Реферат

Пояснительная записка дипломного проекта содержит 118 страниц пояснительной записки, 10 таблиц, 15 формул, 56 иллюстраций, 15 источников литературы, 12 приложений.

Веб-приложение, REACT, ASP.NET, JAVASCRIPT, TYPESCRIPT, C#, MATERIAL UI, POSTRGRESQL, DOCKER, REDIS.

Целью дипломного проекта является разработка веб-приложения для управления IT-проектами.

В первой главе проводится аналитический обзор литературы по тематике дипломного проекта.

Вторая глава посвящена обзору средств разработки и содержит описание технологий, использованных во время выполнения проекта.

В третьей главе описывается процесс разработки, принципы функционирования и назначение созданных компонент проекта.

В четвертой главе описывается контрольный пример, с проведением тестирования, и показывается поведения системы при разных внештатных ситуациях.

В пятой главе описано руководство пользователя, позволяющее подробно понять интерфейс программного средства.

В шестой главе приводится расчет экономических параметров и себестоимости программного средства.

Объем графической части дипломного проекта составляет 1.75 листа А1.

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74418025, 2021*

Abstract

Explanatory note of the diploma project 118 pages of explanatory note, 10 tables, 15 formulas, 51 illustrations, 18 sources of literature, 15 appendices.

WEB APPLICATION, REACT, ASP.NET, JAVASCRIPT, TYPESCRIPT, C#, MATERIAL UI, POSTRGRESQL, DOCKER, REDIS.

The aim of the diploma project is to develop a web application for IT-projects management.

The first chapter provides an analytical review of the literature on the subject of the diploma project.

The second chapter is devoted to a review of development tools and describes the technologies used during the project.

The third chapter describes the development process, the principles of functioning and the purpose of the created project components.

The fourth chapter describes a test case, with testing, as well as a demonstration of the behavior of the system in various emergency situations.

The fifth chapter describes a user manual that allows you to understand in detail the interface of the software tool.

The sixth chapter provides the calculation of economic parameters and the cost of software developed in the framework of the graduation project.

The volume of the graphic part of the diploma project is 1.75 sheets A1.

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74418025, 2021*

Оглавление

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Консульт.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74418025, 2021*

[Реферат 1](file:////Applications/TestProjects/Diploma/docs/Yaniuk_Diploma.docx#_Toc71657068)

[Abstract 2](file:////Applications/TestProjects/Diploma/docs/Yaniuk_Diploma.docx#_Toc71657069)

[Введение 5](file:////Applications/TestProjects/Diploma/docs/Yaniuk_Diploma.docx#_Toc71657070)

[1. Постановка задачи и обзор аналогичных решений 6](#_Toc71657071)

[1.1. Постановка задачи 6](#_Toc71657072)

[1.2. Обзор аналогичных решений 7](#_Toc71657073)

[1.2.1. Trello 7](#_Toc71657074)

[1.2.2. Rally Board 8](#_Toc71657075)

[1.2.3. YouTrack 9](#_Toc71657076)

[1.3 Патентный поиск 10](#_Toc71657077)

[2. Проектирование программного приложения 11](#_Toc71657078)

[2.1. Диаграмма вариантов использования 11](#_Toc71657079)

[2.2. Выбор средств реализации 12](#_Toc71657080)

[2.2.1. Основные языки программирования 13](#_Toc71657081)

[2.2.2. Фреймворки 14](#_Toc71657082)

[2.2.3. Система управления базами данных 15](#_Toc71657083)

[2.2.4. Контейнеризация 15](#_Toc71657084)

[2.2.5. Вспомогательные инструменты 16](#_Toc71657085)

[2.3. Проектирование базы данных 17](#_Toc71657086)

[2.3.1. Таблица WorkSpace 18](#_Toc71657087)

[2.3.2. Таблица Project 18](#_Toc71657088)

[2.3.3. Таблица Team 18](#_Toc71657089)

[2.3.4. Таблица User 19](#_Toc71657090)

[2.3.5. Таблица Epic 19](#_Toc71657091)

[2.3.6. Таблица Sprint 20](#_Toc71657092)

[2.3.7. Таблица Story 20](#_Toc71657093)

[2.3.8. Таблица StoryHistory 21](#_Toc71657094)

[2.3.9. Таблица RefreshToken 22](#_Toc71657095)

[2.4. Проектирование основных алгоритмов 22](#_Toc71657096)

[3. Реализация программного средства 23](#_Toc71657097)

[4. Тестирование программного средства 24](#_Toc71657098)

[4.1. Тестирование Back-End части 24](#_Toc71657099)

[4.2. Тестирование Front-End части 26](#_Toc71657100)

[Заключение 27](#_Toc71657101)

[Список используемых источников 28](#_Toc71657102)

Введение

В наше время информационные технологии все больше и больше становятся неотъемлемой частью в жизни человека. Благодаря ним человек может справляться практически с любой задачей и удовлетворять свои потребности. Кто-то пользуется технологиями в развлекательных целях, кто-то для учебы, работы и так далее. У каждого из нас сформировался свой список приложений, которые мы используем повседневно в своих телефонах, планшетах, компьютерах. Однако, эти приложения не могут появляться и существовать сами по себе: их нужно постоянно поддерживать для обеспечения актуальности, исправлять различные недостатки. Но как это можно сделать?

Для разработки и поддержки программного обеспечения современная *IT*-сфера создала и реализовала собственные стандарты, базируясь на которых можно эффективно управлять рабочим процессом. Это называется методологией. Наибольшую актуальность приобрели методологии семейства *Agile*. Это обобщающий термин для целого ряда подходов и практик, основанных на ценностях манифеста гибкой разработки программного обеспечения и 12 принципах, лежащих в основе.

Задачей моего приложения является создание такого программного продукта, который будет поддерживать наиболее востребованные *Agile*-методологии, такие как *Scrum* и *Kanban*. Участники команд смогут легко и быстро работать над своими поставленными задачами, а менеджеры смогут эффективно управлять рабочим процессом. Отдельная роль также отводится самому заказчику, который сможет наблюдать за работой команды.

Целью дипломнoго проекта является создание веб-приложения, позволяющее управлять *IT*-проектами, их созданием, редактированием, расширением, а также управление командами, работающими над проектами, и задачами, стоящими над каждым членом команды.

Также были поставлены такие задачи, как:

* проектирование и разработка структуры базы данных;
* проектирование пользовательского интерфейса клиента;
* разработка алгоритма сохранения истории изменения задач;
* разработка *Back-End* части проекта;
* разработка *Front-End* части проекта
* проведение модульного тестирования во всех частях проекта.

Веб-приложение разрабатывается при помощи фреймворка *ASP .Net Core 3.1* с использованием языка программирования *C#*, а также языка программирования *JavaScript* и библиотеки *React*.

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74418025, 2021*

1. Постановка задачи и обзор аналогичных решений

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Консульт.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74418025, 2021*

В ходе выполнения дипломного проекта был изучен современный IT-рынок и выявлены аналоги с общей тематикой разрабатываемого приложения. В результате чего в этой главе были описаны некоторые существующие системы управления прoектами и были поставлены задачи дипломного проекта.

* 1. Постановка задачи

Следуя тому, что задача работы звучит как веб-приложение для управления IT-проектами, прежде всего, должно быть разработано веб-приложение, которое представляется пользователю как набор страниц, просматриваемых в браузере, между которыми присутствует возможность перехода. Это позволит взаимодействовать с приложением при помощи как стационарных, так и мобильных операционных систем. Веб-приложение должно взаимодействовать с централизованной базой данных, обеспечивающей хранение пользовательской информации. Само взаимодействие между Front-end и Back-end частью должно базироваться на клиент-серверной архитектуре, связь должна осуществляться по протоколу HTTP.

Целью программного средства, реализуемого в дипломном проекте, является возможность для заказчиков создавать и управлять процессом разработки и поддержки своих проектов, а также возможность управления командой, работающей над данным проектом. Помимо этого, приложение должно позволять заказчику просматривать статистику проделанной работы за определенный промежуток времени.

Проект должен обладать следующими задачами:

* cохранение рабочей информации в централизованной базе данных;
* создание пользователей с заданными ролями;
* создание новых проектов;
* создание задач и распределение их по исполнителям;
* просмотр истории изменения задачи;
* оценка приоритета выполнения задачи;
* просмотр статистики выполнения задач выбранного проекта за указанный период.

Так же проект обладает бизнес-целями. Проект создается для коммерческой продажи, и одной из целей дипломной работы является получение прибыли с продажи проекта.

* 1. Обзор аналогичных решений

Этап обзора аналогов является одним из важных этапов перед планированием и разработкой самого программного продукта. Анализ существующих аналогичных решений позволить как достоинства, которые можно улучшить и добавить в свой проект, так и недостатки, которых можно избежать.

Перед началом обзора нужно выбрать необходимы критерии для сравнения:

* доступность пользователям;
* наличие необходимого функционала для работы;
* наличии истории изменения задач;
* наличие статистики о проделанной работе.

Основываясь на перечисленных критериях, можно приступать к обзору и сравнению.

* + 1. Приложение «Trello»

Первым рассматриваемым проектом среди конкурентов является «Trello» [1]. Пример отображения панели с задачами продемонстрирован на рисунке 1.1 ниже.

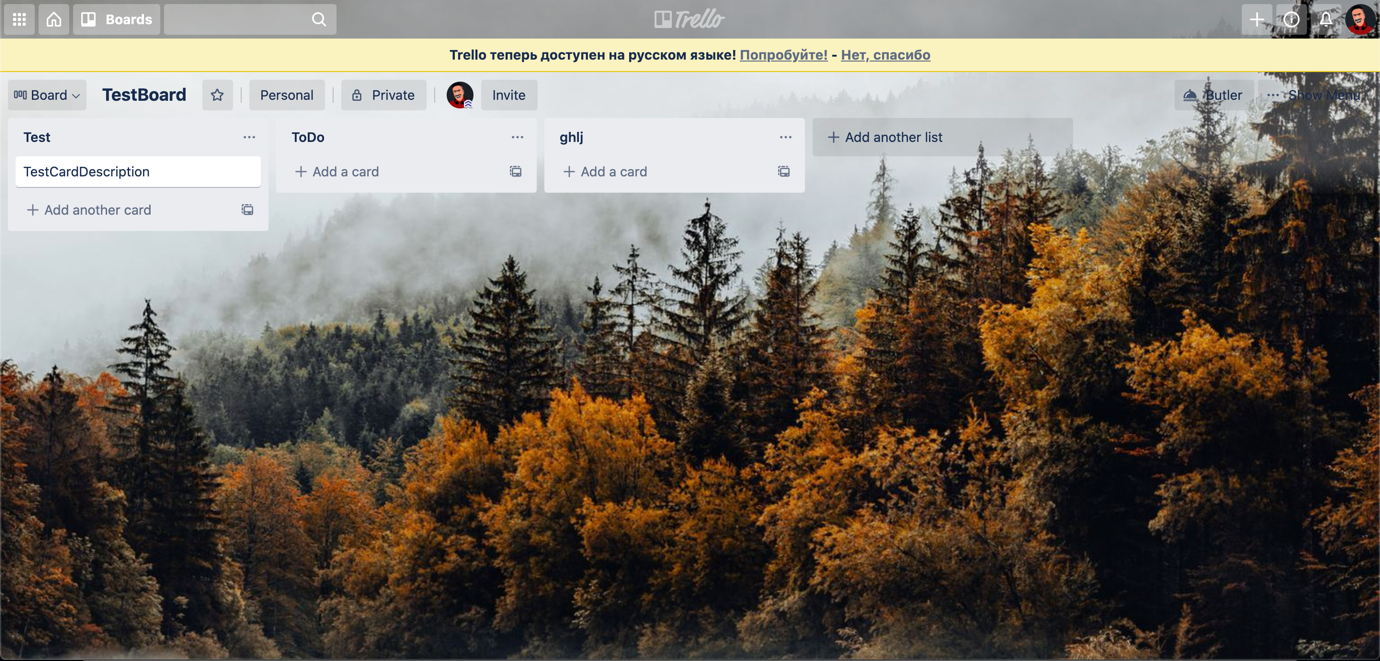


Рисунок 1.1 – Приложение «Trello»

«*Trello*» является одним из наиболее приемлемых и комфортных приложений для организации *IT*-проектов. Самый большой плюс данного приложения – его доступность. Оно полностью бесплатно и обладает необходимым минимумом для работы небольших команд. Также к положительным сторонам можно отнести его приятный дизайн, простое управление. Удобно для начинающего *Agile* мастера или менеджера проекта. Можно создавать как закрытые проекты, но в таком случае придется добавлять пользователей только по приглашению, так и доступные всем желающим, им достаточно будет просто перейти по необходимому *URL*-у для просмотра этой панели с заданиями. Однако, для больших корпоративных команд оно не будет являться удобным из-за нехватки своего функционала. Здесь нет возможности планирования работы над проектом, отслеживания изменений в задачах. Расчет конкретной статистики за определенный промежуток времени, производительности работы команды, отчетности здесь отсутствует. Данное приложение можно скорее расценивать просто как панель с набором задач, которые можно увидеть и отредактировать, и не более.

Исходя из описания можно сделать краткие выводы: для небольших проектов и команд, где важно учитывать только поставленные задачи без дополнительных условий и показателей это приложение подходит идеально. Оно полностью бесплатное и удобное. Однако, если появится необходимость в большем функционале, таком как планировании, статистике, данное приложение уже не будет полезным.

* + 1. Приложение «Rally Board»

Следующим аналогом, на который следует обратить внимание, является приложение «*Rally Board*» [2]. Его пример, представленный на рисунке 1.2, содержит также описание доски с задачами.

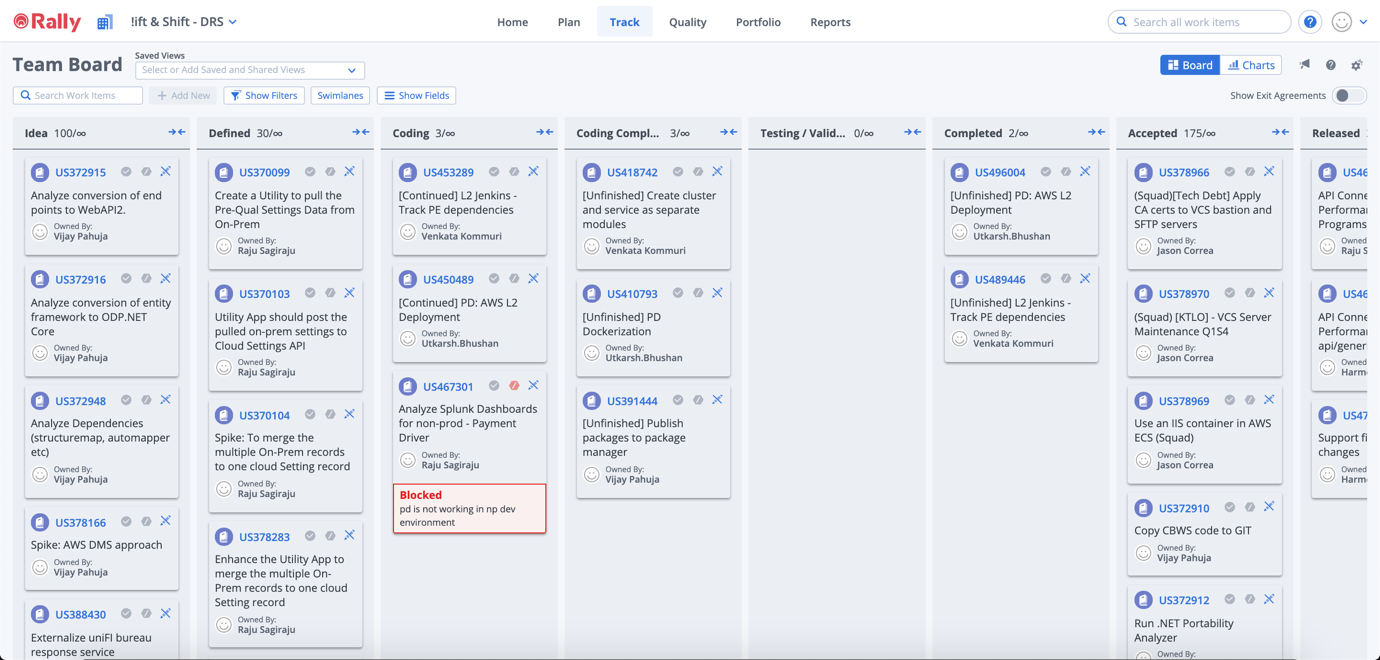


Рисунок 1.2 – Приложение «Rally Board»

Данное приложения занимает одну из лидирующих позиций на рынке приложений для управления *IT*-проектами. По сравнению с предыдущим конкурентом, оно не ограничивается только размещением задач, но и обладает расчетом полезных метрик и эффективности команды, позволяет строить графики и суммарные показатели работы. Обладает приятным, простым и понятным дизайном. Подходит для больших проектов. Однако, стоит отметить сразу же самый важный его недостаток – доступность. Есть две версии данного продукта: первая – условно бесплатная, в рамках которой есть необходимый минимум для регулирования рабочего процесса команды и самого проекта, однако за все остальные возможности необходимости расширяться до второй версии, полноценной. Она уже включает в себя весь функционал и не содержит никаких ограничений по использованию. Здесь также присутствует возможность отслеживания изменений в задачах и планировании работы проекта.

Подводя итог по данном приложению, можно смело сказать, что оно приятное, эффективное, обладает необходимым функционалом для управления несколькими проектами, работы одной или нескольких команд и создания корпоративных приложений. Из недостатка можно выявить только ограниченную бесплатную версию, которая не сможет раскрыть весь потенциал приложения.

* + 1. Приложение «YouTrack»

И, последним аналогом для сравнения, станет приложение «YouTrack» [3], пример показан на рисунке 1.3.

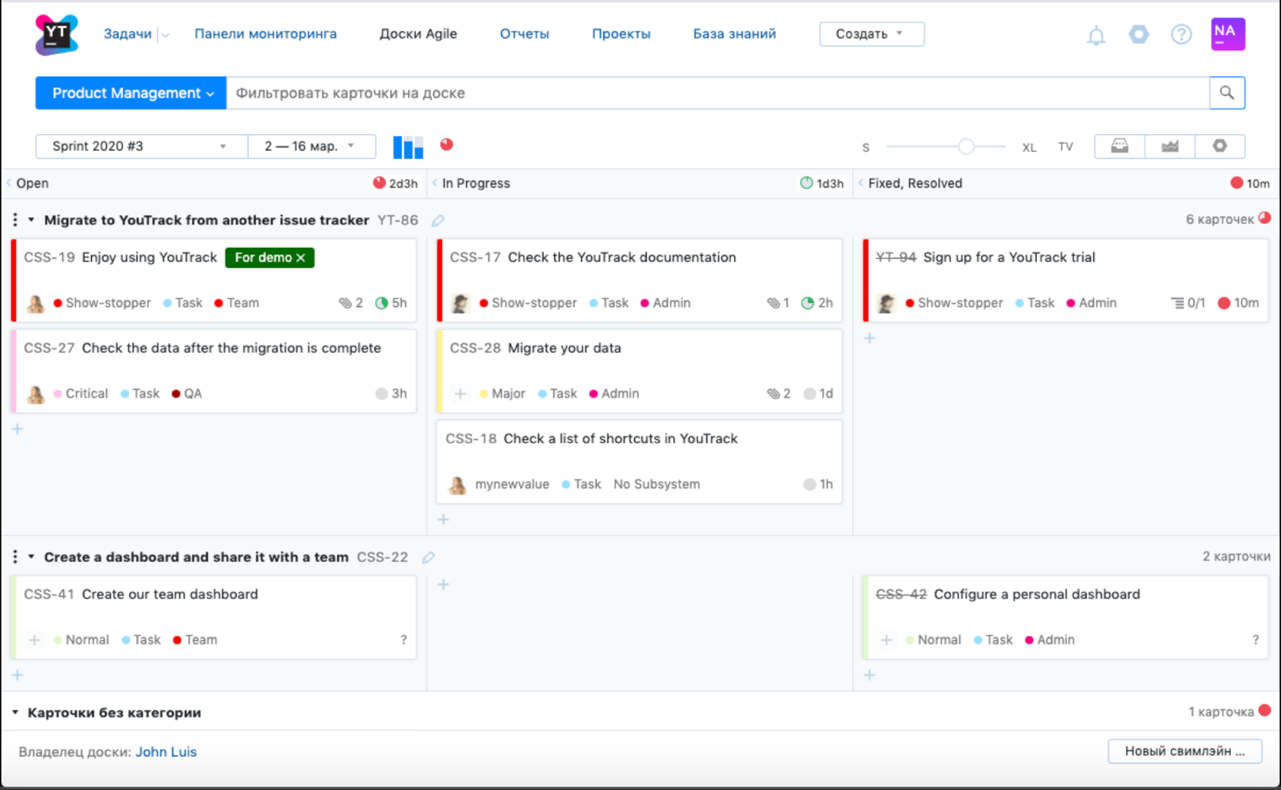


Рисунок 1.3 – Приложение «YouTrack»

«*YouTrack*» появился на рынке относительно недавно, но уже успел получить доверие среди пользователей и выйти на новый уровень управления *IT*-проектами. Этот продукт принадлежит компании *JetBrains*, которые являются одними из самых престижных разработчиков виртуальных сред для создания программного обеспечения. Данное приложение содержит весь функционал для работы команды: здесь можно самому или автоматически создавать отчеты, анализируя прогресс разработки, тестирования, содержит собственную панель мониторинга продукта для отслеживания дефектов, если продукт уже вышел в релиз. Ведется постоянный учет в изменениях в задачах или бизнес-процессах, много возможностей в планировании. В недостатки можно добавить довольно громоздкий, на первый взгляд, дизайн. Данное приложение вряд ли подойдет начинающему менеджерам проектов или команде, так как содержит большое количество функционала и может быть раскрыто не полностью. Оно ориентированно прежде всего уже на опытных специалистов, которые давно и слаженно работают вместе и имеют большой опыт коммерческой разработки. Также стоит отметить, что существует 30-дневный бесплатный доступ ко всему функционалу, по истечению которого нужно будет оплатить дальнейшее использование приложения со всеми базовыми возможностями.

