

Санкт-Петербургский государственный университет

Каргин Глеб Павлович

Выпускная квалификационная работа

Разработка сервиса для работы с учебными практиками

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование
информационных систем»*

Основная образовательная программа *СВ.5162.2021 «Технологии программирования»*

Научный руководитель:
старший преподаватель кафедры системного программирования, к.т.н. Ю. В. Литвинов

Рецензент:
доцент кафедры информатики, СПбГУ, к.ф.-м.н, Д. А. Григорьев

Санкт-Петербург
2025

Saint Petersburg State University

Gleb Kargin

Bachelor's Thesis

Development of a service for working with educational practices

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2021 "Programming Technologies"*

Scientific supervisor:
C.Sc., senior lecturer, Y. V. Litvinov

Reviewer:
docent of computer science department, SPBU, C.Sc, D. A. Grigorev

Saint Petersburg
2025

Оглавление

Введение	4
1. Постановка задачи	6
2. Обзор	7
2.1. Обзор существующих решений	7
2.2. Обзор используемых технологий	10
3. Проектирование	14
4. Реализация	16
4.1. Серверная часть	16
4.2. Пользовательский интерфейс	18
5. Эксперимент	20
6. Заключение	22
Список литературы	23

Введение

В СПбГУ студенты математико-механического факультета, начиная со второго курса, проходят учебные практики и выполняют выпускные квалификационные работы (ВКР), направленные на углубление теоретических знаний и развитие профессиональных навыков. В этот процесс вовлечены несколько участников, каждый из которых играет важную роль. Студент отвечает за выполнение практики или ВКР, выбирает тему из предложенных направлений и согласовывает её с научным руководителем и, при необходимости, с консультантом. Научный руководитель — преподаватель, курирующий выполнение работы, а также помогающий в постановке задач. Консультант — специалист, обладающий узкой экспертизой в выбранной теме, который может быть как преподавателем СПбГУ, так и представителем внешней организации, и оказывает методическую помощь по содержанию. После завершения работы отчёт студента направляется на рецензию, где рецензент оценивает соответствие требованиям, полноту и качество выполненного задания. За общую организацию и координацию процесса отвечает руководитель практики — сотрудник, отслеживающий соблюдение сроков и обеспечивающий взаимодействие между всеми участниками. Такой подход обеспечивает системную и эффективную реализацию практико-ориентированного обучения.

Однако процесс выбора тем, подачи отчётов и отслеживания выполнения практик остаётся трудоёмким и требует значительных усилий со стороны всех участников. В настоящее время на сайте кафедры системного программирования СПбГУ существует сервис, который призван автоматизировать часть этих процессов. Тем не менее, его функциональность имеет ряд существенных недостатков. Система сложна в поддержке, её интерфейс не всегда интуитивно понятен, а архитектура перегружена и малоадаптирована для эффективной работы. Кроме того, сервис интегрирован с основным сайтом кафедры, предназначенным в первую очередь для публикации статического информационного контента, а не для организации и управления учебными процессами.

Попытка реализовать такую многоуровневую систему на статическом сайте привела к трудностям как для разработчиков, так и для пользователей.

В связи с этим было принято решение создать отдельный сервис, независимый от основного сайта кафедры, который будет направлен исключительно на автоматизацию всех этапов учебной практики — от сбора и выбора тем до оценки результатов и размещения материалов успешных практик. Новый сервис предназначен для взаимодействия студентов, преподавателей и внешних экспертов на всех этапах процесса. Он включает сбор и проверку отчётов, а также контроль за выполнением задач.

Для реализации этой идеи разрабатывается специализированное веб-приложение с микросервисной архитектурой, что обеспечивает поддержку и возможность расширения функциональности системы для работы с учебными практиками СПбГУ.

1. Постановка задачи

Целью работы является создание веб-приложения для управления процессом прохождения учебных практик. Для её выполнения были поставлены следующие задачи.

1. Провести обзор аналогичных систем.
2. Спроектировать общую архитектуру системы и пользовательский интерфейс.
3. Реализовать необходимые микросервисы.
4. Разработать модульную авторизацию, настроить ролевые модели, политики.
5. Реализовать пользовательский интерфейс.
6. Провести тестирование реализованного сервиса.

2. Обзор

2.1. Обзор существующих решений

Перед началом разработки нового сервиса управления учебными практиками были изучены существующие решения, которые частично или полностью обеспечивают аналогичную функциональность. Несмотря на то, что уже существует функционирующий сервис, были также рассмотрены некоторые инструменты, которые могут быть адаптированы для подобных задач.

- Веб-сервис для написания учебных практик и выпускных квалификационных работ. Сервис учебных практик¹ (представлено на Рис. 1) уже внедрён и функционирует на сайте кафедры СП СПбГУ. В текущей версии реализованы ключевые функции, включая добавление тем, их распределение по студентам, сопровождение и рецензирование работ [13]. Однако этот сервис имеет ряд серьёзных недостатков: сложность сопровождения, перегруженность функциональностью, а также неудобство интеграции с основным сайтом кафедры. Использование неподходящей архитектуры затрудняет масштабируемость системы и внесение изменений. Кроме того, не реализована часть критически важной функциональности, например, архивирование тем в конце семестра.
- Microsoft Teams. Microsoft Teams² широко используется в образовательных учреждениях, включая СПбГУ, для организации совместной работы. Данный сервис позволяет создавать команды для студентов и преподавателей, использовать чаты для индивидуального взаимодействия, публиковать организационную информацию, хранить файлы и создавать задания с дедлайнами. Тем не менее, в Microsoft Teams отсутствуют функции для автоматизированного учёта студентов, а также возможности для более удобного

