САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Ананьев Никита

Группа К3440

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Привязать то, что было сделано в рамках ЛР1 к внешнему API средствами fetch/axios/xhr. Реализовать моковое API средствами JSON-сервера и подключить к нему авторизацию, как в примерах, рассматриваемых в рамках тем "Иммитация работы с API".

Ход работы

В качестве мокового API был использован npm пакет json-server. На рисунке 1 показана команда выполнения запуска json сервера. Для данной лабораторной в db.json (хранилище, используемое сервером) добавлены две коллекции - users и properties. Пример данных, которые сохраняются в эти коллекции см. на рисунке 2.

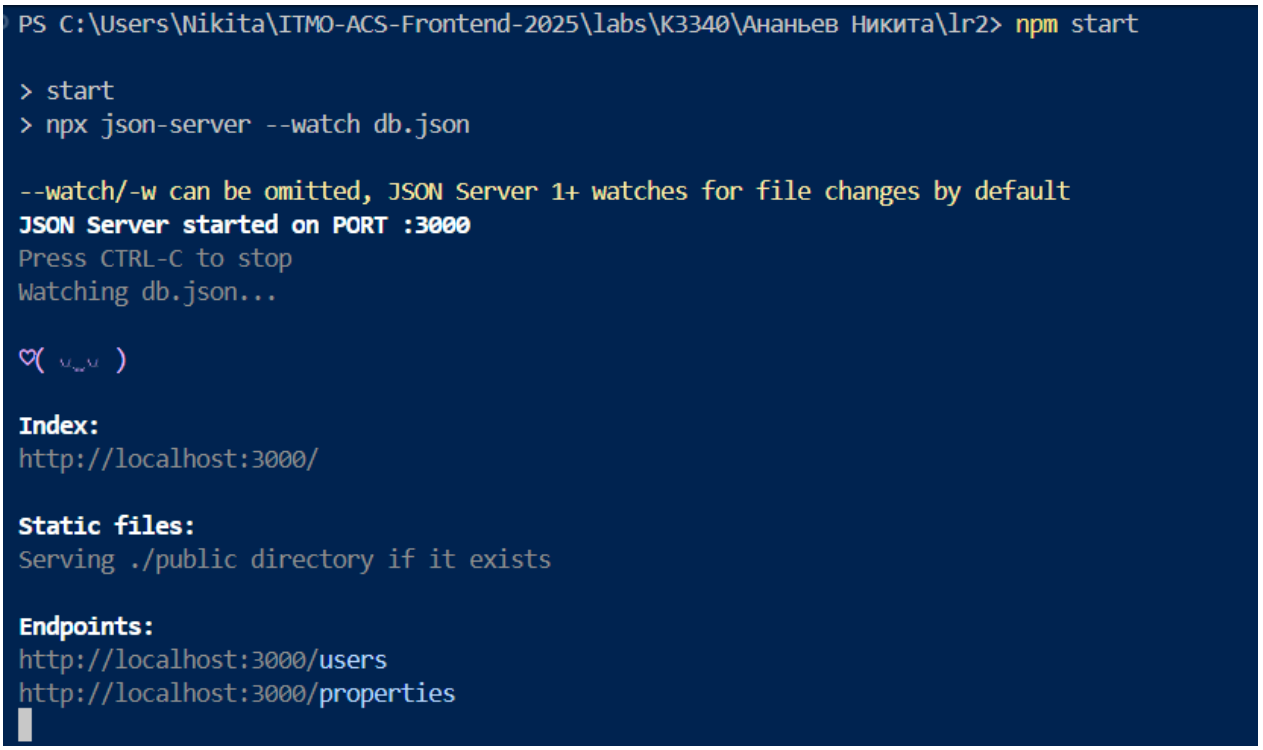

Рисунок 1 – Запуск json сервера

Рисунок 2 – Пример данных в db.json

Авторизация

Теперь при регистрации сначала происходит проверка на существование пользователя в системе (идентификация) по email. Если пользователь проходит идентификацию, регистрация невозможна и предлагается войти в систему под имеющимся аккаунтом (см. рисунок 3 и 4).

```
async function identifyUserByEmail(email) {
    let exists = false;

    await fetch(`${USERS_URL}?email=${email}`)
        .then((response) => {
            return response.json();
        })
        .then((data) => {
            exists = (data.length > 0) ? true : false;
        })

    return exists;
}
```

