

Содержание

[Введение 6](#_Toc153572201)

[1 Постановка задачи 7](#_Toc153572202)

[1.1 Обзор аналогов 7](#_Toc153572203)

[1.1.1 Веб приложение «Toggl» 7](#_Toc153572204)

[1.1.2 Веб приложение «Harvest» 8](#_Toc153572205)

[1.1.3 Веб приложение «Hubstaff» 9](#_Toc153572206)

[1.2 Актуальность задачи 11](#_Toc153572207)

[1.3 Постановка целей 12](#_Toc153572208)

[2 Разработка 13](#_Toc153572209)

[2.1 Разработка модели базы данных 13](#_Toc153572210)

[2.2 Диаграмма вариантов использования 14](#_Toc153572211)

[2.3 Проектирование основных алгоритмов 16](#_Toc153572212)

[3 Разработка программного средства 18](#_Toc153572213)

[3.1 Разработка серверной части 18](#_Toc153572214)

[3.2 Разработка клиентской части 22](#_Toc153572215)

[3.3 Развертывание приложения на базе Docker 24](#_Toc153572216)

[3.4 Вывод 24](#_Toc153572217)

[4 Тестирование программного средства 25](#_Toc153572218)

[4.1 Вывод 27](#_Toc153572219)

[5 Руководство пользователя 28](#_Toc153572220)

[5.1 Руководство менеджера 29](#_Toc153572221)

[5.2 Руководство администратора 30](#_Toc153572222)

[5.3 Вывод 31](#_Toc153572223)

[Заключение 32](#_Toc153572224)

[Список используемых источников 33](#_Toc153572225)

Введение

Веб-приложение для учета рабочего времени разрабатывается с целью упростить и автоматизировать процесс отслеживания рабочего времени. Оно предоставляет возможности регистрации пользователей, управления пользователями и проектами, отслеживания времени работы и анализа статистики. Это приложение позволит организациям более эффективно управлять рабочим процессом и оптимизировать использование ресурсов.

Для создания веб-приложения выбраны JavaScript и платформа Node.js. JavaScript является одним из самых популярных языков программирования, который широко используется для разработки клиентской и серверной частей веб-приложений. Он обладает большим сообществом разработчиков, обширной экосистемой инструментов и библиотек, что облегчает разработку и поддержку приложения.

Платформа Node.js предоставляет возможность запускать JavaScript на сервере, что позволяет разрабатывать эффективные и масштабируемые веб-приложения. Она основана на событийно-ориентированной и неблокирующей архитектуре, что способствует высокой производительности и отзывчивости приложения при обработке большого количества запросов.

Выбор JavaScript и Node.js обеспечивает единообразие языка программирования на клиентской и серверной сторонах, что упрощает разработку и поддержку кода. Кроме того, наличие богатой экосистемы инструментов и библиотек для JavaScript и Node.js позволит эффективно реализовать требуемую функциональность и обеспечить безопасность и стабильность приложения.

В качестве инструмента для разработки интерфейса выбран фреймворк React. React — это популярная JavaScript-библиотека, которая позволяет разрабатывать эффективные и масштабируемые пользовательские интерфейсы. Она основана на компонентной модели, что позволяет создавать переиспользуемые компоненты и эффективно управлять состоянием интерфейса.

Целью данной курсовой работы является разработка эффективного веб-приложения по отслеживанию времени с упором на простоту и удобство использования. Основной упор будет сделан на создание интуитивно понятного и легкого в обращении инструмента, который позволит пользователям управлять своим временем.

1. Постановка задачи
   1. Обзор аналогов
      1. Веб приложение «Toggl»

Первым аналогом для обзора было выбрано веб-приложение «Toggl» [1], изображений на рисунке 1.1.

Стоимость: бесплатно до 5 пользователей (ограниченные возможности); $10 в месяц за пользователя (стартовый); $20 в месяц за пользователя (премиум).

Платформа: веб-сервис.

Язык: английский.

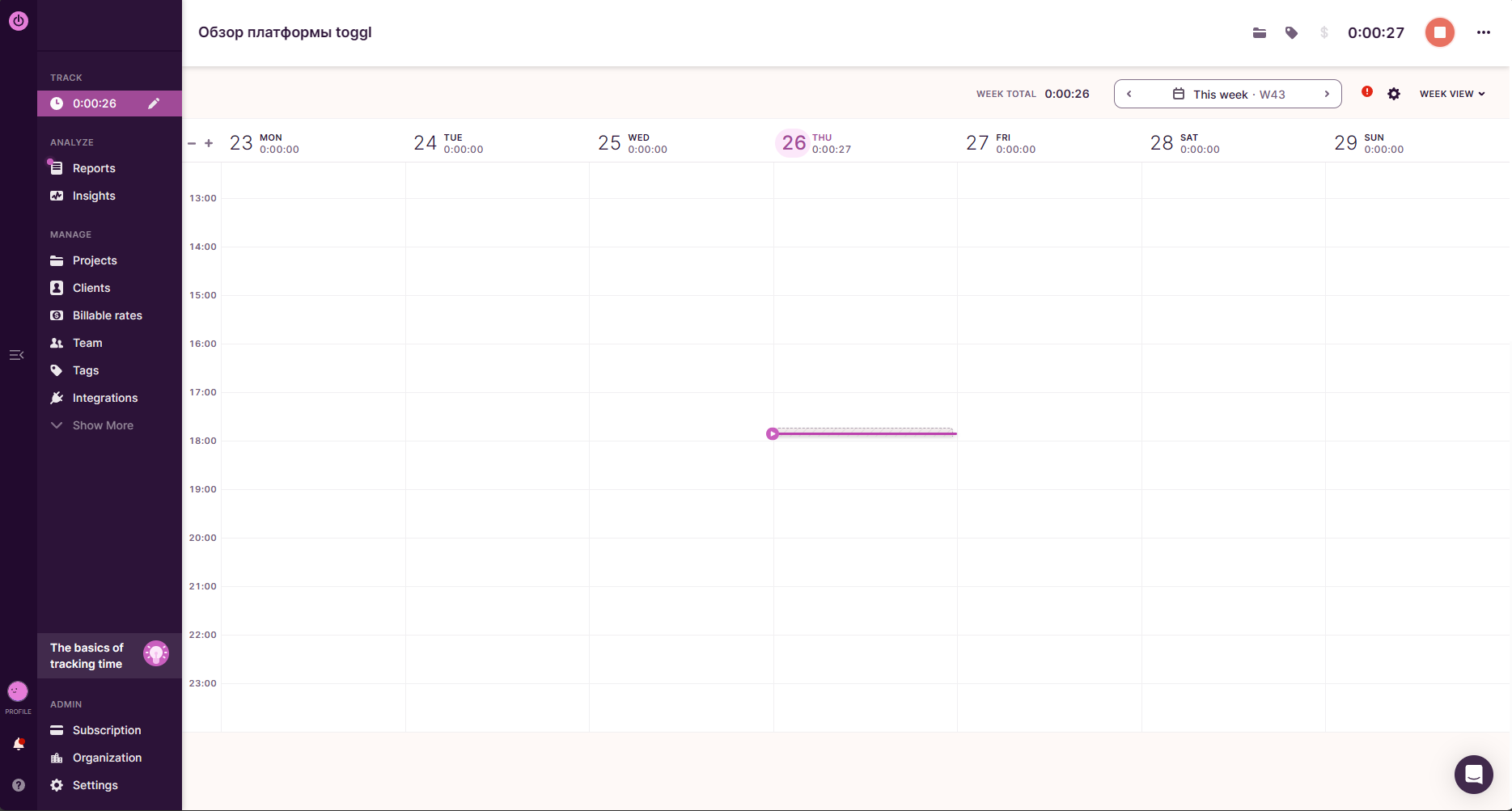


Рисунок 1.1 – Веб-сервис «Toggl»

Toggl — это популярное веб-приложение и инструмент учета времени, который позволяет пользователям отслеживать время, затраченное на различные задачи и проекты. Оно предоставляет функциональность для регистрации времени работы, анализа статистики и генерации отчетов.

Пользователи Toggl могут создавать таймеры для отслеживания времени, начинать и останавливать таймеры при переключении между задачами или проектами. Они также могут добавлять описания, теги и категории к каждому таймеру для более подробного анализа и организации данных.

Toggl предоставляет пользовательский интерфейс, который обеспечивает удобство использования и навигации. Он также предлагает функциональность для группировки задач и проектов, создания отчетов о затраченном времени, анализа производительности и мониторинга временных трендов.

Преимущества:

* Красивый UX/UI дизайн, интуитивный интерфейс.
* Соотношение цена-качество.
* Неограниченное количество клиентов, проектов и отчетов (даже для бесплатных пользователей).
* Подходит для управления личными и командными проектами.
* Управление клиентами.
* Создание ставок оплаты.
* Интеграция с большим количеством сервисов (100+).
* Кроссплатформенность, есть клиенты для iOS, Android, Mac, Windows, Linux.

Недостатки:

* Нет русского языка.

Вывод: доступный инструмент с современным дизайном для отслеживания времени с большим количеством интеграций и возможностей в бесплатной версии.

* + 1. Веб приложение «Harvest»

Вторым аналогом было выбрано веб-приложение «Harvest» [2], интерфейс которого отображен на рисунке 1.2.

Стоимость: бесплатно для одного пользователя, до двух проектов; платная 12 в месяц за пользователя.

Платформа: веб-сервис.

Язык: английский.

