



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Разработка клиент-серверных приложений
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Проектирование и разработка клиент-серверного фуллстек-приложения для автоматизированного гардероба

Студент: Сидоров Станислав Дмитриевич

Группа: ИКБО-20-21

Работа представлена к защите _____ (дата) _____ /Сидоров С.Д. /
(подпись и ф.и.о. студента)

Руководитель: Коваленко Михаил Андреевич, ст. преп.

Работа допущена к защите _____ (дата) _____ /Коваленко М.А. /
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: _____

_____/ _____ /
_____/ _____ /

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2024 г.



МИНОБРАЗОВАНИЯ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ

на выполнение курсовой работы

по дисциплине: Разработка клиент-серверных приложений

по профилю: Разработка программных продуктов и проектирование информационных систем

направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Сидоров Станислав Дмитриевич

Группа: ИКБО-20-21

Срок представления к защите: 20.05.2024

Руководитель: ст.преп. Коваленко Михаил Андреевич

Тема: «Проектирование и разработка клиент-серверного фуллстек-приложения для автоматизированного гардероба»

Исходные данные: HTML5, CSS3, Java, Spring, PostgreSQL, Maven, Git, YandexCloud.

Перечень вопросов, подлежащих разработке, и обязательного графического материала: 1. Провести анализ предметной области для выбранной темы, обосновать выбор клиент-серверной архитектуры и дать её детальное описание с помощью UML для разрабатываемого приложения; 2. Выбрать программный стек для реализации фуллстек разработки, ориентируясь на мировой опыт и стандарты в данной области; 3. Реализовать фронтенд и бекенд части клиент-серверного приложения, обеспечивать авторизацию и аутентификацию пользователя; 4. Разместить готовое клиент-серверное приложение в репозитории GitHub с указанием ссылки в тексте отчёта; 5. Развернуть клиент-серверное приложение в облаке. 6. Провести пользовательское тестирование функционирования минимально жизнеспособного продукта. 7. Разработать презентацию с графическими материалами.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: Болбаков Р. Г. /, « 26 » 02 2024 г.

Задание на КР выдал: Коваленко М.А. /, « 26 » 02 2024 г.

Задание на КР получил: Сидоров С.Д. /, « 26 » 02 2024 г.

Руководитель курсовой работы: ст.прер. Коваленко Михаил Андреевич

Сидоров С.Д., Курсовая работа направления подготовки «Программная инженерия» на тему «Разработка клиент-серверного фуллстек-приложения автоматизированного гардероба»: М. 2024 г., МИРЭА – Российский технологический университет (РТУ МИРЭА), Институт информационных технологий (ИИТ), кафедра инструментального и прикладного программного обеспечения (ИиППО) – 49 стр., 35 рис., 15 источн.

Ключевые слова: паттерн MVC, фуллстек-приложение, автоматизированный гардероб, Spring Framework, PostgreSQL, React, Docker, Docker-compose, Java.

Целью работы является проектирование и разработка клиент-серверного фуллстек-приложения для автоматизированного гардероба. Спроектирована информационная система, разработано решение для клиентских и серверных частей, разработана база данных и настроен удаленный сервер.

Sidorov S.D., Course Work for the program "Software Engineering" on the topic "Development of a client-server full-stack application for automotive wardrobe": Moscow, 2024, MIREA – Russian Technological University (RTU MIREA), Institute of Information Technologies (IIT), Department of Instrumental and Applied Software (IiPPO) – 49 pages, 35 figures, 15 sources.

Keywords: MVC pattern, full-stack application, automated wardrobe, Spring Framework, PostgreSQL, React, Docker, Docker-compose, Java.

The purpose of the work is to design and develop a client-server full-stack application for automotive wardrobe. An information system was designed, solutions for both client and server parts were developed, a database was created, and a remote server was configured.

РТУ МИРЭА: 119454, Москва, пр-т Вернадского, д. 78

кафедра инструментального и прикладного программного обеспечения (ИиППО)

Тираж: 1 экз. (на правах рукописи)

Файл: «ПЗ_РКСП_ИКБО_20_21_СидоровСД.pdf», исполнитель Сидоров С. Д.

© Сидоров С. Д.

СОДЕРЖАНИЕ

ГЛОССАРИЙ.....	7
ВВЕДЕНИЕ.....	8
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	10
1.1 Описание предметной области	10
1.2 Анализ ниши веб-приложений автоматизированных гардеробов	10
1.2.1 Авторизация и аутентификация.....	12
1.2.2 Аренда	13
1.2.3 История пользования	13
1.2.4 Пользовательский интерфейс	13
1.2.5 Рекомендация объекта аренды.....	14
1.3 Функциональные требования на основе анализа.....	14
2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ РАЗРАБОТКИ ПРИЛОЖЕНИЯ	15
2.1 Основные используемые технологии.....	15
2.2 Краткое обоснование использования технологий	15
2.2.1 Язык программирования Java.....	15
2.2.2 Spring	15
2.2.3 Spring Boot	16
2.2.4 Spring Web.....	16
2.2.5 Spring Data JPA	16
2.2.6 Spring Security	16
2.2.7 Maven.....	16
2.2.8 Bruno	16
2.2.9 JSON Web Tokens	16

2.2.10 IntelliJ IDEA	16
2.2.11 Git.....	17
2.2.12 GitHub.....	17
2.2.13 GitVerse	17
2.2.14 PostgreSQL	17
2.2.15 HyperText Markup Language (HTML).....	17
2.2.16 Cascading Style Sheets (CSS).....	17
2.2.17 JavaScript (JS).....	17
2.2.18 React.....	18
2.2.19 Docker	18
2.2.20 Docker-compose.....	18
2.2.21 Lombok.....	18
2.2.22 Yandex Cloud.....	18
2.2.23 Spring WebSocket.....	18
3 АРХИТЕКТУРА ВЕБ-ПРИЛОЖЕНИЯ	19
3.1 Архитектура серверной части приложения.....	19
3.1.1 Описание архитектуры	19
3.1.2 Пример реализации архитектуры	20
3.2 Архитектура клиентской части приложения.....	22
3.2.1 Описание архитектуры	22
3.2.2 Пример реализации архитектуры	22
4 РАЗРАБОТКА БАЗЫ ДАННЫХ.....	24
4.1 Определение сущностей.....	24
4.2 Создание сущностей на уровне СУБД.....	24
5 РЕАЛИЗАЦИЯ СЛОЯ СЕРВЕРНОЙ ЛОГИКИ	25

5.1 Структура проекта.....	25
5.2 Конфигурация приложения.....	26
5.3 Конфигурация безопасности проекта.....	26
5.4 Тестирование	27
6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ	33
6.1 Структура проекта.....	33
6.2 Взаимодействие с серверной частью веб-приложения	34
6.3 Вид конечного приложения	37
7 КОНТЕЙНИРИЗАЦИЯ И ДЕПЛОЙ.....	42
ЗАКЛЮЧЕНИЕ	45
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	46

ГЛОССАРИЙ

В настоящем отчете применяют следующие термины:

- | | |
|------------------------|---|
| Серверная часть | – часть приложения, которая занимается обработкой и хранением данных и передачей результата клиенту. |
| Клиентская часть | – часть приложения, которая предоставляет пользователю данные в удобном для взаимодействия формате. |
| Архитектура приложения | – описывает проектирование, набор компоненто системы и объясняет взаимодействие между элементами. |
| База данных | – структурированное хранилище данных. |
| СУБД | – программное обеспечение, предназначенное для администрирования баз данных. |
| Контейнерезация | – упаковка программного кода с библиотеками операционной системы и всеми зависимостями, которые необходимы для выполнения кода. |
| АГР | – автоматизированный гардеробный ряд, обладающий собственной внутренней нумерацией блоков. |

ВВЕДЕНИЕ

В современном мире каждый сталкивался с большими очередями в гардеробах, с необходимостью хранить полученный жетон и с штрафом за его потерю, что создает множество неудобств посетителям общественных мест. Данная проблема возникает из-за неорганизованности посетителей. Для решения данной проблемы возможен вариант переноса системы работы с жетонами и ячейками гардероба в цифровой формат, что избавит от необходимости толкаться в очереди и беспокоиться за свой жетон. Именно поэтому, темой работы было выбрано фуллстек приложение для автоматизированного гардероба.

Цель данной курсовой работы – разработка комплексного приложения, включающего серверную часть на Java с использованием Spring Framework и принципов REST, базу данных PostgreSQL для хранения данных и клиентскую часть на React, HTML и CSS. Это приложение поможет организациям удобно администрировать имеющиеся у них гардеробы, а также улучшит пользовательский опыт взаимодействия с сотрудниками гардероба, убрав необходимость пользования жетонами, и добавит возможность просматривать свою историю посещений. Для обеспечения высокой отказоустойчивости и масштабируемости приложение будет развернуто на облачном сервере Yandex Cloud с использованием Docker.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) провести анализ предметной области разрабатываемого веб-приложения;
- 2) сформировать функциональные требования к приложению;
- 3) обосновать выбор средств ведения разработки;
- 4) разработать архитектуру веб-приложения;
- 5) реализовать слой серверной логики веб-приложения с использованием выбранных технологий и инструментария;
- 6) реализовать слой логики базы данных;
- 7) разработать слой клиентского представления веб-приложения;

- 8) оформить пояснительную записку по курсовой работе;
- 9) подготовить презентацию выполненной курсовой работы.

В ходе выполнения работы были использованы следующие методы: сравнение, анализ, классификация, обобщение, описание и моделирование.

Работа состоит из введения, оглавления, аннотации, глоссария, семи основных разделов, заключения и списка использованных источников.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Описание предметной области

Автоматизированный гардероб – это специальная система содержащая в себе информацию об АГР и ячейках, а также о сотрудниках, обслуживающих ряды, и данных пользователей, использующих ту или иную ячейку. Она автоматически присваивает новому пользователю ячейку, а также помогает выбрать более свободный ряд, что позволяет равномерно распределить нагрузку.

Основная цель автоматизированного гардероба - уменьшить количество взаимодействий пользователя с сотрудником гардероба и избавить от необходимости хранить жетон с номером ячейки, а также добавить организованности в процесс пользования гардеробом и уменьшить время, затраченное на его посещение.

1.2 Анализ ниши веб-приложений автоматизированных гардеробов

На данный момент в сети не представлено большое количество систем, функционал которых соответствует выше описанному. Однако, описанный выше функционал в значительной степени похож на функционал различных приложений для аренды самокатов или автомобилей, так как в данном случае также присутствует объект аренды, различается только способ выбора и сам объект, которым, в автоматизированном гардеробе, является ячейка. Исходя из этого, для установления необходимого функционала создаваемого веб-приложения были проанализированы 2 приложения с тематикой «кикшеринг» и 1 приложение с тематикой «каршеринг» [1, 2, 3]:

<https://whoosh-bike.ru>,

https://go.yandex.ru_ru/lp/rides/scooter,

<https://yandex.ru/drive/>.

На следующих рисунках 1.1 – 1.3 показаны главные страницы анализируемых сайтов.

