TRICENTIS

Automation Specialist Level 2

Transcript

TABLE OF CONTENTS

Lesson 1: TestSheet Creation

In our scenario, you, as an Automation Specialist, have been tasked with testing the Shirt Size field. Looking at the DemoWebShop, we would need to log in, navigate to Apparel and Shoes and then select "Men's Wrinkle Free Long Sleeve" Shirt. If we want to add the shirt to the shopping cart, we have to choose a size. This is the functionality we need to test.
To do this, you will need to design tests to check that all the sizes are available. In other words, checking that all the valid options work. You must also check that selecting a size that should not be available returns an error message – in other words, test an invalid option. Therefore, we will also try to order a size that is not available and verify what the result is.
To have an overview of what we need to test and to have a central repository for our test data, we are going to create a TestSheet. This makes it much easier to maintain, and later automate, the process of using the data in the TestCases.

In this section, we are going to look at creating a TestSheet.
Objectives
By this end of this lesson you will be able to:
- Understand the basic concepts behind TestSheets
- Identify the differences between Attributes and Instances
- Create a TestSheet from scratch

TestSheet:
A TestSheet is the basic framework of TestCase-Design. It consists of Attributes and Instances which have their own values. These elements, in a structured way, specify the different combinations of the different TestCases. If we look at this complete TestSheet, we can see that it looks like a high-level guide through one slice of our SUT.
The content of a TestSheet can be viewed by focusing on it.
This is usually done by double-clicking on the TestSheet symbol in the Navigation pane.

To test the size selector in the web shop, we will create a simple TestSheet that will include our 3 valid sizes, S, M and L; Also, we will need to test an invalid size, for example, XL –this shirt is not available in our DemoWebShop.

To create a TestSheet, right click on the TestCaseDesign Folder and select Create TestSheet.
This TestSheet will need to be manually completed by us; however, this work is usually done by a Test Design Specialist, with values that are specified by the Business department.

An empty TestSheet is clearly not useful to us.
We need to add Attributes, and to them, we need to add the Instances.
But how do Attributes and Instances differ?
Attributes express the features of a specific person or thing.
For example, the Attributes of a laptop might be design or price, the Attributes of a person might be hair color, or height.
Instances are the different variants of those Attributes.
For example, Instances of the Attribute hair color could be blond, brown or black.
Within Tosca, Attributes are represented by a red square icon, Instances, on the other hand, are represented by a flat white box containing a capital "I".
One common issue is, if you have a limited view where you can only see the Attributes, press F9 to see Instances as well.
When the shortcut is used in the red, TestCaseDesign section of Tosca, it will hide or unhide all Instances.

Attributes have two functions: firstly, as explained before -describing the overall feature.
Secondly, the parent Attribute can be seen as an organizational element for the child / sub-attributes.
Instances also have two functions: They can be placed on the TestSheet level, where they represent the individual TestCases, or also under Attributes, where they represent the variant of the attribute to which they are attached.
Both Attributes and Instances can be added by right-clicking on a TestSheet or an Attribute and selecting Create Attribute or Create Instance, depending on what is needed.

Let's look at Attributes and Instances as features / variants within our scenario. When we apply this to our SUT, we see that the shirt TestSheet will have the Attribute "Size". The different options, S, M, L and XL are the single Instances of the Attribute "size".

Now, we will add the Instances on the TestSheet level to identify the actual TestCases that we need to create later. Click on the TestSheet itself and add Instances. This creates a new column in the TestSheet, which we can fill in with data to indicate the values to be entered at various stages for the test.
Instance values can be entered manually on the Instance itself.
On the Attribute level, the possible values can be selected by a drop-down menu.
This menu becomes available on the Attribute after at least 1 Instance is added.

⟵ ⟶

Notice that the first column on the left is called “ValidValues”.
In our TestSheet, the ValidValues column is considered the most important, most frequently used, or easiest path with the least dependencies. You may be familiar with the term “Happy Path”, which is similar to this concept.
In Tosca terminology, this column is referred to as the StraightThrough.

The final TestSheet contains 3 valid instance columns with the 3 possible options for sizes, and a 4th one with an invalid option, XL. These Instance columns will become the TestCases to create.

