NESTJS RATE LIMITING SERIES

✓ **Framework: NestJS v11.0.5**

- Packages:
  @nestjs/throttler: ^6.4.0

Part 1

- WHY RATE LIMITING
- GLOBAL THROTTLING SETUP

Why Rate Limiting Matters

- **Protects your backend services**
  Without rate limiting, a malicious user or buggy frontend could flood your API with requests and overwhelm your app or microservices.

- **Prevents abuse (e.g. brute-force attacks, scraping)**
  Rate limiting blocks attackers trying to spam login forms or steal your data by scraping endpoints.

- **Ensures fair usage**
  Prevents any single client from hogging your API resources.

- **Helps your system stay stable under load**
  Good rate limiting makes your app more reliable even during peak traffic.

@nestjs/throttler

Why use @nestjs/throttler?

- Built and maintained by the NestJS team.
- Integrates cleanly with NestJS apps.
- Provides both global and per-route rate limiting.
- Works well with API gateways, microservices, and proxies.
- Supports custom strategies (e.g., rate limit by IP, user ID, or both).

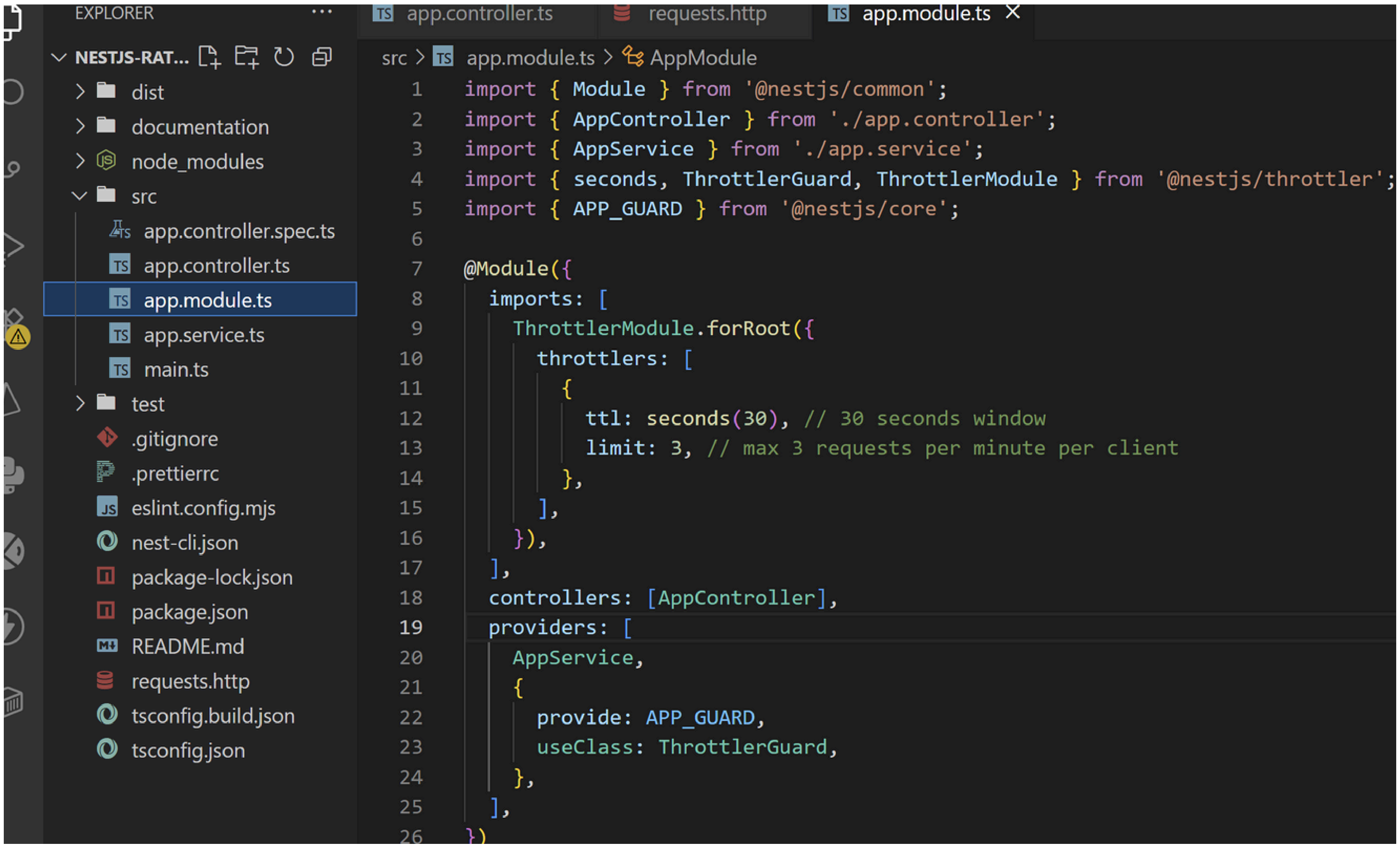
How to install

- npm install @nestjs/throttler

Next >> Global Rate Limiting

Global Rate Limiting

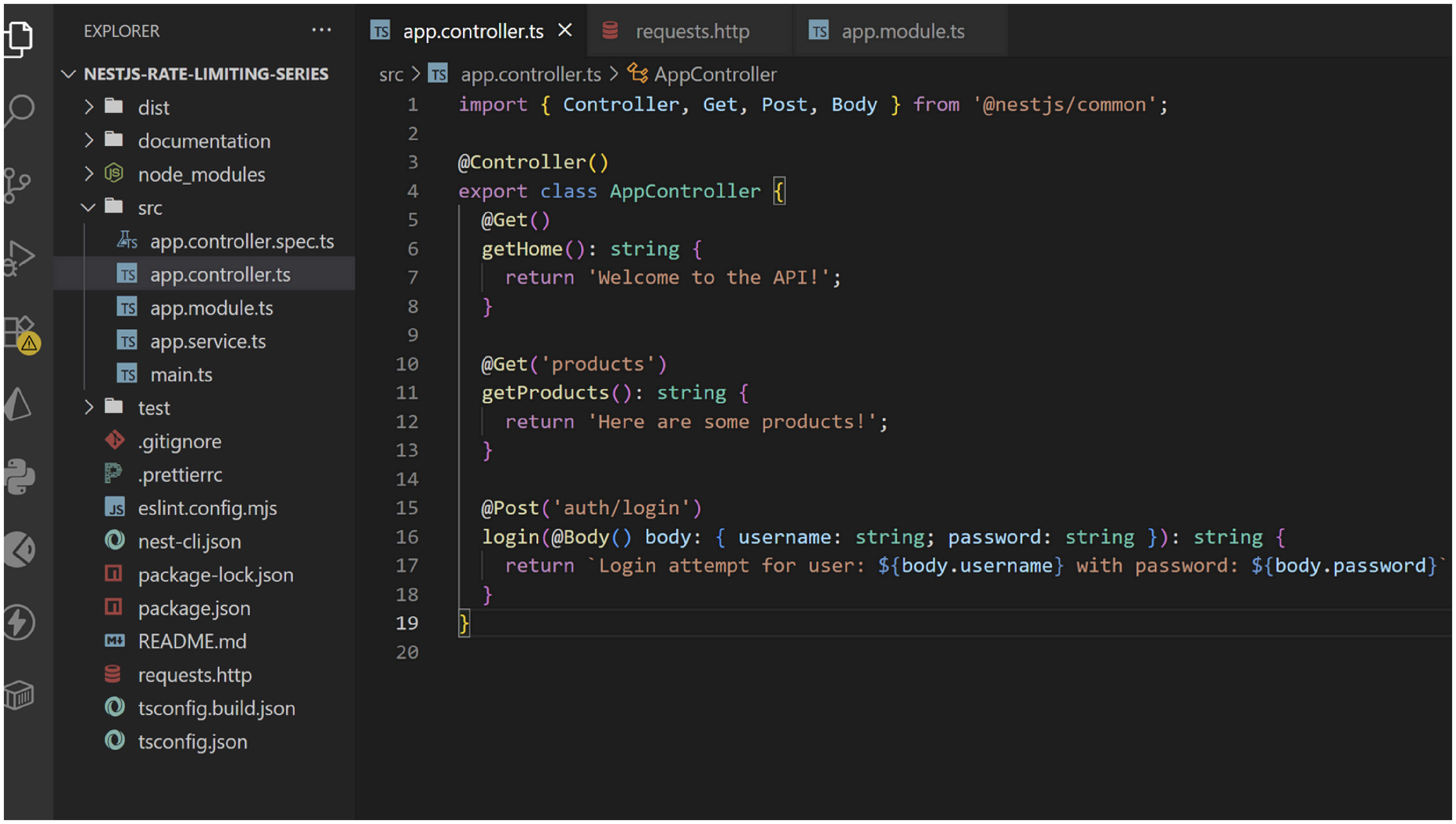
Add ThrottlerModule to AppModule



```typescript
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { seconds, ThrottlerGuard, ThrottlerModule } from '@nestjs/throttler';
import { APP_GUARD } from '@nestjs/core';

@Module({
  imports: [
    ThrottlerModule.forRoot({
      throttlers: [
        {
          ttl: seconds(30), // 30 seconds window
          limit: 3, // max 3 requests per minute per client
        },
      ],
    }),
  ],
  controllers: [AppController],
  providers: [
    AppService,
    {
      provide: APP_GUARD,
      useClass: ThrottlerGuard,
    },
  ],
})
```

