Содержание

Введение

На данный момент Интернет является основой для функционирования многих информационных систем. Он предоставляет возможность быстрого и удобного доступа к различным видам информации, обмена данными, коммуникации и сотрудничества. Благодаря интернету информационные системы становятся более гибкими и масштабируемыми, позволяя пользователям получать доступ к данным из любой точки мира и в любое время. Интернет также способствует развитию облачных информационных систем, которые предоставляют возможность хранения и обработки данных на удаленных серверах, что увеличивает их доступность и безопасность. Таким образом, интернет играет важную роль в развитии и совершенствовании информационных систем, делая их более эффективными и универсальными.

В сети Интернет всё сильнее распространяется веб-приложения. Спектр их функций подстраивается под любую предметную область, в которой проводятся связь между клиентом и основным сервером.

На данный момент, где скорость и эффективность выполнения задач становятся ключевыми факторами успеха как для индивидуумов, так и для организаций, необходимость в инструментах управления задачами и проектами приобретает особую значимость. В условиях постоянного увеличения объема информации и многообразия задач, с которыми сталкиваются пользователи, возникает потребность в эффективных решениях, которые помогут оптимизировать рабочие процессы, повысить продуктивность и улучшить организацию времени.

Согласно исследованиям, объем информации, с которым работают сотрудники, продолжает расти. Это приводит к тому, что управление задачами становится все более сложным и трудоемким процессом. Без эффективного инструмента для отслеживания и управления задачами пользователи рискуют потерять важные детали, что может негативно сказаться на результатах работы.

С увеличением числа удаленных сотрудников и гибридных форматов работы, пользователи требуют доступных и удобных решений, которые можно использовать в любом месте и в любое время.

Актуальность веб-приложения CheckTask основывается на предоставление возможноси управлять задачами через интернет, что делает его доступным для пользователей на различных устройствах от компьютеров до мобильных телефонов. В условиях командной работы важно не только управлять личными задачами, но и обеспечивать эффективное взаимодействие между членами команды. CheckTask будет включать функции для совместного использования задач, комментариев и статусов выполнения, что позволит улучшить коммуникацию внутри команды и повысить уровень ответственности каждого участника.

Веб-приложение это клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется преимущественно на сервере, обмен информацией происходит по сети

Работа веб-приложения основана на взаимодействии пользователя с сервером. Пользователь взаимодействует с веб-приложением, используя веб-браузер, который отправляет запросы на сервер. Сервер обрабатывает эти запросы и отправляет обратно результаты в виде веб-страницы или данных, которые отображаются пользователю на экране.

Разработка веб-приложений – это процесс создания программного обеспечения, которое может работать на различных устройствах через интернет. Используя различные языки программирования, инструменты и фреймворки, разработчики создают функциональные и удобные в использовании приложения.

Существует несколько видов веб-приложений, которые могут быть разработаны. Это может быть стандартное веб-приложение, состоящее из фронтенда и бэкенда, где фронтенд обеспечивает интерфейс пользователя, а бэкенд отвечает за обработку запросов и хранение данных. Также существуют одностраничные веб-приложения, где весь контент загружается один раз, и пользователь взаимодействует с ним без перезагрузки страниц

Разработка веб-приложений включает в себя несколько этапов. Сначала проектируются интерфейс и функциональность приложения. Затем создаются макеты и прототипы, чтобы визуализировать и протестировать идеи. После этого разработчики начинают писать код, используя выбранные технологии и инструменты. После завершения кодирования приложение тестируется и оптимизируется для достижения высокой производительности. Для разработки веб-приложений используются различные технологии, языки программирования и фреймворки. Одним из самых популярных языков программирования для веб-разработки является JavaScript, который позволяет создавать интерактивные элементы на веб-странице и взаимодействовать с сервером. Также часто используются HTML и CSS для создания структуры и оформления веб-приложения.

Также веб-приложения может быть развернуты на сервере или в облаке, чтобы обеспечить доступность и масштабируемость. Разработчики также могут использовать базы данных для хранения и обработки данных приложения.

В нынешней ситуации рассматривается ситуации разработка веб-приложение с функционалом Task менеджера для планирования бизнеса. Разработка веб-приложения для планирования бизнеса. – это процесс создания комплексной программно-аппаратной системы, которая предназначена для автоматизации различных функций и процессов, связанных с планированием задач, постановка целей каких-либо проектов и их контроля в течении выполнения. Веб-приложение является одним из основных инструментов управления и организации образовательного процесса. Разработка веб-приложения для бизнеса включает в себя анализ и проектирование информационной структуры, разработку программного обеспечения, внедрение системы в использование, а также обеспечение ее дальнейшей поддержки и сопровождения.

Главной целью непосредственно это разработка веб-приложения CheckTask для планирования бизнеса. После повышение эффективности планирования и осуществление над ним контроль. Благодаря приложению, бизнес-компания может упростить и ускорить такие процессы, как планирование задач, составление

расписания мероприятий, учет успеваемости и выполнения задач, ведение финансового учета и многое другое. Веб-приложение также позволяет повысить доступность информации для участников бизнес-проекта – сотрудники, разработчки и контроллеры.

Важно учитывать потребности и задачи различных групп пользователей, а также обеспечивать надежность и безопасность системы. При разработке можно использовать различные технологии и инструменты, такие как базы данных, иные веб-приложения, мобильные приложения и т.д.

Веб-приложение для бизнеса – это важный инструмент, позволяющий современным коммерческим организациям оптимизировать проектировочный процесс, повысить качество задач и их детальность.

В соответствие с поставленной целью, задачами являются:

– исследование потребностей и требований пользователей и бизнес-процессов организации;

– изучение предметной области;

– проектирование архитектуры информационной веб-приложения, включая выбор технологий, платформы и инструментов разработки;

– разработка технического задания;

– разработка программного обеспечения, включая создание базы данных, интерфейсов пользователя, бизнес-логики и т.д.

– тестирование и отладка информационной системы для обеспечения ее надежности, безопасности и производительности;

– внедрение информационной системы в рабочую среду организации, обучение пользователей и поддержка в процессе эксплуатации;

– оценка эффективности информационной системы и ее соответствия поставленным целям и требованиям;

– проверка работоспособности и исправление ошибок и недочётов.

Объектом разработки является IT-компания Day Digital (ИП ЮДИН А.В.), где будет проходить практика. Основной филиал данной компания находится в Новосибирске, соответственно разработка будет вестись в дистанционном формате. Компания может быть рассмотрена как объект для исследования в контексте информационной системы, например, для создания электронной системы процессом планирования. Исследование потребностей и требований пользователей (сотрудников, разработчиков, администраторов) позволит определить функциональные возможности, которые должна предоставить приложение. Разработка программного обеспечения будет включать создание базы данных для хранения информации о заметках, личных данных пользователя, данные об управления заметками и т.д. Также будет необходимо разработать интерфейсы для пользователей и администраторам.

Разработка программного обеспечения будет включать создание базы данных для хранения информации о пользователях. Также будет необходимо разработать интерфейсы для заметок пользователя.

Тестирование и отладка веб-приложения будет направлено на обеспечение ее

надёжности, безопасности и производительности, чтобы гарантировать эффективное использование системы в рамках учебного процесса.

Внедрение информационной системы в бизнес среду потребует обучения пользователей и поддержки в процессе эксплуатации. Это включает в себя обучение пользователям работе с системой, а также техническую поддержку и обновление системы по мере необходимости.

Оценка эффективности информационной системы и ее соответствия поставленным целям и требованиям позволит определить, насколько система удовлетворяет потребности компании в автоматизации процесса планирования и оптимизации работы сотрудников, бухгалтерии и администрации.

Предметом является веб-приложение по заданной предметной области. Исследование приложения в бизнес-компании может включать в себя анализ потребностей сотрудников, разработчиков проектов и администраторов, их взаимодействие с текущими информационными технологиями, а также выявление возможностей для оптимизации процессов планирования и ведения над ним контроля.

Таким образом, по окончании проекта будет реализовано веб-приложение по проведении планирования целей и задач с возможностью их контроля через назначение управляющего заметки. Тем самым, планирование заметки будет проводиться в группе под отчётностью и с предоставлениям итогов и результатов сразу же после их внедрения в веб-приложение.

1 Руководство системному программисту

Веб-приложение CheckTask предназначено для организации, хранения и управления заметками и бизнес-идеями пользователей. Оно предоставляет удобный интерфейс для структурированного учета данных, планирования задач и отслеживания прогресса.

Для системы в необходимом порядке нужно разработать Руководство системному программисту. Оно представляет комплексный документ или набор рекомендаций, предназначенных для поддержки специалистов, занимающихся разработкой и оптимизацией системного программного обеспечения. Оно охватывает различные аспекты работы системного программиста, включая архитектуру операционных систем, взаимодействие с аппаратным обеспечением, управление памятью, многопоточность и синхронизацию процессов. В руководстве также могут быть представлены лучшие практики программирования, отладка и тестирование системных приложений, а также рекомендации по использованию инструментов разработки и сред.[5] Кроме того, важно учитывать аспекты безопасности и производительности, что позволяет системным программистам создавать надежные и эффективные решения, соответствующие современным требованиям и стандартам в области информационных технологий.

Основные функции приложения для руководства:

- создание, редактирование и удаление заметок;
- категоризация заметок (личные, учеба, проекты);
- установка приоритетов и сроков выполнения задач;
- установка времени выполнения работы (использование телеграмм-бота для напоминания);
- поиск заметок по ключевым словам, тегам или категориям;
- создание заметок в группе.

Для успешной работы веб-приложения CheckTask необходимо обеспечить все вышеперечисленные функции и определенный требования к технике и ПО(программное обеспечение):

Требования к техническим средствам:

- процессор: минимум двухъядерный процессор с частотой 2 ГГц.
- оперативная память: не менее 4 ГБ.
- свободное место на жестком диске: минимум 500 МБ (для размещения файлов приложения и базы данных).
- сетевое подключение: стабильное подключение к интернету для доступа к серверу.

Требования к программному обеспечению:

- операционная система: Windows 10/11, Linux (Ubuntu 20.04+), macOS;
- веб-сервер: Nginx 1.18+;
- база данных: PostgreSQL 13+;
- интерпретатор JavaScript: Node.js 16+ (для фронтенд-компонента);
- Установленные библиотеки psycopg2 (для PostgreSQL);

– установленный менеджер пакетов npm для управления зависимостями фронтенда.

Для просмотра и анализа веб-приложения необходимо изучить его основную архитектуру и сведения приложения. При просмотре кода в начале сервера подробно описывается все библиотеки и системы, которые были установлены для полноценной работы сайта (Рисунок 1).

```
const express = require('express');
const session = require('express-session');  21.8k (gzipped: 7.5k)
const bodyParser = require('body-parser');  487.3k (gzipped: 212.1k)
const { Pool } = require('pg');  76.5k (gzipped: 22.6k)
const bcrypt = require('bcrypt');
const path = require('path');  211 (gzipped: 166)
const { Telegraf } = require('telegraf');  87.5k (gzipped: 22k)
const cron = require('node-cron');  11.2k (gzipped: 4.3k)
const { user } = require('pg/lib/defaults');  11.1k (gzipped: 4.6k)
const sharp = require('sharp');  113.9k (gzipped: 33.2k)
const fs = require("fs")
const multer = require("multer")  241.4k (gzipped: 47.4k)
const cors = require("cors")  4.5k (gzipped: 1.9k)

const app = express();
const port = 3000;
const bot = new Telegraf('8045210429:AAEIGN_hISxY3p2mTprZ_IDRbxSdN376p7k');
```

Рисунок 1 - Основные библиотеки и системы сайта

Каждая константа и библиотека, которые указаны в коде имеет свои значения и функции.

Express - это основной фреймворк для создания веб-приложений на Node.js. Функционал: Предоставляет инструменты для обработки HTTP-запросов, маршрутизации, middleware и других функций, необходимых для построения сервера. В данном случае `express` используется для создания экземпляра

Session - модуль для управления сессиями в приложении Express. Функционал: Позволяет хранить данные о пользователях между запросами (например, информацию о входе в систему). Обычно используется вместе с middleware для сохранения данных сессии.

bodyParser - Middleware для парсинга тела HTTP-запросов. Преобразует данные из тела запроса в удобный формат (например, JSON или URL-encoded). Необходимо для работы с POST-запросами, где передаются данные в теле запроса.

