

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ**

федеральное государственное автономное образовательное учреждение высшего
образования

Санкт-Петербургский национальный исследовательский университет информационных
технологий механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа №6

По дисциплине «Web-программирование»

Добавление авторизации и пользовательских сессий

**Выполнил студент группы
М33122:**

Федотенко Николай Владимирович

Проверил:

Приискалов Роман Андреевич

САНКТ-ПЕТЕРБУРГ

2022

Цель работы:

Реализация механизмов авторизации через сторонних поставщиков услуг и аутентификации на целевом ресурсе.

Краткие теоретические сведения:

- **Guards** – это встроенные механизмы NestJS, имеющие единственную ответственность. Они определяют, будет ли запрос обработан обработчиком маршрута или нет, в зависимости от определённых условий, существующих во время выполнения.
- **JWT (JSON Web Token)** – это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате **JSON**. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности.
- **Passport** – это встроенная в NestJS библиотека для аутентификации по JWT.
- **Middleware** – это встроенный в NestJS механизм ограничения доступа к страницам.

Ход выполнения работы:

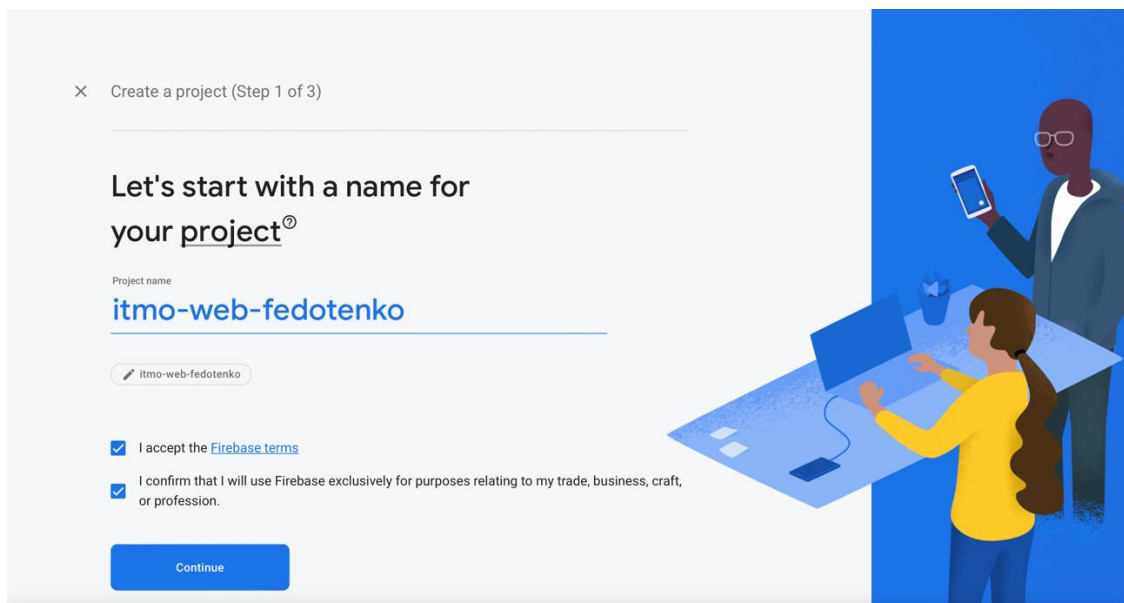
Данная лабораторная работа выполнена в операционной системе *macOS*.

