# Web Screens with Customer-facing Focus

## Getting Started with Angular

GeneXus™

So far, we have seen how to build screens focused on the back-office of the application; that is, the part that the travel agency employees use to enter and maintain the company's data. Now we will see how to design and implement the screens that the agency's clients will use to browse the information, which constitute the customer-facing part of the application.

Agency's request: Customer-facing application

Panel objects
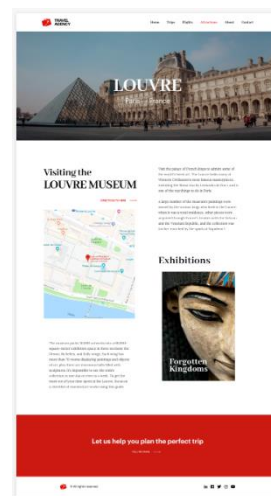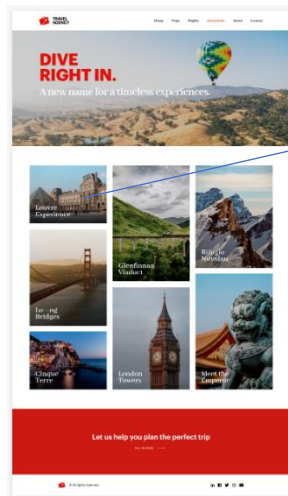
The travel agency has requested a web screen for their clients that shows the available tourist attractions and allows them to interact with the information; for example, filtering the data on screen to refine a search or to see the details of a selected attraction.

Previously we saw how to build screens with web panels, now we will see how to implement them with the panel object, which will allow us to generate the application in Angular.

Remember that we could use these same panel objects to generate the screens of our mobile application, for Android or Apple devices.

Developing the application

| GENEXUS |
|---|

**Start Page** × | **Attractions_CFPanel** ×

**Layout** | Rules | Events | Conditions | Variables

Application Bar

GRID

| AttractionId | AttractionName | CountryName | |
|---|---|---|---|

Propriedade Auto Grow do Grid: True

.Net Environment
- Back end
  - Default (C# Web)
  - Data Stores
  - Services
- Front end
  - Web (C#)
  - Web (Angular)
- Deployment

**Generator: Frontend (Front end)**

| Name | Frontend |
|---|---|
| Generate Android | **False** |
| Generate Apple | **False** |
| Main Platform | Angular |
| Dynamic Services URL | False |
| Services URL | https://trialapps3.gene. |
| SSL Pinning Pin Set | |
| Smart Devices Cache Management | On |
| Generate Angular | **True** |
| **Angular Specific** | |
| Setup Command | **npm install** |
| Run command | npm start |
| Build Mode | Prototype |
| Default Platform Hint | Best fit |

Prerequisites for Angular:
https://wiki.genexus.com/commwiki/servlet/wiki?42541

We will create the list of available attractions, so that we can then click on one that interests us and see more information about that attraction. The idea is to build a panel similar to the WWAttractionsFromScratch web panel we saw earlier.

We create a FrontendAngular folder and create a Panel type object named Attractions_CFPanel. As we can see, here we also have a form where we can drag controls from the toolbar.
We start by dragging a Grid and selecting the AttractionId, AttractionName, CountryName and AttractionPhoto attributes. Then we set the Visible property of AttractionId to False, and in the Base Trn property of the grid we assign the Attraction transaction.
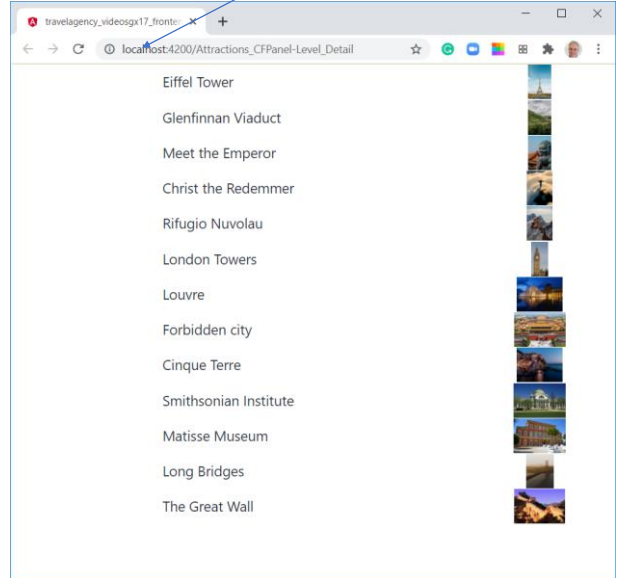
Now we will assign Angular as a generator. In the KB Explorer we click on Frontend and note that there is a property called Generate Angular whose default value is False, so we set it to True. We also see that Generate Android and Generate Apple are by default set to True, but since we are not interested in generating for mobile devices at the moment, we set them to False.

