Forms v2 User Manual For GFV2

Version control:

Date	Description	Created by
30 Dec 2024	**Draft**	**PremiseHQ**
28 Jan 2025	**Update**	**PremiseHQ**
2 May 2025	**Update**	**PremiseHQ**

Table of contents:

Introduction:

Our advanced Forms Application is a robust and versatile platform designed to streamline data collection and enhance user interaction. Built with scalability and flexibility in mind, this application caters to diverse needs, whether for individual users, businesses, or large organizations. The feature-rich platform supports unlimited forms, submissions, and file uploads, offering a seamless experience for both form creators and respondents.

Key characteristic:

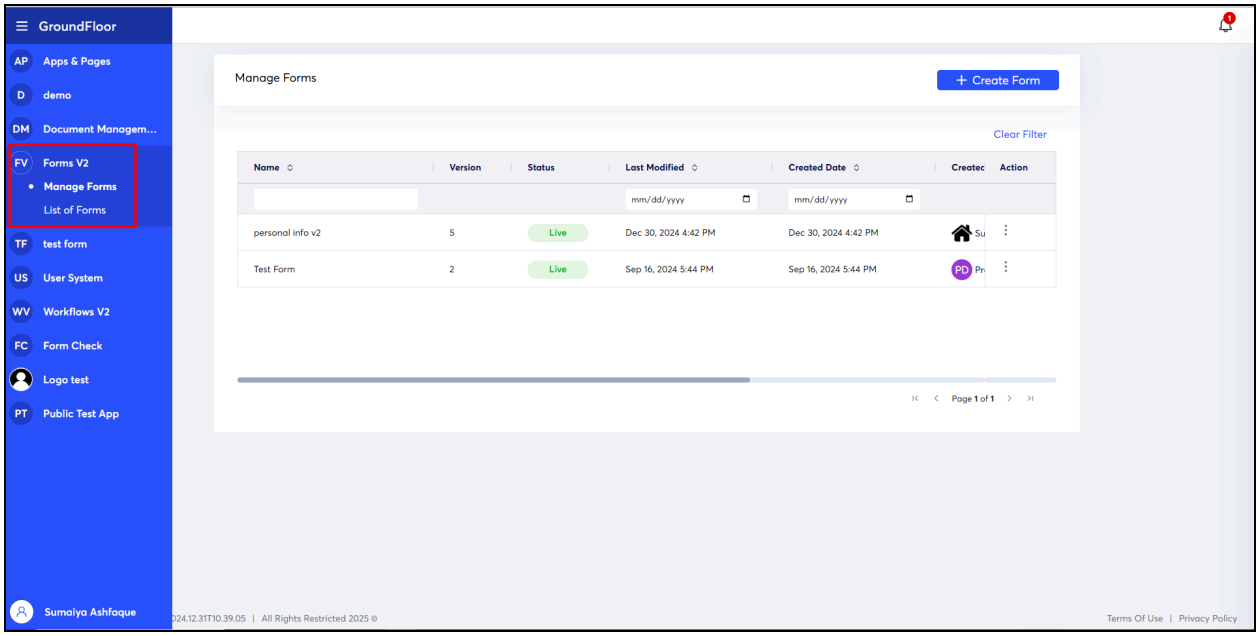
- Dynamic and Customizable Forms
- Enhanced Usability and Efficiency
- Advanced Customization Options
- Integration and Accessibility
- Navigation and Submission
- Security and Management

Application execution Diagram:

Application Access process:

1. The user has to go to the Groundfloor. URL. https://v2.groundfloor.co/login
2. Sign in with Valid credentials.
3. Click on the “Forms V2” from the Application service layer.
4. User will get 2 pages

    - Manage Forms. Users will be able to create or manage the forms from the page.
    - List of Forms. Users will be able to submit the live forms responces from the page.



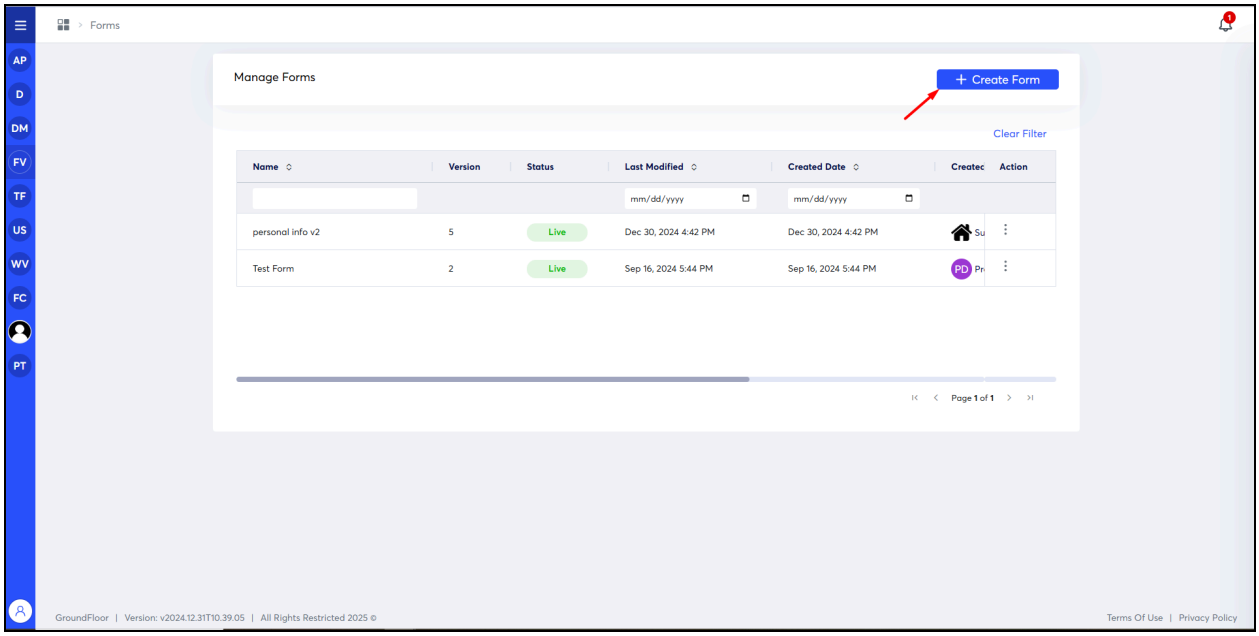
Supportive Applications:

1. Apps & Pages.
2. Document Management.
3. Data vault.
4. User System.

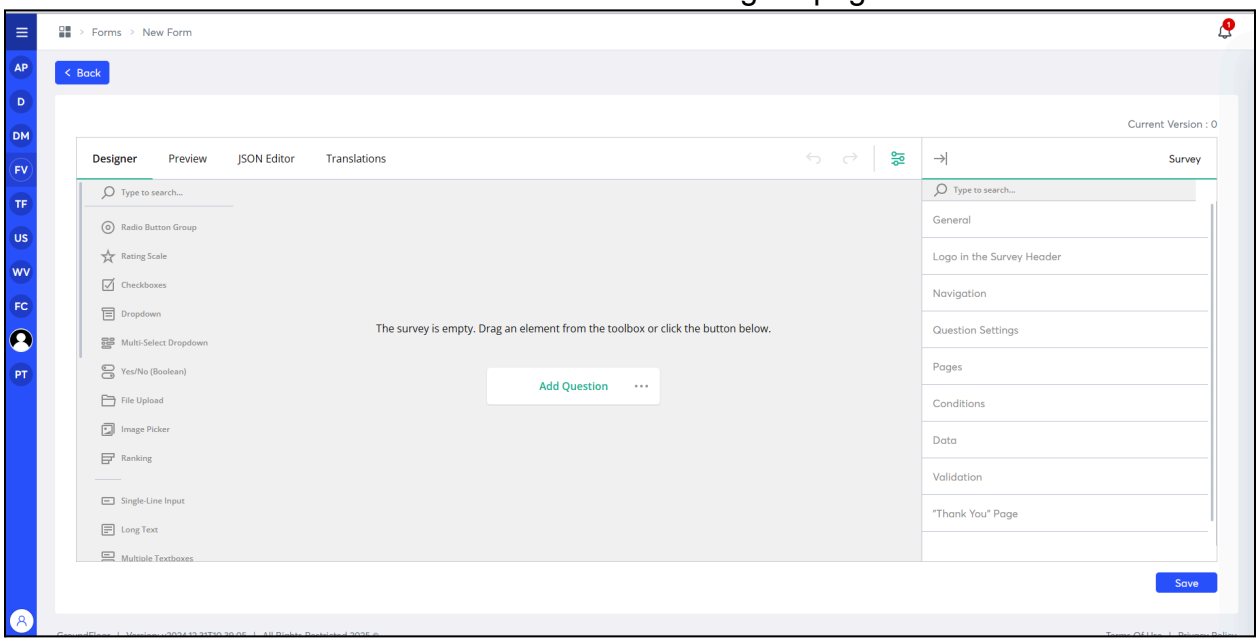
Application execution process:

1. Sign In:
    - Navigate to Groundfloor Login.
    - Sign in with valid credentials.

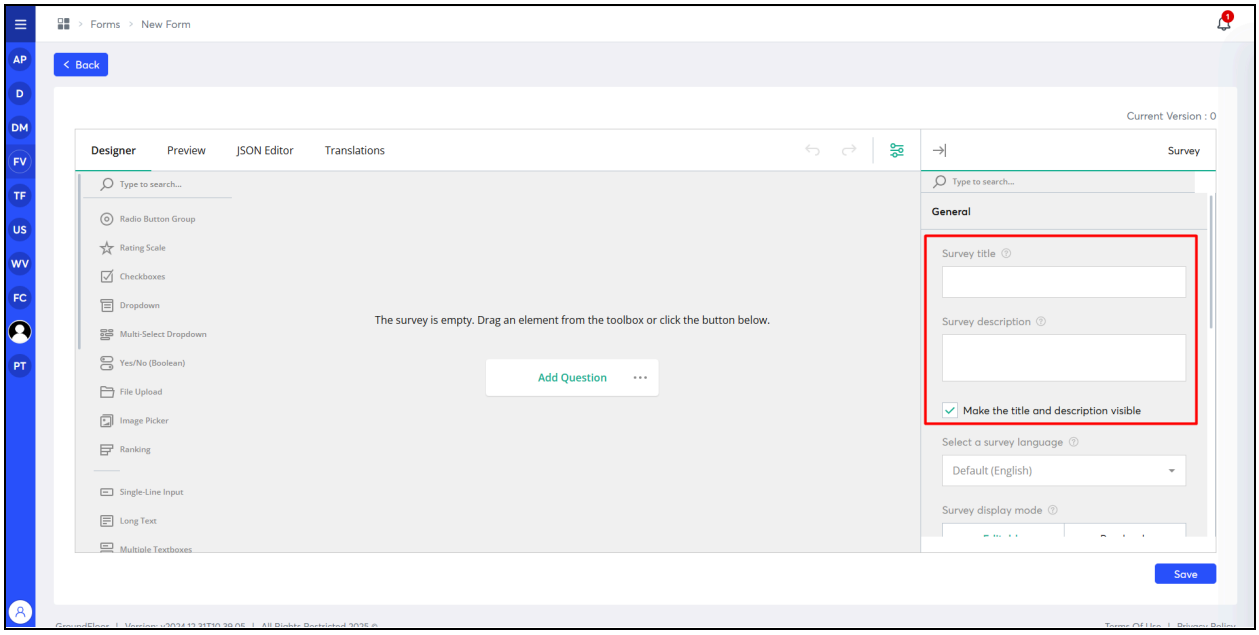
2. Create a New Form:
    - Click on "Forms V2" from the Application Service Layer.
    - Navigate to the Manage Forms page.
    - Click "+ Create Form" in the top-right corner.



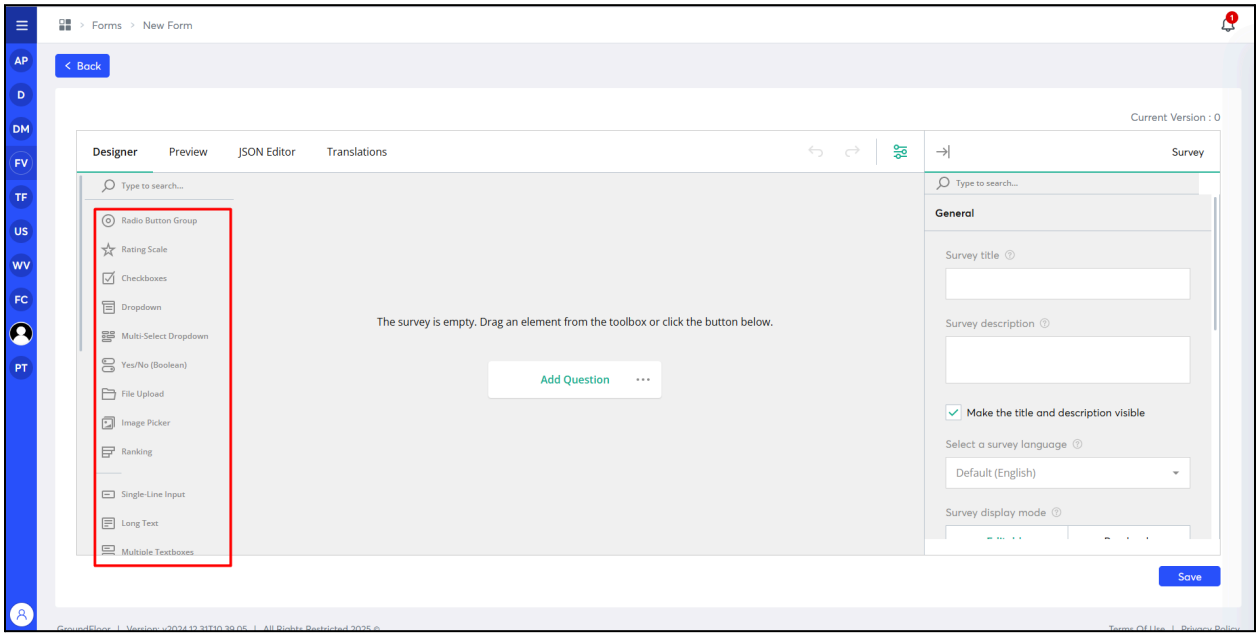
- You will be redirected to the Form Designer page.



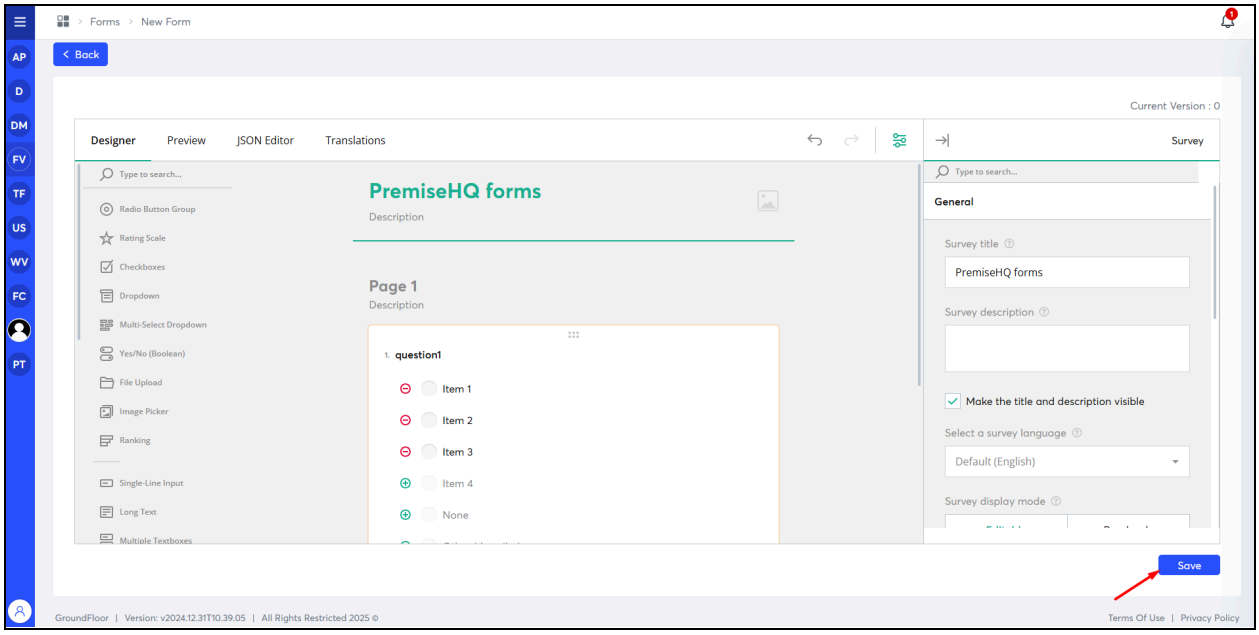
3. Design the Form:
    - Add a Form Title and Description from the General section.



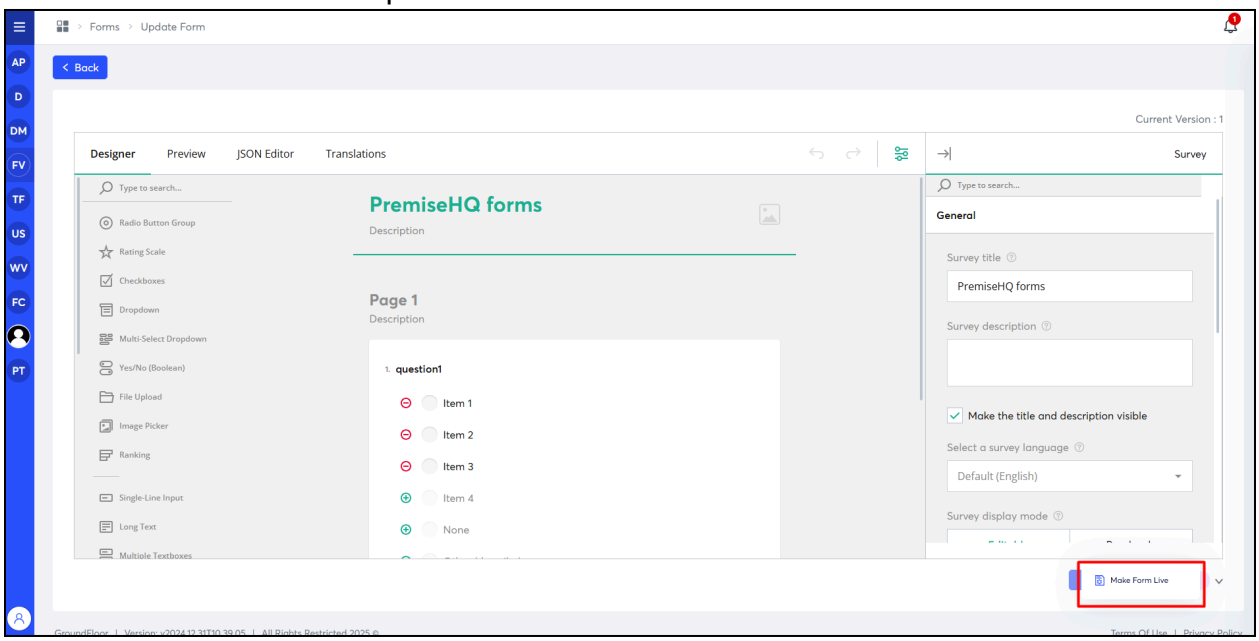
- Drag and drop the required input fields from the left side of the designer to the form canvas.



