

Demo Script

GeneXus™ 16

Step-by-step guide for presenting GeneXus™ 16.

Copyright / GeneXus S.A. 1988-2018.

All rights reserved. This document may not be reproduced by any means without the express permission of GeneXus S.A. The information contained herein is intended for personal use only.

Registered Trademarks:

GeneXus is a trademark or registered trademark of GeneXus S.A. All other trademarks mentioned herein are the property of their respective owners.

TABLE OF CONTENTS

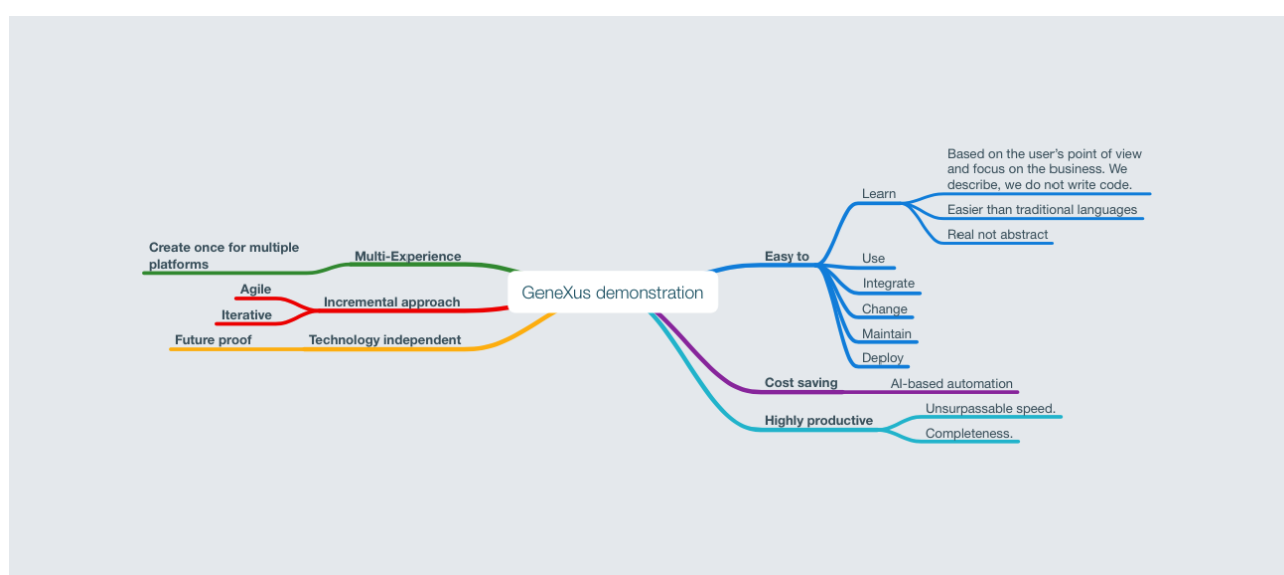
Table of Contents	2
OBJECTIVES	3
GENEXUS CHALLENGE	3
DEMO SCRIPT	4
1. NEW KNOWLEDGE BASE	4
2. CREATING THE ORDER TRANSACTION	4
3. VIEWING THE DATA MODEL	9
4. VIEWING THE WEB FORM	10
5. RUNNING THE APPLICATION FOR THE FIRST TIME	12
6. DEFINING BUSINESS RULES	13
7. CREATING THE PRODUCT TRANSACTION	15
8. CREATING THE CUSTOMER TRANSACTION	18
9. PRODUCT CATALOG	22
10. MULTI-PLATFORM	25
11. SUMMARY	42
12. GX SERVER	42
13. GX TEST – UNIT TESTS	42
14. GX FLOW – BUSINESS PROCESS MODEL	43
15. DEPLOY TO CLOUD – F6	43
16. CONCLUSION	43

OBJECTIVES

This document describes the steps that must be followed to make a GeneXus demo. It includes a script of the story with all the important points that must be conveyed in a GeneXus presentation.

The objective of the demo is to show prospects the main benefits that will be obtained by purchasing GeneXus. The benefits to be presented are based on the pillars of GeneXus, and will be demonstrated with facts in the course of the presentation.

Below are some of the pillars of GeneXus turned into direct benefits for our platform users.



GENEXUS CHALLENGE

To engage the audience, we present the demo as a challenge, inviting them to take part in the *GeneXus Challenge*. To this end, we will explain to participants that, in our demo, we will build a system for the GeneXus company to manage purchase orders of their merchandising products and their customers. *The GeneXus company has been chosen as an example; therefore, we recommend adapting this example to your potential customer.*

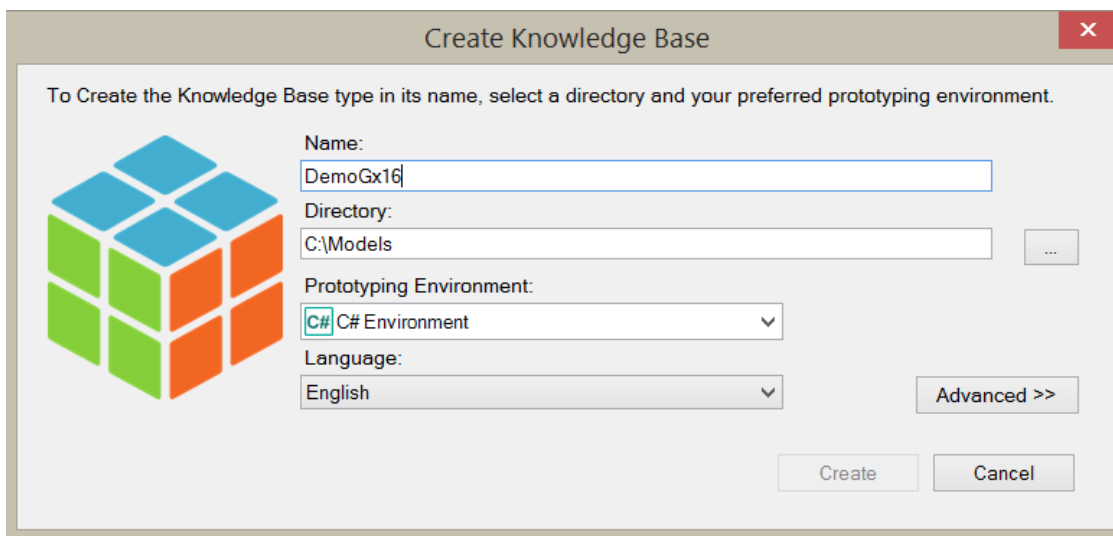
The challenge is that participants will be asked to estimate how long it would take them to build from scratch a system with these features for a web platform, as well as an application for mobile devices. Once this primary survey is complete, it would be good to start a stopwatch and begin with the demo.

DEMO SCRIPT

1. NEW KNOWLEDGE BASE

I'll start by creating a Knowledge Base.

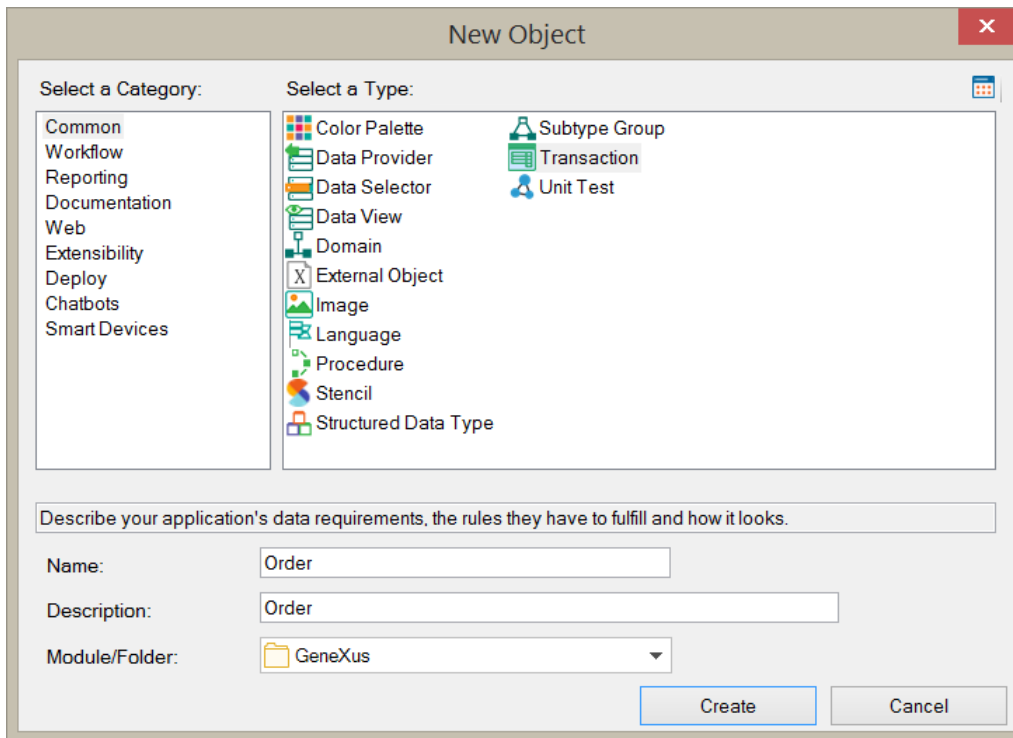
So, I select *File/New/Knowledge Base* and enter a name for this Knowledge Base.



Note: As the KB is being created, explain why GeneXus projects are called Knowledge Base, indicating that in GeneXus applications are described instead of being programmed. They are designed according to the reality of the business being modeled; therefore, the project becomes a knowledge base of the business.

2. CREATING THE ORDER TRANSACTION

I'll start to describe and model the reality by selecting *File / New / Object* [Shortcut CTRL + N].



Note that GeneXus includes several types of objects which, combined, allow us to model any application.

In particular, to represent the entities of this reality we use the Transaction object.

The Transactions defined are used to infer the application data model (3rd normal form). GeneXus also uses the Transaction object to generate the application program that will allow the end user to interactively insert, delete and update records on the database.

So, I create the Order Transaction.

What are we going to record for a purchase order?

- A number that identifies it
- Date
- Customer's details
- Set of products with their price, quantity, and line total
- Invoice total.

Notes: As you enter the attributes in the transaction, explain that as a good practice a naming convention is used in which each field (attribute) is named with a prefix of the transaction to which it belongs.

Use the “Period” key to preload the name of the transaction when naming each attribute.

Use the “Tab” key to move between the attribute name, data type, and description.

Use the “Enter” key to add a new attribute.

Create the “Price” domain when defining the ProductPrice attribute and reuse it in OrderProductTotal and OrderTotal.

Domains allow defining data types in a centralized manner so that they can be reused. In this way, we increase the level of abstraction and ability to adapt to changes. For example: in the year 2000 the dates started to have the year modeled with 4 digits instead of 2.

Name	Type	Description
Order	Order	Order
OrderId	Numeric(4.0)	Order Id
OrderDate	Date	Order Date
CustomerId	Numeric(4.0)	Customer Id
CustomerName	VarChar(40)	Customer Name

So far, I’ve entered the purchase order header fields and will now enter the lines.

To do so, I will:

- Press *CTRL + Right Arrow* to add a new level to the data structure, or right-click and select “Insert Level.”
- Enter the fields of the second level and then press *CTRL + Left Arrow* to return to the header level and enter the data at the bottom of the purchase order.

Name	Type	Description	Formula	Nullable
Order	Order	Order		
OrderId	Numeric(4.0)	Order Id		No
OrderDate	Date	Order Date		No
CustomerId	Numeric(4.0)	Customer Id		No
CustomerName	VarChar(40)	Customer Name		No
Product	Product	Product		
ProductId	Numeric(4.0)	Product Id		No
ProductName	VarChar(40)	Product Name		No
ProductPrice	Price	Product Price		No
OrderProductQty	Numeric(4.0)	Order Product Qty		No
OrderProductTotal	Price	Order Product Total		No
OrderTotal	Price	Order Total		No

- Select the OrderId attribute, and set it as autonumbered in its properties [Shortcut: F4].
- Save the Transaction.

In this way, I've just created the structure of a Transaction composed of two levels:

1. A basic level (Order), where all the information needed for the purchase order header is specified.
2. A nested level (Product), where the item data will be specified.

Note: Explain that certain fields should be calculated automatically, such as: OrderProductQty and OrderTotal. To do so, we give GeneXus the formula to calculate each one of them.

Also, I select the OrderProductTotal attribute formula column and type:

ProductPrice * OrderProductQty

The screenshot shows the GeneXus IDE interface. The main window displays a data model for an 'Order' entity. The model is organized into a tree view on the left and a table on the right. The table lists attributes with their names, types, descriptions, formulas, and nullable status.

Name	Type	Description	Formula	Nullable
Order	Order	Order		
OrderId	Numeric(4.0)	Order Id		No
OrderDate	Date	Order Date		No
CustomerId	Numeric(4.0)	Customer Id		No
CustomerName	VarChar(40)	Customer Name		No
Product	Product	Product		
ProductId	Numeric(4.0)	Product Id		No
ProductName	VarChar(40)	Product Name		No
ProductPrice	Price	Product Price		No
OrderProductQty	Numeric(4.0)	Order Product Qty		No
OrderProductTotal	Price	Order Product Total		No
OrderTotal				No

A 'Formula Editor' dialog box is open, showing the formula `ProductPrice*OrderProductQty` in the text area. The dialog has 'OK' and 'Cancel' buttons at the bottom.

