**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Фронт-энд разработка

Отчет

# Лабораторная работа 2: взаимодействие с внешним API

Выполнил:
Петухов Семён
Алексеевич

K3439

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 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).

**Ход работы**

Для реализации 1 лабораторной работы мной был взят вариант с сайтом для аренды недвижимости.

В моей систем системы необходимо было реализовать 4 сущности – users, advertisement, conversations, transactions. Для реализации mock API мной был использован JSON-server и Axios

I. Поднятие сервера

Перед созданием моделей данных нужно установить json-server

*npm install json-server* – мной он был установлен локально в репозиторий

Для запуска сервера в консоль нужно ввести данную команду

*npx json-server --watch db.json --port 3001*

Сервер будет запущен на порте 3001

II. Заполнение данными

Заполнение данными происходит крайне просто, даже слегка примитивно. В файл db.json мы записываем наши модели с данными

*users*


```
    {
      "id": "101",
      "name": "admin",
      "email": "admin@mail.ru",
      "password": "admin1",
      "location": "Саратов",
      "username": "owner101"
    }
```


*advertisement*


```
    {
      "id": "1",
      "title": "Современный дом на окраине Саратова",
      "type": "Аппартаменты",
      "price": 1200,
      "location": "Саратовская область",
      "beds": 2,
      "baths": 1,
      "ownerId": "101",
      "status": "available",
      "images": [
        "https://images.unsplash.com/photo-1560448204-
e02f11c3d0e2?w=1200&q=60&auto=format&fit=crop",
        "https://images.unsplash.com/photo-1600585154340-
be6161a56a0c?w=1200&q=60&auto=format&fit=crop"
      ],
      "description": "Светлые современные апартаменты с быстрым доступом к
магазинам, остановкам и кафе. Полностью меблированы: гостиная, кухня и две
спальни.",
      "rentTerms": "Минимальный срок аренды — 6 месяцев. Залог: стоимость
одного месяца."
    }
```


*conversations*


```
{
      "id": "1",
      "userId": "102",
      "with": "Владелец А",
      "propertyId": "1",
      "lastMessage": "Жду заселения.",
      "date": "2025-10-18"
    }
```


*transactions*


```
    {
      "id": "1",
      "userId": "102",
```

```
      "propertyId": null,
      "amount": 860,
      "date": "2025-09-01",
      "status": "Завершено"
    }
```

Для каждой модели указывается название и её элементы, проверки структуры не происходит, поэтому данные в одной структуре могут отличаться

## III. Получение данных

Первым делом была написана обёртка для Axios запросов

```
const API_BASE_URL = 'http://localhost:3001';
const api = axios.create({ baseURL: API_BASE_URL, timeout: 5000 });
```

Т.к. id хранятся в string, а в коде используется int – нужно нормализовать значения для исключения ошибок

```
function normalizeId(value){
  if(value === undefined || value === null) return null;
  if(typeof value === 'string'){
    const trimmed = value.trim();
    return trimmed ? trimmed : null;
  }
  if(typeof value === 'number' && !Number.isNaN(value)){
    return String(value);
  }
  return null;
}

function normalizeAdv(prop){
  if(!prop || typeof prop !== 'object') return null;
  const id = normalizeId(prop.id);
  const ownerId = normalizeId(prop.ownerId);
  const tenantId = normalizeId(prop.tenantId);
  return {
    ...prop,
    id: id || prop.id,
    ownerId: ownerId || prop.ownerId,
    tenantId: tenantId || prop.tenantId
  };
}

function normalizeConversation(conv){
  if(!conv || typeof conv !== 'object') return null;
```

```
  return {
    ...conv,
    id: normalizeId(conv.id) || conv.id,
    userId: normalizeId(conv.userId) || conv.userId,
    propertyId: normalizeId(conv.propertyId) || conv.propertyId
  };
}


function normalizeConversation(conv){
  if(!conv || typeof conv !== 'object') return null;
  return {
    ...conv,
    id: normalizeId(conv.id) || conv.id,
    userId: normalizeId(conv.userId) || conv.userId,
    propertyId: normalizeId(conv.propertyId) || conv.propertyId
  };
}

function normalizeTransaction(tx){
  if(!tx || typeof tx !== 'object') return null;
  return {
    ...tx,
    id: normalizeId(tx.id) || tx.id,
    userId: normalizeId(tx.userId) || tx.userId,
    propertyId: normalizeId(tx.propertyId) || tx.propertyId
  };
}
```

*Advertisement*

Чтобы оптимизировать работу программы было решено отправлять запрос на сервер только при необходимости, после первого запроса данные кешируются и хранятся одновременно и на сервере, и в кэше

