Mockito

cuong@techmaster.vn

Câu chuyện thực tế

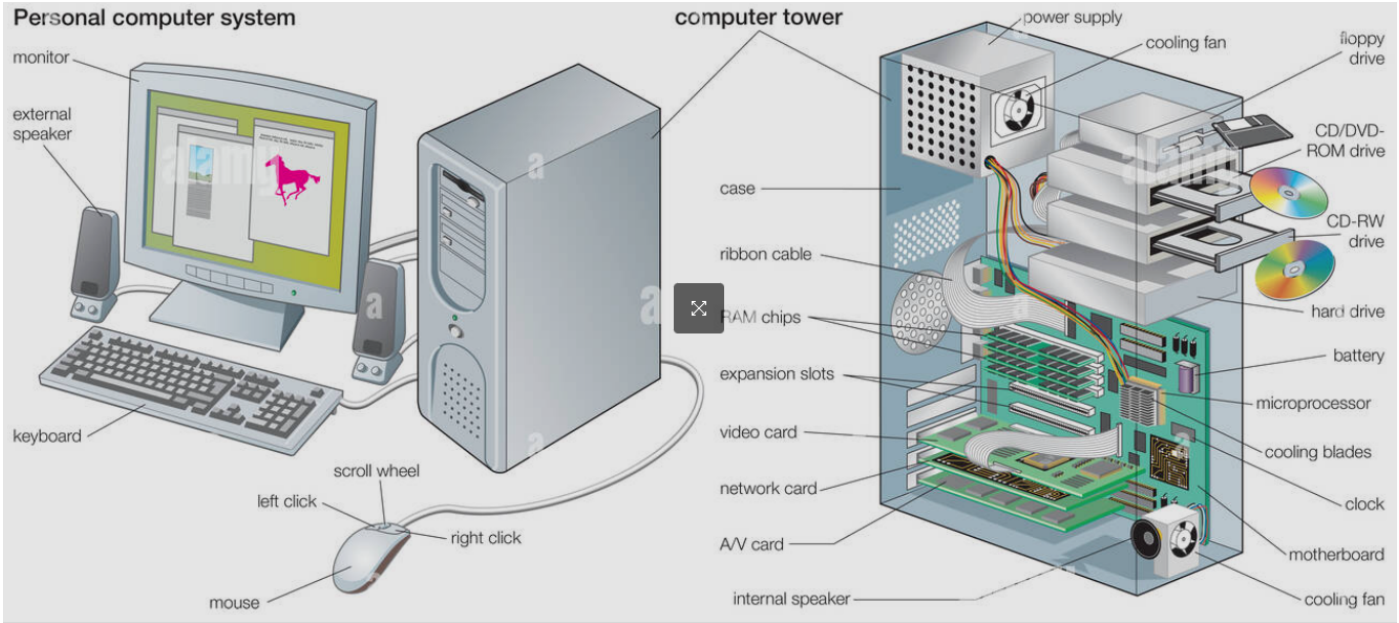
Tôi có một cái máy tính hỏng, bật chỉ nghe tiếng bíp bíp nhưng không boot vào hệ điều hành. Vậy nguyên nhân từ đâu? Có thể là RAM hỏng, CPU hỏng, card đồ hoạ hỏng, ổ cứng hỏng, mainboard hỏng…

Vậy làm sao để tìm ra nguyên nhân?

