

KAIZEN Training Material

(Combined)

Revision History

Version	Date	Description	Author
1.0	4/3/2025	Training Material for both Designer and Developer Tutorials based on KAIZEN v24.10.1	Amanda, Haryanto, Wei Han, Yong Heng, Yong Li
2.0	28/3/2025	Enhance tutorial steps	Amanda, Haryanto, Wei Han, Yong Heng, Yong Li
2.2	26/5/2025	Enhance Tutorial 1 Login Page to be mobile responsive	Amanda

Content Page

Introduction.....	8
Designer Materials	12
Tutorial 0: Foundational Concepts	12
Studio Console.....	13
App Designer	14
Tutorial 1: Creating a Login page	25
Practical 1.1: Add Background Image	27
Practical 1.2: Create Layout	31
Practical 1.3: Add 4 Text components	37
Practical 1.4: Add Form Template	41
Tutorial 2: Creating the Landing Page.....	47
Practical 2.1: Create Initial Layout.....	49
Practical 2.2: Create heading	51
Practical 2.3: Add Image Background to root page.....	53
Practical 2.4: Add Button Group	57
Practical 2.5: Create responsive card.....	60
Tutorial 3: Creating the Listing Page (Optional)	81
Practical 3.1: Create Initial Layout	82
Practical 3.2: Create Heading and Breadcrumb.....	83
Practical 3.3: Create Search form.....	85
Practical 3.4: Create List header	103
Practical 3.5: Create Table	108
Tutorial 4: Creating a Form with Steps.....	135
Introduction: Form Fields	138
Introduction: Form Validation.....	139
Practical 4.1: Setting up the Form	140
4.1.1 Creating the Form skeleton.....	140
4.1.2 Form Setup Complete.....	160
4.1.3 The Source Code Panel	161
Practical 4.2: Form Fields and Form Validation	165
4.2.1 Context.....	165
4.2.2 Objective	167
4.2.4 Course Info Form Validation	175
4.2.5 Form Completion	178
Practical 4.3: Create the 'Instructor Particulars' Form.....	179
4.3.1 Context.....	179
4.3.2 Objective	179
4.3.3 Requirements	180
4.3.4 Hints	180

4.3.5 Expected Result	181
Practical 4.4: 'Class Location' Form	182
4.4.1 Context.....	182
4.4.2 Objective	182
4.4.3 Requirements	183
4.4.4 Creating a Field with 2 inputs.....	184
4.4.5 Styling for the Inputs.....	188
4.4.6 Validation for both input fields.....	191
4.4.8 Form Completion	193
Practical 4.5: Class Schedule Form.....	194
4.5.1 Context.....	194
4.5.2 Objective	194
4.5.3 Requirements	195
4.5.4 Custom Date Validation	196
4.5.5 Creating the Form field that can add new Rows.....	201
4.5.6 Form Completion	213
Practical 4.6: Cascade Dropdown	214
4.6.1 Context.....	214
4.6.2 Requirements	214
4.6.3 Adding the Logic.....	214
Practical 4.7: Setting field values using Form Variables (Optional)	219
4.7.1 Context.....	219
4.7.2 Requirements	219
4.7.3 Adding the 'MyInfo' Button.....	220
4.7.4 Adding the logic	223
Practical 4.8: Binding it all together	225
4.8.1 Adding the code.....	225
4.8.2 Bind currentStep variable to Step Component	226
4.8.3 OnClick Events for the Next buttons	227
4.8.4 Set conditions for the Forms to be visible	229
Practical 4.9: Modal Dialog Box.....	233
4.9.1 Steps for Creating a Modal Dialog	233
4.9.2 Adding the logic for the Modal Dialog	235
And Finally.....	243
Additional Information.....	245
Tutorial 5: Tables, Sorting and Pagination.....	247
Practical 5.1: The Table Component	248
5.1.1 Setting up the Table Component	248
5.1.2 Setting the Table Headers	251
5.1.3 Adding a Static Dataset to the Table.....	253
Practical 5.2: Table Sorting & Pagination	258

5.2.1 Setup.....	258
5.2.2 Sorting.....	259
5.2.3 Pagination	263
5.2.4 Page Sizing	270
Practical 5.3: Download functionality (Optional).....	273
5.3.1 Add a download button	273
5.3.2 Add the download code	274
5.3.3 Event bind the download button.....	274
Tutorial 6: Charts and Graphs.....	276
Practical 6.1: Adding Graph Components.....	276
6.1.1 Setup Steps.....	276
6.1.2 Adding a Bar Graph.....	278
6.1.3 Sending Data to the Graph	278
6.1.4 Adding a Pie Chart	280
6.1.5 Sending Data to the Graph	280
Tutorial 7: Using External JS Libraries.....	283
Practical 7.1: Importing a UUID JS Library (Demonstration)	284
7.1.1 Setup Steps.....	284
7.1.2 Adding the Library	287
7.1.3 Using the library.....	289
Tutorial 8: Theme Designer	290
Practical 8.1: Customize new theme	291
Practical 8.2 Apply theme to your application	302
Additional Information.....	304
Tutorial 9: Preview Mode.....	306
Practical 9.1: Preview Mode and Collaborative Commenting.....	306
Practical 9.2: Navigator Creation / Page Linking.....	308
9.2.1 Navigator Creation.....	309
9.2.2 Customize Navigator Menu.....	311
9.2.3 Page Linking.....	315
Practical 9.3: Page Tagging for External Review (Demonstration).....	317
Tutorial 10: Page Improvement.....	319
Practical 10.1: Page Analysis with Page Scan (Demonstration)	319
Practical 10.2: Mobile Responsive UI (Demonstration)	321
Practical 10.3: Translation using i18n (Demonstration).....	323
Tutorial 11: Page Import From Template.....	326
Tutorial 12: Main and Micro Applications.....	329
Practical 12.1 Export and Import Pages to Main/Micro Apps (Optional).....	331
Practical 12.2 Add Action Page Type to Main App (Optional)	334
Tutorial 13: Build screen skeleton.....	342
Layout: Section vs Block	344

Layout: Block vs Cell	351
Block VS Cell.....	355
Example: Webpage walkthrough example.....	356
Step 1: Create layout.....	357
Step 2: Divide cell horizontally for left block.....	357
Step 3: Split cell into two	357
Step 4: Add a card component	358
Step 5: Add 3 Card components.....	358
Step 6: Add a table component.....	358
Step 7: Add components to the right block	360
Example	361
Developer Materials	362
Architecture	364
Tutorial 14: Git Setup & Creation of New Branch	366
Practical 14.1: Connect with Gitlab.....	366
Practical 14.2: Initialize Git Repository	370
14.2.1 Frontend - Initialize Git Repository.....	370
14.2.2 Backend - Initialize Git Repository	372
Practical 14.3: Create and Switch to new branch	375
14.3.1 Frontend Git Branch Management.....	375
14.3.2 Backend Git Branch Management (Optional).....	378
Tutorial 15: Master Code Integration	380
Practical 15.1: Integrating Master Code.....	380
Practical 15.2: Multi language support (Optional)	384
Tutorial 16: Custom Login Page Integration	387
Practical 16.1: Login Page Integration	387
Tutorial 17: Database Designer.....	389
Practical 17.1: Setup Database and tables	389
Connect to the PostgreSQL Server	389
Creating a New Table	391
Adding Rows to Table	395
Benefits of Database designer.....	399
Tutorial 18: Service / API Designer	401
Practical 18.1: Creating Service and Controller	401
Navigating the Service / API Designer	401
Adding a Service	402
Adding a Database	405
Adding a Controller.....	407
Adding Datasource & Configure API Endpoints	408
Tutorial 19: Datasource API Binding	412
Practical 19.1: Binding API to Table (GET).....	412

Practical 19.2: Binding API to Form submission (POST)	420
Adding DataType & Datasource	420
Import Remaining APIs.....	422
Binding to FormSubmission.....	423
Binding POST Form using “Encode”.....	427
Tutorial 20: Error Handling.....	430
Practical 20.1: Redirect to custom error page.....	430
Tutorial 21: Workflow Designer.....	432
Practical 21.1: Using default workflow (Claim-based routing)	433
Practical 21.2: Designing and Deploying custom workflow	437
Practical 21.3: Workflow observability	457
Tutorial 22: Code Generation (Backend).....	461
Practical 22.2: Generating and Running Backend Code	461
Generate Code.....	461
Understanding Generated Backend Code Structure	463
Tutorial 23: Profile for local development	470
Practical 23.1: Running be-services	471
Practical 23.2: Update Profile feature	473
Tutorial 24: Customize Privilege for Page Permission	477
Tutorial 25: Code Generation (Frontend)	481
Practical 25.1: Generating and Running Frontend Code	481
Installation of Necessary Prerequisites:	481
Generate Code.....	481
Running nginx	485
Running the generated code	486
Triggering Error	487
Tutorial 26: Push and Merge Code to Git	489
Practical 26.1: Frontend Git Management	489
Create New Tag and Git Push.....	489
Merge Code to Main Branch + Push Main Branch	492
Practical 26.2: Backend Git Management (Optional)	496
Create New Tag and Git Push.....	496
Merge Code to Main Branch + Push Main Branch	497
Additional Information - Branch Management.....	500
Tutorial 27: Job Scheduler	505
Practical 27.1: Creating and Running Job (Demonstration)	506
Tutorial 28: IAM	513
Court Case Example: How "Subject" Works	521
Practical Example: Judge Case Assignment.....	521
Practical 28.1: Assignment of custom menu (Optional)	522
Practical 28.2: Endpoint Assignment (Optional).....	525

Introduction

Welcome to the **KAIZEN Training Program!** We are thrilled to have you embark on this learning journey with us. This comprehensive guide is designed to provide you with the information and resources you need to make the most out of your training experience.

About KAIZEN

KAIZEN is a cutting-edge agile development stack designed for individuals and teams who want to accelerate application development without compromising on flexibility and customization. Our platform is built to simplify the entire software development lifecycle, from ideation to deployment, and subsequently to maintenance.



Key Features

In this training, we will be walking you through the following features:

- **App Designer**

The App Designer feature empowers users to create user-friendly pages effortlessly within a visual workspace. With a drag-and-drop interface, you can design and arrange components intuitively, bringing your project to life without writing complex code. Whether you're building interactive interfaces or structuring layouts, the App Designer streamlines the entire process, making development faster and more efficient.

- **Theme Designer**

The Theme Designer allows for the customization of the application's visual appearance, ensuring a consistent and appealing user experience. It enables designers to apply brand-specific styles across the application, adjusting colors, fonts, and layouts. This feature enhances accessibility and visual appeal for users across different regions.

- **Main and Micro Application**

This feature encompasses the architecture where a central main application coordinates with multiple micro applications. The main app serves as the core, integrating various micro apps that handle specific functionalities, promoting modularity and scalability.

- **Git Setup and Branch Management**

Git Setup and Branch Management ensures smooth collaboration by enabling version control, branching, and code integration. Developers can work on separate features without conflict, while maintaining a clear and organized codebase. This tool supports efficient team workflows and ensures up-to-date code management.

- **Workflow Designer**

The Workflow feature allows for the automation of business processes through predefined steps. Users can design and execute workflows that involve multiple stages and actors, ensuring tasks are completed in a systematic and efficient manner.

- **Master Code**

The Master Code feature provides a centralized system for managing reference data used across the application. It supports multilingual capabilities, allowing for localization and consistent data management.

- **Database Designer**

The Database Designer simplifies database schema creation and management, allowing developers to define tables and relationships with ease. It ensures efficient data organization, accessibility, and scalability for the application. This feature improves performance and helps optimize data storage for future development.

- **Service Designer**

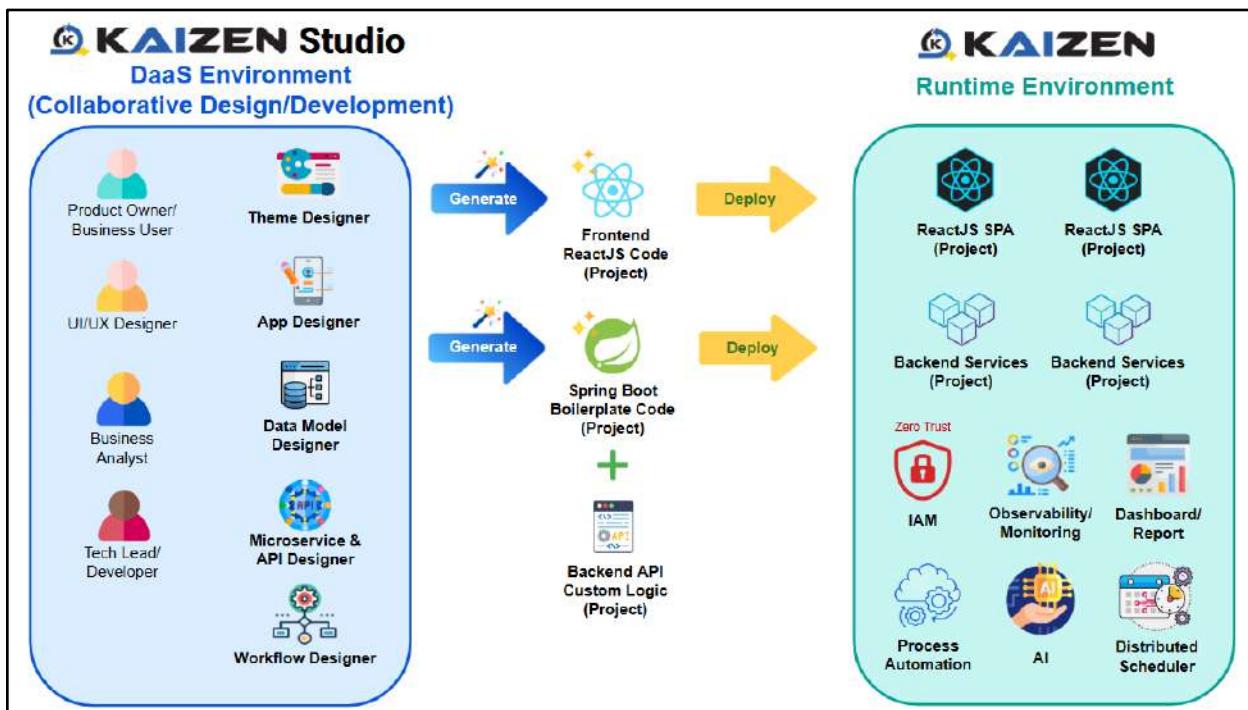
The Service Designer streamlines the creation and management of APIs, connecting application components and external services. It provides an intuitive interface to define endpoints and parameters, making API development faster and more efficient. This feature enhances backend and frontend communication, supporting complex application workflows.

- **Job Scheduler**

The Job Scheduler enables the scheduling and execution of tasks at specified times or intervals. It supports various scheduling strategies, ensuring that tasks are performed efficiently and on time, with detailed configuration options for each job.

- **Code Generation**

The Code Generation feature automates the creation of boilerplate code, reducing development time and errors. It helps in generating consistent and standardized code structures for various components, ensuring best practices and efficiency in the development process.



Getting Ready - Account Setup

Before you dive into the exciting features and possibilities that await you, let's ensure you are fully prepared to make the most of your experience. Follow these steps to get started:

Accounts have been created for all trainees in the Cloud Development Environment as a Service (DaaS) Environment.

DaaS Environment URL: <https://kaizen-daas.toppanecquaria.com/#/login>

User Domain: AGP Designer

Username: username (Example: username@toppanecquaria.com)

Overview of the Training

This training program is designed to equip participants with the essential knowledge and skills to use the KAIZEN platform effectively. It is divided into two main phases: Designer Tutorials and Developer Tutorials, allowing for a comprehensive understanding of application design and development.

Participants will start by learning how to create visually appealing and functional user interfaces in the Designer phase. Then, they will delve into the Developer phase to build robust backend systems and workflows, ensuring the seamless integration of features.

Training Goals:

Designer Phase (Tutorial 1 to 13):

- Understand the fundamentals of creating user-friendly interfaces.
- Learn to design and customize pages such as login, landing, and listing pages.
- Gain expertise in creating forms with steps, tables with sorting and pagination, and visualizing data through charts and graphs.
- Explore advanced features like theme customization, translations, and screen skeleton creation.
- Understand how to structure and manage micro-application architecture.

Developer Phase (Tutorial 14 to 28):

- Master Git setup, branching, and code integration workflows.
- Learn to design custom login pages and manage databases effectively.
- Develop and bind APIs with data sources for seamless connectivity.
- Implement error handling, workflow automation, and job scheduling.
- Explore code generation for backend and frontend development.
- Manage application deployment with local development profiles and Git integration.
- Explore IAM features and assign custom menus, pages, and endpoints.

