Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ЛЕСОСИБИРСКИЙ ПЕДАГОГИЧЕСКИЙ ИНСТИТУТ –
филиал Сибирского федерального университета

Высшей математики, информатики, экономики и естествознания
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Л.Н. Храмова
подпись инициалы, фамилия
«13» 06 2023 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.02 Информационные системы и технологии
код-наименование направления

РЕАЛИЗАЦИЯ ТЕХНОЛОГИИ КОРПОРАТИВНОГО ТАЙМ-МЕНЕДЖМЕНТА (НА ПРИМЕРЕ ООО «ГЕЙМОН ПРОДАКШН»)

Руководитель 13.06.2023 доцент, канд. пед. наук Е. В. Киргизова
подпись, дата должность, ученая степень инициалы, фамилия

Выпускник 13.06.2023 С. В. Куташевский
подпись, дата инициалы, фамилия

Нормоконтролер 13.06.2023 Е. В. Киргизова
подпись, дата инициалы, фамилия

Лесосибирск 2023

РЕФЕРАТ

Выпускная квалификационная работа по теме «Реализация технологии корпоративного тайм-менеджмента (на примере ООО «ГеймОн Продакшн»)» содержит 93 страница текстового документа, 56 иллюстраций, 6 таблиц, 44 использованных источника.

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ, NODEJS, TYPE-SCRIPT, ВЕБ-РАЗРАБОТКА, ВЕБ-СЕРВИС.

Цель исследования – теоретически обосновать и разработать веб-сервиса корпоративного тайм-менеджмента для управленческой деятельности руководителя как инструмент управления временем в IT-организации ООО «ГеймОн Продакшн».

Объект исследования – процесс управления временем в IT-организации.

Предмет исследования – процесс разработки веб-сервиса как инструмента управления временем в IT-организации ООО «ГеймОн Продакш».

Для достижения поставленной цели были сформулированы следующие задачи:

– на основе анализа учебной и научно-технической литературы по теме исследования определить понятийно-категориальный аппарат, критерии, принципы, методы корпоративного тайм-менеджмента;

– обосновать выбор методологии и архитектуры проекта, инструментов разработки для проектирования веб-сервиса корпоративного тайм-менеджмента;

– разработать и протестировать веб-сервис корпоративного тайм-менеджмента для управленческой деятельности руководителя IT-организации ООО «ГеймОн Продакшн».

В результате выполнения выпускной квалификационной работы разработан, апробирован и протестирован веб-сервис корпоративного тайм-менеджмента для управленческой деятельности руководителя IT-организации ООО «ГеймОн Продакшн».

СОДЕРЖАНИЕ

Введение 5
1 Корпоративный тайм-менеджмент как комплексная система управления временем в IT-компании 10
1.1 Сущность понятий «тайм-менеджмент», «корпоративный тайм-менеджмент». Стандарты корпоративного тайм-менеджмента 10
1.2 Критерии и принципы корпоративного тайм-менеджмента 12
1.3 Основные методы и инструменты корпоративного тайм-менеджмента 14
2 Проектирование веб-сервиса корпоративного тайм-менеджмента 18
2.1 Выбор методологии разработки 18
2.2 Выбор архитектуры проекта 23
2.2.1 Проектные архитектуры 23
2.2.2 Сервисные архитектуры 25
2.3 Выбор инструментов разработки 27
2.3.1 Клиентская часть 28
2.3.2 Серверная часть 28
2.3.3 Система управления базами данных 29
2.3.4 Инструменты для тестирования 31
3 Разработка веб-сервиса корпоративного тайм-менеджмента 33
3.1 Разработка клиентской части 36
3.1.1 Разработка статичной страницы первого запуска 38
3.1.2 Разработка статичной страницы авторизации 39
3.1.3 Разработка статичной страницы панели администратора 40
3.1.4 Разработка статичной страницы создания пользователей 40
3.1.5 Разработка статичной страницы удаления пользователей 41
3.1.6 Разработка статичной страницы панели руководителя 42
3.1.7 Разработка статичной страницы создания задач 43
3.1.8 Разработка статичной страницы проверки выполненных задач 43
3.1.9 Разработка статичной страницы списка задач 45
3.1.10 Разработка статичной страницы задачи 46
3.2 Разработка базы данных 46
3.3 Разработка серверной части веб-сервиса 50
3.3.1 Конфигурация проекта 51

3.3.2 Разработка точки сбора сервера 53
3.3.3 Разработка логгера 55
3.3.4 Разработка абстрактного контроллера 58
3.3.5 Разработка контроллера базы данных 59
3.3.6 Разработка базового сервиса 62
3.3.7 Разработка сервиса авторизации 66
3.3.8 Реализация сервиса отображения 69
3.4 Тестирование веб-сервиса 72
Заключение 74
Список использованных источников 76
Приложение А Листинг сервиса отображения 81
Приложение Б Листинг контроллера базы данных 86

ВВЕДЕНИЕ

В современном обществе социально-экономическая ситуация меняется с каждым днем, растут объемы получаемой информации, и в связи с этим особо остро востребована сегодня способность человека овладеть данным объемом знаний, поскольку поток информации во всех областях человеческой деятельности с появлением информационных технологий значительно возрос, а методы работы с информацией пока остаются прежними.

Можно выделить несколько причин, по которым значительно снижается эффективность работы современного человека.

Во-первых, слишком большой для восприятия и обработки поток информации, во-вторых, частые текущие задачи, которые требуют переключения внимания и оперативного решения, в-третьих, неумение найти в обработанном потоке необходимые данные, в-четвертых, значительная трата времени на систематизацию информации, в-пятых, действительная нехватка времени и многие другие причины.

В данных условиях особую важность приобретает умение управлять невосполнимым, неосязаемым ресурсом – временем, что позволит достигать запланированные результаты в установленные сроки. С этой целью используется технология организации времени и повышения эффективности его использования – технология тайм-менеджмента. В современном мире, где существуют понятия достоверности, полноты и актуальности информации, важно понимать, какую информацию стоит учитывать при принятии решений, а какую упустить.

Термин «тайм-менеджмент» отражает наиболее распространенное определение сферы управленческой деятельности, оформившейся в самостоятельное направление менеджмента организации к 70-м г.г. XX века. Тайм-менеджмент включает в себя всю совокупность технологий планирования работы сотрудника организации, которые применяются сотрудником самостоятельно для повышения эффективности использования рабочего

времени и повышения подконтрольности возрастающего объема задач. Иногда для обозначения таких технологий применяется также термин «самоменеджмент», «персональная (личная) организация труда», в отличие от общего менеджмента (корпоративной организации труда).

Потребность в тайм-менеджменте в настоящее время обусловлена следующими факторами:

– быстрые темпы изменений внешней среды;

– нестабильность мировой экономики и конъюнктуры рынка;

– постоянная потребность в изменениях и инновациях, которые уже становятся нормой, а не редким исключением;

– рост удельного веса нематериальных активов в стоимости организации и другие составляющие, непосредственно связанные со сферой функционирования организации.

Все они приводят к конкретным действиям в отношении сотрудников организации, а именно требуют делегирования работникам больших полномочий, постоянного увеличения количества решаемых задач, предоставления возможности принятия самостоятельных решений и планирования своей работы. При этом важно организовать деятельность персонала так, чтобы они четко осознавали свою цель и задачи, стоящие перед ними, грамотно расставляли приоритеты и могли осуществлять планирование своего времени.

В жизни важны не только личные качества руководителя, но и инструменты и методики планирования времени.

В практике выделяют личный и корпоративный тайм-менеджмент.

Личный тайм-менеджмент – это технология организации личного и рабочего времени и повышения эффективности его использования. Набор инструментов, позволяющий эффективно распоряжаться временными ресурсами [12].

Корпоративный тайм-менеджмент – совокупность методов использования инструментов личного тайм-менеджмента в целях повышения эффективности деятельности организации [9].

Инструменты корпоративного тайм-менеджмента, которые использует руководитель, можно автоматизировать с использованием современных интернет-технологий. Для этого необходимо использовать веб-сервис, это повысит эффективность используемого времени персонала, путём сбора всех поручений в одном месте.

Цель исследования – теоретически обосновать и разработать веб-сервиса корпоративного тайм-менеджмента для управленческой деятельности руководителя как инструмент управления временем в IT-организации ООО «ГеймОн Продакшн».

Объект исследования – процесс управления временем в IT-организации.

Предмет исследования – процесс разработки веб-сервиса как инструмента управления временем в IT-организации ООО «ГеймОн Продакшн».

Для достижения поставленной цели были сформулированы следующие задачи:

– на основе анализа учебной и научно-технической литературы по теме исследования определить понятийно-категориальный аппарат, критерии, принципы, методы корпоративного тайм-менеджмента;

– обосновать выбор методологии и архитектуры проекта, инструментов разработки для проектирования веб-сервиса корпоративного тайм-менеджмента;

– разработать и протестировать веб-сервис корпоративного тайм-менеджмента для управленческой деятельности руководителя IT-организации ООО «ГеймОн Продакшн».

Проблема является актуальной для IT-организации ООО «ГеймОн Продакшн», поскольку в производственном процессе отсутствует механизм распределения, фиксирования и контроля выполнения задач.

Практическая значимость исследования заключается в автоматизации процесса распределения, фиксирования и контроля выполнения задач, что приведёт к повышению эффективности используемого времени сотрудниками.

Методы исследования:

– теоретические: анализ учебной и научно-технической литературы по теме исследования, обобщение и сравнительный анализ;

– эмпирический: беседа, моделирование, тестирование программного продукта.

Результаты исследований представлены на следующих научных мероприятиях:

– Всероссийский конкурс научных работ «Молодежный научный потенциал» (10 ноября 2022 г. диплом I степени в секции «Информатика, информационные технологии и экономика»);

– XVII Международная научно-практическая конференция «Инновационные подходы развития экономики и управления в XXI веке» (27 января 2023 г., сертификат участника);

– Международная научно-практическая конференция «Исследования в современной науке» (30 марта 2023 г., сертификат участника);

– XI Международная научно-практическая конференция «Наука и образование» (31 марта 2023 г., диплом участника).

По результатам исследования опубликованы статьи:

1. Куташевский, С. В. Корпоративный тайм-менеджмент в управлении IT-организации / С. В. Куташевский // Актуальные проблемы преподавания дисциплин естественнонаучного цикла: тезисы докладов VI Всероссийской научно-практической конференции преподавателей, учителей, студентов и молодых ученых, Лесосибирск, 14–15 ноября 2022 года / Сибирский федеральный университет – Красноярск, 2022. – С. 81–84.

2. Куташевский, С. В. Интернет-технологии для реализации корпоративного тайм-менеджмента в управлении IT-организации / С. В. Куташевский // Материалы Международных научно-практических

мероприятий Общества Науки и Творчества (г. Казань) за январь 2023 года, Казань, 2023, С. 103-105;

3. Куташевский, С. В. Роль интернет интернет-технологий в создании корпоративного тайм-менеджмента в IT-организации / С. В. Куташевский // Материалы Международной научно-практической конференции: «Исследования в современной науке», Краснодар, 30 марта 2023 года. – С. 123-125;

4. Куташевский, С. В. Проектирование веб-сервиса для корпоративного тайм-менеджмента IT-организации на основе интернет-технологий / С. В. Куташевский // Сборник научно-методических статей XI Международная научно-практическая конференция «Наука и образование», Москва, 31 марта 2023 года. – С. 5-6.

Структура работы – работа состоит из введения, трёх глав, заключения, списка использованных источников и приложения, включающего 44 наименования. Результат работы представлен в 6 таблицах, 56 рисунках. Общий объем работы – 93 страницы.

1 Корпоративный тайм-менеджмент как комплексная система управления временем в IT-компании

Нехватка времени – проблема, с которой сталкивается любой руководитель. Имея положительные личные качества, такие как целеустремлённость, организованность, эффективность, можно страдать из-за плохого распределения времени и ресурсов подчинённых и коллег [19].

1.1 Сущность понятий «тайм-менеджмент», «корпоративный тайм-менеджмент». Стандарты корпоративного тайм-менеджмента

Никого не удивить тем, что в любой компании есть стандарты и правила по управлению деньгами. Использовав деньги из бюджета, нужно сделать отчёт, он един для всех. Но если учитывать, что время – деньги, то такие же отчёты необходимы в компании относительно времени [19].

Множество авторов, которые проводили исследования в области индивидуального и корпоративного тайм-менеджмента, приходили к разным формулировкам понятия, приведённым в таблице 1.

Таблица 1 – Поле мнений на понятие «тайм-менеджмент», «корпоративный тайм-менеджмент»

Автор	Понятие
Бронникова Е.М.	Тайм-менеджмент – технология, позволяющая использовать невосполнимое время жизни в соответствии с целями и ценностями [9]
Бронникова Е.М.	Корпоративный тайм-менеджмент – совокупность методов использования инструментов личного тайм-менеджмента в целях повышения эффективности деятельности организации [9]
Комелягина С.Е.	Тайм-менеджмент – это процесс организации и планирования своего времени между конкретными видами деятельности [17]
Борейшо М.А.	Тайм-менеджмент – система распределения времени на необходимые процессы в максимально эффективной последовательности [6]

Личная эффективность напрямую зависит от того, как выстроены процессы управления временем в подразделении и компании целом.

Поэтому индивидуальный и корпоративный тайм-менеджменты тесно связаны между собой:

– тайм-менеджмент – управление человеком собственной деятельностью, организацией выполнения задач и распределением всех ресурсов [3];

– корпоративный тайм-менеджмент – управление руководителем деятельностью сотрудников компании, организацией выполнения их задач, а также распределением рабочих ресурсов.

В таблице 2 приведены сходства и различия индивидуального и корпоративного тайм-менеджмента:

Таблица 2 – Сравнение индивидуального и корпоративного тайм-менеджмента

Характеристика	**Индивидуальный**	**Корпоративный**
Объект управления	Личность	Группа людей
Цель	Достигнуть максимальной эффективности при выполнении определённой индивидуальной задачи	Оптимизировать процесс и результат выполнения организационных функций, включая и коммуникацию.
Предмет	Личный опыт, навыки, знания, умения	Нормы, правила, стандарты, алгоритмы
Мотивация	Внутренняя: личный интерес, стремление быть лучше	Внешняя: неактивная и малоэффективная работа

Вышеизложенное позволяет определить понятие тайм-менеджмент как совокупность технологий и методов, которые способствуют повышению эффективности распределения времени человека.

Для внедрения корпоративного тайм-менеджмента в компанию, существуют множество корпоративных стандартов, к самым популярным относятся [19]:

1. Стандарт ведения календарей – основой данного стандарта является приложение Outlook (либо любая другая альтернативная программа). Все руководители и сотрудники планируют своё время исключительно в календарях Outlook. Различные приглашения так же осуществляются только

через Outlook, без телефонных звонков и электронных писем. Такая система даёт существенный прирост управляемости для руководителей и удобство работы для всех;

2. Стандарт управления задачами – данный стандарт подразумевает достаточно простые вещи, во-первых, хранение всех поручений руководителя в едином хранилище, во-вторых, выборка по тематике в различных управленческих ситуациях, одним кликом мыши руководитель может получить выборку задач по подчинённому, с которым он ведёт диалог, либо по проекту, которому ведётся собрание;

3. Стандарт управления совещаниями –основные аспекты, которые регулирует этот стандарт:

– кто и как готовит совещания, в каком виде и за какой срок участники извещаются об этом;

– формат совещания;

– как и кем отслеживаются принятые на совещании решения, кто контролирует исполнение.

4. Стандарт коммуникации – данный стандарт описывает, в каких случаях какие информационные каналы использовать, как регулируется срочность коммуникации, чем улучшить коммуникацию.

Таким образом, внедряя корпоративный тайм-менеджмент, руководители обеспечивают более эффективное использование времени сотрудниками компании, а используя вышеприведённые стандарты, сотрудники могут отчитываться о потраченном времени, как о ресурсе.

1.2 Критерии и принципы корпоративного тайм-менеджмента

Система управления рабочим временем, или система корпоративного тайм-менеджмента – это особая методика со своими принципами, критериями и рекомендациями по организации деятельности. Задача такой системы не просто распланировать рабочее время сотрудника, а сделать это эффективно [36].

Главной частью методики корпоративного тайм-менеджмента являются следующие критерии:

1. целеполагание – детальное и правильное представление цели позволит правильно провести планировку её достижения;

2. использование приоритетов – в процессе планировки достижения цели выделяются этапы и устанавливаются приоритеты выполнения задач;

3. время на отдых – из-за физиологических особенностей, человек не может работать постоянно, ведь усталость снижает эффективность работы почти до нуля;

4. борьба с хронофагами – это действия, которые отвлекают человека от его основной деятельности, мешают выполнять работу вовремя и качественно.

Не маловажную роль в системе играют принципы:

1. *принцип лягушки*

«Лягушка» – это мелкие, но неприятные задачи, которые люди не очень любят выполнять. Как правило, это дела, которые связаны с дискомфортом, стрессом, скукой и другими неприятными ощущениями.

У каждого человека свой собственный список «лягушек», например:

– общение с клиентами;

– обращение к строгому руководителю;

– написание скучных отчетов.

Задачу-лягушку можно распознать по двум характерным признакам:

– об этой задаче неприятно даже думать;

– эту задачу стараются отложить на потом.

Игнорирование таких задач может привести к негативным последствиям.

Выражение «съесть лягушку» в тайм-менеджменте означает выполнить какое-нибудь неприятное дело. Именно так и следует поступать с неприятными задачами, выполнять их как можно раньше и желательно в самом начале дня.

2. *принцип слона*

В терминологии тайм-менеджмента «слон» – это огромные, грандиозные задачи, к которым страшно подступиться.

Примеры таких задач:

– написать программу;

– выучить английский.

Проблема «слонов» в том, что долгое игнорирование таких задач приводит к неприятностям или упущенной выгоде.

Со «слонами» нужно бороться следующим образом, задачу нужно разбить на более мелкие дела, которые не выглядят такими большими.

3. *принцип «50-10-50»*

Во время работы, процесс решения задач может занять большое количество времени, за это время страдает не только здоровье, но и снижается концентрация. Что бы избежать этих последствий необходимо каждые 50 минут делать перерыв на 10 минут, при этом меняя вид деятельности.

Например, если задание выполняется на компьютере, то лучшим занятием в 10-минутый перерыв будет являться разминка.

Подведя итог вышесказанному, критерии и принципы корпоративного тайм-менеджмента играют не менее важную роль, чем внедряемая система. Если внедрить систему тайм-менеджмента, но не использовать принципы, то повышения эффективности не произойдёт, так как, например, не будет иметь значение, есть ли распорядок дня или его нет, если задачи «лягушки» не будут выполняться в первую очередь.

1.3 Основные методы корпоративного тайм-менеджмента

Глеб Архангельский в книге «Тайм-драйв: как успевать жить и работать» выделил множество эффективных правил и методик тайм-менеджмента. Не все методы относятся к корпоративному тайм-менеджменту, но другие разработаны именно для него.

Самые популярные из них приведены ниже [36].

Пирамида Франклина – этот метод направлен на долгосрочное планирование целей и их достижение, визуализация пирамиды Франклина приведена на рисунке 1.

Рисунок 1 – Пирамида Франклина

Метод представляет собой пирамиду ценностей сотрудника, разделенный на отдельные элементы, связанный между собой. Каждый отдельный элемент представляет собой планы в различном масштабе (день, месяц, год).

Преимуществами данного метода являются:

– наглядность;

– простота.

Главным недостатком пирамиды является то, что она не устойчива к изменениям.

Матрица Кови является дополненным методом Пирамиды Франклина. Пример приведён в таблице 3.

Таблица 3 – Матрица Кови

Сектора	**Срочно**	**Не срочно**
Важно	Сектор кризисов	Сектор качество
Неважно	Сектор призрачного кризиса	Сектор деградации

Метод состоит из 4 секторов [36]:

– сектор кризисов, куда заносятся срочные и важные дела;

– сектор качества, куда заносятся не срочные, но важные дела;

– сектор призрачного кризиса, куда заносятся срочные и неважные дела;

– сектор деградации, куда заносятся не срочные и не важные дела.