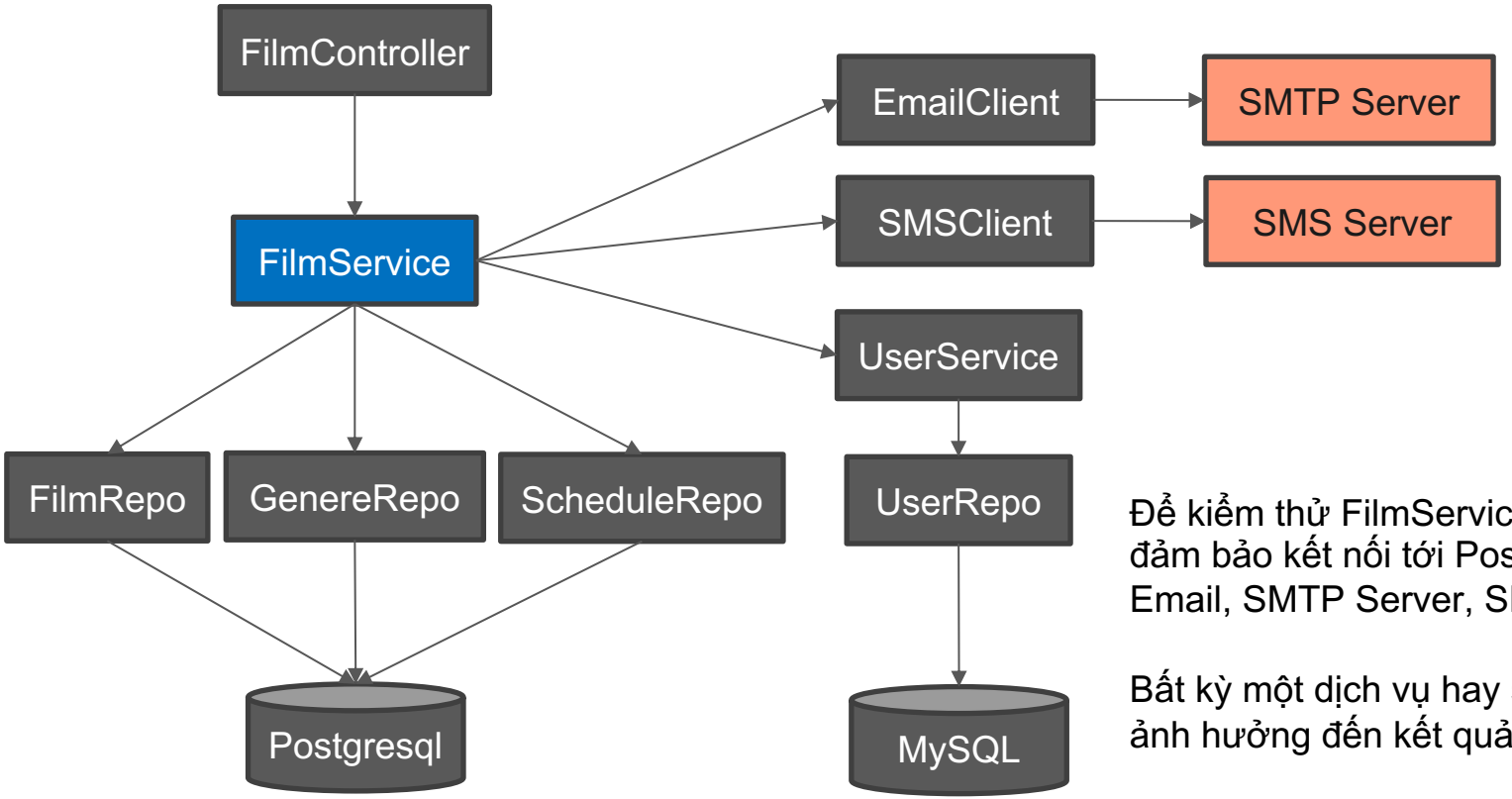
Khi tôi mang máy tính để cửa hàng sửa chữa

Nhân viên kỹ thuật, lần lượt tháo từng bộ phận ra, thay bộ phận tương tự anh ta chắc chắn chạy tốt vào cho đến khi máy tính chạy được.

Cách sửa chữa này ai cũng có thể làm được. Bản chất là thay thế từng thành phần để tìm ra nguyên nhân gây lỗi.



