



Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.03 Прикладная информатика

О Т Ч Е Т

по индивидуальному заданию

Название: Разработка микросервиса экспорта данных

Дисциплина: Проектирование информационных систем

Студент

ИУ6-75Б

(Группа)

(Подпись, дата)

А. А. Панькив

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

А.Ф. Прутик

(И.О. Фамилия)

Москва, 2025

Цель работы: Разработка автономного микросервиса для системы Parser, предназначенного для управления задачами парсинга веб-сайтов. Микросервис обеспечивает создание, хранение, обновление и удаление задач парсинга, а также управление их параметрами и статусами выполнения.

Требования к системе:

Разрабатываемая система должна обеспечивать возможность создания и управления задачами парсинга, каждая из которых описывает отдельную сессию парсинга веб-ресурса. Система должна предоставлять API для получения списка задач с возможностью фильтрации и просмотра детальной информации по каждой задаче.

Для каждой задачи необходимо хранить настраиваемые параметры, включая название задачи, URL сайта, тип сайта и критерии сбора данных в формате JSON. Также система должна поддерживать управление статусом задачи (создана, в работе, приостановлена, завершена, ошибка).

Сервис должен предоставлять REST API для интеграции с другими компонентами системы, такими как парсер-воркеры и планировщики задач, а также веб-интерфейс для конечных пользователей. Микросервис должен работать независимо от других сервисов системы и обеспечивать быстрое время отклика API за счёт использования кеширования.

Описание решения и используемые технологии:

Для реализации системы выбрана микросервисная архитектура с использованием контейнеризации. В качестве бэкенд-фреймворка используется FastAPI на языке Python, который обеспечивает высокую производительность за счёт асинхронной модели обработки запросов, автоматическую генерацию документации в форматах Swagger/OpenAPI, а также строгую типизацию данных с использованием Pydantic.

Для хранения данных применяется реляционная база данных PostgreSQL, обеспечивающая надёжное и масштабируемое хранение информации о задачах парсинга. Взаимодействие с базой данных реализовано с использованием ORM

SQLAlchemy в асинхронном режиме, что позволяет эффективно работать с большим количеством запросов.

Для ускорения операций чтения используется Redis, применяемый в качестве кеша для хранения данных о задачах по идентификатору с ограниченным временем жизни. Это позволяет снизить нагрузку на базу данных и обеспечить быстрый отклик API.

Фронтенд приложения реализован в виде одностраничного приложения (SPA) с использованием React и TypeScript, собранного с помощью инструмента Vite. Пользовательский интерфейс позволяет создавать задачи, просматривать их список, фильтровать по параметрам, изменять статус и удалять задачи без перезагрузки страницы.

Вся инфраструктура проекта развёртывается в контейнерах Docker и управляется с помощью Docker Compose, что обеспечивает воспроизводимость окружения и упрощает запуск системы. Код бэкенда структурирован по слоям для обеспечения читаемости, поддерживаемости и расширяемости. API Layer содержит HTTP-эндпоинты, реализующие REST API и выполняющие валидацию входных данных. Service Layer инкапсулирует бизнес-логику управления задачами, включая работу с кешем и обработку статусов. Data Layer представлен ORM-моделью Task и связанными перечислениями, обеспечивающими работу с базой данных PostgreSQL.

1 Архитектурные схемы

На рисунке 1 показана диаграмма контекста, которая показывает систему в окружении, выделяя основные внешние системы и пользователей, которые взаимодействуют с ней. На рисунке 2 – диаграмма контейнеров, показывающая взаимодействие микросервисов и сторонних систем. На рисунке 3 отображена диаграмма компонентов реализованного микросервиса задач. На 4 рисунке показана диаграмма классов и структура кода в целом.

