Objektovo orientovaná analýza a návrh softvéru

Informačný systém autoservisu

Ing. Lukáš Doubravský

Streda 16:00

Alan Kováč, Richard Križan

21. Marec, 2020

Objektovo orientovaná analýza a návrh softvéru **0**

Informačný systém autoservisu **0**

Motivacia: 4
Opis Projektu: 4
Funkcionálne požiadavky: 5
Identifikované triedy: 5
Architektonické štýly a návrhové vzory: 5
Roly: 6
Prijímací technik 6
Mechanik 6
Zakaznik 7
Používateľ 7
Systém 7
Use cases: 8
Use case diagram: 8
Use case popis: 8
UC01 Registracia zakaznika 9
UC02 Prihlasenie zakaznika 9
UC03 Pridanie vozidla do garaze 9
UC04 Výber vozidla z garáže zákazníka 9
UC05 Odovzdanie vozidla 10
UC06 Vstupná obhliadka 10
UC07 Vytvorenie objednávky na servis 10
UC08 Akceptovanie objednávky servisu 11
UC09 Pridelenie vozidla na servis mechanikovi 12
UC10 Ukončenie objednávky servisu 12
UC11 Manažment skladových zásob 12
UC12 Vykonanie servisu vozidla 12
UC13 Ukončenie servisu vozidla 13
UC14 Vystavenie faktúry za servis 13
UC15 Notifikácia o ukončení servisu 13
UC16 Platba za servis 14
UC17 Štatistika oprav vozidla 15
Activity diagrams: 16
AD01 Pridanie vozidlo do garáže 16
AD02 Výber vozidla z garáže 17
AD03 Prihlásenie používateľa 18
State diagrams: 19
SD01 Vozidlo 19
SD02 Prihlásenie používateľa 20
Sequence diagrams: 21

SqD01 Pridanie vozidla do garáže 21
SqD02 Výber vozidla z garáže 22
SqD03 Prihlásenie používateľa (záložný) 23
Class diagram 24
Celkový 24
Sketch dátový model 24
Vzory 24
Facade pattern 25
Diagram 25
Kód 25
Builder pattern 26
Diagram 26
Kód 26
Composite pattern 27
Diagram 27
Kód 27
Factory method pattern 28
Diagram 28
Kód 28
Template method pattern 29
Diagram 29
Kód 29
Null object pattern 31
Diagram 31
Kód 31
Observer 32
Diagram 32
Kód 32
State pattern 33
Diagram 33
Kód 33
Proxy pattern 34
Diagram 34
Kód 34
Singleton 35
Diagram 35
Kód 35
Záver 36
Príloha A: Organizácia zložiek 37

Motivacia:

V tomto predmete sa zameriame na vytvorenie nového informačného systému pre servis motorovych vozidiel. Takýto systém bude mať za úlohu zjednodušiť plánovanie, evidenciu nových objednávok a historický prehľad dokončených objednávok. Taktiež alokovanie a pridelenie pracovnej sily pre vykonanie servisu nad schválenou objednávkou až po fakturovanie a platbu za servisované vozidlo. Informačný systém bude forme webovej aplikácie nasadený na aplikačnom servery. Aplikácia bude založená na klient-server architektúre, kde v konečnom dôsledku bude na serverovej časti aplikovaný vzor MVC, taktiež budú časti aplikácie založené na vrstvovej architektúre (autentifikačná vrstva, 2x databázová vrstva, gateway vrstva, biznis vrstva a zálohovacia vrstva). Projekt budeme vyvíjať v objektovom programovacom jazyku JAVA 8 na backendovej časti a javascript na frontendovej časti. Odhadujeme, že projekt bude zhruba obsahovať 10 až 30 tried rozdelených do asi 7-10 balikov (account, stock, orders, planning, reporting, invoice, payment), podľa typu použitia a závislostí.

Opis Projektu:

Informačný systém by mal pokrývať bežné úkony od príjmu automobilu s popisom závady na servis, vstupnú prehliadku automobilu, vykonanie servis až po prevzatie opraveného automobilu zákazníkom a platbu za opravu. Platbu za opravu bude možne realizovať online alebo fyzicky v servise, pričom ju môže vykonávať zákazník alebo poisťovňa po ukončení servisnej objednávky. Pomocou informačného systému by technik mal vedieť komunikovať so zákazníkom (pri nejakom nečakanom výdavku), ako aj viesť skladové hospodárstvo náhradných dielov. Servisný technik by taktiež mal mať možnosť pre plánovanie práce rôznych mechanikov. Informačný systém by mal taktiež zákazníkovi po určitom čase poslať správu so žiadosťou o vyplnenie dotazníku spokojnosti (feedback). Informačný systém taktiež podporuje registráciu zákazníkov, kde každý zákazník má prístupnú sekciu “moja garáž”, kde si vie podľa VIN čísla pridať svoje vozidlo, na ktoré si vie neskôr objednať servis, ktorý technik následne priradí mechanikovi.

Cieľom aplikácie je zrýchliť a zefektívniť čakaciu dobu objednávok na servis, vykonávanie a evidovanie opravy vozidla s výkazom zamestnanca. Docieliť chceme taktiež odosielanie automatických notifikácií s informáciami o stave servisovaného vozidla (predĺženie servisu, ukončenie servisu a pod.). Zabezpečiť chceme poskytovanie štatistík servisov a prehľadu objednávok pre celý servis ale aj pre jednotlivých zákazníkov a ich registrované autá.

Funkcionálne požiadavky:

- Softvér disponuje planovacím manažmentom servisu
- Softvér disponuje manažmentom skladových zásob
- Softvér disponuje Webovým rozhraním a platobnou bránou
- Softvér disponuje štatistickým modulom
- Softvér disponuje správou používateľských účtov a objektov k ním prislúchajúcich
- Softvér disponuje booking systémom
- Softvér disponuje modulom pre poisťovne

Identifikované triedy:

Z počiatku sme identifikovali dátové entity, vhodné ako triedy:

- Používateľ (zákazník, mechanik, servisný technik)
- Auto (osobné, nákladné, ..)
- Garáž
- Objednávka
- Servis
- Sklad
- Diely
- Platba
- Notifikácia
- Faktúra
- Poisťovňa

Architektonické štýly a návrhové vzory:

- Observer
  Využitie tohto vzoru vidíme vo viacerých častiach projektu. Napríklad samotné notifikovanie zákazníka, alebo rôzne funkcie notifikujúce ine časti k správnemu dosiahnutiu toku procesov.
- Facade
  Využitie tohto vzoru vidíme vo viacerých funkciách, ktoré sa opakujú a spolu súvisia. Tento vzor nám pomôže jednotlive implementácie jednoducho a efektívne prepojiť.
- Strategy
  Tento vzor by sme vedeli použiť napríklad v štatistikách pre rôzne druhy vozidiel.
- Template method
  Podobne aj pri tomto vzore vidíme rôzne možnosti implementácie pre abstraktnej triedy pre rôzne druhy platby, vozidiel alebo podobne.
- Dependecy injector
  Je jedným zo stavebných kameňov Spring frameworku. Tento framework použijeme na injection potrebných rozhraní do servisnej časti.

- Builder
  Vieme ho dobre použiť v spojení s predchádzajúcim vzorom na objektov, ktoré môžeme napríklad injectovať.
- Authorize
  Na autorizáciu používateľov využívame vzor authorize. Používateľ sa pred vykonaním akcie dopytuje na autorizačnú entitu, či má práva na vykonanie akcie. Týmto sa nám podarilo zabezpečiť bezpečnosť aplikácie.
- Interceptor
  HTTP interceptor pre držanie páru tokenov po autorizácii a ak prístupový token expiruje, zavolá sa callback, ktorý vygeneruje nový token.
- ActiveRecord
  Pre prístup do DB sme sa rozhodli použiť Active Record, ktorý nás bez vynaloženia väčšieho úsilia zbaví bezpečnostných dier a zabezpečí nám extremne jednoduchú vymeniteľnosť databázovej súčiastky do niekoľkých sekúnd
- Proxy
  Pre lepšie debugovanie a schopnosť dohľadania chyby sme mali za cieľ dobré zaznamenávanie udalostí v našom systéme. Uchovávanie udalostí na nižšej úrovni - uchovávanie requestov, časy a podobne. Po iteráciach sme dospeli k riešeniu skrze vzor proxy, ktorý nám zabezpečuje hneď 2 veci:
    - Možnosť zachytávania udalostí pred a po vykonaní requestu v biznis logike.
    - Možnosť filtrovania potrebných atribútov z requestu na uchovanie udalostí - nie vždy ukladáme všetky informácie z requestu, to by bolo príliš pamäťovo náročné.
- Batch
  Pre odosielanie notifikácii zákazníkom využívame batch sequential, keďže notifikácie a správy zákazníkom sa nám v závislosti od veľkosti servisu môžu nazhromaždiť na niekoľko stoviek kusov.

Roly:

- Prijímací technik

  Úlohou roly je celkový manažment aut, objednávok, servisov, skladu a účtovníctva. Taktiež môže vytvárať nové a akceptovať zákazníkové objednávky realizované cez internet. Po prijatí objednávky, plánuje prácu a priraduje mechanikom autá na servis. Táto rola má najvyššie práva pri ukončení a fakturovaní servisných prác.

- Mechanik

  Táto rola sa venuje čisto len realizovaniu servisu na aute. Mechanik začína svoju prácu potom ako mu prijimaci technik pridelí auto. Mechanik vámci priebehu servisu dokáže do systému zaznamenávať potrebné zásahy a pridávať použité diely zo skladu.

- Zakaznik

  Dokáže registrovať svoje vlastné autá do garáže a následne cez internet objednať auto servis. Po potvrdení termínu objednávky bude zákazník informovaný a vyzvaný na fyzické odovzdanie vozidla. Naopak pri dokončení servisu bude informovaný o ukončení s finalnou cenou. Cenu za opravu môže zákazník platiť cez internet alebo servise.

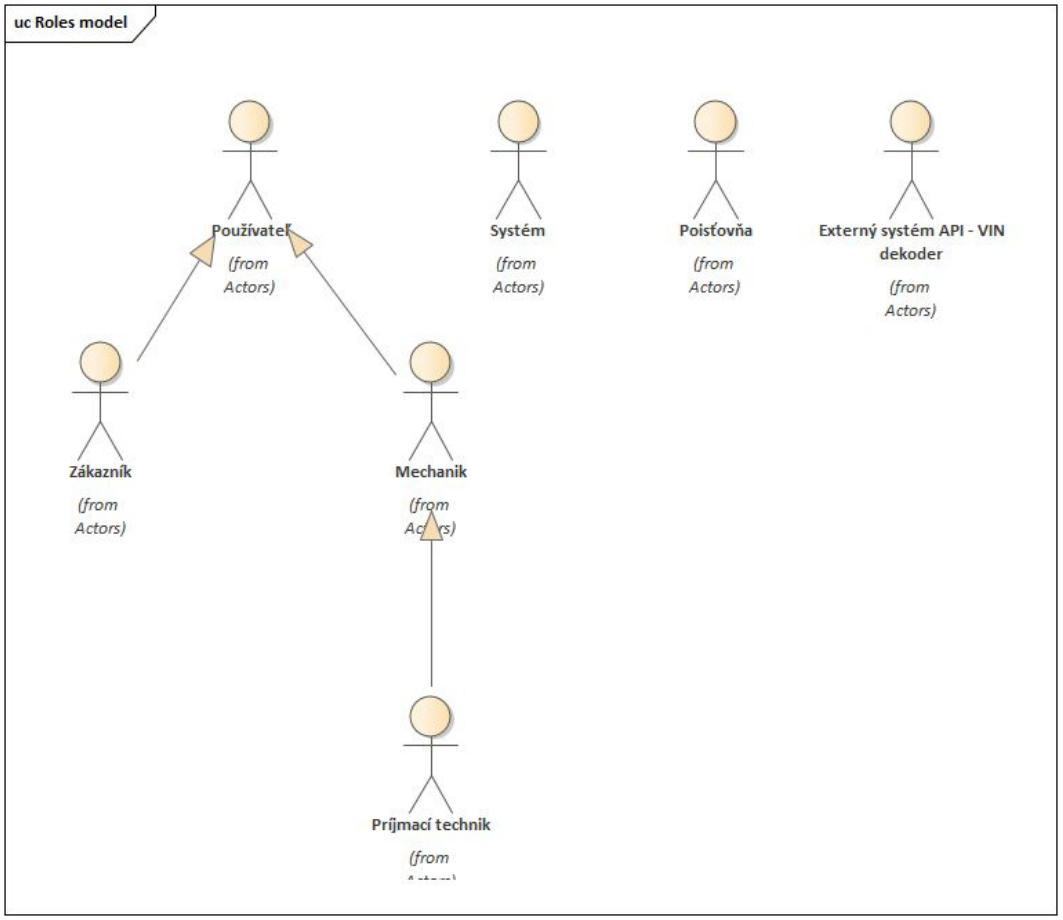
- Používateľ

  Táto rola má štandardne povolené len prehliadanie. Spúšťa proces registrácie, alebo prihlásenia.

- Systém

  Systém automaticky generuje notifikácie pre zákazníka aj servis o stavoch objednávky. Systém vytvára faktúry pre ukončené objednávky servisu.

