**Система автоматизации процессов планирования и управления  IT-проектами**

Здесь должен быть титульный лист

**Содержание**

Введение

1. Современные системы автоматизации процессов планирования и управления IT-проектами

1.1. Теоретические аспекты управления проектами

1.2. Обзор существующих решений

1.2.1 YouGile

1.2.2 Jira

1.2.3. Trello

1.3. Обзор современных инструментов проектирования

1.4. Постановка задачи

2. Обоснование проектных решений

2.1. Разработка функциональной схемы системы Nemo

2.2. Разработка архитектуры системы Nemo

2.3. Программные средства для разработки.

2.3.1 Python

2.3.2 Django

2.3.3 JavaScript

2.3.4 React

2.3.5 Webpack

2.3.6 CSS

2.3.7 PostgreSQL

3. Разработка серверной части приложения

3.1. Описание логической структуры серверной части

3.2. Логическая и физическая реализация базы данных

3.3. Проектирование и разработка серверной части приложения

4. Разработка клиентской части приложения

4.1. Структура графического интерфейса пользователя

4.2. Описание программных модулей

5. Тестирование и отладка

6. Руководство пользователя

Заключение

Список использованных источников

**Введение**

В современных реалиях очень высокое значение имеют те информационные технологии, которые существуют и как системы обмена информацией, и как отдельные услуги, направленные на удовлетворение каких-либо потребительских потребностей. Разработка, внедрение и поддержка нового продукта или сервиса как правило сводится к проектной деятельности, у которой есть конкретная цель, сроки и ограничение в ресурсах.

Однако подходы к управлению IT-проектами сильно отличаются от тех, которые применяются в более традиционных и привычных областях. Проектный менеджмент, который используется в строительстве и производстве, работает с осязаемыми объектами. Управление же IT-проектами затрудняется тем, что в данной сфере быстро меняются тренды и требования рынка. Также приходится учитывать всё в режиме реального времени во избежание ситуации, когда проект становится невостребованным, устаревшим. Поэтому грамотный подход в очень важен в данной сфере.

К сожалению, далеко не все IT-компании уделяют должное внимание проектному менеджменту своей деятельности, вследствие чего на рынок выпускаются сотни и даже тысячи программ, которые устарели ещё до своего выхода; компании теряют клиентов из-за несоблюденных сроков; квалифицированные кадры уходят в другие компании, где их деятельность будет более осмысленной.

Целью данной работы является создание эффективной системы по планированию и управлению IT проектами,

Структура работы состоит из введение 4 основных разделов, заключения и списка литературы. В первом разделе работы описаны теоретические аспекты управления проектами, обзор существующих решений и современных инструментов проектирования. Во втором разделе приведено обзор и обоснование используемых проектных решений. В третьем разделе описывается разработка серверной части приложения, включающая в себя проектирование базы данных. В четвертом разделе описывается структура клиентского интерфейса. В последнем разделе приведено руководство пользователя.

**1. Современные системы автоматизации процессов планирования и управления IT-п****роектами**

**1.1. Теоретические аспекты управления проектами**

Информационные технологии основаны на достижениях научных исследований в таких областях как информатика, информационной безопасности, связи и системы коммуникации. Они призваны решать задачи по улучшению качества человеческой жизни в различных областях, начиная от производства заканчивая обычными бытовыми вещами.

В сфере информационных технологий важное место занимает проектный подход к работе. Термин "IT-проект" обычно используется для обозначения деятельности, связанной с использованием или созданием некоторой информационной технологии. Это приводит к тому, что ИТ-проекты охватывают очень разнообразные сферы деятельности: разработку программных приложений, создание информационных систем, развертывание IT-инфраструктуры и пр. В литературе выделяют несколько основных подходов в управлению IT-проектами.

Каскадная или «водопадная модель» (англ. waterfall model). Свое название она получила из-за последовательной ступенчатой структуры.

В этой модели разработка программы разбита на последовательные этапы, которые следуют один за другим в строго определённой последовательности и не пересекаются. При этом возвраты на предыдущие этапы не предусмотрены. На рис. 1.1 приведена схема каскадной модели, на которой указаны названия этапов. По факту, они могут отличаться в зависимости от организации, типа разрабатываемого продукта и т.д. Данную модель хорошо подходит для разработки проектов, у которых длительные фазы сбора требований и проектирования системы.

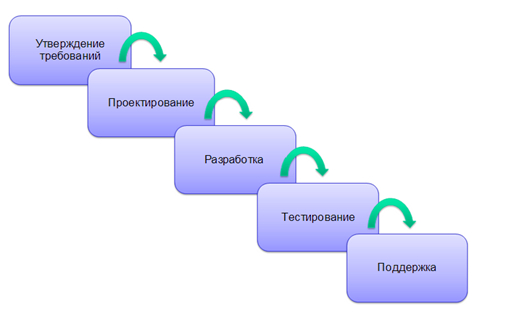


Рис. 1.1 - Каскадная модель

Преимущества:

* последовательное выполнение этапов проекта в строгом фиксированном порядке;
* полная и согласованная документация на каждом этапе;
* понятна в применении;

Недостатки

* запаздывание с получением результатов;
* высокая цена ошибки (чем раньше была допущена ошибка и чем позднее она была найдена, тем стоимость её возрастает);
* тестирование начинается очень поздно;

V-модель — это улучшенная версия каскадной модели. Она решает проблему позднего тестирования. Написание тестов начинается с этапа планирования и разработки требований. На рис. 1.2. представлена схема V-образной модели. По нисходящей ветке представлены этапы разработки, по восходящей ветке - соответствующие им процессы тестирования. Эта модель обладает всеми преимуществами каскадной модели. Также в модели предусмотрены верификация и валидация всех внешних и внутренних полученных данных. V-образная модель очень хорошо подходит для проектов, для которых критическим параметром является высокая надежность и безопасность (например, проекты, связанные с медициной). К недостаткам можно отнести то, что модель не поддерживает динамических изменений при разработке, а также то, что она не предназначена для управления параллельными событиями. Если нужно что-то исправить, нужно снова возвращаться к первому этапу.

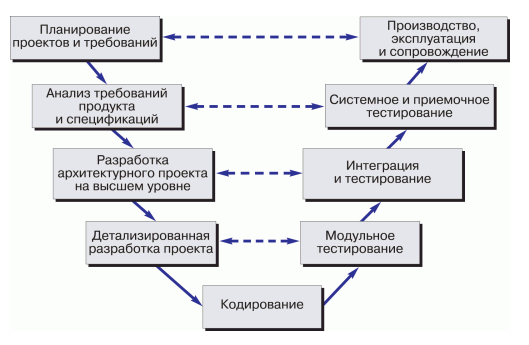
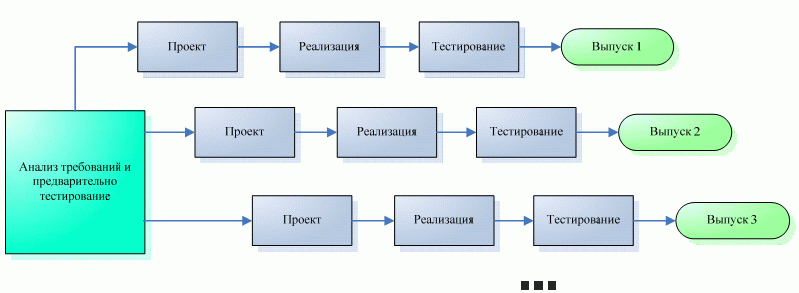


Рис. 1.2 — V-модель

Инкрементная модель — это метод, в котором проект реализуется и тестируется инкрементно (т. е. каждый раз с небольшими добавлениями) до самого окончания разработки. Это включает в себя как разработку, так и дальнейшую поддержку продукта.



#### Рис. 1.3. - инкрементная модель

### Преимущества инкрементной модели

* рабочее приложение выходит на ранней стадии жизненного цикла продукта;
* гибкость - изменить масштабы и требования проекта относительно менее затратно;
* небольшие итерации упрощают тестирование и внесение правок;
* проще идентифицировать риски, справиться с ними;
* каждая итерация — простая в управлении контрольная точка проекта;

### Недостатки инкрементной модели

* каждая фаза итерации неподвижна;
* могут возникнуть проблемы относительно архитектуры системы, так как не все требования собраны заранее для всего жизненного цикла ПО;

RAD-модель (от [англ.](https://ru.wikipedia.org/wiki/Английский_язык) rapid application development — быстрая разработка приложений) — модель быстрой разработки приложений, ключевыми для которой является скорость и удобство программирования.

Технология RAD позволяет обеспечить:

* быстроту продвижения программного продукта на рынок;
* интерфейс, устраивающий пользователя;
* лёгкую адаптируемость проекта к изменяющимся требованиям;
* простоту развития функциональности системы.

Agile-модель (гибкая методология разработки). Данная модель подходит для проектов, требования к которым изменяются часто, и которые не имеют изначально определённого конечного видения. Суть данной модели в том, что всё время, заложенное на исполнение проекта, разбивается на небольшие временные отрезки — спринты. В зависимости от трудоёмкости проекта, спринт может длиться от нескольких дней до нескольких недель. Для того, чтобы спринты проходили эффективно, Agile-методологии подразумевают информированность, способность быстро переключаться с задачи на задачу и регулярные планерки, предназначенныеа для обмена опытом и понимания прогресса.

Преимущества Agile-методологий:

* способность учитывать меняющиеся потребности пользователя
* проект, реализуемый по данной модели, полностью открыт к изменениям
* не требует тщательного планирования на начальном этапе

Итеративная или итерационная модель. В данной модели вместо одной продолжительной последовательности действий весь жизненный цикл продукта разбит на ряд отдельных минициклов. Причем каждый из них состоит из все тех же базовых стадий модели жизненного цикла. Эти минициклы называются итерациями. В каждой из итераций происходит разработка отдельного компонента системы, после чего этот компонент добавляется к уже ранее разработанному функционалу.