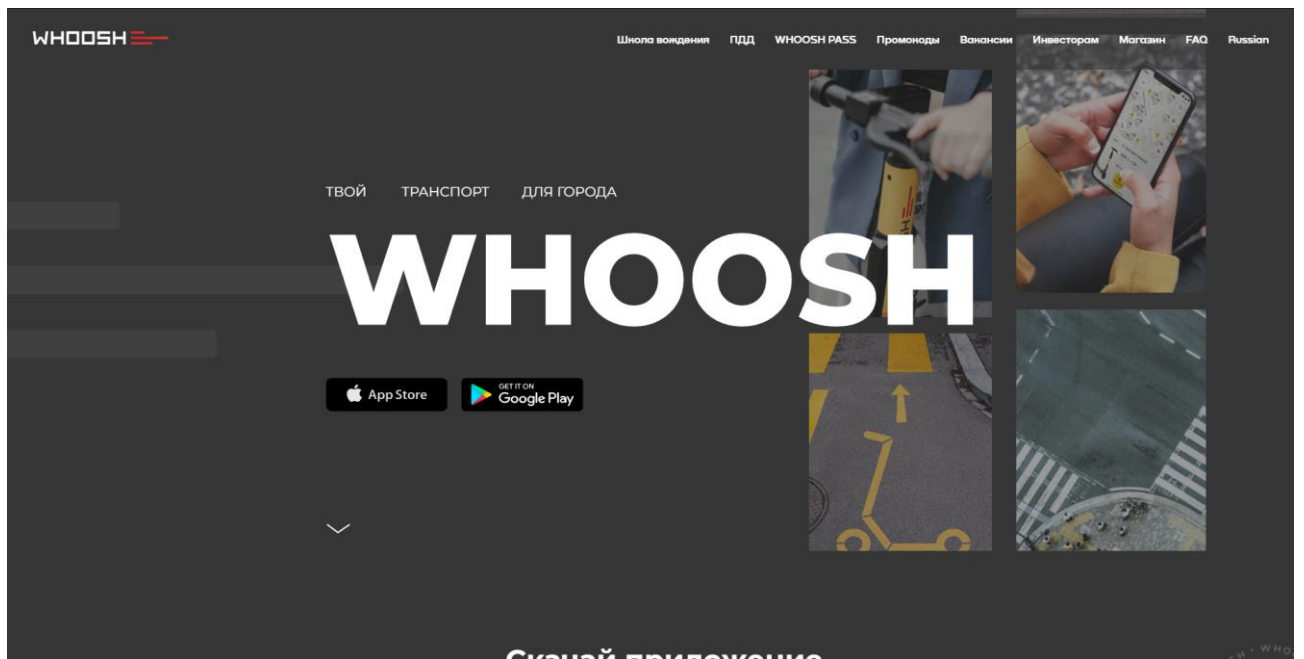


Рисунок 1.1 – Сайт сервиса «Whoosh»

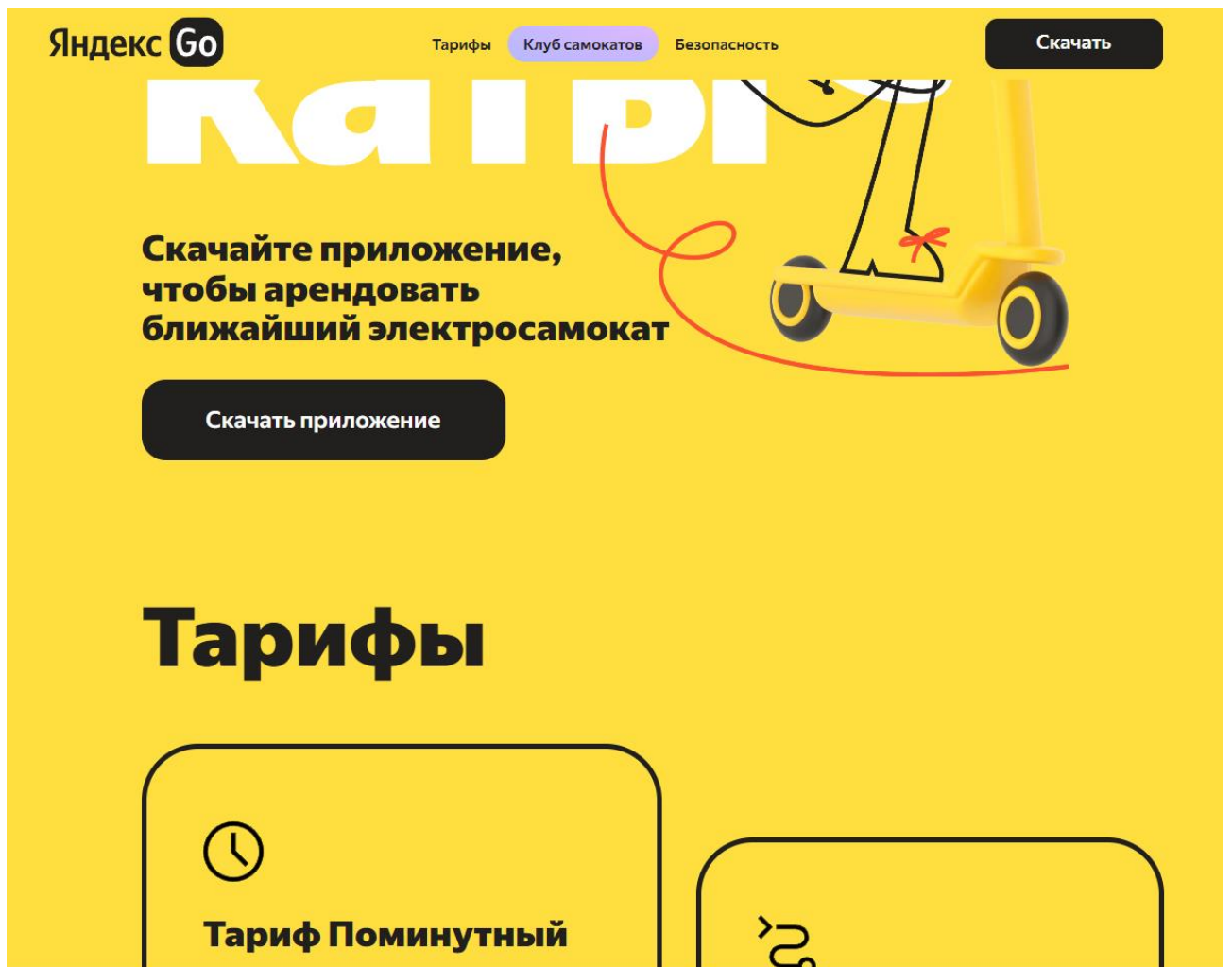


Рисунок 1.2 – Сайт сервиса «Яндекс GO»

СВОБОДА — КОГДА СВОЕЙ МАШИНЫ НЕТ. А КАРШЕРИНГ ЕСТЬ.

ВОТ МАШИНЫ В ВАШЕМ ГОРОДЕ —

Москва

ЛЕГКОВЫЕ / ФУРГОНЫ

Нужны 21 год жизни и 2 года стажа



**GEELY COOLRAY
FLAGSHIP**



CHERY TIGGO 7 PRO



HAVAL JOLION



CHEVROLET NIVA

Рисунок 1.3 – Сайт сервиса «Яндекс Драйв»

1.2.1 Авторизация и аутентификация

Анализ

Во всех 3-х сервисах необходима аутентификация для того, чтобы пользователь мог пользоваться функционалом аренды.

Вывод

В разрабатываемом программном продукте должна поддерживаться аутентификация. Так доступный функционал будет ограничен в зависимости от роли пользователя.

1.2.2 Аренда

Анализ

Сервисы «Whoosh» и «Яндекс Go» позволяют арендовать сразу несколько самокатов, а сервис «Яндекс Драйв» только один автомобиль. Все сервисы позволяют выбрать арендуемый объект.

Вывод

С учетом специфики автоматизированного гардероба, создаваемый программный продукт должен иметь возможность арендовать ячейка в, выбранном, пользователем ряду и организации. Однако количество возможных аренд должно быть ограничено до одной, чтобы избежать оставления вещей на длительное хранение.

1.2.3 История пользования

Анализ

Все 3 сервиса предоставляют возможность просмотреть информацию о предыдущих использованиях сервиса, содержащую данные об объекте и времени аренды.

Вывод

Создаваемое приложение должно поддерживать функционал просмотра данных о посещениях пользователя, содержащую информацию об арендуемой ячейке и времени начала и окончания аренды.

1.2.4 Пользовательский интерфейс

Анализ

Все 3 сервиса обладают современным пользовательским интерфейсом.

Вывод

Интерфейс должен быть современный и достаточно простой для взаимодействия с ним. Он должен включать в себя разделы отображающие основную информацию, а также информацию для навигации по веб-приложению.

1.2.5 Рекомендация объекта аренды

Анализ

Два из трех рассматриваемых сервисов рекомендуют пользователю наиболее удобный для пользователя объект аренды.

Вывод

Создаваемое приложение должно иметь функционал для рекомендации пользователю наиболее подходящего гардеробного ряда, а также автоматическое назначение арендуемой ячейки.

1.3 Функциональные требования на основе анализа

Веб-приложение должно быть разработано с учетом данных требований:

1. Веб-приложение должно обеспечивать возможность регистрации и аутентификации пользователей;
2. Создаваемый программный продукт должен поддерживать возможность одновременной аренды одной ячейки с возможностью выбора расположения конкретного ряда;
3. Должна быть реализована система рекомендации оптимального ряда;
4. Пользователь должен иметь доступ к совершенным посещениям и информации о них;
5. Создаваемое приложение должно реализовывать полноценный функционал администрирования автоматизированного гардероба;
6. Интерфейс приложения должен быть современным и интуитивно понятным. Он должен включать в себя раздел с основной информацией и раздел с навигацией.

2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ РАЗРАБОТКИ ПРИЛОЖЕНИЯ

На этапе, когда ясны все требования к программному продукту, необходимо определить технологический стек, который будет использоваться для достижения поставленных целей.

2.1 Основные используемые технологии

Приложение функционирует в клиент-серверной архитектуре, где большая часть обработки данных выполняется на серверной стороне. Для реализации такого подхода были использованы Spring[4], PostgreSQL[5], React[6].

Использование фреймворка Spring обусловлено его популярностью среди Java-фреймворков[7] и наличием большого количества различных библиотек, позволяющих создавать приложения различного уровня сложности.

Система управления базами данных PostgreSQL выбрана для хранения данных из-за удобства использования, надежности и стабильности. Эта СУБД обладает всеми необходимыми функциями, а также является одной из самых востребованных на текущий момент.

Для разработки клиентской части были выбраны React, CSS и HTML, так как они обеспечивают эффективную разработку масштабируемых веб-приложений, позволяя создавать переиспользуемые компоненты для более удобной поддержки работоспособности пользовательского интерфейса.

2.2 Краткое обоснование использования технологий

2.2.1 Язык программирования Java

Данный язык зарекомендовал себя в написании приложений различного уровня сложности, а также обладает обширным списком библиотек позволяющих решать множество задач.

2.2.2 Spring

Данный фреймворк[8] отлично зарекомендовал себя на рынке решений для разработки веб-приложения на языке Java, а также обладает большим количеством различных дополнений расширяющих его функционал.

2.2.3 Spring Boot

Данное средство упрощает создание автономных приложений на основе Spring, облегчая настройку и развертывание приложений[9].

2.2.4 Spring Web

Данный веб-фреймворк позволяет удобно создавать REST-приложения на основе Spring.

2.2.5 Spring Data JPA

Данная технология является частью семейства Spring Data и позволяет удобно взаимодействовать с базами данных через Java Persistence API [10].

2.2.6 Spring Security

Данный фреймворк позволяет гибко настраивать аутентификацию и фактически является стандартом для защиты основанных на Spring приложений[11].

2.2.7 Maven

Система автоматизированный сборки проекта, используемая для упрощения сборки, тестирования и развертывания приложений.

2.2.8 Bruno

Bruno – это инструмент для тестирования и разработки API веб-сервисов. Он обеспечивает простой и удобный интерфейс, возможность настраивать переменные окружения, а также возможность локально сохранять запросы[12].

2.2.9 JSON Web Tokens

Открытый стандарт для создания токенов безопасности в формате JSON[13]. Данный стандарт часто применяется для аутентификации и авторизации пользователей.

2.2.10 IntelliJ IDEA

Интегрированная среда разработки (IDE) для Java от JetBrains с мощными инструментами для создания, отладки и управления проектами. Она включает интеллектуальные подсказки, автоматическое исправление кода, интеграцию с системами сборки и анализ кода.

2.2.11 Git

Представляет собой распределённую систему контроля версий, отслеживающая изменения в файлах и обычно используемая для координации работы при совместной разработке исходного кода, а также для возможности отменить изменения в случае появления критических ошибок в работе программ.

2.2.12 GitHub

Веб-платформа, предназначенная для хостинга и совместной разработки программного обеспечения с использованием системы контроля версий Git. Предоставляет широкий инструментарий совместно с приятным и удобным интерфейсом.

