Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему:

ВЭБ – СЕРВИС ДЛЯ МОНИТОРИНГА И АНАЛИЗА КУРСОВ КРИПТОВАЛЮТ НА ОСНОВЕ ДАННЫХ С НЕСКОЛЬКИХ КРИПТОВАЛЮТНЫХ БИРЖ

БГУИР КП 6-05-0612-01-001 ПЗ

Студент Михайлов К.А.

Руководитель Болтак С.В.

Минск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные информационные технологии играют ключевую роль в финансовой сфере, предоставляя пользователям инструменты для анализа и принятия решений на основе актуальных рыночных данных. С развитием криптовалютного рынка, включающего такие активы, как Bitcoin, Ethereum и другие, возникла потребность в веб-сервисах, которые обеспечивают централизованный доступ к информации о ценах, объемах торгов и изменениях стоимости с различных криптовалютных бирж. Такие сервисы позволяют трейдерам, инвесторам и аналитикам эффективно мониторить рынок и принимать обоснованные решения.

Криптовалютные биржи, такие как Binance, Bybit, Kraken, Bitfinex и Huobi, предоставляют публичные API для получения рыночных данных, однако их интеграция и обработка требуют удобного интерфейса и оптимизированных механизмов. Кроме того, для пользователей в Республике Беларусь важно отображение цен в местной валюте (BYN), что требует интеграции с надежными источниками курсов валют, такими как API Национального банка Республики Беларусь (НБРБ). Эффективное кэширование данных также является критически важным для минимизации запросов к внешним API и повышения производительности приложения.

В рамках данного курсового проекта рассматривается разработка веб-сервиса для мониторинга и анализа курсов криптовалют на основе данных с нескольких бирж. Сервис предоставляет пользователю интерфейс для просмотра актуальных цен, объемов торгов и изменений стоимости криптовалют, с возможностью фильтрации и сортировки данных. Особое внимание уделено пересчету цен в белорусские рубли с использованием официального курса USD/BYN, получаемого через API НБРБ, а также кэшированию данных для оптимизации производительности. Реализована REST-архитектура, обеспечивающая удобное взаимодействие с клиентскими приложениями. Разработанный сервис ориентирован на локальный рынок, предоставляя пользователям удобный доступ к глобальным данным в привычной валюте. Он позволяет сравнивать цены на различных биржах, выявляя наиболее выгодные предложения. Веб-приложение обеспечивает оперативное обновление данных с минимальными задержками. Применение кэширования снижает нагрузку на внешние API, улучшая отзывчивость системы. Сервис поддерживает гибкую настройку отображаемых данных, что повышает удобство использования. Проект демонстрирует интеграцию современных веб-технологий для решения актуальных задач финансового мониторинга. Решение может быть использовано как самостоятельный инструмент или интегрировано в более сложные системы.

Для реализации проекта использованы современные технологии и инструменты:

Spring MVC — фреймворк на языке Java для создания веб-приложений, обеспечивающий обработку HTTP-запросов и поддержку архитектуры Model-View-Controller;

REST API — архитектурный стиль взаимодействия между клиентом и сервером, основанный на использовании стандартных HTTP-методов;

Maven — инструмент управления зависимостями и сборки проекта, упрощающий конфигурацию и развертывание;

JavaServer Pages (JSP) — технология для создания динамических веб-страниц, обеспечивающая отображение данных в удобной форме;

Jackson — библиотека для обработки JSON, используемая для парсинга и сериализации API-ответов;

RestTemplate — компонент Spring для выполнения HTTP-запросов к внешним API;

Файловое кэширование — механизм хранения данных в JSON-файлах для минимизации запросов к внешним API и повышения скорости работы приложения;

Apache Tomcat — сервер приложений, используемый для развертывания и выполнения веб-приложения, обеспечивающий поддержку сервлетов и JSP.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

На сегодняшний день существует множество веб-сервисов и API, предназначенных для мониторинга и анализа курсов криптовалют. Эти решения предоставляют различные возможности по сбору, обработке и отображению рыночных данных, ориентируясь на трейдеров, инвесторов и разработчиков. Рассмотрим наиболее популярные аналоги, сравнивая их функциональность с разрабатываемым веб-сервисом.

CoinGecko — один из ведущих сервисов для мониторинга криптовалют, предоставляющий данные о ценах, рыночной капитализации, объемах торгов и других метриках для тысяч активов. Сервис агрегирует информацию с множества бирж, таких как Binance, Kraken и Huobi, через публичное API. CoinGecko поддерживает фильтрацию и сортировку данных, а также предоставляет исторические графики и аналитику. Однако сервис не предлагает встроенной конверсии цен в локальные валюты, такие как BYN, и ориентирован на глобальную аудиторию, что может быть неудобно для белорусских пользователей. Кроме того, частые запросы к API требуют платной подписки для повышения лимитов. Внешний вид данного приложения представлен на рисунке 1.1

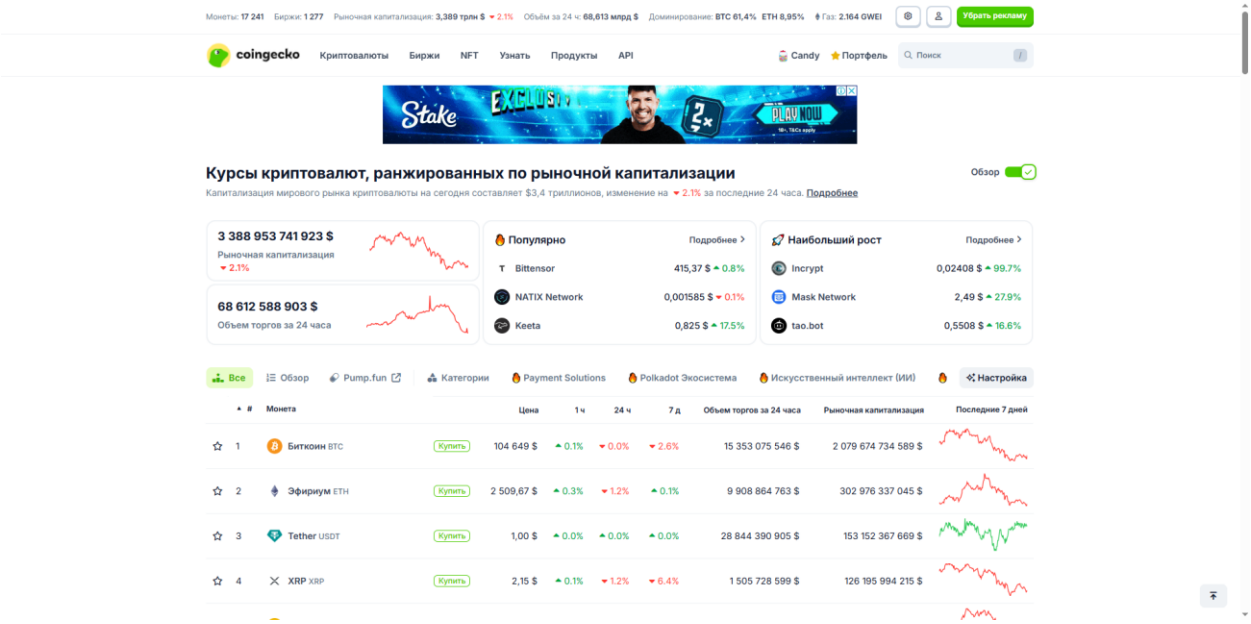

Рисунок 1.1 – CoinGecko

CoinMarketCap — еще один популярный сервис, аналогичный CoinGecko, с акцентом на предоставление данных о криптовалютах и биржах. Платформа поддерживает API для получения цен, объемов и рыночных метрик, а также предлагает интерактивные графики и аналитические инструменты. CoinMarketCap предоставляет возможность настройки

пользовательских портфелей, но, как и CoinGecko, не поддерживает автоматический пересчет цен в BYN. Интерфейс сервиса ориентирован на конечных пользователей, а API требует дополнительных усилий для интеграции в сторонние приложения. Внешний вид на рисунке 1.2

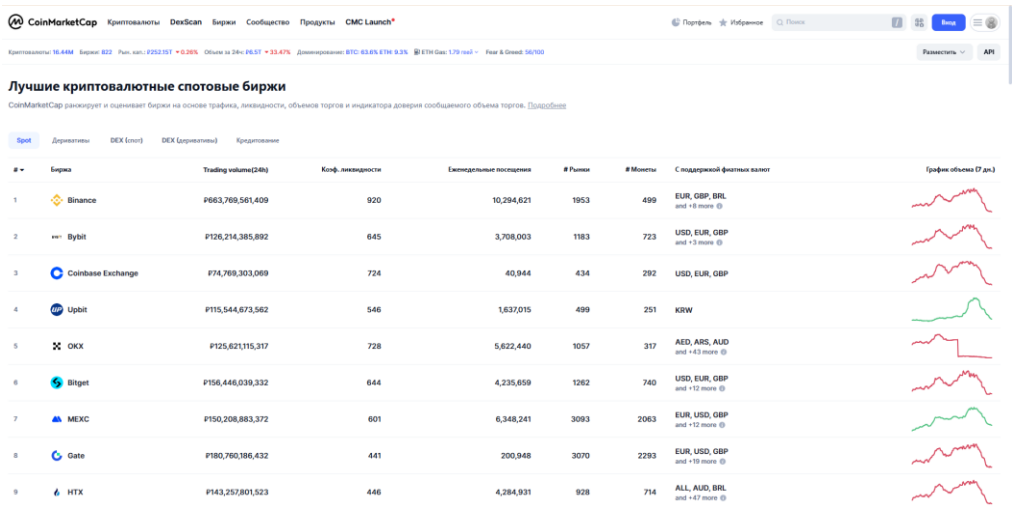

Рисунок 1.2 – CoinMarketCap

TradingView — платформа для технического анализа, предоставляющая данные о криптовалютах и других финансовых инструментах. TradingView поддерживает интеграцию с биржами и предоставляет API для получения рыночных данных. Сервис выделяется мощными инструментами для построения графиков и анализа, но его функциональность ориентирована на профессиональных трейдеров, что делает интерфейс сложным для новичков. Пересчет цен в локальные валюты возможен, но требует ручной настройки, а API не оптимизировано для массового мониторинга данных с нескольких бирж. Внешний вид данного приложения на рисунке 1.3

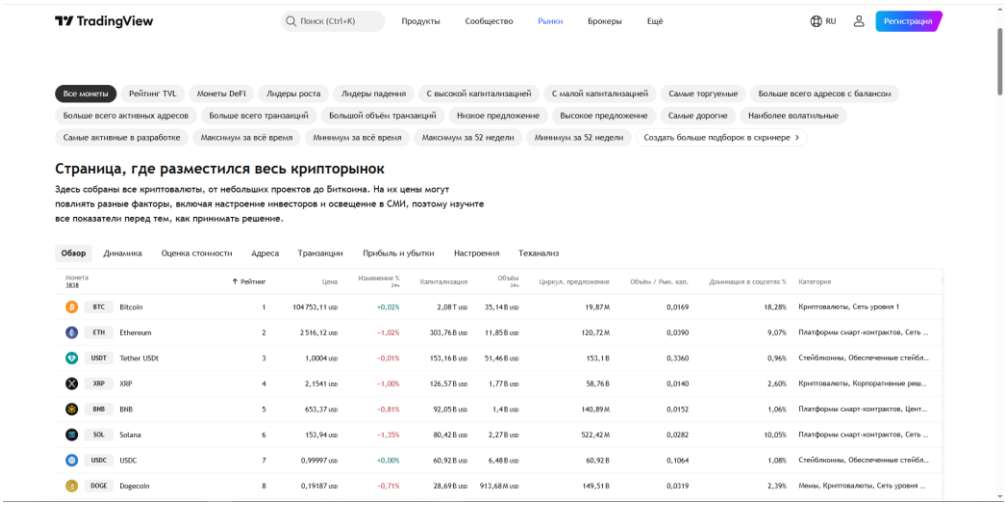

Рисунок 1.3 – TradingView

В отличие от перечисленных решений, разрабатываемый веб-сервис предлагает простой и интуитивно понятный REST-интерфейс для мониторинга курсов криптовалют, адаптированный для локального рынка.

Он позволяет:

- получать данные с нескольких бирж;
- выполнять фильтрацию и сортировку данных по цене, объему или изменению стоимости;
- фильтровать события по заданным временным промежуткам;
- автоматически пересчитывать цены в BYN с использованием официального курса USD/BYN через API НБРБ;
- кэшировать данные в JSON-файлах для минимизации запросов к внешним API и повышения производительности;
- предоставлять удобный интерфейс на основе JSP для визуализации рыночной информации.

Разработанный сервис не требует сложной настройки и ориентирован на пользователей, ищущих простое решение для мониторинга криптовалют с учетом локальных валют. В отличие от CoinGecko и CoinMarketCap, он предоставляет встроенную конверсию в BYN, а по сравнению с TradingView — более легковесный интерфейс и упрощенный доступ к данным. Сервис может быть использован как самостоятельное приложение или интегрирован в другие финансовые системы, что делает его конкурентоспособным решением.

1.2 Постановка задачи

В рамках данной курсовой работы необходимо разработать веб-сервис для мониторинга и анализа курсов криптовалют на основе данных с нескольких бирж. Сервис должен предоставлять REST API для взаимодействия с клиентскими приложениями и удобный веб-интерфейс для отображения рыночных данных. Основное внимание уделяется интеграции с API криптовалютных бирж, пересчету цен в белорусские рубли, кэшированию данных и обеспечению гибкости в обработке информации.

