19. 선택적 도전 과제-2-2 구현

1. 요구사항

- post로 `/auth/refresh` 로 다음을 보내면 새로운 엑세스 토큰을 발급해준다.

```
{
    "refresh_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InRlc3RAZ21haWwuY29tIiwic3ViI
joxLCJyb2xlIjoiTUVNQkVSIiwiaWF0IjoxNzA2MzMzOTYyLCJleHAiOjE3MDY5Mzg3NjJ9.aSsxo6tpAcL
ggpuXM2D-ozBQ3ybAa6jbWeu7uwzoduM"
}
```

- response

```
{
    "access_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InRlc3RAZ21haWwuY29tIiwic3ViI
joxLCJyb2xlIjoiTUVNQkVSIiwiaWF0IjoxNzA2MzMzOTYyLCJleHAiOjE3MDYzMzQwMjJ9.T1gOhw6l0qN
wI9HzRqKPCNEnheaovZc9E6OJbx7fBlY"
}
```

2. 컨트롤러

- `/auth/auth.controller.ts`

```
@Public()
  @Post('refresh')
  async refresh(@Body('refresh_token') refreshToken: string): Promise<{
access_token: string }> {
    return await this.authService.refreshAccessToken(refreshToken);
  }
```

3. 로그인 서비스

- 기본적으로 엑세스 토큰과 리프레쉬 토큰을 구분해야 한다.

- 그리고 db에도 저장되어야 하므로 우선 로그인 서비스를 수정하기 전에 엔티티에 refreshToken이 추가되어야한다.

```
import { Entity, PrimaryGeneratedColumn, Column } from 'typeorm';

// UserRole enum 정의
export enum UserRole {
  MEMBER = 'MEMBER',
  ADMIN = 'ADMIN',
}

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ unique: true })
  username: string;

  @Column()
  password: string;

  @Column({
    type: 'enum',
    enum: UserRole,
    default: UserRole.MEMBER
  })
  role: UserRole;

  // 가입시 말도 안되게 어려운 코드로 설정하여 허위 인증 방지
  @Column({ default: Math.random().toString(36) })
  verificationCode: string;

  @Column({ default: false })
  isVerified: boolean;

  @Column({ nullable: true })
  refreshToken: string;
}
```

- 다음과 같이 유저마다 refresh 토큰이 있고, db에 각각 집어넣을 것임.
- `auth/auth.service.ts`

```
async signIn(username: string, pass: string): Promise<{ access_token: string,
refresh_token: string }> {
    const user = await this.usersService.findByUsername(username);
    if (!user || !(await this.comparePasswords(pass, user.password))) {
      throw new UnauthorizedException();
    }
    const payload = { username: user.username, sub: user.id, role: user.role };
    const refresh_token = await this.jwtService.signAsync(payload, {
      expiresIn: '7d'  // Refresh 토큰 유효기간 설정
    });

    // refresh token 업데이트
    user.refreshToken = refresh_token;
    await this.userRepository.save(user);

    return {
      access_token: await this.jwtService.signAsync(payload, {
        expiresIn: '60s'
      }),
      refresh_token,
    };
  }
```

