Федеральное государственной образовательное учреждение

Высшего профессионального образования

Национальный исследовательский технологический университет «МИСиС»

Институт информационных технологий и компьютерных наук (ИТКН)

Кафедра автоматизированных систем управления (АСУ)

Курсовая работа

по курсу «Технологии программирования»

Тема:

«Разработка прототипа сервиса для перевода денег между банковскими картами, с использованием Spring Framework»

Выполнил: студент 2 курса,

гр. БИВТ-21-6 Тихменев Петр

Руководитель: Козлов М. Е.

МОСКВА 2022

Оглавление

Введение

Одним из наиболее распространённых направлений в разработке программного обеспечения, в настоящее время, является разработка различных веб-сервисов, которые позволяют решать множество прикладных задач, начиная от создания обычных простейших сайтов и заканчивая сложными распределёнными системами управления производством, торговли, логистики, банковскими системами. Можно сказать, что веб-сервисы сейчас присутствуют абсолютно во всех аспектах современной жизни.

В данной курсовой работе, я реализую прототип веб-сервиса, предназначенного для перевода денег между банковскими картами. Это будет ограниченное по возможностям приложение, которое, тем не менее, будет предоставлять основные функции, которые требуются для осуществления денежных переводов в пределах одного банка (или платежной системы).

Постановка задачи

Целью курсовой работы является разработка веб-сервиса с использованием фреймворка Spring. Для удобства демонстрации веб-сервиса необходимо предусмотреть пользовательский веб-интерфейс.

Приложение должно включать следующий функционал:

- Front-end:
  - Возможность ввода данных о переводе (данные карт, сумма) в браузере
  - Отправка введенных данных на сервер
  - Вывод результатов выполнения операции перевода
  - Просмотр доступных карт в системе и их баланса
  - Просмотр всех осуществленных переводов в системе
- Back-end:
  - Структуры для хранения данных о пользователях, картах, переводах
  - Начальное заполнение структур
  - Обработка полученных данных о переводе, проверка введенных данных
  - Осуществление операции перевода – изменение баланса карт, запись информации о переводе
  - Вывод данных о картах и операциях

1. Теоретическая часть

1.1 Язык программирования Java

Для решения данной задачи использовались знания из области базовых средств объектно-ориентированного языка программирования Java. Понятие «объектно- ориентированный» относится к способу написания структурного кода Java, а именно: разделение кода на так называемые «классы», которые запускаются вместе, чтобы обеспечить согласованное порождение объектов. Такая структура программы приводит к универсальному и организованному коду, который легко редактировать и воспринимать. Кроме обычных методов классы могут определять специальные методы, которые называются конструкторами. Конструкторы вызываются при создании нового объекта данного класса и выполняют его инициализацию.

Главные достоинства данного языка – это его простота и объектно-ориентированный подход. Язык Java имеет чёткие синтаксические правила и понятную семантику. Для выполнения курсовой работы используется Java 18.

1.2 Формат HTML

Файл HTML относится к веб-страницам, при создании которых, использовался язык разметки HTML. Большое количество подобных web-страниц, соединенных между собой ссылками, способствуют образованию сайтов. Открыть файл, имеющий расширение HTML, можно при помощи разнообразных браузеров, таких как Mozilla Firefox и Google Chrome, а также Internet Explorer и так далее.

Данный формат файла, можно отредактировать в текстовом редакторе, так как .html файл представляется стандартным текстовым документом, однако специалисты рекомендуют производить какие-либо изменения файла расширением .html при помощи специализированного программного

обеспечения на подобии Adobe Dreamweaver. Примечательно, что HTML файлы, могут выполнять форматирование текстовых файлов, таблиц, изображений и прочего содержимого, отображаемого на страницах сайтов.

1.3 Фреймворк Spring Framework

Для разработки приложения был выбран фреймворк Spring Framework.

Spring Framework (или коротко Spring) — это универсальный фреймворк с открытым исходным кодом для Java-платформы. Особенность данного фреймворка заключается в том, что пользователю надо просто писать классы, а создает объекты классов и вызывает методы уже сам фреймворк. Чаще всего, классы имплементируют какие-то интерфейсы из фреймворка или наследуют какие-то классы из него, таким образом получая часть уже написанной за пользователя функциональности.

Можно сказать, что использование Spring упрощает работу и экономит время, так как большую часть написания кода берет на себя Spring.

1.4 Работа с базой данных PostgreSQL

В данной работе будем использовать базу данных созданную с применением СУБД PostgreSQL.

