České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Vachula Richard

Studijní program: Otevřená informatika
Obor: Softwarové systémy

Název tématu: Databáze a rozhraní pro modul automobilu pro systém sdíleni automobilu vice uživateli

Pokyny pro vypracování:

Navrhnete a implementujte databázi a rozhraní pro modul automobilu pro systém sdíleni vozového parku vice uživateli.
1) Dohodnete s ostatními studenty pracujícími na tomto systému funkcionalitu systému a zdokumentujte všechny způsoby použiti funkci pomoci UML diagramu.
2) Navrhnete rozhraní pro komunikaci s modulem včetně patřičného zabezpečeni, jez bude nainstalován v každém automobilu.
3) Navrhnete vhodnou strukturu databáze tak, aby tato databáze pokrývala všechny způsoby užití systému a poskytovala také důležité auditovací informace.
4) Implementujte navrženou databázi a rozhraní pro modul automobilu. Implementace bude obsahovat jednotkové i integrační testy včetně testu simulujících modul automobilu.
5) Navrhnete budoucí rozšířeni systému.

Seznam odborné literatury:

[1] Fowler, M.: UML Distelled - Third Edition. Addison-Wesley, 2003.
[2] Watson, R. T.: Data Management : Databases and Organizations. John Wiley & Sons, 2006.
[3] Tulach, J.: Practical API Design. Apress, 2008.
[4] Novotny, T.: Technicka zprava: Technicka vize CarSharingu. CVUT FS, 2015.

Vedoucí: doc. Ing. David Šišlák, Ph.D.

Platnost zadání do konce letního semestru 2017/2018

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25.1.2017

České vysoké učení technické v Prahe
Fakulta elektrotechnická
Katedra počítačů

Bakalárska práca

Databáze a rozhraní pro modul automobilu pro systém sdílení automobilů více uživateli

Richard Vachula

Vedúci práce: Doc. Ing. David Šišlák, PhD.

Študijný program: Otevřená informatika, bakalársky

Odbor: Softwarové systémy

24. mája 2017

Poďakovanie

Ďakujem Doc. Ing. Davidovi Šišlákovi, PhD., za odbornú pomoc a konzultácie, ktoré mi pomohli pri vypracovaní mojej bakalárskej práce.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác.

V Prahe dňa 24. 5. 2017 .....

Abstract

VACHULA, Richard: Database and API for car module. [Bachelor thesis] - Czech Technical University in Prague. Faculty of Electrical Engineering, Department of computers. Supervisor: Doc. Ing. David Šišlák, PhD.

Aim of the thesis is to design and implement an interface for car module to communicate with the backend system taking security of this communication into account. Thesis also focuses on the design and implementation of database used in Uniqway system providing necessary audit data. Proper implementation is then verified by means of unit and integration testing.

Keywords: carsharing, Uniqway, database, API

Abstrakt

VACHULA, Richard: Databáze a rozhraní pro modul automobilu pro systém sdílení automobilů více uživateli. [Bakalárska práca] - České vysoké učení technické v Praze. Fakulta elektrotechnická, Katedra počítačů. Vedúci: Doc. Ing. David Šišlák, PhD.

Práca sa zaoberá návrhom a implementáciou rozhrania pre komunikáciu modulu so serverovou jednotkou spolu s vhodne zvoleným spôsobom zabezpečenia. Súčasne si kladie za cieľ i návrh a implementáciu databázy pre perzistenciu dát, poskytujúcej dôležité informácie pre audit, a následné overenie funkčnosti systému jednotkovými a integračnými testami.

Kľúčové slová: carsharing, Uniqway, databáza, API

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

1.1 Motivácia práce

Carsharing, čoraz častejšie skloňovaný pojem v oblasti dopravy, predstavuje inovatívne riešenie zdieľania vozov. Jeho princíp je jednoduchý - umožniť jednotlivcom získať výhody privátneho použitia vozidla bez nákladov a povinností spojených s jeho vlastníctvom.

Ako súčasť zdieľanej mobility má carsharing taktiež mnoho environmentálnych, sociálnych a dopravných benefitov. Medzi nimi napríklad rozšírenie spádovej oblasti verejnej dopravy, zvýšenie prístupu k vytváraniu príležitostí pre nové cesty, ktoré pred tým neboli verejnou dopravou dosiahnuteľné alebo úspora nákladov a pohodlie, často citované ako populárny dôvod pre jeho využitie [5].

Projekt Uniqway je študentským projektom; dielom troch pražských univerzít - ČVUT, VŠE a ČZU - spolupracujucích na tvorbe systému zdieľania áut spolu so Škoda Auto a.s. Cieľovou skupinou tohto projektu sú práve študenti a zamestnanci týchto univerzít. Základnou myšliekou je spostredkovať im možnosť využitia vozidiel v prípadoch ako je preprava do školy či na internát, nákup, víkendový výlet, pracovná cesta, a pod.

Technická stránka projektu je z koncepčného hľadiska tvorená štyrmi základnými jednotkami - backendovým riešením tvoreným databázou a serverom, ktorý je zároveň centrálnym bodom komunikácie, klientskou mobilnou aplikáciou, webovou aplikáciou určenou pre správcu vozového parku a hardvérovou jednotkou (modulom) umiestnenou priamo vo vozidlách.

1.2 Cieľ práce

Cieľom tejto práce je navrhnúť a implementovať rozhranie pre komunikáciu modulu umiestneného vo vozidle so serverom spolu s vhodne zvoleným spôsobom jej zabezpečenia. Cieľom je taktiež navrhnút a vytvoriť databázu využívanú v rámci systému Uniqway, potrebnú pre uloženie dôležitých dát. Súčasťou práce je taktiež overenie funkčnosti implementovaných častí jednotkovými a integračnými testami.

1.3 Popis kapitol

Práca je rozdelená do šiestich kapitol. Obsahom prvej je motivácia práce a jej stanovený cieľ. Kapitola druhá sa zaoberá modulom, a to z hľadiska spôsobu komunikácie spolu s popisom výberu technológií použitých pri tvorbe jej rozhrania a potrebného zabezpečenia. Stručná teória návrhu a typov databáz spolu s výberom pri implemenácii použitej je adresovaná v kapitole tri. Popisu navrhnutého modelu a implementácii je venovaná štvrtá kapitola. V piatej kapitole je uvedený spôsob testovania systému spolu so stručnou charakteristikou programu pre simuláciu modulu, ktorý bol v rámci projektu implementovaný. Zhodnotenie práce spolu s možnosťami ďalšieho vývoja sú obsiahnuté v kapitole 6. Medzi prílohami je uvedená bloková schéma hardvérovej jednotky a rozšírený databázový model.

Kapitola 2

Komunikácia s modulom

2.1 Modul

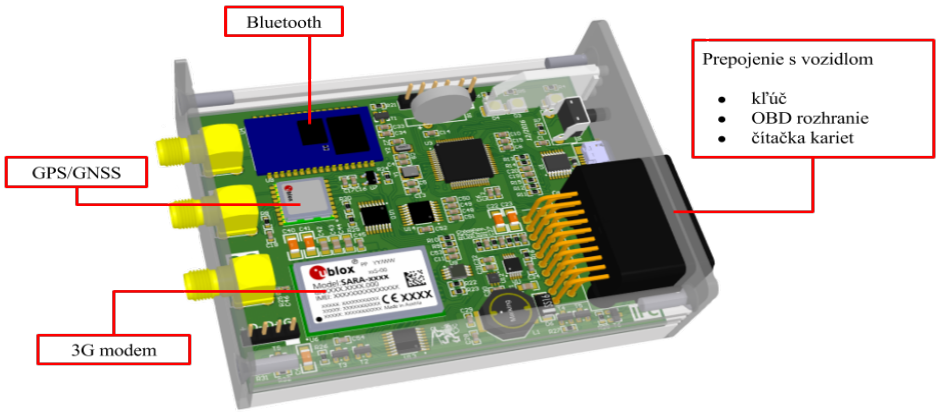
Kapitola stručne charakterizuje modul a jeho význam. Popisuje spôsob jeho komunikácie spolu s možnosťou jej zabezpečenia. Venuje sa taktiež výberu technológií použitých pri tvorbe rozhrania.

2.1.1 Charakteristika

Neoddeliteľnou súčasťou celého systému je modul - hardvérová mobilná časť umiestnená v každom z vozidiel. Jej hlavnou úlohou je komunikácia s nadradenou časťou systému - serverom, za účelom pravidelného sprostredkovania informácií o aktuálnom stave vozidla a zároveň správne reagovanie na ním zaslané inštrukcie.

Jednotka implementuje príkazy na odomknutie a uzamknutie vozidla, ktorých správne vykonanie je zabezpečené pomocou ďalších, k nej pripojených periférií.

Na obrázku 2.1 je zobrazený jej náhľad spolu s popiskom hlavných komunikačných častí. Bloková schéma spolu s popisom jednotlivých častí je umiestnená v prílohe.



Obr. 2.1: Detail hardvérovej jednotky

2.1.2 Spôsob komunikácie

Hardvérová jednotka disponuje možnosťou komunikácie prostredníctvom mobilnej GPRS siete, zasielaním HTTP požiadaviek na vzdialený server.

HTTP je internetový protokol, bežne používaný na prenos dát v sieti modelom *klient-server*, kde klient je strana, ktorá dané spojenie iniciuje. Toto spojenie avšak postráda potrebnú úroveň zabezpečenia. Týmto problémom a jeho riešením sa zaoberá kapitola 2.3.