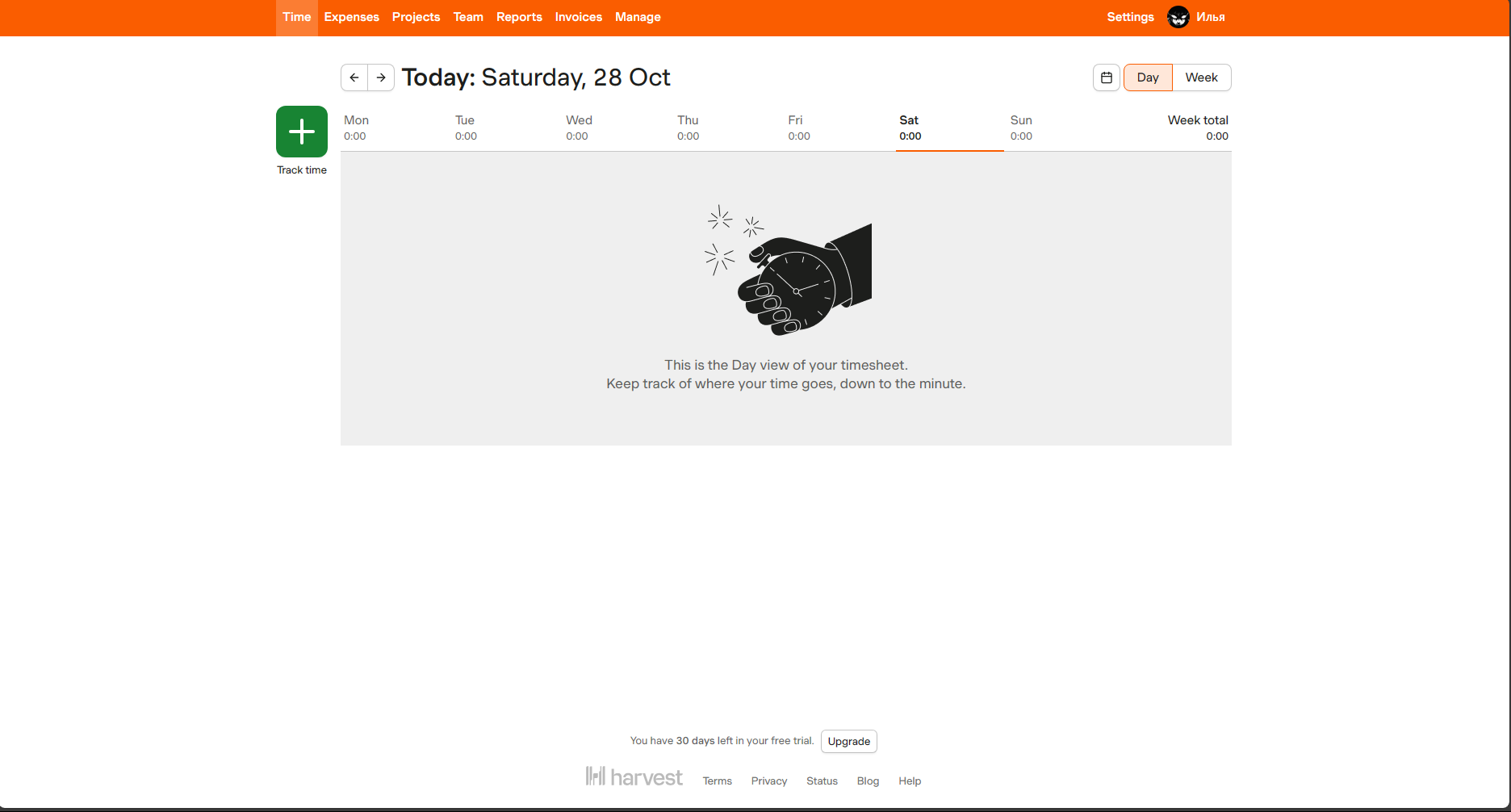


Рисунок 1.2– Веб-сервис «Harvest»

Harvest – еще один лидер в области учета рабочего времени. Как и в Toggl, здесь есть бесплатный тарифный план, но он ограничивается одним пользователем и двумя проектами. С другой стороны, вы получаете массу возможностей интеграции с системами управления проектами, бухгалтерского учета, CRM и другими инструментами.

Кроме того, есть мобильные приложения для учета времени для iOS и Android, а также расширения для Chrome и Safari. Можно также ограничиться настольной версией.

Преимущества:

* Красивый UX/UI дизайн, интуитивный интерфейс.
* Подходит для управления личными и командными проектами.
* Управление клиентами
* Автоматизированное выставление счетов
* Создание ставок оплаты
* Интеграция с большим количеством сервисов ,80+
* Кроссплатформенность, есть клиенты для iOS, Android, Mac,Windows.
* Недостатки:
* Нет русского языка
* Бесплатная ограничивается одним пользователем и двумя проектами.

Вывод:приложение с современным интерфейсом и высокой функциональностью, имеет большое количество интеграций подробные отчеты.

* + 1. Веб приложение «Hubstaff»

Последним аналогом для обзора было выбрано веб-приложение «Hubstaff» [3], интерфейс которого отображен на рисунке 1.3.

Стоимость: бесплатно для одного пользователя (ограниченные возможности); 5$. в месяц для одного пользователя (базовые возможности).

Платформа: веб-сервис.

Язык: английский.

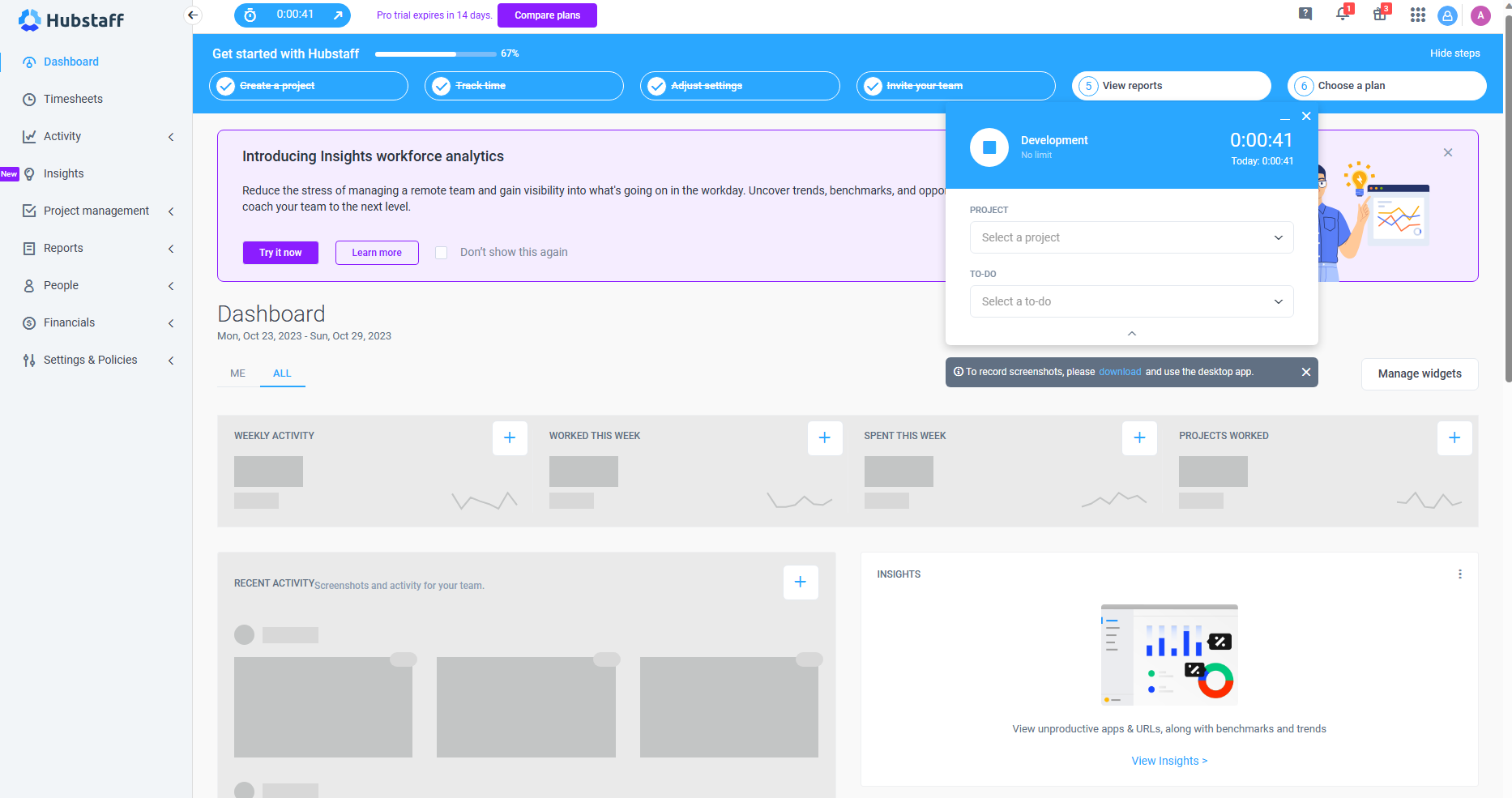


Рисунок 1.3 – Веб сервис «Hubstaff»

Hubstaff – приложение для учета рабочего времени, предназначенное для работодателей, чтобы отслеживать продуктивность работы членов команды.

Hubstaff помогает вам убедиться, что сотрудники рационально используют свое время.

Основная задача Hubstaff – убедиться в том, что команда раскрывают свой потенциал, по крайней мере, с точки зрения управления временем. Система обнаруживает, когда пользователь слишком долго бездействует, и выдает соответствующее сообщение, побуждая вернуться к работе или остановить таймер и отредактировать свою запись.

Есть возможность отслеживать по GPS команду. Hubstaff даже делает случайные скриншоты активности пользователей, а также отслеживает нажатия клавиш в приложении, чтобы вы могли видеть, кто активно использует (или не использует) инструмент.

Преимущества:

* Красивый UX/UI дизайн.
* Подходит для управления личными и командными проектами.
* GPS трекинг.
* Автоматизированное выставление счетов
* Создание ставок оплаты
* Подробные отчеты
* Кроссплатформенность, есть клиенты для iOS, Android, Mac, Windows, Linux
* Недостатки
* Нет русского языка
* Сложный интерфейс

Вывод:приложение имеет широкий функционал и глубокую настройку. Отлично подходит для наблюдения и оптимизации работы команды.

* 1. Актуальность задачи

Разработка веб-приложения по отслеживанию времени остается актуальной и полезной, поскольку:

1. Позволяет учет времени и оплату труда: Веб-приложение позволяет пользователям точно отслеживать время, затраченное на задачи и проекты, что важно для учета рабочего времени и оплаты труда.

2. Увеличивает производительность и эффективность: Отслеживание времени помогает выявить, какие задачи занимают больше времени, что позволяет оптимизировать рабочий процесс и повысить производительность.

3. Управление проектами и задачами: Веб-приложение предоставляет инструменты для управления проектами, установления сроков, определения приоритетов и контроля выполнения задач.

4. Создание аналитики и отчетности: Веб-приложение предоставляет возможность анализировать данные о времени, создавать отчеты и диаграммы, которые помогают оценить эффективность работы и принять информированные решения.

В целом, разработка веб-приложения по отслеживанию времени актуальна, поскольку она помогает учету рабочего времени, повышает производительность и эффективность, облегчает управление проектами и создание аналитики для принятия информированных решений.

* 1. Постановка целей

Анализ веб-приложений, в основном, сводится к удобству пользовательского интерфейса, а также реализации всех функций необходимых пользователю. В данном разделе был произведен обзор аналогов и рассмотрены их плюсы и минусы. Каждый аналог был подробно описан, и было показано, для чего предназначено то или иное программное обеспечение.

