Step	Package	Class	Description	Test Condition	Expected Result	Actual Result	Pass or Fail (P or F)	Tester
1	**Repository**	**UserRepository**	This test checks if the `findByEmail` method of the `userRepositoryMock` object returns the same `User` object that was previously saved with the same email address.	It uses Mockito's `when` method to mock the expected behavior of the `findByEmail` method, and the `assertThat` method with JUnit's `isEqualTo` method to check if the returned `User` object is equal to the expected `user` object.	The expected result of this test is that the `findByEmail` method of the `userRepositoryMock` object should return the same `User` object that was previously saved with the same email address	Actual value returned by the method matches the expected value	P	Sarab
2	**Repository**	**UserRepository**	This test checks if the `findByUsernameOrEmail` method of the `userRepositoryMock` object properly returns an `Optional` object that contains the expected `User` object, when provided with a valid email address	It uses Mockito's `when` method to mock the expected behavior of the `findByUsernameOrEmail` method, and the `assertThat` method with AssertJ's `isPresent` and `contains` methods to check if the returned `Optional` object is not empty and contains the expected `user` object	The expected result of this test is that the `findByUsernameOrEmail` method of the `userRepositoryMock` object should return an `Optional` object that is not empty and contains the expected `User` object, when provided with a valid email address	Actual value returned by the method matches the expected value	P	Sarab

3	**Repository**	**UserRepository**	This test checks if the `findByUsername` method of the `userRepositoryMock` object properly returns an `Optional` object that contains the expected `User` object, when provided with a valid username	It uses Mockito's `when` method to mock the expected behavior of the `findByUsername` method, and the `assertThat` method with AssertJ's `isPresent` and `contains` methods to check if the returned `Optional` object is not empty and contains the expected `user` object.	The expected result of this test is that the `findByUsername` method of the `userRepositoryMock` object should return an `Optional` object that is not empty and contains the expected `User` object, when provided with a valid username	Actual value returned by the method matches the expected value	P	Sarab
4	**Repository**	**UserRepository**	This test checks if the `existsByUsername` method of the `userRepositoryMock` object returns `true` when provided with an existing username	It uses Mockito's `when` method to mock the expected behavior of the `existsByUsername` method, and the `assertThat` method with AssertJ's `isTrue` method to check if the returned `result` is `true`.	The expected result of this test is that the `existsByUsername` method of the `userRepositoryMock` object should return `true` when provided with an existing username.	Actual value returned by the method matches the expected value	P	Sarab

5	**Repository**	**UserRepository**	This test checks if the `existsByUsername` method of the `userRepositoryMock` object returns `false` when provided with a non-existing username	It uses Mockito's when method to mock the expected behavior of the `existsByUsername` method, and the `assertThat` method with AssertJ's `isFalse` method to check if the returned `result` is `false`.	The expected result of this test is that the `existsByUsername` method of the `userRepositoryMock` object should return `false` when provided with a non-existing username	Actual value returned by the method matches the expected value	P	Sarab
6	**Repository**	**UserRepository**	This test checks if the `existsByEmail` method of the `userRepositoryMock` object returns `true` when provided with an existing email address.	It uses Mockito's when method to mock the expected behavior of the `existsByEmail` method, and the `assertThat` method with AssertJ's `isTrue` method to check if the returned `result` is `true`.	The expected result of this test is that the `existsByEmail` method of the `userRepositoryMock` object should return `true` when provided with an existing email address	Actual value returned by the method matches the expected value	P	Sarab

7	**Repository**	**UserRepository**	This test checks if the `existsByEmail` method of the `userRepositoryMock` object returns `false` when provided with a non-existing email address.	It uses Mockito's when method to mock the expected behavior of the `existsByEmail` method, and the `assertThat` method with AssertJ's `isFalse` method to check if the returned `result` is `false`	The expected result of this test is that the `existsByEmail` method of the `userRepositoryMock` object should return `false` when provided with a non-existing email address	Actual value returned by the method matches the expected value	P	Sarab
8	**Repository**	**RoleRepository**	This test checks if the `findByName` method of the `roleRepository` object properly returns an `Optional` object that contains the expected `Role` object when provided with a valid role name	It uses Mockito's when method to mock the expected behavior of the `findByName` method, and the JUnit's `assertTrue` and `assertEquals` methods to check if the returned `Optional` object is not empty, and if the name attribute of the returned `Role` object matches the expected role name	The expected result of this test is that the `findByName` method of the `roleRepository` object should return an `Optional` object that is not empty and contains the expected `Role` object when provided with a valid role name	Actual value returned by the method matches the expected value	P	Sarab

9	**Repository**	**ProductRepository**	This test checks if the `findAll` method of the `productRepository` object properly returns a list of `Product` objects	It uses Mockito's when method to mock the expected behavior of the `findAll` method, and the JUnit's `assertEquals` method to check if the returned list of `Product` objects has the expected size (i.e., 2 in this case).	The expected result of this test is that the `findAll` method of the `productRepository` object should return a list of `Product` objects. The test is expected to pass if the returned list has the expected size, and fail if the returned list size does not match the expected size.	Actual value returned by the method matches the expected value	P	Sarab
10	**Repository**	**ProductRepository**	This test checks if the `findById` method of the `productRepository` object properly returns an `Optional` object that contains the expected `Product` object when provided with a valid product ID	It uses Mockito's when method to mock the expected behavior of the `findById` method, and the JUnit's `assertEquals` method to check if the name attribute of the returned `Product` object matches the expected product name.	The expected result of this test is that the `findById` method of the `productRepository` object should return an `Optional` object that is not empty and contains the expected `Product` object when provided with a valid product ID	Actual value returned by the method matches the expected value	P	Sarab

