Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 25.М71-мм

Разработка сервиса для работы с учебными практиками

Каргин Глеб Павлович

Отчёт по учебной практике
в форме «Производственное задание»

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

Санкт-Петербург
2026

Оглавление

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.

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 пользо-

вателей.

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. Разработка данного сервиса была начата в рамках выпускной квалификационной работы [9], однако масштаб проекта требует его дальнейших реализации и развития.

1. Постановка задачи

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

1. Исправить имеющиеся недочеты системы первой версии.
2. Составить план дальнейшего развития проекта.
3. Улучшить документацию проекта.
4. Начать реализацию сервиса нотификации для проекта.

2. Обзор

2.1. Обзор существующих решений

- **Сервис учебных практик кафедры системного программирования**[1]. На момент начала разработки на кафедре эксплуатировалась действующая версия веб-сервиса для сопровождения учебных практик и выпускных квалификационных работ (Рис. 1). Система реализовывала базовую функциональность: ведение реестра тем, их распределение среди студентов, процесс рецензирования и сопровождения работ [10]. В ходе анализа были выявлены архитектурные и функциональные ограничения, затрудняющие развитие системы: монолитная архитектура, высокая связанность компонентов, отсутствие механизмов архивации данных по завершении учебного периода, а также сложности интеграции с другими кафедральными сервисами.

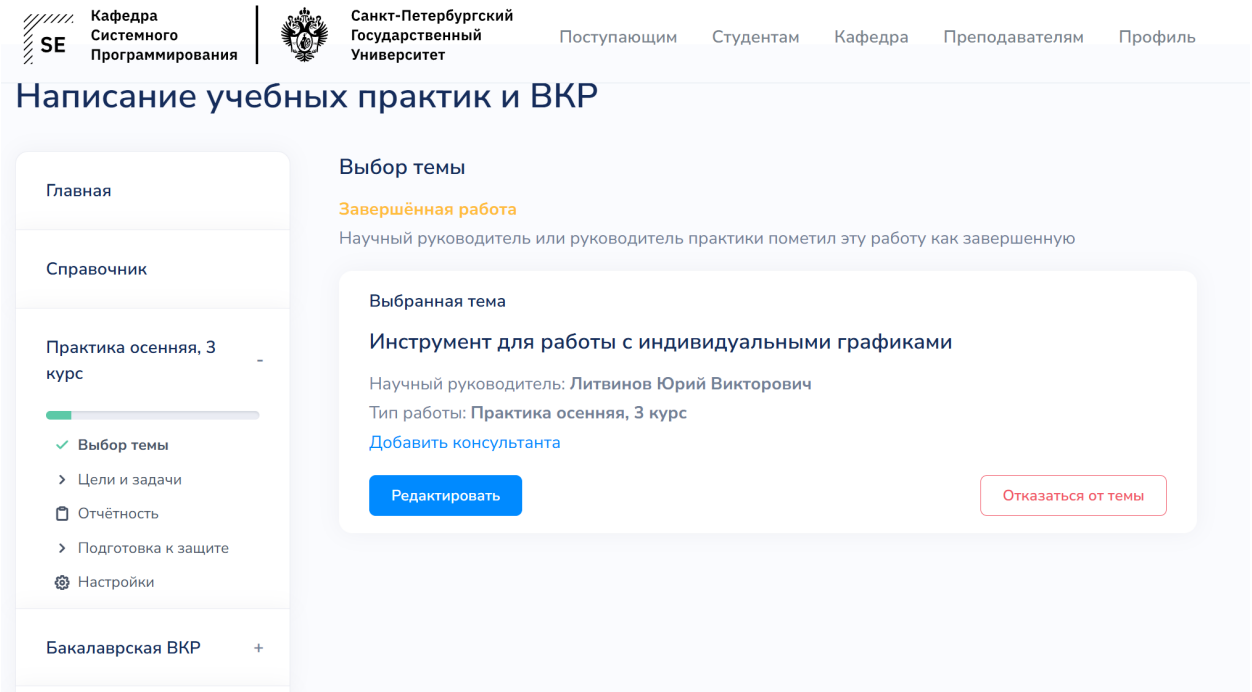

Рис. 1: Интерфейс действующего сервиса учебных практик

[1]Сервис учебных практик и ВКР. URL: https://se.math.spbu.ru/practice (дата обращения: 16.01.2026)

- **Системы управления проектами (Jira)**[2]. Инструменты управления проектами предлагают развитые механизмы управления задачами: создание тикетов, назначение исполнителей, контроль сроков, комментирование, вложение файлов. Их адаптация для образовательных процессов требует существенной доработки и не обеспечивает готовых решений для учёта академических ролей (студент, научный руководитель), автоматизации распределения тем.

