

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА
на тему
Программное обеспечение для мониторинга
животных в зоопарке

Студент	_____	<u>Александр Дмитриевич Шашков</u>
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	<u>Шашков Б.Д.</u>
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролер	_____	<u>Такташкин Д.В.</u>
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	<u>Козлов А.Ю.</u>
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	<u>Попова Н.А.</u>
	(подпись)	(фамилия, инициалы)

Пенза, 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«__» _____ 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Александр Дмитриевич Шашков гр. 19ВП1 факультета ФВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программное обеспечение для мониторинга животных в зоопарке

Тема утверждена приказом ПГУ № _____ от " ____ " _____ 20 ____

3. Руководитель работы к.т.н., Шашков Б.Д.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Назначение: программа предназначена для организации мониторинга за животными в зоопарке

2) Средства разработки: язык программирования Java, JavaScript, интегрированная среда разработки (IDE) IntelliJ IDEA

3) Основные функции программы:

а) просмотр карточек животных;

б) наличие ролевой модели;

в) возможность выдачи администратором задач пользователям;

г) возможность выполнения задач сотрудниками зоопарка;

д) возможность проведения осмотров ветеринара;

е) возможность составления выставок администратором

5. Перечень подлежащих разработке вопросов

Анализ предметной области и постановка задачи

Анализ требований на разработку

<u>Проектирование приложения</u>
<u>Реализация приложения</u>
<u>Тестирование и контроль качества приложения</u>
<u>Планирование разработки и оценка бюджета приложения</u>
<u>Оформление пояснительной записки</u>

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	5%	01.04.2023 08.04.2023	
2	Анализ требований на разработку	5%	08.04.2023 15.04.2023	
3	Проектирование приложения	25%	15.04.2023 30.04.2023	
4	Реализация приложения	35%	30.04.2023 21.05.2023	
5	Тестирование и контроль качества приложения	10%	21.05.2023 28.05.2023	
6	Функционально-стоимостный анализ	10%	28.05.2023 05.06.2023	
7	Оформление пояснительной записки	10%	05.06.2023 13.06.2023	

Дата выдачи задания " ____ " _____ 20 ____

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 71 лист, 19 таблиц, 34 рисунка, 8 использованных источников и 3 приложения.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, МОНИТОРИНГ ЖИВОТНЫХ В
ЗООПАРКЕ, ООП, JAVA, INTELLIJ IDEA, UML

Объект разработки – Программное обеспечение для мониторинга животных в зоопарке.

Цель работы – провести анализ предметной области и поставить задачу на разработку, провести сравнительный анализ конкурирующего программного обеспечения, определиться с инструментальными средствами, спроектировать диаграммы UML и пользовательский интерфейс, разработать программное обеспечение, провести его тестирование и анализ кода с помощью различных метрик, спланировать бюджет разработки.

Средства разработки: язык программирования Java, JavaScript, интегрированная среда разработки IntelliJ IDEA.

Полученные результаты: в ходе выполнения выпускной квалификационной работы была спроектирована программа, которая позволяет организовать мониторинг животных в зоопарке.

					ПГУ 09.03.04 – 08ВР191.13 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Шашков А.Д.			«Программное обеспечение для мониторинга животных в зоопарке»	Лит.	Лист	Листов
Провер.		Шашков Б.Д.					4	71
Н. Контр.		Такташкин Д.В			Пояснительная записка	Группа 19ВП1		
Утверд.		Козлов А.Ю.						

Содержание

Введение	7
1 Анализ предметной области и постановка задачи на разработку	8
1.1 Постановка задачи.....	8
1.2 Анализ информационных систем аналогичного назначения	10
1.3 Jira	10
1.4 Kaiten	12
1.5 Yandex Tracker	13
1.6 Обоснование для разработки программы системы мониторинга за животными в зоопарке.....	14
2 Проектирование программы	16
2.1 Анализ функциональных требований к системе	18
2.2 Проектирование системы	19
2.2.1 Анализ прецедентов.....	19
2.2.3 Проектирование модели предметной области	23
2.2.4 Проектирование структуры БД.....	24
2.2.5 Проектирование обобщенной архитектуры системы.....	25
2.3 Разработка серверного программного обеспечения.....	27
2.4 Разработка клиентского программного приложения	30
2.4.1 Требования к пользовательскому интерфейсу.....	30
2.4.2 Описание архитектуры пользовательского интерфейса	31
3 Тестирование программного обеспечения	37
3.1 Выбор методологии тестирования	37
3.2 Тестирование черного ящика модуля «Регистрация сотрудника»	38
3.2.1 Классы эквивалентности	38

3.2.2 Функциональная диаграмма и таблица решений.....	39
3.3 Модульное тестирование модуля «Регистрация работника»	41
3.4 Функциональное тестирование клиентского программного обеспечения	44
3.5 Метрики кода.....	47
3.5.1 Метрика SLOC.....	48
3.5.2 Метрики Холстеда.....	48
3.5.3 Метрики устойчивости и абстрактности компонент.....	50
3.5.4 Комплексный показатель качества кода.....	52
4 Планирование и оценка бюджета проекта.....	53
Заключение	57
Список использованных источников	58
Приложение А. Подробное описание таблиц базы данных.....	59
Приложение Б. Примеры страниц приложения	63
Приложение В. Код программы.....	65

Введение

Современный мир — мир автоматизации различных сфер жизнедеятельности. Все больше и больше разрабатывается программ, которые выполняют однотипную работу вместо человека.

Одним из примеров, в котором автоматизация еще не обрела популярность является зоопарк. Зоопарки имеют большое количество животных. Сотрудники зоопарка должны выполнять некоторые действия по уходу за животными и наблюдению за их здоровьем. Если зоопарк расположен на большой территории, то координирование действий сотрудников вызывает сложности. Например, двое сотрудников могут покормить одно и то же животное несколько раз.

Данный проект направлен на облегчение работы сотрудников зоопарка. Концепцией приложения является хранение информации о животных, работниках зоопарка и их задачах, доступ к которой имеют одновременно все сотрудники.

1 Анализ предметной области и постановка задачи на разработку

1.1 Постановка задачи

Целью данной работы является разработка и создание системы мониторинга за животными в зоопарке.

Для достижения поставленной задачи были выделены следующие цели:

- разработка базы данных для хранения информации о животных, работниках зоопарка и сопутствующей информации;
- разработка серверного программного обеспечения, обеспечивающего получение, добавление и изменение информации о животных, сотрудниках и их задачах в базе данных;
- разработка клиентского программного обеспечения для комфортной работы пользователей.

Для достижения вышеупомянутых целей были выбраны следующие инструменты:

- postgresSQL для создания базы данных;
- язык программирования Java, Spring Framework для реализации серверного программного обеспечения;
- javaScript и Framework React для создания клиентского программного обеспечения;
- intelliJ IDEA – среда разработки серверного программного обеспечения;
- node JS – программная платформа и Visual Studio Code – среда разработки, используемые для реализации клиентского программного обеспечения.

Выполнение данной работы предполагает разработку WEB-приложения. Приложение состоит из двух основных частей: серверная и клиентская часть. Так как серверная часть написана на языке Java, ее можно запустить на любой системе: Windows, Linux, MacOS. Клиентская часть поддерживается в любом

современном браузере, соответственно, приложение является универсальным и может работать с любыми современными операционными системами.

Теоретической основой разработки в данной работе является руководство принципами «SOLID» и ООП.

При создании программного продукта использование принципов «SOLID» способствует созданию такой программы, которую будет легко поддерживать и расширять в течении долгого времени [1].

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования [2].

Архитектура разрабатываемого приложения подразумевает разбиение частей программы на слои. В основу взят стандарт трехслойной архитектуры, состоящей из следующих слоев: контроллеры, сервисы и репозитории. Контроллеры нужны для связи клиентского ПО с серверным, именно они реализуют те http-запросы, которые отправляются к серверной части. Но контроллерный слой следует реализовывать следующим способом: каждый метод должен принять данные и вернуть результат. Для преобразования принятых данных в результат используется сервисный слой. Он несет всю бизнес логику приложения. Соответственно, контроллерный слой, принимая данные, вызывает логику из сервисного слоя, лишь ожидая результата, чтобы его вернуть. Слой репозитория – это слой для связи серверной части приложения с БД. Соответственно, те данные, которые получены сервисным слоем, сохраняются в БД или по ним происходит выборка именно по средствам слоя репозитория. Далее, получив результат обращения к БД, слой репозитория возвращает его сервисному, тот в зависимости от конкретной логики, преобразует его и возвращает контроллерному, который отправляет его уже как ответ на http-запрос туда, откуда он пришел.

1.2 Анализ информационных систем аналогичного назначения

Первым шагом для написания какого-либо программного обеспечения является понимание целесообразности его создания. Для этого необходимо провести анализ конкурирующего ПО. Для этого было принято решение сравнивать не просто похожие приложения для других зоопарков, ведь идея данной системы заключается в контроле за действиями работников путем выдачи заданий и проверке их выполнения. Но это не единственная предметная область, где применим данный подход. Например, в сфере IT данный режим работы – это нормальная практика. Или в сфере менеджмента вводятся похожие системы. Для анализа возможности использования других систем были выделены следующие пункты, необходимые именно в предметной области зоопарка:

- возможность автоматического пересоздания задач;
- наличие нескольких ролей с возможностью ограничения действий по работе с задачами;
- возможность работы с предметной областью той или иной системы;
- составление отчетности по работе пользователей системы;
- относительная простота работы с ПО.

Для сравнения были выбраны следующие системы:

- Jira;
- Kaiten;
- Yandex Tracker.

1.3 Jira

Jira – одна из самых популярных систем для организации работы компаний. Цель создания приложения – организация процессов именно в IT сфере. ПО ориентировано на программистов и представляет следующий функционал:

- наглядное представление списка задач;
- управление командой, занимающейся развитием проекта;
- ведение проекта от начала до конца;

- отслеживание ошибок и контроль за их своевременным исправлением;
- ведение соответствующей документации.

Перед рассмотрением плюсов и минусов данного приложения нужно пояснить, что последний пункт – ведение документации осуществляется не в самой Jira, а в Confluence, который интегрирован в рассматриваемое приложение. Но, так как в Jira есть возможность перейти в Confluence, можно считать, что данный пункт является возможностью Jira.

На рисунке 1 представлен интерфейс данного программного обеспечения.

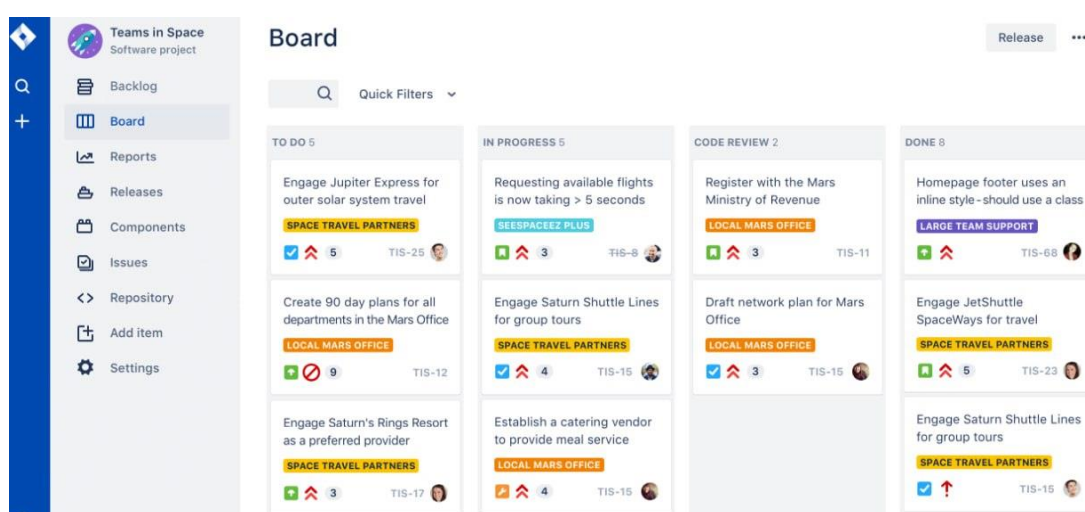


Рисунок 1 – Интерфейс Jira

Перейдя к плюсам приложения можно выделить возможности приложения. Оно покрывает основные запросы пользователей и могло бы быть адаптировано под данную предметную область.

Но в связи со сложившейся политической ситуацией Jira больше не доступно в России. Единственный вариант – это развернуть исходники приложения на сервере, что требует затрат на оборудование, так как само приложение предоставляет большие возможности, оно требует не малых затрат по памяти и процессорному времени. Также из-за своих возможностей данное приложение не просто использовать с точки зрения пользователя. В нем присутствуют не очевидные моменты, которые усложняют работу с приложением.

1.4 Kaiten

Kaiten – приложение, выступающее аналогом Jira, которое условно позиционируется как замена предыдущего рассматриваемого приложения. Если проанализировать его основные возможности, то они не сильно отличаются от Jira:

- представление списка задач;
- ведение проекта на всех этапах его развития;
- разграничение возможностей работы с системой по ролям;
- представление отчетности работы команды;
- ведение документации.

На рисунке 2 представлен интерфейс ПО.

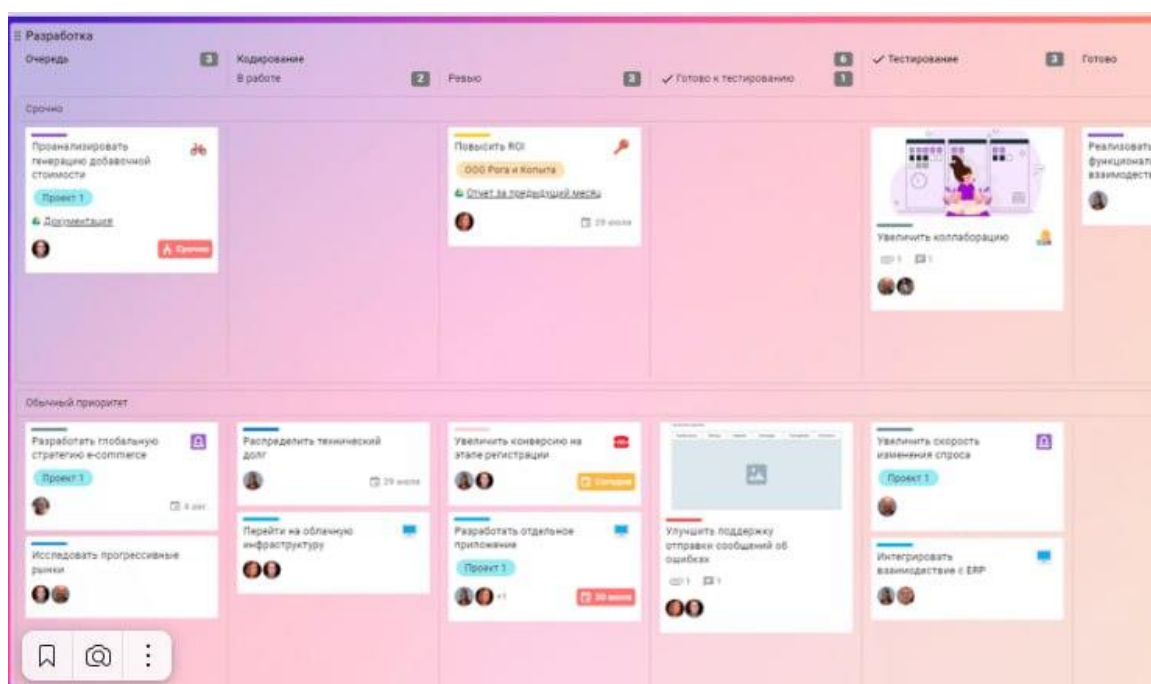


Рисунок 2 – Интерфейс Kaiten

Данный функционал полностью подходит для организации работы в зоопарке, но, чтобы им пользоваться, необходимо приобрести платный тариф, без которого нельзя пользоваться всеми возможностями системы.