Controller

Set up Controller

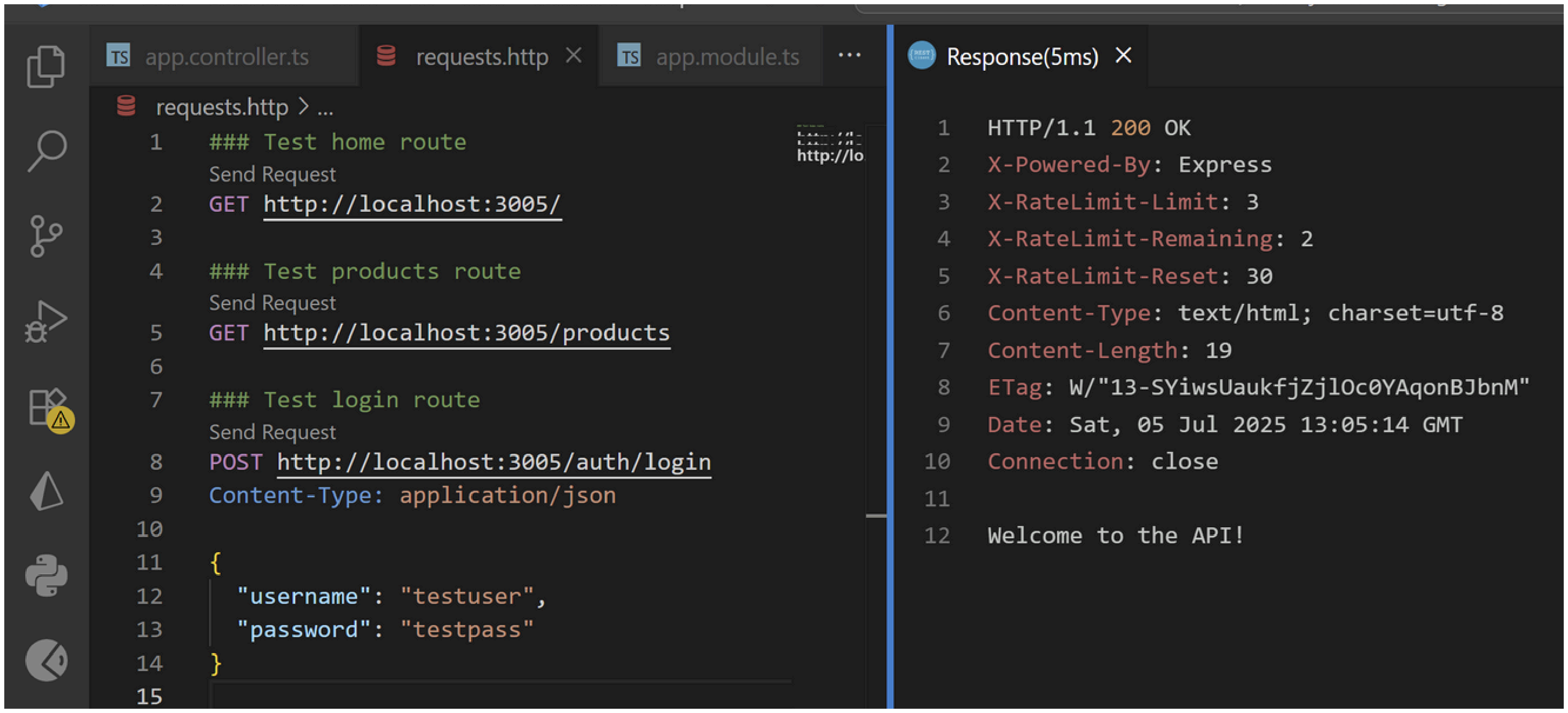


```typescript
import { Controller, Get, Post, Body } from '@nestjs/common';

@Controller()
export class AppController {
  @Get()
  getHome(): string {
    return 'Welcome to the API!';
  }

  @Get('products')
  getProducts(): string {
    return 'Here are some products!';
  }

  @Post('auth/login')
  login(@Body() body: { username: string; password: string }): string {
    return `Login attempt for user: ${body.username} with password: ${body.password}`
  }
}

```

◆ Testing the Endpoints with requests.http

Initial test: successful request

We'll use a request.http file to send test requests directly from our editor



```http
### Test home route
Send Request
GET http://localhost:3005/

### Test products route
Send Request
GET http://localhost:3005/products

### Test login route
Send Request
POST http://localhost:3005/auth/login
Content-Type: application/json

{
  "username": "testuser",
  "password": "testpass"
}

```

```
HTTP/1.1 200 OK
X-Powered-By: Express
X-RateLimit-Limit: 3
X-RateLimit-Remaining: 2
X-RateLimit-Reset: 30
Content-Type: text/html; charset=utf-8
Content-Length: 19
ETag: W/"13-SYiwsUaukfjZjlOc0YAqonBJbnM"
Date: Sat, 05 Jul 2025 13:05:14 GMT
Connection: close

Welcome to the API!
```

The server responded with:

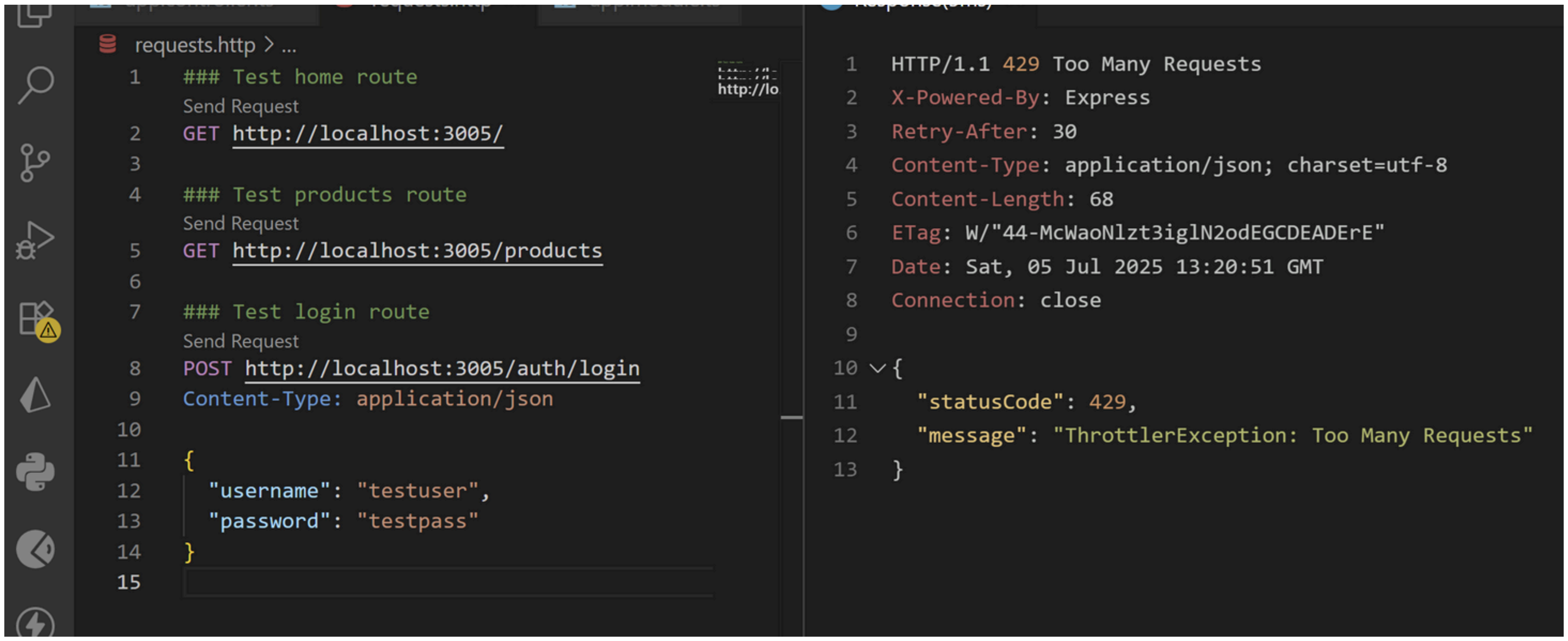
- 200 OK status – the request succeeded.
- X-RateLimit-Limit: 3 – shows our configured limit (3 requests per 30 seconds).
- X-RateLimit-Remaining: 2 – we've used 1 out of 3 allowed requests.
- X-RateLimit-Reset: 30 – time (in seconds) until the rate limit window resets.

🔹 Testing the Endpoints with requests.http

Sending multiple requests

The first three requests returned 200 OK.

❌ The fourth request triggered our rate limiter, and we received a 429 Too Many Requests response.



requests.http > ...

```
### Test home route
Send Request
GET http://localhost:3005/

### Test products route
Send Request
GET http://localhost:3005/products

### Test login route
Send Request
