**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Фронт-энд разработка

Отчет

# Лабораторная работа 2: Лабораторная работа 2: взаимодействие с внешним API

Выполнил:

Фирсов Илья

Группа К3441

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**

1. Общий ход работы

Я настроил JSON-server вместе с json-server-auth. Это позволило имитировать полноценную авторизацию по JWT.

Внутри клиентской части был создан axios-инстанс, который автоматически подхватывает базовый URL API и токен из localStorage.

```
const API_BASE = urlApiOverride || localStorage.getItem('careerApiBase') ||
'http://localhost:3001';

const api = axios.create({ baseURL: API_BASE, timeout: 6000 });
```

Обработка сохранённого токена. Если пользователь уже авторизован, токен автоматически встраивается в каждый запрос:

```
const savedToken = localStorage.getItem(TOKEN_KEY);

if (savedToken) {

 api.defaults.headers.common.Authorization = `Bearer ${savedToken}`;

}
```

Приложение хранит состояние в объекте state.

Здесь лежат вакансии, отклики, резюме, данные работодателя и текущий пользователь.

```
const state = {

 vacancies: [...fallbackVacancies],

 applications: [...fallbackApplications],

 employerVacancies: [...fallbackEmployerVacancies],

 resumes: [],

 currentResume: null,

 currentUser: null

};
```

После загрузки страницы приложение проверяет, доступно ли API. Если нет, появляется уведомление, иначе идут запросы.

Для каждой сущности реализованы функции загрузки, сохранения и отрисовки: вакансии, отклики, резюме, вакансии работодателя.

Вся работа с API вынесена в функции вида:

api.get('/vacancies')

api.post('/applications', payload)

api.patch(`/applications/${id}`, { status: next })

api.post('/register', { ... })

api.post('/login', { ... })

После входа токен сохраняется:

```
function setAuthToken(token) {

 if (token) {
```

```
    api.defaults.headers.common.Authorization = `Bearer ${token}`;

    localStorage.setItem(TOKEN_KEY, token);

  } else {

    delete api.defaults.headers.common.Authorization;

    localStorage.removeItem(TOKEN_KEY);

  }

}
```

А текущий пользователь определяется либо по ответу сервера, либо через расшифровку payload токена.

2. Реализация авторизации

Авторизация реализована через json-server-auth.

Отправляется запрос:

```
    const { data } = await api.get(`/users/${payload.sub}`);
```

Сервер возвращает токен и пользователя:

```
 const token = data?.accessToken;
```

Токен сохраняется в localStorage и применяется ко всем запросам:

```
    setAuthToken(token);
```

При регистрации выполняется похожий процесс:

```
    const { data } = await api.post('/register', { email, password, fullName,
username, role });
```

Я сделал функцию, которая умеет декодировать JWT payload:

```
function decodeJwtPayload(token) {

  try {

    const payload = token.split('.')[1] || '';

    const normalized = payload.replace(/-/g, '+').replace(/_/g, '/');
```

```
    const decoded = atob(normalized.padEnd(Math.ceil(normalized.length / 4) * 4,
'='));

    return JSON.parse(decoded);

  } catch {

    return null;

  }

}
```

Это помогает восстановить данные пользователя, если сервер не вернул их напрямую.

3. Работа с вакансиями

Загрузка вакансий с API:

```
 async function loadFromApi() {

 try {

   const { data } = await api.get('/vacancies');

   state.vacancies = Array.isArray(data) ? data : fallbackVacancies;

 } catch {

   state.vacancies = fallbackVacancies;

 }
```