11	**Repository**	**ProductRepository**	This test checks if the `save` method of the `productRepository` object properly saves a `Product` object and returns the saved `Product` object.	It uses Mockito's when method to mock the expected behavior of the `save` method, and the JUnit's `assertEquals` method to check if the `sku` attribute of the saved `Product` object matches the expected SKU value.	The expected result of this test is that the `save` method of the `productRepository` object should save the provided `Product` object and return the saved `Product` object.	Actual value returned by the method matches the expected value	P	Sarab
12	**Repository**	**ProductRepository**	This test checks if the `deleteById` method of the `productRepository` object properly deletes a `Product` object with the provided ID from the database	It first creates a `Product` object with the specified attributes and sets its ID to 1, and then calls the `deleteById` method of the `productRepository` object with the ID value of 1. Finally, it uses the JUnit's `assertEquals` method to check if the number of products in the database is 0 after the deletion.	The expected result of this test is that the `deleteById` method of the `productRepository` object should properly delete a `Product` object with the provided ID from the database	Actual value returned by the method matches the expected value	P	Sarab

13	**Repository**	**ProductRepository**	This test checks if the `deleteById` method of the `productRepository` object properly throws an `EmptyResultDataAccessException` when provided with a non-existing product ID	It uses Mockito's doThrow method to mock the expected behavior of the `deleteById` method, and the JUnit's `assertThrows` method to check if the `deleteById` method call with the non-existing ID value throws the expected exception.	The expected result of this test is that the `deleteById` method of the `productRepository` object should throw an `EmptyResultDataAccessException` when provided with a non-existing product ID	Actual value returned by the method matches the expected value	P	Sarab
14	**Repository**	**SupplierRepository**	This test checks if the `findById` method of the `supplierRepository` object properly returns an `Optional` object that contains the expected `Supplier` object when provided with a valid supplier ID	It uses Mockito's when method to mock the expected behavior of the `findById` method, and the JUnit's `assertTrue` and `assertEquals` methods to check if the returned `Optional` object is not empty and the attributes of the returned `Supplier` object match the expected attribute values	The expected result of this test is that the `findById` method of the `supplierRepository` object should return an `Optional` object that is not empty and contains the expected `Supplier` object when provided with a valid supplier ID	Actual value returned by the method matches the expected value	P	Sarab

15	**Repository**	**SupplierRepository**	This test checks if the `findAll` method of the `supplierRepository` object properly returns a list of `Supplier` objects when called	It uses Mockito's `when` method to mock the expected behavior of the `findAll` method, and the JUnit's `assertNotNull`, `assertEquals`, and `get` methods to check if the returned list of `Supplier` objects is not null, has one element, and the attributes of the returned `Supplier` object match the expected attribute values.	The expected result of this test is that the `findAll` method of the `supplierRepository` object should return a non-null list of `Supplier` objects with the expected attributes when called	Actual value returned by the method matches the expected value	P	Sarab
16	**Repository**	**SupplierRepository**	This test checks if the `save` method of the `supplierRepository` object properly saves a `Supplier` object and returns the saved `Supplier` object when called	It uses Mockito's `when` method to mock the expected behavior of the `save` method, and the JUnit's `assertNotNull`, `assertEquals`, and `get` methods to check if the returned saved `Supplier` object is not null and the attributes of the saved `Supplier` object match the expected attribute values.	The expected result of this test is that the `save` method of the `supplierRepository` object should save the provided `Supplier` object and return the saved `Supplier` object when called	Actual value returned by the method matches the expected value	P	Sarab

17	**Repository**	**SupplierRepository**	This test checks if the `deleteById` method of the `supplierRepository` object properly deletes a `Supplier` object with a given ID and returns an empty `Optional` object when called	It uses Mockito's when method to mock the expected behavior of the `findById` method, and the JUnit's `assertEquals` method to check if the returned `Optional` object is empty, indicating that the `Supplier` object has been deleted	The expected result of this test is that the `deleteById` method of the `supplierRepository` object should delete the `Supplier` object with the given ID, and the subsequent call to the `findById` method of the `supplierRepository` object with the same ID should return an empty `Optional` object	Actual value returned by the method matches the expected value	P	Sarab
18	**Repository**	**ProductSupplierRepository**	This test checks if the `findById` method of the `productSupplierRepository` object properly retrieves a `ProductSupplier` object with a given ID and returns it when called	It uses Mockito's when method to mock the expected behavior of the `findById` method, and the JUnit's `assertNotNull` and `assertEquals` methods to check if the returned `ProductSupplier` object is not null and its ID attribute matches the expected ID value, respectively	The expected result of this test is that the `findById` method of the `productSupplierRepository` object should retrieve the `ProductSupplier` object with the given ID and return it	Actual value returned by the method matches the expected value	P	Sarab

19	**Repository**	**ProductSupplierRepository**	This test is checking whether the `productSupplierRepository` returns the expected list of `ProductSupplier` objects when the `findAllByProductId` method is called with a product ID as input.	This test ensures that calling the `productSupplierRepository.findById()` method with a product ID parameter returns a list of `ProductSupplier` objects. The test creates a mock `ProductSupplier` object and confirms that the returned list has the correct values for its fields.	The expected result of this test is to confirm that `productSupplierRepository.findById()` returns a list of `ProductSupplier` objects with the correct values for their fields.	Actual value returned by the method matches the expected value	P	Sarab
20	**Repository**	**ProductSupplierRepository**	This test verifies that when a ProductSupplier object is saved, it is correctly persisted in the repository, by checking that the saved object is not null and is equal to the original object.	The test condition is to verify that when a new ProductSupplier instance is saved using the `productSupplierRepository`, it is successfully persisted in the database and the saved instance matches the original instance.	The expected result is that the ProductSupplier object is successfully persisted and that the saved object is the same as the original object.	Actual value returned by the method matches the expected value	P	Sarab

21	**Repository**	**ProductSupplierRepository**	This test verifies that a call to delete a ProductSupplier by id will result in the ProductSupplier being deleted.	The test condition is to delete a product supplier with the specified ID and verify that the corresponding method in the repository is called exactly once with that ID.	The expected result is that the `deleteById` method of the `productSupplierRepository` should be called once with the argument 1L.	Actual value returned by the method matches the expected value	P	Sarab
22	**Repository**	**EnquiryRepository**	This test is checking whether a list of enquiries with a specific ID is returned correctly. It verifies that the list is not null or empty, has a size of 1, and that the attributes of the enquiry match the expected values.	The test is checking if the `enquiryRepository` can find a list of `EnquiryObj` objects by the provided id (in this case 1L).	The expected result is a list of one EnquiryObj with the specified details, which is returned by the `enquiryRepository.findAllById(1L)` method call. The test checks that the list is not null or empty, has a size of 1, and that the details of the EnquiryObj in the list match the expected values.	Actual value returned by the method matches the expected value	P	Sarab

