THIRD-PARTY LIBRARIES

By Sonam Soni(B.E.I.T.)
JAVA FSD/MERN/MEAN (Expert & Instructor)

ANGULAR MATERIAL

- Material Design components for Angular
- How to add:
- ng add @angular/material
- We can import inside component directly if component is standalone

```
@Component ({
  imports: [
    MatSlideToggleModule,
  ]
})
class AppComponent {}
```

<mat-slide-toggle>Toggle me!</mat-slide-toggle>

STABLE MDC-BASED COMPONENTS

- MDC-based (Material Design Components) components in Angular are a set of reusable UI components that implement the Material Design guidelines using the MDC Web library.
- These components offer enhanced accessibility, better performance, and a consistent design language.
- They are part of Angular Material's integration with MDC Web to align with Google's Material Design standards.

HOW TO SETUP?

- ng add @angular/material

- Import MDC-based Components
- We can follow the official documentation just like bootstrap

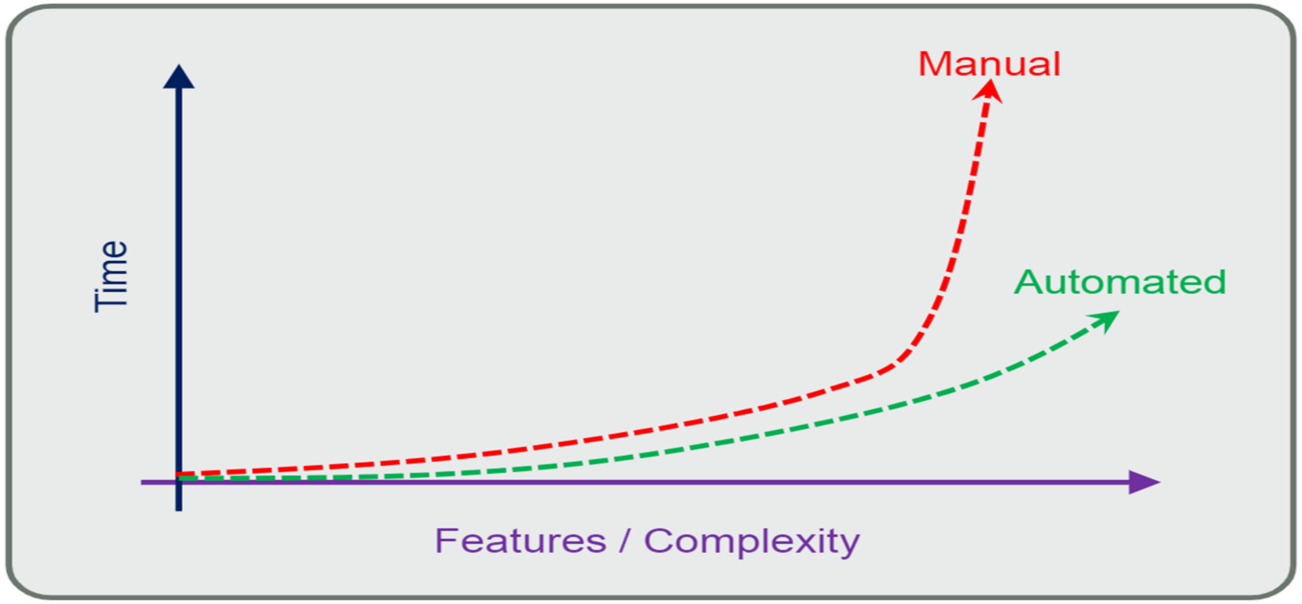
- https://material.angular.io/guide/getting-started

UNIT TESTING IN ANGULAR

- Introduction to Unit testing
- Test Component
- Test Services
- Coverage Report

AUTOMATED TESTING

- Automated testing is the practice of writing a code to test the code. It involves running tests in an automated fashion. It is especially helpful as manual testing is time consuming.
- Figure below compares the time taken for manual and automated testing.



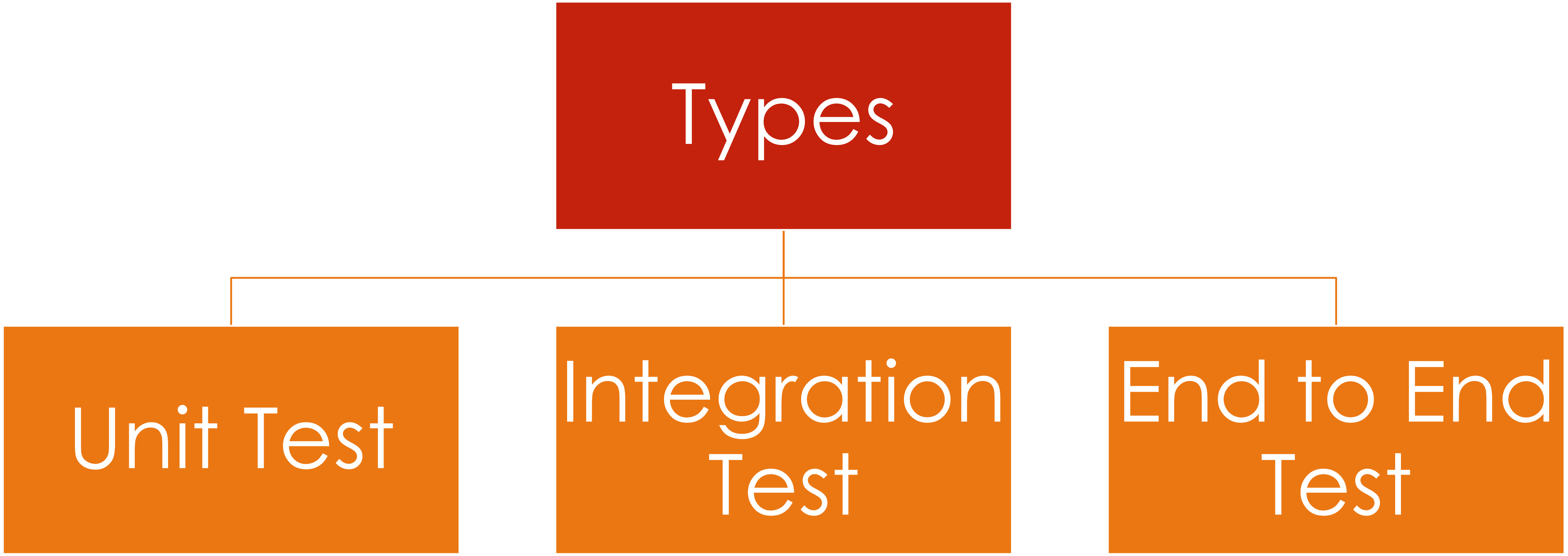
ADVANTAGES OF AUTOMATED TESTING

- Helps to catch defects before releasing the software
- Builds software of better quality
- Enforces to write better and more reliable code
- Reveals mistakes in design
- Helps in regression testing
- Acts as a documentation of app functionality
- Helps you to become a better developer

DISADVANTAGES OF AUTOMATED TESTING

- Automated testing may not be good for start-ups with limited time, limited budget, or uncertainties about product future.
- Automated testing is also not suitable when there are frequent changes in the requirements as it will also require changes in the test code.