2.2.13 GitVerse

Веб-платформа для совместной разработки и хостинга кода, созданная и размещенная в России, что исключает риски недоступности разработок и кода в настоящее время.

2.2.14 PostgreSQL

Система управления реляционными базами данных, часто используемая в веб-разработке и других сферах, обладающая высокой надежностью, стабильностью и большим сообществом разработчиков [14].

2.2.15 HyperText Markup Language (HTML)

Язык разметки, который применяется для создания структуры и представления контента на веб-страницах и поддерживается всеми современными браузерами.

2.2.16 Cascading Style Sheets (CSS)

Язык таблиц стилей, который используется для оформления веб-страниц, позволяющий изменять стандартное отображение тегов HTML.

2.2.17 JavaScript (JS)

Язык программирования, обеспечивающий возможность динамического изменения содержимого веб-страниц, поддерживаемый современными браузерами и лежащий в основе большого количества фреймворков.

2.2.18 React

Библиотека JavaScript библиотека для создания внешних пользовательских интерфейсов. С помощью компонентного подхода React и встроенных систем оптимизации можно просто создавать нагруженные и масштабируемые приложения.

2.2.19 Docker

Является монополистом в сфере контейнеризации приложений, необходимой для удобного и быстрого развертывания в различных системах.

2.2.20 Docker-compose

Данное программное обеспечение для автоматизации развертывания и управления приложениями. Оно позволяет с помощью конфигурационного файла собирать и запускать сразу несколько сервисов, что облегчает разработку и масштабирование.

2.2.21 Lombok

Библиотека для языка программирования Java, предоставляющая аннотации для автоматической генерации методов кода, таких как геттеры, сеттеры, конструкторы и другие.

2.2.22 Yandex Cloud

Публичная облачная платформа от транснациональной интернет-компании «Яндекс», позволяющая размещать решения на российских серверах с удобной панелью управлению и возможностью масштабироваться.

2.2.23 Spring WebSocket

Библиотека семейства Spring, позволяющая использовать возможности постоянного соединения для передачи данных между веб-браузером и сервером.

3 АРХИТЕКТУРА WEB-ПРИЛОЖЕНИЯ

3.1 Архитектура серверной части приложения

3.1.1 Описание архитектуры

Для построения архитектуры веб-приложения важно выявить ключевые бизнес-правила, лежащие в основе разрабатываемой системы. В диаграмме вариантов использования, представленной на рисунке 3.1, отражены возможные сценарии взаимодействия, основанные на данных бизнес-правилах.

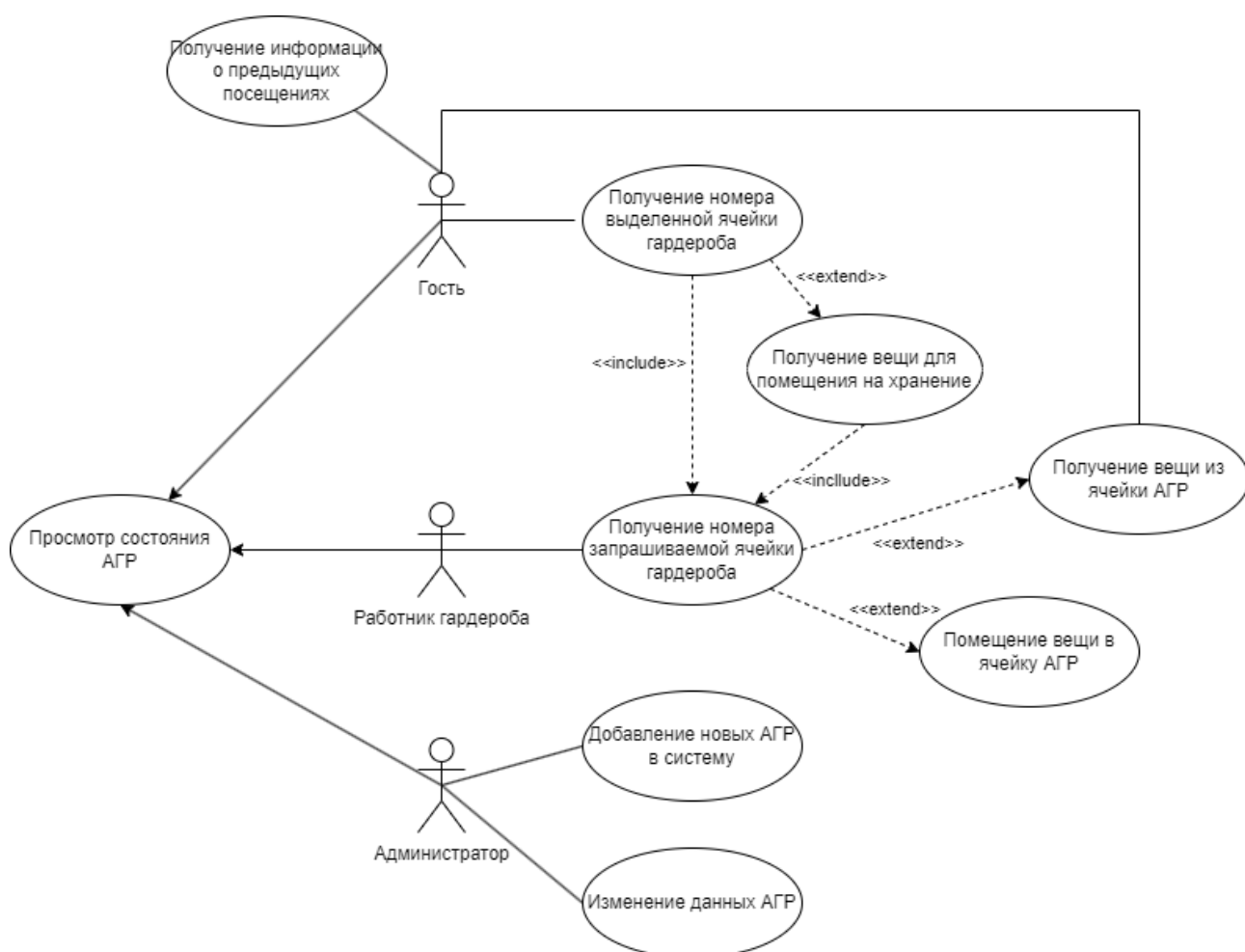


Рисунок 3.1 – Диаграмма вариантов использования веб-приложения

В качестве архитектуры серверной части приложения была использована схема MVC, которая разделяет приложение на несколько слоев, таких как «модель», «представление», «контроллер». Данный подход предполагает полную независимость данных от представления, при которой связующим

звеном выступают контроллеры, выполняющие функции получения и распределения запросов по обработчикам[15].

В бэкенд-приложениях в качестве представления выступают ответы от сервера клиенту в различных форматах. В данном случае для представления данных клиенту используется JSON формат, содержащий данные в соответствии с установленным DTO.

В качестве контроллеров были использованы REST-контроллеры, принимающие запросы по определенным конечным точкам, обрабатывающие JSON тело запроса, переменные пути и GET-параметры, проводящие предварительную валидацию и трансформацию в DTO, который в дальнейшем преобразовывается в объект, используемый «моделью», а также направляющие ответ от «модели» клиенту. Это обеспечивает надежное и эффективное взаимодействие с пользователем.

Понятие «модель» в рамках реализации данного приложения разделяется на различные «сервисы» и «репозитории». Репозитории выполняют задачу взаимодействия с базой данных, а сервисы реализуют бизнес логику приложения.

3.1.2 Пример реализации архитектуры

Обработка запроса состоит из нескольких этапов, изначально запрос попадает в rest-controller, представленный в листинге 3.1

Листинг 3.1 – Фрагмент контроллера для обработки запросов по компаниям

```
@RestController
@Tag(name = "Работа с компаниями")
public class CompanyController {
    private final CompanyService service;
    private final BranchService branchService;
    private final CompanyMapper mapper;

    @Operation(summary = "Создание компании")
    @PostMapping("/company")
    public CompanyCompactResponse createCompany(@RequestBody @Valid
CompanyCreationRequest request) {
        Company company =
service.create(mapper.fromCreateRequest(request));
        return mapper.toCompactResponse(company);
    }
}
```

После ответственность за обработку запрошенных данных берет на себя сервис получающий данные из репозитория, что представлено в листинге 3.2.

Листинг 3.2 – Фрагмент сервиса для обработки данных компаний

```
@Service
@RequiredArgsConstructor
public class CompanyService {
    @Transactional
    public Company create(Company company) {
        var manager =
userService.getByIdStrict(company.getManager().getId());

        return save(
            Company.builder()
                .status(company.getStatus())
                .inn(company.getInn())
                .name(company.getName())
                .physical_address(company.getPhysical_address())
                .legal_address(company.getLegal_address())
                .manager(manager).build());
    }
}
```

После обработки в качестве ответа сервер возвращает DTO, представленный в листинге 3.3.

Листинг 3.3 – DTO для представления данных пользователю

```
@Schema(description = "Сокращенный ответ на запрос данных компании")
public record CompanyCompactResponse(
    @Schema(description = "Идентификатор компании", example =
"1")
    Long id,
    @Schema(description = "Статус компании", example = "active")
    String status,
    @Schema(description = "Название компании", example = "Рога
и Копыта")
    String name,
    @Schema(description = "ИНН компании", example =
"123456789012")
    String inn,
    @Schema(description = "Физический адрес компании", example
= "ул. Пушкина, д. 12")
    String physical_address,
    @Schema(description = "Юридический адрес компании", example
= "ул. Колотушкина, д. 1")
    String legal_address,
    @Schema(description = "Отвественный")
    HiddenUserResponse manager
) {
}
```

3.2 Архитектура клиентской части приложения

3.2.1 Описание архитектуры

Для разработки клиентской части веб-приложения был выбран компонентный подход, при котором в разрабатываемом приложении представлены отдельные компоненты содержащие в себе определенную логику отображения данных, они в свою очередь могут использовать другие компоненты, использующиеся для отправки и обработки запросов на сервер. Данный подход был выбран в следствии простоты и удобства разработки, а также в следствии малого размера приложения.

3.2.2 Пример реализации архитектуры

Для примера, рассмотрим страницу на которую пользователь попадает если он не обладает ролью «администратора» или «исполнителя» и при этом авторизован на сервисе. В ней используется отдельный компонент для отображения страницы контента, отдельные компоненты для отображения списка данных, а также объект store для отправки запросов на сервер, что представлено в листинге 3.4.

Листинг 3.4 – Фрагмент кода страницы выбора компании

```
const ChooseCompanyPage = () => {
  const {store} = useContext(Context);
  const navigate = useNavigate();
  useEffect(() => {
    store.cells.getCurrentCell().then((response) =>
response.length > 0 ? navigate(`/agr/${response[0].agr_id}`) : "")
  }, [])
  return (
    <PageTemplate
      label="Выберите компанию">
      <ItemList
        fetchItems={(filter) =>
store.companies.getCompaniesByFilter(filter.id, "active",
filter.name ,filter.inn, null, null)}
        createItem={null}
        renderItem={({ key, item, wasChanged, setWasChanged
}) => (
          <UserChooseItem key={key} item={item}
baseUrl={"companies"} text={item.name}/>
        )}
      />
    </PageTemplate>
  );
};
```

При попадании на страницу отправляется запрос на сервер для получения активных компаний с учетом изначальных параметров фильтра, которые при получении будут преобразованы в карточки UserChooseItem.

4 РАЗРАБОТКА БАЗЫ ДАННЫХ

4.1 Определение сущностей

Роли пользователей: Таблица «user_roles» представляет собой определение возможных ролей пользователей. Сущность UserRole.

Пользователи: Таблица «users» представляет собой учетные записи всех пользователей приложения. Сущность User.

Компании: Таблица «companies» представляет собой данные компаний, использующих автоматизированный гардероб. Сущность Company.

Филиалы: Таблица «branches» представляет собой данные о филиалах компаний. Сущность Branch.

Автоматизированные гардеробные ряды: Таблица «agrs» представляет собой данные о гардеробных рядах, используемых в филиалах. Сущность Agr.