- Poisťovňa
  Táto rola vystupuje v systéme len v prípade platby poistnej udalosti za servisovanie zákazníkovho auta.

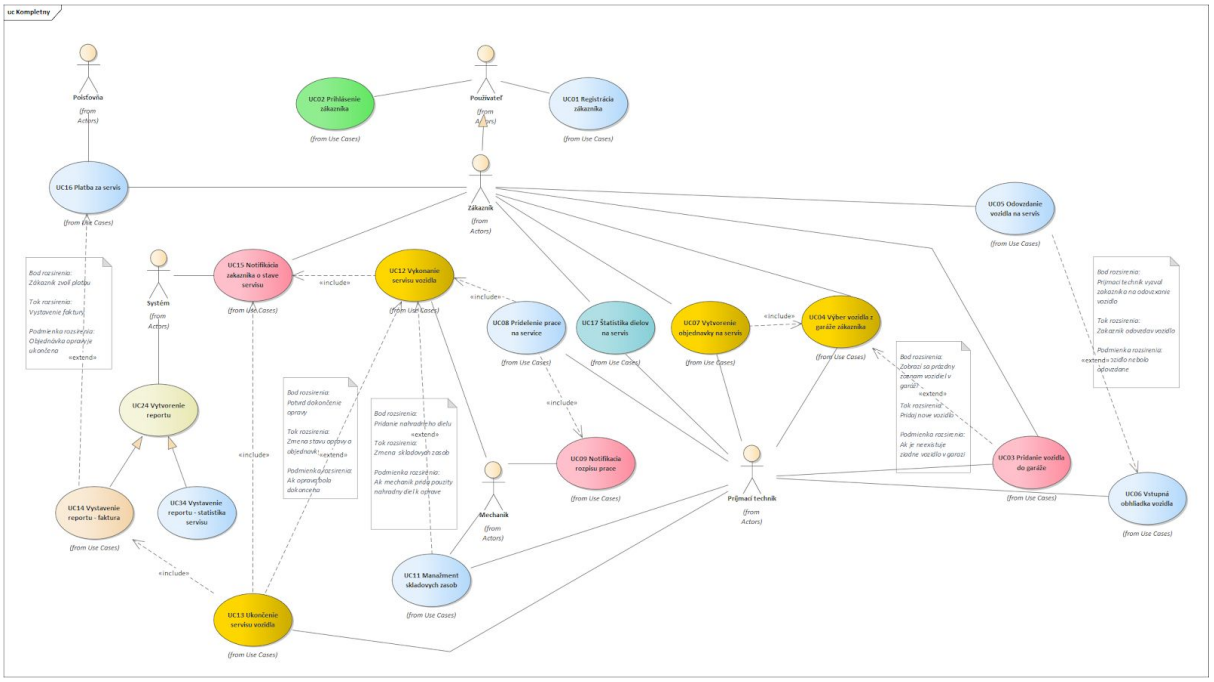


Obr. č.1 Diagram závislosti aktérov

Use cases:

V tejto sekcií dokumentu popíšeme naše use casy. Ak nie su diagramy dobre viditeľné, tak ich nájdete v priloženom .eapx súbore. Use casy boli modelované viacej biznisovo, to nám prišlo ako vhodná granularita pre riešenie daného problému.

Use case diagram:



Obr. č.2 Celkový use case diagram

Use case popis:

UC01 Registracia zakaznika

1. Používateľ klikne na registraciu
2. Používateľ vyplní osobne a kontaktné údaje
3. Používateľ potvrdí registráciu
4. System overí dostupnost a platnosť údajov
5. Používateľ si overi učet prostredníctvom emailu
6. System zaregistruje používateľa do autentifikačného serveru
7. Pripad použitia končí

Alternatívny scenár pre krok 2.

1. Používateľ je už zaregistrovaný pod daným menom
2. Systém notifikuje o danom probléme používateľa.
3. Pripad použitia začína odznovu bodom 2.

UC02 Prihlasenie zakaznika

1. Používateľ klikne na tlačidlo prihlásenie
2. Vyplní správne prihlasovacie údaje
3. Systém overí prihlasovacie údaje
4. Systém oznámi používateľovi úspešné prihlásenie
5. Prípad použitia končí

Alternatívny scenár pre krok 3.

1. Prihlasovacie údaje nie sú správne
2. Systém notifikuje používateľa o zle zadaných udajoch a požiada o retype
3. Po úspešnom vložený správnych údajov pokračuje hlavný scenár krokom 4.

UC03 Pridanie vozidla do garaze

1. Používateľ otvorí sekciu moja garáž
2. Používateľ vyplni označenie a VIN číslo vozidla
3. Používateľ potvrdí VIN vozidla
4. System získa a overi informacie o vozidle podla VIN
5. System zobrazi údaje a namapuje ich na vozidlo používateľa
6. Používateľ potvrdí pridanie auta do garaze
7. Pripad použitia končí

UC04 Výber vozidla z garáže zákazníka

1. Zvolí výber vozidla
2. Načítajú sa vozidla zakaznika
3. Zobrazí sa zoznam existujúcich vozidiel
4. Vyberie zvolené vozidlo
5. Potvrdí vyber vozidla, ktoré sa bude používať naprieč aktívnym UC
6. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

Alternativny scenár 1 pre krok 2.
V prípade, že neexistuje žiadne vozidlo

1. Zobrazí sa prázdny zoznam
2. Pokračuje rozširujúci tok na pridanie vozidla UC03
3. Po úspešnom pridaní pokračuje hlavný scenár krokom 4.

Alternativny scenár pre krok 3.
V prípade, že sa nepodarilo úspešne vybrať vozidlo

1. Zobrazí sa hláška o neúspešnom výbere vozidla, informuje používateľa, že chybou sa bude zaoberať administrátor
2. Hláška sa odošle ako chyba administrátorovi systému
3. Prípad použitia končí

UC05 Odovzdanie vozidla

1. Zákazník privezie vozidlo na servis
2. Servisný technik overí objednávku telefonickú/online/osobnú v systéme
3. Servisný technik pokračuje k vozidlu opísať stav km/paliva atď.
4. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

UC06 Vstupná obhliadka

1. Servisný technik pri vozidle kontroluje vonkajšie oderky, škrabance a preliačiny
2. Servisný technik zapíše hore spomenuté do systému a ak je zjavná závada nafotí ju
3. Servisný technik vytlačí papier o škodá na vozidle zákazníkovi
4. Zákazník ho akceptuje podpisom
5. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

Alternativny scenár pre krok 2.
Ak ide o poistnú udalosť

1. Servisný technik nafotí poškodenie, ktoré vloží do systému.
2. K fotkám a otvorenej zákazke priradí poisťovňu
3. Poisťovňa dopíše k zákazke číslo poistnej udalosti
4. UC pokračuje krokom 3.

UC07 Vytvorenie objednávky na servis

1. Prijímací technik, alebo zákazník zvolil objednanie vozidla na servis cez webové rozhranie
2. Prijímací technik, alebo zákazník vyberie dostupný dátum na servis
3. Prijímací technik vyberie zákazníka zo zoznamu existujúcich
4. Aktivuje sa UC04 Výber vozidla z garáže zákazníka
   4.1. Ak nebolo vybraté žiadne auto z garáže, opakuje sa krok 4
5. Prijímací technik, alebo zakaznik potvrdí vytvorenie objednávky.
6. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

Alternativny scenár pre krok 3.
Ak zákazník zvolil objednanie vozidla na servis

1. V objednávke sa aplikuje zákazník, ktorý realizuje UC
2. UC pokračuje krokom 4

Alternativny scenár pre krok 3.
Ak neexistuje zákazník

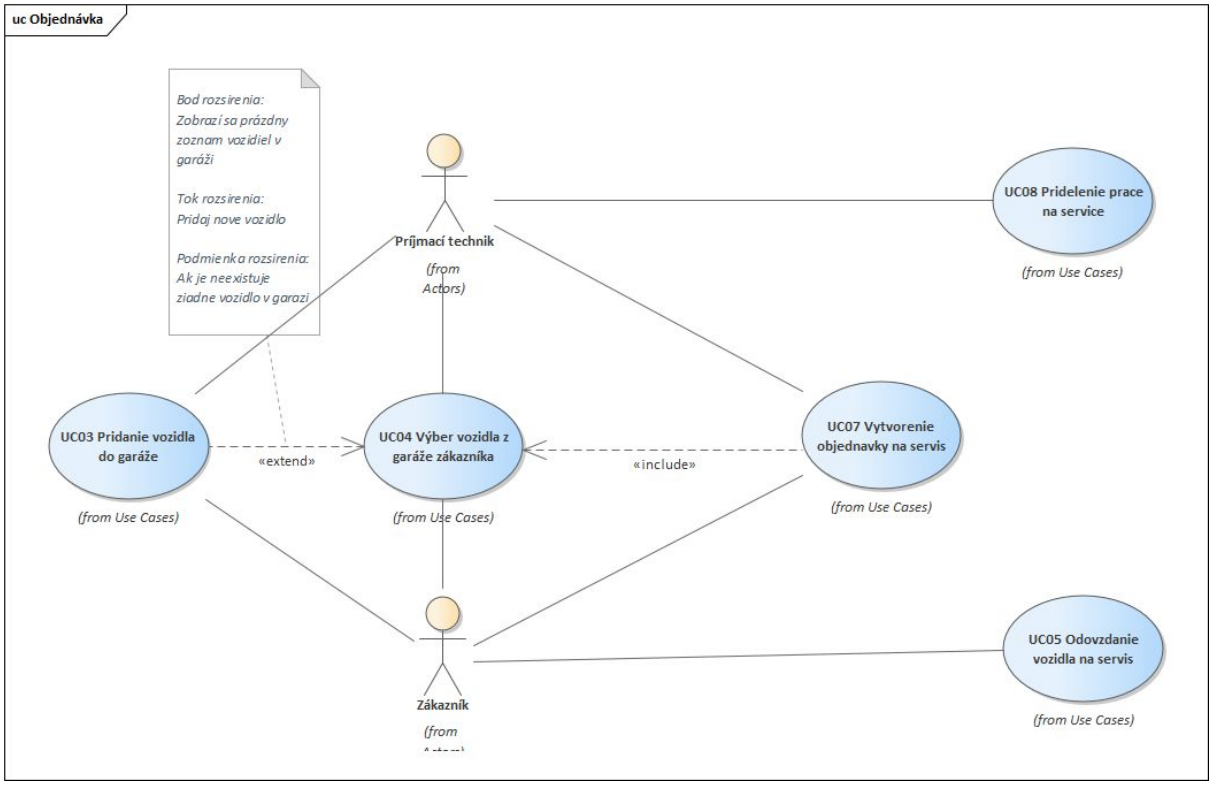
1. Prijímací technik zaregistruje zákazníka, ktorý sa bude používať naprieč aktívnym UC
2. UC pokračuje krokom 4

UC08 Akceptovanie objednávky servisu

1. Servisnému technikovi príde notifikácia o objednávke na servis
2. Zákazník je notifikovaný o akceptovaní jeho terminu na servis
3. Prípad použitia zahŕňa prípad použitia “Pridelenie vozidla na servis mechanikovi”
4. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

Alternativny scenár pre krok 2.
Ak termín nie je voľný

1. Servisný technik kontaktuje zákazníka uvedeným spôsobom
2. Pokusia sa dohodnúť na vyhovujúcom termíne
3. Zákazníkovi je odoslané potvrdenie o zmene jeho navrhovaného termínu
4. Prípad použitia pokračuje krokom 5.



Obr. č.3 Use case diagram procesu objednávky

UC09 Pridelenie vozidla na servis mechanikovi

1. Servisný technik vytvori servis, ku ktorému priradí mechanika
2. Servisný technik priradí mechanikovi časový úsek na zvládnutie servisu
3. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

Alternativny scenár pre krok 2.
Ak sú známe potrebné náhradné diely

1. Servisný technik požaduje náhradné diely zo skladu
2. Ak sklad týmito dielmi nedisponuje doobjedná ich
3. Prípad použitia pokračuje krokom 2.