23	**Controller**	**AuthController**	This test for the getUserProfile() method of the authController class. The test mocks the Authentication and User objects, and verifies that when a valid User object is passed to the method, it returns an HTTP 200 OK response with the same User object as the response body.	The test conditions are that a mock authentication is created with a mocked user object and the email is set to "test@example.com". The user is then retrieved from the mock user repository and the authentication is set in the SecurityContextHolder. The getUserProfile method of the authController is then called passing the user object as an argument.	The expected result is that the test should check if the getUserProfile method in the AuthController returns a ResponseEntity with an HTTP status of 200 (OK) and the user object retrieved by the userRepository.findByEmail() method as its body.	Actual value returned by the method matches the expected value	P	Sarab
24	**Controller**	**AuthController**	The getUserByIdTest test case is checking whether the user object returned by the controller method getUserById() is the same as the user object that was previously saved with the same ID value. The test expects that the HTTP status code returned by the method is 201 CREATED.	The test is checking the behavior of the getUserById() method of the authController object when passed a user ID of 1L. The User object with ID 1L is mocked and returned by the userRepository when its findById() method is called with an argument of 1L.	The test is expecting that calling the getUserById method with a user ID of 1 should return a ResponseEntity object with an HTTP status code of 201 (CREATED) and a response body containing the User object with ID 1.	Actual value returned by the method matches the expected value	P	Sarab

25	**Controller**	**AuthController**	This is a unit test for the `login` method in the `AuthController` class. It creates a `LoginDto` object with test data, mocks the response of the `login` method from the `AuthService` to return a JWT token, and then calls the `login` method of the `AuthController`	The test condition is that a `LoginDto` object is created with a username or email of "user" and a password of "password". The `authService.login()` method is mocked to return a JWT token. The expected result is that the HTTP response status code is `HttpStatus.OK` (200), and the `JWTAuthResponse` object returned in the response body contains the expected JWT token.	The expected result of the test is that the login method of the `authController` should return a response with HTTP status code 200 (OK) and a JWT token in the body. The token should be equal to "jwt_token" which is set up as the expected return value of the `authService.login()` method call.	Actual value returned by the method matches the expected value	P	Sarab
26	**Controller**	**AuthController**	This test checks if the register() method of the authController successfully returns a HTTP 201 CREATED status and a message "User registered" when given a RegisterDto object containing a username, password, and email.	The test condition is registering a new user with a username, password and email.	The expected result is that the user is successfully registered and the HTTP response status code is 201 (Created), and the response body is "User registered".	Actual value returned by the method matches the expected value	P	Sarab

27	**Controller**	**AuthController**	This test is checking the functionality of the `handleConstraintViolationException()` method in the `authController` class. It sets up a mock `ConstraintViolationException` object, adds a mock `ConstraintViolation` object to it, and expects to receive a `ResponseEntity` object with a specific HTTP status code and an empty value for a specific key in the response body map. The purpose of this test is to ensure that the method correctly handles constraint violation exceptions and returns an appropriate response.	The test condition is when a `ConstraintViolationException` is thrown with a set of constraint violations.	The expected result is that the response status code is 400 (Bad Request) and the response body contains an empty key-value pair.	Actual value returned by the method matches the expected value	P	Sarab
28	**Controller**	**AuthController**	This test is checking if a user can be deleted from the system by its ID. The user with ID 1 is created and its data is stored. The test ensures that the HTTP response status code for a successful operation is "OK" (200).	The test condition is to delete the user with the given ID (1L) from the database.	The expected result is that the status code of the response entity is HttpStatus.OK, indicating that the user has been successfully deleted.	Actual value returned by the method matches the expected value	P	Sarab

29	**Controller**	**AuthController**	This is a unit test case for the updateUserById() method of a class. The test checks whether the method returns a ResponseEntity object with HTTP status code 200 (OK) and the expected message "User updated successfully" when a valid user ID and updateDto object are passed as arguments.	The test condition is to update a user with the given ID (1L) with the given UpdateDto object, and to mock the result of the update as "User updated successfully".	The test expects that the updateUserById method in authService returns a success message and that the returned ResponseEntity has a status of OK and the same success message in its body.	Actual value returned by the method matches the expected value	P	Sarab
30	**Controller**	**AuthController**	The testUpdateUserByIdFailure() method tests the scenario when an error is thrown by authService.updateUserById() with the "User not found" message. The expected result is an HTTP 404 (not found) status and a null response body.	The test is checking the scenario where the authService.updateUserById() method throws an EcommerceAPIException with status code NOT_FOUND and message User not found. The authController.updateUserById() method is called with a valid UpdateDto and the user id of 1L.	The expected result is that the test will check if the method updateUserById in the authService throws an EcommerceAPIException with a HttpStatus of NOT_FOUND, the authController will return an HTTP response with a status of NOT_FOUND and the body will be null.	Actual value returned by the method matches the expected value	P	Sarab

31	**Controller**	**AuthController**	This test is testing the update user functionality in the authController class. The test calls the updateUser() method in the authController with a sample UpdateDto object and then checks whether the response from the service method is as expected.	The test condition of this test is that the authService.updateUser method should be called with an UpdateDto object and return a success message "User updated successfully".	The test expects an HTTP status code of 200 OK and a message "User updated successfully" as the response body.	Actual value returned by the method matches the expected value	P	Sarab
32	**Controller**	**AuthController**	This is a JUnit test for the updateUser method in an API controller class. The test checks that if the updateUser method in the service layer throws an EcommerceAPIException, then the controller method returns an HTTP 500 error status and a null response body.	The test condition of this test is to check if the update fails due to an exception, the controller should return an appropriate HTTP status code and a null response body.	The expected result of this test is that the HttpStatus of the response should be INTERNAL_SERVER_ERROR and the body of the response should be null. This is because the authService.updateUser() method is intentionally throwing an EcommerceAPIException with a HttpStatus of INTERNAL_SERVER_ERROR.	Actual value returned by the method matches the expected value	P	Sarab

