МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра программирования и информационных технологий

Курсовая работа

6 семестр

Разработка модуля «Бронирование аудитории» для балльно-рейтинговой системы факультета

Направление 09.03.04 Программная инженерия

Информационные системы и сетевые технологии

Зав. кафедрой_____ С.Д. Махортов, д.ф. – м.н., доцент 09. 06.2025

Обучающийся_____ П.И. Мигачев, 3 курс, д/о 09. 06.2025

Руководитель _____А.И. Чекмарёв, ст. преподаватель 09. 06.2025

Воронеж 2025

СОДЕРЖАНИЕ

Определения, обозначения и сокращения

— API (Application Programming Interface) — интерфейс прикладного программирования; набор определённых правил, позволяющих разным программам взаимодействовать между собой.

— Backend — серверная часть приложения, обрабатывающая бизнес-логику и работу с базой данных (в рамках данной работы будет добавлен на следующем этапе).

— Booking — бронирование. В контексте данной работы — процесс закрепления аудитории за определённым пользователем в указанное время.

— Frontend — клиентская часть веб-приложения, с которой взаимодействует пользователь.

— JSON — JavaScript Object Notation, текстовый формат обмена данными.

— LocalStorage — механизм хранения данных в браузере, позволяющий сохранять информацию между сессиями пользователя.

— React — JavaScript-библиотека для построения пользовательских интерфейсов.

— REST API — архитектурный стиль построения API, использующий стандартные HTTP-методы (GET, POST, PUT, DELETE) для взаимодействия между клиентом и сервером.

— SaaS (Software as a Service) — модель предоставления программного обеспечения как услуги по подписке через интернет, без установки на устройство пользователя.

— SheetJS (xlsx) — JavaScript-библиотека для работы с Excel-файлами, поддерживающая чтение, редактирование и создание документов.

— SQLite — встраиваемая система управления базами данных, не требующая отдельного сервера и хранящая данные в одном файле.

— UI (User Interface) — пользовательский интерфейс; элементы, с которыми взаимодействует пользователь при работе с программой.

— useEffect — хук библиотеки React, позволяющий выполнять побочные эффекты в функциональных компонентах (например, загрузку данных).

— useState — хук библиотеки React, позволяющий добавлять состояние в функциональные компоненты.

— БРС — балльно-рейтинговая система. Электронная система учёта и оценки успеваемости студентов по различным дисциплинам.

— Интеграция — процесс объединения различных систем для обмена данными и совместной работы.

— Интерфейс пользователя (UI) — визуальная часть программы, через которую осуществляется взаимодействие с системой.

— Компонентный подход — архитектурный принцип, при котором интерфейс разбивается на переиспользуемые независимые блоки (компоненты).

— Модальное окно — элемент интерфейса, который появляется поверх содержимого страницы и требует взаимодействия до его закрытия.

— Парсер — программа или модуль, выполняющий синтаксический анализ данных, преобразующий их в структурированный формат.

— Числитель / Знаменатель — деление недель учебного расписания на два чередующихся типа (первая и вторая неделя), используемое в университетской практике.

Введение

Цифровизация образовательного процесса требует создания эффективных и удобных инструментов взаимодействия между участниками — студентами и преподавателями. Одной из повседневных задач, с которой сталкиваются все участники учебного процесса, является бронирование аудиторий для проведения дополнительных занятий, консультаций, собраний и других мероприятий.

Отсутствие единой системы для учёта и контроля бронирований приводит к конфликтам, двойному использованию помещений и затруднениям при планировании учебной деятельности. Решением этой проблемы является создание протеаммного модуля, обеспечивающего удобную и прозрачную систему бронирования аудиторий.

В рамках данной курсовой работы разрабатывается модуль «Бронирование аудитории», который интегрируется в балльно-рейтинговую систему факультета. Модуль реализован в виде веб-приложения, обеспечивающего пользователю возможность просматривать доступное расписание, выбирать нужную аудиторию, а также бронировать её на определённое время. На текущем этапе хранения данных реализовано через механизм localStorage в браузере. Разработка серверной части и интеграция с базой данных запланированы на следующем этапе.

Разработка построена на основе современных веб-технологий и ориентирована на расширяемость, простоту использования и визуальную наглядность расписания. Модуль включает поддержку деления по неделям (числитель/знаменатель), визуальное выделение уже занятых ячеек и окно создания нового бронирования.

1 Постановка задачи

1.1 Цели и задачи работы

Целью курсовой работы является разработка веб-модуля «Бронирование аудитории», предназначенного для интеграции в балльно-рейтинговую систему факультета. Разрабатываемый модуль должен обеспечить возможность интерактивного взаимодействия пользователей с таблицей расписания и выполнять операции по бронированию аудиторий в выбранные временные интервалы.

Для достижения этой цели необходимо решить следующие задачи:

— Реализовать загрузку и парсинг расписания из Excel-документа;

— Отобразить структуру недельного расписания с разделением на «числитель» и «знаменатель»;

— Обеспечить возможность разового и постоянного бронирования конкретной ячейки расписания;

— Реализовать хранение информации о бронированиях локально (с возможностью расширения на серверную часть);

— Предусмотреть пользовательский интерфейс для ввода данных и подтверждения брони;

— Разработать архитектуру модуля с учётом дальнейшей интеграции в единую систему факультета.

1.2 Предметная область

Предметная область проекта включает в себя несколько ключевых аспектов, которые необходимо учитывать при разработке модуля «Бронирование аудитории» для балльно-рейтинговой системы факультета:

— Расписание занятий: необходимо реализовать функциональность отображения расписания в табличной форме, с учётом структуры

учебной недели (числитель/знаменатель), временных слотов и распределения аудиторий по дням недели;

— Бронирование аудиторий: модуль должен позволять пользователю забронировать конкретную аудиторию на определённый временной интервал, как на разовой основе, так и на постоянной, с возможностью указания ФИО и дополнительных параметров брони;

— Обработка данных: требуется реализовать обработку исходных данных из Excel-файла, преобразование их в внутренний формат и сохранение информации о бронированиях. В текущей версии данные сохраняются в локальном хранилище, однако структура должна предусматривать возможность подключения серверной части;

— Интерфейс пользователя: разработка понятного и визуально структурированного интерфейса, позволяющего пользователю легко ориентироваться в расписании, видеть доступные и занятые слоты и осуществлять бронирование;

— Будущая интеграция: при разработке архитектуры следует учитывать возможность дальнейшей интеграции модуля в существующую балльно-рейтинговую систему факультета, включая взаимодействие с другими модулями и переход на централизованное хранение данных.