1.5 Yandex Tracker

Yandex Tracker – также аналог Jira, предоставляющий похожий функционал:

- представление списка задач;
- ведение проекта на всех этапах его развития;
- разграничение возможностей работы с системой по ролям;
- представление отчетности работы команды;
- ведение документации.

На рисунке 3 представлен интерфейс приложения.

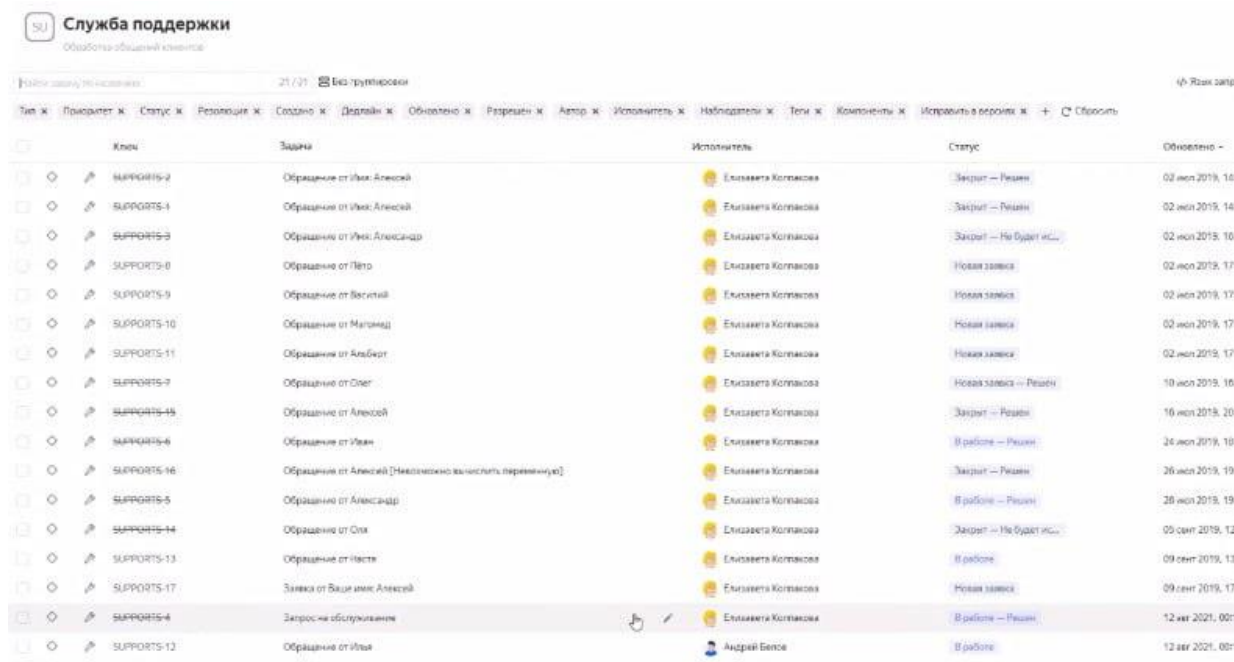


Рисунок 3 – Интерфейс Yandex Tracker

Но, аналогично Kaiten, чтобы использовать данное приложение, нужно приобрести платный тариф. Также, если опираться на отзывы пользователей, нельзя назвать данное приложение достаточно простым в использовании, что можно выделить, как отдельный минус ПО.

1.6 Обоснование для разработки программы системы мониторинга за животными в зоопарке

Первым шагом обоснования нужно предоставить сравнительный анализ конкурирующих приложений, представленный в таблице 1.

Таблица 1 – Сравнительный анализ аналогичных приложений

Название пункта	Jira	Kaiten	Yandex Tracker
Возможность автоматического пересоздания задач	Нет	Нет	Нет
Разделение по ролям	Да	Да	Да
Возможность работы с предметной областью	Да	Да	Да
Представление отчетности	Да	Да	Да
Простота использования	Нет	Да	Нет
Доступность приложения	Нет	Да	Да
Условия приобретения	–	Платное	Платное

Исходя из таблицы 1 можно сделать следующие выводы:

- большинство программ сложны в использовании;
- ни в одной из ПО нет возможности настройки автоматического пересоздания задач один раз в какой-либо промежуток времени;
- все приложения требуют затрат на их использование: будь то платные тарифы использования или высокие требования к аппаратуре для развертки приложения.

Исходя из вышеперечисленных пунктов можно прийти к выводу, что необходимость в создании удобной и бесплатной в использовании программы для системы мониторинга за животными в зоопарке существует.

Исходя из анализа аналогичных приложений было принято решение скорректировать требования к разработке:

- возможность автоматического пересоздания задач;
- наличие нескольких ролей с возможностью ограничения действий по работе с задачами;

- возможность работы с предметной областью зоопарка;
- составление отчетности по работе пользователей системы;
- относительная простота работы с ПО;
- эффективность для работы приложения на нетребовательном сервере и бесплатный функционал.

Главной отличительной особенностью данного приложения будет являться его узкая направленность к сфере зоопарка, что, с одной стороны, не предоставит возможности использовать его в других сферах, но сделает удобным и наглядным для работы в той сфере, для которой он и предназначен. Ведь, как правило, если инструмент сделан для решения большого количества задач, он является или громоздким и сложным в использовании, или просто не позволяет решать данные задачи хорошо. Исходя из данного правила сужение сферы использования является явным плюсом разрабатываемого продукта. Но, используя данную архитектуру и принципы построения ПО, можно с минимальными усилиями перевести данное приложение из одной предметной области в другую.

2 Проектирование программы

При разработке приложения было принято решение разделить его на три основные части:

- frontend – клиентская часть приложения, т. е. интерфейс;
- backend – серверная часть приложения;
- база данных (далее – БД).

Для клиентской части было принято решение использовать язык JavaScript и фреймворк React. Для серверной части – язык Java и фреймворк Spring Boot. СУБД – PostgreSQL.

Так как серверная и клиентская части разделены, для их связи использовался протокол http, соответственно клиентская часть отправляет запросы к серверной, получает ответы – данные из БД, к которой обращается backend. Данная архитектура соответствует классическому WEB-приложению. Также было принято решение для написания серверной части использовать REST архитектуру. Данная архитектура накладывает определенные правила на написание эндпоинтов – методов контроллеров, которые реализуют http-запросы. Данные технологии, используемые как в клиентской, так и в серверной частях приложения, хорошо подходят для написания WEB-приложений, поэтому они и были выбраны. Также, для работы с БД была выбрана идея ORM (Object-relational mapping). По сути – это прослойка между БД и ООП, которая позволяет связать данные с объектами классов-сущностей, повторяющих структуру таблиц БД. Если сказать более простыми словами, то класс-сущность должна полностью повторять таблицу БД, тогда каждая строка данной таблицы – это объект соответствующего класса-сущности. СУБД PostgreSQL – одна из наиболее популярных СУБД. В данном случае требуется хранить и получать различную информацию, для чего данная СУБД и создана. Также PostgreSQL – реляционная СУБД, соответственно можно создать таблицы, связать их внешними ключами и т. д. Полная схема представлена далее.

Для разработки данного приложения было принято решение использовать IDE IntelliJ IDEA, которая полностью удовлетворяет требованиям современной IDE:

- редактор кода;
- компилятор;
- отладчик;
- средство автоподстановки и генерации кода;
- удобный интерфейс.

Так как серверная часть приложения написана на языке Java, который является полностью объектно-ориентированным, данная парадигма используется в полной мере.

Если вернуться к архитектуре приложения в целом, которая описывалась в пункте 1.1, то ее можно дополнить, основываясь на предыдущих суждениях:

- клиентская часть приложения отправляет http-запрос в эндпоинт контроллера;
- эндпоинт контроллера, не преобразуя полученные данные, отправляет их в соответствующий метод сервисного слоя;
- в методах сервисного слоя реализована вся бизнес-логика приложения;
- если по логике того или иного бизнес-процесса необходимо совершить запрос к БД, то вызывается метод слоя репозитория, который формирует и отправляет SQL запрос в СУБД PostgreSQL, возвращая один или список классов-сущностей, в зависимости от самого запроса;
- далее, цепочка вызовов переворачивается, т. е. репозитории возвращают результат сервисам, сервисы – контроллерам, а контроллеры – клиентской части приложения.

Данная архитектура – это классическая архитектура WEB-приложения. При такой архитектуре не важно, в каком виде представлена клиентская часть – будь то WEB-страницы, мобильное приложение, desktop-приложение или другое

серверное приложение, которое интегрируется с данным. В этом заключается основной плюс данной архитектуры, из-за чего она и выбрана.

2.1 Анализ функциональных требований к системе

Необходимо разработать систему мониторинга за животными в зоопарке. Данная система должна включать в себя следующий функционал:

- хранение профилей (карточке) животных с информацией о животном: его параметры (рост, вес, хищное или травоядное и т. д.), особенности содержания, внешние особенности (повреждения), состояние здоровья и статус;
- управление связями сотрудников и животных: связь выбранного ответственного сотрудника с группой животных. Выбранному сотруднику должны приходить уведомления и напоминания, касающиеся только его зоны ответственности;
- напоминание работнику о необходимости выполнить определенное действие: покормить, помыть, вычесать, совершить медосмотр и т. п.;
- получение отчетов от сотрудников зоопарка с информацией о проделанных действиях или изменениях параметров животных;
- мониторинг состояния животного;
- учет оборота животных в зоопарке (завоз новых, продажа имеющихся);
- формирование выставочных программ в зоопарке на основе анализа состояния животных;
- формирование списков для планового ветеринарного осмотра;
- ведение статистики;
- личный кабинет сотрудника, ветеринара и администратора;
- Telegram бот для уведомления о появившихся задачах.

2.2 Проектирование системы

2.2.1 Анализ прецедентов

На основе функциональных требований были разработаны диаграммы вариантов использования для каждой роли приложения.

На рисунке 4 представлена диаграмма вариантов использования для роли «Администратор».

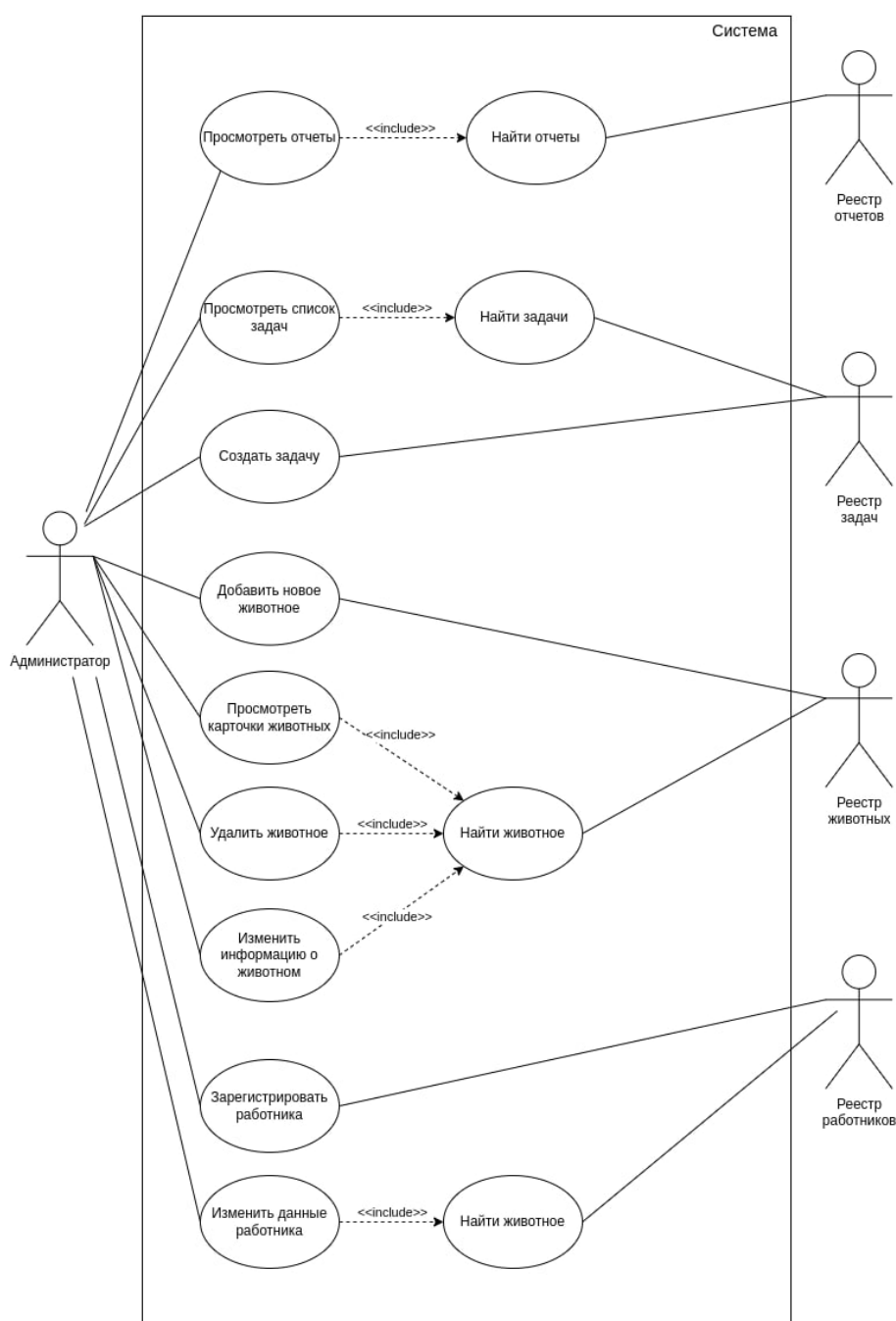


Рисунок 4 – Диаграмма вариантов использования для роли «Администратор»

На рисунке 5 представлена диаграмма вариантов использования для роли «Ветеринар».

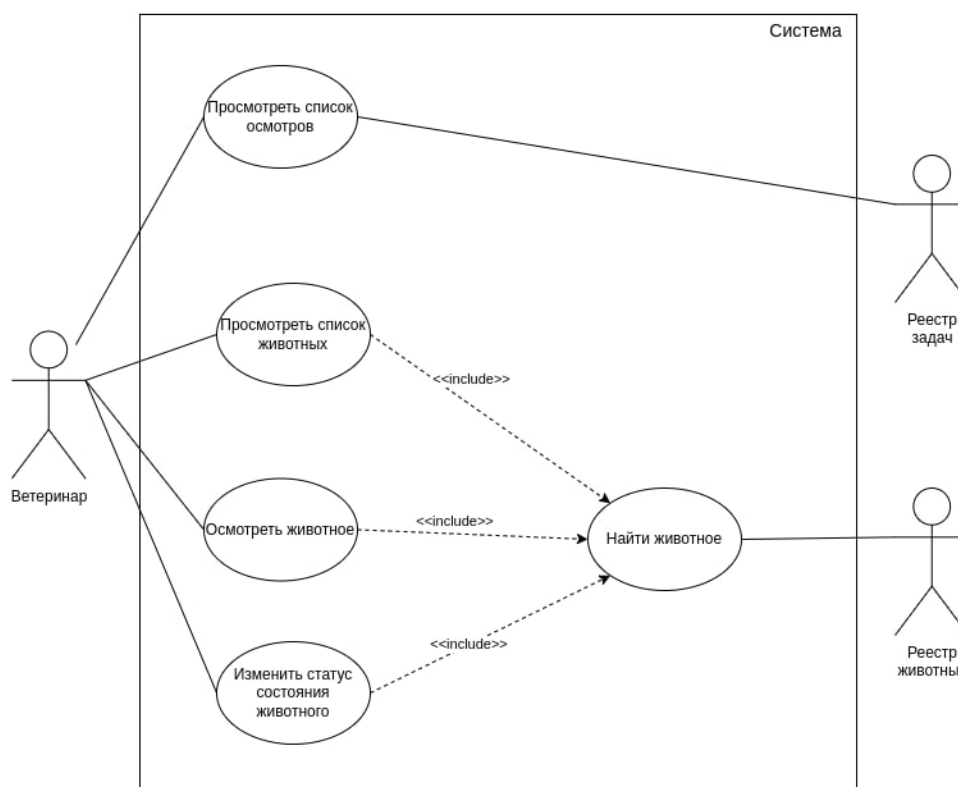


Рисунок 5 – Диаграмма вариантов использования для роли «Ветеринар»

На рисунке 6 представлена диаграмма вариантов использования для роли «Сотрудник зоопарка».

