Chapter 12: Forms Development

This chapter focuses on the advanced use of BIZUIT Forms Designer to create dynamic, interactive, and customized forms, tailored to various business needs. We'll explore the designer environment, configuring and customizing controls, integrating external and internal data sources, and using subforms for modularity and flexibility. We will learn how to configure advanced properties, validate inputs, handle dynamic data, and design attractive and functional interfaces.

Ideal Audience

This chapter is aimed at BPM project implementers who already have experience with developing forms or user interfaces in enterprise environments. Basic knowledge in: Form Design, UX/UI, HTML, and JavaScript is recommended

Objectives

1. Explore the BIZUIT Forms Designer environment: Understand the key functionalities and tools to design efficient and customized forms.
2. Set up basic and advanced controls: Customize properties, create validations, and handle dynamic data to optimize interaction with forms.
3. Design attractive interfaces: Use controls such as tables, tabs, and cards to build organized and functional forms.
4. Integrate external data: Set up secondary data sources (SQL, BIZUIT, and RESTAPI) to enrich forms with real-time information.
5. Implement subforms: Divide complex flows into modular sections to improve flexibility and user experience.

Unit 1: Exploring the BIZUIT Forms Designer Environment

In this unit we are going to explore the BIZUIT form designer environment in depth. Our goal is to know each of its areas in detail, understand its functionality and discover how to make the most of the tools it offers us to build powerful, personalized forms aligned with the processes of our organization.

Menu sidebar

We begin our journey with the menu sidebar, a fundamental area from which we access all the necessary options to create, edit and manage forms, both independent and associated with processes.

File

Within the File menu we find the essential tools to manage our forms. Some of these options vary depending on the type of form:

- **New Form** (for standalone forms only): Allows us to start a design from scratch. The system will ask us for a unique name and, if we want, we can also define category, subcategory and description.
- **Open Existing Form**: A list of saved forms is accessed. We can search by name, category, or subcategory. From this window we can also delete forms, open previous versions or export them.
- **Export/Import Form**: when exporting, we generate a downloadable file that we can import into another environment. If there is already a form with the same name, the system will ask us for a new one to avoid conflicts.
- **Save Form**: we can overwrite the current version or save a new one. For process-associated forms, the Save & Publish option speeds up design implementation.
- **Copy Form**: Ideal for reusing existing forms as a basis for new designs.
- **Form Settings**: We quickly access the property sheet to adjust all the general parameters of the form.

Controls

From this menu we can access all the available components: text boxes, buttons, drop-down lists, text areas, etc. We can drag them into the design area and place them according to our needs. If we have many, we have a search bar to find them easily.

Added Controls

Here are all the controls that we have already incorporated into the form. We can select them to modify their properties or delete them directly.

Subforms

From this section we manage the subforms linked to the main form. We can create, edit, or delete them as the design requires.

Data Sources

Here we configure the sources that will feed the form. For forms associated with processes, in addition to REST or SQL APIs, we can also work with parameters, variables or results of previous process activities.

Design Area

The design area is where we visually organize our form. We place the controls, distribute them in grids of rows and columns and build the logical and visual structure of the interface. This space allows us to work in a clean, structured and agile way.

Toolbar

The top bar concentrates practical tools for editing the form:

- **Code**: We can view and edit the custom code associated with events in the controls. We also have a Global Code space, where we define reusable functions in different forms.
- **Copy, Paste, Cut, Undo, Redo** – the classic editing functions.
- **Toggle Grid**: Allows you to show or hide the grid of rows and columns to make it easier to organize the layout.
- **Copy URL** (for standalone forms): we get the link of the form, useful for integrating it into other systems or sharing it directly.

- **Run Form**: We visualize what the form will look like at runtime, ideal for quick testing.
- **User icon**: we access general settings of the publisher environment. If we have permissions, we can also modify the appearance of the environment, just as we do with the BIZUIT Dashboard.
- **Invite to View/Edit**: we select users whom we want to invite to collaborate. By checking the "Can Edit" option, the guest user will be able to work together with us, in real time. You'll receive an email invitation, and when you accept it, you can start editing right away.

Property Sheet

The property sheet is where we set up the details for each control and the overall form. We adjust names, styles, events, data connections, and any necessary parameters so that each component correctly fulfills its function.

Conclusion

In this unit we went through the BIZUIT Forms Designer work environment in detail, understanding each of its key areas and tools for form design. We learned how to use the side menu to access essential functions, from creating, opening, and exporting forms to managing subforms and data sources.

We explored the design area as a central space to visually structure forms, along with the toolbar that allows us to take quick actions, test designs, and collaborate in real-time with other users.

We also delve into the property sheet, where it is possible to adjust the general parameters of the form and the individual configuration of each control, thus optimizing its appearance, behavior and integration with data and processes.

With this knowledge, we are prepared to effectively leverage the designer's environment, creating powerful, customized forms aligned with the operational needs of our processes at BIZUIT.

Unit 2: Controls

In this unit we will explore the various controls available in BIZUIT and learn how to configure them to maximize their functionality and customization.

We'll look at how forms and controls are key tools for capturing, displaying, and managing data in modern applications. You'll learn how to set up basic and advanced properties, create validations, work with dynamic data, and handle complex interactions. In addition, we will learn to integrate multimedia, geolocation and digital signatures, adapting each control to the specific needs of our projects.

From forms to tables, tabs, and visuals like cards or headers, this unit will give us the foundation to design engaging, interactive, and highly functional interfaces.

1.Form

We'll look at the basic properties of forms in BIZUIT, focusing on how to customize their appearance, behavior, and associated actions, both in standalone forms and those linked to processes.
Our goal is to make the most of the options available to us to tailor each form to our functional and visual needs.

Accessing Form Properties

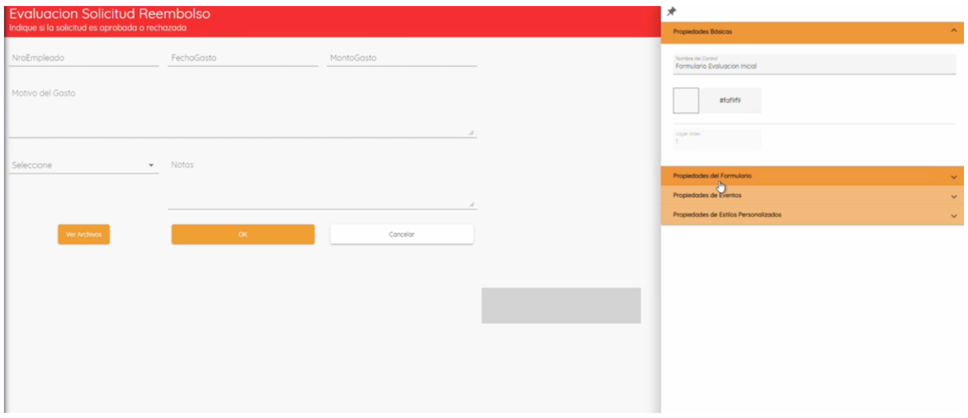
To get started, from the sidebar of the editor we select the option "Edit Form Properties". Doing so displays the configuration options, which vary slightly depending on whether it is a standalone form or one associated with processes.

Options Common to All Forms

In both types of forms we find the Basic Properties tab, from where we can modify the name of the form and its background color.
In the Form Properties tab, we access visual design parameters such as:

- **Controls style:** we apply predefined styles, such as "Standard".

- **Number of columns**: we define the visual structure of the form. In this example, we use a 6-column layout.
- **Row height**: We adjust the height in pixels, for example, to 70 px.
- **Visual theme**: we choose from predefined themes such as "Bizuit", "Red" or "Green", unifying the style of the form with the user's environment or brand.


Evaluacion Solicitud Reembolso
Indique si la solicitud es aprobada o rechazada
NroEmpleado
FechaGasto
MontoGasto
Motivo del Gasto
Seleccione
Notas
Ver Archivos
OK
Cancelar
Propiedades Básicas
Formulario Evaluacion Inicial
#f9f9f9
Propiedades del Formulario
Propiedades de Eventos
Propiedades de Estilos Personalizados

Specific Properties for Process-Associated Forms

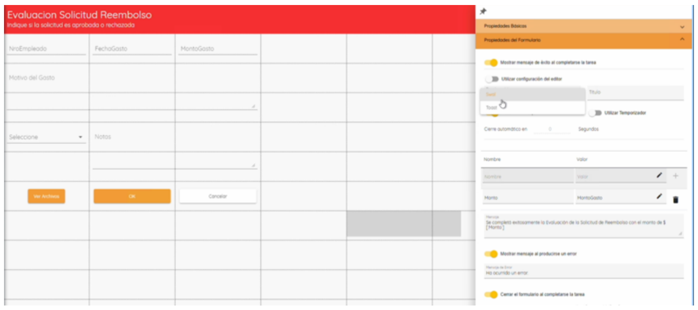
When we work with forms associated with processes, new options are enabled:

Success Message

We can define a custom message that is displayed when the task is successfully completed (for example, start or continue a process). We can even include process variables, such as the amount requested, in the message.

We also choose how this message will be presented:

- **Swal (modal window):** with title, close button and optional timer.
- **Toast (pop-up message):** Appears discreetly in the top corner of the screen.


Evaluacion Solicitud Reembolso
Indique si la solicitud es aprobada o rechazada
NroEmpleado
FechaGasto
MontoGasto
Motivo del Gasto
Seleccione
Notas
Ver Archivos
OK
Cancelar
Propiedades Básicas
Propiedades del Formulario
Mostrar mensaje de éxito al completarse la tarea
Utilizar configuración del editor
Toast
Título
Utilizar Temporizador
Segundos
Nombre
Valor
Monto
MontoGasto
Mostrar mensaje al producirse un error
Ha ocurrido un error
Cerrar el formulario al completarse la tarea

Error Message

We can set up a message to be displayed to the user in case of an error during the processing of the form.

Automatic Form Closure

We decide whether the form closes automatically once the task is completed. This option is useful in startup forms, where we want to allow the user to launch multiple instances without having to manually open the form each time.

Asynchronous Processing

If we enable this option, the form can be closed without waiting for BIZUIT to finish processing the task. Once complete, the result is displayed in the Notifications tab of the Dashboard.

We can also set a timeout so that the form stays open for a few seconds before closing automatically.

For example: if we know that the process can take 30 seconds, we set the automatic close to 5 seconds. This prevents the user from having to wait unnecessarily.

Exclusive Properties for Stand-Alone Forms

On standalone forms, we have an additional tab called Form Security.

From there we can activate the "Requires authentication" option and define which users or roles have access. This functionality allows us to restrict the use of the form according to the permissions set.

Summary

As we saw, the properties of the forms in BIZUIT give us a high level of customization, both visual and functional. We can adjust aesthetics, behavior when completing tasks, access security, and much more.

These options allow us to create custom forms, which are more intuitive, secure and aligned with the processes and users they are aimed at.

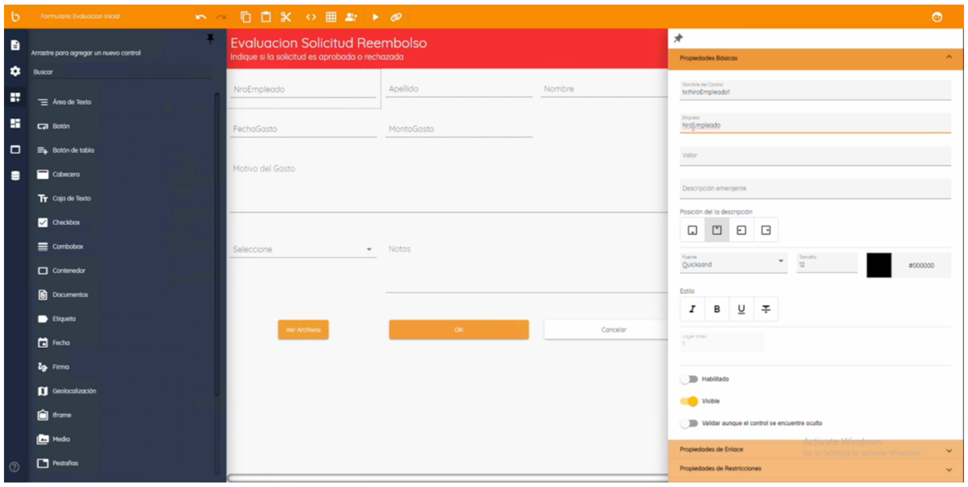
2. Configuring Controls:

Now we will explore in depth all the possibilities of configuring the controls in BIZUIT Forms Designer. From basic properties to dynamic rules and custom events, we'll see how each setting contributes to building more powerful, clear, and interactive forms.

Basic Properties

The basic properties vary depending on the type of control, but there are certain common parameters that we can adjust to customize the behavior and presentation of each element.

- **Control Name**: We assign a unique identifier, such as txtLast Name, which makes it easier for us to use in custom rules, links, or scripts.
- **Label**: Displays descriptive text about the controller, such as "Last Name," to help guide the user.
- **Pop-up Description**: Added a tooltip that appears when you hover your mouse over the controller, providing contextual help (e.g., "Enter your Last Name"). We can even customize your position.
- **Value**: We define an initial value that will be displayed in the control. If the data is linked to a data source, that value will be replaced when the feed is executed.
- **Style**: we adjust typography, size, color, bold, italics and underline. We can apply styles to improve the presentation and readability of the form.
- **Layer Index**: We set the order of overlap between controls. It's useful when multiple controls occupy the same space and we need one to be visible above another, depending on the context.
- **Enabled and Visible**: We decide whether the control will be active or visible during form execution.
- **Validate even when hidden**: Allows you to run control validations even when it's not visible, which is useful for more complex business rules.



Nombre del Control: txtNroEmpleado1
Etiqueta: NroEmpleado
Fuente: Quicksand
Tamaño: 12
#000000
Layer Index: 1

Link Properties

Links are critical for connecting controls to data and processes:

- **Primary Link** (in forms associated with processes): we associate the control with a parameter, variable or result of a previous activity of the process. It can be a bidirectional (read and write) or unidirectional (read-only) linkage.
- **Input Bindings (Data Sources):** We connect the control value with input parameters from secondary sources, such as SQL or APIs. For example, we link txtNroEmployee1 to the pIdEmployee parameter of a feed called GetDataEmployee, and configure its execution by changing the value of the control.
- **Output Links (Data Sources):** We use controls as the destination for values returned by a secondary source. Following the example above, txtLast Name and txtName will display the LastName and FirstName values obtained when querying the GetDataEmployee feed.

Constraint Properties

These properties allow us to control what type of data is entered:

- **Control Restrictions**: We can mark fields as required or limit the allowed values. For example, in a date control, we can restrict the selection to dates after the current one.

Validation Properties

Validations allow you to implement logical rules between fields to ensure data consistency:

- We can define conditions using operators such as "Equals" or "Different from".
- Rules can be compared against fixed values, parameters, process variables, or even against other controls in the form.
- We choose whether validation should be met when all conditions are true or when some are true.
- In addition, we can set up custom messages to help the user correct the data entered. For example: "The name cannot be the same as the last name."

Formatting Properties

Formatting allows us to dynamically alter the appearance of controls based on logical conditions.

- **Formatting Rules**: We define when a control should be visible, enabled, or have a certain color.
- **Formatting Actions**: We specify which changes to apply (e.g., hide a control, change the color of text, or disable it).

Practical example: we hide txtLast Name when the value of txtNoEmployee1 is equal to 2, and we show it in any other case.

Formatting rules can be tested by clicking Run Form, but validations are only executed when processing the form, so they must be tested from the BIZUIT Dashboard.

Custom Styles and CSS

In addition to the standard formatting, we can apply custom styles using CSS.

- From the control's style editor, we write CSS rules that allow us to modify colors, borders, fonts, margins, etc.
- This gives us a layer of advanced customization to visually adapt the form to specific brand or usability criteria.

Event Properties

Finally, events allow dynamic logic to be incorporated into the form, making it truly interactive.
The types of events vary depending on the control:

- **Change**: When you change a value.
- **Input**: with each user input.
- **Click**: when clicked.
- **MouseOver**: On hover.
- **Blur**: when you lose focus.

In each event, we can write code in TypeScript. For example, we get the value of txtNroEmployee1 and use it to construct a message like "Employee ID entered: [value]", which is displayed in another control (txtDemo).

Summary

This completes our tour of configuring controls in the BIZUIT Forms Designer.
We learned how to define basic properties, connect controls to data, apply validations, modify styles, set dynamic rules, and develop advanced behaviors with events.
These tools allow us to build powerful, intelligent forms adjusted to the real needs of our processes.

3. Text Box Control

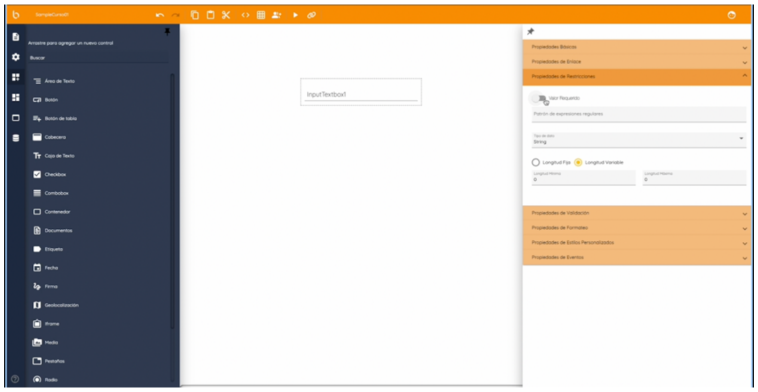
Next, we'll focus on the Text Box control, one of the most versatile and fundamental elements within any form in BIZUIT. This control allows us to capture data in a free or structured way, and thanks to its flexibility it can be adapted to multiple scenarios.
In particular, we're going to focus on constraint properties, which allow us to validate and control the information users enter, ensuring accuracy and consistency in the data right out of the box.

Constraint Properties

Constraints are configured directly from the control's property sheet and vary depending on the type of data selected.

Required Value

We can enable the "Required Value" option to force the user to fill in that field before they can proceed with the form. It's a simple but effective way to make sure key information doesn't go incomplete.



