

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Mobilná aplikácia - Autoservis

Mobilné technológie a aplikácie

Autori: Matej Lánik, Jakub Sorád

Cvičenia: Štvrtok 8:00

Prednášajúci: doc. Ing. Peter Trúchly, PhD.

Akademický rok: 2021/22

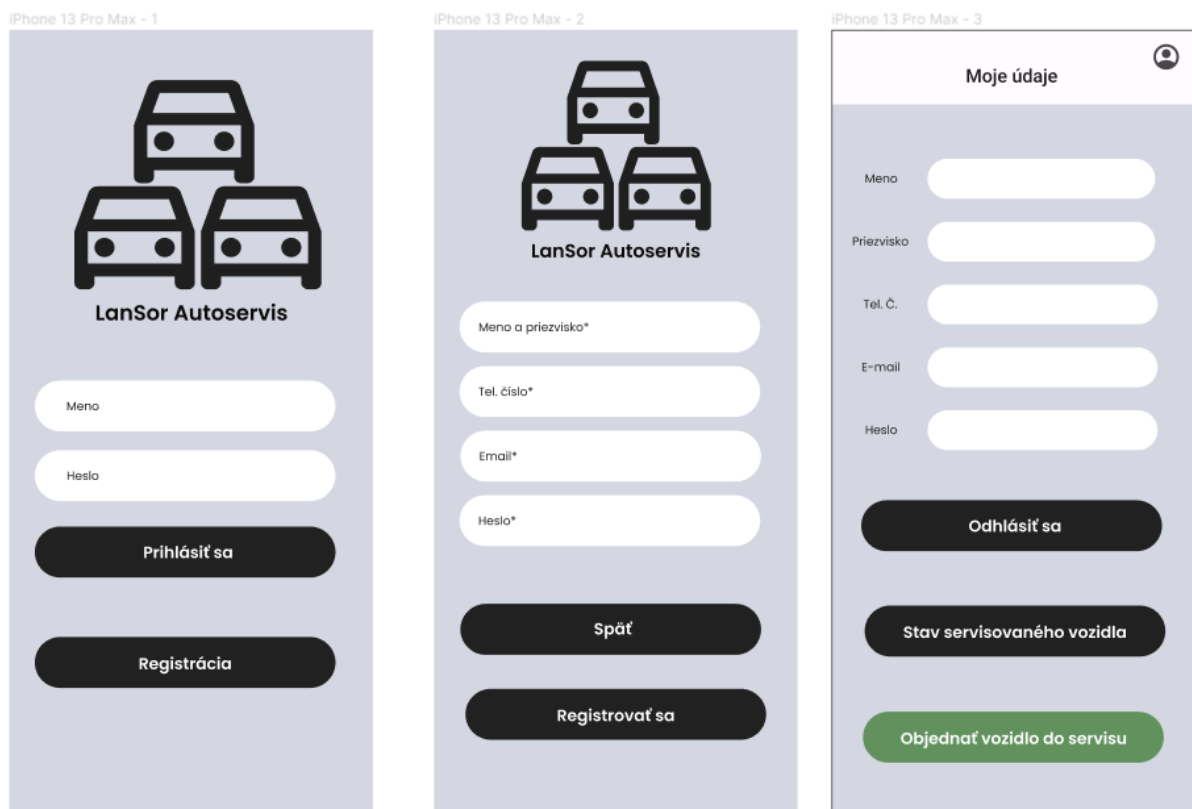
Obsah

Návrh wireframov	2
Návrh databázy	5
Návrh API endpointov	6
Akceptačné testy	8
Frontend aplikácie	8
Kladné testy	8
Záporné testy	11
Backend	13
Kladné testy	13
Záporné testy	15
Backend aplikácie	17
Backend technológie	17
Štruktúra backendu a jeho funkcionalita	17
Testovanie backendu	18
Change log oproti Milestone 1	19
Frontend aplikácie	20
Použité frontend technológie	20
Štruktúra frontendu a jeho funkcionalita	20
Spustenie mobilnej aplikácie a testovanie	22
Zhodnotenie frontendu	23
Change log oproti Milestone 2	25
Doplnková úloha – Websokety	26
Backend aplikácie	26
Frontend aplikácie	28
Zhodnotenie doplnkovej úlohy	30

Návrh wireframov

Použili sme nástroj [Figma](#), kde sme vytvorili jednotlivé wireframy a tiež aj user flows, takže náš návrh je aj interaktívny. Po konzultácií sme zapracovali na pripomienkach a možných zlepšeniach, medzi ne patria aj tieto dve:

- Pri scenári, keď sa prihlasuje technik sme použili inú úvodnú obrazovku, ale dopredu nevieme, kto sa bude prihlasovať, či zákazník alebo technik. Preto bude platiť pre oboch typoch používateľoch jednotná prihlasovacia obrazovka.
- Keď technik opraví auto, potvrdí to v aplikácii a následne aj zákazník potvrdí opravu auta, tak sme vymazali dané auto z databázy. V našej aplikácii by bolo veľmi vhodné si práve tieto opravené autá ukladať do histórie opravených áut. Túto históriu si potom technik môže pozrieť po prihlásení do aplikácie.



iPhone 13 Pro Max - 4

Objednať vozidlo do servisu

Značka*

Model*

Rok výroby*

☐ Želám si aby servis môjho vozidla vykonával konkrétny technik.

Späť

iPhone 13 Pro Max - 7

Zvoľte požadované úkony

☒ Výmena motorového oleja
☒ Výmena filtrov
☒ Výmena pneumatík
☒ Servis motora

Doplňujúce informácie...

