МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИСТ Кафедра Вычислительная техника

|  |  |
| --- | --- |
| УТВЕРЖДАЮ  Руководитель предприятия | К ЗАЩИТЕ ДОПУСТИТЬ  Зав. кафедрой |
|  |  |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / | \_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ |
| подпись инициалы, фамилия | подпись инициалы, фамилия |
| «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. |

Бакалаврская работа

Вид ВКР (дипломный проект (работа) / бакалаврская работа / магистерская диссертация)

Тема Разработка менеджера задач деятельности IT компании

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Г.И. Вольтер /

подпись инициалы, фамилия

Обозначение ВКР БР-УлГТУ-09.03.01-18/1092-2022 Группа ИВТВМбд-41

для технических направлений подготовки (специальностей)

Направление подготовки (специальность) 09.03.01 «Информатика и вычислительная техника» профиль «Вычислительные машины, комплексы, системы и сети»

Руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_Святов К.В.\_\_\_ /

подпись, дата инициалы, фамилия

Ульяновск 2022г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИСТ Кафедра Вычислительная техника

Направление подготовки (специальность) 09.03.01 «Информатика и вычислительная техника» профиль «Вычислительные машины, комплексы, системы и сети»

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ |
|  | Зав. кафедрой |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_/ К.В.Святов / |
|  | подпись инициалы, фамилия |
|  | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г. |

**ЗАДАНИЕ**

на *выпускную работу бакалавра*

указать вид ВКР (дипломный проект (работа) / бакалаврская работа / магистерская диссертация)

обучающемуся Вольтер Григория Игоревича курса \_4\_ группы \_ИВТВЬбд-41 \_

фамилия, имя, отчество

Тема ВКР «Разработка менеджера задач деятельности IT компании»

утверждена приказом по УлГТУ от «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г. №\_\_\_\_\_\_\_\_\_\_

Срок сдачи обучающимся законченной ВКР « » июня 2022

Исходные данные к ВКР Необходимо разработать автоматизированную систему реализующая регистрацию компании и администратора, регистрацию сотрудников администратором в компании, авторизацию пользователей, разделение пользователей по уровню доступа, инструментарий по администрированию информации внутри созданной администратором компании, инструментарий по чтению, редактированию, удалению и постановке задач сотрудниками, личные кабинеты пользователей с возможностью изменения информации в нем. Автоматизированная система реализована в виде web-приложения, программный интерфейс реализован в виде отдельного скрипта исполняемого на сервере необходимый для обмена информацией между клиентской и серверной частью приложения. Для хранения информации о пользователях, задачах и компаниях необходимо использовать реляционную СУБД

Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

1. Расширенная постановка задачи 2. Проектирование системы 3. Реализация системы, примеры работы приложения 4. Тестирование

Перечень графического материала (с точным указанием обязательных чертежей)

Общая структура системы, диаграмма вариантов использования, ER-диаграмма базы данных системы, диаграммы последовательности, IDEF0 диаграмма автоматизируемого бизнес-процесса, алгоритмы работы реализоавнной системы, скриншоты работы системы. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Руководитель\_\_\_\_\_\_\_\_\_\_зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_К.В. Святов\_\_\_\_\_\_/

должность, учёная степень, ученое звание подпись инициалы, фамилия

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_Г.И. Вольтер\_\_\_\_\_ / подпись обучающегосяинициалы, фамилия

**Оглавление**

[Аннотация 6](#_Toc106394188)

[Введение 7](#_Toc106394189)

[Глава 1. Расширенная постановка задачи 10](#_Toc106394190)

[1.1. Анализ предметной области 10](#_Toc106394191)

[1.2. Анализ существующих решений 13](#_Toc106394192)

[1.3. Описание проблемы, которую необходимо решить 14](#_Toc106394193)

[1.4. Цели и задачи 15](#_Toc106394194)

[1.5. Описание функциональных требований 15](#_Toc106394195)

[1.6. Описание нефункциональных требований 16](#_Toc106394196)

[1.7. Описание используемых технологий и инструментов 19](#_Toc106394197)

[1.8. Обоснование выбора технологической платформы 25](#_Toc106394198)

[Глава 2. Проектирование структуры менеджера задач 27](#_Toc106394199)

[2.1. Архитектура web-приложения 27](#_Toc106394200)

[2.2. Моделирование процессов системы, взаимодействие пользователей с системой 32](#_Toc106394201)

[Глава 3. Реализация и тестирование 41](#_Toc106394202)

[3.1. Реализация базы данных и связи между таблицами 41](#_Toc106394203)

[3.2. Разработка программного интерфейса приложения 45](#_Toc106394204)

[3.3. Разработка системы переходов и навигации по страницам 49](#_Toc106394205)

[3.4. Реализация регистрации компании и администратора 53](#_Toc106394206)

[3.5. Реализация авторизации пользователей в системе 59](#_Toc106394207)

[3.6. Реализация личного профиля пользователя 63](#_Toc106394208)

[3.7 Реализация инструментов администрирования информации в компании 65](#_Toc106394209)

[3.8. Реализация инструментов работы с задачами 73](#_Toc106394210)

[Глава 4. Тестирование 85](#_Toc106394211)

[4.1. Сценарии регистрации и авторизации 88](#_Toc106394212)

[4.2. Сценарии общего функционала 93](#_Toc106394213)

[4.3. Сценарии общего функционала 97](#_Toc106394214)

[Заключение 100](#_Toc106394215)

[Список источников 101](#_Toc106394216)

[Приложение 1. HTML-разметка компонентов веб-приложения 105](#_Toc106394217)

[Приложение 2. Серверная часть приложения, API 122](#_Toc106394218)

Аннотация

**к бакалаврской работе Вольтер Григория Игоревича на тему**

**«Разработка менеджера задач деятельности IT компании»** Кафедра ВТ УлГТУ, 2022 г.

Научный руководитель: к.т.н., доцент Святов К.В.

**Работа** содержит 128 страницы, включающих, 48 иллюстраций, 10 таблиц, список информационных источников из 30 наименований и 2 приложения.

**Ключевые слова**: менеджер задач, автоматизированная система.

Представленная бакалаврская работа посвящена разработке менеджера задач деятельности IT компании.

Пояснительная записка содержит следующие разделы:

1. Расширенная постановка задачи. В данной главе описаны требования к системе, цели и задачи проекта, анализ существующих решений, а также приведено описание используемых технических средств с обоснованием их выбора.
2. Проектирование менеджера задач. Эта глава включает в себя описание процессов функционирования системы, ролей пользователей, USE-case диаграммы взаимодействия пользователя с системой, диаграммы последовательностей.
3. В третьей главе описан процесс реализации функций системы: регистрации, авторизации пользователей, работа с задачами, работа администратора, демонстрация работы системы.
4. В четвертой главе описан процесс тестирования приложения за счет сценариев взаимодействия пользователя с системой.

Введение

Сложно не заметить, как бурно развивается IT индустрия вокруг нас, с каждым днем количество IT компаний по всему миру увеличивается, для взаимодействия между сотрудниками внутри компании и для осуществления управления рабочими процессами активно применяются решения в виде менеджеров задач.

Основными целями менеджера задач внутри компании является постановка задачи от одного пользователя к другому, выполнение полученных задач, мониторинг и редактирование поставленных задач, составление отчетности по затраченному времени при выполнении задачи, составление отчетности по проделанной работе сотрудником за месяц, год.

На момент 2022 года существует большой выбор менеджеров задач, предоставляющих функционал разной степени вариативности. Наличие столь большого количества решений делает выбор единого менеджера задач для своей IT компании затруднительным. Большинство решений, представленных на рынке являются платными или предоставляю урезанный функционал, ограниченный платной подпиской, немногие готовы предложить возможность отслеживания времени выполнения задачи, простой и понятный интерфейс. Обилие функционала в решениях для ведения дел внутри компании может отвлечь от конкретики, а именно от создания и мониторинга выполнения задач.

Разработка менеджера задач деятельности IT компании нацелена на создание единого продукта, включающего в себя только основные цели менеджера задач. Разрабатываемое веб-приложение так же включает в себя разделение ролей на администратора и пользователя, роль администратора расширяет права пользователя возможностями редактирования всей информации, происходящей в системе внутри созданной компании, создания личных кабинетов пользователей внутри компании.

При разработке веб-приложения применялись следующие инструменты:

1) Браузер Google Chrome версии 102.0.5005.63 (Официальная сборка), (64 бит)

2) Редактор кода Visual Studio Code версии 1.67

3) Веб-приложение phpMyAdmin для доступа к базам данных и редактирования записей в них.\

При разработке приложения были предъявлены следующие требования:

1) Разработка пользовательского интерфейса

2) Разработка системы авторизации и регистрации пользователей

3) Разделение прав доступа пользователей

4) Личный кабинет пользователя

5) Редактирование данных в личном кабинете

6) Просмотр поставленных задач

7) Просмотр выполненных задач

8) Просмотр полученных задач

9) Редактирование поставленных задач

10) Выполнение полученных задач

11) Изменение статуса задач

12) Редактирование данных о сотрудниках

13) Редактирование данных о компании

14) Создание личных кабинетов

Выбранная тема данной выпускной квалификационной работы является актуальной, так как существует потребность в едином менеджере задач учета деятельности IT компании, реализующего только самые нужные функции и обладающим простым и понятным интерфейсом.

Глава 1. Расширенная постановка задачи

1.1. Анализ предметной области

Исследуемую область можно определить, как «Автоматизированные системы(АС) и системы управления рабочими процессами (СУРБ)».

Автоматизированные системы — это системы, включающие в себя штат сотрудников и совокупность средств автоматизации их деятельности, выполняющих информационную технологию реализации установленных функций. С помощью АС можно формировать решения на основе автоматизации процессов информатизации в разных сферах деятельности и их комбинациях.

Обычно виды автоматизированных систем выделяют согласно их деятельности, также они могут классифицироваться по виду управляемого объекта или процесса. Например, в зависимости от вида деятельности выделяют следующие виды АС: автоматизированные системы управления (АСУ), системы автоматизированного проектирования (САПР), автоматизированные системы научных исследований (АСНИ). В зависимости от вида управляемого объекта или процесса АСУ можно выделить, например, АСУ технологическими процессами (АСУТП) или АСУ предприятиями (АСУП).

Автоматизированные системы выполняют определенные заданные функции, то есть совокупность действий АС, направленную на достижение определенной цели. Функция или часть функции АС представляет собой задачу АС как некоторую совокупность автоматических действий, выполнение которых приводит к результату заданного вида.

В процессе функционирования АС представляет собой совокупность комплекса средств автоматизации (КСА), организационно-методических и технических документов и специалистов, использующих их в процессе своей профессиональной деятельности.

В процессе проектирования АС и ее частей разрабатывают, в общем случае, следующие виды обеспечений: техническое, программное, информационное, организационно-методическое, метрологическое, правовое, математическое, лингвистическое, эргономическое.

Система управления задачами – инструмент, который помогает сделать рабочий процесс прозрачным, понятным и упорядоченным. Используя такую систему, руководитель получает возможность постоянно держать руку на пульсе каждого проекта, а сотрудники четко понимают, чем и когда им нужно заниматься. Программное обеспечение рабочего процесса таким образом не только повышает скорость обработки и разгружает сотрудников, но и предотвращает ошибки с технической стороны. Это связано с тем, что приложение может проверить вводимые данные на соответствие ранее заданным критериям и предотвратить переход к следующему этапу обработки в случае обнаружения нарушений.

Если рабочие процессы структурированы в цифровом формате, обеспечивается стандартизация процедур, что положительно сказывается на качестве результатов работы. Кроме того, проблемы легче отследить, поскольку каждый сотрудник следует одному и тому же рабочему процессу. Изменения в процедуре легко внедрить и обеспечить соблюдение требований с помощью инструментов документооборота.

Оптимизация может быть достигнута в каждой области деятельности компании путем внедрения систем управления рабочими процессами.

Система исключительно простая и эффективная, она помогает отслеживать сроки и статус каждой задачи на одном экране. Одновременно с этим программа для управления задачами собирает подробную статистику: записывает, кто чем занимался, сколько потратил времени, как активно участвовал в процессе.

Система управления задачами повышает эффективность работы руководителя за счёт автоматизации процессов и гибкого подхода к управлению.

Используется единая среда постановки задач и контроля над их выполнением. Поэтому в любой момент времени у пользователя есть возможность видеть, какие задачи и кому поставлены.

Есть возможность вносить изменения в задачи, и сотрудники оперативно получают указания, изменения по задачам, новые задачи. Это снижает затраты времени.

Система управления задачами автоматически оповещает сотрудников и руководителя о сроках окончания выполнения задач. Это повышает дисциплину.

Основными элементами представленной предметной области являются данные рабочего процесса IT компании, пользователи и администратор компании. За рабочий процесс подразумевается работа сотрудников компании с множеством полученных задач, мониторинг выполнения поставленных задач, администрирование информации внутри IT компании, созданием новых пользователей в системе.

1.2. Анализ существующих решений

Перед началом работы по проектирования собственного решения необходимо произвести анализ текущей ситуации в предметной области. Для это был произведен поиск приложений, реализующих функционал системы управления рабочими процессами, в частности системы управления задачами. Ниже приведена сравнительна таблица возможностей самых популярный решений.

Таблица 1. Сравнение менеджеров задач

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название | Учет времени выполнения задачи | Простота использования | Ведение отчетности | Цена | Бесплатный период |
| Trello | - | + | + | 5$ за пользователя в месяц при годовой оплате. | Не ограничен по времени |
| Smartsheet | - | - | + | от 7$ за пользователя в месяц при годовой оплате. | Есть |
| Basecamp | + | + | - | 99$ в месяц независимо от количества членов в команде. | 30дней |
| Wrike | - | - | + | от 9,80$ за пользователя в месяц при годовой оплате. | неограниченно по времени до 5 пользователей |

1.3. Описание проблемы, которую необходимо решить

В первую очередь автоматизация управления рабочими процессами решает проблему взаимодействия сотрудников между собой путем автоматического предупреждения о выходе задачи за сроки реализации. Избавляет сотрудников от потери информации при ведении работы, хранит статусы задач и истории их выполнения.

Благодаря автоматизации управления рабочими процессами сотрудники могут сосредоточиться на основных задачах. Отпадает необходимость в многочисленных согласованиях и запросах, не нужно искать документы и спрашивать контактных лиц.

1.4. Цели и задачи

Цель работы: создание автоматизированной системы управления рабочими процессами для упрощения ведения задач для рядового пользователя и администратора компании, снижение затрат труда и времени на мониторинг выполнение задач, составления отчетности эффективности сотрудников.

1.5. Описание функциональных требований

Менеджер задач деятельности IT компании должен предоставлять администратору возможности внесения в базу данных компании информации о пользователях, создания личных кабинетов для сотрудников компании, мониторинга сроков выполнения задач сотрудников, составления отчетов выполнения задач по всем сотрудникам, а также редактирования информации о себе и о компании в личном кабинете, сотруднику системы предоставляется возможность выполнения операция над поставленными, полученными и выполненными задачами, изменять данные о себе в личном кабинете.

Также менеджер задач реализует следующие функциональные возможности:

1. Регистрация компании и администратора в системе;
2. Регистрация сотрудников администратором в системе;
3. Авторизация пользователей в системе;
4. Разделение пользователей по уровню доступа: администратор, сотрудник;
5. Возможность для сотрудников просматривать список задач: поставленных, выполненных, полученных;
6. Возможность для сотрудника отвечать на полученные задачи изменять статус на: выполняется, приостановлена, завершена;
7. Возможность для сотрудника редактировать информацию о поставленных задачах;
8. Возможность для пользователей изменять информацию о себе в личном кабинете: логин, пароль, имя, телефон;
9. Автоматическое формирование отчета о сотрудниках в конце месяца или по желанию администратора.

1.6. Описание нефункциональных требований

При разработке веб-приложения будут учитываться следующие виды нефункциональных требований:

1. Availability (доступность) - требования ко времени непрерывной работы приложения, например, 24x7, минимальное время простоя и т.п. Приложение будет всегда запущена в окне браузера и отображать последнюю страницу посещению пользователем.
2. Reliability (надежность) - поведение приложения при наступлении нештатных ситуаций, например, автоматический перезапуск, восстановление работы, дублирование важных данных, резервирование логики. При авторизации пользователя его данные будут сохранятся в cooke массив и при перезагрузке страницы данные не слетят и не обнулятся, если пользователь авторизовался но закрыл окно браузера, он сможет вписав в строку url адрес предыдущей вернуться к ней. Так же приложение хранит историю посещения страниц что позволит вернуться обратно в приложение при смены окна браузера.
3. Durability - требования к долговременному хранению результатов работы приложения, например, использование базы данных, требования ко времени продолжительности хранения данных. Основные данные приложения: задачи, пользователи хранятся на стороне сервера и передаются в клиент только когда пользователь взаимодействует с инструментами реализующие взаимодействие с таблицами базы данных. Например при заполнении формы создания личного кабинета данные хранятся в форме до тех пор пока не будет закрыта форма или не будет добавлен сотрудник.
4. Usability - требования к удобству использования приложения с точки зрения использования, поддержки. Приложение реализовано с помощью веб фреймворка VueJS верси 3.0 его работу поддерживает любой браузер совместимый с ECMAScript 5, серверная часть написана с использованием PHP версии 7.4 поддержка разработчиками завершается 28 ноября 2022, поддерживается всеми браузерами.
5. Security - требования к безопасности работы или использования приложения, связанные с разграничением доступа, работой с приватными данным, снижения подверженности рискам от внешних атак. В приложении реализован маршрутизатор, запрещающий неавторизированным пользователям получать доступ к страницам веб-приложения, сотрудникам получать доступ к инструментарию администратора.
6. Configurability - требования к конфигурируемости работы приложения, взаимодействия и расположения компонентов.
7. Reusability - требования к повторному использованию реализации или компонентов приложения, а также реализация приложения с возможностью повторного его использования для различных задач. Каждая страница приложения собирается из заранее созданных компонентов, если есть необходимость вызывать один и тот же блок кода но в разных страницах или списке страниц VueJs предоставляет широкий выбор возможностей по использованию компонента повторно. Как пример использование компонент шапки сайта хранящий навигационные элементы приложения вызывается как у сотрудника, так и у администратора.
8. Extensibility - требования к расширяемости приложения в связи с появлением новых функциональных требований. Для дальнейшего расширения приложения новыми функциями разработчику необходимо создать новый компонент и вызвать его в нужной странице.
9. Interoperability - требования к взаимодействию между компонентами решения, между внешними компонентами, использование стандартных протоколов и технологий взаимодействия. Клиентская часть взаимодействует с серверной за счёт отправки данных из нее по средству POST запросов.
10. Localizability - требования к возможности и простоте локализации приложения, перечень языков, на которые предполагается локализация приложения. Есть возможность создавать словари и переключать изык системы тем самым изменять значения полей кнопок и доугих элементов веб приложения.

1.7. Описание используемых технологий и инструментов

Клиентская часть приложения разработана с использованием современного прогрессивного фреймворка Vue 3, написанного на языке TypeScript и предназначенный специально для разработки пользовательских интерфейсов и создания веб приложений на уровне клиента. Для ускорения разработки и оформления страниц клиентского приложения используется свободный набор инструментов Bootstrap 5 включающий в себя HTML и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса.

Серверная часть приложения состоит из скрипта, отвечающего за программный интерфейс взаимодействия между клиентской частью приложения и базой данных. Скрипт написан с помощью языка общего назначения PHP версии 8.0, предназначенного для создания динамических веб сайтов с применением библиотеки RedBeanPHP необходимой для упрощения процесса сохранения объектов в реляционную базу данных и их извлечением.

Полный список используемых технологий при разработке:

1. HTML5
2. CSS
3. PHP 8.0
4. RedBeanPHP
5. VueJS 3
6. JavaScript
7. MySQL
8. phpMyAdmin
9. Bootstrap 5
10. Visual Studio Code

1. HTML5 — язык для структурирования и представления отображения веб-страниц и контента. Он предназначен для упрощения проектирования и разработки за счет языка разметки, обеспечивающего стандартизированный и интуитивно понятный пользовательский интерфейс. HTML5 предоставляет разработчику средства для секционирования и структуризации веб-страниц, а также позволяет создавать обособленные компоненты, которые обеспечивают логическую организацию сайта. Например, контент может быть структурирован внутри множества параграфов, маркированных списков или с использованием изображений и таблиц данных.

2. CSS (Cascading Style Sheets — каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки. Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML.

CSS используется создателями веб-страниц для задания цветов, шрифтов, стилей, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось отделение описания логической структуры веб-страницы, которое производится с помощью HTML или других языков разметки от описания внешнего вида этой веб-страницы, которое и производится с помощью формального языка CSS. Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом.

3. PHP – это широко используемый язык сценариев общего назначения с открытым исходным кодом. Основным преимуществом PHP является предоставление возможности быстрого создания динамически генерируемых web-страниц, а также создание HTML документов с внедренными командами PHP.

PHP способен генерировать и преобразовывать не только HTML документы, но и изображения разных форматов JPEG, GIF, PNG, файлы PDF и FLASH. PHP способен формировать данные в любом текстовом формате, включая XHTML и XML.

