



## Assignment of bachelor's thesis

<b>Title:</b>	Web Application WeFix
<b>Student:</b>	Aydin Misirzade
<b>Supervisor:</b>	Ing. Michal Valenta, Ph.D.
<b>Study program:</b>	Informatics
<b>Branch / specialization:</b>	Web and Software Engineering
<b>Department:</b>	Department of Software Engineering
<b>Validity:</b>	until the end of summer semester 2022/2023

### Instructions

The goal of the thesis is to design and develop a web application that helps to find a service for broken electronics like smartphones, laptops, washing machines, etc.

1. Make a review of similar solutions on the market, discuss their pros and cons.
2. Design and develop a solution that will solve the customer's problem in a user-friendly way. Choose application architecture and implementation platform is part of the thesis.
3. Focus on correct design, documentation, and testing according to standard software engineering approaches.

Starting requirements for the application are:

- application with web-based frontend with responsive design,
- help to find repair services based on their specialization, rating, pricing, or location,
- ability to manage the content (new services, user's rating, registered users, etc.)

Other requirements will be clarified after analyses of similar applications.



Bachelor's thesis

# WEB APPLICATION WEFIX

**Bc. Aydin Misirzade**

Faculty of Information Technology  
Department of Software Engineering  
Supervisor: Ing. Michal Valenta, Ph.D.  
May 17, 2022

Czech Technical University in Prague  
Faculty of Information Technology

© 2022 Bc. Aydin Misirzade. Citation of this thesis.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

Citation of this thesis: Misirzade Aydin. *Web Application WeFix*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

*First of all, I would like to thank my family and friends for their infinite support throughout my studies.*

*I want to thank Mrs. Ludmila Facer, who always helped me with any of my administrative queries. She was always there to help and was worrying for me.*

*I would like to express my immense gratitude and respect to Ing. Jan Trávníček, Ph.D. for teaching me the basics of coding within the BIE-PA1 lessons and Ing. Ladislav Vagner, Ph.D. for his deep expertise in CS theory and, of course, for the almighty Progtest.*

*My special thanks goes to my supervisor Ing. Michal Valenta, for all the support and guidance with the actual thesis.*

## Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 17, 2022

.....

## Abstrakt

Cílem této práce je vyvinout webovou aplikaci určenou k vyhledání opravárenského servisu pro jakýkoli druh elektroniky, domácích spotřebičů, vozidel včetně dalších druhů strojů a popsat celý proces jejího vývoje. Práce také popisuje proces aplikační analýzy se zkoumáním podobných řešení. Konečným výstupem práce je kromě popisu procesu vývoje také funkční webová aplikace.

**Klíčová slova** Webová aplikace, opravy, design, implementace, Javascript, React

## Abstract

The aim of this thesis is to develop a web application designated to find a repairing service for any kind of electronics, household appliances, vehicles, including other kinds of machinery, and to describe the whole process of its development. The thesis also describes the process of application analysis with examination of similar solutions. Besides the description of the development process, the final output of the thesis is also a working web application.

**Keywords** Web Application, repairment, design, implementation, Javascript, React

# Contents

<b>Acknowledgments</b>	<b>iii</b>
<b>Declaration</b>	<b>iv</b>
<b>Abstrakt</b>	<b>v</b>
<b>List of Abbreviations</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Motivation</b>	<b>3</b>
<b>3 Analysis of existing solutions</b>	<b>5</b>
3.1 NejŘemeslníci.cz . . . . .	5
3.2 Opravárna . . . . .	6
3.3 Final conclusion . . . . .	7
<b>4 The Solution</b>	<b>9</b>
4.1 The Idea . . . . .	9
4.1.1 What is MVP? . . . . .	9
4.1.2 Type of Application . . . . .	9
4.2 Requirements . . . . .	10
4.2.1 Functional Requirements . . . . .	10
4.2.2 Non-functional Requirements . . . . .	11
4.3 Design and Architecture . . . . .	11
4.3.1 Application Architecture . . . . .	11
4.3.2 Database Design . . . . .	13
4.4 Technologies . . . . .	13
<b>5 Implementation</b>	<b>17</b>
5.1 API Implementation . . . . .	17
5.1.1 Authentication . . . . .	17
5.2 The back-end side . . . . .	17
5.3 Security . . . . .	18
5.4 Error handling . . . . .	19
5.5 The front-end side . . . . .	19
5.5.1 Tailwind CSS . . . . .	19
5.5.2 DaisyUI . . . . .	20
5.5.3 Custom Theme . . . . .	20
5.5.4 Card-style components . . . . .	21
5.5.5 Grid . . . . .	22
5.6 Google Maps Integration . . . . .	23



<b>6</b>	<b>Testing and Pipelines</b>	<b>25</b>
6.1	Testing . . . . .	25
6.1.1	Usability Testing . . . . .	25
6.1.2	Unit Testing . . . . .	26
6.2	Pipelines . . . . .	26
6.2.1	GitLab CI . . . . .	27
<b>7</b>	<b>Conclusion</b>	<b>29</b>
7.1	Evaluation . . . . .	29
7.2	Drawbacks and Future Improvements . . . . .	30
7.3	Final Thoughts . . . . .	31
	<b>List of Abbreviations</b>	<b>35</b>
	<b>List of Attached Files</b>	<b>39</b>

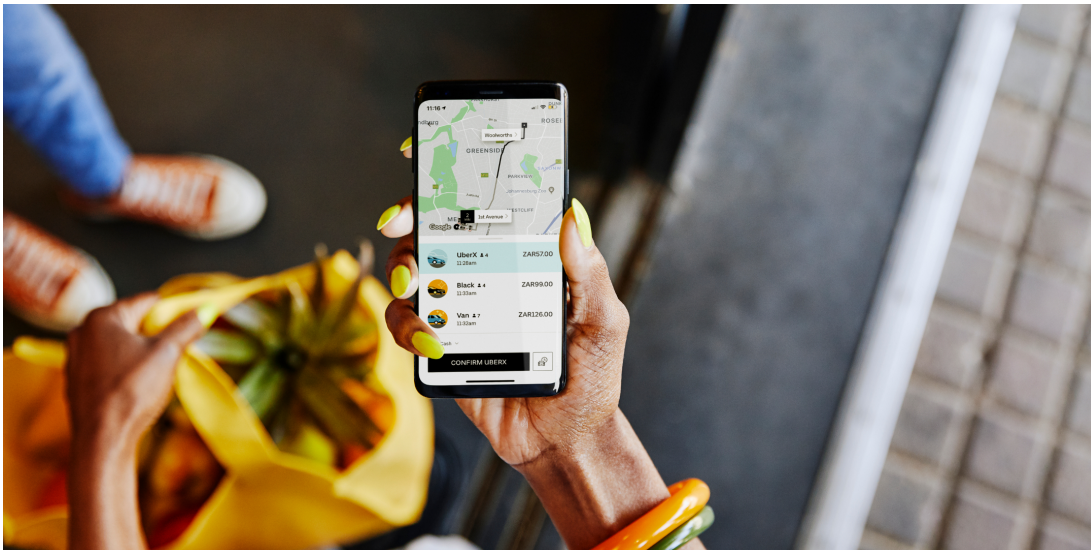
## List of Abbreviations

API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CSS	Cascading Style Sheets
DB	Database
HTML	Hypertext Markup Language
JS	Javascript
JSON	Javascript Object Notation
JWT	JSON Web Token
MVP	Minimum Viable Product
UI	User Interface
UX	User Experience

# Chapter 1

## Introduction

Continuous technological advancements have an impact on every important aspect of our lives, whether it's an indirect effect like the ability to purchase almost every kind of product in our local stores as a result of globalization based on technological progress in logistics, or a more direct effect like vast improvements in medicine or facilitation in human communication all around the world. Even though the humanity has advanced immensely in the latter, it is still possible to find a niche, which has not been filled yet.



■ **Figure 1.1** A person uses his smartphone and Uber application to order a taxi

The idea of this thesis was originally inspired by services such as Uber, Airbnb, Wolt, i.e. online service providers, which make our daily life a little bit more comfortable. Even though the Czech Republic has always been renowned for its advanced competency in IT market and there has been a number of online service providers (Dáme Jidlo, Rohlík.cz) emerging during the last decade, there is one specific field which has not been digitalized yet in the Czech Republic.

This thesis aims to examine and solve a problem of finding a repair service for a broken machinery. It will describe the whole process of development of the minimum viable product, known as MVP, from design to actual implementation, while the practical output of the thesis will be a functional web application which can be used by people.

The thesis is divided into seven chapters. The second chapter provides the motivation and describes the stated problem in more details. Following that, a chapter focused on analysis of the existing solutions on the Czech market is present, outlining their strengths and weaknesses.

Chapter four will discuss the proposed MVP solution. It will focus on the design and the architecture, describe functional and non-functional requirements. Moreover, there will be an overview of the technologies used while implementing the application and reasons behind choosing them.