Valor Requerido

Patrón de expresiones regulares

Tipo de dato: String

Longitud Fija [✓ Longitud Variable]

Longitud Mínima: 0

Longitud Máxima: 0

Regular Expression Pattern

This field allows us to define a specific format that the value entered must meet. Using regular expressions (regex), we can validate inputs such as emails, specific codes, phone numbers, etc.

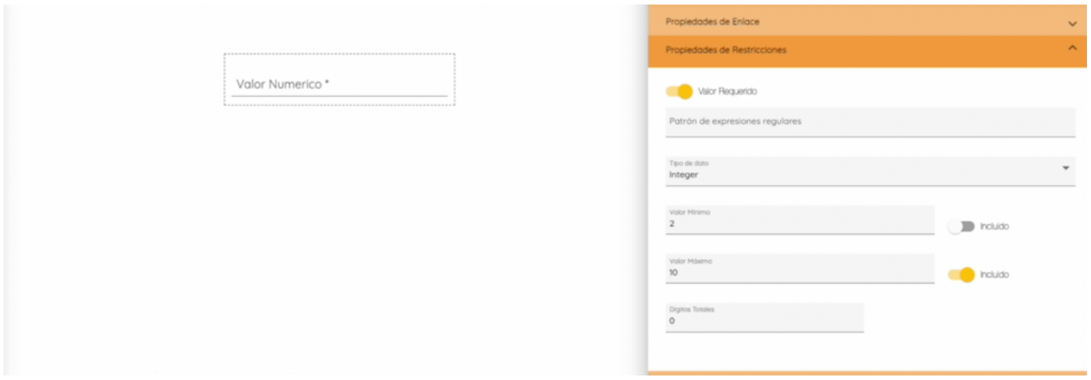
For example: if we require a valid email address, we set the corresponding regular expression. If the user enters a value that does not conform to the pattern, the control will be highlighted and display a contextual error message.

Selecting the Data Type

The type of data we select for the control will define what additional restrictions we can apply. Here are the main options:

a. Integer (Integer)

- **Minimum and Maximum Value**: We set a range allowed (e.g. between 2 and 10).
- **Inclusion of Limits**: We indicate whether or not the range includes extreme values.
- **Total Digits**: We limit the maximum number of digits. Useful for fields such as ID numbers or zip codes.


Valor Numerico *

Propiedades de Enlace

Propiedades de Restricciones

Valor Requerido: on

Patrón de expresiones regulares

Tipo de dato: Integer

Valor Mínimo: 2 — Incluido: off

Valor Máximo: 10 — Incluido: on

Dígitos Totales: 0

b. String (Text)

- **Fixed Length**: The text must have exactly a certain number of characters (e.g., 5 characters for a code).
- **Variable Length**: We set a minimum and maximum range (for example, between 2 and 10 characters for first or last names).

c. Double (Decimal)

Allows you to enter numbers with decimals. We can set minimum and maximum ranges, as well as the number of decimals allowed, depending on what the field requires.

d. DateTime

This type of data allows us to restrict dates within a specific range. We can also set relative dates, such as "do not allow dates older than today."

Summary

The Text Box Control is an essential tool for capturing form entries. Its ability to adapt to different types of data, combined with a robust and customizable validation system, makes it a key player in guaranteeing the integrity of information and improving the user experience.

By applying these constraints judiciously, we can ensure that data is entered correctly from the beginning of the process, avoiding downstream errors and increasing the efficiency of the overall flow.

4. Control Button

Below we will explore in detail the Button control in BIZUIT. This element is essential in any interface, as it allows us to execute specific actions and adapt them to the behavior we need within the form. From the type of button to the available events, we'll look at how to customize it to fulfill its function in a visually effective and functionally accurate way.

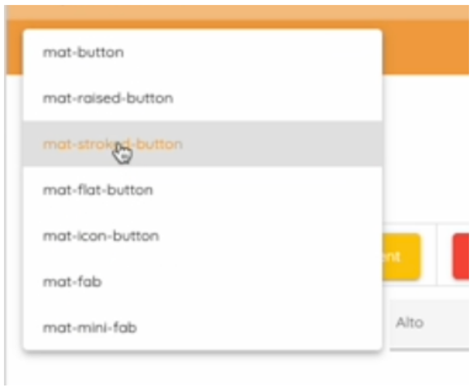
Basic Button Properties

The basic properties define the initial appearance of the button and its overall behavior. Here are the main configurable aspects:

1. Button Type

BIZUIT offers different visual styles for the buttons, each designed for a specific use within the interface:

- **mat-button**: flat button with only text.
- **Mat-raised-button**: Raised button with shade, ideal for highlighting.
- **mat-stroked-button**: button with visible edge.
- **mat-flat-button**: flat button without border or shadow.
- **mat-icon-button**: button with only icon, ideal for quick or repetitive actions.
- **Mat-Fab**: Large floating button, perfect for main actions on mobile devices.
- **mat-mini-fab**: small version of the floating button.

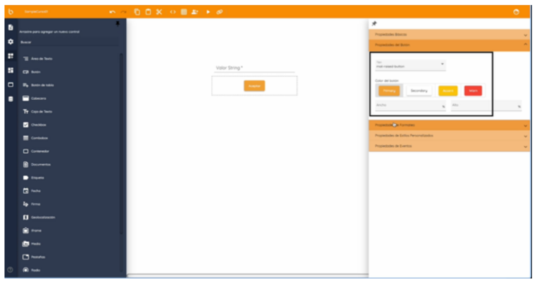


Choosing the right type helps to prioritize actions visually within the form.

2. Button Color

We can define the color of the button according to the intention of the action:

- **Primary**: for main actions.
- **Secondary**: for secondary actions.
- **Accent**: to highlight special interactions.
- **Warn**: For warnings or critical actions (such as delete).



For example, for a "Delete" button, using the Warn color improves the clarity of the action.

In addition, we can adjust the width and height of the button according to the overall design of the form or the needs of the device (percentage or fixed values), ensuring a responsive experience.

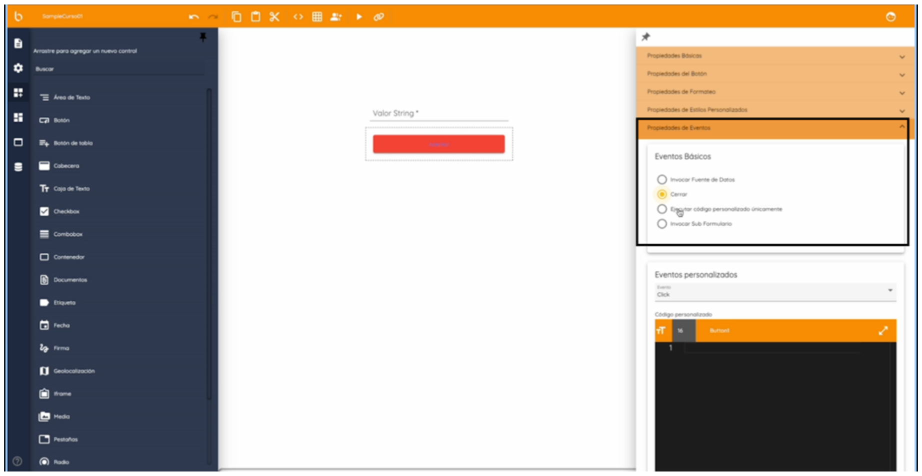
Button Event Settings

Events determine what happens when you click the button. Some options are available in both standalone forms and process-associated forms; others are exclusive to process forms.

Common Options:

- **Close**: Closes the current form.
- **Invoke Data Source**: Runs a configured secondary source. We can include pre-validation and control cleanup after execution.
- **Run Custom Code**: allows us to add advanced logic written in TypeScript.

- **Invoke Subform**: Opens a modal form within the main form, allowing values to be exchanged between the two.



Eventos Básicos: Invocar Fuente de Datos [✓ Cerrar] Ejecutar código personalizado únicamente Invocar Sub Formulario

Specific options for process-associated forms:

- **Invoke Process**:
  - In startup forms: Starts an instance of the process.
  - In interaction activities: the execution of the process continues. Controls linked to parameters or variables automatically pass their values.
- **Traceability Window**: Opens a modal that shows the instance's journey through User Interaction activities up to the current point.
- **End Task**: Ends the process instance, regardless of what stage it is in.

Use Cases and Best Practices

Setting up a button correctly is not only a visual issue, but a functional one. Some key recommendations:

- Use critical colors only for irreversible actions (such as Warn on deletion).
- Make sure that buttons have clear labels that describe their function.
- Test each configured event to confirm that it responds as expected.
- Adapt the size of the button to the device: FABs work especially well on touch screens.

Summary

The Button control in BIZUIT allows us to build more dynamic, interactive interfaces adapted to our functional needs. Its variety of styles, its integration with data sources and processes, and the ability to execute custom logic make it a fundamental tool within form design.

By correctly applying these configurations, we can create user experiences that are intuitive, efficient and aligned with our business objectives.

5. Date Control

At this point we explore the Date control in BIZUIT, a key tool for selecting, displaying, and validating dates within forms. It is especially useful in applications that manage events, expirations, deadlines or any type of temporal data.

Thanks to its configurable properties, we can limit ranges, apply formats, and set defaults that suit the needs of each flow.

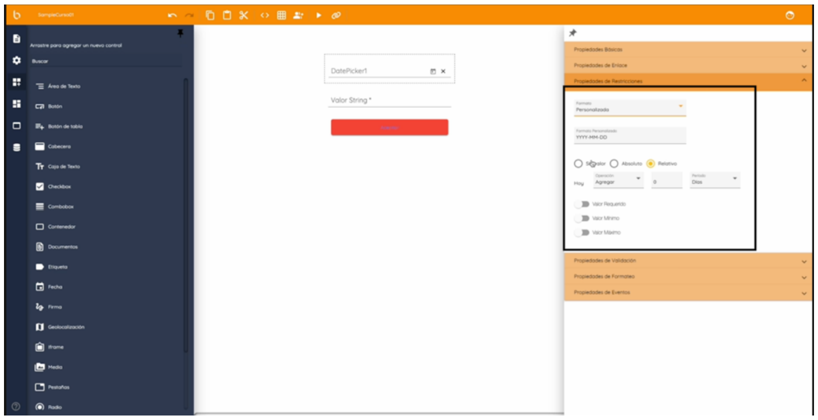
Constraint Properties

The Date control includes multiple settings that allow us to ensure that the value entered is consistent and useful for the process in which it is framed.

1. Date Format

We can choose how the date selected by the user will be displayed. BIZUIT offers different options:

- **Short**: small format, e.g. MM/DD/YYYY.
- **Long**: Includes full text, such as "November 5, 2024."
- **Time**: Displays the date along with the selected time.
- **Custom**: We define a specific pattern such as YYYY-MM-DD.

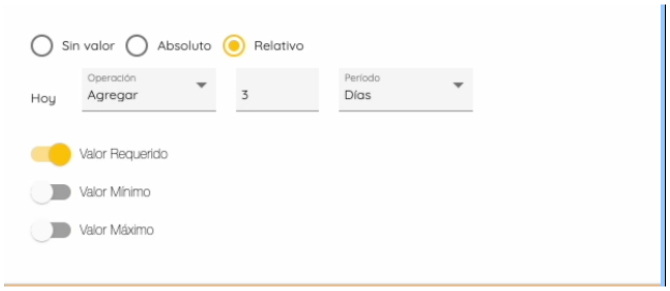


Choosing the right format improves comprehension and reduces errors due to regional or contextual differences.

2. Initial Value: No Value, Absolute or Relative

We can define whether the field is initially empty, with a fixed date or with a relative date:

- **No value**: The field remains empty until the user selects a date.
- **Absolute value**: for example, "05/11/2024".
- **Relative value**: for example, "Today + 3 days".



This is useful for forms that require a predetermined date (such as the current date) or for processes that anticipate a future event.

3. Required Value

When activating this option, the user must select a date before sending the form. It's a simple way to ensure that critical information isn't left out.

4. Minimum Value

We may limit the earliest date the user can enter. This restriction can be configured as:

- **Absolute**: a fixed date such as "01/01/2023".
- **Relative**: for example, "Today + 1 day".
- **Relative operations**: allow us to add or subtract days, months or years from the current date (for example: "Today - 30 days").

This allows us to avoid invalid selections, such as past dates in future event records.

5. Maximum Value

Similarly, we can also set the furthest date that can be selected:

- **Absolute**: "31/12/2025"
- **Relative**: "Today + 90 days".
- **Relative operations**: as in the minimum value, they allow us to adapt the constraints to dynamic rules or to the logic of the process.

Setting a maximum value prevents users from entering dates that are outside of a valid range for the business.

Summary

The Date control in BIZUIT is an essential tool when working with temporal data. It allows us to provide a clear user experience, ensure that the values entered are within valid margins, and adapt the behavior of the form to the rules of the process.
With its format settings, ranges, and dynamic values, this control brings precision and flexibility to any workflow.

6. ComboBox Control

BIZUIT's Combobox control is an essential tool when we need the user to select an option from a drop-down list. Thanks to its versatility, we can populate it with static or dynamic data, adapt it to different flows and significantly improve the experience of using forms.

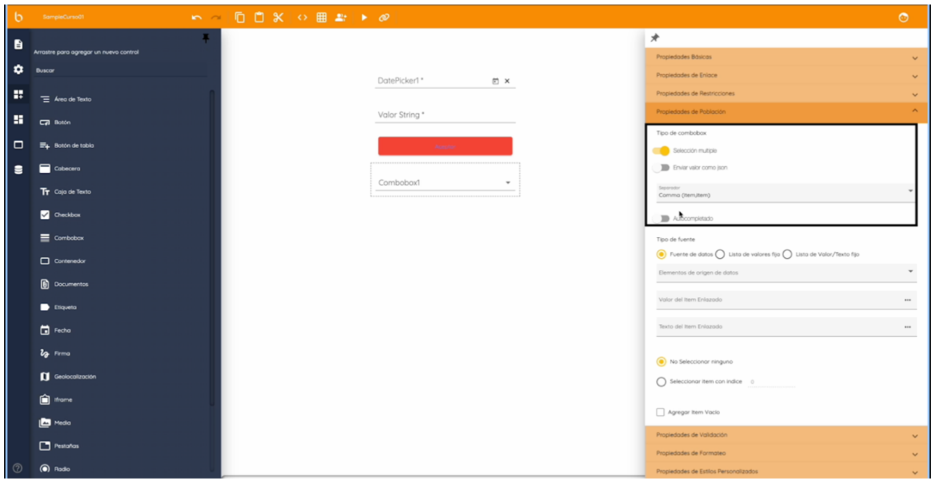
Population Properties

Population properties determine how Combobox options are populated. These can be defined manually (static) or dynamically generated from external sources.
Let's take a closer look at the options available.

1. Population Options

We can configure different behaviors that modify the way the Combobox interacts with the user:

- **Multi-select**: Enables the user to choose more than one option. In addition, we can configure how the selected options are stored and concatenated, which is useful in flows where multiple inputs are required to be captured from a single control.
- **Send value as JSON:** When this option is enabled, the set of selected options is sent in JSON format to the backend. This is ideal for integrations with external systems that expect complex data structures.
- **Autocomplete**: As the user types, the Combobox offers suggestions based on the available list. This functionality speeds up selection on long lists and improves usability.



Tipo de combobox
Selección multiple [✓]
Enviar valor como json
Separador: Comma (item,item)
Autocompletado
Tipo de fuente: [✓ Fuente de datos] Lista de valores fijo / Lista de Valor/Texto fijo
Elementos de origen de datos
Valor del Item Enlazado
Texto del Item Enlazado
[✓ No Seleccionar ninguno] / Seleccionar item con indice 0
Agregar Item Vacío

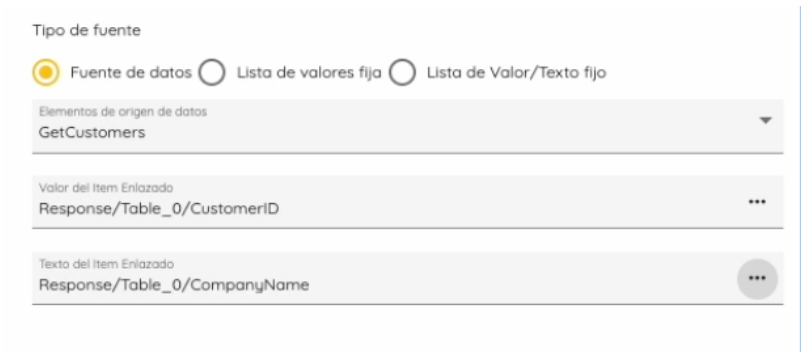
2. Source Type

How the options are loaded depends on the type of font we define:

a. Data Source

This option allows you to populate the Combobox dynamically from an external source, such as a database or REST API.

- We select the secondary data source that contains the values.
- We indicate which field will be used as a value (what is stored) and which as text (what is displayed).
- We can include an empty item to represent the case in which the user does not select any option, assigning it a value and a personalized description.


Tipo de fuente

[✓ Fuente de datos] Lista de valores fija Lista de Valor/Texto fijo

Elementos de origen de datos: GetCustomers

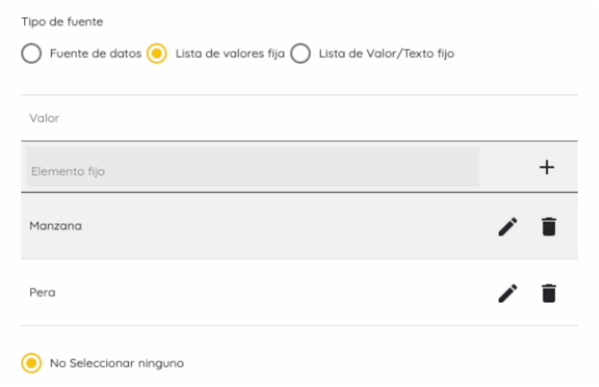
Valor del Item Enlazado: Response/Table_0/CustomerID

Texto del Item Enlazado: Response/Table_0/CompanyName

This setting is ideal when values need to be kept up to date or come from external systems.

b. Fixed Value List

We can manually define a static list of values, useful in forms where the options do not change frequently.