Ячейки: Таблица «cell» представляет собой данные о ячейках внутри гардеробных рядов. Сущность Cell.

Визиты: Таблица «visit» представляет собой данные о использовании пользователями ячеек. Сущность Visit.

4.2 Создание сущностей на уровне СУБД

В разработанном приложении в качестве СУДБ была использована PostgreSQL. ER-диаграмма используемой версии базы данных представлена на рисунке 4.1.

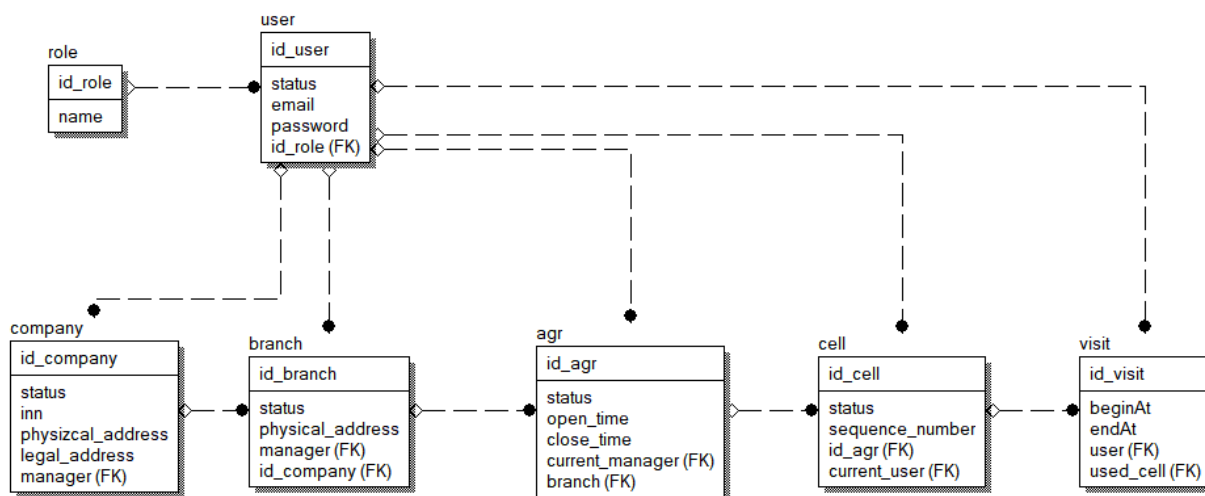


Рисунок 4.1 – ER-диаграмма используемой версии базы данных

5 РЕАЛИЗАЦИЯ СЛОЯ СЕРВЕРНОЙ ЛОГИКИ

5.1 Структура проекта

В директории проекта расположены файлы необходимые для запуска и сборки приложения, такие как `pom.xml` и `dockerfile`. Основные исходные файлы проекта находятся в папке `src`. Содержимое директории представлено на рисунке 5.1.

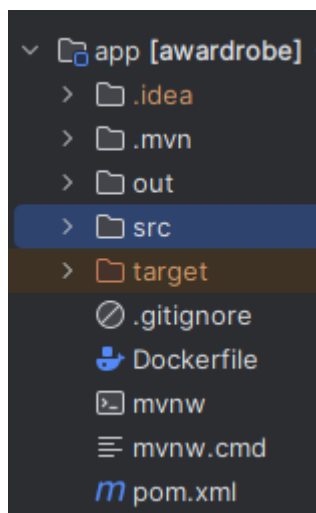


Рисунок 5.1 – Содержимое директории серверной части проекта

На рисунке 5.2 представлена структура директории `src`, в которой находятся ресурсы в директории `resources` для указания конфигурационных переменных, а также основной исходный код приложения.

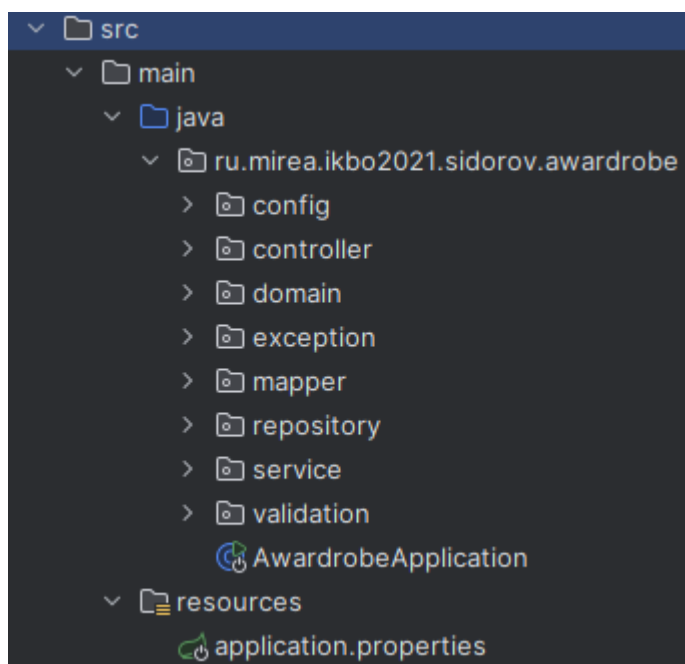


Рисунок 5.2 – Содержимое директории `src`

5.2 Конфигурация приложения

Основная конфигурация приложения расположена в файле `application.properties` и состоит из обращений к переменным окружения установленным из вне. Содержимое данного файла представлено в листинге 5.1.

Листинг 5.1 – Содержимое файла `application.properties`

```
spring.application.name=awardrobe
server.port=${SERVER_PORT}
spring.datasource.url=${SPRING_DATASOURCE_URL}
spring.datasource.driver-class-name=${SPRING_DATASOURCE_DRIVER_CLASS_NAME}
spring.datasource.password=${SPRING_DATASOURCE_PASSWORD}
spring.jpa.database-platform=${SPRING_JPA_DATABASE_PLATFORM}
spring.jpa.properties.hibernate.dialect=${SPRING_JPA_PROPERTIES_HIBERNATE_DIALECT}
spring.jpa.show-sql=${SPRING_JPA_SHOW_SQL}
spring.jpa.hibernate.ddl-auto=${SPRING_JPA_HIBERNATE_DDL_AUTO}
spring.jpa.generate-ddl=${SPRING_JPA_GENERATE_DDL}
token.signing.key:
413F4428472B4B6250655368566D5970337336763979244226452948404D6351
superuser.id = 1
superuser.enabled = true
superuser.default.password = superuser
```

5.3 Конфигурация безопасности проекта

Одним из основных классов конфигурации является класс расширяющий базовые возможности безопасности проекта с использование Spring Security. Данная конфигурация не позволяет пользователю пользоваться частью системы без авторизации, а также автоматически аутентифицирует пользователя при каждом запросе. Содержимое данного класса представлено в листинге 5.2.

