	123456789

Figure 7. Reservations displayed to the restaurant

6 Results

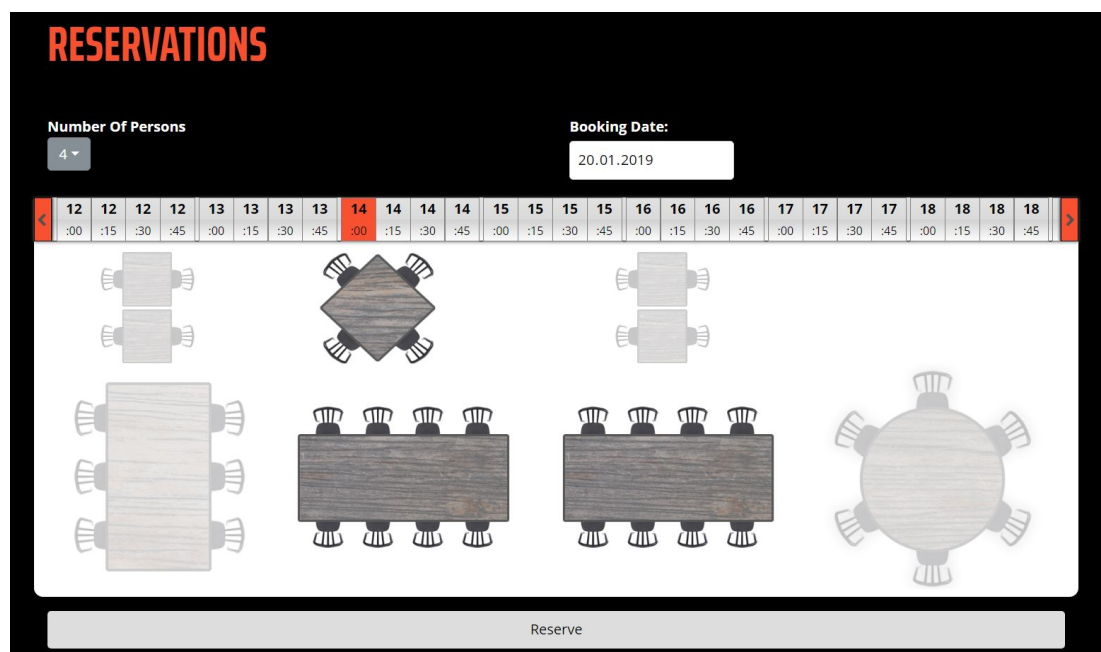


Figure 8. Final Implementation

6.1 Evaluation of Requirements

All must-have functional requirements and partially the optional requirement have been implemented and tested at the end.

Testing

We tested the app by hosting it on a server and giving the link to the server to a few of our friends. The following are some of their opinions of the app (translated to english).

User Test Opinions:

User 1: This application will revolutionize the process of reserving tables in Restaurants! Although the application looks nice on a desktop, it still needs to be improved for mobile use. Most of the users of an application like this will be on a mobile device, which is why this is the most important platform in my opinion.

User 2: I very much dig the layout of the application, it looks very nice. But: the application is very slow, I don't know why, maybe it's the server it's hosted on. *(Authors note: it is actually because of the server, this application was hosted on a free DigitalOcean droplet, we didn't want to pay anything for this demoUser)*

User 3: The app has a very clean design, is nice to look at and feels responsive in use.

Although I don't think I would need this type of app personally, I think it would be useful for a lot of people.

6.2 Future Improvements

If we would keep on going with the development of our application, there are still some features that would be nice to add:

1. As already mentioned in our little test with our friends, the use of the application on mobile devices still needs some improvements. Maybe an extra mode for mobile use where the focus is on the reservation process, with less information text around. Also the application needs to respond to the size of mobile screens for better usability.
2. The Feedback messages in our current implementation are simple Javascript alerts. This could also be implemented with Bootstrap, so it looks better and, again, the overall usability of the application gets better.
3. Our current implementation of the tables is hardcoded, which means that you can't change the number of the tables or seats without changing a few things in the back- and frontend. If we want that this application gets used by more than this one restaurant the implementation needs to be more flexible so you can implement any combinations of tables.
4. In the moment, everyone that knows how to call the "/getReservations" page can access the page where you can see all the reservations. If we would want people to use this application in

public, we would need to make that access more secure. The easiest and most common way would be to implement some sort of restaurant login, so the page can only be accessed with a username and a password.

7 Conclusion

Due to time constraints imposed on this project and the number of students involved, few compromises have been made in the features of the application. As there were countless possibilities of enhancing the app and adding new features, limitations had to be made. The main focus was developing a working web application through its main functionality of searching and booking reservations. Besides the main web application embedded in an HTML page, the rest of the page was static, just developed to demonstrate the general view of real restaurant websites. Further upgrades mentioned in the last chapter remain a possibility for the enhancement of the application.

8 Distribution of Tasks

Task	Responsible
Requirement Analysis	Philipp, Alina
Design	Philipp, Joffrey
Front-End	Philipp, Joffrey
Back-End	Alina
Database Management	Alina, Matthias
Documentation	Everyone

9 Sources

- [1] HTML; <https://html.com/>, 19.01.2019
- [2] Bootstrap; <https://getbootstrap.com/>, 19.01.2019
- [3] Javascript; <https://developer.mozilla.org/bm/docs/Web/JavaScript>,
19.01.2019
- [4] NodeJS; <https://nodejs.org/en/about/>, 19.01.2019
- [5] MongoDB; <https://en.wikipedia.org/wiki/MongoDB>, 19.01.2019
- [6] Mongoose;
[https://medium.freecodecamp.org/introduction-to-mongoose-for-mong
odb-d2a7aa593c57](https://medium.freecodecamp.org/introduction-to-mongoose-for-mongodb-d2a7aa593c57), 19.01.2019
- [7] npm; <https://docs.npmjs.com/about-npm>, 19.01.2019

10 Declaration on oath

We hereby declare that we have written the present work on our own, that we have not used any external help other than declared and that all helping material and sources used for the work have been properly named and referenced.

We are aware that an untruthful declaration will be considered as a deceit.

_____	_____	_____
Place,	Date	Signature (First and last name)

_____	_____	_____
Place,	Date	Signature (First and last name)

_____	_____	_____
Place,	Date	Signature (First and last name)

_____	_____	_____
Place,	Date	Signature (First and last name)