

„Технически университет“ - Варна

Факултет по изчислителна техника и автоматизация

Катедра: Софтуерни и интернет технологии

Курсов проект

по

ООП 2 част

ТЕМА: Коли под наем

(Вариант № IX)

Група 7

Проверил:

/ас. В.Колесниченко /

Изготвил: Анастасия Силичева

Фак. №: 19621833

Курс: 3

Група: 3

Специалност: СИТ

Варна, 2019

Варна, 2021/2022

*Задание*

Да се разработи информационна система, предоставяща възможност за отдаване на коли под наем.

Системата поддържа вида потребители: администратор и оператори с различни роли за достъп до функционалностите в системата.

***Операции за работа с потребители:***

* създаване на фирма за отдаване на коли под наем от администратор;
* създаване на оператори отдаващи коли от администратор;
* регистриране на клиенти от оператор;

***Системата поддържа операции за отдаване на коли под наем:***

* регистрация на автомобили (класове – луксозна, семейна, градска,категории – седан, сув, комби, характеристики, снимки, пушач, непушач);
* отдаване на автомобил на клиент с запазване на текущото състояние на автомобила под формата на опис протокол;
* връщане на автомобил и статус на връщането (безпроблемно, с възникнали проблеми – нов опис протокол за състоянието на автомобила);
* формиране на дължимата цена спрямо броя дни на наемане, изминати

километри и евентуални щети по автомобила(различните класи автомобили

имат различна цена за ден и за километър);

***Системата поддържа Справки по произволен период за:***

* налични автомобили;
* история на отдаване на автомобили;
* по оператори и тяхната работа;
* рейтинговане на клиенти;
* статистики на отдадените автомобили по зададени филтри – по класа, модел, тип и др.

***Системата поддържа известия за:***

* изтичащ или изтекъл период на отдаване;
* рисков клиент;

***Функционадни изисквания:***

Информационната система трябва да предоставя възможност за отдаване на коли под наем.

Съществуват 3 вида потребители: администратор, оператори и клиенти. Те трябва да изпълняват следните операции:

* създаване на фирма за отдаване на коли под наем от администратор.
* създаване на оператори отдаващи коли от администратор.
* регистриране на клиенти от оператор.

Функциите, които трябва да изпълнява изработваната система могат да се дефинират по следният начин:

* Регистрация на автомобили с различни класове – луксозна, семейна, градска; категории – седан, сув, комби; характеристики, снимки, пушач, непушач;
* Отдаване на автомобил на клиент;
* Връщане на автомобил и статус на връщането (безпроблемно, с възникнали проблеми – опис протокол);
* Формиране на дължимата цена спрямо броя дни на наемане, изминати километри и щети по автомобила;

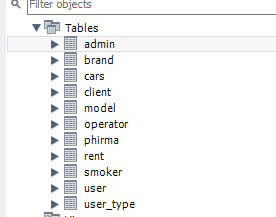
Освен посочените операции, системата трябва да поддържа справки по произволен период за:

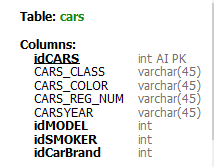
1. налични автомобили;
2. история на отдадените автомобили;
3. рейтинговане на клиенти;
4. статистики на отдадените автомобили по зададени филтри – класа, модел, тип;

