

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

**ИНДИВИДУАЛЬНЫЙ ПЛАН (ЗАДАНИЕ И ГРАФИК)
ПРОВЕДЕНИЯ ПРАКТИКИ**

Али Шериф Ашраф Собхи Яссин

Направление подготовки (код/наименование) 09.03.04 «Программная инженерия»

Профиль (код/наименование) 09.03.04_01 «Технология разработки и сопровождения качественного программного продукта»

Вид практики: научно-исследовательская работа

Тип практики: распределенная

Место прохождения практики ФГАОУ ВО «СПбПУ», ИКНК, ВШПИ, СПб, ул. Политехническая, 29

Руководитель практической подготовки от ФГАОУ ВО «СПбПУ»:

Зайцев Игорь Владимирович, старший преподаватель ВШПИ ИКНК

(Ф.И.О., уч. степень,

должность) Руководитель практической подготовки от профильной

организации: - **Рабочий график проведения практики**

Сроки практики: с 02.09.2024 по 16.12.24

№ п/п	Этапы (периоды) практики	Вид работ	Сроки прохождения этапа (периода) практики
1	Организационный этап	Установочная лекция для разъяснения целей, задач, содержания и порядка выполнения работы	02.09
2	Основной этап	Разработка архитектуры приложения для сбора информации о пожеланиях преподавателей по расписаниям семестра и сессии	03.09-15.12
3	Заключительный этап	Подготовка и защита отчета по работе	16.12

Обучающийся

Али Ш.

/ Али Ш. А.

/ Руководитель практической подготовки
от ФГАОУ ВО «СПбПУ»

Зайцев

/ Зайцев И. В. /

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

Отчет о прохождении научно-исследовательской работы

Али Шериф Ашраф Собхи Яссин

4 курс, 5130904/1010?

09.03.04 «Программная инженерия»

Место прохождения практики: ФГАОУ ВО «СПбПУ», ИКНК, ВШПИ,

СПб, ул. Политехническая, 29

Сроки практики: с 02.09.2024 по 16.12.2024

Руководитель практической подготовки от ФГАОУ ВО «СПбПУ»:

Зайцев Игорь Владимирович, старший преподаватель ВШПИ ИКНК

Руководитель практической подготовки от профильной организации: -

Оценка: зачтено

Руководитель практической подготовки

Руководитель практической подготовки
от профильной организации: -

Обучающийся: *Али Ш. А.* / Али Ш. А. /

Дата: 18.01.2025

СОДЕРЖАНИЕ СОДЕРЖАНИЕ

СПИСОК ИЛЛЮСТРАЦИЙ – 4

ПЕРЕЧЕНЬ ТАБЛИЦ – 4

СПИСОК СОКРАЩЕНИЙ И ОПРЕДЕЛЕНИЙ – 5

ВВЕДЕНИЕ – 6

ГЛАВА 1. АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ ДЛЯ СБОРА ПРЕДПОЧТЕНИЙ ПРЕПОДАВАТЕЛЕЙ – 10

1.1. Обзор существующих систем учета предпочтений преподавателей – 10

1.2. Проблемы и ограничения традиционных методов – 13

1.3. Анализ требований к системе сбора предпочтений – 15

1.4. **Выводы по результатам анализа – 17**

ГЛАВА 2. ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ И ИНСТРУМЕНТОВ – 18

2.1. Выбор языка программирования и фреймворка – 18

2.2. Выбор СУБД и способов взаимодействия – 21

2.3. Выбор технологий для обеспечения безопасности – 23

2.4. **Выводы – 25**

ГЛАВА 3. РАЗРАБОТКА АРХИТЕКТУРЫ И МОДЕЛИ ДАННЫХ ВЕБ-ПРИЛОЖЕНИЯ – 26

3.1. Требования к функциональности системы – 26

3.2. Архитектура приложения – 28

3.3. Структура базы данных – 31

3.4. Диаграммы взаимодействия компонентов – 34

3.5. API-контракты и нефункциональные требования – 35

3.6. **Выводы – 41**

ГЛАВА 4. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ – 42

4.1. Реализация функциональности сбора предпочтений – 42

4.2. Реализация аутентификации и авторизации – 45

4.3. Экспорт данных и работа с отчетами – 50

4.4. Проведение тестирования приложения – 55

4.5. **Выводы – 56**

ЗАКЛЮЧЕНИЕ – 57

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ – 59

ПРИЛОЖЕНИЯ – 60

СПИСОК ИЛЛЮСТРАЦИЙ

- Рисунок 1.** Общая схема архитектуры веб-приложения –
Рисунок 2. Последовательность взаимодействия пользователя с системой –
Рисунок 3. Диаграмма компонентов Spring Boot-приложения –
Рисунок 4. Структура базы данных PostgreSQL –
Рисунок 5. Форма ввода предпочтений преподавателя (веб-интерфейс) –
Рисунок 6. Форма аутентификации пользователя –
Рисунок 7. Страница администратора с общими данными преподавателей –
Рисунок 8. Пример экспорта предпочтений в формате Excel –
Рисунок 9. Сценарий ошибки при неудачном входе в систему –
Рисунок 10. Пример отчета по предпочтениям преподавателей –
Рисунок 11. Круговая диаграмма распределения предпочтений по дням недели –
Рисунок 12. Гистограмма типов занятий (лекции, семинары и т.д.) –
Рисунок 13. Блок-схема обработки данных на стороне сервера –
Рисунок 14. Модель сущностей и связей в БД –
Рисунок 15. Структура папок проекта в среде разработки –

ПЕРЕЧЕНЬ ТАБЛИЦ

- Таблица 1.** Роли пользователей и права доступа –
Таблица 2. Пример записи о предпочтениях преподавателя –
Таблица 3. Описание полей таблицы `semester_preferences` –
Таблица 4. Описание полей таблицы `session_preferences` –
Таблица 5. Форматы экспорта данных и поддерживаемые поля –
Таблица 6. Результаты тестирования системы на примере 10 пользователей –

СПИСОК СОКРАЩЕНИЙ И ОПРЕДЕЛЕНИЙ

ЦОД	Центр обработки данных, комплекс оборудования и инфраструктуры для хранения, обработки и передачи данных. (используется при анализе существующих решений)
ПО	программное обеспечение, совокупность программ для управления аппаратными ресурсами и обработки данных.
ВКР	Выпускная квалификационная работа, итоговая научно-исследовательская работа студента.
БД	База данных, организованная структура хранения информации, используемая для удобного доступа и управления данными.
JDBC	Java Database Connectivity, технология Java для взаимодействия с реляционными базами данных.
UI	User Interface (пользовательский интерфейс), элементы взаимодействия пользователя с программным продуктом.
Spring Boot	фреймворк для упрощённой разработки приложений на Java с минимальной конфигурацией.
Spring Security	модуль фреймворка Spring для обеспечения аутентификации и авторизации пользователей.
REST API	Representational State Transfer Application Programming Interface, архитектурный стиль для взаимодействия между клиентом и сервером через HTTP.
PostgreSQL	объектно-реляционная система управления базами данных с открытым исходным кодом.
Hash-функция	односторонняя функция преобразования данных в уникальное фиксированное значение, используется для защиты паролей.
JWT	JSON Web Token, компактный формат передачи информации между двумя сторонами в виде JSON-объекта с защитой подписью.
CRUD	Create, Read, Update, Delete — базовые операции работы с данными в системах управления базами данных.

ВВЕДЕНИЕ

Актуальность

Современные образовательные учреждения сталкиваются с необходимостью эффективного планирования учебного процесса, учитывая растущие требования к качеству образования и индивидуальным особенностям преподавательского состава. Одним из ключевых факторов успешной организации учебной деятельности является грамотное составление расписания, которое должно максимально учитывать предпочтения преподавателей по времени проведения занятий, видам учебной нагрузки и дополнительным ограничениям.

На практике сбор подобных данных зачастую осуществляется вручную посредством анкетирования или устных опросов, что приводит к значительным временным затратам, увеличению вероятности ошибок и снижению удовлетворенности участников образовательного процесса. Отсутствие автоматизированных систем для сбора и обработки предпочтений преподавателей ограничивает возможности эффективного составления расписаний, особенно в условиях изменения учебных планов и увеличения числа преподавателей.

Развитие веб-технологий и инструментов программирования открывает возможности для создания специализированных приложений, которые обеспечивают оперативный и точный сбор данных, их централизованное хранение и последующую обработку. Внедрение таких решений позволяет повысить качество планирования учебного процесса, сократить административные издержки и улучшить взаимодействие между преподавателями и учебной администрацией.

Таким образом, создание веб-приложения для сбора предпочтений преподавателей является актуальной задачей, способствующей повышению эффективности организации учебной деятельности и качества образовательного процесса в целом.

Цель и задачи исследования

Целью данной выпускной квалификационной работы является разработка веб-приложения для автоматизированного сбора и обработки предпочтений преподавателей с целью повышения эффективности составления учебных расписаний.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ существующих решений для сбора предпочтений преподавателей и выявить их основные недостатки.

2. Определить требования к функциональности разрабатываемого приложения.
3. Обосновать выбор технологий и инструментов для реализации веб-приложения.
4. Разработать архитектуру приложения и модель данных.
5. Реализовать систему аутентификации и авторизации пользователей.
6. Разработать интерфейсы для ввода и редактирования предпочтений.
7. Реализовать механизмы хранения и экспорта данных.
8. Провести тестирование работоспособности приложения на тестовых данных.

Объект и предмет исследования

- **Объект исследования:** процессы сбора, хранения и обработки предпочтений преподавателей в образовательных организациях.
- **Предмет исследования:** разработка и реализация веб-приложения, предназначенного для автоматизации сбора предпочтений преподавателей и обеспечения их интеграции в системы планирования учебного расписания.

Научная новизна

Научная новизна работы заключается в разработке подхода к автоматизации сбора предпочтений преподавателей на базе современных веб-технологий с применением Spring Boot и PostgreSQL, а также в интеграции механизмов аутентификации, авторизации и экспорта данных для повышения гибкости и расширяемости системы.

В отличие от существующих решений, предложенная система ориентирована на удобство ввода данных конечными пользователями (преподавателями) и предусматривает возможность дальнейшего расширения функциональности в зависимости от потребностей образовательного учреждения.

Практическая значимость

Практическая значимость работы заключается в создании прототипа веб-приложения, которое может быть внедрено в образовательные учреждения для оптимизации процесса сбора данных о предпочтениях преподавателей. Применение такого инструмента способствует сокращению времени на организацию учебного процесса, повышению

удовлетворенности преподавательского состава и улучшению качества составляемых расписаний.

Апробация результатов работы

Основные положения и результаты работы планируется представить на научно-практической конференции по информационным технологиям в образовании и обсуждаться в рамках внутренних семинаров кафедры информационных систем университета.

ТРЕБОВАНИЯ К СИСТЕМЕ

Пользовательские истории

1. **Преподаватель** хочет задать и сохранить свои предпочтения по корпусам, аудиториям, времени и типу нагрузки, чтобы при формировании расписания система учитывала его пожелания и не назначала в нежелательные дни или аудитории.
2. **Преподаватель** хочет иметь возможность просмотреть и отредактировать уже сохранённые предпочтения перед началом каждого семестра или сессии, чтобы оперативно вносить изменения без повторного заполнения всех полей.
3. **Администратор** хочет получить список предпочтений всех преподавателей, чтобы легко анализировать загруженность корпусов и аудиторий и более эффективно формировать общее расписание.

Оценка нагрузки и хранения данных

- Число пользователей: до 10 000 активных преподавателей в сутки.
- Период хранения данных: не менее 5 лет (для возможности анализа предпочтений за несколько учебных циклов).

Разработка архитектуры и детальное проектирование

1. Характер нагрузки на сервис:

- Пиковые часы: утро (8–10 ч) и вечер (17–19 ч), при запуске системы планирования и экспорте расписаний.
- Средняя нагрузка: ~ 100 запросов в минуту (6 000 запросов/час).
- Пиковая нагрузка: до 1 000 запросов в минуту.
- Соотношение R/W:

- Чтения (GET): $\approx 55\%$
- Записи (POST, PUT, DELETE): $\approx 45\%$

2. Объемы трафика и дисковой системы:

- Трафик в пике:
 - $1\,000 \text{ req/min} \times 1 \text{ кБ на ответ} \approx 1 \text{ МБ/мин} \rightarrow 60 \text{ МБ/час} \rightarrow 1,4 \text{ ГБ/сутки}$
- Дисковая база (PostgreSQL, JSONB для полей-приоритетов):
 - $10\,000 \text{ преподавателей} \times 10 \text{ записей} \times \sim 2 \text{ кБ/запись} \approx 200 \text{ МБ}$
 - Логи + аудирование: + 100 МБ/год

Запас: 1 ГБ



Диаграмма контекстов (C1)

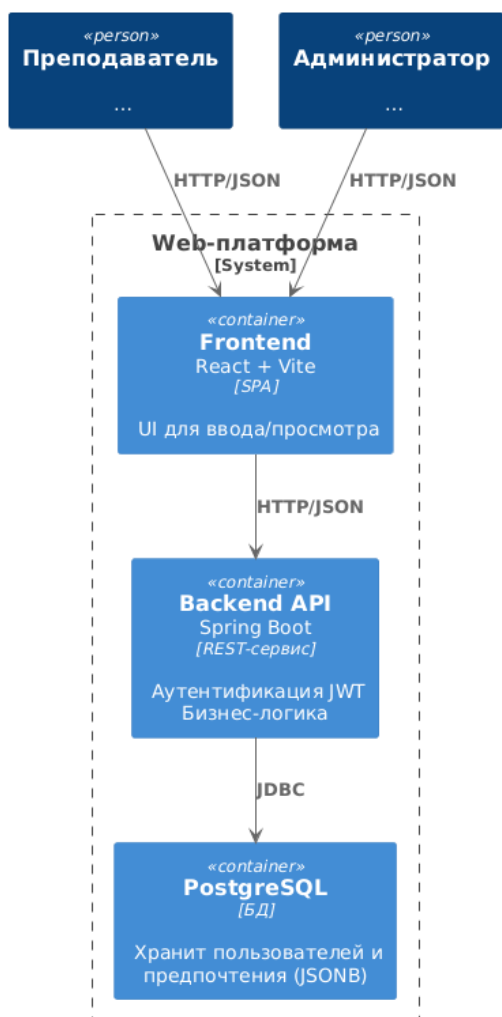


Диаграмма контейнеров (C2)

ГЛАВА 1. АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ ДЛЯ СБОРА ПРЕДПОЧТЕНИЙ ПРЕПОДАВАТЕЛЕЙ

1.1. Обзор существующих систем учета предпочтений преподавателей

Эффективное составление расписания учебных занятий в высших учебных заведениях невозможно без учета индивидуальных предпочтений преподавательского состава. Эти предпочтения могут касаться как времени проведения занятий, так и формата работы, типа занятий (лекция, семинар, лабораторная), а также возможных ограничений, связанных с другими профессиональными или личными обязательствами преподавателя. Следовательно, система учета таких данных должна быть гибкой, точной и простой в использовании как для преподавателей, так и для административного персонала.