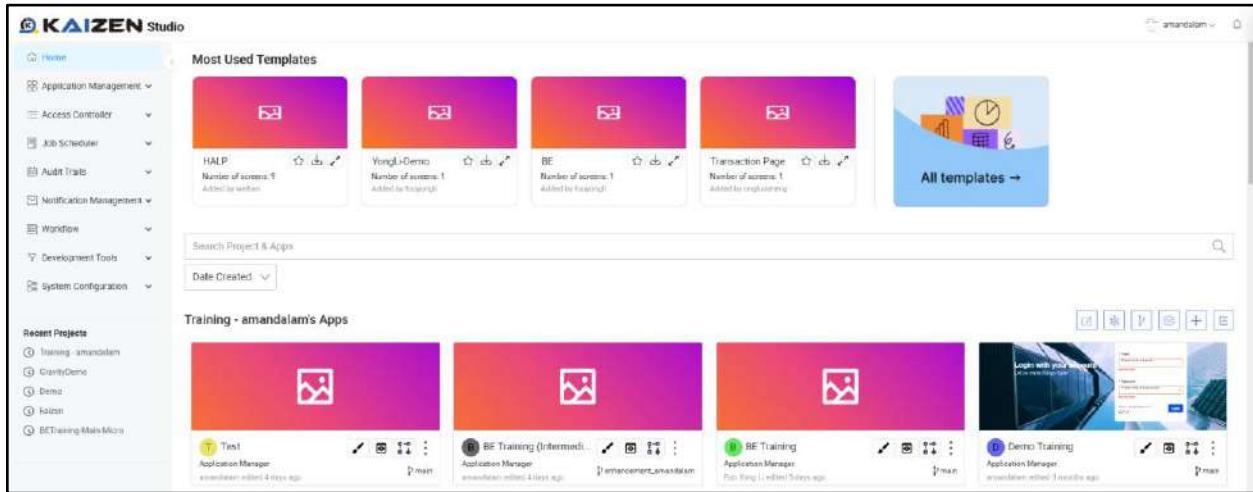
Designer Materials

Tutorial 0: Foundational Concepts

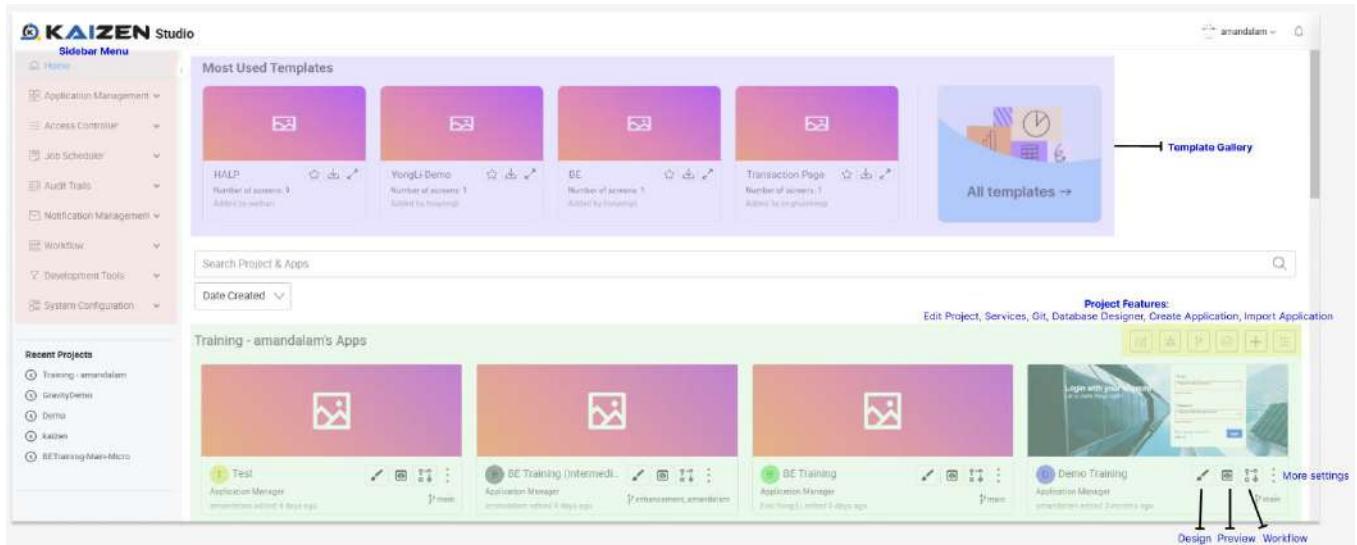
This tutorial covers the following Learning Objectives:

- Understand the basic fundamentals, building blocks, concepts of saving, previewing, and publishing in KAIZEN.

Foundational concepts are the building blocks for working with KAIZEN. In this tutorial, we will explore the platform's core functionalities, such as saving progress, previewing work in progress, and publishing changes to ensure a smooth and efficient development process.



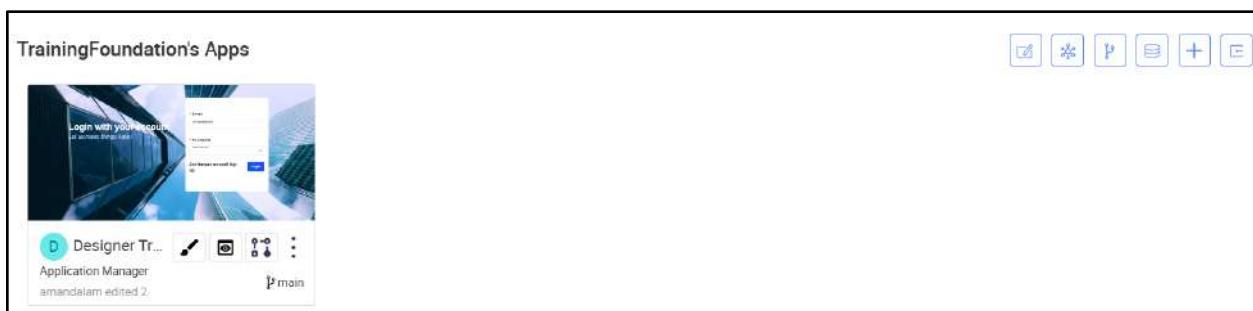
Studio Console



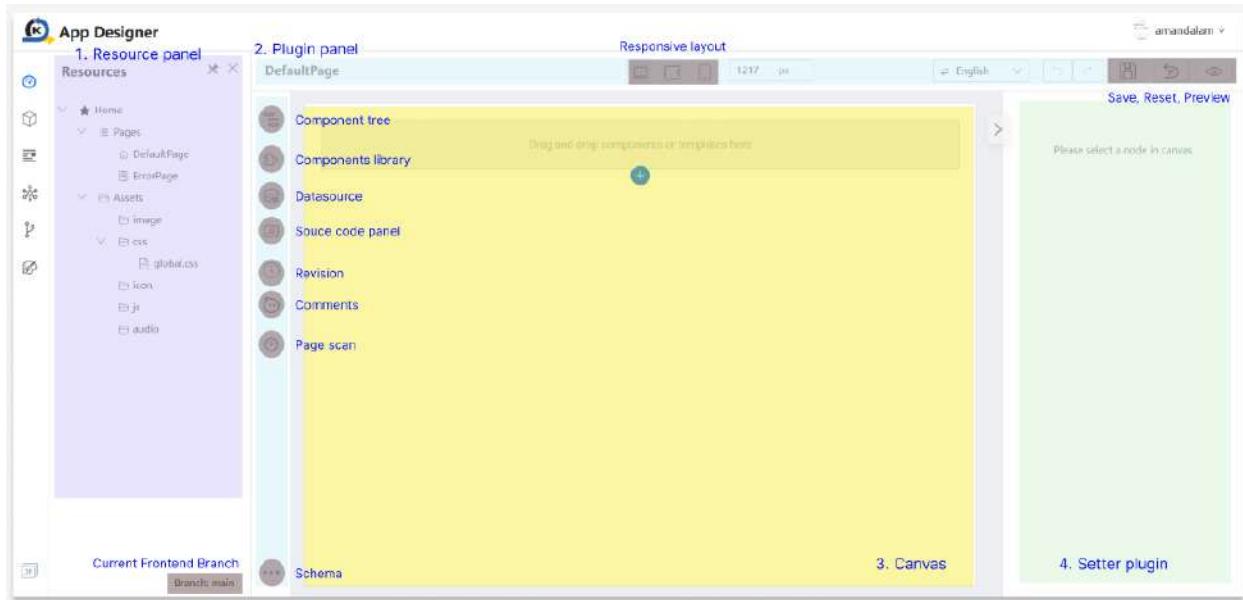
Note that not all the **Sidebar Menu Items** and **Project Features** will be accessible in your trainee account due to Identity and Access Management (IAM) access restrictions. In order to access the features, the respective roles need to be granted to your user account.

In a project scenario, our application offers predefined roles that can be assigned to users, such as **Project Manager**, **Developer**, **UI Designer**, **Internal Viewer**, and **External Viewer**. Each role is tailored with specific access levels, enabling precise control over features and functionalities for managing both applications and the overall project.

For more information on the IAM features, you may refer to [Tutorial 28](#).



App Designer



1. **Resource panel:** The Resource Panel is where you manage and access various pages related to your application.
2. **Plugin panel:** The Plugin Panel is a section that displays and allows you to manage the plugins or extensions integrated into the platform or software. Plugins enhance functionality or provide additional features.
 - **Component Tree:** The Component Tree is a hierarchical view that represents the structure of the components within your project. It visually organizes and displays the relationships between different elements.
 - **Components Library:** The Components Library is a collection of pre-built components that you can easily drag and drop into your application.
 - **Datasource:** Datasource refers to the data connection or repository from which your application retrieves information such as API.
 - **Source code panel:** The Source Code Panel is where you can view and edit the underlying source code of your project. It provides direct access to the code for users who prefer or need to work at the code level.
 - **Revision:** The Revision feature allows you to track changes made to your project over time. It provides a version history, making it easy to revert to a previous state if needed.

- **Comments:** Comments are annotations or notes that you can add to your application. They provide explanations, insights, or instructions for yourself or collaborators.
 - **Page scan:** Page Scan is a tool that analyzes and checks your project or application for potential issues, errors, or improvements. It helps ensure the quality and performance of your work.
 - **Libraries:** Libraries are collections of reusable code or resources that can be imported into your project. They save time and effort by providing pre-built functionalities.
 - **Schema:** Schema defines the structure and organization of your data in JSON format. It outlines the relationships, constraints, and rules governing how data is stored and accessed.
 - **Responsive layout:** Responsive Layout refers to the design approach that ensures your project adapts and looks good on various devices and screen sizes, providing a consistent user experience.
 - **Save Draft, Publish, Reset, Preview:** Save allows you to save your project progress, Reset reverts changes, and Preview enables you to see how your project will appear or function before finalizing.
 - **Page Lock:** Mechanism designed to control and manage collaborative editing in a shared project environment. This ensures that critical or in-progress work on a page is not unintentionally modified or overwritten by others, providing a conflict-free editing process.
3. **Canvas:** The Canvas is the visual workspace where you design and arrange the elements of your project. It's the area where you create and build.
- **Component operation:** Component Operation involves managing and manipulating individual elements or components within your project, such as copying, locking, or deleting them.
4. **Setter plugin:** A Setter Plugin is a tool or extension that allows you to configure or set specific properties or attributes of components within your project.

We will now explore the fundamental concepts behind some key features of the App Designer listed above. These will be used heavily during the creation of your application screens. To

demonstrate these features in action, we have prepared a shared project that all trainees can access - **TrainingFoundation**.

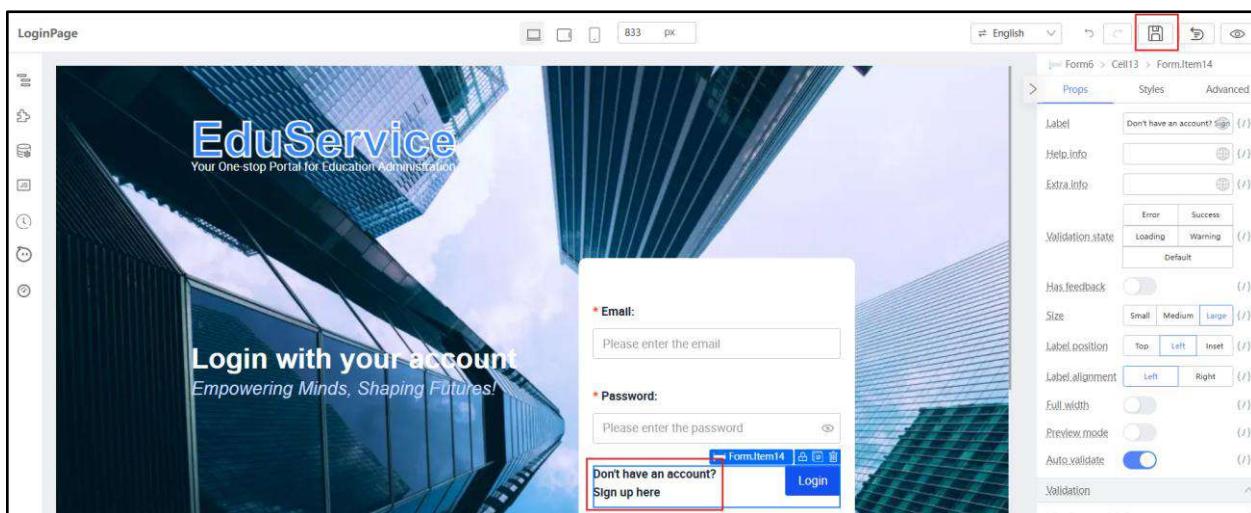


This setup allows you to observe how these features behave in a collaborative environment, particularly when actions are performed by other project members. You will also gain insight into how these actions affect visibility and interaction from your perspective.

Save Draft

The Save Draft feature allows you to save your project progress in KAIZEN app designer if you wish to save the progress without finalizing the design. In the below steps, we will guide you through the save draft function in KAIZEN designer.

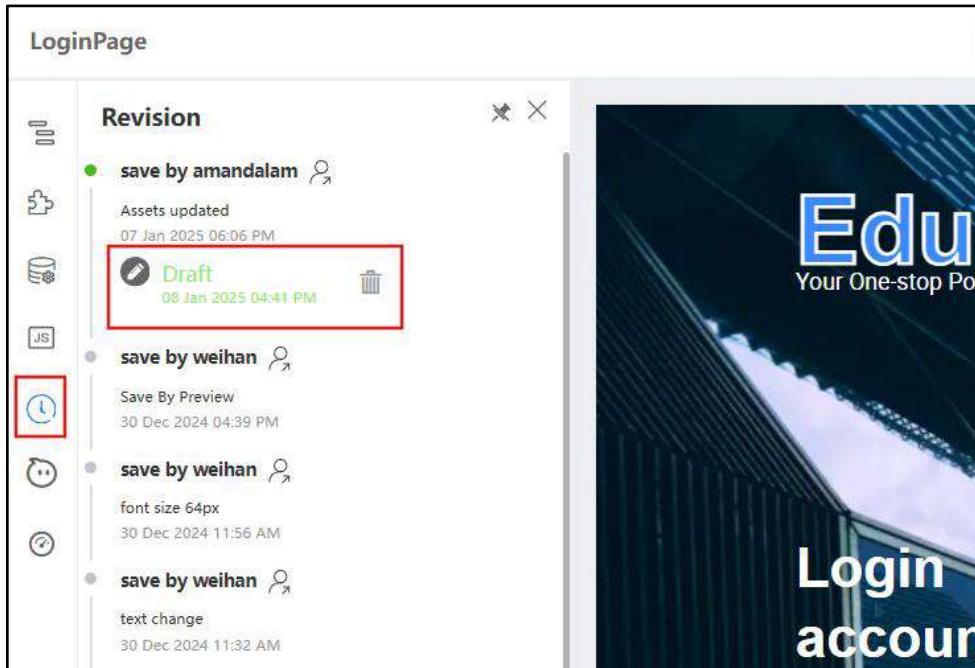
- First, we will amend the text below and save the draft.



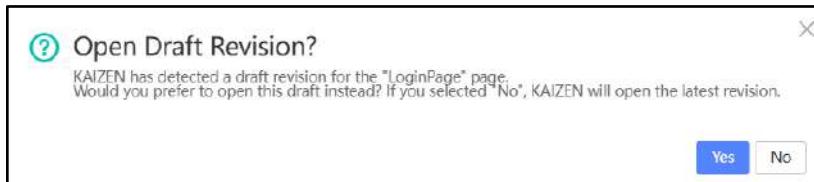
Draft saved successfully

- After which, you may click on the revision button and you can see that a draft has been saved in revision.

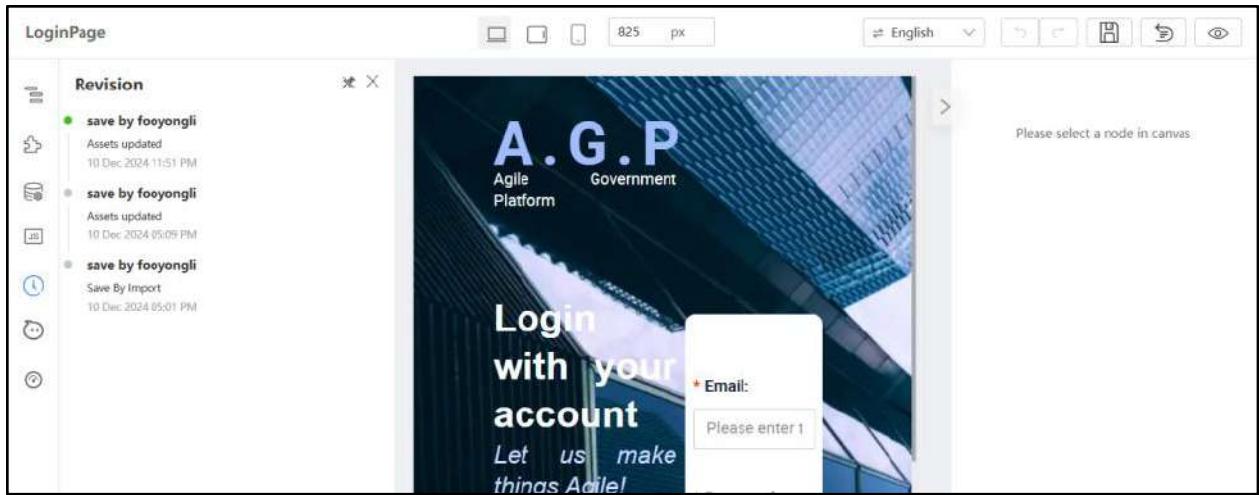
- We also have an auto-save mechanism every 15 minutes in the app designer. This time period can be configured at system level depending on project needs.



- Now, when you close the KAIZEN App Designer and re-open the same application again, or navigate to another page and back to this same page, it will prompt open draft revision dialog if there is a draft saved on this page.



- Click "Yes" if you wish to open the draft saved previously. It will open the page that we saved previously in draft.
- Click "No" if you wish to open the page from the latest revision.
- Do take note that when you save a draft, it is stored privately in your account's application. This means that only you have access to it, and other users cannot view or edit your saved work.
- Revision history from another account differs, where the draft will not be visible to another user.



- However, if you wish to save your changes and visible to other users, you have to **publish** the page.