Chapter five is called Implementation, which consists of description of the actual development process, including back-end and front-end implementation nuances, and describing API implementation and its authorization.

The sixth chapter will give a review of the testing process and CI/CD pipelines. Chapter seven concludes the whole thesis.



from having more people seeing the services they provide, and simple users, or consumers, will have a wide selection of services to choose from. Furthermore, WeFix provides functionality to look for services on a city map, meaning that users can find repairing services nearby, or select the best one based on ratings and reviews left by real people. Those are the features that the Czech market currently lacks, but can be brought in by WeFix.

# Analysis of existing solutions

As mentioned in the previous chapters, the Czech market for online repair services is pretty rare. When you try to find any in the vastness of Czech internet, only two companies catch your eye: NejŘemeslníci.cz and Opravárna. Both of the companies provide assistance in finding a good repair company, but the way they do it is significantly different than the idea of WeFix. In this chapter, I will go deeper into how each of the competitors work, and compare them with WeFix.

## 3.1 NejŘemeslníci.cz

The screenshot shows the 'Nová poptávka' (New order) form on the NejŘemeslníci.cz website. The form is divided into several sections:

- Co potřebujete vyřešit? \*** (What do you need to solve? \*): A dropdown menu with 'Úklid' (Cleaning) selected. A note says 'Povinné položky jsou označeny hvězdičkou.' (Mandatory items are marked with an asterisk).
- Podrobný popis zadání \*** (Detailed description of the order \*): A large text area for describing the job.
- PSČ zakázky \*** (Postcode \*): A text input field.
- Telefon \*** (Phone \*): A text input field. A note says 'Telefon je třeba pro ověření poptávky. Nebude zveřejněn.' (Phone is needed for order verification. It will not be published).
- Chci si vybírat** (I want to choose) and **Chci kontakty hned** (I want contacts now): Radio buttons with 'Více...' (More...) links.
- Jak se poznáme? \*** (How do we know each other? \*): Radio buttons for 'Jsem tu poprvé' (I'm here for the first time), 'Mám tu účet' (I have an account), and 'Přihlásit přes Facebook' (Log in via Facebook).
- Right sidebar:** A vertical list of three steps:
  - 1 Zadáte poptávku** (You will place an order): 'zdarma po celém Česku' (free across the whole Czech Republic). '→ Dostanete až 4 nabídky. Pošleme vám je do mailu, ale najdete je i na stránce své poptávky.' (You will receive up to 4 offers. We will send them to your email, but you will find them on your order page).
  - 2 Vyberete dodavatele** (You will choose a provider): 'podle hodnocení předchozích zákazníků' (according to the ratings of previous customers). '→ Dobré reference = 4x nižší riziko, že se spálíte.' (Good references = 4x lower risk of being scammed).
  - 3 Ohodnotíte práci** (You will rate the work): 'hodnocení přispívá ke kvalitě odvedené práce' (ratings contribute to the quality of the work). '→ Firmám záleží na dobrém hodnocení.' (Companies care about good ratings).

■ **Figure 3.1** An interface for new order at NejŘemeslníci.cz

NejŘemeslníci.cz is a company that has been on the Czech market since 2009, and which has gathered a solid number of partners, such as Velkopopovický Kozel, Stavebniny DEK, and even Saint-Gobain, the French giant, specializing in the construction field.

While analyzing the way NejŘemeslníci works from the user perspective, I found out that, on the high-level, their business follows a very simple process:

1. The user selects the type of work he/she wants to be done.

2. Then, the user fills out the form with all the contact data and the detailed description of the service to be provided
3. After that, the user receives an email from NejŘemeslníci with a number of offers from their partners to choose the service from.

This business process is somewhat similar to that of WeFix, but WeFix does not make users wait for an email with offers and does not make users fill out a detailed description of the work. In other words, WeFix provides one interface for all things and offers its users relevant services based on the user's query. The offering is done based on the location and rating, and it's totally up to the user which company or an individual repairer to select. To put it another way, WeFix does not make individual or tailored offerings, but supplies its users with everything there is to select.

Moreover, the focus of NejŘemeslníci is mainly in the field of construction or in-house works, such as plumbing, cleaning, interior restoration, housekeeping, etc. In contrast to that, WeFix works with a broader range of categories, and is open for partners specializing all types of repair works. NejŘemeslníci is more oriented on business, rather than individual clients, while WeFix can be equally used by both.

Now, let us take a look at the technical part of our analysis. When you type in NejŘemeslníci.cz in your browser, all you get is a plain landing page with one submission form to make an order, and the description of the business and how it works. This contrasts well with WeFix, as it is a fully-fledged Web application with all the common functionalities a user would expect.

In conclusion, after careful examination of NejŘemeslníci.cz, it can be clearly seen that We-Fix is totally a different product. NejŘemeslníci is a well-established business that works in a more traditional way and lacks an online tool similar to WeFix.

## 3.2 Opravárna

The screenshot shows the home page of the 'Opravárna' website. The header includes the logo and navigation links: ČLÁNKY A MÉDIA, KAMENNÁ OPRAVÁRNA, JAK TO FUNGUJE, MOJE OPRAVY, OPRAVME ČESKO, and PRO FIRMY. The main content area is titled 'PŘEHLED OPRAV' and features a '+ NOVÁ POPTÁVKA' button. Below this, there are two sections for order management: 'ZADANÉ ZAKÁZKY (2)' and 'PŘIJATÉ ZAKÁZKY (0)'. The 'ZADANÉ ZAKÁZKY' section contains a table with columns for Datum, Popis, Čas, Platnost, and Stav. Two orders are listed, both with a status of 'ČEKÁ NA PLATBU' and a 'DETAIL OPRAVY' button. The footer includes 'OBCHODNÍ PODMÍNKY', 'KONTAKTY', and a small logo.

Datum	Popis	Čas	Platnost	Stav
22.04.2022 17:37	BMW	Do měsíce	11 dnů	ČEKÁ NA PLATBU
22.04.2022 16:45	iphone	Do měsíce	11 dnů	ČEKÁ NA PLATBU

■ **Figure 3.2** Home page of Opravárna

In contrast with NejŘemeslníci, Opravárna's domain includes not only construction services and household works, but also categories such as PCs and mobile phones, watches, automobiles,



electronics, and many other types. The other important aspect of Opravárna's domain of business which makes it different from NejŘemeslníci, but similar to WeFix, is that Opravárna works both with individual repairers and repair services. As for the user flow, the process is not that different from that of NejŘemeslníci, but has some advantages and additional features over it. The process is:

1. The user logs in, or registers, his/her account in Opravárna.cz.
2. The user selects the type of work he/she wants to be done.
3. Then, the user fills out the form with the detailed description of the service to be provided, or the broken goods.
4. As soon as the user receives submits the form, he/she can see the map with different repairers or services marked on it.
5. The user has to make a payment of 75 Kč in order for Opravárna to send out the user's order to its partner network.
6. The user will get offers from repairers/services to his/her email address, or get his money back

This flow has the same mistakes as the process at NejŘemeslníci. First, Opravárna does not provide one common interface. The user has to make the order on their website, and then proceed to his/her mailbox. Second, they make the user wait for the reply from repairers. Moreover, they make user pay the small fee for the mediation. I will not discuss the pros and cons of such business model, as it is out of the scope of this thesis, but from the user experience point of view, the mentioned process is a poor one.

While Opravárna.cz's platform follows the same "Make an order, and we'll respond" pattern, from technical point of view, they have a working Web application, where a user is able to sign up (log in), select the category of service, and make the order. There is a "My Orders" page, which lists the history of the orders made by user. There is no option to rate a repairer or a service company, though.

Finally, I think Opravárna has a product which can be considered a rival to WeFix. Opravárna.cz lacks some of the key features WeFix has, and operates as a mediator between a client and a repairer. WeFix, on the other hand, is a platform for both actors, and does not get involved into communication between them.

### **3.3** Final conclusion

As a result of my analysis, I can conclude that WeFix has a right to live. It is a platform, which is rather new, different from its potential competitors, and bringing new unique functionality to the Czech market.



# The Solution

## 4.1 The Idea

### 4.1.1 What is MVP?

A minimum viable product (MVP) is a concept from "Lean Startup"<sup>1</sup> that stresses the impact of learning in new product development. Eric Ries defined an MVP as that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort. This validated learning comes in the form of whether your customers will actually purchase your product.

A key premise behind the idea of MVP is that you produce an actual product (which may be no more than a landing page, or a service with an appearance of automation, but which is fully manual behind the scenes) that you can offer to customers and observe their actual behavior with the product or service. Seeing what people actually do with respect to a product is much more reliable than asking people what they would do.

The primary benefit of an MVP is you can gain understanding about your customers' interest in your product without fully developing the product. The sooner you can find out whether your product will appeal to customers, the less effort and expense you spend on a product that will not succeed in the market.