Potrebné údaje o vozidle sú jednotkou zasielané HTTP požiadavkou v pravidelných intervaloch, a to s periódou zníženou až na 10 sekúnd počas jazdy. Tieto požiadavky budú ďalej označované ako „synchrónne“. V odpovedi na ňu zo strany serveru je

očakávaná informácia o tom, či sa dané vozidlo nachádza v oblasti, v ktorej je umožnené parkovanie. Aktuálna poloha vozdidla je získaná z dát zaslaných požiadavkou (viď 2.1.3).

Komunikácia jednotky so serverom môže byť taktiež podmienená dvomi externými vstupmi:

- Použitím RFID čítačky pripojenej k modulu
- Použitím klientskej mobilnej aplikácie

RFID čítačka slúži na odomykanie, resp. zamykanie vozidla prostredníctvom RFID kariet. Pri ich použití je modulom na server zaslaná požiadavka kvôli validácii prístupu. Takáto požiadavka bude ďalej označovaná ako „asynchrónna“. Očakávanou odpoveďou pri úspešnej validácii má byť inštrukcia pre vykonanie požadovaného úkonu.

Na obrázku 2.2 je zobrazený sekvenčný diagram s priebehom komunikácie v prípade použitia RFID čítačky.

Klientská mobilná aplikácia takisto sprostredkúva možnosť odomykania, resp. zamykania vozidla. V tomto prípade ale komunikácia prebieha medzi aplikáciou a serverom. Výsledkom úspešnej validácie má byť zaslanie príslušnej inštrukcie do modulu požadovaného vozidla.

Pri použití mobilnej aplikácie je modul postavený do role *serveru*, čo však nie je z jeho strany podporované. Riešením tejto situácie je odkladanie jednotlivých inštrukcií na strane serveru a ich postupné zasielanie v odpovediach na synchrónne požiadavky. Sekvenčný diagram 2.3 znázorňuje priebeh komunikácie v prípadoch použitia mobilnej aplikácie s uvedeným riešením.

2.1.3 Zasielané dáta

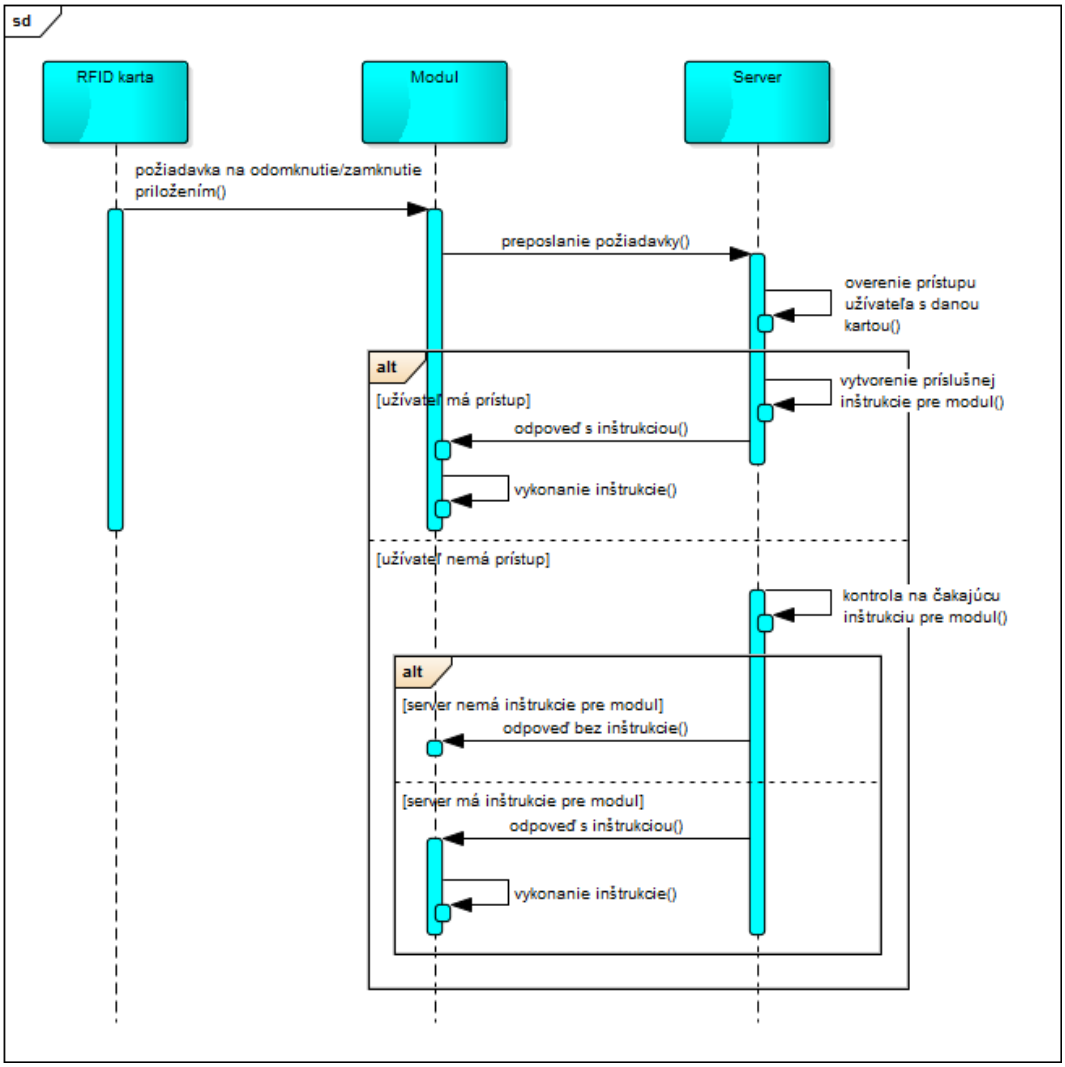
V kapitole 2.1.2 boli rozlíšené dva typy požiadaviek zasielaných hardvérovou jednotkou na server - synchrónne a asynchrónne. Obe sa okrem pravidelnosti odosielania líšia najmä v dátach, ktoré sú nimi prenášané.

Jednou z úloh jednotky je sledovanie prevádzkových veličín vozidla ako napr. rýchlosť, spotreba či stav paliva v nádrži, ktoré sa však môžu líšiť v závislosti od typu vozidla. Okrem toho je nutné mať prehľad o aktuálnej polohe vozidla. Pre tento účel jednotka využíva technológiu GPS. Oba typy údajov so sebou nesú i príznak o validite - prevádzkové dáta sú valídne len v prípade aktívneho napájania vozidla a GPS môžu byť nevalídne z dôvodu zlého signálu.

Spolu s príznakom o tom, či je vozidlo aktuálne zamknuté, časovou značkou a príznakom o dočasnom prerušení jazdy, ktorý slúži na informovanie o tom, že užívateľ vozidlo opustil, ale plánuje sa k nemu vrátiť, napr. pri nakupovaní, sú súčasťou synchrónnej požiadavky.

Asynchrónny požiadavok pozostáva z identifikátoru RFID karty, časovej značky a informácie o tom, či užívateľ požaduje uzamknutie alebo odomknutie vozidla.

Napriek tomu, že množina modulom zasielaných dát spolu so spôsobom ich prenosu sú dané jeho návrhom a vozidlom, v ktorom je umiestnený, výber vhodného formátu prenášaných dát je súčasťou tejto práce (2.2.2).



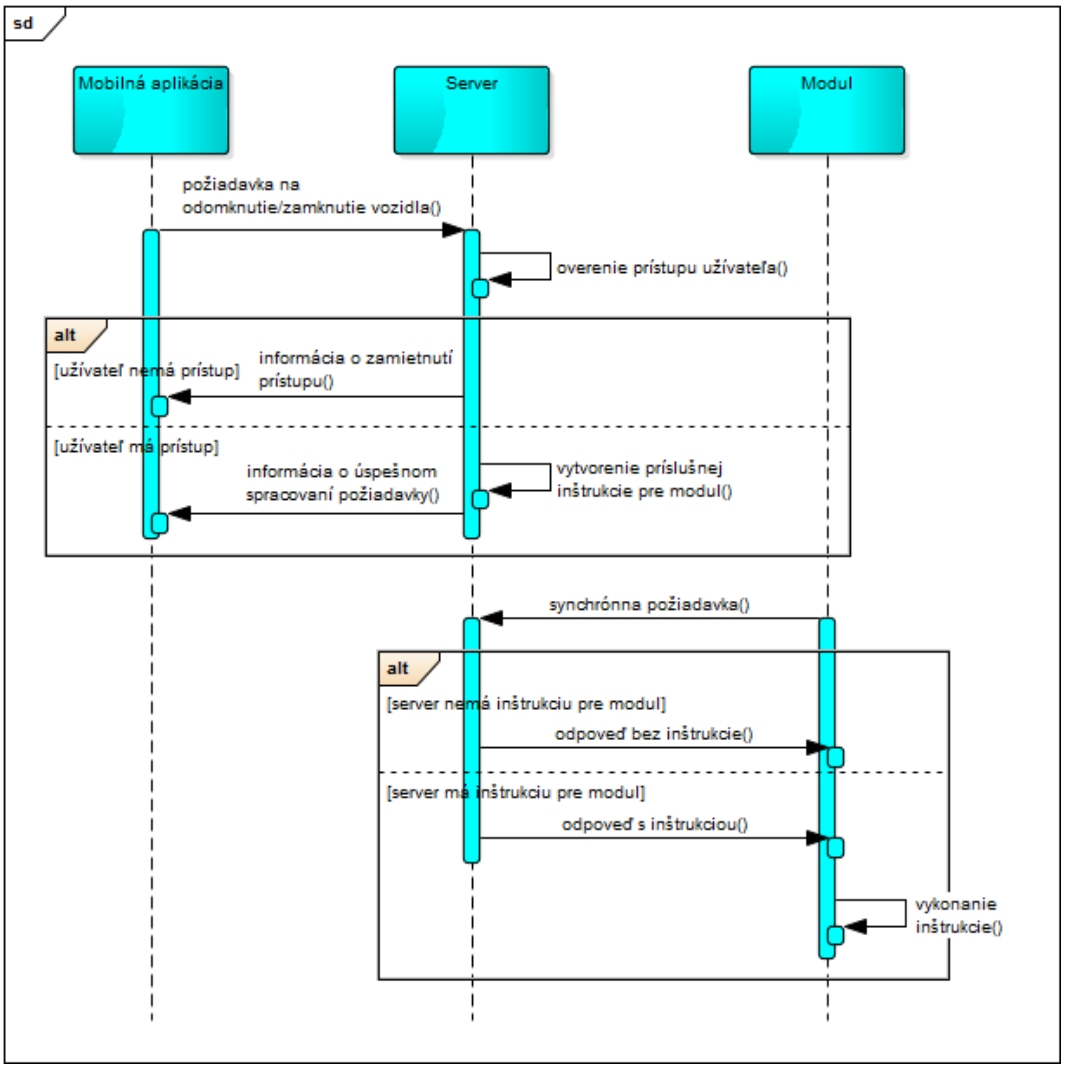
Obr. 2.2: Sekvenčný diagram komunikácie (RFID)