На практике в различных образовательных учреждениях применяются разнообразные подходы и программные инструменты для сбора и хранения информации о предпочтениях преподавателей. Эти инструменты можно условно разделить на три основные категории: традиционные бумажные методы,

электронные таблицы и формы общего назначения, а также специализированные программные комплексы и платформы управления обучением.

Бумажные анкеты и ручной ввод

Несмотря на обилие цифровых решений, до сих пор в ряде вузов практикуется использование бумажных анкет. Преподавателям раздают заранее подготовленные бланки, где они заполняют необходимые поля вручную. Этот метод может быть оправдан в условиях отсутствия ИТ-инфраструктуры, но он несет в себе значительные недостатки:

- высокая вероятность ошибок при расшифровке и ручном переносе данных;
- невозможность централизованного хранения и анализа информации;
- отсутствие механизмов контроля целостности и актуальности данных.

Использование электронных таблиц (Excel, Google Sheets)

Более «продвинутым» способом считается рассылка преподавателям шаблонов в формате Excel или ссылок на Google Sheets, где они указывают свои предпочтения. Это позволяет упростить этап сбора, автоматизировать некоторую часть проверки формата данных, а также обеспечить быстрый доступ администраторам. Однако и этот подход имеет ряд ограничений:

- отсутствие встроенной валидации: легко ввести данные в неправильном формате;
- проблемы с безопасностью: открытые документы могут быть случайно удалены или изменены;
- ограниченные возможности для масштабирования: с ростом количества преподавателей работа с такими таблицами становится всё более трудоемкой.

Google Forms и аналоги

Сбор данных с помощью Google Forms стал особенно популярным в последние годы. Преимущества очевидны: доступность, простота настройки, мгновенное отображение результатов в таблицах. Тем не менее, при более глубоком использовании проявляются слабые стороны:

- практически отсутствует контроль за идентификацией пользователя;
- нет гибкости в структуре формы — сложно настроить сложные взаимозависимости между полями;
- отсутствует интеграция с внутренними системами вуза (например, БД или расписанием).

Системы управления обучением (LMS)

Некоторые платформы LMS, такие как **Moodle**, **Canvas**, **Blackboard** или **OpenEdX**, содержат модули, позволяющие организовать опросы и сбор предпочтений. Эти платформы предоставляют больше возможностей в плане безопасности и интеграции с другими модулями. Однако их использование часто сопровождается следующими трудностями:

- необходимость дополнительной настройки и обучения;
- чрезмерная «тяжеловесность» — LMS часто ориентированы на обучение, а не на административную работу;
- отсутствие специализированных функций по составлению расписания, что делает сбор предпочтений лишь побочной задачей.

Специализированные коммерческие системы

Наиболее комплексные решения предлагаются в составе платных программных продуктов, таких как **АСУ «Расписание»**, **Kronos Scheduler**, и др. Некоторые из них содержат модули учета пожеланий преподавателей. Однако:

- системы часто платные и требуют лицензий;
- требуется длительная настройка и обучение персонала;
- сложность адаптации под специфику конкретного вуза.

Промежуточный вывод

Таким образом, на сегодняшний день существует широкий спектр подходов и решений, направленных на учет предпочтений преподавателей. Тем не менее, ни один из них не является универсальным и полностью соответствующим потребностям типичного российского вуза. Это и обосновывает необходимость разработки собственного специализированного веб-приложения, учитывающего все вышеописанные недостатки и предлагающего адаптированное под конкретную задачу решение.

1.2. Проблемы и ограничения традиционных методов

Несмотря на относительную простоту реализации, традиционные методы сбора предпочтений преподавателей, такие как бумажные анкеты, электронные таблицы и простые формы на основе Google Forms, имеют ряд существенных ограничений, препятствующих их эффективному использованию в условиях современных образовательных учреждений.

1. Низкая точность и высокая вероятность ошибок

Один из главных недостатков традиционных методов заключается в **отсутствии встроенной валидации данных**. Вручную заполненные анкеты могут содержать опечатки, недочеты, противоречивые данные (например, указание двух предпочтительных временных интервалов, которые физически совпадают). Даже при использовании Excel или Google Forms преподаватели нередко заполняют поля в неверном формате: вместо времени — текст, вместо выбора типа занятий — произвольный комментарий.

Дополнительно, при ручной обработке бумажных анкет возможны ошибки на этапе переноса информации в электронные таблицы или базы данных. Человеческий фактор играет здесь ключевую роль — чем больше преподавателей, тем выше риск допущения критичных ошибок.

2. Повышенная нагрузка на административный персонал

Поскольку данные, собранные вручную, нуждаются в проверке, очистке и структурировании, это требует значительных временных и трудовых затрат со стороны сотрудников учебного отдела. Например, если в университете работает более 200 преподавателей, а каждый из них заполняет таблицу на 10–15 параметров, то общее количество обрабатываемых единиц данных исчисляется тысячами. Все это зачастую делается вручную или с минимальной автоматизацией, что значительно увеличивает срок подготовки расписания.

3. Отсутствие единого формата данных

При отсутствии централизованной системы возникает проблема **разнородности представления информации**. Разные кафедры могут использовать разные шаблоны, по-разному трактовать те или иные поля. Это приводит к необходимости дополнительного согласования и «сшивания» данных, что еще больше усложняет работу по интеграции предпочтений в расписание.

4. Проблемы с актуальностью и синхронизацией информации

Традиционные методы не предполагают возможности **динамического обновления информации**. Если преподаватель по ошибке указал некорректные данные или его предпочтения изменились, то исправление требует либо повторной подачи анкеты, либо ручной корректировки, что не только неудобно, но и создает риск потери актуальности сведений.

В случае с электронными таблицами ситуация может быть даже хуже: если преподаватель повторно отправит форму, старая запись останется в базе, и система не будет понимать, какую версию считать актуальной.

5. Нарушение конфиденциальности данных

Особенно остро стоит вопрос **безопасности и конфиденциальности** персональных данных преподавателей. Примеры из практики показывают, что при использовании общедоступных Google-форм или электронных таблиц нередко возникает ситуация, когда ссылка на форму передаётся третьим лицам, и в итоге конфиденциальные данные оказываются в открытом доступе. Также возможны ситуации, когда один преподаватель видит предпочтения другого, что нарушает базовые этические нормы.

6. Ограниченные возможности расширения

Традиционные методы практически не позволяют масштабировать систему под растущие потребности. Например, невозможно безболезненно добавить поддержку новой категории предпочтений (например, желаемая аудитория с интерактивной панелью) или интегрировать существующую таблицу с системой автоматического составления расписания. Это требует полной переработки шаблонов и ручного импорта/экспорта.

Вывод

В целом, традиционные методы, хоть и обладают определённой интуитивной понятностью и низким порогом входа, уже не соответствуют современным требованиям к автоматизации образовательного процесса. Проблемы, связанные с надёжностью, скоростью, безопасностью и масштабируемостью, делают их временным решением, приемлемым лишь

в условиях ограниченных ресурсов. Это подчеркивает необходимость перехода к специализированным цифровым платформам, учитывающим специфику работы вузов и преподавателей.

1.3. Анализ требований к системе сбора предпочтений

На основании выявленных проблем и ограничений традиционных методов, а также анализа существующих цифровых решений, можно сформулировать обоснованные требования к создаваемой системе. Эти требования охватывают как функциональные, так и нефункциональные аспекты, отражая потребности всех заинтересованных сторон — преподавателей, администраторов, технических специалистов и руководства образовательного учреждения.

1. Удобство пользовательского интерфейса

Одним из ключевых факторов успешного внедрения любой информационной системы является её **доступность и понятность для конечных пользователей**. В случае системы сбора предпочтений основными пользователями являются преподаватели, у которых может не быть технической подготовки. Следовательно, интерфейс должен быть интуитивно понятным, с минимальным количеством обязательных полей, логичной навигацией, подсказками и валидацией в режиме реального времени.

Важно предусмотреть адаптивный дизайн, позволяющий работать с системой как на настольных компьютерах, так и на мобильных устройствах, что особенно актуально в условиях удалённой работы и гибкого графика преподавателей.

2. Аутентификация и разграничение прав доступа

Для обеспечения безопасности и целостности данных необходимо реализовать систему **аутентификации пользователей** с возможностью разграничения прав доступа. На базовом уровне это может быть реализовано через регистрацию преподавателей с последующим входом по логину и паролю. Более продвинутые варианты включают двухфакторную аутентификацию и интеграцию с внутренней системой авторизации вуза (LDAP, OAuth и др.).

Система должна чётко разграничивать роли пользователей:

- **преподаватели** — имеют доступ только к собственным предпочтениям;

- **администраторы** — могут просматривать, редактировать и экспортировать данные всех пользователей;
- **руководство** — имеет доступ к статистике и аналитике без возможности редактирования.

3. Гибкость в настройке форм сбора данных

Поскольку вузы могут иметь разные требования к структуре расписания, система должна поддерживать **гибкую настройку форм ввода данных**. Преподавателю может потребоваться указать не только предпочтительное время занятий, но и:

- вид нагрузки ;
- желаемый день недели;
- формат занятий (очно, дистанционно);
- пожелания по аудиториям ;
- ограничения по времени (занятость в других подразделениях, здоровье и др.).

Все поля должны быть валидируемыми и, по возможности, выбираемыми из заранее заданных справочников, чтобы минимизировать риск ошибок и повысить удобство заполнения.

4. Централизованное хранение и структурированность данных

Для обеспечения надежности, совместной работы и возможности последующей интеграции с другими информационными системами, данные должны храниться **в централизованной реляционной базе данных**, например, PostgreSQL. Это позволит:

- обеспечить целостность и однозначность данных;
- удобно проводить агрегацию и фильтрацию;
- быстро экспортировать данные по заданным критериям;
- отслеживать историю изменений .

5. Поддержка экспорта и интеграции

Современные вузы часто используют отдельные системы для автоматического составления расписаний, такие как АСУ Расписание, ПланШкола, Крон-График и другие. Поэтому важным требованием

является поддержка экспорта данных в универсальных форматах: Excel (XLSX).

Более того, система должна быть спроектирована с учётом возможной последующей интеграции с внешними API, что позволит сократить ручной труд и исключить ошибки, связанные с копированием данных.

6. Безопасность и защита персональных данных

Система должна соответствовать нормам информационной безопасности, включая требования Федерального закона №152-ФЗ «О персональных данных». Это включает:

- шифрование паролей (с помощью алгоритма BCrypt);
- защита от SQL-инъекций, XSS и CSRF-атак;
- контроль сессий и автоматический выход при бездействии;
- логирование действий пользователей .

7. Масштабируемость и отказоустойчивость

Даже если на первом этапе система используется только в одном факультете или институте, проект должен предусматривать возможность **масштабирования** — как по количеству пользователей, так и по объёму обрабатываемых данных. Это включает:

- оптимизированные запросы к базе данных;
- возможность развертывания в облаке или на собственных серверах;
- горизонтальное масштабирование.

Итог

Формулировка требований на раннем этапе проекта — это ключевой шаг, от которого зависит успешность всей последующей разработки. Учитывая специфику образовательных учреждений и проблемы, выявленные в ходе анализа существующих решений, предложенные требования охватывают как технические, так и пользовательские аспекты системы и могут служить надёжной основой для проектирования архитектуры и выбора технологического стека.

1.4. Выводы по результатам анализа

Анализ существующих подходов к сбору предпочтений преподавателей

показал, что несмотря на наличие широкого спектра инструментов — от бумажных анкет до интеграций с LMS — на практике большинство решений обладают ограничениями, препятствующими их эффективному использованию в условиях современной цифровой трансформации образования.