UC10 Ukončenie objednávky servisu

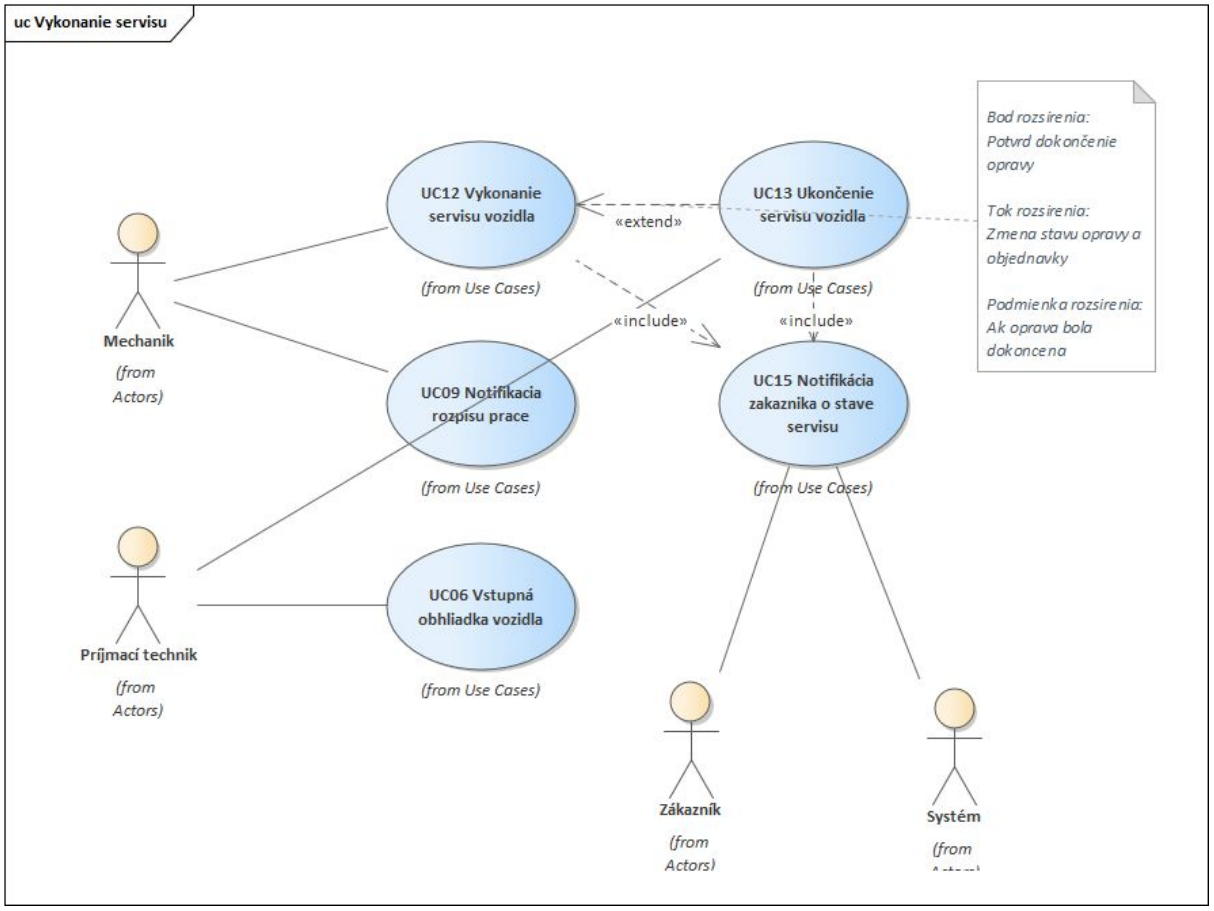
1. Servisný technik do systému zadá informácie o oprave vozidla
2. Servisný technik vykoná všetky požadované akcie zákazníkom (dotankovanie, doliatie vody do ostrekovačov, umytie vozidla atď.)
3. Vozidlo je prichystané na odovzdanie zákazníkovi
4. Zákazník príde pre vozidlo a je mu bezodkladne odovzdané

UC11 Manažment skladových zásob

1. Servisný technik priraďuje k objednávke diely použité na opravu vozidla
2. Mechanik tieto diely použije na opravu vozidla
3. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

UC12 Vykonanie servisu vozidla

1. Mechanik vykoná servis vozidla
2. Prípad použitia zahŕňa prípad použitia “Manažment skladových zásob”
3. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.



Obr. č.4 Use case diagram procesu servisu vozidla

UC13 Ukončenie servisu vozidla

1. Servisnému technikovi je na stol položený list o vykonanej oprave vozidla
2. Zákazníkovi je odoslaná notifikácia o ukončení servisu vozidla
3. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

UC14 Vystavenie faktúry za servis

1. Systém po ukončení servisu vystaví faktúru za servis
2. Servisný technik ju skontroluje a schváli.
3. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

UC15 Notifikácia o ukončení servisu

1. Systém odošle zákazníkovi notifikáciu o ukončení servisu
2. Zákazník je vyzvaný aby sa dostavil pre opravené vozidlo
3. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

UC16 Platba za servis

1. Zákazník po notifikácii, môže za svoj servis zaplatiť online cez IS autoservisu
2. Servisný technik priradí platbu k objednávke
3. Zákazník si vyzdvihne vozidlo u servisného technika
4. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

Alternativny scenár pre krok 1.
Ak ide o poistnú udalosť

1. Poistnú udalosť zaplatí poisťovňa, až potom je možné si vyzdvihnúť vozidlo
2. Prípad použitia pokračuje krokom 2

Alternativny scenár pre krok 1.
Ak ide o hotovostnú platbu

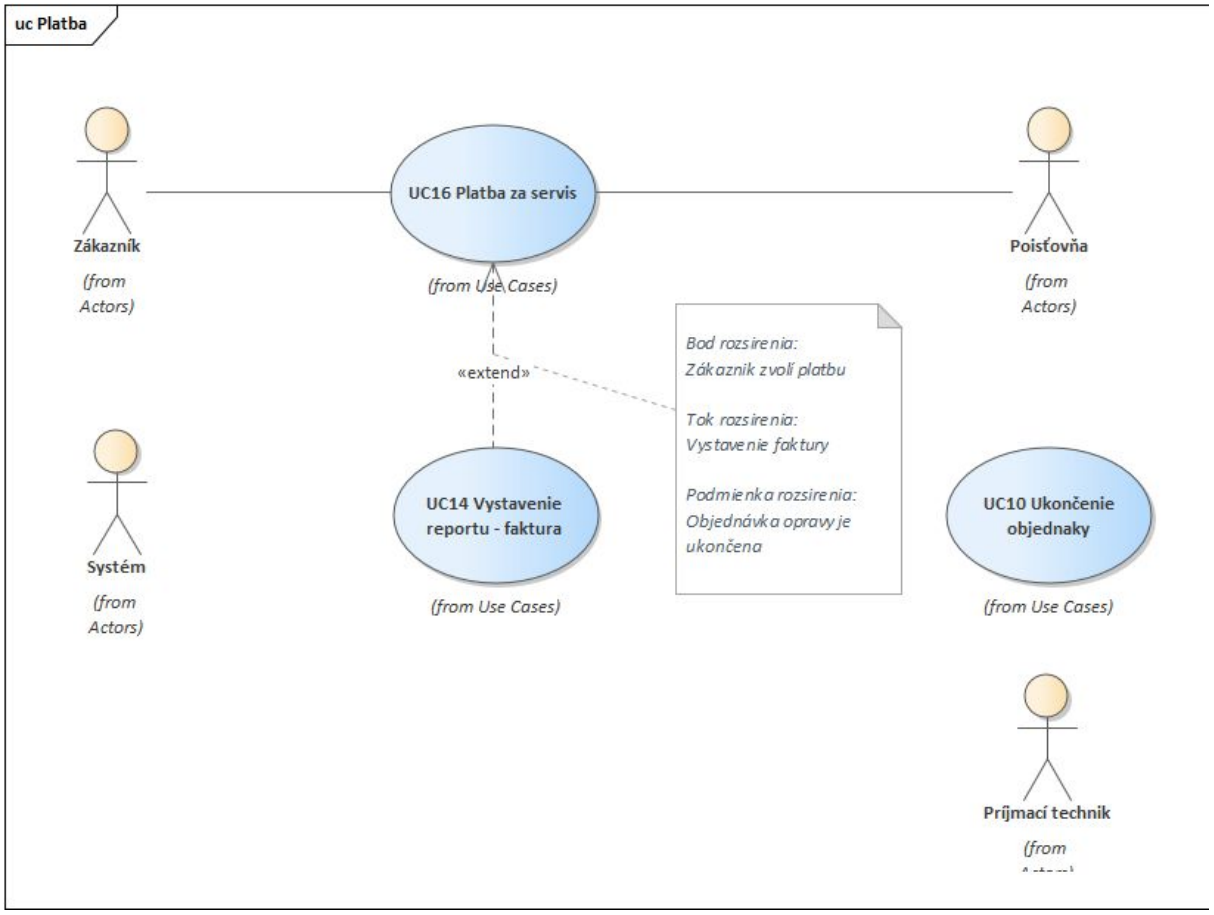
1. Zákazník zaplatí za servis priamo v servise.
2. Prípad použitia pokračuje krokom 2

Alternativny scenár pre krok 1.
Ak sa platba nepodarila (zákazník)

1. Fakturovaný zákazník je požiadaný notifikáciou o platbu v servise
2. Prípad použitia pokračuje krokom 2

Alternativny scenár pre krok 1.
Ak sa platba nepodarila (poisťovňa)

1. Prijímací technik diskutuje s poisťovňou o platbe
2. Pokial platba nedorazila, zákazník si nemôže prebrať vozidlo
3. Prípad použitia pokračuje krokom 2



Obr. č.5 Use case diagram procesu platby

UC17 Štatistika oprav vozidla

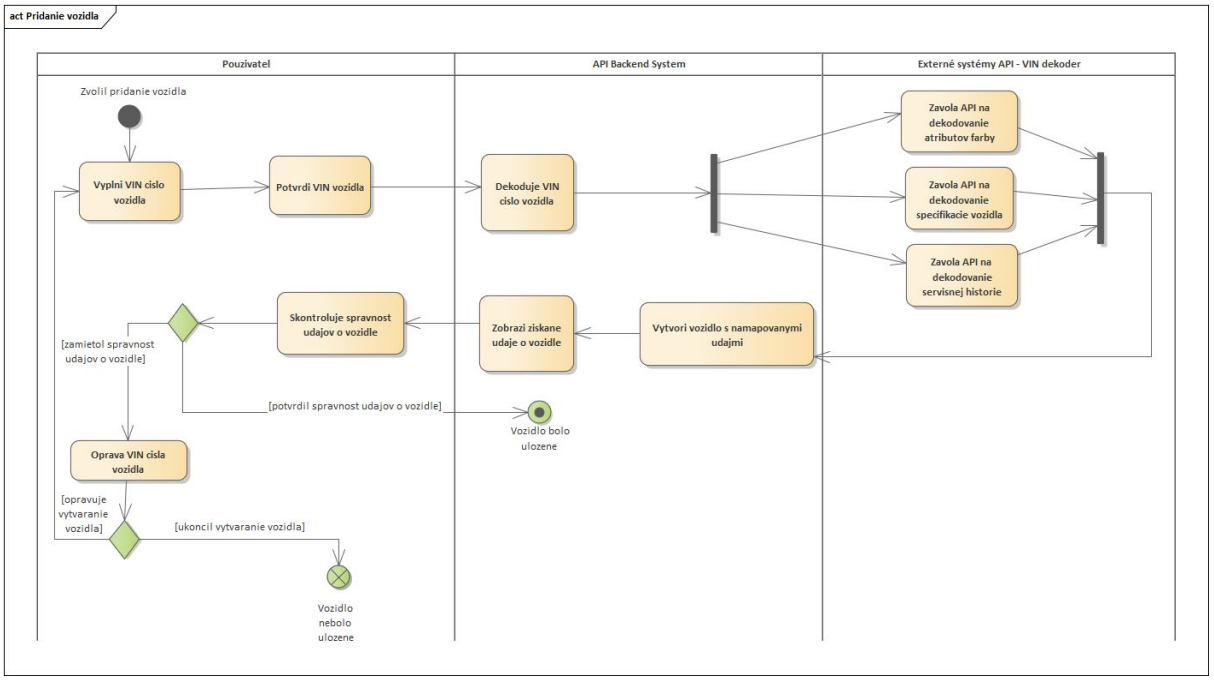
1. Zákazník si otvorí vozidlo vo svojej virtuálnej garáži
2. Zobrazí sa mu štatistika s vizualizáciou opravovaných komponentov na časovej osy
3. Prípad použitia končí sa môže skončiť v ktoromkoľvek kroku hlavného alebo alternatívneho scenáru. S následkom ukončenia všetkých aktivačných UC.

Activity diagrams:

V tejto sekcii si ukážeme aktivity diagramy. Namodelovali sme tri, podľa nás najviac popisujúce systém.