Tipo de fuente

Fuente de datos [✓ Lista de valores fija] Lista de Valor/Texto fijo

Valor

Elemento fijo

Manzana

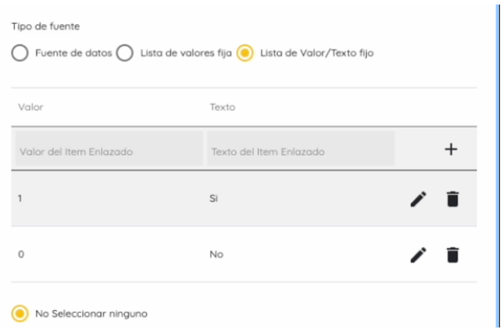
Pera

[✓ No Seleccionar ninguno]

c. Fixed Value/Text List

Here we define key-value pairs, where:

- The key is the data that is saved in the system.
- Text is what the user sees on screen.



Tipo de fuente

( ) Fuente de datos ( ) Lista de valores fija [✓ Lista de Valor/Texto fijo]

Valor	Texto
Valor del Item Enlazado	Texto del Item Enlazado
1	Si
0	No

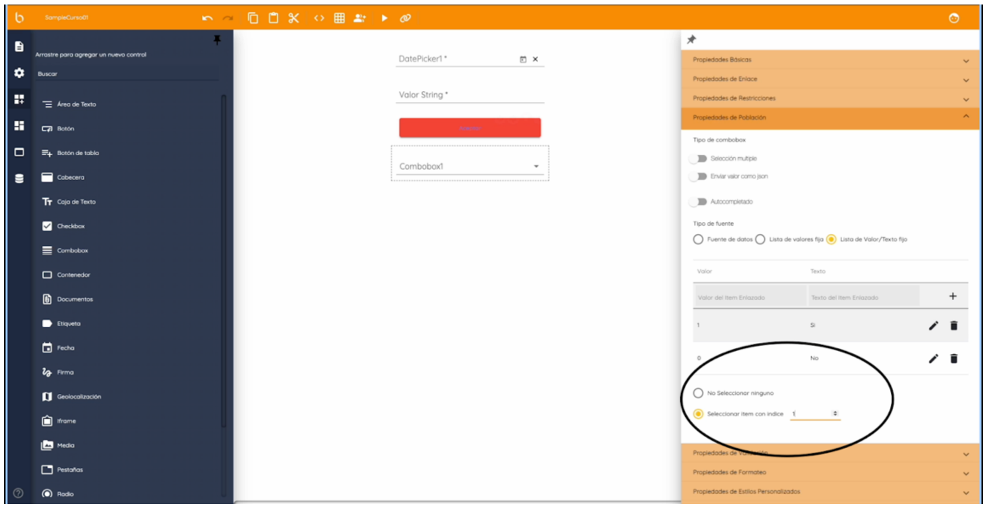
[✓ No Seleccionar ninguno]

This approach allows us to handle internal codes (such as IDs or acronyms) without affecting the clarity of the interface.

3. Additional Options

In addition to defining how the data is loaded, the Combobox offers us other options to adjust its behavior:

- **Select item with index**: we can preselect an option by its position in the list. This is useful for forms that require a default or suggested option.
- **Add empty item**: allows you to show a first option without an assigned value, useful to force the user to make a conscious decision and not leave the preset value by accident.



Tipo de combobox

Selección multiple

Enviar valor como json

Autocompletado

Tipo de fuente

( ) Fuente de datos ( ) Lista de valores fija [✓ Lista de Valor/Texto fijo]

Valor	Texto
Valor del Item Enlazado	Texto del Item Enlazado
1	Si
0	No

( ) No Seleccionar ninguno

[✓ Seleccionar item con indice] 1

Summary

The Combobox control is a powerful tool for improving interaction in forms. It allows us to offer clear, contextualized and easy-to-use lists, either by dynamic loading from external systems or with fixed lists adapted to the business logic.
With its population properties, auto-completion options, and support for multiple selections, we can build efficient interfaces that respect both the rules of the process and the needs of the end user.

7. Label Control

At this point we're going to explore the Tag control in BIZUIT. Although it is not an interactive component, it serves a key function within forms: to display information in a clear, contextual and visually integrated way. It can be used as static text, informational message, or even to expose dynamic data from the user's system or environment.
Thanks to its simplicity and flexibility, this control becomes indispensable in any functional design.

Basic Properties

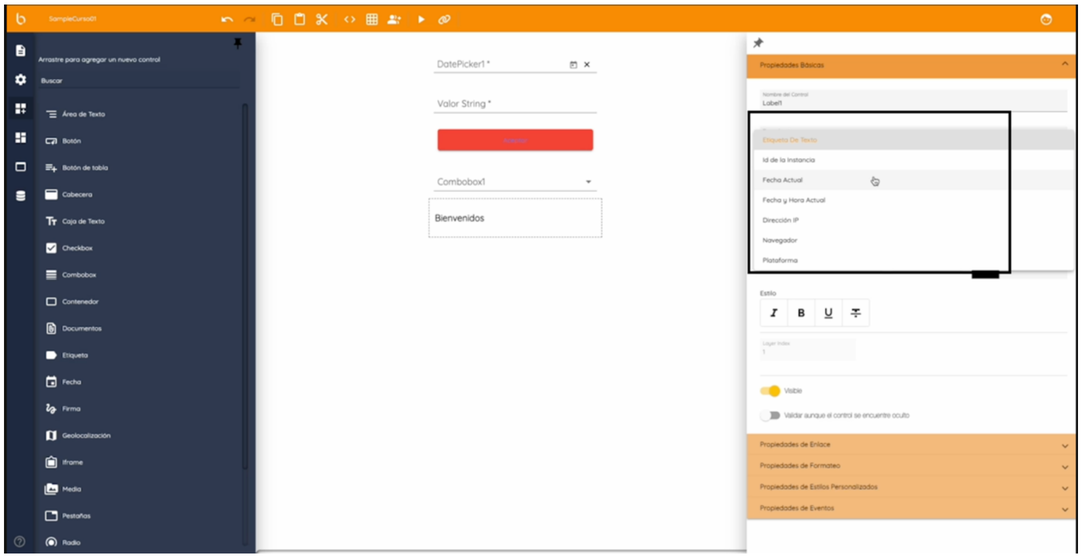
The Label control allows us to define different types of content to display. Depending on the type selected, we may present static data or items that are dynamically updated when the form is loaded.

1. Types of Label
Below we detail the types available and their most common uses:

- **Text Label**
  This is the most basic type. It allows you to display a fixed message or a static label, such as: "Welcome to the system" or "Employee Data". It is useful for titles, instructions, or sections within the form.
- **Instance ID**
  Displays the unique identifier of the in-progress process instance. This data can be useful for audits, cross-referencing, or for the user to keep as a receipt.

- **Current Date:** Displays the system date at the time of uploading the form, in short format. It's useful for validating when an action was performed, recording the creation date, or simply providing context.
- **Current Date and Time:** Similar to the previous one, but adds the hourly value. It is ideal for forms where greater temporal accuracy is needed, such as activity logs, validations, or critical entries.
- **IP address:** Allows you to display the IP address of the device from which the form is accessed. This can be useful in cases of security, traceability, or contextual configurations.
- **Browser:** Displays the browser used by the user when accessing the form (e.g., Chrome, Firefox). This data can be used for technical support or to adapt instructions according to the user's environment.
- **Platform:** Displays the user's operating system (e.g., Windows 10, macOS). This can be useful for specific logs, audits, or visual adaptations.

We can combine different types of tags in the same form to offer the user a more complete and contextualized experience, without the need for additional interaction.



Summary

The Label control in BIZUIT is a simple yet powerful tool to provide context, reinforce form structure, and display useful and timely information.

Whether for static labels, system dates, technical data from the environment or process identifiers, this control improves the overall presentation of the form and brings clarity to the user.
With proper configuration, we can enrich the experience without complicating the design or cluttering the interface. A fundamental tool for those looking for efficient, organized forms focused on relevant information.

8. Text Area Control

The BIZUIT Text Area control is a tool designed to capture large volumes of text, ideal for comments, descriptions, notes or any type of extensive entry. Unlike a simple text field, this control allows you to adjust its size, validate the number of characters, and control the visibility of the content according to the needs of the form.
Let's walk through its most relevant properties and settings to ensure the field is easy to use, visually clear, and technically validated.

Constraint Properties

The Text Area constraint properties allow us to define specific conditions about what the user can or should enter. This is essential to avoid loading errors, standardize the data, and facilitate further processing.

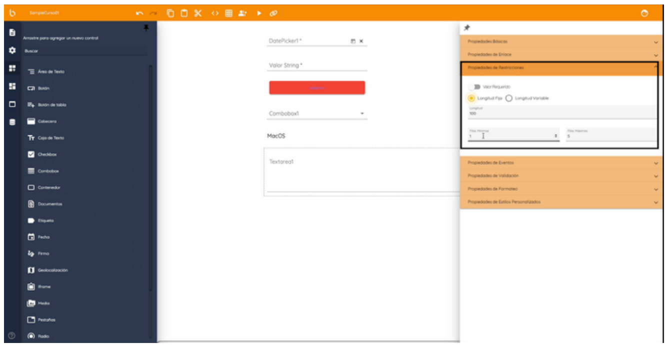
1. Required Value

We can enable "Required Value" validation, which forces the user to enter text before proceeding. It is a common practice in fields where it is necessary to justify an action or leave a mandatory comment.

2. Fixed or Variable Length

To ensure proper input, we may limit the number of characters allowed. There are two variants:

- **Fixed Length:** The text must have an exact number of characters. For example, we can require exactly 100 characters to be entered for a formal description field.
- **Variable Length:** We can define both a minimum length (e.g. at least 20 characters) and a maximum length (e.g. up to 500 characters). This setting is useful to allow some flexibility without losing control over the length of the text.

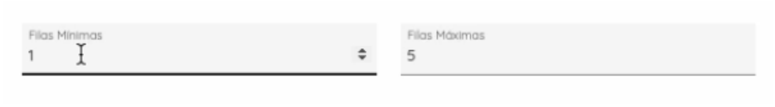


These restrictions are automatically applied at the time of form validation and can be supplemented with custom error messages if the user does not meet the requirements.

3. Minimum and Maximum Rows

In addition to validating the content, we can control the visual size of the text field:

- **Minimum Rows:** Defines the initial height of the text area. For example, 3 rows visible at the time of loading the form.
- **Maximum Rows:** Limits the number of lines visible before the internal scroll is activated. This allows us to display more or less content without compromising the overall design of the form.



Filas Mínimas
1

Filas Máximas
5

This setting improves usability and avoids unnecessary lengthy forms if the text doesn't exceed a certain volume.

Summary

The Text Area control is a versatile and customizable solution that allows you to capture extensive content in a controlled way.

With visualization validations and settings, it easily adapts to different business scenarios, whether it's entering product descriptions, internal observations, or user messages.

With proper configuration, we not only improve the upload experience but also ensure the consistency and quality of the data entered.

9. Control CheckBox

BIZUIT's Checkbox control is one of the simplest and most powerful tools for capturing binary decisions in a form. Whether it's to accept terms and conditions, activate features, or simply register a "yes/no," this component is essential in scenarios where a simple selection defines the course of action.
Let's explore how to configure its constraint and validation properties, so we can integrate it precisely and custom-into our flows.

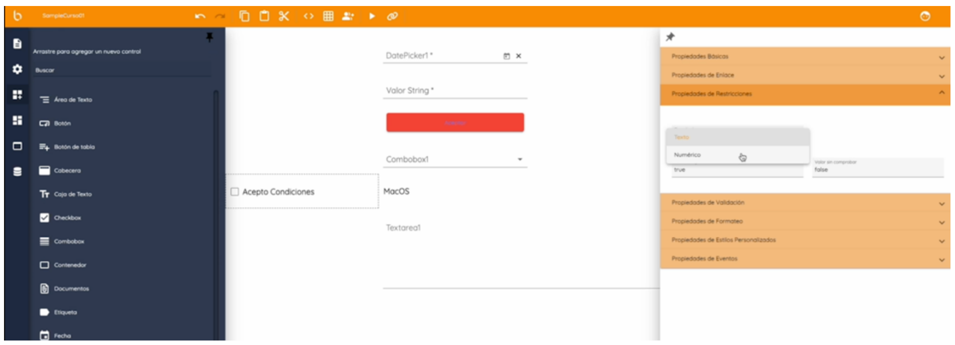
Constraint Properties

The behavior of the checkbox can be adjusted to accurately reflect what each status (checked or unchecked) means for the process or system. Below, we review the main configurations.

1. Type of Validation

The validation type defines how the control state will be interpreted within the form or process:

- **Text:** The status of the Checkbox is represented by values such as "Accepted" or "Rejected". This option is ideal when you need to record decisions in the form of readable text or store more descriptive states.
- **Numeric:** Use values such as 1 for "checked" and 0 for "unchecked." This format is especially useful for integrations with external systems that operate on Boolean values represented in numeric format.

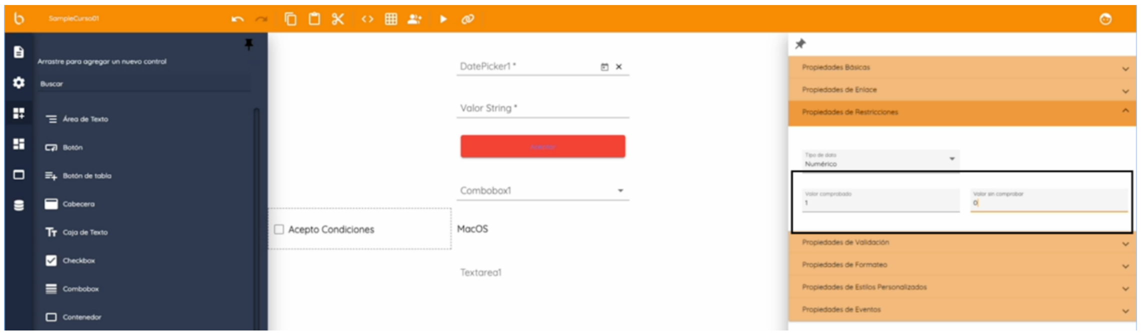


These configurations can be combined with validation and formatting rules to generate more complex behaviors, such as showing or hiding other controls depending on the state of the Checkbox.

2. Custom Values

BIZUIT allows you to define which value will be sent to the system depending on the status of the Checkbox. This is configured in two key fields:

- **Checked Value:** This is the value that will be sent when the Checkbox is **checked**. It can be true, 1, "Yes", "Accepted" or any other string or number depending on what the system needs to receive.
- **Unchecked Value:** This is the value that will be sent when the Checkbox is **unchecked**. It can also be false, 0, "No," or any custom value.



Tipo de dato: Numérico
Valor comprobado: 1
Valor sin comprobar: 0

This enables a direct and clean integration with automated processes, external data sources, or custom validations.

Summary

The Checkbox control is much more than just a checkbox. Its ability to adapt to different types of validation and return custom values makes it a versatile tool for all types of forms. Whether we're building a formal process with multiple conditions or simply need to record a user preference, this component delivers efficiently and clearly.

10. Control Slide Toggle

The Slide Toggle control in BIZUIT represents a visual and contemporary alternative to the classic binary switch. Thanks to its intuitive design, it allows you to switch between two states – such as *On* and *Off* – clearly and efficiently. In this section, we'll explore its constraint properties, ideal for validating its behavior within a form or integrating it with other systems.

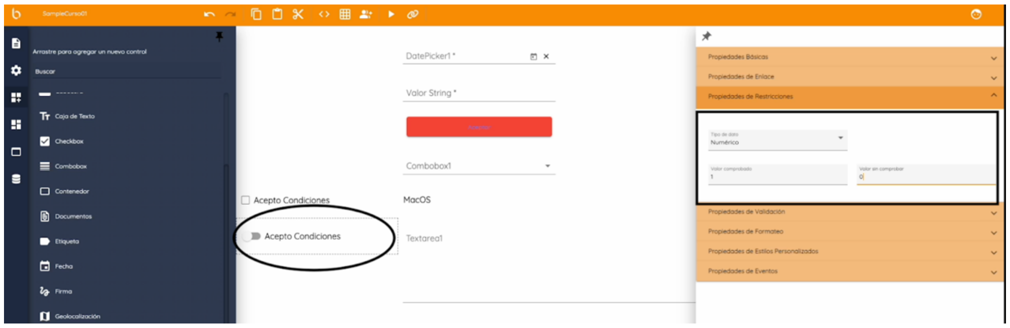
Constraint properties

The constraint properties of the Slide Toggle allow you to control its value and define how it should be interpreted based on system logic or project requirements. They are divided into two large blocks: the validation type and the custom values.

1. Type of validation

This field defines how the Slide Toggle values are interpreted when its state changes:

- Text: The control can be configured to emit readable strings such as "On" and "Off". This option is useful when you need to display end-user understandable statuses or when integrating systems that require text instead of numbers.
- Numeric: In this case, the control is configured to output values such as 1 (for on) and 0 (for off). It is ideal for integrations that handle Boolean states or strictly numerical binary conditions.



Tipo de dato: Numérico
Valor comprobado: 1
Valor sin comprobar: 0

2. Custom Values

In addition to the type of validation, it is also possible to define which specific values should be sent when the Slide Toggle is in one or the other state:

- **Checked Value:** Corresponds to the value that is assigned when the Toggle is enabled. This can be a number (1) or custom text such as "Yes."
- **Unchecked Value:** This is the value that will be sent when the Toggle is disabled. Like the previous one, it can be numerical (0) or textual, for example. "No."

This customization is key when looking for support for databases, external APIs, or business rules that require certain state-specific values.

Summary

The Slide Toggle is much more than just a switch. Thanks to its modern design, it enhances the user's visual experience without sacrificing functionality.
In addition, its ability to adapt to different data formats makes it a versatile tool for any type of project within BIZUIT.

11. Radio Control

The Radio Button control in BIZUIT is designed to capture a single choice among several alternatives. This makes it the ideal tool when dealing with mutually exclusive decisions within a form. In this section, they explore their main properties, with an emphasis on the population options available.