To be able to generate in Angular, we must install the software mentioned in the wiki article, which is shown on screen.

Executing the application

O valor do porto pode mudar e será atribuído automaticamente

To run it, first we set the panel we've just created as Main, right-click on it and select Run. We see that a command line window opens showing the execution of the server and after a few minutes the browser opens with the application running.

Obviously the design is quite poor, but we'll take care of that. The application is running by default on the URL http://localhost:4200, which is the address of the local development server where the application was automatically installed.
We already have our first angular application running.

## Navigation Lists
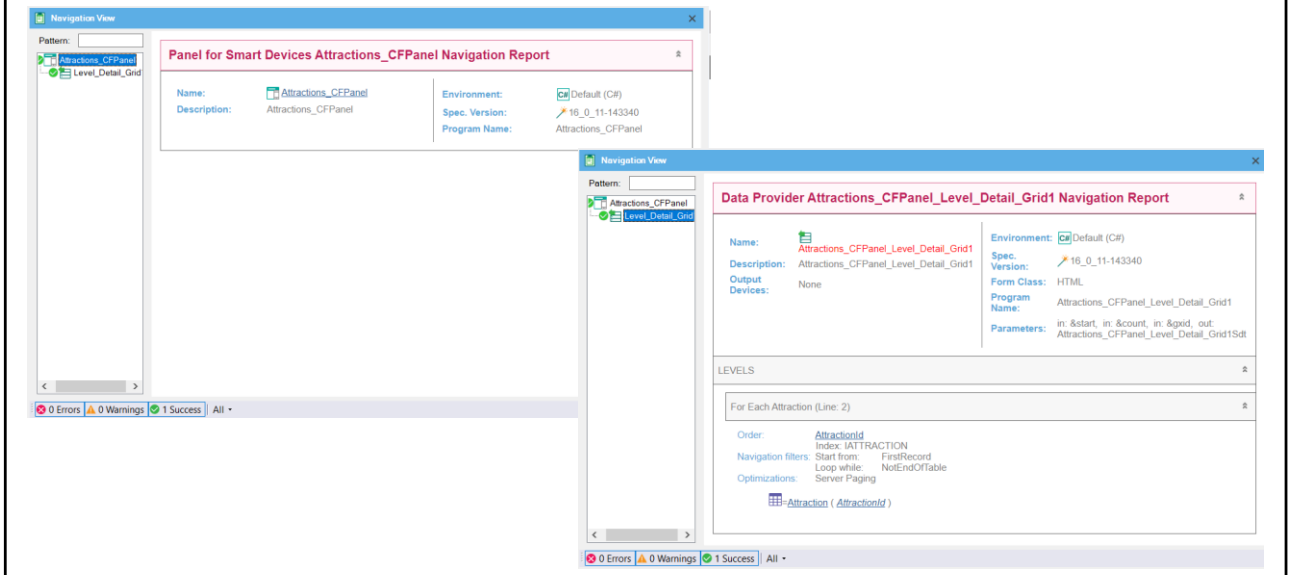


If we go to the navigation list of the Attractions_CFPanel panel object we see that two entries are displayed. If we select the one with the panel name, the list is empty and no reference to the Attraction table is shown. This is in the node called Level_Detail_Grid1, where we see that the information is similar to what we would see with a web panel with Attraction base table, since the panel's grid is running through the Attraction table to show the tourist attractions.

