



**FACULTY
OF MATHEMATICS
AND PHYSICS**
Charles University

BACHELOR THESIS

Patrícia Lakatošová

Taxi system front-end

Department of Distributed and Dependable Systems

Supervisor of the bachelor thesis: doc. RNDr. Tomáš Bureš, Ph.D.

Study programme: General Computer Science

Study branch: Computational linguistics

Prague 2018

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In date

signature of the author

I would like to thank my supervisor for his comments, advices, efficient communication and understanding. My special thanks belongs to my fiancé who was with me in the toughest moments and despite it all decided to marry me. I could not forget my roommate who struggled through the same and worse and still managed to make my life less miserable. My sincere gratitude goes to Taxi Ali Mladá Boleslav for invaluable insight on inner management of a taxi company with a lot of practical advice and plenty of consultations. Last but not least I want to thank my family for their support and patience and especially my uncle Dominik for his review and commentary.

Title: Taxi system front-end

Author: Patrícia Lakatošová

Department: Department of Distributed and Dependable Systems

Supervisor: doc. RNDr. Tomáš Bureš, Ph.D., Department of Distributed and Dependable Systems

Abstract: In this thesis we implement front-end of a taxi system. It is based on our knowledge of how a local taxi company operates and is intended to be used by them to manage orders and staff. It is composed of web applications for dispatching and drivers, and a PWA for customers. We developed it using the newest technologies - Angular 6. Focus was centered on separating logic in services from design in components and reusability of code. We greatly eliminated the need for typed input from driver compared to other applications therefore increasing customer's safety. Since maps are a big part of all our applications we made them comfortable to use. Overall this system automates ordering process and lowers demands on dispatchers.

Keywords: taxi system, front-end, web application, Angular, PWA

Contents

1	Introduction	4
1.1	Motivation	4
1.2	Existing applications	5
1.2.1	Uber	5
1.2.2	Modrý anděl	5
1.3	Goals	5
1.4	Outline	6
2	Problem structure	7
2.1	Roles	7
2.2	Objects	7
2.2.1	Order	8
2.2.2	Employee	9
2.2.3	Car	10
2.2.4	Customer	10
3	Detailed analysis	11
3.1	Dispatching application	11
3.1.1	Login	11
3.1.2	Create new order	11
3.1.3	Latest orders	11
3.1.4	Scheduled orders	12
3.1.5	My phone orders	12
3.1.6	Edit profile	12
3.1.7	Notifications	12
3.1.8	Cars	12
3.1.9	Employees	12
3.1.10	Order history	13
3.2	Driver application	13
3.2.1	Login	13
3.2.2	New order	14
3.2.3	Arriving	14
3.2.4	Arrived	14
3.2.5	Picked up customer	14
3.2.6	Finished order	14
3.3	Customer application	15
3.3.1	Login	15
3.3.2	Registration	15
3.3.3	Forgotten password	15
3.3.4	Start selection	16
3.3.5	Finish selection	16
3.3.6	Standard order confirmation	16
3.3.7	Airport order confirmation	17
3.3.8	Driver's arrival	17
3.3.9	Order cancellation	17

3.3.10	Scheduled orders	18
4	Technical analysis	19
4.1	Web application technology	19
4.1.1	Angular	19
4.1.2	Vue	19
4.1.3	React	20
4.1.4	Selecting Angular	20
4.2	Native application technology	20
4.2.1	NativeScript	20
4.2.2	PWA	20
4.3	Libraries	21
4.3.1	Angular Material	21
4.3.2	Google Material Icons	21
4.3.3	Angular Google Maps	21
4.3.4	Communication with server	21
5	Architecture	22
5.1	Modules	22
5.2	Components	22
5.3	Services	24
5.4	Project structure	24
6	Implementation	26
6.1	Dispatching application	26
6.1.1	Login	26
6.1.2	Create new order	27
6.1.3	My phone orders	28
6.1.4	Latest orders	28
6.1.5	Scheduled orders	29
6.1.6	Notifications	29
6.1.7	Edit profile	30
6.1.8	Cars	30
6.1.9	Edit car	31
6.1.10	Employees	32
6.1.11	Employee profile	33
6.1.12	Order history	34
6.2	Driver application	35
6.2.1	Choose car	36
6.2.2	No orders	37
6.2.3	New order	38
6.2.4	Arriving	39
6.2.5	Arrived	40
6.2.6	Fraud	41
6.2.7	Picked up customer	42
6.2.8	Enqueued orders	43
6.2.9	Break	44
6.3	Customer application	45
6.3.1	Login	45

6.3.2	Registration	46
6.3.3	Forgotten password	47
6.3.4	Start selection	48
6.3.5	Standard order confirmation	49
6.3.6	Airport order confirmation	50
6.3.7	Driver's arrival	51
6.3.8	Scheduled orders	52
6.3.9	Reusing message component	53
7	Evaluation	54
7.1	Testing	54
7.2	Meeting goals	54
	Conclusion	56
	Bibliography	57
	List of Figures	58
	List of Abbreviations	59
A	Appendix	60

1. Introduction

People have always had a desire to travel. Since cars were invented, traveling became easier, faster and more comfortable. However, you don't always have your car around when you need to transfer and that's where taxi steps in. It has been around for a while now but it has barely changed the way it works. With all the technology available these days a significant improvement in complexity, comfort and user experience can be achieved.

In this thesis, we create a front-end for taxi system consisting of dispatching, driver and customer application that offers an easy way to manage orders for a local taxi company and thus enables them to provide better service for their customers. It is designed to meet company's requirements and support smooth transmission from current application-free state. This application communicates with back-end implemented by a different student in a separate bachelor thesis.

Through this application we want our dispatchers to be able to create and manage orders as well as check the history in case of need. Our drivers want to see their orders queue with any changes that may happen in a clear and simple style without distraction. Customers would like to create an order easily and quickly, and receive more information on their driver.

1.1 Motivation

Mladá Boleslav, where targeted taxi company operates, is an area with very low unemployment rates therefore if employers want to offer better services, they have very little leverage on their employees to perform better. A good dispatcher has to know the city thoroughly and be able to make an accurate prediction of arrival time. Someone with these qualifications is very hard to find in this area. Using our application nearly anybody can be hired and the only skill they "need" is to get familiar with our application. It can calculate arrival time more accurately and display it to the customer even with updates in case of unexpected changes.

This step makes dispatcher's job much easier what leads to smaller number of dispatchers needed to process the same amount of orders. We aim to minimize the need for dispatchers and thus boost the profit while keeping the quality.

Additionally, they cannot process more orders at one time than there are dispatchers present. Customers often have to wait on hold to be served for a long time and most likely they decide to call a different taxi company although we still might have free drivers for the job. Our application takes the burden off of the telephone lines and enables creating orders in parallel.

Current state of communication between dispatchers and drivers is alarming. Assigning orders as well as accepting them, canceling and changing is arranged using Facebook Messenger. Customers complain on feeling unsafe when their drivers type while driving. It often escalates to chatting with other drivers and confusion on dispatcher's side. We can bring more safety to our streets limiting this entire communication to a few quick taps on a button.

Ultimately, local taxi companies as we know them are being challenged by global players like Uber that adapted to online world more quickly. If they don't catch up their end is inevitable. This may not seem like a bad thing at first but

it will only support American economy at the expense of our own and absence of competition will yield higher prices and lower quality.

1.2 Existing applications

Naturally, some applications has already been created in an attempt to automate the process of creating orders. However, since they have developed in a commercial and highly competitive environment, they are protected by trade secret which keeps us from being able to observe or compare their approach to the problem. The only publicly available part is usually customer application. In this section we analyze and compare our application with some of the most significant applications in this field.

1.2.1 Uber

Uber is an an American ridesharing company that operates in 633 cities worldwide at the time of writing Ube [2018]. It consists of driver and customer application where anybody can sign up to be a driver. Unquestionably they had developed high quality application but with a different idea in mind. They are dependent on ordinary people willing to wake up and become Uber driver for that day and their prices change depending on supply and demand. We want to keep the stability of taxi service with constant supply of drivers and fixed prices. Additionally we are keeping the option to create an order by calling dispatching for people who prefer it for various reasons.

1.2.2 Modrý anděl

Modrý anděl is a taxi company operating in Prague with a customer application most similar to what we are creating. We got inspired by some good ideas from them and found a different solution for parts that were less user friendly. Main improvements include better integration of airport orders, overall simpler navigation, easier address selection on map, and more modern design. We also used newer technology allowing it to run on both Android and iOS as progressive web app (PWA), in a web browser as a standard web application and even on desktop as Windows application.

1.3 Goals

The goal of our thesis is to create front-end for taxi system. It will provide all basic functionality needed by a common taxi company. We will create three applications - for drivers, dispatchers and customers.

- Each application is designed for a specific device and our design will correspond to that. Dispatching application is designed for desktops because dispatchers sit in the office with one or two monitors where they like to view all the information they need systematically arranged on large screens. This application also has the most features and information that needs to

be displayed so fitting it on a small screen would be inappropriate and overwhelming for the user. Driver and customer applications, on the other hand, will be viewed almost exclusively on mobile devices so we will make them easy to use with simple navigation.

- In the process of choosing technology we will focus on the cross-platform options. We want all of our customers to be able to view it but each of them has a different device with a different operating system. In order for our application to be used, we need it to work well on as many devices as possible. We will choose technology we will consider the most helpful in achieving this.
- We aim to increase safety of customers. Our driver application will minimize driver's input needed allowing them to focus on driving more. It will simplify navigation and have no typing required on the road. More driver input will be required only at time when the vehicle is expected not to be moving.
- During the past three years of our bachelor studies we have learnt the importance of clean structuring and creating reusable parts of code. We have discovered how one can save a lot of time just by spending a little more on thinking it through. We will now use this knowledge in practice and implement it in this complex project. Our goal is not to repeat big chunks of code and structure components in an easily apprehensible way. Our application will be easily extensible and maintainable for the future.

1.4 Outline

In the beginning we introduced our project and explained the motivation behind it. We compared it to related projects and set up our goals.

Chapter 2 discovers what roles and objects will be needed to describe our problem and what properties should they contain.

Chapter 3 analyzes our problem into depth and creates use cases. It defines basic flows and shapes what our application will look like. It is the most important part to avoid confusion during development.

Chapter 4 sums up our technological options and explains why given technologies were chosen. It also lists libraries used.

Chapter 5 shows the architecture of our project and how Angular shaped it. It further describes what are the building blocks of an Angular application.

Chapter 6 presents screenshots of our three applications and comments on their implementation. It explains the most complex parts and show how user navigates between them.

Chapter 7 evaluates our final product. It looks back on the goals we set and whether we managed to reach them. It also re-evaluates the technology choice.

In Conclusion we sum up our work and lay out the possibilities for new features.

2. Problem structure

This chapter analyzes real-world problem into more technical structure than can be implemented and establishes naming conventions for different objects and actions used in the project. It is the result of pre-implementation stage when we tried to simplify the problem and differentiate roles, objects, and actions.

Our project consists of three applications:

- dispatching application
- driver application
- customer application

Different entities can be associated with one or more applications depending on who has the right to use them.

2.1 Roles

Our analysis led us to establish 5 roles:

- administrator
- dispatcher
- driver
- registered customer
- unregistered customer

Dispatcher, driver and customer are derived intuitively from the project structure.

Administrator is the only role that can edit and manage other roles. It has the same privileges as dispatcher but can also edit other dispatchers, customers, drivers and cars.

We decided to split customers into two roles based on registration. Main reason is that when a customer is not registered, they have different rights on what features they can use and what information they see. Additionally, registered users provide us with more information that we can use to skip unnecessary steps in order creation. Customers are identified by their phone number so when an unregistered customer registers in the application, even orders created in unregistered state will appear in their history.

One person can possess multiple roles, only registered and unregistered customers are mutually exclusive.

2.2 Objects

In this section we will describe most common objects needed in this project and what properties we need to store in them.

2.2.1 Order

Order is the most complex object. Although it could be simplified for the needs of this project we decided not to because there is much useful information that we can benefit from in the future after some statistical data had been collected.

There are 3 main types of orders:

- standard
- scheduled - order created earlier for a specified time in the future
- airport - from/to the airport (driver adjusts time to the delay of the plane)

Order has several states it can appear in but only in order specified by diagram 2.1.