Основные функции, подлежащие реализации:

- получение актуальных данных о ценах, объемах торгов и изменениях стоимости криптовалют с бирж через API;
- поддержка агрегированных данных через API CoinGecko для получения общей рыночной информации;
- реализация пересчета цен в BYN с использованием официального курса USD/BYN, запрашиваемого через API НБРБ;
- обеспечение фильтрации данных по различным параметрам, таким как цена, биржа или процентное изменение стоимости.;
- реализация сортировки данных для удобного отображения в пользовательском интерфейсе;

– кэширование рыночных данных и курса USD/BYN в JSON-файлах для минимизации запросов к внешним источникам и повышения производительности;
– разработка веб-интерфейса на основе JSP для отображения данных в виде таблиц или списков с возможностью интерактивного управления.

Веб-служба будет реализована с использованием следующих технологий:

– язык программирования Java;
– фреймворк Spring MVC для обработки HTTP-запросов и реализации REST API;
– инструмент Maven для управления зависимостями и сборки проекта;
– реализация файлового кэширования с использованием JSON-файлов для хранения данных.
– обработка ошибок API-запросов и проверки корректности получаемых данных.
– apache tomcat для обеспечения работы веб-приложения с поддержкой JSP и сервлетов.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Структура программы

Веб-сервис для мониторинга и анализа курсов криптовалют построен на модульной архитектуре, основанной на принципах разделения ответственности, что обеспечивает удобство разработки, тестирования и масштабирования. Программа реализована с использованием фреймворка Spring MVC, который разделяет приложение на модель, представление и контроллер. Архитектура соответствует клиент-серверной модели: серверная часть обрабатывает HTTP-запросы через REST API и предоставляет данные для клиентского интерфейса, реализованного с использованием JSP. Развертывание осуществляется на сервере Apache Tomcat, который поддерживает выполнение сервлетов и JSP. Основные модули приложения включают:

- Controller – модуль отвечает за обработку входящих HTTP-запросов, маршрутизацию и взаимодействие с сервисами. Контроллеры принимают запросы от пользователей, вызывают методы сервисов и возвращают данные для отображения в JSP-страницах или в формате JSON через REST API. Содержит в себе класс CommonController.java управляющий навигацией и отображением данных;
- Service – модуль реализует бизнес-логику приложения, включая получение данных с внешних API, пересчет цен в BYN и управление кэшированием. Содержит в себе класс RecordService.java отвечающий за получение данных о криптовалютах, пересчет цен, кеширование данных, форматирование числовых значений;
- Entity – Модуль содержит классы, описывающие сущности данных для обработки JSON-ответов и передачи данных между слоями, Record.java модель данных о критовалюте, RecordsContainerDto.java DTO-класс для передачи списка объектов Record между слоями приложения.
- Models – модуль, содержащий описания сущностей базы данных (User, Event), а также DTO-классы для входных и выходных данных API;
- Views – модуль содержит JSP-страницы и CSS-стили, формирующие пользовательский интерфейс;
- Configuration – Модуль содержит файлы настройки приложения, определяющие параметры работы, зависимости и интеграцию с внешними сервисами. ApplicationInitializer.java класс инициализации веб-приложения, настраивает контекст Spring, регистрирует WebConfig и добавляет DispatcherServlet для обработки запросов, также содержит pom.xml файл конфигурации Maven, определяющий зависимости, параметры сборки и версию Java.

2.2 Проектирование интерфейса программного средства

Пользовательский интерфейс веб-сервиса спроектирован с акцентом на минимализм, интуитивность и визуальную привлекательность. Он реализован с использованием JSP-страниц, стилизованных через main-page.css, что обеспечивает динамическое формирование контента на основе данных от сервера. Интерфейс включает главное окно, страницы бирж, элементы управления и фильтры, оптимизированные для мониторинга криптовалют.

2.2.1 Главное окно

Она включает навигационную панель, расположенную в верхней части экрана, с логотипом CryptoMarket и ссылками на разделы «Курсы» (активный по умолчанию) и «Биржи», стилизованными с использованием градиентов и эффекта размытия, заданных в main-page.css. Под навигационной панелью размещена область фильтров, содержащая текстовое поле для поиска по названию или тикеру криптовалюты, два числовых поля для ввода минимальной и максимальной цены в USD, а также выпадающий список для сортировки по цене или рыночной капитализации в порядке возрастания или убывания. Фильтры реализованы через HTML-формы с JavaScript-обработчиками, обеспечивающими динамическое обновление данных без перезагрузки страницы. Основная область страницы представляет собой горизонтально прокручиваемый контейнер с карточками криптовалют, каждая из которых отображает название, тикер, стилизованную иконку с градиентом, текущую цену в USD и BYN, изменения цены за 1 час, 24 часа и 7 дней, а также рыночную капитализацию с суффиксами M или B для удобства чтения. Макет главного окна представлен на рисунке 2.1.

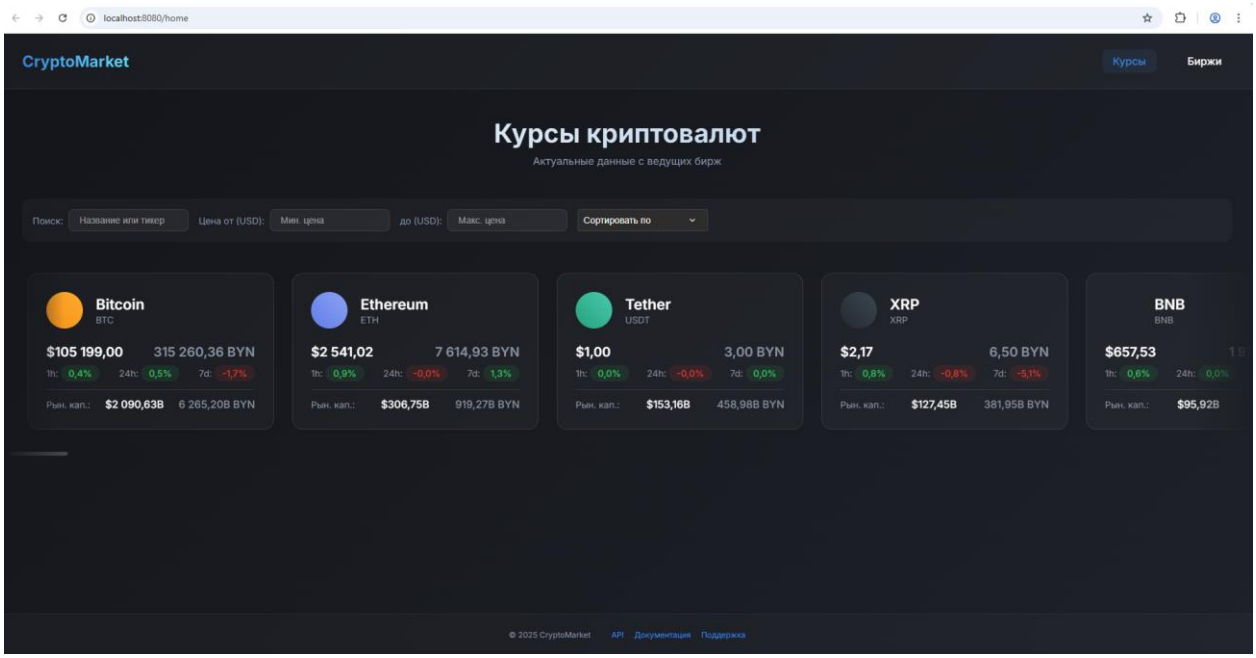

Рисунок 2.1 – Главное окно приложения

2.2.2 Элементы управления

Элементы управления, унифицированные для всех страниц, включают поле поиска, фильтры по цене и выпадающий список сортировки, стилизованный через CSS, а также кнопку обновления данных, реализованную через кликабельный логотип CryptoMarket в навигационной панели, который перенаправляет на текущую страницу с сохранением параметров фильтров. Навигация осуществляется через ссылки в навигационной панели и кликабельные карточки бирж на странице списка бирж. Все элементы управления поддерживают адаптивный дизайн с медиа-запросами в main-page.css, обеспечивая корректное отображение на мобильных устройствах. Внешний вид элементов управления веб-браузером представлен на рисунке 2.2.

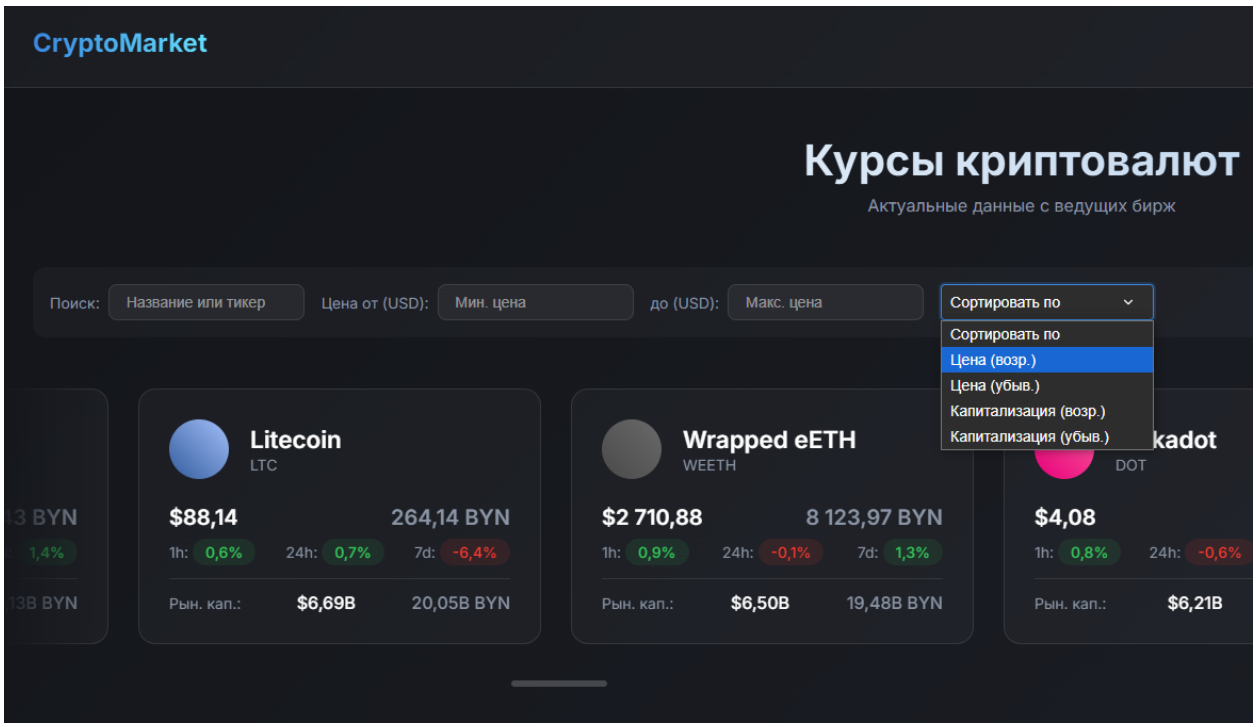

Рисунок 2.2 – Элементы управления приложением

2.2.3 Окно списка бирж и окно биржи

Страница списка бирж, реализованная в exchanges-page.jsp и доступная по пути /exchanges, содержит навигационную панель, поле для поиска по названию биржи и прокручиваемый контейнер с карточками бирж, включая Binance, Bybit, Kraken, Bitfinex и Huobi, каждая из которых имеет название и стилизованную иконку с цветами, соответствующими бренду биржи, заданными в main-page.css. Клик по карточке перенаправляет на страницу конкретной биржи по пути /exchange/<exchange>. Страницы бирж, такие как exchange-binance.jsp, exchange-bybit.jsp, exchange-kraken.jsp, exchange-bitfinex.jsp и exchange-huobi.jsp, аналогичны главной странице по структуре, но отображают данные о криптовалютах с соответствующей биржи. Они включают заголовок с названием биржи, кликабельный для перехода на официальный сайт, область фильтров с поиском, ценовым диапазоном и сортировкой по цене или объему торгов, а также карточки криптовалют с ценами в USD и BYN, объемом торгов и изменением цены за 24 часа. Окно списка бирж и окно после выбора биржи показано на рисунке 2.3 и 2.4.