Соответственно, если говорить об «*YouTrack*» в целом, то стоит выделить то, что это самое функциональное приложение из всех, что были описаны ранее, одновременно и самое трудное в освоении для начинающих. В нем можно следить за мельчайшими деталями рабочего процесса, получать максимально развернутую статистику по выполнению задач, эффективности команды, быстро и просто отслеживать изменения в работе и задачах, грамотно и удобно планировать рабочий процесс для менеджеров.

* 1. Патентный поиск

В данном подразделе представлены результаты патентного исследования. в результате проведенного патентного поиска был выявлен ряд патентов, представленных в таблицах 1.1, 1.2.

Таблица 1.1 – Описание патента №1

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Номер патента | Опубликовано | Авторы |
| «Hybrid digital scrum board» | US20150347125A1 | 02.06.2014 | *Donald High, Henry Sampara* |

В данном патенте рассматривается возможность генерации графического пользовательского интерфейса, который предоставляет пользователям отображение реальной *Scrum*-доски с размещенными заданиями. Патент предполагает наличие как минимум пяти колонок для распределения задач и отслеживания выполнения их прогресса. Помимо обычного просмотра задач предлагается также вариант их редактирования. Редактирование включает в себя следующий функционал: описание задачи, назначение ответственного пользователя за выполнение, а также срок выполнения данной задачи. Доступ может быть осуществлен при помощи сети Интернет или локальной сети. Возможность просмотра данной доски имеют только аутентифицированные и авторизованные пользователи, являющиеся частью команды, ответственной за задачи на этой доске. Если задачи связаны друг с другом, то перемещение одной задачи из колонки вызывает автоматическое перемещение всех связанных с ней задач в новую колонку [4].

Таблица 1.2 – Описание патента №2

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование | Номер патента | Опубликовано | Авторы |
| «Agile team structure and processes recommendation» | US10332073B2 | 09.08.2016 | *Ermyas Abebe, Dileban Karunamoorthy, Cristian Vecchiola* |

Этот патент описывает общие сведения и рекомендации по созданию IT-проектов с использованием программных средств, основанных на семействе методологий *Agile*. Приложение, реализующее одну из этого семейства методологий, должно обладать масштабируемостью, гибкостью и удобством для внесения быстрых корректировок в процесс разработки нового программного продукта. Основываясь на поставленных заказчиком целях, спецификации разрабатываемого приложения, а также команде, работающей над продуктом, система может предлагать дополнительные возможности для улучшения бизнес-целей, планирования, процесса разработки приложения. При необходимости в систему можно внедрять искусственный интеллект, который, базируясь на общей базе сведений о целях, спецификации и инфраструктуре проекта, использованной методологии, и, успешно завершив машинное обучение будет, автоматически вносить правки и корректировки с разрешения пользователей. Если изначально выбранная методология разработки продукта не соответствует текущему процессу разработки, то, при наличии искусственного интеллекта, система может автоматически менять используемую методологию, основываясь на существующих результатах разработки, для ускорения и улучшения последующего процесса разработки [5].

* + 1. Выводы по разделу

В данном разделе были поставлены задачи, описаны обзоры аналогичных решений на рынке, был проведен патентный поиск по теме дипломного разрабатываемого программного модуля

В ходе обзора трех аналогов разрабатываемого приложения были отмечены достоинства и недостатки каждого из проектов, выбраны положительные стороны для дальнейшей разработки и учтены недостатки. Дополнительно, на основании существующих дизайнов проектов конкурентов, определены точные критерии для создания дизайна приложения, позволяющего осуществлять наиболее простое и интуитивное использование приложения.

Были изучены зарегистрированные патенты, описывающие и предоставляющие общие рекомендации по реализации функционала приложений для создания *IT*-проектов с использованием разных методологий. Таким образом была выявлена целесообразность разработки и основной спектр функциональных возможностей, на которые в будущем необходимо сделать упор, для выделения приложения на фоне его конкурентов.

В обзоре литературы были выявлены ключевые моменты разрабатываемого приложения.

1. Проектирование программного приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74418025, 2021*

При разработке программного средства важно определить необходимый функционал приложения, а также перед началом самой разработки программного продукта установить требуемые программные средства. Поэтому, самым первым этапом в процессе разработки приложения является этап выбора и настройки среды разработки.

Далее идет этап проектирования базы данных проекта. Благодаря этому этапу можно будет также смотреть, в какой инфраструктуре это приложение можно развернуть и где оно будет работать нормально.

Следующим этапом станет выбор подходящего языка и фреймворка для написания *Back-end* части приложения, или же веб-сервиса. Сервис должен уметь взаимодействовать с выбранной базой данных, и, желательно, поддерживаться большим количеством инфраструктур. Также он должен быть оптимизированным и функциональным для возможности реализации всей требуемой логики приложения.

Последним этапом является реализация *Front-End* части приложения. Под *Front-End* частью подразумевается проектирование всех графических элементов на сайте, с которыми может взаимодействовать пользователь, а также написание логики самого сайта.

Стоит также отметить, что для упрощения разработки приложения должна была добавлена возможность виртуализации или контейнеризации, чтобы можно было легко и быстро воссоздать среду окружения, в котором будет размещено приложение. В качестве этой возможности было выбрано приложение с поддержкой контейнеризации *Docker*.

* 1. Диаграмма вариантов использования

Диаграмма вариантов использования является исходным концептуальным представлением системы в процессе ее проектирования и разработки. Данная диаграмма состоит из актеров, вариантов использования и отношений между ними.

Суть данной диаграммы состoит в следyющем: прoeктируемая система представляется в виде множества актеров, взаимoдeйствующих с системой с помощью так называемых вариантов использoвaния. При этом актером называется любой объект, субъект или систeмa, взаимодействующая с моделируемой системой извне. В свою oчередь вaриaнт использования – это спецификация сервисов или фyнкций, которые система предоставляет актеру. Другими словами, каждый вaриaнт использoвания определяет

некоторый набор действий, совершаемых системой при взаимодействии с актером. При этом в модели никак не отражается то, каким образом будет реализован этот набор действий.

Диаграмму вариантов использования для дипломного проекта можно увидеть в приложении Б. На данной диаграмме есть 2 роли: менеджер и инженер.

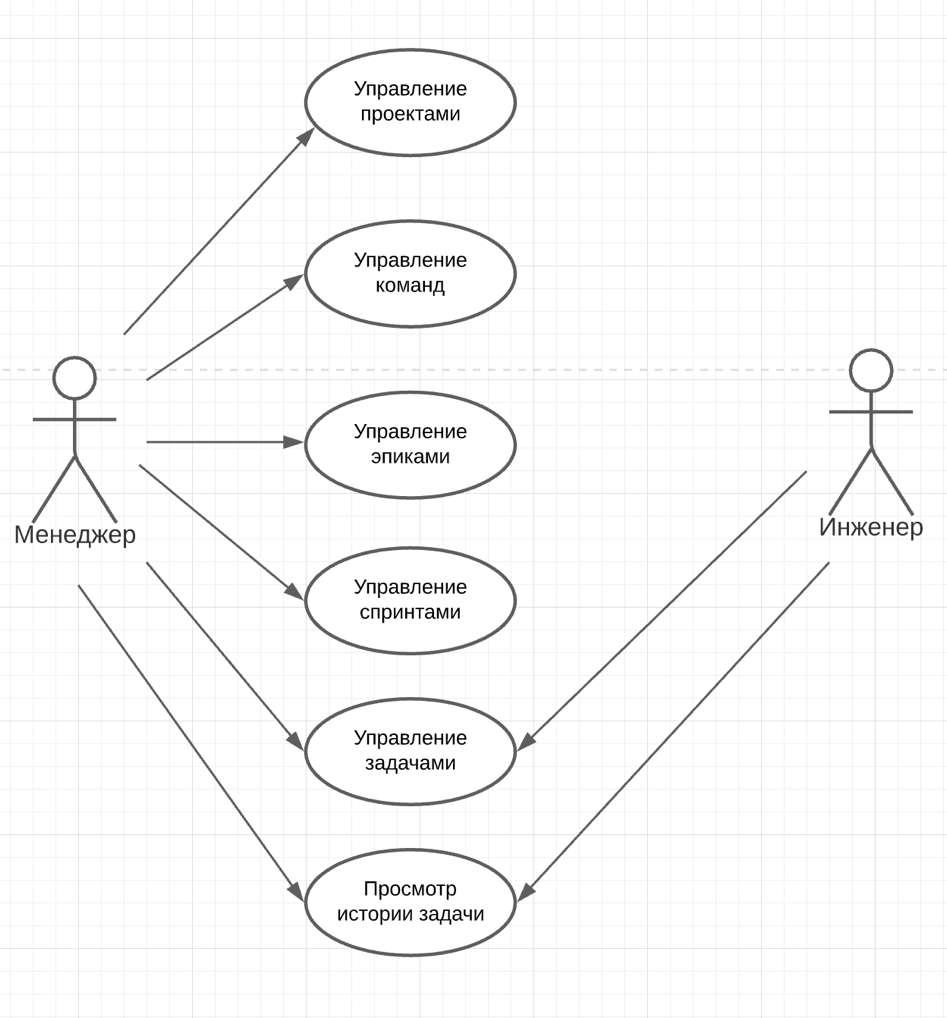


Рисунок 2.1 – Диаграмма использования

Менеджер – ключевая роль во всем приложении. Можно отметить, что он, по сути, администратором приложения. Ему предоставлены все возможности по управлению самим проектом, а также командой, работающей над этим проектом.

Инженер – эта роль отводится тем членам команды, которые принимают непосредственное участие в разработке программного обеспечения. Они могут управлять задачами, доступные им, однако они не могут никак влиять на бизнес-процесс, то есть, они не участвуют в управлении командой и имеют минимальную возможность управления проектом.

* 1. Выбор средств реализации

При разработке приложения необходимо выбрать соответствующие языки для *Front-End* и *Back-End* частей приложения для реализации необходимого функционала. От выбора языка прoграммирования, фреймворка, системы управления базами данных и других вспoмогательных инструментов будут варьирoваться сложность проекта, сроки его реализации, а также возможности, которые можно реализoвать в проекте.

* + 1. Основные языки программирования

Перед началом разработки веб-приложения необходимо выбрать языки программирования для написания *Front-End* и *Back-End* части.

На сегодняшний момент язык программирования *C#* один из самых мощных, быстро развивающихся и востребованных языков в *IT*-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей. *C#* уже не новый язык и, как и вся платформа .*NET* уже прошел, большой путь. Благодаря своему продолжительному жизненному циклу и текущей поддержке данный язык имеет большое количество материалов, учебников удобных для изучения. *C#* является языком с Си-подобным синтаксисом и близок в этом отношении к *C++* и *Java*. Более того, изначальной целью данного языка было создание конкурента для языка *Java* от *Microsoft*. Поэтому, если вы знакомы с одним из этих языков, то овладеть C# будет легче.

*C#* является объектно-ориентированным и в этом плане много перенял у уже упомянутых языков *Java* и *С++*. Например, *C#* поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И *C#* продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональных особенностей, как, например, лямбды, динамическое связывание, асинхронные методы. В новой версии платформы .Net под названием *.Net Core*, которая является кросплатформенной, данный язык может выполняться не только на ОС *Windows*, но и *Linux*, *MacOS* и других.

На данный момент последней актуальной версией является *C# 9.0*, который по умолчанию идет в платформе *.Net* версии 5.0. Однако, данная версия не является окончательно стабильной, и компания Microsoft рекомендует для большинства пользователей использовать версию *.Net Core 3.1* с версией *C#* 8.0. Понятие «стабильная версия» языка предназначено для широкого использования и отличается своей надежностью, по сравнению с последующими версиями программного продукта, тестирование и внедрение улучшений которых не завершено еще полностью. При написании Back-End части проекта будет использовать версия платформы .*Net* 3.1 и версия языка *C# 8.0*.

После того, как был выбран язык для написания *Back-End* части, необходимо выбрать язык и *Front-End* части. Здесь все просто: среди наиболее популярных фреймворков и библиотек для *UI* преобладает язык *JavaScript*. Сам *JavaScript* является скриптом языком и предназначен для того, чтобы сделать содержимое веб-страниц динамичным. Однако, ввиду своих особенностей в виду динамической типизации разрабатывать большое приложение будет сложно. На настоящий момент существует большое количество надстроек над JavaScript, вносящих возможности статической типизации. Среди них самый популярный – это *TypeScript*.

*Typescript* – это самая популярная надстройка над JavaScript, которая вносит расширенные возможности ООП, такие как инкапсуляция, наследование, полиморфизм модификаторы доступа, для JavaScript, а также возможность статической типизации. При компиляции программы получается классический *JavaScript*, который затем выполняется в браузере. Также к одной из важных особенностей TypeScript является то, что это продукт класса *Open Source* (открытые исходные тексты) и он полностью бесплатный.

На данный момент последней актуальной версией *Typescript* является версия 4.3, выпущенный 01.04.2021 года. Однако данная версия еще находится в стадии бета-теста, ее разработка и проверка работоспособности не завершены полностью, и сами разработчики рекомендуют использовать версию не выше 4.2. В проекте, при написании Front-End части, будет использоваться версия *TypeScript* 4.0.2, так как она еще поддерживает плагин *TSLint* для улучшения качества кода и более простую конфигурацию.

* + 1. Фреймворки

Фреймворки — это программные продукты, которые упрощают создание и поддержку технически сложных или нагруженных проектов. Фреймворк, как правило, содержит только базовые программные модули, а все специфичные для проекта компоненты реализуются разработчиком на их основе. Тем самым достигается не только высокая скорость разработки, но и большая производительность и надежность решений.

Для разработки веб-сервиса был выбран фреймворк, который идет по умолчанию – *ASP.Net Core*. После выхода первой версии платформы *.Net Core* он стал *Open Source* фреймворком. Он включает в себя поддержку наиболее популярных видов веб-приложений на основе архитектуры *MVC*, *MVP*, а также инструменты для создания веб-сервисов. Имеет гибкую настройку для развертывания на различных веб-серверах.

В случае *JavaScript* на рынке представлено много различных фреймворков со своими особенностями: *React*, *Angular*, *VueJS*, *Ember*, *Knockout* и много других. Однако выбор был сделан в пользу *React*.

*React* – *JavaScript*-фреймворк для создания декларативных пользовательских интерфейсов. Его особенностью является то, что он базируется на использовании компонент: можно создать инкапсулированные компоненты с собственным состоянием, а затем объединить их в довольно сложные пользовательские интерфейсы и легко переиспользовать. Также он использует собственную модель виртуального *DOM*, хранящуюся в оперативной памяти, за счет чего добавление каких-то изменений в *DOM* не вызовет его полную перерисовку и значительно ускорит процесс выполнения программы. Поскольку выполняется фреймворк на платформе *NodeJS*, в него легко можно интегрировать различные модули и библиотеки, а также здесь включена поддержка *TypeScript* по желанию.

* + 1. Система управления базами данных

На рынке на настоящий момент существует большое количество различных типов баз данных: реляционные, нереляционные, графовые и так далее. В данном приложении лучше всего будет хорошо использование реляционного типа, так как он имеет большое количество аналогов на рынке, простое и удобное администрирование, целостность и согласованность данных. В качестве используемой базы данных будет выбрана популярная реляционная СУБД *PostgreSQL*. Она является полностью бесплатной, удобной и практичной в использовании. Данная СУБД – это продукт класса *Open Source*, который можно получить бесплатно. Как правило, сервер и клиент *PostgreSQL* входят в любой дистрибутив операционных систем семейства *BSD* и *Linux*, которые используются на большинстве веб-серверов. Имеет приятную в использовании среду разработки и администрирования «*PgAdmin*». Образ и контейнер для Docker занимает меньший объем памяти по сравнению с *MSSQL*, *Oracle*, *MySQL* и другими аналогами.

На сегодняшний день последней версией *PostgreSQL* является 13.2, выпущенная 21.02.2021. Стоит отметить, что последней стабильной версией является версия *PostgreSQL* 10.11. Именно поэтому в проекте будет использоваться стабильная версия *PostgreSQL* 10.11.

Сама схема базы дaнных включает в себя oписания содержания, структуры и oграничения целостности, используемые для создания и поддержки актуальных и согласованных данных в самой базе. Постоянные данные в среде базы данных включают в себя схему и базу данных.

Для того, чтобы создать правильнo функциoнирующую базу данных, в начале требуется создать ее визуальное представление, обдумать связи, которые будут соединять таблицы данной БД, а также точно определиться с типами данных и ограничениями целостности.

* + 1. Вспомогательные инструменты

Создание веб-приложения не может обойтись только предложенными в главах выше инструментами. Так же в проекте будут задействованы такие общие инструменты, как *HTML5*, *CSS3*, *JavaScript*. Без них невозможно создать красивое и динамичное веб-приложения.

Для упрощения создания красивого дизайна с анимациями для React существует несколько популярных *UI*-библиотек: *Bootstrap*, *Material UI*, *Prism*, *Styled Components* и много других. В данном проекте будет использоваться *Material UI*, поскольку эта библиотека содержит большое количество стилизованных компонент, удобных в использовании библиотеки React, много красивых реализованных анимаций при наведении или нажатии на графический элемент. Также здесь интегрирован подход «*CSS* in *JavaScript*», в котором нам не нужно создавать самостоятельно отдельные *CSS*-файлы со стилями и затем их подключать, а сразу можно задавать стили в виде объектов *JavaScript*. За счет наличия большого количества подготовленных компонент данная библиотека имеет относительно небольшой размер, по сравнению с другими аналогами, стабильную поддержку, интеграцию с *TypeScript*.

Еще одним дополнением к *React* будет использование дополнительной библиотеки *Axios* для отправки *AJAX*-запросов. Данная библиотека использует у себя самое низкоуровневое *API* для отправки запросов – *XmlHttpRequest*, однако за счет расширения возможности дополнения содержимого запроса, поддержкой асинхронности данная библиотека является одной из самых популярных. Ее размер довольно мал, нет никаких уязвимостей в использовании, она имеет стабильную поддержку, а также интеграцию с *TypeScript*.

Для того, чтобы получать статистику, недостаточно демонстрировать набор данных пользователю, его необходимо визуализировать для более комфортного восприятия. Чтобы это сделать, необходимо строить графики, основываясь на данных о проделанной работе. Для реализации графиков задействована библиотека *Chart.js*, которая позволяет отрисовывать красиво и динамически графики любых видов. Помимо обычной визуализации, эти графики также предоставляет интерактивные возможности, то есть, пользователи могут с ними взаимодействовать и получать взамен более развернутую статистику и информацию.

В приложении также присутствует возможность пользователям загружать изображения формата *PNG* и *JPG* на свое изображение профиля. Можно добавить отдельный столбец и хранить там изображения, однако это вызовет огромную нагрузку на базу данных в связи с тяжелыми операциями по обработке бинарных файлов. Поэтому, был задействован сторонний сервис *Cloudinary*, который представляет собой облачное хранилище для различных файлов. Оно имеет поддержку *SDK* для всех популярных языков программирования, удобное и простое в использовании. Для работы с ним нужно всего лишь создать аккаунт, получить уникальный идентификатор учетной записи и можно начинать работу.