PostgreSQL – это свободная объектно-реляционная система управления базами данных (СУБД). Данная СУБД отличается от других тем, что она обладает объектно- ориентированным функционалом, в том числе полной поддержкой концепта ACID (Atomicity, Consistency, Isolation, Durability).

PostgreSQL обладает многими возможностями, которые привлекают разработчиков, наример:

- PostgreSQL - бесплатное ПО с открытым исходным кодом;

- PostgreSQL - кроссплатформенное ПО, к нему существуют интерфейсы из всех современных языков программирования;
- PostgreSQL хорошо масштабируется и обеспечивает высокую производительность;
- PostgreSQL очень надежна, аварийное завершение случается редко;
- Развитая функциональность позволяет гибко решать различные задачи.
- Развитая экосистема клиентов и административных средств позволяет легко и быстро выполнять такие рутинные задачи, как описание объектов базы данных, экспорт и импорт данных, резервное копирование и восстановление базы;
- PostgreSQL легко интегрируется с другими СУБД, что открывает возможности для гибкой реализации программных проектов.

2. Практическая часть

2.1 Детальные требования к программе

По заданию данное приложение должно уметь обрабатывать данные полученные после заполнения форм на html-страницах и формировать данные операции и результате ее выполнения.

Помимо этого, программа должна обеспечивать возможность выполнения перечисленных ниже функций:

- Автоматическое изменение хранимых данных при успешной операции;
- Вывод результата операции в виде html кода, для отображения в браузере;
- Просмотр списка всех операций (включая те, результат которых был ошибочным)
- Просмотр списка карт

2.2 Разбиение технического задания на задачи

Для разработки приложения необходимо разработать классы для работы с данными, классы для выполнения операций, набор html-страниц и основной код для запуска программы. На основе этого выделим следующие задачи:

- Создать класс для сущности «Card»;
- Создать класс для сущности «User»;
- Создать класс для сущности «Transfer»;
- Создать таблицы в базе данных;
- Создать класс repository;
- Создать класс service для репозитория;
- Создать класс controller для работы с сервисом;
- Создать класс DatabaseService для работы с базой данных;
- Создать html-страницу для операции перевода;

- Создать html-страницу для отображения успеха/ошибки операции;
- Создать html-страницу для отображения хранимых карт;
- Создать html-страницу для отображения данных всех операций;

Также для разработки приложения необходимо выполнить следующие задачи:

- Создать класс для запуска программы;
- Реализовать расчёты, требуемые в задании;
- Тестирование;

2.3 Разработка

Сущность Card предназначена для хранения ID карты, ее номера, даты конца использования, cvv номера, количества денег на карте и данных о владельце карты. Вторая сущность User хранит данные пользователя-держателя карты. Следующая сущность Transfer определяет полную информацию о совершаемой транзакции.

На начальном этапе был создан package model, в котором присутствуют следующие классы: Card, User, Transfer. Таким образом, были созданы три сущности.

Код для создания сущности Card:

Затем создан package repository.

Репозиторий — это слой абстракции, инкапсулирующий в себе всё, что относится к способу хранения данных. Назначение: разделение бизнес-логики от деталей реализации слоя доступа к данным.

Код для создания класса TransferRepository:

```
@Repository
public class TransferRepository {

    public static Map<String, Card> cardsRepository = new
ConcurrentHashMap<>();
    public static Map<String, User> userRepository = new
ConcurrentHashMap<>();
    public static Map<String, Transfer> transferRepository = new
ConcurrentHashMap<>();



    public TransferRepository(Map<String, Card> cardsRepository, Map<String,
Transfer> transferRepository, Map <String, User> userRepository) {
        TransferRepository.cardsRepository = cardsRepository;
        TransferRepository.transferRepository = transferRepository;
        TransferRepository.userRepository = userRepository;
    }
}
```

Таким образом, был создан класс TransferRepository.

Далее был создан package service. Созданный в этом пакете класс отвечает за клиентское представление веб-службы.

Service – это Java класс, который предоставляет с себя основную бизнес-логику.

В классе TransferService были созданы методы для создания транзакции и обновление данных существующих карт.