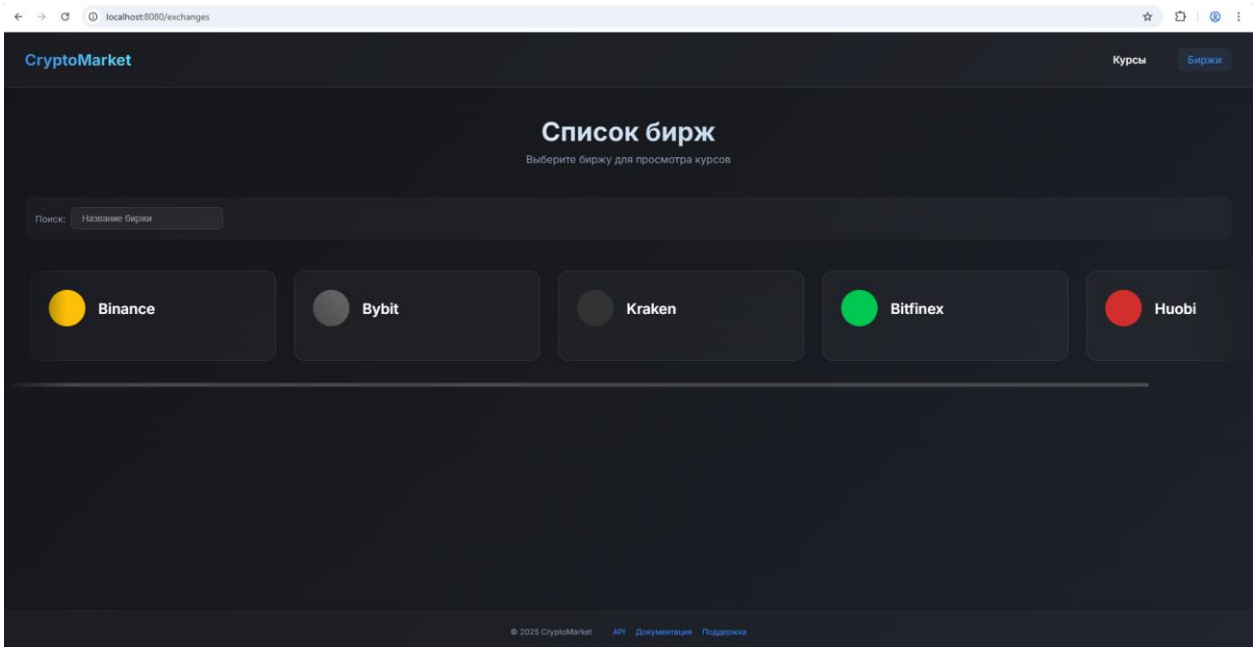

Рисунок 2.3 – Окно списка бирж

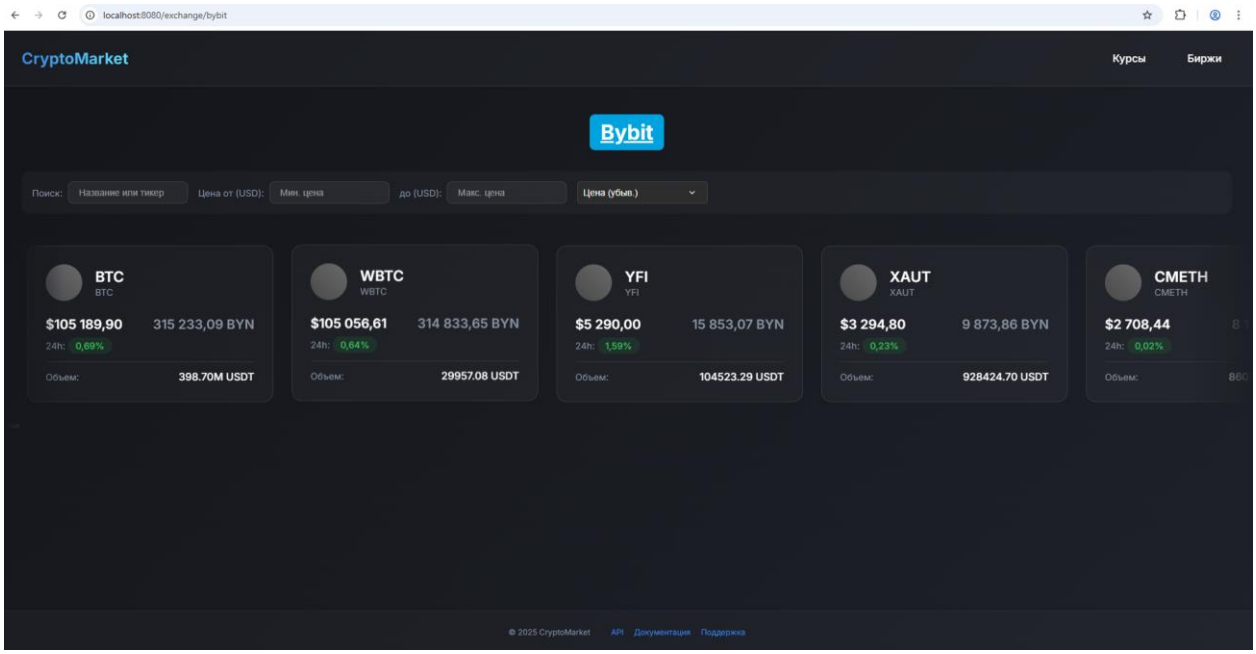

Рисунок 2.4 – Окно после выбора биржи

2.3 Проектирование функционала программного средства

Грамотно поставленная задача и четко определенные алгоритмы — ключевые этапы в проектировании программного средства.

Функционал веб-сервиса спроектирован для мониторинга и анализа курсов криптовалют с учетом требований к производительности и удобству. Основные функции включают получение данных с API, пересчет цен в BYN,

кэширование, фильтрацию и сортировку. Алгоритмы оптимизированы для минимизации сетевых запросов и повышения отзывчивости.:

2.3.1 Получение и кэширование данных

Получение данных реализовано в методах класса RecordService, таких как findAllRecords для API CoinGecko и findExchangeRecords для бирж Binance, Bybit, Kraken, Bitfinex и Huobi. Алгоритм сначала проверяет наличие актуального кэша в JSON-файлах, таких как cache_cryptoData.json для CoinGecko или cache_binance.json для Binance, возвращая данные, если их возраст не превышает пяти минут. При отсутствии кэша, его устаревании или запросе принудительного обновления через параметр forceRefresh, сервис выполняет HTTP-запрос с помощью RestTemplate к соответствующему API, парсит JSON-ответ через ObjectMapper, формирует объекты Record с информацией о криптовалютах, включая название, тикер, цену и рыночную капитализацию, после чего сохраняет данные в кэш методом saveToCache, записывающим JSON в файлы в директории ~/.cryptomarket/cache/. Блок-схема метода findAllRecords представлена на рисунке 2.4

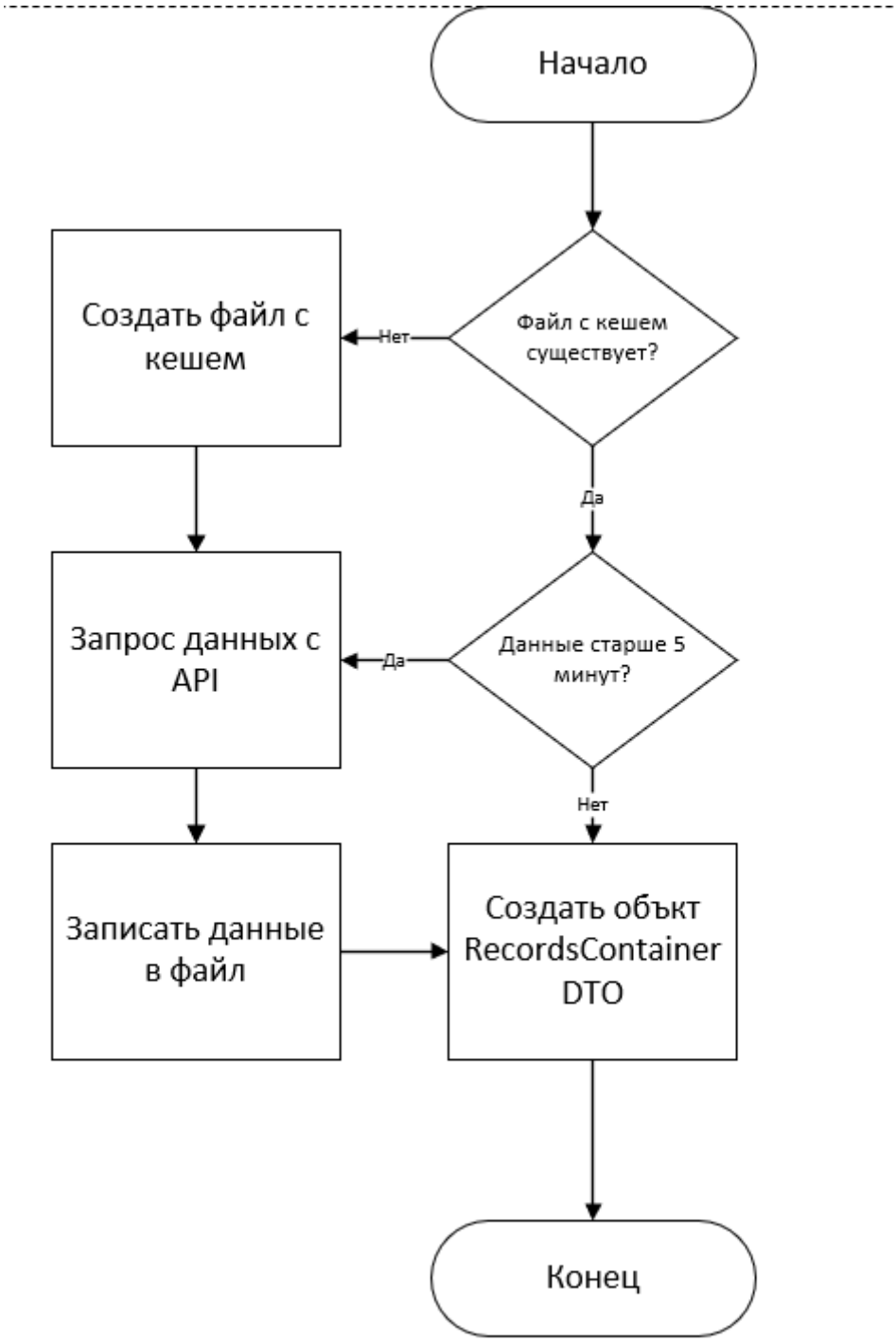

Рисунок 2.4 – Блок-схема метода findAllRecords

2.3.2 Пересчет цен в BYN

Пересчет цен в BYN встроен в методы fetchCryptoData и fetchCryptoDataForExchange класса RecordService, где цена и рыночная капитализация в USD умножаются на курс USD/BYN, полученный из метода getUsdToBynRate. Этот метод проверяет кэш в файле usd_byn_rate.json и, при его отсутствии или устаревании, выполняет запрос к API Национального банка Республики Беларусь, сохраняя курс в кэш через saveRateToCache. При сбое

API возвращается кэшированный курс или значение по умолчанию 3.20. Результаты пересчета сохраняются в поля currentPriceByn и marketCapByn объекта Record, обеспечивая отображение цен в двух валютах на страницах main-page.jsp и exchange-*.jsp. Блок-схема метода getUsdToBynRate представлена на рисунке 2.5.