Традиционные методы, хоть и просты в реализации, сопряжены с высокой вероятностью ошибок, низкой скоростью обработки информации и отсутствием механизмов валидации. Электронные таблицы и формы частично автоматизируют процесс, но не решают проблему безопасного хранения и централизованного управления данными. Даже специализированные программные продукты часто оказываются избыточными, сложными в освоении или слабо адаптируемыми под конкретные требования вузов.

Таким образом, на практике формируется чёткий запрос на создание специализированного веб-приложения, которое бы отвечало следующим критериям:

- Простота использования и доступность для преподавателей без ИТ-компетенций;
- Гибкость настройки форм и сценариев взаимодействия;
- Безопасность хранения и обработки персональных данных;
- Централизованная архитектура и надёжная база данных;
- Возможность экспорта и интеграции с другими информационными системами;
- Масштабируемость и адаптируемость под будущие изменения и расширения.

Важно подчеркнуть, что разработка подобного программного решения не должна рассматриваться как разовая инициатива, ограниченная рамками одного проекта или факультета. Напротив, речь идет о формировании устойчивой цифровой платформы, которая может быть масштабирована и внедрена на уровне всего университета, обеспечивая единый стандарт взаимодействия между преподавателями и учебной администрацией.

На основании вышеизложенного, в следующих главах будет обоснован выбор технологического стека, представлена архитектура создаваемой системы, а также продемонстрированы этапы её реализации и тестирования.

ГЛАВА 2. ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ И ИНСТРУМЕНТОВ

2.1. Выбор языка программирования и фреймворка

Одним из ключевых этапов в разработке любого программного продукта является выбор технологического стека, особенно языка программирования и фреймворка. Эти компоненты напрямую влияют на производительность системы, простоту разработки и сопровождения, безопасность, а также возможности для расширения и масштабирования.

Обоснование выбора языка Java

Для реализации серверной части приложения был выбран язык **Java**. Этот выбор обусловлен несколькими весомыми причинами:

1. Надёжность и зрелость технологии

Java — это один из самых старых и проверенных временем языков программирования, используемый в крупных корпоративных системах, банковской сфере, образовательных платформах. Он обладает стабильным синтаксисом и богатой экосистемой.

2. Кроссплатформенность

Благодаря принципу "Write Once, Run Anywhere", Java-приложения могут выполняться на любой платформе, где установлен Java. особенно важно при возможном будущем переносе проекта на серверы с разной архитектурой.

3. Большое сообщество и доступная документация

Вокруг Java сформировалось мощное профессиональное сообщество, что облегчает поиск решений типовых проблем, повышает доступность обучающих материалов и снижает риски при разработке.

4. Интеграция с промышленными инструментами

Java поддерживается множеством инструментов для сборки (Maven, Gradle), тестирования (JUnit, Mockito), анализа кода (SonarQube) и CI/CD (Jenkins, GitLab CI), что делает её удобной для промышленного использования и долгосрочного сопровождения проекта.

Почему именно Spring Boot

Наиболее логичным выбором фреймворка при работе с Java в контексте веб-разработки является **Spring Boot**. Данный фреймворк предоставляет мощный набор инструментов для создания микросервисов и REST-приложений и поддерживает принципы инверсии управления (IoC) и внедрения зависимостей (DI), что делает код более модульным и тестируемым.

Преимущества Spring Boot:

- **Автоматическая конфигурация**
Spring Boot минимизирует количество шаблонного (boilerplate) кода и избавляет от необходимости настраивать проект вручную.
- **Поддержка REST API из коробки**
Создание REST-контроллеров и маршрутов занимает минимальное количество кода и логично структурируется.
- **Гибкость и расширяемость**
В проект можно легко подключить дополнительные модули (например, Spring Security, Spring Data JPA, Spring WebSocket и др.), что обеспечивает хорошую адаптируемость под любые нужды.
- **Совместимость с базами данных и ORM**
Spring Boot без проблем интегрируется с Hibernate и JPA — стандартными технологиями ORM в Java, позволяя абстрагироваться от SQL и удобно работать с объектной моделью данных.
- **Активная поддержка и документация**
Проект Spring развивается под руководством компании VMware, активно поддерживается и сопровождается подробной документацией.

Анализ альтернатив

В ходе предварительного анализа были рассмотрены и другие возможные варианты:

- **Python + Django** — популярный стек, удобный для быстрого прототипирования, но менее производительный под нагрузкой и требует отдельной конфигурации для развёртывания и безопасности.
- **JavaScript (Node.js + Express)** — лёгкий стек, однако для серьёзных корпоративных решений с высоким уровнем безопасности предпочтительнее строгая типизация и стабильность Java.

- **PHP + Laravel** — мощный фреймворк, но его применяют чаще в проектах с другим профилем (например, блоги, CRM).

Вывод

Учитывая специфику проекта — разработка надёжного, масштабируемого и безопасного веб-приложения для сбора предпочтений преподавателей — выбор Java в связке со Spring Boot представляется оптимальным с точки зрения архитектурной строгости, промышленной зрелости и гибкости расширения проекта в будущем.

2.2. Выбор СУБД и способов взаимодействия

Выбор системы управления базами данных (СУБД) — критически важный этап при проектировании архитектуры любого информационного решения. СУБД определяет, как будет организовано хранение, доступ, целостность и масштабируемость данных, а также каким образом будут реализованы механизмы взаимодействия между приложением и хранилищем информации.

Обоснование выбора PostgreSQL

Для реализации данного проекта была выбрана **объектно-реляционная СУБД PostgreSQL**. Это решение продиктовано рядом технических и практических преимуществ:

1. Поддержка сложных структур данных

PostgreSQL позволяет хранить как структурированные реляционные данные, так и объекты (JSON, массивы, enum-значения). Это особенно удобно при хранении предпочтений, которые могут быть представлены в разных форматах: списках, флагах, наборах параметров.

2. Расширяемость и гибкость

PostgreSQL предоставляет возможность создавать пользовательские типы данных, функции, триггеры, что даёт огромный простор для настройки бизнес-логики на уровне БД. Это особенно важно при необходимости валидации и проверки целостности данных на уровне схемы.

3. Поддержка транзакций и соответствие стандарту ACID

В условиях многопользовательского доступа важно обеспечивать корректное поведение при одновременной работе с данными. PostgreSQL обеспечивает высокий уровень надёжности и консистентности при параллельной работе.

4. Широкая поддержка в Java-среде

PostgreSQL легко интегрируется с Java-проектами через **JDBC** и **Spring Data JPA**, что позволяет реализовать слой доступа к данным (DAO/Repository) в унифицированной и понятной форме, не прибегая к сложному SQL-коду.

5. Активное сообщество и бесплатная лицензия

PostgreSQL распространяется с открытым исходным кодом, что делает его бесплатным для любого типа использования — как в учебных, так и в коммерческих целях. Сообщество активно поддерживает развитие и обновление системы.

Рассмотренные альтернативы

В процессе подготовки архитектуры проекта были проанализированы и другие возможные варианты СУБД:

- **MySQL** — простая в настройке и популярная СУБД, но уступает PostgreSQL в части поддержки сложных типов данных и триггеров. Также имеет менее строгую реализацию транзакционности.
- **SQLite** — удобна для небольших проектов и прототипов, но плохо подходит для многопользовательских систем, где требуется параллельный доступ к данным и высокая производительность.
- **MongoDB** — документно-ориентированная NoSQL СУБД. Несмотря на её гибкость, в контексте данного проекта реляционная модель предпочтительнее из-за чёткой структуры данных и необходимости использования SQL-запросов при построении отчетов.

Выбор способа взаимодействия: JDBC и Spring Data JPA

Для реализации доступа к базе данных используется **Spring Data JPA** — надстройка над **Hibernate**, которая, в свою очередь, работает через **JDBC** (Java Database Connectivity).

Преимущества такого подхода:

- Автоматическое связывание сущностей с таблицами

Использование аннотаций `@Entity`, `@Table`, `@Column` позволяет связать Java-классы напрямую с таблицами БД без необходимости вручную писать SQL-запросы.

- **Реализация CRUD-операций "из коробки"**

Интерфейсы Repository автоматически генерируют методы `findAll()`, `save()`, `deleteById()` и т.д., упрощая работу с данными.

- **Возможность кастомизации**

При необходимости разработчик может самостоятельно писать SQL-запросы или использовать JPQL (Java Persistence Query Language) для более гибкого управления данными.

- **Транзакционность и безопасность**

В связке с Spring Security можно реализовать чёткий контроль доступа к данным, а также обеспечить откат при неудачных операциях благодаря аннотациям `@Transactional`.

Вывод

Выбор PostgreSQL в качестве основной СУБД обусловлен её мощными возможностями, стабильной работой в условиях многопользовательской нагрузки и совместимостью с Java-средой. Использование связки JDBC + JPA обеспечивает удобный, гибкий и безопасный способ взаимодействия между веб-приложением и базой данных.

2.3. Выбор технологий для обеспечения безопасности

Безопасность информационной системы — это не факультативная опция, а **неотъемлемое требование**, особенно в условиях работы с персональными данными пользователей. В случае веб-приложения для сбора предпочтений преподавателей важно обеспечить:

- защиту от несанкционированного доступа;
- сохранность и конфиденциальность вводимой информации;
- безопасное хранение и передачу учетных данных.

Для реализации этих целей в рамках проекта было решено использовать **модуль Spring Security**, встроенный в экосистему Spring Boot, в сочетании с дополнительными мерами безопасности на уровне базы данных, сессий и пользовательского интерфейса.

Использование Spring Security

Spring Security — это промышленный стандарт для обеспечения безопасности в Java-приложениях. Он предлагает гибкую и расширяемую архитектуру, которая позволяет быстро внедрить основные механизмы аутентификации, авторизации и управления сессиями.

Основные функции, реализуемые с помощью Spring Security:

1. Аутентификация пользователей

Поддерживается форма входа с логином и паролем, включая кастомизацию интерфейса. При необходимости можно реализовать интеграцию с внешними поставщиками (Google, GitHub).

2. Авторизация и разграничение прав доступа

Использование аннотаций `@PreAuthorize`, `@Secured`, `@RolesAllowed` позволяет задавать, какие роли пользователей имеют доступ к определённым контроллерам и методам. В проекте реализованы две основные роли: `ROLE_USER` (преподаватель) и `ROLE_ADMIN` (администратор).

3. Шифрование паролей

Для хранения паролей используется алгоритм **BCrypt**, обеспечивающий устойчивость к атакам типа brute force и rainbow tables. Перед сохранением в БД пароль хэшируется, и при входе сравнивается хэш введённого значения.

4. Защита от CSRF-атак

Spring Security по умолчанию предоставляет защиту от подделки межсайтовых запросов (Cross-Site Request Forgery). Это особенно важно при работе с формами, отправляющими POST-запросы.

5. Управление сессиями и тайм-аут

В целях защиты от несанкционированного доступа при оставленном открытом доступе к браузеру, реализован механизм автоматического выхода из системы при бездействии более 30 минут.

Дополнительные меры безопасности

Помимо встроенного функционала Spring Security, проектом предусмотрены дополнительные уровни защиты:

- **Ограничение числа попыток входа**

Система может быть дополнена механизмом временной блокировки аккаунта после N неудачных попыток входа, что защищает от перебора паролей (brute force).

- **Валидация пользовательских данных**
Все формы ввода (регистрация, вход, формы предпочтений) сопровождаются проверками как на стороне клиента (JavaScript), так и на сервере (Java Bean Validation).
- **Шифрование конфиденциальных данных при передаче**
Предполагается использование HTTPS на уровне транспортного протокола. При развёртывании в продакшене SSL-сертификаты обеспечивают защищённую передачу всех данных между клиентом и сервером.
- **Логирование действий пользователей**
В целях отслеживания активности (например, массового экспорта данных) возможно добавление логов для анализа подозрительного поведения и аудита доступа.

Анализ альтернативных решений

Рассматривались и другие инструменты обеспечения безопасности:

- **Apache Shiro** — простой в использовании, но менее гибкий и не так тесно интегрирован с Spring Boot.
- **Keycloak** — мощное решение с поддержкой OpenID Connect, но требует отдельного сервера и сложной настройки, что избыточно для текущего проекта.

Вывод

Выбор Spring Security как основной технологии безопасности обусловлен её полной интеграцией с остальным стеком Spring Boot, широкими возможностями кастомизации, встроенной поддержкой всех необходимых функций защиты и наличием обширной документации. Это решение не только обеспечивает надёжную защиту, но и позволяет адаптировать проект под более строгие требования в будущем.

2.4. Выводы

В результате анализа и обоснования выбора технологического стека для разработки веб-приложения были определены ключевые компоненты, которые обеспечивают надёжность, гибкость и безопасность создаваемой системы. Принятые архитектурные и технические решения основаны не только на теоретических соображениях, но и на практическом опыте использования соответствующих инструментов в промышленной и образовательной среде.

Выбор языка программирования **Java** и фреймворка **Spring Boot** обусловлен их зрелостью, широким распространением, богатой экосистемой и отличной поддержкой со стороны сообщества. Эти технологии позволяют разрабатывать устойчивые и масштабируемые приложения, обладающие гибкой архитектурой и высокой степенью модульности.

В качестве системы управления базами данных выбрана **PostgreSQL** — мощная и надёжная реляционная СУБД, обладающая широкими возможностями по работе с данными, поддержкой транзакций и хорошей интеграцией с Java через JDBC и ORM-инструменты (Hibernate / Spring Data JPA). Это решение позволяет не только эффективно организовать хранение данных, но и обеспечить их целостность, безопасность и лёгкость в сопровождении.