Код для создания класса TransferService:

```
@Service
public class TransferService {

    //основная функция для создания перевода (проверки, создание, апдейт)
    public boolean make_transfer(String card_source, String card_destination,
String card_source_exp_date,int card_source_cvv, String amount){
        boolean res = false;

        Transfer transfer;
```

```
        //проверки на существование введенных карт, если Id=0, то карта не
найдена
        int cardSourceId = checkCard(card_source, card_source_exp_date,
card_source_cvv);
        int cardDestinationId = checkCard(card_destination);

        if ((cardSourceId != 0) && (cardDestinationId !=0)){
            transfer = new Transfer();
            transfer.setSourceID(cardSourceId);
            transfer.setDate(new java.sql.Date(new
Date(System.currentTimeMillis()).getTime()));
            transfer.setDestinationID(cardDestinationId);
            transfer.setAmount(Double.valueOf(amount));
            transfer.setResult(false);
            transfer.save(); //создаем новый перевод
            if (update_cards(cardSourceId, cardDestinationId,
Double.valueOf(amount))){
                //если перевод успешен, то меняем его статус и апдейтим
                transfer.setResult(true);
                transfer.save();
            }
            res = transfer.isResult();
        }

        return res;
    }

    //апдейт сумм на картах
    public boolean update_cards(int cardSourceId, int cardDestinationId,
Double amount){
        boolean res = false;
        DatabaseService dbs = new DatabaseService();

        for (Map.Entry<String, Card> cardRepoEntry :
TransferRepository.cardsRepository.entrySet()) {
            if (cardRepoEntry.getValue().getId() == cardSourceId){
                if (cardRepoEntry.getValue().getAmount() >= amount) {
                    //если на карте есть доступная сумма, то меняем баланс на
обеих картах
                    dbs.updateCard(cardSourceId, -amount);
                    dbs.updateCard(cardDestinationId, amount);
                    res = true;
                }
            }
        }
        return res;
    }

    //проверка существования карты со всеми параметрами
    public int checkCard(String number_check, String exp_date_check, int
cvv_check){
        int res = 0;

        for (Map.Entry<String, Card> cardRepoEntry :
TransferRepository.cardsRepository.entrySet()) {
            if (cardRepoEntry.getValue().getNumber().equals(number_check))
                if (cardRepoEntry.getValue().getCVV() == cvv_check)
                    if
(cardRepoEntry.getValue().getDateExpiration().equals(exp_date_check))
                        res = cardRepoEntry.getValue().getId();
        }

        return res;
```

```
    }

    //проверка существования карты только по номеру (для destination карты)
    public int checkCard(String number_check){
        int res = 0;

        for (Map.Entry<String, Card> cardRepoEntry :
TransferRepository.cardsRepository.entrySet()) {
            if (cardRepoEntry.getValue().getNumber().equals(number_check))
                res = cardRepoEntry.getValue().getId();
        }

        return res;
    }
}
```

После создания класса TransferService был создан package controller.

Контроллер - это класс, предназначенный для непосредственной обработки запросов от клиента и возвращения результатов.

В package controller был создан класс TransferController с методами transfer, transfers, cards для загрузки и обработки соответствующих html – страниц.

Код для создания класса TransferController:

```
@RestController
@RequestMapping("/")
public class TransferController {

    //две строковые константы - результат выполнения перевода success/error
    private static final String string_html_success = "<span
class='success'>success</span>";
    private static final String string_html_error = "<span
class='error'>error</span>";
    private final TransferService service;

    public TransferController(TransferService service) {
        this.service = service;
    }

    //корень сайта
    @GetMapping(path = "/")
    public String Home() throws IOException {
        return getPageContent("templates/transfer.html");
    }

    //вывод результата выполнения операции (обработка значений, полученных из
формы)
    @PostMapping(path = "/transfer")
    public String transfer(@RequestParam("card_source") String card_source,
@RequestParam("card_destination") String card_destination,
@RequestParam("card_source_exp_date") String
card_source_exp_date,@RequestParam("card_source_cvv") int card_source_cvv,
@RequestParam("amount") String amount, HttpServletRequest request,
HttpServletResponse response) throws IOException  {
```

```
        String res;
        if
(service.make_transfer(card_source,card_destination,card_source_exp_date,
card_source_cvv, amount))
            res= string_html_success;
        else
            res= string_html_error;

        return getPageContent("templates/transfer_result.html").replace("<!--
transfer_result-->", res);
    }

    //вывод списка переводов
    @GetMapping(path = "/transfers")
    public String transfers() throws IOException {
        String transfersList = getTransfers();
        return getPageContent("templates/transfers.html").replace("<!--
transfers_list-->", transfersList);
    }

    //вывод списка карт
    @GetMapping(path = "/cards")
    public String cards() throws IOException {
        String cardsList = getCards();
        return getPageContent("templates/cards.html").replace("<!--
cards_list-->", cardsList);
    }

    //получение списка карт в формате строки html таблицы
    public String getCards(){
        String res ="";
        for (Map.Entry<String, Card> cardRepoEntry :
TransferRepository.cardsRepository.entrySet()) {
            res += cardRepoEntry.getValue().toHTMLTableString();
        }

        return res;
    }

    //получение списка переводов в формате строки html таблицы
    public String getTransfers(){
        String res ="";

        for (Map.Entry<String, Transfer> transferRepoEntry :
TransferRepository.transferRepository.entrySet()) {
            res += transferRepoEntry.getValue().toHTMLTableString();
        }
        return res;
    }

    static String getPageContent(String pageName) throws IOException {
        Resource resource = new ClassPathResource(pageName);
        String filepath = resource.getFile().toString();

        return readFile(filepath, Charset.defaultCharset());
    }

    //чтение файла (шаблона html страницы)
    static String readFile(String path, Charset encoding)  throws IOException
{
        byte[] encoded = Files.readAllBytes(Paths.get(path));
        return new String(encoded, encoding);
    }
}
```