TYPES OF AUTOMATED TESTING



<u>Unit Tests</u>:

- They test a component in isolation, without external resources like database, file, etc.
- They do not test the functionality of the app.
- They test the angular component code in isolation without the template.
- They are easy to write.
- They are fast.

<u>Integration Tests</u>:

- They test a component with external resources.
- They test the functionality of the app.
- They test the angular component code with the external template.

<u>End-To-End Tests</u>:

- They test the entire app as a whole.
- They test the app functionality.
- They are slow and fragile.

UNIT TESTING

- Unit tests promote clean coding practices.
- They have the same principles as the functional code.
- Unit tests are composed of small functions or methods in 10 lines or less.
- Unit tests require proper naming.
- Unit tests follow the single responsibility principle, that is, they test only one thing.

INTRODUCTION TO TESTING TOOLS

Jasmine:

- It is a behavior-driven development framework for testing the JavaScript code.
- It is dependency free and doesn't require a DOM.
- URL - https://jasmine.github.io/2.4/introduction.html

Karma:

- It is a test runner for writing and running unit tests while developing Angular apps.
- It increases developer productivity.
- URL - https://karma-runner.github.io/2.0/index.html

Protractor:

- It writes and runs end-to-end (e2e) tests.
- It explores the app as users experience it.
- URL - http://www.protractortest.org/#/

ANGULAR TESTING UTILITIES

- It creates a test environment for the application code under testing.
- It is used to test interactions.
- The test files should have .spec.ts extension.
- **Running tests using Angular CLI**
    - ng test
- **describe()** – This function defines a suite or a group of related tests.
    - For example: describe(‘suite-name’, function)
- **it()** – This function defines a spec or test.
    - For example: it(‘spec-name’, function)

ANGULAR TESTING UTILITIES

- **expect()**
  - It is used in Jasmine API.
  - It takes the actual value as the parameter.
  - It is chained with a Matcher function.
- **Matcher function**
  - It takes the expected value as the parameter.
  - It is responsible for reporting to Jasmine if the expectation is true or false.

```
expect(result).toBe('value')
expect(result).toContain('value')
expect(result).toEqual(12)
expect(result).toBeNull()
expect(result).toBeTruthy()
```

SET UP AND TEAR DOWN

The following methods are used to set up and remove unit tests:

- **Arrange:** It initializes the system under test.
- **Act:** It calls a method or a function.
- **Assert:** It asserts.
- **beforeEach(function) :** This function runs some shared setup before each of the specs in the enclosing describe() are executed.
- **afterEach(function**): This function runs some shared teardown after each of the specs in the enclosing describe() are executed.
- **beforeAll(function):** This function runs some shared setup once before all of the specs in the enclosing describe() are executed.
- **afterAll(function):** This function runs some shared teardown once after all of the specs in the enclosing describe() are executed.

CODE COVERAGE

- If you want to create code-coverage reports every time you run the test suite, you can run the below command in the project root directory.

```
ng test --code-coverage
```

- Whenever a test suite is executed successfully, you will see a “coverage” folder in the project root directory.
- Open the index.html file in your favorite browser to view the test case coverage for all the files in the project.

INTRODUCTION TO TESTBED

- It is the first and most important testing utility.
- It creates an Angular testing module.
- It is used to test interactions between:
    - a component and its template
    - different components
- Package - “@angular/core/testing”

- **TestBed.configureTestingModule(metadataObject)**
    - It is used to create a dynamic Angular testing module.
    - It takes an @NgModule-like metadata object.
    - “metadataObject” can have most of the properties of the NgModule.

WORKING WITH THE COMPONENTS

- **TestBed.createComponent(component)**
  - Used to create an instance of component-under-test
  - Returns the component test fixture
- **ComponentFixture**
  - Wrapper around a component
  - Gives access to component instance as well as its template (DOM representation)
- **ComponentFixture.componentInstance**
  - Returns instance of the component class
- **ComponentFixture.nativeElement**
  - Returns the native DOM element at the root of the component
- **ComponentFixture.debugElement**
  - Provides a wrapper object around the component's root native element
  - Provides useful methods for querying the DOM

TESTING PROPERTY AND EVENT BINDINGS

DebugElement.query(predicate)
- Used to query the DOM
- Predicate is a function that returns true if a condition is met.
- Returns the first element that matches the predicate

By.css()
- Predicate for use with DebugElement's query functions
- Matches elements by the given CSS selector

ComponentFixture.detectChanges()
- Triggers a change detection cycle for the component

DebugElement.triggerEventHandler(‘eventName’, eventObj)
- Used to trigger an event on an element. For e.g., to invoke “click” event on a button with id “save,” the following code is used:
- const button = fixture.debugElement.query(By.css(‘#save’));
- button.triggerEventHandler(‘click’, null);

HANDLING COMPONENT DEPENDENCIES

Providing Dependencies

- Register the service in the testing module by adding it to the “providers” array
- Register any other Angular dependency in the testing module by adding it to “imports” array
- For example: if the service internally uses HTTP, add HTTPModule to “imports” array of testing module configuration