From software engineering perspective, an MVP is a working product, which is able to provide features required for the essential use cases, but not enriched by some additional functionality, or fancy design, and which can be developed by a small team (in our case, by a team of one).

It is important to understand what the term MVP is, because the application discussed in this thesis is a clear example of a minimum viable product.

### 4.1.2 Type of Application

In today's diverse world of software, there are numerous ways to design and implement an application. There are virtually countless possibilities and options to choose from, starting from the choice of the application type, followed up by the choice of environment, programming languages, technologies, and frameworks. This section will discuss these choices. Furthermore, this chapter contains a section about database design.

The first choice that has to be made is what kind of application is best in the circumstances. Considering the domain of the application, potential customers and use cases of the application, there are two main approaches which would be possible – either implementing native applications

---

<sup>1</sup>A book by Eric Ries

both for computers (for the administrators) and smartphones (for the receivers of the messages) or creating a web application – or, of course, combining both approaches.

When making this choice, it has to be acknowledged that recent surveys show that smartphone users are becoming less and less motivated to download extra applications, as not to bloat their own devices. [4][5] The same can be said with desktop applications – installing a desktop application might seem as a kind of commitment for a user. Moreover, developing a single web application with responsive design is far less time demanding than building a native application for every mobile and desktop environment.

So far, a web application seems like a good option. Of course, web applications do have their downsides – it is hard to ensure good user experience for users of all internet browsers and devices of all different sizes, and they tend to be slower and worse adjusted than their native counterparts. However, in this situation, the advantages seem to outweigh the disadvantages, so it was decided to develop a web application, with a possibility to transforming it to Progressive Web App, or developing native mobile applications in the future.

## 4.2 Requirements

Requirements that are well thought through and clearly documented are essential to any successful software engineering project. There are two main different types of system requirements that should be gathered by those working on software projects. System requirements can be categorized as either functional requirements or non functional requirements. Understanding the difference between the two helps to ensure that developers will deliver a product that performs as expected. Research shows that 68% of IT projects fail[6]. One of the main reasons for this is poor definition of requirements at the start.

In this section, two type of system requirements will be presented – functional requirements, summarizing what operations should the user of the application be able to do, and non-functional requirements, summing up everything else about the application but the functionality – for example, reliability or usability of the application.

### 4.2.1 Functional Requirements

The functional requirements are logically divided into groups by their relation to an application user. There are two user types in the application: a general user (or a customer), and a repairer.

**Both user types** are able to:

- Sign up in the system.
- Log in to the system.
- Log out from the system.
- View "My orders" section.
- View a page about a repairing service/repairer.
- View a page about an order.

**A user of type customer** is additionally able to:

- Navigate on a map with repairing services/repairers pinned onto it.
- Leave a rating for the repairing service/repairer.
- See a list of repairing services/repairers.

A **user of type repairer** is additionally able to:

- Accept or decline an order.
- Update the status of order (In progress, completed).

## 4.2.2 Non-functional Requirements

This section will provide a list of non-functional requirements, excluding the choice of programming language, platform, and frameworks – these important aspects will be discussed more thoroughly in the next section.

- **Accessibility** – application is accessible through all standard internet browsers on stable URL to everyone with internet access.
- **Integrability** – application should provide an open API to be easily integrated with other systems.
- **Adaptability** – as should be clear from the previous chapters, the system can be used for a wide variety of use cases, and thus is fairly adaptable.
- **Availability** - application should be available (online) for at least 90% of the time
- **Security** – core functionality is available only to authorized users, moreover divided by user roles as specified above.
- **Usability** – application is easy to use for people of all ages and education levels.
- **Language** - application supports English language only

## 4.3 Design and Architecture

### 4.3.1 Application Architecture

Web application architecture can refer to several design points of the application. In this section, I will provide a brief summary to two of these approaches - one describing options to interconnect the application logic between client and server side [7] and other describing the relationships and interactions between the application inner components, such as database, different point of application logic, etc. [8]

#### Distributing application logic

There are three main options:

##### Server side HTML

Also referred to as a "Legacy HTML Web App Architecture", this architecture is the oldest and most common approach used. It refers to a situation where the client's only job is to send request to the server, which generates a whole page in HTML and sends it back to the client [9]. Due to small interactivity of the app (the whole page has to be reloaded with every request), this approach is nowadays more commonly used for static websites than dynamic web applications [10, 11]. The biggest advantage of this approach is high security [10].

##### JS Widgets

Also known as Widget Web app, this approach divides each of the client's pages into several sections called widgets. Each of these widgets are responsible for loading a different piece of data - and thus are independent. The name JS Widgets is indicating technology used for this type of architecture - which is JavaScript's AJAX, which allows calling HTTP requests from client asynchronously and independently on the actual page load [9]. With this architecture, data can be updated separately for every widget without the need to reload whole page, which provides better experience for data dynamic applications without using the single-page architecture.

## SPA

SPA or Single-page application is the most modern approach. While using this architecture, user only requests the whole page once. After that, reacting on user's actions, parts of the web page are rendered independently - this means not only data on the page, but whole structure can be changed based on received responses from server. The communication is - as with the previous architecture option - based on asynchronous HTTP requests, using AJAX. The biggest advantage of this approach is its interactivity - user can usually interact with parts of the web page while other (for example data-heavy) parts are being rendered [9].

## Chosen architecture

One of the selling features of WeFix is its ease of use. Single-page applications offer a much better user experience, meaning that users can navigate easily between the different pages of an app without waiting for the pages to load. As soon as a user interacts with the app, only the required component will be modified, not the complete application, which makes a single-page app much faster in terms of interactivity. [12] Another point in favor of SPA is that single-page applications are easier to develop, thus making single-page applications the best option for an MVP of WeFix

## Relationship of components

There are three main approaches:

### Monolithic architecture

Monolithic architecture is the most traditional and oldest approach to software architecture - the application is build as one unified component, where every part of the application runs on a single machine [13]. The application may interact with other services, but the main part of its behaviour runs "within its own process" [14]. The application is usually decomposed into several layers - data access layer, business logic layer and presentation layer in order to achieve at least some level of loose coupling [15] - despite this, the application is still very connected - this brings many disadvantages, such as limiting options for application scaling - the only way to horizontally scale such application is to duplicate the whole application on several machines [13]. However, this architecture is easiest to develop and deploy, which still makes it a viable choice for app development[16].

### SOA

Service-oriented architecture is aiming to resolve the tight coupling of monolithic architecture. The application is usually decoupled into several smaller modules, which then communicate between them- selves [15]. With right implementation, the application is very loosely coupled and offers high reusability of individual services and better maintainability - as the application is

much less coupled together, an error in one module usually does not directly affect another module [16]. However, bigger decoupling of services means that communication between individual parts of the application gets harder - which can lead to errors [15].

### Microservices

Microservices is an architecture which is following the decoupling requirement even more thoroughly – same as service-based architecture, microservices divide the application into several loosely coupled, discrete units. The biggest difference between microservices and service-oriented architecture is the granularity of the individual modules, while some sources even state that microservices architecture just a variant of service-oriented architecture[13].

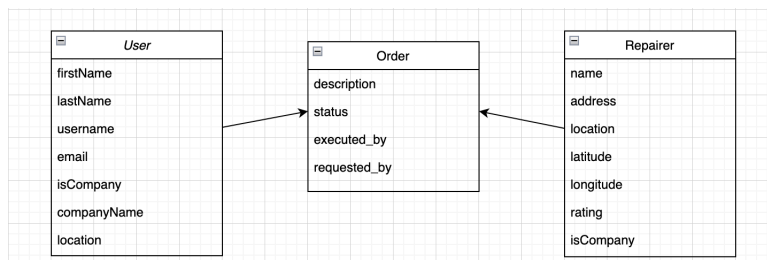
### Chosen architecture

Because I am developing an MVP version of WeFix within this thesis, the back-end of which will not be heavily loaded, it was decided to use monolithic architecture, which is also simplest, and thus the fastest to implement.

## 4.3.2 Database Design

For the MVP, I decided to choose one central database which holds all of the information about orders, repairers and of course all users.

WeFix uses MongoDB as its database platform. We will revise what MongoDB is about in the next section. MongoDB is not a canonical relational database, and does not have tables. It uses a JSON object, which is called a *document*, as an entry in the database, and the documents are grouped into *collections*. There are 3 collections in WeFix’s database: users, orders, repairers. Each of the collection consists of documents which all have common schemas, as depicted in Figure 4.1.



■ **Figure 4.1** Database Design of WeFix MVP

For the relationship between documents, I used the technique called "Embedded documents". To put it simply, it is a JSON object nested in a JSON object. Documents of type *order* have embedded documents for fields *executed\_by* and *requested\_by*.

