

Quiz Questions

(1) Unit Testing

← Classroom

Units Testing
Unit Testing

?

V

Problem Submissions Doubts

Unit Testing

Easy • Score 20/20

Problem statement

What does a unit test typically **NOT** involve?

Send feedback

Options: Pick one correct answer from below

☐ Testing individual functions or methods

☒ Testing the interaction between modules

☐ Ensuring code coverage of specific functionalities

☐ Simulating different scenarios and edge cases

Solution description

Unit tests are designed to verify the functionality of individual units or components. They do not directly assess how different modules or units interact with each other, which is the domain of integration testing.

(2) Integration testing

← Classroom

Units Testing
Integration Testing

?

V

Problem Submissions Doubts

Integration testing

Easy • Score 20/20

Problem statement

The integration testing level verifies interactions between multiple integrated components/modules.

Send feedback

Options: Pick one correct answer from below

☒ True

☐ False

Solution description

Integration testing is specifically concerned with testing the interactions between integrated components or modules to ensure they work together as intended.

(3) Assertions

← Classroom

Units Testing
Assertions

?

V

Problem Submissions Doubts

Assertions

Easy • Score 20/20

Problem statement

Assertion statements in testing are primarily used to

Send feedback

Options: Pick one correct answer from below

☐ Generate random test data.

☐ Define the structure of test cases.

☒ Check and validate expected outcomes

☐ Simulate user interactions.

Solution description

Assertions are employed within test cases to check and validate whether the actual outcomes match the expected results. They ensure that the system behaves as anticipated and that the conditions and outputs conform to the expected behaviour defined within the test. Assertions help maintain the integrity of tests by automatically verifying the correctness and signalling any discrepancies between the expected and actual outcomes.

(4) Assertions in testing

← Classroom

JUnits Testing
Assertions in testing

?

Problem Submissions Doubts

✓ Assertions in testing
Easy • Score 20/20

Send feedback

Problem statement
Which of the following is **NOT** an assertion in testing?

Options: Pick one correct answer from below

Attempts left: 1/2

☐ assertEquals()

☐ assertTrue()

☒ verify()

☐ assertFalse()

Solution description

assertEquals(), assertTrue(), and assertFalse() are commonly used assertion methods in testing frameworks like JUnit or TestNG to verify expected conditions against actual outcomes. verify() is not typically considered an assertion method. It's commonly used in mocking frameworks (like Mockito) to verify whether certain methods have been called or not on mock objects, particularly during interaction-based testing (e.g., verifying method invocations argument matching) rather than verifying outcomes.

(5) Purpose of Mocking

← Classroom

JUnits Testing
Purpose of Mocking

?

Problem Submissions Doubts

✓ Purpose of Mocking
Easy • Score 20/20

Send feedback

Problem statement
What is the primary purpose of mocking in unit testing?

Options: Pick one correct answer from below

☒ To replace actual code with fake implementations

☐ To test the entire application end-to-end

☐ To slow down the testing process

☐ To generate random test data

Solution description

Mocking in unit testing involves substituting actual dependencies of the unit under test with simulated objects or mock objects. These mock objects mimic the behaviour of actual dependencies, allowing the unit to be tested in isolation without relying on or impacting external systems. This facilitates controlled testing environments and helps focus solely on the behaviour of the specific unit being tested.

(6) Course Management System

← Classroom

JUnits Testing
Course Management System

?

Problem Submissions Doubts

✓ Course Management System
Easy • Score 20/20

Send feedback

Problem statement
Suppose you have a Course Management application where you want to create and test the service layer. Choose the correct option to test the addCourse() method in the CourseService.

