**Технически университет – Варна  
Специалност: СИТ**

**К У Р С О В**

**П Р О Е К Т**

**Тема: Пътнически превози**

**Изготвил: Проверил:……….**  
Кристиян Костадинов 19621798

Условие на проекта

Да се разработи информационна система, предоставяща услуга пътнически превози.  
Програмата съхранява и обработва данни за разпространение на билети.

Системата поддържа два вида потребители администратор и клиенти (пътническа  
компания, разпространител, касиер) с различни роли за достъп до функционалностите  
в системата.

* Операции за работа с потребители:
  + Създаване на пътническа компании от администратор;
  + Създаване на разпространители от администратор;
  + Създаване на касиери от разпространител
  + Поддържане на профили с характеристики на клиентите в системата (хонорар и др…)
  + Рейтинговане на клиентите в системата
* Системата поддържа операции за работа със пътувания:
  + Добавяне на ново пътуване от организатор (вид на пътуването, дестинация, дата на заминаване и пристигане, брой места, вид(ове) транспорт, ограничение в закупуването на билет от едно лице и др…);
  + Заявяване на билети за продажба от разпространител и потвърждаване от пътническа компания
  + Продаване на билети за пътуване от касиер, създаване на формуляр за закупуване (информация за купувача, избор на място и др…)
* Системата поддържа Справки по произволен период за:
  + Клиентите в системата:
    - Пътническа компания с наличните пътувания (за разпространители);
    - Разпространители
    - Касиер
  + Закупени билети (дата, статус, и др…);
  + Пътувания;

Пътническата компания достъпва справки само за пътувания, на който е организатор.  
Разпространителя достъпва справки за всички актуални пътувания. Касиерите има право на справки само за пътувания чиито билето-разпространители са.

* Системата поддържа Известия за:
  + Новопостъпила заявка за пътуване (в профила на разпространител);
  + Периодично уведомление за продадени билети от пътуване (в профила на собственика).
  + Отменено пътуване (в профила на касиера и разпространителя)
  + Наближаващо пътуване с не продадени билети (собственик, разпространител)

Анализ на проблема

1.1 Въведение

В настоящата курсова работа е разработена система, предоставяща услуга за пътнически превози, като основните функции на системата са да обработва данни за разпространение на билети.

1.2 Функционални изисквания и дефиниция на модулите

Системата поддържа два вида потребители: администратор и клиент, които имат различни роли за достъп до функционалностите на системата. Администраторът има достъп до всички функции в програмата, докато клиентът е ограничен спрямо добавянето на нови компании и разпространители.

Системата дава възможност за управлението на пътувания създадени от организатор. Поддържа заявяване на билети за продажба от разпространители и потвърждаване от компания. С помощта на системата могат да се продават билети за пътуване от касиер и да се създават формуляри за закупуване. Чрез нея се генерират справки и отчети за съответно нейните клиенти, закупени билети и пътувания.

Системата извежда съобщение за:   
- Новопостъпила заявка за пътуване,  
- Периодично уведомление за продадени билети от пътуване  
- Отменено пътуване  
- Наближаващо пътуване с непродадени билети

Системата е реализирана на програмен език Java, като е използвана технологията JavaFX на Scene Builder за изграждането на потребителския интерфейс. За хранилище на данните, се използва база от данни на Oracle.

Всички събития в системата се запазват във файл, поради наличието на модул log4J.

Когато премине изпълнението на дефинираните задачи следва тестване (и корекции при откриване на грешки и несъответствия). Проверява се дали коректно функционират нововъведенията в самостоятелен режим на работа. При откриване на грешки, те се отстраняват и системата отново бива подлагана на тест.

1.3 Структура на проекта

Кратко описание за структурата - често изборът на правилната структура прави програмата по-ефективна, тъй като се спестява памет и време за изпълнение. Данните биват групирани по определен начин, за да се улесни достъпът до тях и управлението им. За различни задачи са подходящи различни структури. За по-сложни случаи могат да се създадат много по-сложни структури.