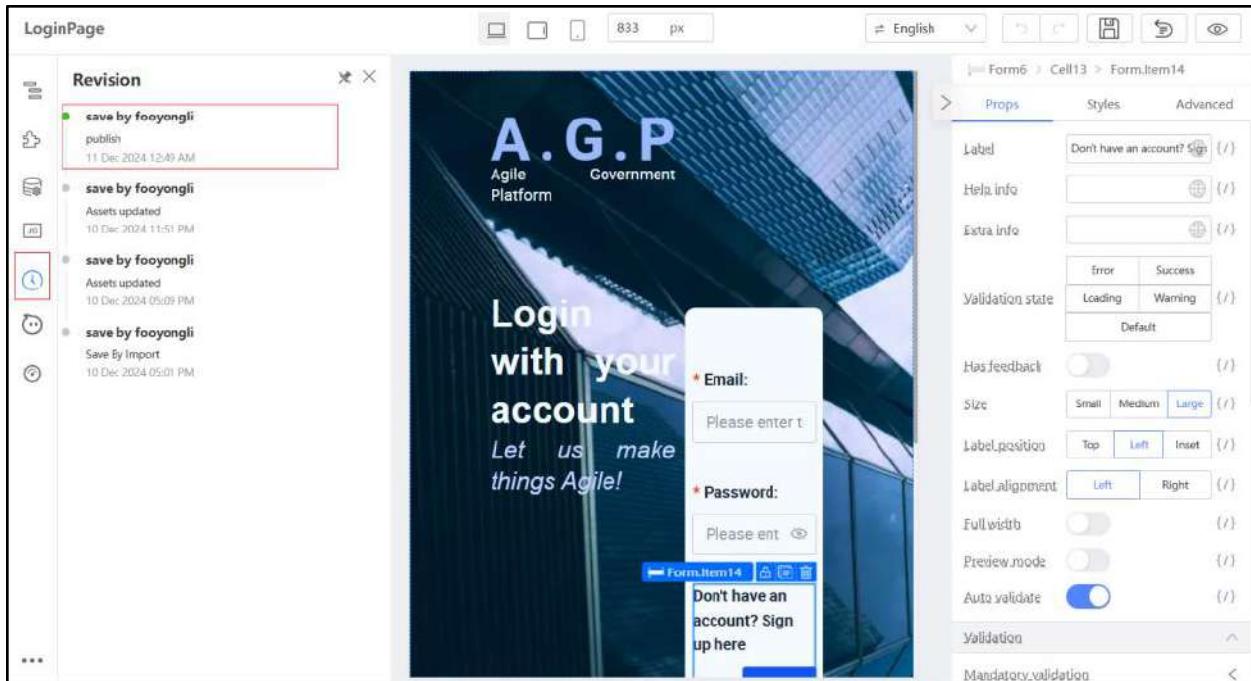
Publish Page

The Publish Page feature allows you to save your finalized page in KAIZEN app designer. Once published, your page becomes visible to other users. This means that team members, collaborators, or any designated user can view and interact with your content. It transforms your private draft into a shared resource. In the below steps, we will guide you through the publish function in KAIZEN designer.

- First, let's try to amend the text below and publish the page.

The screenshot shows the KAIZEN app designer interface. On the left, there's a sidebar with icons for file operations like New, Open, Save, and Delete. The main workspace displays a login page titled "A.G.P Agile Government Platform". The page features a large image of modern buildings and a central login form. The form has fields for "Email" and "Password", both with placeholder text "Please enter the email" and "Please enter the passw". Below the fields are two buttons: "Don't have an account? Sign up here" and "Login". The "Login" button is highlighted in blue. To the right of the workspace is a properties panel for "Form.Item14". The "Props" tab is selected, showing settings for the "Label" field, which currently contains "Don't have an account?". Other settings include "Help info", "Extra info", validation states (Error, Success, Loading, Warning, Default), "Has feedback" (on), size (Medium), label position (Inset), label alignment (Left), full width (off), preview mode (off), auto validate (on), and validation rules. A "Styles" tab is also present. At the bottom of the screen, a green notification bar says "Page *LoginPage* saved successfully".

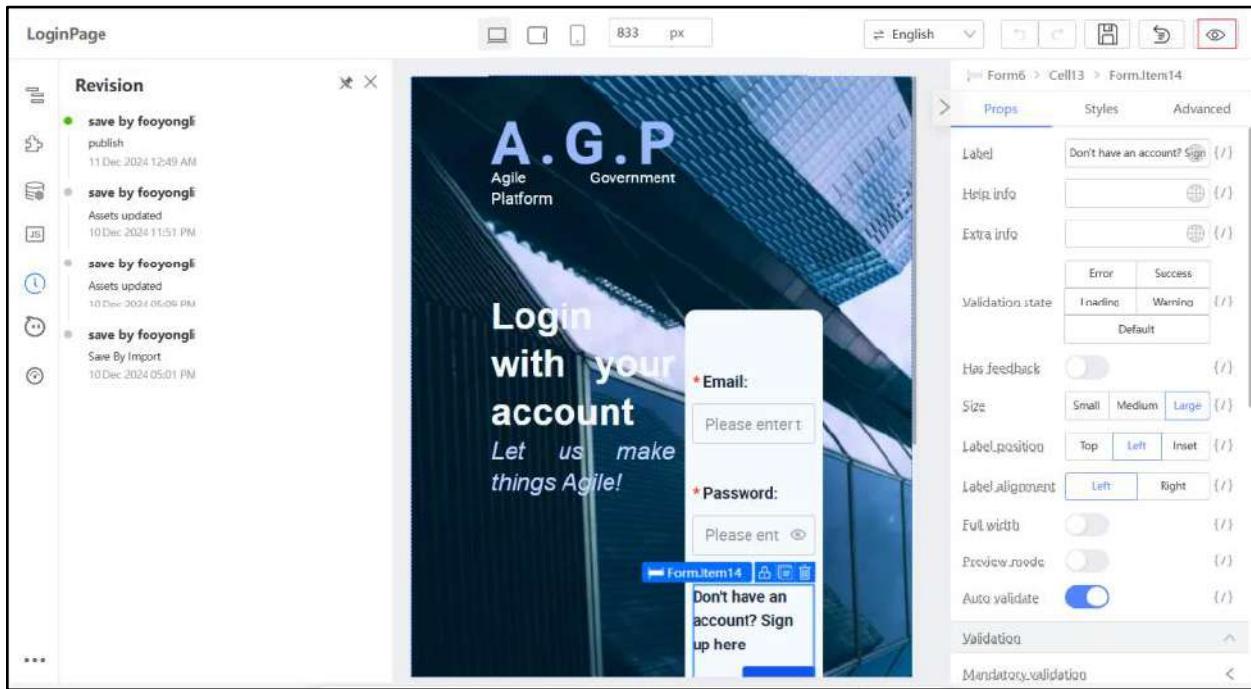
- Click on the revision button, and you can see a new revision history is updated.



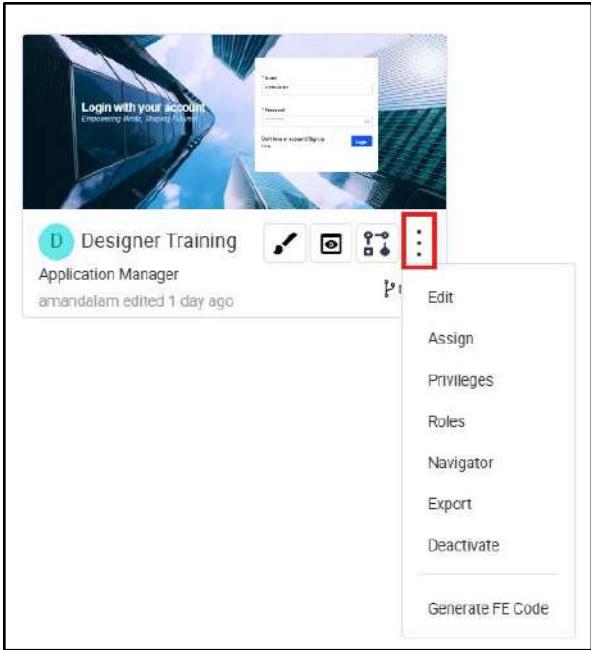
- Now, close the KAIZEN App Designer and re-open the same application again. It will not prompt open draft revision dialog anymore as currently there is no draft saved in the application.
- When you publish the page, it will be visible to other users in the project. This means that other users can view or continue editing your work which has been published. The revision history from another account will be the same.

Preview Page

The Preview Page feature allows you to see how your page will look and function. The preview feature provides a real-time view of your page, showing exactly how it will appear to end-users. Besides, you also can test the functionality of your page, including buttons, links, forms, and other interactive elements. This helps you identify and fix any issues before making the page live.



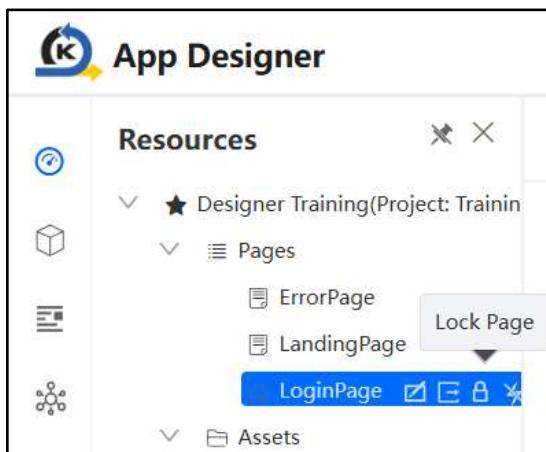
Note that once you click on the preview button, you will be prompted to **select the navigator roles** in which you want to preview. This functionality aids in the configuration of necessary privileges and roles during the design stage to restrict the **access of pages** through the **navigator menus** in the application. However, in this training tutorial, we will not need to indicate or choose the user domain or role name. Simply click on 'Confirm' to skip this selection and your application preview without any access restrictions.



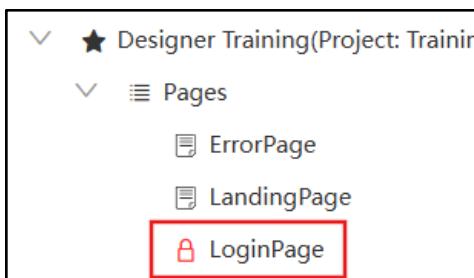
For example, you may configure at application level by clicking on this icon for more settings. This is where you can configure the relevant privileges > roles > navigator for your application pages. We will cover more on navigator in [Practical 9.2](#) and more on privileges and roles in [Tutorial 28](#) on IAM.

Page Lock

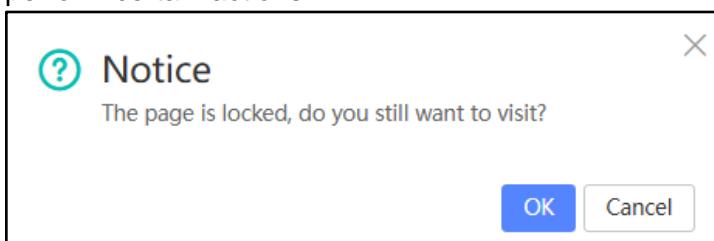
The Page Lock feature is a mechanism designed to control and manage collaborative editing in a shared project environment. This feature ensures that critical or in-progress work on a page is not unintentionally modified or overwritten by others, maintaining the integrity of the content. When a page is locked, it becomes restricted for other users. This means no other users can make significant changes and publish the page until it has been unlocked by either the user who locked the page initially or the project manager.



- In the shared **TrainingFoundation** project, we have locked the Login page to illustrate how this feature functions.



- You can view the locked page and experience first hand what happens when they try to perform certain actions.



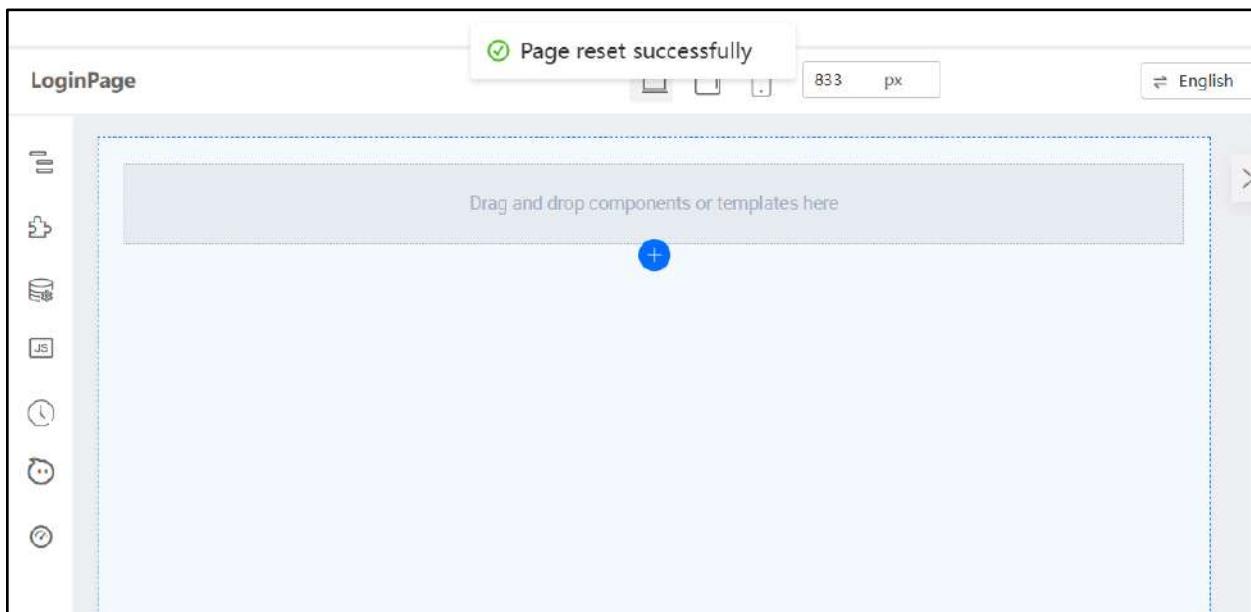
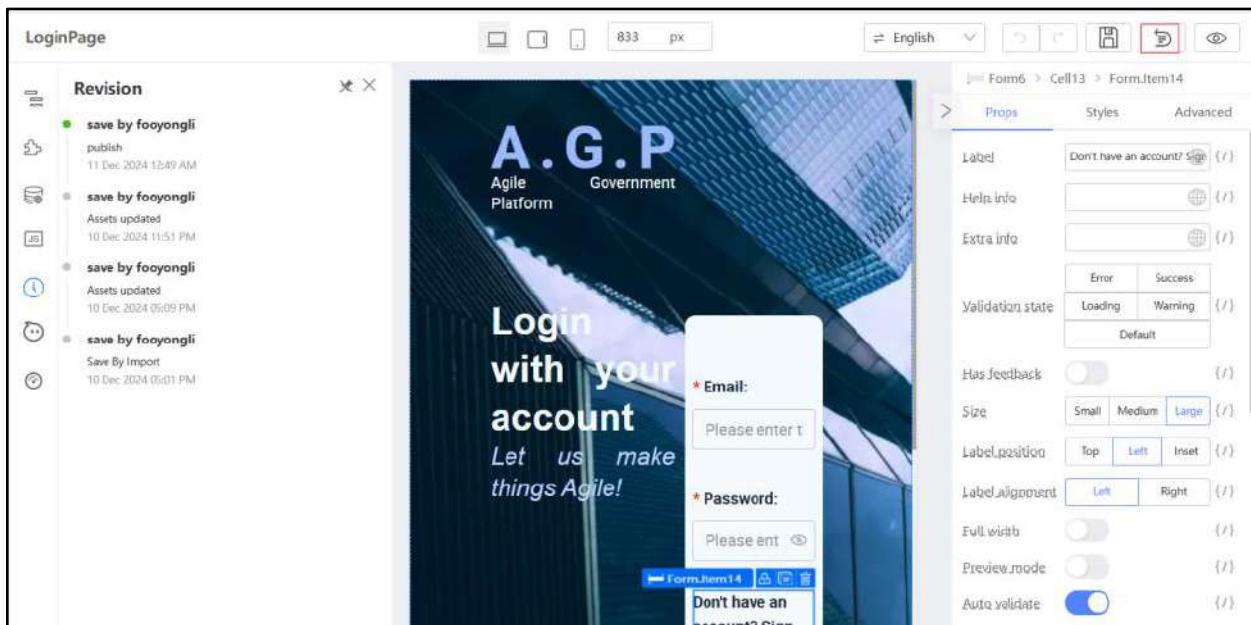
- Notice that if you make changes then **preview** or **publish** the page, it will be restricted.

 The page is locked, so you can't preview or publish.

Reset

The Reset feature will reset your page to an empty page, allowing the user to remove all the components added in the page with a reset button.

Note that you will **not** be able to undo this change!

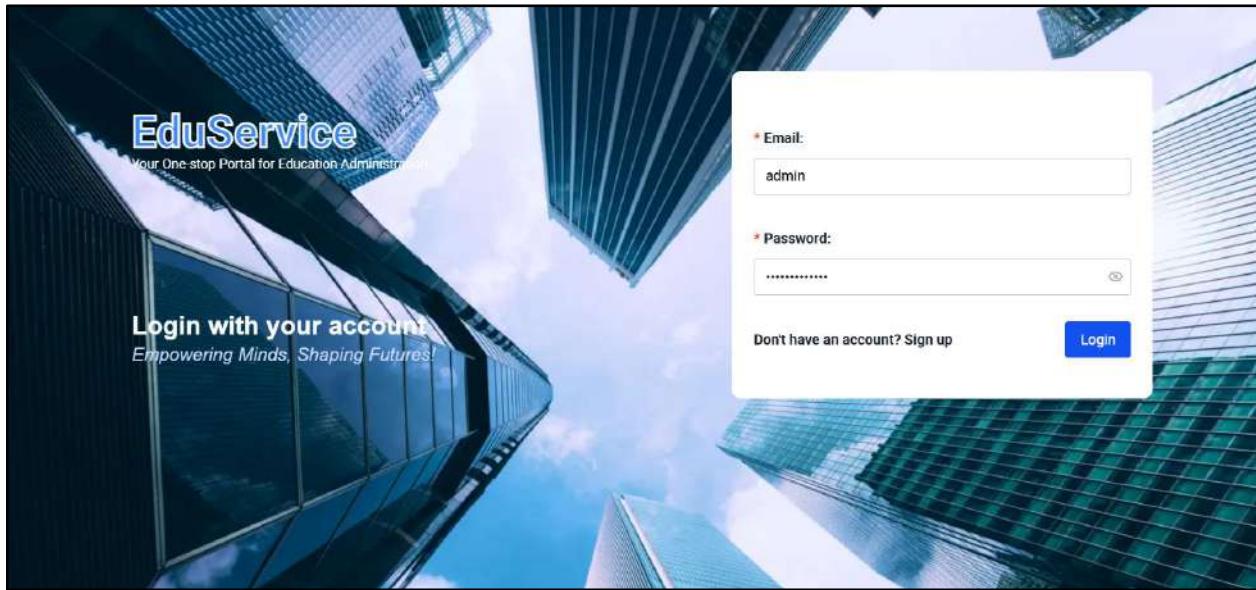


Tutorial 1: Creating a responsive Login page

This tutorial covers the following Learning Objectives:

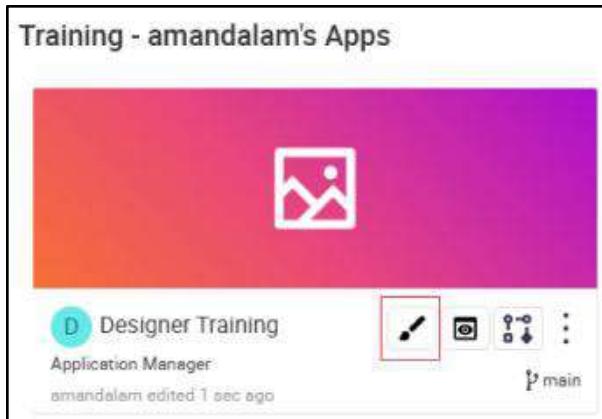
- Understand how to design a basic responsive login page using KAIZEN
- Experience the benefits of KAIZEN for rapid development.