PHP – это кроссплатформенная технология. Дистрибутив PHP доступен для большинства операционных систем, включая Linux, многие модификации Unix, Microsoft Windows, Mac OS и многих других. PHP поддерживается на большинстве веб-серверов, таких, как Apache, Microsoft Internet Information Server (IIS), Microsoft Personal Web Server.

PHP поддерживает работу с ODBC и большое количество баз данных: MySQL, MSQL, Oracle, PostgreSQL, SQLite.

4. RedBeanPHP – это независимая бесплатная библиотека, основанная на модели объектно-реляционного отображения (ORM). RedBeanPHP позволяет адаптировать схему базы данных в соответствии с потребностями без выполнения SQL-запросов в течение всего процесса разработки. Использование библиотеки также дает возможность при развертывании на рабочих серверах сохранить ссылочную целостность базы данных. RedBean поддерживает все основные бесплатные СУБД с открытым кодом: MySQL, MariaDB, SQLite, PostgreSQL, CUBRID и Firebird/Interbase, а также она совместима со всеми версиями PHP от 5.2 до 8.0.

5. VueJS 3 - современный прогрессивный фреймворк, написанный на языке TypeScript и предназначенный для создания веб-приложений на уровне клиента. Основная область применения данного фреймворка - это создание и организация пользовательского интерфейса.

Одним из ключевых моментов в работе Vue 3 является виртуальный DOM. Структура веб-страницы, как правило, описывается с помощью DOM (Document Object Model), которая представляет организацию элементов html на странице. Для взаимодействия с DOM (добавления, изменения, удаления html-элементов) применяется JavaScript. Но когда мы пытаемся манипулировать html-элементами с помощью JavaScript, то мы можем столкнуться со снижением производительности, особенно при изменении большого количества элементов. А операции над элементами могут занять некоторое время, что неизбежно скажется на пользовательском опыте. Однако если бы мы работали из кода js с объектами JavaScript, то операции производились бы быстрее.

Для этого Vue 3 использует виртуальный DOM. Виртуальный DOM представляет легковесную копию обычного DOM. Если приложению нужно узнать информацию о состоянии элементов, то происходит обращение к виртуальному DOM. Если данные, которые используются в приложении Vue 3, изменяются, то изменения вначале вносятся в виртуальный DOM. Потом Vue выбирает минимальный набор компонентов, для которых надо выполнить изменения на веб-странице, чтобы реальный DOM соответствовал виртуальному. Благодаря виртуальному DOM повышается производительность приложения. Vue 3 поддерживается всеми основными современными браузерами.

6. JavaScript - это легковесный, интерпретируемый или JIT-компилируемый, объектно-ориентированный язык, наиболее широкое применение которого это создание сценариев веб-страниц. JavaScript это прототипно-ориентированный, мультипарадигменный язык с динамической типизацией, который поддерживает объектно-ориентированный, императивный и декларативный стили программирования.

С помощью JavaScript доступны к исполнению следующие функции: возможность изменять страницы браузеров; добавление или удаление тегов веб-страницы; изменение стилей страницы; информация о действиях пользователя на странице; запрос доступа к случайной части исходного кода страницы; внесение изменений в этот код; выполнение действия с cookie-файлами.

Область применения языка обширна: среди программ, которые используют JS, присутствуют и тестовые редакторы, и приложения как для компьютеров, так и мобильные, и прикладное ПО. Преимуществами JavaScript являются его повсеместная поддержка современными браузерами, полезные функциональные настройки, постоянное развитие и обновление языка, легкость взаимодействия.

7. MySQL - свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

8. phpMyAdmin - phpMyAdmin — веб-приложение с открытым кодом, написанное на языке PHP и представляющее собой веб-интерфейс для администрирования СУБД MySQL. PhpMyAdmin позволяет через браузер и не только осуществлять администрирование сервера MySQL, запускать команды SQL и просматривать содержимое таблиц и баз данных. Приложение пользуется большой популярностью у веб-разработчиков, так как позволяет управлять СУБД MySQL без непосредственного ввода SQL команд.

Приложение распространяется под лицензией GNU General Public License и поэтому многие другие разработчики интегрируют его в свои разработки, например XAMPP, Denwer, AppServ, Open Server.

9. Bootstrap 5 - свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

Фреймворк Bootstrap является одним из самых популярных и распространённых инструментов, использующих самые новые функции и возможности CSS и HTML, однако он может не поддерживаться старыми версиями браузеров.

К положительным сторонам можно отнести следующие качества Bootstrap:

1. Наличие изменяющегося языка стилей (Less);

2. Кросс-браузерность;

3. Большой набор шаблонов;

4. Высокая скорость работы;

5. Простота в использовании.

10. Visual Studio Code - редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации. Visual Studio Code поддерживает плагины, доступные через Visual Studio Marketplace. Они могут включать в себя дополнения к редактору, поддержку дополнительных языков программирования, а также статические анализаторы кода.

1.8. Обоснование выбора технологической платформы

Выбор представленных технологий обоснован следующими факторами:

1. Написание кода веб-приложения осуществлялось в редакторе код Visual Studio Code, так как текстовый редактор предоставлял множество простых в использовании плагинов для всех используемых в проекте языков и фреймворков, встроена консоль позволяла осуществлять контроль версий приложения.

2. Использование библиотеки стилей Bootstrap позволяло значительно облегчить формирование гипертекста приложения, облегчило создание интерактивных элементов таких как кнопки и модальные окна.

3. Библиотека ReadBeansPHP позволяет упростить написание команд создания таблиц базы данных и создание связей между ними. Автоматически создаются таблицы связей «многие ко многим», а также библиотека предоставляет доступ к набору команд для получения записей из базы данных.

4. VueJS использует виртуальную модель DOM, копию исходной модели DOM веб-сайта, которую фреймворк использует для определения обновлений элементов без необходимости визуализации всей модели DOM, тем самым повышая производительность приложения и делая рендеринг страниц довольно быстрым.

Благодаря двусторонней привязке данных обновление связанных компонентов и отслеживание обновлений данных в целом становится проще.

Связанные данные получают во Vue реактивные обновления, как объекты DOM. Эта функция делает платформу идеальной для приложений с обновлениями в реальном времени. Что касается разработки, такая реактивная природа может упростить и ускорить обновление данных.

Глава 2. Проектирование структуры менеджера задач

На этапе анализа предметной области были выделены основные задачи, которые должен решать менеджер задач деятельности IT компании, определены технические требования, инструменты и технологии, которые будут применятся для реализации программы.

2.1. Архитектура web-приложения

В основе разрабатываемой системы лежит архитектура «клиент-сервер», в которой сетевая нагрузка распределены между поставщиками услуг (сервисов), называемых серверами, и заказчиками услуг, называемых клиентами.

Клиентом является браузер пользователя, где посредством DOM и JavaScript обеспечивается взаимодействие пользователя с данными. Использование JavaScript само по себе не является обязательным, однако функционирование большинства Web-приложений без JS не представляется возможным. В качестве среды взаимодействия клиента с сервером используется сеть Internet.



Рисунок 1. Среда взаимодействия Internet

Основными достоинствами архитектуры «клиент-сервер» являются:

* Возможность, в большинстве случаев, распределить функции вычислительной системы между несколькими независимыми компьютерами в сети. Это
* позволяет упростить обслуживание вычислительной системы. В частности, замена, ремонт, модернизация или перемещение сервера, не затрагивают клиентов.
* Все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов. На сервере проще обеспечить контроль полномочий, чтобы разрешать доступ к данным только клиентам соответствующими правами доступа.
* Позволяет объединить различные клиенты. Использовать ресурсы одного сервера часто могут клиенты с разными аппаратными платформами, операционными системами и т.п.

Основные недостатки архитектуры «клиент-сервер»:

* в случае использования централизованной системы, неработоспособность основного сервера может сделать неработоспособным все Web-приложение;

Для взаимодействия компонентов web-приложения с сервером будет использована технология Rest-api.

Термин состоит из двух аббревиатур, которые расшифровываются следующим образом. API (Application Programming Interface) — это код, который позволяет двум приложениям обмениваться данными с сервера. На русском языке его принято называть программным интерфейсом приложения. REST (Representational State Transfer) — это способ создания API с помощью протокола HTTP. На русском его называют «передачей состояния представления».

Технологию REST API применяют везде, где пользователю сайта или веб-приложения нужно предоставить данные с сервера. Например, при нажатии иконки с видео на видеохостинге REST API проводит операции и запускает ролик с сервера в браузере. В настоящее время это самый распространенный способ организации API. Он вытеснил ранее популярные способы SOAP и WSDL.

У RESTful нет единого стандарта работы: его называют «архитектурным стилем» для операций по работе с серверов. Такой подход в 2000 году в своей диссертации ввел программист и исследователь Рой Филдинг, один из создателей протокола HTTP.

У RESTful есть 7 принципов написания кода интерфейсов.

* Отделения клиента от сервера (Client-Server). Клиент — это пользовательский интерфейс сайта или приложения, например, поисковая строка видеохостинга. В REST API код запросов остается на стороне клиента, а код для доступа к данным — на стороне сервера. Это упрощает организацию API, позволяет легко переносить пользовательский интерфейс на другую платформу и дает возможность лучше масштабировать серверное хранение данных.
* Отсутствие записи состояния клиента (Stateless). Сервер не должен хранить информацию о состоянии (проведенных операций) клиента. Каждый запрос от клиента должен содержать только ту информацию, которая нужна для получения данных от сервера.
* Кэшируемость (Casheable). В данных запроса должно быть указано, нужно ли кэшировать данные (сохранять в специальном буфере для частых запросов). Если такое указание есть, клиент получит право обращаться к этому буферу при необходимости.
* Единство интерфейса (Uniform Interface). Все данные должны запрашиваться через один URL-адрес стандартными протоколами, например, HTTP. Это упрощает архитектуру сайта или приложения и делает взаимодействие с сервером понятнее.
* Многоуровневость системы (Layered System). В RESTful сервера могут располагаться на разных уровнях, при этом каждый сервер взаимодействует только с ближайшими уровнями и не связан запросами с другими.
* Предоставление кода по запросу (Code on Demand). Серверы могут отправлять клиенту код (например, скрипт для запуска видео). Так общий код приложения или сайта становится сложнее только при необходимости.
* Начало от нуля (Starting with the Null Style). Клиент знает только одну точку входа на сервер. Дальнейшие возможности по взаимодействию обеспечиваются сервером.

Сам по себе RESTful не является стандартом или протоколом. Разработчики руководствуются принципами REST API для создания эффективной работы с серверов для своих сайтов и приложений. Принципы позволяют выстраивать серверную архитектуру с помощью других протоколов: HTTP, URL, JSON и XML.

Это отличает REST API от метода простого протокола доступа к объектам SOAP (Simple Object Access Protocol), созданного Microsoft в 1998 году. В SOAP взаимодействие по каждому протоколу нужно прописывать отдельно только в формате XML. Также в SOAP нет кэшируемости запросов, более объемная документация и реализация словаря, отдельного от HTTP. Это делает стиль REST API более легким в реализации, чем стандарт SOAP.

Несмотря на отсутствие стандартов, при создании REST API есть общепринятые лучшие практики, например:

• использование защищенного протокола HTTPS

• использование инструментов для разработки API Blueprint и Swagger

• применение приложения для тестирования Get Postman

• применение как можно большего количества HTTP-кодов (список)

• архивирование больших блоков данных

Архитектура REST API основывается на протоколе передачи гипертекста HTTP (Hypertext Transfer Protocol). Это стандартный протокол в интернете, созданный для передачи гипертекста. Сейчас с помощью HTTP отправляют любые другие типы данных.

Каждый объект на сервере в HTTP имеет свой уникальный URL-адрес в строгом последовательном формате.

В REST API есть 4 метода HTTP, которые используют для действий с объектами на серверах:

• GET (получение информации о данных или списка объектов)

• DELETE (удаление данных)

• POST (добавление или замена данных)

• PUT (регулярное обновление данных)

Такие запросы еще называют идентификаторами CRUD: create (создать), read (прочесть), update (обновить) delete (удалить). Это стандартный набор действий для работы с данными.

В каждом HTTP-запросе есть заголовок, за которым следует описание объекта на сервере — это и есть его состояние.

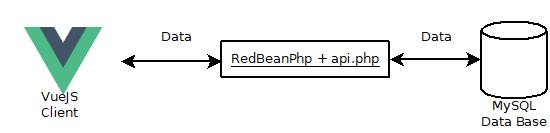


Рисунок. 2. Взаимодействие компонентов web-приложения

2.2. Моделирование процессов системы, взаимодействие пользователей с системой

На основе функциональных требований было смоделировано 2 основных вариантов использования и 15 прецедентов. Обзор прецедентов и их описание представлены в таблице 2.

Таблица 2 – Таблица вариантов использования

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Основной Актер | Наименование | Формулировка |
| 1 | Администратор | Администрирование | Включает в себя изменение данных о сотрудниках, IT компании, создание личных кабинетов |
| 2 | Администратор | Изменение данных о сотрудниках | Включает в себя чтение, редактирование и удаление данных о сотрудниках |
| 3 | Администратор | Редактирование данных о сотрудниках | Включает в себя изменение логина, имени, пароля сотрудника, сохранение данных |
| 4 | Администратор | Изменение данных о IT компании | Включает в себя чтение, редактирование данных о компании |
| 5 | Администратор | Редактирование данных о IT компании | Включает в себя изменение названия, телефона, ИНН, организации, сохранение данных в базу данных |
| 6 | Администратор | Создание личных кабинетов | Включает в себя создание имени, логина и пароля для сотрудника компании, сохранение информации о сотруднике в базу данных |
| 7 | Администратор/Сотрудник | Изменение данных в личном кабинете | Включает редактирование информации о пользователе |
| 8 | Администратор/Сотрудник | Редактирование данных в личном кабинете пользователя | Включает чтение информации и изменение логина, имени и пароля, сохранение в базу данных изменений |
| 9 | Администратор/Сотрудник | Работа с задачами | Включает в себя чтение полученных задач, чтение выполненных задач, чтение созданных задач |
| 10 | Администратор/Сотрудник | Чтение полученных задач | Включает в себя прецедент выполнения задач |
| 11 | Администратор/Сотрудник | Выполнение задач | Включает в себя изменение статуса задачи: Started, Stopped, Closed. Сохранение статуса и времени выполнения в базу данных |
| 12 | Администратор/Сотрудник | Чтение выполненных задач | Включает в себя просмотр списка выполненных задач и время затраченное на выполнение |
| 13 | Администратор/Сотрудник | Чтение созданных задач | Включает в себя создание новых задач, редактирование поставленных задач |
| 14 | Администратор/Сотрудник | Создание новых задач | Включает в себя создания заголовка, описания задачи, выбор исполнителя ,крайнего срока, сохранение в базу данных |

Основные варианты использования web-приложения представляются в виде диаграммы UseCase. Диаграмма использования поясняющая всевозможные взаимодействия Администратора и Сотрудника с системой представлены на рисунке 3 и 4.

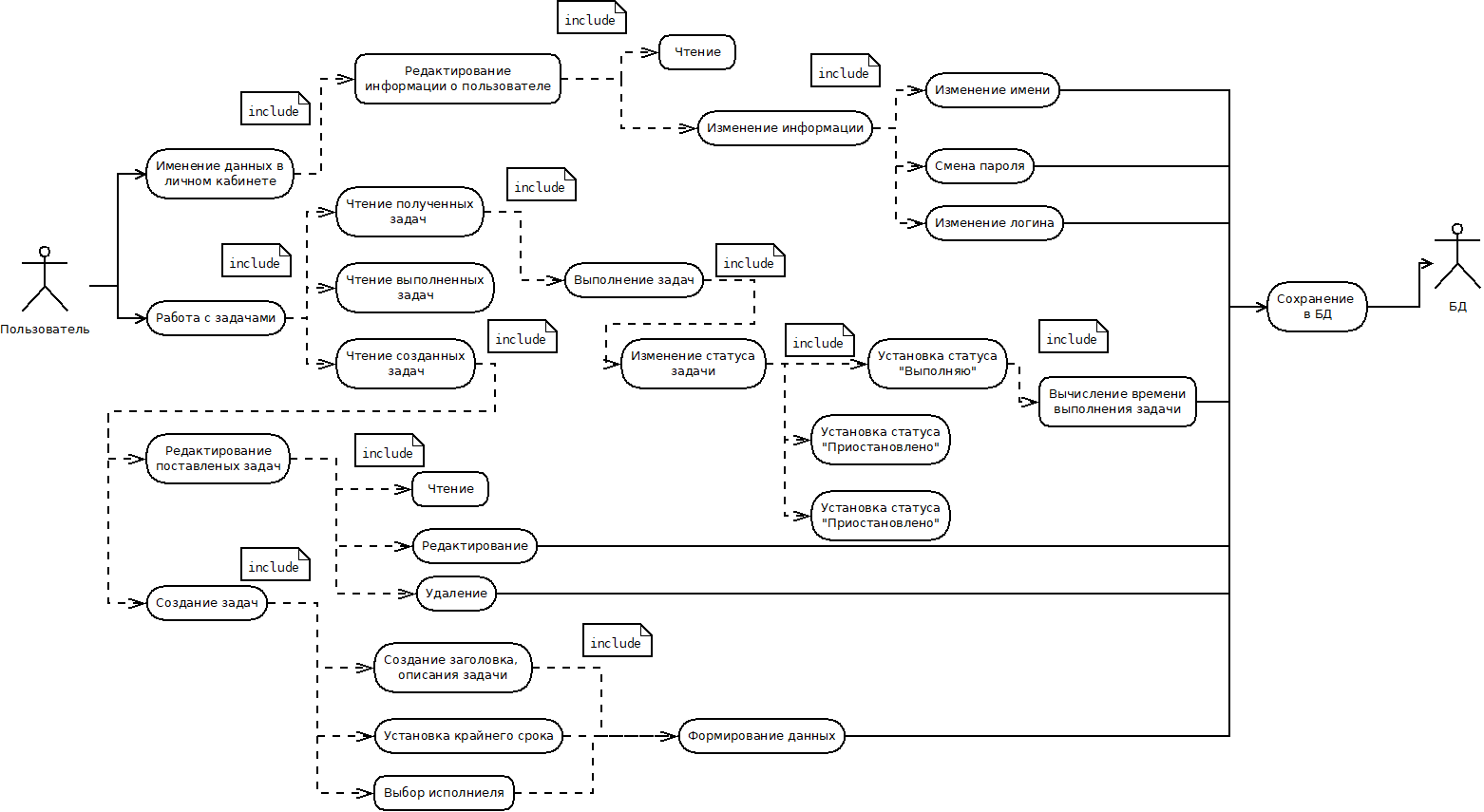


Рисунок. 3. Диаграмма вариантов использования описывающая взаимодействие сотрудника с системой

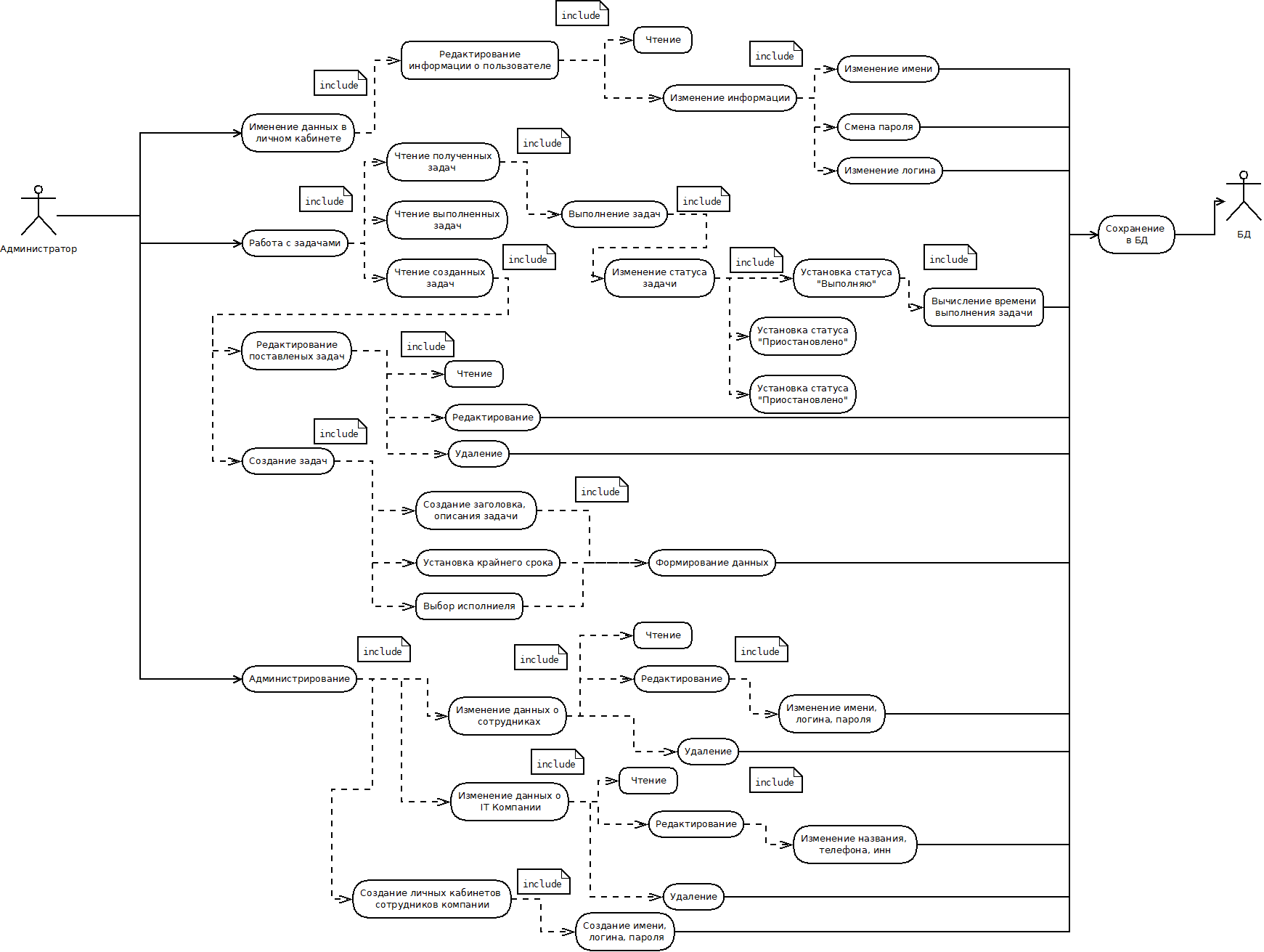


Рисунок. 4. Диаграмма вариантов использования описывающая взаимодействие администратора с системой

Диаграмма вариантов использования показывает общую картину взаимодействия пользователя с системой, однако для более детального описания логики сценария используют диаграмму последовательностей.