Pool (из модуля pg) - Библиотека для работы с PostgreSQL. Предоставляет API для подключения к базе данных PostgreSQL, выполнения SQL-запросов и управления соединениями. Используется для взаимодействия с PostgreSQL-базой данных.

Bcrypt - библиотека для безопасного хеширования паролей. Предоставляет методы для хеширования и проверки паролей с использованием алгоритма bcrypt. Обычно используется для защиты паролей пользователей при регистрации и аутентификации.

Path - Встроенная Node.js библиотека для работы с путями файлов и директорий. Предоставляет методы для манипулирования путями файловой системы независимо от операционной системы.Часто используется для работы с путями к файлам, особенно при работе с multer (для загрузки файлов).

Telegraf - Фреймворк для создания ботов Telegram. Упрощает работу с API Telegram, предоставляя удобные методы для обработки сообщений, команд и других событий. В данном случае используется для создания Telegram-бота (bot = new Telegraf(...)).

Cron - Библиотека для планирования задач на основе cron-синтаксиса. Позволяет запускать определенные функции в заданное время (например, ежедневно, каждый час и т. д.). Может использоваться для автоматического выполнения задач, таких как очистка базы данных, отправка рассылок и т. п.

Sharp Библиотека для обработки изображений. Предоставляет высокопроизводительные методы для ресайза, изменения формата, обрезки и других операций над изображениями. Может использоваться для обработки загружаемых изображений (например, уменьшения размера или изменения формата).

Fs - Встроенная Node.js библиотека для работы с файловой системой. Предоставляет методы для чтения, записи, удаления и управления файлами и директориями. Может использоваться для работы с локальными файлами, например, при сохранении загруженных файлов.

Cors - Middleware для обработки CORS (Cross-Origin Resource Sharing). Разрешает клиентам делать AJAX-запросы к другому домену, чем домен сервера. Нужно для того, чтобы frontend-приложение могло делать запросы к backend-серверу без ошибок CORS.

Последние строчки представляют экземпляр приложения Express, созданный с помощью `express()` для реализации маршрутов на сервере и порта для запуска сервера.

Код представляет собой начальный этап настройки веб-приложения на Node.js с использованием Express. Приложение также включает:

- поддержку сессий (`express-session`);
- работу с PostgreSQL (`pg`);
- хеширование паролей (`bcrypt`);
- создание Telegram-бота (`Telegraf`);
- обработку изображений (`sharp`);
- планирование задач (`node-cron`);
- обработку CORS-запросов (`cors`).

Это типичная структура для приложения, которое может работать с базой данных, обрабатывать файлы, взаимодействовать с Telegram и обеспечивать безопасность.

При работе с сервером, а именно с главной папки со всеми библиотеками необходимо в терминале установить все вышеперечисленные библиотеки и средства. Их устанавливают с помощью библиотеки NPM. Основной пример установки приведён на рисунке 2. Также для отслеживания скаченных файлов необходимо ввести команду npm init -y. Она устанавливает файл package.json, в котором прописывается все скаченные библиотеки с их версиями.

```
админ@LAPTOP-Q8R170GL MINGW64 ~/OneDrive/Рабочий стол/example (main)
$ npm install express express-session
```

Рисунок 2 - Использование пакета npm для скачивания

Таким образом можно прописать все библиотеки сайта без знаков препинания. Все библиотеки установятся в папке node_modules после команды внутри корневой папки сайта

Для подключения СУБД нужно проверить скрипт с основными значениями БД (Рисунок 3)

```
// Настройка подключения к PostgreSQL
const pool = new Pool({
  user: 'postgres',
  host: 'localhost',
  database: 'checktask',
  password: 'password',
  port: 5432,
  application_name: 'CheckTask',
});

pool.connect()
  .then(() => {
    console.log('Подключено к PostgreSQL')
  }).catch((err) => {
    console.error('Ошибка подключения к PostgreSQL:', err)
  });
```

Рисунок 3 - Подключение внешней СУБД

Через константу прописываются значения СУБД - название пользователя, название сервера, название БД, пароль от бд, порт в котором она находится и название внутри сайта. Ниже представлена передача подключения через консоль чтобы программист мог увидеть сразу подключилась ли БД или нет.

При запуске сервера веб-приложения неоходимо ввести команду node index (Рисунок 4). Только после этой команды сервер заработает везде где он подключен, в том числе и на веб-серверах.

```
админ@LAPTOP-Q8R170GL MINGW64 ~/OneDrive/Рабочий стол/example (main)
$ node index
(node:6100) [DEP0040] DeprecationWarning: The `punycode` module is deprecat
(Use `node --trace-deprecation ...` to show where the warning was created)
Сервер запущен - http://localhost:3000
Подключено к PostgreSQL
```

Рисунок 4 - Запуск сервера в корневом каталоге

После запуска сразу же в терминале выводится ссылка сервера, которая задается в самом конце скрипта (Рисунок 5). Ссылка прописана через HTTP-запрос. Также сразу осуществилась проверка подключения БД, следовательно после запуска БД сразу же начала свою работу.

```
module app
const app: Express
app.listen(port, () => {
  console.log(`Сервер запущен - http://localhost:${port}`);
  bot.launch();
});
```

Рисунок 5 - Скрипт запуска сервера с телеграмм ботом

Совместно с сервером сразу же запускается работа телеграмм бота через который будет публиковаться напоминания сроков заметки.

Это базовые настройки сервера для системного программиста. Для дальнейшего сопровождения рекомендуется регулярно обновлять зависимости, выполнять резервное копирование базы данных и проводить тестирование на производительность.

2 Руководство пользователю

Руководство пользователя - это документ, предназначенный для конечных пользователей программного обеспечения, оборудования или системы, который предоставляет подробные инструкции по их эффективному использованию. Оно включает в себя описание функциональности продукта, пошаговые процедуры выполнения различных задач, советы по устранению неполадок и ответы на часто задаваемые вопросы.[8] Целью руководства является упрощение процесса освоения и использования продукта, повышение удовлетворенности пользователей и снижение количества запросов в службу поддержки. Хорошо структурированное руководство пользователя является важным инструментом, позволяющим пользователям максимально эффективно использовать все функции и возможности продукта.

Руководство пользователя служит нескольким важным целям:

- обучение пользователей;
- упрощение освоения;
- поддержка функциональности;
- устранение неполадок;
- снижение нагрузки на службу поддержки;
- повышение удовлетворенности пользователей.

Цели веб-приложения:

- создание учетной записи для сохранения планов, идей пользователя;
- создание заметки для планирования и построение плана проекта или бизнес-идеи;
- возможность визуального управления;
- возможность планирования проекта в группе (под контролем другого сотрудника);

Основные функции веб-приложения:

- аутентификация;
- создание профиля пользователя;
- создание заметки;
- управление заметкой(поиск, этапность, удаление, редактирование, установка времени);
- создание заметки в группе.

При переходе в веб-приложение все функции пользователь может выполнить все эти функции. Сама система также составлена под этот функционал, в определенном алгоритме:

- переход на главную страницу: пользователь нажимает на необходимую кнопку(вход, регистрация)
- пользователь проходит авторизацию или регистрацию и переходит в свой профиль
- с профиля через главное меню выбирает неоходимую страницу с каким-либо функционалом заметки;

– при переходе на страницу «Заметки» пользователю выводится в ней модальное окно для создания заметки;

– через страницы пользователь совершает функционал управления заметки;

– перед выходом пользователь переходит в профиль и от страницы профиля по кнопке «Выход» совершает выход из учетной записи, где переходит на главную страницу с выбором аутентификации.

Через данный алгоритм пользователь может совершать все встроенные функции с заметками под своей учётной записи. Данный алгоритм является общим. Для каждой страницы также есть свой алгоритм действий пользователя для совершения функции в правильном виде. Основываясь на данной логике, страница «Заметки» содержит список заметок пользователя и модальное окно для их создания. Она является основной для веб-приложения, остальные страницы вспомогательные и служат для управления заметкой. Алгоритм действий для создания заметки состоит из нескольких шагов:

– заполнения поля Заголовка;
– выбор стадии;
– выбор темы заметки;
– заполнение поля описания (основная информация заметки);
– сохранение заметки в списке на странице по кнопке «Создать заметку».

Данный алгоритм поддерживает форма на странице с ведением всех вышеперечисленных пунктов. Форма представлена в Контрольном примере. Таким образом создается личная заметка пользователя. Сама заметка представляется в виде карточки. В самой нижней части карточки находится всплывающее меню (Рисунок 6) состоящее из пунктов: В архив, напомнить, редактировать, удалить.

Данное меню основывается на функционале моделирования и изменения заметки, а также фильтрации под необходимые параметры.

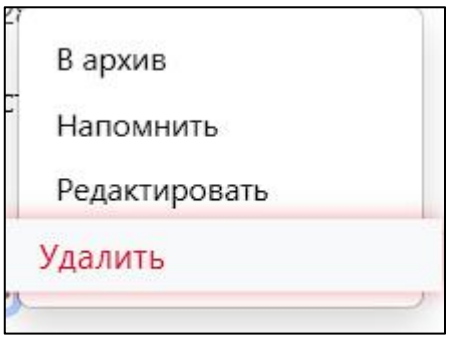

Рисунок 6 - Всплывающее меню карточки заметки

По кнопке «В архив» заметка переходит на страницу «Архив», в которой редактирование и удаление не действует. По логике, в архиве хранятся отложенные заметки или же заметки с какой-либо фиксированной информацией. На самой странице в нижней части карточки заметки вместо всплывающего меню кнопка «Вернуть из архива», после нажатие которой карточка сразу же перемещается обратно в общую страницу «Заметки», где снова появляется возможность управления данными заметки.

Кнопка «Напомнить» выводит всплывающее меню создать напоминание (Рисунок 7). В ней выбирается дата на определённый срок, и пройдя сохранения заметка перемещается на страницу «Напоминания».

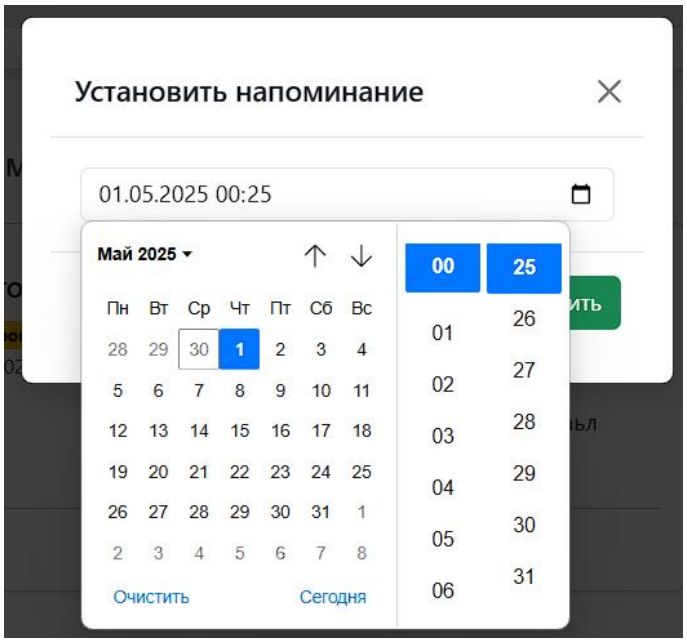

Рисунок 7 - Всплывающее окно для назначения напоминания

Страница содержит блок для привязки телеграмм бота к учетной записи. На рисунке 8 изображен блок до привязки бота, по кнопки «Получить код привязки» телеграмм бот отправит уведомительный код для создания чата с ботом, после боту необходимо отправить команду /link и только после этой команды он отправит код для привязки уже учетной записи с к заметкам. Этот код необходимо ввести в течение 10 минут иначе после он будет не действителен. Данный функционал создан чтобы обезопасить процесс привязки и пользователь мог привязать именно свой аккаунт. Далее после ведения кода нажимая кнопку «Привязать» сервер синхронизирует свою работу с работой бота и напоминания заметки благополучно приходит через бот за 24 часа до конца напоминания. В самом блоке наглядно описаны пошаговые действия для привязки с названием бота.