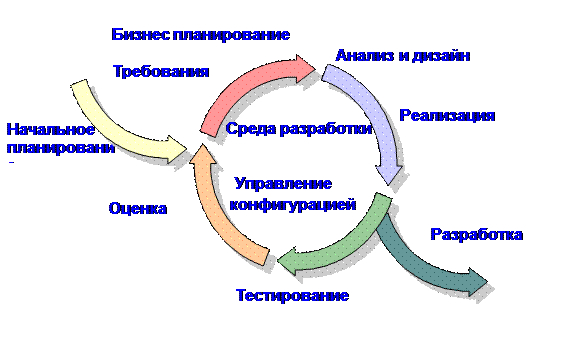


Рис. 1.4 — Итеративная модель

Плюсы итеративной модели:

* раннее создание работающего ПО;
* гибкость – готовность к изменению требований на любом этапе разработки;
* каждая итерация – маленький этап, для которого тестирование и анализ рисков обеспечить проще, чем для всего жизненного цикла продукта.

Минусы итеративной модели:

* каждая фаза – самостоятельна, отдельные итерации не накладываются;
* могут возникнуть проблемы с реализацией общей архитектуры системы, поскольку не все требования известны к началу проектирования.

Спиральная модель — представляет собой шаблон процесса разработки ПО, который сочетает идеи итеративной и каскадной моделей. Суть её в том, что весь процесс создания конечного продукта представлен в виде условной плоскости, разбитой на 4 сектора, каждый из которых представляет отдельные этапы его разработки: определение целей, оценка рисков, разработка и тестирование, планирование новой итерации.

В спиральной модели жизненный путь разрабатываемого продукта изображается в виде спирали, которая, начавшись на этапе планирования, раскручивается с прохождением каждого следующего шага. Таким образом, на выходе из очередного витка мы должны получить готовый протестированный прототип, который дополняет существующий билд. Прототип, удовлетворяющий всем требованиям – готов к релизу.

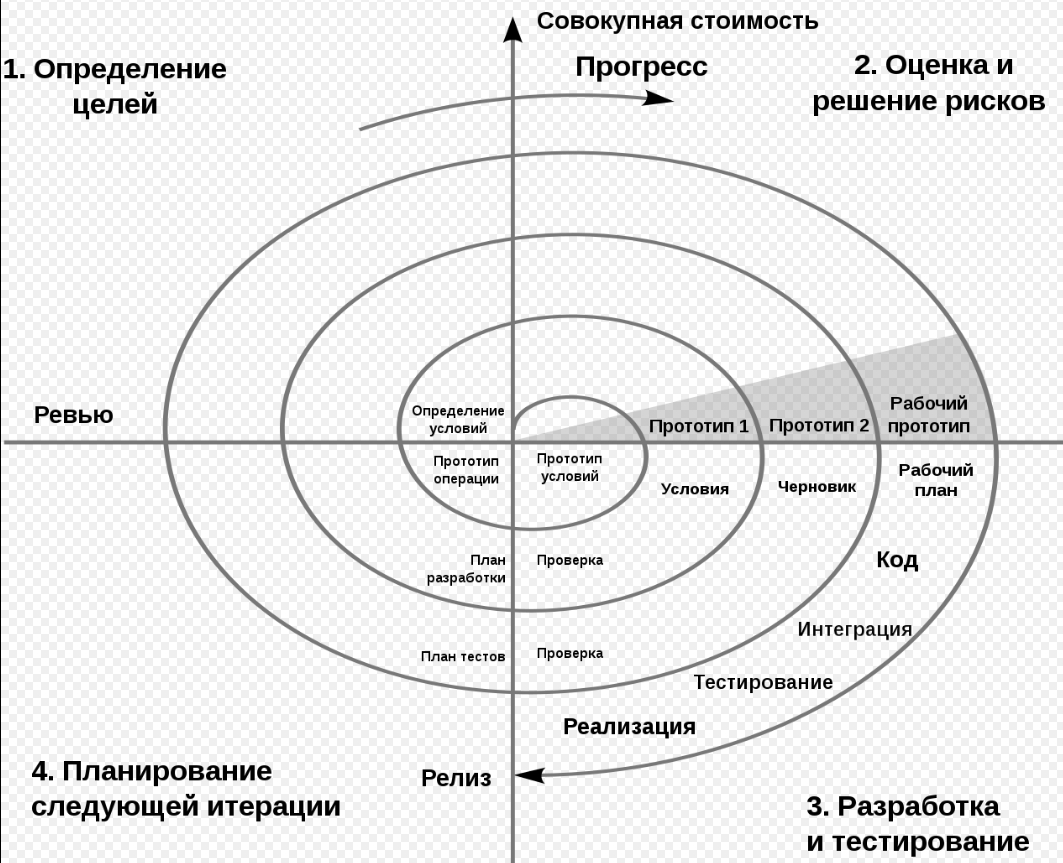


Рис. 1.5. - Спиральная модель

Плюсы спиральной модели:

* улучшенный анализ рисков;
* хорошая документация процесса разработки;
* гибкость – возможность внесения изменений и добавления новой функциональности даже на относительно поздних этапах;
* раннее создание рабочих прототипов.

Минусы спиральной модели

* может быть достаточно дорогой в использовании;
* управление рисками требует привлечения высококлассных специалистов;
* успех процесса в большой степени зависит от стадии анализа рисков;
* не подходит для небольших проектов.

Подводя итоги, мы можем сделать вывод, что даже внутри одной отрасли подходы к управлению проектами могут существенно различаться. А так как IT-индустрия является быстроразвивающейся отраслью, то подходы к разработке программных продуктов часто бывают намного более гибкими, нежели в обычных проектах.

**1.2. Обзор существующих решений**

**1.2.1 YouGile**

YouGile — это современная система управления задачами и общения в больших командах. Преимуществами YouGile являются возможность создания шаблонов проектов, бесплатная версия до 15 пользователей и возможность устанавливать таймер, который будет считать, сколько времени сотрудник работает над задачей. Также в YouGile можно настроить напоминания о дедлайнах по задачам. Также имеются персональные чаты, которые помогают экономить время на обсуждение. К минусам YouGile можно отнести отсутствие srcum-доски и отчетов по спринтам.

**1.2.2 Jira**

Jira — это продукт, разработанный компанией Atlassian, и предназначенный для организации процесса контроля запросов и задач, имеющий часть функциональности обычно присущей большим и дорогим системам управления проектами.

Ключевым понятием в JIRA является задача (англ. ticket или issue). Задача содержит название проекта, тему, тип, приоритет, компоненты и содержание. Также она может быть расширена дополнительными полями, приложениями или комментариями. Задача может редактироваться или просто изменять статус, например, из «открыт» в «закрыт». Какие переходы между состояниями возможны, определяется через настраиваемый [поток операций](https://ru.wikipedia.org/w/index.php?title=Поток_операций&action=edit&redlink=1). Любые изменения в задаче протоколируются в журнал.

Jira имеет большое количество возможностей конфигурации: для каждого приложения может быть определён отдельный тип задачи с собственным рабочим процессом, набором статусов, одним или несколькими видами представления ([англ.](https://ru.wikipedia.org/wiki/Английский_язык) *screens*). Кроме того, с помощью так называемых «схем» можно определить для каждого индивидуального Jira-проекта собственные права доступа, поведение и видимость полей и многое другое.

Благодаря универсальному подходу можно приспособить Jira для многих непрофильных задач, например, [управления требованиями](https://ru.wikipedia.org/wiki/Управление_требованиями_к_программному_обеспечению), [управления рисками](https://ru.wikipedia.org/wiki/Управление_рисками), вплоть до реализации небольшой [системы бронирования](https://ru.wikipedia.org/w/index.php?title=Система_бронирования&action=edit&redlink=1), автоматизации процесса [рекрутинга](https://ru.wikipedia.org/wiki/Рекрутинг).

Для интеграции с внешними системами поддерживает интерфейсы [SOAP](https://ru.wikipedia.org/wiki/SOAP), [XML-RPC](https://ru.wikipedia.org/wiki/XML-RPC) и [REST](https://ru.wikipedia.org/wiki/REST). Поставляется со средствами интеграции с такими [системами управления версиями](https://ru.wikipedia.org/wiki/Система_управления_версиями), как [Subversion](https://ru.wikipedia.org/wiki/Subversion), [CVS](https://ru.wikipedia.org/wiki/CVS), [Git](https://ru.wikipedia.org/wiki/Git), [Clearcase](https://ru.wikipedia.org/wiki/Clearcase), [Team Foundation Server](https://ru.wikipedia.org/wiki/Team_Foundation_Server), [Mercurial](https://ru.wikipedia.org/wiki/Mercurial) и [Perforce](https://ru.wikipedia.org/wiki/Perforce). Существуют дополнения, позволяющие встроить Jira в [интегрированные среды разработки](https://ru.wikipedia.org/wiki/Интегрированная_среда_разработки), в том числе [Eclipse](https://ru.wikipedia.org/wiki/Eclipse_(среда_разработки)) и [IntelliJ IDEA](https://ru.wikipedia.org/wiki/IntelliJ_IDEA). Переведена на многие языки, включая русский, английский, японский, немецкий, французский, испанский. Для сторонних разработчиков предоставляются средства разработки расширений системы — [плагинов](https://ru.wikipedia.org/wiki/Плагин).

**1.2.3. Trello**

Trello — веб-приложение для управления задачами, разработанное Fog Creek Software. В основе его работы лежит японская философия менеджмента «канбан», популяризацией которой занималась корпорация Toyouta в 80-е годы ХХ-века. Все проекты здесь представлены в виде «досок» со списком задач. «Карточки» с проектами и задачами прикрепляются к каждому пользователю в команде. Пользователи могут создавать свои доски задач с несколькими столбцами и перемещать задачи между ними. Обычно столбцы имеют следующие статусы: «к выполнению», «в процессе» и «сделано».

