Содержание

1. Техническое задание

1.1 Название проекта.

Тема: “Система заметок GTD.”

1.2 Исполнитель.

Студент 3-го курса бакалавриата КФ МТГУ им. Баумана по направлению “Информатика и вычислительная техника” профиль “Системы обработки информации и управления” Ли Р. В.

2. Назначение и цели создания системы.

2.1 Назначение системы.

Веб-приложение разработано для предоставления сервиса учета выполнения и структуризации проектных задач пользователей.

2.2 Цель создания системы.

Создание системы бизнес сервиса.

3. Характеристика системы автоматизации.

Объектом автоматизации является структуризация задач по методологии "Getting things done".

4. Требования к системе.

4.1 Функциональные требования.

- регистрация и авторизация на сайте;
- создание задач, проектов, статус-досок и таблиц;
- редактирование задач, проектов, статус-досок и таблиц;
- преобразование задач в проекты;
- изменение статуса задач и проектов;
- редактирование профиля пользователя.

4.2 Состав и содержание работ по созданию системы.

Работы по созданию системы выполняюься в четыре этапа:
25% к 4 нед., 50% к 7 нед., 75% к 10 нед., 100% к 14 нед.

5. Состав и содержание работ по созданию системы

5.1 Этапы проекта.

- Анализ требований и сбор данных об аналогах.
- Проектирование базы данных и пользовательского интерфейса.
- Разработки базы данных и интерфейса.
- Тестирование и оптимизация системы.
- Поддержка и обновление системы.

6. Порядок контроля и приёмки системы

Контроль и приемка осуществляются на каждом этапе разработки согласно утвержденному плану.

7. Требования к документированию

Требуется продоставить:

a) Техническое задание в соответсвии с ГОСТ 34.602-89

b) Расчетно-пояснительную записку, включающую в исследовательскую часть, проектно-конструкторскую часть и проектно-технологическую часть, включающую в себя руководство пользователя и руководство программиста (администратора). Расчетно-пояснительная записка выполняется с учетом требований, предусмотренных ГОСТ 7.32-2001 и 2.105-95

8. Источники разработки.

- Технические спецификации и обучающий материал.
- Библиотери и фреймворки для разработки базы данных и пользовательского интерфейса

2. Научно-исследовательская часть

2.1 Введение

Система управления задачами Getting Things Done (GTD) достаточно популярна среди энтузиастов, стремящихся к упорядоченности и повышению продуктивности. Суть методологии заключается в создании системы, которая помогает держать все задачи и идеи под контролем, освобождая разум для более важных дел.

На сегодняшний день существует множество сервисов и инструментов, которые позволяют применять принципы GTD в повседневной жизни: от классических бумажных планировщиков до современных цифровых приложений. Однако ни один из этих инструментов не направлен на реализацию методологии GTD напрямую. Большинство сервисов предоставляют лишь общие функции планирования, оставляя пользователю задачу самостоятельно адаптировать методологию под их функционал.

Это создает определенные сложности для тех, кто хочет внедрить GTD в свою жизнь, так как для эффективного использования приходится либо дорабатывать существующие инструменты, либо искать компромиссы между удобством и соответствием методологии. Тем не менее, популярность GTD растет, а сообщество энтузиастов продолжает делиться опытом и находить новые подходы к интеграции этой системы в повседневную практику.

2.2 Постановка задачи проектирования

Задачей проектирования данной курсовой работы является реализация веб-сервиса для хранения и организации задач по методологии Getting Things Done пользователем.

2.3 Описание предметной области

Система заметок работает на основе методологии "Getting Things Done", которая разбивает задачи по сложности, важности и условиям выполнения. Элементы системы "Getting Things Done", разработанной Дэвидом Алленом, предусматривают несколько "куч" задач с различными характеристиками, описанными ниже:

1. Задачи — действия и цели пользователя.
2. Проекты — задачи, для выполнения которых требуется несколько шагов.
3. Контексты — условия, при которых эти задачи могут быть выполнены.
4. Справочные материалы.
5. Календарь.
6. Списки ожидания — задачи, выполнение которых зависит от внешних факторов.

Краткое описание методологии GTD

Применение методологии состоит из 4 шагов:

Запись → Обработка → Пересмотр → Выполнение.

Обработка заключается в присвоении категории задачам по следующим вопросам:

1. Является ли это задачей?
2. Эта задача одношаговая?
3. Если это задача, зависит ли она от внешних факторов?

4. Эта задача неотложная?

Следуя этим вопросам, задачи распределяются по нескольким категориям:

1. Корзина — изначальный список задач.
2. Проекты — список проектов.
3. Отложенные задачи — задачи, не требующие немедленного выполнения.
4. Текущие задачи — задачи, выполнение которых началось.
5. Ожидание — задачи, зависящие от внешних факторов.
6. Выполненные задачи.

Понятия системы

- *Система заметок GTD* — система заметок на основе методологии "Getting Things Done", разработанной Дэвидом Алленом, предназначенная для структурированного контроля выполнения задач.
- *Пользователь* — любое зарегистрированное лицо, которое пользуется данной системой.
- *Роль* - роль пользователя в доске, от которой зависит его уровень доступа.
- *Задача* — любая заметка, добавленная пользователем для дальнейшего распределения и выполнения.
- *Проект* — это задача, на выполнение которой требуется несколько шагов (подзадач).
- *Статус-таблица* — это список, в который заносятся задачи и проекты в зависимости от статуса их выполнения.
- *Доска* — это хранилище статус-таблиц, в рамках которого создаются задачи и проекты.

Реализация функционала.

В досках по умолчанию присутствуют 5 статус-таблиц: *Проекты*, *Отложенные задачи*, *Текущие задачи*, *Ожидание*, *Выполнено*. При этом можно добавить дополнительные *статус-таблицы* под нужды пользователя. Дополнительные *статус-таблицы* воспринимаются как стадии выполнения задач и не могут служить изначальным статусом.
Отдельно от досок существует *Корзина* - список выгруженных задач.

Созданные *задачи* изначально попадают в *Корзину*. Присвоить *статус* можно либо при создании, либо после. Присвоение *статуса* осуществляется посредством вопросов и соответствующих кнопок согласно описанию методологии GTD

```
|--> Является ли это задачей?
|   |--> ДА  -> следующий вопрос
|   |--> НЕТ -> Удалить?
|              |--> ДА
|              |--> НЕТ -> остается в корзине
|
|--> Эта задача одношаговая?
|   |--> ДА  -> следующий вопрос
|   |--> НЕТ -> преобразование в проект
|
|--> Эта задача полностью зависит от внешних факторов?
|   |--> ДА  -> задача попадает в доску ожидания
|   |--> НЕТ -> следующий вопрос
|
|--> Эта задача неотложная?
|   |--> ДА  -> задача попадает в текущие задачи
|   |--> НЕТ -> задача попадает в отложенные задачи
```

В окне распределения присутствуют кнопки выбора *статус-таблицы* и *доски*. Проекты не могут находиться в *статус-таблицах:* Отложенные задачи, Текущие задачи и Ожидание. Следовательно, все пункты, содержащиеся в перечисленных досках, являются задачами. Существуют расширенные виды статус-таблиц, задач и проектов, в которых отображается более подробная информация.

Функционал задач

Окно создания задачи содержит:

- редактирование атрибутов;
- кнопку создания в корзину;
- кнопку создания;
- кнопку распределения в зависимости от вида задачи (проектная или самостоятельная).

Расширенный вид задач содержит:

- функции редактирования атрибутов;
- кнопку удаления;
- кнопку преобразования в проект;
- кнопка перемещения на другую доску.

Перенос задач по статус-таблицам осуществляется нажатием кнопки (->) или перетягиванием курсором. Нажатие кнопки (->) имеет следующий функционал:

- Если задача находится в Корзине, начинается ее распределение.
- Если задача находится в Отложенных задачах, она либо попадает в Текущие задачи, либо преобразуется в Проект.
- Если задача находится в Ожидании, она попадает в Выполнено.

- Если задача находится в Текущих задачах, она попадает в Выполнено или пользовательский этап выполнения, если таковой был создан.

Приоритет задачи влияет на ее расположение в доске по вертикали. Если у задачи нет срока, приоритет остается статичным. В противном случае, приоритет будет расти с приближением срока. Например: изначально остается месяц — приоритет 1, остается 2 недели — приоритет 3, остается неделя — приоритет 5, остается день — приоритет 10. Рост приоритета происходит процентно в зависимости от времени до срока и начинается с изначального приоритета. Если приоритет поднялся до 10.