- **Системы контроля версий (GitHub)**[3]. В образовательном контексте системы контроля версий могут использоваться для сдачи текстов работ через механизм pull request, однако данные системы не содержат встроенных функций для администрирования учебного процесса, управления темами и взаимодействия с преподавателями вне контекста репозитория.

- **Облачные офисные платформы (Google Workspace, Яндекс)**[4]. Наборы сервисов для совместной работы (документы, календари, дисковые хранилища) предоставляют базовые возможности для коммуникации и обмена материалами между студентами и научными руководителями. Их функциональность не покрывает потребности в структурированном управлении практиками: отсутствуют механизмы автоматического распределения, контроля этапов выполнения, формирования отчётности и интеграции с системами вуза.

Проведённый анализ показал, что существующие решения либо не обладают необходимой специализированной функциональностью, либо требуют значительных доработок для адаптации к процессу управления учебными практиками. Это привело к необходимости разработки новой системы с учётом специфических требований кафедры.

[2]Jira. URL: https://www.atlassian.com/ru/software/jira (дата обращения: 16.01.2026)

[3]GitHub. URL: https://github.com (дата обращения: 16.01.2026)

[4]Google Workspace. URL: https://workspace.google.com (дата обращения: 16.01.2026); Яндекс.Календарь. URL: https://calendar.yandex.ru (дата обращения: 16.01.2026)

2.2. Обзор применяемых технологий

2.2.1. Платформа .NET и ASP.NET Core

В качестве базовой платформы выбрана .NET [5], что соответствует технологическим стандартам, принятым на кафедре, и обеспечивает поддержку большого числа студентов и преподавателей, знакомых с данной экосистемой. Язык программирования C# используется в связи с его распространённостью в университетских проектах. Для реализации веб-сервисов применён фреймворк ASP.NET Core [1] с использованием Minimal API, что позволяет создавать конечные точки с автоматической сериализацией данных в JSON, минимизируя количество шаблонного кода по сравнению с классическим подходом на контроллерах.

2.2.2. Микросервисная архитектура и YARP

Архитектура системы построена на принципах микросервисов, что обеспечивает декомпозицию функциональности на независимые, слабосвязанные компоненты. Для маршрутизации HTTP-запросов между микросервисами используется YARP (Yet Another Reverse Proxy) [8] — обратный прокси, интегрируемый в среду .NET. Его применение позволяет централизованно управлять правилами маршрутизации, реализовывать балансировку нагрузки и динамически конфигурировать взаимодействие сервисов.

2.2.3. Entity Framework Core и PostgreSQL

Для объектно-реляционного отображения используется Entity Framework Core [4]. Данный фреймворк абстрагирует работу с реляционной базой данных через сущности (`DbSet<TEntity>`) и контекст данных (`DbContext`), что упрощает выполнение CRUD-операций и управление схемой данных. В качестве системы управления базами данных выбрана PostgreSQL [6] — открытая реляционная СУБД, поддерживающая сложные запросы и транзакции, что соответствует требованиям к надёжности хранения академических данных.

2.2.4. RabbitMQ

Для организации асинхронного обмена сообщениями между микросервисами внедрён брокер RabbitMQ. Он реализует модель очередей сообщений и используется для синхронизации данных, запуска фоновых задач и обеспечения согласованности в распределённой системе. Выбор RabbitMQ обусловлен его совместимостью с .NET, наличием механизмов гарантированной доставки и гибкостью маршрутизации.

2.2.5. Docker и Docker Compose

Контейнеризация компонентов системы выполнена с использованием Docker [2], что создать изолированные контейнеры для каждого сервиса, их баз данных, а также обратного прокси, автоматизируя их запуск и настройку. Оркестрация многоконтейнерного развёртывания осуществляется инструментом Docker Compose [3], который упрощает определение зависимостей, настройку сетевого взаимодействия и управление жизненным циклом приложения в средах разработки и тестирования.