Options: Pick one correct answer from below

```
import org.junit.Test;
import org.junit.Before;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import static org.mockito.Mockito.*;

public class CourseServiceTest {

    @Mock
    CourseRepository courseRepository;

    CourseService courseService;

    @Before
    public void setUp() {
        MockitoAnnotations.openMocks(this);
        courseRepository = new CourseRepository();
        courseService = new CourseService(courseRepository);
    }

    @Test
    public void addCourse() {
        Course course = new Course("Java", "Programming Language");
        Course addedCourse = courseService.addCourse(course);
        assertEquals("Course added successfully", addedCourse, course);
    }
}
```

```
import org.junit.Test;
import org.junit.Before;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import static org.mockito.Mockito.*;

public class CourseServiceTest {

    @Mock
    CourseRepository courseRepository;

    CourseService courseService;

    @Before
    public void setUp() {
        MockitoAnnotations.openMocks(this);
        courseRepository = new CourseRepository();
        courseService = new CourseService(courseRepository);
    }

    @Test
    public void addCourse() {
        Course course = new Course("Java", "Programming Language");
        Course addedCourse = courseService.addCourse(course);
        assertEquals("Course added successfully", addedCourse, course);
    }
}
```

```

    @Test
    @DisplayName("Should add a new course")
    public void shouldAddCourse() {
        // Mocking the behavior of the repository's save() method
        Mockito.when(courseRepository.save(Mockito.any()))
            .thenReturn(courseToAdd);

        // Calling the addCourse() method of the CourseService
        Course addedCourse = courseService.addCourse(courseToAdd);
        Assertions.assertEquals(courseToAdd, addedCourse);
    }
}

```

☐ None of the above

Solution description

Explanation of the Test:

- Setup:
 - The test method initializes a CourseService instance and mocks the CourseRepository, simulating the behaviour of the data access layer.
 - A sample Course object (courseToAdd) representing a new course is created with specific attributes (e.g., ID, name, description).
- Mock Behavior:
 - The behaviour of the CourseRepository is mocked using Mockito's when().thenReturn() construct. It mocks the behaviour of the repository's save() method to simulate adding a course to the database.
- Execution:
 - The addCourse() method of the CourseService is called with the courseToAdd object as an argument.
- Assertion:
 - The test verifies whether the addCourse() method returns the expected result. In this case, it expects the addCourse() method to return the same Course object passed to it (courseToAdd).

Example Code Explanation:

```

    @Test
    @DisplayName("Should add a new course")
    public void shouldAddCourse() {
        // Creating a sample Course object
        Course courseToAdd = new Course(1, "Java", "Programming language");

        // Mocking the behavior of the repository's save() method
        Mockito.when(courseRepository.save(Mockito.any()))
            .thenReturn(courseToAdd);

        // Calling the addCourse() method of the CourseService
    }
}

```

(7) Course Management System II

Classroom

JUnits Testing
Course Management System II

Problem
Submissions
Doubts

Course Management System II
Easy
Score 20/20

Problem statement
Send feedback

For the Course Application, choose the correct option to test the getAllCourses() method in the CourseService.

```

    @Service
    public class CourseService {

        private final CourseRepository courseRepository;

        @Autowired
        public CourseService(CourseRepository courseRepository) {
            this.courseRepository = courseRepository;
        }

        public Course addCourse(Course course) {
            return courseRepository.save(course);
        }

        public List<Course> getAllCourses() {
            return courseRepository.findAll();
        }
    }

```

Options: Pick one correct answer from below

☐

```

    @Test
    @DisplayName("Should get all courses")
    public void shouldGetAllCourses() {
        List<Course> expectedCourses = Arrays.asList(
            new Course(1, "Java", "Programming language"),
            new Course(2, "Python", "Scripting language"));
        Mockito.when(courseRepository.findAll())
            .thenReturn(expectedCourses);

        List<Course> actualCourses = courseService.getAllCourses();
        Assertions.assertEquals(expectedCourses, actualCourses);
    }

```

☒

```

    @Test
    @DisplayName("Should get all courses")
    public void shouldGetAllCourses() {
        List<Course> expectedCourses = Arrays.asList(
            new Course(1, "Java", "Programming language"),
            new Course(2, "Python", "Scripting language"));
        Mockito.when(courseRepository.findAll())
            .thenReturn(expectedCourses);

        List<Course> actualCourses = courseService.getAllCourses();
        Assertions.assertEquals(expectedCourses.size(), actualCourses.size());
    }

```

☐

```

    @Test
    @DisplayName("Should get all courses")
    public void shouldGetAllCourses() {
        List<Course> expectedCourses = Arrays.asList(
            new Course(1, "Java", "Programming language"),
            new Course(2, "Python", "Scripting language"));
        Mockito.when(courseRepository.findAll())
            .thenReturn(actualCourses);

        List<Course> actualCourses = courseService.getAllCourses();
        Assertions.assertEquals(expectedCourses.size(), actualCourses.size());
    }

```

(8) Course Management System III

← Classroom

JUnits Testing
Course Management System III

Problem Submissions Doubts

✓ Course Management System III

Easy • Score 20/20

Problem statement

Send feedback

For the course application, you must test the DAL Layer. Choose the correct test case implementation of the findAllCourses functionality.