And do the same for the OrderTotal attribute and type:

`SUM(OrderProductTotal)`

The screenshot displays the GeneXus IDE interface. At the top, there are tabs for 'Start Page' and 'Order'. Below the tabs is a menu bar with options: 'Structure', 'Web Form', 'Win Form', 'Rules', 'Events', 'Variables', 'Help', 'Documentation', and 'Patterns'. The main area shows a 'Structure' view with a tree on the left and a table on the right. The tree shows a hierarchy: Order (containing OrderId, OrderDate, CustomerId, CustomerName, Product, OrderProductTotal, and OrderTotal). The table lists the following fields:

Name	Type	Description	Formula	Nullable
Order	Order	Order		
OrderId	Numeric(4.0)	Order Id		No
OrderDate	Date	Order Date		No
CustomerId	Numeric(4.0)	Customer Id		No
CustomerName	VarChar(40)	Customer Name		No
Product	Product	Product		
ProductId	Numeric(4.0)	Product Id		No
ProductName	VarChar(40)	Product Name		No
ProductPrice	Price	Product Price		No
OrderProductQty	Numeric(4.0)	Order Product Qty		No
OrderProductTotal	Price	Order Product Total	ProductPrice*OrderPro...	
OrderTotal	Price	Order Total		No

A 'Formula Editor' dialog box is open, showing the formula: `sum(OrderProductTotal)`. The dialog has 'OK' and 'Cancel' buttons at the bottom.

3. VIEWING THE DATA MODEL

GeneXus will automatically generate the best data model to represent the Transactions created by the GeneXus Analyst.

Whenever you click on Save or Run, GeneXus will infer the optimal data model (3rd normal form with no redundancies) to support the end user entities represented by the Transaction objects. Based on this data model, GeneXus will generate a physical database for the target DBMS defined in its Environment.

To view the data model created by GeneXus, I select the upper menu View / Tables.

The screenshot shows the KB Explorer interface with the GeneXus data model structure. The left pane shows the project hierarchy for 'DemoGX16', including 'Root Module', 'GeneXus', 'Domains', 'References', 'Tables', 'Order', 'OrderProduct', 'Customization', and 'Documentation'. The main pane displays the structure of two tables: 'Order' and 'OrderProduct'.

Name	Type	Description
Order Structure		Order
OrderId	Numeric(4.0)	Order Id
OrderDate	Date	Order Date
CustomerId	Numeric(4.0)	Customer Id
CustomerName	VarChar(40)	Customer Name
Logical Attributes		
OrderTotal	Price	Order Total

Name	Type	Description
OrderProduct Structure		Product
OrderId	Numeric(4.0)	Order Id
ProductId	Numeric(4.0)	Product Id
ProductName	VarChar(40)	Product Name
ProductPrice	Price	Product Price
OrderProductQty	Numeric(4.0)	Order Product Qty
Logical Attributes		
OrderProductTotal	Price	Order Product Total

As you can see, GeneXus has automatically created a normalized data model, with two tables to support the Order Transaction object:

- Order (order header).
- OrderProduct (order lines).

This brings time and cost benefits thanks to the automation achieved with artificial intelligence.

COST SAVING: AI-BASED AUTOMATION.

4. VIEWING THE WEB FORM

Before running the application, note that the web form was automatically created by GeneXus when the transaction structure was defined. This web form will be displayed to the end user when executing the application in a web platform. It will allow users to enter, change, and delete records as we will see in a few minutes.

These forms can be customized by the business analyst at any time.

Web forms are built in an abstract web editor that will allow creating Responsive Web Applications.

Order X

Structure | Web Form | Win Form | Rules | Events | Variables | Help | Documentation | Patterns

FormButtons | Confirm | Cancel | Delete

MainTable

Order

<ErrorViewer: ErrorViewer>

<Toolbar>

Id OrderId

Date OrderDate

Id CustomerId

Name CustomerName

Product

GRID

Id	Name	Price	Product Qty	Product Total
ProductId	ProductName	ProductPrice	OrderProductQty	OrderProductTotal

Total OrderTotal

<FormButtons>

The GeneXus analyst doesn't have to program any of these actions because they are implicit in the Transaction logic. GeneXus will automatically generate the native code corresponding to the platform selected.

Remember that when defining GeneXus Transactions, we are:

Explicitly: describing the user interface for capturing and displaying data.

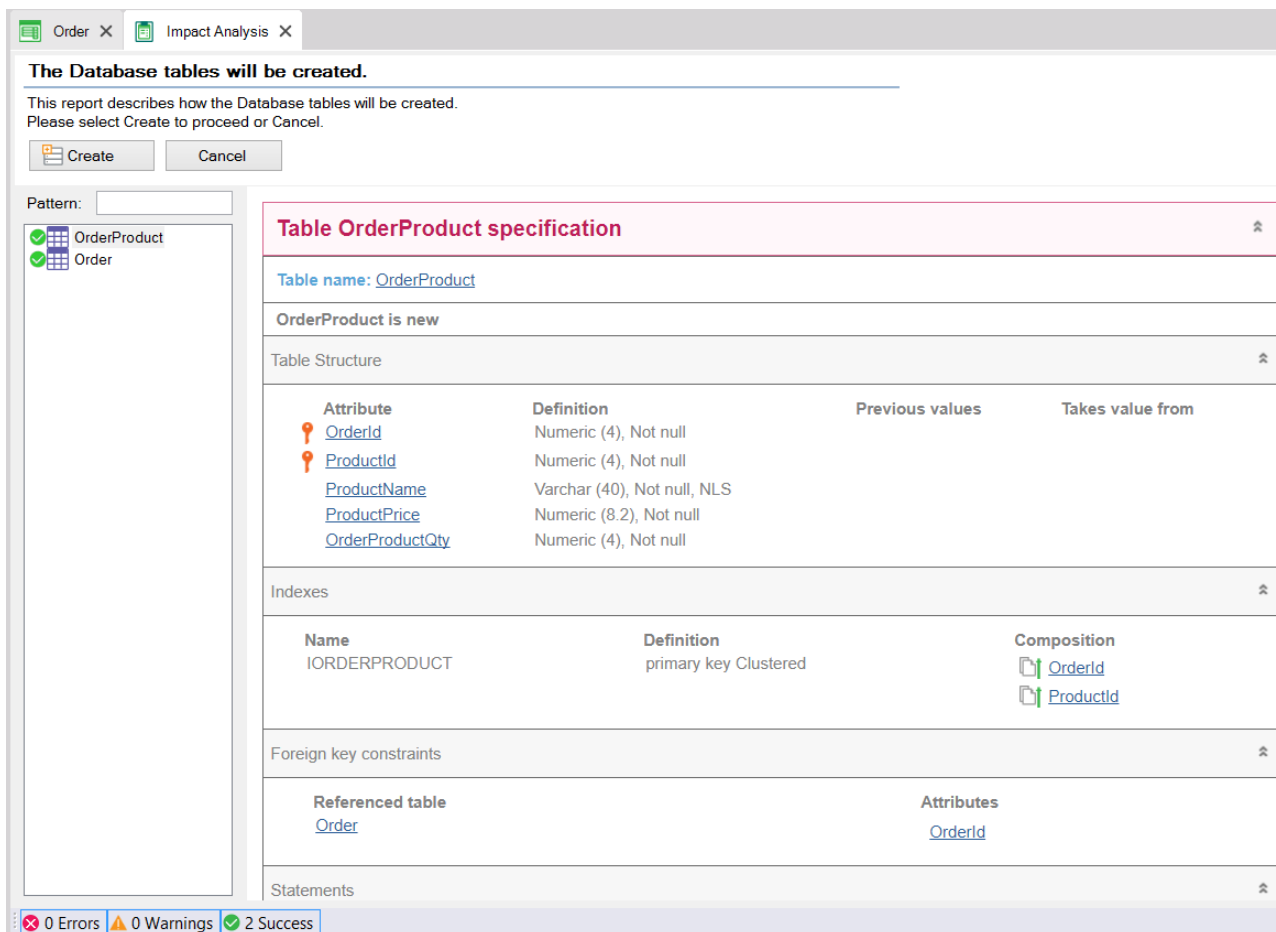
Implicitly: designing the application data model (tables, indexes, etc.).

The fact that GeneXus generates all this automatically contributes to its high productivity.

HIGHLY PRODUCTIVE: COMPLETENESS

5. RUNNING THE APPLICATION FOR THE FIRST TIME

To run the application, I press F5 or select RUN in the Toolbar.



This screen that appears here is the database creation report. GeneXus shows the scripts that will be run on the database.

I click on “Create.”

Note: Use the time it takes to run the application to reinforce that GeneXus will generate all the programs needed to run the web application, as well as all the programs to create and maintain the normalized database.

When this process is complete, a web browser will be opened and a menu will be loaded with the objects created in GeneXus.

The Developer Menu is an XML file that includes all the executable objects. It is a complementary menu for the GeneXus Analyst to try the application.

I click on the Order option to enter some purchase order records.

Order

Id

Date

Id

Name

Product

		Id Name	Price	Product Qty	Product Total
×	<input type="text" value="2"/>	<input type="text" value="Pendrive"/>	<input type="text" value="169.00"/>	<input type="text" value="3"/>	<input type="text" value="507.00"/>
×	<input type="text" value="4"/>	<input type="text" value="Wallet"/>	<input type="text" value="340.00"/>	<input type="text" value="1"/>	<input type="text" value="340.00"/>
×	<input type="text" value="1"/>	<input type="text" value="Pen with holder"/>	<input type="text" value="72.00"/>	<input type="text" value="2"/>	<input type="text" value="144.00"/>
	<input type="text" value="0"/>	<input type="text" value=""/>	<input type="text" value="0.00"/>	<input type="text" value="0"/>	<input type="text" value="0.00"/>
	<input type="text" value="0"/>	<input type="text" value=""/>	<input type="text" value="0.00"/>	<input type="text" value="0"/>	<input type="text" value="0.00"/>
[New row]					
Total			<input type="text" value="991.00"/>		

GeneXus has generated the programs with the latest web technologies: Full Ajax, HTML5, CSS3, Bootstrap, Responsive web design, etc. This is one of the reasons why GeneXus is highly productive.

HIGHLY PRODUCTIVE: COMPLETENESS

As a conclusion of what we've just seen, we can also say that GeneXus allows us to follow an agile development process or even a more traditional one (such as cascading) if we wish.

If we need to deliver more products to the client in a short period of time as suggested by agile methodologies, what we've just built can already be put into production and add value to the client (it is a fully functional application). It's worth pointing out that this is by no means a disposable prototype; in fact, iterations will be made and increments will be delivered to the customer using this development as a basis.

INCREMENTAL APPROACH: AGILE / ITERATIVE

So far, we've created an application based on the existing knowledge. Now, I'll show you how to maintain an application when reality changes or more information becomes available, simply by editing the existing objects and/or adding new ones.

To do so, I'll add some business rules.

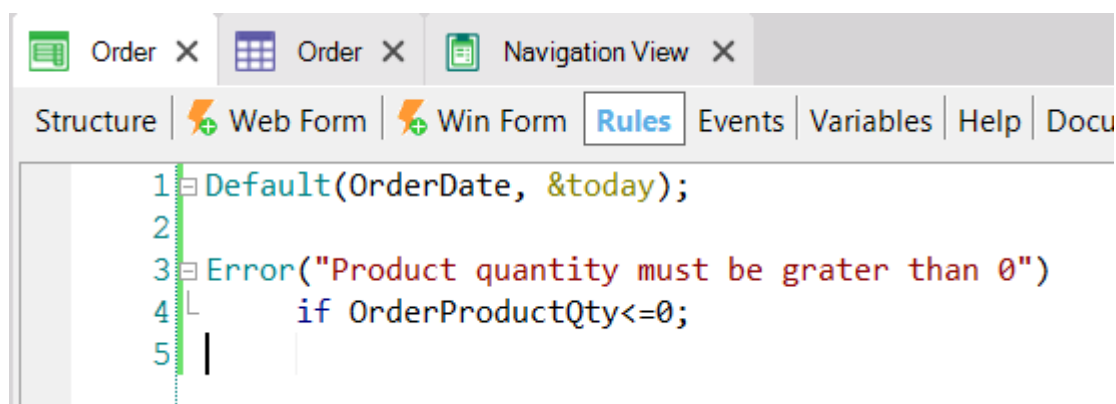
GeneXus rules are the means to define the business logic associated with each object. They are written in a declarative manner, and GeneXus intelligently decides which rule to apply and when to apply it.

Rules play a very important role in Transaction objects because they make it possible to describe their behavior; for example: assigning default values, defining data controls, etc.

Suppose that your client asks you that the date of a purchase order should be the current date by default. To do so, I will:

1. Select the Rules tab in the Order Transaction.
2. Add the Default rule from the upper menu: Insert / Default.
3. Complete it in this way: `Default(OrderDate, &today);`

Now, I'll add another simple rule to show an error if the product quantity is 0.



```
1 Default(OrderDate, &today);
2
3 Error("Product quantity must be grater than 0")
4     if OrderProductQty<=0;
5 |
```

I'll save the changes to see them in action.