Getting Dependencies

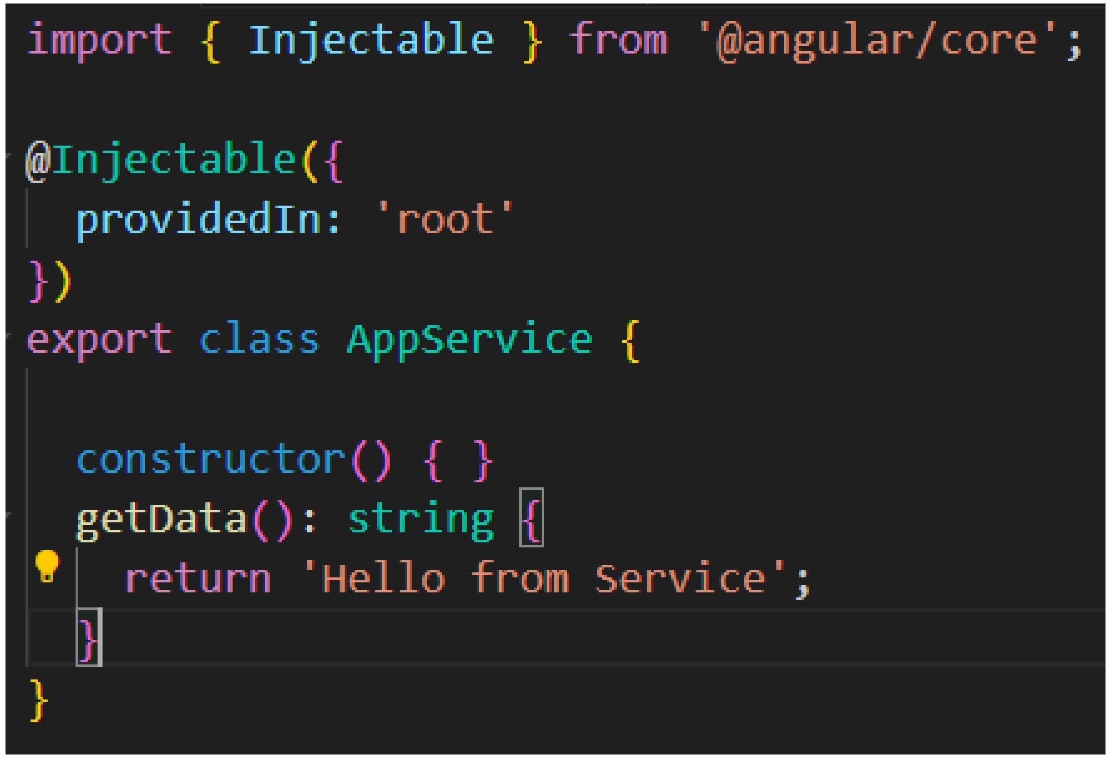
- TestBed.get(service)
- Returns a reference to ‘service’ instance injected in a component

- Providing Stubs
- Identify the dependencies and their methods that are used within the component
- Create a stub class for each of the dependencies
- Define method stubs
- Replace the actual dependency with their corresponding stub implementation within the ‘providers’ array

```
TestBed.configureTestingModule({  declarations: [ UserDetailsComponent ],
providers: [
     { provide: Router, useClass: RouterStub },
     { provide: ActivatedRoute, useClass: ActivatedRouteStub }
 ]
})
```

WRITING TEST FOR SERVICE

- Create one service: ng g service app

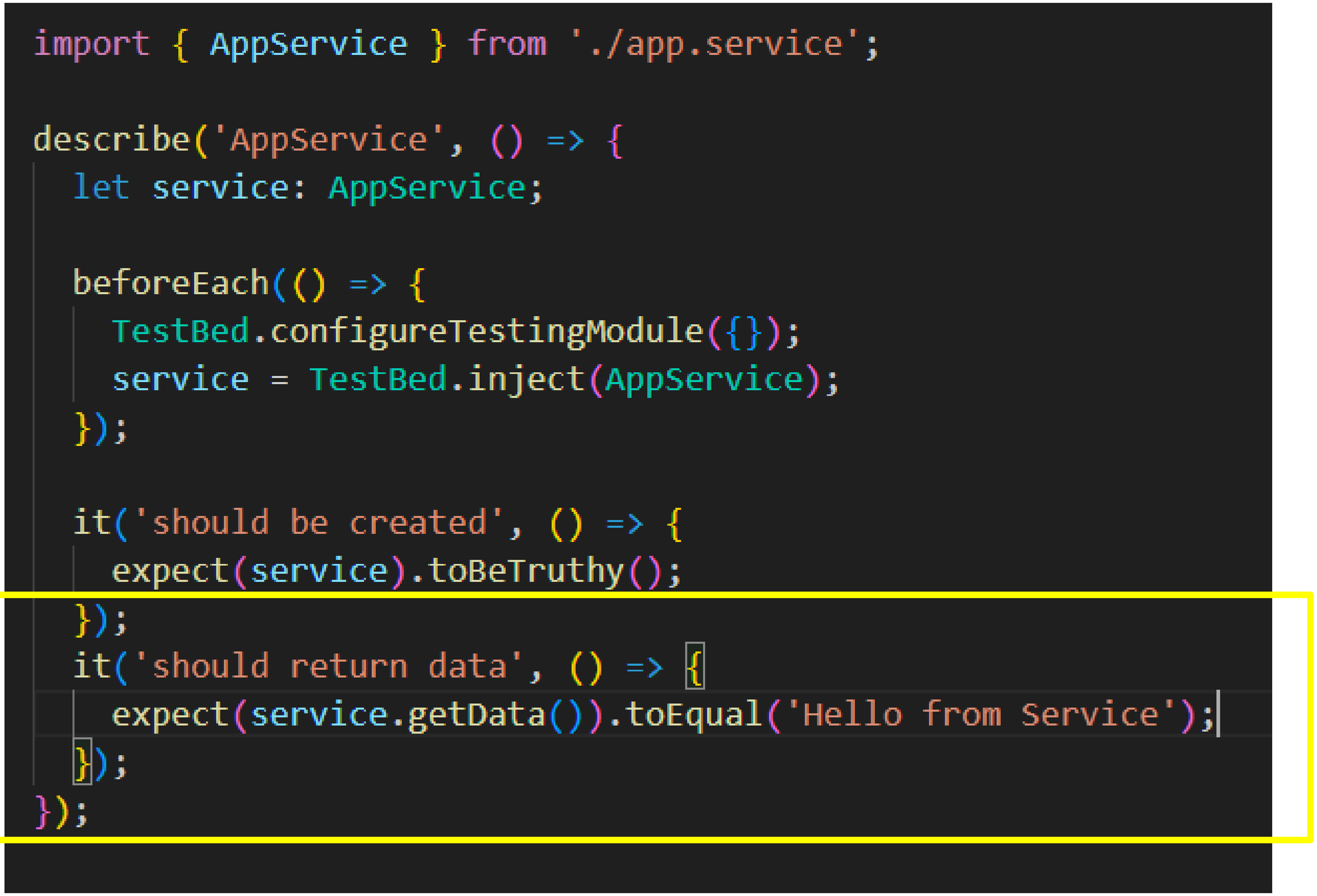


```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class AppService {

  constructor() { }
  getData(): string {
    return 'Hello from Service';
  }
}
```

