АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ МОСКОВСКИЙ МЕЖДУНАРОДНЫЙ КОЛЛЕДЖ ЦИФРОВЫХ ТЕХНОЛОГИЙ
«АКАДЕМИЯ ТОП»

ИНДИВИДУАЛЬНЫЙ ПРОЕКТ

Уровень профессионального образования:
Среднее профессиональное образование

Программа подготовки специалистов среднего звена
по специальности:
09.02.07 Информационные системы и программирование

Квалификация: Программист

Учебный предмет: Индивидуальный проект

Тема: Разработка инструмента в виде сайта
для структуризации найденной информации
из источников сети «Интернет»

Преподаватель:
К.Р. Сагадиева

_____ _____
подпись инициалы фамилия

Обучающийся:
М.А. Серебряков

_____ _____
подпись дата инициалы фамилия

Магнитогорск, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

По данным, приведенным в проекте стратегии развития отрасли связи Российской Федерации на период до 2035 года Минцифры России, объем российского трафика в сети Интернет к 2022 году составил – 11,1 эксабайт. Пользователям сети «Интернет» приходится работать с огромным количеством неструктурированных данных из различных источников, именно это обусловливала актуальность данного индивидуального проекта.

Для работы, даже с небольшим количеством информации, пользователям сети Интернет приходится ее структурировать. Цель данного индивидуального проекта – структурировать информацию из популярных источников информации в сети Интернет.

Продуктом данного индивидуального проекта является специализированный инструмент в виде сайта, функционал которого – автоматизированное структурирование различной информации из популярных источников информации в сети Интернет.

Для выполнения целей индивидуального проекта были поставлены следующие задачи:

1. Изучение материала по структурированию информации из популярных источников сети Интернет
2. Изучение технологии создания сайтов и методов работы с неструктурированной информации
3. Проектирование продукта с учетом изученного материала для дальнейшей разработки
4. Разработка ранее спроектированного сайта в качестве продукта индивидуального проекта

При создании индивидуального проекта использовались следующие методы: анализ литературы, моделирование, проектирование, программирование и тестирование.

Практическая значимость данного проекта – специализированный инструмент может быть использован в повседневной жизни, как индивидуальными пользователями, так и различными группами и организациями.

I. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

1.1. Структурирование информации

Структурирование – это выявление важных элементов в информационном сообщении установление связей между ними.

Структуризация информации необходима по нескольким причинам:

1. Она необходима для упрощения поиск той же информации в будущем

2. Она необходима для ускорения работы с задачами и идеями

3. Она необходима для облегчения восприятие и запоминание материала

4. Она необходима для выявление закономерностей между разными источниками.

5. Она необходима для принятия обоснованных решений на основе информации.

Существует два основных способов структурирования информация:

1. Логическая организация: разбиение информации на логические блоки

2. Визуализация: использование диаграмм графиков и схем и других визуальных элементов

Основными методами структурирования информации является:

1. Множество: в котором перечислены весь набор элементов и их характерные признаки.

2. Линейный список: множество, состоящее из конечного числа элементов и его элементы расположены в строго определенном порядке.

3. Таблицы: позволяет установить связь между несколькими элементами.

Аспектом структуризации информации является – классификация.

Классификация – группировка данных по определенному признаку. Так как, классификация информации позволяет группировать данные, это помогает систематизировать знания и облегчает их дальнейшее использование.

Классификация может быть основана на различных критериях и это позволяет быстро находить нужную информацию, упрощает ее анализ и обработку.

Классификация помогает избежать дублирования информации и обеспечивает ее целостность. Это особенно важно в условиях современного информационного общества, где объем данных постоянно растет.

Примеры структурирования информации – создания оглавлений в книгах, организация файлов на компьютерах и разработка плана проектов.

В целом, структурирование информации играет ключевую роль в обучении, работе и повседневной жизни, помогая людям эффективнее обрабатывать и использовать полученные знания.

1.2.Технологии и подходы в разработке веб-сайтов

В разработки веб-сайтов существует огромное количество технологий и подходов, помогающие достигнуть до результата.

В разработки веб-сайтов принято разделять проекты на серверную и клиентскую часть: back-end, front-end.

Один из архитектурных подходов при разработки серверной части веб-сайта – микросервисная архитектура.

Микросервисная архитектура – подход архитектуры разработки, при котором серверная часть веб-сайт разбивается на небольшие, автономные сервисы, каждый из которых решает конкретную задачу.

Главными преимуществами перед другими подходами являются:

1. Гибкость и независимость: каждый сервис можно независимо разрабатывать, тестировать и развертывать.
2. Масштабируемость: сервисы можно масштабировать независимо друг от друга.
3. Изоляция ошибок: проблемы в конкретном сервисе не влияют на работу всей системы.
4. Свобода выбора технологий: для каждого сервиса можно выбрать наиболее подходящие технологий и язык программирования.
5. Упрощение процесса разработки: сервисы легче поддерживать и обновлять, поскольку каждый сервис имеет свою собственную кодовую базу и зависимости.
6. Улучшение качества кода: сервисы способствуют созданию чистого и модульного кода.
7. Упрощение тестирования: тестирование сервисов можно проводить независимо друг от друга.
8. Упрощение развертывания: развертывание сервисов может происходить параллельно.

В проектах, использующих микросервисную архитектуру в серверной части, присутствует централизующее звено между сервисами и клиентской частью – API Gateway.

API Gateway – маршрутизатор запросов между службами, который принимает все клиентские запросы, декоммунизирует их и отправляет в разные службы, а затем отправляет ответ обратно клиенту.

API Gateway выполняет множество задач – обрабатывает и распределяет запросы, контролирует трафик, осуществляет мониторинг и контроль доступа.

Так как, при микросервисной архитектуре присутствует свобода выбора технологий, сервисы могут быть написаны на разных языках программирования. Например: TypeScript и Python.

TypeScript – строго-типизированная надстройка на слабо-типизированном JavaScript, языком программирования, являющимся одним из основных языков программирования в разработки веб-сайтов.

TypeScript позволяет выявлять ошибки на этапе разработки, облегчает совместную работу в команде и улучшает производительность разработки в больших проектах.

TypeScript, как и JavaScript, отлично подходит для написания клиентской и серверной части веб-сайтов.

Python – высокоуровневый язык программирования, являющийся одним из популярных. Python, в большинстве своем, используется для написания серверной части веб-сайтов, работы с огромным количеством данных и искусственного интеллекта.

В серверной части веб-сайтов используется базы данных для хранения информации. Существует два основных типа базы данных:

1. Реляционная: организует данные в виде взаимосвязанных таблиц, где каждая таблица состоит из строк и столбцов.

2. Нереляционная: предлагает гибкость в структуре данных и позволяет эффективно работать с неструктурированными или полуструктурированными данными.

Пример реляционной базой данных – MySQL.

Пример нереляционной базой данных – MongoDB.

В проектах, использующих микросервисную архитектуру в серверной части, могут использовать брокеры сообщений для работы между сервисами и API Gateway.

Пример брокера сообщений – RabbitMQ.

Клиентскую часть веб-сайта, в большинстве своем, делают как одностраничное веб-приложение и оно строится на определенной библиотеке, одной из таких является React.

