NESTJS RATE LIMITING SERIES

✓ **Framework: NestJS v11.0.5**

- Packages: @nestjs/throttler: ^6.4.0

Part 2

- Customizing Rate Limits in NestJS

Let’s take rate limiting a step further

In real-world APIs, some routes may need stricter or looser limits, or none at all. NestJS gives us the tools to fine-tune that.

In this part, you'll learn:

✅ How to override global rate limits using @Throttle()
✅ How to exclude routes from throttling using @SkipThrottle()
✅ Why per-route control is essential in real-world APIs

NestJS gives you all the tools. Let’s see them in action! ✅

@SkipThrottle() Decorator

What is @SkipThrottle()?

It's a decorator provided by @nestjs/throttler that tells NestJS to ignore rate limiting for the decorated route or controller.

What it does:

✅ Disables throttling temporarily for specific endpoints or entire controllers.
✅ Useful for public routes like /login, /signup, or /healthcheck.

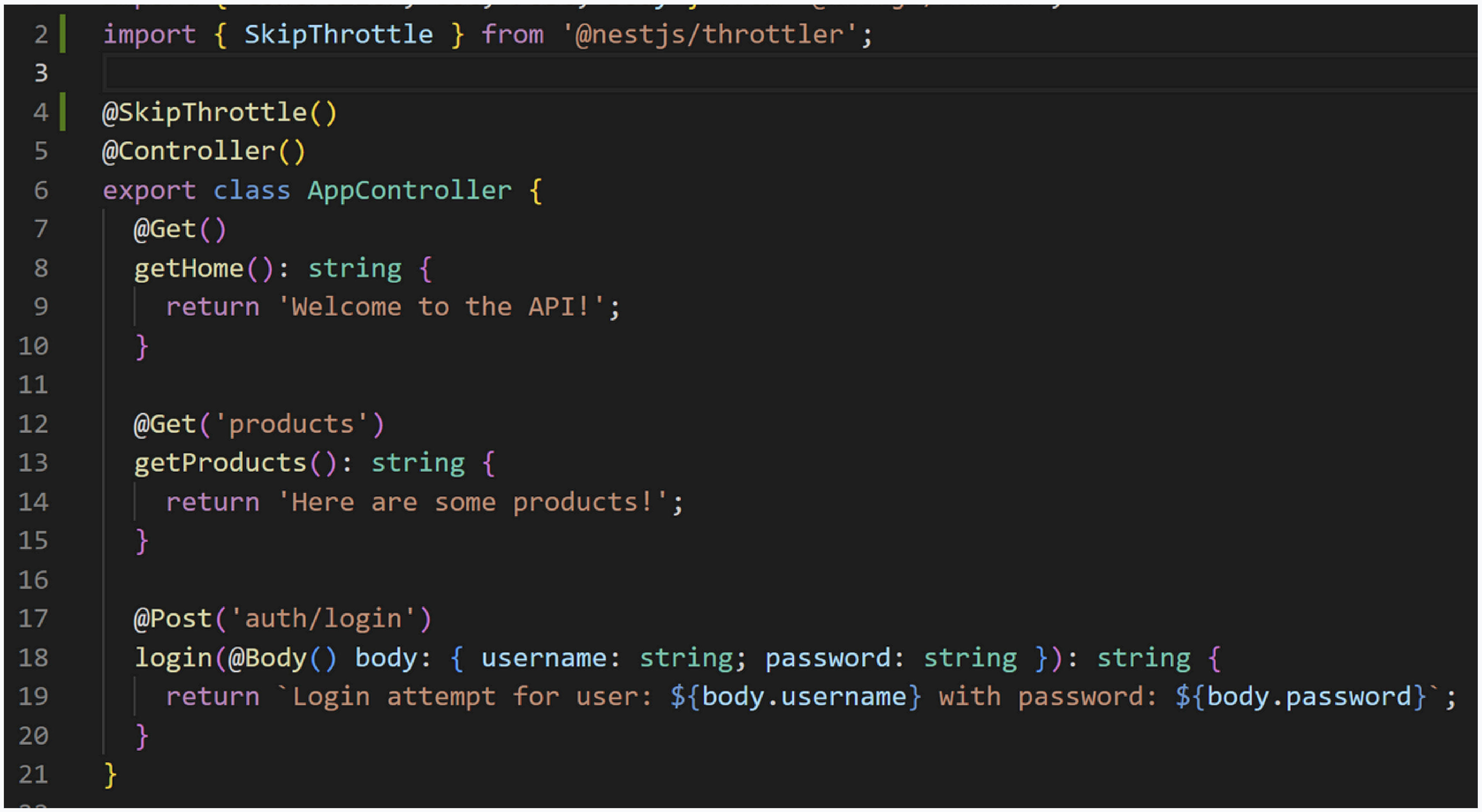
How it works:

Scope	Example Use Case
Method-level	Apply to a single endpoint only
Controller-level	All routes inside the controller are excluded

@SkipThrottle() Usage Examples

Skipping Throttling (Controller-Level)

Apply **@SkipThrottle()** to the whole controller



```
import { SkipThrottle } from '@nestjs/throttler';

@SkipThrottle()
@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}`;
  }
}
```

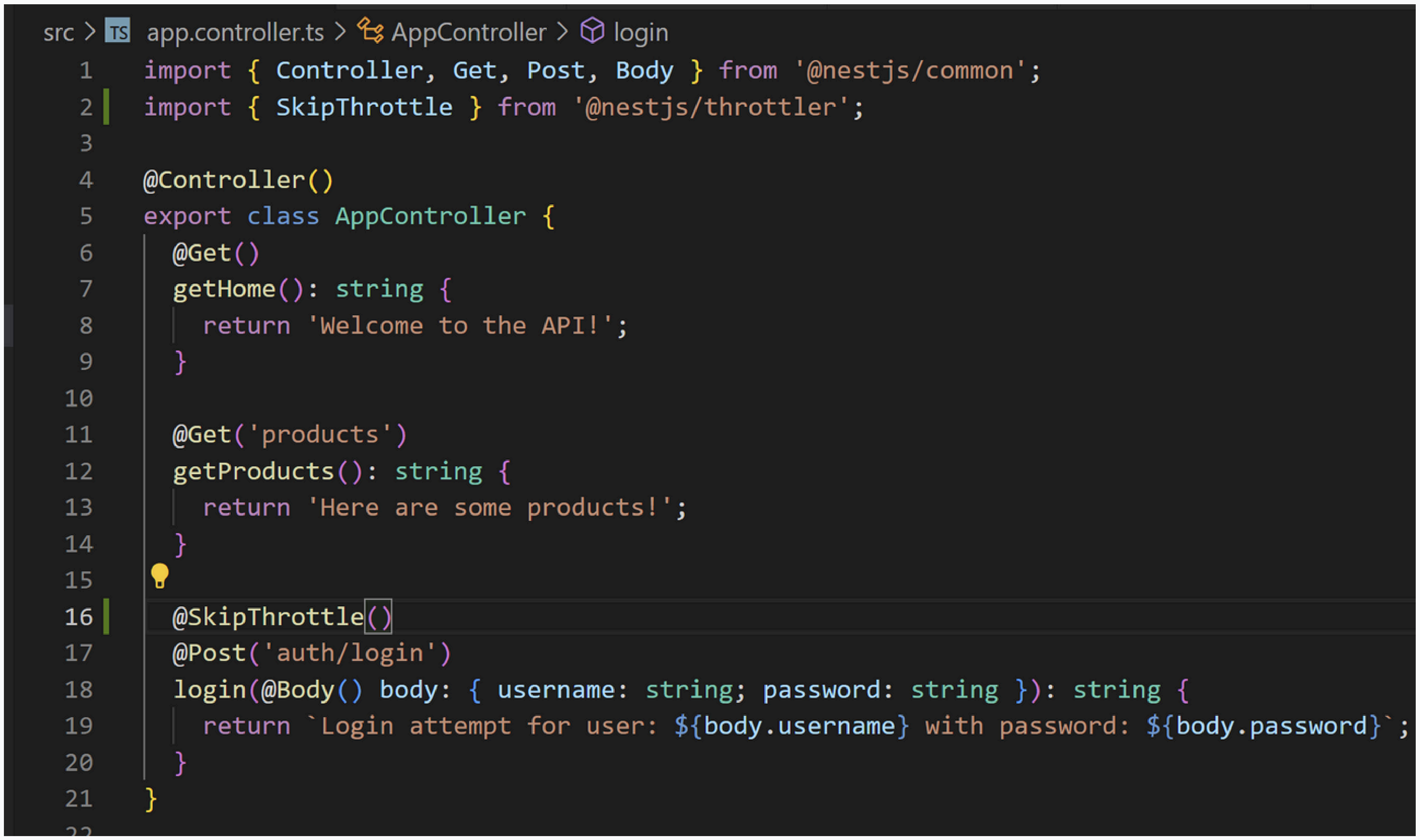
✅ Effect:

- No throttling on any route in this controller.
- Great for public/open-access APIs.

@SkipThrottle() Usage Examples

Skipping Throttling (Method-Level Only)

Apply **@SkipThrottle()** to a specific route



src > app.controller.ts > AppController > login

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

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

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

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

✅ Effect:

- Only the /auth/login route bypasses throttling.
- All other routes are still rate-limited globally.

@Throttle() Decorator

What is @Throttle()?

A decorator from **@nestjs/throttler** that overrides the default/global rate limits for specific routes or controllers.

What It Does:

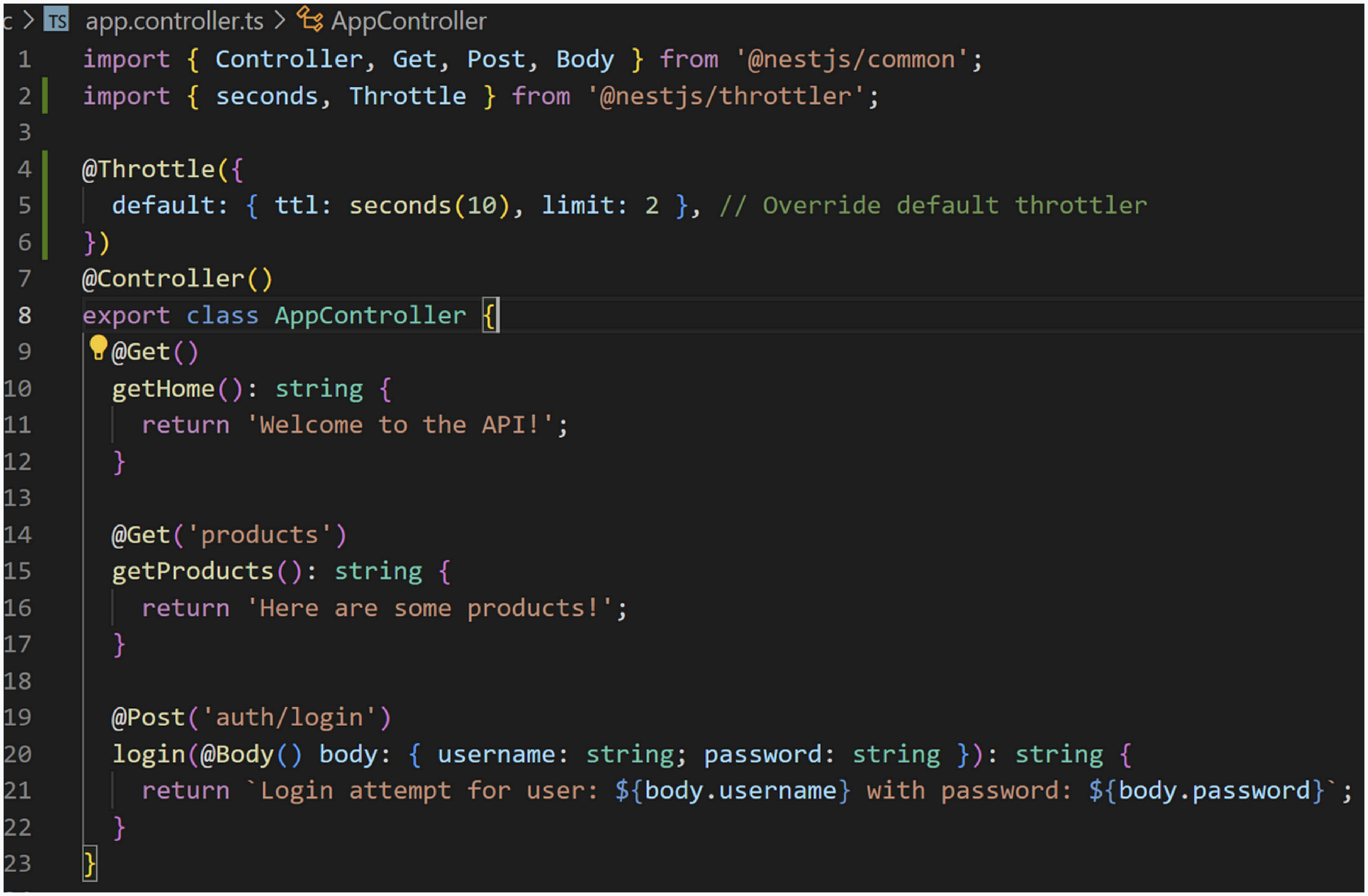
- Apply custom rate limits (requests and time) per:
  - Endpoint (method-level)
  - Whole controller (controller-level)
- Useful when:
  - Some routes need stricter or looser rate limits