Задачу можно в любой момент преобразовать в проект. Преобразование переносит условия в описание проекта, и появляется возможность создавать задачи, связанные с данным проектом.

Функционал проектов

Расширенный вид *проектов* содержит:

- список задач с отображением их статуса;
- функции редактирования атрибутов.

Связанные задачи попадают в *Текущие задачи*. По мере их выполнения шкала выполнения проекта будет расти.

Функционал статус-таблиц

Расширенный вид *статус-таблиц* содержит:

- список задач с отображением их приоритета и срока;
- функции редактирования атрибутов.

Для изменения порядка пользовательских *статус-таблиц* необходимо редактировать таблицу.

Функционал досок

Расширенный вид досок содержит:

- редактор дополнительных *статус-таблиц*;
- кнопку удаления доски;
- добавление пользователей для просмотра;
- функции редактирования атрибутов.

Функция поделиться просмотром:

Пользователи, не являющиеся владельцами доски, не могут просматривать ее содержание без разрешения создателя. В зависимости от выбранной роли у пользователя присутствует позможноть только просматривать или просматривать и редактировать задачи и проекты. Доступ к редактированию атрибутов доски имеет только создатель.

В настройках доски присутсвует функция добавления пользователей на просмотр по *имени пользователя*. У приглашенного пользователя, гостевая доска появляется в общем списке досок.

Функционал пользователя

- функции редактирования атрибутов.

2.4 Анализ аналогов

Список аналогов

1. Weeek - (https://weeek.net/)
2. Trello - (https://trello.com/)
3. Todoist - (https://todoist.com/)
4. Microsoft To Do - (https://to-do.office.com/tasks/)

Основные функции и возможности

Во всех перечисленных системах присутсвуют следующие функции:

- создание задач
- синхронизация с календарем
- система приоритетов
- сортировка задач
- мобильные версии
- интеграции с другими сервисами
- уведомления
- платные микросервисы
- групповые проекты

Weeek предоставляет бесплатный доступ к синхронизации с календарем, уведомления через социальную сеть Telegram, обширный бесплатный план и доступный платный план. Однако интерфейс является перегруженным и неотзывчивым как и в веб версии, так и в мобильном приложении.

Отличительными особенностями Weeek являются уведомления через социальную сеть Telegram и интеграция с популярными калернадрными сервисами.

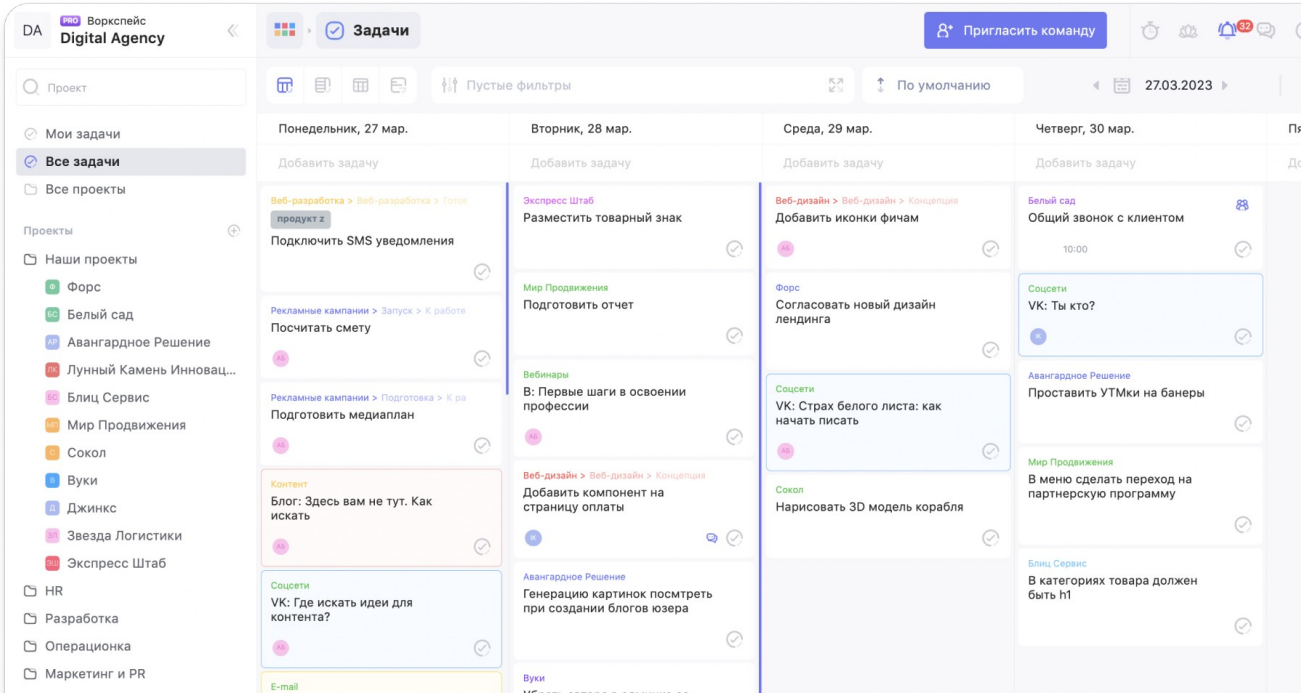

Рисунок 1 – интерфейс приложения Weeek

Trello отличается простотой и отзывчивостью интерфейса, гибкостью использования и обширным набором дополнительных функций платной версии. В то же время бесплатный план крайне ограничивает возможности пользователя: отсутсвие коллабораций, органичения в количестве проектов и интергрированных сервисов. Кроме того, сервис сильно зависит от интернет соединения и ограниченную систему уведомлений.

Основные отличительные черты Trello - это фукнции автоматизации действий и Power-Ups.

Автоматизация добивается встроенным инструментов Butler, в котором можно задач ряд определенных действий, срабатывающих по триггеру.

Power-Ups - это система пользовательских дополнений, интегрируемых с систему.