Рисунок 3 – Функция идентификации пользователя

```
const exists = await identifyUserByEmail(email);
if (exists) {
    alert('Пользователь с таким email уже зарегистрирован');
    return;
}

try {
    const response = await fetch(
        USERS_URL, {
            method: "POST",
            headers: {
                "Content-Type": "application/json; charset=utf-8",
            },
            body: JSON.stringify({name, email, phone, password, role: DEFAULT_ROLE})
        }
    );

    if (!response.ok)
        throw new Error(`HTTP error, status: ${response.status}`);

    window.location.href = 'login.html';

} catch (e) {
    console.error(e);
    alert('Ошибка регистрации');
}
```

Рисунок 4 – Регистрация (создание пользователя)

В процессе авторизации происходит запрос к json серверу на получение данных пользователя, далее идет сравнение пароля из базы с введенным в форму. В случае успешного входа данные о пользователе сохраняются в localStorage и затем уже забираются оттуда по мере необходимости.

Главная страница

Список объектов недвижимости загружается запросом к json-серверу с набором query-параметров, которые формируются на основе применяемых пользователем фильтров. Фильтры сохраняются в карту filters и при формировании запроса применяется функция buildQuery (см. рисунок 5).

```
function buildQuery(fl) {
    if (fl.size === 0) return "";

    const params = new URLSearchParams();

    fl.forEach((value, key) => {
        if (key === DEFAULT_SELECT)
            return;

        if (key === "price") {
            const [min, max] = value.split("-");
            params.append("price_gte", min);
            params.append("price_lte", max);
        }
        else if (Array.isArray(value)) {
            value.forEach(v => params.append(key, v));
        } else {
            params.append(key, value);
        }
    });

    return "?" + params.toString();
}
```

Рисунок 5 – Функция построения query параметров запроса

Когда сервер возвращает отфильтрованные записи, блоки карточек программно добавляются на html страницу (см. рисунок 6, 7 и 8).

```
async function filterProperties() {
    collectFilters();
    const query = buildQuery(filters);

    response = await fetch(PROPERIES_URL + query);

    if (!response.ok) {
        console.error(`Failed to load properties list; response status = ${response.status}`);
        return;
    }

    data = await response.json();
    const cards = document.getElementById("cards");
    cards.innerHTML = "";

    if (data.length == 0) {
        return;
    }

    data.forEach((prop) => {
        cards.append(buildCardHTML(prop));
    });

    window.scrollTo(0, 0);
}
```

Рисунок 6 – Получение объектов недвижимости

```
function buildCardHTML(property) {
    const newCard = document.createElement("div");
    newCard.className = "card";

    const img = document.createElement("img");
    img.src = (property.feed_img !== undefined) ? property.feed_img : DEFAULT_PICTURE_URL;

    const info = document.createElement("div");
    info.className = "card-info";

    const price = document.createElement("div");
    price.className = "price";
    price.innerHTML = `${property.price} ₽ / `;
    let frequency = "месяц";

    if (property.is_daily_payment)
        frequency = "день";

    price.innerHTML += frequency;

    const link = document.createElement("div");
    link.innerHTML = `<a class="card-link" href="property.html?id=${property.id}">${property.address}</a>`;

    const description = document.createElement("p");
    description.innerHTML = `${property.area} м², ${property.rooms} комнат(-ы)`;
    if (property.floor !== undefined)
        description.innerHTML += `, ${property.floor}-й этаж`;
    description.innerHTML += `<br>${property.description}`;

    info.append(price, link, description);
    newCard.append(img, info);

    return newCard;
}
```