AD01 Pridanie vozidlo do garáže

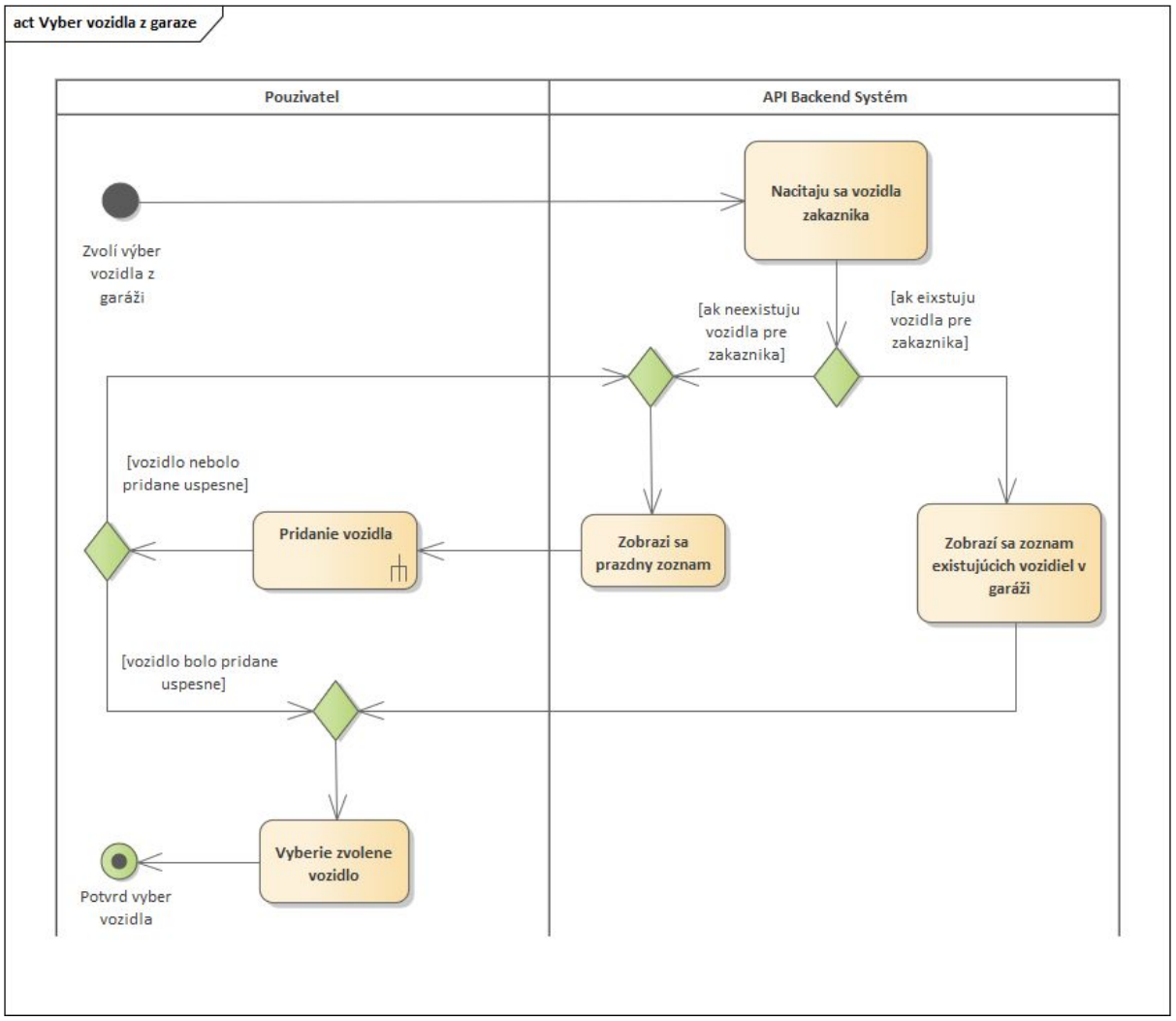
Používateľ v diagrame znamená, že to môže vykonať ktokoľvek (zákazník, servisný technik, mechanik).



Obr. č.6 Aktivity diagram pridania vozidla do garáže

AD02 Výber vozidla z garáže

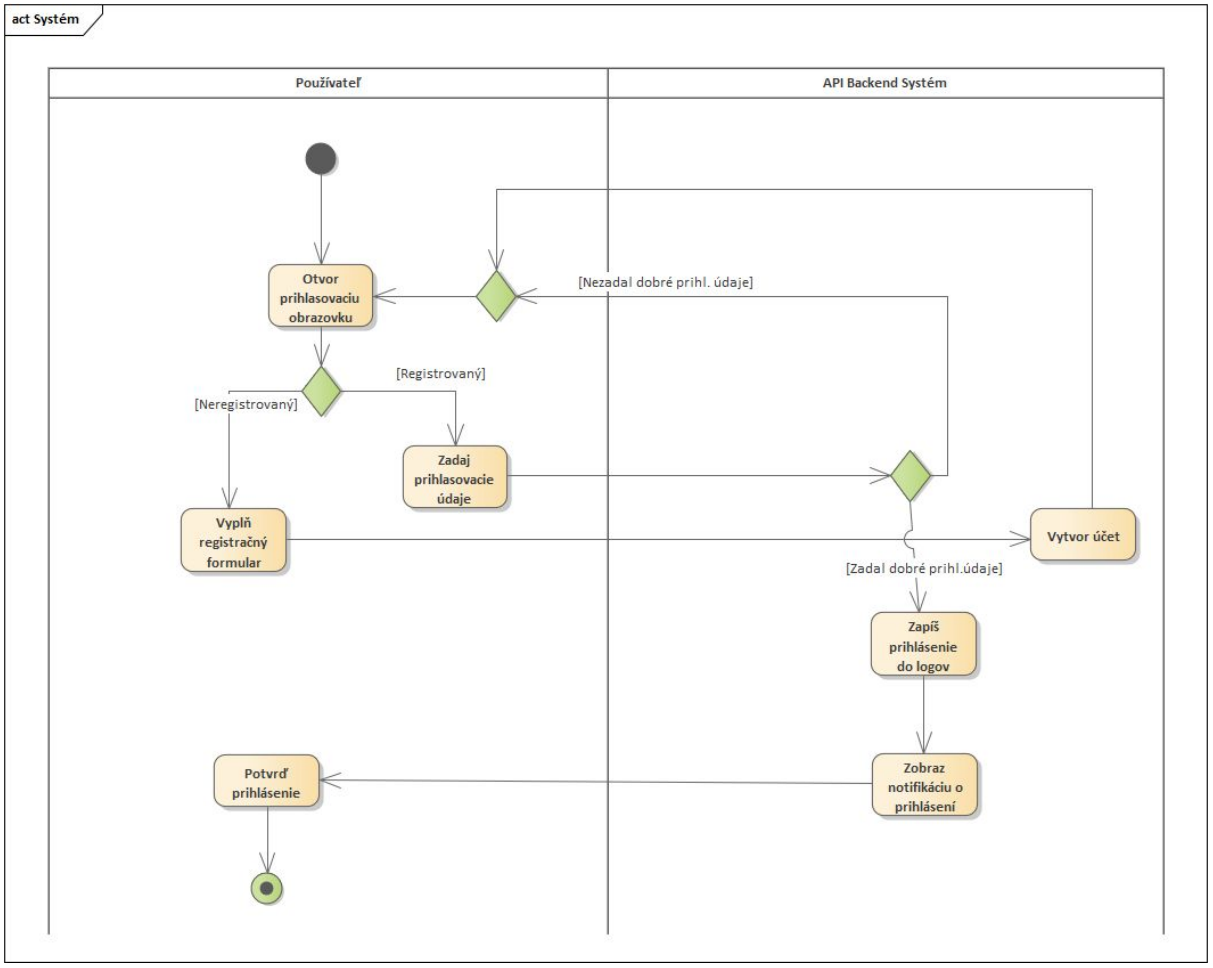
Aktivita pridanie vozidla je sub aktivita, ktorá je popísaná diagramom vyššie.



Obr. č.7 Aktivity diagram výberu vozidla z garáže

AD03 Prihlásenie používateľa

V tomto diagrame sa na používateľa pozeráme výhradne ako na zákazníka.



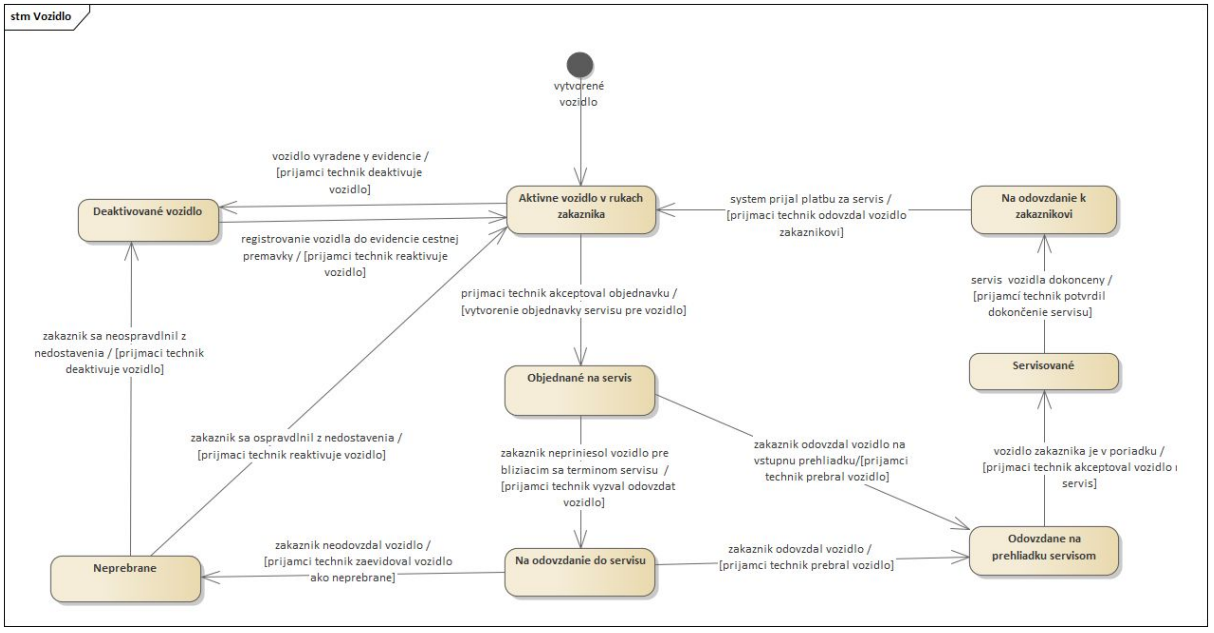
Obr. č.8 Aktivity diagram prihlásenia/registrácie používateľa

State diagrams:

V tejto sekcii si ukážeme stavové diagramy. Namodelovali sme dva, podľa nás najviac popisujúce systém.

SD01 Vozidlo

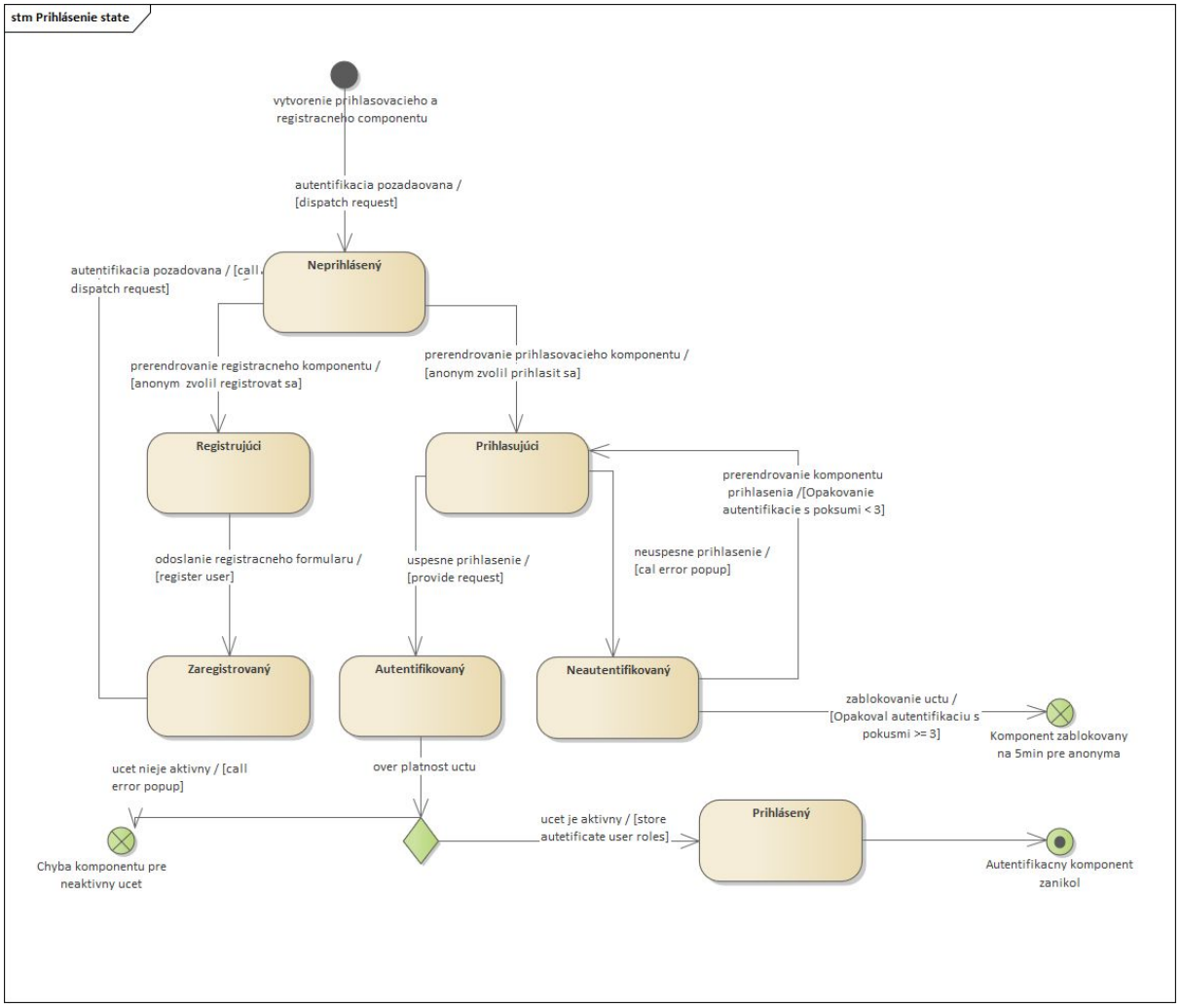
Do prvého stavu sa dostávame okamžite po úspešnom vytvorení (pridaní do garáže) vozidla.