Данная матрица направлена на то, чтобы определить приоритетность задач, а также, чтобы показать наглядно, какие дела являются важными, а какие только отнимают время. Использование матрицы в совокупности с пирамидой Франклина, лишает данный метод каких-либо недостатков и является одним из лучших методов для распределения рабочего времени.

Метод GTD основан на том, чтобы использовать рабочее время только на решение поставленных задач, а отбор по приоритету должен осуществляться за счёт современных технических инструментов.

Девид Аллен, который является разработчиком данного метода, считает, что в работе нужно контролировать сам процесс, а не построение планов и выделение приоритетов [11].

Для увеличения эффективности рабочего времени предлагается использовать три модели:

1. Управление рабочим процессом через постоянный контроль над всеми обязанностями;

2. 6-уровневая модель обзора работы для дальнейших проектов, в модель входят: текущие дела, текущие проекты, круг обязанностей, проекты на

ближайшие годы, проекты на ближайшую пятилетку, жизненный цикл компании;

3. Естественный метод планирования.

Достоинством данного метода являются то, что GTD является наиболее целостным, обладает большим запасом практических инструментов.

Недостатками метода являются: необходимость жесткой дисциплины и самоконтроля сотрудников.

Выше перечислены популярные методы для корпоративного тайм-менеджмента, сравнение этих методов приведено в таблице 4.

Таблица 4 – Сравнительная таблица существующих методов корпоративного тайм-менеджмента

Метод тайм-менеджмента	**Представление**	**Устойчивость к изменениям**	**Сложность**
Пирамида Франклина	Представлена в виде пирамиды, где каждая ступень представляет определённый этап в достижении цели	Неустойчива	Наиболее простой и наглядный метод
Матрица Кови	Представлена в виде матрицы, отражающей то, каким делам отдаётся приоритет в исполнении	Устойчива	Простой из-за наличия готовой формы матрицы, однако требует навыков анализа задач
Метод GTD	Отказ от наглядности в пользу решения самих задач	Устойчива	Требует жесткой дисциплины и самоконтроля, подходит не всем организациям

Для разрабатываемой системы будут использоваться концепции метода матрицы Кови, это обусловлено тем, что задания для сотрудников будут добавляться руководителями команды различных отделов. Они имеют намного больше опыта и могут разделять задачи по приоритетам.

Таким образом, исследуя понятие корпоративного тайм-менеджмента, его критерии и принципы, методы и инструменты, можно сделать вывод о том, что использование различных систем корпоративного тайм-менеджмента является неотъемлемой частью любой организации.

2 Проектирование веб-сервиса корпоративного тайм-менеджмента

Самым важным этапом в реализации любого проекта является его проектирование. Именно на этом этапе можно найти и исправить все критические ошибки, а также упростить процесс разработки. Выбранная методология разработки поможет уложиться в сроки сдачи и определить возможные риски, выявить этапы разработки проекта и чётко обозначить процесс общения с заказчиком.

2.1 Выбор методологии разработки

В наше время существует огромное количество различных методологий. Некоторые из них устарели, а другие используются по сей день и не имеют других альтернатив.

Существуют 5 типов методологий:

1. каскадная модель;
2. итеративная модель;
3. спиральная модель;
4. agile-модель;
5. scrum-модель.

В рамках исследования, методологии Agile и Scrum затрагиваться не будут, по причине их сложности и требованиям к команде.

Рассмотрим оставшиеся модели подробнее.

Одной из самых популярных и применяемых долгое время является каскадная или водопадная модель жизненного, представленная на рисунке 2.

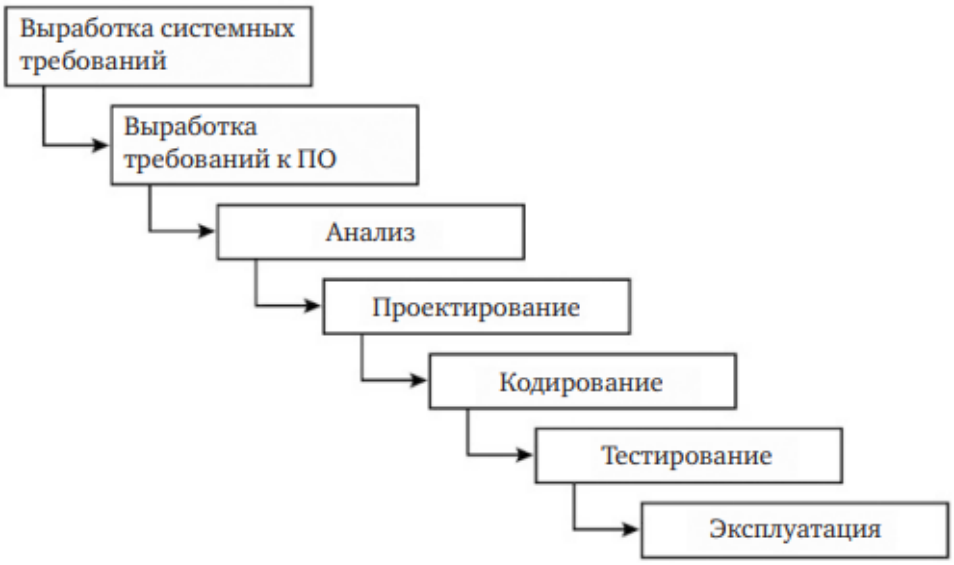


Рисунок 2 – Каскадная модель разработки

Она была разработана в 70-80 годы 20 века. Эта модель предполагает последовательность выполнения различных видов деятельности, начиная с выработки требований и заканчивая сопровождением, с четким определением границ между этапами, на которых набор документов, созданных на предыдущем этапе, передаётся в качество входных данных для следующего [5].

Классическая каскадная модель предполагает только движение вперёд по этой схеме: все необходимые требования должны быть сформированы на предыдущих этапах.

Данная модель создана для ИС, в которой в начале разработки можно точно и полно сформулировать технические требования.

Достоинства модели:

– с начала разработки имеется жесткий план и временной график по всем этапам проекта;

– на каждом этапе формируется законченный набор требований и документов, нужных для реализации следующего этапа;

– благодаря спланированным этапам, выполняемым в логической последовательности, возможно спланировать сроки завершения всех работ и соответствующие затраты.

Недостатки модели:

– необходимость в формировании всех требований ИС и невозможность их динамического изменения;

– непригодность промежуточного продукта для использования;

– позднее обнаружение проблем, связанных со сборкой;

– отсутствие участия заказчика в процессе разработки (заказчик участвует в начале жизненного цикла при разработке требований и в конце во время приемочных испытаниях)

Таким образом, использование каскадной модели разработки рекомендуется использовать при разработке малых ИС без масштабирования с небольшим количеством требований, а также без их изменений во время разработки.

Итеративная модель представляет жизненный цикл ИС в виде множества итераций, когда последовательно уточняются требования заказчика, наращиваются и детализируются функциональные возможности разрабатываемой ИС [43].

Цель каждой итерации – получение работающей версии ИС, включающей функциональность всех предыдущих и текущей итераций. Результатом последней итерации является готовая ИС со всеми реализованными требованиями. Пример итеративной модели представлен на рисунке 3.

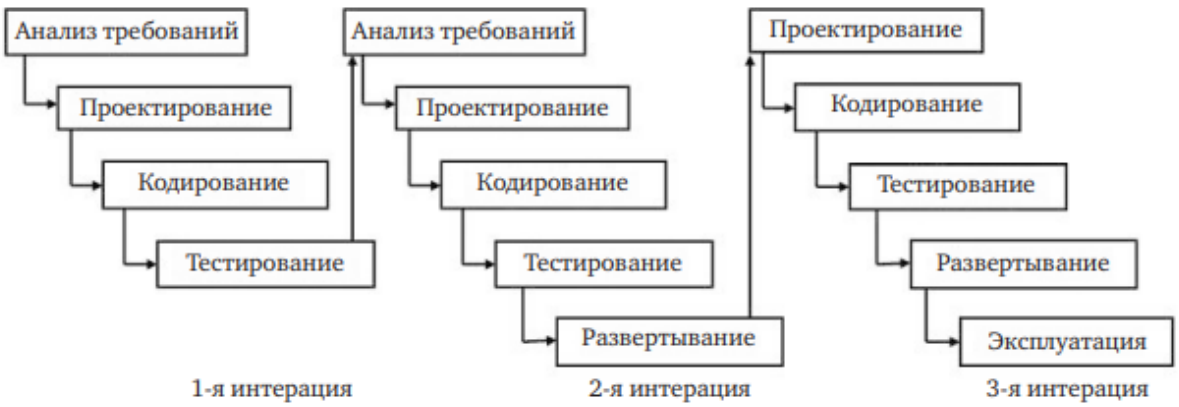


Рисунок 3 – Итеративная модель разработки

Достоинства модели:

– возможность межэтапных изменений требований;

– результатом каждой итерации является функционирующее ПО;

– простота тестирования для каждой итерации.

Недостатки модели:

– нет точной даты завершения разработки ИС из-за возможных изменений в требованиях;

– нелинейное увеличение стоимости обслуживания при масштабируемости системы;

– отсутствие полных требований при начале разработки.

И так, итеративная модель подходит для малых и средних масштабируемых ИС, требования которой формируются не с начала разработки. Влияние заказчика в этой модели выражено больше, поскольку в конце каждой итерации необходимо его подтверждение для перехода к следующему этапу.

Отличием спиральной модели от остальных является внимание рискам, влияющим на организацию жизненного цикла ИС. Каждый виток спирали соответствует итерационной модели создания части системы, для которой уточняются цели и характеристики проекта, пример представлен на рисунке 4 [5].

Рисунок 4 – Спиральная модель разработки

Одной из главных проблем этой модели является определение момента перехода на следующий этап. Решением данной проблемы выступает ввод временных рамок, которые будут ограничивать время жизни каждого из этапов жизненного цикла.

Достоинства модели:

– постоянное участие заказчика в процессе разработки;

– разбиение большого объема работы на небольшие части;

– повышенная вероятность предсказуемого поведения системы (Снижение рисков).

Недостатки модели:

– относительная дороговизна реализации проекта, так как оценка рисков после прохождения каждой спирали требует некоторых ресурсов в виде ресурсов;

– сложная структура жизненного цикла;

– возможность возникновения «бесконечной спирали», каждая ответная реакция заказчика на созданную версию ИС может порождать новую итерацию, что затягивает завершение работы над проектом.

Таким образом, данная модель подойдёт для больших ИС, в которых важно оценивать риски и постоянно общаться с заказчиком. Данная модель рекомендуется к использованию в больших IT-компаниях с несколькими командами разработчиков, чтобы уменьшить затрачиваемое время реализации.

Проведя исследование существующих моделей разработки, пришли к выводу, что для разрабатываемой системы будем использовать итеративную модель, это обусловлено следующим:

– отсутствие полных требований в начале разработки;

– простота тестирования разрабатываемых частей;

– относительная дешевизна изменения проекта.

2.2 Выбор архитектуры проекта

В коммерческой практике существует несколько архитектурных решений, которые используются постоянно. Они делятся на проектные и сервисные решения.

2.2.1 Проектные архитектуры

Проектная архитектура представляет собой базовую организацию системы, воплощенную в её компонентах, их взаимодействие между собой и окружением, а также принципы, определяющие проектирование и развитие системы [27].

Проектную архитектуру разделяют на:

1. *локальную архитектуру*

Данная архитектура использовалась до появления компьютерных сетей, все компоненты проекта располагались на одном компьютере.

Недостаток:

– работать в системе может только один пользователь одновременно.

2. *файл-серверную архитектуру*

С появлением компьютерных сетей появилась возможность хранить данные в файлах на выделенном специально для этой цели компьютере. Такой компьютер называется сервером.

Компьютер пользователей соединён с сервером сетью, поэтому доступ к данным могут получить сразу несколько пользователей. Пример файл-серверной архитектуры представлен на рисунке 5.

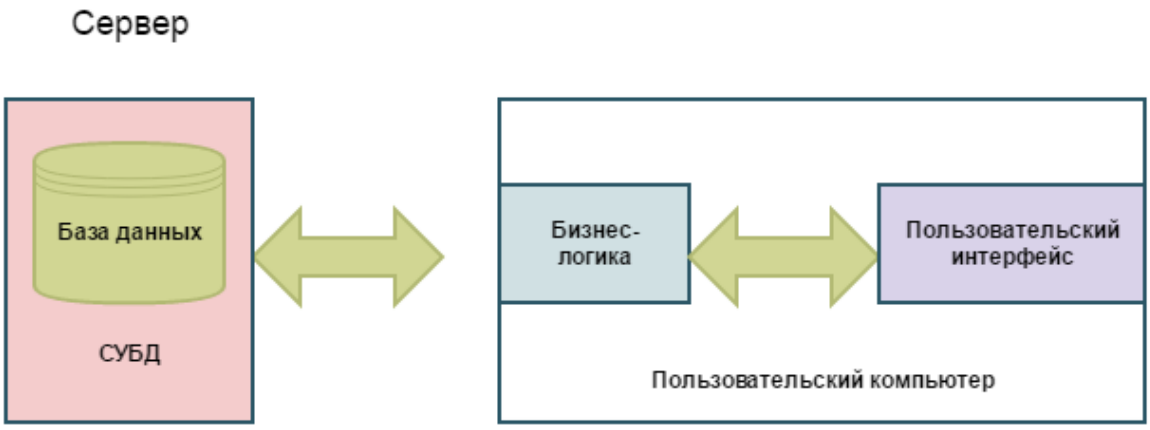


Рисунок 5 – Файл-серверная архитектура

Недостаток:

– сервер обеспечивает только функции хранения данных и предоставления доступа к ним.

3. *клиент-серверную архитектуру*

С увеличением размеров разрабатываемых систем увеличивалась и нагрузка на клиентскую часть. Для того, чтобы избавиться от этой проблемы, разработали клиент-серверную архитектуру. Пример клиент-серверной архитектуры представлен на рисунке 6.

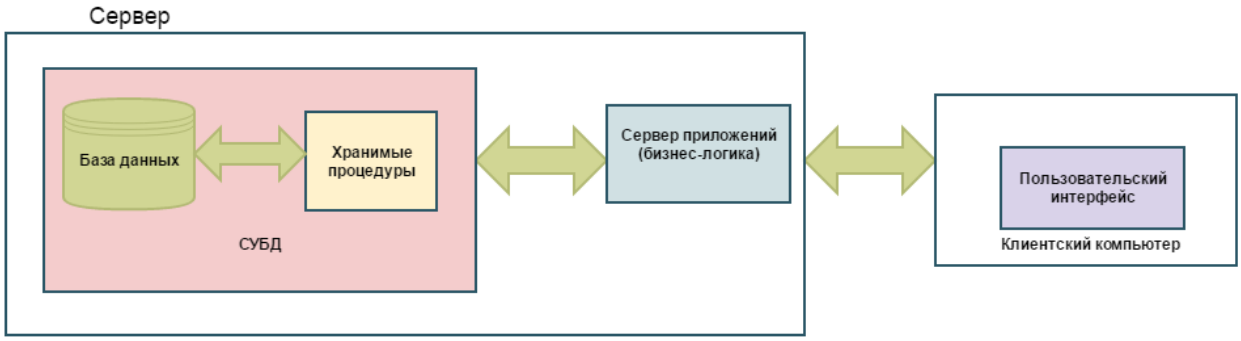


Рисунок 6 – Клиент-серверная архитектура

Суть этой архитектуры заключается в переносе сложных модулей системы с логикой на сервер. Использование сервера приложений позволяет

максимально разгрузить клиентские компьютеры и сделать обработку данных ещё более централизованной, что повышает скорость и надёжность системы.

Недостаток:

– для работы ИС с использованием такой архитектуры необходимо подключение к корпоративной сети или к сети Интернет.

Таким образом, изучив проектные архитектуры, можно сделать вывод о том, что для разрабатываемой системы применима клиент-серверная архитектура, выбор обусловлен тем, что систему планируется использовать в компании с большим числом сотрудников, а значит с большим числом пользователей.

2.2.2 Сервисные архитектуры

В отличии от проектных, сервисные архитектуры обеспечивают структуру отдельных компонентов системы, разделение компонентов на роли (контроллеры, сервисы, драйверы).

Сервисные архитектуры разделяются на:

1. *монолитную архитектуру*

Монолитная архитектура представляет собой один большой файл (монолит), который содержит и обрабатывает в себе всю бизнес-логику системы [28]. Пример монолитной архитектуры представлена на рисунке 7.

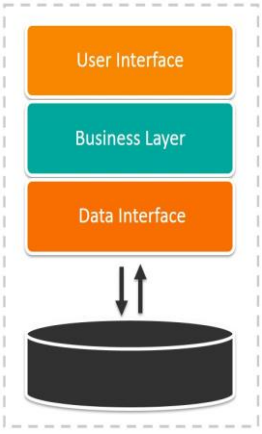


Рисунок 7 – Монолитная архитектура

Преимущества:

– простое развёртывание;

– удобная отладка;

– упрощенное сквозное тестирование.

Недостатки:

– сложность и дороговизна масштабируемости;

– слабая надёжность;

– сниженная скорость разработки.

2. *микросервисную архитектуру*

Микросервисная архитектура – подход к разработке приложений в виде набора небольших независимых сервисов, каждый из которых исполняется в виде отдельного самостоятельного процесса [2]. Все этапы разработки проходят для каждого сервиса отдельно.

Микросервисы не снижают сложность кода, но разбивают её на более мелкие части кода и упрощают его разработку и управление. Пример микросервисной архитектуры представлен на рисунке 8.

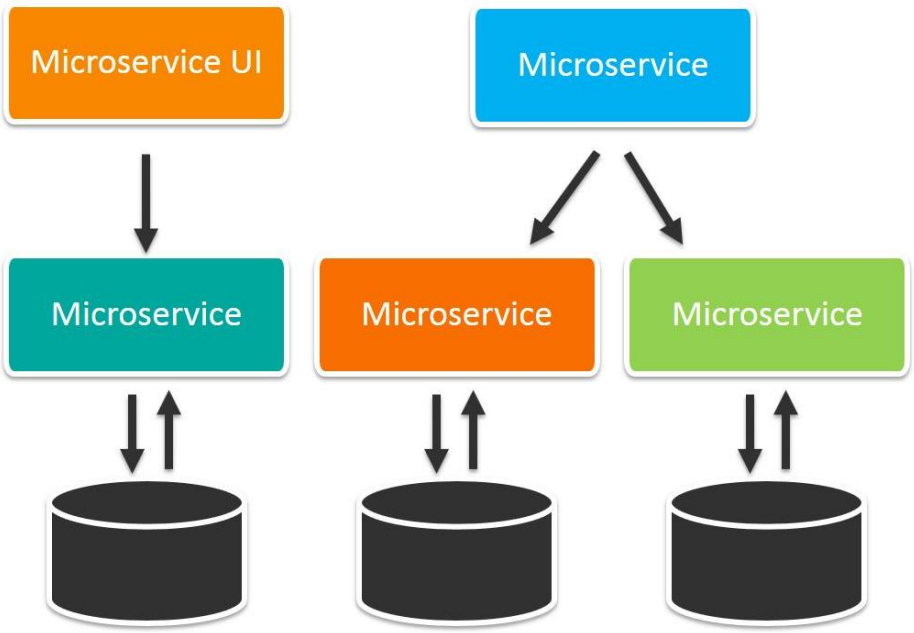


Рисунок 8 – Микросервисная архитектура

Преимущества:

– простота масштабирования;

– высокая надежность;

– простота тестирования.

Недостатки:

– отсутствие стандартизации;

– разрастание процесса разработки.