React – JavaScript-библиотека с открытым исходным кодом, предназначенная для разработки пользовательских интерфейсов.

II. ПРАКТИЧЕСКИЙ РАЗДЕЛ

2.1. ПРОЕКТИРОВАНИЕ ПРОДУКТА

При проектирование продукта, были выделены основные компоненты:

1. Авторизация: возможность регистрации пользователя на сайте
2. Проекты: возможность создания отдельных проектов, в которой будет собираться и структурироваться информация.
3. Коллекции: коллекции информации в проектах
4. Собираемое: информация из коллекцией

Серверная часть веб-сайта должна состоять из:

1. Сервис авторизации: отвечающие за регистрацию и сессии
2. Сервис профилей: отвечающий за профили пользователей
3. Сервис проектов: отвечающий за проекты пользователей
4. Сервис редактора: отвечающий за коллекции проектов
5. Сервис-парсер: собирающий информацию

Все сервисы, кроме сервиса редактора и сервиса-парсера, должны предоставлять HTTP API. Сервис редактора обязан предоставлять RPC по WebSocket.

Сервис редактора и сервис-парсер связаны между собой при помощи брокера сообщений.

Все сервисы, кроме сервис-парсер, должны быть реализованы на языке программирования TypeScript. Сервис-парсер необходимо реализовать на Python.

Сервис авторизации должен использовать реляционную базу данных, пока остальные, кроме сервиса-парсера – нереляционную.

Клиентская часть веб-сайта – одностраничное веб-приложение, написанное на TypeScript и использующая React.

2.2. Разработка продукта

Для разработки продукта необходимо сделать Git-репозиторий для всей кодовой базы.

В подпапки сервиса авторизации создаем NPM-пакет и установляем фраймворк Express, MySQL-клиент, TypeScript, библиотеку для хеширования паролей при помощи bcrypt, библиотеку для создания и проверки ключей в формате JWT. Создаем три точки входа:

1. POST /register: регистрация пользователя
2. POST /login: войти в систему
3. POST /refresh: обновление ключей доступа и обновления

Для точки входа регистрации пользователя необходимо написать функцию записи пользователя в базе данных. Функция должна:

1. Создать уникальный идентификатор пользователя: идентификатор пользователя должен является строкой длиной в 24 символа и состоять из случайных маленьких букв латинского алфавита.
2. Хэшировать пароль при помощи bcrypt: это необходимо для безопасного хранения паролей пользователей
3. Отправить команду INSERT в базу данных MySQL с уникальным идентификатором, имя пользователя и хеша пароля пользователей.
4. Вернуть уникальный идентификатор на выход.

Точка входа регистрации пользователя должна применить тело запроса в формате JSON. Тело запроса обязано иметь имя и пароль будущего пользователя.

При успешном выполнении функции записи пользователя в базу данных, точка входа должна вернуть HTTP-ответ 201 Created с телом в формате JSON, где будет находиться уникальный идентификатор пользователя и дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

Для входа в систему необходима написать две функции:

1. Функция проверки учетных данных и получения данных пользователя

2. Функция создания ключей доступа и восстановления

Функция проверки учетных данных и получения данных пользователя должна:

1. Отправить команду SELECT с фильтром по имени пользователя

2. Получить от базы данных хеш пароля пользователя

3. Проверить пароля пользователя при помощи хеша

4. Вернуть данные пользователя на выход

Функция создания ключей доступа и восстановления должна:

1. Создать ключ доступа с уникальным идентификатором, имени пользователя и булевым значением, означающее, что ключ является ключом доступом: используем алгоритм шифрования RS256 и приватный RSA-ключ.

2. Создать ключ восстановления с уникальным идентификатором, имени пользователя и булевым значением, означающее, что ключ является ключом восстановлением: так же используем алгоритм шифрования RS256 и приватный RSA-ключ.

3. Вернуть оба ключа на выход.

Точка выхода действует по следующему методу:

1. Получает имя и пароля пользователя из тела запроса в формате JSON.

2. Получает данные пользователя при помощи функции проверки учетных данных и получения данных пользователя.

3. Создать ключи при помощи функции создания ключей доступа и восстановления

4. Вернуть HTTP-ответ 200 OK с телом в формате JSON, где будет находиться ключ доступа, ключ восстановления, дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

Для точки входа обновления ключей доступа и обновления необходимо написать функцию проверки ключа обновления и создания обновленных ключей. Функция должна:

1. Проверить ключ обновления в формате JWT: для этого необходимо использовать публичный RSA-ключ
2. Получить данные из ключ обновления
3. Создать обновленные ключи при помощи функции создания ключей доступа и восстановления
4. Вернуть обновленные ключи на выходе

При успешном выполнении функции записи пользователя в базу данных, точка входа должна вернуть HTTP-ответ 200 OK с телом в формате JSON, где будет находиться ключ доступа, ключ восстановления, дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

В подпапки сервиса профилей создаем NPM-пакет и установляем фраймворк Express, Mongoose, TypeScript. Создаем две точки входа:

1. GET /@me: получение профиля пользователя
2. POST /@me: изменение отображаемого имени в профиле

Учитываем, что в каждом запросе будет задано уникальный идентификатор и имя пользователя в заголовках.

Модель документа профиля пользователя в базе данных MongoDB должна иметь:

1. Уникальный идентификатор профиля пользователя в формате ObjectId: записывающийся в поле _id
2. Уникальный идентификатор пользователя
3. Отображаемое имя пользователя

Для точки входа получения профиля пользователя необходимо написать функцию получения или создания нового профиля пользователя. Функция должна:

1. Получить документ профиля пользователя из базы данных, если она существуют
2. Если документ был получен успешно, то функция возвращает данные из него на выход
3. Если документ не был получен, то функция создает новый документ с уникальным идентификатором пользователя и имени пользователя в качестве отображаемого имени и возвращает данные из него на выход

При успешном выполнении функции получения или создания нового профиля пользователя, точка входа должна вернуть HTTP-ответ 200 OK с телом в формате JSON, где будет находиться:

1. Уникальный идентификатор профиля пользователя
2. Уникальный идентификатор пользователя
3. Отображаемое имя пользователя
4. Дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

Для точки входа изменения отображаемого имени в профиле необходимо написать функцию изменения отображаемого имени в профиля пользователя. Функция должна получить документ профиля пользователя из базы данных и обновить поле отображаемого имени.

При успешном выполнении функции изменения отображаемого имени в профиля пользователя, точка входа должна вернуть HTTP-ответ 200 OK с телом в формате JSON, где будет находиться:

1. Уникальный идентификатор профиля пользователя
2. Уникальный идентификатор пользователя
3. Отображаемое имя пользователя
4. Дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

В подпапки сервиса проектов создаем NPM-пакет и установляем фраймворк Express, Mongoose, TypeScript. Создаем пять точки входа:

1. POST /: создание проекта пользователя
2. GET /: получение всех проектов пользователя
3. GET /:id: получение конкретного проекта пользователя
4. POST /:id: изменение названия и описания проекта пользователя
5. DELETE /:id: удаление конкретного проекта пользователя

Учитываем, что в каждом запросе будет задано уникальный идентификатор профиля пользователя в заголовках.