Фильтрация вакансий по форме поиска:

```
function filterVacancies() {

 const query = searchQuery.value.trim().toLowerCase();

 const industry = industryFilter.value;

 const experience = experienceFilter.value;

 const salaryMin = Number(salaryFilter.value);



 return state.vacancies.filter((vacancy) => {
```

```
   if ((vacancy.status || 'опубликована') !== 'опубликована') return false;

   const combined = `${vacancy.title} ${vacancy.company} ${(vacancy.tags ||
[]).join(' ')}`.toLowerCase();

   const matchesQuery = !query || combined.includes(query);

   const matchesIndustry = !industry || vacancy.industry === industry;

   const matchesExperience =

     !experience ||

     vacancy.level === experience ||

     (experience === 'senior' && vacancy.experienceYears >= 5) ||

     (experience === 'middle' && vacancy.experienceYears >= 2 &&
vacancy.experienceYears < 5) ||

     (experience === 'junior' && vacancy.experienceYears < 2);

   const matchesSalary = !salaryMin || (vacancy.salaryFrom || 0) >= salaryMin;

   return matchesQuery && matchesIndustry && matchesExperience && matchesSalary;

 });

}
```

Отрисовка вакансий на странице:

```
function renderVacancies(list) {

 vacancyList.innerHTML = list

   .map(

     (vacancy) => `

       <div class="col-md-6">

         <div class="vacancy-card h-100 d-flex flex-column">

           <div class="d-flex justify-content-between align-items-start mb-2">

             <div>

               <p class="mb-1 text-muted small">${vacancy.company}</p>
```

```
              <h5 class="mb-1">${vacancy.title}</h5>

            </div>

            <span class="pill">${vacancy.posted}</span>

          </div>

          <p class="text-muted small mb-2">${vacancy.city} · ${vacancy.format}</p>

          <div class="d-flex flex-wrap gap-2 mb-3">

            ${(vacancy.tags || []).map((tag) => `<span
class="tag"><i></i>${tag}</span>`).join('')}

          </div>

          <div class="d-flex align-items-center justify-content-between mb-3">

            <span class="fw-semibold">${salaryText(vacancy)}</span>

            <span class="text-muted small">${vacancy.experienceYears}+ лет ·
${vacancy.level}</span>

          </div>

          <p class="text-muted small flex-grow-1">${vacancy.description}</p>

          <div class="d-flex gap-2 mt-3">

            <button class="btn btn-outline-light flex-grow-1"
data-id="${vacancy.id}">Смотреть детали</button>

            <button class="btn btn-primary" data-bs-toggle="modal"
data-bs-target="#applicationModal"
data-vacancy="${vacancy.title}">Откликнуться</button>

          </div>

        </div>

      </div>

    `

  )

  .join('');

}
```

Детальная карточка вакансии обновляется через:

```
function renderDetail(vacancy) {

 if (!vacancy) {

   vacancyDetailCard.innerHTML = '<p class="text-muted mb-0">Выберите вакансию,
чтобы увидеть подробности.</p>';

   detailApplyBtn.dataset.vacancy = '';

   return;

 }
```

5. Работа с откликами кандидатов

При отправке формы создаётся объект заявки:

```
 const payload = {

   name,

   email,

   vacancy: vacancyTitle,

   title: vacancyTitle,

   candidate: name,

   status: 'В рассмотрении',

   message

 };
```

Заявка отправляется на API api.post('/applications', payload)

Если сервер недоступен, заявка сохраняется локально.

Заявки отображаются у кандидата и у работодателя:

```
 renderEmployerApplications();

 renderCandidateApplications();
```

6. Функционал работодателя

Работодатель может создавать вакансию:

api.post('/employerVacancies', newVacancy)

Статус вакансии циклически переключается:

```
const statusOrder = ['опубликована', 'в работе', 'черновик'];
```

Изменение статуса отражается и на общем списке вакансий:

api.patch(`/employerVacancies/${apiId}`, { status: next })

api.patch(`/vacancies/${apiId}`, { status: next })

7. Работа с резюме

Резюме можно создавать и редактировать:

api.post('/resumes', payload)

api.put(`/resumes/${id}`, { ... })

Приложение отображает резюме со всеми полями. Форма резюме открывается в модальном окне Bootstrap.

**Вывод**

Приложение подключено к внешнему API, который полностью реализован через JSON-server + json-server-auth.

Поддержана регистрация, авторизация и автоматическое восстановление пользователя через JWT.

Все данные загружаются, изменяются и сохраняются через HTTP-запросы.

Реализованы CRUD-операции для вакансий, откликов, резюме и списка работодателя.

Интерфейс обновляется динамически, используя данные, полученные от API.