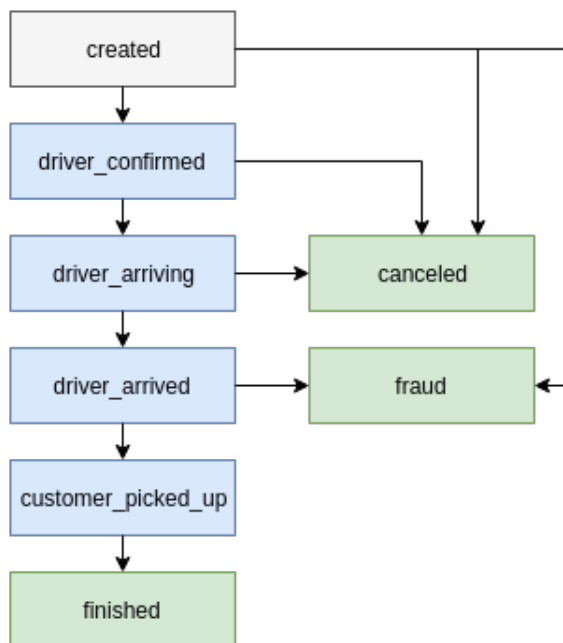


Figure 2.1: Order status

For each order we store this information:

- id
- status - one of statuses listed in 2.1
- driver - who drove this order
- dispatcher (only when order was created by phone call) - who created this order
- vehicle - that was used for this order
- customer
- start - latitude and longitude location, address, date and time when driver started working on this order (both estimated and final)

- finish - latitude and longitude location, address, date and time when driver finished order (original estimate, current estimate, final)
- picked up - time when customer was picked up (original estimate, current estimate, final)
- arrived - time when driver arrived on starting point selected by customer (original estimate, current estimate, final)
- passenger count
- note - form customer to driver
- contact telephone - only when it is different from the phone on which the order was made
- VIP - boolean for luxury services
- estimated price
- scheduled pick up at - time and date of scheduled order
- source - created in customer application or through dispatching
- flight number - airport order only
- explicitly chosen driver - not the fastest driver was chosen
- time of creation and last update of this order

Storing our original and current time estimates for different stages of an order helps us evaluate how good is our system in predicting these times and it can later be improved to shorten waiting times.

2.2.2 Employee

Employee is a driver, dispatcher, administrator or any combination of these. They all need to have these properties:

- id
- name
- image
- email
- employee roles - an array of roles the employee has; may contain: driver, dispatcher, admin

2.2.3 Car

A car object needs to have these properties:

- id
- name - describes car brand
- image - base64 encoded picture of the car
- plate
- max-persons - maximum number of people that can be transported by this car (excluding driver)
- available - true if car can be used for driving (e.g. not currently in repair)
- driver-id - id of driver currently using this car or null if not in use

2.2.4 Customer

We need this information about each customer:

- id
- phone number
- name - not required for unregistered customers
- note - internal note written by dispatchers about a customer (e.g. 'changes his mind very often')
- token - authentication token assigned with each login

3. Detailed analysis

In this chapter we separate what to do from how we do it. Since our application has broad functionality we need to analyze its flow in enough detail not to let our user do things that were not intended. Fully describing our functions before programming them will help us gain an overview of the application as a whole and plan its user interface (UI) better. It also speeds up the programming process because we do not have to think about what is next or how should we handle errors. We can analyze it using standard use cases for each of our applications.

3.1 Dispatching application

In all use cases user represents dispatcher or administrator, unless stated otherwise.

3.1.1 Login

Basic flow: User enters email and password and presses 'Login' button.

Exception flows: Invalid credentials, inform user that login failed.

Post-conditions: Application routes to Create new order screen.

Following actions are available only after a successful login.

3.1.2 Create new order

Basic flow: User enters customer's phone number. A request is sent to server that loads customer's details (e.g.name). User enters start and destination on the map or selects airport icon if it is an airport order. Optionally, VIP mode can be selected or date and time can be set in case of scheduled order. Fastest driver will be calculated based on start and finish address and it can be changed by dispatcher if desired. In case of scheduled order, driver can be picked from a list of all drivers.

Exception flows: Inputs with invalid values will be highlighted.

Post-conditions: Order is confirmed and appears in selected driver's queue or in scheduled orders table.

3.1.3 Latest orders

Basic flow: User can see a table of orders divided into three parts:

1. Ongoing orders
2. Orders waiting to be process (they are in some driver's queue but not on the first place)
3. Last orders that had been finished

Row contain orders and column represent details like driver, car, type, start and finish, status etc.

3.1.4 Scheduled orders

Basic flow: User sees a table with scheduled orders and can change assigned driver or cancel the order.

3.1.5 My phone orders

Basic flow: At all times dispatcher will have a list of orders process by him displayed on a side panel. For each order they will see assigned driver, customer's number and name (if known), estimated arrive time and delay. User should inform the customer in case of a significant delay.

Exception flows: User should be notified if this list fails to refresh.

3.1.6 Edit profile

Basic flow: User can change their name, password and picture.

Exception flows: Empty name and short password is not permitted.

3.1.7 Notifications

Basic flow: User clicks on a notification icon and a list with notifications rolls down. A notification can be of these types:

- A scheduled order in the near future does not have a driver assigned or assigned driver is not on shift.
- Driver complains that their customer is not present on the start.
- A customer with history of fraud created an order.

When new notification arrives, it is emphasized with a sound effect. To remove a notification from the list user has to mark it as solved.

3.1.8 Cars

Actors: administrator only

Basic flow: User views a list of all cars. They can edit and remove cars. To add a new car, user has to supply an image and specify plate, number, maximum number of passengers, name and availability.

Exception flows: User cannot mark a car that is currently in use as unavailable. Invalid input are highlighted.

3.1.9 Employees

Actors: administrator only

Basic flow: User can view a list of all dispatchers and drivers and click on them for more information. They can deactivate user or add a new one. For a new user their email, full name, picture and roles have to be entered. After creating new employee they will receive an email with activation link where they can choose password.

Exception flows: Invalid values are highlighted.

3.1.10 Order history

Basic flow: User can view orders for an arbitrary time period. For each order they will see:

- driver and vehicle
- start and finish
- order creation time
- customer
- note
- type - application or phone call
- time of important events

Exception flows: Check whether the time period is valid.

3.2 Driver application

Except login, all following actions describe various states that the application goes through. At any state, driver sees a short menu with a few simple actions available at the top of the screen. These 3 quick actions include:

- Phone icon to call dispatching.
- An icon to plan a break at the end of their order queue. No more orders will be assigned to this driver and their break will start after they finish remaining orders. During the break, there will be a button to end the break. Upon tapping this button, driver can receive new orders again. To optimize arrival times, orders from other drivers' queues can be reassigned to this driver.
- Logout icon - user will be logged out and all their orders will be reassigned to remaining drivers. User should not log out when their queue is non-empty unless it is an emergency. To end shift, user should plan a break, finish remaining orders in the queue and only then log out.

In all use cases user represents driver.

3.2.1 Login

Identical as in dispatching application. The only difference is that after the login user has to choose a car he drives from the list of available cars in order to start a shift. After starting shift, an order queue is created for the driver where they receive order. Following actions represent first order in the queue, the rest of the queue can be viewed by scrolling down.

3.2.2 New order

Basic flow: Appears as a pop-up covering entire screen. Shows start location on a map, customer's name and note. User can accept or reject the order. When accepted and no orders are in the queue, order will be marked as arriving and order details will be displayed.

Alternate flows: When there are some orders in the queue, user's current order will appear on screen and new order will be displayed on the bottom of the order queue.

Exception flows: When user does not react to new order for a long period of time, it will be reassigned to a different driver.

3.2.3 Arriving

Preconditions: User accepted the order.

Basic flow: User sees current order's customer, note and start location on a map. Current estimate of arrival time is displayed and user can regulate it using simple plus and minus buttons according to current traffic or unexpected complications.

Exception flows: At any time customer can cancel the order, user is notified about it and next order from the queue appears. If customer canceled an order in the queue, user is notified and order disappears from the queue.

Post-conditions: Upon arriving to the start user taps on 'Arrived on start'.

3.2.4 Arrived

Preconditions: User arrived on start.

Basic flow: User is waiting for the customer. They see the customer's name, note, and start location as before. When customer arrives, user taps on 'Customer picked up'.

Alternate flows: Customer is not showing up. User taps on 'Notify dispatching' and waits for them to respond. If it takes too long, user can mark the order as a fraud and proceed to next order in the queue.

3.2.5 Picked up customer

Preconditions: Customer got in the vehicle.

Basic flow: User views a map with finish location along with details about the customer. They can edit finish location in case user chose not to enter it or it was entered incorrectly. Arrival time to finish can be adjusted using plus and minus. This serves to adjust arrival times for next orders in the queue. After arriving to the destination, user marks this order as *finished*.

Exception flows: Order cannot be marked as *finished* if finish location is not entered.

3.2.6 Finished order

Preconditions: User dropped the customer off at the destination.

Basic flow: Next order from the queue is given *arriving* state.

Alternate flows: User does not have more orders in queue; waiting screen appears until new order arrives.

3.3 Customer application

In all use cases arbitrary customer represents user unless stated otherwise.

3.3.1 Login

Preconditions: User started application without having logged in previously or after logging out.

Basic flow: User enters phone number and password and presses 'Log In' button.

Alternate flows: User decides not to log in and presses 'Continue without login' button or does not have an account and presses 'Register' button.

Exception flows:

- Entered credentials are invalid. User will be notified by changing the invalid control's color to red.
- User forgot password. User presses corresponding button and continues to Forgotten password screen.

Post-conditions: Application routes to a screen where he can select order type, view his scheduled orders or sign out.

3.3.2 Registration

Preconditions: User selected Register on login screen.

Basic flow: User fills in full name, phone number, password and password confirmation and presses 'Register'. They receive SMS message with verification code which they enter into application to confirm registration.

Exception flows:

- Inputs turn red if entered values are invalid.
- User did not receive SMS code; they can ask for it to be resent.
- Registration is unsuccessful if user with given phone number already exists.

Post-conditions: User is forwarded to login screen where he can log in and continue.

3.3.3 Forgotten password

Preconditions: User selected Forgotten password on login screen.

Basic flow: Application asks for phone number to which it sends verification SMS code. After entering it new password can be typed in.

Exception flows:

- Inputs turn red on invalid phone number format, incorrect SMS code or if entered passwords do not match.

- User can ask to have the verification code resend if they haven't received it.

Post-conditions: User will be forwarded to login screen.

3.3.4 Start selection

Preconditions: User selected standard order, or airport order with direction to the airport.

Basic flow: User sees a map with their location (if known) and they can change it by moving the map or entering an address. Application will autocomplete addresses from Google maps and user's favorite locations (if logged in). Button to return to user's current location will be present.

Exception flows:

- When user's location is unknown, map will center on default location.
- User will not be allowed to continue unless they select an address within specified area.

Post-conditions: In standard order they will proceed to select finish location, on airport order it will be sent to server and a message will be displayed to wait for confirmation from driver.

3.3.5 Finish selection

Preconditions: User selected start location in standard order, or chose airport order with direction from the airport.

Basic flow: User selects finish location on a map as in Start selection.

Alternate flows: User may decide to skip this step in standard order.

Exception flows: Finish must be in designated area as in Start selection.

Post-conditions: User proceeds to confirm standard order or send airport order.

3.3.6 Standard order confirmation

Preconditions: User selected start location and finish location.

Basic flow: Application shows:

- Fastest driver and car for the order. User can change it by clicking on the driver and selecting from a list of available drivers with their cars and arrival times.
- Minutes it will take for selected car to arrive to given start location.
- Start and finish address
- Number of people (editable, default is 1)
- Contact phone number (editable, default is user's number if known)
- Scheduled order toggle - when checked, date and time for the order can be chosen

- Note for the driver (editable)
- Estimated price (registered users only)

User can edit and check given details and send the order.

Alternate flows: When scheduled order is selected, driver selection will not be possible.

Exception flows:

- Scheduled order cannot be planned for earlier than when some driver is available.
- If selected driver becomes unavailable at the time of sending the order, user will be notified and drivers' times will be updated.
- Number of passengers cannot exceed the capacity of the car.

