

Web API Design with Spring Boot Week 16 Coding Assignment


Points possible: 75

URL to GitHub Repository: https://github.com/CamiGrace/Springboot_JeepSales


URL to Public Link of your Video: <https://www.youtube.com/watch?v=84gyZEfAOPM>

Instructions :

1. Follow the **Coding Steps** below to complete this assignment.

- In Spring Tool Suite (STS), or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed.
- Use your existing repo or create a new repository on GitHub for this week's assignment and push your completed code to the repo, including your entire Maven Project Directory (e.g., jeep-sales) and any additional files (e.g. .sql files) that you create. In addition, screenshot your ERD and push the screenshot to your GitHub repo.
- Include the screenshots into this Assignment Document indicated by: 
- Create a video showcasing your work:
 - In this video: record and present your project verbally while showing the results of the working project.
 - Easy way to Create a video: Start a meeting in Zoom, share your screen, open Eclipse with the code and your Console window, start recording & record yourself describing and running the program showing the results.
 - Your video should be a maximum of 5 minutes.
 - Upload your video with a public link.
 - Easy way to Create a Public Video Link: Upload your video recording to YouTube with a public link.


2. In addition, please include the following in your Coding Assignment Document:

- The requested screenshots, indicated by: 
- The URL for this week's GitHub repository.
- The URL of the public link of your video.

3. Save the Coding Assignment Document as a .pdf and do the following:

- Push the .pdf to the GitHub repo for this week.
- Upload the .pdf to the LMS in your Coding Assignment Submission.

Web API Design with Spring Boot Week 16 Coding Assignment

Here's a friendly tip: as you watch the videos, code along with the videos. This will help you with the homework. When a screenshot is required, look for the icon:  You will keep adding to this project throughout this part of the course. When it comes time for the final project, use this project as a starter.

Project Resources: <https://github.com/promineotech/Spring-Boot-Course-Student-Resources>

Coding Steps:

For this week's homework you need to copy source code from the supplied resources.

For this week's homework you need to copy source code from the Source folder in the supplied resources. Wait until the instructions tell you to copy the resources or you will get errors.


- 1) Select some options for a Jeep order:
 - a) Use the `data.sql` file or the jeep database tables to select options for a Jeep order. Select any one of each of the following for the order:
 - i) color
 - ii) customer
 - iii) engine
 - iv) model
 - v) tire(s)
 - b) Select one or more options from the options table as well. Keep in mind that some options may work better than others – but if you want to put 37-inch tires on your Jeep Renegade, so be it!
- 2) Create a new integration test class to test a Jeep order named `CreateOrderTest.java`. Create this class in `src/test/java` in the `com.promineotech.jeepp.controller` package.
 - a) Add the Spring Boot Test annotations: `@SpringBootTest`, `@ActiveProfiles`, and `@Sql`. They should have the same parameters as the test created in weeks 1 and 2.
 - b) Create a test method (annotated with `@Test`) named `testCreateOrderReturnsSuccess201`.

Web API Design with Spring Boot Week 16 Coding Assignment

- c) In the test class, create a method named `createOrderBody`. This method returns a type of `String`. In this method, return a JSON object with the IDs that you picked in Step 1a and 1b. For example:

```
{
  "customer": "MORISON_LINA",
  "model": "WRANGLER",
  "trim": "Sport Altitude",
  "doors": 4,
  "color": "EXT_NACHO",
  "engine": "2_0_TURBO",
  "tire": "35_TOYO",
  "options": [
    "DOOR_QUAD_4",
    "EXT_AEV_LIFT",
    "EXT_WARN_WINCH",
    "EXT_WARN BUMPER_FRONT",
    "EXT_WARN BUMPER_REAR",
    "EXT_ARB_COMPRESSOR"
  ]
}
```

Make sure that the JSON is correct! If necessary, use a JSON formatter/validator like the one here: <https://jsonformatter.curiousconcept.com/>.