2.2 Rozhranie

Rozhraním sa z pohľadu hardvérovej jednotky myslí webové aplikačné programové rozhranie (API); definovaná množina prístupových bodov na strane serveru, ktoré môže prostredníctvom HTTP požiadaviek adresovať. Ide teda o istú vrstvu abstrakcie nad jeho vnútornou implementáciou a funkcionalitou, ktorými sa zaoberá kapitola 4.

Pri tvorbe rozhrania bolo nutné uvažovať:

- výber vhodného typu rozhrania 2.2.1,
- výber vhodného formátu pre výmenu dát 2.2.2.



Obr. 2.3: Sekvenčný diagram komunikácie (mob. aplikácia)

2.2.1 Výber typu

Medzi rozšírené spôsoby návrhu patria Representational State Transfer (REST) a Simple Object Access Protocol (SOAP).

REST je architektonický štýl rozhrania pre distribuované prostredia [2]. Nie je závislý na použitom protokole. Najčastejšie je ale spájaný práve s HTTP, pri použití ktorého je rozhranie obmedzené na množinu štandardných operácií tohto protokolu (napr. GET, POST, atď.). Je orientovaný dátovo, nie procedurálne. Je použiteľný pre jednotný a jednoduchý prístup ku zdrojom, teda dátam, adresovateľným pomocou URI.

SOAP je, narozdiel od REST, protokolom slúžiacim pre výmenu štrukturovaných informácií v distribuovanom prostredí s využitím XML technológie. Pozostáva z

troch častí, a to definície obsahu a spôsobu spracovania vymieňaných správ, kódovacích pravidiel pre vyjadrenie dátových typov aplikácie a konvencie pre reprezentáciu procedurálnych volaní a ich odpovedí [4].

Pre vytvorenie nami požadované rozhrania bol zvolený REST. Jedným z dôvodou je jednoduchšia implementácia na strane modulu. Výhodou REST je taktiež bezstavovosť, čo môže zjednodušiť prípadné škálovanie aplikácie v budúcnosti. V porovnaní so SOAP, REST predstavuje menšiu záťaž na sieť a zároveň rýchlejšiu odozvu pri komunikácii [4].

2.2.2 Výber formátu pre výmenu dát

REST ako zvolený typ rozhrania nie je viazaný na použitie konkrétneho formátu prenášaných dát. Bežne používanými práve v spojení s ním sú JSON a XML.

JSON je textový, jazykovo nezávislý formát, vychadzajúci z podmnožiny programovacieho jazyka JavaScript. Je kompaktný, pre človeka čitateľný. Je jednoduchý na vytvorenie a následné spracovanie [1].

XML je rozšíriteľný značkovací jazyk, vyvinutý a štandardizovaný konzorciom W3C ako zovšeobecnenie jazyka HTML. Umožňuje špecifikovanie štruktúry prenášaných dát (tzv. schémy) pre možnosť ich validácie na strane prijímateľa. V porovnaní s JSON majú prenášané dáta väčšiu veľkosť a je pomalší na spracovanie, čo je dané jeho značkovacou syntaxou [2].

Nami zvoleným formátom bol JSON, a to z dôvodu menšej veľkosti prenášaných dát s čím je spojené i jeho rýchlejšie spracovanie.

2.3 Zabezpečenie

Dôležitým aspektom komunikácie medzi modulom a serverom je jej zabezpečenie. HTTP protokol sám o sebe neposkytuje jeho potrebnú úroveň a komunikáciu je možné napadnúť treťou stranou. Ide o tzv. *Man-in-the-middle* typ útoku, kedy je útočník schopný „odpočúvať“ priebehajúcu komunikáciu, či sa do nej aktívne zapojiť a simulovať jednotlivé komunikujúce strany voči druhým.

Nutnými požiadavkami z pohľadu zabezpečenia sú preto:

- utajenie - dáta nemôžu byť prezerané treťou stranou,
- integrita - dáta nemôžu byť počas prenosu modifikované,
- autentifikácia - koncové strany sú tými, za ktoré sa vydávajú.

[1] `http://www.json.org/`
[2] `https://www.w3.org/TR/xml/`

2.3.1 HTTP Secure

Riešením uvedených požiadaviek 2.3 je použitie tzv. *HTTP Secure* (HTTPS). Ide o nami zvolený komunikačný protokol HTTP zabezpečený Secure Socket Layer (SSL) protokolom na úrovni relačnej vrstvy OSI modelu.

HTTPS stavia na symetrickom šifrovaní prenášaných dát (pre šifrovanie i dešifrovanie je použitý ten istý kľúč) obomi stranami (označme A a B) zdieľaným kľúčom. Pre jeho výmenu je taktiež použité šifrovanie, v tomto prípade asymetrické. Obe strany zabezpečujú vytvorenie vlastného *verejného* a *privátneho* kľúča. Pred odoslaním dát z A do B je použitý verejný kľúč B pre ich zašifrovanie. Takto zašifrované dáta možno rozšifrovať len privátnym kľúčom B. Utajenie a integrita dát sú teda zaistené touto cestou. Podrobnejšiemu rozboru SSL sa venuje [8].

Pre autentifikáciu jednotlivých strán využíva SSL tzv. *certifikáty*, okrem verejného kľúča nesúce i informáciu o jeho vlastníkovi a tzv. *certifikačnej autorite* (CA), ktorou je daný certifikát podpísaný. CA je treťou stranou, ktorá ak je považovaná za dôveryhodnú, tak i certifikát ňou podpísaný je považovaný za dôveryhodný. Spôsob implementácie daného zabezpečenia je popísaný v kapitole 4.2.2.

Kapitola 3

Databáza

Úloha databázy je v našom systéme nepostradateľnou. Každá interakcia s ním vyžaduje prácu s dátami, ktoré je potrebné vhodným spôsobom evidovať. Klásť dôraz je nutné i na dlhodobé uchovávanie dát, a to najmä pre auditívne účely či pre ich potrebnú analýzu.

Kapitola približuje normalizačné normy ako postup správneho návrhu databázy. Ďalej sú priblížené dve najpoužívanejšie databázové technológie - relačné a nerelačné databázy[1] a výber nami použitej spolu s výberom konkrétneho systému pre jej správu.

3.1 Návrh

Implementácii databázy predchádza jej návrh. Jeho tvorba spočíva v definovaní jednotlivých entít, ktoré v rámci systému vystupujú spolu s ich atribútami a vzťahmi medzi nimi.

3.1.1 Normalizácia

Kľúčovú úlohu pri návrhu zohráva *normalizácia*. Ide o množinu pravidiel, aplikovaním ktorých dochádza k dekompozícii jednotlivých tabuliek (použité pre reprezentáciu entít) za účelom zamedzenia redundacie dát či zlepšenia ich integrity. Normalizácia takisto prispieva i k efektívnejšiemu udržiavaniu samotnej databázy.

Normalizačných pravidiel (alebo foriem, skrátene NF) existuje viacero, najčastejšie používanými sú prvé tri, ktorých autorom je Dr. Edgar F. Codd. Ich nasledujúci popis je spracovaný podľa [6].

- **1. normálna forma**: Tabuľka je v 1. NF vtedy a len vtedy, ak každý jej atribút (sĺpec) obsahuje len atomické hodnoty, teda hodnoty, ktoré z pohľadu databázy nemožno ďalej deliť na menšie. Príkladom nesplnenia tejto normy môže byť napr. ukladanie mena a priezviska v rámci jedného stĺpca, pričom s každým z nich potrebujeme pracovať samostatne. Riešením je teda ich rozdelenie.

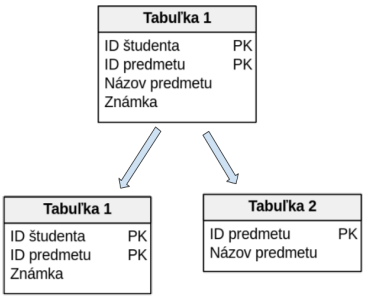
- **2. normálna forma**: Tabuľka je v 2. NF vtedy a len vtedy, ak spĺňa podmienky 1. NF, a každý jej nekľúčový atribút je závislý od kľúčového.