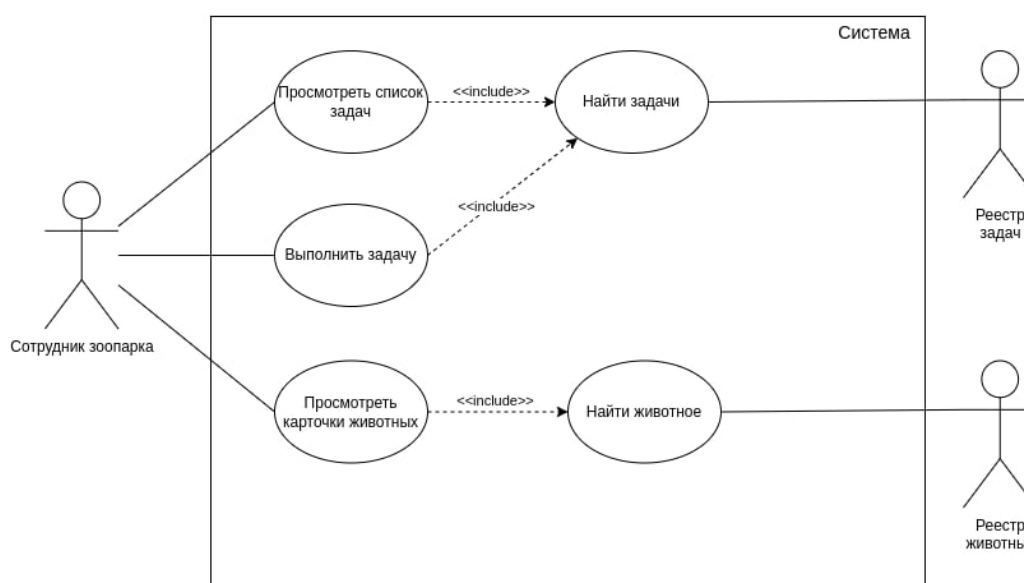


Рисунок 6 – Диаграмма вариантов использования для роли «Сотрудник зоопарка»

В таблице 2 приведена спецификация варианта использования «Регистрация работников».

Таблица 2 – Спецификация прецедента «Регистрация работников»

№	Содержание
1	Прецедент: Регистрация работников
2	ID:1
3	Краткое описание: Занесение информации о новом работнике в реестр пользователей. Информация о работнике: ФИО, логин, пароль, возраст, роль в системе.
4	Главные актеры: Администратор, реестр пользователей
5	Второстепенные актеры: –
6	Предусловия: 1. Администратор вошел в систему.
7	Основной поток: 1 - Прецедент начинается, когда Администратор выбирает опцию «зарегистрировать сотрудника». 2 - Система запрашивает у администратора данные пользователя. 3 - Администратор вводит запрашиваемые данные. 4 - Система проверяет уникальность логина. 5 - Если логин сотрудника уже существует, то 5.1 - Система сообщает, что пользователь с данным логином уже зарегистрирован. 6 - Иначе 6.1 - Система добавляет нового сотрудника в реестр. 6.2 - Система выводит на экран карточку нового сотрудника.
8	Постусловия: 1. Работник добавлен в реестр пользователей
9	Альтернативные потоки: Нет

В таблице 3 приведена спецификация варианта использования «Создание выставок».

Таблица 3 – Спецификация прецедента «Просмотр отчетов»

№	Содержание
1	2
1	Прецедент: Просмотр отчетов
2	ID:2
3	Краткое описание: Администратор просматривает отчет
4	Главные актеры: Администратор, реестр задач
5	Второстепенные актеры: –
6	Предусловия: 1.Администратор авторизован

Продолжение таблицы 3

1	2
7	<p>Основной поток:</p> <ul style="list-style-type: none"> 1 - Прецедент начинается, когда Администратор выбирает опцию «Просмотреть отчеты». 2 - Администратор выбирает пользователя, отчет которого хочет увидеть <ul style="list-style-type: none"> 2.1 - Если сотрудник выбран, то <ul style="list-style-type: none"> 2.1.1 - Система формирует отчеты работника. 2.2 - Иначе <ul style="list-style-type: none"> 2.2.1 - Система формирует отчеты всех сотрудников. <p>Точка расширения: просмотр отчетов за определенную дату.</p> <ul style="list-style-type: none"> 3 - Система запрашивает дату или промежуток времени. 4 - Администратор задает дату или промежуток времени, за который хочет получить отчет. 5 - Система возвращает отчеты за определенный период времени.
8	<p>Постусловия:</p> <p>Администратор видит отчеты сотрудников с выбранными параметрами</p>
9	<p>Альтернативные потоки:</p> <p>Нет.</p>

В таблице 4 приведена спецификация варианта использования «Создание задач».

Таблица 4 – Спецификация прецедента «Создание задач»

№	Содержание
1	Прецедент: Создание задач
2	ID:2
3	<p>Краткое описание:</p> <p>Занесение информации о новой задаче в реестр задач.</p> <p>Информация о задаче: тип задачи, крайний срок выполнения задачи, тип повторения задачи, животное, для которого создается задача, комментарий к задаче.</p>
4	<p>Главные актеры:</p> <p>Администратор, реестр задач</p>
5	Второстепенные актеры: –
6	<p>Предусловия:</p> <p>1. Администратор вошел в систему.</p>
7	<p>Основной поток:</p> <ul style="list-style-type: none"> 1 - Прецедент начинается, когда Администратор выбирает опцию «создать задачу». 2 - Система запрашивает у администратора данные о задаче. 3 - Администратор вводит запрашиваемые данные. 4 - Система проверяет корректность введенных полей. 5 - Если данные введены некорректно, то <ul style="list-style-type: none"> 5.1 - Система сообщает об ошибке. 6 - Иначе <ul style="list-style-type: none"> 6.1 - Система создает задачу. 6.2 - Система уведомляет работника, за которым закреплено животное о создании задачи. 6.3 - Система выводит на экран информацию о задаче.
8	<p>Постусловия:</p> <p>Задача добавлена в реестр задач.</p>
9	<p>Альтернативные потоки:</p> <p>Нет.</p>

В ходе анализа функциональных требований были выделены и описаны основные варианты использования системы.

2.2.3 Проектирование модели предметной области

В рамках анализа предметной области были выделены следующие сущности предметной области «Система мониторинга за животными в зоопарке», представленные на рисунке 7.

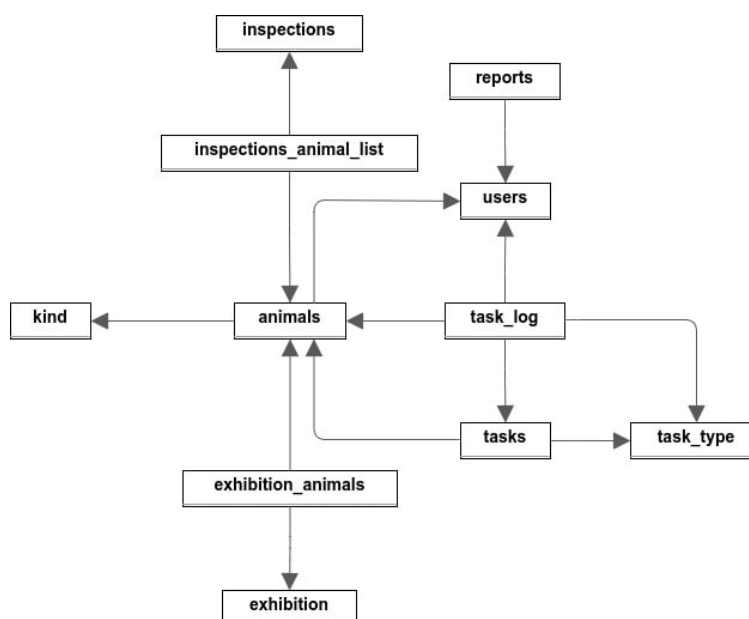


Рисунок 7 – Модель предметной области

Описание сущностей предметной области представлено в таблице 5.

Таблица 5 – Описание сущностей

Название класса	Спецификация класса
Inspection	сущность, описывающая осмотр;
Animal	сущность, описывающая животных
Kind	сущность, описывающая вид животного
Exhibition	сущность, описывающая выставку
Task_type	сущность, описывающая тип задачи
Task	сущность, описывающая задачу
User	сущность, описывающая пользователя
TaskLog	сущность, описывающая логирование выполненных задач;
Report	сущность, описывающая отчет.

Данная модель предметной области соответствует модели БД.

2.2.4 Проектирование структуры БД

На основе модели сущностей была разработана схема БД, представленная на рисунке 8.

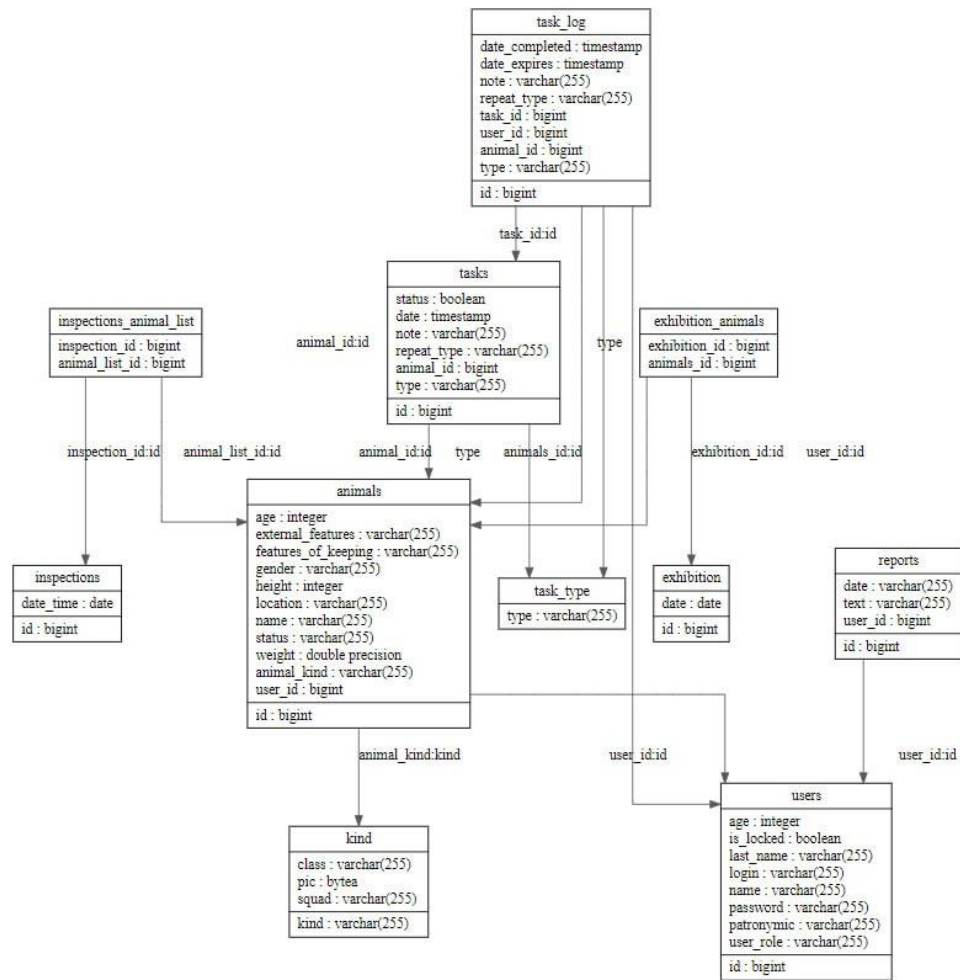


Рисунок 8 – Схема БД

Назначение таблиц, представленных на схеме БД представлено в таблице 6.

Таблица 6 – Описание таблиц БД

Название сущности	Идентификатор сущности	Назначение сущности
1	2	3
Сотрудник	users	Содержит сведения о сотрудниках
Животное	animals	Содержит сведения о животных
Выставка	exhibition	Содержит сведения о выставках
Задача	tasks	Содержит сведения о задачах
Отчет	reports	Содержит сведения об отчетах

Продолжение таблицы 6

1	2	3
Автоотчет	task_log	Содержит сведения о выполненных задачах для формирования автоотчета
Статус животного	animal_status	Содержит информацию о статусах животных
Вид животного	kind	Содержит информацию о видах животных
Тип повторения задач	task_type	Содержит информацию о типах повторения задач
Осмотр	inspection	Содержит информацию об осмотрах

Подробное описание таблиц базы данных представлено в приложении А.

2.2.5 Проектирование обобщенной архитектуры системы

На основе архитектуры, выделенной в пункте 2 были выделены компоненты, показанные на рисунке 9.

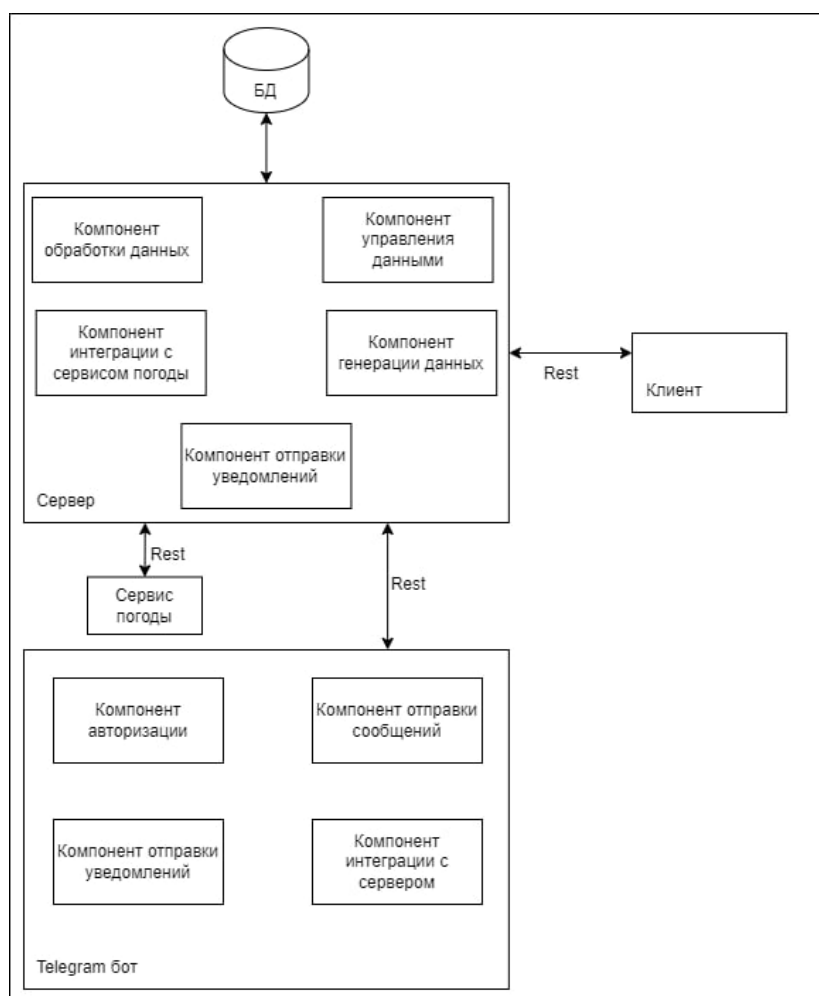


Рисунок 9 – Схема обобщенной архитектуры системы

Один из основных модулей архитектуры – база данных.