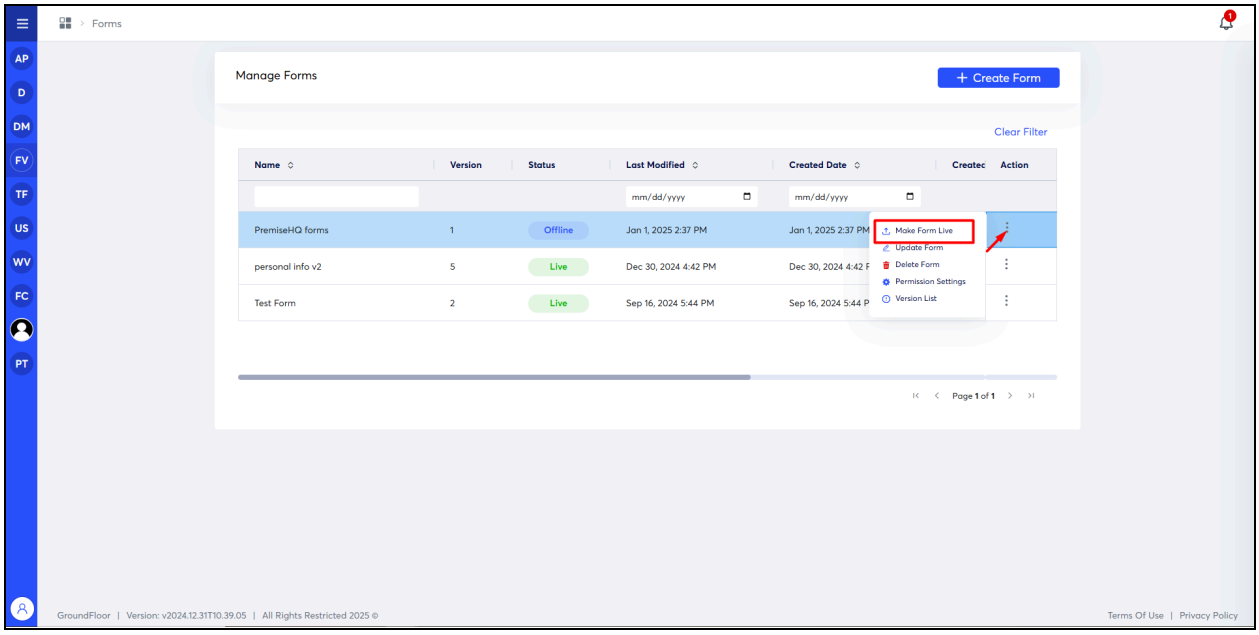
- Update the titles of the fields as needed.
- Click the Save button in the bottom-right corner to save the form.



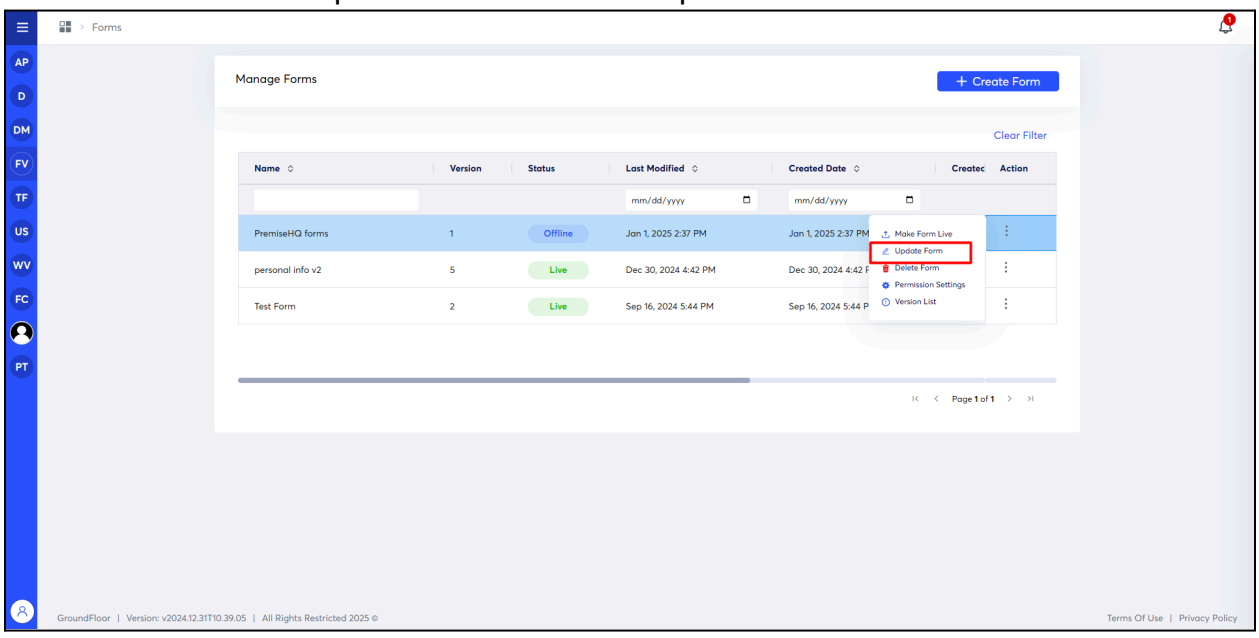
4. Make the Form Live:
    - To make the form live, change the form status from Offline to Live using the status dropdown at the bottom-left corner of the form.



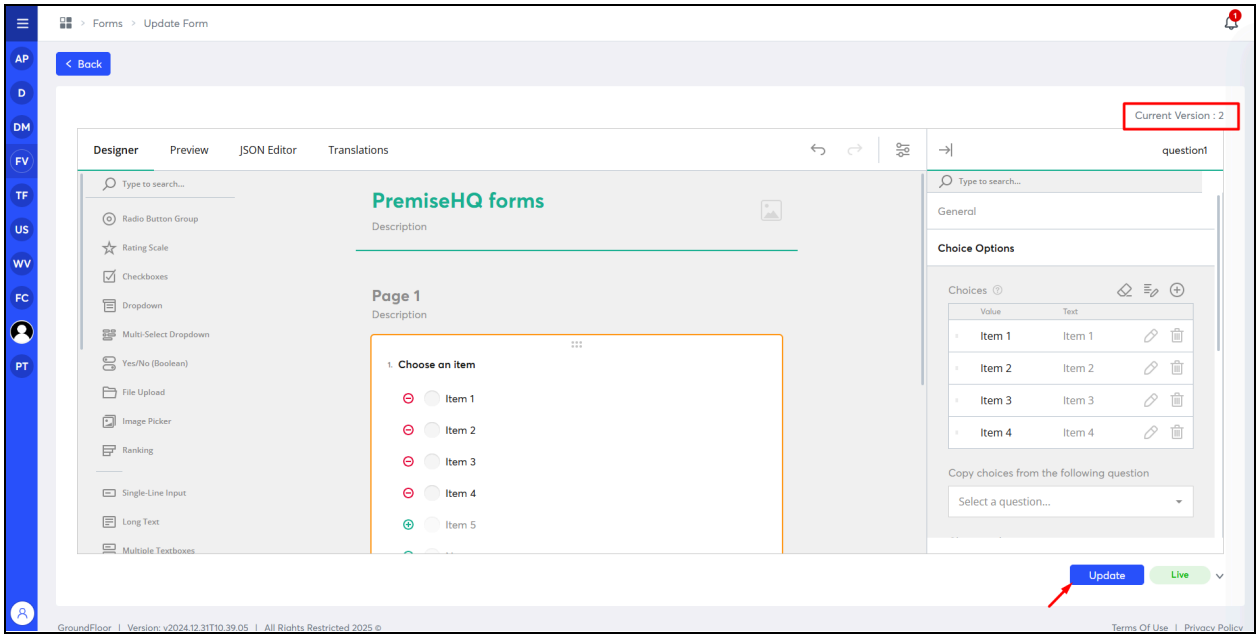
    - Alternatively, change the form status via the Manage Forms Table Action section.



5. Updating a Form:
    - Go to the Manage Forms page.
    - Select the form you want to update.
    - Click on the Action button.
    - Choose Update Form from the dropdown.

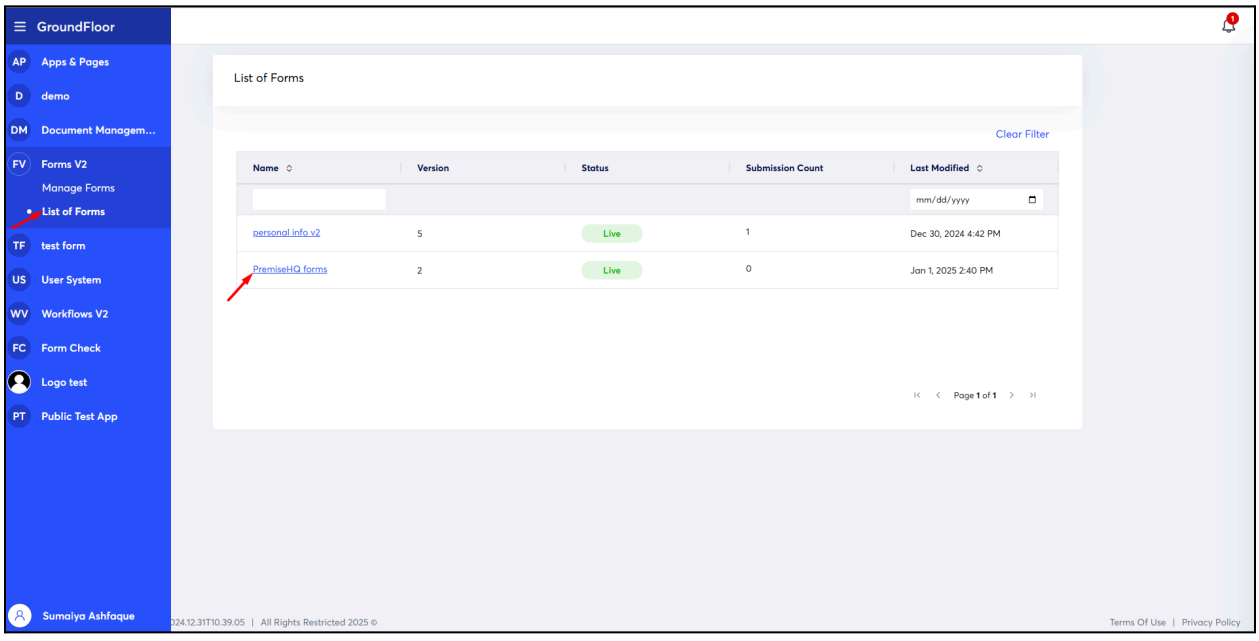


- You will be redirected to the Form Designer page.
- Make the necessary updates to the form and click Update.
- A new version of the form will be created.

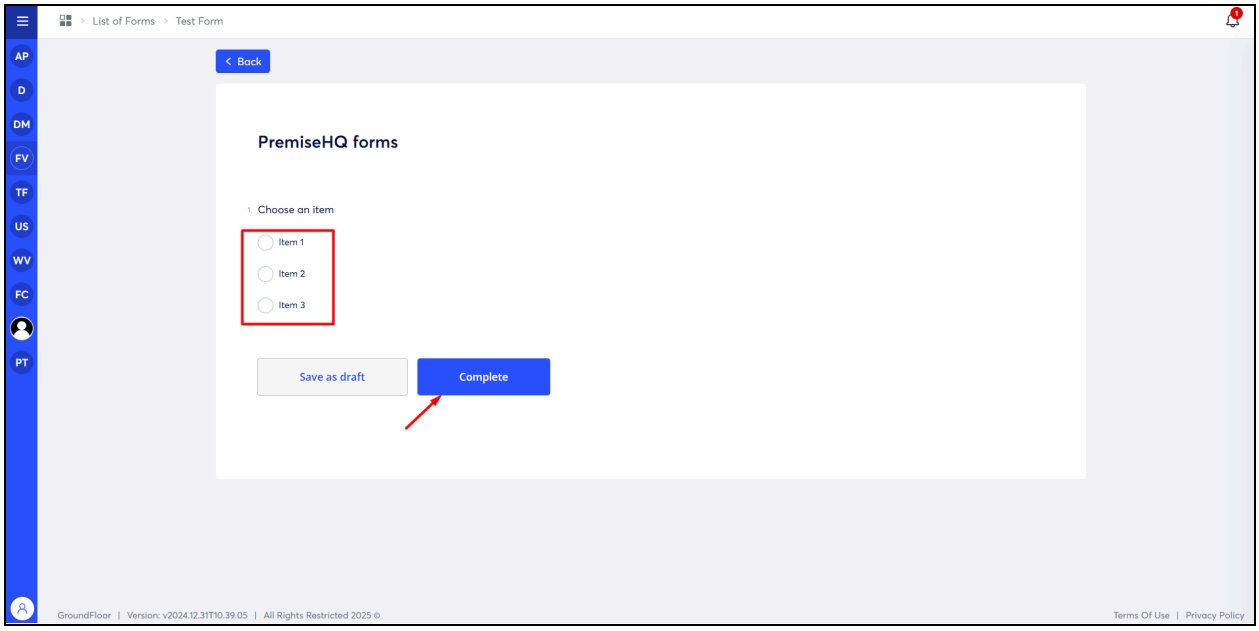


- Only users with the permission to update forms can perform this action. This permission can be managed via the Form Permissions Settings page.

6. Submitting a Form Response:
   - Navigate to the List of Forms page.
   - Locate the form and click on its Title to load it.



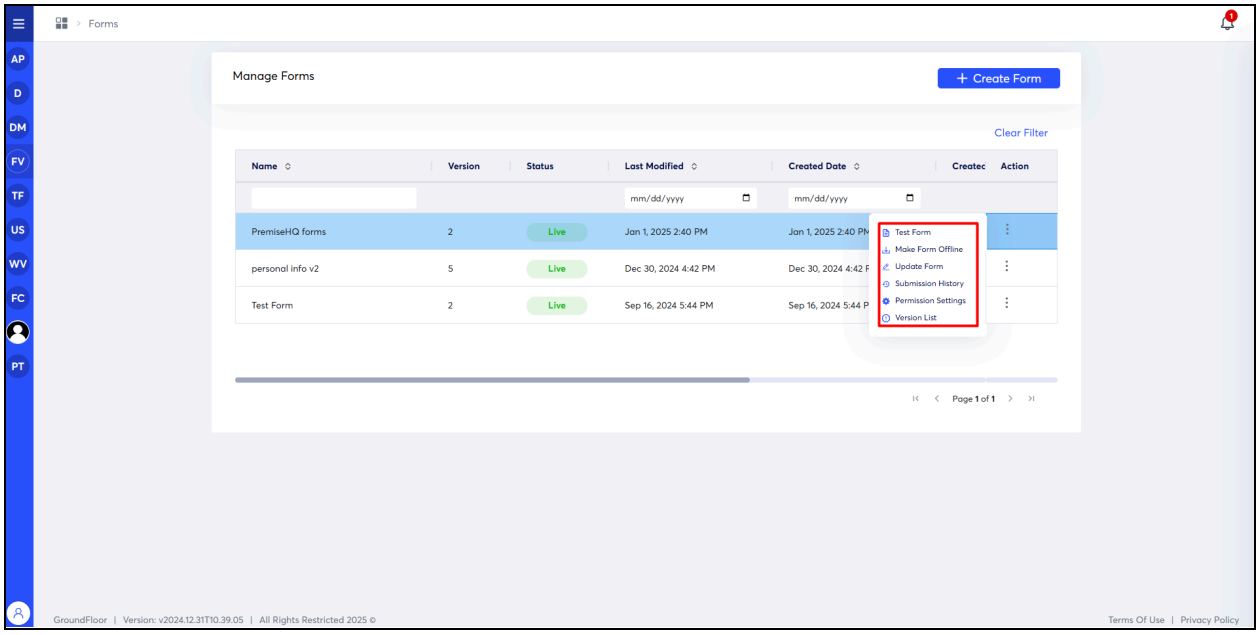
- Fill in the input fields with the required data.
- Submit the form using the Submit button.



- You can also save the response as a draft using the Save as Draft functionality.

7. Admin Functions for Form Management:
   The following functionalities are available from the Manage Forms Table Action section:

   - Testing: Test the form after creation.
   - Status Management: Change the form status (e.g., Offline to Live).
   - Editing: Update the form's details or structure.
   - Response Review: View submission history to see responses.
   - Permissions: Manage user permissions for the form.
   - Version History: View and manage the form version list.



List of Features:

- Unlimited forms, submissions and file uploads
- Dynamic JSON-driven forms
- 20+ accessible input types
- Panels for question grouping
- Matrix for question grouping
- Multi-page forms
- Calculator forms
- Duplicate group option
- Data validation
- Conditional Logic and Dynamic Texts
- Draft-save
- Text formatting
- Data aggregation
- Custom form branding
- Custom input fields
- Conditional logic
- Integration with 3rd-party libraries
- Expression language (Built-in & custom functions)
- Load choices from web services
- e-Signature field
- All popular types of form navigation

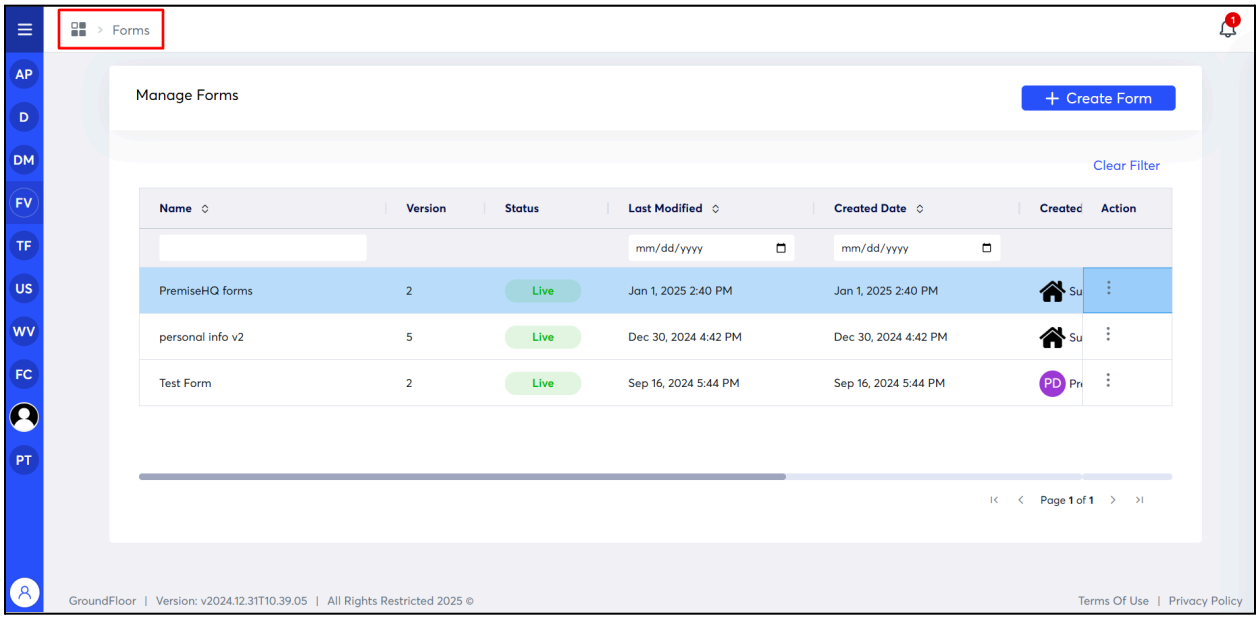
- Support for RTL languages
- Public form submission
- Forms version control
- Permission management
- Offline and online forms
- Submission history list and edit submission

Getting Started:

1. **Manage Forms Page**

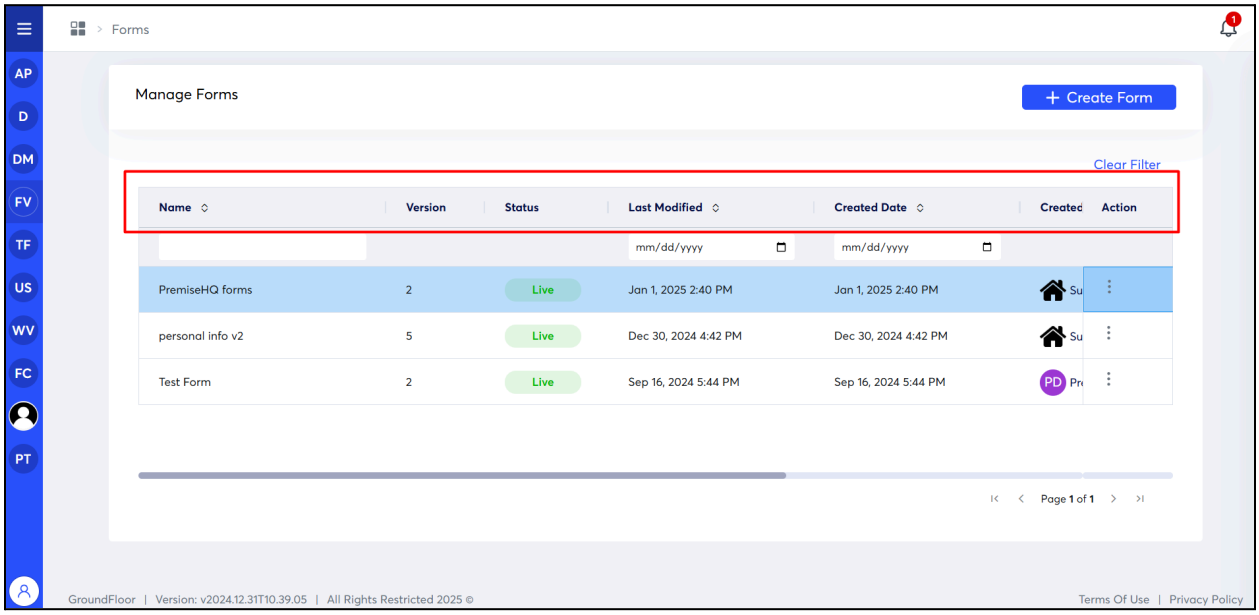
   The Manage Forms page allows users to view, manage, and take actions on forms in the system. Below is a detailed guide to understanding and utilizing the features of this page.
   Page Overview: The page is divided into key sections to help you organize and manage forms effectively:

   a. Header Section:
      - **Breadcrumb Navigation:** Displays the current location within the application (e.g., Forms).