Проведя анализ существующих сервисных архитектур, пришли к выводу, что для разработки системы будет использоваться микросервисная архитектура, выбор обусловлен тем, что серверная часть должна быть реализована при помощи модулей, это позволит менять используемые модули (например, логгер или СУБД) почти не изменяя код.

Таким образом, выбор архитектуры является важным этапом в проектировании системы, от выбранного решения будет зависеть последующий жизненный цикл проекта.

2.3 Выбор инструментов разработки

Важным этапом в проектировании, от которого зависит, как будет выглядеть код системы, его сложность и реализация функционала, является выбор инструментов разработки. Для разрабатываемой системы, инструменты разделены на четыре типа:

– инструменты для клиентской части;

– инструменты для серверной части;

– инструменты СУБД;

– инструменты для тестирования.

2.3.1 Клиентская часть

Для реализации клиентской части будут использоваться HTML, CSSи JS. Этот набор инструментов является стандартным, так как других альтернатив для создания нет.

HTML (Язык разметки гипертекста) – используется для создания документов веб-страниц [33]. Данный язык сообщает браузеру, как нужно отображать страничный документ.

CSS (Каскадные таблицы стилей) – используется для управления представления страничных документов в браузере [33]. Каскадные таблицы используются для стилизации разработанных страничных документов, а также для их адаптации под разное разрешение экранов.

JavaScript (JS) – слабо типизированный язык сценариев, который наделяет веб-страницы интерактивностью и вариантами поведения [33]. JS используется для создания сценариев анимации, а так же для отправления http-запросов и обработки http-ответов.

Использование вышеперечисленных инструментов поможет реализовать современную клиентскую часть, которая будет поддерживать различные устройства, от смартфонов до компьютеров.

2.3.2 Серверная часть

Для реализации серверной части будут использоваться: современная платформа NodeJS, язык TypeScript, а также различные фреймворки.

NodeJS–платформа, основанная на исполняемой JavaScript-библиотеке Chrome, которая позволяет упростить создание быстрых масштабируемых сетевых приложений [44].

TypeScript – это надмножество JavaScript [38], которое вносит жесткую типизацию в JavaScript.

Express – минималистичный и гибкий фреймворк для Node.js веб-приложений, обеспечивающий набор возможностей для построения одно и многостраничных веб-приложений [7].

Nodemon (NodeJS monitoring) – это фреймворк для автоматического перезапуска приложения, который срабатывает при обнаружении изменений файлов в каталоге.

InversifyJS – это контейнер инверсии зависимостей для приложений разработанных на TypeScriptили JavaScript, предназначен для идентификации и внедрения зависимостей одних классов в другие.

DotEnv – это фреймворк, который упрощает работу с переменными окружения.

jsonWebToken – это реализация открытого стандарта для создания токенов доступа, основанный на формате JSON, предназначен для аутентификации в клиент-серверных приложениях.

Использование представленных инструментов поможет реализовать типобезопасный, современный и высокопроизводительный сервер с простотой масштабирования и тестирования, который сможет выдержать большую нагрузку от пользователей.

2.3.3 Система управления базами данных

В современном мире используется большое количество универсальных коммерческих СУБД, предлагаемых различными компаниям.

Среди множества современных СУБД сегодня можно выделить нескольких лидеров, занимающих более 90% рынка:

– Microsoft SQL Server

– Oracle

– PostgreSQL

– Линтер

– MySQL

У каждой СУБД есть свои плюсы и минусы, а также свои различия, по сравнению с другими системами, так, например, Microsoft SQL Server и Oracle являются самыми быстрыми, по сравнению с другими СУБД, но у данных систем нет свободного распространения, что означает, для использования необходима покупка лицензионного ключа.

Если рассматривать PostgreSQL и Линтер, то эти системы свободно распространяемые, нет необходимости покупать лицензионный ключ, но они намного медленнее, чем Microsoft SQL Server и Oracle.

Идеальным решением является MySQL – это структурированная реляционная система управления базами данных с открытым исходным кодом, хорошо известная в силу ее производительности, простоты в использовании и надежности [41]. У этой системы нет встроенного пользовательского интерфейса, что означает, доступ к базам данных происходит через командную строку, по этой причине эта система производительнее, чем PostgreSQLи Линтер.

При разработке веб-сервиса будет использоваться СУБД MySQL, выбор обусловлен следующим:

– бесплатное распространение;

– отсутствие пользовательского интерфейса – в рамках реализации веб-сервиса является преимуществом, поскольку подключение к хостингу СУБД, которое администрируется через командную строку, является бесплатным;

– постоянно обновляемый драйвер для NodeJS.

Подведя итог вышесказанному, MySQL является отличным выбором для разработки собственного веб-сервиса. Данная СУБД является гибкой и простой в использовании, а благодаря протоколам доступа система является одной из самых безопасных, для использования в различных веб-приложениях.

2.3.4 Инструменты для тестирования

Тестирование является последним этапом в разработки модуля или системы. Используя различные виды тестов в процессе и завершении разработки, возможно, избежать серьёзных ошибок в будущем, когда система будет эксплуатироваться.

Для реализации функционального тестирования будет использоваться инструмент Postman, это приложение, которое позволяет отправлять на серверную часть http-пакеты и обрабатывать полученные ответы, Postman имитирует запросы пользователей и помогает разработчикам в отслеживании каких-либо ошибок в построении логики модуля или системы в целом.

Проверка готовой системы называется сквозным тестированием. Существует множество инструментов для этого, в рамках реализации веб-сервиса будет использоваться node-memwatch. Это большой фреймворк, в котором есть множество функций, самой часто используемой и наглядной для тестирования на утечку памяти системы является stats. Данная функция отслеживает используемую системой память, к примеру, график памяти приложения с утечкой памяти представлен на рисунке 9.

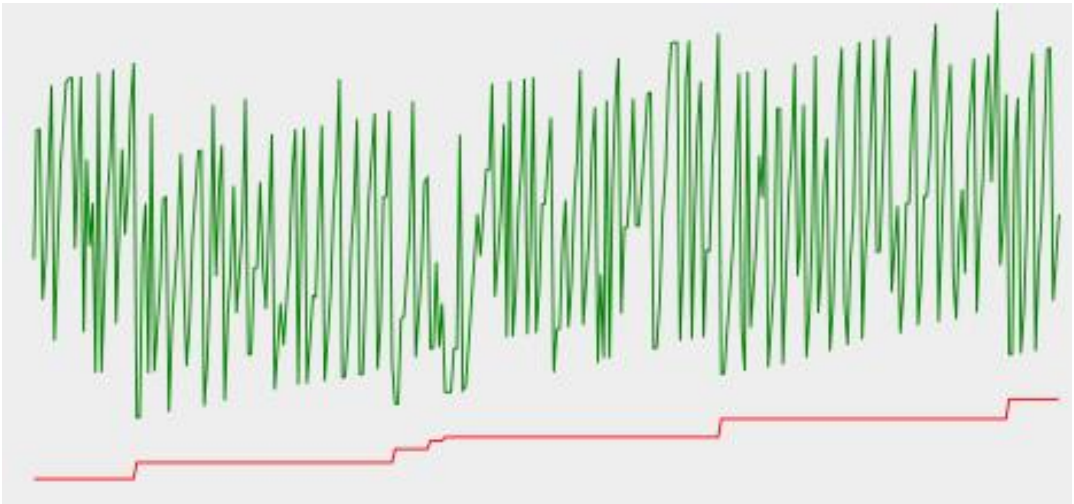


Рисунок 9 – График с утечкой памяти

Обратим внимание на скачки, это плохой знак, указывающий на то, что движку V8 приходится постоянно отчищать память, это серьёзно снижает производительность системы.

Подводя итог вышесказанному, используя современные инструменты тестирования возможно избежать большую часть проблем и ошибок, которые привели бы к выходу системы из строя во время эксплуатации.

3 Разработка веб-сервиса корпоративного тайм-менеджмента

Разработка является самым важным процессом в жизненном цикле системы, здесь реализуется требуемый функционал и выбранные проектные решения, а производительность системы напрямую зависит от умений программиста.

Как и проектирование, разработка делится на этапы, завершением которых является тестирование и внедрение.

В рамках выполнения выпускной квалификационной работы выделили следующие этапы разработки:

1. Разработка клиентской части:

– разработка статичной страницы «Первый запуск» (страница загружается при первом запуске системы, информирует пользователей о назначении системы и о необходимости в создании аккаунта администратора);

– разработка статичной страницы «Авторизация» (эта страница является начальной, пользователь представляет свои идентификационные данные, проходит процесс аутентификации и авторизации для доступа к функционалу веб-сервиса);

– разработка статичной страницы «Панель администратора» (страница предоставляет выбор из доступного функционала для пользователя с ролью администратора);

– разработка статичной страницы «Панель создания пользователей» (страница предоставляет администратору функцию создания нового пользователя системы, в ней указываются уникальный логин для идентификации пользователя, пароль для аутентификации, его имя и фамилия, а также роль (руководитель отдела или программист));

– разработка статичной страницы «Панель удаления пользователей» (страница позволяет администратору удалять пользователей системы, а также все задачи, которые связаны с этими пользователями);

– разработка статичной страницы «Панель руководителя отдела» (страница предоставляет выбор из доступного функционала для пользователя с ролью руководителя отдела);

– разработка статичной страницы «Панель создания задач» (на этой странице руководитель отдела создаёт задачи для подчинённых, для этого необходимо заполнить форму, представленную на рисунке 10);

Рисунок 10 – Форма создания задачи

– разработка статичной страницы «Панель обзора выполненных задач» (страница позволяет руководителю отслеживать выполненные задачи, проверять и, в случае неудовлетворительного результата, отправлять задание назад для исправления, либо если задача выполнена, то направить её в архив);

– разработка статичной страницы «Панель задач» (страница предоставляет список задач, адресованных пользователю с ролью подчинённого, которые отсортированы по датам сроков выполнения);

– разработка статичной страницы «Панель задачи» (страница отображает конкретную информацию по задаче, выбранной на предыдущей панели, здесь отображается подробное описание задачи, сроки выполнения, а также её приоритет).

2. Разработка серверной части:

– разработка точки сбора сервера (сервис предназначен для запуска всего сервера, здесь происходит инициализация других сервисов, а также размещается список-контейнер для реализации внедрения зависимостей);

– разработка логгера (сервис предназначен для более информативного вывода данных в консоль);

– разработка абстрактного контроллера (предназначены для создания обобщенных сущностей, на основе которых в дальнейшем предполагается создавать более конкретные сервисы. В рамках выпускной работы в абстрактном классе будет содержаться логика привязки контроллера к определённым http-запросам);

– разработка контроллера базы данных (описывает логику взаимодействия сервера с базой данных);

– разработка базового сервиса (сервис необходим для обработки события первого запуска);

– разработка сервиса аутентификации (предназначен для реализации логики идентификации, аутентификации и авторизации пользователей в системе);

– разработка сервиса отображения (отвечать за логику разделения ролей и последующего отображения нужных статичных страниц);

Параллельно разработке контроллера и сервисов необходимо тестировать эти объекты, а также реализовывать интерфейсы–это абстрактный стандарт между объектом и субъектом, который подтверждает, что в используемом функциональном объекте существуют требуемые методы.

Так же параллельно разработке необходимо указывать в отдельном контейнере пару (объект: интерфейс) для реализации внедрения зависимостей.

3.1 Разработка клиентской части

Для разработки клиентской части будем использовать Brackets – это свободно распространяемый текстовый редактор для веб-разработки.

Перед началом необходимо настроить рабочее пространство и установить следующие плагины:

– Emmet – разработан для повышения скорости введения текста путём предугадывания нужной команды, пример представлен на рисунке 11, а также ускоренного ввода html-тегов, рисунки 12 и 13;

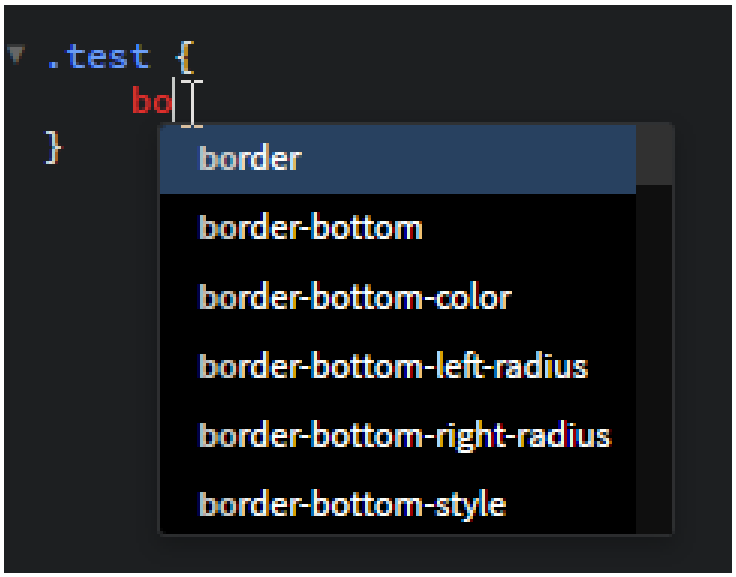


Рисунок 11 – Работа плагина Emmet

```
</head>
<body>
    div.testClass#TestID*5
</body>
</html>
```

Рисунок 12 – Команда для Emmet

```
<body>
    <div class="testClass" id="TestID"></div>
    <div class="testClass" id="TestID"></div>
    <div class="testClass" id="TestID"></div>
    <div class="testClass" id="TestID"></div>
    <div class="testClass" id="TestID"></div>
</body>
```

Рисунок 13 – Результат работы Emmet

– LiveServer – плагин, разработанный для отслеживания изменения статичной страницы в реальном времени;

– Beautify – плагин для автоматического форматирования кода при нажатии на комбинацию клавиш, пример представлен на рисунках 14 и 15

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <div class="block" id="1">
    <div class="block2">
    <p>Test</p>
    </div>
    </div>
</body>
</html>
```

Рисунок 14 – Не отформатированный код

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>

<body>
    <div class="block" id="1">
        <div class="block2">
            <p>Test</p>
        </div>
    </div>
</body>

</html>
```

Рисунок 15 – Отформатированный код

После того, как рабочее пространство было настроено, переходим к разработке.

3.1.1 Разработка статичной страницы первого запуска

Статичная страница содержит в себе два окна, в первом расположено приветствие пользователя и информация о веб-сервисе, во втором размещена форма создания пользователя с ролью администратор. Для создания необходимо ввести логин, пароль, имя и фамилию пользователя.

Данная страница отображается, если в базе данных нет пользователя с ролью администратора. Проверка происходит при переходе на любой обрабатываемый путь.

Визуализация первого окна страницы первого запуска изображена на рисунке 16.

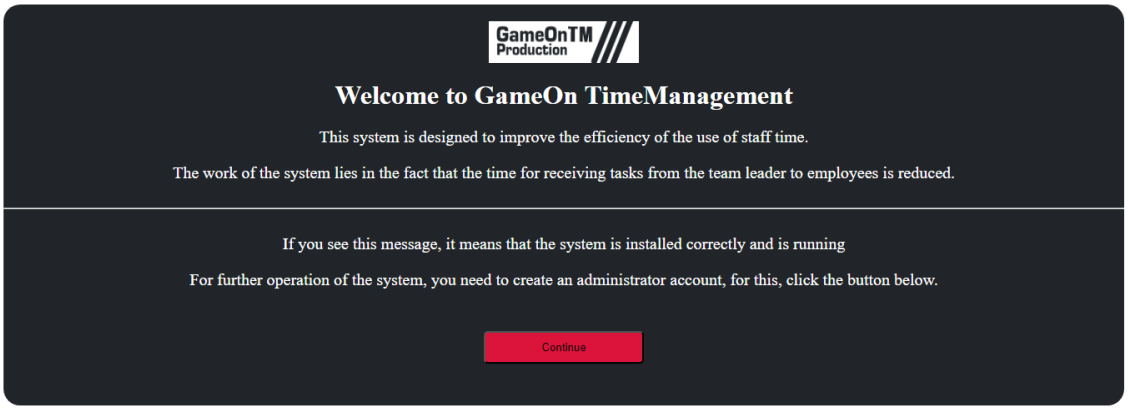


Рисунок 16 – Окно приветствия

Визуализация второго окна первой страницы изображена на рисунке 17.

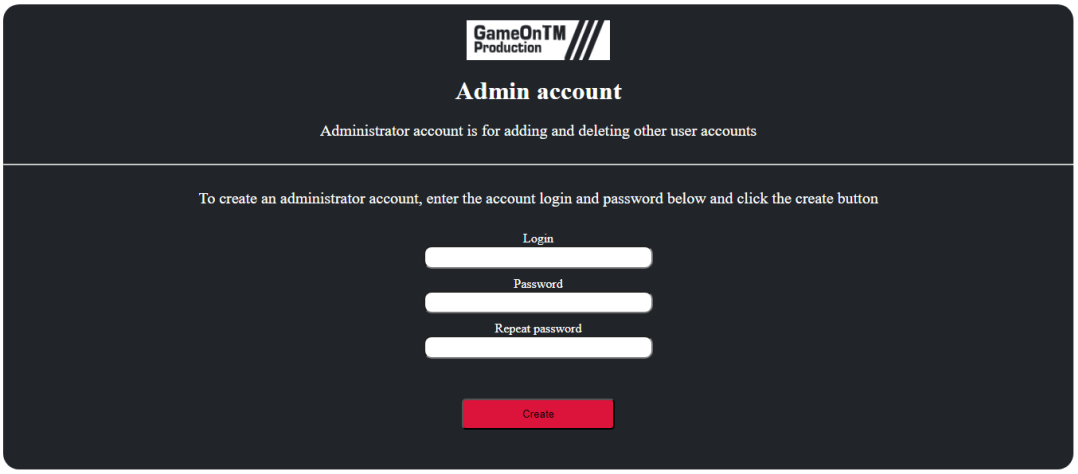


Рисунок 17 – Панель создания администратора

3.1.2 Разработка статичной страницы авторизации

Страница авторизации содержит в себе два блока, первый блок отвечает за разметку заголовка сайта, в котором расположен логотип веб-сервиса. Второй блок отвечает за отображение и обработку панели авторизации.

Для прохождения процесса авторизации необходимо указать логин и пароль. При нажатии на кнопку, отвечающую за авторизацию, JS-скрипт посылает на сервер POST-запрос с паролем и логином.

Визуализация страницы авторизации приведена на рисунке 18.

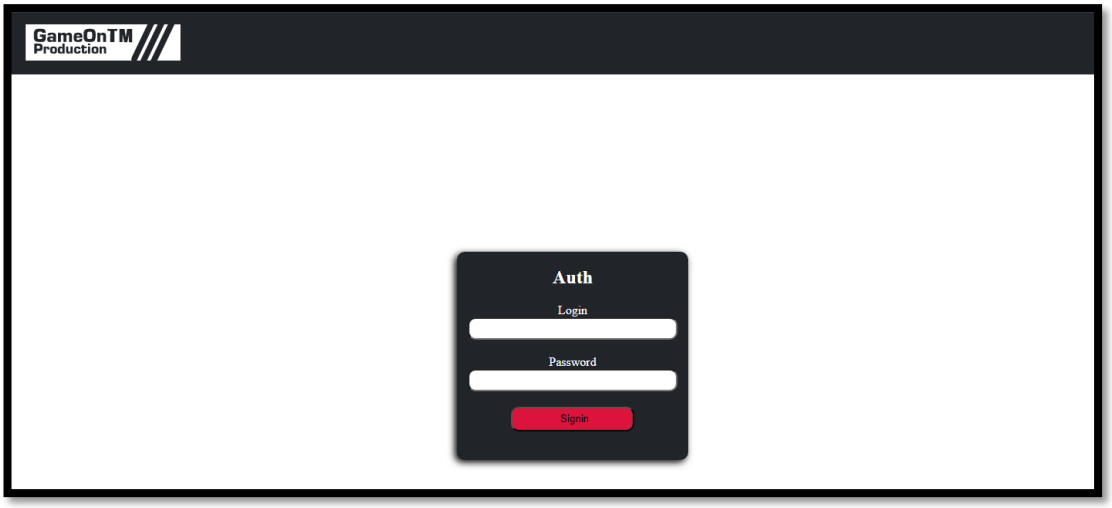


Рисунок 18 – Страница авторизации

3.1.3 Разработка статичной страницы панели администратора

Страница панели администратора содержит в себе заголовок сайта, в котором располагается логотип веб-сервиса, а также меню пользователя с указанным именем, должностью и кнопкой выхода с аккаунта.

