САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Гуторова Инна

Группа K3441

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Варианты остаются прежними. Теперь Вам нужно привязать то, что Вы делали в ЛР1 к внешнему API средствами fetch/axios/xhr. Реализуйте моковое API средствами JSON-сервера и подключите к нему авторизацию, как в примерах, которые мы рассматривали в рамках тем "Иммитация работы с API".

Например, для приложения для просмотра прогнозов погоды задание выглядит следующим образом:

Реализовать получение погоды (прогноз на ближайшие 7 дней) из открытого API OpenWeatherMap, взависимости от геолокации пользователя. Реализовать вывод полученного прогноза в виде 7 карточек в три ряда (первый ряд - крупная карточка, второй ряд - три карточки в меньшем размере, третий ряд - четыре карточки в маленьком размере).

Вариант:

Платформа для фитнес-тренировок и здоровья

- Вход
- Регистрация
- Личный кабинет пользователя (трекинг прогресса, планы тренировок)
- Поиск тренировок с фильтрацией по уровню, типу (кардио, силовые) и продолжительности
- Страница тренировки с видео, описанием и инструкциями
- Блог о здоровье и питании

Каждый из сайтов обязательно должен включать в себя следующие страницы:

- Вход
- Регистрация
- Личный кабинет пользователя
- Страница для поиска с возможностью фильтрации

Ход работы

1. Создание мокового API

Для имитации серверной части был использован JSON-server. В файле db.json описана структура данных приложения.

Основные коллекции:

- users — пользователи системы, содержащие email, имя, хешированный пароль и объект stats;
- workouts — список тренировок с описанием, уровнем сложности, видео и инструкциями;
- history — история выполненных тренировок пользователя;
- planned — запланированные тренировки;
- posts — публикации блога.

Каждый пользователь содержит объект stats, включающий:

- progress — общий прогресс;
- workouts — количество выполненных тренировок;
- hours — общее время тренировок.

2. Реализация авторизации через API

Авторизация реализована с использованием эндпоинтов /register и /login, предоставляемых JSON-server с модулем авторизации.

Файл main.js отвечает за:

- регистрацию пользователя;
- вход в систему;
- выход из системы;
- управление отображением элементов интерфейса.

Добавленные функции и логика:

- отправка POST-запроса /login с email и паролем;
- сохранение accessToken и данных пользователя в localStorage;
- функция checkAuth(), которая скрывает или показывает кнопки входа, регистрации и выхода;

- очистка данных пользователя при выходе.

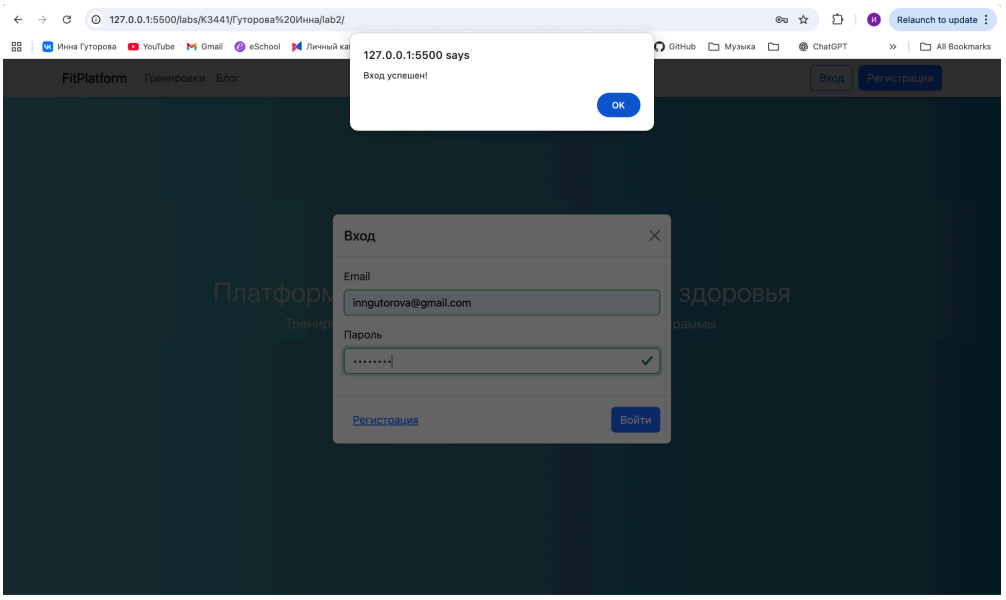

Рисунок 1 - Вход в систему

3. Управление доступом к интерфейсу

Для ограничения доступа к функционалу авторизованных пользователей был создан файл authVisubility.json.