Исходя из сделанных выводов при анализе аналоговых приложений было решено реализовать следующие ключевые функциональные возможности:

* регистрация новых пользователей;
* реализация роли студента и преподавателя;
* возможность создания проектов;
* возможность создания команд;
* возможность добавление задач командам;
* просмотр статистики.
  1. Вывод

Из представленного краткого обзора становится ясно, что платформы отслеживания времени обладают множеством общих функциональных возможностей. Однако различия в подходах каждой из них заметны. В результате такого многообразия пользователю часто бывает сложно определиться с выбором подходящей платформы.

Подводя итог рассмотрению указанных приложений, можно сделать следующий вывод: каждая из них обладает уникальным дизайном, функционалом и возможностями. Однако, как и везде, существуют как положительные стороны, так и недостатки. Разработка оптимального продукта предполагает выбор определенных возможностей из конкурентных платформ с последующим устранением выявленных недостатков.

1. Проектирование веб-приложения
   1. Выбор платформы и технологий
      1. Выбор средств реализации для серверной части

В контексте разработки серверной части приложения было принято решение использовать Node.js. Это высокопроизводительная и масштабируемая среда выполнения JavaScript, построенная на движке V8. Основными преимуществами Node.js являются асинхронная обработка и событийно-управляемая архитектура, что позволяет эффективно обрабатывать множество одновременных запросов и обеспечивать отзывчивость приложения.

* + 1. Выбор средств реализации для клиентской части

В качестве фреймворка для разработки клиентской части приложения был выбран React. Это библиотека JavaScript, созданная Facebook, которая обеспечивает декларативное и компонентное строение пользовательского интерфейса. React обеспечивает высокую производительность и удобство разработки благодаря виртуальному DOM и эффективному механизму обновления компонентов.

* + 1. Выбор средств реализации для базы данных

В качестве системы управления базами данных (СУБД) была выбрана MongoDB. MongoDB является NoSQL базой данных, использующей документоориентированную модель данных. Её гибкость позволяет хранить данные в формате JSON-подобных документов, что особенно удобно для разработки приложений с изменяющейся схемой данных. MongoDB также обеспечивает горизонтальное масштабирование, что важно для обеспечения производительности в случае увеличения объема данных.

* 1. Разработка модели базы данных

Целью данной базы данных является управление проектами и задачами в организации. База данных позволяет хранить информацию о компаниях, пользователях, проектах, задачах, командах, а также отслеживать время выполнения задач и историю действий пользователей. Схема таблицы представлена на рисунке 2.1.

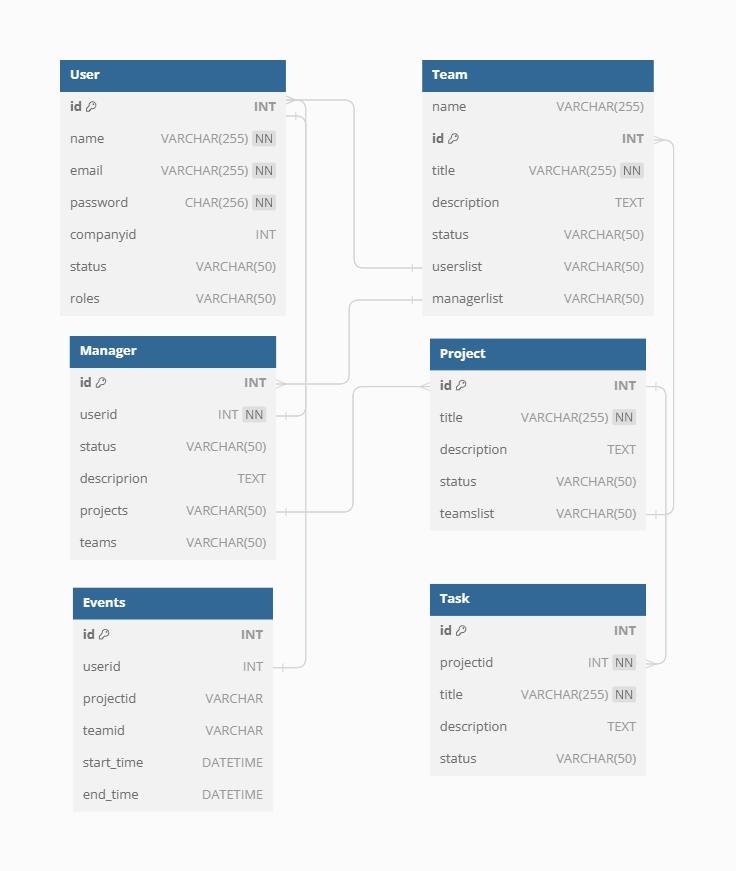


Рисунок 2.1 – Схема базы данных

Приведенная схема базы данных имеет следующую структуру:

Таблица «Users» содержит информацию о пользователе:

* id: Уникальный идентификатор пользователя;
* name: Имя пользователя;
* email: Адрес электронной почты пользователя;
* password: Хэш пароля пользователя;
* teams: Список участия пользователя в командах;
* status: Статус пользователя;
* role: Список ролей пользователя

Таблица «Team» хранит информацию о командах:

* id: Уникальный идентификатор команды;
* title: Название команды;
* description: Описание команды;
* status: Статус команды;
* userlist: Список пользователей;
* manager: Менеджер команды.

Таблица «Project» содержит информацию о проектах:

* id: Уникальный идентификатор проекта;
* title: Название проекта;
* description: Описание проект;
* status: Статус проекта;
* teamlist: Список команд.

Таблица «Task» хранит информацию о задачах:

* id: Уникальный идентификатор задачи;
* projectid: Идентификатор проекта, к которому относится задача;
* title: Название задачи;
* description: Описание задачи;
* status: Статус задачи;
* assignedTo: Кто работает над этой задачей.

Таблица «Events» хранит историю выполнения задач пользователям:

* id: Уникальный идентификатор события;
* userid: Идентификатор проекта, к которому относится задача;
* projectid: Идентификатор проекта, к которому относилась задача;
* teamid: Идентификатор команды к которому относилась задача;
* start\_time: Время начала задачи;
* end\_time: Время окончания задачи.

Схема базы данных содержит следующие связи:

* связь «User.id - Manager.userid»:один к одному;
* связь «Task.projectid - Project.id»:один к одному;
* связь «Events.userid - User.id»:один к одному;
* связь «Events.projectid - Project.id»:один к одному;
* связь « Events.teamid - Team.id»:один к одному;
* связь «Team.manager < Manager.id»: один ко многом;
* связь «Project.teamslist < Team.id»: один ко многом;
* связь «Manager.projects < Project.id»: один ко многом;
* связь «Task.assignedTo < User.id»: один ко многом;
* связь «Team.userslist < User.id»: один ко многом.

Все эти связи позволяют связывать данные из разных таблиц, создавая структуру базы данных, которая отражает взаимосвязи между, пользователями, командами, проектами и задачами.

* 1. Диаграмма вариантов использования

Диаграмма вариантов использования позволяет представить все доступные варианты использования приложения и взаимодействия пользователей с различными ролями, такими как администратор, пользователь и менеджер.

Роль «Пользователь» предоставляет функциональность для обычных пользователей, позволяющие выполнять задачи, создавать личные проекты и отчетность.

Роль «Менеджер» предоставляет функциональность по созданию и управлению, проектами и командами.

Роль «Администратор» предоставляет функциональность для администраторов менеджерами и пользователями.

На диаграмме вариантов использования, обозначенной на рисунке 2.2, отображается основная логика взаимодействия с системой.

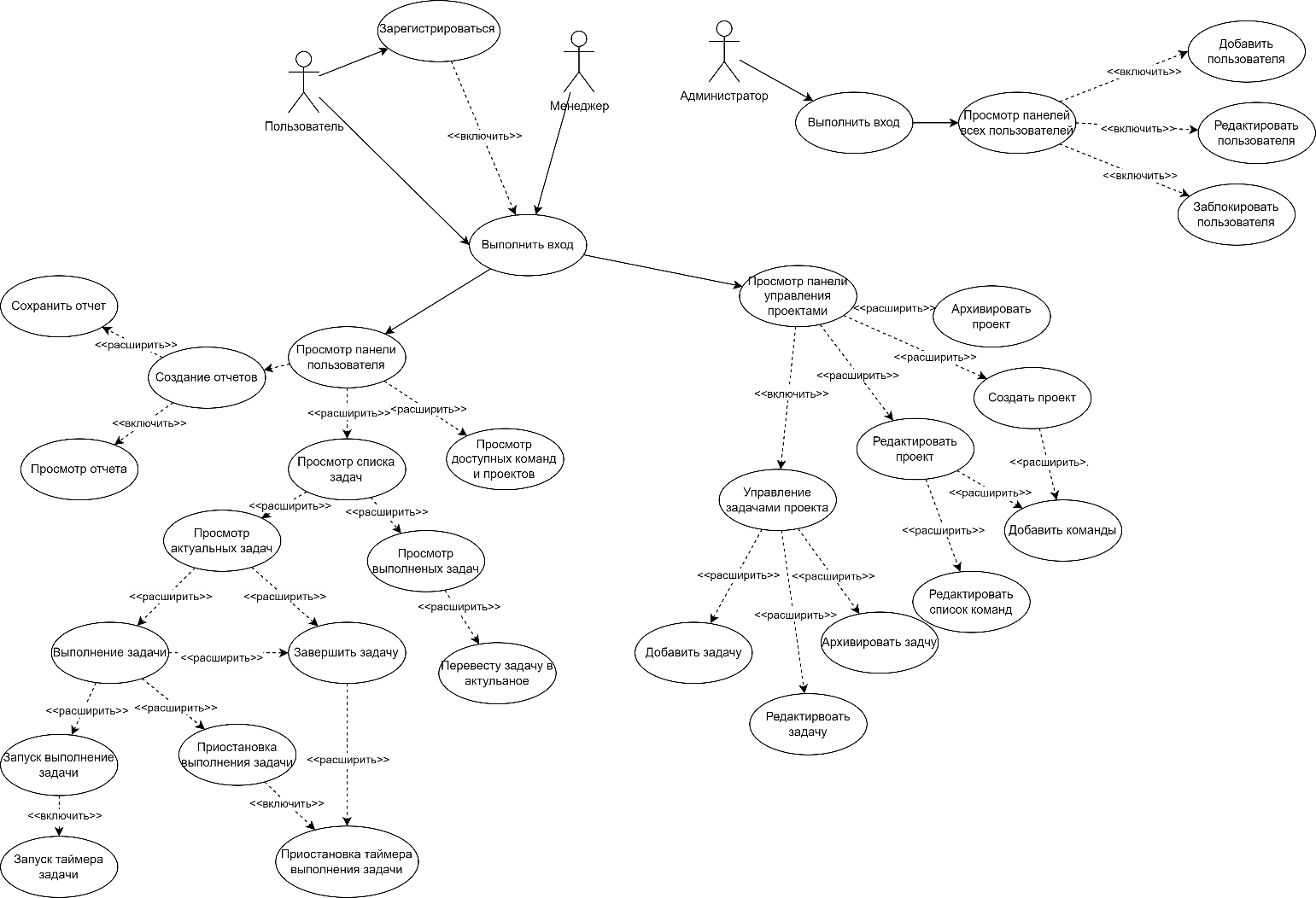


Рисунок 2.2 – Диаграмма использования

* 1. Проектирование основных алгоритмов

Блок-схемы алгоритмов являются визуальным инструментом, который помогает разработчикам понять, проектировать и отлаживать алгоритмы. Они представляют собой графическое представление последовательности шагов, условий и операций, которые выполняются в алгоритме.