После заголовка расположен основной блок с двумя кнопками, первая кнопка отвечает за переход на страницу создания пользователя, вторая – за переход на страницу удаления пользователя.

Визуализация страницы панели администратора приведена на рисунке 19.

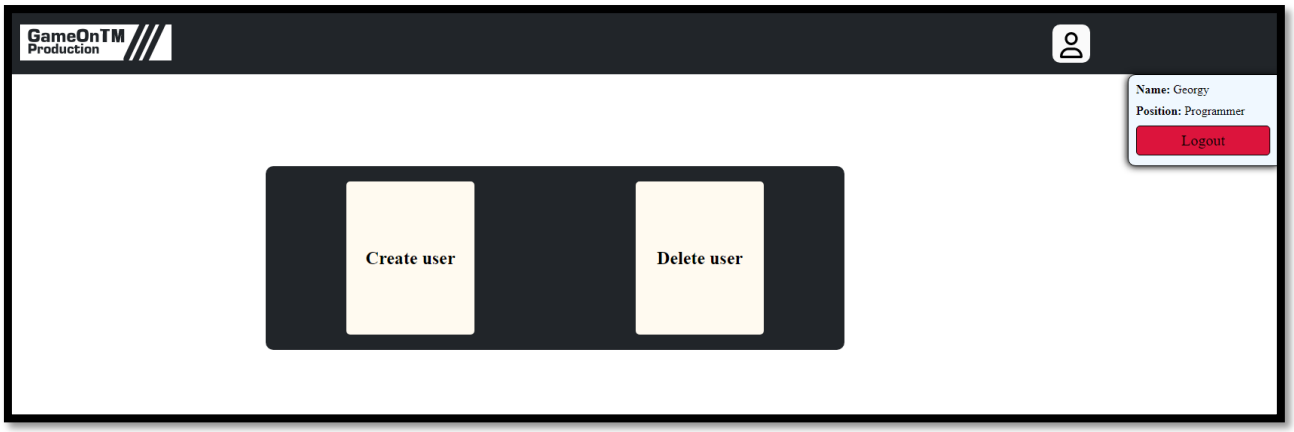


Рисунок 19 – Страница панели администратора

3.1.4 Разработка статичной страницы создания пользователей

Страница создания пользователя содержит в себе заголовок сайта, в котором располагается логотип веб-сервиса, а также меню пользователя с указанным именем, должностью и кнопкой выхода с аккаунта.

После заголовка расположен основной блок с формой создания пользователя, в которую входит: поле ввода логина, поле ввода пароля, поле ввода имения, поле ввода фамилии, поле выбора роли.

Визуализация страницы создания пользователя представлена на рисунке 20.

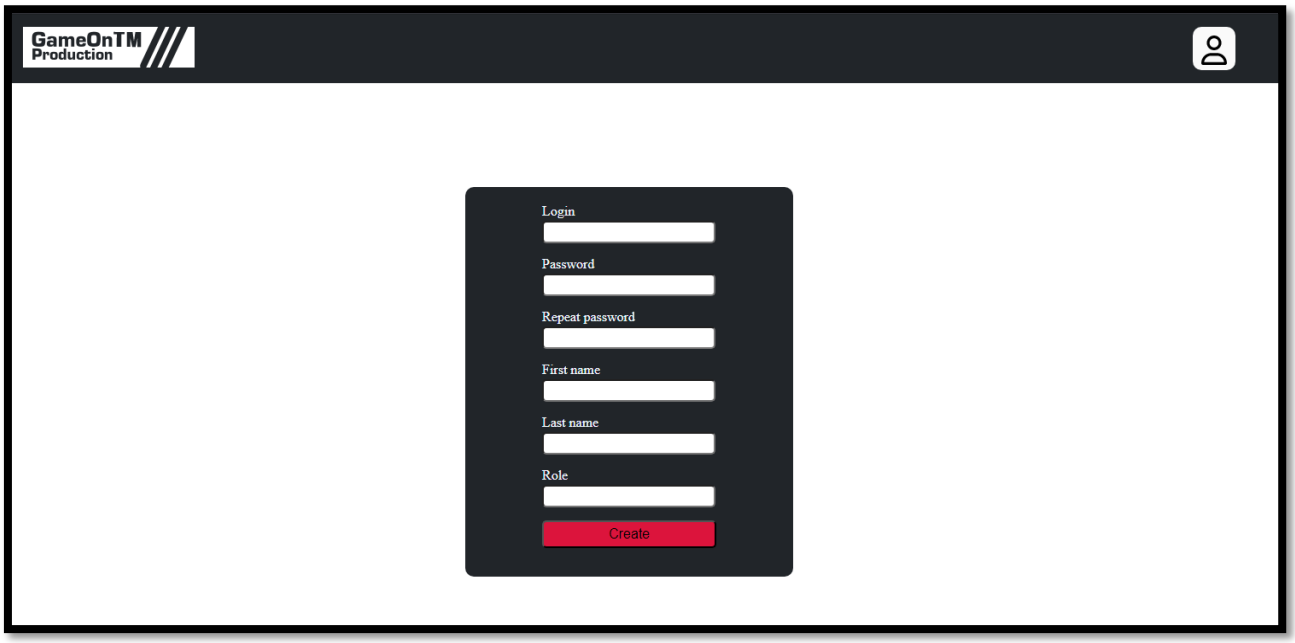


Рисунок 20 – Страница создания пользователя

3.1.5 Разработка статичной страницы удаления пользователей

Страница удаления пользователя содержит в себе заголовок сайта, в котором располагается логотип веб-сервиса, а также меню пользователя с указанным именем, должностью и кнопкой выхода с аккаунта.

После заголовка расположен основной блок с полем поиска и таблицей всех, зарегистрированных в системе, пользователей.

При нажатии по строке с пользователем, выбранная часть выделяется и данные из строки попадают в POST-запрос. При нажатии на кнопку удаления, выбранный пользователь, вместе с связанными с ним задачами, удаляется.

При попытке нажать на кнопку без выбора пользователя, администратора получит ошибку.

При поиске выбирается подходящий аккаунт и выделяется так, если бы по нему просто нажали курсором.

Визуализация страницы удаления пользователя приведена на странице 21.

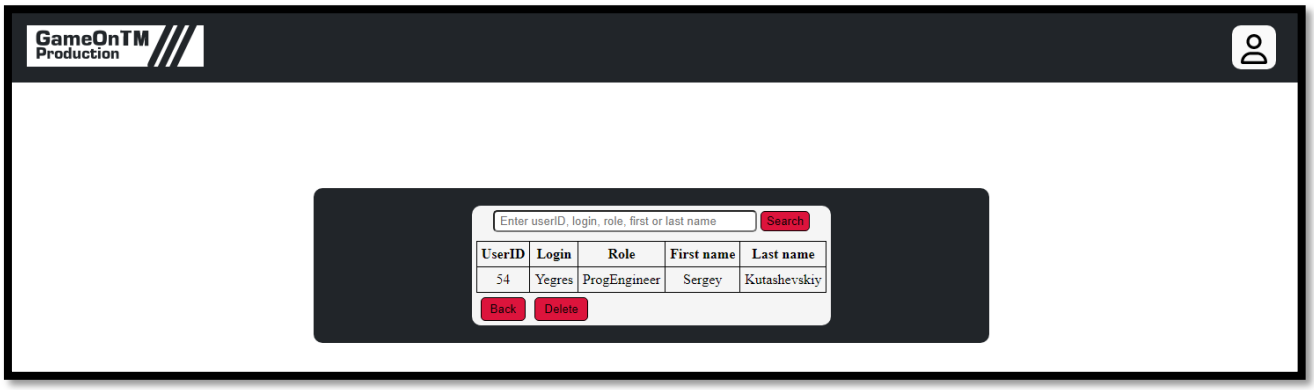


Рисунок 21 – Страница удаления пользователя

3.1.6 Разработка статичной страницы панели руководителя

Страница панели руководителя содержит в себе заголовок сайта, в котором располагается логотип веб-сервиса, а также меню пользователя с указанным именем, должностью и кнопкой выхода с аккаунта.

После заголовка расположен основной блок с двумя кнопками, первая кнопка отвечает за переход на страницу создания задачи, вторая – за переход на страницу обзора выполненных подчиненным задач.

Визуализация страницы панели руководителя отдела приведена на рисунке 22.

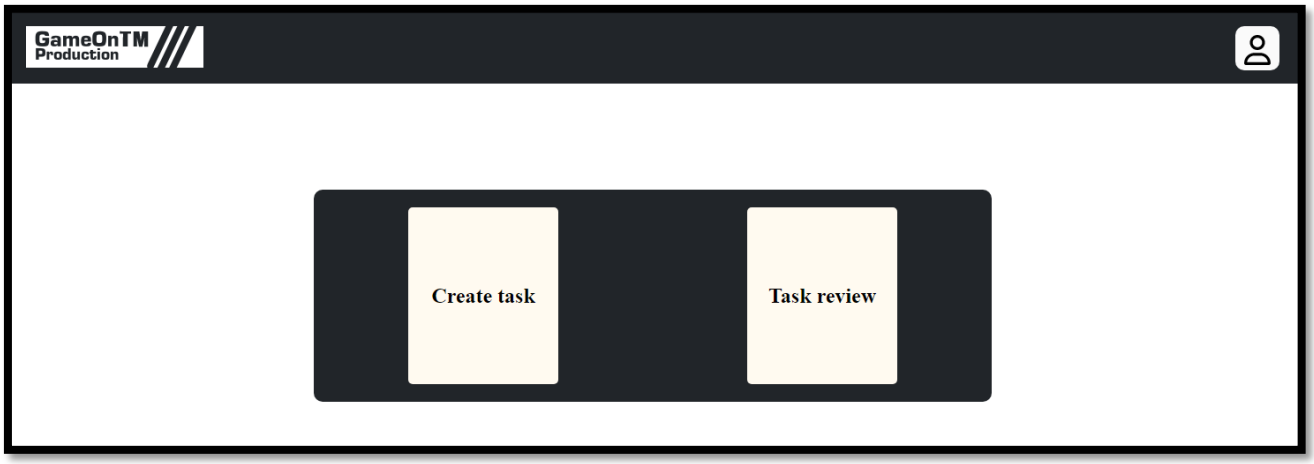


Рисунок 22 – Страница панели руководителя отдела

3.1.7 Разработка статичной страницы создания задач

Страница создания задач содержит в себе заголовок сайта, в котором располагается логотип веб-сервиса, а также меню пользователя с указанным именем, должностью и кнопкой выхода с аккаунта.

После заголовка расположен основной блок с формой создания задачи, в которую входит: заголовок задачи, срочность задачи, важность задачи, текст задачи, адресат задачи и сроки выполнения.

Визуализация страницы создания пользователя представлена на рисунке 23.

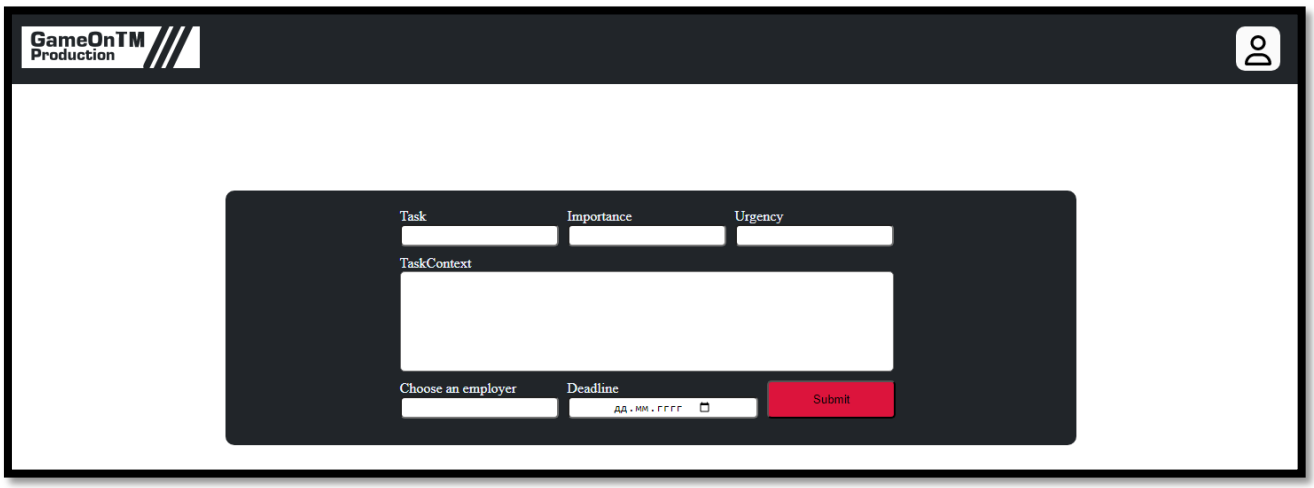


Рисунок 23 – Страница создания задачи

3.1.8 Разработка статичной страницы проверки выполненных задач

Страница проверки выполненных задач содержит в себе заголовок сайта, в котором располагается логотип веб-сервиса, а также меню пользователя с указанным именем, должностью и кнопкой выхода с аккаунта.

После заголовка расположен основной блок со списком выполненных задач. При загрузке страницы все задачи находятся в свернутом виде и отображают только сроки выполнения, номер задачи, приоритет задачи и заголовок задачи.

Свернутый список задач приведён на рисунке 24.

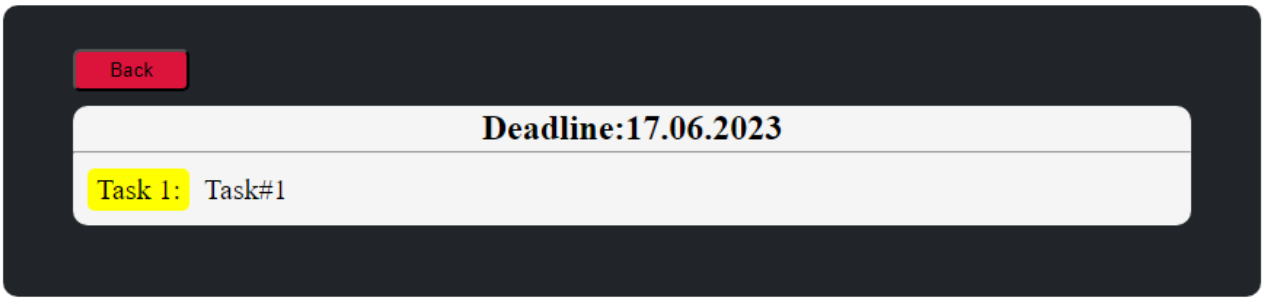


Рисунок 24 – Свернутый список задач

При нажатии на задачу, на экран выводится панель с текстом информации, двумя кнопками управления и блоком комментариев.

Развернутый список задачи представлен на рисунке 25.

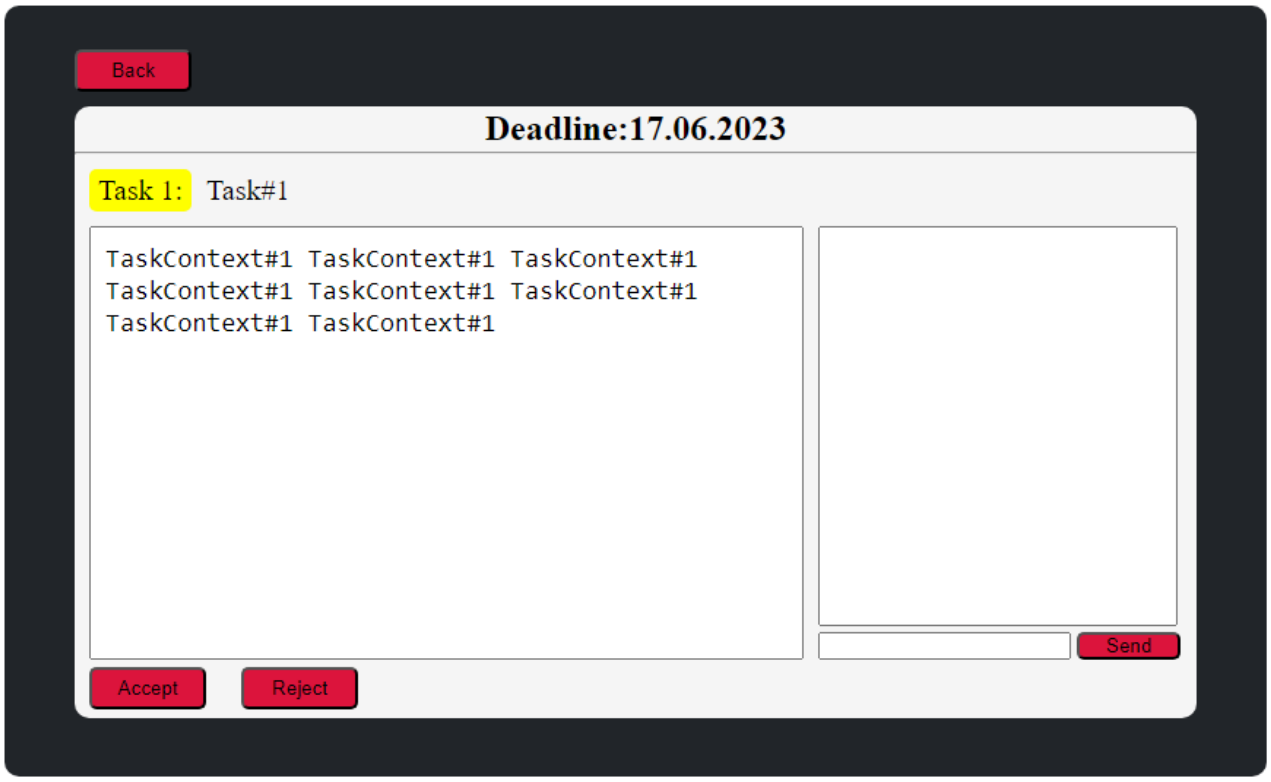


Рисунок 25 – Развернутый список задачи

Визуализация страницы обзора выполненных задач представлена на рисунке 26.

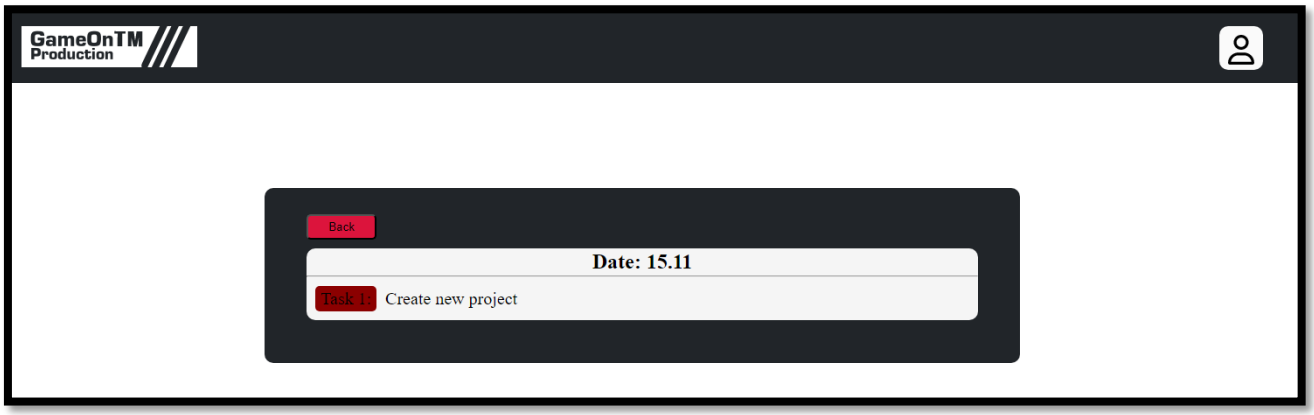


Рисунок 26 – Страница обзора задач

3.1.9 Разработка статичной страницы списка задач

Страница списка задач содержит в себе заголовок сайта, в котором располагается логотип веб-сервиса, а также меню пользователя с указанным именем, должностью и кнопкой выхода с аккаунта.

