

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное

образовательное учреждение высшего образования

«Казанский национальный исследовательский

технологический университет»

(ФГБОУ ВО «КНИТУ»)

Кафедра Интеллектуальных систем и управления информационными ресурсами\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направление (специальность) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направленность (профиль, специализации) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Группа\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КУРСОВОЙ ПРОЕКТ

по дисциплине «Разработка пользовательского интерфейса»

на тему «Разработка клиентской части приложения для ветеринарной клиники»

Исполнитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись) (Ф.И.О.)

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись) должность (Ф.И.О.)

Нормоконтролер\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись) должность (Ф.И.О.)

Работа (проект) защищена с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись)

Казань, 2024г

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc184804790)

[ЗАДАНИЕ 4](#_Toc184804791)

[ЛИСТ НОРМОКОНТРОЛЕРА 5](#_Toc184804792)

[1. Анализ требований 6](#_Toc184804793)

[1.1. Функциональные требования 6](#_Toc184804794)

[1.2. Нефункциональные требования 8](#_Toc184804795)

[2. Технические требования 9](#_Toc184804796)

[2.1. Используемые технологии 9](#_Toc184804797)

[2.2. Структура данных 9](#_Toc184804798)

[3. Дизайн интерфейса 10](#_Toc184804799)

[3.1. Описание структуры приложения 10](#_Toc184804800)

[3.2. Схематичный макет интерфейса 14](#_Toc184804801)

[4. Техническая реализация 15](#_Toc184804802)

[4.1. Общая структура проекта 15](#_Toc184804803)

[4.3. Описание функциональных модулей 18](#_Toc184804804)

[5. Пошаговая реализация 22](#_Toc184804805)

[5.1. Вёрстка страницы по макету 22](#_Toc184804806)

[5.2. Вёрстка на компоненты (MPV) 34](#_Toc184804807)

[5.3. Добавление функционала 47](#_Toc184804808)

[5.4. Работа с сетью 65](#_Toc184804809)

[ЗАКЛЮЧЕНИЕ 78](#_Toc184804810)

[СПИСОК ЛИТЕРАТУРЫ 79](#_Toc184804811)

# ВВЕДЕНИЕ

Современные технологии стремительно развиваются, а их применение в различных сферах жизни становится неотъемлемой частью повседневности. Ветеринарные клиники — не исключение, и развитие удобных и функциональных веб-интерфейсов значительно упрощает процесс управления данными о питомцах. В условиях растущего спроса на цифровизацию услуг, создание веб-сайтов с интуитивно понятным дизайном и интерактивными возможностями становится важной задачей для обеспечения удобства пользователей и повышения эффективности работы организаций.

Разработка клиентской части веб-сайта для ветеринарной клиники имеет высокую практическую значимость. Удобный интерфейс позволяет владельцам животных быстро получать доступ к информации о питомцах, записываться на прием и редактировать данные. Это соответствует современным трендам на рынке технологий, где упор делается на пользовательский опыт (UX) и автоматизацию рутинных задач.

Цель исследования — разработка функционального клиентского веб-интерфейса для управления данными о питомцах с использованием технологий.

Для достижения цели были поставлены следующие задачи:

* Разработка макета клиентской части приложения.
* Создание структуры веб-страницы с использованием языка разметки HTML.
* Стилизация интерфейса с помощью CSS, с применением цветовой схемы, ориентированной на комфортное восприятие.
* Внедрение интерактивных функций через JavaScript, таких как управление модальными окнами, редактирование и добавление данных о питомцах.

# ЗАДАНИЕ

на курсовую работу (проект) студенту кафедры \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Тема работы (проекта): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Исходные данные к проекту: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Содержание расчетно-пояснительной записки (включая перечень подлежащих разработке вопросов, включая вопросы стандартизации и контроля качества) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перечень графического материала (схемной документации) \_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Консультанты по проекту (с указанием относящихся к ним разделов) \_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

# ЛИСТ НОРМОКОНТРОЛЕРА

1. Лист является обязательным приложением к пояснительной записке дипломного (курсового) проекта.

2. Нормоконтролер имеет право возвращать документацию без рассмотрения в случаях:

- нарушения установленной комплектности,

- отсутствия обязательных подписей,

- нечеткого выполнения текстового и графического материала.

3. Устранение ошибок, указанных нормоконтролером, обязательно.

П е р е ч е н ь

замечаний и предложений нормоконтролера по дипломному (курсовому) проекту, студента

(группа, инициалы, фамилия)

|  |  |  |
| --- | --- | --- |
| Л(Лист (страница) | Условное обозначение (код ошибок) | Содержание замечаний и предложений со ссылкой на нормативный документ, стандарт или типовую документацию |
|  |  |  |

Дата «\_\_\_\_»\_\_\_\_\_\_\_\_\_20\_\_\_г.

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (инициалы, фамилия)

# 1. Анализ требований

Анализ требований — это процесс, направленный на выявление и систематизацию всех требований, предъявляемых к программному продукту. Этот этап играет ключевую роль в разработке, так как позволяет определить, какие функции и возможности должен реализовать проект.

В случае конкретно этого проекта анализ требований важен для того, чтобы определить, какие функции необходимы для управления данными о питомцах, такие как добавление, редактирование и удаление информации, а также для создания удобного интерфейса для взаимодействия с пользователями.

## 1.1. Функциональные требования

1. Управление данными о питомцах:
   * Пользователь может просматривать список своих питомцев.
   * Возможность добавления нового питомца через форму.
   * Редактирование информации о питомце (например, изменение имени, породы или возраста).
   * Удаление питомца из списка.
2. Интерактивные компоненты:
   * Открытие модальных окон для добавления или редактирования питомцев.
   * Закрытие модальных окон кнопками "Отмена" или "Закрыть".
3. Синхронизация данных с сервером:
   * Получение списка питомцев с удалённого API.
   * Сохранение новых питомцев на сервере.
   * Обновление информации о питомце через API.
   * Удаление питомца с сервера.
4. Структура интерфейса:

Основной экран включает разделы:

* + - "Личные данные" с информацией о пользователе.
    - "Мои питомцы" с функциональностью управления списком питомцев.

Пользовательские истории

1. Просмотр списка питомцев:

Пользователь может открыть приложение и видеть список моих питомцев, чтобы легко управлять ими.

1. Добавление питомца:

Пользователь может нажать на кнопку "Добавить нового питомца" и заполнить форму, чтобы добавить нового питомца в список.

1. Редактирование питомца:

Пользователь может нажать кнопку "Редактировать" на карточке питомца, чтобы изменить его данные, такие как имя, порода, пол или возраст.

1. Удаление питомца:

Пользователь может нажать кнопку "Удалить" на карточке питомца, чтобы удалить питомца из списка.

1. Просмотр личной информации:

Пользователь может видеть свои личные данные на отдельной карточке.

1. Интерактивные окна:

Пользователь может сделать так, чтобы окно добавления или редактирования питомца открывалось по нажатию на соответствующую кнопку, а также чтобы окно закрывалось без добавления питомца или изменения информации о питомце при нажатии на соответствующую кнопку.

1. Сохранение данных:

Все изменения (добавление, обновление, удаление питомцев) будут сохранятся на сервере, чтобы информация оставалась доступной при следующем входе.

8. При отсутствии питомцев будет видна заглушка, которая показывает, что питомцы ещё не были добавлены.

## 1.2. Нефункциональные требования

1. Удобство использования:

Интерфейс должен быть интуитивно понятным, с чётким разделением функциональных зон.

1. Производительность:

Приложение должно загружать список питомцев и данные пользователя в течение 0,5–2 секунд после открытия.

1. Кроссбраузерность:

Приложение должно корректно работать в современных браузерах (Chrome, Firefox, Edge, Safari).

1. Реакция интерфейса:
   * Окна добавление и редактирования питомца должны открываться и закрываться без задержек.
   * Обновления данных на сервере должны подтверждаться соответствующими изменениями в интерфейсе.

# 2. Технические требования

Технические требования — это ключевые характеристики и условия, которые определяют основу для разработки программного продукта или системы. Их цель состоит в том, чтобы зафиксировать минимальные параметры, необходимые для реализации функционала [1].

## 2.1. Используемые технологии

Технологии, которые будут использоваться:

* HTML (HyperText Markup Language):  
  Создание структуры веб-приложения, включая разметку элементов интерфейса [14].
* CSS (Cascading Style Sheets):  
  Применение для стилизации страницы: оформление цветовой схемы, шрифтов, отступов и визуальных эффектов [7].
* JavaScript (ES6+):  
  Добавление интерактивности, управление модальными окнами, редактирование и синхронизация данных через API [11].