Главный модуль системы – разрабатываемый сервер. Он включает в себя:

- компонент обработки данных, получаемых от клиента;
- компонент управления данными или же слой подключения к БД;
- компонент интеграции с внешним сервисом погоды, который реализует взаимодействие по Rest API для получения данных о прогнозе погоды;
- компонент генерации данных, позволяющий осуществить повторное создание моделей в указанный промежуток времени;
- компонент отправки уведомлений, осуществляющий передачу данных.

Сервис погоды – внешний сервис с открытым API, позволяющий получить прогноз погоды.

Компонент, позволяющий взаимодействовать с сервером по средству пользовательского интерфейса на схеме указан как клиент. Клиент и сервер взаимодействуют по Rest API.

Альтернативным способом получения информации с ограниченным функционалом является Telegram бот. Он служит для уведомления работника зоопарка о появившихся задачах. Telegram бот включает в себя следующие компоненты:

- компонент интеграции с сервером связывает Telegram бота, выступающего в роли клиента, с сервером;
- компонент авторизации позволяет связать сессию пользователя сервера с Telegram ботом;
- компонент отправки сообщений осуществляет передачу данных пользователю;
- компонент отправки уведомлений связывает полученное с сервера уведомление с пользователем Telegram бота.

Таким образом, разработанная архитектура системы соответствует общим концепциям WEB-приложений.

2.3 Разработка серверного программного обеспечения

Для разработки серверной части приложения была выбрана портов и адаптеров.

Шестиугольная архитектура, или архитектура портов и адаптеров, является архитектурным шаблоном, используемым в разработке программного обеспечения. Она направлена на создание слабо связанных компонентов приложений, которые могут быть легко подключены к их программной среде с помощью портов и адаптеров. Это делает компоненты взаимозаменяемыми на любом уровне и облегчает автоматизацию тестирования.

Шестиугольная архитектура делит систему на несколько слабо связанных взаимозаменяемых компонентов, таких как ядро приложения, база данных, пользовательский интерфейс, тестовые сценарии и интерфейсы с другими системами. Этот подход является альтернативой традиционной многоуровневой архитектуре [3].

Каждый компонент соединен с другими через ряд открытых «портов». Связь через эти порты осуществляется по заданному протоколу в зависимости от их назначения. Порты и протоколы определяют абстрактный API, который может быть реализован любыми подходящими техническими средствами (например, вызов метода на объектно-ориентированном языке, удаленные вызовы процедур или веб-службы).

В разработанном приложении используется три компонента: контроллерный слой, серверный слой, слой для общения с базой данных.

Далее, в таблицах 7 – 9 будут описаны компоненты каждого слоя.

Все классы контроллеров отвечают за сетевое взаимодействие (далее эндпоинт) для управления определенной сущностью и делегируют логику сервисному слою. В таблице 7 описаны классы контроллеров.

Таблица 7 – Описание классов контроллеров

Название	Назначение
AuthenticationController	Содержит эндпоинты для аутентификации
AnimalKindController	Содержит эндпоинты для получения информации о видах животных
AnimalController	Содержит эндпоинты для получения информации о животных
InspectionController	Содержит эндпоинты для получения информации об осмотрах
ExhibitionController	Содержит эндпоинты для получения информации о выставках
TaskTypeController	Содержит эндпоинты для получения информации о типах задач
TaskLogController	Содержит эндпоинты для получения информации о логах задач
TaskController	Содержит эндпоинты для получения информации о задачах
UserController	Содержит эндпоинты для получения информации о пользователях
WeatherController	Содержит эндпоинты для получения информации о прогнозе погоды

Все классы сервисов отвечают за бизнес-логику приложения. Для обращения к базе данных они используют слой для общения с БД. В таблице 8 представлено описание классов сервисного слоя.

Таблица 8 – Описание классов сервисного слоя

Название	Назначение
ReportService	Содержит бизнес-логику отчетов
AnimalKindService	Содержит бизнес-логику видов животных
AnimalService	Содержит бизнес-логику животных
InspectionService	Содержит бизнес-логику осмотров
ExhibitionService	Содержит бизнес-логику выставок
TaskTypeService	Содержит бизнес-логику типов задач
TaskLogService	Содержит бизнес-логику логов задач
TaskService	Содержит бизнес-логику задач
UserService	Содержит бизнес-логику пользователей
WeatherService	Содержит бизнес-логику прогнозов погоды
ShedulerService	Содержит методы, обеспечивающие повторение задач

Все классы слоя базы данных используются для доступа к БД. Каждый класс слоя используется для конкретной модели предметной области. В таблице 9 представлено описание классов слоя базы данных.

Таблица 9 – Описание классов слоя базы данных

Название	Назначение
ReportRepository	Обеспечивает связь с таблицей отчетов
AnimalKindRepository	Обеспечивает связь с таблицей видов животных
AnimalRepository	Обеспечивает связь с таблицей животных
InspectionRepository	Обеспечивает связь с таблицей осмотров
ExhibitionRepository	Обеспечивает связь с таблицей выставок
TaskTypeRepository	Обеспечивает связь с таблицей типов задач
TaskLogRepository	Обеспечивает связь с таблицей логов задач
TaskRepository	Обеспечивает связь с таблицей задач
UserRepository	Обеспечивает связь с таблицей пользователей

Данные слои обеспечивают передачу, обработку и изменения данных, переданных клиентом в базу данных и наоборот.

На рисунке 10 приведен пример компоновки классов для работы с информацией о животных.

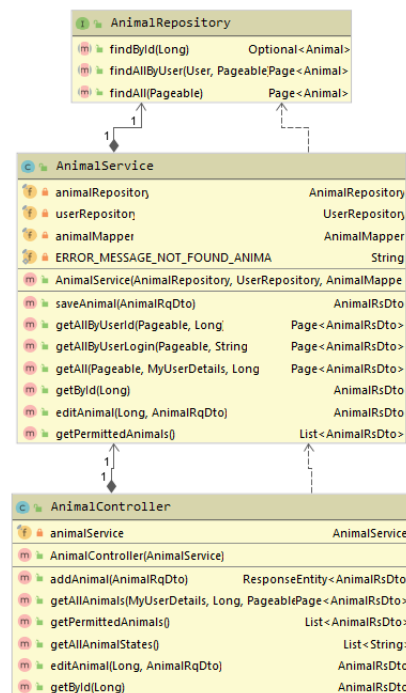


Рисунок 10 – Компоновка классов для обработки информации о животных

Пример реализации класса AnimalService.java представлен в приложении В.

2.4 Разработка клиентского программного приложения

2.4.1 Требования к пользовательскому интерфейсу

Общие требования к пользовательскому интерфейсу были сформированы следующим образом:

- интерфейс системы должен поддерживать русский язык;
- интерфейс системы должен быть спроектирован с учетом ролевой модели системы уровней доступа пользователей;
- интерфейс системы должен обеспечивать наглядное, интуитивно понятное представление структуры размещенной информации, быстрый и логичный переход к соответствующим разделам системы;
- навигационные элементы интерфейса системы должны обеспечивать однозначное понимание пользователем их смысла и обеспечивать навигацию по всем доступным пользователю разделам системы и отображать соответствующую информацию;
- интерфейс системы должен позволять решать задачи пользователя наиболее быстрым, простым и удобным из возможных способов;
- интерфейс системы должен быть рассчитан на пользователей, не имеющих специальных технических знаний и навыков в области компьютерной техники, и быть легко осваиваем ими;
- интерфейс системы должен учитывать контекст использования: где, при каких обстоятельствах, с помощью каких устройств пользователь будет взаимодействовать с системой;
- в целом дизайн и удобство интерфейса должны быть на уровне ожиданий современного пользователя, имеющего опыт работы с лучшими образцами, подобных сервисов и восприниматься им как комфортная, удобная и приятная рабочая среда;
- интерфейс должен иметь отдельную страничку для отображения карточек с именами животных, а также предусматривал вывод и редактирование основной информации питомцев;

- также интерфейс должен иметь страничку для создания, редактирования и просмотра сотрудников;
- интерфейс должен иметь странички для создания и настройки выставок и задач для сотрудников;
- интерфейс должен уведомлять пользователя о грядущих задачах;
- для авторизации интерфейс должен предоставлять отдельную страничку.

2.4.2 Описание архитектуры пользовательского интерфейса

При реализации клиентской части приложения была применена следующая архитектура.

Вся ее суть основана на том, что страницы получаются из маленьких частей, которые в свою очередь объединяются в более крупные части.

Например, в приложении есть кнопки, это атомы. Они объединяются в молекулы, если проводить аналогию с кнопками, то молекулы – блок кнопок. Более крупной частью является организм, далее шаблон страницы и конечный результатом агрегации является сама страница со всеми элементами.

Используя данную архитектуру можно добиться следующего:

- пропадает необходимость в написании стандартных html страниц – все страницы приложения формируются из небольших фрагментов, данный подход называется динамической страницей;
- при формировании страниц пропадает риск ошибки разработчика, связанный с применением стилей – компонент создается лишь один раз и используется везде, соответственно и стили данного компонента будут едины для всех его использований;
- упрощается тестирование приложения, так как, протестировав каждый компонент системы, можно утверждать, что все его использования будут корректными.

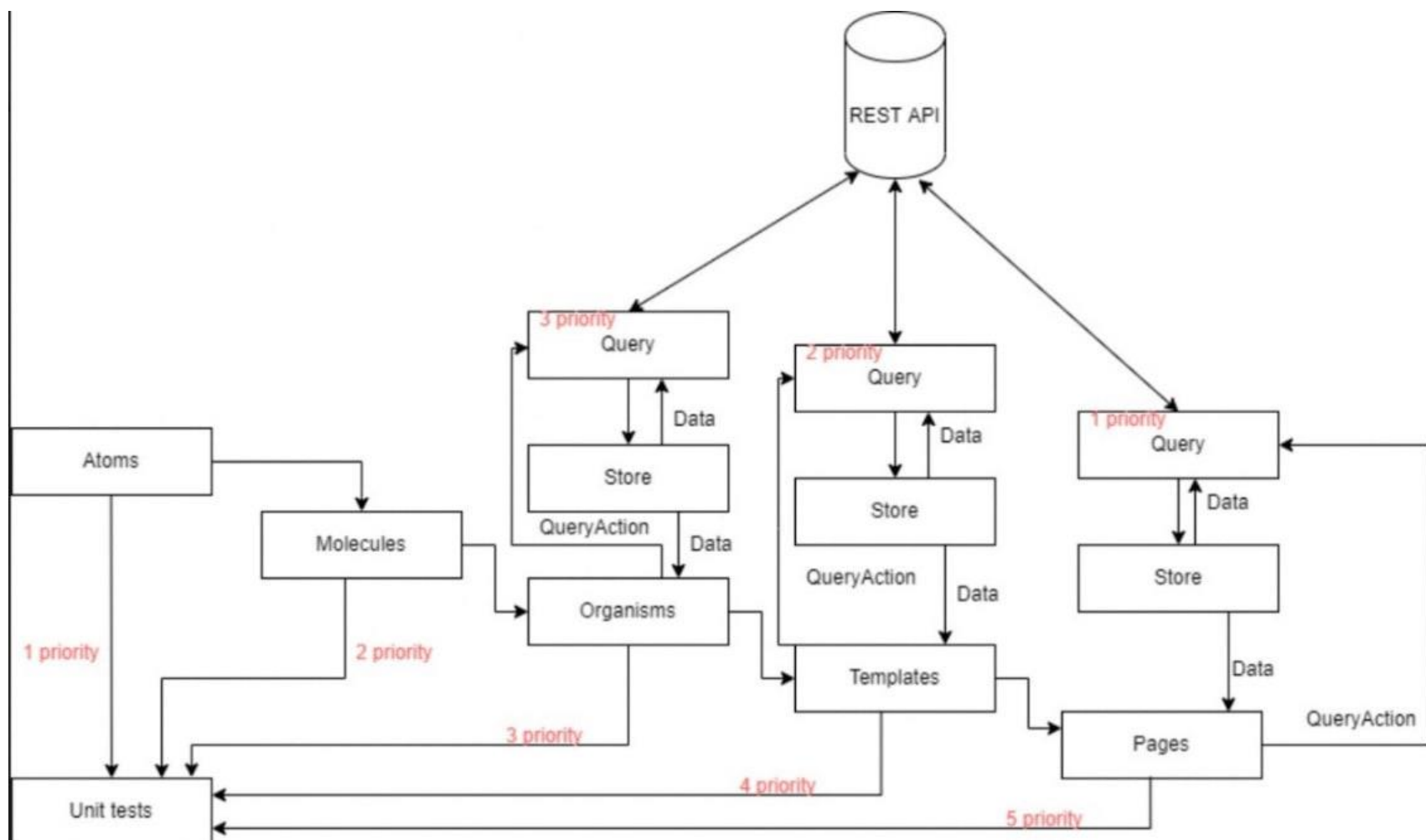
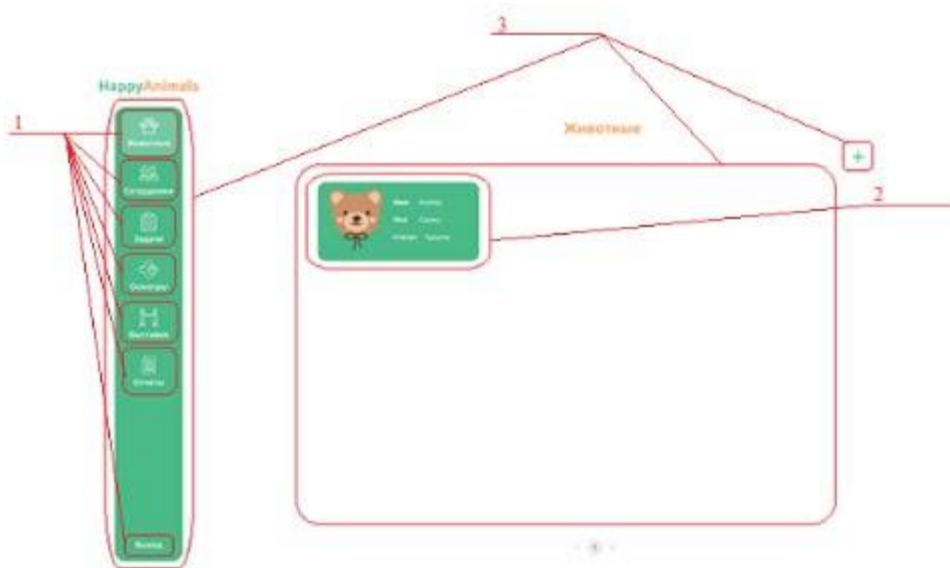


Рисунок 11 – Архитектура клиентского ПО

Пример приведен на рисунке 12, где показано, что молекулы состоят из атомов, организмы из молекул, а отображаемая страница – набор перечисленных элементов. А именно, атомы представлены в виде кнопок меню, которое является организмом. При этом допустимо, что маленькие части, как атомы, сразу могут объединяться в организм [4]. Молекулами на рисунке являются карточки с описанием животного, а организмами – меню, набор карточек животных и добавление нового животного.



1 – атом;

2 – молекула;

3 – организм.

Рисунок 12 – Пример агрегации элементов страницы животных

Почти у всех частей имеется собственное хранилище, в которое складывается информация об ответах на запросы с серверной части приложения, после они отображаются пользователю. Запросы могут отправляться разными частями страничек.

Различные страницы могут включать в себя одинаковые под части, например, организмы или молекулы. Переход между страничками достигается с помощью библиотеки React Route. Он помогает настроить пути для различных страничек.

Также для получения уведомлений и отображения их пользователю использовался протокол WebSocket.

Общение с серверной частью было реализовано с помощью Rest API.

Клиентская часть была разработана на языке JavaScript с использованием Фреймворка React [5].

Далее приведены иерархии страниц. В данном программном обеспечении реализованы три ролевые модели: администратор, работник зоопарка и ветеринар. В зависимости от роли меняется иерархия страниц.

На рисунке 13 приведена иерархия страниц для роли администратор.