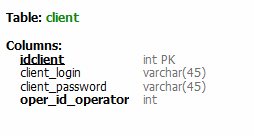
***Структура на проекта:***

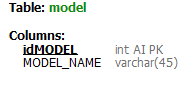


***База данни:***

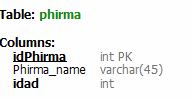


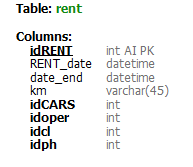


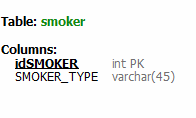


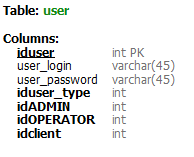


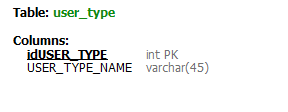


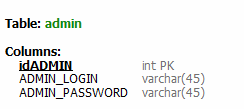


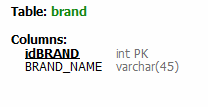


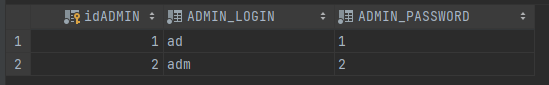


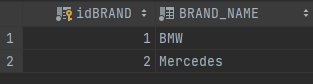


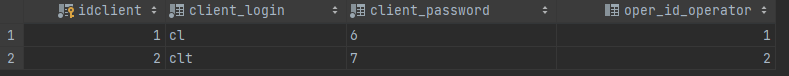


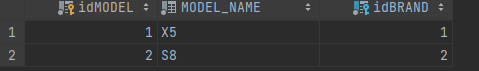


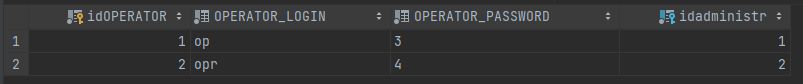


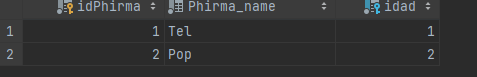


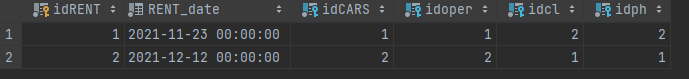


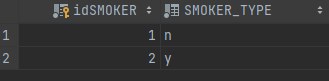


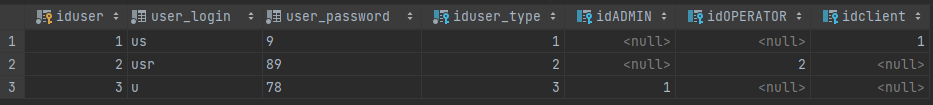


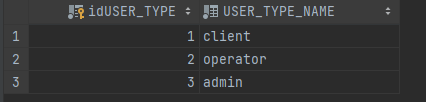


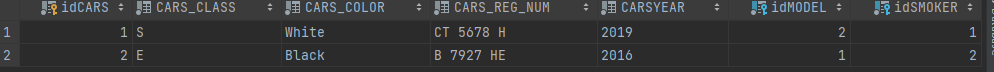




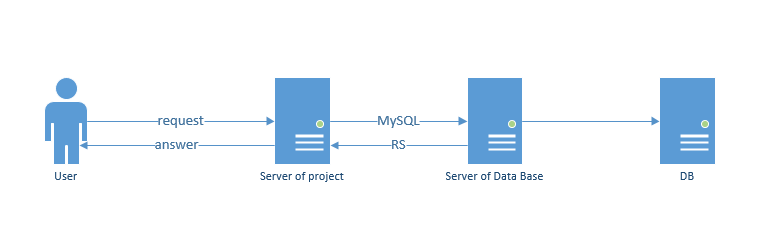




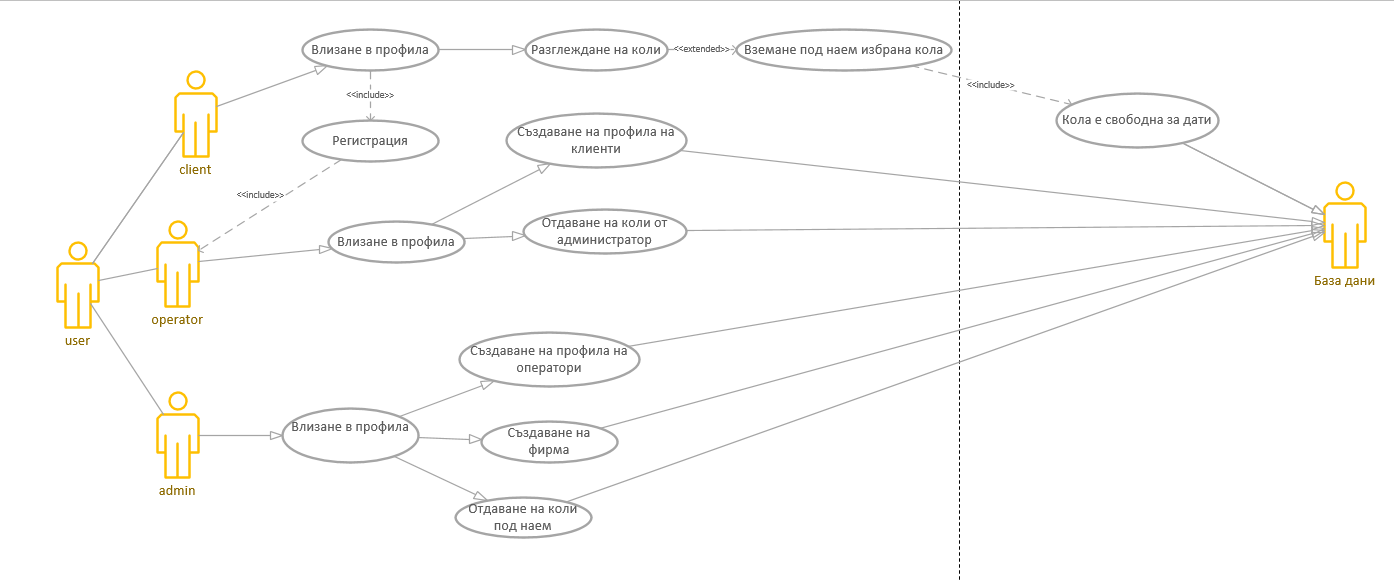




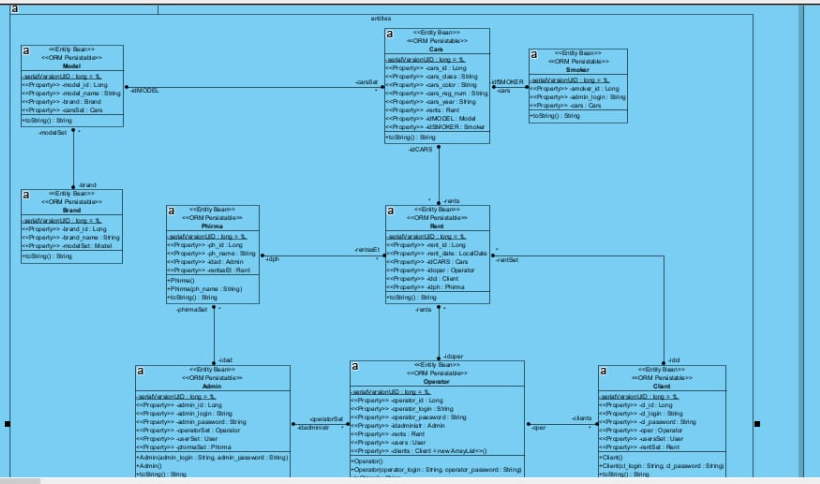
*Бизнес логика:*

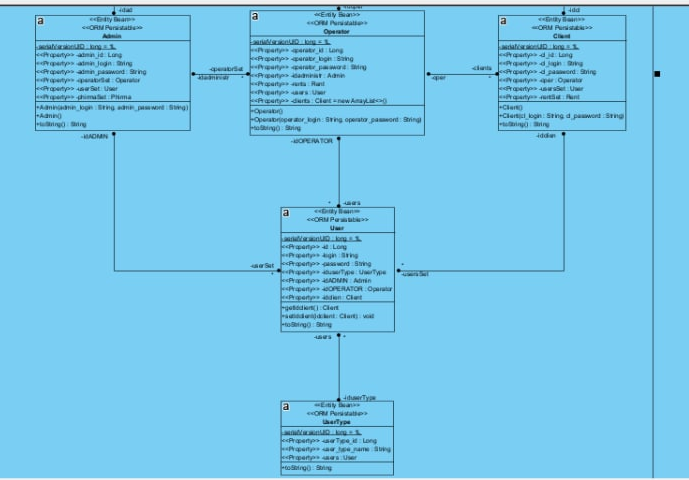


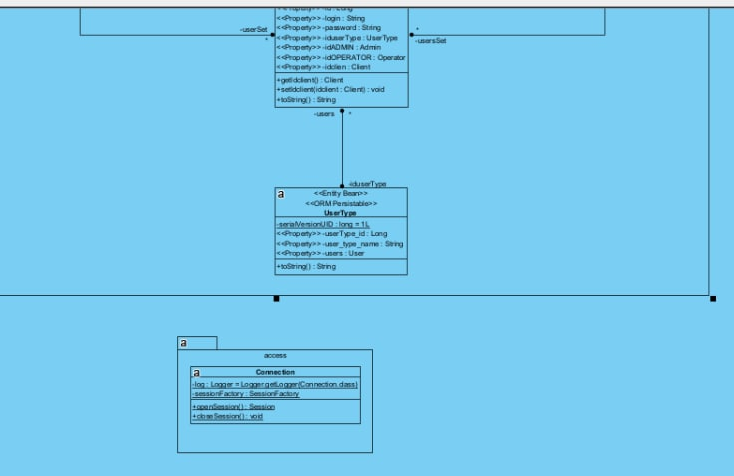
*Use Case:*

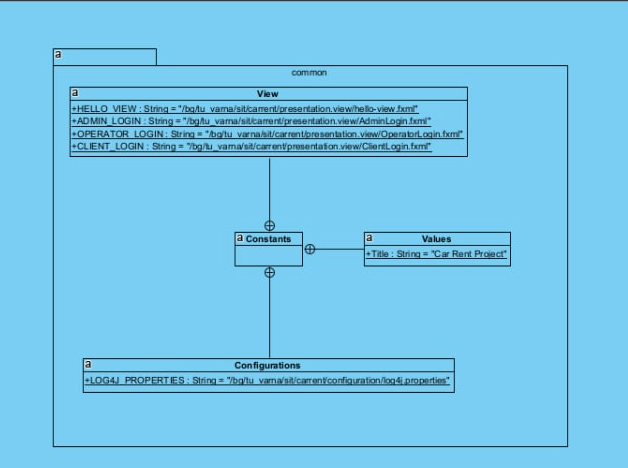


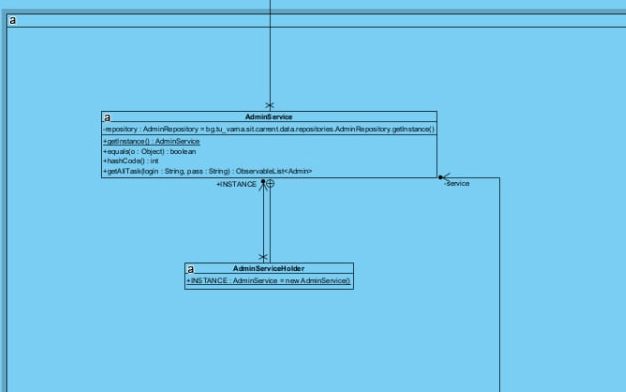
*Class Diagram:*

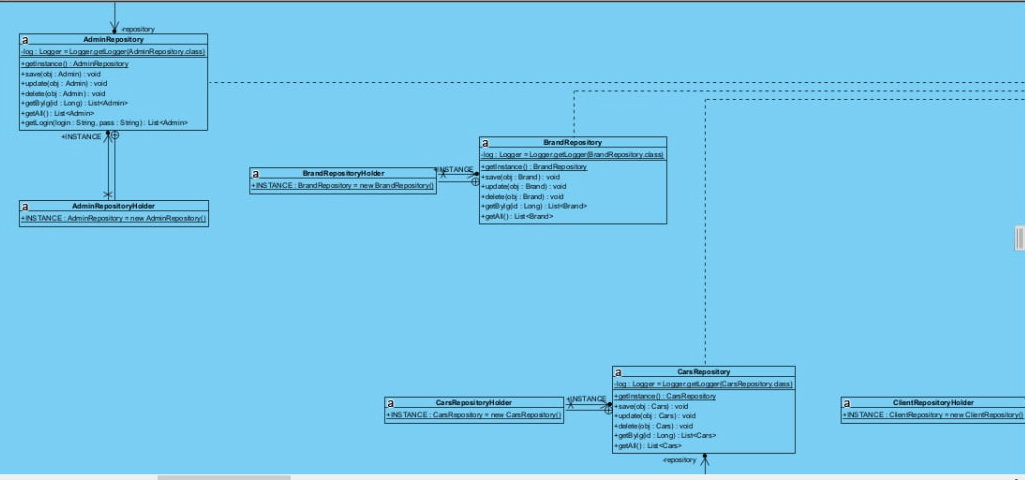


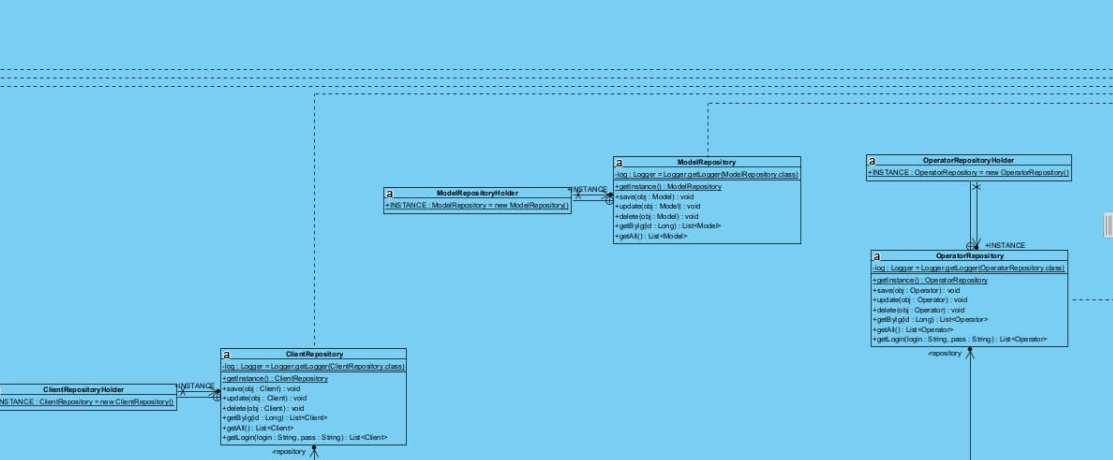


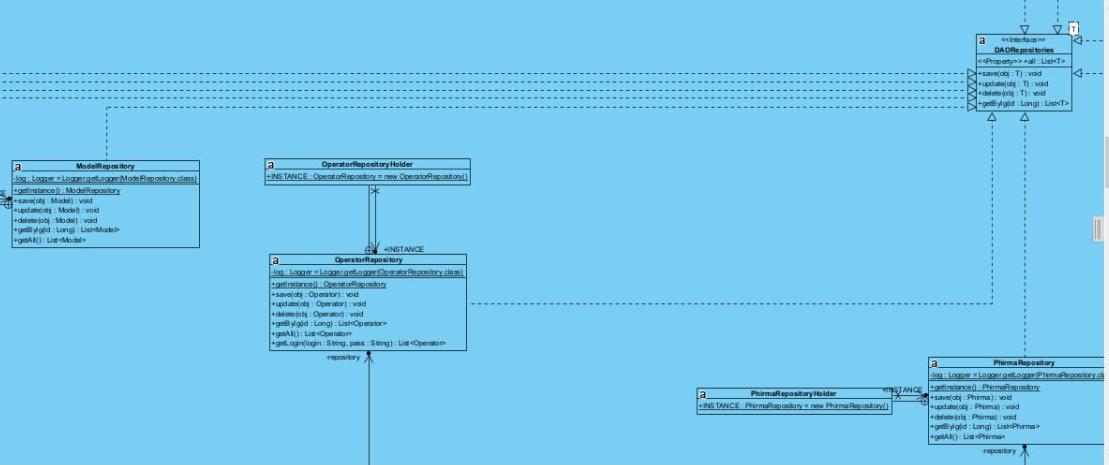


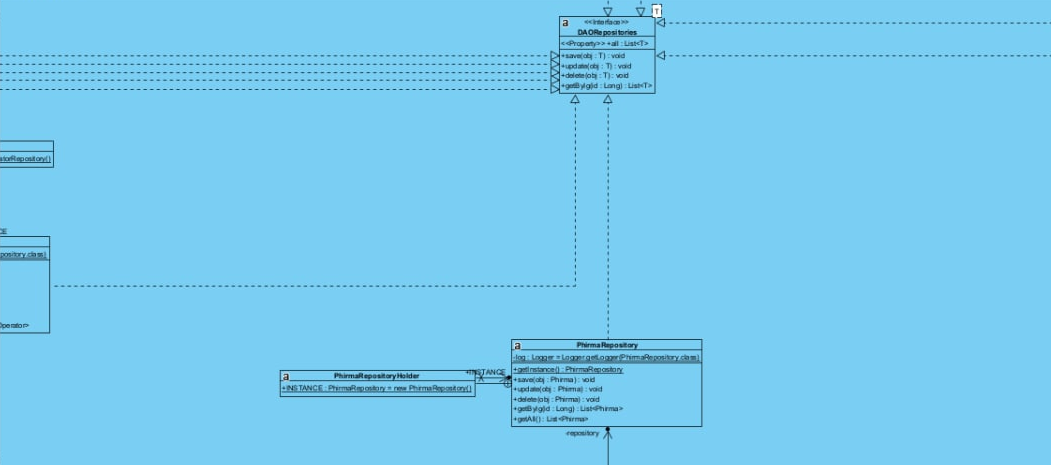


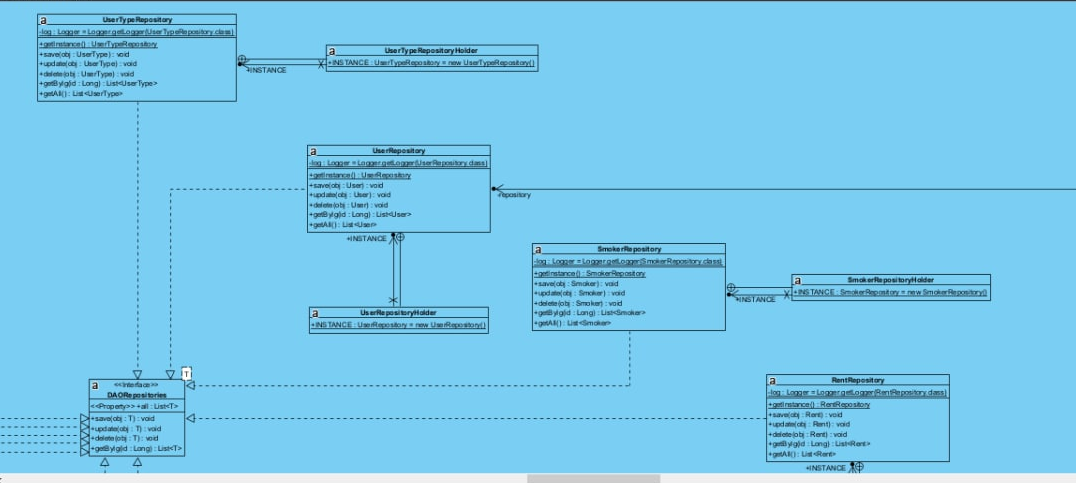


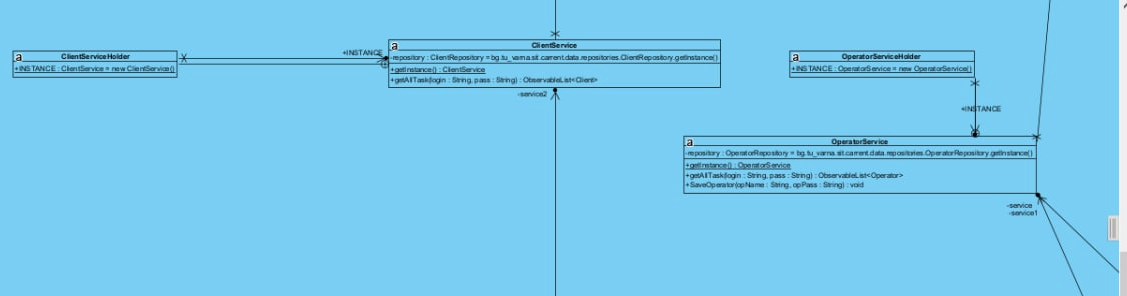


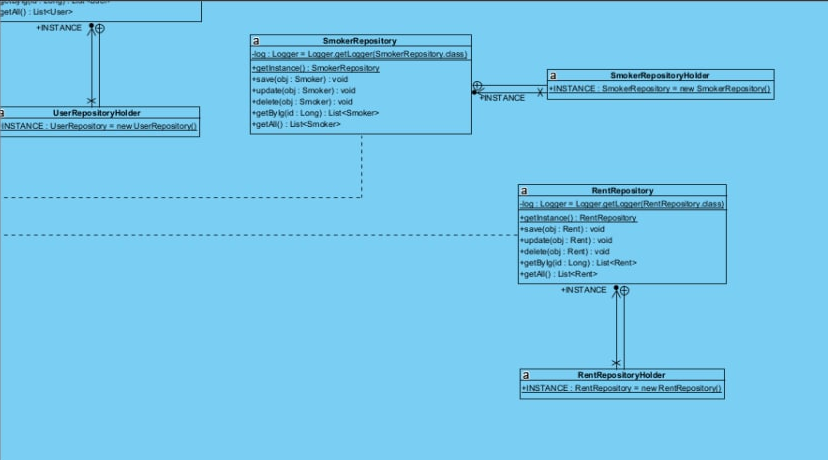


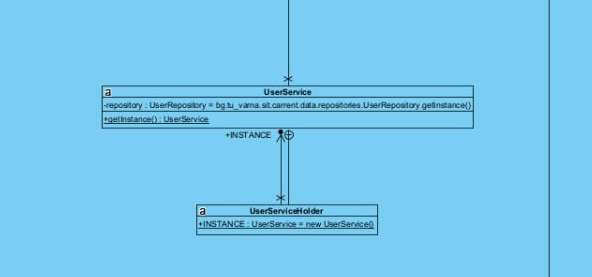


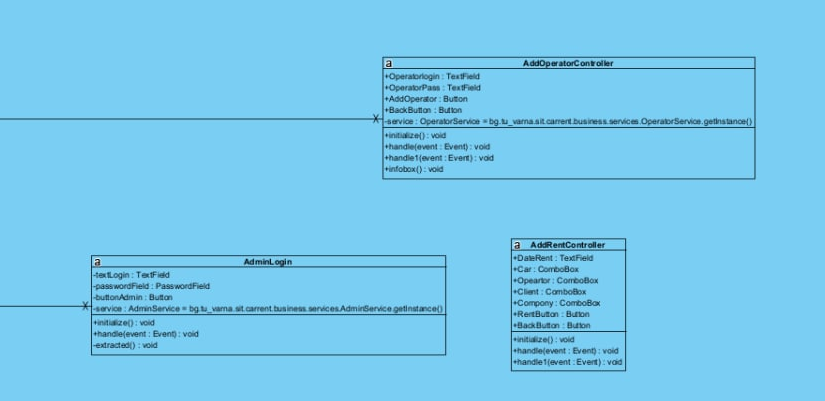


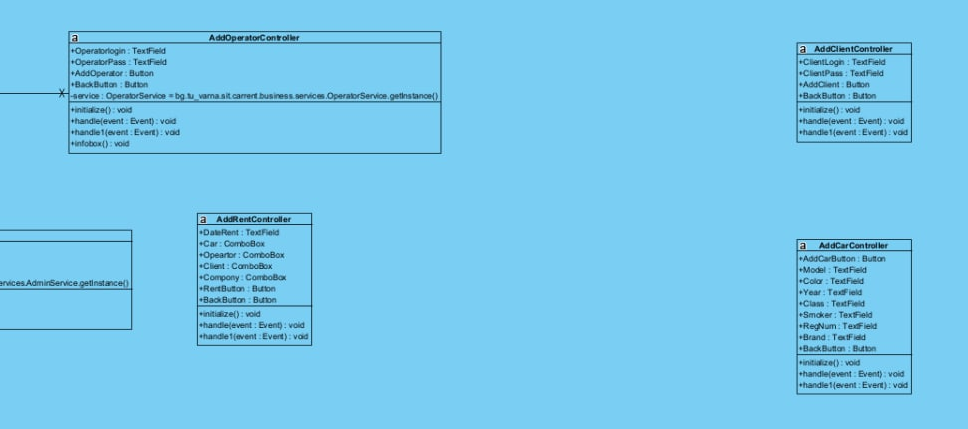


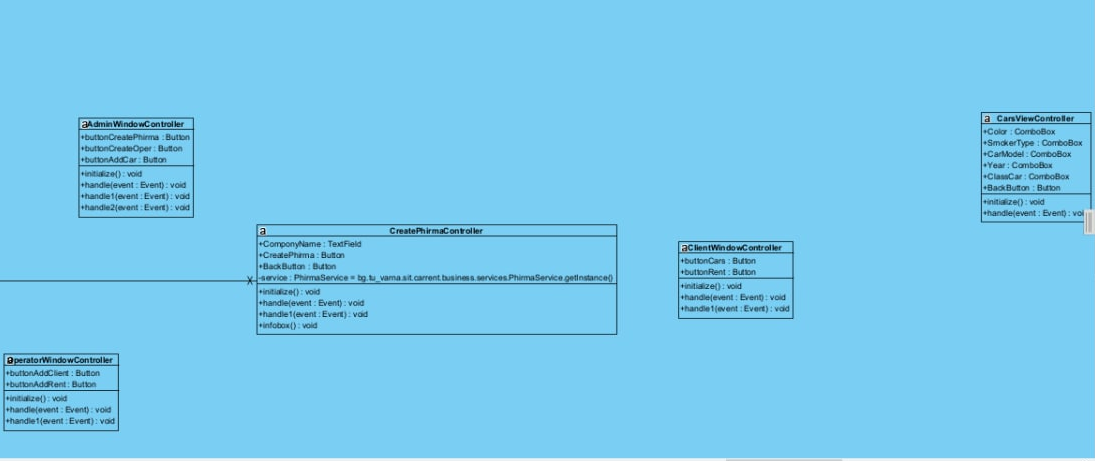


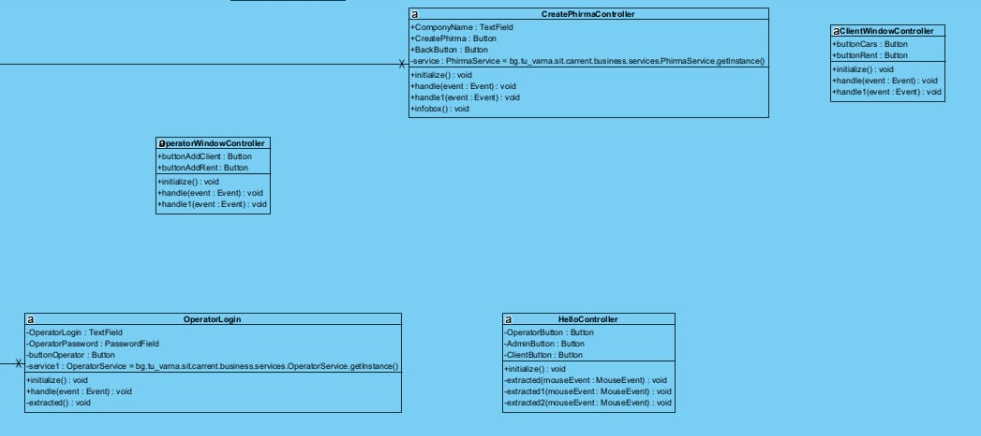


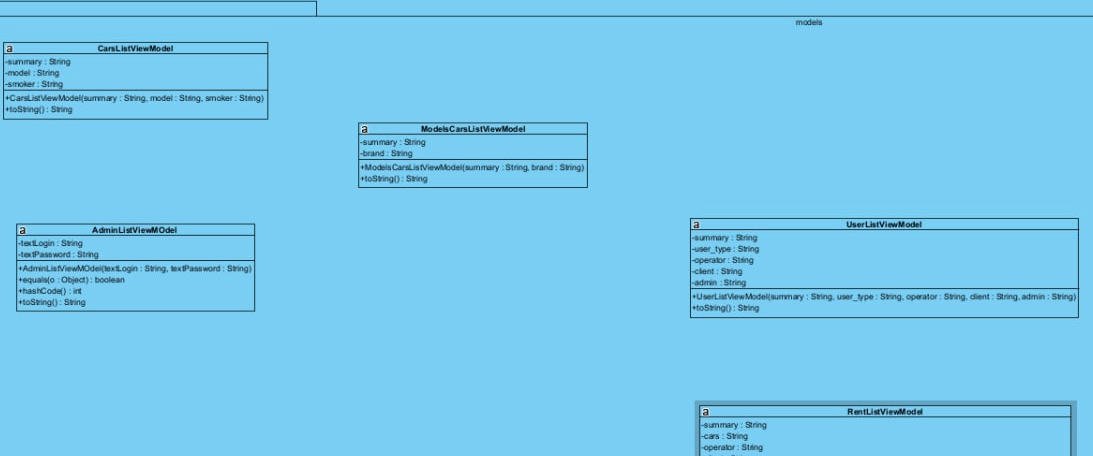


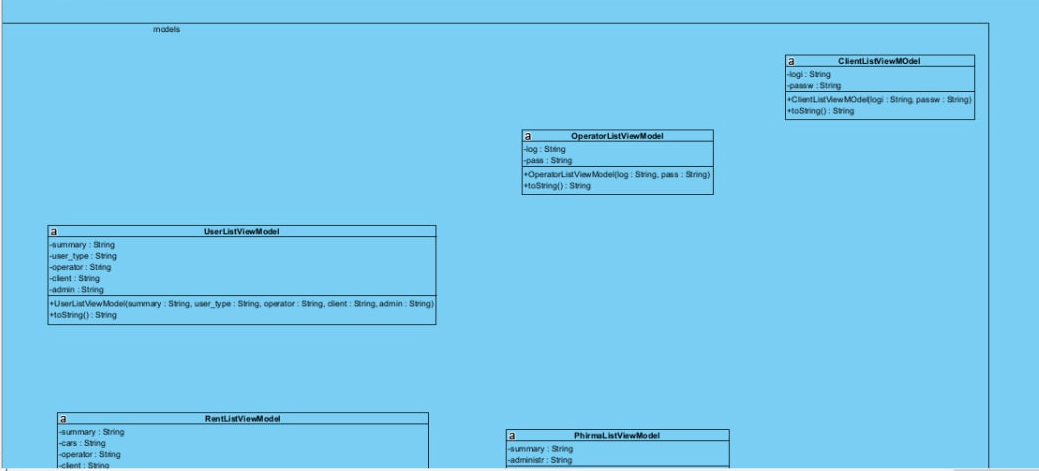


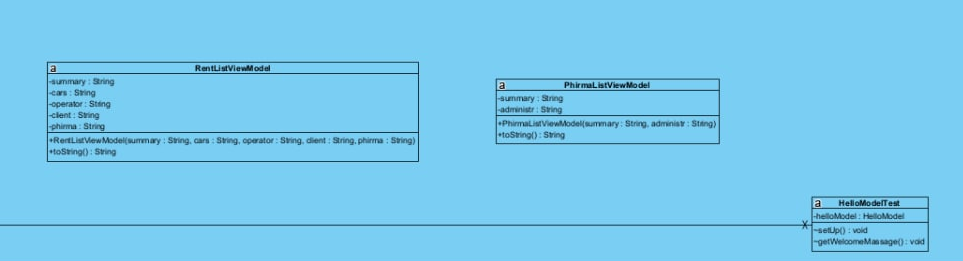


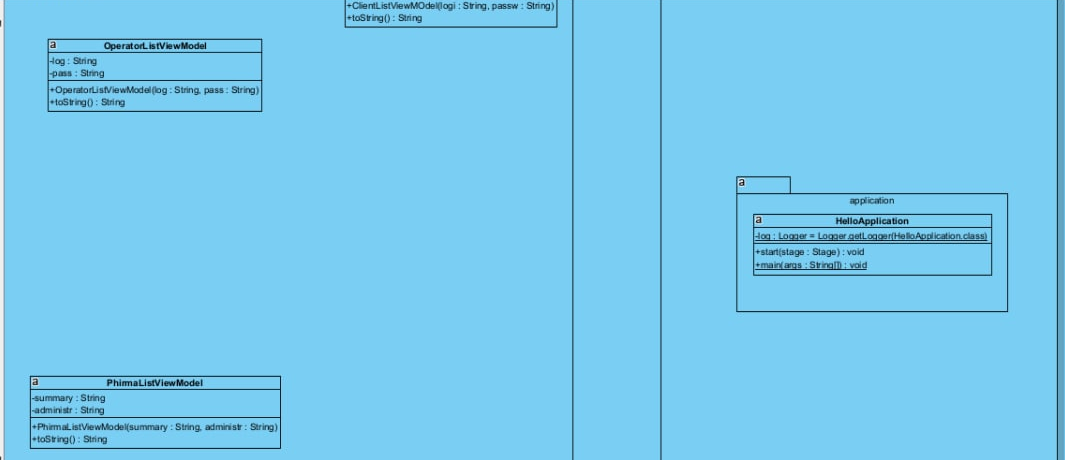




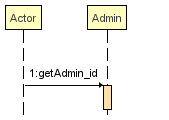
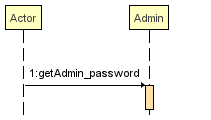
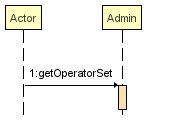
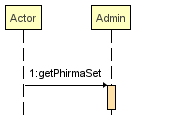
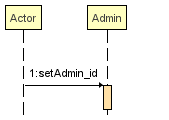
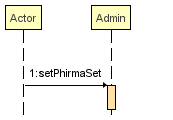
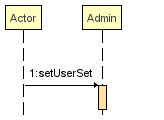
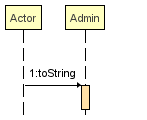
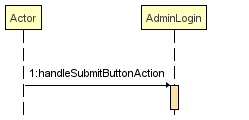
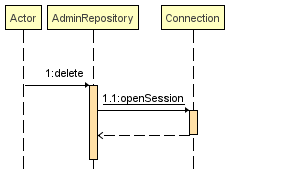
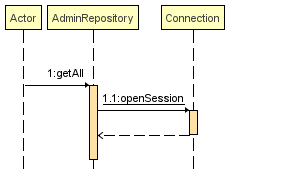
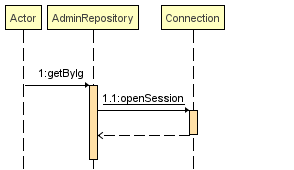
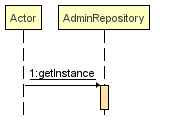
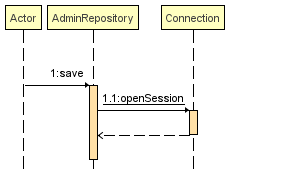
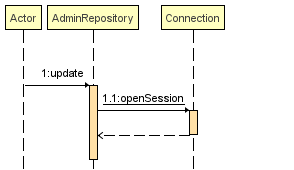
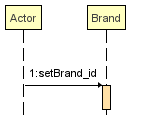
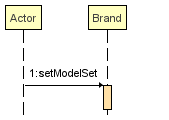
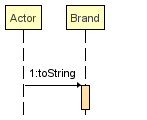