Post-conditions: Order is send and message is displayed to user to wait for confirmation from the driver.

3.3.7 Airport order confirmation

Actors: logged in user

Preconditions: User selected airport order.

Basic flow: User fills in date and time, flight number, number of passengers, contact phone number, note for driver (optional), and direction (to/from the airport).

Exception flows: Invalid values will be highlighted.

Post-conditions: User proceeds to choose start/finish.

3.3.8 Driver's arrival

Preconditions: User sent the order, driver confirmed it and is heading to start location.

Basic flow: User can watch driver's location on map refreshed periodically. Car's plate and current estimate of arrival time are displayed below. When driver arrives, user is notified. Until then, user can cancel the order at any time. When user begins the ride, they can see their estimated time to the final destination.

Exception flows:

- If driver changes for any reason, car plate will be updated.
- If user does not show up at start, order is canceled, marked as fraud and user is notified about it.

Post-conditions: When order is marked as finished, user is thanked and offered and option to create new order.

3.3.9 Order cancellation

Preconditions: User clicked on 'Cancel' button before driver arrived.

Basic flow: Request is sent to server about cancellation.

Post-conditions: Message about success of this request is displayed.

3.3.10 Scheduled orders

Actors: logged in customer

Preconditions: User clicked on 'Scheduled orders' in order selection menu.

Basic flow: User can view their scheduled orders with start, finish, and driver assigned (if any) and cancel them.

4. Technical analysis

This chapter describes the decision making on the technological side. The quality of our application also depends on what technology we choose to use. However this decision is not easy and has significant consequences on the rest of development process. Therefore we must inspect our options thoroughly.

Each of our applications has a different purpose so we need to consider them separately. Dispatching application needs to run on desktop computers while actively interacting with server. Creating it as web application instead of desktop application will allow dispatchers to work from home as well what will make hiring dispatchers easier for the taxi company.

Driver application on the other hand targets mobile devices so we could consider creating a native application. However we require it to run on all phones because we need all our drivers to be able to use it. Therefore we once again decided to create it as a web application.

Customer application also targets mobile devices but this time we are satisfied if vast majority of users can view it. Therefore we would like it to be more native so the customers do not have to remember the URL but they can have it in their phone in a form of an application.

4.1 Web application technology

For our web applications there are 3 most popular frameworks/libraries that we explored - Vue, React and Angular. We will look at their pros and cons and explain why we decided to choose or not to choose given framework.

4.1.1 Angular

Angular is a TypeScript-based open-source web application platform released by Google in 2016 (after its predecessor AngularJS). We like that, since it is a platform, it takes care for the entire front-end including routing, dataflow, etc. With typescript being an opinionated language it shares some similarities with the language we learnt during our studies - C#. It also has very detailed documentation for the platform itself Ang [2018a] as for various libraries it offers. It supports reusing components, PWA, two-way data binding and separating logic in services from design in components.

4.1.2 Vue

Vue is an open-source JavaScript framework released in 2014. It was inspired by AngularJS and shares some of its concepts. Its advantages compared to other frameworks are its fast performance and how lightweight it is. However, it still has significantly smaller community than the other two frameworks. Lately some programmers have raised complaints about its performance in bigger projects Insights [2018].

4.1.3 React

React is a JavaScript library released by Facebook in 2013. It has gained popularity and is now used the most among these Insights [2018]. It is designed to be just a smaller core to which various libraries can be added if needed. It has steeper learning curve than Angular Neuhaus [2017] but a larger community of contributors.

4.1.4 Selecting Angular

Since we would like our application to be sustainable in the future, we prefer technologies with larger communities.

React is targeted more on experienced web developers who are looking for a better technology but since we had absolutely no experience with web development before starting this project, it appeared more confusing. Additionally, from previous experiences we know that a lot of time is wasted on choosing the right library to use, often discovering its pitfalls half-way in implementation. This prolongs the development process and frustrates the programmer.

Although both React and Angular are very capable, reasons stated above made us select Angular. We used the newest version available at the time which was Angular 5 (and updated the project to Angular 6 when it was released).

4.2 Native application technology

For our customer application we are looking for a native technology. To create a native application we would need to code at least one for Android and one for iOS to cover majority of phones. This is too complicated and it would be an overkill because our application does not need a lot of processing power. Native-like application that runs both operating systems would be a best choice.

4.2.1 NativeScript

NativeScript framework seemed like a natural continuation for mobile platform. It supports Angular so we could use our previously gained knowledge and have a native-like application without the need to learn a completely new language. At the beginning of the project this was what we decided to use. However, during the development of the first two application a new technology became available - progressive web applications (PWAs).

4.2.2 PWA

From the beginning, PWAs were considered a breakthrough in mobile development. A developer only creates a web application as they normally would but with addition of service worker and a manifest in can be saved as a native application. No installation is required and a service worker caches data so the application does not need to be loaded from scratch every time, it can show push notifications and load updated content in the background and therefore run offline as well. The same application can still run in a browser for the users that did

not add it to their home screen yet. All this is supported by Angular so it did not take us long to figure out this technology will be even better for our application than NativeScript.

During the development of web applications a new version of Angular was released, so Angular 6 was used for PWA.

For coding we chose Visual Studio Code because it is available for both Windows and Linux and has good autocomplete plugins.

4.3 Libraries

In the section we describe libraries we chose to use in our project. We do not provide reasoning because Angular offers only one library for each functionality. There are some third-party libraries but they are not integrated as well and may not be updated as often therefore we decided not to use them.

4.3.1 Angular Material

Angular Material Ang [2018b] is the library we use the most throughout the project. It provides material styled components and theming that can be managed globally for the whole application. By supplying several palettes from the list, components are styled in these colors creating soothing look without much effort and the color can be changed any time. It offers a wide range of components to choose from and is easily imported. Its undeniable advantage is detailed documentation with examples.

4.3.2 Google Material Icons

Google offers wide range of icons that are supported by Angular Material and can be added as an HTML tag. We used this icons Goo [2018a] in our project to simplify navigation.

4.3.3 Angular Google Maps

Maps are used a lot in our application so we need a library that adds Google maps into our application. This is not an official Angular library but there are only two third-party libraries for this feature and this one has significantly simpler syntax and more users. Communication with Google Maps is via Javascript API provided by Google Maps Goo [2018b].

4.3.4 Communication with server

Communication with server is implemented as HTTPS requests using REST API. Request's body is in JSON format with desired data.

5. Architecture

This chapter explains the architecture of our code. During the development we were trying to implement Angular's best practices to achieve clean structure and comfortable maintenance even in a bigger project.

5.1 Modules

An NgModule is a container for a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities Ang [2018a]. Basically, modules can contain components and services that are related or serve similar purpose. Structuring our code into modules helps make our applications faster because Angular can lazy load modules. For example, when user wants to look at drivers, there's no need to load order module since it has nothing to do with drivers - Angular knows it and loads only driver module.

In each application there is one root module called AppModule. All the other modules are called feature modules and are included in AppModule as child modules. Launching the application means bootstrapping the root module.

Any module is basically a class decorated with @NgModule decorator. It can export components it contains or import an exported component from a different module.

5.2 Components

A component controls and adds logic to a part of application called view. It is also a class just like module only it is decorated with @Component decorator. Each Angular component consists of 4 files:

1. An HTML file describing view's structure.
2. A CSS file adding styles to the view.
3. A TS file describing logic of the view like button presses or lifecycle hooks.
4. An optional TS file for tests.

Component's HTML file does not contain pure HTML. It is enriched with Angular template syntax like directives, pipes and data binding. Directives like *ngIf and *ngFor hide or generate components. It does not simply label the component as hidden; it completely removes it from DOM. Pipes transform data like date and time to desired format. In two-way data binding, Angular combines property binding and event binding as shown in figure 5.1.

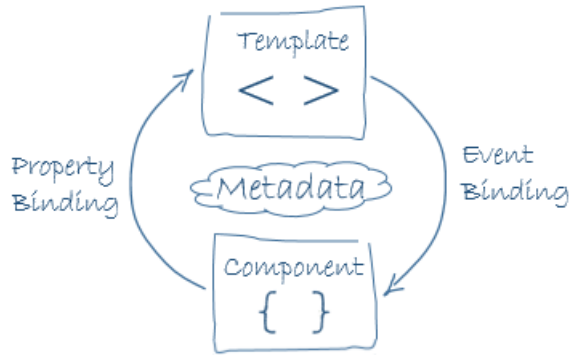


Figure 5.1: Component two-way data binding

A useful feature is that CSS styles affect only given component without affecting its children components. Lifecycle hooks allow you to execute code on components initialization or when the component is destroyed.

Components communicate between each other using @Input and @Output decorators. @Input allows passing a parameter into the component while @Output decorates an EventEmitter which creates an event on value change of the child's property, as illustrated in figure 5.2

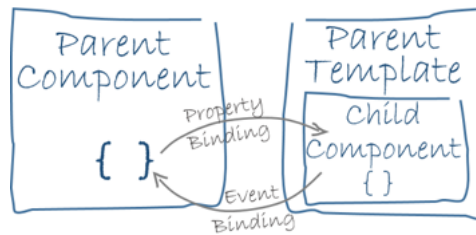


Figure 5.2: Parent-child binding

Component's TS file contains metadata that define which template belongs to the component, which service are needed and how the component can be referenced in HTML. Referencing one component in other component's template allows us to nest components in each other as showed in figure 5.3.

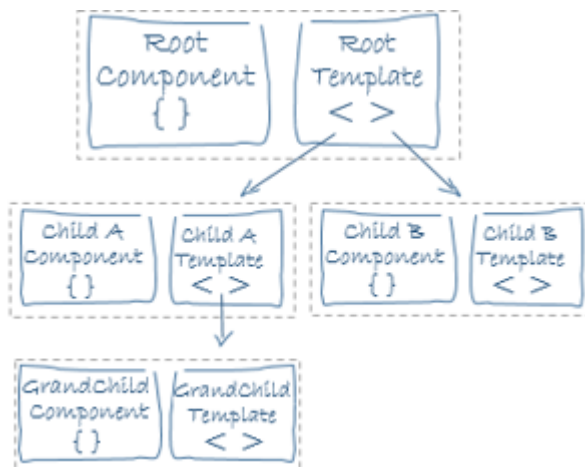


Figure 5.3: Component tree

5.3 Services

Angular's services take care of complicated logic and communication with the server. Each service needs to register at least one provider. Providers can be registered on different levels - in a component, feature module or a root module. This level determines how big of a scope shares the same instance of the service. A service therefore can act as a singleton data holder for multiple components. Service is added to the component in the constructor using dependency injection.

5.4 Project structure

In our project we created modules mostly similar to what objects we have. Our dispatching and driver application contains these feature modules:

- car - components to view, edit and select cars
- driver - component to view drivers
- general - global error handling component and interceptors
- map - two different map components
- menu - dispatching and driver menus
- order - dispatching and driver order managing components
- reusable - useful components used in different modules
- user - components concerning employees and authentication

Our customer PWA consists of these modules (with similar content):

- authentication
- general
- map
- order
- reusable

In figure 5.4 we show how is our project structured. We enclosed all components belonging to the module in its folder with each component in a separate folder. Services provided by the module are grouped in *shared* folder.

parent-child-binding

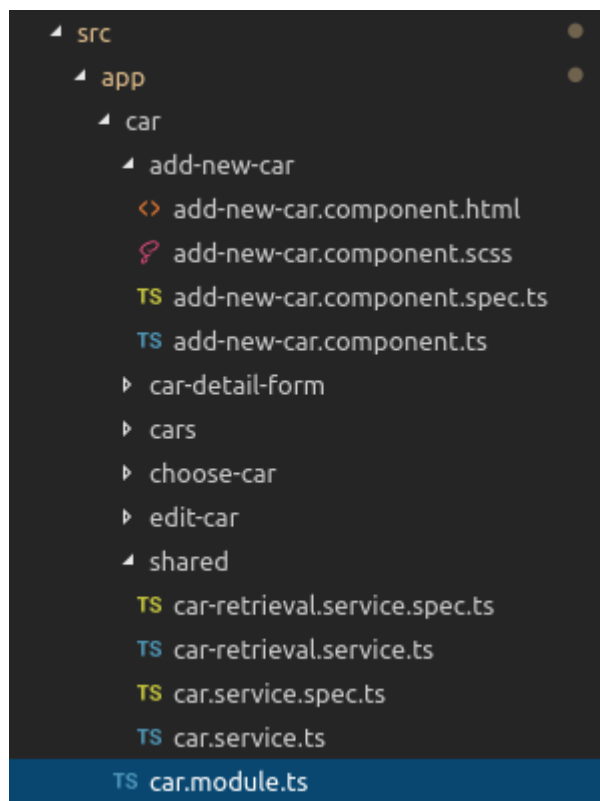


Figure 5.4: Module folder structure

6. Implementation

This chapter goes into details of implementation, describing problems we encountered and how we approached them. It gives an overview of all applications using screenshots and explains some key features in them.