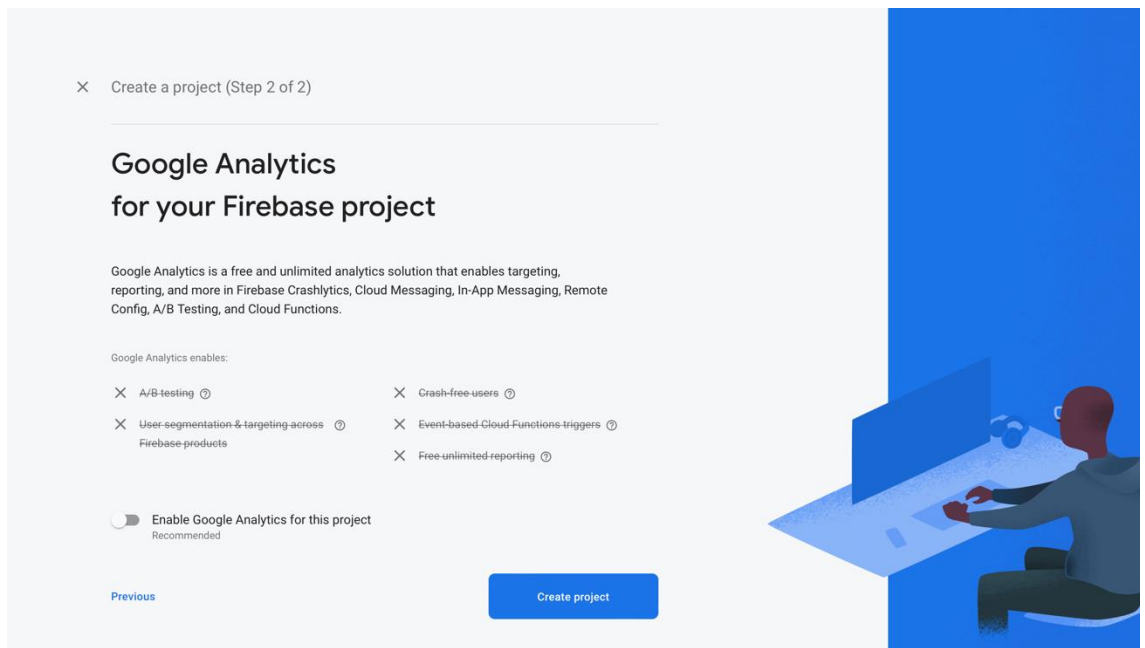
Используемая IDE: *WebStorm 2022.1* (by JetBrains)

Было решено реализовать авторизацию посредством [Google FireBase Auth](#).

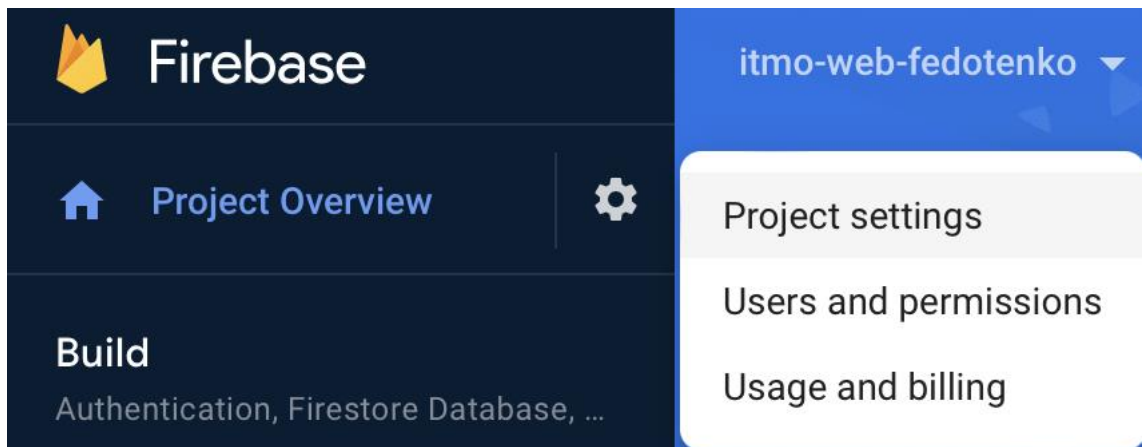
1. Создание приложения Firebase через их [консоль](#):