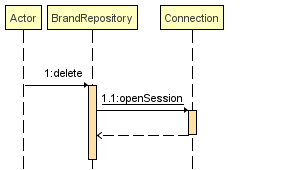
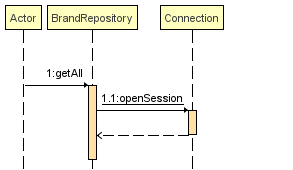
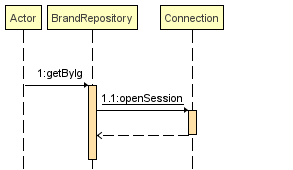
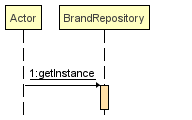
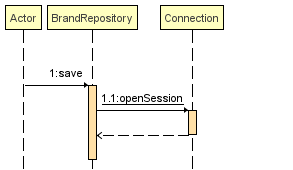
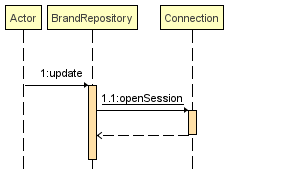
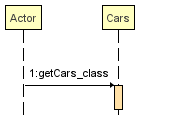
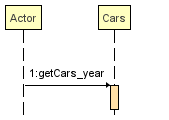
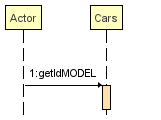
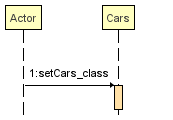
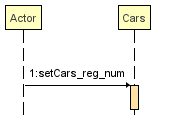
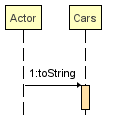
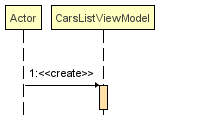
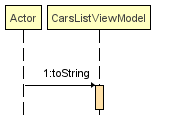
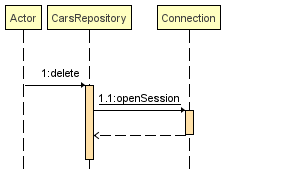
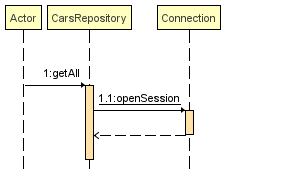
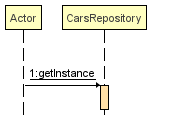
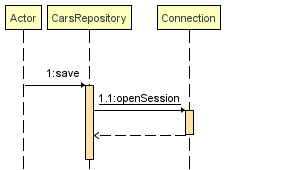
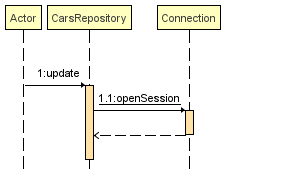
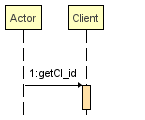
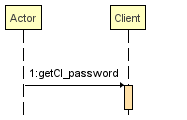
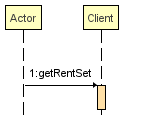
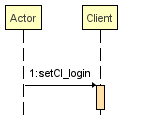
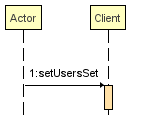
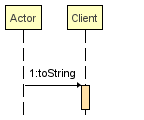
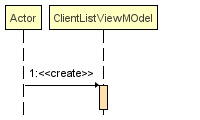
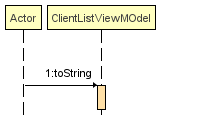
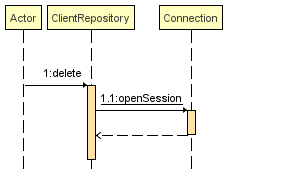
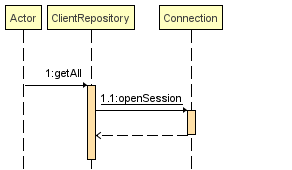
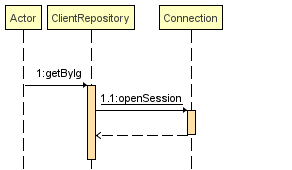
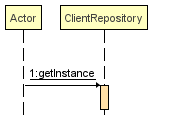
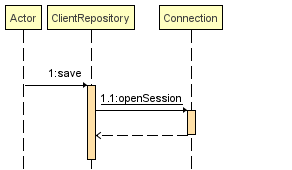
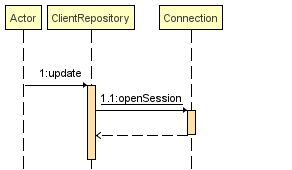
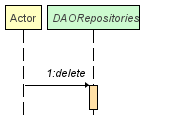
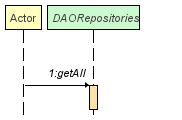
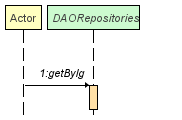
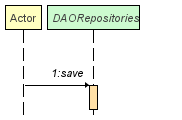
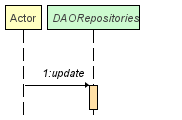
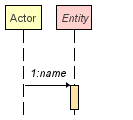
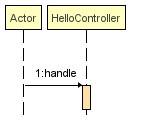
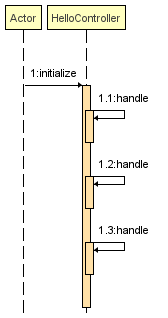
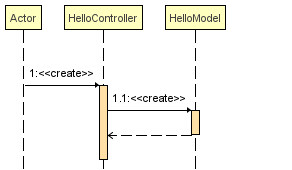
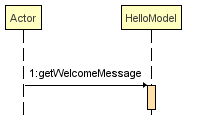
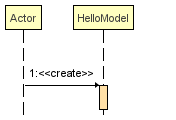
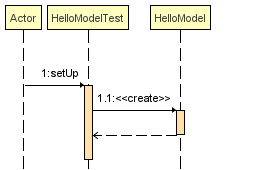
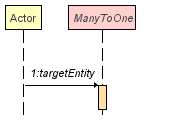
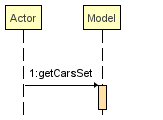
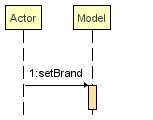
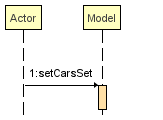
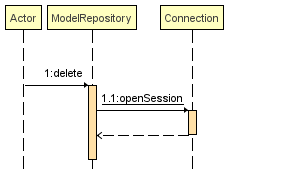
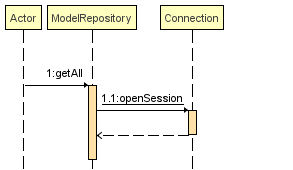
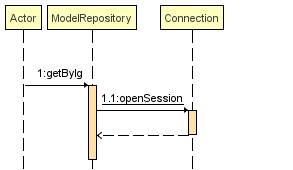
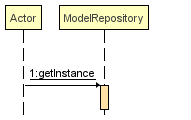
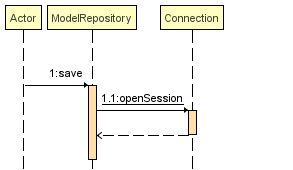
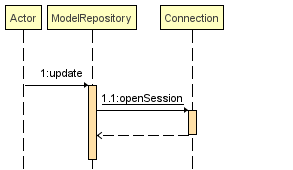
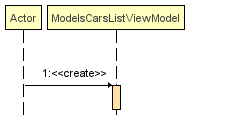
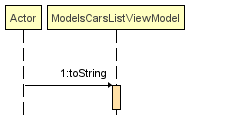
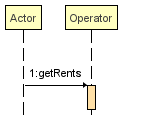
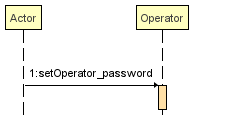
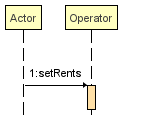
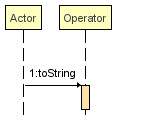
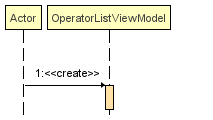
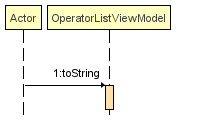
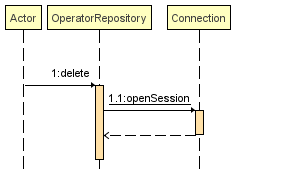
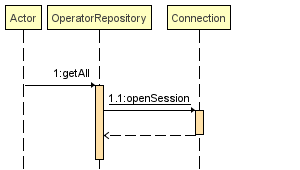
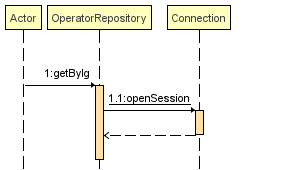