2.2.6. Frontend: React и Vite

Пользовательский интерфейс реализован на библиотеке React с использованием TypeScript для обеспечения типизации. Сборка пользовательского интерфейса приложения осуществляется с помощью инструмента Vite, который предоставляет быструю разработку с Hot Module Replacement (HMR) и оптимизированную сборку для режима.

2.3. Анализ системы версии ВКР

На начальном этапе эксплуатации версии MVP (Minimum Viable Product) [5] был выявлен ряд архитектурных недочетов и технических проблем, требующих устранения для обеспечения корректной работы системы в промышленной эксплуатации.

[5]MVP. URL: https://ru.wikipedia.org/wiki/Минимально_жизнеспособный_продукт (дата обращения: 16.01.2026)

2.3.1. Проблемы синхронизации данных и ролевой модели

Ключевой архитектурной проблемой являлось отсутствие полной согласованности данных между микросервисами. В системе реализована раздельная модель хранения пользователей:

- В сервисе аутентификации и авторизации пользователи представлены сущностью `ApplicationUser`, которой назначаются роли (администратор, студент, научный руководитель, консультант и другие).
- В основном доменном сервисе (Core Service) для этих же пользователей существуют отдельные сущности (например, `Student`, `Lecturer`, `Consultant`), содержащие специфические для предметной области данные.

В версии MVP изменение ролей пользователя через панель администратора не приводило к автоматическому обновлению связанных сущностей. Отсутствие механизма асинхронного обмена сообщениями приводило к нарушению целостности данных: при добавлении или удалении роли в одном сервисе соответствующие профили в другом сервисе не создавались и не деактивировались. Для решения этой проблемы возникла необходимость внедрения новых издателей и потребителей для брокера сообщений RabbitMQ для передачи событий изменения состояния (создание, обновление, удаление ролей и сущностей).

2.3.2. Недостатки пользовательского интерфейса

В клиентской части приложения были зафиксированы ошибки верстки и логики отображения (см. Рис. 2). В частности, наблюдались визуальные дефекты, при которых текст выпадающих списков (`MenuItem`) перекрывал подписи полей (`Label`), что затрудняло ввод данных. Также некорректно работал механизм отображения списка студентов, прикрепленных к практике, и отсутствовала валидация предварительных условий при создании сущностей.