33	**Controller**	**EnquiryController**	This test checks if the `enquiryController` correctly retrieves all enquiries from the database using the `enquiryRepository.findAll()` method	The test condition is checking if the controller method `getAllEnquiries` returns a list of enquiries with size 1. It is using a mocked `enquiryRepository` that will return a list of one enquiry object.	The expected result is that the test should pass if the number of returned EnquiryObj is 1, indicating that the EnquiryRepository is able to retrieve all existing enquiries.	Actual value returned by the method matches the expected value	P	Sarab
34	**Controller**	**EnquiryController**	This is a test for the `getUsersEnquiries` method of an `EnquiryController` class. It creates a list of `EnquiryObj` objects and mocks the `enquiryRepository.findAllById` method to return this list when passed a specific `id`. It then calls `getUsersEnquiries` with the same `id` and asserts that the returned status code is 200 and the returned list of `EnquiryObj` objects is equal to the original list.	The test condition is to check if the method `getUsersEnquiries` of `enquiryController` returns a list of enquiries for a given user id. It creates two `EnquiryObj` objects and adds them to a list of enquiries. Then, it mocks the `enquiryRepository.findAllById(id)` method to return the list of enquiries. Finally, it calls the `getUsersEnquiries` method of the `enquiryController` with the given user id and checks if it returns a response with a status code of 200 and the list of enquiries as the body.	The expected result is a `ResponseEntity` object containing an HTTP status code of 200 and a list of `EnquiryObj` objects retrieved from the repository. The retrieved list should match the expected list of two `EnquiryObj` objects created in the test.	Actual value returned by the method matches the expected value	P	Sarab

35	**Controller**	**EnquiryController**	This test case tests the `createEnquiry` method of the `EnquiryController` class. It creates an `EnquiryObj` object and sets it up for testing. The `enquiryRepository.save()` method is mocked to return the `enquiry` object as it is. Then the `createEnquiry()` method is called and the returned `EnquiryObj` is checked for `not null`.	The test condition is to create a new `EnquiryObj` and verify that it is saved successfully by the `EnquiryRepository`.	The expected result is that a new `EnquiryObj` is created and returned by the `enquiryController.createEnquiry(enquiry)` method call. The test is checking if the createdEnquiry is not null, which indicates that the `enquiry` object was successfully saved to the database by calling `enquiryRepository.save(enquiry)`.	Actual value returned by the method matches the expected value	P	Sarab
36	**Controller**	**EnquiryController**	This test tests the `updateEnquiry` method of the `EnquiryController` class. It creates an `EnquiryObj` instance, sets its properties, mocks the `findById` and `save` methods of the `EnquiryRepository` interface, and then calls the `updateEnquiry` method with the created `EnquiryObj` and `id`. It checks that the returned `EnquiryObj` is not null, and that its properties match the original `EnquiryObj`.	The test condition of this test is to update an Enquiry object with the given `id` using the `updateEnquiry()` method of the `EnquiryController` class with the provided `EnquiryObj` and check that the updated Enquiry object returned from the method is not null, and that it has the same values for `orderId`, `name`, `email`, and `description` as the provided `EnquiryObj`	The expected result is that the `enquiryController.updateEnquiry` method should successfully update an existing `EnquiryObj` with the given `id` using the data in the `enquiry` object, and the returned `updatedEnquiry` should have the same property values as the `enquiry` object	Actual value returned by the method matches the expected value	P	Sarab

37	**Controller**	**EnquiryController**	This test is for testing the `deleteEnquiry` method in the `enquiryController`. It sets up an `EnquiryObj` and mocks a call to `enquiryRepository.findById()` with the ID of the enquiry. It then calls `enquiryController.deleteEnquiry()` with the same ID and expects a non-null `ResponseEntity` with a status code of 200 OK. The test is ensuring that the `deleteEnquiry()` method can successfully delete an enquiry with the given ID.	The test condition of this `testDeleteEnquiry()` method is that there is an enquiry object with the specified ID in the repository.	The test is expected to assert that a successful response with HTTP status code 200 is returned when deleting an enquiry with the given ID. The actual response is checked to make sure it is not null and has a status code of 200.	Actual value returned by the method matches the expected value	P	Sarab
38	**Controller**	**OrderController**	This test is testing the `findAllFromEmail` method of the `OrderController` class. It sets up a mock repository using Mockito to return an empty list of orders when queried with a specific email. It then calls `findAllFromEmail` on an instance of `OrderController` with the same email and asserts that the returned list is equal to the mocked empty list.	The test condition is to call the `findAllFromEmail` method of `orderController` with an email address `test@example.com` and check if it returns a list of orders with an empty list.	The expected result is that the method `findAllFromEmail` of the `orderController` should return a list of `OrderObj` objects, and the list should be equal to the list returned by the `orderRepo.findAllByEmail` method when called with the `email` parameter.	Actual value returned by the method matches the expected value	P	Sarab

39	**Controller**	**OrderController**	This JUnit test case tests the `findAll()` method in the `OrderController` class. It creates an empty list of `OrderObj`, sets up a mock repository to return this list when the `findAll()` method is called, and then calls the `findAll()` method on the `OrderController` instance. Finally, it asserts that the returned list is equal to the original empty list.	The test condition for this test is that the `orderRepo` should return an empty list of `OrderObj`.	The test is checking that when the `findAll()` method of the `orderRepo` is called, it should return an empty list, and the `findAll()` method of the `orderController` should return the same empty list	Actual value returned by the method matches the expected value	P	Sarab
40	**Controller**	**OrderController**	This test is testing the `createOrderObject` method in the `OrderController`. The method takes an `OrderObj` object and a list of `ProductOrderObj` objects as input, and returns a string indicating the status of the operation	The test condition for this unit test is that an OrderObj object and a List of ProductObj objects are passed to the `orderController.createOrderObject` method as parameters. The `orderRepo.save` method is mocked to return an OrderObj object and the `orderRepo.getOrderId` method is mocked to return 1. The `prodOrderRepo.save` method is mocked to return a new ProductOrderObj object.	The expected result is that the `createOrderObject` method in the `OrderController` class should update the orders table, create one row in the product_orders table for each product in the productList, and return a string indicating that the tables were updated	Actual value returned by the method matches the expected value	P	Sarab

