МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**КУРСОВОЙ ПРОЕКТ**

по дисциплине: “Программирование сетевых приложений”

на тему: **“Система «Платежи»”**

Выполнил**:** студент группы 10702121

Алюнина П. А.

Руководитель**:** ст. пр. Кондратёнок Е. В.

Минск 2024

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине: “Программирование сетевых приложений”

на тему: **“Система «Платежи»”**

Выполнил**:** студент группы 10702121

Алюнина П. А.

Руководитель**:** ст. пр. Кондратёнок Е. В.

Минск 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc166754339)

[1. Техническое задание на предмет разработки 5](#_Toc166754340)

[2. Обоснование используемых языков и технологий 6](#_Toc166754341)

[2.1. Язык программирования Java 6](#_Toc166754342)

[2.2. API JDBC 6](#_Toc166754343)

[2.3. MySQL 7](#_Toc166754344)

[3. Моделирование и проектирование программного обеспечения 9](#_Toc166754345)

[3.1. Клиент-сервер 9](#_Toc166754346)

[3.2. MVC 9](#_Toc166754347)

[3.3. Разработка вариантов использования 10](#_Toc166754348)

[3.4. Разработка базы данных 11](#_Toc166754349)

[3.5. Проектирование модульности приложения 12](#_Toc166754350)

[4. Реализация программного обеспечения 13](#_Toc166754351)

[4.1. Структура папки dao 13](#_Toc166754352)

[4.2. Представление приложения 14](#_Toc166754353)

[4.3. Структура контроллера приложения 15](#_Toc166754354)

[5. Развертывание и тестирование программного обеспечения 17](#_Toc166754355)

[Заключение 21](#_Toc166754356)

[Список использованной литературы 22](#_Toc166754357)

[ПРИЛОЖЕНИЕ А 23](#_Toc166754358)

# **ВВЕДЕНИЕ**

С развитием интернет-технологий и электронной коммерции платежные системы стали неотъемлемой частью нашей повседневной жизни. Они обеспечивают безопасные и удобные способы оплаты товаров и услуг, а также позволяют осуществлять денежные переводы в режиме реального времени.

В условиях глобализации и удаленной работы платежные системы становятся все более актуальными.

Целью данного курсового проекта является изучение основ программирования сетевых приложений, связанных с платежными системами. Я рассмотрю различные аспекты, такие как обработка транзакций, клиент-серверная архитектура, связь с системой управления базами данных. Кроме того, я разработаю прототип платежной системы, демонстрирующий принципы ее работы.

В ходе выполнения курсового проекта я надеюсь углубить свои знания в области программирования сетевых приложений и получить практические навыки, которые пригодятся в будущей профессиональной деятельности.

# **Техническое задание на предмет разработки**

*Цель программы:*

Построить веб-систему, поддерживающую заданную функциональность:

— на основе сущностей предметной области создать классы, их описывающие;

— классы и методы должны иметь названия, отражающие их функциональность, и быть грамотно структурированы по пакетам;

— оформление кода должно соответствовать Java Code Convention;

— информацию о предметной области хранить в БД, для доступа использовать API JDBC c использованием пула соединений, разработанного самостоятельно. В качестве СУБД рекомендуется MySQL или Derby;

— приложение должно поддерживать работу с кириллицей (быть многоязычной), в том числе и при хранении информации в БД;

— архитектура приложения должна соответствовать шаблону Model-View-Controller;  
— при реализации алгоритмов бизнес-логики использовать шаблоны GoF: Factory, Method, Command, Builder, Strategy, State, Observer;

— используя сервлеты и JSP, реализовать функции, предложенные в постановке конкретной задачи;

— в страницах JSP применять библиотеку JSTL;

— при разработке бизнес-логики использовать сессии и фильтры;

— использовать Log4j;

— код должен содержать комментарии.

*Детализация*: Система Платежи. Клиент имеет одну или несколько Кредитных Карт, каждая из которых соответствует некоторому Счету в системе платежей. Клиент может при помощи Счета сделать Платеж, заблокировать Счет и пополнить Счет. Администратор снимает блокировку.

*Технологии, используемые в работе*:

1. Объектно-ориентированный язык программирования Java;
2. Система управления базами данных MySQL;
3. Технология для доступа к данным JDBC;
4. Операционная система Windows 11;
5. Среда разработки Intellij IDEA.

# **Обоснование используемых языков и технологий**

## **2.1. Язык программирования Java**

Java — строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems. Разработка ведётся сообществом, организованным через Java Community Process; язык и основные реализующие его технологии распространяются по лицензии GPL. Права на торговую марку принадлежат корпорации Oracle. Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины. Дата официального выпуска — 23 мая 1995 года. Занимает высокие места в рейтингах популярности языков программирования.

Основные преимущества языка Java:

* Простота – первое техническое достоинство Java. У него чёткие синтаксические правила и понятная семантика. Рациональность и краткость очень полезны для обработки кода машинами с ограниченным объёмом ресурсов. Для встроенных устройств создана специальная платформа Java Micro Edition.
* Объектно-ориентированный подход. За 3 десятилетия он доказал свою эффективность. Суть состоит в том, что в центре внимания находятся данные (объекты), интерфейсы и алгоритмы вторичны. Другими словами, мы отталкиваемся от результата при выборе инструментов, способов их применения.
* Безопасность. Важнейший критерий, учитывая применение языка в сетевых / распределенных средах. Разработчики проделали большую работу по защите платформы Java. И она продолжается. Обойти или взломать механизмы защиты крайне сложно. Пример: использование классов, имеющих цифровую подпись. Полные права предоставляются только при полном доверии автору класса.

## **2.2. API JDBC**

API JDBC (Java Database Connectivity) представляет собой стандартный интерфейс Java для взаимодействия с реляционными базами данных. Он позволяет Java-приложениям выполнять SQL-запросы и обрабатывать результаты, обеспечивая универсальный доступ к данным независимо от используемой СУБД.

Этот интерфейс является ключевым компонентом для создания надежных и масштабируемых Java-приложений, работающих с данными.

Особенности:

* Переносимость: Код, написанный с использованием JDBC, легко переносится между различными платформами без изменения кода.
* Безопасность: JDBC поддерживает стандартные механизмы безопасности, включая аутентификацию, шифрование и управление доступом.
* Производительность: Оптимизированные драйверы JDBC и возможность выполнения предкомпилированных SQL-запросов обеспечивают высокую производительность приложений.
* Поддержка SQL и других языков запросов: JDBC не ограничивается только SQL; он также может взаимодействовать с другими языками запросов, поддерживаемыми базой данных.
* Расширяемость: API JDBC позволяет добавлять новые функции и опции через пользовательские расширения.

Эти особенности делают JDBC мощным инструментом для разработки кроссплатформенных и безопасных приложений для работы с базами данных.

## **2.3. MySQL**

MySQL — это одна из систем управления базами данных (Databases Management System) или просто СУБД. Простыми словами, программа, с помощью которой создают и контролируют БД. Она предназначена для работы с базами реляционного типа и использует для взаимодействия с ними язык стандартизированных запросов (SQL).

MySQL существует с 1994 года. Она бесплатная и имеет открытый исходный код, что значит, ее распространение ведется свободно. С недавних пор эта РСУБД стала принадлежать компании Oracle. MySQL является клиент-серверной системой, поэтому вы нередко могли слышать выражение MySQL-сервер. Модель «клиент-сервер» предполагает, что клиент (компьютер пользователя) отправляет запрос на сервер БД, а тот его обрабатывает и возвращает ответ.

С практической точки зрения, система MySQL обладает следующими достоинствами:

* высокая скорость работы благодаря отказу от некоторых стандартов языка;
* легкая установка и понятный даже новичку интерфейс;
* отсутствие ограничений по количеству пользователей;
* гибкость за счет открытого исходного кода;
* большое количество сторонних инструментов (плагинов);
* поддержка до 50 млн строк в таблице;
* хороший уровень безопасности;
* широкий функционал.

Эта СУБД считается надежной и стабильной уже долгие годы. Она поддерживает разные виды таблиц, быстро выполняет команды и может быть модифицирована под индивидуальные нужды проекта.

# **Моделирование и проектирование программного обеспечения**

Для реализации приложения была выбрана клиент-серверная архитектура с использованием MVC.

### **3.1. Клиент-сервер**

«Клиент-сервер» (англ. client–server) – вычислительная или сетевая архитектура, изображенная на рисунке 3.1, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных или в виде сервисных функций.



Рисунок 3.1 – Схема клиент-серверной архитектуры

### **3.2. MVC**

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо:

* модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние;
* представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели;
* контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Схема модели представлена на рисунке 3.2.



Рисунок 3.2 – Схема Model-View-Controller

### **3.3. Разработка вариантов использования**

В системе существует 2 типа пользователей (администратор и клиент). Варианты использования с участием администратора и клиента представлены на рисунках 3.3-3.4.

