Министерство науки и высшего образования Российской Федерации  
федеральное государственное бюджетное образовательное учреждение высшего образования «Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

|  |  |
| --- | --- |
| Отчет защищен с оценкой |  |
|  | В. Г. Лукоянычев |
| « » \_\_\_\_\_\_ 2022 г. | |

**ОТЧЕТ**

О производственной практике

Технологическая (проектно-технологическая) практика

По теме «Разработка клиент-серверного приложения на примере интеграции с сайтом Центробанка»

в ПАО «СберБанк»

|  |  |  |  |
| --- | --- | --- | --- |
| Студент гр. ПИ-91 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | А. Ю. Дружинин |
|  | Подпись | | И. О. Ф |
| Руководитель от профильной организации | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | руководитель направления | В. А. Миняйлов |
|  | Подпись | Должность | И .О. Ф |
| Руководитель от университета | доцент, к.т.н. | | В. Г. Лукоянычев |

Должность, ученая степень

,

2022

ФГБОУ ВО «Алтайский государственный технический университет им. И. И. Ползунова»

Кафедра прикладной математики

**Индивидуальное задание**

на производственную практику

Технологическая (проектно-технологическая) практика

(вид и тип практики по УП)

студенту Дружинину А. Ю. группы ПИ-91.

(Ф.И.О.)

**Профильная организация ПАО «СберБанк»**

(наименование)

**График проведения практики:**

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Содержание работ, выполняемых на практике** | **Сроки выполнения** |
| 1 | Изучение организации работы предприятия и используемого на нем инструментария и ПО | 20.06-21.06 |
| 2 | Формулировка задач для решения в ходе практики, вида и объема результатов | 21.06 |
| 3 | Изучение и анализ предметной области, библиографический поиск, изучение литературы. | 21.06 – 23.06 |
| 4 | Постановка задачи, проектирование состава и структуры ПО | 24.06 – 26.06 |
| 5 | Реализация программного обеспечения | 27.06 -14.07 |
| 6 | Тестирование программного обеспечения | 14.07 – 15.07 |
| 7 | Оформление и сдача отчета по практике | 16.07-17.07 |

Руководитель практики от университета В. Г. Лукоянычев, доцент, к.т.н.

(подпись) (Ф.И.О., должность)

Руководитель практики от

профильной организации Миняйлов В. А., руководитель направления

(подпись) (Ф.И.О., должность)

Задание принял к исполнению Дружинин А. Ю.

(подпись) (Ф.И.О.)

**Инструктаж по ОТ, ТБ, ПБ, ПВТР**

Инструктаж обучающегося по ознакомлению с требованиями охраны труда, техники безопасности, пожарной безопасности, а также правилами внутреннего трудового распорядка проведен « » 2022 г.

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель практики от  профильной организации |  | Бураков А. О.., старший инженер по разработке | |
|  | (подпись) | | (Ф.И.О., должность) |

Содержание

[Введение 2](#__RefHeading___Toc34099_1357953486)

[Знакомство с организацией 3](#__RefHeading___Toc34101_1357953486)

[1. Описание предметной области и постановка задачи 4](#__RefHeading___Toc34103_1357953486)

[1.1 Описание предметной области 4](#__RefHeading___Toc34112_1357953486)

[1.2 Постановка задачи 5](#__RefHeading___Toc34114_1357953486)

[2. Проектирование 7](#__RefHeading___Toc34116_1357953486)

[2.1 Описание структуры классов 7](#__RefHeading___Toc34118_1357953486)

[2.2 Диаграмма классов 9](#__RefHeading___Toc34120_1357953486)

[3. Инструменты и технологии 10](#__RefHeading___Toc34124_1357953486)

[Заключение 12](#__RefHeading___Toc34126_1357953486)

[Список использованных источников 13](#__RefHeading___Toc34128_1357953486)

[Приложение А. Тестирование программного обеспечения 14](#__RefHeading___Toc34130_1357953486)

[Приложение Б. Исходный текст программы 20](#__RefHeading___Toc34132_1357953486)

# **Введение**

Производственная практика проходила в ПАО “Сбербанк”.

Производственная практика была направлена на проектирование и разработку системы с использованием клиент-серверной модели.

Разрабатываемая система позволяет смотреть курсы валют, которые можно найти на сайте Центрального банка Российской Федерации и отслеживать их изменения за промежуток времени.

Разработка велась в команде, в которой состояло три человека. Backend разрабатывался вместе, frontend, мобильное приложение и тестирование было поделено между участниками команды.

# **Знакомство с организацией**

SberTech — IT артерия Сбера. Компания разрабатывает IT-продукты для крупнейшего банка России, помогая ему строить экосистему цифровых сервисов.

Ключевой продукт SberTech — Platform V, которая обеспечивает высокий уровень цифровизации, аналитику данных, скорость принятия решений и вывода продуктов на рынок. Это значит, что тот опыт, который вы сможете приобрести – уникален. Уже сегодня компания работает над проектами, превосходящими по масштабам и сложности все, что когда-либо делалось в мире.

SberTech — это:

* команды, которые ориентируются в стремительно меняющемся технологическом стеке;
* R&D центр для пилотирования прорывных технологий;
* программы обучения и стажировок в МФТИ, МИФИ и других ведущих вузах.

# **Описание предметной области и постановка задачи**

## Описание предметной области

**Модель клиент–сервер** - это распределенная структура приложения, которая разделяет задачи или рабочие нагрузки между поставщиками ресурса или услуги, называемыми серверами, и запросчиками услуг, называемыми клиентами.

В **модели клиент-сервер** сервер часто предназначен для работы в качестве централизованной системы, обслуживающей множество клиентов.

**Клиенты и серверы** обмениваются сообщениями по схеме «запрос-ответ» . Клиент отправляет запрос, а сервер возвращает ответ. Этот обмен сообщениями является примером межпроцессного взаимодействия . Для общения компьютеры должны иметь общий язык и следовать правилам, чтобы и клиент, и сервер знали, чего ожидать. Язык и правила общения определяются в коммуникационном протоколе . Все протоколы работают на прикладном уровне . Протокол прикладного уровня определяет основные шаблоны диалога. Чтобы еще больше формализовать обмен данными, сервер может реализовать интерфейс прикладного программирования (**API**).

**API** — это уровень абстракции для доступа к сервису. Ограничивая общение определенным форматом контента, он облегчает синтаксический анализ Абстрагируя доступ, он облегчает межплатформенный обмен данными. API описывает, в каком формате осуществляется запрос серверу и в каком формате сервер отправляет ответ клиенту. Наиболее распространенный формат, в котором сервер направляет ответ клиенту - это в виде **JSON**.

Засчёт своей лаконичности по сравнению с **XML** формат **JSON** может быть более подходящим для сериализации сложных структур. Применяется в веб-приложениях как для обмена данными между браузером и сервером, так и между серверами (программные HTTP-сопряжения).

Поскольку формат **JSON** является подмножеством синтаксиса языка **JavaScript**, то он может быть быстро десериализован встроенной функцией *JSON.parse()*

**Apache Maven** — фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (*Project Object Model*), являющемся подмножеством XML

**Валю́тный курс** — цена (котировка) денежной единицы одной страны, выраженная в денежной единице другой страны.

## Постановка задачи

Исходя из описания предметной области, разработать клиент-серверное приложение для получения данных о курсах валют с сайта Центробанка и отображения их в удобной для пользователя форме.

Так как приложение представляет собой клиент-серверную модель, оно должно содержать один сервер, где будет реализована вся логика обращения с данными о курсах валют и два клиента - сайта и Android-приложения. Сервер рекомендуется разрабатывать с использованием технологии Spring Boot на языке Java. Сервер должен принимать запросы от клиентов по интерфейсу REST API и выполнять запрос к сайту Центробанка по специальному API.

Описание API Центробанка:

<http://www.cbr.ru/development/SXML/>

Данные по запросу с сайта Центробанка будут возвращаться в формате XML, после чего сервер должен будет их обработать, внести новые данные в базу данных на PostgreSQL и вернуть данные обратно клиенту. Если все данные по запросу клиента уже есть в базе данных, можно не обращаться к сайту Центробанка, а просто вернуть данные из нее. Данные от сервера клиенту возвращаются в формате json.   
Использовать maven для сборки проекта

Клиент-сайт рекомендуется разрабатывать, используя JavaScript-библиотеку React.

**Рекомендации по выполнению:**

* Не писать весь код в одном классе (Single-responsibility principle)
* Использовать Unit-тесты
* Работать с git – разбить выполнение тестового на подзадачи и каждую подзадачу оформлять отдельным коммитом
* Обработка ошибок:
  + пользователь может ввести некорректные данные;
  + запрос к API может завершиться ошибкой – тут можно обратить внимание на HTTP код ответа;
  + прочие возможные ошибки;
* Ресурсы нужно закрывать после использования)

# **Проектирование**

## Описание структуры классов

Основные классы представляют собой 3-уровневый слой абстрации (Controller -> Service -> DAO (data access object pattern).

Контроллеры:

1. DayCurrencyController

Методы:

1. getCurrenciesForPeriod(Date, Date, String)

Метод принимает на вход начальную дату периода, конечную дату периода и строку, обозначающую код валюты.

Возвращает List<DayCurrency> список всех значений определенной валюты за период.

1. getAllCurrenciesForDay(Date)

Метод принимает на вход дату (день).

Возвращает List<FullCurrencyInfo> список всех валют за сегодня с полной информацией о валюте.

Свойства:

private final DayCurrencyService dayCurrencyService;

1. CurrencyController

Сервисы:

1. DayCurrencyService

Методы:

getCurrenciesForPeriod(Date, Date, String)

Метод принимает на вход начальную дату периода, конечную дату периода и строку, обозначающую код валюты.

Возвращает List<DayCurrency> список всех значений определенной валюты за период.

getAllCurrenciesForDay(Date)

Метод принимает на вход дату (день).

Возвращает List<FullCurrencyInfo> список всех валют за сегодня с полной информацией о валюте.

insert(DayCurrency dayCurrency, String currencyName)

Вставляет данные в Currency таблицу БД.

fillInEmptyLines(Date fromDate, Date toDate,List<DayCurrency> dayCurrencyList,String currencyId)

Заполняет данные по валютам, если есть пропуски (например, с ЦБ не пришли данные по валютам за определенные дни (праздники/выходные). Метод запрашивает за прошлые даты.

List<DayCurrency> fillEmptyList(List<DayCurrency> dayCurrencyList,Date fromDate, Date toDate,String currencyId)

Заполняет данные по валютам, если пришел пустой список. Метод запрашивает за прошлые даты.

List<Date> getMissingDates(Date fromDate, Date toDate, List<DayCurrency> dayCurrencyList).

Метод определяет, за данные по каким датам пропущены.

static Date removeTime(Date date)

Обнуляет часы, минуты, секунды, миллисекунды у даты.

Свойства:

1. private final DayCurrencyDAO dayCurrencyDAO;
2. private final CurrencyDAO currencyDAO;
3. private final XMLParser xmlParser;
4. private final Environment environment;

CurrencyService

Методы:

List<CurrencyMenuItem> getAllCurrencyDesignations()

Возвращает данные по всем валютам для меню.

initDB()

Инициализирует Currency таблицу.

Свойства:

1. private final CurrencyDAO currencyDAO;
2. private final XMLParser xmlParser;

DAO:

1. DayCurrencyDAO

Методы:

getCurrenciesForPeriod(Date, Date, String)

Метод принимает на вход начальную дату периода, конечную дату периода и строку, обозначающую код валюты.

Возвращает List<DayCurrency> список всех значений определенной валюты за период.

getAllCurrenciesForDay(Date)

Метод принимает на вход дату (день).

Возвращает List<FullCurrencyInfo> список всех валют за сегодня с полной информацией о валюте.