¹Веб-сервис для написания учебных практик и выпускных квалификационных работ. URL: <https://se.math.spbu.ru/practice> (дата обращения: 2025-05-24)

²Microsoft Teams. URL: <https://www.microsoft.com/ru-ru/microsoft-teams> (дата обращения: 2025-05-24)

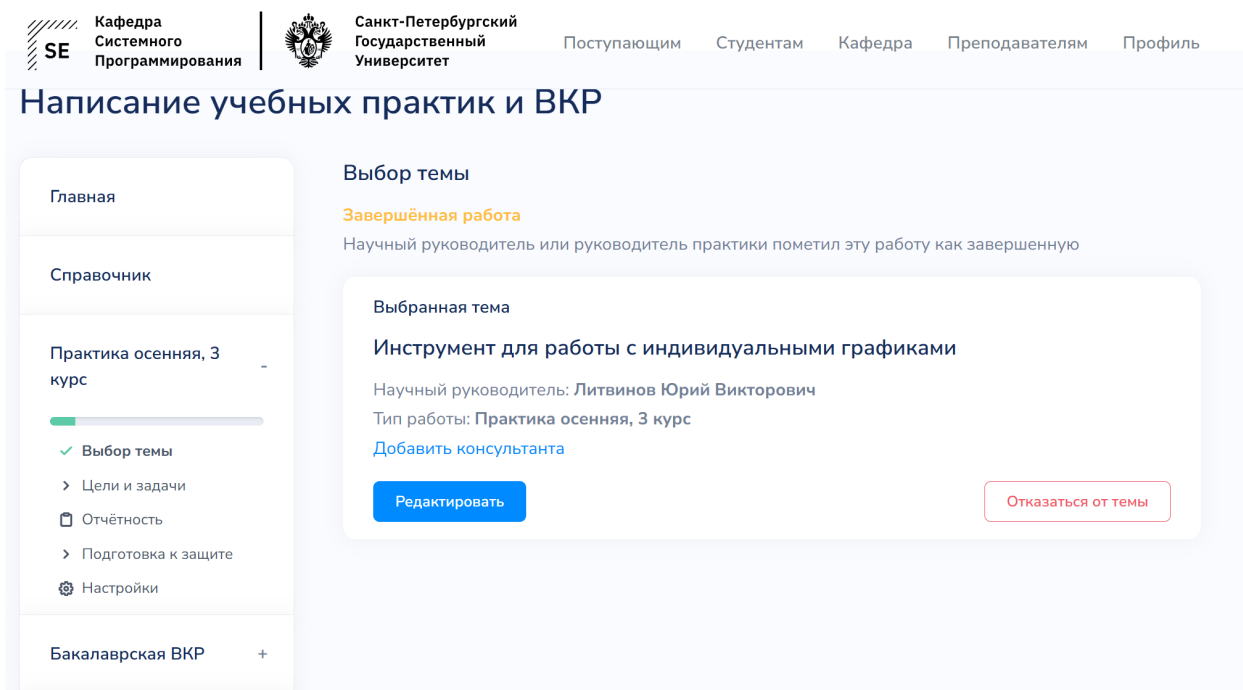


Рис. 1: Текущий веб-сервис для написания учебных практик и выпускных квалификационных работ

го структурирования данных по учебным практикам. Кроме того, большое количество команд при большом числе студентов может значительно усложнить управление процессом.

- Системы управления проектами (например, Jira³). Системы управления проектами предоставляют инструменты для организации работы над задачами и обеспечивают разнообразную настройку рабочих процессов, возможность создавать задачи, назначать исполнителей, отслеживать сроки и статусы, работать с комментариями и вложениями. Однако они не обладают специализированной функциональностью для образовательных процессов, такой как учёт студентов, автоматизация распределения тем и взаимодействие с научными руководителями. Кроме того, их адаптация для образовательных целей может быть более сложной и трудоёмкой, чем написание системы с нуля.

³Jira. URL: <https://www.atlassian.com/ru/software/jira> (дата обращения: 2025-05-24)

- Системы контроля версий (например, GitHub⁴). Данные системы активно применяются для хранения проектов и совместной работы. Они поддерживают коллективное редактирование файлов, управление версиями и предоставление обратной связи через механизмы проверки изменений. Это делает их удобными для хранения программных и текстовых материалов. Однако такие платформы изначально не рассчитаны на процессы, касаемые написания учебных практик. Это значит, что преподавателям придётся самостоятельно разрабатывать способы их адаптации для работы со студентами. Также отсутствуют возможности для автоматического учёта студентов и распределения тем. Тем не менее, в некоторых случаях такие системы используются в процессе работы с учебными практиками. Например, в магистратуре по программе «Системное программирование» студенты обязаны сделать pull request с материалами практики в специальный кафедральный репозиторий, чтобы получить допуск к защите.
- Google Workspace и Яндекс.Календарь. Сервисы Google Workspace (Docs, Drive, Calendar)⁵ и Яндекс.Календарь⁶ могут быть использованы для организации взаимодействия студентов и научных руководителей. Они позволяют совместно работать над документами, вести календари, отправлять уведомления и хранить материалы, что делает их удобными для базовой организации процессов. Тем не менее, в данных системах отсутствуют функции структурированной организации учебных практик и автоматизации задач, такие как контроль выполнения, рецензирование и распределение тем.