После заголовка расположен основной блок со списком задач подчиненного. В данном блоке находятся те задачи, которые создал руководитель. Каждый пользователь видит только те задачи, которые адресованы лично.

Список представляет собой отсортированные по датам блоки, в которых отображены приоритет и заголовок задачи, а при нажатии на задачу загружается страница конкретной задачи.

Визуализация страницы списка задач представлена на рисунке 27.

Рисунок 27 – Страница списка задач

3.1.10 Разработка статичной страницы задачи

Страница задачи содержит в себе заголовок сайта, в котором располагается логотип веб-сервиса, а также меню пользователя с указанным именем, должностью и кнопкой выхода с аккаунта.

После заголовка расположен основной блок с панелью, в которой расположена полная информация выбранной задачи, а именно: заголовок задачи, приоритет задачи, сроки выполнения задачи, текст задачи, кнопка для возврата к списку задач, кнопка для отправки задачи руководителю и блок с комментариями.

Визуализация страницы задачи представлена на рисунке 28.

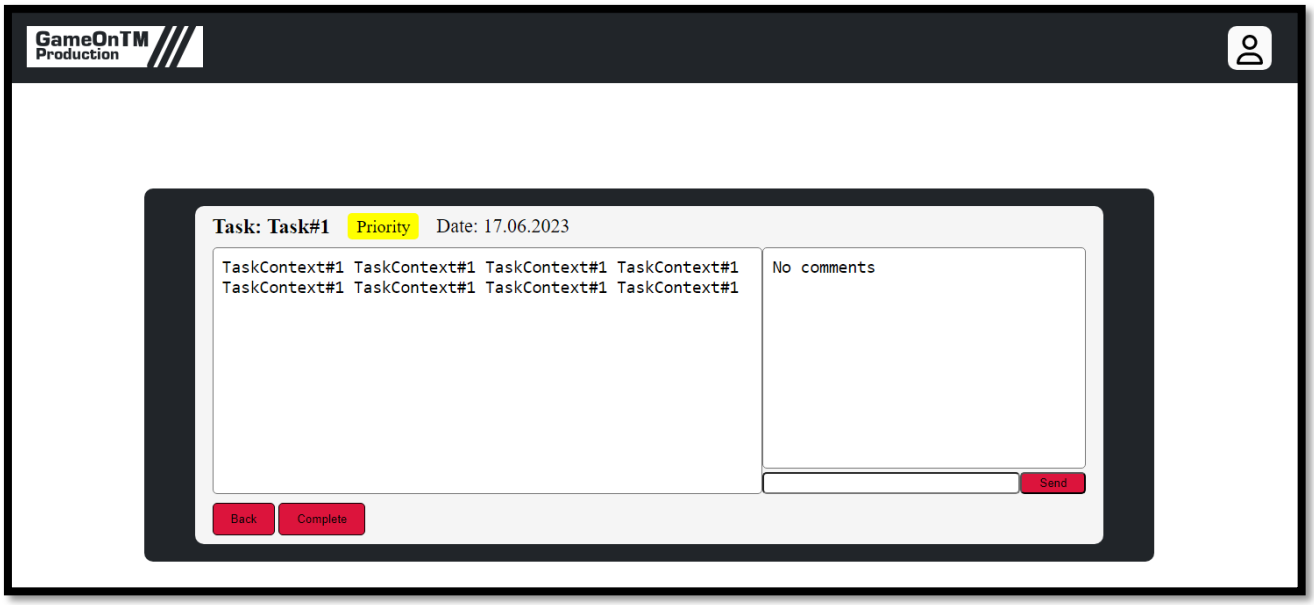


Рисунок 28 – Страница задачи

3.2 Разработка базы данных

Перед разработкой серверной части необходимо создать базу данных и таблицы, для данных, которые будут обрабатываться на сервере и отображаться на клиентской части.

Данные серверной части делятся на следующие категории:

– данные пользователя;

– данные задач;

– комментарии;

– архив выполненных задач.

На основе категорий необходимо создать следующие таблицы:

– Users (для хранения данных о пользователях);

– Tasks (для хранения данных о задачах);

– Comments (для хранения комментариев);

– TaskArchive (для хранения задач, которые были выполнены подчиненным и приняты руководителем).

Для создания базы данных и таблиц необходимо подключиться к серверу MySQL, для этого необходимо запустить «MySQLShell»и написать в командной строке следующие команды:

– «\sql» (переводит ввод в режим sql);

–«\connect User@addres:port» (подключается к серверу СУБД, где «User» - имя пользователя, «addres» - адрес, по которому доступен сервер, «port» - порт, по которому доступен сервер)

– «create database gameontm» (создаёт на сервере базу данных с именем «gameontm»)

База данных создана, далее необходимо добавить в базу таблицы:

1. «create table users (id int auto_increment, role varchar(15), fname varchar(15), lname varchar (25), login varchar(15), password varchar(25));» (создаёт таблицу users с полями:

– id типа число с автозаполнением;

– role типа нефиксированная строка с максимальной длиной 15;

– fname типа нефиксированная строка с максимальной длиной 15;

– lname типа нефиксированная строка с максимальной длиной 25;

– login типа нефиксированная строка с максимальной длиной 15;

– password типа нефиксированная строка с максимальной длиной 25.);

2. «create table tasks (id int auto_increment, LeadID int, target varchar(15), title varchar (50), context varchar(255), deadline date, priority float(3,2), complete int);» (создаёт таблицу tasks с полями:

– id типа целое число с автозаполнением;

– LeadID типа целое число;

– target типа нефиксированная строка с максимальной длиной 15;

– title типа нефиксированная строка с максимальной длиной 50;

– context типа нефиксированная строка с максимальной длиной 255;

– deadline типа дата;

– priority типа число с плавающей точкой;

– complete типа целое число.);

3. «create table comments (TaskID int, UserID int, comment varchar(255));» (создаёт таблицу comments с полями:

– TaskID типа целое число;

– UserID типа целое число;

– comment типа нефиксированная строка с максимальной длиной 255);

4. «create table taskarchive (ID int, LeadID int, target varchar(15), context varchar(255), priority float(3,2), completeDate date)» (создаёт таблицу taskarchive с полями:

– ID типа целое число;

– LeadID типа целое число;

– target типа нефиксированная строка с максимальной длиной 15;

– context типа нефиксированная строка с максимальной длиной 255;

– priority типа число с плавающей точкой;

– completeDate типа дата.).

Таким образом, используя SQL, была реализована база данных для хранения информации, которая используется системой. Созданные таблицы отображены на рисунке 29.

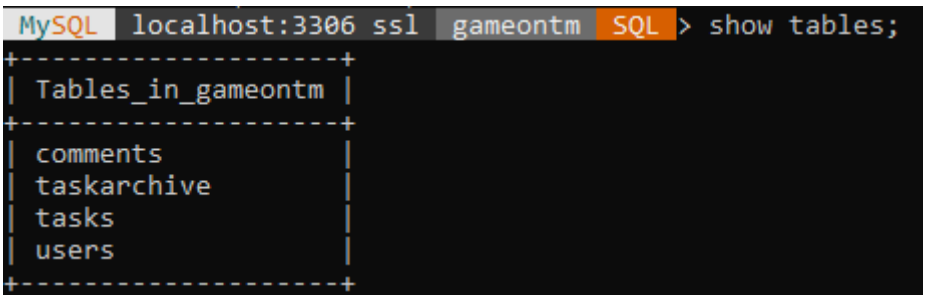


Рисунок 29 – Созданные таблицы в базе данных «gameontm»

3.3 Разработка серверной части веб-сервиса

Серверная часть является самой важной частью веб-сервиса, поскольку именно здесь принимаются, обрабатываются и отправляются данные, от правильности разработки серверной части зависит эффективность и стабильность системы в целом.

Для четкого представления этапов разработки серверной части, необходимо составить структурную схему, представленную на рисунке 30.

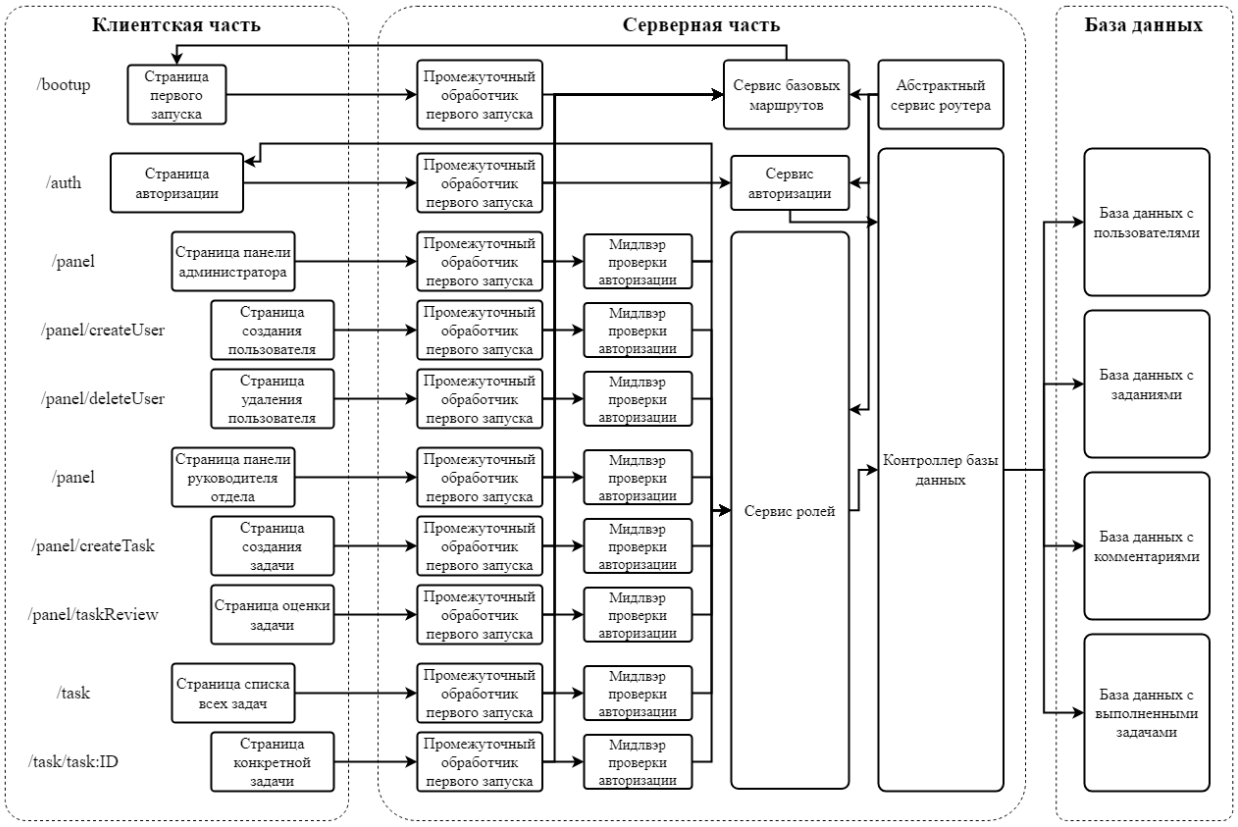


Рисунок 30 – Структурная схема веб-сервиса

3.3.1 Конфигурация проекта

Перед началом разработки серверной части веб-сервиса необходимо настроить окружение. Для этого создаём папку «backend», открываем эту папку как корневую в WebStorm и во встроенной консоли пишем следующую команду «npminit» (инициализация проекта).

После инициализации в папке с проектом появляется файл «package.json», который является конфигуратором проекта. В файле изменяем параметр: «type: commonjs» на «type: module» (этот параметр отвечает за метод импорта и экспорта модулей JS, module более современный и удобный метод).

Далее устанавливаем TypeScript. Пишем команду «npm i -D TypeScript» (i – сокращение от слова «install», -D – параметр, который указывает, что инструмент устанавливается только для разработки и во время эксплуатации использоваться не будет, TypeScript – название устанавливаемого модуля).

После установки TypeScript необходима корректировка конфигурации, для этого:

1. Напишем команду «npx tsc --init» (npx – система запуска пакетов, tsc – пакет TypeScript, --init – инициализация TypeScript);

2. В появившемся файле «tsconfig.json» изменим:

– «target: es6» (указывает версию JS, на которой будет писаться проект);

– «experimentalDecorators: true» (включает поддержку декораторов);

– «moduleResolution: node» (указывает, с чем будет работать TypeScript);

– «outDir: /dist» (директория, в которую будет компилироваться проект);

– «removeComments: true» (при компиляции удаляет комментарии из кода);

– «esModuleInterop: true» (указывает, что методом импорта и экспорта является module);

– «strict: true» (включает жесткую типизацию в проекте).

NodeJS и TypeScript настроены, последним этапом настройки окружения является установка всех нужных инструментов. Для этого напишем следующую команду «npm i argon2 cors dotenv ejs express inversify jsonwebtoken jwt-decode mysql reflect-metadata tslog»:

– argon2 – это инструмент, позволяющий хэшировать пароли и валидировать их;

– cors – это инструмент, позволяющий обрабатывать запросы с заголовками «origin» (эти заголовки необходимы, когда пользователь системы будет пытаться получить доступ к страницам сервиса с внешнего IP-адреса);

– dotenv – это инструмент, позволяющий добавлять свои данные в переменные окружения;

– ejs – это инструмент, который позволяет шаблонизировать статичные страницы и использовать различные конструкции языков программирования прямо внутри html-кода;

– express – это инструмент, позволяющий создавать сервера на базе http-запросов;

– inversify – это инструмент, позволяющий реализовать внедрение зависимостей с использованием контейнера;

– jsonwebtoken – это инструмент, который позволяет генерировать и валидировать токены доступа;

– jwt-decode – это инструмент, позволяющий расшифровывать JWT-токены;

– mysql – это драйвер, который позволяет делать запросы к базе данных mysql;

– reflect-metadata – это инструмент, расширяющий возможности декораторов;

– tslog – это инструмент, который позволяет выводить в консоль более информативную информацию.

После установки всех инструментов, которые будут использоваться во время эксплуатации, необходимо установить инструменты, которые будут

использоваться во время разработки, для этого напишем следующую команду «npm i -D @types/cors @types/dotenv @types/express @types/jsonwebtoken @types/mysql @types/tedious nodemon ts-node»:

– @types – это пакеты с типами разных инструментов, необходимые для разработки сервисов на TypeScript;

– nodemon – это инструмент, позволяющий системе автоматически перезапускаться при изменении кода;

– ts-node – это инструмент, который вносит поддержку NodeJS конструкций в TypeScript.

Таким образом, используя пакетный менеджер NodeJS и пакетный установщик NodeJS, провели настройку окружения, что является основным этапом в разработке серверной части веб-сервиса.

3.3.2 Разработка точки сбора сервера

Для создания точки сбора необходимо создать 2 сервиса, в первом будет реализован http-сервер на базе фреймворка express. Во втором будет содержаться контейнер для внедрения зависимостей, а также функция, которая запускает сервер.

Для разработки http-сервера необходимо создать внедряемый экспортируемый класс с нужными свойствами и асинхронным методом инициализации, который будет запускать express с выбранным портом.

Порт и некоторая другая важная информация, доступ к которой необходимо ограничить, будет доступен из файла внешнего окружения.

Разработанный сервис http-сервера представлен на рисунке 31.

Рисунок 31 – Листинг кода сервиса с http-сервером

После создания кода необходимо создать сервис запуска. Для этого создадим новую константу, в которую помещаем новый образ контейнера и добавляем внутрь разработанный http-сервер.

Создаём функцию для запуска сервера. Для этого загружаем в переменную с контейнером список зависимостей. Из контейнера внедряем в функцию сервис http-сервера и запускаем при помощи метода «init».

Разработанный сервис запуска http-сервера представлен на рисунке 32.

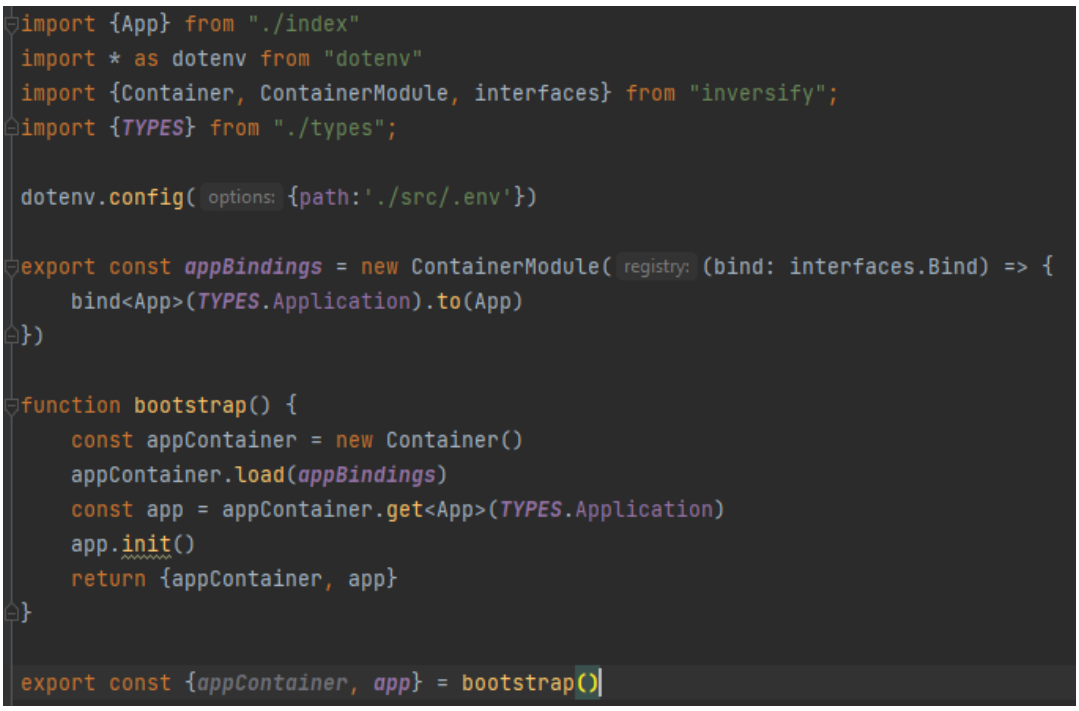


Рисунок 32 – Листинг кода сервиса запуска http-сервера

Для проверки запустим точку сбора сервера, результат представлен на рисунке 33.

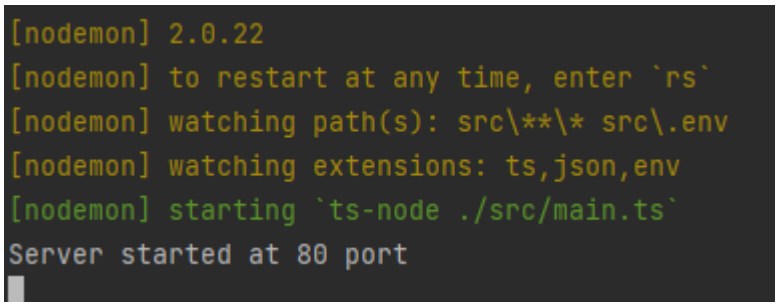


Рисунок 33 – Результат запуска сервера

В консоли появилась запись, которую указали в сервисе с http-сервером, это является показателем того, что сервер запустился и ошибок нет.