## 2.2. Структура данных

Формат данных для работы с API:

{

"name": "Громопетр",

"type": "Собака",

"sex": "Мужской",

"breed": "Без породы",

"age": "34",

"id": "2"

}

# 3. Дизайн интерфейса

Дизайн интерфейса играет ключевую роль в разработке приложений, поскольку он напрямую влияет на восприятие, удобство использования и эффективность взаимодействия пользователя с программой [5]. Вот несколько причин, почему дизайн так важен:

Хорошо продуманный дизайн помогает создать положительный пользовательский опыт, что способствует увеличению удовлетворенности пользователей и уменьшению вероятности отказа от использования приложения [15]. Интуитивно понятный интерфейс делает взаимодействие с приложением легким и быстрым. Также он упрощает навигацию и доступ к нужной информации или функционалу. Чем легче пользователю найти нужную информацию или выполнить действие, тем выше шанс, что он останется довольным продуктом.

Эффективный дизайн способствует снижению количества ошибок пользователей. Например, очевидно расположенные кнопки для действий помогают предотвратить неправильные действия и улучшить точность взаимодействия.

Эстетически привлекательный дизайн помогает создать положительное первое впечатление о приложении. Это также способствует повышению доверия к продукту и может повлиять на решение о его использовании.

В целом, правильный дизайн приложения влияет на успешность в решении задач, стоящих перед пользователем, что делает его необходимым элементом при разработке любого продукта [10].

## 3.1. Описание структуры приложения

Приложение состоит из нескольких основных компонентов, обеспечивающих удобное взаимодействие с пользователем.

Всё начинается с шапки страницы (верхняя часть страницы) сайта, который включает несколько важных элементов интерфейса. Он начинается с логотипа слева, далее идет основное меню с разделами: "Личный кабинет", "О нас", "Новости и акции", "Контакты", "Блог".

Справа от меню отображается иконка местоположения, с указанием города "Казань", что может служить для определения текущего местоположения. Цвета и шрифты, используемые в шапке страницы, создают чистый и лаконичный дизайн, что улучшает восприятие информации.

Такая шапка страницы, которая изображена на рисунке 2, помогает пользователям быстро найти нужные разделы и создает положительное первое впечатление о сайте.



Рисунок 1 – Шапка страницы

В разделе личных данных отображается информация о пользователе, включая имя, фамилию, отчество, телефон и описание "О себе". Все эти данные представлены в виде карточки с блоками информации и будут выглядеть в соответствии с рисунком 1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – Личные данные

Для добавления нового питомца предусмотрена кнопка в разделе "Мои питомцы", которая выглядит в соответствии с рисунком 3.

Изображение выглядит как текст, Шрифт, снимок экрана, логотип

Автоматически созданное описание

Рисунок 3 – Кнопка для добавления питомца

При нажатии на кнопку открывается форма добавления питомца в модальном окне (рис. 4). В нем пользователь может ввести имя питомца, тип, пол, породу и возраст.

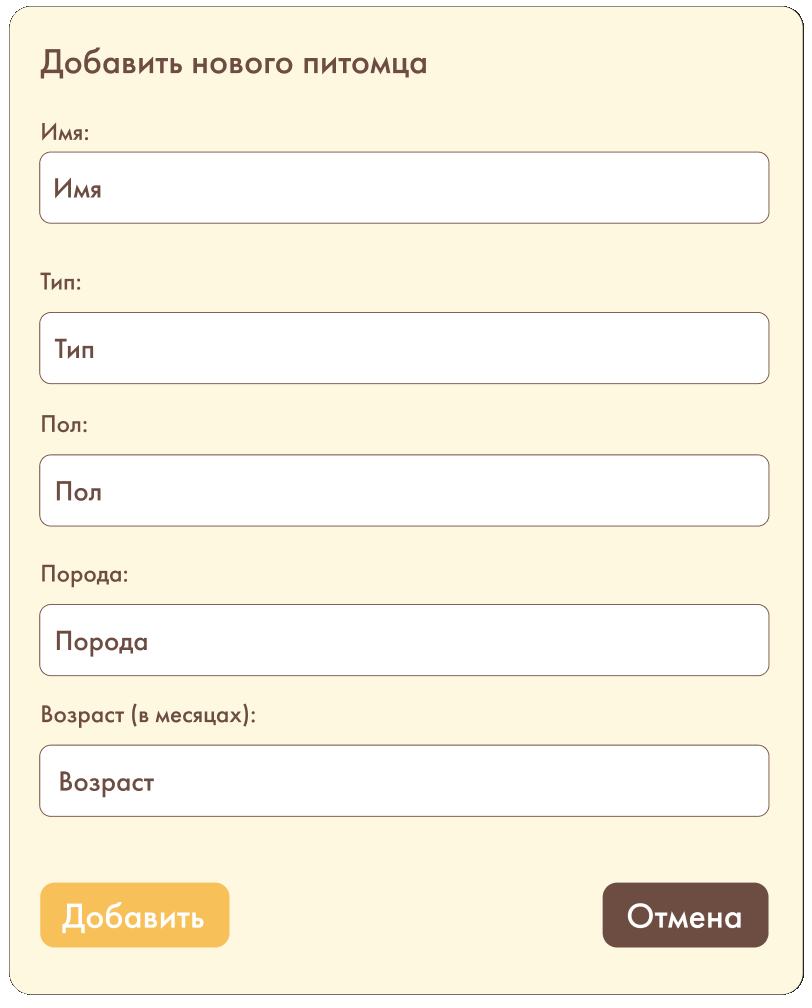


Рисунок 4 – Окно добавления питомца

В разделе "Мои питомцы" отображается список всех питомцев пользователя. Каждая карточка питомца включает информацию о его имени, типе, поле, породе и возрасте. Также на карточке доступны действия "Редактировать" и "Удалить", что позволяет пользователю управлять данными питомцев. Карточка питомца выглядит в соответствии с рисунком 5.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 5 - Карточка питомца

Для редактирования данных существующего питомца предусмотрено отдельное модальное окно, которое открывается при нажатии на кнопку "Редактировать" в карточке питомца (рис. 6). Это окно позволяет изменять существующие данные.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 6 – Окно редактирования данных о питомце

Интерфейс включает несколько ключевых элементов управления. Кнопка "Добавить питомца" расположена в разделе "Мои питомцы" и открывает форму для добавления нового питомца. В окнах для добавления питомца или редактирования данных о питомце имеются кнопки "Добавить/Сохранить" и "Отмена", которая закрывает окно без сохранения изменений.

Все элементы интерфейса организованы таким образом, чтобы пользователю было удобно управлять данными о питомцах и быстро находить необходимую информацию.

## 3.2. Схематичный макет интерфейса

Макет интерфейса представлен на рисунке 7.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 7 - Макет интерфейса

Макеты добавления питомца и изменения данных о питомце уже были представлены ранее.

# 4. Техническая реализация

Реализация выполнена с использованием современных веб-технологий, включая HTML, CSS, и JavaScript. Приложение построено на модульной архитектуре, что способствует удобству разработки и дальнейшей масштабируемости.

## 4.1. Общая структура проекта

Основой архитектуры проекта является паттерн MVP (Model-View-Presenter), который обеспечивает модульность, легкость в сопровождении и тестировании [2]. Эта архитектура разделяет приложение на три независимых компонента:

Model (Модель) – отвечает за управление данными, бизнес-логику и взаимодействие с источниками данных (в данном случае, API). В проекте модель обеспечивает обработку данных, таких как информация о питомцах [3].

View (Представление) – обеспечивает отображение данных пользователю и отвечает за пользовательский интерфейс. В проекте View реализован с использованием Java Script, который обеспечивает динамическую, интерактивную и адаптивную работу с пользователем [4].

Presenter (Презентер) – связывает Model и View, выступая посредником [8]. Он получает данные из модели, преобразует их в формат, удобный для представления, и отправляет обратно в View. Презентер также обрабатывает события от пользователя, такие как нажатия кнопок, ввод текста и взаимодействия с элементами интерфейса.

Полная и сокращённая структуры каталога представлены на рисунке 8:

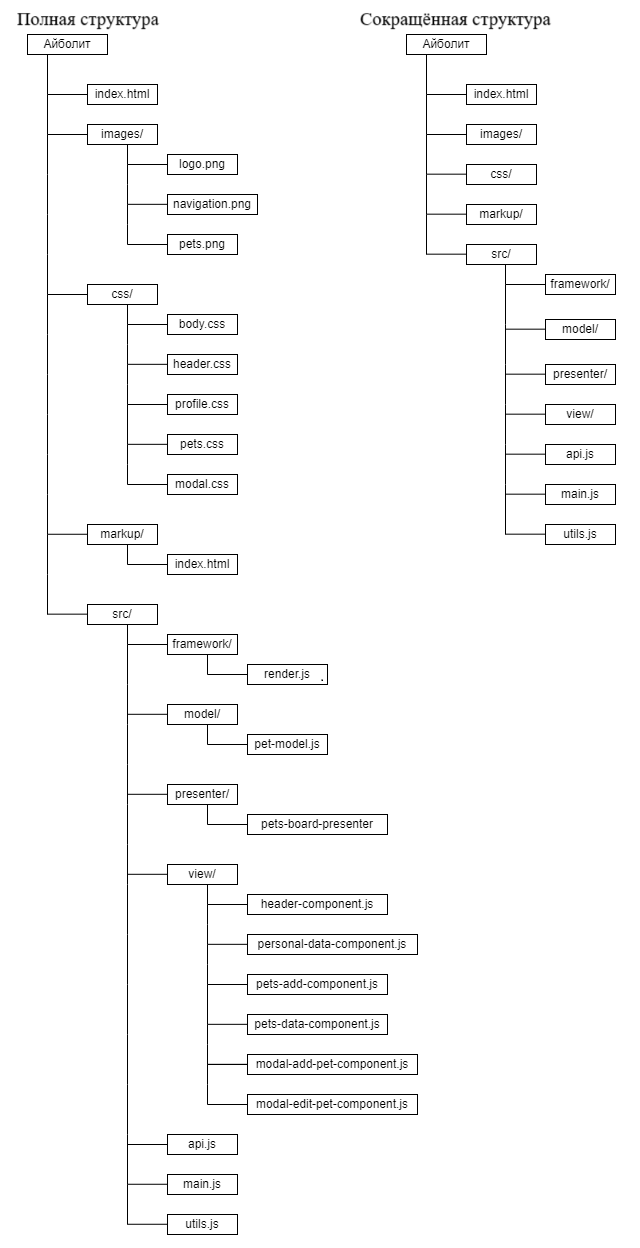


Рисунок 8 - Полная и сокращённая структуры проекта

4.2. Описание основных модулей

* index.html. Главный HTML-файл, содержащий структуру страницы. Включает секции для личных данных и данных о питомцах, подключает стили и JavaScript-файлы.
* render.js. Модуль для работы с DOM-элементами (Document Object Model). Содержит функции для создания HTML-элементов из шаблонов и добавления их на страницу.
* pet-model.js. Модуль для работы с данными о питомцах. Обеспечивает взаимодействие с API для получения, добавления, обновления и удаления данных. Содержит методы для работы с внутренним состоянием списка питомцев.
* pets-board-presenter.js. Презентер нужен для управления разделом данных о питомцах. Он отвечает за отображение карточек питомцев, обрабатывает события, такие как добавление, редактирование и удаление питомцев, управляет открытием и закрытием модальных окон.
* modal-add-pet-component.js и modal-edit-pet-component.js. Компоненты модального окна для добавления нового питомца и изменения данных о существующем питомце. Обеспечивает функционал открытия, закрытия и отправки данных.
* pets-data-component.js. Компонент для отображения карточки питомца. Включает информацию о питомце (имя, тип, пол, порода, возраст). Содержит кнопки для редактирования и удаления.
* api.js. Модуль для взаимодействия с сервером. Реализует функции для работы с API: получение, добавление, обновление и удаление данных о питомцах.
* main.js. Точка входа в приложение. Управляет основными рендерами компонентов (заголовок, личные данные, данные о питомцах). Создаёт экземпляр презентера PetsBoardPresenter и запускает его.

## 4.3. Описание функциональных модулей

1. Удаление карточки с питомцем

Функция реализована в классе PetsModel в файле pet-model.js и взаимодействует с API для удаления питомца по его id. Код функции:

async deletePet(petId) {

const deleted = await deletePet(petId);

if (deleted) {

this.boardpets = this.boardpets.filter((pet) => pet.id !== petId);

}

}

Метод deletePet выполняет удаление питомца через API, а затем обновляет локальный массив boardpets, исключая удалённого питомца. Удаление вызывается в обработчике событий кнопки "Удалить". Код вызова в презентере (файл pets-board-presenter.js):

async #handleDeletePet(pet) {

await this.model.deletePet(pet.id);

this.\_renderPets();

}

Удаление выполняется через API для синхронизации данных с сервером. Локальное обновление массива повышает производительность интерфейса. Логика вынесена в модель, чтобы разделить ответственность.

2. Реализация редактирования карточек питомцев

Метод updatePet находится в файле pet-model.js.

async updatePet(petId, updatedPet) {

const updated = await updatePet(petId, updatedPet);

if (updated) {

const index = this.boardpets.findIndex((pet) => pet.id === petId);

if (index !== -1) {

this.boardpets[index] = updated;

}

}

}

Метод обновляет данные питомца на сервере через API. Если обновление успешно, локальный массив boardpets модифицируется, заменяя старую карточку питомца обновлённой.

Код вызова в презентере (файл: pets-board-presenter.js):

async #handleEditPet(pet) {

if (this.modal) {

this.modal.removeElement();

this.modal = null;

}

this.modal = new ModalEditPetComponent(pet);

render(this.modal, document.body, RenderPosition.BEFOREEND);

const form = this.modal.getElement().querySelector('.modal\_\_form');

form.addEventListener('submit', async (event) => {

event.preventDefault();

const formData = new FormData(form);

const updatedPet = {

name: formData.get('name'),

type: formData.get('type'),

sex: formData.get('sex'),

breed: formData.get('breed'),

age: formData.get('age'),

};

await this.model.updatePet(pet.id, updatedPet);

this.\_renderPets();

this.#closeModal();

});

}

Редактирование реализовано через модальное окно, предоставляющее удобный интерфейс для изменения данных. Используется метод API PUT для актуализации информации на сервере, а обновление на клиенте обеспечивает отзывчивость.

3. Добавление нового питомца

Код функции (файл pet-model.js):

async addNewPet(newPet) {

const addedPet = await addPet(newPet);

if (addedPet) {

this.boardpets.push(addedPet);

}

}

Метод добавляет нового питомца на сервер и сохраняет его в локальный массив boardpets. Данные питомца передаются в API через метод POST.

Код вызова в презентере (файл pets-board-presenter.js):

async #handleAddPetClick() {

if (this.modal) {

this.#closeModal();

}

this.modal = new ModalAddPetComponent();

render(this.modal, document.body, RenderPosition.BEFOREEND);

const addButton=this.modal.getElement().querySelector('.modal\_\_button--add');

addButton.addEventListener('click', async () => {

const form = this.modal.getElement().querySelector('.modal\_\_form');

const formData = new FormData(form);

const newPet = {

id: generateUniqueId(),

name: formData.get('name'),

type: formData.get('type'),

sex: formData.get('sex'),

breed: formData.get('breed'),

age: formData.get('age'),

};

await this.model.addNewPet(newPet);

this.\_renderPets();

this.#closeModal();

});

}

Добавление через модальное окно делает процесс интуитивно понятным. Валидация формы на стороне клиента снижает вероятность ошибок [12]. Генерация уникального id позволяет добавлять питомцев без конфликтов.

# 5. Пошаговая реализация

Пошаговая реализация — это методический подход к выполнению сложной задачи, который разбивает её на отдельные этапы или шаги. Каждый шаг подробно описан, что позволяет легко следовать инструкции, даже если вы не обладаете глубокими знаниями в данной области. Такой подход широко используется в различных сферах, включая программирование, проектирование интерфейсов и создание веб-приложений.

## 5.1. Вёрстка страницы по макету

Для начала нужно сверстать по макету страницу. Макет уже был изображен на рисунке 7. HTML (HyperText Markup Language) — это язык разметки, используемый для создания и структурирования содержимого веб-страниц [10]. Он описывает структуру веб-документа с помощью различных элементов, называемых тегами. HTML не занимается стилизацией или функциональностью, а только задаёт структуру и содержание страницы. Файл: index.html:

<!DOCTYPE html>

<html lang="ru">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Личный кабинет</title>

</head>

<body>

    <header class="header">

        <div class="container">

            <div class="header\_\_inner">

                <div class="header\_\_logo"><img src="images/logo.png"></div>

                <nav>

                    <a class="nav\_\_link" href="#">ЛИЧНЫЙ КАБИНЕТ</a>

                    <a class="nav\_\_link" href="#">О НАС</a>

                    <a class="nav\_\_link" href="#">НОВОСТИ И АКЦИИ</a>

                    <a class="nav\_\_link" href="#">КОНТАКТЫ</a>

                    <a class="nav\_\_link" href="#">БЛОГ</a>

                </nav>

                <div class="navigation">

                    <a class="navigation\_\_link" href="#">КАЗАНЬ</a>

                </div>

            </div>

        </div>

    </header>

    <section class="main\_section">

        <div class="container">

            <section class="personal-data">

                <h2>Личные данные</h2>

                <div class="personal-data\_\_card">

                    <div class="personal-data\_\_info">

                        <div class="info-block">

                            <span class="label">имя</span>

                            <span class="value">Виктория</span>

                        </div>

                        <div class="info-block">

                            <span class="label">фамилия</span>

                            <span class="value">Иванова</span>

                        </div>

                        <div class="info-block">

                            <span class="label">отчество</span>

                            <span class="value">Александровна</span>

                        </div>

                        <div class="info-block">

                            <span class="label">телефон</span>

                            <span class="value">+7 999 598-12-45</span>

                        </div>

                    </div>

                    <div class="personal-data\_\_about">

                        <span class="label">О себе</span>

                        <p>Живу в Волгограде, очень люблю животных! Мечтаю завести много зверей и построить приют. Пока что у меня собака, кошка и рыбки :)</p>

                    </div>

                </div>

            </section>

            <section class="pets-data">

                <h2>Мои питомцы</h2>

                <div class="add\_new\_pets">

                    <img class="pets\_\_img" src="images/pets.png">

                    <h1 class="add">ДОБАВИТЬ НОВОГО ПИТОМЦА</h1>

                </div>

                <div class="pets-data\_\_card">

                    <div class="pets-data\_\_info">

                        <h2 class="pet\_name">Громопётр</h2>

                        <div class="info\_block">

                            <span class="label">тип</span>

                            <span class="value">Собака</span>

                        </div>

                        <div class="info-block">

                            <span class="label">пол</span>

                            <span class="value">Мужской</span>

                        </div>

                        <div class="info-block">

                            <span class="label">порода</span>

                            <span class="value">Без породы</span>

                        </div>

                        <div class="info-block">

                            <span class="label">возраст</span>

                            <span class="value">34 месяца(-ев)</span>

                        </div>

                    </div>

                    <div class="pets-data\_\_actions">

                        <button class="pets-data\_\_edit">Редактировать</button>

                        <button class="pets-data\_\_delete">Удалить</button>

                    </div>

            </section>

        </div>

    </section>

</body>

</html>

Теперь нужно добавить стилей. Стили нужны для управления внешним видом и стилем HTML-элементов на веб-странице. Создадим папку css и добавим туда файлы.