Trello предназначен как для личного, так и для делового использования. Кроме его использования в управлении проектами в области программного обеспечения, Trello так же используется в управлении недвижимостью, планировании уроков, бухгалтерском учете, управлении делами в адвокатских конторах и т. д. Богатый API позволяет осуществлять интеграцию с корпоративными системами или с облачными сервисами интеграции, такими как IFTTT и Zapier.

**1.3. Обзор современных инструментов проектирования**

CASE (англ. computer-aided software engineering) — набор инструментов и методов программной инженерии для проектирования программного обеспечения, который помогает обеспечить высокое качество программ, отсутствие ошибок и простоту в обслуживании программных продуктов.

Основной целью CASE-технологии является разграничение процесса проектирования программных продуктов от процесса кодирования и последующих этапов разработки, максимально автоматизировать процесс разработки. Для выполнения поставленной цели CASE-технологии используют два принципиально разных подхода к проектированию: структурный и объектно-ориентированный.

CASE-инструменты можно классифицировать по типам и категориям. Классификация по типам отражает ориентацию средств на те или иные процессы жизненного цикла разработки ПО, и, в основном, совпадают с компонентным составом крупных интегрированных CASE-систем, и включает следующие типы:

* средства анализа — предназначены для построения и анализа модели предметной области;
* средства [проектирования баз данных](https://ru.wikipedia.org/wiki/Проектирование_баз_данных);
* средства документирования.
* средства разработки приложений;
* средства [реинжиниринга процессов](https://ru.wikipedia.org/wiki/Реинжиниринг_бизнес-процессов);
* средства планирования и управления проектом;
* средства тестирования;

Классификация по категориям определяет степень интегрированности по выполняемым функциям и включают отдельные локальные средства, решающие небольшие автономные задачи, набор частично интегрированных средств, охватывающих большинство этапов жизненного цикла и полностью интегрированных средств, охватывающий весь жизненный цикл информационной системы и связанных общим репозиторием. Типичными CASE-инструментами являются:

* инструменты управления конфигурацией;
* инструменты моделирования данных;
* инструменты анализа и проектирования;
* инструменты преобразования моделей;
* инструменты редактирования программного кода;
* инструменты рефакторинга кода;
* генераторы кода;
* инструменты для построения UML-диаграмм.

**1.4. Постановка задачи**

Для того, чтобы обеспечить простой и понятный интерфейс взаимодействия программных модулей веб-приложении, необходимо обеспечить разделение функциональных возможностей между модулями. Поскольку решение задачи требует разработки интерактивного интерфейса для взаимодействия с пользователем, клиентской части веб-приложения понадобятся мощные инструменты для отображения интерфейса. Соответственно, для того, чтобы изолировать эту функциональность в одном модуле, необходимо сконцентрировать все действия по формированию отображения интерфейса в клиентском приложении. В качестве решения был выбран подход с созданием одностраничного приложения (SPA). Одностраничное приложение подразумевает вынос большей части логики в модуль клиентского приложения: обработка данных для отображения, отображение, обработка пользовательских событий, отправка данных на сервер. Таким образом, интерфейс для клиентского модуля представляет из себя методы по приему и отправке необработанных данных без отображения. Для решения задач клиентского приложения были выбраны следующие технологии:

* библиотека React для работы с отображением пользовательского интерфейса;
* библиотека semantic-ui-react для использования уже готовых компонентов интерфейса;
* библиотека Redux для управления состоянием приложения;

Основными задачами серверного приложения при реализации SPA являются:

* взаимодействие с базой данных;
* обработка запросов клиентского приложения и исполнение бизнес-логики приложения;
* формирование веб-страницы для первого пользовательского запроса.

Для решения задач серверного приложения были выбраны язык программирования Python и его фреймворк Django.

Взаимодействие базы данных и веб-сервера следует организовать, следуя принципу, что бизнес логика находится в коде серверного приложении, а не в базе данных. В этом случае база данных хранит данные, и предоставляет прямой доступ к данным, тогда как вся бизнес-логика реализована в подсистемах серверного приложения. База данных позволяет выполнять транзакции для проведения атомарных операций над данными. Изоляция бизнес-логики в подсистемах серверного приложения позволяет обеспечить прозрачный интерфейс взаимодействия сервера с базой данных. Для обеспечения подобной изоляции, можно определить следующую иерархическую структуру задач:

1. Разработка модуля клиентской части веб-приложения;
2. Разработка модуля серверной части веб-приложения;
3. Модуль базы данных.
4. Протокол обмена данных между клиентской и серверной сторонами.
5. Проектирование интерфейса взаимодействия между модулями клиентской и серверной части.
6. Проектирование интерфейса взаимодействия между модулем серверной части и базой данных.

**2. Обоснование проектных решений**

**2.1. Разработка функциональной схемы системы Nemo**

Функциональный состав системы должен обеспечивать необходимый набор возможностей для управления проектами. Для этого составим примерный перечень функций, которые должны быть реализованы в нашей системе.

1) Работа с задачами:

* создание задачи;
* обновление задачи;
* добавление комментариев по задаче;
* просмотр текущих задач;

2) Работа с проектами:

* создание проекта;
* обновление проекта;
* просмотр текущих проектов:

3) Работа с компаниями:

* создание компании;
* приглашение сотрудников в компании;
* выход из компании;

4) Прочие функции:

* механизм регистрации и авторизации пользователя;
* обновления профиля пользователя.

Исходя из указанного перечня задач, была составлена диаграмма прецедентов (англ. use case diagram), которую ещё также называют диаграмма вариантов использования.

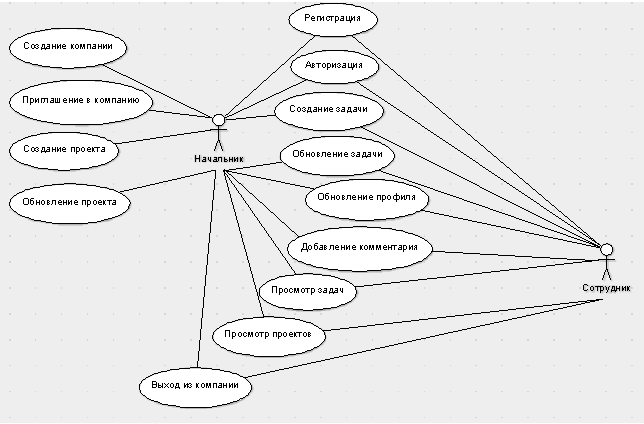


Рис. 2.1 — Диаграмма прецедентов

Диаграмма прецедентов представляет собой взаимодействие пользователя с системой, которое показывает взаимосвязь между пользователем и различными вариантами использования, в которые вовлечен пользователь. Диаграмма прецедентов может идентифицировать различные типы пользователей системы и разные прецеденты. Варианты использования, как правило, представлены либо кружками, либо эллипсами.

**2.2. Разработка архитектуры системы Nemo**

Система состоит из клиентской и серверной части, тем самым реализуя сетевую архитектуру «клиент-сервер». Клиентская часть реализует пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него. Серверная часть принимает запросы от пользователя, выполняет необходимые вычисления и затем формирует веб-страницу и отправляет её клиенту по сети с использованием протокола HTTP или HTTPS. Долгое время при разработке веб-ресурсов наиболее популярным подходом было создание «тонкого клиента» — веб-приложения, основная логика работы которого вынесена в серверную часть. Интенсивное развитие интернет-ресурсов и высокие требования к интерактивности сервисов привели к тому, что сейчас все больше веб-приложений создается с использованием подхода «толстый клиент» (или rich-клиент), в котором клиентская часть обеспечивает расширенную функциональность по предоставлению и обработке информации, а серверная занимается преимущественно хранением данных. Основой для реализации такого подхода в веб-приложениях стал AJAX (от англ. Asynchronous JavaScript and XML). При использовании AJAX клиент и сервер в фоновом режиме обмениваются данными без перезагрузки страницы. Использование AJAX привело к появлению «одностраничных приложений», где с сервера веб-страница загружается всего один раз в начале сессии, а вся дальнейшая работа по обмену данными осуществляется в фоновом режиме. Такой подход позволяет быть веб-приложению более интерактивным и производительным. Также широко используется технология WebSocket, которая не требует постоянных запросов от клиента к серверу, а создает двунаправленное соединение, при котором сервер может отправлять данные, без запроса от последнего. Таким образом предоставляется возможность динамически управлять контентом в режиме реального времени. При создании веб-приложения на стороне сервера существует возможность контролировать среду исполнения программы. Это позволяет применять для разработки таких программ широкий перечень разнообразных технологий и языков программирования. В качестве среды для исполнения клиентской части веб-приложений чаще всего выступает веб-браузер, поэтому для разработки таких программ используется устоявшийся набор технологий — HTML, CSS и JavaScript. Для формирования и обработки запросов, создания интерактивного и независимого от браузера интерфейса: ActiveX, Java, JavaScript.

**2.3. Программные средства для разработки.**

**2.3.1 Python**

Python – это высокоуровневый язык программирования общего назначения, разработанный сотрудником голландского института CWI Гвидо ван Россумом.

На сегодняшний день можно выделить 3 основных направления применения Python:

1) Веб-разработка. Здесь Python используются

2) Data Science: машинное обучение, анализ данных и визуализация

3) Автоматизация процессов

У Python ряд (достоинств):

1) читаемость и поддерживаемость кода

2) он поддерживает несколько парадигм программирования

3) Python совмести с основными платформами и системами

4) имеет надежную стандартную библиотеку

5) для Python создано огромное количество open-source фреймворков и инструментов.

6) он существенно упрощает разработку программного обеспечения

**2.3.2 Django**

Django – это веб-фреймворк с открытым исходным кодом на Python, используемый для быстрой разработки прагматичных и удобных в обслуживании веб-сайтов.

Основными целями Django являются простота, гибкость, надежность и масштабируемость.

У Django имеется целый ряд технических функций:

1) Панель администратора

2) Собственный веб-сервер

3) Концепция Model-View-Controller (MVC) (модель-представление-контроллер) архитектуры приложений.

4) собственная ORM (Object-Relational Mapping) – технология, которая связывает базы данных с концепциями объектно-ориентированных языков программирования.

5) HTTP библиотеки

6) Поддержка middleware – промежуточный слой, механизм для обработки запросов и ответов в Django. Это простая низкоуровневая система “плагинов”, которая глобально влияет на ввод и вывод в Django.

7) Фреймворк для unit-тестирования.

**2.3.3 JavaScript**

JavaScript – мультипарадигменный язык программирования, который соответствуетспецификации ECMAScript. Наряду с HTML и CSS, JavaScript является одной из основных технологий World Wide Web. JavaScript делает веб-страницы интерактивными и является важной частью веб-приложений. В подавляющем большинстве случаем он используется для программирования поведения на стороне клиента, и все основные веб-браузеры имеют специальный механизм для его выполнения.

Основными преимуществами данного языка являются его распространненость, а такжеогромное количество библиотек и фреймворков призванных упростить разработку.

**2.3.4 React**

React – это JavaScript-библиотека с открытым исходным кодом, созданная для разработки пользовательского интерфейса. Разрабатывается и поддерживается компанией Facebook, а также огромным сообществом разработчиков и компаний.

React может быть использован как основа для одностраничных (англ. Single page application, SPA ) и мобильных приложений.

Одной из основных особенностей React, является то, что он использует так называемый виртуальный DOM (от [англ.](https://ru.wikipedia.org/wiki/Английский_язык) Document Object Model— «объектная модель документа»). React создает структуру в памяти, с помощью которой вычисляет разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления DOM браузера. Программист, работающий с библиотекой, может считать, что страница обновляется вся целиком. Но по факту, библиотека самостоятельно решает, какие компоненты страницы необходимо обновить.

React обладает очень удобным и понятным API. Для работы с React нужно понимать некоторое количество терминов и различия между ними. Элементами называют объекты JavaScript, представляющие собой HTML-элементы. Например, h1, section или p. Компонентами же называют элементы React, которые создаются разработчиком и могут носить произвольное название. Как правило они содержат свою специфичную структуру и выполняют ряд функций. Элементы и компоненты React создаются с помощью JSX. JSX представляет собой углубленный синтаксис JavaScript, который внешне напоминает XML. React создает из компонентов аналог реального DOM дерева – VirtualDOM и представляет его в браузере. Библиотека следит за изменениями в виртуальном дереве и при изменении его обновляет реальный DOM, чтобы реальное и виртуальное дерево были одинаковыми. Выше упомянуты основные концепции, которые нужно знать, чтобы начинать работу с React.js. Так же, стоит добавить о существовании состояния в React, на на практике мы будем реализовывать единое состояние приложения при помощи Redux. React + Redux достаточно успешный союз, который встречается довольно часто. Компоненты обладают жизненным циклом, например: создание, время жизни и удаление. Библиотека предоставляет возможность определять различные моменты жизненных циклов компонентов и взаимодействовать с ними.

**2.3.5 Webpack**

Webpack является достаточно быстрым и мощным инструментом для сборки веб-проектов. Главная его особенность, это поддержка модульной системы. С помощью него можно организовать любую логику, любые форматы, быстро собрать проект, использовать множество других сторонних инструментов совместно с ним. Webpack сборку можно вызвать двумя способами: через консоль и через NPM скрипты. Логика и поведение того, как webpack будет собирать проект и какие преобразования он будет совершать описывается в его конфигурационном файле, который нужно указать при запуске webpack, либо ему нужно дать название, принятое по умолчанию webpack. В конфигурационном файле мы описываем, какой JavaScript файл является входным и в какой файл мы преобразуем все импортируемые в него JavaScript (и не только) файлы. Описываем, что делаем с файлами других форматоё при их импортировании в проект, например, при импортировании файлов с расширением css мы применяем плагины, входящие в список наших плагинов postcss. Так же, мы можем создать различные конфигурации для сборки в режиме разработки и сборки для публичной версии. Опираясь на переменные окружения будет назначаться та или иная конфигурация. Webpack обладает рядом встроенных плагинов, а также дополнительных, которые без труда можно установить. Многие из них помогают во время разработки, а некоторые нужны нам и в режиме публикации.

**2.3.6 CSS**

CSS (англ. Cascading Style Sheets – каскадные таблицы стилей) – это язык описания внешнего вида документа. Преимущественно используется для описания внешнего вида документов, написанного на каком-то языке разметке, например HTML. Наряду с HTML и JavaScript является одной из основополагающих технологий World Wide Web.

**2.3.7 PostgreSQL**

PostgreSQL представляет собой объектно-реляционную систему управления базами данных (СУБД) с открытым исходным кодом.

У PostgreSQL имеет множество различных функций среди которых:

1) пользовательские типы;

2) наследование таблиц;

3) ссылочная целостность внешнего ключа;

4) представления, правила, подзапросы;

5) вложенные транзакции (savepoints)

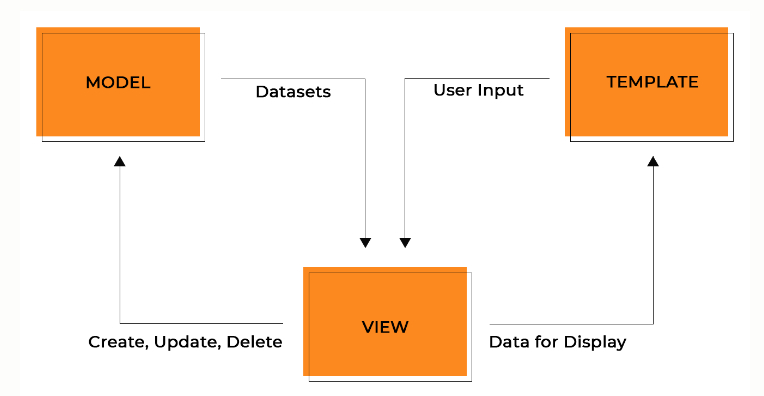
6) поддержка модификации базы данных с помощью Multiversion Concurrency Control (MVCC). Благодаря этому соблюдаются требования ACID и практически отпадает нужда в блокировках чтения.

7) асинхронные репликации.

**3. Разработка серверной части приложения**

**3.1. Описание логической структуры серверной части**

Серверная часть приложения реализована при помощи языка программирования Python и его фреймворка Django. Создатели данного фреймворка реализовали в нем паттерн MVC. Архитектура MVC позволяет разработчику работать с визуальным представлением и бизнес-логикой приложения отдельно. Хотя при работе с Django специалисты чаще используют термин MVT — Model-View-Template или модель-представление-шаблон. Компоненты MVT можно использовать независимо друг от друга.

Рис. 1.6. Схема архитектуры MVT в Django

Документация Django определяет модель (model) как «источник информации о данных, в которых содержаться ключевые поля и поведение данных». Обычно одна модель указывает на одну таблицу в базе данных.

Модели содержат информацию о данных. Эти данные представлены атрибутами и полями. Поскольку модель представляет

собой простой класс, она ничего не знает о других уровнях Django. Взаимодействие между уровнями происходит через API.

Модель отвечает за бизнес-логику, методы, свойства и другие элементы, связанные с манипуляцией данными. Также модели позволяют разработчикам создавать, читать, обновлять и удалять объекты в базе данных.

Представление (view) решает три задачи: принимает HTTP-запросы, реализует бизнес-логику, определённую методами и свойствами, отправляет HTTP-ответ на запросы. Т.е. представление получает данные от модели и предоставляет шаблонам (templates) доступ к этим данным или предварительно обрабатывает данные и затем предоставляет к ним доступ шаблонам.

В Django реализован мощный движок шаблонов и собственный язык разметки. Шаблоны представляют собой файлы с HTML-кодом, с помощью которого отображаются данные. Содержимое файлов может быть статическим или динамическим. Шаблоны не содержат бизнес-логики. Поэтому они только отображают данные. В приложении, которое было разработано в рамках данного проекта, используется всего несколько шаблонов — несколько шаблонов для создания писем пользователям приложениям, и один шаблон используется как точка монтирования для пользовательского интерфейса (более подробно об этом будет сказано в четвертом разделе).

**3.2. Логическая и физическая реализация базы данных**

Структура таблиц разработанной базы данных представлена ниже:

Таблица 1 — users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| id | integer | NO | PRIMARY | ID сотрудника |
| password | character varying(128) | NO |  | пароль |
| last\_login | timestamp with time zone | YES |  | Время последней активности на сайте |
| is\_super\_user | boolean | NO |  | Признак суперпользователя |
| name | character varying(64) | NO |  | имя |
| email | character varying(254) | NO | UNIQUE | e-mail |
| telegram | character varying(255) | YES |  | Аккаунт в Telegram |
| skype | character varying(255) | YES |  | Аккаунт в Skype |
| is\_staff | boolean | NO |  | Статус персонала |
| is\_active | boolean | NO |  | Активен? |
| date\_joined | timestamp with time zone | NO |  | Дата регистрации |
| avatar | character varying(100) | YES |  | Путь к файлу с аватаркой |

Таблица 2 — companies

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| id | integer | NO | PRIMARY | ID компании |
| name | character varying(255) | YES |  | название |
| creator\_id | integer | YES | FOREIGN | ID сотрудника, который создал |

Таблица 3 — company\_users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL-значения** | **Ключ** | **Описание** |
| company\_id | integer | NO | PRIMARY | ID компании |
| user\_id | integer | NO | ID сотрудника |
| is\_admin | boolean | NO |  | Признак того, является ли сотрудник администратором в компании |