There is an important distinction to be made, between invalid Instances and a failed TestCase.
A TestCase fails when the values we are trying to verify are different from the values expected in the System Under Test.
An Invalid Instance just indicates that the TestCase should be verifying that the correct error message is shown when an invalid value is entered. So the SUT recognizes when something is incorrect.
As we expect the error message to appear, the TestCase will pass if this appears correctly.
However, this TestCase would fail if we are able to order a shirt in a size that is not available.

Our TestSheet is now ready to be used.

Lesson 2: Templates

Scenario
To recap, your task is to thoroughly test the size selector function for shirts on our SUT.
We could create individual TestCases to cover each possibility in the TestSheet. This approach has its downsides, even with using Libraries, as this will be a repetitive process. Also, each TestCase would require individual maintenance. For example, when the developers change something that falls in an area of the SUT not suitable for a ReusableTestStepBlock. To avoid such problems, we will create a TestCase Template, which is used to automatically create a number of TestCases, using the Test Data contained in the TestSheet.

By this end of this lesson you will be able to:
- Define what Templates are and why we use them.
- Create a Template

Templates are used to speed up and simplify the creation of TestCases.
Let us look at an abstract example for the different options when designing cars. Some cars are blue , some black, and some silver. Additionally, some of these cars have manual transmission and some automatic transmission.
It would be inefficient and very expensive to design and build every variant of our car from scratch. What we can do is to have the majority of the car designed the same. Then each car will differ only for the certain attributes.

The original car design acts as the blueprint and can spin off into different variations. This is effectively how Templates work.

Our Template acts as a blueprint for a series of TestCases that are similar but have a slightly different test focus each time .

Templates are designed to contain all possible paths of a TestSheet. Then, by linking the data from the TestSheet with the TestCase, we can quickly and easily create a number of TestCases.
This also makes it possible to quickly add, modify, or remove TestCases when there are changes to our System Under Test.
When the Template is modified to reflect changes, these modifications will feed through to the TestCases.
That's why, Templates are of great help when trying to cover all risk areas of our SUT in a time efficient way .

Let us look at our a real example, our the shirt sizes TestSheet that covers 4 possible paths.
One for each valid size possible to be chosen S,   M, L.
Additionally, 1 more path which uses an invalid size, namely XL.
With the help of the Template and our TestSheet, we will be able to create 4 TestCases to test the possible instances.

We will create the Template from an existing TestCase.
This TestCase has many References to ReusableTestStepBlocks in the Library.
To use this Template, to reflect all possible combinations in the TestSheet, we will have to modify it.
This means getting rid of, or "Resolving", some References, so that we can make changes to the TestSteps without changing the Library.
As you will remember from AS1, to do so, we simply right click on the folder then select Resolve Reference.

Once the TestCase is ready, converting it to a Template is a quick process – simply right-click on the TestCase and select "Convert to Template". You will notice a blue square with a T inside, appearing in front of the   TestCase icon.
That means the TestCase is now a Template.
The Template, if needed, can be converted back to TestCase, by right-clicking on it and selecting "Convert to TestCase".

in the next lesson, we will use this Template to create the 4 TestCases defined in the corresponding TestSheet in the TestCaseDesign section.w

Lesson 3: How to use Templates

We are now one big step closer to being able to test the shirt size selector function of our SUT.
However, a template cannot be run as a test itself.
A Template lacks some of the specific information needed for a test to successfully complete .

The Template will create a series of slightly different TestCases that use the Template TestCase as a starting point.
In order for the Template to be able to do this it needs more information on how to create the "amended" TestCases.
Returning to our Car analogy we have the base car designed.
Now we need to know what color and what transmission to put in it.

We already have this information in the form of a TestSheet. So all we need to do now, is link the TestSheet to the Template.

By this end of this lesson you will be able to:

- Link a TestSheet to a Template
- Link the TestSheet Values to the TestStep Values in the Template
- Use the Check Template and Jump to Schema functions
- Instantiate the Template

Now that we've converted our size selector TestCase to a Template, the sizes TestSheet can be linked to our Template.
To do that, select the TestSheet, from the TestCaseDesign section in Tosca, then drag and drop it onto the Template in the TestCase section of Tosca.