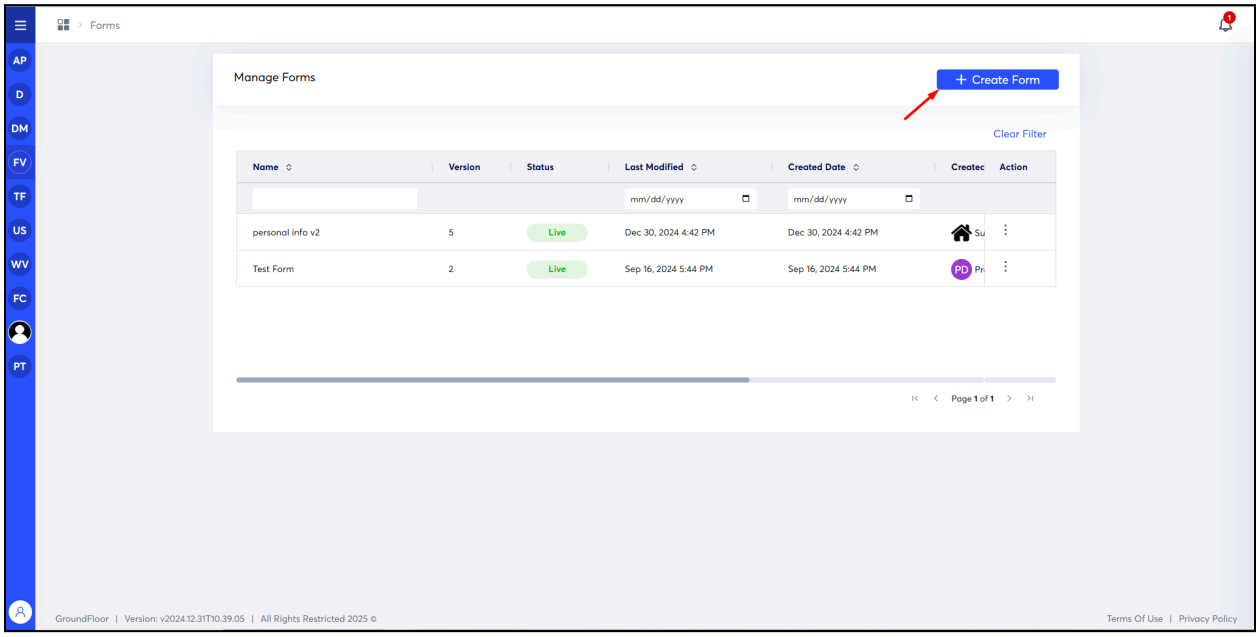


   b. Table View The table provides an overview of existing forms with the following columns:
      - **Name**: The name of the form. You can filter the list by entering keywords in the input field under the column header.
      - **Version**: The version number of the form.
      - **Status**: Displays the current status of the form (e.g., Live).
      - **Created Date**: The date and time the form was created.
      - **Created By**: The name of the user who created the form.

- **Last Modified**: The most recent date and time the form was updated.
- **Updated By**: The name of the user who last modified the form.
- **Action Menu**: Provides additional actions for managing forms.

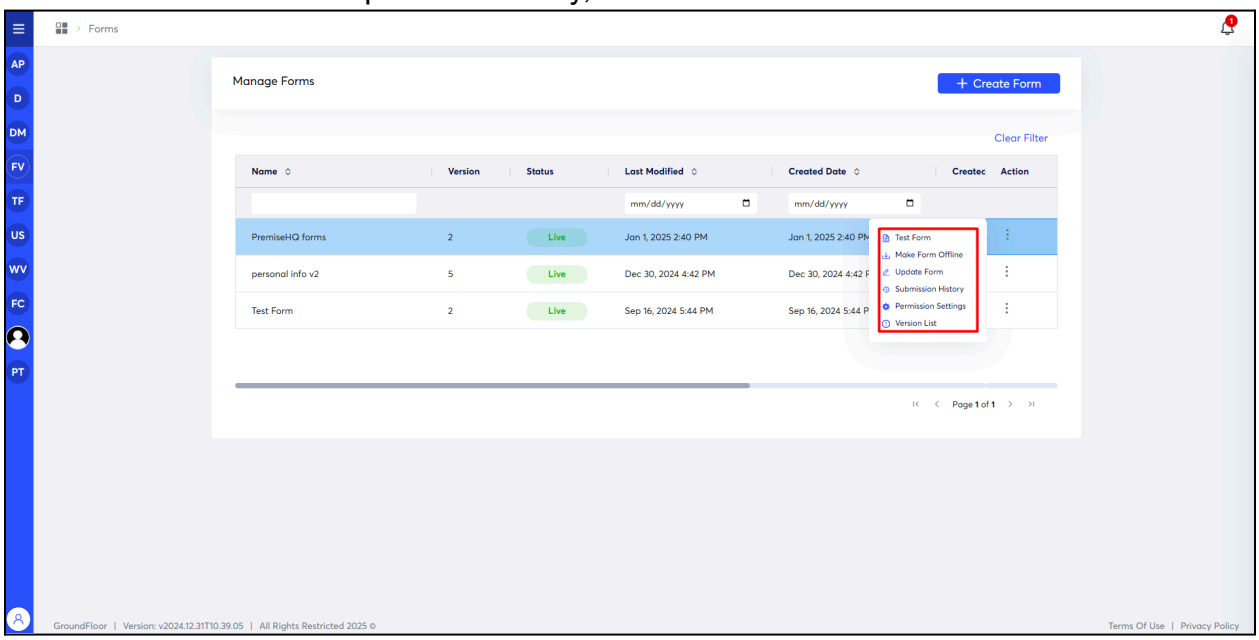


c. Create a New Form:
   - Click the **+ Create Form** button to navigate to the form creation page.



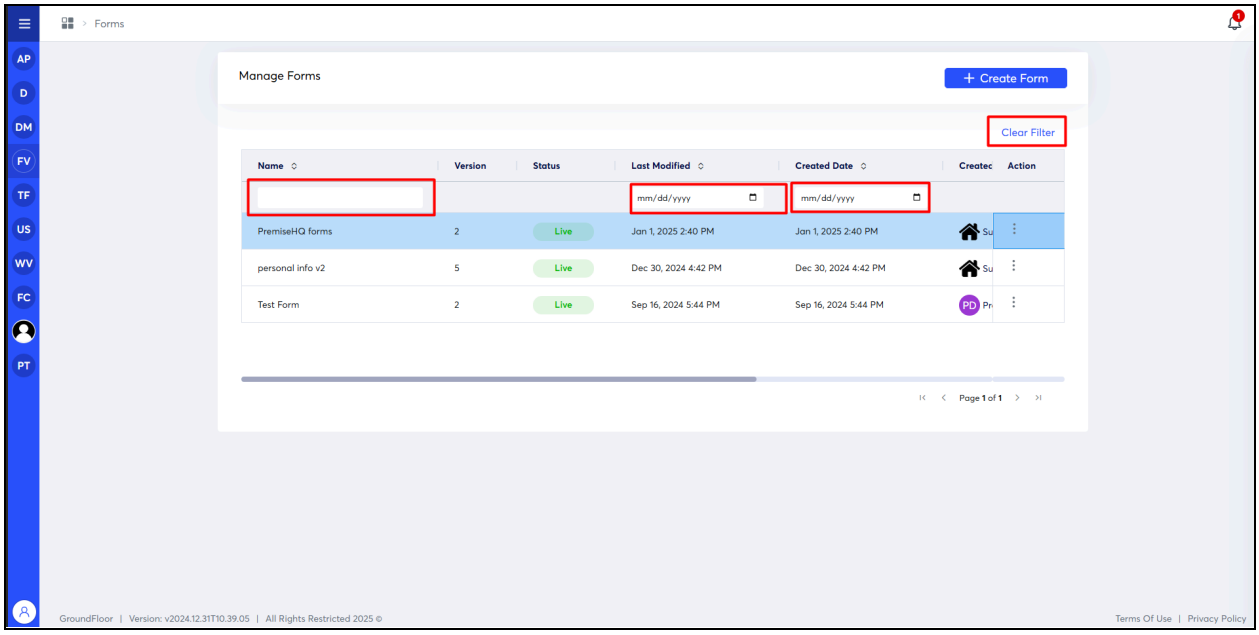
d. Action Menu The three-dot menu under the Action column provides the following options:

- **Test Form**: Opens the form in a test environment to review its functionality.
- **Make Form Offline**: Temporarily disables the form. This is available only if the form is currently in a **Live** status.
- **Make Form Live**: Activates a form that is currently offline.
- **Update Form**: Opens the form editor to make changes to the form.
- **Submission History**: Displays a log of submissions for the selected form.
- **Permission Settings**: Configures access permissions for the form.
- **Version List**: Views the history of versions for the form.
- **Delete Form**: Permanently removes the form from the system. Use this option cautiously, as deleted forms cannot be recovered.



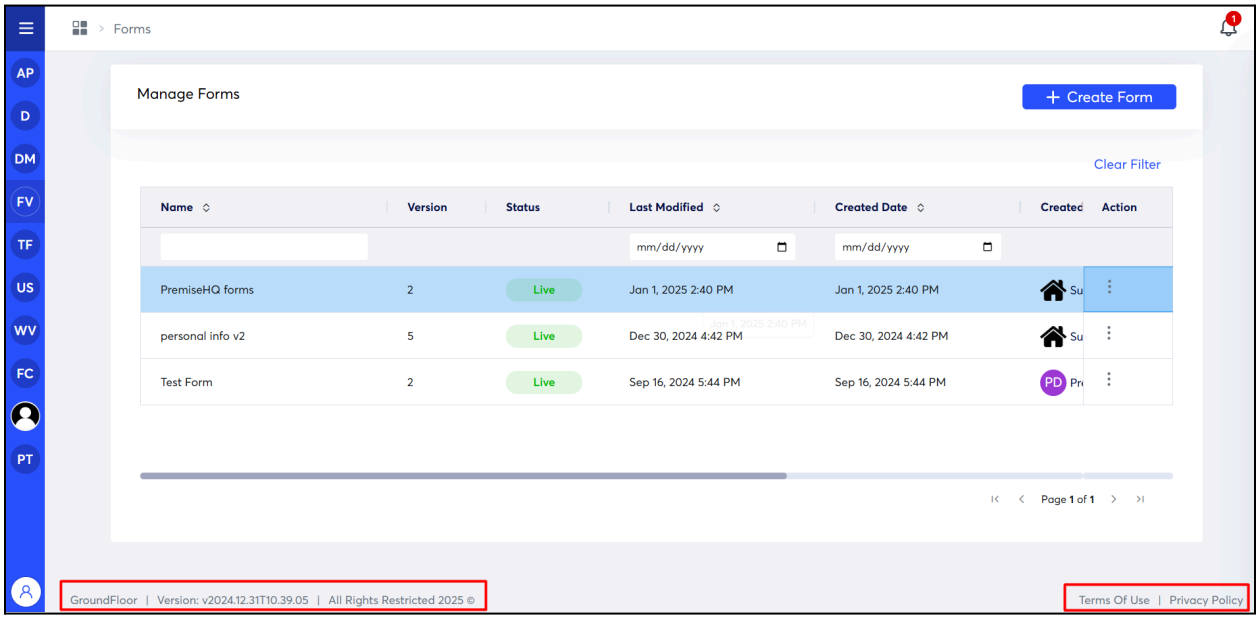
e. Filters

- **Search Field**: Allows you to search for forms by name.
- **Date Filters:** Specify date ranges for the Created Date and Last Modified fields to narrow down your search.



f. Footer Section

- Pagination controls are available to navigate through multiple pages of forms.
- Application details such as version information and links to the Terms of Use and Privacy Policy are displayed at the bottom.



g. Additional Notes

- The form statuses (Live, Offline, etc.) indicate whether the form is currently active and available for use.
- Users with appropriate permissions can access all features listed in the Action Menu.

2. **Forms General properties and settings:**
   The forms have a general settings which will be applied for all the fields and core forms, the settings option includes the following features:

   a. **General**
      - Configure basic properties for the form, such as the form title, description, and display mode.
      - Manage form-wide settings, like setting a unique form ID, choosing the form's language, and customizing styles.

   b. **Logo in the Form Header**
      - Add a logo to the top of the form.
      - Specify the logo's URL, alignment (left, center, or right), and size settings.

   c. **Navigation**
      - Customize navigation settings for the form.
      - Control buttons such as "Start," "Previous," "Next," and "Complete."
      - Enable or disable navigation options like "show progress bar" or "enable back navigation."

   d. **Question Settings**
      - Define global settings for all questions, such as their default state (e.g., required or optional).
      - Set default values, visibility logic, or other properties that apply to all questions.

   e. **Pages**
      - Manage the structure of the form by adding, deleting, or reordering pages.
      - Configure page-specific settings like page titles, descriptions, or visibility logic.

   f. **Conditions**
      - Set up conditional logic to display questions or pages based on user responses.
      - Configure "if-then" rules to create dynamic forms.

   g. **Data**
      - Define how data is collected and stored.
      - Enable features like data encryption or custom metadata fields for user tracking.

   h. **Validation**
      - Specify validation rules for user input, such as minimum and maximum lengths, numeric ranges, or regular expression checks.

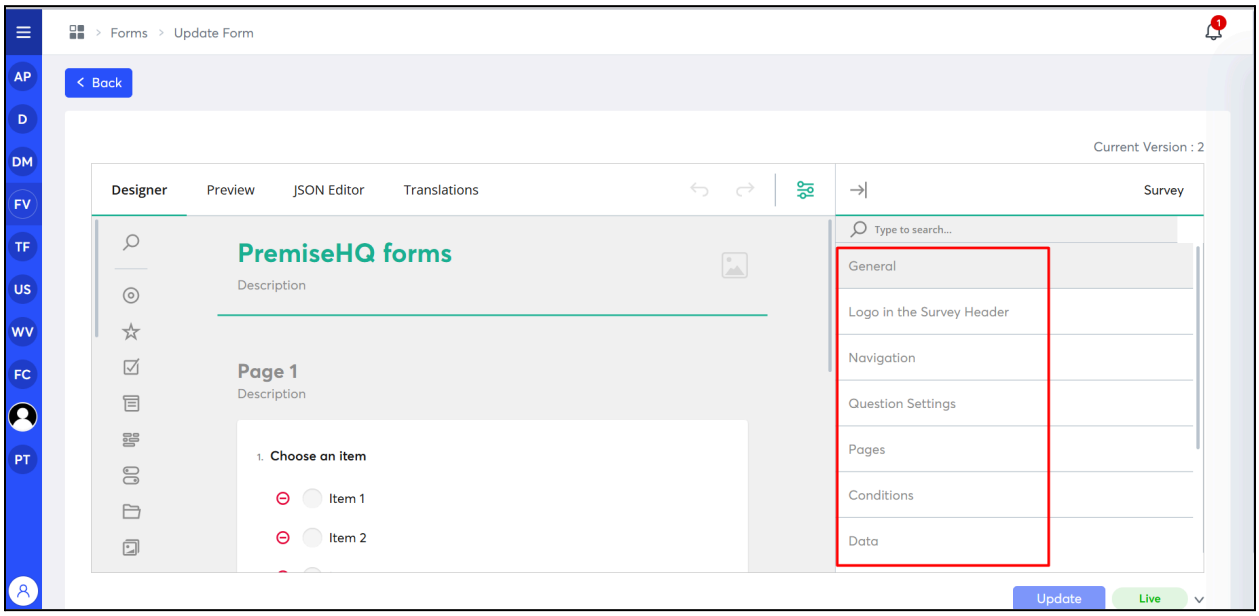
- Manage error messages and define validation triggers.

i. “Thank You” Page

- Design the final page displayed to users after form completion.
- Add a custom thank-you message, links, or call-to-action elements.

j. Quiz Mode

- Enable quiz functionality in the form to evaluate respondents’ knowledge or skills.
- Add correct answers and scores for each question to calculate total results.
- Customize result messages or feedback for different score ranges.
- Configure settings for score display, such as whether to show the respondent’s score after completion or during the quiz.