1. CurrencyServiceDAO

Модели:

1. Currency

Методы:

Сеттеры, геттеры, toString()

Свойства:

private String PK\_id;

private int numCode;

private String charCode;

private String name;

1. DayCurrency

Методы:

Сеттеры, геттеры, toString()

Свойства:

private int PK\_daycur;

private double value;

private Date date;

private int nominal;

private String PK\_id;

1. FullCurrencyInfo

Методы:

Сеттеры, геттеры, toString()

Свойства:

private double value;

private Date date;

private int nominal;

private String charcode;

private String name;

1. CurrencyMenuItem

Методы:

Сеттеры, геттеры, toString()

Свойства:

1. private String charCode;
2. private String name;

Парсер:

* XMLParser

Методы:

xmlDailyValutes(Date date)

Метод принимает дату (день).

Вытягивает из возвращает из API центробанка данные по валютам за определенный день (List<DayCurrency>).

xmlConnectPeriod(Date startDate, Date endDate, String currencyID)

Метод принимает начальную дату, конечную дату и код валюты.

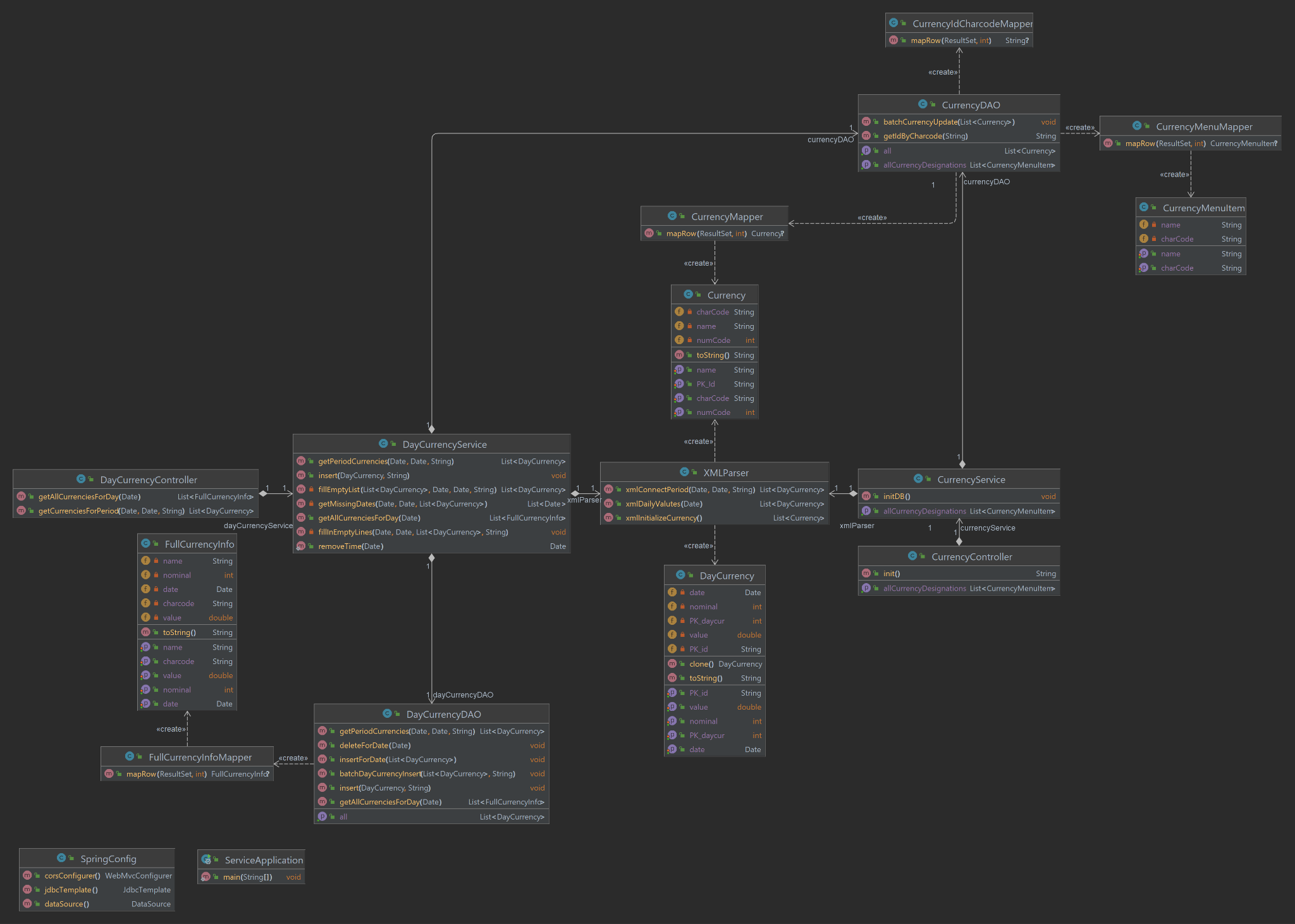
Вытягивает из возвращает из API центробанка данные по определенной валюте за определенный период (List<DayCurrency>).

xmlInitializeCurrency()

Свойства:

1. List<String> IDList
2. List<Integer> numCodeList
3. List<String> charCodeList
4. List<String> nameList

## Диаграмма классов



# **Инструменты и технологии**

Для разработки были использованы следующие технологии: язык программирования Java, фреймфорк Spring Boot, база данных PostgreSQL, язык программирования JavaScript, библиотека React, фреймворк Junit, система управления версиями Git.

**Spring Boot** — это полезный проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода.

**Spring Boot** обладает большим функционалом, но его наиболее значимыми особенностями являются: управление зависимостями, автоматическая конфигурация и встроенные контейнеры сервлетов.

**Spring Boot** собирает все общие зависимости и определяет их в одном месте, что позволяет разработчикам просто использовать их, вместо того, чтобы изобретать колесо каждый раз, когда они создают новое приложение.

**PostgreSQL** (произносится *«Пост-Грэс-Кью-Эл»*) — [свободная](https://ru.wikipedia.org/wiki/Свободное_ПО) [объектно-реляционная](https://ru.wikipedia.org/wiki/Объектно-реляционная_СУБД) [система управления базами данных](https://ru.wikipedia.org/wiki/Система_управления_базами_данных) (СУБД). **PostgreSQL** базируется на языке [SQL](https://ru.wikipedia.org/wiki/SQL) и поддерживает многие из возможностей стандарта SQL:2011.

Сильными сторонами **PostgreSQL** считаются:

* высокопроизводительные и надежные механизмы [транзакций](https://ru.wikipedia.org/wiki/Транзакция_(информатика)) и [репликации](https://ru.wikipedia.org/wiki/Репликация_(вычислительная_техника));
* расширяемая система встроенных языков программирования: в стандартной поставке поддерживаются [PL/pgSQL](https://ru.wikipedia.org/wiki/PL/pgSQL), [PL/Perl](https://ru.wikipedia.org/wiki/Perl), [PL/Python](https://ru.wikipedia.org/wiki/Python) и [PL/Tcl](https://ru.wikipedia.org/wiki/PL/Tcl); дополнительно можно использовать [PL/Java](https://ru.wikipedia.org/wiki/Java), [PL/PHP](https://ru.wikipedia.org/wiki/PHP), PL/Py, [PL/R](https://ru.wikipedia.org/wiki/R_(язык_программирования)), [PL/Ruby](https://ru.wikipedia.org/wiki/Ruby), [PL/Scheme](https://ru.wikipedia.org/wiki/Scheme), [PL/sh](https://ru.wikipedia.org/wiki/Командная_оболочка_Unix) и [PL/V8](https://ru.wikipedia.org/wiki/V8_(движок_JavaScript)), а также имеется поддержка загрузки модулей расширения на языке [C](https://ru.wikipedia.org/wiki/Си_(язык_программирования));
* [наследование](https://ru.wikipedia.org/wiki/Наследование_(программирование));
* возможность индексирования геометрических (в частности, географических) объектов и наличие базирующегося на ней расширения [PostGIS](https://ru.wikipedia.org/wiki/PostGIS);
* встроенная поддержка [слабоструктурированных данных](https://ru.wikipedia.org/wiki/Слабоструктурированные_данные) в формате [JSON](https://ru.wikipedia.org/wiki/JSON) с возможностью их индексации;
* расширяемость (возможность создавать новые типы данных, типы индексов, языки программирования, модули расширения, подключать любые внешние источники данных).

**React** — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов.

**React** может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость разработки, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, Redux и GraphQL.

**JUnit** — фреймворк для модульного тестирования программного обеспечения на языке Java.

Созданный Кентом Беком и Эриком Гаммой, **JUnit** принадлежит семье фреймворков xUnit для разных языков программирования, берущей начало в SUnit Кента Бека для Smalltalk. JUnit породил экосистему расширений — JMock, EasyMock, DbUnit, HttpUnit и т. Д.

**JUnit** был портирован на другие языки, включая PHP (PHPUnit), C# (NUnit), Python (PyUnit), Fortran (fUnit), Delphi (DUnit), Free Pascal (FPCUnit), Perl (Test::Unit), C++ (CPPUnit), Flex (FlexUnit), JavaScript (JSUnit), COS (COSUnit).

Опыт, полученный при работе с JUnit, важен в разработке концепций тестирования программного обеспечения.

**Git** (произносится «гит») — распределённая [система управления версиями](https://ru.wikipedia.org/wiki/Система_управления_версиями). Проект был создан [Линусом Торвальдсом](https://ru.wikipedia.org/wiki/Торвальдс,_Линус) для управления разработкой [ядра Linux](https://ru.wikipedia.org/wiki/Ядро_Linux), первая версия выпущена [7 апреля](https://ru.wikipedia.org/wiki/7_апреля) [2005 года](https://ru.wikipedia.org/wiki/2005_год). На сегодняшний день его поддерживает [Джунио Хамано](https://ru.wikipedia.org/wiki/Хамано,_Джунио).

Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.

# **Заключение**

В результате выполнения работы был подробно изучен процесс функционирования клиент-серверных приложений. Стратегия построения REST API. Был частично изучен фреймворк Spring Boot, с помощью которого было легко реализовать REST API.

Также были применены знания полученные во время обучения в АлтГТУ, в числе которых PostgreSQL.

С помощью данных инструментов было спроектировано и реализовано клиент-серверное приложение, которое выполняет все поставленные задачи, в числе которых: отображение всех валют с сайта Центробанка РФ, а также получение и отображение графически выбранной валютой за период.

Для дальнейшего развития приложения можно реализовать репликацию базы данных с помощью технологии Debezium.

# **Список использованных источников**

1. Официальный сайт Центробанка. URL: [https://cbr.ru](https://cbr.ru/)

2. <https://java-course.ru/begin/xml/>

3. Документация Spring Boot.

