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»

Выполнил:

Студент группы №M33091 Сабитов Искандэр

САНКТ-ПЕТЕРБУРГ

2022

Установим необходимые библиотеки:

```
npm install --save @nestjs/passport passport passport-local


npm install --save-dev @types/passport-local
```

Настроим jwt стратегию в файле jwt.strategy.ts так, чтобы токен извлекался из файлов cookies:

```
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor() {
    super({
      //jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),

      jwtFromRequest: ExtractJwt.fromExtractors( extractors: [
        JwtStrategy.fromCookies,
        ExtractJwt.fromAuthHeaderAsBearerToken(),
      ]),
      ignoreExpiration: false,
      secretOrKey: jwtConstants.secret,
    });
  }

  private static fromCookies(@Req() req: Request): string | null {
    if (
      req.cookies &&
      'token' in req.cookies &&
      req.cookies.token.length > 0
    ) {
      return req.cookies.token;
    }
    return null;
  }

  async validate(payload: any) {
    return { id: payload.sub, email: payload.email };
  }
}
```

Добавим ключ, который используется для создания jwt и проверки его на изменения со стороны пользователя, в constants.ts:

```
export const jwtConstants = {
  secret: 'EzhIsBetterThanEsh',
};
```

Настроим регистрацию и проверку пользователя с хешированием пароля в auth.service.ts:

```
async validateUser(email: string, password: string): Promise<any> {
  const user = await this.usersService.findOneByEmail(email);
  if (user && await bcrypt.compare(password, user.password)) {
    const { password, ...result } = user;
    return result;
  }
  return null;
}

async registration(createUserDto: CreateUserDto): Promise<any> {
  const salt = await bcrypt.genSalt( rounds: 10);
  createUserDto.password = await bcrypt.hash(createUserDto.password, salt);
  const user = await this.usersService.create(createUserDto);
  return user;
}
```

Авторизация пользователя с созданием токена:

```
async login(user: any) {
  const validateRes = await this.validateUser(user.email, user.password);
  if (validateRes) {
    const payload = { email: user.email, sub: validateRes['id'] };
    return {
      access_token: this.jwtService.sign(payload),
      userId: validateRes['id'],
      status: 201
    };
  }
  return { status: validateRes.status };
}
```

Получение id пользователя через токен:

```
getUserId(jwt: string): number{
  return this.jwtService.decode(jwt)['sub'];
```

Добавляем конечную точку для выхода из системы:

```
@Get( path: '/logout')
logout(@Res( options: { passthrough: true }) res: Response){
  res.clearCookie( name: 'token');
```

Скрипт для отправки данных для авторизации(регистрации) на сервер и для записи токена в cookies с установлением времени жизни:

```
async function register() {
  const email = document.querySelector( selectors: "input[name=email]").value;
  const username = document.querySelector( selectors: "input[name=username]").value;
  const password = document.querySelector( selectors: "input[name=password]").value;
  let res = await post(
    '/auth/register',
    {
      name: username,
      email: email,
      password: password
    },
    'POST'
  );
  if(res.ok){
    await login(email, password)
  }
  else
  {
    alert('Error status: ' + res.status)
  }
}
```

```
async function login(...data) {
  let userEmail, userPassword
  if(data.length){
    userEmail = data[0];
    userPassword = data[1];
  }
  else {
    userEmail = document.querySelector( selectors: "input[name=email]").value;
    userPassword = document.querySelector( selectors: "input[name=password]").value;
  }
  let res = await post(
    'auth/login',
    {
      email: userEmail,
      password: userPassword,
    },
    'POST'
  );
  const cookies = await res.json();
  document.cookie = 'token=' + cookies.access_token;
  console.log('meow')
  if(cookies.status === 201){
    let lifeTime = new Date();
    lifeTime.setTime(lifeTime.getTime() + 1200 * 1000);
    document.cookie = 'token=' + cookies.access_token + '; expires=' + lifeTime.toUTCString();
    window.location.replace( url: 'index.html');
  }
```

Проверим наличие токена в браузере:

token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImV4YW1wbG...