2. Pages General properties and settings:

The forms page have a general settings which will be applied for all the fields and core forms, the settings option includes the following features:

a. General

- This specifies the name or label of the page, displayed to users at the top of the page (e.g., "Page 1").
- Add a brief description or instructions for the page. This helps users understand the purpose of the questions on the page.
- Set the visibility mode and read only mode will be configured by enable the checkbox

b. Question Settings

- Configure options specific to the questions on the page. This could include settings like alignment, error message alignment, or question ordering on the page.

c. Conditions

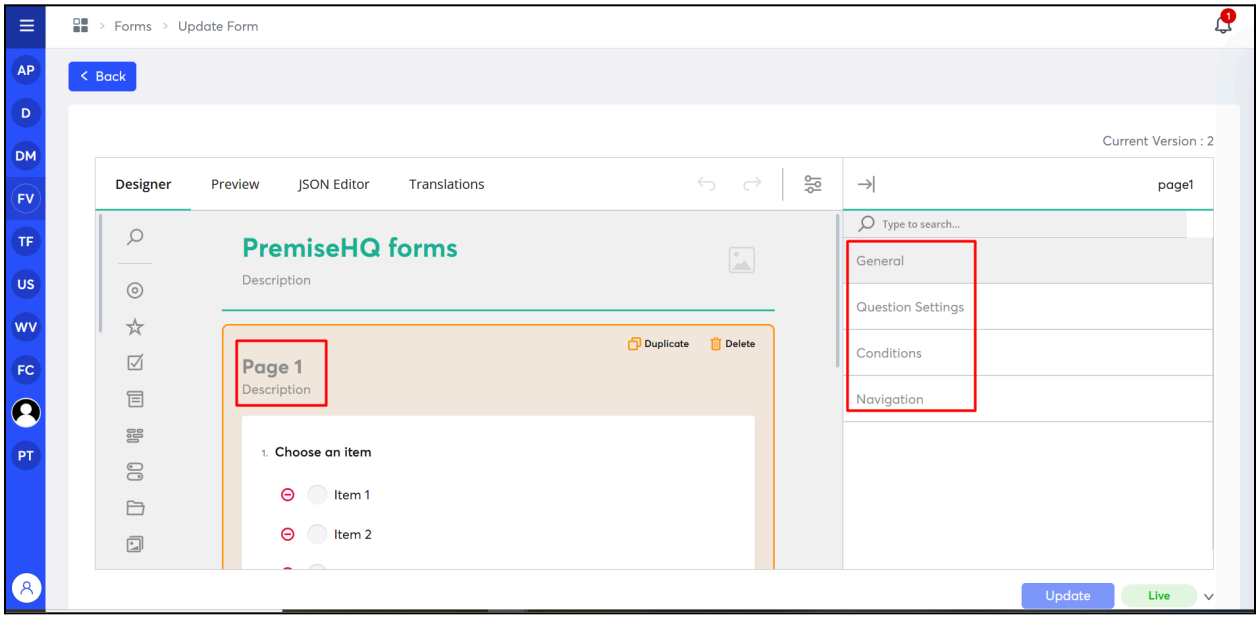
- Set conditions to show or hide the page, read-only mode and required of the page based on user inputs in previous questions. For example:Display this page only if a particular question's response matches a certain value.

d. Navigation

The **Navigation** settings in the image allow you to control the visibility of navigation buttons for a page in the form.
The options are:

- **Inherit**: The navigation button settings will follow the default behavior or settings applied globally to the form.
- **Visible**: Navigation buttons (e.g., "Next" or "Previous") will be explicitly shown on this page, regardless of global settings.
- **Hidden**: Navigation buttons will be hidden on this page, preventing users from moving to the next or previous page using these buttons.

These settings give you flexibility to manage the user's flow and interaction with the form on a per-page basis.



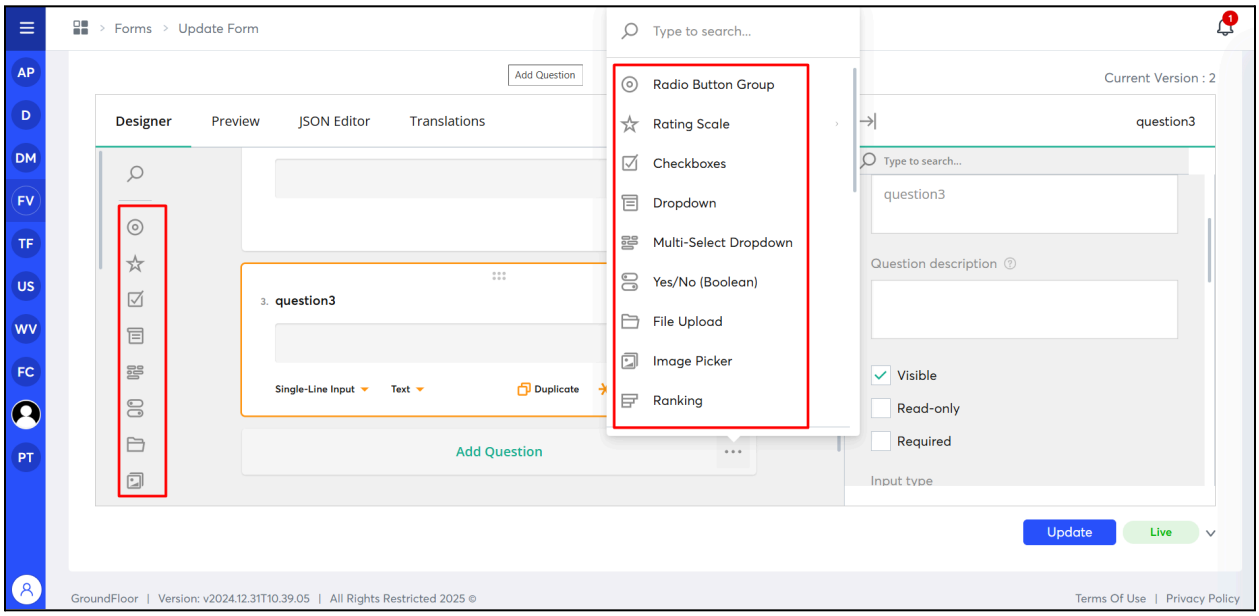
3. Adding Questions to a Form

To add questions to a form click **"Add Question"**: Choose from various question types such as text, checkbox, dropdown, and more.

a. Configure the Question:
- **Title**: Enter the question text.
- **Description**: Add optional details to guide users.
- **Is Required**: Toggle to make the question mandatory.
- **Placeholder**: Add sample text or hints for input fields.

b. Available Form Fields:
- **Radio Button Group:** Presents multiple options where the user can select only one. Useful for single-choice questions.
- **Rating Scale:** Allows users to rate an item on a scale, such as 1 to 5 or 1 to 10.
- **Checkboxes:** Lets users select multiple options from a given list.
- **Dropdown:** Provides a dropdown menu with predefined options for single selection.
- **Multi-Select Dropdown:** Similar to the dropdown but allows multiple selections.
- **Yes/No (Boolean):** Provides a simple Yes/No toggle option.
- **File Upload:** Allows users to upload files as part of their responses.
- **Image Picker:** Enables users to select one or multiple images.
- **Ranking:** Allows users to rank items in order of preference or priority.
- **Single-Line Input:** A text field for short answers like names or email addresses.
- **Long Text:** A text area for longer responses, such as feedback or descriptions.
- **Multiple Textboxes:** A group of input fields displayed together for structured data.
- **Panel:** Groups related questions visually on the form.
- **Dynamic Panel:** A group of fields that users can duplicate dynamically.
- **Single-Select Matrix:** A table format where users select one option per row.
- **Multi-Select Matrix:** Similar to Single-Select Matrix but allows multiple selections per row.
- **Dynamic Matrix:** A dynamic table where users can add rows with predefined columns.
- **HTML:** Allows embedding custom HTML content for instructions, styling, or visuals.
- **Expression (read-only):** Displays calculated values or expressions that are non-editable.
- **Image:** Inserts an image into the form for display purposes.
- **Signature:** Allows users to sign digitally using a touchscreen or mouse.

- **Async Select Dropdown:** A dropdown with dynamic data loading for large datasets.
- **Button:** Adds a clickable button for custom actions.
- **Smart Text Input:** An advanced input field with features like auto-suggestions or formatting.



4. All Input fields properties and settings:

a. Single Line Input:

The Single Line Input Field Settings page is divided into six key sections: General Settings, Layout, Conditions, Input Mask Settings, Data, and Validation. Each section is designed to configure specific aspects of the input field in the form. Below is a detailed explanation of each section.

- **General Settings:**
  This section configures basic properties for the field:
  - Question Name: An internal identifier not visible to respondents.
  - Question Title: A user-friendly title displayed to respondents.
  - Question Description: A subtitle providing additional information about the question.
  - Visibility:
    - Hidden: Makes the field invisible to respondents.
    - Read-Only: Displays the field but restricts input.
    - Required: Ensures the field must be filled before submission.
  - Input Type: Validates the input against the selected type. Options include:

        - Text
        - Number
        - Email
        - URL
        - Date/Time (with variations like "Date", "Month", "Time", etc.)
        - Password
        - Phone Number
        - Range
        - Week
    - Placeholder Text: Sets default text within the input field for guidance.
    - Autocompletion Type: Defines the type of data that the user's browser retrieves for autofill.
    - Auto-Suggest Items: Provides a list of predefined suggestions during input.
- **Layout:** Controls the field's appearance:
    - Display Settings:
        - Display on a new line or inline with previous questions.
        - Hide or show the question number.
    - Question Box State:
        - Expanded: Fully displayed and collapsible.
        - Collapsed: Initially minimized but expandable.
        - Locked: Fully displayed and non-collapsible.
    - Alignment Options:
        - Title Alignment: Choose "Top", "Bottom", "Left", "Hidden", or "Inherit".
        - Description Alignment: Place under the input field, under the title, or inherit default rules.
        - Error Message Alignment: Set to "Top", "Bottom", or inherit default settings.
        - Inner Indent: Adds margin between the question content and its container.
    - Width Settings:
        - Inline question width (CSS values like %, px, etc.).
        - Minimum and maximum width of the question box.
        - Input field width (measured in characters).

- **Conditions:** Define rules for field behavior:
    - Visibility Rules: Make the question visible under specific conditions.
    - Read-Only Rules: Enable/disable read-only mode based on conditions.
    - Required Rules: Make the field mandatory only when conditions are met.

        - Default Value Expression: Calculate default values using expressions or functions.
        - Reset Value Rules: Reset the field's value based on conditions.
        - Dynamic Value Assignment:
            - Use expressions to set the value dynamically.
            - Respondent inputs can override this value.
    - **Input Mask Settings:** Format input fields with specific patterns:
        - Mask Type:
            - None
            - Pattern (custom placeholders and literals)
            - Date and Time Formatting:
            - Placeholder patterns like mm/dd/yyyy or hh:mm:ss.
        - Numeric Formatting:
            - Set minimum/maximum values and decimal precision.
            - Define separators for decimals and thousands.
            - Allow or disallow negative values.
        - Currency Formatting:
            - Add prefix/suffix symbols.
            - Define separators and value ranges.
    - **Data:** Specify data-related configurations:
        - Set default values.
        - Manage data bindings or integrations (e.g., connecting to an external database).
    - **Validation:** Ensure input accuracy and adherence to rules:
        - Define custom validation rules.
        - Specify error messages for invalid inputs.
        - Limit the maximum character length.

    This section provides a comprehensive explanation of the single-line input field's settings to help users effectively configure forms.

b. **Radio Button Group**

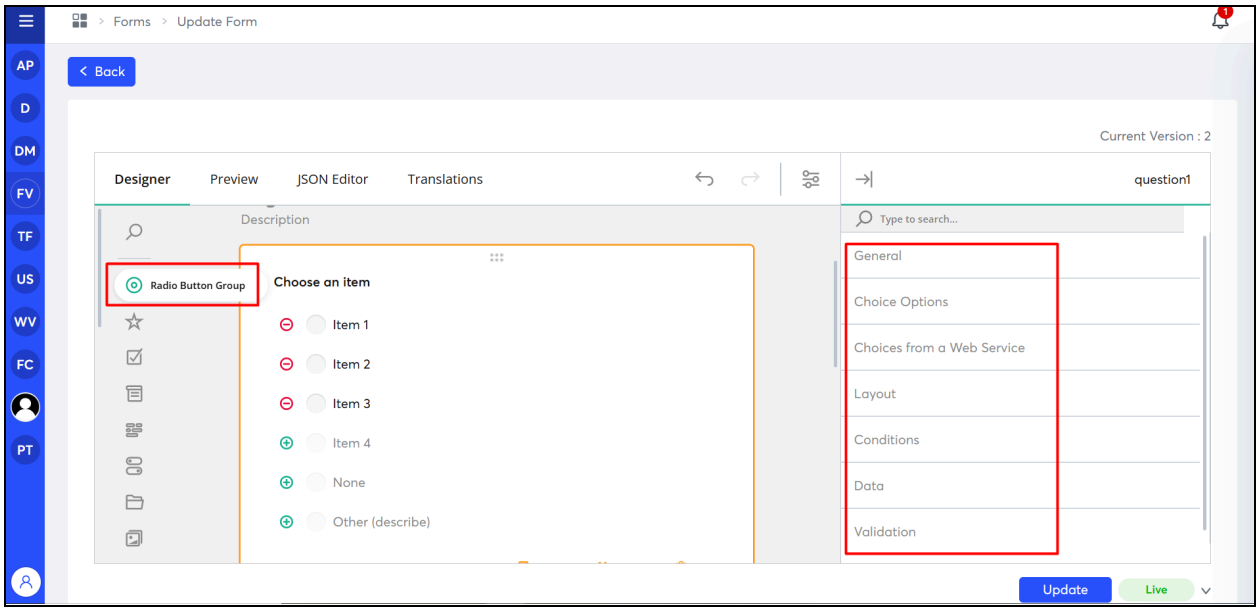
A radio group field lets users pick one option from a list by clicking a small circle next to their choice. Once you select one, the others are automatically unselected. Users can customize the field using the settings in the following ways:

- **General Settings:**
  In the General settings, you can configure the following:
    - **Title:** This is the question text displayed to users. For example, "What is your favorite color?"
    - **Description:** An optional field to provide additional information or context for the question. For example, "Please select one option that best matches your preference."

- **Choices:** Add, remove, or edit the list of options. Each choice represents a single selectable option in the radio button group. Users can reorder choices using drag-and-drop or randomize their display order.
- **Default Value:** You can set a preselected option for users. For example, "Red" can be pre-selected if it's the most common answer.
- **Other Option:** Enable an "Other" choice, allowing users to type a custom response that is not listed among the predefined options.
- **None Option:** Add a "None" option to allow users to indicate that none of the provided choices apply to them.
- **Visibility Condition:** Use conditional logic to show or hide the question based on other responses. For example: "Display this question only if the answer to 'Do you like colors?' is 'Yes.'"
- **Read-Only Mode:** Make the question read-only to display it as non editable for users. This is often used in cases where you want to show a pre-selected option but not allow changes.
- **Validation and Logic:** Required Mark the question as mandatory, ensuring users select one of the options before proceeding.
- **Custom Validation Logic:** Define advanced logic using expressions, such as ensuring an answer matches specific conditions.

- **Choices from a Web Service**
  - **Dynamic Choices**: Fetch options dynamically from an external web service or API.
  - **Configuration**: Specify the URL or endpoint to retrieve the data.
  - **Real-Time Updates**: Ensure the list updates in real-time based on external changes.

- **Styling Options**
  - Customize the layout of the choices (vertical or horizontal alignment).
  - Adjust spacing between options.
  - Use custom CSS classes to change colors, fonts, or the size of the radio buttons.



- **Validation Rules**
  Validation rules can be customized for each field to ensure proper data input and guide users effectively.

    - **Text Validation:** Throws an error if the entered text's length is outside the specified minLength and maxLength range.
    - **Expression Validation:** Allows adding logic or performing calculations in the form fields. Expressions are evaluated dynamically at runtime.

- **Supported expression types:**
    - **String Expression:** Evaluates to a string value. Example: "expression": "iif(age({birthdate}) >= 21, 'Adult', 'Minor')"
    - **Numeric Expression:** Evaluates to a number. Example: "expression": "sum({total1}, {total2})"
    - **Boolean Expression:** Evaluates to true or false.
    - **Number Validation:** Throws an error if the input is not a number or falls outside the specified minValue and maxValue range.
    - **Email Validation:** Throws an error if the entered value is not in a valid email format.
    - **Regex Validation:** Throws an error if the input does not match the pattern defined in the regex property.

c. Rating Scale

A rating scale field is a simple explanation of what the scale is for. It helps users to rate things correctly. This makes sure everyone's answers are

clear and consistent. On our forms, user can use a rating scale field and update the features for the field in the following process:

Field Settings and Features for the Rating Scale

- **General**
  - **Question Title**: Set or modify the question's title. Example: "How satisfied are you with our service?"
  - **Description**: Add a short description or instruction under the question to provide context to the user.
  - **Required**: Mark the question as mandatory or optional for form completion.
  - **Read-only**: Option to make the question non-editable by respondents.

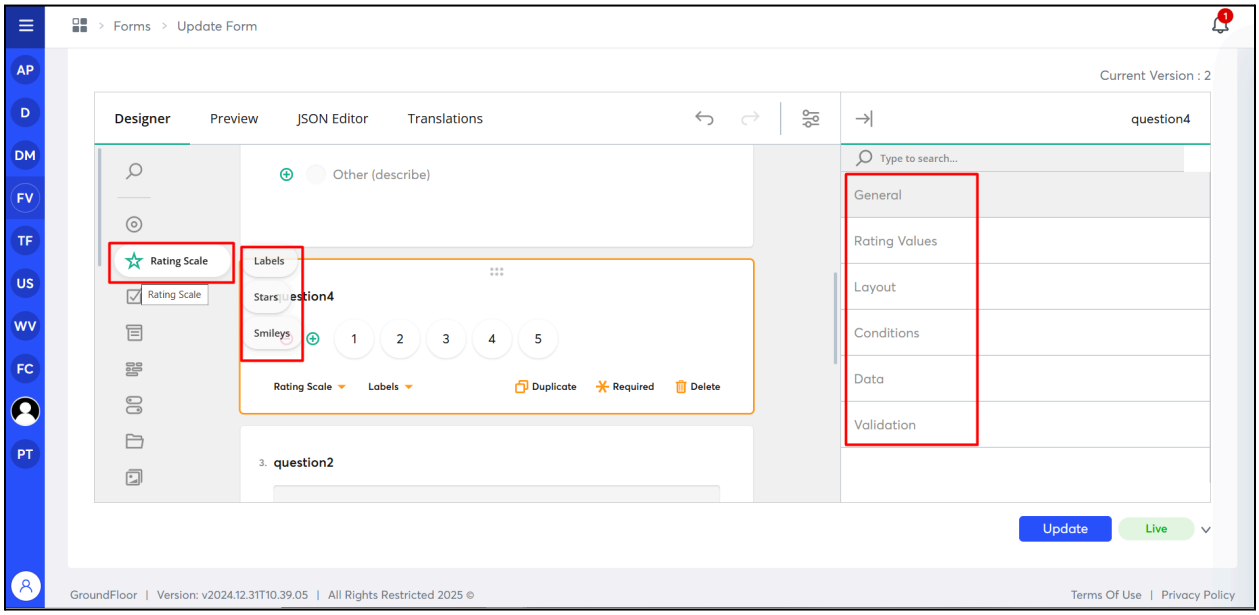
- **Rating Values**
  - **Define Rating Range**: Specify the range of values (e.g., 1 to 5, or 1 to 10).
  - **Custom Labels**: Assign meaningful labels to each value (e.g., 1 = "Very Dissatisfied," 5 = "Very Satisfied").
  - **Default Value**: Set a preselected value if needed.
  - **Allow Partial Ratings**: Enable decimal values for more precise ratings (e.g., 4.5).

- **Layout**
  - **Orientation**: Choose between a horizontal or vertical layout for the rating scale.
  - **Step Size**: Adjust the increment between rating values (e.g., 1, 0.5).
  - **Width Adjustment**: Define the width or spacing of the rating scale to fit the design.

- **Conditions**
  - **Display Logic**: Show or hide the question based on answers to previous questions. For example:
  - If "Overall Satisfaction" is less than 3, display this question.
  - **Enable/Disable Logic**: Control if the field should be active or inactive under certain conditions.
  - **Advanced Logic**: Use multiple conditions combined with "AND"/"OR" operators.
- **Data**
  - **Field Name**: Assign a unique internal name for easy data reference in reports.
  - **Data Validation**: Ensure responses fall within the valid range or adhere to specific rules.
  - **Skip Logic**: Skip to a specific page or question based on the respondent’s input.