3.3.3 Разработка логгера

Для реализации логгера необходимо создать интерфейс и сервис логгера. Интерфейс содержит все необходимые методы, который будут использоваться

в методах других сервисов. Для создания необходимо создать экспортируемый интерфейс, в котором укажем все методы, их аргументы и что метод возвращает.

Реализованный интерфейс представлен на рисунке 34.

```
import {ILogObj, Logger} from "tslog";

export interface ILogger {
    logger: Logger<ILogObj>
    log: (...args: unknown[]) => void
    error: (...args: unknown[]) => void
    warn: (...args: unknown[]) => void
}
```

Рисунок 34 – Реализация интерфейса логгера

Далее реализуем сервис логгер, для этого создадим внедряемый, экспортируемый класс, который реализует ранее созданный интерфейс.

Укажем все нужные свойства, а также три метода:

– Метод вывода сообщения;

– Метод вывода ошибки;

– Метод вывода предупреждения.

Готовый сервис логгер представлен на рисунке 35.

Добавим логгер в контейнер, для этого в сервисе запуска http-сервера, представленного ранее на рисунке 32, добавим в контейнер зависимостей пару «ILogger: LoggerService».

Рисунок 35 – Сервис логгер

Результат проверки логгера представлен на рисунке 36.

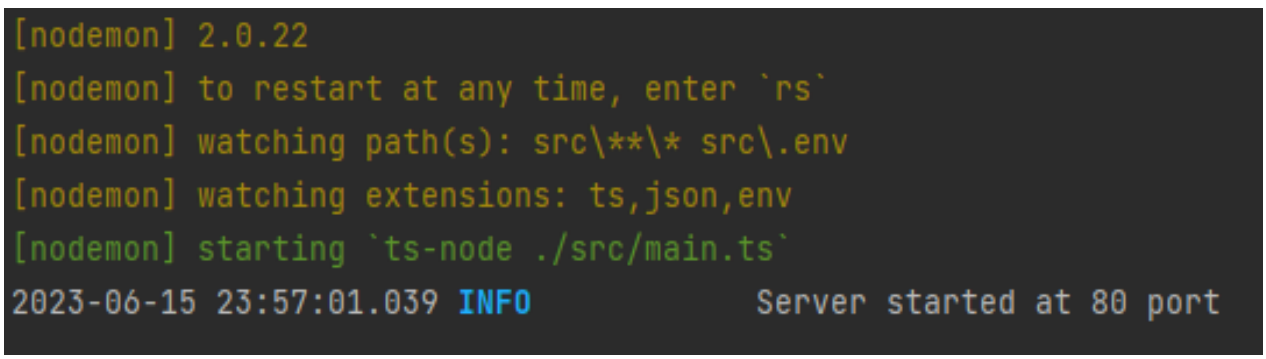


Рисунок 36 – Результат работы логгера

3.3.4 Разработка абстрактного контроллера

Перед началом разработки контроллера, необходимо реализовать интерфейс. Для этого создаём экспортируемый интерфейс, с нужными методами и свойствами.

Реализованный интерфейс представлен на рисунке 37.

```
import {NextFunction, Request, Response, Router} from "express";
import {IMiddleware} from "./middleware.interface";

export interface IRouteController {
    path: string
    func: (req:Request, res:Response, next:NextFunction) => void
    method: keyof Pick<Router, 'get' | 'post' | 'delete' | 'put' | 'patch'>
    middlewares?: IMiddleware[]
}
```

Рисунок 37 – Реализация интерфейса абстрактного контроллера

Далее реализуем контроллер. Для этого создадим внедряемый, экспортируемый класс с нужными свойствами и методами, который реализует ранее созданный интерфейс.

В защищенный метод «bindRoutes» собирает все привязанный пути, выводит в консоль и связывает с промежуточными обработчиками и конечной функцией.

Реализованный класс нигде не внедряется, но описан как внедряемый. Это необходимо для того, чтобы другие классы, которые его расширяют имели возможность внедрения, иначе компилятор TypeScript вызовет ошибку.

Таким образом, базовый контроллер позволяет избежать многочисленный повтор кода путем вынесения часто используемого функционала в отдельный метод.

Готовый абстрактный контроллер представлен на рисунке 38.

```
import {Response, Router} from 'express'
import {IRouteController} from "./route.interface";
import {inject, injectable} from "inversify";
import {ILogger} from "../logger/logger.interface";
import 'reflect-metadata'
import {TYPES} from "../types";

@injectable()
export abstract class BaseController {
    private readonly _router: Router
    constructor(@inject(TYPES.ILogger) private logger: ILogger) {
        this._router = Router()
    }

    get router(){
        return this._router
    }

    public send<T>(res:Response, code: number, message: T){
        res.type( type: 'application/json')
        return res.status(code).json(message)
    }

    protected bindRoutes(routes: IRouteController[]){
        for (const route of routes) {
            this.logger.log( args: `[${route.method}] ${route.path}`)
            const middlewares = route.middlewares?.map( callbackfn: m => m.execute.bind(m))
            const handler = route.func.bind(this)
            const pipeline = middlewares ? [...middlewares, handler] : handler
            this.router[route.method](route.path, pipeline)
        }
    }
}
```

Рисунок 38 – Реализация абстрактного контроллера

3.3.5 Разработка контроллера базы данных

Перед началом разработки контроллера базы данных необходимо реализовать интерфейс. Для этого создаём экспортируемый интерфейс с нужными свойствами и методами.

Реализованный интерфейс представлен на рисунке 39.

Рисунок 39 – Реализация интерфейса контроллера базы данных

Далее добавим в файл внешнего окружения переменные для базы данных: хост, имя пользователя, пароль и название базы данных.

Реализуем контроллер базы данных. Для этого создадим внедряемый, экспортируемый класс с нужными свойствами и методами, который реализует ранее созданный интерфейс.

Метод «connect» является проверочным и запускается вместе с http-сервером, если база данных запускается без ошибок, то метод вернёт сообщение о том, что база данных онлайн, иначе вернёт ошибку.

Разработанный контроллер базы данных представлен на рисунке 40.

```
import * as mysql from 'mysql'
import {Connection, MysqlError} from "mysql"
import {inject, injectable} from "inversify";
import {ILogger} from "../logger/logger.interface";
import {TYPES} from "../types";
import {IDBController} from "./DB.interface";
import 'reflect-metadata'


@injectable()
export class DBController implements IDBController{
    config: {}
    connection: Connection

    constructor(@inject(TYPES.ILogger) private logger: ILogger) {
        this.config = {
            host: process.env.DB_HOST,
            user: process.env.DB_USER,
            password: process.env.DB_PWD,
            database: process.env.DB_DATABASE,
            connectionLimit: 10,
            multipleStatements: true
        }
    }


    public async connect() {
        this.connection = mysql.createConnection(this.config)
        await this.connection.connect( callback: (err:MysqlError) => {
            this.connection.end()
            if (err){
                this.logger.error( args: `DataBase is offline: ${err.message}`)
            } else {
                this.logger.log( args: 'DataBase is online')
            }
        })
    }
}
```

Рисунок 40 – Реализованный контроллер базы данных

Добавим логгер в контейнер, для этого в сервисе запуска http-сервера, представленного ранее на рисунке 32, добавим в контейнер зависимостей пару «IDBController: DBController».

Проверяем работоспособность контроллера. Результат запуска точки сбора представлен на рисунке 41.

```
[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src\**\* src\.env
[nodemon] watching extensions: ts,json,env
[nodemon] starting `ts-node ./src/main.ts`
2023-06-16 01:07:23.822 INFO            Server started at 8001 port
2023-06-16 01:07:23.827 INFO            DataBase is online
```

Рисунок 41 – Результат работы контроллера базы данных

Таким образом, в консоли отобразился статус подключения к базе данных. Контроллер базы данных разработан без ошибок.

3.3.6 Разработка базового сервиса

Перед началом разработки базового сервиса необходимо реализовать интерфейс. Для этого создаём экспортируемый интерфейс с нужными свойствами и методами.

Реализованный интерфейс представлен на рисунке 42.

```
import {NextFunction, Request, Response} from "express";

export interface IBaseRootController {
    check: (req:Request, res:Response, next:NextFunction) => void
    bootup: (req:Request, res:Response, next:NextFunction) => void
}
```

Рисунок 42 – Интерфейс базового сервиса

Далее реализуем базовый сервис. Для этого создадим внедряемый, экспортируемый класс, который расширяет абстрактный класс и реализует ранее разработанный интерфейс.

В сервисе необходимо создать 2 метода, первый метод срабатывает, когда в базе данных есть пользователь с ролью администратора, второй наоборот, когда в базе данных отсутствует пользователь с ролью администратора.

Для реализации сервиса необходимо добавить новый метода в контроллер базы данных, который будет искать пользователя с ролью администратора. Для этого добавим в контроллер базы данных код, который представлен на рисунке 43.

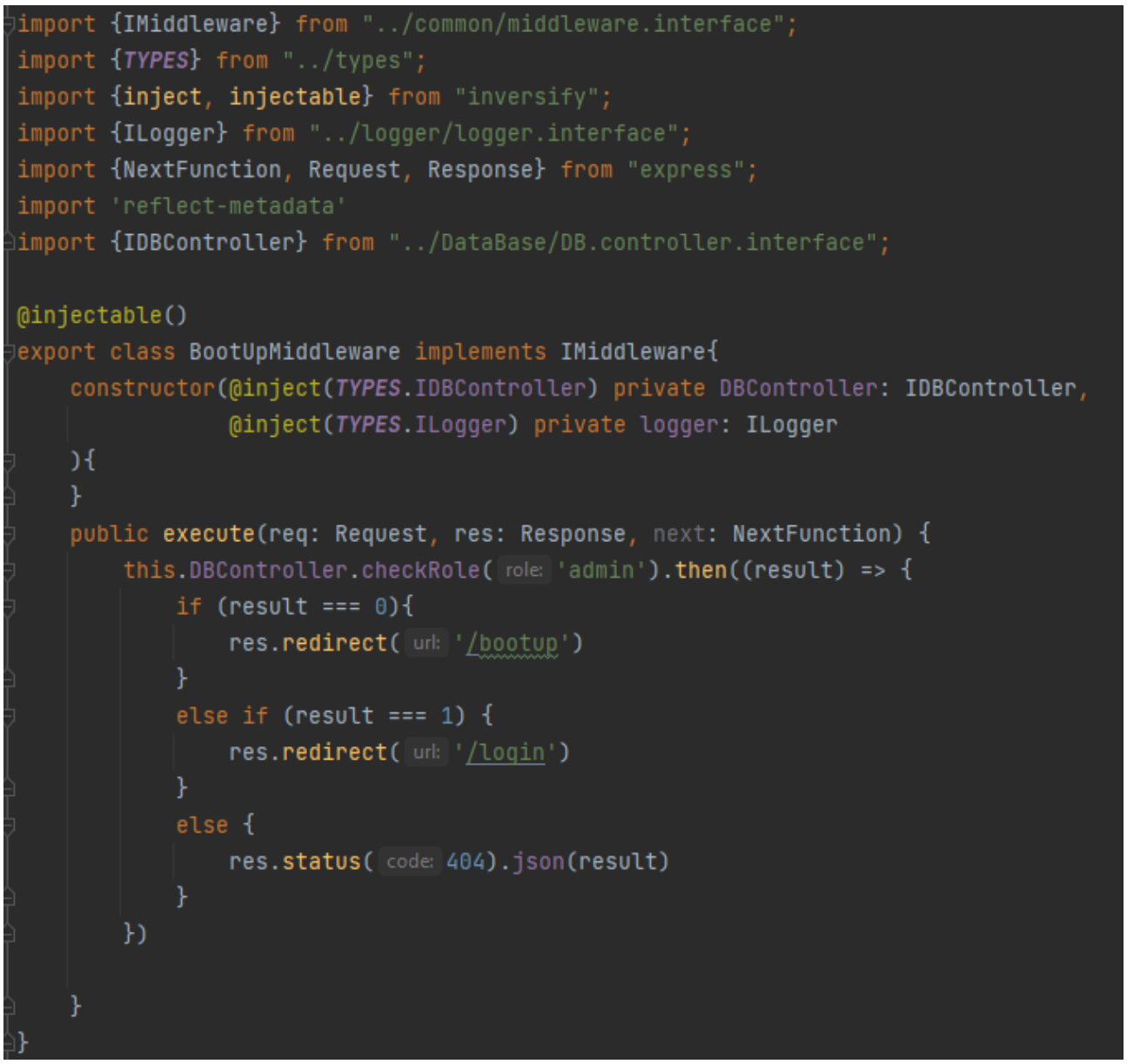
Рисунок 43 – Метод поиска администратора

Добавим в интерфейс контроллера базы данных информацию о новом методе.

Создадим промежуточный обработчик, который отвечает за логику обработки присутствия или отсутствия администратора.

Обработчик делает запрос к базе данных и проверяет длину полученного массива. В зависимости от длины, обработчик перенаправляет пользователя на соответствующий маршрут.

Разработанный промежуточный обработчик представлен на рисунке 44.



```
import {IMiddleware} from "../common/middleware.interface";
import {TYPES} from "../types";
import {inject, injectable} from "inversify";
import {ILogger} from "../logger/logger.interface";
import {NextFunction, Request, Response} from "express";
import 'reflect-metadata'
import {IDBController} from "../DataBase/DB.controller.interface";

@injectable()
export class BootUpMiddleware implements IMiddleware{
    constructor(@inject(TYPES.IDBController) private DBController: IDBController,
                @inject(TYPES.ILogger) private logger: ILogger
    ){
    }
    public execute(req: Request, res: Response, next: NextFunction) {
        this.DBController.checkRole( role: 'admin').then((result) => {
            if (result === 0){
                res.redirect( url: '/bootup')
            }
            else if (result === 1) {
                res.redirect( url: '/login')
            }
            else {
                res.status( code: 404).json(result)
            }
        })

    }
}
```

Рисунок 44 – Промежуточный обработчик базового сервиса

Далее в базовый контроллер укажем путь «/», данный символ обозначает, что обрабатываются все пути, укажем функцию, которая обрабатывает данный путь и ранее разработанный обработчик.

Реализованный базовый сервис представлен на рисунке 45.

```
import {inject, injectable} from "inversify";
import {BaseController} from "../common/base.controller";
import {TYPES} from "../types";
import {ILogger} from "../logger/logger.interface";
import {IDBController} from "../DataBase/DB.controller.interface";
import {IBaseRootController} from "./baseRoot.interface";
import 'reflect-metadata'
import {BootUpMiddleware} from "../Middlewares/BootUpMiddleware";
import {NextFunction, Request, Response} from "express";
import {CheckBootUpMiddleware} from "../Middlewares/CheckBootUpMiddleware";

@injectable()
export class baseRootController extends BaseController implements IBaseRootController{
    constructor(@inject(TYPES.ILogger) logger: ILogger,
                @inject(TYPES.IDBController) DBController:IDBController) {
        super(logger)
    this.bindRoutes( routes: [
                        {path: '/', method: 'get', func: this.check, middlewares: [new BootUpMiddleware(DBController, logger)]},
                        {path: '/bootup', method: 'get', func: this.bootup, middlewares: [new CheckBootUpMiddleware(DBController, logger)]}
                    ])
    }

    check(req:Request, res:Response, next:NextFunction){
        next()
    }

    bootup(req:Request, res:Response, next:NextFunction){
        res.render( view: 'bootup.ejs')
    }
}
```

Рисунок 45 – Реализованный базовый сервис

Протестируем сервис, для этого добавим в базу данных пользователя с ролью администратора, запустим сервер и в программе Postman отправим GET-запрос по пути «http://localhost/bootup».

Результат тестирования представлен на рисунке 46.

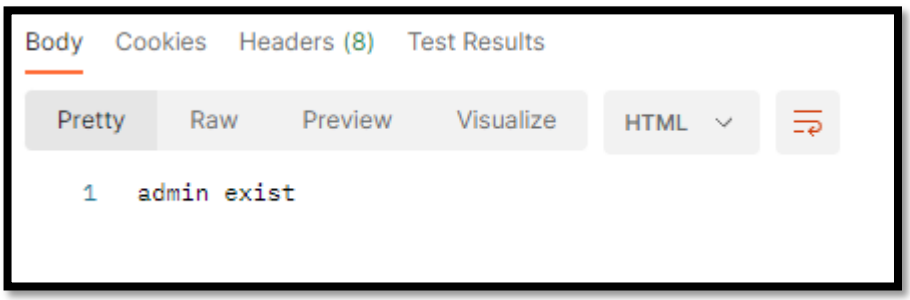


Рисунок 46 – Результат запроса

Сообщение говорит о том, что в базе данных пользователь с ролью администратора найден. Сервис работает корректно, ошибок не найдено.

3.3.7 Разработка сервиса авторизации

Перед началом разработки сервиса авторизации необходимо реализовать интерфейс. Для этого создаём экспортируемый интерфейс с нужными свойствами и методами.

Реализованный интерфейс представлен на рисунке 47.

```
import {NextFunction, Request, Response} from "express";

export interface IAuthController {
    login: (req: Request, res: Response, next: NextFunction) => void
    checkUser: (req: Request, res: Response, next: NextFunction) => void
    registerAdmin: (req: Request, res: Response, next: NextFunction) => void
    logout: (req: Request, res: Response, next: NextFunction) => void
}
```

Рисунок 47 – Интерфейс сервиса авторизации

Далее необходимо добавить в контроллер базы данных два новых метода, первый отвечает за проверку логина и пароля пользователя, второй за создание пользователя-администратора.

Для реализации первого метода пишем код, представленный на рисунке 48.

```
public async checkUser(login: string, password:string) {
    this.connection = mysql.createConnection(this.config)
    return new Promise( executor: (resolve) => {
        this.connection.query( options: `select * from users where login = '${login}'`, callback: async(error: MysqlError, results) => {
            this.connection.end()
            if (results.length != 0){
                if (await argon.verify(results[0].password, password)){
                    resolve(jwt.sign( payload: {id:results[0].id, role:results[0].role, login:results[0].login, fname:results[0].fname, lname:results[0].lname}, this.salt, options: {expiresIn:"8h"}))
                }
                else {
                    resolve( value: 0)
                }
            }
            if (results.length === 0) {
                resolve( value: 0)
            }
            if (error) {
                resolve( value: -1)
            }
        })
    })
}
```

Рисунок 48 – Метода проверки пользователя

Данный метод сверяет представленный логин и пароль, с тем, что хранится в базе данных, если данные полностью совпали, то генерируется JWT-токен, иначе возвращается ошибка.

Для реализации второго метода пишем код, представленный на рисунке 49.

Рисунок 49 – Метод создания пользователя

Данный метод добавляет нового пользователя в базу данных, аргументами метода являются: логин, пароль, имя, фамилия и роль пользователя.

Далее создадим новый внедряемый, экспортируемый класс, который расширяет абстрактный класс и реализует разработанный ранее интерфейс.

При помощи метода абстрактного класса свяжем пути, связанные с авторизацией, с обработчиками.

Создание класса и привязка путей представлена на рисунке 50.