Модель документа проекта пользователя в базе данных MongoDB должна иметь:

1. Уникальный идентификатор проекта пользователя в формате ObjectId: записывающийся в поле _id
2. Название проекта в виде строки

3. Описание проекта в виде строки

Для точки входа создания проекта пользователя необходимо написать функцию создания документа проекта пользователя. Функция должна создавать документ проекта пользователя в базе данных и возвращать уникальный идентификатор проекта пользователя.

При успешном выполнении функции создания документа проекта пользователя, точка входа должна вернуть HTTP-ответ 201 Created с телом в формате JSON, где будет находиться уникальный идентификатор проекта пользователя и дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

Для точки входа получения всех проектов пользователя необходимо написать функцию получения всех проектов пользователя. Функция должна получать документы проектов пользователя с фильтром по уникальному идентификатору профиля пользователя в базе данных и возвращать массив с данными проектов пользователя.

При успешном выполнении функции создания документа проекта пользователя, точка входа должна вернуть HTTP-ответ 200 OK с телом в формате JSON, где будет находиться массив с данными проектов пользователя и дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

Для точки входа получения конкретного проекта пользователя необходимо написать функцию получения конкретного проекта пользователя. Функция должна получить документ проекта пользователя с

фильтром по уникальному идентификатору профиля и проекта пользователя в базе данных и возвращать данные проекта пользователя из него.

При успешном выполнении функции получения конкретного проекта пользователя, точка входа должна вернуть HTTP-ответ 200 OK с телом в формате JSON, где будет находиться данные проекта пользователя и дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

Для точки входа изменения конкретного проекта пользователя необходимо написать функцию изменения конкретного проекта пользователя. Функция должна получить документ проекта пользователя с фильтром по уникальному идентификатору профиля и проекта пользователя в базе данных и обновить название и описание проекта на новый вариант.

При успешном выполнении функции изменения конкретного проекта пользователя, точка входа должна вернуть HTTP-ответ 200 OK с телом в формате JSON, где будет находиться обновленные данные проекта пользователя и дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

Для точки входа изменения конкретного проекта пользователя необходимо написать функцию изменения конкретного проекта пользователя. Функция должна получить документ проекта пользователя с фильтром по уникальному идентификатору профиля и проекта пользователя в базе данных и обновить название и описание проекта на новый вариант.

При успешном выполнении функции изменения конкретного проекта пользователя, точка входа должна вернуть HTTP-ответ 200 OK с телом в

формате JSON, где будет находиться обновленные данные проекта пользователя и дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

Для точки входа удаления конкретного проекта пользователя необходимо написать функцию удаления конкретного проекта пользователя. Функция должна получить документ проекта пользователя с фильтром по уникальному идентификатору профиля и проекта пользователя в базе данных и обновить название и удалить его.

При успешном выполнении функции удаления конкретного проекта пользователя, точка входа должна вернуть HTTP-ответ 200 OK с телом в формате JSON, где будет находиться дополнительный код успешного выполнения в виде строки SUCCESS.

При ошибке, точка входа должна вернуть HTTP-ответ 500 Internal Server Error с телом в формате JSON, где будет находится код ошибки в виде строки.

В подпапки сервиса-парсера необходимо создать виртуальное окружение Python и установить зависимости:

1. Клиент RabbitMQ
2. HTTP-клиент под названием requests
3. Библиотеку для работы с HTML-разметкой
4. Библиотеку для работы с WikiText-разметкой
5. Библиотеку для работы с URL-ссылками

В главном скрипте, сервис-парсер должен подключиться к RabbitMQ и создать канал с очередью под получения запросов от других сервисов.

При получении сообщении сервис-парсер обязан проверить его и получить URL-ссылки из него.

В сервис-парсере должно быть реализовано два робота, собирающих информацию сайтов:

1. Робот, собирающий информацию статьей с сайтов на основе WordPress: пример – веб-сайт игровой студии Mundfish

2. Робот, собирающий информацию статьей с сайтов на основе MediaWiki: пример – Wikipedia

Оба робота имеют схожую схему:

3. Проверка, если робот сможет сработаться с сайтом

4. Получение данных статьи

5. Сбор информации статьи и форматирование ее в более простой формат

При успешным получении информации статьи, сервис-парсер возвращает ее обратно по другому каналу. При ошибке получении информации – возвращает код и сообщение ошибки.

В подпапки сервиса редактора необходимо сделать NPM-пакет и установить зависимости:

6. Библиотеку, реализующую протокол WebSocket

7. Mongoose для работы с базой данных

8. RabbitMQ-клиент для работы с сервисом-парсером

Сервис редактора должен применить сообщения от клиента в формате JSON и следующими данными:

1. Название вызываемого метода
2. Уникальный идентификатор вызова
3. Аргументы метода

Сервис редактора должен иметь реализацию следующих методов:

1. createCollection: создание коллекции в проекте пользователя
2. getCollections: получение коллекции в проекте пользователя
3. updateCollection: изменение данных коллекции в проекте пользователя

4. deleteCollection: удаление коллекции в проекте пользователя

5. createCollectible: создание собираемого в коллекцию проекта пользователя

6. getCollectibles: получение всего собираемого из коллекции проекта пользователя

7. setCollectiblePriority: изменение приоритета собираемого в коллекции проекта пользователя

8. deleteCollectible: удаление собираемого из коллекции проекта пользователя

Методы createCollection, getCollections, updateCollection, deleteCollection реализуется похожим образом как проекты в сервисе проектов.

Модель документа коллекции проекта пользователя должна иметь:

1. Уникальный идентификатор проекта пользователя в формате ObjectId: записывающийся в поле _id

2. Название коллекции в виде строки

3. Цвета коллекции в формате HEX и в виде строки

4. Массив собираемого: является субдокументы

Методы связанные с коллекциями не должны возвращать массив собираемого.

Методы getCollectibles, setCollectiblePriority, deleteCollectible реализуется похожим образом как методы коллекции проекта пользователя.

Метод createCollectible должен использовать сервис-парсер для получения данных с URL-ссылок.

В подпапки клиентского части веб-сайта необходимо создать приложение React на TypeScript при помощи Vite и установить следующие зависимости:

1. react-hook-form для удобной работы с формами

2. react-router-dom для разделение приложения на отдельные страницы

Необходимо создать семь страниц в приложении:

1. /: индексная страница
2. /auth/login: страница входа
3. /auth/register: страница регистрации
4. /dashboard/projects: страница проектов
5. /dashboard/profile: страница профиля пользователя
6. /editor/projects/:id: страница редактора проекта пользователя

Единственная функция индексной страницы – переводить пользователя на страницу проектов, если он авторизован, а если нет – на страницу входа.

Страница входа и регистрации должна использовать сервис авторизации, сохранять полученные ключи в локальное хранилище браузера клиента.

Страница проектов должна использовать сервис проектов при работы с проектами.

Страница профиля пользователя должна использовать сервис профилей и иметь возможность выйти из сессии.

Страница редактора должна использовать сервис редактора и возможность просматривать содержимое собираемого.

Для работы с API сервисов должны быть созданы отдельные функции. Для работы с RPC сервиса редактора должен быть создан отдельный класс.