A login page is the entry point for user authentication, ensuring secure access to your application. In this tutorial, we will create a login page using KAIZEN



In KAIZEN Studio, you will see a Training Project which has been created with your trainee username. An empty application, 'Designer Training' has also been created for you to design and create your application following the tutorials below.

You can start editing the Designer Training App by clicking on the highlighted icon as shown in the image below.



Since we are creating a Login page, change the Name and Page ID of the 'DefaultPage' to 'LoginPage' and click **Save**.

Two side-by-side screenshots. On the left, the "Resources" panel shows a tree structure with "Demo Training" expanded, "Pages" selected, and "DefaultPage" highlighted with a red box. An "Edit Page" button is visible above the list. On the right, the "Edit Page" dialog is open, showing fields for Application (set to "Demo Training"), Name (set to "LoginPage"), Page ID (set to "LoginPage"), Type (set to "Page"), Status (set to "Active"), Default Page (set to "Yes"), Is Protected (set to "No"), and Privileges (set to "Anonymous Resource"). The "Save" button at the bottom right is highlighted with a red box.

You will see a success message pop up:

Page "LoginPage" updated successfully

Practical 1.1: Add Background Image

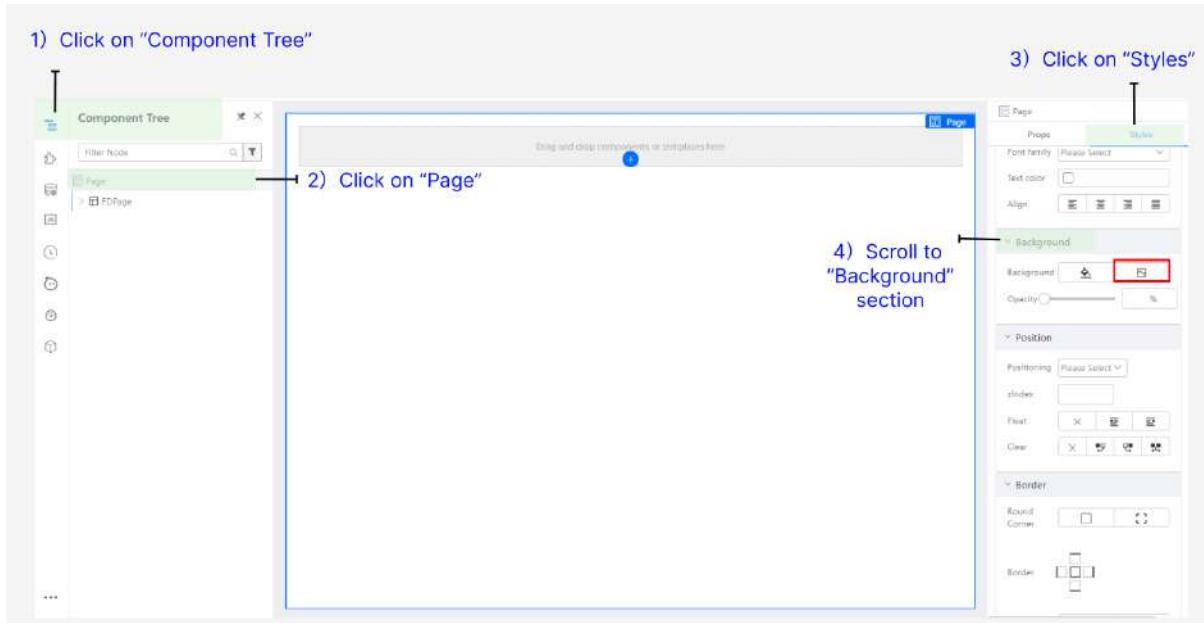
- **Download** image url: [Tutorial1-Background.png](#)
- **Upload** the image asset in the resources panel by clicking on + icon



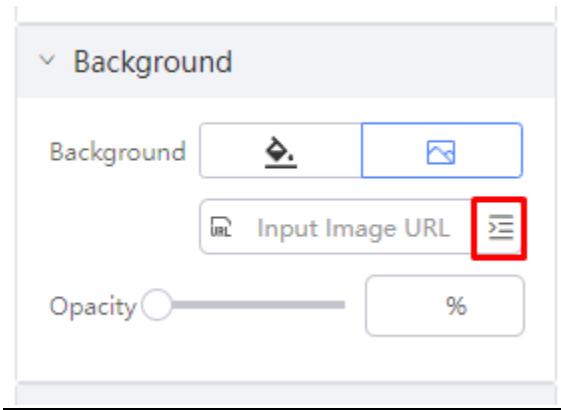
- Click **Save**

Asset "Tutorial1-Background.png" created successfully

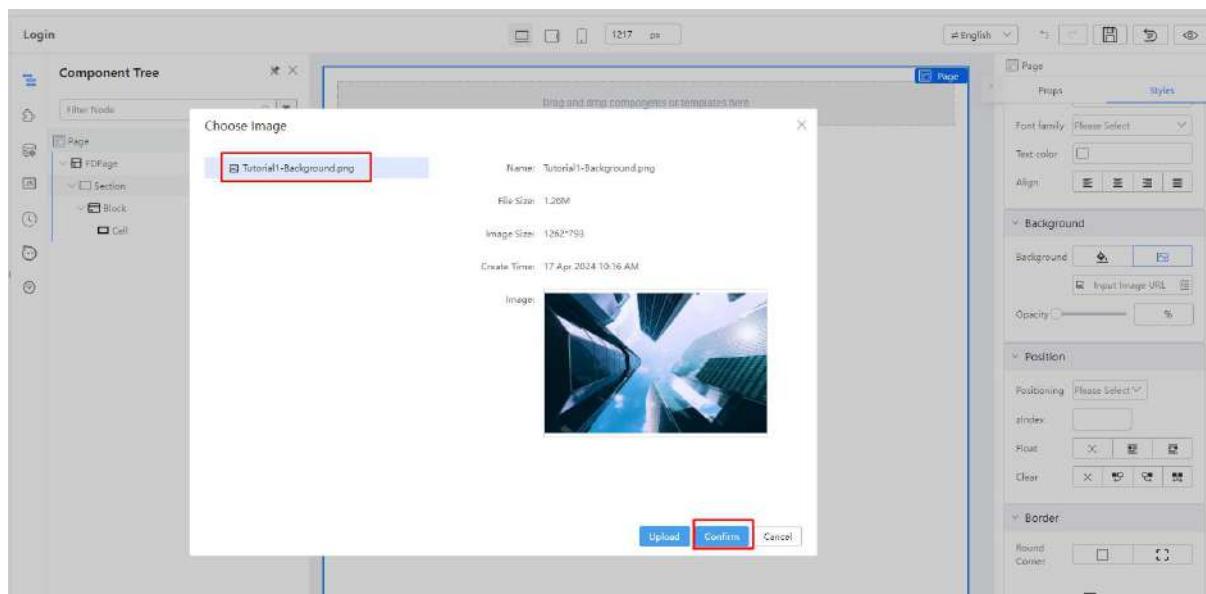
- Select **Page** from the **Component Tree** and navigate to **Styles** tab
 - Step 1: Click on **Component Tree**
 - Step 2: Click on **Page**
 - Step 3: Click on **Styles**
 - Step 4: Scroll down to **Background** section



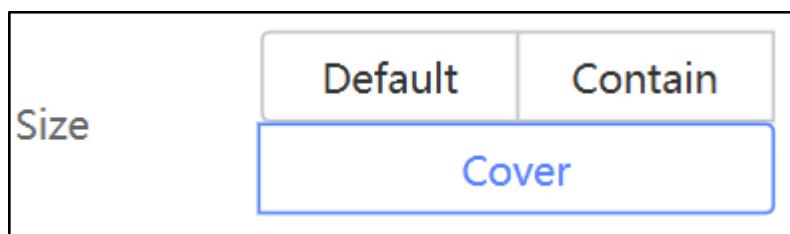
- Click on the image upload  icon



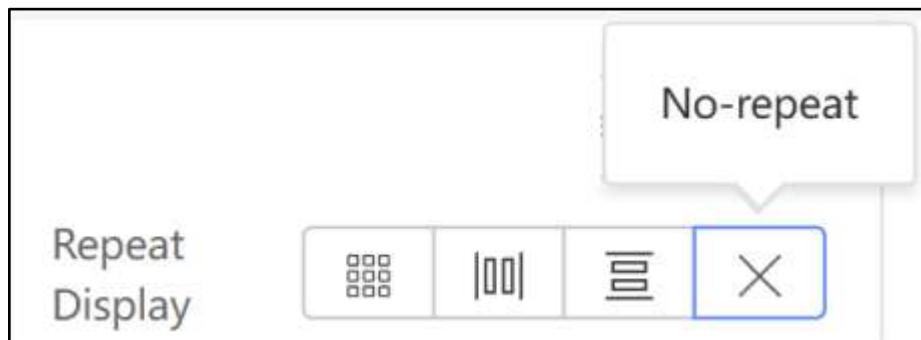
- Select on the background uploaded earlier and click **Confirm**



- Set the background-size (Size) as **Cover**



- Set the Repeat Display as **No-repeat**



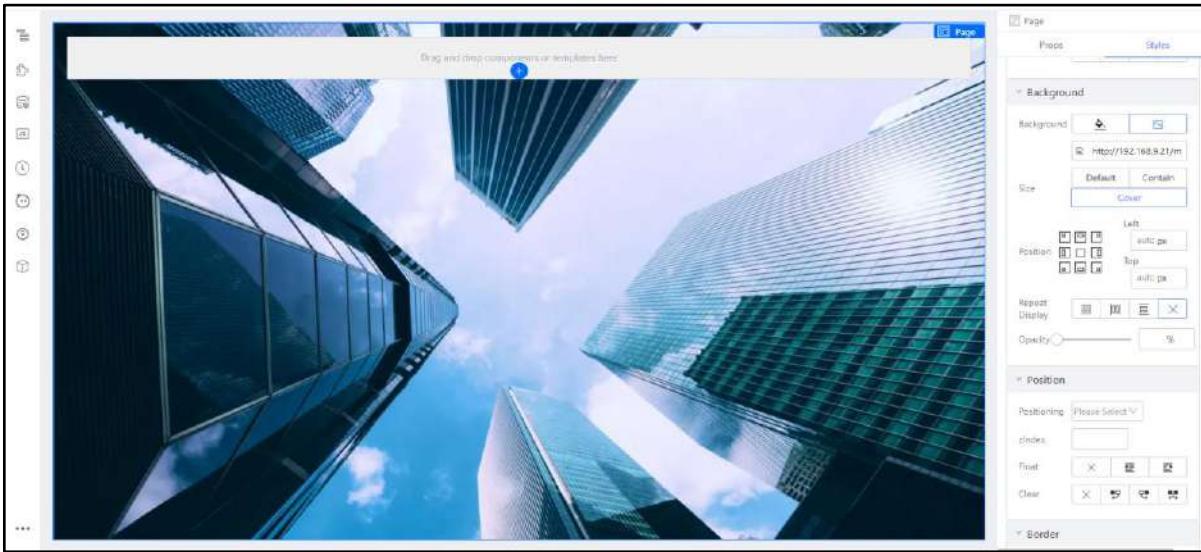
- The resulting styles will be as follows:

Styles

```
height: 100%;  
background-image:  
url(/gateway/console/api/v1/asset/designertraining/assets/image/Tutorial1-  
Background.png?branchName=main);  
background-size: cover;  
background-repeat: no-repeat;
```

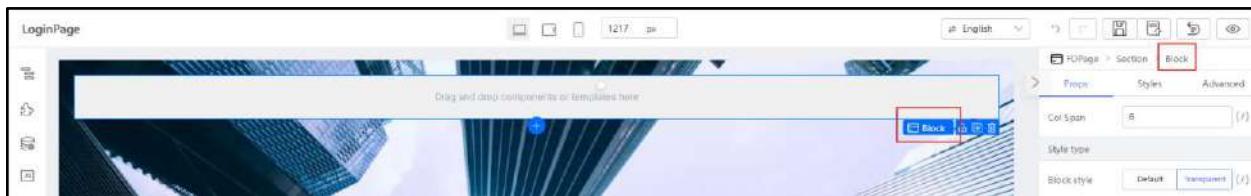
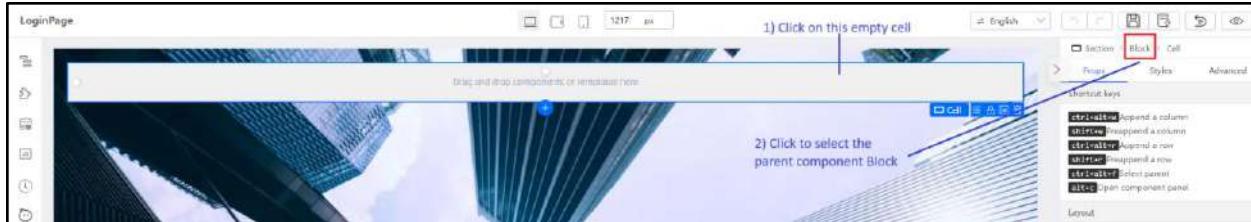
This image shows the "Background" settings panel from the Figma design tool. The "Size" dropdown is set to "Cover". The "Position" dropdown is set to "Left Top" with values "auto px". The "Repeat Display" dropdown is set to "No-repeat". The "Opacity" slider is at 100%.

Expected result will look like:

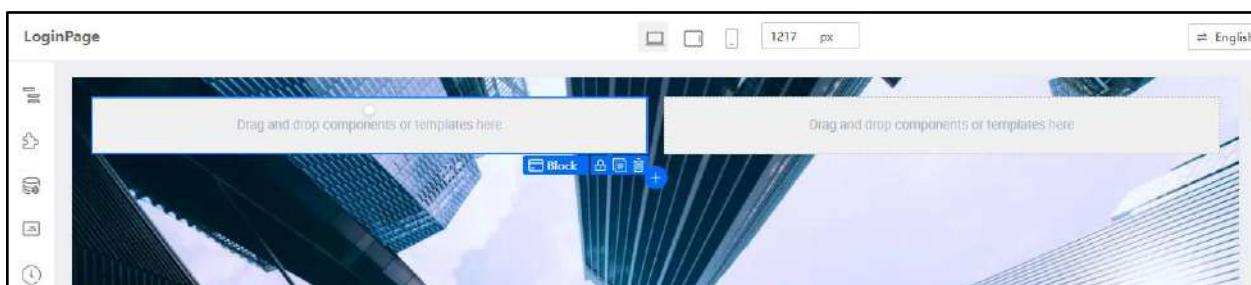
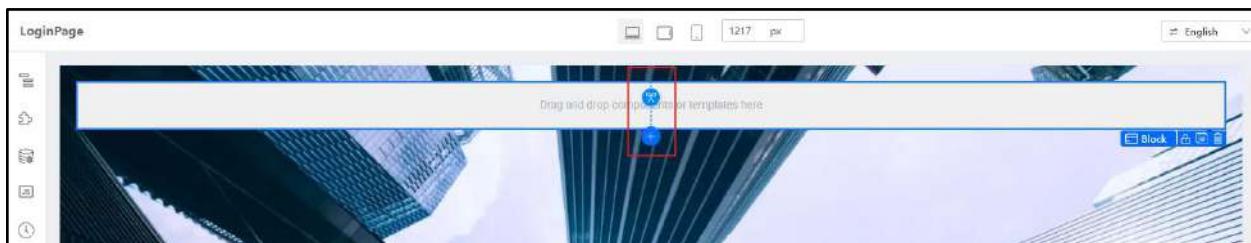


Practical 1.2: Create Responsive Layout

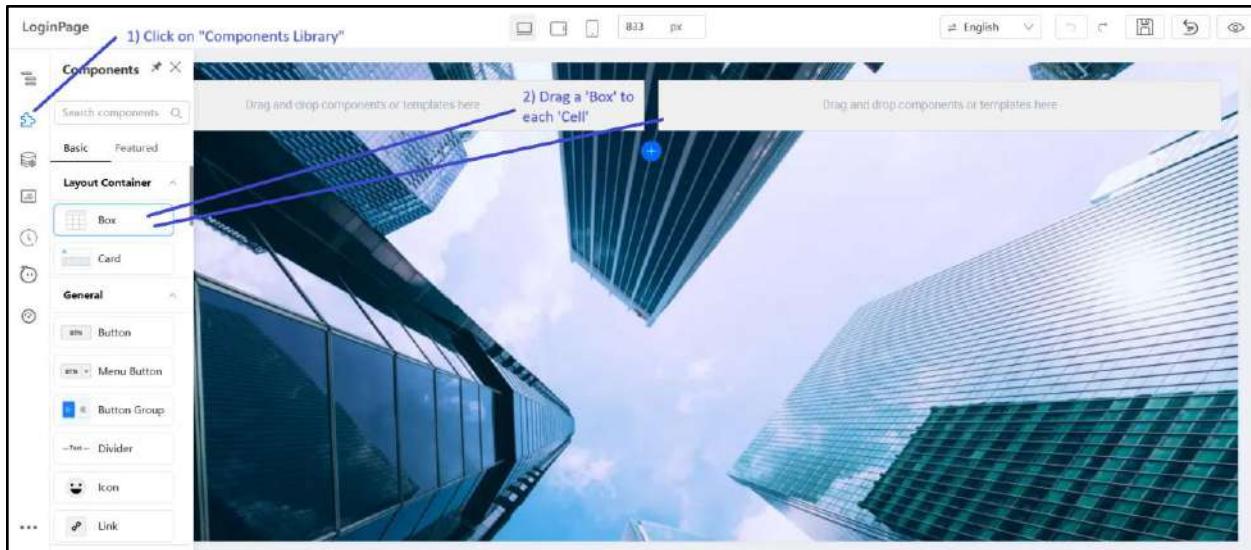
- Click on the empty **Cell** and navigate to the breadcrumb at the top right corner of your page. Click to select the parent component **Block** to be selected instead.