6.1 Dispatching application

6.1.1 Login

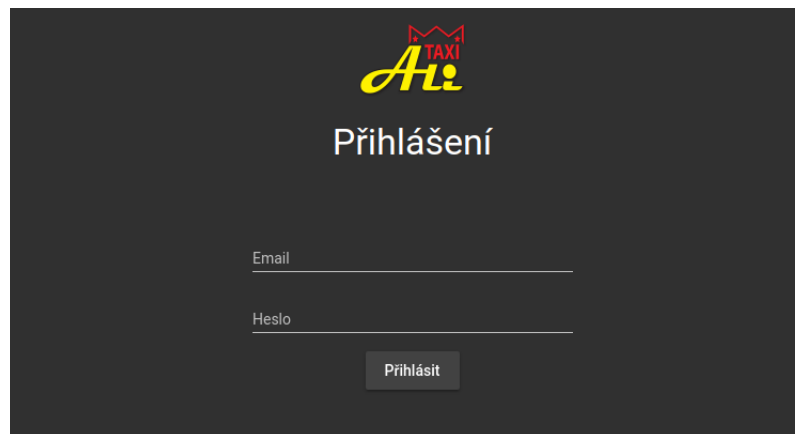


Figure 6.1: Dispatching and driver login

To enter the application, login is necessary. It is the same for dispatchers and drivers.

As shown in figure 6.1, user enters email and password which are then sent to server. If given credentials are correct, server responds with user's details and a token which is saved in local storage. It is later added into header of every outgoing request to authorize it. This is done using Angular's interceptors. They intercept each request and add the token if it's present.

In case user attempts to access other routes without logging in, router navigates them back to login screen. In case user is already logged in, they are routed to main menu automatically. To achieve this we implemented Angular's route guards which check whether user should be allowed to open requested url before it is loaded.

6.1.2 Create new order

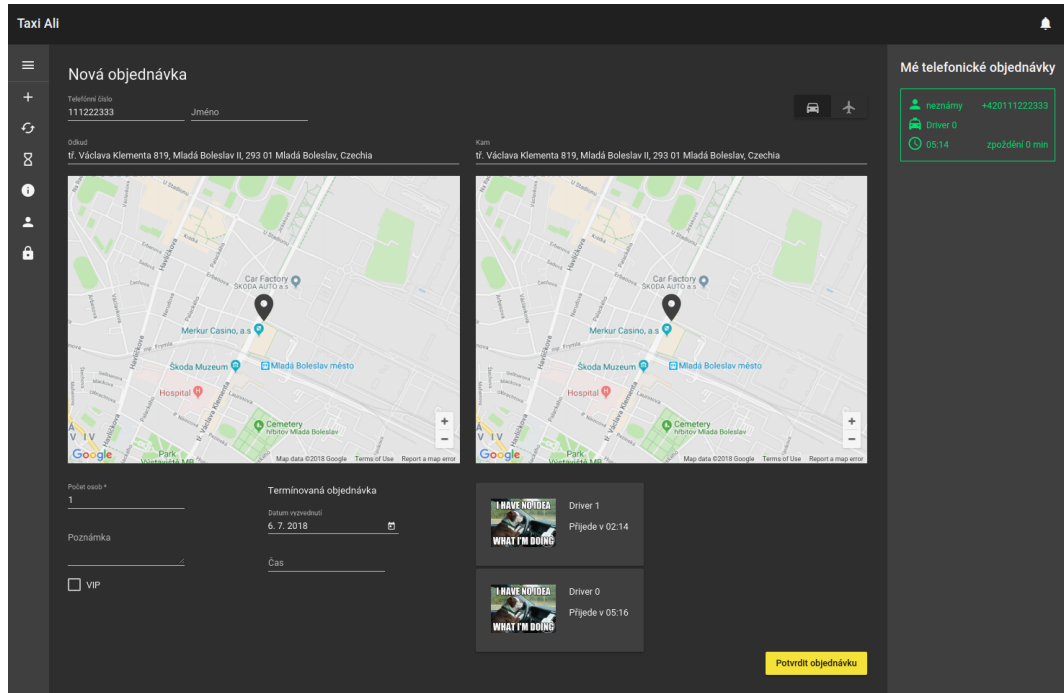


Figure 6.2: Dispatching - create new order

Figure 6.2 displays the standard screen that a dispatcher uses when they want to create new order while talking to a customer on the phone. They first copy customer's phone number into the input. Input dynamically validates it and sends a request to the server when a valid phone number is entered. If the customer had previously made an order using our system, their details (e.g. name) will be downloaded and filled in.

In the next step, dispatcher enters start and finish location using map or address input. On the map, marker is fixed in map's center while map can be dragged. On drag finish, address is filled in for the place that the marker is pointing to. When entering address using input, Google Places Autocomplete is used to hint addresses while typing (Love [2018]). Choosing places as well as autocomplete are limited to a larger rectangular area surrounding Mladá Boleslav.

Optionally, number of passengers can be changed, or a note or VIP can be added. A scheduled order can be created by filling in date and time.

After start and finish are chosen, dispatcher click on 'Select driver' which triggers a request to server where the arrival times for each driver are calculated. Dispatcher then selects driver according to customer's preference.

Only now can dispatcher 'Confirm order'. This triggers form validation. FormBuilder is used to create FormGroup from controls and add validators to them. Angular offers some basic validators like 'required', 'minLength', etc. For phone number, 'pattern' validator is used which validates it using regex. Inputs are validated dynamically - while user types, input turns red and shows error message when the input is invalid. These validators are all evaluated when the form is submitted. if it's invalid, faulty entries are highlighted and form does not get submitted.

When airport order is desired, dispatcher selects airport icon in upper right corner what triggers adding a flight number input and airport icons to start and finish. On selecting one of these icons, default airport address is filled in. Only one airport icon can be selected at a time. On form submission, custom validator checks whether exactly one airport icon is selected and only then allows the form to be submitted.

6.1.3 My phone orders

Orders that were created by dispatcher and are not yet finished are displayed on the right side (as shown in figure 6.2) at all times. They are polled from server every 10 seconds and driver, customer, and delay time are displayed. Order changes color when the delay increases above trigger amount to signal dispatcher that they should inform the customer about the delay.

6.1.4 Latest orders

řidič	Auto	Start	Cíl	Přidpokládaný čas naložení	Typ objednávky	Stav
Driver 0	Auto 0	—	—	05:16	📞	+ vytvořena

řidič	Auto	Start	Cíl	Přidpokládaný čas vyložení	Typ objednávky	Stav
Nenašli se žádné objednávky.						

řidič	Auto	Start	Cíl	Čas vyložení	Typ objednávky	Stav
Driver 0	Auto 1	Suite 966 294 Ketein Island, South Zetta, MA 63422	6132 Ara Junction, South Zinfurt, SD 42023	20:37	📞	✓ ukončena
Driver 4	Auto 1	Apt. 529 88553 Cole Track, Abernathystad, DE 73795	727 Ranall Forge, Peckshire, NJ 75184	21:18	📞	✓ ukončena
Driver 2	Auto 2	Suite 997 27882 Walter Dale, North Retahaven, NV 86923	1247 Medhurst Curve, Schowallertown, AL 27972	20:04	📞	✓ ukončena
Driver 2	Auto 2	727 Steuber Stravenus, South Rosalia, KY 42825	66907 Della Street, North Owenchester, NM 83115-0882	20:07	📞	✓ ukončena
Driver 1	Auto 0	6665 Anya Mountain, Trevashire, NH 69132-1092	13269 Rice River, Roobburgh, KY 89019	17:39	📞	✓ ukončena
Driver 11	Auto 0	794 Maeteri Vladud, North Domingoburgh, NV 68096	Apt. 723 651 Yolanda Mountans, New Karlians, MN 11175	04:19	📞	✓ ukončena
Driver 2	Auto 4	540 Madge Green, West Bemeice, LA 95498	8274 Terry Falls, West Juvenal, TN 95014-5195	06:31	📞	✓ ukončena
Driver 3	Auto 3	Apt. 584 4041 Hessel Club, East Aimee, MS 25804	Suite 468 674 Cade Plains, Pfanneratillmouth, NY 86795	17:59	📞	✓ ukončena
Driver 14	Auto 0	79112 Stephen Parkway, West Wellington, OK 28458	Apt. 422 66617 Boehm Squares, Port Verdsville, NJ 24734-2635	23:31	📞	✓ ukončena
Driver 3	Auto 4	30215 O'Hara Route, East Rebeka, NC 45647	7353 Johnathon Mountain, New Dillan, TX 57164-2369	18:42	📞	✓ ukončena

Figure 6.3: Dispatching - latest orders

Navigating to the next item on the menu, we can look at latest orders as shown in figure 6.3. Since the three tables we can see here have nearly all columns similar, we decided to create one component we can reuse for all tables. Which columns to display and what orders to load are the only parameters this component consumes. They are declared with @Input decorator inside the child component.

We used Angular Material Table to display orders in. It additionally supports pagination and number of items per page. Since it is a useful feature we decided

to implement it in communication with server.

6.1.5 Scheduled orders

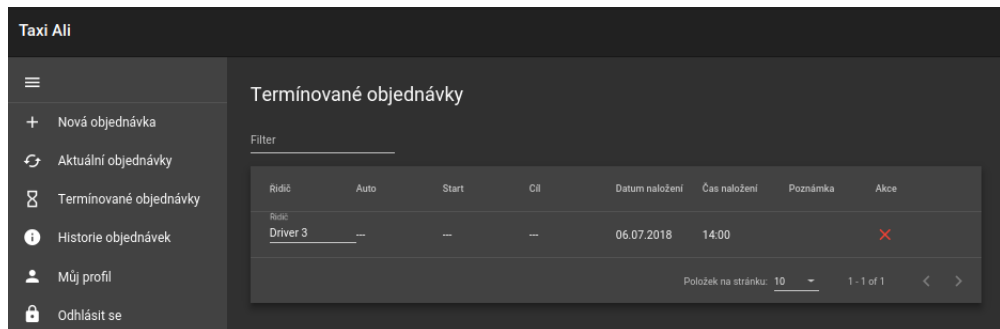


Figure 6.4: Dispatching - scheduled orders

Figure 6.4 additionally shows how the menu looks when users click on the menu icon in the upper left corner. The lowest icon triggers logout and deletes user's token from local storage.

This component loads data about scheduled orders from the server and displays it in a table. It also contains an actions column where a garbage icon can be clicked to remove an order. It will generate a pop-up dialog where dispatcher can confirm their choice to remove the order. In drivers column, driver can be selected or changed for the order using a select component. It communicates the change to the server using a service and selects them only if the request succeeds.

6.1.6 Notifications

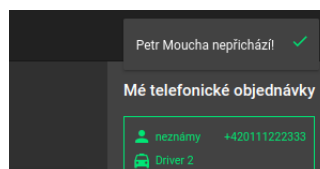


Figure 6.5: Dispatching - notifications

A notification icon is displayed in the upper right corner with a badge showing the number of unread notifications. When user clicks on it, a list of notifications pops up as illustrated in figure 6.5. Each notification describes the issue in a short message. A check button is present to mark the issue as resolved (after dispatcher contacts the customer, driver, etc.). After resolving, the corresponding notification disappears from the notification tray. Notification tray is updated in the background by pulling from the server every 10 seconds. It shows maximum of 100 last received notifications, including more is considered unnecessary.