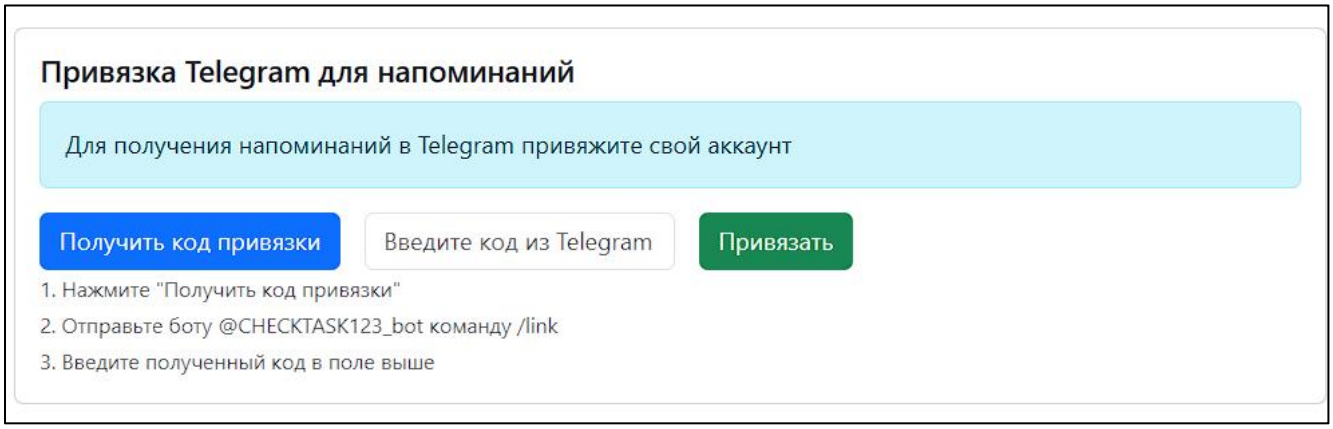

Рисунок 8 - Блок для привязки телеграм бота

Само же поле привязки изменяется и остается только надпись о успешной привязки бота (Рисунок 9). После привязки в карточке заметки также будет всплывающее меню но без пункта передачи в архив. В меню будет пункт «Удалить срок сдачи» для удаления напоминания и «Напомнить» для изменения срока напоминания.

Рисунок 9 - Успешная соединение с телеграмм ботом

Для пункта редактирования нет отдельной страницы, но имеется отдельное всплывающее окно. (Рисунок 10). Оно полностью схоже с окном создания заметки, тем самым пользователь спокойно может изменить заголовок, тему, стадию выполнения и описание. Данное окно доступно и на других страницах(Напоминания, Группы). В странице «Группы» оно доступно самому создателю заметки и другому участнику в зависимости от его роли. Доступ к редактирующему окну имеют редактор и контроллер. Пользователь под роли участник не имеет доступ так как для его роли по функционалу и назначению доступен только просмотр заметки.

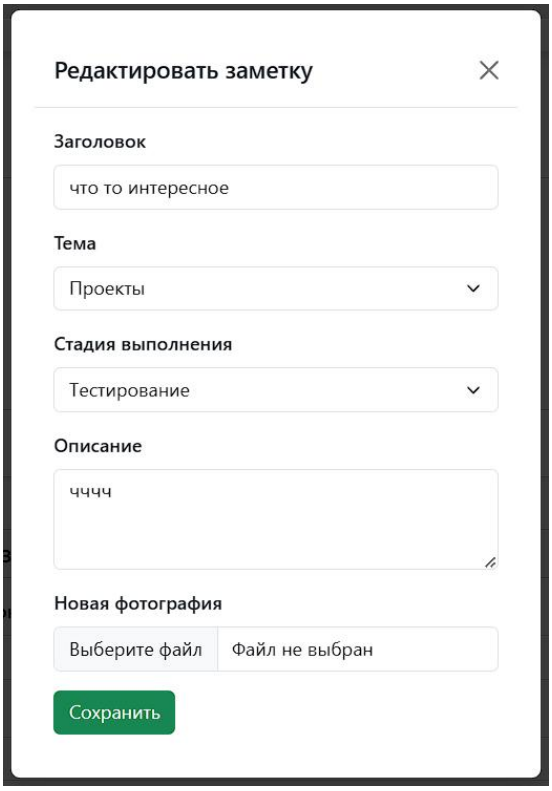

Рисунок 7 - Всплывающее окно редактирования

Страница «Группы» содержит в себе схожий алгоритм с основной страницей «Заметки». В ней также по той же логике создается заметка, но добавляется пункт выбора пользователя для группы. Он является обязательным при создании заметки в данной странице, в противном случае заметка не будет создаваться, а валидация бужет указывать на пропущенное обязательное поле. При выборе участника вводится его ФИО и почта, если данные верны выводится оповещение о создании заметки. Форма расположена под списком с карточками заметки (Рисунок 11).

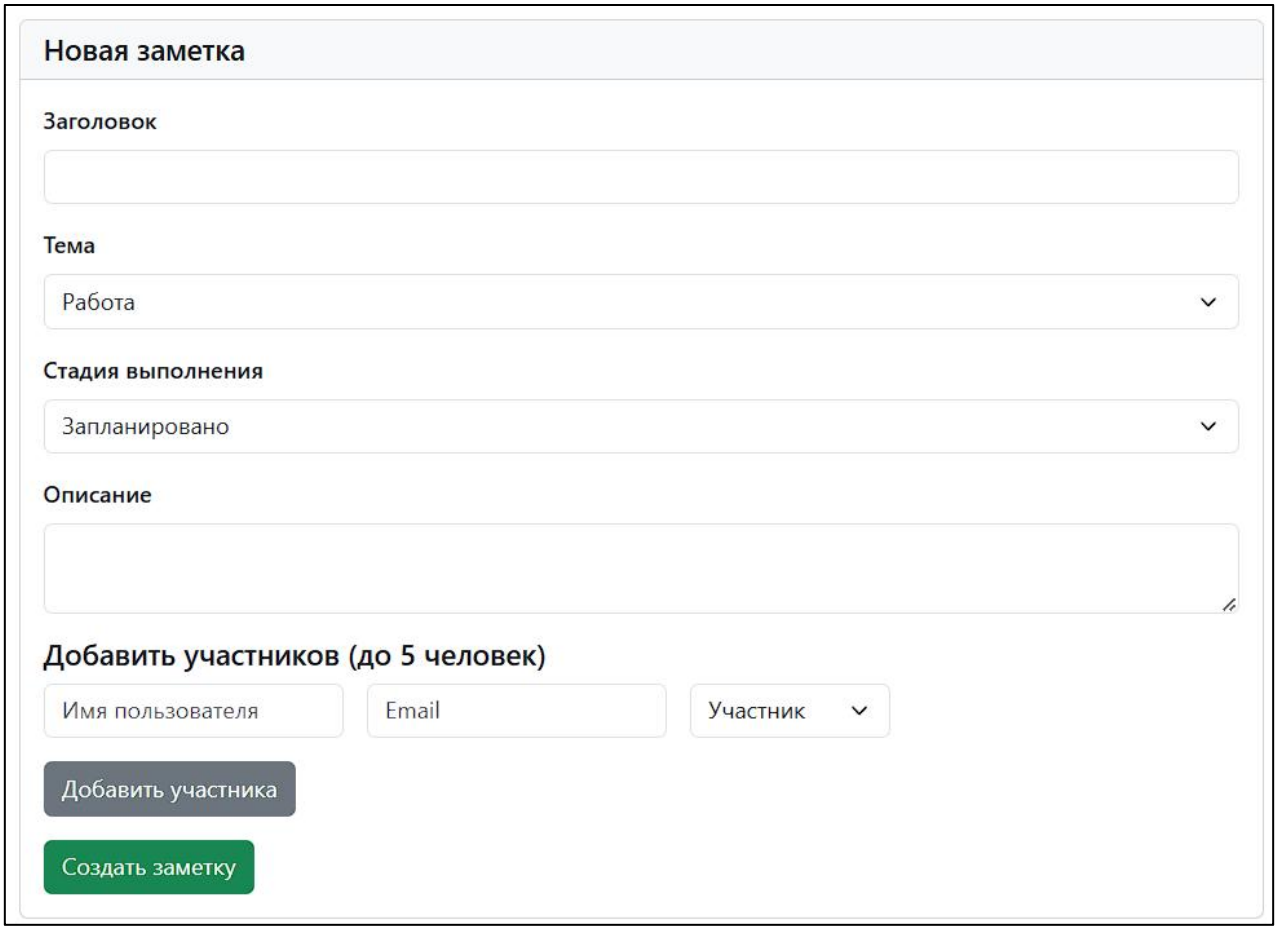

Рисунок 11 - Форма создания заметки в группе

Карточка заметки теперь имеет пункт с выводом создателя заметки(ФИО и почта) и выводом всех участников заметки. Подробный наглядный пример заметок, модальных окон и страниц представлены в Контрольном примере. Так было представлено детальное описание действий пользователя на каждой странице веб-приложения.

3 Ревьюирование программного кода

Каждая разработка несет в себе планирование и перестройку каких-либо систем для корректной работы сайта.

Ревьюирование программного кода (code review) - это процесс систематической проверки исходного кода с целью выявления ошибок, улучшения читаемости, соответствия стандартам и повышения общего качества программного обеспечения. Оно проводится другими разработчиками до того, как изменения будут добавлены в основную ветку репозитория или интегрированы в продукт. Ревьюирование способствует обнаружению багов на ранних этапах, улучшает совместную работу команды и позволяет передавать знания между разработчиками.

Существует несколько распространённых методов ревьюирования кода, каждый из которых имеет свои особенности и область применения.[20]

Формальное код-ревью (Formal Code Review) - представляет собой структурированный и строго регламентированный процесс, в котором участники заранее готовятся к встрече, изучая код, а затем обсуждают его на отдельной сессии. Часто используется в критически важных системах, где ошибка может привести к серьёзным последствиям. Каждый шаг документируется, назначаются ответственные за исправления.

Парное программирование (Pair Programming) - один из методов Agile, при котором два разработчика работают за одним компьютером: один пишет код, другой одновременно его проверяет. Это позволяет сразу выявлять потенциальные проблемы и снижает необходимость последующих формальных ревью. Плюсом является также передача знаний в режиме реального времени.

Инструменты для ревью по запросу изменений (Tool-assisted Reviews) - например, использование систем контроля версий, таких как Git, в связке с платформами вроде GitHub, GitLab или Bitbucket. Разработчик создаёт pull request (PR), после чего другие члены команды могут просматривать изменения, оставлять комментарии, предлагать правки и принимать решение о мерже. Такой подход позволяет проводить асинхронный обзор с возможностью обсуждения отдельных строк кода.

Неформальное ревью (Over-the-shoulder review) - один из самых простых и быстрых методов, когда разработчик показывает свою реализацию коллеге лично, чтобы тот дал обратную связь. Подходит для небольших изменений, но менее формализован и требует физического или виртуального присутствия.

Самостоятельное ревью (Self-review) - хотя и не заменяет внешнюю проверку, но полезно для первичного анализа собственного кода перед отправкой на ревью. Разработчик перечитывает свой код, проверяя его на соответствие стандартам, наличие явных ошибок и понятность.

Выбор конкретного метода зависит от размера команды, сложности проекта, используемых инструментов и корпоративной культуры. Эффективное ревьюирование кода помогает поддерживать высокий уровень качества продукта,

развивать профессиональные навыки разработчиков и создавать более устойчивую и поддерживаемую кодовую базу.

В нашем случае будет проведено самостоятельное ревьюирование. После его проведение при передачи разработанного веб приложения в работе с другими системными программистами ревьюирование проедется в неформальном виде.