TEST CASE FOR SERVICE



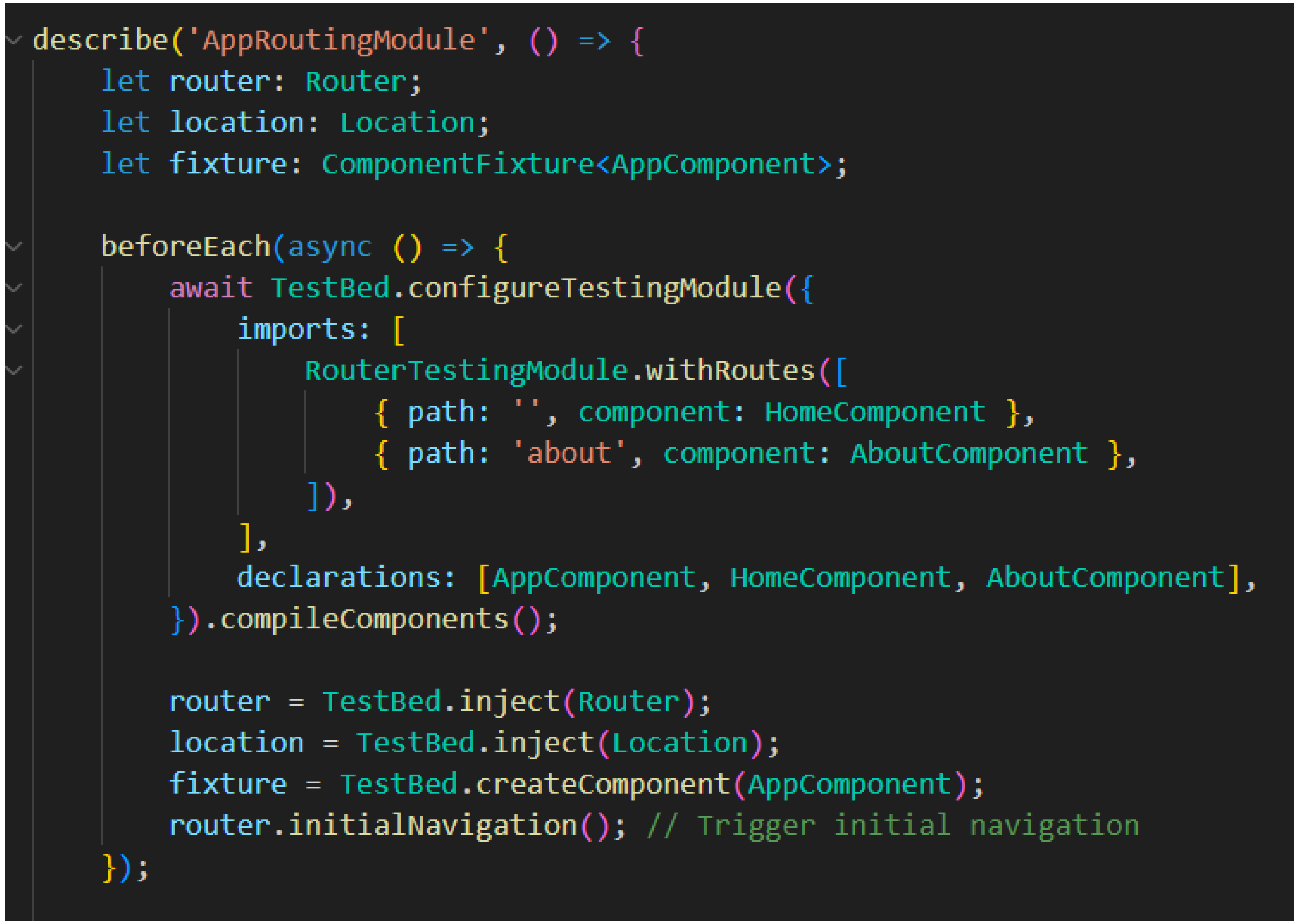
```
import { AppService } from './app.service';

describe('AppService', () => {
  let service: AppService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(AppService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
  it('should return data', () => {
    expect(service.getData()).toEqual('Hello from Service');
  });
});
```

TESTING NAVIGATION

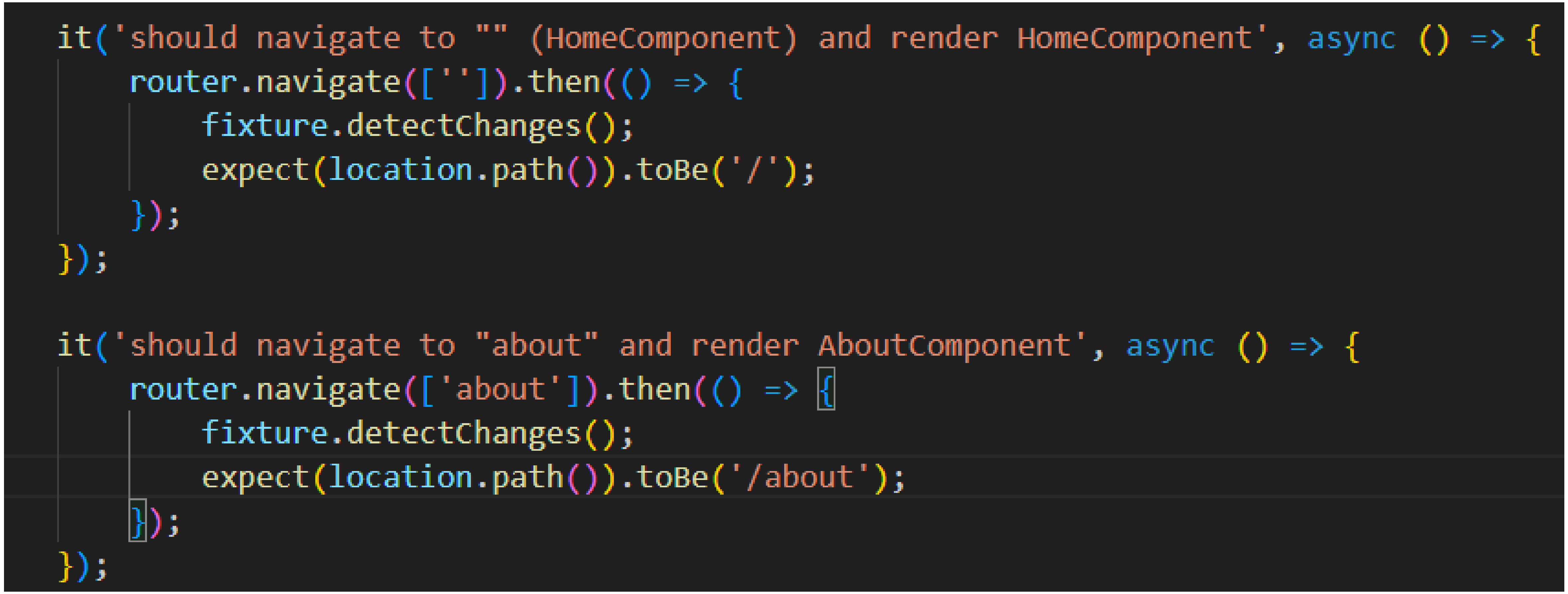


```
describe('AppRoutingModule', () => {
    let router: Router;
    let location: Location;
    let fixture: ComponentFixture<AppComponent>;

    beforeEach(async () => {
        await TestBed.configureTestingModule({
            imports: [
                RouterTestingModule.withRoutes([
                    { path: '', component: HomeComponent },
                    { path: 'about', component: AboutComponent },
                ]),
            ],
            declarations: [AppComponent, HomeComponent, AboutComponent],
        }).compileComponents();

        router = TestBed.inject(Router);
        location = TestBed.inject(Location);
        fixture = TestBed.createComponent(AppComponent);
        router.initialNavigation(); // Trigger initial navigation
    });
```

