

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Фронт-энд разработка

Отчет

**Лабораторная работа №2
Взаимодействие с внешним API**

Выполнил:
Чернышев Михаил Павлович
Группа К3441

Проверил:
Добряков Д. И.

Санкт-Петербург
2026 г.

Содержание

1 Задача	2
2 Ход работы	2
2.1 Настройка JSON Server	2
2.2 Структура базы данных	2
2.3 Базовый сервис для API-запросов	3
2.4 Сервис авторизации	3
2.5 Сервис недвижимости	5
2.6 Сервис избранного	5
2.7 Сервис бронирований	7
2.8 Интеграция с UI: форма входа	7
2.9 Интеграция с UI: загрузка объектов	9
2.10 Альтернатива: Axios	9
3 Вывод	11

1 Задача

Привязать разработанный в ЛР1 сайт сервиса аренды недвижимости к внешнему API средствами fetch/axios. Реализовать моковое API средствами JSON Server с авторизацией.

Требуется реализовать: систему авторизации, получение списка объектов с фильтрацией, работу с избранным, создание бронирований, систему сообщений.

2 Ход работы

2.1 Настройка JSON Server

JSON Server позволяет быстро создать REST API на основе JSON-файла.

```
npm install json-server --save-dev

# package.json
"scripts": {
  "server": "json-server --watch db.json --port 3001"
}
```

2.2 Структура базы данных

db.json (фрагмент)

```
{
  "users": [
    {
      "id": 1, "email": "ivan@example.com",
      "password": "hashed_password", "firstName": "Иван",
      "verified": true
    }
  ],
  "tokens": [
    {
      "id": 1, "userId": 1,
      "token": "eyJhbGciOiJIUzI1NiIs...",
      "expiresAt": "2026-01-17T10:00:00Z"
    }
  ],
  "properties": [
    {
      "id": 1, "title": "2-комнатная квартира",
      "price": 55000, "type": "apartment", "rooms": 2,
      "address": { "city": "Москва", "metro": "Арбатская" },
      "amenities": ["wifi", "furniture"], "status": "available"
    }
  ],
  "bookings": [], "favorites": [], "messages": []
}
```

2.3 Базовый сервис для API-запросов

```
api/config.js
const API_BASE_URL = 'http://localhost:3001';

function getHeaders() {
    const headers = { 'Content-Type': 'application/json' };
    const token = localStorage.getItem('authToken');
    if (token) headers['Authorization'] = `Bearer ${token}`;
    return headers;
}

async function apiRequest(endpoint, options = {}) {
    const response = await fetch(`${API_BASE_URL}${endpoint}`, {
        ...options,
        headers: { ...getHeaders(), ...options.headers }
    });

    if (!response.ok) {
        const error = await response.json().catch(() => ({}));
        throw new Error(error.message || 'HTTP Error: ${response.status}');
    }
    return response.json();
}

export { apiRequest };
```

2.4 Сервис авторизации

```
api/authService.js (ключевые методы)
const AuthService = {
    async login(email, password) {
        const users = await apiRequest('/users?email=${email}');
        if (users.length === 0) throw new Error('Пользователь не найден');

        const user = users[0];
        if (user.password !== this.hashPassword(password)) {
            throw new Error('Неверный пароль');
        }

        const token = await this.createToken(user.id);
        localStorage.setItem('authToken', token.token);
        localStorage.setItem('userId', user.id);
        return { user, token };
    },

    async logout() {
        const token = localStorage.getItem('authToken');
        if (token) {
            const tokens = await apiRequest('/tokens?token=${token}');
            if (tokens.length > 0) {
                await apiRequest('/tokens/${tokens[0].id}', { method: 'DELETE' });
            }
        }
    }
};
```

```
        }
        localStorage.removeItem('authToken');
        localStorage.removeItem('userId');
    },
    async checkAuth() {
        const token = localStorage.getItem('authToken');
        if (!token) return null;

        const tokens = await apiRequest('/tokens?token=${token}');
        if (tokens.length === 0 || new Date(tokens[0].expiresAt) < new Date()) {
            this.logout();
            return null;
        }
        return apiRequest('/users/${localStorage.getItem('userId')}`);
    }
};
```

2.5 Сервис недвижимости

```
api/propertyService.js
const PropertyService = {
    async getProperties(filters = {}) {
        let params = ['status=available'];

        if (filters.type) params.push(`type=${filters.type}`);
        if (filters.rooms) params.push(`rooms=${filters.rooms}`);
        if (filters.sortBy) {
            params.push(`_sort=${filters.sortBy}`);
            params.push(`_order=${filters.order || 'asc'}`);
        }

        const properties = await apiRequest('/properties?${params.join("&")}`);

        // Фильтрация по цене на клиенте
        return properties.filter(p => {
            if (filters.priceFrom && p.price < filters.priceFrom) return false;
            if (filters.priceTo && p.price > filters.priceTo) return false;
            return true;
        });
    },
    async getPropertyById(id) {
        return apiRequest('/properties/${id}');
    }
};
```