⁴GitHub. URL: <https://github.com> (дата обращения: 2025-05-24)

⁵Google Workspace. URL: <https://workspace.google.com> (дата обращения: 2025-05-24)

⁶Яндекс.Календарь. URL: <https://calendar.yandex.ru> (дата обращения: 2025-05-24)

2.2. Обзор используемых технологий

Согласно заранее сформулированным требованиям к разрабатываемой системе, архитектура должна быть основана на микросервисном подходе. Эти требования были определены ещё до начала работы и касались не только общей архитектуры, но и других аспектов реализации. Выбор в пользу микросервисной архитектуры был сделан исходя из предположения, что проект в дальнейшем смогут развивать студенты младших курсов. Чёткая декомпозиция системы на независимые компоненты позволит минимизировать взаимозависимость между частями проекта и существующей реализацией, тем самым облегчая процесс включения новых участников в разработку. Кроме того, микросервисный подход повышает модульность, упрощает внедрение новых функций и масштабирование системы.

2.2.1. .NET

Платформа .NET [5] была выбрана благодаря поддержке языков программирования, которые активно изучаются на факультете, а также применяются при разработке различных университетских сервисов. Все это гарантирует наличие студентов и преподавателей, способных поддерживать систему, и позволяет с лёгкостью интегрировать новую функциональность с существующими решениями. Был выбран язык программирования C#, поскольку большинство университетских сервисов используют именно его, что обеспечит дальнейшую поддержку и соответствие стандарту, а также потому, что автор имеет опыт его использования. Кроме того, в требованиях для разработки основного сервиса для работы с данными была также определена платформа .NET.

2.2.2. ASP.NET Core

ASP.NET Core [1] — фреймворк для создания веб-приложений на основе платформы .NET. В классическом подходе разработки с использованием ASP.NET применяются контроллеры, которые обрабатывают запросы к определённым URL-адресам (эндпойнтам). Каждый метод

контроллера отвечает за выполнение конкретной задачи и часто возвращает сгенерированную веб-страницу, например, с помощью движка Razor. В данном случае предпочтение было отдано подходу Minimal API, который упрощает разработку небольших веб-сервисов и лучше подходит в случае микросервисной архитектуры, благодаря отсутствию лишних абстракций. В этом подходе ответ формируется как результат выполнения функции, данные автоматически преобразуются в JSON, а для реализации необходимо лишь задать метод HTTP-запроса, путь и соответствующую функцию обработки.

2.2.3. Микросервисная архитектура и YARP

Для обеспечения модульности и масштабируемости системы был реализован подход с микросервисной архитектурой. Это позволяет разделить систему на независимые сервисы, каждый из которых выполняет отдельные задачи, такие как управление пользователями, темами или рецензиями. Для маршрутизации запросов между сервисами использовался YARP (Yet Another Reverse Proxy) [12] — обратный прокси-сервер, оптимизированный для работы в среде .NET. Его использование обусловлено интеграцией с экосистемой .NET, возможностью конфигурации через код и поддержкой динамической маршрутизации. Это позволяет реализовать централизованное управление запросами, балансировку нагрузки и настраивать схемы взаимодействия между микросервисами без необходимости в дополнительных внешних инструментах.

2.2.4. Entity Framework

Entity Framework [4] — это инструмент для работы с базами данных, который реализует подход ORM (Object-Relational Mapping). Этот фреймворк позволяет взаимодействовать с базами данных через объектно-ориентированные модели, автоматически преобразуя данные между объектами кода и реляционными таблицами. Для подключения в Entity Framework в качестве интерфейса для работы с базой данных используется класс DbContext, через который осуществляется подклю-

чение к базе данных и выполнение запросов. Он хранит в себе свойства типа `DbSet<TEntity>`, где `TEntity` представляет класс, соответствующий таблице. `DbContext` также хранит различные ограничения и аннотации для свойств классов, представляющих таблицы. Это позволяет работать с базой на более абстрактном уровне, избавляя от необходимости писать сложные SQL-запросы вручную и упрощая поддержку кода.

2.2.5. PostgreSQL

PostgreSQL [10] — популярная система управления базами данных с открытым исходным кодом. Она широко используется благодаря поддержке сложных запросов и расширенной функциональности. Автор проекта выбрал PostgreSQL, поскольку уже имел опыт работы с данной системой, а также из-за её возможности эффективно обрабатывать сложные операции, что особенно важно для микросервисной архитектуры приложения.

2.2.6. RabbitMQ

Для организации асинхронного взаимодействия между микросервисами системы был использован RabbitMQ — брокер сообщений, реализующий модель очередей. Выбор этого решения обусловлен следующими факторами: поддержка различных шаблонов обмена сообщениями, наличие встроенных средств управления надёжной доставкой сообщений, а также стабильная интеграция с платформой .NET через официальную клиентскую библиотеку.

В рамках архитектуры проекта RabbitMQ используется в качестве шины событий, обеспечивающей передачу сообщений между сервисами, запуск фоновых обработчиков и синхронизацию распределённых процессов. Использование брокера позволяет декомпозировать взаимодействие между компонентами, снизить связанность и упростить масштабирование отдельных элементов системы.

Выбор RabbitMQ был сделан на основе требований к гибкости марш-

рутизации сообщений и совместимости со стеками, используемыми в проекте. В отличие от Kafka⁷, ориентированной на высокую пропускную способность и долговременное хранение, и ZeroMQ⁸, акцентирующей внимание на минимальных накладных расходах, RabbitMQ предоставляет баланс между функциональностью и простотой настройки, что соответствовало целям и масштабу учебного проекта.