It's best practice to always link a TestSheet to a Template as soon as it is created, instead of leaving it unlinked. This will prevent confusion later on as to which Template relates to which TestSheet.
While Tosca allows you to link a TestSheet to more than one template, it should be avoided as it is harder to maintain and keep an overview of several templates. Each TestSheet should be used for one template only. If the same data is required for multiple Templates, we recommend using Classes. This will be covered in the Test Design Specialist Level One Training Course.
Once the TestSheet is linked to the Template, a new option will become available in the ribbon, under the TestCase tab: Jump to Schema Definition. If you are not sure which TestSheet is linked to the Template, this command will show you.
This allows you to immediately jump to the linked TestSheet from the Template itself.
The linked TestSheet can also be reached by right-clicking on the Template and selecting Jump to Schema Definition from the menu, the shortcut for this is CTRL+J.

Additionally, you can also check the name of the linked TestSheet using the Property tab

The next step is to link the Values from the TestSheet to the relevant TestStepValues in the Template
Without this, we would not be able to tell Tosca which parts of the TestSheet are used for the different TestSteps.
Best practice, when linking values, is to drag and drop them onto the relevant TestStepValue from the TestSheet.

Lets look at our Size selector example.
We drag and drop the Attribute "size" from the TestSheet onto the "Size" TestStepValue in the TestCase, which is the point in the TestCase where the size will be chosen.
This will automatically create an XL reference.
The XL reference shows that the value for this TestStepValue will be taken from the TestSheet.
The XL reference itself contains the path, for Tosca to follow in order to find the value in the TestSheet.

Once this linking is done, we will need to check the Template for errors in the XL references.
We can do this by right-clicking on the template and selecting the Check Template option.
This will run through the links to the TestSheet and will validate their existence.
However, it is important to remember that this will not check the template for functionality errors in the TestSteps or formulas.
The Check Template function will highlight the TestStepValue containing an error, not the XL reference itself.

At this point, the Workstate should be set to complete.

We now have a complete Template. The Next step is to use it to create the TestCases. We do this through a process called "Instantiation".

By instantiating a Template, Tosca creates TestCases according to the TestSheet and the Conditions placed on the different objects in that Template.
We will cover conditions in later lessons.

This process will create 4 TestCases. One TestCase for each of the 4 TestCase instances in our TestSheet.
To Instantiate, select the Template and then click on the "Instantiate" button in the Tosca's Ribbon, in the TestCase Tab.

Alternatively, you can right click the Template and select "Create TemplateInstance" from the Mini Toolba r.
You will be prompted with the question "start instantiation now?"
Select yes.

You can also select the Template and use the shortcut Ctrl+N, Ctrl+I.
The Template remains the same, but below, a new folder called TemplateInstance is created.
If you expand this folder, you will see all TestCases that were created based on our Template, with the correct values for the test focus.
It's important to check the TestCases . If the Template was correct, the TestCases will ONLY show the steps that are relevant to that TestCase.
For example, the ValidValues TestCase only uses the default size, M.
In the Wrong size TestCase, the invalid size (XL) is entered.
Also, you will see the Values from the TestSheet being used instead of the References.

We can now run all TestCases in the Scratchbook, and confirm they run as expected.

Lesson 4: Modifications to Templates

Our SUT is developed using an Agile Methodology. In this sprint the developers introduced a new feature to the Add to Cart button. The SUT now shows a message after a number of shirts has been entered, and the Add to Cart button has been clicked. The purpose of this message is to show the user if they have entered a valid quantity, or an invalid one.
Your Task is now to test that this new functionality is working as it is supposed to.

Let's look at the new functionality: if the quantity entered is a positive whole number , bigger than 0, the SUT will display a success message with a green background. This means that it is a valid entry.

In case of a negative whole number , the SUT will display an error message with a red background. This tells us that the value is invalid.
We must verify that the correct message appears.

This feature adds
to a part of the SUT, that was already tested. You already have the Modules, Template, TestCases and TemplateInstances from the previous version.
You receive a new TestSheet from the Test Design Specialist including the Verification Attributes needed to Verify the message from the SUT.

By this end of this lesson you will be able to:
- Define what a Verification Attribute is,
- Modify an existing template
- Re-Instantiate a modified template

Let's look at the new TestSheet:
A new type of Attribute has been added.
The green Attribute is called a Verification Attribute.
Instead of being an input to the SUT, it has a value that Tosca uses to verify against the SUT output.
Once entered into the TestCase, if the 2 values match, the verification is successful. Otherwise it fails.

Let's take a more detailed look at this new TestSheet.
There are only 2 Instances for the Quantity Attribute: +4 and -4.
+4 is a valid Instance, while -4, is an Invalid one.
We see that, for the Instance +4, the Validation Message is: "The product has been added to your shopping cart".
While, for the instance -4, the Validation verifies that the error message is: "Quantity should be positive".
Once we have modified our previous Template to match the new TestSheet, we will get 2 TestCases, that validate the system message after Instantiation.