Таблица 4 — projects

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL-значения** | **Ключ** | **Описание** |
| id | integer | NO | PRIMARY | ID проекта |
| name | character varying(255) | NO |  | Название проекта |
| company\_id | integer | NO | FOREIGN | ID компании, которой относится проект |
| creator\_id | integer | NO | FOREIGN | ID пользователя, который создал проект |
| description | text | NO |  | описание |

Таблица 5 — project\_participants

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| project\_id | integer | NO | PRIMARY | ID проекта |
| user\_id | integer | NO | ID сотрудника |

Таблица 6 — tasks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| id | integer | NO | PRIMARY | ID задачи |
| title | character varying(255) | NO |  | заголовок |
| description | text | NO |  | описание |
| status | smallint | NO |  | статус |
| deadline | date | YES |  | дедлайн |
| work\_hours | double precision | YES |  | Фактическая трудоемкость |
| planned\_work\_hours | double precision | YES |  | Планируемая трудоемкость |
| author\_id | integer | NO |  | ID автора |
| executor\_id | integer | NO | FOREIGN | ID исполнителя |
| manager\_id | integer | NO | FOREIGN | ID приемщика |
| project\_id | integer | NO | FOREIGN | ID проекта |

Таблица 7 — task\_participants

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| project\_id | integer | NO | PRIMARY | ID проекта |
| user\_id | integer | NO | ID сотрудника |

Таблица 8 — comments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| id | integer | NO | PRIMARY | ID комментария |
| text | text | NO |  | Текст комментария |
| timestamp | timestamp | NO |  | время добавления |
| task\_id | integer | NO | FOREIGN | ID задачи |
| user\_id | integer | NO | FOREIGN | ID сотрудника |

Таблица 9 — tags

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| id | integer | NO | PRIMARY | ID тега |
| tilte | character varying(255) | NO |  | название |

Таблица 10 — task\_tags

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| tag\_id | integer | NO | PRIMARY | ID тега |
| task\_id | integer | NO | ID задачи |

Таблица 11 — worktimehistory

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| id | integer | NO | PRIMARY | ID отметки затраченного времени |
| work\_date | date | NO |  | Дата проведения работ |
| work\_hours | double precision | NO |  | Затраченное время |
| minus\_work\_time | boolean | NO |  | признак вычитания |
| text | text | NO |  | комментарий |
| task\_id | integer | NO | FOREIGN | ID задачи |
| user\_id | integer | NO | FOREIGN | ID сотрудника |

Таблица 12 — knox\_authtoken

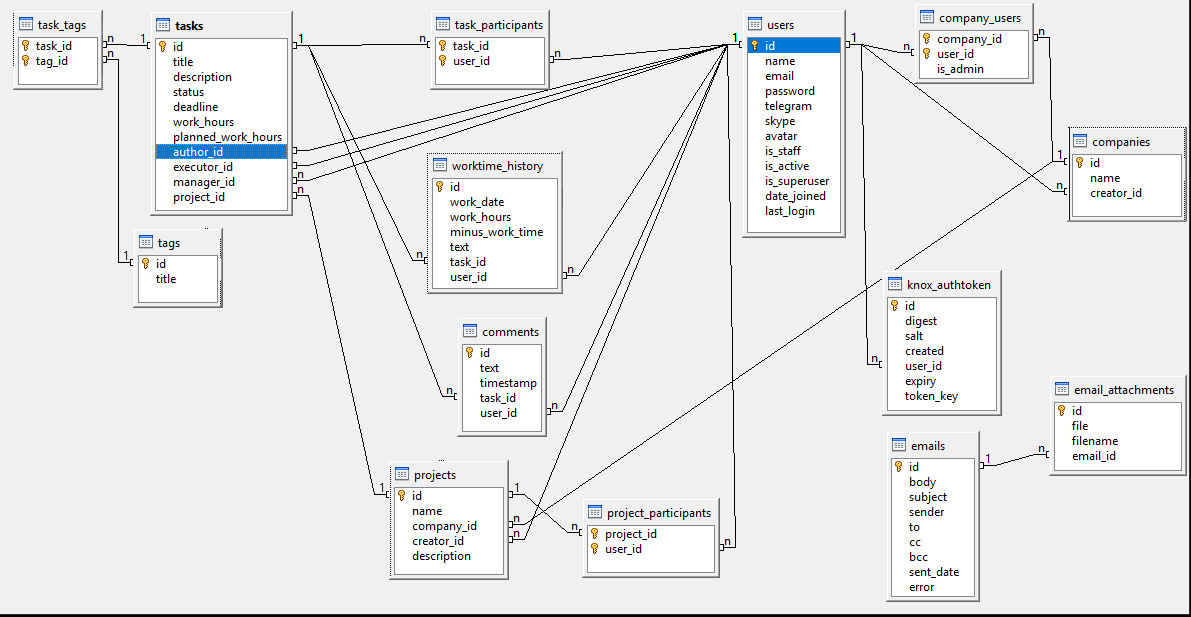
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| digest | character varying(128) | NO | PRIMARY |  |
| salt | character varying(16) | NO |  |  |
| created | timestamp with time zone | NO |  | Дата создания токена |
| user\_id | integer | NO |  | ID сотрудника |
| expiry | timestamp with time zone | YES |  | Дата окончания действия токена |
| token\_key | character varying(8) | NO |  |  |

Таблица 13 — emails

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| id | integer | NO | PRIMARY | ID письма |
| body | text | NO |  | Тело письма |
| subject | character varying(254) | NO |  | тема |
| sender | character varying(254) | NO |  | От кого |
| to | character varying(254)[] | NO |  | кому |
| cc | character varying(254)[] | YES |  | копия |
| bcc | character varying(254)[] | YES |  | Скрытая копия |
| sent\_date | timestamp with time zone | YES |  | Дата отправки |
| error | text | YES |  | Ошибка |

Таблица 14 — email\_attachments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Атрибут** | **Тип данных** | **Допускает NULL -значения** | **Ключ** | **Описание** |
| id | integer | NO | PRIMARY | ID приложения к письму |
| file | character varying(100) | NO |  | Путь к файлу |
| filename | character varying(254) | NO |  | Имя файла |
| email\_id | integer | NO |  | ID письма |



**Схема базы данных**

**3.3. Проектирование и разработка серверной части приложения**

Серверная часть приложения состоит из нескольких основных частей.

Папка **models -** содержит в себе файлы с описанием моделей, которые отображают информацию о данных, с которыми мы работаем.

Рис. 1.7. - Модель User, описывающая сущность «Пользователь»

Ниже представлен список моделей, которые используются в проекте:

* User — пользователи;
* Company — компании;
* CompanyUser — модель, описывающая связь компаний и сотрудников;
* Task — задачи;
* Comment - комментарии;
* WorkTimeHistory — модель, для хранения данных об учете затраченного времени;
* Project - проекты;
* Tag — теги;
* Email — модель, для хранения электронных писем;
* EmailAttachment — модель, для хранения вложений электронной почты;

Папка **migrations** — Django использует миграции для переноса изменений в моделях (добавление поля, удаление модели и т. д.) на структуру базы данных. Данные файлы генерируются автоматически, а работать с ними можно при помощи следующих команд:

* **makemigrations** — отвечает за создание новых миграций на основе изменений в моделях.
* **migrate** — отвечает за выполнение миграций.

Данные команды, как и все остальные команды Django, запускаются следующим образом:

python manage.py <название\_команды>

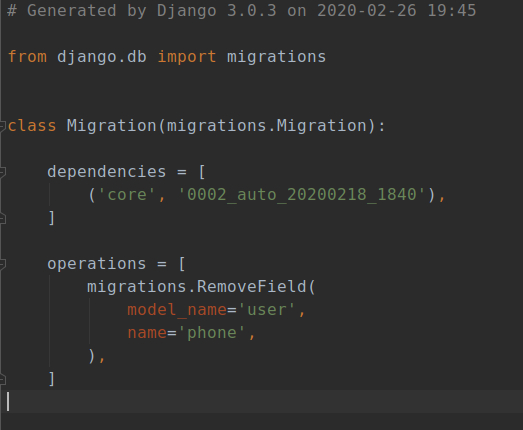


Рис. 1.8. - пример миграции, которая удаляет поле phone в модели User

В файлах nemo/urls.py и core/endpoints.py описана схема URL приложения. Чистая, элегантная схема URL-ов — важная часть качественного приложения. Django позволяет проектировать URL-адреса как пожелаем, без ограничений фреймворка. При запросе к странице нашего Django-приложения, используется следующий алгоритм для определения какой код выполнить:

1. Django определяет корневой модуль URLconf, который будет использован.
2. Загружается модуль Python и ищет переменную urlpatterns. Это должна быть последовательность экземплярова django.urls.path() и/или django.urls.re\_path()
3. Django проверяет один за другим все URL шаблоны по порядку и останавливается на первом шаблоне, который соответствует запрошенному URL.
4. Как только один из URL-шаблонов совпадает, Django импортирует и вызывает нужное представление, которое может быть функцией или классом.
5. Если ни один URL-шаблон не соответствует запросу, Django вызывает соответствующее представление обработки ошибок.

В файле core/api.py содержатся основные представления, которые обрабатывают запросы к приложению.

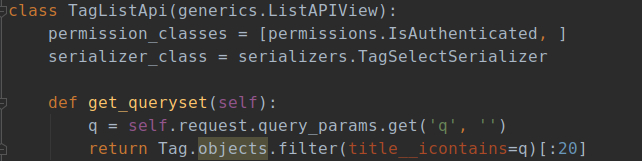


Рис. 1.9. - пример представления, которое возвращает список тегов

В файлах settings.py, local\_settings.py и test\_settings.py содержат полную конфигурацию проекта.

Папка static содержит в себе так называемые «статические файлы» . Веб-приложения обычно требует различные дополнительные файлы для своей работы (изображения, CSS, JavaScript и др.). В Django их принято называть «статическими файлами».

Папка meida содержит в себе файлы, которые были загружены пользователями (например, аватарки), а также файлы, которые были приложены к письмам.