Population properties

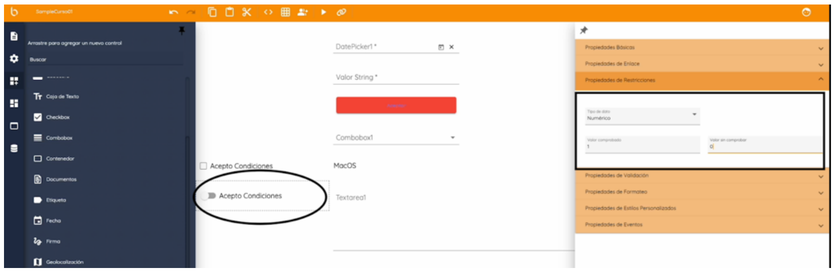
The behavior of the Radio Button can be adjusted according to how you want its options to be filled. BIZUIT offers different ways to define this list, depending on the context or the source of the data.

1. Source Type

The options that the user will see can come from different sources:

- **Data Source:** This option allows you to link the control to a secondary data source. You can select which field will act as a value (what is saved) and which as text (what is displayed). It is especially useful on forms that need to be kept in sync with external databases or that rely on dynamic information.
- **Fixed Value List:** Consists of defining a static list of options. It is the ideal choice when the possible alternatives do not change and are known in advance (e.g., days of the week, priority levels, etc.).
- **Fixed Text/Value List:** Allows you to specify key-value pairs. For example: 1: Red, 2: Yellow, 3: Green. In this case, the key (e.g. 1) is stored while the user sees

the corresponding text ("Red"). This separation between what is displayed and what is stored gives greater flexibility when handling data internally.



2. Select Index Item

Another important feature is the ability to define a default option within the set of buttons. This is done through the index of the list, i.e. its position.

For example, if you want "Red" to be selected from the start, simply assign it the index 0 (assuming it is the first element). This allows you to pre-configure forms intuitively and quickly.

Summary

The Radio Button control in BIZUIT not only simplifies the selection of unique options, but also offers great versatility in the way you populate your data. Whether with fixed lists or through external sources, it allows you to maintain clear, consistent forms aligned with the logic of the system.

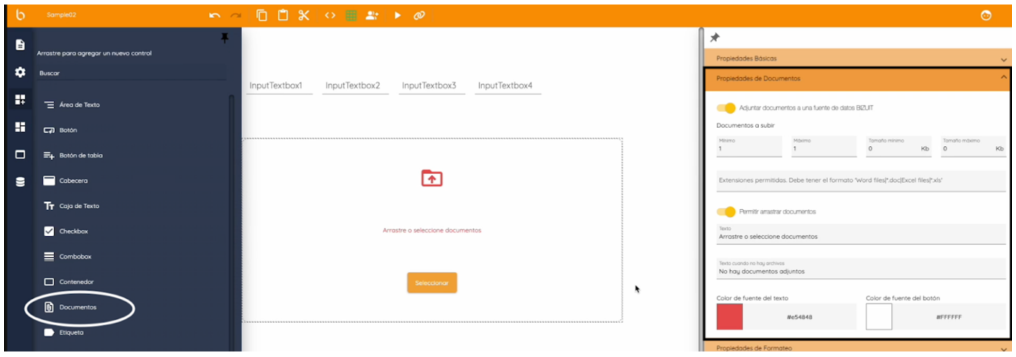
13. Document Control

BIZUIT's Documents control is designed to make it easy to upload, view, and manage files within forms. Its configuration adapts to both standalone forms and those integrated into processes, offering flexibility in multiple scenarios. In this section, we will analyze its properties according to the context of use.

1. On standalone forms

When the Documents control is used within a separate form, the configuration options focus on how and where the files will be stored.

- **Attach documents to a BIZUIT data source:** If this option is not selected, you can manually specify the BIZUIT instance ID to which the documents will be associated. If selected, the files will automatically link to a secondary data source of type BIZUIT, which must be pre-configured.
- **Documents to upload:** This property allows you to set the minimum and maximum number of files that the user can or must upload, as well as the maximum size allowed per file.
- **Allowed extensions:** The type of files that will be accepted can be restricted, for example .pdf, .docx, .jpg, among others. This ensures that only formats supported by the associated process or system are loaded.
- **Allow Document Dragging:** If this option is enabled, the user will be able to drag files directly to the controller, in addition to using the traditional upload button. In this case, it is also possible to customize the visual appearance of the cargo area.



Adjuntar documentos a una fuente de datos BIZUIT

Documentos a subir

Mínimo	Máximo	Tamaño mínimo	Tamaño máximo
1	1	0 Kb	0 Kb

Extensiones permitidas. Debe tener el formato 'Word files\|*.doc\|Excel files\|*.xls'

Permitir arrastrar documentos

Texto: Arrastre o seleccione documentos

Texto cuando no hay archivos: No hay documentos adjuntos

Color de fuente del texto: 🟥 #e54848

Color de fuente del botón: ⬜ #FFFFFF

2. On forms associated with processes

When the Documents control is used on a form that is part of a process, it changes its settings slightly:

- **Show documents from the current instance:** You can indicate that the files are linked to the active instance of the process, or to another specific instance by using another control, parameter, or variable.
- **Allow adding documents:** If this option is enabled, users will be able to upload new files during the execution of the process.

- **Allow Deleting Existing Documents:** It is also possible to allow users to delete previously uploaded files, giving greater control over the content linked to the instance

Summary

The Documents control offers a robust and flexible solution for managing files within forms. Whether in a standalone context or as part of a structured process, it allows you to define clear boundaries, set allowed file types, and control actions such as adding, deleting, or viewing documents.

14. Header Control

The Header Control in BIZUIT allows you to structure and customize the top of a form or screen, integrating logos, titles, subheadings, and menus. This tool not only fulfills an aesthetic role, but also a functional one, as it facilitates the organization of content and improves the user experience. In this section, we walk through its core properties and how to configure them to suit each project.

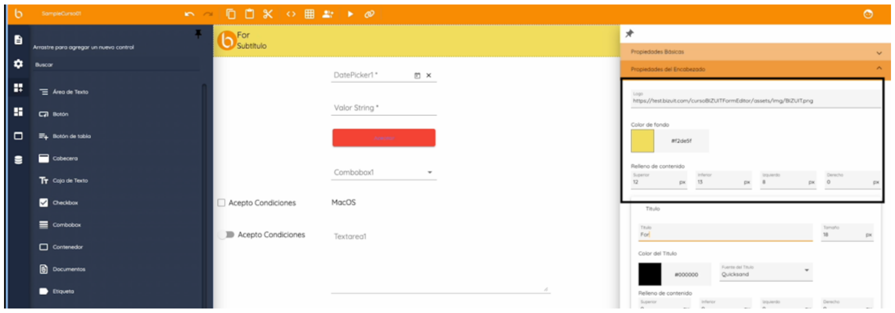
Header Properties

The control is made up of five configurable elements: logo, title, subtitle, menu, and the order of the components. Each can be adjusted in content, style, and layout.

1. Logo

This element allows you to visually identify the application or form:

- **Image source:** A logo can be uploaded using a URL or a base64-encoded image can be inserted.
- **Background color:** Define the background of the header, useful for highlighting the logo or aligning the aesthetic with the institutional identity.
- **Content Fill:** Allows you to adjust the top, bottom, and side margins in pixels, improving visual spacing.



Logo: https://test.bizuit.com/cursoBIZUITFormEditor/assets/img/BIZUIT.png

Color de fondo: #f2de5f

Relleno de contenido: Superior 12 px, Inferior 13 px, Izquierdo 8 px, Derecho 0 px

Titulo

Titulo: For — Tamaño: 18 px

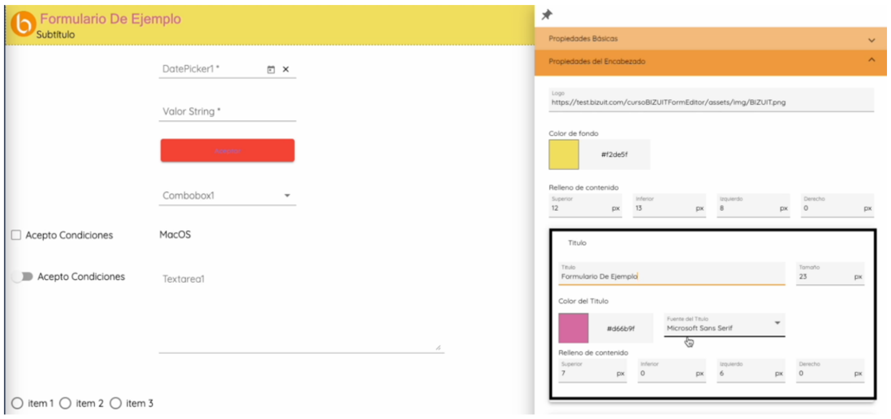
Color del Titulo: #000000 — Fuente del Titulo: Quicksand

Relleno de contenido

2. Title

Usually located next to the logo, it features the name or main theme of the form:

- **Text:** The content of the main title is defined.
- **Style:** The color, font and size of the text can be customized. You can also adjust the margins to control the spacing relative to the other elements in the header.



Logo: https://test.bizuit.com/cursoBIZUITFormEditor/assets/img/BIZUIT.png

Color de fondo: #f2de5f

Relleno de contenido: Superior 12 px, Inferior 13 px, Izquierdo 8 px, Derecho 0 px

Titulo

Titulo: Formulario De Ejemplo — Tamaño: 23 px

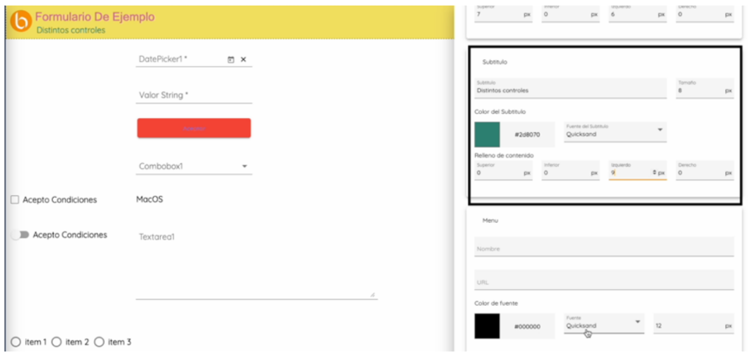
Color del Titulo: #d66b9f — Fuente del Titulo: Microsoft Sans Serif

Relleno de contenido: Superior 7 px, Inferior 0 px, Izquierdo 6 px, Derecho 0 px

3. Subtitle

It acts as a complement to the title, providing additional or contextual information:

- **Text:** Used to add relevant details that clarify the purpose of the form or section.
- **Style:** Like the title, it allows you to adjust font, color, size, and margins.



Superior 7 px | Inferior 0 px | Izquierdo 6 px | Derecho 0 px

Subtitulo
Subtitulo: Distintos controles | Tamaño: 8 px
Color del Subtitulo: #2d8070 | Fuente del Subtitulo: Quicksand
Relleno de contenido: Superior 0 px | Inferior 0 px | Izquierdo 9 px | Derecho 0 px

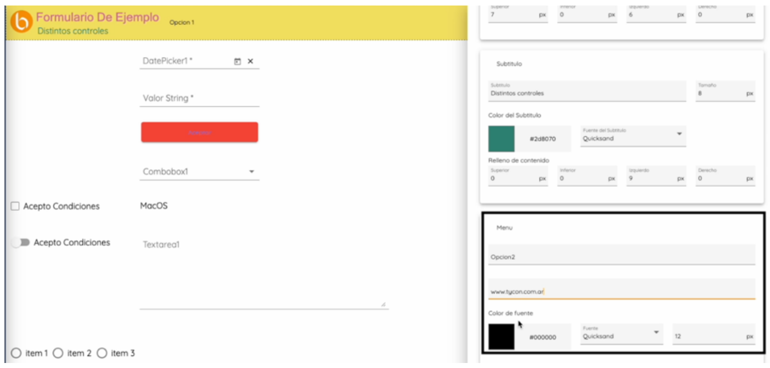
Menu
Nombre
URL
Color de fuente: #000000 | Fuente: Quicksand | 12 px

4. Menu

This component provides quick access to key sections or external resources:

- **Menu Options:**
  - **Name:** This is the visible text in each option.
  - **URL:** Specifies the destination address. It is important that you include http:// or https://, otherwise it will be interpreted as a path relative to the base of the system.
  - **Style:** Fully customizable color, font, and text size.

- **Dynamic Management:** Options can be added, edited, deleted, or reordered by simply dragging in the editor.



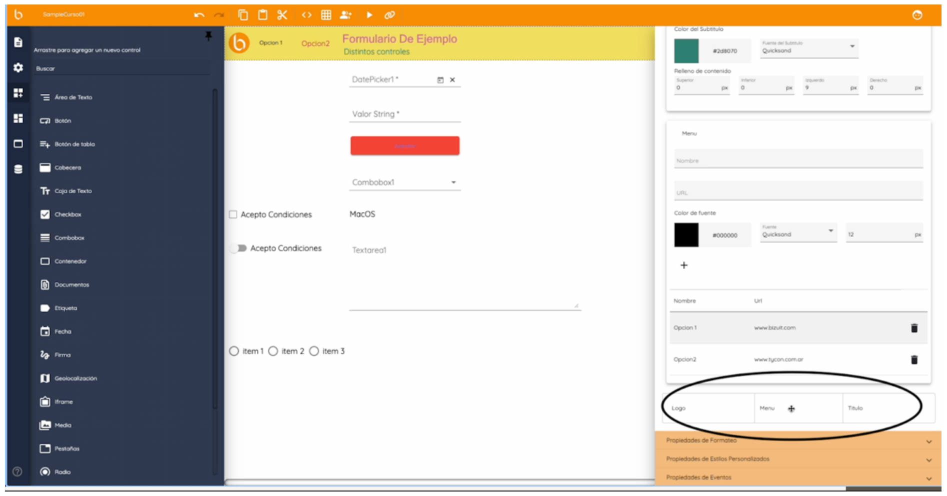
Superior 7 px | Inferior 0 px | Izquierdo 6 px | Derecho 0 px

Subtitulo
Subtitulo: Distintos controles | Tamaño: 8 px
Color del Subtitulo: #2d8070 | Fuente del Subtitulo: Quicksand
Relleno de contenido: Superior 0 px | Inferior 0 px | Izquierdo 9 px | Derecho 0 px

Menu
Opcion2
www.tycon.com.ar
Color de fuente: #000000 | Fuente: Quicksand | 12 px

5. Order of the components

BIZUIT allows you to easily rearrange header elements to fit the desired layout. You just need to drag the components (logo, title, subtitle, menu) into the visual editor until you achieve the ideal layout.



Summary

The Headend Control is a fundamental piece in the construction of clear, accessible and coherent interfaces. Its ability to integrate visuals, informational texts and functional links allows you to consolidate a professional user experience from the first glance. Whether in simple forms or more complex screens, this control ensures identity, order and navigability.

15. Control Table

The Table Control in BIZUIT is a fundamental tool for presenting large volumes of structured information within forms. Thanks to its flexibility and level of customization, it allows you to build from simple lists to interactive tables, adapted to the specific needs of each project.

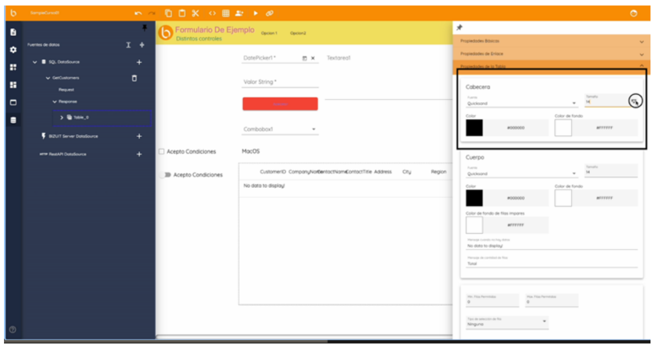
Creation and linking

The process begins by dragging a data structure from a secondary source (such as a SQL query or a previous activity variable) into the form. Doing so automatically generates a Table control with the corresponding columns, already bound to that data source.

1. Header Properties

The header allows you to define the visual style of the column titles, improving the organization and readability of the content.

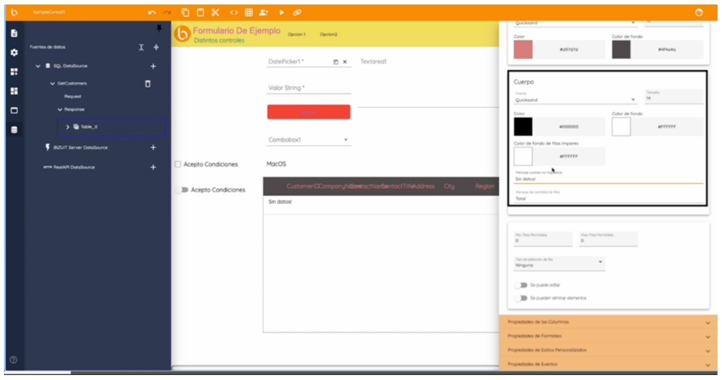
- **Font and size:** Allows you to select the font and its size.
- **Text and Background Colors:** Custom colors can be set to visually align the table with the form design.



2. Body Properties

This section focuses on the contents of the table: the rows and cells where the data is presented.

- **Font and text size:** Allows you to adjust the style of the displayed content.
- **Custom colors:** Can be defined for both text and background, including alternating color schemes for easy reading.
- **Personalized messages:**
    - **No data:** Text that is displayed when the table is empty.
    - **Number of rows:** Text that indicates the total number of visible records.