Первым алгоритмом является создание команд. Данный алгоритм выполняется менеджером и представлен на рисунке 2.3

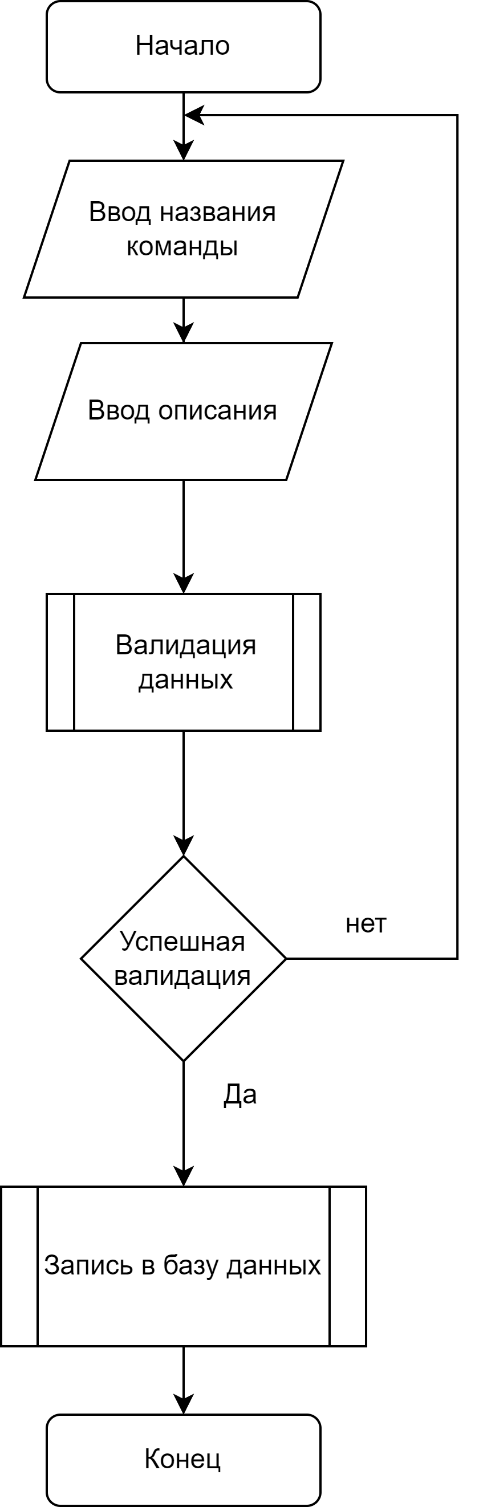


Рисунок 2.3 – Схема алгоритма создание команды

Вторым алгоритмом будет запуск таймера для подсчета, времени выполнения пользовательской задач. Алгоритм подсчета времени можно наблюдать на рисунке 2.4.

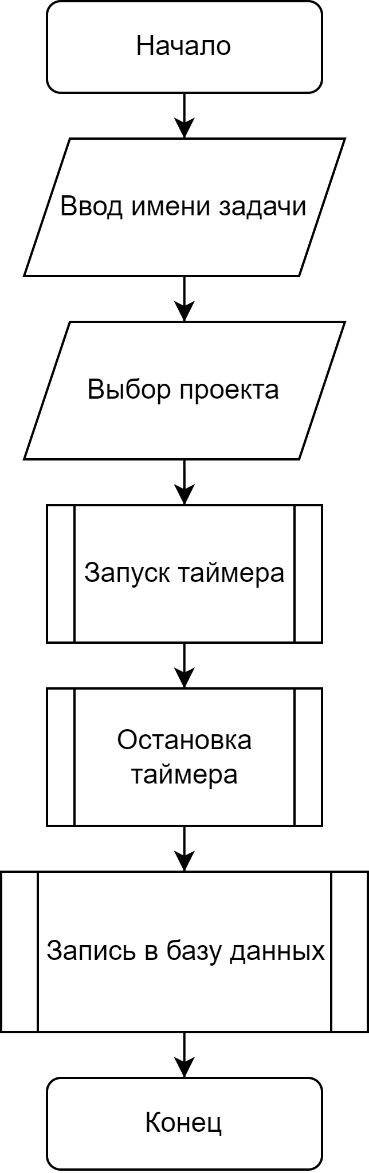


Рисунок 2.4 – Алгоритм подсчета времени задачи

* 1. Вывод

Таким образом в конце данной главы была спроектирована база данных, описаны UML-диаграммы вариантов использования для разных типов пользователей, были описаны алгоритмы подсчета времени, и создания команды, а также определен стек технологий, используемый в данном проекте.

1. Разработка программного средства

Приложение, которое разрабатывается, представляет собой веб-сервис трекера времени, использующий технологии Node.js, React и MongoDB. Для обеспечения взаимодействия между клиентской и серверной частями приложения выбран архитектурный стиль RESTful API.

RESTful API было использовано для установления стандартных методов передачи данных через HTTP (GET, POST, PUT, DELETE) между клиентской и серверной частями. Четкие URL-адреса ресурсов позволили легко осуществлять доступ к различным компонентам приложения, таким как задачи, пользователи и временные записи.

Приложение использует элементы модели MVC. Файлы API с функциями запросов на клиентской стороне (React) выполняют функцию контроллера, обеспечивая логику и взаимодействие с сервером. На сервере присутствуют модели данных и (Routes), которые подобны компонентам модели (Model) и контроллера (Controller) соответственно. Это не строгая реализация MVC.

* 1. Разработка серверной части

Структура серверной части приложения состоит из папок «model» и «routes» (рисунок 3.1). В папке «model» содержатся модели данных, определяющие схемы и структуру информации, а также функции для работы с данными. Описываются сущности данных, которые хранятся в базе данных и определяют способы взаимодействия с этими данными через запросы на сервер.

В папке «routes» содержатся URL пути для различных запросов клиентов к серверу. Пути обрабатывают входящие запросы и определяют, какие действия выполнять, направляя их к соответствующим функциям в модели данных для выполнения операций с данными.

Таким образом, разделение на «model» и «routes» позволяет организовать приложение с учетом MVC-подобной структуры, где «model» функционирует как компонент Model, определяющий структуру данных и их обработку, а «routes» подобны контроллерам (Controller), обрабатывающим запросы и направляющим их к соответствующим функциям в модели данных для выполнения операций с данными.

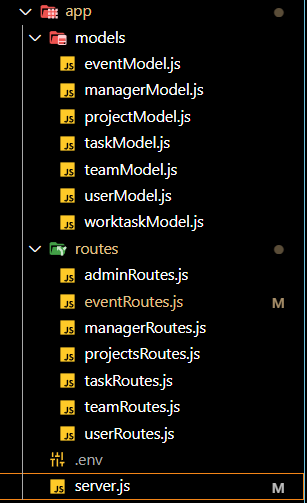


Рисунок 3.1 – Структура проекта

Директория «Models» содержит определения моделей для таблиц базы данных. Они служат основой для взаимодействия сервера с базой данных. Каждая модель соответствует примеру, представленному в листинге 3.2.

const worktaskSchema = new Schema({

    projectId: {

        type: String,

        ref: 'Project'

    },

    title: {

        type: string,

        default: 'Новая задача',

        required: true

    },

    status: {

        type: string,

        required: true,

        default: 'active'

    },

    description: {

         type: string,

        default: 'Описание задачи'},});

Листинг 3.2 – Структура сущности

Папка «Queries» предназначена для обработки запросов пользователей приложения к данным базы данных, чтобы можно было на основе этих запросов получить определенную информацию. Пример одного из запросов проекта к базе данных, для получения команд пользователя, представлен в листинге 3.3.

router.get('/my', async (req, res) => {

    try {

        const user = await User.findById(req.user);

        const login = user ? user.login : null;

        const teams = await Team.find({ manager: login });

        res.json(teams);

    } catch (error) {

        res.status(500).json({ message: error.message });

    }

});

Листинг 3.3 – Структура запроса

Файл server.js, представленный в листинге 3.4 — это основной файл серверного приложения на Node.js с использованием Express. В файле происходит подключение необходимых модулей, таких как Express, CORS и Mongoose для работы с MongoDB. Определены маршруты для различных сущностей (пользователи, администраторы, менеджеры, проекты, команды, задачи, события). Подключаются переменные окружения из файла .env. Express используется для обработки CORS и парсинга JSON. Устанавливается соединение с базой данных MongoDB. Маршруты привязаны к соответствующим путям (например, «/api» для пользовательских маршрутов). Приложение запускается на указанном порту из переменной окружения.

const express = require("express");

const cors = require("cors");

const mongoose = require("mongoose");

const userRoutes = require("./routes/userRoutes.js");

const adminRoutes = require("./routes/adminRoutes.js");

const managerRoutes = require("./routes/managerRoutes.js");

const projectsRoutes = require("./routes/projectsRoutes.js");

const teamRoutes = require("./routes/teamRoutes.js");

const taskRoutes = require("./routes/taskRoutes.js");

const eventRoutes = require("./routes/eventRoutes.js");

require("dotenv").config(); // Загрузка переменных окружения из файла .env

const app = express();

app.use(cors());

const PORT = process.env.PORT;

// Подключение к базе данных MongoDB из переменной окружения

mongoose

  .connect(process.env.MONGODB\_URI)

  .then(() => {

    console.log("Connected to MongoDB");

  })

  .catch((err) => {

    console.error("Error connecting to MongoDB:", err.message);

  });

// Определение middleware для парсинга JSON

app.use(express.json());

app.use("/api", userRoutes);

app.use("/admin", adminRoutes);

app.use("/manager", managerRoutes);

app.use("/project", projectsRoutes);

app.use("/team", teamRoutes);

app.use("/task", taskRoutes);

app.use("/event", eventRoutes);

app.get("/", (req, res) => {

  res.send("Сервер работает!");

});

// Запуск сервера

app.listen(PORT, () => {

  console.log(`Server is running on port ${PORT}`);

});

Листинг 3.4 – Файл server.js

Для реализации аутентификации и авторизации в приложении была использована технология JSON Web Token (JWT), предоставляющая эффективный и безопасный способ передачи информации о пользователях между клиентом и сервером после успешной аутентификации.