Produce a screenshot of the `createOrderBody()` method. 

```
}

protected String createOrderBody() {
    // @formatter:off
    return "{\n"
        + "  \"customer\": \"ATTAWAY_HECKTOR\", \n"
        + "  \"model\": \"COMPASS\", \n"
        + "  \"trim\": \"Freedom\", \n"
        + "  \"doors\": 4, \n"
        + "  \"color\": \"EXT_GRANITE_CRYSTAL\", \n"
        + "  \"engine\": \"2_0_TURBO\", \n"
        + "  \"tire\": \"35_TOYO\", \n"
        + "  \"options\": [\n"
        + "    \"DOOR_QUAD_4\", \n"
        + "    \"EXT_AEV_LIFT\", \n"
        + "    \"EXT_WARN_WINCH\", \n"
        + "    \"EXT_WARN BUMPER_FRONT\", \n"
        + "    \"EXT_WARN BUMPER_REAR\", \n"
        + "    \"EXT_ARB_COMPRESSOR\" \n"
        + "  ] \n"
        + "}";
    // @formatter:on
}
```

Web API Design with Spring Boot Week 16 Coding Assignment

In the test method, assign the return value of the `createOrderBody()` method to a variable named `body`.

- d) In the test class, add an instance variable named `serverPort` to hold the port that Tomcat is listening on in the test. Annotate the variable with `@LocalServerPort`.

- e) Add another instance variable for an injected `TestRestTemplate` named `restTemplate`.

- f) In the test method, assign a value to a local variable named `uri` as follows:

```
String uri = String.format("http://localhost:%d/orders", serverPort);
```

- g) In the test method, create an `HttpHeaders` object and set the content type to "application/json" like this:

```
HttpHeaders headers = new HttpHeaders();  
headers.setContentType(MediaType.APPLICATION_JSON);
```

Make sure to import the package `org.springframework.http.HttpHeaders`.

- h) Create an `HttpEntity` object and set the request body and headers:

```
HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);
```

- i) Send the request body and headers to the server. The `Order` class should have been copied earlier from the supplied resources. Ensure that you import `com.promineotech.jeepp.entity.Order` and not some other `Order` class.

```
ResponseEntity<Order> response = restTemplate.exchange(uri,  
    HttpMethod.POST, bodyEntity, Order.class);
```

- j) Add the AssertJ assertions to ensure that the response is correct. Replace the expected values to match the JSON in step 2c.

```
assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);  
assertThat(response.getBody()).isNotNull();
```

```
Order order = response.getBody();  
  
assertThat(order.getCustomer().getCustomerId()).isEqualTo("MORISON_LINA");  
  
assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.WRANGLER);  
;  
  
assertThat(order.getModel().getTrimLevel()).isEqualTo("Sport Altitude");
```

Web API Design with Spring Boot Week 16 Coding Assignment

```
assertThat(order.getModel().getNumDoors()).isEqualTo(4);

assertThat(order.getColor().getColorId()).isEqualTo("EXT_NACHO");

assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");

assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");

assertThat(order.getOptions()).hasSize(6);
```

k) Produce a screenshot of the test method. 


```
//test class
@Test
void testCreateOrderReturnsSuccess201() {
    // Given: an order as JSON
    String body = createOrderBody();
    String uri = getBaseUrlForOrders();
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<String> bodyEntity = new HttpEntity<>(body, headers);

    //When: the order is sent
    // Personal note: when creating new data use POST
    ResponseEntity<Order> response = getRestTemplate().exchange(uri, HttpMethod.POST, bodyEntity, Order.class);

    //Then: a 201 status is returned

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
    assertThat(response.getBody()).isNotNull();

    //And: the returned order is correct
    Order order = response.getBody();
    assertThat(order.getCustomer().getCustomerId()).isEqualTo("ATTAWAY_HECKTOR");
    assertThat(order.getModel().getModelId()).isEqualTo(JeepModel.COMPASS);
    assertThat(order.getModel().getTrimLevel()).isEqualTo("Freedom");
    assertThat(order.getModel().getNumDoors()).isEqualTo(4);
    assertThat(order.getColor().getColorId()).isEqualTo("EXT_GRANITE_CRYSTAL");
    assertThat(order.getEngine().getEngineId()).isEqualTo("2_0_TURBO");
    assertThat(order.getTire().getTireId()).isEqualTo("35_TOYO");
    assertThat(order.getOptions()).hasSize(6);
}
```