  Príkladom nesplnenia 2. NF je Tabuľka 1 na obrázku 3.1. Názov predmetu je závislý na predmete, avšak nie na študentovi. Riešením je teda vytvorenie novej tabuľky s ID a názvom predmetu.

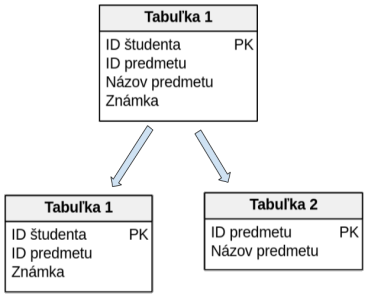
- **3. normálna forma**: Tabuľka je v 3. NF vtedy a len vtedy, ak spĺňa podmienky 2. NF, a všetky jej nekľúčové atribúty sú navzájom nezávislé.

  Tabuľka 1 na obrázku 3.2 je príkladom nesplnenia 3. NF. Názov predmetu je opäť závislý na predmete, ktorý je ale tentokrát nekľúčovým atribútom. Riešením je vytvorenie novej tabuľky s ID a názvom predmetu.

[1] `https://db-engines.com/en/ranking` (13.5.2017)



Obr. 3.1: Príklad použitia 2. normálnej formy



Obr. 3.2: Príklad použitia 3. normálnej formy

3.2 Typy databáz

V nasledujúcich podkapitolách sú stručne predstavené relačné a nerelačné typy databáz. Súčasťou je i výber konkrétnej databázy použitej v rámci práce.

3.2.1 Relačné databázy

Organizácia dát databáz tohto typu je založená na relačnom modely, navrhnutom E.F. Coddom v roku 1970 [1]. Dáta sú organizované do tabuliek (relácií) reprezentujúcich entitné typy. Stĺpce definujú atribúty danej entity a v riadkoch sú ukladané jej konkrétne inštancie, pričom každý z nich musí byť v rámci danej tabuľky unikátne adresovateľný.

Spolu so vzťahmi, vyjadrenými prostredníctvom primárnych a cudzích kľúčov, sú tabuľky súčasťou databázovej schémy, ktorú je nutné definovať ešte pred tým, ako je možné ukladať samotné dáta. Pre manipuláciu s nimi používa štandardizovaný jazyk Structure Query Language (SQL) .

Relačné databázy pre zabezpečenie integrity dát definujú štyri vlastnosti transakcií, v anglickom znení známych pod akronymom ACID - atomicitu, konzistenciu, izoláciu a trvácnosť. Nutnosť ich plnenia je však obmedzujúca pri horizontálnom škálovaní.

3.2.2 Nerelačné databázy

Základným rozdielom v porovnaní s relačnými je štruktúra ukladania dát. Podľa nej nerelačné (NoSQL) databázy ďalej delíme na dokumentové, grafové, key-value či stĺpcové.

Využitie nachádzajú najmä v oblasti Big Data technológií pri práci s neštrukturovanými dátami, kedy schému nemožno vopred zadefinovať.

Rozdielnym je taktiež prístup ku transakciám. Na úkor atomicity, trvácnosti a konzistencie dát naprieč systémom stavia NoSQL na ich dostupnosti a škálovateľnosti.

3.2.3 Voľba

Pre tvorbu nášho systému bol zvolený relačný typ databáz, najmä pre zaistenie konzistencie a integrity dát pri práci s nimi. Dôvodom je taktiež vopred známa schéma a použitie štandardizovaného jazyka SQL.

Konkrétnym databázovým systémom bol zvolený PostgreSQL [2]. Ide o open-source riešenie s aktívnou komunitou a so širokou podporou v rámci operačných systémov. Dodržuje SQL štandard a jeho výhodou je i podpora mnohých rozšírení. Jedným z nich je napr. PostGIS, rozšírenie pre možnosť práce s geografickými dátami, ktorého použitie je v rámci systému plánované.

Navrhnutému modelu databázy sa venuje kapitola 6.2.

[2] `https://www.postgresql.org/`

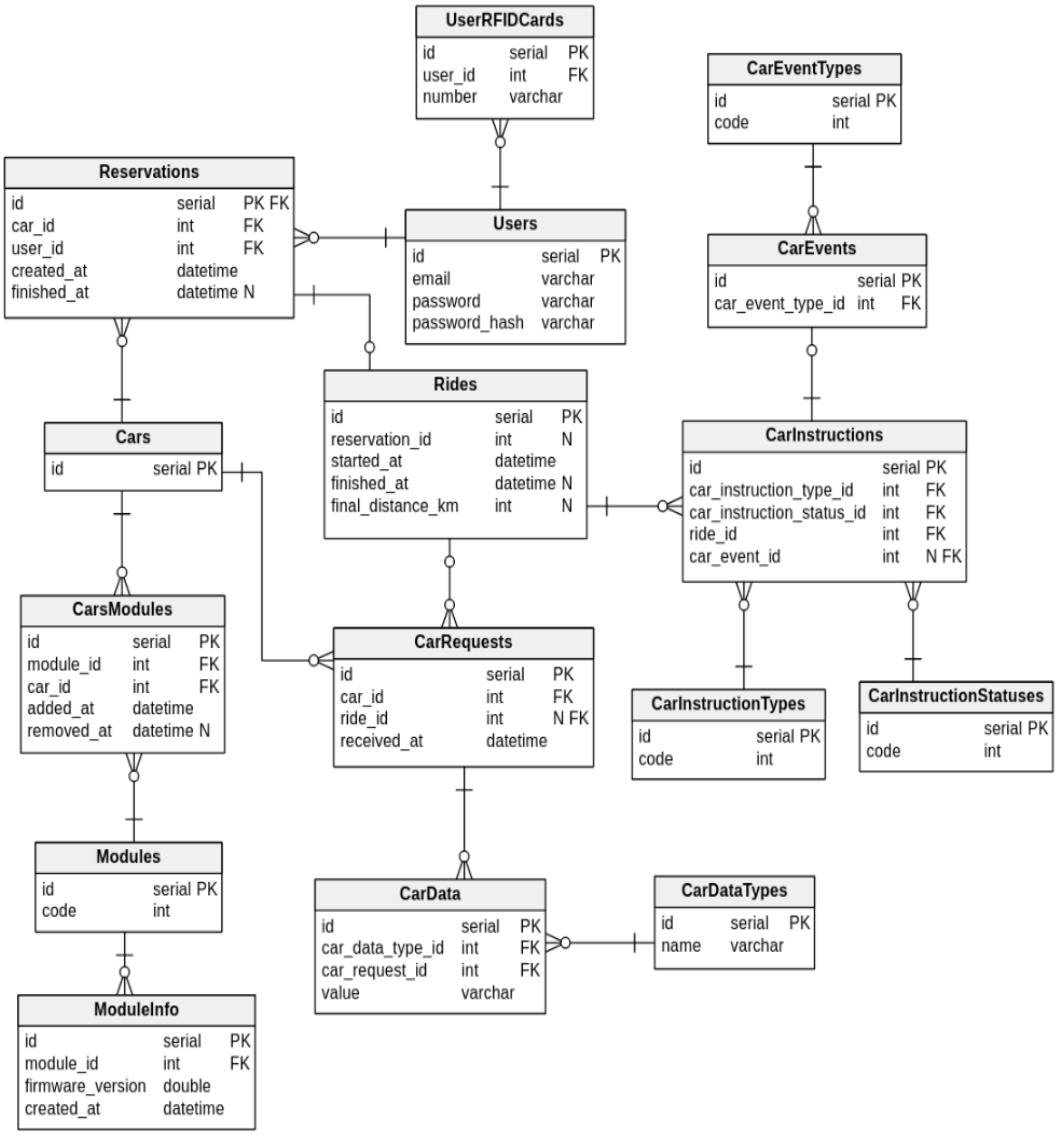
Kapitola 4

Implementácia

Kapitola popisuje proces implementácie databázového modelu s detailnejším popisom jeho časti nutnej pre správne fungovanie komunikácie s modulom. Taktiež sa venuje implementácii jednotlivých aspektov uvažovaných v rámci zmienenej komunikácie, presnejšie rozhrania a patričného zabezpečenia.

4.1 Databáza

Ako prvé bolo nutné identifikovať jednotlivé entity vystupujúce v systéme a ich vzájomné vzťahy, a to na základe požiadaviek, ktoré boli vypracované v rámci iných častí projektu.



Obr. 4.1: Časť databázového modelu

Na obrázku 4.1 je zobrazená časť modelu pre zaistenie hlavnej funkcionality pri komunikácii s modulom (celý model je umiestnený v prílohe).

Tabuľky obsahujú základné atribúty popísané názvom, typom a voliteľným špecifikátorom. Špecifikátor nadobúda hodnoty *PK* (primárny kľúč), *FK* (cudzí kľúč) a *N* (nepovinná hodnota). Pre zaznačenie vzťahov je použitá tzv. *Crow's foot* notácia[1].

4.1.1 Popis modelu

Jednotlivé moduly sú reprezentované tabuľkou *Modules*. Informácie o nich sú umiestnené v samostatnej tabuľke *ModuleInfo*, pričom jednému modulu môže prislúchať viacero záznamov tejto tabuľky. Takéto riešenie, použité i v prípade iných tabuliek (napr. *Users* či *Cars*, viď. príloha), umožňuje uchovávať okrem aktuálnych i predošlé dáta pri zmenách, čím je zabezpečená možnosť ich spätného dohľadania.

Vyjadrenie vzťahu modulu a vozidla, kedy s časovým odstupom môže byť modul použitý vo viacerých vozidlách, a naopak v vozidle použitých viac modulov, rieši tabuľka *CarModules*.