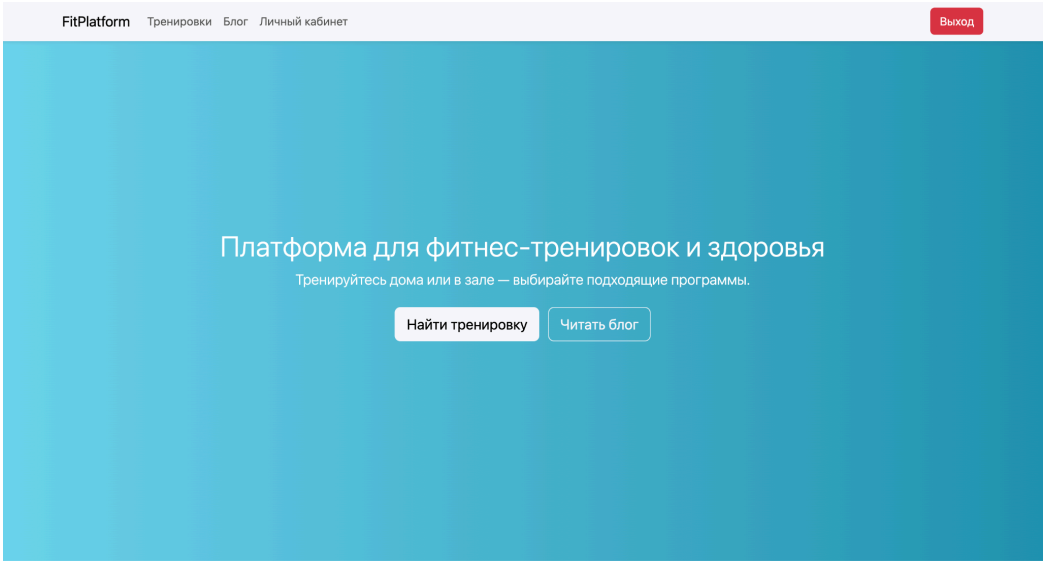

Рисунок 2 - Главная страница у авторизованного пользователя

В нём реализована логика:

- проверка наличия пользователя в localStorage;
- скрытие элементов с классом .auth-only для неавторизованных пользователей;

- отображение этих элементов после входа в систему.

Это позволяет динамически управлять интерфейсом без перезагрузки страницы. Например, вкладка личного кабинета доступна только для авторизованных пользователей.

4. Получение данных из API (fetch)

Для загрузки данных используется стандартный API fetch с асинхронной обработкой через async/await.

Примеры реализованных запросов:

- GET /users — загрузка пользователей (блог);
- GET /users/{id} — загрузка данных текущего пользователя;
- GET /workouts — получение списка тренировок с фильтрацией;
- GET /history?userId= — история тренировок пользователя;
- GET /planned?userId= — запланированные тренировки;
- GET /posts — публикации блога.

5. Личный кабинет пользователя

В файле profile.js реализована загрузка данных личного кабинета через API.

Добавленный функционал:

- запрос данных пользователя GET /users/{id};
- парсинг объекта stats;
- динамический вывод трёх показателей:
    - прогресс,
    - количество тренировок,
    - общее время занятий;
- загрузка истории и планируемых тренировок;
- удаление записей через DELETE /history/{id} и DELETE /planned/{id}.

Все данные личного кабинета формируются исключительно на основе ответа API.

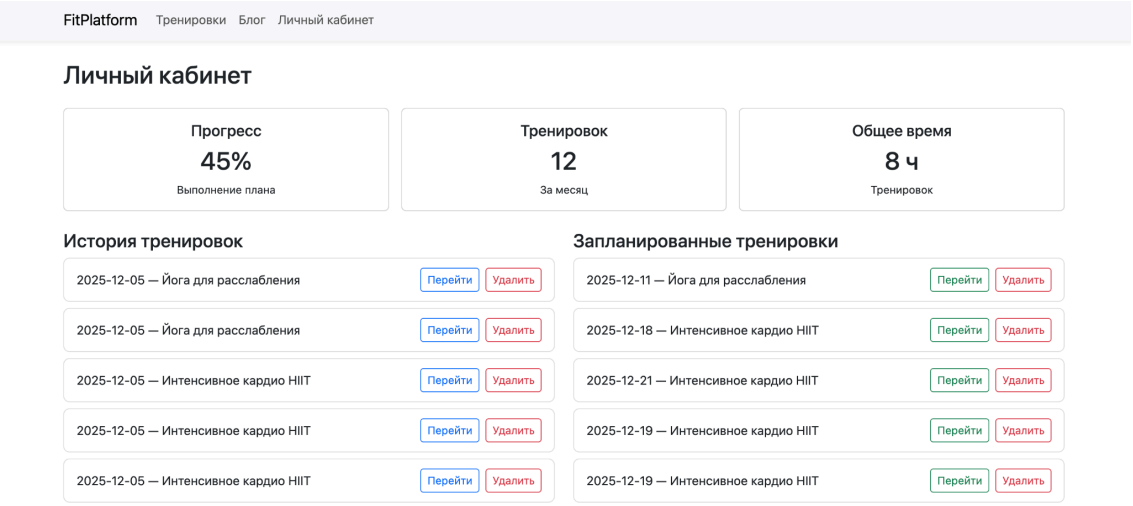

Рисунок 3 - Личный кабинет пользователя

6. Работа с тренировками

Файлы workouts.js и workout.js обеспечивают взаимодействие с API тренировок.