Späť

Nahrať fotografie vozidla

iPhone 13 Pro Max - 8

Nahrať fotografiu vozidla

Výbrať obrázok

Späť

Objednať vozidlo do servisu

iPhone 13 Pro Max - 11

Stav vozidla

Značka

Model

Pridelený technik

Servis vozidla bol dokončený
Potvrdiť vyzdvihnutie vozidla

iPhone 13 Pro Max - 6

Stav vozidla

Značka

Model


Pridelený technik

Stav vozidla

Začať videohovor s technikom

Pokračovať

login

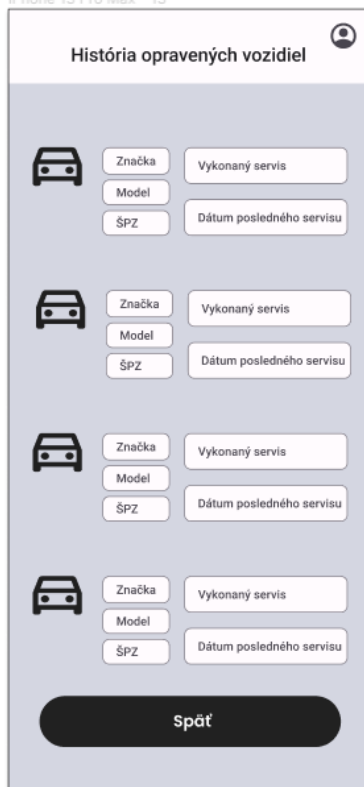

LanSor Autoservis

Meno

Heslo

Prihlásiť sa

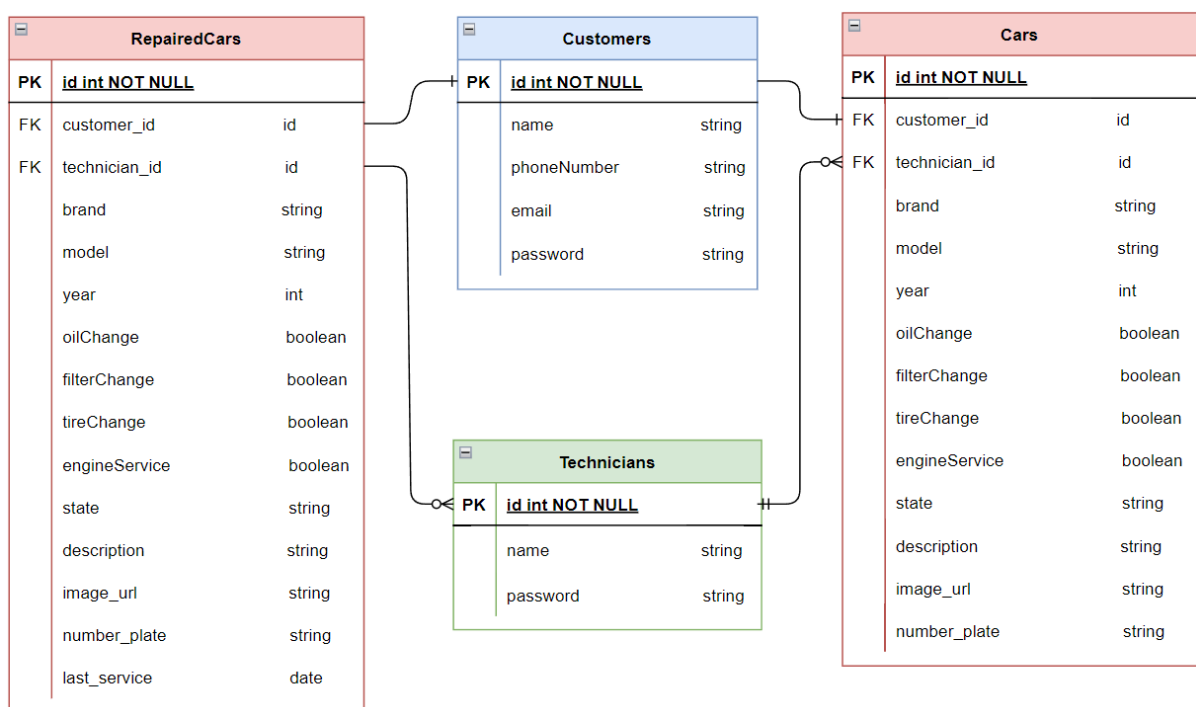
Registrácia



Návrh databázy

Súčasťou projektu mobilnej aplikácie je návrh fyzického modelu databázy, ktorú použijeme. Na vytvorenie návrhu sme použili nástroj draw.io, kde sme jednotlivé schémy vedeli navrhnuť a tiež aj vzťahy medzi nimi.

Po zapracovaní na pripomienke ohľadom histórie opravených áut sme upravili náš databázový model. Použijeme ďalšiu schému, ktorá slúži na uloženie už opravených áut. Bude tam aj pridaný údaj kedy prebehol posledný servis alebo oprava auta, čo môže pomôcť technikovi pri riešení problémov s autom. Taktiež prebehli aj drobné úpravy v jednotlivých schémach, kde u zákazníka ani technika nemali uložené heslo pre prístup od aplikácie. Ďalej sme pridalí evidenčné číslo pre každé auto.



Návrh API endpointov

Použili sme nástroj [Swagger](#), pre správu API dokumentácií. Poskytuje množstvo funkcionalít, ktoré nám umožnia v prehľadnej forme poskytovať údaje a detaily o API volaniach a endpointoch, ktoré budú súčasťou nášho projektu.

Prihlásenie používateľa		^
POST	/login Skontroluje zadané používateľské údaje	▼
Zoznam používateľov		^
GET	/customers Vráti zoznam všetkých používateľov.	▼
Vkladanie používateľa do DB		^
POST	/customers Vytvorí nový záznam používateľa v databáze	▼
Zobrazenie informácií o používateli		^
GET	/customers/{id} Vráti informácie o používateli	▼
Zoznam vozidiel		^
GET	/cars Vráti zoznam všetkých vozidiel	▼
Vloženie auta do DB		^
POST	/cars Vytvorí nový záznam používateľa v databáze	▼
Zoznam opravených vozidiel		^
GET	/repairedCars Vráti zoznam všetkých vozidiel	▼
Vloženie auta do histórie DB		^
POST	/repairedCars Vytvorí nový záznam používateľa v databáze	▼
Zobrazenie informácií o vozidle		^
GET	/cars/{id} Vráti údaje o vozidle	▼
Odstránenie vozidla z DB		^
DELETE	/cars/{id} Odstráni vozidlo z DB	▼
Úprava stavu vozidla		^
PUT	/cars/{id} Upraví údaje o Vozidle	▼
Zobrazenie technických vozidiel		^
GET	/technicianCars/{id} Vráti všetky vozidlá na ktorých pracuje konkrétny technik	▼
Zobrazenie informácií o zákazníkovi vozidle		^
GET	/customerCar/{id} Vráti údaje o vozidle podľa prislúchajúceho zákazníka	▼
Zoznam technikov		^
GET	/technicians Vráti zoznam všetkých technikov	▼
Zobrazenie informácií o technikovi		^
GET	/technicians/{id} Vráti údaje o technikovi	▼

Každý z API endpointov obsahuje popis funkcionality, detaily parametrov a odpovedí. Na spodnom obrázku je príklad detailov po rozkliknutí endpointu.

Zobrazenie informácií o vozidle

GET `/cars/{id}` Vráti údaje o vozidle

Toto volanie využívame na získanie informácií o aute, na ktorom sa práve vykonáva servis.