- **Validation**
  - **Range Enforcement**: Ensure values fall within the defined range (e.g., 1 to 5).
  - **Error Messages**: Customize error messages if validation fails. For instance:
  - "Please select a rating between 1 and 5."
  - **Dependencies**: Set dependencies for this field based on responses to other fields.

- **Toolbar Features for Rating Scale**
  - **Duplicate**: Quickly duplicate the question for reuse elsewhere in the form.
  - **Required**: Toggle to make the question mandatory.
  - **Delete**: Remove the question from the form.
  - **Labels**: Easily customize labels for each rating value from the toolbar.



d. Checkboxes

In **Checkboxes** question type within a form designer interface, this type of question allows users to select multiple options from a list of predefined choices. Below is a structured description of its key components:

- **General**
  - **Question Title**: Set or update the question's title. Example: "Select the items you need."
  - **Description**: Add a brief explanation or instruction under the question to guide the respondent.

- **Required**: Mark the question as mandatory for Form completion.
- **Read-only**: Make the field non-editable to display predefined selections.

- **Choice Options**
  - **Add/Edit Choices**: Define the list of checkbox items. Example: "Item 1," "Item 2," etc.
  - **Select All Option**: Automatically include a "Select All" checkbox for easier selection.
  - **None Option**: Include a "None" option for respondents who don't choose any items.
  - **Other Option**: Add an "Other" checkbox with an optional text field for custom input.

- **Choices from a Web Service**
  - **Dynamic Choices**: Fetch options dynamically from an external web service or API.
  - **Configuration**: Specify the URL or endpoint to retrieve the data.
  - **Real-Time Updates**: Ensure the list updates in real-time based on external changes.

- **Layout**
  - **Orientation**: Choose between vertical or horizontal alignment of checkboxes.
  - **Columns**: Split the options into multiple columns for better readability.
  - **Spacing**: Adjust the spacing between checkbox items.
  - **Custom Styling**: Apply custom styles to match the form design.

- **Conditions**
  - **Display Logic**: Show or hide the field based on responses to previous questions. For example:
  - If "Product Type" is "Electronics," show specific checkbox options.
  - **Enable/Disable Logic**: Activate or deactivate the field based on conditions.
  - **Advanced Logic**: Use combinations of "AND"/"OR" operators to set complex conditions.
- **Data**
  - **Field Name**: Assign a unique identifier for easier reference in data exports or reports.
  - **Default Selections**: Pre-select one or more checkboxes by default.

- **Data Validation**: Ensure the respondent selects a valid number of options (e.g., "Select at least 2 items").

- **Validation**
  - **Minimum/Maximum Selection**: Set rules for the number of items a respondent can or must select.
  - **Error Messages**: Customize error prompts for invalid selections. For example: "Please select at least one item."
  - "You cannot select more than three items."
  - **Dependencies**: Ensure the validation aligns with other fields or form rules.

- **Toolbar Features for Checkboxes**
  - **Duplicate**: Clone the question for reuse within the form.
  - **Required**: Toggle to make the field mandatory or optional.
  - **Delete**: Remove the question from the form.

e. Dropdowns:

- **General**
  - **Question Title**: Set or update the question's title. Example: "Select the items you need."
  - **Description**: Add a brief explanation or instruction under the question to guide the respondent.
  - **Required**: Mark the question as mandatory for form completion.
  - **Read-only**: Make the field non-editable to display predefined selections.

- **Choice Options:**
  - Enables you to manage the predefined options that users can select from the dropdown list.
  - You can add, edit, or remove items (e.g., "Item 1," "Item 2") displayed in the dropdown menu.

- **Choices from a Web Service:**
  - Provides functionality to dynamically populate dropdown options by fetching data from an external web service or API.

- **Layout:**
  - Configures the visual layout and appearance of the dropdown question, such as alignment, width, or display style.

- **Conditions:**

- Allows you to set conditional logic for the question. For instance, show or hide the dropdown based on responses to previous questions.

- **Data:**
  - Configures data-specific properties, such as variable naming for backend integration or data validation requirements.

- **Validation:**
  - Provides options to set validation rules, ensuring the user's selection meets predefined criteria (e.g., requiring a selection or restricting choices).

These settings ensure the dropdown question is fully customizable to meet both functional and design requirements.

f. **Multi Select Dropdown**

The marked field in the image represents the settings panel for a **Multi-Select Dropdown** question. Here's an explanation of each section:

- **General**
  - **Question Title**: Set or update the question's title. Example: "Select the items you need."
  - **Description**: Add a brief explanation or instruction under the question to guide the respondent.
  - **Required**: Mark the question as mandatory for form completion.
  - **Read-only**: Make the field non-editable to display predefined selections.

- **Choice Options**:
  Allows you to manage the available options for the dropdown:
  - Add, edit, or delete individual options (e.g., "Item 1," "Item 2").
  - Enable or disable the "Select All" option for ease of selection.

- **Choices from a Web Service**:
  - Provides functionality to dynamically populate options using data fetched from an external API or web service.

- **Layout**:
  - Manages the appearance of the dropdown:
  - Control its size, width, and alignment.
  - Customize its visual placement within the form.

- **Conditions**:
  - Defines conditional logic for showing, hiding, or enabling the dropdown based on responses to other questions.
- **Data**:
  - Sets up integration-specific properties, including variable names for backend data storage and system compatibility.

- **Validation**:
  Configures rules for ensuring valid user responses:
  - Set minimum or maximum selections.
  - Enforce required selections based on specific criteria.

These settings allow you to fully customize the behavior and appearance of the **Multi-Select Dropdown** question, making it adaptable for various use cases.

g. **Yes-No (boolean)**

The marked field in the image represents the **settings panel** for a **Yes/No (Boolean)** question. Here's a detailed description of the available settings:

- **General:**
  - **Question Label**: The text displayed as the question prompt (e.g., "question8").
  - **Mandatory Toggle**: Enables or disables whether the user must answer the question.
  - **Tooltip or Help Text**: Provides additional information or guidance to users about the question.
  - **Default Value**: Specifies whether "Yes" or "No" is pre-selected.

- **Layout:**
  - **Alignment**: Adjusts the placement of the question (e.g., left, center, or right).
  - **Style Options**: Customizes the design, such as button size, spacing, or visual theme.

- **Conditions:**
  - **Visibility Rules**: Configures when this question should be shown or hidden based on responses to previous questions.
  - **Enable/Disable Logic**: Allows the question to be enabled or disabled under specific conditions.

- **Data:**
  - **Variable Name**: Sets a unique identifier for the question, useful for data collection and backend integration.

- **Export Options**: Defines how this response is included in data outputs (e.g., JSON format).

- **Validation:**
  - **Required Response**: Ensures the user selects either "Yes" or "No" before proceeding.
  - **Custom Error Message**: Displays a specific error message if the validation criteria are not met.

These settings allow you to fully customize the behavior, appearance, and data handling for the **Yes/No (Boolean)** question.

h. **File Upload**

- **General**
  Provides basic settings for the file upload field:
  - Field title (e.g., "question9" in the image).
  - Description or tooltip for additional context.
  - Options for enabling/disabling the field.

- **Layout**
  Handles visual customization:
  - Adjust alignment and positioning of the field within the form.
  - Define margins and spacing.
  - Choose field size (e.g., small, medium, or large).

- **Conditions**
  - Allows you to add logic for showing or hiding the field based on responses:
  - Define conditions to make the field visible or invisible dynamically.
  - Specify criteria for enabling or disabling the field.

- **Data**
  Configures data-related options:
  - Bind uploaded files to variables or external storage systems.
  - Specify default files (if applicable).
  - Set file upload limits like maximum file size and types.

- **Validation**
  Enforces restrictions on user inputs:
  - Mark the field as mandatory.
  - Restrict file types (e.g., .pdf, .jpg, etc.).
  - Limit the maximum file size or number of files.
  - The **File Upload** field enables respondents to upload files completion while giving creators flexibility in design and validation through these settings.

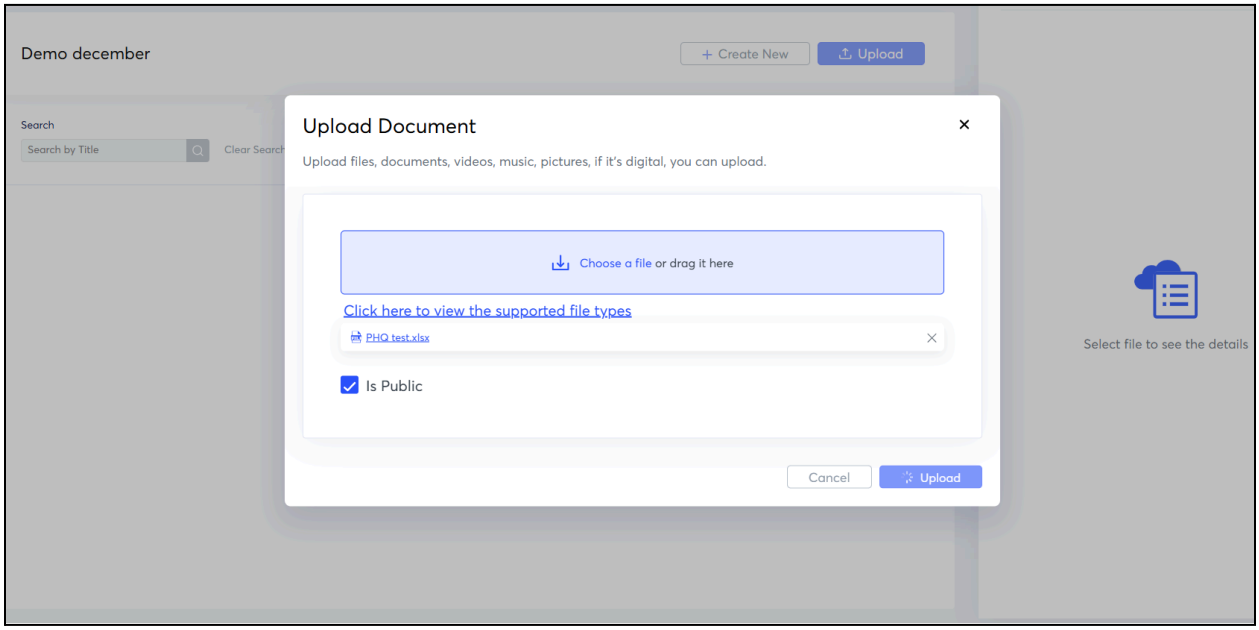
5. **Adding a Sample File to a Form:**

   To include a sample file in a form, users need to utilize the Document Management application and follow a two-task process as described below.

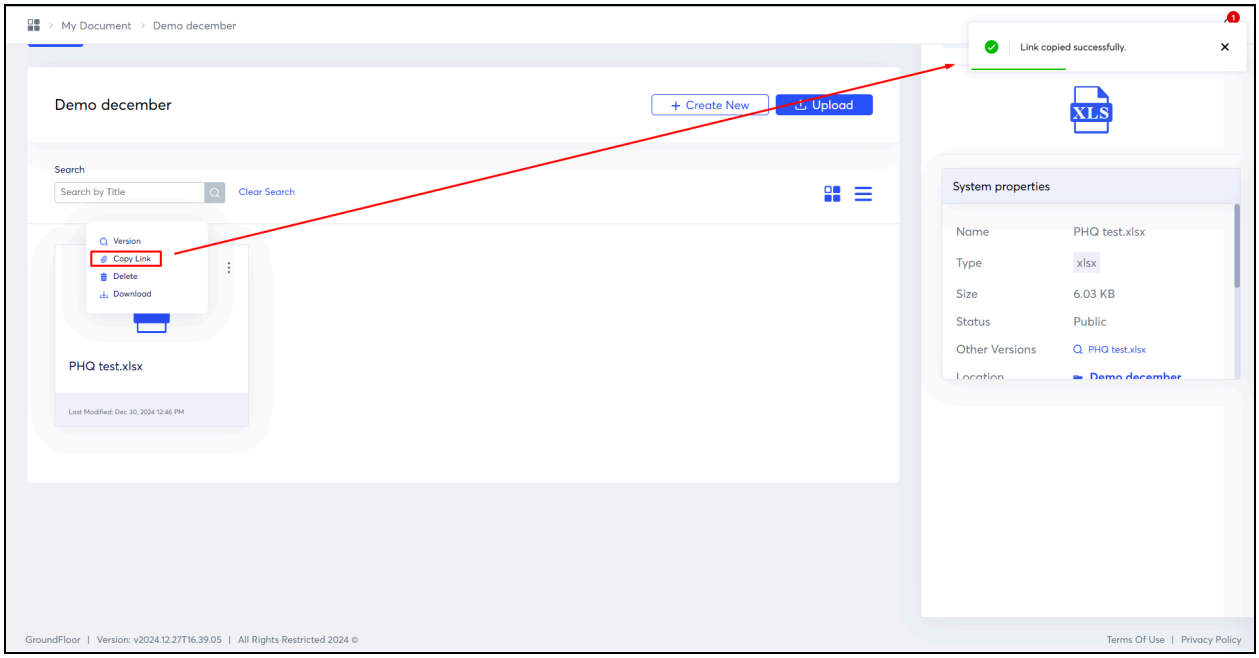
Task 1: Upload the Sample File in the Document Management Application

A. Create a Folder:
- Navigate to the **Document Management** application.
- Create a new folder to organize your files (optional but recommended).

B. Upload the Sample File:
- Open the newly created folder.
- Upload the sample file.
- Ensure the file is set as **Public**.



C. Copy the File Link:
- Click the **Action** button (three-dot menu) for the uploaded file.
- Select the option to **Copy File Link**.
- The file URL will be copied to your clipboard.



Task 2: Add the File Link to the Form

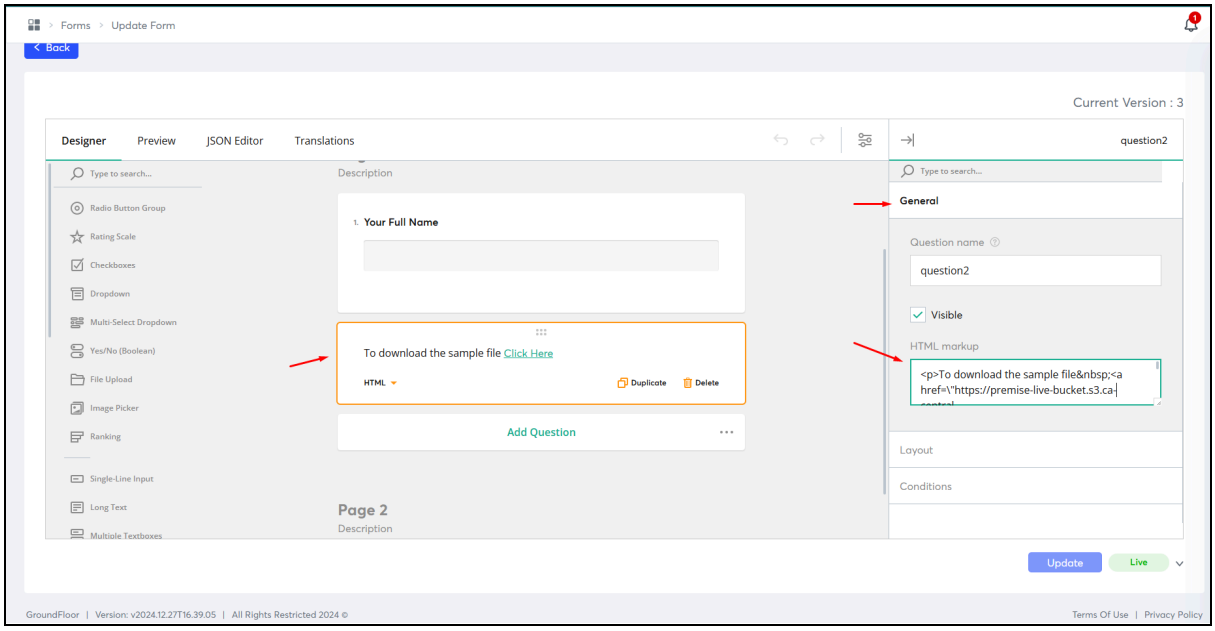
A. Edit the Form:
- Go to the Forms Application and locate the desired form.
- Open the form for editing.

B. Add an HTML Field:
- Drag and drop an HTML Field into the form.

C. Modify the HTML Field Settings:
- Navigate to the HTML General Settings for the new field.
- Locate the HTML Markup section.

D. Insert the File URL:
- Paste the copied file link into the markup section.
- Example of an HTML markup:

```
<p>To download the sample file&amp;nbsp;<a download=\"\" href=\"https://premise-live-bucket.s3.ca-central-1.amazonaws.com/GroundFloorV2/26b27e505e7a434caff87bd59c777786/07e2df23a01149b4a726e5c89d4e2226/PHQ test.xlsx\">Click Here</a></p>
```

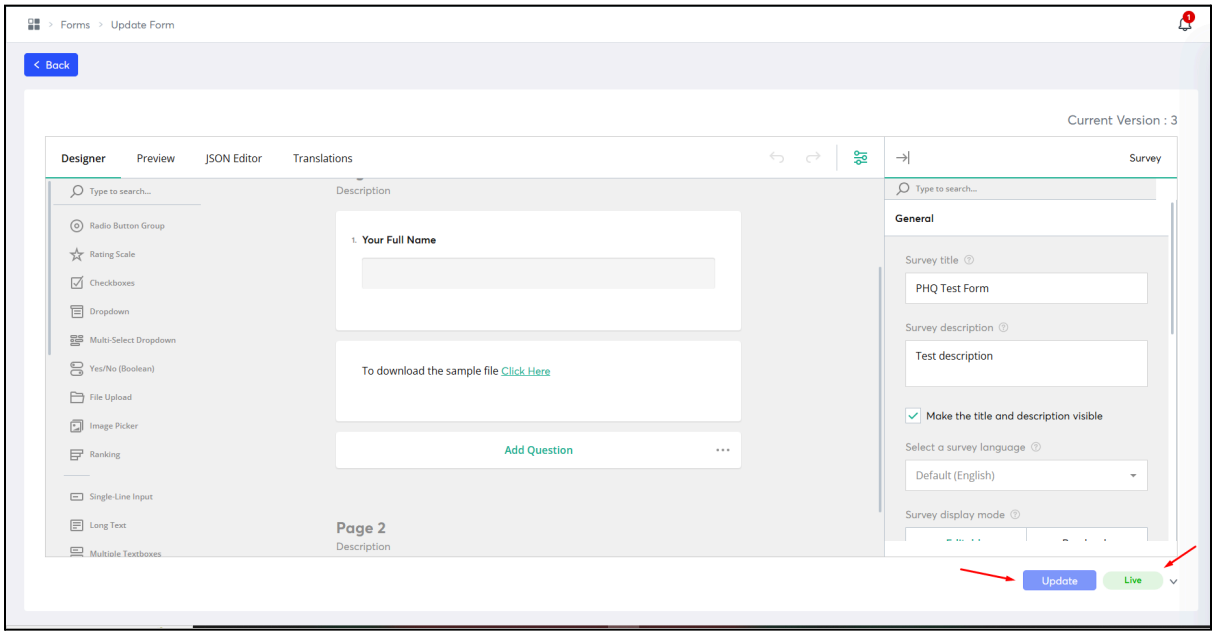


E. Save the Form:

- Click Save to save your changes to the form.