« < > » SELECT

Id

Date

Id

Name

Product

	Id	Name	Price	Product Qty	Product Total
×	1	Coke Pack of Cans x12	10.00	<input type="text" value="0"/>	<input type="text" value="0.00"/>
	0		0.00	<input type="text" value="0"/>	<input type="text" value="0.00"/>
	0		0.00	<input type="text" value="0"/>	<input type="text" value="0.00"/>
	0		0.00	<input type="text" value="0"/>	<input type="text" value="0.00"/>
	0		0.00	<input type="text" value="0"/>	<input type="text" value="0.00"/>
	[New row]				
	Total		0.00		

CONFIRM CANCEL

This shows how easy it is to describe business rules in GeneXus.

EASY TO: USE

7. CREATING THE PRODUCT TRANSACTION

In this last interaction with the form, it was odd for us to have to type the product names all over again. Wouldn't it be better to have a catalog of products with their definition and price? This observation is correct, so we will change our system to have a catalog of products.

In the current modeling, the product was part of the Order Transaction, but we understand that these entities should be independent from the order, and therefore they should be defined as a Transaction.

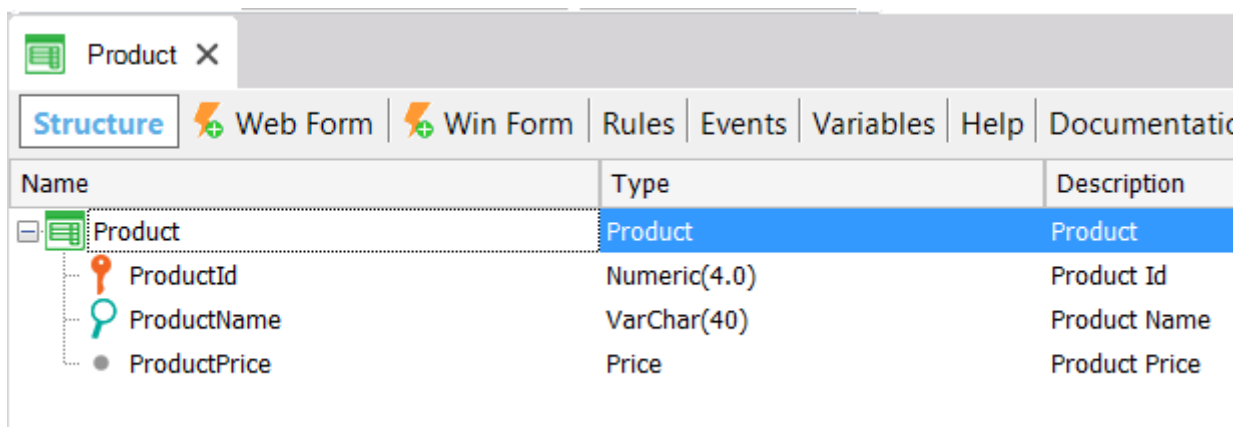
There is no problem with this, as GeneXus will maintain the normalized database based on the transaction schema.

So, I'll create the Product Transaction.

The product details entered in the Order Transaction are as follows:

- ProductId

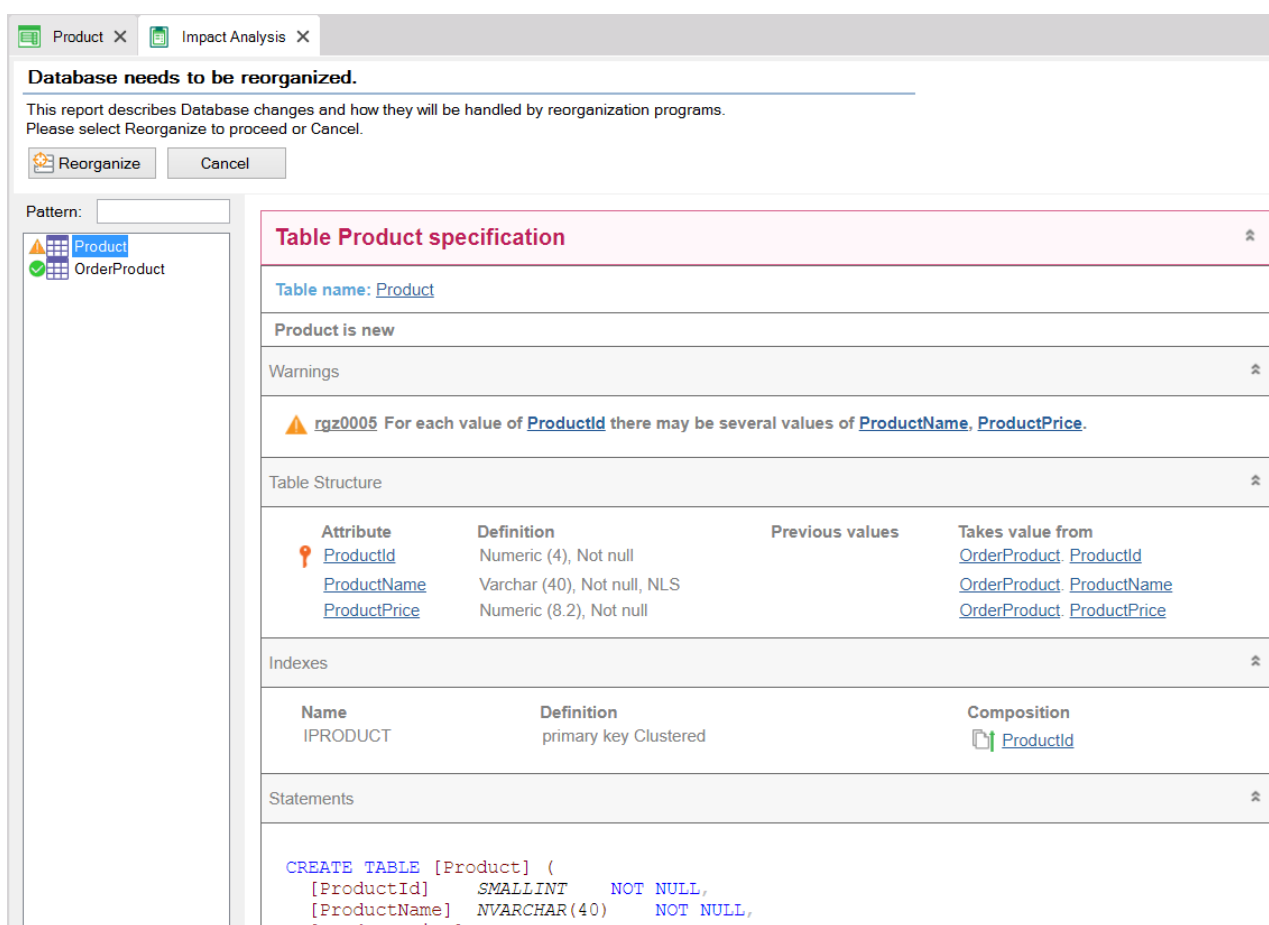
- ProductName
- ProductPrice



Name	Type	Description
Product	Product	Product
ProductId	Numeric(4.0)	Product Id
ProductName	VarChar(40)	Product Name
ProductPrice	Price	Product Price

Note that it wasn't necessary to define the data type of each one of the attributes, because GeneXus realized that they had already been defined in the Order transaction.

I save and click on Run.



Database needs to be reorganized.

This report describes Database changes and how they will be handled by reorganization programs.
Please select Reorganize to proceed or Cancel.

Reorganize Cancel

Pattern:

Product
OrderProduct

Table Product specification

Table name: [Product](#)

Product is new

Warnings

rgz0005 For each value of [ProductId](#) there may be several values of [ProductName](#), [ProductPrice](#).

Table Structure

Attribute	Definition	Previous values	Takes value from
ProductId	Numeric (4), Not null		OrderProduct . ProductId
ProductName	Varchar (40), Not null, NLS		OrderProduct . ProductName
ProductPrice	Numeric (8.2), Not null		OrderProduct . ProductPrice

Indexes

Name	Definition	Composition
IPRODUCT	primary key Clustered	ProductId

Statements

```
CREATE TABLE [Product] (
  [ProductId] SMALLINT NOT NULL,
  [ProductName] NVARCHAR(40) NOT NULL,
  [ProductPrice] SMALLMONEY NOT NULL
```


GeneXus makes an impact analysis and examines the changes that have to be made to the database according to the changes made to the transactions. Then, it indicates, for each object, the necessary changes.

Note that GeneXus indicates that the Product table has to be created and warns us (with a specification warning) that for the same ProductId there may be several ProductName and ProductPrice values.

It also indicates that it will have to delete the ProductName and ProductPrice attributes from the Order transaction, because they will now be part of the Product transaction and can be inferred through ProductId. Remember that GeneXus maintains a normalized database, and that's why it wouldn't be possible to have a secondary attribute in more than one physical table.

Pattern:

Product
OrderProduct

Table OrderProduct specification

Table name: [OrderProduct](#)

OrderProduct needs conversion

Table Structure

Attribute	Definition	Previous values	Takes value from
OrderId	Numeric (4), Not null		OrderProduct. OrderId
ProductId	Numeric (4), Not null		OrderProduct. ProductId
OrderProductQty	Numeric (4), Not null		OrderProduct. OrderProductQty
Del ProductName	Varchar (40), Not null, NLS		
Del ProductPrice	Numeric (8.2), Not null		

Once the data model reorganization is complete, when browsing the Product transaction records we can see the details of the products we had entered through the Order. This automatic feature of GeneXus makes it possible to change the data model. GeneXus automatically changed the structure and migrated the data from one table to another.

But... what happened with the Product information that was stored in the Order Transaction?

Name	Type	Description
Order	Order	Order
OrderId	Numeric(4.0)	Order Id
OrderDate	Date	Order Date
CustomerId	Numeric(4.0)	Customer Id
CustomerName	VarChar(40)	Customer Name
Product	Product	Product
ProductId	Numeric(4.0)	Product Id
ProductName	VarChar(40)	Product Name
ProductPrice	Price	Product Price
OrderProductQty	Numeric(4.0)	Order Product Qty
OrderProductTotal	Price	Order Product Total
OrderTotal	Price	Order Total

Note that the attributes that now physically belong to the Product table are marked with a "down arrow" in the Order Transaction. This indicates that they are being inferred from the ProductId attribute, and demonstrates that GeneXus automatically maintains a normalized database.

EASY TO: MAINTAIN **EASY TO: CHANGE**

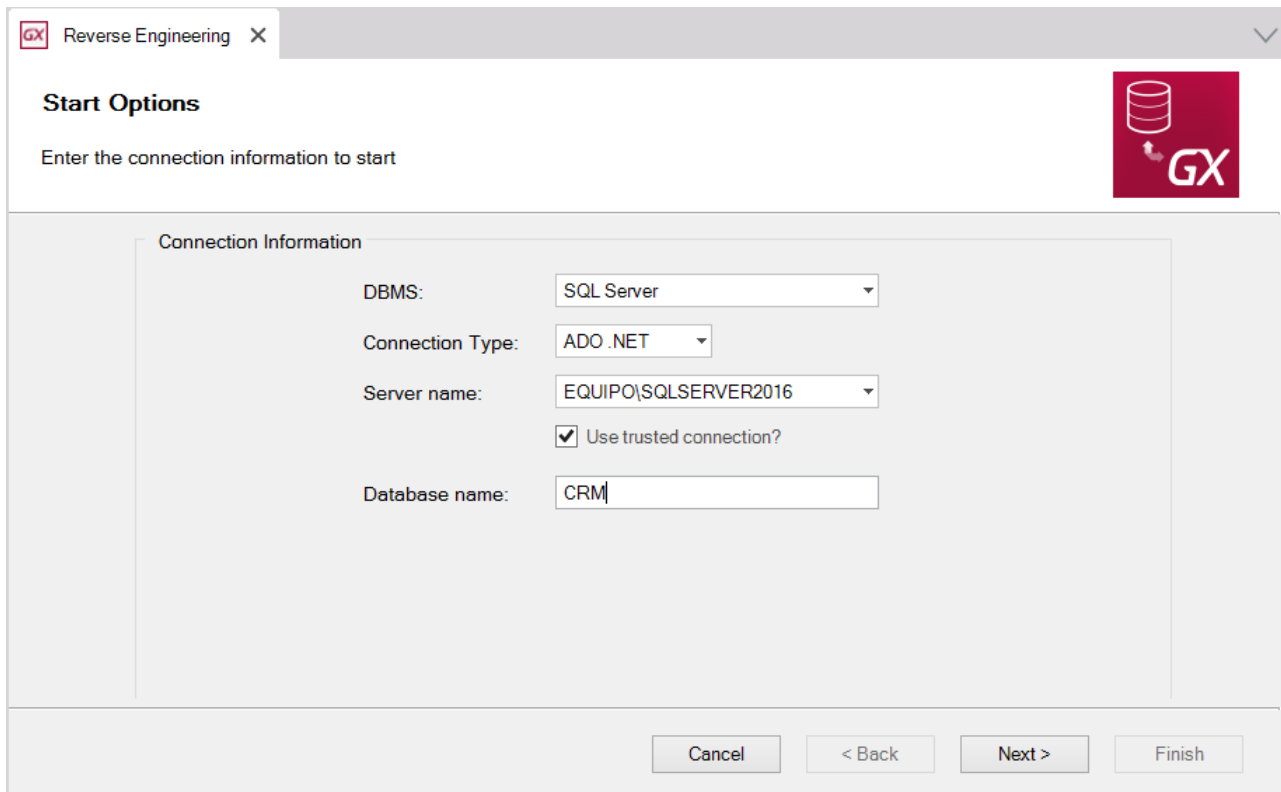
8. CREATING THE CUSTOMER TRANSACTION

Advancing a little further with the inquiry, we are told that the company has a customer directory, so it is not correct to store customer data as belonging to the Order transaction. Instead, they should be modeled in an independent transaction.