Рисунок 7 – Формирование html для карточки

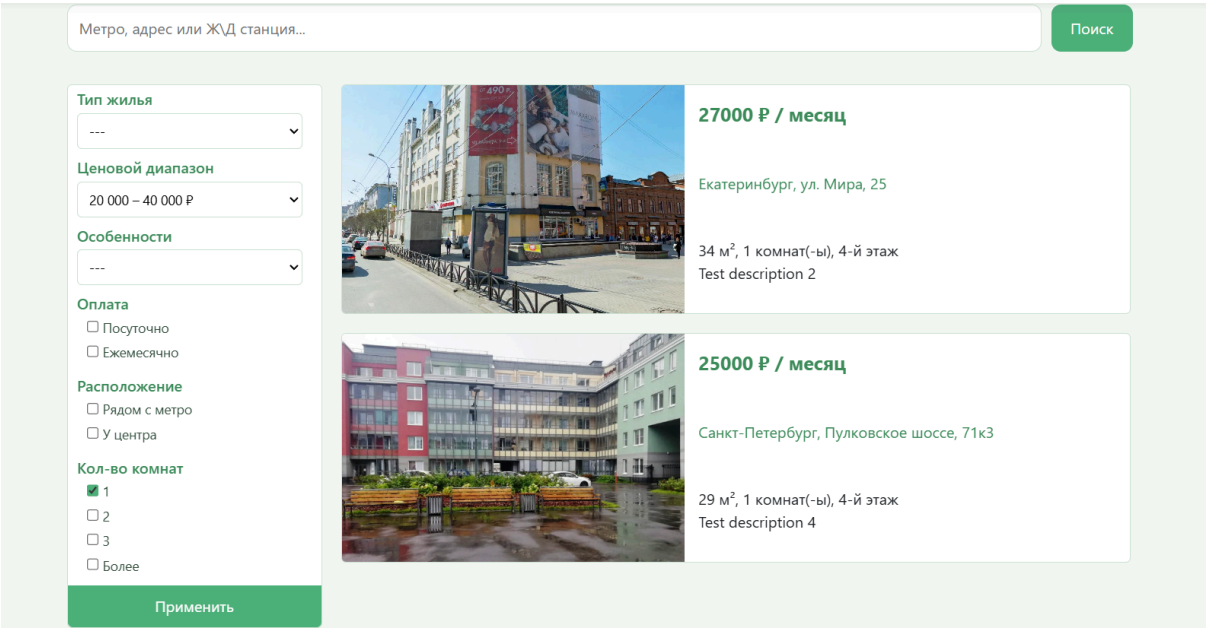

Рисунок 8 – Отфильтрованные объекты недвижимости

Карточка недвижимости

Была добавлена страница для арендуемой недвижимости (см. рисунок 9).