Для создания токена была разработана функция «generateAccessToken» (листинг 3.5), которая принимает объект пользователя в качестве аргумента. С использованием библиотеки «jsonwebtoken», в токен включаются уникальный идентификатор пользователя и его роль. Для подписи токена используется секретный ключ, а время жизни токена установлено на 3 дня.

function generateAccessToken(user) {

    const token = jwt.sign({ id: user.\_id, role: user.roles },

'secret-key', { expiresIn: '3d' });

    return token;}

Листинг 3.5 – Функция генерации JWT

Для проверки валидности токена была реализована функция-посредник «checkToken». Данная функция извлекает токен из заголовка запроса и, в случае его отсутствия, возвращает статус 401 с сообщением «No token provided». При наличии токена, используя «jwt.verify», производится его верификация с использованием секретного ключа. В случае ошибки, возвращается статус 500 с сообщением «Failed to authenticate token». При успешной верификации, информация из токена (идентификатор пользователя и роль) добавляется к объекту запроса для дальнейшего использования.

function checkToken(req, res, next) {

    const token = req.headers['authorization'].split(' ')[1];

    if (!token) {

        return res.status(401).json({ isValid: false, message: 'No token provided.' });

    }

    jwt.verify(token, 'your-secret-key', (err, decoded) => {

        if (err) {

            return res.status(500).json({ isValid: false, message: 'Failed to authenticate token.' });

        }

        req.user = decoded.id;

        req.role = decoded.role;

        next();

    });}

router.use(checkToken);

Листинг 3.6 – Функция проверки JWT

Эта реализация обеспечивает безопасную аутентификацию пользователей в приложении, где каждый токен предоставляет уникальные идентификаторы и роли пользователей, а также поддерживается сроком действия токена в 3 дня для обеспечения безопасности.

* 1. Разработка клиентской части

В разработке клиентской части приложения была выбрана библиотека React, предоставляющая эффективные средства для создания интерфейса пользователя. Структура проекта представлена на рисунке 3.7

. 

Рисунок 3.7 – Структура приложения клиентской части

В самом проекте можно выделить несколько основных директорий:

* src/App.js: Основной файл приложения, где происходит сборка компонентов и установка основных настроек.
* src/pages/: Директория, содержащая отдельные файлы для каждой страницы приложения. Например, файлы LoginPage.jsx, LoginPage.jsx, и другие.
* src/services/: Здесь расположены функции для выполнения запросов к серверу. Например, файлы apiEvents.js, apiUser.js, обеспечивающие взаимодействие с серверным API и авторизацией.

Остальные директории в src/: Включают компоненты React и их стили. Каждый компонент оформлен в собственной директории, что способствует легкости разработки и поддержки.

В рамках проекта были разработаны API-функции, предназначенные для взаимодействия с сервером. Это подход обеспечивает четкую структуру кода, легкость поддержки и масштабирования, а также облегчает обработку ответов и ошибок. В листинге 3.8 представлен пример запроса к серверу для получения данных о событиях.

export function getEvents() {

    return fetch(`${API\_URL}/event/my`, {

        method: 'GET',

        headers: { 'Content-Type': 'application/json' ,

        'Authorization': `Bearer ${localStorage.getItem('authToken')}`},

    })

    .then(response => {

        if (!response.ok) {

            throw new Error(`HTTP error! status: ${response.status}`);

        }

        return response.json();

        })

        .catch(error => {

            // Handle any errors

            throw new Error(error.message);

        });

}

Листинг 3.8 – Функция запроса на сервер

Этот пример иллюстрирует использование API-функции getEvents(). Она выполняет GET запрос к /event/my, включая в заголовках авторизационный токен. Это обеспечивает безопасность взаимодействия с сервером. Использование таких API-функций в компонентах приложения способствует более прозрачной и удобной разработке.

В приложении использован «react-router-dom» для маршрутизации между различными страницами. Реализован механизм контроля доступа к определенным роутам через компонент «RouteWrapper», который осуществляет проверку аутентификации пользователя и соответствия его роли ожидаемым значениям. В случае успешной аутентификации и соответствия роли, дочерние компоненты рендерятся, иначе происходит перенаправление на страницу входа. Для проверки валидности токена и ролей используется функция «checkAuthentication». Этот подход обеспечивает безопасный и структурированный доступ пользователя к функционалу приложения. Реализация представлена в app.js приложения А.

* 1. Развертывание приложения на базе Docker

Docker представляет собой открытую платформу для автоматизации развертывания, масштабирования и управления приложениями в контейнерах. Контейнеры обеспечивают легковесное, переносимое и самодостаточное окружение, что упрощает процессы разработки и доставки программного обеспечения.

В проекте был создан Dockerfile для клиентского приложения с использованием Node.js. Этот файл определяет шаги для построения Docker-образа, включая установку зависимостей и конфигурацию окружения. преставленым в листинге 3.9.

FROM node:14

# Устанавливаем рабочую директорию внутри контейнера

WORKDIR /usr/src/app

# Копируем зависимости и файлы проекта в контейнер

COPY package\*.json ./

# Устанавливаем зависимости

RUN npm install

# Копируем остальные файлы приложения

COPY . .

# Определяем команду для запуска приложения

CMD ["npm", "start"]

Листинг 3.9 – DockerFile клиентской части приложения

Для создания Docker-образов и запуска контейнеров приложением выполнены следующие команды:

Сборка Docker-образа: «docker build -t имя\_образа ./путь \_приложению».

Запуск Docker-контейнера: «docker run -p 3000:3000 имя\_образа»

Для всех частей используются собственные названия и порты. Листинг dockerfile для базы данных и серверной части приложения представлены в Приложении Б.

* 1. Вывод

В результате выполненных работ в данном разделе было успешно разработано приложение, полностью соответствующее заранее определенному функционалу. Завершающим этапом в процессе разработки стало создание контейнера для программного средства, обеспечивающего удобное развертывание приложения. В следующем этапе, который является не менее важным, планируется провести тестирование приложения. Этот этап предполагает обширное тестирование в основных областях функционала для выявления и устранения потенциальных ошибок перед выпуском приложения для конечных пользователей.

1. Тестирование программного средства

Тестирование приложения – важный этап разработки, направленный на выявление и устранение ошибок, а также подтверждение соответствия функционала заданным требованиям. Оно обеспечивает надежность и стабильность работы приложения, позволяет выявлять потенциальные проблемы на ранних этапах разработки и обеспечивает защиту от нежелательных изменений.   
 Рассмотрим вариант регистрации пользователя, когда он вводит логин, который уже существует. При заполнении формы неправильными данными должно быть получено сообщение об ошибке. Заполнение формы неправильными данными изображено на рисунке 4.1.

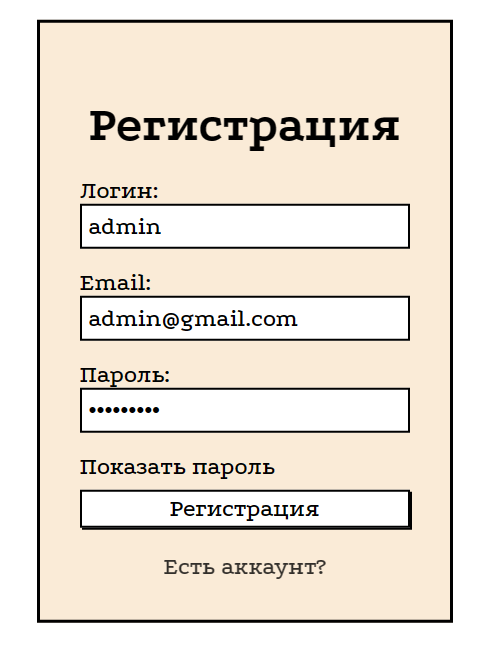


Рисунок 4.1 – Форма регистрации с занятыми данными

После попытки регистрации с уже имеющимся данными, пользователю выводится ошибка, изображенная на рисунке 4.2.

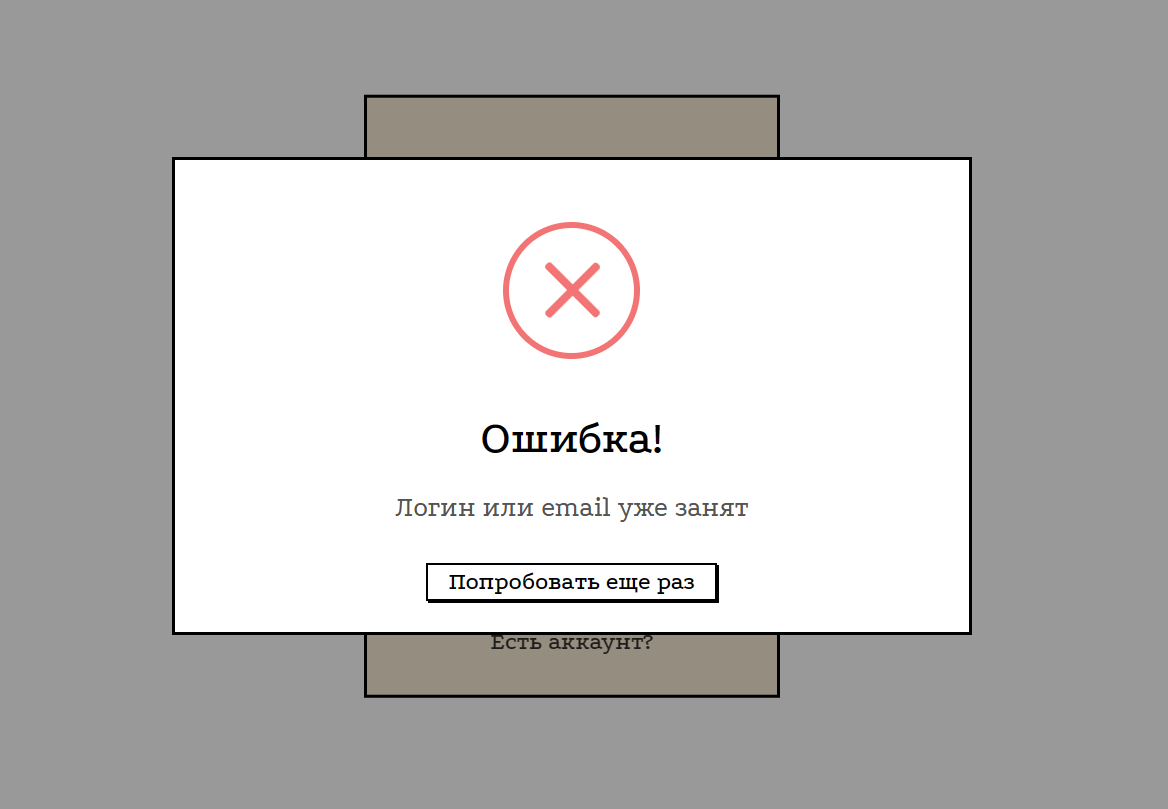


Рисунок 4.2 – Ошибка при регистрации

После успешной регистрации с валидными и уникальными данными, пользователь получит сообщение о успешной регистрации. Пример успешной регистрации представлен на рисунке 4.3.