Основной класс для запуска программы, здесь запускается приложение и производится начальное заполнение структуры данных:

```
@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class })
public class CoursachApplication implements CommandLineRunner {

    public static void main(String[] args){
        SpringApplication.run(CoursachApplication.class, args);
        System.out.println("Program data:");
        printRepo();
    }
    @Override
    public void run(String... args){
        DatabaseService dbs = new DatabaseService();
        dbs.getFullData();
    }

    public static void printRepo() {
        System.out.println("Users:");
        TransferRepository.userRepository.forEach((key, value) ->
System.out.println("UserId: " + key + " UserData: " + value.toString()));
        System.out.println("Cards:");
        TransferRepository.cardsRepository.forEach((key, value) ->
System.out.println("CardId: " + key + " DataCard: " + value.toString()));
        System.out.println("Transfers:");
        TransferRepository.transferRepository.forEach((key, value) ->
System.out.println("TransferId: " + key + " TransferData: " +
value.toString()));

    }
}
```

Для хранения данных была создана база данных и таблицы сущностей. Для подключения к СУБД PostgreSQL использовался клиент pgAdmin.

Создание таблиц:

```
CREATE TABLE IF NOT EXISTS public.card
(
    "Id" integer NOT NULL DEFAULT nextval('"Card_Id_seq"'::regclass),
    "UserId" integer NOT NULL,
    "Number" character varying(50) COLLATE pg_catalog."default",
    "DateExp" character varying(30) COLLATE pg_catalog."default",
    "CVV" smallint,
    "Amount" numeric,
    CONSTRAINT "Card_pkey" PRIMARY KEY ("Id"),
    CONSTRAINT "UserId" FOREIGN KEY ("UserId")
        REFERENCES public.card_user ("Id") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)
```

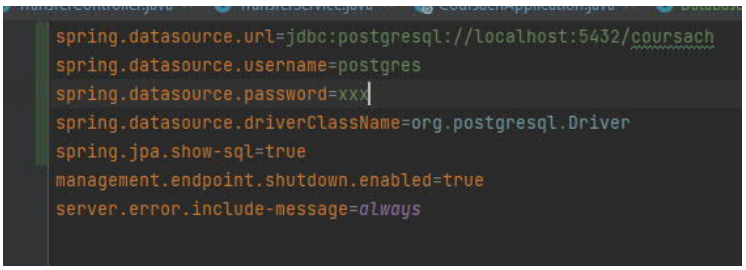
```
CREATE TABLE IF NOT EXISTS public.card_user
(
    "Id" integer NOT NULL DEFAULT nextval('"User_id_seq"'::regclass),
    "FirstName" character varying(30) COLLATE pg_catalog."default",
    "LastName" character varying(30) COLLATE pg_catalog."default",
    "Birthday" date,
    "Address" character varying(50) COLLATE pg_catalog."default",
    "Phone" bigint,
    CONSTRAINT "User_pkey" PRIMARY KEY ("Id")
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.card_user
    OWNER to postgres;
```

```
CREATE TABLE IF NOT EXISTS public.transfer
(
    "Id" integer NOT NULL DEFAULT nextval('"Transfer_TransferId_seq"'::regclass),
    "TransactionDate" date,
    "CardSourceId" integer,
    "CardDestinationId" integer,
    "Amount" numeric,
    "Result" boolean,
    CONSTRAINT "Transfer_pkey" PRIMARY KEY ("Id"),
    CONSTRAINT card_destination FOREIGN KEY ("CardDestinationId")
        REFERENCES public.card ("Id") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT card_source FOREIGN KEY ("CardSourceId")
        REFERENCES public.card ("Id") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)
```