Options: Pick one correct answer from below

☒

```
***
@Test
public void shouldFindAllCourses() {
    Course course1 = new Course(1, "C++", "General-purpose language");
    Course course2 = new Course(2, "JavaScript", "Web scripting language");

    courseRepository.save(course1);
    courseRepository.save(course2);

    List<Course> courses = courseRepository.findAll();

    assertThat(courses, isEmpty());
    assertThat(courses, contains(course1, course2));
}
```

☐

```
***
@Test
public void shouldFindAllCourses() {
    Course course1 = new Course(1, "C++", "General-purpose language");
    Course course2 = new Course(2, "JavaScript", "Web scripting language");

    courseRepository.save(course1, course2);
    List<Course> courses = courseRepository.findAll();

    assertThat(courses, isEmpty());
    assertThat(courses, contains(course1, course2));
}
```

☐

```
***
@Test
public void shouldFindAllCourses() {
    Course course1 = new Course(1, "C++", "General-purpose language");
    Course course2 = new Course(2, "JavaScript", "Web scripting language");

    courseRepository.save(course1);
    courseRepository.save(course2);

    List<Course> courses = courseRepository.findAll();

    assertThat(courses, isEmpty());
    assertThat(courses, contains(course1, course2));
}
```

(9) Course Management Application IV

← Classroom

JUnits Testing
Course Management Application IV

Problem Submissions Doubts

✓ Course Management Application IV

Easy • Score 20/20

Problem statement

Send feedback

For the course application, we must test the DAL Layer using a dummy database. The code below is the courseRepositoryTest class for testing the CourseRepository.

```
***
@RunWith(Spring.class)
public class CourseRepositoryTest {

    @Autowired
    private CourseRepository courseRepository;

    private static final MySQLContainer MY_SQL_CONTAINER = new MySQLContainer("mysql:latest")
        .withDatabaseName("test-db")
        .withUsername("testuser")
        .withPassword("password");

    static {
        MY_SQL_CONTAINER.start();
    }

    @Before
    public void setUp() {
        // test cases
    }
}
```

Options: Pick one correct answer from below

☐ The MySQLContainer is not properly configured with the necessary initialization or configurations for the test database, leading to connection failures.

☐ The absence of a proper @SpringBootTest annotation resulted in the application context not loading properly, leading to the database connection issue.

☒ The missing @Testcontainers, and @AutoConfigureTestDatabase annotations led to the MySQLContainer not starting, causing a lack of a running database instance during the test.

☐ None of the above

Solution description

@Testcontainers is used to mark the test class CourseRepositoryTest to enable the usage of Testcontainers. It provides a way to manage and use containers within the test environment, like the MySQL container (MySQLContainer).
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE): This annotation is a part of Spring Boot's testing support. It's used to specify the behaviour of auto-configuration for the test database. In this case, replace = AutoConfigureTestDatabase.Replace.NONE indicates that Spring Boot should not replace the configured database.