Успешная реализация модуля требует комплексного подхода к учёту всех перечисленных аспектов и их объединения в единое удобное и расширяемое веб-решение.

2 Конкурентный анализ

Разработка модуля «Бронирование аудитории» актуальна в условиях цифровизации образовательных процессов. На рынке уже существуют решения, предоставляющие функции планирования и бронирования помещений, однако большинство из них рассчитаны на коммерческое использование или имеют сложную архитектуру. В данном разделе проводится анализ двух существующих решений с последующим сравнением с собственной разработкой. Это позволит выявить сильные и слабые стороны аналогов и определить, какие аспекты могут быть улучшены в рамках проектируемого приложения. Ниже отображены ключевые параметры (Таблица 1), по которым происходило сравнение продуктов.

Таблица 1 — Ключевые параметры анализа конкурентов.

Параметр	Моя разработка	Конкурент 1: Skedda	Конкурент 2: Robin
Тип системы	Веб-приложение	Облачное веб-приложение	Облачное веб-приложение
Авторизация пользователей	Нет (будет позже)	Да	Да
Бронирование по типу недели	Да (Числитель / Знаменатель)	Нет	Нет
Поддержка повторяющихся броней	Да	Да	Да
Язык интерфейса	Русский	Английский	Английский
Доступ к расписанию	Импорт из Excel	Визуальный редактор	Интеграции с календарями
Хранение данных	LocalStorage (пока что)	Облако	Облако

Открытый исходный код	Да	Нет	Нет
Возможность адаптации под ВУЗ	Да	Нет	Нет

2.1 Skedda

Skedda — это облачное решение для бронирования помещений, используемое в офисах, коворкингах, спортивных комплексах и образовательных учреждениях. Приложение позволяет гибко управлять бронированиями, устанавливать правила доступа, интегрироваться с календарями Google и Outlook (Рисунок 1). Пользователь может видеть доступность помещений в режиме реального времени и совершать бронирование через удобный интерфейс.

Основной функционал:

- Онлайн-бронирование помещений: предоставляет веб-интерфейс для поиска и бронирования доступных аудиторий, переговорных, спортзалов и других помещений;
- Гибкая настройка правил доступа: позволяет администраторам задавать правила и ограничения для разных категорий пользователей, определяя кто, когда и какие помещения может бронировать;
- Календарь с визуализацией занятости: отображает в едином окне свободные и занятые временные интервалы для всех помещений;
- Повторяющиеся и разовые бронирования: поддерживает как разовые, так и регулярные бронирования на определённые дни и интервалы.

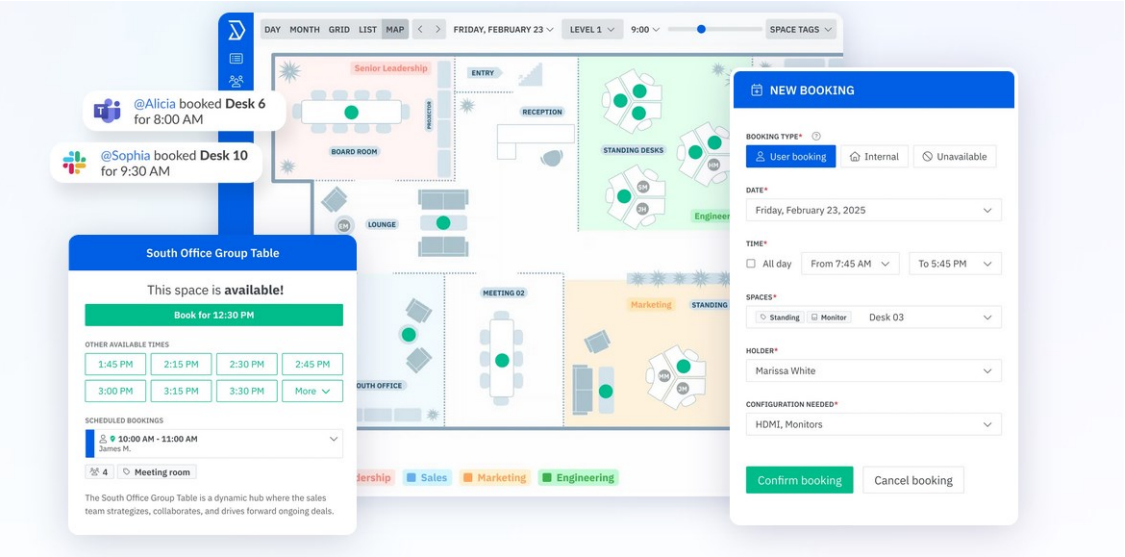

Рисунок 1 — Интерфейс бронирования помещений в Skedda

Skedda — это современное и удобное решение для онлайн-бронирования помещений, которое автоматизирует управление доступом, оптимизирует использование ресурсов и облегчает организацию совместной работы. Приложение подходит для учебных заведений, офисов и спортивных объектов, позволяя пользователям быстро находить и бронировать свободные аудитории, а администраторам — эффективно контролировать процесс и анализировать загрузку помещений.

В то же время Skedda ориентирован в первую очередь на коммерческие пространства и коворкинги, а не на образовательные учреждения. В нём отсутствует поддержка специфических для вузов функций — таких как учёт расписания занятий, разделение бронирований по типу недели (числитель/знаменатель), интеграция с университетскими учетными системами и гибкая настройка прав для разных ролей (например, староста, преподаватель). Поэтому для решения задач, связанных с бронированием аудиторий именно в вузе, требуется доработка или поиск специализированных решений.