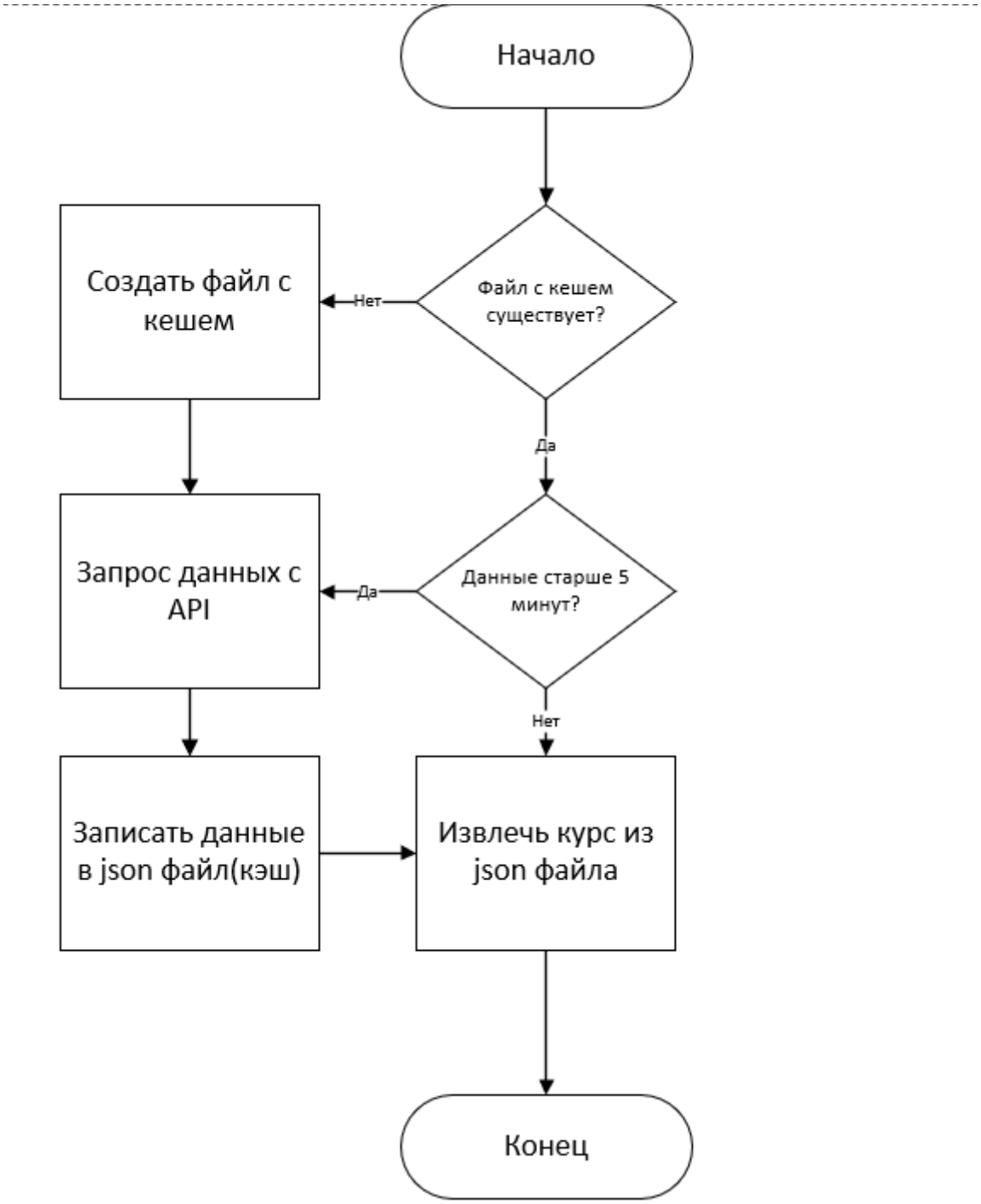

Рисунок 2.5 – Блок-схема метода getUsdToBynRate

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Настройка среды разработки

Настройка среды разработки была первым этапом, направленным на обеспечение стабильной работы инструментов и зависимостей. Для разработки использовалась среда IntelliJ IDEA Ultimate, поддерживающая интеграцию с Maven и Apache Tomcat. Установлена Java Development Kit (JDK) версии 17, указанная в файле pom.xml, что обеспечило совместимость с современными библиотеками Spring. Maven, как инструмент управления зависимостями, был настроен через pom.xml, где определены зависимости для Spring MVC (версия 5.3.27), Jackson (2.14.2), JSTL (1.2), Servlet API (4.0.1) и JSP API (2.3.3), а также плагины для сборки WAR-архива. Apache Tomcat версии 9.0 был установлен и сконфигурирован как сервер приложений, поддерживающий развертывание веб-приложения. В IntelliJ IDEA создан проект с структурой Maven, включающей директории src/main/java для Java-кода, src/main/webapp для веб-ресурсов (JSP, CSS) и src/main/webapp/WEB-INF/views для JSP-страниц. Настроен запуск приложения через Tomcat с автоматической пересборкой при изменении кода. Для работы с внешними API (CoinGecko, Binance, Bybit, Kraken, Bitfinex, Huobi, НБРБ) не требовалось дополнительной регистрации, так как использовались публичные конечные точки. Локальная директория ~/.cryptomarket/cache/ была создана вручную для проверки работы кэширования, хотя в дальнейшем ее создание автоматизировано в RecordService.java. Настройка окружения завершилась тестированием сборки проекта (mvn clean install) и успешным запуском пустого приложения на http://localhost:8080.

3.2 Рарзработка серверной части

Серверная часть веб-сервиса реализована с использованием фреймворка Spring MVC, обеспечивающего разделение приложения на модель, представление и контроллер. Разработка началась с создания конфигурационных классов в пакете ru.codekitchen.config. Класс ApplicationInitializer.java настроил инициализацию приложения, реализуя интерфейс WebApplicationInitializer, регистрируя DispatcherServlet и указывая WebConfig как конфигурацию Spring. Класс WebConfig.java включил поддержку Web MVC через аннотацию @EnableWebMvc, настроил сканирование компонентов в пакете ru.codekitchen, определил обработку статических ресурсов (/resources/**), настроил ViewResolver для JSP-страниц с префиксом /WEB-INF/views/ и суффиксом .jsp, а также создал бины RestTemplate для HTTP-запросов и ObjectMapper для обработки JSON. Файл pom.xml был дополнен зависимостями, необходимыми для этих компонентов.

Основная бизнес-логика реализована в классе RecordService.java (пакет ru.codekitchen.service). Этот класс отвечает за получение данных с внешних

API, пересчет цен в BYN, кэширование и форматирование данных. Метод findAllRecords обрабатывает запросы данных с CoinGecko, используя кэш cache_cryptoData.json, и поддерживает фильтрацию и принудительное обновление. Метод findExchangeRecords аналогично обрабатывает данные с бирж (Binance, Bybit, Kraken, Bitfinex, Huobi), сохраняя их в файлы cache_<exchange>.json. Метод getUsdToBynRate запрашивает курс USD/BYN через API НБРБ, кэшируя его в usd_byn_rate.json, с резервным значением 3.20 при сбоях. Для кэширования реализованы методы loadFromCache, saveToCache, loadRateFromCache, saveRateToCache, использующие ObjectMapper для сериализации/десериализации JSON. Конструктор RecordService создает директорию ~/.cryptomarket/cache/ при инициализации. Модель данных определена в пакете ru.codekitchen.entity: класс Record.java описывает криптовалюту (название, тикер, цена в USD/BYN, изменения цены, объем), RecordsContainerDto.java используется для передачи списка записей, а RecordStatus.java оставлен как заглушка для возможного расширения. Класс RecordDao.java (пакет ru.codekitchen.dao) реализован как заглушка с пустым методом findAllRecords, предполагая будущую интеграцию с базой данных.

Контроллер CommonController.java (пакет ru.codekitchen.controller) управляет HTTP-запросами. Метод redirectToHomePage перенаправляет корневой путь (/) на /home. Метод getMainPage отображает главную страницу (main-page.jsp) с данными от RecordService.findAllRecords, поддерживая фильтры (имя/тикер, ценовой диапазон) и сортировку. Метод getExchangesPage отображает список бирж (exchanges-page.jsp), а getExchangePage — данные конкретной биржи (exchange-<exchange>.jsp). Метод refreshData предоставляет REST API для обновления данных. Контроллер использует аннотации @GetMapping, @PostMapping, @ResponseBody для маршрутизации и обработки запросов.

3.3 Разработка клиентской части

Клиентская часть реализована с использованием JSP-страниц, размещенных в директории src/main/webapp/WEB-INF/views, и стилизована через CSS-файл main-page.css в src/main/webapp/resources/css. Главная страница main-page.jsp отображает карточки криптовалют с данными от CoinGecko, включая название, тикер, цену в USD/BYN, изменения цены и рыночную капитализацию. Страница содержит навигационную панель с логотипом CryptoMarket и ссылками на разделы «Курсы» и «Биржи», область фильтров (поиск по имени/тикеру, ценовой диапазон, сортировка по цене/капитализации) и прокручиваемый контейнер с карточками. Страница exchanges-page.jsp отображает карточки бирж (Binance, Bybit, Kraken, Bitfinex, Huobi) с фильтром по названию. Страницы exchange-binance.jsp, exchange-bybit.jsp, exchange-kraken.jsp, exchange-bitfinex.jsp и exchange-huobi.jsp показывают данные о криптовалютах с соответствующей биржи, повторяя структуру главной страницы, но с заголовком, кликабельным для перехода на

сайт биржи, и сортировкой по цене/объему. Файл main-page.css унифицирует дизайн, задавая градиенты для навигационной панели, стили карточек, анимации для интерактивных элементов и адаптивный макет через медиа-запросы для мобильных устройств.

Интерактивность интерфейса реализована через встроенные JavaScript-скрипты в JSP-страницах. Функции фильтрации и сортировки обрабатывают пользовательский ввод (поиск, ценовой диапазон, выбор сортировки), динамически скрывая/показывая карточки в DOM на основе атрибутов data-name, data-price, data-volume и сортируя их по выбранному критерию. Параметры фильтров сохраняются в localStorage, обеспечивая их восстановление при обновлении страницы. JavaScript-код использует события input и change для немедленного обновления интерфейса без обращения к серверу, что повышает отзывчивость. Логотип CryptoMarket в навигационной панели реализован как кнопка обновления, перенаправляющая на текущую страницу с сохранением фильтров.

3.4 Работа с базой данных

Веб-сервис для мониторинга курсов криптовалют использует JSON-файлы как основное средство хранения данных, что обеспечивает простоту реализации, высокую скорость доступа и независимость от внешних СУБД. JSON-файлы (cache_cryptoData.json, cache_<exchange>.json для бирж, usd_byn_rate.json) выступают в роли базы данных, заменяя традиционное кэширование или реляционные таблицы. Они содержат данные о криптовалютах с API CoinGecko и бирж (Binance, Bybit, Kraken, Bitfinex, Huobi), а также курс USD/BYN, получаемый от НБРБ. Данный подраздел описывает организацию работы с JSON-файлами, включая их структуру, процессы создания, чтения, записи, обновления и управления, реализованные в классе RecordService.java, а также тестирование функциональности.

Работа с JSON-файлами организована в директории ~/.cryptomarket/cache/, которая создается автоматически при инициализации сервиса. Основные файлы включают: cache_cryptoData.json для хранения данных о криптовалютах с CoinGecko, используемых на главной странице (main-page.jsp); cache_binance.json, cache_bybit.json, cache_kraken.json, cache_bitfinex.json, cache_huobi.json для данных с соответствующих бирж, отображаемых на страницах exchange-<exchange>.jsp; и usd_byn_rate.json для хранения курса USD/BYN. Структура cache_cryptoData.json и cache_<exchange>.json представляет собой массив объектов, где каждый объект соответствует сущности Record и содержит поля: id (идентификатор), name (название криптовалюты), ticker (тикер), current_price_usd (цена в USD), current_price_byn (цена в BYN), market_cap_usd (рыночная капитализация в USD), market_cap_byn (в BYN), price_change_1h (изменение цены за 1 час), price_change_24h (за 24 часа), price_change_7d (за 7 дней), volume_24h (объем

торгов). Файл usd_byn_rate.json содержит объект с полями rate (курс USD/BYN) и timestamp (временная метка).

Создание и управление JSON-файлами реализовано в классе RecordService.java (пакет ru.codekitchen.service). При инициализации сервиса конструктор RecordService проверяет наличие директории ~/.cryptomarket/cache/ и создает ее с помощью метода File.mkdirs(), если она отсутствует. Это обеспечивает готовность файловой системы для хранения данных. Метод saveToCache отвечает за запись данных о криптовалютах в файлы cache_cryptoData.json и cache_<exchange>.json. Он принимает путь к файлу и список объектов Record, сериализует их в JSON-строку с помощью ObjectMapper.writeValueAsString и записывает в файл через Files.write. Если файл не существует, он создается автоматически; если существует, перезаписывается. Аналогично, метод saveRateToCache сохраняет курс USD/BYN в usd_byn_rate.json, сериализуя объект RateCache (внутренний класс, содержащий rate и timestamp). Чтение данных выполняется методами loadFromCache и loadRateFromCache, которые десериализуют JSON-файлы обратно в объекты List<Record> или RateCache с помощью ObjectMapper.readValue. Если файл отсутствует или поврежден, методы возвращают null, что обрабатывается в логике сервиса.

Обновление данных реализовано в методах findAllRecords, findExchangeRecords и getUsdToBynRate. Метод findAllRecords проверяет наличие актуального кэша в cache_cryptoData.json через loadFromCache и isCacheExpired (срок жизни кэша — 5 минут). Если кэш отсутствует, устарел или запрошено обновление (forceRefresh=true), вызывается fetchCryptoData для получения данных с API CoinGecko, после чего данные сохраняются через saveToCache. Метод findExchangeRecords работает аналогично для бирж, используя файлы cache_<exchange>.json. Метод getUsdToBynRate проверяет usd_byn_rate.jso` и, при необходимости, запрашивает курс через API НБРБ, сохраняя его через saveRateToCache. В случае сбоев API используется кэшированный курс или значение по умолчанию (3.20). Метод isCacheExpired определяет актуальность кэша, сравнивая временную метку файла с текущим временем, что предотвращает избыточные запросы к API.

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

На этапе тестирования веб-сервиса для мониторинга курсов криптовалют проводилась проверка корректности функционирования всех основных компонентов приложения: клиентского интерфейса (frontend), серверной логики (backend) и интеграции с внешними API. Веб-сервис не использует систему аутентификации, а вместо базы данных применяются JSON-файлы (cache_cryptoData.json, cache_<exchange>.json, usd_byn_rate.json) для кэширования данных. Тестирование включало модульные, интеграционные, функциональные, нагрузочные и пользовательские тесты, направленные на обеспечение надежности, производительности и удобства использования. В результате тестирования были выявлены и устранены следующие недочеты, обеспечившие стабильную работу сервиса:

1) Проблема с некорректной фильтрацией по цене

При тестировании фильтрации на странице main-page.jsp было замечено, что фильтр по ценовому диапазону (минимальная и максимальная цена) иногда исключал корректные записи, если цена содержала дробные значения с высокой точностью (например, 0.00012345). Это происходило из-за неправильной обработки числовых значений в JavaScript.

Решение:

В JavaScript-коде фильтрации была реализована нормализация цен с использованием метода parseFloat и округления до 8 десятичных знаков перед сравнением. Кроме того, добавлена проверка на валидность пользовательского ввода (например, исключение отрицательных цен), что устранило ошибку и улучшило точность фильтрации.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

5.1 Интерфейс программного средства

5.1.1 Главная страница

При запуске приложения пользователь попадает на главную страницу (main-page.jsp), доступную по адресу http://localhost:8080/home. Страница разделена на три основные зоны: навигационная панель вверху, область фильтров и сортировки, и прокручиваемый контейнер с карточками криптовалют. Навигационная панель содержит логотип CryptoMarket (кликабельный для обновления страницы) и ссылки на разделы «Курсы» (текущая страница) и «Биржи» (переход к exchanges-page.jsp). Область фильтров включает поля для поиска по названию или тикеру, ввода минимальной и максимальной цены, а также выпадающий список для сортировки (по цене, рыночной капитализации, объему торгов, возрастанию/убыванию). Карточки криптовалют отображают название, тикер, цену в USD и BYN, изменения цены (1 час, 24 часа, 7 дней), объем торгов и капитализацию. Внешний вид главной страницы представлен на рисунке 5.1.