Jazdy vozidla eviduje tabuľka *Rides*. Podmienkou jej vytvorenia je existujúca rezervácia (v tabuľke *Reservations*), ktorá zároveň nesie informáciu o užívateľovi a vozidle. *Rides* obsahuje atribút s celkovou prejdenou vzdialenosťou, ktorá je nastavená po ukončení jazdy, aby sa predišlo jej opakovanému výpočtu.

Pre uloženie synchrónnych požiadaviek modulu 2.1.2 slúži tabuľka *CarRequests*. Okrem jázd je viazaná i priamo na tabuľku vozidiel, a to z dôvodu, že požiadavka môže byť zaslaná i v čase, kedy s vozidlom nikto nejazdí. Prevádzkové dáta vozidla ňou zaslané sú evidované v tabuľke *CarData*. Ich počet sa však môže líšiť v závislosti od jeho typu. Jednotlivé druhy parametrov sú preto ukladané v samostatnej tabuľke *CarDataTypes* a väzbou spojené s *CarData*, pričom hodnota parametru je ukladaná ako reťazec, a pred použitím vhodne pretypovaná. Takýmto spôsobom možno predísť plýtvaniu miesta v podobe nevyužitých atribútov tabuľky.

Tabuľkou *CarInstructions* sú reprezentované inštrukcie zasielané modulu. Ich možné typy sú uložené oddelene (tabuľka *CarInstructionType*), podobne ako pri už zmienených typoch parametrov vozidla. Toto riešenie umožňuje ich dynamické pridávanie a jednoduchú validáciu. Okrem typov je pri inštrukciách evidovaný i stav podľa štádia ich spracovania. Riešenie je obdobné ako pri typoch, teda ich odčlenenie do separátnej tabuľky.

Vznik inštrukcií môže byť podmienený priamou interakciou s vozidlom. Udalosti ňou vyvolané reprezetuje tabuľka *CarEvents*. Momentálne rozlišujeme iba jeden druh udalosti - použitie RFID čítačky užívateľom 2.1.2. Pri vyvolaní tejto udalosti a úspešnej validácii údajov na strane serveru je uložený záznam o nej. To implikuje vytvorenie príslušnej inštrukcie pre modul, čo naznačuje väzba medzi nimi.

[1] `http://www2.cs.uregina.ca/~bernatja/crowsfoot.html`

4.2 Komunikácia s modulom

Nasledujúce podkapitoly popisujú jednotlivé prístupové body implementovaného rozhrania a spôsobu zabezpečenia jeho časti určenej pre modul.

4.2.1 Rozhranie

Pre tvorbu rozhrania bola zvolená technológia REST ako návrhový štýl (2.2.1). Ako formát prenášaných dát bol vybraný JSON (2.2.2).

Vzhľadom na potreby komunikácie serveru s modulom boli implementované nasledujúce prístupové body.

4.2.1.1 Sychrónna požiadavka

Slúži pre zasielanie synchrónnych požiadaviek (2.1.2) modulu na server s nameranými prevádzkovými dátami vozidla. Kvôli zabezpečeniu je serverom vyžadovaný správny klientský SSL certifikát (2.3).

Metóda: **POST**
URI: **/api/modules/data**
Parametre (iba základné, môžno rozšíriť podľa typu vozidla):

Parameter	Povinný (x)	Dátový typ	Popis
timestamp	x	reťazec	čas odoslania (rrrr-MM-dd HH:mm:ss)
locked	x	boolean	príznak o zamknutí auta
gps.valid	x	booelan	príznak o valídnosti GPS dát
gps.data.lat	x	des. číslo	zemepisná šírka polohy vozidla
gps.data.lon	x	des. číslo	zemepisná dĺžka polohy vozidla
gps.data.speed	x	celé číslo	rýchlosť v km/h nameraná cez GPS
obd.valid	x	boolean	príznak o valídnosti OBD dát
obd.data.consumption	x	des. číslo	aktuálna spotreba v l/km
obd.data.fuelLevel	x	celé číslo	stav nádrže v %
obd.data.speed	x	des. číslo	rýchlosť v km/h nameraná cez OBD
obd.data.rpm	x	celé číslo	otáčky motora
obd.data.odometer	x	celé číslo	aktuálny stav odometra
obd.data.beBackBtn	x	boolean	príznak o vrátení užívateľa k vozidlu

Tabuľka 4.1: Tabuľka parametrov 1

Očakávaná odpoveď zo serveru :

```
{
    "canPark": true,
    "instruction" {
        "id": 1,
        "code": 2
    }
}
```

Tak, ako je zmienené v 2.1.2, odpoveď obsahuje príznak o možnosti parkovania vozidla v danej oblasti (*canPark*). Súčasťou je taktiež inštrukcia pre vykonanie, pozostávajúca z jej identifikátoru (*instId*) a kódu (instCode) reprezentujúceho jej význam. V prípade, že pre modul nie je vytvorená žiadna inštrukcia, majú parametre *instId* a *instCode* hodnotu -1.

4.2.1.2 Asynchrónna požiadavka

Slúži pre zasielanie asynchrónnej požiadavky (2.1.2) modulu na server pri interakcii užívateľa s vozidlom. Kvôli zabezpečeniu je serverom vyžadovaný správny klientský SSL certifikát (2.3).

Metóda: **POST**
URI: **/api/modules/events**
Parametre (iba základné, môžno rozšíriť podľa typu vozidla):

Parameter	Povinný (x)	Dátový typ	Popis
timestamp	x	reťazec	čas odoslania (rrrr-MM-dd HH:mm:ss)
type	x	celé číslo	číslo reprezentujúce typ udalosti
cardNumber	x	reťazec	číslo RFID karty užívateľa

Tabuľka 4.2: Tabuľka parametrov 2

Očakávaná odpoveď zo serveru:

```
{
    "canPark": true,
    "instruction" {
        "id": 1,
        "code": 2
    }
}
```

4.2.1.3 Vytvorenie inštrukcie

Slúži pre vytvorenie inštrukcie pre určené vozidlo použitím mobilnej aplikácie.

Metóda: **POST**
URI: **/api/instructions**
Parametre:

Parameter	Povinný (x)	Dátový typ	Popis
timestamp	x	reťazec	čas odoslania (rrrr-MM-dd HH:mm:ss)
type	x	celé číslo	číslo reprezentujúce typ udalosti
carId	x	celé číslo	identifikátor vozidla

Tabuľka 4.3: Tabuľka parametrov 3

Očakávaná odpoveď zo serveru:

```
{ "id": 1 }
```

4.2.1.4 Detail inštrukcie

Slúži pre získanie informácií o konkrétnej inštrukcii. Je využívaný mobilnou aplikáciou.

Metóda: **GET**
URI: **/api/instructions/{id}**
Parametre:

Parameter	Povinný (x)	Dátový typ	Popis
id	x	celé číslo	identifikátor inštrukcie

Tabuľka 4.4: Tabuľka parametrov 4

Očakávaná odpoveď zo serveru:

```
{ "instruction": {
      "id": 1,
      "status": {
          "id": 1,
          "name": "new"
}}}
```

4.2.1.5 Vytvorenie modulu

Slúži pre vytvorenie nového modulu v systéme.

Metóda: **POST**
URI: **/api/modules**
Parametre:

Parameter	Povinný (x)	Dátový typ	Popis
code	x	reťazec	kód modulu
firmwareVersion	x	des. číslo	číslo verzie firmware modulu

Tabuľka 4.5: Tabuľka parametrov 5

Očakávaná odpoveď zo serveru:

```
{ "id": 1 }
```

4.2.1.6 Detail modulu

Slúži pre získanie informácií o konkrétnom module.

Metóda: **GET**
URI: **/api/modules/{id}**
Parametre:

Parameter	Povinný (x)	Dátový typ	Popis
id	x	celé číslo	identifikátor modulu

Tabuľka 4.6: Tabuľka parametrov 6

Očakávaná odpoveď zo serveru:

```
{
    "id": 1,
    "code": 42,
    "info": {
        "firmwareVersion": 1.0
    }
}
```

4.2.2 Zabezpečenie

Pre šifrovanie prenosu dát a autentifikáciu modulov v systéme bolo použité HTTPS (2.3.1). Krokmi jeho implementácie boli:

- vytvorenie potrebných certifikátov,
- konfigurácia serveru.