2.2 Robin

Robin — это платформа для онлайн-бронирования рабочих мест и переговорных, которая предоставляет интуитивно понятный интерфейс, поддерживает интерактивную карту офиса, интеграцию с корпоративными календарями и аналитикой по использованию офисного пространства (Рисунок 2). Решение активно применяется в офисах и компаниях с гибридным форматом работы, позволяя сотрудникам самостоятельно выбирать рабочие места и планировать присутствие в офисе.

Основной функционал:

- Онлайн-бронирование рабочих мест и переговорных: предоставляет платформу для поиска и бронирования доступных рабочих столов, переговорных комнат и других офисных зон;
- Интерактивная карта офиса: позволяет сотрудникам видеть актуальную схему офиса, выбирать и бронировать конкретные рабочие места на плане;
- Гибкие правила бронирования: поддерживает настройку прав доступа, определяет, кто и какие ресурсы может бронировать;
- Интеграция с календарями и мессенджерами: синхронизируется с Google Calendar, Outlook, Slack и другими сервисами для автоматического напоминания и управления встречами;
- Управление гостями: позволяет создавать приглашения и бронирования для внешних посетителей;
- Поддержка гибридного и удалённого формата работы: помогает сотрудникам заранее планировать своё присутствие в офисе или удалённую работу.

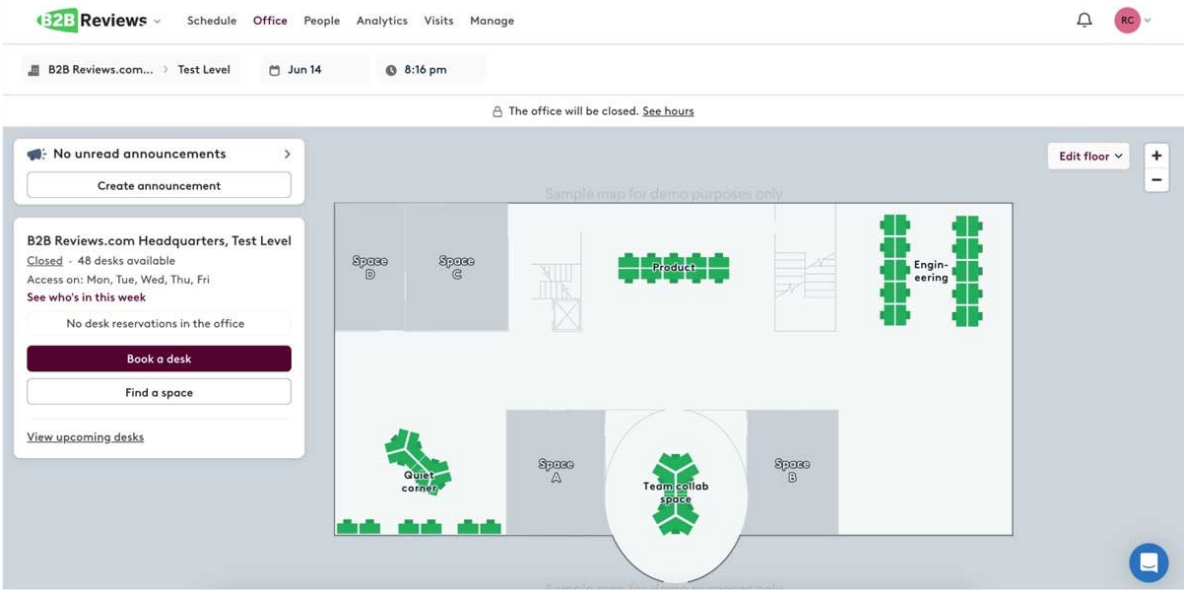

Рисунок 2 — Интерфейс бронирования рабочих мест в Robin Booking

Однако Robin Booking ориентирован, прежде всего, на корпоративную среду и организацию рабочих процессов в бизнесе. В системе нет поддержки таких важных для вузов функций, как автоматическая сверка с учебным расписанием, учёт разных ролей студентов и преподавателей, организация бронирования аудиторий для занятий, а также работа с особенностями учебного процесса (например, числитель/знаменатель и постоянные бронирования под учебные группы). Это ограничивает применение Robin Booking для задач управления аудиториями в образовательных учреждениях.

2.3 Результат обзора аналогов

В результате анализа существующих решений можно сделать следующие выводы:

— Существующие платформы бронирования, такие как Skedda и Robin, предоставляют богатый функционал, однако изначально разрабатывались не под образовательную сферу, а под нужды бизнеса бронирование рабочих мест, переговорных комнат, спортзалов и

прочего. Поэтому они не учитывают специфики вузовского расписания, например: деление недели на «Числитель» и «Знаменатель», фиксированное количество временных слотов и необходимость отображения расписания в виде сетки по дням и аудиториям;

— Ограничения по адаптивности: Оба сервиса (Skedda и Robin) являются SaaS-решениями, их невозможно доработать под собственные требования без лицензии или API-доступа, который часто закрыт или ограничен. Моя система разрабатывается с учётом возможной адаптации под любую структуру расписания и любую платформу факультета. Например, возможно будет подключение авторизации, интеграция с балльно-рейтинговой системой и дальнейшее расширение функций без ограничений.

— Язык интерфейса и локализация: и Robin, и Skedda ориентированы на англоязычную аудиторию. В отличие от них, моя разработка полностью русифицирована, понятна преподавателям и студентам без необходимости адаптации интерфейса под локальные реалии.

— Удобство и фокус на учебную задачу: в отличие от коммерческих конкурентов, которые перегружены возможностями и требуют времени на обучение персонала, моя система сосредоточена только на одной задаче — быстром и понятном бронировании аудиторий преподавателями и студентами.

Таким образом, вывод по результатам анализа аналогов можно сформулировать следующим образом:

— Ни одна из проанализированных платформ не решает задачу бронирования учебных аудиторий с учётом особенностей расписания высших учебных заведений. Моя разработка заполняет эту нишу и обеспечивает базовую, но эффективную функциональность для

вузовской среды, с возможностью гибкой доработки и масштабирования в будущем.