Диаграммы последовательностей обычно содержат объекты, которые взаимодействуют в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями. Впрочем, часто возвращаемые результаты обозначают лишь в том случае, если это не очевидно из контекста.

Объекты обозначаются прямоугольниками с подчеркнутыми именами (чтобы отличить их от классов).

Сообщения (вызовы методов) - линиями со стрелками.

Возвращаемые результаты - пунктирными линиями со стрелками.

Прямоугольники на вертикальных линиях под каждым из объектов показывают “время жизни” (фокус) объектов.

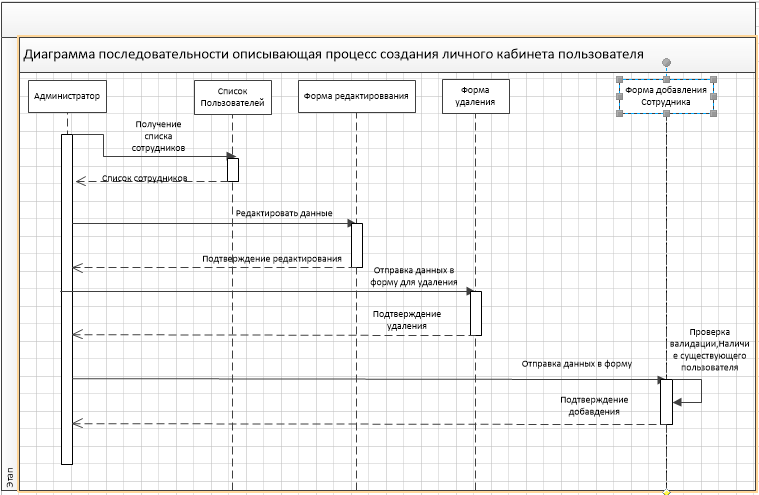


Рисунок. 5. Диаграмма последовательности описывающая процесс создания личного кабинета пользователя



Рисунок. 6. Диаграмма последовательности описывающая процесс работы пользователя с задачами

Для описания автоматизируемого бизнес-процесса создания и выполнения задач используют методологию IDEF0. Графическая нотация, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов. В IDEF0 рассматриваются логические отношения между работами, а не их временная последовательность.

Стандарт IDEF0 представляет организацию как набор модулей, здесь существует правило — наиболее важная функция находится в верхнем левом углу, кроме того, существуют правила сторон:

* стрелка входа всегда приходит в левую кромку активности,
* стрелка управления — в верхнюю кромку,
* стрелка механизма — нижняя кромка,
* стрелка выхода — правая кромка.

Бизнес процесс создания задач состоит из входящих стрелки – «Авторизация в системе». Эта вводная необходима для начала работы над созданием задачи. Управляющие для создания задач – это «Заголовок», «Описание», «Крайний срок». В роли «Механизмов» выступают исполнитель и постановщик задачи. Постановщик – это человек, который задает задаче заголовок, описание и устанавливает крайний срок, а исполнитель является лицом, указанным для выполнения задачи.



Рисунок. 7. IDEF0 диаграмма бизнес-процесса создания задачи

Бизнес процесс выполнения задач состоит из входящих стрелкок – «Авторизация в системе», «Список полученных задач». Эта вводные необходимы для начала выполнения задачи. Управляющие – это «Требования задачи». В роли «Механизмов» выступает исполнитель.



Рисунок. 8. IDEF0 диаграмма бизнес-процесса выполнения задачи

Глава 3. Реализация и тестирование

3.1. Реализация базы данных и связи между таблицами

Для представления базы данных используют ER-диаграммы сущностей связи. Диаграммы сущность-связь позволяют использовать наглядные графические обозначения для моделирования сущностей и их взаимосвязей.

Различают концептуальные и физические ER-диаграммы. Концептуальные диаграммы не учитывают особенностей конкретных СУБД. Физические диаграммы строятся по концептуальным диаграммам и представляют собой прообраз конкретной базы данных. Сущности, определенные в концептуальной диаграмме, становятся таблицами, атрибуты

становятся колонками таблиц, связи реализуются путем миграции ключевых атрибутов родительских сущностей и создания внешних ключей. В данном разделе представлены физические ER-диаграммы для СУБД MySQL.

На рисунке 3.1.1 представлена ER-диаграмма базы данных системы, которая содержит все сущности системы и отображает связи между ними.

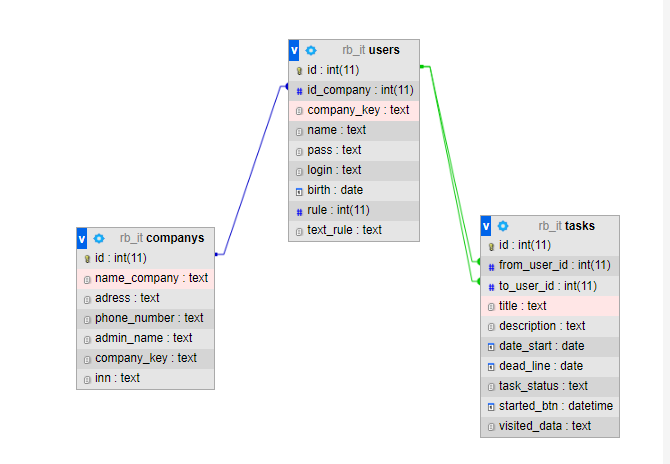


Рисунок. 3.1.1 ER-диаграмма всех сущностей базы данных системы

На рисунке видно, что система состоит из 3 таблиц: таблицы компаний(companys), таблицы пользователей(users), таблицы задач(tasks). Таблицы имеют первичные ключи id, сущность user устанавливает связь с сущностью companys посредством вторичного ключа id\_company, сущность tasks связанна с таблицей users за счет вторичных ключей Идентификатора постановщика задачи(from\_user\_id) и Идентификатором исполнителя задачи(from\_user\_id) ссылающих на id пользователя таблицы users.

Таблица companys предназначена для хранения списка зарегистрированных компаний в системе. Каждая компания имеет следующие поля: Уникальный номер записи(id), Название компании(name\_company), Адрес компании(address), Контактный номер телефона(phone\_number), Имя администратора, указанного при регистрации(admin\_name), ИНН(inn). Таблица компаний изображена на рисунке 3.1.2.

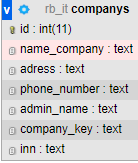


Рисунок 3.1.2. Изображение сущности компании

Таблица users предназначена для хранения всех типов пользователей, зарегистрированных в системе, запись об администраторе сохраняется при регистрации компании, запись о сотруднике компании сохраняется при регистрации администратором личного кабинета сотрудника. Каждый пользователь системы обладает следующими полями: Уникальный номер записи(id), Уникальный номер компании(id\_company), Имя(name), Пароль(pass), Логин(login), Дата рождения(birth), Роль(rule), Текстовое обозначение роли в компании(text\_rule). Таблица пользователи изображена на рисунке 3.1.3.

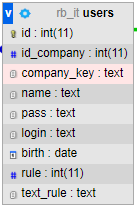


Рисунок 3.1.3. Изображение сущности пользователи

Таблица tasks предназначена для хранения информации о задачах. Таблица состоит из следующих полей: Уникальный номер записи(id), идентификатор пользователя поставивший задачу(from\_user\_id), Уникальный идентификатор пользователя ответственный за выполнение(to\_user\_id), Заголовок задачи(title), Описание задачи(description), Дата начала задачи(date\_start), Крайний срок задачи(dead\_line), Статус задачи(task\_status), Дата начала выполнения задачи(started\_btn), Время выполнения задачи(visited\_data). Таблица задач изображена на рисунке 3.1.4.

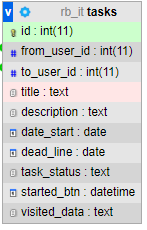


Рисунок 3.1.4. Изображение сущности задачи

3.2. Разработка программного интерфейса приложения

Разрабатывая web-приложение в первую очередь необходимо реализовать клиент-серверное взаимодействие самого web-приложения с базой данных. Основные задачи программного интерфейса это: регистрация пользователей, компаний в системе, отправка и получение данных.

Входные данные:

* Сервер - Apache – 2.4, MySQL/MariaDB - 10.4, PHP 7.4
* Клиент – Любой web-браузер

Для запросов к серверу и ответов от него со стороны клиента будет использоваться JSON формат данных, так как формат легок в использовании, широко поддерживается в PHP. Запросы будут отсылаться через POST запросы, такое решение позволит тестировать API в любом доступном браузере.

Внешний вид запросов будет иметь следующий вид:

**http:// [адрес сервера] / [путь к папке api]/? [название\_api] . [название\_метода]**

**[JSON вида {«Hello»:«Hello world»}]**

Программный интерфейс приложения состоит из 3 PHP файлов:

* Api.php
* Functions.php
* Index.php

Файл Api.php описывает настройки программного интерфейса, доступ к подключению к api(Листинг 3.2.1).

Листинг. 3.2.1. Файл настройки api интерфейса

|  |
| --- |
| <?php  header("Access-Control-Allow-Origin: \*");  header("Access-Control-Allow-Methods: GET, POST, PUT");  header("Access-Control-Allow-Headers: Content-Type");  ?> |

Файл Functions.php является классом реализующий функции по формировании данных для дальнейшей отправки в базу данных. Передачи информации из базы данных обратно в клиент в формате JSON. В листинге 3.2.2. приведен фрагмент класса с функциями удаления, добавления поля в таблицу базы данных, а также функцией по созданию записи о новом пользователе.

Листинг. 3.2.2. Фрагмент кода класса Functions.php

|  |
| --- |
| <?php  class Functions  {  function deleteRowById($id, $tablename)  {  //DELETE FROM `cities` WHERE `id` = 13  R::exec('DELETE FROM ' . $tablename . ' WHERE id = ' . $id);  $table = R::getAll('SELECT \* FROM ' . $tablename);  return $table;  }  function insertIntoTable($tablename, $formData)  {  $table = R::dispense($tablename);  // Заполняем объект свойствами  //var\_dump($table);  foreach ($formData as $key => $value) {  print "$key => $value\n";  $table->$key = $value;  }  R::store($table);  }  function createUser($userData, $companyData)  {  $table = R::dispense('users');  $checkUser = R::findOne('users', 'login = ?', [$userData->login]);  if (!empty($checkUser)) {  return "001";  } else {  $userData->id\_company = $companyData->id;  $userData->company\_key = $companyData->company\_key;  // Заполняем объект свойствами  $userData->rule = 100;  $userData->pass = password\_hash($userData->pass, PASSWORD\_DEFAULT);  foreach ($userData as $key => $value) {  print "$key => $value\n";  $table->$key = $value;  }  R::store($table);  }  }  function updateTableRow($tablename, $rowId, $formData)  {  $row = R::load($tablename, $rowId);  // Заполняем объект свойствами  if($formData->pass){  $formData->pass = password\_hash($formData->pass, PASSWORD\_DEFAULT);  }  foreach ($formData as $key => $value) {  print "$key => $value\n";  $row->$key = $value;  }  R::store($row);  } |

Файл index.php является основным файлом программного интерфейса взаимодействия клиентской и серверной части приложения, включает в себя подключение базы данных, ORM RedBeanPhp и является обработчиком запросов со стороны клиента. В зависимости от указанного метода обработчик будет вызывать нужную клиенту функцию из класса Function.php. Фрагмент кода обработчика post запросов представлен в листинге 3.2.3.

Листинг. 3.2.3. Обработчик post запросов

|  |
| --- |
| <?php  include "api.php";  include "functions.php";  require "../rb/rb.php";  R::setup('mysql:host=localhost;dbname=rb\_it', 'root', 'root'); //for both mysql or mariaDB//rb\_pharmacy  R::freeze(true);  R::useJSONFeatures(TRUE);  header('Content-type: json/aplication');  if (!R::testConnection()) {  echo 'нет подключения';  }  $recivedData = json\_decode(file\_get\_contents("php://input"));  $func = new Functions;  if ($recivedData->action == 'getTableNames') {  $tables = R::getAll('SHOW TABLES');  foreach ($tables as $table) {  $data[] = $table;  }  echo json\_encode(($data));  }  if ($recivedData->action == 'deleteRowById') {  $fromVueParamId = $recivedData->selectedId;  $fromVueParamTableName = $recivedData->selectedTableName;  ///$rows = deleteRowById($fromVueParamId, $fromVueParamTableName);  $rows = $func->deleteRowById($fromVueParamId, $fromVueParamTableName);  foreach ($rows as $row) {  $data[] = $row;  }  echo json\_encode(($data));  }  if ($recivedData->action == 'insertIntoTable') {  $fromVueParamTableName = $recivedData->selectedTableName;  $fromVueInsertData = $recivedData->insertInto;  echo $func->insertIntoTable($fromVueParamTableName, $fromVueInsertData);  }  if ($recivedData->action == 'updateTableRow') {  $fromVueParamTableName = $recivedData->selectedTableName;  $fromVueInsertData = $recivedData->insertInto;  $fromVueParamTableName = $recivedData->selectedTableName;  $fromVueRowId = $recivedData->rowId;  $fromVueUpdateData = $recivedData->updateData;  echo $func->updateTableRow($fromVueParamTableName, $fromVueRowId, $fromVueUpdateData);  } |

3.3. Разработка системы переходов и навигации по страницам

Для того чтобы в проекте работали переходы между страницами необходимо реализовать маршрутизацию. На стороне клиента в VueJS за навигацию отвечает библиотека Vue-router.

Для реализации маршрутизатора необходимо создать отдельную папку и объявить в ней файл типа javaScript, затем в самом файле подключить библиотеку vue-router. Далее необходимо подключить созданный файл в корень приложения. В листинге 3.3.1. приведен полный код исполняемого маршрутизатора, используемый системой.

Листинг 3.3.1. Маршрутизатор

|  |
| --- |
| import { createRouter, createWebHistory } from "vue-router";  import Home from '../views/home.vue'  import Auth from '../views/auth.vue'  import Reg from '../views/regist.vue'  import Profile from '../views/profile.vue'  import Tasks from '../views/user.vue'  import store from "../store.js";  import adminLayout from "../layouts/adminLayout";  import userLayout from "../layouts/userLayout";  let routes;  // let getterRule = store.getters.getUserRule;  console.log(store.state.userRule);  routes = [  {  path: '/',  name: 'auth',  component: Auth,    },  {  path: '/regist',  name: 'regist',  component: Reg,  },  {  path: '/home',  name: 'Home',  component: adminLayout,  children: [  {  path: '',  name: 'admHome',  component: Home,  beforeEnter: (to, from, next) => {  if (store.state.userRule == '900') {  next()  }else{  next(from)  }  }  },  {  path: '/profile',  name: 'profileAdm',  component: Profile,  beforeEnter: (to, from, next) => {  if (store.state.userRule == '900') {  next()  }else{  next(from)  }  }  },  {  path: '/tasks',  name: 'AdminTasks',  component: Tasks,  beforeEnter: (to, from, next) => {  if (store.state.userRule == '900') {  next()  }else{  next(from)  }  }  },  ],  beforeEnter: (to, from, next) => {  if (store.state.userRule == '900') {  next()  } else {  next({ name: 'user' })    }  }  },  {  path: '/user',  name: 'user',  component: userLayout,  children: [  {  path: '',  name: 'userTasks',  component: Tasks,  beforeEnter: (to, from, next) => {  if (store.state.userRule == '100') {  next()  }else{  next(from)  }  }    },  {  path: '/userProfile',  name: 'userProfile',  component: Profile,  beforeEnter: (to, from, next) => {  if (store.state.userRule == '100') {  next()  }else{  next(from)  }  }    },  ],  }  ]  const router = createRouter({  history: createWebHistory(process.env.BASE\_URL),  routes  })  export default router |

Свойство routes - это массив, который будет содержать объекты, каждый из которых будет конкретным маршрутом для web-приложения.

Vue-Router принимает множество свойств для маршрутизируемых объектов:

* path : это относительный путь к базовому маршруту вашего веб-приложения, на котором вы хотите, чтобы определенный компонент отображался.
* name : свойство name будет полезно при ссылке на этот конкретный объект внутри ваших компонентов Vue.
* component : с этим свойством компонент Vue будет вставлен в назначенный вами путь маршрута.
* children : это массив с большим количеством маршрутов, которые будут объединены с путем, при этом родительский путь останется в URL-адресе.

Маршрутизатор в данном примере загружает компоненты как только получает нужный pacth при переходе по ссылке, перед каждым переходом маршрутизатор осуществляет проверку befireEnter где проверяется условие роли пользователя, проверка роли пользователя перед переходом исключает возможность доступа сотрудника к компонентам администратора и наоборот.

Среди компонентов следует выделить:

* adminLayout – оберточный компонент для администратора содержит в себе шапку сайта с навигацией по страницам доступным только администратору.
* userLayout – оберточный компонент сотрудника содержит в себе шапку сайта с навигацией по страницам доступным только сотруднику.
* Home – главная страница администратора, будет загружена если свойство пользователя Роль(userRule) будет равно 900.
* Tasks – страница содержащая инструменты для работы пользователя с задачами, является домашней страницей сотрудника, так же доступна администратору.
* Profile – страница отображает информацию о пользователе, дает возможность изменять информацию о пользователе.
* Auth – страница с формой авторизации
* Reg – страница с формой регистрации.

3.4. Реализация регистрации компании и администратора

Регистрация в системе осуществляется за счет обращения к форме регистрации в 2 этапа: регистрация компании, регистрация администратора.

Первый этап регистрации — заполнение формы данными о компании необходимыми для создания организации: Имя компании, Адрес компании, Телефон Компании, Инн.

Второй этап регистрации – регистрация администратора, наступает если заполнены все поля формы регистрации компании, для успешной регистрации главы компании необходимо заполнит поля: Имя администратора, логин в системе применяемый при авторизации, Пароль (повторяется дважды), дата рождения.

После заполнения всех полей формы регистрации и нажатия на кнопку «Создать Компанию» данные будут отправлены в API в функцию регистрации где будут проверены на: совпадения пароля, наличие существующей компании по имени, наличие пользователя с таким де логином. Если данные проходят проверку, то они сохраняются в базу данных, а пользователю возвращают сообщение об успешной регистрации. На рисунке 3.4.1. изображена логика работы формы регистрации.

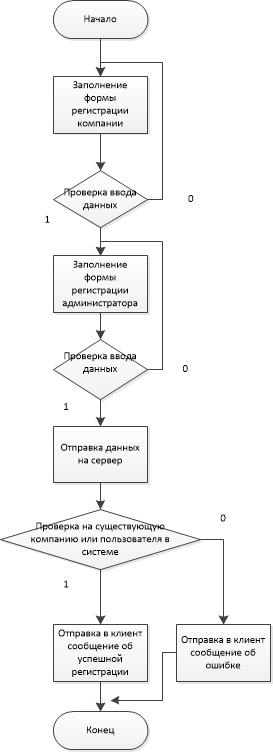


Рисунок. 3.4.1. Логика формы регистрации

Форма регистрации в структуре VueJS проекта выглядит как исполняемый блок кода вызываемый внутри компонента router-vue при переходе на страницу регистрации. Сама форма представляет из себя блок HTML-разметки класса bootstrap form. Данные с формы привязаны к переменным VueJS через компонент v-model[Приложение 1., Файл regist.vue].

Внутри страницы так же располагается скрипт, обрабатывающий все действия в компоненте (Листинг. 3.4.2). В основе способа отправки данных в API лежит библиотека AXIOS позволяющая передавать данные через POST запросы в функцию отправки данных на сервере (Листинг. 3.4.3). В таблице 3.4.4. приведено описание всех функций выполняемые на стороне клиента в VueJS компонента регистрации пользователя.