Общие стили, body.css:

body {

    margin: 0;

    font-size: 15px;

    line-height: 1.6;

}

h1, h2, h3, h4, h5, h6 {

    margin: 0;

}

h2 {

    word-break: break-all;

}

.container {

    width: 100%;

    max-width: 1760px;

    margin: 0 auto;

    position: relative;

}

.main\_section {

    width: 100%;

    height: 100vh;

    padding-top: 150px;

}

Добавим стилей для шапки страцицы, header.css:

.header {

    width: 100%;

    padding-top: 60px;

    position: absolute;

    top: 0;

    left: 0;

    right: 0;

    z-index: 1000;

}

.header\_\_inner {

    display: flex;

    justify-content: space-between;

    align-items: center;

}

.nav\_\_link {

    display: inline-block;

    vertical-align: top;

    margin: 0 30px;

    text-decoration: none;

    font-family: Futura New;

    font-size: 20px;

    font-weight: 700;

    line-height: 20px;

    text-align: left;

    color: #C3B9B5;

}

.nav\_\_link:hover {

    color: #59443F;

    text-decoration: none;

    transition: color 0.2s linear;

}

.navigation\_\_link {

    padding-right: 15px;

    font-family: Futura New;

    font-size: 20px;

    font-weight: 700;

    line-height: 20px;

    text-align: left;

    text-decoration: none;

    color: #FFC59E;

}

.header\_\_logo {

    cursor: pointer;

    padding-left: 15px;

}

Добавим стилей для личных данных, profile.css:

.profile\_\_container {

    justify-content: center;

    display: inline-block;

    position: relative;

    width: 1760px;

}

.personal-data {

    margin: 20px;

    font-family: 'Futura New', sans-serif;

    display: flex;

    justify-content: center;

    align-items: center;

    flex-direction: column;

    font-size: 24px;

    font-weight: 700;

    margin-bottom: 20px;

    text-align: left;

}

.personal-data\_\_card {

    display: flex;

    align-items: flex-start;

    padding: 20px;

    border: 1px solid #EAEAEA;

    border-radius: 10px;

    background-color: #fdfdfd;

    max-width: 800px;

    box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.05);

}

.personal-data\_\_info {

    display: flex;

    flex-direction: column;

    width: 350px;

}

.label {

    font-size: 12px;

    color: #A3A3A3;

    display: block;

}

.value {

    font-size: 18px;

    font-weight: 500;

    color: #000;

    margin-top: 5px;

}

.personal-data\_\_about {

    margin-top: 20px;

}

.personal-data\_\_about .label {

    font-size: 12px;

    color: #A3A3A3;

    display: block;

}

.personal-data\_\_about p {

    font-size: 14px;

    color: #080606;

    margin-top: 5px;

}

Добавим стилей для раздела «Мои питомцы», pets.css:

.my\_pets {

    font-family: 'Futura New', sans-serif;

    display: flex;

    justify-content: center;

    align-items: center;

    font-size: 32px;

    margin-bottom: 20px;

    text-align: left;

}

.add\_new\_pets {

    display: flex;

    align-items: center;

    width: 909px;

    height: 150px;

    background-color: rgb(248, 192, 89);

    border-radius: 20px;

    margin-bottom: 50px;

    cursor: pointer;

}

.add {

    font-family: Futura New;

    font-size: 30px;

    line-height: 30px;

    margin-left: 20px;

    text-align: left;

    color: white;

}

.pet\_name {

    font-weight: normal;

    font-size: 24px;

}

.pets-data {

    margin: 20px;

    font-family: 'Futura New', sans-serif;

    display: flex;

    justify-content: center;

    align-items: center;

    flex-direction: column;

}

.pets-data\_\_card {

    display: flex;

    align-items: flex-start;

    justify-content: center;

    padding: 20px;

    margin-bottom: 20px;

    border: 1px solid #EAEAEA;

    border-radius: 10px;

    background-color: #fdfdfd;

    max-width: 800px;

}

.pets-data\_\_about {

    padding-left: 20px;

}

.pets\_\_img {

    height: 95%;

}

.label {

    font-size: 12px;

    color: #A3A3A3;

    display: block;

}

.pets-data\_\_info {

    display: flex;

    flex-direction: column;

    width: 350px;

}

.pets-data\_\_empty {

    text-align: center;

    font-size: 18px;

    color: #555;

    padding: 20px;

    border: 2px dashed #ccc;

    border-radius: 8px;

    background-color: #f9f9f9;

    margin-top: 20px;

}

.loading {

    text-align: center;

    font-size: 18px;

    color: #555;

    margin-top: 20px;

}

.pets-data\_\_edit,

.pets-data\_\_delete {

    display: inline-block;

    padding: 10px 16px;

    margin-left: 10px;

    font-size: 14px;

    font-weight: bold;

    text-align: center;

    text-decoration: none;

    border-radius: 6px;

    border: none;

    cursor: pointer;

    transition: background-color 0.3s ease, color 0.3s ease, transform 0.2s ease;

}

.pets-data\_\_edit {

    background-color: rgb(248, 192, 89);

    color: #FFFFFF;

}

.pets-data\_\_edit:hover {

    background-color: #FB8C00;

    transform: scale(1.05);

}

.pets-data\_\_delete {

    background-color: #6D4C41;

    color: #FFFFFF;

}

.pets-data\_\_delete:hover {

    background-color: #5D4037;

    transform: scale(1.05);

}

Подключим все эти css файлы к странице. Для этого в тег <head> нужно добавить ссылки:

<link rel="stylesheet" href="css/header.css">

<link rel="stylesheet" href="css/body.css">

<link rel="stylesheet" href="css/profile.css">

<link rel="stylesheet" href="css/pets.css">

Также, нельзя забывать добавить нужные рисунки для сайта. Создадим папку images, в которую загрузим три рисунка: logo.png (рис. 9), navigation.png (рис. 10) и pets.png (рис. 11).



Рисунок 9 - Рисунок для лого шапки страницы



Рисунок 10 - Рисунок для навигации шапки страницы

Изображение выглядит как графическая вставка, символ, дизайн, иллюстрация

Автоматически созданное описание

Рисунок 11 - Рисунок для кнопки добавления питомца

## 5.2. Вёрстка на компоненты (MPV)

Прежде чем приступать к следующему этапу необходимо сделать копию файла index.html. В дальнейшем нужно будет использовать код разметки частями, и придется удалить содержимое файла. Создаём папку markup, в который копируем index.html.

В папке проекта создадим папку src. Теперь нужно добавить в папку src папку framework, а в ней создать файл render.js. Он нужен, чтобы создавать DOM-элементы на странице. Теперь добавим туда следующий код:

const RenderPosition = {

    BEFOREBEGIN: 'beforebegin',

    AFTERBEGIN: 'afterbegin',

    BEFOREEND: 'beforeend',

    AFTEREND: 'afterend',

  };

  function createElement(template) {

    const newElement = document.createElement('div');

    newElement.innerHTML = template;

    return newElement.firstElementChild;

  }

  function render(component, container, place = RenderPosition.BEFOREEND) {

  container.insertAdjacentElement(place, component.getElement());

  }

  export {RenderPosition, createElement, render};

Создадим модули (отдельные файлы) и опишите в них классы для генерации DOM-элементов. Эти модули — компоненты, или же View в рамках MVP. Поэтому:

* 1. для них нужно завести отдельную директорию src/view;
  2. именовать файлы стоит как существительные;
  3. для экспорта классов необходимо использовать экспорт по умолчанию.

В папке src/view создадим файлы: header-component.js, personal-data-component.js, pets-add-component.js, pets-data-component.js. В коде сосдаётся HTML-шаблон для определённого компонента страницы и класс компонента управляет им. Методы:

* getTemplate: возвращает шаблон шапки.
* getElement: создаёт и кэширует DOM-элемент из шаблона при первом вызове.
* removeElement: удаляет кэшированный элемент.