- 다음과 같이 리프레쉬 토큰을 추가로 발급하여 user db에 저장하는 코드를 짜게되었다.

```
mysql> select * from opstest.user;
+----+----------------+------------------------------------------------------------
--+--------+-----------------+------------+-------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------+
| id | username       | password
                     | role   | verificationCode | isVerified | refreshToken


           |
+----+----------------+------------------------------------------------------------
--+--------+-----------------+------------+-------------------------------------
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------+
|  1 | test@gmail.com |
$2b$10$kp1sLaHJUZbRzB6mq62QHuCNHtD0CtEPR3t4HhyGqXwySwE.PyWT6 | MEMBER |
0.s9rzakrnl6     |          0 |
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InRlc3RAZ21haWwuY29tIiwic3ViIj
oxLCJyb2xlIjoiTUVNQkVSIiwiaWF0IjoxNzA2MzMzOTYyLCJleHAiOjE3MDY5Mzg3NjJ9.aSsxo6tpAcLg
gpuXM2D-ozBQ3ybAa6jbWeu7uwzoduM |
+----+----------------+-----------------------------------------------------------
--+--------+-----------------+------------+-------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

- refreshToken 잘 발급되어 들어가는 것을 볼 수 있음.

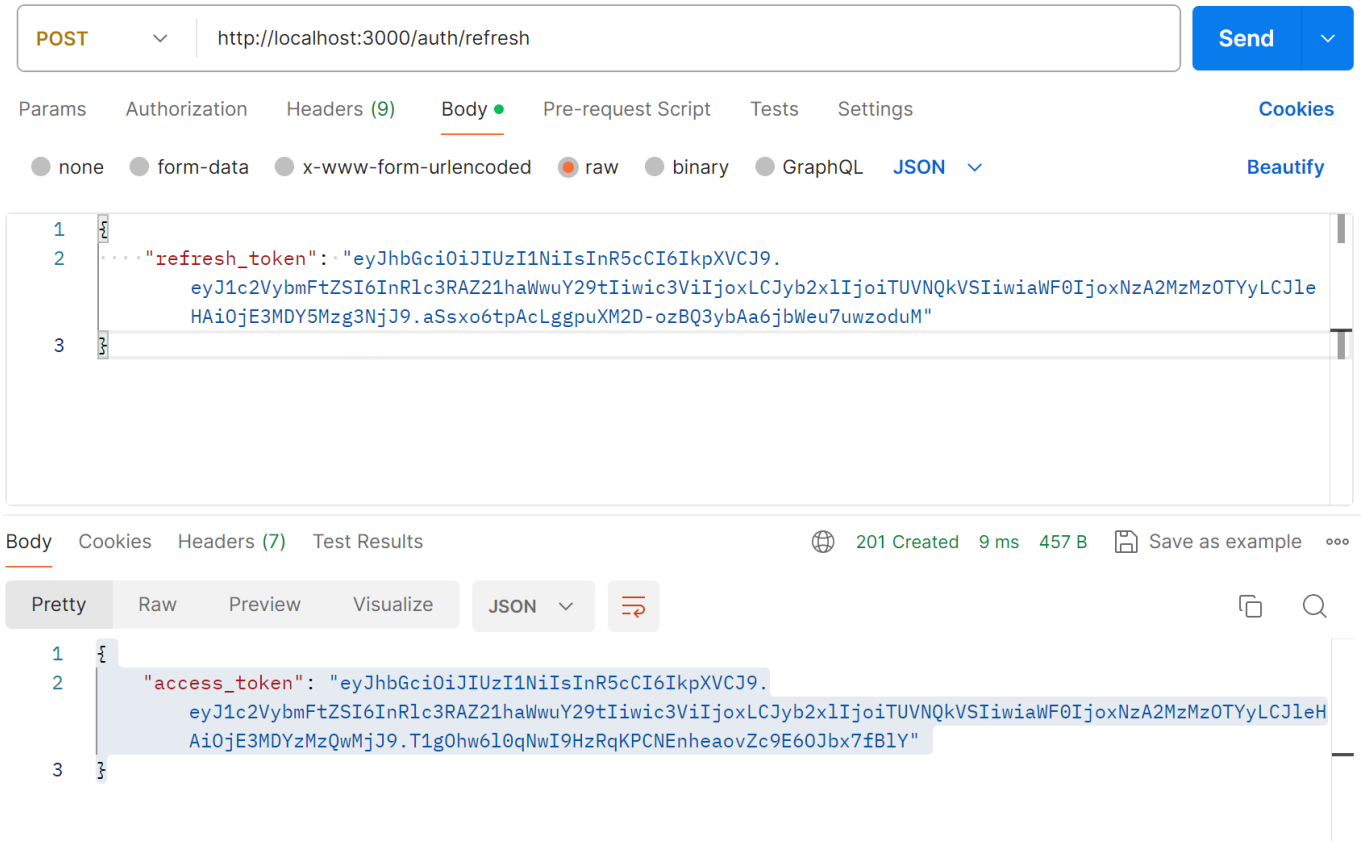
4. refreshAccessToken

- 다음과 같이 refresh 토큰을 넣으면 새로운 access token을 발급해주는 코드로 짰다.

```
  async refreshAccessToken(refreshToken: string): Promise<{ access_token: string }>
{

    console.log('refreshToken : ' + refreshToken);

    try {
      const payload = await this.jwtService.verifyAsync(refreshToken);

      // 'exp' 속성 제거
      delete payload.exp;

      const newAccessToken = await this.jwtService.signAsync(payload, {
        expiresIn: '60s'
      });
      return { access_token: newAccessToken };
    } catch (error) {
      console.log(error);

      throw new UnauthorizedException('Refresh token is invalid');
```

```
    }
  }
```

5. 결과 확인



- 다음과 같이 성공적으로 refrsh 토큰을 넣어 access_token으로 발급해주는 것을 볼 수 있다.