Кроме этого, есть модули настройки административной модели (папка admins), а также необходимые модули для тестирования приложения.

**4. Разработка клиентской части приложения**

**4.1. Структура графического интерфейса пользователя**

Графический интерфейс пользователя реализован в виде одностраничного приложения (англ. single page application, SPA). Не считая страницу для восстановления паролы, он использует единственный HTML-документ как оболочку для всех веб-страниц. Интерфейс разработан при помощи языка JavaScript и его библиотеки React, с использованием CSS.

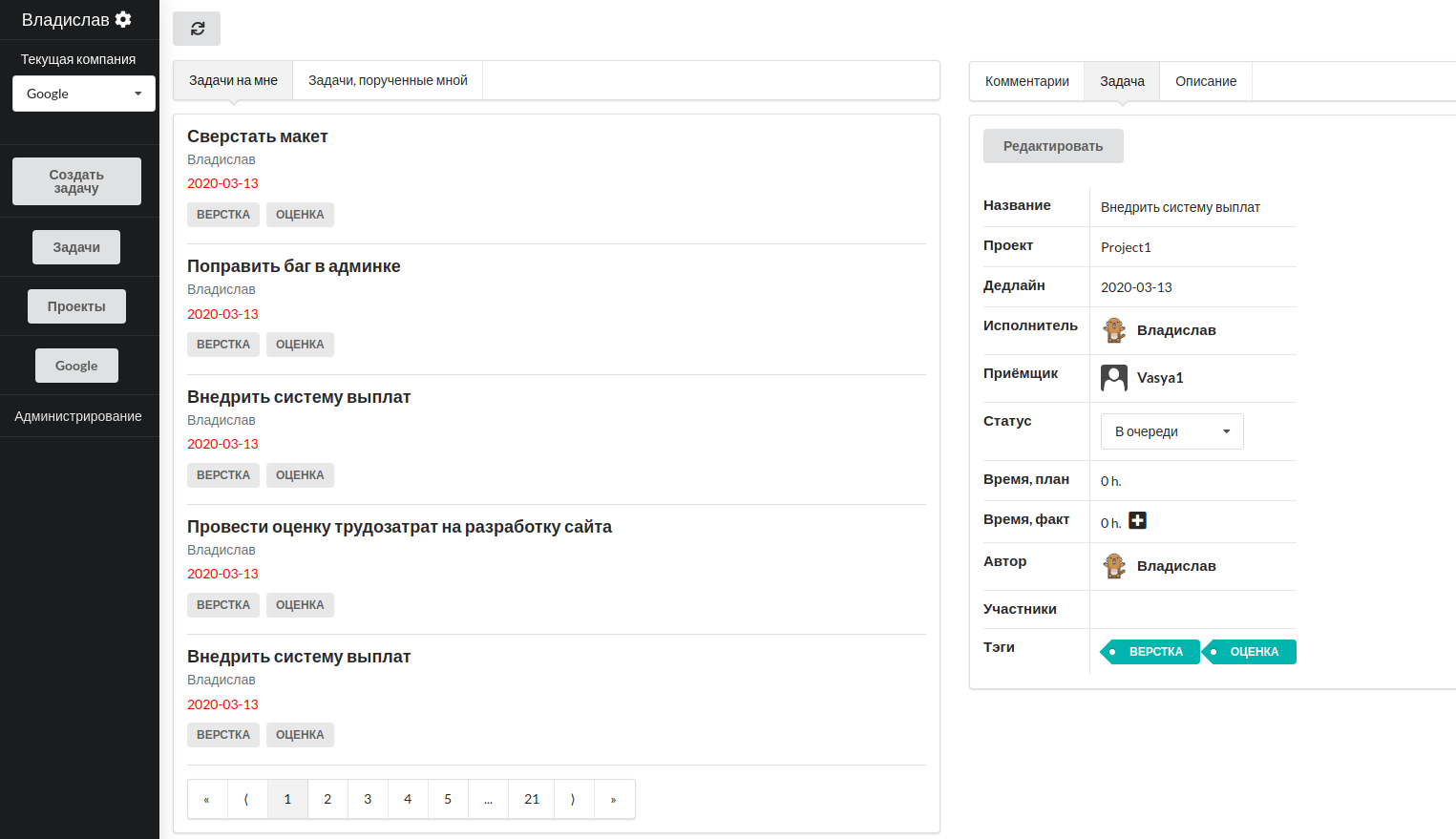


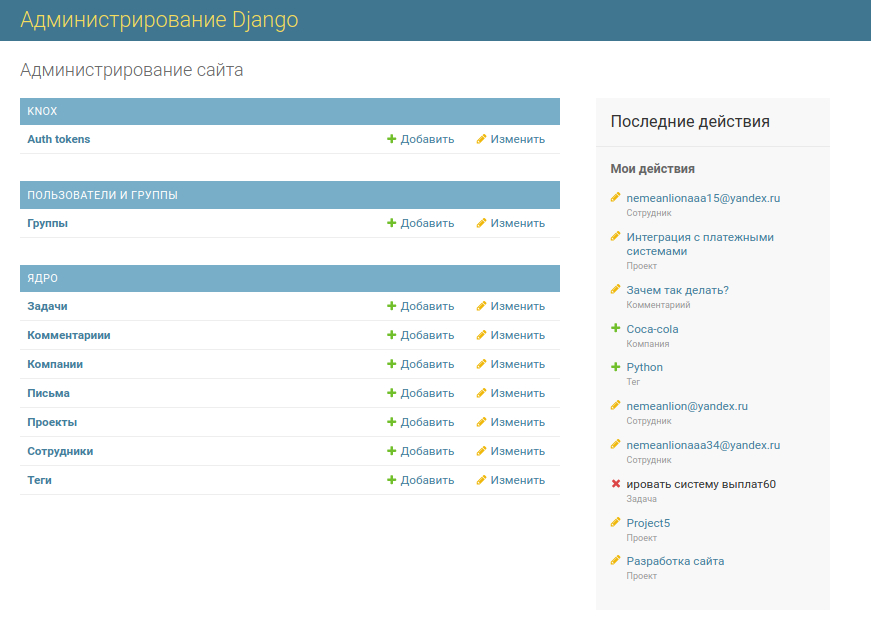
Рис. 1.10 — Интерфейс пользователя

(представлен список задач и карточка задачи)

Интерфейс пользователя состоит из следующих частей:

1. Формы регистрации и авторизации пользователя.
2. Профиль пользователя
3. Карточка компании
4. Карточка проекта
5. Списка проектов
6. Карточка задачи (описание, данные по задаче, комментарии)
7. Блок со списком задач

Помимо пользовательского интерфейса, который был разработан самостоятельно, стоит также упомянуть об административной панели, которую предоставляет Django.

Рис. 1.11. - Административная панель

Для её использования достаточно добавить пару настроек в settings.py и определить несколько Python классов, которые должны быть унаследованы от django.contrib.admin.ModelAdmin и связать их с вашими моделями. В каждом классе мы можем явно указать, что мы хотим разрешить делать с данными модели и какие данные выводить.

**4.2. Описание программных модулей**

При непосредственной разработке был использован компонентный подход. Компонентный подход – парадигма программирования, существенным образом опирающаяся на понятие компонента – независимого модуля программного кода, предназначенного для конструкции. Для этих целей я использовал библиотеку React, которая поддерживает данный подход. Ниже перечислены основные компоненты клиентского интерфейса.

* AddProject.js — представляет собой модальное окно, в котором содержится форма для создания нового проекта;
* AddTask.js — модальное окно, к котором будет отрисован компонент TaskAddForm;
* TaskAddForm.js — форма, которая отвечает за редактирование и создание задачи;
* Comments.js — блок с комментариями к задаче;
* Companies.js — список компаний, в которых числится пользователь
* CompanyCard.js — карточка компании
* Description.js — компонент, который отвечает за вывод описания задачи;
* InfoTable.js — блок с основной информацией по задаче;
* NotFound.js — данный компонент отрисовывает сообщение о 404 ошибке;
* ProjectDetail.js — карточка проекта
* ProjectEditForm.js — форма для редактирования и создания нового проекта;
* ProjectList.js — список проектов
* RecoverPassword.js — форма восстановления пароля
* SignInForm.js — форма авторизации
* SignUpForm.js — форма регистрации
* TaskDetail.js — служебный компонент, который отвечает за отображение Comment.js, InfoTable.js и Description.js
* TaskList.js — список задач;
* TeamBase.js — основной компонент, на котором основан весь интерфейс;
* TinyEditorComponent.js — редактор, который используется для изменения описания задачи, проекта и т. п.;
* UserProfile.js — профиль пользователя;
* WorkTimeEditModal.js — форма для отметки потраченного времени;

**5. Тестирование и отладка**

Для тестирование системы применялось функциональное тестирование — тестирование в целях проверки выполнения функциональных требований, т. е. способности программного обеспечения решать задачи, поставленные пользователем.

Для написания тестового использовался фреймворк PyTest, а также библиотека factory-boy. PyTest — это Python фреймворк, который упрощает написание тестов. Factory-boy — библиотека, целью которой является предоставление способа получения нового экземпляра класса (модели), все ещё имея возможность переопределять некоторые поля для каждого вызова. Все тесты, фабрики и вспомогательные файлы лежат в папке core/tests.

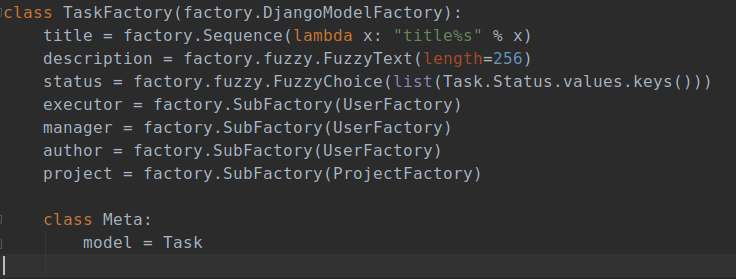


Рис. 1.12. - Пример определения «фабрики» при помощи библиотеки factory-boy

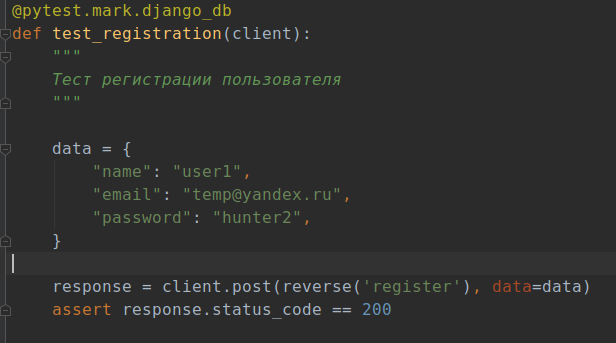


Рис. 1.13 — Тестирование механизма регистрации пользователя

Отладка производилась при помощи визуального отладчика в IDE PyCharm. Он делает процесс максимально простым, наглядно визуализируя отладку в удобном формате. Отладчик в PyCharm позволяет создавать так называемые «точки оставновки» (англ. breakpoint) — преднамеренная приостановка выполнения программы, при которой вызывается отладчик, после чего программист может исследовать состояние программы (содержимое переменных, стек вызывов функций и т.д.)

**6. Руководство пользователя**

Перед использованием системы необходимо сначала авторизоваться или зарегистрироваться (если это не было сделано ранее). После чего появится главное окно программы.

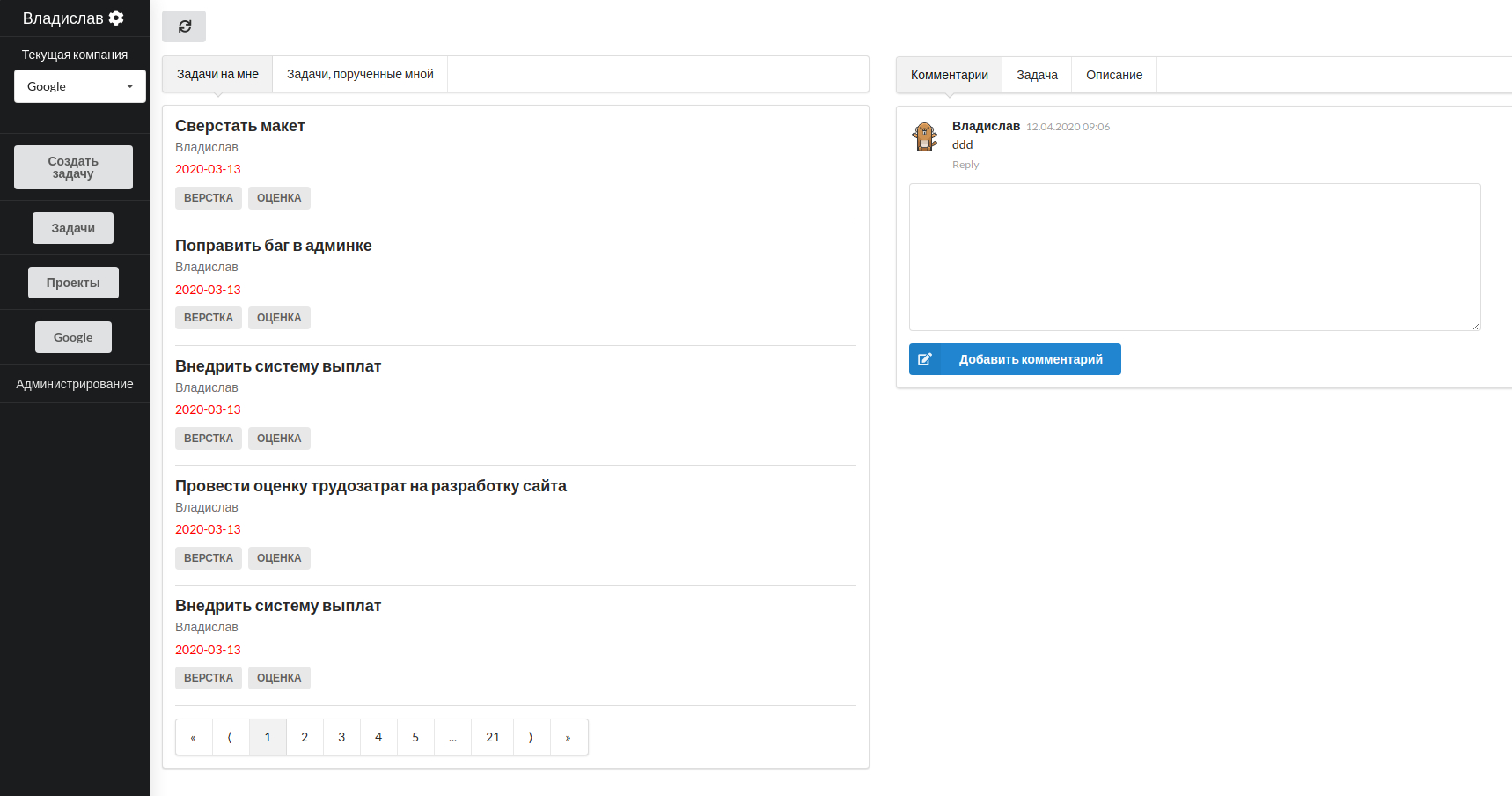


Рис. 6.1. - Главное окно программы

Данное окно состоит из двух частей — бокового меню, при помощи которого мы можем переключаться между компонентами, и основной частью,

где эти компоненты будут отрисовываться.

В самом верху бокового меню отображается имя текущего пользователя, и значок шестирёнки, при клике на который откроется профиль пользователя. В своем профиле можно загружать аватарку, добавлять / менять контактные данные, создавать новую компанию или покидать уже существующую.

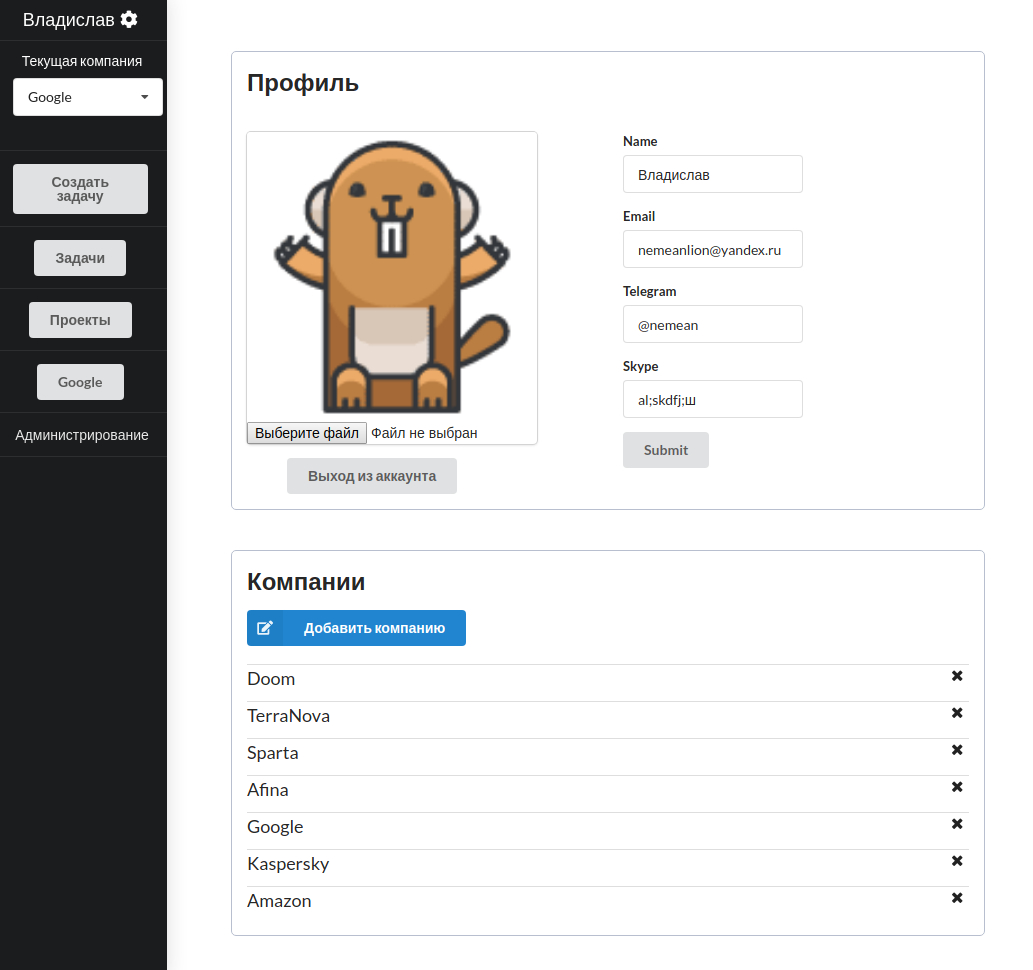


Рис. 6.2. - Профиль пользователя

При нажатии на кнопку «Создать задачу» появится модальное окно с формой, заполнив которую и нажав кнопку «Добавить» мы тем самым создадим новую задачу. После своего создания, задача появится в соответствующих списка у Исполнителя и Приёмщика.

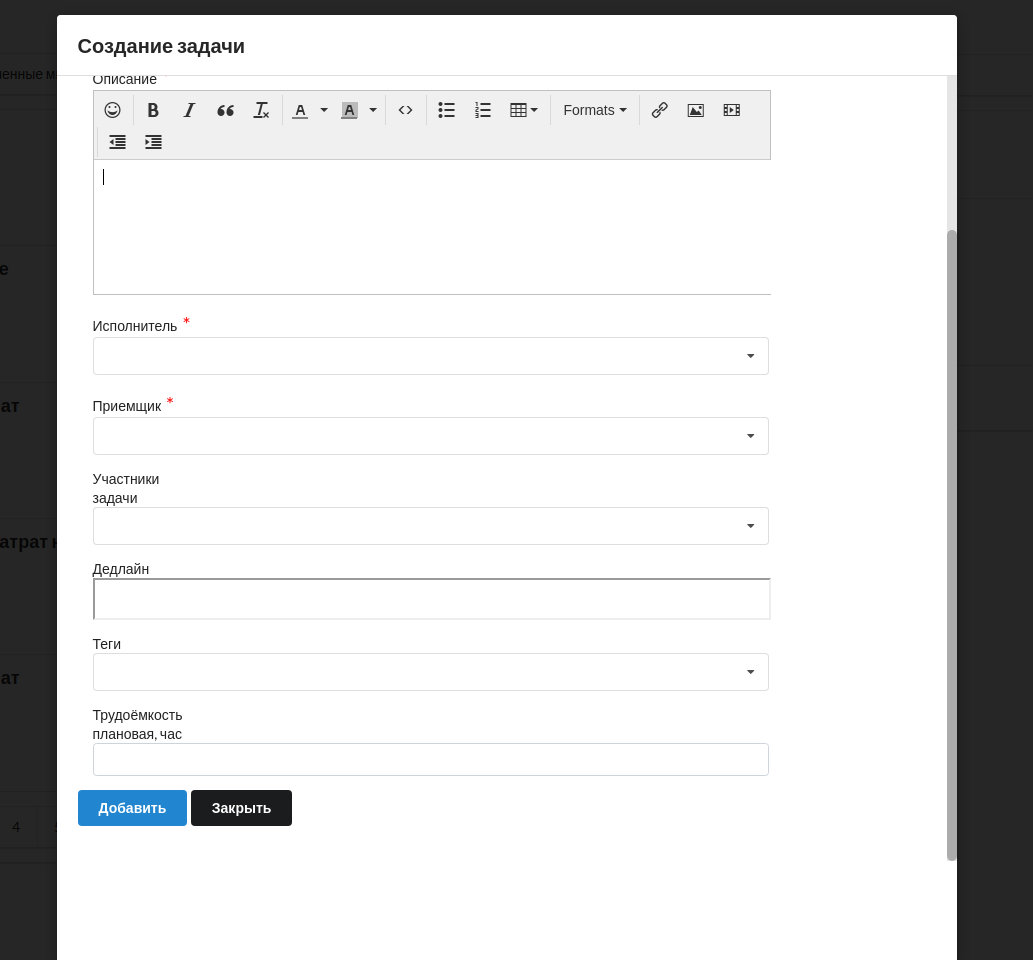
****

Рис. 6.3. - Окно создания задачи

При нажатии кнопки бокового меню «Задачи», в левой части экрана

у нас появится список задач, которые пользователь должен выполнить, а также, список задач, в которых пользователь числится приемщиком. При клике на конкретной задаче, в правой части экрана появится карточка задачи, в которой можно оставить комментарий, либо поменять какие-либо данные.

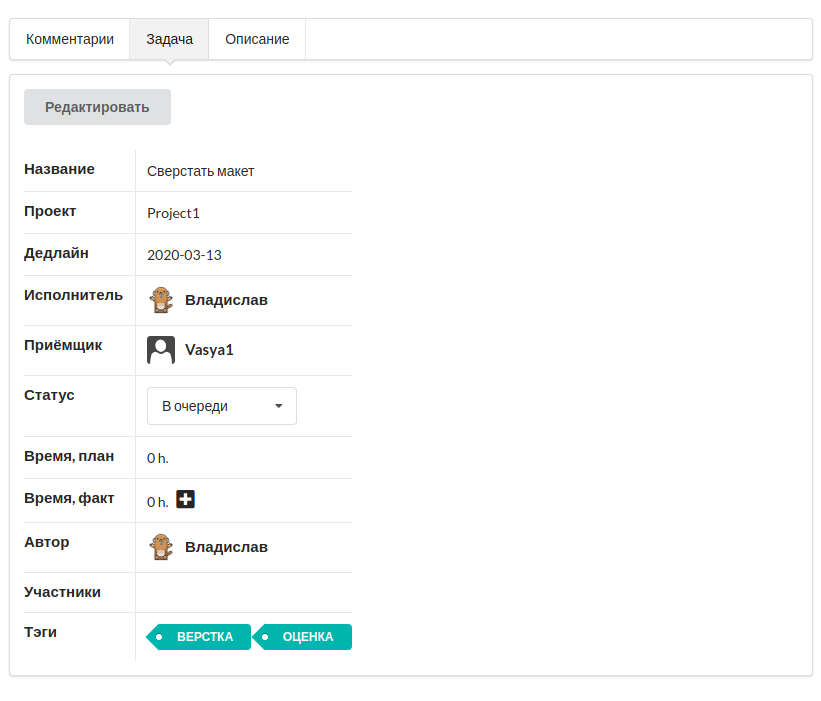
****

Рис. 6.4. - Карточка задачи

При нажатии кнопки «Проекты»отобразится раздел. Данный раздел поход на раздел для управления задачами. В нем можно просматривать, добавлять и изменять данные проектов.

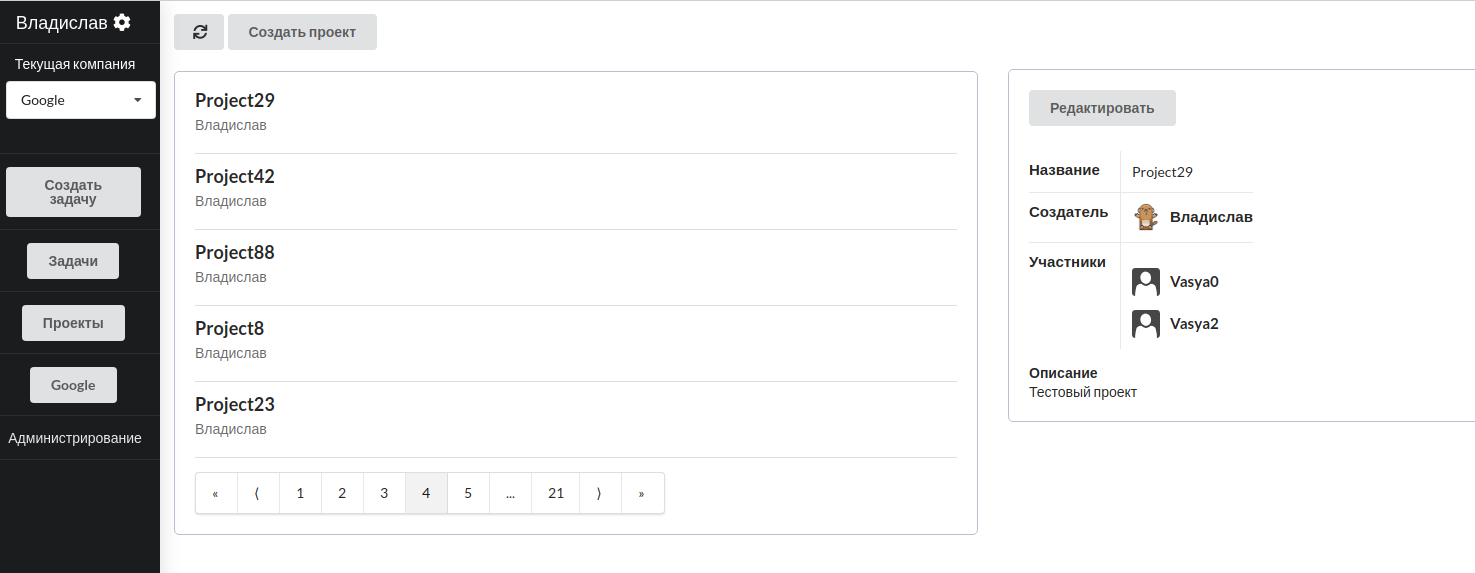
****

Рис.6.5 — Раздел «Проекты»

В нижней части бокового меню также есть кнопка, на которой указано название текущей компании. При нажатии, у нас отобразится список сотрудников данной компании. Помимо этого, там можно изменить название компании, а также пригласить новых сотрудников. После приглашения, пользователю придём на почту письмо с уведомлением.

Отдельно стоит отметить, что система имеет механизм восставновления пароля. Для этого нужно:

1. В форме авторизации кликнуть по «Забыли пароль?»
2. В появившейся форме указать email, который пользователь указал при регистрации.
3. После этого на указанный почтовый адрес придёт письмо с по ссылкой.
4. При переходе по этой ссылке появится форма, в которой уже нужно будет указать новый пароль.

**Заключение**

В ходе проделанной работы была создана система для управления проектами. Была достигнута цель: Создание веб-сервиса, предоставляющего возможность удобного распределения задач и обязанностей меду сотрудниками компании. Веб-сервис был разработан в соответствии с установленными требованиями:

* разработать систему, учитывающую все пользовательские сценарии использования приложения: авторизация, поставновка задач, отслеживание выполнения;
* обеспечение возможности масштабирования сервиса, в частности, увеличение количества разделов.

В процессе тестирования были найдены и исправлены проблемные места сервиса. Потенциал сервиса является достаточно большим, так как в его рамках можно быстро и четко ставить задачи сотрудникам и отслеживать их выполнение. В дальнейшем планируется доработка сервиса, в частности, добавление диаграммы Ганта, а также механизма выгрузки XLSX-отчетов.

**Список использованных источников**

<https://docs.python.org/3.6/>

<https://docs.djangoproject.com/en/3.0/releases/3.0/>

<https://developer.mozilla.org/ru/docs/Web/JavaScript>

<https://reactjs.org/>

<https://www.postgresql.org/docs/>

[https://www.postgresqltutorial.com](https://www.postgresqltutorial.com/)

<https://docs.pytest.org/en/latest/contents.html>