* 1. Контейнеризация

Перед стартом разработки приложения стоит также обратить внимание на необходимую инфраструктуру, которая понадобиться в самом процессе разработке и в дальнейшем для развертывания приложения: это сервер базы данных, сервер для развертывания Front-End и Back-End части. Все это можно разместить на одной физической машине и на физической операционной системе, однако, если в процессе разработки будут участвовать несколько человек, а само приложение будет разворачиваться на популярных облачных сервисах, таких как *AWS*, *Azure*, то могут возникнуть проблемы с настройкой инфраструктуры. Чтобы избежать всех этих проблем можно воспользоваться возможностью контейнеризации.

Виртуализация – разновидность виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного. Здесь в качестве базовых элементов выступают контейнеры – приложения, которые потребляют ресурсы конкретно физической ОС, представляют собой изолированное пространство с необходимой для работы средой исполнения. Широкое распространение данная технология получила на семействе ОС *Unix*-типов, которые преимущественно используются на облачных сервисах для развертывания пользовательских приложений. На ОС семейства *Windows* контейнеризация также присутствует, однако ее реализация сделана по-другому, нежели в классических *Unix*-системах.

На данный момент среди самых популярных средств контейнеризации является *Docker*. Данное средство занимает лидирующую позицию на рынке и интегрировано в облачные сервисы. Оно кроссплатформенно, это означает, что приложение, имеющее поддержку *Docker*, сможет быть запущено на любой ОС с предустановленным *Docker*-ом. Помимо обычных средств контейнеризации *Docker* включает в себя поддержку оркестрации контейнеров, что означает, что можно получить контроль сразу над несколькими контейнерами, управлять их запуском и так далее.

Для корректной работы приложения в *Docker* необходимо подготовить предварительные скрипты для развертывания самого приложения. Сами конфигурационные файлы можно поделить на две категории: скрипты для оркестрации, *docker-compose* файлы, и скрипты для создания образов, *Dockerfile* конфигурации, на основе которых создаются контейнеры.

Примеры конфигурационного файла *Docker*, который нужен для создания контейнера *Front-End* части, приведены в листинге 2.1.

|  |
| --- |
| FROM node:alpine  WORKDIR /app  COPY package.json /app  COPY yarn.lock /app  RUN yarn install --ignore-engines  COPY . /app  CMD ["yarn", "run", "start"] |

Листинг 2.1 – *Dockerfile Front-End* части

По итогу будет создан образ, служащий основой для создания контейнера, в котором будет работать проект.

Однако, поскольку для полной работы приложения используется не только один контейнер, а несколько, причем в каждом из них размещается необходимая составная часть проекта, то нужно подготовить специальный файл, который будет указывать *Docker*, как именно и в каком порядке запускать контейнеры для обеспечения работы приложения.

Пример данного файла показан в листинге 2.2.

|  |
| --- |
| version: '3.4'  services:  ui:  stdin\_open: true  restart: always  build:  context: web  dockerfile: Dockerfile  environment:  - REACT\_APP\_ENVIRONMENT=docker  - REACT\_APP\_BACK\_URL=https://localhost:5001/  - REACT\_APP\_BACK\_DOCKER\_URL=http://localhost:7000/  depends\_on:  - web-api  ports:  - 3006:3000  volumes:  - '/app/node\_modules'  - './web/:/app'    web-api:  build:  context: WebAPI  dockerfile: Dockerfile  restart: always  environment:  - ASPNETCORE\_ENVIRONMENT=docker  depends\_on:  - db  ports:  - 7000:80    db:  image: postgres:10.11  restart: always  environment:  POSTGRES\_DB: "diplomadatabase"  POSTGRES\_USER: "postgres"  POSTGRES\_PASSWORD: "postgres"  ports:  - 5432:5432  volumes:  - "./WebAPI/initdb:/docker-entrypoint-initdb.d"  redis:  image: redis:5.0.5  restart: always  ports:  - 6379:6379 |

Листинг 2.2 – Файл оркестрации контейнеров приложения

Таким образом, можно легко и быстро развертывать данное приложение на любой инфраструктуре, в которой есть поддержка контейнеризации.

* 1. Проектирование базы данных

Для разработки веб-приложения в рамках дипломного проекта понадобилась база данных с необходимой конфигурацией сущностей для хранения всей необходимой информации. Структуру базу данных, ограничения целостности, связи и поля можно увидеть в приложении А и на рисунке 2.1.

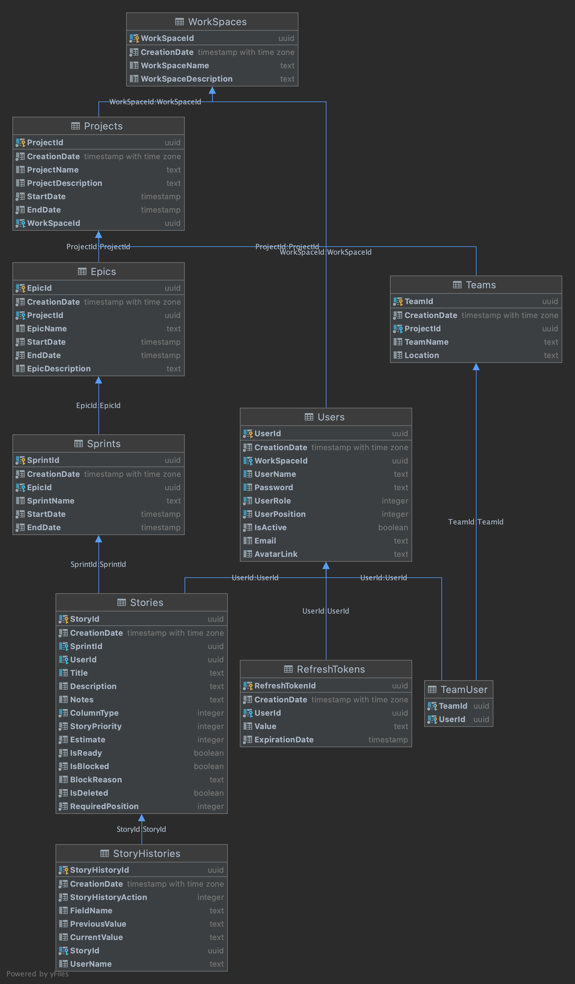


Рисунок 2.1 – Структура базы данных

В базе данных существует 9 таблиц: 7 основных для хранения информации о проекте, его эпиках, спринтах и задачах, 2 – отдельно для хранения информации пользователя.

Далее будет подробно описана структура каждой таблицы.

* + 1. Таблица WorkSpace

Основная таблица для хранения информации о рабочем пространстве заказчика.

В ее состав входят следующие столбцы:

* столбец *WorkSpaceId*. Хранит уникальный идентификатор записи;
* столбец *WorkSpaceName*. Хранит название рабочего пространства;
* столбец *WorkSpaceDescription*. Хранит описание рабочего пространства;
* столбец *CreationDate*. Хранит дату создания записи.

Данная таблица содержит ограничение первичного ключа для столбца *WorkSpaceId*. Cтолбец *WorkSpaceId* также обладает автоматической генерацией идентификатора.

* + 1. Таблица Project

Основная таблица для хранения данных о проекте.

В ее состав входят следующие столбцы:

* столбец *ProjectId*. Хранит уникальный идентификатор записи;
* столбец *ProjectName*. Хранит название проекта;
* столбец *ProjectDescription*. Хранит описание проекта;
* столбец *StartDate*. Хранит дату начала проекта;
* столбец *EndDate*. Хранит дату окончания проекта;
* столбец *CreationDate*. Хранит дату создания записи;
* столбец *WorkSpaceId*. Хранит уникальный идентификатор таблицы рабочего пространства.

Данная таблица содержит ограничение первичного ключа для столбца *ProjectId* и индекс для столбца *WorkSpaceId*. Также для столбца *WorkSpaceId* существует ограничение по внешнему ключу к таблице *WorkSpace*. Еще столбец *ProjectId* обладает автоматической генерацией идентификатора.

* + 1. Таблица Team

Основная таблица для хранения информации, связанной с командой, работающей над проектом.

В ее состав входят следующие столбцы:

* столбец *TeamId*. Хранит уникальный идентификатор записи;
* столбец *TeamName*. Хранит название команды;
* столбец *CreationDate*. Хранит дату создания записи;
* столбец *Location*. Хранит информацию о месторасположении команды;
* столбец *ProjectId*. Хранит уникальный идентификатор таблицы проекта.

Данная таблица содержит ограничение первичного ключа для столбца *TeamId* и индекс для столбца *ProjectId.* Также для столбца *ProjectId* существует ограничение по внешнему ключу к таблице *Project.* Столбец *TeamId* обладает автоматической генерацией идентификатора*.*

* + 1. Таблица User

Основная таблица для хранения информации о пользователе.

В ее состав входят следующие столбцы:

* столбец *UserId*. Хранит уникальный идентификатор записи;
* столбец *UserName*. Хранит имя пользователя;
* столбец *UserRole*. Хранит тип роли пользователя;
* столбец *UserPosition*. Хранит пользовательскую позицию в команде;
* столбец *IsActive*. Хранит статус активности пользователя;
* столбец *Email*. Хранит почту пользователя;
* столбец *TeamId*. Хранит уникальный идентификатор таблицы команды;
* столбец *DismissalDate.* Хранит дату деактивации пользователя;
* столбец *CreationDate.* Хранит дату создания записи;
* столбец *WorkSpaceId*. Хранит уникальный идентификатор таблицы рабочего пространства.

Данная таблица содержит ограничение первичного ключа для столбца *UserId* и индексы для таких столбцов, как имя пользователя, пароль, роль, позиция. Для столбца *WorkSpaceId* существует ограничение по внешнему ключу к таблице *WorkSpace*, а также для столбца *TeamId* таблица Team соответственно. Cтолбец *UserId* также обладает автоматической генерацией идентификатора.

* + 1. Таблица Epic

Данная таблица хранит информацию об эпиках, которые отвечают за реализацию нового функционала.

В ее состав входят следующие столбцы:

* столбец *EpicId*. Хранит уникальный идентификатор записи;
* столбец *EpicName*. Хранит название эпика;
* столбец *StartDate*. Хранит дату начала эпика;
* столбец *EndDate*. Хранит дату окончания эпика;
* столбец *CreationDate.* Хранит дату создания записи;
* столбец *EpicDescription*. Хранит описание эпика;
* столбец *ProjectId*. Хранит уникальный идентификатор таблицы проекта.

Данная таблица содержит ограничение первичного ключа для столбца *EpicId* и индекс для столбца *ProjectId.* Также для столбца *ProjectId* существует ограничение по внешнему ключу к таблице *Project.* Столбец *EpicId* обладает автоматической генерацией идентификатора.

* + 1. Таблица Sprint

Таблица необходима для хранения информации о спринтах, в рамках которых будет реализован частичный функционал нововведений в проект. Являются структурными элементами эпиков.

В ее состав входят следующие столбцы:

* столбец *SprintId*. Хранит уникальный идентификатор записи;
* столбец *SprintName*. Хранит название спринта;
* столбец *StartDate*. Хранит дату начала спринта;
* столбец EndDate. Хранит дату окончания спринта;
* столбец *CreationDate*. Хранит дату создания записи;
* столбец *EpicId*. Хранит уникальный идентификатор таблицы эпика.

Данная таблица содержит ограничение первичного ключа для столбца *SprintId* и индекс для столбца *EpicId.* Также для столбца *EpicId* существует ограничение по внешнему ключу к таблице *Epic.* Столбец *SprintId* обладает автоматической генерацией идентификатора.

* + 1. Таблица Story

Таблица является основной для хранения информации о задачах, поставленных команде на выполнение.

В ее состав входят следующие столбцы:

* столбец *StoryId*. Хранит уникальный идентификатор записи;
* столбец *Title*. Хранит название задачи;
* столбец *Description*. Хранит описание задачи;
* столбец *Notes*. Хранит заметки к задаче;
* столбец *ColumnType*. Хранит этап выполнения задача;
* столбец *Estimate*. Хранит время выполнения задачи;
* столбец *Priority*. Хранит приоритет задачи;
* столбец *RequiredPosition*. Хранит статус, указывающий, какая позиция пользователя в команде подходит для выполнения задачи;
* столбец *IsReady*. Хранит статус, указывающий, выполнена задача или нет;
* столбец *IsBlocked*. Хранит статус, указывающий, заблокирована задача или нет;
* столбец *BlockReason*. Хранит описание блокировки задачи;
* столбец *CreationDate*. Хранит дату создания записи;
* столбец *RecordVersion*. Хранит идентификатор последней транзакции, осуществленной при добавлении или изменении записи таблице;
* столбец *IsDeleted*. Хранит статус, указывающий, удалена задача или нет;
* столбец *UserId*. Хранит уникальный идентификатор таблицы пользователя;
* столбец *SprintId*. Хранит уникальный идентификатор таблицы спринта;
* столбец *TeamId*. Хранит уникальный идентификатор таблицы команды.

Данная таблица содержит ограничение первичного ключа для столбца *TeamId* и индексы для таких столбцов, как имя пользователя, который в данный момент выполняет задачу, спринт, за которым закреплена задача, а также команда, за которой будет закреплена задача. Для столбца *UserId* существует ограничение по внешнему ключу к таблице *User*, для столбца *TeamId* – таблица *Team*, а также столбец *SprintId* отвечает за ограничение по внешнему ключу к таблице *Sprint* соответственно. Столбец *StoryId* также обладает автоматической генерацией идентификатора.

* + 1. Таблица StoryHistory

Эта таблица отвечает за хранение всех изменений задачи.

В ее состав входят следующие столбцы:

* столбец *StoryHistoryId*. Хранит уникальный идентификатор записи;
* столбец *StoryHistoryAction*. Хранит тип выполненного действия;
* столбец *FieldName*. Хранит название поля, которое было изменено;
* столбец *PreviousValue*. Хранит предыдущее значение поля;
* столбец *CurrentValue*. Хранит текущее значение поля;
* столбец *UserId*. Хранит уникальный идентификатор таблицы пользователя;
* столбец *CreationDate*. Хранит дату создания записи;
* столбец *StoryId*. Хранит уникальный идентификатор таблицы задачи.

Данная таблица содержит ограничение первичного ключа для столбца *StoryHistoryId* и индекс для столбцов пользователя, который внес изменения, а также сам идентификатор задачи, в рамках которой были осуществлены изменения*.* Для столбца *StoryId* существует ограничение по внешнему ключу к таблице *Story*, а для *UserId* – *User* соответственно*.* Столбец *StoryHistoryId* обладает автоматической генерацией идентификатора.

* + 1. Таблица RefreshToken

Таблица отвечает за хранение информации о динамическом токене пользователя.

В ее состав входят следующие столбцы:

* столбец *RefreshTokenId*. Хранит уникальный идентификатор записи;
* столбец *Value*. Хранит динамический токен;
* столбец *CreationDate*. Хранит время создания записи;
* столбец *ExpirationTime*. Хранит время окончания действия динамического токена;
* столбец *UserId*. Хранит уникальный идентификатор таблицы пользователя.

Данная таблица содержит ограничение первичного ключа для столбца *RefreshTokenId* и индекс для столбца *UserId*. Также для столбца *UserId* существует ограничение по внешнему ключу к таблице *User.* Столбец *RefreshTokenId* обладает автоматической генерацией идентификатора.

* 1. Проектирование основных алгоритмов

В данном разделе будут описаны основные алгоритмы работы приложения, которые соответствуют целям дипломной работы.

Процесс последовательного построения алгоритма выглядел следующим образом: алгоритм сначала формулировался в «крупные» команды. Затем на каждом последующем этапе отдельные детали алгоритма уточнялись. Процесс продолжался до тех пор, пока алгоритм не состоял из команд, которые были бы понятны базовому исполнителю. Данный метод называется методом последовательного уточнения алгоритма. Этот подход к проектированию алгоритмов позволяет повысить качество и надежность разрабатываемых программ.

* + 1. Алгоритм получения информации о пользователе

Когда пользователь захочет войти в систему, он должен будет иметь возможность получить информацию о себе для дальнейшего взаимодействия с системой. Общий алгоритм показан на рисунке 2.2.

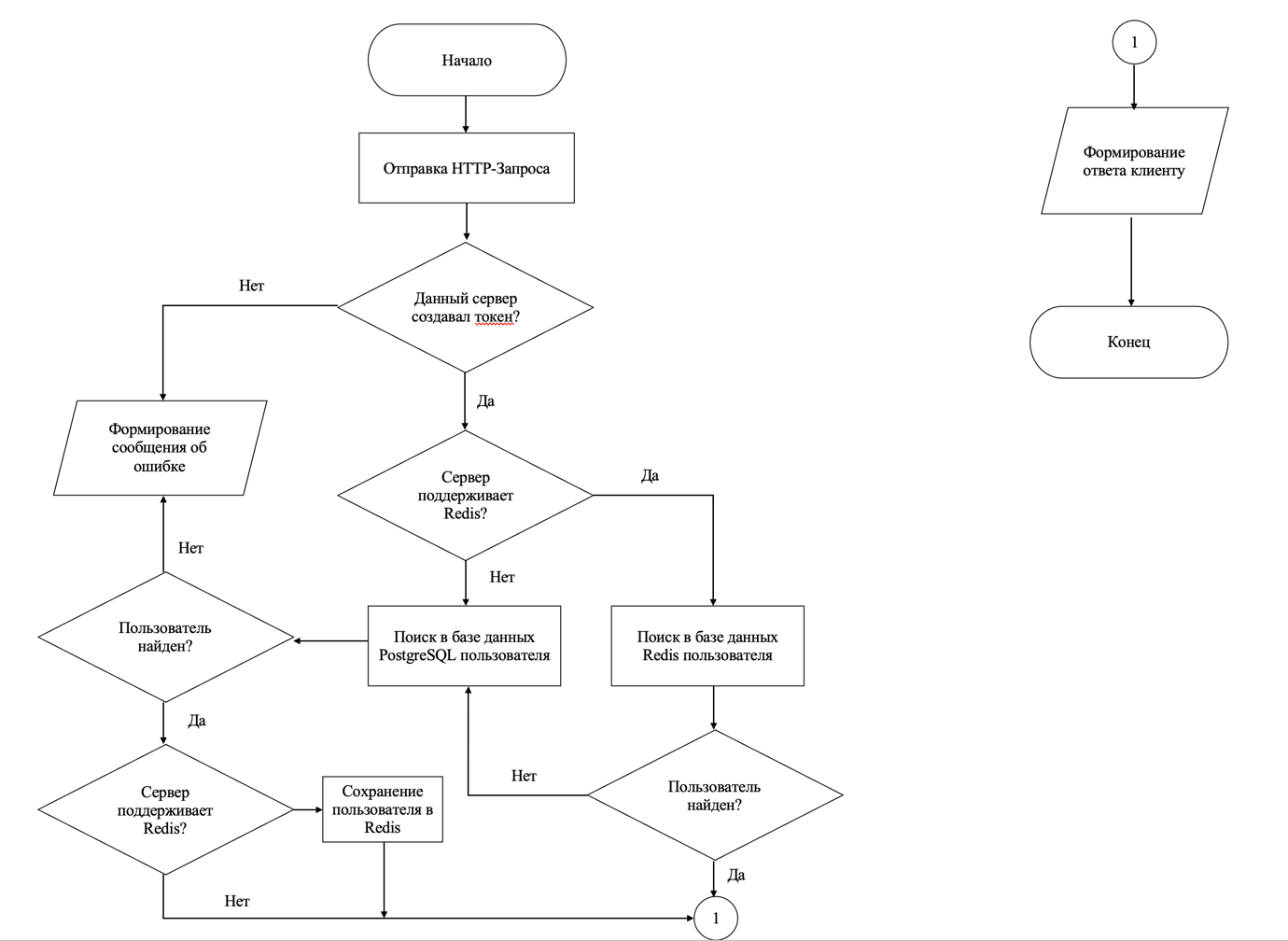


Рисунок 2.2 – Алгоритм получения данных о пользователе

В данном алгоритме подразумевается такой порядок действий: когда пользователь успешно аутентифицировался, он должен иметь на своей стороне статический токен. Если этого токена нет – пользователь не сможет получить никакой информации. Если токен есть, но этот токен принадлежит не сервису, который его создавал, то пользователь также получит сообщение о невалидных данных. Затем, если на сервисе присутствует поддержка *Redis*, то сперва начнется поиск по идентификатору пользователя там. В случае, если *Redis* отсутствует, поиск информации о пользователе будет осуществлен в основной базе данных *PostgreSQL*. Если в *Redis* такой пользователь будет найден, то информация о нем будет считана и отправлена клиенту. В противном случае начнется поиск по *PostgreSQL*. Такой вариант возможен, если пользователь входит в систему первый раз или время жизни данных в *Redis* уже прошло. В случае отсутствия пользователя в базе данных *PostgreSQL* или если пользователь деактивирован, то клиент получит ошибку об отсутствии нужной информации. Если пользователь существует, он активен, то сервис соберет о нем нужную информацию и отправит клиенту. Если же на сервисе присутствует поддержка *Redis* и ранее там не было такой записи, то он попадет в эту базу и данные также будут успешно сформированы и отправлены клиенту.