(10) Course Management Application V

Problem Submissions Doubts

✓ Course Management Application V

Easy • Score 20/20

Problem statement

Send feedback

For the course application, we must test the controller. Choose the correct implementation for testing the `getCourseById()` API.
API: `GET /courses/{id}`

Options: Pick one correct answer from below

☒

```
***
@Test
void shouldTestGetCourseById() throws Exception {
    Course expectedCourse = new Course();
    expectedCourse.setId(1L);
    expectedCourse.setName("Java Programming");
    Mockito.when(courseService.getCourseById(1L))
        .thenReturn(Optional.of(expectedCourse));

    mockMvc.perform(MockMvcRequestBuilders.get("/courses/1"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath("$.name")
            .value("Java Programming"));
}
```

☐

```
***
@Test
void shouldTestGetCourseById() throws Exception {
    Course expectedCourse = new Course();
    expectedCourse.setId(1L);
    expectedCourse.setName("Java Programming");
    Mockito.when(courseService.getCourseById(1L))
        .thenReturn(Optional.of(expectedCourse));

    mockMvc.perform(MockMvcRequestBuilders.get("/courses/1"))
        .andExpect(MockMvcResultMatchers.status().isOk());
}
```

☐

```
***
@Test
void shouldTestGetCourseById() throws Exception {
    Course expectedCourse = new Course();
    expectedCourse.setId(1L);
    expectedCourse.setName("Java Programming");
    Mockito.when(courseService.getCourseById(1L))
        .thenReturn(Optional.of(expectedCourse));

    mockMvc.perform(MockMvcRequestBuilders.get("/courses/1"))
        .andExpect(MockMvcResultMatchers.status().isOk());
        .andExpect(MockMvcResultMatchers.jsonPath("$.name")
            .value("C++ Programming"));
}
```

(11) Course Management Application VI

← Classroom

JUnits Testing
Course Management Application VI

Problem Submissions Doubts

✓ Course Management Application VI

Easy • Score 20/20

Problem statement

Send feedback

For the course application, choose the correct implementation of testing the `createCourse()` API: `POST /courses`

Options: Pick one correct answer from below

☒

```
***
@Test
void shouldTestCreateCourse() throws Exception {
    Course courseToCreate = new Course();
    courseToCreate.setName("New Course");
    Mockito.when(courseService.createCourse(courseToCreate))
        .thenReturn(courseToCreate);

    String request = objectMapper.writeValueAsString(courseToCreate);

    mockMvc.perform(MockMvcRequestBuilders.post("/courses")
        .contentType("application/json").content(request))
        .andExpect(MockMvcResultMatchers.status().isOk());
}
```

☐

```
***
@Test
void shouldTestCreateCourseIncorrectMocking() throws Exception {
    Course courseToCreate = new Course();
    courseToCreate.setName("New Course");
    Mockito.when(courseService.createCourse(courseToCreate)) .thenReturn(null);
    String request = objectMapper.writeValueAsString(courseToCreate);

    mockMvc.perform(MockMvcRequestBuilders.post("/courses/")
        .contentType("application/json").content(request))
        .andExpect(MockMvcResultMatchers.status().isOk());
}
```

☐

```
***
@Test
void shouldTestCreateCourseIncorrectRequest() throws Exception {
    Course courseToCreate = new Course();
    courseToCreate.setId(1L);
    Mockito.when(courseService.createCourse(courseToCreate)) .thenReturn(courseToCreate);
    String request = objectMapper.writeValueAsString(courseToCreate);

    mockMvc.perform(MockMvcRequestBuilders.post("/courses/")
        .contentType("application/json").content(request))
        .andExpect(MockMvcResultMatchers.status().isOk());
}
```