Так, при проведение анализа кода на стороне сервера, были выявлены ошибки касаемо безопасности подключения. В коде на рисунке 12 мы видим прямое написание информации о БД для её подключение

```
// Настройка подключения к PostgreSQL
const pool = new Pool({
  user: 'postgres',
  host: 'localhost',
  database: 'checktask',
  password: 'Artur060608',
  port: 5432,
  application_name: 'CheckTask',
});
```

Рисунок 12 - Подключение БД в открытом виде

Открытый вид подключение очень небезопасен так как при передаче веб приложения есть возможность для мошенников сразу же выйти в бд и получить все данные зарегистрированых пользователей. Для решения данной проблемы используется Dotenv - это модуль npm, позволяющий загружать переменные среды в любое Node.js-приложение.

Особенности dotenv:

– упрощение загрузки. Dotenv упрощает процесс загрузки переменных среды из файла «env» в приложения Node.js, уменьшая необходимость ручной конфигурации;

– улучшение безопасности. Dotenv помогает улучшить безопасность, сохраняя конфиденциальную информацию, такую как ключи API, пароли к базам данных, в отдельном файле «.env»;

– совместимость. Пакет широко совместим с различными фреймворками и библиотеками Node.js, обеспечивая плавную интеграцию в разные типы приложений.

Как работает dotenv: при запуске приложения Node.js dotenv обращается к файлу .env. Он читает его содержимое и создаёт индивидуальные настройки (переменные среды) внутри программы. Через эти настройки переменные среды легко доступны в коде с помощью process.env.VARIABLE_NAME.

Используя данную методику на Рисунке 13 в самом сервере все пункты уже прописываются user: process.env.DB_USER, условно проводя подключение сервера

к БД не на прямую а через посредника.

```
const pool = new Pool({
  user: process.env.DB_USER,
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  password: process.env.DB_PASSWORD,
  port: process.env.DB_PORT,
});
```

Рисунок 13 - Подключение БД через файл .env

Рассматривая безопасность в самих же маршрутах стоит учесть добавление постоянной проверки в случае внедрения данных в сам сайт. Регистрация и авторизация пользователя должна всегда быть с валидацией и проверкой введеных данных. В разрабатываемом приложении при регистрации ФИО пользователя должно быть уникальным в обязательном порядке. На рисунке 14 представлена валидация имени пользователя и пароля (пароль должен содержать не больше 10 символов)

```
app.post('/register', async (req, res) => {
  const { username, email, phone, post, password } = req.body;
  const hashedPassword = await bcrypt.hash(password, 10);

  try {
    // Проверяем существование пользователя
    const userExists = await pool.query('SELECT * FROM users WHERE username = $1', [username]);

    if (userExists.rows.length > 0) {
      // Передаем сообщение об ошибке в шаблон EJS
      return res.render('register', { error: 'Пользователь с таким именем уже существует' });
    }

    // Создаем нового пользователя
    await pool.query(
      'INSERT INTO users (username, email, phone, post, password) VALUES ($1, $2, $3, $4, $5)',
      [username, email, phone, post, hashedPassword]
    );
    alert(`Добро пожаловать в CheckTask ${username}!`)
    // Перенаправляем пользователя после успешной регистрации
    res.redirect('/profile');
```

Рисунок 14 - Валидация маршрута регистрации

Такая же валидация находится и при входе уже в зарегистрированную учетную запись. В ней проверяется схожесть пароля и ФИО с имеющимися данными о пользователе в БД.

В разделе с напоминанием также были выявлены ошибки с проверкой написание даты. На рисунке 15 изображен маршрут сохранения введых в модальное окно даты для напоминания. В нем отсутствует проверка даты и ID

заметки, в котором оно вводится. Без проверки сервер может сразу выдать ошибку даже если дата введена корректно.

```
// Установка напоминания
app.post('/set-reminder', async (req, res) => {
  const { noteId, reminderDate } = req.body;

  try {
    await pool.query(
      `UPDATE notes SET deadline = $1 WHERE note_id = $2`,
      [new Date(reminderDate), noteId]
    );
    res.redirect('/time');
  } catch (err) {
    console.error(err);
    res.status(500).send('Ошибка при установке напоминания');
  }
});
```

Рисунок 15 - Некорректный маршрут сохранения напоминания.

Исправленный формат (Рисунок 16) должен содержать переменную date в которой будет хранится дата для отображение на странице веб приложения

```
app.post('/set-reminder', async (req, res) => {
  const { noteId, reminderDate } = req.body;

  if (!noteId || !reminderDate) {
    return res.status(400).json({ message: 'Не указан ID заметки или дата' });
  }

  const date = new Date(reminderDate); // ← Здесь мы создаём переменную date

  if (isNaN(date.getTime())) {
    return res.status(400).json({ message: 'Некорректный формат даты' });
  }

  try {
    await pool.query(
      `UPDATE notes SET deadline = $1 WHERE note_id = $2`,
      [date, noteId] // ← Теперь всё работает
    );

    return res.json({ message: 'Напоминание успешно установлено' });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Ошибка при установке напоминания' });
  }
});
```

Рисунок 16 - Правильный маршрут с проверкой введеных данных

Каждая строчка и функция подробно закомментирована в самих строчказ кода для лучшего понимания всех последовательностей в скрипте бэкенда. Такие валидации содержатся в остальных маршрутах обработки данных при создании ключевых объектов сайта и при их эксплуатации.

Самостоятельное ревьюирование предполагает, что разработчик самостоятельно перечитывает и анализирует написанный им код до его отправки на согласование или коммита. Этот этап позволяет выявить очевидные ошибки, проверить соответствие кодирования стандартам проекта, убедиться в корректности логики, а также улучшить читаемость и структуру кода. Разработчик может использовать чек-лист, инструменты статического анализа, линтеры и тесты для более тщательной проверки.[27] Данный подход экономит время команды, снижает количество мелких замечаний на последующих этапах и способствует повышению ответственности программиста за качество своей работы.

После самостоятельного ревью следует неформальное, при котором коллега или несколько участников команды кратко просматривают изменения, обсуждая их в дружеской, непринуждённой форме - лично или через сообщения в чате. Особенностью этого этапа является отсутствие строгих правил и формальностей: акцент делается на быстрое получение обратной связи, выявление потенциальных проблем и обмен мнениями. Неформальное ревью особенно эффективно в маленьких командах или при срочных задачах, когда требуется оперативная проверка без излишней бюрократии.[28]

Такой двухэтапный подход - самостоятельное ревью перед неформальной проверкой - обеспечивает баланс между индивидуальной ответственностью и командным взаимодействием. Он помогает заранее отсеять типовые ошибки, ускоряет процесс согласования и делает финальную проверку более продуктивной. В результате повышается качество кода, укрепляется культура самоконтроля и создаются предпосылки для построения культуры коллективного владения кодом.

4 Рефакторинг программного кода

Во время самой разработки каждый раз пересматривая код программист спокойно может находить свои ошибки и их обрабатывать. По мимо ошибок в процессе разработки также будут выявляться отсутствие каких-либо деталей из-за которых работа пользователя на стороне клиентской части может показаться не до конца выполненной. Для выявления и добавление таких деталей существует процесс рефакторинга.

Рефакторинг программного кода -это процесс улучшения структуры и качества существующего кода без изменения его внешнего поведения. Основная цель рефакторинга - повысить читаемость, поддерживаемость и расширяемость приложения, а также снизить риск возникновения ошибок при дальнейшей разработке. Это не добавляет новых функций, а помогает сделать уже реализованные части более понятными, логически связанными и соответствующими общепринятым стандартам программирования.

Рефакторинг включает в себя множество действий: от простого переименования переменных и выделения повторяющихся фрагментов в отдельные функции до более сложных изменений, таких как переработка архитектуры, внедрение паттернов проектирования или разделение монолитного приложения на модульные компоненты. Он позволяет устранить технический долг, повысить производительность и облегчить дальнейшее развитие проекта.

К основным признакам, указывающим на необходимость рефакторинга, относятся: дублирование кода, избыточная сложность, плохая читаемость, тесные связи между компонентами, отсутствие тестов и нарушение принципов SOLID. Грамотный рефакторинг проводится постепенно, с постоянной проверкой работоспособности, чтобы избежать появления новых багов.[10] Он способствует созданию более устойчивой, масштабируемой и легко поддерживаемой системы, упрощает работу как текущих, так и будущих разработчиков.

Изучая разработанный локальный сервер. можно сказать что сколько бы новшеств не было введено, с каждым разом скрипт можно обновлять и добавлять новые функции и логики для полноценной работы веб-приложения. В нашем случае, рефакторинг будет проведен так чтобы веб-приложение могло стать начальным объектом для дальнейшей публикации в интернет браузерах.

Первоначально необходимо рассмотреть логику работы маршрутов аутентификации. На рисунке 17 наглядно показан маршрут авторизации. Он работает исправно и код построен правильно с добавлением валидации. Однако для пользователя не приходят оповещения об успешной авторизации или же вывод ошибки неправильного пароля или ФИО пользователя. Более того в маршруте прописан alert. Alert в JavaScript — это встроенная функция для отображения диалогового окна с сообщением. Оно появляется поверх всех остальных элементов страницы и требует от пользователя нажатия кнопки «ОК» для продолжения работы.

```
// Маршруты аутентизации (ваши существующие)
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  try {
    const result = await pool.query('SELECT * FROM users WHERE username = $1', [username]),
    if (result.rows.length > 0) {
      const user = result.rows[0];
      const validPassword = await bcrypt.compare(password, user.password);
      if (validPassword) {
        req.session.userId = user.id;
        res.redirect('/profile');
      } else {
        res.redirect('/login');
      }
    } else {
      res.redirect('/login');  // Проверка пароля и существования ФИО пользователя
    }
  } catch (err) {
    alert('Ошибка авторизации')
    console.error(err);
    res.redirect('/login');
  }
});
```

Рисунок 17 - Маршрут авторизации до рефакторинга

Рефакторинг данной ситуации заключается в удалении всех функций alert(), а для реализации ошибок авторизации необходимо использовать библиотеку j.query. Через её оператор «?» к объекту response res.redirect добавляется оператор и через оператор присваивается переменная ошибки и её тип. Тем самым по переходу по URL вместе с переходом сервер обрабатывает ошибку и в случае её возникновение выводит на шаблоне страницы EJS.

```
app.get('/login', (req, res) => {
  const error = req.query.error;
  res.render('login.ejs', { error });
});

// Маршруты аутентизации
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  try {
    const result = await pool.query('SELECT * FROM users WHERE username = $1',

    if (result.rows.length > 0) {
      const user = result.rows[0];
      const validPassword = await bcrypt.compare(password, user.password);

      if (validPassword) {
        req.session.userId = user.id;
        return res.redirect('/profile');
      } else {
        // Неверный пароль
        return res.redirect('/login?error=invalid_password');
      }
    } else {
      // Пользователь не найден
      return res.redirect('/login?error=user_not_found');
    }
  } catch (err) {
    console.error(err);
    return res.redirect('/login?error=server_error');
  }
});
```

Рисунок 19 - Обновленный маршрут для авторизации

Также на стороне клиентской части для переменной error устанавливается красный цвет

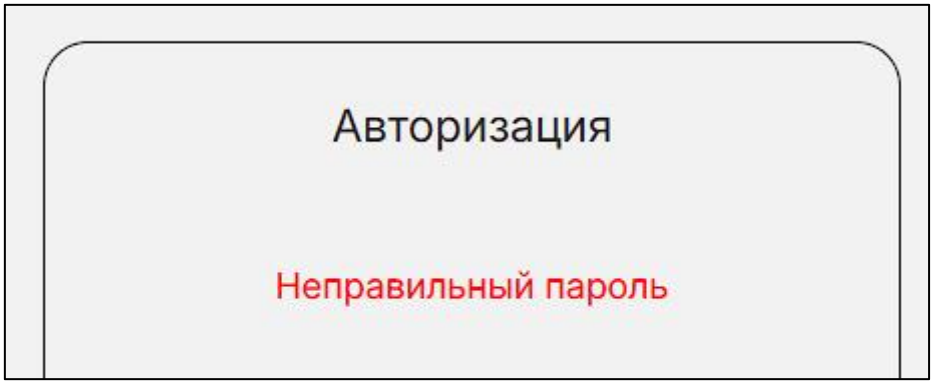

Рисунок 20 - Всплывающий текст при вводе неправильного пароля

Таким образом для пользователя был обновлён код, чтобы выводились ошибки при неправильном вводе данных.

