Лабораторная работа 2: Взаимодействие с внешним API

Студент: Даньшин Семён
Группа: K3440
Дата: 12 января 2026 г.

Тема работы

Взаимодействие с внешним API средствами `fetch`/`axios`. Подключение авторизации и работа с динамическими данными.

Вариант задания

Вариант 6: Свой вариант (Мультибанкинговая платформа)

Приложение взаимодействует с backend-ом микросервисной архитектуры (Core Service, Bank Connector, LLM Agent).

Реализация взаимодействия с API

1. Инструменты

В проекте используются:

- **Axios:** Популярная библиотека для HTTP-запросов, поддерживающая интерцепторы и автоматическое преобразование JSON.
- **Swagger CodeGen:** API-клиент генерируется автоматически на основе спецификации OpenAPI/Swagger (`api.generated.ts`). Это обеспечивает строгую типизацию запросов и ответов.

2. Конфигурация API (`api/api.ts`)

Настроен базовый экземпляр API с интерцепторами для обработки авторизации.

```
import { Api } from "./api.generated";

// Экземпляр API с базовым URL
export const authApi = new Api({
  baseURL: process.env.NEXT_PUBLIC_API_URL ||
"https://core.bankingthing.ru/api",
});

// Интерцептор запросов: добавляет Bearer токен
authApi.instance.interceptors.request.use((config) => {
  const token = localStorage.getItem(ACCESS_TOKEN_KEY);
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
```

```
  }
  return config;
});
```

3. Авторизация и защита маршрутов

Реализована полноценная JWT-авторизация:

1. **Login:** Пользователь вводит учетные данные -> отправляется POST запрос.
2. **Tokens:** Сервер возвращает `accessToken` (живет недолго) и `refreshToken` (живет долго), которые сохраняются в `localStorage`.
3. **Guard:** Защита приватных маршрутов через проверку токена.
4. **Refresh Token:** Реализован механизм автоматического обновления токена при получении ошибки 401.

Код обновления токена (`forceRefreshToken`):

```
export const forceRefreshToken = async (): Promise<string> => {
    const refreshToken = localStorage.getItem(REFRESH_TOKEN_KEY);
    // ... проверки ...
    const response = await tempApi.v1.bankingServiceRefreshToken({
refreshToken });
    // ... сохранение новых токенов ...
    localStorage.setItem(ACCESS_TOKEN_KEY, tokens.accessToken);
    return tokens.accessToken;
};
```

4. Получение данных (Моковое API / Реальное API)

В данной работе вместо простого `json-server` использовано полноценное API.

Пример 1: Получение списка счетов При загрузке дашборда происходит запрос к нескольким эндпоинтам агрегатора.

```
// Компонент SyncStatus.tsx
const fetchSyncStatus = useCallback(async () => {
  try {
    setIsLoading(true);
    // Вызов сгенерированного метода API
    const response = await authApi.v1.bankingServiceGetSyncStatus();
    setSyncStatus(response.data);
  } catch (err) {
    console.error("Failed to fetch sync status", err);
  } finally {
    setIsLoading(false);
  }
}, []);
```

Пример 2: Синхронизация данных (Trigger) Кнопка обновления вызывает метод, который триггерит обновление данных в фоновом режиме.

```
const handleRefresh = async () => {
    // ...
    await authApi.v1.bankingServiceRefreshData({
        bankConnectionId: status.id,
        // ...
    });
    // ...
}
```

5. Динамическое отображение данных

Полученные данные (статусы, балансы, транзакции) рендерятся в компонентах.

Пример отображения статуса синхронизации:

```
{syncStatus?.bankConnections?.map((connection) => (
  <div key={connection.id} className="flex flex-col gap-2">
    <div className="flex justify-between items-center">
      <div className="flex items-center gap-2">
         {/* Иконка и название банка */}
         <Chip>{connection.bankName}</Chip>
      </div>
      {/* Статус синхронизации */}
      <StatusBadge status={connection.status} />
    </div>
    {/* ... */}
  </div>
))}
```

Заключение

В рамках лабораторной работы было реализовано взаимодействие frontend-приложения с внешним REST API. Использование Axios и автогенерируемого клиента позволило упростить работу с сетевыми запросами и обеспечить типобезопасность. Реализован полный цикл аутентификации (вход, хранение токенов, обновление токена) и CRUD операции для банковских данных.