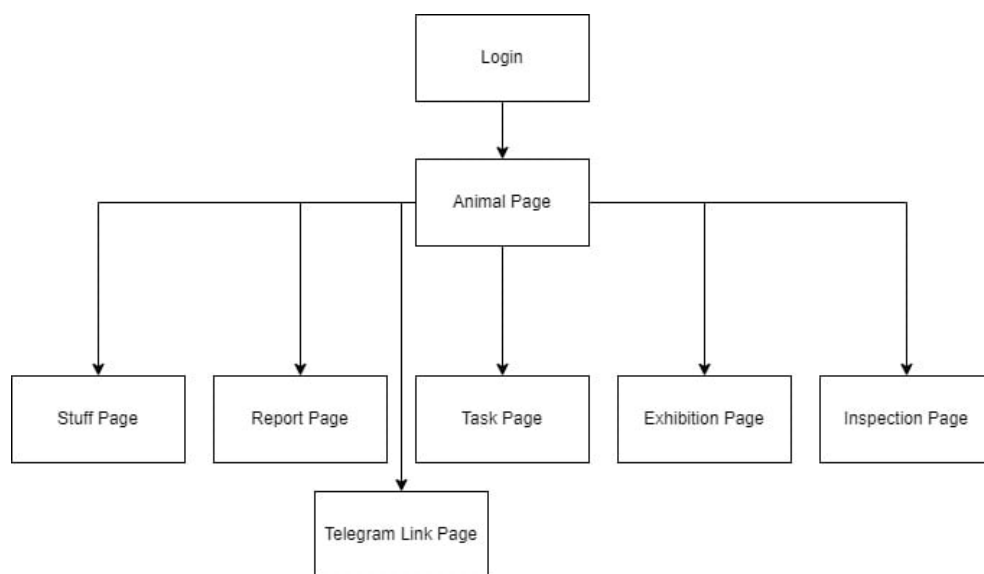


Рисунок 13 – Иерархия страниц для роли администратора

На рисунке 14 приведена иерархия страниц для роли работника зоопарка.

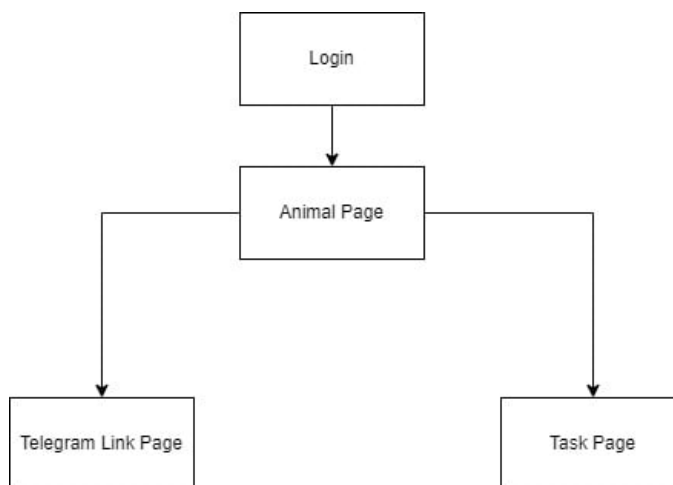


Рисунок 14 – Иерархия страниц для роли работника зоопарка

На рисунке 15 приведена иерархия страниц для роли ветеринара.

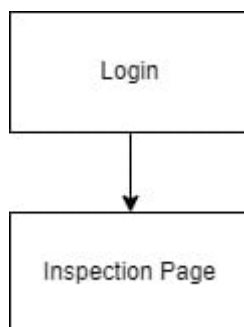


Рисунок 15 – Иерархия страниц для роли ветеринара

В приложении Б показаны примеры страниц системы.

В листинге 1 приведен код атома для компонента MenuItem. Данный компонент – это кнопка для перехода на страницы с информацией о животных, пользователях, задачах и т. д. (на рисунке 12 они находятся в левой колонке на зеленом фоне).

Листинг 1 – Описание атома для компонента MenuItem на языке JavaScript

```

import styles from './menuItem.module.css';
type MenuItemProps = {
  text?: string;
  img?: string;
  active?: boolean;
};
export function MenuItem({ text, img, active = false }:
MenuItemProps) {
  return (
    <div
      className={
        active ? `${styles.wrapper} ${styles.active}` :
styles.wrapper
      }
    >
      {img && <img src={img} alt="img"
className={styles.img} />}
      <div className={` ${styles.text}
menuItemTitle`}>{text}</div>
    </div>
  );
}
  
```

В листинге 2 приведен код для описываемого атома на языке разметки css.

Листинг 2 – Описание стилей атома для компонента MenuItem с помощью css

```
.wrapper {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  padding: 10px;  
  margin: 10px;  
  border-radius: 17px;  
  cursor: pointer;  
}  
.active {  
  background: rgba(255, 255, 255, 0.23);  
}  
.img {  
  width: 40px;  
}
```

Разделение на разметку и стили – это основа написания клиентской части любых приложений. Выбранная архитектура не отменяет данного правила.

3 Тестирование программного обеспечения

3.1 Выбор методологии тестирования

Существует два основных вида тестирования – сценарное (scripted) и исследовательское (exploratory).

Основная особенность сценарного тестирования заключается в делении задачи на этапы (подготовка, выполнение, завершение и пр.) и затем действие согласно этим этапам. То есть этот подход можно охарактеризовать как «я думаю над этим заранее и затем выполняю».

Исследовательское тестирование подразумевает совершенно иной подход – разработка процесса тестирования происходит непосредственно во время работы. Тестирование начинается с некой важной точки, в процессе появляются другие важные точки тестирования, и каждый раз специалист решает, какая из них наиболее важна и в каком направлении двигаться дальше. То есть продумывание и запуск тестов происходят в постоянном взаимодействии [6].

В данном случае был выбран сценарный подход. Так как он наиболее эффективен для проверки работоспособности приложения именно разработчиком, который понимает логику его работы, потому что он может корректно разработать тестовые сценарии.

В сценарном подходе выделяют следующие виды:

- тестирование чёрного ящика – метод тестирования ПО, который не предполагает доступа (полного или частичного) к системе и основывается на работе исключительно с внешним интерфейсом тестируемой системы;
- тестирование белого ящика – метод тестирования ПО, который предполагает полный доступ к коду проекта;
- модульное тестирование – проводится для тестирования какого-либо одного логически выделенного и изолированного элемента (модуля) системы в коде, проводится самими разработчиками, так как предполагает полный доступ к коду;

- интеграционное тестирование – тестирование, направленное на проверку корректности взаимодействия нескольких модулей, объединенных в единое целое;
- системное тестирование – процесс тестирования системы, на котором проводится не только функциональное тестирование, но и оценка характеристик качества системы – ее устойчивости, надежности, безопасности и производительности;
- функциональное тестирование – направлено на проверку корректности работы функциональности приложения [7].

В данной работе были выбраны следующие виды тестирования: тестирование черного ящика, модульное тестирование, функциональное тестирование.

3.2 Тестирование черного ящика модуля «Регистрация сотрудника»

3.2.1 Классы эквивалентности

В таблице 10 представлены классы эквивалентности поля «Роль» для варианта использования «Регистрация работника».

Таблица 10 – Классы эквивалентов для поля «Роль»

№	Входные данные	Правильность класса
1	Администратор, Работник зоопарка, Ветеринар	+
2	a..я, A..Я	-
3	a..z, A..Z	-
4	(-9999...0]	-
5	(9999...+∞)	-
6	∅	-
7	Иные символы	-

В таблице 11 представлены тесты для поля «Роль» в соответствии с классами эквивалентности.

Таблица 11 – Тесты для поля «Роль»

Входные данные	Предполагаемые выходные данные	ЭК	Результат
Администратор	Ввод данных	1	Данные успешно добавлены в базу
Работник зоопарка	Ввод данных	2	Данные успешно добавлены в базу
Ветеринар	Ввод данных	3	Данные успешно добавлены в базу
123	Невозможно ввести данные в поле	4	Данные не вводятся в поле
абв	Невозможно ввести данные в поле	5	Данные не вводятся в поле
abc	Невозможно ввести данные в поле	6	Данные не вводятся в поле
-1	Сообщение об ошибке	7	Данные не вводятся в поле
∅	Невозможно ввести данные в поле	8	Данные не вводятся в поле
*#	Невозможно ввести данные в поле	9	Данные не вводятся в поле

Тестирование пройдено успешно.

3.2.2 Функциональная диаграмма и таблица решений

Метод причинно-следственных связей. Функциональная диаграмма.

Только администратор системы может зарегистрировать нового пользователя.

В таблице 12 представлены причины и следствия различных исходов варианта использования «Регистрация работника».

Таблица 12 – Причины и следствия возможных исходов варианта использования «Регистрация работника»

Условное обозначение причины	Название причины	Условное обозначение следствия	Название следствия
1	2	3	4
П1	Роль регистрирующего сотрудника – Администратор	С1	Сообщение: «Не достаточно прав на совершение данного действия»

Продолжение таблицы 12

1	2	3	4
П2	Не пустая строка	С2	Сообщение: «Роль должна быть выбрана из подготовленных»
П3	Выбрана из подготовленных ролей: Администратор, Работник зоопарка, Ветеринар	С3	Сообщение: «Поле не может быть пустым»
		С4	Сообщение: «Регистрация прошла успешно»

На рисунке 16 представлена функциональная диаграмма, составленная по таблице 12.

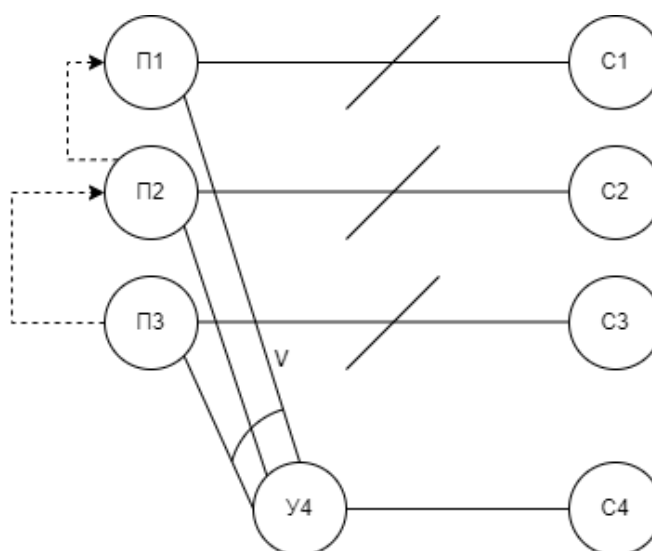


Рисунок 16 – Функциональная диаграмма

В таблице 13 представлены решения по функциональной диаграмме (рисунок 16).

Таблица 13 – Таблица решений по функциональной диаграмме

	1	2	3	4	6	5	7	8
П1	0	1	0	1	0	1	0	1
П2	–	0	–	1	–	0	–	1
П3	–	–	–	0	–	–	–	1
С1	1	0	1	0	1	0	1	0
С2	–	1	–	0	–	1	–	0
С3	–	–	–	1	–	–	–	0
С4	–	–	–	0	–	–	–	1

В таблице 14 представлено тестирование по функциональной диаграмме (рисунок 16).

Таблица 14 – Тестирование по функциональной диаграмме

Набор данных	Результат
Регистрирующий сотрудник – не администратор	Невозможно ввести данные в поле
Регистрирующий сотрудник – администратор, роль – «абв»	Сообщение об ошибке
Регистрирующий сотрудник – администратор	Ввод данных
Регистрирующий сотрудник – администратор, роль – «»	Сообщение об ошибке

На рисунке 17 представлена диаграмма s&t для варианта использования «Регистрация работника».

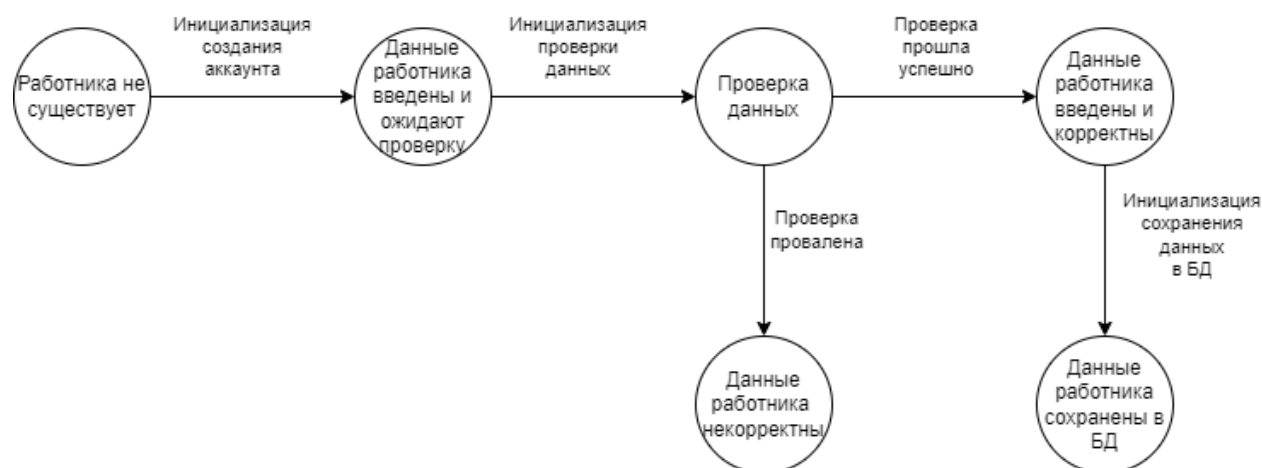


Рисунок 17 – Диаграмма s&t

Тестирование пройдено успешно.

3.3 Модульное тестирование модуля «Регистрация работника»

Для проверки работоспособности приложения было совершено Unit-тестирование всех вариантов использования. Суммарно проведено более 100 тестов. Пример тестирования модуля «Регистрация работника» приведен в таблице 15.

Таблица 15 – Unit-тестирование

Вариант использования/название теста	Состав теста	Результат
Регистрация работника (добавление работника в пустую таблицу)	Имя: string Фамилия: string Отчество: string Возраст: 1 Роль: ADMIN Логин: string Пароль: string	Тест пройден (рисунок 18)
Регистрация работника (добавление работника в таблицу с одной записью)	Имя: string1 Фамилия: string1 Отчество: string1 Возраст: 1 Роль: ADMIN Логин: string1 Пароль: string1	Тест пройден
Регистрация работника (добавление работника в таблицу, где N записей)	Имя: string3 Фамилия: string3 Отчество: string3 Возраст: 1 Роль: ADMIN Логин: string3 Пароль: string3	Тест пройден (рисунок 19)
Регистрация работника (добавление работника, пустое тело запроса)		Тест пройден
Регистрация работника (добавление работника, некорректные данные)	Имя: string3 Фамилия: string3 Отчество: string3 Возраст: 1 Роль: ADMIN Логин: string3 Пароль: string3	Тест пройден (рисунок 20)

id	age	is_locked	last_name	login	name	password	patronymic	user_role	
1	2	1	• true	string	string	string	\$2a\$12\$rbQGmNkN0cD/iRWE78qYZ.VzsefbFLF06dVtC.Q1...	string	ADMIN

Рисунок 18 – Добавление работника в пустую таблицу

	id	age	is_locked	last_name	login	name	password	patronymic	user_role
1	2	1	• true	string	string	string	\$2a\$12\$rbQGmNkN0cD/iRWE78qYZ.VzsefbFLF06dVtC.Q1...	string	ADMIN
2	3	1	• true	string1	string1	string1	\$2a\$12\$RqXU2upsootXE6ZCfT3nIenvxikR0WwqR0P4p52U...	string1	ADMIN
3	4	1	• true	string3	string3	string3	\$2a\$12\$TiRLVZssZZbXA/AN4MiFbeYsrcR.xvzbza/EMFfDm...	string3	ADMIN

Рисунок 19 – Добавление работника в таблицу с одной записью

✓ Test Results	232 ms
✓ UserControllerTest	232 ms
✓ addNotValidUser()	232 ms

Рисунок 20 – Добавление работника, некорректные данные

Для тестирования добавления сотрудника в таблицу с разным количеством записей используется метод, листинг которого представлен на рисунке 21.

```
void addUserTest() throws Exception {
    final var userRqDto : UserRqDto = getUserRqDto();
    mockMvc.perform(postJson(userRqDto))
        .andDo(print())
        .andExpect(status().isCreated())
        .andExpect(jsonPath("name", equalTo(DEFAULT_STRING)))
        .andExpect(jsonPath("lastName", equalTo(DEFAULT_STRING)))
        .andExpect(jsonPath("patronymic", equalTo(DEFAULT_STRING)))
        .andExpect(jsonPath("userRole", equalTo(UserRole.ADMIN.name())))
        .andExpect(jsonPath("age", equalTo(DEFAULT_INT)))
        .andExpect(jsonPath("login", equalTo(DEFAULT_STRING)));
}
```

Рисунок 21 – Листинг Unit-теста добавления работника

Для тестирования добавления сотрудника с пустыми данными в таблицу используется метод, листинг которого представлен на рисунке 22.

```
void addEmptyUser() throws Exception {
    mockMvc.perform(postJson( body: null))
        .andDo(print())
        .andExpect(status().isBadRequest());
}
```

Рисунок 22 – Листинг Unit-теста добавления работника с пустым телом

Для тестирования добавления сотрудника с пустыми данными в таблицу используется метод, листинг которого представлен на рисунке 23.

```
void addNotValidUser() throws Exception {
    var userRqDto : UserRqDto = getUserRqDto();
    userRqDto.setName(null);
    mockMvc.perform(postJson(userRqDto))
        .andDo(print())
        .andExpect(status().isBadRequest());
}
```

Рисунок 23 – Листинг Unit-теста добавления работника с некорректными данными

Тестирование пройдено успешно.