ЗАКЛЮЧЕНИЕ

В ходе выполнения индивидуального проекта была достигнута поставленная цель – разработан инструмент в виде сайта для структуризации найденной информации из интернет-источников.

Созданный веб-сайт позволяет пользователям легко находить и систематизировать необходимую информацию, что значительно упрощает процесс обучения и исследований.

Проект имеет большой потенциал для дальнейшего развития. Планируется добавить новые функции, улучшить интерфейс и расширить возможности интеграции с другими сервисами.

В целом реализация этого проекта позволила мне получить ценный опыт в области разработки веб-приложений и анализа данных. Проект демонстрирует, что создание инструментов структурирования информации из интернет-источников является актуальной и востребованной задачей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Проект стратегии развития отрасли связи Российской Федерации на период до 2035 года // Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации URL: https://digital.gov.ru/ru/documents/9120/ (дата обращения: 15.08.2023)

2. Как структурировать большой объем информации // Unicraft — платформа для онлайн обучения URL: https://www.unicraft.org/blog/7986/structurirovanie-informatcii/ (дата обращения: 13.08.2022).

3. Структурирование информации // справочник от автор24 URL: https://spravochnick.ru/informatika/informacionnye_processy_i_informaciya/strukturirovanie_informacii/

4. Структурирование информации: методы, приемы, полезные советы и приложения // Alpina Digital URL: https://alpinadigital.ru/blog/strukturirovanie-informacii-metody-i-priemy/

5. Про API Gateway простыми словами // GETIT URL: https://getit.academy/apigateway

6. Что такое TypeScript и зачем он нужен // Хекслет URL: https://ru.hexlet.io/blog/posts/vse-chto-nuzhno-znat-novichku-o-typescript-ischerpyvayuschiy-gayd

7. Эсперанто для машин. Что нужно знать о языке программирования Python // Skillbox.by URL: https://blog.skillbox.by/kod/jesperanto-dlja-mashin-chto-nuzhno-znat-o-jazyke-programmirovanija-python/ (дата обращения: 11.07.2023)

8. React.js для новичков в программировании: что это, как устроен и зачем нужен // Skillbox Media URL: https://skillbox.ru/media/code/reactjs-dlya-novichkov-v-programmirovanii-chto-eto-kak-ustroen-i-zachem-nuzhen/ (дата обращения: 03.04.2023)