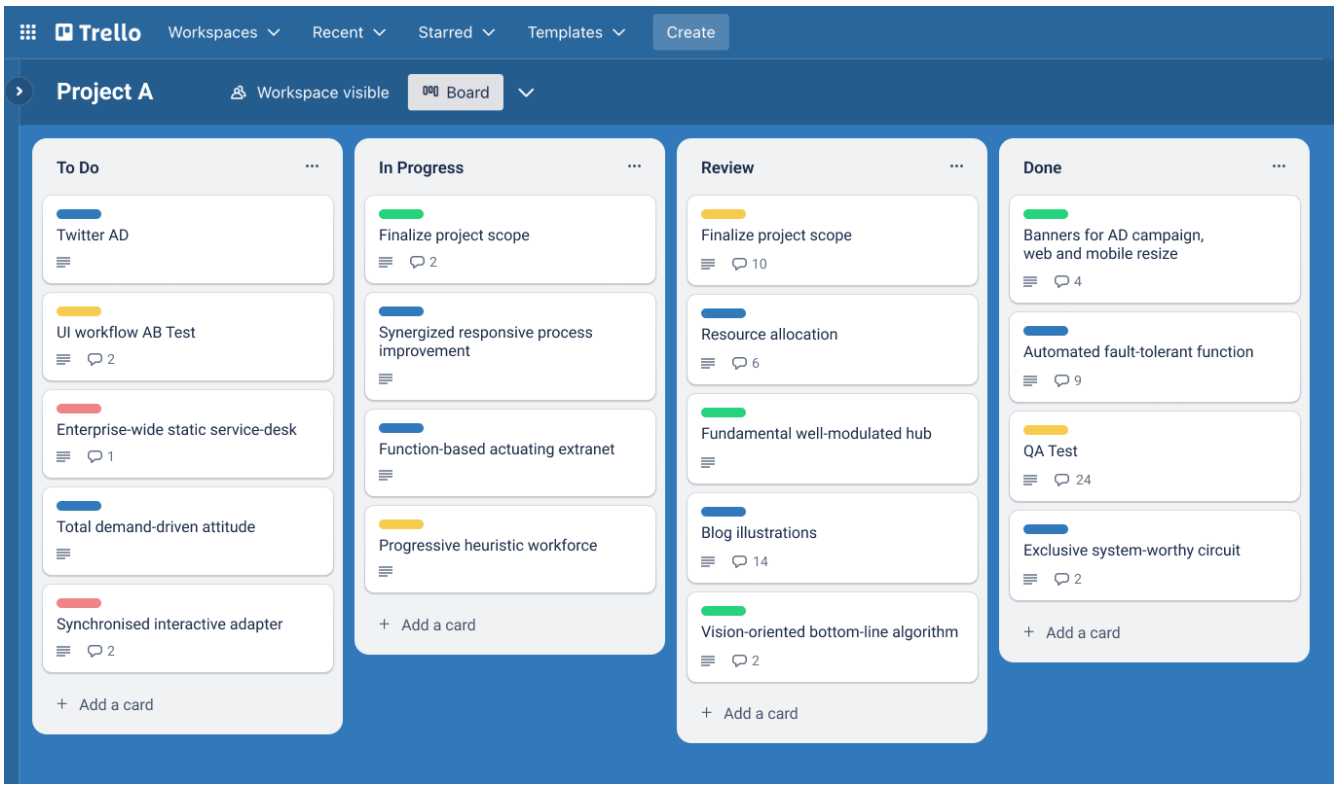

Рисунок 2 – интерфейс приложения Trello

Todoist имеет приятный отзывчивый интерфейс как в веб, так и в мобильной версии, а также множество полезных функций. Недостатками данной системы является то, что множество предлагаемых функций недоступны в бесплатной версии, отсутствие тайм-трекинга, и чрезмерное упрощение интерфейса.

Приложение предоставляет пользователю систему меток и фильтров. Метки используются для классификации задач по тегам, а фильтры позволяют выводить список задач по нескольким критериям: теги, дата и время, приоритет и д.р., например: "Сегодня & @работа".

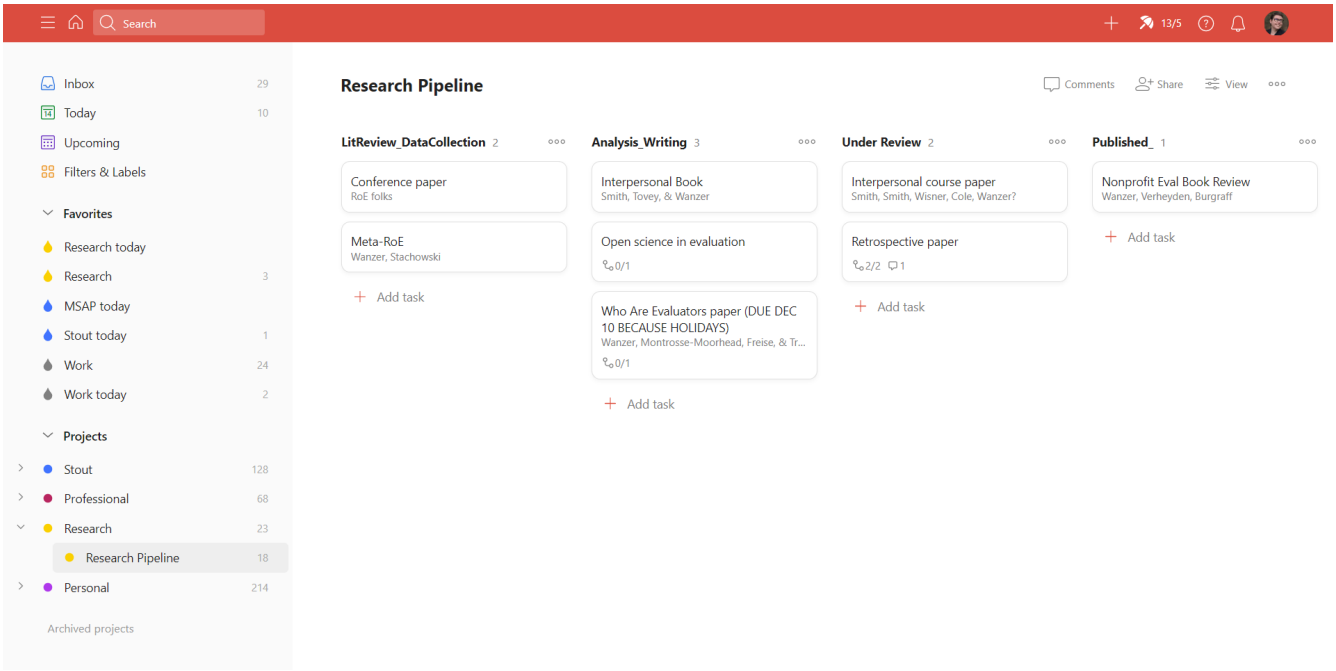

Рисунок 3 – интерфейс приложения Todoist

Microsoft To Do является простым менеджером задач с обширной интеграцией в приложениях Microsoft и облачной сихронизацией, что позволяет получить доступ к данным из любого приложения или устройства и отличается от аналогов отсутсвием платной версии. В то же время, по сравнению с другими аналогами у него крайне ограниченный функционал.

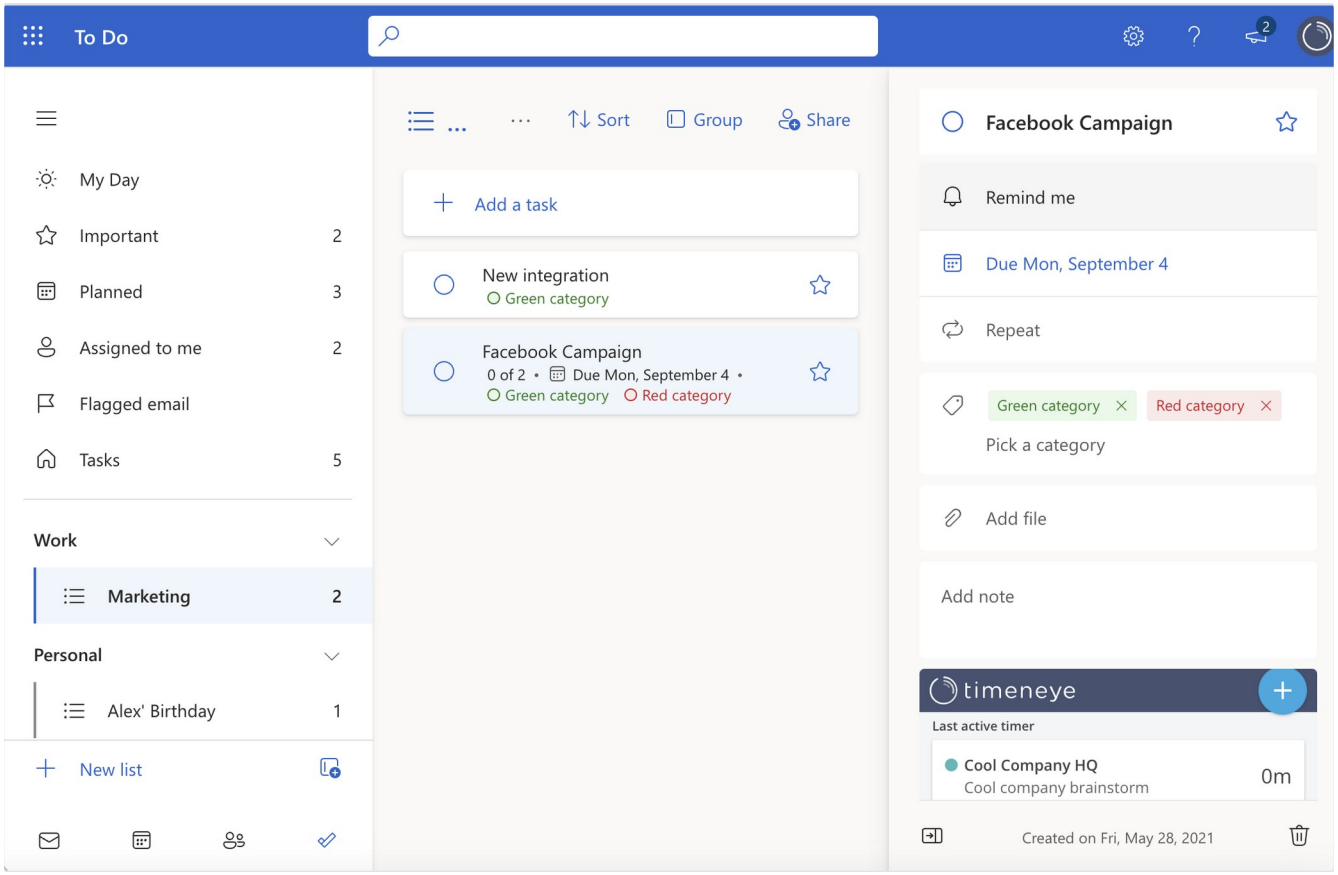

Рисунок 4 – интерфейс приложения Microsoft To Do

Особенность разрабатываемого веб-приложения

По сравнению с аналогами, основной особенностью разрабатываемого веб-приложения является автоматизация сортировки задач по метологии "Getting Things Done".