Obr. č.9 Stavový diagram entity vozidla

SD02 Prihlásenie používateľa

Prihlasovanie používateľa, ktorý ak nemá účet je požiadaný o registráciu.



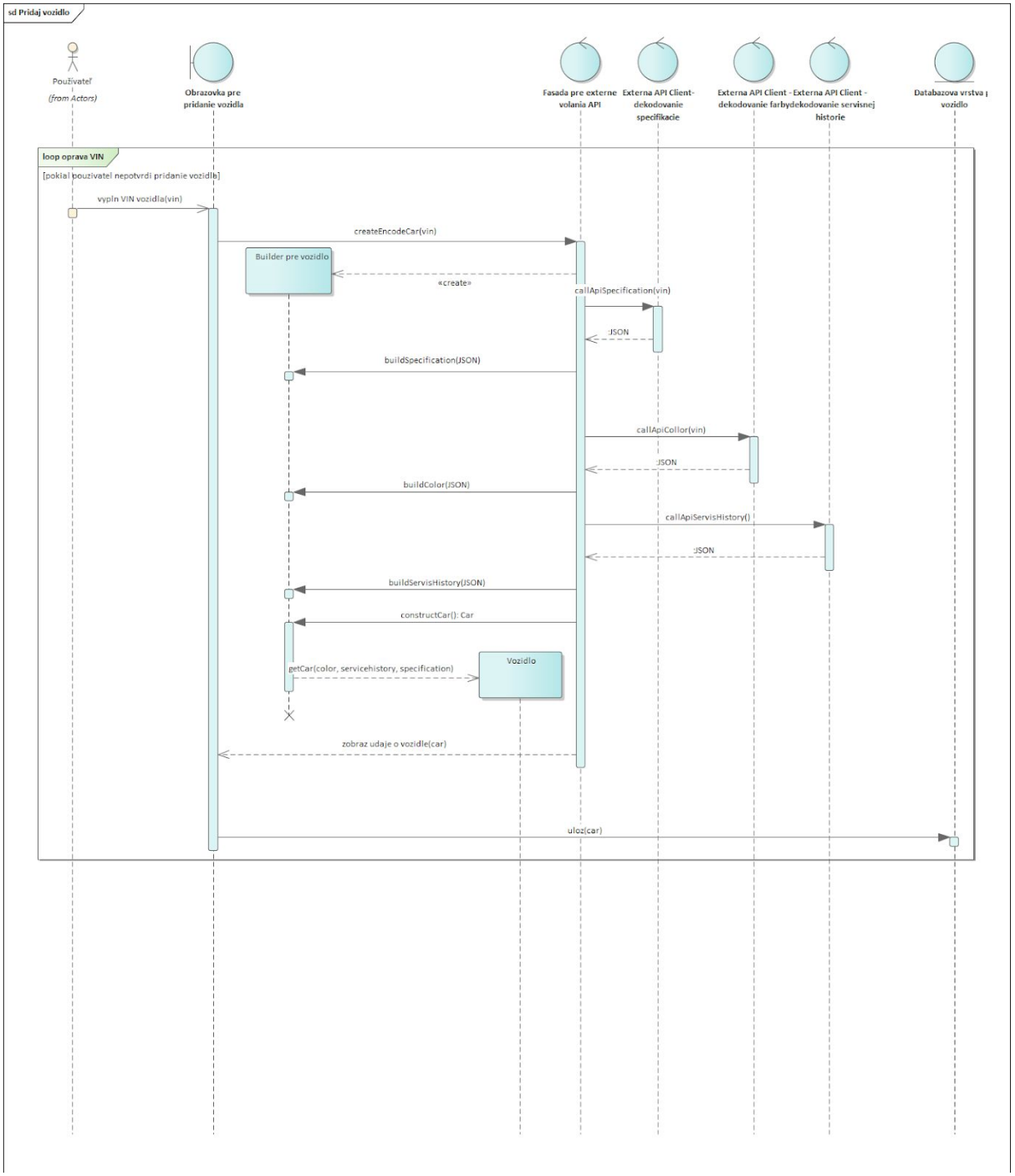
Obr. č.10 Stavový diagram entity používateľa

Sequence diagrams:

V tejto sekcii si ukážeme sekvenčné diagramy. Namodelovali sme tri, podľa nás najviac popisujúce systém.

SqD01 Pridanie vozidla do garáže

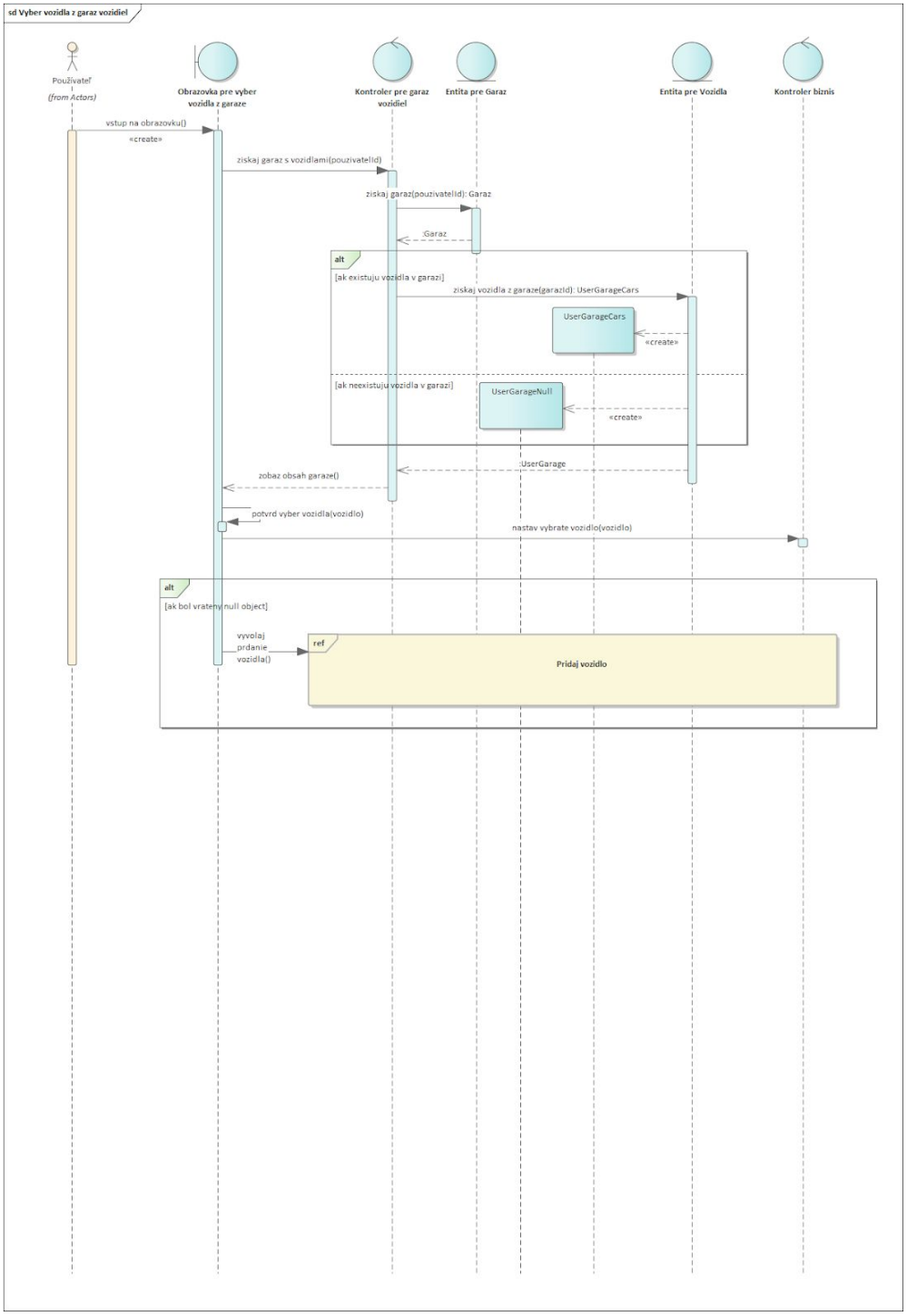
V tejto sekvencií volaní je jasne viditeľné spájanie dát s použitím návrhového vzoru facade pattern. Tieto dáta následne ešte spojíme builderom. Návrtom bude doménový objekt.



Obr. č.11 Sekvenčný diagram procesu pridaj vozidlo

SqD02 Výber vozidla z garáže

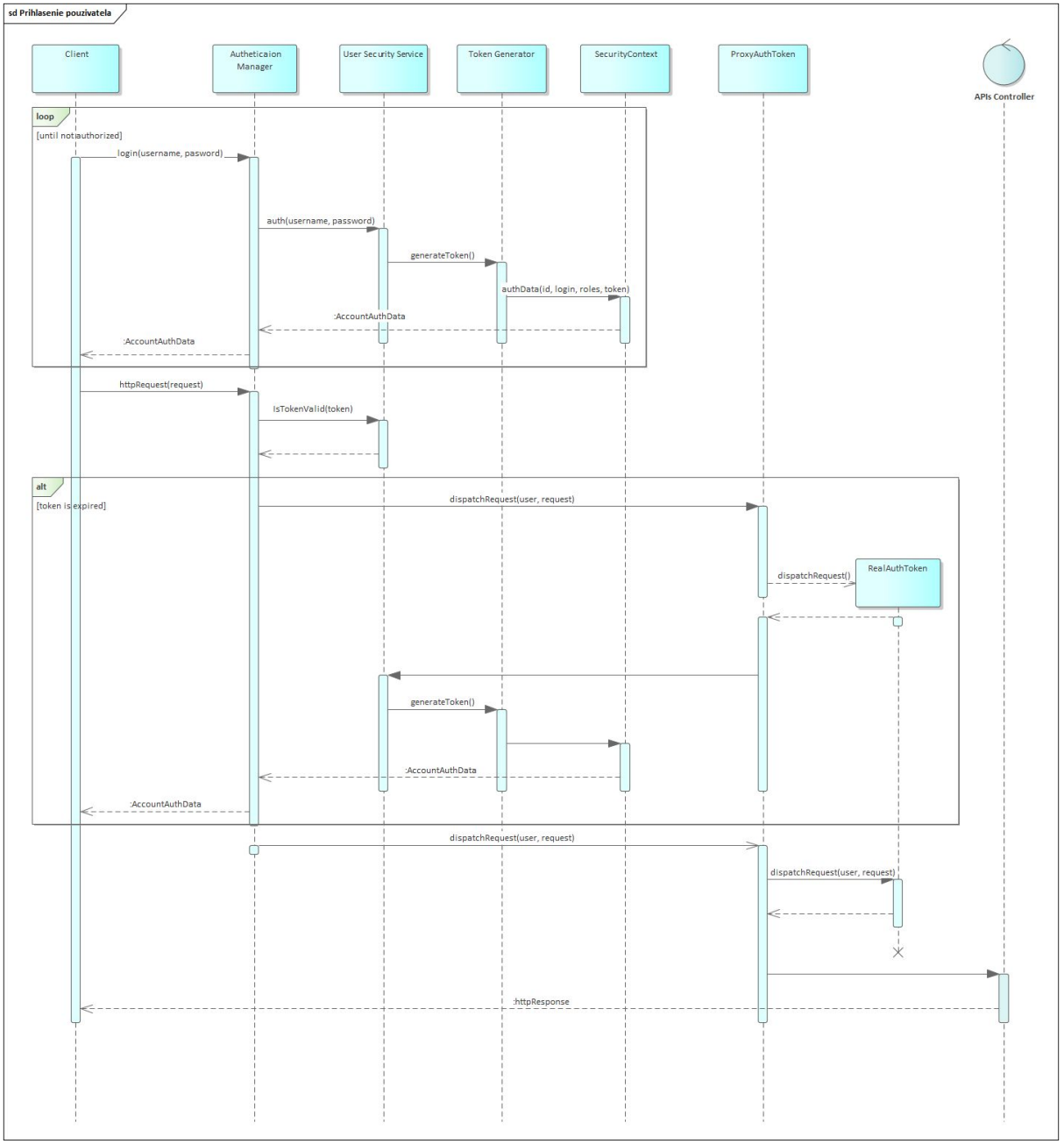
Pri výbere vozidla z garáže sme použili návrhový vzor null object. Návratom z volanej metódy je rozhranie s typom *UserGarage.* Toto rozhranie sme implementovali dvoma triedami. V prípade, ze neexistujú žiadne vozidlá, tak vrátime NullObject implementáciu *UserGarage.* Ak existujú vozidlá vrátime reálnu implementáciu s naplneným polom vozidiel.



Obr. č.12 Sekvenčný diagram procesu vyber vozidlo z garáže

SqD03 Prihlásenie používateľa (záložný)

HTTP interceptor predstavuje vzor obnovovania access tokenu. Sekvencia volaní sa mení len v prípade, ak vyprší platnost tokenu. V tom prípade sa vytvorí *RealAuthToken* pomocou, ktorého už dispatchujeme všetky ďalšie requesty.