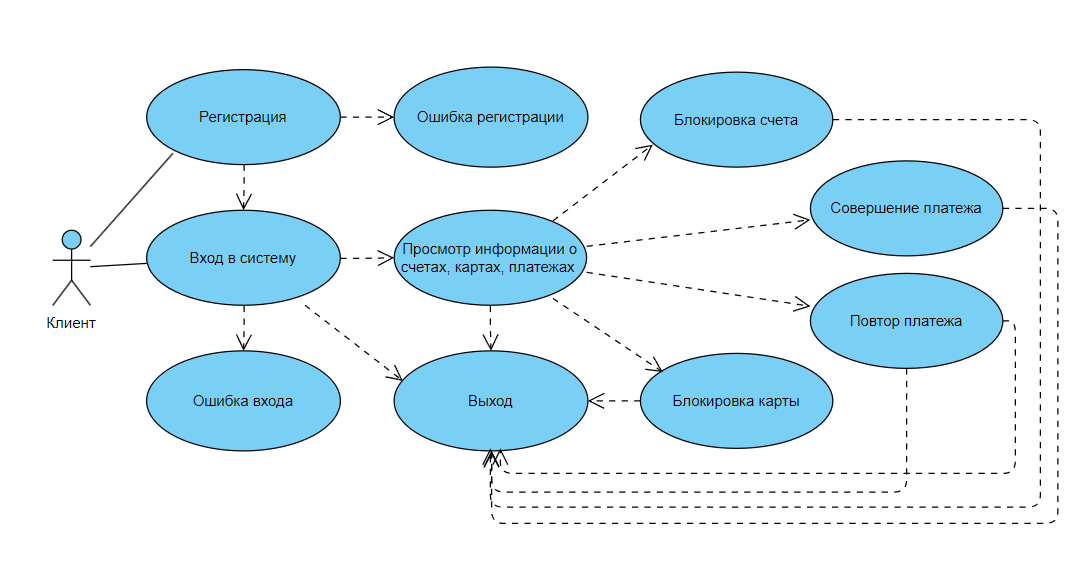


Рисунок 3.3 – Диаграмма вариантов использования (Клиент)

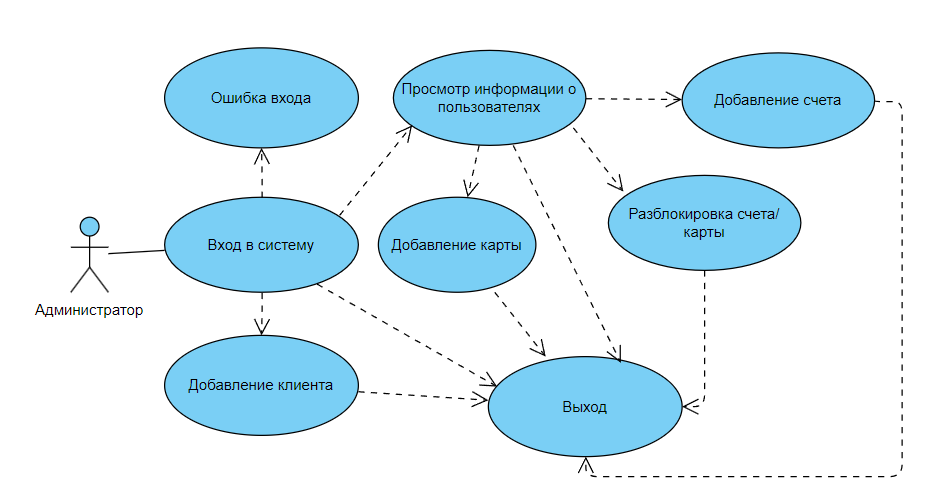


Рисунок 3.4 – Диаграмма вариантов использования (Администратор)

### **3.4. Разработка базы данных**

Для разработки базы данных используется система управления базами данных MySQL. Ниже, на рисунке 3.5, приведена модель базы данных.

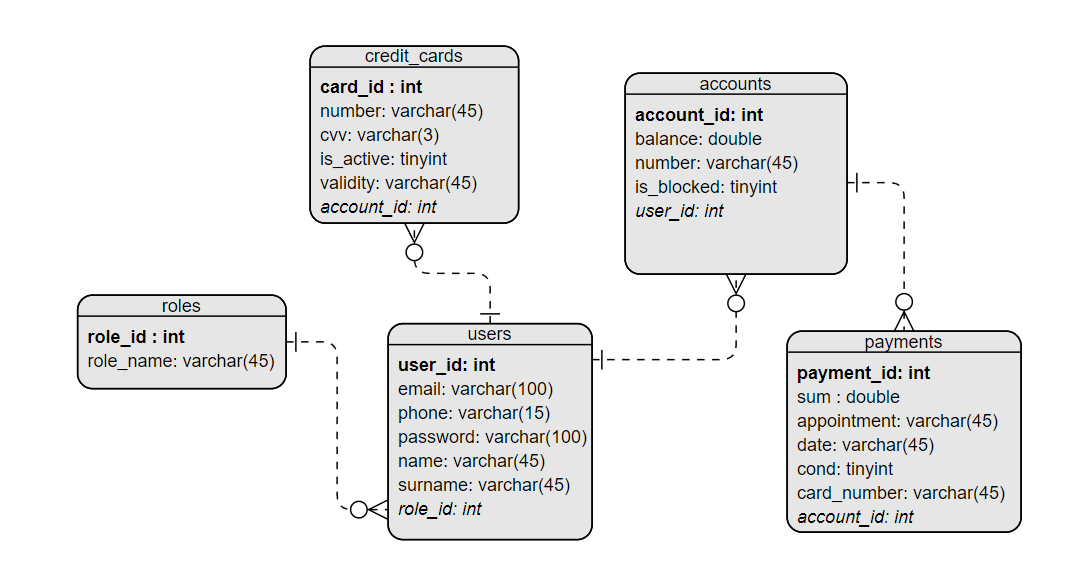


Рисунок 3.5 – ERM-диаграмма базы данных

### **3.5. Проектирование модульности приложения**

Ниже, на рисунке 3.6, приведена диаграмма основных компонентов (модулей) приложения.

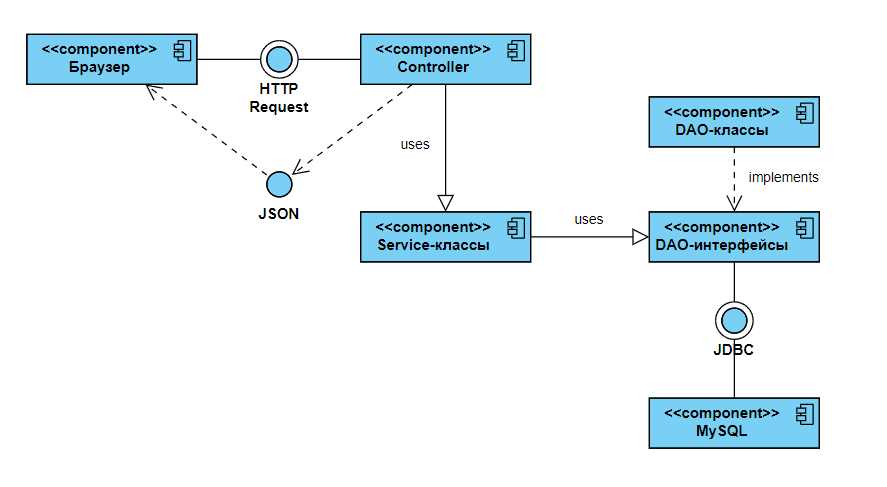


Рисунок 3.6 – Диаграмма основных компонентов приложения

# **Реализация программного обеспечения**

Так как приложение спроектировано по принципу MVC, в процессе реализации выстроилась следующая структура папок и файлов (рисунок 4.1):

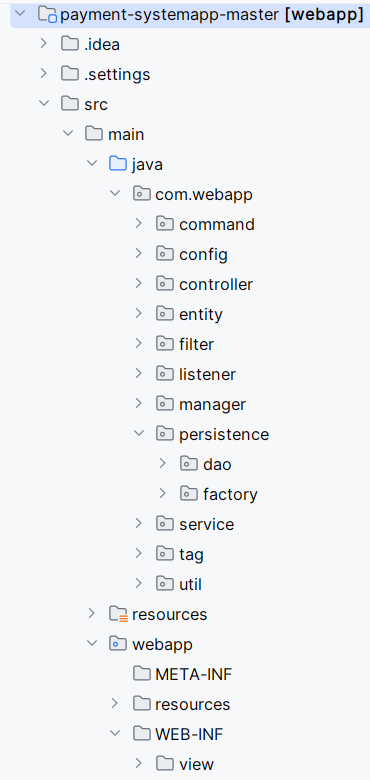


Рисунок 4.1 – Структура приложения

### **4.1. Структура папки dao**

DAO (Data Access Object) обеспечивает доступ к данным, которые хранятся в базе данных, обеспечивает получение и изменение данных следующих видов: счета, карты, платежи, пользователи и др. компоненты. Объекты доступа к данным представлены на рисунке 4.2.

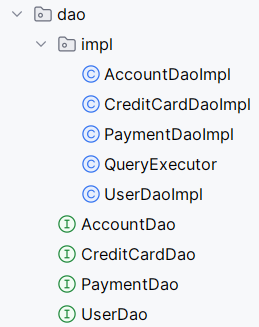


Рисунок 4.2 – Структура папки dao

### **4.2. Представление приложения**

Представление реализуется JSP страницами. Для защищенного доступа к страницам регистрации и авторизации, их представление хранится в папке WEB-INF, что предотвращает доступ к ним путем введения их в адресную строку. Структура представления показана на рисунке 4.3.

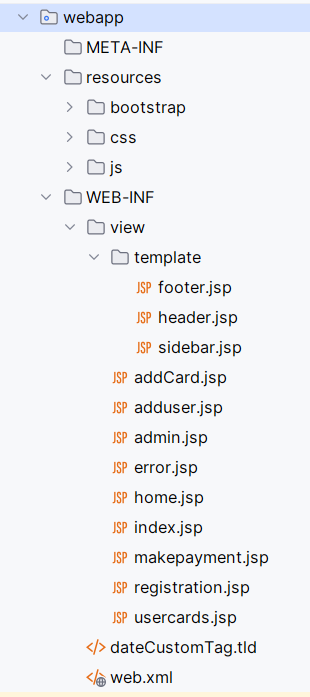


Рисунок 4.3 – Структура представления

### **4.3. Структура контроллера приложения**

Для описания логики работы приложения используются классы Controller и ControllerHelper. Класс ControllerHelper содержит коллекцию (ключ-значение) всех команд приложения, которые запускает класс Controller. Эти команды хранятся в папке command. Данная структура представлена на рисунке 4.4.