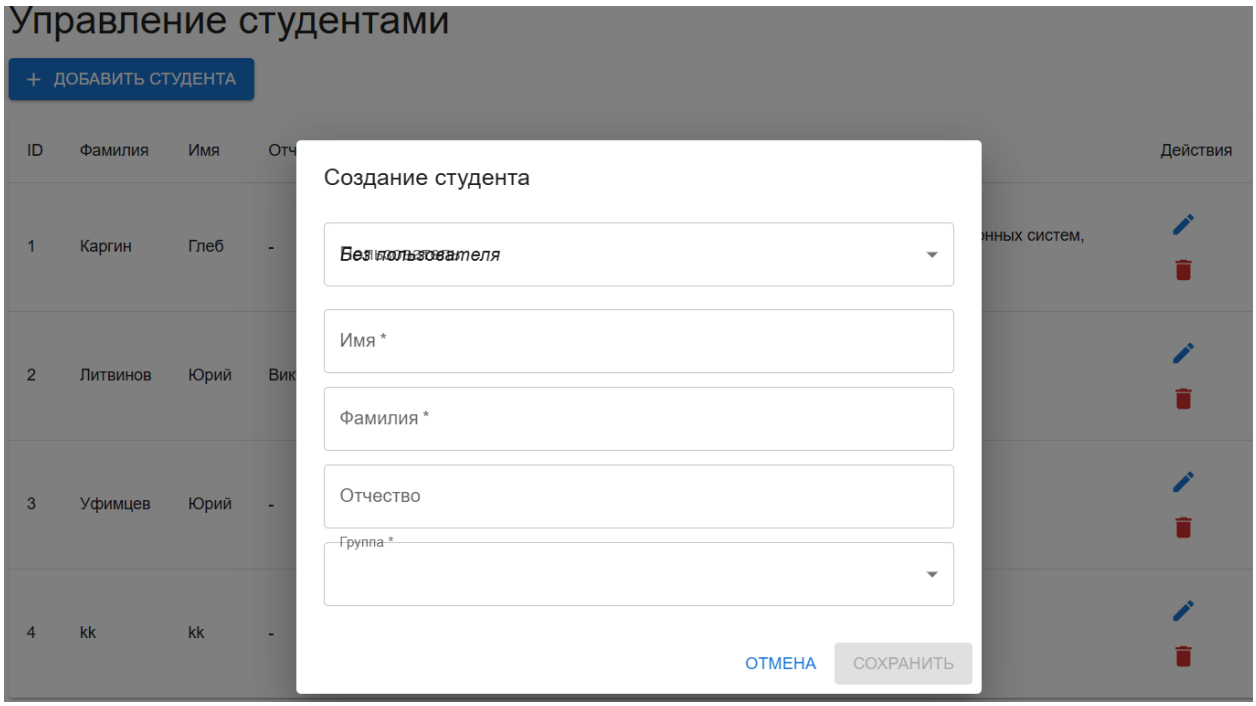

Рис. 2: Накладывание текста в интерфейсе

2.3.3. Проблемы информационной безопасности и развертывания

В ходе аудита информационной безопасности были получены предупреждения (alerts), касающиеся конфигурации инфраструктуры. Файл оркестрации `docker-compose` и конфигурационные файлы сервисов содержали настройки, не соответствующие обновленным требованиям безопасности, что требовало их переработки.

Для устранения выявленных недостатков и развития системы был выполнен комплекс работ по модернизации архитектуры и расширению функциональности.

3. Проектирование

Этап проектирования включал в себя определение архитектурного стиля, декомпозицию системы на микросервисы, разработку схем баз данных и создание прототипов пользовательского интерфейса.

3.1. Архитектура системы

Для обеспечения масштабируемости и возможности независимой разработки модулей была выбрана микросервисная архитектура (см. Рис. 3). Система предполагает наличие веб-клиента и, в перспективе, мобильного приложения. Входной точкой для всех запросов служит шлюз (API Gateway), который выполняет маршрутизацию к функциональным сервисам:

- **Сервис аутентификации (Auth Service)** — управление пользователями и токенами доступа;
- **Основной сервис (Core Service)** — управление темами, практиками и учебными группами;
- **Сервис артефактов** — хранение отчетов, текстов работ и комментариев;
- **Сервис рецензирования** — обеспечение процесса оценки работ.

Однако после использования MVP версии было предпринято внести изменения в архитектуру (см. Рис. 4). Был добавлен сервис нотификации, который позволяет уведомлять пользователей о событиях, а также будет служить инструментом для подтверждения, восстановления аккаунта. Кроме того, для обеспечения мониторинга систем была добавлена база данных, которая посредством потребителя будет сохранить записи о действиях и ошибках внутри микросервисной архитектуры.