2.6 Обоснование выбора инструментов и платформы для разработки

Приложение основано на платформе Web ввиду следующих факторов:

- кроссплатформенность;
- доступность;
- низкая стоимость разработки;
- низкая стоимость поддержки и обновлений;
- маштабируемость.

Выбор используемых технологий

Клиентская часть:

Языки программирования: HTML5, CSS3, TypeScipt с библиотекой React. Библиотека React была выбрана ввиду возможности многовазового использования компонентов и инструментов работы с их жизненным циклом. TypeScript был выбран ввиду возможности четкой типизации в сравнении с JavaScript.

Инструмент сборки: npm.

Серверная часть:

Язык программирования Java с использованием фреймворка Spring.

Библиотеки:

- Spring boot для автоматической конфигурации фреймворка Spring, подключения внедренного Tomcat сервера и экосистемы Spring boot;
- Spring JPA для встраивания ORM - Hibernate;
- Spring Security и JJWT API для реализации авторизации пользователя по протоколу JWT.

Инструмент сборки: Maven

Spring boot был выбран ввиду высокой производительности и общирной экосистемы, позволяющей беспрепятственную интеграцию иснтрументов работы с СУБД, JWT и MVC.

СУБД:

PostgreSQL - реляционная СУБД с открытым исходным кодом, предназначенная для работы с большими объемами данных. Решение использовать PostgrSQL было принято ввиду высокой производительности, расширяемости, объемной документации и активной поддержке драйверов JDBC. Такие минусы, как сложности конфигурации и отсутствие встроенное пользовательского интерфейска были нивелированны использованием ORM. Минусы высоких требований к ресурсам при использовании ORM незначительны.

Инструменты разработки

Intellij IDEA Commutiny Edition - интерированная среда разработки прораммного обеспечения со встроенным функционалом для работы с языком программирования Java и инструментом сборки проетов Maven. Использован для разработки серверной части приложения.

Neovim - высоко персонализируемый текстовый редактор, основанный на Vim, с поддержской обширной системы пользовательских дополнений и интеграции с внешними инструментами. Tmux - терминальный мультиплексор, позволяющий управлять несколькими сессиями и окнами в одном окне терминальной оболочки (shell). Использованы для разработки клиетской части приложения.

В качестве вспомогательных технологических средств используется:

- git для управления версиями проекта;
- docker использован для контейнеризации сервера Postgres и запуска smpt сервера mail-dev в процессе разработки.

3. Проектно-конструкторская чать

3.1 Разработка архитектуры системы.

Для разработки данного приложения используется клиент-серверная архитектура.

Архитектура клиент-сервер – это расрпеделенная архитектура, в которой приложение разделяется на два основных компонента: клиент и сервер. Клиент – это часть приложения, выполняющаяся на стороне пользователя, а сервер – это чать, выполняющаяся на удаленной машине, предоставляющей какие-то услуга или ресурсы.

Клиент:

1. Интерфейс: клиентская часть обеспечаивает пользовательский интерфейс с которым взаимодействует пользовать. Это может быть глафический интерфейс, веб-страница, мобильное приложение и т.д.

2. Запросы к серверу: Клиент отправляет запросы на сервер для получения данных, обновлений или выполнения каких-то операций.

Взаимодействие между клиентом и сервером:

1. Установление соединения: клиаент устанавливает соединение с сервером используля сетевой протокол HTTP.

2. Отправка запроса: клиент отправляет запрос с данными серверу.

3. Обработка запроса на сервере.

4. Отправка ответа: сервер ответ с данными клиенту.

5. Обработка ответа на клиенте.

3.2 Наполнение базы данных

Для управления базой данных и хранения информации используется PostgreSQL.

Реляционная база данных – это структурированная коллекция данных, организованных в виде таблиц, где каждая таблица представляет отдельный тип данных, а отношения между таблицами устанавливаются посредством ключей. В такой базе данных информация хранится в виде записей, что обеспечивается эффективное управление данными и поддерживает стандартный язык запросов SQL для манипулирования информацией.

Для интеграции PostgreSQL с проектом используется библиотека фреймворка Spring “Spring JPA”, работающая на основе ORM (Object-Relational-Mapping) Hibernate. Схемы определяются классами Java с аннотациями @Entity.
Пример создания модели:

```
@Entity
@Table(name = "tasks")
public class Task {
    @Id
    @GeneratedValue
    private Long id;
    private String title;
    private String description;
    private LocalDate creationDate;
    private LocalDate deadline;
    private LocalDate completionDate;

    @ManyToOne
    @JoinColumn(name = "project_id", referencedColumnName = "id")
    private Project project;

    @ManyToOne
    @JoinColumn(name = "status_table_id", referencedColumnName =
"id")
    private StatusTable statusTable;
}
```

Взаимодействие с данными осуществляется с помощью интерфейсов, наследующихся от интерфейса Spring JPA “JpaRepostiry”.
Пример интерфейса:

```
public interface StatusTableRepository extends
JpaRepository<StatusTable, Long>  {
    @Query("""
            select st
            from StatusTable st
            where st.board.id = :boardId
            and st.name = 'Bucket'
            """)
    Optional<StatusTable> findBucketByBoardId(Long boardId);

    @Query("""
           select st
           from StatusTable st
           where st.board.id = :boardId
           """)
    List<StatusTable> findAllByBoardId(Long boardId);

    @Query("""
            select st
            from StatusTable st
            where st.board.id = :boardId
            and st.displayOrder = :displayOrder
            """)
    Optional<StatusTable> findByDisplayOrderAndBoardId(Integer
displayOrder, Long boardId);

    @Query("""
            select case when count(st) > 0 then true else false
end
            from StatusTable st
            where st.name = :name
            and st.board.id = :boardId
            """)
    boolean existsByNameAndBoardId(String name, Long boardId);

    @Query("""
            select st
            from StatusTable st
            where st.board.id = :boardId
            and st.name = 'Await'
            """)
    Optional<StatusTable> findAwaitByBoardId(Long boardId);

    @Query("""
            select st
            from StatusTable st
            where st.board.id = :boardId
            and st.name = 'Delayed'
            """)
    Optional<StatusTable> findDelayedByBoardId(Long boardId);

    @Query("""
            select st
            from StatusTable st
            where st.board.id = :boardId
            and st.name = 'Current'
            """)
    Optional<StatusTable> findCurrentByBoardId(Long boardId);
```

```
    @Query("""
            select st
            from StatusTable st
            where st.board.id = :boardId
            and st.displayOrder = 4
            """)
    Optional<StatusTable> findFirstCompletionStatus(Long boardId);
}
```

Истекшие данные удаляются в течение интервала времени, установленного в конфигурационном файле application.yaml. Пароли пользователей хешируются с помощью одностороннего метода хеширования при поступлении данных.

3.3 Интеграция Json Web Token

JSON Web Token (JWT) — это стандарт открытого формата (RFC 7519) для создания токенов, которые представляют собой закодированные структуры данных, используемые для передачи информации между сторонами.

Токены представляют собой зашиброванную строку, участки которой разделяются на Заголовок (используемый алгоритм шифровки и тип), Полезную нагрузку (надоб утверждений claim, в которых хранится информация) и Подпись (обеспечивается целостность данных от санкционированного изменения.

Для интеграции JWT была использована библиотека “io.jsonwebtoken”.