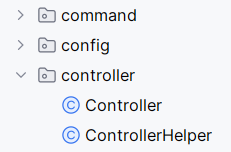


Рисунок 4.4 – Структура контроллера

**4.4. UML-диаграммы классов приложения**

В результате реализации приложения получаем следующую структуру классов, представленную на рисунках 4.5 и 4.6.

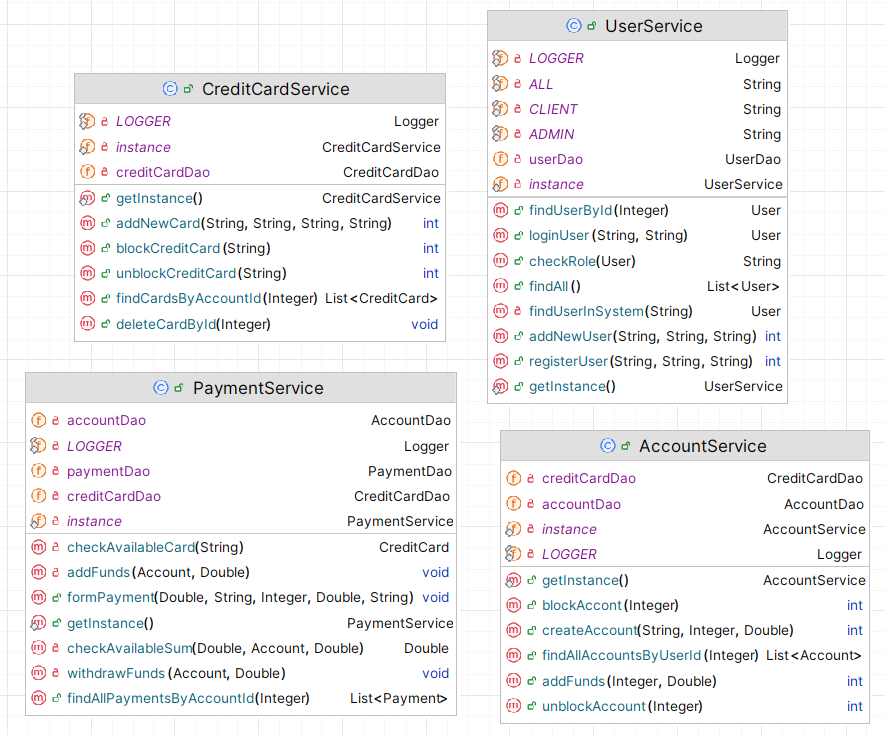


Рисунок 4.5 – UML-диаграмма сервисов

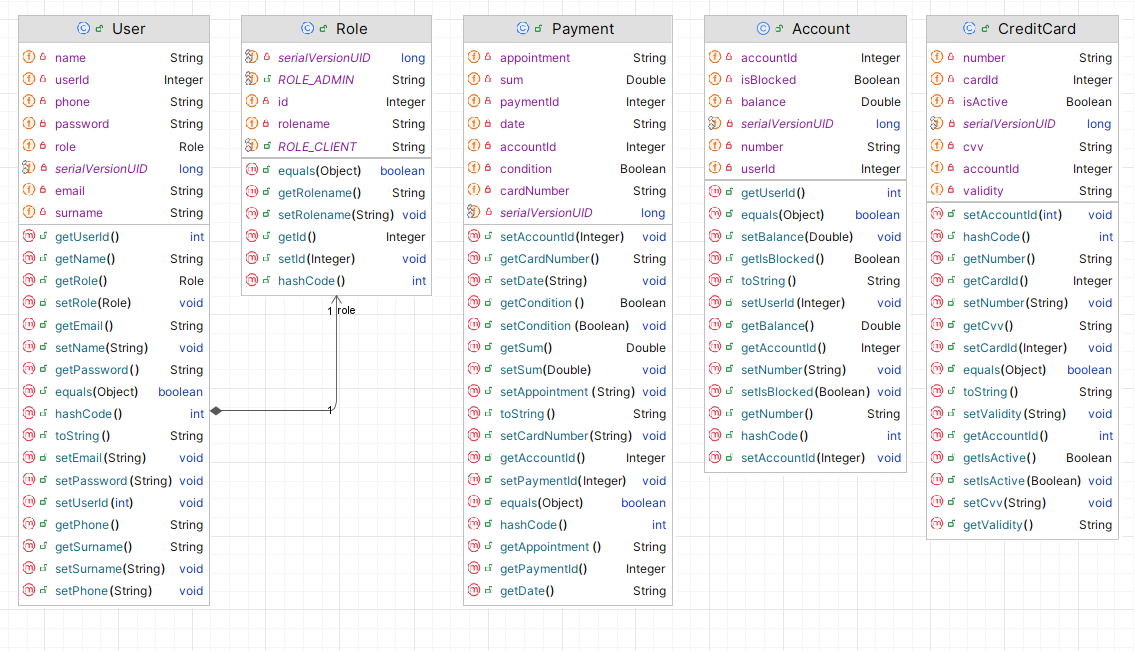


Рисунок 4.6 – UML-диаграмма сущностей

# **Развертывание и тестирование программного обеспечения**

Приложение развернуто на сервере Apache Tomcat и предоставляет возможность администратору и клиенту выполнять определенные функции в зависимости от их роли. При запуске приложения отображается страница входа (рисунок 5.1).

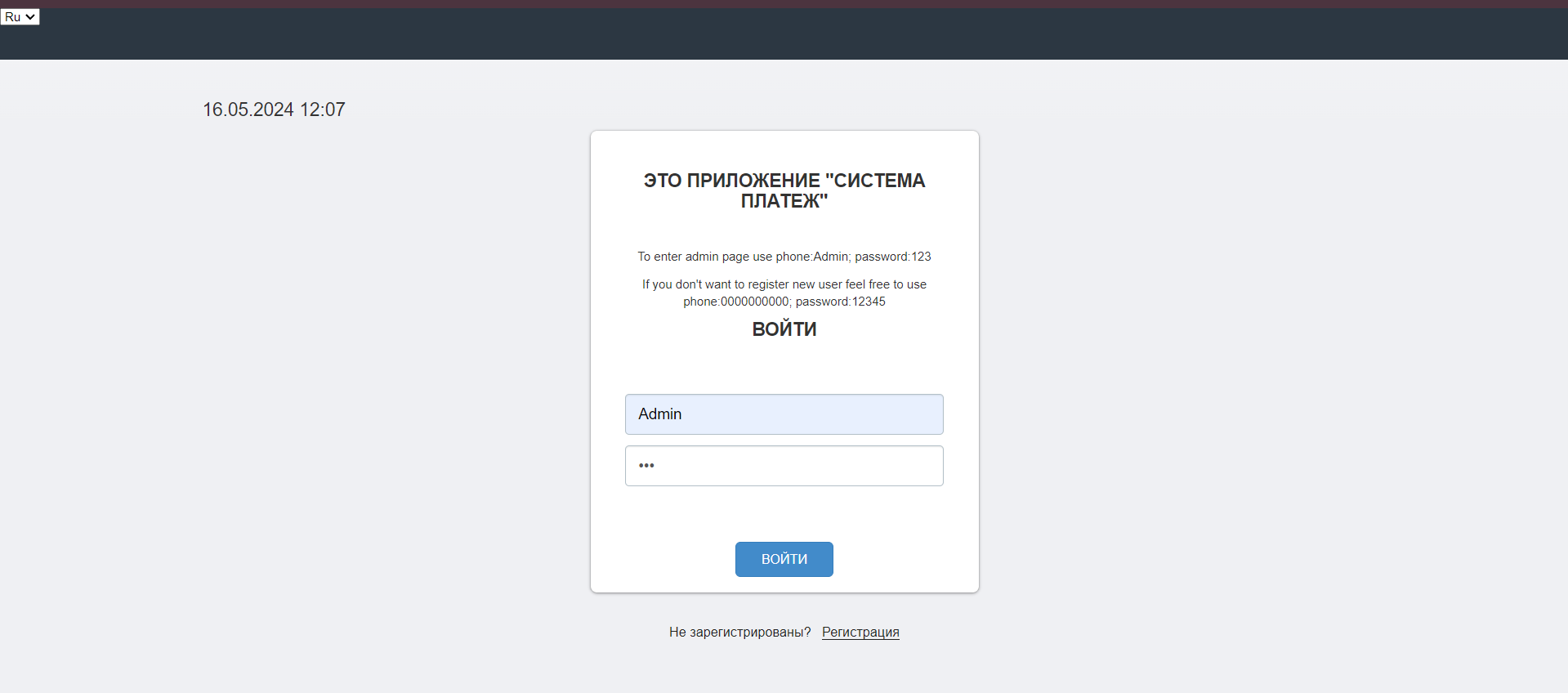


Рисунок 5.1 – Страница входа

При нажатии на кнопку «Войти» пользователь-*администратор* перенаправляется на страницу просмотра клиентов (рисунок 5.2).