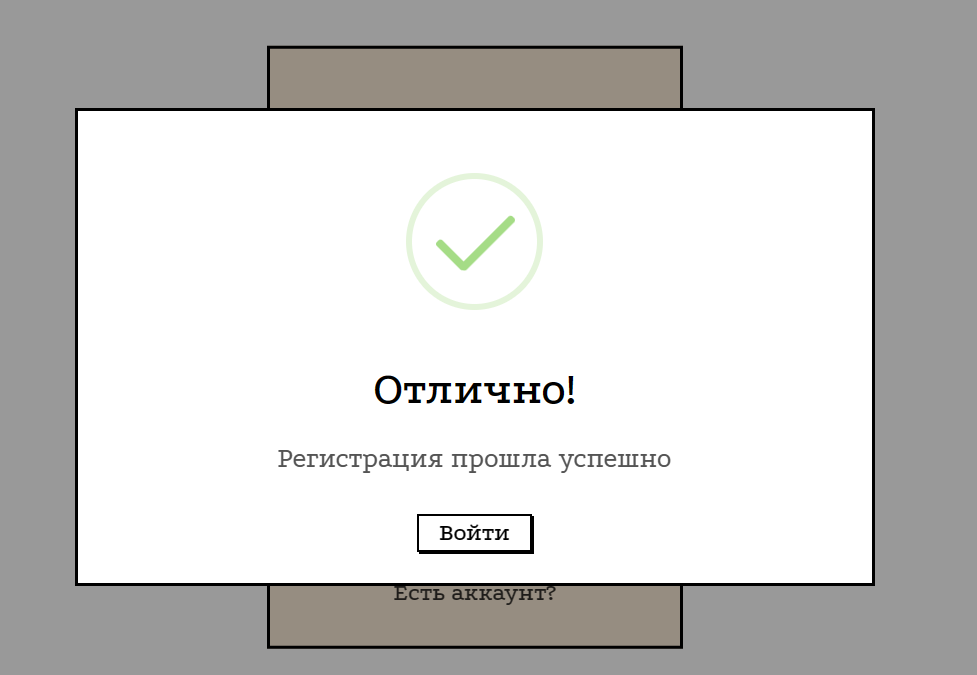


Рисунок 4.3 – Сообщение о успешной регистрации

Далее проверим страницу «Проекты», попробуем изменить проект не валидными данными, к примеру, добавить несуществующую команду и попробовать сохранить, пример таких данных представлен на рисунке 4.4. Будет выведена ошибка о некоренных данных, представленная на рисунке 4.5.

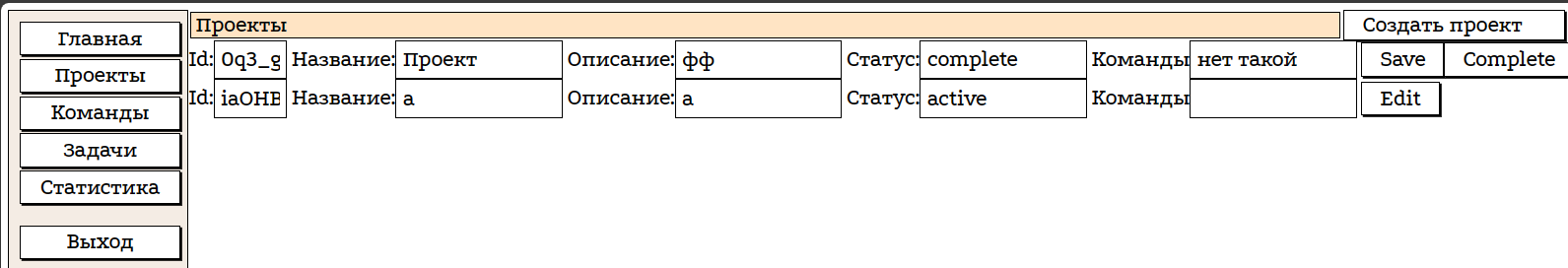


Рисунок 4.4 – Неверные данные на странице с проектами

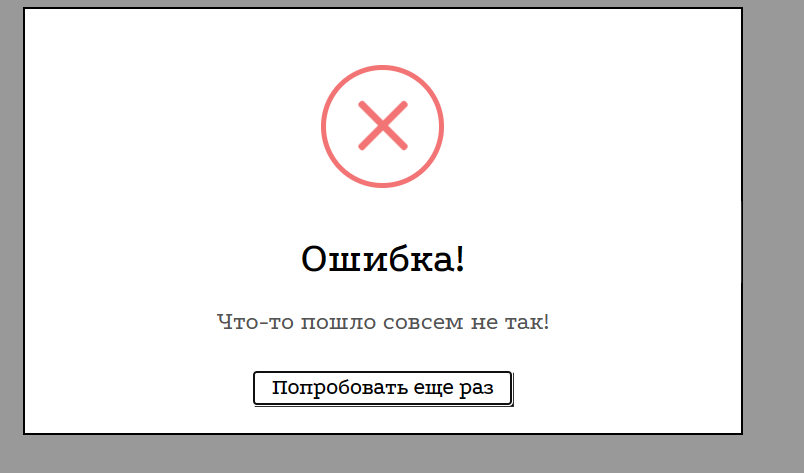


Рисунок 4.5 – Сообщение об ошибке

При вводе корректных данных, пользователь получит сообщение об успешном редактировании проекта, такое сообщение представлено на рисунке 4.6. Так работает валидация всех редактируемых полей проекта.

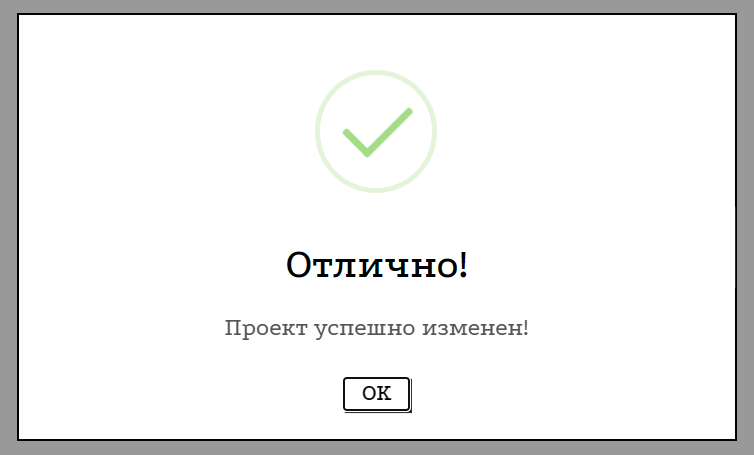


Рисунок 4.6 – Сообщение о успешном изменении

На главной странице при попытке изменении времени окончания события на время раньше, чем начало события, как представлено на рисунке 4.7. Будет получена сообщение с соответствующей ошибкой, как и изображено на рисунке 4.8.

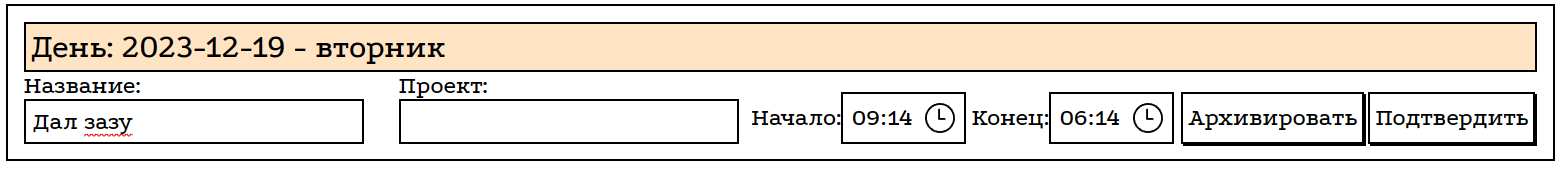


Рисунок 4.7 – Некорректные временные рамки события

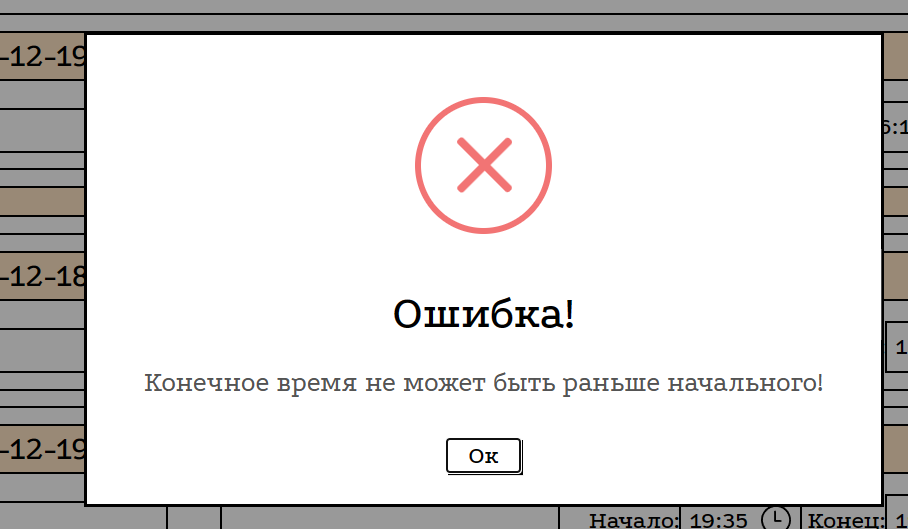


Рисунок 4.8 – Ошибка из-за некорректных временных рамок

В приложении пользователи могут брать задачи из списка задач, и завершать их

* 1. Вывод

После проведения ручного тестирования можно утверждать, что функционал, подвергнутый проверке, исправно функционирует, и в процессе выполнения тестов выявлено никаких ошибок. Все функции приложения выполняются без нарушений, и в случае возникновения ошибки пользователь получает соответствующее уведомление.