The node in the list named Attractions_CFPanel corresponds to the panel itself, and includes information on User Interface elements, such as the navigation of the Dynamic combos. Since our panel doesn't have any elements in the fixed part, the list appears empty.

The node called Level_Detail_Grid1 corresponds to the grid we inserted. The Attraction base table is run as we expected, after having inserted Attraction attributes in the grid. The title of the report says that the navigation corresponds to the Data Provider Attractions_CFPanel_Level_Detail_Grid1, which is the data provider published as a service in the back-end and invoked by the panel to access the database and retrieve the attractions.

Here is the evidence of what we saw before: that the architecture of these

customer-facing applications with Angular has logic at the client level, and that services are invoked on the server to access the information in the database.

We add the total trips of each attraction and overall total of trips



To make our panel more similar to the Working with Attractions web panel seen before, we will add one more column to the grid with the total trips of each attraction and outside the grid, the total trips of all the attractions. We create the variables &Trips and &TotalTrips, add &Trips to the grid by setting its Label position property to None, and place &TotalTrips below the grid.

To get the number of trips for an attraction, you have to run through the TripAttraction table. As this is related to the database, we have to schedule an event to be triggered on the server. In the panel objects we also have the Start, Refresh and Load events like in the web panels.

We will schedule the events just like we did with the web panel WWAttractionsFromScratch. In the Load event we use a Count formula that uses the TripDate attribute to count the trip records. As we saw before, although the TripDate attribute belongs to the Trip table, GeneXus will not choose the Trip table as the base table of the formula, but the TripAttraction table instead.

As we know that the grid runs through the Attraction table because we saw it in the navigation list, the Load event will be triggered once for each row of the grid, so the count formula will count the trips of each attraction shown. Then we accumulate in &TotalTrips the total of all trips.

The same clarification we saw before can be made: in this case we use the general Load event because we have only one grid, but we could have used the Grid1.Load event, which allows us to add another grid to the panel in the future and not change the programming.

Since we want the total trips to be initialized at zero before starting to count the trips of the attractions, in the Refresh event we will add the initialization to the &TotalTrips variable, in a similar way to what we had done in the web panel.

To run it again, we right-click on the panel and select Run.

Looking at the navigation list, we see that there is a new level of detail different from the grid, called Level_Detail. This is where the load of the fixed part of the panel is shown, i.e. all the controls of the form that are not included in a grid.

Unlike web panels, in a panel object the fixed part is independent of the grid and, as we will see later, the fixed part can even have a different base table than the grid base table.

The navigation list of the Level_Detail node (which invokes the data provider mentioned in the title) appears empty, because no field is being loaded from the database, there is only the &TotalTrips variable, which is updated within the Load event.

If we go to the navigation list of the Level_Detail_Grid1, we see that the corresponding data provider is accessing the Attraction table, ordered by AtractionId that is the default order, and there is also the navigation of the Count formula since it is triggered in the Load event.

The overall total of trips is miscalculated!

```
Layout *  Rules  Events *  Conditions  Variables

Events

1 ⊟ Event Load
2        &Trips = Count(TripDate)
3        &TotalTrips = &TotalTrips + &Trips
4   └ Endevent
5
6 ⊟ Event Refresh
7        &TotalTrips = 0
8   └ Endevent
9
```

If we look at the application running in the browser, we see that the total number of trips of each attraction is displayed correctly, but the overall total of trips comes out at 0.
What did we do wrong? The &TotalTrips variable is being increased in the Load event as we did in the web panel and we are sure that this event is being triggered because we see that some attractions have trips... So?

The reason is that panel objects do not work in the same way as web panels. In panel objects, the fixed part is loaded independently of the grid. In this case, the &TotalTrips variable is in the fixed part of the panel, which is the first thing to be loaded. Then the grid is loaded, so when the variable is displayed, the grid has not been loaded, the Load event has not been triggered, and the value of &TotalTrips has not been added yet.