Customer details are recorded in the company's CRM system and should be taken from there to record the purchase orders.

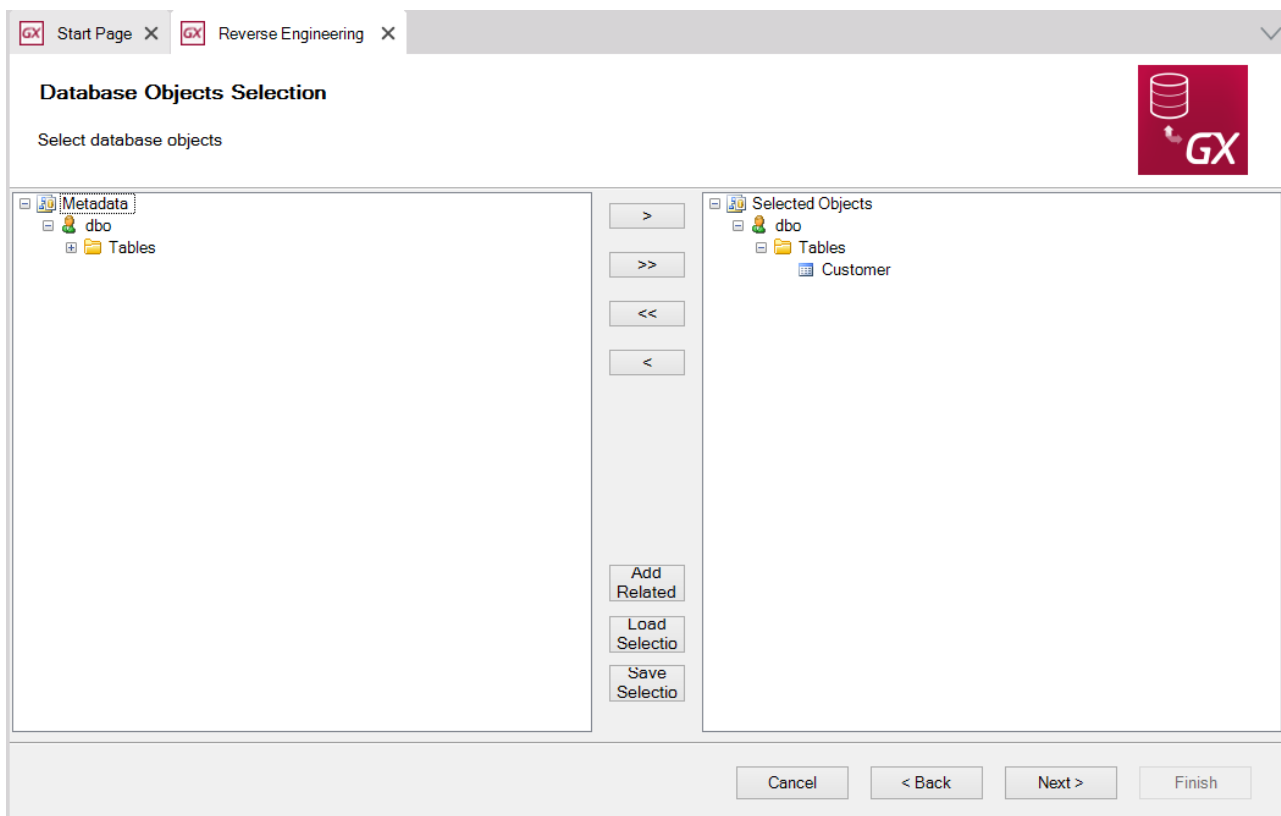
In sum, GeneXus makes it possible to integrate with any external system and take data from its database to use it in the system to be developed.

To do so, I select *Tools / Database Reverse Engineering* and enter the name of the database server, and the name of the database to integrate with.

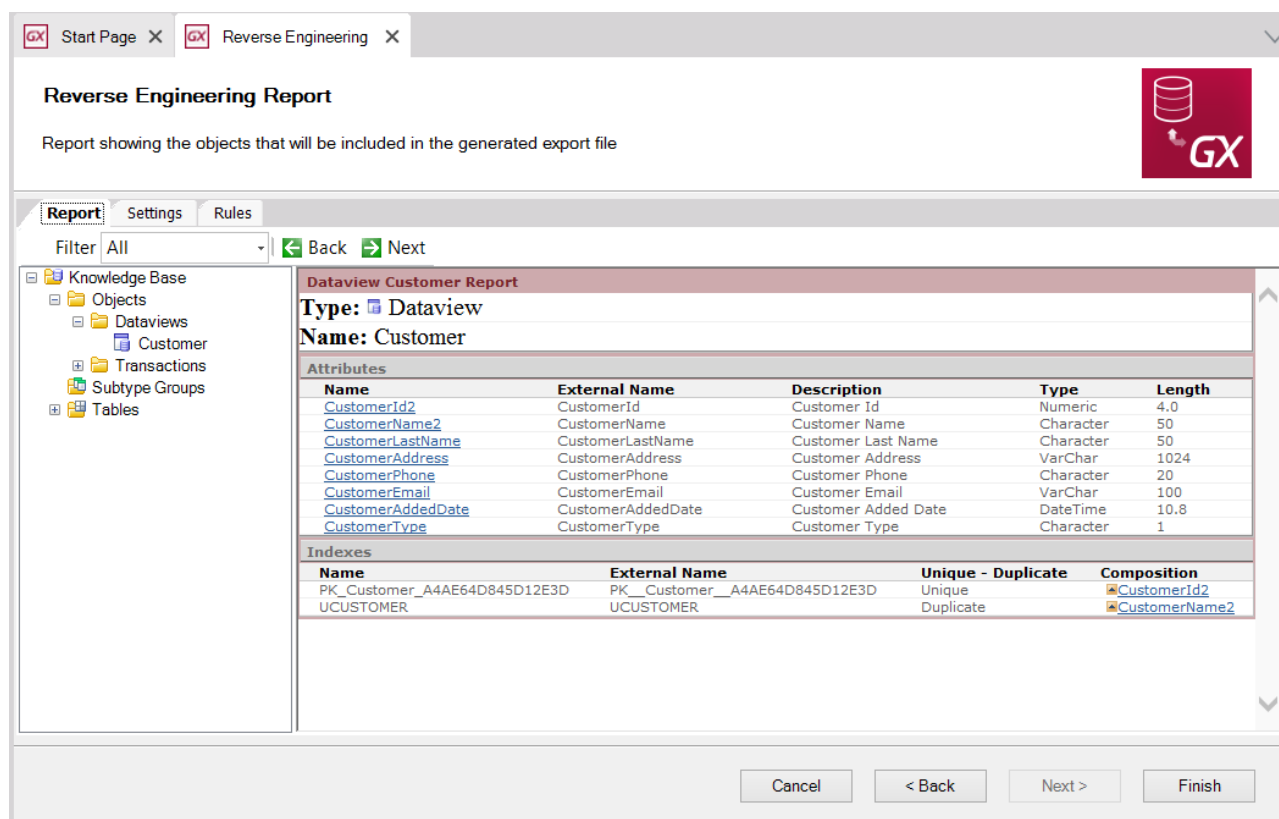


Clicking on *Next* opens a list with all the database tables for me to select the ones I need.

I select the Customer table.

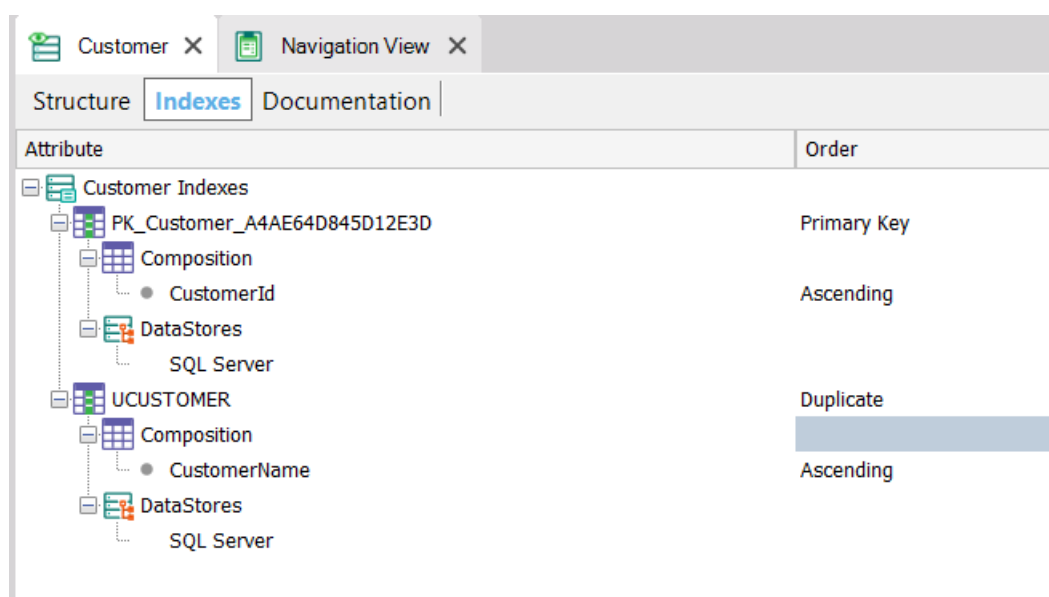


By clicking on *Next*, GeneXus indicates that it will create the Customer Data View.



Note that it has added a number 2 to CustomerId and CustomerName, because they already existed in my model. If I want them to be the same, I need to indicate it explicitly. To do so, I select the Customer Transaction and in the Customer Data View I change these attribute names.

Also, I need to change the name of the attributes in the index definition.



Lastly, I'll delete the attributes I won't use from the Customer Transaction and the Customer Data View. I leave only CustomerId, CustomerName and CustomerLastName.

The screenshot shows two windows in the GeneXus IDE. The top window displays the structure of the 'Customer' entity with the following table:

Name	Type	Description	Formula	Nullable
Customer	Customer	Customer		
CustomerId	Numeric(4.0)	Customer Id		No
CustomerName	VarChar(40)	Customer Name		No
CustomerLastName	Character(50)	Customer Last Name		No

The bottom window shows the 'Customer Structure' data view with the following table:

Internal Name	External Name	Description
Customer Structure		Customer
Composition		
CustomerId	CustomerId	Customer Id
CustomerName	CustomerName	Customer Name
CustomerLastName	CustomerLastName	Customer Last Name
DataStores		
SQL Server		SQL Server

Note: When the names are changed, the Data View loses the content of the *Associated table* property. Assign Customer to it.

The screenshot shows the 'Customer Structure' data view window with the 'Properties' window open on the right. The 'Data View: Customer' properties are as follows:

Property	Value
Associated table	Customer
Datastore	DataStore2 (SQL Server)
Description	Customer
Module/Folder	Root Module
Name	Customer
Object Visibility	Public
Qualified Name	Customer

I click on RUN. Note that the Impact Analysis indicates that the Customer transaction is associated with the Customer Data View, and that it won't be reorganized; that is, an associated physical table will not be created.

After running it, we can see that there is a Customer transaction that takes data from the Customer table in the CRM system.

Customer	
« < > » SELECT	
Id	3
Name	Vanessa
Customer Last Name	Goldberg
CONFIRM CANCEL DELETE	

In this way, we have easily integrated with an external system regardless of the technology or the target database management system used.

EASY TO: INTEGRATE

9. PRODUCT CATALOG

Now I'll show how we can generate applications with richer and more user-friendly interfaces. I'll apply the Work With for Web pattern and see its benefits in action.

Patterns allow defining and encapsulating behavior automatically, as well as replicating it massively based on a definition template.

Let's see an example.

Suppose that in addition to inserting, changing and deleting products through the transaction, we would like to see a product catalog and be able to filter them by different criteria, sort them, etc.

To make it more realistic, I'll add a photo to the product and apply the Work With for Web pattern.

Name	Type	Description	Formula	Nullable
Product	Product	Product		
ProductId	Numeric(4.0)	Product Id		No
ProductName	VarChar(40)	Product Name		No
ProductPrice	Price	Product Price		No
ProductImage	Image	Product Image		Yes

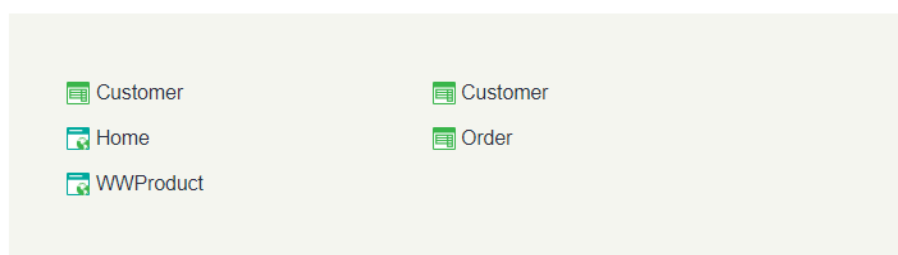
The Image data type is one of those we call “semantic domain”, and which have their behavior implicitly defined. For example, this data type will provide a file chooser in the form to upload an image. There are other semantic domains, such as Address, Phone, Email.

Note: When applying the Work With for Web pattern, provide an overview of how it is defined, indicating that it allows adding new orders and filters, customizing the columns, adding actions, etc.

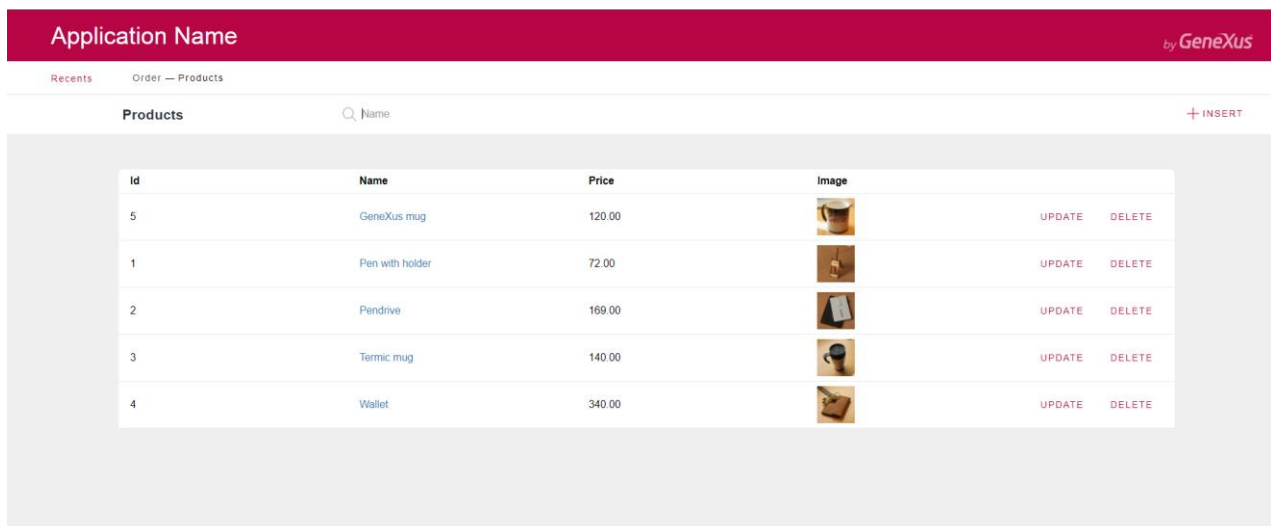
When running it, note that the shortcut to the Product transaction has changed and now it is accessed through WWProduct.

DEVELOPER MENU

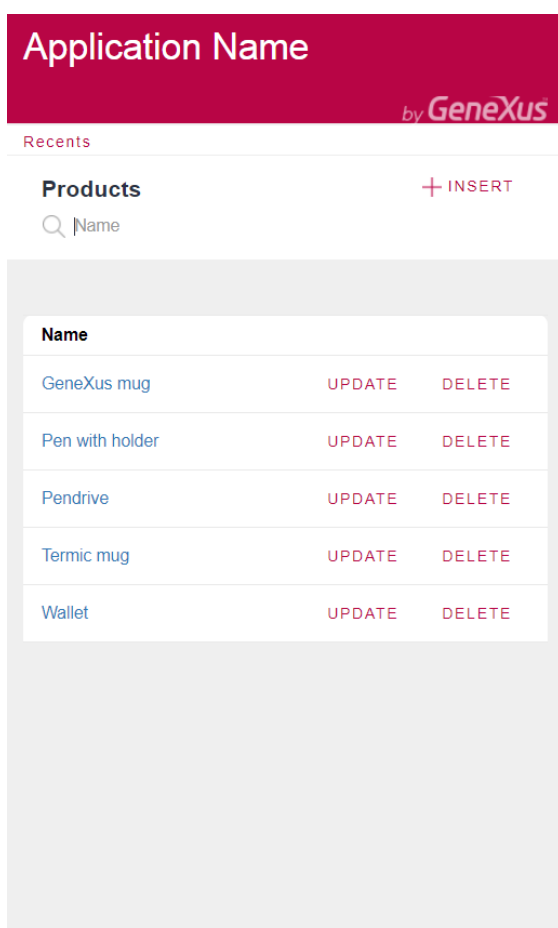
Browse Web Objects



Note that GeneXus has automatically generated a new web page with the list of all products, from where we can search, insert, edit, delete and view each product sheet.



If I shrink the browser, we can see that the application is responsive, adapting to the screen size available and prioritizing to show the most relevant information.



If I click on the name of a product, its sheet will be displayed. There, we will see all its information as well as the information of related entities. For example, a tab with the purchase orders in which this product was included.

Application Name
by GeneXus

Recent Order — Termic mug — Products — Pendrive

Product Information

← PRODUCTS

Name Pendrive


General

Order

UPDATE

DELETE

Id	2
Name	Pendrive
Price	169.00



This makes the application very user-friendly and easy to navigate, where all the information we need about the entity we are working with is one click away.

Take a few moments to realize that GeneXus has generated all this with a single click.

We've seen how GeneXus helps us create applications rich in user experience by applying artificial intelligence and automation, with a small cost for the developer.

COST SAVING: AI-BASED AUTOMATION.

10. MULTI-PLATFORM

Now, I'll generate an application for mobile devices with Android operating system. The application will be generated using native Android code.

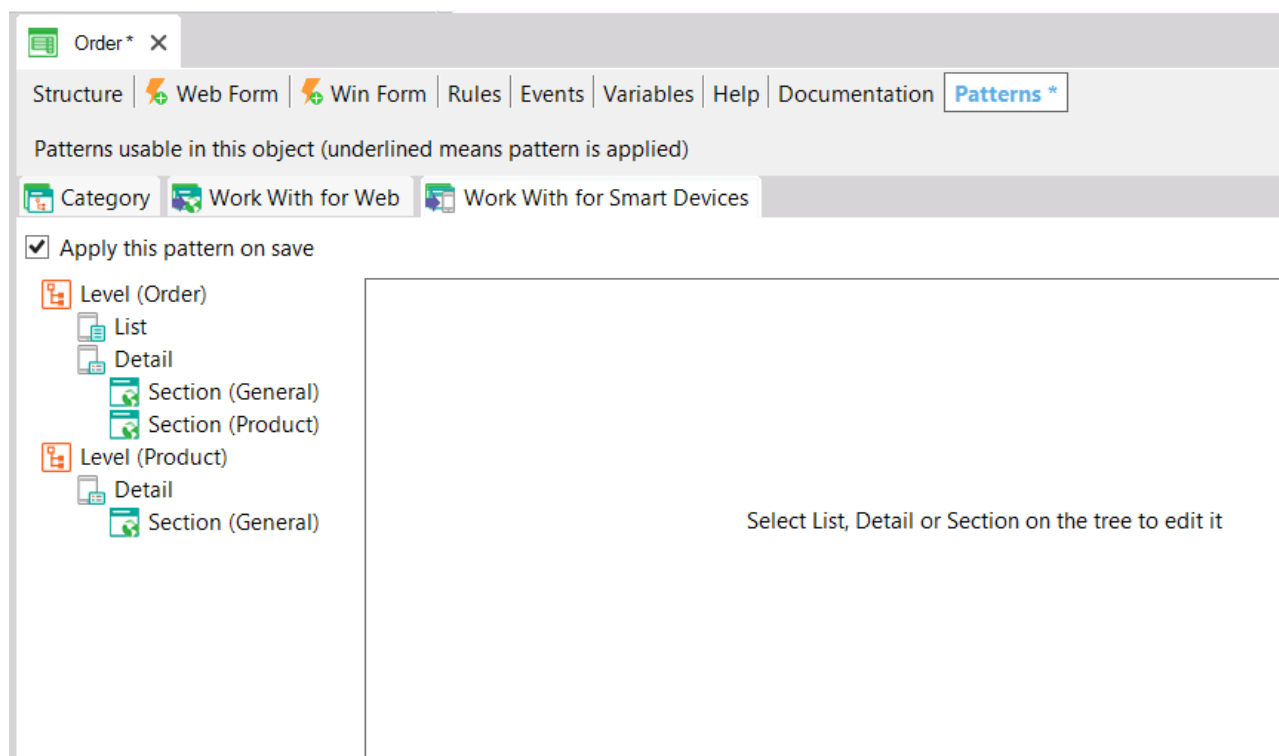
The mobile application should have the following screens:

- List of purchase orders arranged in descending order.

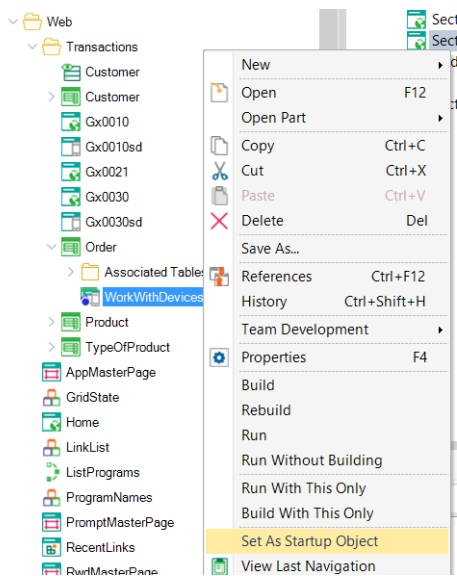
- For each purchase order, the list of products included with their subtotal and total values.
- Menu to access options.

I'll start with the list of purchase orders.

To do so, I select the Order transaction and apply the Work With for Smart Devices pattern.



After saving, I need to set a smart device object as Start Up Object. In this way, when prototyping, clicking on Run will automatically start the application in the emulator. I locate the Order transaction in the object explorer and right-click on WorkWithDevicesOrder to set it as Start Up Object.



Note: Indicate that if users have an Android device connected by USB to their computer, with USB debug setting enabled, clicking on RUN will automatically install the application on the device to try it there. This is very useful to make testing more realistic.

I click on Run to have GeneXus generate the application for Android.

Note: While the emulator is being started (it may take a few minutes), use this time to explain that GeneXus is generating all the programs, screens, and web services required to create the mobile application using native code; in this case, it is Android.

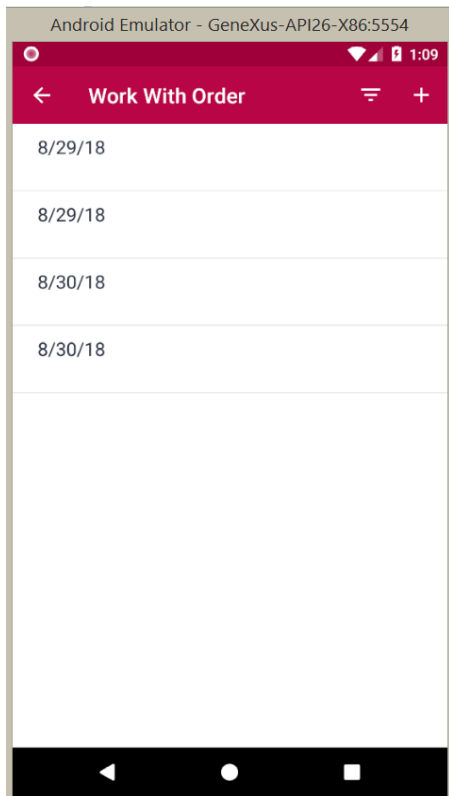
Also, highlight that since GeneXus generates the programs, it makes us independent from any given technology. For example, when iOS changed its generation language from Objective-C to Swift, while many had to rewrite their applications, GeneXus users didn't have to do anything because the generator was updated so that applications could be generated using Swift.

The same happens on the web, for instance, with versions of .Net and .Net Core.

This demonstrates the benefit of being FUTURE PROOF: TECHNOLOGY-INDEPENDENT as seen in the diagram.

Note that GeneXus has created a screen with the list of orders.

The date of each one of them is displayed by default.

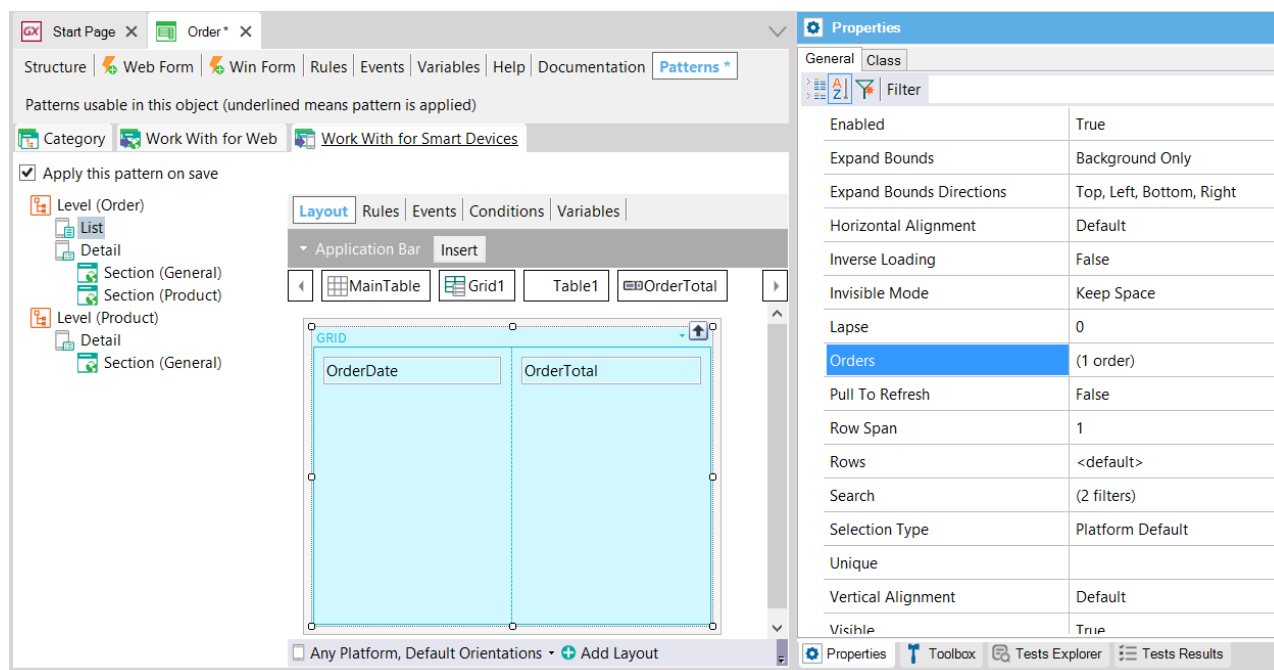


It is preferable that:

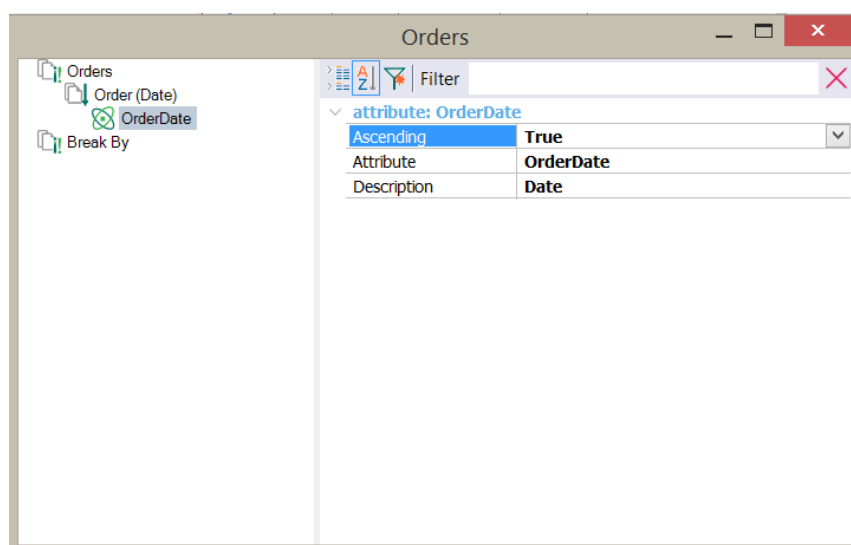
- The screen title is *Orders*.
- That the total of each order is displayed.
- That they are ordered by date in descending order.
- That the contents of a line are vertically aligned.

To make these changes, I will:

- Click on the List node of the Work With for Smart Devices instance of the Order transaction, and change the *Caption* property to *Orders*.
- Drag the OrderTotal attribute to the grid.
- Change the *LabelPosition* property of the OrderTotal attribute and call it *None* so that the description is not displayed.
- Click on the Grid and select the *Orders* property.



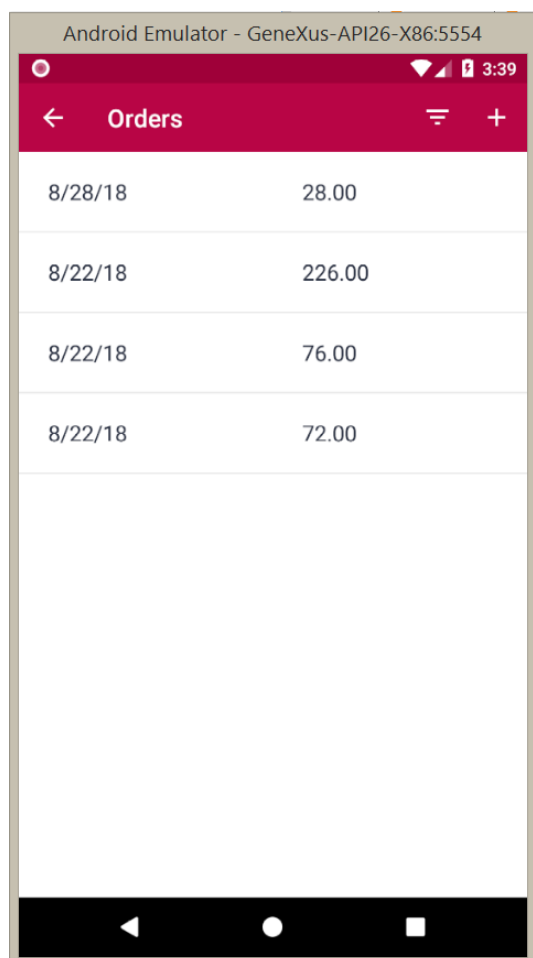
Clicking on it displays a screen to change the orders specified.



To fulfill this request, I change the value from *Ascending* a *False*.

Lastly, I change the *Vertical Alignment* property of both attributes to *Middle*.

I click on Run again to see the changes at runtime.



We've seen how easy it is to change the content of the screens. Now I'll advance a bit faster and apply the design sent by the designer.

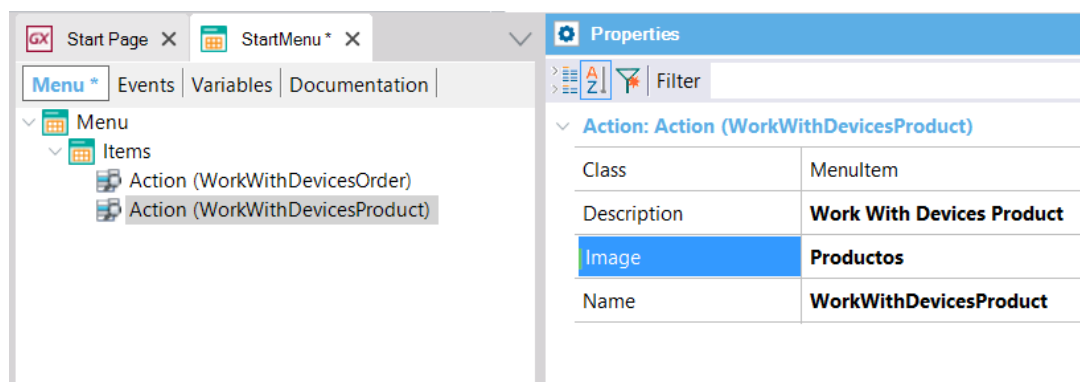
Note: Show slides with designs sent by the designer.

1. Menu

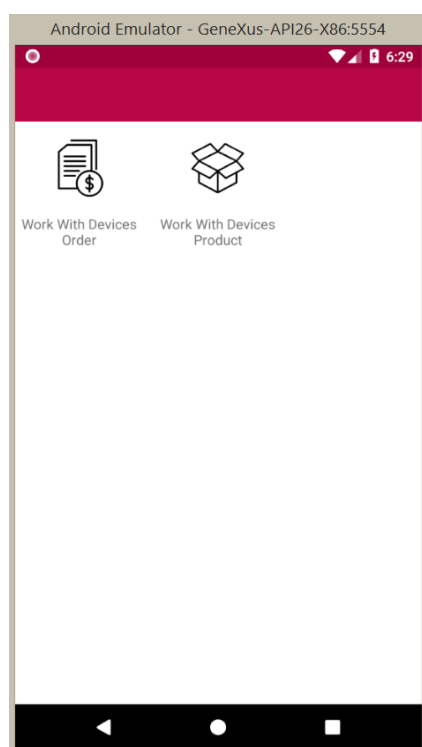
Note that the first screen is an access menu with options.

To this end, I will create in GeneXus a new object of Menu for Smart Devices type.

To do so, I'll add an Action to the menu for each access I want to have.



Also, I'll load an image for it in the Image property to include the icons sent by the designer.



2. List of purchase orders

Note that we want the total to be shown with the \$ sign, and that it makes sense to use the same criteria for all amounts in the system. So, I select the Price domain definition and add a prefix using the *Prefix* property.

Name	Type	Module	Description
EventAction	Numeric(4.0)	GeneXus	Event Action
EventStatus	Numeric(4.0)	GeneXus	Event Status
EventData	LongVarChar(2M)	GeneXus	Event Data
EventErrors	LongVarChar(2M)	GeneXus	Event Errors
ApplicationState	Numeric(1.0)	GeneXus	Application State
SynchronizationReceiv...	Numeric(4.0)	GeneXus	Synchronization Receive Result
RegionState	Numeric(1.0)	GeneXus	Region State
BeaconProximity	Numeric(1.0)	GeneXus	Beacon Proximity
MediaFinishReason	Numeric(4.0)	GeneXus	Media Finish Reason
HttpMethod	Character(7)	GeneXus	Http Method
HttpAuthenticationType	Numeric(4.0)	GeneXus	Http Authentication Type
CommonCallTarget	Character(20)	GeneXus	Common Call Target
BarcodeType	VarChar(40)	GeneXus	Barcode Type
FlexDirection	Character(20)	GeneXus	Flex Direction
FlexWrap	Character(20)	GeneXus	Flex Wrap
FlexJustifyContent	Character(20)	GeneXus	Flex Justify Content
FlexAlignItems	Character(20)	GeneXus	Flex Align Items
FlexAlignContent	Character(20)	GeneXus	Flex Align Content
Id	Numeric(4.0)	Root Module	Id
Price	Numeric(8.2)	Root Module	Price
Page	Numeric(4.0)	Root Module	Page
TabCode	Character(50)	Root Module	Tab Code

Left fill	Blank
Length	8
Module	Root Module
Name	Price
Notify Context Change	False
Object Visibility	Public
Picture	\$ ZZZZ9.99
Prefix	\$
Qualified Name	Price
Signed	False
Thousand separator	False
Tooltip Text	
Validation Failed Message	
Value range	
Width	10chr

Also, the order total must be shown in bold.

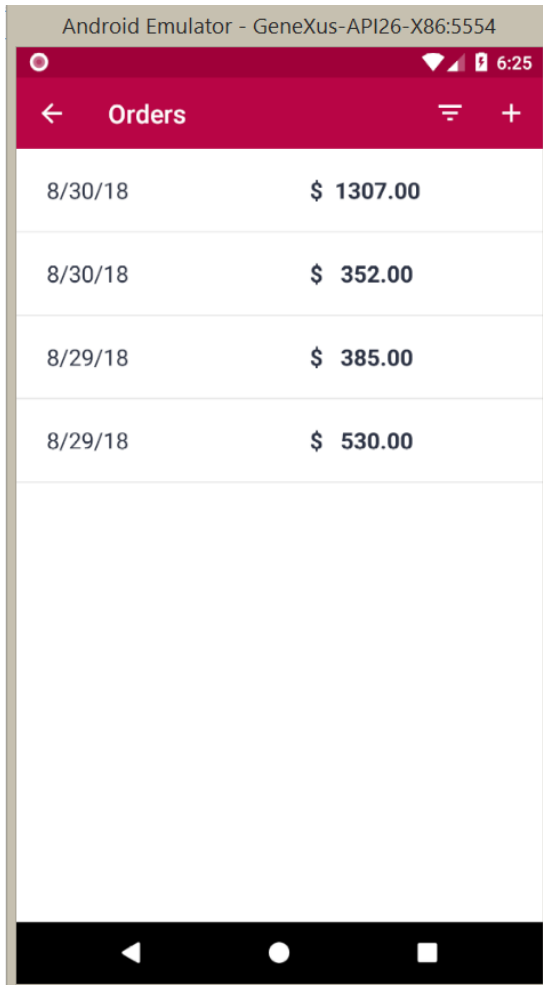
To do so, I open the object called CarmineAndroid (of Theme type). Themes encapsulate the definition of the components' appearance.

For each type of component (for example, Attributes, Tables, Grids, Buttons, etc.), there are several classes defined. I'll create a new attribute class with bold font to use it in the order total.

Next, I right-click on Attribute / New Class. GeneXus copies the definition of the parent class and allows customizing its appearance from the properties. I call this new class *AttributeBold* and change the *Font Weight* property to *Bold*.

Lastly, I select the OrderList object and change the OrderTotal attribute class so that it uses the *AttributeBold* class.

I click on Run to see the changes.



Android Emulator - GeneXus-API26-X86:5554

8/30/18	\$ 1307.00
8/30/18	\$ 352.00
8/29/18	\$ 385.00
8/29/18	\$ 530.00

Note that the changes have effectively been made and that this screen already meets the aesthetic requirements.

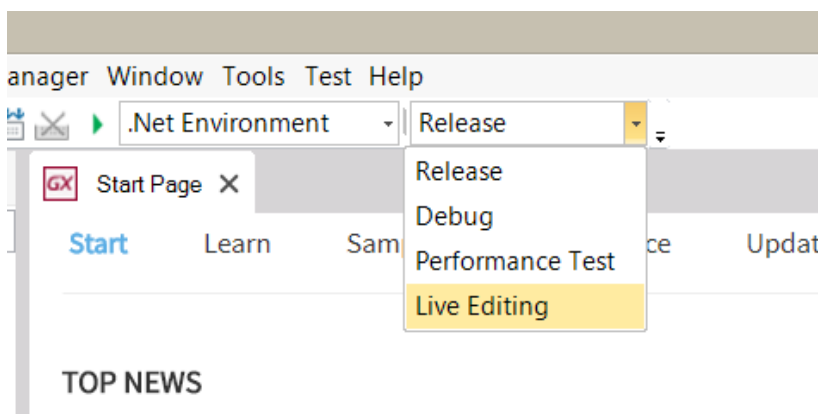
But we have an even more dynamic way of applying design features to screens.