TEST CASE



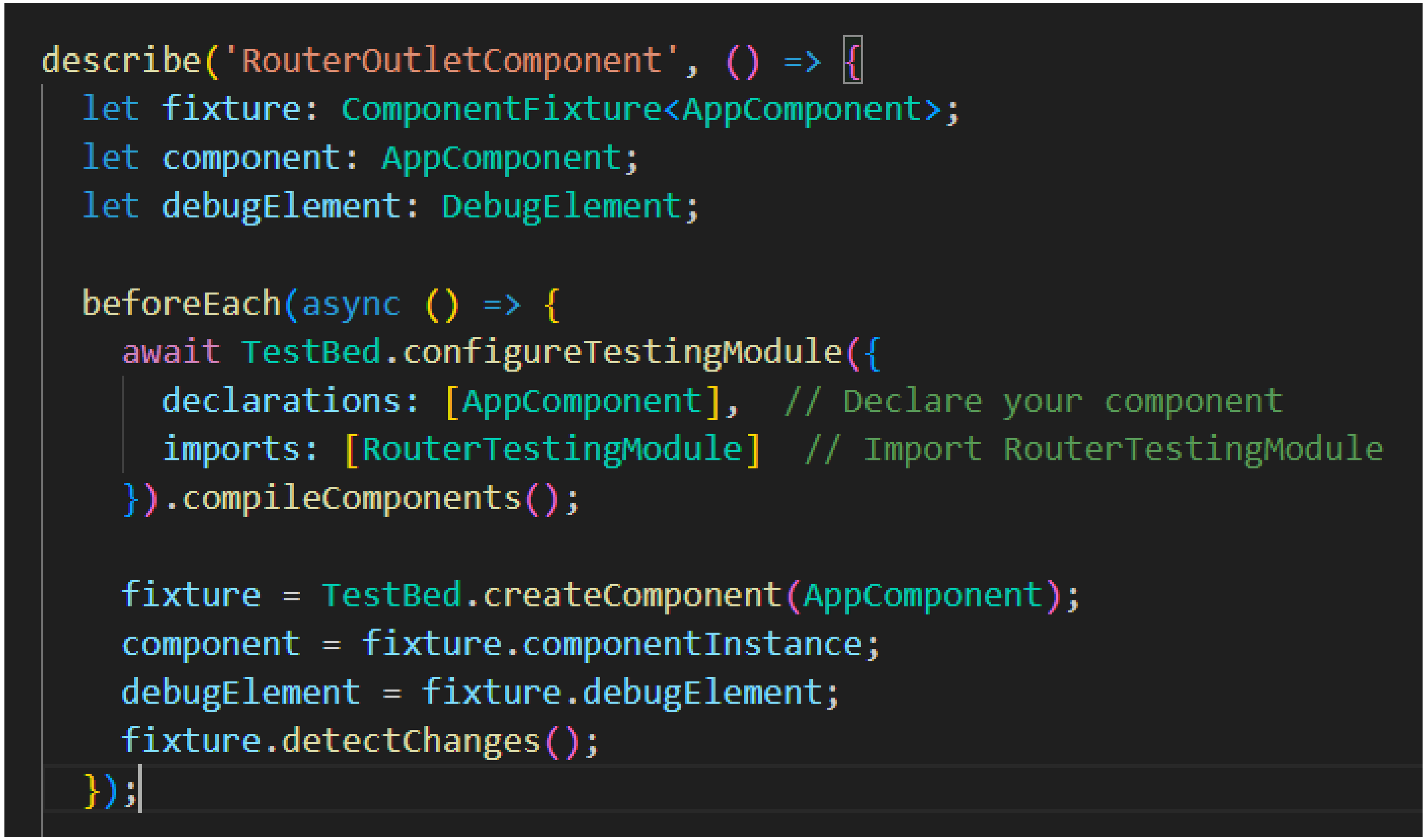
```
it('should navigate to "" (HomeComponent) and render HomeComponent', async () => {
    router.navigate(['']).then(() => {
        fixture.detectChanges();
        expect(location.path()).toBe('/');
    });
});

it('should navigate to "about" and render AboutComponent', async () => {
    router.navigate(['about']).then(() => {
        fixture.detectChanges();
        expect(location.path()).toBe('/about');
    });
});
```