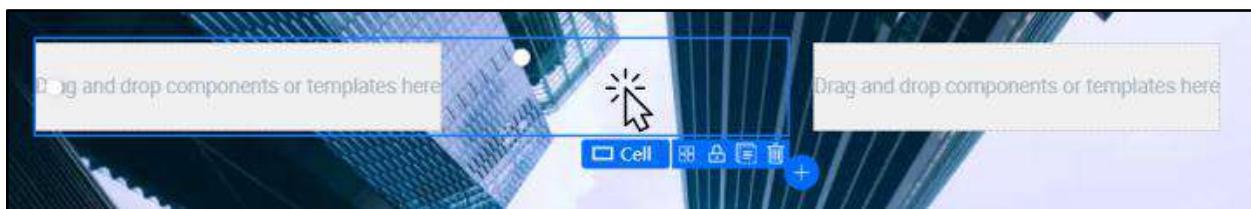
- In order to design this login page to be responsive and adapt to smaller screen sizes, we will be utilising the Block component to design the left and right panel of the page. For more information, you may refer to [Tutorial 13](#).
- With the **Block** component selected, hover on the middle white dot for the Cut icon to appear, click on it to vertically cut your block into two.



- Click on **Components Libraries**
- Drag a **Box** component to each cell shown in screenshot below



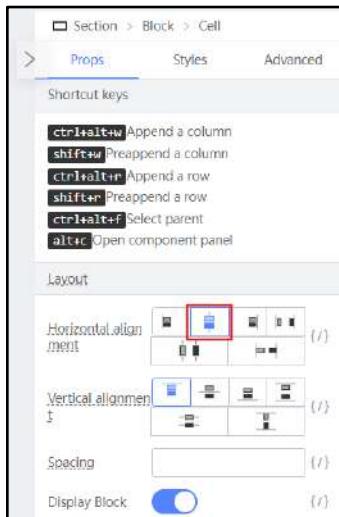
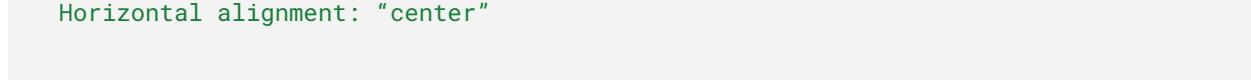
- Select the **Cell (Outer layer of Box)** by **clicking** on the containing area



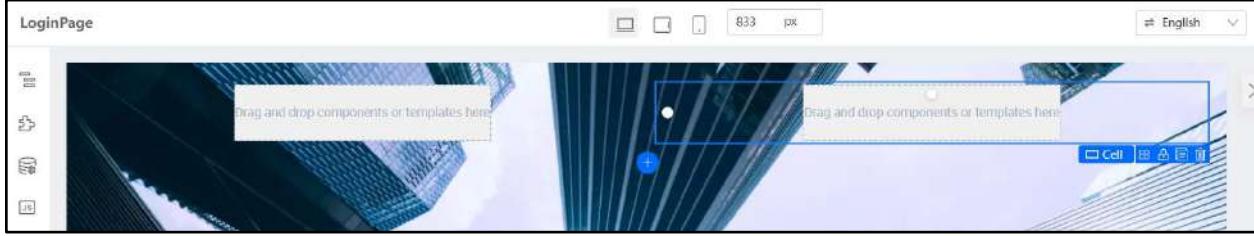
- Set the following properties

Props

Horizontal alignment: "center"



- Set the same for the cell on the right so achieve the following



- Select the first **Box (Left)**, and set

Props

```
Justify: "flex-start"
Align: "flex-start"
```

1) Select the **Box (Left)**

The screenshot shows the UI builder with the left box selected. A callout points to the top-left corner of the left box. The properties panel on the right is open, showing the 'Justify' and 'Align' sections, both of which have their 'flex-start' options selected. Other settings like 'Direction', 'Width', and 'Spacing' are also visible.

- In the same **Box**, and set the **Styles**

Styles

```
width: 65%;  
height: 100%;  
margin-top: 50px;
```

The height of the box is set to 100% so that it is relative. Since it is empty with no child components inside it yet, it is expected for the box to currently look empty as follows.



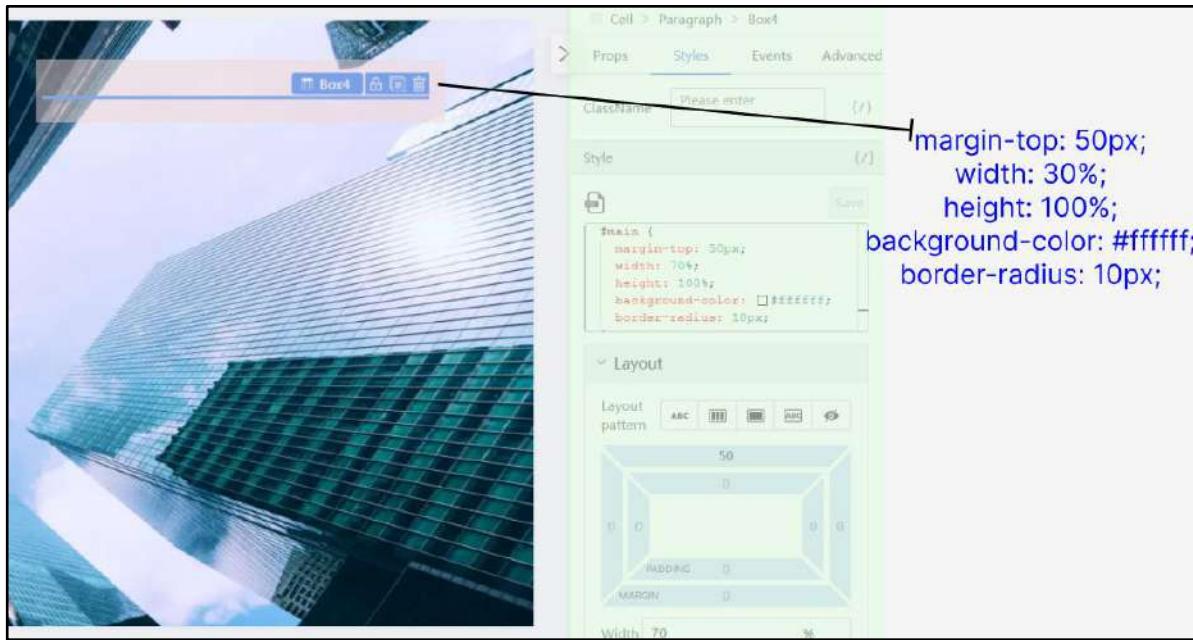
- Select the second **Box (Red-highlighted)**



- Set the **Styles** to the following:

Styles

```
margin-top: 50px;
width: 70%;
height: 100%;
background-color: #ffffff;
border-radius: 10px;
```

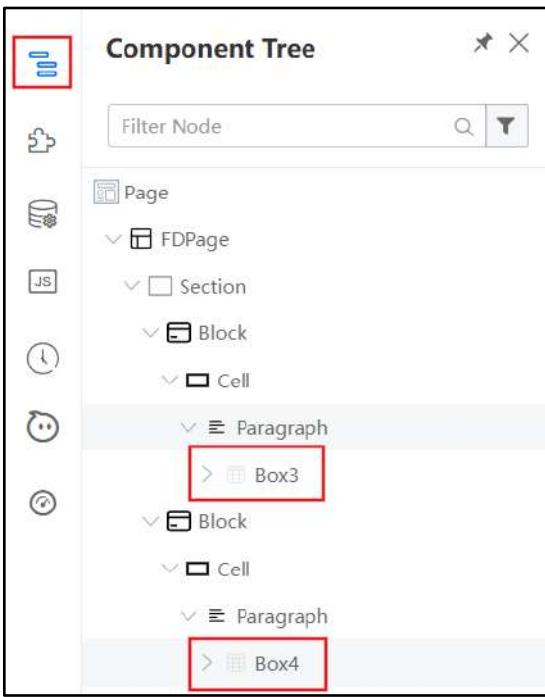


The following sections (**Layout**, **Background**, **Border**) are to be edited:

Similar to the box on the left, due to the relative height configured in percentage, the expected result will be:



You will still be able to select your respective **Box** components under the left and right **Block** components via the **Component Tree**.



Practical 1.3: Add 4 Text components

- Ensure that the left **Box** component is selected
- In the **Component Library**
 - Search for keyword “text”,
 - Drag one **Text Component** into **Box (left)**



- Set the following properties and styles

Props
Text: EduService

Styles
margin-top: 10%;
font-size: 50px;
line-height: 50px;
color: #3c89fc;
font-weight: 700;
-webkit-text-stroke: 2px white;

The screenshot shows the configuration interface for a Text component. At the top, there's a toolbar with icons for Text, EduService, and a globe. Below it is a code editor with the opening brace of a CSS style block visible. The main area is divided into two sections: Layout and Font.

Layout:

- Layout pattern:** ABC, Grid, Box, ABC, Grid.
- Vertical margin:** 10px (highlighted with a black box).
- Note:** "Note: change this to 10% in the CSS Style box."
- Width:** 0px
- Height:** 0px

Font:

- Font size:** 50px
- Line height:** 50px
- Font weight:** 700 Bold
- Font family:** Please Select
- Text color:** #3c89fc
- Align:** Left, Center, Right, Justify

- Expected result will be:



- Add three additional text components and set the following properties and styles:

Second text	<u>Props</u>
	Text: Your One-stop Portal for Education Administration
	<u>Styles</u>

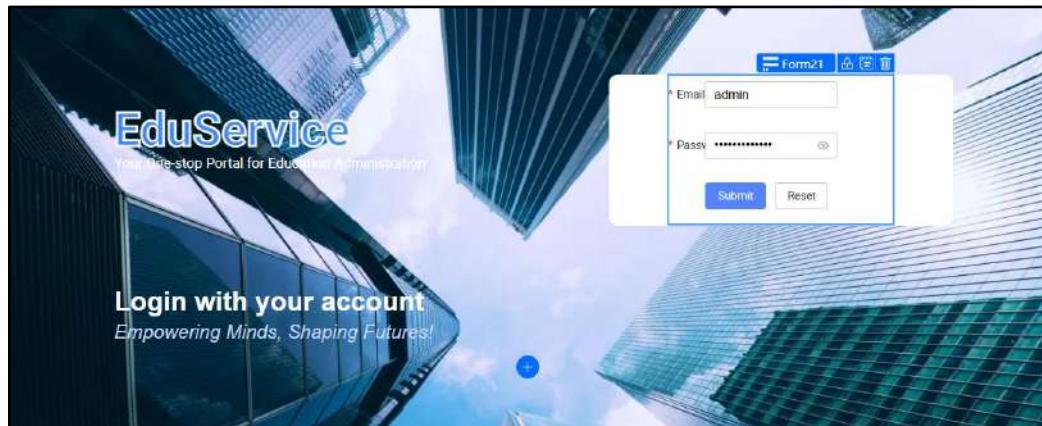
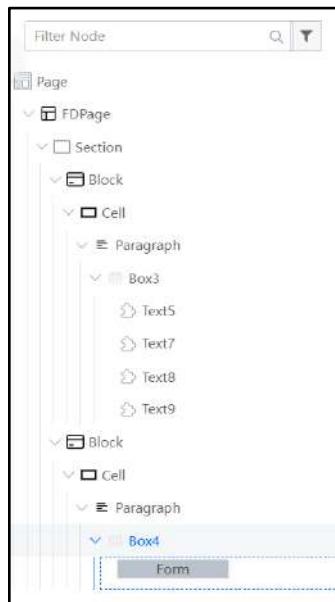
	<pre>font-size: 15px; color: #ffffff;</pre>
Third text	<p>Props</p> <p>Text: Login with your account</p> <p>Styles</p> <pre>margin-top: 35%; font-size: 28px; font-weight: 600; font-family: Helvetica; color: #ffffff;</pre>
Fourth text	<p>Props</p> <p>Text: Empowering Minds, Shaping Futures!</p> <p>Styles</p> <pre>font-size: 20px; font-family: Helvetica; font-style: italic; color: #c8d8ff;</pre>

- Your completed left box should look like this:



Practical 1.4: Add Form Template

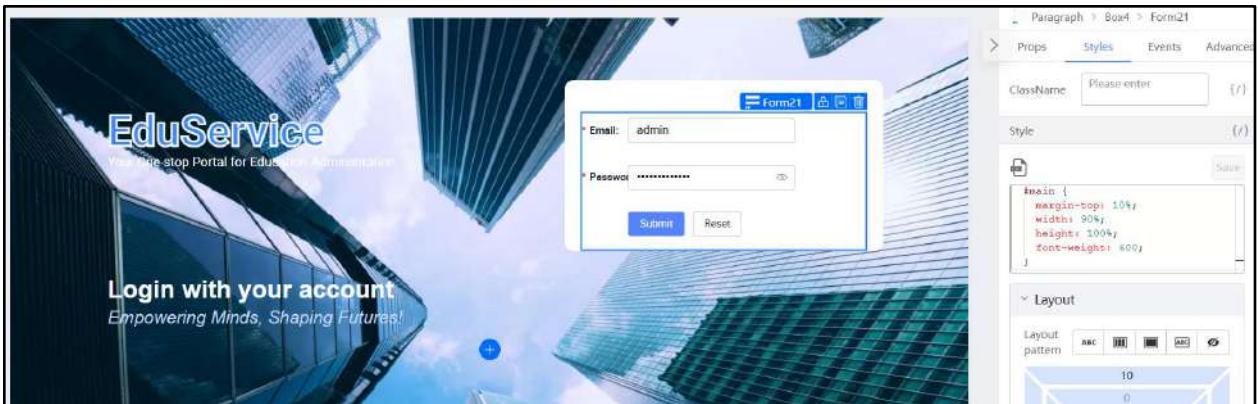
- In the **Component Library**:
 - **Search for “Form Template”**
 - Drag the component into right **Box**



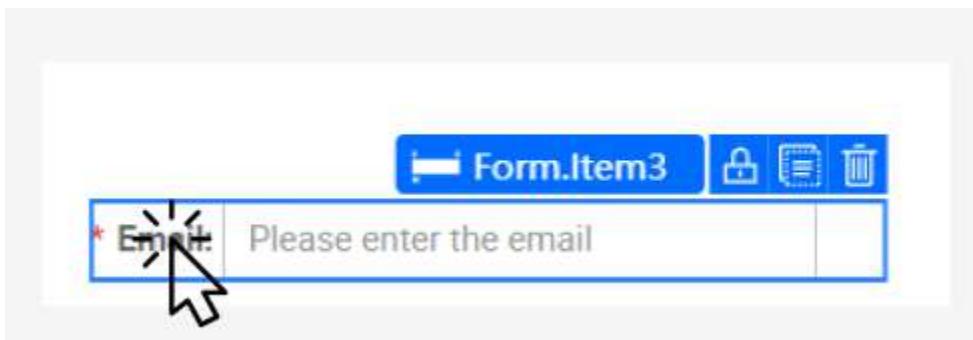
- Set the **Styles** of Form Template

Styles

```
margin-top: 10%;  
width: 90%;  
height: 100%;  
font-weight: 600;
```



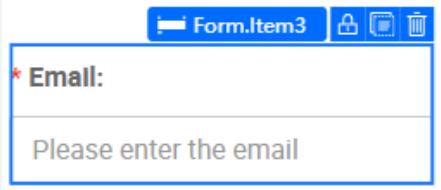
- Select the first Form Item by clicking on the label “Email”



- Set the properties to

Props

Size: Large
Label Position: Top

	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">Size</td> <td style="padding: 5px; border: 1px solid #ccc; border-radius: 5px; text-align: center;">Small</td> <td style="padding: 5px; border: 1px solid #ccc; border-radius: 5px; text-align: center;">Medium</td> <td style="padding: 5px; border: 1px solid #ccc; border-radius: 5px; text-align: center; background-color: #0078D4; color: white;">Large</td> <td style="padding: 5px;">{</td> </tr> <tr> <td style="padding: 5px;">Label position</td> <td style="padding: 5px; border: 1px solid #ccc; border-radius: 5px; text-align: center;">Top</td> <td style="padding: 5px; border: 1px solid #ccc; border-radius: 5px; text-align: center;">Left</td> <td style="padding: 5px; border: 1px solid #ccc; border-radius: 5px; text-align: center;">Inset</td> <td style="padding: 5px;">{</td> </tr> </table>	Size	Small	Medium	Large	{	Label position	Top	Left	Inset	{
Size	Small	Medium	Large	{							
Label position	Top	Left	Inset	{							

- Select the Form Item (label “Password”) and set

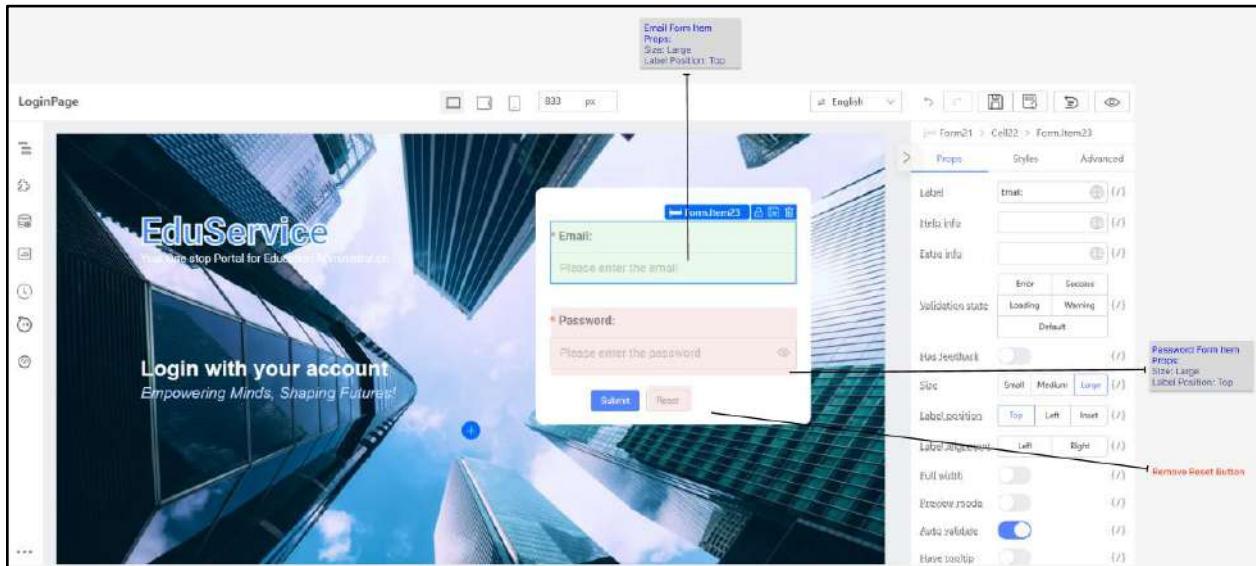
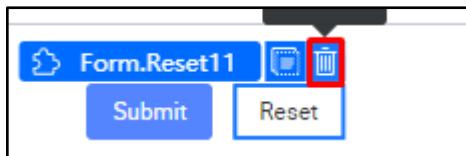
Props