41	**Controller**	**ProductController**	This is a test case to check if the `getAllProducts` method of the `ProductController` class returns all the products in the database	The test condition of this test is to check if the `getAllProducts` method of the `ProductController` class returns a list of `Product` objects from the repository correctly.	The test condition is that the `repository.findAll()` method returns a list containing two `Product` objects. The test is checking if the `productController.getAllProducts()` method returns a list of `Product` objects with the correct size and names. The expected result is that the `result` list should have a size of 2 and the names of the products in the list should be "product1" and "product2".	Actual value returned by the method matches the expected value	P	Sarab
42	**Controller**	**ProductController**	This test checks if the `createProduct()` method of the `ProductController` correctly creates a new product in the database by verifying if the `name` of the created product is the expected one	The test condition for this test is not explicitly given in the code, but it can be inferred from the code that a new product instance `product1` is created and passed to the `createProduct` method of the `productController`. Then we check we return the same instance of the product1	The expected result is that the `repository.save` method is called with the product and that the `createProduct` method returns the same product with name "product1".	Actual value returned by the method matches the expected value	P	Sarab

43	**Controller**	**ProductController**	The test condition is that the product with ID 1 exists in the repository and has the name "product1". The expected result is that the `getProduct()` method should return a `Product` object with the same ID and name.	The test condition of this test is that there is a product with id 1L in the repository and it can be retrieved by calling getProduct method with id 1L.	The test is verifying that when the method `getProduct` is called with an ID of 1, it returns a `Product` object with a name of "product1".	Actual value returned by the method matches the expected value	P	Sarab
44	**Controller**	**ProductController**	The test is checking whether the method successfully updates a product with the given ID. The test checks whether the updated product returned by the method has the same name as the original product.	The test condition for this test is that the repository should be able to find a product by ID, and when it does, the updated product should be returned	The expected result is that the updated `Product` object returned from the `updateProduct` method has the same name as `product1`	Actual value returned by the method matches the expected value	P	Sarab

45	**Controller**	**ProductController**	This test checks the deleteProduct() method in the product controller, which should return an HTTP 200 status code when given an existing product ID.	The test condition is to call the `deleteProduct()` method with a product ID of 1, and to have the `findById()` method of the repository return an optional containing `product1`.	The expected result is that the status code of the response entity should be 200.	Actual value returned by the method matches the expected value	P	Sarab
46	**Controller**	**ProductSupplierController**	The test is ensuring that when the `findAll()` method of the `productSupplierRepository` object is called, it returns a list containing one element of `ProductSupplier`. It then calls the `getAllProductSuppliers()` method and verifies that the returned list contains one element.	The test condition for this code is that the `productSupplierController` should return a list of `ProductSupplier` objects containing at least one element when the `getAllProductSuppliers` method is called.	The expected result is that the `productSuppliers` list returned by `productSupplierController.getAllProductSuppliers()` should have a size of 1	Actual value returned by the method matches the expected value	P	Sarab

47	**Controller**	**ProductSupplierController**	The test checks if creating a new product supplier object through the controller results in a non-null object.	The test condition for the `testCreateProductSupplier` is that a `ProductSupplier` object is created, and the `productSupplierRepository.save()` method is called on this object. The method returns the same object, and the created object is not null.	The expected result is for the `createdProductSupplier` object returned from the `createProductSupplier` method to not be null.	Actual value returned by the method matches the expected value	P	Sarab
48	**Controller**	**ProductSupplierController**	This test checks if a product supplier can be retrieved by ID. The expected result is that the product supplier is successfully retrieved.	The test condition for the given code is to call the `getProductSupplierById` method on the `productSupplierController` object with a Long ID of 1L. The `productSupplierRepository` is mocked to return an `Optional` containing a `ProductSupplier` object when the `findById` method is called with the specified ID.	The expected result is that `retrievedProductSupplier` should be present (not null) after calling the `getProductSupplierById` method on `productSupplierController` with the given `id`.	Actual value returned by the method matches the expected value	P	Sarab

49	**Controller**	**ProductSupplierController**	This is a test to update a `ProductSupplier` object. It sets the lead time and cost price of the product supplier and verifies that the updated product supplier object has the same values.	This test checks if a product supplier can be successfully updated with new information in the system. The test sets a new lead time and cost price for the product supplier, and then checks that the updated product supplier has these values.	The test is verifying that the updated product supplier object has the same lead time and cost price as the original product supplier object.	Actual value returned by the method matches the expected value	P	Sarab
50	**Controller**	**ProductSupplierController**	This is a test to verify that the `deleteProductSupplier` method in the `ProductSupplierController` deletes a product supplier with the given ID by calling the `deleteById` method of the `productSupplierRepository` once.	The test condition for this test is that a product supplier with ID `id` exists in the system.	The expected result is that the `deleteProductSupplier` method of the `productSupplierController` should call the `deleteById` method of the `productSupplierRepository` once with the id passed to it as an argument.	Actual value returned by the method matches the expected value	P	Sarab

51	**Controller**	**SupplierController**	This is a test to verify that the `getAllSuppliers()` method returns a list of `Supplier` objects from a mocked repository. The expected result is that the size of the returned list is 1	The test condition for the `testGetAllSuppliers` test is that the `findAll()` method of the `supplierRepository` object should be called.	The expected result is that the returned list of suppliers has a size of 1.	Actual value returned by the method matches the expected value	P	Sarab
52	**Controller**	**SupplierController**	Test creates a supplier and verifies that the created supplier is not null.	The test condition for this test is that the `supplierRepository` should return a list of one `Supplier` object when the `findAll` method is called.	The expected result is that the size of the list returned by the `getAllSuppliers` method of `supplierController` should be 1.	Actual value returned by the method matches the expected value	P	Sarab