Теперь защитим конечные точки и добавим фильтр для того, чтобы пользователя направляло на страницу авторизации, если он не авторизован:

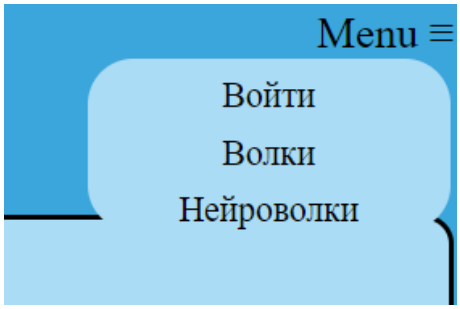
```
@UseGuards(JwtAuthGuard)
@UseFilters(AuthExceptionFilter)
@Get( path: '/myWolves.html')
@Render( template: 'myWolves.pug')
@UseInterceptors(LoggingInterceptor)
getMyWolves(@Request() req) {
  return req;
}
```

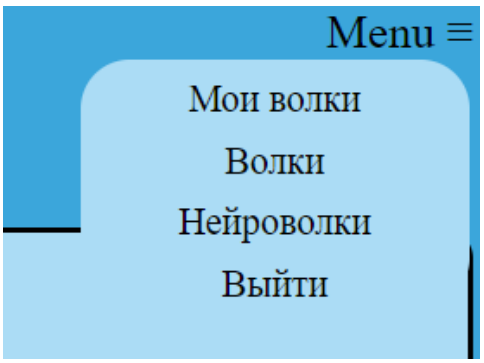
```
import {
  ExceptionFilter,
  Catch,
  ArgumentsHost,
  HttpException,
} from '@nestjs/common';
import { Response } from 'express';
import { UnauthorizedException } from '@nestjs/common';

@Catch(UnauthorizedException)
export class AuthExceptionFilter implements ExceptionFilter {
  catch(exception: HttpException, host: ArgumentsHost) {
    const ctx = host.switchToHttp();
    const response = ctx.getResponse<Response>();
    const status = exception.getStatus();

    response
      .status(status)
      .redirect( url: '/login.html');
  }
}
```

Также изменим конечные точки так, чтобы id пользователя извлекался из токена, передаваемого вместе с запросом в cookies:

```
@ApiBearerAuth()
@ApiOperation( options: {summary: 'Create post'})
@Post()
create(@Req() req: Request, @Body() createPostDto: CreatePostDto) {
  const authorId = this.authService.getUserId(req.cookies.token);
  const data = { imageURL: createPostDto.imageURL, authorId: authorId, wolfType: createPostDto.wolfType }
  return this.postsService.create(data);
}
```

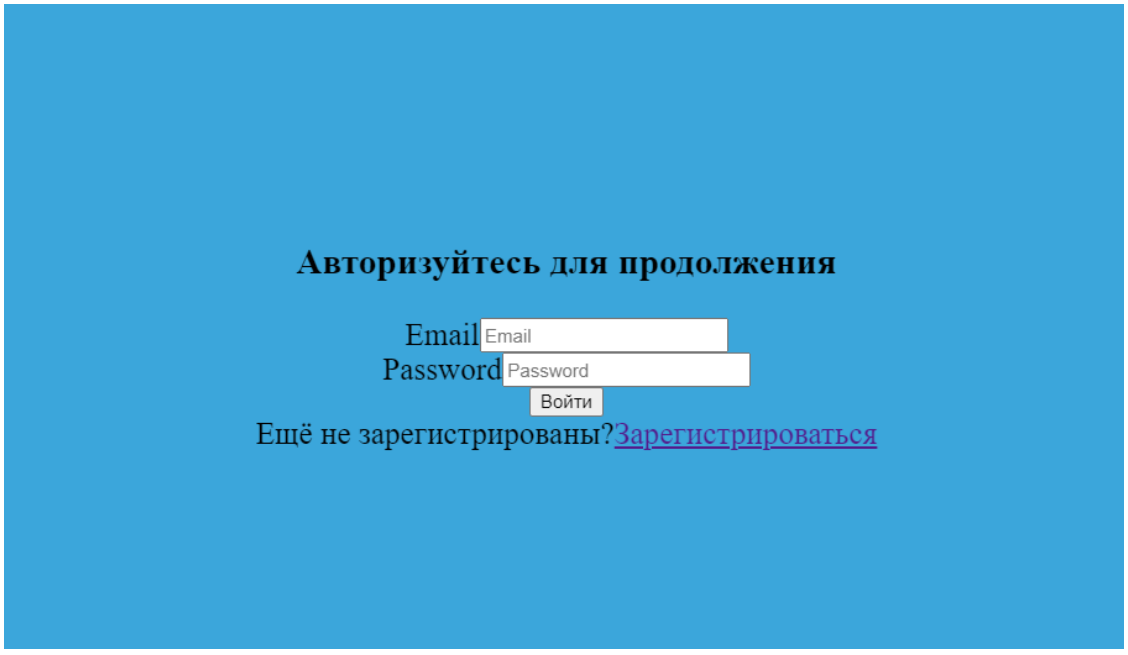
Проверим состояния для авторизованного и неавторизованного пользователя:





Попробуем получить доступ к странице пользователя, когда он не авторизован:

Нас отправляет на страницу авторизации:



На странице пользователя отображаются только его посты, полученные по id из jwt: