**СОДЕРЖАНИЕ**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

УО «ВГТУ» ПП.008 1-40 05 01-01 ПЗ

Разраб.

Провер.

Реценз.

Н. Контр.

Утверд.

Разработка системы учета рабочего времени

Лит.

Листов

[Введение 3](#_Toc131277614)

[1 Анализ объекта 5](#_Toc131277615)

[1.1 Описание предметной области 5](#_Toc131277616)

[1.2 Построение концептуальной модели предметной области 7](#_Toc131277617)

[2 Постановка задачи 11](#_Toc131277618)

[2.1 Определение требований к программной системе. 11](#_Toc131277619)

[2.2 Описание аналогов системы 14](#_Toc131277620)

[2.3 Обзор и обоснование выбора инструментальных средств 20](#_Toc131277621)

[3 Проектирование 26](#_Toc131277622)

[3.1 Разработка архитектуры программного продукта 26](#_Toc131277623)

[3.2 Проектирование структур хранения данных 30](#_Toc131277624)

[3.3 Описание реализации вариантов использования 34](#_Toc131277625)

[4 Реализация 37](#_Toc131277626)

[4.1 Разработка классов информационной системы 37](#_Toc131277627)

[4.2 Разработка интерфейса программного продукта. 37](#_Toc131277628)

[4.3 Разработка алгоритмов реализации вариантов использования 51](#_Toc131277629)

[4.3 Модульное тестирование алгоритмов реализации вариантов использования 55](#_Toc131277630)

[5 Системное тестирование 60](#_Toc131277631)

[5.1 Функциональное тестирование 61](#_Toc131277632)

[5.2 Оценка безопасности 62](#_Toc131277633)

[6 Экономическая часть 66](#_Toc131277634)

[6.1 Расчет трудоемкости разработки программного обеспечения 66](#_Toc131277635)

[6.2 Определение стоимости разработки программного обеспечения 66](#_Toc131277636)

[6.3 Расчёт экономического эффекта от внедрения программного продукта 76](#_Toc131277637)

[7 Охрана труда 79](#_Toc131277638)

[8 Промышленная экология 93](#_Toc131277639)

[9 Ресурсосбережение 97](#_Toc131277640)

[Заключение 101](#_Toc131277641)

[Список использованной литературы 103](#_Toc131277642)

# Введение

В современном мире роль информационных технологий в управлении ресурсами и рабочим процессом компаний продолжает увеличиваться. В связи с этим возрастает потребность в системах, которые способны организовать и контролировать рабочий процесс, позволяя руководству и сотрудникам эффективно управлять своим временем и ресурсами. Одной из таких систем является система учета рабочего времени.

Система учета рабочего времени предоставляет возможность руководителям выдавать задания и отслеживать ход их выполнения, а исполнителям – вести учет времени, затраченного на выполнение каждого задания. Это позволяет улучшить планирование проектов, оптимизировать рабочий процесс и повысить эффективность команды.

Целью данной дипломной работы является разработка системы учета рабочего времени, которая будет удовлетворять потребностям различных организаций и предоставлять возможность для гибкой настройки и интеграции с существующими бизнес-процессами. Для достижения этой цели необходимо выполнить следующие задачи:

1. Изучить существующие методы учета рабочего времени и подходы к управлению проектами.
2. Определить основные функциональные требования к системе учета рабочего времени.
3. Разработать архитектуру и модели данных системы.
4. Реализовать систему с использованием современных технологий и инструментов разработки.
5. Протестировать и оценить эффективность разработанной системы.

В ходе выполнения дипломной работы были изучены и использованы различные технологии и инструменты разработки, такие как Java, Spring Framework, Thymeleaf, PostgreSQL и Docker. Были разработаны модели данных, контроллеры и представления, а также интеграция с базой данных и аутентификацией пользователей.

Данная работа представляет собой полноценную систему учета рабочего времени, которую можно использовать для оптимизации рабочего процесса и управления проектами в организациях различного типа и размера.

# 1 Анализ объекта

## 1.1 Описание предметной области

Система учета рабочего времени представляет собой комплексное программное решение, которое позволяет управлять, контролировать и анализировать рабочее время сотрудников [1]. Данные системы широко используются в различных организациях, таких как корпорации, малый и средний бизнес, государственные учреждения, некоммерческие организации и т.д. В данной главе будут рассмотрены основные аспекты систем учета рабочего времени, их применение, существующие методы учета времени и подходы к управлению проектами.

1. Системы учета рабочего времени

Система учета рабочего времени предназначена для автоматизации процесса учета времени, которое сотрудники тратят на выполнение своих рабочих обязанностей. Она позволяет руководителям детально отслеживать ход выполнения задач, а сотрудникам – вести учет затраченного времени. Важными особенностями систем учета рабочего времени являются возможность сбора статистических данных о рабочем времени, анализ эффективности работы сотрудников, а также контроль соблюдения трудового законодательства и рабочих норм.

1. Применение систем учета рабочего времени

Системы учета рабочего времени активно применяются в различных сферах деятельности, таких как производство, информационные технологии, финансы, образование, медицина и многих других. Они помогают руководителям эффективно планировать и контролировать рабочий процесс, оптимизировать использование рабочего времени сотрудников, снижать риск переработок и выгорания персонала. Кроме того, системы учета рабочего времени обеспечивают прозрачность и контроль над выполнением задач, что способствует повышению мотивации и дисциплины сотрудников.

1. Методы учета рабочего времени

Существуют различные методы учета рабочего времени, которые могут быть применены в зависимости от особенностей организации и ее требований. Некоторые из наиболее распространенных методов включают:

3.1. Традиционный учет рабочего времени

Традиционный учет рабочего времени предполагает использование журналов, таблиц или карточек, в которых сотрудники отмечают свое рабочее время. Этот метод обладает низкой степенью автоматизации, требует большого времени на обработку данных и может привести к ошибкам из-за человеческого фактора.

3.2. Электронный учет рабочего времени

Электронный учет рабочего времени предполагает использование специализированных программных средств, которые позволяют автоматически фиксировать время начала и окончания работы сотрудников. Этот метод обеспечивает более высокую точность и надежность данных, а также возможность быстрого анализа информации о рабочем времени.

3.3. Учет рабочего времени с использованием мобильных приложений

Мобильные приложения для учета рабочего времени позволяют сотрудникам регистрировать свое рабочее время с помощью смартфонов или планшетов. Это обеспечивает удобство и гибкость, а также возможность отслеживать рабочее время в режиме реального времени.

1. Подходы к управлению проектами

Управление проектами является важным аспектом в организации работы с системами учета рабочего времени. Существует несколько подходов к управлению проектами, которые могут быть применены в зависимости от специфики организации и задач, стоящих перед ней.

4.1. Водопадный подход

Водопадный подход к управлению проектами предполагает строгую последовательность выполнения этапов проекта. Этот подход подразумевает детальное планирование всех этапов проекта и предполагает низкую степень гибкости в случае изменения требований или условий выполнения проекта.

4.2. Гибкие методологии

Гибкие методологии управления проектами, такие как Agile, Scrum или Kanban, предполагают гибкость и адаптивность в управлении проектами. Они позволяют быстро реагировать на изменения требований и условий выполнения проекта, а также эффективно распределять ресурсы и управлять рабочим временем сотруд ников. Гибкие методологии фокусируются на непрерывной коммуникации между членами команды и включают регулярные совещания, на которых обсуждаются прогресс проекта и возможные корректировки.

4.3. Гибридные подходы

Гибридные подходы к управлению проектами сочетают элементы водопадного и гибкого подходов, позволяя достичь оптимального баланса между структурой и гибкостью. Этот подход подходит для организаций, которые хотят сохранить определенный уровень структурированности, но при этом иметь возможность быстро адаптироваться к изменяющимся условиям.

1. Интеграция систем учета рабочего времени с управлением проектами

Интеграция систем учета рабочего времени с управлением проектами является важным шагом для повышения эффективности работы организации. Это позволяет лучше планировать и контролировать рабочий процесс, а также анализировать затраты времени на различные этапы проекта. В результате, руководители могут оптимизировать использование ресурсов, повышать эффективность работы команды и улучшать качество выполнения проектов.

## 1.2 Построение концептуальной модели предметной области

Включают широкий спектр символических моделей [2]:

* модель жизненного цикла систем, описывающую процессы существования системы от зарождения замысла до прекращения функционирования;
* модели операций, выполняемых объектом и представляющих описание взаимосвязанной совокупности процессов функционирования отдельных элементов объекта при реализации тех или иных функций объекта (в их состав могут входить модели надежности, характеризующие выход элементов системы из строя под влиянием эксплуатационных факторов, и модели живучести, характеризующие выход элементов системы из строя под влиянием целенаправленного воздействия внешней среды);
* информационные модели, отображающие во взаимосвязи источники и потребители информации, виды информации, характер ее преобразования, а также временные и количественные характеристики данных;
* процедурные модели, описывающие порядок взаимодействия элементов исследуемого объекта при выполнении различных операций (например, обработка материалов, деятельность персонала, использование информации, в том числе и различные процедуры принятия управленческих решений);
* временные модели, описывающие процедуру функционирования объекта во времени распределения ресурса «время» по отдельным компонентам объекта.

Модель обычно является комбинацией физических, математических и гибридных элементов либо целиком реализована в виде компьютерной имитации.

На практике для прогнозирования ожидаемого поведения сложной системы в целом обычно нецелесообразно строить модель функционирования, способную отразить все детали поведения системы. Поэтому при испытаниях системы в целом ее наблюдаемое поведение обычно анализируется на двух уровнях. Первый уровень относится к сквозным показателям функционирования, определенным в требованиях к системе. Второй - к тем подсистемам и компонентам, которые характеризуются каким-то критическим поведением. Последний уровень особенно важен, когда сквозное испытание не дает ожидаемого результата и требуется найти источник отклонений.

Решение о том, насколько детальной должна быть модель, применяемая для испытаний на уровне системы, является главным образом функцией системной инженерии, поскольку требуется искать компромисс между риском, сопутствующим игнорированию включения в модель некоторых функций, и ценой их включения.

Поскольку испытать все невозможно, при выборе того, что подвергать испытаниям в первую очередь (а значит, и того, что должна прогнозировать модель), необходимо принимать во внимание результаты анализа относительных рисков игнорирования некоторых характеристик.

Проектирование, построение и валидация моделей функционирования системы - сама по себе трудная задача, при решении которой должны применяться те же системно-инженерные методы, что и при разработке самой системы.

Но в то же время нужно стремиться ограничить затраты на моделирование экономически целесообразной долей от всех затрат на разработку системы. Соблюдение баланса между реалистичностью и стоимостью моделирования - одна из самых трудных задач системной инженерии.

При построении концептуальной модели предметной области была сформирована функциональная структура программного средства учета рабочего времени и управления проектами. Она состоит из следующих подсистем:

1. Подсистема работы с пользователями. Предназначена для добавления, удаления и редактирования информации о пользователях, а также для управления их ролями в рамках проектов.
2. Подсистема управления проектами. Позволяет создавать, редактировать и удалять проекты, а также управлять их составом, включая членов проекта, колонки, задачи и метки.
3. Подсистема управления задачами. Отвечает за создание, редактирование, перемещение и удаление задач в рамках проекта. Включает функциональность по работе с приоритетами, статусами, метками и комментариями.
4. Подсистема отчетности. Позволяет пользователям и руководителям генерировать отчеты о затраченном времени, выполненных задачах и общей продуктивности команды.
5. Подсистема управления метками. Отвечает за создание, редактирование и удаление меток, которые могут быть применены к задачам для улучшения их категоризации и фильтрации.

Построенная концептуальная модель предметной области отражает основные сущности и связи между ними, которые используются в разрабатываемой системе учета рабочего времени и управления проектами. На основе данной модели была разработана структура приложения, включая модели, контроллеры и представления.

В соответствии с выделенными объектами и подсистемами был разработан набор функциональных требований к системе, охватывающий основные аспекты работы с пользователями, проектами, задачами, метками и отчетностью. Эти функциональные требования стали основой для разработки технического задания и дальнейшей реализации программного средства.

Таким образом, построение концептуальной модели предметной области позволяет определить основные сущности и взаимосвязи между ними в рамках разрабатываемой системы учета рабочего времени и управления проектами. В дальнейшем это облегчает процесс анализа требований, проектирования и разработки программного средства.

# 2 Постановка задачи

## 2.1 Определение требований к программной системе.

Перед тем как начать разрабатывать программное средство необходимо проанализировать требования к программному средству и поставить задачи.

Цель процесса анализа требований к программному средству заключается в установлении и документировании требований к программному обеспечению.

Разработка программного средства будет вестись на операционной системе Windows 10.

При разработке программного средства будут использованы следующие технологии: язык программирования Java, сервер базы данных H2. В качестве фреймворка будет выбран фреймворк Spring Framework.

Для реализации Back-end части будут использована технология Spring Framework.

Для реализации Front-end части будут использованы следующие технологии: Thymeleaf, HTML, CSS и JavaScript.

Каждая из выбранных технологий отвечает за разные аспекты работы программы.

Существует огромное количество инструментария для программирования как на стороне сервера, так и на стороне клиента, как платных, так и бесплатных.

Основные программные средства, которые будут использованы при разработке программного средства:

интегрированная среда разработки Eclipse для разработки на языках программирования Java и JavaScript, языке разметки HTML и каскадных таблиц стилей CSS с возможностями анализа кода на лету, предотвращения ошибок в коде и автоматизированными средствами рефакторинга для Java;

браузеры Opera и Google Chrome для проверки на кроссбраузерность программного средства, а также проверки результатов работы программного средства при разработке.

Все перечисленные выше серверные и клиентские технологии, а также соответствующий инструментарий идеально подходят для разработки веб-приложений.

Любой цикл разработки программного средства начинается с анализа требований. Цель этой стадии – определение детальных требований к программному средству.

Основные требования к программному средству системы управления персоналом предприятия представлены в таблице 1.

Таблица 1 - Основные требования к программному средству

|  |  |
| --- | --- |
| Вид требования | Содержание требования |
| Бизнес-требование (*B*1) | Все формы программного средства должны быть удобными для пользователей |
| Бизнес-требование (*B*2) | Для эксплуатации интерфейса программного средства от пользователей не должно требоваться специальных технических навыков, знания технологий или программных продуктов, за исключением общих навыков работы с персональным компьютером и стандартным веб-браузером |
| Нефункциональное требование (*NF*1) | Программное средство должно быть адаптивно под любое устройство |
| Нефункциональное требование (*NF*2) | Интерфейс программного средства должен быть на русском языке |
| Функциональное требование (*F*1) | В программном средстве должно быть предусмотрено разделение ролей на гостя, пользователя и администратора |
| Функциональное требование (*F*2) | Не авторизовавшись в системе, пользователь должен иметь доступ к навигационной панели, которая обеспечивает переход к основным пунктам меню |
| Функциональное требование (*F*3) | Каждому пользователю, в зависимости от роли, показывается свое меню |

Для обеспечения полноценной работы программы на персональном компьютере требуется:

* процессор AMD Athlon X4 840 3.10 GHz;
* ОЗУ: не менее 4Гб;
* HDD: не менее 250Гб;
* видеоадаптер: DirectX9;
* видеопамять: не менее 512 МБ;
* Windows 7 и выше;

Из программного обеспечения, для запуска и работы приложения требуется:

* Java 11;
* Maven 3.6.0

Целью разработки данного дипломного проекта является создание системы учета рабочего времени и управления проектами, которая позволяет пользователям создавать проекты, управлять задачами и отслеживать затраченное время. В данной системе каждый пользователь может выполнять разные функции в зависимости от своего статуса.

Требования к режиму гостя:

* Просмотр списка всех проектов
* Регистрация в системе
* Вход в систему

Требования к режиму пользователя (обычный участник проекта):

* Наследование от возможностей гостя
* Просмотр своего профиля
* Редактирование своего профиля (с валидацией полей)
* Создание нового проекта
* Просмотр списка своих проектов и проектов, в которых является участником
* Просмотр детальной информации о проекте, задачах, метках и столбцах
* Комментирование задач
* Просмотр статистики по проектам и задачам

Требования к режиму владельца проекта (owner):

* Наследование от возможностей пользователя
* Добавление участников в проект
* Редактирование и удаление проекта
* Создание, редактирование и удаление задач, меток и столбцов
* Перемещение задач между столбцами
* Просмотр списка участников проекта
* Возможность назначить другого пользователя владельцем проекта

Таким образом, требования к программной системе определены в соответствии с различными ролями пользователей в системе. Эти требования служат основой для разработки технического задания и дальнейшей реализации программного средства.

## 2.2 Описание аналогов системы

В данной главе будут рассмотрены аналоги системы учета рабочего времени и управления проектами. Были выбраны следующие аналоги:

1. Trello (<https://trello.com/>)
2. Asana (<https://asana.com/>)
3. ClickUp (<https://clickup.com/>)
4. Monday.com ([https://monday.com/](https://monday.com/" \t "_new))
5. Wrike ([https://www.wrike.com/](https://www.wrike.com/" \t "_new))

**Trello** - это популярный инструмент для управления проектами, основанный на методологии Kanban [3]. Он предлагает простой и интуитивно понятный интерфейс для управления задачами с помощью столбцов и карточек.

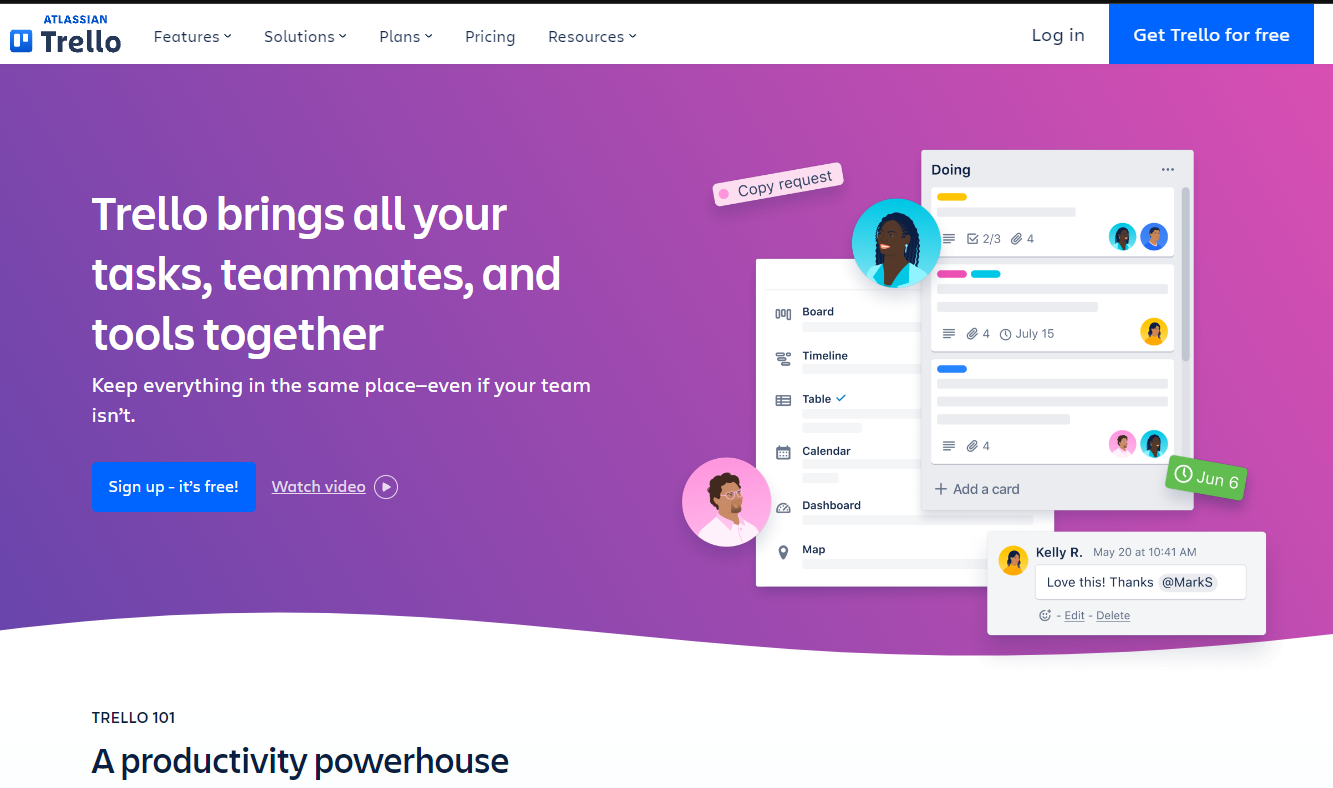


Рисунок 2.1 – Trello

Плюсы:

* Простота и удобство использования
* Бесплатная версия с ограниченным функционалом
* Интеграция с другими сервисами Минусы:
* Отсутствие встроенного учета времени
* Более сложные функции доступны только в платных версиях

**Asana** - это мощный инструмент для управления проектами, предоставляющий возможности для планирования, организации и отслеживания работы команды [4].

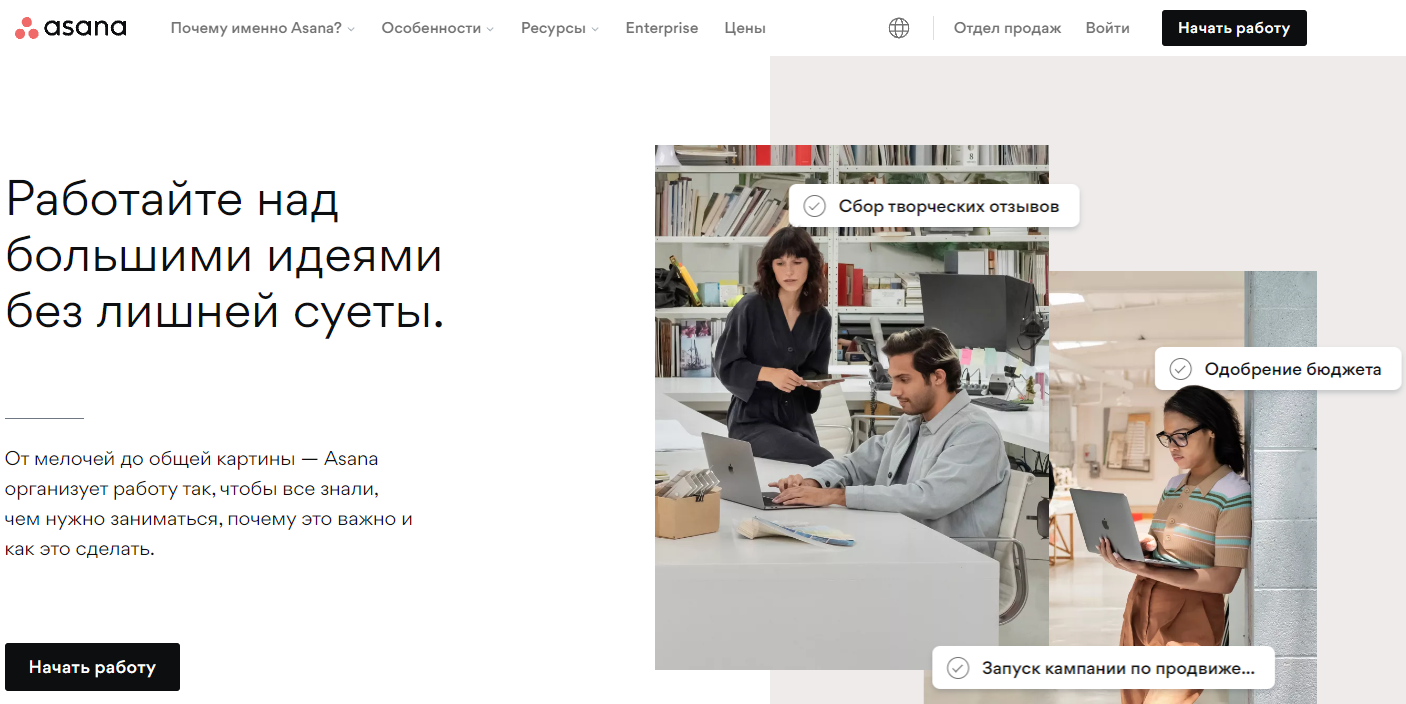


Рисунок 2.2 - Asana

Плюсы:

* Гибкая система организации задач
* Встроенный учет времени
* Интеграция с другими сервисами Минусы:
* Высокая стоимость платных версий
* Может быть сложным для новых пользователей

**ClickUp** - это универсальный инструмент для управления проектами, который включает в себя разнообразные функции и предлагает несколько видов отображения задач [5].

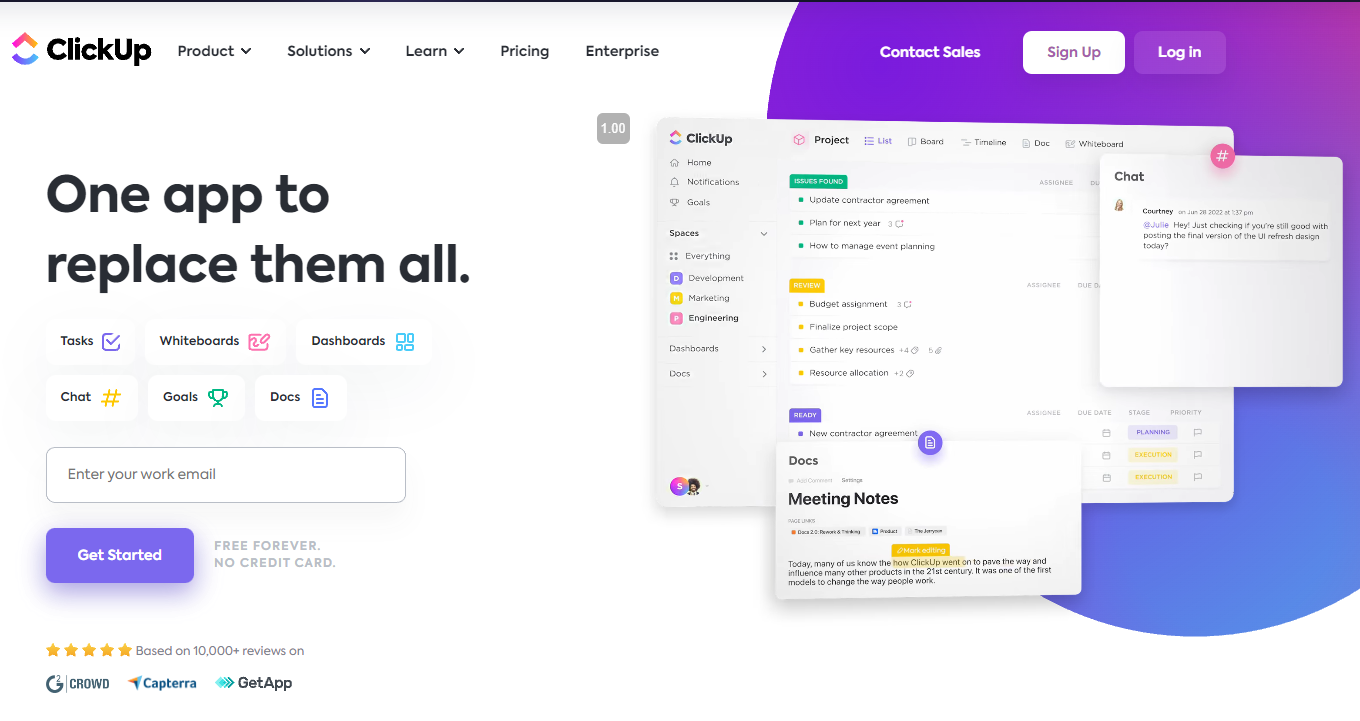


Рисунок 2.3 - ClickUp

Плюсы:

* Широкий набор функций
* Гибкие настройки отображения задач
* Бесплатная версия с ограниченным функционалом Минусы:
* Сложность и перегруженность интерфейса
* Некоторые функции доступны только в платных версиях

**Monday.com** - это платформа управления проектами, которая предоставляет интуитивно понятный интерфейс и широкий набор функций для планирования, организации и отслеживания работы команды [6].

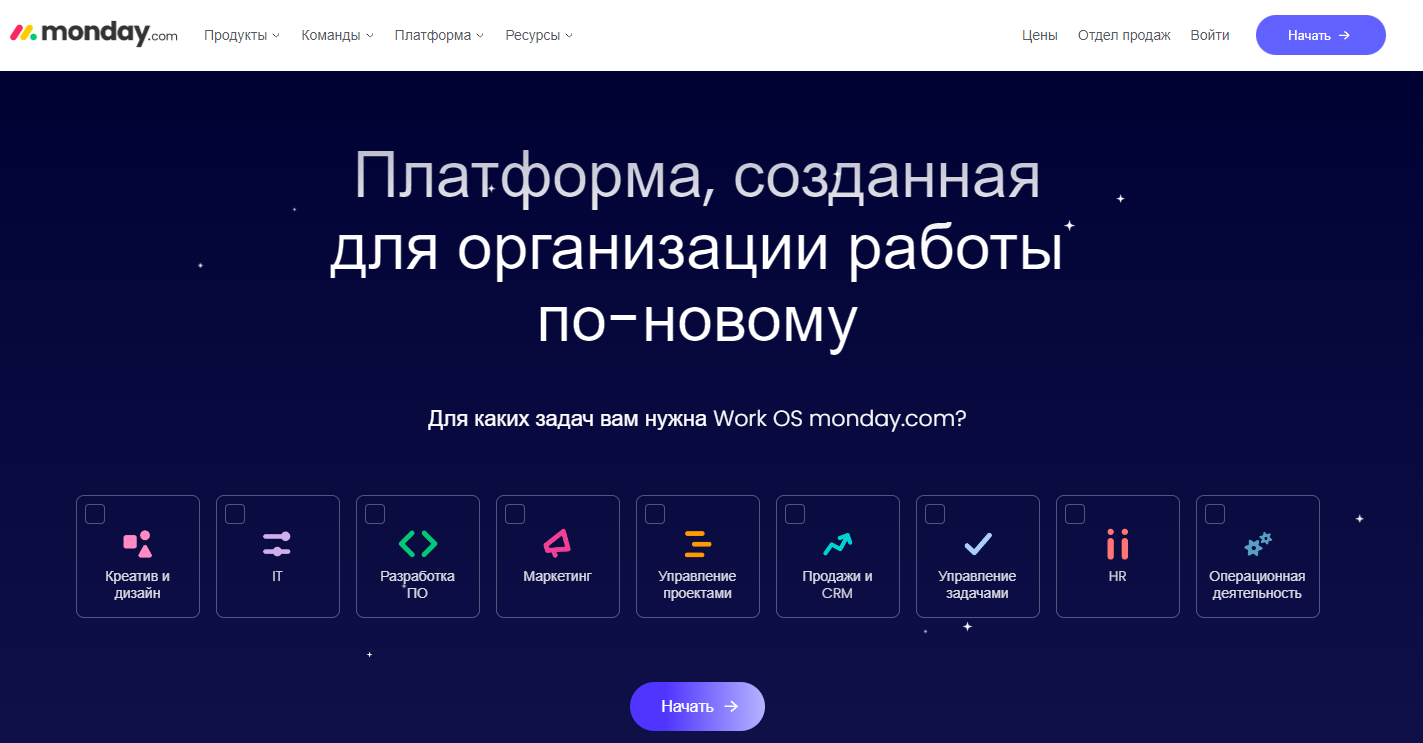


Рисунок 2.4 - Monday

Плюсы:

* Визуальное представление задач и проектов
* Широкий набор функций
* Интеграция с другими сервисами Минусы:
* Отсутствие бесплатной версии
* Высокая стоимость платных версий

**Wrike** - это инструмент для управления проектами, который предлагает гибкость и прозрачность для командной работы. Он позволяет настраивать рабочие процессы, а также предоставляет различные виды отображения задач для разных типов проектов [7].

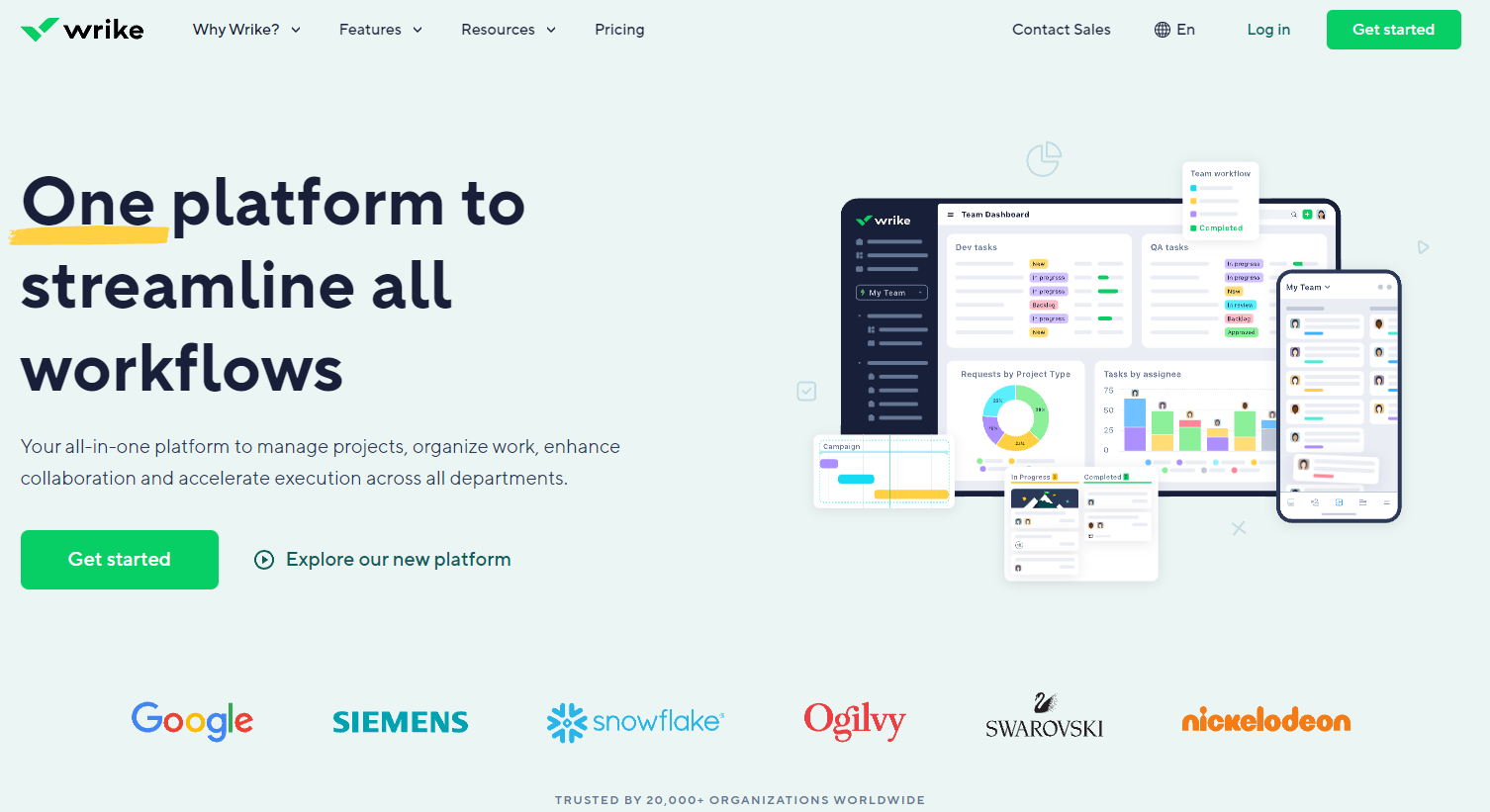


Рисунок 2.5 - Wrike

Плюсы:

* Гибкость и настраиваемость рабочих процессов
* Встроенный учет времени
* Интеграция с другими сервисами Минусы:
* Отсутствие бесплатной версии
* Сложный интерфейс для новых пользователей
* Высокая стоимость платных версий

Обобщение и сравнение аналогов с разрабатываемым приложением:

Все рассмотренные аналоги предоставляют полезные функции для управления проектами и отслеживания работы команды. Однако, некоторые из них имеют высокую стоимость платных версий или отсутствуют бесплатные версии. Также многие из них не предоставляют встроенного учета времени или имеют сложный интерфейс для новых пользователей.

Разрабатываемое приложение учета рабочего времени и управления проектами направлено на то, чтобы предоставить простой и интуитивно понятный интерфейс, а также функциональность, необходимую для эффективного управления проектами и контроля времени.

Наше приложение будет доступным для всех пользователей и предоставит встроенный учет времени, что является ключевой особенностью для контроля и оптимизации рабочего времени команды. Также, наше приложение позволит пользователям просматривать статистику и комментировать задачи, что поможет улучшить общение и сотрудничество в команде.

В целом, разрабатываемое приложение будет являться удобным и доступным инструментом для учета рабочего времени и управления проектами, учитывая потребности пользователей и предоставляя функциональность, отсутствующую в некоторых аналогах.

## 2.3 Обзор и обоснование выбора инструментальных средств

Программное обеспечение напрямую зависит от стека технологий и архитектуры разрабатываемого приложения. Архитектура данного проекта – это веб-приложение. Архитектура приложения во многом предопределило и технологии, используемые для построения проекта.

Основные технологии, применяемые при создании приложения:

* серверный язык *JAVA*;
* язык запросов *SQL* (*H2*);
* фреймворк *Spring Boot*;
* язык разметки *HTML*;
* таблица стилей *CSS*;
* клиентский язык *JavaScript*.

Уже на этом этапе можно определить необходимое программное обеспечение для разработки:

* IntelliJ IDEA;
* H2 console;

Каждая из выбранных технологий отвечает за разные аспекты работы программы.

Язык Java появился в 1995 году – 90-е годы были вообще урожайными на новые языки и концепции программирования. В таком Эдеме языков важно было не заблудиться, по ошибке приняв за Священный Грааль технологию, которая не пройдет испытания временем. Java прошел испытания, хотя и очень долгие. Очень не рекомендуется путать этот язык с JavaScript – они по виду похожи, но это совсем разные языки [8].

Вероятно, в Java впервые реализовали концепцию того, что язык должен быть максимально изолирован от платформы разработки, чтобы применять его без изменений везде: в компьютерах, часах, сотовых телефонах, бытовой технике. С «железной частью» должна была справляться виртуальная машина (JVM), которая, собственно, и создавалась индивидуально под каждое устройство. Сам же язык был неизменен и в качестве результата выдавал байт-код.

С самого начала было известно, что код не может исполняться очень быстро, но многие устройства не требовали высокой скорости исполнения. Кроме того, со временем появились оптимизирующие компиляторы, так что, в среднем, программа на Java работает раза в 2-3 медленнее, чем на C++.

Постоянное сравнение с C/C++ здесь не случайно: многие современные языки взяли за основу его конструкции и синтаксис, так что, бывает, узнать сходу язык очень трудно. Вместе с тем, Java с тех пор сильно «размножилась», и даже J#, J и прочие аналоги являются не родными братьями, а лишь подобием.

Сама идея языка, вполне, кстати, достаточного для создания софта любой сложности, была сначала не понята: был ли, мол, смысл создавать между аппаратурой и кодом промежуточные слои исполняющих машин? Со временем сомнения рассеялись: появилась мультиязычная платфома .

NET, и даже в Windows появились слои – аппаратно-зависимые, платформо-независимые. Самое же простое объяснение – софт стал очень сложным, а программисты очень ленивыми, чтобы переписывать программы под каждый отдельный аппарат.

Но вернемся к языку. Как уже говорилось, чем-то он похож на C++, чем-то на старый добрый Бейсик. Нет сейчас ни одного языка, который бы не хвалился своими возможностями ООП, и Java здесь не отличается от канонов: классы и объекты здесь используются везде, даже в самых примитивных задачах вроде вывода строки на экран. Из особенностей можно отметить, что все объекты в языке создаются только динамически, а все функции являются методами классов.

Множественное наследование не поддерживается, как в C++, как и «опасные» указатели. ООП дает много преимуществ, но и требует слишком многого – в случае Java памяти устройства никогда не будет слишком много.

В остальном же, имеются библиотеки классов для практически всех задач; преимущественно – под написание клиентских и серверных приложений. Хозяин Java – Oracle – успешно использует язык для использования в разработках своей одноименной СУБД. На сегодняшний день язык считается наиболее востребованным на рынке.

Spring Framework (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET [9].

Первая версия была написана Родом Джонсоном, который впервые опубликовал её вместе с изданием своей книги «Expert One-on-One Java EE Design and Development» (Wrox Press, октябрь 2002 года).

Фреймворк был впервые выпущен под лицензией Apache 2.0 license в июне 2003 года. Первая стабильная версия 1.0 была выпущена в марте 2004. Spring 2.0 был выпущен в октябре 2006, Spring 2.5 — в ноябре 2007, Spring 3.0 в декабре 2009, и Spring 3.1 в декабре 2011. Текущая версия — 5.2.x.

Несмотря на то, что Spring не обеспечивал какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring предоставляет бо́льшую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба.

Между тем, особенности ядра Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк.

Следующие две технологии – это *HTML* и *CSS*, которые предопределены архитектурой проекта. Имеется единственное обоснование их выбора – это веб-архитектура разрабатываемого программного средства.

*HTML* – язык гипертекстовой разметки, который используется для структурирования и отображения веб-страницы и ее контента. Для разработки данного программного средства будет применяться версия этого языка *HTML*5 [10].

*CSS* – это формальный язык описания внешнего вида документа, написанного с использованием языка разметки. Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки *HTML.* Применяемая версия данной технологии – *CSS*3 [11].

Последняя из основных применяемый технологий – это *JavaScript*. С помощью данного языка программирования, можно создавать динамически обновляемый контент, управлять мультимедиа, анимировать изображения. Что касается обоснований выбора в качестве клиентского языка *JavaScript*, то здесь выбор достаточно очевиден, ведь конкурентов у данного языка в данной сфере практически нет [12].

Лишь некоторые достоинства данного языка: возможность навигации и управления по *DOM HTML*-страницы, возможность управления браузером, гибкость подхода объектно-ориентированного программирования, поддержка асинхронности.

Для проектирования программного средства используется язык графического описания *UML*.

*UML* – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур [13].

*UML* является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой *UML*-моделью. *UML* был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. *UML* не является языком программирования, на основании *UML*-моделей возможна генерация кода.

Сегодня об IntelliJ IDEA знает, без преувеличения, весь мир. Хотя платформа ориентирована, в первую очередь, на Java-кодинг, ее универсальность позволяет работать с разными языками.

Конечно, здесь нет такого массированного набора плагинов, как у любого open-source конкурента, но зато в их качестве можно быть относительно уверенным. Если сравнивать эту среду с операционными системами, то IntelliJ будет сродни Mac OS — закрытая, слегка консервативная, но надежности не занимать. Да и нет больших сложностей в работе. Как только вы захотите создать новый проект, интуитивное меню подскажет, как это сделать .

Но лишь сегодня у IntelliJ IDEA красивый и интуитивный интерфейс, который позволяет разобраться даже новичку. Несколько лет назад, когда команда только стартовала, они разрабатывали свою среду разработки на основе тех же продуктов с открытым кодом. Сначала это даже не должно было быть полноценной IDE.

Вместо этого компания создавала инструменты, которые должны были взаимодействовать со средой, обладающей популярностью в те годы — JBuilder. Но с учетом того, что последняя была далека от совершенства, первая IntelliJ IDEA увидела мир уже через год после начала работы стартапа.

В первую очередь, команда сконцентрировалась на создании средств для рефакторинга и анализа. Как раз из этого родилась современная IDEA. Развиваясь в этом направлении, создатели смогли разработать продукт, в котором код пишется частично вами, а частично машинным интеллектом. А все начиналось с того, чтобы обеспечить вовсе утилитарные функции: комфортное переименование классов, методов, автоматическое определение и прочие.

# 3 Проектирование

## 3.1 Разработка архитектуры программного продукта

Фреймворк Spring MVC обеспечивает архитектуру паттерна Model — View — Controller (Модель — Отображение (далее — Вид) — Контроллер) при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними [14].

* Model (Модель) инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).
* View (Отображение, Вид) отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.
* Controller (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

Вся логика работы Spring MVC построена вокруг DispatcherServlet, который принимает и обрабатывает все HTTP-запросы (из UI) и ответы на них. Рабочий процесс обработки запроса DispatcherServlet'ом проиллюстрирован на следующей диаграмме:

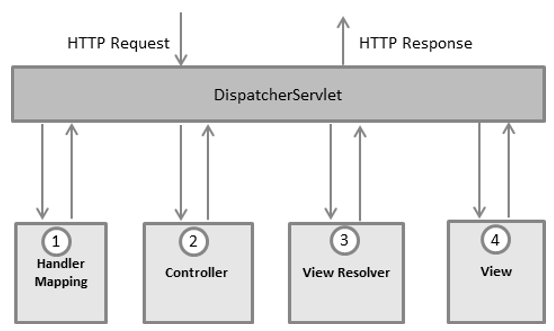


Рисунок 3.1 – Рабочий процесс обработки запроса

Ниже приведена последовательность событий, соответствующая входящему HTTP-запросу:

* После получения HTTP-запроса DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет, какой Контроллер должен быть вызван, после чего, отправляет запрос в нужный Контроллер.
* Контроллер принимает запрос и вызывает соответствующий служебный метод, основанный на GET или POST. Вызванный метод определяет данные Модели, основанные на определённой бизнес-логике и возвращает в DispatcherServlet имя Вида (View).
* При помощи интерфейса ViewResolver DispatcherServlet определяет, какой Вид нужно использовать на основании полученного имени.
* После того, как Вид (View) создан, DispatcherServlet отправляет данные Модели в виде атрибутов в Вид, который в конечном итоге отображается в браузере.