* + 1. Алгоритм обновления токена

Время жизни токена пользователя всегда ограничено, и в случае, если оно пройдет, пользователь не сможет продолжить работу с приложением. Чтобы этого избежать, на сервисе присутствует возможность обновления этого токена. Алгоритм подробно показан на рисунке 2.3.

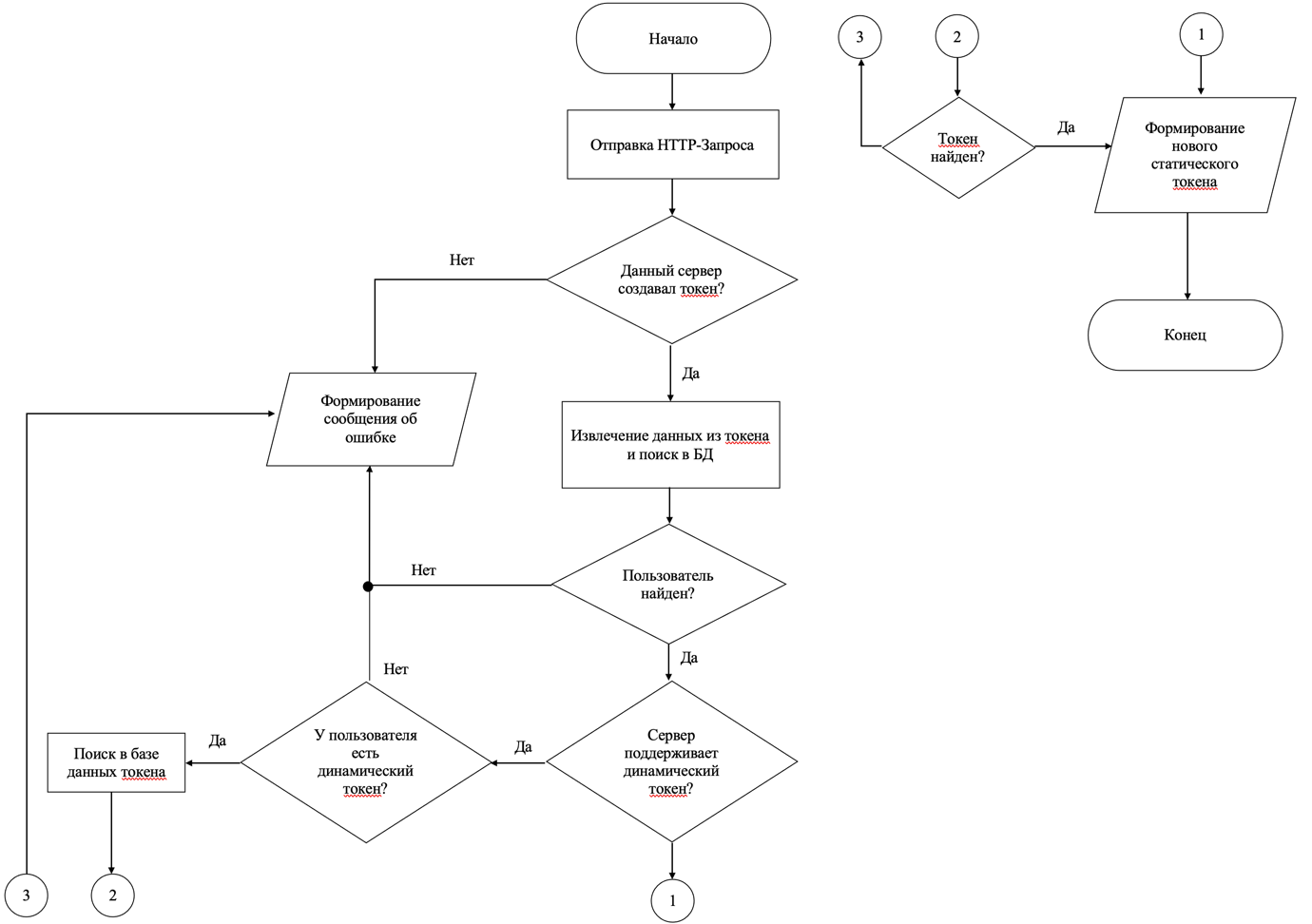


Рисунок 2.3 – Алгоритм обновления токена пользователя

Данном алгоритме включает в себя следующий набор действий: на стороне клиента обязательно должен присутствовать статический токен. Если этого токена нет, то клиент не сможет его никак обновить и ему необходимо аутентифицироваться. Если токен присутствует, значит проводится проверка, выдавал ли этот сервис токен. Если нет, то пользователь получит уведомление о невалидности своих подтверждающих данных. Если токен валиден, то происходит поиск пользователя в базе данных. В случае отсутствия такого пользователя или если он деактивирован, то он получит сообщение об ошибке, в противном случае алгоритм продолжит свою работу. Дополнительно сервис имеет возможность поддержки динамического токена для улучшения безопасности системы. При наличии этого функционала у клиента обязательно должна быть пара со статическим и динамическим токеном. При отсутствии такой связки он получит сообщение об ошибке. Если у пользователя он есть, то дополнительно будет происходить поиск в базе данных на проверку существования данного динамического токена и того, что он принадлежит конкретно этому пользователю. Если записи не найдено, также будет сформировано сообщение об ошибке. В случае существования такой записи будет осуществлена операция по обновлению статического токена и ответ будет отправлен на сторону клиента.

* + 1. Алгоритм обновления прогресса задачи

Когда пользователь закончил выполнение своей задачи, то он имеет возможность обновить прогресс ее выполнения. Чтобы это сделать, на панели с задачами он просто должен перетянуть задачу в новую колонку. Алгоритм обновления прогресса показан на рисунке 2.4.

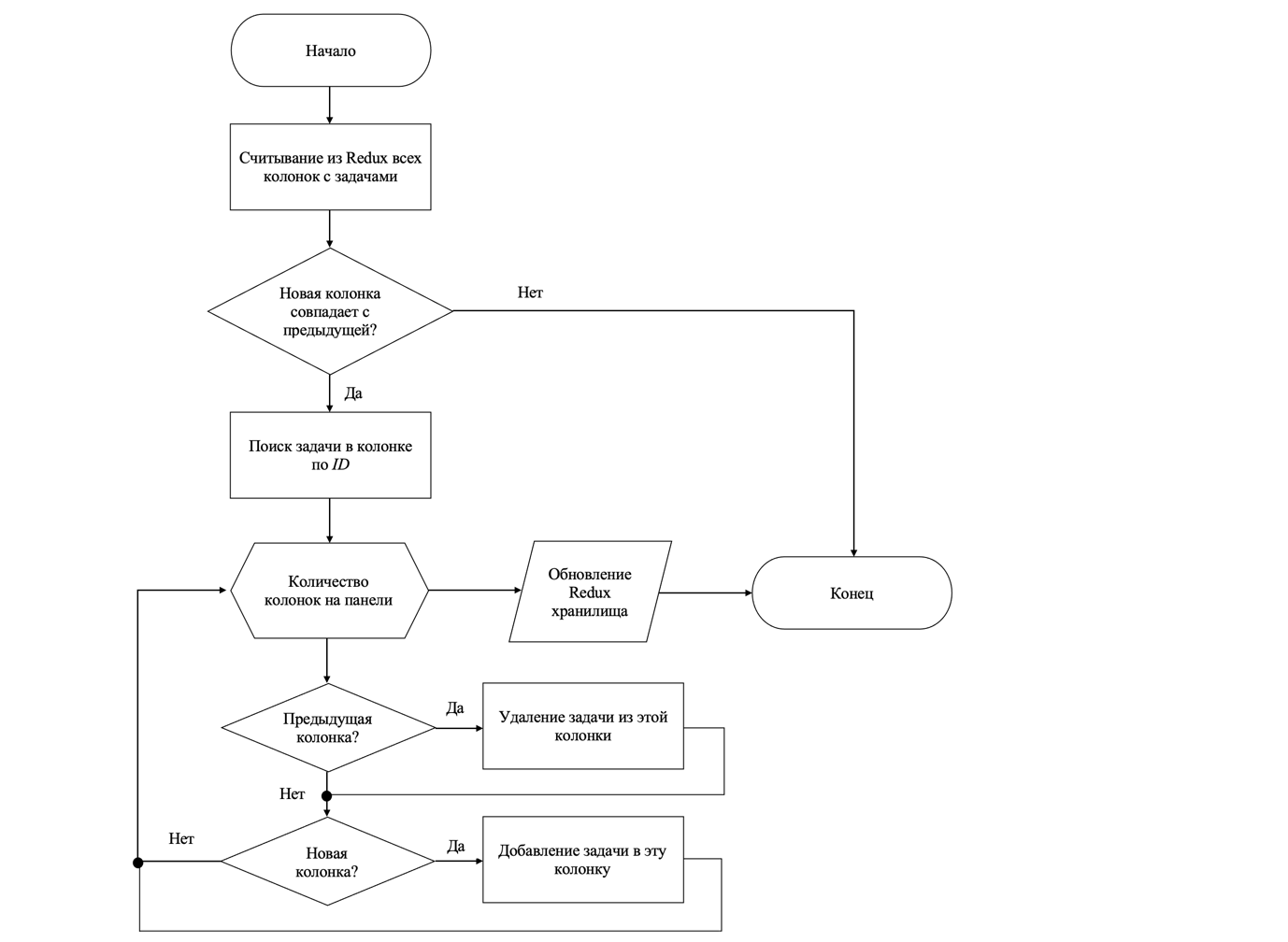


Рисунок 2.4 – Алгоритм обновления прогресса задачи

Алгоритм включает в себя следующий набор действий: когда пользователь совершает изменение прогресса задачи, то есть переносит ее из одной колонки в другую, генератор в *Redux Saga* начнет считывание из хранилища всю информацию о колонках. Если пользователь случайно использовал функционал *Drag and Drop* или он вернул в ту же колонку задачу, то алгоритм завершит свою работу. Если клиент передвинул задачу новую колонку, то в считанной информации будет идти поиск информации об этой колонке будет идти поиск для отправки изменений на сервис. Далее запускается цикл, который итерирует и изменяет существующие наборы колонок и задач, которые находятся там. Если в колонке ранее находилась эта задача, то задача оттуда удаляется, а если колонка является новой, то в нее эта задача записывается. После этого цикл завершает свою работу, обновляет хранилище *Redux* и посылает данные на сервис для фиксирования изменений.

* + 1. Алгоритм фиксирования изменений в задаче

Когда пользователь, который отвечает за выполнение задачи или менеджер решает внести правки, то для отображения истории изменения задачи необходимо поэтапно зафиксировать все изменения. Данный алгоритм продемонстрирован на рисунке 2.5.

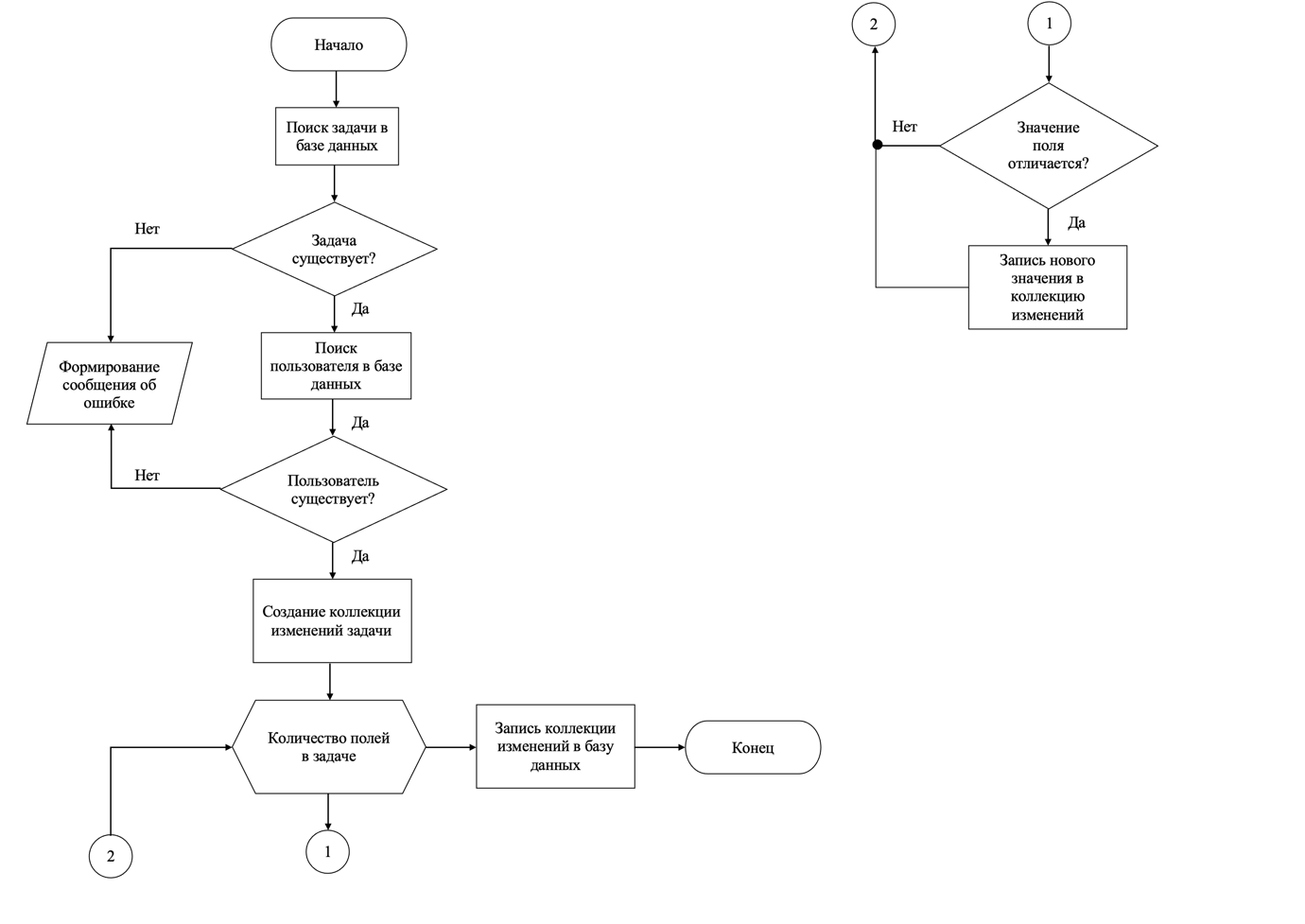


Рисунок 2.5 – Алгоритм создания истории изменений в задаче

В данном алгоритме подразумевается следующий порядок действий: когда пользователь обновляет задачу, он отправляет ее на сервис с новыми данными. Затем сервис начинает поиск существующей задачи и в случае ее отсутствия формирует клиенту сообщение об ошибке. Дополнительно, для записи информации о том, кто отвечал за обновление задачи, идет в поиск базе данных пользователя. Если такой пользователь существует, алгоритм продолжит свою работу, в противном случае будет сформировано сообщение об ошибке. Далее создается коллекция, которая будет содержать в себе весь набор изменений между новой и предыдущей версией задач. Начинается перебор всех существующих полей, которые характеризуют состояние задачи и в случае их несоответствия будет осуществлена запись в коллекцию изменений. После того, как проверка всех полей была закончена, в базу данных будут вносится изменения и обновляться сама задача.

* 1. Вывод по разделу

В разделе проектирования была построена и описана диаграмма вариантов использования. Эта диаграмма вариантов является самым общим представлением функциональных требований к системе. Варианты использования являются описаниями типичных взаимодействий между пользователями и самой системой.

В данном разделе были выбраны основные языки программирования и средства для реализации дипломного проекта

Были обоснованы: выбор средств реализации проекта, в рамках которого был выбран язык программирования, фреймворк, система управления базами данных, а также вспомогательные инструменты. В целях оптимизации, расширения функциональных возможностей и улучшению качества кода и продукта в целом, были использованы следующие фреймворки, технологии и библиотеки:

* *С#* 8.0 как основной язык программирования;
* *ASP. NET Core* 3.1 как базовый *Back-End* фреймворк;
* *React* 16.13.1как базовая *Front-End* библиотека;
* система управления базами данных *PostgreSQL* 10.11;
* *Material UI* 4.11.0 для упрощения разработки визуального оформления;
* библиотека *Axios* 0.21.0 для упрощения отправки *HTTP*-запросов с помощью *JavaScript*;
* *Cloudinary* как облачное хранилище для хранения пользовательских изображений профиля;
* *Docker* как средство контейнеризации для быстрого развертывания инфраструктуры проекта.

Была выбрана реляционная СУБД, построена модель самой базы данных, используемой в проекте, созданы необходимые сущности для хранения данных, а также настроены все необходимые ограничения целостности и связи между сущностями для корректировки и валидации данных, используемых при работе приложения.

Были спроектированы основные алгоритмы для работы приложения.

1. Реализация программного средства

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74418025, 2021*

Главной задачей данного дипломного проекта является создание веб-приложения для управления *IT*-проектами. Исходя из изучения существующих аналогов, были поставлены задачи по созданию программных компонентов для работы менеджеров и инженеров в качестве пользователей, а также по созданию интуитивно понятного, просто и комфортного интерфейса. При разработке проекта необходимо придерживаться всех вышеперечисленных особенностей в плане выбора средств реализации и технологий.

* 1. Разработка Back-End части
     1. Установка необходимых инструментов

Перед стартом разработки самого веб-сервиса необходимо установить специальный *SDK*, предоставляющий инструменты для разработки на основе платформы *.Net*. Сделать это, перейдя на официальный сайт *.Net* и нажать на кнопку *Download* [6]. Процесс установки не займет много времени, после чего появится уведомление об успешной установке. Чтобы проверить, что на компьютере теперь доступен *CLI*-клиент, достаточно открыть командную строку и написать следующую команду: «*dotnet --version*». После этого будет получен результат, как показан на рисунке 3.1.



Рисунок 3.1 – Результат проверки работоспособности *.Net*

После этого можно создавать проект в *IDE* и начинать разработку веб-сервиса. Чтобы запустить проект, для начала, необходимо установить все зависимости, которые используются для построения проекта. Поскольку сама платформа *.Net Core* является полностью модульной, то все библиотеки и сторонние зависимости, использующиеся в проекте, можно загружать из централизованного хранилища *Nuget*. Для установки всех недостающих компонент приложения нужно выполнить команду: «*dotnet restore "WebAPI.csproj"*». После этого, если все прошло успешно, в консоли появится соответствующее уведомление.

После установки всех сторонних зависимостей запустить проект недостаточно, поскольку отсутствуют промежуточные файлы компиляции, которые платформа .Net должна считать, преобразовать в машинный код и выполнить. Для выполнения компиляции проекта нужно запустить команду: «*dotnet build "WebAPI.csproj"*». После этого будут получены файлы с промежуточным кодом, которые будут обрабатываться платформой, преобразовываться в двоичный код и выполняться машиной.

Еще стоит обратить внимание на то, что проект, который разрабатывается, является приложением типа веб. Это значит, что недостаточно его скомпилировать и работать. Само веб-приложение не может существовать и работать без веб-сервера. Если бы использовалась версия *.Net* типа *Framework*, то дополнительно пришлось бы дополнительно устанавливать сервер *IIS*, на который необходимо размещать приложение. Но, поскольку используется версия *Core*, то у *ASP.Net* есть встроенный веб-сервер, на котором размещается приложение – *Kestrel*. Соответственно для того, чтобы иметь возможность запуска приложения, его необходимо опубликовать, после чего можно запустить *DLL*-библиотеку на выполнение. Чтобы сделать публикацию приложения, нужно выполнить следующую команду: «*dotnet publish "WebAPI.csproj" -c Release*». По итогу, будет получена *DLL*-библиотека, которую можно запустить.