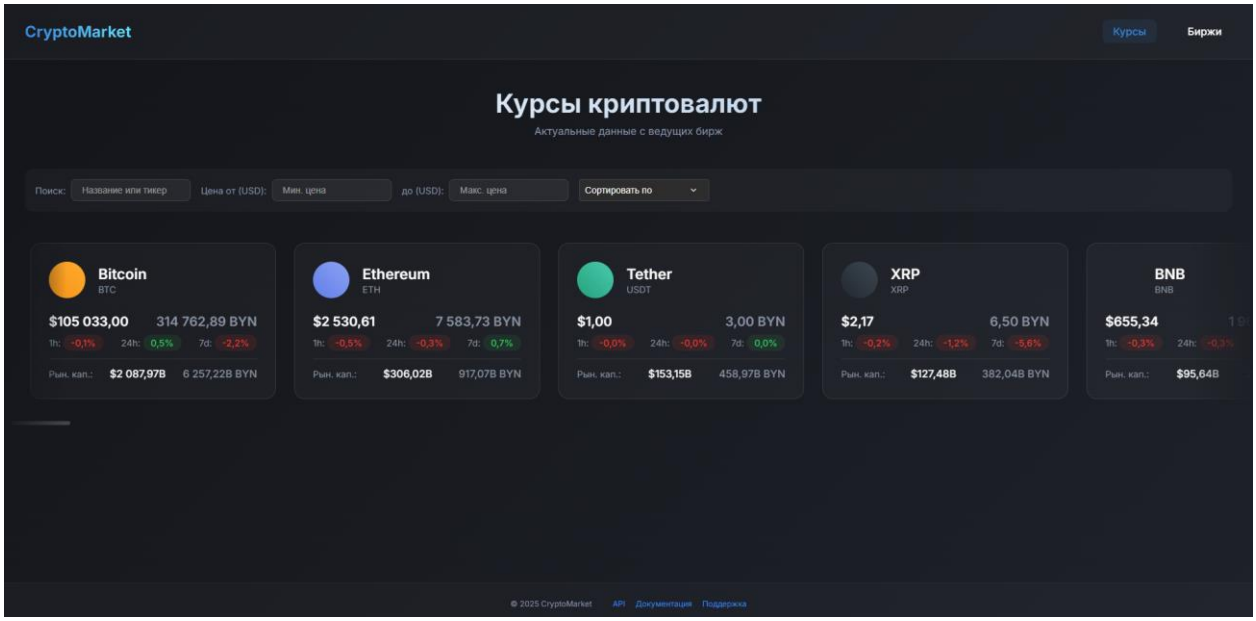

Рисунок 5.1 – Окно регистрации

5.1.2 Страница бирж

Страница бирж (exchanges-page.jsp), доступная по адресу http://localhost:8080/exchanges, отображает список поддерживаемых бирж (Binance, Bybit, Kraken, Bitfinex, Huobi) в виде карточек. Каждая карточка содержит название биржи и ссылку на соответствующую страницу (exchange-<exchange>.jsp). В верхней части страницы расположена навигационная панель, аналогичная главной странице, и поле поиска для фильтрации бирж по названию. Внешний вид страницы бирж показан на рисунке 5.2.

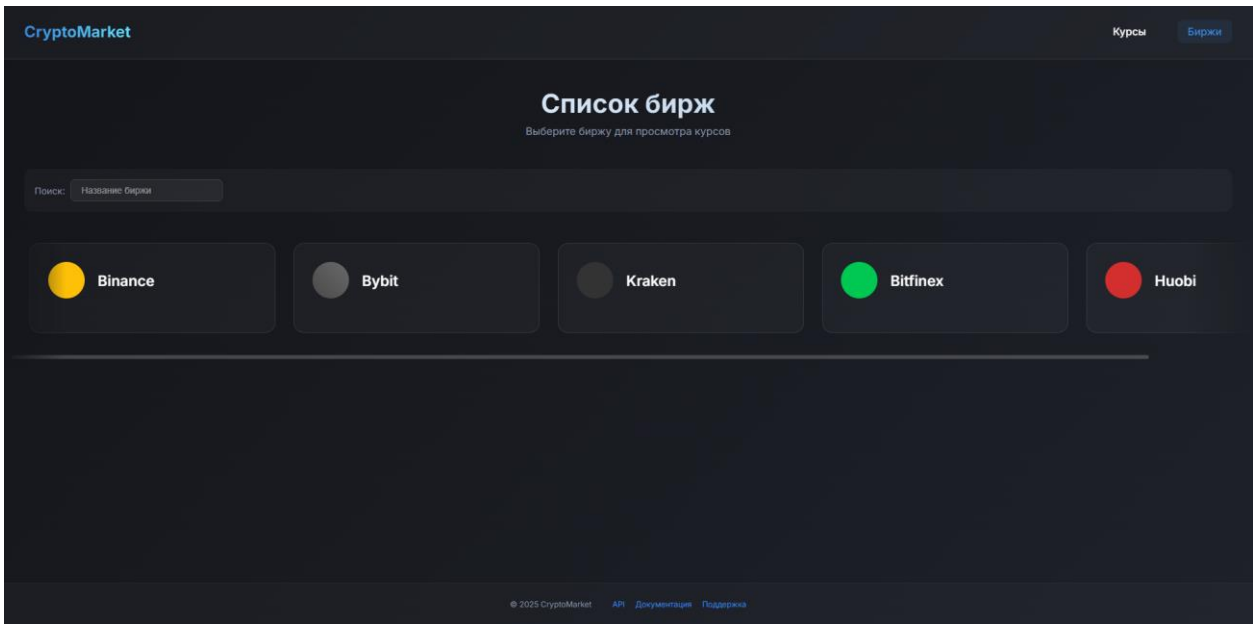

Рисунок 5.2 – Страница бирж

5.1.3 Страница отдельных бирж

Страницы отдельных бирж (например, exchange-binance.jsp, доступная по адресу http://localhost:8080/exchange/binance) имеют структуру, схожую с главной страницей, но отображают данные о криптовалютах с конкретной биржи. Заголовок страницы содержит название биржи, кликабельное для перехода на официальный сайт биржи. Область фильтров и сортировки позволяет искать по названию/тикеру, задавать ценовой диапазон и сортировать по цене или объему торгов. Карточки криптовалют включают те же данные, что на главной странице. Внешний вид страницы биржи представлен на рисунке 5.3.

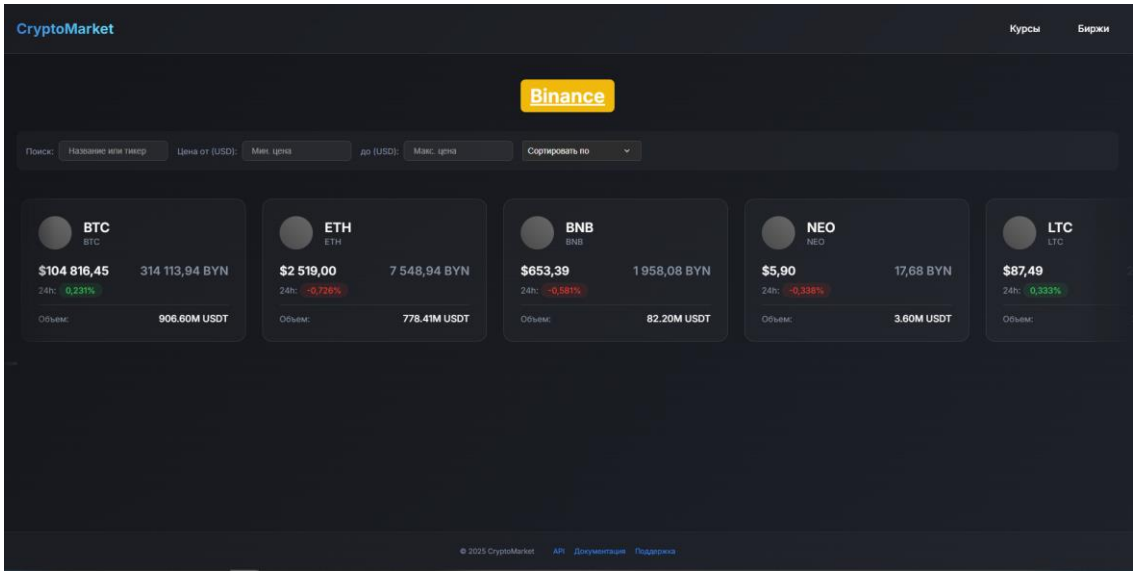

Рисунок 5.3 – Страница биржи binance

5.2 Управление программным средством

5.2.1 Элементы управления приложением

правление веб-сервисом осуществляется через интерактивные элементы интерфейса, включая кнопки, поля ввода и выпадающие списки. Все элементы снабжены всплывающими подсказками, описывающими их назначение, для повышения удобства использования.

Пользователю доступны следующие функции:

- Использование ссылок «Курсы» и «Биржи» в навигационной панели для перехода между main-page.jsp и exchanges-page.jsp.;
- Ввод названия или тикера в поле поиска на страницах для фильтрации;
- Ввод минимальной и максимальной цены в соответствующие поля для фильтрации карточек;
- Клик по карточке биржи на вкладке «Биржи» для просмотра курсов на этой бирже;
- Переход на сайт биржи по нажатию на название биржи;
- Прокрутка данных с помощью скрола;
- Принудительное обновление данных нажатием на «CryptoMarket»;

Такая организация интерфейса обеспечивает удобную и интуитивно понятную работу с календарём.

ЗАКЛЮЧЕНИЕ

В современном мире цифровые инструменты для мониторинга и анализа финансовых рынков, включая криптовалюты, становятся важной частью как профессиональной, так и повседневной деятельности. Веб-сервисы, предоставляющие актуальную информацию о курсах криптовалют, помогают пользователям принимать информированные решения и следить за динамикой рынка в реальном времени.

В рамках данного курсового проекта было разработано программное средство — веб-сервис для мониторинга курсов криптовалют, предназначенное для получения, обработки и отображения данных с платформы CoinGecko и бирж (Binance, Bybit, Kraken, Bitfinex, Huobi), а также пересчета цен в белорусские рубли с использованием курса USD/BYN от НБРБ. Приложение реализовано с использованием современных технологий: Java с фреймворком Spring MVC для серверной части, JSP для клиентского интерфейса, Apache Tomcat в качестве сервера приложений и JSON-файлов для кэширования данных. Взаимодействие с внешними API осуществляется через REST-запросы, обеспечивая актуальность и надежную информацию.

В процессе разработки были успешно реализованы все поставленные задачи. Функционал веб-сервиса включает:

- Отображение актуальных данных о криптовалютах на главной странице с указанием цен в USD и BYN, изменений за 1 час, 24 часа и 7 дней, объема торгов и рыночной капитализации;
- Возможность фильтрации данных по названию, тикеру и ценовому диапазону;
- Добавление событий на выбранную дату;
- Редактирование и удаление событий (с помощью иконок карандаша и мусорной корзины);
- Сортировку данных по цене, капитализации или объему торгов в порядке возрастания или убывания;
- Просмотр данных с отдельных бирж на специализированных страницах с поддержкой аналогичных фильтров и сортировки;
- Автоматическое обновление данных через запрос к API с использованием кэша (срок жизни 5 минут);
- Сохранение параметров фильтров в браузере для удобства повторного использования.

Перспективы развития приложения включают:

- Интеграцию дополнительных источников данных, таких как другие криптовалютные биржи или аналитические платформы;
- Внедрение уведомлений о значительных изменениях цен или объемов торгов;
- Добавление графиков и визуализации динамики цен для каждой криптовалюты;

Разработанное программное средство предоставляет удобный и интуитивно понятный интерфейс для мониторинга криптовалютного рынка, что делает его полезным инструментом для трейдеров, инвесторов и энтузиастов цифровых активов. Сервис обеспечивает быстрый доступ к актуальной информации и гибкие возможности анализа, способствуя эффективному принятию решений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Spring. Spring Framework Documentation [Электронный ресурс]. – Режим доступа: https://docs.spring.io/spring-framework/docs/, свободный..

[2] Oracle. Java Platform, Standard Edition Documentation [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/java/javase/, свободный.

[3] Apache. Apache Tomcat Documentation [Электронный ресурс]. – Режим доступа: https://tomcat.apache.org/tomcat-9.0-doc/, свободный.

[4] CoinGecko. CoinGecko API Documentation [Электронный ресурс]. – Режим доступа: https://www.coingecko.com/en/api/documentation, свободный.

[6] Хорстманн К. Java. Библиотека профессионала. Том 1. Основы. — М.: Вильямс, 2020. — 864 с.

[7] Фримен Э., Робсон Э. Head First. Изучаем Java. — СПб.: Питер, 2021. — 720 с.

[8] Фленаган Д. JavaScript. Подробное руководство. — СПб.: Символ-Плюс, 2020. — 960 с.

[9] Мартин Р. Чистый код: создание, анализ и рефакторинг. — СПб.: Питер, 2018. — 464 с.