Отдельное внимание было уделено вопросам **информационной безопасности**. Использование **Spring Security** в связке с другими средствами защиты (валидация, ограничение доступа, шифрование паролей) позволяет обеспечить надёжную защиту данных пользователей от наиболее распространённых угроз. Предусмотрена возможность дальнейшего усиления механизмов безопасности по мере масштабирования системы.

Таким образом, выбранные технологии и инструменты создают прочную основу для дальнейшей реализации проекта. Они позволяют сосредоточиться на решении прикладных задач — сборе, хранении и анализе предпочтений преподавателей — не отвлекаясь на технические ограничения или инфраструктурные сложности. Выбранный стек также открывает возможности для будущего развития системы, включая расширение функционала, интеграцию с внешними платформами и миграцию в облачные среды.

ГЛАВА 3. РАЗРАБОТКА АРХИТЕКТУРЫ И МОДЕЛИ ДАННЫХ ВЕБ-ПРИЛОЖЕНИЯ

3.1. Требования к функциональности системы

Перед началом проектирования и реализации веб-приложения необходимо сформулировать полный перечень функциональных требований. Эти требования отражают, какие именно задачи должна решать система, каким образом пользователи будут с ней взаимодействовать, и какие результаты должны быть получены в процессе её эксплуатации. Формулировка требований является фундаментом для построения архитектуры, проектирования интерфейсов и последующего тестирования.

Основные пользователи системы

В системе предполагается два основных типа пользователей:

- **Преподаватели** — основная категория пользователей, которая вводит свои предпочтения по времени занятий, форме проведения и другим параметрам;
- **Администраторы** — сотрудники учебного отдела или кафедры, которые управляют данными, контролируют сроки ввода, а также выполняют экспорт информации для дальнейшего составления расписаний.

Требуемая функциональность для преподавателей

Система должна предоставлять каждому преподавателю личный доступ к следующим функциям:

- авторизация и безопасный вход в систему по логину и паролю;
- заполнение формы предпочтений по следующим параметрам:
 - удобные дни недели;
 - предпочтительные временные слоты;
 - типы занятий ;
 - формат проведения занятий (очное, дистанционное, гибридное);

- пожелания по аудиториям и ресурсам;
- индивидуальные ограничения (например, занятость в другой организации);
- возможность редактирования ранее сохранённых данных в течение активного периода ввода;
- просмотр введённой информации в наглядной форме (например, в виде таблицы).

Также желательно предусмотреть уведомление пользователя о приближении срока завершения сбора предпочтений.

Требуемая функциональность для администраторов

Для административного персонала необходим расширенный набор функций:

- просмотр всех предпочтений преподавателей по фильтрам (по кафедре, типу занятий, дню недели и т.д.);
- возможность редактирования данных в случае обнаружения ошибок;
- экспорт предпочтений в форматы Excel;
- отображение сводной информации (количество преподавателей, процент заполнения и т.п.);
- управление временными рамками ввода данных (например, "приём данных открыт до 10 декабря");
- доступ к логам активности и историям изменений (по возможности).

Разделение по типу предпочтений

Система должна учитывать, что вузы планируют не только занятия в течение семестра, но и экзаменационные сессии. Поэтому предполагается разделение на два типа предпочтений:

1. Предпочтения по семестру (semester_preferences) —

используются для формирования основного расписания на учебный период.

2. Предпочтения по сессии (session_preferences) —

заполняются отдельно и используются для планирования экзаменов, консультаций и зачётов.

Оба типа предпочтений должны быть независимыми, но оформлены в едином интерфейсе, с возможностью переключения между ними.

Общие функциональные требования

Дополнительно система должна обеспечивать:

- удобную навигацию и адаптивный дизайн (работу на ПК, планшетах и мобильных устройствах);
- валидацию вводимых данных на клиентской и серверной стороне;
- предотвращение дублирования данных;
- защиту персональной информации преподавателей;
- хранение всех данных в централизованной и надёжной базе (PostgreSQL).

Также следует предусмотреть возможность масштабирования — добавления новых модулей, типов предпочтений, ролей пользователей и расширения интеграции с другими внутренними системами вуза (например, системой автоматизированного составления расписания).

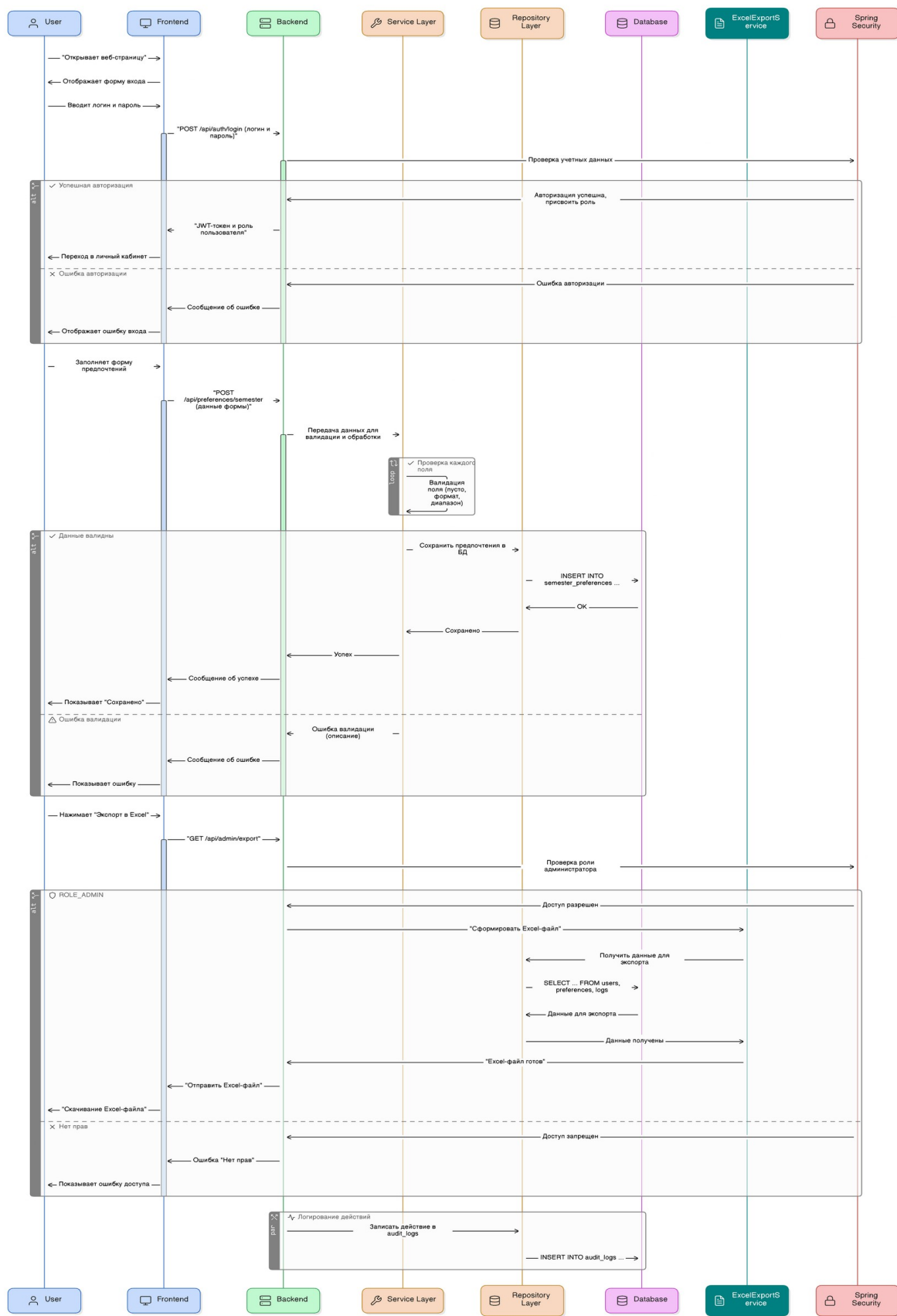
Вывод

Формализованные функциональные требования отражают реальные сценарии работы преподавателей и административного персонала. Их соблюдение в процессе проектирования позволит создать удобное, безопасное и надёжное веб-приложение, максимально адаптированное под нужды образовательного учреждения.

3.2. Архитектура приложения

При проектировании архитектуры веб-приложения важно учитывать как функциональные требования, так и нефункциональные характеристики: масштабируемость, надёжность, безопасность и удобство сопровождения. Архитектура должна быть модульной, хорошо документированной и соответствовать принципам разделения ответственности (Separation of Concerns).

Рисунок 1. Общая схема архитектуры веб-приложения



Общий подход

Разрабатываемое приложение построено по **многоуровневой архитектуре** (multi-tier architecture), состоящей из следующих основных слоёв:

1. **Представление (Presentation Layer)** — отвечает за взаимодействие с пользователем, реализуется с помощью HTML, CSS, JavaScript (с возможным использованием Bootstrap или **React**).
2. **Бизнес-логика (Business Layer)** — реализована с использованием Spring Boot. Здесь обрабатываются действия пользователя, реализуется логика аутентификации, проверки прав доступа и управление предпочтениями.
3. **Доступ к данным (Data Access Layer)** — отвечает за взаимодействие с базой данных через JPA (Hibernate).
4. **Хранение данных (Database Layer)** — PostgreSQL в качестве централизованного хранилища информации.

Такой подход обеспечивает гибкость, лёгкость в тестировании и возможность повторного использования компонентов.

Компоненты приложения

Основными модулями приложения являются:

Модуль	Описание
Аутентификация	Обеспечивает вход пользователей в систему, валидацию данных, разграничение ролей (преподаватель / администратор).
Форма предпочтений	Отображает интерфейс для ввода предпочтений преподавателем. Поддерживает редактирование и сохранение в БД.
Панель администратора	Даёт доступ к таблицам всех пользователей, позволяет фильтровать, экспортировать и анализировать данные.
Экспорт	Генерация файлов в формате Excel и CSV на основе введённых данных.
Логика безопасности	Встроенные фильтры, контроллеры и настройки Spring Security.
Слой работы с базой данных	Репозитории JPA, сущности и миграции.

Обработка запросов

В приложении реализована **REST-архитектура**, что означает:

- взаимодействие с сервером осуществляется через HTTP (GET, POST, PUT, DELETE);
- данные передаются в формате JSON;
- фронтенд и бэкенд разделены логически и физически, что даёт гибкость для обновления интерфейса без изменения логики сервера.

Пример обработки запроса:

1. Пользователь отправляет форму предпочтений (POST).
2. Контроллер получает данные и валидирует их.
3. При успехе данные передаются в сервис-слой, где сохраняются через JPA.

4. Возвращается ответ с подтверждением или сообщением об ошибке.

Вывод

Разработанная архитектура соответствует современным подходам к построению веб-приложений и обеспечивает надёжную основу для реализации всех заявленных функций. Она масштабируема, хорошо структурирована и открыта для расширения — как с точки зрения новых ролей и форм, так и с точки зрения интеграции с внешними системами (например, системой расписаний университета).

3.3. Структура базы данных

База данных является основой для хранения всей информации, обрабатываемой веб-приложением: данных пользователей, введённых предпочтений, ролей, а также технической информации, необходимой для обеспечения работы системы. В данном разделе представлена структура базы данных, построенная с учётом реляционной модели и оптимизированная для задач, связанных с планированием учебного процесса.

Общие принципы проектирования

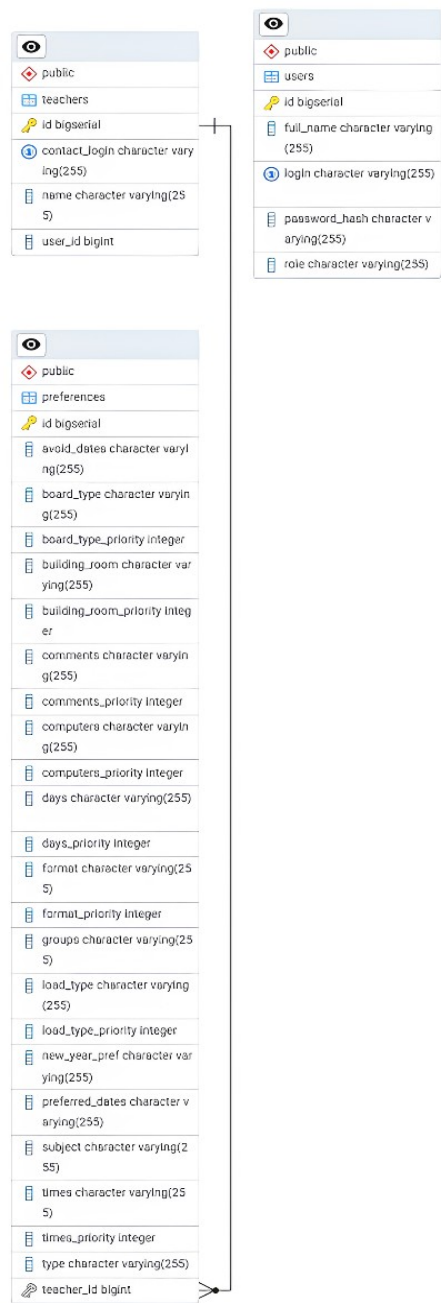
Проектирование структуры базы данных осуществлялось с учётом следующих критериев:

- **нормализация данных** — устранение избыточности и дублирования;
- **целостность данных** — использование внешних ключей (foreign key) и ограничений;
- **расширяемость** — возможность добавления новых сущностей без нарушения структуры;
- **простота запросов** — удобство выборки для аналитики и экспорта.