- 3) In the controller sub-package in src/main/java, create an interface named JeepOrderController. Add @RequestMapping("/orders") as a class-level annotation.
- a) Create a method in the interface to create an order (createOrder). It should return an object of type order (see below). It should accept a single parameter of type OrderRequest as described in the video. Make sure it accepts an HTTP POST request and returns a status code of 201 (created).
 - b) Add the @RequestBody annotation to the orderRequest parameter. Make sure to add the RequestBody annotation from the org.springframework.web.bind.annotation package.
 - c) Produce a screenshot of the finished JeepOrderController interface showing no compile errors. 

Web API Design with Spring Boot Week 16 Coding Assignment

```
package com.promineotech.jeepp.controller;


import javax.validation.Valid;

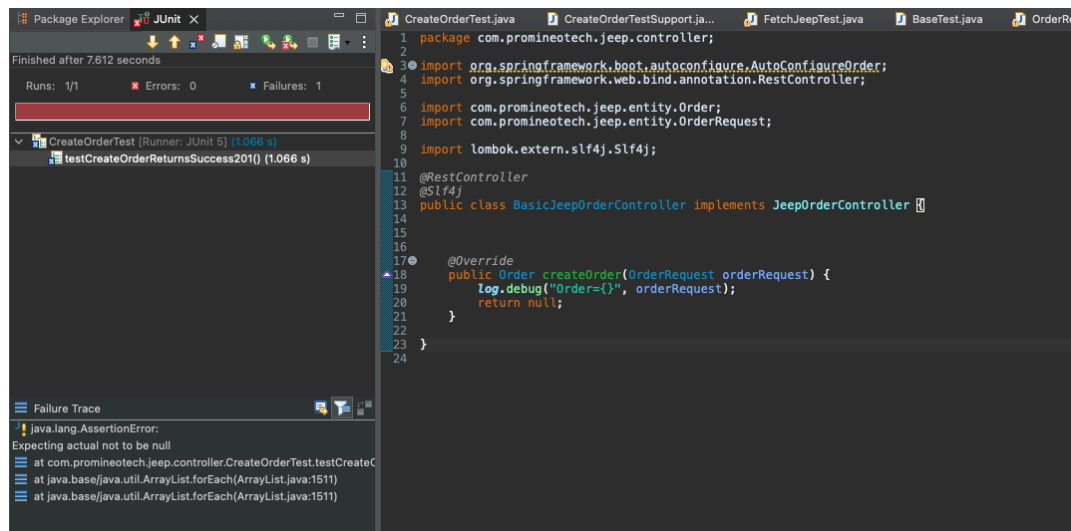
@RequestMapping("/orders")
@OpenAPIDefinition(info = @Info(title = "Jeep Order Service"), servers = {
    @Server(url = "http://localhost:8080", description = "Local server.")})

public interface JeepOrderController {
    //formatter:off
    @Operation(
        summary = "Create an order for a Jeep",
        description = "Return the create Jeep",
        responses = {
            @ApiResponse(
                responseCode = "200",
                description = "The created Jeep is returned",
                content = @Content(mediaType = "application/json", schema = @Schema(implementation = Order.class))),
            @ApiResponse(
                responseCode = "400",
                description = "The request parameters are invalid",
                content = @Content(mediaType = "application/json")),
            @ApiResponse(
                responseCode = "404",
                description = "A Jeep component was not found with the input criteria",
                content = @Content(mediaType = "application/json")),
            @ApiResponse(
                responseCode = "500",
                description = "An unplanned error occurred",
                content = @Content(mediaType = "application/json"))
        }
    ),
    parameters = {
        @Parameter(
            name = "orderRequest",
            required = true,
            description = "The order as JSON"
        )
    }
}

@PostMapping
@ResponseStatus(code = HttpStatus.CREATED)
Order createOrder(@Valid @RequestBody OrderRequest orderRequest);

//formatter:on
}
```

- d) Create a class that implements JeepOrderController named DefaultJeepOrderController.
- e) Add @RestController as a class-level annotation.
- f) Add a log line to the implementing controller method showing the input request body (orderRequest)
- g) Run the test to show a red status bar. Produce a screenshot that shows the test method, the log line, and the red JUnit status bar. 