Листинг. 3.4.2. Скрипт VueJS отвечающий за страницу регистрации

|  |
| --- |
| <script>  methods: {  companyDataValidate() {  if (  this.formReg.company.name\_company &&  this.formReg.company.adress &&  this.formReg.company.phone\_number &&  this.formReg.company.inn  ) {  this.formState++;  }  },  isValidate() {  return (  this.formReg.company.name\_company &&  this.formReg.company.adress &&  this.formReg.company.phone\_number &&  this.formReg.company.inn &&  this.formReg.admin.name &&  this.formReg.admin.pass &&  this.formReg.admin.pass2 &&  this.formReg.admin.login &&  this.formReg.admin.birth  );  },  registerUser() {  if (this.isValidate()) {  console.log("data from regist form", this.formReg);  this.sendDataToApi();  }  },  errorAlert(errorCode) {  if (errorCode == "001") {  alert("Компания уже существует или Логин уже занят");  }else if(errorCode == "002"){  alert("Пароли не совпадают");  }else{  alert("Компания создана");  }  //to do иначе придет обект его передать во vuex  },  sendDataToApi() {  axios  .post("http://apiks/api/", {  action: "registCompanyAndAdmCompany",  companyData: this.formReg.company,  adminData: this.formReg.admin,  })  .then(  (response) => (  (this.errorMsg = response.data),  console.log(response),  this.errorAlert(this.errorMsg)  )  );  },  },  };  </script> |

Листинг 3.4.3. Функция registCompanyAndAdmCompany отвечающая за обработку данных отправленных их клиента

|  |
| --- |
| function registCompanyAndAdmCompany($companyData, $adminData)  {  //to do получить дату и отправить в базу данных информацию о компании и ее созадтеле  //создать записть о создателе как админе, его дату обработать(шифр пароля)  //$companyR = R::dispense("about\_company");  $nameCompany = $companyData->name\_company;  $admLogin = $adminData->login;  $checkCompany = R::findOne('companys', 'name\_company = ?', [$nameCompany]);  $checkAdm = R::findOne('users', 'login = ?', [$admLogin]);  $func = new Functions;  //$errorMSG = "001";  if (!empty($checkCompany) || !empty($checkAdm)) {  return "001";  } else if ($adminData->pass != $adminData->pass2) {  return "002";  } else {  //to do зарегать компанию и админа  //удаление проверочного пароля из объекта  unset($adminData->pass2);  //добавление к записи компании имя автора(администратора)  $uniqIdCompany = $nameCompany . uniqid();  $companyData->company\_key = $uniqIdCompany;  $companyData->admin\_name = $adminData->name;  $func->insertIntoTable('companys', $companyData);  //добавление к записи юзера id компании  $findNewComp = R::findOne('companys', 'name\_company = ?', [$nameCompany]);  $findIdComp = $findNewComp->id;  $adminData->company\_key = $uniqIdCompany;  $adminData->id\_company = $findIdComp;  //добавление к записи юзера его роль  $adminData->rule = 900;  $adminData->text\_rule = 'Администратор';  $adminData->pass = password\_hash($adminData->pass, PASSWORD\_DEFAULT);  $func->insertIntoTable("users", $adminData);  }  } |

Таблица. 3.4.4. Функции страницы регистрации и их описание

|  |  |
| --- | --- |
| Название Функции | Описание |
| companyDataValidate() | Проверяет заполнение формы регистрации компании |
| isValidate() | Проверяет заполнение формы регистрации администратора |
| registerUser() | Вызывает функцию sendDataToApi() если все поля форм были заполнены |
| errorAlert() | Функция принимает в себя строку – ответ полученный от API и если это ошибка выведет пользователю уведомление об ошибке, иначе сообщение об успешной регистрации |
| sendDataToApi() | Функция реализующая отправку данных с формы регистрации в API, принимает ответ от сервера и возвращает его в функцию errorAlert() |

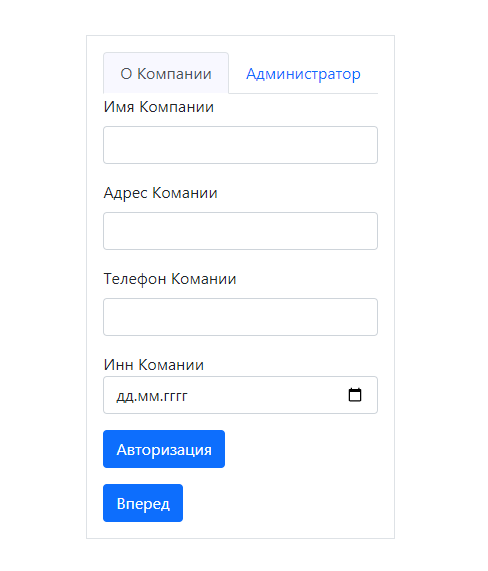


Рисунок.3.4.5. Регистрация компании

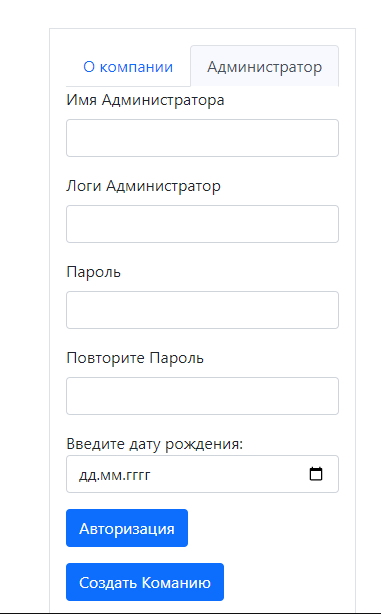


Рисунок.3.4.6. Регистрация администратора

3.5. Реализация авторизации пользователей в системе

Авторизация в системе происходит за счет обращения к форме авторизации. Форма состоит из двух полей: Логин, Пароль, а также кнопки авторизации. Если пользователь заполнил все поля и нажал на кнопку система будет проверять наличие введенного пользователя по логину и пароля и в зависимости от ответа сервера либо перейдет на свою домашнюю страницу, либо получит сообщение об ошибке.

Страница авторизации является компонентом VueJs и открывается только когда маршрутизатор получает ссылку на страницу авторизации. Страница состоит из HTML-разметки и VueJS скрипта проверяющий наличие пользователя в системе за счет отправки данных в API и получения ответа с сервера [Приложение 1. Файл auth.vue]. Листинг 3.5.3. показывает зависимую функцию на стороне сервера. В таблице 3.5.4. подробно описаны функции страницы авторизации.

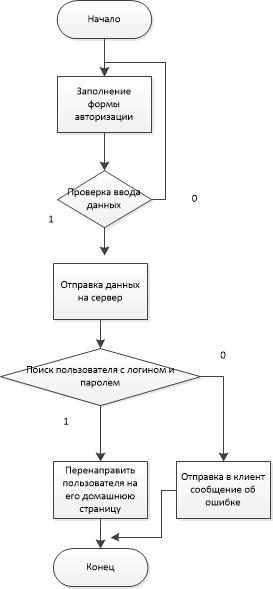


Рисунок. 3.5.1. Логика работы авторизации

Листинг. 3.5.2. Скрипт VueJS отвечающий за страницу авторизации

|  |
| --- |
| methods: {  isValidate() {  return this.formAuth.userLogin && this.formAuth.userPass;  },  checkResponsForErrorOrData(errorCode) {  if (errorCode == 0) {  alert("Неверный логин или пароль");  } else {  //запись данных об пользователев store  this.$store.dispatch("setUserDataAction", this.responseUserData);  this.$store.dispatch("setUserRuleAction", this.responseUserData);  //запись данных о компании пользователя в store  this.$store.dispatch('setCompanyDataAction',this.responseCompanyData)  //this.$store.commit("setUserData", this.responseUserData);  this.$store.commit("setAuth", true);  this.$router.push("/home");  //console.log(this.$store.state.userRule);  //подмена router  }  },  aithorizationUser() {  if (this.isValidate()) {  this.findUserAndAuth();  }  },  findUserAndAuth() {  axios  .post("http://apiks/api/", {  action: "authorizationUser",  authForm: this.formAuth,  })  .then(  (response) => (  ((this.responseUserData = response.data.userInfo),  (this.responseCompanyData = response.data.companyInfo)),  console.log(this.responseUserData),  //вызвать функцию проверки даты  this.checkResponsForErrorOrData(this.responseUserData)  //  )  );  },  },  };  </script> |

Листинг. 3.5.3. Функция обработки данных авторизации на сервере

|  |
| --- |
| function chekUserAndAuthorizationUser($authFormData)  {  //to do найти пользователя по логину и паролю  $userLogin = $authFormData->userLogin;  $checkUser = R::findOne('users', 'login = ?', [$userLogin]);  $checkUserCompany = R::findOne('companys', 'id = ?', [$checkUser->id\_company]);  if (!empty($checkUser)) {  $hashPass = $checkUser->pass;  $verifyPwd = password\_verify($authFormData->userPass, $hashPass);  if ($verifyPwd) {  //создать объект с информацией о пользователе и о его компании  $array1 = array($checkUser, $checkUserCompany);  return $array1;  } else {  return "003";  }  } else {  return "003";  }  } |

Таблица. 3.5.4. Описание функция страницы авторизации

|  |  |
| --- | --- |
| Название Функции | Описание |
| isValidate() | Проверяет заполнение формы авторизации |
| checkResponsForErrorOrData() | Аргументом принимает строку и если она является кодом ошибки то выводи сообщение о неверном логине или пароле, иначе производит переход на домашнюю страницу пользователя |
| aithorizationUser() | Если функция isValidate() возвращает true, вызывает функцию findUserAndAuth() |
| findUserAndAuth() | Передает данные в Api где проверяется существование пользователя с логином и паролем указанным в форме, возвращает ответ в функцию checkResponsForErrorOrData() |

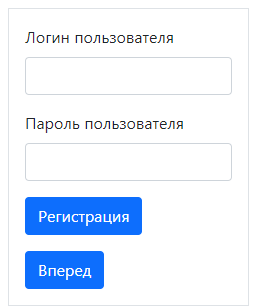


Рисунок. 3.5.5. Авторизация

3.6. Реализация личного профиля пользователя

Каждый пользователь, успешно авторизовавшийся в системе имеет возможность изменять данные о себе, для этого ему необходимо перейти в личный профиль. Профиль пользователя реализован в виде отдельной страницы состоящий из компонента реализующий функционал смены имени, логина и пароля пользователя [Приложение 1. Файл profile.vue].При заполнении полей формы и нажатии на кнопку «Изменить» срабатывает функция changeUserData() (Листинг.3.6.2.) которая отправляет данные с формы в функцию API на сервере (Листинг.3.6.3.). Таблица 3.6.4 описывает функции работы профиля пользователя.

Листинг. 3.6.2. VueJS код профиля пользователя

|  |
| --- |
| <script>  methods: {  getUserData() {  this.profileData = this.$store.getters.getAuthUserData;  console.log(this.$store.state.userData);  },  changeUserData() {  console.log(this.profileData);  axios  .post("http://apiks/api/", {  action: "updateTableRow",  selectedTableName: "users",  rowId: this.profileData.id,  updateData: this.profileData,  })  .then(  (response) => console.log(response),  this.$store.commit("setUserData", this.response)  );  this.$store.dispatch('setChangeUserRegistData',this.profileData)  this.getUserData();  },  },  };  </script> |

Листинг. 3.6.3. Функция обновления записи в базе данных

|  |
| --- |
| function updateTableRow($tablename, $rowId, $formData)  {  $row = R::load($tablename, $rowId);  // Заполняем объект свойствами  if($formData->pass){  $formData->pass = password\_hash($formData->pass, PASSWORD\_DEFAULT);  }  foreach ($formData as $key => $value) {  print "$key => $value\n";  $row->$key = $value;  }  R::store($row);  } |

Таблица. 3.6.4. Описание функций компонента профиля пользователя

|  |  |
| --- | --- |
| Название Функции | Описание |
| getUserData () | Сохраняет данные в форму изменения личной информации из глобального хранилища |
| changeUserData() | Отправляет данные из формы изменения личных данных пользователя в api функцию обновления строки в таблице пользователей |

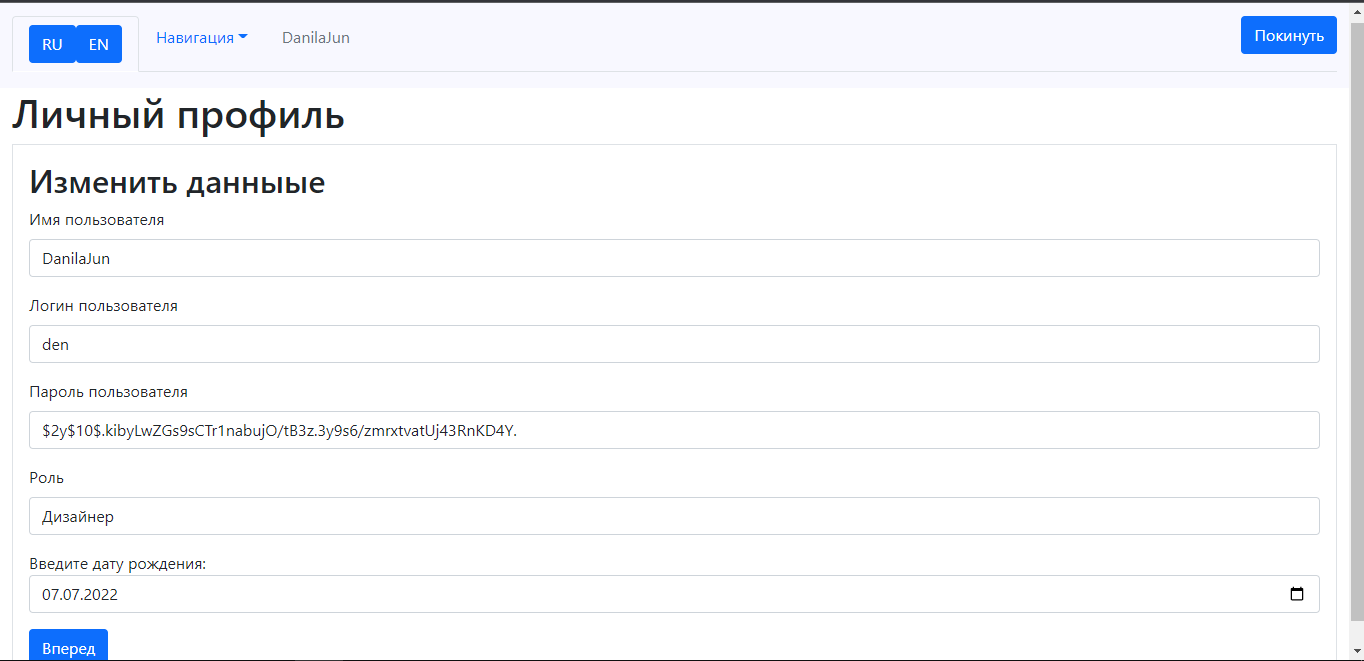


Рисунок. 3.6.5. Личный профиль пользователя

3.7 Реализация инструментов администрирования информации в компании

Вместе с регистрацией компании в системе регистрируется, и пользователь с ролью администратора связанный с созданной компанией за счет поля id\_company.

В обязанности администратора компании входит: добавление новых пользователей в компанию за счет создания личных кабинетов, редактирование данных сотрудников, удаление сотрудников изменения информации о компании. Базовый функционал роли администратора расширяется инструментами для работы с задачами: постановка задачи, редактирование поставленных задач, изменение данных в личном профиле.

Главная страница администратора состоит из:

* Блока управления информацией о компании
* Блока управления пользователями

Блок управления пользователями включает в себя функционал по добавлению новых сотрудников в компанию реализованный в виде компонента VueJS. Компонент становится виден при выборе опции «Пользователи» на главной странице администратора и выполнен в виде кнопки при нажатии на которую происходит вызов формы добавления сотрудника, форма содержит следующие поля: Имя, Логин и Пароль, указываемые при авторизации, Дата рождения, Текстовая запись роли в компании.

Перед созданием нового личного кабинета пользователя происходит проверка на заполнение всех полей формы, проверка на наличие в системе пользователя с таким же логином. Если ошибок не обнаружено, данные отправляются на сервер в Api где сохраняются в базу данных. Логика работы формы добавления нового сотрудника приведена на рисунке 3.7.1. HTML-разметка компонента [Приложение 1. Файл addUserBtn.vue].

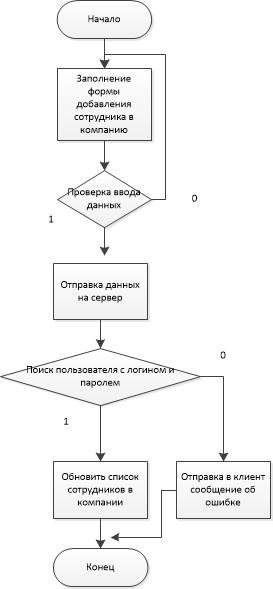


Рисунок. 3.7.1. Логика компонента добавление сотрудника в компанию

Листинг 3.7.2 Скрипт VueJS отвечающий за компонент добавления сотрудников в компанию

|  |
| --- |
| <script>  methods: {  checkResponsForErrorOrData(errorCode) {  if (errorCode == "001") {  alert("Пользователь уже существует");  } else {  alert("Пользователь создан");  this.$refs.Close.click();  this.$emit("reloadTableEmit")  }  },  addRow(formData) {  //ролучил данные из формы  console.log(formData);  // console.log(this.$store.getters.getUserCompanyData);  // //to do отправить запрос на добавление вернуть отвте ошибку или успех  if (  !this.data.formDataList.name ||  !this.data.formDataList.login ||  !this.data.formDataList.pass ||  !this.data.formDataList.birth ||  !this.data.formDataList.text\_rule  ) {  alert("Заполните Форму");  } else if(!/^[a-zA-Z]\*$/.test(this.data.formDataList.login)){  alert("Логин не формат");  }  else {  axios  .post("http://apiks/api/", {  action: "createUser",  newUserData: formData,  companyData: this.$store.getters.getUserCompanyData,  })  .then((response) => this.checkResponsForErrorOrData(response.data));  //перезагрузить таблицу  //обнулить поля формы  this.formDataList = {};  }  },  },  };  </script> |

Листинг 3.7.3. Функция по добавлению данных о новом сотруднике в базу данных

|  |
| --- |
| function createUser($userData, $companyData)  {  $table = R::dispense('users');  $checkUser = R::findOne('users', 'login = ?', [$userData->login]);  if (!empty($checkUser)) {  return "001";  } else {  $userData->id\_company = $companyData->id;  $userData->company\_key = $companyData->company\_key;  $userData->rule = 100;  $userData->pass = password\_hash($userData->pass, PASSWORD\_DEFAULT);  foreach ($userData as $key => $value) {  print "$key => $value\n";  $table->$key = $value;  }  R::store($table);  }  } |

В таблице 3.7.4. приведено описание функций, используемых в листинге 3.7.2.

Таблица. 3.7.4. Описание функция страницы авторизации

|  |  |
| --- | --- |
| Название Функции | Описание |
| addRow() | Функция принимает аргументом объект данных с формы создания нового сотрудника и отправляет ее на сервер где проверяется на существование пользователя с таким же логином, если пользователь уже существует то в клиент возвращается ошибка иначе данные сохраняются в базу данных и в клиент возвращается сообщение об успешном добавлении |
| checkResponsForErrorOrData() | Принимает аргументом строку содержащую код ошибки, возвращает уведомление о том что пользователь уже существует или уведомляет администратора об успешном создании личного кабинета |

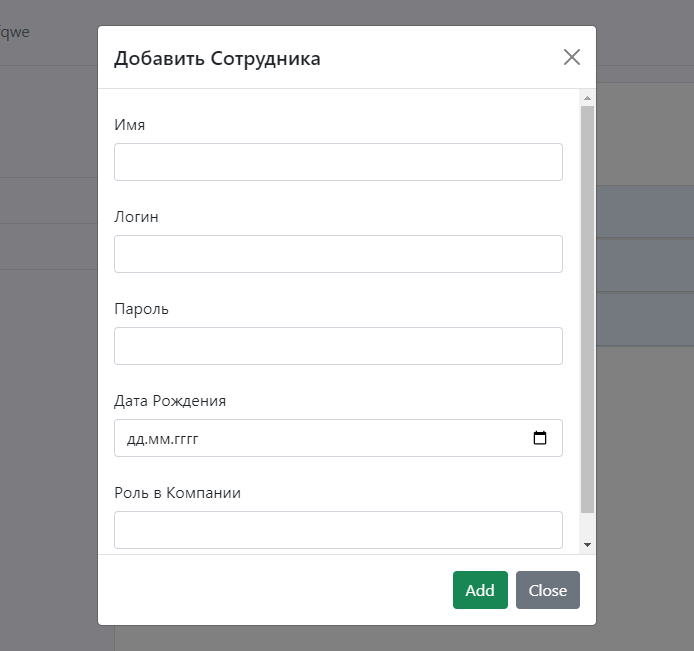


Рисунок. 3.7.5. Добавление нового пользователя

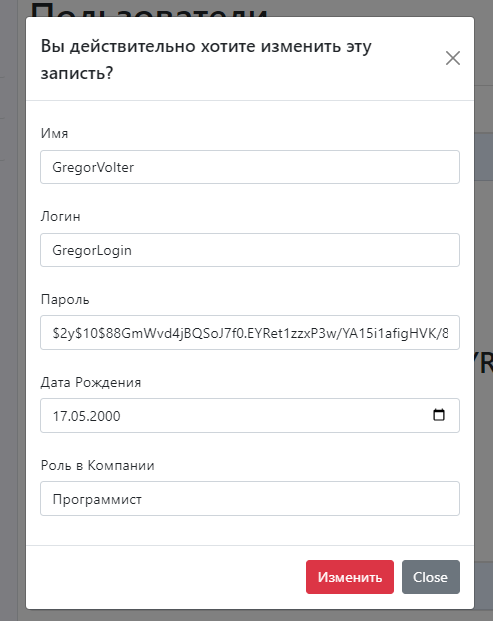


Рисунок 3.7.6. Редактирование данных о сотруднике

Блок управления информацией о компании включает в себя форму изменения данных о компании указанных в момент регистрации. Форма выполнена в виде отдельного компонента VueJS который подключается к главной странице администратора и вызывается только когда выбран пункт меню главной страницы «О компании». Форма изначально хранит данные о текущей компании и изменяет их, как только администратором будет нажата кнопка «Изменить». Данные отправятся на сервер в функцию обновления строки в таблице companys по id компании. В листинге представлена HTML-разметка компонента блока управления информации, относящейся к компании [Приложение 1. Файл aboutCompany.vue]. Листинг 3.7.7. отражает скрипт взаимодействия с компонентом. Таблица 3.7.8. поясняет работу функций, использующихся в листинге 3.7.7.