TESTING ROUTEROUTLET COMPONENTS

Testing RouterOutlet components

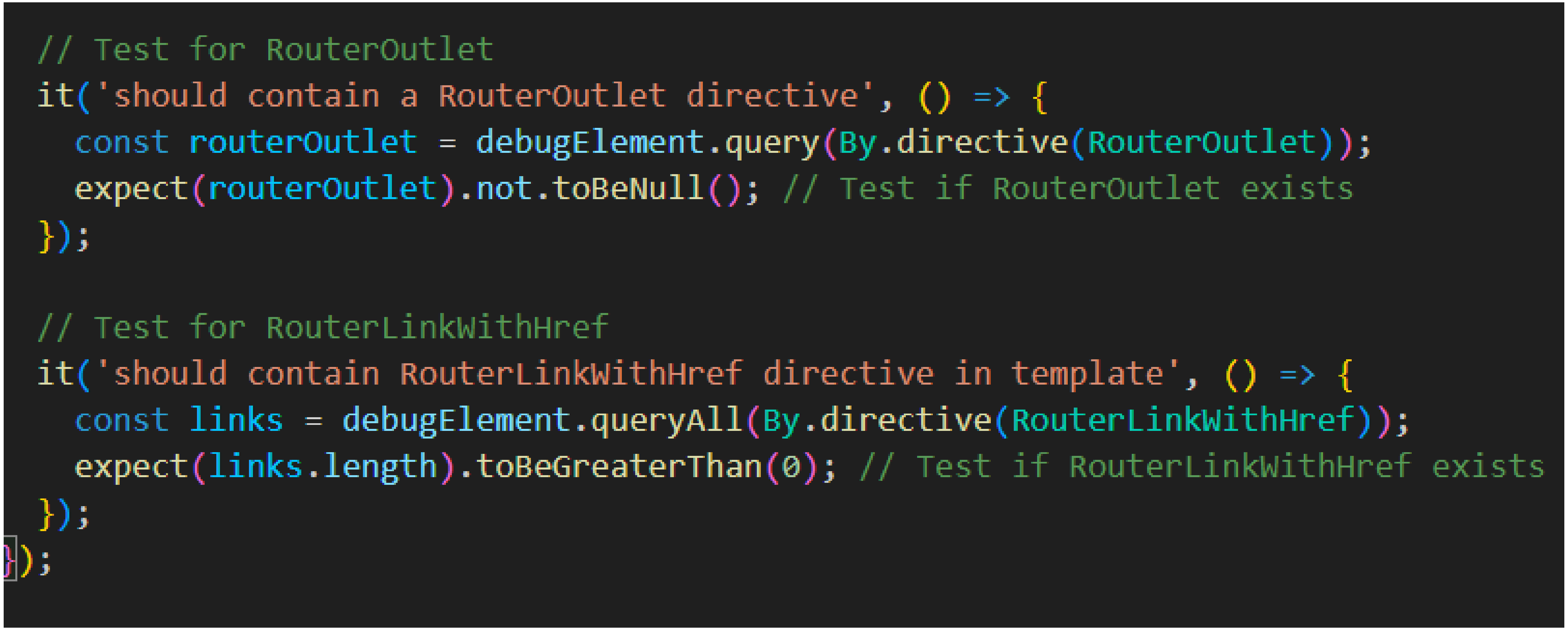
- Verify if the component template contains ‘RouterOutlet’ directive.
- Verify if the component template contains ‘RouterLinkWithHref’ directive.
- Add ‘RouterTestingModule’ to ‘imports’ array of the testing module.



```
describe('RouterOutletComponent', () => {
  let fixture: ComponentFixture<AppComponent>;
  let component: AppComponent;
  let debugElement: DebugElement;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [AppComponent],  // Declare your component
      imports: [RouterTestingModule]  // Import RouterTestingModule
    }).compileComponents();

    fixture = TestBed.createComponent(AppComponent);
    component = fixture.componentInstance;
    debugElement = fixture.debugElement;
    fixture.detectChanges();
  });
```



```
// Test for RouterOutlet
it('should contain a RouterOutlet directive', () => {
  const routerOutlet = debugElement.query(By.directive(RouterOutlet));
  expect(routerOutlet).not.toBeNull(); // Test if RouterOutlet exists
});

// Test for RouterLinkWithHref
it('should contain RouterLinkWithHref directive in template', () => {
  const links = debugElement.queryAll(By.directive(RouterLinkWithHref));
  expect(links.length).toBeGreaterThan(0); // Test if RouterLinkWithHref exists
});
});
```

TESTING ASYNC OPERATIONS

async()
- Runs the body of a test(it) or setup(beforeEach) function within a special async test zone

ComponentFixture.whenStable()
- Returns a promise that resolves when the fixture is stable
- To resume testing after completion of asynchronous activity or asynchronous change detection, hook that promise

fakeAsync()
- Runs the body of a test (it) within a special fakeAsync test zone, enabling a linear control flow coding style

tick()
- Simulates the passage of time and the completion of pending asynchronous activities

MODULE 10: NEW FEATURE

- Extended esbuild support
- Cleaner, Better Stack Traces
- Automatic Imports in Language Service
- Optimized CLI with a Bunch of Improvements

ESBUILD

- Angular 15 introduced extended support for **esbuild** to improve the speed and efficiency of both development and production builds
- Esbuild is a modern JavaScript bundler that focuses on performance, significantly reducing build times compared to traditional tools like Webpack.
- **esbuild** is used for bundling, minification, and tree shaking in Angular 15.
- Faster rebuilds during development, especially for large projects.
- Faster production builds with efficient code minification.
- esbuild handles both JavaScript and TypeScript natively, making the build process more streamlined.
- TypeScript compilation is faster due to optimizations in esbuild.

ESBUILD

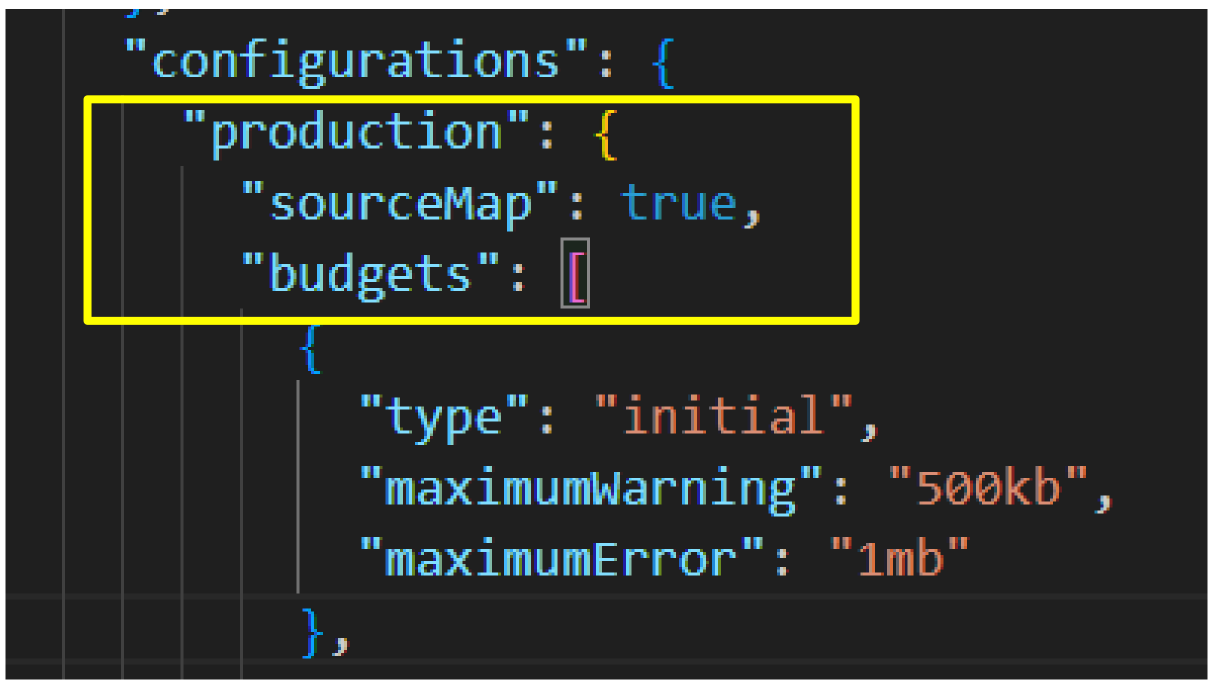
- Faster source map generation, allowing better debugging experiences in development environments.
- Minification is performed by esbuild, which is significantly faster than previous solutions.
- Efficient tree shaking with esbuild to reduce bundle size by removing unused code.
- esbuild support is integrated directly into Angular CLI, so developers don’t need to configure it manually.