Файл header-component.js:

import {createElement} from '../framework/render.js';

function createHeaderComponentTemplate() {

    return (

        `<header class="header">

        <div class="container">

            <div class="header\_\_inner">

                <div class="header\_\_logo"><img src="images/logo.png"></div>

                <nav>

                    <a class="nav\_\_link" href="#">ЛИЧНЫЙ КАБИНЕТ</a>

                    <a class="nav\_\_link" href="#">О НАС</a>

                    <a class="nav\_\_link" href="#">НОВОСТИ И АКЦИИ</a>

                    <a class="nav\_\_link" href="#">КОНТАКТЫ</a>

                    <a class="nav\_\_link" href="#">БЛОГ</a>

                </nav>

                <div class="navigation">

                    <a class="navigation\_\_link" href="#">КАЗАНЬ</a>

                </div>

            </div>

        </div>

    </header>`

      );

}

export default class HeaderComponent {

  getTemplate() {

    return createHeaderComponentTemplate();

  }

  getElement() {

    if (!this.element) {

      this.element = createElement(this.getTemplate());

    }

    return this.element;

  }

  removeElement() {

    this.element = null;

  }

}

Файл personal-data-component.js:

import {createElement} from '../framework/render.js';

function createPersonalDataComponentTemplate() {

    return (

        `<section class="personal-data">

                <h2 class="personal\_data">Личные данные</h2>

                <div class="personal-data\_\_card">

                    <div class="personal-data\_\_info">

                        <div class="info-block">

                            <span class="label">имя</span>

                            <span class="value">Виктория</span>

                        </div>

                        <div class="info-block">

                            <span class="label">фамилия</span>

                            <span class="value">Иванова</span>

                        </div>

                        <div class="info-block">

                            <span class="label">отчество</span>

                            <span class="value">Александровна</span>

                        </div>

                        <div class="info-block">

                            <span class="label">телефон</span>

                            <span class="value">+7 999 598-12-45</span>

                        </div>

                    </div>

                    <div class="personal-data\_\_about">

                        <span class="label">О себе</span>

                        <p>Живу в Волгограде, очень люблю животных! Мечтаю завести много зверей и построить приют. Пока что у меня собака, кошка и рыбки :)</p>

                    </div>

                </div>

            </section>`

      );

}

export default class PersonalDataComponent {

  getTemplate() {

    return createPersonalDataComponentTemplate();

  }

  getElement() {

    if (!this.element) {

      this.element = createElement(this.getTemplate());

    }

    return this.element;

  }

  removeElement() {

    this.element = null;

  }

}

Файл pets-add-component.js:

import {createElement} from '../framework/render.js';

function createPetsAddComponentTemplate() {

    return (

        `<section class="pets-data">

            <h2 class="my\_pets">Мои питомцы</h2>

            <div class="add\_new\_pets">

                <img class="pets\_\_img" src="images/pets.png">

                <h1 class="add">ДОБАВИТЬ НОВОГО ПИТОМЦА</h1>

            </div>

        </section>`

      );

}

export default class PetsAddComponent {

  getTemplate() {

    return createPetsAddComponentTemplate();

  }

  getElement() {

    if (!this.element) {

      this.element = createElement(this.getTemplate());

    }

    return this.element;

  }

  removeElement() {

    this.element = null;

  }

}

Файл pets-data-component.js:

import { createElement } from '../framework/render.js';

function createPetsDataComponentTemplate() {

    return `

        <div class="pets-data\_\_card">

            <div class="pets-data\_\_info">

                <h2 class="pet\_name">Громопётр</h2>

                <div class="info\_block">

                    <span class="label">тип</span>

                    <span class="value">Собака</span>

                </div>

                <div class="info-block">

                    <span class="label">пол</span>

                    <span class="value">Мужской</span>

                </div>

                <div class="info-block">

                    <span class="label">порода</span>

                    <span class="value">Без породы</span>

                </div>

                <div class="info-block">

                    <span class="label">возраст</span>

                    <span class="value">34 месяца(-ев)</span>

                </div>

            </div>

            <div class="pets-data\_\_actions">

                <button class="pets-data\_\_edit">Редактировать</button>

                <button class="pets-data\_\_delete">Удалить</button>

            </div>

        </div>`;

}

export default class PetsDataComponent {

    constructor(pet) {

        this.pet = pet;

        this.element = null;

    }

    getTemplate() {

        return createPetsDataComponentTemplate(this.pet);

    }

    getElement() {

        if (!this.element) {

            this.element = createElement(this.getTemplate());

        }

        return this.element;

    }

    removeElement() {

        this.element = null;

    }

}

Теперь создадим файл src/main.js и добавим в него код:

import HeaderComponent from './view/header-component.js';

import PersonalDataComponent from './view/personal-data-component.js';

import PetsAddComponent from './view/pets-add-component.js';

import { render, RenderPosition } from './framework/render.js';

const bodyContainer = document.querySelector('.board-app');

const PersonalDataContainer = document.querySelector('.personal-data');

const PetsDataContainer = document.querySelector('.pets-data')

render(new HeaderComponent(), bodyContainer, RenderPosition.BEFOREBEGIN);

render(new PersonalDataComponent(), PersonalDataContainer);

render(new PetsAddComponent(), PetsDataContainer);

render(new PetsDataComponent(), PetsDataContainer);

Добавим строчку в тег <body> в index.html:

<script src="src/main.js" type="module"></script>

Файл index.html на данный выглядит таким образом:

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Личный кабинет</title>

<link rel="stylesheet" href="css/header.css">

<link rel="stylesheet" href="css/body.css">

<link rel="stylesheet" href="css/profile.css">

<link rel="stylesheet" href="css/pets.css">

</head>

<body class="board-app">

<section class="main\_section">

<div class="container">

<section class="personal-data">

</section>

<section class="pets-data">

</section>

</div>

</section>

<script src="src/main.js" type="module"></script>

</body>

</html>

После всех этих манипуляций страница выглядит в соответствии с рисунком 12:

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 12 - Внешний вид страницы на данный момент

Добавим данные для проекта. Для этого нужно добавить в папку src папку mock. В ней создадим файл pet.js:

export const pets = [

    {

        "name": "Громопетр",

        "type": "Собака",

        "sex": "Мужской",

        "breed": "Без породы",

        "age": "12",

        "id": "2"

    },

    {

        "name": "Барсик",

        "type": "Кот",

        "sex": "Мужской",

        "breed": "Без породы",

        "age": "34",

        "id": "1"

    }

  ]

Создадим модель для работы с данными. Для этого в папке src создадим папку model, а в ней файл pet-model.js. В нем будет находиться класс для работы с данными и метод для получения данных о питомцах. Файл pet-model.js:

import { pets } from '../mock/pet.js';

export default class PetsModel {

 boardpets = pets;

 getPets() {

   return this.boardpets;

 }

}

Теперь создадим презентер для доски с задачами. В папке src создадим папку presenter, в ней создадим файл pets-board-presenter.js:

import PetsDataComponent from '../view/pets-data-component.js';

import PetsModel from '../model/pet-model.js';

import { render, RenderPosition } from '../framework/render.js';

export default class PetsBoardPresenter {

    constructor(container) {

        this.container = container;

        this.petsModel = new PetsModel();

    }

    init() {

        const pets = this.petsModel.getPets();

        pets.forEach((pet) => {

            const petComponent = new PetsDataComponent(pet);

            render(petComponent, this.container, RenderPosition.BEFOREEND);

        });

    }

}

И обновим функцию в pets-data-component.js:

function createPetsDataComponentTemplate({ name, type, sex, breed, age }) {

    return `

        <div class="pets-data\_\_card">

            <div class="pets-data\_\_info">

                <h2 class="pet\_name">${name}</h2>

                <div class="info\_block">

                    <span class="label">тип</span>

                    <span class="value">${type}</span>

                </div>

                <div class="info-block">

                    <span class="label">пол</span>

                    <span class="value">${sex}</span>

                </div>

                <div class="info-block">

                    <span class="label">порода</span>

                    <span class="value">${breed}</span>

                </div>

                <div class="info-block">

                    <span class="label">возраст</span>

                    <span class="value">${age} месяца(-ев)</span>

                </div>

            </div>

            <div class="pets-data\_\_actions">

                <button class="pets-data\_\_edit">Редактировать</button>

                <button class="pets-data\_\_delete">Удалить</button>

            </div>

        </div>`;

}

Класс PetsBoardPresenter используется для отображения списка питомцев на доске. Он берёт данные из класса PetsModel (в файле pet-model.js) и рендерит компоненты питомцев (PetsDataComponent) в контейнер. В файле pets-data-component.js обновлена функция createPetsDataComponentTemplate, для генерации HTML-шаблона карточки питомца.