Обычно рефакторинг производится на стороне сервера, но так как разработка включает в себя и клиентскую часть, то в ней также следует провести Рефакторинг по необходимости. Рефакторинг клиентской части представляет собой процесс улучшения структуры, организации и читаемости клиентской части кода без изменения внешнего поведения интерфейса.[17] Это включает в себя переработку HTML-разметки, CSS-стилей и JavaScript-логики с целью сделать их более понятными, поддерживаемыми и расширяемыми. Он предполагает упорядочение файловой структуры проекта, чтобы элементы интерфейса хранились логически и были легко доступны для редактирования. Например, шаблоны страниц разделяются на повторяющиеся компоненты, такие как заголовок, меню или кнопки, что способствует повторному использованию кода и уменьшению дублирования. Также происходит оптимизация EJS-шаблонов, где вместо монолитных страниц создаются базовые макеты и частичные представления, которые можно переиспользовать на разных страницах.

Важной частью рефакторинга является разделение и модулизация, при которой общий стиль разбивается на отдельные файлы по функциональным модулям - например, стили для профиля, заметок, групп и навигации.[13]

Разделение и модулизация CSS-стилей, при которой общий стиль разбивается на отдельные файлы по функциональным модулям - например, стили для профиля, заметок, групп и навигации. Это позволяет быстрее находить нужные правила, избегать конфликтов классов и ускоряет загрузку страниц за счёт целевой подгрузки стилей.

Рассматривая структуру нашего сайта на каждой странице расположены похожие блоки такие как шапка, главное меню и подвал. Данные блоки можно вынести в отдельную папку и через ejs прописывать данные компоненты через строки кода. На рисунке 21 представлена файловая система веб-приложения.

Рисунок 21 - Обновленная файловая структура веб-приложения

В папке views для отдельных компонентов создается папка partials где будут повторяющиеся блоки. Теперь повторяющиеся блоки благодаря функционалу ejs прописываются только одно строчкой и присваивают стили страницы, в которых они отображаются (Рисунок 22). Скрипт прописывается в каждых страницах с функционалом заметок. Только на странице профиля остался блок шапки так как в нем нет поле поиска заметок из-за отсутствия для него функционала на странице.

```
<body>
  <%- include('partials/header') %>
  <section>
    <div id="search-results" class="search-results-container">
    </div>
    <div class="content-profile">
      <%- include('partials/menu') %>
      <div class="content">
```

Рисунок 22 - Обновленный код фронтенда

В ходе рефакторинга серверный код был разделён на модули, такие как маршруты, контроллеры, сервисы, middleware и конфигурации, что позволило избежать дублирования, и ускорить дальнейшую разработку. Конфиденциальные данные и параметры подключения были вынесены в отдельный .env-файл, что повысило безопасность и гибкость настройки приложения. Фронтенд-часть также была реорганизована: созданы повторно используемые шаблоны (header.ejs, footer.ejs, menu.ejs), CSS-стили разделены по модулям, а JavaScript-логика - на отдельные файлы, соответствующие функциональным разделам приложения. Рефакторинг способствовал улучшению взаимодействия между клиентом и сервером, централизации обработки данных и оптимизации работы с базой данных. В результате приложение стало проще в сопровождении.

5 Программа и методика испытания веб-приложения

Методика испытаний веб-приложения - комплекс мероприятий, направленных на проверку корректности функционирования всех компонентов системы перед её внедрением или обновлением.[15] Под этим понимается чётко структурированный процесс тестирования, включающий определение целей, задач, этапов, видов тестов, используемых инструментов и критериев успешности. Программа охватывает как функциональное тестирование (проверка работы форм регистрации, авторизации, создания заметок, отправки напоминаний через Telegram-бота и т.д.), так и нагрузочное, чтобы убедиться в устойчивости приложения под высокой нагрузкой. Методика включает также тестирование интерфейса (UI) , проверку отображения страниц на разных устройствах и браузерах, безопасность (например, защиту сессий, хеширование паролей), а также интеграционное тестирование, чтобы убедиться в корректной работе взаимодействия между сервером, базой данных и внешними сервисами, такими как Telegram API. Все выявленные ошибки фиксируются, классифицируются по степени критичности и передаются на доработку, что позволяет обеспечить стабильную, безопасную и удобную работу приложения для конечных пользователей.[16]

Процесс тестирования имеет некоторые виды:

– функциональное тестирование - определяет, работает ли каждая функция веб-приложения согласно спецификации;

– тестирование удобства пользования - определяет характеристики взаимодействия пользователя с веб-приложением, чтобы обнаружить недостатки и эффективно их устранить;

– тестирование интерфейса пользователя - в ходе диагностики оценивают внешний вид интерфейса, его удобство и простоту использования;

– тестирование производительности - проводится проверка веб-приложения на способность выдерживать большие нагрузки;

– диагностика безопасности - позволяет оценить степень защиты приложения от разного рода уязвимостей, атак и других рисков.

По мимо видов также различаются и подходы тестирования:

– модульное тестирование - проверка отдельных частей базы кода с помощью модульных тестов;

– интеграционное тестирование - проверка различных сегментов кода сайта в виде независимых функций или модулей с помощью тестовой программы или прочих инструментов;

– системное тестирование - тестирование сайта на уровне пользовательского интерфейса и функционала вроде авторизации, регистрации и прочих потоков;

– приёмочное тестирование - это последняя стадия тестирования, на которой полностью собранное приложение с данными проверяется в продакшн-среде или среде интеграции.

Нынешняя разработка будет содержать функциональное тестирование с подходами модульного и системного тестирования, тем самым полностью подготовив веб-приложение к демонстрации предприятию и в дальнейшем его обновления.

Для тестирования веб-приложения необходимо обозначить, что конкретно будет тестироваться сразу же после разработки. Стоить отметить что после внедрение какой-либо функции во время самой разработки проводились небольшие тестирование для выявления ошибок и проверки работоспособности функционала.[18] В данном случае через программу испытания веб-приложение следует провести проверку главным задачам сайта, а именно:

- вход и регистрация пользователя;
- создание заметки;
- удаление заметки;
- создание заметки в группе;
- удаление участника из группы;

Скрины полноценных форм и заметок будут представлены в Контрольном примере.

Аутентификация пользователя должна в обязательном порядке содержать валидацию данных, точнее говоря, проверка корректности пароля, имени.[20] Так на сервере в маршруте авторизации ранее пройдя рефакторинг программного кода при внесение неправильного пароля или некорректного имени высвечивается оповещение: Неправильный пароль или Такого пользователя не существует. Оповещение о неправильном пароле расположено в разделе описания рефакторинга, а оповещение о некорректном имени показано на рисунке 23.

При регистрации когда пользователь будет вводить ФИО, которое ранее было зарегистрировано также будет выводит оповещение о некорректных данных (Рисунок 24).

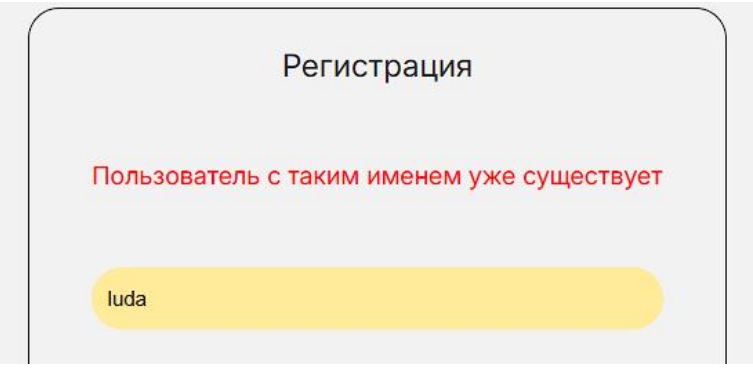

Рисунок 23 - Оповещение о неправильно введеном имени при входе в учетную запись

Оповещение для регистрации сделано специально для того чтобы, в базе данных не было повторяющихся имён иначе могут произойти ошибки у обоих пользователей при регистрации или утечка личных данных каждого пользователя.

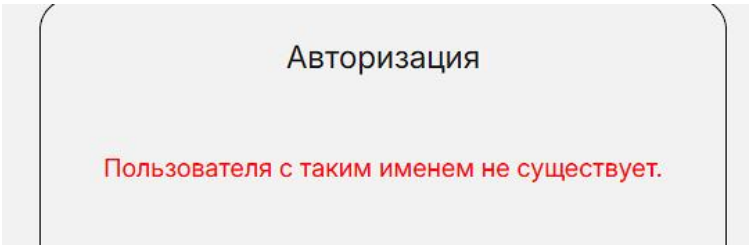

Рисунок 24 - Оповещение об уже существующем введеного имени пользователя при регистрации

Функционал по созданию заметок более сложен и требует в каждом процессе точного описания кода для функции, чтобы процесс по управлению заметок работал без ошибок. При создании заметки следует учесть, чтобы каждое обязательное поле было заполнено. В разработанной форме создание заметки все поля обязательны. При создании заметки с пропуском главного поля появится всплывающий блок с оповещение о том, что поле было пропущено (Рисунок 25)

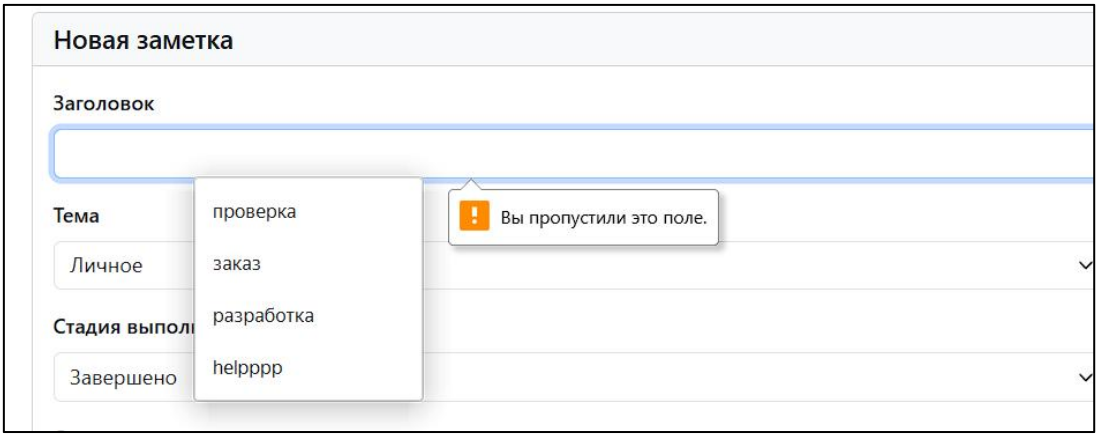

Рисунок 25 - Оповещение о пропуске главного поля при создании заметки

Когда заметка создалась успешно и прошла всю валидацию на сервере, поверх страницы всплывает showalert (специальное всплывающее окно), в котором говорится об успешном создании заметки (Рисунок 26). К данному виду всплывающего окна установлены специальные стили из библиотеки Bootstrap

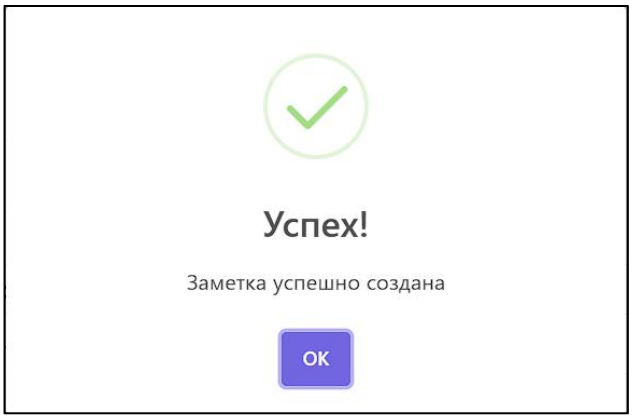

Рисунок 26 - Создание заметки при правильном введение данных для нее

Удаление заметки также является главной задаче по её управлению и относится к способу очищению ненужных или непригодных заметок. Перед удалением сервер отправляет пользователю всплывающее серверное окно с подтверждением удаления заметки, дабы убедиться в правильности решения самого пользователя(Рисунок 27). Если пользователь выбирает пункт «Нет» заметка остается а окно исчезает, если пользователь выбирает «Да» высвечивается то же самое всплывающее окно showalert() как и при создании заметки в случае правильного использования введеных данных.(Рисунок 28)

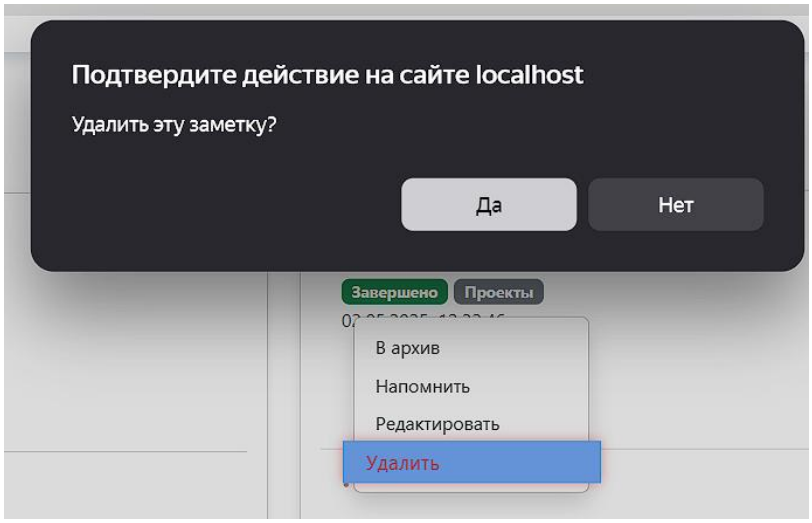

Рисунок 27 - Подтверждение удаление заметки

Таким образом подтверждение помогает пользователю избежать случайных действий нажатия важных кнопок на страницах сайта.