Листинг 5.2 – Фрагмент кода класса SecurityConfiguration

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
@EnableMethodSecurity
@Import(SecurityProblemSupport.class)
public class SecurityConfiguration {
    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final UserSecurityService userSecurityService;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
        http.csrf(AbstractHttpConfigurer::disable)
            .authorizeHttpRequests(request -> request
                .requestMatchers("/auth/**").permitAll()
                .requestMatchers("/swagger-ui/**",
"/swagger-resources/**", "/v3/api-docs/**").permitAll()
                .anyRequest().permitAll())
            .sessionManagement(manager ->
manager.sessionCreationPolicy(STATELESS))
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
}
```

5.4 Тестирование

В данном подразделе продемонстрирован фрагмент процесса тестирования серверной части веб-приложения, в ходе которого, были протестированы все функции заложенные в приложение.

Запросы для регистрации и авторизации пользователя представлены на рисунках 5.3 – 5.4.

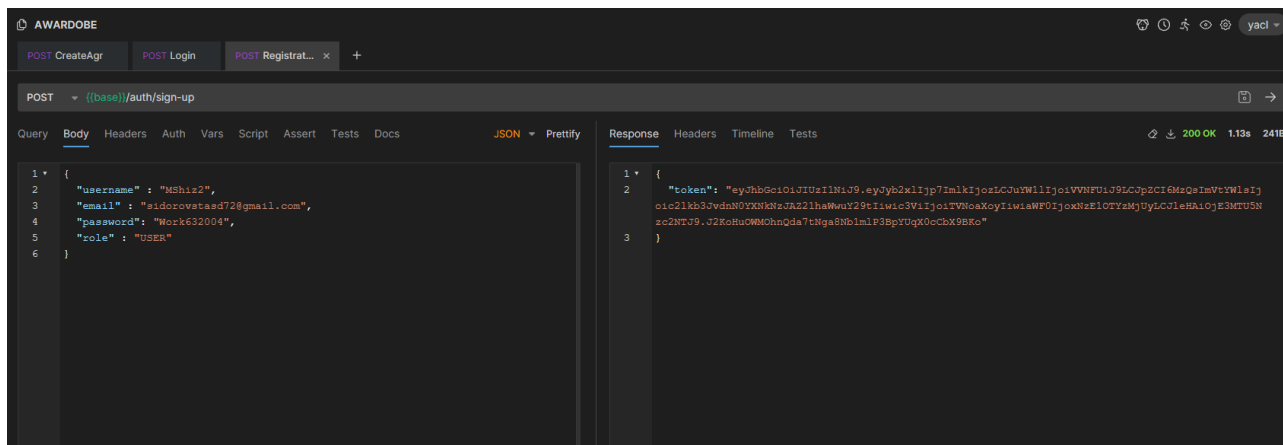


Рисунок 5.3 – Регистрация пользователя

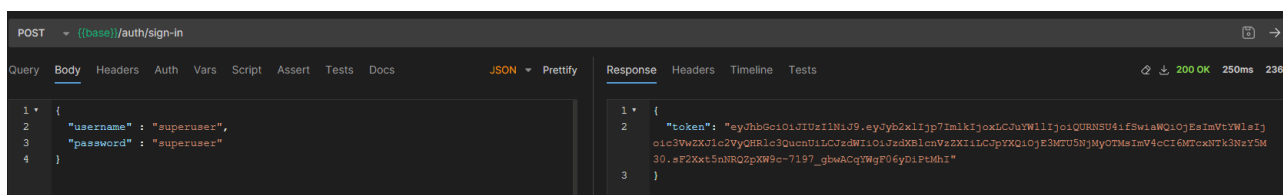


Рисунок 5.4 – Авторизация пользователя

Запросы на взаимодействие с сущностью компании представлены на рисунках 5.5 – 5.8.

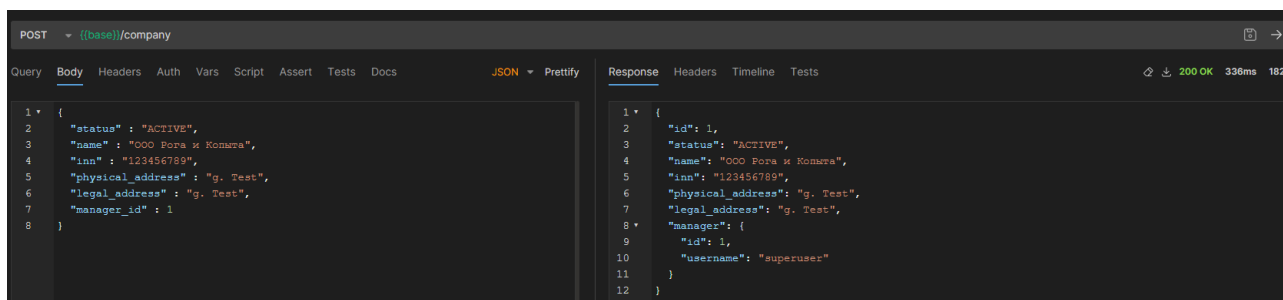


Рисунок 5.5 – Создание новой компании

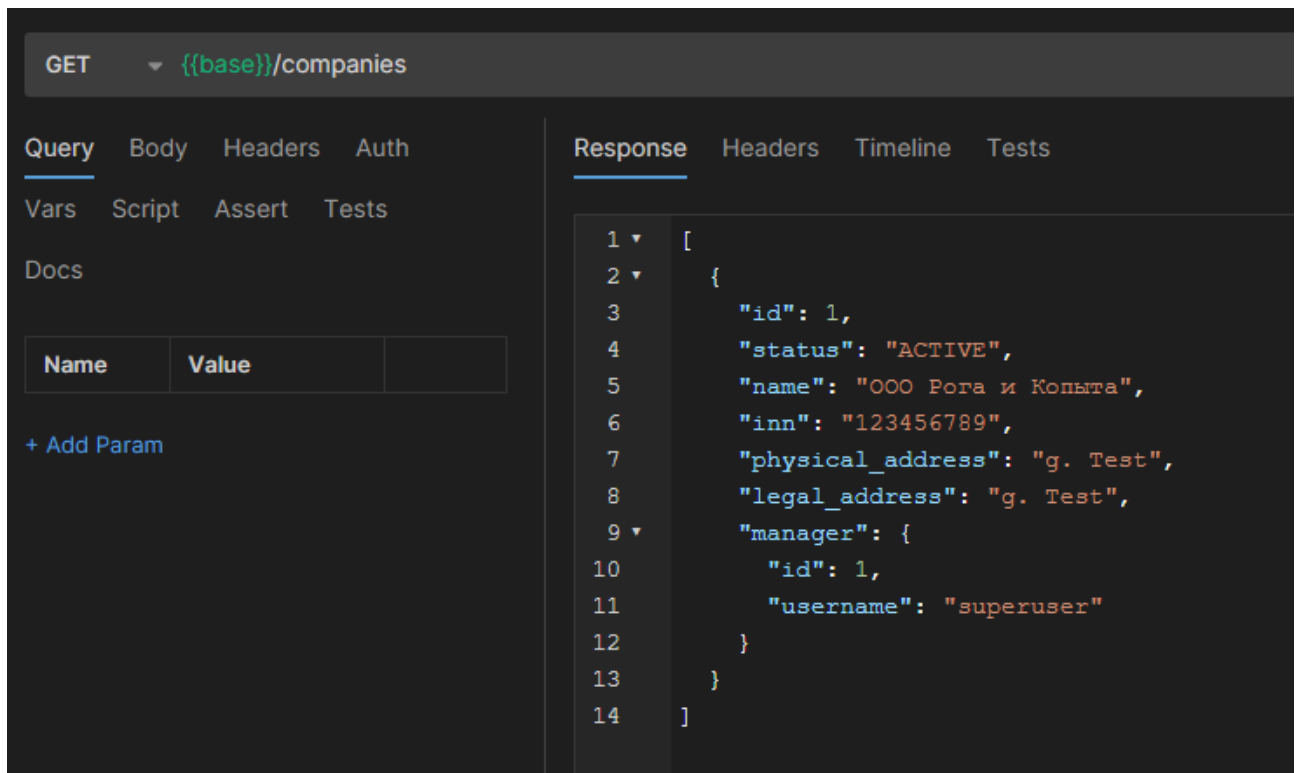