2.2.7. Docker

Для удобного развертывания базы данных, сервера и клиента на разных платформах использовалась контейнеризация. Docker [2] позволяет создать изолированные контейнеры для каждого сервиса, их баз данных, а также обратного прокси, автоматизируя их запуск и настройку. Для управления зависимыми контейнерами применялся Docker Compose [3], который обеспечивает простую настройку и отлично подходит для небольших проектов по сравнению с Kubernetes⁹. Кроме того, Kubernetes не поддерживается большинством облачных провайдеров как готовое решение, что требует аренды виртуальных машин для развертывания и настройки кластера.

⁷Kafka. URL: <https://kafka.apache.org> (дата обращения: 2025-05-24)

⁸ZeroMQ. URL: <https://zeromq.org> (дата обращения: 2025-05-24)

⁹Kubernetes. URL: <https://kubernetes.io> (дата обращения: 2025-05-24)

3. Проектирование

Для реализации веб-приложения сначала было необходимо спроектировать микросервисную архитектуру (выделить отдельные сервисы, продумать их взаимодействие), базу данных и макеты пользовательского интерфейса.

Архитектура веб-приложения (представлено на Рис. 2) была спроектирована с учетом дальнейшего развития. В качестве пользовательского интерфейса представлены мобильное и веб приложения. Они посылают запросы в сервис обратного прокси, который обрабатывает их и пересылает к соответствующему сервису. Микросервисная архитектура включает в себя следующие сервисы: сервис аутентификации, отображения тем учебных практик, сервис для хранения различных артефактов, связанных с написанием учебных практик, таких как текст, отчеты, комментарии, а также сервис для рецензирования учебных практик.

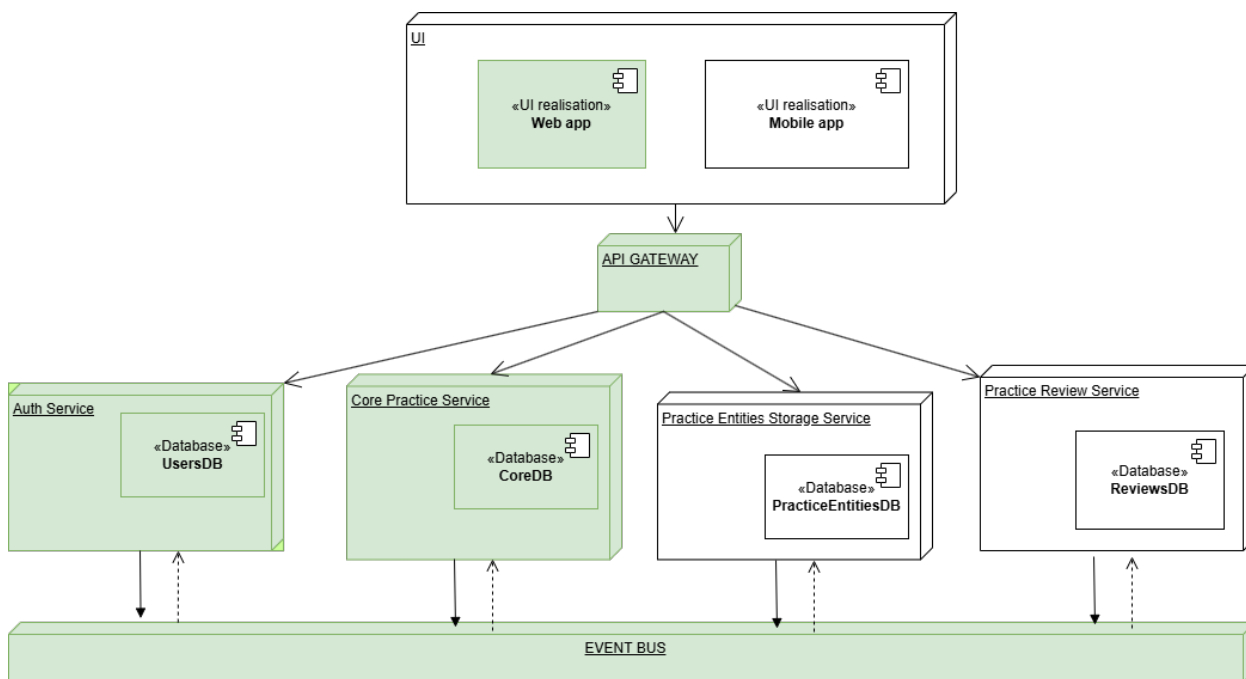


Рис. 2: Диаграмма компонентов приложения. Зеленым цветом выделены компоненты, которые реализованы на данный момент

Для основного сервиса была спроектирована и реализована следующая схема базы данных (представлено на Рис. 3). Здесь представлены таблицы Themes, которая хранит в себе темы учебных практик,

Practices — данные об учебной практике, Groups — данные группы и образовательной программы, Lecturers — сущности преподавателей и их возможность руководить ВКР, Students — сущности студентов и их связь с группой, а также Consultants — сущности консультантов.