4.2.2.1 Certifikáty

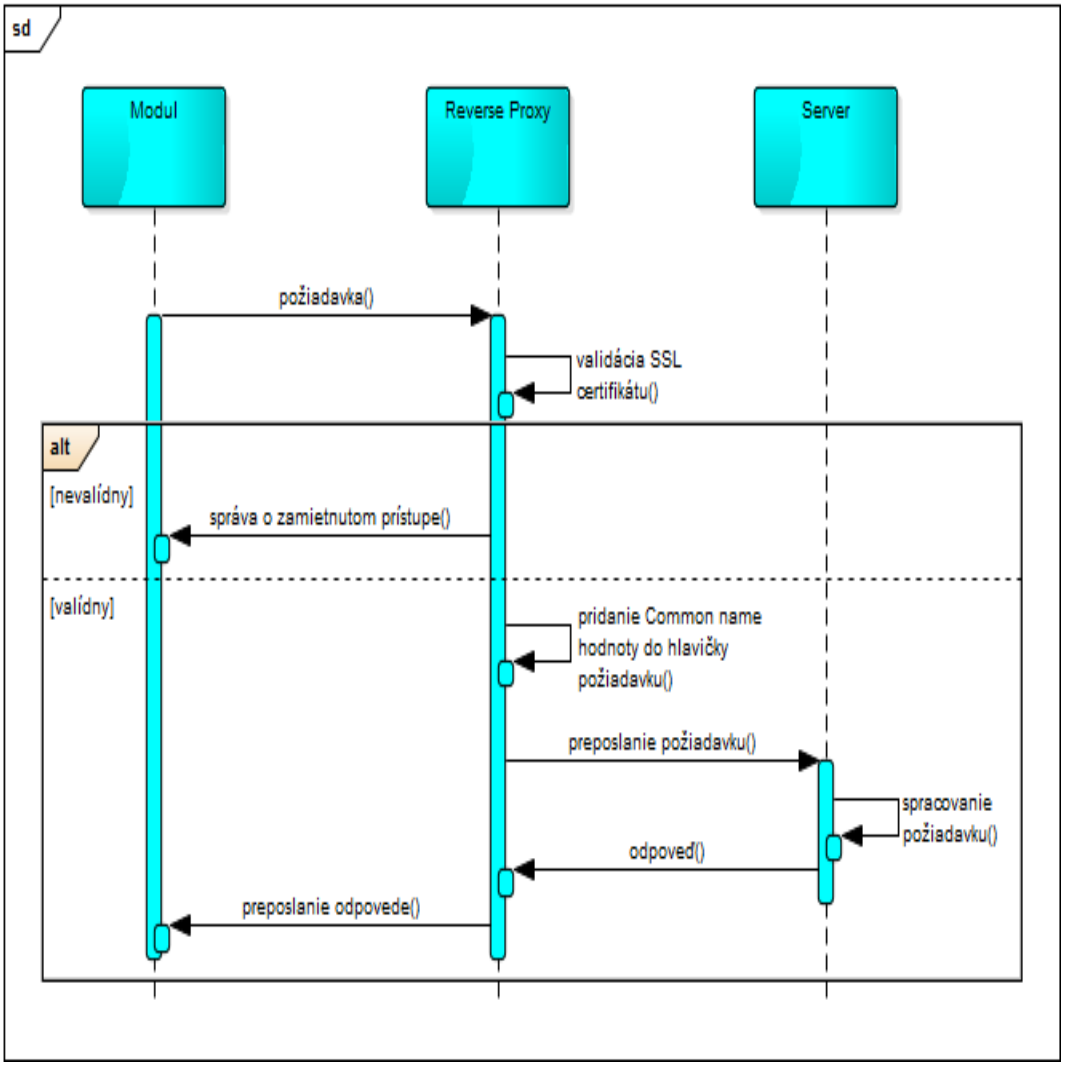
Pre vytvorenie certifikátov bol použitý nástroj EasyRSA (verzia 3.0.1)[2]. Vzhľadom na podporu zo strany modulu bola pre asymetrické šifrovanie výmeny kľúča zvolená šifra RSA s 2048 bitovou dĺžkou kľúča, poskytujúca dostatočnú úroveň zabezpečenia[7]. Prostredníctvom EasyRSA bola vygenerovaná certifikačná autorita a dva certifikáty touto autoritou podpísané. Jeden určený pre server a druhý pre modul. Pre potreby autentifikácie bol do parametru certifikátu *Common Name* umiestnený kód modulu. Jeho následné spracovanie po odoslaní je popísané v ďalšej časti.

4.2.2.2 Konfigurácia serveru

Pred tým, ako je akákoľvek požiadavka spracovaná serverom, prechádza cez *reverse proxy* vrstvu, konkrétne implementovanou pomocou Apache 2.4[3]. Apache je bezplatným riešením, podporovaným na mnohých OS. Jednou z jeho úloh je práve autentifikácia modulov na základe SSL certifikátov porovnaním ich certifikačnej autority voči nami vytvorenej. Valídnu požiadavku potom preposiela ďalej na server, pričom do jej hlavičky je umiestnená hodnota parametru Common name, obsahujúca kód modulu pre jeho rozpoznanie. Priebeh tejto komunikácie znázorňuje sekvenčný diagram 4.2.

Okrem zmienenej autentifikácie plní proxy i úlohu presmerovania nešifrovanej HTTP komunikácie na šifrovanú HTTPS. Pre potreby správcovskej webovej aplikácie je poskytovateľom jej statického obsahu. Zároveň bolo nutné správne ošetrenie ňou zaslaného požiadavku na URL odlišnú od používaného rozhrania (spôsobené znovunačítaním stránky). Pri jej vývoji Apache taktiež poslúžil pri riešení *Cross-Origin Resource Sharing* (CORS) problému.

[2] `https://github.com/OpenVPN/easy-rsa`
[3] `https://httpd.apache.org/`



Obr. 4.2: Diagram komunikácie s využitím proxy

4.3 Použitie frameworku

Rozhranie spolu s navrhnutým modelom databázy boli ďalej implementované použitím frameworku Play[4]. Ide o open-source MVC (Model View Controller) framework umožňujúci tvorbu webových aplikácií v jazyku Java, so zabudovanými testovacími nástrojmi (napr. JUnit, Mockito), podporou pre REST a prácu s JSON. Umožňuje tzv. *hot-reload* konfiguračných a zdrojových súborov, ktorý spolu so zobrazovaním kompilačných a behových chýb aplikácie priamo v prehliadači uľahčuje jej vývoj. Je komerčne podporovaný a dobre zdokumentovaný.

[4] `https://playframework.com/`

4.3.1 Architektúra aplikácie

Z dôvodu ľahšej udržateľnosti a väčšej prehľadnosti bola aplikácia po stránke štruktúry rozdelená do viacerých vrstiev. Každá z nich zodpovedá za vykonanie odlišnej časti aplikačnej logiky pri spracovávaní požiadavky a je reprezentovaná iným typom programovej komponenty. Danými komponentami sú *Router*, *Controller*, *Service* a *DAO*, bližšie popísané v ďalších častiach.

Závislosť použitia jednej komponenty v rámci druhej je riešená prostrednítvom *Dependency Injection*. Ide o návrhový vzor, ktorého podstatou je oddelenie rozhrania od jeho konkrétnej implementácie. Výhodou je zníženie počtu priamych závislostí a najmä lepšia testovateľnost.

4.3.1.1 Router

V konfiguračnom súbore *routes* bolo potrebné definovať jednotlivé REST prístupové body, ktoré je možné adresovať spolu s ich mapovaním na príslušné metódy tried v Controller vrstve. Router, komponenta manipulujúca s prijatou požiadavkou ako prvá, na základe jej HTTP metódy a URL volá vhodne zvolenú Controller metódu.

4.3.1.2 Controller

Jednou z úloh tejto vrstvy je typová validácia hodnôt dát zaslaných požiadavkou. V prípade úspešnej validácie sú komplexnejšie dáta (nejedná sa len o jeden údaj, napr. modulom namerané dáta) ukladané do objektu príslušnej DTO (Data Transfer Object) triedy; v prípade primitívnych typov sú použité ich objektové reprezentácie.

DTO objekty slúžia ako uniformný spôsob prenosu komplexnejších dát medzi Controller a Service vrstvou. Neimplementujú žiadnu business logiku. Umožňujú iba získavanie a nastavovanie hodnôt jednotlivých atribútov.

Controller vrstva je zároveň zodpovedná za formu, v akej sú dáta zaslané späť klientovi. V našom prípade sa jedná o JSON formát.

Zabezpečenie prístupu ku metódam tried tejto vrstvy, a teda prístupu do aplikácie, je riešené prostredníctvom triedy *ModuleSecurity*, ktorá kontroluje hodnotu v hlavičke požiadavky pridanú proxy serverom (4.2.2.2).

4.3.1.3 Service

Service vrstva je zodpovedná za vykonávanie business validácie a logiky (v prípade modulu ide napr. o vytvorenie inštrukcie) nad dátami prijatými z Controller vrstvy. Neúspešná validácia je v mnohých prípadoch riešená „vyhodením“ príslušnej, nami vytvorenej výnimky, ktorá je spracovaná na úrovni Controller vrstvy, a následne vhodne prezentovaná klientovi.

Service triedy taktiež pracujú i s objektami *Model* tried. Model triedy slúžia pre programovú reprezentáciu tabuliek databázy, pričom jednotlivé vlastnosti a väzby medzi nimi sú vyjadrené prostredníctvom atribútov týchto tried.

Každá Service trieda rozširuje nami vytvorenú triedu *BaseService* s využitím generického určenia typu identifikátoru a Model triedy. Podľa nich vytvára objekt triedy *BaseDao*, pomocou ktorého v implementovaných metódach rozhrania *GenericService* pracuje s perzistentnou vrstvou. Tá je bližšie popísaná v 4.3.1.4. Pri potrebe využitia objektu inej DAO triedy je použitá jeho príslušná Service trieda, ktorá zodpovedá i za validáciu vstupných parametrov. Takýmto spôsobom docielime väčšiu prehľadnosť a zamedzíme opakovanej implementácii.

4.3.1.4 Data Access Object (DAO)

Ide o poslednú aplikačnú vrstvu, ktorej úlohou je vykonávať operácie čítania, zápisu či mazania nad perzistentným úložiskom - databázou. Záznamy z nej sú pomocou Ebean ORM mapované na Model objekty, ktoré sú následne vrátené späť do Service vrstvy pre spracovanie.

Každá DAO trieda rozširuje nami vytvorenú abstraktnú *BaseDao* triedu s využitím generického určenia typu identifikátoru a Model triedy. Podľa nich je vytvorený objekt triedy Finder slúžiaci ako abstrakcia nad pripojenou databázou poskytujúci základné CRUD (Create Read Upadate Delete) operácie. BaseDao ďalej implementuje metódy rozhrania *GenericDao* slúžiace pre prácu so zmieneným Finder objektom, spoločné pre všetky Dao triedy.