3 Реализация

3.1 Архитектура

3.1.1 Общая структура

Разрабатываемое веб-приложение «Бронирование аудиторий» представляет собой клиентскую систему, реализованную с использованием фреймворка React и сохраняющую данные о бронированиях в localStorage браузера. В дальнейшем планируется расширение проекта за счёт добавления серверной части, взаимодействующей с базой данных и API балльно-рейтинговой системы.

Приложение содержит следующие основные компоненты:

— Модуль загрузки расписания из Excel-файла (.xlsx);

— Визуализация таблицы расписания по дням недели и времени;

— Переключение типа недели (числитель/знаменатель);

— Модальное окно бронирования;

— Система хранения и отображения активных бронирований.

3.1.2 Паттерны проектирования

Для организации кода использованы следующие подходы:

— Компонентный подход React: вся функциональность разбита на переиспользуемые компоненты (ClassroomTable, BookingForm);

— Модульность и изоляция бизнес-логики: логика парсинга (Parser.js) и хранения (bookingStorage.js) вынесена в отдельные файлы;

— Состояние управляется через useState и useEffect, обеспечивая реактивное поведение и перерисовку интерфейса при изменении данных;

— Локальное хранилище (localStorage) используется для имитации баз данных на клиенте и временного хранения бронирований до появления полноценного API.

3.1.3 Взаимодействие компонентов

На данный момент структура взаимодействия компонентов выглядит следующим образом (Рисунок 3):

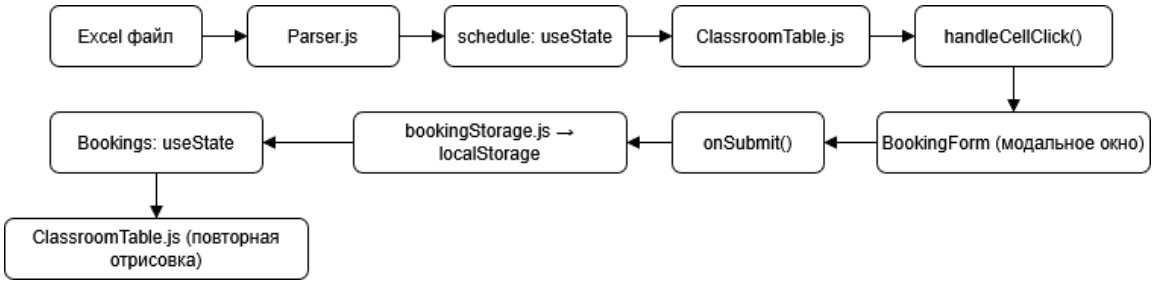

Рисунок 3 — Схема потока данных в модуле бронирования аудиторий

3.1.4 Ключевые модули

В составе веб-приложения бронирования аудиторий реализованы следующие ключевые модули. Каждый из них отвечает за отдельный функциональный блок и подробно представлен в Приложениях:

— ClassroomTable.js — основной компонент интерфейса веб-приложения, реализующий отображение таблицы расписания. Модуль отвечает за загрузку и разметку расписания из Excel-файла, переключение между числителем и знаменателем, отображение занятых и свободных ячеек. Подробный код компонента приведён в ПРИЛОЖЕНИЕ А.

— BookingForm.js — модуль, реализующий форму для бронирования. Открывается при выборе свободной ячейки. Пользователь указывает

имя, комментарий и тип бронирования. Код формы представлен в ПРИЛОЖЕНИЕ Б.

— bookingStorage.js — отвечает за сохранение и извлечение информации о бронированиях из localStorage, так как серверная часть пока не реализована. Используется для хранения всех данных о занятиях. Приведён в ПРИЛОЖЕНИЕ В.

— Parser.js — модуль обработки Excel-файла расписания. Преобразует данные в универсальный формат для рендеринга. Поддерживает извлечение текста, цвета и комментариев. Подробная реализация — в ПРИЛОЖЕНИЕ Г.

3.2 Хранение данных и логика бронирования

В текущей версии приложения взаимодействие с серверной частью отсутствует. Все данные хранятся и обрабатываются локально в браузере пользователя с помощью localStorage.

3.2.1 Стек технологий

— JavaScript / React — основной язык и фреймворк для построения интерфейса;

— localStorage — механизм хранения бронирований;

— xlsx (SheetJS) — библиотека для парсинга Excel-файлов;

— CSS — стилизация таблицы и интерфейса;

— HTML5 input type="file" — используется для загрузки .xlsx файлов с расписанием.

3.2.2 Логика бронирования

Каждая ячейка таблицы представляет собой пересечение дня недели, времени и аудитории. При клике на ячейку открывается модальное окно с формой бронирования. Пользователь указывает:

— ФИО;

— тип бронирования (разовое / постоянное).

После подтверждения данные бронирования сохраняются в localStorage с полями:

— classroom — номер аудитории;

— time — время пары;

— date — день недели;

— user — инициалы пользователя;

— permanent — тип бронирования.

При загрузке расписания данные из localStorage проверяются на наличие конфликтов. Забронированные ячейки окрашиваются в разные цвета:

— зеленый — постоянное бронирование;

— желтый — разовое бронирование.

3.2.3 Обработчик пользователя

Предусмотрена возможность расширения логики хранения с переходом на полноценный сервер:

— создание REST API для отправки и получения бронирований;

— подключение базы данных PostgreSQL или SQLite;

— реализация авторизации и разграничения прав доступа.

3.3 Клиентская часть

3.3.1 Роли пользователей в системе

В текущей версии приложения роли пользователей ещё не реализованы, однако при дальнейшем подключении к базе данных и разработке серверной части предусмотрено разделение прав доступа.

В системе предусмотрены две основные роли пользователей:

— Преподаватель;

— Студент.

Преподаватель: имеет возможность бронировать аудитории как на постоянной, так и на разовой основе. Может видеть все существующие бронирования, включая созданные другими пользователями.