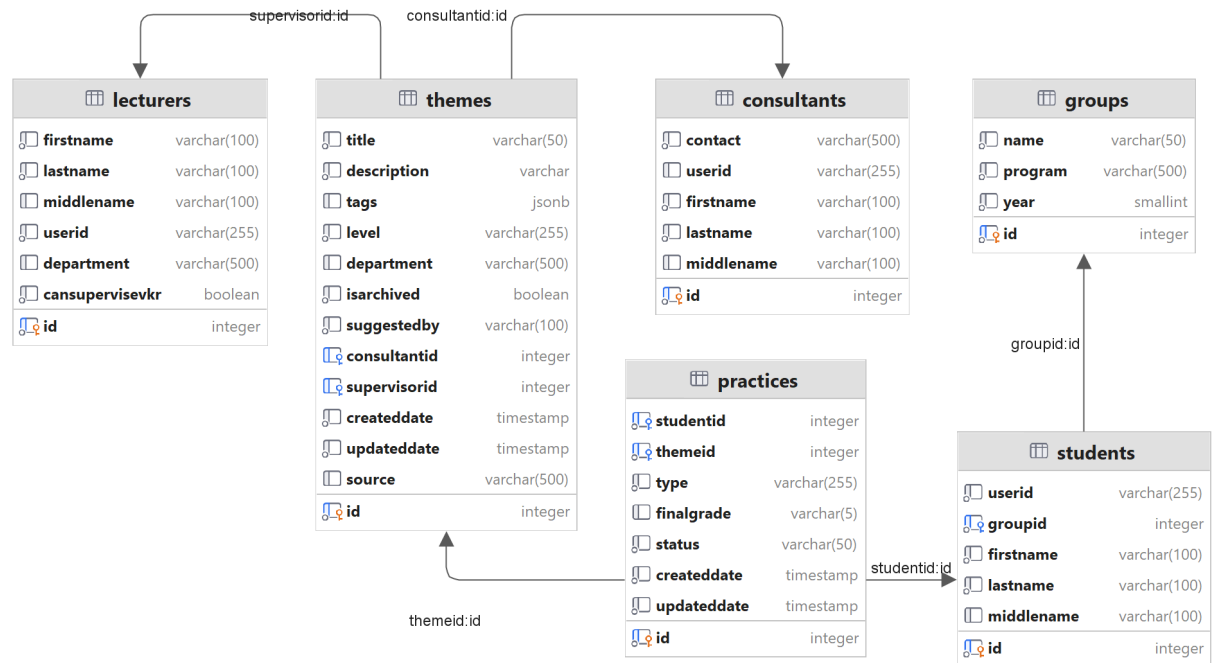


Рис. 3: Схема базы данных основного сервиса

Для реализации было необходимо спроектировать пользовательский интерфейс для веб-приложения. В основу легли абстракции текущего сервиса с рядом поправок, полученных в требованиях. Поскольку основной упор данной работы делается на саму реализацию, а не дизайн, было принято решение использовать сервис для проектирования (представлено на Рис. 4) Ninjamock [7], который позволяет сфокусироваться на основных компонентах без тонкостей.

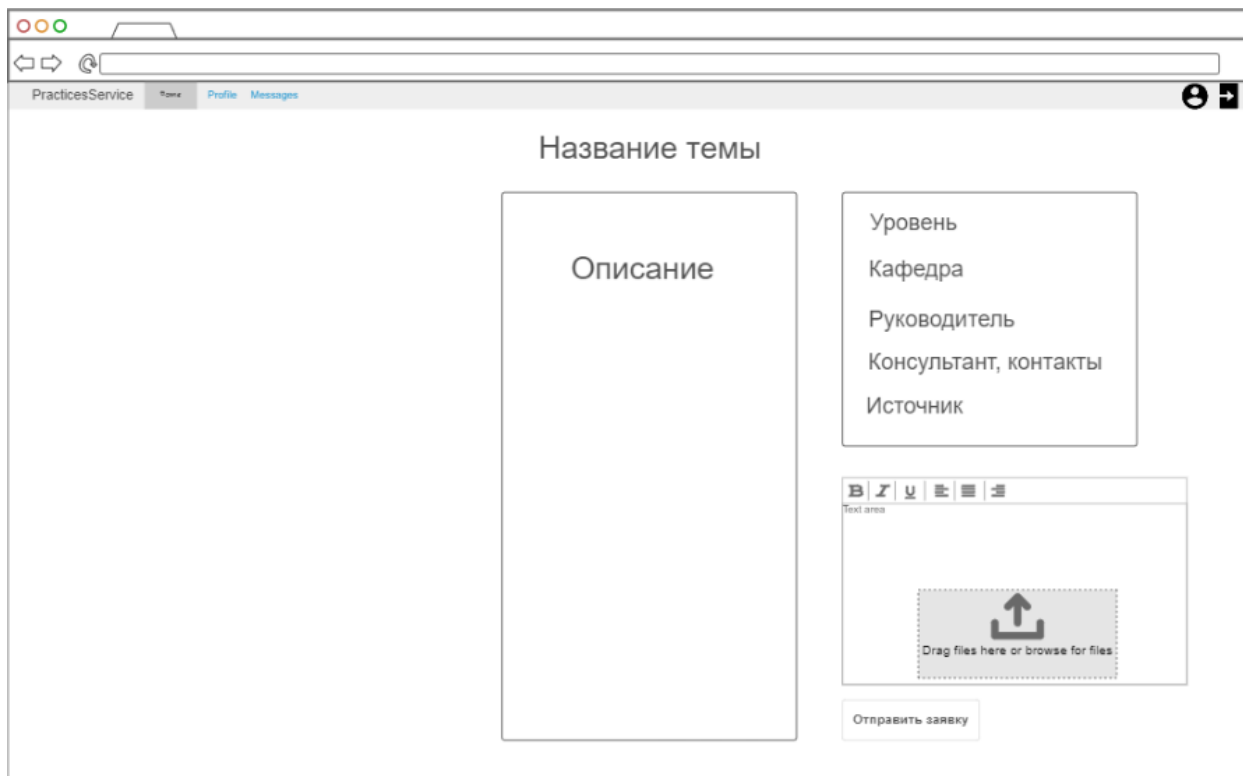


Рис. 4: Мокап страницы отображения темы

4. Реализация

4.1. Серверная часть

Было принято решение начать разработку с основного сервиса, чтобы в дальнейшем можно было привлечь других студентов для реализации вспомогательных сервисов и интеграции с ним.