Ứng dụng đặt vé xem phim

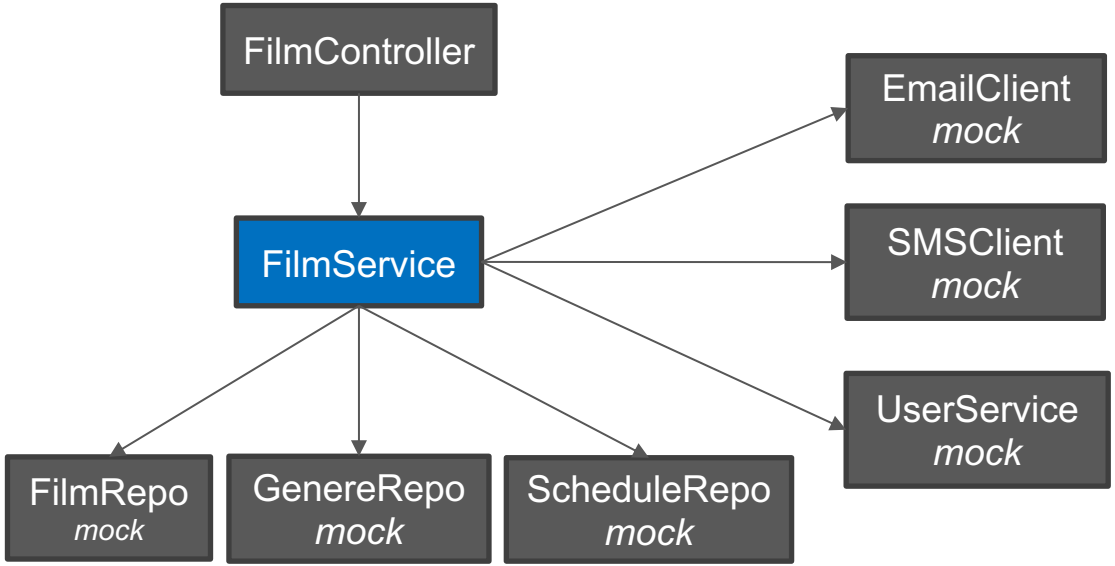


Để kiểm thử FilmService, bạn sẽ phải đảm bảo kết nối tới Postgresql, MySQL, Email, SMTP Server, SMS Server….

Bất kỳ một dịch vụ hay server nào lỗi sẽ ảnh hưởng đến kết quả kiểm thử.

Mock: làm giả đối tượng

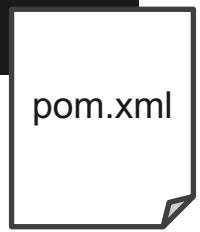
Ứng dụng clean code cần phải đảm bảo từng thành phần phải kiểm thử được. Để kiểm thử tự động logic của FilmService, cần đảm bảo các hàm test luôn chạy mà không phụ thuộc vào CSDL hay dịch vụ bên ngoài. Mock là kỹ thuật / thư viện tạo ra đối tượng giả có những phương thức (stubbing methods) luôn chạy đúng khi được gọi.



Giả mà như thật !

Sử dụng mockito

```xml
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-core</artifactId>
  <version>4.1.0</version>
  <scope>test</scope>
</dependency>
```



Sử dụng khi có và không có @SpringBootTest

- Nếu testing class đã được đánh dấu bởi @SpringBootTest thì bạn có thể dùng các mock annotation luôn.
- Nếu testing class không đánh dấu bởi @SpringBootTest thì bạn cần phải dùng @ExtendWith(MockitoExtension.class)
- Khác biệt @SpringBootTest và @ExtendWith(MockitoExtension.class) là gì? Xem https://stackoverflow.com/questions/61433806/junit-5-with-spring-boot-when-to-use-extendwith-spring-or-mockito
    - @SpringBootTest hỗ trợ @MockBean, nạp đối tượng mock vào ApplicationContext
    - @ExtendWith(MockitoExtension.class) chỉ hỗ trợ @Mock. Tốc độ thực thi nhanh hơn, nhưng không hỗ trợ @Autowired

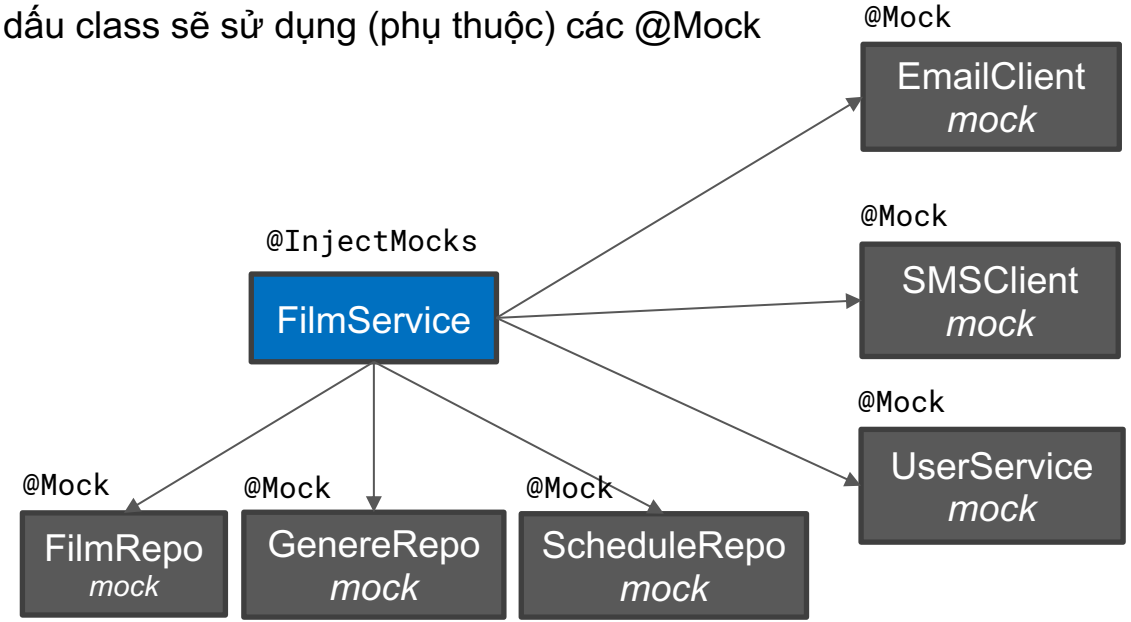
```java
@ExtendWith(MockitoExtension.class)
//@SpringBootTest
class FilmRentTest {
  @Mock
  private FilmRepo filmRepo;

  @Mock
  private CustomerRepo customerRepo;

  @Mock
  private EmailService emailService;

  @InjectMocks
  private FilmService filmService = new
FilmServiceImpl(filmRepo, customerRepo, emailService);
```