## 5.3. Добавление функционала

Теперь добавим функционал кнопкам «Добавить нового питомца», «Редактировать» и «Удалить. Для этого создадим новый компоненты, создадим файлы view/modal-add-pet-component.js и view/modal-edit-pet-component.js.

modal-add-pet-component.js:

import { createElement } from '../framework/render.js';

function createModalAddPetComponentTemplate() {

    return `

        <div class="modal">

            <div class="modal\_\_content">

                <button class="modal\_\_close" aria-label="Закрыть">×</button>

                <h2>Добавить нового питомца</h2>

                <form class="modal\_\_form">

                    <div>

                        <label>Имя:</label>

                        <input type="text" name="name" required>

                    </div>

                    <div>

                        <label>Тип:</label>

                        <input type="text" name="type" required>

                    </div>

                    <div>

                        <label>Пол:</label>

                        <select name="sex" required>

                            <option value="Мужской">Мужской</option>

                            <option value="Женский">Женский</option>

                        </select>

                    </div>

                    <div>

                        <label>Порода:</label>

                        <input type="text" name="breed">

                    </div>

                    <div>

                        <label>Возраст (в месяцах):</label>

                        <input type="number" name="age" required>

                    </div>

                    <div class="modal\_\_buttons">

                        <button type="button" class="modal\_\_button modal\_\_button--add">Добавить</button>

<button type="button" class="modal\_\_button modal\_\_button--cancel">Отмена</button>

                    </div>

                </form>

            </div>

        </div>;`

}

export default class ModalAddPetComponent {

    constructor() {

        this.element = null;

    }

    getTemplate() {

        return createModalAddPetComponentTemplate();

    }

    getElement() {

        if (!this.element) {

            this.element = createElement(this.getTemplate());

        }

        return this.element;

    }

    removeElement() {

        this.element = null;

    }

    open() {

        this.getElement().classList.add('open');

    }

    close() {

        this.getElement().classList.remove('open');

    }

}

modal-edit-pet-component.js:

import { createElement } from '../framework/render.js';

function createModalEditPetComponentTemplate({ name, type, sex, breed, age }) {

    return `

        <div class="modal">

            <div class="modal\_\_content">

                <button class="modal\_\_close" aria-label="Закрыть">×</button>

                <h2>Редактировать питомца</h2>

                <form class="modal\_\_form">

                    <div>

                        <label>Имя:</label>

                        <input type="text" name="name" value="${name}" required>

                    </div>

                    <div>

                        <label>Тип:</label>

                        <input type="text" name="type" value="${type}" required>

                    </div>

                    <div>

                        <label>Пол:</label>

                        <select name="sex" required>

                            <option value="Мужской" ${sex === 'Мужской' ? 'selected' : ''}>Мужской</option>

                            <option value="Женский" ${sex === 'Женский' ? 'selected' : ''}>Женский</option>

                        </select>

                    </div>

                    <div>

                        <label>Порода:</label>

                        <input type="text" name="breed" value="${breed}">

                    </div>

                    <div>

                        <label>Возраст (в месяцах):</label>

                        <input type="number" name="age" value="${age}" required>

                    </div>

                    <div class="modal\_\_buttons">

                        <button type="submit" class="modal\_\_button modal\_\_button--add">Сохранить</button>

                        <button type="button" class="modal\_\_button modal\_\_button--cancel">Отмена</button>

                    </div>

                </form>

            </div>

        </div>`;

}

export default class ModalEditPetComponent {

    constructor(pet) {

        this.pet = pet;

        this.element = null;

    }

    getTemplate() {

        return createModalEditPetComponentTemplate(this.pet);

    }

    getElement() {

        if (!this.element) {

            this.element = createElement(this.getTemplate());

        }

        return this.element;

    }

    removeElement() {

        this.element = null;

    }

    open() {

        this.getElement().classList.add('open');

    }

    close() {

        this.getElement().classList.remove('open');

    }

}

Создадим вспомогательный файл src/utils.js, в который поместим генератор id карточек питомцев:

export function generateUniqueId() {

    return String(Date.now());

}

Изменим pets-data-component.js:

import { createElement } from '../framework/render.js';

function createPetsDataComponentTemplate({ name, type, sex, breed, age }) {

    return `

        <div class="pets-data\_\_card">

            <div class="pets-data\_\_info">

                <h2 class="pet\_name">${name}</h2>

                <div class="info\_block">

                    <span class="label">тип</span>

                    <span class="value">${type}</span>

                </div>

                <div class="info-block">

                    <span class="label">пол</span>

                    <span class="value">${sex}</span>

                </div>

                <div class="info-block">

                    <span class="label">порода</span>

                    <span class="value">${breed}</span>

                </div>

                <div class="info-block">

                    <span class="label">возраст</span>

                    <span class="value">${age} месяца(-ев)</span>

                </div>

            </div>

            <div class="pets-data\_\_actions">

                <button class="pets-data\_\_edit">Редактировать</button>

                <button class="pets-data\_\_delete">Удалить</button>

            </div>

        </div>`;

}

export default class PetsDataComponent {

    constructor(pet) {

        this.pet = pet;

        this.element = null;

    }

    getTemplate() {

        return createPetsDataComponentTemplate(this.pet);

    }

    getElement() {

        if (!this.element) {

            this.element = createElement(this.getTemplate());

        }

        return this.element;

    }

    removeElement() {

        this.element = null;

    }

    setEditHandler(callback) {

        this.getElement().querySelector('.pets-data\_\_edit')

            .addEventListener('click', callback);

    }

    setDeleteHandler(callback) {

        this.getElement().querySelector('.pets-data\_\_delete')

            .addEventListener('click', callback);

    }

}

Изменим презентер, добавим функции добавления нового питомца, изменения данных существующего и удаления питомца из списка. Файл pets-board-presenter.js:

import { render, RenderPosition } from '../framework/render.js';

import { generateUniqueId } from '../utils.js';

import PetsDataComponent from '../view/pets-data-component.js';

import ModalAddPetComponent from '../view/modal-add-pet-component.js';

import ModalEditPetComponent from '../view/modal-edit-pet-component.js';

import PetsModel from '../model/pet-model.js';

export default class PetsBoardPresenter {

    constructor(container) {

        this.container = container;

        this.model = new PetsModel();

        this.petsContainer = null;

        this.modal = null;

    }

    init() {

        this.#renderBaseStructure();

        const addPetButton = document.querySelector('.add\_new\_pets');

        addPetButton.addEventListener('click', this.#handleAddPetClick.bind(this));

        this.\_renderPets();

    }

    #renderBaseStructure() {

        const petsSection = `

            <section class="pets-data">

                <h2 class="my\_pets">Мои питомцы</h2>

                <div class="add\_new\_pets">

                    <img class="pets\_\_img" src="images/pets.png">

                    <h1 class="add">ДОБАВИТЬ НОВОГО ПИТОМЦА</h1>

                </div>

                <div class="pets-data\_\_list"></div>

            </section>`;

        this.container.innerHTML = petsSection;

        this.petsContainer = this.container.querySelector('.pets-data\_\_list');

    }

    \_renderPets() {

        this.petsContainer.innerHTML = '';

        const pets = this.model.getPets();

        pets.forEach((pet) => {

            const petCard = this.#createPetCard(pet);

            render(petCard, this.petsContainer);

        });

    }

    #createPetCard(pet) {

        const petCard = new PetsDataComponent(pet);

        petCard.setEditHandler(() => this.#handleEditPet(pet));

        petCard.setDeleteHandler(() => this.#handleDeletePet(pet));

        return petCard;

    }

    #handleEditPet(pet) {

        if (!this.modal) {

            this.modal = new ModalEditPetComponent(pet);

            render(this.modal, document.body, RenderPosition.BEFOREEND);

            const form = this.modal.getElement().querySelector('.modal\_\_form');

            const closeButton = this.modal.getElement().querySelector('.modal\_\_close');

            const cancelButton = this.modal.getElement().querySelector('.modal\_\_button--cancel');

            form.addEventListener('submit', (event) => {

                event.preventDefault();

                const formData = new FormData(form);

                pet.name = formData.get('name');

                pet.type = formData.get('type');

                pet.sex = formData.get('sex');

                pet.breed = formData.get('breed');

                pet.age = formData.get('age');

                this.\_renderPets();

                this.modal.close();

                this.modal = null;

            });

            closeButton.addEventListener('click', () => this.#closeModal());

            cancelButton.addEventListener('click', () => this.#closeModal());

        }

        this.modal.open();

    }

    #handleDeletePet(pet) {

        this.model.boardpets = this.model.boardpets.filter((item) => item.id !== pet.id);

        this.\_renderPets();

    }

    #handleAddPetClick() {

        if (!this.modal) {

            this.modal = new ModalAddPetComponent();

            render(this.modal, document.body, RenderPosition.BEFOREEND);

            const closeButton = this.modal.getElement().querySelector('.modal\_\_close');

            const addButton = this.modal.getElement().querySelector('.modal\_\_button--add');

            const cancelButton = this.modal.getElement().querySelector('.modal\_\_button--cancel');

            closeButton.addEventListener('click', this.#handleCancelButtonClick.bind(this));

            addButton.addEventListener('click', this.#handleAddButtonClick.bind(this));

            cancelButton.addEventListener('click', this.#handleCancelButtonClick.bind(this));

