

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Игнатьев Алексей

К3440

Проверил:

Добряков Д. И.

Санкт-Петербург 2025 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 "Имитация работы с API".

Ход работы

Для авторизации создан специальный модуль с Axios, реализующий работу с access и refresh токенами. Сначала делается запрос по заданному URL с передачей необходимых параметров, а затем парсится ответ и сохраняется в куки сайта

```
export async function POST(req: NextRequest): Promise<NextResponse<{ ok: boolean }> | NextResponse<{ message: string }>> {
  const body: any = await req.json().catch(() => ({}));
  const email: any = (body as any).email;
  const password: any = (body as any).password;
  if (!email || !password) {
    return NextResponse.json({ message: "Email и пароль обязательны" }, { status: 400 });
  }

  const upstream: Response = await upstreamFetch({ path: "/admin-management/auth/login", {
    method: "POST",
    body: JSON.stringify({ email, password }),
    auth: false,
  }});
  const data: any = await upstream.json().catch(() => null as any);
  if (upstream.ok && data?.accessToken && data?.refreshToken) {
    const res: NextResponse<{ ok: boolean }> = NextResponse.json({ ok: true });
    const isHttps: boolean = new URL(req.url).protocol === "https:";
    res.cookies.set({ ...args: "accessToken", data.accessToken, { httpOnly: true, path: "/", sameSite: "lax", secure: isHttps }});
    res.cookies.set({ ...args: "refreshToken", data.refreshToken, { httpOnly: true, path: "/", sameSite: "lax", secure: isHttps }});
    return res;
  }
  return NextResponse.json({ message: data?.message || "Ошибка авторизации" }, { status: upstream.status || 401 });
}
```

При истечении срока аксес токена реализован метод refresh, с аналогичной записей данных в память. Если Refresh token истёк – то обновление не пройдет и выдастся ошибка.

```
export async function POST(req: NextRequest): Promise<NextResponse<{ message: string }> | NextResponse<{ ok: boolean }>> {
  const cookieStore: Promise<ReadonlyRequestCookies> = cookies();
  const refresh: any = cookieStore.get("refreshToken")?.value;
  if (!refresh) return NextResponse.json({ message: "No refresh token" }, { status: 401 });
  const upstream: Response = await fetch(input: `${API_BASE}/admin-management/auth/refresh`, { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify({ refreshToken: refresh })});
  if (!upstream.ok || !data?.accessToken || !data?.refreshToken) return NextResponse.json({ message: "Refresh failed" }, { status: 401 });
  const res: NextResponse<{ ok: boolean }> = NextResponse.json({ ok: true });
  const isHttps: boolean = new URL(req.url).protocol === "https:";
  res.cookies.set({ ...args: "accessToken", data.accessToken, { httpOnly: true, path: "/", sameSite: "lax", secure: isHttps }});
  res.cookies.set({ ...args: "refreshToken", data.refreshToken, { httpOnly: true, path: "/", sameSite: "lax", secure: isHttps }});
  return res;
}
```

Также реализован метод логута, поскольку на сервере не требуется отдельного метода для выхода из профиля, то на фронтенде в нем реализована только чистка данных в куках.

```
export async function POST(): Promise<NextResponse<{ ok: boolean }>> {  
  const res: NextResponse<{ ok: boolean }> = NextResponse.json({ ok: true });  
  res.cookies.set(...args: "accessToken", "", { path: "/", maxAge: 0 });  
  res.cookies.set(...args: "refreshToken", "", { path: "/", maxAge: 0 });  
  return res;  
}
```

Вывод

В результате выполнения работы создан функциональный модуль взаимодействия сайта с методами API и подключена авторизация, рефреш и логат