## 4.4 Technologies

When choosing the production environment and programming language, several factors have to be taken into consideration – firstly, the decision made about the type of the application has to be reflected on – which makes the list of technologies possible to use substantially smaller. Another aspect worth taking into account is popularity of a language and environment, as more popular languages have bigger communities and therefore it is much easier to find solutions to

problems which will undoubtedly occur. Final and the most influencing element, which has to be considered is that it is preferable to use the languages and environment which are usually used in the company I am employed in while developing products, as WeFix will be maintained and modified by other employees of the company. The company is based on Microsoft technologies, which narrowed the final choice of technologies to only a few options. Considering all the previous points, I chose following technologies:

## **Back-end Technologies**

### **JavaScript**

JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles.

### **Express.js**

Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based Web applications. Following are some of the core features of Express framework:

- Allows to set up middlewares to respond to HTTP Requests.
- Defines a routing table which is used to perform different actions based on HTTP Method and URL.
- Allows to dynamically render HTML Pages based on passing arguments to templates.

### **MongoDB**

MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and function which is the equivalent of relational database tables.

#### Why MongoDB?

- Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
- The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clear structure with key-value pairs.
- The rows (or documents as called in MongoDB) does not need to have a schema defined beforehand. Instead, the fields can be created on the fly.
- The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
- Scalability – The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database



## Front-end Technologies

### JavaScript + TypeScript

TypeScript is an object-oriented language designed by Microsoft in 2012. It is a superset of JavaScript and can be used both a language and a set of tools. It provides static typing, which can help to catch errors as they appear making it ideal for large team collaborations. One of the big benefits is to enable IDEs to provide a richer environment for spotting common errors as you type the code. For a large JavaScript project, adopting TypeScript might result in more robust software, while still being deployable where a regular JavaScript application would run.

### React

React.js is an open-source JavaScript library that is used for building user interfaces specifically for single-page applications. It's used for handling the view layer for web and mobile apps. React also allows us to create reusable UI components. React was first created by Jordan Walke, a software engineer working for Facebook. React first deployed on Facebook's newsfeed in 2011 and on Instagram.com in 2012.

React allows developers to create large web applications that can change data, without reloading the page. The main purpose of React is to be fast, scalable, and simple. It works only on user interfaces in the application.

#### Why React?

- **Simplicity** – ReactJS is just simpler to grasp right away. The component-based approach, well-defined lifecycle, and use of just plain JavaScript make React very simple to learn, build a professional web (and mobile applications), and support it. React uses a special syntax called JSX which allows you to mix HTML with JavaScript. This is not a requirement; Developer can still write in plain JavaScript but JSX is much easier to use.
- **Native Approach** – React can be used to create mobile applications (React Native). And React is a diehard fan of reusability, meaning extensive code reusability is supported. So at the same time, we can make IOS, Android and Web applications.
- **Testability** – ReactJS applications are super easy to test. React views can be treated as functions of the state, so we can manipulate with the state we pass to the ReactJS view and take a look at the output and triggered actions, events, functions, etc.
- **Components Support** – The use of HTML tags and JS codes makes it easy to work with a huge dataset containing DOM. React acts as an intermediary that represents the DOM and helps you decide which component requires changes to get accurate results.
- **One-way Data Binding** – One-way data-binding implies that absolutely anyone can trace all the changes that have been made to a segment of the data. This is also one of the reasons that makes React so easy.

## Other used Technologies

### JWT (Json Web Token)

Json Web Token (JWT) is a RFC standard for secure transmission of information as a JSON object. It can be used for two purposes – authorization or information exchange between two parties. The JWT consists of three parts, which are separated by dots: header, payload, and signature. The header typically contains information about the type of token being sent (JWT in our case) and the signing algorithm used. Payload contains the actual data being transmitted

– in case of authorization a set of claims which unambiguously determine the user. Signature is created from the Base64Url encoded and concatenated header and payload. Created token is then sent with HTTP request, typically passed in the Authorization header.[17]

### Why JWT?

Instead of storing information on the server after authentication, JWT creates a JSON web token and encodes, sterilizes, and adds a signature with a secret key that cannot be tampered with. This key is then sent back to the browser. Each time a request is sent, it verifies and sends the response back.

The main difference here is that the user's state is not stored on the server, as the state is instead stored inside the token on the client-side.

JWT also allows us to use the same JSON Web Token in multiple servers that you can run without running into problems where one server has a certain session, and the other server doesn't.

Most modern web applications use JWT for authentication reasons like scalability and mobile device authentication.

### Mongoose

Mongoose is an object document modeling (ODM) layer that sits on top of Node's<sup>2</sup> MongoDB driver. It's similar to an object-relational mapping for a relational database.

### Why Mongoose?

While it's not required to use Mongoose with the MongoDB, there is a bunch of reasons why using Mongoose with MongoDB is generally a good idea.

- MongoDB is a denormalized NoSQL database. This makes it inherently schema-less as documents have varying sets of fields with different data types. While this provides your data model with flexibility as it evolves over time, it can be difficult to cope with coming from a SQL background. Mongoose defines a schema for the data models so the documents follow a specific structure with pre-defined data types.
- Mongoose has built in validation for schema definitions. This saves from writing a bunch of validation code that you have to otherwise write with the MongoDB driver. By simply enforcing in your schema definitions, Mongoose provides out-of-the-box validations for the collections.
- Mongoose abstracts away most of the MongoDB code from the rest of the application.

### Jest

Jest is a JavaScript open-source framework mainly used for testing. Jest is majorly used to work with React-based web applications, and it mostly focuses on simplicity while doing any unit testing. The framework helps developers validate everything built with JavaScript, whether it is browser rendering of web applications or any mobile applications. Along with this, Jest also provides a blended package of a built-in mocking library, an assertion library and a test runner.

---

<sup>2</sup>Node.js. A runtime environment for JavaScript

# Implementation

## 5.1 API Implementation

One of the application requirements was a functioning API, as the application may supposedly be integrated with other applications. So far, only an easy API has been implemented, allowing the user of the API to get all possible data that the current MVP operates on. The API is planned to be extended accordingly to third-party applications needs.

### 5.1.1 Authentication

The hardest part of API implementation was without a doubt creating functional and secure authentication. As there is no way how to keep session between individual API requests, every request has to be authenticated. After some consideration it was decided to use authentication based on Json Web Tokens (JWT) – basic principles of this authentication are explained in chapter 4. To make the implementation easier, library *jsonwebtoken* was used for the elemental functions:

- JWT creation – using function `generateAccessToken`, which returns a signed JWT token with a validity of 1 hour. The duration of an hour was selected as the most optimal time a user might spend in the system.
- JWT validation – using function `authenticateToken`, which extracts the token sent in Authorization header, checks the validity, and blocks the request if the token sent is not valid.

## 5.2 The back-end side

The back-end part of WeFix is fairly simple. The entry point for the whole app is *server.js* file, in which the main connection to the database happens, the app subscribes to the routers, and the middleware for error handling is set up. Controllers, or routers, are the separate modules logically divided by their domains:

- `UserController`
- `OrdersController`
- `RepairersController`
- `MapsController`

While it is obvious what the first 3 routers do, we shall talk about the MapsController in the coming sections. The behavior of all controllers is similar: they listen to a specific endpoint, and when that endpoint is called, they delegate the work to the relevant *service* and send the data returned by that service.

Now, services are the places where all the data processing and business logic processes happen. As with the controllers, services are also divided by the field they are responsible for:

- UserServices
- OrdersServices
- RepairersServices
- GMapsServices

Services use Mongoose to interact with the MongoDB, namely, to extract and update the data inside. UserServices handle the authentication logic described in section 5.1.1, while OrdersServices and RepairersServices both take care of creating, updating, and retrieving lists of orders and repairers, respectively. GMapsServices use interact with Google Maps API to get the needed data.

### 5.3 Security

Security is very important when dealing with real users' data. One of the most important aspects of user security on any platform is the user's password and the way the platform stores it. Storing passwords in clear text is the equivalent of writing them down in a piece of digital paper. If an attacker was to break into the database and steal the passwords table, the attacker could then access each user account. This problem is compounded by the fact that many users re-use or use variations of a single password, potentially allowing the attacker to access other services different from the one being compromised.

A more secure way to store a password is to transform it into data that cannot be converted back to the original password. This mechanism is known as hashing. By dictionary definition, hashing refers to "chopping something into small pieces" to make it look like a "confused mess". That definition closely applies to what hashing represents in computing.

In cryptography, a hash function is a mathematical algorithm that maps data of any size to a bit string of a fixed size. We can refer to the function input as message or simply as input. The fixed-size string function output is known as the hash or the message digest. As stated by OWASP<sup>1</sup>, hash functions used in cryptography have the following key properties:

- It's easy and practical to compute the hash, but "difficult or impossible to re-generate the original input if only the hash value is known."
- It's difficult to create an initial input that would match a specific desired output.

Thus, in contrast to encryption, hashing is a one-way mechanism. The data that is hashed cannot be practically "unhashed".

But hashing plain-text passwords is not enough. Without adding a "salt", passwords are still vulnerable. Password salting is a technique of adding a random sequence of data (approximately 32 characters) to each password and then hashing it. Password hashing means turning your password into a string of random numbers by using a mathematical algorithm.

This protects the password from being reverse-engineered by hackers.

---

<sup>1</sup>Open Web Application Security Project®, <https://owasp.org/>

- If a platform stored your password in plain text, then during a data breach, the hacker could easily access it, steal it, and use it against you.
- If a platform only hashed your password, hackers could still reveal the password by figuring out the encryption key (or hash) used. One of the ways this is achieved is with a rainbow table attack that cracks the hashes.
- If the platform salted and only then hashed the password, they then ensured your password is extra difficult to expose.

In WeFix, I used *bcryptjs* library to address this important nuance. Using *bcrypt*'s *genSaltSync* the salt is generated, which subsequently gets consumed by *hashSync* method along with the password. The resulting output is the hashed password with the generated salt. The *UserService* then saves this secure version of the password to the database.

## 5.4 Error handling

Continuing the topic of the secureness of a web application, it is very important to take care of the error handling. According to OWASP, improper handling of errors can introduce a variety of security problems for a web site. The most common problem is when detailed internal error messages such as stack traces, database dumps, and error codes are displayed to the user (hacker). These messages reveal implementation details that should never be revealed. Such details can provide hackers important clues on potential flaws in the site and such messages are also disturbing to normal users. These errors must be handled according to a well thought out scheme that will provide a meaningful error message to the user, diagnostic information to the site maintainers, and no useful information to an attacker.

Due to the size and the scale of the current MVP, there was no need in a thoroughly-structured error handling. But still, a middleware for error handling was setup in our Express.js server. The middleware intercepts and filters all the caught errors and exceptions, both the server and database errors, and sends a generic "Internal Server Error" message to the client. Only for the cases, when a user attempts to access a restricted resource, the server would respond with an HTTP status 401 and "Unauthorized" message.

## 5.5 The front-end side

The ultimate goal of WeFix is to provide a smooth user experience and an eye-catching user interface. The go-to-market feature of WeFix is the user-friendly and modern front-end, which none of the competitors of WeFix currently provides. By keeping it minimalistic, I tried to make the interface of WeFix easy to navigate in, and will described how I managed to achieve this in the further sections.

### 5.5.1 Tailwind CSS

According to the official documentation, Tailwind CSS is a utility-first CSS framework for rapidly building custom user interfaces. In other words, it is a nice and convenient way to write inline styling and achieve an awesome interface without writing a single line of your own CSS. Tailwind CSS provides a set of utility classes that lets developers work with exactly what they need. This allows to create user interfaces that are more flexible to developers' creativity. Tailwind CSS is also easy to set up with a React application.

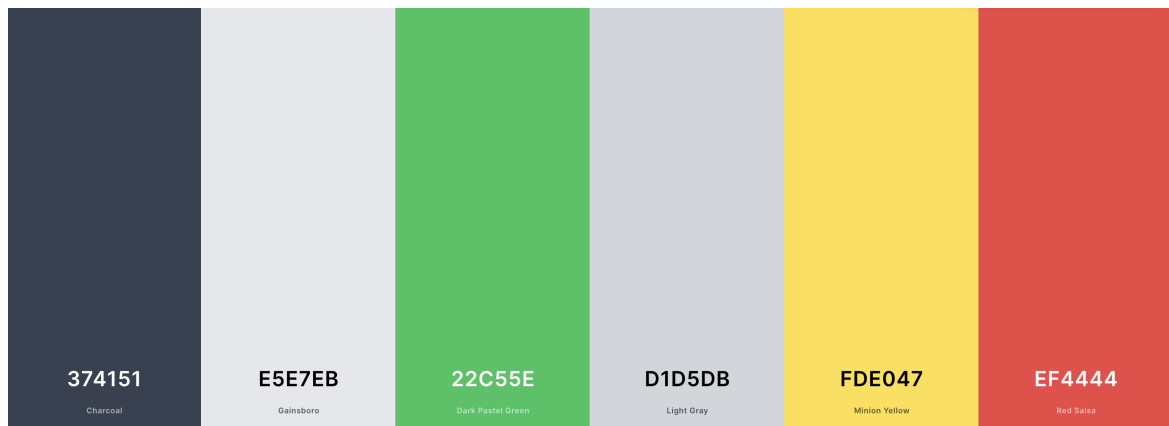
The codebase of WeFix’s front-end inventory is fully written using Tailwind’s classes, and does not contain any `.css` files.

## 5.5.2 DaisyUI

Writing your own UI library is always a great idea, but when the time and resources are not in the developer’s favor, it is worth checking out some lightweight and popular UI kits. The one I selected for WeFix is DaisyUI. DaisyUI is a plugin for TailwindCSS, which boosts the development and makes the HTML code cleaner. It is written in pure CSS, and thus can work with any framework. DaisyUI provides a set of components, such as buttons, toggles, footers, cards and other elements, which can be easily integrated into the code.

## 5.5.3 Custom Theme

DaisyUI provides the theming feature. Each theme defines a set of colors which will be used on all daisyUI elements. In WeFix, I took it even further, and replaced the default theme with my customly designed one. Below is the list of colors used in WeFix theme, their codes, and how



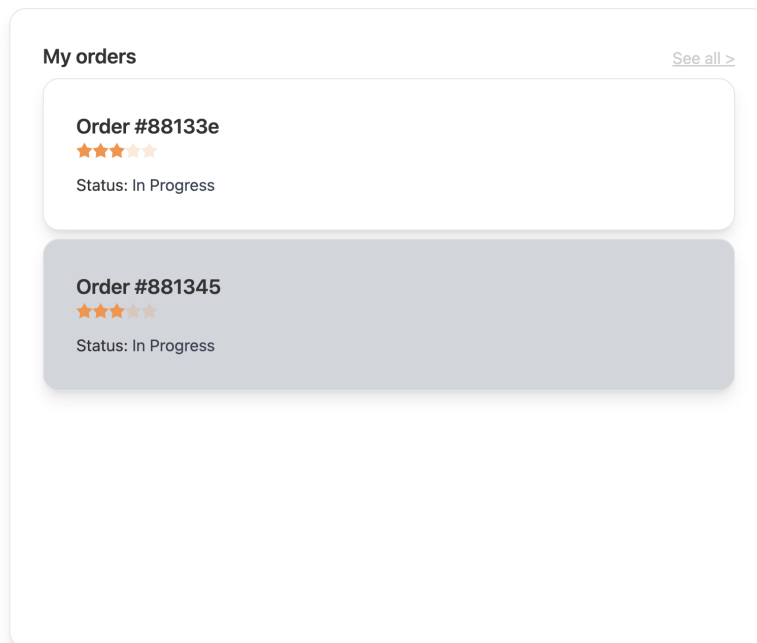
■ **Figure 5.1** WeFix Theme Palette

they are used in the UI.

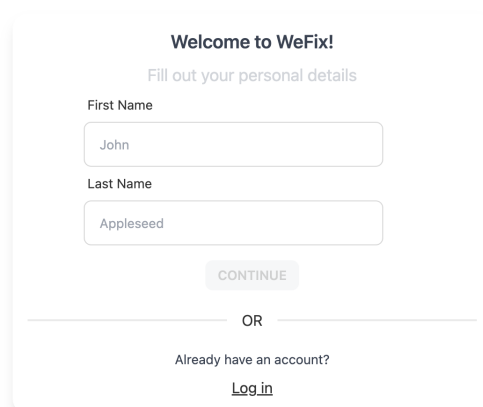
- The "Charcoal" (`#374151`) color is the primary contrasting color. It is used in texts and buttons.
- The "Gainsboro" (`#E5E7EB`) color is the secondary color of the theme, and is used for suggestive texts, like labels
- The "Dark Pastel Green" (`#22C55E`) color is also one of primary colors of WeFix. It is used for success messages, and in the logo.
- The "Light Gray" (`#D1D5DB`) color is a neutral color intended for buttons and hover colors.
- The "Minion Yellow" (`#FDE047`) color is the color for warning messages. Currently, there are none in the app, so this color will not appear, when browsing WeFix. But any theme needs a color of this type, and the "Minion Yellow" will definitely be present in future updates of WeFix
- The "Red Salsa" (`#EF4444`) color is used for important error messages in the interface.

### 5.5.4 Card-style components

To make WeFix even more outstanding and its design more catchy, I decided to build the whole interface of WeFix around the so-called "card" components. Technically, a card is just a centered rectangle with some content inside. I put significant focus on making the cards look more realistic, so I added the three-dimensional effect by rendering shadow along the bottom border line of the cards. This looks fresh, modern, and unique.



■ **Figure 5.2** WeFix "My Orders" widget

The image shows a registration form titled "Welcome to WeFix!". Below the title is the instruction "Fill out your personal details". There are two input fields: "First Name" with the value "John" and "Last Name" with the value "Appleseed". Below the fields is a "CONTINUE" button. At the bottom, there is an "OR" separator, the text "Already have an account?", and a "Log in" link.