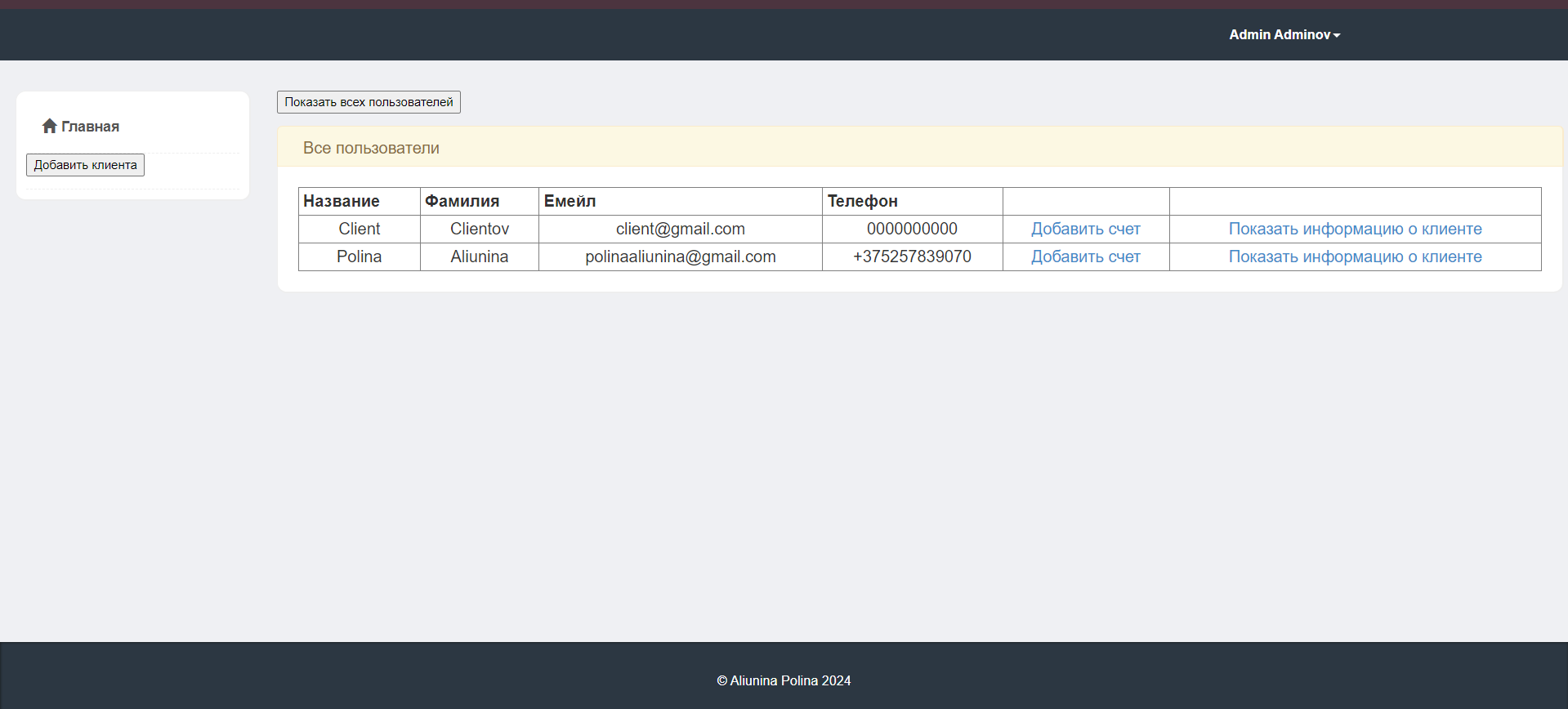


Рисунок 5.2 – Страница просмотра клиентов

Администратор может воспользоваться кнопкой «Добавить клиента» (рисунок 5.3).

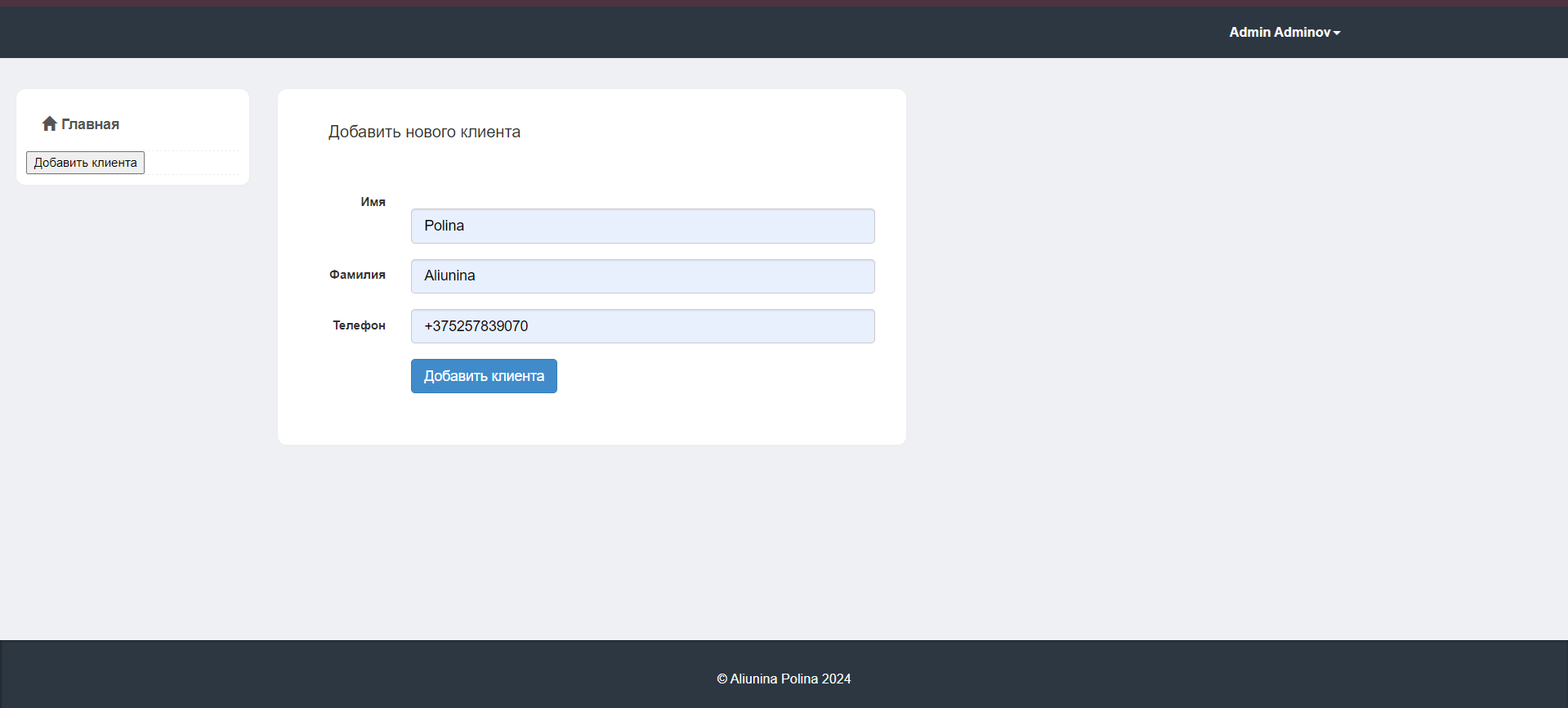


Рисунок 5.3 – Страница добавления клиента

Далее администратор может добавлять и изменять счета клиентов, а также добавлять карты клиентам (рисунки 5.4 и 5.5).