3.4 Функциональное тестирование клиентского программного обеспечения

Подготовка для функционального тестирования заключалась в анализе функциональных требований. Все тесты производились вручную. В таблице 16 представлены основные тесты и их описание.

Таблица 16 – Интеграционное тестирование

Состав теста	Ожидаемый результат	Наблюдаемый результат
Тестирование работоспособности страницы для просмотра, создания и редактирования животных.	Тест пройден	Тест пройден (рисунок 24)
Тестирование работоспособности страницы для просмотра, создания и редактирования сотрудников.	Тест пройден	Тест пройден (рисунок 25)
Тестирование работоспособности страницы для просмотра, создания и редактирования задач.	Тест пройден	Тест пройден (рисунок 26)
Тестирование работоспособности страницы для просмотра осмотров.	Тест пройден	Тест пройден (рисунок 27)
Тестирование работоспособности страницы для просмотра и редактирования выставок.	Тест пройден	Тест пройден (рисунок 28)
Тестирование работоспособности страницы для авторизации пользователя.	Тест пройден	Тест пройден (рисунок 29)

Далее приведены скриншоты отражающие результаты тестирования. На рисунке 24 представлен результат тестирования страницы для работы с животными.

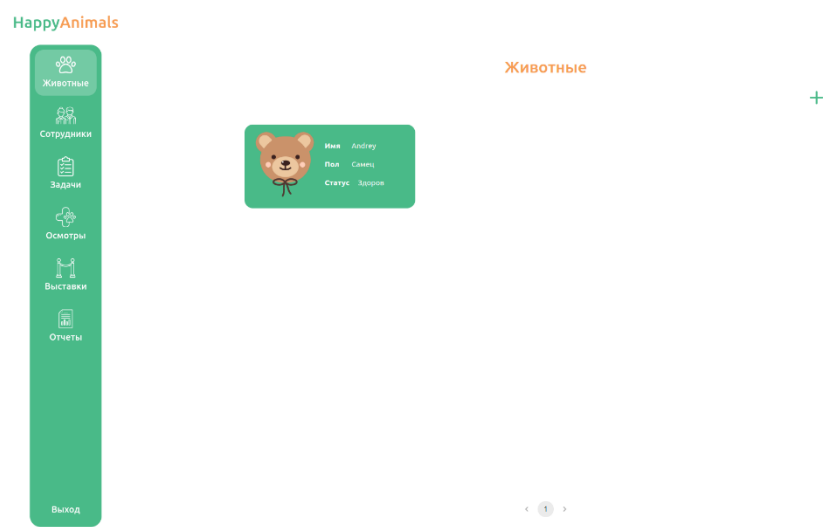


Рисунок 24 – Тестирование страницы для работы с животными

На рисунке 25 представлен результат тестирования страницы для работы с сотрудниками.

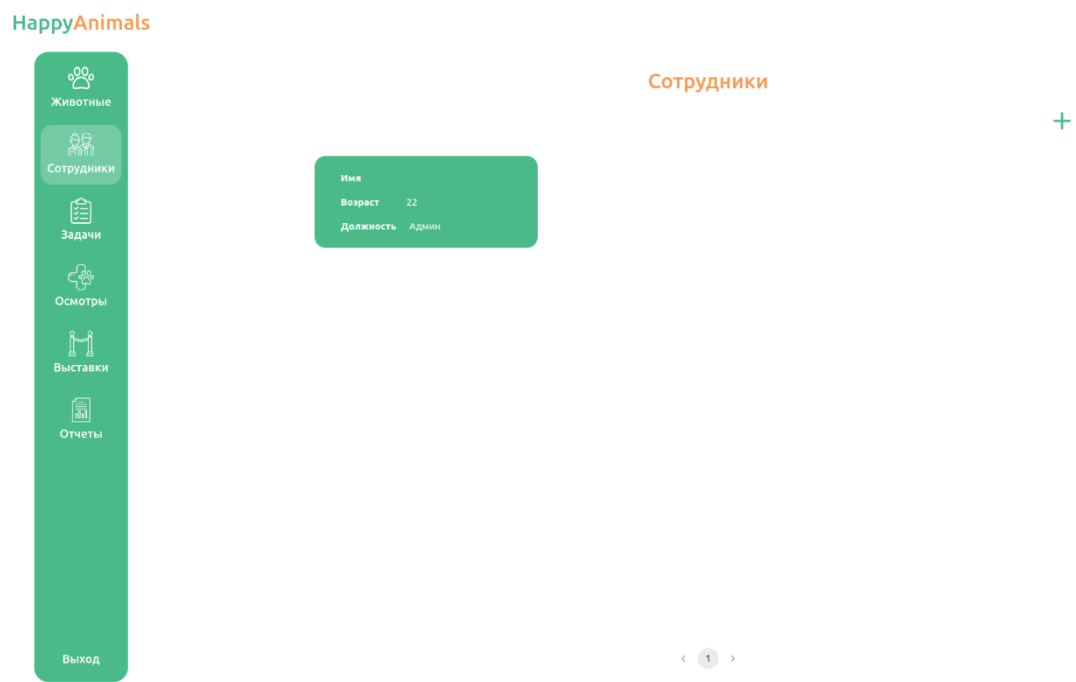


Рисунок 25 – Тестирование страницы для работы с сотрудниками

На рисунке 26 представлен результат тестирования страницы для работы с задачами.



Рисунок 26 – Тестирование страницы для работы с задачами

На рисунке 27 представлен результат тестирования страницы для просмотра осмотров.

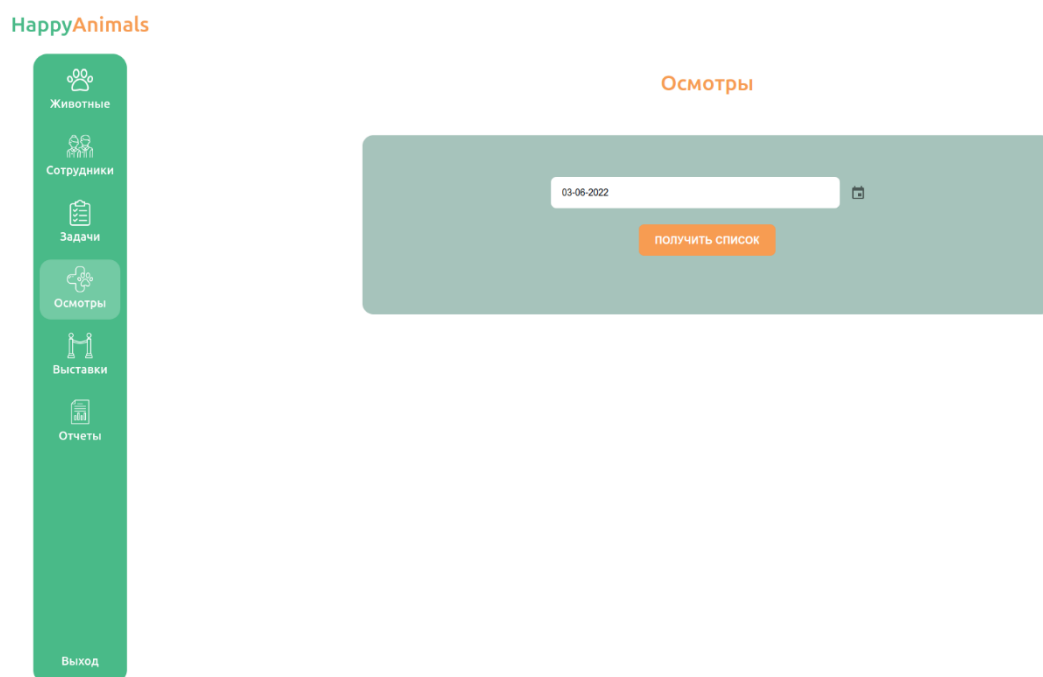


Рисунок 27 – Тестирование страницы для просмотра осмотров

На рисунке 28 представлен результат тестирования страницы для работы с выставками.

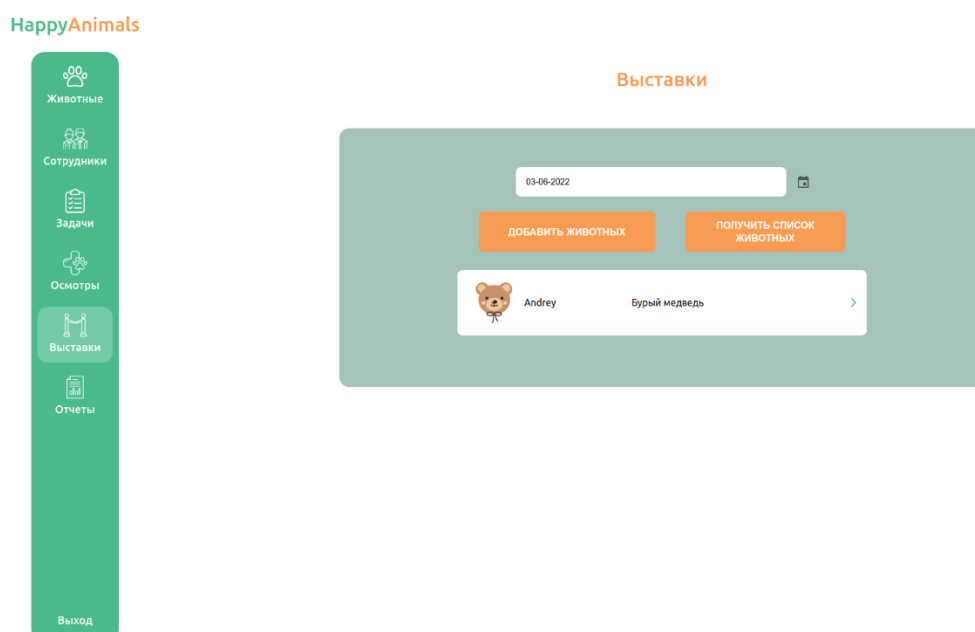


Рисунок 28 – Тестирование страницы для работы с выставками

На рисунке 29 представлен результат тестирования страницы для авторизации.

Рисунок 29 – Тестирование страницы для авторизации

Тестирование пройдено успешно.

3.5 Метрики кода

Мерой называется количественный показатель степени, количества, или размеров некоторых атрибутов продукта или процесса. Примером меры является количество ошибок.

Метрикой программного кода называется количественная мера, позволяющая оценить, в какой степени система, компоненты или процесс обладают заданным атрибутом. Примером метрики является обнаруженное количество ошибок на затраченный человеко-час [8].

Метрики программного кода можно разделить на группы:

- метрики размера программ (Source Lines of Code – SLOC-метрика);
- метрики сложности потока управления программ (метрики Холстеда).

Вычислим соответствующие метрики кода для варианта использования «Изменить данные о животном».

3.5.1 Метрика SLOC

SLOC-метрика позволяет подсчитать физические и логические строки кода. К физическим строкам относят все непустые строки исходного программного файла, к логическим строкам кода относят количество команд, использующихся в программе. Стоит отметить, что определение команды зависит от конкретного языка программирования.

Применим SLOC-метрику к программному коду для изменения данных о животном (рисунок 30). Результаты данной метрики представлены в таблице 17.

```
public AnimalRsDto editAnimal(Long animalId, AnimalRqDto animalRqDto) {
    Animal optAnimal = animalRepository.findById(animalId).orElseThrow(
        () -> new NotFoundException(ERROR_MESSAGE_NOT_FOUND_ANIMAL));
    if (!optAnimal.getStatus().getName().equals(animalRqDto.getStatus())) {
        if (animalRqDto.getStatus().equals(AnimalStatus.DEAD.getName())
            || animalRqDto.getStatus().equals(AnimalStatus.SOLD.getName())) {
            List<Animal> animalList = userRepository.findById(animalRqDto.getUserId()).getAnimals();
            animalList.remove(animalMapper.mapToAnimal(animalRqDto));
            User user = userRepository.findById(animalRqDto.getUserId()).orElseThrow(
                () -> new NotFoundException(ERROR_MESSAGE_NOT_FOUND_USER));
            user.setAnimals(animalList);
        }
    }
    Animal animal = animalMapper.mapToAnimal(animalRqDto);
    animal.setId(animalId);
    return animalMapper.mapToDto(animalRepository.save(animal));
}
```

Рисунок 30 – Программный код для изменения животного

Таблица 17 – SLOC-метрики

Число физических строк	17
Число логических строк	13
Число пустых строк	2
Число строк-комментариев	0

Данная метрика показывает, что код легко поддерживаемый.

3.5.2 Метрики Холстеда

Метрики Холстеда представлены четырьмя измеряемыми характеристиками программы:

- число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций;
- число уникальных операндов программы;
- общее число операторов в программе;
- общее число операндов в программе.

На основе метрик Холстеда можно получить следующие оценки:

- словарь программы = число уникальных операторов + число уникальных операндов;
- длина программы = общее число операторов + общее число операндов;
- объем программы = длина программы $\log_2$ словарь программы;
- сложность программы = (число уникальных операторов/2) * (общее число операндов/число уникальных операндов);
- показатель усилий программиста = сложность программы * объем программы.

Применим метрики Холстеда к программному коду для расчета обратной матрицы. Результаты данной метрики представлены в таблице 18.

Таблица 18 – Метрики Холстеда

Число операторов (уникальных)	Число операндов (уникальных)
6 (3)	31 (18)

Для данного метода были рассчитаны следующие параметры:

- словарь программы = $3 + 18 = 21$;
- длина программы = $6 + 31 = 39$;
- объем программы = $39 \log_2 21 \approx 4631$;
- сложность программы = $(3/2) * (31/18) \approx 2,58$;
- показатель усилий программиста $\approx 11947,98$.

3.5.3 Метрики устойчивости и абстрактности компонент

Один из способов определения устойчивости компонент – подсчитать количество входящих и исходящих зависимостей. Вместе они позволяют вычислить позиционную устойчивость компонента:

$$I \text{ (неустойчивость): } I = \frac{Ce}{Ca + Ce},$$

где Ce (исходящие связи) – количество классов внутри данного компонента, зависящих от классов вне компонента;

Ca (входящие связи) – количество классов вне данного компонента, зависящих от классов внутри компонента.