В основе реализации лежала разработка логики веб-приложения. Для этого были созданы модели сущностей, а также класс `ScheduleContext`, наследующий `DbContext`. В данном классе были определены свойства типа `DbSet<TEntity>`, обеспечивающие связь сущностей с базой данных. Строка подключения к базе данных хранится в файле «`appsettings.json`». Эти компоненты позволили обеспечить корректное взаимодействие приложения с базой данных и сформировать основу для реализации API.

Для построения API были разработаны классы, содержащие необходимые запросы к базе данных для каждой сущности. Затем были до-

бавлены соответствующие маршруты в API, организованные в группы по типам сущностей.

Для упрощения взаимодействия между сервисами и создания единой точки доступа был реализован сервис API Gateway обратного прокси. API Gateway выполняет функции маршрутизации запросов к соответствующим внутренним сервисам, а также добавляет возможность настройки авторизации и валидации на уровне шлюза. Маршруты API Gateway настраивались с помощью конфигурационных файлов и NuGet [8] пакета Yarp.ReverseProxy, что позволяет гибко изменять логику перенаправления запросов. Кроме того, была выполнена настройка преобразований в API Gateway для изменения входящих и исходящих запросов. Это позволило унифицировать структуру запросов и ответов, обеспечивая согласованность между различными сервисами.

Контейнеризация приложения была выполнена с использованием Docker. Каждый компонент системы, включая API, базу данных, API Gateway и вспомогательные сервисы, был упакован в отдельный контейнер. Это позволило создать изолированную среду для работы приложения и упростить его развертывание. Для оркестрации контейнеров применялся Docker Compose, а контейнеры были объединены сетевым мостом.

Затем был разработан сервис авторизации и аутентификации. Для его реализации использовалась ASP.NET Identity, встроенная в ASP.NET система аутентификации и авторизации, которая позволяет пользователям создавать учётные записи, аутентифицироваться, а также управлять учётными записями. Были добавлены и настроены политики авторизации. Кроме того, для внедрения аутентификации в API Gateway потребовалось создать обработчик аутентификации GatewayAuthHandler, который использовался во избежание повторной проверки токена внутри микросервисов.

Для поддержания согласованности данных в микросервисной архитектуре был интегрирован брокер сообщений RabbitMQ, обеспечивающий асинхронную коммуникацию между сервисами. Были разработаны унифицированные события, включающие стандартизированные модели

сообщений и механизмы их обработки.

4.2. Пользовательский интерфейс

Для реализации пользовательского интерфейса был применён инструмент сборки Vite [11], с использованием которого создан клиентский проект на основе библиотеки React. Для осуществления HTTP-взаимодействия с серверной частью был сконфигурирован объект Axios, предусматривающий передачу access-токена в заголовке запроса. Аутентификация и авторизация пользователя реализованы на клиентской стороне, в том числе посредством страницы входа (см. Рис. 5), которая принимает учетные данные пользователя и инициирует запрос к API с целью получения пары токенов: access и refresh.

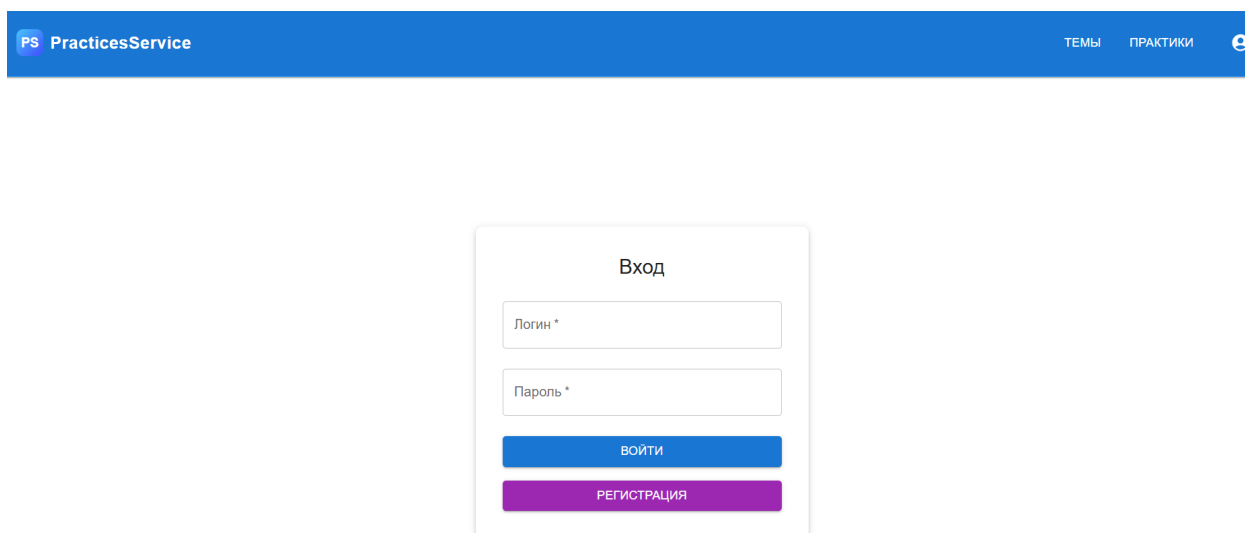


Рис. 5: Страница входа

Также были реализованы интерфейсные страницы для отображения списка тем учебных практик (см. Рис. 6). В левой части экрана размещён блок фильтрации, а также управляющая кнопка, обеспечивающая переход на страницу создания новой темы. Для каждой темы, принадлежащей текущему пользователю, предусмотрены управляющие элементы, позволяющие редактировать соответствующую запись или переместить её в архив. Дополнительно, при взаимодействии с иконкой поль-

зователя отображается выпадающее меню, содержащее опцию выхода из системы через кнопку «Выйти».