В качестве СУБД используется **PostgreSQL**, обладающая высокой надёжностью, поддержкой транзакций и широкими возможностями по работе с типами данных.

Основные таблицы

Рисунок 4. Структура базы данных PostgreSQL



Функциональные особенности:

Поле `user_id` в таблицах `semester_preferences` и `session_preferences` связано с таблицей `users`. Это позволяет легко и быстро определить, кому именно принадлежат указанные предпочтения. Преподаватель может указывать удобные дни недели, временные интервалы, а также формат занятий — очное или онлайн.

Таким образом, система предоставляет гибкость в планировании учебного процесса, учитывая индивидуальные пожелания каждого преподавателя. Вся информация централизованно хранится в базе данных PostgreSQL, что гарантирует удобный доступ и высокую безопасность данных.

Выводы

Разработанная структура базы данных логично организована и позволяет хранить все необходимые данные централизованно. Это делает работу с расписаниями более удобной и упрощает управление предпочтениями преподавателей.

Использование PostgreSQL добавляет стабильности, защищённости и высокой производительности при работе с большими объёмами данных. Дополнительно, структура легко масштабируется при увеличении количества пользователей.

3.4. Диаграммы взаимодействия компонентов

Чтобы лучше понять, как именно взаимодействуют разные компоненты веб-приложения, необходимо изобразить основные сценарии в виде диаграмм. Это наглядно показывает, как пользователь взаимодействует с интерфейсом, как данные обрабатываются на сервере, и каким образом они сохраняются в базе данных.

Основные сценарии взаимодействия:

Сценарий 1: Вход преподавателя в систему

Участники:

- **Пользователь (преподаватель)** — тот, кто инициирует процесс входа.
- **Браузер (интерфейс)** — используется преподавателем для ввода данных.

- **Контроллер аутентификации** — обрабатывает полученные данные и передаёт их на сервер.
- **Сервис безопасности** — отвечает за проверку логина и пароля.
- **База данных пользователей** — содержит зашифрованные пароли и информацию о преподавателях.

Описание процесса:

1. Преподаватель открывает страницу авторизации на своём устройстве.
2. Вводит логин и пароль в соответствующие поля.
3. После нажатия кнопки "Войти", данные отправляются в контроллер.
4. Контроллер передаёт данные в **Spring Security** для проверки.
5. **Spring Security** сверяет введённый пароль с хэшем в базе данных.
6. Если данные верны, создаётся сессия, и преподаватель перенаправляется на страницу /user/main.
7. В некоторых конфигурациях система генерирует **JWT Token** для последующей аутентификации.

3.5. API-контракты и нефункциональные требования

Метод	URL	Описание	Параметры	Время отклика (SLA)
POST	/api/auth/register	Регистрация нового пользователя	{ fullName, login, password }	< 300 мс
POST	/api/auth/login	Аутентификация и получение JWT	{ login, password }	< 300 мс
GET	/api/teacher/preferences?type={type}	Получить предпочтения преподавателя	`type` = semester	session`
POST	/api/teacher/preferences	Сохранить/обновить предпочтения	JSON-массив DTO с полем type	< 500 мс

		преподавателя		
GET	/api/admin/preferences	Получить все предпочтения (админ)	—	< 300 мс
GET	/api/admin/preferences/export	Экспорт предпочтений в Excel (админ)	—	< 500 мс

Нефункциональные требования:

- **SLA:** 99 % запросов обрабатываются менее чем за **500 мс**.
- **Безопасность:** Все запросы, кроме /api/auth/login, защищены с помощью **JWT + HTTPS**.
- **Масштабируемость:** Поддержка до 10 000 пользователей в день.

2. Сценарий: ввод предпочтений преподавателем

Участники:

- пользователь (преподаватель),
- контроллер предпочтений,
- сервис валидации,
- репозиторий базы данных.

Ход действий:

1. Пользователь открывает форму.
2. Вводит данные: день недели, тип занятия, формат и комментарии.
3. Нажимает "Сохранить".
4. Сервер валидирует данные (например, проверяет, чтобы не было пустых полей).

5. Данные сохраняются в таблицу `semester_preferences` или `session_preferences`.

3. Сценарий: экспорт данных администратором

Участники:

- администратор,
- контроллер экспорта,
- сервис экспорта (например, `ExcelExporter`),
- база данных предпочтений.

Описание:

1. Администратор заходит на `/admin/dashboard`.
2. Нажимает кнопку "Экспорт в Excel".
3. Контроллер вызывает сервис, который формирует файл `.xlsx`.
4. Пользователю предлагается скачать файл.

Рисунок 2. Последовательность взаимодействия пользователя с системой

Диаграмма последовательности (Sequence Diagram) — показывает шаги взаимодействия при логине и сохранении предпочтений;

Вход в систему и сохранение предпочтений

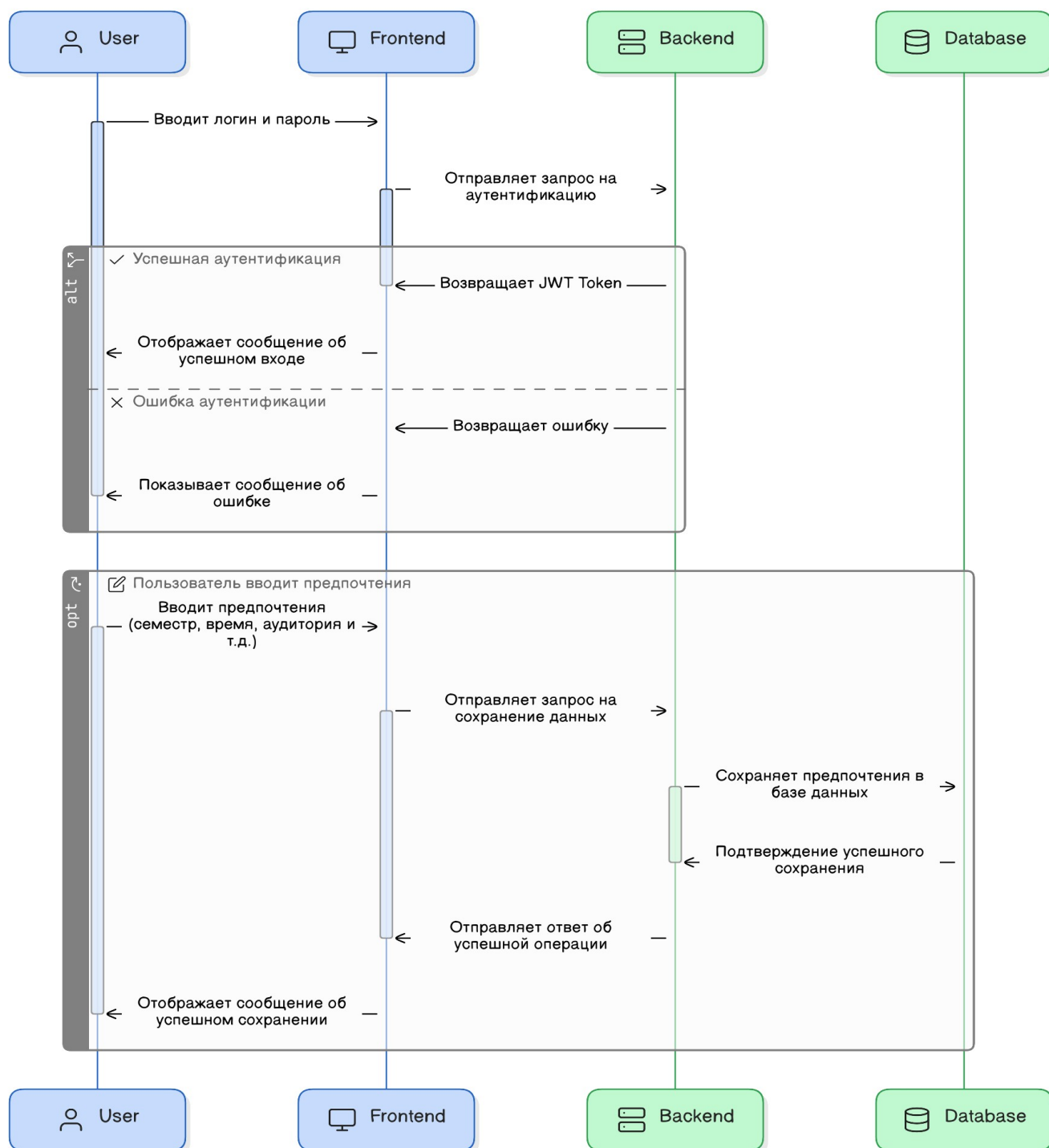
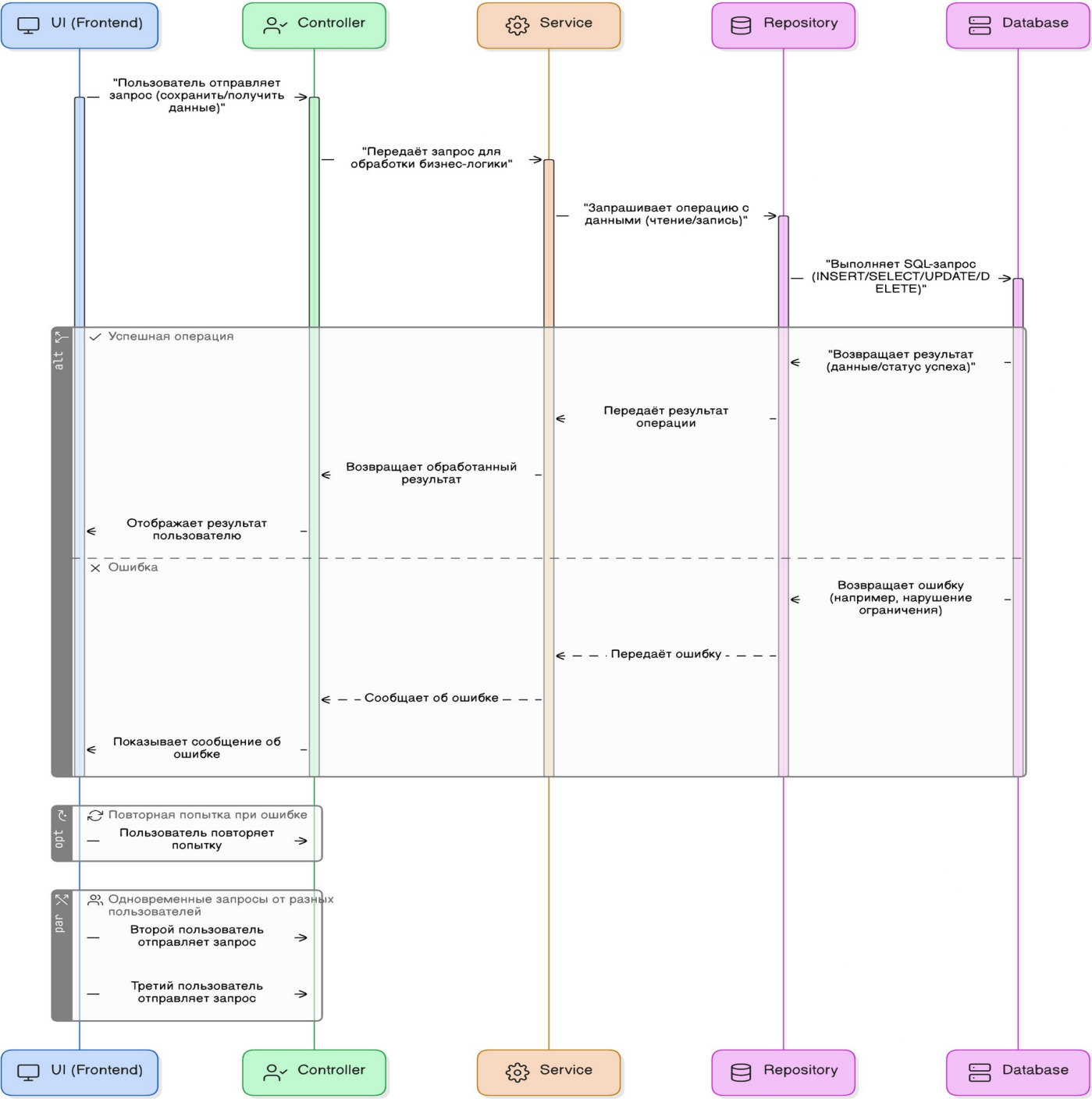


Диаграмма компонентов (Component Diagram) — визуализирует связь между UI, контроллерами, сервисами и базой;