Syntax:

```
@Throttle({
 default: { ttl: numberInSeconds, limit: number },
})
```

@Throttle() Usage Examples

Controller-Level @Throttle() Example



c > app.controller.ts > AppController

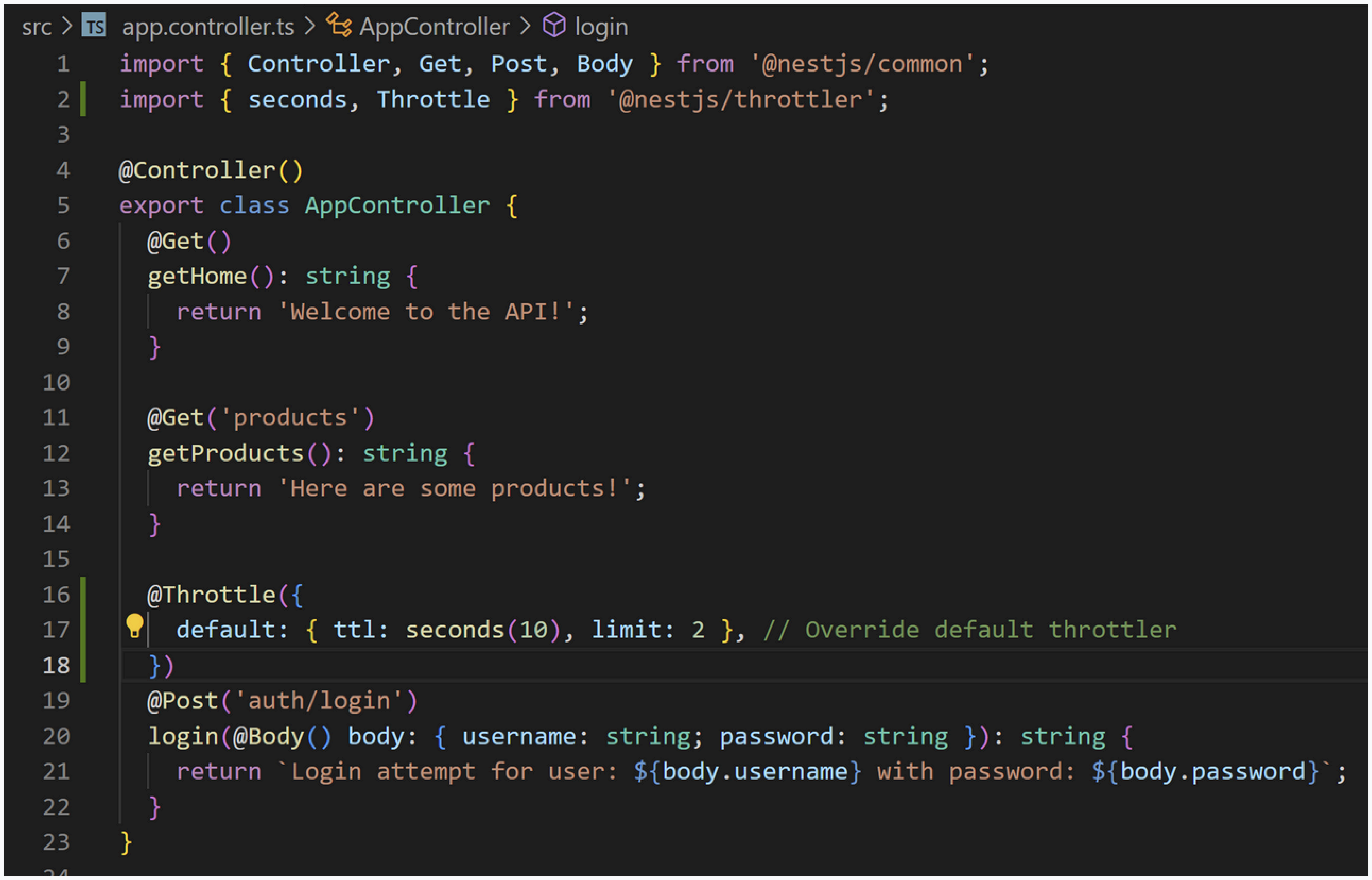
```typescript
import { Controller, Get, Post, Body } from '@nestjs/common';
import { seconds, Throttle } from '@nestjs/throttler';

@Throttle({
  default: { ttl: seconds(10), limit: 2 }, // Override default throttler
})
@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}`;
  }
}
```

✅ Effect:

- All routes in this controller limited to 2 requests every 10 seconds

@Throttle() Usage Examples

Method-Level @Throttle() Example



src > app.controller.ts > AppController > login

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

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

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

  @Throttle({
    default: { ttl: seconds(10), limit: 2 }, // Override default throttler
  })
  @Post('auth/login')
  login(@Body() body: { username: string; password: string }): string {
    return `Login attempt for user: ${body.username} with password: ${body.password}`;
  }
}
```

✅ Effect:

- Only /auth/login is limited to 2 request every 10 seconds
- Other routes follow the global limit

What’s Coming Next — Advanced Throttling in NestJS

In the upcoming advanced section of this series, we’ll take rate limiting to the next level using powerful features provided by @nestjs/throttler.

Here’s a preview of what we’ll be exploring:

✅ **Named Throttlers:** create multiple throttling strategies for different scenarios

✅ **Overriding Named Throttlers:** apply the right limit for the right route

✅ **Using @SkipThrottle() with Named Throttlers:** fine-grained control

✅ **Custom @Throttle() with Named Contexts:** smart, targeted protection

✅ **Storage Backends:**
→ In-Memory (easy to start)
→ Redis (best for scalability & real-world apps)

Ready to build truly robust API protection?

Stay tuned. We’re just getting started with NestJS throttling!