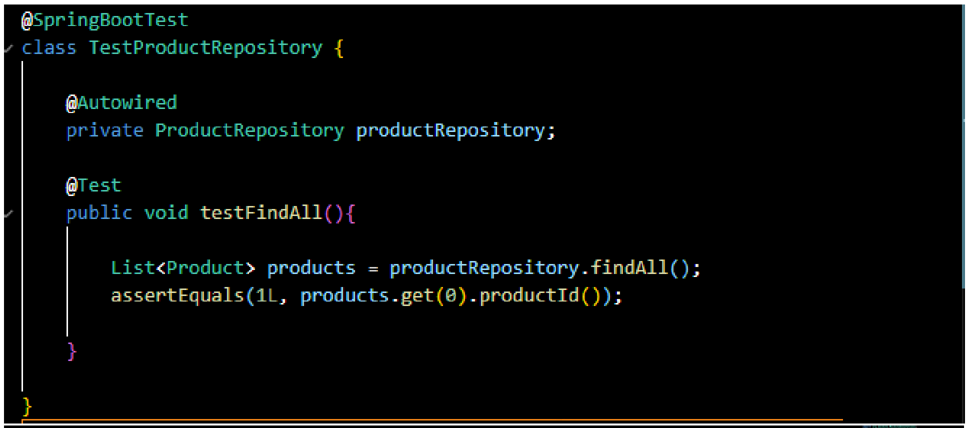
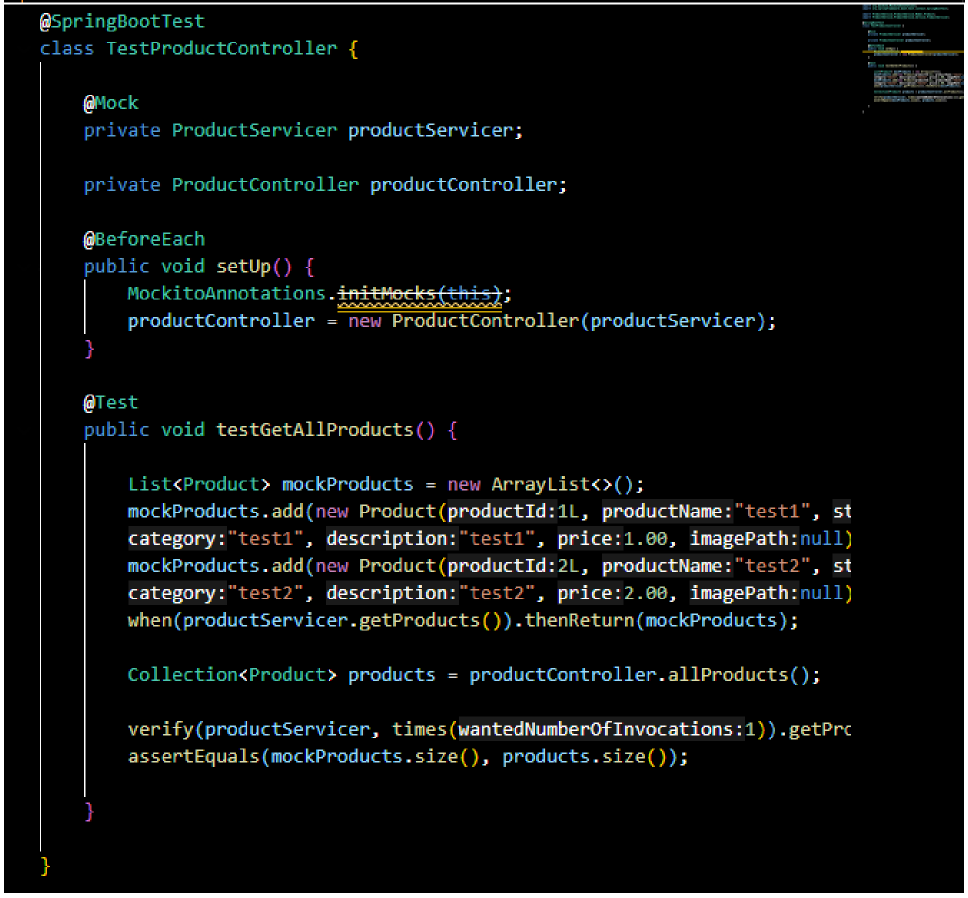
Product Service Unit Tests:

