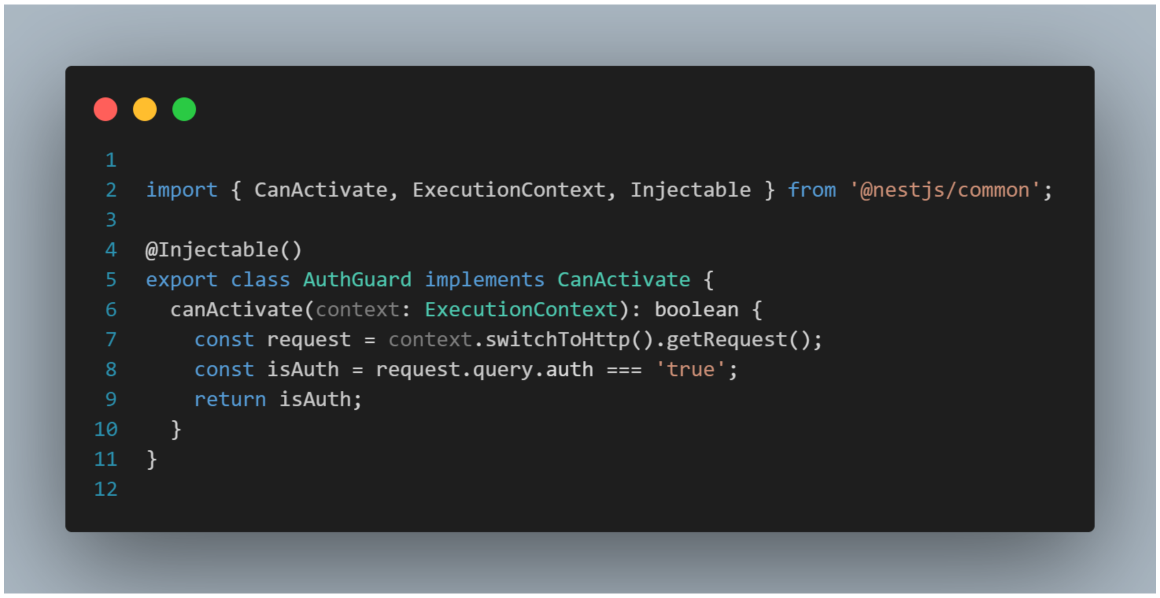
NestJS Assignment - 10

1. What Are Guards in NestJS?

Guards are used to control access to routes. They determine whether a request should be handled based on custom logic like authentication or roles.

2. How Do You Create a Custom Guard in NestJS?



```
import { CanActivate, ExecutionContext, Injectable } from '@nestjs/common';

@Injectable()
export class AuthGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean {
    const request = context.switchToHttp().getRequest();
    const isAuth = request.query.auth === 'true';
    return isAuth;
  }
}
```

3. What Is the ExecutionContext in NestJS Guards and How Is It Used?

ExecutionContext provides details about the current request. It helps extract request data like headers, body, and user for decision-making.

4. At What Levels Can Guards Be Applied in a NestJS Application?

Guards can be applied at method, controller, or global level using decorators like @UseGuards() or app.useGlobalGuards().

5. Implement Role-Based Access Control Using a Guard.

Create a RolesGuard that checks the user role from request object. Apply it using @UseGuards(RolesGuard) and pass required roles via a custom @Roles() decorator.

6. How would you use an interceptor to transform the response before sending it to the client?

Use a class implementing NestInterceptor and override the intercept method. Use map() to modify the response inside RxJS pipe.

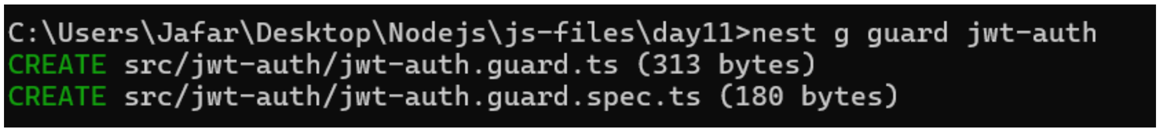
7. How can you apply interceptors to specific routes or controllers?