Size: Large

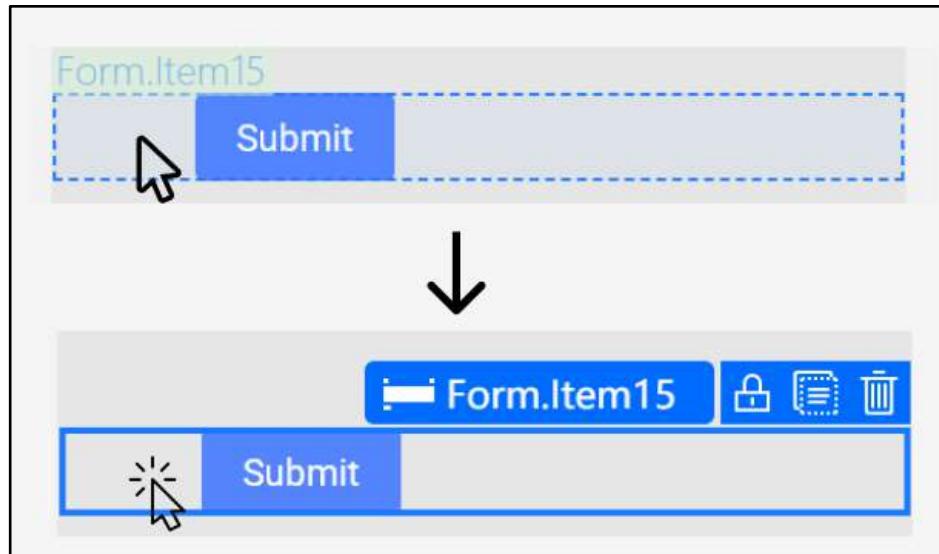
Label Position: Top



- Delete the **Reset** button by clicking on trash icon



- Select the last form.item by clicking on the containing area (**Beside Submit button**)



- Set the last form item with the following:

Props

Label: Don't have an account? Sign up

Size: Large

Label Alignment: Left

Layout

Col Span: 24

Label Column

Span: 18

Offset: 0

Wrapper Column

Span: 6

Offset: 0

Styles

text-align: right;

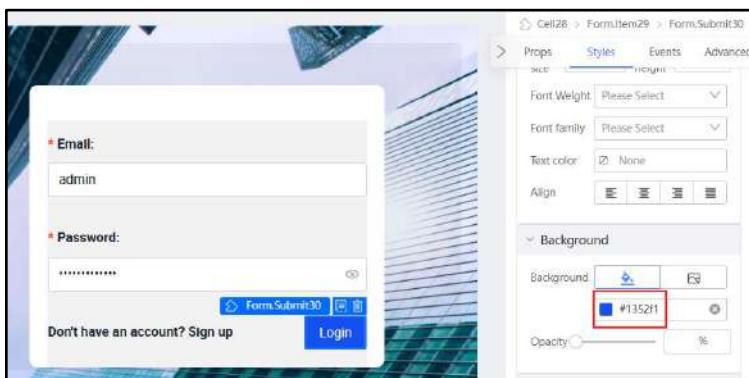
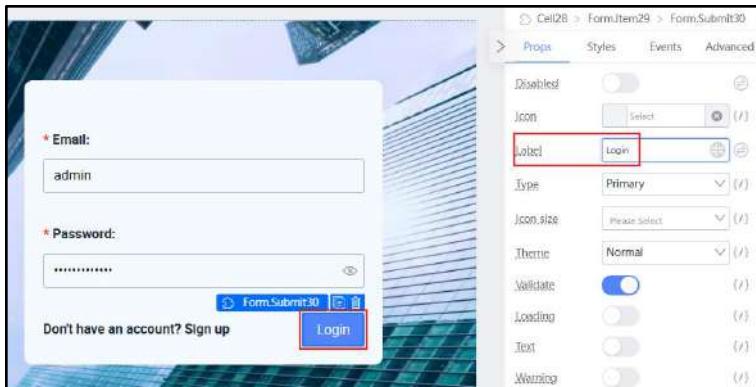
The screenshot shows a UI configuration interface with several tabs and sections:

- Props Tab:**
 - Label:** Don't have an account? Sign up
 - Help.info:** [Input field]
 - Extra.info:** [Input field]
 - Validation.state:** Error, Success, Loading, Warning (Default)
 - Has feedback:** [Switch]
 - Size:** Small, Medium, Large
 - Label position:** Top, Left, Inset
 - Label alignment:** Left, Right
 - Full width:** [Switch]
- Layout Tab:**
 - Col Span: 24
 - Label column: 18
 - Start: 0
 - Offset: 0
 - Header column: 6
 - Span: 6
 - Offset: 0
- Font Styles Tab:**
 - Font size:** 0 px
 - Line height:** 0 px
 - Font weight:** Please Select
 - Font family:** Please Select
 - Text color:** [Color picker] Right
 - Align:** [Buttons] (The center button is highlighted with a red border)

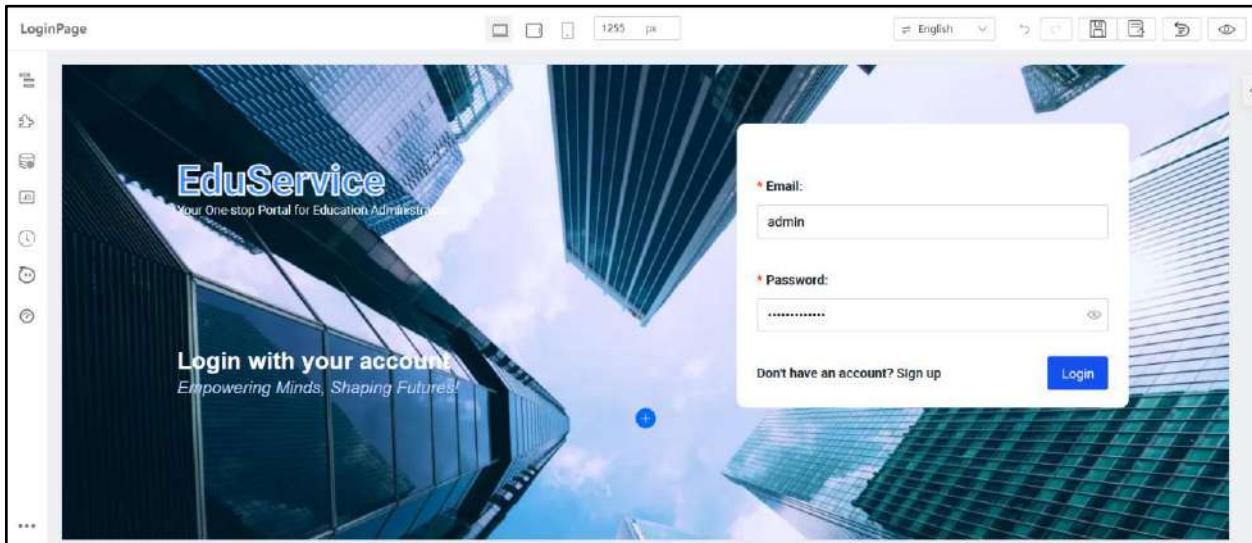
- Set the last form item (Submit button) with the following:

Props
Label: Login

Styles
background-color: #1352f1;



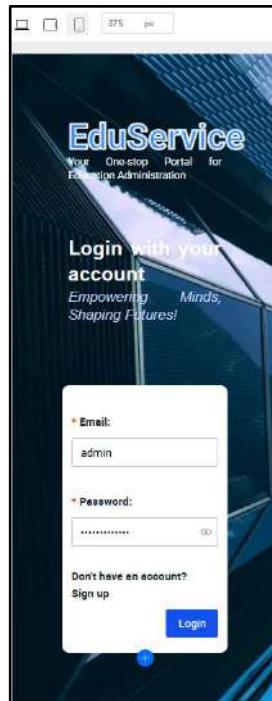
The expected result will be:



This page is also designed to be mobile responsive and adapts to smaller screen sizes. You may toggle between the screen sizes using the **responsive layout** at the top of the page here.



Since we have used blocks to design the two left and right segments of this login page, when toggled to mobile view, the blocks automatically stack in a user-friendly interface as follows.



For more information on mobile responsive features, refer to [Practical 10.2](#).

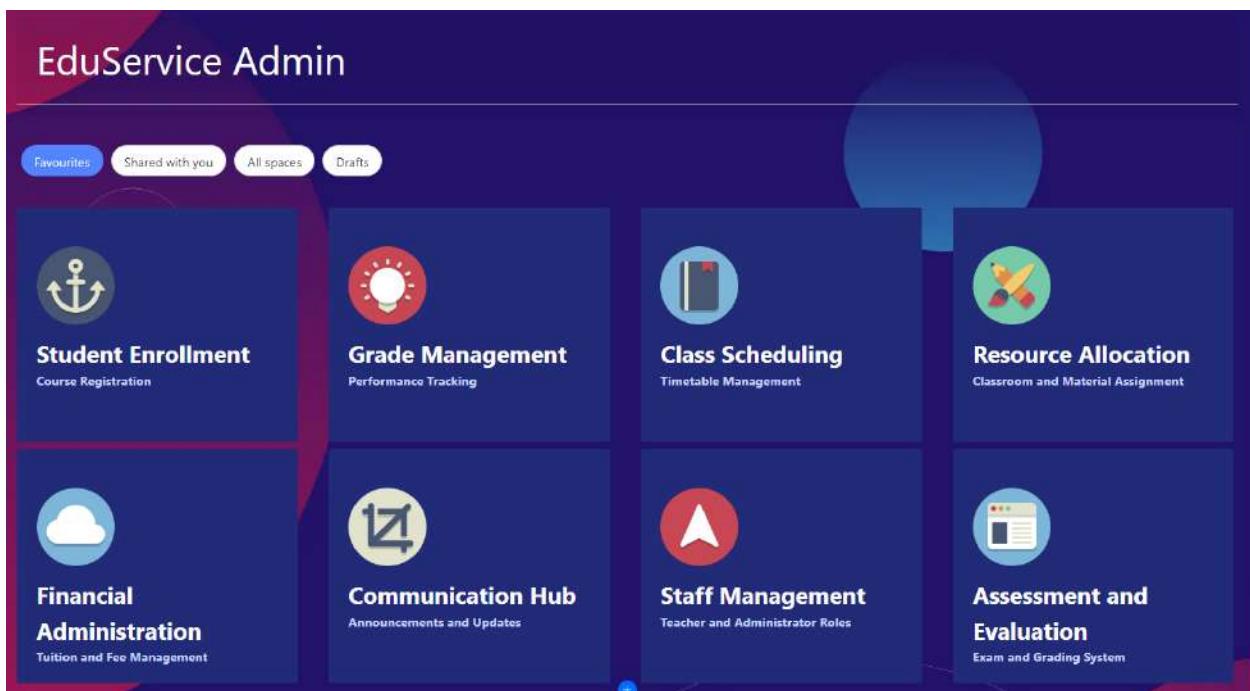
For more information on the process of structuring the screen layout to ensure flexibility and for it to adapt to varying screen sizes, refer to [Tutorial 13](#).

Tutorial 2: Creating the Landing Page

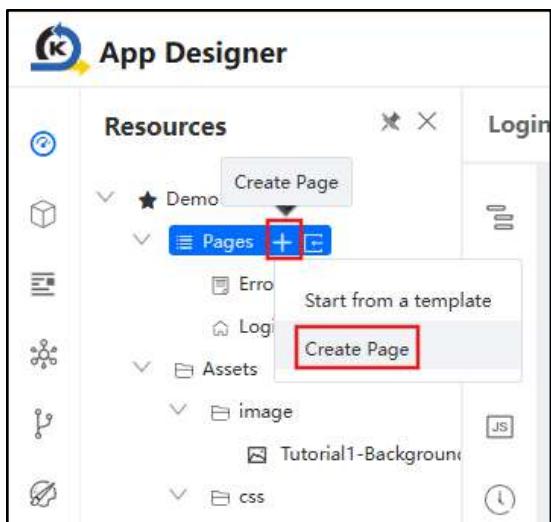
This tutorial covers the following Learning Objectives:

- Understand how to design and build an effective landing page using KAIZEN.
- Learn how to structure the landing page to provide intuitive navigation and a welcoming user experience.
- Explore how KAIZEN simplifies the process of customizing the landing page, including adding images, text, and buttons.

In this tutorial, we will walk you through the steps to design and build an effective landing page using KAIZEN. You'll learn how to structure the page to ensure intuitive navigation and create a welcoming user experience. Additionally, we'll explore how KAIZEN makes customizing the landing page easy, allowing you to add images, text, and buttons seamlessly.



In order to create a new page, click on 'Create Page' of your application's Pages and indicate your page details accordingly before clicking **Save**.



Create Page

* Application: Demo Training

* Name: LandingPage

* Page ID: LandingPage

* Type: Page

Description:

* Status: Active Inactive

* Default Page: Yes No

* Is Protected: Yes No

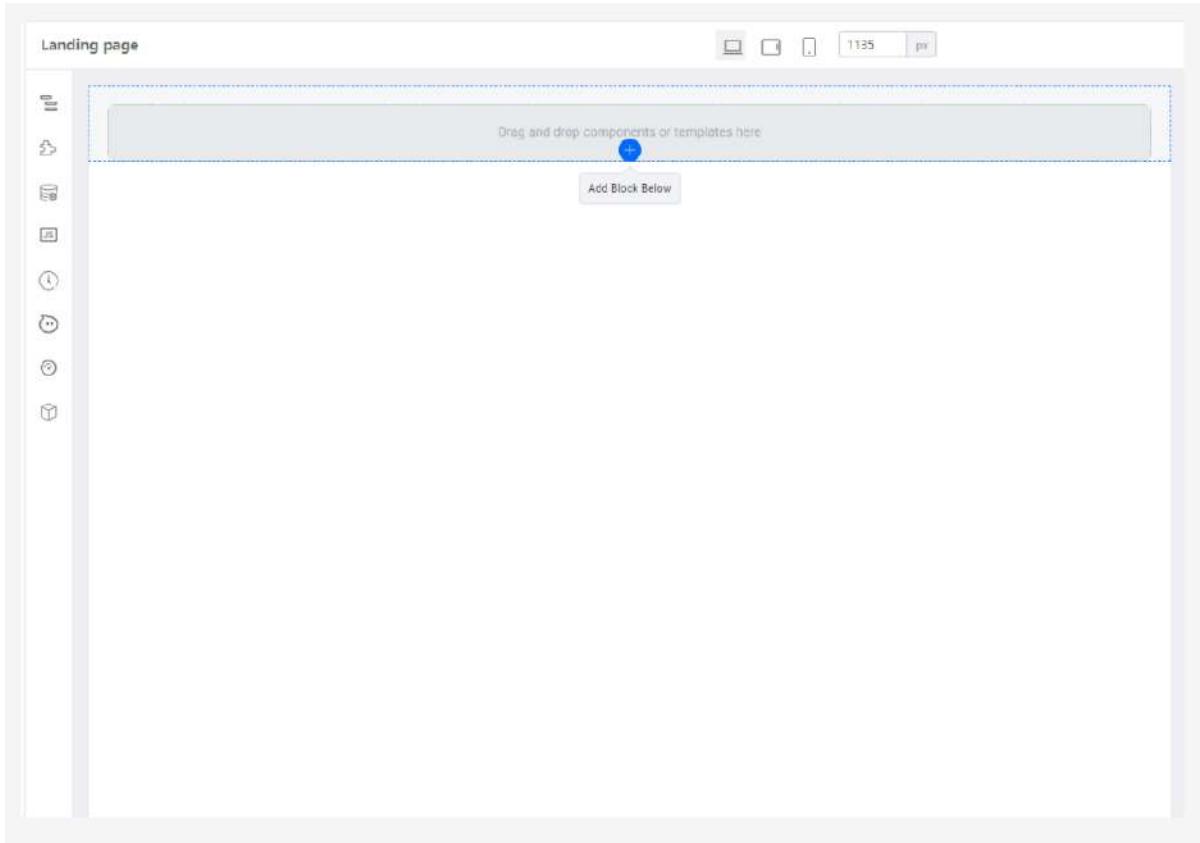
Privileges: Anonymous Resource

Cancel Save

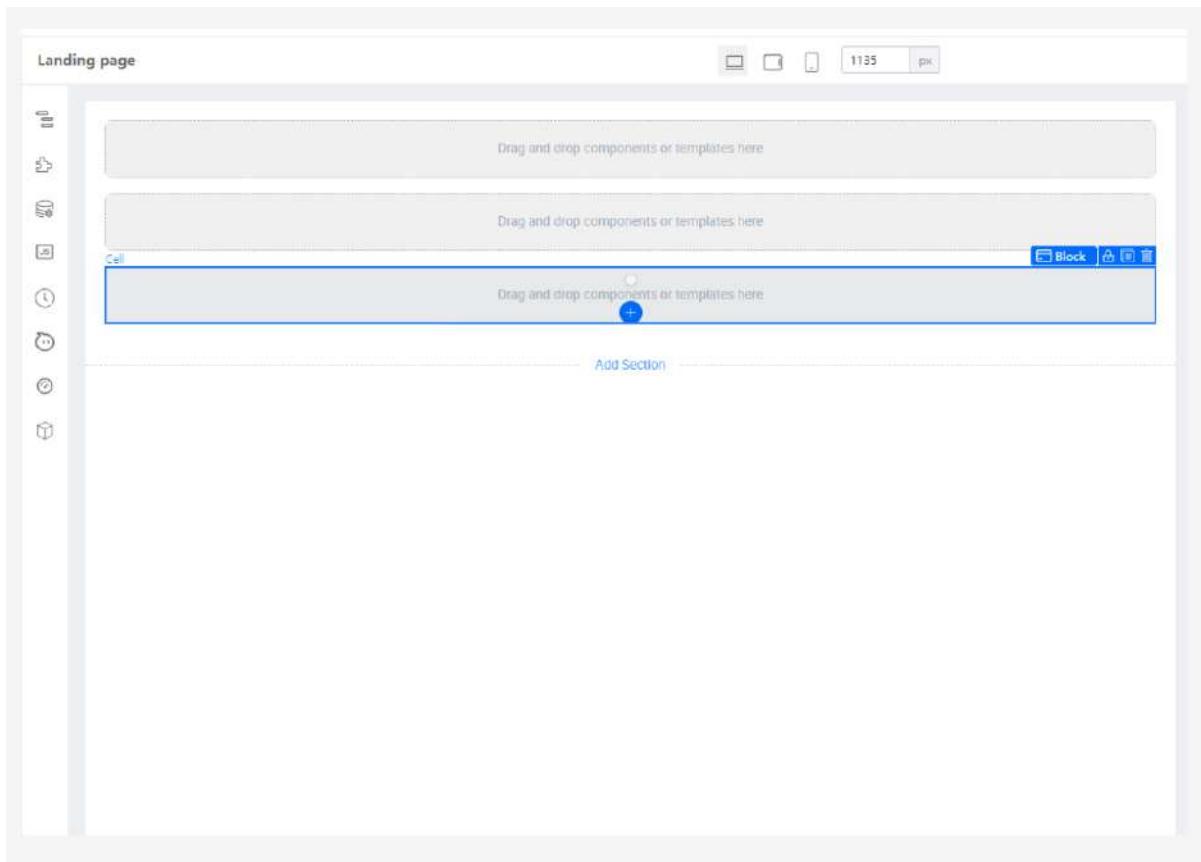
✓ Page "LandingPage" created successfully