Пример использования методов библиотеки:

```
@Service
@RequiredArgsConstructor
public class JwtTokenService {
    private final UserRepository userRepository;
    private final RefreshTokenRepository refreshTokenRepository;
    @Value("${application.security.jwt.access-expiration}")
    private long accessTokenExpiration;
    @Value("${application.security.jwt.refresh-expiration}")
    private long refreshTokenExpiration;
    @Value("${application.security.jwt.secret-key}")
    private String secret;

    public String generateAccessToken(User user) {
        return String.format(buildToken(new HashMap<>(), user,
accessTokenExpiration));
```

```
    }

    public String generateRefreshToken(User user) {
        String sessionId = UUID.randomUUID().toString();
        String token = buildToken(Map.of("sessionId", sessionId),
user, refreshTokenExpiration);
        RefreshToken rsToken = RefreshToken.builder()
                .token(token)
                .issuedAt(new Date(System.currentTimeMillis()))
                .expiresAt(new Date(System.currentTimeMillis() +
refreshTokenExpiration))
                .revoked(false)
                .user(user)
                .sessionId(sessionId)
                .build();
        refreshTokenRepository.save(rsToken);
        return token;
    }

    private String buildToken(Map<String, Object> claims,
UserDetails userDetails, long expiration) {
        var authorities = userDetails.getAuthorities();
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(userDetails.getUsername())
                .claim("authorities", authorities)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis()
+ expiration))
                .signWith(getSignInKey())
                .compact();
    }

    public boolean isTokenValid(String token, UserDetails
userDetails) {
        final String username = extractUsername(token);
        return username.equals(userDetails.getUsername()) && !
isTokenExpired(token);
    }

    public boolean validateRefreshToken(String token) {
        RefreshToken rsToken =
refreshTokenRepository.findByToken(token)
                .orElseThrow(() -> new RecordNotFound("Refresh
Token not found"));
        boolean isExpired = rsToken.getExpiresAt().before(new
Date(System.currentTimeMillis())) ;
        return isExpired && !rsToken.isRevoked();
    }

    private boolean isTokenExpired(String token) {
        return extractClaim(token,
Claims::getExpiration).after(new
Date(System.currentTimeMillis()));
    }

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
```

```
    }


    private <T>T extractClaim(String token, Function<Claims, T>
claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
        return Jwts.parserBuilder()
                .setSigningKey(getSignInKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private Key getSignInKey() {
        byte[] keyBytes = Decoders.BASE64.decode(secret);
        return Keys.hmacShaKeyFor(keyBytes);
    }

    public void invalidateAllRefreshTokens(User user) {
        List<RefreshToken> tokens =
refreshTokenRepository.findTokensByUserId(user.getId());
        refreshTokenRepository.deleteAll(tokens);
    }

    public void invalidateRefreshToken(String refreshToken) {
        refreshTokenRepository.deleteByToken(refreshToken);
    }
}
```

Для реализации авторизации была использована механизм парных ключей. Refresh Token – токен, отправляемый пользователю через Cookies, истекающий за продолжительный период времени (недели, месяцы). Access Token – токен, отправляемый пользователю в качестве HTTP зоголовка Authorization, истекающий в течение нескольких минут. При авторизации пользователю отправляется оба токена. К каждому запросу серверу прикрепляется Access Token для идентификации пользователя. Когда Access Token истекает, отправляется запрос на обновление пары токенов, содержащий Refresh Token в Cookies. Если Refresh Token истекает, система запрашивает повторную авторизацию.

Пример сущности Refresh Token:

```
public class RefreshToken {
    @Id
    @GeneratedValue
    private Long id;
    @Column(length = 512, unique = true, nullable = false)
    private String token;
    private Date issuedAt;
    private Date expiresAt;
    private String sessionId;
    private boolean revoked;

    @ManyToOne
    @JoinColumn(name = "user", referencedColumnName = "id")
    private User user;
}
```

Проверка на наличие Access Token в заголовках запроса производится посредством использования библиотеки Spring Security.

Пример:

```
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http)
throws Exception {
        http
                .cors(Customizer.withDefaults())
                .csrf(CsrfConfigurer::disable)
                .authorizeHttpRequests(request ->
                        request.requestMatchers(
                                        "/auth/**"
                                ).permitAll()
                                .anyRequest()
                                .authenticated()
                ).sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authenticationProvider(authenticationProvider)
                .addFilterBefore(jwtFilter,
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
```

Каждый получаемый сервером запрос проходит через ряд проверок с применением фильтра JWTFilter.

Пример JWTFilter:

```
public class JwtFilter extends OncePerRequestFilter {
    private final JwtTokenService jwtTokenService;
    private final UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(
            @NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response,
```

```
            @NonNull FilterChain filterChain
    ) throws ServletException, IOException {
        if (request.getContextPath().contains("/api/auth")) {
            filterChain.doFilter(request, response);
            return;
        }

        final String authHeaders =
request.getHeader(HttpHeaders.AUTHORIZATION);
        final String accessToken;
        final String email;

        if (authHeaders == null || !
authHeaders.startsWith("Bearer")) {
            filterChain.doFilter(request, response);
            return;
        }
        accessToken = authHeaders.substring(7);
        try {

            email = jwtTokenService.extractUsername(accessToken);
            if (email == null ||
SecurityContextHolder.getContext().getAuthentication() != null) {
                throw new AuthenticationException("Jwt token
invalid");
            }

            UserDetails userDetails =
userDetailsService.loadUserByUsername(email);
            UsernamePasswordAuthenticationToken
authenticationToken =
                    new UsernamePasswordAuthenticationToken(
                            userDetails, null,
userDetails.getAuthorities()
                    );

SecurityContextHolder.getContext().setAuthentication(authenticatio
nToken);
            filterChain.doFilter(request, response);
        } catch (AuthenticationException | ExpiredJwtException e)
{
            response.setContentType("application/json");
            response.setStatus(HttpStatus.UNAUTHORIZED.value());
            response.getWriter().write(
                    new ObjectMapper().writeValueAsString(
                            ExceptionDto.builder()
                                    .error(e.getMessage())
                                    .build()
                    )
            );
        }
    }
}
```

Фильтр проверяет на наличие токена и его действительность, а также извлекает данные о пользователе.

3.4 Интеграция SMTP протокола

Для реализации функции подтвердения аккаунта пользователя через почту, была использована библиотека “spring-boot-starter-mail”, которая реализует отправку электронных писем с помощью SMTP протокола.

Пример класса по отправке писем:

```
public class EmailService {
    private final JavaMailSender mailSender;
    private final SpringTemplateEngine templateEngine;
    private final ActivationTokenService tokenService;
    @Value("${application.mailing.sender-email}")
    private String senderEmail;
    @Value("${application.mailing.activation-url}")
    private String activationUrl;

    public void sendValidationEmail(User user) throws
MessagingException {
        String token = tokenService.generateActivationToken();
        String url = tokenService.generateActivationUrl();
        tokenService.saveActivationToken(token, url, user);
        sendEmail(user.getEmail(), user.getUsername(),
EmailTemplateName.ACTIVATE_ACCOUNT, token, url, "Account
activation");
    }

    @Async
    private void sendEmail(String receiver,
                           String username,
                           EmailTemplateName emailTemplate,
                           String token,
                           String url,
                           String subject
    ) throws MessagingException {
        String templateName = emailTemplate == null ? "confirm-
email" : emailTemplate.name();
        MimeMessage mimeMessage = mailSender.createMimeMessage();
        MimeMessageHelper mimeMessageHelper;

        mimeMessageHelper = new MimeMessageHelper(
                mimeMessage,
                MimeMessageHelper.MULTIPART_MODE_MIXED,
                StandardCharsets.UTF_8.name()
        );
        Map<String, Object> properties = new HashMap<>();
        properties.put("username", username);
        properties.put("confirmationUrl", activationUrl + url);
        properties.put("activationCode", token);
```

```
        Context context = new Context();
        context.setVariables(properties);

        mimeMessageHelper.setSubject(subject);
        mimeMessageHelper.setTo(receiver);
        mimeMessageHelper.setFrom(senderEmail);
        String template = templateEngine.process(templateName,
context);
        mimeMessageHelper.setText(template, true);
        mailSender.send(mimeMessage);
    }

}
```

После успешной регистрации запускается система подтверждения акаунта, которая генерирует случайное шестизначиное число и ссылку, которые отправляется на почту пользователя.