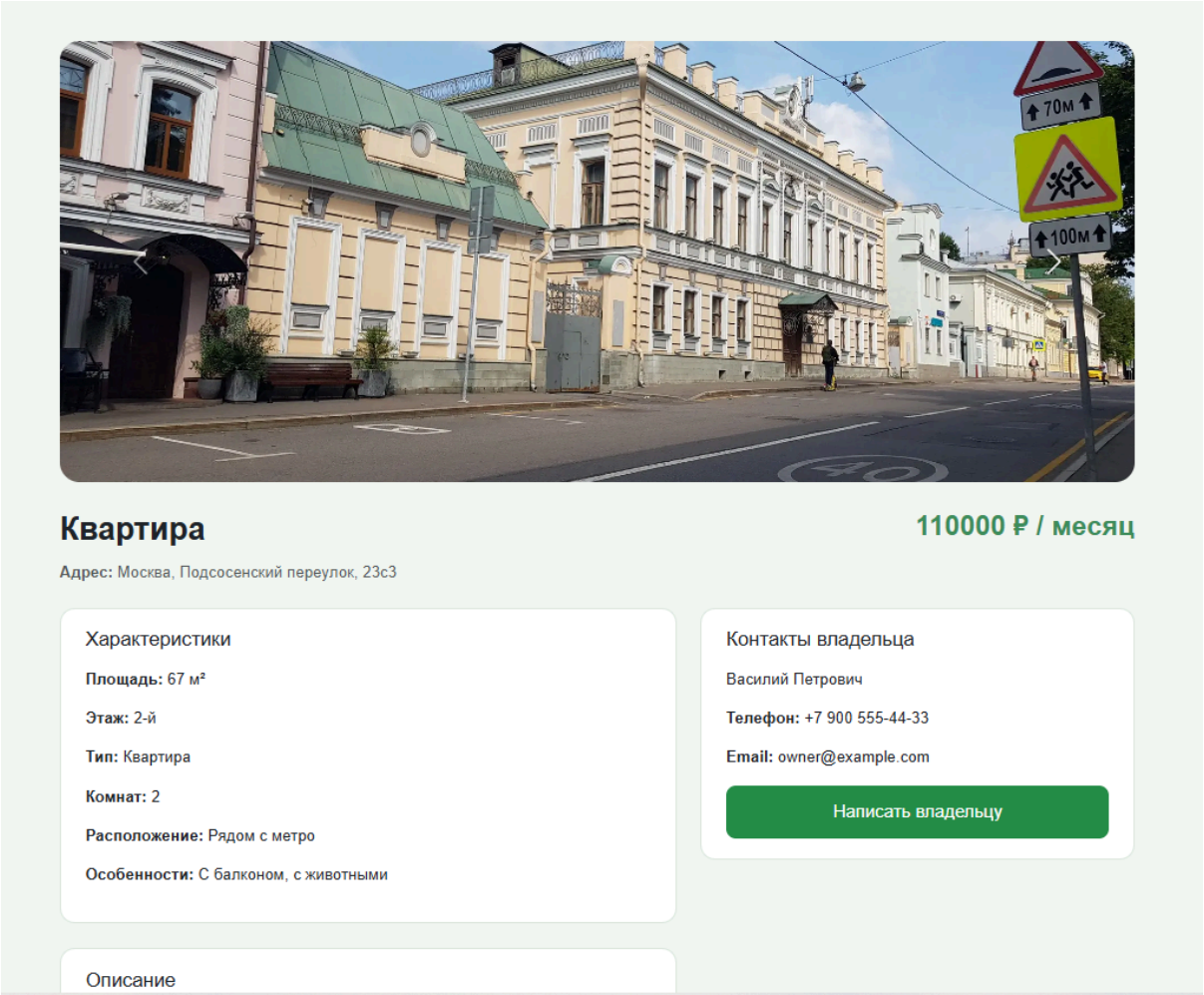

Рисунок 9 – Страница недвижимости

Данные по недвижимости подтягиваются отдельным запросом, после чего специальная функция вставляет полученные данные в элементы html страницы (см. рисунок 10).

```
function fillPropertyContent(property) {
    const carousel = document.getElementById("carouselInner");
    const images = (property.images !== undefined && property.images.length !== 0) ? property.images : [DEFAULT_PICTURE_URL];

    images.forEach((img) => {
        imgDiv = document.createElement("div");
        imgDiv.className = "carousel-item active";
        imgDiv.innerHTML = `<img src="${img}" class="d-block w-100">`;
        carousel.append(imgDiv);
    });

    const title = document.getElementById("propertyTitle");
    title.textContent = property.title;

    const price = document.getElementById("propertyPrice");
    price.textContent = `${property.price} ₽ / `;
    let frequency = "месяц";

    if (property.is_daily_payment)
        frequency = "день";

    price.textContent += frequency;

    const address = document.getElementById("propertyAddress");
    address.innerHTML = `<b>Адрес:</b> ${property.address}`;

    const specs = document.getElementById("propertySpecs");
    let floor = (property.floor !== undefined) ? property.floor + "-й" : "-";

    specs.innerHTML = `
        <h5 class="mb-3">Характеристики</h5>
        <p><b>Площадь:</b> ${property.area} м²</p>
        <p><b>Этаж:</b> ${floor}</p>
        <p><b>Тип:</b> ${typeMapper.get(property.type)}</p>
        <p><b>Комнат:</b> ${property.rooms}</p>
        <p><b>Расположение:</b> Рядом с метро</p>
        <p><b>Особенности:</b> С балконом, с животными</p>
    `;

    const description = document.getElementById("propertyDescription");
    description.innerHTML = `
        <h5 class="mb-3">Описание</h5>
        ${property.description}
    `;
}
```

Рисунок 10 – Заполнение страницы недвижимости

Выводы

Выполнение данной лабораторной работы стало отличным упражнением во взаимодействии с внешним API со стороны фронтенда. Конечно, сейчас мало кто разрабатывает сайты, используя многостраничный подход с чистым js, однако на мой взгляд, опыт использования классического js + html улучшает понимание современных фреймворков и библиотек, асинхронного взаимодействия и того, как в целом формируется динамический контент на странице сайта.