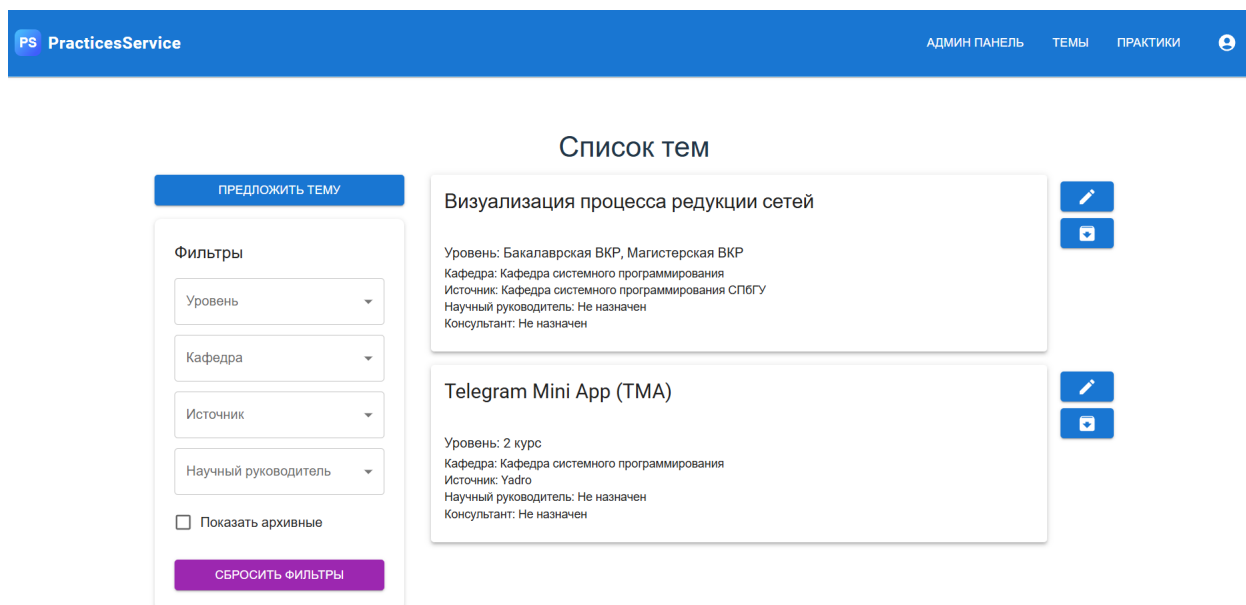


Рис. 6: Страница отображения тем учебных практик

Кроме того, были реализованы страницы написания учебных практик (представлено на Рис. 7). Страница отображения учебных практик предоставляет возможность просматривать как актуальные, так и завершённые учебные практики, а также создавать новые записи.

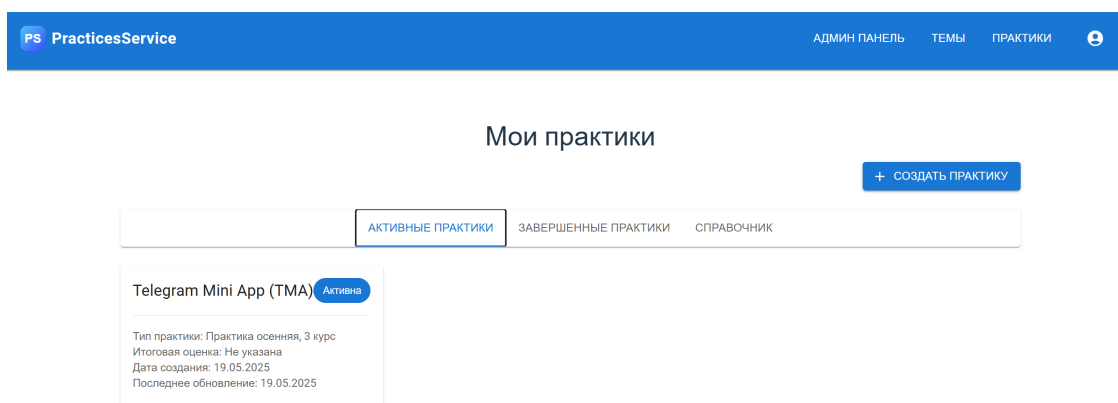


Рис. 7: Страница отображения учебных практик

5. Эксперимент

Для обеспечения качества разработанного веб-приложения был проведен комплекс мероприятий по тестированию. Поскольку приложение имеет микросервисную архитектуру было проведено модульное тестирование каждого сервиса с использованием библиотеки NUnit [6].

Стоит отметить, что были реализованы тесты, проверяющие работоспособность системы в целом, миграции и другие файлы не были протестированы. Поэтому процент покрытия составляет 51% (представлено на Рис. 8). Реализованные модульные тесты были внедрены в процесс непрерывной интеграции, что позволяет использовать их в качестве регрессионных тестов — они автоматически проверяют ранее протестированные сценарии на наличие ошибок при внесении изменений в кодовую базу.