■ **Figure 5.3** WeFix Registration Page

## 5.5.5 Grid

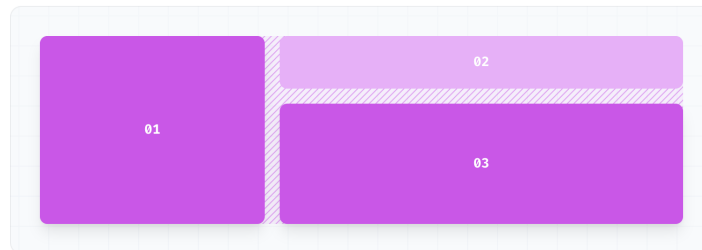
CSS Grid Layout is a CSS layout method designed for the two-dimensional layout of items on a webpage or application. I have been working with the specification over the last five years. On this site is a growing collection of example code, video tutorials and other resources to help you learn the specification.

Grid Layout gives us a method of creating grid structures that are described in CSS and not in HTML. It helps us to create layouts that can be redefined using Media Queries and adapt to different contexts.

Grid Layout lets us properly separate the order of elements in the source from their visual presentation. As a designer this means you are free to change the location of page elements as is best for your layout at different breakpoints and not need to compromise a sensible structured document for your responsive design.

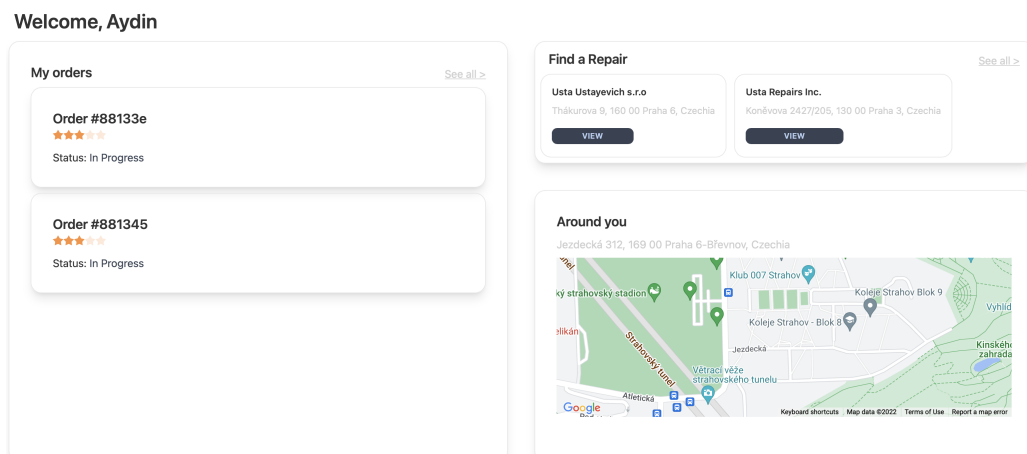
It's very easy to make grid adapt to the available space. With each element having an area on the grid, things are not in risk of overlapping due to text size change, more content than expected or small viewports.

The goal for the homepage was to make it 2x2, i.e. two rows and two columns, but with a little catch. The tricky part was to make the three widgets fit into this layout, namely, somehow handle the way that each widget spans different rows/columns, as depicted in the picture:



■ **Figure 5.4** Desired home page layout

Tailwind CSS provides a native support for grid, so it was easy to plug in the grid layout for the home page. The actual output looks exactly as expected.



■ **Figure 5.5** WeFix Home Page



## 5.6 Google Maps Integration

### Google Maps API

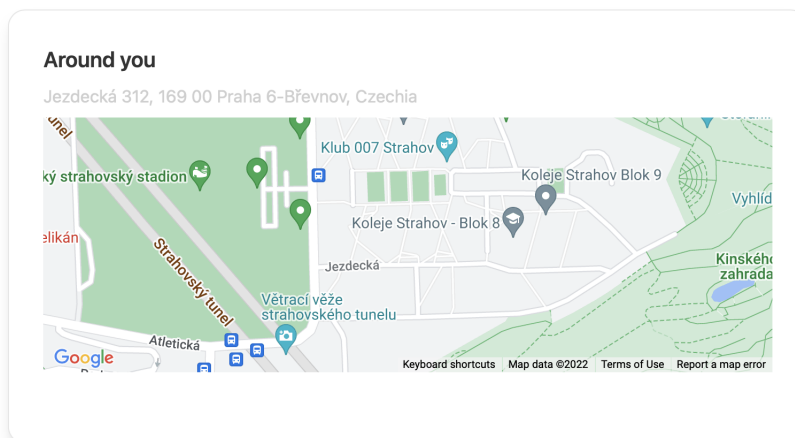
A Google Maps API is an API that allows developers to access Google Maps data and functionality for their own projects. Developers may also incorporate customized Google Maps on their websites or applications using the Google Maps API.

The Google Maps API is actually much broader than only maps and navigation. The specific API I used for WeFix is called Geocoding API. Geocoding API, provided latitude and longitude of a location, returns a detailed information about the place: street name, district name, city, and the country in which the specified place is located. This is called reverse geocoding. And conversely, you may get the exact geometrical information about the place, information such as its latitude and longitude, if you feed the Geocoding API with an address like *Thákurova 9, Prague, Czechia*.

The above mentioned is exactly what is used in WeFix. I will start with the Geocoding API. Currently, our back-end exposes only one endpoint related to Google Maps. As described earlier in this chapter, *MapsController* is the router responsible for this. *GMapsServices*, used by the *MapsController*, has two methods:

- *getLatLng* - takes in the address provided by user, talks to the Geocoding API to retrieve information about the place, parses the response, and sends the latitude and longitude of the location back to the client
- *getUserAddress* - the name of the method speaks for itself. It takes in the latitude and longitude of the user, and returns human-readable address. This service is currently used only to retrieve the current address of the user and properly display the map in "Around You" widget.

But the aforementioned is not the only way WeFix utilizes the Google Maps Platform. By means of the *@react-google-maps* library, written specifically for React apps, I was able to easily embed the map itself into the interface.



■ **Figure 5.6** "Around You" widget



# Testing and Pipelines

## 6.1 Testing

Testing is an integral part of a software development – it aims to find defects of the application, check that the application meets business requirements and can even serve as a mean to find out new requirements. If conducted correctly, it is very powerful tool which can be used to improve the application immensely. In this chapter, testing of the application functionality is described.

### 6.1.1 Usability Testing

As the name suggests, the primary goal of usability testing is to improve the usability of the application. The premise is fairly simple – participants who represent potential users of the product are chosen and are assigned series of the tasks which would be typically carried out in the application. Participants are observed while completing the tasks and are asked about their decisions throughout the process, recording everything any of them says. Next, all of the notes are analyzed and used for creating list of problems which need to be fixed and list of requirements which might serve for implementing new features in the future.

The application was tested on a total of 4 people, with most of them being tested on completing all the chosen tasks. The main purpose was to reveal problems with the application and finding new potential functional requirements or need to modify already existing functional requirements. Main drawbacks pointed out by participants were related to missing features at certain steps or pages, for example:

- Missing hints about password strength and length in the registration step
- Missing section for user profile settings to be able to perform actions such as username change, or password update.
- No option to invite friends.
- No way to modify the layout of the Home Page
- Missing feature to clean order history
- Missing contact details for a repairer

Moreover, participants were asked three simple questions about the application:

1. On a 1-5 scale, how easy is the application to use?

2. On a 1-5 scale, how useful is the application?
3. On a 1-5 scale, would you recommend the application to your friend?

Below, we can see the answers of each of the respondents to the simple questionnaire.

- Person 1: 5, 4, 2
- Person 2: 4, 4, 3
- Person 3: 4, 4, 1
- Person 4: 5, 5, 4

In conclusion, the user testing showed that despite certain shortcomings, the application is fairly easy to use and serves its purpose well. There were fairly low numbers for the 3rd question due to a number of reported missing functionality, which was expected. It is not possible to deduce much because of the small number of responses, however it is fair to assume that the application will follow the same trend as applications do in general – getting more user recognition and custom acquisition as the application grows and improves.

### 6.1.2 Unit Testing

Unit testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit testing should be done during the development of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.

A unit test typically comprises of three stages: plan, cases and scripting and the unit test itself. In the first step, the unit test is prepared and reviewed. The next step is for the test cases and scripts to be made, then the code is tested. Each test case is tested independently in an isolated environment, as to ensure a lack of dependencies in the code. The developer should note criteria to verify each test case, and a testing framework can be used to report any failed tests.

In WeFix, unit tests were written using the Jest framework. The framework allows to easily test React components. Due to lack of a massive business logic on the front-end side, in our unit tests, only things like proper component rendering were tested.