6.1.7 Edit profile

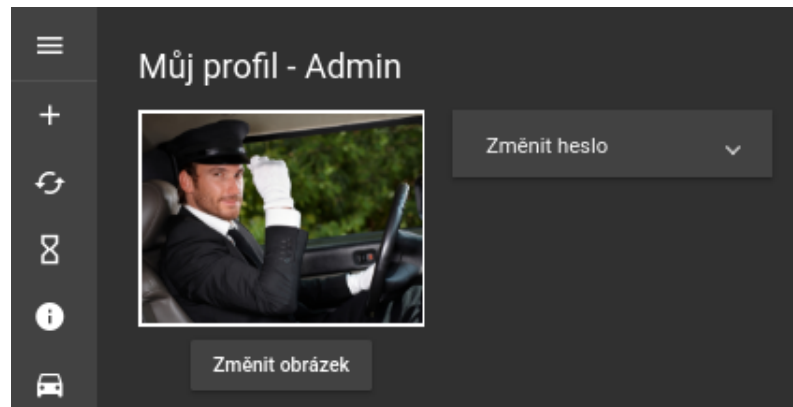


Figure 6.6: Dispatching - edit profile

Each employee can view their profile and edit it. To change their profile picture, an invisible input of type 'file' is used, triggered with the press of the button. To change their password the same form validation as mentioned before is used. Only this time a custom validator is added to validate two controls at once - it checks whether the two entered passwords match. It validates it dynamically so the user can see it while typing.

6.1.8 Cars

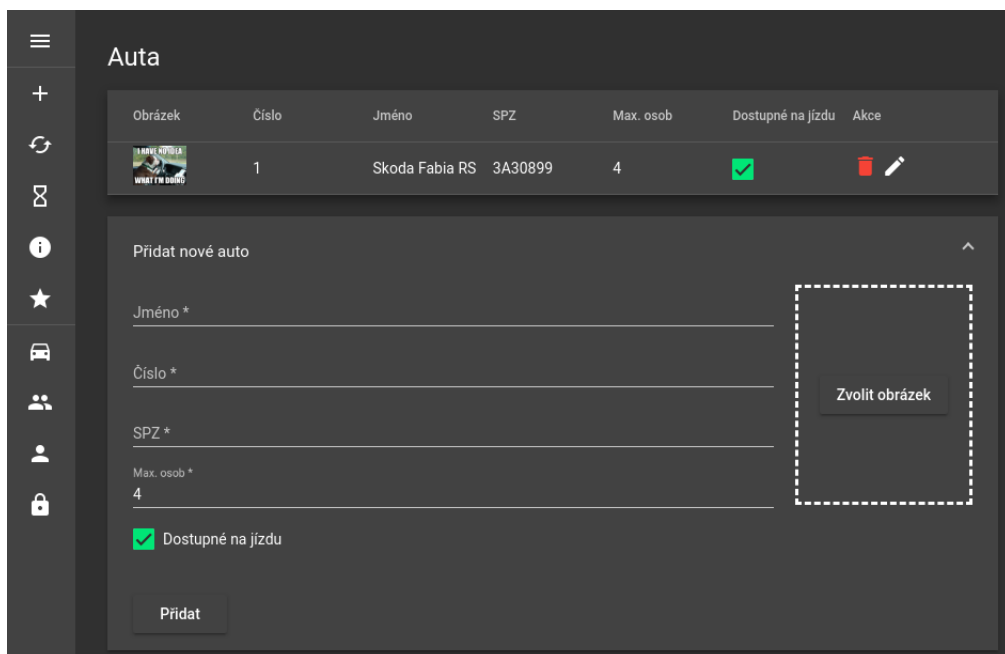


Figure 6.7: Admin - cars

This icon in the menu is available only for an administrator account. Cars are loaded to the table and additional actions can be performed. Administrator can delete a car what triggers a dialog to make sure they did not click it accidentally.

Car can be edited by clicking on the pencil icon or made unavailable for rides by unchecking the green checkbox.

A new car can be added in the following form. We created a reusable component to choose the picture. On selecting new picture, white rectangle is replaced with the picture. A red cross is added in the corner to delete the picture and choose a different one if needed. Form validates dynamically and on submission as described in previous screenshots. New car immediately appears in the table.

6.1.9 Edit car

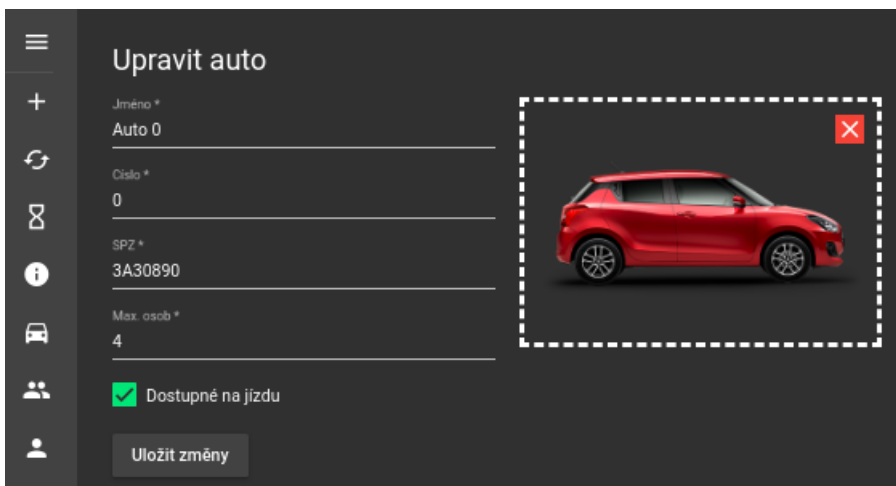


Figure 6.8: Admin - edit car

When user decides to edit car, a form prefills with current information about the car. User may edit it and then hit 'Save changes'. The form from 'add new car' is reused so the validation stays the same.

6.1.10 Employees

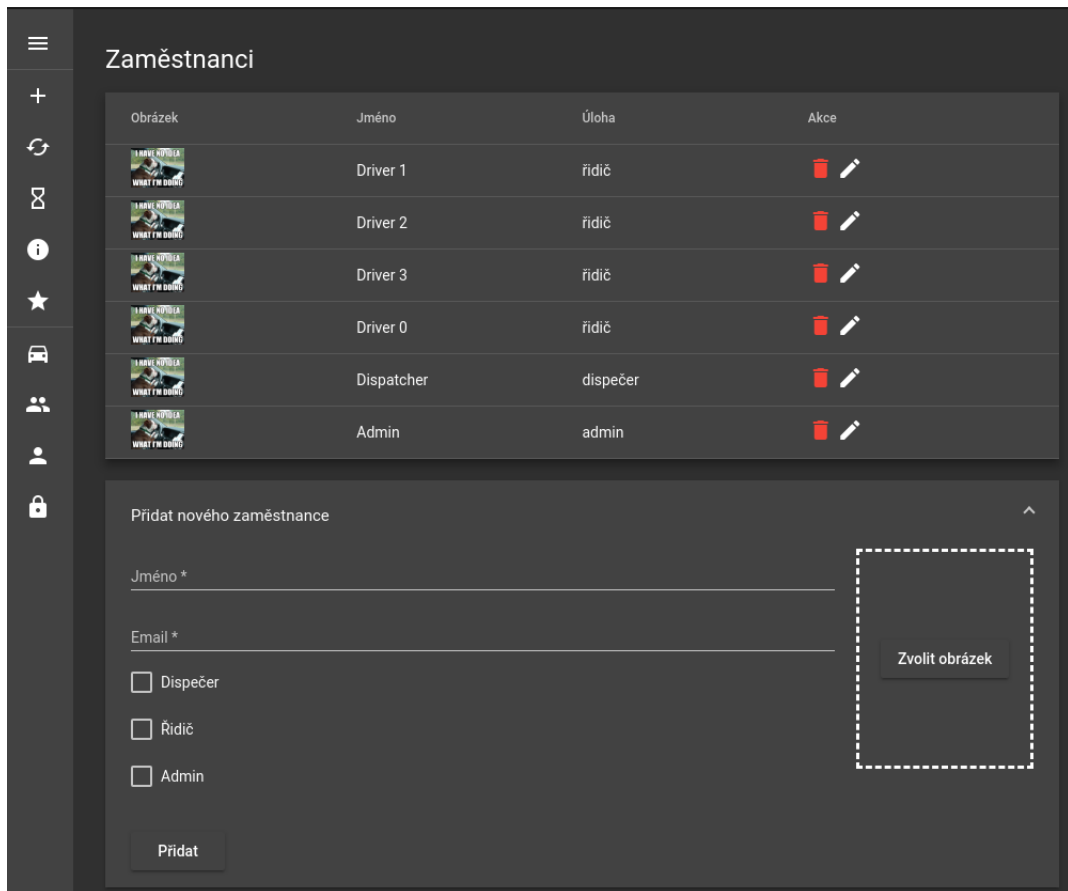


Figure 6.9: Admin - employees

This component is only available for administrator. It loads employees into the table and they can be deleted or edited similarly as in the car table. A new employee can be added and their roles need to be selected. Image selector is reused from previous components. Form is validated and a request with new employee is sent. Server sends an email with activation link where the new employee can set their password. Link contains a token to identify the user. After entering new password as shown in 6.10, we send a request with new password and this token to the server. Without completing this step, new employee cannot log in.

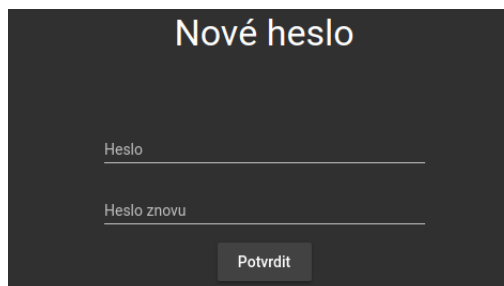


Figure 6.10: Dispatching - password confirmation

6.1.11 Employee profile

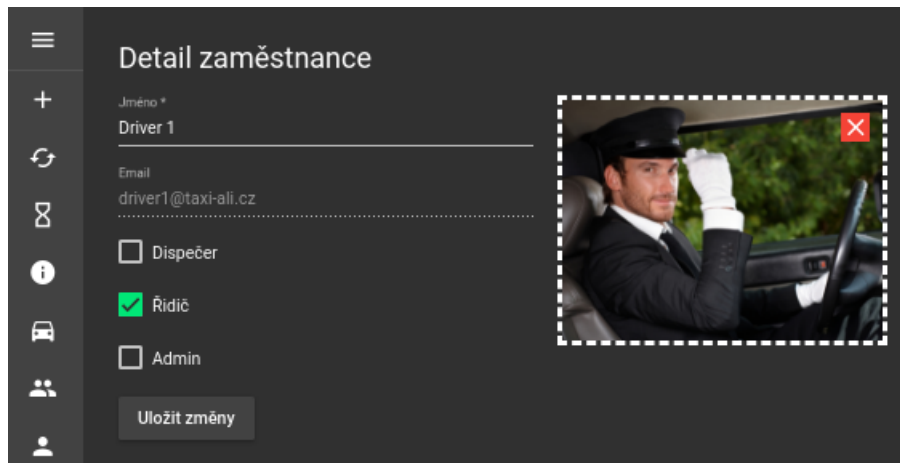
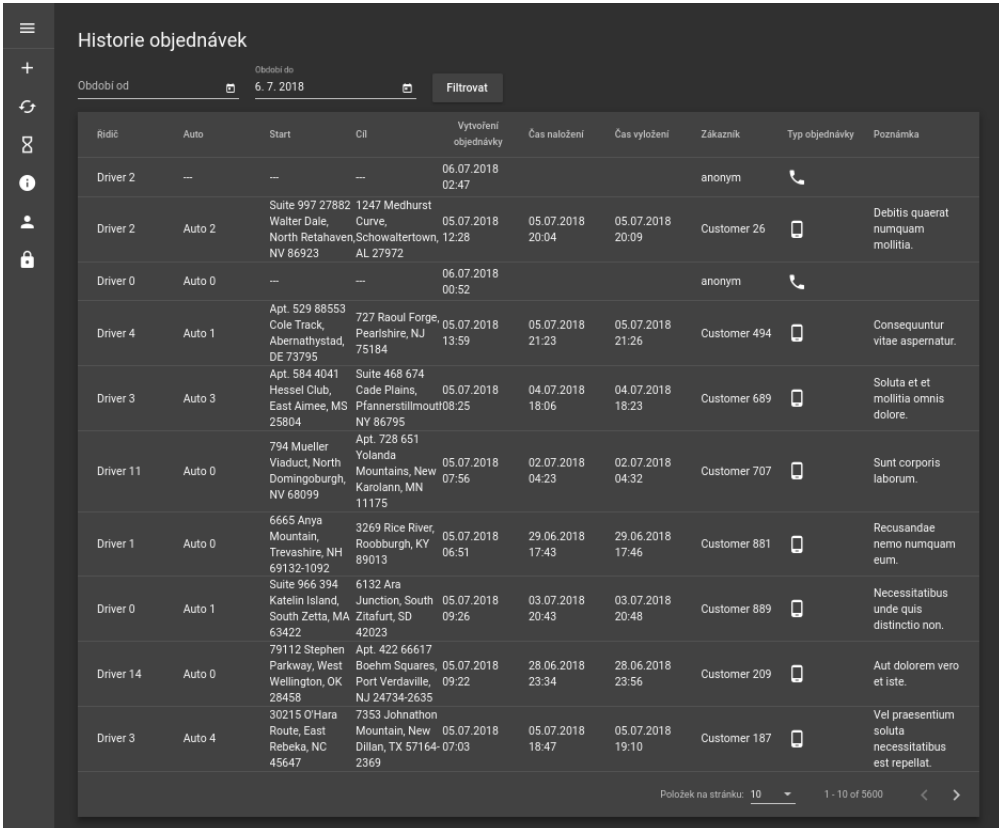


Figure 6.11: Admin - edit employee

This component provides equivalent functionality as edit car only for employees.