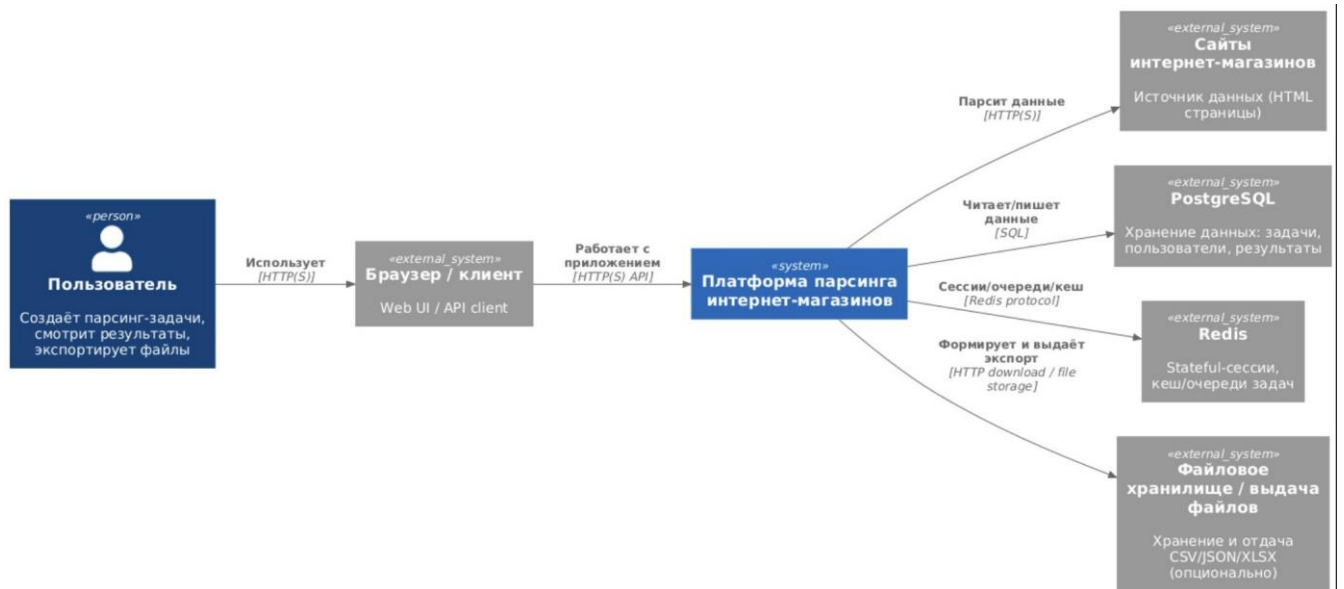


Рисунок 1 – Диаграмма контекста

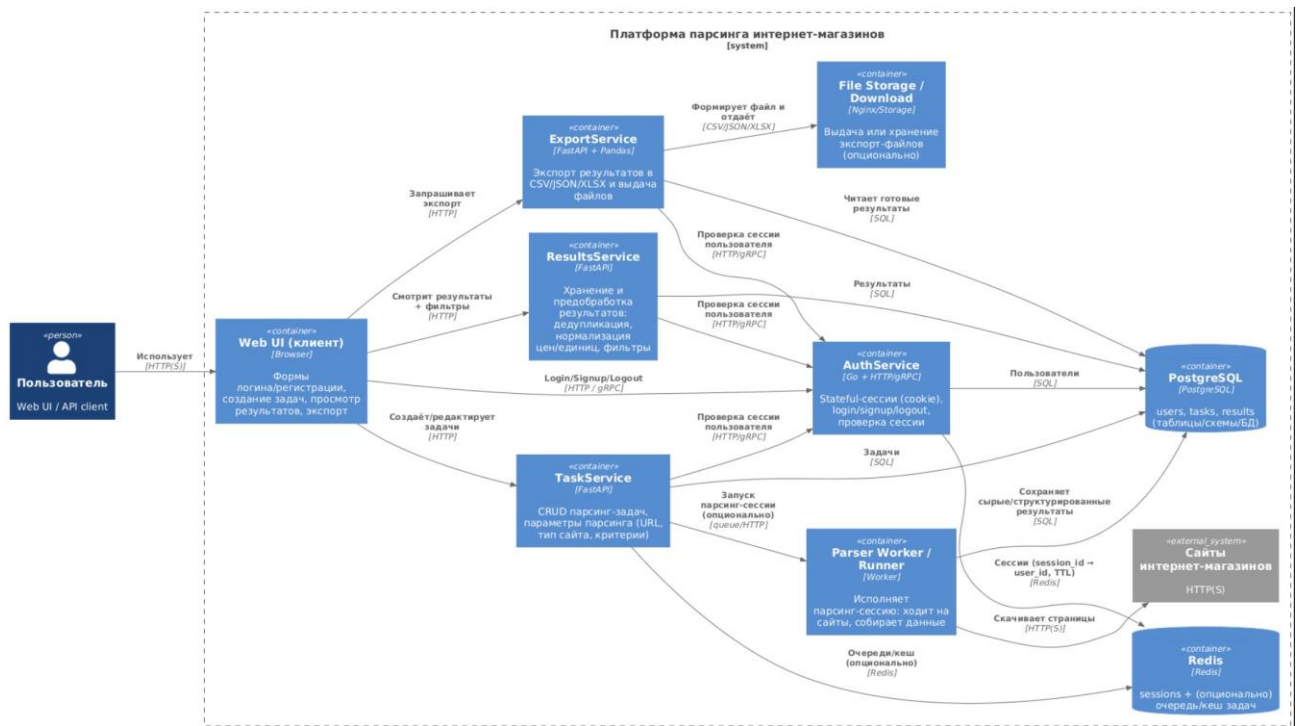