Студент: По умолчанию не имеет права бронировать аудитории. Однако для старост и заместителей старост групп будет предоставлена возможность разового бронирования аудиторий для проведения дополнительных занятий, собраний и т.п. Также будет реализована функция просмотра свободных и занятых аудиторий без возможности редактирования.

Реализация разграничения доступа будет осуществлена на этапе подключения серверной части и авторизации.

3.3.2 Интерфейсы и пользовательские пути

Веб-приложение предлагает простой и понятный интерфейс для пользователей. Интерфейс реализован с использованием библиотеки React и позволяет работать с расписанием в интерактивной форме. Ниже описаны ключевые сценарии взаимодействия пользователя с приложением.

3.3.2.1 Загрузка расписания

После запуска веб-приложения пользователю предлагается загрузить Excel-файл с актуальным расписанием (Рисунок 4). Расписание автоматически парсится и визуализируется в табличной форме с учётом:

— дня недели;

— времени занятий;

— номеров аудиторий;

— различия между числителем и знаменателем.

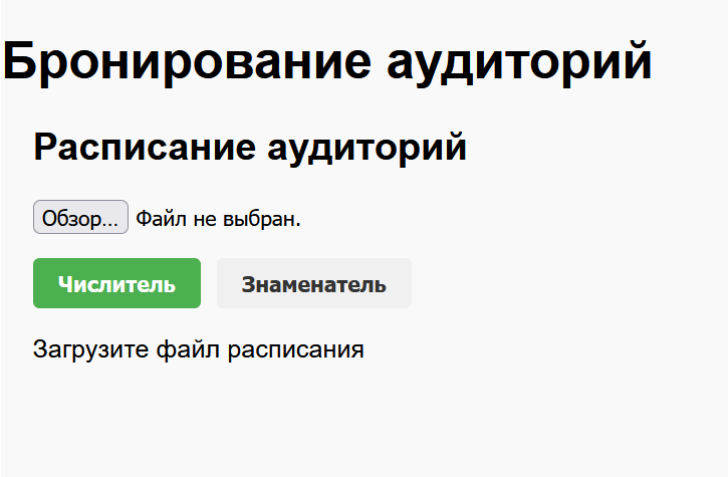

Рисунок 4 — Главное меню и форма загрузки файла расписания

3.3.2.2 Отображение расписания

После успешной загрузки данных отображается таблица с расписанием аудиторий, разбитая по дням и временным интервалам (Рисунок 5).

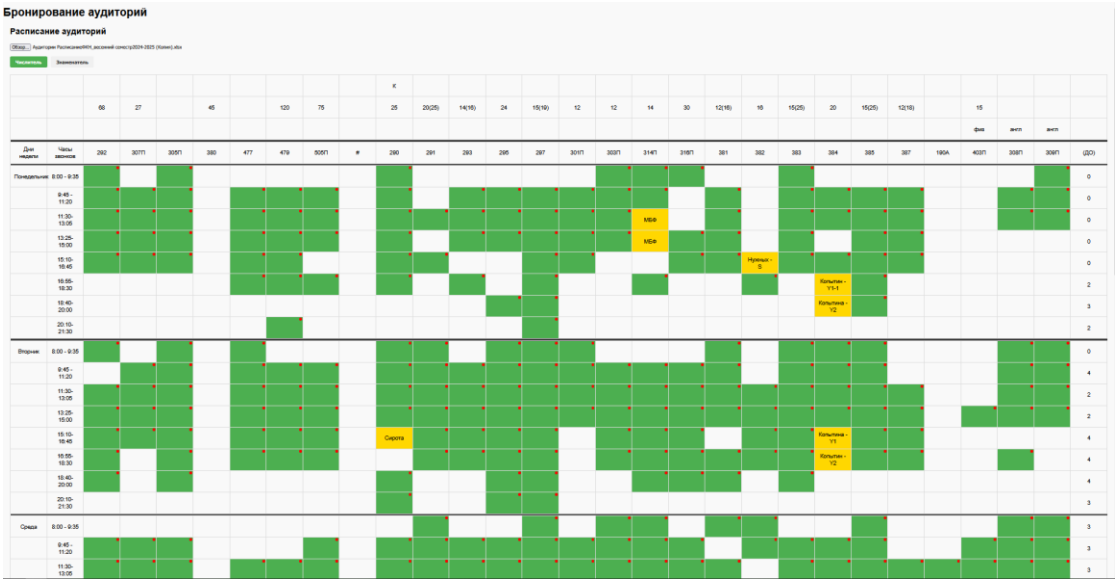

Рисунок 5 — Таблица расписания после загрузки данных

3.3.2.3 Наведение на забронированную ячейку

Каждая ячейка таблицы представляет собой сочетание день–время–аудитория. Ячейки могут содержать:

— текстовое описание пары;

— цветовую маркировку (например, если ячейка была окрашена в Excel);

— иконку комментария при наличии дополнительной информации в исходном файле.

Если ячейка уже занята (например, занятие было импортировано из расписания), при наведении на неё отображается всплывающая подсказка с деталями (Рисунок 6).

Рисунок 6 — Информация при наведении на занятую ячейку

3.3.2.4 Переключение между числителем и знаменателем

Пользователь может выбрать тип недели — числитель (Рисунок 7) или знаменатель (Рисунок 8) — с помощью переключателя. В зависимости от выбора отображаются соответствующие строки расписания, связанные с текущей неделей.