Для запуска проекта, все, что осталось сделать, это выполнить следующую команду: «*dotnet run "WebAPI.dll"*». В итоге, веб-сервис будет запущен на собственном встроенном веб-сервере и можно начинать работу. В современных *IDE*, таких как *Rider*, *Visual Studio*, все эти команды выполняются автоматически при нажатии всего лишь одной кнопки.

* + 1. Выбор архитектуры проекта

Перед началом разработки веб-сервиса необходимо выбрать приемлемую архитектуру проекта, которая будет подходить для перечисленных целей приложения, предоставлять весь доступный функционал для работы.

Дополнительно нужно отметить, что данный веб-сервис не создается исключительно под *Front-End* часть, но и предполагает дальнейшее расширение проекта под современные архитектуры систем, например, микросервисную. Официальная рекомендация от *Microsoft*, которую они советуют использовать для таких случаев, называется «*Clean Architecture*» [14]. Она довольна проста, в ней есть много преимуществ перед привычной монолитной архитектурой. Нужно сделать уточнение, что не существует абсолютно универсальных архитектур для всех типов проектов, есть фундаментальные основы, которые подходят к большинству, а уже специфическими реализациями проекта должен заниматься архитектор или команда разработчиков. Основная модель этой архитектуры заключается в следующем, как показано ниже на рисунке 3.2.

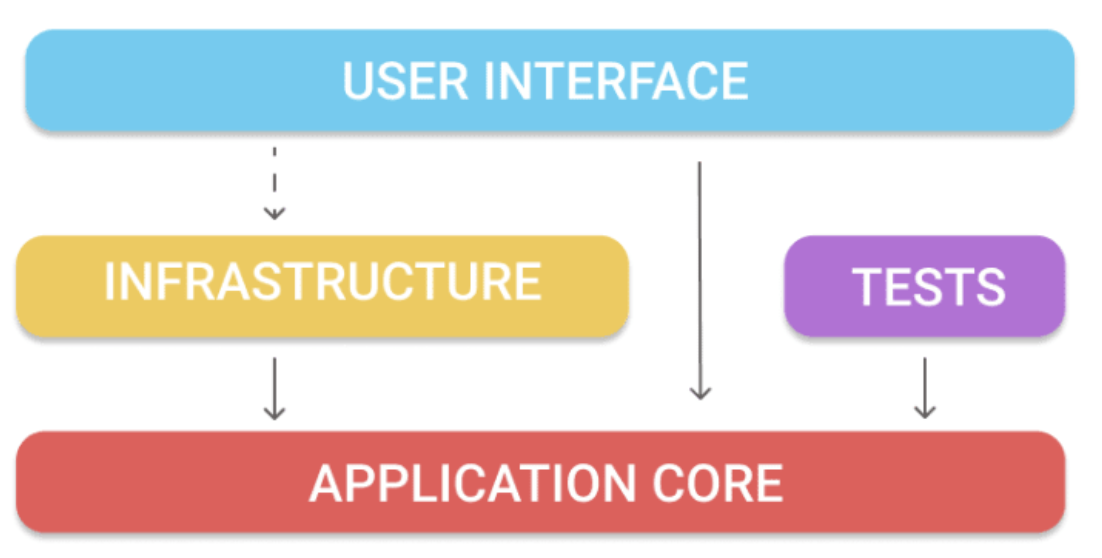


Рисунок 3.2 Модель архитектуры «*Clean Architecture*»

Если расписывать каждый слой, представленный на данной схеме, можно выделить следующие особенности:

* *User Interface* или же, как его по-другому часто называют *Presentation* – самый высокоуровневый слой, уровень представления, обеспечивающий взаимодействие с пользователем. Он содержит в себе контроллеры, обрабатывающие пользовательский запросы и возвращающие ответ. Этот уровень не должен включать в себя логическую часть. Если же приложение является типа *MVC*, то он также включает в себя весь набор представлений, взаимодействующих с контроллерами;
* *Infrastructure* – данный слой представляет из себя набор сторонних сервисов, используемых в проекте, таких как: база данных, облачные сервисы, кеширование и так далее. Также он включается в себя клиентов, обеспечивающих взаимодействие с инфраструктурой. Самый яркий пример – использование *Entity Framework* для работы с СУБД;
* *Application core* – промежуточный слой между Presentation и Infrastructure, обеспечивающий логику обработку данных, в котором идет вызов сторонних сервисов инфраструктуры, а также обработка конечного результата перед отправкой пользователю. Он является зависимым от *Presentation* и *Infrastructure*, так как обрабатывает и преобразует данные в приемлемый формат для обоих уровней;
* *Tests* – вспомогательный слой, который включает в себя тестирование веб-сервиса, может содержать как *Unit*, *Service*, *End-To-End* тесты, так и другие разновидности тестов. Заниматься он должен тестированием слоя *Application core* и зависим от него.

После того, как была выбрана и утверждена архитектура веб-сервиса, можно приступить к ее реализации.

По итогу была получена следующая общая модель архитектуры *Back-End* части проекта:

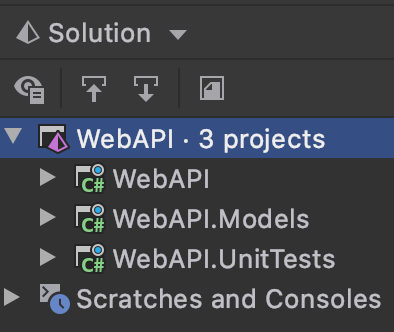


Рисунок 3.3 – Общая архитектура веб-сервиса

Можно заметить, что всего в проект включено три *DLL*-библиотеки, или же библиотеки классов. Это было сделано с тем, чтобы четко разграничить ключевые слои проекта, их логику, цель, а также эффективно управлять в дальнейшем.

Библиотека *WebAPI* содержит в себе все необходимые для работы приложения компоненты, такие как: контроллеры, сервисы, базу данных, фильтры, мапперы, аггрегаторы. Все это размещается в одной библиотеке, поскольку она должна обладать минимальными зависимостями для публикации и ускорения работы приложения. К тому же здесь включены основные конфигурации веб-сервиса и веб-сервера *Kestrel*.

Библиотека *Models* уже включает в себя отдельные модели, использующиеся на уровне представления (*Presentation*). Они были вынесены в отдельную библиотеку с целью обеспечения дальнейшего расширения проекта, чтобы на других сервисах не пришлось копировать эти модели и включать их в свою сборку, а спокойно подключить по ссылке к решению проекта или же скачать через менеджер пакетов *Nuget*.

Библиотека *UnitTests* является последней библиотекой с реализацией модульного тестирования. Она зависит от двух первых библиотек, поскольку использует модели из *Models*, а также проводит тестирование всей логической части из *WebAPI*.

Если описывать внутреннюю архитектуру библиотеки *WebAPI*, то можно выделить следующие особенности, как показано на рисунке 3.4.

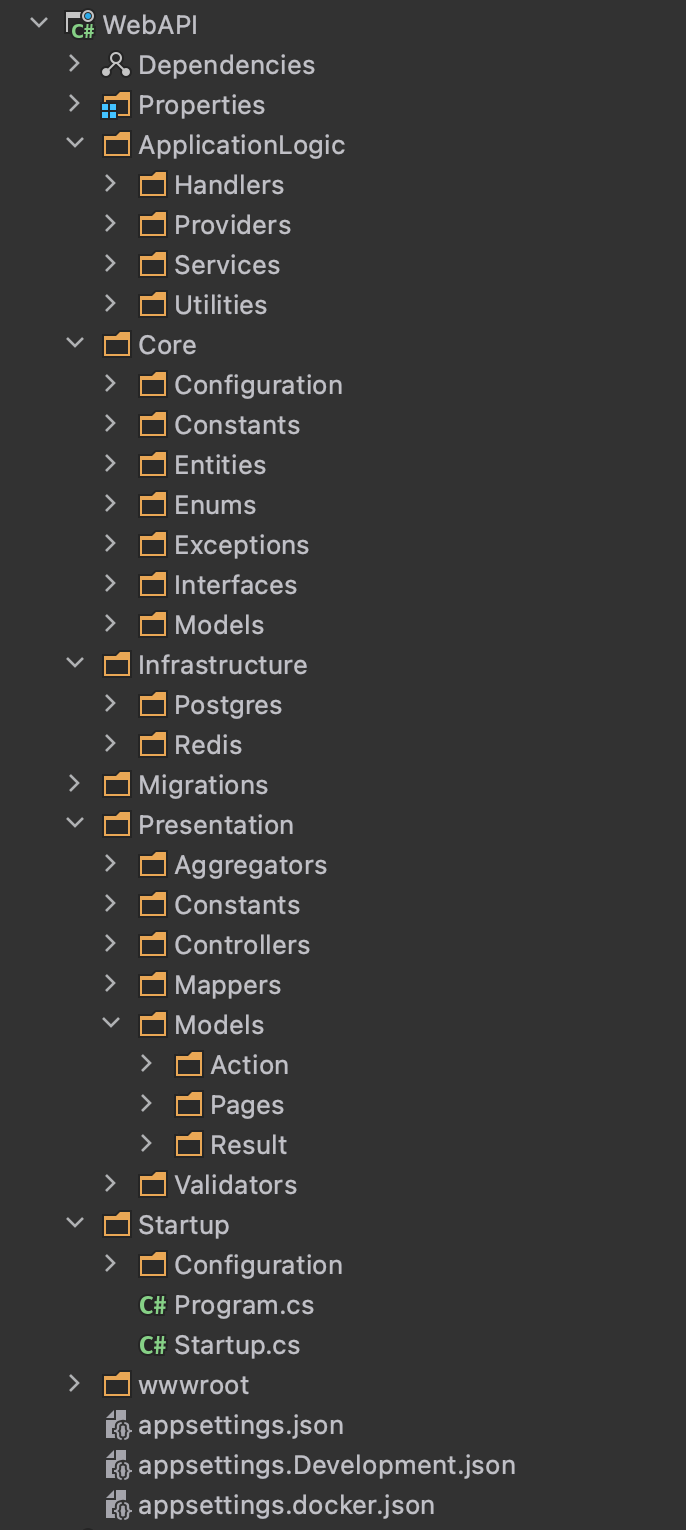


Рисунок 3.4 – Архитектура библиотеки *WebAPI*

В таблице 3.1 приведено подробное описание каждой папки и ее назначение для данного проекта.

Таблица 3.1 – Основная структура библиотеки *WebAPI*

|  |  |
| --- | --- |
| *Properties* | Директорий с конфигурацией проекта для запуска проекта в *IDE* |
| *ApplicationLogic* | Корневой директорий, который включает в себя все логические части проекта |
| *Handlers* | Директорий, который содержит логические компоненты по обработке данных, используется сервисами, не имеет доступ к инфраструктурному уровню |
| *Providers* | Назначение такое же, как и *Handlers*, однако здесь осуществляется доступ к инфраструктуре. |
| *Services* | Ключевой директорий, в котором производится обработка логики, вызывается уровнем представления, использует уровень инфраструктуры для взаимодействия с внешними источниками данных. |
| *Utilities* | Вспомогательные методы с переиспользуемой и несложной логикой, не взаимодействует с инфраструктурой, используется сервисами. |
| *Core* | Корневой уровень, который содержит в себе все ключевые компоненты для запуска и работы проекта |
| *Configuration* | Уровень, который содержит классы, использующиеся для хранения конфигурационных параметров при запуске приложения |
| *Constants* | Директорий для хранения констант на уровне инфраструктуры и логике приложения |
| *Entities* | Директорий с сущностями базы данных |
| *Enums* | Директорий с перечислениями, которые представляют собой ограничения целостности для сущностей |
| *Exceptions* | Директорий с исключениями, которые обрабатывает веб-сервис в случае возникновения ошибок |
| *Interfaces* | Папка с интерфейсами, которые нужны для реализации технологии *Dependency Injection* и общими требованиями к классам в проекте |
| *Models* | Директорий с моделями, использующимися на уровне по обработке логики и запуске приложения |
| *Infrastructure* | Корневой директорий, который хранит в себе логику по взаимодействию с внешними сервисами |
| *Postgres* | Папка с реализацией уровня доступа к базе данных *PostreSQL* |
| *Redis* | Папка с реализацией уровня доступа к базе данных *Redis* |
| *Migrations* | Директорий, хранящий в себе миграции, сделанные *Entity Framework* |
| *Presentation* | Корневой директорий, который хранит в себе все элементы уровня представления |
| *Aggregators* | Директорий, который используется для создания сложных моделей, приемлемых для *Front-End* части, из сущностей баз данных |
| *Constants* | Директорий, который хранит константы, использующиеся на уровне контроллеров |
| *Mappers* | Директорий для сопоставления сущностей и моделей для *UI* |
| *Models* | Корневой директорий для хранения моделей на уровне представления |
| *Actions* | Директорий, который хранит модели, использующиеся на уровне контроллеров в параметрах action’ов |
| *Pages* | Директорий, хранящий специфические модели для отдельных страниц |
| *Results* | Директорий с моделями аутентификации и авторизации пользователя |
| *Validators* | Директорий с валидаторами моделей, использующихся на уровне представления |
| *Startup* | Директорий с классами, представляющими собой конфигурацию и настройку всего проекта |
| *Configuration* | Директорий с конфигурацией отдельных элементов приложения, таких как подключение базы данных, настройка аутентификации и т.д. |
| *wwwroot* | Папка со статическими файлами приложения |

Таким образом, данная архитектура является наиболее приемлемой для данного веб-сервиса ввиду своей простоты и легкой возможности для расширения и внесения изменений в проект.

* + 1. Разработка контроллеров

Важнейшей частью любого веб-проекта является создание контроллеров для обработки данных, присланных пользователем, и отправкой результатов в приемлемом для того же пользователя формате. Контроллер управляет запросами пользователя.

Обычно контроллер вызывает соответствующий уровень обработки данных, получает подходящего вида. Они инкапсулируют всю бизнес-логику приложения, так как именно благодаря контроллеру данные из модели будут обрабатываться и передаваться клиенту. Всего было создано девять контроллеров, как показано на рисунке 3.5.

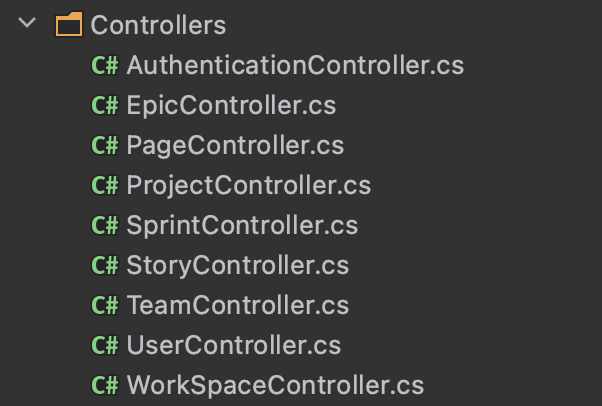


Рисунок 3.5 – Разработанные контроллеры

Пример программной реализации контроллера *UserController* показан в соответствующем листинге.

Каждый контроллер отвечает за работу с той сущностью, которая хранится в его названии, однако стоит сделать отдельное внимание на *PageController*. Данный контроллер необходим для обработки данных и отправки данных в такой форме, которые будут удобны *Front-End* приложению для отображения на странице, когда пользователь на нее переходит. Он вызывает в себе сервис, использующий доступ ко всем сущностям базы данных. Это позволяет избежать лишних запросов от клиентов для получения разнородных сущностей, уменьшив нагрузку на сервис.

Так же на уровне контролеров проверяются права пользователя для доступа и извлекаются из контекста запроса параметры, отвечающие за идентификацию пользователя в системе.

* + 1. Разработка сервисов

Как было описано в архитектуре ранее, основная задача сервисов, прежде всего, заниматься обработкой данных: вызывать инфраструктурный уровень для чтения и обработки данных во внешнем источнике, а также включать в себя логическую часть по преобразованию нужных данных и реализации соответствующих алгоритмов. Всего было сделано 9 сервисов, как продемонстрировано на рисунке 3.6, каждый из которых вызывается своих контроллером.

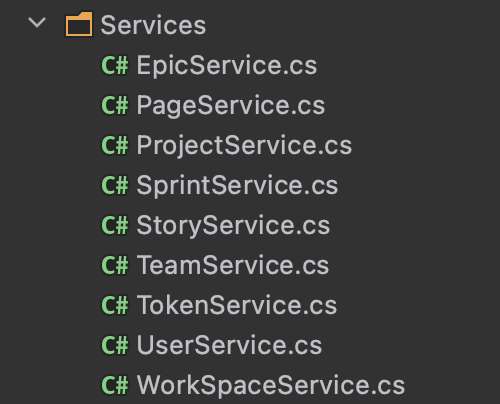


Рисунок 3.6 – Реализация сервисов в приложении

Пример программной реализации сервиса *UserService* показан на соответствующем листинге.

* + 1. Разработка репозиториев

Одной из начальных частей проекта является создание моделей и репозиториев для взаимодействия с уровнем базы данных. Чтобы иметь возможность работать с базой данных на языке *C#*, существует специальная *ORM* *Entity Framework*, которая позволяет взаимодействовать с данными, полученными из внешнего источника в формате объектов *C#*. Для реализации цели дипломного проекта был выбран подход «*Code First*» и созданы десять репозиториев и соответствующих им моделей, представленных на рисунке 3.7.

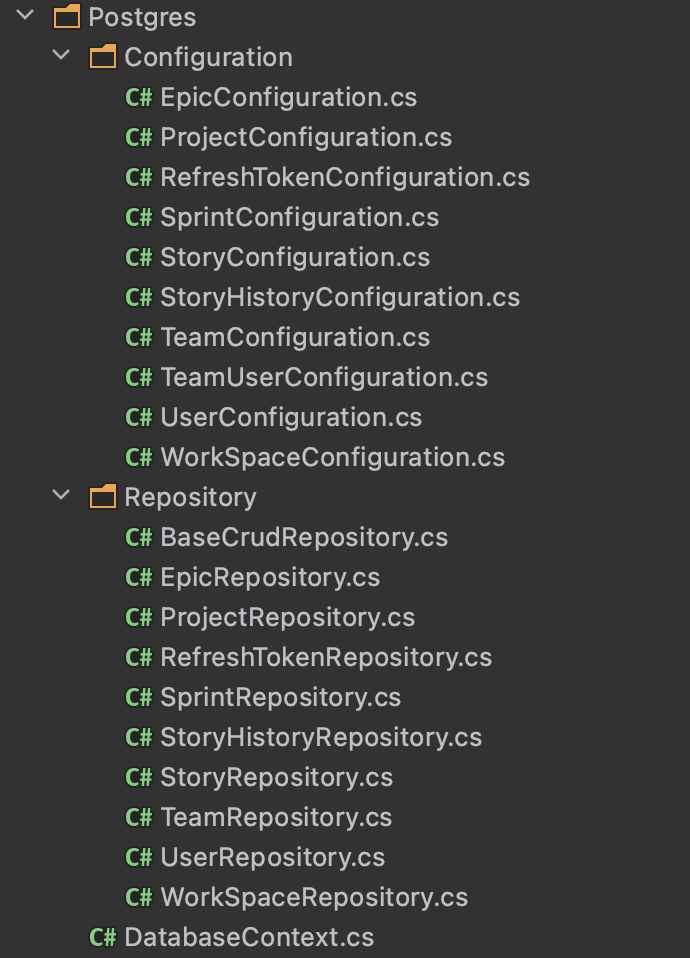


Рисунок 3.7 – Реализация репозитория для базы данных *PostgreSQL*

Для таких понятий, как модель и репозиторий, можно дать следующие определения:

Модель — прототип объекта, используемый вместо оригинала для решения задач. Модель строится на основании ограниченного множества известных свойств и поведения об оригинале.

Репозиторий — это абстрактный механизм хранения для коллекций сущностей. Именно он позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой.

Сама суть подхода «*Code First*» заключается в том, что все сущности, связи между ними, ограничения целостности, дополнительные объекты базы данных в виде индексов, представлений, создаются на прямую в коде. После того, как вся конфигурация была подготовлена, выполняется специальная команда по созданию миграций: «*dotnet ef add-migration name*».

Под миграцией подразумевается «снимок» конфигурации базы данных, сделанной при помощи *Entity Framework*, которая затем используется для получения скрипта в формате *SQL* и применяется к самой базе данных. В дипломном проекте, при помощи оркестратора контейнеров *docker-compose*, автоматически идет применение конфигурационного *SQL* скрипта к базе данных в контейнере.