Parameters Try it out

Name	Description
id * <i>required</i>	unikátny identifikátor vozidla
number (path)	<input type="text" value="id"/>

Responses

Code	Description	Links
200	Informácie o vozidle	No links
<div>Media type <div>application/json</div><div>Controls Accept Header.</div><div>Example Value Schema</div><pre>{ "id": 1, "customer_id": 1, "technician_id": 1, "brand": "BMW", "model": "X3", "year": 2011, "oilChange": true, "filterChange": true, "tyreChange": true, "engineService": true, "state": "not_finished", "description": "veľmi pokazene", "image_url": "obrazok.jpg", "number_plate": "TT816DX" }</pre></div>		
404	Vozidlo neexistuje	No links
<div>Media type <div>application/json</div><div>Example Value Schema</div><pre>{ "message": "Vozidlo neexistuje." }</pre></div>		

Akceptačné testy

Vytvorili sme dokopy 10 akceptačných testov pre frontendovú časť aplikácie a tiež pre backend aplikácie. Rozdelili sme testy na dva typy, kedy správanie používateľa je kladné a kedy je záporné. Podľa toho dostávame rôzne výstupy pri interakcii s našou aplikáciou.

Frontend aplikácie

Kladné testy

Test 1: Prihlásenie sa	
Vstupné podmienky:	Používateľ, ktorý má vytvorený účet a nachádza sa v databáze.
Výstupné podmienky:	Používateľ sa úspešne prihlási do svojho účtu.
Postup:	<ol style="list-style-type: none">1. Používateľ zadá svoje prihlasovacie údaje.2. Používateľ stlačí tlačidlo PRIHLÁSIŤ SA.3. Používateľovi sa zobrazí obrazovka s jeho údajmi, údajmi o aute v servise a možnosťou objednania sa do servisu.
Výsledok: PASS / FAIL	

Test 2: Vyplnenie požadovaných servisných úkonov.	
Vstupné podmienky:	Používateľ, ktorý zadal základné informácie o jeho aute a stlačil tlačidlo POKRAČOVAŤ.
Výstupné podmienky:	Zákazník vyplnil formulár s požiadavkami o servise a prechádza na posledný krok objednávky do servisu.
Postup:	<ol style="list-style-type: none"> 1. Používateľ nastaví togglebary všetkých požadovaných úkonov. 2. Ak používateľ má ďalšie servisné požiadavky, napíše ich do poľa DOPLŇUJÚCE INFORMÁCIE. 3. Po vyplnení všetkých požadovaných servisných úkonov klikne používateľ na tlačidlo NAHRAŤ FOTOGRAFIE VOZIDLA. 4. Používateľovi sa zobrazí obrazovka, v ktorej musí nahrat' fotografiu/fotografie vozidla.
Výsledok: PASS / FAIL	

Test 3: Vloženie fotografií servisovaného vozidla	
Vstupné podmienky:	Používateľ, ktorý vyplnil všetky potrebné informácie o servisnom úkone.
Výstupné podmienky:	Používateľove auto bude zaradené do databázy. Servisovanie auta následne bude pridelené požadovanému technikovi(ak si takého používateľ zvolil) a zákazník si bude môcť pozrieť stav opravy vozidla.
Postup:	<ol style="list-style-type: none"> 1. Používateľ klikne na tlačidlo VYBRAŤ OBRÁZOK. 2. Používateľ vloží fotografiu/fotografie jeho vozidla. 3. Používateľovi sa zobrazia vložená fotografia na obrazovke. 4. Používateľ klikne na OBJEDNAŤ VOZIDLO DO SERVISU. 5. Používateľovi je zobrazená začiatočná obrazovka s jeho údajmi, kde už môže skontrolovať stav jeho servisovaného vozidla.
Výsledok: PASS / FAIL	

Záporné testy

Test 4: Vytvorenie nového účtu	
Vstupné podmienky:	Používateľ je na hlavnej stránke a chce sa registrovať
Výstupné podmienky:	Používateľ zadal nesprávne registračné údaje a je upozornený.
Postup:	<ol style="list-style-type: none">1. Používateľ stlačí tlačidlo REGISTRÁCIA.2. Zobrazí sa registračný formulár.3. Používateľ vyplní všetky potrebné polia, no zadá tel.č. alebo e-mail, ktorý už existuje v databáze.4. Používateľ klikne na tlačidlo REGISTROVAŤ.5. Používateľ bude upozornený, že tento účet už existuje.6. Používateľovi sa objaví upozornenie a musí skontrolovať vstupné údaje.
Výsledok: PASS / FAIL	

Test 5: Vyplnenie základných informácií o vozidle	
Vstupné podmienky:	Používateľ, ktorý je registrovaný a chce objednať svoje auto do servisu.
Výstupné podmienky:	Zákazník úspešne informácie o jeho vozidle, a prejde na druhý krok objednávky do servisu. V prípade nevyplnenia jedného z povinných polí je používateľ upozornený.
Postup:	<ol style="list-style-type: none"> 1. Používateľ stlačí tlačidlo OBJEDNAŤ VOZIDLO DO SERVISU. 2. Používateľovi sa zobrazí formulár, v ktorom musí vyplniť informácie o jeho aute. 3. Používateľ vyplní formulár, s tým že zabudne zadať niektoré z povinných polí. 4. Ak používateľ nemá požiadavku vybrať si konkrétneho technika, preskočiť na krok 7. 5. Ak si používateľ žiada konkrétneho technika, musí zaškrtnúť checkbox. 6. Zákazník si zo zobrazeného dropdown menu zvolí preferovaného technika, ktorému bude servis pridelený. 7. Zákazník klikne na tlačidlo POKRAČOVAŤ. 8. Zákazník bude upozornený, že zabudol vyplniť všetky povinné údaje. 9. Zákazník po upozornení vyplní potrebné údaje, preskočiť na krok 7.
Výsledok: PASS / FAIL	

Backend

Kladné testy

Test 6: Správna registrácia	
Vstupné podmienky:	Pri registrácii užívateľ užívateľ polia.
Výstupné podmienky:	Užívateľské údaje sú zapísané v databáze.
Postup:	<ol style="list-style-type: none">1. Je odoslaná POST požiadavka na endpoint <code>/customers</code>, ktorá obsahuje request body s vyplnenými údajmi2. V backende sa spracujú údaje a skontroluje sa, či povinné polia majú správny tvar a boli vyplnené3. Ak boli správne vyplnené, tak vraciame response s pôvodným body a so status kódom 201, inak príde odpoveď vo formáte JSON s informáciou, aká nastala chyba so status kódom 400
Výsledok: PASS / FAIL	