LET'S USE

- If you're using Angular 15, esbuild is automatically used for development builds.
- Just run: ng serve
- By default, Angular 15 also uses esbuild for production builds.
- Command for production build:
- ng build --configuration=production

CUSTOMIZATION

- Although Angular 15 uses esbuild automatically, you can modify the angular.json file for certain settings.

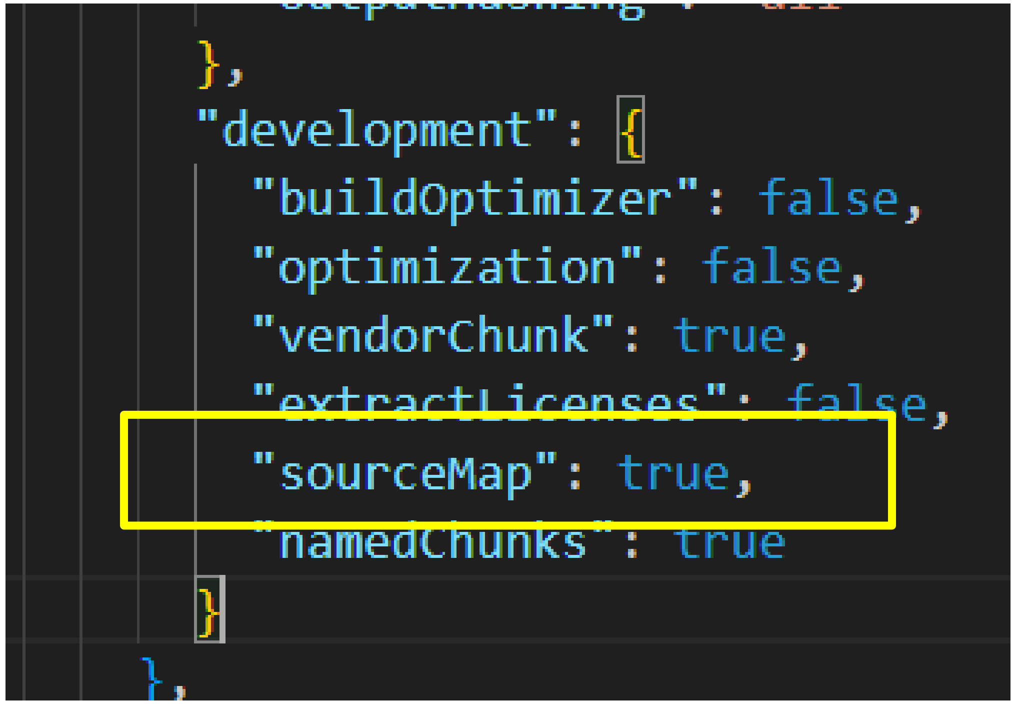


```
  "configurations": {
    "production": {
      "sourceMap": true,
      "budgets": [
        {
          "type": "initial",
          "maximumWarning": "500kb",
          "maximumError": "1mb"
        },
```

CLEANER, BETTER STACK TRACES

- In Angular, improvements were made to provide **cleaner, better stack traces** to enhance the debugging experience for developers.
- The goal is to make stack traces more readable, by removing unnecessary noise and focusing on the core application code.
- We can improve it using
  - **Enhanced Error Messages**
  - **Readable Stack Traces**
  - **Source Map Integration**
  - **Optional Verbose Error Mode**

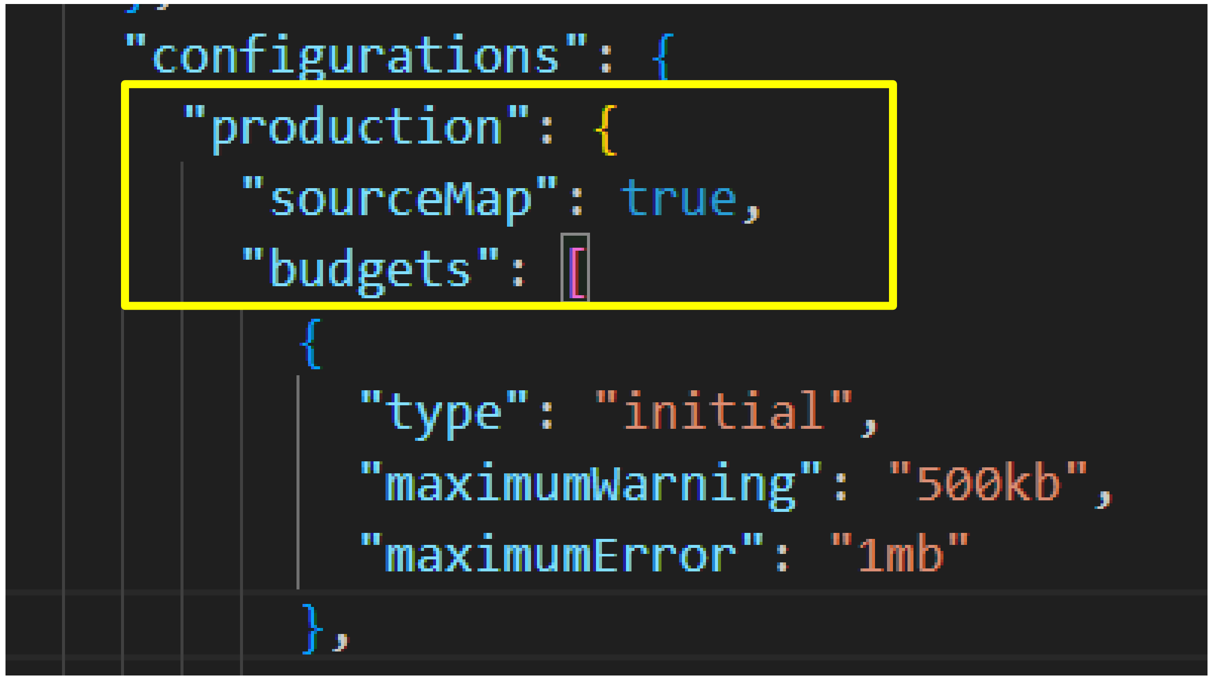
SOURCEMAP



```
    },
    "development": {
      "buildOptimizer": false,
      "optimization": false,
      "vendorChunk": true,
      "extractLicenses": false,
      "sourceMap": true,
      "namedChunks": true
    }
  },
```

- Using Source Maps in Development Mode
- With source maps enabled, stack traces will point to your original TypeScript files instead of compiled JavaScript.

PRODUCTION MODE



```
"configurations": {
  "production": {
    "sourceMap": true,
    "budgets": [
      {
        "type": "initial",
        "maximumWarning": "500kb",
        "maximumError": "1mb"
      },
```

- Even in production builds, you can enable source maps to improve debugging of issues.
- Keep in mind that this could expose some parts of your source code to the public if your application is deployed to production with source maps enabled.