URL: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>

4. Сайт Baeldung с обучающими материалами по Java. URL: [https://www.baeldung.com](https://www.baeldung.com/)

5. Документация по Android Studio. URL: <https://developer.android.com/>

6. Android Programming: The Big Nerd Ranch Guide: Учебное пособие / [Bryan Sills и др.] - 5-e изд., 2022. - 993 с : - ISBN-13 978-0137645732;

# **Приложение А. Тестирование программного обеспечения**

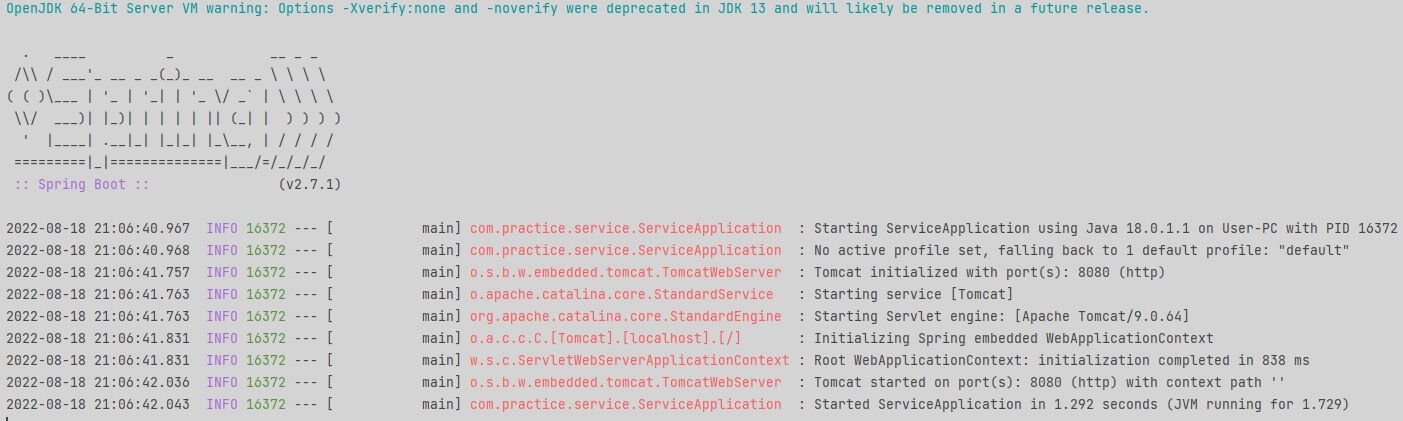
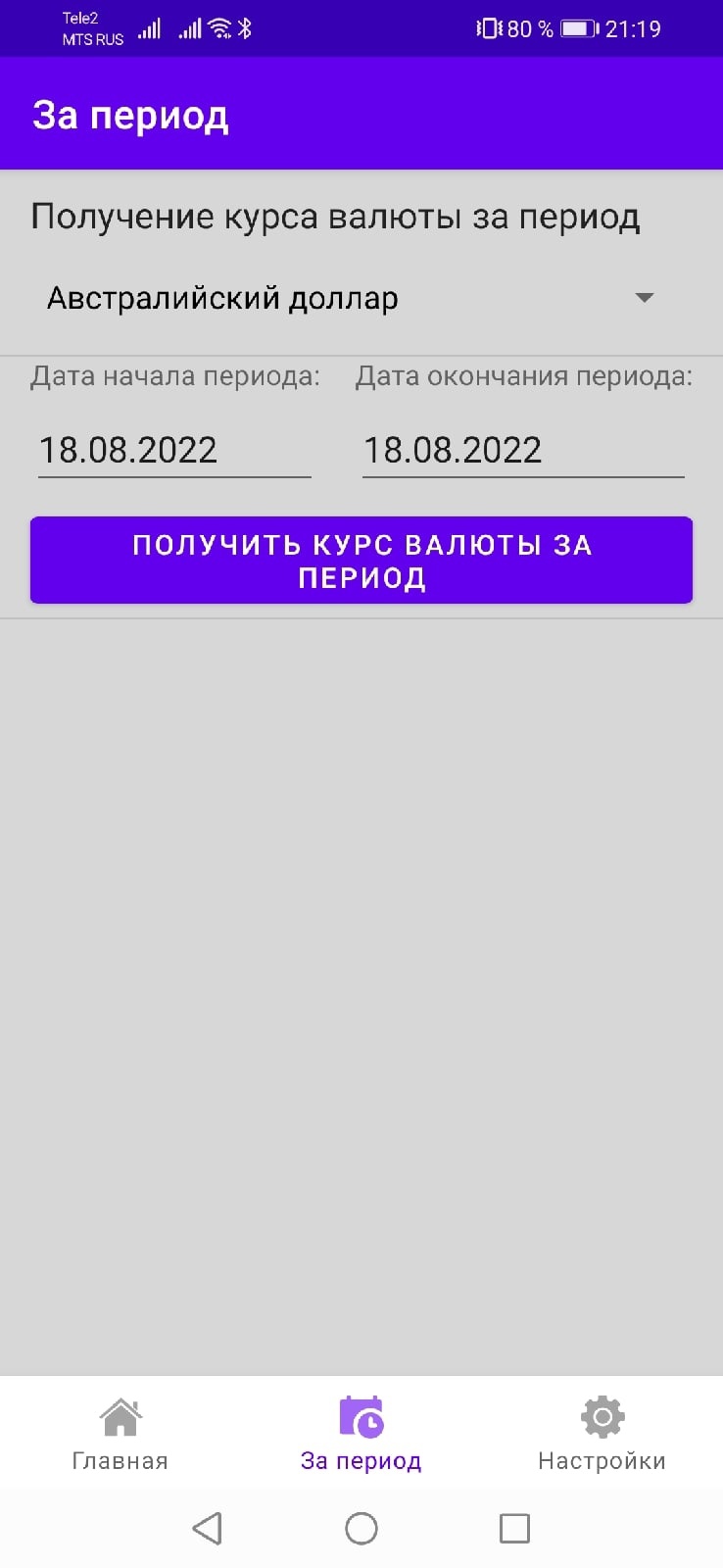
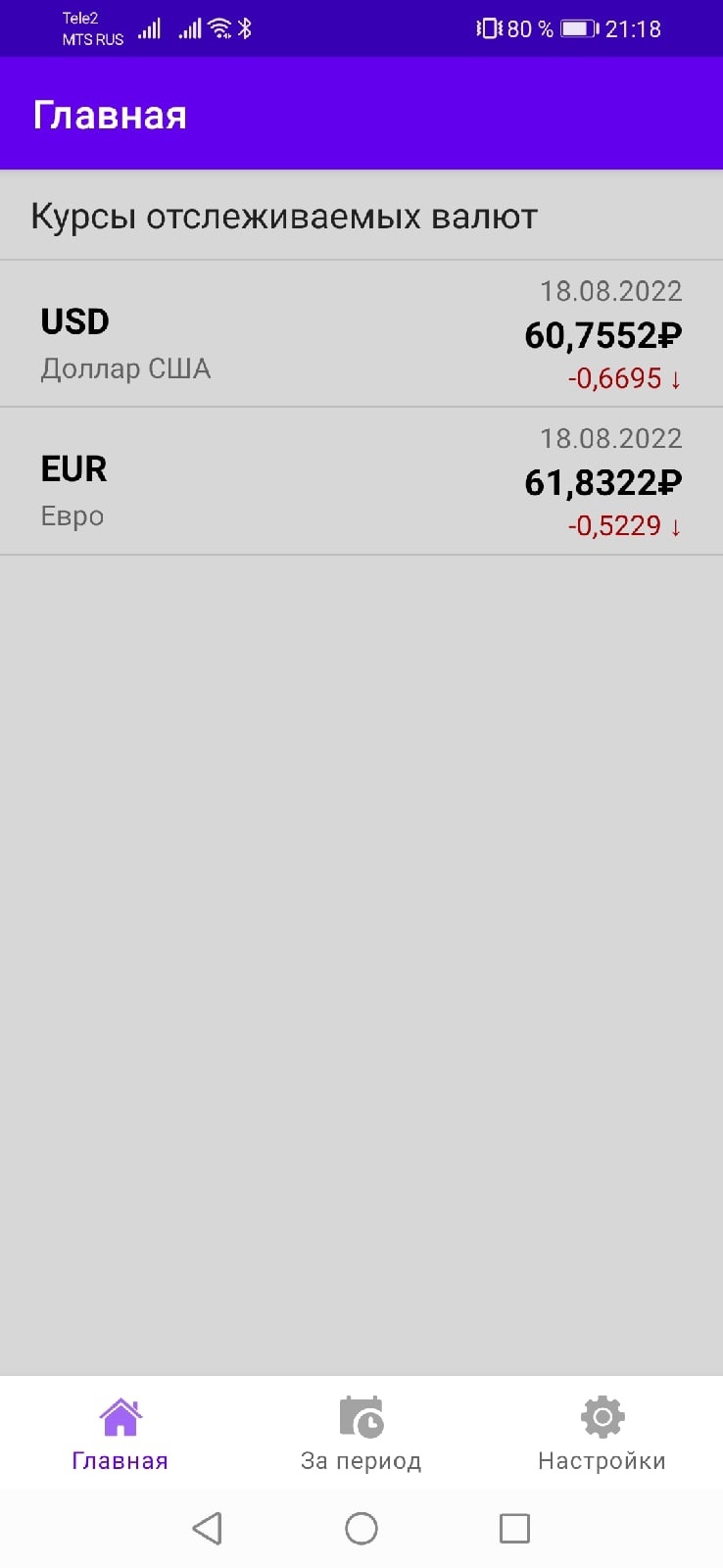
****

Рисунок 1 - Бэкенд-приложение на Spring запущено

Рисунок 2 — Главная страница Рисунок 3 — Страница «За период»  
Android-приложения