Test 7: Dokončený servis auta	
Vstupné podmienky:	Používateľ má auto v servise, je opravené a pripravené na vyzdvihnutie.
Výstupné podmienky:	Opravené auto sa vymaže z databázy áut, ktoré sú servisované.
Postup:	<ol style="list-style-type: none"> 1. Je odoslaná DELETE požiadavka na endpoint <code>/cars/:id</code> 2. Obsahuje request body s údajmi o konkrétnom aute, ktoré databáza spracuje 3. Následne príde response s kódom 200 a informačným body, že auta sa vymazalo zo databázy v prípade úspešného odstránenia. Inak príde informácia v response body o chybe, že dané aute nie je evidované
Výsledok: PASS / FAIL	

Test 8: Získanie údajov o aute zákazníka	
Vstupné podmienky:	Používateľ má aute momentálne v servise.
Výstupné podmienky:	Používateľ vidí stav jeho auta, aké úkony sa vykonali na jeho aute.

Postup:	<ol style="list-style-type: none"> 1. Je odoslaná GET požiadavka na endpoint <code>/cars/:id</code>. 2. V backende sa request spracuje a hľadá sa auto s daným ID. 3. Ak sa auto s konkrétnym a jeho údaje sa vrátia v reponse body spolu s kódom 200. V prípade ak také auto nie je v databáze, vracia sa chybový kód 500 a response body s ohlásenou chybou, že auto nie je v databáze
Výsledok: PASS / FAIL	

Záporné testy

Test 9: Nepridelené autá technikovi	
Vstupné podmienky:	Technik chce vidieť stav pridelených áut, ale nemá žiadne auto pridelené.
Výstupné podmienky:	Technik neuvidí pridelené autá, zobrazí sa informačná chybová hláška.
Postup:	<ol style="list-style-type: none"> 1. Je odoslaná GET požiadavka na endpoint <code>/TechnicianCars/:id</code>. a request body s obrázkom 2. Na backende sa zistí, pri vyhľadávaní áut v databáze je odpoveďou prázdne pole 3. Posiela sa response body s chybovým kódom 404 a správou, že daný technik nemá pridelené ani jedno auto
Výsledok: PASS / FAIL	

Test 10: Nekompletné vyplnenie údajov auta	
Vstupné podmienky:	Používateľ zadáva objednávku, ale zabudol vyplniť správne rok výroby auta, čo je povinný údaj.
Výstupné podmienky:	Používateľovi sa zobrazí chybová správa o tom, že je potrebné tento údaj vyplniť.
Postup:	<ol style="list-style-type: none"> 1. Je odoslaná POST požiadavka na endpoint <code>/cars</code>. 2. Na backende sa zistí, že nie sú vyplnené všetky povinné polia pri pridávaní auta a jeho údajov do databázy. 3. Posiela sa response body s chybovým kódom 400 a správou, údaje o aute neboli úspešne registrované.
Výsledok: PASS / FAIL	

Backend aplikácie

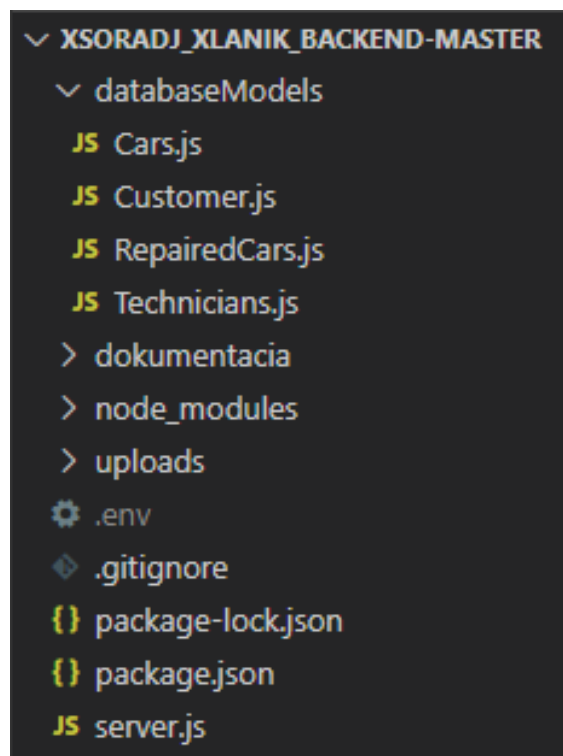
Súčasťou projektu je samotný backend aplikácie. Programovali sme v programovacom prostredí Visual Studio Code. Zdrojový kód aj s jeho súčasťami je dostupný v [Github repozitári](#).

Backend technológie

Pre naprogramovanie backendu a funkčných endpointov sme použili [Node.js](#) spolu s frameworkom [Express.js](#). Spoločná kombinácia poskytuje množstvo funkcionalít, na programovanie webových a mobilných aplikácií. Použili sme databázu [MongoDB](#) a tiež aj “[mongoose](#)” pre prehľadné objektové modelovanie.