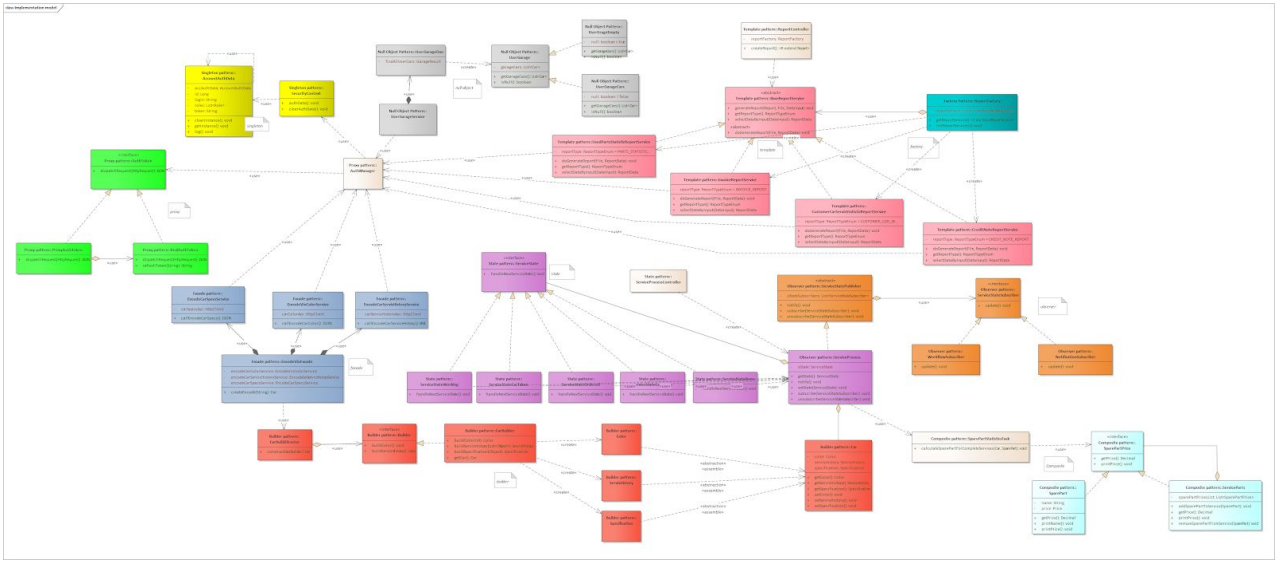


Obr. č.13 Sekvenčný diagram prihlásenia používateľa

Class diagram

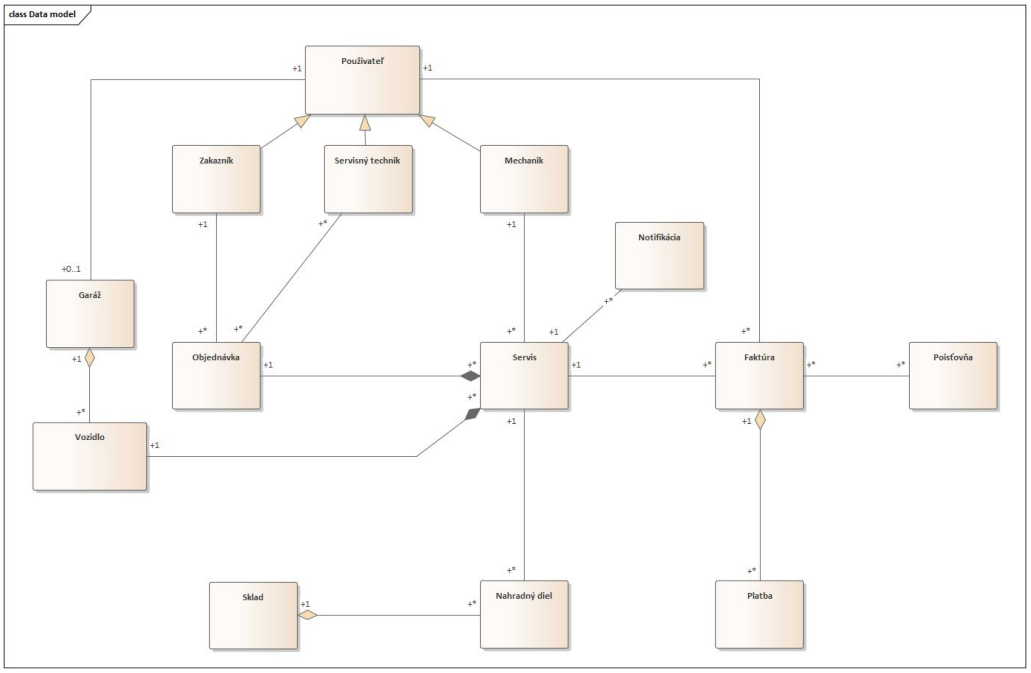
Obrázky bohužiaľ, nie sú v dokumente moc čitateľné, sú však prílohou odovzdaného zip súboru. Celkový implementačný class diagram ukazuje a popisuje prepojenie návrhových vzorov. Tieto vzory budú popísané v ďalších častiach.

Celkový



Obr. č.14 Celkový class diagram

Sketch dátový model



Obr. č.15 Navrhovaný dátový model

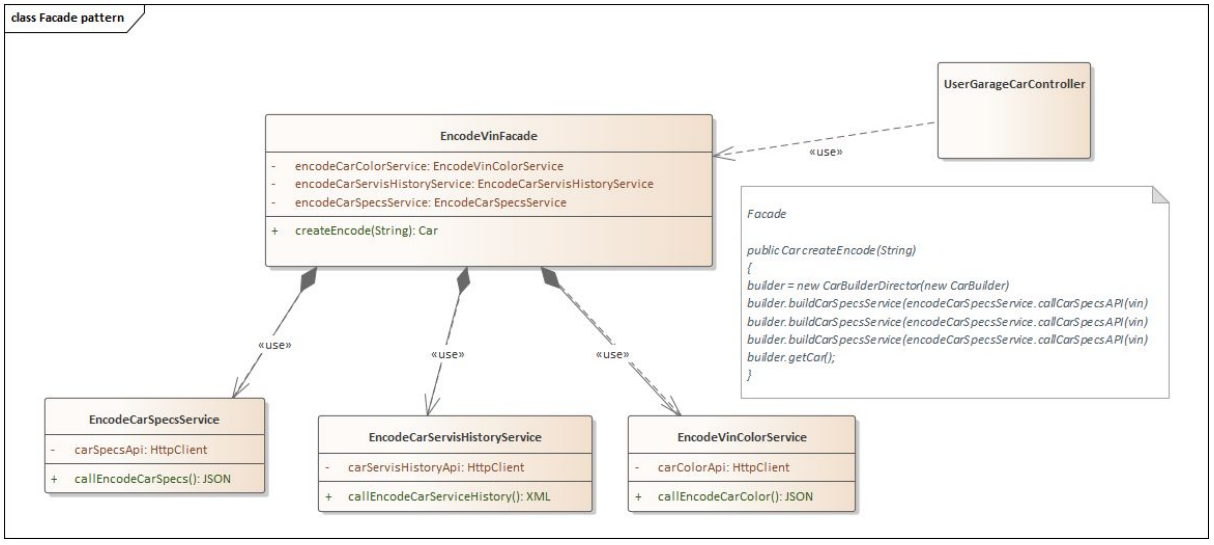
Vzory

V tejto sekcii implementujeme návrhové vzory. V sekcií použijeme formu opis vzoru, diagram a kód. Kód vzoru ilustruje vždy najsamopopisnejšiu triedu zo vzoru, ak by ste chceli vidieť viac k dokumentu je priložený spustiteľný kód. Main sa náchádza v súbore OoansAutoservisApplication.java.

Facade pattern

Vzor facade nám pomáha zjednodušiť, sprehľadniť a izolovať časť kódu, kde príchádza k dekódovaniu VIN čísla z troch rôznych API web servicov. Facade volá používateľ prostredníctvom UserGarageControllera, ked pridá vozidlo do svojej garáže prostredníctvom UI.

Diagram



Obr. č.16 Class diagram Facade

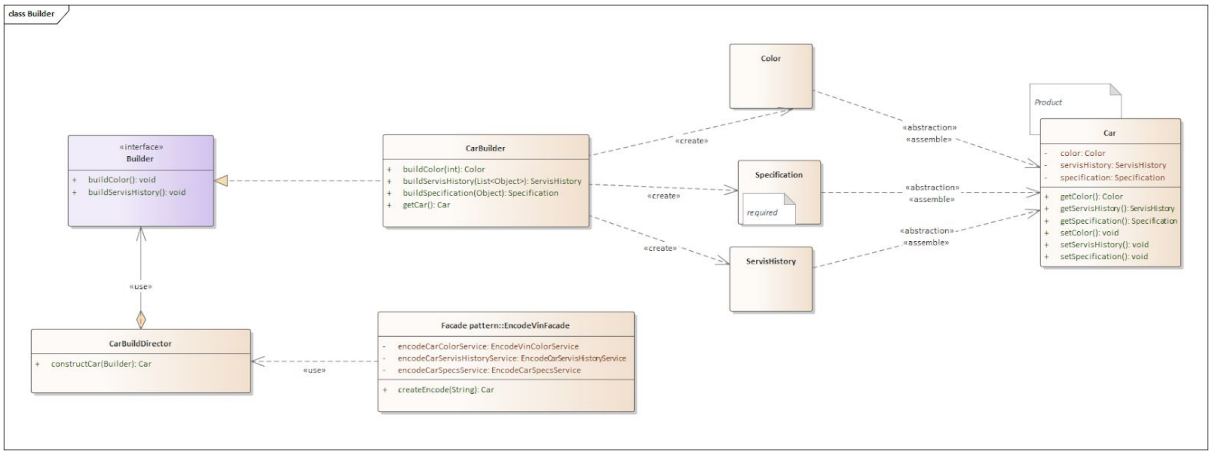
Kód

```
public EncodeVinFacade() throws JsonProcessingException {
    this.encodeVinColorService = new EncodeVinColorService();
    this.encodeCarServiceHistoryService = new EncodeCarServiceHistoryService();
    this.encodeCarSpecsService = new EncodeCarSpecsService();

}
```

Builder pattern

Vzor Builder nám napomáha k skonštruovaniu objektov vozidiel. Každé vozidlo sa u nás skladá z servisnej historie, farby a technických špecifikácii (výstupy z ext. API servicov). V systéme sa taktiež môže vyskytnúť vozidlo, ktoré nemá servisnú historiu alebo farbu (resp. nebola zistená prostredníctvom API). Builder nám teda rieši problém s mohutným konštruktorom. Builder sa volá automatický z Facade triedy po obdržaní odpovedí z APín.

Diagram



Obr. č.17 Class diagram Builder

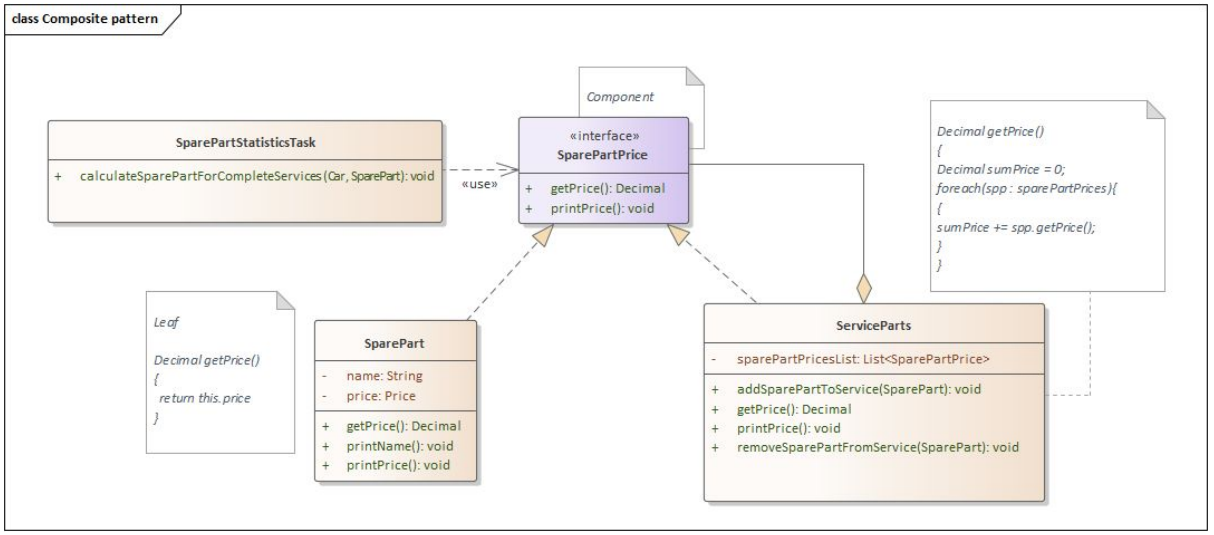
Kód

```
public class CarBuilder implements Builder {
    ObjectMapper om = new ObjectMapper();
    private Color c;  //optional
    private ServiceHistory sh; //optional
    private final Specification spec; //required

    public CarBuilder(String specJSON) throws JsonProcessingException {
        this.spec=this.buildSpecification(specJSON);
    }


    @Override
    public void buildColor(String JSON) throws JsonProcessingException {
        this.c = om.readValue(JSON, Color.class);
    }

    @Override
    public void buildServiceHistory(String JSON) throws JsonProcessingException {
        this.sh = om.readValue(JSON,ServiceHistory.class);
    }


    private Specification buildSpecification(String JSON) throws JsonProcessingException {
        return om.readValue(JSON,Specification.class);
    }

    public Car getCar() { return new Car(c,sh,spec); }
}
```

Composite pattern

Vzor composite nám napomáha udržiavať si hierarchiu nahr. dielov vymenených v automobile. Composite “strom” si držíme pre každý automobil v systéme. Následne z tejto hierarchie vieme povedať používateľovi kedy, za koľko a pri koľko km boli menené napr. brzdy ak by sme nemali vzor composite musel by otvárať faktúry z minulých servisov a hladať manuálne.