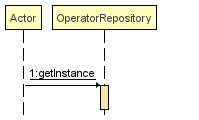
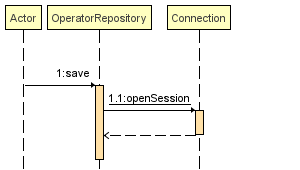
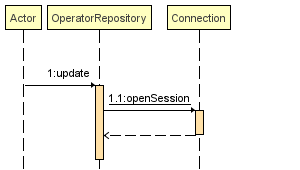
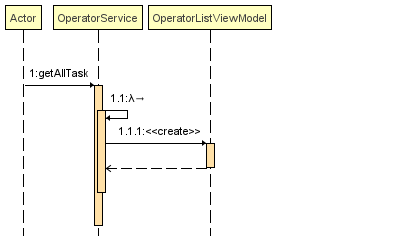
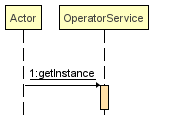
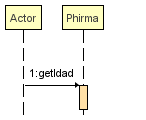
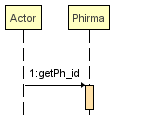
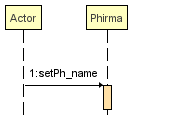
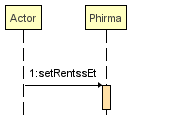
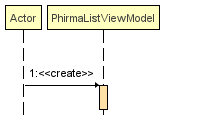
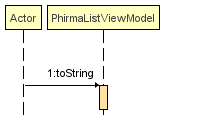
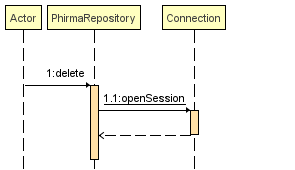
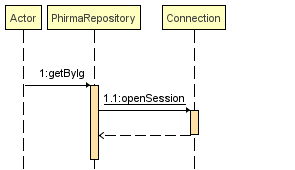
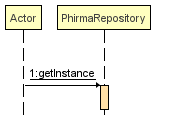
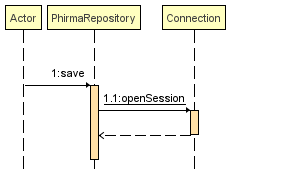
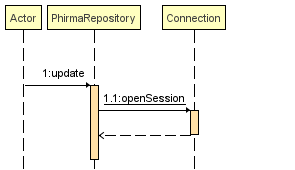
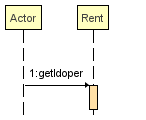
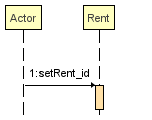
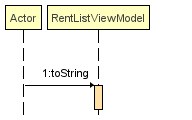
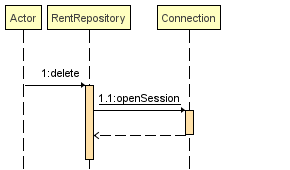
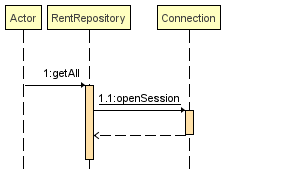
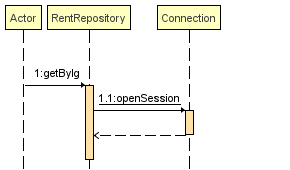
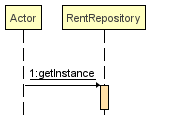
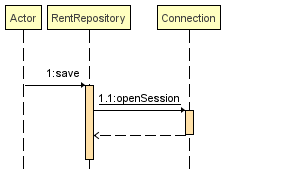
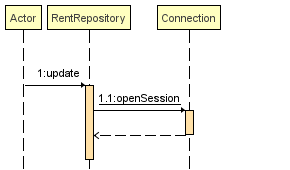
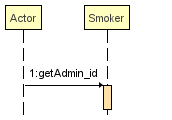
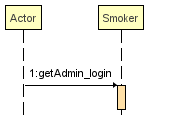
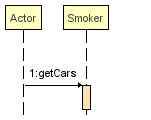
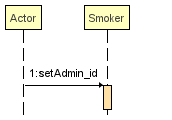
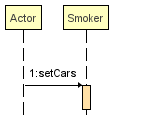
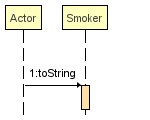
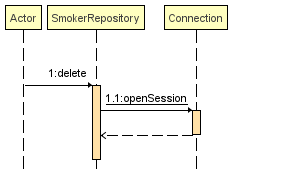
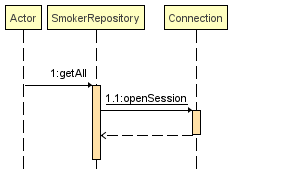
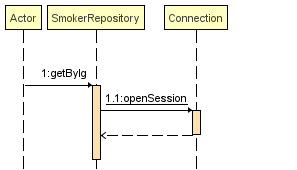
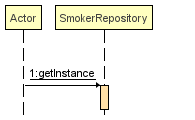
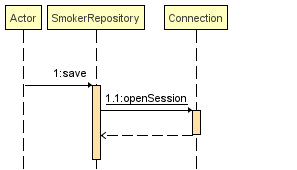
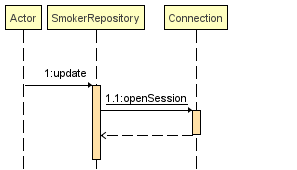
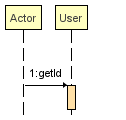
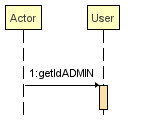
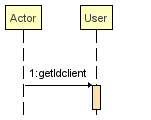
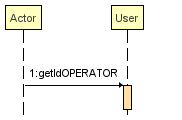
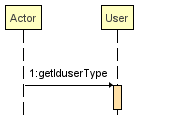
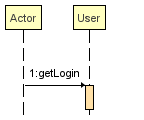
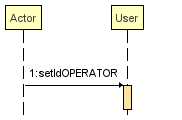
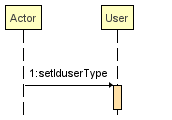
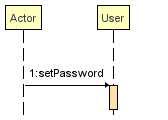
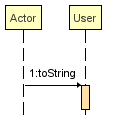
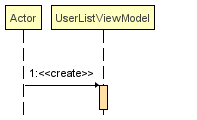
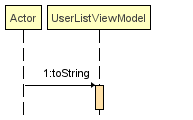
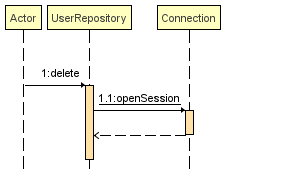
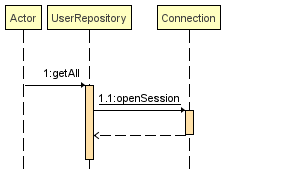
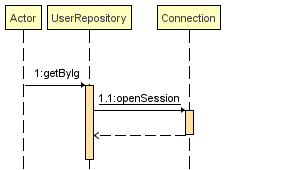
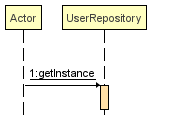
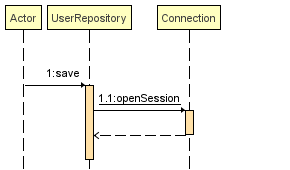
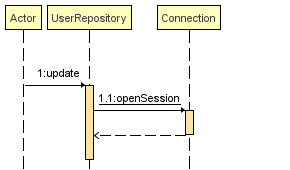
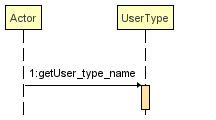
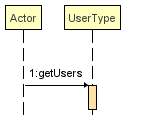
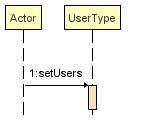
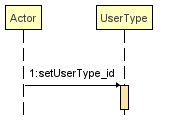
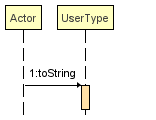
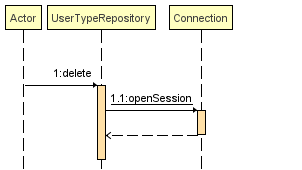
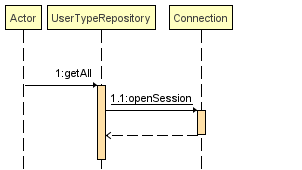
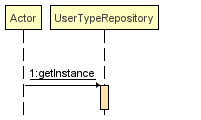
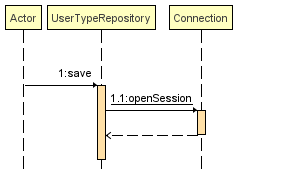
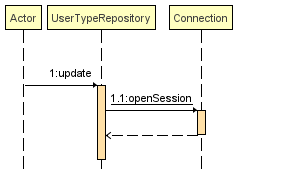