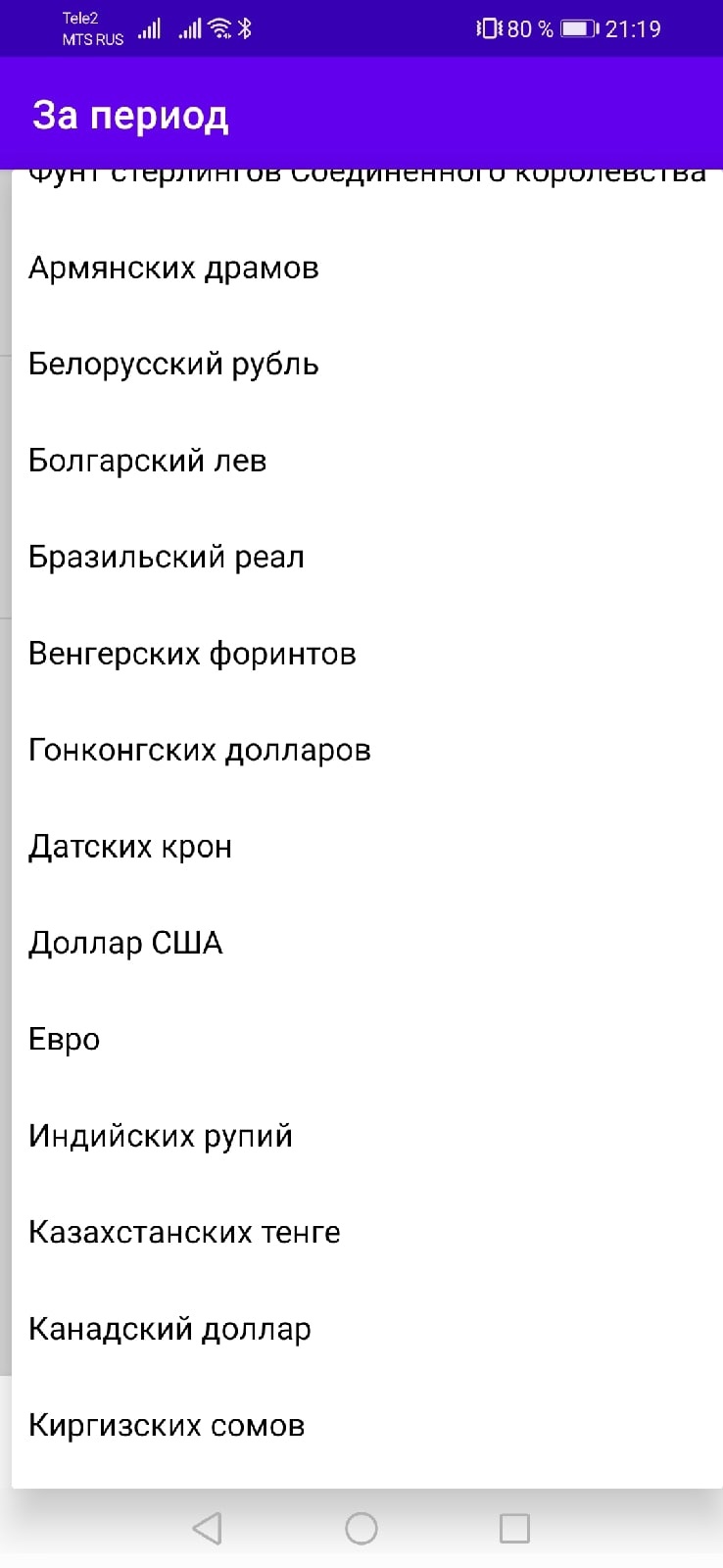
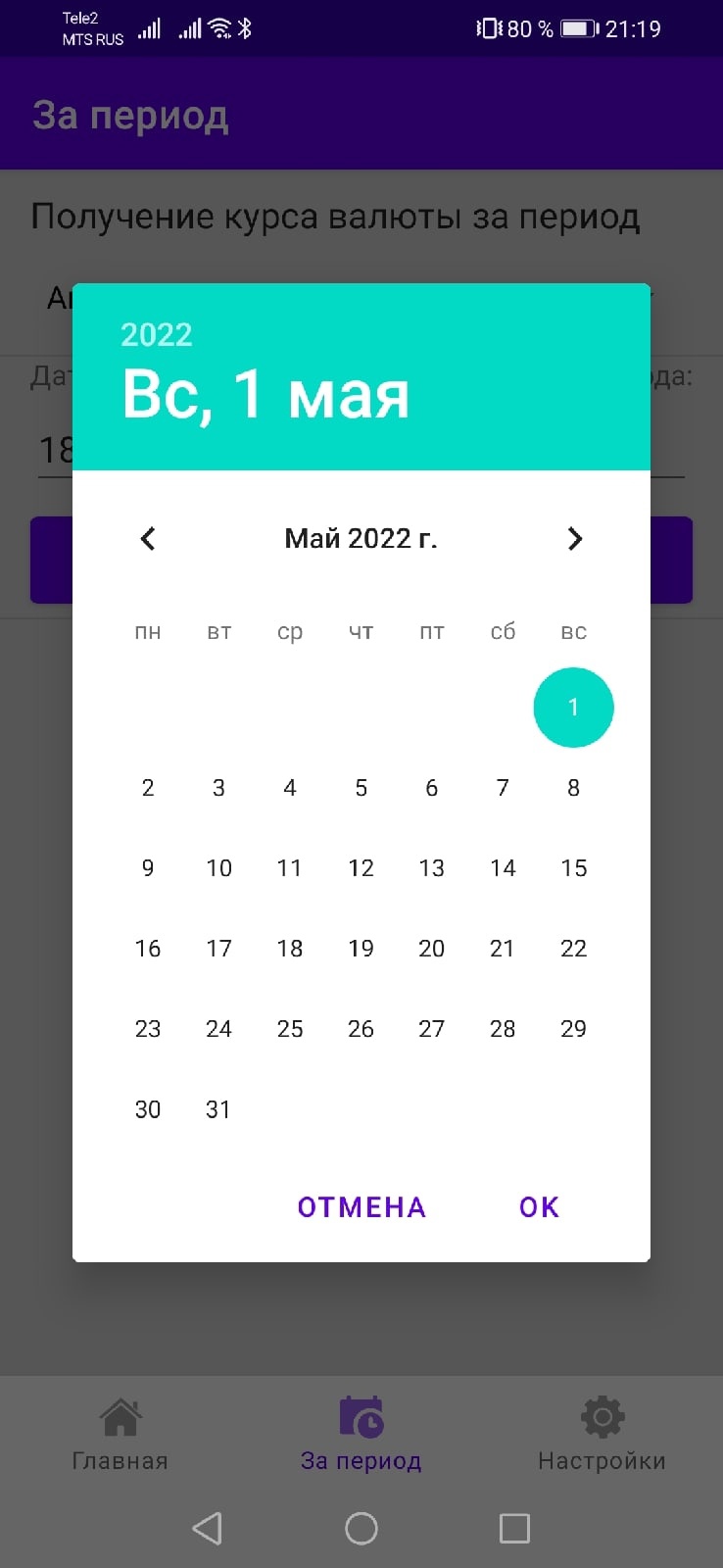
****

Рисунок 4 — Выбор даты для Рисунок 5 — Выбор валют, курс   
получения курса валюты за период которой за период нужно получить

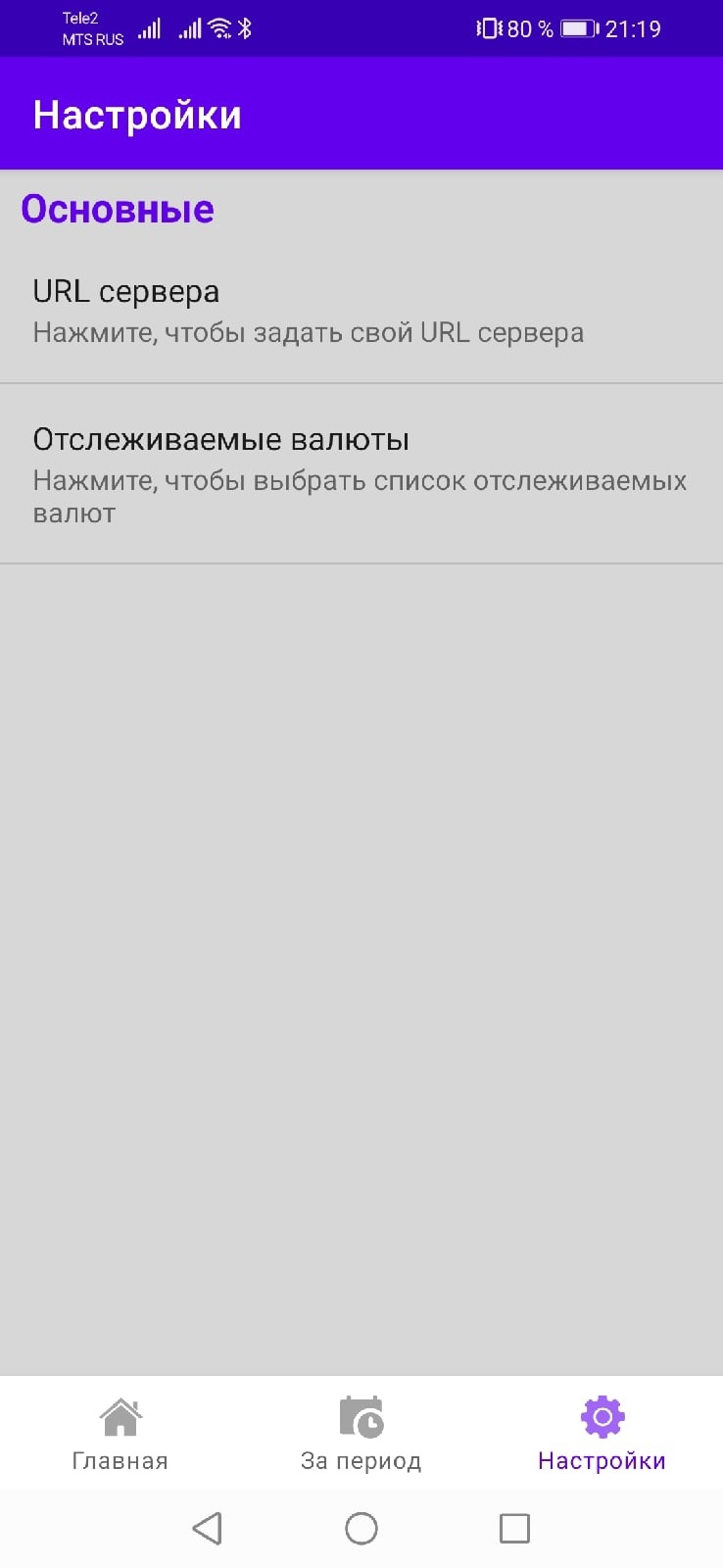
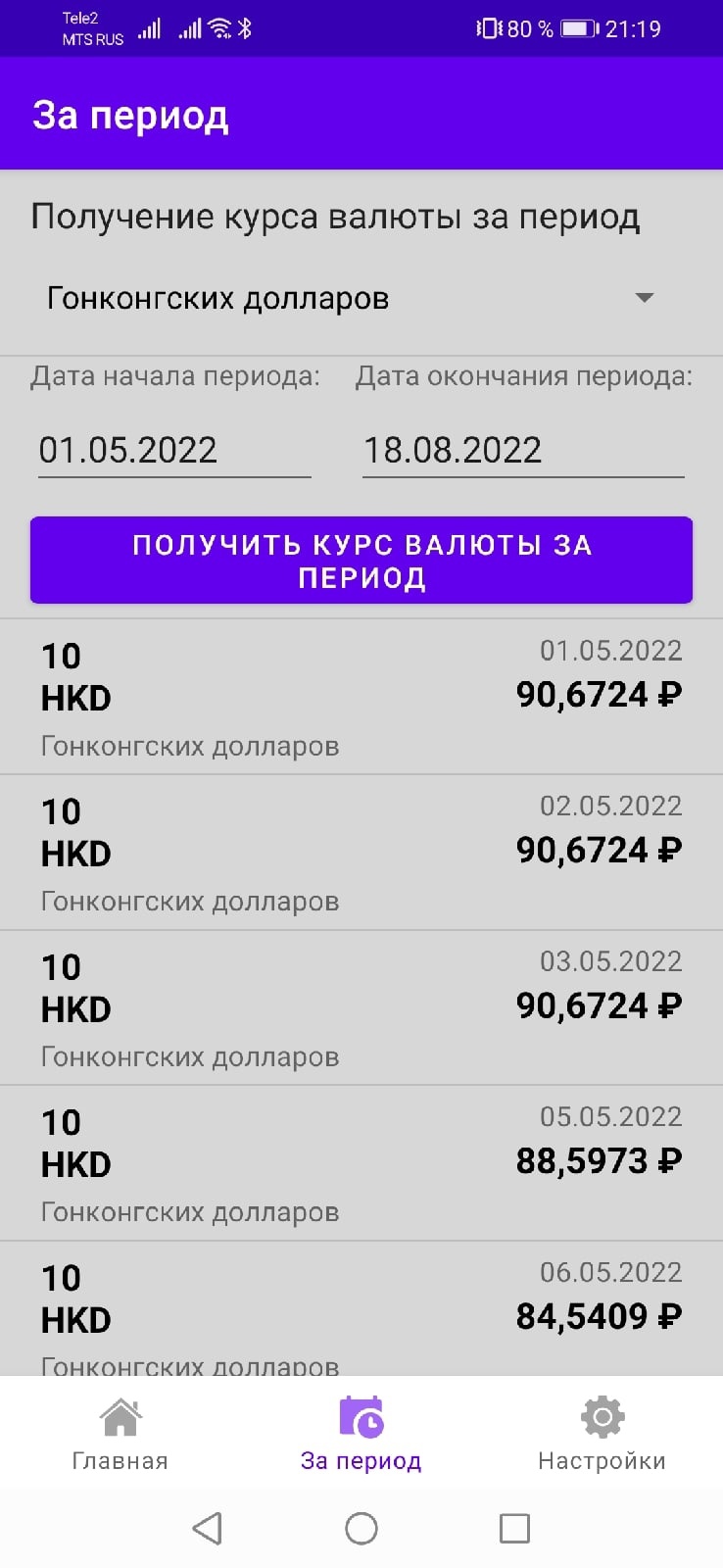


Рисунок 6 — Получение курса Рисунок 7 — Страница настроек  
Гонконгского доллара за период