AUTOMATIC IMPORTS IN LANGUAGE SERVICE

- **automatic imports** were introduced in the **Angular Language Service** enhances the developer experience by automatically suggesting and inserting import statements as you code.
- This feature aims to improve productivity by reducing manual effort in handling imports, allowing developers to focus on writing code.
  - Automatic Import Suggestions
  - Autocomplete Support
  - Faster Development

HOW TO ENABLE

- Most popular IDEs like **VS Code** and **WebStorm** already have the Angular Language Service extension available.
- In **VS Code**, install the **Angular Language Service** extension from the marketplace.
- Ensure your IDE's TypeScript version is up to date, as the Angular Language Service relies on it for automatic imports.

OPTIMIZED CLI WITH A BUNCH OF IMPROVEMENTS

- Angular CLI received several optimizations and improvements aimed at enhancing the overall developer experience, improving build performance, and simplifying project configuration.
- These changes, coupled with the integration of esbuild and other tooling improvements, help developers create Angular applications more efficiently.
- **esbuild** is now used for both development and production builds, significantly speeding up build times.
- **Tree-shaking**, **bundling**, **minification**, and **source map generation** are all faster due to esbuild's integration.

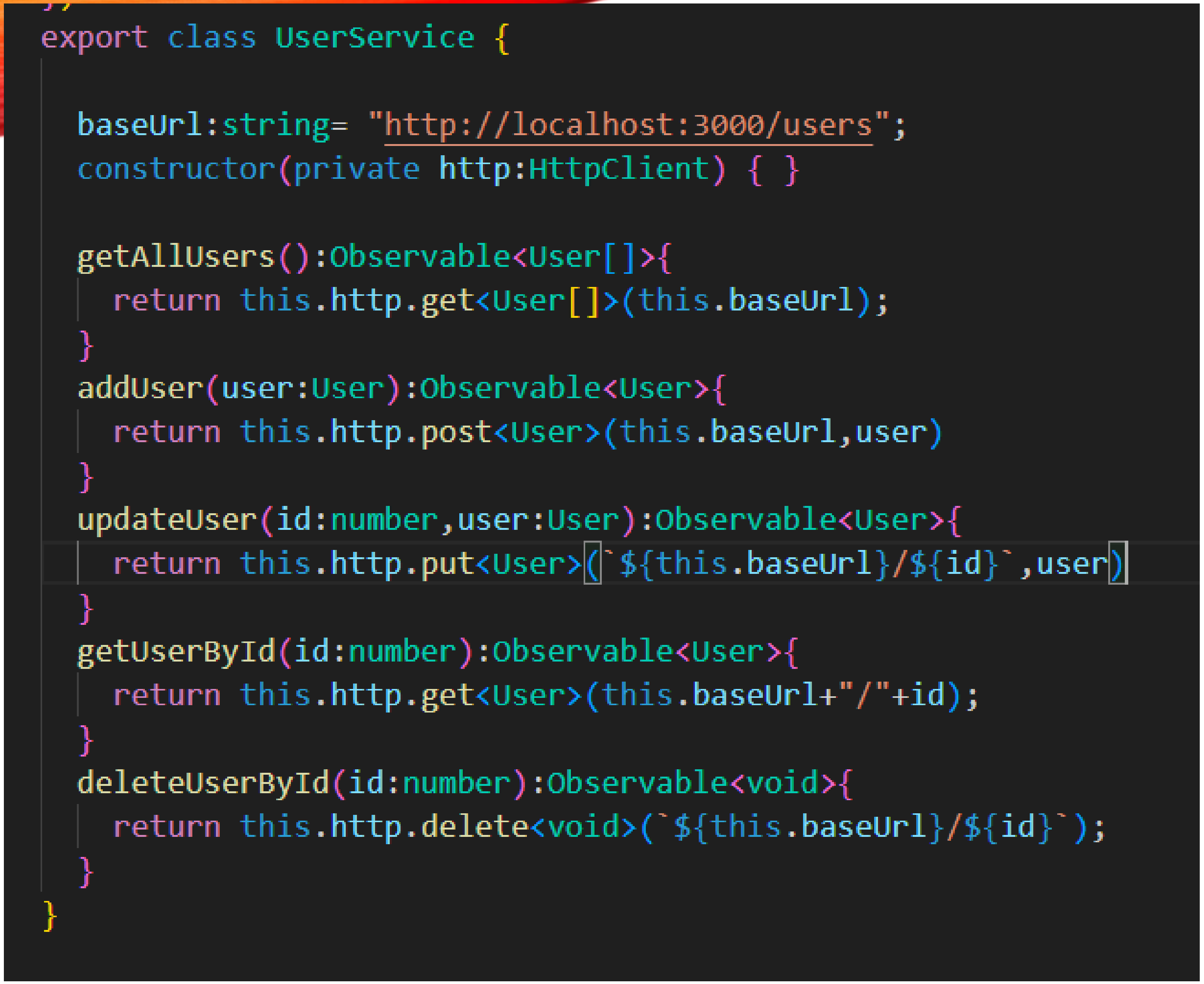
- In production builds, the CLI leverages **esbuild** for minification, which is faster and produces smaller bundles compared to older tools like Terser.
- The CLI can also remove unused code from Angular modules, components, and services for even better optimization.
- As you write code, the CLI, combined with your IDE, suggests imports for missing symbols and automatically adds them to your files.

VITE SUPPORT

- Angular 15 introduced **experimental support for Vite**, a faster development server.Vite offers hot module replacement (HMR) and faster server start times,
- making the development experience smoother, especially for larger projects.
- ng add @angular/vite

- The CLI now supports **faster unit tests** and **end-to-end (e2e) tests**.
- It encourages the use of **standalone components**, which allow for a more modular and lightweight setup.
- ng generate component my-component --standalone
- The CLI's development server now has better **Hot Module Replacement (HMR)**, allowing developers to update modules without a full page reload.
- To enable HMR: ng serve --hmr
- **Optimize Testing: ng test (command)**

PROJECT



```
export class UserService {

  baseUrl:string= "http://localhost:3000/users";
  constructor(private http:HttpClient) { }

  getAllUsers():Observable<User[]>{
    return this.http.get<User[]>(this.baseUrl);
  }
  addUser(user:User):Observable<User>{
    return this.http.post<User>(this.baseUrl,user)
  }
  updateUser(id:number,user:User):Observable<User>{
    return this.http.put<User>(`${this.baseUrl}/${id}`,user)
  }
  getUserById(id:number):Observable<User>{
    return this.http.get<User>(this.baseUrl+"/"+id);
  }
  deleteUserById(id:number):Observable<void>{
    return this.http.delete<void>(`${this.baseUrl}/${id}`);
  }
}
```