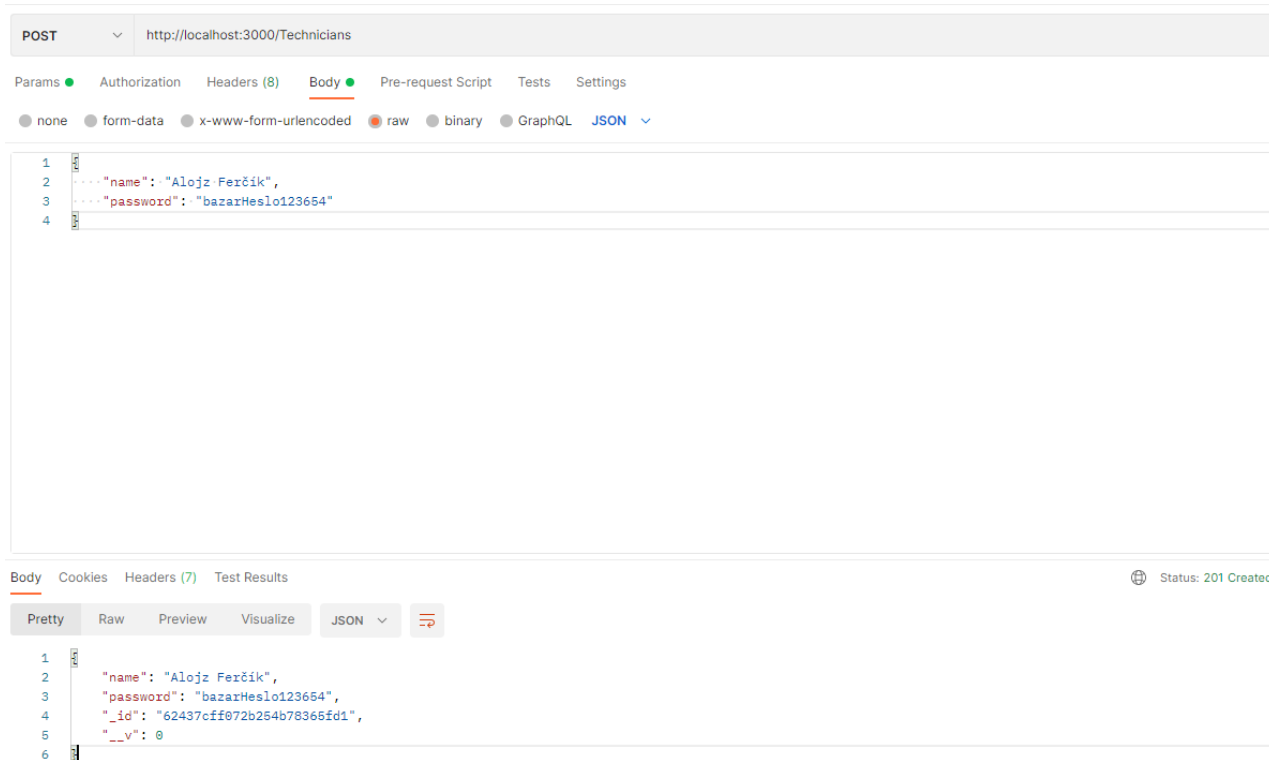
Štruktúra backendu a jeho funkcionalita

Štruktúra projektu obsahuje dôležité súčasti nato, aby sme mohli spúšťať projekt (*node_modules*, *package.json*, *package-lock.json*). Hlavná logika programu je naprogramovaná v súbore *server.js*, kde je základná funkcionalita, definícia API endpointov, pripojenie na databázu a samotné operácie s JSON objektami a databázou. Priečink *databaseModels* obsahuje modely, pre každú tabuľku nášho fyzického databázového modelu. V ňom sú definované štruktúry/modely každej tabuľky databázy.



Testovanie backendu

Pre overenie funkčnosti API endpointov sme použili nástroj alebo aplikáciu [Postman](#), ktorým sme sa správali ako HTTP klient a posielali sme HTTP request na náš backend. Následne sme správanie backendu mohli overovať odpoveďami priamo v Postman-ovi alebo tiež aj v samotnej databáze v prípade pridania, upravenia alebo odstránenia položky v databáze.



Tu vidíme príklad testovania funkčnosti nášho backendu, kde sme poslali POST so vstupnými údajmi vo formáte JSON. Dostali sme aj odpoveď so status kódom a v databáze sa takýto technik zapísal do príslušnej tabuľky technikov. Testovali sme všetky metódy a to GET, POST, PUT (PATCH) a DELETE. K dispozícii v repozitári je aj kolekcia volaní v nástroji Postman. Všetky API endpointy sú funkčné tak, ako sme pôvodne chceli a sme pripravení na prácu na frontendovej časti projektu.

Change log oproti Milestone 1

V milestone 1 odovzdaní projektu a teda oproti samotnému návrhu nastalo pár zmien, ktoré boli urobené v záujme zvýšenia kvality a konzistentnosti projektu:

- správne upravené názvy API endpointov, niektorých premenných v API dokumentácii nástroja Swagger.
- upravené detaily pri akceptačných testoch (status kódy, typ response body)
- pridaný POST endpoint */login*, na prihlásenie zákazníka alebo technika do systému. Súčasťou je aj validácia vstupných údajov.
- pridaný GET endpoint */CustomerCar/:id*, pre získanie auta príslušného zákazníka
- pridaný GET endpoint */TechnicianCars/:id*, pre získanie všetkých áut, ktoré má daný technik pridelené
- pridaná funkcionality uloženia obrázku do databázy, kde sa zapíše celý obsah obrázka vo formáte base64
- prerobený akceptačný test č.9, kde sme použili jeden z nových endpointov.

Frontend aplikácie

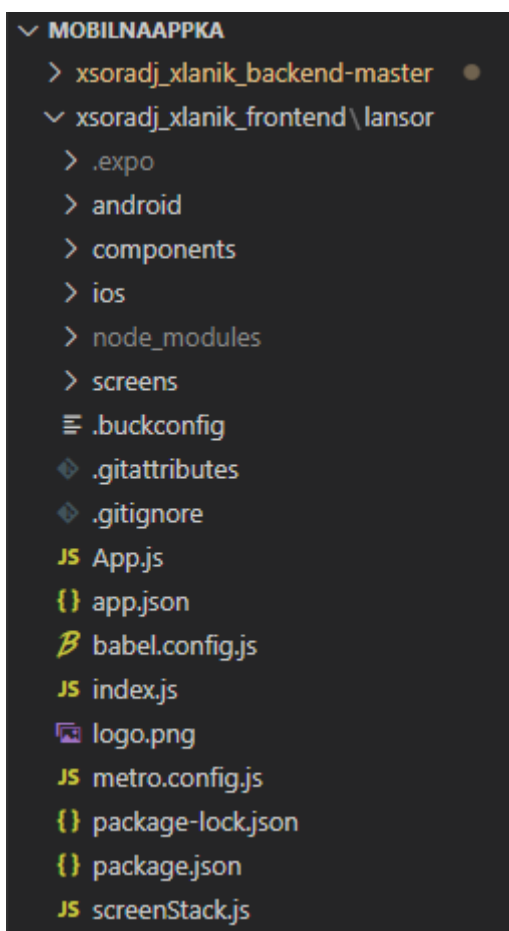
Túto časť projektu sme programovali taktiež v vývojovom prostredí Visual Studio Code. Zdrojový kód frontendu a všetky potrebné súčasti sú dostupné v [Github repozitári](#). Zároveň backend projektu je v tomto [Github repozitári](#).

Použité frontend technológie

Použili sme cross-platform technológiu [React Native](#), ktorá poskytuje natívny kód pre Android a pre iOS. Samotná platforma aplikácie je pomocou [Expo](#) frameworku. Programovali sme v jazyku JavaScript.