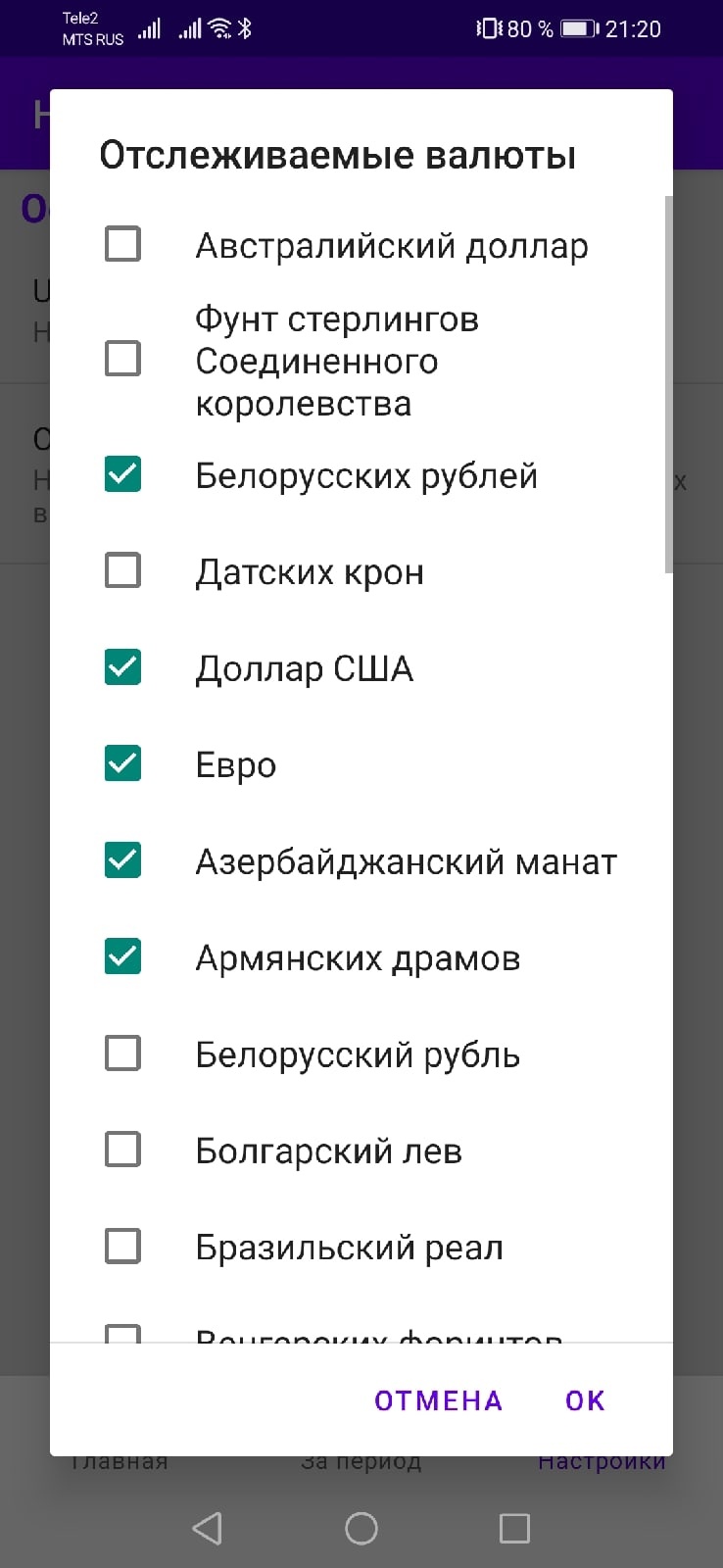
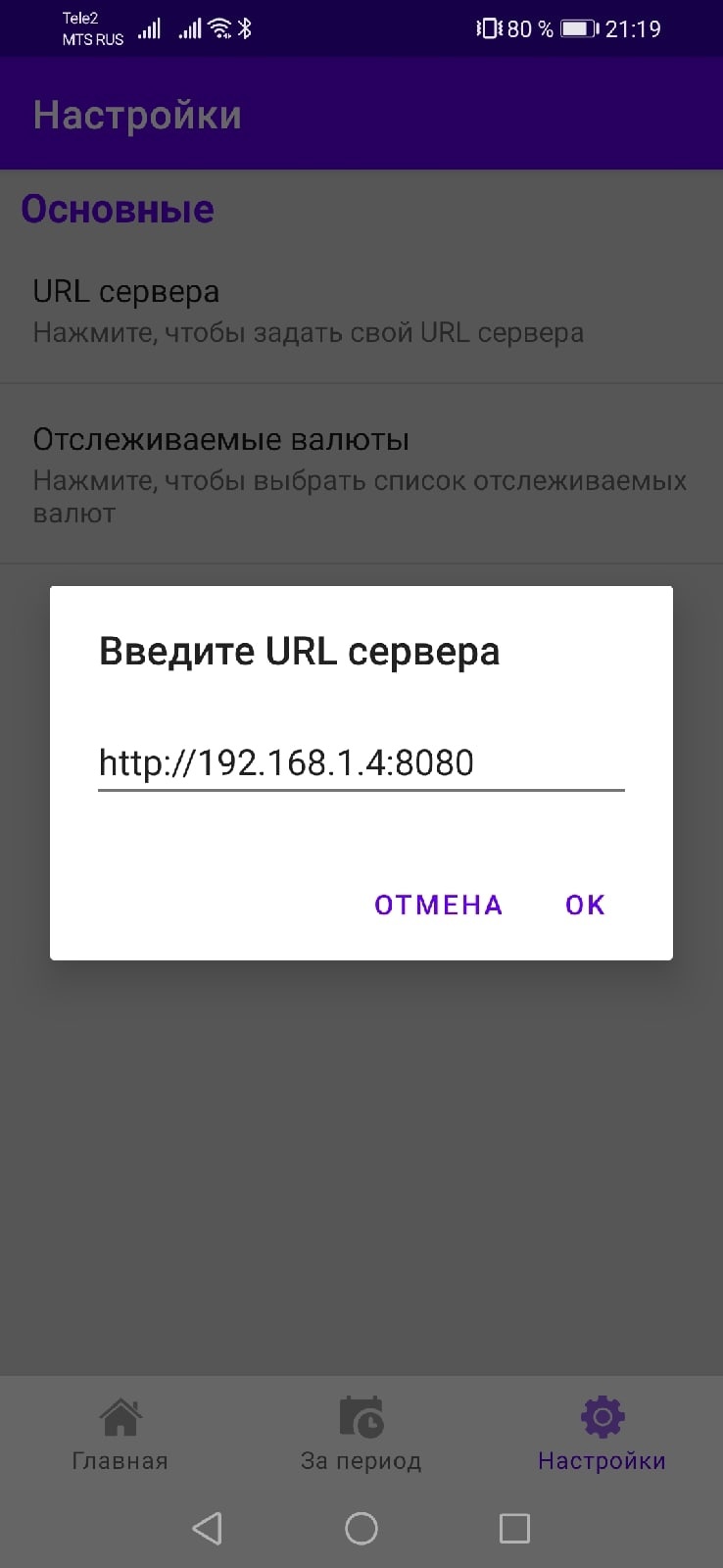


Рисунок 8 — Настройка Рисунок 9 — Выбор  
URL сервера отслеживаемых валют

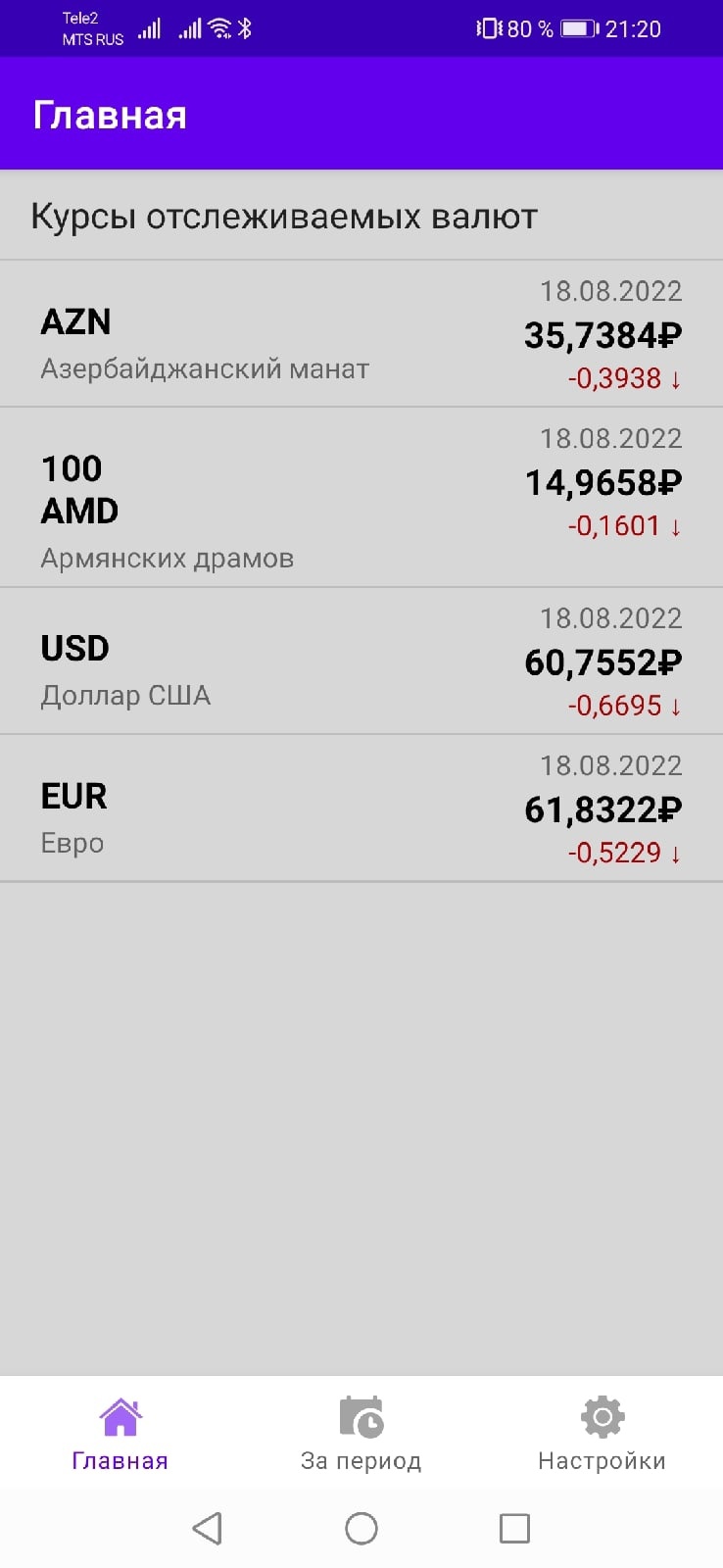
****

Рисунок 10 — Переход обратно Рисунок 11 — Ошибка загрузки   
на главную страницу — появились данных с сервера в случае, если   
новые отслеживаемые валюты URL сервера указан неправильно или сервер не запущен

# **Приложение Б. Исходный текст программы**

Исходный текст программы также расположен в репозитории GitHub:

<https://github.com/ShulV/sber-practice>

**Текст бэкенд-приложения на Spring**

*Код класса SpringConfig:*

package com.practice.service.config;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.\*;  
import org.springframework.core.env.Environment;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.jdbc.datasource.DriverManagerDataSource;  
import org.springframework.web.servlet.config.annotation.CorsRegistry;  
import org.springframework.web.servlet.config.annotation.EnableWebMvc;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
import javax.sql.DataSource;  
import java.util.Objects;  
  
@Configuration  
@ComponentScan("com.practice")  
@EnableWebMvc  
@PropertySource("classpath:database.properties")  
public class SpringConfig implements WebMvcConfigurer {  
 private final ApplicationContext applicationContext;  
 private final Environment environment;  
  
 @Autowired  
 public SpringConfig(ApplicationContext applicationContext, Environment environment) {  
 this.applicationContext = applicationContext;  
 this.environment = environment;  
 }  
  
 *// разрешение отравлять запросы с других адресов/портов (разрешение CORS)*  
@Bean  
 public WebMvcConfigurer corsConfigurer() {  
 return new WebMvcConfigurer() {  
 @Override  
 public void addCorsMappings(CorsRegistry registry) {  
 registry.addMapping("/\*\*")  
 .allowedMethods("GET", "POST", "PUT", "DELETE")  
 .allowedHeaders("\*")  
 .allowedOrigins("http://localhost:3000/");  
 }  
 };  
 }