Diagram



Obr. č.18 Class diagram Composite

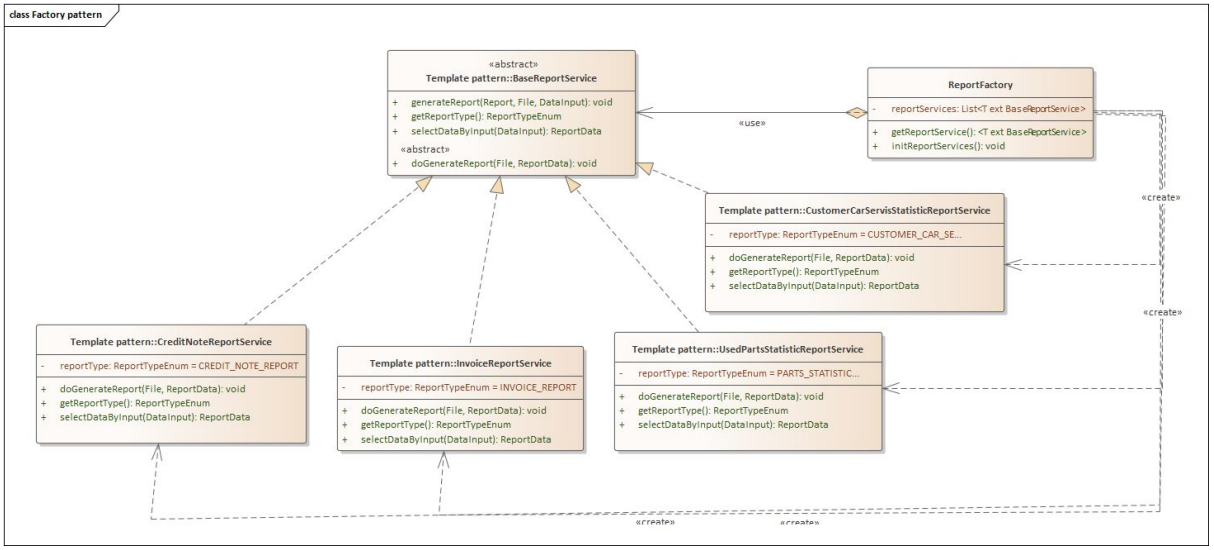
Kód

```
public class ServiceParts implements SparePartPrice {
    private List<SparePartPrice> partsList = new ArrayList<~>();

    @Override
    public void printPrice() {
        for(SparePartPrice sp:partsList)
        {
            sp.printPrice();
        }
    }
    public void add2Service(SparePartPrice sp) { partsList.add(sp); }

    public void removeFromService(SparePartPrice sp) { partsList.remove(sp); }
}
```

Factory method pattern

Tento návrhový vzor sme implementovali na vytváranie inštancií. Inštancie tried, ktoré ním vytvárame sú rôzne templaty (z ďalšieho návrhové vzoru). Vytvorené inštancie si uchovávame v lokálnom poli factory triedy Report*Factory.*

Diagram



Obr. č.19 Class diagram Factory

Kód

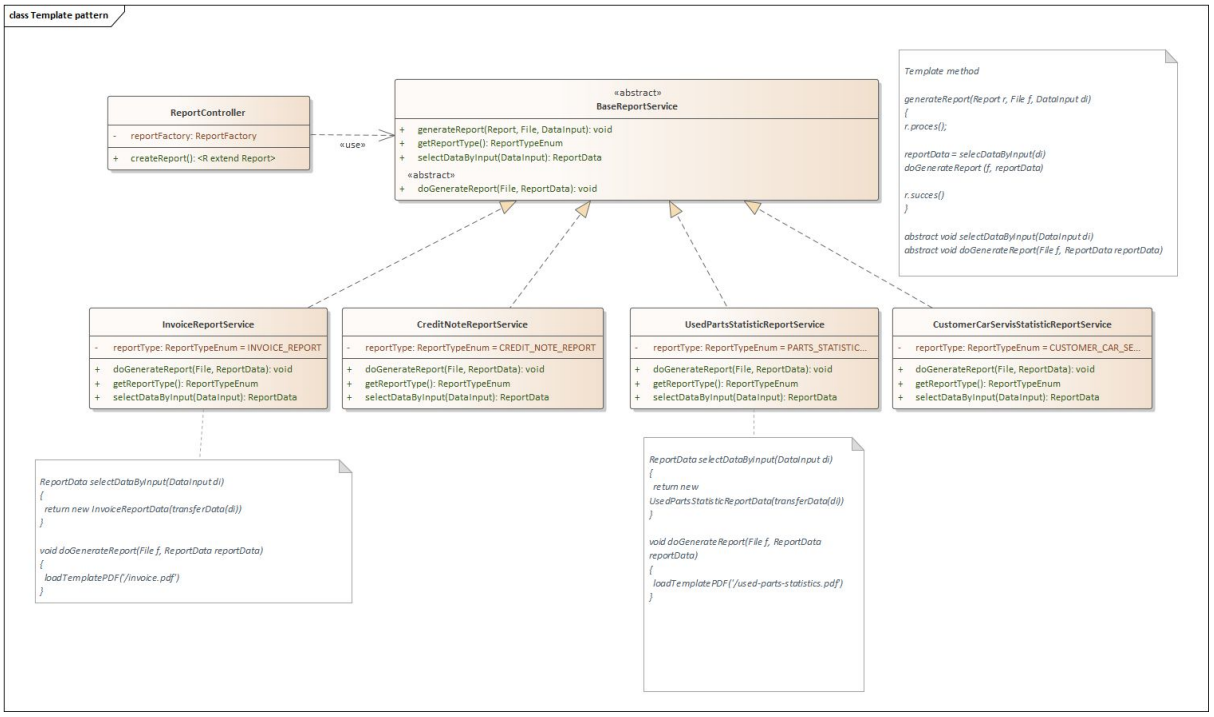
Môžeme v kóde vidieť implementáciu faktory metódy, ktorá vytvára dané inštancie. Následne vytvorene inštancie získame z poľa na základe *ReportEnumType.*

```
public class ReportFactory {

    List<ReportService> reportServices;

    public void initReportServices() {
        reportServices = new ArrayList<>();
        reportServices.add(new CreditNoteReportService());
        reportServices.add(new CustomerCarServiceStatisticsReportService());
        reportServices.add(new CustomerInvoiceServiceReportService());
        reportServices.add(new UsedPartStatisticReportService());
    }

    public ReportService getReportService(ReportTypeEnum repotTypeEnum) throws Exception {
        for (ReportService rs: reportServices) {
            if(rs.getReportType() == repotTypeEnum) {
                return rs;
            }
        }
        throw new Exception("Report service type not exist");
    }
}
```

Template method pattern

Tento návrhový vzor sme využili pri vytvárani podobných service template na generovanie PDF reportov. Generovanie PDF reportov má vždy rovnaký postup. Tento postup je implementovaný v triede *BaseReportService* v metóde *generateReport.* V tejto metóde zavoláme aj abstraktné metódy triedy. Abstraktná metóda je implementovaná rôzne v každej triede, ktora je rozšírená abstraktnou triedou *BaseReportService.* Rozdiely v jednotlivých implementácii templatov sa líšia napr. iný dátový dopyt, iný súbor pre vstupnú PDF šablónu a pod.

Diagram



Obr. č.20 Class diagram Template method

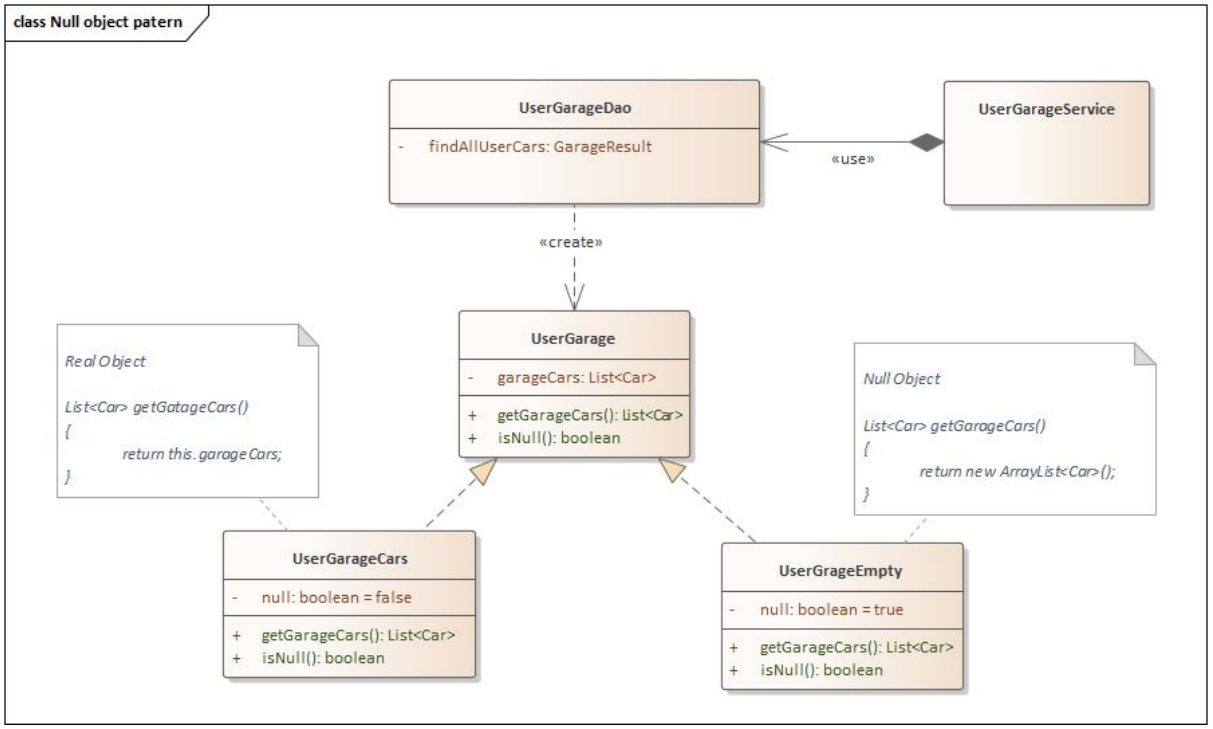
Kód

```
public abstract class BaseReportService implements  ReportService{

    private ReportDao reportDao;
    public BaseReportService() { this.reportDao = new ReportDao(); }

    public void generateReport(Report r, String fileName, String reportDataInput){
        try {
            if(AccountAuthData.getInstance().getToken() == null)    throw new Exception("Treba sa prihlasit");
            this.reportDao.process(r);
            System.out.println(r.toString());

            String reportData = selectDataByInput(reportDataInput);
            r.setPrintableData(reportData);

            doGenerateReport(fileName, reportData);

            this.reportDao.success(r);
            System.out.println(r.toString());
        }
        catch (Exception e){
            this.reportDao.fail(r);
            System.out.println(r.toString());
        }
    }

    public abstract String selectDataByInput(String reportDataInput);
    public abstract void doGenerateReport(String fileName, String reportData);
}
```

Null object pattern

Vzor null object nám napomáha pri výbere vozidla z garáže. Ak systém obdrží objekt garáže vylistuje používateľovi na UI obsah vozidiel v nej. Ak systém obdrží null object presmeruje používateľa na screen pridávanie vozidla do garáže (možné vidieť v sekvenčnom diagrame). Vzor je použitý na backende systému, a preto je volaný prostredníctvom DAO objectu, ktorý obsluhuje samotný Service.