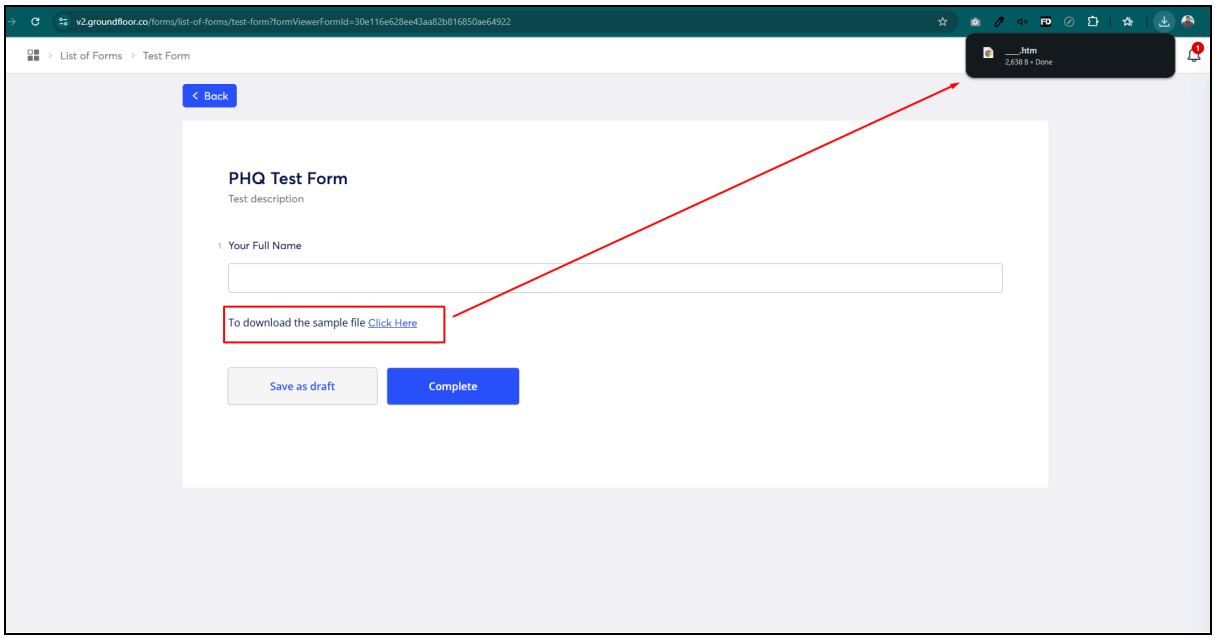
F. Make the Form Live:

- Change the form's status to Live.



G. Testing the Sample File Download

- Go to the section where the form is used (e.g., List of Forms, Public Form, or Common Form, Workflow initiate page ETC).
- Click on the hyperlink for the sample file.
- Verify that the sample file is downloaded as expected.



H. Additional Notes

- Ensure the file permissions are set to Public in the Document Management application to avoid access issues.
- Use the correct file URL format in the HTML Markup to create a functional hyperlink.

6. Choices from a Web Service:

The **Choices from a Web Service** feature allows you to dynamically load choices for form fields such as **Dropdown**, **Multiple Choice**, and **Radio Group** from an external API. This ensures that the options are always up-to-date and relevant.

A. Feature Overview

This feature supports dynamic integration with APIs and allows you to:

- Fetch and display choices from an API.
- Customize which property from the response data is displayed and stored.
- Handle hierarchical or nested data structures using a **Path to Data** field.
- Optionally accept empty responses for flexibility.

Steps to Configure Choices from a Web Service:

B. **Navigate to the Form**:
- Open the form where you want to enable this feature.

C. **Add the Field**:

- Drag and drop or select a **Dropdown**, **Multiple Choice**, or **Radio Group** field.

D. **Access Field Properties**:
   - Click the **three dots** on the top-right corner of the field.
   - Select **Property** to open the settings.

E. **Enable Choices from Web Service**:
   - Scroll down to the **Choices from a Web Service** section.
   - Expand the section to view its settings.

F. **Configure the Web Service**:
   - **Web Service URL**: Enter the API endpoint URL that returns the data (e.g., https://surveyjs.io/api/CountriesExample).
   - **Path to Data**: If the response contains nested data, specify the path to the required array using dot notation (e.g., categories.fiction).
   - **Get Value to Store**: Enter the property name from the API response that should be saved when the user selects an option (e.g., id).
   - **Get Value to Display**: Enter the property name from the API response that should be shown to the user (e.g., name).

G. **Optional Settings:**
   - **Accept Empty Response:** Check this box if the API might return an empty response and the form should handle it without errors.

H. **Remove Form Entry Field** (if applicable):
   - Delete unnecessary form entry fields that may interfere with the configuration.

I. **Save and Preview:**
   - Save the form and click Preview to test the functionality.
   - Verify that the field dynamically loads data and behaves as expected.

J. **Correct Response Format:**
   - The API must return an array of objects with consistent keys. Below is an example of a supported response format:
   - In this case: The name field contains the values to be displayed as options.

[{ "name": "Afghanistan", "officialName": "Islamic Republic of Afghanistan", "region": "Asia", "cca2": "AF", "ccn3": "AF", "cca3": "AFG", "cioc": "AFG" }, { "name": "Angola", "officialName": "Republic of Angola", "region": "Africa", "cca2": "AO", "ccn3": "AO", "cca3": "AGO", "cioc": "ANG" }, { "name": "Albania", "officialName": "Republic of Albania", "region": "Europe", "cca2": "AL", "ccn3": "AL", "cca3": "ALB", "cioc": "ALB" }]

K. **Incorrect Response Format:**
   - The feature does not support nested objects or arrays. Below is an example of a wrong response format:
   - In this case: The name field is nested, making it incompatible with the Choices from Web feature.

[{ "name": { "common": "Moldova", "official": "Republic of Moldova", "nativeName": {

"ron": { "official": "Republica Moldova", "common": "Moldova" } } }, "tld": [".md"], "cca2": "MD", "ccn3": "498", "cca3": "MDA", "cioc": "MDA", "independent": true, "status": "officially-assigned", "unMember": true, "currencies": { "MDL": { "name": "Moldovan leu", "symbol": "L" } } }]

L. **Example API Response and Configuration**

- **Correct Response Format** (Array of Objects):

[{ "id": 1, "name": "Book A", "category": "Fiction" }, { "id": 2, "name": "Book B", "category": "Non-fiction" }]

- **Configuration**:
  i. **Web Service URL**: https://api.example.com/books
  ii. **Path to Data**: Leave blank (the data is at the root level).
  iii. **Get Value to Store**: id
  iv. **Get Value to Display**: name

M. **Advanced Use Cases**

- **Handling Nested Data**: If the data is nested, use the **Path to Data** field to specify the path. For example:

{ "categories": { "fiction": [{ "id": 1, "name": "Book A" }, { "id": 2, "name": "Book B" }] } }

- **Configuration**:
  i. **Path to Data**: categories.fiction
  ii. **Get Value to Store**: id
  iii. **Get Value to Display**: name
- **Accepting Empty Responses**: Check **Accept Empty Response** if your API might return an empty array and you still want the form to function. For example:

[]

N. **Use Cases**

- Real-time country lists for user selection.
- Dynamically updating product categories or options.
- Fetching up-to-date organizational or project names from an external system.

O. **Notes:**

- This feature is compatible only with APIs returning an array of objects.
- Nested arrays or deeply nested structures may require additional configuration in the **Path to Data** field.

- If the API response format changes, you may need to update the configuration.
- Always test the API response format before saving the form to ensure compatibility.
- Ensure the API is accessible and provides the required data in the correct format.
- The same configuration applies to **Dropdown**, **Multiple Choice**, and **Radio Group** fields.

7. **Data Validation:**

   The Validation section ensures that the input provided by users meets specified requirements before it is accepted. This section supports both client-side and server-side validation. Validation can occur immediately after input, upon page navigation, or when the form is submitted.

   a. **Immediate Data Validation:** Validation is triggered under different scenarios:
      - **Default Behavior:** Errors are checked when respondents navigate to the next page.
      - **Immediate Validation:**
        - Enable this by setting checkErrorsMode to "onValueChanged".
        - To validate user input while typing, set textUpdateMode to "onTyping".
          Example:

```
const surveyJson = {
  "checkErrorsMode": "onValueChanged",
  "textUpdateMode": "onTyping",
  "elements": [
    // ...
  ]
};
```

        - Postponed Validation: Validation occurs only when the form is submitted. Set checkErrorsMode to "onComplete".

   b. **Built-In Client-Side Validators:** These validators are available to validate user input against common requirements:
      - Required Validator: Ensures the field is not left empty. Use isRequired to activate this, and specify requiredErrorText for

custom error messages.
Example:

```
const surveyJson = {
  "elements": [{
    "name": "question1",
    "type": "text",
    "isRequired": true,
    "requiredErrorText": "Value cannot be empty"
  }]
};
```

- Other Validators:
- Numeric Validator: Ensures input is a number and optionally within a range (defined by minValue and maxValue).
- Text Validator: Ensures the text length is within minLength and maxLength.
- Email Validator: Validates email format.
- Expression Validator: Checks conditions using custom expressions.
- Regex Validator: Ensures input matches a defined regular expression.
- AnswerCount Validator: Validates the number of selected options for multiple-choice questions. Example with multiple validators:

```
const surveyJson = {
  "elements": [{
    "name": "question1",
    "type": "text",
    "validators": [
      { "type": "numeric", "text": "Value must be a number" },
      { "type": "regex", "regex": "^[A-Z]+$", "text": "Only uppercase letters are allowed" }
    ]
  }]
};
```

c. **Custom Client-Side Validation:**

You can implement custom logic using the onValidateQuestion event.
Example:

```
survey.onValidateQuestion.add((survey, options) => {
```

```
  if (options.name === "memo" && options.value.indexOf("survey") === -1) {
    options.error = 'Your answer must contain the word "survey"';
  }
});
```

Alternatively, use expressions:

```
const surveyJson = {
  "elements": [{
    "name": "memo",
    "type": "comment",
    "validators": [{
      "type": "expression",
      "text": "Your answer must contain the word \"survey\"",
      "expression": "validateComment({memo}) >= 0"
    }]
  }]
};
```

d. **Server-Side Validation:**

Server-side validation is useful when the validation logic requires data from external sources. Use the onServerValidateQuestions event to perform asynchronous checks.
Example:

```
survey.onServerValidateQuestions.add((survey, { data, errors, complete }) => {
  const country = data["country"];
  if (!country) {
    complete();
    return;
  }
  fetch("https://example.com/validate?country=" + country)
    .then(response => response.json())
    .then(isValid => {
      if (!isValid) {
        errors["country"] = "Country is not valid";
```

```
    }
    complete();
  });
});
```

e. **Postpone Validation Until Survey Ends:**

You can delay all validation checks until the respondent clicks the Submit button by setting checkErrorsMode to "onComplete". This allows respondents to review and correct all errors at once. Example:

```
const surveyJson = {
  "checkErrorsMode": "onComplete",
  "elements": [
    // ...
  ]
};
```

f. **Switching Between Pages with Errors:**

By default, respondents cannot navigate away from a page containing validation errors. You can override this by setting validationAllowSwitchPages to true. This feature is helpful in long surveys where respondents may want to revisit sections. Example:

```
survey.validationAllowSwitchPages = true;
```

g. **Best Practices:**

- Use Immediate Validation for fields where errors need to be corrected in real-time (e.g., email addresses).
- Employ Postponed Validation for complex forms where users may prefer to review inputs before submission.
- Combine Built-In Validators with Custom Validation for advanced scenarios like integrating external data checks.

8. **Build-in function for Expression Validation:**

Description: Functions allow you to perform additional calculations within an expression.

a. **Iif:**
   Description: iif(condition: expression, valueIfTrue: any, valueIfFalse: any): any Returns the valueIfTrue value if the condition is truthy or the valueIfFalse value if the condition is falsy.
   Example: "expression": "iif({question1} + {question2} > 20, 'High', 'Low')"

b. **isContainerReady:**
   Description: isContainerReady(nameOfPanelOrPage: string): boolean Returns true if all questions in a given panel or page have valid input; otherwise, returns false. An empty question value is considered valid if neither validators nor required status is defined for it.
   Example: "expression": "isContainerReady('page1')"

c. **isDisplayMode:**
   Description: isDisplayMode(): boolean Returns true if the form is in display or preview mode.
   Example: "expression": "isDisplayMode()"

d. **Age:**
   Description: age(birthdate: any): number Returns age according to a given birthdate. The date argument (which is typically taken from a question) should be defined as a valid JavaScript Date.
   Example: "expression": "age({birthdate})"

e. **currentDate:**
   Description: currentDate(): Date Returns the current date and time.
   Example: "expression": "currentDate()”

f. **Today:**
   Description: today(daysToAdd?: number): Date Returns the current date or a date shifted from the current by a given number of days.
   For example, today() returns the current date, 0 hours, 0 minutes, 0 seconds; today(-1) returns yesterday's date, same time; today(1) returns tomorrow's date, same time.
   Examples: "expression": "today()" "expression": "today(2)"

g. **Year:**
   Description: year(date?: Date): number Returns the year of a given date.
   Example: "expression": "year({birthdate})"

h. **Month:**
Description: month(date?: Date): number Returns the month of a given date as a value from 1 (January) to 12 (December).
Example: "expression": "month({birthdate})”

i. **Day:**
Description: day(date?: Date): number Returns the day of the month for a given date as a value from 1 to 31.
Example: "expression": "day({birthdate})"

j. **Weekday:**
Description: weekday(date?: Date): number Returns the day of the week for a given date as a value from 0 (Sunday) to 6 (Saturday).
Example: "expression": "weekday({birthdate})"

k. **getDate:**
Description: getDate(questionName: expression): Date Returns a Date value converted from a given question's value.
Example: "expression": "getDate({birthdate})"

l. **dateDiff:**
Description: dateDiff(fromDate: any, toDate: any, "days" | "months" | "years"): number Returns a difference between two given dates in full days (default), months, or years.
Example: "expression": "dateDiff({birthdate}, today(), "months")”

m. **diffDays:**
This function is obsolete. Use the dateDiff function instead. Description: diffDays(fromDate: any, toDate: any): number Returns the number of days between two given dates.
Example: "expression": "diffDays({startDate}, {endDate}) < 7"

n. **Sum:**
Description: sum(param1: number, param2: number, ...): number Returns the sum of passed numbers.
Example: "expression": "sum({total1}, {total2})"

o. **Max:**
Description: max(param1: number, param2: number, ...): number Returns the maximum of passed numbers.

Example: "expression": "max({total1}, {total2})"

p. **Min:**
Description: min(par1: number, par2: number, ...): number Returns the minimum of passed numbers.
Example: "expression": "min({total1}, {total2})"

q. **Avg:**
Description: avg(par1: number, par2: number, ...): number Returns the average of passed numbers.
Example: "expression": "avg({total1}, {total2}, {total3})"

r. **sumInArray:**
Description: sumInArray(questionName: expression, dataFieldName: string, filter?: expression): number

Returns the sum of numbers taken from a specified data field. This data field is searched in an array that contains a user response to a Multi-Select Matrix, Dynamic Matrix, or Dynamic Panel question. The optional filter parameter defines a rule according to which values are included in the calculation.

The following code sums up values from a "total" matrix column but includes only the rows where a "categoryId" column equals 1:

Here,
matrixdynamic = question title or name
Total = column name
categoryId = column name.

Example: "expression": "sumInArray({matrixdynamic}, 'total', {categoryId} = 1)"

9. **Conditional Logic and Dynamic Texts:**

This section describes how to implement custom conditional logic and add dynamic texts to forms. These features allow you to personalize forms by dynamically updating content and applying logic based on user inputs or predefined conditions.

a. **Dynamic Texts:**

Dynamic texts use placeholders to display content that updates in real-time based on user responses or calculated values.
They can be applied to:

- **Titles and Descriptions:** Surveys, pages, panels, and questions.
- **HTML Properties:** Attributes like completedHtml, loadingHtml, and more.
- **Expressions:** Used in logic or calculated values.

- **Placeholders in Dynamic Texts:**
  Question Values: Refer to question responses using {questionName}. Example:

```
"html": "<p>Hello, {firstName} {lastName}!</p>"
```

- **Variables:** Dynamically set and get variables using setVariable() and getVariable() methods. Example:

```
"html": "&copy; 2015-{currentyear}"
```

- **Calculated Values:** Dynamically compute values using expressions in the calculatedValues array. Example:

```
{
  "name": "fullname",
  "expression": "{firstName} + ' ' + {lastName}"
}
```

- **Accessing Array Values:** Certain question types store multiple values. Use dot notation or indexes to access specific values:

```
Multiple Textboxes: {questionName.itemName}
Dynamic Panel:
{dynamicPanelName[index].questionName}
Checkboxes: {checkboxQuestionName[index]}
```

- **Dynamic HTML:** users can configure different HTML content based on conditions using completedHtmlOnCondition.
  Example:

```
"completedHtmlOnCondition": [
  { "expression": "{score} > 80", "html": "<p>Excellent!</p>"
},
  { "expression": "{score} > 50", "html": "<p>Good Job!</p>"
}
]
```

b. **Conditiuonal Logic:**

Conditional logic dynamically updates the form's behavior or content based on user responses. It uses expressions to define conditions.