Все вышеупомянутые компоненты, а именно, HandlerMapping, Controller и ViewResolver, являются частями интерфейса WebApplicationContext extends ApplicationContext, с некоторыми дополнительными особенностями, необходимыми для создания web-приложений.

DispatcherServlet отправляет запрос контроллерам для выполнения определённых функций. Аннотация @Controllerannotation указывает, что конкретный класс является контроллером. Аннотация @RequestMapping используется для мапинга (связывания) с URL для всего класса или для конкретного метода обработчика.

Аннотация Controller определяет класс как Контроллер Spring MVC. В первом случае, @RequestMapping указывает, что все методы в данном Контроллере относятся к URL-адресу "/hello".

Следующая аннотация @RequestMapping(method = RequestMethod.GET) используется для объявления метода printHello() как дефолтного метода для обработки HTTP-запросов GET. Вы можете определить любой другой метод как обработчик всех POST-запросов по данному URL-адресу.

Вы можете написать вышеуказанный Контроллер по-другому, указав дополнительные атрибуты для аннотации @RequestMapping следующим образом

@RequestMapping(value = "/hello", method = RequestMethod.GET)  
Атрибут «value» указывает URL, с которым мы связываем данный метод (value = "/hello"), далее указывается, что этот метод будет обрабатывать GET-запросы (method = RequestMethod.GET). Также, нужно отметить важные моменты в отношении приведённого выше контроллера:

* Вы определяете бизнес-логику внутри связанного таким образом служебного метода. Из него Вы можете вызывать любые другие методы.
* Основываясь на заданной бизнес-логике, в рамках этого метода Вы создаёте Модель (Model). Вы можете добавлять аттрибуты Модели, которые будут добавлены в Вид (View). В примере выше мы создаём Модель с атрибутом «message».
* Данный служебный метод возвращает имя Вида в виде строки String. В данном случае, запрашиваемый Вид имеет имя «hello».

Spring MVC поддерживает множество типов Видов для различных технологий отображения страницы. В том числе — JSP, HTML, PDF, Excel, XML, Velocity templates, XSLT, JSON, каналы Atom и RSS, JasperReports и проч. Но чаще всего используются шаблоны JSP, написанные при помощи JSTL или HTML файлы, используя Thymeleaf.

На рисунке 3.2 представлена диаграмма последовательности, которая представляет собой один конкретный экземпляр работы программы под управлением пользователя.

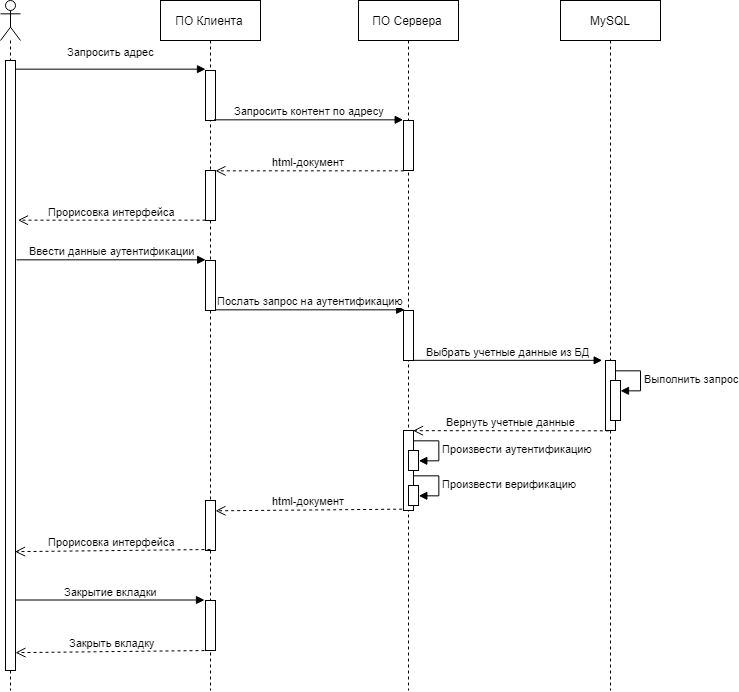


Рисунок 3.2 – Диаграмма последовательности

Диаграмма последовательности – это диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта (создание – деятельность – уничтожение некой сущности) и взаимодействие актеров (действующих лиц) информационной системы в рамках какого-либо определённого прецедента (отправка запросов и получение ответов).

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни», отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

На диаграмме последовательности изображен один актёр – пользователь системы, а также следующие объекты:

– ПО Клиента;

– ПО Сервера;

– MySQL.

Представлена последовательность действия в три этапа:

1. Пользователь вводит адрес сайта в браузере, а сервер отправляет клиенту html-документ.

2. Клиент вводит данные аутентификации. Браузер посылает запрос серверу на аутентификацию, а сервер посылает запрос на сервер MySQL, который выбирает данные и посылает их серверу. Сервер их преобразует и возвращает клиентскому браузеру. Браузер отображает информацию для клиента.

3. Клиент нажимает кнопку закрытия в браузере – вкладка закрывается. Работа системы завершена.

## 3.2 Проектирование структур хранения данных

Информационное обеспечение — совокупность системы классификации и кодирования, системы показателей, языков записи данных, унифицированных систем документации и массивов информации, используемых в автоматизированных системах управления [15].

Информационное обеспечение включает:

* состав информации, т. е. перечень информационных единиц или информационных совокупностей (показателей, констант, переменных, документов, других сообщений, необходимых для решения комплекса задач системы);
* структуру информации и закономерности ее преобразования, т. е. правила построения показателей, документов, агрегации и декомпозиции информационных единиц, преобразования информационных единиц в цепочке «вход — система — выход»;
* характеристики движения информации, т. е. количественные оценки потоков информации (объем, интенсивность), определение маршрутов движения документов, построение схем документооборота, временные характеристики функционирования источников информации, получения первичных данных, использования исходных данных, продолжительности хранения, старения и обновления данных.

Иными словами, в этом подразделе следует разобрать, какая информация нужна для построения системы, как с ней будет взаимодействовать пользователь и как она будет храниться и обрабатываться.

В данном подразделе будет представлено описание информационной модели данных и ее структура, которая обеспечивает работу системы. При этом данное описание будет, как в текстовом формате, так и при помощи диаграммы в нотации *IDEF*1*X* в *MYSQL Workbench*.

*IDEF*1*X* является методом для разработки реляционных баз данных и использует условный синтаксис, специально разработанный для удобного построения концептуальной схемы. Концептуальной схемой называется универсальное представление структуры данных в рамках коммерческого предприятия, независимое от конечной реализации базы данных и аппаратной платформы . Логическая модель данных является визуальным представлением структур данных, их атрибутов и бизнес-правил. Логическая модель представляет данные таким образом, чтобы они легко воспринимались бизнес-пользователями. Модель базы данных представлена на рисунке ниже.

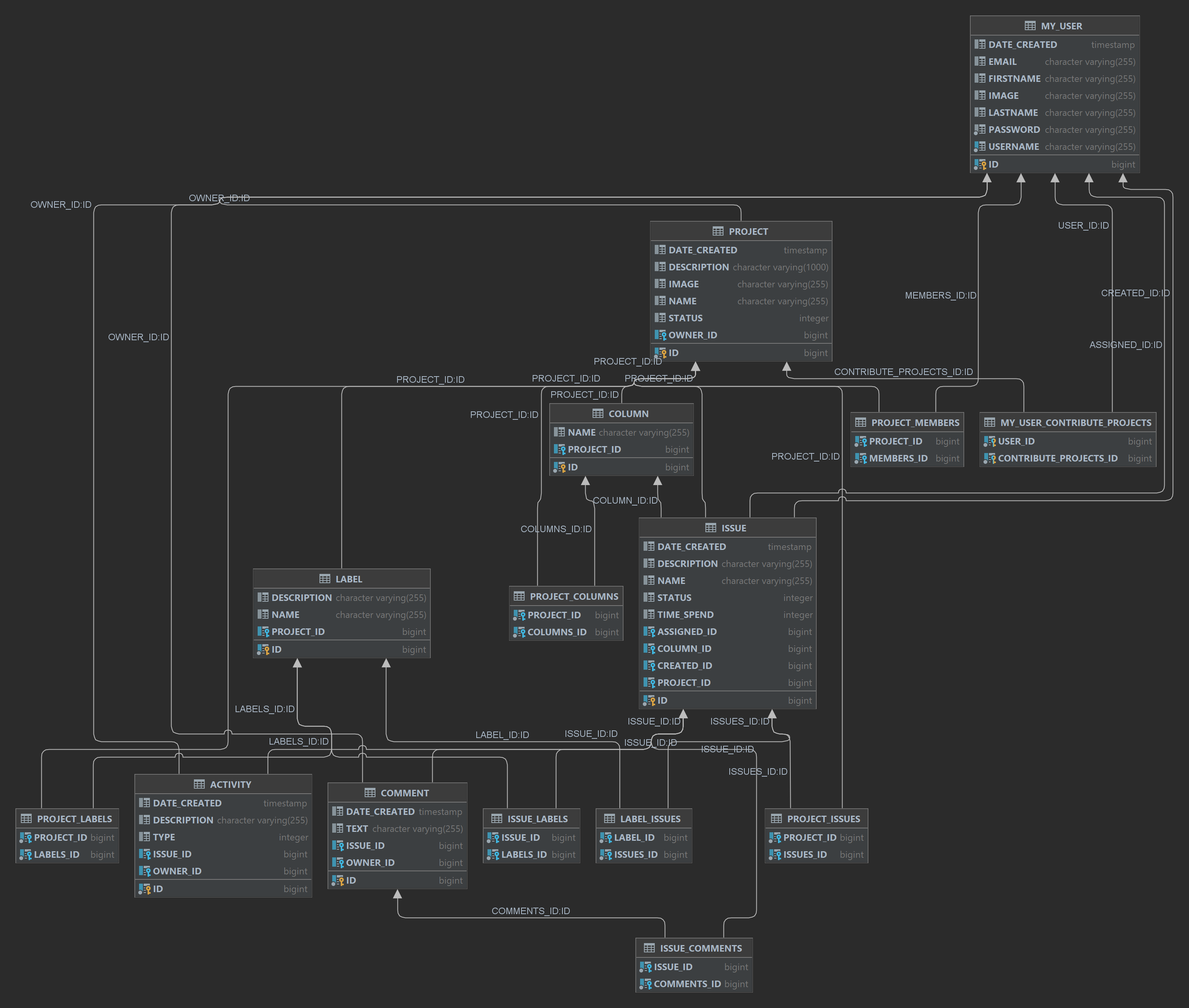


Рисунок 3.3 – Информационная модель

В данной информационной модели присутствует девять основных сущностей. Перечень сущностей:

1. Пользователь (MY\_USER). Хранит информацию о всех пользователях системы, такую как имя, фамилия, адрес электронной почты, пароль и имя пользователя.
2. Проект (PROJECT). Содержит информацию о проектах, включая описание, название, статус и владельца проекта.
3. Колонка (COLUMN). Отображает колонки в рамках проекта, в которых хранятся задачи. Содержит название колонки и ссылку на проект, к которому она принадлежит.
4. Задача (ISSUE). Хранит информацию о задачах, такую как название, описание, статус, затраченное время, создатель задачи, ответственный за задачу, колонка и проект, к которому задача относится.
5. Активность (ACTIVITY). Содержит информацию об активностях, связанных с задачами, такую как описание, тип активности, идентификатор задачи и владелец активности.
6. Комментарий (COMMENT). Содержит текст комментария, идентификатор задачи, к которой относится комментарий, и идентификатор владельца комментария.
7. Метка (LABEL). Хранит информацию о метках, которые можно присваивать задачам, такую как название и описание метки, а также ссылку на проект, к которому она принадлежит.
8. ISSUE\_COMMENTS. Связующая таблица, которая связывает задачи с комментариями.
9. ISSUE\_LABELS и LABEL\_ISSUES. Связующие таблицы, которые связывают задачи с метками и метки с задачами соответственно.

Также имеются дополнительные связующие таблицы для обеспечения связей между сущностями:

* MY\_USER\_CONTRIBUTE\_PROJECTS: связывает пользователей с проектами, в которых они участвуют.
* PROJECT\_COLUMNS: связывает проекты с колонками, которые в них содержатся.
* PROJECT\_ISSUES: связывает проекты с задачами, которые в них содержатся.
* PROJECT\_LABELS: связывает проекты с метками, которые в них содержатся.
* PROJECT\_MEMBERS: связывает проекты с пользователями, которые являются членами этих проектов.

## 3.3 Описание реализации вариантов использования

Моделирование необходимо для понимания системы. Обычно, при этом единственной модели никогда не бывает достаточно. Наоборот, для понимания практически любой нетривиальной системы приходится разрабатывать большое количество взаимосвязанных моделей. В применении к программным системам это означает, что необходим язык, с помощью которого можно с различных точек зрения описать представления архитектуры системы на протяжении цикла ее разработки .

Диаграмма вариантов использования показывает, какая функциональность должна быть реализована в системе, основные функции, которые должны быть включены в систему, их окружение и взаимодействие функций с окружением.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых вариантов использования. При этом актером или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Спецификация программного средства содержит описание или ссылки на описания исполняемого программного обеспечения, исходных файлов и информацию о программной реализации, включая информацию проекта построения, компиляции, построения и процедуры модификации.

Описание спецификации вариантов использования будет произведено в графическом варианте при помощи диаграммы *UML* 2.0 – диаграммы вариантов использования .

Для данного проекта диаграмма вариантов использования представлена на рисунке 3.4.

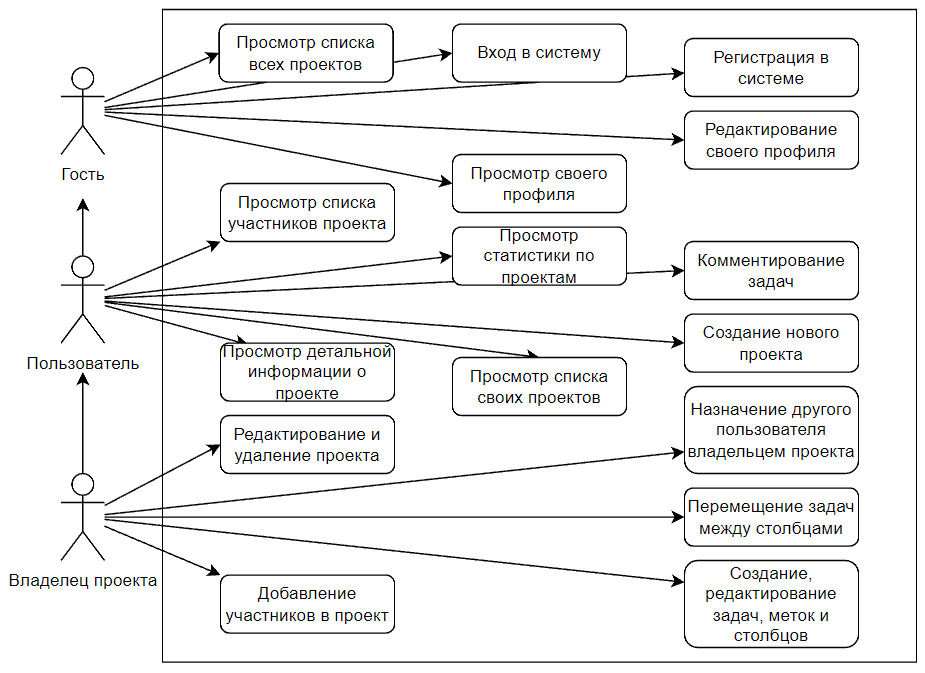


Рисунок 3.4– Диаграмма вариантов использования

Данная диаграмма демонстрирует все варианты использования системы всеми основными актёрами. Всего есть 3 актера: гость, пользователь и Владелец проекта.

Гость может просматривать список проектов и задач. Также может зарегистрироваться в системе и войти в нее. Как только гость вошел в систему, он становится пользователем.

Пользователь уже имеет больше прав: он может управлять задачами и проектами, в которых участвует.

Владелец имеет больше всего прав в системе. Он может напрямую работать с данными из таблиц проектов и задач, а также администрировать их. Администратор также может наблюдать за активностью пользователей и их взаимодействием с проектами и задачами.

В качестве основы для реализации вариантов использования служат все приведенные выше таблицы, которые хранят необходимые данные и обеспечивают их доступ для разных актеров системы.

В результате анализа предметной области и постановки задачи были определены варианты использования системы. Для реализации данных вариантов необходимо определить взаимодействие частей проектируемого программного средства. Диаграмма вариантов использования была отображена в разделе 3.3.

# 4 Реализация

## 4.1 Разработка классов информационной системы

Так как мы используем шаблон MVC, идет отдельно разработка классов модели (сущности), отдельно вида (страниц для браузера) и контроллеров (обработки действий пользователя).

Так как данные в шаблоне будут изменяться по требованию администратора ресурса, нам нужно создать класс-сущность для хранения этих данных.

Сущность (entity) – это реальный или представляемый тип объекта, информация о котором должна сохраняться и быть доступна.

Сущности уже были спроектированы раньше для базы данных. На их основе пишутся классы сервиса. Пример описания сущности приведен ниже.

Листинг 1. Сущность пользователя

package com.hescha.minijira.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import lombok.Data;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.ManyToMany;

import javax.persistence.OneToMany;

import javax.persistence.Table;

import java.time.LocalDateTime;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Objects;

import java.util.Set;

@Data

@Table(name = "myUser")

@Entity

public class User extends AbstractEntity {

private String firstname;

private String lastname;

@Column(unique = true, nullable = false)

private String username;

private String email;

@Column(nullable = false)

private String password;

private String image;

@JsonIgnore

@OneToMany(cascade = {CascadeType.PERSIST, CascadeType.REFRESH, CascadeType.DETACH}, mappedBy = "owner")

private List<Project> ownProjects = new ArrayList<>();

@JsonIgnore

@ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.REFRESH, CascadeType.DETACH})

private Set<Project> contributeProjects = new HashSet<>();

private LocalDateTime dateCreated;

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

if (!super.equals(o)) return false;

User user = (User) o;

return Objects.equals(firstname, user.firstname)

&& Objects.equals(lastname, user.lastname)

&& Objects.equals(username, user.username)

&& Objects.equals(email, user.email)

&& Objects.equals(password, user.password)

&& Objects.equals(image, user.image);

}

@Override

public int hashCode() {

return Objects.hash(super.hashCode(), firstname, lastname, username, email, password, image);

}

@Override

public String toString() {

return "User{" +

"username='" + username + '\'' +

", id=" + id +

'}';

}

}

По описанию сущности автоматически будет сгенерирована таблица в базе данных, при ее отсутствии.

Сущность написана, переходим к созданию сервисов для доступа к бд. Сперва опишем интерфейс с основными командами.

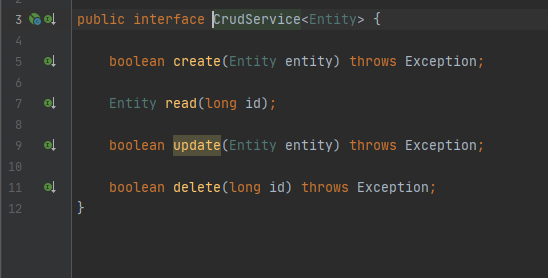


Рисунок 4.1 – Интерфейс с основными командами

Затем переходим к написанию реализации этого интерфейса. На самом деле, реализация простая, мы вызываем соответствующие методы репозитория.

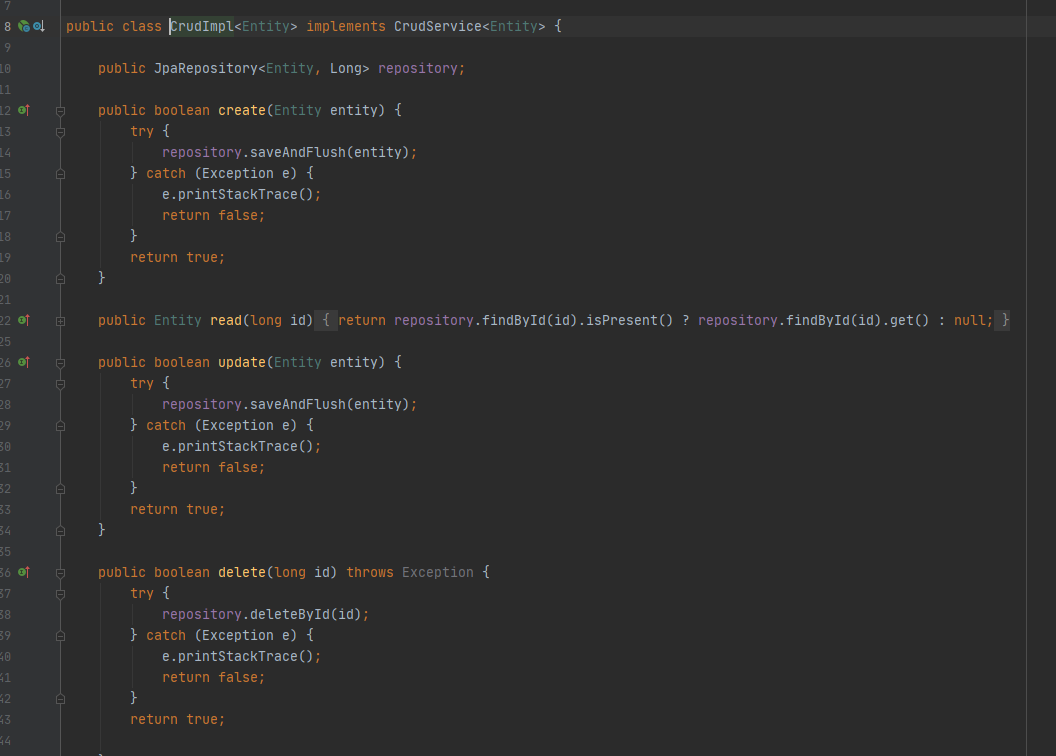


Рисунок 4.2 – Реализация интерфейса

Реализация есть, осталось связать нашу сущность шаблона с доступом к бд. Для этого наследуемся от только-что созданной реализации, и в конструктор передаем нашу сущность.