Реализовано:

- загрузка тренировок с сервера;
- фильтрация по уровню, типу и длительности через query-параметры;
- загрузка конкретной тренировки по id;
- добавление выполненной тренировки в историю (POST /history);
- планирование тренировки (POST /planned).

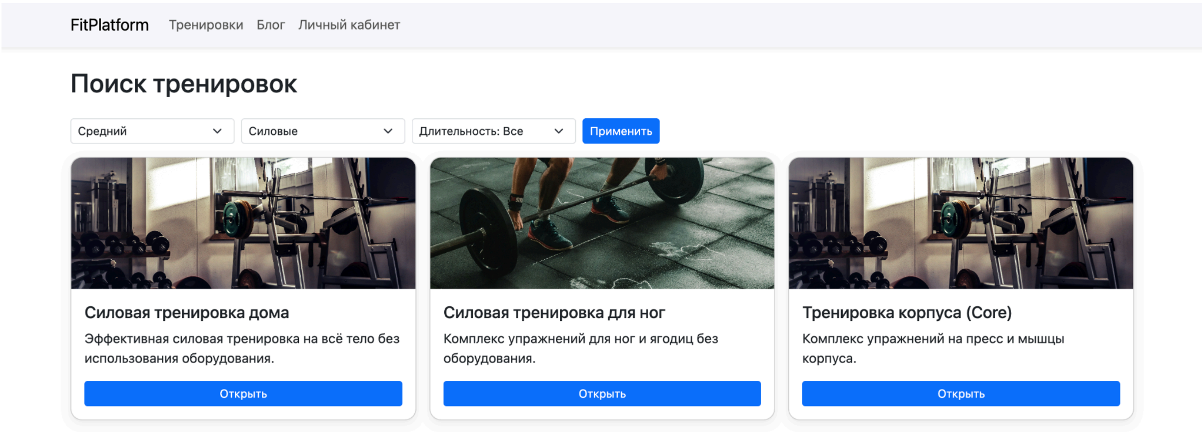

Рисунок 4 - Фильтры тренировок

7. Блог и публикации

В файле blog.js реализована загрузка постов и пользователей.

Добавленные функции:

- fetchUsers() — получение списка пользователей;
- fetchPosts() — загрузка постов;
- сопоставление поста с автором;
- отправка нового поста через POST /posts (доступно только авторизованным пользователям);
- динамическое отображение длинных текстов с кнопкой «Читать дальше».

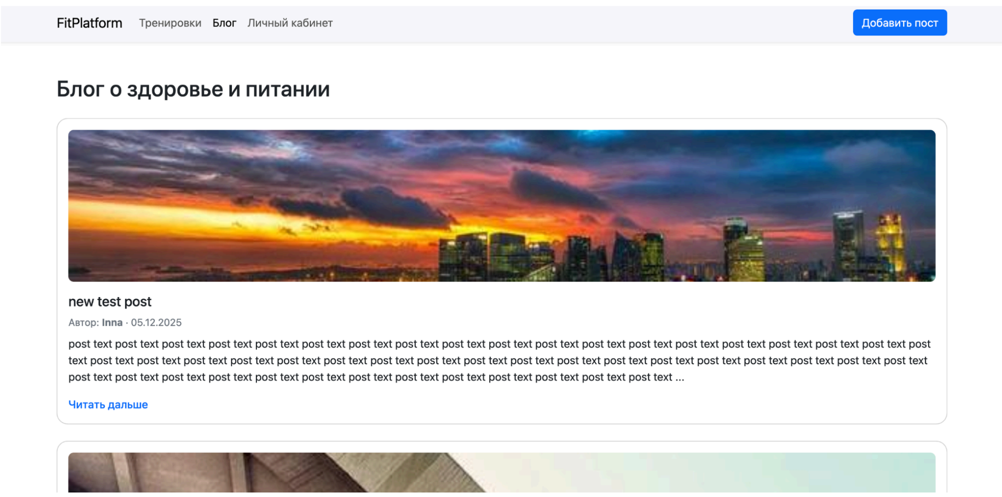

Рисунок 5 - Новый пост в блоге

Вывод

В ходе выполнения лабораторной работы были получены практические навыки работы с API, асинхронными запросами и моковым сервером. Использование JSON-server позволило реализовать серверную логику без написания собственного backend, а fetch обеспечила удобное взаимодействие клиента с API.