6.1.12 Order history



Ridič	Auto	Start	Cíl	Vytvoření objednávky	Čas naložení	Čas vyložení	Zákazník	Typ objednávky	Poznámka
Driver 2	--	--	--	06.07.2018 02:47			anonym	📞	
Driver 2	Auto 2	Suite 997 27882 Walter Dale, North Retahaven, NV 86923	1247 Medhurst Curve, Schowaltertown, AL 27972	05.07.2018 12:28	05.07.2018 20:04	05.07.2018 20:09	Customer 26	📱	Debitis queraet numquam mollitia.
Driver 0	Auto 0	--	--	06.07.2018 00:52			anonym	📞	
Driver 4	Auto 1	Apt. 529 88553 Cole Track, Abernathystad, DE 73795	727 Raoul Forge, Pearlishire, NJ 75184	05.07.2018 13:59	05.07.2018 21:23	05.07.2018 21:26	Customer 494	📱	Consequuntur vitae aspernatur.
Driver 3	Auto 3	Apt. 584 4041 Hessel Club, East Aimee, MS 25804	Suite 468 674 Cade Plains, Pfannerstillmoutt, NY 86795	05.07.2018 08:25	04.07.2018 18:06	04.07.2018 18:23	Customer 689	📱	Soluta et et mollitia omnis dolore.
Driver 11	Auto 0	794 Mueller Viaduct, North Domingobourgh, NV 68099	Apt. 728 651 Yolanda Mountains, New Karolann, MN 11175	05.07.2018 07:56	02.07.2018 04:23	02.07.2018 04:32	Customer 707	📱	Sunt corporis laborum.
Driver 1	Auto 0	6665 Anya Mountain, Trevashire, NH 69132-1092	3269 Rice River, Roobburgh, KY 89013	05.07.2018 06:51	29.06.2018 17:43	29.06.2018 17:46	Customer 881	📱	Recusandae nemo numquam eum.
Driver 0	Auto 1	Suite 966 394 Katelin Island, South Zetta, MA 63422	6132 Ara Junction, South Zitafurt, SD 42023	05.07.2018 09:26	03.07.2018 20:43	03.07.2018 20:48	Customer 889	📱	Necessitatibus unde quis distinctio non.
Driver 14	Auto 0	79112 Stephen Parkway, West Wellington, OK 28458	Apt. 422 66617 Boehm Squares, Port Verdaville, NJ 24734-2635	05.07.2018 09:22	28.06.2018 23:34	28.06.2018 23:56	Customer 209	📱	Aut dolore vero et iste.
Driver 3	Auto 4	30215 O'Hara Route, East Rebeka, NC 45647	7353 Johnathon Mountain, New Dillan, TX 57164-0703	05.07.2018 07:03	05.07.2018 18:47	05.07.2018 19:10	Customer 187	📱	Vel praesentium soluta necessitatibus est repellat.

Figure 6.12: Dispatching - order history

Order history implicitly loads the newest finished orders. A different time period can be selected using Angular Material Datepickers. A custom validator is implemented that does not allow to choose an older to-date than from-date. Datepicker contains a filter that does not allow user to select a day in the future.

6.2 Driver application

A basic overview of driver application and its routing is shown in figure 6.13.



Figure 6.13: Driver application lifecycle

6.2.1 Choose car

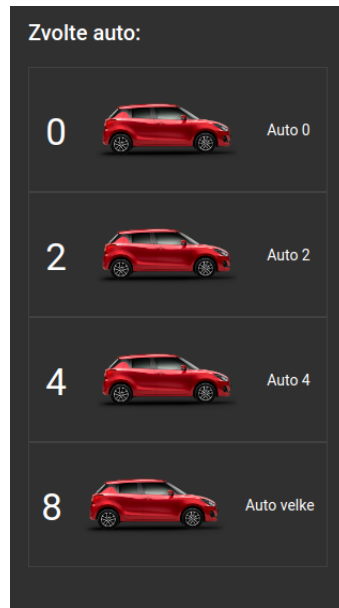


Figure 6.14: Driver - choose car

After logging in, driver needs to choose which car they will be driving. Available cars are pulled from the server and displayed. Driver can choose a car by clicking on it. Application then sends a request to start the shift with selected vehicle. Since the start of the shift, driver's application constantly sends driver's location coordinates to server. This allows customers to watch their driver.

6.2.2 No orders

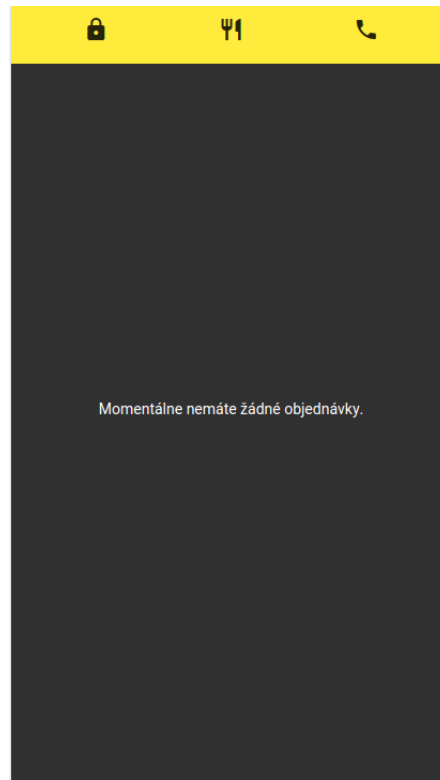


Figure 6.15: Driver - no orders

This is the default when no orders are available. In the quick menu in the toolbar, driver can log out, go on a break or call dispatching.

Logout should be used only when no orders are available - driver should plan a break ahead before logging out. However, driver can still log out in case of emergency situations and their orders will be redistributed to remaining drivers.

Going on a break is implemented as planning "break" as last order in the queue and not allowing anything else to be planned from that moment on. Driver's break starts when they finish last order in the queue.

6.2.3 New order

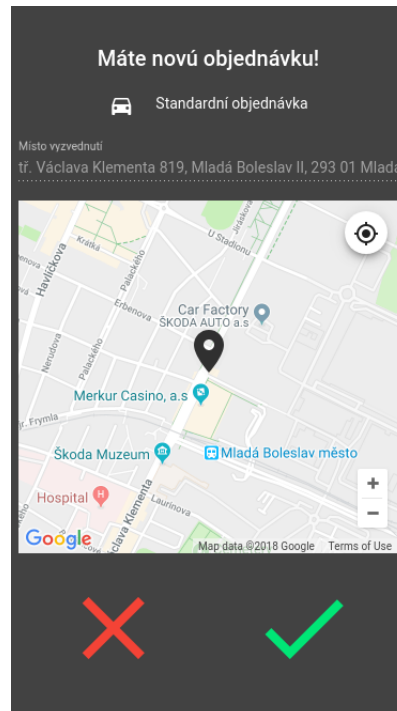


Figure 6.16: Driver - new order

When new order arrives, a notification is displayed in a dialog covering entire screen. Order details are shown and start location is displayed on a map using fixed marker. Driver has to accept or reject the order using simple icons.

6.2.4 Arriving

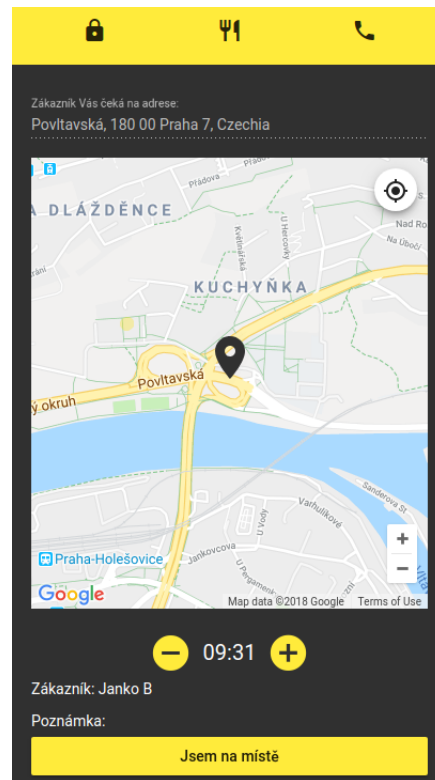


Figure 6.17: Driver - arriving

Driver starts the order on this screen. They can see the start location on a map and estimated arrival time. They can adjust this time using plus and minus buttons. These adjustments are immediately communicated to the server so the customer can see an updated arrival time. Driver has an option to focus the map on their location using the button in the upper right corner of the map. Upon arrival to start location, pressing the 'Arrived' button will send a request to the server which further notifies the customer about it.

6.2.5 Arrived

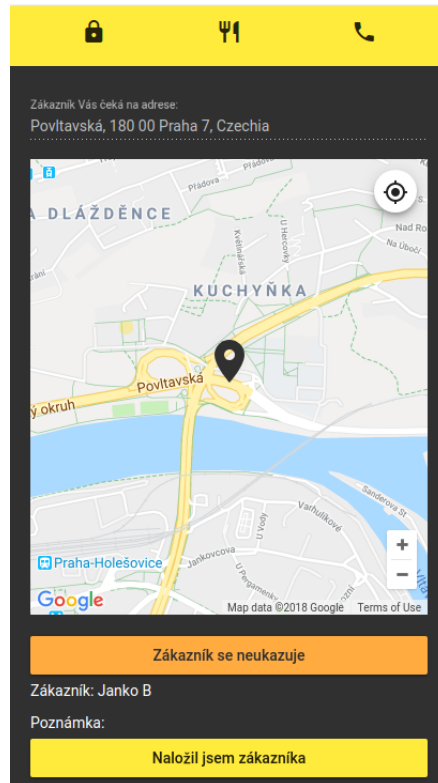


Figure 6.18: Driver - arrived

Since this view is very similar to the previous one, we reused the same component only displaying different controls using `*ngIf` and `*ngSwitch`. Driver can now signalize to server whether the customer was present on the start location or not. From this point on, customer cannot cancel the order.

6.2.6 Fraud

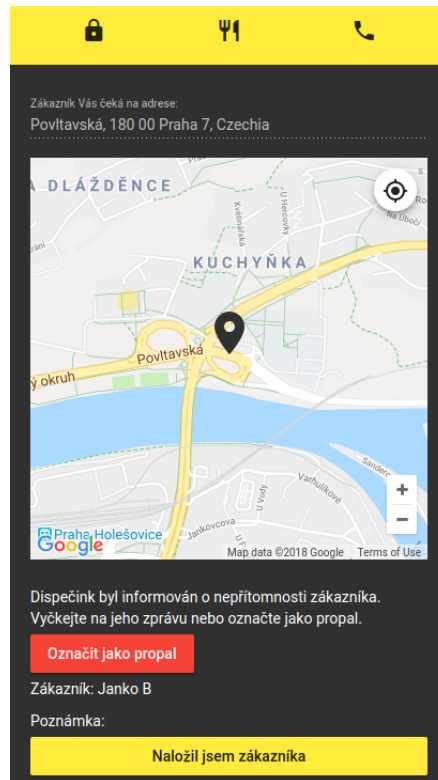


Figure 6.19: Driver - fraud

We again reused the same component. Driver can now continue to the next order by marking this one a fraud. Server will then inform the customer that their order was canceled due to their absence. If the customer does show up, order continues.