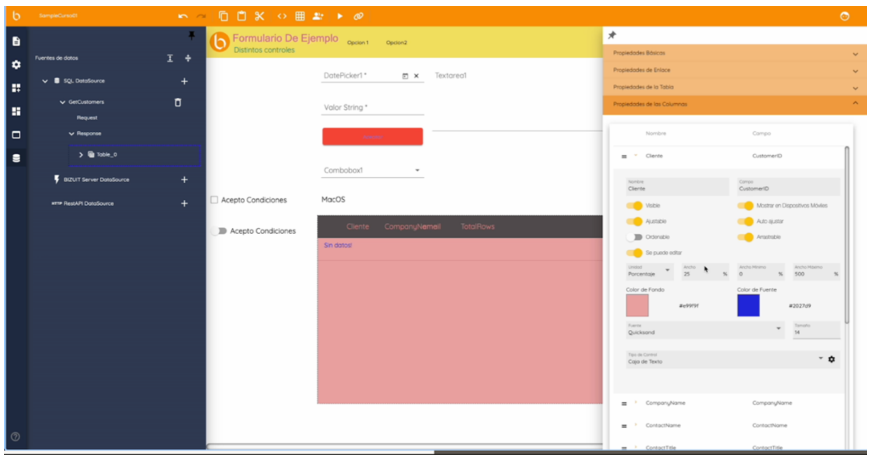
3. Configuring Rows

This section allows you to control how the rows in the table behave and display:

- **Minimum and Maximum Number of Rows:** You can limit the number of visible records.

- **Row Selection Type:**
  - *None:* No interaction.
  - *Singular:* Only one row can be selected.
  - *Multiple:* Allows you to select multiple rows (Ctrl+Click, Multiple Click, or CheckBoxes).

- **Visual customization:** Specific colors can be defined for selected rows and for the hover effect (when the cursor hovers over them).



4. Enabled Actions

The table can be configured so that users not only visualize, but also interact with the data:

- **Row Editing:** Enables direct modification of records from the form.
- **Add Items:** Allows you to add new rows manually.
- **Delete Items:** Provides the option to remove unnecessary or erroneous records.

5. Column Properties

Each column of the control can be configured independently:

- **General Settings:**
  - *Name and field:* Identify the column and its data source.

   - *Visible:* Determines whether the column should be displayed.
   - *Show on mobile:* Optimize the table for small screens.
- **Width adjustment:**
   - *Adjustable:* The user can modify the width manually.
   - *Auto adjust:* Adapts to the content automatically.
   - *Unit of measurement:* Pixels or percentages can be used.
   - *Minimum and Maximum Width:* Define boundaries to avoid columns that are too wide or narrow.
- **Interactivity:**
   - *Sortable:* Allows you to sort the data by clicking on the header.
   - *Draggable:* Allows you to rearrange columns by dragging.
- **Inline editing:** Cells can be made editable and associated with different controls such as text boxes, selectors, or checkboxes.

Summary

The Table Control is indispensable for building powerful and usable forms in BIZUIT. From stylized headers to editable cells, every aspect can be adjusted to suit different business scenarios. Its ability to manage data clearly, dynamically and effectively makes it one of the most complete components of the system.

16. Table Button Control

One of the most practical tools that BIZUIT offers when working with forms that include tables is the Table Button control. This component allows us to manage records quickly and orderly, either to add, edit or delete data directly from the form. Thanks to its design, it optimizes workflow and significantly improves the user experience.
Let's see how it works and what possibilities it offers us.

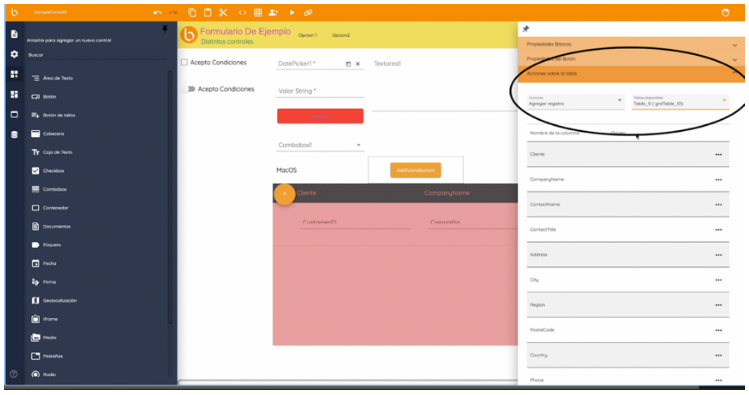
Actions on the Table

When we incorporate a Table Button, the first thing we must do is link it to a specific table of the form. This is done from a drop-down list, and it's a key step: every action we configure with that button will be limited exclusively to that table, which gives us more precise control and avoids errors.
From there, we can configure three types of actions:

1. Add Registration

This action allows us to add new data to the table automatically. To set it up:

- We indicate which values we want to assign to each column of the table.
- Those values can come from constants, other form controls, or system parameters.



For example, we could set a constant value as "TEST" for the CustomerId field, and at the same time link the CompanyName field to whatever the user enters in an InputTextbox1.

When you run the form, clicking the button adds a new record with those defined values.

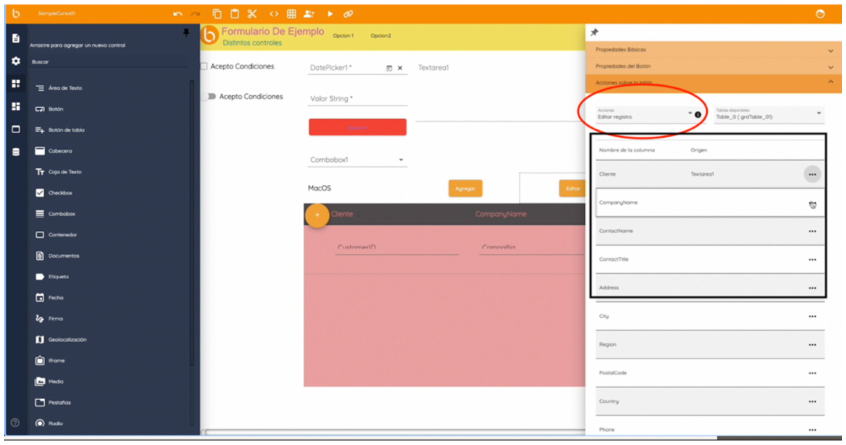
2. Edit Registry

With this action, we can modify an existing record. Its operation is very intuitive, but there is a fundamental requirement: First we must select the record we want to edit.

Once selected, we can:

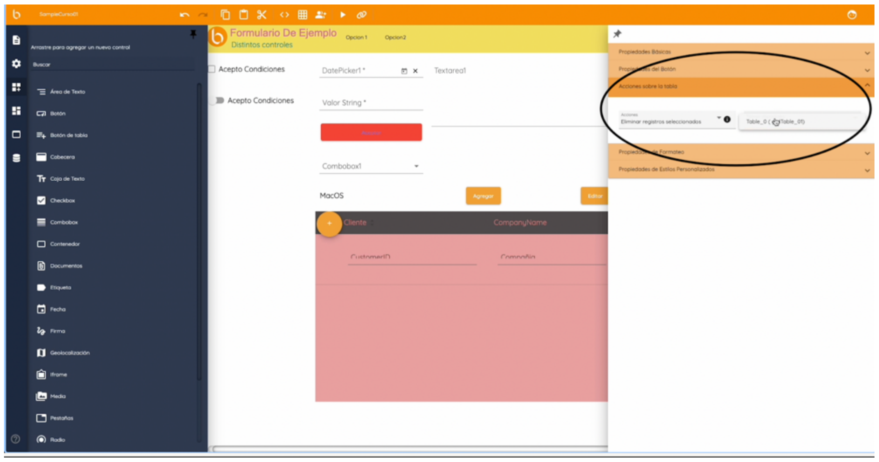
- Link column values to form controls.
- When you select the record, the controls automatically take the corresponding value.
- If we modify the contents of the control, that new value is automatically updated in the table.

For example, we can make the CustomerId field link to a TextArea1 and CompanyName relate to InputTextbox1.



3. Delete selected records

This action allows us to delete one or more records from the table efficiently. You just need to select the rows you want to delete and click on the button. Ideal for keeping the board clean and updated.



Associating the Button with a Table

It is important to remember that each Table Button must be associated with a specific table, which is defined from a drop-down list in the control's properties. This ensures that the actions we configure do not interfere with other tables in the form.

Summary

The Table Button control is a key piece when we need to work with records within a form. It allows us to:

- Add new data

- Edit existing records.
- Delete selected records.

In addition, by being able to link values to constants, controls or system parameters, and limit each button to a specific table, we gain flexibility and control.

A simple, yet powerful tool that cannot be missing from our dynamic forms.

17. Control Tab

When we work with complex forms, we need a practical and visual way to organize information. That's where tab control comes in. This component allows us to divide the form into differentiated sections, making it easier to navigate and improving the user experience.

Basic Properties

As soon as we add the tab control, we access a series of properties that allow us to customize its general appearance:

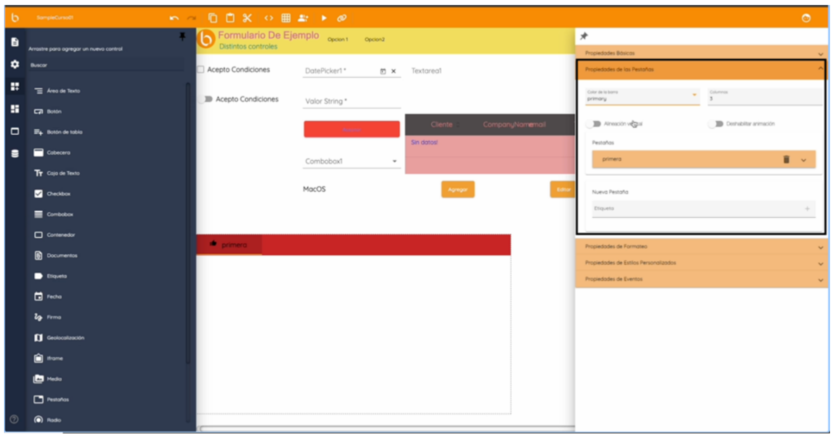
- **Background color:** we can define a color that highlights the area of the tabs or that harmonizes with the general aesthetics of the form. This helps to visually separate the sections and guide the user.

Eyelash Properties

Each tab can be independently customized to fit the design we're building. Some of the most relevant options are:

- **Bar color:** we choose between predefined values of the theme such as Primary, Secondary, Accent or Warn, depending on the visual intention we want to convey.
- **Columns: We** can define how many columns will be available within each tab. This is very useful for organizing controls in an orderly manner.
- **Vertical alignment: allows** us to adjust how elements are distributed in the tab, which can be key for certain interfaces.

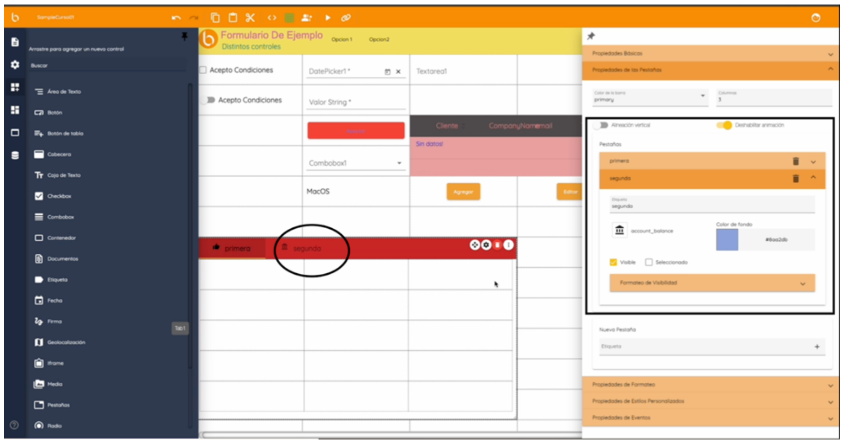
- **Disable animation:** when we need a faster or minimalist experience, we can eliminate smooth transitions between tabs.



Tab Management

In addition to the layout, each tab has a number of individual settings that determine its behavior and visibility:

- **Label**: This is the text that is displayed to the user to identify the tab.
- **Icon**: we can add an icon next to the text to reinforce the idea or function of the tab (for example, a user icon for a profile section).
- **Background Color**: Another layer of customization to visually differentiate each tab.
- **Visible**: Defines whether or not that tab is displayed. Ideal when working with conditional rules.
- **Selected**: Indicates which tab we want to be active when the form is uploaded.



Within each tab, we simply drag and drop the controls that we're going to need. This allows us to build complete and functional sections in a modular way.

Visibility Format

One of the most powerful features of this control is the ability to configure dynamic visibility rules.

We can make certain tabs appear or hide based on conditions that we define based on the data in the form.

The rules are built by combining:

- **Conditions:** Based on controls, operators (such as equal, different, greater than, etc.), and defined values.
- **Condition joining:** We can combine multiple conditions with AND or OR logic.
- **Action:** We define whether the tab should be shown or hidden when those conditions are met.

A simple example would be to hide a tab if the user types the word "HIDE" in a specific field. This type of logic allows us to adapt the interface in real time, improving the usability of the form.

Summary

In short, tab control is an essential tool when we want to organize information and improve the user experience within more complex forms. It allows us to split, hide, show and customize sections in a very flexible way.

Thanks to its extensive configuration possibilities – from aesthetics to dynamic behavior – we can build cleaner, more intuitive and more effective forms.

18. Card Control

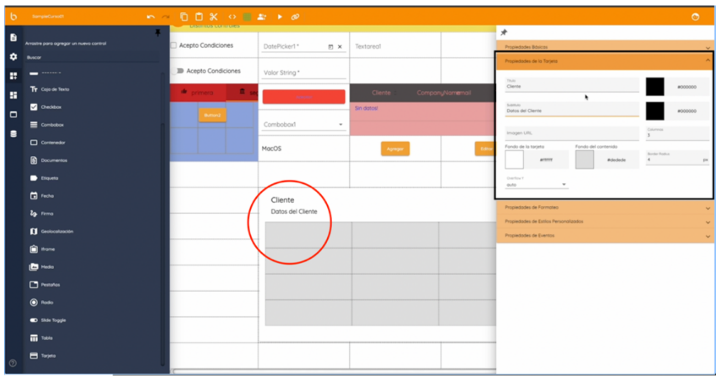
In this section we are going to learn about the Card control, a very useful visual element when we want to group and highlight key information within a form. Thanks to its structured and attractive design, it allows us to organize data in a clear and hierarchical way.

Card Control Properties

When you incorporate a card into a form, you'll see that you have several configurable properties. Let's go through them one by one to understand how to get the most out of them.

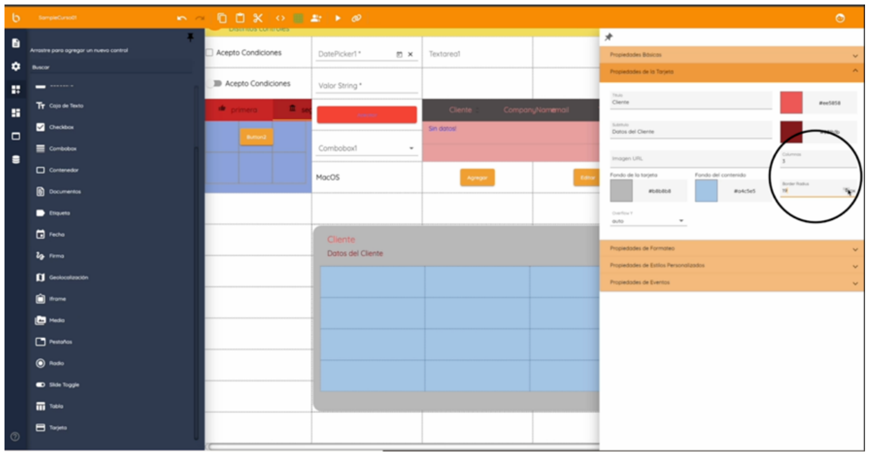
1. Title and subtitle

We can assign a main title that acts as the header of the card. It is also possible to include a subtitle, useful to provide complementary information. In both cases, we can customize the color of the text using hexadecimal values. For example, #000000 for black.



2. Columns

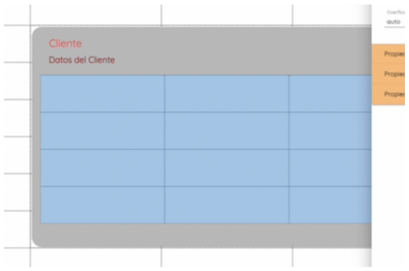
This property allows us to define the number of internal columns within the card. Thus, we can distribute different controls (such as text, buttons or images) in parallel, achieving a more orderly and visually balanced design.



3. Card and content background

The control offers us two options to work with background colors:

- **Card background**: Changes the overall color of the component.
- **Content background**: affects only the area where we place the internal elements.



This gives us the possibility to play with contrasts and visual hierarchies.

4. Rounded Edges (Border Radius)

Through this property we can define the style of the corners:

- 0 px: straight corners.
- 4px, 8px or more: rounded corners, which smooth out the overall design.

5. Overflow Y

When the contents inside the card exceed its visible size, we can control its vertical behavior with the Overflow Y property:

- **Auto**: Adds a scroll bar.
- **Hidden**: Hides excess content.
- **Visible**: Allows content to overflow without restriction.

6. Image

We can also include an image by specifying a URL. This is very useful if we want to visually highlight a section or complement textual information with graphic content.

Summary

The Card control is a very versatile tool within the BIZUIT ecosystem. It allows us to present information in a way that is clear, engaging, and adaptable to the overall design of our forms. If we adjust its properties correctly, we can transform a basic section into a neater and more professional visual experience.

19. Container Control

When we design forms in BIZUIT, we often need to group, order, or simply visually structure the different elements that make it up. For this, we have a fundamental tool: the Container control.

This component allows us to not only group controls within a single section, but also customize their appearance and layout to improve the organization and visual experience of the form.

Let's walk through its configurable properties one by one to understand how to use it effectively.