Для начала создадим проект

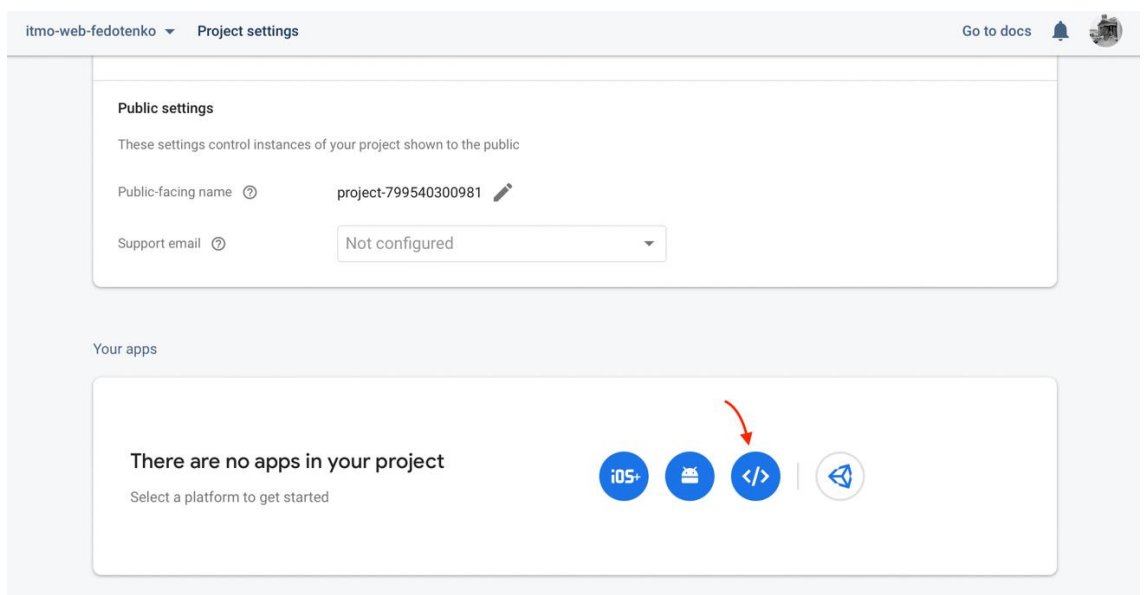




Убедимся, что проект создан, и перейдём в настройки



В настройках нажмём на кнопку создания Web-приложения



× Add Firebase to your web app

1 Register app

App nickname [?](#)

itmo-web-fedotenko-auth

☐ Also set up **Firebase Hosting** for this app. [Learn more](#)

Hosting can also be set up later. There is no cost to get started anytime.

Register app

2 Add Firebase SDK

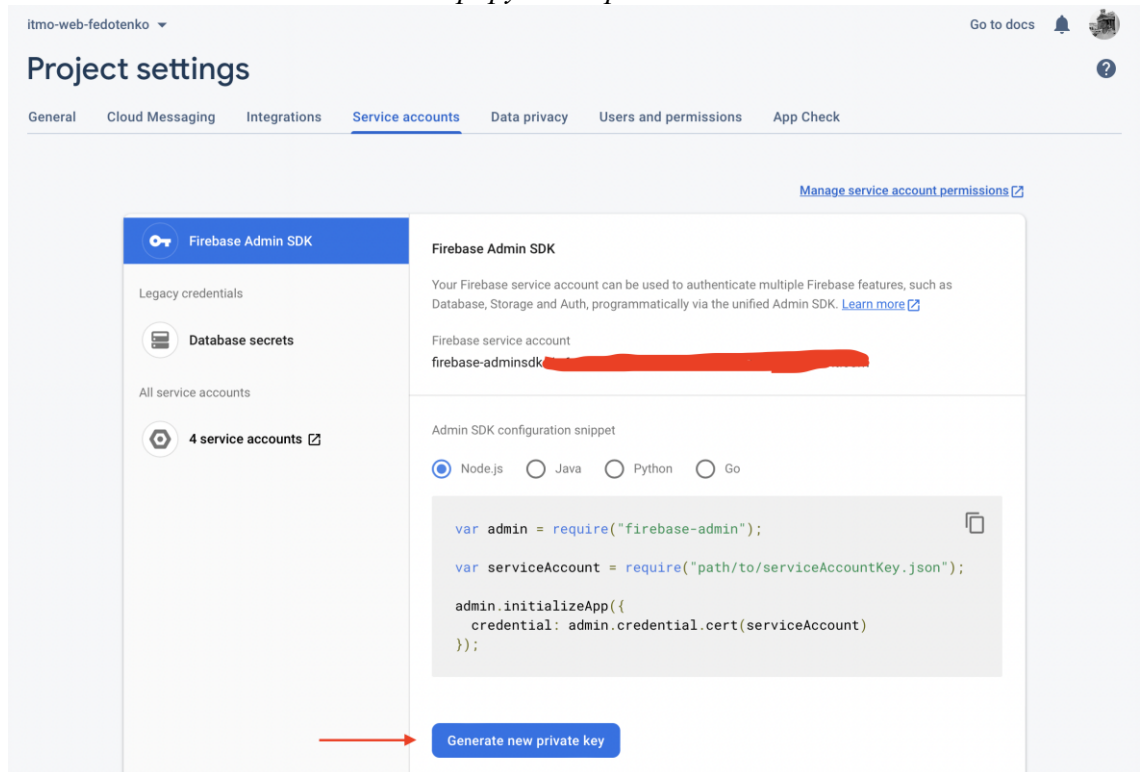
<На втором шаге будет предложен способ инициализации SDK в проекте и также предоставлены все необходимые для этого данные, пока что пропустим этот шаг, предварительно сохранив данные, и завершим создание приложения.>

После этого, в меню проекта обновим данные об аутентификации (будем использовать вход по email и паролю)

The screenshot shows the Firebase console interface. On the left is a dark sidebar with the 'Build' menu expanded, showing options like Authentication, Firestore Database, Realtime Database, Storage, Hosting, and Functions. The main content area is titled 'Authentication' and has tabs for 'Users', 'Sign-in method', 'Templates', and 'Usage'. The 'Sign-in method' tab is active. Under 'Sign-in providers', there is a table with one provider: 'Email/Password', which is 'Enabled' (indicated by a green checkmark). An 'Add new provider' button is in the top right corner of the providers list.