3.5 Бизнес логика

Пример обработки запроса на создание доски.

```
@PostMapping("/create")
ResponseEntity<BoardCreateResponse> create(
     @Valid @RequestBody BoardCreateRequest request,
     Authentication authentication
) {
     user = (User) authentication.getPrincipal();
     ResponseEntity.status(HttpStatus.CREATED)
     body(
          service.create(request, user));
}
```

Сервер принимает запрос на создание доски с данными: название, список дополнительных таблиц и список добавленных пользователей. Система извлекает пользователя с помощью JWTFilter и передает данные с пользователем в слой бизнес логики.

Пример слоя бизнес логики:

```
    @Transactional
     public BoardCreateResponse create(BoardCreateRequest request,
User authenticatedUser) {
        if
(repository.boardNameExistsByUserId(authenticatedUser.getId(),
request.getName())) {
```

```
            throw new IllegalRequestContentException("board with
this name already exists");
        }
        Board board = Board.builder()
                .name(request.getName())
                .build();

        repository.save(board);

boardPermissionUtil.grantCreatorPermissions(authenticatedUser,
board);
        if (request.getUserList() == null) {
            return boardMapper.mapToBoardCreateResponse(board);
        }
        for (UserPermissionDto dto : request.getUserList()) {
            User user =
userRepository.findUserByUsername(dto.getUsername()).orElseThrow((
) -> new RecordNotFound("User not found"));
            if (dto.getPermissionLevel() == 1) {
                boardPermissionUtil.grantViewerPermissions(user,
board);
            }
            if (dto.getPermissionLevel() == 2) {
                boardPermissionUtil.grantEditorPermissions(user,
board);
            }
        }
        statusTableUtil.createStatusTables(board,
request.getAdditionalStatusList());

        return boardMapper.mapToBoardCreateResponse(board);
    }
```

Система проверяет наличие доски с таким же названием у пользователя. Создает новую доску, создает таблицы и выдает права пользователям на доступ к создаваемой доске. После чего, из данных новой доски собирается объет ответа и отправляется на клиент.

4. Проектно-технологическая часть

4.1 Порядок развертывания системы

Развертывание системы состоит из следующих этапов:

1. Запуск docker контейнера postgres.
2. Запуск docker контейнера mail-dev.
3. Запуск серверной части:
    1. mvn install
    2. mvn spring-boot:run
4. Запуск клиентской части:
    1. npm install
    2. npm start

4.2 Тестирование веб-приложения.

Данная система тестировалась в браузере Mozi\\a Firefox версии 133.0.3 с помощью иструментов разработчика с дополнением React Developer Tools.

Порядок тестирования:

1. Проверка запросов.
    1. Коррестность URL-эндпоинтов.
    2. Передача параметров, объектов и заголовков.
2. Проверка ответов сервера:
    1. Коррестность формата данных JSON.
    2. Проверка содержимого ответов.
    3. Коррестность формата ошибок.
    4. Коррестность передачи заголовков и Cookies.
3. Проверка функциональность приложения.
    1. Аутентификация и авторизация.
    2. Формирование, валидация и отправка форм.

Результаты тестирования:

1. Все тестируемые эндпоинты корректно обрабатывали и возвращали ожидаемые данные.
2. Неверно сформированные запросы приводили к соответствующим сообщениями об ошибках.
3. Основные функциональные модули приложения прошли проверку без ошибок.

4.2 Разработки руководства пользователя

При загрузке сайта пользователь попадает на страницу приветствия с возможностью перейти на страницу авторизации или регистрации. (Рисунок 5)

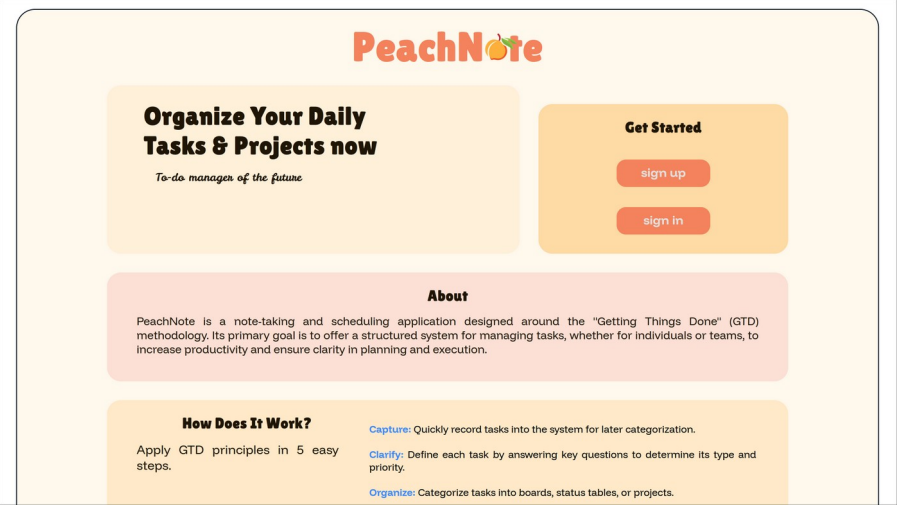

Рисунок 5 – страница приветствия.

Если пользователь не был зарегистрирован, он должен заполнить соответствующую форму и подтвердить свою почту с помощью кода подтверждения или перейдя по ссылке. При авторизации в систему пользователю пользователю необходимо заполнить соответствующую форму. (Рисунок 6).

Рисунок 6 – страница авторизации и регистрации

После авторизации пользоватею доступна главная страница /dashboard и страница профиля /profile.

На главной странице в боковом меню пользователя может создать первую доску по нажанию на кнопку в нижней части меню. (Рисунок 7)

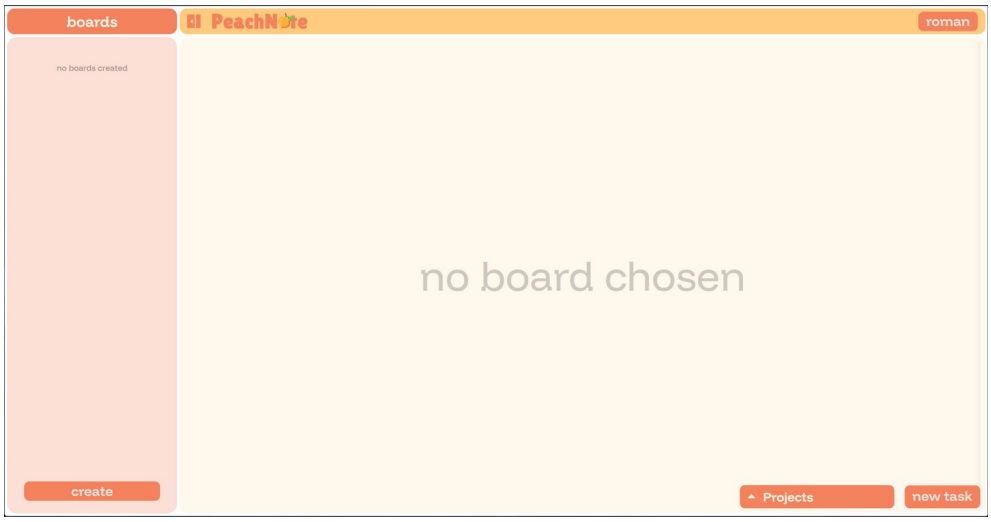

Рисунок 7 – главная страница /dashboard

В окне нажатия на кнопку открывается окно создания. (Рисунок 8)

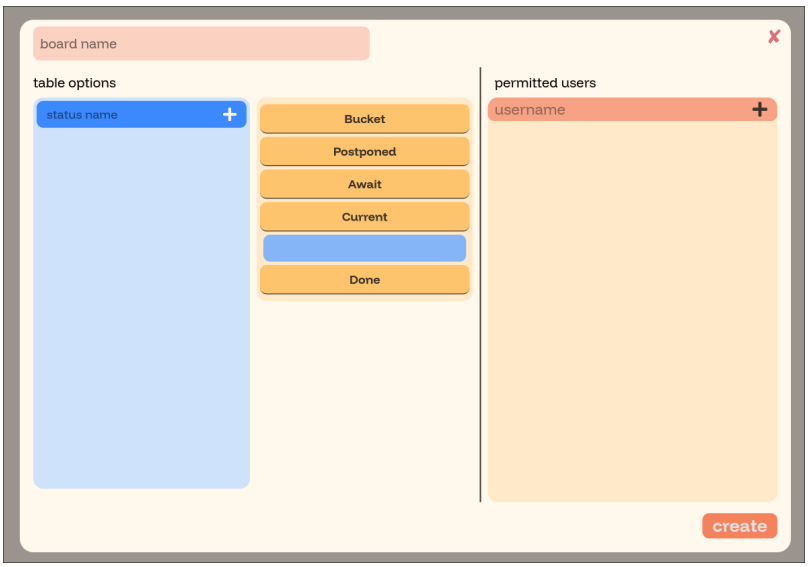

Рисунок 8 – окно создания доски.