Use the @UseInterceptors() decorator on the controller or route handler. You can also apply them globally in main.ts.

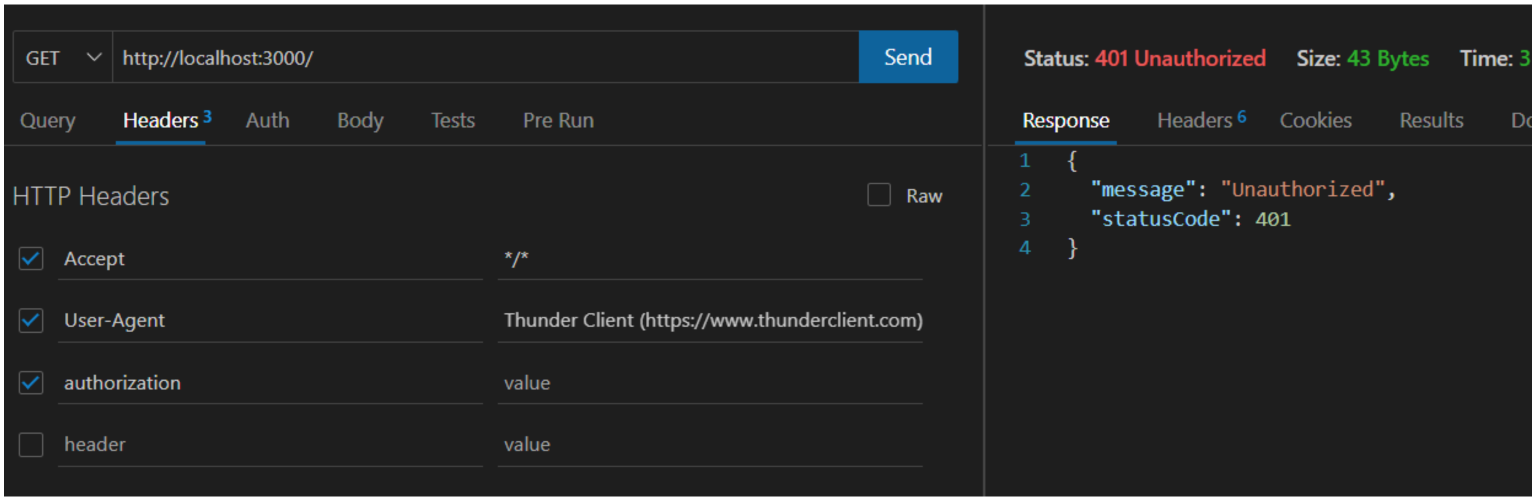
8. How Do Guards Interact with Other Components in the NestJS Lifecycle?

Guards run before interceptors and pipes. They decide if execution continues. If a guard returns false, the request is denied.

9. Handling Asynchronous Validation in a Guard



```
C:\Users\Jafar\Desktop\Nodejs\js-files\day11>nest g guard jwt-auth
CREATE src/jwt-auth/jwt-auth.guard.ts (313 bytes)
CREATE src/jwt-auth/jwt-auth.guard.spec.ts (180 bytes)
```