Provider	Status
Email/Password	Enabled

Сгенерируем секретный ключ



2. Установка необходимых компонентов:

```
npm i @nestjs/passport firebase-admin passport passport-firebase-jwt
```

3. Создание отдельного модуля для авторизации:

```
nest g module auth
```

4. Создание конфигурационного файла Firebase (`src/auth/firebase-config.ts`):

```
const firebaseConfig = {
  projectId: process.env.PROJECT_ID,
  clientEmail: process.env.CLIENT_EMAIL,
  privateKey: process.env.PRIVATE_KEY,
};

export default firebaseConfig;
```

Также, пропишем в уже имеющийся `.env` файл (для Prisma) данные трёх указанных переменных среды окружения, которые можно найти в полученном нами JSON'е при генерации секретного ключа. Помимо этого, вынесем каждую из этих переменных в конфигурацию Heroku с помощью команды по типу:

```
heroku config:set PROJECT_ID=MyProjectIDFromJSON
```

5. Создание сервиса **FirebaseApp** (*src/auth/firebase-app.ts*):

```
import { Injectable } from '@nestjs/common';
import * as firebase from 'firebase-admin';
import { firebaseConfig } from '../firebase-config';

@Injectable()
export class FirebaseApp {
  private firebaseApp: firebase.app.App;

  constructor() {
    this.firebaseApp = firebase.initializeApp({
      credential: firebase.credential.cert({ ...firebaseConfig }),
    });
  }

  getAuth = (): firebase.auth.Auth => {
    return this.firebaseApp.auth();
  };

  firestore = (): firebase.firestore.Firestore => {
    return this.firebaseApp.firestore();
  };
}
```

6. Реализация динамического модуля авторизации (*src/auth/auth.module.ts*):

```
import { Module, DynamicModule } from '@nestjs/common';
import { FirebaseApp } from '../firebase-app';

@Module({})
export class AuthModule {
  static forRoot(): DynamicModule {
    return {
      module: AuthModule,
      providers: [FirebaseApp],
    };
  }
}
```

7. Реализация своего Middleware для авторизации (*src/auth/pre-auth-middleware.ts*):

```
import { Injectable, NestMiddleware } from '@nestjs/common';
import * as firebase from 'firebase-admin';
import { Request, Response } from 'express';
import { FirebaseApp } from '../firebase-app';

@Injectable()
export class PreAuthMiddleware implements NestMiddleware {
  private auth: firebase.auth.Auth;

  constructor(private firebaseApp: FirebaseApp) {
    this.auth = firebaseApp.getAuth();
  }

  use(req: Request, res: Response, next: () => void) {
    const token = req.headers.authorization;
    if (token != null && token != '') {
      this.auth
        .verifyIdToken(token.replace('Bearer ', ''))
        .then(async (decodedToken) => {
          // ...
        });
    }
    next();
  }
}
```

```

        req['user'] = {
            email: decodedToken.email,
            roles: decodedToken.roles || [],
            type: decodedToken.type,
        };
        next();
    })
    .catch(() => {
        PreAuthMiddleware.accessDenied(req.url, res);
    });
} else {
    PreAuthMiddleware.accessDenied(req.url, res);
}
}

private static accessDenied(url: string, res: Response) {
    res.status(403).json({
        statusCode: 403,
        timestamp: new Date().toISOString(),
        path: url,
        message: 'Access denied',
    });
}
}
}

```

8. Создание стратегии авторизации (*src/auth/auth.strategy.ts*):

```

import { Injectable } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { firebaseConfig } from '../firebase-config';

@Injectable()
export class AuthStrategy extends PassportStrategy(Strategy) {
    constructor() {
        super({
            jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
            secretOrKey: firebaseConfig.privateKey,
        });
    }

    async validate(payload) {
        console.log(payload);
        return {
            user_id: payload.user_id,
            email: payload.email,
        };
    }
}

```

9. Обновляем информацию в модуле приложения (*src/app.module.ts*):

```

import {
    MiddlewareConsumer,
    Module,
    NestModule,
    RequestMethod,
} from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { AppController } from './app.controller';
import { AppService } from './app.service';