        }

        this.modal.open();

    }

    #handleCancelButtonClick() {

        this.#closeModal();

    }

    #handleAddButtonClick() {

        const form = this.modal.getElement().querySelector('.modal\_\_form');

        const formData = new FormData(form);

        const newPet = {

            id: generateUniqueId(),

            name: formData.get('name'),

            type: formData.get('type'),

            sex: formData.get('sex'),

            breed: formData.get('breed'),

            age: formData.get('age'),

        };

        this.model.boardpets.push(newPet);

        this.\_renderPets();

        this.#closeModal();

    }

    #closeModal() {

        if (this.modal) {

            this.modal.close();

            this.modal = null;

        }

    }

}

Добавим стилей для модальных окон, для этого создадим css/modal.css:

.modal {

    position: fixed;

    top: 0;

    left: 0;

    width: 100%;

    height: 100%;

    background-color: rgba(0, 0, 0, 0.7);

    display: flex;

    justify-content: center;

    align-items: center;

    z-index: 1000;

    opacity: 0;

    visibility: hidden;

    transition: visibility 0s linear 0.25s, opacity 0.25s ease-in-out;

}

.modal\_\_content {

    background: #ffffff;

    padding: 30px;

    border-radius: 10px;

    max-width: 700px;

    width: 90%;

    box-shadow: 0 4px 20px rgba(0, 0, 0, 0.3);

    position: relative;

}

.modal.open {

    opacity: 1;

    visibility: visible;

    transition-delay: 0s;

}

.modal\_\_buttons {

    display: flex;

    justify-content: space-between;

    margin-top: 20px;

}

.modal\_\_button {

    padding: 10px 20px;

    border: none;

    cursor: pointer;

    border-radius: 4px;

    font-size: 16px;

}

.modal\_\_button--add {

    background-color: #4caf50;

    color: white;

}

.modal\_\_button--cancel {

    background-color: #f44336;

    color: white;

}

.modal\_\_close {

    position: absolute;

    top: 10px;

    right: 10px;

    font-size: 18px;

    font-weight: bold;

    cursor: pointer;

    background: none;

    border: none;

}

.modal {

    display: none;

    position: fixed;

    top: 0;

    left: 0;

    width: 100%;

    height: 100%;

    background-color: rgba(0, 0, 0, 0.5);

    justify-content: center;

    align-items: center;

    z-index: 1000;

}

.modal.open {

    display: flex;

}

.modal\_\_content {

    background-color: #FFF8E1;

    border-radius: 12px;

    padding: 20px;

    width: 400px;

    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);

    text-align: left;

    position: relative;

}

.modal\_\_close {

    position: absolute;

    top: 10px;

    right: 10px;

    background: none;

    border: none;

    font-size: 18px;

    font-weight: bold;

    color: #6D4C41;

    cursor: pointer;

    transition: color 0.3s ease;

}

.modal\_\_close:hover {

    color: #D84315;

}

.modal\_\_content h2 {

    color: #6D4C41;

    margin-bottom: 20px;

    font-size: 20px;

}

.modal\_\_form div {

    margin-bottom: 15px;

}

.modal\_\_form label {

    display: block;

    font-size: 14px;

    margin-bottom: 5px;

    color: #6D4C41;

}

.modal\_\_form input,

.modal\_\_form select {

    width: 100%;

    padding: 10px;

    padding-left: 10px;

    border: 1px solid #D7CCC8;

    border-radius: 6px;

    font-size: 14px;

    color: #6D4C41;

    background-color: #FFF;

    box-sizing: border-box;

}

.modal\_\_buttons {

    display: flex;

    justify-content: space-between;

    margin-top: 20px;

}

.modal\_\_button {

    padding: 10px 20px;

    font-size: 14px;

    font-weight: bold;

    border-radius: 6px;

    border: none;

    cursor: pointer;

    transition: background-color 0.3s ease, transform 0.2s ease;

}

.modal\_\_button--add {

    background-color: #FFA726;

    color: #FFFFFF;

}

.modal\_\_button--add:hover {

    background-color: #FB8C00;

    transform: scale(1.05);

}

.modal\_\_button--cancel {

    background-color: #6D4C41;

    color: #FFFFFF;

}

.modal\_\_button--cancel:hover {

    background-color: #5D4037;

    transform: scale(1.05);

}

## 5.4. Работа с сетью

Теперь свяжем страницу с сервером. Для этого я использую MockAPI. Нельзя забывать, что нужно указать URL вашего mockAPI в переменной API\_URL. Этот код предоставляет функции для взаимодействия с Mock API, обеспечивая базовые CRUD-операции (создание, чтение, обновление, удаление) для данных о питомцах [6]. Функция fetchPets отправляет GET-запрос на API\_URL для получения списка питомцев. Если запрос успешен, возвращает данные в формате JSON. При ошибке логирует её и возвращает пустой массив. Функция addPet отправляет POST-запрос на API\_URL с данными нового питомца в формате JSON. Функция updatePet отправляет PUT-запрос на API\_URL/{petId} для обновления информации о питомце. Передаёт обновлённые данные (updatedPet) в теле запроса. Возвращает обновлённый объект питомца. Функция deletePet отправляет DELETE-запрос на API\_URL/{petId} для удаления питомца по его ID и возвращает ответ сервера [9].

Создадим файл src/api.js:

const API\_URL = ' Здесь нужно указать URL вашего mockAPI';

export async function fetchPets() {

    try {

        const response = await fetch(API\_URL);

        if (!response.ok) {

            throw new Error('Ошибка при получении данных');

        }

        return await response.json();

    } catch (error) {

        console.error('Ошибка при получении питомцев:', error);

        return [];

    }

}

export async function addPet(newPet) {

    try {

        const response = await fetch(API\_URL, {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json',

            },

            body: JSON.stringify(newPet),

        });

        if (!response.ok) {

            throw new Error('Ошибка при добавлении питомца');

        }

        return await response.json();

    } catch (error) {

        console.error('Ошибка при добавлении питомца:', error);

    }

}

export async function updatePet(petId, updatedPet) {

    try {

        const response = await fetch(`${API\_URL}/${petId}`, {

            method: 'PUT',

            headers: {

                'Content-Type': 'application/json',

            },

            body: JSON.stringify(updatedPet),

        });

        if (!response.ok) {

            throw new Error('Ошибка при обновлении питомца');

        }

        return await response.json();

    } catch (error) {

        console.error('Ошибка при обновлении питомца:', error);

    }

}

export async function deletePet(petId) {

    try {

        const response = await fetch(`${API\_URL}/${petId}`, {

            method: 'DELETE',

        });

        if (!response.ok) {

            throw new Error('Ошибка при удалении питомца');

        }

        return await response.json();

    } catch (error) {

        console.error('Ошибка при удалении питомца:', error);

    }

}

Обновим pet-model.js:

import { fetchPets, addPet, updatePet, deletePet } from '../api.js';

export default class PetsModel {

    constructor() {

        this.boardpets = [];

    }

    async init() {

        this.boardpets = await fetchPets();

    }

    async addNewPet(newPet) {

        const addedPet = await addPet(newPet);

        if (addedPet) {

            this.boardpets.push(addedPet);

        }

    }

    async updatePet(petId, updatedPet) {

        const updated = await updatePet(petId, updatedPet);

        if (updated) {

            const index = this.boardpets.findIndex((pet) => pet.id === petId);

            if (index !== -1) {

                this.boardpets[index] = updated;

            }

        }

    }

    async deletePet(petId) {

        const deleted = await deletePet(petId);

        if (deleted) {

            this.boardpets = this.boardpets.filter((pet) => pet.id !== petId);

        }

    }

    getPets() {

        return this.boardpets;

    }

}

Так же изменим презентер. Добавляем асинхронную загрузка данных при инициализации. Использование асинхронного метода init для загрузки данных питомцев с сервера. Временно показывается индикатор загрузки через метод #renderLoading(). После завершения загрузки данные отображаются. Меняем операции с данными (добавление, редактирование, удаление) с использованием асинхронности. Используется await для ожидания завершения операций перед обновлением интерфейса, это гарантирует, что интерфейс обновляется только после завершения операций [13].

Это позволяет загружать данные асинхронно, делая интерфейс более отзывчивым и улучшает пользовательский опыт за счёт отображения состояния загрузки.