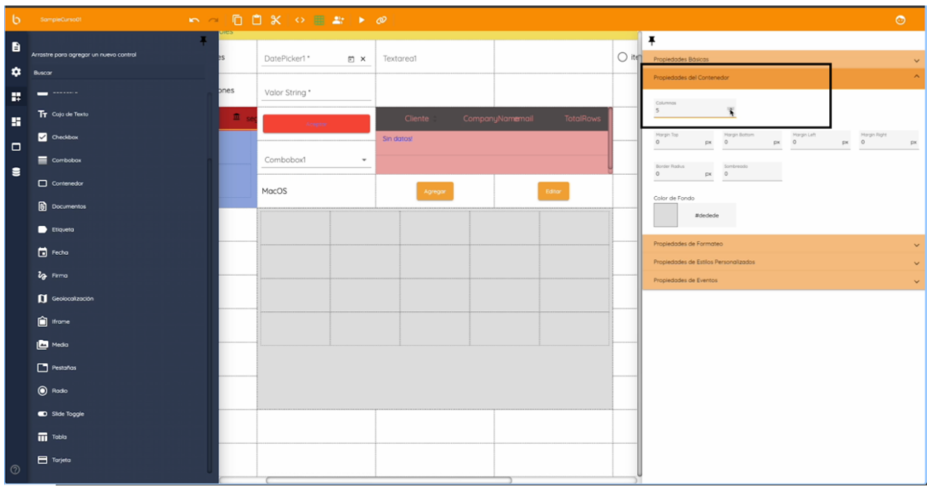
Container Properties

Through its configuration options, the Container control gives us a lot of flexibility to define both the aesthetics and the functionality of the space it groups.

1. Columns

One of the most useful properties is the ability to define how many columns the container will have. This allows us to distribute internal controls in a grid format, in an orderly and adaptable way.

For example, if we configure the container with 5 columns, the controls we add will be automatically accommodated according to that structure.



2. Margins (Margin Top, Bottom, Left, Right)

We can control the outdoor space surrounding the container using the margins:

- **Top and Bottom:** adjust the vertical space.
- **Left and Right:** control the horizontal space with respect to other elements.

This is very useful for maintaining a clear visual separation between sections or components.

3. Border Radius

The edge radius defines how rounded the corners of the container will be.

- If we use a value of 0 px, the design will have straight corners.
- As we increase the value (e.g. 8 px), the corners are rounded, giving a more modern and smooth appearance.

4. Shading

Another interesting aesthetic property is shading, which adds depth to the container.

- A value of 0 implies no shadow, ideal for flat designs.
- Higher values increase the intensity of the shadow, generating a three-dimensional effect that makes the container stand out against the background.

Applying subtle shading can help visually guide the user to important blocks of the form.

5. Background Color

We can customize the background color of the container using a hex code, which allows us to adapt it to the overall design of the form.

For example:

- #dedede for a light gray.
- #ffffff for white.
- Or any other color we need according to the visual identity of the project.

This also helps to visually distinguish sections or groups of controls.

Summary

The Container control is much more than just a control grouper. It allows us to:

- Visually organize forms.
- Apply consistent styles.
- Work with columnar structures.
- Reuse or move control groups with ease.

Thanks to its customizable properties, it becomes an essential tool when we are looking for neat, clear, and visually appealing forms.

20. IFrame Control

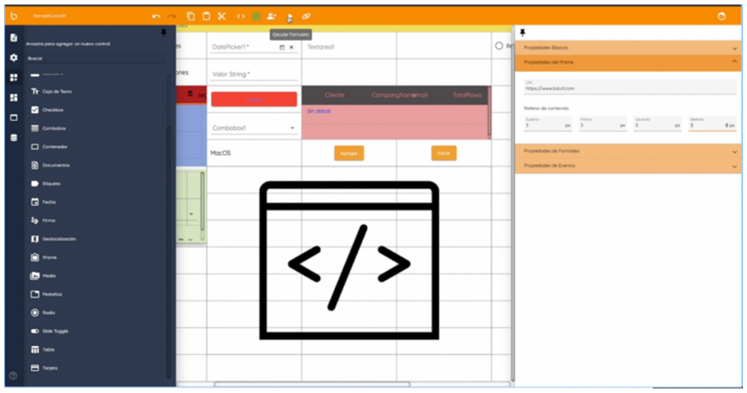
In certain projects, we need to embed external content within our forms: from a web page to charts, dashboards, or even embedded applications. For this, at BIZUIT we have the IFrame control.
This component allows us to integrate that content without the user having to leave the form interface, making the experience much more fluid and professional.

Properties of IFrame

Once we add an IFrame control to the form, we can define two key properties for it to work properly:

1. URL

This is the most important property: it allows us to specify what content is to be displayed within IFrame.



By doing so, the IFrame will load that page directly within the form, allowing the user to interact with it without leaving the BIZUIT environment.

2. Content Stuffing

Another useful option is the ability to configure the IFrame's internal padding. This defines the space between the edges of the component and the content being displayed.
For example, we can apply a 3-pixel padding on all sides so that the content does not stick to the edge and looks cleaner.

This is especially useful when we embed resources with our own interfaces, such as graphs, maps or external forms, as it improves the overall aesthetics.

Summary

The IFrame control is a simple but very powerful tool that allows us to expand the possibilities of our forms with dynamic external content. From informational pages to third-party solutions, we can integrate it all without losing cohesion in the design.
Thanks to its easy configuration, it becomes an excellent option when we need to display external information in an integrated and professional way.

21. Geolocation Control

When we need to work with specific maps, locations or coordinates within a form, the Geolocation control becomes an indispensable tool. This component allows us to visualize geographic positions, link data to locations, and capture coordinates entered or selected by the user.

Geolocation Control Properties

The controller has several settings that determine how the map is displayed and how the user interacts.

1. Bookmark and Information

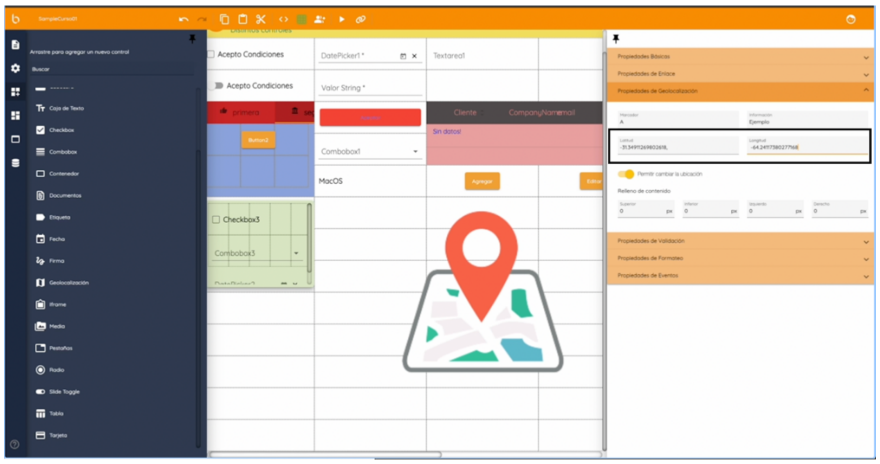
Once the control is inserted, we can set up a marker that points to a specific location within the map.

- Marker: Visually represents the exact point on the map. For example, we could locate a company like "TYCON SA" in its actual coordinates.
- Information: Allows us to associate descriptive text with the marker, such as "Main Office" or a specific address. This enriches the visual experience and provides context

2. Latitude and Longitude

Another key configuration is the possibility of defining exact coordinates to position the marker.

- Latitude: A value between -90 and 90 that determines vertical location.
- Length: Value between -180 and 180 for the horizontal location.


Marcador: A
Información: Ejemplo
Latitud: -31.34911269802618,
Longitud: -64.24117380277168
Permitir cambiar la ubicación
Relleno de contenido
Superior: 0 px
Inferior: 0 px
Izquierdo: 0 px
Derecho: 0 px

We can manually enter these values or associate them with controls or variables, which is useful when the map needs to be updated based on previously entered data.

3. Allow Location Change

This option allows us to define whether the user can move the marker directly on the map:

- If enabled, the user can drag the marker, and the coordinates are automatically updated.
- If it is disabled, the marker remains fixed, useful when we only want to show a location without the possibility of modifying it.

4. Content Filling

Finally, we can configure the internal padding of the control, that is, the margins that separate the map from the edge of the component.
This is defined in pixels for each side (top, bottom, left, and right), and allows us to achieve a cleaner and more visually comfortable design.

Common Use Cases

Geolocation control can be applied in multiple contexts. Below, we review some of the most common:

- Logistics Applications: to locate delivery or collection points, allowing the user to select the exact address.
- Tracking Systems: Display real-time locations, such as the position of a vehicle, device, or person.
- Interactive Mapping: Allow the user to mark relevant locations such as branches, properties, sites of interest, etc.

Summary

The Geolocation control provides a very powerful visual and functional layer to our forms. It allows us to work with geographic data in a simple, interactive and adaptable way.
Whether it's displaying a location, allowing the user to select one, or associating coordinates with dynamic information, this component is key for projects that require integration with maps.

22. Signature Control

When a form requires some form of personal validation, whether it's approving conditions, confirming deliveries, or digitally signing a document, the proper control is the Signature control.

This component allows the user to draw their signature directly on the form, offering a simple, modern and fully digital solution for processes that previously required paper or external instances.

Let's see how it is configured and in which cases it is especially useful.

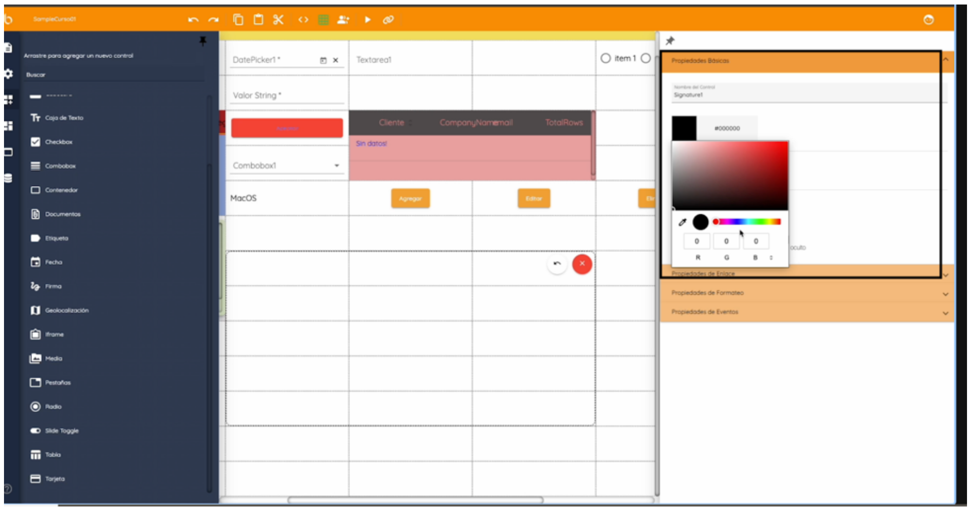
Basic Properties

The Signature control is easy to set up and adapts well to different situations. Among its most relevant properties we find:

Ink Color

We can customize the color of the ink used for the signature.

- By default, the color black is used.
- However, it is possible to choose other colors–for example, blue or red–depending on the visual style of the form or the customer's preferences.



Nombre del Control: Signature1

#000000

R	G	B
0	0	0

This may seem like a minor detail, but it helps to reinforce the overall aesthetic or visually differentiate signatures between different users.

Common Use Cases

Signature control integrates seamlessly into numerous workflows, especially those where some form of direct user authentication or validation is required. Here are some examples:

1. Approvals and Consents

Ideal for situations where the user must agree to terms and conditions, validate information or authorize a procedure. With this control, you can sign directly on the form, streamlining the process and giving it immediate validity.

2. Delivery Processes

It is also very useful in logistics processes: for example, when we need the receiver to confirm the delivery of a product or service. The digital signature acts as proof of receipt, replacing traditional paper.

3. Electronic Forms

On general forms, such as:

- Event registrations.
- Medical authorizations.
- Surveys with validation sections.

The Signature control becomes a versatile tool that improves the presentation of the form and facilitates the collection of signed data without having to leave the digital environment.

Summary

Signature control allows us to incorporate personal validations in a professional, simple and completely digital way. It brings fluidity to processes, improves the user experience, helps us reduce paper usage and processing times.

Thanks to its ease of use and customization options, it integrates seamlessly into all types of modern forms, whether for approvals, deliveries, or registrations.

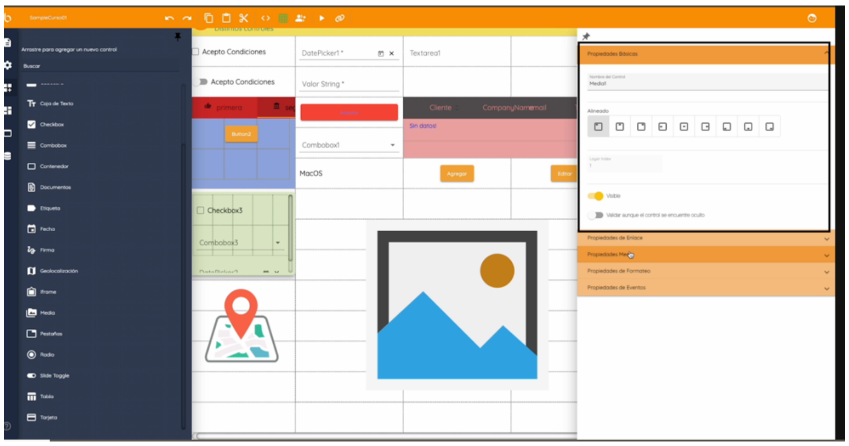
23. Media Control

When we want to enrich our forms with multimedia content – whether images, videos, audios or even access to the camera – BIZUIT's Media control is the right tool.
This component allows us to integrate visual and sound elements directly into the interface, providing dynamism and improving the user experience

Basic Properties

Before defining the type of media we want to incorporate, the Media control allows us to adjust some general properties related to its presentation:

- **Aligned**: Defines the position of the content within the control area (left, center, or right), useful for achieving an accurate layout in the layout.
- **Layer Index:** Sets the priority of control over other items on the same screen. The higher the value, the higher it is displayed.
- **Visible**: Allows you to indicate whether the content will be displayed from the beginning or if its visibility will depend on a dynamic condition.



Properties by Content Type

One of the main advantages of the Media control is that it supports multiple types of content, each with its own specific properties. Let's look at them one by one.

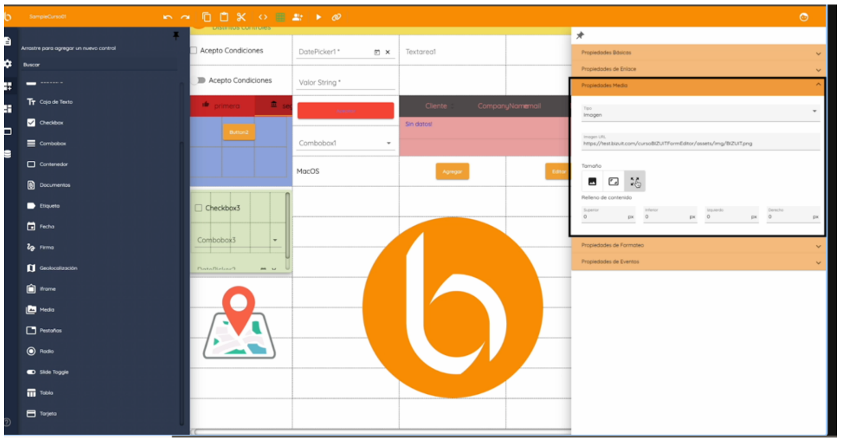
1. Image

Ideal for displaying static visuals such as logos, banners, graphics, or illustrations.
Specific properties:

- **Image URL:** Path of the image file to be displayed (this can be an external URL or a file uploaded to the system).
- **Size:** Defines how the image fits the container:

    - Adjust content.
    - Scale proportionally
    - Fill Completely

- **Content Fill:** Internal margin between the image and the edges of the control, useful for a cleaner design.

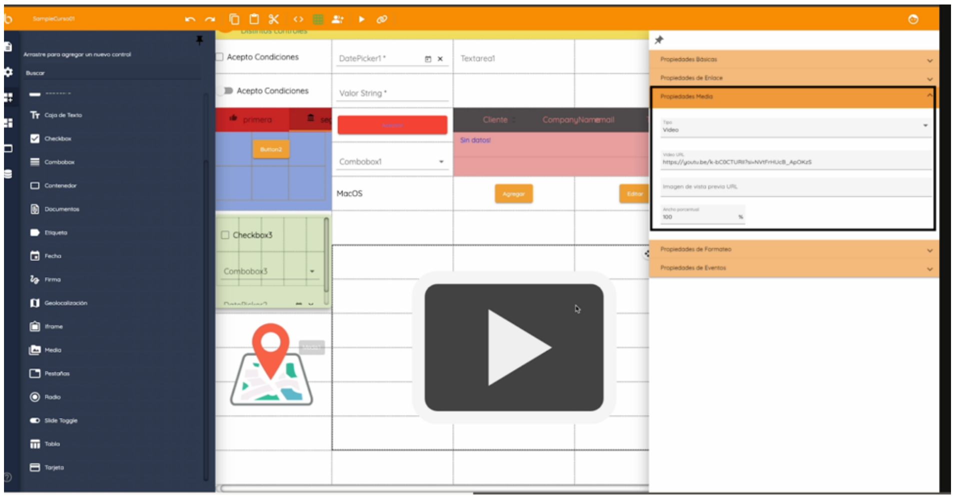


Tipo: Imagen
Imagen URL: https://test.bizuit.com/cursoBIZUITFormEditor/assets/img/BIZUIT.png
Superior: 0 px
Inferior: 0 px
Izquierdo: 0 px
Derecho: 0 px

2. Video

Very useful for incorporating dynamic content such as tutorials, demos, or visual presentations.
Specific properties:

- **Video URL:** Path of the video file.
- **Preview Image URL:** A static image that is displayed before starting playback, functioning as a thumbnail.
- **Percentage width:** Allows you to set what percentage of the available width the video will occupy.