Курсовата работа има следната структура, която е съставена от 4 части:

**Част първа** „Анализ на проблема” – Кратко описание на системата и нейните функционални изисквания.

**Част втора** „Проектиране на системата” – Разработване на базата от данни, потребителския интерфейс, UML диаграми.

**Част трета** „Реализация на системата” – Реализация на слоя за работа с базата от данни – DAO с достъп чрез JBDC, реализация на графичния интерфейс, както и на модула за регистриране на събития в системата - log4J.

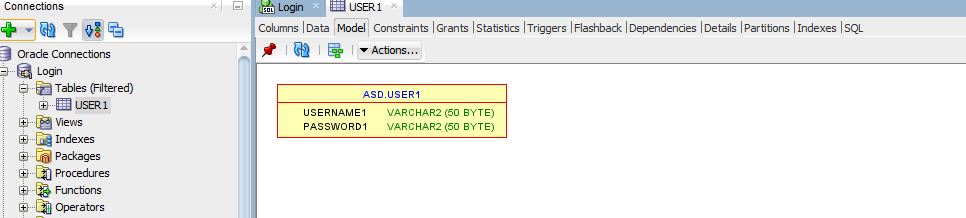
**Част четвърта** „Тестови резултати” – Разработване на тестове за проверка на функционалността на системата.

Проектиране на системата.

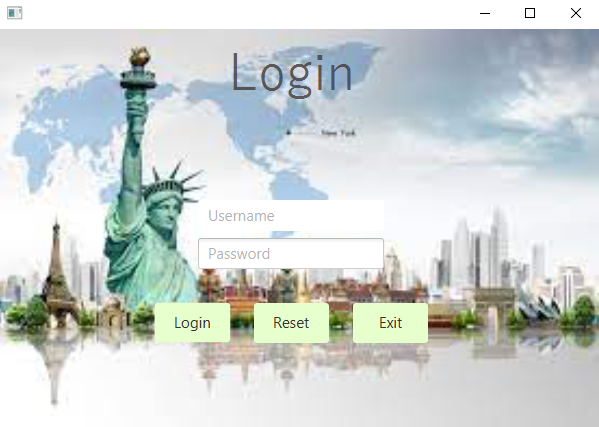
* 1. Проектиране на отделните модули

А) **База данни** - Проектиране на базата данни – Data Modeler, Oracle

* + - Въведени са пробни данни с една таблица (username1, password1) за проба на връзката с базата данни.



Б) **Проектиране на GUI** – чрез Scene builder, където се добавя всеки контролер и се разработва.

1. **Логин форма** (администратор или клиент)
2. **Главна Форма (USER):**



1. **Главна Форма (ADMIN):**



* 1. UML диаграми:

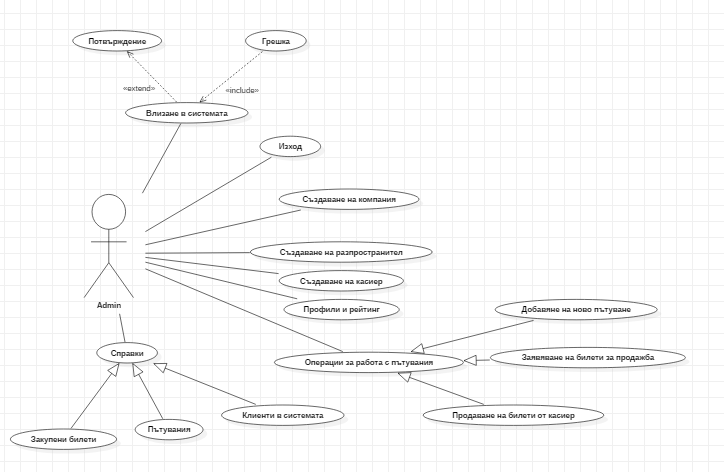
Изработено на **StartUML**

Основни участници в системата са:

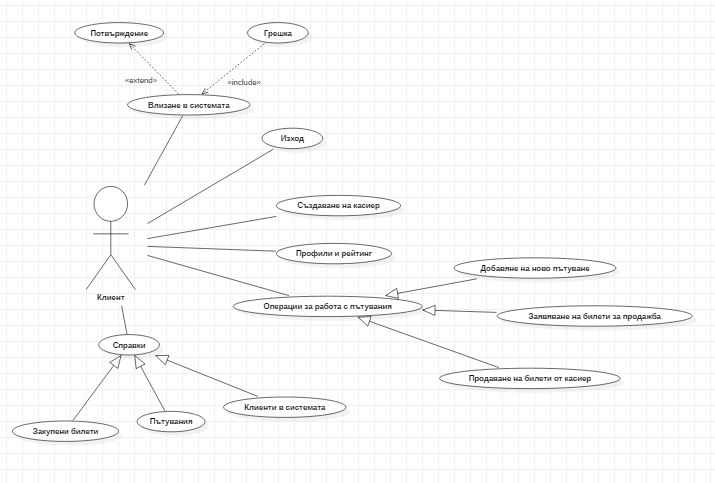
• **Администратор**

**• Клиент**

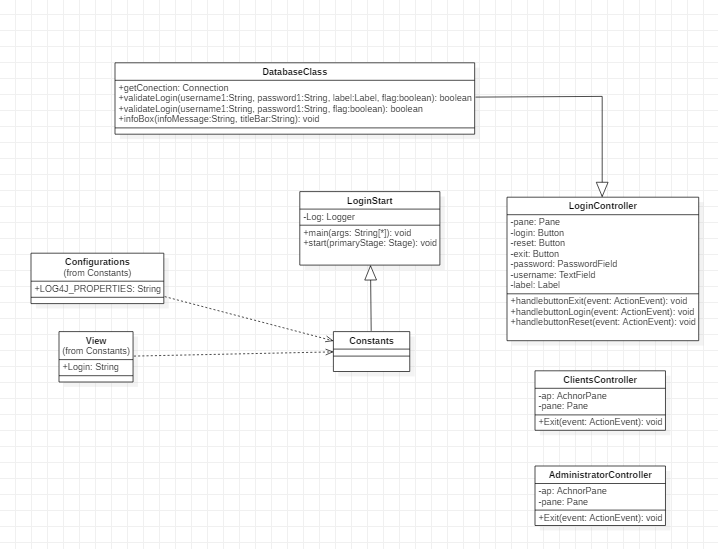
А) **Use case диаграма за администратор:**



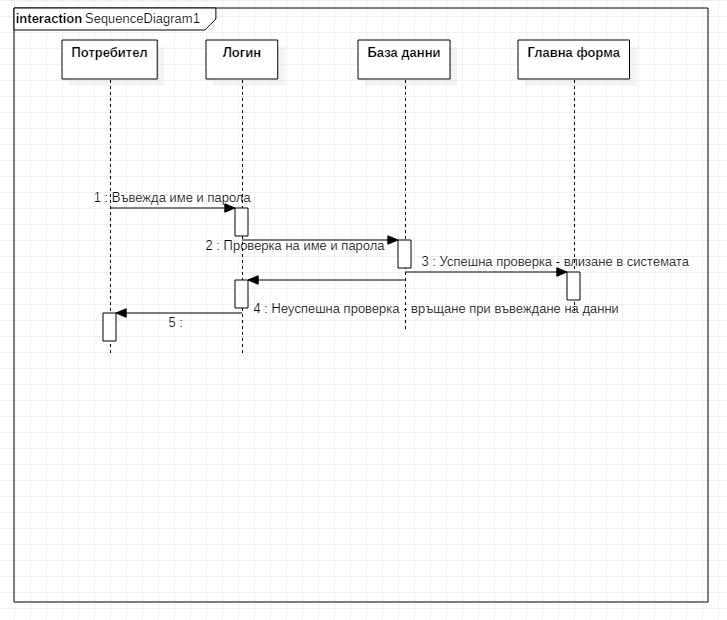
Б) **Use case диаграма за клиент:**



В) **Class диаграма**:



Г) **Sequence диаграма:**



Реализация на системата.

* 1. Реализация на базата от данни (Oracle)

С помощта на **SQL Developer**, създавам таблицата и въвеждам данни в нея, които използвам за съхранение на информацията.