Remember that in a panel object used to develop customer-facing applications, to load the screen in the client device, services located in the server are invoked which are the ones that access the database. These services are data providers, which are independent for the fixed part and for the grid (or each grid) of the screen.

When the panel starts running, a local event is triggered on the client that invokes the data provider to load the fixed part and causes the Start and

Refresh events to be triggered on the server. Then a second data provider is executed which triggers the Load event on the server N times and the grid is loaded.

In our example, as the Refresh is triggered first, the &TotalTrips variable is initialized and the fixed part is loaded; later, the Load event is triggered which is where &TotalTrips is loaded with the correct value and the grid is updated, but the fixed part was already loaded before and is not redrawn.

Due to this feature we cannot program the panel object as if it were a web panel.

In another video we will have a closer look at the triggering of events and the determination of base tables; for now, to make the example work we will change the programming.

## Correct Solution

```
1  Event Load
2      &Trips = Count(TripDate)
3  Endevent
4
5  Event Refresh
6      &TotalTrips = 0
7      For each Trip.Attraction
8          &TotalTrips += 1
9      Endfor
10 Endevent
```

Pattern: _____

- Attractions_CFPanel
  - Level_Detail_Grid1
  - Level_Detail

**Data Provider Attractions_CFPanel_Level_Detail Navigation Report**

| | | |
|---|---|---|
| **Name:** | Attractions_CFPanel_Level_Detail | **Environment:** C# Default (C#) |
| **Description:** | Attractions_CFPanel_Level_Detail | **Spec. Version:** 16_0_11-143493 |
| **Output Devices:** | None | **Form Class:** HTML |
| | | **Program Name:** Attractions_CFPanel_Level_Detail |
| | | **Parameters:** in: &gxid, out: Attractions_CFPanel_Level_Detail: |

**LEVELS**

For Each Trip (Line: 11)

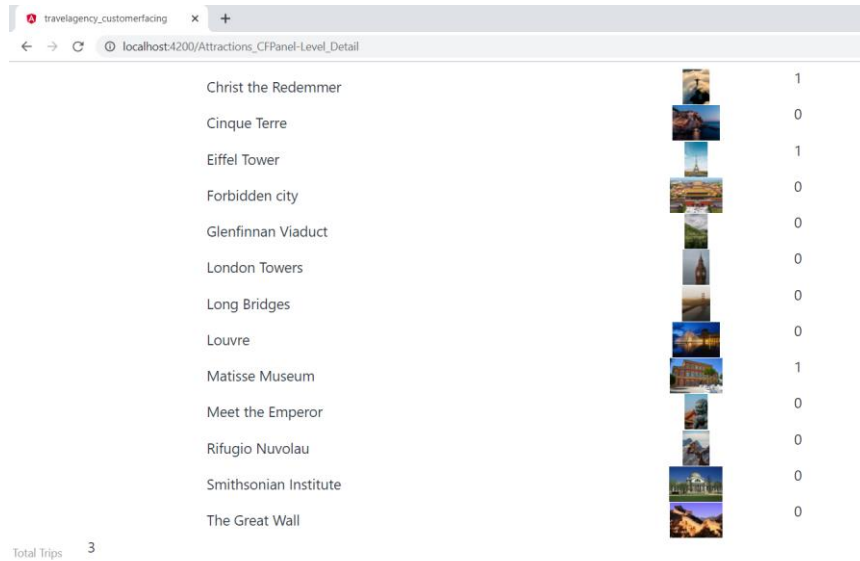| | |
|---|---|
| Order: | TripId |
| | Index: ITRIP |
| Navigation filters: | Start from: FirstRecord |
| | Loop while: NotEndOfTable |
| Optimizations: | count( *) |

⊞=Trip ( TripId )

---

The solution is to include in the Refresh event a For Each command that accesses the TripAttraction table and counts the overall total of trips. Let's not forget to remove the calculation we had in the Load event.