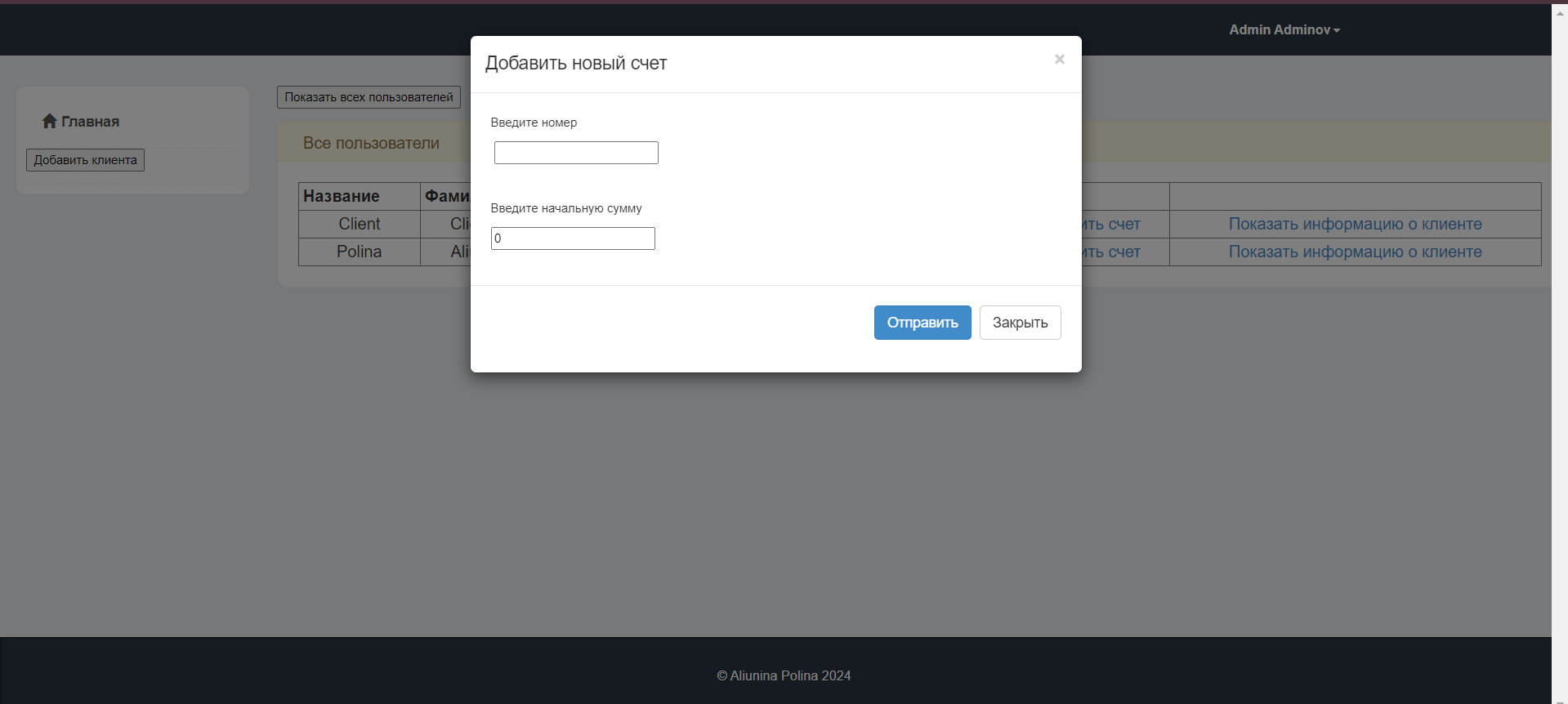


Рисунок 5.4 – Добавление счета клиенту

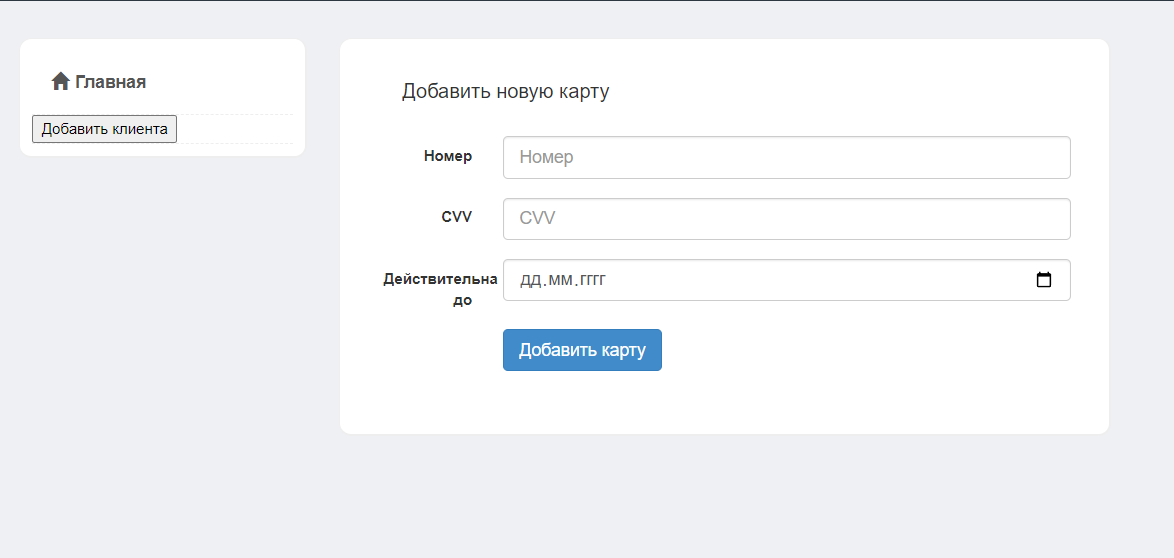


Рисунок 5.5 – Добавление карты клиенту

При нажатии на кнопку «Войти» пользователь-*клиент*перенаправляется на страницу просмотра своих счетов, карт и платежей (рисунок 5.6).

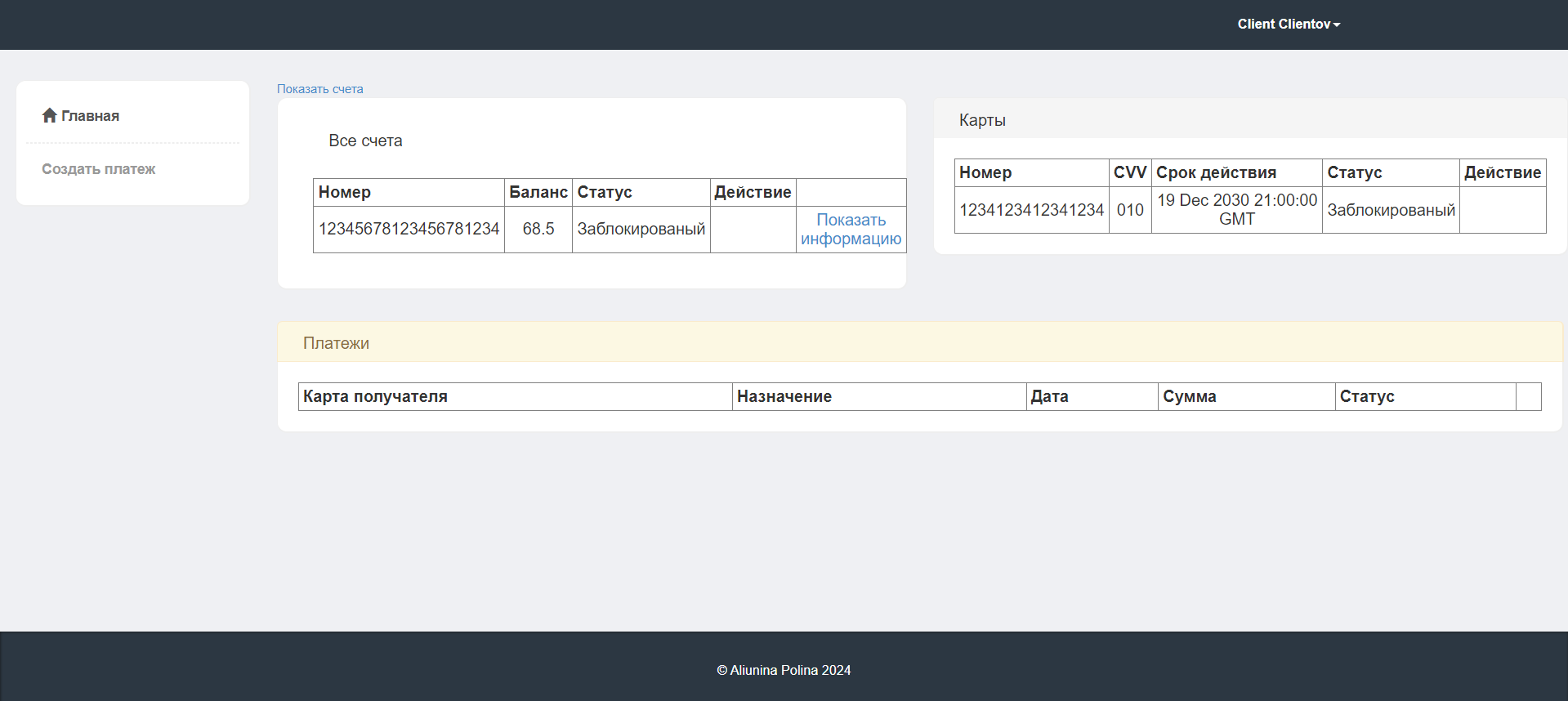


Рисунок 5.6 – Просмотр информации о счетах, картах, платежах

Также клиент может блокировать свои счета и карты (рисунок 5.7).

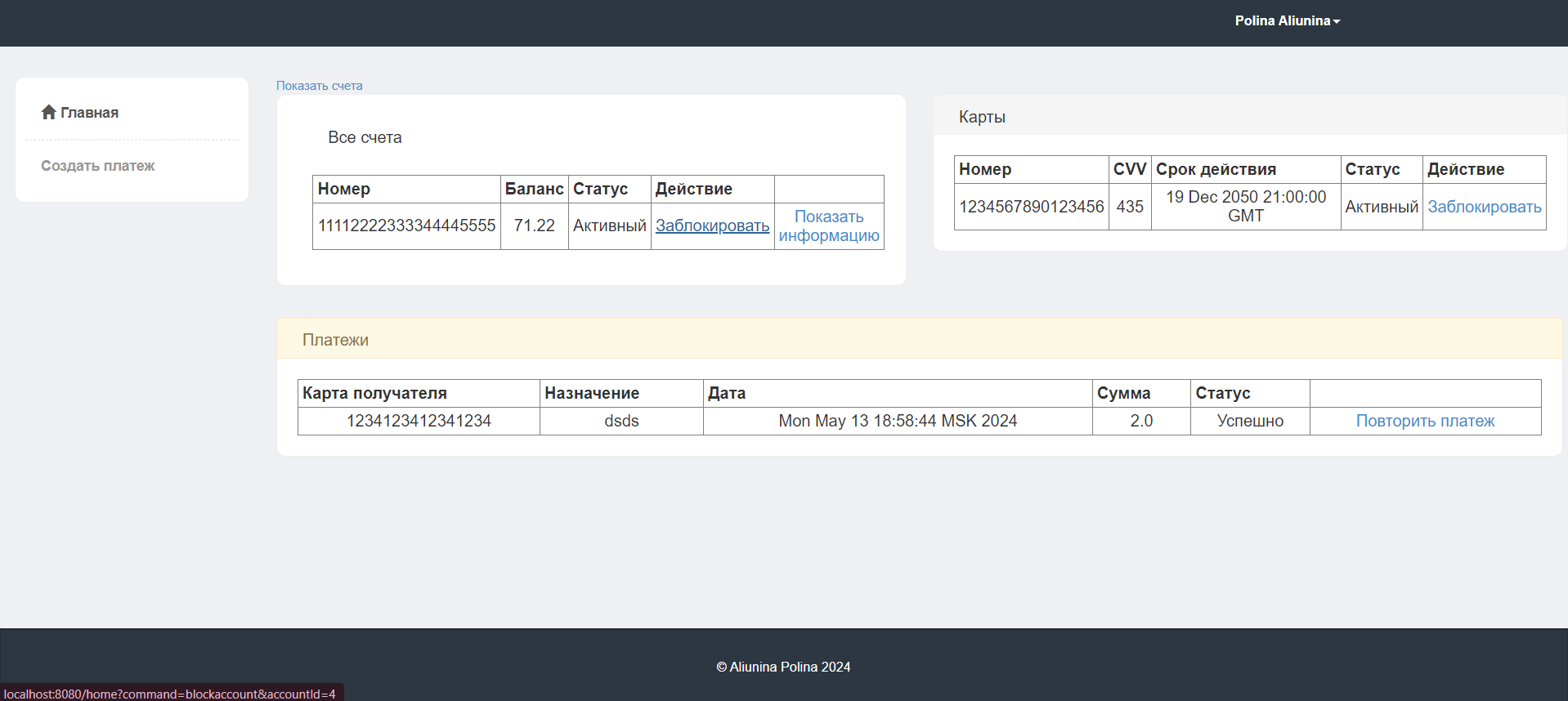


Рисунок 5.7 – Блокировка карты

При нажатии на кнопку «Создать платеж», клиент может совершить платеж (рисунок 5.8). Кроме того, пользователю доступна возможность «Повторить платеж».

