САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа №2
Взаимодействие с внешним API

Выполнил:

Тюмин Никита

К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Выбранный вариант: Музыкальный плеер.

Задача: Привязать сверстанные в ЛР1 страницы к внешнему API

Ход работы

В качестве внешнего API был использован firebase. Был настроен метод аутентификации и регистрации по логину и паролю. Для хранения моделей пользователей, музыкальных треков, плейлистов использовался сервис firestore database – NoSql база данных. Для хранения аудиофайлов и иззображений использовался сервис firebase storage.

Файл конфигурации firestore:

```
import { initializeApp } from "firebase/app";
import { getFirestore } from "firebase/firestore";
import { getAuth } from "firebase/auth";
import { getStorage } from "firebase/storage";

const firebaseConfig = {
  apiKey: "AIzaSyAHej2dnt5SgPrrllr5rvCgGAVNKIBjakA",
  authDomain: "musec-a84ea.firebaseapp.com",
  projectId: "musec-a84ea",
  storageBucket: "musec-a84ea.appspot.com",
  messagingSenderId: "260194696890",
  appId: "1:260194696890:web:fcf2ecc158bf52d6168c83"
};

// init firebase
initializeApp(firebaseConfig)

// init firestore service
const db = getFirestore()
const auth = getAuth()
const storage = getStorage()

export {
  auth as auth,
  db as db,
  storage as storage,
}
```

Для взаимодействия с базой данных и файловым хранилищем были написаны сервисы storage.js и db.js.

Storage.js:

```
import { storage } from '@/firebase/firebase'
import { ref, getDownloadURL } from "firebase/storage";

class Storage {
```

```
  async getUrl(filename) {
    return getDownloadURL(ref(storage, filename))
  }
}


export default new Storage
```

С помощью класса Storage можео получить ссылку на файл в файловом хранилище по имени файла.

Db.js:

```
import { db } from '@/firebase/firebase'
import { collection, getDocs, getDoc, doc, setDoc, query, where, arrayUnion,
updateDoc } from "firebase/firestore";
import Storage from '@/services/storage'

class DB {
  async index(collectionName, withRefs = [], withLinks = []) {
    const data = []

    let snapshot = await getDocs(collection(db, collectionName))
    for (let doc of snapshot.docs) {
      let nextItem = doc.data()
      nextItem.id = doc.id

      for (let ref of withRefs) {
        getDoc(nextItem[ref]).then(snap => {
          nextItem[ref] = snap.data()
        })
      }
      for (let link of withLinks) {
        let url = await Storage.getUrl(nextItem[link])

        nextItem[link + '_url'] = url
      }

      data.push(nextItem)
    }

    return data
  }

  async find(collectionName, id) {
    let item = await getDoc(doc(db, collectionName, id))

    return item.data()
  }

  async store(collectionName, item, uid) {
    return setDoc(doc(db, collectionName, uid), item)
  }

  async getPlaylists(user_uid) {
    let q

    if (user_uid) {
      q = query(
        collection(db, "playlists"),
        where("is_public", "==", false),
        where('user_uid', '==', user_uid)
      )
    } else {
```

```
      q = query(
        collection(db, "playlists"),
        where("is_public", "==", true),
      )
    }

    const snapshot = await getDocs(q)

    let playlists = []

    for (let doc of snapshot.docs) {
      let nextItem = doc.data()
      nextItem.id = doc.id

      let url = await Storage.getUrl(nextItem.cover)
      nextItem['cover_url'] = url

      let songs = []

      for (let song of nextItem.songs) {
        let nextSong = await getDoc(song)
        nextSong = nextSong.data()

        let url = await Storage.getUrl(nextSong['audio'])
        nextSong.audio_url = url

        let artist = await getDoc(nextSong['artist'])
        nextSong.artist = artist.data()

        songs.push(nextSong)
      }

      nextItem.songs = songs

      playlists.push(nextItem)
    }

    return playlists
  }

  async addToFavorite(user_uid, song_id) {
    console.log('uid', user_uid)
    console.log('sid', song_id)
    let q = query(
      collection(db, "playlists"),
      where("is_public", "==", false),
      where('user_uid', '==', user_uid)
    )

    const snapshot = await getDocs(q)

    let favPlaylist = snapshot.docs[0]
    let song = await getDoc(doc(db, 'songs', song_id))

    if (favPlaylist) {
      let playlistRef = doc(db, 'playlists', favPlaylist.id)

      await updateDoc(playlistRef, {
        songs: arrayUnion(song.ref)
      })
    } else {
      await setDoc(doc(db, 'playlists', (Math.random() +
1).toString(36).substring(2)), {
        name: 'Favorite',
        is_public: false,
        user_uid: user_uid,
```

```
        cover: 'liked.webp',
        songs: [
          song.ref
        ],
      })
    }
  }
}

export default new DB()
```