GeneXus provides a mechanism called Live Editing that shows in real time (with no need to compile) the changes made to the design.

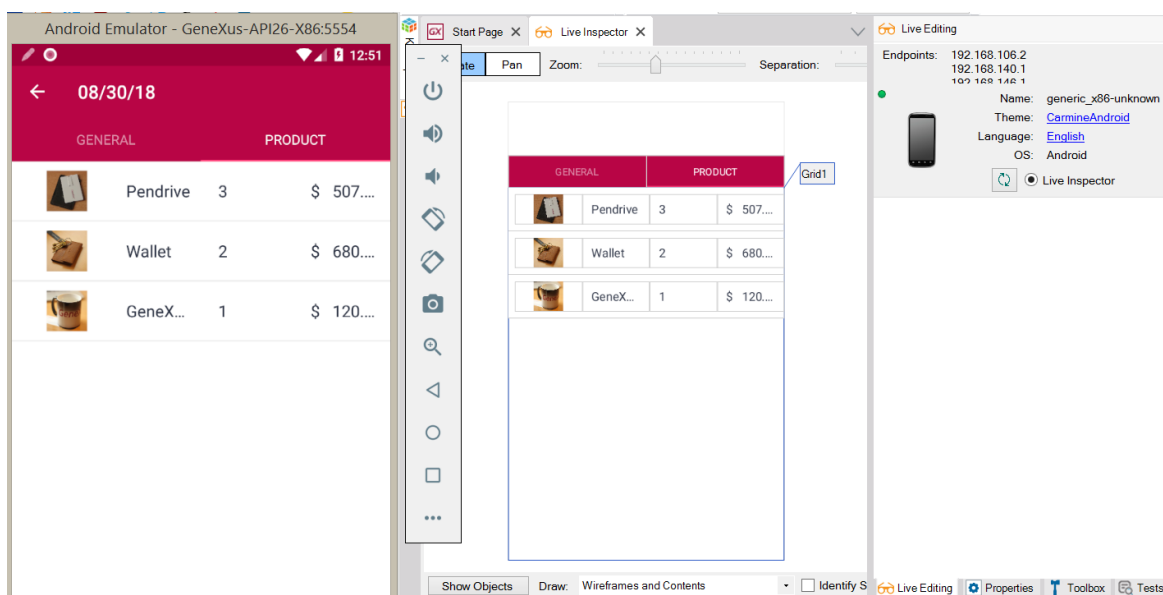
Let's try it with the following screen.

3. List of Products in a Purchase Order

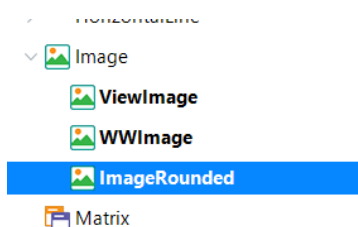
In the combo box located in the upper bar of the IDE I select Live Editing.



After doing this, a screen called Live Inspector will be opened to show the design I'm working on with the changes applied in real time.



The most important visual change to comply with the design of this screen is to make the images look rounded. To do so, I'll create a new theme class inside the Image node, call it ImageRounded,



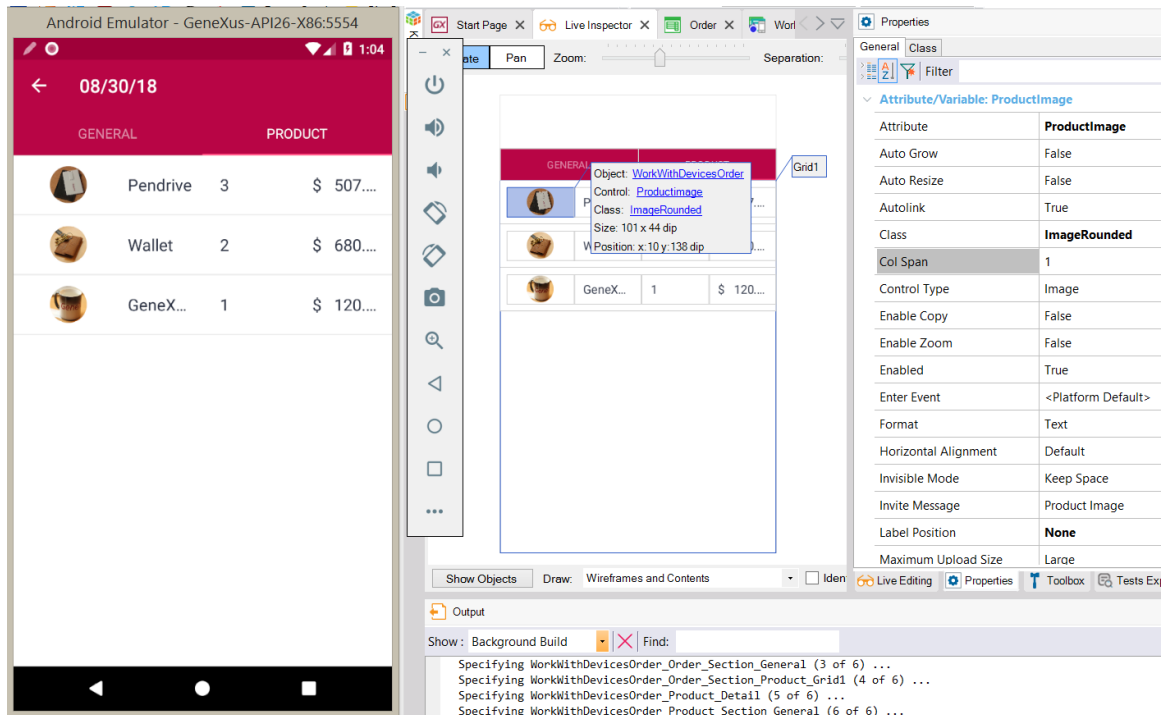
Max Horizontal Offset	0
Max Vertical Offset	0
Name	ImageRounded
No Accept Drag	
Padding	

and set the following properties:

- Height: 40dip
- Width: 40 dip

- Border Radius: 100 dip

I save the changes made to the Theme and assign the class to the image in the Live Inspector window to check if it looks as expected.



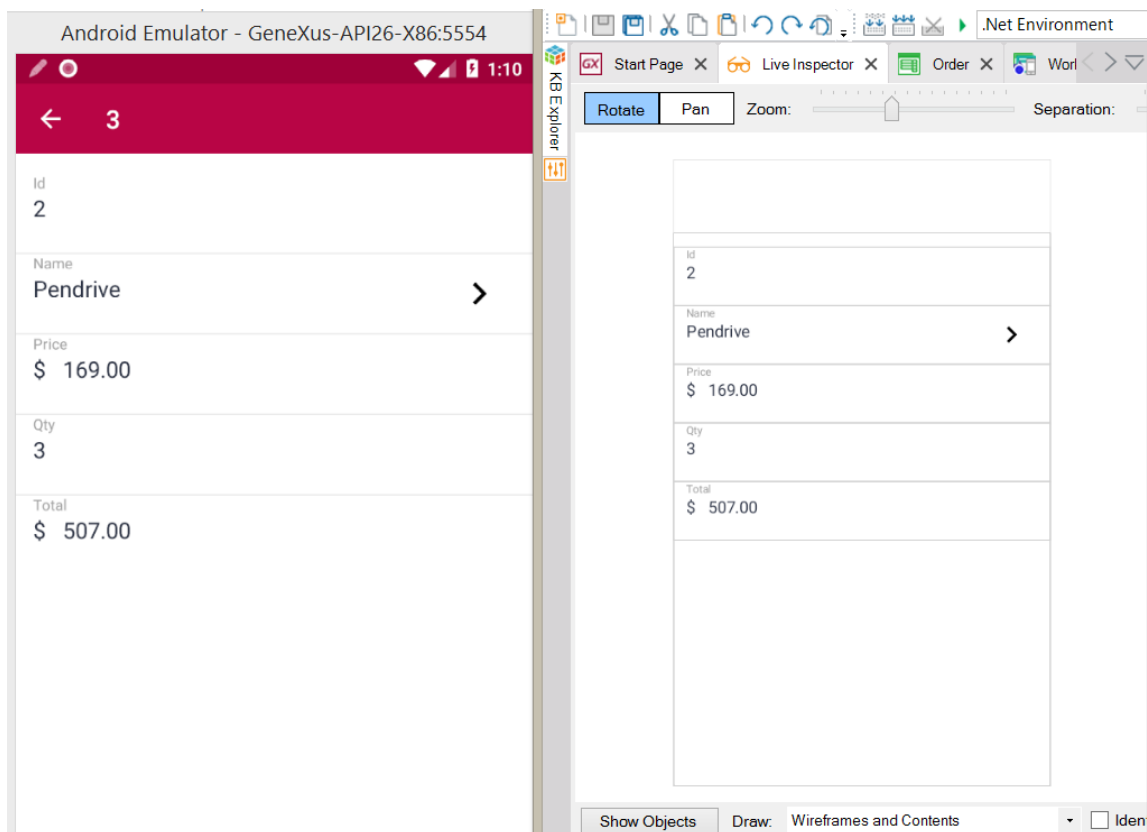
Note that the changes automatically applied are seen in the emulator.

The class assignment made in the Live Inspector is automatically applied in the corresponding GeneXus object.

It's worth mentioning that the Live Editing feature makes a bi-directional connection between the KB and the application at runtime.

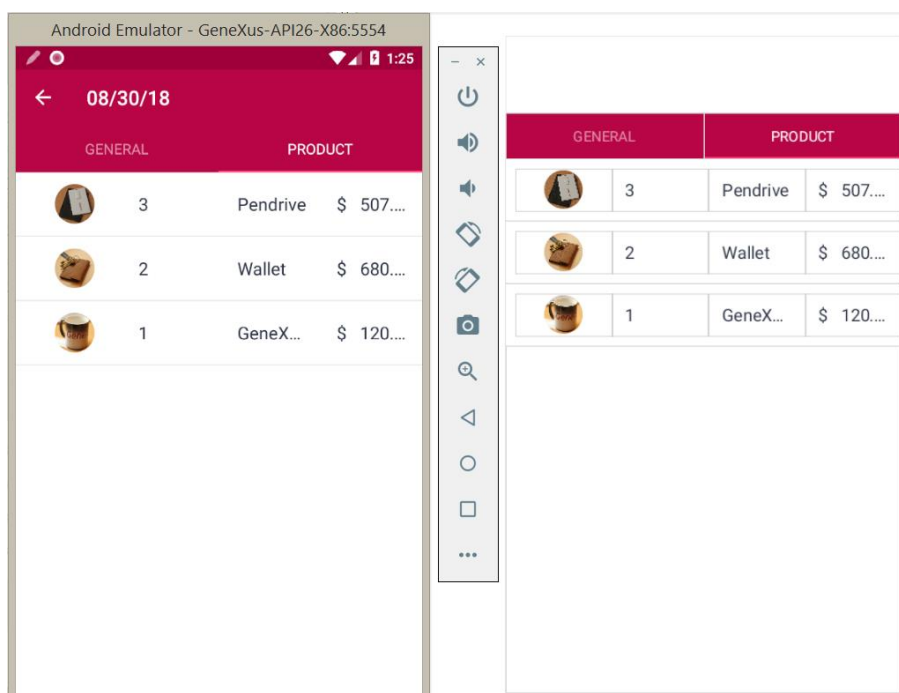
We've seen that the changes made to the KB are reflected in the application.

Now I'll browse the application, for example, by tapping on a product line to see its details. Note that the same screen is displayed in the Live Inspector.



This mechanism significantly accelerates the application of design features.

Lastly, I'll reorder the grid columns to display them in the order established by the designer.

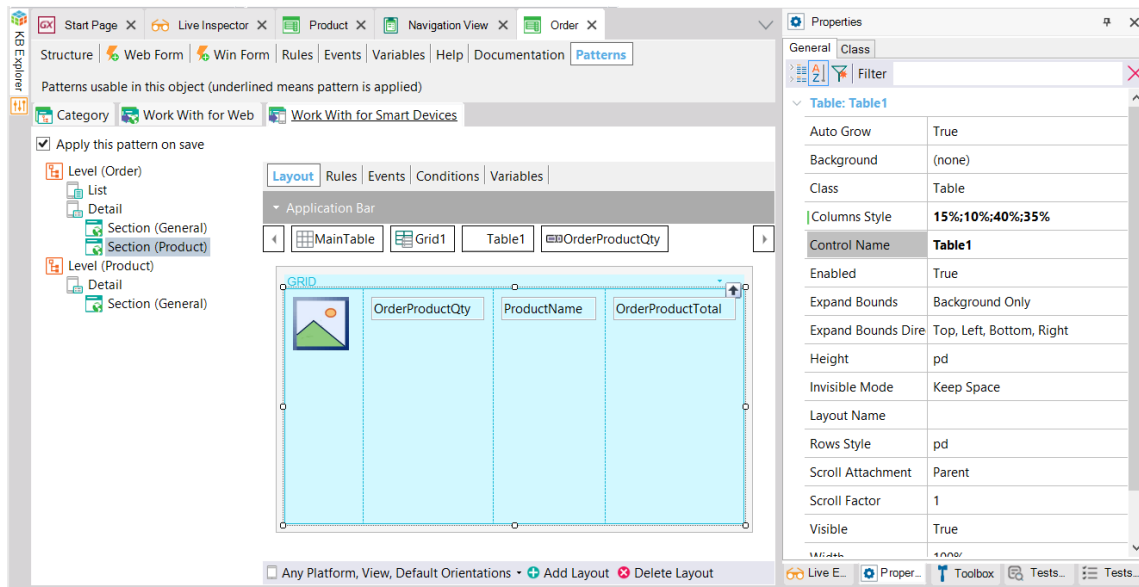


Note that this change in layout was also automatically reflected in the emulator (without compiling).