Now that we understand the new TestSheet, we can start modifying our template.
It's best practice to check the template for any References to resolve, before proceeding to modify it.
It prevents any changes feeding through to our Library,

All TestSteps that require modification, need to be identified.
If any of them have references to ReusableTestStepBlocks, those references will need to be resolved .
In this case, we must also add some new steps to verify the messages.
Additionally, we will need to link the new, revised TestSheet to the Template. When you Link the new TestSheet, the link to the old TestSheet is automatically removed.
Once the TestSheet has been linked, we need to take care of the XL links. If the Attributes and Instances have been named the same way in both the old and the new TestSheet, then Tosca will update the links automatically. These should, however, be carefully checked.
Any new links, in this case, the Verification Values, will also need to be added manually to the respective TestStepValues.
Once you have linked the TestSheet and its Values to the Template, it's important to check for errors.
When updating an existing Template is easy to make mistakes, for example, not linking a new value or forgetting to change an old one.
The Check Template function is very helpful in these cases.

Now the Template is almost ready to be Instantiated. Before that can happen, delete the old TemplateInstances folder from the previous Template.

We are now ready to Instantiate.
This step is very important because, up until now, the changes and new values haven't been transferred into the TestCasesInstances.
Only after Instantiation, we will have complete TestCases that include the verification.
This can also be done by highlighting the Template and using the keyboard shortcut CTRL+N, CTRL+I.

What would happen if the Test Design specialist had amended the existing TestSheet instead of providing a new one? This is even easier to deal with. Make the changes to the Template to take account of the new TestCases. Then use the command "Re-instantiate" to run the instantiation process again. It will refresh the data within the existing Template instances, as well as add any new Template instances.

Lesson 5: Conditions

With other slightly more complex Templates, if we were to run the TestCase Instances now most of them are likely to fail.

The reason is because our TestCases would contain contradictory Validation steps for the same Values.

Some Validations can only be verified if specific Conditions are fulfilled.

Returning to the car analogy, we want to validate that a green car is automatic and a blue one is manual. This would be done in one TestStep. However, the TestCase would try to validate that the car had both automatic and manual transmission.
Therefore, TestCases will always fail, we can't have two competing validations in a TestCase.
To solve this, we will tell one validation to apply only with a green car, the other only with a blue car.
This is achievable with Conditions, in Tosca.
We can set Conditions on different levels of the Template including TestStepFolder, TestStep and TestStepValue levels.
These conditions tell Tosca when and how to run those elements.

By this end of this lesson you will be able to:
- Define what Conditions are
- Know how and when to use Conditions
- Set Conditions on a Template

To use a Template and create multiple TestCases from it, we will have to set Conditions.
These Conditions can be set at different levels: TestStepFolder-, TestStep-, and TestStepValue level.
Conditions instruct Tosca on when to use (or not to use) certain TestSteps, TestFolders, or Values.
For example: A condition on a TestStep Folder will act as a rule for when the content of the entire folder will be run.
A condition on a TestStepValue will just be applicable on that specific Value.
This means that we can test multiple paths with the same Template.
This is because we will specify that certain parts of our Template are only run if they meet the specified prerequisites.

For example, if 0 or a negative amount is selected during the shirt order process, then Tosca is instructed to skip the verification of the green success message when the TestCase is run.
Alternatively, if a positive whole number is selected, Tosca will be instructed to skip the red message verification.

Conditions avoid Template repetition.
Using appropriately set Conditions means only one Template is needed.
This avoids the use of different Templates where elements will need to be changed manually.
The result is: easier maintenance as everything is centralized and there is only one place to apply changes manually.
These changes will then be applied to the TestCases automatically, as seen in the previous lesson.

What we want for our example is that, if the shirts are added successfully, Tosca will navigate to the shopping cart, empty it, log out and then close the browser.
If the shirts cannot be added, Tosca will just log out and then close the browser.
In this Template, we will set a condition on the TestStepFolder level.
The "Navigate to Shopping Cart and Delete" TestStepFolder will need a Condition, so that it is only executed, if the product is added.

The first step when adding Conditions, is to make sure the Condition column is visible. We do this using the column chooser.
To set the condition, we drag and drop the Instance of the Verification Attribute for the positive value, from the TestSheet onto the corresponding Template cell.