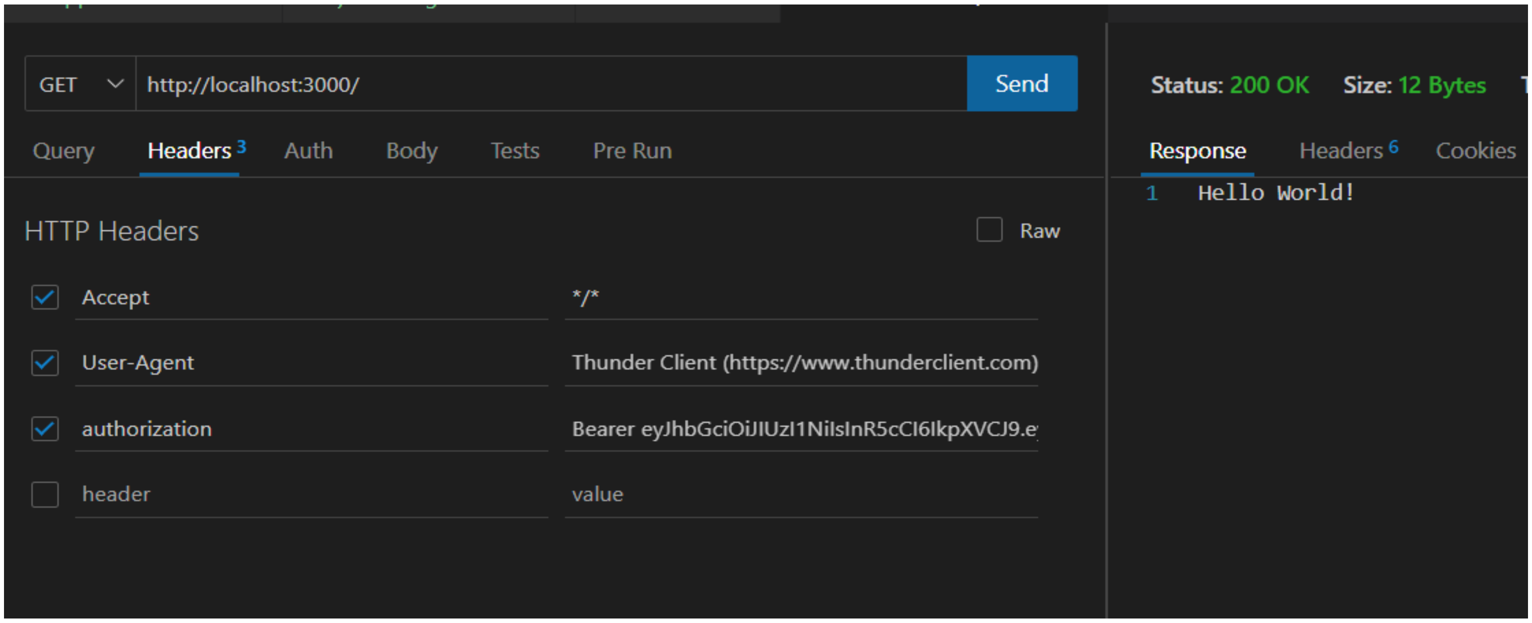
http://localhost:3000/

Status: 401 Unauthorized Size: 43 Bytes Time: 3

HTTP Headers

✓	Accept	*/*
✓	User-Agent	Thunder Client (https://www.thunderclient.com)
✓	authorization	value
	header	value

```
{
  "message": "Unauthorized",
  "statusCode": 401
}
```