## 6.2 Pipelines

Today's world of software engineering cannot be imagined without the famous terms "Continuous Integration" (CI) and "Continuous Delivery" (CD). CI/CD is the combination of principles, practices, and capabilities that allow for software changes of all kinds to get users in a quick, repeatable, and safe manner. This allows for software developers to integrate their feature or service changes continuously and for IT and operations teams to deliver with the standards, security, and confidence businesses need.

For the sake of this project, I will only focus on the continuous integration, because the continuous delivery is not relevant for the project in the scope of this thesis.

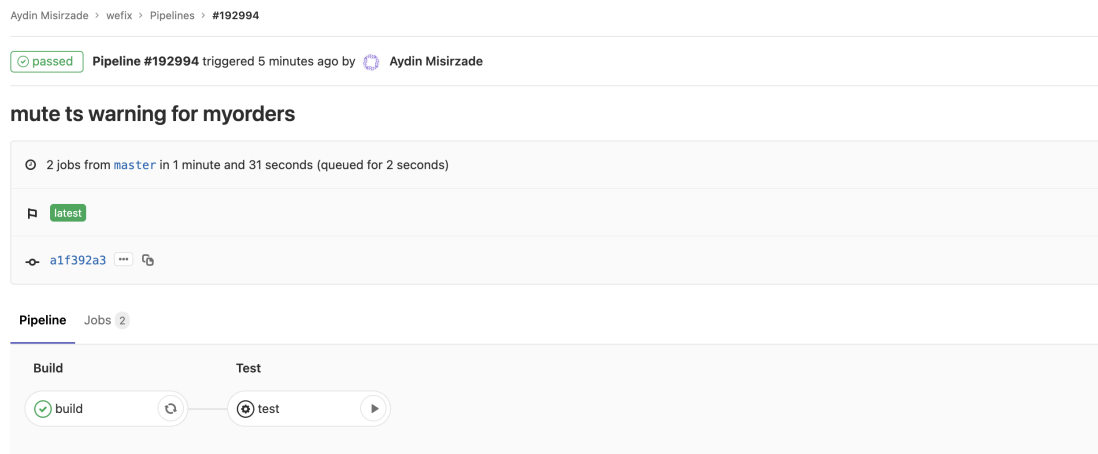
CI requires a version control system that can track changes and versions of software code. Git is a popular version control system. Using a git workflow, you can start the CI process. Developers working on a codebase use their version control systems to push their changes to a repository of code. The developers working on the codebase merge these changes into the main code branch once it has been reviewed.

Commonly, a developer will commit some code to a version control system like Git, which will trigger the CI process. This codebase is often scanned or analyzed using a static code analysis tool to determine code quality. If the source code passes all checks, including unit testing, the CI process will attempt to package or compile the code. CI can also involve tagging or annotating specific versions of code or managing resources such as branches within a version control system.

## 6.2.1 GitLab CI

Because I use GitLab version control system to store the project repository, I decided to use GitLab's native continuous integration tool called GitLab CI. The way it works is very easy: whenever you push a code to the GitLab, GitLab CI starts a pipeline with that specific commit and runs scripts against the code in that commit. The scripts are defined in `.gitlab-ci.yml` file, which is stored in the root of the repository. Creating the `.gitlab-ci.yml` file is all it takes to set up the CI tool.

GitLab CI also provides functionality to schedule pipelines. The pipelines schedule runs pipelines in the future, repeatedly, for specific branches or tags. Those scheduled pipelines will inherit limited project access based on their associated user. Another helpful feature of GitLab CI is analytical dashboard, which shows the statistics for pipelines



■ **Figure 6.1** An example of a CI pipeline run after commit

It can be seen from the figure 6.1, that there are two stages in our pipelines: build and test. For the current scale of our project, I decided that these two stages will be enough. After all, the project is developed by me only. Normally, companies would add a bunch of more stages, like smoke tests and integration tests, with the deployment to some staging (sometimes called alpha) environment as the last stage. On the figure, `test` stage has been figured to be run manually, while the `build` was run automatically and passed. Whenever any of the stages fails, that specific stage will be colored red, and the whole pipeline will be marked as failed.

The build stage, as the name suggests, builds the application. If at this stage, the pipeline fails, it means that there are serious problems in the code, and WeFix will not start. The test stage triggers the unit tests, and will fail if any of the tests fail.

Clicking on the failed stage will reveal the terminal window with the detailed errors. The terminal window shows all the processes related to scripts specified in the pipeline configuration file, including all the "print statements" and errors. This is very convenient to understand what happened wrong in your last commit.

```
$ npm install --legacy-peer-deps
npm WARN deprecated source-map-resolve@0.6.0: See https://github.com/lydell/source-map-resolve#deprecated
npm WARN deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.

added 1432 packages, and audited 1433 packages in 28s

175 packages are looking for funding
  run `npm fund` for details

7 vulnerabilities (6 moderate, 1 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
$ npm run build
> wefix@0.1.0 build
> react-scripts build

Creating an optimized production build...

* daisyUI components 2.14.1 https://github.com/saadeghi/daisyui
  ✓ Including: base, components, themes[], utilities

Treating warnings as errors because process.env.CI = true.
Most CI servers set it automatically.

Failed to compile.

src/components/order/MyOrdersPage.tsx
  Line 25:44: Unexpected any. Specify a different type @typescript-eslint/no-explicit-any

ERROR: Job failed: exit code 1
```

■ **Figure 6.2** Detailed view of failed CI stage

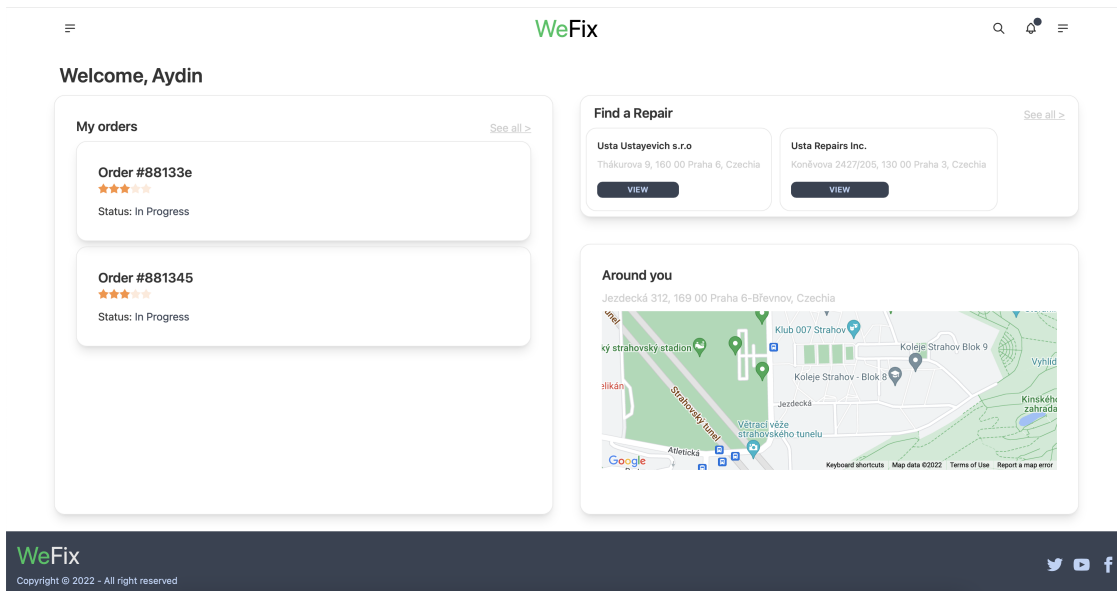
# Chapter 7

## Conclusion

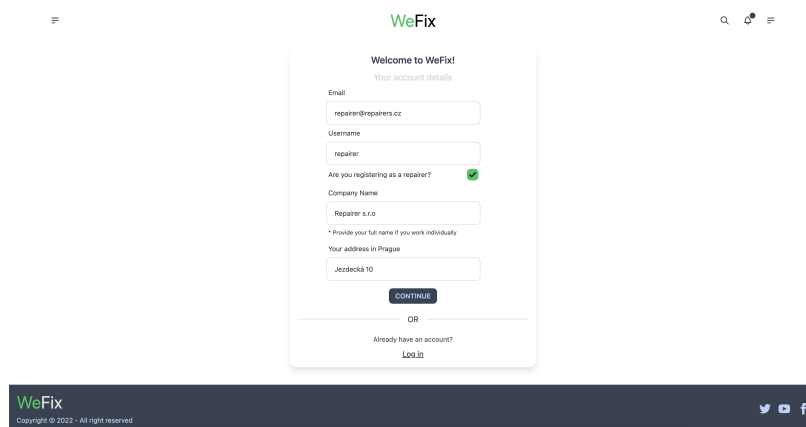
### 7.1 Evaluation

In this section, the final results of the development process in a form of screenshots are shown, drawbacks of the application are mentioned and corresponding solutions to them examined and new functionality proposed.