The reason why we include the update in the Refresh event, using a For Each command, is because the Refresh event will be triggered when the fixed part is loaded, which will be before the grid is loaded.
Therefore when loading the fixed part, the value of &TotalTrips will have the correct value and only after that the grid part will be updated.

We run it.

Note that now the navigation list corresponding to the Level_Detail node includes the For Each command with the navigation to the TripAttraction table.

## Running it with the right overall total of trips



In the browser we see that the overall total of trips is now displayed correctly.

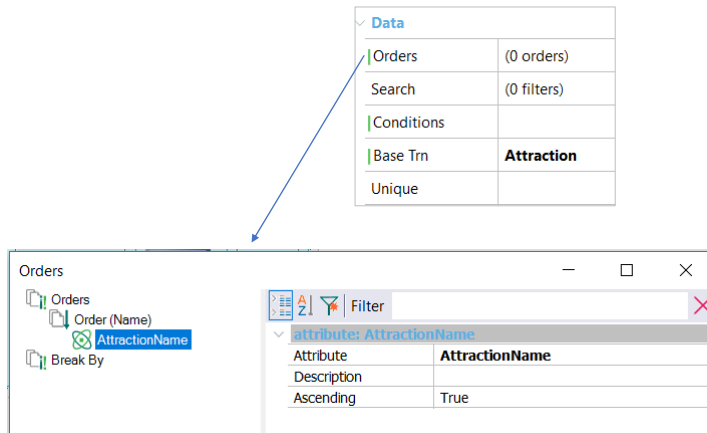We add filters by country and name of attraction

**Editing Conditions**

```
CountryId = &CountryId when not &CountryId.IsEmpty();
AttractionName >= &AttractionNameFrom when not &AttractionNameFrom.IsEmpty();
AttractionName <= &AttractionNameTo when not &AttractionNameTo.IsEmpty();
```

To complete the Work With Attractions panel, we would need to add the filters by the country identifier and by attraction names.

We add above the grid a &CountryId variable as Dynamic Combo box, with the Item Descriptions property showing the CountryName, and set the Empty Item property to True. We also add the &AttactionNameFrom and &AttractionNameTo variables to filter the attractions by name.

In a web panel we would add the filter in the grid conditions, and here we can also do the same. Let's define a condition to filter by country and others to filter from and to the name of the attraction.

We order the list of attractions by name

| Data | |
|---|---|
| Orders | (0 orders) |
| Search | (0 filters) |
| Conditions | |
| Base Trn | **Attraction** |
| Unique | |

Orders

Orders
  Order (Name)
    AttractionName
Break By

| Filter | |
|---|---|
| attribute: AttractionName | |
| Attribute | **AttractionName** |
| Description | |
| Ascending | True |

We also have an Order property, where we are going to set the grid to be ordered by attraction name, so, we right-click on Orders, give it a name – for example, Name– and then right-click again to open the AttractionName attribute.

We can define an order by several attributes, as well as create other orders by different criteria. The part where it reads Break By allows us to group the grid records; for example, if we want the attractions to be grouped by country name.

If we right-click on the panel and select View Navigation, in the navigation list of the panel itself, we see that the Country table is accessed to fill the Dynamic Combo box (where it says FILL &CountryId WITH CountryId, CountryName IN).
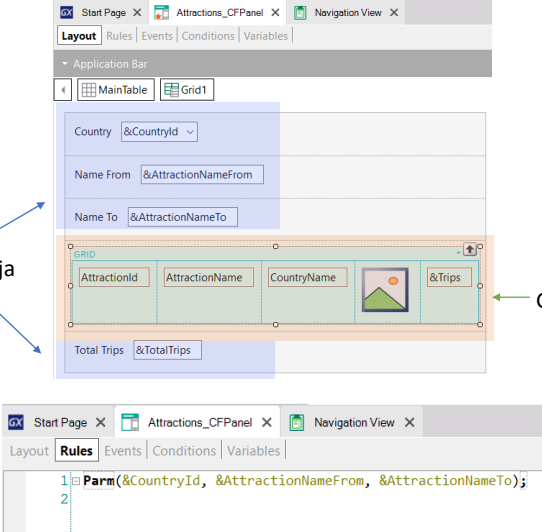
The navigation list of the Level_Detail node shows the Data Provider report (automatically created by GeneXus but not displayed in the KB) that is invoked in the server to retrieve the data from the database, which is necessary to load the fixed part of the panel. This Data Provider triggers the Start and Refresh events on the server.
In the list we see the For Each command that we programmed in the Refresh event, and that accesses the TripAttraction table to count the total of trips. Also, we see the access to the Country table filtered by the CountryId selected in the Dynamic combo.