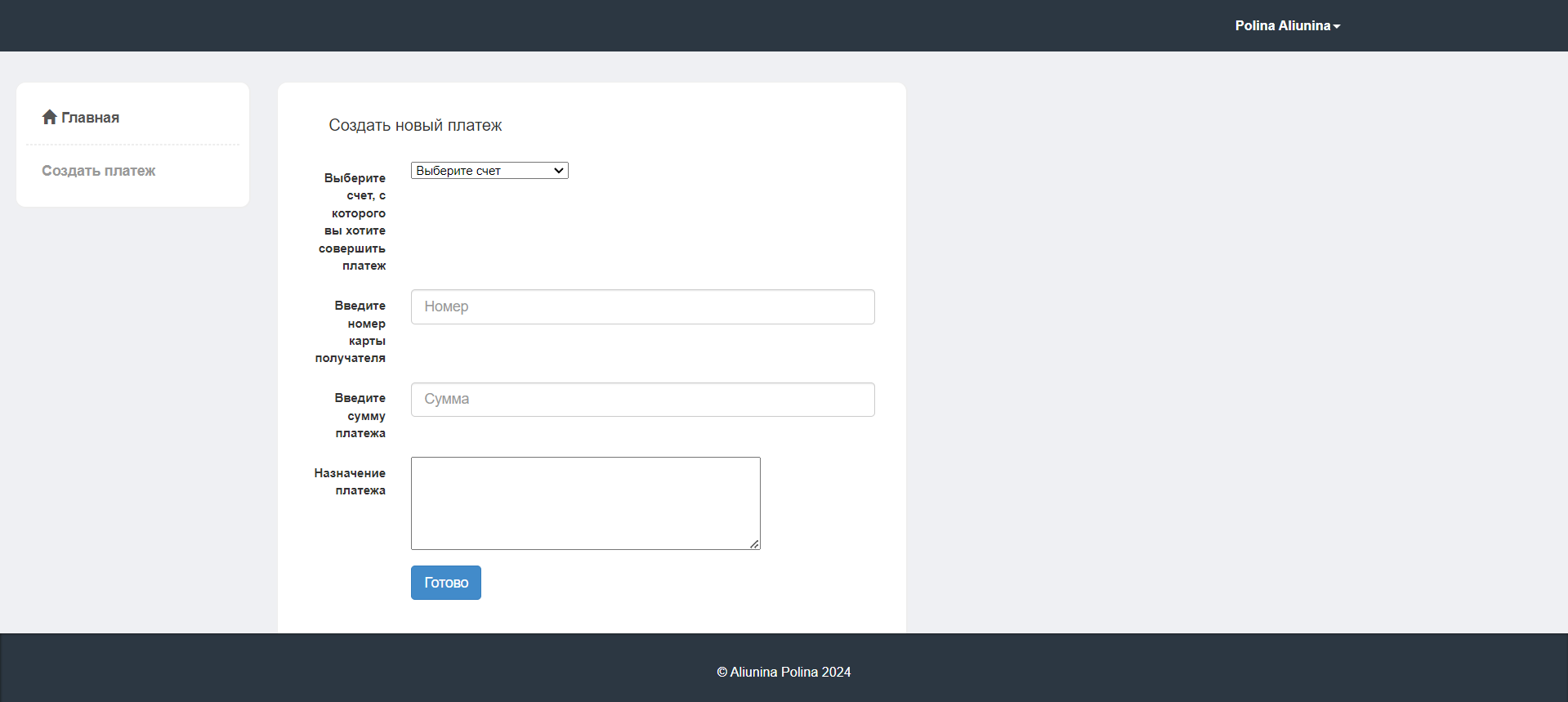


Рисунок 5.8 – Создание платежа

# **Заключение**

Банковские приложения играют важную роль в современном мире, так как позволяют клиентам осуществлять финансовые операции в любое время и в любом месте, без необходимости посещения физического отделения банка. Это делает процесс управления финансами более удобным и эффективным. Банковские приложения предлагают широкий спектр функций, таких как переводы, платежи, управление счетами и кредитными картами, уведомления о транзакциях и многое другое. Это делает возможным эффективное управление финансами и контроль над своими средствами.

В результате выполнения данного курсового проекта было разработано сетевое приложение на языке программирования Java, реализующее функционал банковской системы платежей.

В ходе работы была успешно спроектирована и реализована бизнес-логика, позволяющая клиенту управлять своими кредитными картами и счетами, осуществлять платежи, блокировать счета. Администратор имеет возможность управлять пользователями, их блокировками.

Этот проект демонстрирует понимание принципов работы сетевых приложений и навыки программирования на Java в контексте реального бизнес-процесса.

# **Список использованной литературы**

1. Давыдов, С. IntelliJ IDEA. Профессиональное программирование на Java / С. Давыдов. - СПб.: BHV, 2005. - 800 c.
2. Машнин, Т.С. Web-сервисы Java. Профессиональное программирование / Т.С. Машнин. - СПб.: BHV, 2012. - 560 c.
3. Дашнер, С. Изучаем Java EE. Современное программирование для больших предприятий / С. Дашнер. - СПб.: Питер, 2015. - 94 c.
4. Log4j – Log4j 2 Web Applications - руководство от Apache Software Foundation - <https://logging.apache.org/log4j/2.x/manual/webapp.html>
5. Блох, Д. Java Эффективное программирование / Д. Блох. - М.: Лори, 2016. - 440 c.

# **ПРИЛОЖЕНИЕ А**

# **Листинг программного кода**

Пакет Entity

package com.webapp.entity;  
  
import java.io.Serializable;  
  
*/\*\*  
 \* Account  
 \*/*public class Account implements Serializable {  
  
 private static final long *serialVersionUID* = 1L;  
 private Integer accountId;  
 private String number;  
 private Double balance;  
 private Integer userId;  
 private Boolean isBlocked;  
  
 public Account() {  
 }  
  
 public Integer getAccountId() {  
 return accountId;  
 }  
  
 public void setAccountId(Integer accountId) {  
 this.accountId = accountId;  
 }  
  
 public String getNumber() {  
 return number;  
 }  
  
 public void setNumber(String number) {  
 this.number = number;  
 }  
  
 public Double getBalance() {  
 return balance;  
 }  
  
 public void setBalance(Double balance) {  
 this.balance = Math.*round*(balance \* 100.0) / 100.0;  
 }  
  
 public int getUserId() {  
 return userId;  
 }  
  
 public void setUserId(Integer userId) {  
 this.userId = userId;  
 }  
   
  
 public Boolean getIsBlocked() {  
 return isBlocked;  
 }  
  
 public void setIsBlocked(Boolean isBlocked) {  
 this.isBlocked = isBlocked;  
 }  
  
 @Override  
 public int hashCode() {  
 final int prime = 31;  
 int result = 1;  
 result = prime \* result + ((accountId == null) ? 0 : accountId.hashCode());  
 result = prime \* result + ((balance == null) ? 0 : balance.hashCode());  
 result = prime \* result + ((isBlocked == null) ? 0 : isBlocked.hashCode());  
 result = prime \* result + ((number == null) ? 0 : number.hashCode());  
 result = prime \* result + ((userId == null) ? 0 : userId.hashCode());  
 return result;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj)  
 return true;  
 if (obj == null)  
 return false;  
 if (getClass() != obj.getClass())  
 return false;  
 Account other = (Account) obj;  
 if (accountId == null) {  
 if (other.accountId != null)  
 return false;  
 } else if (!accountId.equals(other.accountId))  
 return false;  
 if (balance == null) {  
 if (other.balance != null)  
 return false;  
 } else if (!balance.equals(other.balance))  
 return false;  
 if (isBlocked == null) {  
 if (other.isBlocked != null)  
 return false;  
 } else if (!isBlocked.equals(other.isBlocked))  
 return false;  
 if (number == null) {  
 if (other.number != null)  
 return false;  
 } else if (!number.equals(other.number))  
 return false;  
 if (userId == null) {  
 if (other.userId != null)  
 return false;  
 } else if (!userId.equals(other.userId))  
 return false;  
 return true;  
 }  
  
 @Override  
 public String toString() {  
 return "Account [accountId=" + accountId + ", number=" + number + ", balance=" + balance + ", userId=" + userId  
 + ", isBlocked=" + isBlocked + "]";  
 }   
}

package com.webapp.entity;  
  
import java.io.Serializable;  
  