It's best practice to always drag and drop conditions from the TestSheet onto the Template, as this reduces the chance of human errors.
When an Instance is dragged and dropped in the condition column, the logical statement will show == by default.
It's possible to combine conditions so that multiple conditions are used for a single TestStep using logical statements.
{Fade into next point}
The statement AND indicates that all conditions must be met in order for Tosca to use this TestStep in the automation
{Fade into next point}
The statement OR indicates that only one of the conditions must be met for Tosca to use this TestStep in the automation.
By default, Tosca will set the logical statement to OR if more than one condition is added. You should limit the use of OR conditions, they can overcomplicate our Conditions set, with several layers of OR statements piled up.

{Fade into next point}
When the Operator == is used, the element must be equal to a property for Tosca to use this TestStep in the automation. For example, an Instance must be valid for Tosca to automate a specific TestStep.

{Fade into next point}
The Operator != can be used to simplify the Conditions set.
This operator means not equal to, or everything except.
When we use this Condition, the section on which it is applied, is used every time, except when the 2 parameters of the Condition are equal. For example, we want to apply a condition about who in a class can take an exam. The condition is: "Take the exam =! Teacher". This Condition means the only people who can take the exam are the students.

This is an example of more "complex" Conditions.
This is a completed Template that tests the Payment Process of the DemoWebshop.
To test all of the credit cards, we use 'Workflow.Payment Method.Credit Card' != "No Credit Card used" condition. The double negative means that as long as a Credit Card is used, the TestStep can run.
this prevents the overuse of the Condition "OR". Instead of using OR for each credit card, we use the != operation. This makes the Conditions more business readable.

In this next example, your selected payment method is cash or check.
Given this, you will have to pay some extra fee.
The condition verifies that when you use one of those payment methods, the correct Math function is run to verify the correct fee amount.

These examples were covering what are the most used conditions.
On our online manual you can find a list of more case-specific ones.

Finally, the Template is fully complete.
As seen in the previous lesson there are 2 steps to take after modifications to a template are done.
The first is to use the Check Template function to find possible errors.
The second is to re-Instantiate the Template.
After the re-Instantiation is completed, we will have our set of perfectly working TestCases ready to be used for testing.

Lesson 6: Run and Report Automated Tests

The Test Automation process is now set up.
The next step is to run these tests and provide a report of what has been tested and if any bug was found.
As we learned in Automation Specialist Level 1, we use ExecutionLists and RequirementSets, to run our tests and obtain an overview of the project.

In this lesson, we will create and run an ExecutionList for the two templates you have been working on in this course.

By the end of this lesson you will be able to :

- Create and link to Requirements an ExecutionList
- Run linked ExecutionLists

The Execution section in Tosca displays and saves the information generated by the tests. To allow Execution, link the TestCases to an ExecutionList. The ExecutionList can then be run to action the tests.
Any previous results are stored and collected, ready for any report.
A developer can then be informed exactly which step failed and what the error message was.

First, we will have to create an Execution List for the Order Shirts TemplateInstances we created.
As seen in the previous course, once you are in the Execution section, there are three ways to create Execution Lists:

1) By clicking Create Object in the EXECUTIONLIST Tab in the Ribbon ;
2) Right-clicking on the ExecutionList Folder selecting Create Execution from the Context Menu; or
3) Using the keyboard Shortcut Ctrl+N; Ctrl+E

Once created, drag and drop the TestCases generated by the Template into the relevant ExecutionList.

It's best practice to drag and drop the whole TemplateInstance folder onto the ExecutionList folder, rather than individual TestCases.
This is for easier maintenance.
If the folder is linked rather than the individual TestCases, it is easier to synchronize when the Template is changed.
This synchronization is manually triggered on the ExecutionList ribbon.
Then all ExecutionLists that are linked to the TemplateInstance folder are automatically updated.

The TemplateInstance Folder will also need to be linked to the RequirementSet.
Thi s will populate the results data in the appropriate requirement and RequirementSet, providing a more complete overview of the requirement coverage.

Before linking the TemplateInstances to the ExecutionList, and the RequirementSet, it's important to make sure that the WorkState is set to COMPLETED.
If you've already linked the TemplateInstances, you will need to set the Template WorkState to COMPLETED and then reinstantiate it.
This can be done by right-clicking on the TemplateInstance Folder and selecting "ReInstantiate".