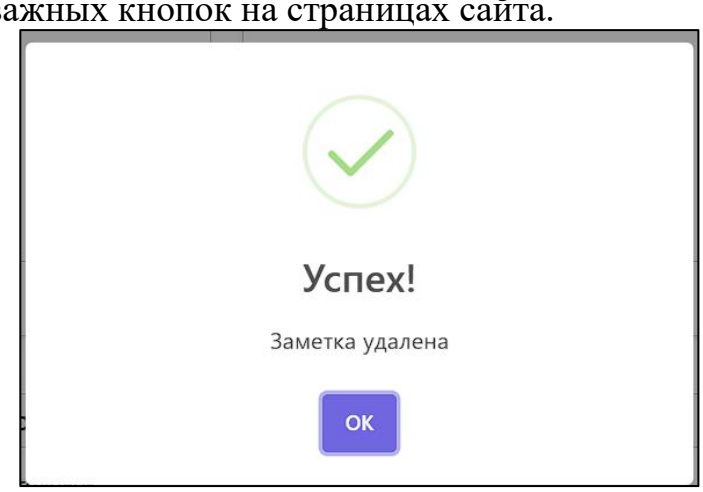

Рисунок 28 - Успешное удаление заметки

По мимо создания личных заметок пользователь также может создать заметки с другими пользователями веб-приложения. На странице «Группы» расположено похожая модальная форма как и при создании личной заметки только уже с выбором участников. При заполнение формы к обязательным полям

добавился выбор участника. При пропуске данного поля заметка не создаться а на странице выйдет ошибка с пропуском создание личной заметки (Рисунок 29)

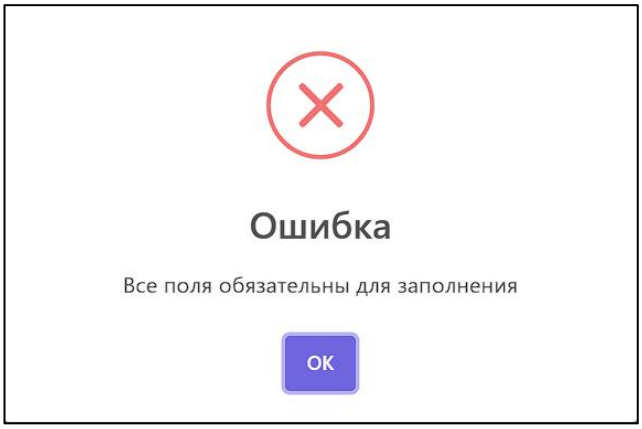

Рисунок 29 - Оповещение о пропуске главного поля при создании заметки в группе

При успешном создании заметки в группе также будет высвечиваться showalert() о создании заметки как на рисунке 26. Если на сервере не соблюдены условия валидации не были соблюдены, на сайте выйдет оповещение об ошибке в создании заметки. Такое же оповещение будет высвечиваться и в случае создание личной заметки

В момент удаление участника при нажатии кнопки удаление высветится окно о подтверждении удаление какого-либо участника (Рисунок 31). Такие подтверждения выходят на основе серверной части. При удалении участника сначала сервер проверяет существования заметки и участника в ней. Если валидация проходит участник удаляется из списка в самой заметки.

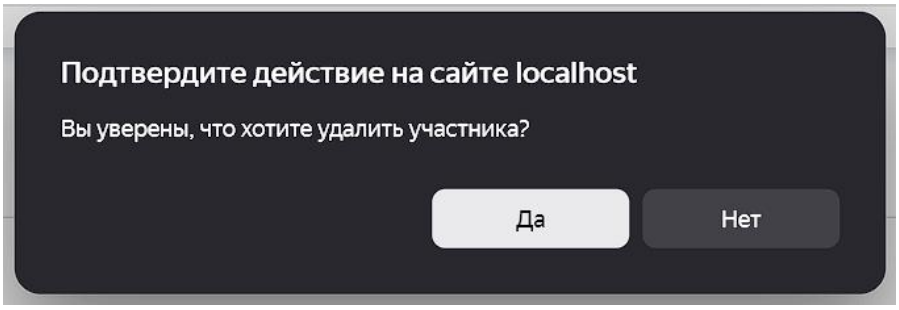

Рисунок 30 - Подтверждения удаления участника

После подтверждение в списке карточки заметки участник исчезает тем самым больше не имея доступ к созданной карточке заметки. Тем самым в заметке группа пользователей полностью управляема и в зависимости от роли может модернизироваться.

Для пользователя работа с приложением является визуальным опытом, и его перемещение по сайту определяется особенностями дизайна, установленными в ходе бизнес анализа и проектирования. Для того, чтобы итоговый сайт оставался верен заложенным идеям дизайна, необходимо обеспечить подходящий режим

визуального тестирования и утверждения вносимых в базу кода изменений. Проделав все описанное, можно сказать, что при правильной конфигурации всех тестовых кейсов и плавном протекании потока CI/CD, программистам потребуется меньше времени проводить за проверкой и исправлением ошибок, вместо этого занимаясь разработкой нового функционала.

После предварительного функционального тестирования следующим важным шагом идет проверка совместимости устройства и браузера, что позволяет понять, как приложение выглядит и работает на их разных комбинациях.[25]

В связи с тем, что разработчику недоступен весь спектр устройств для тестирования, отловить всевозможные баги становится сложно. Несмотря на то, что в веб-среде установлены определенные стандарты и спецификации, при их трактовке различными браузерами возникают разночтения, чему способствуют отличия в реализации браузерных движков и используемых интерпретаторов JS-кода. В целом современный браузер сам по себе предоставляет инструментарий для открытия и отладки любого аспекта приложения.

Прогоны тестов на всех этапах и на всех уровнях должны отслеживаться, поскольку автоматизация поможет лишь с выполнением и обнаружением ошибок, а для понимания серьезности неполадок и поиска способа их исправления необходима хорошая коммуникация внутри команды.

Существует множество подходов к тестированию сайтов, и здесь можно применить разные комбинации инструментов, а с появлением новых облачных технологий веб-тестирование стало еще богаче на функционал и надежнее. Возможность сотрудничать с командой на одной платформе для построения планов тестирования дает превосходный результат, поскольку все участники имеют доступ к общим ресурсам.

Заключение

На сегодняшний день в сети интернет, существует большое количество сайтов, которые выглядят привлекательно, имеют чёткие определенные информационные разделы и удобный функционал.

Сайты хранят в себе массу полезной и важной информации, они находятся в открытом доступе для любого пользователя, а потому так широко пользуются спросом на сегодняшний день.

Каждый сайт ориентирован на группу пользователей, которые объединяются одними интересами и ищут информацию определенного характера, а если эта информация хранится в одном месте, собрана по крупицам и может дать развернутый ответ на вопрос, который ставит перед собой посетитель — такой сайт будет цениться вдвойне.

Исходя из этого, можно с уверенностью предположить, что сайты создаются именно для сбора и хранения информации, которая находит своего конечного потребителя, а чем больше таких потребителей посещает сайт, тем большую выгоду из своего проекта, извлекает его владелец.

Во всем мире превыше всего ценится качество, поэтому информация на сайтах обязательно должна быть качественной, интересной, развернутой и полезной. На какую бы тему ни был создан сайт, он должен раскрывать ее максимально обширно, от самого начала и до малейших мелочей.

Если говорить об актуальности создания сайта, то можно утверждать что любой качественный проект - будет актуален в своем информационном сегменте, если его довести до ума, работать над ним и постоянно наполнять качественным контентом.

Интернет - сравнительно молодой, инновационный источник информации, который только набирает свои масштабы во всем мире. С каждым днем все дальше распространяется интернет-покрытие, практически в каждой семье уже имеются интернет-проводники, такие как современные мобильные телефоны, персональные компьютеры, ноутбуки, планшеты и другие коммуникационные устройства.

Количество активных пользователей в сети Интернет также стремительно возрастает и если сегодня вас сайт привлекает только 100 человек в сутки, то уже через год, если проект не стоит на месте и развивается, эта цифра может увеличиться в разы.

Благодаря тому, что интернет является неисчерпаемым источником предоставления разнообразной информации, сайты стали чаще использоваться даже в компаниях со старыми видами планировки бизнеса. Все возможные источники зафиксированы электронно, что способствует формированию удобного рабочего места. На какую бы тему ни был создан сайт, он должен раскрывать ее максимально.

Но стоит уделять внимание не столько созданию своего сайта, сколько его наполнению, потому что актуальным будет только качественный сайт, который пробьется в топ выдачи поисковой системы, заинтересует пользователя и принесет определенную выгоду своему владельцу.

В ходе проведения работы были решены все поставленные задачи, а именно:

– изучена предметная область;

– разработан сайт с использованием современных технологий и предоставляет пользователям из целевой группы наиболее нужные им сервисы;

– разработан удобный интерфейс и дизайн сайта соответствует предполагаемым предпочтениям целевой группы;

– уделено особое внимание безопасности пользователей, контролю над корректностью входных данных, защита от переполнения базы данных;

– реализован внутренний код для полной работы с заметками.

Результатом работы на производственной практике стал разработанный web-сайт с возможностью создания заметок «CheckTask».

Для реализации сайта были использованы следующие средства:

– Figma;
– Visual Studio Code;
– языки разметки html и css;
– EJS;
– JavaScript;
– PostgreSQL, pgadmin4;
– Express, cors, body-parser, multer.

Результаты разработки представляют собой программно-технический комплекс индивидуального использования, предназначенный для автоматизации работ связанные с проектированием задач, проектов, построение списков и целейБ а именно разработка веб-приложения «CheckTask».

Хороший сайт должен быть информационно насыщенным и это одно из самых главных условий, которые требуются для создания успешного сайта. Причём информация на портале должна быть.

Чем больше пользы сможет принести пользователям сайт, тем более рациональным и актуальным окажется его создание.

Применение разработанного сайта позволяет существенно повысить эффективность управление данными внутри учебного заведения, при возникновений ошибок в главных электронных журналах.