- **Using Expressions:** Expressions evaluate conditions and perform calculations dynamically at runtime.
  Examples:

```
Boolean: {age} >= 18 and {country} = 'USA'
Numeric: sum({question1}, {question2})
String: iif({score} > 50, 'Pass', 'Fail')
```

- **Supported Operators:** Expressions support various logical, comparison, and arithmetic operators:

Operator	Description	Expression example
empty	Returns true if the value is undefined or null.	"{q1} empty"
notempty	Returns true if the value is different from undefined and null.	"{q1} notempty"
|| / or	Combines two or more conditions and returns true if any of them is true.	"{q1} empty or {q2} empty"
&&" / and	Combines two or more conditions and returns true if all of them are true.	"{q1} empty and {q2} empty"
! / negate	Returns true if the condition returns false, and vice versa.	!{q1}

<= / lessorequal	Compares two values and returns true if the first is less or equal to the second.	"{q1} <= 10"
>= / greaterorequal	Compares two values and returns true if the first is greater or equal to the second.	"{q1} >= 10"
#ERROR!	Compares two values and returns true if they are loosely equal (that is, their type is disregarded).	"{q1} = 10"
!= / <> / notequal	Compares two values and returns true if they are not loosely equal.	"{q1} != 10"
< / less	Compares two values and returns true if the first is less than the second.	"{q1} < 10"
> / greater	Compares two values and returns true if the first is greater than the second.	"{q1} > 10"
#ERROR!	Adds up two values.	"{q1} + {q2}"
-	Subtracts the second value from the first.	"{q1} - {q2}"
*	Multiplies two values.	"{q1} * {q2}"
/	Divide the first value by the second.	"{q1} / {q2}"
%	Returns the remainder of the division of the first value by the second.	"{q1} % {q2}"
^ / power	Raises the first value to the power of the second.	"{q1} ^ {q2}"
*= / contains / contain	Compares two values and returns true if the first value contains the second value within it.	"{q1} contains 'abc'"
notcontains / notcontain	Compares two values and returns true if the first value doesn't contain the second value within it.	"{q1} notcontains 'abc'"
anyof	Compares a value with an array of values and returns true if the value is present in the array.	"{q1} anyof ['value1', 'value2', 'value3']"
allof	Compares two arrays and returns true if the first array includes all values from the second.	"{q1} allof ['value1', 'value2', 'value3']"

#	Disables type conversion for a referenced value (e.g., string values "true", "false", "123" won't be converted to corresponding Boolean and numeric values).	"{#q1}"

c. **Variables and Calculated Values:**

- **Variables:** Variables are set and retrieved programmatically.

```
Set a Variable: survey.setVariable("currentYear", 2023);
Get a Variable: survey.getVariable("currentYear");
```

- **Calculated Values:** These are defined in the calculatedValues array of the JSON schema and update dynamically when their dependencies change. Example:

```
"calculatedValues": [
 { "name": "totalScore", "expression": "sum({score1}, {score2})"
}
]
```

d. **Practical Examples:**

- **Conditional Visibility:** Show a question only if a specific condition is met.

```
{
  "name": "age",
  "type": "text",
  "visibleIf": "{age} >= 18"
}
```

- **Dynamic HTML Based on User Inputs:** Change completion message based on survey results.

```
"completedHtmlOnCondition": [
  { "expression": "{score} >= 90", "html": "<p>Excellent!</p>" },
  { "expression": "{score} >= 50", "html": "<p>Good Job!</p>" }
]
```

- **Referencing Values from Arrays:** Access specific selections or responses.

```
{
  "html": "Selected Option: {checkboxQuestionName[0]}"
}
```

- **Combining Variables and Calculated Values:**

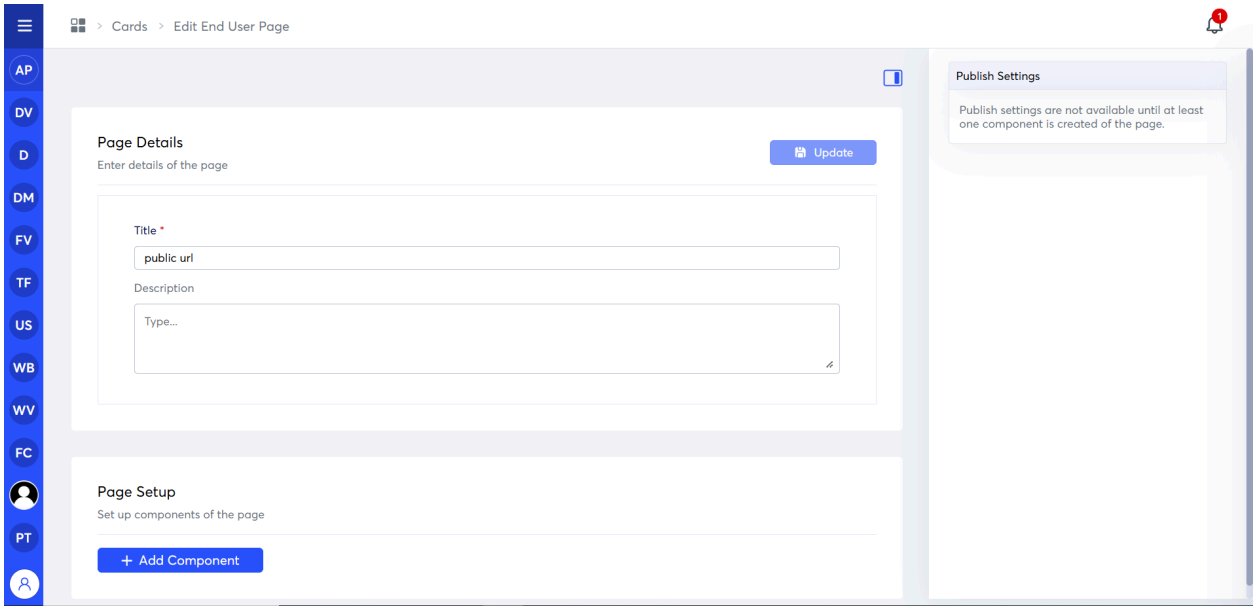
```
"calculatedValues": [
  { "name": "total", "expression": "sum({item1}, {item2})" }
],
"html": "<p>Total: {total}</p>"
```

10. Publishing Forms as Public:

This section provides a step-by-step guide for creating and publishing forms and apps in the system. There are two ways to create a public URL for a form: using the **Form V2 Component** or using **Custom JSON**. Additionally, we provide steps for publishing an app that hosts the form.

a. Publishing a Form Using "Form V2" Component

- Create a Page:
  - Navigate to **Apps & Pages** and click on **Pages**.
  - Create a new page by clicking **+ Create Page**.
  - Open the newly created page.



- Add a Form V2 Component:
  - Click on **Add Component**.
  - From the component type dropdown, select **Form V2**.
  - Choose the desired form to publish.
  - Provide a title for the form in the **Section Title** field.
- Optional settings:
  - **Submission ID**: Allows viewing submission data for the form.
  - **Edit Data Checkbox**: Enables users to edit form data.
  - Click **Save**.
- Preview the Form:
  - After adding the component, click **Preview** to view the form on the page.
- Publish the Page:
  - Open the **Menu Settings** on the right.
  - Set **Menu Visibility** to "Show" for the page to appear in the sidebar menu.
  - Define the **Menu Label** (name of the page in the menu).
  - Set the **Path URL** (e.g., /test-form/customer-feedback).
  - Mark **Active** as "True" to make the page active.
  - Assign appropriate permissions to users who need access to the page.
  - Click **Save**.
- Publish the Form:
  - Click the **Publish** button to make the form live on the service layer.

b. Publishing a Form Using "JSON"
   - Create a Page and Add a JSON Component:
     - Follow the steps to create a page.
     - Select **JSON** as the component type while adding a new component.
   - Add JSON Code:
     - Paste the JSON configuration in the **Component JSON** field. Ensure the formDid in the JSON matches the form's ID. Example JSON:

{"id":"containerBlock","type":"Block","permission":"","title":"","subTitle":"","helpText":"","style":{},"className":"page-container","active":true,"attr":{"description":""},"components":[{"id":"fo06","type":"Block","permission":"a-permission-string","title":"Users","subTitle":"Users","helpText":"test","style":{},"active":true,"className":"page-default-height","attr":{},"components":[{"id":"blockId0.3054341829359801","type":"Block","permission":"","title":"","subTitle":"","helpText":"","style":{},"className":"","active":true,"attr":{},"components":[{"id":"formViewer","type":"pageFormViewer","permission":"","title":"","subTitle":"","helpText":"","style":{},"className":"","active":true,"attr":{"edit":true,"formDid":"4bef361cf8b14f089a79d944db8e6aaf","submissionDid":""},"components":[]}]}]}]}

   - Save and Preview:
     - Click **Save** to add the component to the page.
     - Click **Preview** to ensure the form displays correctly.
   - Publish the Form:
     - Follow the same publishing steps as in the **Form V2 Component** section.

c. Publishing an App
   - Create an App:
     - Navigate to **Apps & Pages** and click on **Apps**.
     - Click **Create App**.
     - Fill in the following details:
       - **App Title**: Name of the app (e.g., "Form Test App").
       - **Path**: Specify the app's path (e.g., /form-test-app).
       - **Maintenance Mode**: Enable or disable based on the app's development status.
       - **Show the App**: Set to "True" to display the app in the sidebar.
       - **Core**: Set to "True" if this is a core app.
       - **Priority**: Assign a priority number.
       - **Description**: Add a description of the app.
       - **App Logo**: Optionally, upload a logo.
   - Assign Pages to the App:
     - Scroll to the **All Pages of the App** section at the bottom of the page.

- Click the **+** button to add pages.
- Select the desired page(s) and click **Save**.

d. Enable Public Visibility:
   - Enable the **Public Visibility** option.
   - Provide the following:
     - **App Name**: The app name (e.g., "Form-Test-App").
     - **Landing URL**: Define a landing URL for the app (e.g., /test-form/customer-feedback-json).
     - **Users**: Assign specific users who can access the app.

e. Save and Publish:
   - Click **Save** to finalize the app.
   - Click the **Publish** button to make the app live.

f. Access Public Site URL:
   - After publishing, a **Public Site URL** and an **Embed Code** will be generated. These can be copied for sharing or embedding.

g. Notes:
   - The public URL for forms works only if the app or page is published.
   - Permissions must be assigned to ensure visibility for the intended users.
   - JSON configurations must match the required schema and the form ID.

11. Form Permissions Management:

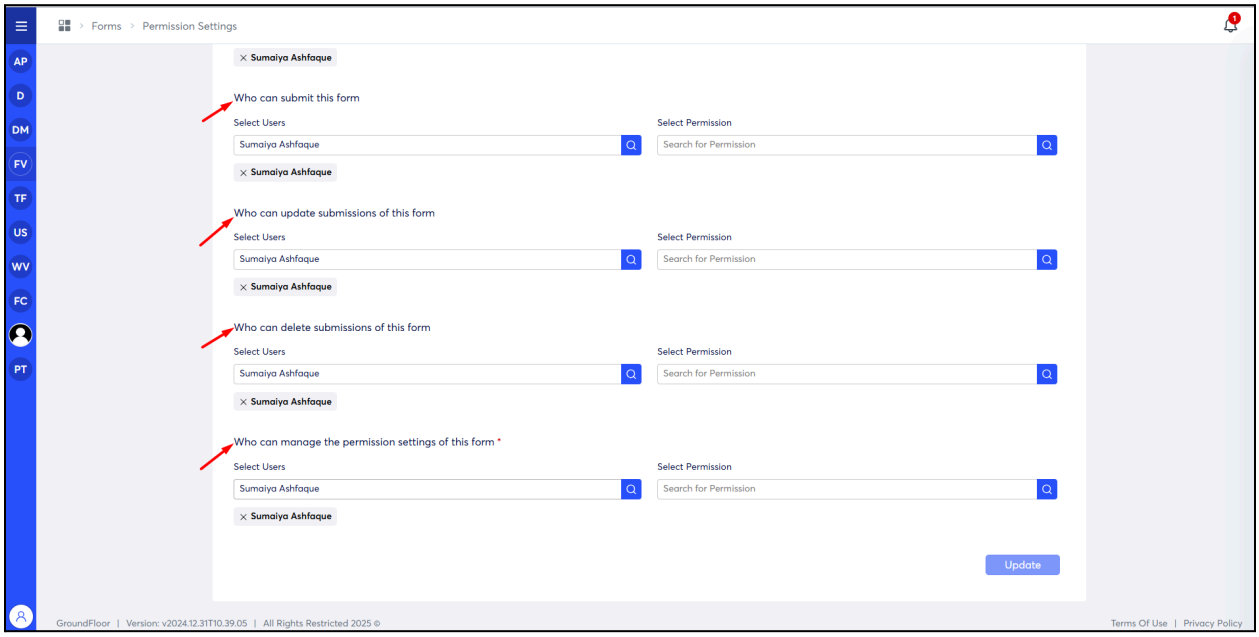
The Form Permissions page allows administrators to manage user permissions for specific forms. By default, the user who created the form is granted full access to all permissions. Additional users and permissions can be assigned through the interface.

a. Permissions Overview

Below is the list of permissions available for forms:

- **Who can view this form:** Users assigned here can view the form.
- **Who can edit this form:** Users assigned here can make changes to the form structure and content.
- **Who can delete this form:** Users assigned here can delete the form entirely.
- **Who can view submissions of this form:** Users assigned here can access the data submitted through the form.
- **Who can submit this form:** Users assigned here can submit responses to the form.
- **Who can update submissions of this form:** Users assigned here can edit existing submissions.

- **Who can delete submissions of this form:** Users assigned here can delete submission entries.
- **Who can manage the permission settings of this form:** Users assigned here can modify the permissions for this form.



b. Steps to Assign Permissions

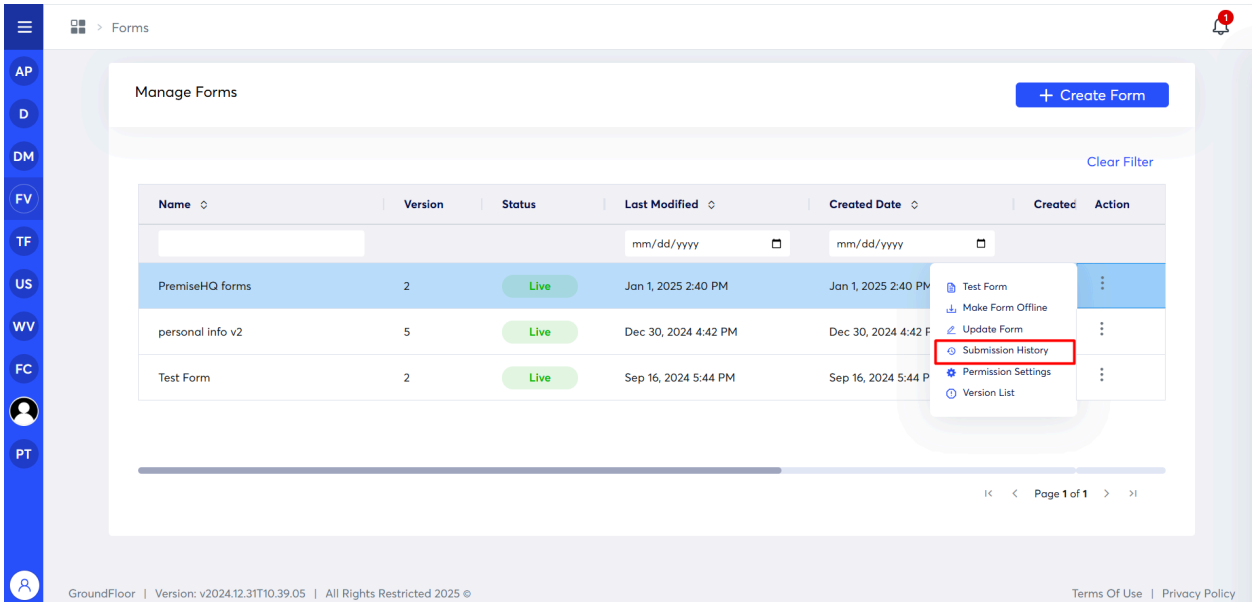
- **Assign Users:** In the **Select Users** dropdown, search for and select users who need access to the form.
- **Assign Permissions:** Use the **Select Permission** field to specify the type of access each user should have. Permissions are
- pre-defined in the system.
- **Default Creator Permissions:** The form creator automatically has access to all permissions. This cannot be revoked unless managed by another user with appropriate permissions.
- **Save Changes:** After assigning users and permissions, click the **Update** button at the bottom of the page to save the changes.

c. Additional Notes

- Only users with the "Manage Permission Settings" privilege can modify the permissions for a form.
- Permissions are tied to the user management system, ensuring that only authorized users can access or modify form data.

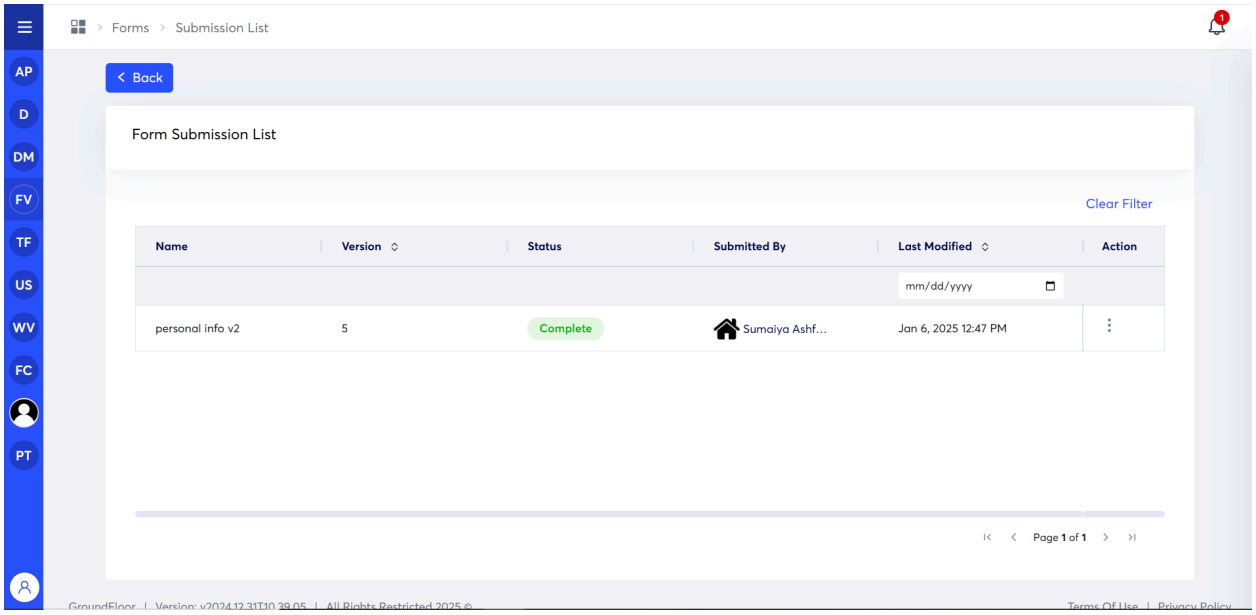
12. Form Submission History Page:

The **Submission History Page** provides users with a comprehensive interface to view, manage, and interact with form submissions. It includes features for reviewing, editing, updating, and deleting submissions.



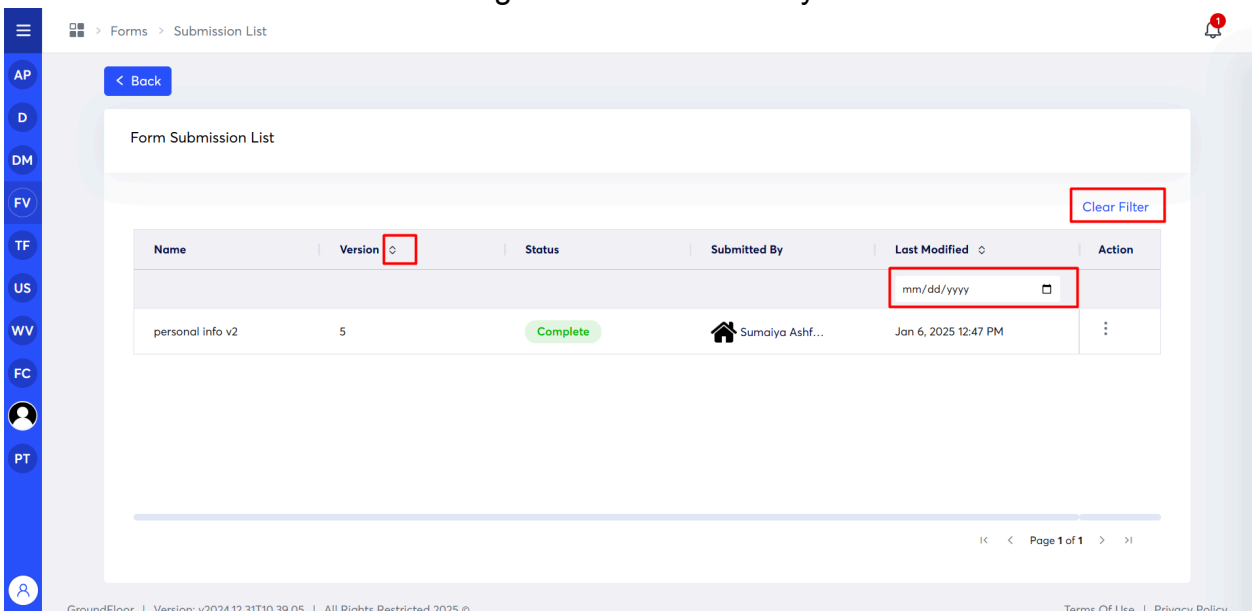
Here is the Key Features of the submission history page.

a. **Submission Overview Table Columns**:
   - **Name**: The name of the form.
   - **Version**: The version number of the form used for the submission.
   - **Status**: Indicates the submission state, such as:
     - **Draft**: A submission that is saved but not completed.
     - **Complete**: A submission that has been finalized and submitted.
   - **Submitted By**: The user who made the submission (displayed with their name and avatar).
   - **Last Modified**: The timestamp of the last modification to the submission.
   - **Action**: Provides additional functionality, such as viewing, editing, or deleting submissions.