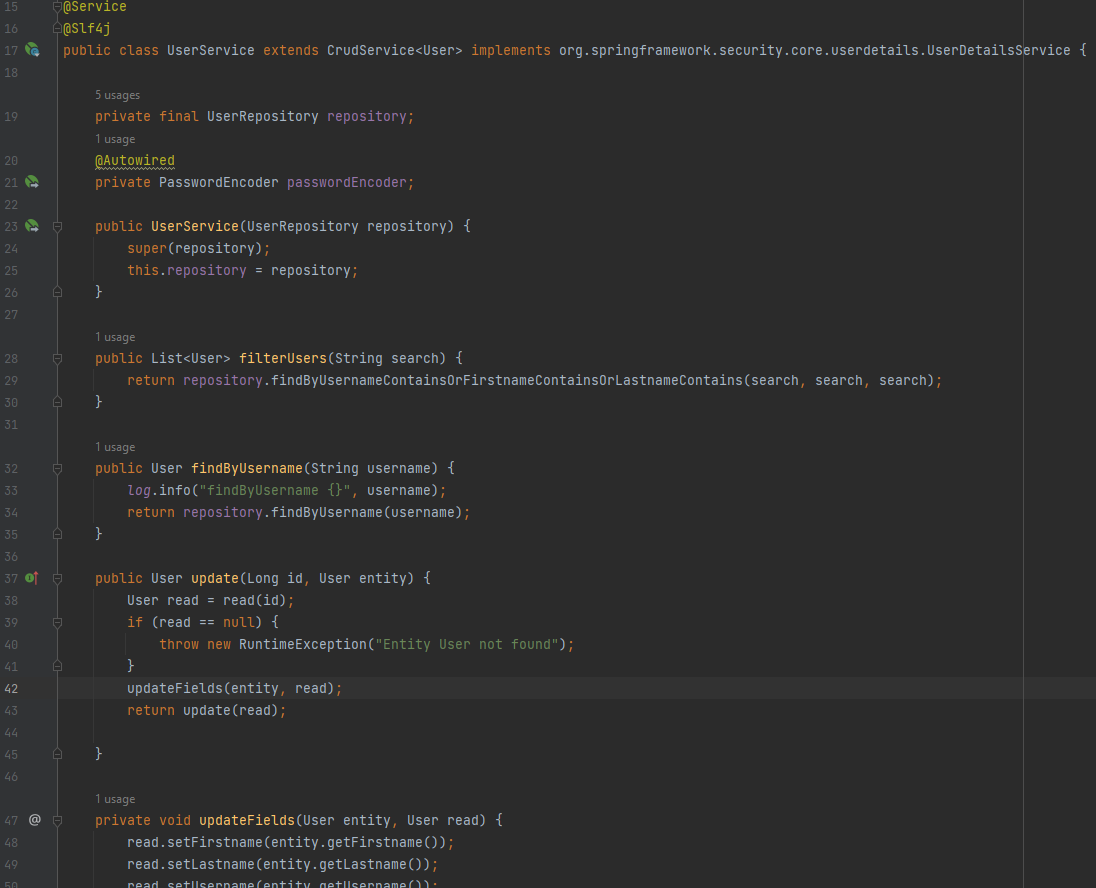


Рисунок 4.3 – Сервис доступа к таблице

Проделываем так для каждой сущности и готово, теперь, используя данный сервис, мы можем добавлять/редактировать/удалять данные в коде приложения.

Но это только в коде, пользователю это пока недоступно. Чтобы пользователь смог как-то взаимодействовать с приложением, создаем контроллеры страниц.

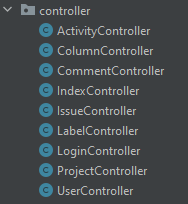


Рисунок 4.4 – Контроллеры страниц

По одному контроллеру для работы с соответствующими таблицами в базе данных.

Контроллер работает с данными таблиц базы данных, считывая или записывая в нее данные от пользователя. При запросе отображения, он возвращает страницу отображения. В этом контроллере реализованы методы создания, редактирования и удаления шаблонов по соответствующим адресам.

Листинг 2. Контроллер пользователей

package com.hescha.minijira.controller;

import com.hescha.minijira.model.User;

import com.hescha.minijira.service.SecurityService;

import com.hescha.minijira.service.UserService;

import lombok.RequiredArgsConstructor;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import java.util.List;

@Controller

@RequiredArgsConstructor

@RequestMapping(UserController.CURRENT\_ADDRESS)

public class UserController {

public static final String API\_PATH = "user";

public static final String CURRENT\_ADDRESS = "/" + API\_PATH;

public static final String MESSAGE = "message";

public static final String THYMELEAF\_TEMPLATE\_ALL\_ITEMS\_PAGE = API\_PATH;

public static final String THYMELEAF\_TEMPLATE\_ONE\_ITEM\_PAGE = THYMELEAF\_TEMPLATE\_ALL\_ITEMS\_PAGE + "-one";

public static final String THYMELEAF\_TEMPLATE\_EDIT\_PAGE = THYMELEAF\_TEMPLATE\_ALL\_ITEMS\_PAGE + "-edit";

public static final String REDIRECT\_TO\_ALL\_ITEMS = "redirect:" + CURRENT\_ADDRESS;

private final UserService service;

private final SecurityService securityService;

private final PasswordEncoder passwordEncoder;

@GetMapping

public String readAll(Model model) {

model.addAttribute("list", service.readAll());

return THYMELEAF\_TEMPLATE\_ALL\_ITEMS\_PAGE;

}

@GetMapping("/current")

public String readCurrent(Model model) {

model.addAttribute("entity", securityService.getLoggedIn());

return THYMELEAF\_TEMPLATE\_ONE\_ITEM\_PAGE;

}

@GetMapping("/search")

@ResponseBody

public List<User> filterUsers(@RequestParam(value = "search") String search) {

if (search.isEmpty()) return service.readAll();

return service.filterUsers(search);

}

@GetMapping("/{id}")

public String read(@PathVariable("id") Long id, Model model) {

model.addAttribute("entity", service.read(id));

return THYMELEAF\_TEMPLATE\_ONE\_ITEM\_PAGE;

}

@GetMapping(path = {"/edit", "/edit/{id}"})

public String editPage(Model model, @PathVariable(name = "id", required = false) Long id) {

if (id == null) {

model.addAttribute("entity", new User());

} else {

User read = service.read(id);

read.setPassword("");

model.addAttribute("entity", read);

}

return THYMELEAF\_TEMPLATE\_EDIT\_PAGE;

}

@PostMapping

public String save(@ModelAttribute User entity, RedirectAttributes ra) {

entity.setPassword(passwordEncoder.encode(entity.getPassword()));

if (entity.getId() == null) {

try {

User createdEntity = service.create(entity);

ra.addFlashAttribute(MESSAGE, "Creating is successful");

return REDIRECT\_TO\_ALL\_ITEMS + "/" + createdEntity.getId();

} catch (Exception e) {

ra.addFlashAttribute(MESSAGE, "Creating failed");

e.printStackTrace();

}

return REDIRECT\_TO\_ALL\_ITEMS;

} else {

try {

service.update(entity.getId(), entity);

ra.addFlashAttribute(MESSAGE, "Editing is successful");

} catch (Exception e) {

e.printStackTrace();

ra.addFlashAttribute(MESSAGE, "Editing failed");

}

return REDIRECT\_TO\_ALL\_ITEMS + "/" + entity.getId();

}

}

@GetMapping("/{id}/delete")

public String delete(@PathVariable Long id, RedirectAttributes ra) {

try {

service.delete(id);

ra.addFlashAttribute(MESSAGE, "Removing is successful");

} catch (Exception e) {

e.printStackTrace();

ra.addFlashAttribute(MESSAGE, "Removing failed");

}

return REDIRECT\_TO\_ALL\_ITEMS;

}

}

Класс наследует стандартный класс настройки безопасности. В нем мы сразу создаем данные для входа администратора и в методе конфигурации прописываем, какие пользователи куда могут попасть.

Поскольку использованные шаблоны нужно где-то хранить, создаем настройку подключения к базе данных. Все это делается в файле application.properties.

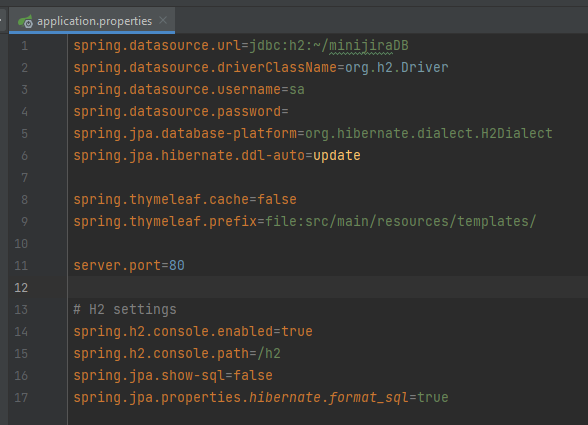


Рисунок 4.5 – Файл настройщик приложений

Проделав все эти операции, мы имеем следующую диаграмму классов

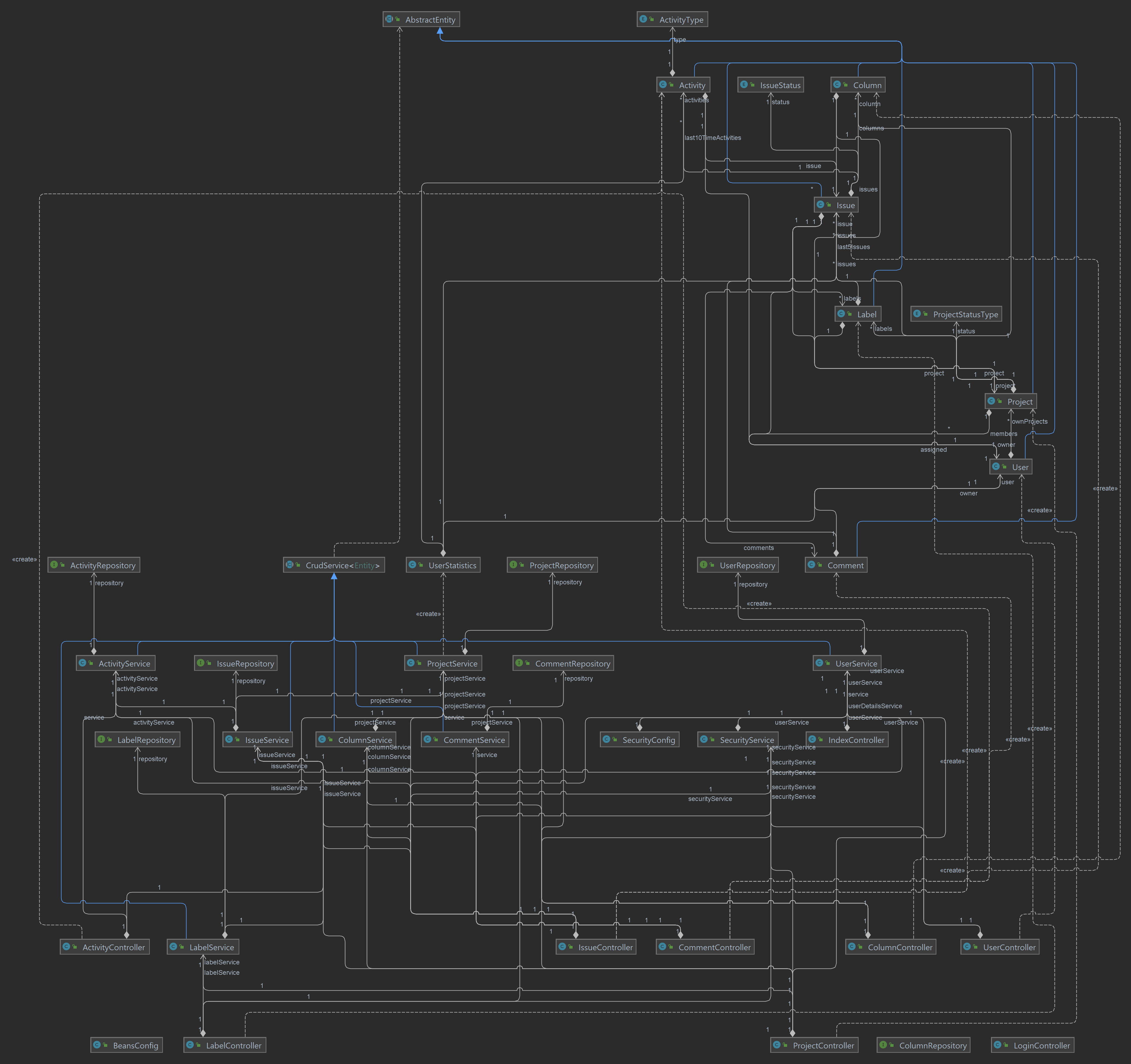


Рисунок 4.6 – Диаграмма классов

## 4.2 [Разработка интерфейса программного продукта](#_Toc474749001).

Определившись что и как должно быть, осталось спроектировать примерный пользовательский интерфейс. Пользовательский интерфейс представляет собой совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя и вычислительной системы. Важно помнить, что интерфейсы на стадии планирования и реализации могут отличатся, поскольку при написании приложения выявляется множество проблем и изменений.

ГОСТ «Эргономика взаимодействия человек-система», введенный в 2012 г., определяет пользовательский интерфейс (ПИ) как «компоненты интерактивной системы, предоставляющие пользователю информацию и являющиеся инструментами управления для выполнения определенных задач».

Проектирование пользовательского интерфейса – это создание тестовой версии приложения. Это начальный этап разработки пользовательского интерфейса, когда распределяются функции приложения по экранам, определяются макеты экранов, содержимое, элементы управления и их поведение.

Взаимодействие с системой происходит по разному, в зависимости от роли пользователя. При открытии приложения, пользователю показывается главная страница, отображенная на рисунке ниже.

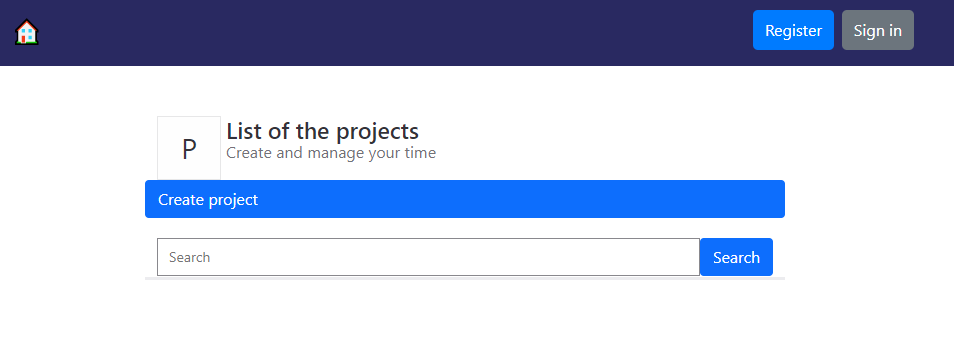


Рисунок 4.7 – Главная страница

На главной странице мы видим список который отображает все проекты нашего приложения. В данный момент в приложении нет проектов поэтому давайте создадим новый. Поскольку мы сейчас являемся гостями надо приложение нам сперва нужно зарегистрироваться.

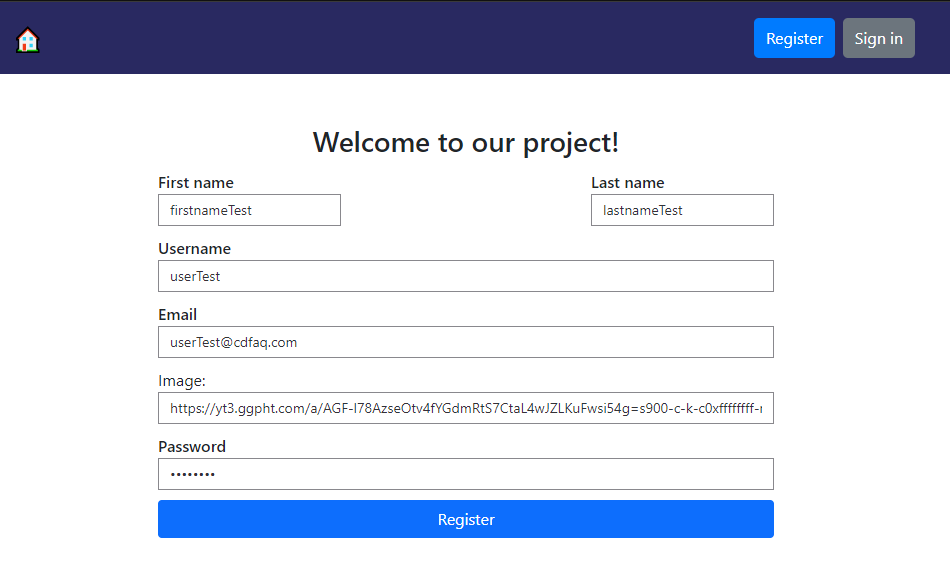


Рисунок 4.8 – Регистрация

На странице регистрации мы видим несколько полей, которые необходимо заполнить чтобы зарегистрироваться. Вводим в поля необходимые данные, в поле картинки вводим ссылку на картинку которая будет отображается в нашем профиле. После всех этих манипуляций жмём кнопку регистрации и видим, что мы успешно зарегистрировались.

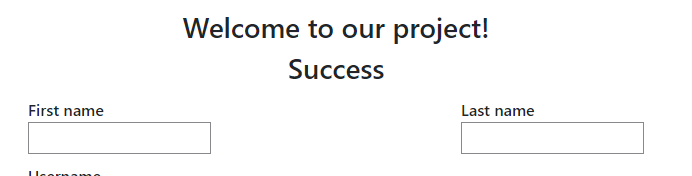


Рисунок 4.9 – Сообщение об успешной регистрации

Переходим на страницу входа в наше приложение. На этой странице нам необходимо заполнить наш юзернейм и пароль.

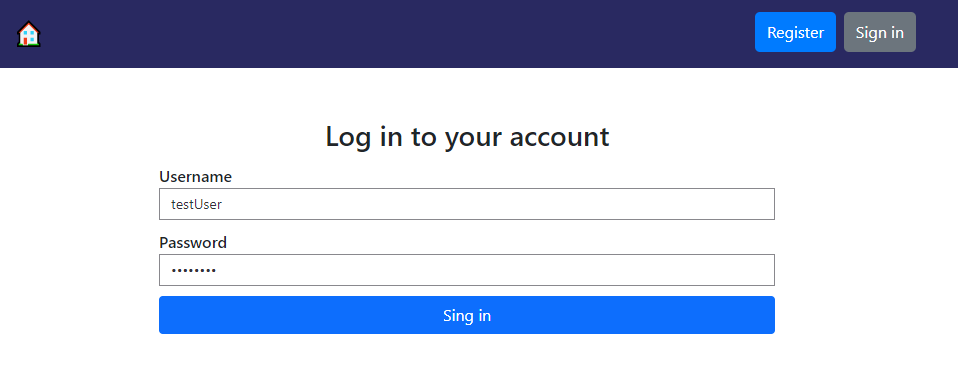


Рисунок 4.10 – Страница входа

После того как мы нажмем на кнопку входа, будем перенаправлены на страницу отображение проектов, так давайте же создадим свой первый проект.

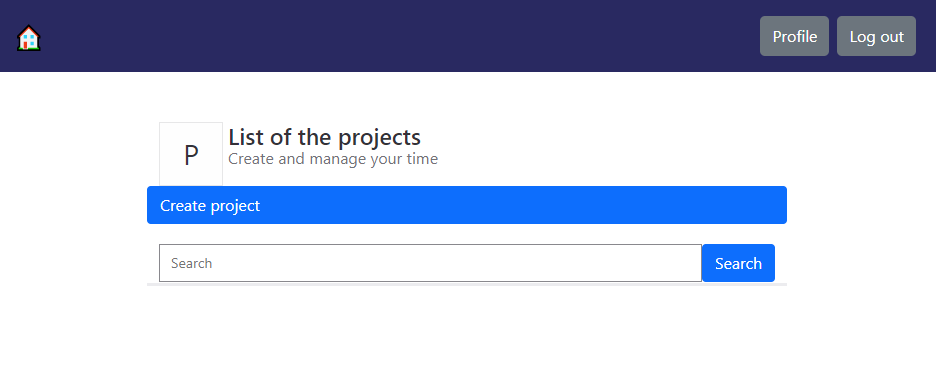


Рисунок 4.11 – Главная страница

На этой странице так и необходимо заполнить первоначальные данные, такие как название, описание и так далее. Дальше нажмем на кнопку сохранить проект мы увидим страницу нашего проекта.

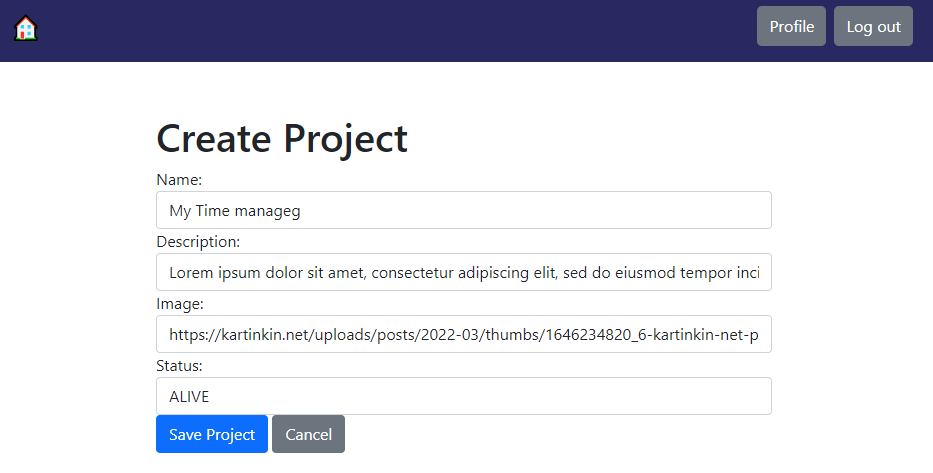


Рисунок 4.12 – Создание проекта

На этой странице мы можем управлять нашим проектом. Например, внизу страницы мы видим фильтр. Этот фильтр позволяет искать новых членов команды, и сразу же их добавлять.