Štruktúra frontendu a jeho funkcionality

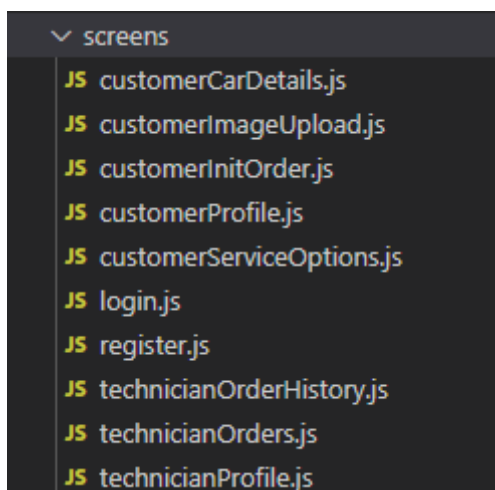
V štruktúre frontendovej časti sa nachádzajú potrebné súčasti projektu. Podobne ako v backende sa tu nachádza *package.json*, *package-lock.json*, *node_modules*. Taktiež obsahuje Expo súčasti pre spustenie aplikácie v development móde či už pre Android, iOS alebo aj webový prehliadač (*.expo*, *android*, *ios*, *.buckconfig*, *app.json*, *atd'*). Súbory a priečinky, kde sme programovali aplikáciu sú: *screenStack.js*, *App.js*, *screens*, *compoments*.



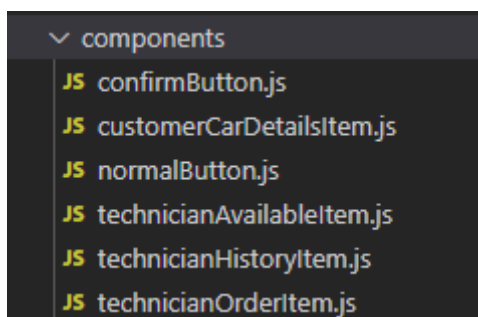
App.js - najvrchnejší základný komponent v ktorom sa nachádza funkcia pre renderovanie komponentu `<Navigator>`, ktorý má za úlohu zobrazit' danú obrazovku.

screenStack.js - samotná navigácia aplikácie a základ pre preklikávanie sa cez rôzne obrazovky. Použili sme jeden typ navigácie a to *Stack Navigation*.

Priečink "screens" - obsahuje Javascriptové súbory, v ktorých sa nachádza kód pre jednotlivé obrazovky (napr. *login.js*, *register.js*, *customerProfile.js*, atď.). Tieto jednotlivé obrazovky sú potom importované v súbore *stackScreens.js*. V jednotlivých obrazovkách sa nachádza hlavná logika danej obrazovky a tiež aj používanie "*fetch-u*" pre získavanie konkrétnych dát z nášeho backendu.



Priečink "components" - v tomto priečinku sa nachádzajú menšie a pomocné komponenty, ktoré sú importované neskôr v súboroch priečinku *screens*. V tomto priečinku "*components*" nájdeme nami definované tlačidlá a tiež aj jednotlivé "*itemy*", ktoré používame vo *FlatListoch*, kde zobrazujeme list záznamov zákazníkových vozidiel, list zákaziek, list opravených vozidiel a tiež aj zoznam technikov.



Spustenie mobilnej aplikácie a testovanie

Aplikáciu spúšťame v development móde pomocou príkazu “*expo start*”. Aplikácia bude fungovať či už na emulátore Android, iOS, webovom prehliadači ale aj na fyzickom mobilnom zariadení.

```
Try the new cross-platform PowerShell https://aka.ms/powershell
```

```
PS C:\Users\Jakub Sorád\Desktop\FIIT\8.semester\MTAA\MobilnaAppka> cd .\xsoradj_xlanik_frontend\  
PS C:\Users\Jakub Sorád\Desktop\FIIT\8.semester\MTAA\MobilnaAppka\xsoradj_xlanik_frontend> cd .\lansor\  
PS C:\Users\Jakub Sorád\Desktop\FIIT\8.semester\MTAA\MobilnaAppka\xsoradj_xlanik_frontend\lansor> expo start
```

Pre testovanie funkčnosti a správnej funkcionality sme používali spočiatku webový prehliadač, kde sme videli vizuálnu stránku aplikácie a tiež v konzole sme mohli kontrolovať, ako nám prichádzajú dáta z databázy, správnosť volaní a hlavičiek pri dopytoch na databázu a backend. Neskôr sme aplikáciu testovali na reálnych mobilných zariadeniach so systémom Android aj iOS.

The screenshot displays the Expo CLI interface. On the left, the 'Metro Bundler' section shows the process status as 'PROCESS (0) - 11:57:56 AM'. Below it, the 'Mi A1' device is listed as 'DEVICE (1) - 11:58:25 AM'. A list of actions is provided: 'Run on Android device/emulator', 'Run on iOS simulator', 'Run in web browser', 'Send link with email...', and 'Publish or republish project...'. The 'PRODUCTION MODE' toggle is turned off. The 'CONNECTION' section shows 'Tunnel' as the selected option, with 'LAN' and 'Local' as alternatives. A QR code is displayed at the bottom left for scanning. On the right, the 'LOGGED IN AS' section shows the 'METRO BUNDLER' logs, which include the following information: 'Starting Metro Bundler', 'Successfully ran adb reverse', 'Using legacy dev server: false', 'Tunnel connected.', 'Tunnel ready.', and 'Building JavaScript bundle: finished in 3187ms.'

Po spustení príkazu “*expo start*” sa nám otvorí v prehliadači okno (viď obrázok hore) na URL: localhost:19002. Na pripojenie reálneho zariadenia zvolíme možnosť pripojenia na typ “**Tunnel**”. Následne v reálnom zariadení sa vieme pomocou naskenovania QR kódu alebo zadáním v aplikácii Expo Go linku na náš projekt dostať do stavu, kedy vieme s našou aplikáciou reálne interagovať používať ju.

Zhodnotenie frontendu

Zhodnotenie rozdelíme do dvoch celkov. Prvé spočíva v tom, ako sa nám podaril frontend z hľadiska dizajnu a tiež aj z programovacieho hľadiska.

Keď si porovnáme ako vyzerajú naše wireframy v nástroji Figma a naša reálna aplikácia, môžeme tvrdiť, že sme sa k návrhu wireframov priblížili na veľmi dobrej úrovni. Sú tam malé zmeny a odlišnosti od návrhu, ktoré ale vyplývali pri samotnom programovaní a tým, ako sme to z programovacieho hľadiska zakomponovali.