2.6 Сервис избранного

```
api/favoriteService.js
const FavoriteService = {
    async toggleFavorite(propertyId) {
        const userId = localStorage.getItem('userId');
        if (!userId) throw new Error('Необходима авторизация');

        const existing = await apiRequest(
            '/favorites?userId=${userId}&propertyId=${propertyId}'
        );

        if (existing.length > 0) {
            await apiRequest('/favorites/${existing[0].id}', { method: 'DELETE' });
            return false;
        } else {
            await apiRequest('/favorites', {
                method: 'POST',
                body: JSON.stringify({
                    userId: parseInt(userId), propertyId,
                    createdAt: new Date().toISOString()
                })
            });
        }
        return true;
    }
};
```

```
        }  
    };  
};
```

2.7 Сервис бронирований

```
api/bookingService.js
const BookingService = {
    async createBooking(bookingData) {
        const userId = localStorage.getItem('userId');
        if (!userId) throw new Error('Необходима авторизация');

        // Проверка доступности дат
        const conflicts = await this.checkAvailability(
            bookingData.propertyId, bookingData.checkIn, bookingData.checkOut
        );
        if (conflicts.length > 0) throw new Error('Даты заняты');

        return apiRequest('/bookings', {
            method: 'POST',
            body: JSON.stringify({
                ...bookingData,
                userId: parseInt(userId),
                status: 'pending',
                createdAt: new Date().toISOString()
            })
        });
    },
    async checkAvailability(propertyId, checkIn, checkOut) {
        const bookings = await apiRequest(
            '/bookings?propertyId=${propertyId}&status_ne=cancelled'
        );
        return bookings.filter(b => {
            const s = new Date(checkIn), e = new Date(checkOut);
            const bs = new Date(b.checkIn), be = new Date(b.checkOut);
            return !(e <= bs || s >= be);
        });
    }
};
```

2.8 Интеграция с UI: форма входа

```
login.js
document.getElementById('loginForm').addEventListener('submit', async (e) => {
    e.preventDefault();

    const email = document.getElementById('email').value;
    const password = document.getElementById('password').value;
    const submitBtn = e.target.querySelector('button[type="submit"]');

    submitBtn.disabled = true;
    submitBtn.innerHTML = '<span class="spinner-border spinner-border-sm"></span>';

    try {
        const { user } = await AuthService.login(email, password);
        showToast('Добро пожаловать, ${user.firstName}!', 'success');
```

```
    setTimeout(() => window.location.href = 'profile.html', 1000);
} catch (error) {
    document.getElementById('errorAlert').textContent = error.message;
    document.getElementById('errorAlert').classList.remove('d-none');
    submitBtn.disabled = false;
    submitBtn.innerHTML = 'Войти';
}
});
```

2.9 Интеграция с UI: загрузка объектов

```
index.js
async function loadProperties() {
    const propertyList = document.getElementById('propertyList');

    try {
        const properties = await PropertyService.getProperties(state.filters);
        document.getElementById('count').textContent = properties.length;

        propertyList.innerHTML = properties.map(p => `
            <div class="col-md-4 mb-4">
                <div class="card property-card" data-id="${p.id}">
                    
                    <div class="card-body">
                        <h5>${p.title}</h5>
                        <p class="text-muted">${p.address.city}</p>
                        <strong class="text-primary">
                            ${p.price.toLocaleString()} руб/мес
                        </strong>
                    </div>
                </div>
            </div>
        `).join('');
    } catch (error) {
        showError('Не удалось загрузить объекты');
    }
}

// Фильтры
document.getElementById('applyFilters').addEventListener('click', () => {
    state.filters = {
        priceFrom: document.getElementById('priceFrom').value,
        priceTo: document.getElementById('priceTo').value,
        type: document.getElementById('propertyType').value
    };
    loadProperties();
});

document.addEventListener('DOMContentLoaded', loadProperties);
```

2.10 Альтернатива: Axios

```
axiosConfig.js
import axios from 'axios';

const api = axios.create({
    baseURL: 'http://localhost:3001',
    timeout: 10000
});

api.interceptors.request.use(config => {
    const token = localStorage.getItem('authToken');
```

```
if (token) config.headers.Authorization = 'Bearer ${token}';
return config;
});

api.interceptors.response.use(
  response => response.data,
  error => {
    if (error.response?.status === 401) {
      localStorage.clear();
      window.location.href = '/login.html';
    }
    throw new Error(error.response?.data?.message || 'Ошибка сервера');
}
);

export default api;
```

3 Вывод

В ходе выполнения лабораторной работы было реализовано взаимодействие веб-приложения с внешним API.

Реализованные компоненты:

- **JSON Server** — моковый REST API с базой данных в JSON
- **Сервис авторизации** — регистрация, вход, выход, проверка токенов
- **Сервис недвижимости** — получение списка с фильтрацией и сортировкой
- **Сервис избранного** — добавление/удаление объектов
- **Сервис бронирований** — создание брони с проверкой доступности дат

Использованные технологии:

- JSON Server для прототипирования API
- Fetch API и Axios для HTTP-запросов
- LocalStorage для хранения токенов
- Async/Await для асинхронного кода

Приложение успешно взаимодействует с моковым API, что позволяет разрабатывать клиентскую часть независимо от бэкенда.