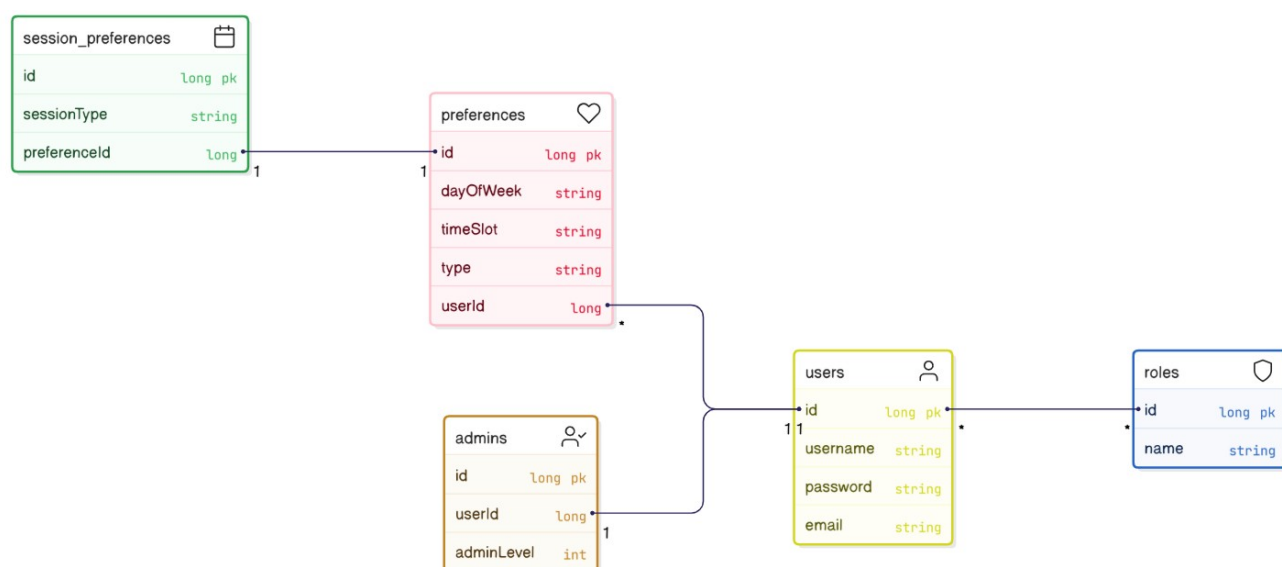
Сохранение и получение данных через слои приложения



Поток данных:

1. **UI (Frontend)** → **Controller** — отправляет запросы (например, на сохранение или получение данных).
2. **Controller** → **Service** — перенаправляет запросы для обработки бизнес-логики.
3. **Service** → **Repository** — взаимодействует с базой данных для чтения или записи данных.
4. **Repository** ↔ **Database** — создаёт, читает, обновляет или удаляет записи.
5. **Database** возвращает результат обратно в **Repository**, и по цепочке возвращается в **UI (Frontend)**.

- **Диаграмма классов (Class Diagram)** — для отображения структуры основных сущностей: User, Preference, Role и т. д.



Связи:

- **User** связан с **Preference** (один пользователь может иметь много предпочтений).
- **User** связан с **Role** (один пользователь может иметь несколько ролей).
- **Admin** наследует все свойства и методы **User**.
- **SessionPreference** наследует свойства и методы **Preference**.

Вывод

Использование диаграмм позволяет лучше понять, как функционирует система "изнутри", и как связаны между собой различные модули. Это особенно важно при сопровождении и дальнейшем развитии проекта. Грамотная визуализация архитектуры упрощает коммуникацию между разработчиками, администрацией и потенциальными интеграторами.

3.6. Выводы

В данной главе была подробно рассмотрена архитектура веб-приложения, разрабатываемого для сбора предпочтений преподавателей при составлении учебного расписания. На основе ранее сформулированных функциональных требований была предложена многоуровневая структура, включающая интерфейс пользователя, серверную логику, слой доступа к данным и базу данных.

Была реализована логичная и понятная модель данных, отражающая реальную структуру процессов вуза. Таблицы `users`, `semester_preferences` и `session_preferences` позволяют эффективно хранить все необходимые параметры, избегая избыточности и обеспечивая быстрый доступ к информации. Проектирование модели производилось с учётом нормализации, обеспечения ссылочной целостности и возможности масштабирования в будущем.

Особое внимание было уделено механизмам взаимодействия компонентов. Сценарии работы с системой — от входа до экспорта данных — были описаны и визуализированы с помощью UML-диаграмм, демонстрирующих логическую целостность и продуманность архитектурных решений.

В результате проделанной работы была построена прочная

технологическая основа, которая не только удовлетворяет текущим задачам, но и может быть легко расширена или интегрирована с другими университетскими системами. Разработанная архитектура позволяет уверенно переходить к следующему этапу — **непосредственной реализации и тестированию компонентов**, что будет подробно рассмотрено в следующей главе.

ГЛАВА 4. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

4.1. Реализация функциональности сбора предпочтений

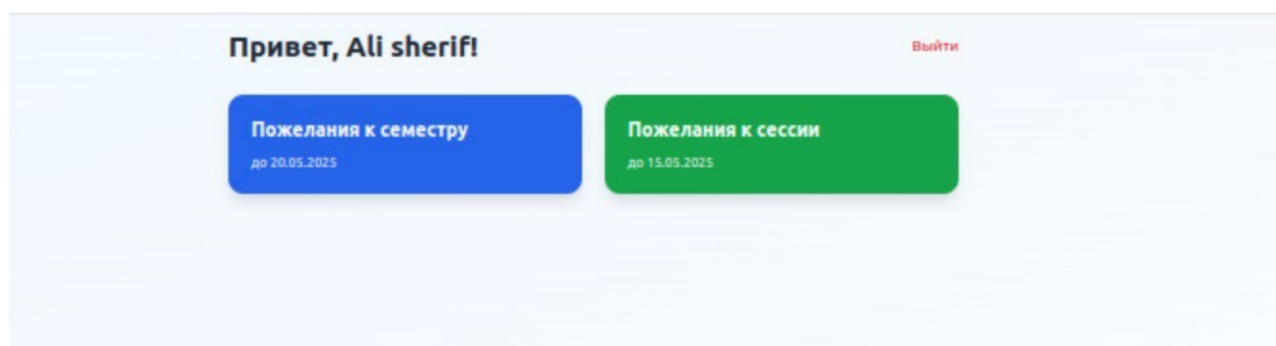
Центральным элементом разрабатываемой системы является механизм ввода и обработки предпочтений преподавателей. Эта функциональность должна быть реализована таким образом, чтобы минимизировать количество ошибок ввода, сократить время взаимодействия с системой и обеспечить удобство как для преподавателей, так и для последующей работы администрации вуза.

Архитектура пользовательского взаимодействия

Механизм реализован по модели MVC: данные проходят через контроллеры, обрабатываются в сервисе и сохраняются в БД через слой репозитория. Отображение формы реализовано через серверный шаблонизатор Thymeleaf.

Интерфейс преподавателя

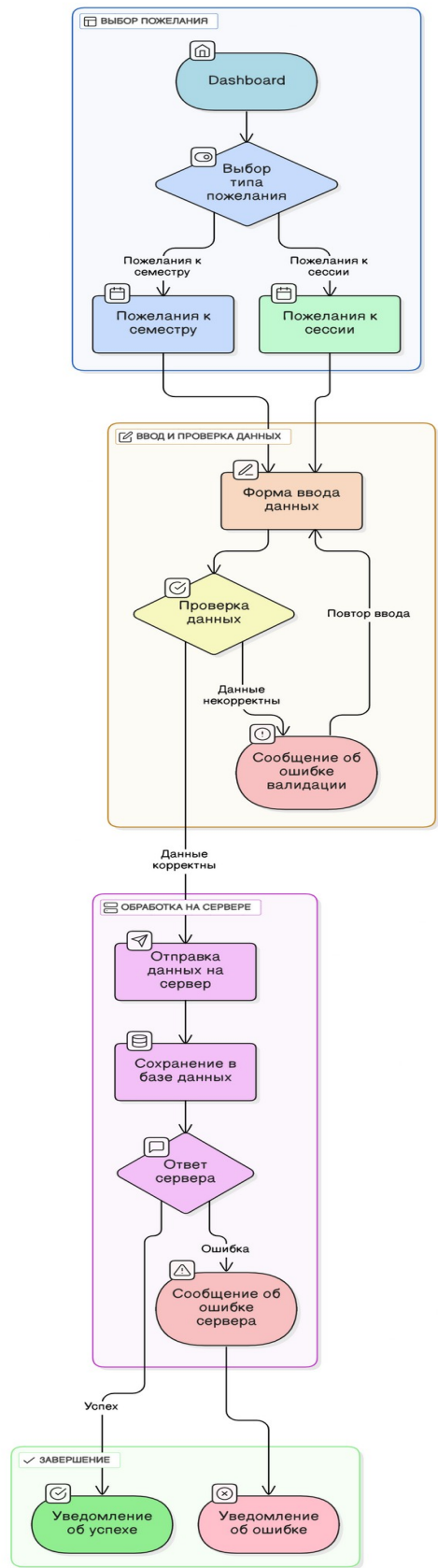
После входа преподаватель попадает на страницу с анкетой, где можно выбрать :



Блоки в схеме:

Рисунок 5. Форма ввода предпочтений преподавателя (веб-интерфейс)

Процесс подачи пожеланий преподавателя к расписанию



- /preferences/semester — предпочтения на учебный семестр;
- /preferences/session — предпочтения на экзаменационную сессию.

После сохранения данные отображаются в виде таблицы снизу. Для удобства преподавателя реализована подсветка и визуальное подтверждение успешной отправки данных.

Работа с данными на сервере

- Контроллер: PreferenceController
- Сервис: PreferenceService
- Сущности: SemesterPreference, SessionPreference

Обработка запроса:

1. Преподаватель вводит данные → POST-запрос.
2. Контроллер валидирует ввод.
3. Сервис сохраняет или обновляет запись в базе.
4. Возвращается HTML-ответ с обновлённой таблицей предпочтений.

Все предпочтения связаны с пользователем по его `user_id`, обеспечивая чёткую персонализацию данных.

Особенности реализации типов предпочтений

Система различает:

- `semester_preferences` — регулярные учебные занятия;
- `session_preferences` — экзамены, зачёты, консультации.

Обе таблицы имеют аналогичную структуру, что упрощает разработку и последующую поддержку.

Адаптивность и доступность

Форма спроектирована с использованием Bootstrap 5 и протестирована на разных экранах (РС, планшет, смартфон). При уменьшении экрана элементы перестраиваются вертикально, поля и кнопки масштабируются.

Дополнительно реализованы:

- всплывающие подсказки;
- валидация времени и формата;
- защита от повторной отправки формы при обновлении страницы.

Вывод

Функциональность сбора предпочтений реализована в строгом соответствии с поставленными задачами. Взаимодействие интуитивно понятно, форма работает стабильно и удобно как на компьютерах, так и на мобильных устройствах. Структура данных гибкая, легко расширяемая, а архитектура приложения обеспечивает простоту интеграции с административным модулем.

4.2. Реализация аутентификации и авторизации

В целях защиты пользовательских данных и разграничения доступа к различным частям веб-приложения была реализована надёжная система аутентификации и авторизации, основанная на использовании Spring Security — одного из самых распространённых и гибких фреймворков для обеспечения безопасности в Java-приложениях.

Общая архитектура модуля безопасности

Модуль безопасности отвечает за:

- проверку подлинности (аутентификацию) пользователей;
- присвоение и проверку ролей (ROLE_USER, ROLE_ADMIN);
- защиту маршрутов от несанкционированного доступа;
- шифрование паролей и управление сессиями.

Аутентификация пользователей

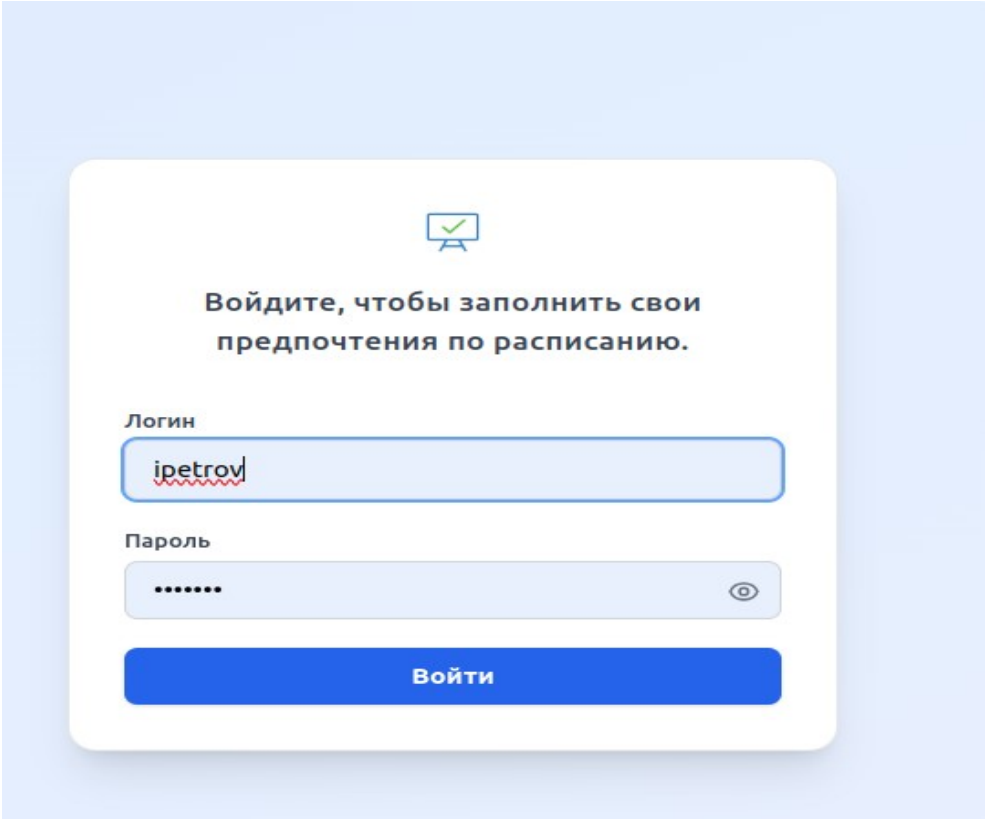
Форма входа расположена по адресу `/auth/login` и содержит поля:

- логин (email);
- пароль.