Създаване на таблица **USER1,** която съдържа следните данни:

|  |  |
| --- | --- |
| USERNAME1 | Име на акаунт в системата |
| PASSWORD1 | Парола на акаунт в системата |

* 1. Реализация на слоя с базата данни (DAO)

public static Connection getConnection() throws SQLException  
{  
 Connection com = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe","ASD","1234");  
 return com;  
}

Следната функция прави връзка към сървъра на базата данни, чрез **JDBC driver** като в конекцията (връзката) се задават **hostname** – името на сървъра/хоста, **port number** – номер на порт към който сочи хоста. Съответно ако връзката има име и парола и те се въвеждат, за да може да се свърже успешно.Също така и **listener (xe)**, който е много важен, защото ако е различен от този, който е създаден в Oracle свързването ще бъде неуспешно. Това е свързване с базата данни и когато това се осъществи, могат да се използват ресурсите и данните от там.

DАО позволява на системата да се свърже и да се използват различни функции и данни от базата данни, които могат да се реализират в проекта.

Следният код ще покаже един пример за свързване и проиграване на таблиците от базата данни чрез data access object (DAO)

public static boolean validateLogin(String username1, String password1, Label label, boolean flag) throws SQLException  
{  
 Connection com= *getConnection*();  
 Statement st=com.createStatement();  
 ResultSet rs=st.executeQuery("SELECT \* FROM user1 WHERE username1='"+username1+"' AND password1='"+password1+"'");  
 if (rs.next())  
 {  
 flag=true;  
 }  
 else  
 {  
 label.setText("Check username and password!");  
 *infoBox*("Invalid Credentials", "Warning");  
 flag=false;  
 }  
 return flag;  
}

Чрез тази функция описана по-горе се вижда как се използва най-пряко DAO(data access object), чрез свързване към сървъра на базата данни и използване на заявка за търсене на **username и password от таблица User1**, за да може да се потвърди дали името и паролата съществуват в базата данни и да върнат съответния резултат от изпълнението.

3.3 Реализация на бизнеслогикa и графичен интерфейс – Java, JavaFX

А) Интерфейс Java, JavaFX

Интерфейсът който използваме е разновидност на потребителски интерфейс, в който елементите, предоставени на потребителя за управление, са изпълнени във вид на графични изображения (менюта, бутони, списъци и др.).

Софтуерният интерфейс предлага достъп до ресурсите на компютъра – памет, микропроцесор и т.н. Позволява директен достъп до такива ресурси, където софтуерът може да има много разклонения – понякога катастрофални – за функционалността и стабилността.

Потребителският слой е представен от **Java, JavaFX форми**, чрез които се манипулират данните в базата.

Използван е **JavaFX Scene Builder** – инструмент за визуално оформление, който позволява на потребителите бързо да проектират потребителски интерфейси на приложението JavaFX, без кодиране. Потребителите могат да плъзгат и пускат компоненти на потребителския интерфейс в работна област, да променят свойствата си, да прилагат таблици със стилове и FXML кодът за оформлението, което създават, се генерира автоматично във фонов режим. Резултатът е **FXML файл**, който след това може да се комбинира с проект на Java чрез обвързване на потребителския интерфейс с логиката на приложението.

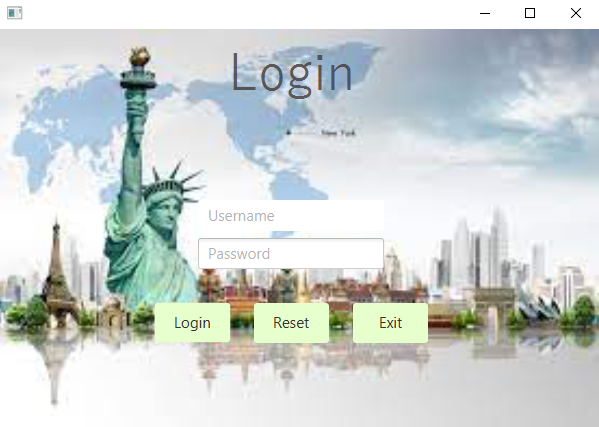
Инструментът улеснява работата с графичен интерфейс и ни дава възможност за по-добра реализация на проекта. Съдържа автоматично генериране на код и задаване на имена на контролери на класове, пакети, бутони и други компоненти, които се разработват в контролер класа в по-късен етап.