Далее был создан класс DatabaseService для работы с БД:

```java
3 usages
public DatabaseService() {

}

//создание коннекции
6 usages
private Connection getCnn(){
    Connection cnn;
    DBProperties properties = DBProperties.getProperties();
    try {
        cnn = DriverManager.getConnection(
                properties.getUrl(),
                properties.getUser(),
                properties.getPassword());
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return cnn;
}

//загрузка всей информации из БД в классы
4 usages
public void getFullData(){
    getUsers();
    getCards();
    getTransfers();
}

//получение инфы о юзерах из БД в userRepository
1 usage
public void getUsers() {
    User user;
    String query = "SELECT * FROM card_user";
    try(PreparedStatement statement = getCnn().prepareStatement(query)) {
```

Для подключения к БД используем Connection, в который передается путь к базе и параметры (имя/пароль) пользователя, которым мы подключаемся к базе. Параметры хранятся в файле application.properties:



Для работы с параметрами используются вспомогательные классы ApplicationProperties и DBProperties.

Получение и изменение информации в базе данных происходит с использованием методов класса DatabaseService:

- getUsers()
- getTransfers()
- getCards()
- addTransfer()
- updateTransfer()
- updateCard()

Ниже приведен код методов getTransfers() и addTransfer():

```
//получение инфы о переводах из БД в transferRepository
1 usage
public void getTransfers(){
    Transfer transfer;
    String query = "SELECT * FROM transfer";
    try(PreparedStatement statement = getCnn().prepareStatement(query)) {
        try(ResultSet result = statement.executeQuery()) {
            while (result.next()) {
                transfer = new Transfer();
                transfer.setId(result.getInt( columnLabel: "Id"));
                transfer.setDate(result.getDate( columnLabel: "TransactionDate"));
                transfer.setSourceID(result.getInt( columnLabel: "CardSourceId"));
                transfer.setDestinationID(result.getInt( columnLabel: "CardDestinationId"));
                transfer.setAmount(result.getDouble( columnLabel: "Amount"));
                transfer.setResult(result.getBoolean( columnLabel: "Result"));
                TransferRepository.transferRepository.put(String.valueOf(result.getInt( columnLabel: "Id")), transfer);
            }
        }
    } catch (SQLException e) {}
}
```

```
//сохранение записи о переводе в БД (возвращает Id записи в БД)
1 usage
public int addTransfer(Transfer transfer) {
    int key = 0;
    String query = "INSERT INTO transfer (\"TransactionDate\", \"CardSourceId\", \"CardDestinationId\", \"Amount\", \"Result\") "
            + " VALUES (?, ?, ?, ?, ?)";
    try (PreparedStatement statement = getCnn().prepareStatement(query, Statement.RETURN_GENERATED_KEYS)) {
        statement.setDate( parameterIndex: 1, transfer.getDate());
        statement.setInt( parameterIndex: 2, transfer.getSourceID());
        statement.setInt( parameterIndex: 3, transfer.getDestinationID());
        statement.setDouble( parameterIndex: 4, transfer.getAmount());
        statement.setBoolean( parameterIndex: 5, transfer.isResult());
        statement.executeUpdate();
        ResultSet rs = statement.getGeneratedKeys();
        if ( rs.next() ) {
            key = rs.getInt( columnIndex: 1);
        }
    } catch (SQLException e) { }

    getFullData();
    return key;
}
```

Код остальных методов аналогичный. Создается SQL-запрос, если необходимо, то с параметрами. Затем выполняется подключение к БД и выполнение текста SQL-запроса. Результаты запроса сохраняем в репозитории, либо в случае INSERT'а – получаем Id добавленной в таблицу БД записи.

После создания классов, проект имеет следующую структуру, рисунок 1.