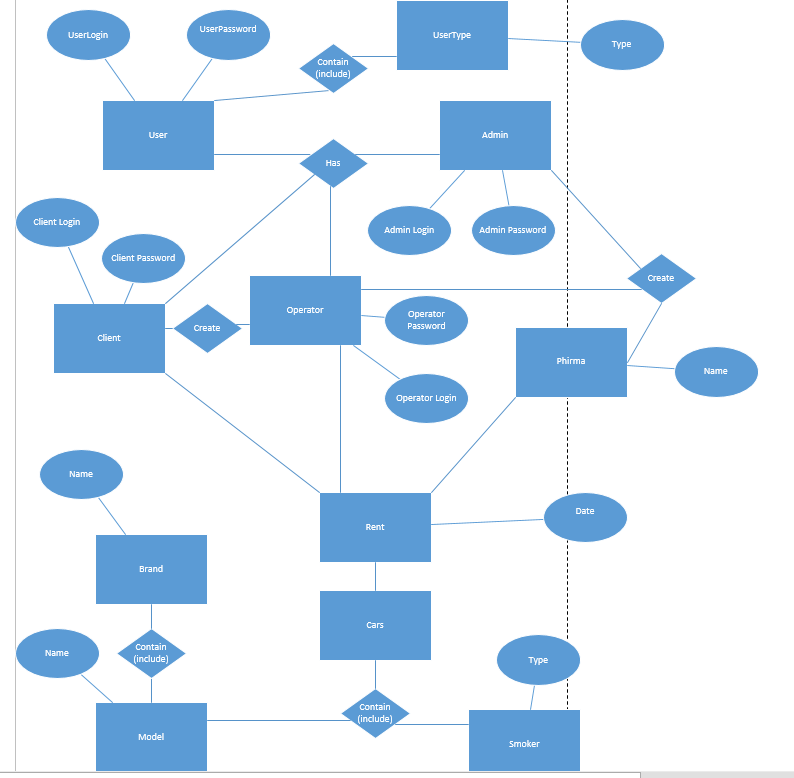




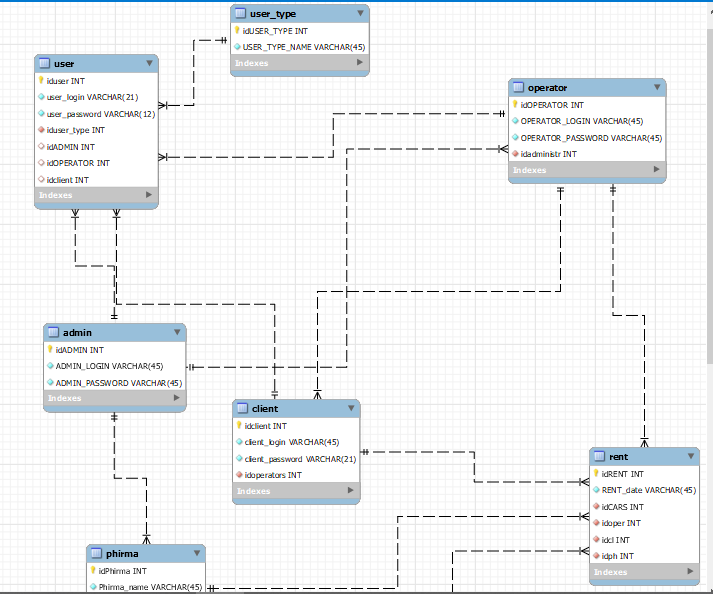
*Sequence diagram:*

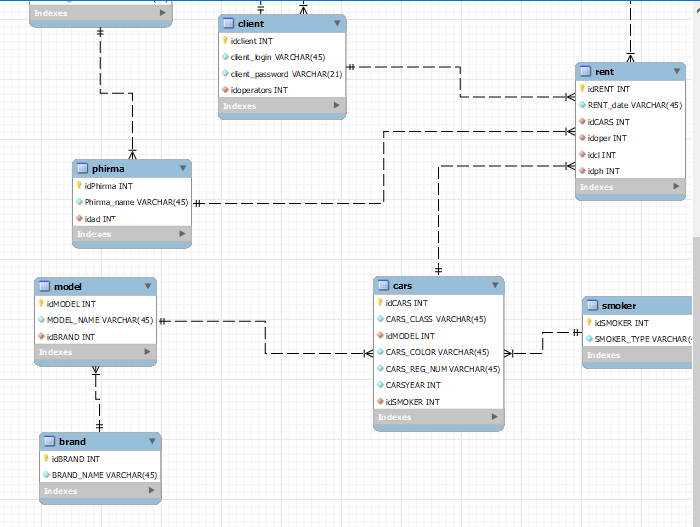


*Концептуален модел на базата от данни (ER диаграма (модел на Чен)):*



*Релационна схема описание на таблиците:*

**

**

*Реализация на слоя за работа с базата данни (връзка с Hibernate чрез конфигурационен фаил):*

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/car\_rent</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">Student19621833</property>

<property name="hibernate.show\_sql">true</property>

<property name="hibernate.hbm2ddl.auto">update</property> <!-- or create-drop -->

<property name="hibernate.enable\_lazy\_load\_no\_trans">true</property>

<property name="show\_sql">true</property>

<mapping class="bg.tu\_varna.sit.carrent.data.entities.User" />

<mapping class="bg.tu\_varna.sit.carrent.data.entities.Admin" />

<mapping class="bg.tu\_varna.sit.carrent.data.entities.Operator" />

<mapping class="bg.tu\_varna.sit.carrent.data.entities.Client" />

<mapping class="bg.tu\_varna.sit.carrent.data.entities.Phirma" />

<mapping class="bg.tu\_varna.sit.carrent.data.entities.Brand" />

<mapping class="bg.tu\_varna.sit.carrent.data.entities.Model" />

<mapping class="bg.tu\_varna.sit.carrent.data.entities.Smoker" />

<mapping class="bg.tu\_varna.sit.carrent.data.entities.Cars" />

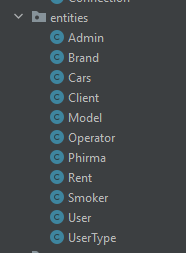
<mapping class="bg.tu\_varna.sit.carrent.data.entities.Rent" />

<mapping class="bg.tu\_varna.sit.carrent.data.entities.UserType" />

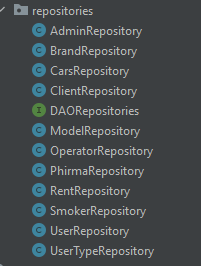
</session-factory>

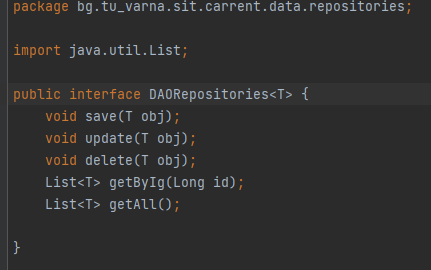
</hibernate-configuration>

В директория entities сложени всички таблици които после mapped в хибернет за да ги свържа с база дани:



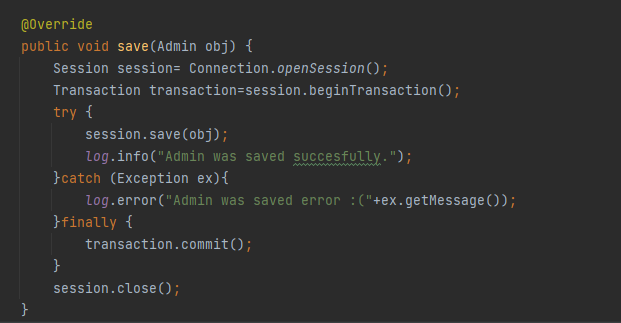
Към тези класове имам репозитории (в които включен интерфейс за функции за всеки клас(таблица):



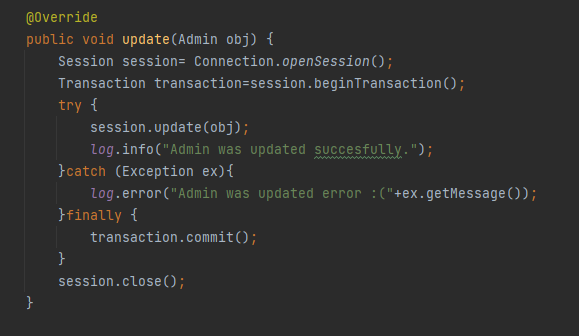


1. Функция save приема като параметр обект от определен клас,

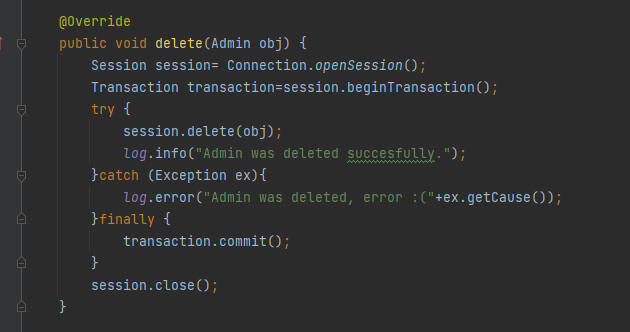
Открива сессия (база дани ) и испълнява insert в този клас на определен параметр. При хващане на exception отпечатва причина в конзола чрез функция getCause().