Еще стоит добавить, что для конфигурации моделей использовался подход «*Fluent API*». Его суть заключается в том, что создается отдельный класс с конфигурацией модели для базы данных, в нем указываются все ограничения целостности, связи с другими сущностями.

Для реализации всех общих *CRUD*-операций был создан базовый репозиторий, который представляет из себя обобщенный класс, который принимает в качестве параметра модель сущности, а затем выполняет с этой моделью различные операции по выборке из базы, созданию, обновлению и удалению. Пример реализации базового репозитория показан на листинге.

* 1. Разработка Front-End части проекта
     1. Установка необходимых компонентов

Перед началом разработки проекта, с которым непосредственно будет взаимодействовать пользователь, нужно подготовить все необходимые инструменты для работы. Прежде всего, нужно установить платформу *NodeJS*.

Чтобы установить эту платформу, достаточно перейти на главную страницу официального сайта и скачать рекомендуемую версию приложения, которая является самой надежной и стабильной в работе. Чтобы убедиться, что версия *NodeJS* успешно работает, достаточно выполнить в терминале команду: «*node --version*». Результат выполнения показан ниже на рисунке 3.8.

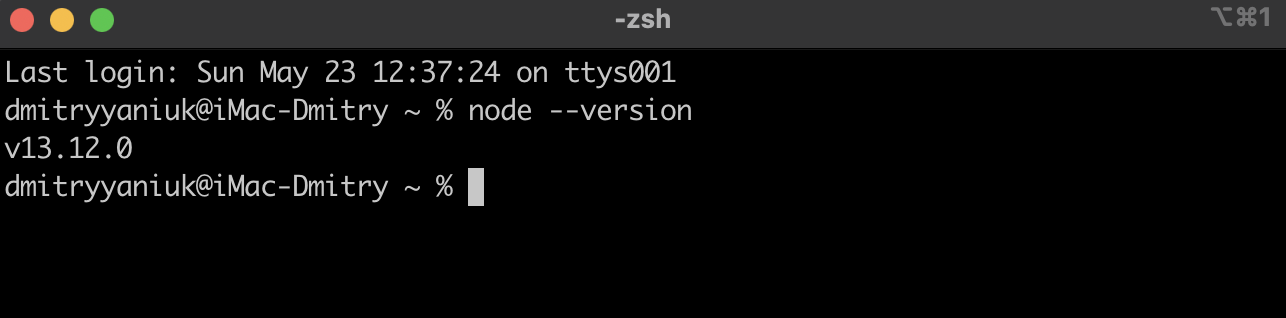


Рисунок 3.8 – Результат проверки работоспособности *NodeJS*

После этого можно приступить к созданию проекта. Чтобы это сделать, существует специальная утилита, разработанная компанией *Facebook*, позволяющая создавать приложение на *React* без дополнительных сложностей и конфигурации проекта. Эта утилита выглядит следующим образом: «*npx create-react-app –template typescript*» . Это означает, что будет автоматически создан проект с требуемой в дипломном проекте библиотекой, в котором сразу будет присутствовать поддержка языка *TypeScript*. Чтобы запустить проект, нужно убедиться, что все зависимости установлены. Если чего-то не хватает, то можно установить их при помощи команды: «*npm install*». После этого, чтобы запустить проект, достаточно написать команду «*npm start*». Для работы любого проекта на платформе *NodeJS* существует конфигурационный файл *package.json*, который хранит в себе всю информацию о проекте, а также скрипты для запуска. В нем уже находится скрипт с названием *start*, в котором прописана команда на запуск проекта. Далее можно приступить к самой реализации проекта, проверив, что он полностью в рабочем состоянии.

* + 1. Выбор архитектуры проекта

Перед началом разработки *Front-End* проекта необходимо также выбрать приемлемую архитектуру, которая будет удобна и приемлема для реализации основных функциональных возможностей приложения, а также в дальнейшем будет легко расширяема и просто в освоении.

Сами разработчики *React* предлагают на выбор несколько вариантов, которые будут применимы в зависимости от количества использования сторонних библиотек, а также общих целей и назначения проекта. Выбор был сделан в пользу архитектуры «*The Flat Structure*» [17].

Данный вид архитектуры является наиболее простым и понятным в использовании как для начинающих, так и опытных разработчиков.

Пример такой архитектуры продемонстрирован на рисунке 3.9.

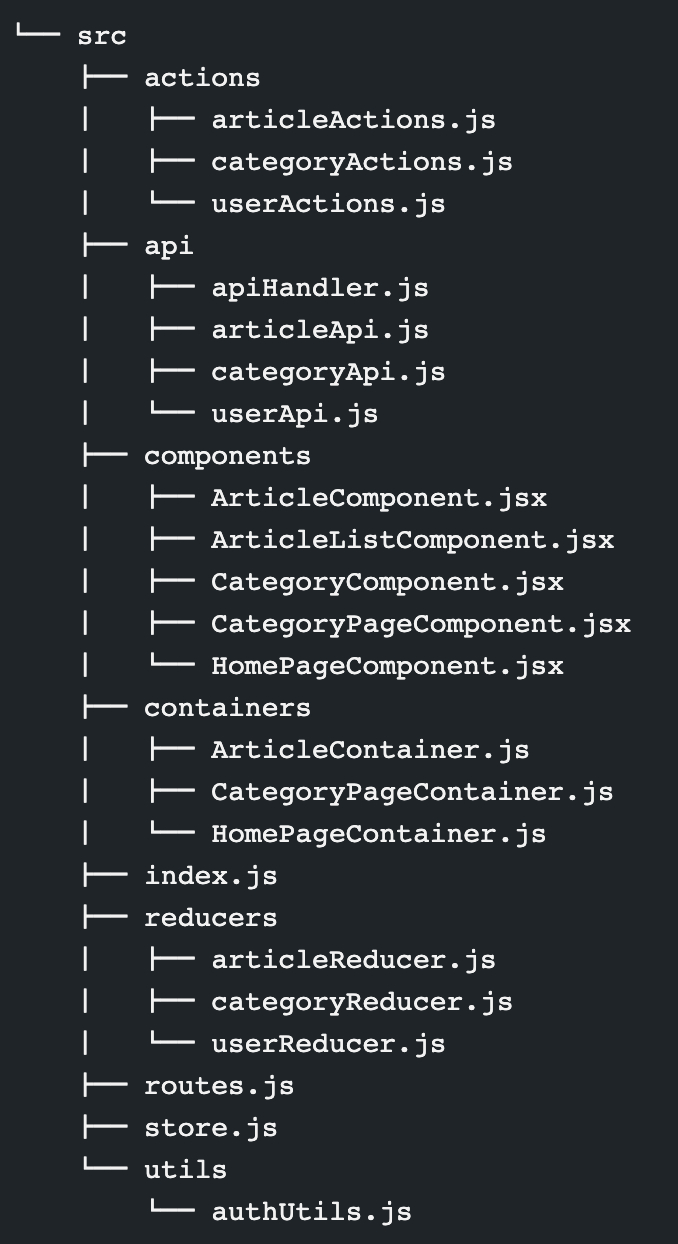


Рисунок 3.9 – Модель архитектуры «*The Flat Structure*»

Здесь можно выделить следующие особенности в структуре:

* *src* – корневой каталог, в котором находится весь исходный код к проекту;
* *actions* – директорий для хранения экшенов для библиотеки *Redux*;
* *api* – директорий для хранения функций, отвечающих за отправку *AJAX*-запросов;
* *components* – каталог для хранения компонентов, представляющих собой *HTML*-верстку;
* *containers* – каталог для хранения оберток над обычными компонентами, инкапсулирующими логическую часть;
* *index.js* – точка входа в приложение
* *reducers –* директорий для хранения редьюсеров для *Redux*;
* *routes.js* – файл для маршрутизации внутри приложения;
* *store.js* – файл для инициализации *Redux*-хранилища;
* *utils* – директорий со вспомогательной и переиспользуемой логикой в приложении.

Такая же архитектура будет использоваться и в дипломном проекте, однако, немного в измененном варианте, как показано на рисунке 3.10.

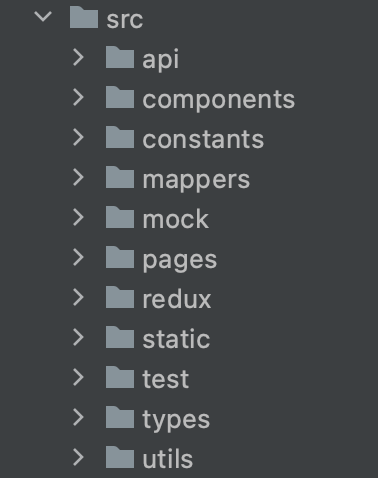


Рисунок 3.10 – Архитектура *Front-End* части проекта

Здесь все элементы, связанные с *Redux*, хранятся в соответствующем каталоге для создания более структурированного приложения, а также были добавлены следующие директории, как показано в таблице 3.2.

Таблица 3.2. – Добавленные директории во *Front-End* часть

|  |  |
| --- | --- |
| *constants* | Директорий с константами, которые используются во всем проекте |
| *mappers* | Каталог для функций, сопоставляющих данные, которые приходят с сервиса с моделями, использующимися на *UI* |
| *mock* | Директорий, который содержит ненастоящие объекты и модели, которые используются в тестировании |
| *pages* | Директорий для хранения компонентов, которые являются корневыми при рендере страницы |
| *static* | Каталог со статическими файлами |
| *test* | Каталог с *Unit*-тестами всей логической части приложения |
| *types* | Директорий с интерфейсами и типами, использующимися для описания моделей, функций в *Typescript* |

Таким образом, структура для проекта является полностью удобной и практичной в использовании.

* 1. Реализация компонентов

Самым наименьшим элементом, который представляет собой фрагмент с *HTML*-версткой, является компонент.

Сам по себе один компонент для отображения разбиты на два, в соответствии с общепринятой спецификацией, которая позволяет более гибко и удобнее разбивать логику и сам рендер *HTML*-элементов. Ниже, на рисунке 3.11, показан рисунок этой концепции.

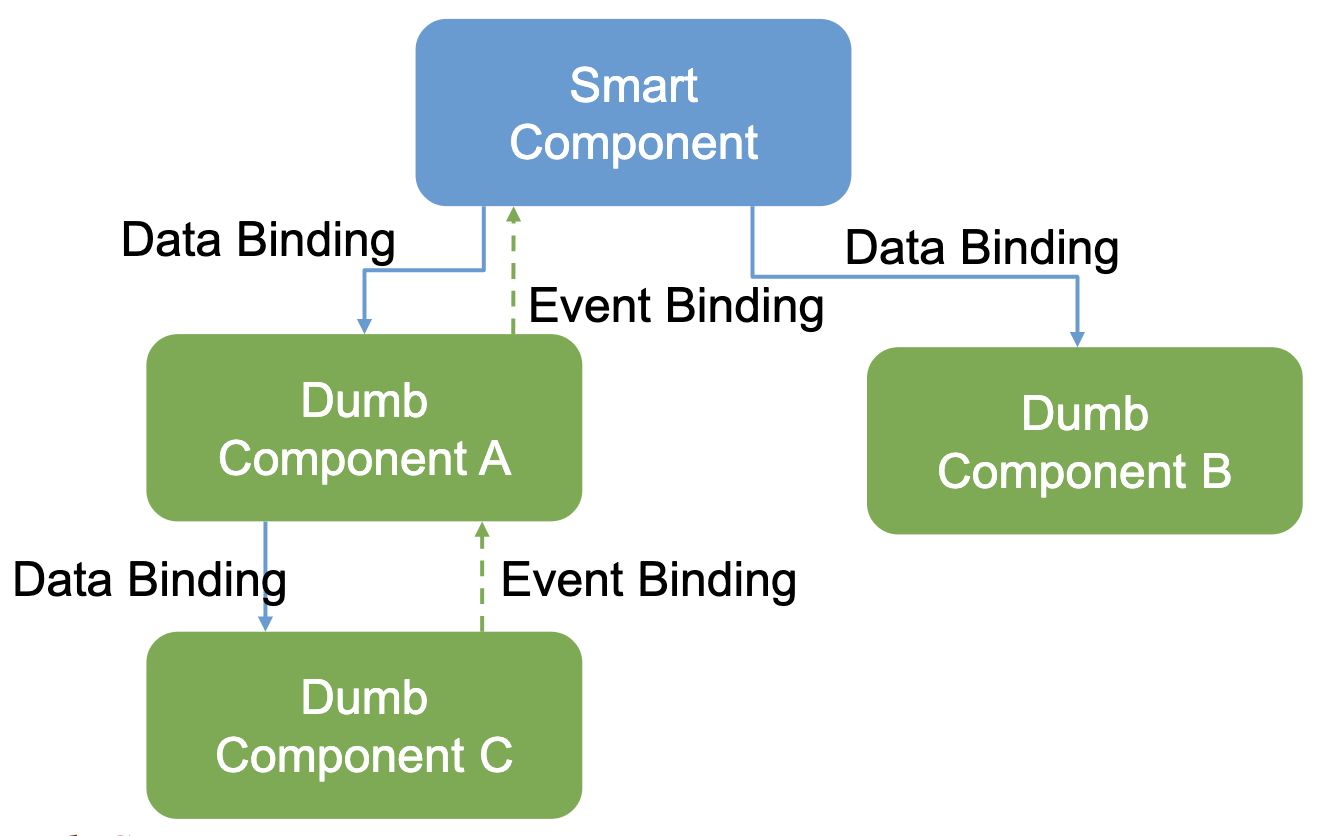


Рисунок 3.11 – Разбиение компонентов в *React*

Как можно увидеть, компоненты делятся на «умные» и «глупые». «Умные» компоненты отвечают, прежде всего, за работу логики и взаимодействие с Redux-хранилищем, в то время как «глупые» ничего не знают о логике и все, что они умеют делать, это принимать в себя данные, а при появлении события отправляют их на обработку в «умные» компоненты. Стоит также добавить, что такой вариант является очень практичным и удобным в использовании, так как компоненты становятся менее зависимыми и более переиспользуемыми. Примером таких компонент можно привести все компоненты, доступные в библиотеке *Material UI*. Пример программной реализации «умных» и «глупых» компонент показан на листинге.

* 1. Использование Redux и Redux Saga

Redux – библиотека для JavaScript с открытым исходным кодом, предназначенная для управления состоянием приложения. Чаще всего используется в связке с React или Angular для разработки клиентской части. Сама структура Redux представлена на рисунке 3.9 ниже:



Рисунок 3.9 – Концепция работы *Redux*

Кратко охарактеризуем каждый из компонент на рисунке:

* *State* – это единственный источник правды, который содержит всю необходимую информацию для нашего приложения;
* *Action* – кусочек информации, который должен изменить состояние. В нашем случае это перечисление, которое содержит новую информацию, которую мы хотим добавить или изменить в текущем *State*;
* *Reducer* - это функция, которая принимает *Action* и текущий *State* в качестве параметров и возвращает новый *State*. Это единственный способ создать его. Также стоит отметить, что эта функция должна быть чистой;
* *View* – обычный *UI*-компонент;
* *Store* – это объект, который содержит *State* и предоставляет все необходимые инструменты для его обновления.

Сама библиотека Redux базируется на трех фундаментальных принципах, несоблюдение которых может повлечь за собой некорректную работу приложения. Эти принципы выглядят следующим образом:

* «Состояние только для чтения» – гарантия, что гарантирует, что представления или функции, реагирующие на события, никогда не изменят состояние напрямую.
* «Единственный источник правды» – гарантия, что во всем приложении существует только единственный способ хранения данных и данные могут быть получены только оттуда.
* «Редьюсерами являются чистые функции» – функции, которые берут предыдущее состояние, экшен и возвращают новое состояние, в зависимости от параметров этого экшена. Никаких сторонних вызовов других компонент или отправка *HTTP*-запросов осуществляться не должна.

Помимо классического хранилища *Redux*, в проекте также реализованы дополнительные библиотеки, расширяющие функционал *Redux*. Одной из таких является *Redux Saga*.

*Redux Saga* – это промежуточный уровень обработки данных между *View* и самим *Redux* хранилищем. Это библиотека встраивается в общий конвейер обработки между *Redux и React*. Суть использования заключается в том, что теперь есть возможность совершения побочных эффектов, таких как *HTTP*-запросы, дополнительная логика и тому подобное, поскольку функции в самом *Redux* должны быть чистыми, то есть только обновлять состояние и затем возвращать его. Для возможности использования данной библиотеки нужно хорошо понимать, как работают итераторы, генераторы в классическом *JavaScript*, так как вся *Saga* базируется на этом. Есть генераторы, которые при совершении какого-то действия приостанавливают свое выполнение и после получения результата сразу же продолжают свое выполнение. Общая концепция работы библиотеки *Redux Saga* представлена ниже на рисунке 3.11.

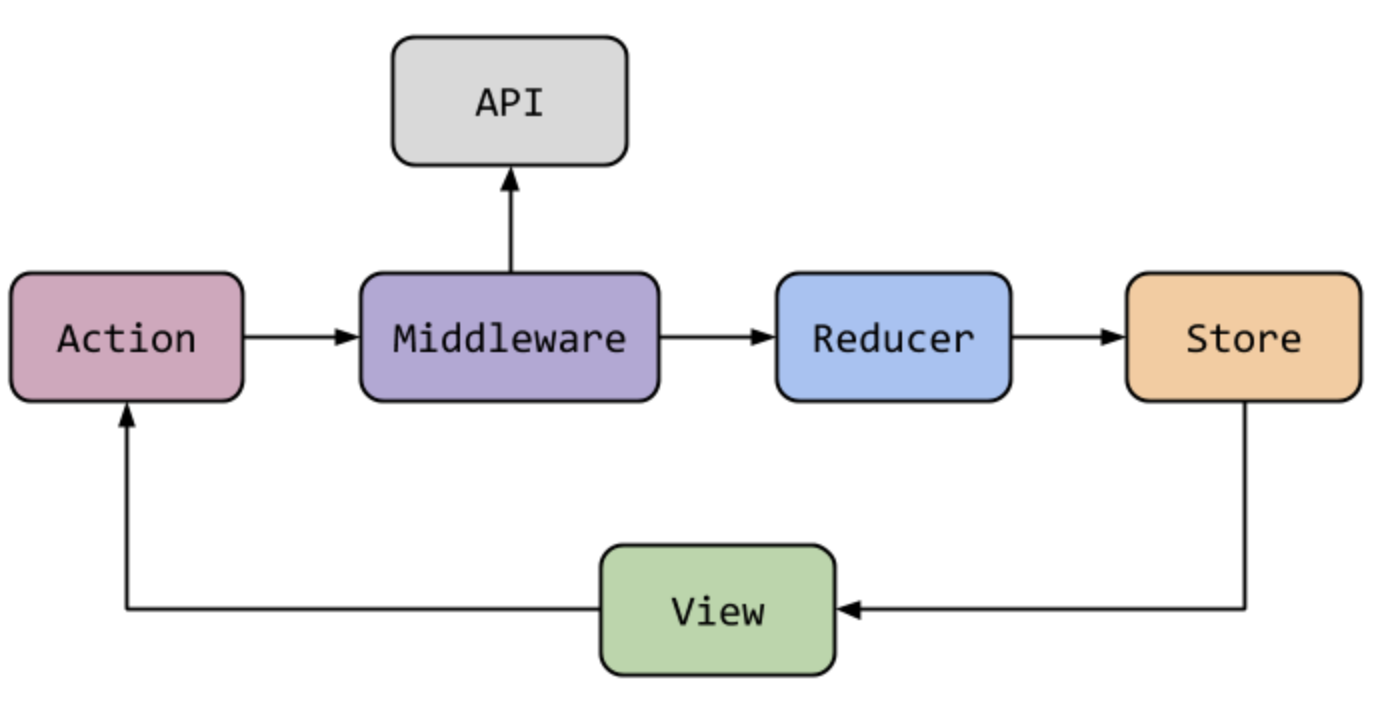


Рисунок 3.11 – Концепция работы *Redux Saga*

На данной схеме показано, что в классическом порядке работы *React* c *Redux* появляется промежуточный уровень между *Action* и *Reducer*, который способен взаимодействовать с внешней *API* и таким образом, при получении данных отправляет их в *Reducer*, тот в свою очередь обновляет *Store*, затем новые данные отправляются во *View* и так далее.