Odlišnosť je napríklad pri zozname zákazníkových áut, zákazok alebo histórii opravených áut, kde v návrhu sme mali údaje zobrazované viac menej v riadku, ale pri programovaní sme sa rozhodli údaje dávať pod seba. Prišlo nám to lepšie, jasnejšie a používateľ nemá problém prstom posúvať sa na ďalšie autá v zozname smerom nadol alebo naopak hore. Malá odlišnosť je pri výbere technika, kde nemáme Scrollbar, ale máme tlačidlá na výber s aktuálnou informáciou o vybranom technikovi.

Druhé zhodnotenie sa týka samotnej funkcionality aplikácie. Po dlhej a náročnej snahe zakomponovať videohovor medzi zákazníkom a technikom cez WebRTC sa nám to nepodarilo. Naskytlo sa veľké množstvo komplikácií spojených s React Native a Expom, ktoré sme nedokázali dotiahnuť do funkčného celku. Tým pádom súčasťou mobilnej aplikácie a projektu bude jedna z bonusových úloh, ktorá je v našom prípade povinná a budeme sa venovať komunikácií cez websokety.

Keď sa pozrieme na funkcionality mimo WebRTC, tak spĺňa všetko to, čo bolo stanovené na začiatku. Prihlasovanie a registrácia funguje správne aj s viacerými validáciami. Zákazník vidí údaje o sebe. Vie si pozrieť stav svojich vozidiel a opravené auto si prevziať a potvrdiť to. Toto opravené vozidlo bude potom viditeľné v histórii opravených áut na strane technikov. Zákazník má k dispozícii kompletný objednávací proces do servisu spolu s nahratím fotografie auta. Súčasťou sú aj viaceré validácie vstupných údajov používateľa. Na opačnej strane technik má prístup k svojim zákazkám. Ak ide potvrdiť opravenie auta vie poskytnúť ďalšie informácie a potvrdiť to. Má možnosť pozrieť si históriu opravených áut vo firme.

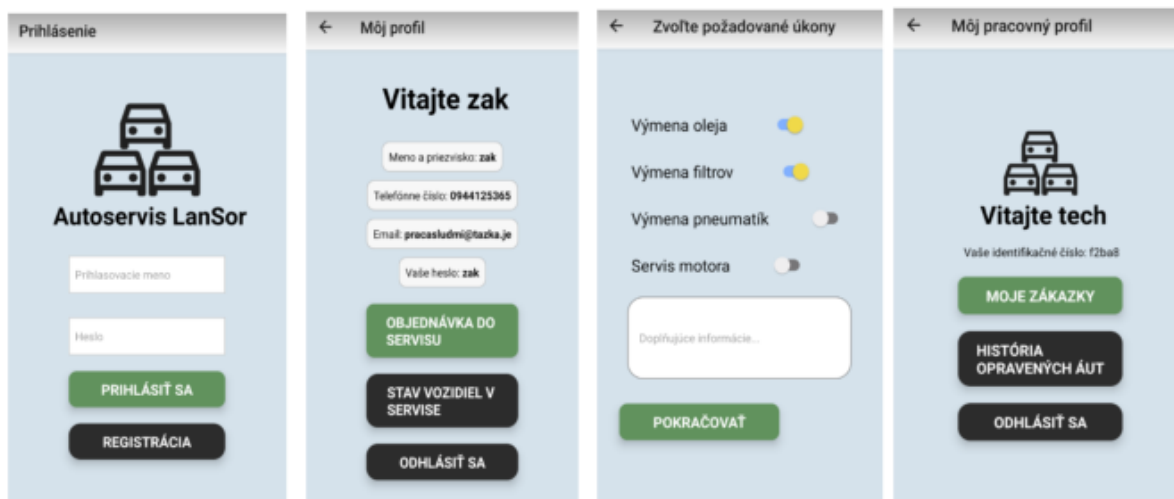
Všetky tieto vyššie spomenuté funkcionality sú úspešne implementované a aplikácia tvorí funkčný celok nami zadanom rozsahu.

Na záver tejto podkapitoly pridáme jeden obrázok niektorých obrazoviek a porovnania návrhu s reálnou aplikáciou.

Figma návrh



Reálna aplikácia



Change log oproti Milestone 2

Spravili sme niekoľko zmien oproti predošlému milestone. Tieto zmeny vnímame opäť ako prínos pre zlepšenie funkcionality a kvality nášho projektu:

- malé úpravy vo frontend akceptačných testoch. Korekcia niektorých krokov a názvosloví
- vylepšenia na backende v hlavnom súbore *server.js*. Pridanie middleware *Cors* pre správne fetchovanie dát z frontendu. Ďalej úprava volania POST pri pridávaní auta do databázy. Pôvodne v Milestone 1 sme obrázok vkladali a posielali cez Formdata a v *server.js* sme museli dáta parsovať. Na frontende sme výber a následne enkódovanie obrázku dali do Base64 a vedeli sme odoslať POST požiadavku ako *application/json*. Ďalej nastala malá zmena pri volaní metódy PATCH, ktorá zabezpečila správnu úpravu dát parametru “description” a “state”.
- V predošlom Milestone sme backend aplikácie spúšťali na našich zariadeniach (*localhost*). To nám spôsobovali problémy pri frontende pri fetchovaní dát. Z toho dôvodu sme sa rozhodli celý backend repozitár deploynúť na Heroku hostovaciu službu. Problém je tým pádom vyriešený a backend aplikácie je dostupný kedykoľvek a kdekoľvek na adrese <https://lansormtaa.herokuapp.com/>

Doplnková úloha – Websokety

Rozhodli sme sa implementovať túto doplnkovú úlohu, pri ktorej spočíva aby naša mobilná aplikácia nepoužívala API endpointy a volania na nich, ale aby všetky informácie a dáta išli práve cez websoket. Bolo potrebné prerobiť a upraviť či už backend, ale aj frontend aplikácie. Naše riešenie je verejne dostupné na týchto dvoch Github repozitároch:

- Repozitár pre [frontend](#)
- Repozitár pre [backend](#)

Backend aplikácie

Pre základnú funkčnosť backendu, ktorý je v súbore *server.js* sme použili verejne dostupnú knižnicu [express-ws](#), ktorá podporuje websokety a aplikovali sme ju na samotnú *express app* v zdrojovom kóde. Následne sme mohli prijímať vstupné dáta cez websoket.

```
app.ws('/', function(ws, req) {
  ws.on('message', async (message) => {

    const parsedMessage = JSON.parse(message);
    console.log(parsedMessage.data);

    switch(parsedMessage.information){
      case "loginCustomer":
```