Листинг 3.7.6 Скрипт VueJS отвечающий за компонент изменения данных компании

|  |
| --- |
| <script>  methods: {  setCompanyData(){  this.data.myCompanyData = this.$store.getters.getUserCompanyData;  },  editRow(formData) {  axios  .post("http://apiks/api/", {  action: "updateTableRow",  selectedTableName: 'companys',  rowId: formData.id,  updateData: formData,  })  .then(  (response) => (  console.log(response),  //перезагрузить таблицу  this.$emit("reloadTableEmit")  )  );  },  },  };  </script> |

Таблица. 3.7.7. Описание функций компонента изменения данных компании

|  |  |
| --- | --- |
| Название Функции | Описание |
| setCompanyData() | Функция сохраняет в форму данные сохраненные в глобальном хранилище |
| editRow() | Функция принимает аргументом объект хранящий информацию из формы, отправляет данные из формы в Api в функцию обновления строки |

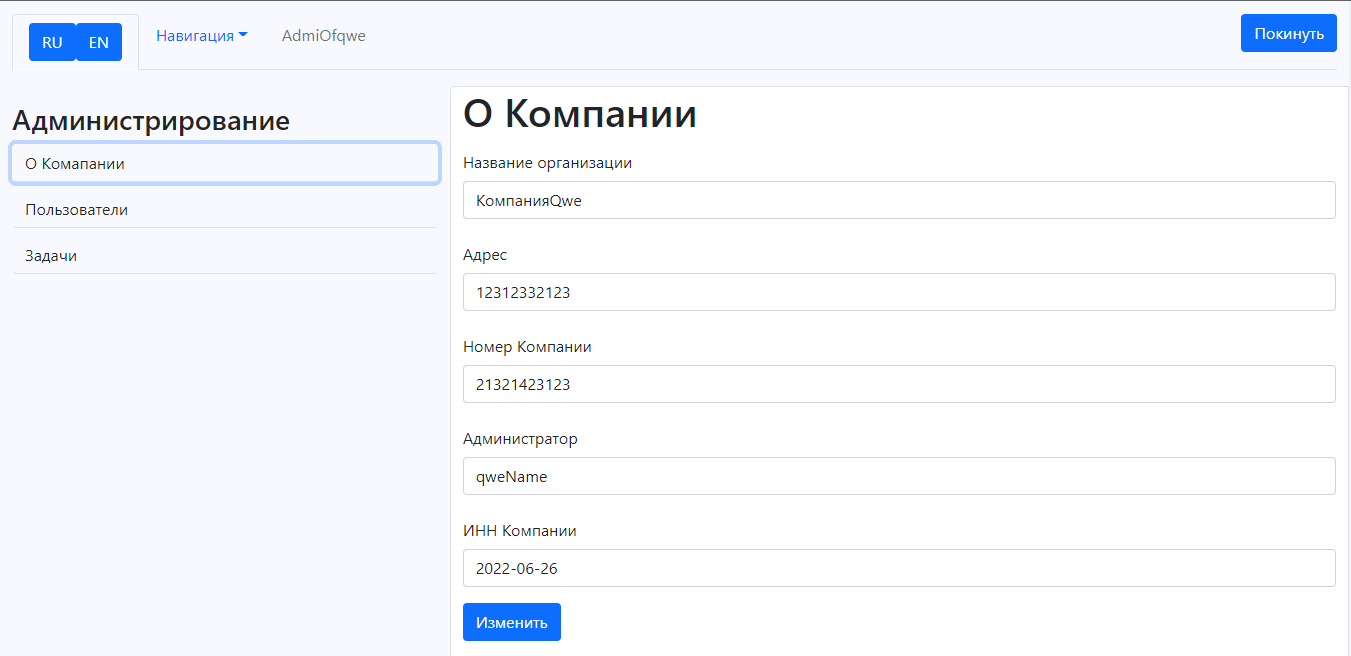


Рисунок. 3.7.7б. Редактирование данных о компании

3.8. Реализация инструментов работы с задачами

Работа с задачами основной инструмент менеджера задач, доступ к нему имеют все пользователи, прошедшие авторизацию. Весь функционал работы с задачами реализован в виде страницы состоящая из элементов навигации выбор которых определяющих какой тип задачи необходимо отобразить.

Существует 3 типа задач:

* Мои задачи – задачи, поставленные мне как сотруднику;
* Поставленные задачи – задачи которые я как сотрудник назначил кому – то, есть возможность удалить или изменить описание, заголовок задачи, крайний срок и ответственного;
* Выполненные задачи – история выполненных задач, хранящая информацию по задачам и времени их выполнения;

Страница реализована в виде HTML-разметки с применением особенности технологии фреймворка VueJS, а именно «условной отрисовки». Каждому типу задач соответствует своя кнопка бокового меню и при нажатии на которую изменяется переменная отвечающая за отрисовку блок той или иной задачи (Листинг. 3.8.1). Директива v-if реализует рендеринг блока по условию, в данном случае условием является значение переменной activeItem. Таким образом если значение переменной VueJs на странице работы с задачами будет равно нулю, то будет отображаться блок «Мои задачи», значение равное 1 загрузит блок «Поставленные задачи», а значение равное 2 выведет список выполненных задач.

Листинг 3.8.1 Условная отрисовка по нажатию кнопки «Мои задачи»

|  |
| --- |
| <div class="bd-tables-container py-3">  <h3>Задачи</h3>  <div class="d-grid gap-2">  <button type="button" class="btn border-bottom" @click="getMyTasks">  Мои Задачи  </button>  <button  type="button"  class="btn border-bottom"  @click="getPostedTasks"  >  Поставленные задачи  </button>  <button  type="button"  class="btn border-bottom"  @click="getComplitedTasks"  >  Выполненные задачи  </button>  </div>  </div> |

Блок мои задачи состоит из блоков HTML-разметки, представляющие список задач. Каждая задача выражается в виде элемента Accorion библиотеки Bootstrap, это удобное представление позволяет получать более детальную информацию о задаче нажав на нее.

При нажатии на задачу выпадает окно с детальным описанием задачи, временем выполнения и кнопками взаимодействия. Все задачи, адресованные сотруднику, выводятся через цикл v-for внутри блока элемента задачи [Приложение 1. Файл addTaskBtn]. Список всех задач, требующих выполнения предоставляет Api функция при загрузке блока «Мои задачи» (Листинг. 3.8.3). В таблице 3.8.4. Описаны функции работающие с блоком «Мои задачи»

Листинг. 3.8.3.PHP код функции возвращающий список моих задач

|  |
| --- |
| function getMyTasks($userId){  //вернуть задачи где to\_user\_id == авторизированный пользователь  $users = R::getAll("SELECT \* FROM tasks WHERE to\_user\_id = ". $userId . " AND task\_status != 'closed'");  return $users;  } |

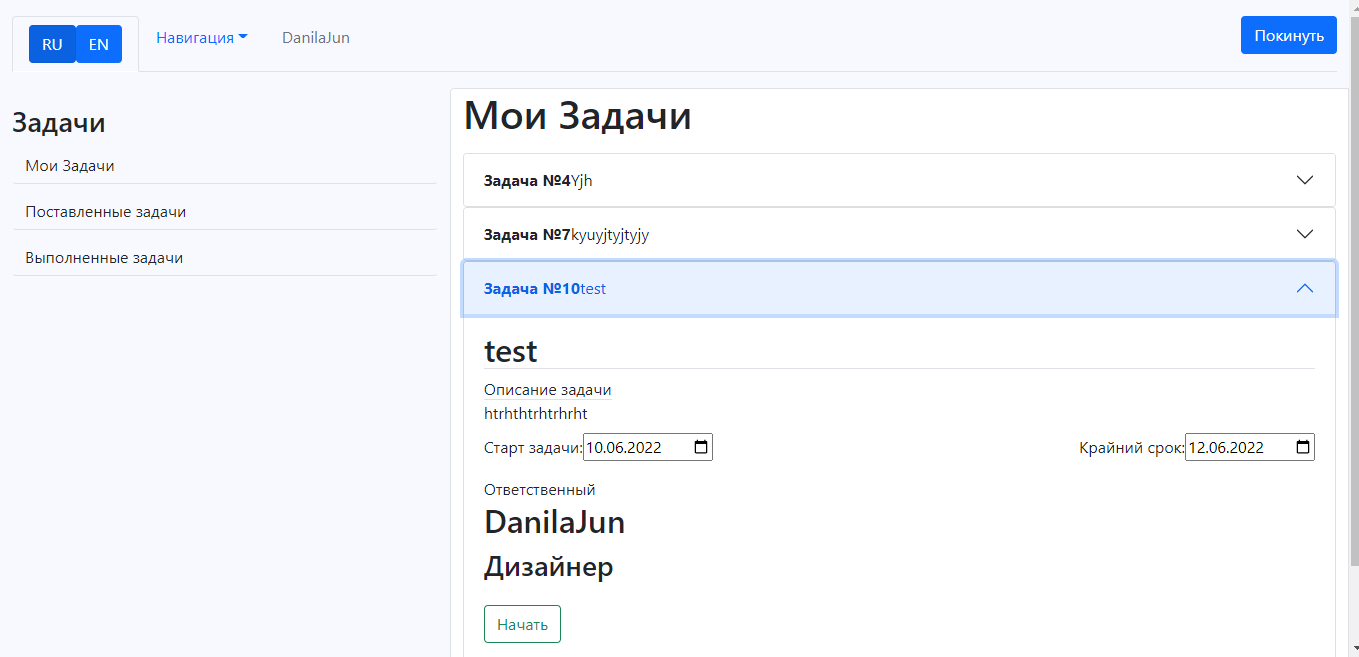


Рисунок. 3.8.3б. Мои задачи

Рисунок 3.8.3в. Выполнение задачи

Блок «Поставленные задачи» отличается от блока «Мои задачи» наличием компонента, реализующего форму добавления новой задачи, наличием компонентов, реализующих удаление и редактирования данных о созданной задачи [Приложение 1. Файл tasks.vue]

Приведенный листинг выше содержит в себе такие компоненты как:

* addModal – компонент вызывающий модальное окно создания новой задачи с полями: Заголовок задачи, Описание, Крайний срок, Ответственный
* deliteModal – компонент выполняющий функцию модального окна предупреждающего об удалении задачи
* editModal – компонент вызывающий форму редактирования выбранной задачи

Компонент addModal представляет собой кнопку «Создать» при нажатии на которую произойдёт вызов модального окна формы создания новой задачи. (Листинг. 3.8.5.). После заполнения формы и нажатии на кнопку модального окна «Создать» произойдёт вызов функции VueJS AddRow (Листинг. 3.8.6.) которая отправит данные с формы в api функцию createNewTask где произойдёт сохранение в базу данных [Приложение 1. Файл addTaskBtn].

Листинг.3.8.6. VueJS code компонента addModal

|  |
| --- |
| <script>    methods: {  close(){  this.$emit("reloadTableEmit")  },  saveData(param) {  this.data.companyUsers = param;  console.log(this.data.companyUsers);  },  getCompanyUsers() {  //console.log("Получаю список пользователей");  axios  .post("http://apiks/api/", {  action: "getCompanyUsers",  companyId: this.$store.getters.getUserCompanyData.id,  })  .then((response) => this.saveData(response.data));  },  addRow(formData) {  console.log(formData);  if (  !this.data.formDataList.title ||  !this.data.formDataList.description ||  !this.data.formDataList.date\_start ||  !this.data.formDataList.dead\_line ||  !this.data.formDataList.to\_user\_id  ) {  alert("Логин не формат");  } else {  console.log("newTaskFormData",this.data.formDataList);  axios  .post("http://apiks/api/", {  action: "createNewTask",  newTaskData:formData,  fromUser: this.$store.getters.getAuthUserData.id,  })  .then((response) => console.log(response.data));  //перезагрузить таблицу  //обнулить поля формы  this.data.formDataList = {};  this.$emit("reloadTableEmit");  this.$refs.Close.click();  }  },  },  };  </script> |

Листинг.3.8.7. PHP код функции добавления новой задачи

|  |
| --- |
| function createNewTask($taskData,$fromUserId)  {  $func = new Functions;  $taskData->from\_user\_id = $fromUserId;  $taskData->task\_status = "created";  $func->insertIntoTable("tasks", $taskData);  } |

Таблица. 3.8.8. Поясняющая работу функций компонента addModal

|  |  |
| --- | --- |
| Название Функции | Описание |
| close () | Функция вызвает перезагрузку страницы |
| saveData() | Сохраняет пользователей компании в список для возможности выбора ответственного при создании задачи |
| getCompanyUsers() | Функция получает всех сотрудников компании |
| addRow() | Функция отправляет данные с формы в api функцию createNewTask() |
| CreateNewTask() | Принимает данные с клиента и выполнеят сохранение записи в базу данных |

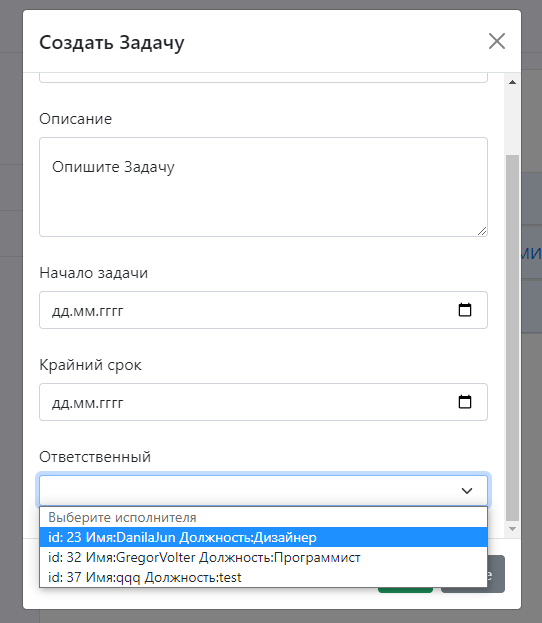


Рисунок. 3.8.8б. Создание задачи

Компонент deleteModal представляет собой модальное окно удаления выбранной задачи. [Приложение 1. Файл deleteTask]. При нажатии на кнопку подтверждения удаления произойдет вызов функции удаление задачи из базы данных (Листинг. 3.8.10).В листинге 3.8.11. представлен код функции удаления записи из базы данных.

Листинг. 3.8.10. VueJS код компонента deliteModal

|  |
| --- |
| <script>  methods: {  deleteUser() {  console.log("удалить строку с id = ", this.$store.state.tempData);  axios  .post("http://apiks/api/", {  action: "deleteRowById",  selectedTableName: "tasks",  selectedId: this.$store.state.tempData,  })  .then(  (response) => (  console.log(response),    this.$emit("reloadTableEmit")  )  );    },  },  };  </script> |

Листинг. 3.8.11. PHP код функции удаления записи из базы данных

|  |
| --- |
| function deleteRowById($id, $tablename)  {  //DELETE FROM `cities` WHERE `id` = 13  R::exec('DELETE FROM ' . $tablename . ' WHERE id = ' . $id);  $table = R::getAll('SELECT \* FROM ' . $tablename);  return $table;  } |

Таблица. 3.8.12. Поясняющая работу функций компонента deleteModal

|  |  |
| --- | --- |
| Название Функции | Описание |
| deleteUser() | Функция передает в api метод deleteRowById() идентификатор записи, затем перезагружает список задач |
| deleteRowById() | Функция удаляет из таблицы tasks запись с полученным идентификатором |

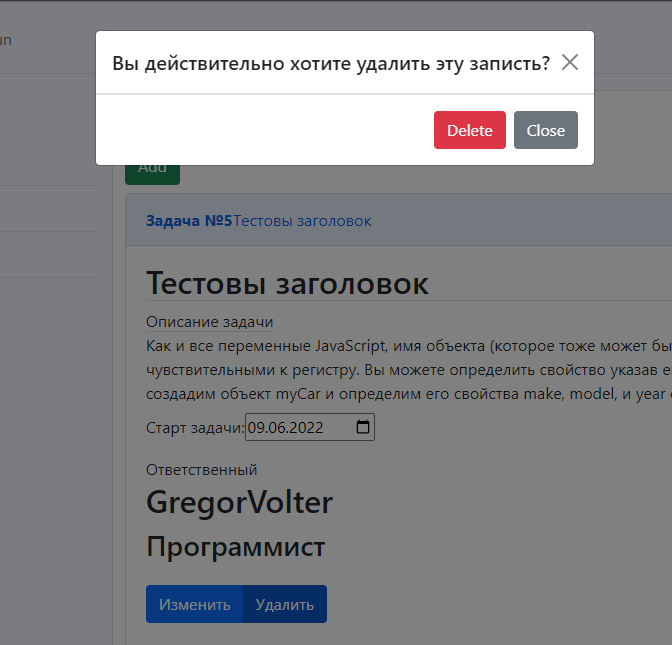


Рисунок. 3.8.12б. Удаление задачи

Компонент editModal представляет собой модальное окно с формой, хранящей информацию о выбранной задаче [Приложение 1. Файл editTask]. При нажатии на кнопку изменить произойдёт вызов функции изменения (Листинг. 3.8.13.) и данные с формы отправятся на сервер где и будут использованы для обновления строки в таблице tasks(Листинг. 3.8.14).

Листинг. 3.8.13. VueJS код компонента editModal

|  |
| --- |
| <script>  methods: {  close(){  this.$emit("reloadTableEmit")  },  setDataProp(){  this.data.taskData = this.$store.state.tempData;  //console.log(this.data.taskData);  },  editRow(formData) {  //ролучил данные из формы  //to do отправить запрос на добавление вернуть отвте ошибку или успех  axios  .post("http://apiks/api/", {  action: "updateTableRow",  selectedTableName: 'tasks',  rowId: formData.id,  updateData: formData,  })  .then(  (response) => (  console.log(response),  //перезагрузить таблицу  this.$emit("reloadTableEmit")  )  );  },  saveData(param) {  this.data.companyUsers = param;  console.log(this.data.companyUsers);  },  getCompanyUsers() {  //console.log("Получаю список пользователей");  axios  .post("http://apiks/api/", {  action: "getCompanyUsers",  companyId: this.$store.getters.getUserCompanyData.id,  })  .then((response) => this.saveData(response.data));  },  },  };  </script> |

Листинг. 3.8.14. PHP код функции обновления записи компонента editModal

|  |
| --- |
| function updateTableRow($tablename, $rowId, $formData)  {  $row = R::load($tablename, $rowId);  // Заполняем объект свойствами  if($formData->pass){  $formData->pass = password\_hash($formData->pass, PASSWORD\_DEFAULT);  }  foreach ($formData as $key => $value) {  print "$key => $value\n";  $row->$key = $value;  }  R::store($row);  } |

Таблица. 3.8.15. Поясняющая работу функций компонента editModal

|  |  |
| --- | --- |
| Название Функции | Описание |
| close () | Функция вызвает перезагрузку страницы |
| saveData() | Сохраняет пользователей компании в список для возможности выбора ответственного при создании задачи |
| getCompanyUsers() | Функция получает всех сотрудников компании |
| editRow() | Функция отправляет данные с формы в api функцию updateTableRow () |
| CreateNewTask() | Принимает данные с клиента и выполнеят сохранение записи в базу данных |

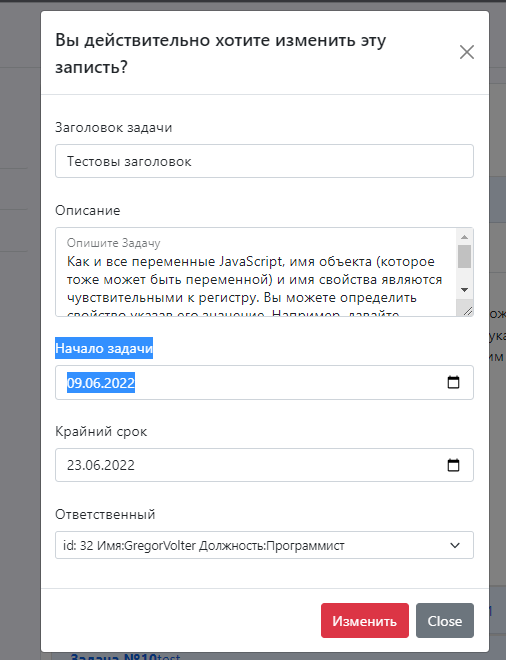


Рисунок 3.8.15б. Редактирование поставленной задачи

Глава 4. Тестирование

Для тестирования web-приложения необходимо обозначить сценарии на основе UML – диаграмм вариантов использования (Рисунок.3 – 4.)