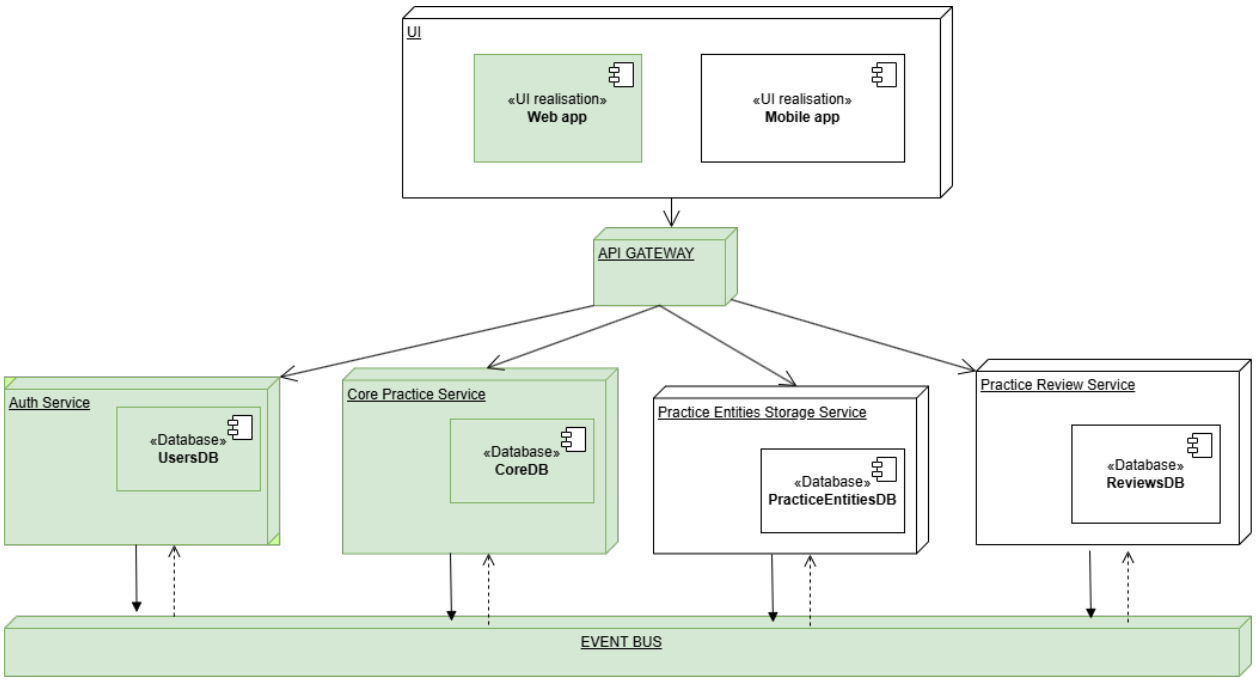

Рис. 3: Предыдущая диаграмма системы. Зеленым цветом выделены реализованные компоненты

4. Реализация

4.1. Серверная часть

4.1.1. Асинхронное взаимодействие и согласованность данных

В системе изначально использовался брокер сообщений RabbitMQ, однако в ходе тестирования версии MVP было выявлено, что не все сценарии изменения состояния пользователей обрабатывались корректно. В частности, изменение ролей пользователя (например, назначение студента или научного руководителя) в административной панели сервиса не приводило к созданию соответствующих профилей в основном сервисе.

Для решения этой проблемы была доработана архитектура событийного взаимодействия:

- Реализованы новые издатели событий в сервисе аутентификации, которые генерируют сообщения при изменении ролевой модели пользователя.