POST http://localhost:3005/auth/login
Content-Type: application/json

{
  "username": "testuser",
  "password": "testpass"
}
```

```
HTTP/1.1 429 Too Many Requests
X-Powered-By: Express
Retry-After: 30
Content-Type: application/json; charset=utf-8
Content-Length: 68
ETag: W/"44-McWaoNlzt3iglN2odEGCDEADErE"
Date: Sat, 05 Jul 2025 13:20:51 GMT
Connection: close

{
  "statusCode": 429,
  "message": "ThrottlerException: Too Many Requests"
}
```

- request.http is a simple and effective way to test your API directly from your code editor.
- The X-RateLimit-* headers help you monitor rate limit status and remaining allowance.
- Our global rate limit is working: after 3 requests in 30 seconds, further requests are blocked as expected.

What our current setup does

- We’ve applied global rate limiting at the app level using ThrottlerModule.forRoot().
- Each route is rate-limited individually:

If a client hits the limit on one endpoint, they can still access other endpoints without restriction during the same window.

- Example: Exceeding the limit on /auth/login doesn’t block access to /products.

What’s the challenge?

➡ Sometimes you want rate limits to apply across all routes for a client, not just per route.

➡ This helps stop attackers from spamming different endpoints to bypass limits

What’s next in Part 2

- We’ll explore how to override limits per route using @Throttle() and @SkipThrottle().
- We’ll also cover how to create a custom throttler that applies limits globally across all endpoints for a client!