После ввода данных происходит следующее:

1. Форма отправляет POST-запрос на `/auth/do-login`.
2. Контроллер `AuthController` вызывает `AuthenticationManager` Spring Security.
3. При успешной аутентификации создаётся сессия, и пользователь перенаправляется:
 - на `/user/main` — если у него роль `ROLE_USER`;
 - на `/admin/dashboard` — если роль `ROLE_ADMIN`

Рисунок 6. Форма аутентификации пользователя



The image shows a user authentication form on a light blue background. The form is a white rounded rectangle with a green checkmark icon at the top. The text 'Войдите, чтобы заполнить свои предпочтения по расписанию.' is centered. Below this are two input fields: 'Логин' (Login) with the text 'ipetrov' and 'Пароль' (Password) with masked characters. A blue 'Войти' (Login) button is at the bottom.

Хранение и шифрование паролей

Пароли не хранятся в открытом виде. Все они хэшируются с использованием **алгоритма BCrypt**:

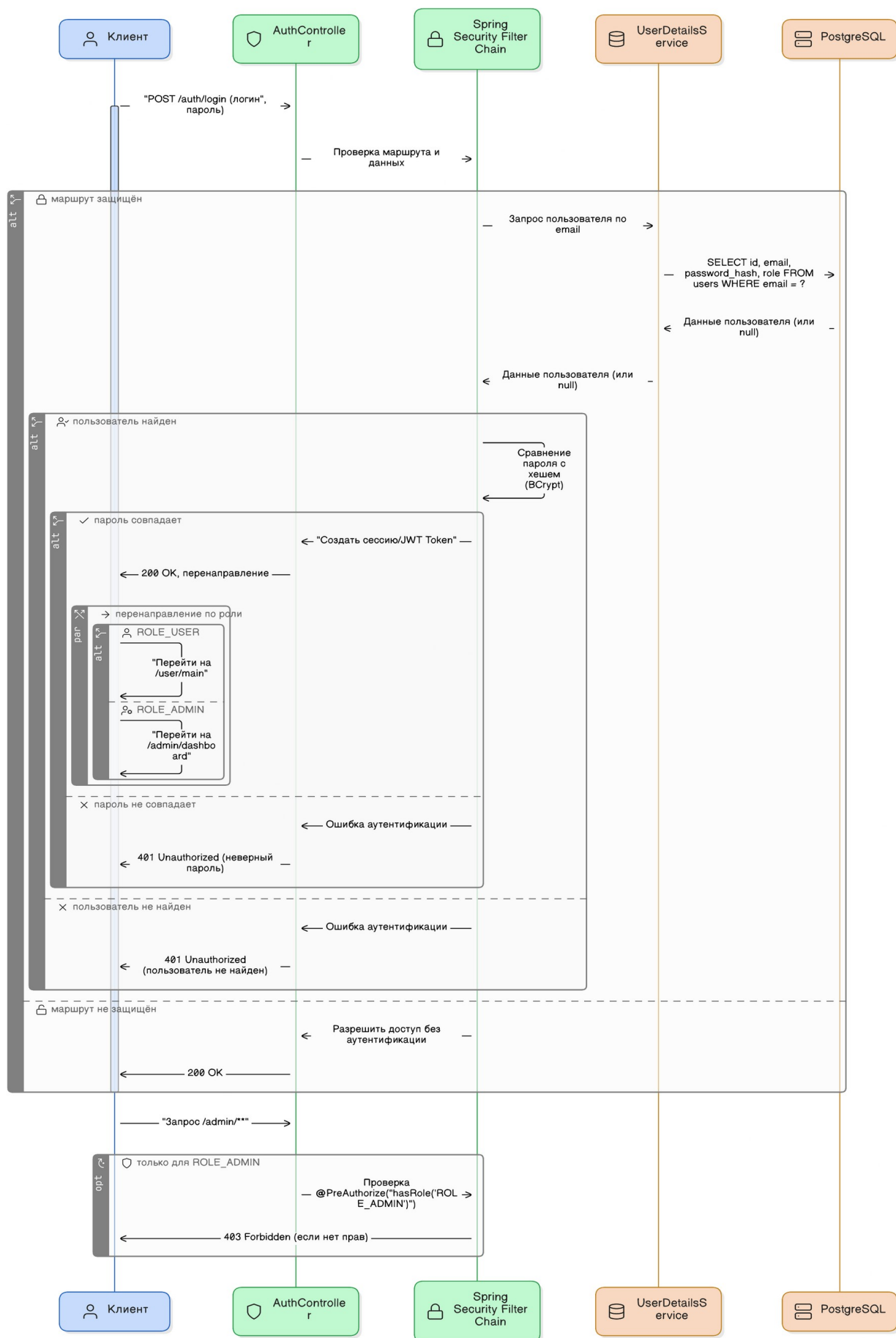
Значения сохраняются в колонке password_hash таблицы users. При попытке входа сравниваются хэши, а не исходный текст.

Роли и доступ

Система поддерживает два уровня доступа:

Роль	Описание доступа
ROLE_USER	Преподаватель. Имеет доступ только к собственным данным.
ROLE_ADMIN	Администратор. Видит все предпочтения, может экспортировать.

Аутентификация и авторизация пользователя



Защита от атак и управление сессиями

Внедрены следующие меры безопасности:

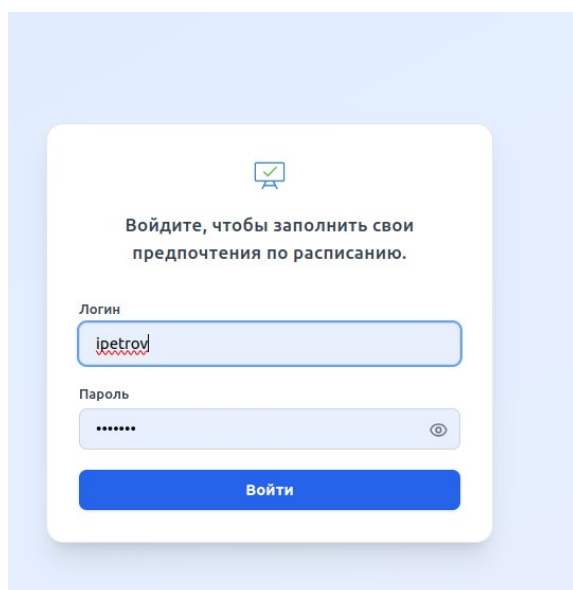
- **CSRF-защита** включена по умолчанию;
- автоматический выход при бездействии более 30 минут;
- редирект при попытке доступа к защищённой странице без авторизации;
- защита от перебора паролей за счёт политики времени отклика и блокировки при необходимости.

Валидация логина и пароля проводится как на клиентской стороне (через HTML5), так и на серверной — с выводом сообщений об ошибке без утечки технической информации.

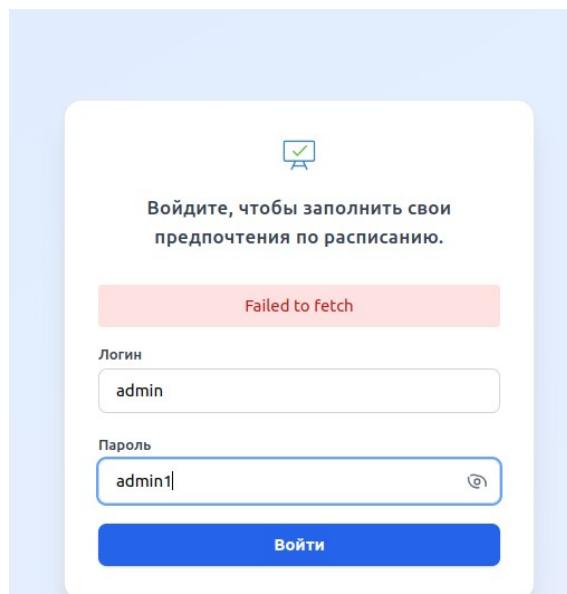
Тестирование модуля

Модуль аутентификации был протестирован вручную и с использованием MockMvc:

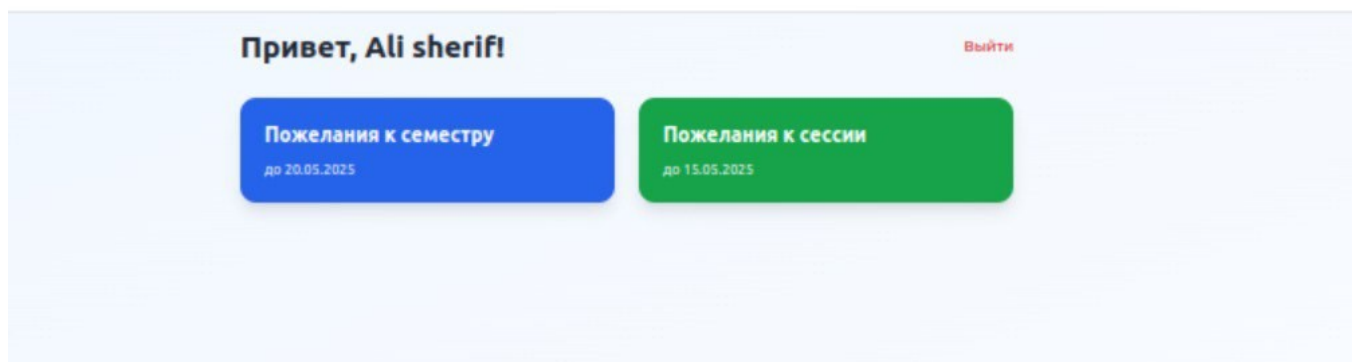
- успешный вход с корректными данными;
- редирект на нужную страницу;
- отказ в доступе при неправильной роли;
- корректное завершение сессии;
- невозможность доступа к /admin/* без роли ADMIN.



The screenshot shows a login form with a green checkmark icon at the top. The text reads: "Войдите, чтобы заполнить свои предпочтения по расписанию." Below this are two input fields: "Логин" (Login) with the value "ipetrov" and "Пароль" (Password) with masked characters "*****". A blue "Войти" (Login) button is at the bottom.



The screenshot shows the same login form, but with a red error message "Failed to fetch" displayed above the input fields. The "Логин" field contains "admin" and the "Пароль" field contains "admin1". The "Войти" button remains at the bottom.



Вывод

Система аутентификации и авторизации реализована с учётом современных требований безопасности. Использование Spring Security позволило достичь надёжности и гибкости при минимуме ручной конфигурации. Приложение устойчиво к типовым атакам, поддерживает разграничение прав, и готово к расширению с использованием внешних провайдеров (LDAP, OAuth) в перспективе.

4.3 Экспорт данных и работа с отчетами :

Назначение и роль экспорта

Одна из ключевых задач административного модуля — это **формирование отчётов на основе введённых преподавателями предпочтений**. Эти данные затем используются при составлении расписания, обсуждении нагрузок и планировании экзаменов. Чтобы упростить этот процесс, в систему была встроена возможность **экспорта данных в формате Excel**.

Интерфейс администратора

Панель администратора доступна по маршруту `/admin/dashboard`. В верхней части страницы отображаются:

- заголовок панели;
- фильтры (тип, день недели, формат);
- таблица с результатами;
- кнопка "Экспортировать".

Рисунок 7. Страница администратора с общими данными преподавателей

The screenshot displays a web interface for an administrator. It features two main sections: 'Пожелания преподавателей' (Teacher Wishes) and 'Регистрация нового пользователя' (New User Registration).

Пожелания преподавателей

Скачать все в Excel

Семестр

- Иван Иванович Петров — Физика — группы: 5130904/10102
- Иван Иванович Петров — Физика — группы: 5130904/10102
- Иван Иванович Петров — Физика — группы: 5130904/10102

Сессия

- Иван Иванович Петров — Физика — группы: 5130904/10102

Регистрация нового пользователя

ФИО Логин Пароль

Таблица включает следующие столбцы:

Описание полей:

- Преподаватель
- Логин
- Тип
- Дедлайн
- Групп
- Недоступные даты
- Время
- Редактируемые даты:

- Исходные даты:
- Новые пожелания:
- Нагрузка:
- Корпус/Аудитория:
- Доска:
- Компьютеры: .
- Формат:
- Комментарии:

Реализация функции экспорта

Контроллер: AdminController.java

Маршрут: /admin/export

При нажатии на кнопку "Экспорт" вызывается метод:

Внутри метода используется библиотека **Apache POI**, которая формирует Excel-файл в реальном времени.