Рисунок 5.6 – Получение списка текущих компаний

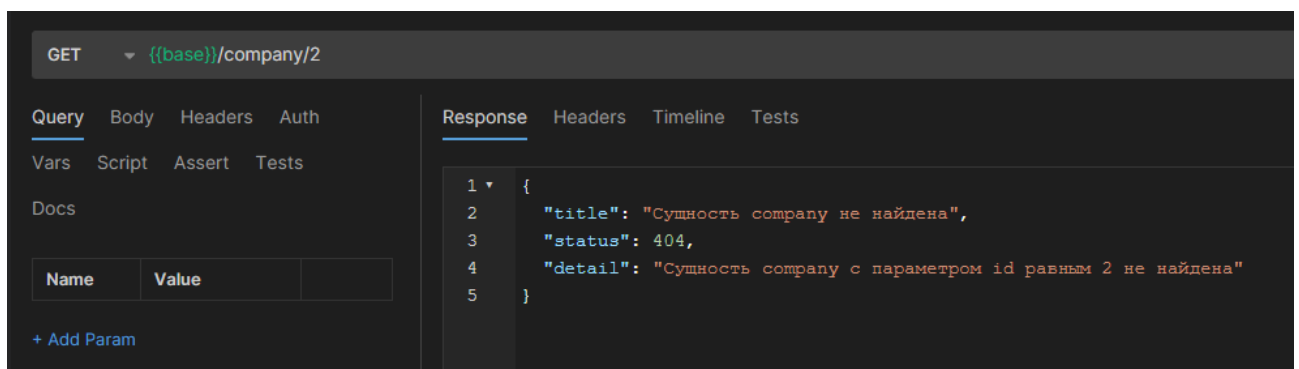


Рисунок 5.7 – Обработка ошибки при получении некорректного ID

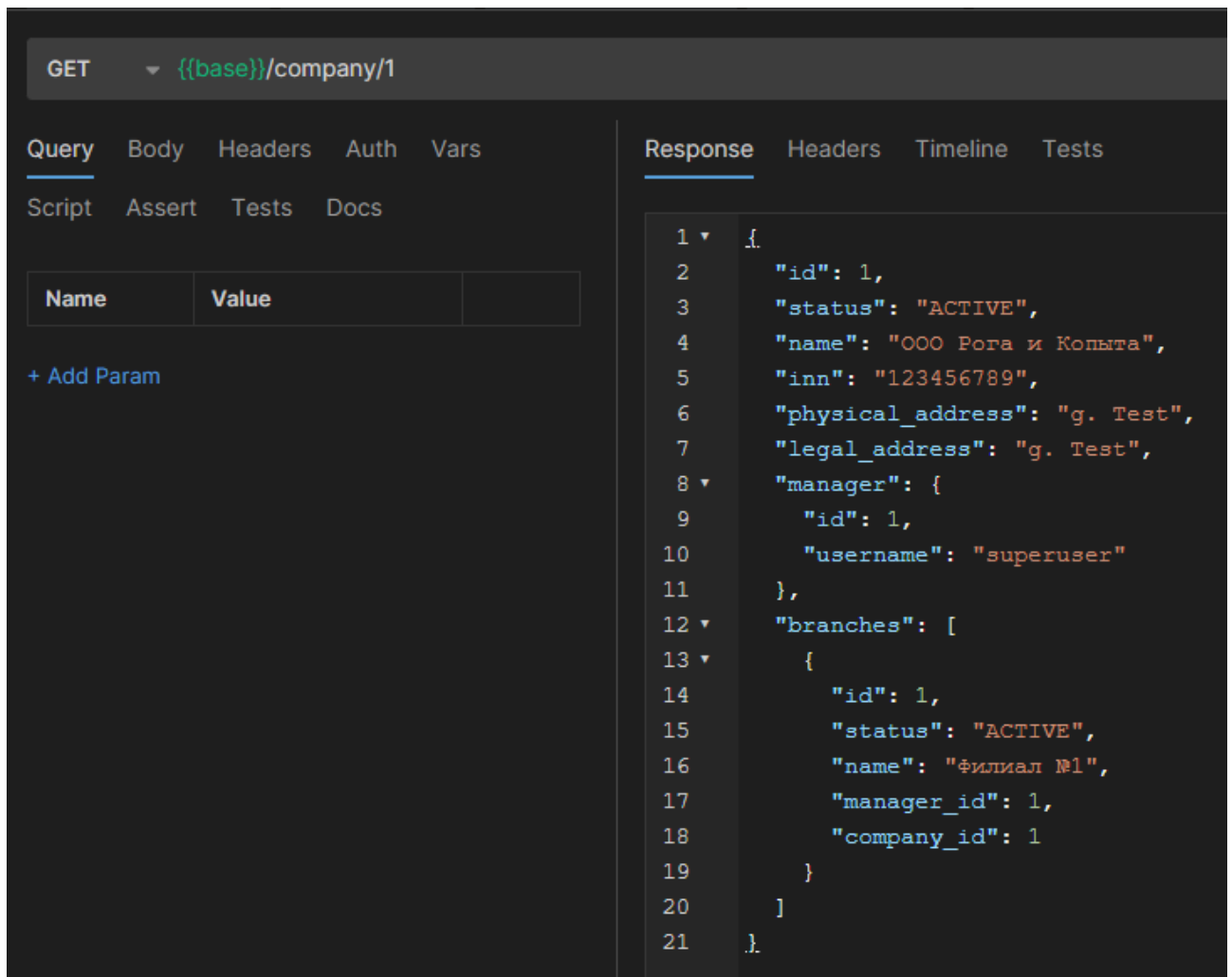


Рисунок 5.8 – Получение полной информации о компании по ID

Запросы для взаимодействия с сущностью ячеек приведены на рисунках 5.9 – 5.12.

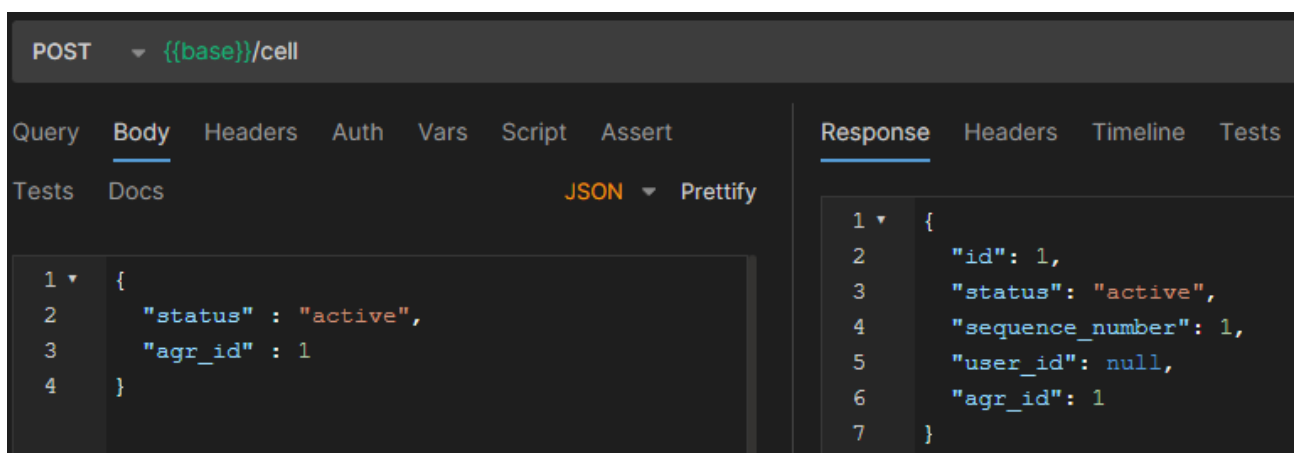


Рисунок 5.9 – Создание одной ячейки

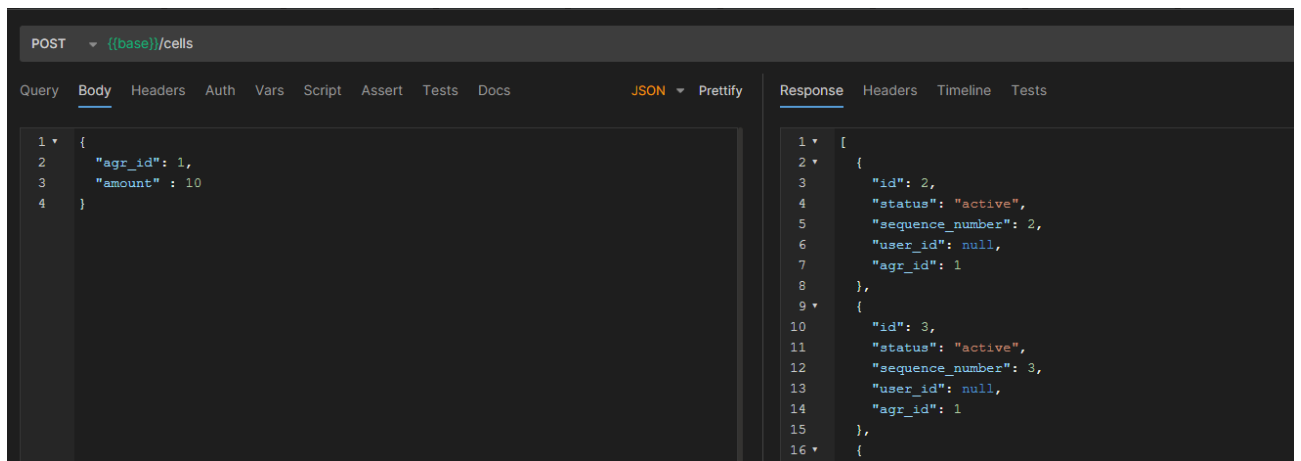


Рисунок 5.10 – Создание 10 ячеек

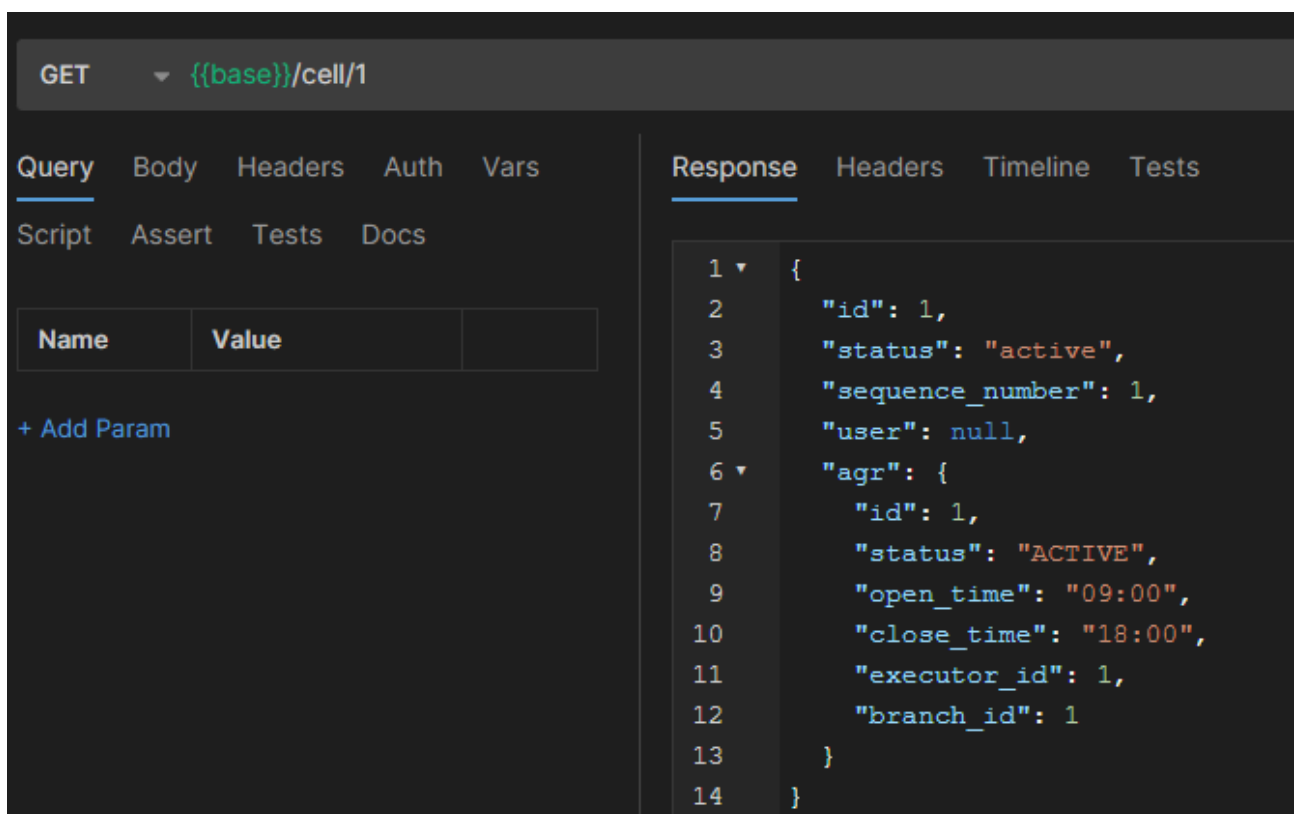


Рисунок 5.11 – Получение полных данных одной ячейки

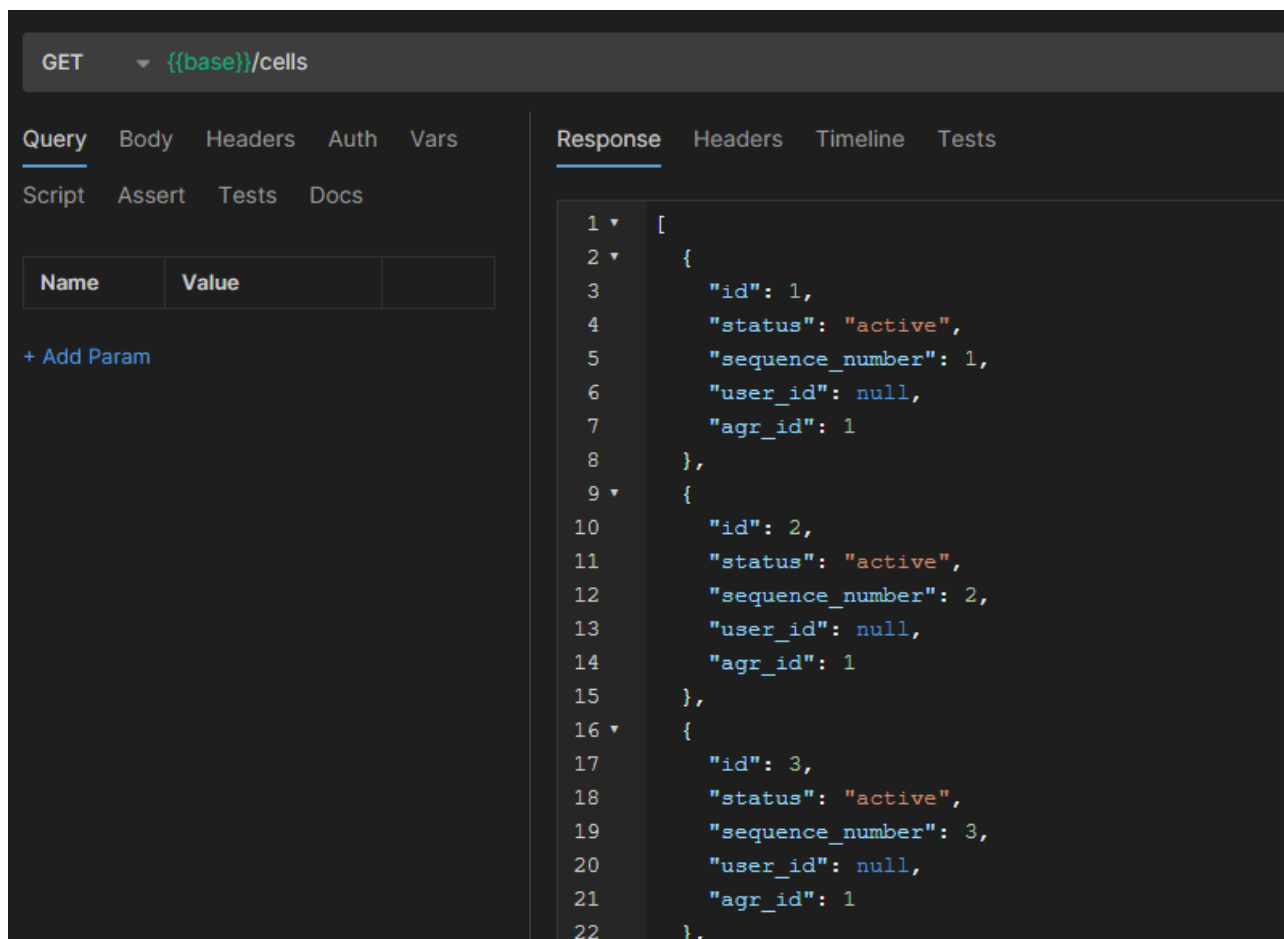


Рисунок 5.12 – Получение списка ячеек без фильтрации

6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ

6.1 Структура проекта

В директории проекта содержатся технические файлы и директории, а также файлы содержащие Глогику взаимодействия с сервером, которые находятся в директории store, файлы содержащие компоненты страниц, которые находятся в дирректории pages, и файлы для часть повторяющихся компонентов в директориях components, entities, UI. Структура проекта представлена на рисунке 6.1.

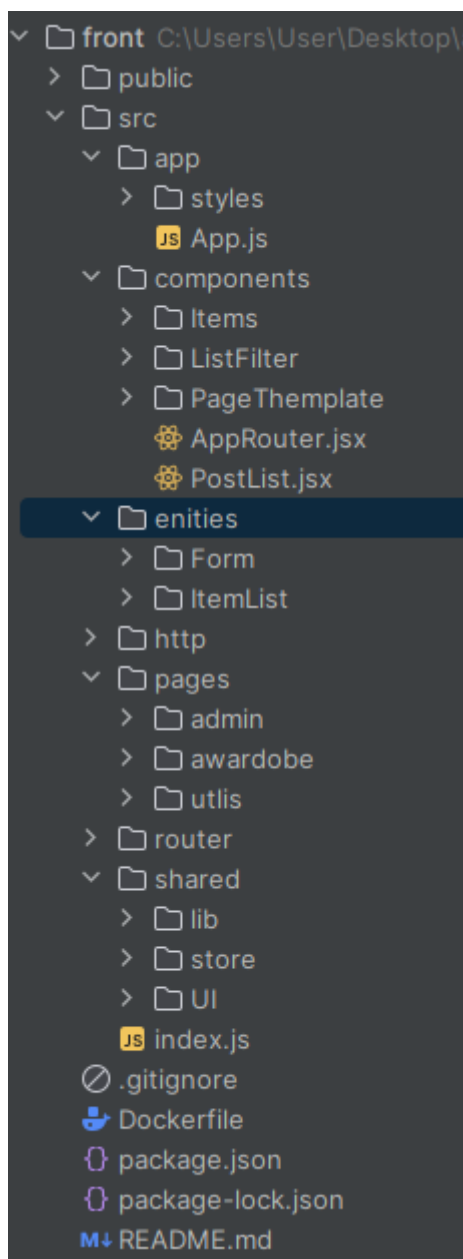


Рисунок 6.1 – Структура клиентской части веб-приложения

6.2 Взаимодействие с серверной частью веб-приложения

Для отправки запросов на сервер используется `axios`, конфигурация которого представлена в листинге 6.1.

Листинг 6.1 – Код конфигурации `axios` для отправки запросов на сервер