ПРИЛОЖЕНИЯ

Приложения 1

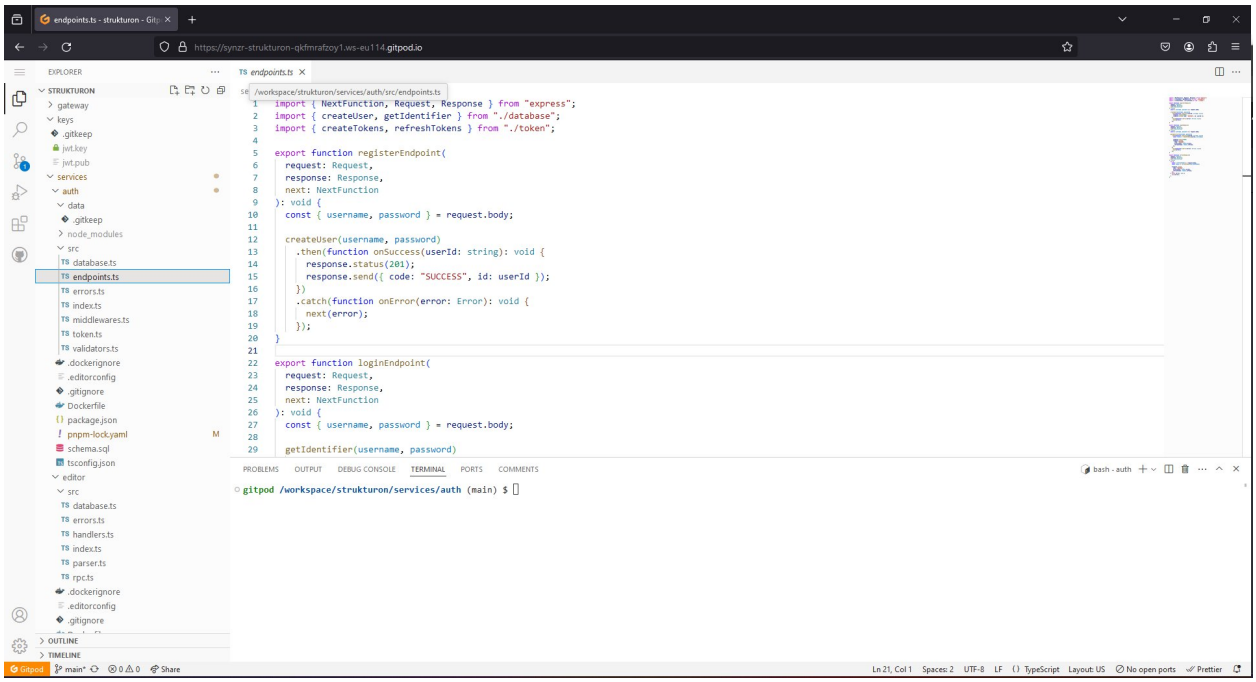


Рис. 1 – скриншот исходного кода сервиса авторизации

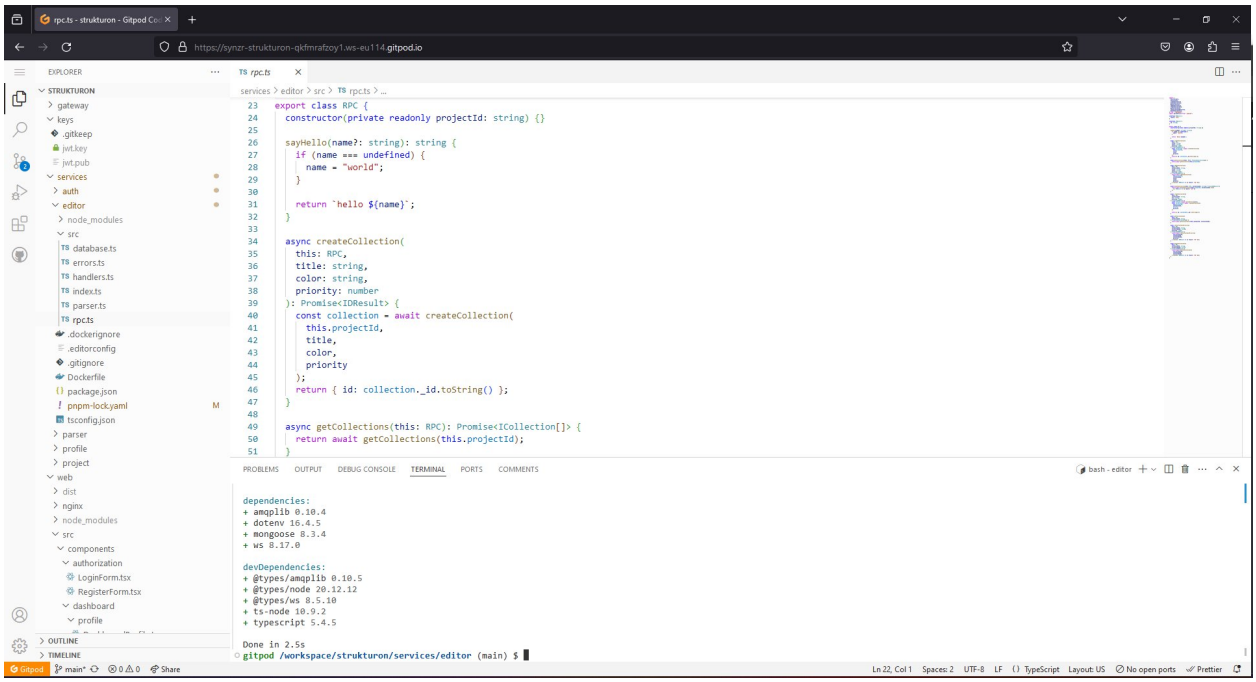
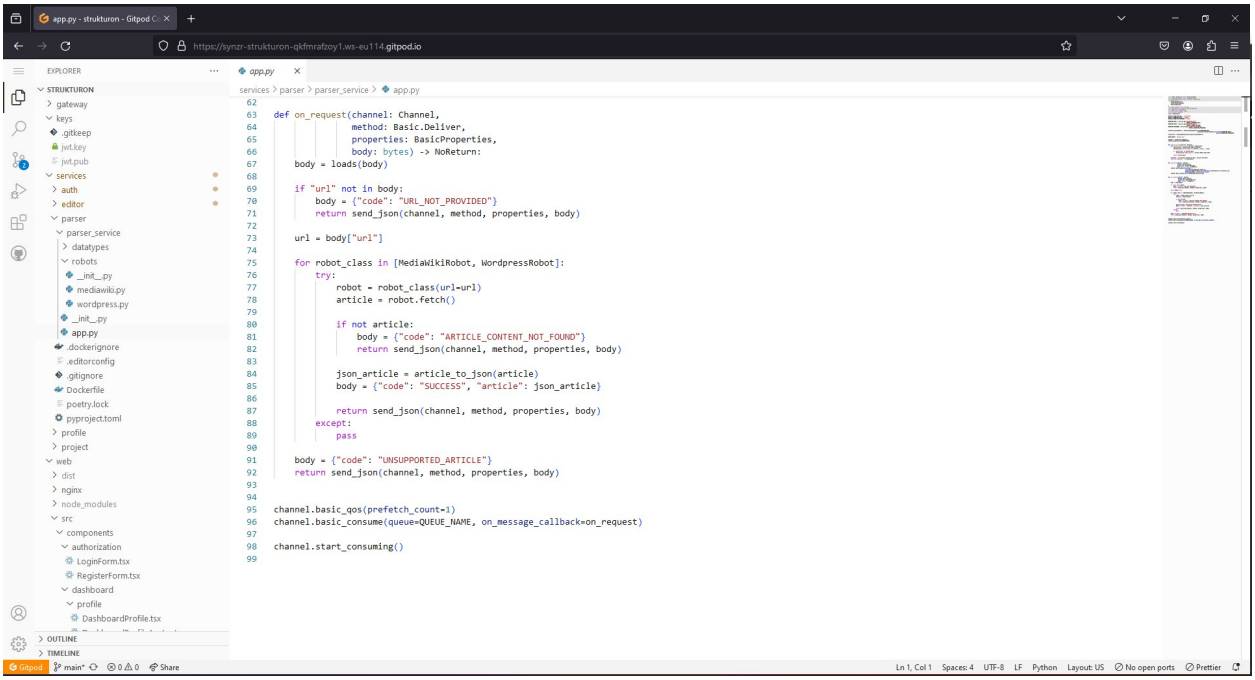
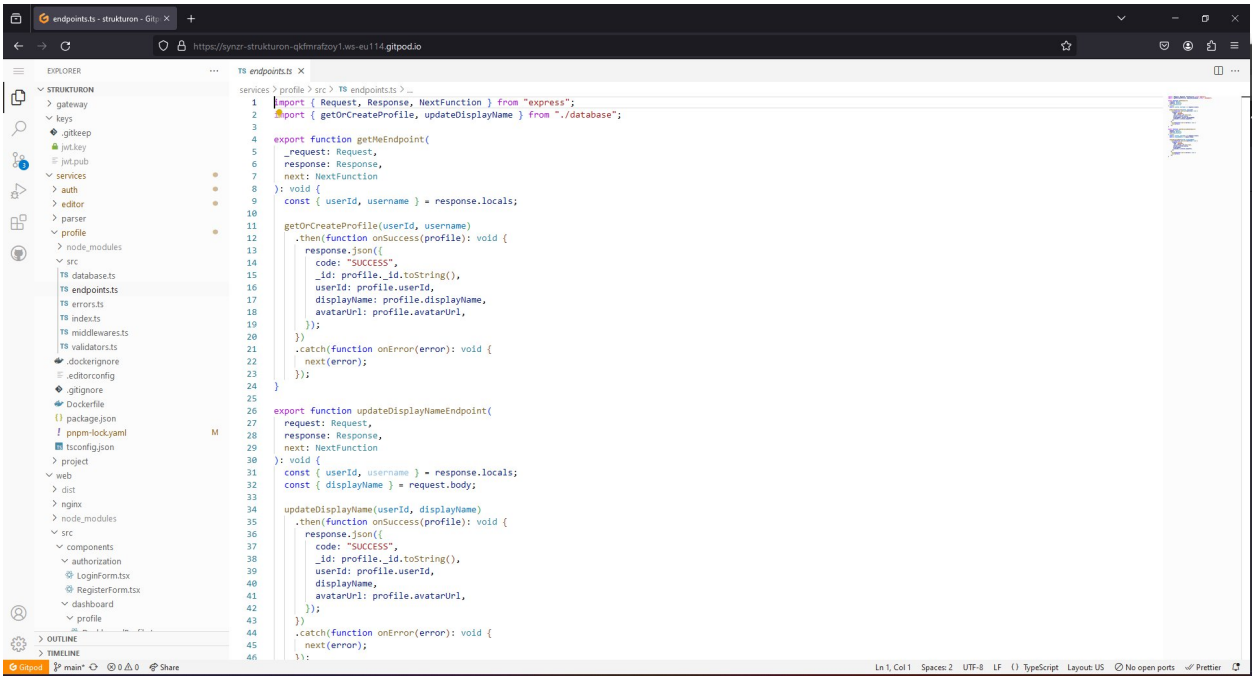


Рис. 2 – скриншот исходного кода сервиса редактора



```python
def on_request(channel: Channel,
               method: Basic.Deliver,
               properties: BasicProperties,
               body: bytes) -> NoReturn:
    body = loads(body)

    if "url" not in body:
        body = {"code": "URL_NOT_PROVIDED"}
        return send_json(channel, method, properties, body)

    url = body["url"]

    for robot_class in [MediaWikiRobot, WordpressRobot]:
        try:
            robot = robot_class(url=url)
            article = robot.fetch()

            if not article:
                body = {"code": "ARTICLE_CONTENT_NOT_FOUND"}
                return send_json(channel, method, properties, body)

            json_article = article_to_json(article)
            body = {"code": "SUCCESS", "article": json_article}

            return send_json(channel, method, properties, body)
        except:
            pass

    body = {"code": "UNSUPPORTED_ARTICLE"}
    return send_json(channel, method, properties, body)


channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue=QUEUE_NAME, on_message_callback=on_request)

channel.start_consuming()
```