Список источников

1. "ГОСТ Р ИСО/МЭК 27001-2019 Системы менеджмента информационной безопасности. Требования".;
2. Кори Альтхофф. «Сам себе программист» / Кори Альтхофф. / 2019;
3. Эрик Эванс. «Предметно-ориентированное проектирование. Структуризация сложных программных систем» / Эрик Эванс / 2020;
4. Дональд Кнут. «Искусство программирования» / Дональд Кнут / 2019;
5. Абельсон, Сассман. «Структура и интерпретация компьютерных программ» / Абельсон, Сассман / 2021;
6. Мартин Фаулер. «Шаблоны корпоративных приложений» / Мартин Фаулер / 2022;
7. ДеМарко, Листер. «Человеческий фактор. Успешные проекты и команды» / ДеМарко, Листер / 2023;
8. Михаил Круг. «Не заставляйте меня думать. Веб-юзабилити и здравый смысл» / Михаил Круг / .2024;
9. Г. Л. Макдауэлл. «Карьера програмиста» / Г. Л. Макдауэлл / 2019;
10. Э. Гамма, Р. Хелм «Приемы объектно-ориентированного проектирования. Паттерны проектирования» / 2019;
11. Роберт Мартин. «Идеальный программист» / 2021;
12. Фредерик Брукс. «Мифический человеко-месяц, или Как создаются программные системы» / 2022;
13. Фримен, Робсон. «Head First. Паттерны проектирования» / 2020;
14. Алексашина, И. Ю. Методика преподавания интегрированного курса "Естествознание" с использованием ресурсов дистанционного обучения : метод. рек. / И. Ю. Алексашина, О. А. Абдулаева ; под науч. ред. И. Ю. Алексашиной. - СПб.;
15. Бушина. - Текст: электронный // Образование. Наука. Карьера : сборник научных статей 2-й Междунар. науч.-метод. конф. Курск, 22 янв. 2019г. - Курск, 2019.;
16. Вайндорф-Сысоева, М. Е. Методика дистанционного обучения : учебное пособие для вузов /М. Е. Вайндорф-Сысоева, Т. С. Грязнова, В. А. Шитова ; под общей редакцией М. Е. Вайндорф-Сысоевой. - Москва :Юрайт, 2018. - 194 с. - (Высшее образование);
17. Волкова, В. А. Организация дистанционного обучения в условиях обновления образования в Санкт-Петербурге / В. А. Волкова;
18. "ОК 009-2016. Общероссийский классификатор специальностей по образованию" (принят и введен в действие Приказом Росстандарта от 08.12.2016 N 2007-с;
19. ГОСТ Р 50922-2006 «Защита информации. Основные термины и определения»;
20. Сайт ЭБС Юрайт: [Электронный ресурс] ссылка: https://urait.ru/bcode/413604. - 19.04.2025;

21. Сайт Социальная сеть работников образования «Наша сеть» nsportal.ru. [Электронный ресурс] ссылка: https://nsportal.ru/shkola/raznoe/library/2019/11/11/organizatsiya-distantsionnogo-obucheniya-v-usloviyah-obnovleniya. - 22.04.2025;

22. Сайт Интернет-технологии в образовании : сборник материалов Всерос. науч.-практ. конф. [Электронный ресурс] ссылка: https://elibrary.ru/item.asp?id=41384586. - 26.04.2025;

23. Сайт ПЕДСОВЕТ. Персональный помощник педагога: [Электронный ресурс] ссылка: https://pedsovet.org/beta/article/kak-s-pomosuaklass-organizovat-distancionnoe -obucenie-vo-vrema-kanikul. – Заглавие с экрана. - 30.04.2025;

24. Сайт Мастерская Марины Курвитс «Как организовать обучение в дистанционном формате [Электронный ресурс] ссылка: https://marinakurvits.com/kak_organizovat_distancionnoe _obuchenie/#. - 09.05.2025;

25. Сайт Построение онлайн диаграмм. Онлайн конструктор: [Электронный ресурс] ссылкаL: https://draw.io (diagrams.net) - 11.05.2025;

26. Сайт Руководство по JavaScript: [Электронный ресурс] ссылка: Руководство по JavaScript, часть 1: первая программа, особенности языка, стандарты / Хабр (https:/habr.com) - 13.05.2023;

27. Сайт Как создать приложение для создания заметок с помощью HTML, CSS и JavaScript: [Электронный ресурс] ссылка: https://dev.to/aliyuadeniji/how-to-build-a-note-taking-app-with-html-css-and-javascript-37ae - 10.05.2025;

28. Сайт Построение сайта для заметок: [Электронный ресурс] ссылка: https://www.codingnepalweb.com/build-a-notes-app-in-html-css-javascript/ 15.05.2025.

Приложение А
(обязательное)

Код реализации базы данных

```
CREATE TABLE IF NOT EXISTS public.users
(
    id integer NOT NULL DEFAULT nextval('users_id_seq'::regclass),
    username character varying(50) COLLATE pg_catalog."default" NOT NULL,
    email character varying(100) COLLATE pg_catalog."default" NOT NULL,
    phone character varying(15) COLLATE pg_catalog."default" NOT NULL,
    post character varying(100) COLLATE pg_catalog."default" NOT NULL,
    password text COLLATE pg_catalog."default" NOT NULL,
    telegram_id bigint,
    CONSTRAINT users_pkey PRIMARY KEY (id),
    CONSTRAINT unique_username UNIQUE (username)
)

CREATE TABLE IF NOT EXISTS public.notes
(
    note_id integer NOT NULL DEFAULT nextval('notes_note_id_seq'::regclass),
    user_id integer NOT NULL,
    title character varying(100) COLLATE pg_catalog."default" NOT NULL,
    theme character varying(50) COLLATE pg_catalog."default" NOT NULL,
    description text COLLATE pg_catalog."default" NOT NULL,
    stage character varying(20) COLLATE pg_catalog."default" NOT NULL
DEFAULT 'planned'::character varying,
    is_archived boolean DEFAULT false,
    deadline timestamp without time zone,
    "time" timestamp without time zone DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT notes_pkey PRIMARY KEY (note_id),
    CONSTRAINT fk_user FOREIGN KEY (user_id)
      REFERENCES public.users (id) MATCH SIMPLE
      ON UPDATE NO ACTION
      ON DELETE CASCADE,
    CONSTRAINT notes_theme_check CHECK (theme::text = ANY
(ARRAY['Работа'::character varying, 'Личное'::character varying, 'Учеба'::character
varying, 'Проекты'::character varying, 'Другое'::character varying]::text[])),
    CONSTRAINT stage_check CHECK (stage::text = ANY
(ARRAY['planned'::character varying, 'in_progress'::character varying,
'testing'::character varying, 'completed'::character varying]::text[]))
)
```

```
CREATE TABLE IF NOT EXISTS public.note_users
(
    id integer NOT NULL DEFAULT nextval('note_users_id_seq'::regclass),
    note_id integer,
    user_id integer,
    role character varying(50) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT note_users_pkey PRIMARY KEY (id),
    CONSTRAINT note_users_note_id_fkey FOREIGN KEY (note_id)
        REFERENCES public.notes (note_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT note_users_user_id_fkey FOREIGN KEY (user_id)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

CREATE TABLE IF NOT EXISTS public.telegram_link_codes
(
    id integer NOT NULL DEFAULT
nextval('telegram_link_codes_id_seq'::regclass),
    user_id integer,
    telegram_id bigint,
    code character varying(16) COLLATE pg_catalog."default" NOT NULL,
    created_at timestamp without time zone NOT NULL DEFAULT now(),
    expires_at timestamp without time zone NOT NULL,
    is_used boolean NOT NULL DEFAULT false,
    CONSTRAINT telegram_link_codes_pkey PRIMARY KEY (id),
    CONSTRAINT telegram_link_codes_user_id_fkey FOREIGN KEY (user_id)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

Приложение Б
(обязательное)

Программный код до/после рефакторинга

```
// Маршруты аутентизации (ваши существующие)
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  try {
    const result = await pool.query('SELECT * FROM users WHERE username = $1', [username]);
    if (result.rows.length > 0) {
      const user = result.rows[0];
      const validPassword = await bcrypt.compare(password, user.password);
      if (validPassword) {
        req.session.userId = user.id;
        res.redirect('/profile');
      } else {
        res.redirect('/login');
      }
    } else {
      res.redirect('/login');  // Проверка пароля и существования ФИО пользователя
    }
  } catch (err) {
    alert('Ошибка авторизации')
    console.error(err);
    res.redirect('/login');
  }
});
```

Рисунок Б.1 - Маршрут авторизации до рефакторинга

```
app.get('/login', (req, res) => {
  const error = req.query.error;
  res.render('login.ejs', { error });
});

// Маршруты аутентизации
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  try {
    const result = await pool.query('SELECT * FROM users WHERE username = $1',

    if (result.rows.length > 0) {
      const user = result.rows[0];
      const validPassword = await bcrypt.compare(password, user.password);

      if (validPassword) {
        req.session.userId = user.id;
        return res.redirect('/profile');
      } else {
        // Неверный пароль
        return res.redirect('/login?error=invalid_password');
      }
    } else {
      // Пользователь не найден
      return res.redirect('/login?error=user_not_found');
    }
  } catch (err) {
    console.error(err);
    return res.redirect('/login?error=server_error');
  }
});
```

Рисунок Б.2 - Маршрут авторизации после рефакторинга

```
<body>
    <header>
        <div class="text-logo"><img src="/img/log (1).svg" alt="logo">CheckTask</div>
        <div class="search-container">
          <div class="input-with-icon">
            <i class="icon"><img src="/img/search.svg" alt="icon"></i>
            <input type="text" id="search" placeholder="поиск" required>
            <button id="clear-search" class="clear-button">×</button>
          </div>

        </div>
        <div class="btn-menu"><a href="/profile" class="logout">Профиль</a></div>
    </header>
    <section>
        <div id="search-results" class="search-results-container">
        </div>
        <div class="content-profile">
            <div class="main-menu">
                <a href="/main" class="point-menu"><img src="img/bookmark.svg" alt="">Заметки</a>
                <a href="/time" class="point-menu"><img src="/img/meter.svg" alt="">Напоминания</a>
                <a href="/archive" class="point-menu"><img src="/img/archive (1).svg" alt="">Архив</a>
                <a href="/group" class="point-menu"><img src="/img/user-add.svg" alt="">Группы</a>
            </div>
```

Рисунок Б.3 - Начальный код на страницах веб-приложения до рефакторинга