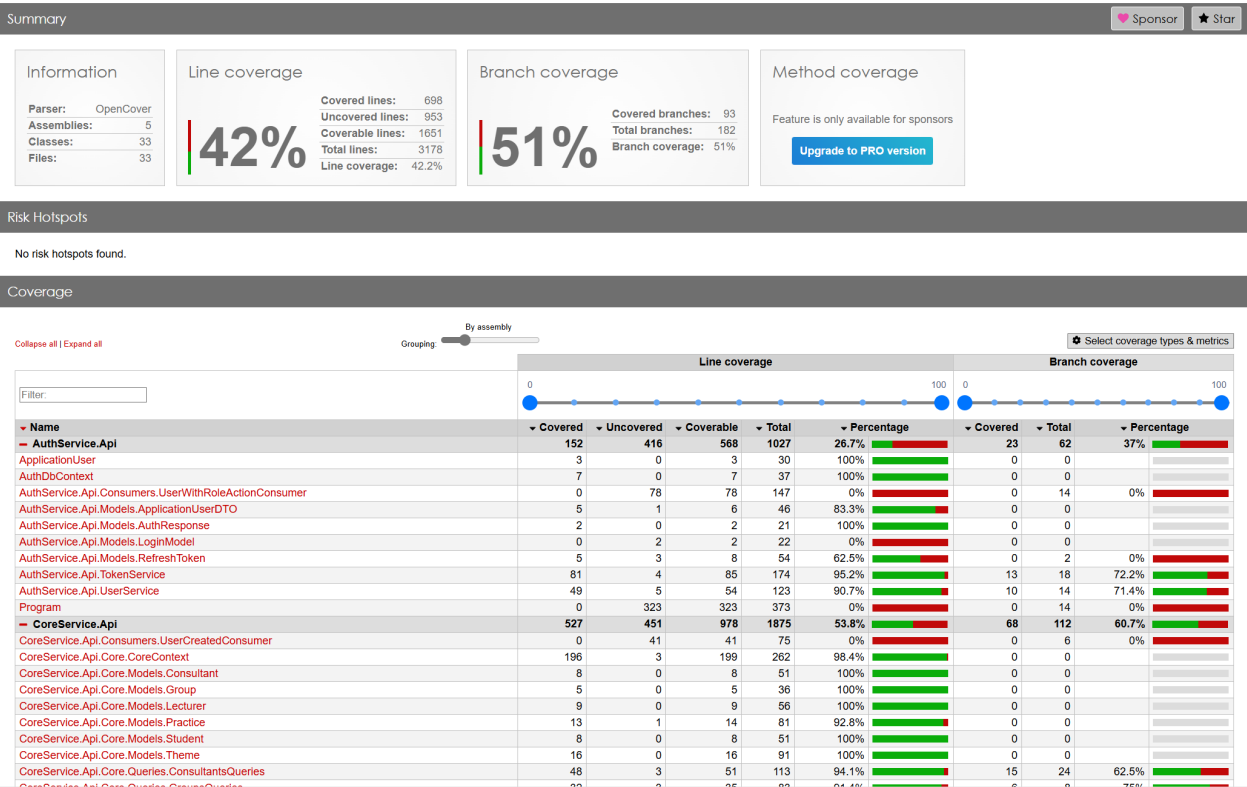


Рис. 8: Тестовое покрытие серверной части

Кроме того, для автоматизации сквозного тестирования использовалась широко используемая библиотека Playwright [9]. Был проведен ряд

тестов, которые позволяют оценить качество пользовательского опыта и реакцию интерфейса на действия пользователя.

6. Заключение

В ходе выполнения данной работы были получены следующие результаты.

- Проведен обзор аналогичных систем.
- Спроектирована общая архитектура системы и пользовательский интерфейс.
- Реализован основной сервис по добавлению/отображению тем и учебных практик.
- Разработана модульная авторизация, настроены ролевые модели, политики.
- Реализован пользовательский интерфейс.
- Проведено тестирование реализованного сервиса.

Код доступен в репозитории на GitHub¹⁰.

¹⁰PracticesService. URL: <https://github.com/spbu-se/PracticesService> (дата обращения: 2025-05-24). [Имя аккаунта: Belgrak]

Список литературы

- [1] ASP.NET Core. — URL: <https://learn.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-8.0> (дата обращения: 2025-05-24).
- [2] Docker. — URL: <https://www.docker.com> (дата обращения: 2025-05-24).
- [3] Docker Compose. — URL: <https://docs.docker.com/compose> (дата обращения: 2025-05-24).
- [4] Entity Framework. — URL: <https://learn.microsoft.com/ru-ru/ef> (дата обращения: 2025-05-24).
- [5] .NET. — URL: <https://dotnet.microsoft.com/ru-ru> (дата обращения: 2025-05-24).
- [6] NUnit. — URL: <https://nunit.org> (дата обращения: 2025-05-24).
- [7] Ninjamock. — URL: <https://ninjamock.com> (дата обращения: 2025-05-24).
- [8] NuGet. — URL: <https://www.nuget.org/packages> (дата обращения: 2025-05-24).
- [9] Playwright. — URL: <https://playwright.dev> (дата обращения: 2025-05-24).
- [10] PostgreSQL. — URL: <https://www.postgresql.org> (дата обращения: 2025-05-24).
- [11] Vite. — URL: <https://vitejs.dev> (дата обращения: 2025-05-24).
- [12] YARP. — URL: <https://microsoft.github.io/reverse-proxy> (дата обращения: 2025-05-24).
- [13] Слугин А. Н. Разработка веб-сервиса для написания учебных практик и выпускных квалификационных работ. — URL: <https://se.>

math.spbu.ru/thesis_download?thesis_id=1288 (дата обращения:
24 мая 2025 г.).