6.2.7 Picked up customer

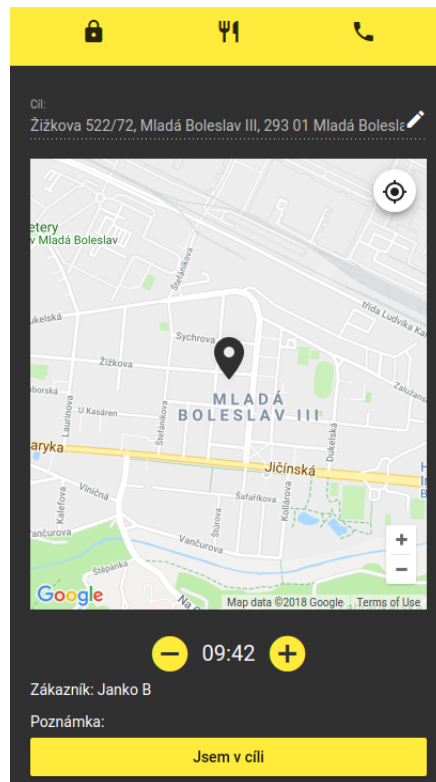


Figure 6.20: Driver - picked up customer

Again, we used the same component and even reused the time modifier component. Map component became editable, since finish may not be entered. When finish is known, it is marked on the map by a static marker. To change it, driver can tap on pencil icon, what causes address input to become editable. It also changes the static marker to a marker floating over the center of the map. Driver can now move the map to make the it point on the correct finish location. They can also change it in the address input, with help of autocomplete. When driver finishes editing, they need to confirm the position by clicking on the check icon. Check icon turns back into pencil icon and finish location is marked by a static marker.

We considered adding navigation for the driver but agreed it might become a disadvantage. If drivers would not be forced to remember the roads and plan their route, they would only simply drive on the route recommended by the application. We would therefore loose the human element and our drivers would no longer be considered the professionals that know all the shortcuts better than anyone.

If driver tries to finish the order before filling in the finish location, they will be shown an error message in a snackbar.

6.2.8 Enqueued orders

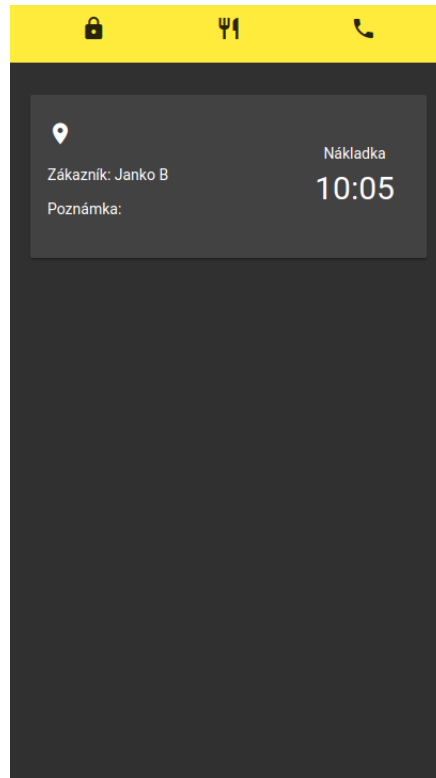


Figure 6.21: Driver - enqueued orders

Driver can view their other enqueued orders simply by scrolling down. Current order always covers the entire screen but the rest of the order queue is displayed under it. Order queue is pulled from the server every 30 seconds using a `TimerObservable`.

6.2.9 Break

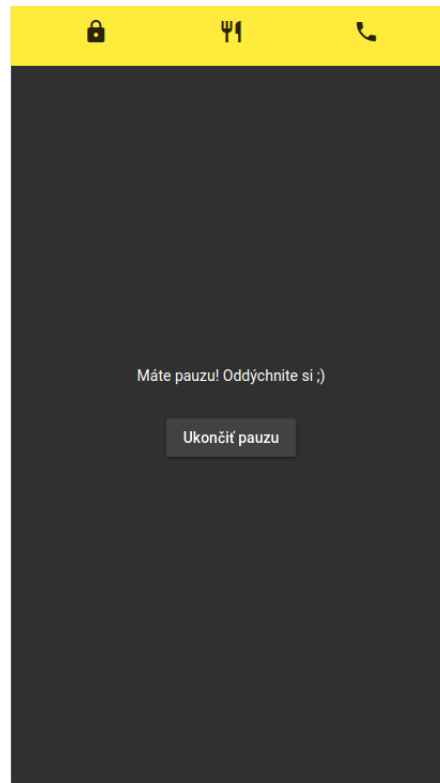


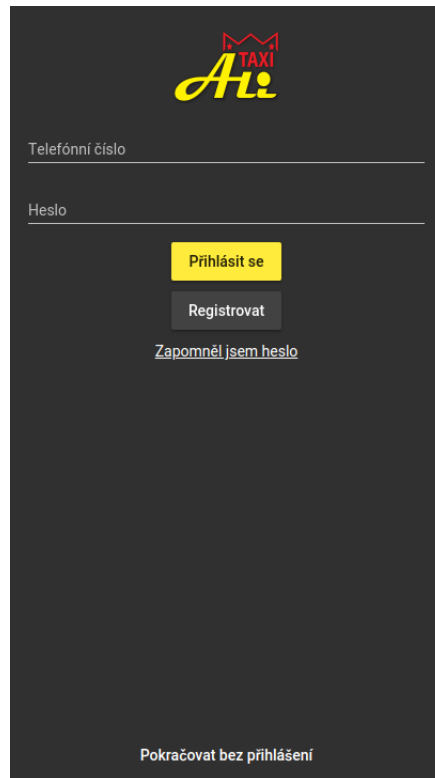
Figure 6.22: Driver - break

When driver planned a break and has no orders in the queue, this component shows up instead of 'no orders' component. However, they are both part of menu component, displayed using `*ngIf`.

6.3 Customer application

Following section describes customer application using screenshots.

6.3.1 Login

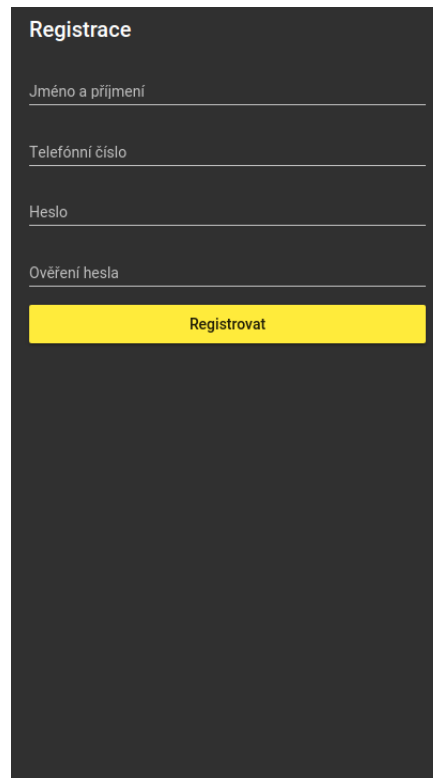


The screenshot shows a dark-themed login page for the 'Ai: TAXI' application. At the top center is the logo, which consists of the word 'Ai:' in a yellow, stylized font with a red outline, and the word 'TAXI' in a smaller, red, sans-serif font above it. Below the logo are two input fields: the first is labeled 'Telefónní číslo' (Phone number) and the second is labeled 'Heslo' (Password). Below these fields are three buttons: a yellow button labeled 'Přihlásit se' (Login), a grey button labeled 'Registrovat' (Register), and a link labeled 'Zapomněl jsem heslo' (Forgot my password). At the bottom of the page, there is a link labeled 'Pokračovat bez přihlášení' (Continue without login).

Figure 6.23: Customer - login

This login is similar to the login in previous application, the difference is that it validates for phone number and contains more router links.

6.3.2 Registration



The image shows a dark-themed registration form titled "Registrace". It contains four input fields with labels: "Jméno a příjmení", "Telefónní číslo", "Heslo", and "Ověření hesla". Below the fields is a prominent yellow button labeled "Registrovat".

Figure 6.24: Customer - registration

A registration form with validation. When valid values are sent to the server, it sends an SMS message to given phone number. We show a dialog window where this code can be entered or user can ask to send it again. If a valid code is entered, registration is confirmed and application routes to login.

6.3.3 Forgotten password

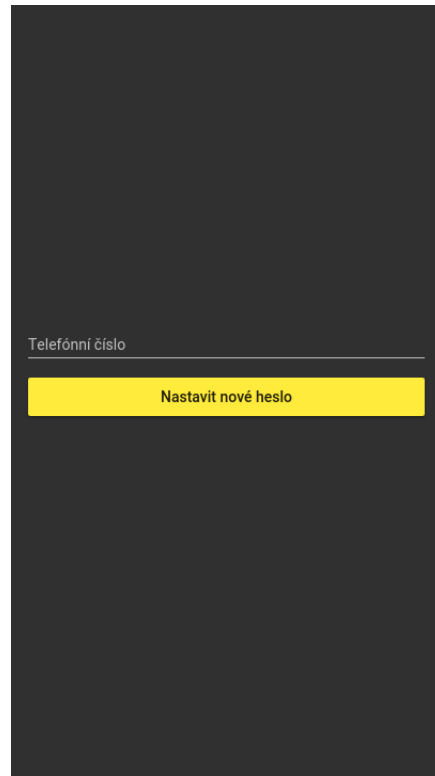


Figure 6.25: Customer - forgot password

Forgotten password behaves similarly to registration. It asks for phone number and receives confirmation code using SMS.

6.3.4 Start selection

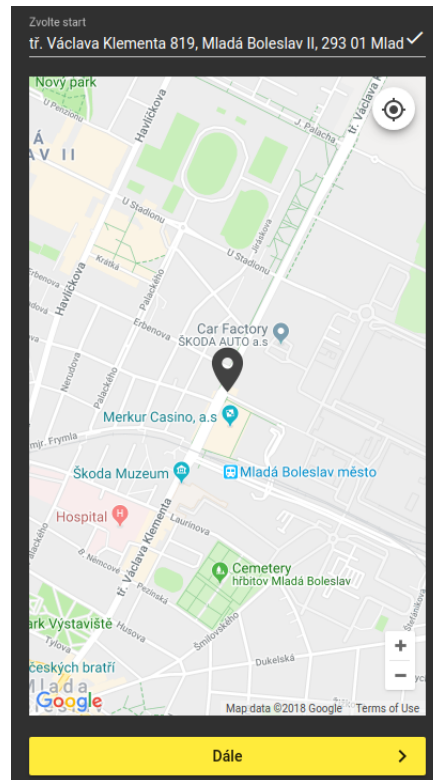


Figure 6.26: Customer - choose start

User is asked to select start on the map. Selecting reuses the map component from driver's application. It is reused again to select finish location.

6.3.5 Standard order confirmation

Driver 3 je u Vás za 21 minut.

Start
Dukelská 411/76, Mladá Boleslav III, 293 01 Mladá Boleslav

Cíl
třída Ludvika Kalmy a Volkharda Köhlera, 293 01 Mladá Boleslav

Počet osob
1 VIP

Telefónní číslo

Poznámka pro řidiče

Terminovaná

Odeslat

Figure 6.27: Customer - confirm standard order

A request is sent to server to calculate arrival times of drivers. The fastest driver is shown with their arrival time. User can change the driver by clicking on it. A list of drivers with their arrival times is shown and user can click on desired driver.

Selected start and finish address are viewed for the customer to review. Additional order details can be provided. When user toggles scheduled toggle, a date and time selection appears. Upon tapping 'Send' button, form is validated and order is sent only if it is valid.