@Mock khác gì @MockBean?

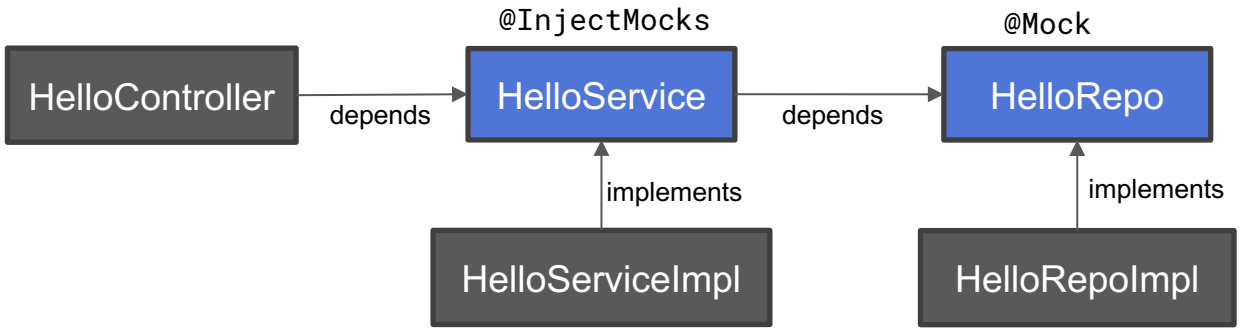
- @Mock đánh dấu một đối tượng giả
- @MockBean tương tự như @Mock nhưng nạp vào Application Context, yêu cầu testing class đánh dấu bằng @SpringBootTest. Khi thực thi sẽ chậm hơn. Dùng với @WebMvcTest để kiểm thử Controller

@InjectMocks

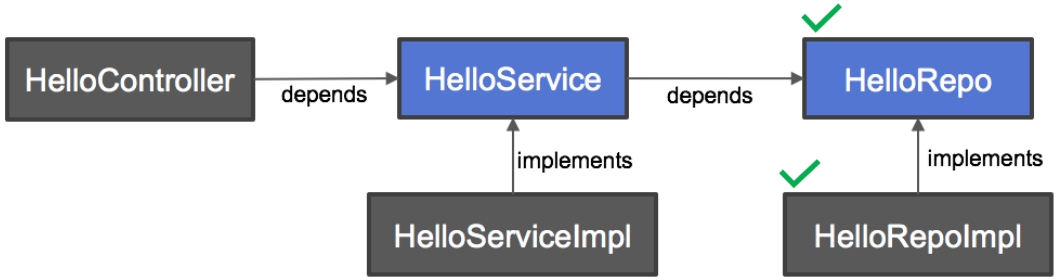
@InjectMock đánh dấu class sẽ sử dụng (phụ thuộc) các @Mock class khác