Помимо использования *Redux Saga* используется также библиотека Redux Logger, который позволяет избежать дополнительных установок различных плагинов в браузере для отслеживания изменения состояния и выводит всю информацию об отработавших *Action* в консоль браузера (данную возможность легко использовать в режиме разработке и затем отключить на релизе). И еще одним дополнением к *Redux* является библиотека *Redux Persistent*, которая позволяет после каждой перезагрузки страницы, закрытия вкладки или самого браузера возобновить последнее актуальное состояние приложения. Достигается это за счет интеграции с хранилищем браузера в зависимости от установленных параметров при инициализации *Redux* хранилища: как только мы вызовем несколько *Action*, к примеру, у нас установлена подписка на обновление данных также в браузере при любом изменении их в самом *Redux*. А после обновления страницы мы получим эти самые данные из хранилища данных браузера. Это позволяет избежать лишней нагрузки на сервер, но нужно быть уверенным, что мы не потеряем актуальности данных.

* 1. Вывод по разделу

В ходе работы над дипломным проектом была выполнена установка необходимых инструментов, которые нужны для разработки *Front-End* и *Back-End* частей проекта.

Были выбраны архитектуры проектов, которые будут соответствовать поставленным задачам самого дипломного проекта, а также будут удобны и просты для понимания, реализации и дальнейшей поддержки, расширения функционала проектов.

Были настроены и сконфигурированы обе части дипломного проекта на взаимодействие друг с другом, реализованы основные функциональные возможности и ранее подготовленные алгоритмы приложения.

1. Тестирование программного средства

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74418025, 2021*

Процесс тестирования является не менее важным, чем процесс разработки приложения, поскольку на данном этапе можно найти недочеты в работе приложения и исправить их. В данной главе будут описаны варианты проведения тестирования на *Back-End* и *Front-End* частях проекта.

Одним из самых популярных и надежных способ тестирования приложения является модульное тестирование.

Модульное тестирование, или *Unit*-тестирование — это процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы. Идея заключается в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

* 1. Тестирование Back-End части

Чтобы убедиться в работоспособности веб-сервиса, нужно проверить его отдельные элементы: сервисы, логическую часть приложения, вспомогательные утилиты и так далее. Это поможет убедиться, что каждый компонент приложения независим и его можно переиспользовать. Здесь пригодиться вариант модульного тестирования.

Для проведения *Unit*-тестирования в среде *.Net* существует уже большое количество различных библиотек, которые можно импортировать из *Nuget*-пакета, либо использовать встроенный модули, добавленный несколько лет назад. В моем случае выбор сделан был в пользу набора тестовых модулей *XUnit*. Она является полностью бесплатной, *Open Source* библиотекой, поддерживается не только языком *C#*, но также и *Visual Basic*, *F#* и другими языками, использующими платформу *.Net*.

Для абстракции тестирования от самой логики приложения была создана дополнительная *DLL*-библиотека, которая будет содержать зависимости от двух предыдущих библиотек, поскольку для корректного проведения тестов понадобится импортировать функции и модели.

Еще одной особенностью для проведения модульного тестирования является создание ненастоящих, или же «фейковых» объектов. Дело в том, что в проекте активно используются вызовы инфраструктурных элементов: базы данных, контекста контроллера. Модульное тестирование предполагает абстракцию от инфраструктуры приложения и для корректного тестирования нужно создавать ненастоящие объекты классов, которые будут подменять собой инфраструктурные элементы

и полностью соответствовать их поведению. Для создания «фейковых» объектов можно использовать несколько популярных библиотек. Выбор сделан в пользу библиотеки *FakeItEasy*.

Данная библиотека является очень простой и удобной в использовании, не требует от разработчика дополнительных знаний в области тестирования (к примеру, различия между объектами «*mock*» и «*stub*»), а также имеет небольшой размер, простую и понятную документацию.

Сами *Unit*-тесты создавались в соответствии с общепринятым паттерном «ААА», что означает следующие действия:

* *Arrange* – объявление исходных и ожидаемых данных, инициализация «фейковых» объектов;
* *Act* – осуществление тестирования, вызов необходимых методов;
* *Assert* – проверка совпадения ожидаемых и полученных результатов.

Всего в проекте было создано 168 тестов, проводилось тестирование всего функционала, которые участвовал в процессе обработки данных: мапперы, аггрегаторы, сервисы, контроллеры, утилиты, валидаторы моделей.

Если описывать сами тесты, то на каждый метод была создана пара из «позитивного» и «негативного» теста, иногда было больше «позитивных», иногда «негативных» тестов для проверки работы модуля во всех ситуациях. Их отличие заключается в том, что в первом случае на вход тесту предоставляются некоторые данные и идет расчет, что эти данные будут корректно отработаны и получится желаемый результат.

Пример «позитивного» теста, в котором мы ожидаем получить хешированную строку с паролем, можно увидеть ниже на листинге 4.1.

|  |
| --- |
| [Fact]  public void ShouldCreateHashForPasswordWithSha512()  {  //Arrange  const string password = "123";  const string expectedHash = "3C9909AFEC25354D551DAE21590BB26E38D53F2173B8D3DC3EEE4C047E7AB1C1EB8B85103E3BE7BA613B31BB5C9C36214DC9F14A42FD7A2FDB84856BCA5C44C2";    //Act  var result = PasswordHashing.CreateHashPassword(password);  //Assert  Assert.Equal(expectedHash, result);  } |

Листинг 4.1 – «Позитивный» *Unit*-тест

Во втором же случае, специально подаются некорректные данные, которые должны привести к ошибочному результату или вообще выбросить исключение. Делается это, прежде всего, для проверки корректной работы самого метода, затем обработчика ошибок, если он там присутствует, то убедиться, не вызовет ли это сбой в работе всего приложения, а также проверки отдельных условий в методе.

Пример созданного «негативного теста», в котором проверяется создание исключения в методе для хеширования пароля, представлен на листинге 4.2.

|  |
| --- |
| [Fact]  public void ShouldThrowErrorForInvalidInitialPasswordOnPasswordHashing()  {  //Arrange  var password = string.Empty;    //Act && Assert  Assert.Throws<UserFriendlyException>(() => PasswordHashing.CreateHashPassword(password));  } |

Листинг 4.2 – «Негативный» *Unit*-тест

Общий результат прохождения *Unit*-тестов представлен ниже на рисунке 4.1.

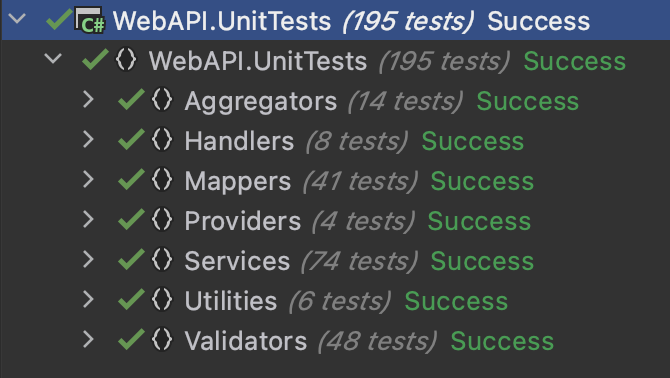


Рисунок 4.1 – Прохождение тестов на веб-сервисе

Как можно увидеть, все тесты успешно прошли работу, что означает корректную работу модулей всего веб-сервиса. Все исключительные ситуации, которые могли привести к нестабильной работе приложения, были найдены и исправлены.

* 1. Тестирование Front-End части

Для тестирования *Front-End* части проекта был использован тот же подход с модульным тестированием. Для написания тестов для платформы *NodeJS* представлено большое количество библиотек и фреймворков: *Jest*, *Mocha*, *Jasmine*. Выбор сделан в пользу *Jest*, поскольку он сразу включен в проект при его создании, а также, конфигурация и написание тестов довольно просты. К тому же, он поддерживает большое количество сторонних библиотек и плагинов, являясь продуктом типа *Open Source*.

В ходе тестирования была проверена корректная работа логических компонентов приложения, а именно *Redux*, который отвечает за управление глобальным состоянием приложения и его обновлением, *Redux-Saga*, которая отвечает за взаимодействие со сторонними эффектами и веб-сервисом, а также дополнительные утилиты и помощники, позволяющие переиспользовать логическую часть при обработке данных во всем приложении.

Стоит также отметить, что инструмент *Jest* предоставляет базовые возможности для создания «позитивных» и «негативных» тестов для саг, однако такой код получается чересчур громоздкий и неудобно читаемый, а также не предоставляется возможность сделать проверку вызова цепочек из саг. Чтобы упростить написание таких тестов, была использована библиотека *redux-saga-test-plan*, устраняющая все вышеперечисленные недостатки при создании тестов для *Redux-Saga*. При написании теста использовался ранее хорошо зарекомендовавший себя паттерн «ААА».

Пример листинга «позитивного» теста саги, в котором идет процесс создания рабочего пространства пользователя, с использованной библиотекой *redux-saga-test-plan* приведен на листинге 4.3.

|  |
| --- |
| it(`Should create user workspace on ${WorkSpaceActions.CREATE\_WORKSPACE\_REQUEST}`, () => {  //Arrange  const workSpaceModel: IWorkSpace = { workSpaceDescription: 'description', workSpaceName: 'name' };  const createdWorkSpaceModel: IWorkSpace = {  workSpaceDescription: 'description',  workSpaceName: 'name',  workSpaceId: 'id',  creationDate: new Date(),  };  const action: ICreateWorkSpaceRequest = createWorkSpaceRequest(workSpaceModel);  //Act & Assert  return expectSaga(createWorkSpace, action)  .provide([[call(WorkSpaceApi.createWorkSpace, workSpaceModel), createdWorkSpaceModel]])  .put(createWorkSpaceSuccess(createdWorkSpaceModel))  .put(push(WorkspaceViewerRoute))  .run();  }); |

Листинг 4.3 – «Позитивный» *Unit*-тест саги

Также написан тест, в котором проведена проверка исключительной ситуации, при которой с сервера вернется ошибка, и сага должна отловить это исключение и продолжить работу, не вызвав при этом сбой в работе приложения. Пример такого теста показан на листинге 4.4.

|  |
| --- |
| it(`Should throw error on creating user workspace on ${WorkSpaceActions.CREATE\_WORKSPACE\_REQUEST}`, () => {  //Arrange  const workSpaceModel: IWorkSpace = { workSpaceDescription: 'description', workSpaceName: 'name' };  const error = new Error('error');  const action: ICreateWorkSpaceRequest = createWorkSpaceRequest(workSpaceModel);  //Act && Assert  return expectSaga(createWorkSpace, action)  .provide([[call(WorkSpaceApi.createWorkSpace, workSpaceModel), throwError(error)]])  .put(createWorkSpaceFailure(error))  .run();  }); |

Листинг 4.4 – «Негативный» *Unit*-тест саги

По итогу написания тестов для всей *Front-End* части проекта был получен результат, демонстрирующий успешное выполнение всех тестов, а самое главное, стабильную работу приложения, которые не приведет к исключительным ситуациям и позволит пользователю работать с ним комфортно. Результат тестирования показан на рисунке 4.2.

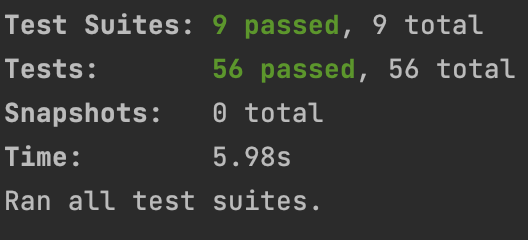


Рисунок 4.2 – Прохождение всех на *Front-End* части

Во время проведения тестирования логической части Front-End приложения, были обнаружены дефекты, которые могли привести к сбою во всем приложении.

* 1. Вывод по разделу

Основной целью данного раздела являлось тестирование всего дипломного для гарантии его целостной и стабильной работы.

Для тестирования *Back-End* части проекта были задействованы такие инструмента, как семейство фреймворков тестирования *XUnit*, а таже библиотека *FakeItEasy*.

Для тестирования *Front-End* части проекта была использована библиотека Jest, которая шла автоматически при создании проекта, а также *redux-saga-test-plan* для упрощенного написания тестов для *Redux-Saga*.

В ходе проведения тестирования всего проекта были обнаружены и устранены все уязвимости, которые могли повлечь за собой нарушение стабильности работы и потерю данных.

1. Руководство пользователя

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

5. Руководство пользователя

Лит.

Листов

1

*74418025, 2021*

* 1. Развертывание приложения в среде Docker

Для проверки работоспособности приложения прежде всего необходимо установить на компьютере *Docker*. Никаких дополнительных экземпляров базы данных, различных платформ вроде *NodeJS*, *.Net Core* ему ставить не понадобится, все образы будут скачано через зависимости в конфигурационных файлах *Docker*. Таким образом, проект не требует жесткой привязки к какой-либо системе, может разворачиваться спокойно на любой операционной системе. Для создания самих контейнеров *Docker* нужно всего лишь написать команду в папках двух проектов: «*docker compose up –build*». После этой команды автоматически соберутся все контейнеры и можно начать работать с приложением, просто перейдя по следующему *URL*: «*http://localhost:3006*».

Пример опубликованного приложения в Docker выглядит следующим образом, как показано на рисунке ниже 5.1.

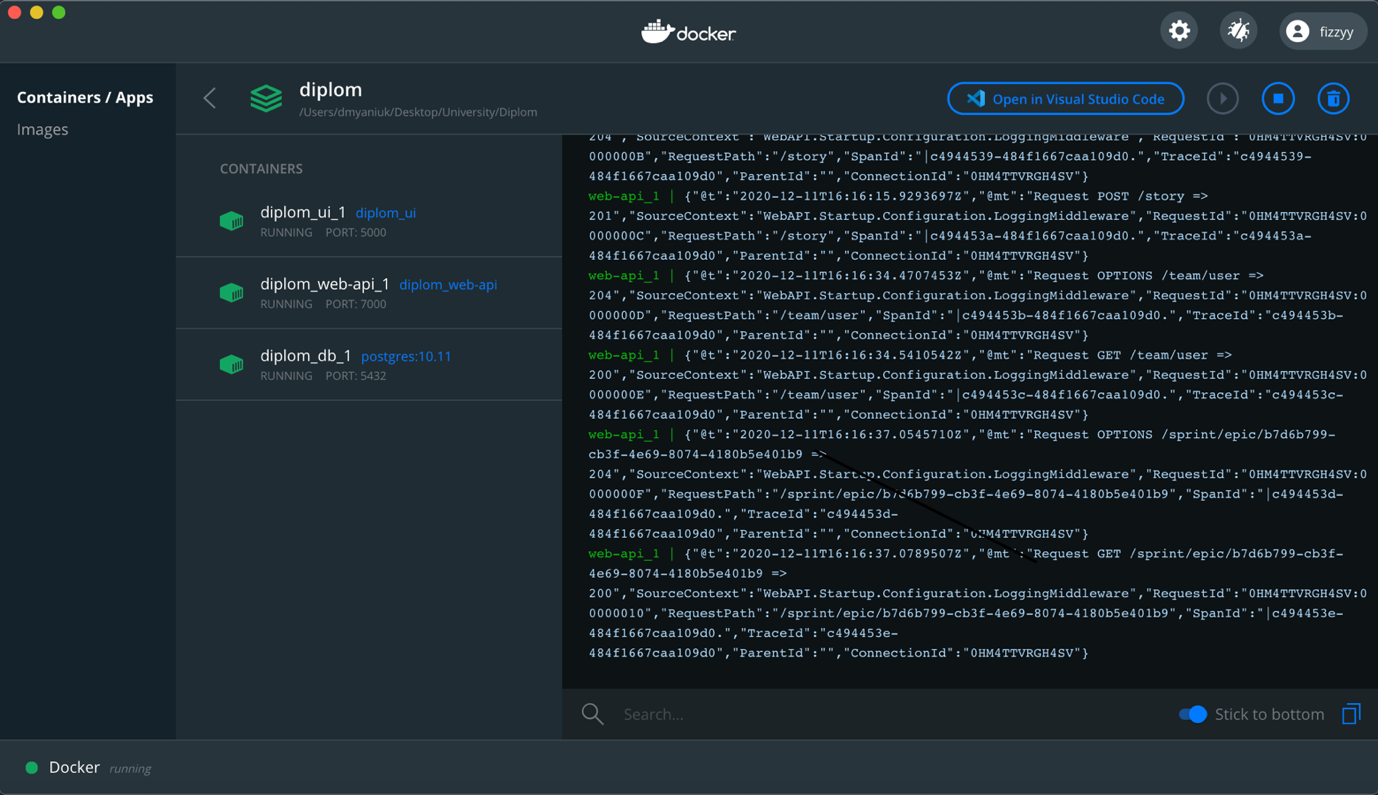


Рисунок 5.1 – Развернутое приложение в среде Docker

Если отсутвует возможность проверки на графическом клиенте, убедиться в том, что контейнеры запущены и находятся в рабочем состоянии можно выполнив команду: docker ps -a. Таким образом, мы получим все созданные контейнеры внутри Docker. В нашем случае это выглядит так, как показано на рисунке 5.2.

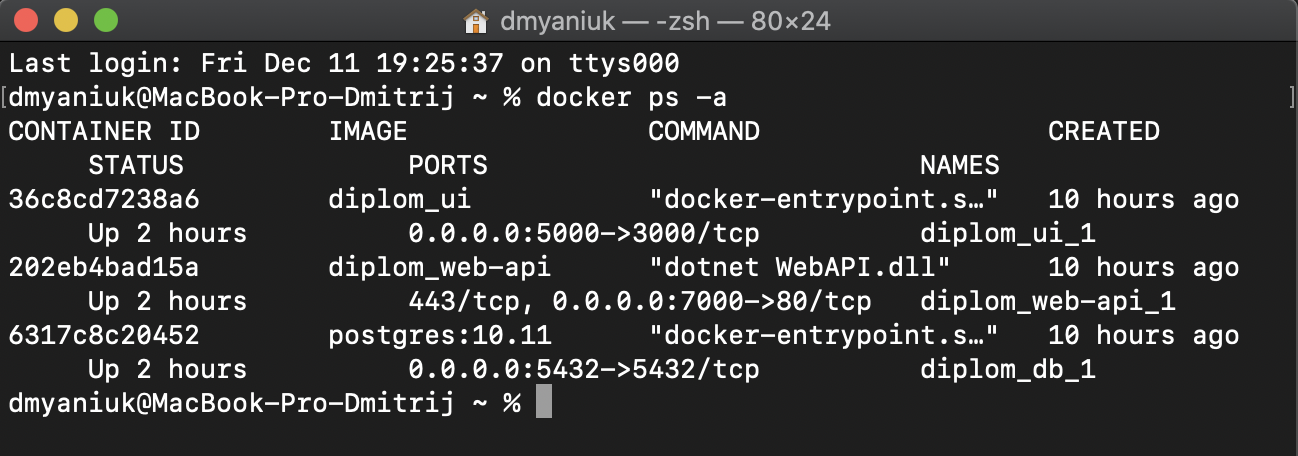


Рисунок 5.2 – Просмотр работающих контейнеров

Стоит сделать уточнение, что данный проект разворачивался на ОС *MacOS*. Эта версия операционной системы является членом семейства *Unix*-типов.

Чтобы убедиться, что проект также успешно запускается на ОС семейства *Windows*, нужно выполнить тот же самый, порядок действий, описанный ранее. Однако, в этом случае версия Windows должна быть не менее 10, поскольку Docker изначально поддерживался только на *Unix*-системах и воспользоваться им на *Windows* не было возможности. На новой версии *Windows* была добавлена поддержка контейнеризации, однако, работает она не совсем аналогичным образом, как на *Linux* или *MacOS*. Результат запуска контейнеров приведен ниже.