**Сценарии регистрации и авторизации:**

* Сценарий успешной регистрации. Пользователь регистрирует компанию, регистрирует администратора компании, после чего получает уведомления об успешной регистрации.
* Сценарий ошибки регистрации. Пользователь регистрирует компанию, регистрирует администратора, однако если пользователь с указанным логином уже существует либо компания с таким названием уже существует, либо пароли не совпадают. Пользователь получает уведомление об ошибке.
* Сценарий успешной авторизации. Пользователь заполняет форму авторизации, после чего программа находит пользователя с указанным логином и паролем, после чего перенаправляет на главную страницу пользователя в зависимости от роли пользователя.
* Сценарий ошибки авторизации. При авторизации пользователь получает уведомление о неверном логине или пароле.

**Сценарий общего функционала:**

* Сценарий обновления личных данных пользователем. Пользователь посещает страницу личного кабинета где изменяет на свой выбор поля его личной карточки: Логин, Пароль, Имя, Должность, затем при нажатии на кнопку изменить система проверяет поле логина по правилу: поле логин не должно содержать цифр. Если все поля заполнены информация о пользователе изменяется.
* Сценарий неправильности введённого логина. Система видит цифры в логине и при попытке изменить данные выводит ошибку.
* Сценарий создания задачи. Пользователь нажимает кнопку «Создать задачу» заполняет поля: Заголовок, Описание, Крайний срок, Ответственный, если все поля формы заполнены система создает задачу.
* Сценарий ошибки создания задачи. Пользователем при создании задач были пропущены поля и система просит пользователя заполнить их.
* Сценарий удаления задачи. Пользователь может удалить собственные задачи, для этого во вкладке «Поставленные задачи» необходимо выбрать задачу и нажать кнопку действия «Удалить», при нажатии на кнопку пользователь получит уведомление об удалении и если будет повторно выбрана опция удалить произойдет удаление из базы данных и блок поставленных задач обновится, в противном случае ничего не произойдёт.
* Сценарий выполнения задачи. Пользователь открывает вкладку «Мои задачи» и выбирает задачу

**Сценарии кабинета администратора:**

* Сценарий успешного создания личного кабинета пользователя. Администратор заполняет форму добавления нового сотрудника в компании. Система проверяет на наличие сотрудника с таким же логином и формат введённого логина, если пользователь с указанным в форме создания логином не найден и логин заполнен корректно система сохраняет данные о новом сотруднике в системе и обновляет список сотрудников на странице.
* Сценарий ошибки создания личного кабинета сотрудника. Система обнаружила в форме ошибку оформления логина или нашла в базе данных пользователя с таким же логином и вернула ошибку.
* Сценарий успешного обновления данных пользователя. Администратор заполняет форму редактирования профиля выбранного сотрудника в компании. При нажатии на кнопку «Изменить» система проверяет на наличие сотрудника с таким же логином и формат введённого логина, если пользователь с указанным в форме создания логином не найден и логин заполнен корректно система сохраняет данные о новом сотруднике в системе и обновляет список сотрудников на странице.
* Сценарий ошибки изменения данных сотрудника. Система обнаружила в форме ошибку оформления логина или нашла в базе данных пользователя с таким же логином и вернула ошибку.
* Сценарий удаления сотрудника. Администратор может удалять пользователей компании, для этого во вкладке «Сотрудники» необходимо выбрать сотрудника и нажать кнопку действия «Удалить», при нажатии на кнопку администратор получит уведомление об удалении и, если будет повторно выбрана опция произойдет удаление из базы данных и блок сотрудников обновится, в противном случае ничего не произойдёт.
* Сценарий изменения данных о компании. На странице изменения данных о компании администратор изменяет информацию о компании в форме и при нажатии на кнопку изменить данные с формы сохраняются в базе данных, и информация о компании обновляется.

4.1. Сценарии регистрации и авторизации

Для тестирования сценария регистрации и автоизации необходимо запустить приложение и сервер. Информация о состоянии приложения будет отображаться как в диспетчере браузера, так и прямо на странице формы авторизации и регистрации.

Для того чтобы отслеживать поведение системы вовремя регистрация был временно реализован вывод в консоль браузера ответа, как видно на рисунке 4.1.1б. переменная data хранит ответ сервера, информацию о добавленной записи. В случае ошибки регистрации такой как наличие незаполненных полей формы (Рисунок. 4.1.2.). В случае если в системе уже есть компания с указанным именем или пользователем с логином, указанным при регистрации администратора (Рисунок. 4.1.2б.) код ответа будет равен «001» (Рисунок. 4.1.2в.) что вызовет всплывающее уведомление. Не совпадение паролей указанных пи регистрации администратора (Рисунок. 4.1.2г.) вернет в клиент ошибку «002» что вызовет всплывающие уведомление о несовпадении паролей (Рисунок.4.1.2д.).

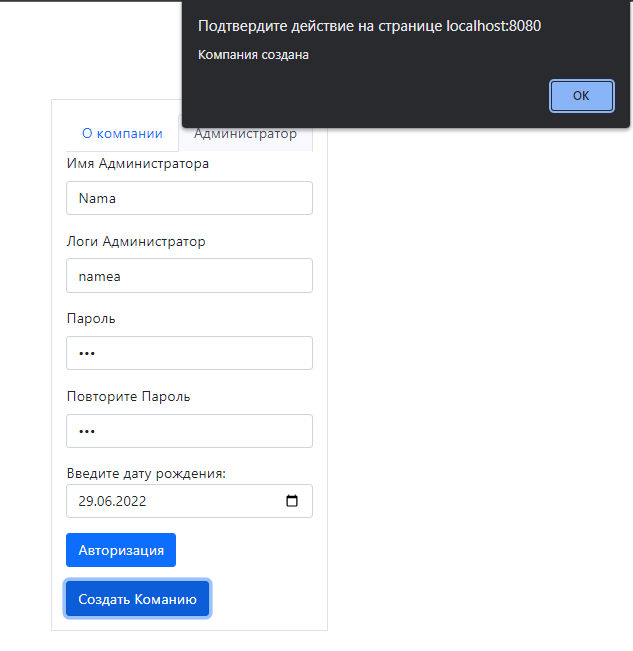


Рисунок. 4.1.1 Успешная регистрация на стороне клиента

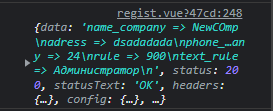


Рисунок. 4.1.1б Ответ сервера при успешной регистрации

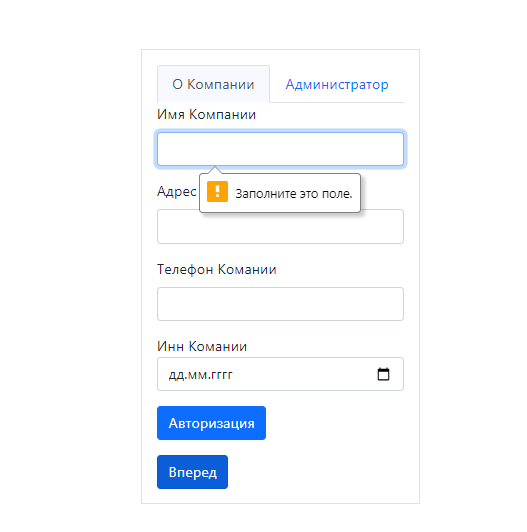


Рисунок. 4.1.2. Ошибка регистрации на стороне клиента (не заполнены поля ввода)

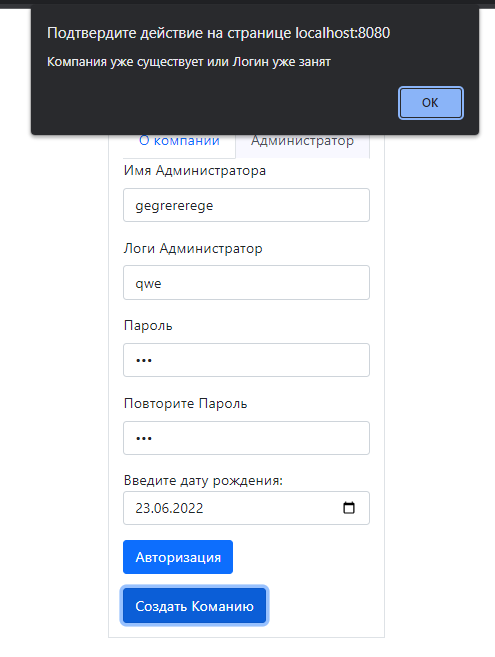


Рисунок. 4.1.2б. Ошибка регистрации на стороне клиента (Компания уже существует или логин уже занят)

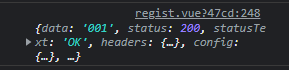


Рисунок. 4.1.2в. Ответ сервера при ошибке 4.1.2б.

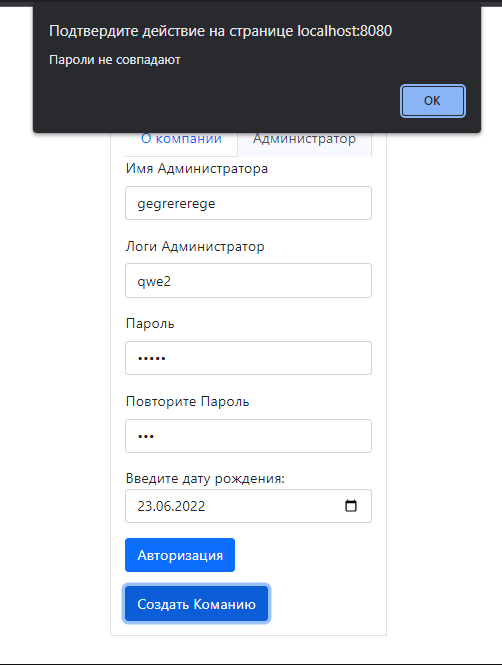


Рисунок. 4.1.2г. Ошибка регистрации на стороне клиента (Пароли не совпадают)

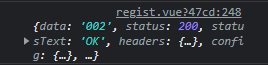


Рисунок. 4.1.2д. Ответ сервера (Компания уже существует или логин уже занят)

При успешной авторизации пользователя перебрасывает на его домашнюю страницу в зависимости от роли, ответ сервера содержит информацию о пользователе (Рисунок. 4.1.3). Если пользователь не заполнил поле формы оно подсвечивается как ошибка (Рисунок. 4.1.4), в случае несовпадения логина или пароля с базой пользователей система вернет в консоль «0» и отобразит пользователю сообщение о неправильном логине или пароле (Рисунок. 4.1.5).

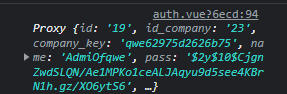


Рисунок. 4.1.3. Ответ сервера при успешной авторизации

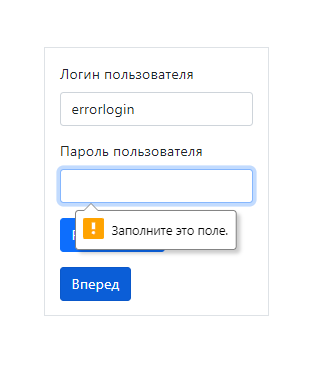


Рисунок. 4.1.4. Ошибка авторизации на стороне клиента (Незаполненное поле)

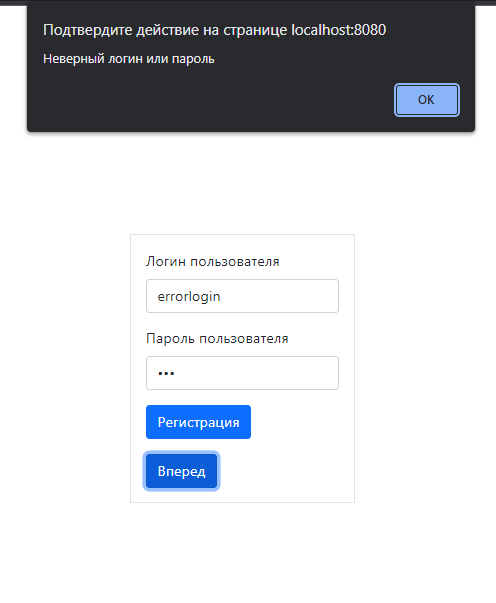


Рисунок. 4.1.5 Ответ сервера при ошибке авторизации

4.2. Сценарии общего функционала

Сценарии общего функционала системы проверяются теми же инструментами что и сценарии авторизации и регистрации. Реакция формы изменения личных данных изображена на рисунке 4.2.1., ответ сервера на рисунке 4.2.1б. При выполнении сценариев работы с задачами осуществляется только проверка наличия введенной информации в поле формы, при наличии хотя бы одного незаполненного поля формы клиент выдает уведомление об ошибке иначе отправляет данные на сервер и уведомляет об успешной операции (Рисунок. 4.2.2-5).

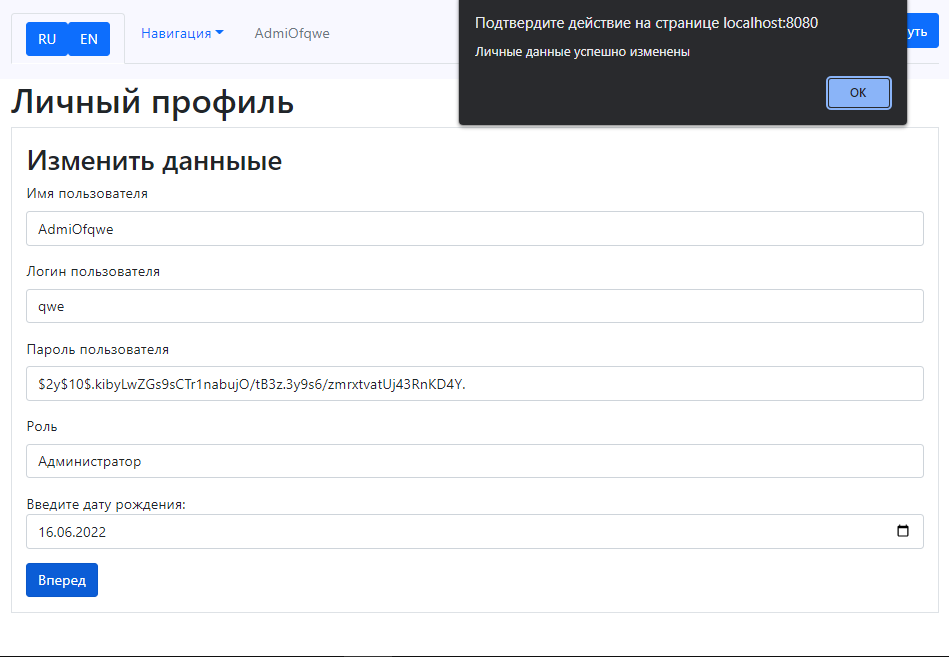


Рисунок. 4.2.1. Форма редактирования данных в личном кабинете

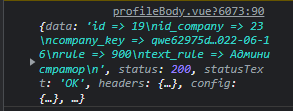


Рисунок. 4.2.1б. Ответ сервера на изменение личных данных

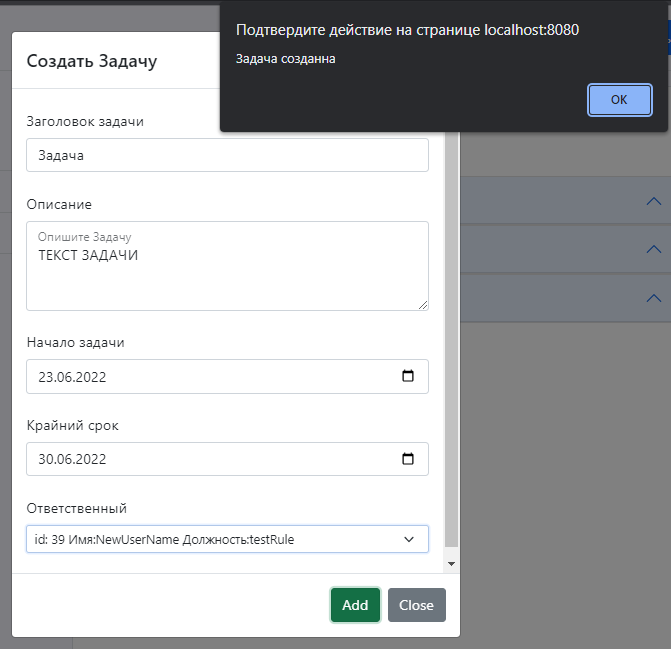


Рисунок. 4.2.2. Успешное добавление задачи

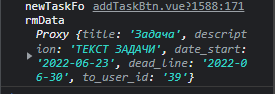


Рисунок. 4.2.2б. Ответ сервера при спешном добавлении задачи

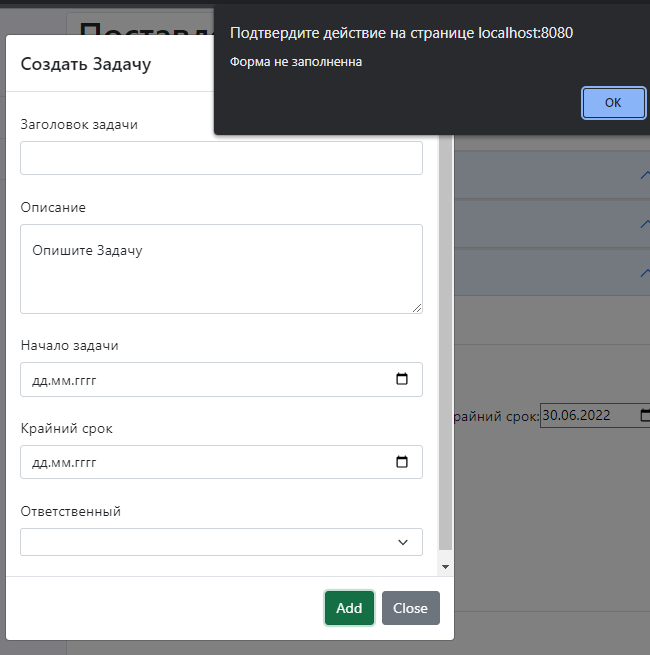


Рисунок. 4.2.3. Незаполненное поле формы создания задачи

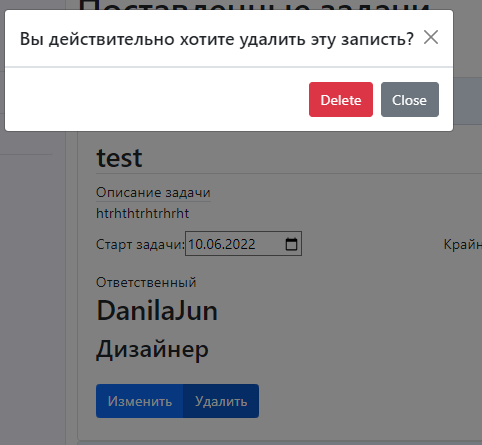


Рисунок. 4.2.4. Удаление задачи

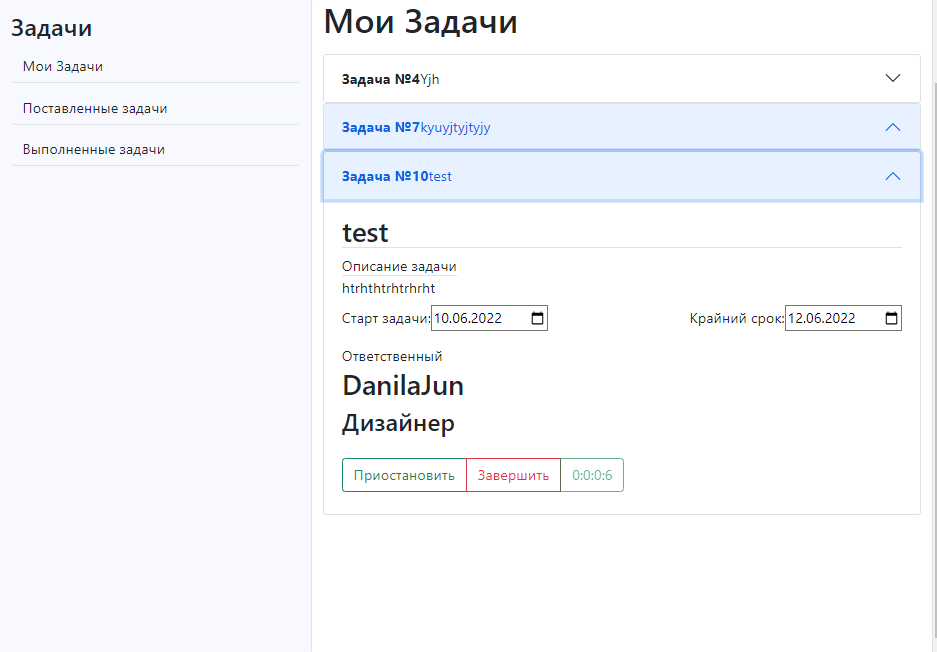


Рисунок. 4.2.5. Выполнение задачи

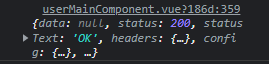


Рисунок. 4.2.5. Ответ сервера на начало выполнения задачи

4.3. Сценарии общего функционала

В обязанности администратора входит создание новых пользователей в компании, процесс должен быть отлажен и работать без сбоев и ошибок, за администратором закреплена компания и информация о ней, данные о компании администратор может изменять во вкладке «О компании», при изменении новые данные обновляются в базе данных. Администратор имеет доступ ко всем полям сотрудников, зарегистрированных в компании, есть возможность изменять: Имя, Логиy, Пароль, Роль. Процесс изменения придерживается правилом проверки логина на строку, проверки на дублирование пользователя в системе, проверки полей форм на наличие в них записей

При успешной регистрации пользователя в систему администратор получит оповещение об успешной операции (Рисунок.4.3.1), сервер вернет информацию о созданном пользователе (Рисунок. 4.3.1б.).