Một ví dụ đơn giản nhất - basicmock

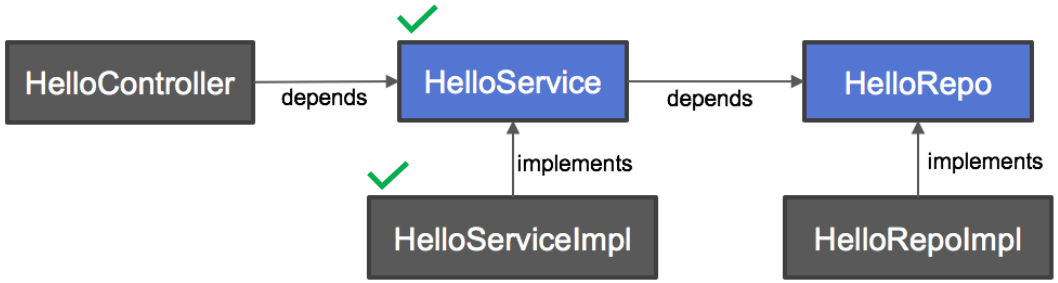


```java
public interface HelloRepo {
  public String hi();
}
```




```java
@Repository
public class HelloRepoImpl implements HelloRepo {
 @Override
 public String hi() {
   return "Hello World";
 }
}
```

```java
public interface HelloService {
  public String say();
}
```




```java
@Service
public class HelloServiceImpl implements HelloService {
  @Autowired private HelloRepo helloRepo;
  @Override
  public String say() {
    return helloRepo.hi();
  }
}
```

```java
@SpringBootTest 1
public class HelloServiceTest {
  @Mock 2
  private HelloRepo helloRepo;

  @InjectMocks 3 // auto inject helloRepo
  private HelloService helloService = new HelloServiceImpl();


  @Test
  void testSay() {
    when(helloRepo.hi()).thenReturn("Mock is great"); 4

    String result = helloService.say();
    assertThat(result).isEqualTo("Mock is great");
  }
}
```

1. @SpringBootTest tạo application context
2. @Mock HelloRepo sẽ tạo đối tượng giả nạp vào application context
3. @InjectMocks đánh dấu helloService sẽ dùng các đối tượng giả (mock object)
4. Khai báo giả lập phương thức ~ stubbing method

Cú pháp when ... thenReturn

- Linh hồn của mocking đó là giả lập phương thức (stubbing method).
- Stubbing method giúp dev chủ động tối đa trong việc xử lý như thế nào khi đối tượng cần kiểm thử gọi đến các đối tượng phụ thuộc
- Stubbing method đảm bảo tất cả đối tượng phụ thuộc chạy đúng theo kịch bản mong muốn để duy nhất đối tượng cần kiểm thử quyết định đúng hay sai.

```
@Test
void testSay() {
  when(helloRepo.hi()).thenReturn("Mock is great");

  String result = helloService.say();
  assertThat(result).isEqualTo("Mock is great");
}
```

Còn nhiều biểu thức khác

- `when thenReturn` trả về đối tượng cụ thể đơn giản
- `when thenAnswer` viết Lambda expression lấy tham số từ vế when, trả về đối tượng động hoặc quăng exeception
- `when thenThrow` quăng exception
- when thenCallRealMethod gọi phương thức thật

https://stacktraceguru.com/unittest/thenreturn-vs-thenanswer

Lấy tham số từ vế when

```
when(filmRepo.findById(anyString()))
.thenAnswer(invocation -> Optional.of(new
Film((String)invocation.getArguments()[0], "Titanic")));
```

Ngoài anyString còn có anyChar, anyInt, anyFloat, anyDouble, anyShort, anyList, anySet, anyMap, anyCollection, anyIterable… Xem thêm về **Mock Argument Matchers**

https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/ArgumentMatchers.html

Nhờ có argument matchers

Có thể viết kịch bản stubbing method như sau:

- Ứng với tham số như thế này, chạy hàm này, trả về kết quả này
- Ứng với tham số như thế kia, chạy hàm khác, trả về kết quả khác
- Ứng với tham số như thế khác nữa, thì chạy hàm thực sự (callRealMethod)

Stubbing void method

```
public interface EmailService {
  public void send(String to, String subject, String body);
}
```

```
when(emailService.send(anyString(), anyString(), anyString()))
.thenThrow(new RuntimeException("Email is invalid"));
```

The method when(T) in the type Mockito is not applicable for the arguments (void)

doThrow

```
doThrow(new RuntimeException("Email is invalid"))
.when(emailService)
.send(anyString(), anyString(), anyString());
```