```

```

import { UsersModule } from '../users/users.module';
import { FactModule } from '../fact/fact.module';
import { SkillModule } from '../skill/skill.module';
import { ProjectModule } from '../project/project.module';
import { APP_FILTER } from '@nestjs/core';
import { HttpExceptionFilter } from '../http-exception.filter';
import { AuthModule } from '../auth/auth.module';
import { FirebaseAuth } from '../auth/firebase-app';
import { PreAuthMiddleware } from '../auth/pre-auth-middleware';
import { AuthStrategy } from '../auth/auth.strategy';

@Module({
  imports: [
    ConfigModule.forRoot(),
    AuthModule,
    UsersModule,
    FactModule,
    SkillModule,
    ProjectModule,
  ],
  controllers: [AppController],
  providers: [
    AppService,
    {
      provide: APP_FILTER,
      useClass: HttpExceptionFilter,
    },
    FirebaseAuth,
    AuthStrategy,
  ],
})
export class AppModule implements NestModule {
  configure(consumer: MiddlewareConsumer): any {
    consumer.apply(PreAuthMiddleware).forRoutes({
      path: '/secure/*',
      method: RequestMethod.ALL,
    });
  }
}

```

10. Добавление авторизации в контроллер (src/app.controller.ts):

```

@Post('auth/login')
async login(@Req() req, @Res() res) {
  try {
    const user = await firebase
      .auth()
      .signInWithEmailAndPassword(req.body.email, req.body.password);
    const idToken = await user.user?.getIdToken();
    res.cookie('access_token', idToken);
    this.signed_in = true;
    this.user_email = req.body.email;
    return res.redirect('back');
  } catch (e) {
    console.log('Failed to sign in');
    return res.redirect('back');
  }
}

@Post('auth/register')
async register(@Req() req, @Res() res) {
  try {
    await firebase

```



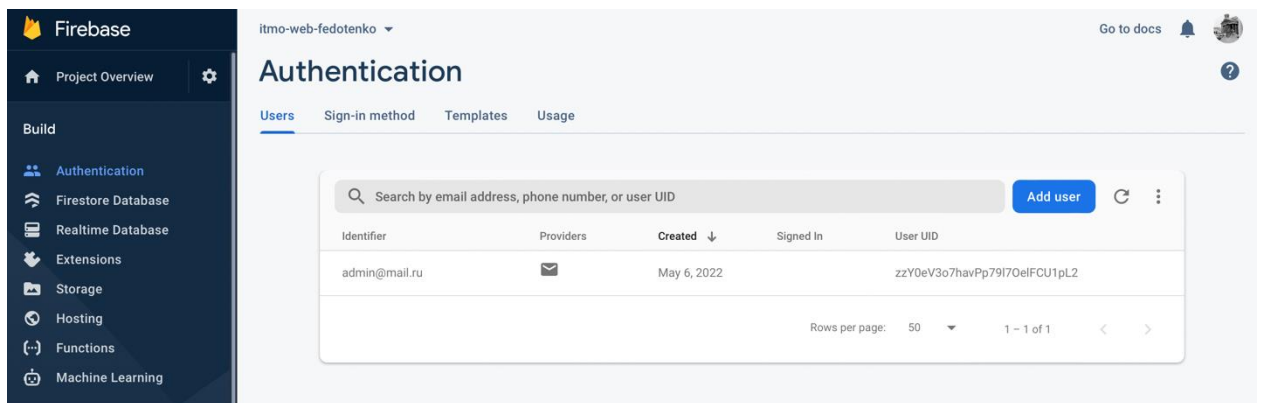
```

    .auth()
    .createUserWithEmailAndPassword(req.body.email,
req.body.password);
    return res.redirect('back');
  } catch (e) {
    console.log('Failed to register');
    return res.redirect('back');
  }
}

@Post('logout')
async logout(@Req() req, @Res() res) {
  res.clearCookie('access_token');
  this.signed_in = false;
  return res.redirect('back');
}

```

11. Добавление пользователя в Firebase:



The screenshot shows the Firebase Authentication console for the project 'itmo-web-fedotenko'. The 'Users' tab is selected, displaying a table of users. The table has columns for Identifier, Providers, Created, Signed In, and User UID. One user is listed: 'admin@mail.ru' with a provider icon, created on 'May 6, 2022', and a User UID of 'zzY0eV3o7havPp79l70eIFCU1pL2'.

Identifier	Providers	Created	Signed In	User UID
admin@mail.ru		May 6, 2022		zzY0eV3o7havPp79l70eIFCU1pL2

12. Проверка результата (<https://itmo-web-fedotenko.herokuapp.com/>):

Николай Федотенко

Николай Федотенко

Вы авторизованы под именем admin@mail.ru

Вывод:

Я реализовал механизмы авторизации через сторонних поставщиков услуг и аутентификации на целевом ресурсе (Google Firebase).