*/\*\*  
 \* CreditCard  
 \*/*public class CreditCard implements Serializable {  
  
 private static final long serialVersionUID = 1L;  
 private Integer cardId;  
 private String number;  
 private String cvv;  
 private String validity;  
 private Boolean isActive;  
 private Integer accountId;  
  
 public CreditCard() {  
 }  
  
 public Integer getCardId() {  
 return cardId;  
 }  
  
 public void setCardId(Integer cardId) {  
 this.cardId = cardId;  
 }  
  
 public String getNumber() {  
 return number;  
 }  
  
 public void setNumber(String number) {  
 this.number = number;  
 }  
  
 public String getCvv() {  
 return cvv;  
 }  
  
 public void setCvv(String cvv) {  
 this.cvv = cvv;  
 }  
  
 public String getValidity() {  
 return validity;  
 }  
  
 public void setValidity(String validity) {  
 this.validity = validity;  
 }  
  
 public Boolean getIsActive() {  
 return isActive;  
 }  
  
 public void setIsActive(Boolean isActive) {  
 this.isActive = isActive;  
 }  
  
 public int getAccountId() {  
 return accountId;  
 }  
  
 public void setAccountId(int accountId) {  
 this.accountId = accountId;  
 }  
  
 @Override  
 public int hashCode() {  
 final int prime = 31;  
 int result = 1;  
 result = prime \* result + ((accountId == null) ? 0 : accountId.hashCode());  
 result = prime \* result + ((cardId == null) ? 0 : cardId.hashCode());  
 result = prime \* result + ((cvv == null) ? 0 : cvv.hashCode());  
 result = prime \* result + ((isActive == null) ? 0 : isActive.hashCode());  
 result = prime \* result + ((number == null) ? 0 : number.hashCode());  
 result = prime \* result + ((validity == null) ? 0 : validity.hashCode());  
 return result;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj)  
 return true;  
 if (obj == null)  
 return false;  
 if (getClass() != obj.getClass())  
 return false;  
 CreditCard other = (CreditCard) obj;  
 if (accountId == null) {  
 if (other.accountId != null)  
 return false;  
 } else if (!accountId.equals(other.accountId))  
 return false;  
 if (cardId == null) {  
 if (other.cardId != null)  
 return false;  
 } else if (!cardId.equals(other.cardId))  
 return false;  
 if (cvv == null) {  
 if (other.cvv != null)  
 return false;  
 } else if (!cvv.equals(other.cvv))  
 return false;  
 if (isActive == null) {  
 if (other.isActive != null)  
 return false;  
 } else if (!isActive.equals(other.isActive))  
 return false;  
 if (number == null) {  
 if (other.number != null)  
 return false;  
 } else if (!number.equals(other.number))  
 return false;  
 if (validity == null) {  
 if (other.validity != null)  
 return false;  
 } else if (!validity.equals(other.validity))  
 return false;  
 return true;  
 }  
  
 @Override  
 public String toString() {  
 return "CreditCard [cardId=" + cardId + ", number=" + number + ", cvv=" + cvv + ", validity=" + validity  
 + ", isActive=" + isActive + ", accountId=" + accountId + "]";  
 }  
}

package com.webapp.entity;  
  
import java.io.Serializable;  
import java.util.Date;  
  
*/\*\*  
 \* Payment  
 \*/*public class Payment implements Serializable {  
  
 private static final long serialVersionUID = 1L;  
 private Integer paymentId;  
 private Double sum;  
 private String appointment;  
 private String date;  
 private Boolean condition;  
 private Integer accountId;  
 private String cardNumber;  
  
 public Payment() {  
 }  
  
 public Integer getPaymentId() {  
 return paymentId;  
 }  
  
 public void setPaymentId(Integer paymentId) {  
 this.paymentId = paymentId;  
 }  
  
 public Double getSum() {  
 return sum;  
 }  
  
 public void setSum(Double sum) {  
 this.sum = sum;  
 }  
  
 public String getAppointment() {  
 return appointment;  
 }  
  
 public void setAppointment(String appointment) {  
 this.appointment = appointment;  
 }  
  
 public String getDate() {  
 return date;  
 }  
  
 public void setDate(String date) {  
 this.date = date;  
 }  
  
 public Boolean getCondition() {  
 return condition;  
 }  
  
 public void setCondition(Boolean condition) {  
 this.condition = condition;  
 }  
  
 public Integer getAccountId() {  
 return accountId;  
 }  
  
 public void setAccountId(Integer accountId) {  
 this.accountId = accountId;  
 }  
  
 public String getCardNumber() {  
 return cardNumber;  
 }  
  
 public void setCardNumber(String cardNumber) {  
 this.cardNumber = cardNumber;  
 }  
  
 @Override  
 public int hashCode() {  
 final int prime = 31;  
 int result = 1;  
 result = prime \* result + ((accountId == null) ? 0 : accountId.hashCode());  
 result = prime \* result + ((appointment == null) ? 0 : appointment.hashCode());  
 result = prime \* result + ((cardNumber == null) ? 0 : cardNumber.hashCode());  
 result = prime \* result + ((condition == null) ? 0 : condition.hashCode());  
 result = prime \* result + ((date == null) ? 0 : date.hashCode());  
 result = prime \* result + ((paymentId == null) ? 0 : paymentId.hashCode());  
 result = prime \* result + ((sum == null) ? 0 : sum.hashCode());  
 return result;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj)  
 return true;  
 if (obj == null)  
 return false;  
 if (getClass() != obj.getClass())  
 return false;  
 Payment other = (Payment) obj;  
 if (accountId == null) {  
 if (other.accountId != null)  
 return false;  
 } else if (!accountId.equals(other.accountId))  
 return false;  
 if (appointment == null) {  
 if (other.appointment != null)  
 return false;  
 } else if (!appointment.equals(other.appointment))  
 return false;  
 if (cardNumber == null) {  
 if (other.cardNumber != null)  
 return false;  
 } else if (!cardNumber.equals(other.cardNumber))  
 return false;  
 if (condition == null) {  
 if (other.condition != null)  
 return false;  
 } else if (!condition.equals(other.condition))  
 return false;  
 if (date == null) {  
 if (other.date != null)  
 return false;  
 } else if (!date.equals(other.date))  
 return false;  
 if (paymentId == null) {  
 if (other.paymentId != null)  
 return false;  
 } else if (!paymentId.equals(other.paymentId))  
 return false;  
 if (sum == null) {  
 if (other.sum != null)  
 return false;  
 } else if (!sum.equals(other.sum))  
 return false;  
 return true;  
 }  
  
 @Override  
 public String toString() {  
 return "Payment [paymentId=" + paymentId + ", sum=" + sum + ", appointment=" + appointment + ", date=" + date  
 + ", condition=" + condition + ", accountId=" + accountId + ", cardNumber=" + cardNumber + "]";  
 }  
}

package com.webapp.entity;  
  
import java.io.Serializable;  
  
*/\*\*  
 \* User role  
 \*/*public class Role implements Serializable {  
  
 private static final long *serialVersionUID* = 4583195843033194616L;  
   
 public static final String *ROLE\_ADMIN* = "admin";  
 public static final String ROLE\_CLIENT = "client";  
  
 private Integer id;  
 private String rolename;  
   
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public String getRolename() {  
 return rolename;  
 }  
  
 public void setRolename(String rolename) {  
 this.rolename = rolename;  
 }  
  
 @Override  
 public int hashCode() {  
 final int prime = 31;  
 int result = 1;  
 result = prime \* result + ((id == null) ? 0 : id.hashCode());  
 result = prime \* result + ((rolename == null) ? 0 : rolename.hashCode());  
 return result;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj)  
 return true;  
 if (obj == null)  
 return false;  
 if (getClass() != obj.getClass())  
 return false;  
 Role other = (Role) obj;  
 if (id == null) {  
 if (other.id != null)  
 return false;  
 } else if (!id.equals(other.id))  
 return false;  
 if (rolename == null) {  
 if (other.rolename != null)  
 return false;  
 } else if (!rolename.equals(other.rolename))  
 return false;  
 return true;  
 }   
}

package com.webapp.entity;  
  
import java.io.Serializable;  
  
*/\*\*  
 \* User  
 \*/*public class User implements Serializable {  
  
 private static final long serialVersionUID = 1L;  
 private Integer userId;  
 private String phone;  
 private String password;  
 private String email;  
 private String name;  
 private String surname;  
 private Role role;  
  
 public User() {  
 }  
  
 public int getUserId() {  
 return userId;  
 }  
  
 public void setUserId(int userId) {  
 this.userId = userId;  
 }  
  
 public String getPhone() {  
 return phone;  
 }  
  
 public Role getRole() {  
 return role;  
 }  
  
 public void setRole(Role role) {  
 this.role = role;  
 }  
  
 public void setPhone(String phone) {  
 this.phone = phone;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 @Override  
 public int hashCode() {  
 final int prime = 31;  
 int result = 1;  
 result = prime \* result + ((email == null) ? 0 : email.hashCode());  
 result = prime \* result + ((phone == null) ? 0 : phone.hashCode());  
 result = prime \* result + ((name == null) ? 0 : name.hashCode());  
 result = prime \* result + ((password == null) ? 0 : password.hashCode());  
 result = prime \* result + ((surname == null) ? 0 : surname.hashCode());  
 result = prime \* result + userId;  
 return result;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj)  
 return true;  
 if (obj == null)  
 return false;  
 if (getClass() != obj.getClass())  
 return false;  
 User other = (User) obj;  
 if (email == null) {  
 if (other.email != null)  
 return false;  
 } else if (!email.equals(other.email))  
 return false;  
 if (phone == null) {  
 if (other.phone != null)  
 return false;  
 } else if (!phone.equals(other.phone))  
 return false;  
 if (name == null) {  
 if (other.name != null)  
 return false;  
 } else if (!name.equals(other.name))  
 return false;  
 if (password == null) {  
 if (other.password != null)  
 return false;  
 } else if (!password.equals(other.password))  
 return false;  
 if (surname == null) {  
 if (other.surname != null)  
 return false;  
 } else if (!surname.equals(other.surname))  
 return false;  
 if (userId != other.userId)  
 return false;  
 return true;  
 }  
  