ПРИЛОЖЕНИЕ А. Исходный код программы

ApplicationInitializer.java:

```
package ru.codekitchen.config;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.ContextLoaderListener;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

import javax.servlet.ServletContext;
import javax.servlet.ServletRegistration;

public class ApplicationInitializer implements WebApplicationInitializer {
   private static final String DISPATCHER = "dispatcher";

   @Override
   public void onStartup(ServletContext servletContext) {
      AnnotationConfigWebApplicationContext context = new
AnnotationConfigWebApplicationContext();
      context.register(WebConfig.class);
      servletContext.addListener(new ContextLoaderListener(context));

      ServletRegistration.Dynamic servlet = servletContext.addServlet(DISPATCHER, new
DispatcherServlet(context));
      servlet.addMapping("/");
      servlet.setLoadOnStartup(1);
   }
}
```

WebConfig.java:

```
package ru.codekitchen.config;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
```

```
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
@ComponentScan("ru.codekitchen")
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/resources/**").addResourceLocations("/resources/");
    }

    @Bean(name = "viewResolver")
    public ViewResolver getViewResolver() {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/views/");
        resolver.setSuffix(".jsp");
        return resolver;
    }

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

    @Bean
    public ObjectMapper objectMapper() {
        return new ObjectMapper();
    }
}
```

CommonController.java:

```
package ru.codekitchen.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
```

```
import org.springframework.web.bind.annotation.*;
import ru.codekitchen.entity.dto.RecordsContainerDto;
import ru.codekitchen.service.RecordService;

@Controller
public class CommonController {
    private final RecordService recordService;

    @Autowired
    public CommonController(RecordService recordService) {
        this.recordService = recordService;
    }

    @RequestMapping("/")
    public String redirectToHomePage() {
        return "redirect:/home";
    }

    @RequestMapping("/home")
    public String getMainPage(Model model,
                              @RequestParam(name = "filter", required = false) String filterMode,
                              @RequestParam(name = "minPrice", required = false) String minPrice,
                              @RequestParam(name = "maxPrice", required = false) String maxPrice,
                              @RequestParam(name = "sort", required = false) String sort) {
        RecordsContainerDto container = recordService.findAllRecords(filterMode, false);
        model.addAttribute("records", container.getRecords());
        model.addAttribute("filterMode", filterMode);
        model.addAttribute("minPrice", minPrice);
        model.addAttribute("maxPrice", maxPrice);
        model.addAttribute("sort", sort);
        return "main-page";
    }

    @RequestMapping("/exchanges")
    public String getExchangesPage(Model model,
                                   @RequestParam(name = "filter", required = false) String filterMode) {
        String[] exchanges = {"binance", "bybit", "kraken", "bitfinex", "huobi"}; // Добавляем
Huobi
        model.addAttribute("exchanges", exchanges);
```

```
        model.addAttribute("filterMode", filterMode);
        return "exchanges-page";
    }

    @RequestMapping("/exchange/{exchange}")
    public String getExchangePage(@PathVariable String exchange, Model model,
                                  @RequestParam(name = "filter", required = false) String filterMode,
                                  @RequestParam(name = "minPrice", required = false) String minPrice,
                                  @RequestParam(name = "maxPrice", required = false) String maxPrice,
                                  @RequestParam(name = "sort", required = false) String sort) {
        RecordsContainerDto container = recordService.findExchangeRecords(exchange,
filterMode, false);
        model.addAttribute("records", container.getRecords());
        model.addAttribute("exchange", exchange.substring(0, 1).toUpperCase() +
exchange.substring(1));
        model.addAttribute("filterMode", filterMode);
        model.addAttribute("minPrice", minPrice);
        model.addAttribute("maxPrice", maxPrice);
        model.addAttribute("sort", sort);
        return "exchange-" + exchange; // Используем разные JSP для разных бирж: exchange-
binance, exchange-bybit, exchange-kraken, exchange-bitfinex, exchange-huobi
    }

    @RequestMapping(value = "/refresh-data", method = RequestMethod.POST)
    @ResponseBody
    public RecordsContainerDto refreshData() {
        return recordService.findAllRecords(null, true);
    }
}
```

RecordsContainerDto.java:

```
    package ru.codekitchen.entity.dto;

    import ru.codekitchen.entity.Record;

    import java.util.List;

    public class RecordsContainerDto {
        private List<Record> records;
```

```
    public RecordsContainerDto(List<Record> records) {
        this.records = records;
    }

    public List<Record> getRecords() {
        return records;
    }

    public void setRecords(List<Record> records) {
        this.records = records;
    }
}
```

Record.java:

```
package ru.codekitchen.entity;

public class Record {
    private String id;
    private String name;
    private String symbol;
    private double currentPrice; // в USD
    private double currentPriceByn; // в BYN
    private double marketCap; // в USD
    private double marketCapByn; // в BYN
    private double priceChange1h;
    private double priceChange24h;
    private double priceChange7d;
    private double volume; // Объем торгов

    public Record(String id, String name, String symbol, double currentPrice, double
currentPriceByn,
            double marketCap, double marketCapByn, double priceChange1h, double
priceChange24h,
            double priceChange7d, double volume) {
        this.id = id;
        this.name = name;
        this.symbol = symbol;
        this.currentPrice = currentPrice;
        this.currentPriceByn = currentPriceByn;
        this.marketCap = marketCap;
```

```
        this.marketCapByn = marketCapByn;
        this.priceChange1h = priceChange1h;
        this.priceChange24h = priceChange24h;
        this.priceChange7d = priceChange7d;
        this.volume = volume;
    }

    // Геттеры и сеттеры
    public String getId() { return id; }
    public String getName() { return name; }
    public String getSymbol() { return symbol; }
    public double getCurrentPrice() { return currentPrice; }
    public double getCurrentPriceByn() { return currentPriceByn; }
    public double getMarketCap() { return marketCap; }
    public double getMarketCapByn() { return marketCapByn; }
    public double getPriceChange1h() { return priceChange1h; }
    public double getPriceChange24h() { return priceChange24h; }
    public double getPriceChange7d() { return priceChange7d; }
    public double getVolume() { return volume; }

    public void setId(String id) { this.id = id; }
    public void setName(String name) { this.name = name; }
    public void setSymbol(String symbol) { this.symbol = symbol; }
    public void setCurrentPrice(double currentPrice) { this.currentPrice = currentPrice; }
    public void setCurrentPriceByn(double currentPriceByn) { this.currentPriceByn =
currentPriceByn; }
    public void setMarketCap(double marketCap) { this.marketCap = marketCap; }
    public void setMarketCapByn(double marketCapByn) { this.marketCapByn = marketCapByn;
}
    public void setPriceChange1h(double priceChange1h) { this.priceChange1h = priceChange1h;
}
    public void setPriceChange24h(double priceChange24h) { this.priceChange24h =
priceChange24h; }
    public void setPriceChange7d(double priceChange7d) { this.priceChange7d = priceChange7d;
}
    public void setVolume(double volume) { this.volume = volume; }
}
```