Controller, Service a DAO triedy boli vytvorené pre jednotlivé entity databázového modelu (4.1).

Kapitola 5

Testovanie

Súčasťou práce bolo okrem implementácie rozhrania pre komunikáciu s modulom i testovanie jeho správnej funkčnosti s využitím jednotkových a integračných testov. Pri ich tvorbe boli použité tieto frameworky:

- JUnit[1] - open-source framework umožňujúci vytváranie a spúšťanie testov napísaných v jazyku Java. Použitý bol JUnit s verziou 4.12.
- Mockito[2] - open-source framework, slúžiaci na vytváranie *mockov* - zástupných objektov. Ich hlavným rysom je, že sa navonok tvária ako obyčajné objekty, avšak ich funkcionalitu možno ľubovoľne definovať.

5.1 Test autentifikácie

Testovanou bola i autentifikácia modulov v systéme, ktorá je realizovaná prostredníctvom SSL certifikátov (4.2.2).

V rámci testu boli použité dva certfikáty, pričom iba jeden z nich bol podpísaný nami vytvorenou certifikačnou autoritou. Pomocou *cURL*[3], terminálového nástroja slúžiaceho na prenos dát s podporou mnohých protokolov, vrátane HTTPS, boli následne vykonané požiadavky na serverovú url vyžadujúcu SSL autentifikáciu.

Výsledkom použitia nevhodného certifikátu bola očakávaná chyba pri naväzovaní šifrovaného spojenia, naopak použitím certifikátu podpísaného našou autoritou sme docielili požadovaného výsledku.

5.2 Unit testy

Unit test je časť kódu (zvyčajne metóda), ktorá vyvolá spustenie iného kódu, a následne kontroluje správnosť istých predpokladov. Ak sú predpoklady nesprávne, unit test zlyhal. Unit testy by mali byť automatizované, opakovateľné, ľahko implementovateľné, a ich vykonanie by malo byť rýchle [3].

V rámci práce boli implementované unit testy pre Service vrstvu. Každý z nich pozostáva z následujúcich krokov.

1. Vytvorenie mock objektov tried, na ktorých je Service trieda v rámci testovanej metódy závislá. Je nutné definovať návratové hodnoty volaných metód či prípadnú zmenu zaslaných parametrov vo funkcii bez návratovej hodnoty.
2. Priradenie vytvorených mock objektov za závislé rozhrania testovanej Service triedy.
3. Vyvolanie testovanej metódy danej Service triedy (v prípade potreby s vhodne zvolenými parametrami).

[1] `http://junit.org/junit4/`
[2] `http://site.mockito.org/`
[3] `https://curl.haxx.se/`

4. Kontrola správnosti predpokladov s využitím JUnit.

Uvedený je príklad testu vyhľadania modulu v systéme podľa kódu za predpokladu, že modul s daným kódom nie je evidovaný.

```
@Test
public void testFindByCodeIfNotExists() {
   ModuleDaoA dao = Mockito.mock(ModuleDaoA.class);
   Mockito.when(dao.findByCode(Matchers.anyString()))
          .thenReturn(null);

   moduleService.setModuleDao(dao);

   Module result = moduleService.findByCode("1");
   assertNull(result);
}
```

5.3 Integračné testy

Ich podstata spočíva v otestovaní dvoch alebo viacerých navzájom závislých modulov ako celok [3]. V našom prípade moduly predstavujú jednotlivé vrstvy aplikácie (viď 4.3.1).

Pri implementácii integračných testov v rámci práce bol použitý testovací webový server Netty zabudovaný priamo vo frameworku Play so spustenou inštanciou našej aplikácie. Vytvorená bola taktiež testovacia databáza PostgreSQL naplnená dátami využítými v rámci testov. Pre pripojenie na testovací server bol použitý WSClient, asynchrónny HTTP klient, ktorý je taktiež súčasťou frameworku.

Každý z testov pozostáva z nasledujúcich krokov:

1. vykonanie požiadavky prostredníctvom zmieneného HTTP klienta na testovanú, url

2. kontrola správnosti predpokladov na vrátenej odpovedi s využitím JUnit.

Uvedený je príklad testu vrátenia chybového HTTP kódu 400 pri vyhľadaní modulu s neexistujúcim id. Zároveň testuje, či text vrátenej správy odpovedá textu výnimky, ktorá má byť v tomto prípade „vyhodená“ na úrovni Service triedy.

```
@Test
public void testGetModuleWrongId() throws Exception {
   try {
      String url = getBaseUrl() + "/modules/-1";
      CompletionStage<WSResponse> stage = client.url(url).get();
      WSResponse response = stage.toCompletableFuture().get();
      assertEquals(Http.Status.BAD_REQUEST, response.getStatus());
      Exception e = new EntityNotFoundException(Module.class, "id", -1);
```

```
      assertEquals(e.getMessage(), response.asJson().asText());
   } catch (InterruptedException e) {
      e.printStackTrace();
   }
}
```

5.4 Simulácia modulu

V rámci projektu bol taktiež v jazyku Python implementovaný program pre simuláciu modulu používaného vo vozidlách. Jeho aktuálna verzia umožňuje:

- zasielanie synchrónnych požiadaviek (viď 2.1.2) v definovaných intervaloch s prednastavenými hodnotami dát. Taktiež umožňuje použitie dát získaných z reálne vykonaných jázd vozidlom.
- dynamické zmeny statusu vozidla a jeho odomknutia počas zasielania požiadaviek.

Program zatiaľ neobsahuje použitie SSL certifikátov pre autentifikáciu a možnosť dynamického nastavenia väčšiny typov zasielaných dát.

Kapitola 6

Záver

6.1 Zhodnotenie práce

Výsledkom práce je implementované rozhranie pre komunikáciu s modulom umiestneným vo vozidlách, využívaných v rámci projektu Uniqway. Vďaka nemu je umožnená interakcia s vozidlom v zmysle odomykania/zamykania, a to prostredníctvom RFID čítačky či mobilnej aplikácie. Vytvorené rozhranie zároveň umožňuje modulu pravidelné zasielanie prevádzkových dát vozidla na server.

Pre zabezpečenie spojenia bola zvolená SSL autentifikácia použitím nami vytvorených certifikátov. Vykonávaná je na úrovni Apache reverse-proxy, taktiež implementovanom v rámci práce. Okrem zmienenej autentifikácie je využívaný i pre smerovanie nezabezpečenej komunikácie na zabezpečenú či pre poskytovanie statického obsahu webovej aplikácie.

Pre perzistenciu dát v rámci systému bola použitá relačná databáza PostgreSQL. Pri jej návrhu bolo prihliadané na možnosť dlhodobej evidencie dát a možnosť ich spätného dohľadania, napr. z dôvodu neskoršej analýzy či potrebného auditu.

Implementované rozhranie bolo taktiež otestované jednotkovými a integračnými testami. Tie poslúžili pre overenie funkčnosti serverovej aplikácie, no ich výhodou je najmä možnosť použitia v rámci ďalšieho vývoja pre kontrolu spätnej kompatibility. Pre testovacie účely bol implementovaný taktiež program na simuláciu modulu, avšak nie ako súčasť tejto práce.

6.2 Možnosti ďalšieho vývoja

V rámci ďalšieho vývoja systému navrhujem nasledovné.

- Serverový SSL certifikát nahradiť iným, podpísaným niektorou z uznávaných certifikačných autorít, keďže nami vytvorená nie je registrovaná žiadnym z prehliadačov,
- Implementácia tzv. geofencingu - kontroly výskytu vozidla vo vopred určených oblastiach na základe ich GPS polohy. Pre tento účel možno využiť PostGIS, ktorý je rozšírením použitej databázy PostgreSQL.
- Rozšíriť rozhranie o možnosť filtrovania dát vrátených v odpovedi na základe parametrov uvedených v požiadavke, najmä z dôvodu zrýchlenie ich prenosu, a teda i celkového zrýchlenia aplikácie. Výhodou je taktiež šetrenie dátových paušálov v prípade mobilných aplikácii.

Zoznam bibliografických odkazov

[1] CODD, E. F. A relational model of data for large shared data banks. *Communications of the ACM*. 1970, 13, 6, s. 377–387.

[2] FIELDING, R. N. R. T. a. T. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[3] OSHEROVE, R. *The art of unit testing with examples in .NET*. Manning Publications Co., 2009. ISBN 978-1-933988-27-6.

[4] POTTI, P. K. On the design of web services: SOAP vs. REST. 2011.

[5] SHAHEEN S, C. N. Mobility and the sharing economy: impacts synopsis, 2015.

[6] T., W. R. *Data Management: Databases and Organizations (5th edition)*. Wiley, 2005. ISBN 0471715360.

[7] YADAV, M. A comparative study of performance analysis of various encryption algorithms. *International Conference On Emanations in Modern Technology and Engineering*. 2017, 5, 3.

[8] ZHAO, R. a. M. S. a. B. L. L. a. I. Anatomy and performance of SSL processing. In *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, s. 197–206. IEEE, 2005.