Рис. 3 – скриншот исходного кода сервиса-парсера



```typescript
import { Request, Response, NextFunction } from "express";
import { getOrCreateProfile, updateDisplayName } from "./database";

export function getMeEndpoint(
  _request: Request,
  response: Response,
  next: NextFunction
): void {
  const { userId, username } = response.locals;

  getOrCreateProfile(userId, username)
    .then(function onSuccess(profile): void {
      response.json({
        code: "SUCCESS",
        _id: profile._id.toString(),
        userId: profile.userId,
        displayName: profile.displayName,
        avatarUrl: profile.avatarUrl,
      });
    })
    .catch(function onError(error): void {
      next(error);
    });
}

export function updateDisplayNameEndpoint(
  request: Request,
  response: Response,
  next: NextFunction
): void {
  const { userId, username } = response.locals;
  const { displayName } = request.body;

  updateDisplayName(userId, displayName)
    .then(function onSuccess(profile): void {
      response.json({
        code: "SUCCESS",
        _id: profile._id.toString(),
        userId: profile.userId,
        displayName,
        avatarUrl: profile.avatarUrl,
      });
    })
    .catch(function onError(error): void {
      next(error);
    });
```

Рис. 4 – скриншот исходного кода сервиса профилей

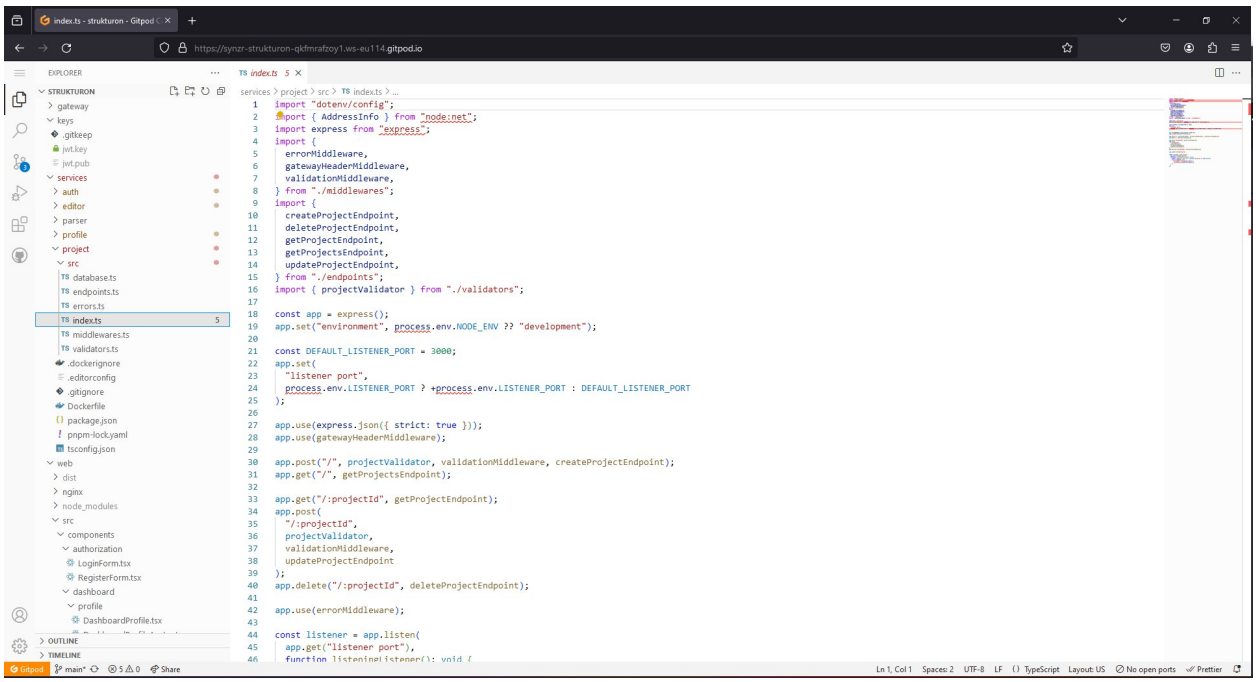


Рис. 5 – скриншот исходного кода сервиса проектов

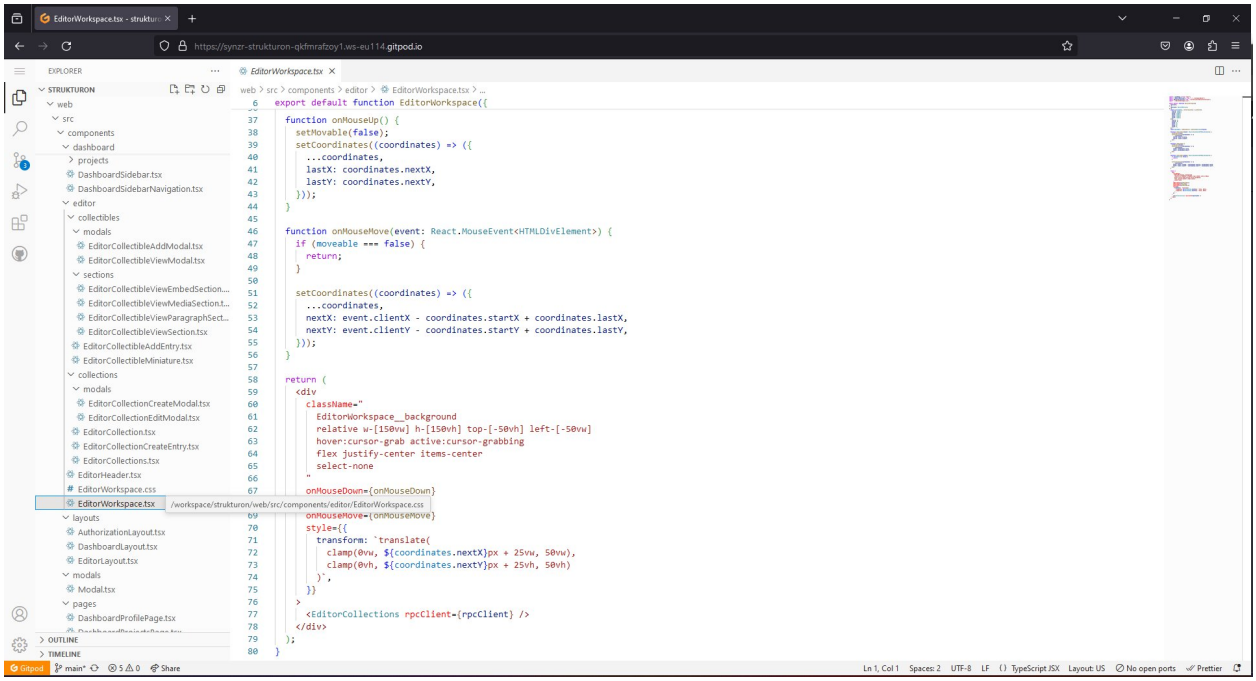


Рис. 6 – скриншот исходного кода клиентской части веб-сайта

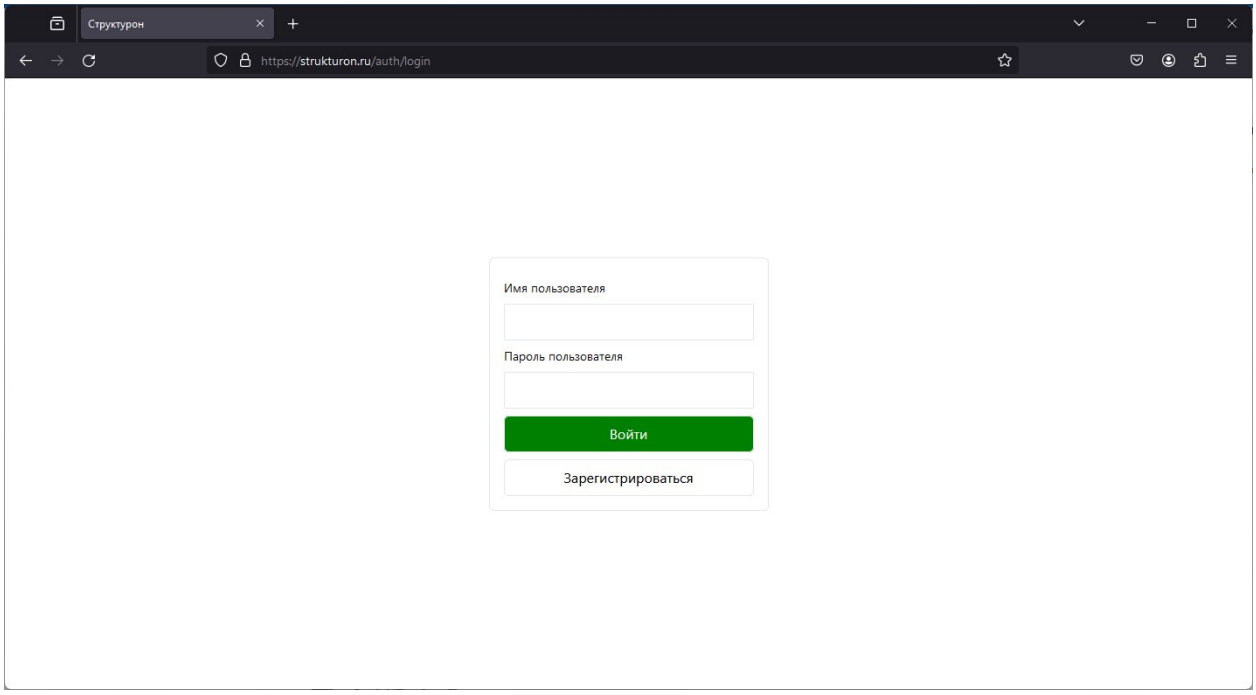


Рис. 7 – страница входа

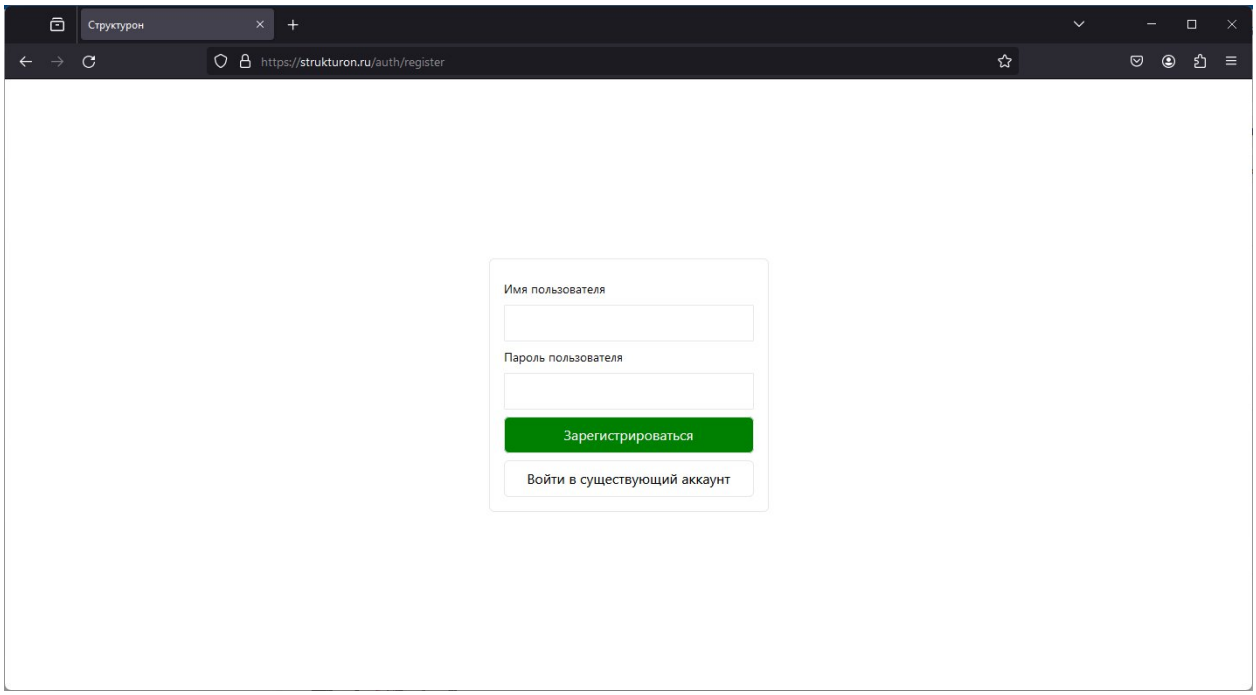


Рис. 8 – страница регистрации

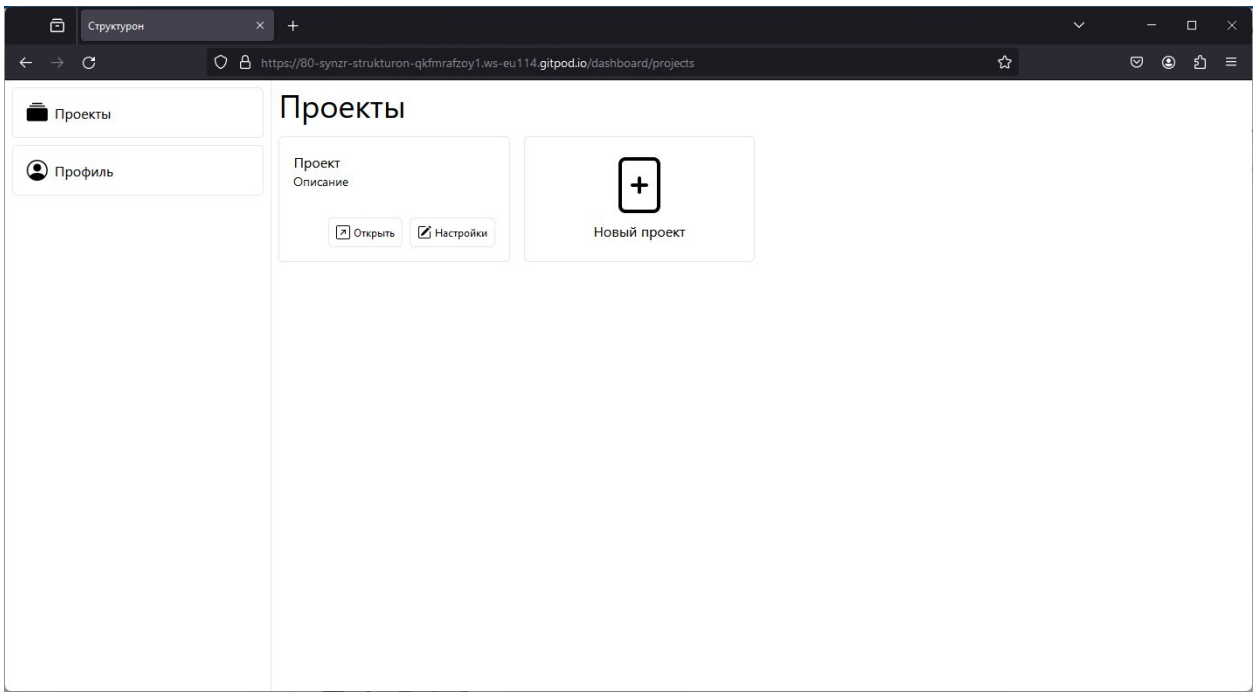


Рис. 9 – страница проектов

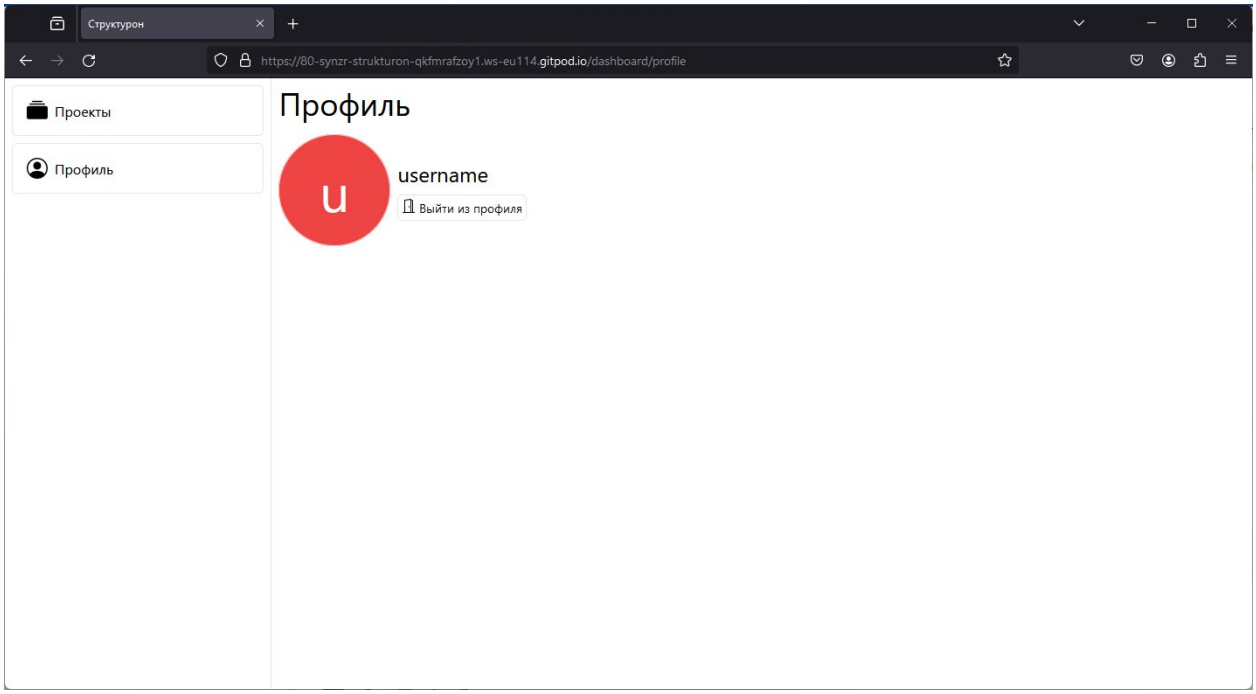


Рис. 10 – страница профиля

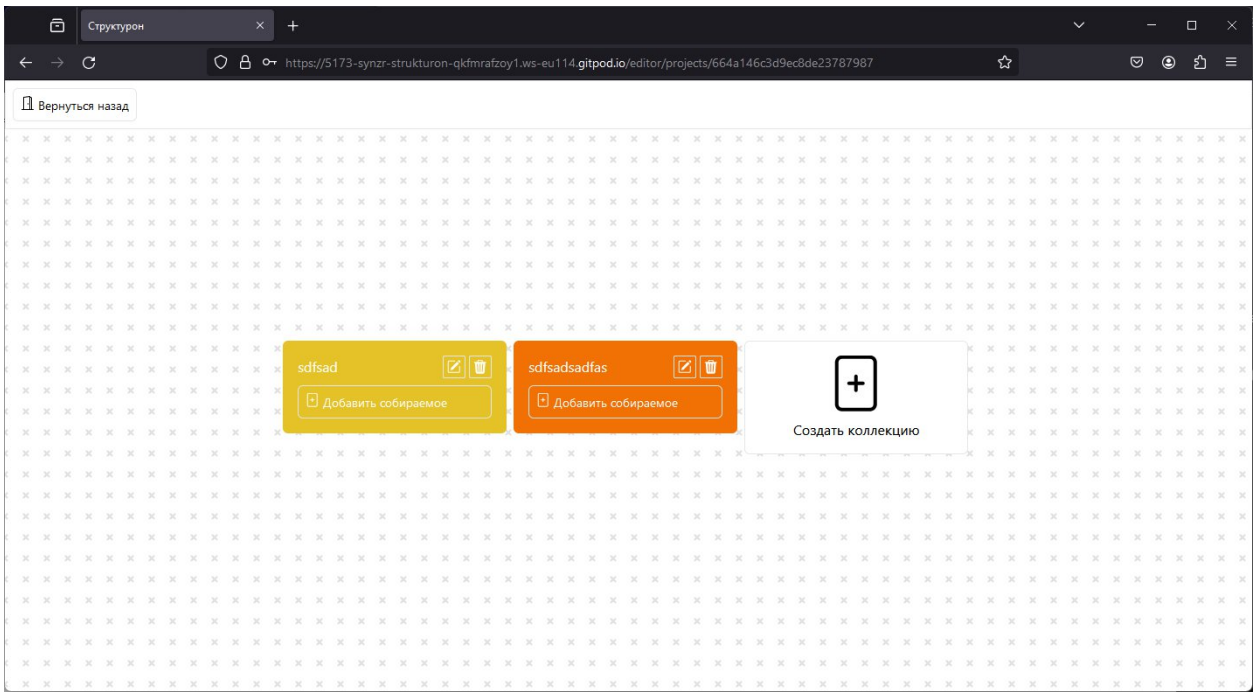


Рис. 11 – страница редактора проектов