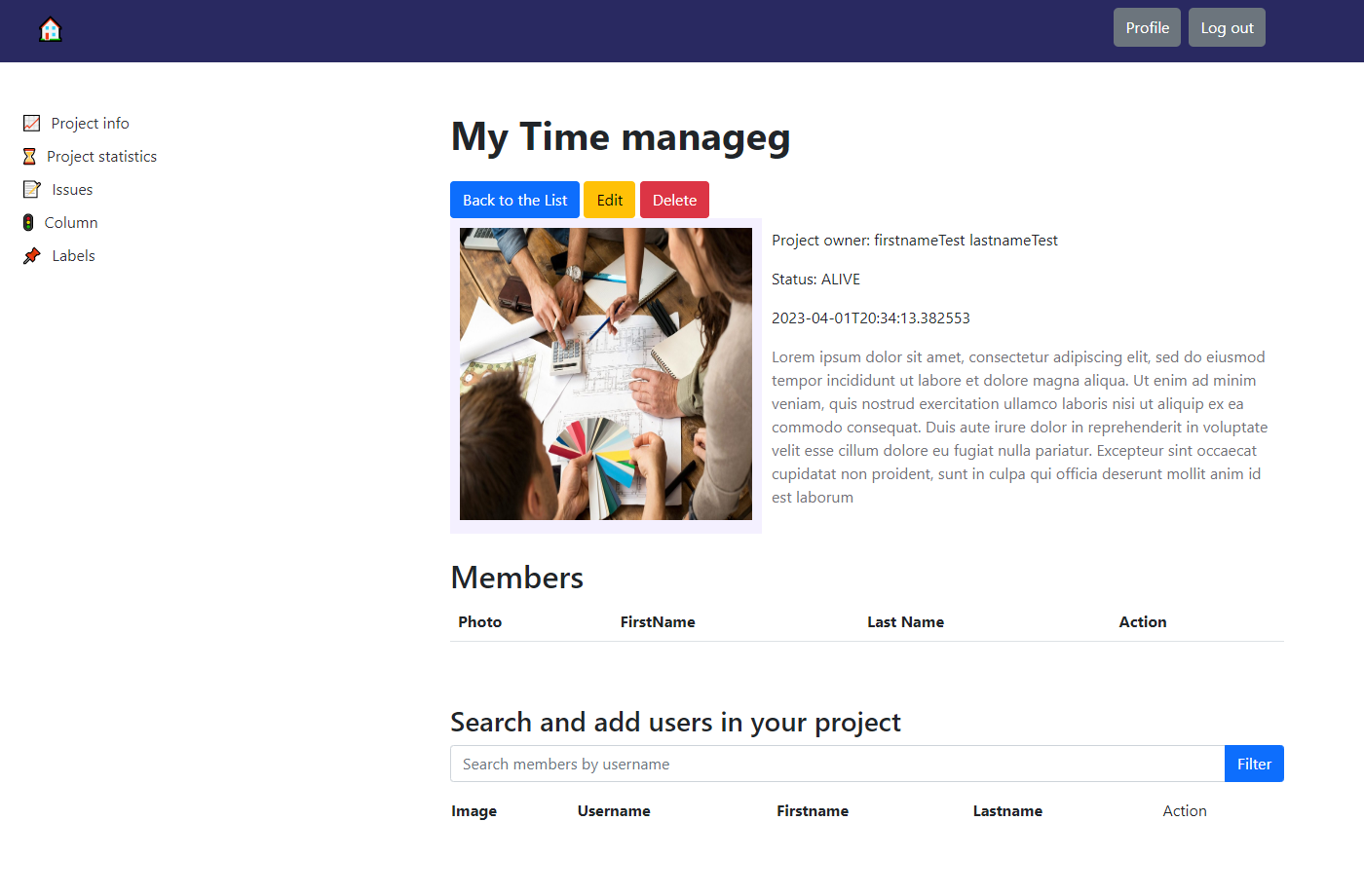


Рисунок 4.13 – Страница проекта

Сейчас мы видим список всех пользователей, которых можно добавить в проект. Сделаем это нажав соответствующую кнопку.

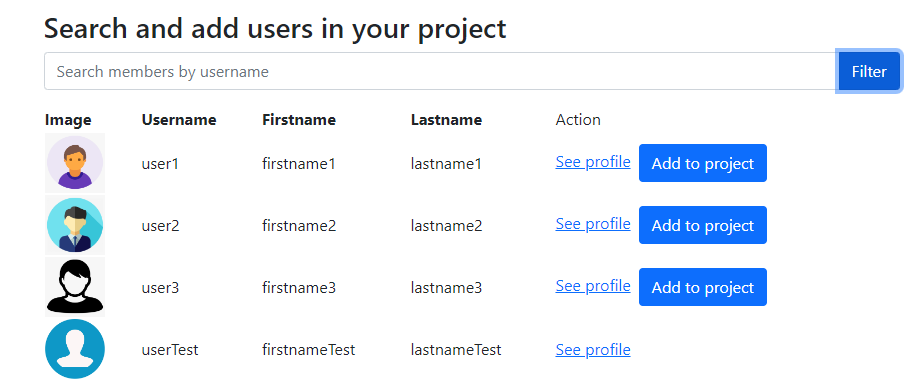


Рисунок 4.14 – Поиск и фильтрация пользователей

Мы видим что страница обновилась, и в списке участников проекта выводится новый пользователь. Также можно заметить, что в секции фильтрации выводятся уже другая кнопка, а именно кнопка удаления пользователя из проекта поскольку он уже находится в нем.

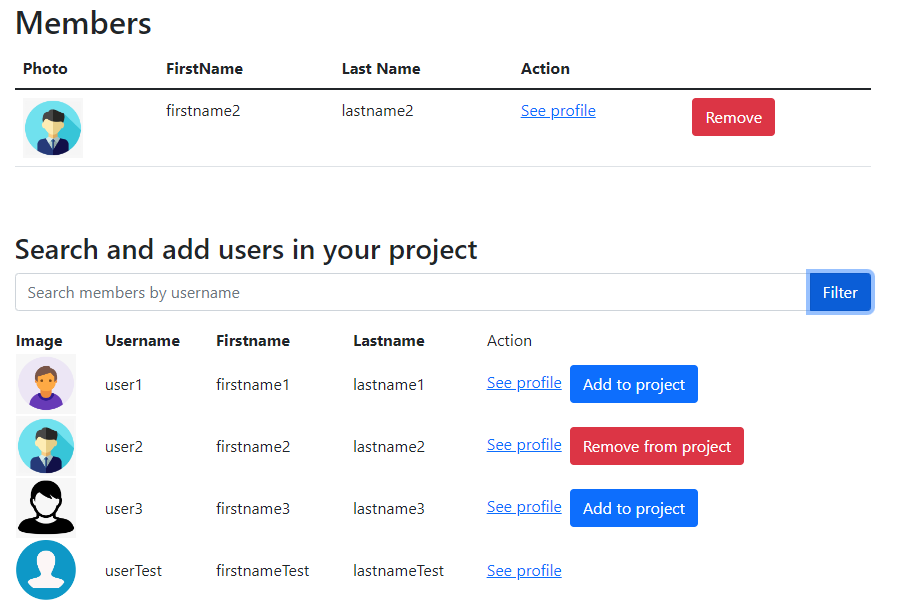


Рисунок 4.15 – Результат добавления пользователя

И перейдём уже на страницу меток. Это страница будут выводить со всей метки проекта. А давайте создадим свою метку.

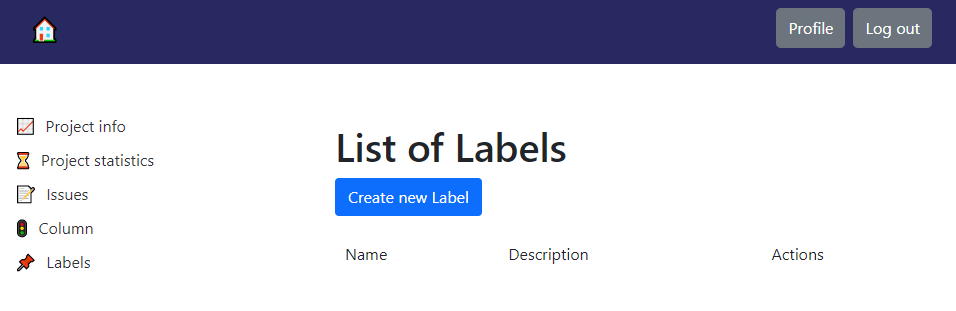


Рисунок 4.16 – Страница меток

На странице создания метки Необходимо вписать её имя и описание.

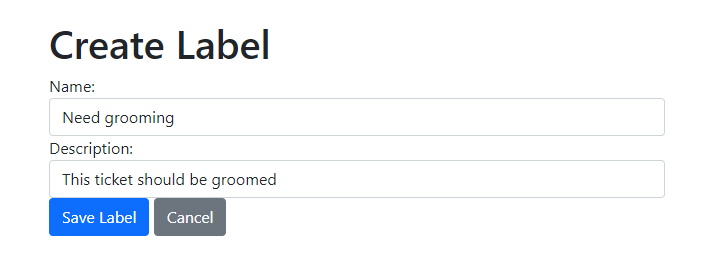


Рисунок 4.17 – Создание метки

После нажатия на кнопку сохранить метку мы переходим на страницу где отображаются все метки проекта.

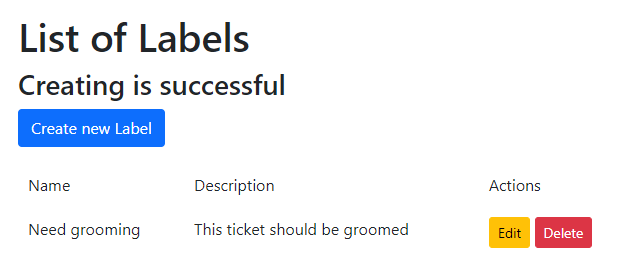


Рисунок 4.18 – Результат создания метки

Тем же способом добавляется и колонки в проект. Давайте создадим несколько это просмотрим их список.



Рисунок 4.19 – Результат создания колонок

Теперь давайте перейдём на страницу задач. На этой странице мы уже можем видеть колонки, добавленные ранее. В этих колонках будет находиться в нашей задачи. Давайте же создадим новую задачу, Для этого необходимо нажать на кнопку создать задачу.

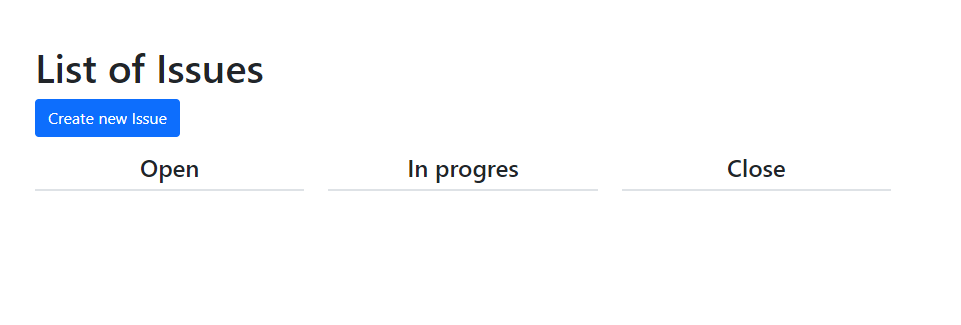


Рисунок 4.20 – Страница задач

Тут также необходимо заполнить все поля, в списке меток мы можем выбрать сразу несколько.



Рисунок 4.21 – Создание задачи

После нажатия на кнопку сохранение, мы будем переадресованы на страницу всех задач, и увидим нашим созданную задачу.

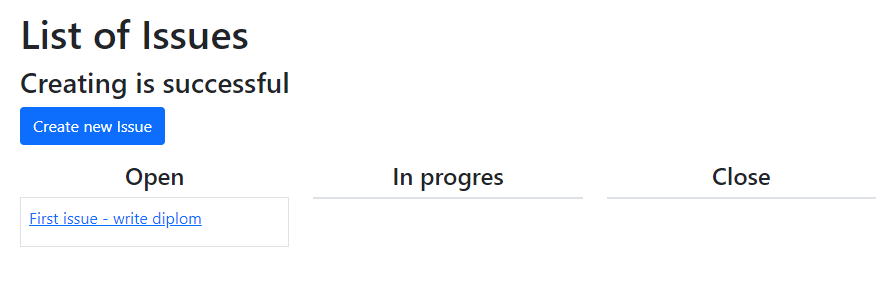


Рисунок 4.22 – Результат создание задачи

Как можно заметить, в колонке открытых находится наша задача. Можно переместить в другую колонку использовал только свою нишу для перемещения.



Рисунок 4.23 – Перемещение задачи в другую колонку

Все действия с задачей записываются в её истории. Давайте посмотрим на страницу этой задачи.

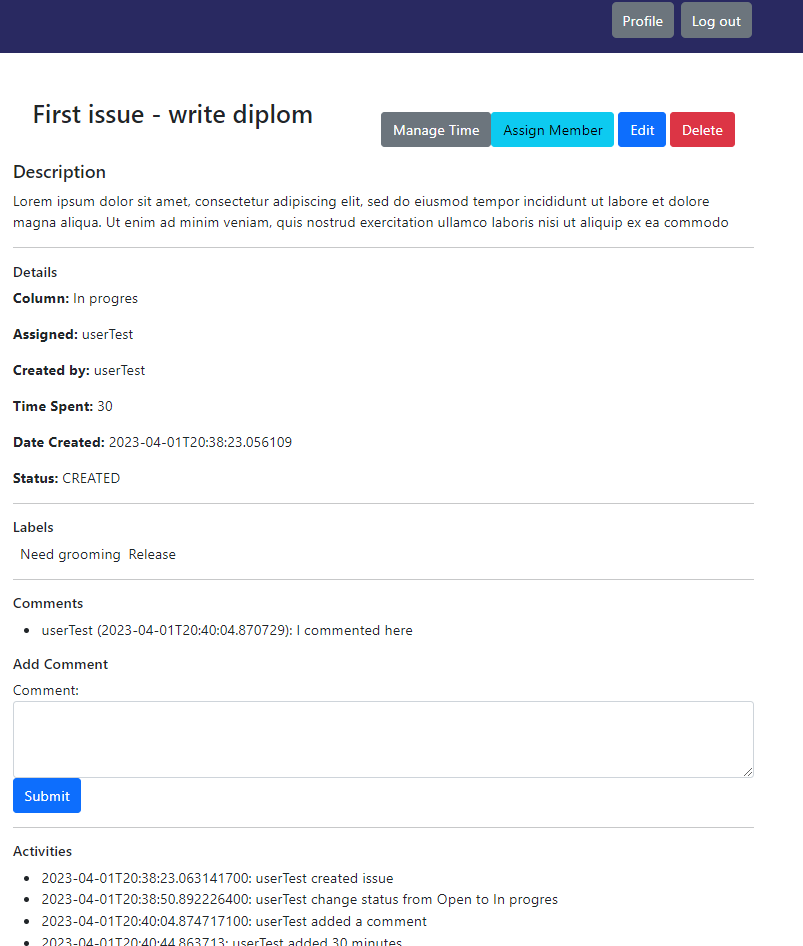


Рисунок 4.24 – Страница подробности задачи

На этой странице отображена вся информация о задачах. Здесь же мы можем добавить к задаче комментарии. Введём свой текст и нажмем на кнопку Отправить.

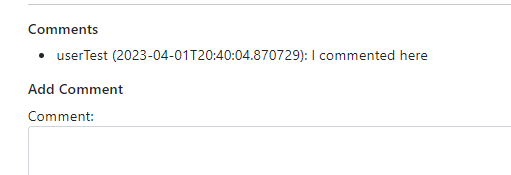


Рисунок 4.25 – Добавление комментария

Сейчас мы видим, что наш комментарий отображается в этой задаче. Вверху каждой задачи есть кнопка назначение пользователя. Давайте нажмём на неё.

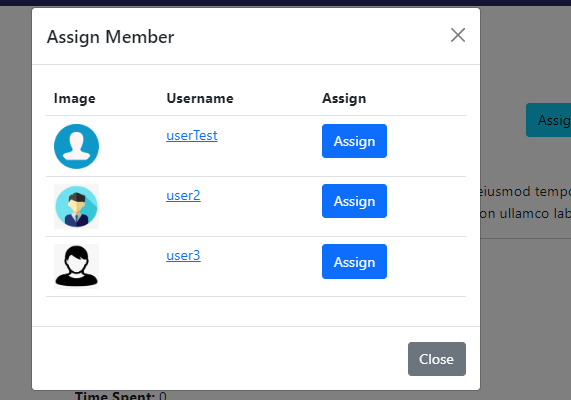


Рисунок 4.26 – Назначения пользователя на задачу

При нажатии на эту кнопку, мы видим форму назначения пользователей. Это Форма отвечает за то, какой пользователь проекта будет выполнять текущую задачу, и добавлять в неё своё время.

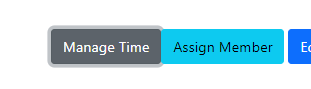


Рисунок 4.27 – Кнопка управления временем

После того, как пользователь назначен на задачу, он может увидеть около неё кнопку управления временем.При нажатии на эту кнопку, мы увидим новую форму. В этой форме пользователей вводит количество минут, которая он потратил на эту задачу.

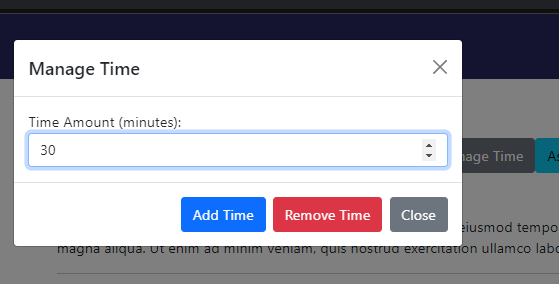


Рисунок 4.28 – Форма времени

Давайте введём сюда 30 минут, и на одном кнопку добавить время. Сейчас на странице проекта мы видим, что потраченное время на задачу составляет 30 минут. Также можно заметить, что внизу страницы, в секции активностей задачи добавлена новая строка, в которой сказано Кто и сколько минут потратил на эту задачу.

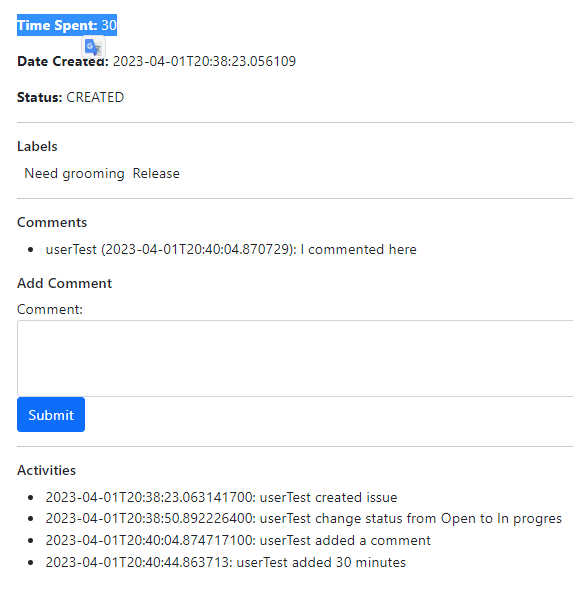


Рисунок 4.29 – Результат действий

При переходе на страницу статистики, мы увидим список всех пользователей проекта, которые вы о нём участвуют. Около каждого пользователя мы видим число активной задачи, за которые они ответственны. Сразу же около этого числа мы видим количество времени, которые они потратили в общем на проекте.

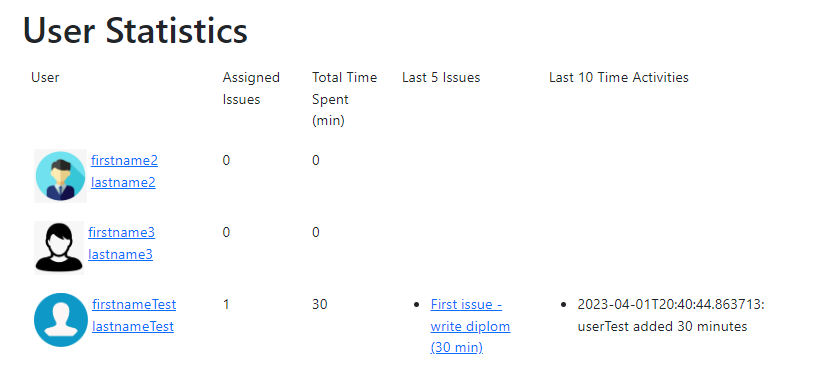


Рисунок 4.30 – Страница статистики

Рядом также выводится список из пяти задач, с которыми они работали в последнее время. Также выводится их последняя активность на задачу. Любой пользователь является ссылкой, так что мы можем перейти на их страницу и посмотреть информацию о них.

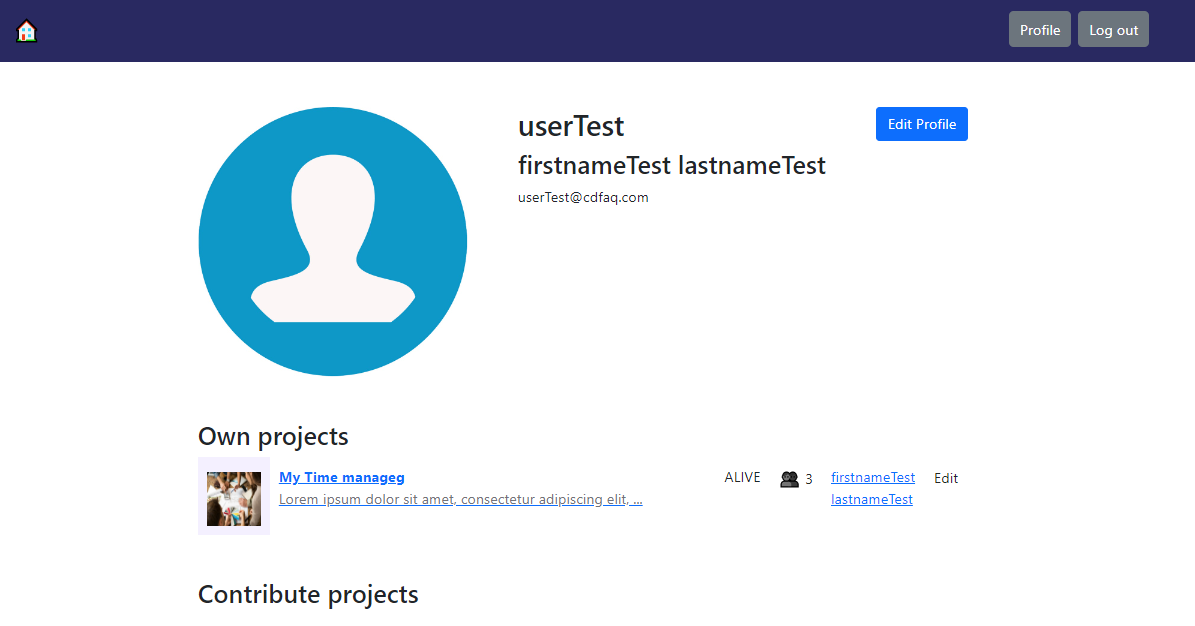


Рисунок 4.31 – Профиль пользователя

При переходе на страницу мы видим общую информацию о них, а также список проектов, которые они владеют и в которых участвуют.