Рисунок 2 – Диаграмма контейнеров

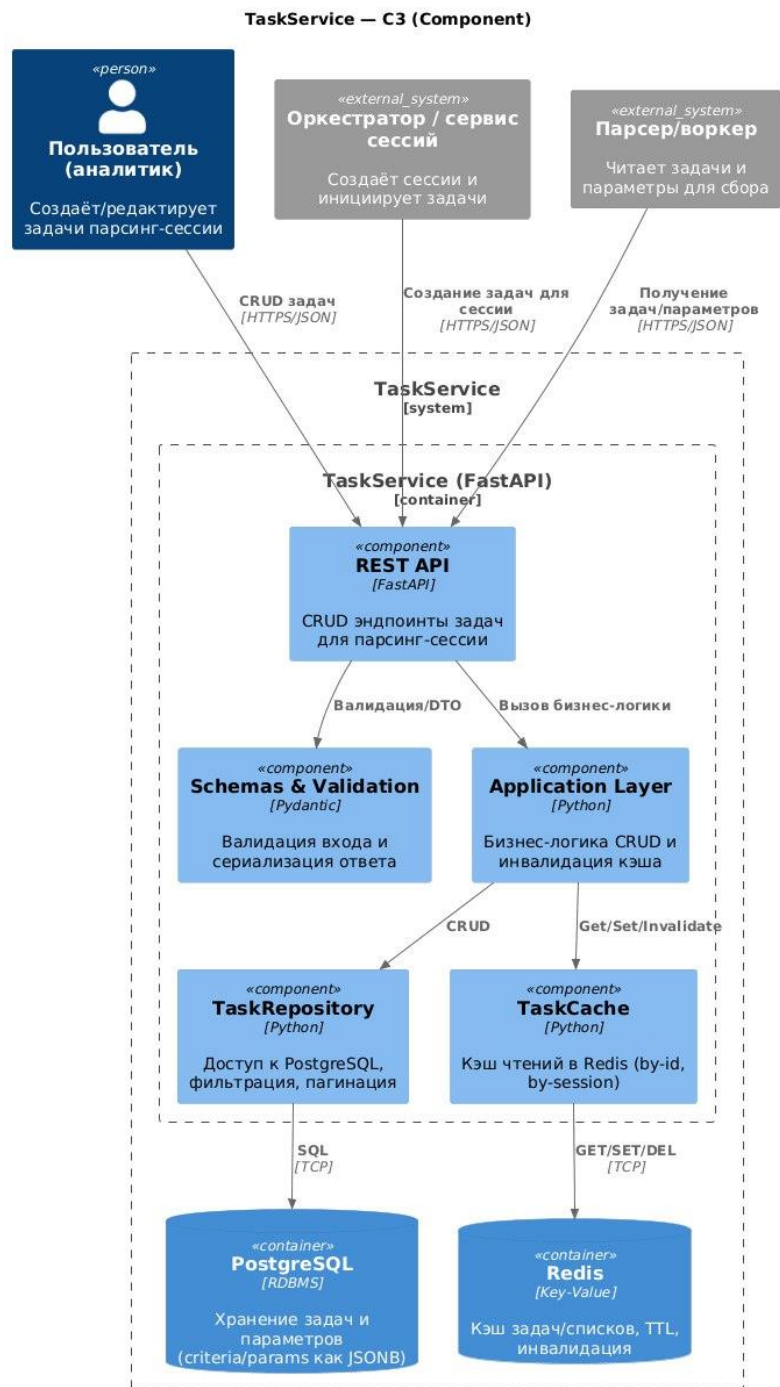


Рисунок 3 – Диаграмма компонентов

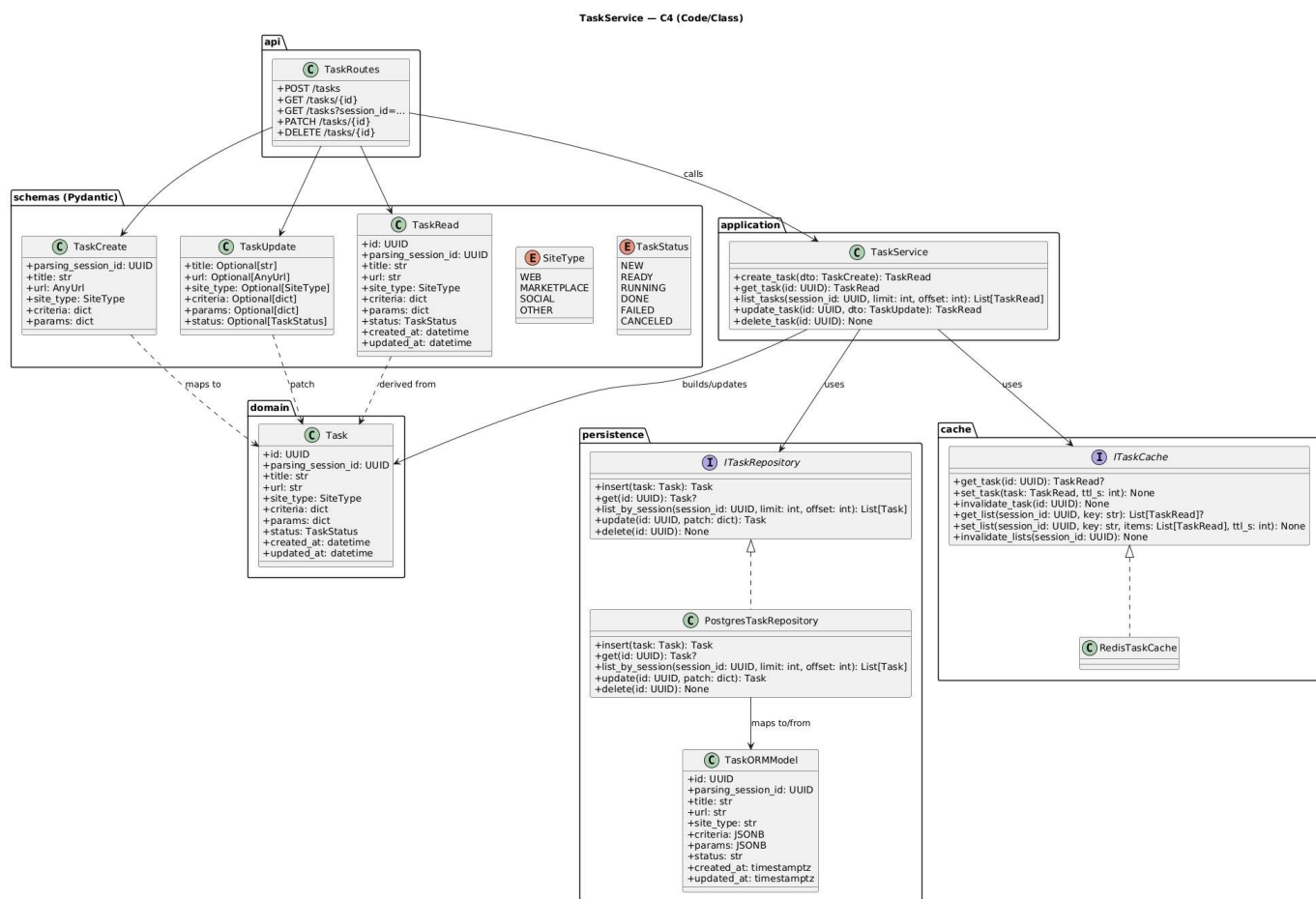


Рисунок 4 – Диаграмма кода

2 Детали реализации:

В ходе выполнения работы был реализован набор REST API endpoints, обеспечивающих полный функционал микросервиса управления задачами парсинга. Endpoint GET /health выполняет проверку работоспособности сервиса и подтверждает его готовность к обработке запросов. Документация API автоматически генерируется с использованием Swagger и доступна по адресу /docs.

Для управления задачами парсинга реализован endpoint POST /api/tasks, который позволяет создавать новую задачу с указанием названия, URL сайта, типа ресурса и критериев сбора данных в формате JSON. Endpoint GET /api/tasks возвращает список всех доступных задач с поддержкой фильтрации по статусу, типу сайта и поиску по названию, а также параметрами пагинации.

Получение детальной информации по конкретной задаче осуществляется через GET /api/tasks/{id}, который возвращает полную конфигурацию задачи, включая параметры и текущий статус выполнения. Для ускорения обработки данного запроса используется кеширование данных в Redis с ограниченным временем жизни. При изменении или удалении задачи кеш автоматически инвалидируется.

Обновление параметров задачи и управление её состоянием реализовано через endpoint PATCH /api/tasks/{id}, позволяющий изменять отдельные поля задачи без необходимости полной перезаписи данных. Удаление задачи выполняется с помощью DELETE /api/tasks/{id}, при этом запись физически удаляется из базы данных, а связанные кешированные данные очищаются.

Веб-интерфейс микросервиса реализован в виде одностраничного приложения (SPA) с использованием React и TypeScript. При загрузке страницы пользователь видит список задач парсинга с основной информацией, включая название, URL, тип сайта и статус. Интерфейс поддерживает создание новых задач через форму ввода, фильтрацию списка задач и изменение статуса выполнения в режиме реального времени без перезагрузки страницы.

Для обеспечения высокой производительности чтения используется Redis,

который хранит данные отдельных задач по идентификатору. Это позволяет сократить количество обращений к базе данных PostgreSQL и обеспечить стабильное время отклика API. Основное хранилище данных реализовано на PostgreSQL, где задачи сохраняются в таблице `tasks` с использованием JSONB поля для хранения критериев сбора данных.

Микросервис разработан таким образом, чтобы быть полностью автономным и не зависеть от работы других компонентов системы Parser. Все необходимые сервисы, включая бэкенд, базу данных и кеш, разворачиваются в отдельных Docker контейнерах и взаимодействуют между собой в изолированной сети Docker Compose. Это обеспечивает предсказуемость поведения системы и упрощает процесс развертывания.

Для тестирования реализован набор ручных и автоматизированных проверок API с использованием Swagger UI. Проверяется корректность создания задач, получение списков и детальных данных, обновление статусов, удаление задач, а также обработка ошибочных ситуаций, таких как запрос несуществующей задачи или передача некорректных данных. Все основные сценарии работы сервиса успешно протестированы.

Развертывание микросервиса осуществляется с помощью Docker Compose одной командой, которая поднимает контейнеры `taskservice-api`, `taskservice-postgres` и `taskservice-redis`. После запуска сервис становится доступен по адресу `localhost:8000`, а пользовательский веб-интерфейс — по адресу `localhost:5173`. Документация API доступна через Swagger UI, что облегчает интеграцию сервиса с другими компонентами системы.

Вывод: в ходе работы был успешно спроектирован и реализован автономный микросервис управления задачами парсинга. Система предоставляет удобный интерфейс для работы с задачами, поддерживает гибкую конфигурацию параметров парсинга и обеспечивает масштабируемость и готовность к интеграции в распределённую систему.