RecordService.java:

```
package ru.codekitchen.service;
```

```
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
import ru.codekitchen.dao.RecordDao;
import ru.codekitchen.entity.Record;
import ru.codekitchen.entity.dto.RecordsContainerDto;

import java.io.File;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

@Service
public class RecordService {
    private final RecordDao recordDao;
    private final RestTemplate restTemplate;
    private final ObjectMapper objectMapper;

    private static final long CACHE_TTL = 5 * 60 * 1000; // 5 минут в миллисекундах
    private static final int PAGE_SIZE = 20; // Количество записей на странице (для справки)
    private static final String CACHE_DIR = System.getProperty("user.home") +
"/.cryptomarket/cache/";
    private static final String CACHE_FILE_PREFIX = "cache_";
    private static final String USD_BYN_RATE_CACHE_FILE = CACHE_DIR +
"usd_byn_rate.json";

    @Autowired
    public RecordService(RecordDao recordDao, RestTemplate restTemplate, ObjectMapper
objectMapper) {
```

```
    this.recordDao = recordDao;
    this.restTemplate = restTemplate;
    this.objectMapper = objectMapper;

    // Создаём директорию для кэша, если она не существует
    File cacheDir = new File(CACHE_DIR);
    if (!cacheDir.exists()) {
        boolean created = cacheDir.mkdirs();
        if (created) {
            System.out.println("Cache directory created: " + CACHE_DIR);
        } else {
            System.err.println("Failed to create cache directory: " + CACHE_DIR);
        }
    } else {
        System.out.println("Cache directory already exists: " + CACHE_DIR);
    }
}

// Геттер для PAGE_SIZE
public static int getPageSize() {
    return PAGE_SIZE;
}

// Метод форматирования чисел (кроме рыночной капитализации и объема)
private String formatPrice(double value) {
    if (Double.isNaN(value) || Double.isInfinite(value)) {
        return "Invalid";
    }
    if (value != 0 && Math.abs(value) < 0.01) {
        DecimalFormat df = new DecimalFormat("0.000000"); // 6 знаков после запятой
        return "~" + df.format(value);
    } else {
        DecimalFormat df = new DecimalFormat("0.00"); // 2 знака после запятой
        return df.format(value);
    }
}

// Метод форматирования объема с суффиксами M и B
```

```
    private String formatVolume(double value) {
        if (Double.isNaN(value) || Double.isInfinite(value)) {
            return "Invalid USDT";
        }
        if (value >= 1_000_000_000) {
            DecimalFormat df = new DecimalFormat("0.00");
            return df.format(value / 1_000_000_000) + "B USDT";
        } else if (value >= 1_000_000) {
            DecimalFormat df = new DecimalFormat("0.00");
            return df.format(value / 1_000_000) + "M USDT";
        } else {
            DecimalFormat df = new DecimalFormat("0.00");
            return df.format(value) + " USDT";
        }
    }

    public RecordsContainerDto findAllRecords(String filterMode, boolean forceRefresh) {
        String cacheFile = CACHE_DIR + CACHE_FILE_PREFIX + "cryptoData.json";
        List<Record> records = loadFromCache(cacheFile);

        if (!forceRefresh && records != null && !isCacheExpired(cacheFile)) {
            System.out.println("Returning cached data from: " + cacheFile);
            return new RecordsContainerDto(records);
        }

        System.out.println("Fetching fresh data for findAllRecords...");
        records = fetchCryptoData();
        if (!records.isEmpty()) {
            System.out.println("Saving " + records.size() + " records to cache: " + cacheFile);
            saveToCache(cacheFile, records);
        } else {
            System.err.println("No records fetched for findAllRecords, skipping cache save.");
        }
        return new RecordsContainerDto(records);
    }

    public RecordsContainerDto findExchangeRecords(String exchange, String filterMode,
boolean forceRefresh) {
        String cacheFile = CACHE_DIR + CACHE_FILE_PREFIX + exchange + ".json";
```

```
        List<Record> records = loadFromCache(cacheFile);

        if (!forceRefresh && records != null && !isCacheExpired(cacheFile)) {
            System.out.println("Returning cached data from: " + cacheFile);
            return new RecordsContainerDto(records);
        }

        System.out.println("Fetching fresh data for exchange: " + exchange);
        records = fetchCryptoDataForExchange(exchange);
        if (!records.isEmpty()) {
            System.out.println("Saving " + records.size() + " records to cache: " + cacheFile);
            saveToCache(cacheFile, records);
        } else {
            System.err.println("No records fetched for exchange " + exchange + ", skipping cache
save.");
        }
        return new RecordsContainerDto(records);
    }

    private List<Record> fetchCryptoData() {
        String url =
"https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=market_cap_desc&per_page=
100&page=1&sparkline=false&price_change_percentage=1h,24h,7d";
        HttpHeaders headers = new HttpHeaders();
        headers.set("x-cg-api-key", "CG-WkWLv5AULbRVVACEgZ5RXEu8");
        HttpEntity<String> entity = new HttpEntity<>(headers);

        try {
            System.out.println("Fetching data from CoinGecko API...");
            ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.GET, entity,
String.class);
            JsonNode jsonNode = objectMapper.readTree(response.getBody());
            List<Record> records = new ArrayList<>();

            double usdToBynRate = getUsdToBynRate();

            for (JsonNode coin : jsonNode) {
                String id = coin.get("id").asText();
                String name = coin.get("name").asText();
                String symbol = coin.get("symbol").asText().toUpperCase();
```

```
                double currentPrice = coin.get("current_price").asDouble();
                double marketCap = coin.get("market_cap").asDouble();
                double priceChange1h = coin.has("price_change_percentage_1h_in_currency") ? coin.get("price_change_percentage_1h_in_currency").asDouble() : 0.0;
                double priceChange24h = coin.has("price_change_percentage_24h_in_currency") ? coin.get("price_change_percentage_24h_in_currency").asDouble() : 0.0;
                double priceChange7d = coin.has("price_change_percentage_7d_in_currency") ? coin.get("price_change_percentage_7d_in_currency").asDouble() : 0.0;
                double volume = 0.0; // CoinGecko не используется для Huobi, поэтому объем не заполняется здесь

                double currentPriceByn = currentPrice * usdToBynRate;
                double marketCapByn = marketCap * usdToBynRate;

                records.add(new Record(id, name, symbol, currentPrice, currentPriceByn, marketCap, marketCapByn, priceChange1h, priceChange24h, priceChange7d, volume));
                System.out.println("Processed ticker for " + symbol + ": price=" + formatPrice(currentPrice) + " USD, priceBYN=" + formatPrice(currentPriceByn) + " BYN, marketCap=" + marketCap + " USD, change1h=" + formatPrice(priceChange1h) + "%, change24h=" + formatPrice(priceChange24h) + "%, change7d=" + formatPrice(priceChange7d) + "%");
            }
            System.out.println("Successfully fetched " + records.size() + " records from CoinGecko");
            return records;
        } catch (Exception e) {
            System.err.println("Error fetching data from CoinGecko: " + e.getMessage());
            e.printStackTrace();
            return List.of();
        }
    }

    private List<Record> fetchCryptoDataForExchange(String exchange) {
        if (exchange.equals("binance")) {
            String url = "https://api.binance.com/api/v3/ticker/24hr";
            try {
                System.out.println("Fetching data from Binance API...");
                ResponseEntity<String> response = restTemplate.getForEntity(url, String.class);
                JsonNode jsonArray = objectMapper.readTree(response.getBody());
                List<Record> records = new ArrayList<>();

                double usdToBynRate = getUsdToBynRate();
```

```
                for (JsonNode ticker : jsonArray) {
                    String symbol = ticker.get("symbol").asText();
                    if (!symbol.endsWith("USDT")) {
                        continue;
                    }

                    String coinSymbol = symbol.replace("USDT", "").toUpperCase();
                    String name = coinSymbol;
                    String id = coinSymbol.toLowerCase();
                    double currentPrice = ticker.get("lastPrice").asDouble();
                    double priceChange24h = ticker.get("priceChangePercent").asDouble();
                    double marketCap = 0.0;
                    double priceChange1h = 0.0;
                    double priceChange7d = 0.0;
                    double volume = ticker.get("quoteVolume").asDouble(); // Объем в USDT

                    double currentPriceByn = currentPrice * usdToBynRate;
                    double marketCapByn = marketCap * usdToBynRate;

                    if (currentPrice == 0 || volume == 0) {
                        System.err.println("Skipping " + coinSymbol + " due to zero price or volume");
                        continue;
                    }

                    records.add(new Record(id, name, coinSymbol, currentPrice, currentPriceByn,
marketCap, marketCapByn, priceChange1h, priceChange24h, priceChange7d, volume));
                    System.out.println("Processed ticker for " + coinSymbol + ": price=" +
formatPrice(currentPrice) + " USD, priceBYN=" + formatPrice(currentPriceByn) + " BYN, change24h="
+ formatPrice(priceChange24h) + "%, volume=" + formatVolume(volume));
                }
                System.out.println("Successfully fetched " + records.size() + " records from Binance");
                return records;
            } catch (Exception e) {
                System.err.println("Error fetching data from Binance: " + e.getMessage());
                e.printStackTrace();
                return List.of();
            }
        } else if (exchange.equals("bybit")) {
            String url = "https://api.bybit.com/v5/market/tickers?category=spot";
```

```
        try {
            System.out.println("Fetching data from ByBit API...");
            ResponseEntity<String> response = restTemplate.getForEntity(url, String.class);
            JsonNode jsonNode = objectMapper.readTree(response.getBody());
            List<Record> records = new ArrayList<>();

            double usdToBynRate = getUsdToBynRate();

            for (JsonNode ticker : jsonNode.get("result").get("list")) {
                String symbol = ticker.get("symbol").asText();
                if (!symbol.endsWith("USDT")) {
                    continue;
                }

                String coinSymbol = symbol.replace("USDT", "").toUpperCase();
                String name = coinSymbol;
                String id = coinSymbol.toLowerCase();
                double currentPrice = ticker.get("lastPrice").asDouble();
                double priceChange24h = ticker.has("price24hPcnt") ?
ticker.get("price24hPcnt").asDouble() : 0.0;
                double marketCap = 0.0;
                double priceChange1h = 0.0;
                double priceChange7d = 0.0;
                double volume = ticker.get("turnover24h").asDouble(); // Объем в USDT

                double currentPriceByn = currentPrice * usdToBynRate;
                double marketCapByn = marketCap * usdToBynRate;

                if (currentPrice == 0 || volume == 0) {
                    System.err.println("Skipping " + coinSymbol + " due to zero price or volume");
                    continue;
                }

                records.add(new Record(id, name, coinSymbol, currentPrice, currentPriceByn,
marketCap, marketCapByn, priceChange1h, priceChange24h, priceChange7d, volume));
                System.out.println("Processed ticker for " + coinSymbol + ": price=" +
formatPrice(currentPrice) + " USD, priceBYN=" + formatPrice(currentPriceByn) + " BYN, change24h="
+ formatPrice(priceChange24h) + "%, volume=" + formatVolume(volume));
            }
            System.out.println("Successfully fetched " + records.size() + " records from ByBit");
```

```
                return records;
            } catch (Exception e) {
                System.err.println("Error fetching data from ByBit: " + e.getMessage());
                e.printStackTrace();
                return List.of();
            }
        } else if (exchange.equals("kraken")) {
            String url =
"https://api.kraken.com/0/public/Ticker?pair=XBTUSD,ETHUSD,LTCUSD,XRPUSD,ADAUSD,BCHU
SD,DASHUSD,DOTUSD,EOSUSD,LINKUSD";
            try {
                System.out.println("Fetching data from Kraken API...");
                ResponseEntity<String> response = restTemplate.getForEntity(url, String.class);
                JsonNode jsonNode = objectMapper.readTree(response.getBody());
                List<Record> records = new ArrayList<>();

                double usdToBynRate = getUsdToBynRate();

                JsonNode resultNode = jsonNode.get("result");
                Iterator<String> pairNames = resultNode.fieldNames();
                while (pairNames.hasNext()) {
                    String pairName = pairNames.next();
                    if (!pairName.endsWith("ZUSD")) {
                        continue;
                    }

                    JsonNode ticker = resultNode.get(pairName);

                    String coinSymbol = pairName.startsWith("X") ? pairName.substring(1,
pairName.length() - 4) : pairName.replace("ZUSD", "");
                    if (coinSymbol.equals("XBT")) {
                        coinSymbol = "BTC";
                    }
                    String name = coinSymbol;
                    String id = coinSymbol.toLowerCase();
                    double currentPrice = ticker.get("c").get(0).asDouble();
                    double openPrice = ticker.get("o").asDouble();
                    double priceChange24h = (openPrice != 0) ? ((currentPrice - openPrice) / openPrice)
* 100 : 0.0;
                    double marketCap = 0.0;
```

```
                    double priceChange1h = 0.0;
                    double priceChange7d = 0.0;
                    double volume = ticker.get("v").get(0).asDouble() * currentPrice; // Объем за 24 часа в USDT

                    double currentPriceByn = currentPrice * usdToBynRate;
                    double marketCapByn = marketCap * usdToBynRate;

                    if (currentPrice == 0 || volume == 0) {
                        System.err.println("Skipping " + coinSymbol + " due to zero price or volume");
                        continue;
                    }

                    records.add(new Record(id, name, coinSymbol, currentPrice, currentPriceByn, marketCap, marketCapByn, priceChange1h, priceChange24h, priceChange7d, volume));
                    System.out.println("Processed ticker for " + coinSymbol + ": price=" + formatPrice(currentPrice) + " USD, priceBYN=" + formatPrice(currentPriceByn) + " BYN, change24h=" + formatPrice(priceChange24h) + "%, volume=" + formatVolume(volume));
                }
                System.out.println("Successfully fetched " + records.size() + " records from Kraken");
                return records;
            } catch (Exception e) {
                System.err.println("Error fetching data from Kraken: " + e.getMessage());
                e.printStackTrace();
                return List.of();
            }
        } else if (exchange.equals("bitfinex")) {
            String url = "https://api-pub.bitfinex.com/v2/tickers?symbols=tBTCUSD,tETHUSD,tLTCUSD,tXRPUSD,tADAUSD,tDOTUSD,tEOSUSD,tLINKUSD";
            try {
                System.out.println("Fetching data from Bitfinex API...");
                ResponseEntity<String> response = restTemplate.getForEntity(url, String.class);
                JsonNode jsonArray = objectMapper.readTree(response.getBody());
                List<Record> records = new ArrayList<>();

                if (!jsonArray.isArray() || jsonArray.size() == 0) {
                    System.err.println("Bitfinex API returned an invalid or empty response");
                    return List.of();
                }
```

```
                double usdToBynRate = getUsdToBynRate();

                for (JsonNode ticker : jsonArray) {
                    if (!ticker.isArray() || ticker.size() != 11 || !ticker.get(0).isTextual()) {
                        System.err.println("Invalid ticker data for Bitfinex");
                        continue;
                    }

                    String symbol = ticker.get(0).asText().replace("t", "");
                    String coinSymbol = symbol.substring(0, symbol.length() - 3);
                    String name = coinSymbol;
                    String id = coinSymbol.toLowerCase();
                    double currentPrice = ticker.get(7).asDouble();
                    double priceChange24h = ticker.get(6).asDouble();
                    double marketCap = 0.0;
                    double priceChange1h = 0.0;
                    double priceChange7d = 0.0;
                    double volume = ticker.get(8).asDouble() * currentPrice; // Объем в USDT

                    double currentPriceByn = currentPrice * usdToBynRate;
                    double marketCapByn = marketCap * usdToBynRate;

                    if (currentPrice == 0 || volume == 0) {
                        System.err.println("Skipping " + coinSymbol + " due to zero price or volume");
                        continue;
                    }

                    records.add(new Record(id, name, coinSymbol, currentPrice, currentPriceByn,
marketCap, marketCapByn, priceChange1h, priceChange24h, priceChange7d, volume));
                    System.out.println("Processed ticker for " + coinSymbol + ": price=" +
formatPrice(currentPrice) + " USD, priceBYN=" + formatPrice(currentPriceByn) + " BYN, change24h="
+ formatPrice(priceChange24h) + "%, volume=" + formatVolume(volume));
                }
                System.out.println("Successfully fetched " + records.size() + " records from Bitfinex");
                return records;
            } catch (Exception e) {
                System.err.println("Error fetching data from Bitfinex: " + e.getMessage());
                e.printStackTrace();
```

```
            return List.of();
        }
    } else if (exchange.equals("huobi")) {
        String url = "https://api.huobi.pro/market/tickers";
        try {
            System.out.println("Fetching data from Huobi API...");
            ResponseEntity<String> response = restTemplate.getForEntity(url, String.class);
            JsonNode jsonNode = objectMapper.readTree(response.getBody());
            List<Record> records = new ArrayList<>();

            if (!jsonNode.has("data") || !jsonNode.get("data").isArray()) {
                System.err.println("Huobi API returned invalid data structure");
                return List.of();
            }

            double usdToBynRate = getUsdToBynRate();

            for (JsonNode ticker : jsonNode.get("data")) {
                String symbol = ticker.get("symbol").asText().toLowerCase();
                if (!symbol.endsWith("usdt")) {
                    continue;
                }

                String coinSymbol = symbol.replace("usdt", "").toUpperCase();
                String name = coinSymbol; // Временное решение, можно улучшить через
дополнительный API
                String id = coinSymbol.toLowerCase();
                double currentPrice = ticker.get("close").asDouble();
                double openPrice = ticker.has("open") ? ticker.get("open").asDouble() : 0.0;
                double priceChange24h = (openPrice != 0) ? ((currentPrice - openPrice) / openPrice)
* 100 : 0.0;
                double marketCap = 0.0;
                double priceChange1h = 0.0;
                double priceChange7d = 0.0;
                double volume = ticker.get("vol").asDouble(); // Объем в USDT

                double currentPriceByn = currentPrice * usdToBynRate;
                double marketCapByn = marketCap * usdToBynRate;
```

```
                    if (currentPrice == 0 || volume == 0) {
                        System.err.println("Skipping " + coinSymbol + " due to zero price or volume");
                        continue;
                    }

                    records.add(new Record(id, name, coinSymbol, currentPrice, currentPriceByn,
marketCap, marketCapByn, priceChange1h, priceChange24h, priceChange7d, volume));
                    System.out.println("Processed ticker for " + coinSymbol + ": price=" +
formatPrice(currentPrice) + " USD, priceBYN=" + formatPrice(currentPriceByn) + " BYN, change24h="
+ formatPrice(priceChange24h) + "%, volume=" + formatVolume(volume));
                }
                System.out.println("Successfully fetched " + records.size() + " records from Huobi");
                return records;
            } catch (Exception e) {
                System.err.println("Error fetching data from Huobi: " + e.getMessage());
                e.printStackTrace();
                return List.of();
            }
        } else {
            System.err.println("Exchange " + exchange + " is not supported yet.");
            return List.of();
        }
    }

    private double getUsdToBynRate() {
        // Проверяем кэш курса
        Double cachedRate = loadRateFromCache();
        if (cachedRate != null && !isCacheExpired(USD_BYN_RATE_CACHE_FILE)) {
            System.out.println("Returning cached USD to BYN rate: " + cachedRate);
            return cachedRate;
        }

        // Запрашиваем курс через API НБРБ
        String url = "https://api.nbrb.by/exrates/rates/431?parammode=0";
        try {
            System.out.println("Fetching USD to BYN rate from NBRB API...");
            ResponseEntity<String> response = restTemplate.getForEntity(url, String.class);
            JsonNode jsonNode = objectMapper.readTree(response.getBody());
            double rate = jsonNode.get("Cur_OfficialRate").asDouble();
```

```
                System.out.println("USD to BYN rate: " + rate);

                // Сохраняем курс в кэш
                saveRateToCache(rate);
                return rate;
            } catch (Exception e) {
                System.err.println("Error fetching USD to BYN rate from NBRB API: " +
e.getMessage());
                // Если есть кэшированный курс, используем его, даже если он устарел
                if (cachedRate != null) {
                    System.out.println("Using expired cached USD to BYN rate due to API failure: " +
cachedRate);
                    return cachedRate;
                }
                // Значение по умолчанию, если API недоступен и кэша нет
                System.err.println("Falling back to default USD to BYN rate: 3.20");
                return 3.20;
            }
        }

        private Double loadRateFromCache() {
            try {
                File file = new File(USD_BYN_RATE_CACHE_FILE);
                if (!file.exists()) {
                    System.out.println("USD to BYN rate cache file does not exist: " +
USD_BYN_RATE_CACHE_FILE);
                    return null;
                }
                System.out.println("Loading USD to BYN rate from cache file: " +
USD_BYN_RATE_CACHE_FILE);
                String json = new
String(Files.readAllBytes(Paths.get(USD_BYN_RATE_CACHE_FILE)));
                JsonNode jsonNode = objectMapper.readTree(json);
                double rate = jsonNode.get("rate").asDouble();
                System.out.println("Loaded USD to BYN rate from cache: " + rate);
                return rate;
            } catch (Exception e) {
                System.err.println("Error loading USD to BYN rate from cache: " + e.getMessage());
                return null;
            }
```

```
    }

    private void saveRateToCache(double rate) {
        try {
            System.out.println("Saving USD to BYN rate to cache file: " +
USD_BYN_RATE_CACHE_FILE);
            String json = objectMapper.writeValueAsString(new RateCache(rate));
            Files.write(Paths.get(USD_BYN_RATE_CACHE_FILE), json.getBytes());
            System.out.println("Successfully saved USD to BYN rate to cache: " + rate);
        } catch (Exception e) {
            System.err.println("Error saving USD to BYN rate to cache: " + e.getMessage());
        }
    }

    private List<Record> loadFromCache(String cacheFile) {
        try {
            File file = new File(cacheFile);
            if (!file.exists()) {
                System.out.println("Cache file does not exist: " + cacheFile);
                return null;
            }
            System.out.println("Loading from cache file: " + cacheFile);
            String json = new String(Files.readAllBytes(Paths.get(cacheFile)));
            Record[] recordsArray = objectMapper.readValue(json, Record[].class);
            System.out.println("Loaded " + recordsArray.length + " records from cache");
            return new ArrayList<>(Arrays.asList(recordsArray));
        } catch (Exception e) {
            System.err.println("Error loading from cache: " + e.getMessage());
            e.printStackTrace();
            return null;
        }
    }

    private void saveToCache(String cacheFile, List<Record> records) {
        try {
            System.out.println("Saving to cache file: " + cacheFile);
            String json = objectMapper.writeValueAsString(records);
            Files.write(Paths.get(cacheFile), json.getBytes());
            System.out.println("Successfully saved " + records.size() + " records to cache");
```

```
            } catch (Exception e) {
                System.err.println("Error saving to cache: " + e.getMessage());
                e.printStackTrace();
            }
        }

        private boolean isCacheExpired(String cacheFile) {
            File file = new File(cacheFile);
            if (!file.exists()) {
                System.out.println("Cache file does not exist, treating as expired: " + cacheFile);
                return true;
            }
            long lastModified = file.lastModified();
            boolean expired = System.currentTimeMillis() - lastModified > CACHE_TTL;
            System.out.println("Cache file " + cacheFile + " last modified: " + lastModified + ", expired:
" + expired);
            return expired;
        }

        // Вспомогательный класс для кэширования курса
        private static class RateCache {
            private final double rate;

            public RateCache(double rate) {
                this.rate = rate;
            }

            public double getRate() {
                return rate;
            }
        }
    }
```