С помощью класса DB можно вывести все документы коллекции, найти документ по id, сохранить новый документ. Отдельно были реализованы методы получения плейлистов и добавления трека в понравившиеся.

Скрипт регистрации:

```
export default {
  data: () => ({
    user: {
      name: '',
      email: '',
      password: '',
      password_confirm: '',
    },
    errors: {
      name: '',
      email: '',
      password: '',
      password_confirm: '',
    },
  }),

  methods: {
    ...mapMutations({
      setUser: 'auth/setUser'
    }),
    register() {
      this.clearErrors()
      let valid = true

      if (this.user.password !== this.user.password_confirm) {
        this.errors.password_confirm = 'Passwords do not match'
        valid = false
      }
      if (!this.user.name) {
        this.errors.name = 'Name field is required'
        valid = false
      }
      if (!this.user.email) {
        this.errors.email = 'Email field is required'
        valid = false
      }

      if (!valid) return

      createUserWithEmailAndPassword(auth, this.user.email, this.user.password)
        .then((userCredential) => {
          const user = userCredential.user
```

```
          let userData = {
            uid: user.uid,
            name: this.user.name,
            email: this.user.email,
          }

          DB.store('users', userData, userData.uid).then(res => {
            this.setUser(userData)
            this.$router.push({name: 'home'})
          })
        })
        .catch((error) => {
          const errorCode = error.code
          const errorMessage = error.message
        });
    },

    clearErrors() {
      for (let el in this.errors) {
        this.errors[el] = ''
      }
    }
  },
}
```

Скрипты получения треков, плейлистов:

```
export default {
  components: {
    SongRowComponent,
    PlaylistCardComponent,
  },

  data: () => ({
    songs: [],
    playlists: [],
    filter: null,
  }),

  computed: {
    query() {
      return this.$route.query.q
    },

    songsFiltered() {
      return this.songs.filter(el => {
        if (!this.query) {
          return true
        }

        let regexp = new RegExp(this.query.toLowerCase())

        if (regexp.test(el.name.toLowerCase())) {
          return true
        }
        if (regexp.test(el.artist.name.toLowerCase())) {
          return true
        }

        return false
      })
    },

    playlistsFiltered() {
      return this.playlists.filter(el => {
```

```
        if (!this.query) {
          return true
        }

        let regexp = new RegExp(this.query.toLowerCase())

        if (regexp.test(el.name.toLowerCase())) {
          return true
        }

        return false
      })
    },
    showSongs() {
      return this.songsFiltered && this.songsFiltered.length > 0 &&
(this.filter === null || this.filter === 'songs')
    },
    showPlaylists() {
      return this.playlistsFiltered && this.playlistsFiltered.length > 0 &&
(this.filter === null || this.filter === 'playlists')
    },
  },

  watch: {
    query() {
      this.getSongs()
      this.getPlaylists()
    }
  },

  mounted() {
    this.getSongs()
    this.getPlaylists()
  },

  methods: {
    async getSongs() {
      DB.index('songs', ['artist'], ['cover', 'audio']).then(songs => {
        this.songs = songs
      })
    },

    getPlaylists() {
      DB.getPlaylists().then(playlists => {
        this.playlists = playlists
      })
    },

    changeFilter(filter) {
      if (this.filter === filter) {
        this.filter = null
      } else {
        this.filter = filter
      }
    },

    isFilterActive(filter) {
      return this.filter === filter
    }
  },
}
```

Вывод

В ходе Лабораторной работы был реализован музыкальный плеер на Vue.js, который, используя API firebase, позволяет пользователям регистрироваться, аутентифицироваться, прослушивать аудиодорожки и плейлисты, добавлять песни в раздел «понравившиеся».