1. Руководство пользователя

В данном руководстве описано как пользователю работать с веб-приложением, которое имеет первоначальный вид, представленный на рисунке 5.1. Прилоежение имеет интуитивно понятный интерфейс и основной функционал доступен на главной странице. Пользователь может ввести над чем он работает и запустить таймер. После остановки таймера появится это события в истории, пользователь может редактировать информации о событии и архивировать его. Может выбрать проект над которым работал.

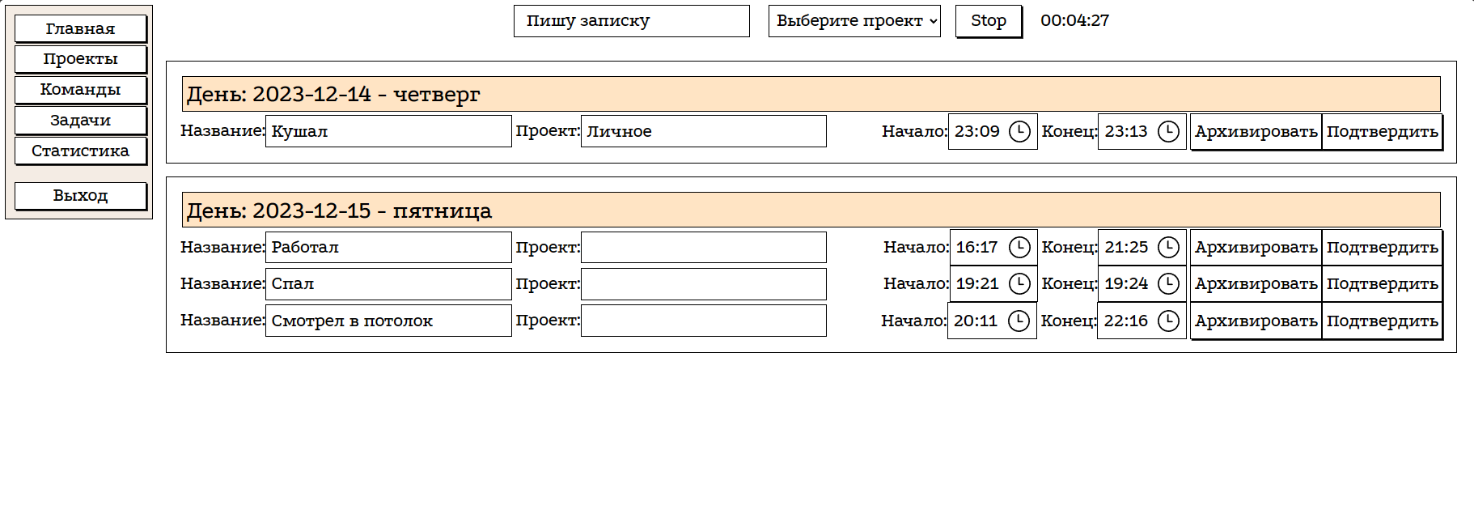


Рисунок 5.1 – Главная страница Приложения

Пользователь может выбрать задачу из списка задач по проектам, доабвив ее к себе, другие участники в таком случае видят что задача вщятва и не могут ее взять.

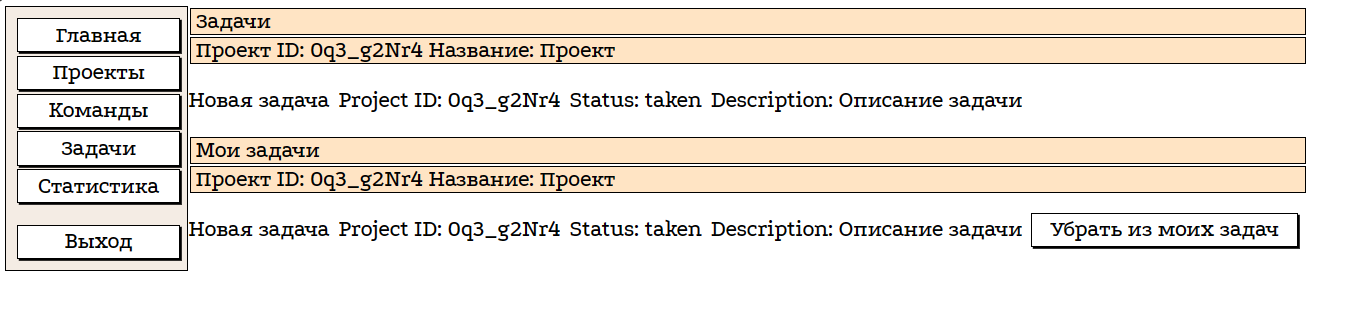


Рисунок 5.2 – Страница с задачами

Так же на странице с статистикой пользователь может посмотреть статистику потраченного времени, и сохранить результат в PDF . Страницу с статистикой можно наблюдать на рисунке 5.3.

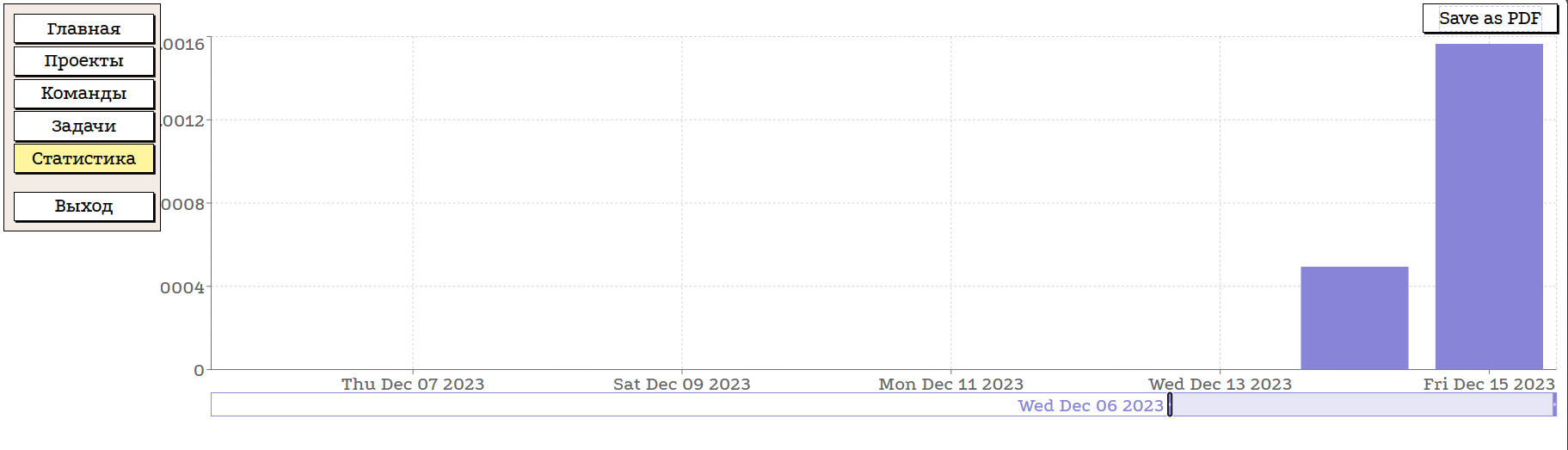


Рисунок 5.3 – Страница с статистикой

* 1. Руководство менеджера

Каждый пользователь при создании проекта становится менеджером. Менеджер может редактировать свой проект, добавлять доступные ему команды, создавать проекты и задачи. Страница проектов представлена на рисунке 5.4.

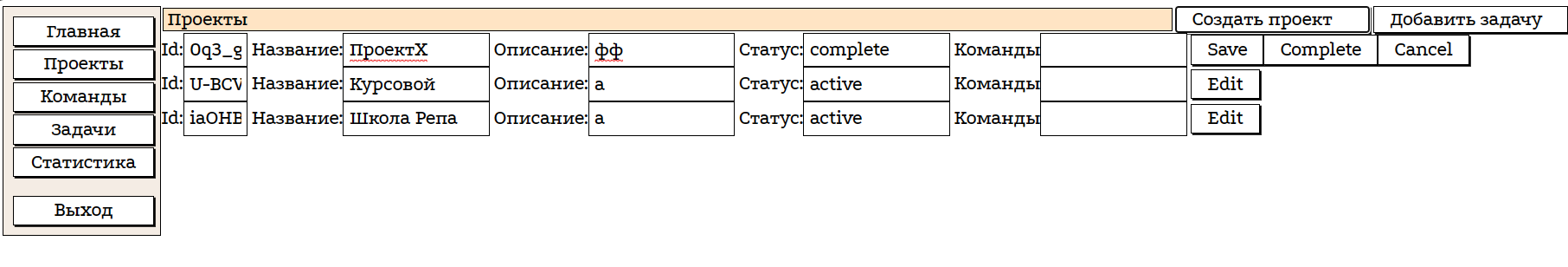


Рисунок 5.4 – Страница проектов

Для создания проекта, форму создания которой можно наблюдать на рисунке 5.5, достаточно только названия и описания, после чего менеджер может добавить команды, редактируя проект.

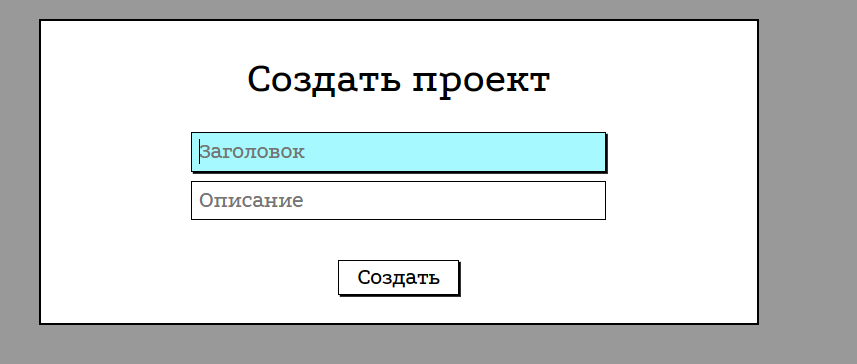


Рисунок 5.5 – Форма создания проекта

Страница с пользователями, представленная на рисунке 5.6, по структуре такая же, как и на странице с проектами.