Main-page.css:

```
/* Base styles */
body {
    margin: 0;
    font-family: 'Inter', sans-serif;
    background: linear-gradient(135deg, #13151a 0%, #1e2128 100%);
    color: #fff;
```

```
    min-height: 100vh;
    display: flex;
    flex-direction: column;
}

.page-wrapper {
    flex: 1;
    display: flex;
    flex-direction: column;
}

/* Navbar */
.navbar {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 1.5rem 2rem;
    background: rgba(255, 255, 255, 0.03);
    backdrop-filter: blur(10px);
    border-bottom: 1px solid rgba(255, 255, 255, 0.1);
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    z-index: 1000;
}

.navbar-brand {
    font-size: 1.5rem;
    font-weight: 700;
    background: linear-gradient(90deg, #3A87E0, #64E0FF);
    -webkit-background-clip: text;
    background-clip: text;
    -webkit-text-fill-color: transparent;
    cursor: pointer;
}

.navbar-menu {
```

```
    display: flex;
    gap: 2rem;
}

.navbar-menu a {
    color: #fff;
    text-decoration: none;
    font-weight: 500;
    padding: 0.5rem 1rem;
    border-radius: 8px;
    transition: all 0.3s ease;
}

.navbar-menu a.active {
    background: rgba(58, 135, 224, 0.1);
    color: #3A87E0;
}

.navbar-menu a:hover {
    background: rgba(255, 255, 255, 0.1);
}

/* Main Content */
.main-content {
    flex: 1;
    padding: 8rem 2rem 2rem;
}

.hero-section {
    text-align: center;
    margin-bottom: 3rem;
}

.hero-section h1 {
    font-size: 2.5rem;
    margin: 0;
    background: linear-gradient(90deg, #fff, #a5c5e7);
    -webkit-background-clip: text;
```

```
    background-clip: text;
    -webkit-text-fill-color: transparent;
}

.subtitle {
    color: #8b95a5;
    margin-top: 0.5rem;
}

/* Контейнер для фильтров */
.filters-container {
    display: flex;
    gap: 1rem;
    margin-bottom: 2rem;
    padding: 1rem;
    background: rgba(255, 255, 255, 0.03);
    border-radius: 12px;
    align-items: center;
    flex-wrap: wrap;
}

.filter-group {
    display: flex;
    align-items: center;
    gap: 0.5rem;
}

.filter-input {
    background: rgba(255, 255, 255, 0.05);
    border: 1px solid rgba(255, 255, 255, 0.1);
    border-radius: 8px;
    padding: 0.5rem 1rem;
    color: #fff;
    font-size: 0.875rem;
    width: 200px;
}

.filter-input::placeholder {
```

```
        color: rgba(255, 255, 255, 0.5);
    }

    .filter-label {
        color: #8b95a5;
        font-size: 0.875rem;
    }

    .filter-select {
        background: rgba(255, 255, 255, 0.05);
        border: 1px solid rgba(255, 255, 255, 0.1);
        border-radius: 8px;
        padding: 0.5rem 1rem;
        color: #fff;
        font-size: 0.875rem;
        width: 200px;
        appearance: none;
        -webkit-appearance: none;
        -moz-appearance: none;
        background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 24 24' fill='none' stroke='white' stroke-width='2'
stroke-linecap='round' stroke-linejoin='round'%3e%3cpolyline points='6 9 12 15 18
9'%3e%3c/polyline%3e%3c/svg%3e");
        background-repeat: no-repeat;
        background-position: right 1rem center;
        background-size: 1em;
    }

    .filter-select:focus {
        outline: none;
        border-color: #3A87E0;
        box-shadow: 0 0 0 2px rgba(58, 135, 224, 0.2);
    }

    /* Контейнер для скролла */
    .crypto-scroll-container {
        width: 100%;
        overflow-x: auto;
        padding: 0.5rem 0 1.5rem 0;
```

```
    margin: 0 -2rem;
    padding-left: 2rem;
    padding-right: 2rem;
    position: relative;
    -webkit-mask-image: linear-gradient(
        to right,
        transparent 0%,
        black 5%,
        black 95%,
        transparent 100%
    );
    mask-image: linear-gradient(
        to right,
        transparent 0%,
        black 5%,
        black 95%,
        transparent 100%
    );
    scrollbar-width: thin;
    scrollbar-color: rgba(255, 255, 255, 0.2) transparent;
}

.crypto-scroll-container::-webkit-scrollbar {
    height: 6px;
}

.crypto-scroll-container::-webkit-scrollbar-track {
    background: transparent;
}

.crypto-scroll-container::-webkit-scrollbar-thumb {
    background-color: rgba(255, 255, 255, 0.2);
    border-radius: 3px;
}

/* Cards Wrapper */
.crypto-cards-wrapper {
    display: flex;
```

```
    gap: 1.75rem;
    padding: 0.5rem;
    min-width: min-content;
}

/* Карточки */
.crypto-card {
    background: rgba(255, 255, 255, 0.03);
    border: 1px solid rgba(255, 255, 255, 0.1);
    border-radius: 16px;
    padding: 1.75rem;
    min-width: 320px;
    transition: all 0.3s ease;
    cursor: pointer;
}

.crypto-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 8px 24px rgba(0, 0, 0, 0.2);
    background: rgba(255, 255, 255, 0.05);
}

.crypto-card-header {
    display: flex;
    align-items: center;
    margin-bottom: 1.5rem;
}

.crypto-icon {
    width: 56px;
    height: 56px;
    border-radius: 50%;
    margin-right: 1.25rem;
    background-size: cover;
}

/* Иконки для криптовалют */
.bitcoin { background: linear-gradient(45deg, #F7931A, #FFAB2E); }
```

```
.ethereum { background: linear-gradient(45deg, #627EEA, #8CA3F2); }
.tether { background: linear-gradient(45deg, #26A17B, #4ACCAF); }
.binance-coin { background: linear-gradient(45deg, #F3BA2F, #FFD966); }
.solana { background: linear-gradient(45deg, #9945FF, #14F195); }
.xrp { background: linear-gradient(45deg, #23292F, #3D4852); }
.avalanche { background: linear-gradient(45deg, #E84142, #FF6B6B); }
.chainlink { background: linear-gradient(45deg, #2A5ADA, #4D7FFF); }
.polygon { background: linear-gradient(45deg, #8247E5, #B47FFF); }
.cosmos { background: linear-gradient(45deg, #2E3148, #5B6078); }
.cardano { background: linear-gradient(45deg, #0033AD, #00A3E0); }
.dogecoin { background: linear-gradient(45deg, #C2A633, #EAD861); }
.polkadot { background: linear-gradient(45deg, #E6007A, #FF5A9E); }
.tron { background: linear-gradient(45deg, #EF3038, #FF666A); }
.stellar { background: linear-gradient(45deg, #0D1C2E, #3B5A9A); }
.ripple { background: linear-gradient(45deg, #23292F, #3D4852); }
.litecoin { background: linear-gradient(45deg, #345D9D, #A3BFFA); }
.bitcoin-cash { background: linear-gradient(45deg, #0AC18E, #4DE8B4); }
.uniswap { background: linear-gradient(45deg, #FF007A, #FF66A1); }

/* Иконки для бирж */
.binance { background: #FFC107; }
.coinbase-pro { background: #1A73E8; }
.kraken { background: #333333; }
.bitfinex { background: #00C853; }
.huobi { background: #D32F2F; }

.crypto-
icon:not([class*='bitcoin']):not([class*='ethereum']):not([class*='tether']):not([class*='binance-
coin']):not([class*='solana']):not([class*='xrp']):not([class*='avalanche']):not([class*='chainlink']):not([cl
ass*='polygon']):not([class*='cosmos']):not([class*='cardano']):not([class*='dogecoin']):not([class*='polk
adot']):not([class*='tron']):not([class*='stellar']):not([class*='ripple']):not([class*='litecoin']):not([class*='
bitcoin-cash']):not([class*='uniswap']):not([class*='binance']):not([class*='coinbase-
pro']):not([class*='kraken']):not([class*='bitfinex']):not([class*='huobi']) {
    background: linear-gradient(45deg, #4A4A4A, #6A6A6A);
}

.crypto-basic-info h3 {
    margin: 0;
    font-size: 1.4rem;
    font-weight: 600;
```

```
}

.ticker {
    color: #8b95a5;
    font-size: 0.875rem;
}

.price-section {
    display: flex;
    justify-content: space-between;
    align-items: center;
    flex-wrap: wrap;
    gap: 0.5rem;
}

.current-price, .current-price-byn {
    font-size: 1.25rem;
    font-weight: 600;
}

.current-price-byn {
    color: #8b95a5;
}

.price-changes {
    display: flex;
    justify-content: space-between;
    margin-top: 0.5rem;
}

.price-change-wrapper {
    display: flex;
    align-items: center;
    gap: 0.25rem;
}

.price-change-label {
    color: #8b95a5;
```

```
    font-size: 0.875rem;
}

.price-change {
    padding: 0.25rem 0.75rem;
    border-radius: 100px;
    font-size: 0.875rem;
    font-weight: 500;
}

.price-change.positive {
    background: rgba(52, 199, 89, 0.1);
    color: #34c759;
}

.price-change.negative {
    background: rgba(255, 59, 48, 0.1);
    color: #ff3b30;
}

.market-cap {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding-top: 0.75rem;
    margin-top: 0.75rem;
    border-top: 1px solid rgba(255, 255, 255, 0.1);
    flex-wrap: wrap;
    gap: 0.5rem;
}

.market-cap-label {
    color: #8b95a5;
    font-size: 0.875rem;
}

.market-cap-value, .market-cap-value-byn {
    font-weight: 500;
```

```
}

.market-cap-value-byn {
    color: #8b95a5;
}

/* Footer */
.footer {
    text-align: center;
    padding: 0.75rem;
    background: rgba(255, 255, 255, 0.02);
    border-top: 1px solid rgba(255, 255, 255, 0.05);
}

.footer-content {
    display: flex;
    justify-content: center;
    align-items: center;
    gap: 2rem;
    color: #8b95a5;
    font-size: 0.75rem;
}

.footer-links {
    display: flex;
    gap: 1rem;
}

.footer-links a {
    color: #3A87E0;
    text-decoration: none;
    transition: color 0.2s;
}

.footer-links a:hover {
    color: #64E0FF;
}
```

```
/* Responsive Design */
@media (max-width: 768px) {
    .navbar {
        padding: 1rem;
    }

    .navbar-menu {
        gap: 1rem;
    }

    .main-content {
        padding: 6rem 1rem 1rem;
    }

    .hero-section h1 {
        font-size: 2rem;
    }

    .crypto-card {
        min-width: 260px;
        padding: 1.25rem;
    }

    .filter-input, .filter-select {
        width: 150px;
    }
}
```