Пример результата вычисления метрики для компонента сервисного слоя, предназначенного для обработки информации о животных, который находится в классе `AnimalServiceImpl`.

В данном случае величина $Ca = 3$, $Ce = 1$, $I = 1/4$.

Метрика A , являющаяся мерой абстрактности компонента, равна отношению количества абстрактных классов к общему количеству классов в компоненте.

$$A \text{ (абстрактность): } A = \frac{Na}{Nc},$$

где Na – количество абстрактных классов в компоненте;

Nc – количество классов в компоненте.

В коде программы абстракцией для класса `AnimalServiceImpl` является `AnimalService`. Соответственно метрика $A = 0.5$.

Соотношение между устойчивостью (I) и абстрактностью (A) компонента обработки информации о животных можно представить на следующем графике, изображенном на рисунке 31.

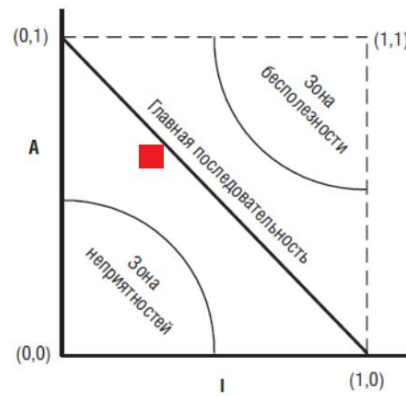


Рисунок 31 – Соотношение между устойчивостью (I) и абстрактностью (A) для компонента `AnimalServiceImpl`

На рисунке 31 максимально устойчивые и абстрактные компоненты находятся в левом верхнем углу, в точке (0,1), а максимально неустойчивые и конкретные – в правом нижнем углу, в точке (1,0). Желательно, чтобы компонент находился на главной последовательности или близко к ней. Для измерения, насколько далеко компонент отстоит от идеала, определяют следующую метрику:

D (расстояние от главной последовательности):

$$D = \frac{|A + I - 1|}{\sqrt{2}},$$

где A – абстрактность;

I – неустойчивость.

Величина D изменяется в диапазоне [0; ~0,707]. Для компонента `AnimalServiceImpl` это значение приблизительно равно 0,18.

Для удобства анализа можно использовать нормированное расстояние:

$$D' = |A + I - 1|.$$

Метрика D' изменяется в диапазоне [0,1]. При этом 0 означает, что компонент находится на главной последовательности, а 1 – что он максимально удален от нее. Для компонента `AnimalServiceImpl` D'=0,25.

3.5.4 Комплексный показатель качества кода

Наиболее часто используемой метрикой является комплексный показатель качества кода, которая рассчитывается по следующей формуле:

$$MI = \text{MAX}(0, (171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LoC)) * 100 / 171),$$

где HV – Halstead Volume, вычислительная сложность. Чем больше операторов, тем больше значение этой метрики;

CC – Cyclomatic Complexity. показывает структурную сложность кода, т.е. количество различных ветвей в коде. Чем больше этот показатель, тем больше тестов должно быть написано, для полного покрытия кода;

LoC – количество строк кода.

Комплексный показатель качества кода может принимать значения от 0 до 100 и показывает относительную сложность поддержки кода. Чем больше значение этой метрики, тем легче поддерживать код.

Рассмотрим подсчет данной метрики на примере метода получения всех животных, привязанных к пользователю. Данный метод находится в классе UserService.

$$HV = 8, CC = 4, LoC = 7, MI = 93,14$$

Данное значение показывает, что метод легко поддерживать.

В соответствии с полученными метриками можно считать код приложения легко поддерживаемым.

4 Планирование и оценка бюджета проекта

Управление проектами является неотъемлемой частью выполнения работы, в ходе которой намечаются и достигаются цели проекта. Для достижения поставленного результата, учитывая при этом риски и ограничения на трудозатраты, ресурсы и качество, необходимо составить план, рационально применить свои знания и инструментальные средства.

Существует несколько основных групп при планировании проектов:

- планирование проектных задач;
- планирование расписания проекта;
- планирование команды проекта;
- управление рисками.

При планировании разработки было выделено 30 задач. Перечень задач и их виды, предшественники и длительность каждой задачи представлены в таблице 19.

Таблица 19 – Перечень задач проекта

№	Название	Вид задачи	Предшественник	Длительность, дни
1	2	3	4	5
1	Начало реализации проекта	Веха	–	–
2	Программирование	Фаза	–	31
3	Разработка базы данных	Фаза	–	9
4	Постановка задачи	Задача	3	1
5	Разработка структуры базы данных	Задача	4	3
6	Заполнение базы данных	Задача	5	2
7	Разработка триггеров	Задача	6	3
8	Разработка базы данных завершена	Задача	7	–
9	Разработка серверной части приложения	Фаза	3	9
10	Постановка задачи	Задача	9	1
11	Разработка запросов к базе данных	Задача	10	4
12	Разработка endpoint'ов	Задача	11	4
13	Разработка завершена	Задача	12	–
14	Разработка клиентской части приложения	Фаза	9	5

Продолжение таблицы 19

1	2	3	4	5
15	Постановка задачи	Задача	14	1
16	Разработка интерфейса	Задача	15	4
17	Разработка клиентской части завершена	Задача	16	—
18	Программирование завершено	Задача	17	—
19	Отладка	Фаза	14	5
20	Тестирование и исправление ошибок	Задача	19	
21	Составление программной документации	Задача	20	
22	Отладка завершена	Задача	21	—
23	Рефакторинг	Фаза	19	3
24	Оптимизация кода	Задача	23	3
25	Проверка соответствия SOLID	Задача	24	2
26	Документирование	Фаза	23	2
27	Создание установочного файла	Задача	26	1
28	Написание руководства пользования	Задача	27	1
29	Документация завершена	Задача	28	—
30	Конец проекта	Веха	29	—

Планирование разработки и оценка бюджета были выполнены с помощью программы для управления проектами Microsoft Project 2016. Проект был начат 3 мая 2023 года. Результат внесения информации о проекте представлен на рисунке 32.

Ид.	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало реализации проекта	0 дней	Ср 03.05.23	Ср 03.05.23		
2	Программирование	31 дней	Ср 03.05.23	Ср 14.06.23		Интернет[600,00 Р]
3	Разработка базы данных	9 дней	Ср 03.05.23	Пн 15.05.23		
4	Постановка задачи	1 день	Ср 03.05.23	Ср 03.05.23	1	Ведущий программист,Пр
5	Разработка структуры базы данных	3 дней	Чт 04.05.23	Пн 08.05.23	4	Ведущий программист
6	Заполнение базы данных	2 дней	Вт 09.05.23	Ср 10.05.23	5	Ведущий программист
7	Разработка триггеров	3 дней	Чт 11.05.23	Пн 15.05.23	6	Ведущий программист
8	Разработка базы данных завершена	0 дней	Пн 15.05.23	Пн 15.05.23	7	
9	Разработка серверной части приложения	9 дней	Вт 16.05.23	Пт 26.05.23	3	
10	Постановка задачи	1 день	Вт 16.05.23	Вт 16.05.23		
11	Разработка запросов к базе данных	4 дней	Ср 17.05.23	Пн 22.05.23	10	
12	Разработка endpoint'ов	4 дней	Вт 23.05.23	Пт 26.05.23	11	
13	Разработка завершена	0 дней	Пт 26.05.23	Пт 26.05.23	12	
14	Разработка клиентской части приложения	5 дней	Пн 29.05.23	Пт 02.06.23	13	
15	Постановка задачи	1 день	Пн 29.05.23	Пн 29.05.23	1	Ведущий программист,Пр
16	Разработка интерфейса	4 дней	Вт 30.05.23	Пт 02.06.23	15	Программист,Ведущий пр
17	Разработка клиентской части завершена	0 дней	Пт 02.06.23	Пт 02.06.23	16	
18	Программирование завершено	0 дней	Пт 02.06.23	Пт 02.06.23	8, 17	
19	Отладка	5 дней	Пн 05.06.23	Пт 09.06.23	18	Интернет[600,00 Р]
20	Тестирование и исправление ошибок	3 дней	Пн 05.06.23	Ср 07.06.23	18	Тестировщик
21	Составление программной документации	2 дней	Чт 08.06.23	Пт 09.06.23	20	Программист,Тестировщ
22	Отладка завершена	0 дней	Пт 09.06.23	Пт 09.06.23	21	
23	Рефакторинг	3 дней	Пн 12.06.23	Ср 14.06.23	22	
24	Оптимизация кода	3 дней	Пн 12.06.23	Ср 14.06.23		
25	Проверка соответствия SOLID	2 дней	Пн 12.06.23	Вт 13.06.23		
26	Документирование	2 дней	Чт 15.06.23	Пт 16.06.23		Интернет[600,00 Р]
27	Создание установочного файла	1 день	Чт 15.06.23	Чт 15.06.23	23	Тестировщик,Лицензия п
28	Написание руководства пользования	1 день	Пт 16.06.23	Пт 16.06.23	27	Тестировщик
29	Документация завершена	0 дней	Пт 16.06.23	Пт 16.06.23	28	
30	Конец проекта	0 дней	Пт 16.06.23	Пт 16.06.23	29	

Рисунок 32 – Задачи проекта

В соответствии с занесенными задачами проекта, проанализированными ресурсами и соответствующими им затратами, Microsoft Office Project 2016 автоматически построит диаграмму Ганта, которая иллюстрирует календарный график выполнения работ по проекту. Диаграмма Ганта представлена на рисунке 33.

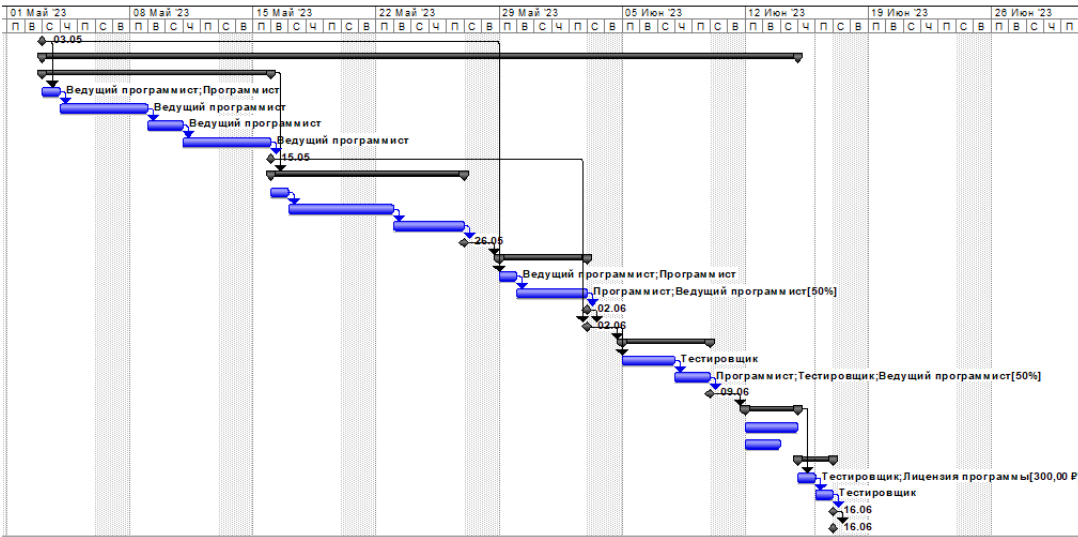


Рисунок 33 – Диаграмма Ганта

На рисунке 34 представлено распределение ресурсов, где синим цветом отмечены нормы рабочего времени работников, а красным – переработки.

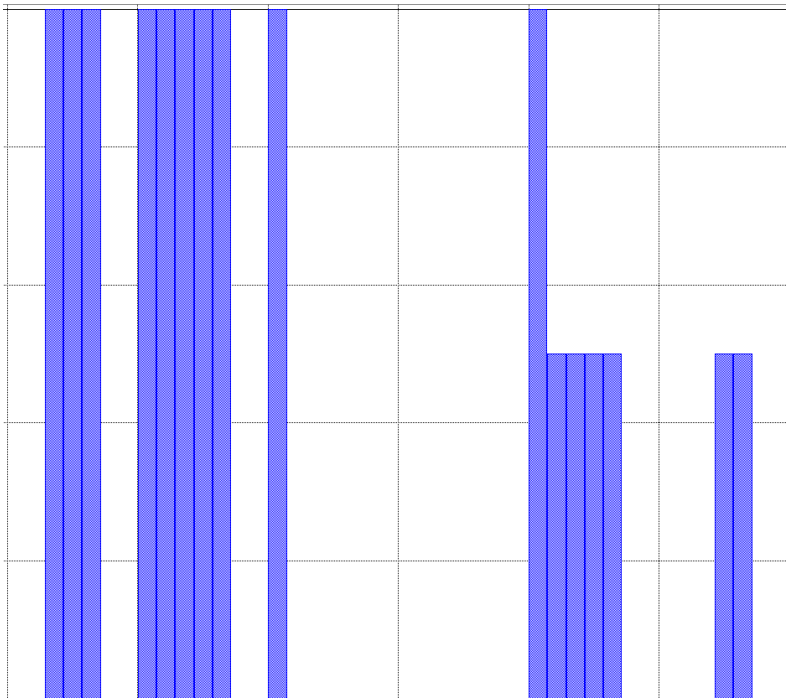


Рисунок 34 – Распределение ресурсов

Как видно по рисунку, переработок в плане нет, более того, в некоторые дни работник занят на 50% времени, соответственно данный план можно считать корректным.

Также приложение Microsoft Office Project 2016 может рассчитать стоимость проекта, она составила: 38925 рублей. Она состоит из затрат на заработную плату разработчиков, а также на электричество и интернет.

Заключение

В результате выполнения данной работы была разработана система мониторинга за животными в зоопарке и были реализованные поставленные задачи.

В ходе выполнения работы были выполнены следующие этапы:

- постановка задачи и анализ предметной области;
- проектирование элементов системы;
- тестирование программы;
- расчет метрик кода.

В результате работы было разработано Web-приложение. После некоторой доработки на основе жалоб и пожеланий пользователей, оно может быть использовано и внедрено, как основное средство для мониторинга за животными в различных зоопарках.

Список использованных источников

1. SOLID (объектно-ориентирование программирование) // Википедия [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/SOLID_объектно-ориентированное_программирование (дата обращения: 01.05.2023);
2. Объектно-ориентированное программирование // Википедия [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование (дата обращения: 03.05.2023);
3. Юлиана Козмина, Роб Харроп, Крис Шефер, Кларенс Хо. Spring 5 для профессионалов / Диалектика, 2019 – 521 с.;
4. Оригинал Брэда Фроста (Brad Frost), перевод Naissur. Атомарный веб-дизайн // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/249223/> (дата обращения: 10.05.2023);
5. Оригинал Илья Suzdanitski, перевод ru_vds. React.js: понятное руководство для начинающих // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/company/ruvds/blog/428077/> (дата обращения: 10.05.2023);
6. В поисках самого лучшего способа тестирования системы // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/jugru/articles/341852/> (дата обращения: 11.05.2023);
7. Фундаментальная теория тестирования // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/549054/> (дата обращения: 11.05.2023);
8. Л. В. Гурьянов, П. П. Макарычева. Выполнение выпускной квалификационной работы: учебное пособие для бакалавров. / Пензенский государственный университет, 2015. -119 с.