53	**Controller**	**SupplierController**	Update a supplier and assert the returned values	The test condition for `testUpdateSupplier` is to pass a supplier object with updated values for name, location, email, and phone and an ID to `supplierController.updateSupplier` method. The `findById` method of the `supplierRepository` should return an Optional of a new `Supplier` object. Finally, the `save` method of the `supplierRepository` should return the updated `Supplier` object.	The expected values for the updated `Supplier` object are the same as the ones that were set in the `supplier` object used for the update.	Actual value returned by the method matches the expected value	P	Sarab
54	**Controller**	**SupplierController**	This is a unit test that verifies whether a `deleteSupplier()` method of a `SupplierController` class works correctly. The test ensures that a supplier with a given id is deleted when the method is called by verifying that the `deleteById` method of the `supplierRepository` is called exactly once with the same id.	The test condition is to delete a supplier with the given ID using the `deleteSupplier()` method in `supplierController`.	Verifies if the `deleteSupplier` method of the `supplierController` class calls the `deleteById` method of the `supplierRepository`	Actual value returned by the method matches the expected value	P	Sarab

55	**Service**	**AuthServiceImpl**	This test checks the successful login scenario where the user provides the correct username and password, and the authentication process is successful. It expects that the correct JWT token is returned to the user upon successful authentication.	The test condition is checking the success of the login process by calling the `authService.login()` method with valid login credentials `usernameOrEmail` and `password` and checking if the method returns a valid access token `token`. It also mocks the authentication and token generation processes to simulate a successful login.	The expected result is that `authService.login(loginDto)` should return a String token. In this test case, the `jwtTokenProvider.generateToken(authentication)` method is mocked to return "token", which is asserted to be the same as the result of `authService.login(loginDto)`.	Actual value returned by the method matches the expected value	P	Sarab
56	**Service**	**AuthServiceImpl**	This test is for the register method of the AuthService. It tests if the method returns the success message when valid input is given, and if a new user is saved to the UserRepository.	The test condition is when the input is valid. The input is provided as a `RegisterDto` object which has a name, username, email, and password. The test verifies that the user with the provided username and email does not already exist and that the provided password is encoded before saving the user to the repository. The test checks that the returned message is "User registered successfully!" and that the user is saved to the repository.	The expected result is that the test should return a success message "User registered successfully!." after registering a new user with valid input, and the user should be saved in the UserRepository.	Actual value returned by the method matches the expected value	P	Sarab

57	**Service**	**AuthServiceImpl**	This test checks if the `register` method in `authService` throws an exception when a user with the same username already exists in the system.	The test condition is when a username already exists in the user repository.	The test expects that the `register()` method of the `authService` object will throw an `EcommerceAPIException` with a message "Username is already exists!." if the provided username already exists in the database.	Actual value returned by the method matches the expected value	P	Sarab
58	**Service**	**AuthServiceImpl**	This test checks if the `register` method in `authService` throws an exception when a user with the same email already exists in the system.	The test condition is when a email already exists in the user repository.	The test expects that the `register()` method of the `authService` object will throw an `EcommerceAPIException` with a message "Email is already exists!." if the provided email already exists in the database.	Actual value returned by the method matches the expected value	P	Sarab

59	**Service**	**AuthServiceImpl**	This is a test that verifies that when the input validation fails, an exception is thrown by the `authService.register()` method	The test condition for this test is when the validation fails. The input for the `registerDto` has an empty name which is an invalid input. The `validator.validate()` method is mocked to return a `Set` of constraint violations to simulate a failed validation.	The test expects that an instance of `ConstraintViolationException` is thrown when the validation fails for the `RegisterDto` object.	Actual value returned by the method matches the expected value	P	Sarab
60	**Service**	**AuthServiceImpl**	This is a test for the `updateUserById` method of the `AuthService` class. The test checks if the method updates the user with the given ID correctly	The test sets up an `UpdateDto` object with new user details, mocks the `UserRepository` to return a user with the given user ID, and mocks the `PasswordEncoder` to return an encoded password. The test then calls the `updateUserById` method with the user ID and the `UpdateDto` object, verifies that the updated user is saved using the `UserRepository`, and asserts that the method returns the expected success message.	The expected result is that the user is successfully updated and the message "User saved successfully!" is returned by the `updateUserById` method. The test is verifying that the user data is properly updated and saved by the repository.	Actual value returned by the method matches the expected value	P	Sarab

61	**Service**	**AuthServiceImpl**	This test is checking if the user can be successfully updated	The test condition for this test is that a valid updateDto is passed to the update user service method, and the email of the user is found in the UserRepository. A new encoded password is generated and the user is saved with the updated information. Lastly, the security context is set and the returned message is checked.	The expected result is that the method should return the string "User saved successfully!" after updating the user with the given UpdateDto object.	Actual value returned by the method matches the expected value	P	Sarab
62	**Util**	**PasswordGeneratorEncoder**	This test sets a plain text password, encodes it, and then verifies that the encoded password matches the plain text password	This test is checking that a password can be encoded and successfully matched to its original unencoded value using the `BCryptPasswordEncoder` from Spring Security.	The expected result is that the encoded password matches the original password when passed to the `matches` method of the password encoder.	Actual value returned by the method matches the expected value	P	Sarab