Link the ExecutionList to the appropriate RequirementSet via drag and drop.
The column ExecutionState shows the result of the executed TestCases, taking the risk into account leading to a risk-based coverage statement. The column CoverageSpecified shows what percentage of the overall risk has been covered within the project.
These numbers will be useful for reporting, as they provide status report or dashboard in a readable format.

Now we run the ExecutionLists.
Once finished, we can see our test execution results in further detail and report back if and where our test found any errors.

⟵ ⟶

Lesson 7: Build Template & Link Values recap

In the previous lessons and exercises, we only worked with one step of the Template creation process per lesson.
This allowed us to focus on the smaller individual processes that, when combined, create a complete Template.

By the end of this lesson you will be able to:
- Automate an end to end process, covering a more complicated use case

To attract more users to the online store, the developers have implemented a new Discount Code feature.
A user will receive a discount when they input the correct discount code.
The text that activates a discount has to be entered in the correct syntax. If the text does not match exactly, the code is invalid.
In case of an invalid discount code, a message will appear, stating that it was not accepted.
We have several codes, and each discount code unlocks a different benefit, for example:
One removes all Shipping costs. Another has a $5 flat discount, meaning that $5 will be subtracted from the Total price, regardless of the value of the total order.

You are to test this new function of the SUT, so you need to test that the right discount is applied when you use each Discount Code.
Additionally, you must check if any invalid code is being rejected by the SUT, and that no discount is applied.
In order to prepare for the exercise, re-watch Lesson 2 for Building a Template, Lesson 3 for linking Values and Instantiation, and finally Lesson 5 for Setting Conditions.
Important tips and guidelines can also be found in the related exercises.
As a summary, the steps to create a fully working Template are:
- First, create the TestCase that will be converted
- Convert the TestCase to a Template
- Next, link the Template to the relevant TestSheet and link the single Instances from the TestSheet to the relevant Template TestStepValues.
- Add Conditions to define when to use, and when not to use parts of the Template. This will create functioning TestCases following Instantiation.

To be able to complete the task, we need to touch on an additional concept.

In lesson 1 we introduced Invalid Instances as part of our TestSheet.
These Instances are symbolized by an X in the center of a square and contain values that should be rejected by the SUT.
In our invalid TestCase, when a Discount code is incorrect, the request for a discount must be rejected. The order process then stops – our verification is complete, and we do not need to spend time completing the order process.

In our TestCase this means, on the TestStepFolder level, we know that the invalid Instance TestCase stops after Verify Application.
It should skip over the Workflow and Verification folders and go directly to the Postcondition.

We could achieve this by writing a Condition string that includes all Instances, separated by an OR operator. This would be impractical as it would be very long, not business readable and high maintenance.

To avoid this, we can actually set the condition to read the Instance character - Invalid or Valid (as we saw in the Discount Code TestSheet). Here, we will set both the Workflow and the Verification TestStepFolder conditions to 'Instance.Character' != "Invalid".
The code != is the operator that stands for "Not equal to".

The "Not equal to" operator allows us to complete the same action but in a simpler way that is business readable, and much easier to maintain.
In our example, the Condition set would state that Tosca will run the selected TestStep or TestCase every time the instance in use is not Invalid.
It is also required to add the verification calculation conditions on the TestStepValue level.
Finally, link the Discount Code TemplateInstance to a Requirement and ExecutionList for a better project overview.
The final product should be a successful run of the ExecutionList.

Remember to re-visit the "Hints" sections in the exercise notes for additional guidance.

← →

Lesson 8: Create API TestCases

As in the previous lesson, your goal is to test if the countries and the country IDs are combined correctly by the system.
You have been sent a series of artefacts.

- A subset, directly from the developer.
- A TestSheet, from the TestDesign Specialist.

You will now have to create the TestCases needed to test the new DemoWebShop functionality.

In this lesson, we will explore the differences and similarities between UI and API testing practices within Tosca.

By the end of this lesson you will be able to:

- Convert the API TestCase into a functioning Template
- Link values from the TestSheet to the API Template
- Instantiate the API Template and run the TemplateInstances

Before we start working on our TestCases, it is important to know how they were generated.
It is possible to scan the API of an application using the API Scan feature of Tosca.
In our case, developers have already done some preparations.
They scanned the application, collected relevant Request and Response pairs and exported the result.
You can learn more about the API Scan itself in our “Automation Specialist for API” training.