При необходимости можно подтягивать данные с сервера и после, но данная функция в моём приложении пока не используется за ненадобностью

```
let advertisementCache = null;

//Обновление
async function fetchAdvertisement(forceRefresh = false){
  if(advertisementCache && !forceRefresh) return advertisementCache;
  try{
    const { data } = await api.get('/advertisement');
    const list = Array.isArray(data) ? data : [];
```

```
    advertisementCache = list.map(p => normalizeAdv(p) || p);
    return advertisementCache;
  } catch (error) {
    handleApiError('Не удалось загрузить объявления. Убедитесь, что JSON-
server запущен.', error);
    return [];
  }
}

//Удаление из записи
async function deleteAdvertisement(id){
  const idStr = normalizeId(id);
  if(!idStr) return false;
  try{
    await api.delete(`/advertisement/${idStr}`);
    removeAdvertisementFromCache(idStr);
    return true;
  } catch (error) {
    handleApiError('Не удалось удалить объявление. Проверьте JSON-server.',
error);
    return false;
  }
}
//Удаление из cache
function removeAdvertisementFromCache(id){
  const idStr = normalizeId(id);
  if(!idStr || !Array.isArray(advertisementCache)) return;
  advertisementCache = advertisementCache.filter(prop => prop.id !== idStr);
}
```

Для заполнения экрана объявлений реализована функция

```
function renderAdvertisements(container, advertisements){
  if(!container) return;
  container.innerHTML = '';
  if(advertisements.length === 0){
    container.innerHTML = `<div class="text-muted p-4 bg-white card-
custom">Объявления не найдены.</div>`;
    return;
  }
  advertisements.forEach(prop=>{
    const cover = prop.images && prop.images.length ? prop.images[0] :
'https://via.placeholder.com/600x400?text=No+image';
    const propId = normalizeId(prop.id) || prop.id;
    const col = document.createElement('div');
    col.className = 'col-md-6 col-lg-4 mb-4';
```

```
    col.innerHTML = `
      <div class="card card-custom h-100">
        <img src="${cover}" class="property-img w-100" alt="${prop.title}">
        <div class="property-card-body">
          <div class="d-flex justify-content-between align-items-start">
            <div>
              <h6 class="mb-1">${prop.title}</h6>
              <div class="property-meta">${prop.location} • ${prop.type} •
${prop.beds} спал. • ${prop.baths} ван.</div>
            </div>
            <div class="text-end">
              <div class="h6 mb-1">₽${prop.price}/мес</div>
              <div class="badge-ghost">${statusText(prop.status)}</div>
            </div>
          </div>
          <p class="mt-2 mb-2 text-muted
small">${(prop.description||'').slice(0,90)}...</p>
          <div class="d-flex justify-content-between">
            <button class="btn btn-sm btn-outline-primary view-btn" data-
id="${propId}">Посмотреть</button>
            <button class="btn btn-sm btn-primary book-btn" data-
id="${propId}" ${prop.status !== 'available' ? 'disabled' :
''}>Забронировать</button>
          </div>
        </div>
      </div>
    `;
    container.appendChild(col);
  });
```

*User*

Для пользователя были реализованы необходимы CRUD`ы аналогично Advertisement

```
//Получение пользователя по email и паролю (для входа)
async function queryUserByCredentials(email, password){
  const params = { email, password };
  try{
    const { data } = await api.get('/users', { params });
    const user = Array.isArray(data) ? data[0] || null : null;
    return user ? withStringId(user) || user : null;
  } catch (error) {
    handleApiError('Ошибка входа. Проверьте подключение к JSON-server.',
error);
    return null;
  }
```

```
}

//Поиск пользователя по Email
async function findUserByEmail(email){
  try{
    const { data } = await api.get('/users', { params: { email } });
    const user = Array.isArray(data) ? data[0] || null : null;
    return user ? withStringId(user) || user : null;
  } catch (error) {
    handleApiError('Не удалось проверить наличие пользователя.', error);
    return null;
  }
}
//Создание пользователя
async function createUser(payload){
  try{
    const { data } = await api.post('/users', payload);
    return withStringId(data) || data;
  } catch (error) {
    handleApiError('Не удалось создать пользователя. Проверьте JSON-server.',
error);
    return null;
  }
}

async function updateUser(id, payload){
  const idStr = normalizeId(id);
  if(!idStr) return null;
  try{
    const { data } = await api.patch(`/users/${idStr}`, payload);
    return withStringId(data) || data;
  } catch (error) {
    handleApiError('Не удалось обновить профиль. Проверьте JSON-server.',
error);
    return null;
  }
}
//Получение пользователя по id
async function getUserById(id){
  const idStr = normalizeId(id);
  if(!idStr) return null;
  try{
    const { data } = await api.get(`/users/${idStr}`);
    return withStringId(data) || data;
  } catch (error) {
    if(error?.response?.status === 404){
      try {
```

```
        const { data: fallback } = await api.get('/users', { params: { id:
idStr } });
        if(Array.isArray(fallback) && fallback.length){
          return withStringId(fallback[0]) || fallback[0];
        }
        alert(`Пользователь с ID ${idStr} не найден на сервере. Пожалуйста,
войдите заново.`);
      } catch(fallbackErr){
        handleApiError('Fallback-запрос пользователя не удался. Проверьте
JSON-server.', fallbackErr);
      }
    } else {
      handleApiError('Не удалось получить пользователя. Проверьте JSON-
server.', error);
    }
    return null;
  }
}
```