V hlavnom switchi sme si vždy zistili o akú informáciu ide a podľa toho sme vedeli spracovať dáta a poslať prípadnú odpoveď na frontend aplikácie. Takto sme nahrali potrebu používania endpointov. V tomto prípade sme odpovede posielali pomocou základnej funkcie *ws.send*. Tým, že s websoketmi sa pracuje odlišne, tak v kóde nastalo aj viacero signifikantnejších zmien, ale stále sme sa mohli vo veľkej miere oprieť o riešenie v druhom a treťom milestone. Uvedieme príklad spôsobu programovania s API endpointami, druhý obrázok bude s websoketmi v konkrétnom prípade, kedy sme na backend server dostali požiadavku na validácie a prihlásenia používateľa/technika a následne posielame spätnú väzbu.

```

app.route('/login')
  .post(async (req, res) => {
    try{
      const loginCustomer = await Customer.findOne({ name: req.body.name })
      if(loginCustomer){
        if(loginCustomer.password == req.body.password){
          res.status(200).json({ loginCustomer })
          return
        }
        else{
          res.status(400).json({ message: "Zle prihlasovacie udaje" })
          return
        }
      }
    }
    else{
      const loginTechnician = await Technician.findOne({ name: req.body.name })
      if(loginTechnician){
        if(loginTechnician.password == req.body.password){
          res.status(200).json({loginTechnician })
          return
        }
        else{
          res.status(400).json({ message: "Zle prihlasovacie udaje" })
          return
        }
      }
    }
    res.status(400).json({ message: "Zle prihlasovacie udaje" })
    return
  }
  catch(err){
    res.status(400).json({ message: err.message })
    return
  }
})

```

```

switch(parsedMessage.information){
  case "loginCustomer":
    try{
      const loginData = JSON.parse(parsedMessage.data);
      const loginCustomer = await Customer.findOne({ name: loginData.name })
      if(loginCustomer){
        if(loginCustomer.password == loginData.password){
          ws.send(JSON.stringify({ loginCustomer }));
          break;
        }
        else{
          ws.send(JSON.stringify({ message: "Zle prihlasovacie udaje" }));
          break;
        }
      }
    }
    else{
      const loginTechnician = await Technician.findOne({ name: loginData.name })
      if(loginTechnician){
        if(loginTechnician.password == loginData.password){
          ws.send(JSON.stringify({ loginTechnician }));
          break;
        }
        else{
          ws.send(JSON.stringify({ message: "Zle prihlasovacie udaje" }));
          break;
        }
      }
    }
    ws.send(JSON.stringify({ message: "Zle prihlasovacie udaje" }));
    break;
  }
  catch(err){
    ws.send(JSON.stringify({ message: err.message }));
    break;
  }
}

```

Frontend aplikácie

Signifikantné zmeny nastali aj na frontende. Samotné renderovanie obsahu zostalo až na názvoslovie volanie niektorých funkcií nezmenené. Opäť sme potrebovali kompletne upraviť logiku a štýl posielania a prijímania dát cez websokety. Dôležité bolo samotné pripojenie sa na websocket.

```
var ws = new WebSocket('ws://192.168.0.109:3000/')
```

Neskôr bolo nutné, aby sme zhodnotili, ako a kde budeme dáta posielat' a kde ich budeme prijímať. Na posielanie dát sme použili pôvodné funkcie, v ktorých sme si pripravili štruktúru dát a následne sme pomocou funkcie *ws.send* ich aj poslali smerom na backend.

Pokiaľ sme očakávali spätnú správu, alebo akúkoľvek inú správu, tak sme použili funkciu *ws.onmessage*. Táto funkcia sa zavola v prípade, ak nám prišla správa z backendu. Spracovali sme si vstupné dáta a následne sme ich validovali a potom väčšina operácii spočívala v tom, aby sme sa vedeli korektne prepnúť na ďalšiu obrazovku. Uvedieme porovnanie prihlasovania sa na frontende pri spôsobe s API volaniami a druhý obrázok bude práca s websocketmi.

```
const pressLoginHandle = async () => {
  const userCredentials = {
    name: name,
    password: password
  }

  const fetchObj = {
    method: 'POST',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(userCredentials)
  }

  try {
    const response = await fetch(`https://lansormtaa.herokuapp.com/login`, fetchObj);
    const userJsonRes = await response.json();
    console.log(userJsonRes);

    //Zistenie podľa údajov, ci pokračujeme ako customer alebo technik, alebo boli nespravne udaje
    if(userJsonRes.loginCustomer) navigation.navigate('CustomerProfile', userJsonRes);
    if(userJsonRes.loginTechnician) navigation.navigate('TechnicianProfile', userJsonRes);
    if(userJsonRes.message){
      Alert.alert(
        "Nesprávne prihlasovacie údaje",
        "Prosím skontrolujte správnosť mena a hesla",
        [
          { text: "OK", onPress: () => console.log("Zly login alert") }
        ]
      );
    }
  } catch (error) {
    console.error(error);
  }
  return;
}
```

```

// ak pride sprava z backendu
ws.onmessage = (e) => {
  const message = (e.data)
  const userData = JSON.parse(message);
  console.log(userData);

  try {
    //Zistenie podľa údajov, ci pokračujeme ako customer alebo technik, alebo boli nespravne udaje
    if(userData.loginCustomer) navigation.navigate('CustomerProfile', userData);
    if(userData.loginTechnician) navigation.navigate('TechnicianProfile', userData);
    if(userData.message){
      Alert.alert(
        "Nesprávne prihlasovacie údaje",
        "Prosím skontrolujte správnosť mena a hesla",
        [
          { text: "OK", onPress: () => console.log("Zly login alert") }
        ]
      );
    }
  } catch (error) {
    console.error(error);
  }
}

const pressLoginHandleWS = async () => {
  const userCredentials = {
    name: name,
    password: password
  }
  ws.send(JSON.stringify({
    information: 'loginCustomer',
    data: JSON.stringify(userCredentials)
  }));
  return;
}

```

Zhodnotenie doplnkovej úlohy

Doplnkovú úlohu považujeme za úspešne zvládnutú. Backend sme testovali a pracovali na ňom s tým, že fungoval na *lokalhoste* (resp. lokálna IP adresa zariadenia, na ktorom zdrojový kód beží). Frontend aplikácie fungoval takisto správne ako pri predošlých milestonech, tým pádom samotný používateľský zážitok je nezmenený. Na spúšťanie frontendu sme používali *Expo Go*, na ktorom sme testovali funkčnosť aplikácie priamo na mobilných zariadeniach.