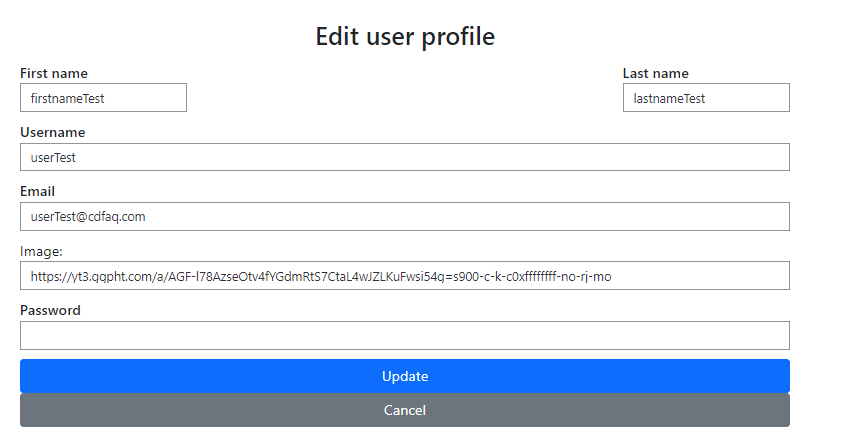


Рисунок 4.32 – Редактирование профиля

Кроме того, если мы зашли на свою страницу, мы можем редактировать информацию о себе. На этой странице можно ввести новое данные или обновить их.

## 4.3 [Разработка алгоритмов реализации вариантов использования](#_Toc474749002)

При решении общей задачи, возникали более мелкие, для решения которых необходимо было использовать и разрабатывать алгоритмы. Несколько таких примеров будет показано ниже.

1. Удаление колонки с привязанными к ней задачами

public Project delete(Long id, RedirectAttributes ra) {

Column column = read(id);

Project project = column.getProject();

if (!column.getIssues().isEmpty()) {

ra.addFlashAttribute(MESSAGE, "Cannot remove the column. It contains issues.");

}

try {

project.getColumns().remove(column);

projectService.update(project);

column.setProject(null);

delete(column.getId());

ra.addFlashAttribute(MESSAGE, "Removing is successful");

} catch (Exception e) {

e.printStackTrace();

ra.addFlashAttribute(MESSAGE, "Removing failed");

}

return project;

}

1. Отображения информации с учетом авторизации пользователя

<!DOCTYPE html>

<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>

<title>List of Issues</title>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/dragula/3.7.2/dragula.min.css"/>

<script src="https://cdnjs.cloudflare.com/ajax/libs/dragula/3.7.2/dragula.min.js"></script>

<div th:replace="~{commons :: head}"></div>

</head>

<body>

<div th:replace="~{commons :: header}"></div>

<section class="mt-5">

<div class="container">

<div class="row">

<!-- left menu -->

<div th:replace="~{commons :: left-menu}"></div>

<!-- issues list -->

<div class="col-8 m-auto">

<h1>List of Issues</h1>

<h3 th:if="${message != null}" th:text="${message}">Message</h3>

<a th:href="@{/issue/{id}/edit(id=${project.id})}"

th:classappend="${isOwner != null and isOwner == false} ? 'hidden' : ''"

class="btn btn-primary mb-3">Create new Issue</a>

<div class="row">

<div class="col-md-4" th:each="column : ${project.getColumns()}" th:attr="data-column-id=${column.id}">

<h4 class="text-center" th:text="${column.name}"></h4>

<div class="issue-container border">

<div class="issue-card p-2 mb-2" th:each="issue : ${column.getIssues()}" th:attr="data-issue-id=${issue.id}">

<a th:href="'/issue/get/' + ${issue.id}" th:text="${issue.name}"></a>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</section>

<script>

document.addEventListener('DOMContentLoaded', function () {

var issueContainers = Array.prototype.slice.call(document.querySelectorAll('.issue-container'));

var drake = dragula(issueContainers, {

revertOnSpill: true,

removeOnSpill: false

});

drake.on('drop', function (el, target, source, sibling) {

var issueId = el.getAttribute('data-issue-id');

var newColumnId = target.closest('[data-column-id]').getAttribute('data-column-id');

var xhr = new XMLHttpRequest();

var formData = new FormData();

formData.append('statusId', newColumnId);

xhr.open('POST', '/issue/' + issueId + '/status', true);

xhr.setRequestHeader('X-Requested-With', 'XMLHttpRequest');

xhr.onload = function () {

if (xhr.status === 200) {

console.log('Status updated');

} else {

console.error('An error occurred while updating the status');

}

};

xhr.send(formData);

});

});

</script>

<div th:replace="~{commons :: footer}"></div>

</body>

</html>

## 4.3 Модульное тестирование алгоритмов реализации вариантов использования

Модульное тестирование, или юнит-тестирование - это процесс в программировании, который направлен на проверку отдельных небольших частей приложения, также называемых атомарными, которые можно исследовать изолированно от других подобных частей. При выполнении данного тестирования могут проверяться как отдельные функции или методы классов, так и сами классы, взаимодействие классов, небольшие библиотеки и отдельные части приложения [16].

Довольно часто данный вид тестирования реализуется с использованием специальных технологий и инструментальных средств автоматизации тестирования, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов. Таким образом, оценивая каждый элемент отдельно и подтверждая правильность его работы, установить проблему значительно проще чем, если бы элемент был частью системы.

Для проверки работы спроектированного ПС разработаны тесты, с помощью которых можно оценить корректную работу веб-сервиса.

В таблице 2 приведены результаты тестирования для формы авторизации пользователя, которая отображается на главной странице веб-сервиса, были рассмотрены более вероятные сценарии поведение пользователей.

Таблица 2 – Набор тестов авторизации в веб-сервисе

| **Место проведения теста** | **Содержание теста** | **Ожидаемый результат** | **Отметка о прохождении теста** |
| --- | --- | --- | --- |
| Страница входа с адресом /login | Введены некорректные данные в любое из полей, нажата кнопка «LogIn». | Сообщение пользователю об ошибке, очистка заполненных полей. | Да |
| Страница входа с адресом /login | Введен корректный логин и пароль, нажата кнопка «LogIn». | Отображение главной страницы сайта, отображение соответствующих функциональных возможностей. | Да |

В таблице 3 приведены результаты тестирования возможностей администратора веб-сервиса. Проверен функционал добавления нового мероприятия и редактирования уже существующей информации о мероприятии. Рассмотрены основные сценарии использования администратором.

Таблица 3 – Набор тестов для проверки возможностей владельца проекта

| **Место проведения теста** | **Содержание теста** | **Ожидаемый результат** | **Отметка о прохождении теста** |
| --- | --- | --- | --- |
| Страница «Create project», добавление записи | Все поля корректно заполнены, нажата кнопка «Save». | Переход на страницу с списком проектов, где есть только что добавленный | Да |
| Страница «Create project», добавление записи | Любое из полей не заполнено, нажата кнопка «Отправить». | Отображение информации об ошибке, ожидание заполнения полей | Да |
| Страница «Project info» | Ожидание вывода информации. | Отображение информации только выбранного тарифа. | Да |

Таблица 4 – Набор тестов для проверки возможностей пользователя

| **Место проведения теста** | **Содержание теста** | **Ожидаемый результат** | **Отметка о прохождении теста** |
| --- | --- | --- | --- |
| Главная страница | Выбрать проект для просмотра, пользователь которого является владельцем | Переход на страницу подробностей. В верху страницы видны кнопки для управления проектом | Да |
| Главная страница, страница подробностей | Выбрать проект для просмотра, пользователь которого не является владельцем | Переход на страницу подробностей. В верху страницы не видны кнопки для управления проектом | Да |

В таблице 5 приведены результаты тестирования для формы авторизации пользователя, которая отображается на главной странице веб-сервиса, были рассмотрены более вероятные сценарии поведение пользователей.

Таблица 5 – Набор тестов для проверки возможностей владельца проекта

| **Место проведения теста** | **Содержание теста** | **Ожидаемый результат** | **Отметка о прохождении теста** |
| --- | --- | --- | --- |
| Страница проекта | Добавление нового участника в проект | Новый участник добавлен к проекту, отображение его имени в списке участников | Да |
| Страница проекта | Редактирование информации о проекте и сохранение изменений | Отображение обновленной информации о проекте на странице проекта | Да |
| Страница проекта | Удаление проекта | Проект удален, перенаправление на список проектов пользователя | Да |
| Страница проекта | Создание, редактирование и удаление задач, меток и столбцов | Созданные, измененные или удаленные элементы отображаются соответствующим образом на странице проекта | Да |
| Страница проекта | Перемещение задач между столбцами | Задача перемещена в выбранный столбец, отображение задачи в новом столбце | Да |
| Страница проекта | Просмотр списка участников проекта | Отображение списка участников проекта | Да |
| Страница проекта | Назначение другого пользователя владельцем проекта | Уведомление о передаче прав владельца проекта, новый владелец проекта отображается в списке участников с соответствующей ролью | Да |

В таблице 6 приведены результаты тестирования возможностей пользователя веб-сервиса. Рассмотрены основные сценарии использования возможностей веб-сервиса пользователем.

Таблица 6 – Набор тестов отображения информации

|  |  |  |  |
| --- | --- | --- | --- |
| **Место проведения теста** | **Содержание теста** | **Ожидаемый результат** | **Отметка о прохождении теста** |
| Все страницы | Перейти на любую из вкладок в произвольном порядке. | Корректное отображение информации, отображение того, чего ожидает пользователь. | Да |
| Все вкладки | Увеличение масштаба отображения средствами браузера. | Появление скроллов справа и снизу, корректное отображение информации. | Да |
| Все вкладки | Обновление страницы используя средства браузера. | Состояние веб-сервиса до обновления и после не изменилось. | Да |
| Меню браузера | Нажатие функциональных кнопок браузера «Назад» и «Вперед» в произвольный момент работы веб-приложения. | Корректное поведение веб-сервиса. Отображение информации. | Да |

Были проведены экспериментальные проверки на реальных данных. Разработанный веб-сервис полностью реализует все поставленные задачи. Весь реализованный функционал работает корректно, что было подтверждено как наборами тестов, так и визуальным тестированием.

# 5 Системное тестирование

Системное тестирование — это тестирование программного обеспечения, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования чёрного ящика, и, тем самым, не требует знаний о внутреннем устройстве системы. Тестирование бывает:

* функциональное тестирование;
* тестирование пользовательского интерфейса;
* тестирование совместимости;
* тестирование безопасности;
* тестирование производительности;
* автоматическое тестирование;
* интеграционное тестирование.

Испытания системы имеют две основные цели:

* показать разработчику и клиенту, что программное обеспечение отвечает заявленным требованиям. С точки зрения клиента это означает, что для каждой функциональности, желаемой с его стороны и записанной в документе требований, проведен как минимум в один тест (как правило, конечно, больше). В случае общедоступного программного обеспечения то, что в программном обеспечении протестированы все заданные основные свойства. Соответствующий данной цели тест называют вариацией (проверкой достоверности). Успешное вариация указывает, что система работает как надо;
* найти ситуации, когда программное обеспечение ведет себя ошибочно, нежелательно или не соответствует спецификации. Следовательно, поиск ошибок в этом смысле предназначен для того, чтобы ликвидировать нежелательное поведение системы, как, например, крах системы, нежелательное взаимодействие с другими системами, неправильные расчеты, поврежденные данные. Выполняющее эту задачу тестирование именуется тестированием дефектов (defect testing). Здесь являющийся успешным тест показывает действие ошибки системы, или, другими словами, находит в системе ошибку (к дальнейшему исправлению которой, и приступают).

## 5.1 Функциональное тестирование

Функциональное тестирование — это тестирование программного обеспечения в целях проверки реализуемости функциональных требований, то есть способности программного обеспечения в определённых условиях решать задачи, нужные пользователям [17]. Функциональные требования определяют, что именно делает программное обеспечение, какие задачи оно решает.

Функциональные требования включают в себя:

* функциональная пригодность;
* точность;
* способность к взаимодействию;
* соответствие стандартам и правилам;
* защищённость.

Тестирование и отладка программы являются наиважнейшими этапами разработки любых программных продуктов. Цель этого этапа - проверка правильности и точности реализации функций, выполнение которых возлагается на данный программный продукт. В случае выявления некоторых неточностей и ошибок необходимо проведение работ по их исправлению и доработке программного продукта до требуемого уровня.

На основе функций, которые должны быть протестированы в разработанной информационной обучающей системе, составлен чек-лист (checklist), приведенный в таблице 5.1.

Таблица 5.1 - Чек-лист

|  |  |  |  |
| --- | --- | --- | --- |
| **Тестируемый модуль** | | **Тестируемая функция** | **Результат** |
| Вход в систему | | Вход в систему | Выполнено успешно |
| Выход из системы | | Выход из системы | Выполнено успешно |
| Просмотр проектов | | Отображение информации и зависимых списков | Выполнено успешно |
| Просмотр задач проекта | Отображение информации других таблиц на основе выбранного значения | | Выполнено успешно |
| Работа с колонками | Добавление, редактирование и удаление данных с выводом информации подчиненной таблицы | | Выполнено успешно |
| Работа с профилем | Добавление, редактирование и удаление данных с выводом информации подчиненной таблицы | | Выполнено успешно |

В качестве тестирования программного продукта был выбран тест-кейс (**Test Case**).

**Test Case** - это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

Тестирование производилось на ОС «Windows 10».

## 5.2 Оценка безопасности

Обеспечение безопасности информационных систем представляет собой ряд мероприятий, направленных на предотвращение несанкционированных воздействий на защищаемую информацию, а также её утечки. Поскольку приложение построено на базе фреймворка Spring, вопросы безопасности берет на себя Spring Security.

Spring Security это Java/JavaEE framework, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework. Проект был начат Беном Алексом (Ben Alex) в конце 2003 года под именем «Acegi Security», первый релиз вышел в 2004 году. Впоследствии проект был поглощён Spring'ом и стал его официальным дочерним проектом. Впервые публично представлен под новым именем Spring Security 2.0.0 в апреле 2008 года.

Ключевые объекты контекста Spring Security:

SecurityContextHolder, в нем содержится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе(Principal) работающем в настоящее время с приложением. По умолчанию SecurityContextHolder используетThreadLocal для хранения такой информации, что означает, что контекст безопасности всегда доступен для методов исполняющихся в том же самом потоке. Для того что бы изменить стратегию хранения этой информации можно воспользоваться статическим методом класса SecurityContextHolder.setStrategyName(String strategy). Более подробно SecurityContextHolder.

SecurityContext, содержит объект Authentication и в случае необходимости информацию системы безопасности, связанную с запросом от пользователя.

GrantedAuthority отражает разрешения выданные пользователю в масштабе всего приложения, такие разрешения (как правило называются «роли»), например ROLE\_ANONYMOUS, ROLE\_USER, ROLE\_ADMIN.

UserDetails предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности. Объект UserDetailsсодержит имя пользователя, пароль, флаги: isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled и Collection — прав (ролей) пользователя.

UserDetailsService, используется чтобы создать UserDetails объект путем реализации единственного метода этого интерфейса.

UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;

Позволяет получить из источника данных объект пользователя и сформировать из него объект UserDetails который будет использоваться контекстом Spring Security.

**5.3 Тестирование производительности**

Тестирование производительности**-** это тестирование, которое проводится для определения скорости работы системы или её части при заданной нагрузке. Тестирование производительности стремится учесть производительность уже на стадии проектирования и моделирования и системы, до начала основной стадии разработки.

Тестирование производительности служит таким типичным целям:

* для демонстрации того, что система удовлетворяет критериям производительности;
* для определения производительность какой из двух или нескольких систем лучше;
* для определения, какой элемент нагрузки или часть системы приводит к снижению производительности.
* Для оценки времени загрузки страниц системы был использован специализированный веб-сервис «https://developers.google.com». PageSpeed Insights анализирует содержание веб-страницы и предлагает решения, которые позволят ускорить ее загрузку.

В процессе тестирования были произведены замеры:

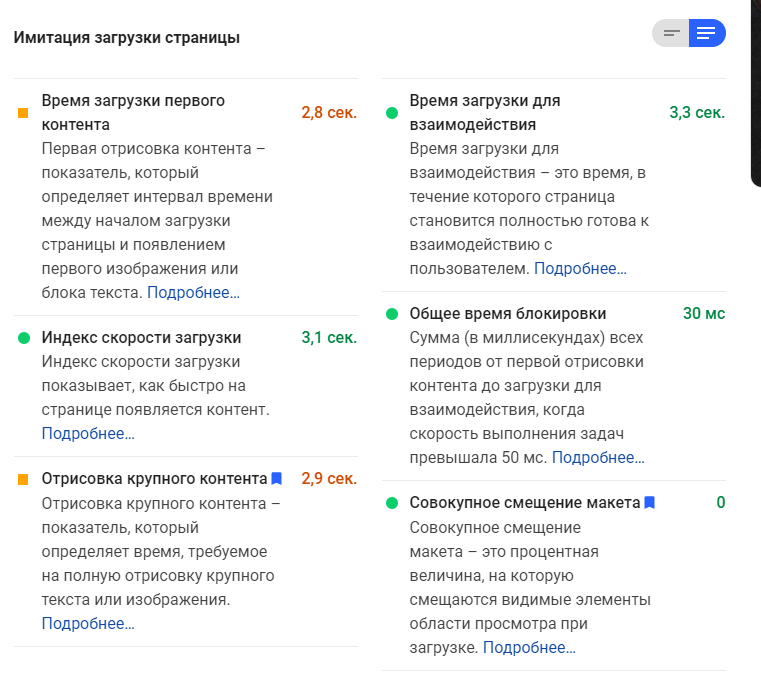
****

Рисунок 5.1 - Тестирование

# 6 Экономическая часть

## 6.1 Расчет трудоемкости разработки программного обеспечения

Экономическая целесообразность разработки и внедрения программного обеспечения определяется экономическим эффектом, который будет получен произво­дителями при их реализации и потребителями при их использовании. По величине ожидаемого экономического эффекта принимается решение о целесообразности ин­вестиций в разработку того или иного программного продукта. По характеру объекта вложений инвестиции в разработку программного обеспечения относят к интеллектуальным инвестициям.

При создании программного продукта важно оценить его себестоимости (затраты на разработку).

## 6.2 Определение стоимости разработки программного обеспечения

Затраты времени на разработку ПО могут также определяться эмпирическим путем. В этом случае затраты времени могут включать:

- затраты труда на подготовку и описание задачи – *tоп*;

- затраты труда на исследование алгоритма решения задачи – *tис*;

- затраты труда на разработку алгоритма (блок-схем) – tал;

- затраты труда на программирование алгоритма по блок-схеме – *tпр*;

- затраты труда на отладку программы – *tотл*;

- затраты труда на подготовку документов по задаче состоят из затрат труда на подготовку рукописей и времени на оформление документов – *tд.*

Суммарные затраты труда рассчитываются как сумма составных затрат труда по формуле (6.1):

. (6.1)

Расчет суммарных затрат времени представлен в таблице 6.1.

Таблица 6.1 - Ориентировочное распределение затрат времени

|  |  |  |
| --- | --- | --- |
| Вид работ | Трудоемкость в часах | |
| Всего, человеко-часов | в том числе машинное время, машино-часов |
| Подготовку и описание задачи | 14 | - |
| Исследование алгоритма решения задачи | 30 | - |
| Разработка алгоритма | 32 | - |
| Программирование алгоритма | 240 | 240 |
| Отладка программы | 62 | 62 |
| Подготовка и оформление документов | 36 | 30 |
| Итого: | =414 | Σ*tмаш*=332 |

### 

6.2.1 Расчет затрат на разработку программного обеспечения

Затраты на оплату (*ЗОТ*) труда разработчика ПО включают затраты на оплату труда и отчисления от фонда заработной платы.

Затраты на оплату трударазработчика ПО рассчитывается в руб. по формуле (6.2):

(6.2)

где – суммарные затраты труда на разработку и сопровождение ПО, (таблица 6.1) ч.;

*КЧР* – среднемесячная расчетная норма рабочего времени (среднее количество часов работы в месяц), ч.;

– месячная заработная плата инженера-программиста, руб.

Согласно данным Министерства труда и социальной защиты Республики Беларусь среднемесячная расчетная норма рабочего времени при пятидневной рабочей неделе равна 169,3 ч.

Месячная заработная плата инженера-программиста включает:

а) оклад;

б) стимулирующие выплаты (надбавки и премии);

в) компенсирующие выплаты (доплаты), которые не учитываются при расчете заработной платы в условиях дипломного проекта.

(6.3)

где – оклад работника, руб.;

– надбавка за работу в бюджетной организации, руб.;

– надбавка за стаж работы в бюджетной организации, руб.;

– надбавка за контрактную форму найма, руб.;

– ежемесячная премия, руб.

С 1 января 2020 года в Беларуси появилась новая система начисления зарплаты работникам бюджетной сферы. Правовое основание для этого – Указ Президента № 27 «Об оплате труда работников бюджетных организаций».

Вместо 27 тарифных разрядов для всех категорий трудящихся, вводится 18 разрядов для бюджетников. Основой для расчета будет не ставка 1 разряда, а базовая тарифная ставка, которая предполагается близкой по сумме к бюджету прожиточного минимума.

Базовая ставка – определяемая Правительством величина, на основании которой, через систему коэффициентов и доплат, будет формироваться заработная плата работников бюджетной сферы.

В соответствии с данным постановление постановлением расчет оплаты труда инженера-программиста, работающего в бюджетной организации, производится исходя из 4 разряда работ (тарифный коэффициент составляет 1,21). Базовая ставка принятая приказом на предприятии в году составляет 185 руб.

