САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронд-энд разработка

Отчет

Лабораторная работа 2

Выполнил:

Рыбкин Михаил

Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

Нужно привязать то, что Вы делали в ЛР1 к внешнему API средствами fetch/axios/xhr.

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 создания серверной части приложения я использовал low-code решение «supabase». Продукт позволяет реализовывать RESTful API посредством создания таблиц и разделения доступов в СУБД PostgreSQL. Подключение созданного сервиса к фронтенду осуществляется путем использования своего JavaScript SDK.

Работу с API я бы разделил на четыре этапа:

Этап 1. Инициализация клиента.

Для начала работы с SDK нужно выполнить функцию createClient, передав ей в аргументах URL-адрес сервера и ключ сервисной авторизации. Чтобы отделить аспекты, касающиеся настроек сервиса, я создал отдельный файл:

src/supabase.js

```
import {createClient} from "@supabase/supabase-js";

const supabaseUrl = process.env.SUPABASE_URL
const supabaseAnonKey = process.env.SUPABASE_ANON_KEY

export const supabase = createClient(
    supabaseUrl,
    supabaseAnonKey
)
```

Переменные для инициализации я беру из environment-переменных окружения, в котором запускается контейнер. Благодаря этому, я могу «запушить» файл в гит-репозиторий без страха раскрыть ключи к сервису.

После инициализации переменную supabase (экземпляр класса SupabaseClient) я могу использовать в любом месте проекта.

Этап 2. Авторизация пользователя.

В Supabase есть набор встроенных методов авторизации и регистрации пользователей. В своей ЛР я использую метод email-password. Для этого мне необходимо создать две формы:

src/components/SignInForm.vue

```
<el-form
    :model="formData"
    :rules="rules"
    label-position="top"
    label-width="auto"
>
  <el-form-item label="Email" prop="email">
    <el-input v-model="formData.email" name="email" type="email"/>
  </el-form-item>
  <el-form-item :label="t('auth.password')" prop="password">
    <el-input v-model="formData.password" show-password type="password"/>
  </el-form-item>
  <el-form-item>
    <el-button
        block
        type="primary"
        @click="signIn">
      {{ t('auth.signInPrompt') }}
    </el-button>
  </el-form-item>
</el-form>
```

src/componenets/SignInForm.vue

```
<el-form
    ref="formDataRef"
    :model="formData"
    :rules="rules"
>
  <el-form-item label="Email" prop="email">
    <el-input v-model="formData.email" name="email" type="email"/>
  </el-form-item>
  <el-form-item :label="t('auth.fio')" prop="fio">
    <el-input v-model="formData.fio" name="name" type="text"/>
  </el-form-item>
  <el-form-item :label="t('auth.password')" prop="password">
    <el-input v-model="formData.password" show-password type="password"/>
  </el-form-item>
  <el-form-item :label="t('auth.repeatPassword')" prop="repeatPassword">
    <el-input v-model="formData.repeatPassword" show-password
type="password"/>
  </el-form-item>
  <el-form-item>
    <el-button
        block
        type="primary"
        @click="signUp">
      {{ t('auth.signUpPrompt') }}
    </el-button>
  </el-form-item>
</el-form>
```

Для обработки запросов и связи с бекендом я создал отдельную папку src/queries, куда по тематическим файлам разложил функции, выполняющие запросы:

src/queries/users.js

```
export const signUp = async function (email, password) {
    const {data, error} = await supabase.auth.signUp({
        email: email,
        password: password
    })
    processError(error)
    return {data}
}
export const signIn = async function (email, password) {
    const {data, error} = await supabase.auth.signInWithPassword({
        email: email,
        password: password
    })
    processError(error)
    return {data}
}
```

После успешной авторизации также нужно повесить EventListener на обновление токенов, для этого вызываем функцию setAuthStateChangeListener:

src/queries/users.js

```
export const setAuthStateChangeListener = async function (setSession,
fetchUserData) {
    supabase.auth.onAuthStateChange(async (event, _session) => {
        await setSession(_session)
        console.debug('Session updated')
        if (event === 'USER_UPDATED') {
            console.debug(event)
            await fetchUserData()
        }
    })
}
```

Этап 3. Выполнение запросов.

После успешной авторизации для выполнения запросов на бекенд нам остается создавать функции и файлы в папке src/queries:

src/queries/courses.js

```
export const createInvite = async function (formData) {
    let {data, error} = await supabase
        .from('courses_invites')
        .upsert(formData)
        .select()
    processError(error)
    return {data}
}
export const fetchInviteInfo = async function (inviteLink) {
    let {data, error} = await supabase
        .from('courses_invites')
        .select(`
            id,
            creator:created_by (
                fio
            ),
            course:course_id (
                id,
                name,
                short_description
            ),
            courses_users (
                user_id
            )
         `)
        .eq('link', inviteLink)
        .maybeSingle()
    processError(error)
    return {data}
}
export const acceptInvite = async function (inviteId, userId, courseId) {
    let {data, error} = await supabase
        .from('courses_users')
        .insert({
            course_id: courseId,
            user_id: userId,
            invite_id: inviteId
        })
        .select()
    processError(error)
    return {data}
}
```

Этап 4. Обработка ошибок.

Каждый запрос возвращает в JSON-ответе стандартизированный ключ error. Для того, чтобы базово обрабатывать ошибки, я создал функцию processError, которая показывает пользователю уведомление о том, что произошла ошибка, если значение ключа error не равно null, и запускаю ее после выполнения каждого запроса на бекенд:

src/helpers.js

```
export const processError = function (error) {
    if (error !== null) {
        console.error(error)
        ElNotification({
            title: 'Error',
            message: "Something's wrong I can feel it",
            type: 'error',
        })
    }
}
```

Вывод

По итогу выполнения лабораторной работы были получены навыки подключения фронтенда к серверной части по REST API.