The screenshot displays an IDE with two main panels. The left panel shows the JUnit test runner interface, indicating that the test 'testCreateOrderReturnsSuccess201()' failed after 1.066 seconds. The right panel shows the source code for 'BasicJeepOrderController', which implements the 'JeepOrderController' interface. The 'createOrder' method is annotated with '@Override' and includes a 'log.debug' statement to log the 'orderRequest' parameter before returning null. The failure trace at the bottom indicates an 'AssertionError' where the actual result was null, but the test expected a successful status (201).


Web API Design with Spring Boot Week 16 Coding Assignment

- 4) Find the Maven dependency `spring-boot-starter-validation` by looking it up at <https://mvnrepository.com/>. Add this repository to the project POM file (`pom.xml`).
- 5) Add the class-level annotation `@Validated` to the `JeepOrderController` interface.
- 6) Add Bean Validation annotations to the `OrderRequest` class as shown in the video.
 - a) Use these annotations for String types:
 - i) `@NotNull`
 - ii) `@Length(max = 30)`
 - iii) `@Pattern(regexp = "[\\w\\s]*")`
 - b) Use these annotations for integer types:
 - i) `@Positive`
 - ii) `@Min(2)`
 - iii) `@Max(4)`
 - c) Add `@NotNull` to the enum type.
 - d) Add validation to the list element (type String) by adding the validation annotations *inside* the generic definition. So, to add the String validation to the options, you would do this:

```
private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*")
String> options;
```


```
1 package com.promineotech.jeep.entity;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13
14
15
16 @Data
17 public class OrderRequest {
18
19     @NotNull
20     @Length(max = 30)
21     @Pattern(regexp = "[\\w\\s]*")
22     private String customer;
23
24     @NotNull
25     private JeepModel model;
26
27     @NotNull
28     @Length(max = 30)
29     @Pattern(regexp = "[\\w\\s]*")
30     private String trim;
31
32     @Positive
33     @Min(2)
34     @Max(4)
35     private int doors;
36
37
38     @NotNull
39     @Length(max = 30)
40     @Pattern(regexp = "[\\w\\s]*")
41     private String color;
42
43
44     @NotNull
45     @Length(max = 30)
46     @Pattern(regexp = "[\\w\\s]*")
47     private String engine;
48
49
50     @NotNull
51     @Length(max = 30)
52     @Pattern(regexp = "[\\w\\s]*")
53     private String tire;
54
55     private List<@NotNull @Length(max = 30) @Pattern(regexp = "[\\w\\s]*")String> options;
56 }
57
```

Do not apply a `@NotNull` annotation to the `List` because if you have no options the `List` may be null.

e) Produce a screenshot of this class with the annotations. 

Web API Design with Spring Boot Week 16 Coding Assignment

- 7) In the `jeep.service` sub-package, create the empty (no methods yet) order service interface (named `JeepOrderService`) and implementation (named `DefaultJeepOrderService`).
- Inject the interface into the order controller implementation class.
 - Add the `@Service` annotation to the service implementation class.
 - Create the `createOrder` method in the interface and implementing service. The method signature should look like this:

```
Order createOrder(OrderRequest orderRequest);
```
 - Call the `createOrder` method from the controller and return the value returned by the service.
 - Add a log line in the `createOrder` method and log the `orderRequest` parameter.
 - Run the test `CreateOrderTest` again. Produce a screenshot showing that the service layer `createOrder` method correctly prints the log line in the console. (e.g. prints out the `OrderRequest` in the console from within the Service Layer). 

- 8) In the `jeep.dao` sub-package, create the empty (no methods yet) DAO interface (named `JeepOrderDao`) and implementation (named `DefaultJeepOrderDao`).
- Inject the DAO interface into the order service implementation class.
 - Add the `@Component` annotation to the DAO implementation class.
- 9) Replace the entire content of `JeepOrderDao.java` with the source found in `JeepOrderDao.source`. The source file is found in the Source folder of the supplied project resources.