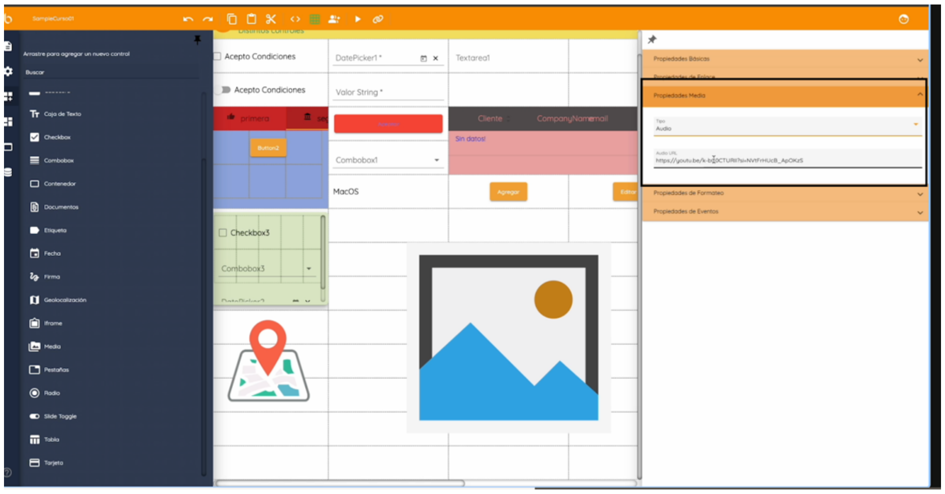


3. Audio

When we want to incorporate music, voice messages, podcasts or sound effects, this is the right option.

Specific Property:

- **Audio URL:** Path of the audio file to be played. The controller includes a built-in player so the user can start, pause, or control the volume directly.



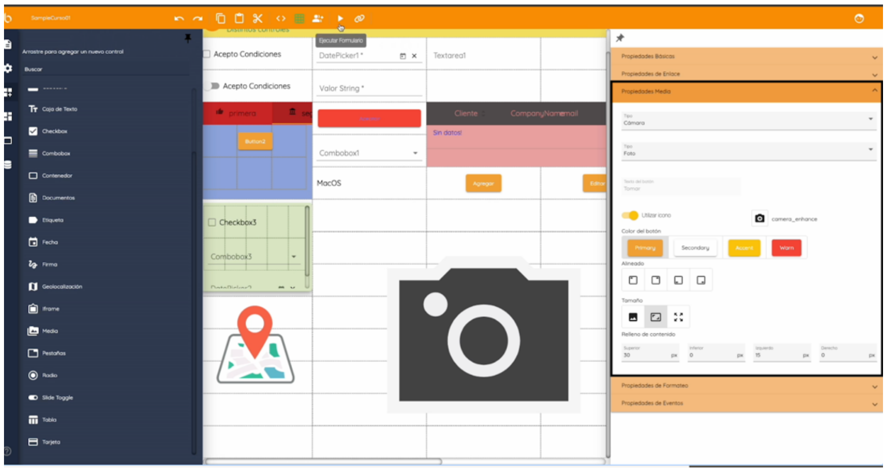
4. Camera

An advanced option that allows us to activate the device's camera, either to capture images or read QR codes directly from the form.

Specific properties:

- **Type:** We indicate whether the use will be to capture a photo or to scan QR codes.

- **Size:** As with images, it allows you to define how the content adapts to the available area.
- **Content Fill:** We can add internal margins for better visual integration.
- In addition, we can include custom buttons with icons to start the camera or capture the image, improving user interaction.



Summary

The Media control is an extremely versatile tool that allows us to integrate multimedia content into our forms in an orderly, functional and visually attractive way.

Whether we need to display an image, play a video, include an audio, or access the device's camera, this component gives us all the options needed to build interactive experiences within our applications.

Conclusion

Throughout this unit, we went through in detail a wide variety of controls available in BIZUIT, understanding how to configure, customize, and combine them to build powerful, visually appealing, and highly functional forms.

We learned how to work with basic elements such as text boxes, headers, and forms, and moved on to more complex components such as tables, action buttons, tabs, and geolocation controls, all of which are critical to managing data and interactions efficiently.

We also explore visual controls that elevate design, such as cards, media, and IFrames, and specific tools for capturing information, such as Signature control or QR code reading using the camera.

Throughout the tour, we saw how to use properties, bindings, validations, and conditions to tailor each component to the needs of each project.
With this knowledge, we are prepared to design dynamic and professional forms, integrating data, automating processes, and improving the end-user experience.

BIZUIT gives us the flexibility to respond to almost any scenario, and we now have the tools to make the most of it.

Congratulations on completing this unit! See you in the next one, where we'll explore in depth the use of secondary data sources and how to leverage them to feed our forms with dynamic and contextual information.

Unit 3: Secondary Data Sources

In this unit we will address Secondary Data Sources in BIZUIT, exploring how to use and configure different types of data sources to integrate external information and internal processes directly into forms.

Secondary data sources in BIZUIT are tools that allow interaction with external databases, native platform processes or APIs directly from forms, facilitating consultations, validations and visualization of data in real time. These sources enrich the user experience by integrating dynamic information without relying on the logic of the main process. They can be SQL, BIZUIT, or RestAPI, and their use optimizes efficiency and flexibility in data management within forms.

We will see how to use:

- SQL sources, for querying and working with external databases.
- BIZUIT fonts, which allow you to execute native logic of the platform within the form.
- RestAPI fonts, ideal for connecting to external web services using HTTP methods.

In addition, we will learn how to optimize the performance of our cached queries, avoiding unnecessary calls.

Secondary data sources are key to building more dynamic, integrated, and efficient forms, without the need to modify the process flow.

SQL Secondary Data Source

In BIZUIT, SQL secondary data sources allow us to interact with external databases directly from the form, without relying on process logic. This gives us the possibility to consult, insert, update or delete data in real time, integrating dynamic information and enriching the user experience.
This type of font is ideal for:

- Load dynamic lists.
- Validate data entered by the user.
- Display external information without altering the process structure

In this section, we're going to take a step-by-step look at how to set up and use a SQL font with a practical example.

Example: Obtaining Customer Information

1. Initial setup

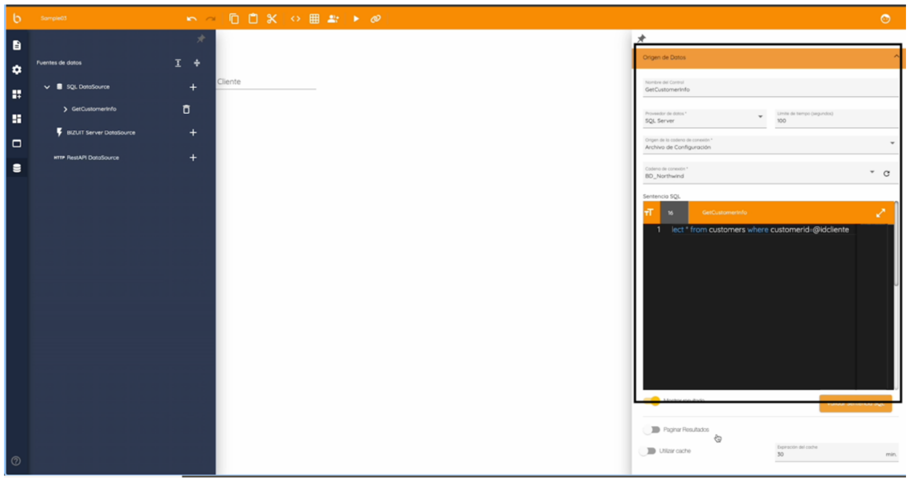
We created a text control where the user can enter a customer ID. Then, we add a secondary SQL data source, which we'll call GetCustomerInfo.
From the "Data Sources" sidebar, we select SQL and configure the connection, either by uploading a file or manually typing the connection string.

2. SQL Query

We write a query with parameters, for example:

SELECT * FROM Customers WHERE CustomerId = @IdCliente

BIZUIT will automatically recognize the parameters and output structure of the query.



Nombre del Control: GetCustomerInfo
Proveedor de datos *: SQL Server
Límite de tiempo (segundos): 100
Origen de la cadena de conexión *: Archivo de Configuración
Cadena de conexión *: BD_Northwind
Sentencia SQL

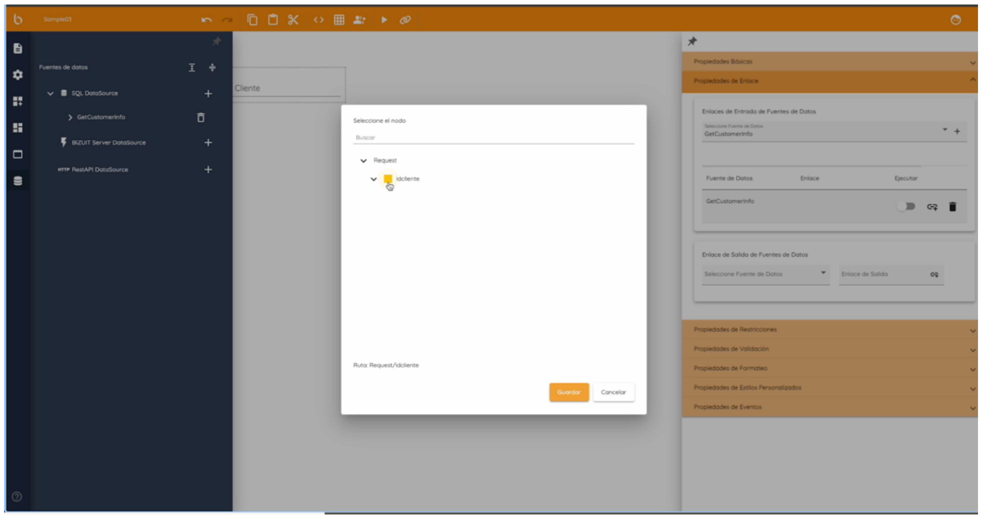
```
lect * from customers where customerid=@idcliente
```

Expiración del cache: 30 min.

3. Binding the control to the parameter

In the input field properties:

- We use the link icon to associate the @IdCliente parameter with the control value.
- We instruct the data source to run when you change the value of the field.



4. Visualization of the results

We drag the response fields into the form. For example:

- If we drag CompanyName, a field bound to that value will be automatically created.
- We can also use existing controls and manually bind them to any returned field, such as ContactName.

Summary

In this section, we learned how to use the SQL secondary data sources in BIZUIT to interact with external databases directly from forms. We explore how these allow information to be displayed in real time without the need to model it in the processes through parameters or variables.

Summary of what was seen:

1. Obtain Customer Information: We set up a SQL source to query data based on a user-entered identifier and display the results automatically on the screen.
2. Get a List of Results: We created a SQL feed to load and visualize records in an interactive table with pagination.
3. Optimization: We enabled the use of cache to improve performance in cases of data that changes little.

With these configurations, we can now dynamically and efficiently integrate external data into BIZUIT, optimizing both the user experience and the performance of our forms.

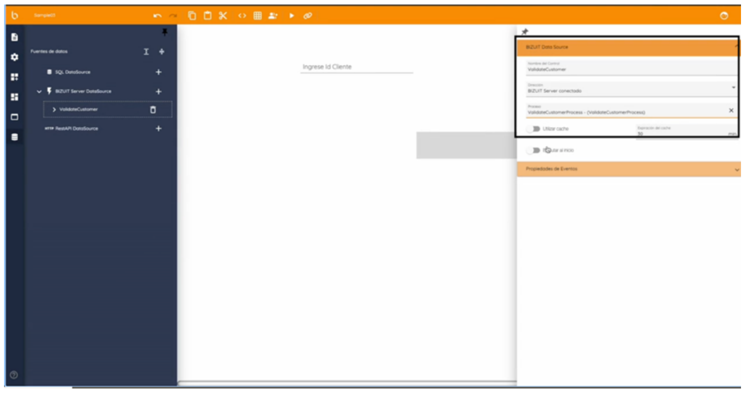
BIZUIT Secondary Data Source

In BIZUIT, secondary data sources of type BIZUIT allow us to interact directly with native processes of the platform from the forms. With them, we can validate information, query data, or execute operations on instances without relying on external integrations. Unlike other font types, these are optimized to work directly with the local or remote BIZUIT server, offering a fully integrated experience.

Case Study: Customer Validation

1. Initial setup

We create a form where the user enters a Customer ID and add a secondary BIZUIT data source called ValidateCustomer.



Nombre del Control: ValidateCustomer

Dirección: BIZUIT Server conectado

Proceso: ValidateCustomerProcess - (ValidateCustomerProcess)

Expiración del cache: 30

2. Server and Process Selection

The source can be connected:

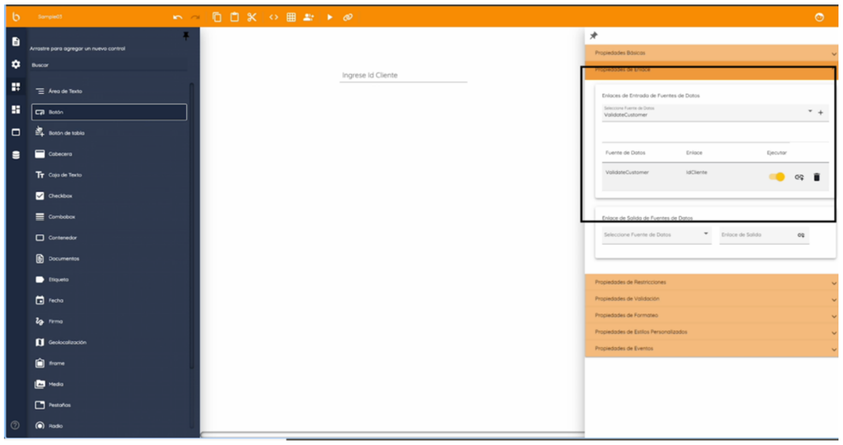
- To the local BIZUIT server.
- To a remote server (requires URL and API credentials).

In this example we use the current server and select the ValidateCustomerProcess process, which process receives the CustomerId parameter and returns the output parameter called "Enabled" along with the execution InstanceId.

3. Control Link

On the form:

- We associate the input field with the CustomerID parameter.
- We set the font to run when changing the value.



Enlaces de Entrada de Fuentes de Datos

Seleccione Fuente de Datos: ValidateCustomer

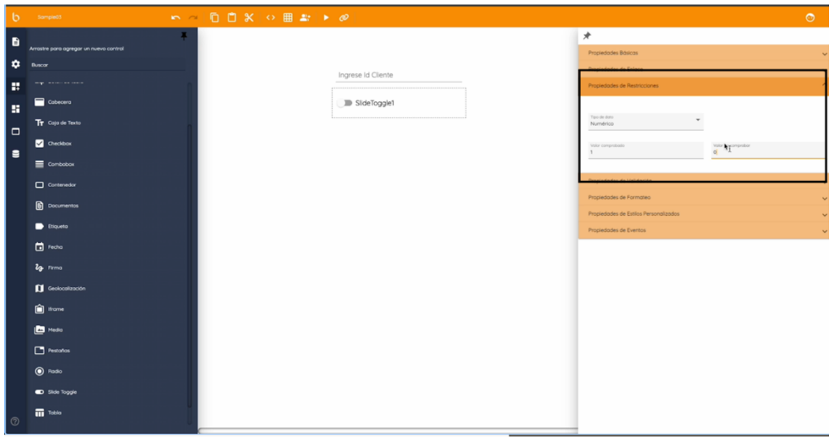
Fuente de Datos	Enlace	Ejecutar
ValidateCustomer	IdCliente	●

Enlace de Salida de Fuentes de Datos

Seleccione Fuente de Datos | Enlace de Salida

4. Visualization of the result

We bind the Enabled field to a SlideToggle control and set it to interpret 1 as enabled and 0 as disabled.



Tipo de dato: Numérico

Valor comprobado: 1

Valor no comprobado: 0

5. Execution in a secure context

In order for the process to run, the form must be within a security context:

1. In BIZUIT Dashboard, we open a page with the Independent Forms module.
2. We set up that module to use our form.

Now, when you enter different customer IDs, the form executes the process, automatically validates and displays the result in the SlideToggle.

Optimization with Cache

If the validation doesn't change frequently, we turn on the cache option. This prevents repetitive executions, improves performance, and reduces server load.

Summary

With BIZUIT data sources we can:

- Run native processes from a form.
- Get real-time results.
- Optimize performance with cache.

This functionality allows us to integrate business logic directly into forms, achieving more agile processes and a better user experience.

RestAPI Secondary Data Source

In BIZUIT, secondary data sources of type RestAPI allow you to connect forms with external web services, either public or private.

With them we can send HTTP requests (GET, POST, PUT, DELETE) to obtain, update or delete data in real time and enrich the user experience.

Case Study: Querying a Public API and Displaying Data

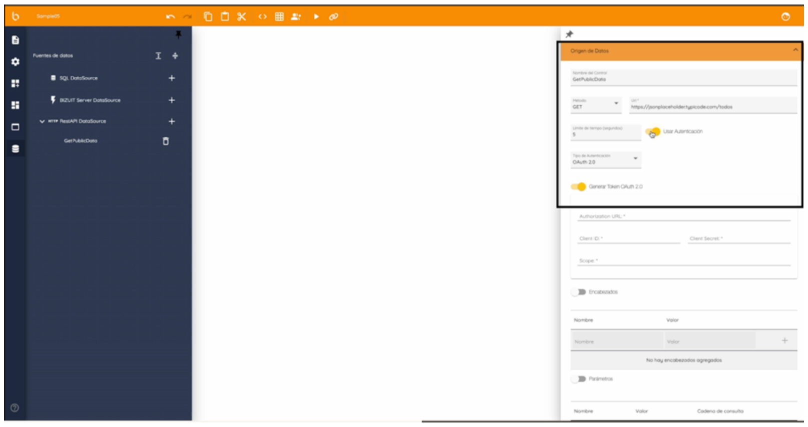
1. Initial setup

We create a form that will query information from a public API and add a secondary RestAPI data source called GetPublicData.

2. Font Settings

- Select the HTTP **GET** method and enter the URL.
- If the API requires authentication:

- **Basic Auth:** enter username and password.
- **OAuth 2.0:** Use an existing token or generate it on the spot.