Diagram



Obr. č.21 Class diagram Null object

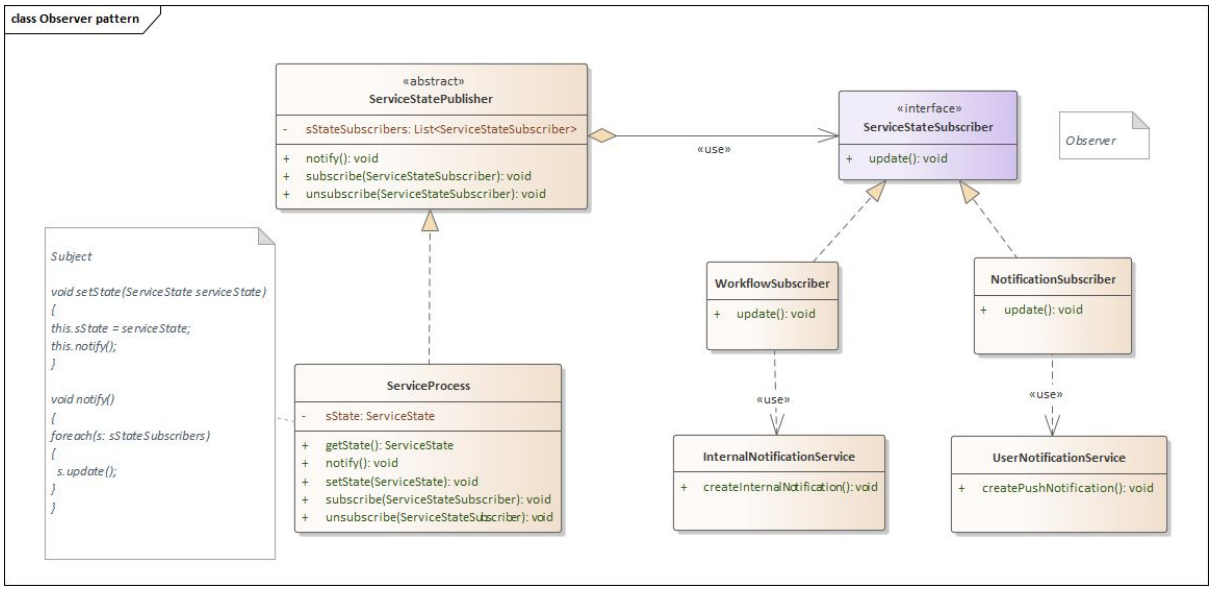
Kód

```
public abstract class UserGarage {
    public List<Car> cars;

    public List<Car> getCars() { return cars; }

    public abstract boolean isNull();
}
```

Observer

Vzor observer používame na zasielanie notifikácii zákazníkom a zmene stavu ich objednávky. Taktiež ho využívame pri internom procesy notifikácie pracovnej tabule priradenia práce. Notifikácie je vykonaná vždy pri zmene stavu servisu.

Diagram



Obr. č.22 Class diagram Observer

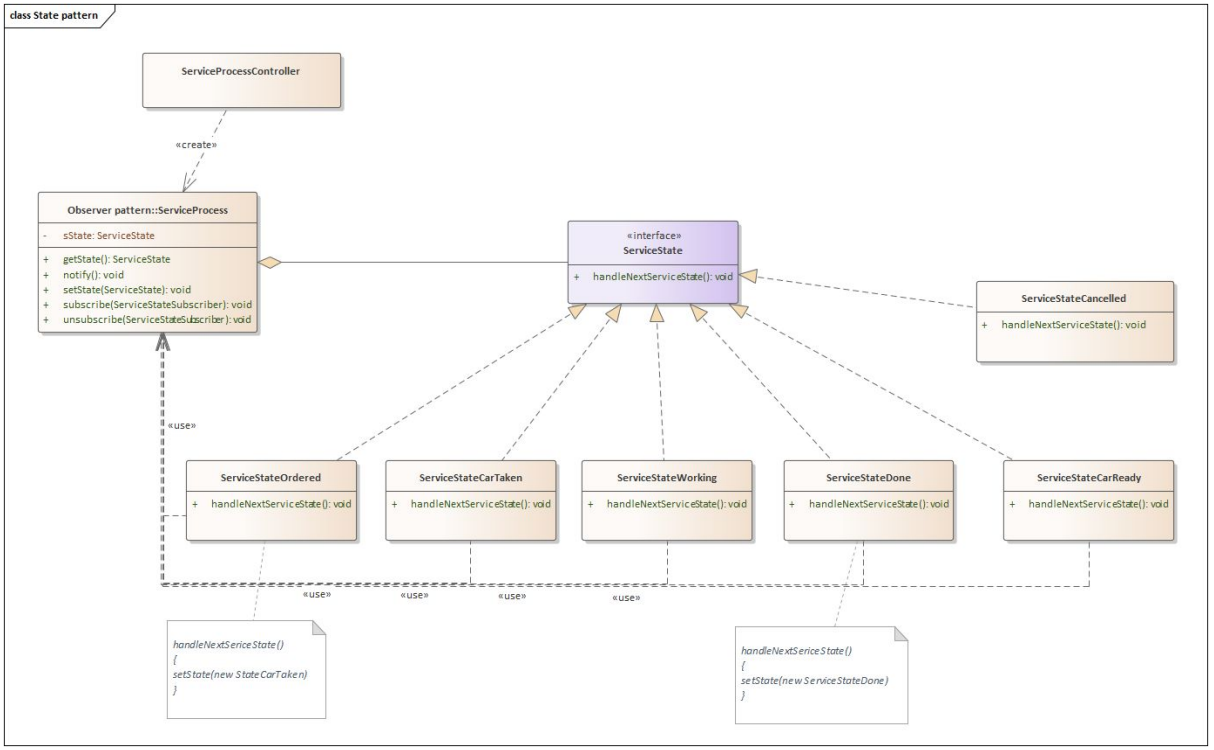
Kód

```
public abstract class ServiceStatePublisher {

    private List<ServiceStateSubcriber> serviceStateSubcribers;

    public ServiceStatePublisher() { this.serviceStateSubcribers = new ArrayList<>(); }

    public void notifyServiceStateSubscribers() {
        for (ServiceStateSubcriber sss: this.serviceStateSubcribers){
            sss.update();
        }
    }

    public void subscribe(ServiceStateSubcriber sss) { this.serviceStateSubcribers.add(sss); }
    public void unsubscribe(ServiceStateSubcriber sss) { this.serviceStateSubcribers.remove(sss); }
}
```

State pattern

Vzor state používame na udržiavanie stavu objednávky. Stav sa mení prostredníctvom inputu od používateľa ktorému je v aktuálnom stave pridelené vozidlo. Existuje teda jedna reťaz prechodu stavou objednávky. Ak sa oprava nepodarí objednávka zotrvá v stave canceled, to pre systém znamená, že nemá vystavovať žiadnu faktúru. Novou objednávkou sa jej automaticky priradí stav ordered.

Diagram



Obr. č.23 Class diagram State

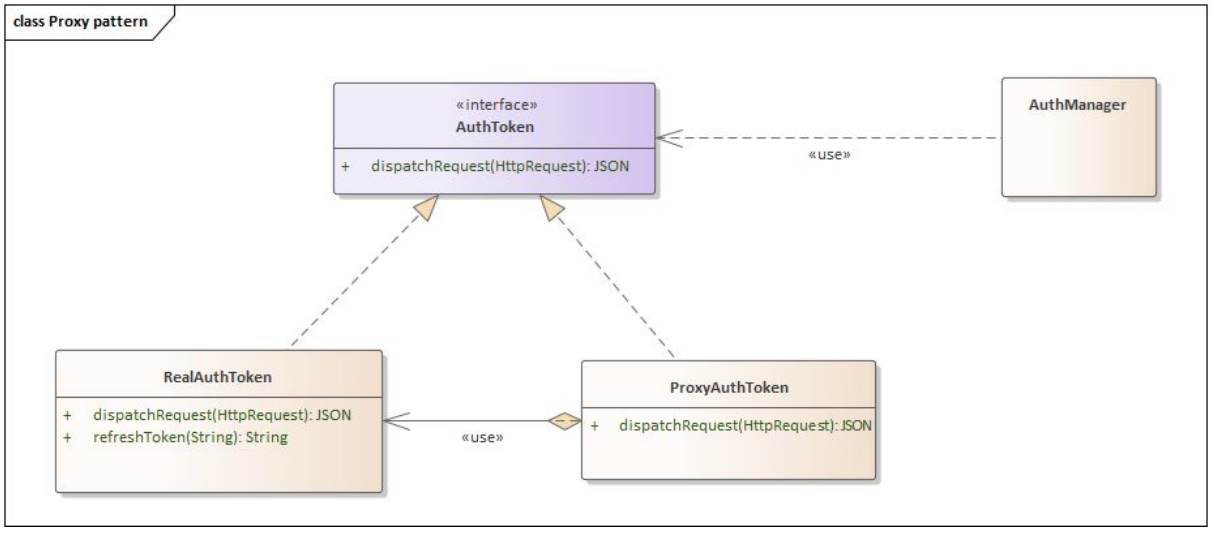
Kód

```
public class ServiceProcess extends ServiceStatePublisher {

    public ServiceState serviceState;

    public Car serviceCar;

    public Car getServiceCar() { return serviceCar; }

    public void setServiceCar(Car serviceCar) { this.serviceCar = serviceCar; }

    public void setServiceState(ServiceState serviceState) {
        this.serviceState = serviceState;
        this.notifyServiceStateSubscribers();
    }

    public ServiceState getServiceState() { return serviceState; }


}
```

Proxy pattern

Vzor proxy používame pri autorizácii a autentifikácii používateľa. Prvým prihlásením je priradený používateľovi autorizačný bearer token. Tento token má istú dobu, kedy vyprší. Proxy ne implementovaný na frontende. Na Backende máme funkcionalitu, ktorá nám povie či je aktuálny token ešte v platnosti. Ak chce teda používateľ odoslať akýkoľvek request z frontendu tak pomocou proxy vzoru je mu validovaná platnosť tokenu ak je v platnosti proxy request forwardne, ak nie je obnoví token, ktorý mu prepíše aktuálny. Vzor teda volá systém pri odoslaní akéhokoľvek requestu.

Diagram



Obr. č.24 Class diagram Proxy

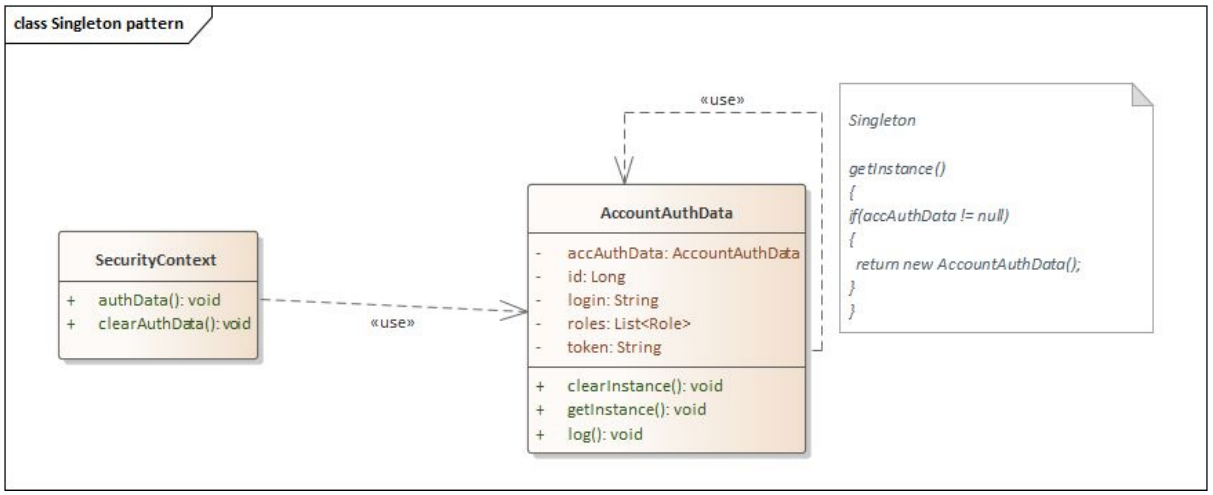
Kód

```
public class ProxyAuthToken implements AuthToken {
    private RealAuthToken realAuthToken;
    boolean first_time= true;

    @Override
    public String dispatchRequest(String httpRequest) {
        if(realAuthToken == null){
            realAuthToken = new RealAuthToken();
        }
        if(checkIfTokenIsOutDated()){
            AccountAuthData.getInstance().setToken( realAuthToken.refreshToken());
        }
        System.out.println("Request zavolany s tokenom: " + AccountAuthData.getInstance().getToken());
        return realAuthToken.dispatchRequest(httpRequest);
    }
    public boolean checkIfTokenIsOutDated(){
        if (first_time){
            first_time=false;
            return false;
        }

        Random random = new Random();
        return random.nextBoolean();
    }

}
```

Singleton

Vzor singleton používame na uchovanie používateľských informácií. Uchovávame napr. id používateľ, autorizačný token atď. Pre každého používateľa.

Diagram



Obr. č.25 Class diagram Singleton

Kód

```
public class AccountAuthData {
    Number id;
    String login;
    List<String> roles;
    String token;



    public static AccountAuthData getInstance() {
        if (user_instance == null)
            user_instance = new AccountAuthData();

        return user_instance;
    }

    public static void  clearInstance() { user_instance=null; }
    @Override
    public String toString() {
        return "AccountAuthData{" +
                "id=" + id +
                ", login='" + login + '\'' +
                ", roles=" + roles +
                ", token='" + token + '\'' +
                '}';
    }
```

Záver

Podarilo sa nám naplniť požiadavky semestrálneho projektu na predmet Objektovo orientovaná analýza a návrh softvéru. Taktiež projekt prešiel obhajobou pred cvičiacimi bez markantných pripomienok. Implementovali sme 9 plnohodnotných a 1 polovičný vzor.

Po úspešnom namodelovaní a navrhnutí je projekt, veríme, pripravený na plnohodnotnú implementáciu a následnú prevádzku.

Príloha A: Organizácia zložiek

Opis súborovej štruktúry priloženého zip súboru.

/Documentation/ - Obsahuje dokumentáciu, ktorú práve čítate.
/Implementation/ - Obsahuje implementovaný spustitelný kód v jazyku JAVA (Spring boot).
/EA/ - Obsahuje .eapx súbor z Enterprise Architectu.
/Full-size Diagrams/ - Obsahuje obrázky všetky diagramov vygenerované z EA v plne kvalite.