Web API Design with Spring Boot Week 16 Coding Assignment

10) * The next steps require you to copy source code from the Source directory in the supplied resources. Please follow the instructions EXACTLY. Some steps require you to replace ALL the source in a file. Some steps require you to ADD source to a file.**

11) Copy the *contents* of the file `DefaultJeepOrderDao.source` into `DefaultJeepOrderDao.java`. The source file is found in the Source folder of the supplied project resources.

In Eclipse, click the "Source" menu and select "Organize Imports". Pick packages from your project where applicable. Make sure you pick the import `java.util.Optional`, `java.util.List`, and `org.springframework.jdbc.core.RowMapper`.

12) Copy the *contents* of the file `DefaultJeepOrderService.source` into `DefaultJeepOrderService.java`. Add the source after the `createOrder()` method, but *inside* the class body. The source file is found in the Source folder of the supplied project resources.


In Eclipse, click the "Source" menu and select "Organize Imports". Pick packages from your project where applicable.

13) In `DefaultJeepOrderService.java`, work with the method `createOrder`.

- a) Add the `@Transactional` annotation to the `createOrder` method.
- b) In the `createOrder` method call the copied methods: `getCustomer`, `getModel`, `getColor`, `getEngine`, `getTire` and `getOption`, assigning the return values of these methods to variables of the appropriate types.
- c) Calculate the price, including all options.

14) In `JeepOrderDao.java` and `DefaultJeepOrderDao.java`, add the method:

```
Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire, BigDecimal price, List<Option> options);
```

- a) Call the `jeepOrder.Dao.saveOrder` method from the `jeepOrderSalesService.createOrder` service. Produce a screenshot of the `jeepOrderSalesService.createOrder` method. 

Web API Design with Spring Boot Week 16 Coding Assignment

```
22 @Service
23 public class DefaultJeepOrderService implements JeepOrderService {
24
25
26     @Autowired
27     private JeepOrderDao jeepOrderDao;
28
29     @Transactional
30     @Override
31     public Order createOrder(OrderRequest orderRequest) {
32         Customer customer = (getCustomer(orderRequest));
33         Jeep jeep = getModel(orderRequest);
34         Color color = getColor(orderRequest);
35         Engine engine = getEngine(orderRequest);
36         Tire tire = getTire(orderRequest);
37         List<Option> options = getOption(orderRequest);
38
39         BigDecimal price = jeep.getBasePrice()
40             .add(color.getPrice())
41             .add(engine.getPrice())
42             .add(tire.getPrice());
43
44         for (Option option : options) {
45             price = price.add(option.getPrice());
46         }
47
48         return jeepOrderDao.saveOrder(customer, jeep, color, engine, tire, price, options);
49     }
50 }
```

b) Write the implementation of the saveOrder method in the DAO.

- i) Call the supplied generateInsertSql method, passing in the customer, jeep, color, engine, tire and price. Assign the return value of the method to a SqlParameter object.
- ii) Call the update method on the NamedParameterJdbcTemplate object, passing in a keyHolder object as shown in the video. Create the keyHolder like this:

```
KeyHolder keyHolder = new GeneratedKeyHolder();
```

Be sure to extract the order primary key from the keyHolder object into a variable of type Long named orderPK.


- iii) Write a method named saveOptions as shown in the video. This method should have the following method signature:

```
private void saveOptions(List<Option> options, Long orderPK)
```


For each option in the options list, call the supplied generateInsertSql method passing the parameters option and order primary key (orderPK). Call the update method on the NamedParameterJdbcTemplate object.