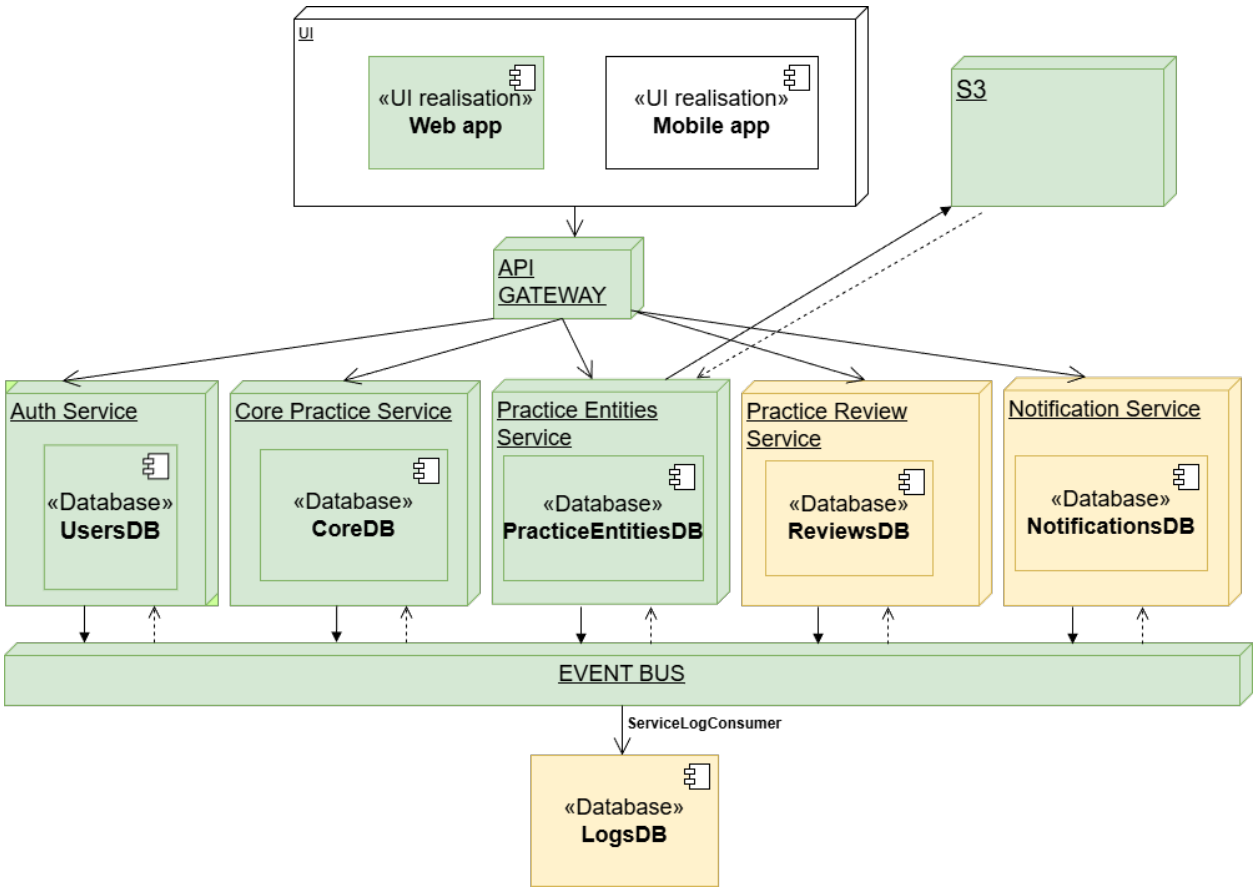

Рис. 4: Обновленная компонентная диаграмма системы

- В основном сервисе внедрены новые потребители, которые при получении события `UserEditedEvent` создают, обновляют или деактивируют сущности `Student`, `Lecturer` или `Consultant`.

Это обеспечило строгую согласованность данных между микросервисами без необходимости синхронных HTTP-вызовов.

4.1.2. Сервис уведомлений

Архитектура системы была расширена новым микросервисом — Notification Service. На текущем этапе реализована интеграция с SMTP-сервером для отправки электронных писем. Это позволило внедрить функциональность восстановления доступа: при запросе сброса пароля сервис генерирует уникальный токен и отправляет его пользователю на электронную почту.

4.1.3. Инфраструктура и безопасность

Контейнеризация компонентов выполнена с помощью Docker. В процессе аудита безопасности конфигурационных файлов были выявлены риски, связанные с настройками сети и портов. В результате был полностью переработан файл `docker-compose`:

- Настройка изолированной внутренней Docker-сети (`proxybackend`) для безопасного взаимодействия микросервисов;
- Вынесение чувствительных данных (учётные записи баз данных, параметры подключения к RabbitMQ) в переменные окружения;
- Замена директивы проброса портов (`ports`) на механизм публикации портов внутри Docker-сети (`expose`) для всех внутренних сервисов, за исключением шлюзового API и интерфейса управления RabbitMQ.

В связи с существенными изменениями в конфигурации развёртывания была произведена полная ревизия документации проекта. Файл `README.md` был полностью переработан с целью актуализации инструкций по запуску системы в новых условиях.

4.2. Пользовательский интерфейс

Клиентская часть приложения («Frontend») реализована как SPA (Single Page Application) с использованием библиотеки React и сборщика Vite [7]. Взаимодействие с API осуществляется через HTTP-клиент Axios, настроенный на автоматическое добавление токена авторизации в заголовки запросов.

4.2.1. Функциональность работы с темами и практиками

В ходе доработки интерфейса были устранены критические ошибки верстки и логики, выявленные на этапе тестирования:

- Исправлены визуальные дефекты наложения текста выпадающих списков (`select`) на подписи полей (`label`).

- Реализована валидация при создании практики: добавлено строгое ограничение, обязывающее пользователя выбрать учебную группу перед сохранением, что исключает появление некорректных записей.
- Внедрен механизм фильтрации табличных данных, упрощающий поиск тем и студентов.
- Исправлена ошибка отсутсвия отображения студента, прикрепленного к конкретной практике.
- Обновлен интерфейс страницы профиля (см. Рис. 5).

