Testes camada Controller

Laboratório de Programação

Objetivo

- implementar testes de unidade dos endpoints
- entender e usar Mocks
- aprender o mecanismo da requisição e tratamento da resposta

Testes em UsuarioController

- São simples:
    - deveSalvarUsuario
    - deveAutenticarUsuario
    - deveObterValorInvestimento
- Essencialmente realizam um teste sobre os métodos básicos
- Outros testes também podem ser criados para avaliar os erros e se estão sendo tratados adequadamente

Mas, ...

- O controller usa o Service que usa o Repository

> “
>
> Numa condição ideal, seria melhor usar Service sem que ele realmente use algo de banco de dados
>
> ”

- Ou seja, precisamos `Mock` das funcionalidades

Mocks

- Mock serve para criar uma versão fictícia de um objeto ou comportamento de um objeto

> “
>
> Um objeto mock é uma implementação fictícia para uma interface ou classe na qual você define a saída de certas chamadas de método.
>
> São configurados para executar um determinado comportamento durante um teste.
>
> ”

Mocks

- Com o Junit, nós usamos o framework `Mockito`
    - já está dentro do Junit
- Códigos essenciais:
    - `Usuario dummy = Mockito.mock(Usuario.class)`
    - `Mokito.anyString()`
    - `Mockito.when( ... ).then ...`
    - `@MockBean`: para injeção de dependência
    - `MockMvc`: para um mock de classe com funcionalidade de request
    - `MockHttpServletRequestBuilder`: builder de requisição
    - `mvc.perform(request).andExpect (.. matcher ..)` : teste em si
    - `MockMvcResultMatchers`: testador

Começando a fazer o UsuarioControllerTest

- Continua carregando o contexto `@ExtendWith(SpringExtension.class)`
- Mantemos o `@ActiveProfiles("test")`

```
@ExtendWith(SpringExtension.class)
@ActiveProfiles("test")
public class UsuarioControllerTest {

}
```

Começando a fazer o UsuarioControllerTest

- Primeira alteração: dizer que é um teste de um RestController (ou Controller)
  - para isso adicionamos o `@WebMvcTest`

```
@ExtendWith(SpringExtension.class)
@ActiveProfiles("test")
@WebMvcTest (controllers =  UsuarioController.class)
public class UsuarioControllerTest {

}
```

Começando a fazer o UsuarioControllerTest

- finalmente, adicionamos `@AutoConfigureMockMvc` para a injeção de dependência funcionar

```
@ExtendWith(SpringExtension.class)
@ActiveProfiles("test")
@WebMvcTest (controllers =  UsuarioController.class)
@AutoConfigureMockMvc
public class UsuarioControllerTest {

}
```

Finalizando a configuração

- Adicionar referência `MockMvc` para realizar requisições
- Adicionar o servico como `Mock`
- Para facilidar, vamos colocar a URL da api

```
@ExtendWith(SpringExtension.class)
@ActiveProfiles("test")
@WebMvcTest (controllers =  UsuarioController.class)
@AutoConfigureMockMvc
public class UsuarioControllerTest {

    static final String API = "/api/usuarios";

    @Autowired
    MockMvc mvc;

    @MockBean
    UsuarioService service;
```

Ok, vamos aos teste.

- São 3:

```
    @Test
    public void deveSalvarUsuario() {}
    @Test
    public void deveAutenticarUsuario() {}
    @Test
    public void deveObterValorInvestimento(){}
```

Testando endpoint de salvar usuário

```
@Test
public void deveSalvarUsuario() throws Exception{
    //cenário
    //dto para virar json
    UsuarioDTO dto = UsuarioDTO
                        .builder().nome("teste")
                        .email("teste@teste.com").senha("123").bu
    //resposta que será mock
    Usuario usuario = Usuario
                        .builder().id(1l).nome("teste")
                        .email("teste@teste.com").senha("123").bu
    //mock salvar
    Mockito.when(service.salvar(
              Mockito.any(Usuario.class))).thenReturn(usuario);

    //converte DTO para json
    String json = new ObjectMapper().writeValueAsString(dto);
```

Testando endpoint de salvar usuário

```
...
  //ação
  //constrói a requisição post
  MockHttpServletRequestBuilder request =
              MockMvcRequestBuilders.post(API)   //na URI
              .accept ( MediaType.APPLICATION_JSON)
              .contentType (MediaType.APPLICATION_JSON)
              .content (json); //mandando o DTO

  //ação e verificação
  mvc.perform(request)
      .andExpect(   //teste em si
          MockMvcResultMatchers.status().isCreated()); //espera 200
}
```

Testando autenticar

```
@Test
public void deveAutenticarUsuario() throws Exception{
    //cenário
    UsuarioDTO dto = UsuarioDTO.builder()
                            .email("teste@teste.com")
                            .senha("123").build();

    Mockito.when(
          service.efetuarLogin(
              Mockito.anyString(), Mockito.anyString()))
              .thenReturn(true);
    String json = new ObjectMapper().writeValueAsString(dto);
```

Testando autenticar

```
@Test
public void deveAutenticarUsuario() throws Exception{
    //cenário
    UsuarioDTO dto = UsuarioDTO.builder()
                            .email("teste@teste.com")
                            .senha("123").build();

    Mockito.when(
          service.efetuarLogin(
              Mockito.anyString(), Mockito.anyString()))
              .thenReturn(true);
    String json = new ObjectMapper().writeValueAsString(dto);
    //ação
    MockHttpServletRequestBuilder request =
                MockMvcRequestBuilders.post(API.concat("/autenticar
                .accept (MediaType.APPLICATION_JSON)
                .contentType (MediaType.APPLICATION_JSON)
                .content (json);

    //ação e verificação
    mvc.perform(request)
        .andExpect(MockMvcResultMatchers.status().isOk());
}
```

Testando obter investimento

```
@Test
public void deveObterValorInvestimento() throws Exception{
    //cenário
    List<InvestimentoSaldo> res =
        new ArrayList<InvestimentoSaldo>();
    res.add(new InvestimentoSaldo(new Investimento(), 500.0));

    Mockito.when(
        service.obterSaldos(Mockito.anyLong()))
               .thenReturn(res);
```

Testando obter investimento

```
@Test
public void deveObterValorInvestimento() throws Exception{
    //cenário
    List<InvestimentoSaldo> res =
        new ArrayList<InvestimentoSaldo>();
    res.add(new InvestimentoSaldo(new Investimento(), 500.0));

    Mockito.when(
        service.obterSaldos(Mockito.anyLong()))
               .thenReturn(res);

    //ação
    MockHttpServletRequestBuilder request =
                MockMvcRequestBuilders
                    .get(API.concat("?usuario=1"))
                    .accept(MediaType.APPLICATION_JSON);

    //ação e verificação
    mvc.perform(request)
        .andExpect(
            MockMvcResultMatchers.status().isOk())
        .andExpect(
            MockMvcResultMatchers
                .jsonPath("$[0].valor").value(500.0));
}
```

Agora é construir os testes para os demais controllers