The result of the export will be a raw TestCase.
It will need adjusting. Some TestSteps will have to be re-named and polished.
While applying any changes to the exported results, we have to keep a clear structure and naming convention.
It is important, so that anyone can understand the flow when working on the project.
The final structure of the TestCase should be similar to a UI TestCase.
This way, is not only easier to understand but also easier to automate.

There is one important difference to UI TestCases to remember.
In API TestCases each Message, which performs a specific function, is expressed as a Request and Response TestStep pair.
This is because, like in our restaurant example, you cannot receive food if you don’t make an order first.
Once you have placed your order, you will expect to receive some kind of response back.
Likewise, for API, without a Request you won’t be getting a Response back, and a Response is useless without its Request.
This is why we need to make sure that for each step they are both present.

Once our TestCase structure is ready, we will proceed to convert the TestCase to a Template.
This is done exactly like in previous lessons.

At this point we can continue as we would with a UI Template.
First, we will need to link it to the TestSheet provided.
After that, we link the single TestSheet Attributes to the Template’s TestStep Values.
Conditions have to be added accordingly.
As you see, the basic flow is the same as in UI test automation

Once the template has been checked for errors, we can instantiate it.
Even here, the procedure is the same as in the previous lessons, for the UI testing automation .
As we have already seen before, this will create the TemplateInstances

We can now run our TestCases and test the new DemoWebShop functionality.

If you are interested in becoming an expert in API testing with Tricentis Tosca, please check out our course “Automation Specialist for API”.

⟵ ⟶

Sit and Talk

Hello and welcome to the Automation Specialist Level 2 course.

My Name is Jordan Grantham, I am Head of the Tricentis Academy in Vienna.

Today we'll continue our journey through the automation of TestCases, using Tricentis Tosca.

After completing Automation Specialist Level 1, we are now looking at how we can become more efficient in creating and maintaining our TestCases.
So far, we have concentrated our efforts in creating 1 TestCase at a time.

However, testing an SUT thoroughly requires the creation of many TestCases.
At the same time, time and resources are usually limited.

Therefore, it's vital to find out which part of the SUT to test first, and also which parts require the most focus and effort.
To solve this, we could, theoretically, copy and paste one TestCase and modify each and every of them.
However, this process is time consuming.
Finally, this would also create a structure that is difficult to maintain, as each TestCase would need to be updated individually.

Why is this?
Every time any shared element changes, there will be a need for manual updates.
Normally, we can take account of this with a library and ReusableTestStepBlocks.
But what if this change happens in an area with a verification or a value that changes depending on the focus of the TestCase?

This is made even worse when it comes to bigger and more complex tests.
All these problems can be addressed with appropriate TestCase Design, paired with Risk-Based Testing.

To increase test coverage, we need a number of similar, but slightly different Testcases.
These TestCases will test the same feature, but in different ways.

In this course we will focus on creating these multiple TestCases at the same time.
All paired with Tosca's trademark ease of use and maintainability.

The focus of our training will be on centralizing our TestData and TestCases.
This will be done by introducing TestSheets and Templates.
These elements, are not only invaluable tools in the creation of effective TestCases but also benefit the maintenance process greatly.
Effectively, by centralizing our resources, we will have only 1 place where we can apply changes and update data, not hundreds.
Therefore, we can make the change in one place, and it is fed to hundreds of TestCases.

This more flexible, maintainable and agile structure is a perfectly fitting piece of our DevOps Cycle. It increases adaptability and shortens the overall duration of the testing part.
It grants a smooth transition between development and testing phases, cutting long and costly maintenance times at each iteration.

In this course we will also introduce the concept of API Testing.
We will focus mostly on how to automate and maintain API tests.

After completing the Automation Specialist Level 2 course, you will not only have gained a deep and effective knowledge of Tosca's automation techniques.
You will also gain the skills needed to dive deeper into more complex testing territories.
1 possible path, to extend your Tosca knowledge, is to choose the Test Design Specialist Level 1.

Alternatively, specialist courses also follow AS2, like the Automation Specialist for API, which looks deeper into the API testing capabilities of Tosca.

Finally, enjoy the course! If you want to access and expand on your Tricentis Tosca skills and testing knowledge, remember to register for our MOOCs, follow us on Facebook and LinkedIn and subscribe to our YouTube channel for useful additional videos.

← →