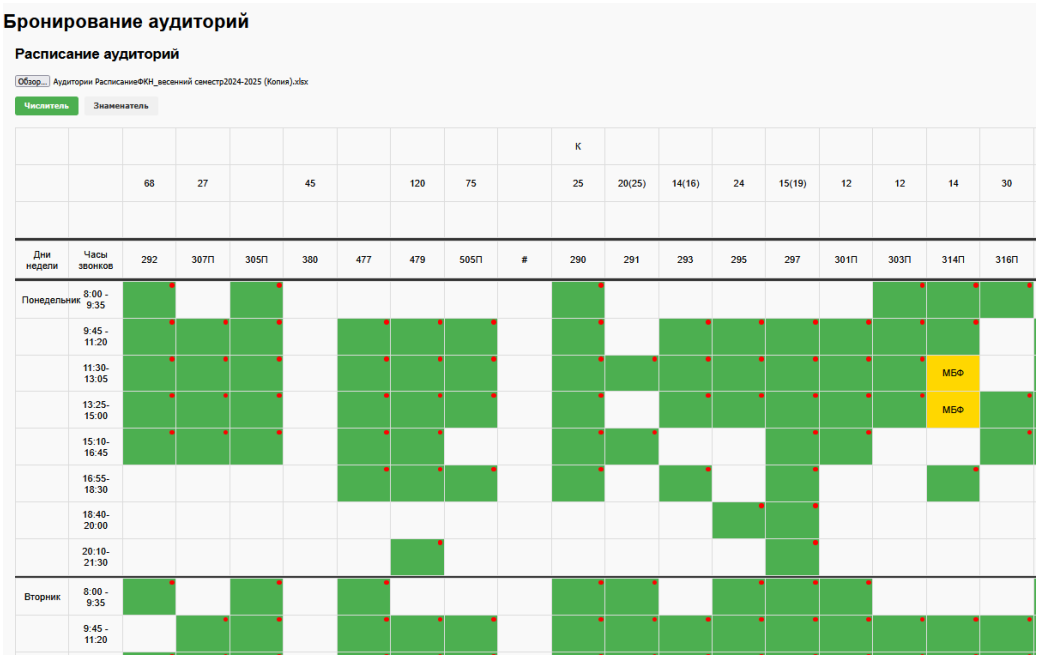

Рисунок 7 — Числитель

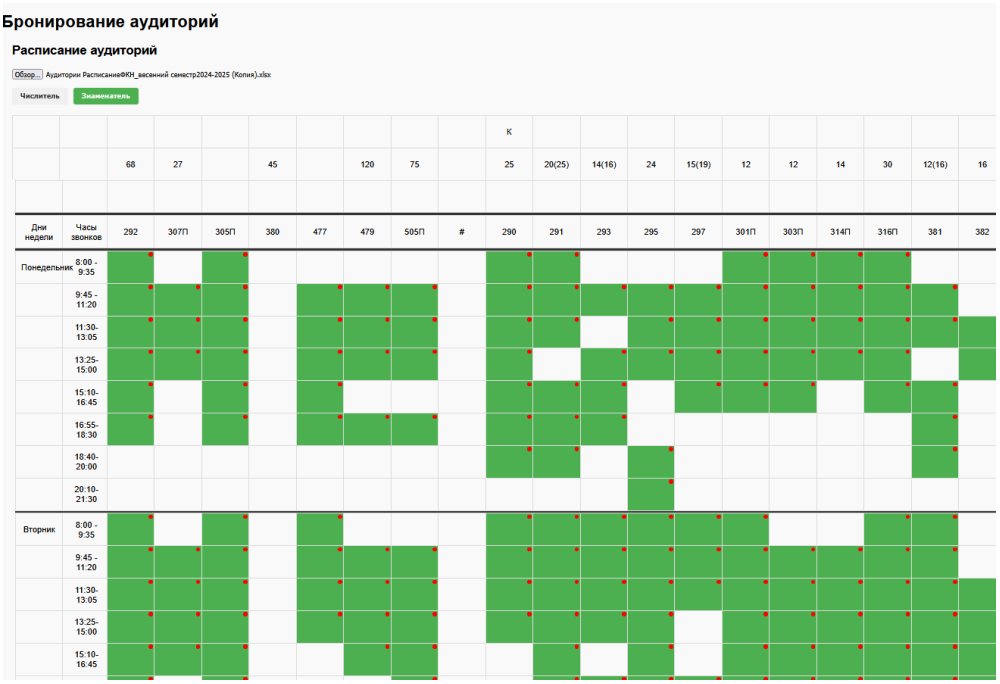

Рисунок 8 — Знаменатель

3.3.2.5 Создание бронирования

Пользователь может кликнуть на любую свободную ячейку, после чего откроется модальное окно с формой бронирования (Рисунок 9). В форме необходимо указать:

— имя пользователя;

— цель бронирования;

— тип бронирования: разовое или постоянное.

После сохранения бронирования, соответствующая ячейка в таблице будет визуально помечена как занятая. Разные типы бронирования выделяются разными цветами.

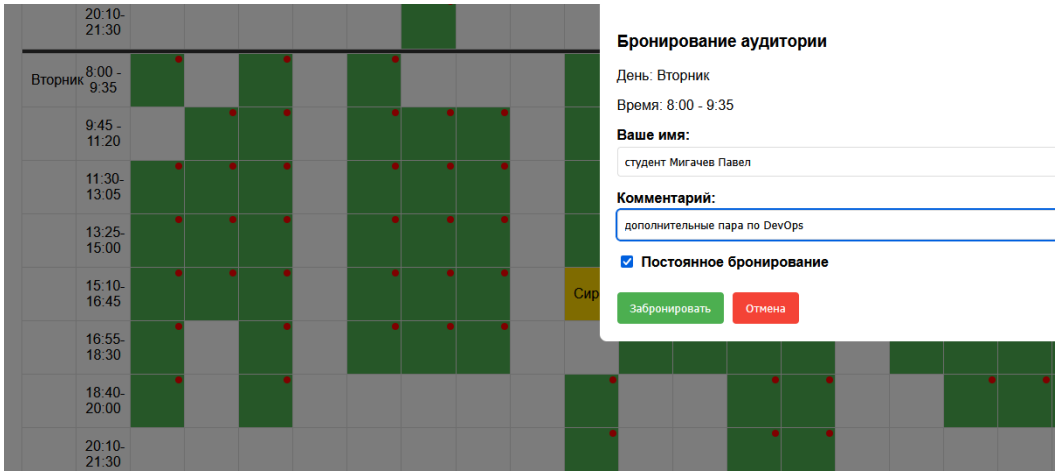

Рисунок 9 — Модальное окно бронирования аудитории

3.3.2.6 Подтверждение бронирования

После заполнения формы и нажатия кнопки "Забронировать" появляется всплывающее уведомление об успешном бронировании (Рисунок 10).