Также для корректной работы страницы профиля была реализована логика хранения текущего пользователя

```
function getCurrentUser(){
  const raw = localStorage.getItem('rental_currentUser');
  if(!raw) return null;
  try{
    const parsed = JSON.parse(raw);
    const normalized = withStringId(parsed);
    if(normalized) return normalized;
  } catch(err){
    console.warn('Не удалось распарсить сохранённого пользователя', err);
  }
  localStorage.removeItem('rental_currentUser');
  return null;
}

function setCurrentUser(user){
  const normalized = withStringId(user);
  if(!normalized) return;
  localStorage.setItem('rental_currentUser', JSON.stringify(normalized));
}
```

И заглушка логики аутентификации

```
function logout(){
  localStorage.removeItem('rental_currentUser');
  window.location = 'index.html';
}
```

```
function handleRegisterForm(){
  const form = document.getElementById('registerForm');
  if(!form) return;
  form.addEventListener('submit', async (e)=>{
    e.preventDefault();
    const name = form.name.value.trim();
    const email = form.email.value.trim();
    const p1 = form.password.value.trim();
    const p2 = form.confirm.value.trim();
    if(!name || !email || !p1 || !p2){ alert('Пожалуйста, заполните все
поля'); return;}
    if(p1 !== p2){ alert('Пароли не совпадают'); return; }
    const existing = await findUserByEmail(email);
    if(existing){ alert('Пользователь с такой эл. почтой уже существует');
return; }
    const username = email.split('@')[0];
    const payload = { name, email, password: p1, username };
    const created = await createUser(payload);
    if(!created){ return; }
    setCurrentUser(created);
    window.location = 'dashboard.html';
  });
}

function handleLoginForm(){
  const form = document.getElementById('loginForm');
  if(!form) return;
  form.addEventListener('submit', async (e)=>{
    e.preventDefault();
    const email = form.email.value.trim();
    const pwd = form.password.value.trim();
    if(!email || !pwd){ alert('Пожалуйста, заполните все поля'); return; }
    const found = await queryUserByCredentials(email, pwd);
    if(found){
      setCurrentUser(found);
      window.location = 'dashboard.html';
    } else {
      alert('Неверные данные или сервер недоступен.');
    }
  });
}
```

Также текущий имя текущего пользователя подтягивается в header

```
function renderNavbar(){
  const user = getCurrentUser();
  const navUser = document.getElementById('nav-user');
```

```
  if(!navUser) return;
  if(user){
    navUser.innerHTML = `
      <span class="me-3">${user.name}</span>
      <a href="profile.html" class="btn btn-sm btn-outline-primary me-
2">Профиль</a>
      <a href="dashboard.html" class="btn btn-sm btn-outline-primary me-
2">Объявления</a>
      <button id="logoutBtn" class="btn btn-sm btn-danger">Выйти</button>
    `;
    document.getElementById('logoutBtn').addEventListener('click', logout);
  } else {
    navUser.innerHTML = `<a href="login.html" class="btn btn-sm btn-
primary">Войти</a>`;
  }
}
```

*Conversations и Transactions*

Аналогично объявлениям было реализовано получение данных Сообщений и транзакций для конкретного пользователя. Так же записи были кэшированы

```
async function fetchConversations(userId, forceRefresh = false){
  const idStr = normalizeId(userId);
  if(!idStr) return [];
  if(!forceRefresh && conversationsCache.has(idStr)){
    return conversationsCache.get(idStr);
  }
  try{
    const { data } = await api.get('/conversations', { params: { userId: idStr
} });
    const list = Array.isArray(data) ? data : [];
    const normalized = list.map(c => normalizeConversation(c) || c);
    conversationsCache.set(idStr, normalized);
    return normalized;
  } catch (error) {
    handleApiError('Не удалось загрузить список сообщений.', error);
    return [];
  }
}

async function fetchTransactions(userId, forceRefresh = false){
  const idStr = normalizeId(userId);
  if(!idStr) return [];
  if(!forceRefresh && transactionsCache.has(idStr)){
    return transactionsCache.get(idStr);
  }
  try{
```

```
    const { data } = await api.get('/transactions', { params: { userId: idStr
} });
    const list = Array.isArray(data) ? data : [];
    const normalized = list.map(t => normalizeTransaction(t) || t);
    transactionsCache.set(idStr, normalized);
    return normalized;
  } catch (error) {
    handleApiError('Не удалось загрузить историю платежей.', error);
    return [];
  }
}
```

**Вывод**

В ходе работы была настроена система получения данных через моковое API. Была реализована аутентификация (без шифрования данных), удаление объявлений, отслеживание сообщения и транзакций конкретного пользователя