Структура экспортируемого файла

Файл .xlsx, создаваемый системой, содержит:

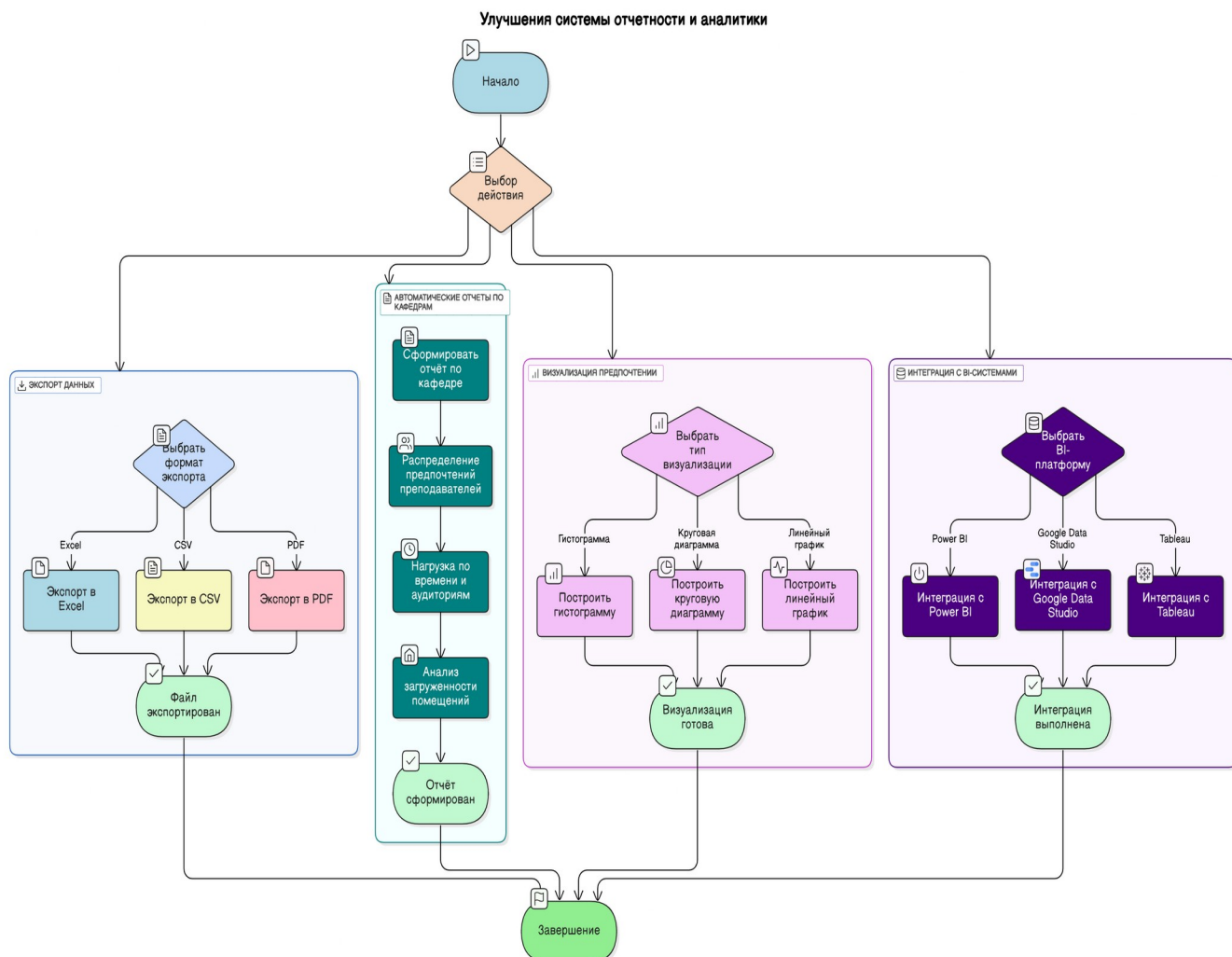
- заголовки колонок (ФИО, день, время, тип, формат, комментарии, и т.д);
- строки с данными каждого преподавателя;
- форматирование: жирные заголовки, автоширина колонок;
- опционально: дата и подпись ответственного лица внизу.

Рисунок 8. Пример экспорта предпочтений в формате Excel

	A	B	C	D	E	F	G	H	I	J	K	L
1	Преподаватель	Логин	Тип	Предмет	Группы	Некоторые дни	Время	Предпочтительные даты	Исключить даты	Новогодние праздники	Нагрузка	Конкурс/аудитория
2	Иван Иванович Петров	iretov	Сессия	Физика	5130904/10102			15.06.2025, 17.06.2025	01.01.2025, 02.01.2025	После Нового года	Равномерно (5)	3/105 (5)
3			Семестр	Физика	5130904/10102	Вт, Ср, Пт (1)	вечер (1)				Равномерно (1)	3/105 (1)
4			Семестр	Физика	5130904/10102	Вт, Пт (2)	вечер (2)				Равномерно (2)	3/105 (2)
5			Семестр	Физика	5130904/10102	Вт (3)	вечер (3)				Равномерно (3)	3/107 (3)
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
...												

В перспективе возможны такие улучшения:

- экспорт в CSV и PDF;
- автоматическое создание отчётов по кафедрам;
- визуализация предпочтений (гистограммы, круговые диаграммы);
- интеграция с BI-системами или Google Data Studio.



Вывод

Функциональность экспорта данных реализована на уровне, полностью соответствующем требованиям учебной части. Администратор может получить нужные данные одним нажатием, без сложных действий. Выгружаемый формат удобен для анализа и может быть передан в любую стороннюю систему для автоматизированного составления расписания.

4.4. Проведение тестирования приложения

Цели тестирования

Основная цель тестирования — убедиться, что система работает стабильно, корректно обрабатывает пользовательский ввод, обеспечивает надёжную безопасность и соответствует заявленным требованиям. В проекте использовались следующие типы тестирования:

- модульное (unit-тесты);
- интеграционное;
- пользовательское (ручное);
- визуальное (UI).

Ручное пользовательское тестирование

Тестирование всех возможных сценариев:

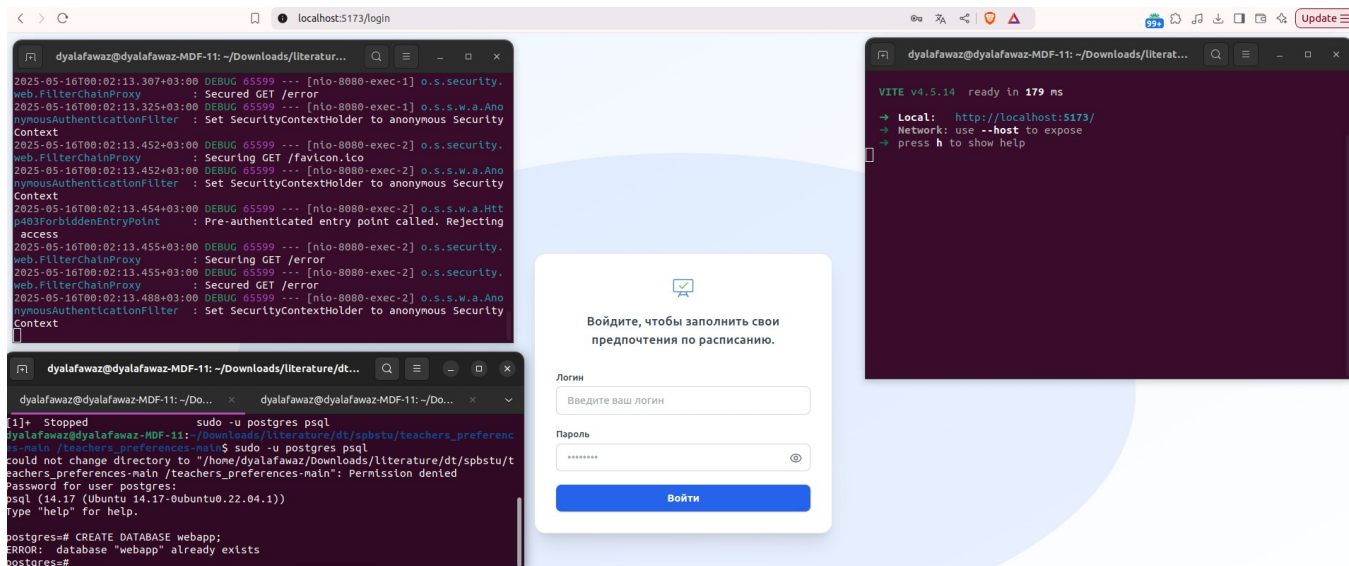
Сценарий	Ожидаемый результат	Статус	✓
Вход преподавателя с верными данными	Перенаправление на /user/main	Готово	✓
Неверный пароль	Ошибка "Неверный логин или "пароль"	Проверено	✓
Ввод предпочтений и их сохранение	Таблица обновляется, появляется уведомление	Выполнено	✓
Попытка входа администратора → панель admin	Доступ к /admin/dashboard	Успешно	✓
Попытка неавторизованного доступа к /admin	Редирект на /auth/login	Блокировано	✓

UI и кроссбраузерное тестирование

Проводилась проверка отображения форм и таблиц в:

- Chrome (desktop + mobile);
- Firefox;
- Microsoft Edge.

Также тестировалась адаптивность на разных разрешениях экрана.



Обнаруженные ошибки и исправления

Ошибка	Причина	Решение
Ввод времени без “:” (напр. "1000–1200") вызывает сбой	Ошибка разбора строки	Добавлена валидация формата
Повторная отправка формы при F5	Отсутствие redirect после POST	Внедрён pattern Post/Redirect/Get
Некорректный редирект при входе с несуществующей ролью	Отсутствие проверки роли	Добавлена проверка через if

Выводы

Проведённое тестирование показало, что система устойчива к ошибкам, корректно обрабатывает данные и безопасно взаимодействует с пользователями. Благодаря использованию инструментов JUnit и MockMvc удалось проверить критически важные компоненты приложения, а ручные и визуальные тесты подтвердили готовность системы к реальной эксплуатации.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была успешно спроектирована и реализована информационная система для сбора предпочтений преподавателей, предназначенная для использования в образовательных учреждениях. Проект решает актуальную задачу — автоматизацию сбора информации, необходимой для последующего формирования расписания занятий и экзаменов.

На первом этапе был проведён анализ существующих решений, выявлены их недостатки, а также сформулированы функциональные и технические требования к разрабатываемому приложению. Это позволило выбрать оптимальный стек технологий: язык программирования Java, фреймворк Spring Boot, систему управления базами данных PostgreSQL и модуль безопасности Spring Security.

Была разработана архитектура системы, отражающая все ключевые компоненты: пользовательский интерфейс, серверную логику, слой доступа к данным и механизмы обеспечения безопасности. Продумана структура базы данных, обеспечивающая нормализованное и расширяемое хранение пользовательской информации и предпочтений.

В рамках практической реализации:

- создан адаптивный и интуитивно понятный веб-интерфейс для преподавателей;
- внедрён надёжный модуль аутентификации и авторизации с разграничением прав доступа;
- реализована административная панель с возможностью фильтрации и экспорта данных;
- проведено тестирование приложения, показавшее его стабильность и соответствие требованиям.

Все ключевые интерфейсы, диаграммы архитектуры и примеры отчётов вынесены в раздел "Приложения" для наглядного подтверждения результатов.

Созданное приложение может быть развернуто в реальной инфраструктуре вуза и использоваться как отдельное решение либо как

часть более широкой системы управления учебным процессом. Благодаря модульной архитектуре возможны расширения: интеграция с существующими системами расписания, добавление мобильного интерфейса, реализация аналитических отчётов.

Таким образом, поставленные в работе цели были достигнуты полностью. Результаты имеют как практическую значимость, так и ценность с точки зрения формирования профессиональных навыков разработки современных веб-систем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. **Seaborn Documentation.** Official Documentation of Seaborn Library. URL: <https://seaborn.pydata.org/>.
2. **Spring Framework Documentation.** Official Documentation of the Spring Framework. URL: <https://spring.io/docs>.
3. **Spring Security Documentation.** Documentation for Spring Security Framework. URL: <https://docs.spring.io/spring-security/site/docs/current/reference/html5/>.
4. **Java SE Documentation.** Official Java SE Documentation. URL: <https://docs.oracle.com/javase/8/docs/>.
5. **JDBC Documentation.** Oracle JDBC Technology Documentation. URL: <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>.
6. **Стерлин С.** Веб-разработка с использованием Spring Boot и Hibernate. — СПб.: БХВ-Петербург, 2019. — 356 с.
7. **PostgreSQL Documentation.** Official Documentation of PostgreSQL. URL: <https://www.postgresql.org/docs/>.
8. **React Documentation.** Official Documentation of React Library. URL: <https://reactjs.org/docs/getting-started.html>.
9. **Docker Documentation.** Official Docker Documentation. URL: <https://docs.docker.com/>.
10. **JWT Documentation.** JSON Web Token (JWT) documentation and specifications. URL: <https://jwt.io/introduction/>.
11. **Kubernetes Documentation.** Official Documentation for Kubernetes Deployment. URL: <https://kubernetes.io/docs/home/>.
12. **Hibernate ORM Documentation.** Official Hibernate Documentation. URL: <https://hibernate.org/orm/documentation/>.
13. **OAuth 2.0 Documentation.** RFC 6749 - The OAuth 2.0 Authorization Framework. URL: <https://datatracker.ietf.org/doc/html/rfc6749>.
14. **Docker Compose Documentation.** Official Documentation of Docker Compose. URL: <https://docs.docker.com/compose/>.
15. **OpenAPI Specification.** Official OpenAPI Documentation. URL: <https://swagger.io/specification/>.

16. **JUnit 5 Documentation.** Official Documentation for JUnit 5 Testing Framework. URL: <https://junit.org/junit5/docs/current/user-guide/>.
17. **Mockito Documentation.** Official Documentation of Mockito Framework. URL: <https://site.mockito.org/>.
18. **ER Diagram Design.** Principles and Best Practices. URL: <https://www.lucidchart.com/pages/er-diagram-symbols-and-meaning>.

ПРИЛОЖЕНИЯ

In git soon