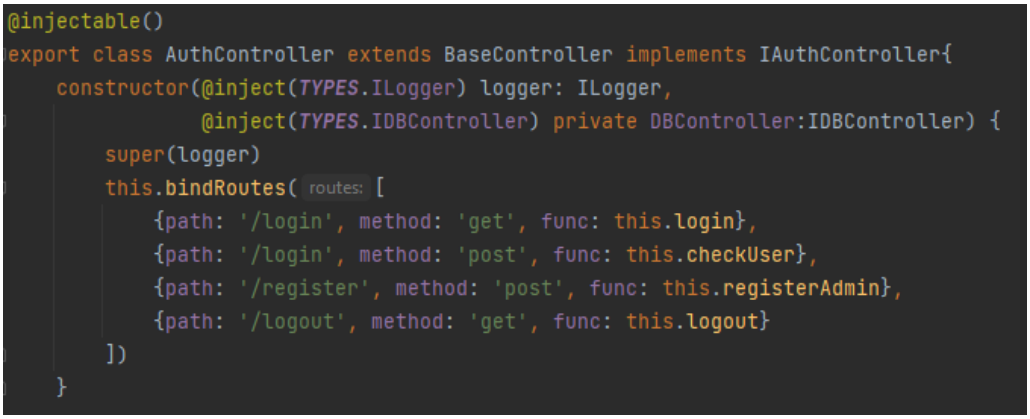


Рисунок 50 – Создание класса и привязка путей

Далее реализуем методы для каждого маршрута:

– GET «/login» отвечает за отправку страницы авторизации пользователю. Реализация метода «login» представлена на рисунке 51.

```
login(req: Request, res: Response, next: NextFunction){
    res.render( view: 'auth.ejs')
}
```

Рисунок 51 – GET метод «login»

– POST «/login» отвечает за поиск пользователя в базе данных, отправку «cookie» с JWT-токеном и доступом к панели пользователя. Реализация метода «login» представлена на рисунке 52.

```
checkUser(req: Request, res: Response, next: NextFunction) {
    res.clearCookie( name: 'bearer')
    this.DBController.checkUser(req.body.login, req.body.password).then((status) => {
        if (status != 0 && status != -1){
            res.cookie( name: 'bearer', status)
            res.redirect( url: 'panel')
        }
        else {
            res.status( code: 401).json(status)
        }
    })
}
```

Рисунок 52 – POST метод login

– POST «/register» отвечает за регистрацию пользователя-администратора. Реализация метода «login» представлена на рисунке 53.

Рисунок 53 – POST метод register

– GET «/logout» отвечает за очистку cookie и выход с аккаунта. Реализация метода «login» представлена на рисунке 55.

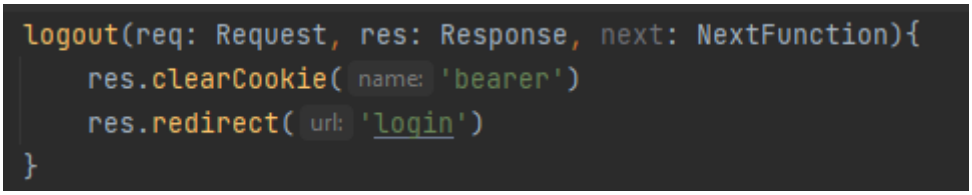


Рисунок 54 – GET метод logout

3.3.8 Реализация сервиса отображения

Перед началом разработки сервиса отображения необходимо реализовать интерфейс. Для этого создаём экспортируемый интерфейс с нужными свойствами и методами.

Реализованный интерфейс представлен на рисунке 55.

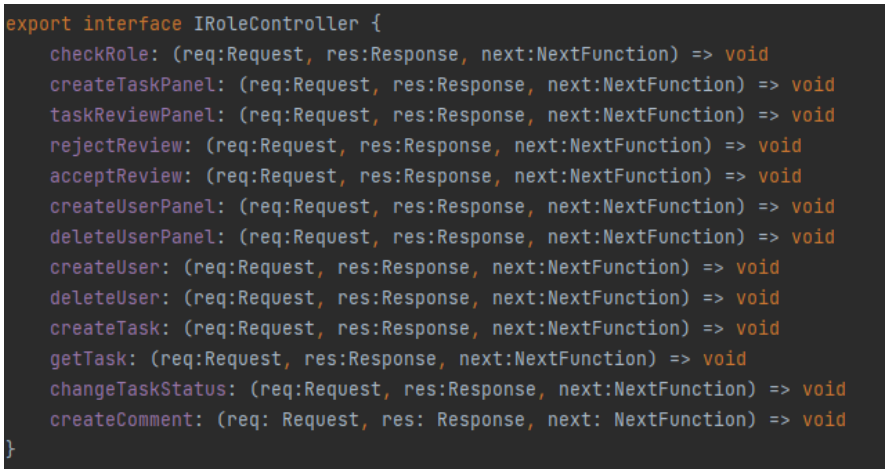


Рисунок 55 – Интерфейс сервиса отображения

Далее создаём внедряемый, экспортируемый класс, который расширяет абстрактный класс и реализует ранее разработанный интерфейс.

В классе сервиса отображения содержатся методы, приведенные в таблице 5.

Таблица 5 – Описание методов сервиса отображения

Название метода	**Обрабатываемый путь**	**Описание**
checkRole	GET «/panel»	Метод проверяет роль в JWT-токене и в зависимости от роли отправляет клиенту страницу (для администратора – панель администратора и т.д.)
createTaskPanel	GET «/panel/createTask»	Отправляет пользователю-руководителю страницу создания задания
taskReviewPanel	GET «/panel/reviewTask»	Отправляет пользователю-руководителю страницу обзора выполненных заданий
rejectReview	POST «/panel/rejectreview»	По полученному ID задания отклоняет выполнение задания
acceptReview	POST «/panel/acceptreview»	По полученному ID задания принимает выполнение задания
createUserPanel	GET «/panel/createUser»	Отправляет пользователю-администратору страницу создания пользователей
deleteUserPanel	GET «/panel/deleteUser»	Отправляет пользователю-администратору страницу удаления пользователей
createUser	POST «/panel/createUser»	По полученным данным с формы создания пользователей, создает нового пользователя в базе данных
deleteUser	POST «/panel/deleteUser»	По полученным данным с панели удаления пользователей, удаляет пользователя из базы данных
createTask	POST «/panel/createTask»	По полученным данным с формы создания задания, добавляет новое задание в базу данных
getTask	GET «/panel/task:ID»	Отправляет пользователю-подчиненному страницу с выбранным заданием
changeTaskStatus	POST «/panel/changeStatus»	По полученному ID задания изменяет статус выполнения в базе данных и отправляет в список выполненных заданий
createComment	POST «/panel/createcomment»	По полученным данным с блока комментариев, добавляет новый комментарий в базу данных

Для некоторых методов в сервисе отображения необходимо создать новые методы в контроллере базы данных. Новые методы приведены в таблице 6.

Таблица 6 – Описание новых методов контроллера базы данных

Название метода	**Описание**
checkTask	Аргументом принимается логин пользователя, возвращается список задач для указанного пользователя
getCompleteTasks	Аргументом принимается ID пользователя, возвращается список выполненных задач для указанного пользователя
getAllComments	Возвращает список всех комментариев
rejectionTask	Аргументов принимается ID задачи, изменяет значение столбца «complete» на 0
acceptionTask	Аргументом принимается ID задачи, перемещает задачу из таблицы «Tasks» в «TaskArchive»
getAllUsers	Возвращает список всех пользователей из таблицы «Users»
deleteUser	Аргументом принимается ID, логин, удаляет из таблицы «Users» запись с указанными параметрами
createTask	Аргументом принимается роль, имя, фамилия, логин, пароль, добавляет пользователя в таблицу «Users» с указанными параметрами
getTask	Аргументом передаётся ID задачи, возвращает запись с указанным ID
getComments	Аргументом передаётся ID задачи, возвращается список комментариев этого задания
changeTaskStatus	Аргументом передаётся ID задачи, изменяет значение столбца «complete» на 1
createComments	Аргументом передаётся ID задачи, ID пользователя, текст комментария, добавляет в таблицу «Comments» новый комментарий с указанными параметрами

Добавляем новые методы в интерфейс контроллера базы данных. Листинг кода сервиса отображения представлен в приложении А, листинг кода контроллера базы данных представлен в приложении Б.

3.4 Тестирование веб-сервиса

Тестирование является заключительным этапом разработки, благодаря тестированию оценивается эффективность использования выделенных ресурсов, скорость отклика сервера и удобство интерфейса.

Для проверки эффективности и скорости будем использовать, встроенный в браузер, инструмент Lighthouse. Для этого запустим веб-сервис, откроем инструменты разработчика и перейдём во вкладку «Lighthouse». Выберем режим «Анализ временного диапазона», а устройство «Десктопная версия». Нажимаем кнопку «Запустить анализ временного диапазона» и работаем с веб-сервисом некоторое время.

Результат тестирования приведен на рисунке 56.

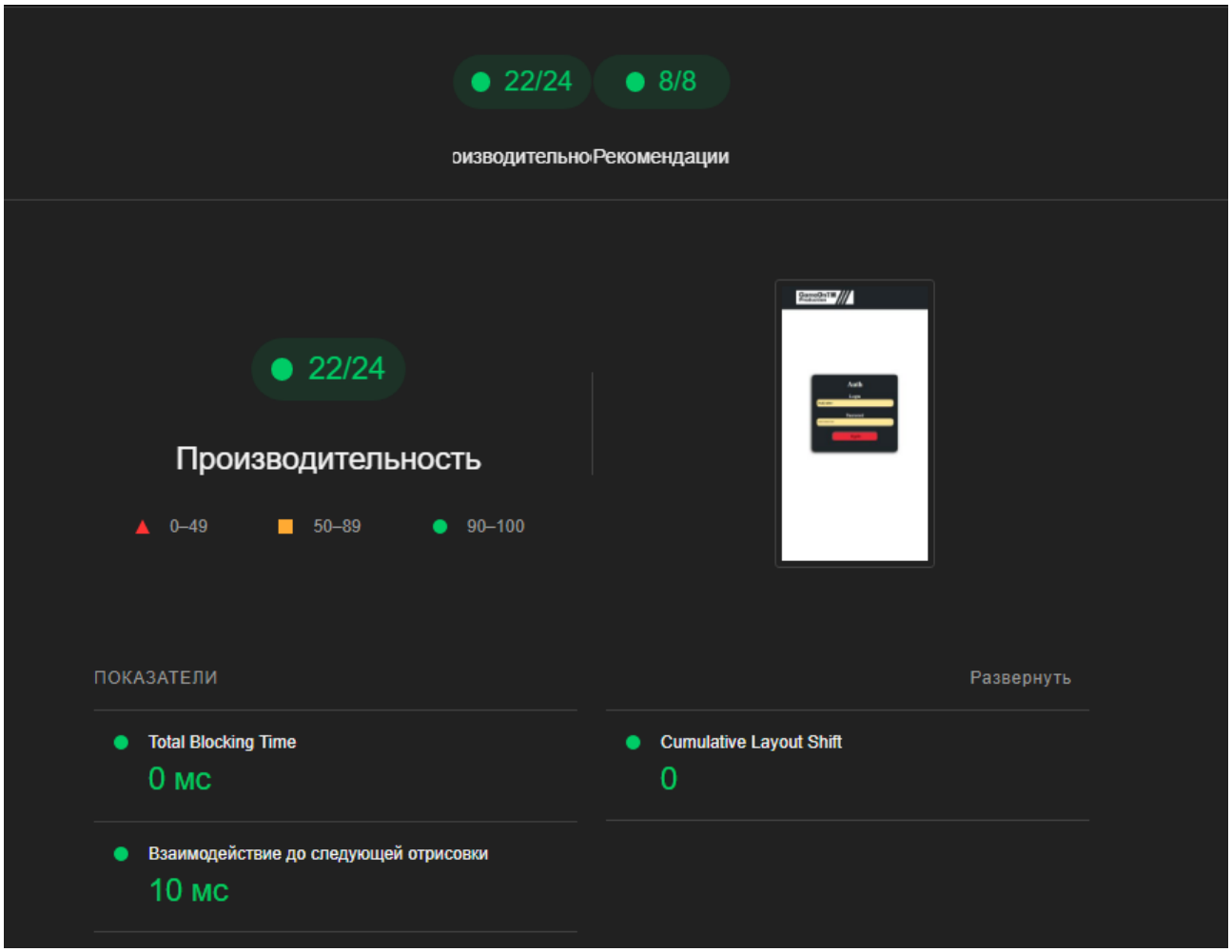


Рисунок 56 – Тестирование Lighthouse

Веб-сервис оценен на 90-100 баллов что является отличным показателем.

Для оценки удобства и понятности интерфейса проведём юзабилити тестирование.

Суть данного теста заключается в разработке кейс-заданий для разных пользователей, выполнении этих кейсов пользователями и сборе обратной связи от пользователей.

Среди 7 респондентов сформировались следующие оценки удобства и понятности интерфейса:

– Удобство – 8,4 из 10;

– Понятность – 8.8 из 10.

Основными причинами непонятности интерфейса является английский язык, а также монотонность кнопок навигации.

Во время проведения тестирования была выявлена некритичная ошибка, кнопка выхода с аккаунта срабатывала только при нажатии на центр.

Таким образом, тестирование помогло определить показатели разработанной системы, удобство использования, а также выявить некоторые ошибки и недоработки.

Подводя итоги, используя современные инструменты разработки, удалось реализовать веб-сервис корпоративного тайм-менеджмента, который является безопасный с точки зрения хранения и обработки информации, типобезопасным с точки зрения разработки серверной части веб-сервиса, а также удобным и понятным с точки зрения разработки дизайна клиентской части.

ЗАКЛЮЧЕНИЕ

В заключение работы отметим следующее. Теоретические основы корпоративного тайм-менеджмента как комплексной системы управления временем IT-организации раскрыты. Разработан и протестирован веб-сервис корпоративного тайм-менеджмента управленческой деятельности руководителя IT-организации ООО «ГеймОн Продакшн».

В ходе выполнения выпускной квалификационной работы решены все все задачи:

1. Проведен анализ учебной и научно-технической литературы по теме исследования, определена сущность корпоративного тайм-менеджмента и рассмотрены критерии, принципы и методы корпоративного тайм-менеджмента.

Наиболее распространенными методами корпоративного тайм-менеджмента являются «Пирамида Франклина», «Матрица Кови», «Естественный метод планирования».

При разработке веб-сервиса корпоративного тайм-менеджмента как комплексной системой управления временем в IT-организации, использовали матрицу Кови, концепция которой позволяет добавлять сотрудникам задания разных приоритетов руководителем отдела.

2. Теоретически обоснован выбор методологии и архитектуры проекта. В качестве методологии выбрана инкрементная модель, которая обеспечивает возможность межэтапных изменений требований.

Самой распространенной архитектурой для веб-сервисов является «клиент-серверная» архитектура, которая позволяет полностью перенести вычисления с клиентской части на серверную, тем самым разгружая рабочие компьютеры пользователей.

Для разработки веб-сервиса выбран оптимальный набор инструментов, в который вошли: язык разметки HTML, каскадные таблицы CSS, язык программирования JavaScript, платформа NodeJS, язык программирования

TypeScript, база данных MySQL, а также сторонние библиотеки для упрощения разработки.

3. Разработана клиентская и серверная части веб-сервиса корпоративного тайм-менеджмента для управленческой деятельности руководителя IT-органгизации ООО «ГеймОн Продакшн».

Тестирование является заключительным этапом разработки, благодаря тестированию оценена эффективность использования выделенных ресурсов, скорость отклика сервера и удобство интерфейса. Для проверки эффективности и скорости использовали, встроенный в браузер, инструмент Lighthouse.

Таким образом, поставленные задачи решены, цель достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Аллен, Д. Как привести дела в порядок. Искусство продуктивности без стресса / Д. Аллен. – Москва: «Манн, Иванов и Фербер» – 2016. – 407 с.

2. Аниче, М. Эффективное тестирование программного обеспечения / М. Аниче. – Москва: ДМК Пресс, 2023. – 370 с. – ISBN 978-5-9706-0997-2.

3. Артамонов, Ю. С. Разработка распределенных приложений сбора и анализа данных на базе микросервисной архитектуры/ Ю. С. Артамонов, С. В. Востокин // Известия Самарского научного центра Российской академии наук. – 2016. – №4. – с. 2.

4. Архангельский, Г. А. Время. Большая книга тайм-менеджмента / Г. А. Архангельский. – Москва: АСТ, 2019. – 416 с. – ISBN 978-5-17-117597-9.

5. Архангельский, Г. А. Тайм-менеджмент: полный курс / Г. А. Архангельский. – Москва: «Альпина Диджитал», 2008. – 370 с. – ISBN 978-5-9614-2813-1.

6. Берг, Д. Б. Модели жизненного цикла: учебное пособие / Д. Б. Берг, Е. А. Ульянова, П. В. Добряк. – Оренбург: Издательство Уральского университета, 2014. – 74 с. – ISBN 978-5-7996-1311-2.

7. Борейшо, М. А. Тайм-менеджмент / М. А. Борейшо // Экономика и социум. – 2021. – с. 1.

8. Браун, И. Веб-разработка с применением Node и Express / И. Браун. – Санкт-Петербург: Питер, 2021. – 334 с.

9. Браун, И. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript / И. Браун. – СПб.: Питер, 2017. – 336 с. –ISBN 978-5-496-02156-2.

10. Бронникова, Е. М. Инструменты личного и корпоративного ТМ в деятельности сотрудников в бизнес среде / Е. М. Бронникова // Бизнес и дизайн ревью. – 2016. – Т. 1, №4. – с. 1–2.

11. Веллинг, Л. Разработка веб-приложений с помощью PHP и MySQL / Л. Веллинг, Л. Томсон. – Вильямс: БХВ, 2010. – 847 с. – ISBN 5845915740

12. Зайцева, Н. А. Научно-практические аспекты применения тайм-менеджмента для повышения профессиональной конкурентоспособности выпускников вузов / Н. А. Зайцева // Российские регионы: взгляд в будущее. – 2016. – Т. 3, № 3. – с. 33–49.

13. Исаев, Г. Н. Проектирование информационных систем. Учебное пособие / Г. Н. Исаев. – Москва: Омега-Л, 2015. – 424 с. – ISBN 978-5-370-02508-2.

14. Кантелон, М. Node.js в действии / М. Кантелон, М. Хартер, Т. Головайчук, Н. Райлих. – СПб.: Питер, 2014. – 548 с. – ISBN978-5-496-01079-5.

15. Кент, Б. Экстремальное программирование. Разработка через тестирование / Б. Кент. – Санкт-Петербург: Питер, 2022. – 224 с. – ISBN 978-5-4461-1439-9.

16. Кириченко, А. В. Web на практике. CSS, HTML, JavaScript, MySQL, PHP для fullstack-разработчиков / А. В. Кириченко, А. П. Никольский, Е. В. Дубовик. – Санкт-Петербург: Питер, 2021. – 432 с. – ISBN 978-5-94387-271-6.

17. Комелягина, С. Е. Тайм-менеджмент / С. Е. Комелягина // Электронная наука. – 2021. – с. 2.

18. Корабейников, И. Н. Тайм-менеджмент: учебное пособие / И. Н. Корабейников, Н. Е. Рябикова; М-во науки и высш. образования Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. Образования «Оренбург. гос. ун-т». – Оренбург: ОГУ, 2020. – 121 с. – ISBN 978-5-7410-2483-6.

19. Корпоративный тайм-менеджмент. Как внедрить в компании единые правила управления временем // HR-Portal: официальный сайт. – 2004. – URL: https://hr-portal.ru/article/korporativnyy-taym-menedzhment-kak-vnedrit-v-kompanii-edinye-pravila-upravleniya-vremenem (дата обращения 15.03.2023)

20. Куперт, Н. Как создать сайт. Комикс-путеводитель по HTML, CSS и WordPress Подробнее / Н. Куперт. – Москва: Манн, Иванов и Фербер, 2019. – 256 с. – ISBN 978-5-00100-914-6.

21. Куташевский, С. В. Интернет-технологии для реализации корпоративного тайм-менеджмента в управлении IT-организации / С. В. Куташевский. – 2022.

22. Куташевский, С. В. Корпоративный тайм менеджмент в управлении IT-организации / С. В. Куташевский. – 2022.

23. Куташевский, С. В. Проектирование веб-сервиса для корпоративного тайм-менеджмента IT-организаций на основе интернет-технологий / С. В. Куташевский. – 2022.

24. Куташевский, С. В. Роль интернет-технологий в создании корпоративного тайм-менеджмента в IT-организации / С. В. Куташевский. – 2022.