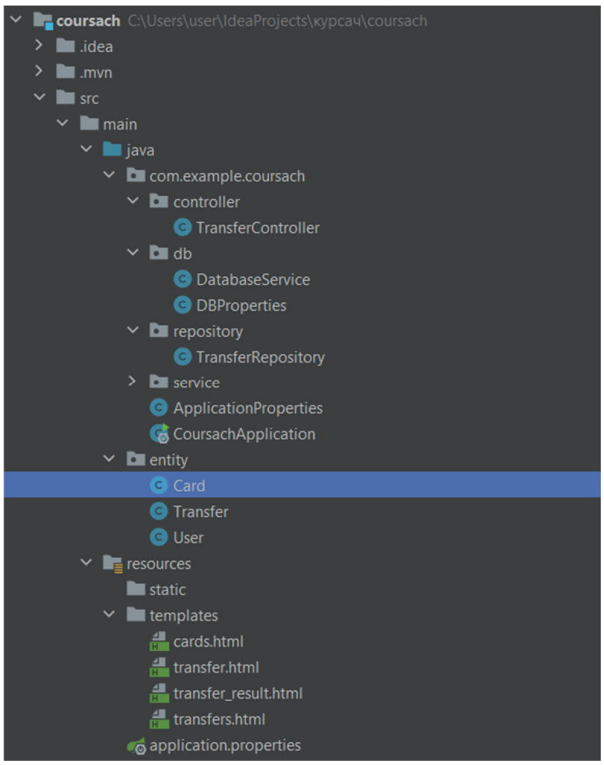


Рисунок 1 – структура проекта

2.4 Результат работы программы

Для запуска и выполнения программы, пользователю необходимо открыть любой браузер и обратиться к серверу, на котором работает приложение. При вводе адреса сервера будет открыта начальная страница сервиса, со страницей интерфейса перевода с карты на карту, рисунок 2.

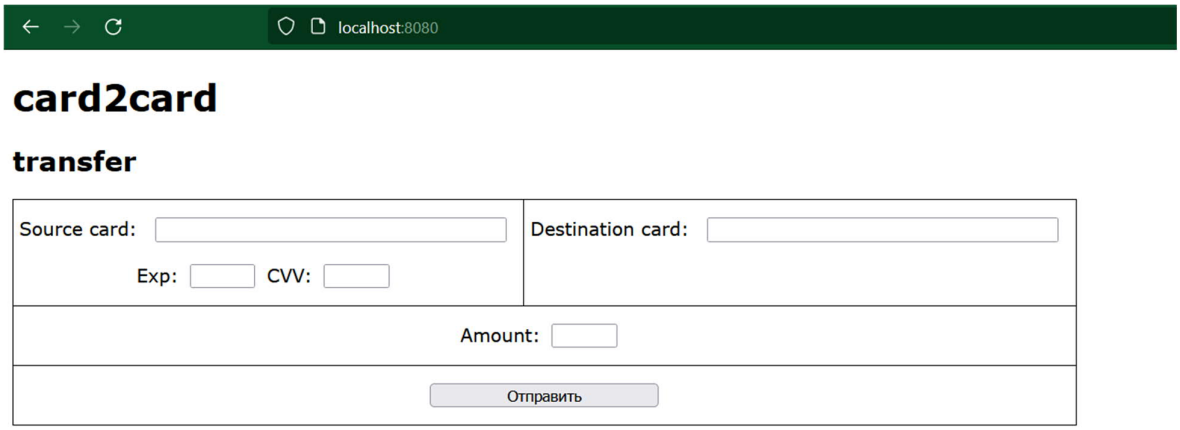


Рисунок 2 - страница для переводов с карты на карту

Html- код страницы:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Transfer card2card</title>
    <style>
        body {font-family: Verdana;Arial;}
        table, td {border: solid 1px;   border-collapse: collapse; padding: 5px; vertical-align: top; text-align: center;}
        div {padding: 10px;}
        input {width:200px;margin: 10px;}
        input.big {width:300px;}
        input.small {width:50px;}
    </style>
</head>
<body>
<h1>card2card</h1>
<h2>transfer</h2>
<form name="form_transfer" method="post" action="http://localhost:8080/transfer">
    <table>
        <tr>
            <td>Source card: <input class="big" name="card_source"/><br>Exp:<input class="small" name="card_source_exp_date"/>CVV:
            <td>Destination card: <input class="big" name="card_destination"/></td>
        </tr>
        <tr><td colspan="2">Amount:<input class="small" name="amount"/></td></tr>
        <tr><td colspan="2">    <input type="submit" value="Отправить"></td></tr>
    </table>
</form>
</body>
</html>
```

Здесь используется форма со строго заданными именами полей, которые после нажатия на кнопку «Отправить», будут отправлены методом POST на сервер, для обработки обработчиком /transfer в REST контроллере:

```
//вывод результата выполнения операции (обработка значений, полученных из формы)
@PostMapping(path = "/transfer")
public String transfer(@RequestParam("card_source") String card_source,
@RequestParam("card_destination") String card_destination,
@RequestParam("card_source_exp_date") String
```

```
card_source_exp_date,@RequestParam("card_source_cvv") int card_source_cvv,
@RequestParam("amount") String amount, HttpServletRequest request,
HttpServletResponse response) throws IOException  {
    String res;
    if
(service.make_transfer(card_source,card_destination,card_source_exp_date,
card_source_cvv, amount))
        res= string_html_success;
    else
        res= string_html_error;

    return getPageContent("templates/transfer_result.html").replace("<!--
transfer_result-->", res);
}
```

Аналогично реализованы обработчики /cards – вывод списка карт и /transfers – вывод списка все переводов:

```
//вывод списка переводов
@GetMapping(path = "/transfers")
public String transfers() throws IOException {
    String transfersList = getTransfers();
    return getPageContent("templates/transfers.html").replace("<!--
transfers_list-->", transfersList);
}

//вывод списка карт
@GetMapping(path = "/cards")
public String cards() throws IOException {
    String cardsList = getCards();
    return getPageContent("templates/cards.html").replace("<!--cards_list--
>", cardsList);
}
```

Основной алгоритм работы обработчиков – чтение шаблона html-страницы:

```
4 usages
static String getPageContent(String pageName) throws IOException {
    Resource resource = new ClassPathResource(pageName);
    String filepath = resource.getFile().toString();

    return readFile(filepath, Charset.defaultCharset());
}

1 usage
static String readFile(String path, Charset encoding)  throws IOException {
    byte[] encoded = Files.readAllBytes(Paths.get(path));
    return new String(encoded, encoding);
}
```

затем текстовая замена тэга с комментариями в шаблоне, на полученную информацию (список карт, либо список переводов):

```
//получение списка карт в формате строки html таблицы
```

```java
public String getCards(){
    String res ="";
    for (Map.Entry<String, Card> cardRepoEntry :
TransferRepository.cardsRepository.entrySet()) {
        res += cardRepoEntry.getValue().toHTMLTableString();
    }

    return res;
}

//получение списка переводов в формате строки html таблицы
public String getTransfers(){
    String res ="";

    for (Map.Entry<String, Transfer> transferRepoEntry :
TransferRepository.transferRepository.entrySet()) {
        res += transferRepoEntry.getValue().toHTMLTableString();
    }
    return res;
}
```

Ниже представлен метод .toHTMLTableString() для сущности Card – отображает информацию о карте, в формате html строки таблицы (для сущности Transfer реализован аналогичный метод):