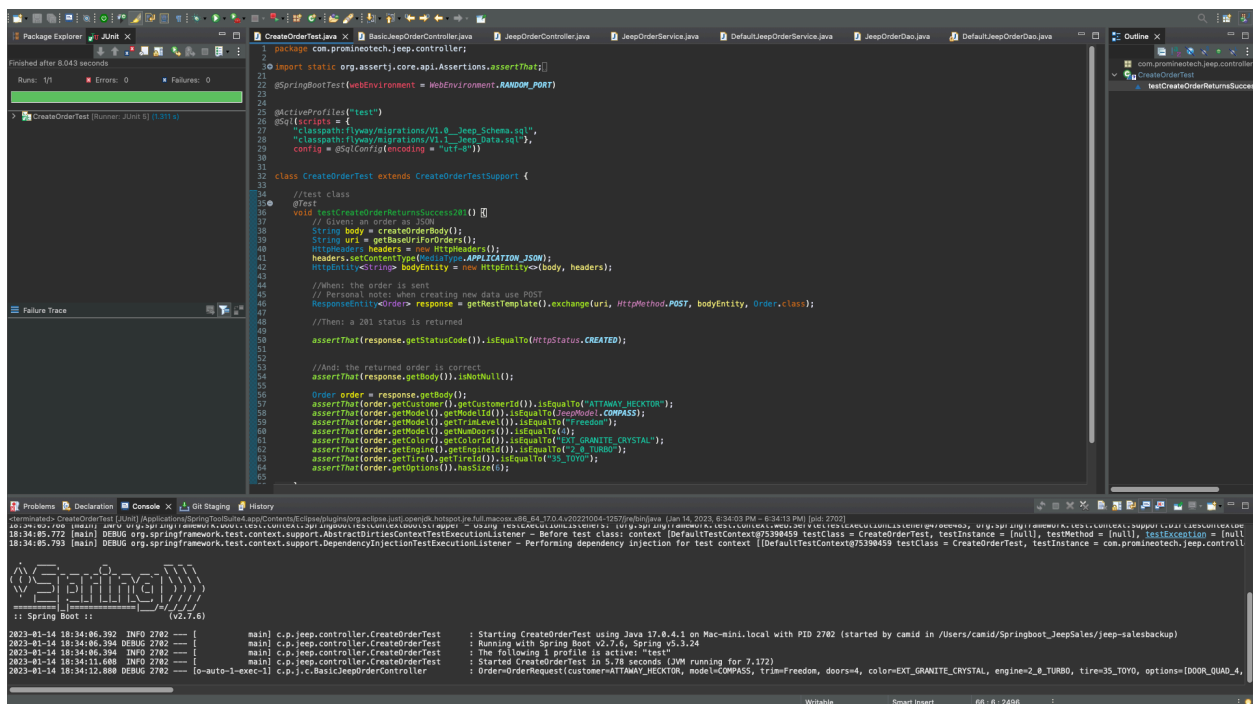
Web API Design with Spring Boot Week 16 Coding Assignment

iv) In the saveOrder method in the DAO implementation, return an order object using the order.builder. The order should include orderPK, customer, jeep (model), color, engine, tire, options and price.

v) Produce a screenshot of the saveOrder method. 

```
3
4
5 @Override
6 public Order saveOrder(Customer customer, Jeep jeep, Color color, Engine engine, Tire tire, BigDecimal price, List<Option> options) {
7     SqlParams params =
8         generateInsertSql(customer, jeep, color, engine, tire, price);
9
10    KeyHolder keyHolder = new GeneratedKeyHolder();
11    jdbcTemplate.update(params.sql, params.source, keyHolder);
12
13    Long orderPK = keyHolder.getKey().longValue();
14    saveOptions(options, orderPK);
15
16    //formatter:off
17    return Order.builder()
18        .orderPK(orderPK)
19        .customer(customer)
20        .model(jeep)
21        .color(color)
22        .engine(engine)
23        .tire(tire)
24        .options(options)
25        .price(price)
26        .build();
27    //formatter:on
28 }
29
```

c) Run the integration test in CreateOrderTest. Produce a screenshot of the test method that shows the green JUnit status bar, the console output, and the test class. 



The screenshot displays an IDE with the following components:

- Package Explorer:** Shows the project structure with a green status bar for the `CreateOrderTest` class.
- Code Editor:** Contains the `CreateOrderTest` class, which is an integration test for the `JeepOrderController`. It includes a `@SpringBootTest` annotation and a `testCreateOrderReturnsSuccess280()` method that performs a POST request and asserts the response.
- Console:** Shows the output of the test execution, including the Spring Boot logo and the text "Starting CreatedOrderTest using Java 17.0.4.1 on Mac-mini.local with PID 2782".
- Problems:** Shows no errors or warnings.
- History:** Shows the sequence of test execution steps.