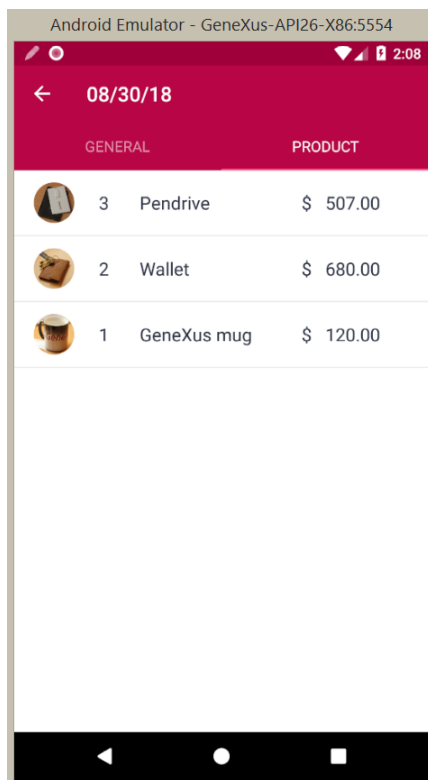
There's still one thing left to see the desired design, which is to adjust the width of the columns so that the product name is fully displayed and the price is not cut.

To achieve this, I click on the table containing the grid columns (Table1) and change the *Columns Style* property to better distribute the percentage for the width column.

I set the percentages: 15%; 10%; 40%; 35%, respectively.

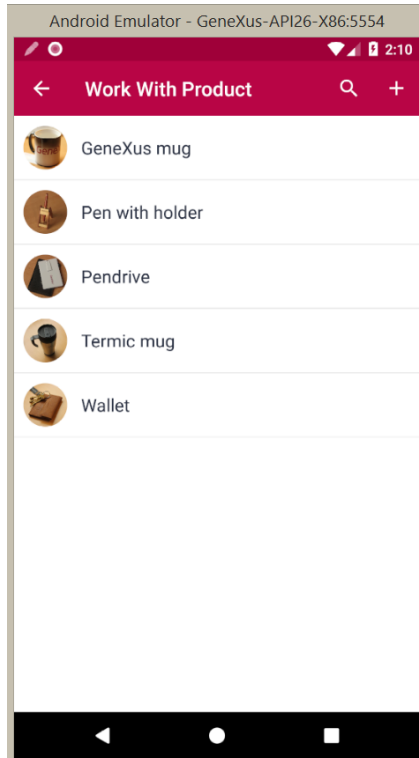


At the same time, we can see the changes applied in the emulator.



4. List of Products

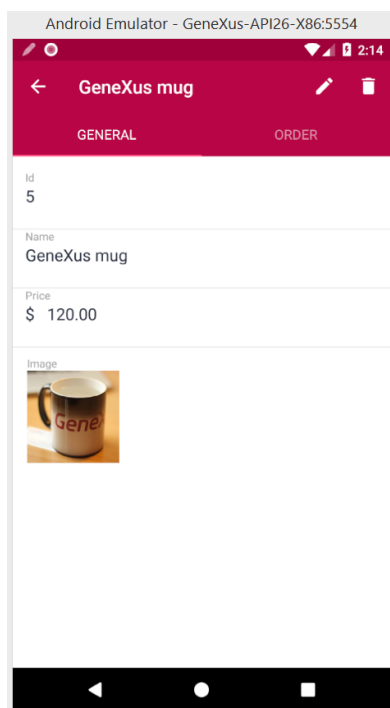
Let's see how far we are from the desired design on the product listing screen. I return to the main menu and select Work With Devices Product.



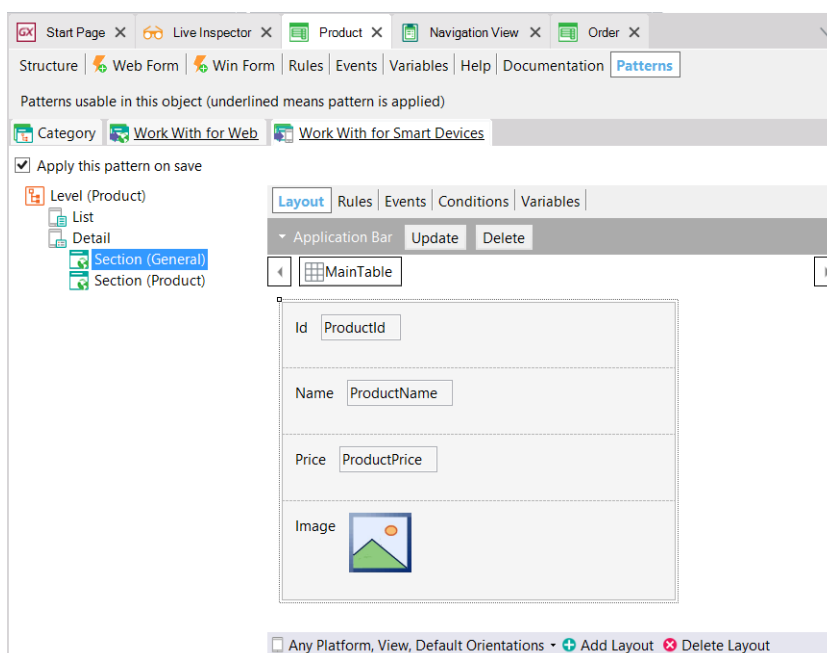
It would be enough to distribute the width of the columns a little better to give more space to the image and change the screen title. As they are things that I've already done in previous screens, I'm going to skip this customization and go to the last screen.

5. Product Sheet

The screen created by GeneXus by default looks as follows:



I open the Product transaction, Section General to edit this screen.

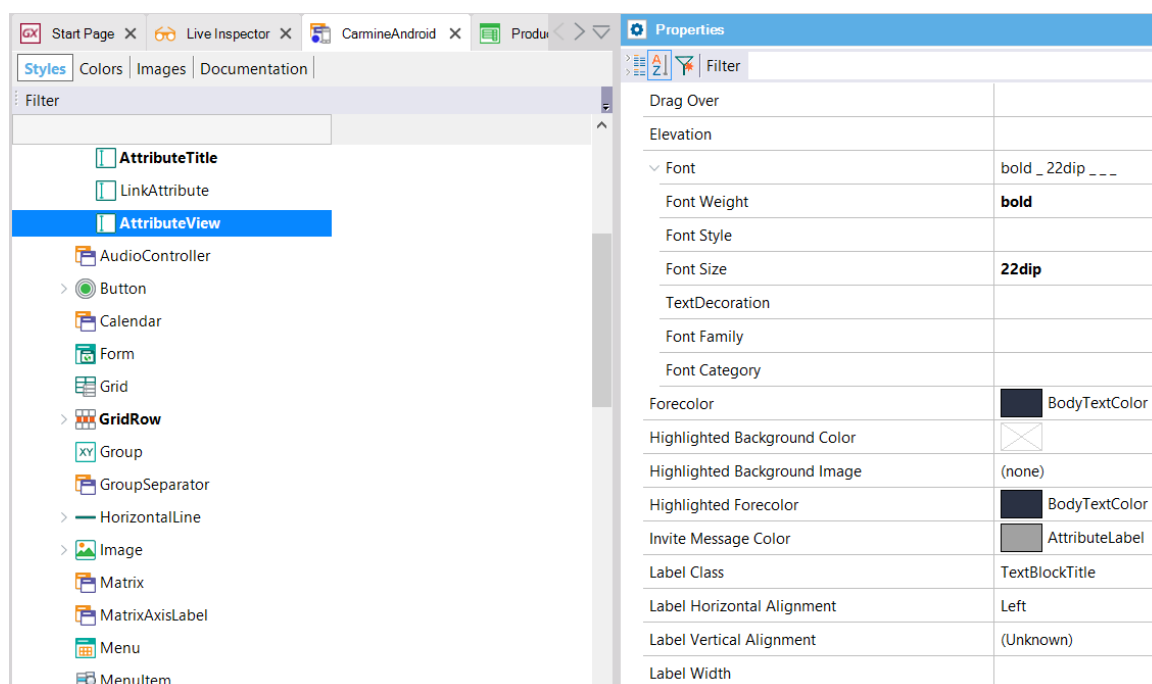


I'll make the following changes:

- Set the property *Label Position* = *None* for the image attribute and drag it to the upper part of the screen.
- Edit the properties of the theme class that the image is using (ImageView).
- Select the theme and change the following properties of the ImageView class:

- Width: (Use default, unspecified)
 - Height: (Use default, unspecified)
 - Margin: (Use default, unspecified)
- Change the class of the MainTable containing the product sheet so that the dividing lines are not shown. I set it the Table class.

Lastly, I create a new class of Attribute to highlight its name and price a bit more. So, I create the AttributeView class and set its properties: *FontSize = 22 dip and Font Weight = bold*.



I apply this class to the attributes ProductName and ProductPrice.



In sum, we've seen that from what we have already defined for the web system (structure and behavior) we can generate native applications for mobile devices reusing the knowledge of the business encapsulated in the Knowledge Base. This is the benefit described as **MULTI-EXPERIENCE: CREATE ONCE FOR MULTIPLES PLATFORMS**

We've also seen that GeneXus allows us to be independent from any given technology, because the generators are in charge of writing the application based on the current state of the art. This is the benefit described as **FUTURE PROOF: TECHNOLOGY INDEPENDENT**

Also, remember that GeneXus created all the mobile and web screens required for the application to interact with the back-end, saving us a lot of time. That's why we say that GeneXus helps us to be highly productive by ensuring the necessary completeness to create an application for mobile devices. **HIGHLY PRODUCTIVE: COMPLETENESS**

The Work With for Smart Devices pattern shows the power of applying patterns, managing the creation and maintenance of all the mobile screens from the definition of the instance and the definition of the general settings of the pattern.

This was achieved by applying artificial intelligence and automation.

COST SAVING: AI-BASED AUTOMATION

Lastly, to close the demo I'd like to mention that creating mobile systems and applications is becoming increasingly complex, as there are more and more technologies that have to be combined, more systems we need to integrate with, and so on. In this context, GeneXus appears simple on the outside, being complex on the inside. Many things are solved internally and automatically, so it frees us (GeneXus Analysts) from having to do them. That's why we say it is:

EASY TO: USE (simple on the outside, complex on the inside)

If you're interested in training and starting to use GeneXus, the self-study course consists of videos with a total duration of 7.5 hours. GeneXus is very easy to learn.

EASY TO: LEARN

11. SUMMARY

So far, we've seen how GeneXus' features provide tangible benefits for users.

Other features and products included in the GeneXus suite:

GXserver is a code management tool for collaborative work.

DevOps allows you to deploy automatically in the cloud by pressing a key (F6).

GXtest allows you to automate unit tests of procedures and user interface.

GXflow allows creating applications based on the definition of process workflows.

12. GX SERVER

GeneXus Server is a product that automates the integration of knowledge. It makes distributed development possible, as if you were working in a centralized setting, by consolidating the project in an automatic and efficient manner.

For more information, visit: <http://gxserver.com/en/main>

INCREMENTAL APPROACH

EASIER TO: MAINTAIN

13. GX TEST – UNIT TESTS

GXtest is integrated into the GeneXus IDE and allows us to define and save unit and user interface tests, so that we can run them again as necessary; for example, after having made changes in the line of development.

For more information, visit: <https://www.genexus.com/en/global/products/gxtest>

EASIER TO: CHANGE

COST SAVING: AI-BASED AUTOMATION

14. GX FLOW – BUSINESS PROCESS MODEL

GXflow is a workflow tool integrated into GeneXus for modeling, managing and optimizing the business processes of a company to create applications in a simple and effective manner.

For more information, visit:

<https://genexus.es/gxflow/>

COST SAVING: AI-BASED AUTOMATION

15. DEPLOY TO CLOUD – F6

The Deploy to Cloud feature allows us to deploy an application in the cloud by pressing a key.

For more information, visit:

<https://wiki.genexus.com/commwiki/servlet/wiki?29606,GeneXus%20Cloud%20Sandbox>

EASIER TO: DEPLOY

HIGHLY PRODUCTIVE: DEVOPS

16. CONCLUSION

So, have we achieved the challenge? Could you have done all this with another technology in less than an hour?

HIGHLY PRODUCTIVE: UNSURPASSABLE SPEED

17. ANNEX

KB available at <http://samples.genexusserver.com/v16/kbdashboard.aspx?DemoGX16>

(You must be signed in with your gxaccount user in order to download it)

Thank you, we really hope to hear from you soon!!

The GeneXus Team