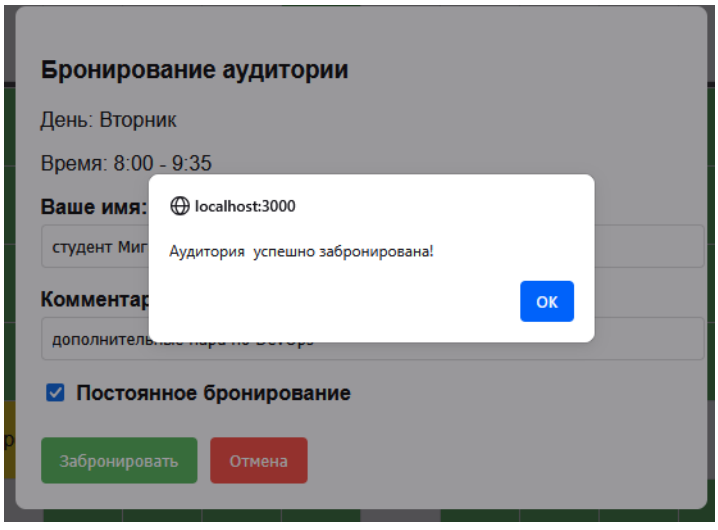

Рисунок 10 — Уведомление об успешном бронировании

3.4 База данных

На текущем этапе проект реализован без подключения к серверной части и базам данных — все данные о бронированиях сохраняются локально в localStorage браузера. Однако, в дальнейших планах предусмотрена интеграция полноценной базы данных для обеспечения многопользовательского доступа, авторизации и долговременного хранения информации.

Планируемая структура базы данных:

Для обеспечения необходимой функциональности планируется использовать реляционную базу данных — например, PostgreSQL. Предполагается следующая структура таблиц:

users (пользователи):

— id — идентификатор пользователя;

— full_name — ФИО;

— role — роль (преподаватель, студент, староста и т.д.);

— login / password_hash — данные для входа.

classrooms (аудитории):

— id — идентификатор аудитории;

— number — номер/название аудитории;

— capacity — вместимость (при необходимости);

— description — дополнительная информация.

schedule (расписание):

— id — идентификатор строки;

— day_of_week — день недели;

— time_slot — время;

— classroom_id — ссылка на таблицу аудиторий;

— text — описание пары;

— week_type — числитель/знаменатель.

bookings (бронирования):

— id — идентификатор бронирования;

— user_id — кто забронировал;

— classroom_id — какая аудитория;

— day_of_week — день;

— time_slot — время;

— permanent — флаг: разовое/постоянное;

— purpose — цель бронирования.

Дополнительные возможности при подключении БД:

Интеграция базы данных откроет следующие возможности:

— авторизация и разграничение доступа по ролям;

— отображение бронирований от всех пользователей;

— защита от конфликтов при одновременном бронировании;

— статистика использования аудиторий;

— расширение функций администрирования.

Заключение

В рамках курсового проекта была разработана клиентская часть модуля «Бронирование аудиторий» для балльно-рейтинговой системы факультета. Целью работы являлось создание удобного пользовательского интерфейса для отображения расписания и управления процессом бронирования учебных аудиторий.

На текущем этапе реализована следующая функциональность:

— визуализация расписания с разделением на числитель и знаменатель;

— выбор и бронирование свободной ячейки с указанием целей;

— сохранение данных о бронированиях в localStorage;

— переключение между неделями, отображение комментариев и цветов ячеек;

— подготовка к ролификации пользователей (преподаватель / студент);

— адаптация логики под структуру расписания, выгружаемого из Excel-файла.

Проведён конкурентный анализ существующих решений, таких как Skedda и Robin. Было выявлено, что большинство из них не предоставляет гибкой возможности локального бронирования без централизованной авторизации, что делает предложенное решение более независимым и адаптированным к потребностям конкретного факультета.

В дальнейшем планируется реализация серверной части, подключение к базе данных, а также внедрение авторизации с разграничением прав доступа по ролям. Это обеспечит многопользовательскую работу с данными, повысит надёжность хранения информации и откроет возможности для интеграции с другими модулями факультетской информационной системы.

Таким образом, данный модуль является первым шагом к созданию полноценной системы управления расписанием и пространственными ресурсами факультета.

Список использованных источников

1. Документация по библиотеке React [Электронный ресурс]. – Режим доступа: https://react.dev/ – (Дата обращения 09.05.2025)
2. Документация библиотеки SheetJS (xlsx) [Электронный ресурс]. – Режим доступа: https://github.com/SheetJS/sheetjs – (Дата обращения 09.05.2025)
3. Документация по JavaScript [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/JavaScript – (Дата обращения 01.06.2025)
4. Документация по CSS [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/CSS – (Дата обращения 01.06.2025)
5. Документация по Web API: localStorage [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/API/Window/localStorage (Дата обращения 24.05.2025)