If we look at the navigation list corresponding to the Data Provider that loads the grid, we see that now the attractions will be run through in order by AttractionName and that the constraints show the filters that we defined. This Data Provider executes the Load event internally once for each line of the grid to be loaded as in web panels, if the grid has a base table and returns the information to the panel to be displayed.

Need to refresh the content with the filters

```
1  □ Event Load
2        &Trips = Count(TripDate)
3  └ Endevent
4
5  □ Event Refresh
6        &TotalTrips = 0
7        For each Trip.Attraction
8            &TotalTrips += 1
9        Endfor
10 └ Endevent
11
12 □ Event &CountryId.ControlValueChanged
13        Grid1.Refresh()
14 └ Endevent
15
16 □ Event &AttractionNameFrom.ControlValueChanged
17        Grid1.Refresh()
18 └ Endevent
19
20
21 □ Event &AttractionNameTo.ControlValueChanged
22        Grid1.Refresh()
23 └ Endevent
```

```
1 □ Parm(&CountryId, &AttractionNameFrom, &AttractionNameTo);
2
```

Something we mentioned was that the fixed part of the panel is loaded independently from the loading of the grid. Different data providers are invoked that are published in the server as services and access the database to retrieve the information of each part.

When we change the value of a filter, the grid information needs to be refreshed. For this, we need to add the grid's Refresh method, which will trigger the server's Refresh and Load events. In turn, it will cause the grid to be reloaded, applying the programmed conditions and displaying the filtered results as expected.

Since the grid's Refresh method must be invoked after we change the value of the filter variable, we use the ControlValueChanged event of each variable to invoke the method. In this way, after changing a value, when leaving the field the corresponding event will be triggered which will end up refreshing the content of the grid.

However, there is something else we need to take into account which is that in this architecture, since the objective is to have the page loaded as few times as possible, the data cache is prioritized; that is to say, we always try to retrieve the information previously saved. For the server to understand that we want to bring new data, the URL sent to the server

must be changed, so that it understands that it is a new page and obtains the corresponding information to send to the client.
To do so, we added a Parm rule containing the values of the variables used in the filters. In this way, if a variable's value changes, the page is updated with the new data.

Now we run it to test what we have seen.

# Execution with the new filters and order

| | | | |
|---|---|---|---|
| Country | (None) | ⇕ | |
| Name From | Name From | | |
| Name To | Name To | | |

| | | |
|---|---|---|
| Christ the Redemmer | | 1 |
| Cinque Terre | | 0 |
| Eiffel Tower | | 1 |
| Forbidden city | | 0 |
| Glenfinnan Viaduct | | 0 |
| London Towers | | 0 |
| Long Bridges | | 0 |
| Louvre | | 0 |
| Matisse Museum | | 1 |
| Meet the Emperor | | 0 |
| Rifugio Nuvolau | | 0 |
| Smithsonian Institute | | 0 |
| The Great Wall | | 0 |

Total Trips    3

Note that at the top we now have the filters we added and the attractions are sorted by name as we expected.