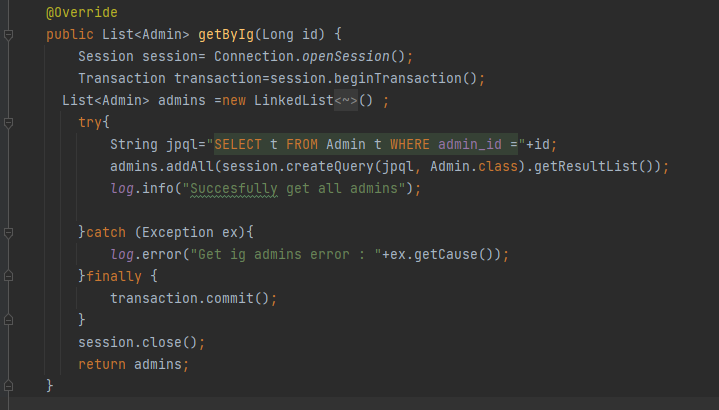
1. Функция update приема като параметр обект от определен клас, открива сессия (база дани ) и испълнява update в този клас на определен параметр. При хващане на exception отпечатва причина в конзола чрез функция getCause().



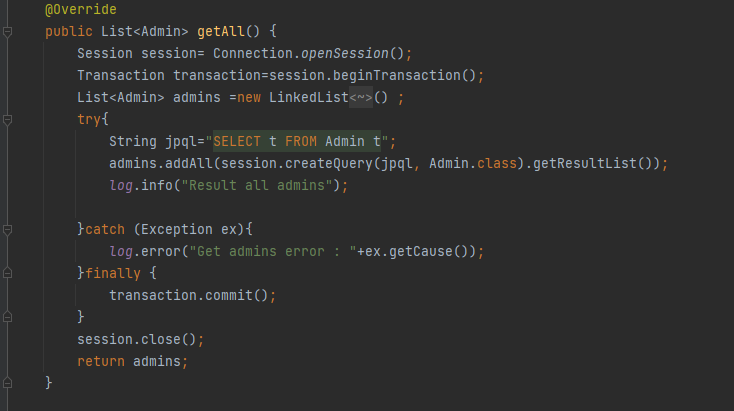
1. Функция delete приема като параметр обект от определен клас, открива сессия (база дани ) и испълнява delete в този клас на определен параметр. При хващане на exception отпечатва причина в конзола чрез функция getCause().



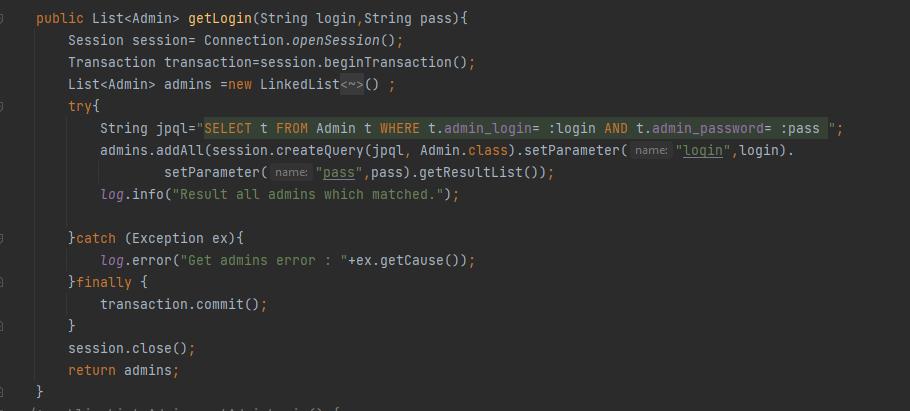
1. Функция от тип List<Име\_на\_клас> getByIg приема като параметр primary key == id (от тип Long) от определен клас, открива сессия (база дани ) , създава нов лист от този клас и испълнява select заявка в този клас къде id == на подаден като параметр id и ги добавя в лист. При хващане на exception отпечатва причина в конзола чрез функция getCause().



1. Функция от тип List<Име\_на\_клас> getAll не приема параметр, открива сессия (база дани ) , създава нов лист от този клас и испълнява select заявка на всички дани в този клас и ги добавя в лист. При хващане на exception отпечатва причина в конзола чрез функция getCause().



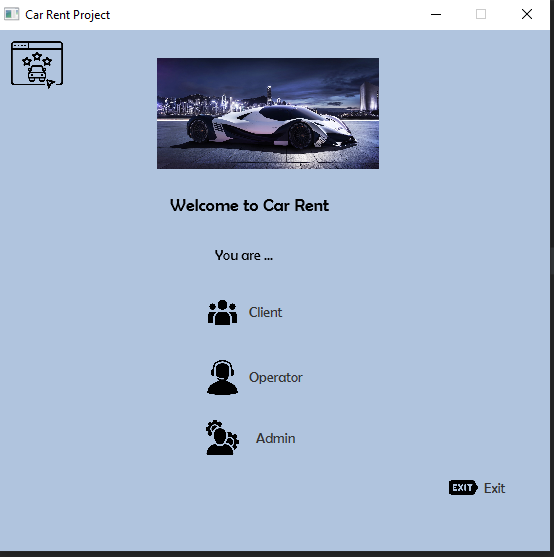
1. При класове за логване(влизане в системата( Администратор, Оператор и Клиент)) имам още една функция getLogin от тип List<Име\_на\_клас> приема като параметр логин и парола въведени от потребителя в текстово поле и преведени в тип String. Открива сессия (база дани ) , създава нов лист от този клас и испълнява select заявка на логин и парола (ако има в базата) и ги добавя в лист. При хващане на exception отпечатва причина в конзола чрез функция getCause().

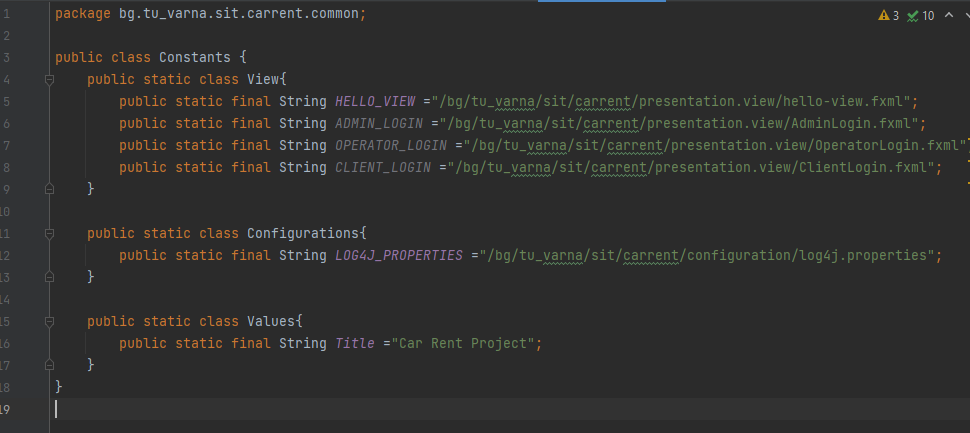


В директория application имам HelloApplication което стартира начална страница на приложение 1) . В директория business.services имам сервисите които свъръват репозиторий(sql заявки) . Контролер вика сервис за да изпълни действие от клиента (някаква операция от репозиторий).

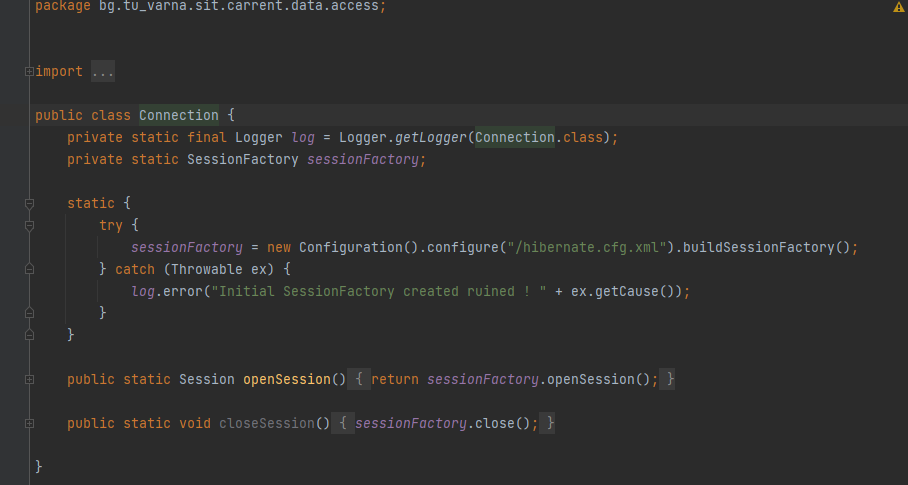
В директория common имам фаил Constants имам път към свързваненто LOG4J и дифиниране на име на първоначална страница проекта.



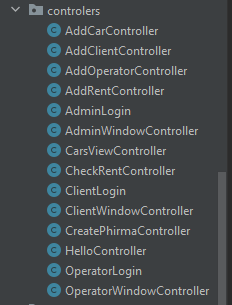
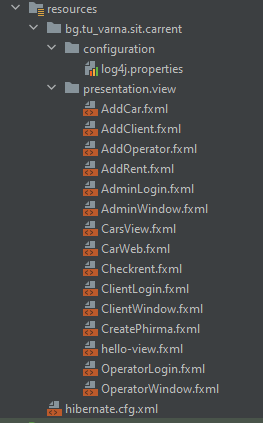
1. 



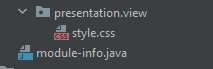
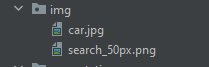
В директория common имам фаил Connection който произвежда конфигурация с Hibernate фаил и открива и закрива session ако не може да свържи с базата хваща exception и отпечатва причина в конзолата чрез функция getCause().

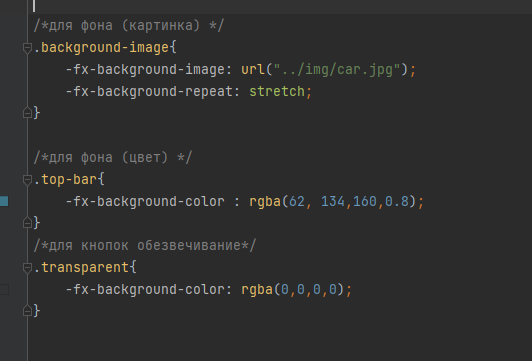


В директория controlers имам контролери към всеки прозорец от ресурсите(fxml файлове).

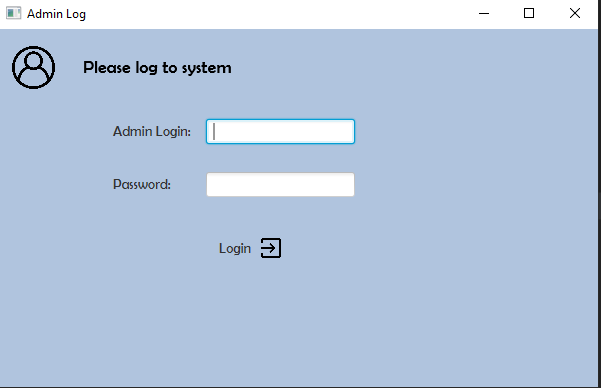
В директория presentation.view имам css фаил за дизайн на приложение в който исползвам background image(от директория img) ,color и transparent background color за бутоните бъдат без цвят.