```java
//вывод инфы о переводе в строку html-таблицы (с тегами)
1 usage  Petroivch *
public String toHTMLTableString() {
    String res = result ? "<td><span class=success>" + result + "</span></td>" : "<td><span class=error>" + result + "</span></td>";
    return "<tr>" +
            "<td>" + id + "</td>" +
            "<td>" + DateTransaction + "</td>" +
            "<td>" + SourceID + "</td>" +
            "<td>" + DestinationID + "</td>" +
            "<td>" + Amount + "</td>" +
            res +
            "</tr>";
}
```

После отправки данных, введенных на начальной странице сервиса (рис.2), отображается страница с результатом выполнения операции перевода, рисунок 3.

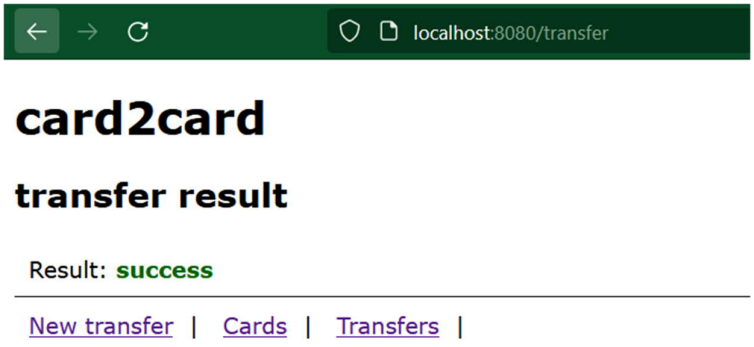


Рисунок 3 - результат работы сервиса

По ссылкам на странице, можно перейти на просмотр списка всех зарегистрированных карт, а также списка переводов (включая неудачные), рисунки 4,5.

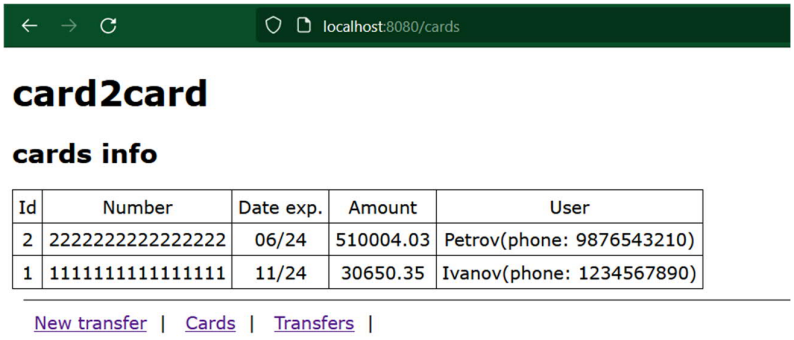


Рисунок 4 – список карт, зарегистрированных в сервисе

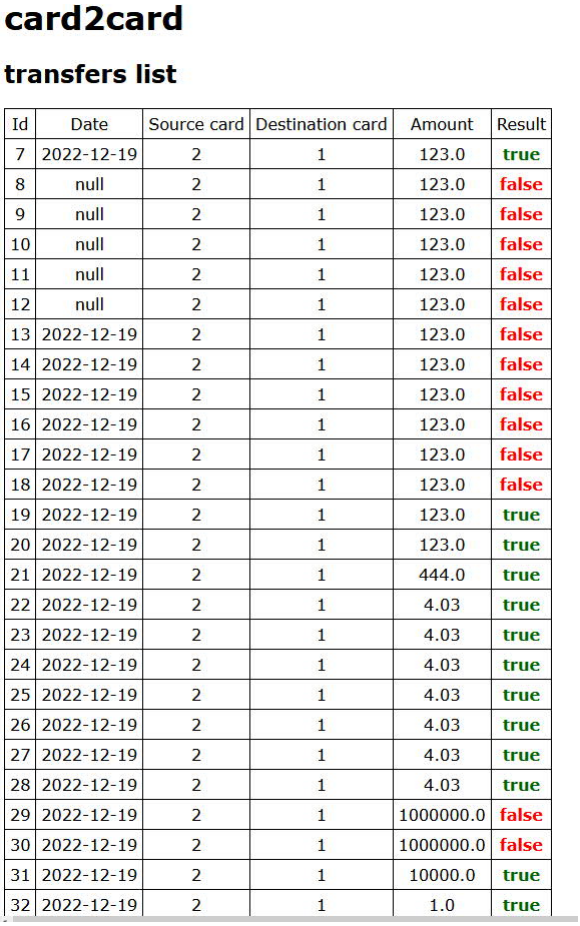


Рисунок 5 – список переводов

Заключение

В последнее время, количество сетевых приложений неуклонно растет, так как большинство современных системы являются или полностью веб-ориентированными, либо имеют полноценные веб-версии, т.к. интернет стал неотъемлемой частью нашей жизни.

Поэтому сетевые приложения часто имеют сложный пользовательский интерфейс, а также полноценный back-end, в котором реализуются сложные алгоритмы. Сюда можно отнести различные сетевые приложения, сервисы и т. д. Также сетевые приложения являются очень популярными, из-за того, не требуют локальной установки на устройства пользователей, все обновления происходят на сервере, доставляются пользователям сразу — достаточно просто перезагрузить страницу или выйти, а потом снова зайти в аккаунт. Но один из минусов таких приложений состоит в том, что их сложнее реализовать, тестировать и отлаживать. Ещё хочу добавить, что разработка сетевого приложения также актуальна для начинающих разработчиков, так как это хороший вариант начала изучения сетевых приложений.

В результате выполнения курсовой работы была изучена специальная литература, описаны теоретические аспекты и раскрыты ключевые понятия исследования, а самое главное было создано сетевое приложение-сервис с использованием Spring Framework. Данное приложение поддерживает работу с html-страницами, создаёт операции и выводит соответствующий результат по операции.

При создании сетевого приложения были использованы и изучены такие инструменты, как язык программирования Java и фреймворк Spring. Также могу добавить, что созданный прототип сервиса для перевода имеет хороший потенциал для дальнейшей модернизации и вполне может быть использован, как часть платежной системы или шлюза.

Список использованных источников и литературы

1. Spring Quickstart Guide [электронный ресурс]: https://spring.io/quickstart

2. Thymeleaf [электронный ресурс]: https://www.thymeleaf.org/documentation.html

3. С. А. Орлов «Программная инженерия. Технологии разработки программного обеспечения». 5-е издание обновленное и дополненное – СПб.: Питер, 2016–640 с.

4. К. Сьерра «Изучаем Java» - Эксмо, 2012–720 с.

5. Г. Шилдт «Java. Полное руководство» 10-е издание - Диалектика-Вильямс, 2018– 1488 с.

6. Ю. Козмина, Р. Харроп «Spring 5 для профессионалов» - Диалектика-Вильямс, 2019– 1120 с.

7. А. Швец «Погружение в паттерны проектирования» - Самиздат, 2018–406 с.