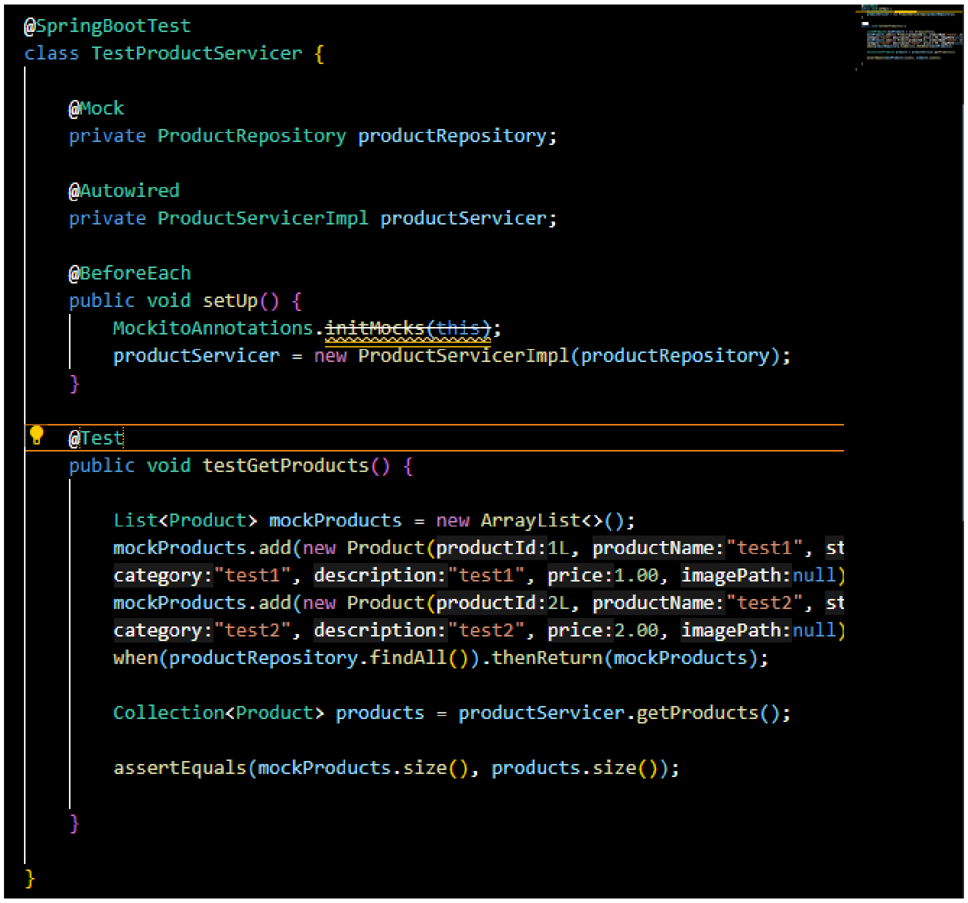
Screenshots:



```
@SpringBootTest
class TestProductRepository {

    @Autowired
    private ProductRepository productRepository;

    @Test
    public void testFindAll(){

        List<Product> products = productRepository.findAll();
        assertEquals(1L, products.get(0).productId());

    }

}
```



```
@SpringBootTest
class TestProductController {

    @Mock
    private ProductServicer productServicer;

    private ProductController productController;

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.~~initMocks(this)~~;
        productController = new ProductController(productServicer);
    }

    @Test
    public void testGetAllProducts() {

        List<Product> mockProducts = new ArrayList<>();
        mockProducts.add(new Product(productId:1L, productName:"test1", st
        category:"test1", description:"test1", price:1.00, imagePath:null)
        mockProducts.add(new Product(productId:2L, productName:"test2", st
        category:"test2", description:"test2", price:2.00, imagePath:null)
        when(productServicer.getProducts()).thenReturn(mockProducts);

        Collection<Product> products = productController.allProducts();

        verify(productServicer, times(wantedNumberOfInvocations:1)).getPrc
        assertEquals(mockProducts.size(), products.size());

    }

}
```



```
@SpringBootTest
class TestProductServicer {

    @Mock
    private ProductRepository productRepository;

    @Autowired
    private ProductServicerImpl productServicer;

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.~~initMocks(this)~~;
        productServicer = new ProductServicerImpl(productRepository);
    }

    @Test
    public void testGetProducts() {

        List<Product> mockProducts = new ArrayList<>();
        mockProducts.add(new Product(productId:1L, productName:"test1", st
        category:"test1", description:"test1", price:1.00, imagePath:null)
        mockProducts.add(new Product(productId:2L, productName:"test2", st
        category:"test2", description:"test2", price:2.00, imagePath:null)
        when(productRepository.findAll()).thenReturn(mockProducts);

        Collection<Product> products = productServicer.getProducts();

        assertEquals(mockProducts.size(), products.size());

    }

}
```