25. Ломов, А. Ю. HTML, CSS, скрипты: практика создания сайтов: самоучитель / А. Ю. Ломов. – СПб.: БХВ-Петербург, 2007. – 399 с. – ISBN 5-94157-698-6.

26. Мрочковский, Н. С. Экстремальный тайм-менеджмент / Н. С. Мрочковский, А. Толкачев. – Москва: Альпина Паблишер, 2015. – 214 с. – ISBN 978-5-9614-1938-2.

27. Назаров, С. В. Архитектура и проектирование программных систем / С. В. Назаров. – Москва: ИНФРА-М, 2013. – 413 с. – ISBN 978-5-16-011753-9.

28. Никитин, И. В. Сравнение подходов монолитной архитектуры и микросервисной архитектуры при реализации серверной части веб-приложения / И. В. Никитин, Т. Ю. Гриценко // Дневник науки. – 2020.– №3. – с. 3.

29. Остроух, А. В. Проектирование информационных систем / Н. Е. Суркова, А. В. Остроух. – Санкт-Петербург: Издательство «Лань», 2021. – 164 с. – ISBN 978-5-8114-8377-8.

30. Пауэрс, Ш. Изучаем Node. Переходим на сторону сервера. / Ш. Пауэрс. – Санкт-Петербург: Питер, 2017. – 304 с. – ISBN 978-5-496-02941-4.

31. Прохоренко, Н. А. JavaScript и Node.js для веб-разработчиков / Н. А. Прохоренко, В. А. Дронов. – Санкт-Петербург: БХВ-Петербург, 2022. – 768 с. – ISBN 978-5-9775-6847-0.

32. Прохоренок, Н. А. Bootstrap и CSS-препроцессор Sass. Самое необходимое / Н. А. Прохоренок. – Санкт-Петербург: БХВ, 2021. – 496 с. – ISBN 978-5-9775-6769-5.

33. Роббинс, Дж. HTML5, CSS и JavaScript. Исчерпывающее руководство / Дж. Роббинс; [перевод с английского М. А. Райтман] – Москва: Эксмо, 2014. – 528 с. – ISBN 978-5-699-67603-3.

34. Сидельников, Г. Наглядный CSS / Г. Сидельников. – Санкт-Петербург: Питер, 2022. – 224 с. – ISBN 978-5-4461-1618-8.

35. Симан, М. Внедрение зависимостей на платформе .NET / М. Симан, С. Дерсен. – СПб: Питер, 2021. – 608 с. – ISBN 978-5-4461-1166-4.

36. Симченко, Н. Н. Автоматизация тайм-менеджмента, как способ повышения эффективности деятельности сотрудника / Н. Н. Симченко, А. В. Бочкарев // Научные междисциплинарные исследования. – 2020. – с. 3–5.

37. Тамре, Л. Введение в тестирование программного обеспечения. Руководство / Л. Тамре. – Киев: Диалектика/Вильямс, 2018. – 368 с. – ISBN 0-201-71974-6.

38. Файн, Я. TypeScript быстро / Я. Файн, А. Моисеев. – СПб.: Питер, 2021. – 540 с.–ISBN 978-5-4461-1725-3.

39. Фрейн, Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Б. Фрейн. – Санкт-Петербург: Питер, 2018. – 304 с. – ISBN 978-5-496-00185-4.

40. Хэррон, Д. Node.js Разработка серверных веб-приложений на JavaScript / Д. Хэррон. – Москва: ДМК, 2014. – 144 с.–ISBN 978-5-94074-809-0.

41. Чаллавала, Ш. MySQL8 для больших данных / Ш. Чавалла, Д. Лакхатария, Ч. Мехта. – Москва: «ДМК Пресс», 2017. – 228 с. – ISBN 978-5-97060-653-7.

42. Черный, Б. Профессиональный TypeScript. Разработка масштабируемых JavaScript-приложений / Б. Черный. – СПб: Питер, 2022. – 352 с. – ISBN 978-5-4461-1651-5.

43. Чистов, Д. В. Проектирование информационных систем: учебник и практикум для СПО / Д. В. Чистов – Москва: Юрайт, 2019. – 258 с. – ISBN 978-5-534-03173-7.

44. Янг, А. Node.js в действии / А. Янг, Б. Мек, М. Кантелон. – 2-е издание. – Санкт-Петербург: Питер, 2018. – 432 с. – ISBN 978-5-496-03212-4.

ПРИЛОЖЕНИЕ А

Листинг сервиса отображения

```
import {inject, injectable} from "inversify"
import {BaseController} from "../common/base.controller"
import {TYPES} from "../types"
import {ILogger} from "../logger/logger.interface"
import {IDBController} from "../DataBase/DB.controller.interface"
import {NextFunction, Request, Response} from "express"
import 'reflect-metadata'
import {IRoleController} from "./role.controller.interface";
import jwt_decode from "jwt-decode"
import {JWTAuthMiddleware} from "../Middlewares/AuthMiddleware";

@injectable()
export class RoleController extends BaseController implements IRoleController {

    constructor(@inject(TYPES.ILogger) logger: ILogger,
                @inject(TYPES.IDBController) private DBController: IDBController) {
        super(logger)
        this.bindRoutes([
            {path: '/panel', method: 'get', func: this.checkRole, middlewares: [new
JWTAuthMiddleware(logger)]},
            {path: '/panel/createUser', method: 'get', func: this.createUserPanel,
middlewares: [new JWTAuthMiddleware(logger)]},
            {path: '/panel/deleteUser', method: 'get', func: this.deleteUserPanel,
middlewares: [new JWTAuthMiddleware(logger)]},
            {path: '/panel/createUser', method: 'post', func: this.createUser},
            {path: '/panel/deleteUser', method: 'post', func: this.deleteUser},
            {path: '/panel/createcomment', method: 'post', func: this.createComment},
            {path: '/panel/createTask', method: 'get', func: this.createTaskPanel,
middlewares: [new JWTAuthMiddleware(logger)]},
            {path: '/panel/createTask', method: 'post', func: this.createTask},
            {path: '/panel/reviewTask', method: 'get', func: this.taskReviewPanel,
middlewares: [new JWTAuthMiddleware(logger)]},
            {path: '/panel/rejectreview', method: 'post', func: this.rejectReview},
            {path: '/panel/acceptreview', method: 'post', func: this.acceptReview},
            {path: '/panel/task:ID', method: 'get', func: this.getTask, middlewares: [new
JWTAuthMiddleware(logger)]},
            {path: '/panel/changeStatus', method: 'post', func: this.changeTaskStatus}
        ])
    }
```

```
public async checkRole(req: Request, res: Response, next: NextFunction) {
    const data:any = jwt_decode(res.locals.token)
    if (data.role === 'admin'){
        res.render('achoosepanel.ejs', {
            name: data.fname,
            role: data.role
        })
    } else if (data.role === 'TeamLeader') {
        res.render('teamleadpanel.ejs', {
            id: data.id,
            name: data.fname,
            role: data.role
        })
    } else if (data.role === 'ProgEngineer') {
        let tasks: unknown = []
        tasks = await this.DBController.checkTask(data.login)
        res.render('todopanel.ejs', {
            name: data.fname,
            role: data.role,
            red: 'red',
            yellow: 'yellow',
            green: 'green',
            task: tasks
        })
    } else {
        res.json('Unknown role')
    }
}

public async createTaskPanel(req: Request, res: Response, next: NextFunction) {
    const data:any = jwt_decode(res.locals.token)
    res.render('teamleadtaskcreator.ejs', {
        id: data.id,
        name: data.fname,
        role: data.role
    })
}

public async taskReviewPanel(req: Request, res: Response, next: NextFunction) {
    const data:any = jwt_decode(res.locals.token)
    let tasks = await this.DBController.getCompleteTasks(data.id)
    let comments = await this.DBController.getAllComments()
    res.render('teamleadreviewpanel.ejs', {
        id: data.id,
        name:data.fname,
```

```
            role:data.role,
            task:tasks,
            high:'high',
            medium:'medium',
            low:'low',
            comments:comments
        })
    }

    public async rejectReview(req: Request, res: Response, next: NextFunction) {
        let status = await this.DBController.rejectionTask(req.body.ID)
        res.json(status)
    }

    public async acceptReview(req: Request, res: Response, next: NextFunction) {
        let status = await this.DBController.acceptionTask(req.body.ID)
        res.json(status)
    }

    public async createUserPanel(req: Request, res: Response, next: NextFunction) {
        const data:any = jwt_decode(res.locals.token)
        res.render('apanelcreate.ejs', {
            name:data.fname,
            role:data.role
        })
    }

    public async deleteUserPanel(req: Request, res: Response, next: NextFunction) {
        const data:any = jwt_decode(res.locals.token)
        const users = await this.DBController.getAllUsers()
        res.render('apaneldelete.ejs', {
            name:data.fname,
            role:data.role,
            users: users
        })
    }

    public async createUser(req: Request, res: Response, next: NextFunction) {
        let status = await this.DBController.createUser(req.body.Role, req.body.FName,
req.body.LName, req.body.Login, req.body.Password)
        if (status === 'Created'){
            res.status(201).json('Created')
        } else {res.json(status)}
    }
```

```
  public async deleteUser(req: Request, res: Response, next: NextFunction) {
    let status = await this.DBController.deleteUser(req.body.UserId,
req.body.Login)
    if (status === 'Deleted'){
      res.status(201).json('Deleted')
    } else {res.json(status)}
  }

  public async createTask(req: Request, res: Response, next: NextFunction) {
    let status = await this.DBController.createTask(req.body.ID,req.body.Target,
req.body.Title, req.body.Context, req.body.Deadline, req.body.Priority)
    if (status === 'Created'){
      res.status(201).json('Created')
    } else {res.json(status)}
  }

  public async getTask(req: Request, res: Response, next: NextFunction) {
    let task = await this.DBController.getTask(req.params.ID)
    let comments = await this.DBController.getComments(req.params.ID)
    const data:any = jwt_decode(res.locals.token)
    // @ts-ignore
    if (!data.hasOwnProperty('error') && data !== 'Not searched') {
      res.render('taskpanel.ejs', {
        UserID:data.id,
        name:data.fname,
        role:data.role,
        // @ts-ignore
        TaskID:task[0].id,
        // @ts-ignore
        title: task[0].title,
        // @ts-ignore
        context: task[0].context,
        // @ts-ignore
        deadline: task[0].deadline,
        // @ts-ignore
        priority: task[0].priority,
        comments: comments,
        Low: 'Low',
        Medium: 'Medium',
        High: 'High'
      })
    }
    else{res.json(data)}
  }
```

```
    public async changeTaskStatus(req: Request, res: Response, next: NextFunction) {
        let status = await this.DBController.changeStatus(req.body.id)
        res.json(status)
    }

    public async createComment(req: Request, res: Response, next: NextFunction){
        let status = await this.DBController.createComment(req.body.TaskID,
req.body.UserID, req.body.comment)
        res.json(status)
    }
}
```

ПРИЛОЖЕНИЕ Б

Листинг контроллера базы данных

```
import * as mysql from 'mysql'
import {Connection, MysqlError} from "mysql"
import {inject, injectable} from "inversify";
import {ILogger} from "../logger/logger.interface";
import {TYPES} from "../types";
import {IDBController} from "./DB.controller.interface";
import 'reflect-metadata'
import * as argon from 'argon2'
import * as jwt from 'jsonwebtoken'
import {Secret} from "jsonwebtoken";


@injectable()
export class DBController implements IDBController{
    config: {}
    connection: Connection
    salt: Secret

    constructor(@inject(TYPES.ILogger) private logger: ILogger) {
        this.config = {
            host: process.env.DB_HOST,
            user: process.env.DB_USER,
            password: process.env.DB_PWD,
            database: process.env.DB_DATABASE,
            connectionLimit: 10,
            multipleStatements: true
        }
        // @ts-ignore
        this.salt = process.env.SALT
    }


    public async connect() {
        this.connection = mysql.createConnection(this.config)
        await this.connection.connect((err:MysqlError) => {
            if (err){
                this.logger.error(`DataBase is offline: ${err.message}`)
            } else {
                this.logger.log('DataBase is online')
            }
```

```
        })
        this.connection.end()
    }

    public async checkRole(role:string) {
        this.connection = mysql.createConnection(this.config)

        return new Promise((resolve) => {
            let status:number
            return this.connection.query(`select * from users where role = '${role}'`,
(error: MysqlError, results) => {
                this.connection.end()
                if (results.length > 0) {
                    status = 1
                } else if (results.length === 0) {
                    status = 0
                } else {
                    resolve(error)
                }
                resolve(status)
            })

        })
    }

    public async checkUser(login: string, password:string) {
        this.connection = mysql.createConnection(this.config)
        return new Promise((resolve) => {
            this.connection.query(`select * from users where login = '${login}'`,
async(error: MysqlError, results) => {
                this.connection.end()
                if (results.length != 0){
                    if (await argon.verify(results[0].password, password)){
                        resolve(jwt.sign({id:results[0].id,
role:results[0].role,
login:results[0].login, fname:results[0].fname, lname:results[0].lname}, this.salt,
{expiresIn:"8h"}))
                    }
                    else {
                        resolve(0)
                    }
                }
                if (results.length === 0) {
                    resolve(0)
                }
                if (error) {
```

```
                resolve(-1)
            }
        })
    })
}

public async createUser(role: string, fname: string, lname: string, login: string, password:string) {
    this.connection = mysql.createConnection(this.config)
    const hashPassword = await argon.hash(password)
    return new Promise((resolve) => {
        let status:string
        return this.connection.query(`insert into users (role, fname, lname, login, password) values ('${role}', '${fname}', '${lname}', '${login}', '${hashPassword}')`, (error: MysqlError, results) => {
            this.connection.end()
            if (results) {
                resolve('Created')
            }
            else {
                resolve(error)
            }
            resolve(status)
        })
    })
}

public async getAllUsers() {
    this.connection = mysql.createConnection(this.config)
    return new Promise((resolve) => {
        return this.connection.query(`select * from users where role <> 'admin'`, (error: MysqlError, results) => {
            this.connection.end()
            if (results.length != 0) {
                resolve(results)
            }
            else {
                resolve('No users')
            }
            resolve(error)
        })
    })
}

public async deleteUser(UserID: number, Login:string) {
```

```
        this.connection = mysql.createConnection(this.config)
        return new Promise((resolve) => {
            let status:string
            return this.connection.query(`delete from users where id = ${UserID}; delete from tasks where target = '${Login}'`, (error: MysqlError, results) => {
                this.connection.end()
                if (results) {
                    resolve('Deleted')
                }
                else {
                    resolve(error)
                }
                resolve(status)
            })
        })
    }

    public async createTask(LeadID:number, target: string, title: string, context: string, deadline: string, priority:number) {
        this.connection = mysql.createConnection(this.config)

        return new Promise((resolve) => {
            return this.connection.query(`insert into tasks (LeadID, target, title, context, deadline, priority, complete, done) values ('${LeadID}','${target}','${title}','${context}',str_to_date('${deadline}','%Y-%m-%d'),'${priority}', 0, '0') ;`, (error: MysqlError, results) => {
                this.connection.end()
                if (!error) {
                    if (results) {
                        resolve('Created')
                    }
                    else {
                        resolve('Not created')
                    }
                } else {
                    resolve(error)
                }
            })
        })
    }

    public async checkTask(login:string) {
        this.connection = mysql.createConnection(this.config)
        return new Promise((resolve) => {
```

```
        return this.connection.query(`select y.id, y.deadline, y.title, y.context, y.priority from users x inner join tasks y on  y.target = x.login and y.target = '${login}' and y.complete = 0 order by deadline;`, (error: MysqlError, results) => {
            this.connection.end()
            if (!error) {
                if (results.length > 0) {
                    resolve(results)
                }
                else {
                    resolve('Not searched')
                }
            } else {
                resolve(error)
            }
        })
    })
  }

  public async getTask(ID:string) {
    this.connection = mysql.createConnection(this.config)
    return new Promise((resolve) => {
        this.connection.query(`select y.id, y.deadline, y.title, y.context, y.priority, x.fname, x.role from users x inner join tasks y on  y.target = x.login and y.id = '${ID}';`, (error: MysqlError, results) => {
            this.connection.end()
            if (!error) {
                if (results.length > 0) {
                    resolve(results)
                }
                else {
                    resolve('Not searched')
                }
            } else {
                resolve({error: error})
            }
        })
    })
  }

  public async changeStatus(ID:string) {
    this.connection = mysql.createConnection(this.config)
    return new Promise((resolve) => {
        this.connection.query(`update tasks set complete = 1 where id = ${ID};`, (error: MysqlError, results) => {
            this.connection.end()
```

```
            if (!error) {
                if (results) {
                    resolve('Updated')
                }
                else {
                    resolve('Not Updated')
                }
            } else {
                resolve(error)
            }
        })
    })
}

public async getCompleteTasks(leadID:string) {
    this.connection = mysql.createConnection(this.config)
    return new Promise((resolve) => {
        this.connection.query(`select y.id, y.deadline, y.title, y.context, y.priority from
users x inner join tasks y on  y.LeadID = x.id and x.id = ${leadID} and y.complete =
1 order by deadline;`, (error: MysqlError, results) => {
            this.connection.end()
            if (!error) {
                if (results.length > 0) {
                    resolve(results)
                }
                else {
                    resolve('Not searched')
                }
            } else {
                resolve({error: error})
            }
        })
    })
}

public async rejectionTask(taskID:string) {
    this.connection = mysql.createConnection(this.config)
    return new Promise((resolve) => {
        this.connection.query(`update tasks set complete = 0 where id = ${taskID};`,
(error: MysqlError, results) => {
            this.connection.end()
            if (!error) {
                if (results) {
                    resolve('Updated')
                }
```

```
                else {
                    resolve('Not Updated')
                }
            } else {
                resolve(error)
            }
        })
    })
}

public async acceptionTask(taskID:string) {
    this.connection = mysql.createConnection(this.config)
    return new Promise((resolve) => {
        this.connection.query(`insert into taskArchive (ID, LeadID, target, title, context, priority) select id, LeadID, target, title, context, priority from tasks where id = '${taskID}';update taskArchive set completeData = curdate() where ID = '${taskID}'; delete from tasks where id = ${taskID}`, (error: MysqlError, results) => {
            this.connection.end()
            if (!error) {
                if (results) {
                    resolve('Accepted')
                }
                else {
                    resolve('Not Accepted')
                }
            } else {
                resolve(error)
            }
        })
    })
}

public createComment(TaskID:number, UserID:string, comment:string) {
    this.connection = mysql.createConnection(this.config)
    return new Promise((resolve) => {
        this.connection.query(`insert into comments (TaskID, UserID, comment) values ('${TaskID}', '${UserID}', '${comment}');`, (error: MysqlError, results) => {
            this.connection.end()
            if (!error) {
                if (results) {
                    resolve('Created')
                }
                else {
                    resolve('Not Created')
```

```
                }
            } else {
                resolve(error)
            }
        })
    })
}

public getComments(TaskID:string) {
    this.connection = mysql.createConnection(this.config)
    return new Promise((resolve) => { //insert into taskArchive (id, LeadID, target, title, context, priority) select id, LeadID, target, title, context, priority from tasks where id = 36; update taskArchive set completeDate = str_to_date('2023-06-04','%Y-%m-%d') where id = 36;
        this.connection.query(`select y.TaskID, y.UserID, x.login as User, y.comment from users x inner join comments y on x.id = y.UserID and y.TaskID = ${TaskID};` , (error: MysqlError, results) => {
            this.connection.end()
            if (!error) {
                resolve(results)
            } else {
                resolve(error)
            }
        })
    })
}

public getAllComments() {
    this.connection = mysql.createConnection(this.config)
    return new Promise((resolve) => {
this.connection.query(`select y.TaskID, y.UserID, x.login as User, y.comment from users x inner join comments y on x.id = y.UserID;` , (error: MysqlError, results) => {
            this.connection.end()
            if (!error) {
                resolve(results)
            } else {
                resolve(error)
            }
        })
    })
}
}
```