```
import axios from 'axios';
export const API_URL = 'http://158.160.157.247:8081'

const $api = axios.create({
  withCredentials: true,
  baseURL: API_URL,
  headers: {
    'Access-Control-Allow-Origin': "*",
    'Access-Control-Allow-Headers': 'Origin, X-Requested-With, Content-Type, Accept',
    'Access-Control-Allow-Methods': 'GET, POST, PATCH, PUT, DELETE, OPTIONS',
    'Content-Type': 'application/json',
  }
})

$api.interceptors.request.use((config) => {
  let token = localStorage.getItem('token')
  if (token !== null) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config;
} )
Export default $api;
```

Для отправки запросов использовался метод класса `AppStore`, обрабатывающий ошибки и устанавливающий процесс загрузки, отслеживаемый с помощью библиотеки `mobx`. Фрагмент кода `AppStore` представлен в листинге 6.2.

Листинг 6.2 – Фрагмент кода класса AppStore.

```
export default class AppStore {

  users = new UserStore(this);
  companies = new CompanyStore(this);
  branches = new BranchStore(this);
  agrs = new AgrStore(this);
  cells = new CellStore(this);
  visits = new VisitStore(this);
  execution = new ExecutionStore(this);

  userState = null;
  isAuthState = false;
  isSuperAdminState = null;

  isLoading = false;
  hasUpdate = false;
  performRequest = async (request, isUpdate = false) => {
    try {
      this.loading = true;
      const response = await request;
      return response.data;
    }
    catch (e) {
      this.httpError(e);
    }
    finally {
      if (isUpdate){
        this.hasUpdate = !this.hasUpdate;
      }
      this.loading = false;
    }
    return [];
  }
}
```

Для работы с конкретными сущностями используются классы входящие в состав AppStore, например CompanyStore фрагмент кода, которого представлен в листинге 6.3.

Листинг 6.3 – Фрагмент кода класс CompanyStore.

```
export default class CompanyStore{
  rootStore: AppStore;
  isLoading = false;

  constructor(rootStore: AppStore) {
    makeAutoObservable(this);
    this.rootStore = rootStore;
  }

  get isLoadingState() {
    return this.isLoading;
  }

  set isLoadingState(state) {
    this.isLoading = state;
  }

  getCompaniesByFilter = async (id, status, name, inn, p_address,
                                l_address, manager_id) => await
    this.rootStore.performRequest($api.post('/companies/filter', {
      id: id,
      status: status,
      name: name,
      inn: inn,
      physical_address: p_address,
      legal_address: l_address,
      manager_id: manager_id
    }));

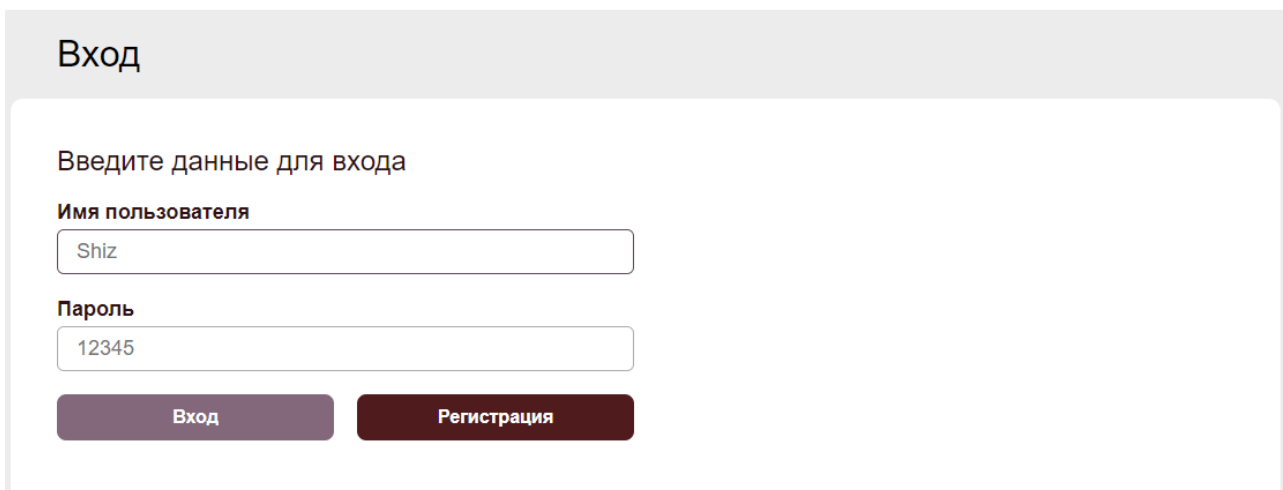
  getAllCompanies = async () => await
    this.rootStore.performRequest($api.get('/companies'));
}
```

6.3 Вид конечного приложения

На рисунках 6.1 – 6. Изображена часть страниц получившегося ресурса.

AWARDROBE

Вход



Вход

Введите данные для входа

Имя пользователя

Shiz

Пароль

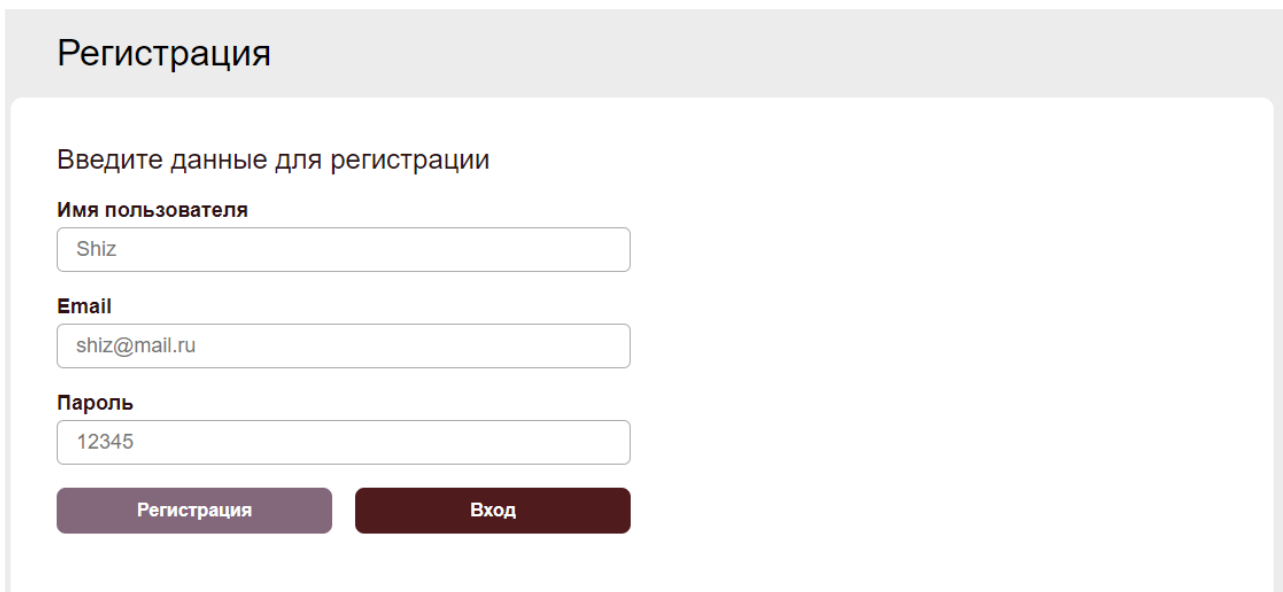
12345

Вход Регистрация

Рисунок 6.1 – Страница авторизации

AWARDROBE

Вход



Регистрация

Введите данные для регистрации

Имя пользователя

Shiz

Email

shiz@mail.ru

Пароль

12345

Регистрация Вход

Рисунок 6.2 – Страница регистрация

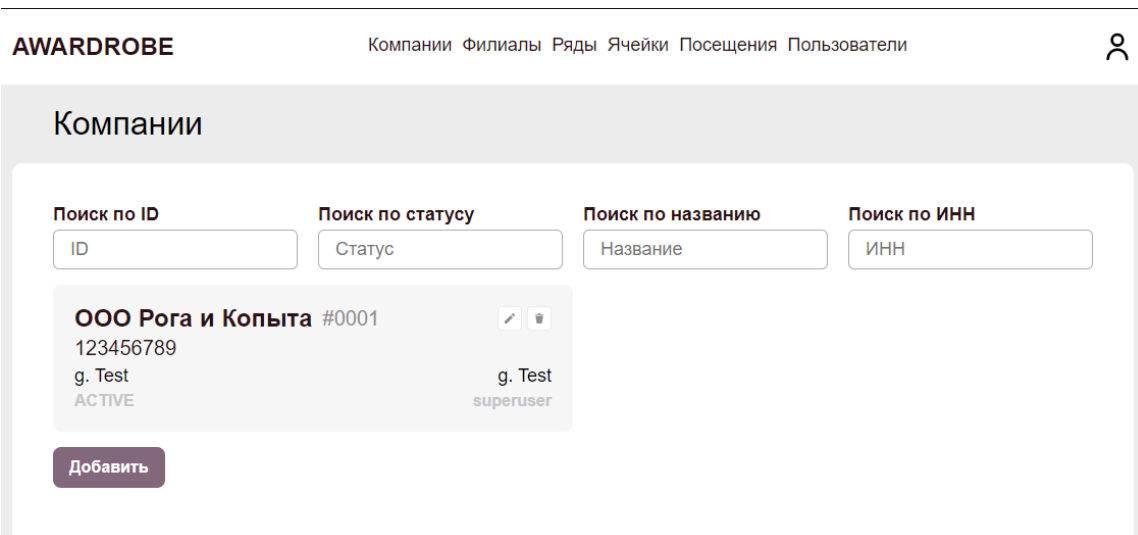


Рисунок 6.3 – Страница взаимодействия с компаниями

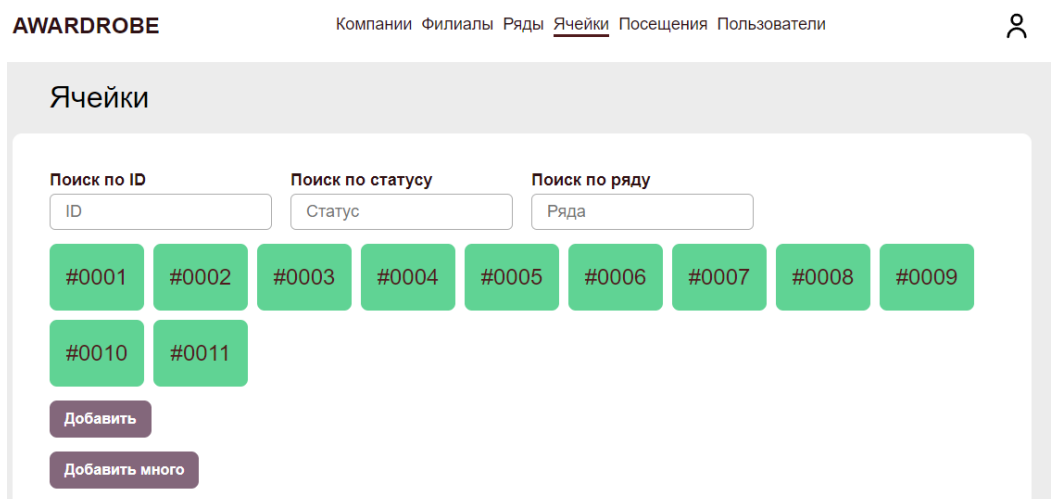


Рисунок 6.6 – Страница взаимодействия с ячейками

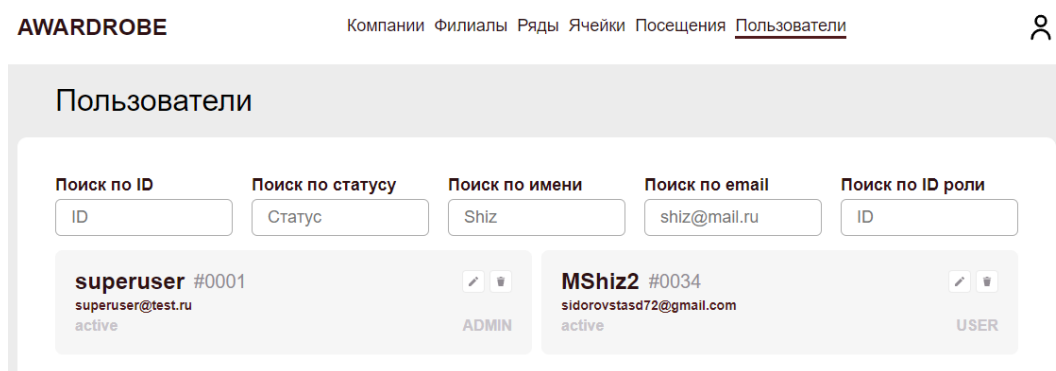


Рисунок 6.7 – Страница взаимодействия с пользователями

AWARDROBE

Пользователь

Имя пользователя

Sidorov

Электронная почта

sid@mail.ru

Роль

USER

Пароль

Выйти

Изменить

Ваши посещения

Поиск по ID

ID

Поиск по ID ячейки

ID ячейки

Посещение #0001

С 2024-05-17 18:53:46

По 2024-05-17 18:54:16

Ячейка: 12 Пользователь: 35

Рисунок 6.8 – Страница пользователя

AWARDROBE

Выберите компанию

Поиск по ID

ID

Поиск по названию

Название

Поиск по ИНН

ИНН

ООО Рога и Копыта

Рисунок 6.9 – Страница выбора компании пользователем

AWARDROBE

Выберите ряд

Рекомендован ряд #0001

Поиск по ID

ID

Ряд #0001

Рисунок 6.11 – Страница выбора ряда пользователем

Отправьте запрос

Сдать вещь

Рисунок 6.12 – Страница взаимодействия с сотрудником гардероба при
отсутствии прежних действий



Рисунок 6.13 – Выбор ряда со стороны исполнителя

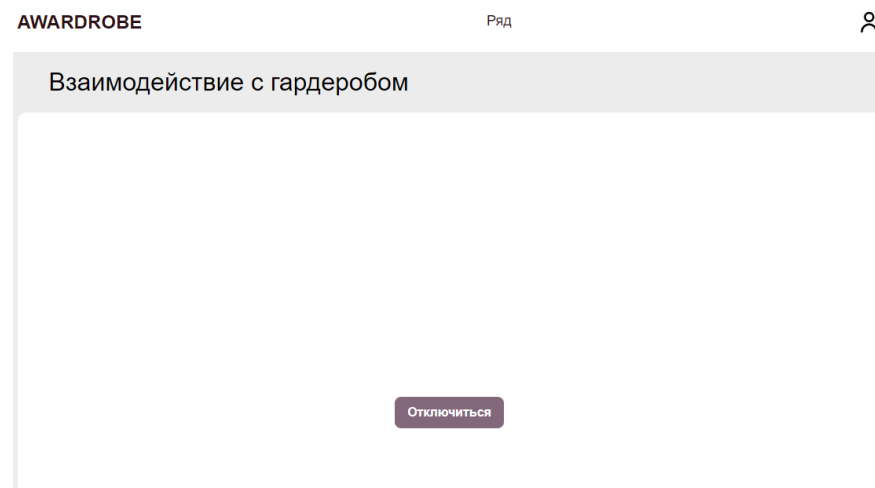


Рисунок 6.14 – Страница ожидания пользователя исполнителем

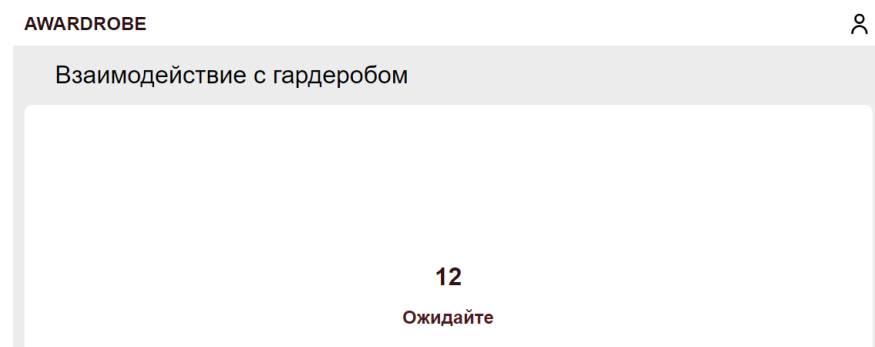


Рисунок 6.15 – Страница сдачи пользователем вещи



Рисунок 6.16 – Страница взаимодействия исполнителя с вещью



Рисунок 6.17 – Страница получения пользователем вещи

7 КОНТЕЙНИРИЗАЦИЯ И ДЕПЛОЙ

Для контейнеризации и развертывания использовался Docker и docker-compose на виртуальной машине Yandex Cloud.

На листингах 7.1 и 7.2 показаны dockerfile для сборки серверной и клиентской части соответственно.

Листинг 7.1 – Содержимое dockerfile серверной части