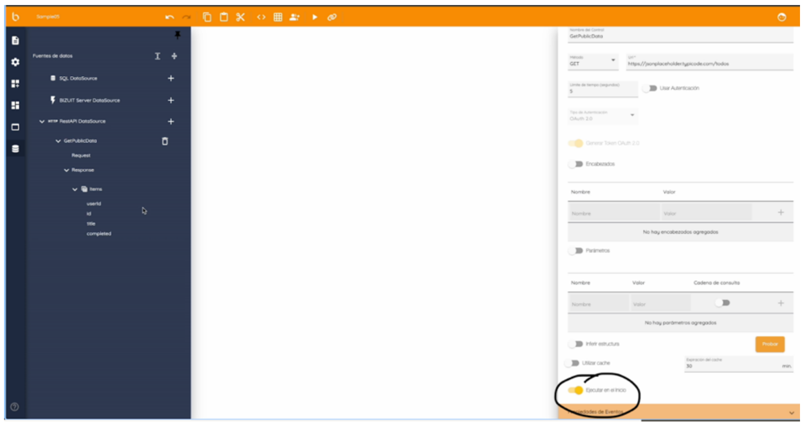
Nombre del Control: GetPublicData
Método: GET
Url *: https://jsonplaceholder.typicode.com/todos
Límite de tiempo (segundos): 5
Usar Autenticación
Tipo de Autenticación: OAuth 2.0
Generar Token OAuth 2.0
Authorization URL: *
Client ID: *
Client Secret: *
Scope: *
Encabezados

Nombre	Valor
No hay encabezados agregados	

Parámetros

Nombre	Valor	Cadena de consulta

If necessary, we add HTTP headers.
Then, we set the feed to run automatically when you start the form (no input parameters required).



Nombre del Control: GetPublicData
Método: GET
Url *: https://jsonplaceholder.typicode.com/todos
Límite de tiempo (segundos): 5
Usar Autenticación
Tipo de Autenticación: OAuth 2.0
Generar Token OAuth 2.0
Encabezados

Nombre	Valor
No hay encabezados agregados	

Parámetros

Nombre	Valor	Cadena de consulta
No hay parámetros agregados		

Inferir estructura
Utilizar cache
Expiración del cache: 30 min.
Ejecutar en el inicio

3. Response testing and analysis

- We run a test to confirm that the API responds correctly.
- We review the structure of the response to identify the fields that we will display on the form.

4. Table Display

- We drag the desired fields to a table in the form.
- We customize the columns and display format.

5. Form execution

When you start the form, the table is automatically populated with the data obtained from the API.

Optimization with Cache

If the information doesn't change frequently, we enable caching to prevent unnecessary queries, improve speed, and reduce the load on the external API.

Summary

With RestAPI fonts we can:

- Integrate external data in real time.
- Display information in pivot tables.
- Optimize performance with cache.

This functionality expands the possibilities of integration in BIZUIT, allowing us to build forms connected to any web service.

Conclusion

In this unit we delve into the use of secondary data sources in BIZUIT, understanding how to connect them to external databases, native platform processes, and web services, to enrich forms with dynamic information in real time.

We learned to work with SQL sources, configuring parameterized queries that allow us to obtain, validate and display external data directly in the form. We also explore BIZUIT fonts, which give us the possibility to execute internal processes and return results in a secure and optimized way. Finally, we analyze the RestAPI fonts, which expand the possibilities of integration by connecting our forms with any web service through HTTP requests.

In all cases, we review how to optimize performance using caching to reduce the load on systems, as well as best practices for linking data, handling parameters, and displaying results across different controls.

With these capabilities, we now have the tools to create intelligent, integrated, and highly efficient forms, capable of interacting with multiple data sources and responding in an agile way to business needs.

Unit 4: SubForms

In this unit we will explore in depth the use of **subforms** in BIZUIT Forms Designer, a key functionality to modularize and optimize the design of complex forms. Subforms allow you to divide an extensive flow into independent, reusable and easier to maintain sections, without losing consistency with the main process.

We'll discuss how to create, link, and configure subforms so that they interact seamlessly with the parent form, exchanging data using parameters and variables. We'll also review common use cases, such as nested forms to capture detailed information, modal windows to populate supplemental data, and modular structures that make it easy to scale projects.

The Subforms functionality in BIZUIT allows you to manage dynamic sections within a parent form.
They are ideal for handling related data, breaking down complex processes into smaller parts, and optimizing the user experience.

With them we can:

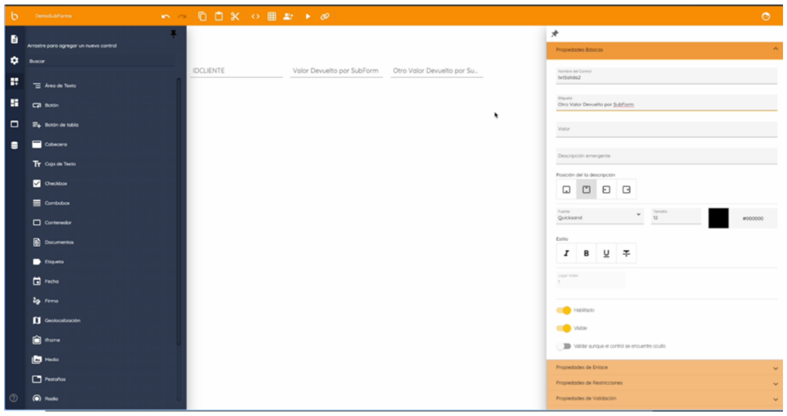
- Display or capture related information without reloading the main form.
- Control input and output parameters to maintain data consistency.
- Customize layout, buttons, and events to suit each flow.

At the end of this unit, you will have the ability to implement subforms that improve the user experience, reduce maintenance complexity, and boost efficiency in data management within your BIZUIT processes.

Practical example: Subform connected to a caller form

1. Creating the caller form
On the main form:

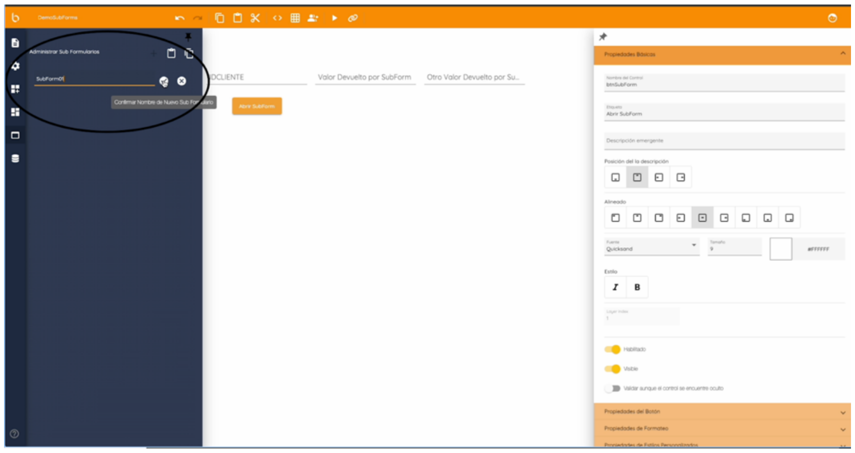
- We added three text boxes:
    - txtClientId(tag: ClientID): Value to be sent to the subform.
    - txtOutput1 and txtOutput2: Will display values returned by the subform.
- We added a btnSubForm button (tag: Open SubForm) to open the subform.



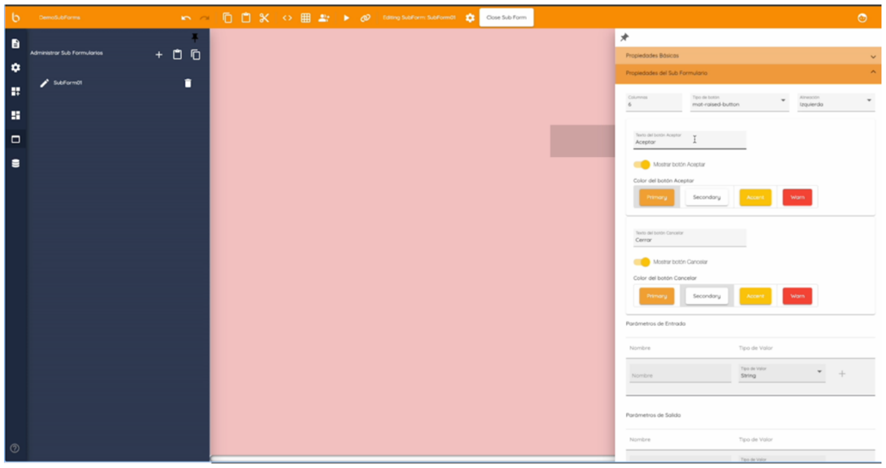
2. Creating the subform

From the Subforms option:

1. We create a new Subform called SubForm01.



2. We edit your settings:
    - We changed the background color to distinguish it from the main form.
    - We activate or customize Accept and Cancel buttons.


DemoSubForms
Editing SubForm: SubForm01
Close Sub Form
Administrar Sub Formularios
SubForm01
Propiedades Básicas
Propiedades del Sub Formulario
Primary
Secondary
Accent
Warn

Columnas: 6
Tipo de botón: mat-raised-button
Alineación: Izquierda
Texto del botón Aceptar: Aceptar
Mostrar botón Aceptar
Color del botón Aceptar: Primary
Texto del botón Cancelar: Cerrar
Mostrar botón Cancelar
Color del botón Cancelar: Secondary

Parámetros de Entrada

Nombre	Tipo de Valor
Nombre	Tipo de Valor: String

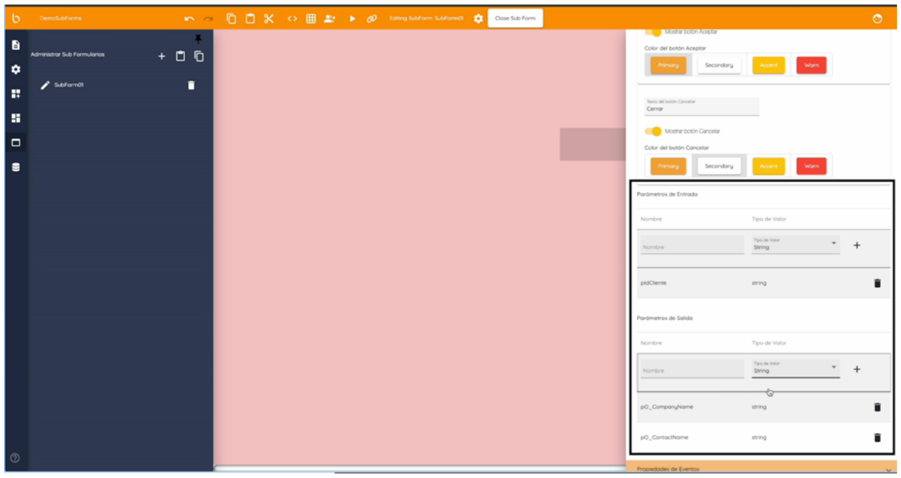
Parámetros de Salida

Nombre	Tipo de Valor

3. Parameter Configuration

In Subform Properties:

- **Input Parameters**:
  - pIdClient (String): Receives the value from the caller form.
- **Output Parameters**:
  - pO_CompanyName (String)
  - pO_ContactName (String)



Texto del botón Cancelar: Cerrar
Mostrar botón Cancelar
Color del botón Cancelar: Secondary

Parámetros de Entrada

Nombre	Tipo de Valor
Nombre	Tipo de Valor: String
pIdCliente	string

Parámetros de Salida

Nombre	Tipo de Valor
Nombre	Tipo de Valor: String
pO_CompanyName	string
pO_ContactName	string

4. Controls within the subform

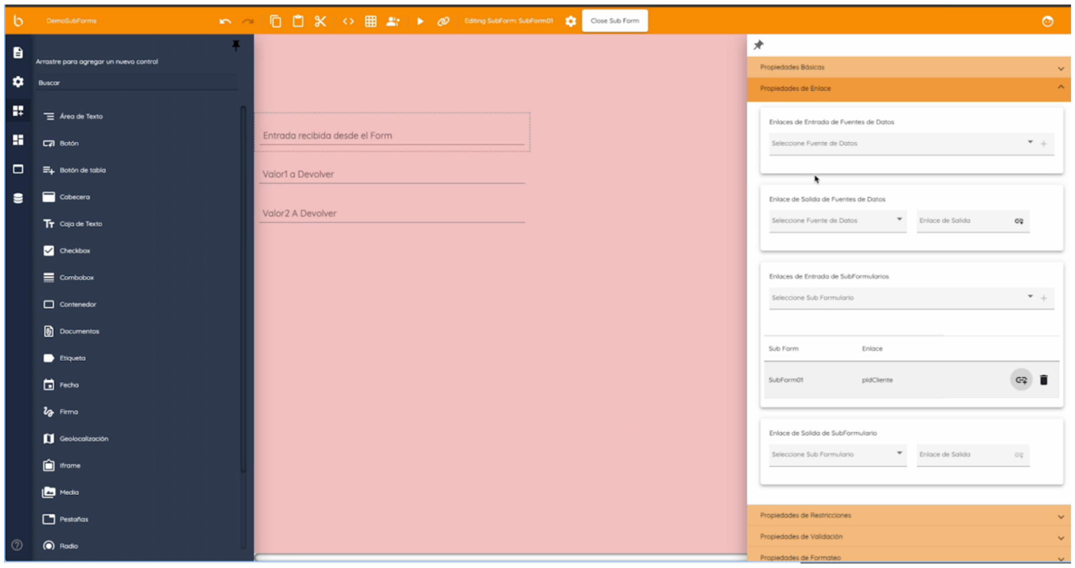
- txtInput: Bound to pIdClient (disabled so as not to modify it manually).

- txtOutput1 and txtOutput2: bound to pO_CompanyName and pO_ContactName respectively.

5. Integration with the main form

On the btnSubForm button of the caller form:

1. Event: Invoke Subform → select SubForm01.
2. Request Mapper: map txtIdClient → pIdClient.
3. Response Mapper: Map pO_CompanyName → txtOutput1 and pO_ContactName → txtOutput2.
4. Disable txtOutput1 and txtOutput2 to avoid manual modifications.


DemoSubForms
Editing SubForm: SubForm01
Close Sub Form
Arrastre para agregar un nuevo control
Buscar
Área de Texto
Botón
Botón de tabla
Cabecera
Caja de Texto
Checkbox
Combobox
Contenedor
Documentos
Etiqueta
Fecha
Firma
Geolocalización
Iframe
Media
Pestañas
Radio
Entrada recibida desde el Form
Valor1 a Devolver
Valor2 A Devolver
Propiedades Básicas
Propiedades de Enlace
Enlaces de Entrada de Fuentes de Datos
Seleccione Fuente de Datos
Enlace de Salida de Fuentes de Datos
Seleccione Fuente de Datos
Enlace de Salida
Enlaces de Entrada de SubFormularios
Seleccione Sub Formulario
Enlace de Salida de SubFormulario
Seleccione Sub Formulario
Enlace de Salida
Propiedades de Restricciones
Propiedades de Validación
Propiedades de Formateo

Sub Form	Enlace
SubForm01	pIdCliente

6. Execution

- Enter a value in Client ID and open the subform.
- Complete Output Values and Accept: The fields in the main form are automatically populated.
- If canceled, the values are not updated.

Proceeds

- **Centralization**: Groups specific logics into a separate component.
- **Dynamic Interaction**: Updates the main form based on the activity on the subform.

- **Design optimization**: improves usability by avoiding overloading the main form.
- **Fewer errors**: isolates complex logics.
- **Visual consistency**: Consistent styles across all subforms.
- **Scalability**: Easy to add new sections without modifying the entire form.
- **External integration**: Supports SQL, BIZUIT, or RestAPI.
- **Performance**: Reduces the initial load of the main form.
- **Security**: Control access to sensitive data using parameters.
- **Step-by-step flows**: ideal for processes that require defined steps.

Use Cases

- **Order Details**: Capture items and prices without overloading the main form.
- **Commits**: Validate critical actions before executing them.
- **Real-time validations**: verify data before processing.
- **Approval** flows: Manage approvals or rejections.
- **Notes and observations**: record related comments.
- **User preferences**: Personalization without interrupting the main flow.
- **Master-detail**: Handling relationships such as invoices and detail lines.
- **Temporal data**: capture relevant information without saving it in the main database.
- **Secondary Reports**: Display related reports or charts.
- **Simulations**: Show potential results before confirming actions.

Conclusion

Throughout this unit, we delve into the use of subforms within BIZUIT Forms Designer as a key solution for structuring and optimizing complex forms. We learned how to create, configure, and link them to the main form, ensuring seamless data exchange using parameters, variables, and events.

We saw how subforms can improve the modularity of the design, allowing components to be reused in different contexts, and how to adapt them to different needs through validations, visibility rules and dynamic updating of information.

We also explore specific use cases, such as the addition of nested or modal forms, which make it easy to capture supplemental information without overloading the main interface.

With these tools, we now have the ability to design forms that are more orderly, scalable, and easier to maintain, enhancing both the user experience and the operational efficiency of our processes.

Chapter Summary

In this chapter, we go through, step by step, the main tools and techniques for designing powerful, customized, and process-aligned forms in BIZUIT Forms Designer, ranging from exploring the work environment to implementing modular and reusable structures.

In Unit 1, we become familiar with the designer's environment, understanding the function of each of its areas: the side menu with its creation, editing and management options; the design area to visually structure the forms; the toolbar with editing, testing, and collaboration functions; and the property sheet to configure both general parameters and specific details of each control.

Unit 2 took us on a full tour of the controls available at BIZUIT. We learned how to configure, customize, and combine them to capture, display, and manage data efficiently. We review basic and advanced properties, validations, data bindings, dynamic formatting, custom styles, and events, addressing everything from simple elements like text boxes to complex components like tables, tabs, and media controls.

In Unit 3, we focus on secondary data sources, which allow external information and internal logic to be integrated in real time within forms. We work with three types: SQL: to interact with external databases through parameterized queries, BIZUIT: to execute native processes and obtain integrated results, and RestAPI: to connect with web services and display dynamic data.
In all cases, we incorporate optimization techniques with caching and parameter binding to improve performance and user experience.

Finally, in Unit 4 we explore subforms as a key tool for modularizing designs, dividing long forms into independent and reusable sections. We learned how to create, link and configure their two-way communication with the main form, applying validations, visibility rules and automatic data updating, which allows us to build more orderly, scalable and easy-to-maintain interfaces.

With what we have seen in this chapter, we have a solid set of resources to create professional forms, integrated and adapted to any business scenario, maximizing flexibility, scalability and user experience in BIZUIT.