 @Override  
 public String toString() {  
 return "User [userId=" + userId + ", phone=" + phone + ", password=" + password + ", email=" + email + ", name="  
 + name + ", surname=" + surname + "]";  
 }  
}

Класс Controller

package com.webapp.controller;  
  
import java.io.IOException;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
import org.apache.log4j.Logger;  
  
import com.webapp.command.ICommand;  
  
*/\*\*  
 \* Servlet implementation class Controller  
 \*/*public class Controller extends HttpServlet {  
 private static final long *serialVersionUID* = 1L;  
 */\*\*  
 \* LOG for this class.  
 \*/* private static final Logger *LOGGER* = Logger.*getLogger*(Controller.class);  
  
 */\*\*  
 \* Main method of this controller.  
 \*   
 \* @param request  
 \* request.  
 \* @param response  
 \* response.  
 \* @throws ServletException  
 \* @throws IOException  
 \*/* private void process(final HttpServletRequest request, final HttpServletResponse response)  
 throws IOException, ServletException {  
 ICommand command = ControllerHelper.*getInstance*().getCommand(request);  
 String path = command.execute(request, response);  
 request.getRequestDispatcher(path).forward(request, response);  
 }  
  
 */\*\*  
 \* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse  
 \* response)  
 \*/* protected void doGet(final HttpServletRequest request, final HttpServletResponse response)  
 throws ServletException, IOException {  
 *LOGGER*.info("===================GET works");  
 process(request, response);  
 }  
  
 */\*\*  
 \* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse  
 \* response)  
 \*/* protected void doPost(final HttpServletRequest request, final HttpServletResponse response)  
 throws ServletException, IOException {  
 *LOGGER*.info("===================POST works");  
 process(request, response);  
 }  
}

Класс MySqlDaoFactory

package com.webapp.persistence.factory;  
  
import com.webapp.persistence.dao.AccountDao;  
import com.webapp.persistence.dao.CreditCardDao;  
import com.webapp.persistence.dao.PaymentDao;  
import com.webapp.persistence.dao.UserDao;  
import com.webapp.persistence.dao.impl.AccountDaoImpl;  
import com.webapp.persistence.dao.impl.CreditCardDaoImpl;  
import com.webapp.persistence.dao.impl.PaymentDaoImpl;  
import com.webapp.persistence.dao.impl.UserDaoImpl;  
  
*/\*\*  
 \* MySqlDaoFactory returns new instances of data access objects  
 \*  
 \*/*public class MySqlDaoFactory {  
  
 */\*\*  
 \* @return UserDao  
 \*/* public static UserDao createUserDao() {  
 return UserDaoImpl.*getInstance*();  
 }  
  
 */\*\*  
 \* @return PaymentDao  
 \*/* public static PaymentDao createPaymentDao() {  
 return PaymentDaoImpl.*getInstance*();  
 }  
  
 */\*\*  
 \* @return CreditCardDao  
 \*/* public static CreditCardDao createCreditCardDao() {  
 return CreditCardDaoImpl.*getInstance*();  
 }  
  
 */\*\*  
 \* @return AccountDao  
 \*/* public static AccountDao createAccountDao() {  
 return AccountDaoImpl.*getInstance*();  
 }  
}

home.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8"  
 pageEncoding="UTF-8"%>  
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>  
<c:set var="language"  
 value="${not empty param.language ? param.language : not empty language ? language : pageContext.request.locale}"  
 scope="session" />  
<fmt:setLocale value="${language}" />  
<fmt:setBundle basename="message" />  
<html>  
<head>  
<title><fmt:message key="home.title"></fmt:message></title>  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
*<!-- Bootstrap -->*<link href="resources/bootstrap/css/bootstrap.min.css" rel="stylesheet">  
*<!-- styles -->*<link href="resources/css/styles.css" rel="stylesheet">  
<script src="https://code.jquery.com/jquery.js"></script>  
*<!-- HTML5 Shim and Respond.js IE8 support of HTML5 elements and media queries -->  
<!-- WARNING: Respond.js doesn't work if you view the page via file:// -->  
<!--[if lt IE 9]>  
 <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>  
 <script src="https://oss.maxcdn.com/libs/respond.js/1.3.0/respond.min.js"></script>  
 <![endif]-->*</head>  
<body>  
 <jsp:include page="template/header.jsp" />  
  
 <div class="page-content">  
 <div class="row">  
 <div class="col-md-2">  
 <jsp:include page="template/sidebar.jsp" />  
 </div>  
 <div class="col-md-10">  
 <div class="row">  
 <div class="col-md-6">  
 <a href="?command=showAccounts"><fmt:message key="home.showAccounts"></fmt:message></a>  
 <c:if test="${showAccouts}">  
 <div class="content-box-large">  
 <div class="panel-heading">  
 <div class="panel-title">  
 <fmt:message key="home.allaccounts"></fmt:message>  
 </div>  
  
 </div>  
 <fmt:message key="home.account.number" var="number"></fmt:message>  
 <fmt:message key="home.account.status" var="status"></fmt:message>  
 <fmt:message key="home.account.status.active" var="statusActive"></fmt:message>  
 <fmt:message key="home.account.status.blocked"  
 var="statusBlocked"></fmt:message>  
 <fmt:message key="home.account.button.block" var="block"></fmt:message>  
 <fmt:message key="home.account.button.showInfo" var="showInfo"></fmt:message>  
  
 <div class="panel-body">  
 <table border="1" width="100%" cellpadding="4" cellpacing="3">  
 <th>${number}</th>  
 <th><fmt:message key="home.account.balance"></fmt:message></th>  
 <th>${status}</th>  
 <th><fmt:message key="userCards.action"></fmt:message></th>  
 <th></th>  
   
 <c:forEach items="${accounts}" var="account">  
 <tr align="center">  
 <td>${account.number}</td>  
 <td>${account.balance}</td>  
 <td><c:if test="${!account.isBlocked}">${statusActive }</c:if>  
 <c:if test="${account.isBlocked}">${statusBlocked }</c:if></td>  
 <td><c:if test="${!account.isBlocked}">  
 <a  
 href="?command=blockaccount&accountId=${account.accountId}">${block }</a>  
 </c:if></td>  
 <td><a  
 href="?command=showinfo&accountId=${account.accountId}">${showInfo }</a></td>  
   
 </tr>  
 </c:forEach>  
 </table>  
 </div>  
 </div>  
 </c:if>  
 </div>  
  
 <br/>  
  
 <div class="col-md-6">  
 <div class="row">  
 <div class="col-md-12">  
 <c:if test="${showAccouts}">  
 <div class="content-box-header">  
 <div class="panel-title">  
 <fmt:message key="home.cards"></fmt:message>  
 </div>  
 </div>  
  
 <div class="content-box-large box-with-header">  
 <table border="1" width="100%" cellpadding="4" cellpacing="3">  
 <th>${number}</th>  
 <th><fmt:message key="home.card.cvv"></fmt:message></th>  
 <th><fmt:message key="home.card.date"></fmt:message></th>  
 <th>${status }</th>  
 <th><fmt:message key="userCards.action"></fmt:message></th>  
   
 <c:forEach items="${cards}" var="card">  
 <tr align="center">  
 <td>${card.number}</td>  
 <td>${card.cvv}</td>  
 <td>${card.validity}</td>  
 <td><c:if test="${card.isActive}">${statusActive}</c:if>  
 <c:if test="${!card.isActive}">${statusBlocked }</c:if></td>  
 <td><c:if test="${card.isActive}">  
 <a href="?command=blockCard&cardNumber=${card.number}">${block }</a>  
 </c:if></td>  
   
 </tr>  
 </c:forEach>  
 </table>  
 </div>  
 </c:if>  
 </div>  
 </div>  
 </div>  
 </div>  
  
 <div class="row">  
 <div class="col-md-12 panel-warning">  
 <c:if test="${showAccouts}">  
 <div class="content-box-header panel-heading">  
 <div class="panel-title ">  
 <fmt:message key="home.payments"></fmt:message>  
 </div>  
 </div>  
 <div class="content-box-large box-with-header">  
 <table border="1" width="100%" cellpadding="4" cellpacing="3">  
 <th><fmt:message key="home.payments.receiverCard"></fmt:message></th>  
 <th><fmt:message key="home.payments.appointment"></fmt:message></th>  
 <th><fmt:message key="home.payments.date"></fmt:message></th>  
 <th><fmt:message key="home.payments.summa"></fmt:message></th>  
 <th>${status }</th>  
 <th></th>  
 <c:forEach items="${payments}" var="payment">  
 <tr align="center">  
 <td>${payment.cardNumber}</td>  
 <td>${payment.appointment}</td>  
 <td>${payment.date}</td>  
 <td>${payment.sum }</td>  
 <td><c:if test="${payment.condition}">  
 <fmt:message key="home.payments.success"></fmt:message>  
 </c:if> <c:if test="${!payment.condition}">  
 <fmt:message key="home.payments.error"></fmt:message>  
 </c:if></td>  
 <td><a  
 href="?command=repeatPayment&paymentId=${payment.paymentId}"><fmt:message  
 key="home.payments.repeat"></fmt:message></a></td>  
 </tr>  
 </c:forEach>  
 </table>  
 </div>  
 </c:if>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 <jsp:include page="template/footer.jsp" />  
</body>  
</html>