САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа 2: Взаимодействие с внешним API

Выполнил:

Едигарева Дарья

Группа К3439

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. Авторизация и работа с токеном, хранение состояния

Во фронтенде заводится объект состояния:

```
const state = {
 token: localStorage.getItem('accessToken') || '',
 user: localStorage.getItem('user') ?
JSON.parse(localStorage.getItem('user')) : null,
 companies: [],
 vacancies: [],
 employer: null,
 profile: null,
};
```

Если токен есть, он сразу пробрасывается в axios:

```
if (state.token)
api.defaults.headers.common.Authorization = `Bearer
${state.token}`;
```

Функция setAuth(token, user) отвечает за:

- сохранение токена и пользователя в state и localStorage;
- установку/удаление заголовка Authorization;
- смену текста статуса работодателя.

```
function setAuth(token, user) {

 state.token = token || '';

 state.user = user || null;

 if (token) {

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

   localStorage.setItem('accessToken', token);

   localStorage.setItem('user',
JSON.stringify(user));

   employerStatus.textContent = 'Загружаем профиль
работодателя…';

 } else {

   delete api.defaults.headers.common.Authorization;

   localStorage.removeItem('accessToken');

   localStorage.removeItem('user');
```

```
    employerStatus.textContent = 'Требуется
авторизация';
  }
}
```

5.3. Обработка форм логина и регистрации

Обе формы (loginForm и registerForm) обрабатываются одной функцией handleAuthForm:

- определяем, регистрация это или вход по form.id;
- собираем email, password, при регистрации ещё и fullName;
- отправляем POST-запрос на /login или /register:

```
async function handleAuthForm(event) {
 event.preventDefault();
 const form = event.target;
 const isRegister = form.id === 'registerForm';
 const emailInput =
form.querySelector('input[type="email"]');
 const passwordInput =
form.querySelector('input[type="password"]');
 const nameInput =
form.querySelector('input[type="text"]');
 const payload = {
   email: emailInput?.value,
   password: passwordInput?.value,
 };
```

```
 if (isRegister) payload.fullName =
nameInput?.value;


 try {
   const url = isRegister ? '/register' : '/login';
   const { data } = await api.post(url, payload);
   setAuth(data.accessToken, data.user);

bootstrap.Modal.getInstance(form.closest('.modal'))?
.hide();
   await loadEmployerProfile();
   await loadProfile();
 } catch (err) {
   alert('Ошибка авторизации.');
 }
}
```

После успешной авторизации:

- модальное окно закрывается;
- подтягиваются профиль работодателя (loadEmployerProfile()) и профиль соискателя (loadProfile()).

2. Работа с моковым API (JSON-server)

В коде создаётся один axios-клиент с базовым URL (API_BASE), и дальше все операции с данными — загрузка компаний и вакансий, получение/сохранение профиля соискателя (/profiles), создание профиля

работодателя (/employers), создание/редактирование/удаление вакансий (/vacancies) — выполняются через HTTP-запросы к этому моковому API. Авторизация реализована тоже через JSON-сервер: фронт отправляет логин/пароль на /login или /register, получает accessToken и объект пользователя, сохраняет их в localStorage и подставляет токен в заголовок Authorization для всех дальнейших запросов.

Загрузка компаний:

```
async function loadCompanies() {

 try { const { data } = await api.get('/companies');
state.companies = data; }

 catch { state.companies = FALLBACK_COMPANIES; }

 populateCompanies();

}
```

Компании подставляются в селекты для работодателя (employerCompanySelect) и при нормализации вакансий.

Загрузка вакансий:

```
async function loadVacancies() {

 try {

   const { data } = await api.get('/vacancies', {
params: { _expand: 'company' } });

   state.vacancies = data.map(normalizeVacancy);

 } catch {

   state.vacancies =
FALLBACK_VACANCIES.map(normalizeVacancy);

 }

 renderVacancies();
```

```
 renderFavorites();

 renderEmployerTable();

}
```

Используется _expand=company, чтобы JSON-server подставил объект компании внутрь вакансии. Далее вызывается normalizeVacancy, которая приводит вакансию к единому формату: гарантирует наличие полей company, companyId, зарплат, опыта и т.д.

Если запрос упал, берутся заранее зашитые FALLBACK_VACANCIES.

Создание/редактирование вакансии:

POST /vacancies — создание;

PUT /vacancies/:id — обновление, используется в saveVacancy.

Удаление вакансии:

```
async function deleteVacancy(id) {

 if (!state.user) { alert('Нужно авторизоваться');
return; }

 if (!confirm('Удалить вакансию?')) return;

 try { await api.delete(`/vacancies/${id}`); await
loadVacancies(); renderEmployerTable(); }

 catch { alert('Не удалось удалить вакансию'); }

}
```

Эти действия доступны только авторизованному работодателю (проверяется state.user и state.employer).

Создание профиля работодателя:

```
async function createEmployerProfile(event) {
```

```
 event.preventDefault();

 if (!state.user) { alert('Нужно авторизоваться');
return; }

 const companyId = employerCompanySelect.value;

 const phone = employerPhoneInput.value;

 if (!companyId || !phone) { alert('Заполните
данные'); return; }

 try {

   const { data } = await api.post('/employers', {
userId: state.user.id, companyId: Number(companyId),
phone });

   state.employer = data;

   employerStatus.textContent = `Компания:
${state.companies.find((c) => String(c.id) ===
String(companyId))?.name || ''}`;

   employerModal.hide();

   renderEmployerTable();

 } catch (err) { alert('Не удалось сохранить профиль
работодателя.'); }

}
```

Загрузка:

```
async function loadEmployerProfile() {

 if (!state.user) return;

 try {
```

```
   const { data } = await api.get('/employers', {
params: { userId: state.user.id } });

   state.employer = data[0] || null;

   if (state.employer) employerStatus.textContent =
`Компания: ${state.companies.find((c) =>
String(c.id) ===
String(state.employer.companyId))?.name || '—'}`;

   else employerStatus.textContent = 'Профиль
работодателя не найден';

 } catch {

   employerStatus.textContent = 'Ошибка загрузки
профиля работодателя';

 }

 renderEmployerTable();

}
```

Вывод

В ходе лабораторной работы №2 было реализовано взаимодействие фронтенд-приложения JobBridge с внешним (моковым) API:

- подключен JSON-server с набором сущностей: пользователи, компании, вакансии, работодатели и профили;
- настроена авторизация: регистрация, вход, работа с JWT-токеном и заголовком Authorization;
- реализованы операции чтения и изменения данных через HTTP-запросы (GET/POST/PUT/DELETE);
- добавлена работа с localStorage для токена, фильтров, избранного и быстрых настроек поиска.