Оклад рассчитывается по формуле:

(6.4)

где БС – базовая ставка работников бюджетных организаций, руб.;

ТК – тарифный коэффициент, соответствующий разряду работ разработчика ПО.

Подставляя данные в формулу (6.4) получаем:

Стимулирующие выплаты:

1. Надбавка за работу в бюджетной организации (70% от оклада):

(6.5)

Подставляя данные в формулу (6.5) получаем:

1. Надбавка за стаж работы в бюджетной организации при стаже работы до 5 лет устанавливается в размере 10% от базовой ставки.

(6.6)

Подставляя данные в формулу (6.6) получаем:

1. Надбавка за контрактную форму найма (до 40% от оклада):

(6.7)

Подставляя данные в формулу (6.7) получаем:

1. Премия ежемесячная (5% от оклада):

(6.8)

Подставляя данные в формулу (6.8) получаем:

На основе приведенных выше данных подставляя данные в формулу (6.3) мы получим месечную зароботную плату:

Рассчитаем затраты на оплату трударазработчика ПО по формуле (6.2):

Отчисления от фонда оплаты труда включают:

* отчисления в Фонд социальной защиты населения – 34% от ФЗП;
* страхование нанимателя от несчастных случаев на производстве – 0,6% от ФЗП.

Отчисления от фонда оплаты труда рассчитываются по формуле (6.9):

(6.9)

где – отчисления в Фонд социальной защиты населения (ставка отчислений составляет 34% от всех выплат работнику), руб.;

– страхование нанимателя от несчастных случаев на производстве и профзаболеваний (ставка отчислений составляет 0.6% от всех выплат работнику), руб.

Подставляя данные в формулу (6.9) получаем:

Затраты на оплату труда с учетом отчислений рассчитываются по формуле:

(6.10)

Подставляя данные в формулу (6.10) получаем:

6.2.2 Эксплуатационные затраты на оборудование

Стоимость оборудования хоть и не включается в себестоимость разработки программного обеспечения, но все же используется при расчете отдельных статей расходов. При написании программы в качестве оборудования предполагается использовать персональный компьютер, стоимость которого составляет: .

Суммарная стоимость эксплуатационных затрат рассчитывается по формуле (6.11):

(6.11)

где – годовые затраты на техническое обслуживание и ремонт оборудования, руб.;

– годовая стоимость электроэнергии, руб.;

– годовые амортизационные отчисления, руб.

Затраты на техническое обслуживание и ремонт (*СТО*) принимаются в размере 3-5% от стоимости оборудования, например:

(6.12)

Подставляя данные в формулу (6.12) получаем:

Амортизационные отчисления, процесс постепенного перенесения стоимости средств труда по мере их физического и морального износа на стоимость производимых с их помощью продукции в целях аккумуляции денежных средств для последующего полного восстановления. Годовые амортизационные отчисления производятся по установленным нормам амортизации, выражаются, в процентах к стоимости оборудования и рассчитываются по формуле (6.13):

(6.13)

где – стоимость компьютера;

– норма амортизации, которая рассчитывается по формуле (6.14):

(6.14)

где – нормативный срок службы (для персонального компьютера = 3 года).

Подставляя данные в формулы (6.14) и (6.13) получаем:

Годовая стоимость электроэнергии вычисляется по формуле (6.15):

(6.15)

где – мощность компьютера, КВт (примем равным 0.4 кВт);

– коэффициент загрузки, учитывающий использование оборудования по времени (0.9);

– стоимость 1 кВтч электроэнергии (0,27274 руб./кВтч для бюджетных организаций);

– коэффициент, учитывающий потери в сети (Kc=1,05);

– эффективный фонд рабочего времени, рассчитывается по формуле (6.16):

(6.16)

где = 255 – количество рабочих дней в 2020 году при пятидневной рабочей неделе (данные Министерства труда и соцзащиты РБ);

*d* = 7,97 – продолжительность рабочего дня, ч.;

*f* = 2% – планируемый процент времени на ремонт оборудования.

Подставляя данные в формулу (6.16) получаем:

Подставляя данные в формулу (6.15) получаем:

Однако, полученная стоимость эксплуатационных затрат – это значения годовых расходов. Необходимо их скорректировать в соответствии с временным коэффициентом (так как оборудование будет эксплуатироваться не весь год, а только в течение времени ), который определяется исходя из суммарных годовых эксплуатационных затрат, которые рассчитываются по формуле (6.17):

(6.17)

где – суммарная годовая стоимость эксплуатационных затрат, высчитываемая по формуле (6.18);

– эффективный фонд рабочего времени, (формула (6.16));

– общее время использования оборудования (из таблицы 6.1 – 343 ч.).

(6.18)

Подставляя данные в формулу (6.18) получаем:

Подставляя данные в формулу (6.17) получаем:

6.2.3 Затраты на материалы

Затраты на материалы включают расходы на бумагу, канцелярские принадлежности и другие материалы, используемые при разработке ПО.

Затраты на печать пояснительной записки к дипломному проекту составили – 15 бел. руб.

6.2.4 Накладные расходы

Накладные расходы, связанные с управлением, организационными расходами и прочими дополнительными затратами, составляют 30% от фонда заработной платы и вычисляются по формуле (6.19):

(6.19)

Подставляя данные в формулу (6.19) получаем:

Суммарные затраты на разработку программного обеспечения считаются как сумма фонда заработной платы и отчислений от него, эксплуатационных затрат, затрат на материалы, накладных расходов.

Себестоимость разработки программного обеспечения представлена в таблице 6.2.

Таблица 6.2 – Стоимость программного обеспечения

|  |  |  |
| --- | --- | --- |
| Наименование статей затрат | Обозначение | Сумма, руб. |
| 1. Затраты на материалы |  | 15 |
| 2. Эксплуатационные затраты |  | 70,6 |
| 3. Основная заработная плата |  | 1222,1 |
| 4. Отчисления от заработной платы |  | 422,8 |
| 5. Накладные расходы |  | 366,6 |
| 6. Полная себестоимость разработки и сопровождения ПО, *Сполн* | п.1+п.2+п.3+п.4+п.5 | 2097,1 |

### 6.3 Расчёт экономического эффекта от внедрения программного продукта

6.3.1 Экономический эффект у разработчика программного обеспечения

Заказчик оплачивает разработчику всю сумму расходов по проекту (полная себестоимость ПО из таблицы 6.2) с учетом прибыли разработчика и налога на добавленную стоимость с учетом качества, потребительских свойств продукции (ПО) и конъюнктуры рынка. Таким образом, в дипломном проекте отпускная цена программного обеспечения, представляет собой не цену за единицу продукции, а цену проекта вместе с его исходными кодами и документацией, за которую его можно продать и получить определенную выгоду. Прогнозируемая отпускная цена ПО (*ЦПО*) с учетом НДС рассчитывается по формуле (6.20):

, (6.20)

где *Сполн*– полная (плановая) себестоимость ПО, руб.,

*П* – прибыль разработчика ПО, руб.,

*СТНДС* – ставка налога на добавленную стоимость (=20%), в %.

Прибыль закладывается в цену исходя из уровня рентабельности (устанавливается студентом самостоятельно), расчет производится по формуле (6.21)

, (6.21)

где R– уровень рентабельности, % (в рамках дипломного проекта рекомендуемый уровень рентабельности ≈ 20%).

Подставляя данные в формулу (6.21) получаем:

Подставляя данные в формулу (6.20) получаем:

Ввиду того, что программное обеспечение разрабатывается для одного объекта, в качестве экономического эффекта разработчика от реализованного программного обеспечения можно рассматривать чистую прибыль (*ЧП*), которая рассчитывается по формуле (6.22):

, (6.22)

где *СТП* – ставка налогообложения прибыли равная 18%.

Подставляя данные в формулу (6.22) получаем:

Таким образом, разработчик программного обеспечения может продать заказчику программное обеспечение по цене 3019 рублей 8 копеек с учетом НДС, что покроет затраты на разработку ПО, и обеспечит получение чистой прибыли от реализации в размере 343 рубля 9 копеек.

6.3.2 Экономический эффект от использования программного обеспечения у пользователя (заказчика)

Данный программный продукт отличается от аналогичных разработок конкурентов, следующими параметрами:

а) цена меньше, чем у конкурентов;

б) это универсальный программный продукт для любого пользователя;

в) хранения данных нас устройстве и отправка их в облако;

г) интуитивно понятный интерфейс.

Системные требования для данного программного продукта для ОС Windows:

- Windows 7/8/10;

- процессор – «Intel Pentium» (2 ядра по 2,0 ГГц) или аналог AMD;

- жёсткий диск – HDD 250 Гб;

- видеокарта – Nvidia или ATI Radeon 1024Мб DDR3;

- ОЗУ – DDR3 4000 Mб 1333 МГц.

Согласно различным источникам, текущая рыночная цена на подобный

программный продукт (лицензия на год) в Республике Беларусь колеблется в диапазоне от 3 000 бел. руб. до 4 000 бел. руб. Такая разбежка в ценовом диапазоне объясняется различным сроком полезного использования ПО (лицензия на год или на два).

Таким образом, рассчитанная отпускная цена на программный продукт, разрабатываемый в рамках данного дипломного проекта, является конкурентоспособной.

# 7 Охрана труда

Согласно Закону об охране труда от 23 июня 2008 г. № 356 *-* З (в ред. Закона Республики Беларусь от 12.07.2013 N 61-З) дается следующее определение понятию охраны труда:

Охрана труда - система обеспечения безопасности жизни и здоровья работающих в процессе трудовой деятельности, включающая правовые, социально-экономические, организационные, технические, психофизиологические, санитарно-противоэпидемические, лечебно- профилактические, реабилитационные и иные мероприятия и средства.

Требования по охране труда - нормативные предписания, направленные на сохранение жизни, здоровья и работоспособности работников в процессе их трудовой деятельности, содержащиеся в нормативных правовых актах, в том числе технических нормативных правовых актах.

Систему законодательных актов, регулирующих вопросы охраны труда в республике, составляют Конституция Республики Беларусь, Концепция государственного управления охраной труда Республики Беларусь, Трудовой кодекс Республики Беларусь, Законы Республики Беларусь «Об охране труда», «Об основах государственного социального страхования», «О пенсионном обеспечении», «О санитарно-эпидемическом благополучии населения», «О техническом нормировании и стандартизации», «О пожарной безопасности», «О промышленной безопасно­сти», «О радиационной безопасности на­селения», «О защите населения и территорий от чрезвычайных ситуаций при­родного и техногенного характера», «О здравоохранении», «О предприяти­ях» и др. НПА, ТНПА, ЛНПА.

Охрана труда имеет большое социальное, экономическое и правовое значение.

Социальное значение охраны труда заключается в следующем:

* сохранение работоспособности и трудового долголетия работника;
* охрана жизни и здоровья работника от возможных воздействий вредных условий производства;
* охрана труда способствует гуманизации труда, содействует его культурно-техническому росту;

Экономическое значение охраны труда заключается в следующем:

* способствует росту производительности труда работников, росту производства и экономики;
* способствует экономии фонда социального страхования и сокращению потерь рабочего времени.

Работать с разработанным веб-сервисом планируется в жилом помещении, без предъявления каких-либо специальных требований. Специальной службы по охране труда не предусмотрено.

Рассмотрим характеристику объекта с точки зрения охраны труда на примере администратора веб-сервиса.

Проведем оценку факторов производственной среды, тяжести и напряжённости трудового процесса (таблицы 7.1-7.3).

Таблица 7.1 – Оценка факторов производственной среды

|  |  |  |
| --- | --- | --- |
| Факторы и показатели производственной среды | Гигиенические нормативы (ПДК, ПДУ) | Фактические величины |
| 1 | 2 | 3 |
| 2.4 Шум, дБА, дБ | 60 | 40 |
| 2.9 Электромагнитные поля и неионизирующие излучения |  |  |

Окончание таблицы 7.1

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Напряженность электрического поля, В/м |  |  |
| – от 5 Гц до 2 кГц | 25 | 21 |
| – от 2 кГц до 400 кГц | 2,5 | 0,6 |
| Плотность магнитного потока, нТл |  |  |
| – от 5 Гц до 2 кГц | 250 | 210 |
| – от 2 кГц до 400 кГц | 25 | 6 |
| Электростатические поля, кВт/м | 15 | 5,6 |
| 2.11 Микроклимат |  |  |
| 2.11.1 Температура воздуха, оС | 18-24 | 20 |
| 2.11.2 Относительная влажность, % | не более 60 | 50 |
| 2.11.3 Скорость движения воздуха, м/с | не более 0,3 | 0,1 |
| 2.12 Освещенность, лк | 300 | 560 |

Таблица 7.2 – Оценка тяжести трудового процесса

|  |  |
| --- | --- |
| Показатели тяжести трудового процесса | Фактическое значение показателя |
| 1 | 2 |
| 3.1 Физическая динамическая нагрузка, кгм |  |
| 3.1.1 Региональная нагрузка при перемещении груза на расстояние до 1 м | До 2 500 |
| 3.1.2 Общая нагрузка при перемещении груза на расстояние: - от 1 до 5 м | До 12 500 |
| - более 5 м |  |
| 3.2 Масса поднимаемого и перемещаемого груза вручную, кг | До 2 |
| 3.2.1 Подъем и перемещение тяжести при чередовании с другой работой | 3-12,5 |
| 3.2.2 Подъем и перемещение тяжестей постоянно в течение рабочей смены | 10 |

Окончание таблицы 7.2

|  |  |
| --- | --- |
| 1 | 2 |
| 3.2.3 Суммарная масса грузов, перемещаемых в течение каждого часа смены:  - с рабочей поверхности | До 350 |
| - с пола |  |
| 3.3 Стереотипные рабочие движения, количество за смену |  |
| 3.3.1 При локальной нагрузке | 12 000 |
| 3.3.2 При региональной нагрузке |  |
| 3.4 Статическая нагрузка, кг (силы) · с |  |
| 3.4.1 Одной рукой | До 36 000 |
| 3.4.2 Двумя руками | 20 000 |
| 3.4.3 С участием мышц корпуса, ног |  |
| 3.5 Рабочая поза (стоя) | Стоя 20 % |
| 3.6 Наклоны корпуса | 10 |
| 3.7 Перемещения в пространстве, обусловленные технологическим процессом, км |  |
| 3.7.1 По горизонтали | До 4 |
| 3.7.2 По вертикали |  |

Таблица 7.3 – Оценка напряженности трудового процесса

|  |  |
| --- | --- |
| Показатели напряженности трудового процесса | Характеристика показателей в соответствии с гигиеническими критериями |
| 1 | 2 |
| 4.1 Интеллектуальные нагрузки |  |
| 4.1.1 Содержание работы | Решение задач по инструкции |
| 4.1.2 Восприятие сигналов (информации) и их оценка | Восприятие сигналов, но не требуется коррекция действий |
| 4.1.3 Распределение функций по степени сложности задания | Обработка и выполнение задания |

Продолжение таблицы 7.3

|  |  |
| --- | --- |
| 1 | 2 |
| 4.1.4 Характер выполняемой работы | Работа по установленному регламенту |
| 4.2 Сенсорные нагрузки |  |
| 4.2.1 Длительность сосредоточенного наблюдения (в % от времени смены) | До 25 |
| 4.2.2 Плотность сигналов (световых, звуковых) и сообщений в среднем за 1 час работы | 60 |
| 4.2.3 Число производственных объектов одновременного наблюдения | 1 |
| 4.2.4 Размер объекта различения (при расстоянии от глаз работающего до объекта различения не более 0,5 м) в мм при длительности сосредоточенного наблюдения (% времени смены) | 0,3-0,5 мм-до 30%  более 0,5 мм-до 70% |
| 4.2.5 Работа с оптическими приборами (микроскопы, лупы и т.п.) при длительности сосредоточенного наблюдения (% времени смены) |  |
| 4.2.6 Наблюдение за экранами видеотерминалов (часов в смену): - при буквенно-цифровом типе отображения информации; | 5 |
| - при графическом типе отображения | До 3 |
| 4.2.7 Нагрузка на слуховой анализатор (при производственной необходимости восприятия речи или дифференцированных сигналов) | Разборчивость слов и сигналов от 75% до 50%. Помехи присутствуют |
| 4.2.8 Нагрузка на голосовой аппарат (суммарное количество часов, наговариваемое в неделю) |  |
| 4.3 Эмоциональные нагрузки |  |
| 4.3.1 Степень ответственности за результат собственной деятельности. Значимость ошибок | Ответственность за качество работы, влечёт дополнительные усилия со стороны руководства |

Окончание таблицы 7.3

|  |  |
| --- | --- |
| 1 | 2 |
| 4.3.2 Степень риска для собственной жизни | Исключена |
| 4.3.3 Степень ответственности за безопасность других лиц | Исключена |
| 4.4 Монотонность нагрузок |  |
| 4.4.1 Число элементов (приемов), необходимых для реализации простого задания или в многократно повторяющихся операциях | 8 |
| 4.4.2 Продолжительность выполнения простых производственных заданий или повторяющихся операций, с | 25-100 |
| 4.4.3. монотонность производственной обстановки (время пассивного наблюдения за ходом техпроцесса в % от времени смены) | 76-80 |
| 4.5 Режим работы |  |
| 4.5.1 Сменность работы | Односменная |

Из таблиц 7.2 – 7.3 следует, что администратор веб-сервиса подвержен всем видам нагрузок: эмоциональной, сенсорной и интеллектуальной.

На основании представленных данных разработаем карту рисков рабочего места администратора.

Для оценки рисков применяем классический метод. Оценка рисков рассчитывается по формуле (7.1):

*R = P × S,* (7.1)

где R – риск, балл; P – вероятность возникновения опасности, балл; S – серьезность последствий воздействия опасности, балл.

Исходя из значений P и S, определяем категорию риска. Категории рисков подразделяются на следующие: низкие (R < 6); умеренные (6 ≤ R ≤ 12); высокие (R > 12). Риски, отнесенные к категории «низкие» считаются допустимыми и управляемыми в соответствии с существующими в организации мерами (имеются в наличии необходимые процедуры и инструкции, оборудование поддерживается в технически исправном состоянии, своевременно проводится обучение, инструктаж и проверка знаний работников). Риски, отнесенные к категориям «умеренные» и «высокие» считаются недопустимыми и требуют разработки мер по управлению ими. Карта опасностей и рисков представлена в таблице 7.4.

Таблица 7.4 – Карта управления (умеренными) рисками

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Профессия, должность | Вид деятельности | Идентификационная  опасность | Серьёзность послед­ствий возникновения опасности, S | Вероятность возник­новения опасности, Р | Риск, R | Осуществляемые меры управления | Рекомендуе-мые действия | Срок исполнения |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Администратор веб-сервиса   * Дана характеристика объекта с точки зрения охраны труда: условия труда администратора интернет-магазина относятся к допустимым условиям (2 класс), которые характеризуются такими уровнями факторов среды и трудового процесса, которые не превышают установленных гигиенических нормативов для рабочих мест, а возможные изменения функционального состояния организма, возникающие под их воздействием, восстанавливаются во время регламентированного отдыха или к началу следующей смены и не оказывают неблагоприятного действия на состояние здоровья работников в ближайшем и отдаленном периоде. * Разработана карта рисков для администратора интернет-магазина. * Произведена оценка организации охраны труда, производственной санитарии, промышленной и пожарной безопасности.   Администратор веб-сервиса | трудовая | Нервно-психические перегрузки | 2 | 4 | 8 | Инструкция по охране труда при работе с персональными электронно-вычислительными машинами | Самоконтроль | постоянно |
| Умственное напряжение | 2 | 5 | 10 | Соблюдение распорядка дня | Самоконтроль | постоянно |
| Поражение электрическим током | 2 | 3 | 6 | Инструкция по охране труда | Соблюдение и выполнение требований инструкции | постоянно |
| Пожарная опасность | 1 | 2 | 2 | Инструкция по пожарной безопасности | Соблюдение правил пожарной безопасности | постоянно |

Окончание таблицы 7.4

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Администратор веб-сервиса | трудовая | Напряжение зрительных анализаторов | 2 | 4 | 8 | Инструкция по охране труда при работе с персональными компьютерами | Соблюдение требований инструкции | постоянно |
| Статическая поза (заболевания кистей рук) | 3 | 3 | 9 | Самоконтроль | Соблюдение распорядка дня, производственная гимнастика | постоянно |
| Простудные заболевания | 3 | 4 | 12 | Самоконтроль | Обеспечение соответствующих условий производственной среды | постоянно |

Оценка организации охраны труда, производственной санитарии и промышленной безопасности приведена в таблице 7.5.

Таблица 7.5 – Характеристика производственной санитарии и промышленной безопасности

|  |  |
| --- | --- |
| Исходные параметры | Характеристика реа­лизуемого парамет­ра |
| 1 | 2 |
| Организационные мероприятия по обеспечению охраны труда | - |
| Количество имевших место за отчетный период: | - |
| - аварий/количество пострадавших | - |
| - инцидентов/количество пострадавших | - |
| - несчастных случаев/количество пострадавших | - |

Окончание таблицы 7.5

|  |  |
| --- | --- |
| 1 | 2 |
| Технические средства и оборудование, обеспечивающие параметры микрокли­мата: | |
| - предусматриваемые системы вентиляции | естественная |
| - система отопления в помещении | Централизованное (водяное) |
| - способ уборки помещения | влажная |
| Технические средства и оборудование, обеспечивающие параметры освеще­ния: | |
| - характеристика зрительной работы, разряд и подразряд зрительной работы | III |
| - вид и система искусственного освещения в помещении | общая |
| - источники искусственного освещения / мощность ламп | 9 Вт |
| - исполнение светильников / количество | светодиодные лампы / 2 шт |
| - исполнение естественного освещения (боковое или бо­ковое и верхнее) | Боковое |
| - коэффициент естественной освещенности (КЕО, %) | 1,5 |
| - мероприятия по обеспечению нормальной зрительной работы (до нормируемых значений) на рабочих местах |  |
| Технические средства и оборудование, обеспечивающие техническую безопас­ность: | |
| - знаки безопасности на оборудовании | - |
| - класс помещения по опасности поражения электриче­ским током | без повышенной опасности |
| - класс электрооборудования по способу защиты челове­ка от поражения электрическим током | I |
| - сопротивление изоляции токоведущих частей, МОм | 0,5 |
| - тип заземления | T-N |
| - места (зоны) накопления зарядов статического электри­чества. | ПЭВМ |
| - средства технической и коллективной защиты от пора­жения электрическим током и статического электриче­ства | изоляция, УЗО |
| - основные и дополнительные электрозащитные средства | - |

В соответствии с информацией, представленной в таблице 7.5 представленные мероприятия по обеспечению электробезопасности соответствуют ТКП 181-2009 (02230) «Правила технической эксплуатации электроустановок потребителей» и ТКП 427–2012 «Правила техники безопасности при эксплуатации электроустано­вок».