```
<body>
    <%- include('partials/header') %>
    <section>
        <div id="search-results" class="search-results-container">
        </div>
        <div class="content-profile">
            <%- include('partials/menu') %>
            <div class="container mt-4">
```

Рисунок Б.4 - Начальный код нам страницах после рефакторинга

Приложение В
(обязательное)

Контрольный пример

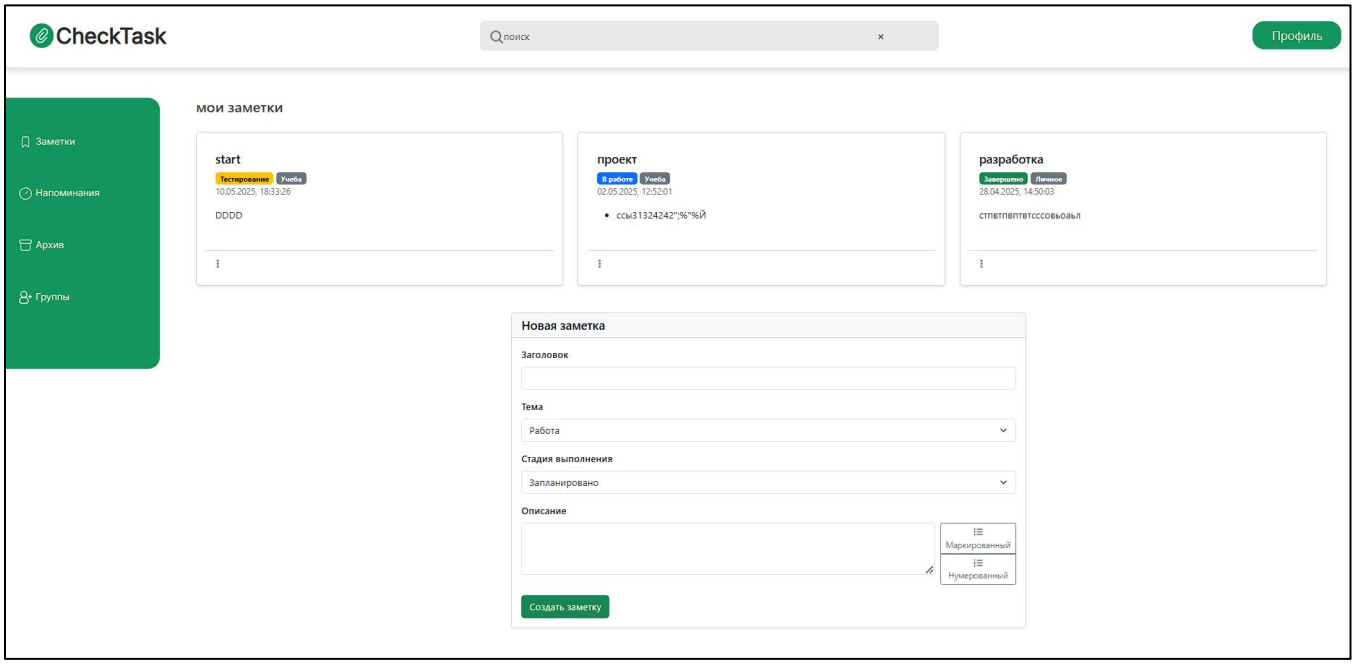

Рисунок В.1 - Контрольный пример главной страницы

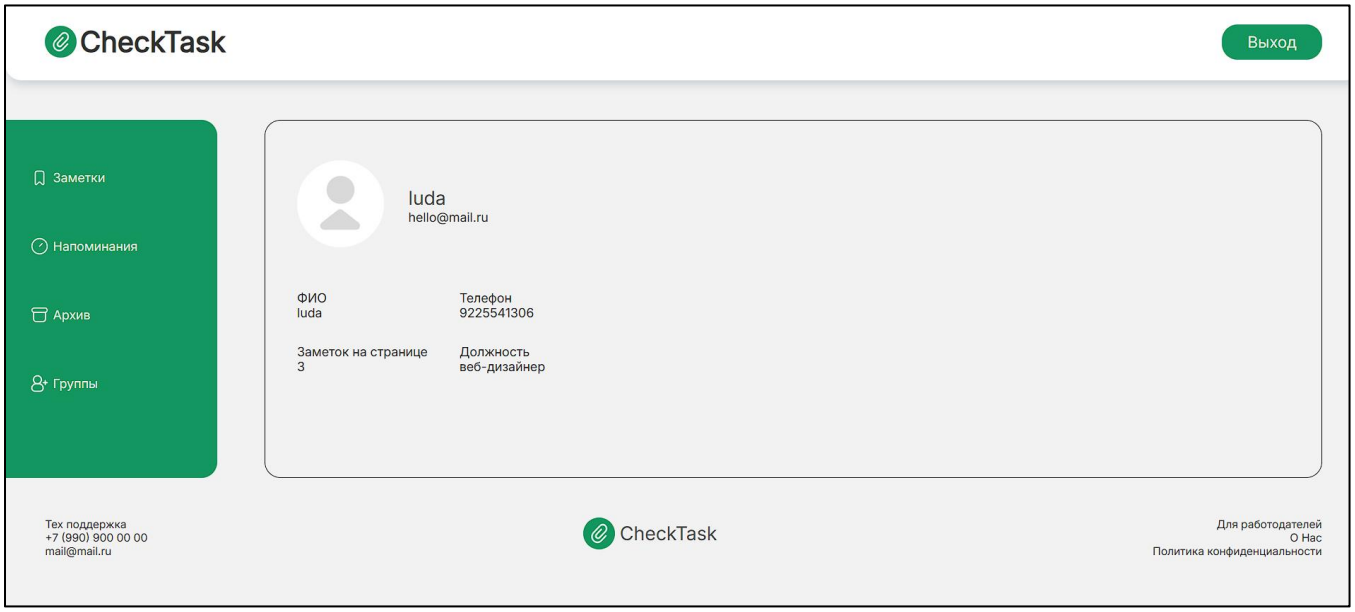

Рисунок В.2 - Контрольный пример личного кабинета

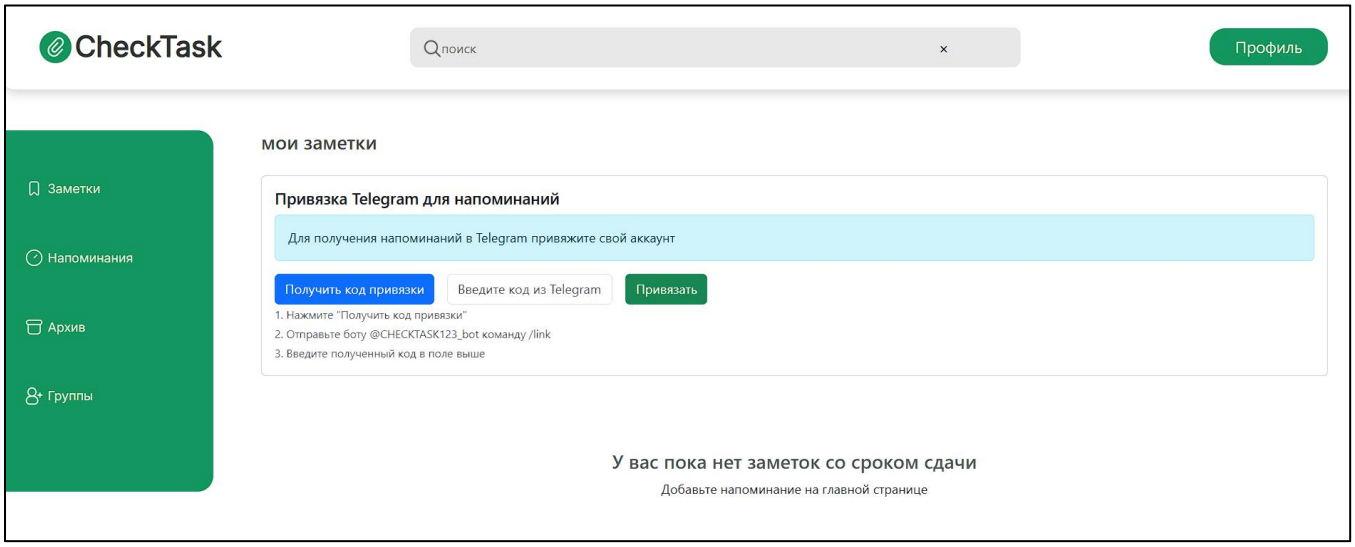

Рисунок В.3 - Контрольный пример страницы напоминаний

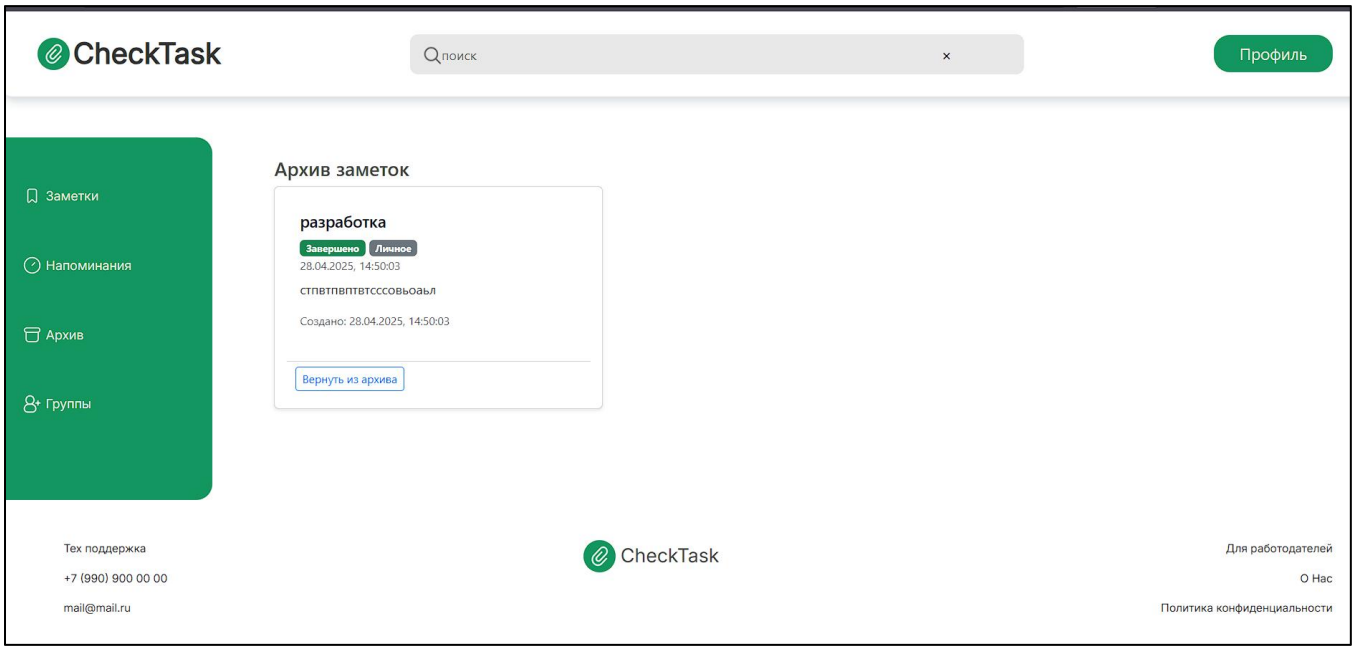

Рисунок В.4 - Контрольный пример страницы архива

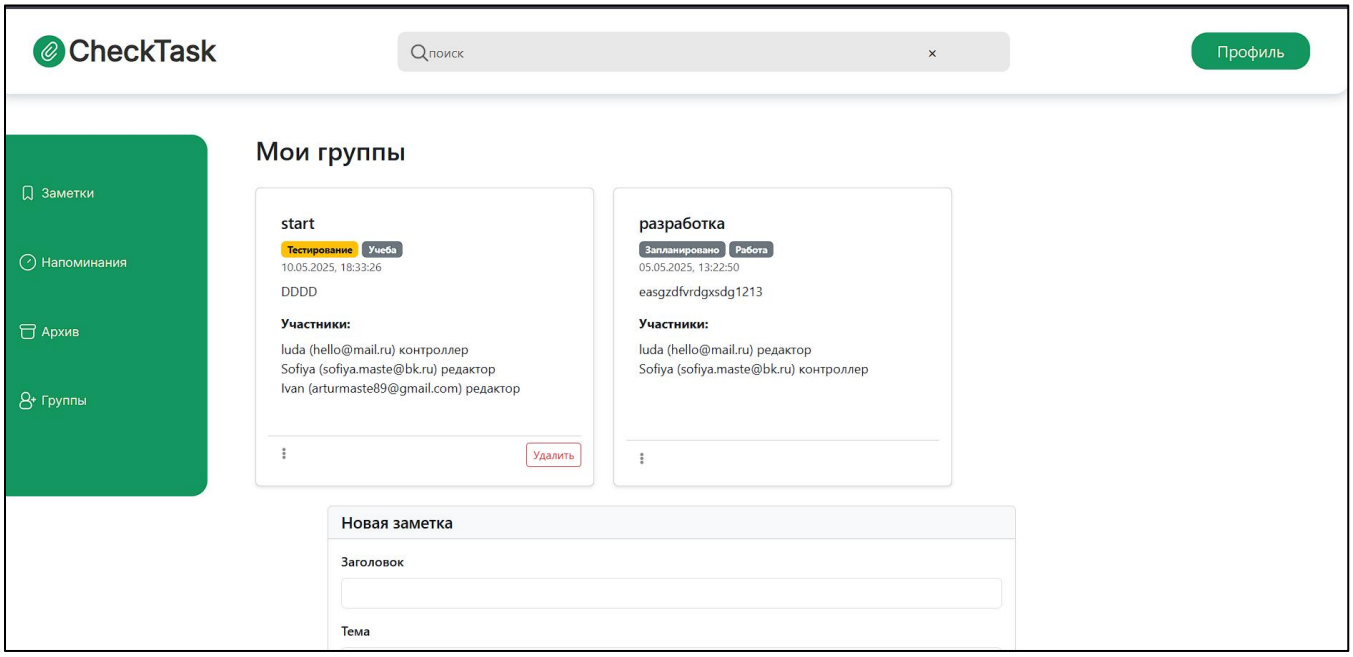

Рисунок В.5 - Контрольный пример страницы с групповыми заметками