Поля проверяются как на клиентской, так и на сервверной части приложения. В противном случае отображается ошибка.

Создав доску, пользователь может заполнять ее задачами. (Рисунок 9)

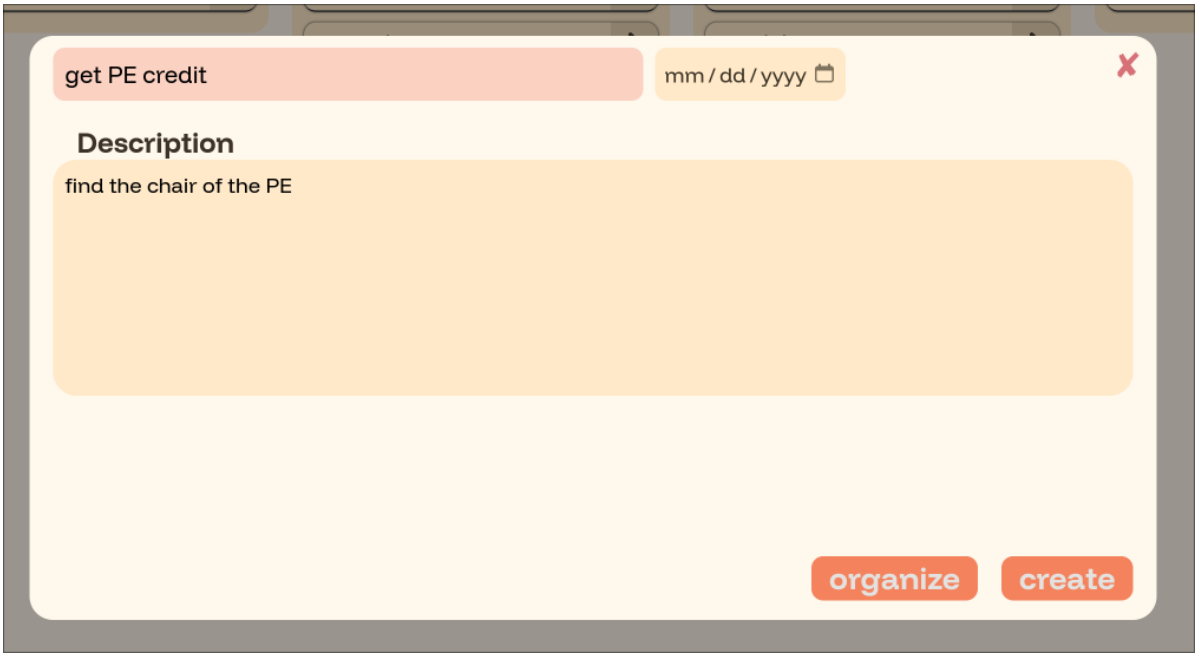

Рисунок 9 – окно создания задачи.

При нажатии на кнопку сортировки задачи открывается окно для присвоения статуса задачи. (Рисунок 10)

Рисунок 10 – окно сортировки задач

Нажав на кнопку преобразования задачи в проект, новосозданный проект помещается в список проектов, который можно открыть для редактирования и управления. (Рисунок 11)

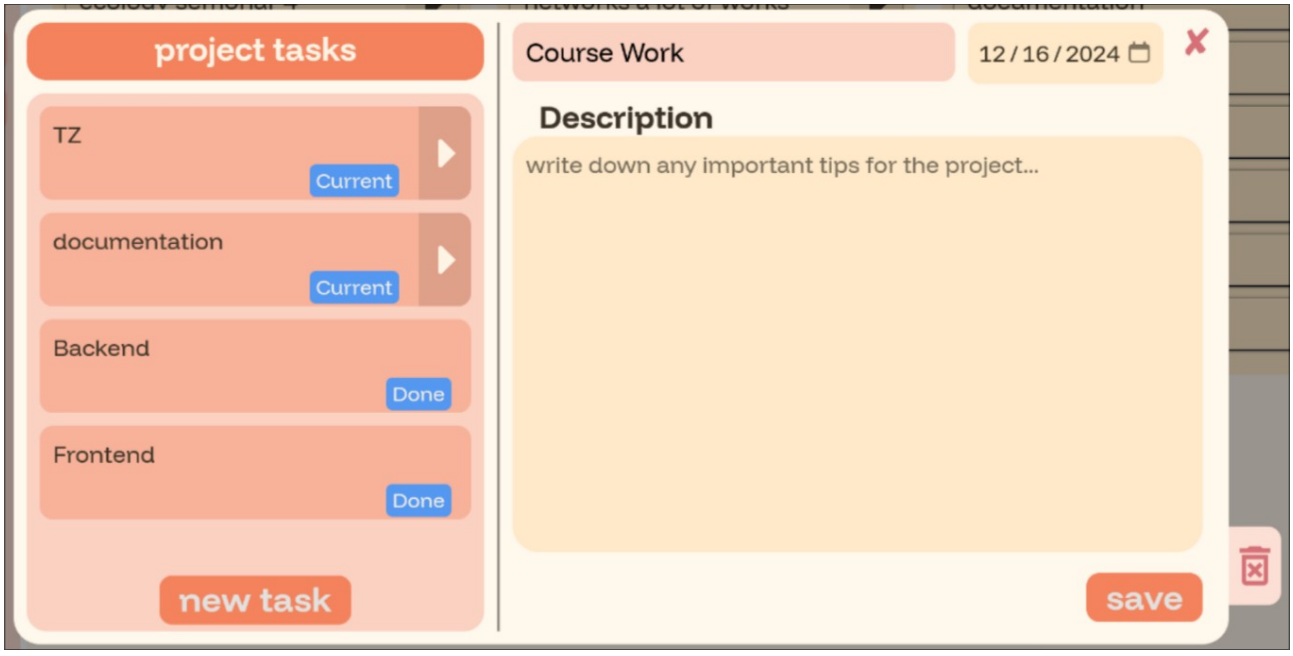

Рисунок 11 – окно редактирования проекта.

Нажав кнопку создания задачи из окна проекта, задачи автоматически свяжется с проектом, как подзадача.

Также, существуют окна редактирования задач и досок, с такими же по функционалу полями, как при их создании. (Рисунок 12 и Рисунок 13)

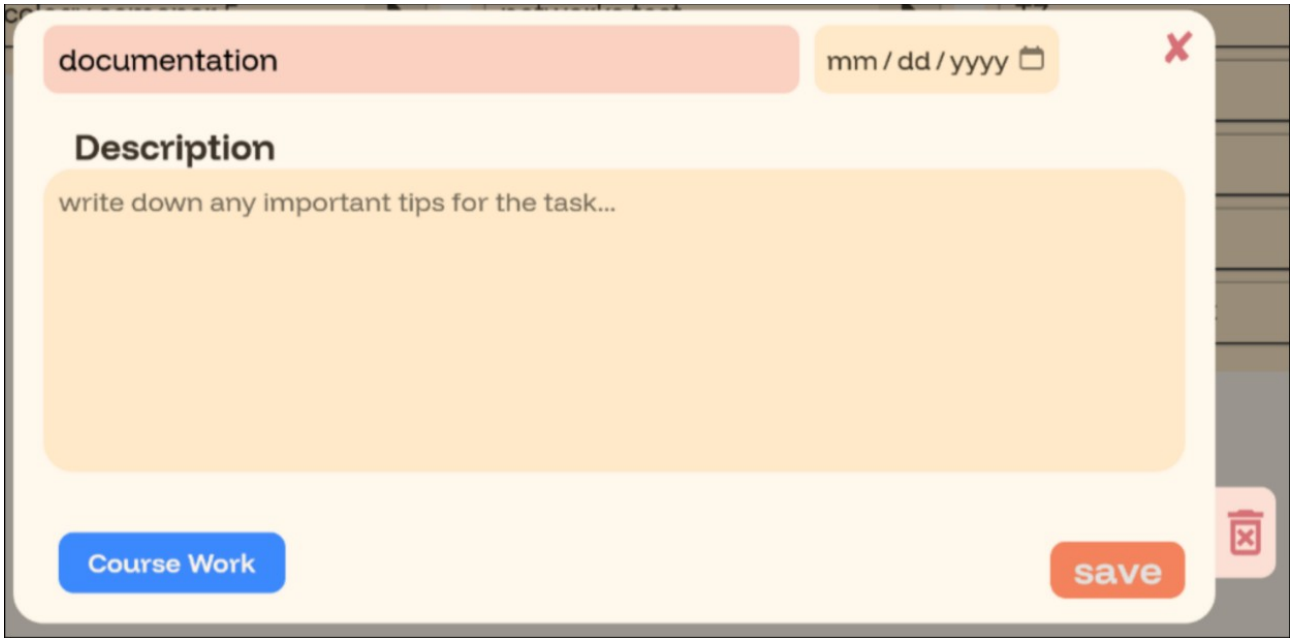

Рисунок 12 – окно редактирования задачи.