Unit Test 1 (TestProductRepository):

- This test class is part of the ProductService's repository layer.
- It is annotated with @SpringBootTest, which means it's also an integration test.
- It autowires an instance of ProductRepository.
- In the testFindAll method, it calls productRepository.findAll() to retrieve a list of products and then uses assertEquals to check if the ID of the first product in the list is equal to 1.

Unit Test 2 (TestProductServicer):

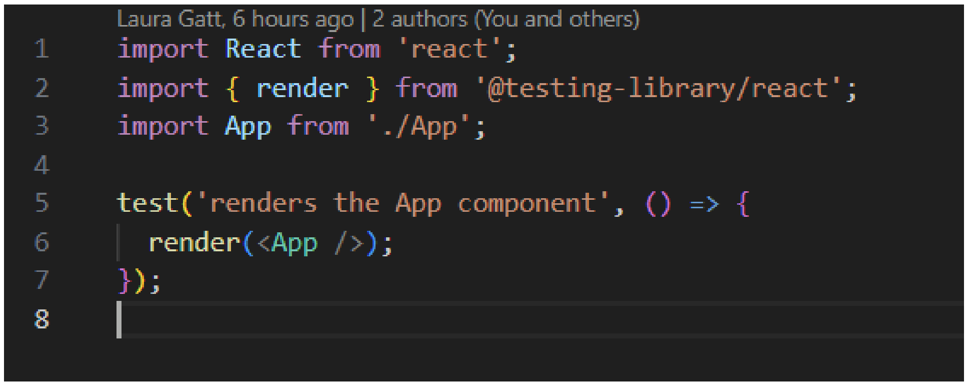
- This test class is part of the ProductService's service layer.
- It is annotated with @SpringBootTest, which means it's an integration test, and it will start the Spring application context.
- It uses Mockito to mock the ProductRepository and autowires an instance of ProductServicerImpl.
- In the testGetProducts method, it creates a list of mock Product objects and uses Mockito to mock the behavior of productRepository.findAll() to return this list.

- It then calls productServicer.getProducts() and compares the size of the returned products with the size of the mock products list using assertEquals.

Unit Test 3 (TestProductController):

- This test class is part of the ProductService's controller layer.
- It uses Mockito to mock the ProductServicer and creates an instance of ProductController with the mocked ProductServicer.
- In the testGetAllProducts method, it creates a list of mock Product objects and uses Mockito to mock the behavior of productServicer.getProducts() to return this list.
- It then calls productController.allProducts() and verifies that the productServicer.getProducts() method was called once with verify(productServicer, times(1)).getProducts(). Finally, it checks if the size of the returned products matches the size of the mock products list using assertEquals.

Frontend Test:


Laura Gatt, 6 hours ago | 2 authors (You and others)

```
import React from 'react';
import { render } from '@testing-library/react';
import App from './App';

test('renders the App component', () => {
  render(<App />);
});

```

Unit Test 1 (App.js Rendering Correctly):

- The test ensures that App.js renders correctly without any errors.