@Bean  
 @Profile("!test")  
 public DataSource dataSource() {  
 DriverManagerDataSource dataSource = new DriverManagerDataSource();  
  
 dataSource.setDriverClassName(Objects.*requireNonNull*(environment.getProperty("driver")));  
 dataSource.setUrl(environment.getProperty("url"));  
 dataSource.setUsername(environment.getProperty("\_username"));  
 dataSource.setPassword(environment.getProperty("password"));  
  
 return dataSource;  
 }  
  
 @Bean  
 @Profile("!test")  
 public JdbcTemplate jdbcTemplate() {  
 return new JdbcTemplate(dataSource());  
 }  
}

*Код класса CurrencyController:*

package com.practice.service.controllers;  
import com.practice.service.model.CurrencyMenuItem;  
import com.practice.service.services.CurrencyService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.MediaType;  
import org.springframework.web.bind.annotation.\*;  
import java.io.IOException;  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/currency")  
public class CurrencyController {  
 private final CurrencyService currencyService;  
  
 @Autowired  
 public CurrencyController(CurrencyService currencyService) {  
 this.currencyService = currencyService;  
 }  
 *// Получение списка названия валют для раскрывающегося меню*  
@GetMapping(value = "/all-currency-designations", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public List<CurrencyMenuItem> getAllCurrencyDesignations() {  
 return currencyService.getAllCurrencyDesignations();  
 }

@GetMapping(value = "/testDB", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public String testDB() throws IOException {  
 currencyService.initDB();  
 return "testDB";  
 }  
}

*Код класса DayCurrencyController:*

package com.practice.service.controllers;  
import com.practice.service.model.DayCurrency;  
import com.practice.service.model.FullCurrencyInfo;  
import com.practice.service.services.DayCurrencyService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.MediaType;  
import org.springframework.web.bind.annotation.\*;  
import java.io.IOException;  
import java.sql.Date;  
import java.text.ParseException;  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/currency")  
public class DayCurrencyController {  
 private final DayCurrencyService dayCurrencyService;  
  
 @Autowired  
 public DayCurrencyController(DayCurrencyService dayCurrencyService) {  
 this.dayCurrencyService = dayCurrencyService;  
 }  
  
 @GetMapping(value = "/period-currencies", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public List<DayCurrency> getCurrenciesForPeriod(@RequestParam(name="fromDate") Date fromDate,  
 @RequestParam(name="toDate") Date toDate,  
 @RequestParam(name="charcode") String charcode) throws IOException, ParseException {  
 return dayCurrencyService.getPeriodCurrencies(fromDate, toDate, charcode);  
 }  
  
 @GetMapping(value = "/all-currencies-for-day", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public List<FullCurrencyInfo> getAllCurrenciesForDay(@RequestParam(name="date") Date date) throws IOException, ParseException {  
 return dayCurrencyService.getAllCurrenciesForDay(date);  
 }  
}

*Код класса CurrencyIdCharcodeMapper:*

package com.practice.service.dao.mappers;  
import org.springframework.jdbc.core.RowMapper;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
  
public class CurrencyIdCharcodeMapper implements RowMapper<String> {  
 @Override  
 public String mapRow(ResultSet rs, int rowNum) throws SQLException {  
 return rs.getString("PK\_id");  
 }  
}

*Код класса CurrencyMapper:*

package com.practice.service.dao.mappers;  
import com.practice.service.model.Currency;  
import org.springframework.jdbc.core.RowMapper;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
  
public class CurrencyMapper implements RowMapper<Currency> {  
 @Override  
 public Currency mapRow(ResultSet rs, int rowNum) throws SQLException {  
 Currency currency = new Currency();  
 currency.setPK\_Id(rs.getString("PK\_id"));  
 currency.setNumCode(rs.getInt("numcode"));  
 currency.setCharCode(rs.getString("charcode"));  
 currency.setName(rs.getString("name"));  
 return currency;  
 }

*Код класса CurrencyMenuMapper:*

package com.practice.service.dao.mappers;  
import com.practice.service.model.CurrencyMenuItem;  
import org.springframework.jdbc.core.RowMapper;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
  
public class CurrencyMenuMapper implements RowMapper<CurrencyMenuItem> {  
 @Override  
 public CurrencyMenuItem mapRow(ResultSet rs, int rowNum) throws SQLException {  
 return new CurrencyMenuItem(rs.getString("charcode"), rs.getString("name"));  
 }  
}

*Код класса FullCurrencyInfoMapper:*

package com.practice.service.dao.mappers;  
import com.practice.service.model.FullCurrencyInfo;  
import org.springframework.jdbc.core.RowMapper;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
  
public class FullCurrencyInfoMapper implements RowMapper<FullCurrencyInfo> {  
 @Override  
 public FullCurrencyInfo mapRow(ResultSet rs, int rowNum) throws SQLException {  
 FullCurrencyInfo fullCurrencyInfo = new FullCurrencyInfo();  
  
 fullCurrencyInfo.setValue(rs.getDouble("value"));  
 fullCurrencyInfo.setDate(rs.getDate("date"));  
 fullCurrencyInfo.setNominal(rs.getInt("nominal"));  
 fullCurrencyInfo.setCharcode(rs.getString("charcode"));  
 fullCurrencyInfo.setName(rs.getString("name"));  
 return fullCurrencyInfo;  
 }  
}

*Код класса CurrencyDAO:*

package com.practice.service.dao;  
import com.practice.service.dao.mappers.CurrencyIdCharcodeMapper;  
import com.practice.service.dao.mappers.CurrencyMapper;  
import com.practice.service.dao.mappers.CurrencyMenuMapper;  
import com.practice.service.model.Currency;  
import com.practice.service.model.CurrencyMenuItem;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.jdbc.core.BatchPreparedStatementSetter;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Component;  
import java.sql.PreparedStatement;  
import java.sql.SQLException;  
import java.util.List;  
  
@Component  
public class CurrencyDAO {  
 private final JdbcTemplate jdbcTemplate;  
  
 @Autowired  
 public CurrencyDAO(JdbcTemplate jdbcTemplate) {  
 this.jdbcTemplate = jdbcTemplate;  
 }  
  
 public List<Currency> getAll() {  
 return jdbcTemplate.query("SELECT \* FROM \"Currency\"", new CurrencyMapper());  
 }  
 *// Получение списка названия валют для раскрывающегося меню*  
public List<CurrencyMenuItem> getAllCurrencyDesignations() {  
 return jdbcTemplate.query("SELECT charcode, name FROM \"Currency\"", new CurrencyMenuMapper());  
 }

public void batchCurrencyUpdate(List<Currency> currencyList) {  
 jdbcTemplate.batchUpdate("INSERT INTO \"Currency\" (\"PK\_id\", \"numcode\", \"charcode\", \"name\") " +  
 "VALUES (?, ?, ?, ?) ON CONFLICT DO NOTHING",  
 new BatchPreparedStatementSetter() {  
 @Override  
 public void setValues(PreparedStatement ps, int i) throws SQLException {  
 ps.setString(1, currencyList.get(i).getPK\_Id());  
 ps.setInt(2, currencyList.get(i).getNumCode());  
 ps.setString(3, currencyList.get(i).getCharCode());  
 ps.setString(4, currencyList.get(i).getName());  
 }  
 @Override  
 public int getBatchSize() {  
 return currencyList.size();  
 }  
 });  
 }  
 public String getIdByCharcode(String charcode) {  
 return jdbcTemplate.query("SELECT \* FROM \"Currency\" WHERE \"charcode\" =?",  
 new Object[]{charcode}, new CurrencyIdCharcodeMapper()).get(0);  
 }  
  
}

*Код класса DayCurrencyDAO:*

package com.practice.service.dao;  
import com.practice.service.dao.mappers.FullCurrencyInfoMapper;  
import com.practice.service.model.DayCurrency;  
import com.practice.service.model.FullCurrencyInfo;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.jdbc.core.BatchPreparedStatementSetter;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Component;  
import java.sql.PreparedStatement;  
import java.sql.SQLException;  
import java.sql.Date;  
import java.util.List;  
  
@Component  
public class DayCurrencyDAO {  
 private final JdbcTemplate jdbcTemplate;  
  
 @Autowired  
 public DayCurrencyDAO(JdbcTemplate jdbcTemplate) {  
 this.jdbcTemplate = jdbcTemplate;  
 }  
  
 public List<DayCurrency> getAll() {  
 return jdbcTemplate.query("SELECT \* FROM \"DayCurrency\"",  
 new BeanPropertyRowMapper<>(DayCurrency.class));  
 }  
  
 public List<DayCurrency> getPeriodCurrencies(Date fromDate, Date toDate, String charcode) {  
  
 return jdbcTemplate.query("SELECT \"PK\_daycur\", \"value\", \"date\", \"nominal\", dc.\"PK\_id\" " +  
 "FROM \"DayCurrency\" as dc " +  
 "join \"Currency\" as c on c.\"PK\_id\" = dc.\"PK\_id\" " +  
 "where \"charcode\" = ? and " +  
 "\"date\" between ? and ? " +  
 "ORDER BY \"date\";",  
 new BeanPropertyRowMapper<>(DayCurrency.class),  
 new Object[]{charcode, fromDate, toDate}).stream().toList();  
  
 }  
  
 public void insert(DayCurrency dayCurrency, String currencyName) {  
 Double value = dayCurrency.getValue();  
 Date date = dayCurrency.getDate();  
 int nominal = dayCurrency.getNominal();  
 String pkId = dayCurrency.getPK\_id();  
  
 jdbcTemplate.update("INSERT INTO \"DayCurrency\"(\"value\", \"date\", \"nominal\", \"PK\_id\") " +  
 "VALUES (?, ?, ?, (SELECT \"PK\_id\" FROM public.\"Currency\" " +  
 "where \"name\"=?));",  
 value, date, nominal, currencyName);  
 }

public void batchDayCurrencyInsert(List<DayCurrency> dayCurrencyList, String charcode) {  
 jdbcTemplate.batchUpdate("INSERT INTO \"DayCurrency\"(\"value\", \"date\", \"nominal\", \"PK\_id\") " +  
 "VALUES (?, ?, ?, " +  
 "(SELECT \"PK\_id\" " +  
 "FROM \"Currency\" where \"charcode\"=?));",  
 new BatchPreparedStatementSetter() {  
 @Override  
 public void setValues(PreparedStatement ps, int i) throws SQLException {  
 ps.setDouble(1, dayCurrencyList.get(i).getValue());  
 ps.setDate(2, dayCurrencyList.get(i).getDate());  
 ps.setInt(3, dayCurrencyList.get(i).getNominal());  
 ps.setString(4, charcode);  
 }  
 @Override  
 public int getBatchSize() {  
 return dayCurrencyList.size();  
 }  
 });  
 }  
  
 public void deleteForDate(Date date) {  
 jdbcTemplate.update("DELETE FROM public.\"DayCurrency\"" +  
 "WHERE \"date\"= ?;", date);  
 }  
  
  
 public List<FullCurrencyInfo> getAllCurrenciesForDay(java.util.Date date) {  
 return jdbcTemplate.query("SELECT \"value\", \"date\", \"nominal\", c.\"charcode\", c.\"name\"" +  
 "FROM \"DayCurrency\" as dc " +  
 "join \"Currency\" as c on c.\"PK\_id\" = dc.\"PK\_id\" " +  
 "where \"date\" = ? " +  
 "ORDER BY c.\"name\"",  
 new FullCurrencyInfoMapper(),  
 new Object[]{date}).stream().toList();  
 }

public void insertForDate(List<DayCurrency> dayCurrencyList) {  
 jdbcTemplate.batchUpdate("INSERT INTO \"DayCurrency\"(\"value\", \"date\", \"nominal\", \"PK\_id\") " +  
 "VALUES (?, ?, ?, ?);",  
 new BatchPreparedStatementSetter() {  
 @Override  
 public void setValues(PreparedStatement ps, int i) throws SQLException {  
 ps.setDouble(1, dayCurrencyList.get(i).getValue());  
 ps.setDate(2, dayCurrencyList.get(i).getDate());  
 ps.setInt(3, dayCurrencyList.get(i).getNominal());  
 ps.setString(4, dayCurrencyList.get(i).getPK\_id());  
 }  
 @Override  
 public int getBatchSize() {  
 return dayCurrencyList.size();  
 }  
 });  
 }  
}