## Execution with the new filters and order (continued)

| | | | | |
|---|---|---|---|---|
| Country | China ⇕ | | | |
| Name From | Name From | | | |
| Name To | Name To | | | |
| | Forbidden city | | 0 | Details |
| | Meet the Emperor | | 0 | Details |
| | The Great Wall | | 0 | Details |

| | | | | |
|---|---|---|---|---|
| Country | China ⇕ | | | |
| Name From | A | | | |
| Name To | N | | | |
| | Forbidden city | | 0 | Details |
| | Meet the Emperor | | 0 | Details |

We will filter by the country China and see that China's attractions will be displayed.

Now we choose to see the attractions that begin with the letter A to the letter N, and we only see the attractions in China whose name is in that range.

We implement an attraction's detail

GeneXus

Layout * | **Rules *** | Events | Conditions | Variab

```
1 ⊟ Parm(in:AttractionId);
```

Start Page ✕ | Attractions_CFPanel ✕ | Navigation View ✕ | AttractionDetail_CFPanel ✕
Layout | Rules | Events | Conditions | Variables
▼ Application Bar
◄ | ▦ MainTable

BACK

AttractionName

CityName

CountryName

AttractionDescription

GRID

AttractionInfoName

AttractionInfoDescription

Name
⊟ ▦ Attraction
  📍 AttractionId
  🔑 AttractionName
  ↗ CountryId
  ↗ CountryName
  ↗ CategoryId
  ↗ CategoryName
  🖼 AttractionPhoto
  ↗ CityId
  🔑 CityName
  ● AttractionAddress
  △ AttractionDescription
⊟ ≣ Info
  📍 AttractionInfoId
  🔑 AttractionInfoName
  🖼 AttractionInfoImage
  ● AttractionInfoDescription
  ● AttractionInfoPriority

Atributos agregados para cumplir con nuevos requisitos

Country | &CountryId ▾

Name From | &AttractionNameFrom

Name To | &AttractionNameTo

GRID
AttractionId | AttractionName | CountryName | 🖼 | &Trips | &Details

Total Trips | &TotalTrips

```
14  └Endevent
15
16 ⊟ Event &AttractionNameFrom.ControlValueChanged
17      Grid1.Refresh()
18  └Endevent
19
20
21 ⊟ Event &AttractionNameTo.ControlValueChanged
22      Grid1.Refresh()
23  └Endevent
24
25 ⊟ Event Start
26      &Details = "Details"
27  └Endevent
28
29 ⊟ Event &Details.Tap
30      AttractionDetail_CFPanel.Call(AttractionId)
31  └Endevent
```

Now let's complete the agency's request, which is to show the details of each listed attraction by clicking on them. For this we will use another panel object named AttractionDetail_CFPanel. To save time, I have already created it.

We see that in the rules we define a Parm rule, with an input parameter, the AttractionId attribute.
Remember that this will allow showing only the information of the attraction passed by parameter, which was the one we selected in the panel of the attractions list.

In the form we added the attribute AtractionPhoto and the attributes AtractionName, CityName, and CountryName. Below we inserted the AttractionDescription attribute. Next, we inserted a grid and selected the attributes AttractionsInfoName, AttractionInfoImage and AttractionInfoDescription to see the attraction details. As we can see, we have inserted some Table controls for better alignment.
We also added a BACK button at the top right corner that will invoke a Return that will allow us to return to the list of attractions.

Now we go to the panel Attractions_CFPanel and add a &Detail variable of Character type to the grid. In the Start event we assign the text "Details."

Next, we right-click on the grid variable and select Go to Event, Tap and write the invocation to AttractionDetail_CFPanel, passing AttractionId as a parameter.

Let's try this at runtime,

# Execution with an attraction's details

| Country | (None) ⬍ | | | |
|---|---|---|---|---|
| Name From | Name From | | | |
| Name To | Name To | | | |
| | Christ the Redemmer | | 1 | Details |
| | Cinque Terre | | 0 | Details |
| | Eiffel Tower | | 1 | Details |
| | Forbidden city | | 0 | Details |
| | Glenfinnan Viaduct | | 0 | Details |
| | London Towers | | 0 | Details |
| | Long Bridges | | 0 | Details |
| | Louvre | | 0 | Details |
| | Matisse Museum | | 1 | Details |
| | Meet the Emperor | | 0 | Details |
| | Rifugio Nuvolau | | 0 | Details |
| | Smithsonian Institute | | 0 | Details |
| | The Great Wall | | 0 | Details |
| Total Trips | 3 | | | |

We will see the information of the Louvre museum, so in the corresponding line we click on Details.