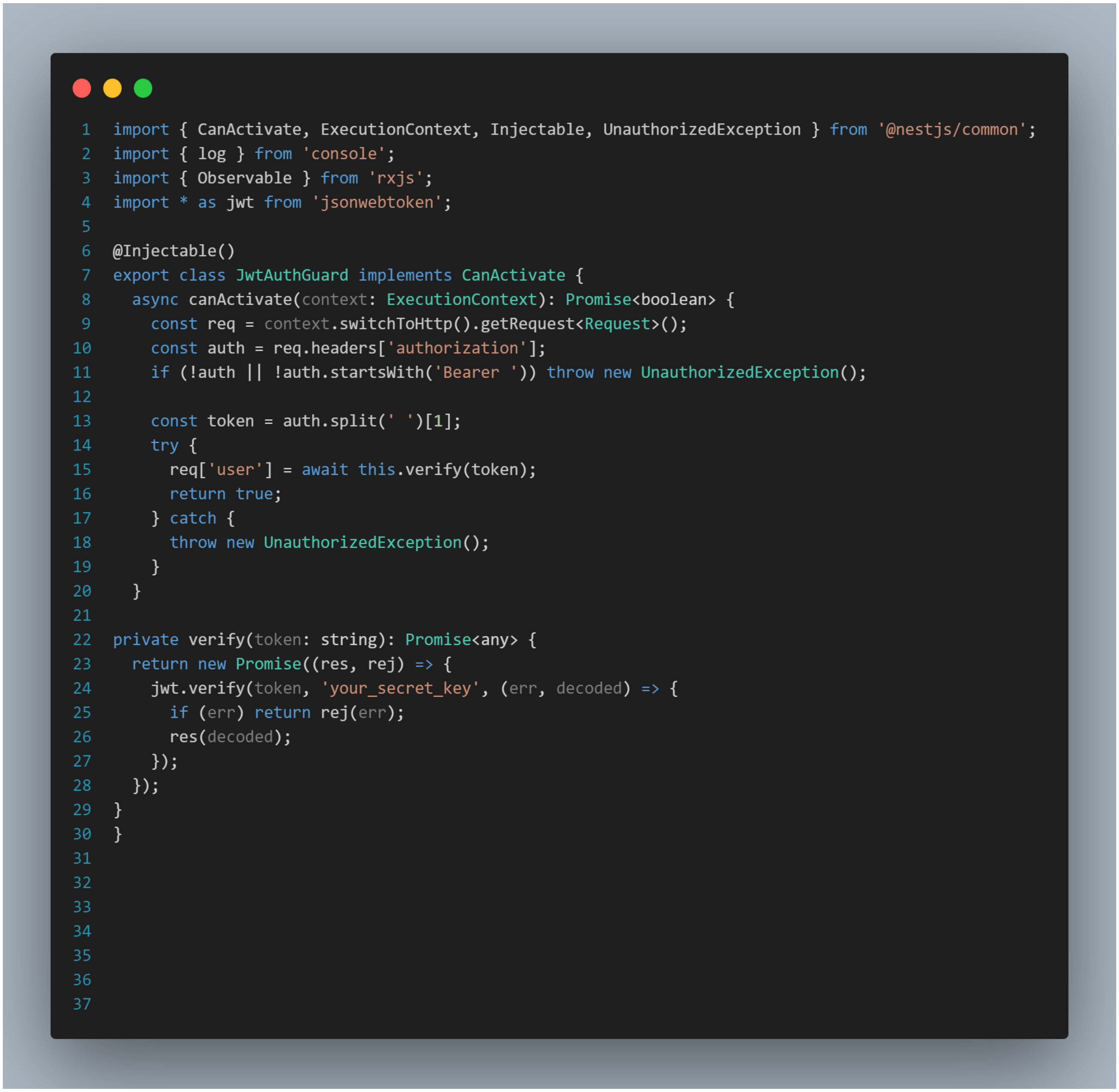
http://localhost:3000/

Status: 200 OK Size: 12 Bytes

HTTP Headers

✓	Accept	*/*
✓	User-Agent	Thunder Client (https://www.thunderclient.com)
✓	authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e
	header	value

```
Hello World!
```