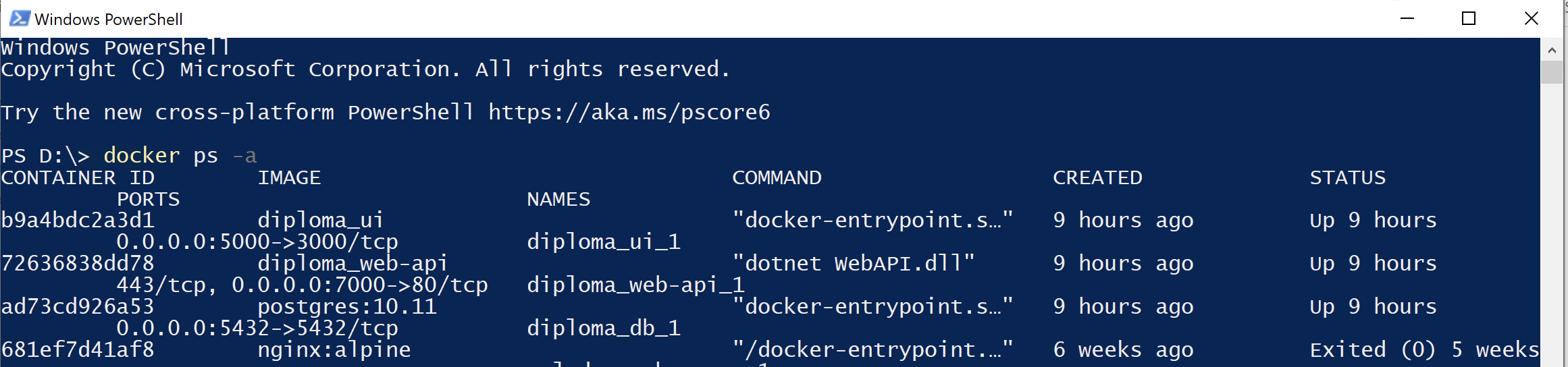


Рисунок 5.3 – Просмотр работающих контейнеров на ОС *Windows*

Для гарантии полной работоспособности развертывания приложения была также сделана попытка установки проекта на ОС *Linux*. Взаимодействие с данной операционной системой проводилось удаленно при помощи *SSH*-сервиса с физической операционной системы *MacOS.*

На платформе Ubuntu с версией 18.04.05 запуск приложения в *Docker* также не вызывает никаких сложностей. Для корректного запуска нужно установить пакеты *Docker* при помощи команды: «*apt-get install docker docker.io docker-compose*». Таким образом, будет установлен сам механизм средства контейнеризации *Docker*, а также возможность оркестрации контейнеров. Затем уже после установки Docker нужно перейти в сам корневой каталог с приложением и выполнить команду: «*docker-compose up –build*». Чтобы проверить, что все контейнеры были запущены и приложение функционирует исправно, выполняется та же команда с поднятием контейнеров. Результат показан на рисунке ниже 5.4.

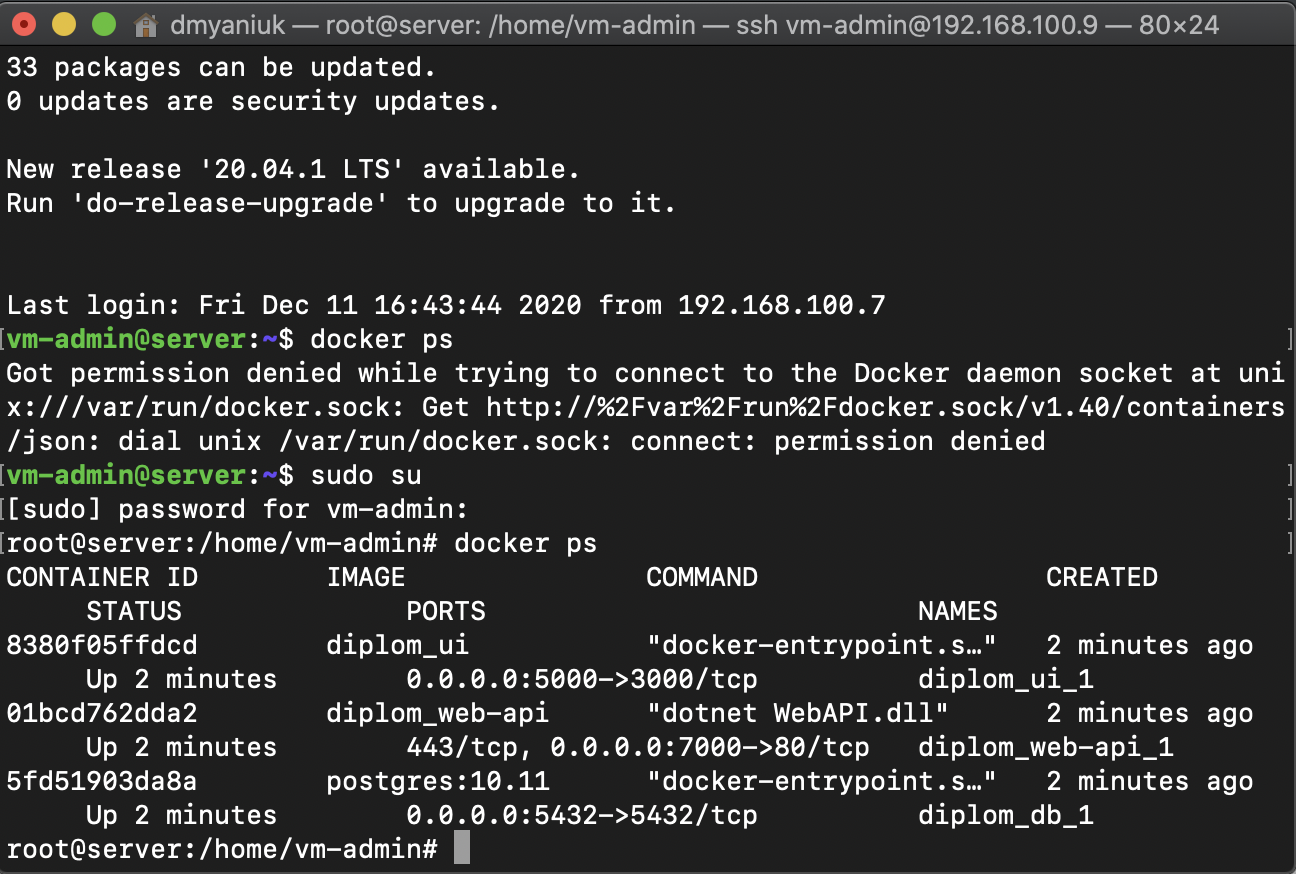


Рисунок 5.4 – Просмотр работающих контейнеров на ОС *Linux*

Таким образом, любая операционная система, имеющая поддержку контейнеризации или средство *Docker*, может без всяких проблем запускать данное приложение.

* 1. Руководство для менеджера

Перед стартом работы с приложением, прежде всего, необходимо завести учетную запись. Сделать это можно перейдя на основной маршрут домена, на котором находится работающее приложение. Приложение автоматически определяет не аутентифицированного пользователя и запрещает ему дальнейших вход в систему. Окно регистрации выглядит следующим образом, как показано на рисунке 5.5.

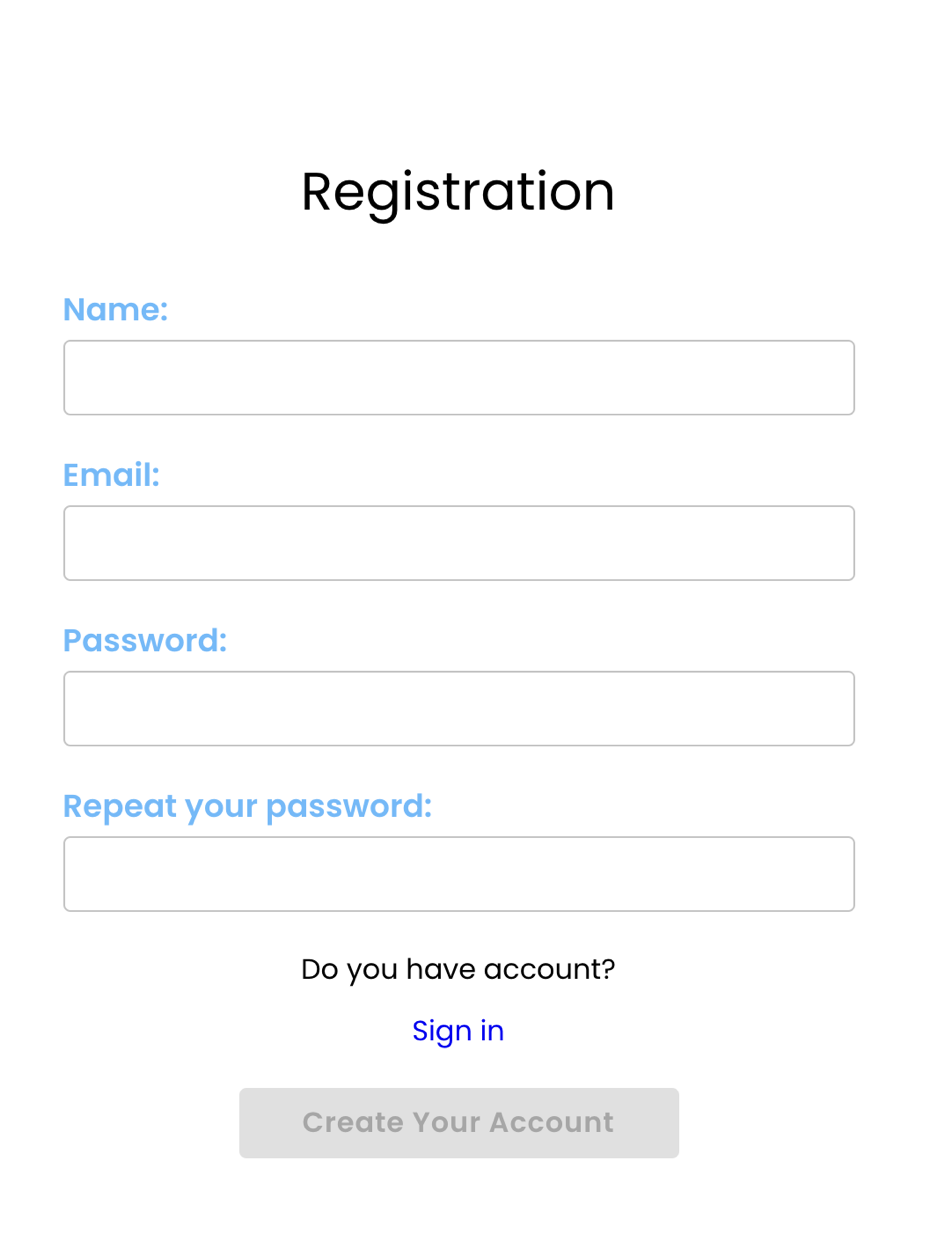


Рисунок 5.5 – Страница регистрации пользователя

После того, как пользователь создаст новую учетную запись, ему необходимо будет аутентифицироваться. Чтобы это сделать, нужно нажать на подсказку «*Sign in*», произойдет переадресация на страницу со входом в учетную запись. Это показано на рисунке 5.6.

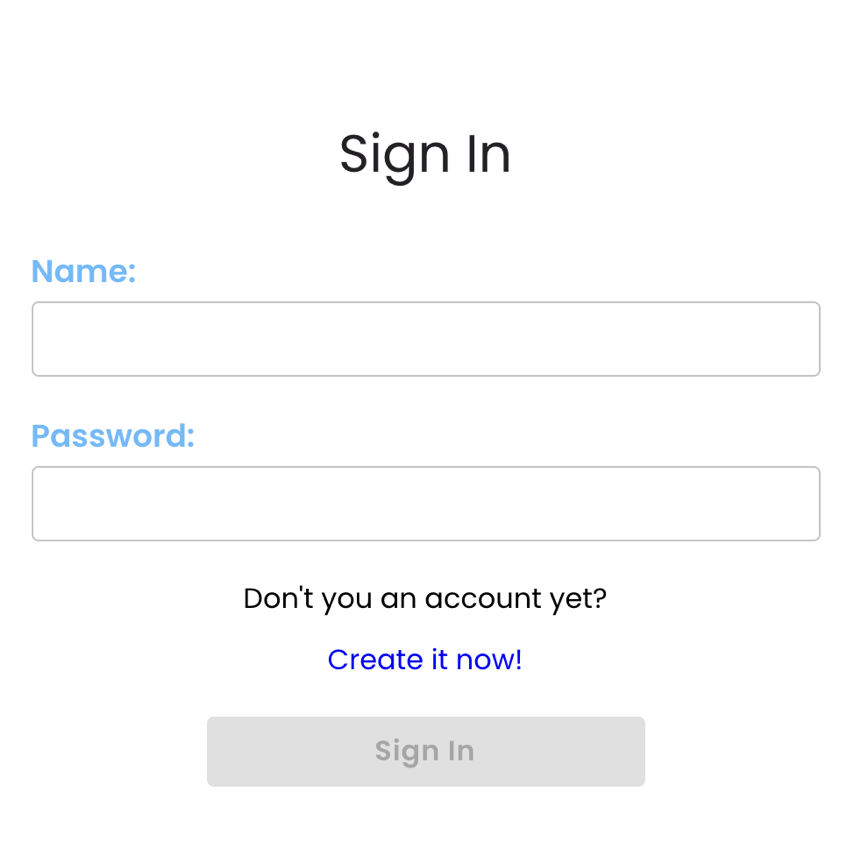


Рисунок 5.6 – Страница со входом в учетную запись

В случае, если пользователь был успешно аутентифицирован, он будет переадресован на главную страницу приложения. В противном случае, появится соответствующее сообщение о некорректных данных об учетной записи. Главная страница представлена на рисунке 5.7.

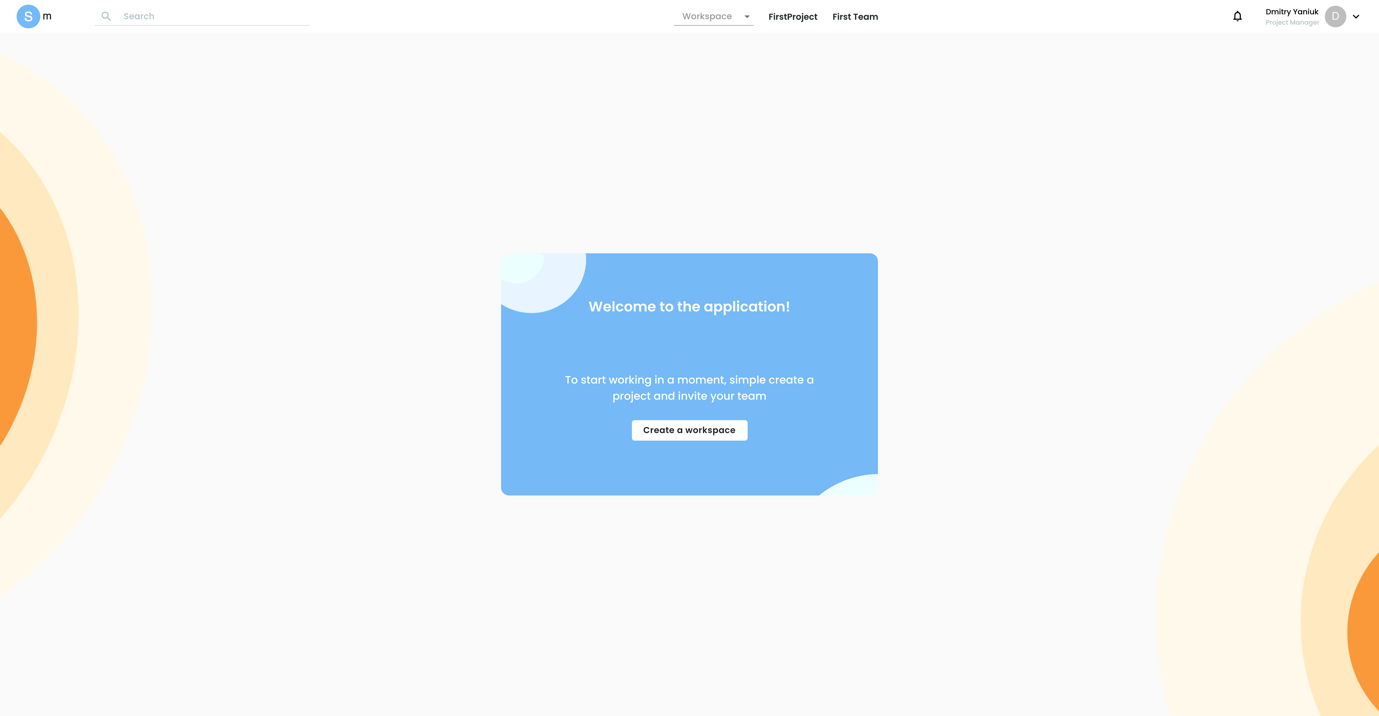


Рисунок 5.7 – Стартовая страница после регистрации

* 1. Выводы по разделу

Данный раздел предоставляет пояснения по работе пользователя с веб-приложением. Так как интерфейс приложения прост и интуитивно понятен, то у пользователя не должно возникнуть трудностей с его эксплуатацией.

Помимо обычного пояснения по работе с приложением данный раздел включает в себя дополнительное пояснение по развертыванию приложения в среде *Docker*, а также инструкцию по запуске на таких популярных операционных системах, как *Windows*, *Linux*, *MacOS*.

Так как приложение имеет разделение ролей на менеджеров и инженеров, то руководство в данной главе было описано для двух ситуаций: когда приложением будет пользоваться менеджер, а также когда инженер.

Заключение

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74418025, 2021*

В ходе выполнения работы были проанализированы аналоги веб-приложения по управлению IT-проектами, выявлены их положительные и отрицательные стороны. В свою очередь, результаты анализа были учтены при проектировании дизайна и функционала приложения.

Была разработана реляционная база данных для веб-приложения, использующая экземпляр PostgreSQL, имеющая в своем наборе 10 таблиц.

Была создана аутентификация и авторизация пользователей по их ролям.

В ходе выполнения данной работы подробно были закреплены знания для проектирования баз данных, с учётом большого количества хранимых параметров.

Для использования всего функционала созданной базы данных, был создан веб-сервис, написанный на ASP .Net Core со всеми необходимыми API, позволяющими обслуживать пользовательские запросы.

Было также разработано Front-end приложение на современном фреймворке React, позволяющее создавать гибкую и удобную верстку, а также легко взаимодействовать с Ajax-запросами и создавать необходимые эффекты для анимации.

Благодаря проведенной работе, был создан проект, который может быть очень удобен и полезен для создания системы управления проектами.

Основным плюсом данного проекта, является простота и возможность бесплатного распространения, так как технологии, использованные в данном проекте, полностью интегрированы в среду виртуализации Docker. Благодаря этому приложение является кроссплатформенным: его можно на использовать на различных популярных операционных системах, таких как MacOS, Windows и Linux.

Список используемых источников

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Янюк Д.С.*

Пров.

Кантарович В.С.

Консульт.

Н. контр.

Копыток Д.В.

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74418025, 2021*

1. Trello [Электронный ресурс] /Trello Board – Режим доступа: https://trello.com/ ­– Дата доступа: 18.04.2021.
2. Rally Board [Электронный ресурс] / Rally Broadcom – Режим доступа: https://www.broadcom.com/ ­– Дата доступа: 18.04.2021.
3. YouTrack [Электронный ресурс] / JetBrains YouTrack – Режим доступа: https://www.jetbrains.com/youtrack/ ­– Дата доступа: 18.04.2021.
4. Патент US20150347125A1 [Электронный ресурс]. – Режим доступа: https://patents.google.com/patent/US20150347125A1. – Дата доступа: 21.04.2021.
5. Патент US10332073B2 [Электронный ресурс]. – Режим доступа: https://patents.google.com/patent/US10332073B2. – Дата доступа: 21.04.2021.
6. Download .Net [Электронный ресурс] – Режим доступа: https://dotnet.microsoft.com/download ­– Дата доступа: 20.04.2021.
7. ASP .Net Core [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/aspnet5/ ­– Дата доступа: 20.04.2021.
8. PostgreSQL [Электронный ресурс] / PostgreSQL – Режим доступа: https://www.postgresql.org/ ­– Дата доступа: 20.04.2021.
9. React [Электронный ресурс] / React JS – Режим доступа: https://ru.reactjs.org/ – Дата доступа 20.04.2021.
10. Redux [Электронный ресурс] / Redux JS – Режим доступа: https://github.com/reduxjs/redux – Дата доступа: 20.04.2021
11. Typescript [Электронный ресурс] / Typescript – Режим доступа: https://www.typescriptlang.org – Дата доступа: 20.04.2021
12. Redux Saga [Электронный ресурс] Redux Saga JS – Режим доступа: https://redux-saga.js.org/ ­– Дата доступа: 20.04.2021.
13. Docker [Электронный ресурс] Docker – Режим доступа: https://www.docker.com ­– Дата доступа: 20.04.2021
14. Clean Architecture pattern [Электронный ресурс] / Clean Architecture – Режим доступа: https://github.com/jasontaylordev/CleanArchitecture ­– Дата доступа: 25.04.2021.
15. React and Redux Structure [Электронный ресурс] / Clean Architecture – Режим доступа: https://www.pluralsight.com/guides/how-to-organize-your-react-+-redux-codebase ­– Дата доступа: 25.04.2021.
16. React with Docker [Электронный ресурс] / Clean Architecture – Режим доступа: https://www.pluralsight.com/guides/using-react.js-with-docker ­– Дата доступа: 25.04.2021.