Dodatok A

Zoznam použitých skratiek

API Application Programming Interface

CA Certificate Authority

GPRS General Packet Radio Service

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

ORM Object Relational Mapping

OS Operačný systém

OSI Open Systems Inteconnection

RFID Radio Frequency IDentification

SQL Structured Query Language

SSL Secure Socket Layer

VŠE Vysoká škola ekonomická

W3C World Wide Web Consortium

XML eXtensible Markup Language

ČVUT České vysoké učení technické

ČZU Česká zemědělská univerzita

Dodatok B

Modul

Hlavnú časť prototypu modulu (viď obrázok B.1) tvorí procesor STM32F415RG. Na doske prototypu sú taktiež umiestnené napájacie obvody, ktoré vytvárajú za palubnej siete vozidla potrebné napäťové úrovne pre všetky periférie hardvérových častí.

Ďalej je doska vybavená niekoľkými komunikačnými perifériami. Periféria rozhrania CAN 2.0B umožňuje pripojenie prototypu z palubnej zbernici vozidla. Z tejto zbernice je prototyp schopný čítať potrebné dáta a počas jazdy ich periodicky odosielať na server.

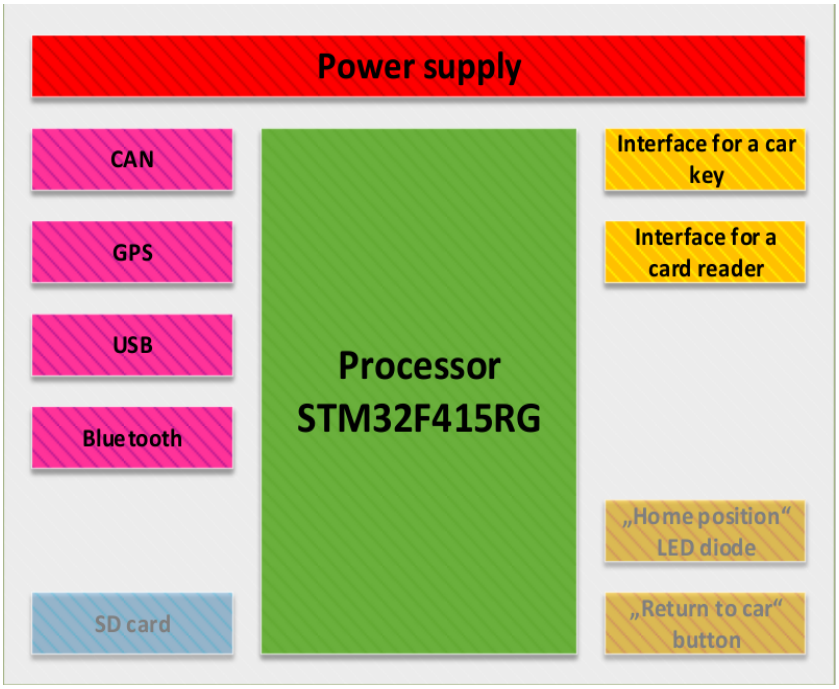
Ďalšou komunikačnou perifériou je rozhranie USB. Pomocou tohto rozhrania je možné prototyp pripojiť k PC a prostredníctvom k tomuto účelu špeciálne navrhnutej aplikácie aktualizovať firmvér prototypu. Rozhranie Bluetooth nie je v prípade prototypu využité na žiadny prenos dát medzi užívateľom a systémom. Toto rozhranie slúži na overenie fyzickej prítomnosti užívateľa pri vozidle v prípade, že sa užívateľ chystá odomknúť vozidlo prostredníctvom mobilnej aplikácie. Ďalšou perifériou je prijímač GPS signálu, ktorý zaisťuje zisťovanie zemepisnej polohy vozidla.

Ďalším rozhraním prototypu je rozhranie pre pripojenie kľúča vozidla. Rozhranie tvorí sada polovodičových spínačov. Týmto rozhraním je prototypu umožnené odomykanie a uzamykanie vozidla bez potreby zasahovať do komunikácie riadiacej jednotky.

Pre pripojenie čítačky RFID kariet je na doske prototypu umiestnené rozhranie RS232. Nad týmto fyzickým rozhraním je implementovaný protokol, ktorý umožňuje prenos načítaných ID kariet ako aj inicializáciu akustickej a svetelnej indikácie priamo na čítačke kariet.

Prototyp je ďalej vybavený rozhraním pre MicroSD kartu. Kartu je možné odnímať z čelného panelu prototypu bez nutnosti demontáže prototypu. V tejto fáze projektu zatiaľ táto funkcia nebola využitá, ale v budúcnosti sa počíta s využitím logovania nameraných prevádzkových dát.

Ďalším zatiaľ nevyužitým rozhraním je tzv. „Home position led“. Táto led dióda by mala v budúcnosti signalizovať vodičovi, že sa nachádza v oblasti, kde môže vozidla zaparkovať bez sankcie zo strany správy systému. Jedná sa o prípad, kedy systém umožňuje zanechať vozidlo len obmedzenom množstve parkovacích miest. Taktiež sa počíta s tlačidlom, ktoré umožní užívateľovi oznámiť systému, že sa k vozidlu chystá vrátiť aj po jeho uzamknutí a systém ho neuvoľní inému užívateľovi.



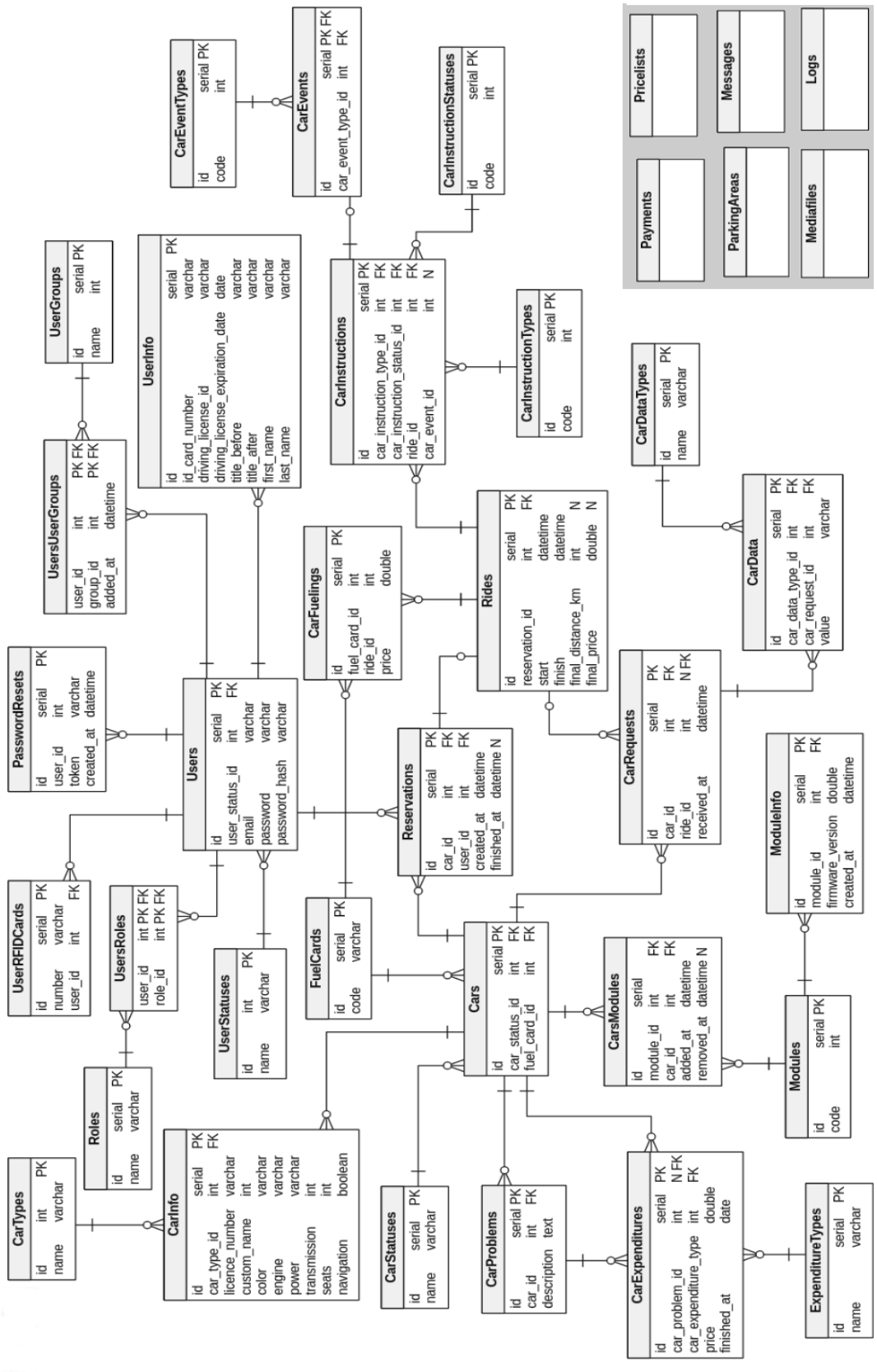
Obr. B.1: Bloková schéma modulu

Dodatok C

Rozšírený databázový model

Na obrázku C.1 je zobrazený rozšírená verzia databázového modelu nášho systému. Šedým pozadím sú vyznačené tabuľky, ktoré ešte neboli do systému zakomponované, ako

- ParkingAreas - pre uloženie povolených parkovacích oblastí pre vozidlá,
- Pricelists - pre evidenciu cenníkov jázd,
- Messages - pre správy generované systémom a zobrazované správcom vozového parku,
- Logs - pre evidenciu systémových záznamov (napr. o zmenách stavu vozidla či užívateľa),
- Mediafiles - pre uloženie meta dát o súboroch nahratých do systému (napr. faktúry nákladov spojených s pravidelnou údržbou vozidiel),
- Payments - pre evidenciu platieb jednotlivých rezervácií.



Obr. C.1: Rozšírený databázový model