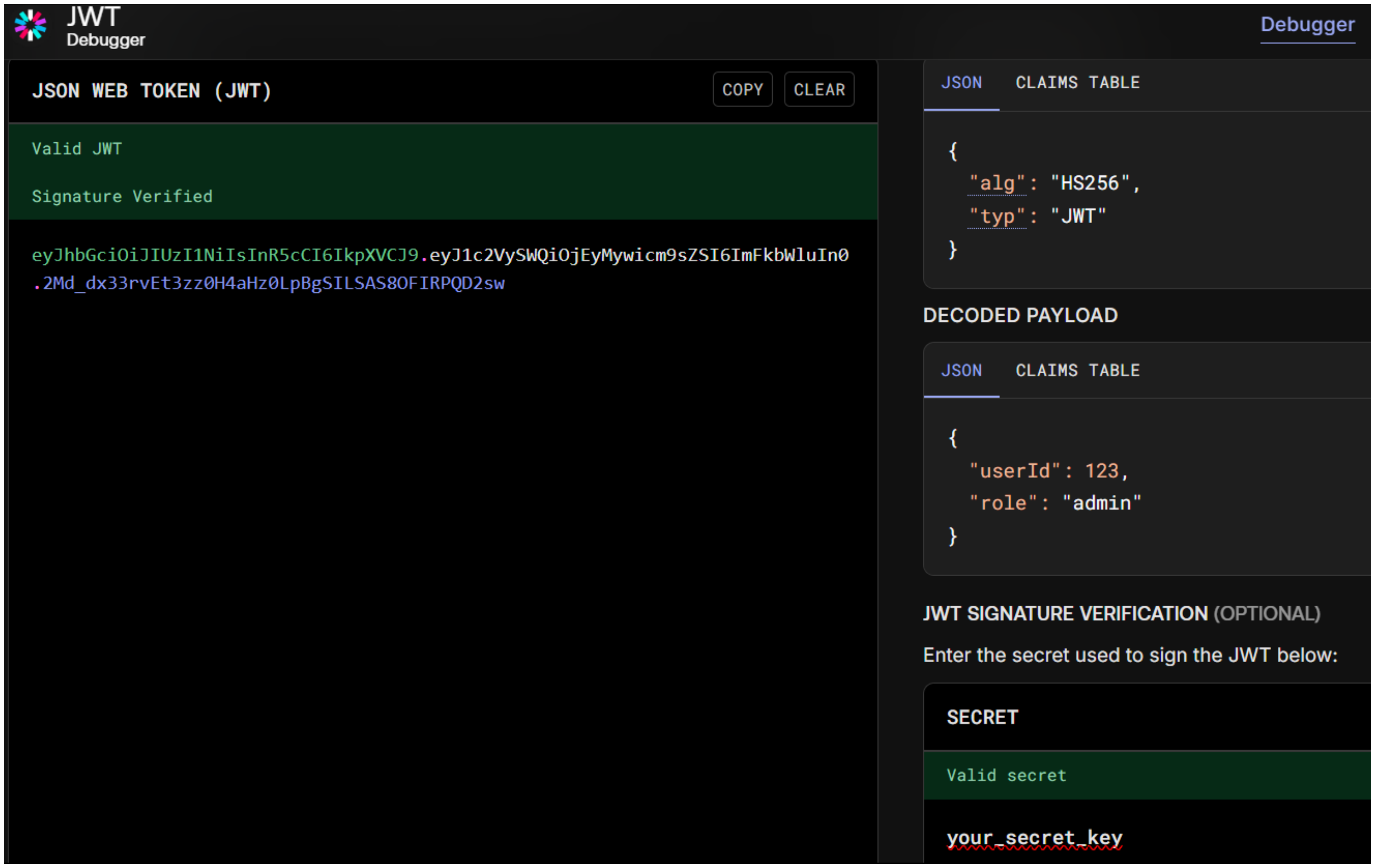


```
import { CanActivate, ExecutionContext, Injectable, UnauthorizedException } from '@nestjs/common';
import { log } from 'console';
import { Observable } from 'rxjs';
import * as jwt from 'jsonwebtoken';

@Injectable()
export class JwtAuthGuard implements CanActivate {
  async canActivate(context: ExecutionContext): Promise<boolean> {
    const req = context.switchToHttp().getRequest<Request>();
    const auth = req.headers['authorization'];
    if (!auth || !auth.startsWith('Bearer ')) throw new UnauthorizedException();

    const token = auth.split(' ')[1];
    try {
      req['user'] = await this.verify(token);
      return true;
    } catch {
      throw new UnauthorizedException();
    }
  }

private verify(token: string): Promise<any> {
  return new Promise((res, rej) => {
    jwt.verify(token, 'your_secret_key', (err, decoded) => {
      if (err) return rej(err);
      res(decoded);
    });
  });
}
}
```



JSON WEB TOKEN (JWT)

Valid JWT

Signature Verified

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEyMywicm9sZSI6ImFkbWluIn0
.2Md_dx33rvEt3zz0H4aHz0LpBgSILSAS8OFIRPQD2sw
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

DECODED PAYLOAD

```
{
  "userId": 123,
  "role": "admin"
}
```

JWT SIGNATURE VERIFICATION (OPTIONAL)

Enter the secret used to sign the JWT below:

SECRET

Valid secret

your_secret_key