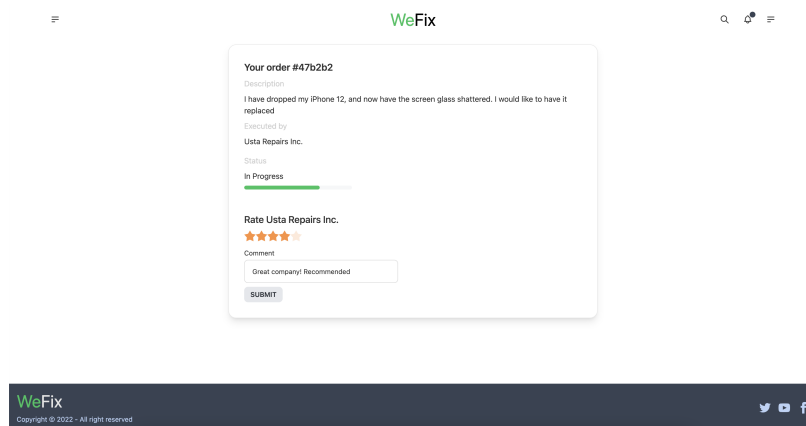
The application is currently deployed to Microsoft Azure server and accessible from the internet on <https://we-fix.cz> and is already fully functional. Figures 7.1, 7.2, 7.3 and 7.4 on next pages show screenshots of the current version of the application.



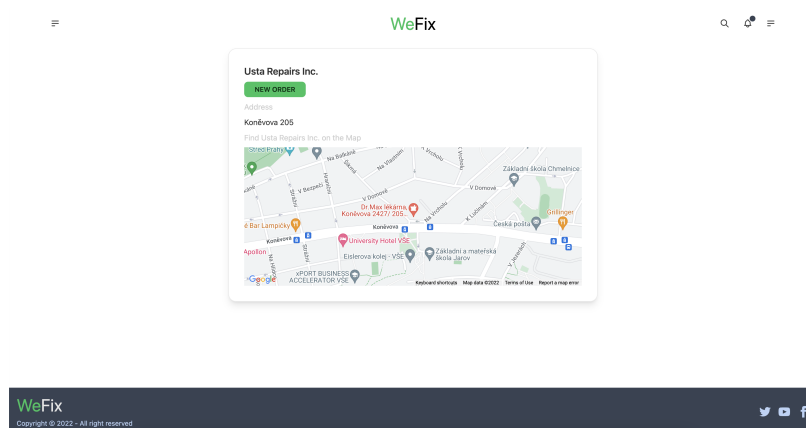
■ Figure 7.1 Home Page of WeFix



■ Figure 7.2 Registration View



■ Figure 7.3 Order View



■ Figure 7.4 Repairer Page

## 7.2 Drawbacks and Future Improvements

The main drawbacks of the application is that it is not really a fully-fledged product, ready for a wide number of customers. It is still an MVP, which showcases how the problem described in



the first two chapters may be solved. The product lacks some of the basic use cases, which were described in chapter 5.

The plans for the future of the product is clear: continue supplying the product with new features, support more use cases, and continue researching and improving the UI of the app. Some of the future features currently under consideration are:

- Search System
- Notification System
- Full support for account management
- Interactive (or customizable) homepage layout

I believe that at least with these features shipped, the product has a potential to conquer Czech market and easily become number one, given that there is still no similar product in existence.

### 7.3 Final Thoughts

The objective of this thesis was an implementation of a web application designed for facilitating repairer, or repairing services, finding. The whole process of the application creation was described – starting from the motivation, showing the reason it was decided to implement this application, continuing with thorough analysis, describing our brainstorming, from which the basic principles and functional requirements of the application emerged. The analysis continued with comparing the application with similar solutions already available on the Czech market, while emphasizing the main advantage of our application – uniqueness, and ease of use. Following the analysis, the next chapter focused on the design of the application, briefly going through technologies which were used, explaining the database design and briefly describing web application architectures. This served as basis to next chapter, implementation, which describes certain parts of implementation process in detail, such as API implementation, integration with Google Maps, and the front-end architecture.

The methodology of continuous integration was also integrated into the development process. The application was deployed to a Microsoft Azure server and deployed there. The concluding chapter, *Testing and Pipelines*, described this integration, and the way the application was tested. The user testing turned out to be quite useful, as it revealed a number of mistakes and the notes taken while conducting the testing were the basis for list of improvements and features to be implemented in next version.

To conclude, the application was successfully analysed, designed, implemented and tested and every part of the process was described in the thesis. Despite few listed drawbacks, the application is currently fully functional and has a potential to be offered to potential customers all over the Czech Republic soon.



# Bibliography

1. ALLBETTER. *5 Best On-Demand Handyman Apps of 2022*. Available also from: <https://allbetterapp.com/5-handyman-apps/>.
2. *NejŘemeslníci.cz*. Available also from: <https://nejremslnici.cz>.
3. *Opravárna*. Available also from: <http://opravarna.cz>.
4. LI, Shirley. *Most People Can't Be Bothered to Download Apps*. Available also from: <https://www.theatlantic.com/technology/archive/2014/08/most-people-cant-be-bothered-to-download-apps/378989/>.
5. KAFKA, Peter. *Most People Can't Be Bothered to Download Apps*. Available also from: <https://www.vox.com/2016/6/8/11883518/app-boom-over-snapchat-uber>.
6. KRIGSMAN, Michael. *Study: 68 percent of IT projects fail*. Available also from: <https://www.zdnet.com/article/study-68-percent-of-it-projects-fail-6103001175/>.
7. YASKEVICH, Anastasia. *Web application architecture: Components, models and types*. Available also from: <https://www.scnsoft.com/blog/web-application-%20architecture>.
8. TEDIKOV, Oleksandr. *Uncovering web application architecture: How to Choose the right type*. Available also from: <https://perfectial.com/blog/web-application-%20architecture/>.
9. OSETSKYI, Victor. *Web application architecture*. Available also from: <https://medium.com/existek/web-application-architecture-da77ea0cb520>.
10. *Web Application Architecture: Definition, Working And Types*. TechSocial. Available also from: <http://techsocial.net/web-application-architecture/>.
11. *WEB APPLICATION ARCHITECTURE: THE BASICS*. Intellectsoft US. Available also from: <https://www.intellectsoft.net/blog/web-application-architecture/>.
12. SINGHAL, Gaurav. *Why Do We Need Single-page Applications?* Available also from: <https://www.pluralsight.com/guides/why-do-we-need-a-single-page-application>.
13. *Understanding the Differences Between Microservices, Monoliths, SOA and APIs*. CMS Wire. Available also from: <https://www.cmswire.com/web-development/understanding-the-differences-between-microservices-monoliths-soa-and-apis/>.
14. *Common web application architectures*. Microsoft. Available also from: <https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/common-web-application-architectures>.
15. ARSHED, Saad. *Monolithic vs SOA vs Microservices: How to Choose Your Application Architecture*. Available also from: [https://medium.com/@saad\\_66516/monolithic-vs-soa-vs-microservices-how-to-choose-your-application-architecture-1a33108d1469](https://medium.com/@saad_66516/monolithic-vs-soa-vs-microservices-how-to-choose-your-application-architecture-1a33108d1469).

16. *Best Architecture for an MVP: Monolith, SOA, Microservices, or Serverless*. Ruby Garage. Available also from: <https://rubygarage.org/blog/monolith-soa-microservices-serverless>.
17. *JSON Web Token*. Available also from: <https://jwt.io>.

# List of Abbreviations

API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CSS	Cascading Style Sheets
DB	Database
HTML	Hypertext Markup Language
JS	Javascript
JSON	Javascript Object Notation
JWT	JSON Web Token
MVP	Minimum Viable Product
UI	User Interface
UX	User Experience



# List of Figures

1.1	A person uses his smartphone and Uber application to order a taxi . . . . .	1
3.1	An interface for new order at NejŘemeslníci.cz . . . . .	5
3.2	Home page of Opravárna . . . . .	6
4.1	Database Design of WeFix MVP . . . . .	13
5.1	WeFix Theme Palette . . . . .	20
5.2	WeFix "My Orders" widget . . . . .	21
5.3	WeFix Registration Page . . . . .	21
5.4	Desired home page layout . . . . .	22
5.5	WeFix Home Page . . . . .	22
5.6	"Around You" widget . . . . .	23
6.1	An example of a CI pipeline run after commit . . . . .	27
6.2	Detailed view of failed CI stage . . . . .	28
7.1	Home Page of WeFix . . . . .	29
7.2	Registration View . . . . .	30
7.3	Order View . . . . .	30
7.4	Repairer Page . . . . .	30





# List of Attached Files

```
├── readme.txt..... the file that contains all the technical explanation
├── src
│   ├── wefix.....source code of the front-end application
│   ├── backend.....source code of the back-end application
│   └── thesis ..... the directory of LATEX source codes of the thesis
├── text
│   └── thesis.pdf ..... the thesis text in PDF format
```