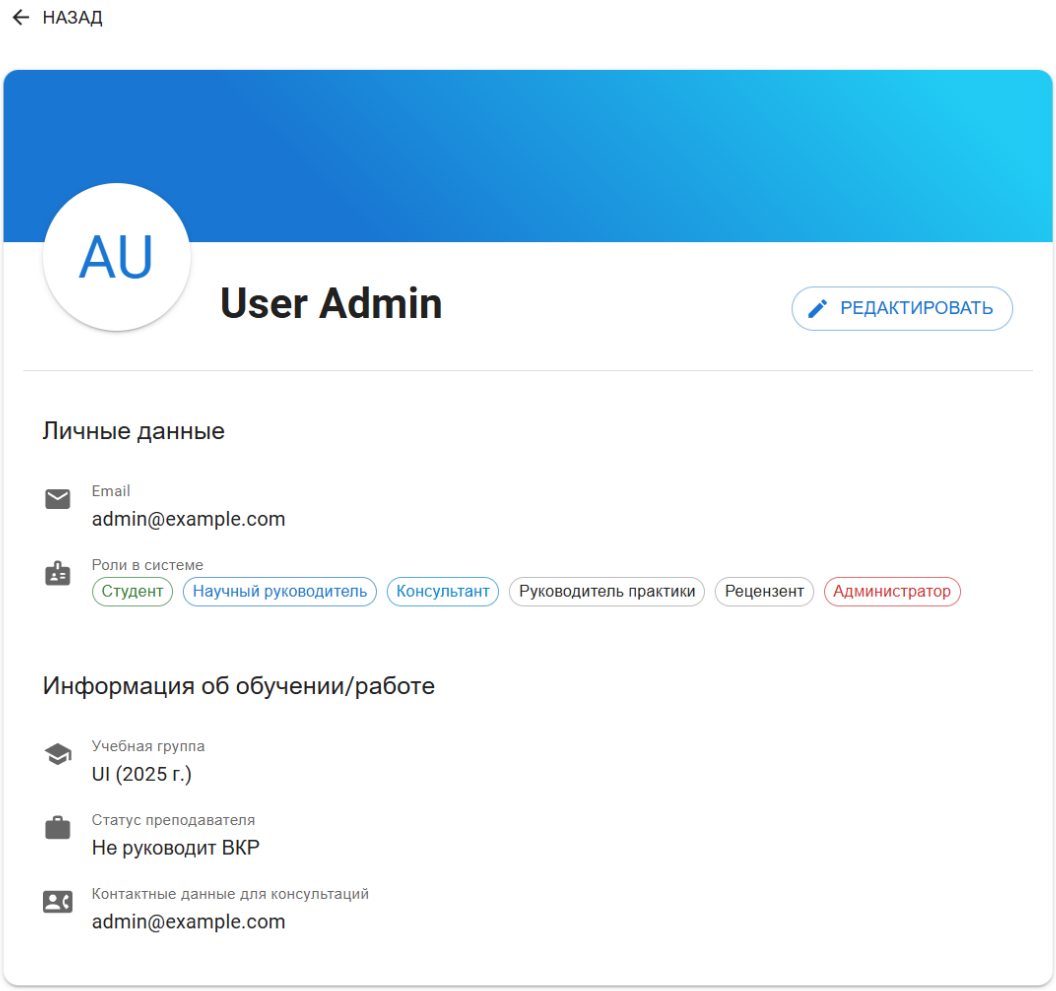

Рис. 5: Обновленная страница профиля

5. Заключение

В ходе выполнения данной работы были получены следующие результаты.

- Проведен обзор аналогичных систем.
- Были выявлены и исправлены имеющиеся недочеты системы.
- Обновлена архитектура системы для дальнейшего реализации проекта.
- Обновлен README.
- Начата реализация сервиса нотификации для проекта.

Код доступен в репозитории на GitHub[6].

[6]PracticesService. 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: 2026-01-16).

[2] Docker. — URL: https://www.docker.com (дата обращения: 2026-01-16).

[3] Docker Compose. — URL: https://docs.docker.com/compose (дата обращения: 2026-01-16).

[4] Entity Framework. — URL: https://learn.microsoft.com/ru-ru/ef (дата обращения: 2026-01-16).

[5] .NET. — URL: https://dotnet.microsoft.com/ru-ru (дата обращения: 2026-01-16).

[6] PostgreSQL. — URL: https://www.postgresql.org (дата обращения: 2026-01-16).

[7] Vite. — URL: https://vitejs.dev (дата обращения: 2026-01-16).

[8] YARP. — URL: https://microsoft.github.io/reverse-proxy (дата обращения: 2026-01-16).

[9] Каргин Г. П. Разработка сервиса для работы с учебными практиками. — URL: https://se.math.spbu.ru/thesis/texts/Kargin_Gleb_Pavlovich_Bachelor_Thesis_2025_text.pdf (дата обращения: 16 января 2026 г.).

[10] 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: 16 января 2026 г.).