Използвани са библиотеки на Java и JavaFX, за да могат да се вкарат всички компоненти от FXML документа в кода на Java проекта и така се стартират успешно, без никакви грешки и проблеми.

Първо е създадена формата за въвеждане, която включва следните компоненти и техните обяснения:

* **Label** – това е компонент, в който могат да се пишат различни текстове и думи;
* **Username field (text field)** – това е компонент, в който могат да се пишат текстови полета като име и други;
* **Password field (text field)** – това е компонент, в който могат да се пишат текстови полета като парола и други;
* **Button** – това е компонент от тип бутон за извършване на действия – при активирането му могат да се зададат различни функционалности на избраните детайли;
* **Pane** – това е компонент от тип прозорец, в който могат да се слагат отделни компоненти.
* **ImageView** – илюстрация на снимка, която се зарежда чрез избор на местоположението на файла .

Графично изобразяване на формата за въвеждане.



След формата за въвеждане се активира главната форма, която включва следните компоненти и тяхните обяснения:

* **Label** – това е компонент, в който могат да се пишат различни текстове и думи;
* **Button** – това е компонент от тип бутон - за извършване на действия – при активирането му могат да се зададат различни функционалности на избраните детайли;
* **Pane** – това е компонент от тип прозорец, в който могат да се слагат отделни компоненти;
* **ImageView** – илюстрация на снимка, която се зарежда чрез избор на местоположението на файла .
* **SplitPane** (horizontal) – който съдържа 2 AnchorPane 30% на 70%, които разделят менюто на 2 части и го подравняват хоризонтално.
* **VBox** – е част от единия AnchorPane, и е компонент за подравняване на различните бутони във вертикална колона – едни върху други.

Графично изображение на главната форма:

Б) Бизнес логика

Бизнес логиката е описание на функционалността на системата и действията, които се развиват.

Примерна бизнес логика включваща приложението за услуга – пътнически превози.

Първо се стартира Login формата, като използваме класа Login Start, чрез който зареждаме формата.

В системата могат да влязат 2 типа потребители, с различни права на достъп:

* + - **Администратор**
    - **Клиент**

При влизане в системата като **клиент** се зарежда главната форма за навигация към другите форми. В нея съответно всеки клиент може да въвежда различни данни както следва:

* + - * **Създаване на касиери**
      * **Профили и рейтинги**
      * **Операции за работа със пътувания**
      * **Справки**

При избор на едно от тези въвеждания излиза отделна форма за всяко едно от тях, в която се записват съответните данни и се съхраняват в базата данни на системата.

При влизане в системата като **администратор** се зарежда друга главна форма, подобна на тази на клиента, но с тази разлика, че има достъп до някои допълнителни функции, които са недостъпни за него. На практика, администраторът може да използва всички функции на клиента, като има и някои допълнителни екстри, които са:

* + - **Създаване на компания**
    - **Създаване на разпространител**

Системата поддържа 4 известия:

* + - **Новопостъпила заявка за пътуване**
    - **Периодично уведомление за продадени билети от пътуване (в профила на собственика)**
    - **Отменено пътуване (в профила на касиера и разпространителя)**
    - **Наближаващо пътуване с не продадени билети (собственик, разпространител)**

Когато един от двата потребители свърши работата си в системата може да излезе от системата от **,,Exit”**. Така ще се прекратят всички дейности, които е предприемал и ще се затвори приложението.

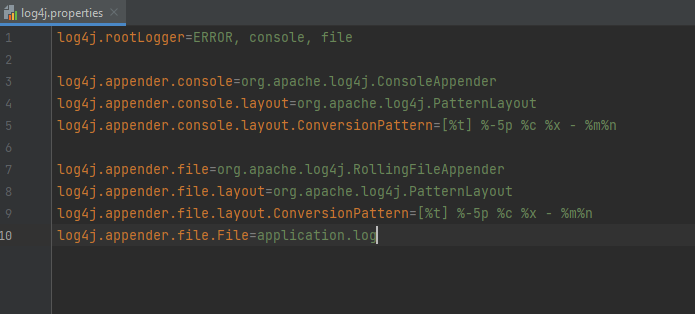
* 1. Реализация на модул за регистриране на събития в системата – log4j.