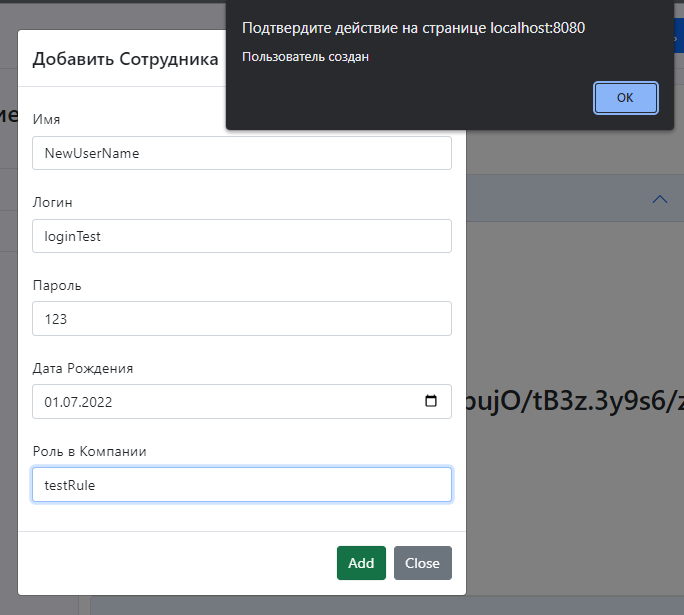


Рисунок. 4.3.1. Успешная регистрация сотрудника



Рисунок. 4.3.1б. Ответ сервера при успешной регистрации сотрудника

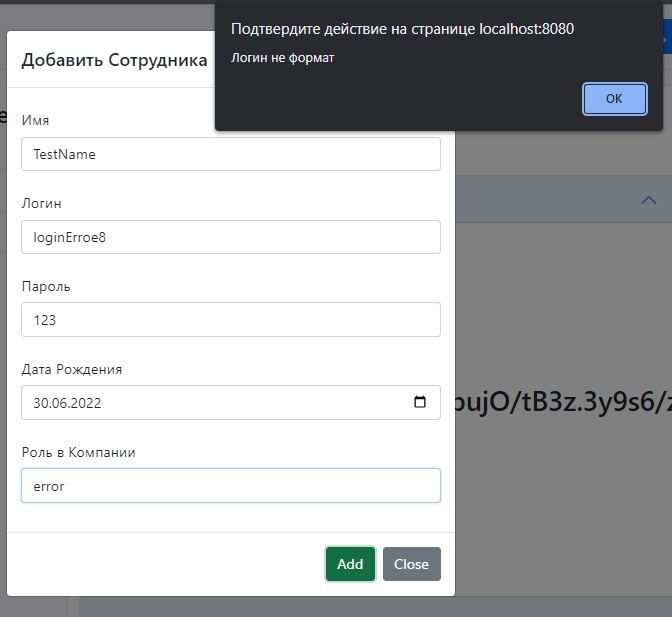


Рисунок. 4.3.1в. Ошибка регистрации сотрудника (Неправильный формат логина)

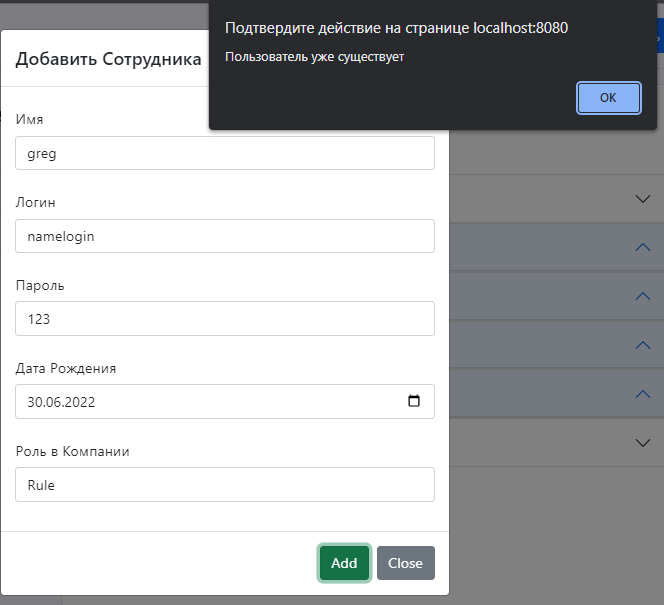


Рисунок. 4.3.1г. Ошибка регистрации сотрудника (Пользователь существует)

В процессе редактирования информации о пользователях так же нужно учитывать правильность формата измененного логина, наличие информации в полях формы. Проверка осуществляется так же, как и в форме регистрации пользователя.

Заключение

В результате выполнения представленной выпускной квалификационной работы была достигнута поставленная цель: разработать менеджер задач деятельности IT компании.

В ходе выполнения поставленной задачи были пройдены следующие этапы разработки web-приложения:

1) Расширенная постановка задачи: на данном этапе были описаны требования к системе, цели и задачи проекта, анализ существующих решений, а также приведено описание используемых технических средств с обоснованием их выбора.

2) Проектирование менеджера задач, в ходе которого были описаны процессы функционирования системы, ролей пользователей, USE-case диаграммы взаимодействия пользователя с системой, диаграммы последовательностей.

3) Реализованы функций системы: регистрации, авторизации пользователей, работа с задачами, работа администратора, работа системы.

В результате разработки была создана автоматизированная система управления рабочими процессами, упрощающая ведения задач для сотрудника и администратора компании, снижающая затраты труда и времени на мониторинг выполнение задач, составления отчетности эффективности сотрудников.

Список источников

1. ГОСТ 34.003-90 Информационная технология (ИТ). Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения [Электронный ресурс]. - Режим доступа: <http://docs.cntd.ru/document/1200006979> (дата обращения: 09.04.2022).
2. РД 50-680-88 Методические указания. Автоматизированные системы. Основные положения [Электронный ресурс]. - Режим доступа: http://docs.cntd.ru/document/1200006976 (дата обращения: 09.04.2022).
3. RedBeanPHP::CRUD [Электронный ресурс]. - Режим доступа: <https://www.redbeanphp.com/index.php?p=/crud> (дата обращения: 12.05.2022).
4. Усеинов Ш. М. ЗАГРУЗКА ЗАПИСЕЙ ИЗ БАЗЫ ДАННЫХ MYSQL В ORM-БИБЛИОТЕКЕ REDBEANPHP: Информационно-компьютерные технологии в экономике, образовании и социальной сфере. – 2018. – №. 1. – С. 33-38. (дата обращения: 12.05.2022).
5. Хасан Г. UML. Проектирование систем реального времени, распределенных и параллельных приложений: М.: Издательство «ДМК. – 2012. (дата обращения: 12.05.2022).
6. Шмаков С. Э., Смекалин Н. Н., Щенникова Е. В. Создание WEB-приложения с использованием сторонних API. – 2021. (дата обращения: 12.05.2022).
7. Reading and writing to file - PhpSpreadsheet Documentation [Электронный ресурс]. - Режим доступа: https://phpspreadsheet.readthedocs.io/en/latest/topics/reading-and-writing-to-file/ (дата обращения: 12.05.2022).
8. RedBeanPHP:: Connection [Электронный ресурс]. - Режим доступа: https://redbeanphp.com/index.php?p=/connection (дата обращения: 12.05.2022).
9. RedBeanPHP :: Querying [Электронный ресурс]. - Режим доступа: https://redbeanphp.com/index.php?p=/querying (дата обращения: 12.05.2022).
10. RedBeanPHP :: Many-to-many [Электронный ресурс]. - Режим доступа: <https://www.redbeanphp.com/index.php?p=/many_to_many> (дата обращения: 12.05.2022).
11. JavaScript [Электронный ресурс]. - Режим доступа: https://developer.mozilla.org/ru/docs/ Web/JavaScript (дата обращения: 23.05.2022).
12. Bootstrap v5.2 документация [Электронный ресурс]. - Режим доступа: <https://getbootstrap.com/docs/5.2/getting-started/introduction/> (дата обращения: 23.05.2022.
13. Bootstrap Accordion // [Электронный ресурс]. - Режим доступа: <https://getbootstrap.com/docs/5.2/components/accordion/#content> (дата обращения: 23.05.2022).
14. Bootstrap Buttons // [Электронный ресурс]. - Режим доступа: https://getbootstrap.com/docs/5.2/components/buttons/#content (дата обращения: 23.05.2022).
15. REST API Best Practices ­— havr.com. [Электронный ресурс]. - Режим доступа: https://habr.com/ru/post/351890/ (дата обращения: 23.05.2022).
16. Принципы построения REST JSON API – habr.com [Электронный ресурс]. - Режим доступа: <https://habr.com/ru/post/447322/>). (дата обращения: 23.05.2022).
17. Классы и объекты PHP [Электронный ресурс]. - Режим доступа: <https://www.php.net/manual/ru/language.oop5.php> (дата обращения: 23.05.2022).
18. Jsone\_encode PHP [Электронный ресурс]. - Режим доступа: <https://www.php.net/manual/ru/function.json-encode.php> (дата обращения: 23.05.2022).
19. JavaScript Object Notation [Электронный ресурс]. - Режим доступа: <https://www.php.net/manual/ru/book.json.php> (дата обращения: 23.05.2022).
20. Vue CLI VueJS guide [Электронный ресурс]. - Режим доступа: <https://cli.vuejs.org/guide/> (дата обращения: 23.05.2022).
21. Отрисовка списков [Электронный ресурс]. - Режим доступа: <https://ru.vuejs.org/v2/guide/list.html> (дата обращения: 23.05.2022).
22. Условная отрисовка [Электронный ресурс]. - Режим доступа:: <https://ru.vuejs.org/v2/guide/conditional.html> (дата обращения: 23.05.2022).
23. Хуки жизненного цикла [Электронный ресурс]. - Режим доступа: <https://v3.ru.vuejs.org/ru/api/options-lifecycle-hooks.html> (дата обращения: 23.05.2022).
24. Входные параметры [Электронный ресурс]. - Режим доступа: <https://ru.vuejs.org/v2/guide/components-props.html> (дата обращения: 23.05.2022).
25. Vuex Single State Tree [Электронный ресурс]. - Режим доступа: https://vuex.vuejs.org/guide/state.html (дата обращения: 31.05.2022).
26. Vuex Getters [Электронный ресурс]. - Режим доступа: https://vuex.vuejs.org/guide/getters.html (дата обращения: 31.05.2022).
27. Vuex Mutation [Электронный ресурс]. - Режим доступа: https://vuex.vuejs.org/guide/mutations.html (дата обращения: 31.05.2022).
28. Vuex Action [Электронный ресурс]. - Режим доступа:: https://vuex.vuejs.org/guide/actions.html (дата обращения: 31.05.2022).
29. Dynamic Route Matching with [Электронный ресурс]. - Режим доступа: https://router.vuejs.org/guide/essentials/dynamic-matching.html (дата обращения: 31.05.2022).
30. Navigation Guards [Электронный ресурс]. - Режим доступа: https://router.vuejs.org/guide/advanced/navigation-guards.html (дата обращения: 31.05.2022).

Приложение 1. HTML-разметка компонентов веб-приложения

1. Файл regist.vue

|  |
| --- |
| <template>  <langSwither />  <div class="container-fluid">  <form class="p-3 border" @submit.prevent>  <div v-if="formState == 0" class="form-state">  <div class="form-state"></div>  <ul class="nav nav-tabs">  <li class="nav-item">  <a class="nav-link active" aria-current="page" href="#"  >О Компании</a  >  </li>  <li class="nav-item">  <a  class="nav-link"  @click="companyDataValidate"  aria-current="page"  href="#"  >Администратор</a  >  </li>  </ul>  <div class="mb-3">  <label for="companyName" class="form-label">Имя Компании</label>  <input  type="text"  class="form-control"  id="companyName"  aria-describedby="emailHelp"  v-model="formReg.company.name\_company"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <label for="companyAdres" class="form-label">Адрес Комании</label>  <input  type="text"  class="form-control"  id="companyAdres"  v-model="formReg.company.adress"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <label for="companyPhone" type="password" class="form-label"  >Телефон Комании</label  >  <input  type="text"  class="form-control"  id="companyPhone"  v-model="formReg.company.phone\_number"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <label for="companyInn">Инн Комании</label>  <input  type="date"  class="form-control"  id="companyInn"  v-model="formReg.company.inn"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <button class="btn btn-primary" @click="this.$router.push('/')">  {{ $t("localeAuth") }}  </button>  </div>  <button  type="submit"  class="btn btn-primary go"  @click="companyDataValidate"  >  {{ $t("loacleBtnSubmite") }}  </button>  </div>  <div class="form-state" v-if="formState == 1">  <ul class="nav nav-tabs">  <li class="nav-item">  <a  class="nav-link"  @click="formState--"  aria-current="page"  href="#"  >О компании</a  >  </li>  <li class="nav-item">  <a class="nav-link active" aria-current="page" href="#"  >Администратор</a  >  </li>  </ul>  <div class="mb-3">  <label for="adminName" class="form-label">Имя Администратора</label>  <input  type="text"  class="form-control"  id="adminName"  aria-describedby="emailHelp"  v-model="formReg.admin.name"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <label for="adminLogin" class="form-label">Логи Администратор</label>  <input  type="text"  class="form-control"  id="adminLogin"  v-model="formReg.admin.login"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <label for="adminPass1" type="password" class="form-label"  >Пароль</label  >  <input  type="password"  class="form-control"  id="adminPass1"  v-model="formReg.admin.pass"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <label for="adminPass1" type="passwor2" class="form-label"  >Повторите Пароль</label  >  <input  type="password"  class="form-control"  id="adminPass2"  v-model="formReg.admin.pass2"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <label for="inputDate">{{ $t("localeDateUser") }}</label>  <input  type="date"  class="form-control"  v-model="formReg.admin.birth"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <button class="btn btn-primary" @click="this.$router.push('/')">  {{ $t("localeAuth") }}  </button>  </div>  <button type="submit" class="btn btn-primary" @click="registerUser">  Создать Команию  </button>  </div>  </form>  </div>  </template> |

1. Файл auth.vue

|  |
| --- |
| <template>  <langSwither />  <div class="container-fluid">  <form class="p-3 border" @submit.prevent>  <div class="mb-3">  <label for="exampleInputPassword1" class="form-label">{{  $t("localeLoginUser")  }}</label>  <input  type="text"  class="form-control"  id="login-input"  v-model="formAuth.userLogin"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <label for="exampleInputPassword1" type="password" class="form-label">{{  $t("localePassUser")  }}</label>  <input  type="password"  class="form-control"  id="password-input"  v-model="formAuth.userPass"  autocomplete="on"  required  />  </div>  <div class="mb-3">  <button class="btn btn-primary" @click="this.$router.push('regist')">  {{ $t("localeRegist") }}  </button>  </div>  <button type="submit" class="btn btn-primary" @click="aithorizationUser">  {{ $t("loacleBtnSubmite") }}  </button>  </form>  </div>  </template> |

1. Profile.vue

|  |
| --- |
| <template>  <div class="container-fluid">  <h1>{{ $t("localeProfileh1") }}</h1>  <form class="p-3 border" @submit.prevent>  <h2>{{ $t("localeProfileh2") }}</h2>  <div class="mb-3">  <label for="exampleInputEmail1" class="form-label">{{  $t("localeNameUser")  }}</label>  <input  type="text"  class="form-control"  id="exampleInputEmail1"  aria-describedby="emailHelp"  v-model="profileData.name"  />  </div>  <div class="mb-3">  <label for="exampleInputPassword1" class="form-label">{{  $t("localeLoginUser")  }}</label>  <input  type="text"  class="form-control"  id="exampleInputPassword1"  v-model="profileData.login"  />  </div>  <div class="mb-3">  <label for="exampleInputPassword1" type="password" class="form-label">{{  $t("localePassUser")  }}</label>  <input  type="text"  class="form-control"  id="exampleInputPassword1"  v-model="profileData.pass"  />  </div>  <div class="mb-3">  <label for="inputDate">{{ $t("localeDateUser") }}</label>  <input type="date" class="form-control" v-model="profileData.birth" />  </div>  <button type="submit" class="btn btn-primary" @click="changeUserData">  {{ $t("loacleBtnSubmite") }}  </button>  </form>  </div>  </template> |

1. Файл addUserBtn.vue

|  |
| --- |
| <template>  <button  type="button"  class="btn btn-primary btn-success"  data-bs-toggle="modal"  data-bs-target="#addNewUser"  >  Add  </button>  <div  class="modal fade"  id="addNewUser"  tabindex="-1"  aria-labelledby="exampleModalLabel"  aria-hidden="true"  >  <div class="modal-dialog modal-dialog-scrollable">  <div class="modal-content">  <div class="modal-header">  <h5 class="modal-title" id="exampleModalLabel">  Добавить Сотрудника  </h5>  <button  type="button"  class="btn-close"  data-bs-dismiss="modal"  aria-label="Close"  ></button>  </div>  <div class="modal-body">  <form @submit.prevent>  <div  class="mb-3"  v-for="col in data.formLableNamesObj"  v-bind:key="col"  v-bind:col="col"  >  <label for="recipient-name" class="col-form-label">  {{ col.lable }}</label  >  <input  v-if="col.vmodelName == 'birth'"  v-model="data.formDataList[col.vmodelName]"  type="date"  name="first\_name"  class="form-control class-formDataList"  autocomplete="on"  required  />  <input  v-else  v-model="data.formDataList[col.vmodelName]"  type="text"  name="first\_name"  class="form-control class-formDataList"  autocomplete="on"  required  />  </div>  </form>  </div>  <div class="modal-footer addRowBtn">  <button  @click="addRow(data.formDataList)"  class="btn btn-primary btn-success"  type="submit"  data-bs-dismiss="modalModeIfValidate"  >  Add  </button>  <button  type="button"  class="btn btn-secondary"  data-bs-dismiss="modal"  ref="Close"  >  Close  </button>  </div>  </div>  </div>  </div>  </template> |

5. Файл aboutCompany.vue

|  |
| --- |
| <template>  <div class="about-container">  <form @submit.prevent>  <div  class="mb-3"  v-for="col in data.formLableNamesObj"  v-bind:key="col"  v-bind:col="col"  >  <label for="recipient-name" class="col-form-label">  {{ col.lable }}</label  >    <input  v-model="data.myCompanyData[col.vmodelName]"  type="text"  name="first\_name"  class="form-control class-formDataList"  autocomplete="on"  required  />  </div>  <button type="submit" class="btn btn-primary" @click="editRow(data.myCompanyData)">  Изменить  </button>  </form>  </div>  </template> |