Далее приведен расчёт необходимого количества светильников для освещения помещения методом светового потока.

Расчет искусственного освещения в цехе производится методом светового потока по формуле (7.2):

 (7.2)

где N – число светильников, обеспечивающее требуемую освещенность в помещении, шт.;

– нормируемая освещенность, лк (для III разряда зрительной работы и малого, среднего и большого контраста объекта с фоном – 300 лк);

F – световой поток одной лампы, лм (для светодиодной лампы мощностью 9 Вт – 700);

S – площадь помещения, м2 (17,3 м2);

k – коэффициент запаса, зависящий от состояния воздушной среды в помещении (примем равным 1);

z – поправочный коэффициент, учитывающий неравномерность освещенности в помещении (примем равным 1,2);

– коэффициент использования светового потока, зависит от типа светильника, индекса помещения i, коэффициентов ρп, ρст, и ρр отражения потолка, стен и рабочей поверхности (в формулу значение коэффициента нужно подставлять в долях единицы).

Индекс помещения определяется по формуле (7.3):

 (7.3)

где a и b – длина и ширина помещения, м (для рассматриваемого помещения – 2,88 и 4 м);

hp – высота подвеса светильников, м (2,2 м).

Коэффициент отражения побеленных потолков принимается равным   
ρп= 50 %, стен, покрытых на высоту 1,8 м глазурованной плиткой,   
ρст = 50…70 %. Коэффициент отражения стен и потолка ξ зависит от характера отражающей поверхности: учитывая, что в помещении побеленные стены при не завешанных окнах и светлый деревянный потолок – ξ = 50 %;

Подставляя данные в формулу (7.3) получаем:

.

При данном индексе площади помещения и коэффициенте отражения стен и потолка ξ (50 %), коэффициент использования светового потока для светодиодных светильников η составляет 18. Подставляя данные в формулу (7.2), получаем необходимое количество светильников:



Принимаем количество светильников – 1 шт. В помещении установлено 2 лампы, значит, количество установленных ламп превышает необходимое. Вывод: одну лампу можно убрать, либо установить лампы с меньшей мощностью.

Система пожарной безопасности – это комплекс экономических, социальных, организационных, научно-технических и правовых мер, а также сил и средств, направленных на предупреждение возможных причин пожаров в дирекции.

Возможные причины возникновения пожара: неисправность электропроводки, неосторожное обращение с огнем, нахождение в помещении горюче-смазочных материалов и других легко воспламеняющихся веществ.

В таблице 7.6. отражены основные характеристики организации по степени подверженности пожарам.

Таблица 7.6 – Противопожарные мероприятия

|  |  |
| --- | --- |
| Исходные параметры | Значение реализуемого  параметра |
| Наименование помещения | Кабинет |
| Категория производства по пожароопасности | Д |
| Классификация производственного помещения по взрыво- и пожароопасности | – |
| Характеристика материалов стен по сгораемости | Несгораемая |
| Степень огнестойкости стен | II R 90-КО |
| Степень огнестойкости перекрытий | II R 60-КО |
| Расстояние от наиболее удаленного рабочего места до эвакуационного выхода, м | 20 |
| Количество эвакуационных выходов, шт. | 2 |
| Автоматические установки огнетушения | – |
| Тип извещателей о пожаре | дымовой |
| Первичные средства огнетушения | – |

Мероприятия по обеспечению пожарной безопасности соответствуют требованиям Декрета № 7, ТНПА противопожарного нормирования и стандартизации.

Во исполнение Закона Республики Беларусь «О пенсионном обеспече­нии» все объекты хозяйственной деятельности независимо от формы собствен­ности обязаны проводить не реже одного раза в пять лет аттестацию рабочих мест по условиям труда.

Аттестация проводится в соответствии с Положением о порядке проведе­ния аттестации рабочих мест по условиям труда и Инструк­цией по оценке условий труда при аттестации рабочих мест по условиям труда и предоставлению компенсаций по ее результатам.

В основу аттестации рабочих мест положены гигиенические критерии оценки условий труда, установленные в Санитарных нормах, правилах и гигие­нических нормативах «Гигиеническая классификация условий тру­да», утвержденных Постановлением Министерства здравоохранения Республи­ки Беларусь от 28.12.2012 г. № 211.

В соответствии с этим документом условия труда подразделяются на че­тыре класса: оптимальные, допустимые - относятся к безопасным, вредные и опасные. Компенсация профессиональных вредностей, а также средства защиты и личная гигиена рабочих представлены в таблице 7.7.

Таблица 7.7 – Компенсация профессиональных вредностей. Средства индивидуальной защиты и личная гигиена работающих

|  |  |
| --- | --- |
| Исходные параметры | Значение реализуемого параметра |
| 1 | 2 |
| Профессия (должность) | Администратор веб-сервиса |
| Условия труда | 2 класс – допустимые |
| Продолжительность дополнительного отпуска, дни  Пенсионный возраст, лет (2020) | 1 (по контракту) |
| – женщин | 57 |
| – мужчин | 62 |
| Обеспечение ЛПП |  |
| Спецодеждой | – |
| Спецобувью | – |
| Средствами индивидуальной защиты органов зрения и дыхания | – |
| Средства обеззараживания кожи | вода, мыло, антисептик |
| Метод обеззараживания кожи | мытье рук |
| Периодичность медосмотра | 1 р. в 2 года |

В ходе выполнения раздела «Охрана труда» была проделана следующая работа:

* Дана характеристика объекта с точки зрения охраны труда: условия труда администратора веб-сервиса относятся к допустимым условиям (2 класс), которые характеризуются такими уровнями факторов среды и трудового процесса, которые не превышают установленных гигиенических нормативов для рабочих мест, а возможные изменения функционального состояния организма, возникающие под их воздействием, восстанавливаются во время регламентированного отдыха или к началу следующей смены и не оказывают неблагоприятного действия на состояние здоровья работников в ближайшем и отдаленном периоде.
* Разработана карта рисков для администратора веб-сервиса.
* Произведена оценка организации охраны труда, производственной санитарии, промышленной и пожарной безопасности.

# [8 Промышленная экология](#_Toc474749010)

В данном разделе будут рассмотрены некоторые аспекты промышленной экологии.

Промышленная экология – прикладная наука о взаимодействии промышленности и окружающей среды, и наоборот – влияние условий природной среды на функционирование предприятий и их комплексов. Общая характеристика экологической деятельности организации приведена в таблице 8.1.

Таблица 8.1 – Общая характеристика экологической деятельности организации

|  |  |
| --- | --- |
| Исходные параметры | Значение реализуемого параметра |
| Нормативы допустимых выбросов (НДВ) (из экологического паспорта) | не требуется |
| Объем сброса сточных вод, м3 (из экологического паспорта) | 0,14 м3/день |
| Количество (объем) образования твердых бытовых отходов, т (м3) /день | 0,08 м3/ день |
| Наличие систем очистки воды и сточных вод | Отсутствует |
| Обращение (утилизация, рециклинг, переработка, захоронение и т. п.) с отходами | Раздельный сбор, складирование в контейнер и вывоз, сдача макулатуры, ежегодно |
| Мероприятия по энергосбережению | Рациональное использование электроэнергии |

В таблице 8.2 приведены экологические аспекты деятельности и виды воздействия экологических аспектов на окружающую среду, а также возможные мероприятия по сокращению воздействия.

Таблица 8.2 – Экологические аспекты деятельности и виды воздействия экологических аспектов на окружающую среду

|  |  |  |
| --- | --- | --- |
| Экологический аспект | Воздействие на окружающую среду (работающих) | Предложения по сокращению воздействия |
| Освещенность | Расход энергии | Рациональное использование электроэнергии |
| Отработанные лампы | Загрязнение тяжелыми металлами | Сортировка, централизованный сбор и утилизация |
| Энергия | Загрязнение атмосферы | Рационально использование, мероприятия по энергосбережению |
| ЭМП | воздействие ЭМП на работающих | Соблюдение режима труда, современное оборудование |
| Информация | перенапряжение анализаторов | Более эффективные системы поиска информации |
| Мусор | Твердые отходы производства | Раздельный сбор. Переработка вторичного сырья |
| Сточная вода (бытовая) | Загрязнение гидросферы | Установка счетчика, фильтра, использование рециркуляции бытовой воды |

Схема материальных потоков при работе с веб-сервисом представлена на рисунке 8.1.

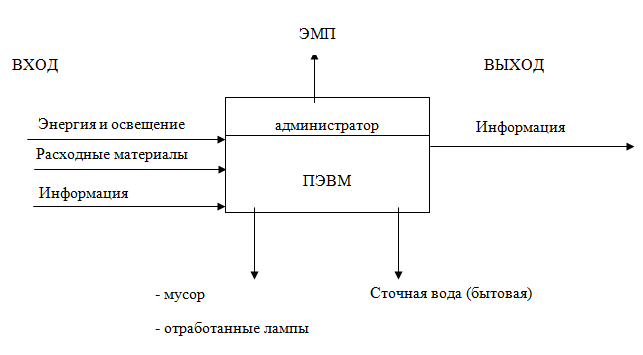


Рисунок 8.1 – Схема материальных потоков при работе с веб-сервисом

Утилизация компьютерной и офисной техники – это передовой подход к сохранению окружающей среды с пользой для бюджета. Речь идет о специальной услуге, которую оказывают профильные компании, принимая устаревшие единицы электрооборудования и комплектующие для дальнейшей переработки. Сдача компьютерной техники на переработку целесообразна и в рамках небольших компаний, и в крупных международных корпорациях.

Утилизация устаревшей компьютерной техники включает несколько этапов:

* оценка технического состояния устройств и составление "дефектных" актов;
* перевозка утилизируемой техники;
* извлечение ценных деталей и материалов из списанных устройств;
* сортировка;
* упаковка;
* переработка полученного лома с последующим извлечением драгоценных металлов.
* Пример. На переработку поступает 20 кг печатных плат персональных компьютеров поколения Pentium (материнские платы). Во вращающийся барабан объемом 150 л, оборудованный электрическим подогревом, загружают 20 кг печатных плат с радиодеталями навесного монтажа, помещенные в контейнер из металлической сетки с ячейкой 5×5 мм, и приливают 40 л щелочного водного раствора, содержащего 40% NaOH и 10% KOH.

Процесс растворения лака с поверхности плат ведут при перемешивании и температуре 70°C в течение 2 ч. Затем для хлопьеобразования и коагуляции лакового покрытия добавляют катионный флокулянт «Праестол 650» из расчета 0,2 кг/м3и дополнительно перемешивают в течение 1 часа. Полученный щелочной раствор сливают и отстаивают. После чего осветленную часть раствора вновь используют для обработки новых порций печатных плат, а сгущенную часть накапливают и утилизируют.

В результате достигается полное растворение лакового покрытия с печатных плат. Обработанные таким образом платы промывают в барабане водой и для растворения оловянного припоя заливают 10%-ный раствор метансульфоновой кислоты и процесс ведут по способу-прототипу.

Полученную суспензию метаоловянной кислоты в растворе метансульфоновой кислоты коагулируют путем введения добавки ПАВ с последующим кипячением в течение 30 мин. После охлаждения раствор декантируют от осевшей метаоловянной кислоты, а платы и взвесь метаоловянной кислоты сепарируют.

Выделенную таким образом метаоловянную кислоту отфильтровывают на вакуумном фильтре, промывают водой, сушат и прокаливают при температуре 800°C с получением товарного продукта оксида олова, а из раствора метансульфоновой кислоты осаждают сульфат свинца серной кислотой.

Полученный фильтрат корректируют по содержанию метансульфоновой кислоты и повторно используют для растворения припоя следующих порций плат. После сепарации платы вновь промывают водой и сушат.

Затем платы загружают во вращающийся барабан и приливают 30 л раствора, содержащего 30% NaCl и 20% CuCl2. Растворение меди с поверхности плат ведут при слабом перемешивании в течение 60 мин и температуре 80°C. Обработанные таким образом платы промывают водой и далее используют по назначению, а растворы отправляют на электрохимическую переработку в электролизер.

В ходе выполнения данного раздела был определён экологический аспект деятельности и виды воздействия экологических аспектов на окружающую среду.

# 9 Ресурсосбережение

Ресурсосбережение – система научно-технических, организационных, экономических и воспитательных мер, направленных на наиболее рациональное и эффективное использование всех видов ресурсов.

Основным ресурсом, который потребляет компьютерная техника, является электроэнергия. Подавляющее большинство технических средств механизации и автоматизации производственных процессов (оборудование, приборы ЭВМ), замена человеческого труда машинным в быту имеют электрическую основу.

Основными принципами государственной политики Республики Беларусь в сфере ресурсосбережения являются:

- осуществление государственного надзора за рациональным использованием топливно-энергетических ресурсов;

- разработка государственных и межгосударственных научно-технических, республиканских, отраслевых и региональных программ энергосбережения и их финансирование;

- создание системы финансово-экономических механизмов, обеспечивающих экономическую заинтересованность производителей и пользователей в эффективном использовании топливно-энергетических ресурсов, вовлечении в топливно-энергетический баланс нетрадиционных и возобновляемых источников энергии, а также в инвестировании средств в энергосберегающие мероприятия;

- повышение уровня самообеспечения республики местными топливно-энергетическими ресурсами;

- осуществление государственной экспертизы энергетической эффективности проектных решений;

- создание и широкое распространение экологически чистых и безопасных энергетических технологий, обеспечение безопасного для населения состояния окружающие среды в процессе использования топливно-энергетических ресурсов;

- реализация демонстрационных проектов высокой энергетической эффективности;

- информационное обеспечение деятельности по энергосбережению и пропаганда передового отечественного и зарубежного опыта в этой области;

- обучение производственного персонала и населения методам экономии топлива и энергии;

- создание других экономических, информационных, организационных условий для реализации принципов энергосбережения.

Основные направления энергосбережения регламентируются международными, межгосударственными и государственными нормативными правовыми актами, техническими нормативными правовыми актами, утвержденными специально уполномоченными государственными органами надзора и контроля.

Для реализации ресурсосбережения при работе с компьютером необходимо экономно использовать электроэнергию. Компьютеры потребляют много энергии, тем более, зачастую от них требуется оставаться включенными сутками. Наибольшее количество энергии тратится на поддержание работы монитора и жесткого диска. Поэтому в любой операционной системе есть несколько энергосберегающих режимов, использование которых может значительно сократить потребляемую энергию и, как следствие, ресурсы и деньги.

Например, в операционной системе «Windows», начиная с версии «Windows 7», существуют три энергосберегающих режима – сон, гибернация (спящий режим) и гибридный сон.

Сон – это энергосберегающий режим, позволяющий компьютеру за несколько секунд вернуться во включенное состояние. При переходе в режим сна, питание компьютера не отключается полностью, а лишь переходит на сниженное энергопотребление. Открытые программы и документы сохраняются в оперативной памяти, чтобы сразу после вывода компьютера из режима сна пользователь смог возобновить работу. Если во время сна питание компьютера будет полностью отключено, то все несохраненные настройки и изменения файлов будут утрачены.

Гибернация (спящий режим) – это энергосберегающий режим, разработанный специально для ноутбуков. В отличие от режима сна, помещающего открытые программы и документы в оперативную память, спящий режим сохраняет открытые документы и программы на жесткий диск (в файл hiberfil.sys) и затем переводит компьютер в режим сниженного энергопотребления. Все настройки и изменения в документах после выхода из режима гибернации сохранятся даже при полном отключении питания. Выход компьютера из спящего режима происходит обычно быстро – быстрее, чем включение «Windows» после завершения работы, но дольше, чем выход из режима сна. Все открытые на момент входа в спящий режим документы и программы восстанавливаются из файла «hiberfil.sys», после чего вы сразу можете вернуться к работе, продолжив её с того места, где вы остановились.

Гибридный сон разработан специально для настольных компьютеров. Режим гибридного сна представляет собой комбинацию режимов сна и гибернации – режим гибридного сна помещает ваши настройки, открытые документы и программы в оперативную память и на жесткий диск, после чего компьютер переходит в режим пониженного энергопотребления. Вы сможете быстро вывести компьютер из состояния гибридного сна и продолжить работу. Все настройки и изменения в документах после выхода из режима гибридного сна сохранятся даже при полном отключении питания. Обычно режим гибридного сна на настольных компьютерах по умолчанию включен.

Все современные операционные системы поддерживают механизмы энергосбережения, позволяющие выключать отдельные компоненты после определённого периода бездействия. Жёсткие диски могут останавливать вращающиеся пластины, мониторы могут выключаться, да и весь компьютер может переходить в режим ожидания или даже гибернации. Последний способ является весьма эффективным для выключения системы, поскольку при повторном включении содержимое памяти до гибернации считывается с жёсткого диска, поэтому заново операционная система не загружается.

Наконец, пользователь тоже может немало сделать, начиная от включения отдельных устройств только тогда, когда это нужно, и заканчивая характером своей деятельности в рабочем или игровом окружении. Компьютер, который ничего не делает, лучше перевести в состояние гибернации или выключить, а внешнюю периферию лучше выключать или переводить в режим ожидания, когда она не нужна.

# Заключение

В результате выполнения данного дипломного проекта было разработано программное веб-приложение системы учета рабочего времени. Приложение предоставляет возможность создавать, редактировать и удалять проекты и задачи, назначать исполнителей и отслеживать время, затраченное на выполнение задач. Также реализована система аналитики, позволяющая пользователям получать статистику о своей работе и эффективности.

Данное веб-приложение предоставляет интуитивно понятный и функциональный интерфейс, что обеспечивает удобство использования для пользователей с разными уровнями опыта. Приложение соответствует всем предъявленным требованиям и может быть использовано в организациях различного типа и размера для управления проектами и задачами.

Программа является веб-приложением, что обеспечивает возможность одновременного доступа и взаимодействия с сервером для множества пользователей, расположенных в разных местах. Вся используемая информация хранится в базе данных, специально разработанной для данного проекта. Доступ к базе данных осуществляется только через веб-сервис, что обеспечивает безопасность данных.

Безопасность данных также гарантируется путем разграничения прав доступа и системы авторизации. Различные роли пользователей, такие как администратор, менеджер и исполнитель, имеют разные уровни доступа к функциям приложения.

Разработанное приложение позволяет производить учет проектов, задач и пользователей, а также предоставлять статистику об эффективности работы каждого пользователя и команды в целом. Это позволяет определить возможные узкие места и проблемы, а также предпринять меры по их устранению.

В процессе разработки данного дипломного проекта были использованы современные технологии и фреймворки, такие как Spring Boot, Thymeleaf и Bootstrap. Это обеспечивает быстроту и гибкость при разработке, а также возможность дальнейшего расширения и модернизации приложения.

В заключении можно сказать, что разработка данного веб-приложения для управления проектами и задачами успешно выполнена. Оно обеспечивает удобный и эффективный способ организации работы над проектами и задачами, помогая пользователям следить за прогрессом и эффективностью своей работы. В дальнейшем, приложение может быть адаптировано для интеграции с другими системами и сервисами, что позволит обеспечить еще более гибкий и функциональный инструмент для управления проектами.

Кроме того, веб-приложение имеет потенциал для дополнительного развития и расширения функционала. В будущем могут быть добавлены новые возможности, такие как интеграция с календарями, автоматическое формирование отчетов и другие полезные функции, которые сделают приложение еще более удобным и полезным для пользователей.

В целом, результаты данного дипломного проекта демонстрируют успешное применение современных технологий и методологий разработки для создания удобного и функционального инструмента управления проектами и задачами. Веб-приложение может быть полезным для организаций различного размера и направлений деятельности, облегчая процесс управления проектами и повышая эффективность работы команды. В будущем, разработанное приложение может быть улучшено и расширено для предоставления еще большего функционала и удовлетворения растущих потребностей пользователей.

# Список использованной литературы

1. Способы контроля рабочего времени [Электронный ресурс]. – Режим доступа: https://planfact.io/blog/posts/sistema-ucheta-rabochego-vremeni-sotrudnikov. – Дата доступа: 01.04.2023.
2. Модели функционирования [Электронный ресурс]. – Режим доступа: http://econtool.com/modeli-funktsionirovaniya.html. – Дата доступа: 01.04.2023.
3. Trello [Электронный ресурс]. – Режим доступа: https://trello.com/. – Дата доступа: 01.04.2023.
4. Asana [Электронный ресурс]. – Режим доступа: https://asana.com/ru. – Дата доступа: 01.04.2023.
5. ClickUp [Электронный ресурс]. – Режим доступа: https://clickup.com/. – Дата доступа: 01.04.2023.
6. Monday [Электронный ресурс]. – Режим доступа: https://monday.com/lang/ru/. – Дата доступа: 01.04.2023.
7. One platform to streamline all workflows [Электронный ресурс]. – Режим доступа: https://www.wrike.com/vv/. – Дата доступа: 01.04.2023.
8. Серверные языки [Электронный ресурс]. – Режим доступа: https://codomaza.com/article/servernye-jazyki-java-obzor. – Дата доступа: 01.04.2023.
9. Spring Framework [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Spring\_Framework. – Дата доступа: 01.04.2023.
10. Язык HTML: что это такое и как он работает [Электронный ресурс]. – Режим доступа: https://skillbox.ru/media/code/chto\_takoe\_html/. – Дата доступа: 01.04.2023.
11. CSS [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/glossary/css/. – Дата доступа: 01.04.2023.
12. Основы JavaScript [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Learn/Getting\_started\_with\_the\_web/JavaScript\_basics. – Дата доступа: 01.04.2023.
13. UML [Электронный ресурс]. – Режим доступа: https://dic.academic.ru/dic.nsf/ruwiki/40660. – Дата доступа: 01.04.2023.
14. Spring MVC — основные принципы [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/336816/. – Дата доступа: 01.04.2023.
15. Информационное обеспечение [Электронный ресурс]. – Режим доступа: https://itteach.ru/avtomatizirovannie-sistemi-upravleniya/informatsionnoe-obespechenie. – Дата доступа: 01.04.2023.
16. Модульное тестирование (юнит-тестирование): что это, типы, инструменты [Электронный ресурс]. – Режим доступа: https://codernet.ru/articles/drugoe/modulnoe\_testirovanie\_yunit-testirovanie\_chto\_eto\_tipyi\_instrumentyi/. – Дата доступа: 01.04.2023.