**Java log4j** (логваща рамка) е комютърен пакет данни, отнасящи се за „логването“ в Java платформата.

Логването служи за следене на активността. То е основен проблем при екипите за разработване. Няколко рамки улесняват и стандартизират процеса на логване в Java платформата. Логърът е отговорен за запазване на съобщенията и предназначението им да бъдат вкарани с точно определена информация и изпратени към логващата рамка.

Пример за log4j:

public void start(Stage primaryStage) throws Exception {  
  
 PropertyConfigurator.*configure*(getClass().getResource(Constants.Configurations.*LOG4J\_PROPERTIES*));  
 URL path = getClass().getResource(Constants.View.*Hey*);  
  
 if (path != null)  
 {  
 Parent root = FXMLLoader.*load*(path);  
 Scene scene = new Scene(root);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
 else  
 {  
 *log*.error("Sorry the main fxml was not found");  
 System.*exit*(-1);  
 }  
  
}



Системата log4j включва няколко файла:

* **log4j.properties**;
* **pom.xml** – в него се слага dependency за log4j;
* **application.log** – файл, в който се добавя информацията за всяко логване.

Log4j използва **PropertyConfigurator**, за да зареди **log4j.properties**. Ние дефинираме собствената си реализация на конфигуратора, която се простира от PropertyConfigurator и пишем код, за да сме сигурни че местоположението на файла на приложението е валидно.

Тестови резултати

* 1. JUnit tests

Компонентното тестване (JUnit test) е процес в програмирането, чрез който се тестват отделни единици (компоненти) от сорс код – един или повече програмни модула и техните контролни данни, процедури за използване, оперативни процедури с цел да се установи дали работят правилно. Под *компонент* се разбира най-малката част на едно програмно приложение, която може да бъде самостоятелно тествана. В т.нар. процедурно програмиране компонент би могъл да бъде целия програмен модул, но в по-честите случаи е отделна функция или процедура. В случая на обектно ориентирано програмиране компонент е най-често цял клас, но в някои случаи може и да е отделен метод. Компонентните тестове се създават от програмист или в някои случаи от white box тестери по време на процеса на разработка, за да се гарантира правилното поведение на софтуера и дали той покрива всички първоначално поставени изисквания.

Целта на компонентното тестване е да се изолира всяка част от програмата и да се покаже, че отделните части работят правилно. Компонентният тест предоставя строго определени правила, които частите от кода трябва да удовлетворяват. От това произтичат серия предимства:

* Ранно откриване на проблема
* Улесняване на промените
* Опресняване
* Добре завършена документация и дизайн.

#### 4.2 Функционални тестове

Функционалното тестване е тестване на „функционалността“ на тестван софтуер или приложение. Той тества поведението на тествания софтуер. Въз основа на изискванията на клиента, документ, наречен софтуерна спецификация или спецификация на изискванията, се използва като ръководство за тестване на приложението.

Въз основа на тях се извайват тестови данни и се подготвят набор от тестови случаи. След това софтуерът се тества в реална среда, за да се провери дали действителният резултат е в синхрон с очаквания резултат. Тази техника се нарича “Техника на черна кутия” и се извършва предимно ръчно и също така е много ефективен при намирането на грешки.

4.3 Тестове на ниво интеграция (Integration testing)

Тестовете на ниво интеграция имат за цел да проверя взаимовръзките и съвместната работа на отделните компоненти, когато са поставени в една среда. Компонентите може да се интегрират на вълни или накуп (известно още като Големият Взрив – „big bang“). Счита се, че варианта на вълни е по-добър, защото позволява новопоявили се дефекти да се открият и отстранят по-бързо преди да се интегрира следващата вълна и дефектите са се мултиплицират.

Тестовете на ниво интеграция изваждат на преден план проблеми при свързването и съвместната работа на интегрираните компоненти (units, модули). Прогресивно нарастващи групи елементи се тестват спрямо архитектурния план на програмата и се сливат, докато целия софтуер не заработи като единна система.