Execution with an attraction's details (continued)

BACK

Louvre

Paris

France

Visit the palace of French kings to admire some of the world's finest art. The
Louvre holds many of Western Civilization's most famous masterpieces.

Visiting the Louvre Museum

Visit the palace of French kings to admire some of the world's finest art. The Louvre holds many of
Western Civilization's most famous masterpieces, including the Mona Lisa by Leonardo da Vinci,
and is one of the top things to do in Paris. A large number of the museum's paintings were owned

The panel opens with the attraction's details, where we can see a map and
the available exhibitions.

# Generating the app in native Android

Prerequisites for Android: https://wiki.genexus.com/commwiki/servlet/wiki?14449
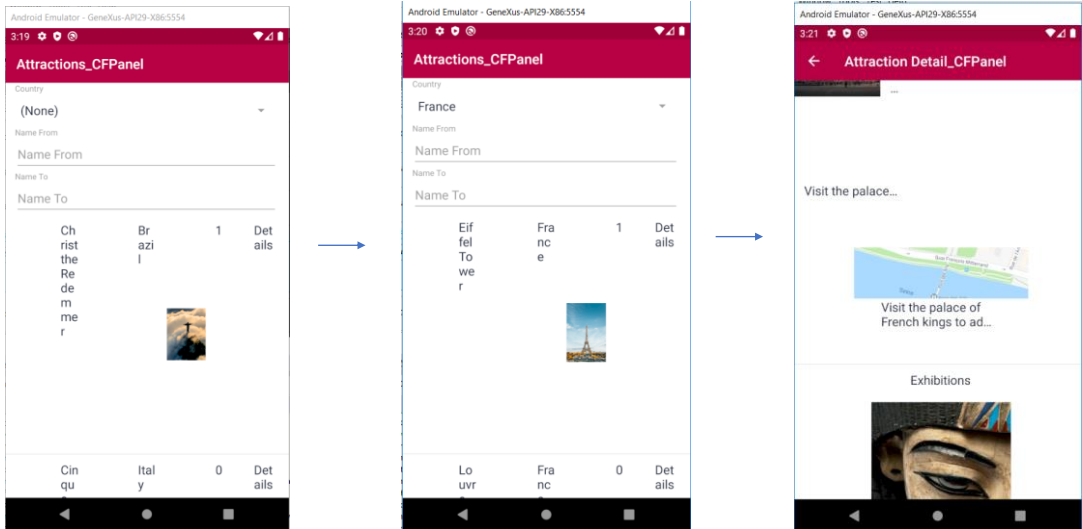


At the beginning of this video we said that if we wanted to, we could use the panel objects we built to generate the screens for our mobile application. Let's try this out.

To be able to generate in Android language, you must install the software mentioned in the wiki article, which is displayed on the screen.

In the KB Explorer, we click on the Front end node and set the Generate Android property to True and the Main Platform property to Android. Now we execute the main object Attractions_CFPanel.

We can see that the Android emulator opens showing the Attractions_CFPanel panel, with the list of tourist attractions. Obviously, we should define a layout according to the screen size of a phone, since when we designed the panel we were thinking about the screen of a web system that would run on the screen of a desktop computer.

But beyond the design, let's check if it works properly. In the filter we chose France and it shows only the attractions of France. Now we click on the Details of the Louvre and see that the screen opens with the details of the museum, showing the map and the exhibits as we expected.

In this video, we started to get familiar with the panel objects, generating the application in the Angular framework. We confirmed that with the same objects created we could also generate the application in native Android language.

Next, we will learn more about the event management at client and server level.

training.genexus.com
wiki.genexus.com