(12) Course Application VII

← Classroom

Junits Testing
Course Management Application VII

100% — +

Problem Submissions Doubts

✓ Course Application VII

Easy • Score 20/20

Send feedback

In the course application, we have included user authentication and authorization. Choose the correct implementation for testing the updateUser() API.

```
***
@Test
public void shouldTestUpdateUser() {
    User existingUser = new User();
    existingUser.setId(1L);
    existingUser.setUsername("oldUsername");
    existingUser.setPassword("oldPassword");

    UserRequest updatedUserRequest = new UserRequest();
    updatedUserRequest.setUsername("newUsername");
    updatedUserRequest.setPassword("newPassword");

    Optional<User> optionalUser = Optional.of(existingUser);
    when(userRepository.findById(1L)).thenReturn(optionalUser);

    BCryptPasswordEncoder bCryptPasswordEncoder = mock(BCryptPasswordEncoder.class);
    when(bCryptPasswordEncoder.encode(updatedUserRequest.getPassword()))
        .thenReturn("encodedNewPassword");

    User updatedUser = new User();
    updatedUser.setId(1L);
    updatedUser.setUsername("newUsername");
    updatedUser.setPassword("encodedNewPassword");

    when(userRepository.save(any(User.class))).thenReturn(updatedUser);

    userService.updateUser(1L, updatedUserRequest);
}
***
```

```
***
@Test
public void shouldTestUpdateUser() {
    User existingUser = new User();
    existingUser.setId(1L);
    existingUser.setUsername("oldUsername");
    existingUser.setPassword("oldPassword");

    UserRequest updatedUserRequest = new UserRequest();
    updatedUserRequest.setUsername("newUsername");
    updatedUserRequest.setPassword("newPassword");

    Optional<User> optionalUser = Optional.of(existingUser);
    when(userRepository.findById(1L)).thenReturn(optionalUser);

    User updatedUser = new User();
    updatedUser.setId(1L);
    updatedUser.setUsername("newUsername");
    updatedUser.setPassword("encodedNewPassword");

    when(userRepository.save(any(User.class))).thenReturn(updatedUser);

    userService.updateUser(1L, updatedUserRequest);
    verify(userRepository).save(any(User.class));
}
***
```

```
***
@Test
public void shouldTestUpdateUser() {
    User existingUser = new User();
    existingUser.setId(1L);
    existingUser.setUsername("oldUsername");
    existingUser.setPassword("oldPassword");

    UserRequest updatedUserRequest = new UserRequest();
    updatedUserRequest.setUsername("newUsername");
    updatedUserRequest.setPassword("newPassword");

    Optional<User> optionalUser = Optional.of(existingUser);
    when(userRepository.findById(1L)).thenReturn(optionalUser);

    BCryptPasswordEncoder bCryptPasswordEncoder = mock(BCryptPasswordEncoder.class);
    when(bCryptPasswordEncoder.encode(updatedUserRequest.getPassword()))
        .thenReturn("encodedNewPassword");

    User updatedUser = new User();
    updatedUser.setId(1L);
    updatedUser.setUsername("newUsername");
    updatedUser.setPassword("encodedNewPassword");

    when(userRepository.save(any(User.class))).thenReturn(updatedUser);

    userService.updateUser(1L, updatedUserRequest);
    verify(userRepository).save(any(User.class));
}
***
```

☐ None of the above

Solution description

- Setup:
 - The test method initializes an existing User object (existingUser) representing a user in the system with some predefined details.
 - It creates a UserRequest object (updatedUserRequest) containing the updated information for the user.
- Mock Behavior:
 - The userRepository.findById() method is mocked to return an Optional containing the existingUser when searching by the user's ID.
- Updating User Information:
 - The updateUser() method of the UserService is invoked, passing the user ID and the updated user details.
- Assertions and Verification:
 - The test validates whether the updateUser() method correctly handles the update process by ensuring the repository's save() method is called with the updated user information.
 - It verifies that the repository's save() method is called with the updated user object.

(13) Course Management Application VIII

← Classroom

Junits Testing
Course Management Application VIII

Problem Submissions Doubts

✓ Course Management Application VIII

Easy • Score 20/20

Problem statement

[Send feedback](#)

In our course application, we must test the login API; we have implemented jwt authentication and authorization. Choose the correct implementation of the test.

Options: Pick one correct answer from below

☒

```
***
@Test
public void shouldTestForValidLogin() {
    JwtRequest jwtRequest = new JwtRequest("username", "password");
    UserDetails userDetails = Mockito.mock(UserDetails.class);
    String token = "testToken";

    when(userDetailsService.loadUserByUsername(jwtRequest.getUsername()))
        .thenReturn(userDetails);
    when(jwtAuthenticationHelper.generateToken(userDetails))
        .thenReturn(token);

    JwtResponse jwtResponse = authService.login(jwtRequest);

    assertNotNull(jwtResponse);
    assertEquals(token, jwtResponse.getJwtToken());
}
```