6.3.6 Airport order confirmation

The screenshot shows a dark-themed mobile application form titled "Detaily objednávky". The form contains the following fields and controls:

- Počet osob:** A text input field containing the number "1".
- VIP:** A checkbox labeled "VIP" which is currently unchecked.
- Telefónní číslo:** A text input field containing the phone number "+420111222333".
- Číslo letu:** An empty text input field.
- Poznámka pro řidiče:** An empty text input field.
- Datum:** A date selection field.
- Čas:** A time selection field.
- Toggle:** A toggle switch labeled "Jdu z letiska" (left) and "Jdu na letisko" (right). The switch is currently positioned towards "Jdu z letiska".
- Submit:** A prominent yellow button at the bottom labeled "Zvolit cíl".

Figure 6.28: Customer - confirm airport order

Order details are the first step of airport order creation process. Since it is only available for registered users, phone number is pre-filled. Customer fills in flight number and date. Toggle indicates whether the airport is the start location or finish. According to toggle selection, next step is either choosing start or finish. Location selection is the same as in standard order. After location is selected, order is sent.

6.3.7 Driver's arrival

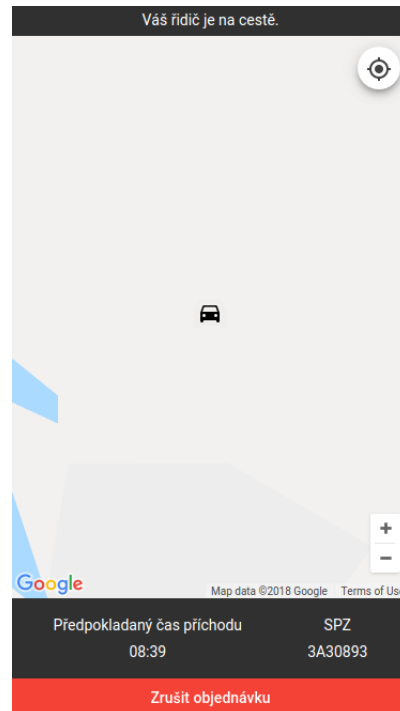


Figure 6.29: Customer - watch driver arrive

When driver starts this order, customer is shown a map with car icon representing the location of the driver. Driver's location is polled from the server as well as order details shown below it. Until driver arrives, customer can cancel the order.

When driver arrives, this screen transforms. Option to cancel the order disappears and information about arrival is shown.

When driver picks up the customer, estimated time to finish is show.

At the end, a thank note is shown to the customer.

6.3.8 Scheduled orders

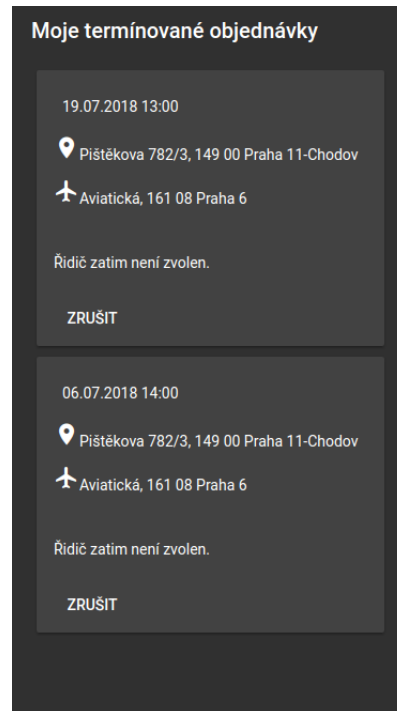


Figure 6.30: Customer - scheduled orders

Registered users can see an overview of their scheduled orders when they select it in the menu at order selection. Details of each order is shown along with an option to cancel the order. After the driver is selected it is shown to the user. When the driver start their shift, car is added too.

6.3.9 Reusing message component

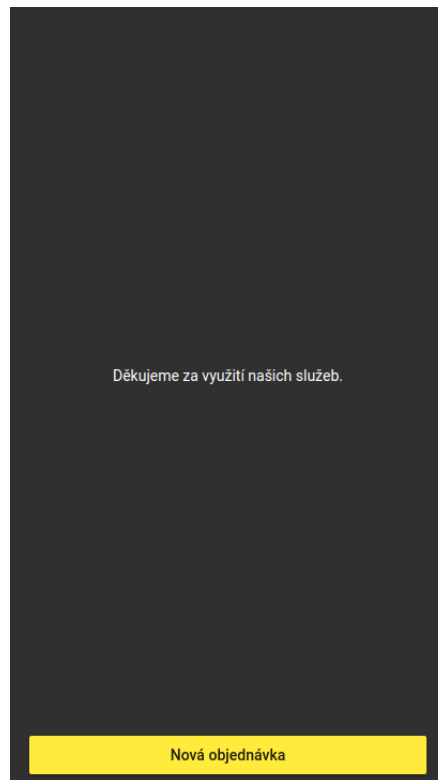


Figure 6.31: Customer - order finished

We created a simple component containing a message and a button. It is reused in customer application for various messages. It is implemented using parent-child component communication. It always resides in a parent that passes in the parameters and implements click events.

7. Evaluation

After implementing our applications, we need to test them and evaluate whether they fulfill our requirements.

7.1 Testing

We tested our applications on Chrome version 66.0.3359.170. PWA was additionally tested on Lenovo P70 with Android version 5.1.

To test our application we made use of detailed use cases we created in the beginning. We tested each application on all of its use cases to ensure it fulfills all the criteria and behaves as expected. Boundary values were tested during development and only randomly tested in the final testing since there is too many of them.

We tested our applications on a screen size it was designed to be used for. We executed main testing for mobile screens on iPhone 5 (in Chrome) since it is the smallest screen size available therefore we can expect more problems to occur there. Occasionally, we tried a different screen to make sure it behaves as expected too.

PWA was tested on a physical phone to ensure it can run independently as native mobile application, runs reasonably fast and caches screens as planned.

Backend provided a generator to create one year's order history with employees and customers which allowed us to test our application under more real conditions.

All our tests ensured that our application is ready for testing in real traffic and help taxi companies modernize their businesses.

7.2 Meeting goals

Looking back at the goals we set in the beginning, we can conclude they were all successfully met.

We believe we made a good choice picking Angular. It really seems to be a language that meets the developer half way and makes their job less painful. It also simplified our transition to PWA as it provided great build-in support for it and generated service workers needed.

Choosing PWA was a very good step towards having our applications cross-platform. Although it is only supported by Android at the time of writing, iOS claims to be working on implementing it and promises to release it soon. Many bigger companies like Aliexpress or Google Photos have already released a PWA version of their application. During the development process we encountered many expert opinions saying it is the future of applications. Additionally, it is responsible for a much better performance of our customer application thanks to caching and pre-loading.

We successfully styled our applications to fit the desired screen sizes. We tried to add scaling and wrapping of components for a smaller screen everywhere we thought it might be needed.

Our driver application successfully relieves driver of all typing. The only typing element is map's address input which can be fully replaced using map gestures. Nevertheless, if used, it is most probably at a time when vehicle is stopped - when driver picks up customer or finishes the order.

We humbly consider our final code to be well structured. Once again we believe it is mainly thanks to Angular, TypeScript and their detailed documentation and best practices. We successfully created multiple larger reusable components controlled by parameters.

Conclusion

We successfully implemented a complete taxi system front-end with three ready-to-use applications for automation of ordering process. All required functionality was accomplished using the newest technologies. We believe our application will serve well in real traffic providing customers with intuitive and comfortable experience. We are looking forward to improve it further based on user reviews after we set it up for local taxi company.

Future

There is certainly a lot of space for extensions and new features to add to our applications. Some ideas we seriously consider for the future are:

- Login with Facebook or Google
- New types of orders
- Order reviews/evaluation
- Using machine learning to calculate better transfer times

Although Angular tests were not a part of our implementation, test files were added to all components. Optionally, tests can be added if needed for all applications.

There might be a need to translate our customer application to different language so that even foreign customers can use our services. Angular offers Internationalization tools to make translations easier. However, so far internationalization is only implemented for messages in HTML templates therefore it is not possible to use to translate error messages shown at runtime. Angular team promises to add this feature to the next release. Therefore we implemented internationalization for template messages so far. When the feature is released, error messages can be translated and our PWA can become fully international.

Bibliography

Angular, June 2018a. URL <https://angular.io/>.

Angular Material, 2018b. URL <https://material.angular.io/>.

Icons - Material Design, 2018a. URL <https://material.io/tools/icons/?style=baseline>.

Maps JavaScript API, 2018b. URL <https://developers.google.com/maps/documentation/javascript/tutorial>.

How does uber work, 2018. URL <https://www.uber.com/en-CZ/ride/>.

Cuelogic Insights. Angular vs React vs Vue: A 2018 comparison, January 2018. URL <http://www.cuelogic.com/blog/angular-vs-react-vs-vue-a-2018-comparison/>.

Brian Love. Angular 2+ google maps places autocomplete, April 2018. URL <http://brianflove.com/2016/10/18/angular-2-google-maps-places-autocomplete/>.

Jens Neuhaus. Angular vs. React vs. Vue: A 2017 comparison, August 2017. URL <https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176>.

List of Figures

2.1	Order status	8
5.1	Component two-way data binding	23
5.2	Parent-child binding	23
5.3	Component tree	23
5.4	Module folder structure	25
6.1	Dispatching and driver login	26
6.2	Dispatching - create new order	27
6.3	Dispatching - latest orders	28
6.4	Dispatching - scheduled orders	29
6.5	Dispatching - notifications	29
6.6	Dispatching - edit profile	30
6.7	Admin - cars	30
6.8	Admin - edit car	31
6.9	Admin - employees	32
6.10	Dispatching - password confirmation	32
6.11	Admin - edit employee	33
6.12	Dispatching - order history	34
6.13	Driver application lifecycle	35
6.14	Driver - choose car	36
6.15	Driver - no orders	37
6.16	Driver - new order	38
6.17	Driver - arriving	39
6.18	Driver - arrived	40
6.19	Driver - fraud	41
6.20	Driver - picked up customer	42
6.21	Driver - enqueued orders	43
6.22	Driver - break	44
6.23	Customer - login	45
6.24	Customer - registration	46
6.25	Customer - forgot password	47
6.26	Customer - choose start	48
6.27	Customer - confirm standard order	49
6.28	Customer - confirm airport order	50
6.29	Customer - watch driver arrive	51
6.30	Customer - scheduled orders	52
6.31	Customer - order finished	53

List of Abbreviations

AoT Ahead of Time compilation
API Application programming interface
CLI Command line interface
CSS Cascading style sheets
DOM Document Object Model
HTML Hyper-text markup language
ID unique identification number
PWA Progressive web application
REST Representational state transfer
TS Typescript
UI User Interface
URL Uniform Resource Locator

A. Appendix

In the following part we describe how to build and run the source code submitted. Source code contains two folders:

- customer-pwa: PWA application for customers
- web-dispatching-drivers: driver and dispatching web application

To run these applications, following steps are necessary:

1. Install Node.js version 8.x or greater.
2. Install npm version 5.x or greater.
3. Run following command to install Angular CLI globally.

```
npm install -g @angular/cli
```

4. Navigate to application's root folder and run following command to download dependencies.

```
npm install
```

5. Run following command to start the application.

```
ng serve --open
```

Using option `--open` will open browser on `http://localhost:4200/`. To start application on a different port, use option `--port 4201`.

These applications communicate with backend which is not a part of this thesis. Its source files are located on a private bitbucket account. To receive access, please send an email to *lukas@brezinovi.sk*.

To run backend take following steps:

1. Install Docker CE. (<https://docs.docker.com/install/linux/docker-ce/ubuntu/#set-up-the-repository>)
2. Install docker-compose. (<https://docs.docker.com/compose/install/>)
3. Navigate to backend folder containing file `docker-compose.yml`.
4. Add following to `/etc/hosts` (or its alternatives on other operating systems).

```
127.0.0.1      taxiali.local
127.0.0.1      api.taxiali.local
```

5. Run following commands:


```
docker-compose build
docker-compose run api-development rake db:init
docker-compose run api-development rake db:migrate
docker-compose run api-development rake
cml:generator:generate\_1\_year\_data
```

6. Start by running:

```
docker-compose up
```

Since SMS with verification codes were not yet implemented by backend, verification codes can be viewed using command

```
docker attach server-background\_1\_api-development\_1.
```