Файл pets-board-presenter.js:

import { render, RenderPosition } from '../framework/render.js';

import { generateUniqueId } from '../utils.js';

import PetsDataComponent from '../view/pets-data-component.js';

import ModalAddPetComponent from '../view/modal-add-pet-component.js';

import ModalEditPetComponent from '../view/modal-edit-pet-component.js';

import PetsModel from '../model/pet-model.js';

export default class PetsBoardPresenter {

    constructor(container) {

        this.container = container;

        this.model = new PetsModel();

        this.petsContainer = null;

        this.modal = null;

        this.isLoading = true;

    }

    async init() {

        this.#renderBaseStructure();

        this.#renderLoading();

        await this.model.init();

        this.isLoading = false;

        this.\_renderPets();

        const addPetButton = document.querySelector('.add\_new\_pets');

        addPetButton.addEventListener('click', this.#handleAddPetClick.bind(this));

    }

    #renderBaseStructure() {

        const petsSection =

            `<section class="pets-data">

                <h2 class="my\_pets">Мои питомцы</h2>

                <div class="add\_new\_pets">

                    <img class="pets\_\_img" src="images/pets.png">

                    <h1 class="add">ДОБАВИТЬ НОВОГО ПИТОМЦА</h1>

                </div>

                <div class="pets-data\_\_list"></div>

            </section>`;

        this.container.innerHTML = petsSection;

        this.petsContainer = this.container.querySelector('.pets-data\_\_list');

    }

    #renderLoading() {

        this.petsContainer.innerHTML = `

            <div class="loading">

                <p>Загрузка...</p>

            </div>

        `;

    }

    \_renderPets() {

        if (this.isLoading) return;

        this.petsContainer.innerHTML = '';

        const pets = this.model.getPets();

        if (pets.length === 0) {

            this.#renderNoPets();

        } else {

            pets.forEach((pet) => {

                const petCard = this.#createPetCard(pet);

                render(petCard, this.petsContainer);

            });

        }

    }

    #renderNoPets() {

        this.petsContainer.innerHTML = `

            <div class="no-pets">

                <p>У вас пока нет питомцев.</p>

            </div>

        `;

    }

    #createPetCard(pet) {

        const petCard = new PetsDataComponent(pet);

        petCard.setEditHandler(() => this.#handleEditPet(pet));

        petCard.setDeleteHandler(() => this.#handleDeletePet(pet));

        return petCard;

    }

    async #handleEditPet(pet) {

        if (this.modal) {

            this.modal.removeElement();

            this.modal = null;

        }

        this.modal = new ModalEditPetComponent(pet);

        render(this.modal, document.body, RenderPosition.BEFOREEND);

        const form = this.modal.getElement().querySelector('.modal\_\_form');

        const closeButton = this.modal.getElement().querySelector('.modal\_\_close');

        const cancelButton = this.modal.getElement().querySelector('.modal\_\_button--cancel');

        form.addEventListener('submit', async (event) => {

            event.preventDefault();

            const formData = new FormData(form);

            const updatedPet = {

                name: formData.get('name'),

                type: formData.get('type'),

                sex: formData.get('sex'),

                breed: formData.get('breed'),

                age: formData.get('age'),

            };

            await this.model.updatePet(pet.id, updatedPet);

            this.\_renderPets();

            this.#closeModal();

        });

        closeButton.addEventListener('click', () => this.#closeModal());

        cancelButton.addEventListener('click', () => this.#closeModal());

        this.modal.open();

    }

    async #handleDeletePet(pet) {

        await this.model.deletePet(pet.id);

        this.\_renderPets();

    }

    async #handleAddPetClick() {

        if (this.modal) {

            this.#closeModal();

        }

        this.modal = new ModalAddPetComponent();

        render(this.modal, document.body, RenderPosition.BEFOREEND);

        const closeButton = this.modal.getElement().querySelector('.modal\_\_close');

        const addButton = this.modal.getElement().querySelector('.modal\_\_button--add');

        const cancelButton = this.modal.getElement().querySelector('.modal\_\_button--cancel');

        closeButton.addEventListener('click', this.#handleCancelButtonClick.bind(this));

        addButton.addEventListener('click', this.#handleAddButtonClick.bind(this));

        cancelButton.addEventListener('click', this.#handleCancelButtonClick.bind(this));

        this.modal.open();

    }

    async #handleAddButtonClick() {

        const form = this.modal.getElement().querySelector('.modal\_\_form');

        const formData = new FormData(form);

        const newPet = {

            id: generateUniqueId(),

            name: formData.get('name'),

            type: formData.get('type'),

            sex: formData.get('sex'),

            breed: formData.get('breed'),

            age: formData.get('age'),

        };

        await this.model.addNewPet(newPet);

        this.\_renderPets();

        this.#closeModal();

    }

    #handleCancelButtonClick() {

        this.#closeModal();

    }

    #closeModal() {

        if (this.modal) {

            this.modal.close();

            this.modal = null;

        }

    }

}

Теперь можно удалить папку mock с файлом pet.js, потому что у нас данные приходят с сервера, а локальные данные нам больше не нужны.

На рисунке 13 изображён внешний вид страницы.

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 13 - Внешний вид страницы

На рисунке 14 изображён индикатор загрузки данных с сервера.

Изображение выглядит как текст, снимок экрана, Шрифт, логотип

Автоматически созданное описание

Рисунок 14 - Индикатор загрузки данных с сервера

На рисунке 15 изображена заглушка, если питомцы в списке отсутствуют.

Изображение выглядит как текст, снимок экрана, Шрифт, логотип

Автоматически созданное описание

Рисунок 15 - Заглушка, при отсутствии питомцев в списке

На рисунках 16 и 17 соответственно представлены модальные окна добавления нового питомца и редактирования данных о питомце соответственно.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 16 - Модальное окно добавления нового питомца

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 17 - Модальное окно редактирования данных о питомце

# ЗАКЛЮЧЕНИЕ

В результате данного курсового проекта на тему «Разработка системы учета питомцев» была выполнена разработка полноценного веб-приложения для управления данными о питомцах. В ходе работы был создан функционал для добавления, редактирования, удаления и отображения информации о питомцах, включая интеграцию с внешним API для хранения данных.

В рамках проекта было осуществлено проектирование пользовательского интерфейса с использованием современных технологий, таких как HTML, CSS и JavaScript, что позволило создать интуитивно понятный и удобный интерфейс. Для управления данными использовалась модель MVP (Model-View-Presenter), что позволило разделить логику приложения и интерфейс. Также были реализованы модальные окна для добавления и редактирования питомцев, что улучшило взаимодействие с пользователем.

Кроме того, в ходе работы был улучшен процесс взаимодействия с сервером для получения, добавления, редактирования и удаления данных, что позволяет создавать динамичные и отзывчивые компоненты на странице.

В ходе работы над проектом были приобретены навыки работы с современными фреймворками, а также углублены знания в области работы с веб-технологиями и созданием интерфейсов. Были освоены методы асинхронного взаимодействия с сервером, а также принципы создания и использования модульных компонентов, что обеспечило масштабируемость и гибкость приложения.

Проект достиг поставленных целей, и в будущем возможны дополнительные улучшения, такие как добавление дополнительных функций, например фильтрации и поиска питомцев, и расширения функционала с учетом потребностей пользователей.

# СПИСОК ЛИТЕРАТУРЫ

1. Грэм, Д. Build Web APIs with Node.js / Д. Грэм – Москва: Вильямс, 2023 – 384 с.
2. Дэвидсон, Дж. JavaScript: The Good Parts / Дж. Дэвидсон – Санкт-Петербург: БХВ-Петербург, 2022 – 228 с.
3. Захарова, И. JavaScript для профессионалов / И. Захарова – Москва: ДМК Пресс, 2022 – 560 с.
4. Куинн, Д. Learning Node.js / Д. Куинн – Санкт-Петербург: Питер, 2023 – 432 с.
5. Ли, Б. Mastering CSS Grid and Flexbox / Б. Ли – Москва: О'Рейли, 2022 – 332 с.
6. Максвелл, М. API Design Patterns / М. Максвелл – Москва: О'Рейли, 2022 – 432 с.
7. Мейер, Э. CSS: The Definitive Guide / Э. Мейер – Санкт-Петербург: Питер, 2022 – 824 с.
8. Моррисон, Б. Modern JavaScript for the Impatient / Б. Моррисон – Москва: О'Рейли, 2023 – 336 с.
9. Ричардсон, Ч. RESTful Web APIs / Ч. Ричардсон, С. Вулфсон – Москва: БХВ-Петербург, 2023 – 512 с.
10. Рэйчел, У. Responsive Web Design with HTML5 and CSS3 / Рэйчел, У. – Санкт-Петербург: БХВ-Петербург, 2022 – 576 с.
11. Флеминг, Дж. JavaScript: The Definitive Guide / Дж. Флеминг – Санкт-Петербург: Питер, 2023 – 1043 с.
12. Флорес, Л. Learning JavaScript: The Easy Way / Л. Флорес – Санкт-Петербург: Вильямс, 2023 – 340 с.
13. Шмидт, М. Fullstack JavaScript: Learn Backbone.js, Node.js and MongoDB / М. Шмидт – Москва: Вильямс, 2023 – 376 с.
14. Шнайдер, М. HTML & CSS: Design and Build Websites / М. Шнайдер – Москва: Вильямс, 2023 – 432 с.
15. Эмс, Дж. CSS Secrets: Better Solutions to Everyday Web Design Problems / Дж. Эмс – Санкт-Петербург: Питер, 2022 – 432 с.