Các biểu thức với void method

- doCallRealMethod
- doAnswer
- doNothing
- doReturn

```java
public class HelloRepoImpl implements HelloRepo {
  @Override
  public void foo() {
    System.out.println("Foo is called");
  }
}
```

```java
@Test
void testDoNothing() {
  doNothing().when(helloRepo).foo();
  helloRepo.foo();
}
```

Không làm gì nếu phương thức được gọi

```java
@Test
void testCallRealMethod() {
  HelloRepo helloRepo = mock(HelloRepoImpl.class);
  doCallRealMethod().when(helloRepo).foo();
  helloRepo.foo();
}
```

Chạy phương thức thật. Chỉ hợp lý khi xen kẽ gọi phương thức thật và giả

Trộn lẫn việc gọi hàm thật và hàm giả lập

```
HelloRepo helloRepo = mock(HelloRepoImpl.class);

when(helloRepo.bar(anyInt())).thenAnswer(invocation ->{
  int number = (int)invocation.getArguments()[0];
  var list = new ArrayList<String>(List.of("a", "b", "c", "d", "e"));
  if (number < 5) {
    return list.subList(0, number); //Giả lập
  } else {
    return invocation.callRealMethod(); //Gọi hàm thật
  }
});
```

Có nhiều cách giả lập return kết quả

```
doReturn("BII").when(helloRepo).hi(); 1

doAnswer(i -> { 2
   return "BOO";
}).when(helloRepo).hi();

when(helloRepo.hi()).thenReturn("BAA"); 3
```

verify kiểm tra số lần thực thi

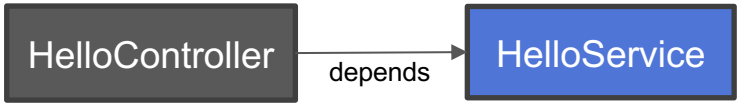
```
verify(mock, times(5)).someMethod("called five times");
verify(mock, atLeast(2)).someMethod("called least 2 times");
verify(mock, atLeastOnce()).someMethod("at least once");
```

```
@Test
void testSay() {
  when(helloRepo.hi()).thenReturn("Mock is great");

  String result = helloService.say();
  assertThat(result).isEqualTo("Mock is great");

  verify(helloRepo, times(1)).hi(); // helloRepo.hi() được gọi ít nhất một lần
}
```

Tổng kết lại mockito làm được gì

1. Giả lập phương thức của mocking object, đảm bảo phương thức này đúng ý đồ của dev để dev chuyên tâm test logic của đối tượng được inject mock
2. Dùng argument marcher để tuỳ biến các trường hợp giả lập theo tham số
3. Giả lập kết quả trả về, giả lập logic, giả lập quăng exception
4. Kiểm tra được phương thức giả lập được gọi bao nhiêu lần

Kiểm thử Controller




```java
@RestController
public class HelloController {
  @Autowired private HelloService helloService;

  @GetMapping("/hi")
  public String hello() {
   return helloService.say();
  }
}
```

```java
@WebMvcTest(HelloController.class)
public class HelloControllerTest {
  @Autowired
  private MockMvc mvc; //Phải có đối tượng này

  @MockBean //Thử thay bằng @Mock xem thế nào
  private HelloService helloService;

  @Test
  public void testHello() throws Exception {
    when(helloService.say()).thenReturn("Umbala");

    mvc.perform(MockMvcRequestBuilders.get("/hi")
     .accept(MediaType.APPLICATION_JSON))
     .andExpect(MockMvcResultMatchers.status().isOk())
     .andExpect(MockMvcResultMatchers.content().string("Umbala"));
  }
}
```

`@WebMvcTest` that can be used for a Spring MVC test that focuses **only** on Spring MVC components. Using this annotation will disable full auto-configuration and instead apply only configuration relevant to MVC tests
(i.e. @Controller, @ControllerAdvice, @JsonComponent, Converter/GenericConverter, Filter, WebMvcConfigurer and HandlerMethodArgumentResolver beans but
not @Component, @Service or @Repository beans).
By default, tests annotated with @WebMvcTest will also auto-configure Spring Security and MockMvc (include support for HtmlUnit WebClient and Selenium WebDriver). For more fine-grained control of MockMVC the @AutoConfigureMockMvc annotation can be used.
Typically @WebMvcTest is used in combination with @MockBean or @Import to create any collaborators required by your @Controller beans.
If you are looking to load your full application configuration and use MockMVC, you should consider @SpringBootTest combined with @AutoConfigureMockMvc rather than this annotation.