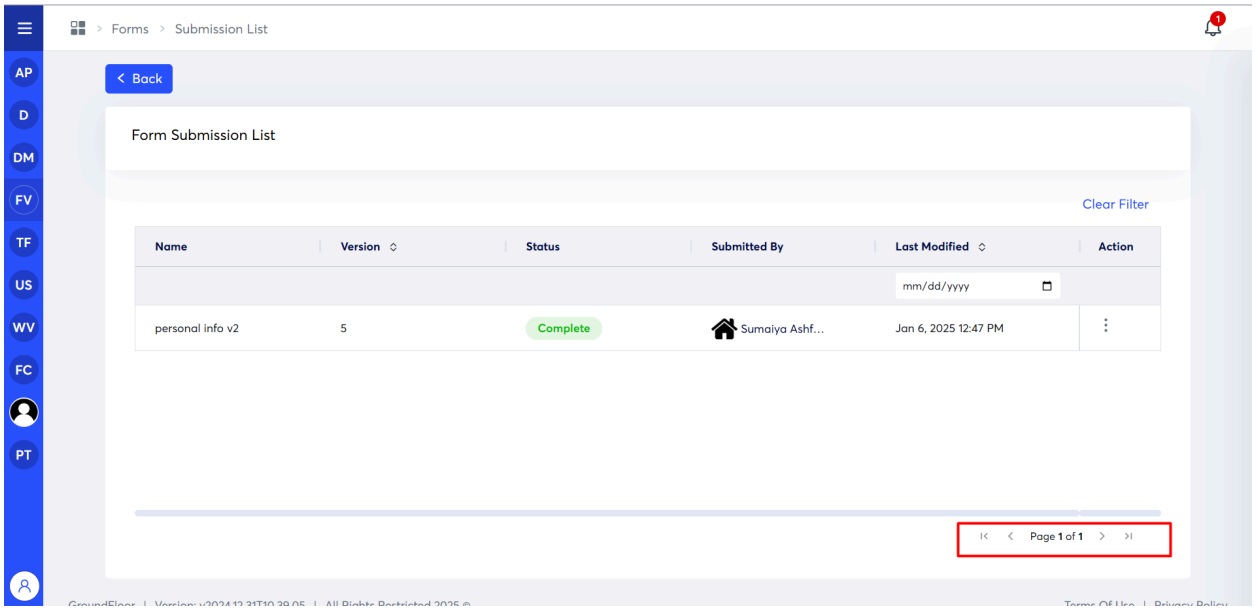
b. Filters and Sorting

- **Clear Filter**: Allows users to reset applied filters to view all submissions.
- **Sort Options**: Columns such as **Version** and **Last Modified** can be sorted to organize submissions by their values.



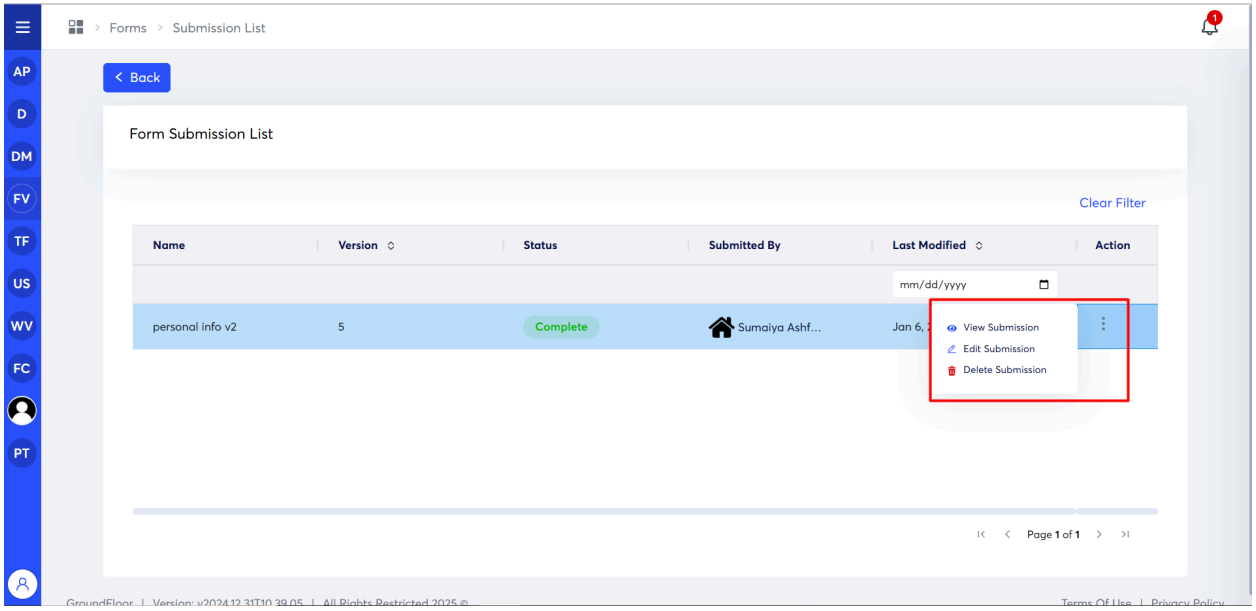
c. Pagination

- The page supports pagination to handle large datasets efficiently, with options to navigate through pages.



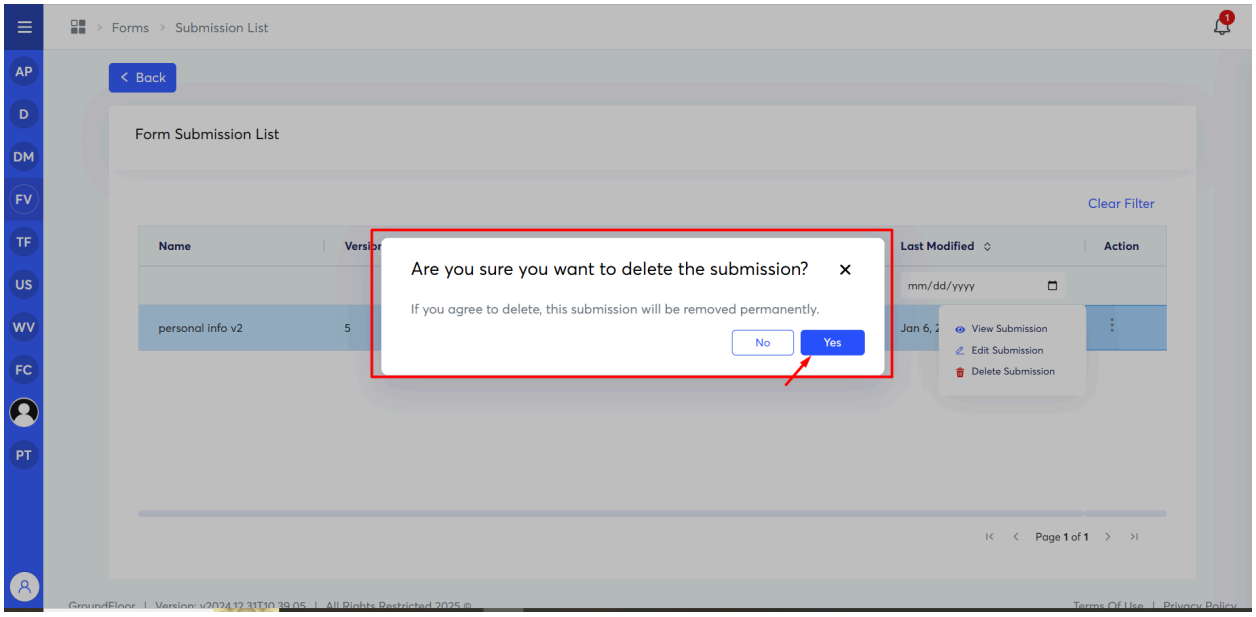
d. Actions

- **View Submission Details**: Users can click on the submission name or use the action menu (three vertical dots) to open the full details of a submission, which includes all form responses and metadata.
- **Edit and Update Submissions**: Users with appropriate permissions can modify a submission by selecting the **Edit** option in the action menu. Changes made can be saved to update the submission.



e. Delete Submissions:

- The **Delete** option in the action menu allows users to permanently remove a submission from the system.
- A confirmation prompt ensures accidental deletions are avoided.

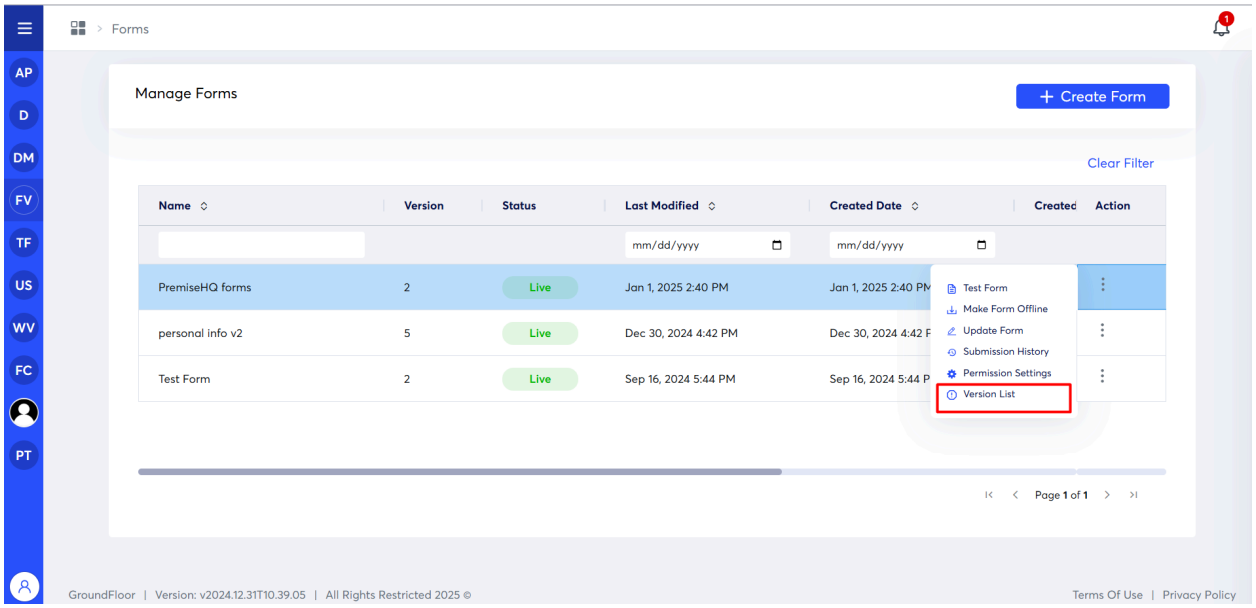


f. Use Case Scenarios

- **Review Submissions**: View the details of all submissions to analyze responses and ensure accuracy.
- **Update Data**: Modify existing submissions to correct errors or make necessary updates.
- **Manage Submissions**: Delete outdated or incorrect submissions to maintain clean data records.
- **Track Progress**: Use the status column to identify incomplete submissions (Draft) and take necessary action.

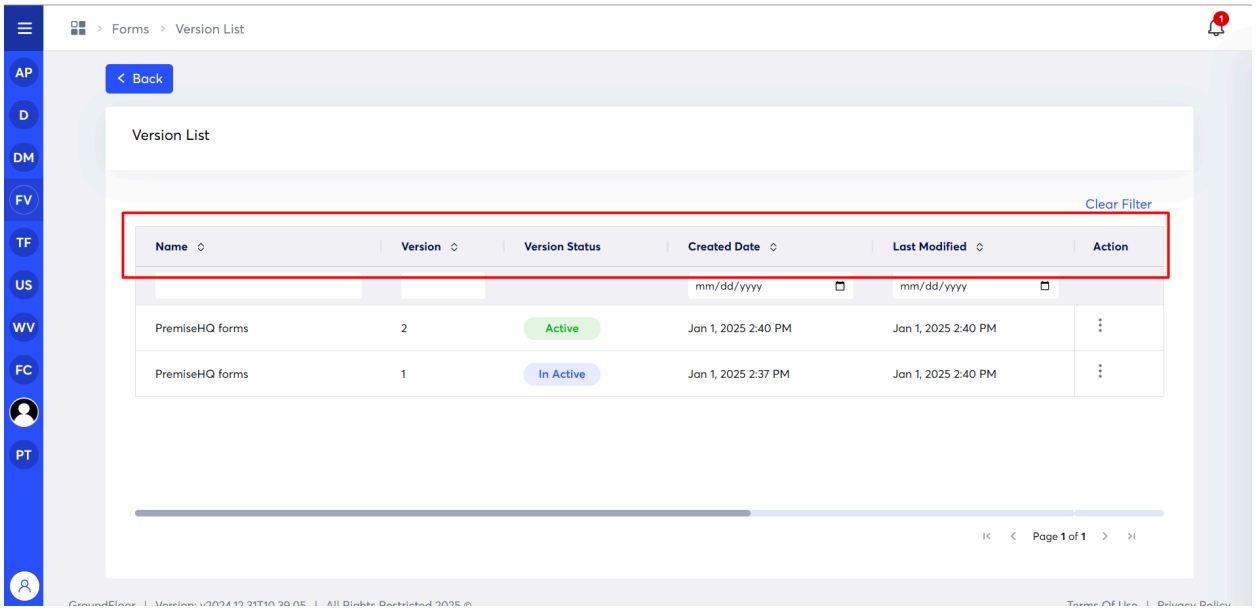
13. Form Version List Page:

The Version List Page in the forms application provides an organized view of all versions of a specific form, enabling users to track and manage the different iterations of their forms.

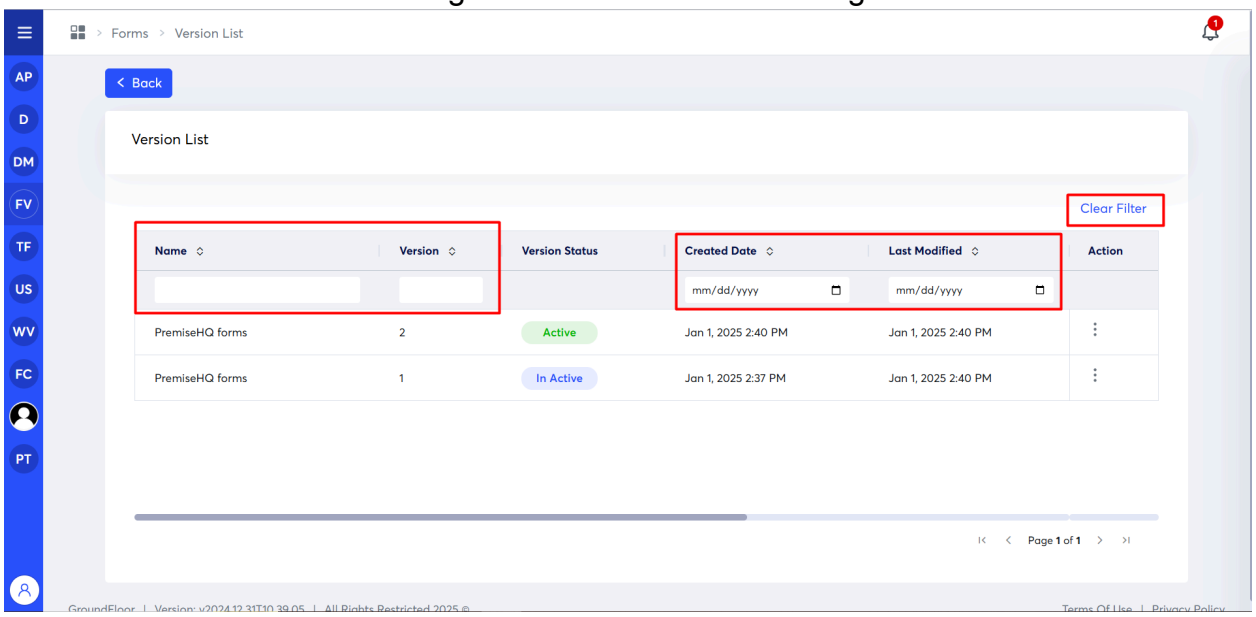


Key Features:

a. **Version Overview Table**
   - Columns:
     - Name: The name of the form.
     - Version: The version number of the form, incremented with each update.
     - Version Status: Indicates the current state of each version:
       - Active: The version currently in use.
       - Inactive: Previous or deprecated versions.
     - Created Date: The timestamp when the version was created.
     - Last Modified: The timestamp when the version was last updated.
     - Action: Provides additional functionality for managing the version.



- Filters and Sorting
  - Filter Options: Users can filter versions based on name, version status, or date ranges for created/modified timestamps.
  - Sort Options: Users can sort columns (e.g., version, dates) to organize the list for better navigation.



- Pagination:
  - The page supports pagination to manage long lists of versions efficiently.

b. Actions

Users can click on the Action Menu (three vertical dots) to access detailed information about a specific version, including its configuration and usage.

- Preview

- Purpose: Allows users to view the form as it will appear to the end-user (Read only view).
- Details:
  - Redirected to the preview of the selected form.
  - Helps users verify the form's design and layout before activation or publishing.
  - Ensures that any changes or updates made to the form version appear as intended.

- Editor View:
  - Purpose: Opens the form in edit mode, allowing users to modify the structure, fields, or logic of the selected version.
  - Details:
    - Redirects to the form editor interface specific to the selected version.
    - Users can add, remove, or update form fields, validation rules, and other configurations.
    - Saves updates as a new version or overwrites the current version, depending on the user's permissions and system design. These functions ensure users have complete control over form iterations, improving their ability to manage updates and ensure high-quality forms.

14. Translation Settings:

The **Translation Settings Page** in your **Forms V2 Application** provides a centralized interface for managing multilingual content for forms. Users can set a local or global language for each question. The default language is English. The user can change the language and use it in the forms. Here's an overview of the page's features and functionality:

a. Translation Sections:

The page allows users to translate specific elements of a form, including:

- Form Title.
- Form Description.
- Page Title.
- Page Description.
- “Thank you” page markup.
- Markup to show if the user already filled this form.
- Markup to show while the form is loading.
- Questions: Titles and labels for each question.
- HTML Markup: Custom HTML content that may include links or instructions.

Each translatable field has an editable translation area for every selected language.

b. **Language Settings Panel:**
   - **Add Languages**:.
     - Users can add new languages using the **plus (+)** button in the right-hand panel.
     - The dropdown on the **Language Settings** panel displays a wide range of languages available for selection.
     - Users can type in the search bar or scroll through the list to select additional languages.
     - Once a language is added, all fields become available for translation into that language.
   - **Default Language**:
     - The default language is pre-selected (e.g., **English**).
     - Other added languages are listed below with the option to manage or remove them.

c. **Steps to Adding and Managing Translations**
   - **Add a New Language**:
     - Click the **plus (+)** button in the language settings panel.
     - Select the desired language from the list.
   - **Input Translations**:
     - For each form element, provide translations under the selected language's column.
   - **Preview Changes**:
     - Switch to the **Preview** tab to check how the translated form looks.
   - **Save and Publish**:
     - Click **Update** to save changes.
     - Use the status toggle (if applicable) to make the updated form live.

This page ensures accessibility for a global audience by supporting multilingual forms with ease.

15. ETC 0
16. ETC 1
17. ETC 2
18. ETC 3
19. ETC 4