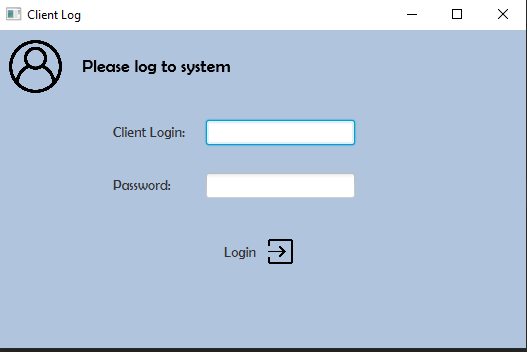


Прозорци за логване(влизане в система) 1) Оператор 2) Клиент 3) Администратор. Ако има такъв потребител в система показва прозорец за извършване на някакво от достъпни на потребителя действие. При натискане на един от бутоните открива нов прозорец за изпълняване на действие.

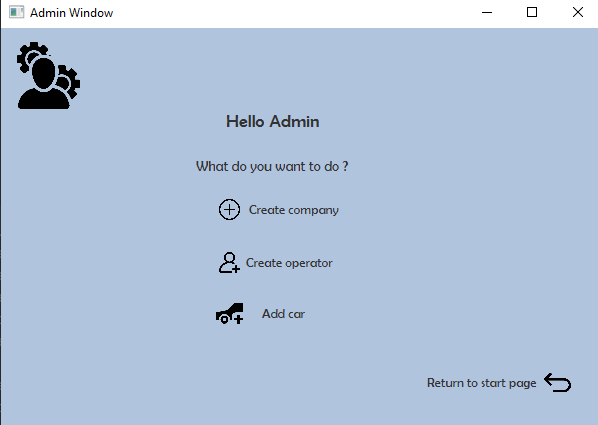
При натискане на бутон Back потребитель връща се към начален прозорец за да изпълни друго действие. В начална странаца на приложение освен бутони за избор на потребитель имам бутон за изход от приложение. При натискане на бутон Table в нов прозорец извежда таблица за определени дани. За връщане към главен прозорец на потребител има бутон Return to Window.

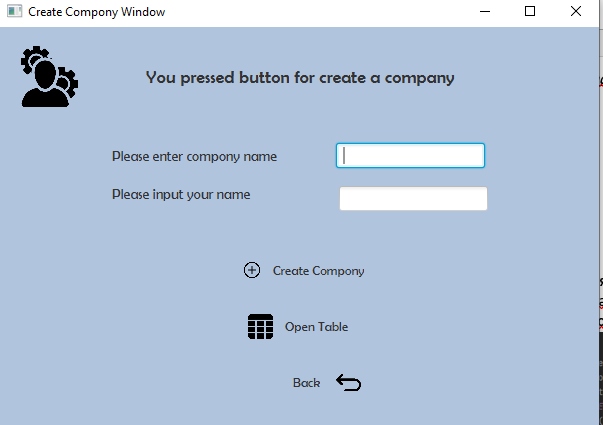


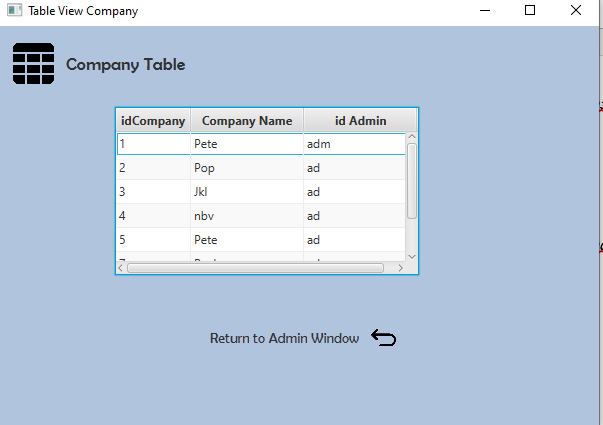




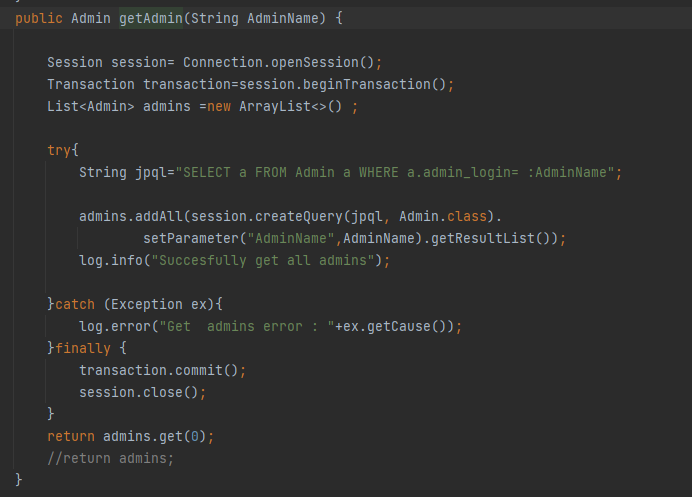
*Прозорец за администратор*

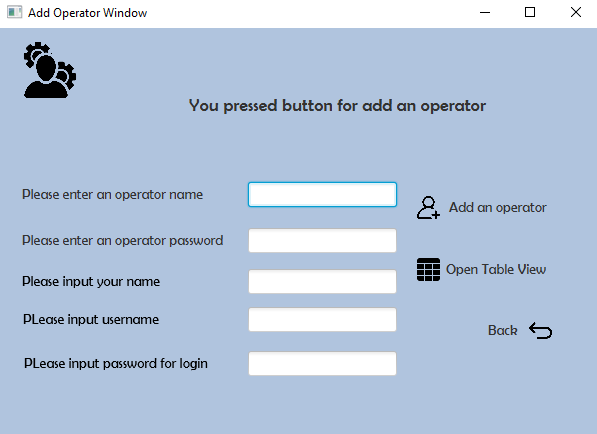


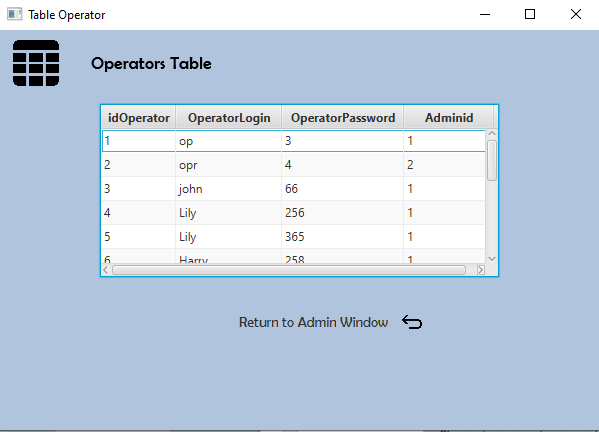




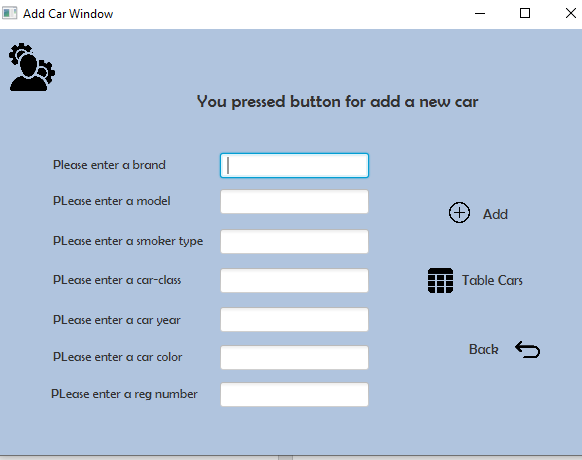
За добаването на нова фирма администратор ввежда име на фирма и своето име. Чрез функция save администратор съхранява име на фирма и id на администратор чрез търсене по име чрез функция getAdmin(String AdminName). При успешно добаването извежда съобщение че фирмата бяха успешно добавена.



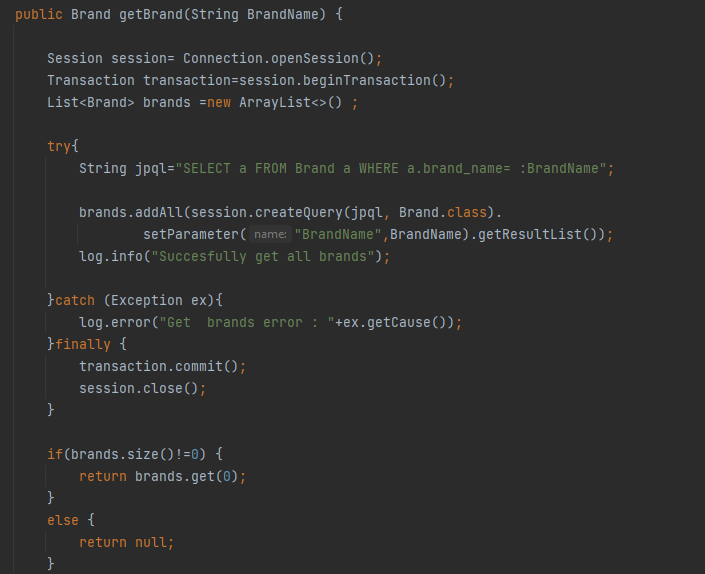




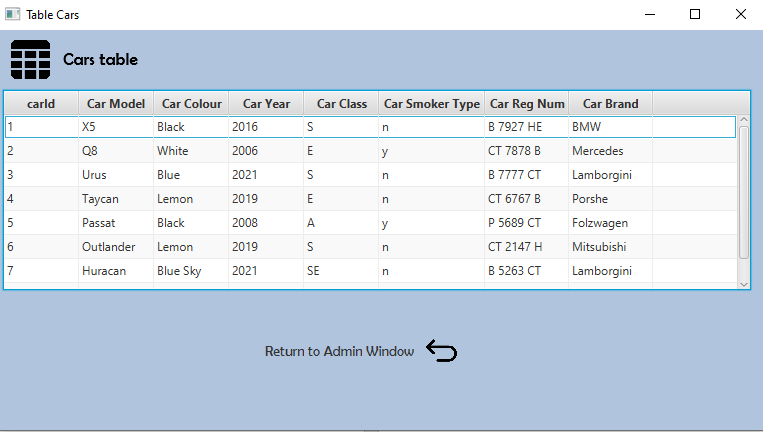
За добаването на оператор администратор ввежда имената на оператор , неговата парола своите имена и логин и парол за вход в система. При save имената на оператор , парола и взема id на администратор по име чрез функция getAdmin(String AdminName) съхранява се в таблица за оператори , username and password for login добавят се в таблица на потребител за логване в система. При успешно добаването извежда съобщение че оператор бяха успешно добавен.



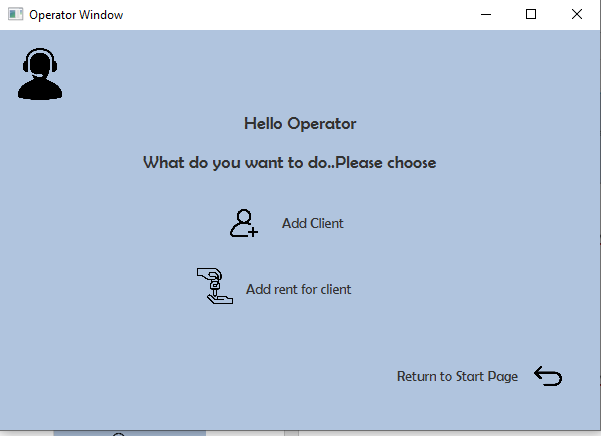
След натискане на бутон add введени дани за кола добавят се втаблица коли с вземане на id model (чрез функция getModel(String ModelName)),id brand(чрез функция getBrand(StringBrandName)),id smoker\_type, ако няма така кола по brand или model те се автоматически добавят в таблица. Като за smoker\_type няма добаването. При успешно добаването извежда съобщение че кола бяха успешно добавена.