*Код класса Currency:*

package com.practice.service.model;  
  
public class Currency {  
 private String PK\_id;  
 private int numCode;  
 private String charCode;  
  
 private String name;  
 public Currency() {  
 }  
  
 public Currency(String PK\_id,int numCode, String charCode, String name) {  
 this.PK\_id = PK\_id;  
 this.numCode = numCode;  
 this.charCode = charCode;  
 this.name = name;  
 }  
  
 public String getPK\_Id() {  
 return PK\_id;  
 }  
  
 public void setPK\_Id(String PK\_id) { this.PK\_id = PK\_id; }  
  
 public int getNumCode() {  
 return numCode;  
 }  
  
 public void setNumCode(int numCode) { this.numCode = numCode; }  
  
 public String getCharCode() {  
 return charCode;  
 }  
  
 public void setCharCode(String charCode) {  
 this.charCode = charCode;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String toString() {  
 return "Currency{" +  
 "PK\_id='" + PK\_id + '\'' +  
 ", numCode='" + numCode + '\'' +  
 ", charCode='" + charCode + '\'' +  
 ", name='" + name + '\'' +  
 '}';  
 }  
}

*Код класса CurrencyMenuItem:*

package com.practice.service.model;  
  
public class CurrencyMenuItem {  
 private String charCode;  
 private String name;  
  
 public CurrencyMenuItem(String charCode, String name) {  
 this.charCode = charCode;  
 this.name = name;  
 }  
  
 public String getCharCode() {  
 return charCode;  
 }  
  
 public void setCharCode(String charCode) {  
 this.charCode = charCode;  
 }  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

*Код класса DayCurrency:*

package com.practice.service.model;  
import java.sql.Date;  
  
public class DayCurrency implements Cloneable {  
 private int PK\_daycur;  
 private double value;  
 private Date date;  
 private int nominal;  
 private String PK\_id;  
  
 public DayCurrency() {  
  
 }  
  
 public DayCurrency(int PK\_daycur, double value, Date date, int nominal, String PK\_id) {  
 this.PK\_daycur = PK\_daycur;  
 this.value = value;  
 this.date = date;  
 this.nominal = nominal;  
 this.PK\_id = PK\_id;  
 }  
  
 public DayCurrency(double value, Date date, int nominal) {  
 this.value = value;  
 this.date = date;  
 this.nominal = nominal;  
 }  
  
 public DayCurrency(double value, Date date, int nominal, String PK\_id) {  
 this.value = value;  
 this.date = date;  
 this.nominal = nominal;  
 this.PK\_id = PK\_id;  
 }  
  
 public int getPK\_daycur() {  
 return PK\_daycur;  
 }  
  
 public void setPK\_daycur(int PK\_daycur) {  
 this.PK\_daycur = PK\_daycur;  
 }  
  
 public double getValue() {  
 return value;  
 }  
  
 public void setValue(double value) {  
 this.value = value;  
 }  
  
 public Date getDate() {  
 return date;  
 }  
  
 public void setDate(Date date) {  
 this.date = date;  
 }  
  
 public int getNominal() {  
 return nominal;  
 }  
  
 public void setNominal(int nominal) {  
 this.nominal = nominal;  
 }  
  
 public String getPK\_id() {  
 return PK\_id;  
 }  
  
 public void setPK\_id(String PK\_id) {  
 this.PK\_id = PK\_id;  
 }  
  
 @Override  
 public String toString() {  
 return "DayCurrency{" +  
 "PK\_daycur=" + PK\_daycur +  
 ", value=" + value +  
 ", date=" + date +  
 ", nominal=" + nominal +  
 ", PK\_id='" + PK\_id + '\'' +  
 '}';  
 }

@Override  
 public DayCurrency clone() {  
 try {  
 *// Глубокое копирование*  
DayCurrency clone = (DayCurrency) super.clone();  
 Date date = (Date) this.getDate().clone();  
 clone.setDate(date);  
 return clone;  
 } catch (CloneNotSupportedException e) {  
 throw new AssertionError();  
 }  
 }  
}

*Код класса FullCurrencyInfo:*

package com.practice.service.model;  
import java.sql.Date;  
  
public class FullCurrencyInfo {  
 private double value;  
 private Date date;  
 private int nominal;  
 private String charcode;  
 private String name;  
  
 public double getValue() {  
 return value;  
 }  
  
 public void setValue(double value) {  
 this.value = value;  
 }  
  
 public Date getDate() {  
 return date;  
 }  
  
 public void setDate(Date date) {  
 this.date = date;  
 }  
  
 public int getNominal() {  
 return nominal;  
 }

public void setNominal(int nominal) {  
 this.nominal = nominal;  
 }  
  
 public String getCharcode() {  
 return charcode;  
 }  
  
 public void setCharcode(String charcode) {  
 this.charcode = charcode;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public FullCurrencyInfo(double value, Date date, int nominal, String charcode, String name) {  
 this.value = value;  
 this.date = date;  
 this.nominal = nominal;  
 this.charcode = charcode;  
 this.name = name;  
 }  
  
 public FullCurrencyInfo() {  
  
 }  
   
 @Override  
 public String toString() {  
 return "FullCurrencyInfo{" +  
 "value=" + value +  
 ", date=" + date +  
 ", nominal=" + nominal +  
 ", charcode='" + charcode + '\'' +  
 ", name='" + name + '\'' +  
 '}';  
 }  
}

*Код класса XMLParser:*

package com.practice.service.parser;  
import com.practice.service.model.Currency;  
import com.practice.service.model.DayCurrency;  
import org.jsoup.Jsoup;  
import org.jsoup.nodes.Document;  
import org.jsoup.nodes.Element;  
import org.springframework.stereotype.Component;  
import java.io.IOException;  
import java.text.DateFormat;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.ArrayList;  
import java.sql.Date;  
import java.util.List;  
  
import static java.lang.Integer.*parseInt*;  
  
@Component  
public class XMLParser {  
  
 DateFormat fromFormat = new SimpleDateFormat("dd.MM.yyyy");  
 DateFormat myFormat = new SimpleDateFormat("yyyy-MM-dd");  
 DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");  
  
 public List<DayCurrency> xmlDailyValutes(Date date) {  
 List<DayCurrency> dayCurrencyList = new ArrayList<>();  
  
 List<String> IDList = new ArrayList<>();  
 List<Double> valueList = new ArrayList<>();  
 List<Integer> nominalList = new ArrayList<>();  
  
 Date todayDate;  
  
 String xml = "http://www.cbr.ru/scripts/XML\_daily.asp?date\_req=" + dateFormat.format(date);  
 Document doc;  
 try {  
 doc = Jsoup  
 .*connect*(xml)  
 .userAgent("Chrome/4.0.249.0 Safari/532.5")  
 .referrer("http://www.google.com")  
 .get();  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
  
 try {  
 todayDate = Date.*valueOf*(  
 myFormat.format(  
 fromFormat.parse(doc.select("ValCurs")  
 .attr("Date"))));  
 } catch (ParseException e) {  
 throw new RuntimeException(e);  
 }  
  
 for (Element e : doc.select("Valute")) {  
 IDList.add(e.attr("ID"));  
 }  
  
 for (Element e : doc.select("Value")) {  
 valueList.add(Double.*parseDouble*((e.text().replace(",", "."))));  
 }  
  
 for (Element e : doc.select("Nominal")) {  
 nominalList.add(*parseInt*(e.text()));  
 }  
  
 for(int i = 0; i < IDList.size(); i++){  
 dayCurrencyList.add(new DayCurrency(  
 valueList.get(i),  
 todayDate,  
 nominalList.get(i),  
 IDList.get(i)));  
 }  
 return dayCurrencyList;  
 }  
  
 public List<DayCurrency> xmlConnectPeriod(Date startDate,  
 Date endDate, String currencyID) throws IOException, ParseException {  
 List<DayCurrency> dayCurrencyList = new ArrayList<>();  
  
 List<Double> valueList = new ArrayList<>();  
 List<Integer> nominalList = new ArrayList<>();  
 List<Date> dateList = new ArrayList<>();  
  
 String xml = "https://www.cbr.ru/scripts/XML\_dynamic.asp?date\_req1=" + dateFormat.format(startDate)  
 + "&date\_req2=" + dateFormat.format(endDate)+"&VAL\_NM\_RQ=" + currencyID;  
 Document doc = Jsoup  
 .*connect*(xml)  
 .userAgent("Chrome/4.0.249.0 Safari/532.5")  
 .referrer("http://www.google.com")  
 .get();  
 doc = Jsoup.*parse*(String.*valueOf*(doc));  
  
 for (Element e : doc.select("Value")) {  
 valueList.add(Double.*parseDouble*((e.text().replace(",","."))));  
 }  
  
 for (Element e : doc.select("Nominal")) {  
 nominalList.add(*parseInt*(e.text()));  
 }  
  
 for (Element e : doc.select("Record")) {  
 dateList.add(Date.*valueOf*(myFormat.format(fromFormat.parse(e.attr("Date")))));  
 }  
  
 for(int i = 0; i < nominalList.size(); i++){  
 dayCurrencyList.add(new DayCurrency(  
 valueList.get(i),  
 dateList.get(i),  
 nominalList.get(i)  
 ));  
 }  
 return dayCurrencyList;  
 }

public List<Currency> xmlInitializeCurrency() throws IOException {  
 List<Currency> currencyList = new ArrayList<>();  
  
 List<String> IDList = new ArrayList<>();  
 List<Integer> numCodeList = new ArrayList<>();  
 List<String> charCodeList = new ArrayList<>();  
 List<String> nameList = new ArrayList<>();  
  
 String xml = "https://www.cbr.ru/scripts/XML\_daily.asp";  
 Document doc = Jsoup  
 .*connect*(xml)  
 .userAgent("Chrome/4.0.249.0 Safari/532.5")  
 .referrer("http://www.google.com")  
 .get();  
  
 for (Element e : doc.select("Valute")) {  
 IDList.add(e.attr("ID"));  
 }  
  
 for (Element e : doc.select("NumCode")) {  
 numCodeList.add(*parseInt*(e.text()));  
 }  
  
 for (Element e : doc.select("CharCode")) {  
 charCodeList.add(e.text());  
 }  
  
 for (Element e : doc.select("Name")) {  
 nameList.add(e.text());  
 }  
  
 for(int i = 0; i < nameList.size(); i++){  
 currencyList.add(new Currency(  
 IDList.get(i),  
 numCodeList.get(i),  
 charCodeList.get(i),  
 nameList.get(i)));  
 }  
 return currencyList;  
 }  
}