6. Файл addTaskBtn

|  |
| --- |
| <div v-if="activeItem == 0" class="selected-option">  <h1>Мои Задачи</h1>  <div class="user-list-main pt-2">  <div  v-for="task in data.myCreatedTask"  v-bind:key="task"  v-bind:task="task"  class="accordion"  id="accordionExample"  >  <div class="accordion-item">  <h2 class="accordion-header" :id="'flush-headingOne' + task.id">  <button  class="accordion-button"  type="button"  data-bs-toggle="collapse"  :data-bs-target="'#' + 'task' + task.id"  aria-expanded="true"  :aria-controls="'task' + task.id"  @click="getUser(task.from\_user\_id);countTimeTask(task)"  >  <b>Задача №{{ task.id }}</b  ><span>{{ task.title }}</span>  </button>  </h2>  <div  :id="'task' + task.id"  class="accordion-collapse collapse"  :aria-labelledby="'flush-headingOne' + task.id"  data-bs-parent="#accordionExample"  >  <div class="accordion-body">  <h2 class="border-bottom">{{ task.title }}</h2>  <span class="border-bottom">Описание задачи</span>  <br />  <span>{{ task.description }}</span>  <br />  <div class="calendar-div py-2">  <div class="left">  <span>Старт задачи:</span>  <input  v-model="task.date\_start"  type="date"  name=""  id=""  />  </div>  <div class="right">  <span>Крайний срок:</span>  <input  v-model="task.dead\_line"  type="date"  name=""  id=""  />  </div>  </div>  <div class="to-user-div py-2">  <span>Ответственный</span>  <h2>{{ data.taskMaker.name }}</h2>  <h3>{{ data.taskMaker.text\_rule }}</h3>  </div>  <div  v-if="task.task\_status == 'started'"  class="btn-group py-2"  role="group"  >  <button  type="button"  class="btn btn-outline-success"  @click="setStatusTask(task.id, 'stoped')"  >  Приостановить  </button>  <button  type="button"  class="btn btn-outline-danger"  @click="setStatusTask(task.id, 'closed')"  >  Завершить  </button>  <button  type="button"  class="btn btn-outline-success"  disabled  >  {{task.visited\_data}}  </button>  </div>  <div  v-if="task.task\_status == 'created' || task.task\_status == 'stoped'"  class="btn-group py-2"  role="group"  >  <button  type="button"  class="btn btn-outline-success"  @click="setStatusTask(task.id, 'started')"  >  Начать  </button>    </div>    </div>  </div>  </div>  </div>  </div>  </div> |

1. Файл tasks.vue

|  |
| --- |
| <template>  <div class="container-fluid">  <div class="row dbcomponent-row">  <div class="col bd-tables">  <div class="bd-tables-container py-3">  <h3>Администрирование</h3>  <div class="d-grid gap-2">  <button  type="button"  class="btn border-bottom"  @click="getCompanyData"  >  О Комапании  </button>  <button  type="button"  class="btn border-bottom"  @click="getCompanysUser"  >  Пользователи  </button>  <button  type="button"  class="btn border-bottom"  @click="activeItem = 2"  >  Задачи  </button>  </div>  </div>  </div>  <div class="col-8 main-table-container rounded-start border">  <!-- aboutComp -->  <div v-if="activeItem == 0" class="selected-option">  <h1>О Компании</h1>  <aboutCompany />  </div>  <!-- users -->  <div v-else-if="activeItem == 1" class="selected-option">  <h1>Пользователи</h1>  <addUserModal @reloadTableEmit="getCompanysUser()" />  <div class="user-list-main pt-2">  <div  v-for="user in companyUsers"  v-bind:key="user"  v-bind:user="user"  class="accordion"  id="accordionExample"  >  <div class="accordion-item">  <h2 class="accordion-header" id="headingOne">  <button  class="accordion-button"  type="button"  data-bs-toggle="collapse"  :data-bs-target="'#' + user.login"  aria-expanded="true"  :aria-controls="user.login"  >  Имя пользователя: {{ user.name }} Роль: {{ user.text\_rule }}  </button>  </h2>  <div  :id="user.login"  class="accordion-collapse collapse"  aria-labelledby="headingOne"  data-bs-parent="#accordionExample"  >  <div class="accordion-body">  <span class="border-bottom">Имя пользователя</span>  <h2>{{ user.name }}</h2>  <span class="border-bottom">Логин Пользователя</span>  <h2>{{ user.login }}</h2>  <span class="border-bottom">Пароль Пользователя</span>  <h2>{{ user.pass }}</h2>  <span class="border-bottom">Дата рожденияя</span>  <h2>{{ user.birth }}</h2>  <span class="border-bottom">Роль в компании</span>  <h2>{{ user.text\_rule }}</h2>  <div class="btn-group">  <a  href="#"  class="btn btn-primary"  data-bs-toggle="modal"  data-bs-target="#editUserModal"  @click="setEditData(user)"  >Изменить</a  >  <a  href="#"  class="btn btn-primary active"  aria-current="page"  data-bs-toggle="modal"  data-bs-target="#deleteModal"  @click="this.$store.commit('setTempData', user.id)"  >Удалить</a  >  </div>  </div>  </div>  </div>  </div>  </div>  <deleteModal @reloadTableEmit="getCompanysUser()" />  <editUserModal @reloadTableEmit="getCompanysUser()" ref="editModal" />  </div>  <!-- tasks -->  <div v-else-if="activeItem == 2" class="selected-option">  <h1>2</h1>  </div>  </div>  </div>  </div>  </template> |

1. Файл addTaskBtn

|  |
| --- |
| <template>  <button  type="button"  class="btn btn-primary btn-success"  data-bs-toggle="modal"  data-bs-target="#addNewTask"  @click="getCompanyUsers"  >  Add  </button>  <div  class="modal fade"  id="addNewTask"  tabindex="-1"  aria-labelledby="exampleModalLabel"  aria-hidden="true"  >  <div class="modal-dialog modal-dialog-scrollable">  <div class="modal-content">  <div class="modal-header">  <h5 class="modal-title" id="exampleModalLabel">Создать Задачу</h5>  <button  type="button"  class="btn-close"  data-bs-dismiss="modal"  aria-label="Close"  ></button>  </div>  <div class="modal-body">  <form @submit.prevent>  <div  class="mb-3"  v-for="col in data.formLableNamesObj"  v-bind:key="col"  v-bind:col="col"  >  <label for="recipient-name" class="col-form-label">  {{ col.lable }}</label  >  <input  v-if="  col.vmodelName == 'date\_start' ||  col.vmodelName == 'dead\_line'  "  v-model="data.formDataList[col.vmodelName]"  type="date"  name="first\_name"  class="form-control class-formDataList"  autocomplete="on"  required  />  <input  v-else-if="  col.vmodelName != 'description' &&  col.vmodelName != 'to\_user\_id'  "  v-model="data.formDataList[col.vmodelName]"  type="text"  name="first\_name"  class="form-control class-formDataList"  autocomplete="on"  required  />  <div v-if="col.vmodelName == 'description'" class="viftextarea">  <div class="form-floating">  <textarea  class="form-control"  placeholder="Leave a comment here"  id="floatingTextarea2"  style="height: 100px"  v-model="data.formDataList[col.vmodelName]"  ></textarea>  <label for="floatingTextarea2">Опишите Задачу</label>  </div>  </div>  <div v-if="col.vmodelName == 'to\_user\_id'" class="viftextarea">  <select  class="form-select form-select-sm"  aria-label=".form-select-sm example"  v-model="data.formDataList[col.vmodelName]"  >  <option disabled value="">Выберите исполнителя</option>  <option  v-for="selectUser in data.companyUsers"  v-bind:key="selectUser"  v-bind:selectUser="selectUser"  :value="selectUser.id"    >  id: {{selectUser.id}} Имя:{{selectUser.name}} Должность:{{selectUser.text\_rule}}  </option>  </select>  </div>  </div>  </form>  </div>  <div class="modal-footer addRowBtn">  <button  @click="addRow(data.formDataList)"  class="btn btn-primary btn-success"  type="submit"  data-bs-dismiss="modalModeIfValidate"  >  Add  </button>  <button  type="button"  class="btn btn-secondary"  data-bs-dismiss="modal"  @click="close"  >  Close  </button>  </div>  </div>  </div>  </div>  </template> |

1. Файл deleteTask

|  |
| --- |
| <template>  <div  class="modal fade"  id="deleteModal"  tabindex="-1"  aria-labelledby="exampleModalLabel"  aria-hidden="true"  >  <div class="modal-dialog">  <div class="modal-content">  <div class="modal-header">  <h5 class="modal-title" id="exampleModalLabel">  Вы действительно хотите удалить эту записть?  </h5>  <button  type="button"  class="btn-close"  data-bs-dismiss="modal"  aria-label="Close"  ></button>  </div>  <div class="modal-footer">  <button  @click="deleteUser()"  type="button"  class="btn btn-danger"  data-bs-dismiss="modal"  >  Delete  </button>  <button  type="button"  class="btn btn-secondary"  data-bs-dismiss="modal"  >  Close  </button>  </div>  </div>  </div>  </div>  </template> |

1. Файл editTask

|  |
| --- |
| <template>  <div  class="modal fade"  id="editTaskModal"  tabindex="-1"  aria-labelledby="exampleModalLabel"  aria-hidden="true"  >  <div class="modal-dialog">  <div class="modal-content">  <div class="modal-header">  <h5 class="modal-title" id="exampleModalLabel">  Вы действительно хотите изменить эту записть?  </h5>  <button  type="button"  class="btn-close"  data-bs-dismiss="modal"  aria-label="Close"  ></button>  </div>  <div class="modal-body">  <form @submit.prevent>  <div  class="mb-3"  v-for="col in data.formLableNamesObj"  v-bind:key="col"  v-bind:col="col"  >  <label for="recipient-name" class="col-form-label">  {{ col.lable }}</label  >  <input  v-if="  col.vmodelName == 'date\_start' ||  col.vmodelName == 'dead\_line'  "  v-model="data.taskData[col.vmodelName]"  type="date"  name="first\_name"  class="form-control class-formDataList"  autocomplete="on"  required  />  <input  v-else-if="  col.vmodelName != 'description' &&  col.vmodelName != 'to\_user\_id'  "  v-model="data.taskData[col.vmodelName]"  type="text"  name="first\_name"  class="form-control class-formDataList"  autocomplete="on"  required  />  <div v-if="col.vmodelName == 'description'" class="viftextarea">  <div class="form-floating">  <textarea  class="form-control"  placeholder="Leave a comment here"  id="floatingTextarea2"  style="height: 100px"  v-model="data.taskData[col.vmodelName]"  ></textarea>  <label for="floatingTextarea2">Опишите Задачу</label>  </div>  </div>  <div v-if="col.vmodelName == 'to\_user\_id'" class="viftextarea">  <select  class="form-select form-select-sm"  aria-label=".form-select-sm example"  v-model="data.taskData[col.vmodelName]"  >  <option disabled value="">Выберите исполнителя</option>  <option  v-for="selectUser in data.companyUsers"  v-bind:key="selectUser"  v-bind:selectUser="selectUser"  :value="selectUser.id"    >  id: {{selectUser.id}} Имя:{{selectUser.name}} Должность:{{selectUser.text\_rule}}  </option>  </select>  </div>  </div>  </form>  </div>  <div class="modal-footer">  <button  @click="editRow(this.data.taskData)"  type="button"  class="btn btn-danger"  data-bs-dismiss="modal"  >  Изменить  </button>  <button  type="button"  class="btn btn-secondary"  data-bs-dismiss="modal"  @click="close"  >  Close  </button>  </div>  </div>  </div>  </div>  </template> |

Приложение 2. Серверная часть приложения, API

1. Файл Api.php

|  |
| --- |
| <?php  header("Access-Control-Allow-Origin: \*");  header("Access-Control-Allow-Methods: GET, POST, PUT");  header("Access-Control-Allow-Headers: Content-Type");  ?> |

1. Файл Function.php

|  |
| --- |
| <?php  class Functions  {  function deleteRowById($id, $tablename)  {  //DELETE FROM `cities` WHERE `id` = 13  R::exec('DELETE FROM ' . $tablename . ' WHERE id = ' . $id);  $table = R::getAll('SELECT \* FROM ' . $tablename);  return $table;  }  function insertIntoTable($tablename, $formData)  {  $table = R::dispense($tablename);  // Заполняем объект свойствами  //var\_dump($table);  foreach ($formData as $key => $value) {  print "$key => $value\n";  $table->$key = $value;  }  R::store($table);  }  function createUser($userData, $companyData)  {  $table = R::dispense('users');  $checkUser = R::findOne('users', 'login = ?', [$userData->login]);  if (!empty($checkUser)) {  return "001";  } else {  $userData->id\_company = $companyData->id;  $userData->company\_key = $companyData->company\_key;  // Заполняем объект свойствами  $userData->rule = 100;  $userData->pass = password\_hash($userData->pass, PASSWORD\_DEFAULT);  foreach ($userData as $key => $value) {  print "$key => $value\n";  $table->$key = $value;  }  R::store($table);  }  }  function updateTableRow($tablename, $rowId, $formData)  {  $row = R::load($tablename, $rowId);  // Заполняем объект свойствами  if($formData->pass){  $formData->pass = password\_hash($formData->pass, PASSWORD\_DEFAULT);  }  foreach ($formData as $key => $value) {  print "$key => $value\n";  $row->$key = $value;  }  R::store($row);  }  function registCompanyAndAdmCompany($companyData, $adminData)  {  //to do получить дату и отправить в базу данных информацию о компании и ее созадтеле  //создать записть о создателе как админе, его дату обработать(шифр пароля)  //$companyR = R::dispense("about\_company");  $nameCompany = $companyData->name\_company;  $admLogin = $adminData->login;  $checkCompany = R::findOne('companys', 'name\_company = ?', [$nameCompany]);  $checkAdm = R::findOne('users', 'login = ?', [$admLogin]);  $func = new Functions;  //$errorMSG = "001";  if (!empty($checkCompany) || !empty($checkAdm)) {  return "001";  } else if ($adminData->pass != $adminData->pass2) {  return "002";  } else {  //to do зарегать компанию и админа  //удаление проверочного пароля из объекта  unset($adminData->pass2);  //добавление к записи компании имя автора(администратора)  $uniqIdCompany = $nameCompany . uniqid();  $companyData->company\_key = $uniqIdCompany;  $companyData->admin\_name = $adminData->name;  $func->insertIntoTable('companys', $companyData);  //добавление к записи юзера id компании  $findNewComp = R::findOne('companys', 'name\_company = ?', [$nameCompany]);  $findIdComp = $findNewComp->id;  $adminData->company\_key = $uniqIdCompany;  $adminData->id\_company = $findIdComp;  //добавление к записи юзера его роль  $adminData->rule = 900;  $adminData->text\_rule = 'Администратор';  $adminData->pass = password\_hash($adminData->pass, PASSWORD\_DEFAULT);  $func->insertIntoTable("users", $adminData);  }  }  function chekUserAndAuthorizationUser($authFormData)  {  //to do найти пользователя по логину и паролю  $userLogin = $authFormData->userLogin;  $checkUser = R::findOne('users', 'login = ?', [$userLogin]);  $checkUserCompany = R::findOne('companys', 'id = ?', [$checkUser->id\_company]);  if (!empty($checkUser)) {  $hashPass = $checkUser->pass;  $verifyPwd = password\_verify($authFormData->userPass, $hashPass);  if ($verifyPwd) {  //создать объект с информацией о пользователе и о его компании  $array1 = array($checkUser, $checkUserCompany);  return $array1;  } else {  return "003";  }  } else {  return "003";  }  }    function getCompanyUsers($idCompany)  {  $users = R::getAll('SELECT \* FROM users WHERE id\_company = '. $idCompany . ' AND rule = 100');  return $users;  }  function createNewTask($taskData,$fromUserId)  {  $func = new Functions;  $taskData->from\_user\_id = $fromUserId;  $taskData->task\_status = "created";  $func->insertIntoTable("tasks", $taskData);  }  function getCreatedTask($fromUserId){  //вернуть задачи где from\_user\_id == авторизированный пользователь    $users = R::getAll("SELECT \* FROM tasks WHERE from\_user\_id = ". $fromUserId . " AND task\_status = 'created' OR task\_status = 'stareted' OR task\_status = 'stoped'");    return $users;  }  function getMyTasks($userId){  //вернуть задачи где to\_user\_id == авторизированный пользователь  $users = R::getAll("SELECT \* FROM tasks WHERE to\_user\_id = ". $userId . " AND task\_status != 'closed'");  return $users;  }  function getCompliteTasks($userId){  //вернуть задачи где status == авторизированный пользователь  $status = 'closed';  $users = R::getAll('SELECT \* FROM tasks WHERE to\_user\_id = '. $userId . ' AND task\_status ==' . $status);  return $users;  }  function getUserById($userId){  //вернуть задачи где status == авторизированный пользователь;  $checkUser = R::findOne('users', 'id = ?', [$userId]);  return $checkUser;  }  function setTaskStatus($rowId, $status)  {  $row = R::load('tasks', $rowId);  // Заполняем объект свойствами  $row->task\_status = $status;  if($status == 'started'){  $today = date("Y-m-d H:i:s");  $row->started\_btn = $today;  }  R::store($row);  }  function countTimeTask($rowId,$status){  //формируем время визита в дате и в часах  //получаем дату старта выполнения задачи  //вычесляем разницу возвращаем  $row = R::load('tasks', $rowId);  // Заполняем объект свойствами  if($status == 'started'){  $today = date("Y-m-d H:i:s");  $startedDate = $row->started\_btn;  $first\_date = new DateTime($today);  $second\_date = new DateTime($startedDate);  $difference = $first\_date->diff($second\_date);  $days = $difference->d;  $hour = $difference->h;  $min = $difference->i;  $sec = $difference->s;  $string1 = $days.':'.$hour.':'.$min.':'.$sec;  //return $array1;  $row->visited\_data = $string1;  }  R::store($row);  }  } |

1. Файл index.php

|  |
| --- |
| <?php  include "api.php";  include "functions.php";  require "../rb/rb.php";  R::setup('mysql:host=localhost;dbname=rb\_it', 'root', 'root'); //for both mysql or mariaDB//rb\_pharmacy  R::freeze(true);  R::useJSONFeatures(TRUE);  header('Content-type: json/aplication');  if (!R::testConnection()) {  echo 'нет подключения';  }  $recivedData = json\_decode(file\_get\_contents("php://input"));  $func = new Functions;  if ($recivedData->action == 'getTableNames') {  $tables = R::getAll('SHOW TABLES');  foreach ($tables as $table) {  $data[] = $table;  }  echo json\_encode(($data));  }  if ($recivedData->action == 'deleteRowById') {  $fromVueParamId = $recivedData->selectedId;  $fromVueParamTableName = $recivedData->selectedTableName;  ///$rows = deleteRowById($fromVueParamId, $fromVueParamTableName);  $rows = $func->deleteRowById($fromVueParamId, $fromVueParamTableName);  foreach ($rows as $row) {  $data[] = $row;  }  echo json\_encode(($data));  }  if ($recivedData->action == 'insertIntoTable') {  $fromVueParamTableName = $recivedData->selectedTableName;  $fromVueInsertData = $recivedData->insertInto;  echo $func->insertIntoTable($fromVueParamTableName, $fromVueInsertData);  }  if ($recivedData->action == 'updateTableRow') {  $fromVueParamTableName = $recivedData->selectedTableName;  $fromVueInsertData = $recivedData->insertInto;  $fromVueParamTableName = $recivedData->selectedTableName;  $fromVueRowId = $recivedData->rowId;  $fromVueUpdateData = $recivedData->updateData;  echo $func->updateTableRow($fromVueParamTableName, $fromVueRowId, $fromVueUpdateData);  }  if ($recivedData->action == 'registCompanyAndAdmCompany') {  $fromVueDataCompany = $recivedData->companyData;  $fromVueDataAdmin = $recivedData->adminData;  echo $func->registCompanyAndAdmCompany($fromVueDataCompany, $fromVueDataAdmin);  }  if ($recivedData->action == 'authorizationUser') {  $fromVueDataAuth = $recivedData->authForm;  //var\_dump($fromVueDataAuth->userLogin);  $respons = $func->chekUserAndAuthorizationUser($fromVueDataAuth);  $userAndCompanyInfo = array("userInfo" => $respons[0], "companyInfo" => $respons[1]);  echo json\_encode($userAndCompanyInfo);    }  if ($recivedData->action == 'createUser') {  $fromVueNewUserData = $recivedData->newUserData;  $fromVueUserCompanyData = $recivedData->companyData;  echo $func->createUser($fromVueNewUserData, $fromVueUserCompanyData);  }  if ($recivedData->action == 'getCompanyUsers') {  $fromVueCompanyid = $recivedData->companyId;    echo json\_encode($func->getCompanyUsers($fromVueCompanyid));  }  if ($recivedData->action == 'createNewTask') {  $fromVueTask = $recivedData->newTaskData;  $fromVueFromUserid = $recivedData->fromUser;  echo json\_encode($func->createNewTask($fromVueTask,$fromVueFromUserid));  }  if ($recivedData->action == 'getCreatedTask') {  $fromVueFromUserid = $recivedData->userId;  echo json\_encode($func->getCreatedTask($fromVueFromUserid));  }  if ($recivedData->action == 'getUserById') {  $fromVueFromUserid = $recivedData->userId;  echo json\_encode($func->getUserById($fromVueFromUserid));  }  if ($recivedData->action == 'getMyTasks') {  $fromVueFromUserid = $recivedData->userId;  echo json\_encode($func->getMyTasks($fromVueFromUserid));  }  if ($recivedData->action == 'setTaskStatus') {  $fromVueid = $recivedData->taskId;  $fromVueStatus = $recivedData->status;  echo json\_encode($func->setTaskStatus($fromVueid,$fromVueStatus));  }  if ($recivedData->action == 'countTimeTask') {  $fromVueid = $recivedData->taskId;  $fromVueStatus = $recivedData->status;  $taskTime = $func->countTimeTask($fromVueid,$fromVueStatus);  echo json\_encode($taskData);  }  //countTimeTask |