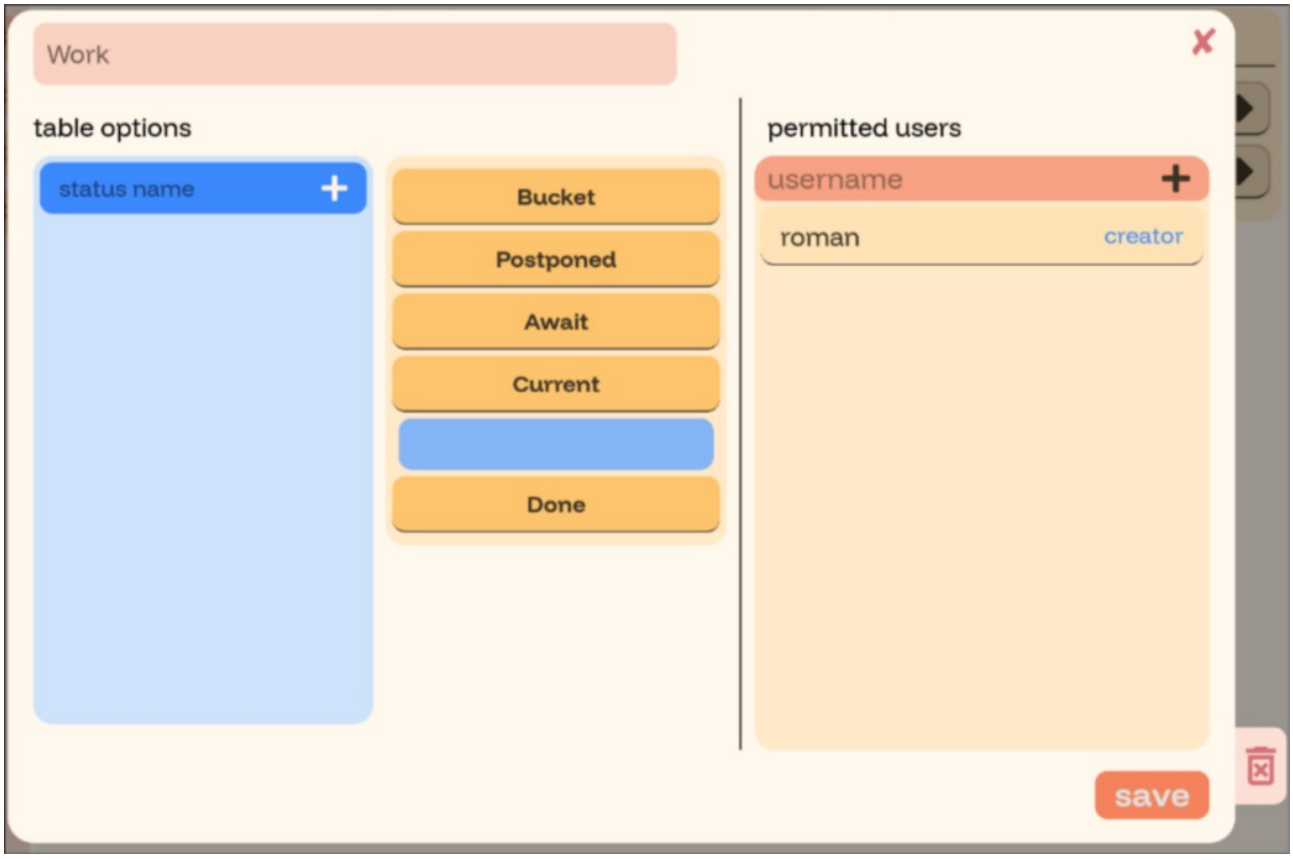

Рисунок 13 – окно редактирования досок.

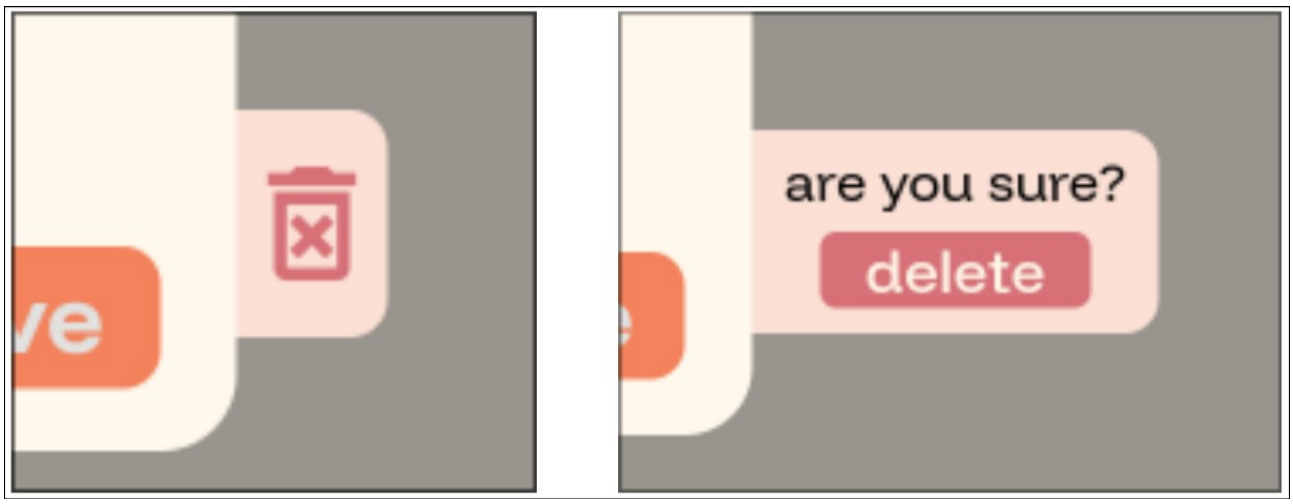

Рисунок 14 – кнопка удаления сущностей.

Так же пользователь может изменить свои данные на странице профиля, перейдя в него по кнопке навигационной панели справа. (Рисунок 10)

Рисунок 10 – профиль пользователя.

Заключение

При выполнении курсовой работы на тему “Система заметок GTD” была исследована и описана предметная область, проведен анализ аналогов данной системы, а также выбор инструментов и платформы для разработки.

Проведен анализ обектов автоматизации и разработаны методы решения технических задач. Также была разработана и реализована структура базы данных, серверная часть приложения и веб-интерфейс системы. База данных была наполнена тестовыми данных.

Результатом данной курсовой работы является рабочее веб-приложение.

Список использованных источников

1. Сакулин, С. А. Основы интернет-технологий: HTML, CSS, JavaScript, XML: учебное пособие / С. А. Сакулин. Москва: МГТУ им. Н.Э. Баумана, 2017. - 112 с.

2. Проектирование объектов баз данных В среде Access: учебное пособие / Брешенков А. В., Губарь А. М.; МГТУ им. Н. Э. Баумана.- М.: Изд-во МГТУ им. Н. Э. Баумана, 2006.- 180 с.

3. Введение в пректирование баз данных: учебное пособие для вузов / Лукин В. Н. - М. : Вузовская книга, 2013. Библиогр.: с. 140.

4. Walls, C. Spring Boot in Action. Manning, 2018. - 320 с.

5. Ревунков, Г.И. Проектирование баз данных Баумана, 2009. - 20 с.