63	**Util**	**PasswordGeneratorEncoder**	This test case tests the `encode` method of the password encoder. It sets up a raw password string and a corresponding encoded password string, and then mocks the `encode` method of the password encoder to return the encoded password when called with the raw password. Finally, the test case invokes the `encode` method with the raw password and verifies that it returns the expected encoded password.	The test condition is that `rawPassword` is "sarab" and `encodedPassword` is "$2a$10$KBuCBRGW.CQT26GcM0tnVeLa9X9B8HIGhjJblHX2gnoz1bvnC6mb2". The mock `passwordEncoder.encode(rawPassword)` is expected to return `encodedPassword`.	The expected result of the test is the `encodedPassword` which is the result of encoding the `rawPassword` using the `passwordEncoder`. The test verifies that the `passwordEncoder` is working correctly by ensuring that it returns the expected encoded password.	Actual value returned by the method matches the expected value	P	Sarab
64	**Exception**	**EcommerceAPIException**	This test is testing the `EcommerceAPIException` class. It sets a message and HTTP status code, creates an instance of `EcommerceAPIException` using those values, and checks that the message and status code are set correctly in the instance.	The test is checking that the `EcommerceAPIException` class correctly sets its status and message fields when an instance of the class is created.	The expected result of the test is to verify that the `EcommerceAPIException` class correctly initializes with the provided status and message, and that its getter methods return the expected values.	Actual value returned by the method matches the expected value	P	Sarab

65	**Exception**	**GloablExceptionHandler**	It tests if the method returns a ResponseEntity with status code HttpStatus.NOT_FOUND when it is passed a ResourceNotFoundException, and if the returned body is null.	The test condition for this test is when a ResourceNotFoundException is thrown.	The test condition is that a ResourceNotFoundException is thrown. The expected result is that the method handleResourceNotFoundException of the globalExceptionHandler returns a ResponseEntity object with a status code of HttpStatus.NOT_FOUND and a null message.	Actual value returned by the method matches the expected value	P	Sarab
66	**Exception**	**GloablExceptionHandler**	This test ensures that the GlobalExceptionHandler returns a HTTP Bad Request status when handling a EcommerceAPIException, and that the response message contains the error message.	EcommerceAPIException with message "Bad request" and HTTP status code HttpStatus.BAD_REQUEST.	A ResponseEntity object with HTTP status code HttpStatus.BAD_REQUEST and the message "Some error" in the response body.	Actual value returned by the method matches the expected value	P	Sarab

67	**Exception**	**GloablExceptionHandler**	This is a unit test for the `handleGlobalException` method in the `GlobalExceptionHandler` class. The test verifies that if the method receives an `Exception` object, it returns an HTTP status code of 500 (internal server error) and an `ErrorDetails` object with the exception's message.	An exception is thrown with the message "Internal server error".	The global exception handler should return a response entity with the HTTP status code 500 (Internal Server Error) and the message "Internal server error".	Actual value returned by the method matches the expected value	P	Sarab
68	**Exception**	**GloablExceptionHandler**	This test checks that the `handleAccessDeniedException` method in the `GlobalExceptionHandler` class returns an UNAUTHORIZED HTTP status code and the expected error message when an `AccessDeniedException` is thrown.	An AccessDeniedException is thrown with the message "Unauthorized".	The globalExceptionHandler should catch the exception and return a ResponseEntity with an HTTP status code of 401 (Unauthorized) and a message of "Unauthorized".	Actual value returned by the method matches the expected value	P	Sarab

69	**Exception**	**GloablExceptionHandler**	This test checks if the `handleMethodArgumentNotValid` method in the `GlobalExceptionHandler` class correctly handles `MethodArgumentNotValidException` exceptions by returning a map of field names and error messages.	The test condition is when a method argument is not valid, and there is a field error with the message "message"	The expected result is that the global exception handler should return a map with a key-value pair of the field name and error message, as well as an HTTP status of 400 Bad Request.	Actual value returned by the method matches the expected value	P	Sarab
70	**Exception**	**ResourceNotFoundException**	This test checks if the constructor of the ResourceNotFoundException class works correctly by passing in the resource name, field name, and field value as parameters. It verifies that the constructed exception message contains the expected field value and that the resource name, field name, and field value are properly set as instance variables.	The test checks if the `ResourceNotFoundException` constructor sets the message, resource name, field name, and field value correctly.	The exception message should be "Post not found with id : '1'"	Actual value returned by the method matches the expected value	P	Sarab

71	**Exception**	**ResourceNotFoundException**	This test checks the behavior of the `ResourceNotFoundException` constructor when only the resource name is provided. The expected behavior is that the message and resource name properties of the exception are both set to null.	The test condition is creating a new `ResourceNotFoundException` object with a resource name "Post".	The expected result is that the `message` and `resourceName` fields of the exception object are null.	Actual value returned by the method matches the expected value	P	Sarab
72	**Exception**	**ResourceNotFoundException**	his test is testing the response status of a custom exception `ResourceNotFoundException` in the `getMessage()` method.	The test condition is that a `ResourceNotFoundException` is created with resource name `Post` and field name `id` with a field value of `1`.	The expected result is that the `getMessage()` method will return a string "Post not found with id : '1'".	Actual value returned by the method matches the expected value	P	Sarab

73	**Config**	**SecurityConfig**	This is a unit test for a password encoder	The test condition is a plain text password "test123"	The expected result is that the password should be correctly encoded using the password encoder and should match the encoded password	Actual value returned by the method matches the expected value	P	Sarab
74	**Config**	**SecurityConfig**	This is a test for the `authenticationManager` method in a Spring Security configuration.	The test condition is that a mock `AuthenticationConfiguration` and `AuthenticationManager` are created	The expected result is that the `authenticationManager` method returns the same `AuthenticationManager` object that was passed into the mock `AuthenticationConfiguration`.	Actual value returned by the method matches the expected value	P	Sarab

75	**Config**	**SecurityConfig**	This is a test case checks if a request to a protected endpoint is correctly authenticated. It uses the `@WithMockUser` annotation to create a mock user with a specific username, password, and role. The test sends a GET request to the endpoint and verifies that the request was successful and returns a status code of 200 (OK).	This test verifies that the authentication works correctly by making a GET request to a protected resource ("/api/v1/test") using a mocked user with the "USER" role	The expected result is that the request is successfully authenticated and the response returns the expected data for the protected resource.	Actual value returned by the method matches the expected value	P	Sarab
76	**Security**	**CustomUserDetailService**	This is a unit test for the `loadUserByUsername` method of the `UserDetailsServiceImpl` class. The test sets up a mock `UserRepository` to return a user object for a given email address. It then calls the `loadUserByUsername` method of the `UserDetailsServiceImpl` and asserts that the returned `UserDetails` object has the correct username and password values from the mocked user object, as well as the correct role assigned to it.	The test condition is the existence of a user with the specified email address in the mock `UserRepository`	The expected result is that the `UserDetails` object returned by the `loadUserByUsername` method has the correct username, password, and role values.	Actual value returned by the method matches the expected value	P	Sarab