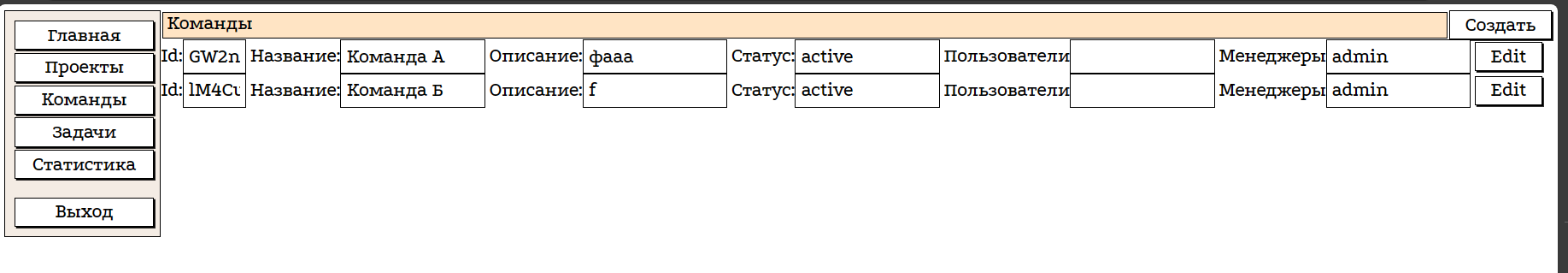


Рисунок 5.6 – Страница с командами

* 1. Руководство администратора

Для доступа к странице администрирования, необходима отдельная авторизация. Страница авторизация изображена на рисунке 5.7.

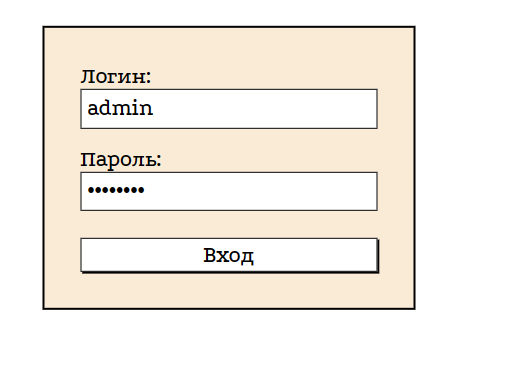


Рисунок 5.7 – Страница авторизации администратора

После успешной авторизации в аккаунт с правами администратор, попадаем на страницу администрирования, представленная на рисунке 5.8. На этой странице есть возможность редактирования пользователей, блокировки их путем редактирования статуса.

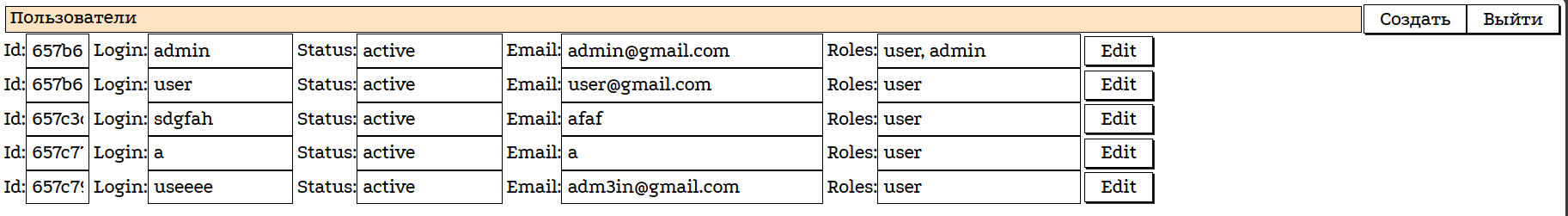


Рисунок 5.8 – Страница администрирования

. Также на этой странице есть возможность вызова формы для создания пользователей с выдачей прав доступа.

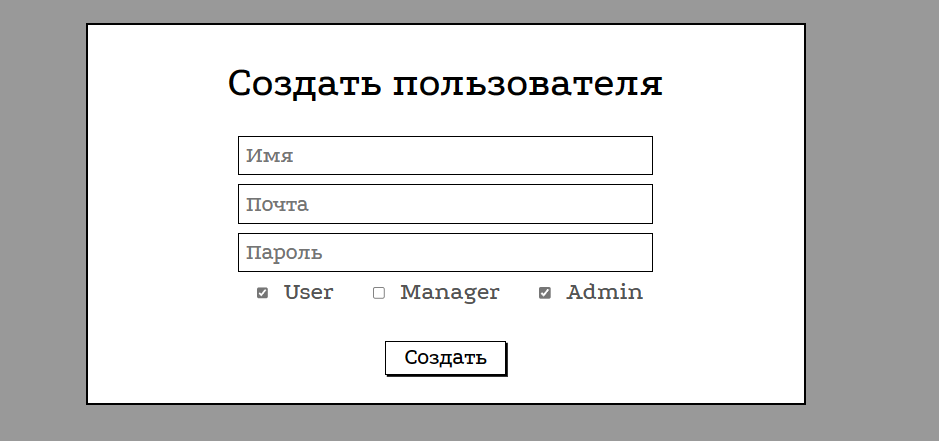


Рисунок 5.9 – Форма создания пользователя

* 1. Вывод

По итогам руководства пользователя можно отметить, что приложение имеет простой интерфейс и позволяет моментально приступить к работе пользователю. Также имеет широкий набор организации работы путем создания проектов, команд и задач

1. Заключение

В процессе выполнения курсового проекта были изучены основные аналоги разрабатываемого программного средства, выявлены недостатки в рассматриваемых системах. Сформулированы функциональные требования, охватывающие необходимый функционал.

Для реализации веб-приложения был выбран стек технологий, включающий Node.js для серверной части, React для клиентской части, и MongoDB для хранения данных. Node.js обеспечивает выполнение серверного кода, React – разработку динамичного пользовательского интерфейса, а MongoDB служит базой данных для приложения.

Был разработан веб-сервис, используя архитектуру REST для обработки клиентских запросов и взаимодействия с базой данных MongoDB. Схемы документов были созданы для эффективного хранения данных в базе.

Кроме того, в ходе проекта были освоены основы Docker. Использование Docker позволило упростить развертывание приложения и обеспечить его независимость от окружения, создав и управляя контейнерами для компонентов приложения.

1. Список используемых источников
2. Toggl [Электронный ресурс] – Режим доступа: https://toggl.com/ – Дата доступа: 18.09.2023
3. Harvestapp [Электронный ресурс] – Режим доступа: https://ikla.harvestapp.com/time – Дата доступа: 18.09.2023
4. Hubstaff [Электронный ресурс] – Режим доступа: : https://hubstaff.com/ – Дата доступа: 18.09.2023
5. JavaScript | MDN | MDN Web Docs [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/JavaScript//. – Дата доступа: 25.09.2023.
6. Node.js v21.4.0 documentation [Электронный ресурс]. – Режим доступа: https://nodejs.org/api/all.html// – Дата доступа: 25.09.2023.
7. Документация для работы с «React» [Электронный ресурс]. – Режим доступа: https://ru.react.js.org/docs/. – Дата доступа: 25.09.2023.
8. MongoDB documentation [Электронный ресурс]. – Режим доступа: https://www.mongodb.com/docs//– Дата доступа: 28.11.2023.
9. Руководство по Docker [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/ruvds/blog/450312//. – Дата доступа: 26.11.2023.
10. Docker Hub [Электронный ресурс]. – Режим доступа: https://hub.docker.com//. – Дата доступа: 04.12.2023.

ПРИЛОЖЕНИЕ А

Листинг app.js

import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom';

import MainPage from './components/Pages/MainPage';

import RegisterPage from './components/Pages/RegisterPage';

import LoginPage from './components/Pages/LoginPag';

import AdminPage from './components/Pages/AdminLoginPage';

import AdminPanel from './components/Pages/AdminPanel';

import ProjectsPage from './components/Pages/ProjectsPage';

import TeamsPage from './components/Pages/TeamsPage';

import TaskPage from './components/Pages/TaskPage';

import StatPage from './components/Pages/StatPage';

import React, { useEffect } from 'react';

import { useState } from 'react';

function App() {

  return (

    <BrowserRouter>

      <Routes>

        <Route path="/" element={<RouteWrapper  allowedRoles={['user']}><MainPage /></RouteWrapper>} />

        <Route path="/signup" element={<RegisterPage />} />

        <Route path="/signin" element={<LoginPage />}/>

        <Route path="/admin" element={<AdminPage  />}/>

        <Route path="/adminpanel" element={<RouteWrapper  allowedRoles={['admin']}><AdminPanel /></RouteWrapper>} />

        <Route path="/projects" element={<RouteWrapper  allowedRoles={['user']}><ProjectsPage /></RouteWrapper>} />

        <Route path="/teams" element={<RouteWrapper  allowedRoles={['user']}><TeamsPage /></RouteWrapper>} />

        <Route path="/task" element={<RouteWrapper  allowedRoles={['user']}><TaskPage /></RouteWrapper>} />

        <Route path="/stat" element={<RouteWrapper  allowedRoles={['user']}><StatPage /></RouteWrapper>} />

      </Routes>

    </BrowserRouter>

  );

}

function RouteWrapper({ children, allowedRoles }) {

  const [isLoading, setIsLoading] = useState(true);

  const [isAuthenticated, setIsAuthenticated] = useState(false);

  useEffect(() => {

    checkAuthentication(allowedRoles).then((Authenticated) => {

      setIsAuthenticated(Authenticated);

      setIsLoading(false);

    });

  }, []);

  if (isLoading) {

    return <div>Loading...</div>;

  }

  console.log(isAuthenticated);

  if (!isAuthenticated) {

    return <Navigate to="/signin" />;

  }

  return children;

}

async function checkAuthentication(rols) {

  const token = localStorage.getItem('authToken');

  if (!token) return false;

  try {

    const response = await fetch('http://localhost:3001/api/check-token', {

      headers: {

        'Authorization': `Bearer ${token}`,

      },

    });

    if (response.status !== 200) {

      console.log('Token verification failed');

      return false;

    }

    const data = await response.json();

    if (data.roles && data.roles.some(val => rols.includes(val))) {

      return data.isValid;

    } else {

      alert('Недостаточно прав');

      return false;

    }

  } catch (err) {

    return false;

  }

}

export default App;

ПРИЛОЖЕНИЕ Б

Листинг Dockerfile для сервера

# Используем базовый образ Node.js

FROM node:14

# Устанавливаем рабочую директорию внутри контейнера

WORKDIR /usr/src/server

# Копируем зависимости и файлы проекта в контейнер

COPY package\*.json ./

# Устанавливаем зависимости

RUN npm install

# Копируем остальные файлы сервера

COPY . .

# Определяем порт, на котором будет работать сервер

EXPOSE 3001

# Определяем команду для запуска сервера

CMD ["npm", "start"]

Листинг Dockerfile для базы данных Mongodb

# Используем официальный образ MongoDB

FROM mongo:latest

# Опционально: копируем исходные данные в контейнер

# COPY ./data /data/db

# Определяем порт, который будет открыт для внешнего доступа

EXPOSE 27017

# Определяем команду для запуска MongoDB

CMD ["mongod"]