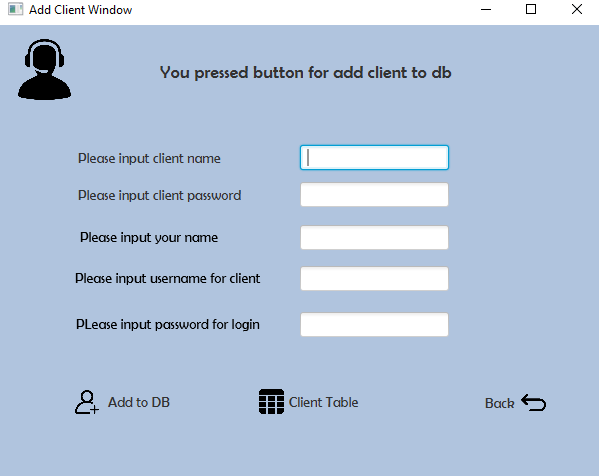


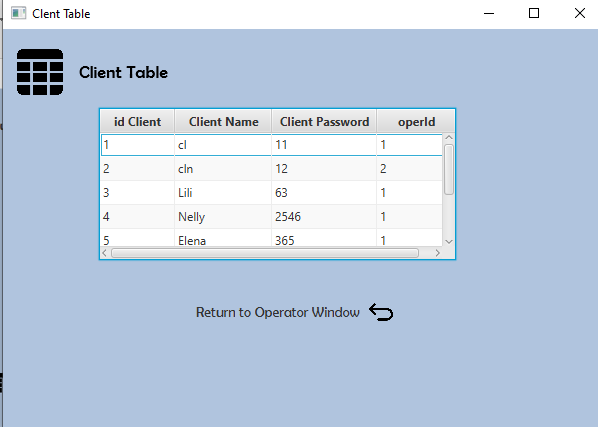




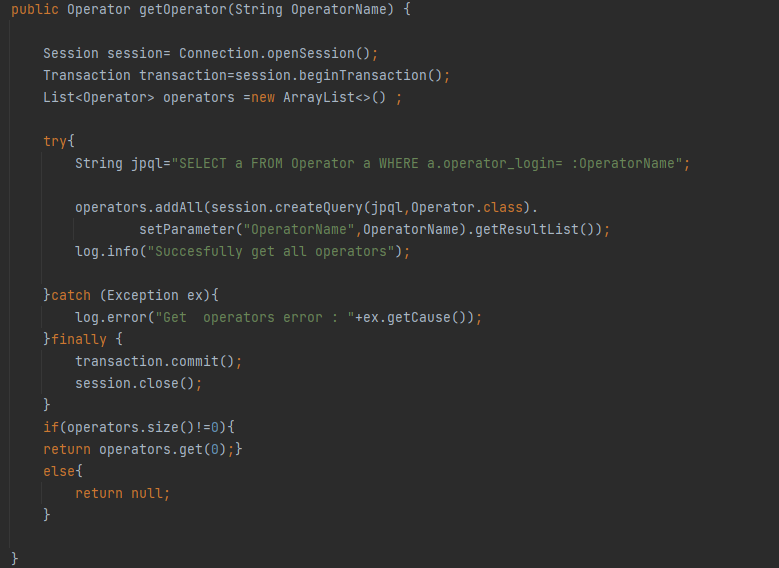
*Прозорец за оператор*

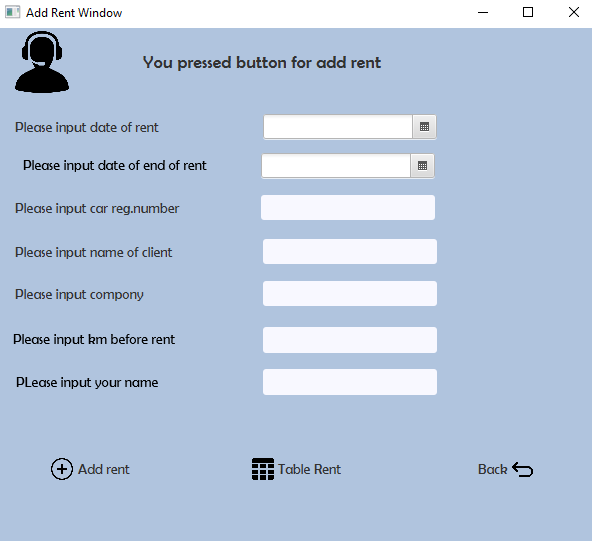


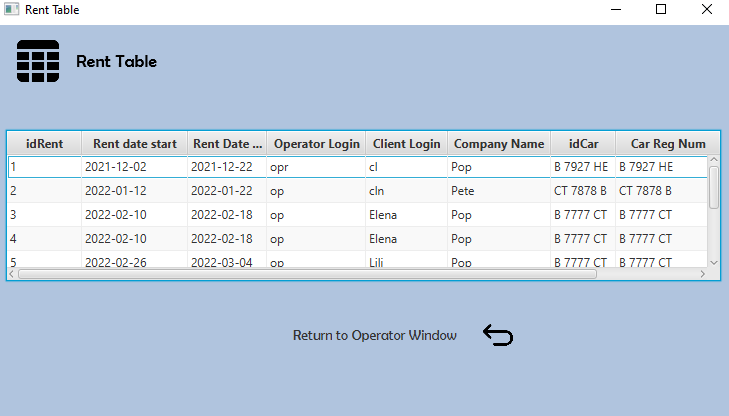




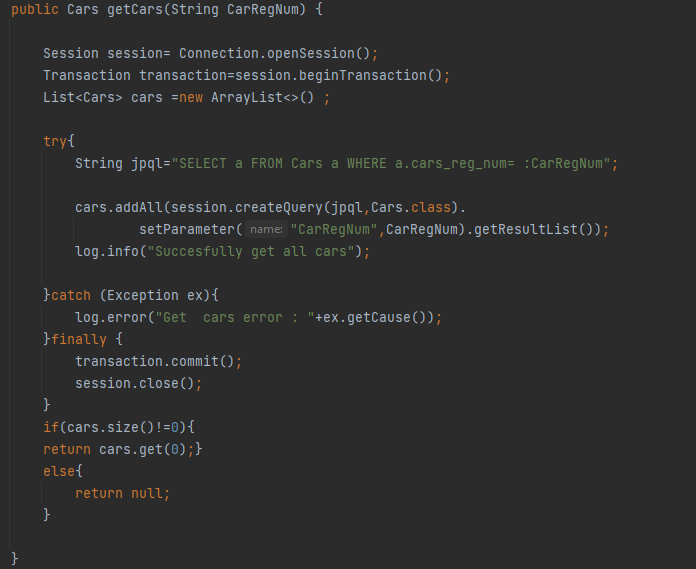
За добаването на клиент оператор ввежда имената на клиент , неговата парола своите имена и логин и парол за вход в система. При save имената на клиент , парола и взема id на администратор по име чрез функция getOperator(String OperatorName) съхранява се в таблица за клиенти , username and password for login добавят се в таблица на потребител за логване в система. При успешно добаването извежда съобщение че клиент бяха успешно добавен.

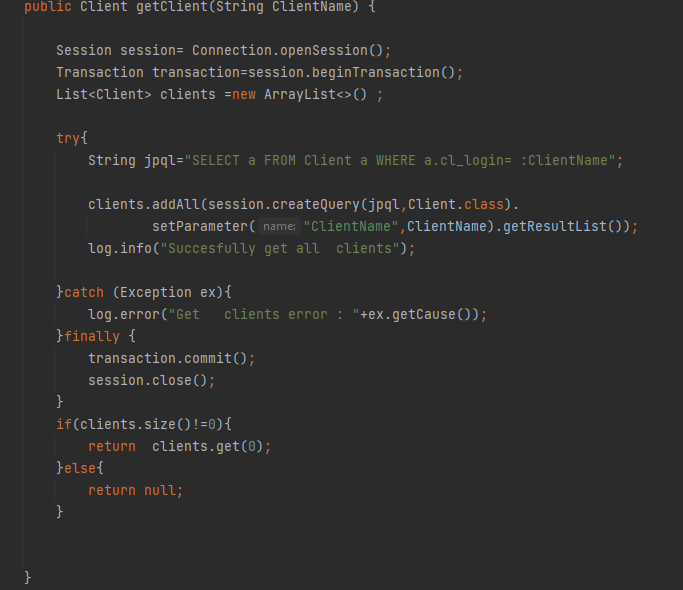


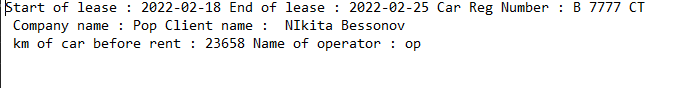


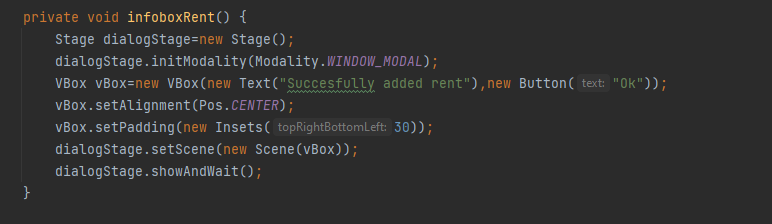


При добаането на нов наем оператор избира дати за начало и край наем. Ввежда регистрационен номер на кола (по който чрез функция getCar(StrinCarRegNum) вземаме и добаваме id на кола в таблица rent ) , имената на клиент (чрез функция getClient(String ClientName) вземаме id на клиент за таблица наем) , име на фирмата(чрез функция getPhirma(String PhirmaName) вземаме id на фирма за таблица наем), пробег на кола в км преди наем и име на оператор който регистрира този наем.При успешно добаването извежда съобщение че оператор бяха успешно добавен(чрез функция infoboxRent()) и записва този наем в текстов фаил за протокол на текущо състояние(RentCars). Ако оператор въведох некорекни дани относно име на клиент,регистационен номер на кола, име на компания или своето име извежда съобщение с съответна грешка за да провери този поле с информация.

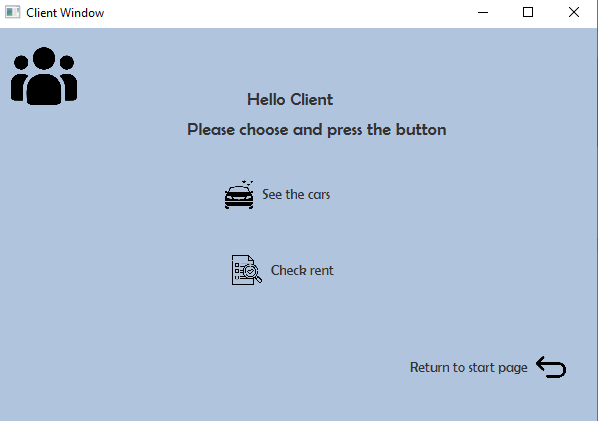


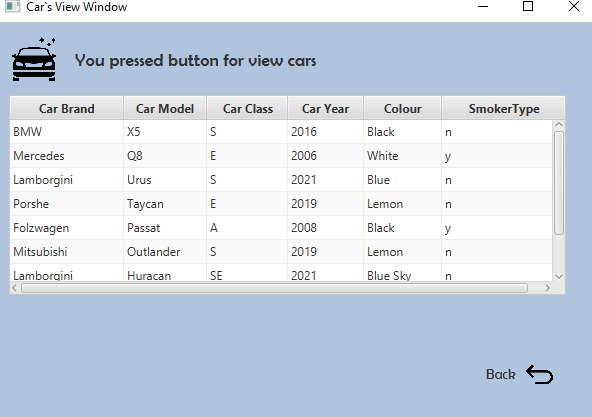
**

**

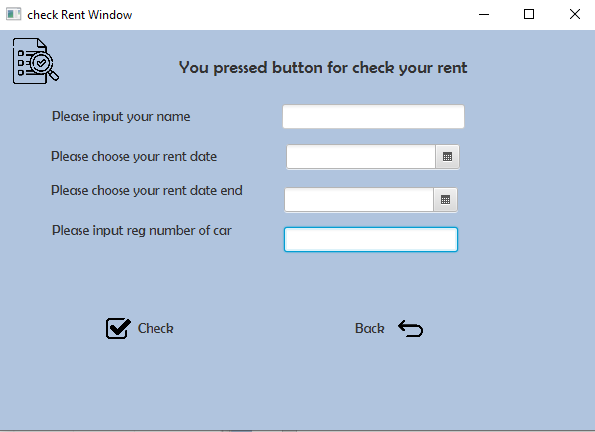
**

*Прозорец за клиент*

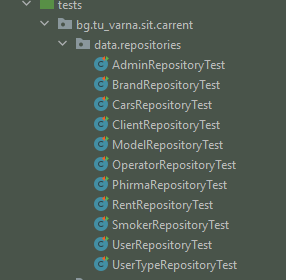




При натискане на бутон See the Cars извежда нов прозорец къде в вида на TableView клиент може да разгледа всички коли в системата.



Клиент може още да провери своят наем (действителен ли той). Той въвежда дани (своите имена, избира дати за начало и край наем и ввежда регистационен номер на кола за наем), след натискане на бутона check система проверява начало и край дати на наем в таблица наем чрез CheckRent( LocalDate dateStartRent,LocalDate dateEndRent), имена на клиент в таблица клиент и регистрационен номер на кола в таблица коли . При успешна проверяване извежда съобщение че наем е действителен. Ако няма таково име на клиент , регистрационен номер на кола или дати извежда съответно съобщение за да клиент провери введена информация в този поле.



В директория tests имам Junit тестове за всички функции в репозитории.