Приложение А
ПОДРОБНОЕ ОПИСАНИЕ ТАБЛИЦ БАЗЫ ДАННЫХ
(рекомендуемое)

Таблица А.1 – Подробное описание таблицы «Сотрудник»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Код сотрудника	целый	4	0001-9999	нет	PK	2
name	Имя	символьный	50	'А-я'	нет		Александр
last_name	Фамилия	символьный	50	'А-я'	нет		Шашков
patronymic	Отчество	символьный	50	'А-я'	нет		Дмитриевич
age	Дата рождения	дата	8	01.01.1960 – 31.12.2100	нет		23.07.2001
user_role	Роль пользователя	символьный	13	'Администратор', 'Работник зоопарка', 'Ветеринар'	нет		Администратор
is_locked	Заблокирован пользователь или нет	логический	4	'true', 'false'	нет		true
login	Логин	символьный	100	'А-я', 'А-z', '0-9', знаки препинания	нет		Alexsh_2307
password	Пароль	символьный	100	'А-z', '0-9', знаки препинания	нет		Eg89h4398hng98sdnv8hf

Таблица А.2 – Подробное описание таблицы «Животное»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Код животного	целый	4	0001-9999	нет	PK	2
name	Кличка животного	символьный	50	'А-я'	нет		Симба
age	Дата рождения	дата	8	01.01.1900 – 31.12.2022	нет		100
weight	Вес	вещественный	3	000-999	нет		38.7
max_temp	Максимальная комфортная температура	вещественный	4	-30.0 – 40.0	нет		10
min_temp	Минимальная комфортная температура	вещественный	4	-30.0 – 40.0	нет		-10
location	Местоположение	символьный	50	'А-я'	нет		Зимний вольер
external_features	Внешние особенности	символьный	50	'А-я'	да		Альбинос
features_of_keeping	Особенности ухода	символьный	50	'А-я'	да		Требуется прием антибиотиков вместе с пищей
health_status	Статус здоровья	целый	4	0001-9999	нет	FK	2
user_id	Код сотрудника	целый	4	0001-9999	нет	FK	2
animal_kind	Вид	целый	4	0001-9999	нет	FK	2

Таблица А.3 – Подробное описание таблицы «Выставка»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Код выставки	целый	4	0001-9999	нет	PK	2
date	Дата выставки	дата	4	01.01.2022 – 31.12.2030	нет		28.07.2027
animal_id	Код животного	целый	4	0001-9999	нет	FK	1

Таблица А.4 – Подробное описание таблицы «Задача»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Код задачи	целый	4	0001-9999	нет	PK	2
type	Тип задачи	символьный	50	‘А-я’	нет		Кормление
animal_id	Код животного	целый	4	0001-9999	нет	FK	76
date_time	Дата и время получения	Дата и время	8	01.01.2022 – 31.12.2030, 00:00 – 23:59	нет		01.12.2023, 9:00
state	Состояние	символьный	50	‘А-я’	нет		Выполнено
repeat_type	Тип повторения	символьный	50	‘А-я’	нет		Ежедневно

Таблица А.5 – Подробное описание таблицы «Отчет»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Код отчета	целый	4	0001-9999	нет	PK	2
date	Дата отчета	Дата	8	01.01.2022 – 31.12.2030	нет		29.05.2025
user_id	Код пользователя	целый	4	01.01.1900 – 10.02.2022	нет	FK	2
text	Текст отчета	символьный	10000	‘А-я’, знаки препинания	да		Без происшествий

Таблица А.6 – Подробное описание таблицы «Автоотчет»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Код автоотчета	целый	4	0001-9999	нет	PK	2
date	Дата отчета	Дата	8	01.01.2022 – 31.12.2030	нет		29.05.2025
user_id	Код пользователя	целый	4	0001-9999	нет	FK	2
task_id	Код задачи	целый	4	0001-9999	да	FK	2

Таблица А.7 – Подробное описание таблицы «Вид животного»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
Id	Код_вида	Целый	4	0001-9999	нет	РК	2
kind	Вид_животного	символьный	50	'А-я'	нет		Лев
class	Класс животного	символьный	50		нет		Млекопитающее
Squad	Отряд животного	символьный	50		нет		Кошачьи
pic	Картинка вида	массив байт	10000	'A-z', '0-9'	нет		Eworh32hfe89dh32-2q98rhf...

Таблица А.8 – Подробное описание таблицы «Тип повторения задач»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
type	Тип повторения задач	символьный	50	'А-я'	нет	РК	Ежедневно

Таблица А.9 – Подробное описание таблицы «Осмотр»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Код_осмотра	целый	4	0001-9999	нет	РК	1
date_time	Дата и время осмотра	дата и время	12	01.01.2022 – 31.12.2030, 00:00 – 23:59	нет		23.11.2024, 13:00

Приложение Б
ПРИМЕРЫ СТРАНИЦ ПРИЛОЖЕНИЯ
(рекомендуемое)

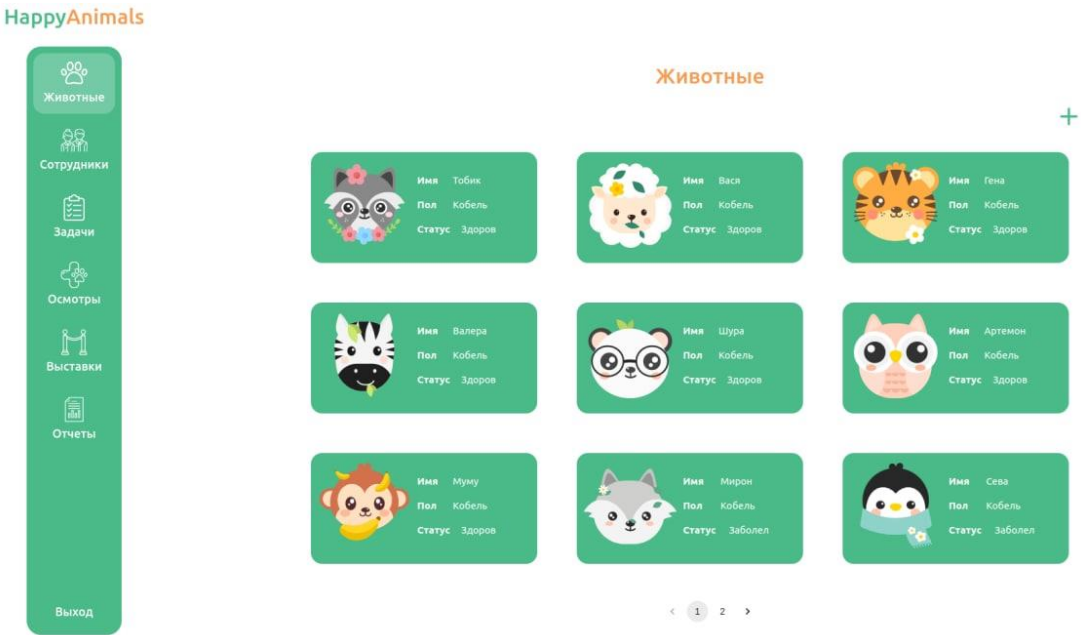


Рисунок Б.1 – Страница для работы с животными

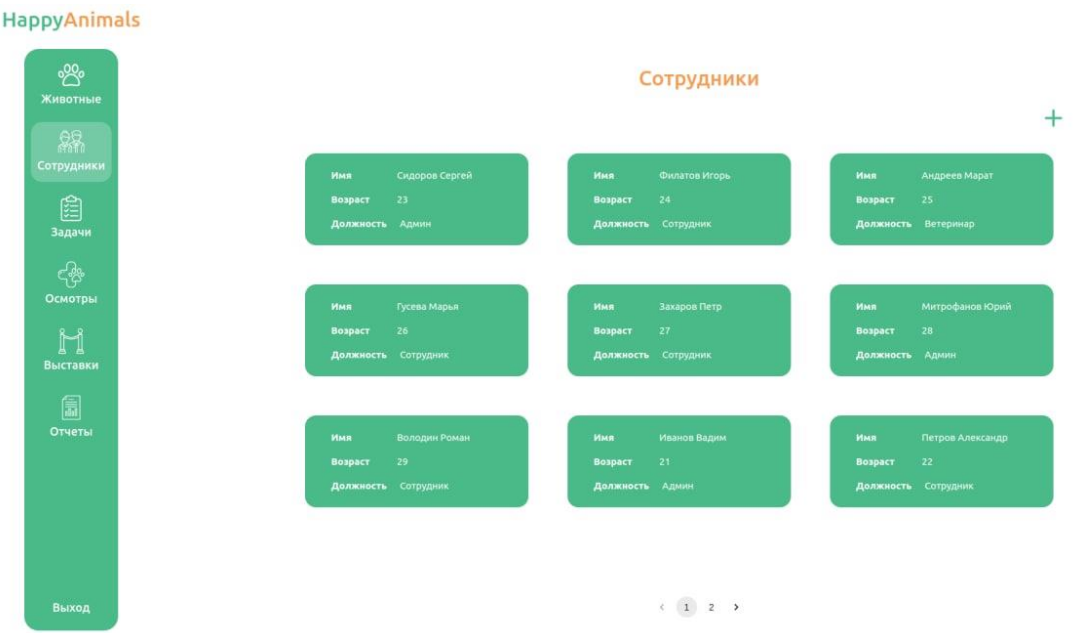


Рисунок Б.2 – Страницы для работы с сотрудниками



Рисунок Б.3 – Страница для работы с задачами

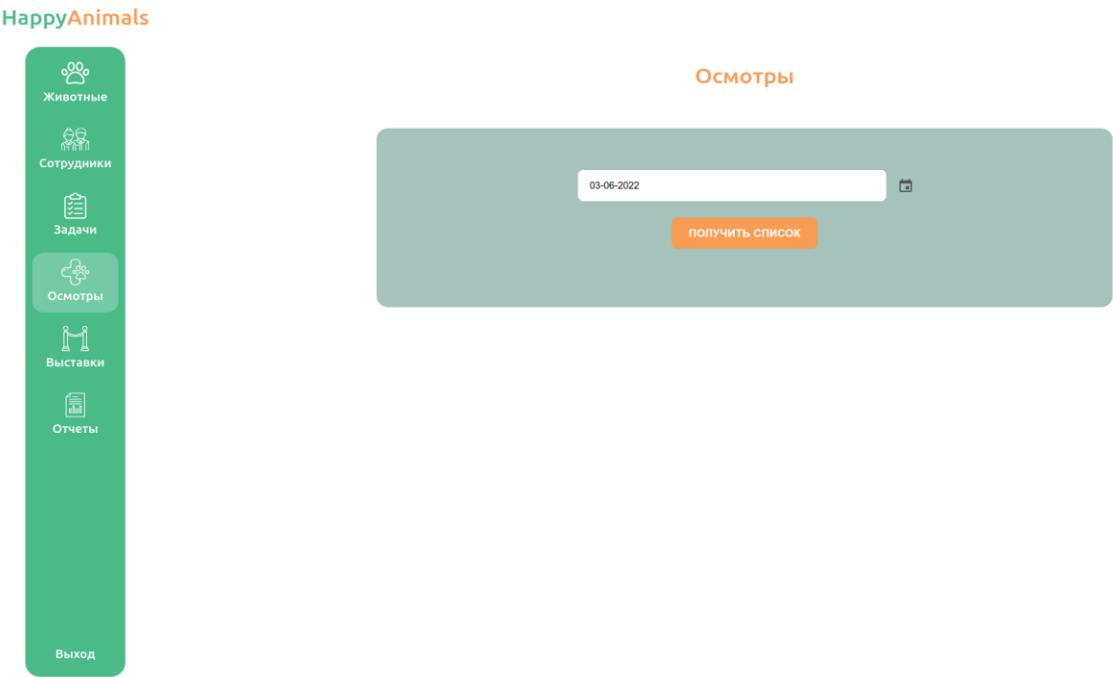
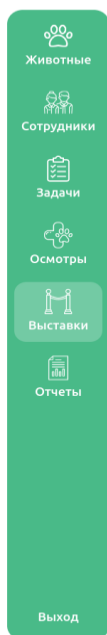


Рисунок Б.4 – Страницы для просмотра осмотров

HappyAnimals



Выставки

03-06-2022

ДОБАВИТЬ ЖИВОТНЫХ

ПОЛУЧИТЬ СПИСОК ЖИВОТНЫХ

	Andrey	Бурый медведь	>
---	--------	---------------	---

Рисунок Б.5 – Страница для просмотра и редактирования выставок

Авторизация

Логин

tgadmin

Пароль

ВОЙТИ

Рисунок Б.6 – Страница для авторизации

Приложение В
КОД ПРОГРАММЫ
(рекомендуемое)

Листинг В.1 – AnimalService.java

```

@Service
public class AnimalService {
    private final AnimalRepository animalRepository;
    private final UserRepository userRepository;
    private final AnimalMapper animalMapper;

    private static final String ERROR_MESSAGE_NOT_FOUND_ANIMAL =
        "Зверь не найден";

    @Autowired
    public AnimalService(AnimalRepository animalRepository,
                        UserRepository userRepository,
                        AnimalMapper animalMapper) {
        this.animalRepository = animalRepository;
        this.userRepository = userRepository;
        this.animalMapper = animalMapper;
    }

    @Transactional
    public AnimalRsDto saveAnimal(AnimalRqDto animalRqDto) {
        Animal animal = animalMapper.mapToAnimal(animalRqDto);
        return
            animalMapper.mapToDto(animalRepository.save(animal));
    }

    @Transactional
    public Page<AnimalRsDto> getAllByUserId(Pageable pageable,
        Long userId) {
        User user = userRepository.findById(userId).orElseThrow(
            () -> new
                NotFoundException(ERROR_MESSAGE_NOT_FOUND_ANIMAL));
    }

```

```

        Page<Animal> allByUser =
animalRepository.findAllByUser(user, pageable);

        if (allByUser.isEmpty()) {

            throw new
NotFoundException(ERROR_MESSAGE_NOT_FOUND_ANIMAL);

        }

        return allByUser.map(animalMapper::mapToDto);
    }

```

```

@Transactional

    public Page<AnimalRsDto> getAllByUserLogin(Pageable pageable,
String login) {

        User user = userRepository.findByLogin(login).orElseThrow(

            () -> new
NotFoundException(ERROR_MESSAGE_NOT_FOUND_ANIMAL));

        Page<Animal> allByUser =
animalRepository.findAllByUser(user, pageable);

        if (allByUser.isEmpty()) {

            throw new
NotFoundException(ERROR_MESSAGE_NOT_FOUND_ANIMAL);

        }

        return allByUser.map(animalMapper::mapToDto);
    }

```

```

@Transactional

    public Page<AnimalRsDto> getAll(Pageable pageable,
MyUserDetails userDetails, Long userId) {

        if (userDetails.getUserRole() == UserRole.EMPLOYEE) {

            return getAllByUserLogin(pageable,
userDetails.getUsername());

        }

        if (userId == null) {

            Page<Animal> allAnimals =
animalRepository.findAll(pageable);

```

```

        return allAnimals.map(animalMapper::mapToDto);
    }

    return getAllByUserId(pageable, userId);
}

@Transactional

public AnimalRsDto getById(Long id) {
    Animal animal = animalRepository.findById(id).orElseThrow(
        () -> new
NotFoundException(ERROR_MESSAGE_NOT_FOUND_ANIMAL));
    return animalMapper.mapToDto(animal);
}

@Transactional

public AnimalRsDto editAnimal(Long animalId, AnimalRqDto
animalRqDto) {
    if
(!animalRepository.getById(animalId).getStatus().getName().equals(
animalRqDto.getStatus())) {
        if
(animalRqDto.getStatus().equals(AnimalStatus.DEAD.getName()) ||
animalRqDto.getStatus().equals(AnimalStatus.SOLD.getName())) {
            List<Animal> animalList =
userRepository.getById(animalRqDto.getUserId()).getAnimals();

            animalList.remove(animalMapper.mapToAnimal(animalRqDto));

            User user =
userRepository.getById(animalRqDto.getUserId());
            user.setAnimals(animalList);
            userRepository.save(user);
        }
    }

    Animal animal = animalMapper.mapToAnimal(animalRqDto);
}

```

```
        animal.setId(animalId);

        return
        animalMapper.mapToDto(animalRepository.save(animal));
    }

    @Transactional

    public List<AnimalRsDto> getPermittedAnimals() {
        return animalRepository.findAll()
            .stream()
            .filter(animal ->
animal.getStatus().isPermissionToParticipate())
            .map(animalMapper::mapToDto)
            .collect(Collectors.toList());
    }
}
```