САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа №2

Выполнила:

Гусейнова Марьям

Группа ФРЭНД 2.2

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Варианты остаются прежними. Теперь нужно привязать то, что было сделано в ЛР1, к внешнему API средствами fetch/axios/xhr. Реализовать моковое API средствами JSON-сервера и подключить к нему авторизацию.

Ход работы

Этап 1. Подготовка мокового API.

Первым шагом был создан файл db.json, содержащий структуру данных для мокового сервера. В файле определены три основные сущности:

- пользователи (users) с полями: id, email, пароль, имя, прогресс тренировок и план занятий;
- тренировки (workouts) с детальной информацией: уровень сложности, тип, продолжительность, видео и инструкции;
- статьи блога (blogPosts) с заголовками, текстом и изображениями.

Этап 2. Реорганизация структуры приложения

Была изменена архитектура папок проекта: файл db.json размещен в корне, а файлы фронтенда (index.html, style.css, scripts.js) – в папке public. Это соответствует стандартной структуре проектов с JSON Server, где db.json является корневым файлом базы данных.

Этап 3. Рефакторинг HTML-разметки

Из файла index.html были удалены статические данные:

- карточки блога на главной странице;
- карточки тренировок на странице поиска;
- статическое содержимое аккордеона плана тренировок в личном кабинете;
- жестко заданный список инструкций в модальном окне тренировки.

Все данные теперь должны динамически загружаться с сервера. Пустые контейнеры остались в разметке для последующего заполнения через JavaScript.

Этап 4. Переработка JavaScript-логики

Основные изменения в файле scripts.js:

4.1. Настройка подключения к API

Добавлена константа API_URL = 'http://localhost:3000', определяющая базовый адрес мокового сервера. Это централизованная настройка, позволяющая легко изменить адрес API при необходимости.

4.2. Модернизация системы авторизации

Функции входа и регистрации были полностью переработаны:

- handleLogin() теперь выполняет асинхронный GET-запрос к эндпоинту /users?email=${email} для поиска пользователя по email;
- после нахождения пользователя происходит проверка пароля на клиенте;
- при успешной аутентификации сохраняются userId и userName в localStorage для управления сессией;
- добавлена обработка ошибок сети и неверных учетных данных через унифицированную функцию showError();
- handleRegistration() выполняет два запроса: сначала проверяет уникальность email, затем отправляет POST-запрос для создания нового пользователя;
- новый пользователь создается со структурой по умолчанию для прогресса и плана тренировок.

Переход с клиентского хранения в localStorage на серверное хранение в REST API требует выполнения HTTP-запросов и обработки асинхронных операций.

4.3. Динамическая загрузка данных

Реализованы три ключевые функции загрузки:

- loadDashboardData(userId) – загружает данные конкретного пользователя по ID для отображения в личном кабинете;
- loadWorkouts() – получает все тренировки с сервера и отображает их с помощью функции-шаблона renderWorkoutCard();
- loadBlogPosts() – загружает статьи блога для главной страницы.

Данные должны запрашиваться с сервера при каждом посещении соответствующих страниц, что имитирует работу реального приложения.

4.4. Улучшение системы фильтрации

Функция applyFilters() модифицирована для работы с динамически загруженными карточками. Добавлен параметр showAlert, позволяющий контролировать показ уведомлений о результатах фильтрации.

4.5. Детализация тренировок

Функция showTrainingDetails() теперь выполняет GET-запрос за конкретной тренировкой по ее ID и динамически заполняет модальное окно полученными данными (видео, инструкции, описание).

Весь код, работающий с API, переведен на асинхронную модель с использованием async/await, что обеспечивает корректную работу с сетевые задержками.

Вывод

В результате выполнения лабораторной работы №2 было успешно интегрировано фронтенд-приложение "FitLife" с внешним REST API, реализованным на базе JSON Server.