*Код класса CurrencyService:*

package com.practice.service.services;  
import com.practice.service.dao.CurrencyDAO;  
import com.practice.service.model.Currency;  
import com.practice.service.model.CurrencyMenuItem;  
import com.practice.service.parser.XMLParser;  
import org.springframework.stereotype.Component;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
@Component  
public class CurrencyService {  
 private final CurrencyDAO currencyDAO;  
 private final XMLParser xmlParser;  
 public CurrencyService(CurrencyDAO currencyDAO, XMLParser xmlParser) {  
 this.currencyDAO = currencyDAO;  
 this.xmlParser = xmlParser;  
 }  
 public List<CurrencyMenuItem> getAllCurrencyDesignations() {  
 List<CurrencyMenuItem> currencyListFromDB = new ArrayList<>();  
 List<Currency> currencyListFromParser;  
 try {  
 currencyListFromDB = currencyDAO.getAllCurrencyDesignations();  
 currencyListFromParser = xmlParser.xmlInitializeCurrency();  
  
 if (currencyListFromDB.size() != currencyListFromParser.size()) {  
 currencyDAO.batchCurrencyUpdate(currencyListFromParser);  
 currencyListFromDB = currencyDAO.getAllCurrencyDesignations();  
 }  
 }  
 catch (IOException e) {  
 System.*out*.println(Arrays.*toString*(e.getStackTrace()));  
 }  
 return currencyListFromDB;  
 }

public void initDB() throws IOException {  
 List<Currency> currencyListDB = currencyDAO.getAll();  
 List<Currency> currencyListFromParser = xmlParser.xmlInitializeCurrency();  
 if (currencyListDB.size() == currencyListFromParser.size()) {  
 System.*out*.println("equal");  
 System.*out*.println("Parser" + (currencyListFromParser.size()));  
 System.*out*.println("DB" + (currencyListDB.size()));  
 } else {  
 System.*out*.println("Parser" + (currencyListFromParser.size()));  
 System.*out*.println("DB" + (currencyListDB.size()));  
 currencyDAO.batchCurrencyUpdate(currencyListFromParser);  
 }  
 }  
}

*Код класса DayCurrencyService:*

package com.practice.service.services;  
import com.practice.service.dao.CurrencyDAO;  
import com.practice.service.dao.DayCurrencyDAO;  
import com.practice.service.model.DayCurrency;  
import com.practice.service.model.FullCurrencyInfo;  
import com.practice.service.parser.XMLParser;  
import org.springframework.context.annotation.PropertySource;  
import org.springframework.core.env.Environment;  
import org.springframework.stereotype.Component;  
import java.io.IOException;  
import java.sql.Date;  
import java.text.ParseException;  
import java.util.\*;  
  
@Component  
@PropertySource("classpath:application.properties")  
public class DayCurrencyService {  
 private final DayCurrencyDAO dayCurrencyDAO;  
 private final CurrencyDAO currencyDAO;  
 private final XMLParser xmlParser;  
 private final Environment environment;  
 public DayCurrencyService(DayCurrencyDAO dayCurrencyDAO, CurrencyDAO currencyDAO, XMLParser xmlParser, Environment environment) {  
 this.dayCurrencyDAO = dayCurrencyDAO;  
 this.currencyDAO = currencyDAO;  
 this.xmlParser = xmlParser;  
 this.environment = environment;  
 }

public List<DayCurrency> getPeriodCurrencies(Date fromDate, Date toDate, String charcode) throws IOException, ParseException {  
 String curId = currencyDAO.getIdByCharcode(charcode);  
 List<DayCurrency> periodCurrencies = dayCurrencyDAO.getPeriodCurrencies(fromDate, toDate, charcode);  
 List<DayCurrency> dayCurrencyList;  
 if (periodCurrencies.size() > 0) {  
 List<Date> missingDateList = getMissingDates(fromDate, toDate, periodCurrencies);  
 if (missingDateList.size() == 0) {  
 return periodCurrencies;  
 }  
 missingDateList.sort(java.util.Date::compareTo);  
  
 Date minDate = missingDateList.get(0);  
 Date maxDate = missingDateList.get(missingDateList.size() - 1);  
  
 dayCurrencyList = xmlParser.xmlConnectPeriod(minDate, maxDate, curId);  
 if (dayCurrencyList.isEmpty()) {  
 dayCurrencyList = fillEmptyList(dayCurrencyList, minDate, maxDate, curId);  
 }  
 else {  
 fillInEmptyLines(minDate, maxDate, dayCurrencyList, curId);  
 }  
  
 dayCurrencyList.removeIf(dc -> !missingDateList.contains(dc.getDate()));  
 }  
 else {  
 dayCurrencyList = xmlParser.xmlConnectPeriod(fromDate, toDate, curId);  
 if (dayCurrencyList.isEmpty()) {  
 dayCurrencyList = fillEmptyList(dayCurrencyList, fromDate, toDate, curId);  
 }  
 else {  
 fillInEmptyLines(fromDate, toDate, dayCurrencyList, curId);  
 }  
 }  
 dayCurrencyDAO.batchDayCurrencyInsert(dayCurrencyList, charcode);  
 return dayCurrencyDAO.getPeriodCurrencies(fromDate, toDate, charcode);  
 }  
 public void insert(DayCurrency dayCurrency, String currencyName) {  
 dayCurrencyDAO.insert(dayCurrency, currencyName);  
 }

private void fillInEmptyLines(Date fromDate, Date toDate,  
 List<DayCurrency> dayCurrencyList,  
 String currencyId) throws IOException, ParseException {  
 int MS\_IN\_DAY = Integer.*parseInt*(Objects.*requireNonNull*(environment.getProperty("time.MS\_IN\_DAY")));  
 *// Начальная дата запрашиваемого периода.*  
Date startDate = (Date) fromDate.clone();  
 *// Конечная дата запрашиваемого периода.*  
Date endDate = (Date) toDate.clone();  
 *// Коллекция недостающих незаполненных дней.*  
List<DayCurrency> newDayCurrencyList = new ArrayList<>();  
 *// Предыдущий объект для инициализации незаполненных дней.*  
if (!Objects.*equals*(dayCurrencyList.get(0).getDate().toString(), startDate.toString())) {  
 List<DayCurrency> startMissingDates = new ArrayList<>();  
 Date lastMissingDate = (Date) dayCurrencyList.get(0).getDate().clone();  
 lastMissingDate.setTime(lastMissingDate.getTime() - MS\_IN\_DAY);  
 startMissingDates = fillEmptyList(startMissingDates, fromDate,  
 lastMissingDate, currencyId);  
 newDayCurrencyList.addAll(0, startMissingDates);  
 startDate = (Date) dayCurrencyList.get(0).getDate().clone();  
 }  
 DayCurrency prevDayCurrency = dayCurrencyList.get(0).clone();  
  
 int elemNum = 0;  
 *// Проходим по валютам для каждого дня из периода*  
while (startDate.compareTo(endDate) <= 0) {  
 *// Если данных для текущей даты нет в коллекции, копируем предыдущий dayCurrencyList в текущий*  
if (elemNum > dayCurrencyList.size() - 1 ||  
 !Objects.*equals*(dayCurrencyList.get(elemNum).getDate().toString(), startDate.toString())) {  
 prevDayCurrency.setDate((Date) startDate.clone());  
 newDayCurrencyList.add(prevDayCurrency.clone());  
 } else {  
 prevDayCurrency = dayCurrencyList.get(elemNum).clone();  
 elemNum++;  
 }  
 startDate.setTime(startDate.getTime() + MS\_IN\_DAY);  
 startDate = *removeTime*(startDate);  
 }  
 dayCurrencyList.addAll(newDayCurrencyList);*// Объединение коллекций*  
}

private List<DayCurrency> fillEmptyList(List<DayCurrency> dayCurrencyList,  
 Date fromDate, Date toDate,  
 String currencyId ) throws IOException, ParseException {  
 int MS\_IN\_DAY = Integer.*parseInt*(Objects.*requireNonNull*(environment.getProperty("time.MS\_IN\_DAY")));  
 *// Начальная дата запрашиваемого периода.*  
Date startDate = (Date) fromDate.clone();  
 *// Конечная дата запрашиваемого периода*  
Date endDate = (Date) toDate.clone();  
 while (dayCurrencyList.isEmpty()) {  
 *// Расширение периода влево на 10 дней (в прошлое).*  
startDate.setTime(startDate.getTime() - 10L \* MS\_IN\_DAY);  
 dayCurrencyList = xmlParser.xmlConnectPeriod(startDate, endDate, currencyId);  
 }  
 DayCurrency prevDayCurrency = dayCurrencyList.get(dayCurrencyList.size() - 1).clone();  
 startDate = (Date) fromDate.clone();  
 List<DayCurrency> newCurrencyList = new ArrayList<>();  
 *// Проходим по валютам для каждого дня из периода*  
while (startDate.compareTo(endDate) <= 0) {  
 prevDayCurrency.setDate(startDate);  
 newCurrencyList.add(prevDayCurrency.clone());  
 startDate.setTime(startDate.getTime() + MS\_IN\_DAY);  
 startDate = *removeTime*(startDate);  
 }  
 return newCurrencyList;  
 }  
 private List<Date> getMissingDates(Date fromDate, Date toDate, List<DayCurrency> dayCurrencyList) {  
 int MS\_IN\_DAY = Integer.*parseInt*(Objects.*requireNonNull*(environment.getProperty("time.MS\_IN\_DAY")));  
  
 Date startDate = (Date) fromDate.clone(); *// Начальная дата запрашиваемого периода*  
Date endDate = (Date) toDate.clone(); *// Конечная дата запрашиваемого периода*  
List<Date> missingDateList = new ArrayList<>(); *// Коллекция недостающих незаполненных дней*  
  
int elemNum = 0;  
 *// Проходим по валютам для каждого дня из периода*  
while (startDate.compareTo(endDate) <= 0) {  
 if (elemNum > dayCurrencyList.size() - 1 ||  
 !Objects.*equals*(dayCurrencyList.get(elemNum).getDate().toString(), startDate.toString())) {  
 missingDateList.add((Date) startDate.clone());  
 } else {  
 elemNum++;  
 }  
  
 startDate.setTime(startDate.getTime() + MS\_IN\_DAY);  
 startDate = *removeTime*(startDate);  
 }  
 return missingDateList;  
 }

public static Date removeTime(Date date) {  
 Calendar cal = Calendar.*getInstance*();  
 cal.setTime(date);  
 cal.set(Calendar.*HOUR\_OF\_DAY*, 0);  
 cal.set(Calendar.*MINUTE*, 0);  
 cal.set(Calendar.*SECOND*, 0);  
 cal.set(Calendar.*MILLISECOND*, 0);  
 return new java.sql.Date(cal.getTimeInMillis());  
 }  
  
 public List<FullCurrencyInfo> getAllCurrenciesForDay(Date date) throws IOException, ParseException {  
 int MS\_IN\_DAY = Integer.*parseInt*(Objects.*requireNonNull*(environment.getProperty("time.MS\_IN\_DAY")));  
 List<FullCurrencyInfo> fullCurrencyInfos = dayCurrencyDAO.getAllCurrenciesForDay(date);  
 if (fullCurrencyInfos.size() == currencyDAO.getAll().size()) {  
 return fullCurrencyInfos;  
 } else {  
 Date startDate = (Date) date.clone();  
 List<DayCurrency> dayCurrencyListTmp = new ArrayList<DayCurrency>();  
 *//ищем дату, за которую есть данные по курсам валют*  
while (dayCurrencyListTmp.isEmpty()) {  
 *// Начальная дата запрашиваемого периода.*  
startDate.setTime(startDate.getTime() - 20L \* MS\_IN\_DAY); *//возврат на 20 дней назад*  
dayCurrencyListTmp = xmlParser.xmlConnectPeriod(startDate, date, currencyDAO.getIdByCharcode("USD"));  
 }  
 Date existingDate = dayCurrencyListTmp.get(dayCurrencyListTmp.size() - 1).getDate();  
  
 List<DayCurrency> dayCurrencyList = xmlParser.xmlDailyValutes(existingDate);  
 *//присваиваем сегодняшнюю дату*  
for (DayCurrency dayCurrency: dayCurrencyList  
 ) {  
 dayCurrency.setDate(date);  
 }  
 dayCurrencyDAO.deleteForDate(date);  
 dayCurrencyDAO.insertForDate(dayCurrencyList);  
  
 return dayCurrencyDAO.getAllCurrenciesForDay(date);  
 }  
 }  
}

*Код класса ServiceApplication:*

package com.practice.service;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class ServiceApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(ServiceApplication.class, args);  
 }  
}