☐

```
***
@Test
public void shouldTestForInvalidLogin() {
    JwtRequest jwtRequest = new JwtRequest("username", "password");
    UserDetails userDetails = Mockito.mock(UserDetails.class);
    String token = "testToken";

    when(userDetailsService.loadUserByUsername(jwtRequest.getUsername()))
        .thenReturn(null);
    when(jwtAuthenticationHelper.generateToken(userDetails)).thenReturn(token);

    JwtResponse jwtResponse = authService.login(jwtRequest);

    assertNotNull(jwtResponse);
    assertEquals(token, jwtResponse.getJwtToken());
}
```

(14) Course Management Application IX

← Classroom

Junits Testing
Course Management Application IX

Problem Submissions Doubts

✓ Course Management Application IX

Easy • Score 20/20

Problem statement

[Send feedback](#)

In the Course Application, we must test the DAL/Repository layer. Choose the correct test case implementation of testing deletion of the course.

Options: Pick one correct answer from below

☒

```
***
@Test
public void shouldTestDeleteCourseRepo() {
    Course course = new Course();
    course.setName("Introduction to Spring boot");
    course.setDescription("Web development with spring boot");

    Course savedCourse = courseRepository.save(course);

    courseRepository.deleteById(savedCourse.getId());
    boolean isCoursePresent = courseRepository.existsById(savedCourse.getId());

    assertThat(isCoursePresent).isFalse();
}
```

☐

```
***
@Test
public void shouldTestDeleteCourseRepo() {
    Course course = new Course();
    course.setName("Introduction to Spring boot");
    course.setDescription("Web development with spring boot");

    courseRepository.deleteById(savedCourse.getId());
    boolean isCoursePresent = courseRepository.existsById(savedCourse.getId());

    assertThat(isCoursePresent).isFalse();
}
```

(15) Course Management Application X

← Classroom

Junits Testing
Course Management Application X

?

V

Problem Submissions Doubts

Course Management Application X

Easy • Score 20/20

Problem statement

Send feedback

For our Course Application, we are trying to test the courseController. The codes are given below.

```
***
public class CourseControllerTest {

    @MockBean
    private CourseService courseService;

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void shouldTestGetAllCourses() throws Exception {
        Course course = new Course();
        course.setId(1);
        course.setName("Introduction to Programming");
        course.setInstructor("John Doe");

        Mockito.when(courseService.getAllCourses())
            .thenReturn(Collections.singletonList(course));

        mockMvc.perform(get("/courses")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk());
    }
}
```

Options: Pick one correct answer from below

☒ The @WebMvcTest annotation is missing from the class. ✓

☐ We have to autowire the courseService.

☐ The @SpringBootTest annotation is missing.

☐ None of the above

Solution description

The @WebMvcTest annotation is used in Spring Boot applications to specifically test the web layer components, such as controllers, without starting the full application context. It focuses on testing slices of the application of MVC (Model-View-Controller) components. It provides methods (e.g., mockMvc.perform(...)) to simulate HTTP requests and validate the responses, enabling thorough testing of controller endpoints.

(16) Course Management Application XI

← Classroom

Junits Testing
Course Management Application XI

?

V

Problem Submissions Doubts

Course Management Application XI

Easy • Score 20/20

Problem statement

Send feedback

In the Course Application, we now have to perform integration testing. Choose the correct test case implementation of testing updation of the course. The update course API is:
PUT "/courses/{id}"

☐

```
***
@Test
public void shouldTestUpdateCourse() throws Exception {
    Course course = new Course();
    course.setName("Java Programming");
    course.setDescription("Learn Java programming");

    mockMvc.perform(post("/courses")
        .contentType("application/json")
        .content(asJsonString(course)))
        .andExpect(status().isCreated())
        .andExpect(jsonPath("$.name").value("Java Programming"));

    course.setName("Advanced Java Programming");

    mockMvc.perform(put("/courses/{id}", 1))
        .contentType("application/json")
        .content(asJsonString(course))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.id").value(1));
}
```

☒

```
***
@Test
public void shouldTestUpdateCourse() throws Exception {
    Course course = new Course();
    course.setName("Java Programming");
    course.setDescription("Learn Java programming");

    mockMvc.perform(post("/courses")
        .contentType("application/json")
        .content(asJsonString(course)))
        .andExpect(status().isCreated())
        .andExpect(jsonPath("$.name").value("Java Programming"));

    course.setName("Advanced Java Programming");

    mockMvc.perform(put("/courses/{id}", 1))
        .contentType("application/json")
        .content(asJsonString(course))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.name").value("Advanced Java Programming"));
}
```


(17) Course Management Application XII

← Classroom

JUnits Testing
Course Management Application XII

?