```
FROM maven:latest AS builder
COPY src /home/app/src
COPY pom.xml /home/app
RUN mvn -f /home/app/pom.xml clean package

FROM openjdk:21-slim
COPY --from=builder /home/app/target/awardrobe-0.0.1-SNAPSHOT.jar
/usr/local/lib/awardrobe.jar
EXPOSE $SERVER_PORT
ENTRYPOINT ["java", "-jar", "/usr/local/lib/awardrobe.jar"]
```

Листинг 7.2 – Содержимое dockerfile клиентской части

```
FROM node:13.12.0-alpine
WORKDIR /app
ENV PATH /app/node_modules/.bin:$PATH
COPY package.json .
RUN npm install
RUN npm install react-scripts@3.4.1 -g
COPY . ./
CMD ["npm", "start"]
```

Фрагмент содержимого файла docker-compose.yml, используемого для общей сборки сервисов представлен в листинге 7.3.

Листинг 7.3 –Фрагмент содержимого файла docker-compose.yml

```
version: '3'

services:
  app:
    build:
      context: ./app
      dockerfile: Dockerfile
    ports:
      - "8081:8080"
    depends_on:
      - db
    networks:
      - delivery
  react-app:
    build:
      context: ./front
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    volumes:
      - ./front:/app
      - /app/node_modules
    stdin_open: true
    depends_on:
      - app
    networks:
      - delivery

networks:
  delivery:
```

На рисунке 7.1 представлено изображение виртуальной машины на облачном хостинге Yandex Cloud используемом для развертывания сервиса.

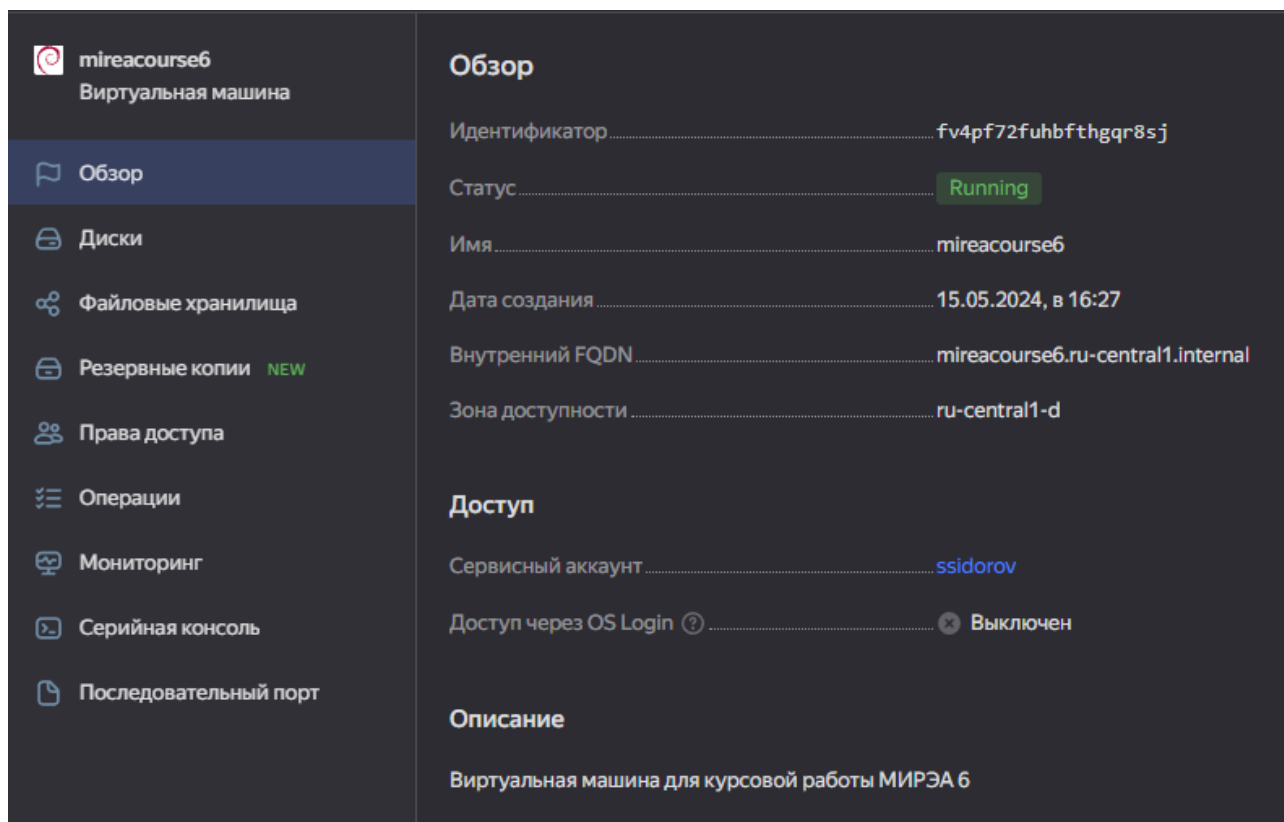


Рисунок 7.1 – Обзор виртуальной машины на хостинге

ЗАКЛЮЧЕНИЕ

В результате выполнения данной курсовой работы была успешно достигнута цель – разработка клиент-серверного фуллстек-приложения для автоматизированного гардероба. Проект, представленный в репозитории по ссылке <https://github.com/MShizikU/awardrobe>, представляет собой серверную и клиентскую часть веб-приложения.

Для достижения данной цели был проведен анализ предметной области, были сформированы функциональные требования, была разработана архитектура приложения и реализована логика функционирования серверной и клиентской частей, а также была произведена систематизация знаний и навыков по использованию различных методов и подходов к разработке программного обеспечения.

В ходе курсовой работы были определены основные сущности системы, спроектирована база данных с необходимыми таблицами и зависимостями, разработана и протестирована бизнес-логика приложения. Осуществлена настройка системы позволяющая менять часть конфигурации не изменяя код приложения. Полученные навыки и знания позволили эффективно разработать и оформить клиент-серверное фуллстек-приложение для автоматизированного гардероба.

Доступ к веб-приложению можно получить по адресу <http://158.160.157.247:3000/>

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Whoosh [Электронный ресурс]. – Режим доступа: <https://whoosh-bike.ru> (дата обращения 30.04.2024).
2. Яндекс Go [Электронный ресурс]. – Режим доступа: <https://go.yandex.ru/ru/lp/rides/scooter> (дата обращения 30.04.2024).
3. Яндекс Drive [Электронный ресурс]. – Режим доступа: <https://yandex.ru/drive/> (дата обращения 30.04.2024).
4. Уоллс К. Spring в действии – М.: ДМК Пресс, 2013. – 752 с.
5. Оптимизация запросов PostgreSQL – М.: ДМК Пресс, 2022. – 25 с.
6. Тилен Томас М. React в действии – М.: Санкт-Петербург: Питер (дата обращения: 01.05.2024).
7. Сайт компании JavaRush. Программирование на языке Java [Электронный ресурс] – URL: <https://javarush.com> (Дата последнего обращения: 30.04.2024).
8. Раджпут Д. Spring. Все паттерны проектирования. – М.: СПб.: Питер, 2019 – 320 с.
9. Сайт технологии Spring. Документация по использованию Spring Data Jpa [Электронный ресурс] – URL: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html> (Дата последнего обращения: 30.04.2024).
10. Сайт технологии Spring. Начало работы со Spring Security [Электронный ресурс] – URL: <https://docs.spring.io/spring-security/reference/index.html> (Дата последнего обращения: 30.04.2024).
11. Хоффман Э. Безопасность веб-приложений – СПб.: Питер, 2021. – 354 с.
12. Сайт Bruno. Re-Inventing the API Client [Электронный ресурс] – URL: <https://www.usebruno.com> (Дата последнего обращения: 17.05.2024).
13. JWT-token — [Электронный ресурс] — URL — <https://jwt.io/> — (Дата обращения: 28.04.2024)
14. Документация PostgreSQL и Postgres Pro [Электронный ресурс] – URL: <https://postgrespro.ru/docs> (Дата последнего обращения: 07.05.2024).

15. Реализация паттерна проектирования MVC с использованием фреймворка spring MVC / А. И. Тымкив, А. В. Федоренко, Ю. Г. Худасова, О. Г. Худасова // Системная трансформация - основа устойчивого инновационного развития: сборник статей Международной научно-практической конференции, Новосибирск, 20 декабря 2021 года. — Уфа: Общество с ограниченной ответственностью "Аэтерна", 2021. — С. 104-108.— (дата обращения: 30.03.2024)