77	**Security**	**CustomUserDetailService**	This test is for the `loadUserByUsername` method in the `UserDetailsServiceImpl` class.	The test condition is when a user with a specific username or email exists in the database, and the expected result is that the user details are loaded correctly. The test asserts that the email and password of the user are correct and that the user has the expected role	The expected result is that a `UsernameNotFoundException` is thrown with the appropriate message.	Actual value returned by the method matches the expected value	P	Sarab
78	**Security**	**JwtAuthenticationEntryPoint**	This test checks that the JwtAuthenticationEntryPoint class properly sends an HTTP error response with the "Unauthorized" message when an authentication error occurs. It sets up mock objects for the HttpServletRequest, HttpServletResponse, and AuthenticationException, and uses them to test the behavior of the commence() method of the JwtAuthenticationEntryPoint class. The test verifies that the servletResponse object receives a call to the sendError() method with the appropriate HTTP status code and error message.	The test condition is that an `HttpServletRequest` and an `HttpServletResponse` are mocked, with the former having attributes for an error status code of 401 and an error message of "Unauthorized". An `AuthenticationException` is also mocked.	The expected result is that the `sendError` method of the `HttpServletResponse` should be called with the `SC_UNAUTHORIZED` status code and the message of the `AuthenticationException`.	Actual value returned by the method matches the expected value	P	Sarab

79	**Security**	**JwtAuthenticationFilter**	The `testDoFilterInternal` test checks whether the `JwtAuthenticationFilter` correctly loads a user based on a valid JWT token contained in the `Authorization` header of an incoming HTTP request. The test mocks the `jwtTokenProvider` and `userDetailsService`, and verifies that the `loadUserByUsername` method is called with the expected username, and that the `filterChain` is executed after the user has been loaded.	The test condition for this test is a valid JWT token passed in the Authorization header of the request	The expected result is that the filter correctly extracts the token from the header, validates it, and retrieves the associated user details from the userDetailsService, before allowing the request to continue down the filter chain	Actual value returned by the method matches the expected value	P	Sarab
80	**Security**	**JwtAuthenticationFilter**	This is a unit test for a method named `testDoFilterInternalWithInvalidToken` that tests the behavior of a JWT authentication filter when an invalid token is used	The test condition is an invalid token, and the expected result is that the `userDetailsService` should not be called, and the `filterChain` should be called to proceed with the request.	The expected result is that if the provided token is invalid, the `jwtAuthenticationFilter` should not interact with the `userDetailsService` and should simply let the request pass through to the next filter in the chain	Actual value returned by the method matches the expected value	P	Sarab

81	**Security**	**JwtAuthenticationFilter**	This is a unit test for the `doFilterInternal()` method of the `JwtAuthenticationFilter` class. The test is checking the behavior when the request does not contain a token.	The test condition is when the request does not contain an authorization header, i.e., when `request.getHeader("Authorization")` returns `null`.	The expected result is that `jwtTokenProvider` and `userDetailsService` should not be invoked and the `filterChain` should continue the filter chain. The test verifies these conditions using the `verifyNoInteractions()` and `verify()` methods from Mockito.	Actual value returned by the method matches the expected value	P	Sarab
82	**Security**	**JwtTokenProvider**	This is a unit test for the `generateToken()` method of the `JwtTokenProvider` class. The test generates a token using a mock `Authentication` object, and then checks that the token is not null and has a length greater than 0. The test is checking that the `generateToken()` method returns a valid JWT token.	An instance of JwtTokenProvider and an instance of Authentication are created.	The generated token should not be null and should have a length greater than zero	Actual value returned by the method matches the expected value	P	Sarab

83	**Security**	**JwtTokenProvider**	This is a unit test for the `getUsername()` method of a `JwtTokenProvider` class. The test generates a JWT token using an authentication object, then calls the `getUsername()` method of the token provider to retrieve the username from the generated token. The test checks if the retrieved username is equal to the name of the user in the authentication object.	The test condition is that a valid token is generated from the `generateToken()` method using an `authentication` object.	The expected result is that the extracted username should be equal to the username in the provided `authentication` object.	Actual value returned by the method matches the expected value	P	Sarab
84	**Security**	**JwtTokenProvider**	This is a test that checks if the `validateToken()` method in the `JwtTokenProvider` class is working as expected. It generates a JWT token using the `generateToken()` method with a given authentication object and then checks if the token is valid by calling the `validateToken()` method.	The test is checking whether the generated token is valid or not	The expected result is that the `validateToken()` method should return `true`, indicating that the token is valid.	Actual value returned by the method matches the expected value	P	Sarab

85	**Security**	**JwtTokenProvider**	This is a unit test for the `validateToken` method of the `JwtTokenProvider` class, which is expected to throw an `EcommerceAPIException` when an invalid token is passed as an argument	The test condition is an invalid token passed as a string to the `validateToken` method	The expected result is that an `EcommerceAPIException` should be thrown.	Actual value returned by the method matches the expected value	P	Sarab
86	**Security**	**JwtTokenProvider**	In this test, the `jwtTokenProvider` is tested to throw an `EcommerceAPIException` when validating an expired token. To simulate an expired token, a token with an expiration date set to 7 days ago is generated using `Jwts.builder()` method, then this token is passed to `jwtTokenProvider.validateToken()` method which is expected to throw an exception.	Generating a JWT token with an expired date.	The test is expected to throw an `EcommerceAPIException` because the token is expired.	Actual value returned by the method matches the expected value	P	Sarab