Problem Submissions Doubts

✓ Course Management Application XII

Easy • Score 20/20

Problem statement

Send feedback

For the course application, the integration test class is given below. Which annotations should be used in the class to test the order controllers?

```
public class CourseControllerIntegrationTest {  
  
    private static final MySQLContainer MY_SQL_CONTAINER = new MySQLContainer<>("mysql:latest")  
        .withDatabaseName("hotel-test-db")  
        .withUsername("testUser")  
        .withPassword("password");  
  
    @BeforeAll  
    static void beforeAll(){  
        MY_SQL_CONTAINER.start();  
    }  
  
    @BeforeAll  
    static void afterAll(){  
        MY_SQL_CONTAINER.stop();  
    }  
  
    @DynamicPropertySource  
    static void registerDatabaseProperties(DynamicPropertyRegistry registry) {  
        registry.add("spring.datasource.url", MY_SQL_CONTAINER::getJdbcUrl);  
        registry.add("spring.datasource.username", MY_SQL_CONTAINER::getUsername);  
        registry.add("spring.datasource.password", MY_SQL_CONTAINER::getPassword);  
    }  
}
```

Options: Pick one correct answer from below

☒ @SpringBootTest and @AutoConfigureMockMvc

☐ @SpringBootTest and @Testcontainers

☐ @WebMvcTest and @AutoConfigureMockMvc

☐ None of the above

Solution description

@SpringBootTest initialises the complete Spring Boot application context, allowing comprehensive integration testing across different layers. @AutoConfigureMockMvc configures the MockMvc instance within the Spring context, enabling controlled testing of controller endpoints without making actual HTTP requests. Together, these annotations facilitate testing the integration of MVC components (controllers) with other layers, such as services, repositories, etc., providing a robust framework for integration testing while controlling the MVC layer's behaviour. When used together, these annotations allow for powerful integration testing, ensuring that the various components of the application work harmoniously while focusing on the behaviour and interaction of the MVC components in a controlled environment.

(18) AAA Pattern

← Classroom

JUnits Testing
AAA Pattern

?

Problem Submissions Doubts

✓ AAA Pattern

Easy • Score 20/20

Problem statement

Send feedback

Which aspect is NOT typically associated with the 'Assert' phase in unit testing following AAA?

Options: Pick one correct answer from below

☐ Verifying the expected behaviour or outcomes

☒ Executing the code being tested

☐ Checking whether the actual results match the expected results

☐ Reporting test failures or successes

Solution description

The Assert phase is not about executing the code; instead, it involves checking or asserting whether the actual outcomes match the expected results.

(19) When Phase

← Classroom

JUnits Testing
When Phase

?

Problem Submissions Doubts

✓ When Phase

Easy • Score 20/20

Problem statement

Send feedback

What role does the 'When' phase play in the Given-When-Then pattern?

Options: Pick one correct answer from below

☐ It defines the expected outcomes of the test case.

☐ It sets up the initial conditions for the test.

☒ It executes the specific action or event being tested.

☐ It validates the results of the test.

Solution description

In the Given-When-Then pattern, the When phase executes the specific action or event being tested.

(20) Timeout in test cases

← Classroom

JUnits Testing
Timeout in test cases

?

V

≡

Problem

Submissions

Doubts

✔ Timeout in test cases

Easy • Score 20/20

Send feedback

Problem statement

What is the purpose of the 'timeout' parameter in test annotations?

Options: Pick one correct answer from below

Attempts left: 1/2

☐ It specifies the time a test case takes to execute successfully.

☐ It determines the time delay before starting a test case.

☒ It sets a time limit for executing a test case. ✔

☐ It defines the interval between consecutive test executions.

Solution description

The timeout parameter sets a time limit for executing a test case, and if the test exceeds this limit, the test framework marks it as failed.