ПРИЛОЖЕНИЕ А

```
const ClassroomTable = () => {
  const [schedule, setSchedule] = useState([]);
  const [parsed, setParsed] = useState(false);
  const [bookings, setBookings] = useState([]);
  const [selectedCell, setSelectedCell] = useState(null);
  const [weekType, setWeekType] = useState('Числитель');

  useEffect(() => {
    setBookings(getBookings());
  }, []);

  // Группировка по дням и рендер всей таблицы
  const renderRows = () => {
    if (!schedule.length) return null;

    let lastDay = '';
    let dayRows = [];
    let rows = [];

    // Группируем по дням
    schedule.forEach((row, rowIndex) => {
      const currentDay = typeof row[0]?.text === 'string' ?
row[0].text.trim() : '';
      const isNewDay = currentDay && currentDay !== lastDay;

      if (isNewDay) {
        if (dayRows.length > 0) {
          processDayRows(lastDay, dayRows, rows);
          dayRows = [];
        }
        lastDay = currentDay;
      }
      dayRows.push({ row, rowIndex });
    });
  };

 числитель/знаменатель
  const processDayRows = (day, dayRows, outputRows) => {
    if (dayRows.length === 0) return;

    if (outputRows.length > 0) {
      outputRows.push(
        <tr key={`div-${day}`} className="day-divider">
          {dayRows[0].row.map((_, i) => <td key={`div-${i}`} />)}
        </tr>
      );
    }
```

```
    dayRows.forEach(({ row, rowIndex }, k) =>
      const isHeaderRow = rowIndex < 4;

      const shouldShow = isHeaderRow ||
        (weekType === 'Числитель' && k % 2 === 0) ||
        (weekType === 'Знаменатель' && k % 2 !== 0);

      if (!shouldShow) return;

      const isStriped = !isHeaderRow && k % 2 === 0;

            // Для обычных строк проверяем бронирования
            const classroom = schedule[1]?.[cellIndex]?.text;
            const time = row[1]?.text;
            const dayStr = typeof day === 'string' ? day.trim() : '';
            const classroomStr = typeof classroom === 'string' ?
classroom.trim() : '';
            const timeStr = typeof time === 'string' ? time.trim() : '';
            const booking = classroomStr && timeStr
              ? isCellBooked(dayStr, timeStr, classroomStr)
              : null;
            const bgClass = booking
              ? (booking.permanent ? 'permanent' : 'temporary')
              : cell.color || '';
```

ПРИЛОЖЕНИЕ Б

```
import React, { useState } from 'react';

const BookingForm = ({ onSubmit, onCancel, initialData }) => {
  const [formData, setFormData] = useState({
    user: '',
    comment: '',
    isPermanent: false,
    ...initialData
  });

  const handleChange = (e) => {
    const { name, value, type, checked } = e.target;
    setFormData(prev => ({
      ...prev,
      [name]: type === 'checkbox' ? checked : value
    }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    onSubmit({
      ...formData,
      permanent: formData.isPermanent
    });
  };

  return (
    <form onSubmit={handleSubmit}>
      <div className="form-group">
        <label>Ваше имя:</label>
        <input
          type="text"
          name="user"
          value={formData.user}
          onChange={handleChange}
          required
        />
      </div>

      <div className="form-group">
        <label>Комментарий:</label>
        <input
          type="text"
          name="comment"
          value={formData.comment}
          onChange={handleChange}
        />
```

```
        </div>

        <div className="form-group checkbox">
          <label>
            <input
              type="checkbox"
              name="isPermanent"
              checked={formData.isPermanent}
              onChange={handleChange}
            />
            Постоянное бронирование
          </label>
        </div>

        <div className="form-actions">
          <button type="submit" className="btn-submit">
            Забронировать
          </button>
          <button
            type="button"
            className="btn-cancel"
            onClick={onCancel}
          >
            Отмена
          </button>
        </div>
      </form>
    );
};
```

ПРИЛОЖЕНИЕ В

```
const STORAGE_KEY = 'classroom_bookings';

export const getBookings = () => {
  const raw = localStorage.getItem(STORAGE_KEY);
  return raw ? JSON.parse(raw) : [];
};

export const addBooking = (booking) => {
  const bookings = getBookings();
  bookings.push(booking);
  localStorage.setItem(STORAGE_KEY, JSON.stringify(bookings));
};

export const removeBooking = (index) => {
  const bookings = getBookings();
  bookings.splice(index, 1);
  localStorage.setItem(STORAGE_KEY, JSON.stringify(bookings));
};

export const clearBookings = () => {
  localStorage.removeItem(STORAGE_KEY);
};
```

ПРИЛОЖЕНИЕ Г

```
import * as XLSX from 'xlsx';

export const parseExcelFile = async (file) => {
  const data = await file.arrayBuffer();
  const workbook = XLSX.read(data, {
    type: 'array',
    cellStyles: true,
    cellComments: true
  });

  const sheet = workbook.Sheets[workbook.SheetNames[0]];
  const range = XLSX.utils.decode_range(sheet['!ref']);
  const parsed = [];

  for (let row = range.s.r; row <= range.e.r; row++) {
    const rowData = [];

    for (let col = range.s.c; col <= range.e.c; col++) {
      const address = XLSX.utils.encode_cell({ r: row, c: col });
      const cell = sheet[address];

      if (cell) {
        const text = cell.v ?? '';
        const comment = cell.c?.[0]?.t || '';
        const hasComment = Boolean(comment);

        // Получаем цвет ячейки
        const excelColor = cell.s?.fill?.fgColor?.rgb || '';
        const cleanColor = excelColor.startsWith('FF') ? excelColor.slice(2)
: excelColor;

        let colorClass = '';

        // Регулярки
        const isTextLike = typeof text === 'string' && /[a-zA-Za-яA-
Я]/.test(text);
        const isTimeFormat = /^\d{1,2}:\d{2}\s*-
\s*\d{1,2}:\d{2}$/.test(text);
        const isJustDigits = /^\d+$/.test(text);
        const isRoomCode = /^\d+[ПпАаАа]?$/.test(text);
        const excludedWords = [
          'дни недели', 'часы звонков',
          'понедельник', 'вторник', 'среда', 'четверг', 'пятница', 'суббота',
          'физ', 'англ', '(до)', 'к', 'х'
        ];
```

```
        const isExcludedWord =
excludedWords.includes(String(text).trim().toLowerCase());

        if (hasComment) {
          colorClass = 'occupied';
        } else if (
          isTextLike &&
          !isTimeFormat &&
          !isJustDigits &&
          !isRoomCode &&
          !isExcludedWord
        ) {
          colorClass = 'permanent';
        } else {
          if (['4CAF50', '66BB6A'].includes(cleanColor)) colorClass =
'occupied';
          else if (['FFD700', 'FBC02D'].includes(cleanColor)) colorClass =
'permanent';
          else if (['1E90FF', '42A5F5'].includes(cleanColor)) colorClass =
'temporary';
        }

        rowData.push({
          text,
          color: colorClass,
          comment,
          hasComment
        });
      } else {
        rowData.push({ text: '', color: '', comment: '', hasComment: false
});
      }
    }

    parsed.push(rowData);
  }

  return parsed;
};
```