Practical 2.1: Create Initial Layout

- Click **Add Block Below (2 times)** to add 2 more block

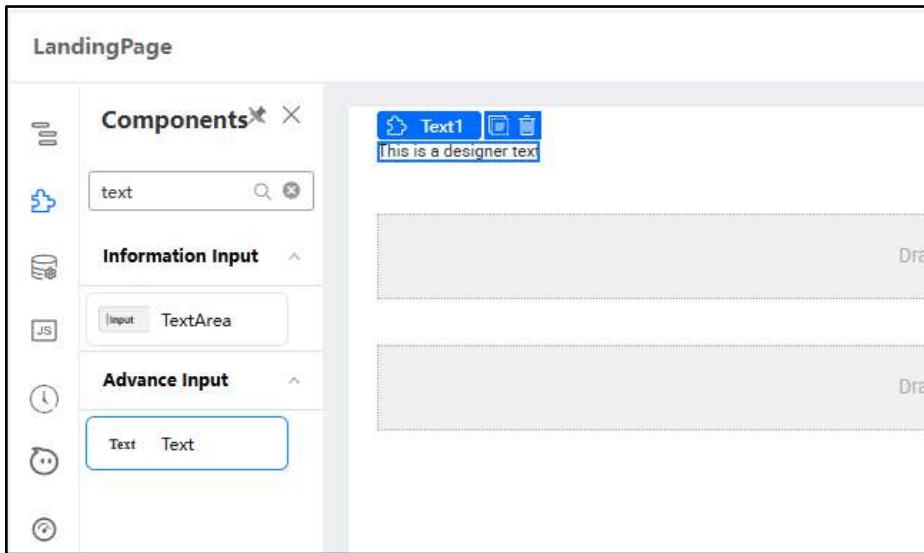


The expected result will be:



Practical 2.2: Create heading

- In the **first top Cell**, add a **Text** component



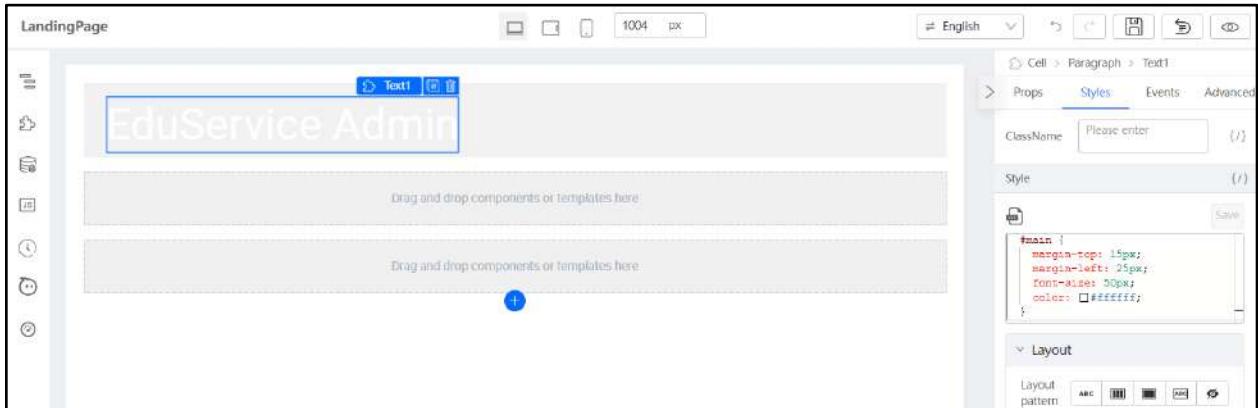
- Set the properties and styles to

Props
Text: EduService Admin

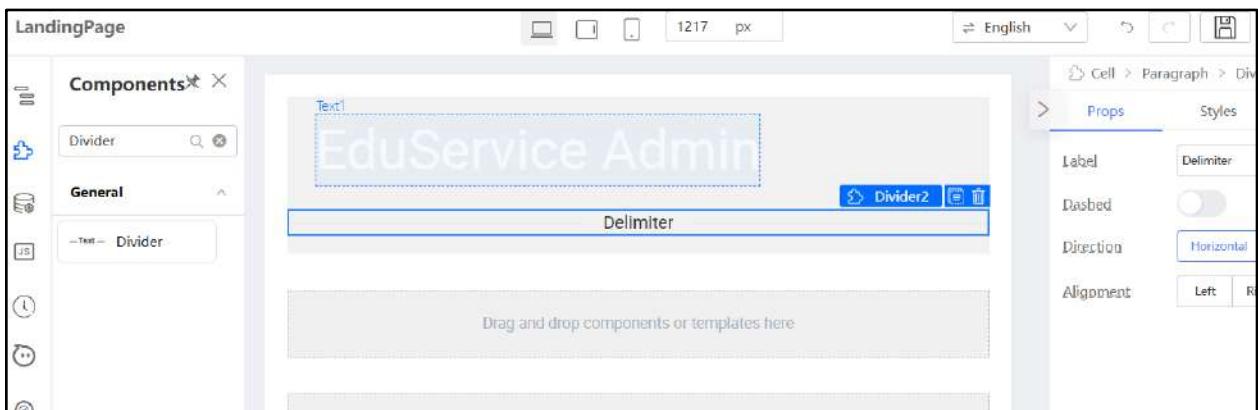
Styles
margin-top: 15px;
margin-left: 25px;
font-size: 50px;
color: #ffffff;

A screenshot of the Figma interface focusing on the properties panel. It shows the 'Props' section with the text value 'EduService Admin' and the 'Styles' section containing CSS rules for margin, font size, and color.

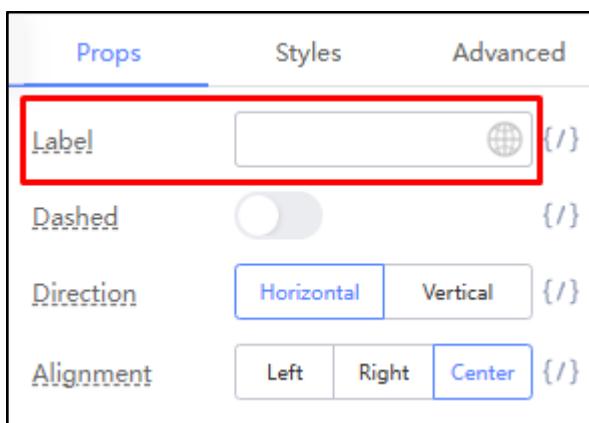
The expected result will be:



- Add a **Divider** (in the first top Cell, below the ‘EduService Admin’ text)

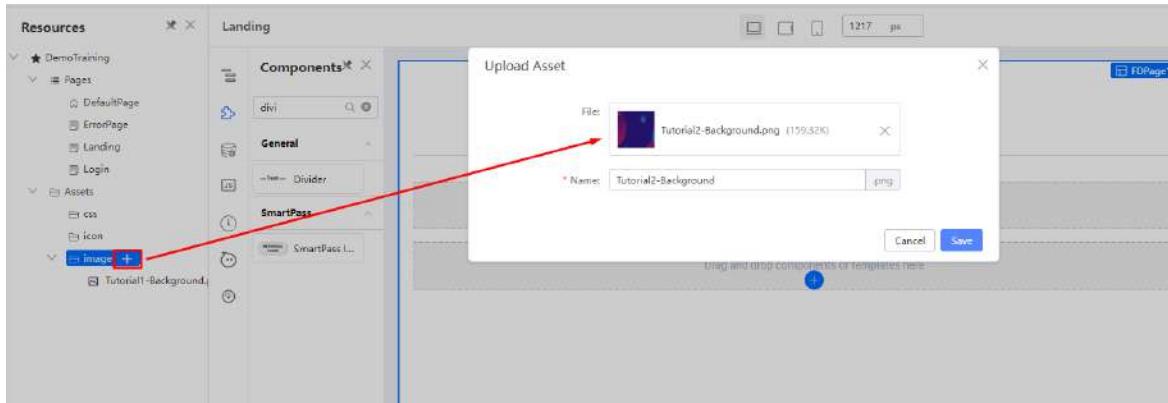


- Delete the text “Delimiter” from the Divider’s label

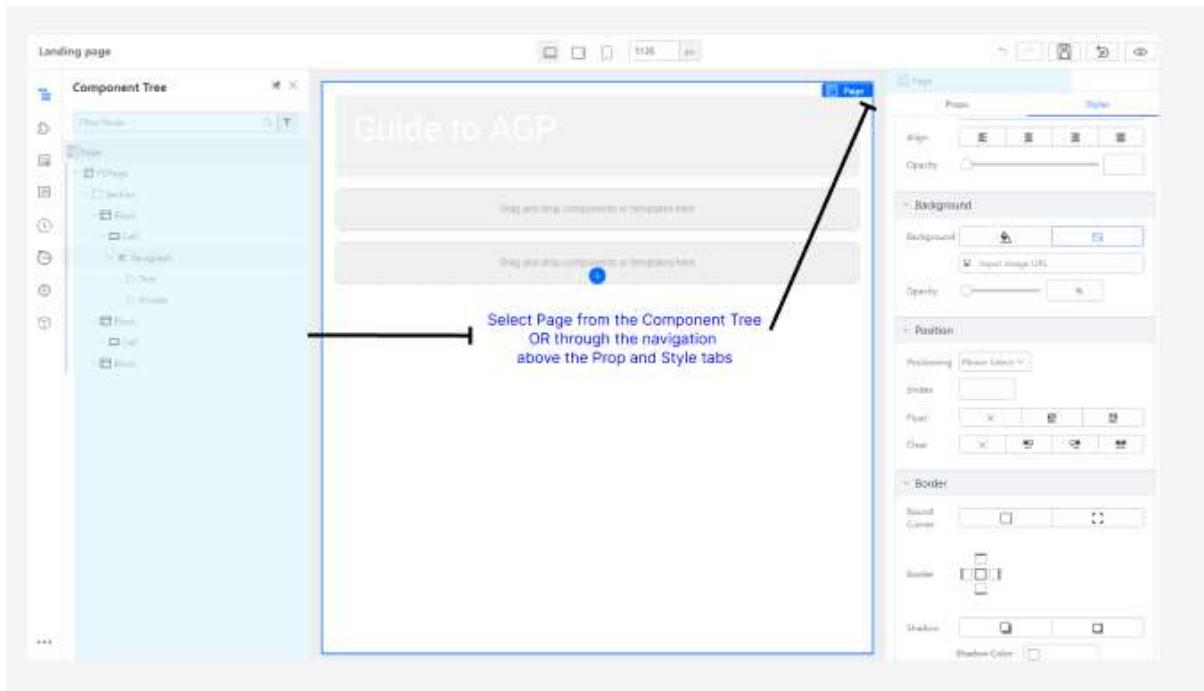


Practical 2.3: Add Image Background to root page

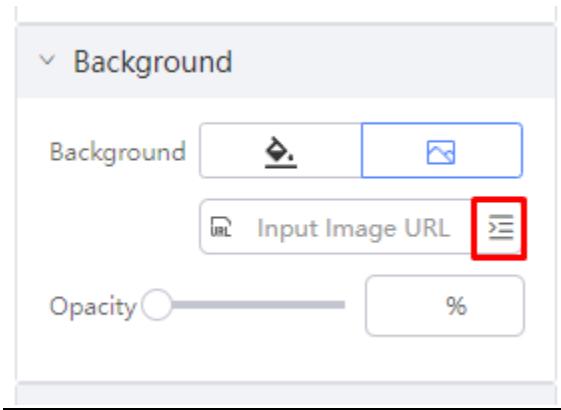
- **Download** image url: [Tutorial2-Background.png](#)
- **Upload** the image asset in the resources panel



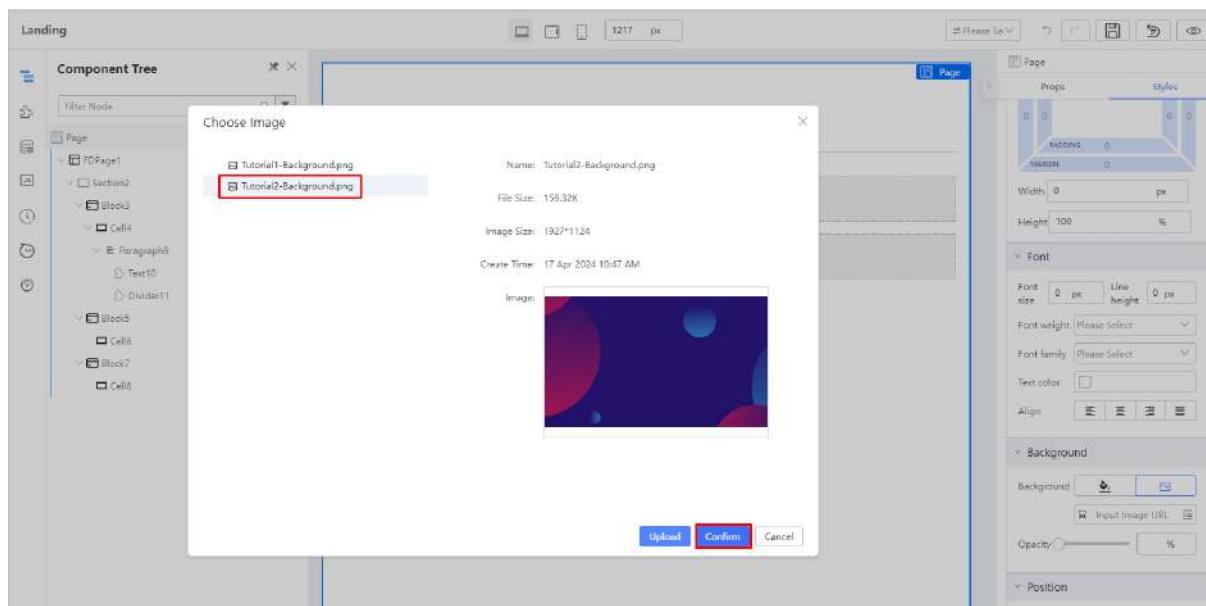
- Select **Page** (you can do via either of the following):
 - Select **Page** from the **Component Tree** OR
 - Via the **navigation** above the Prop and Style tabs



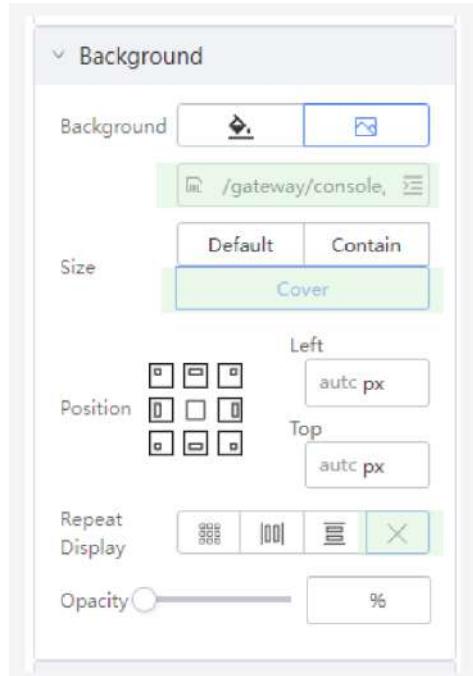
- Click on the image upload  icon



- Select on the background uploaded earlier and click **Confirm**



- Set the following styles

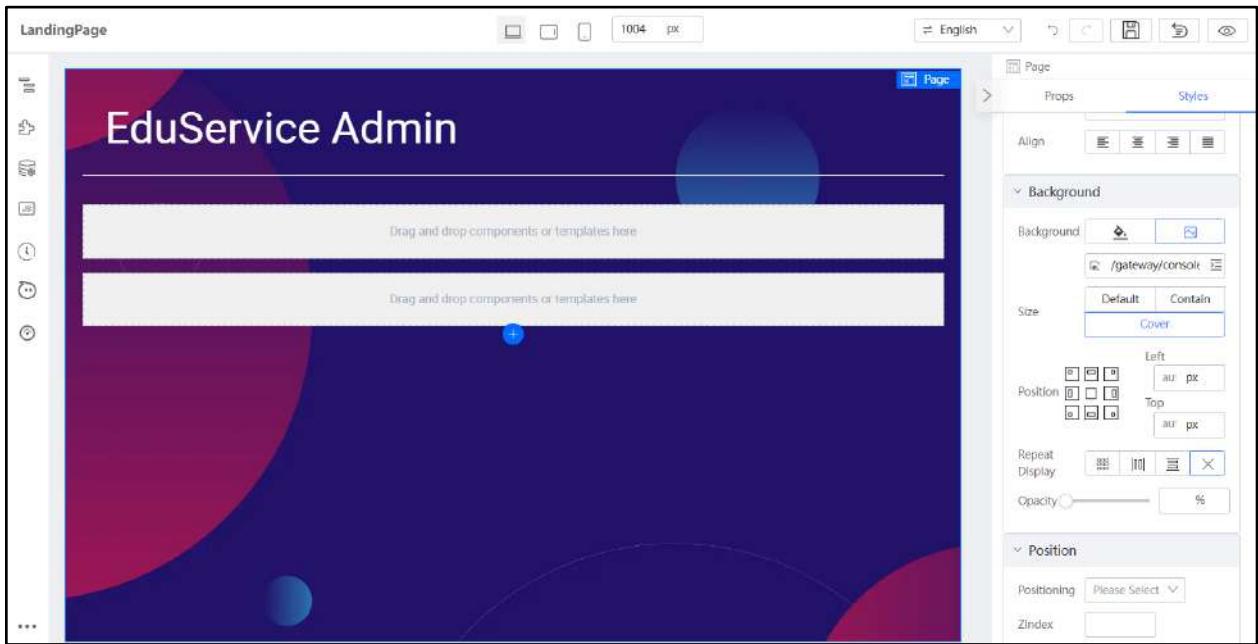


The resulting styles will be as follows:

Styles

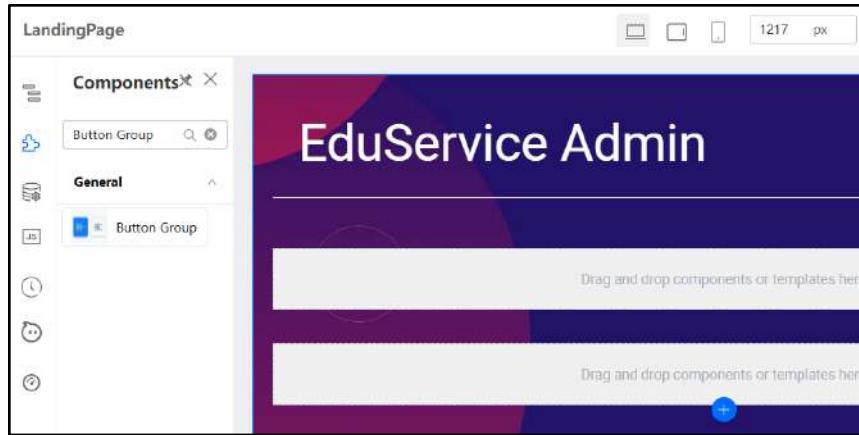
```
height: 100%;  
background-image:  
url(/gateway/console/api/v1/asset/DemoTraining/assets/image/Tutorial2-  
Background.png?branchName=main);  
background-size: cover;  
background-repeat: no-repeat;
```

The expected result will be:

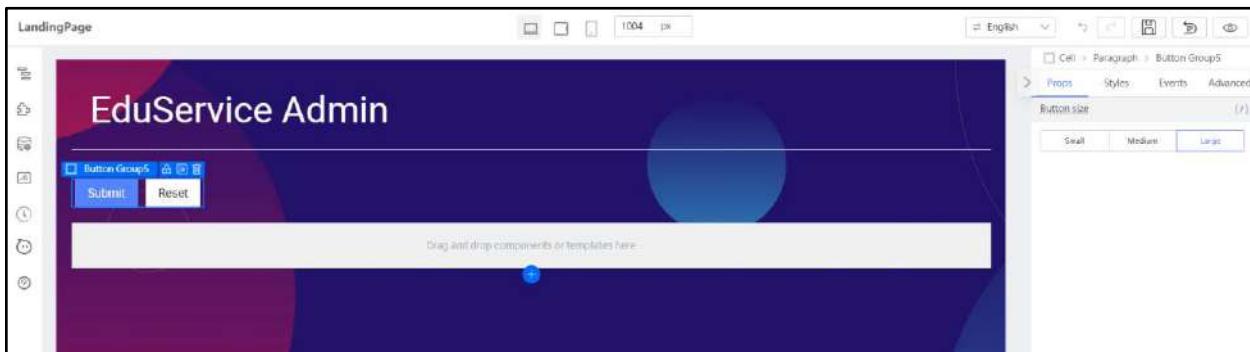
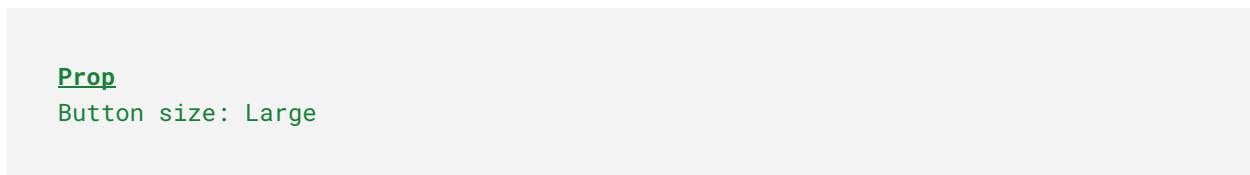


Practical 2.4: Add Button Group

- In Component Library:
 - Search for “Button Group”



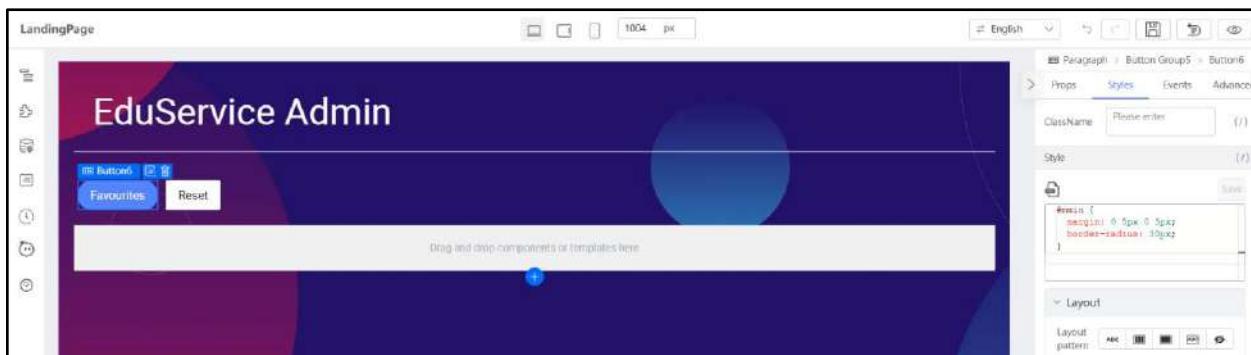
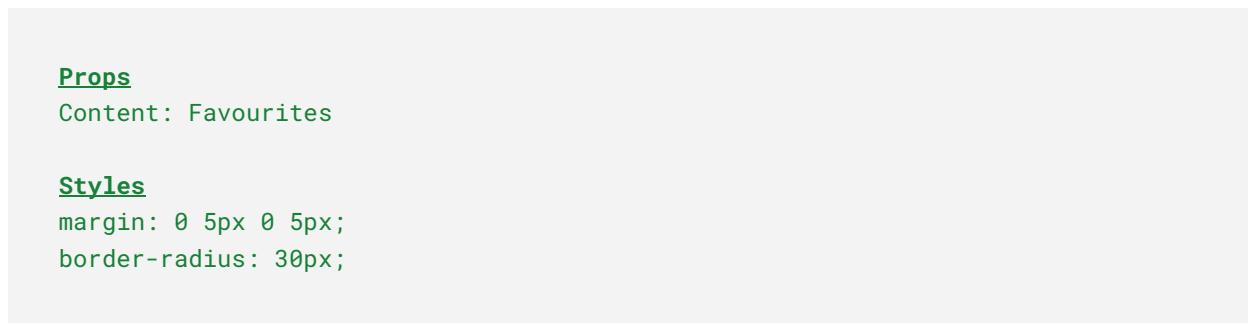
- Drag into **Block** (second block as shown in screenshot below)
- Set the Button Group with the following property



- Select the **first Button** and set

Props
Content: Favourites

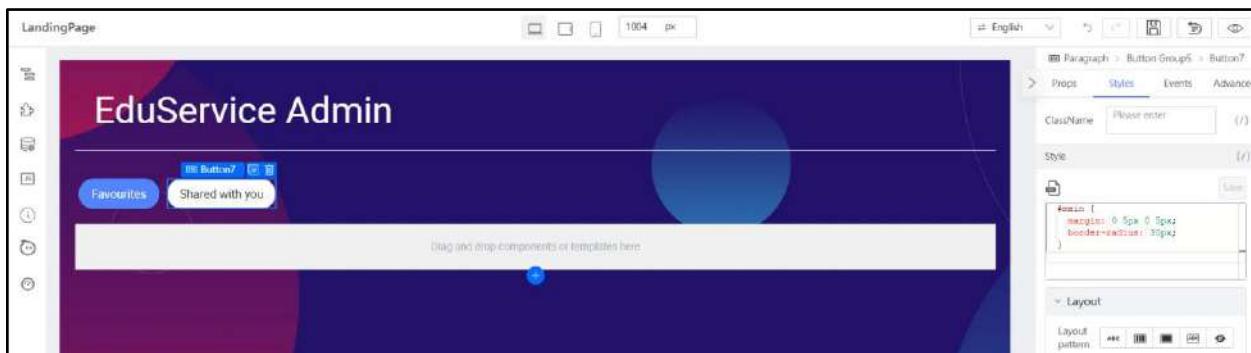
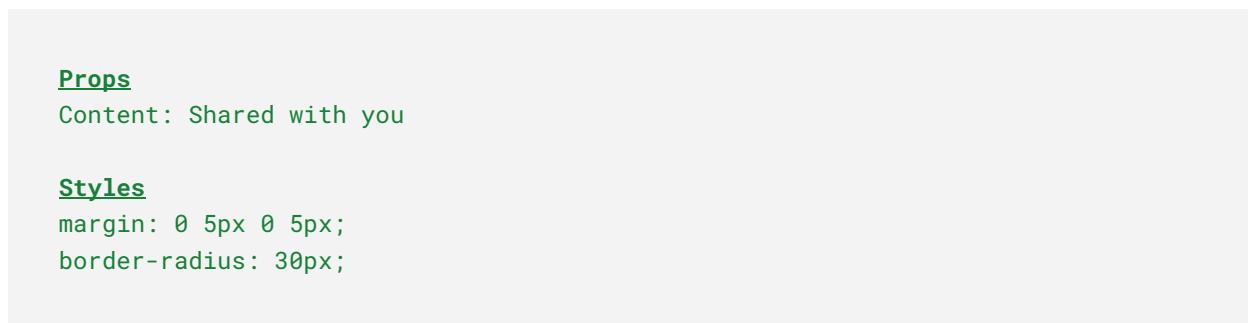
Styles
`margin: 0 5px 0 5px;
border-radius: 30px;`



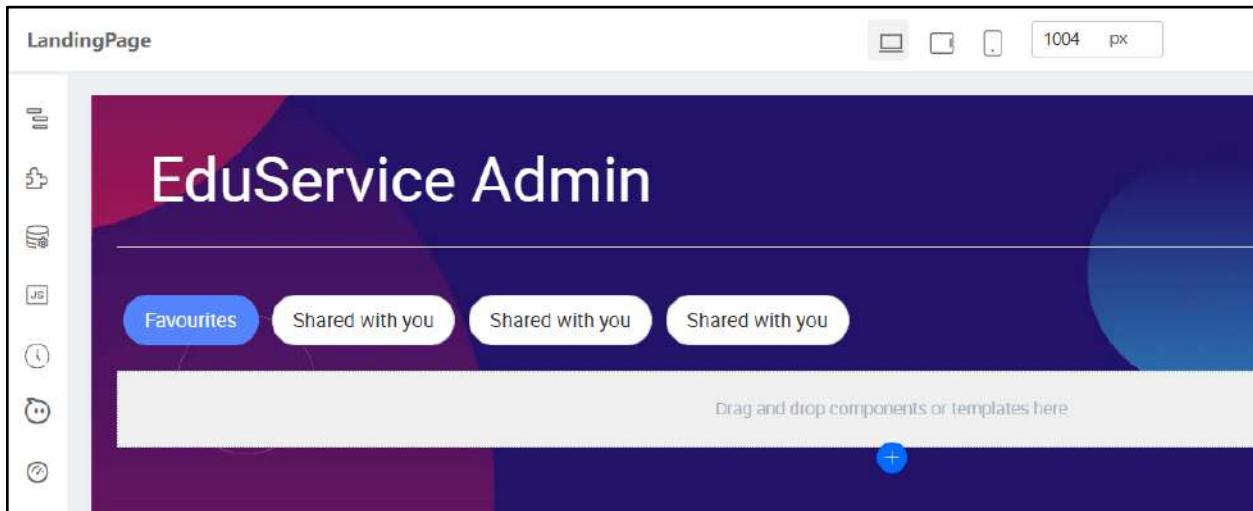
- Select the **second Button** and set

Props
Content: Shared with you

Styles
`margin: 0 5px 0 5px;
border-radius: 30px;`



- Click **Duplicate (2 times)** to duplicate 2 more buttons



- Set the third button with the following property:

Props

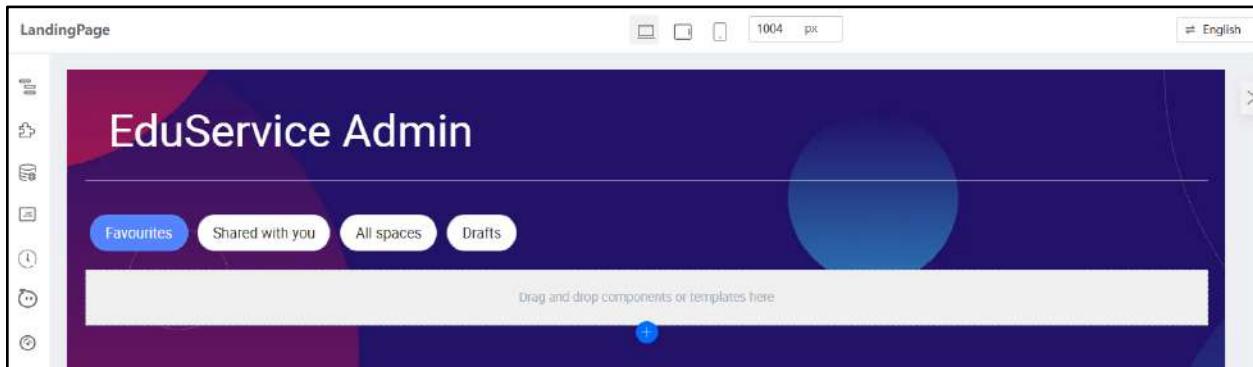
Content: All spaces

- Set the fourth button with the following property:

Props

Content: Drafts

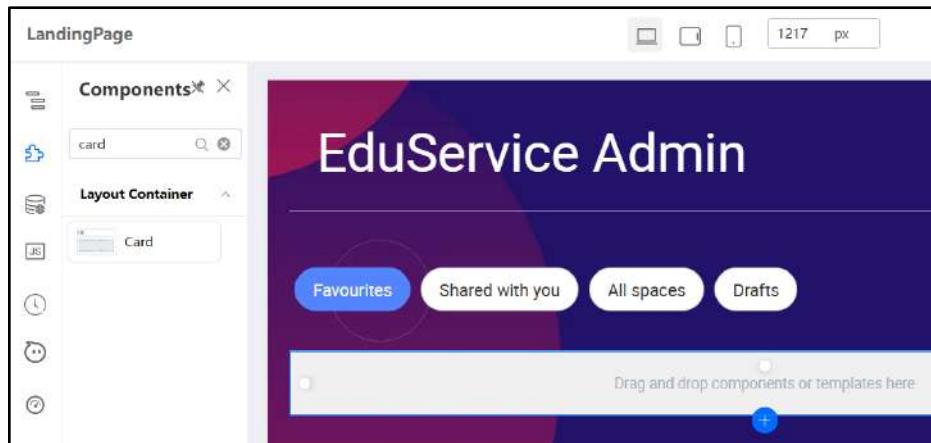
The expected result will be:



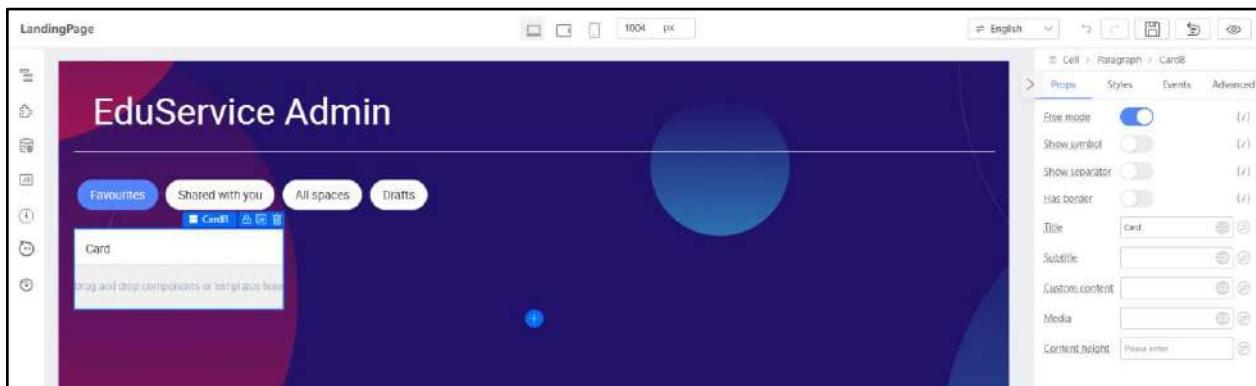
Practical 2.5: Create responsive card

In this practical, we will guide you through creating a simple, responsive card component. A responsive card adapts to different screen sizes, providing a clean and readable layout for your content on mobile, tablet, and desktop devices.

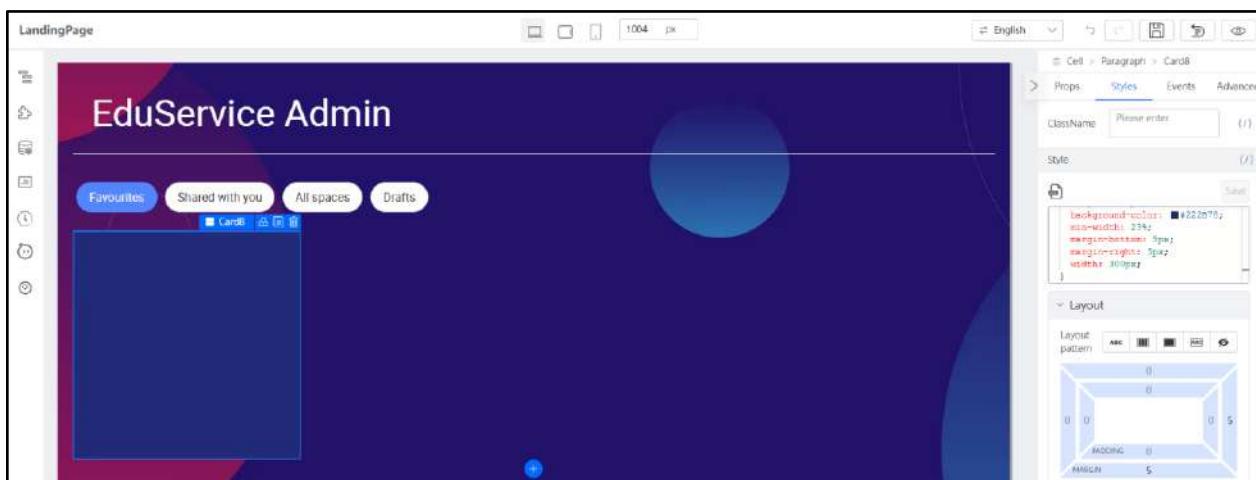
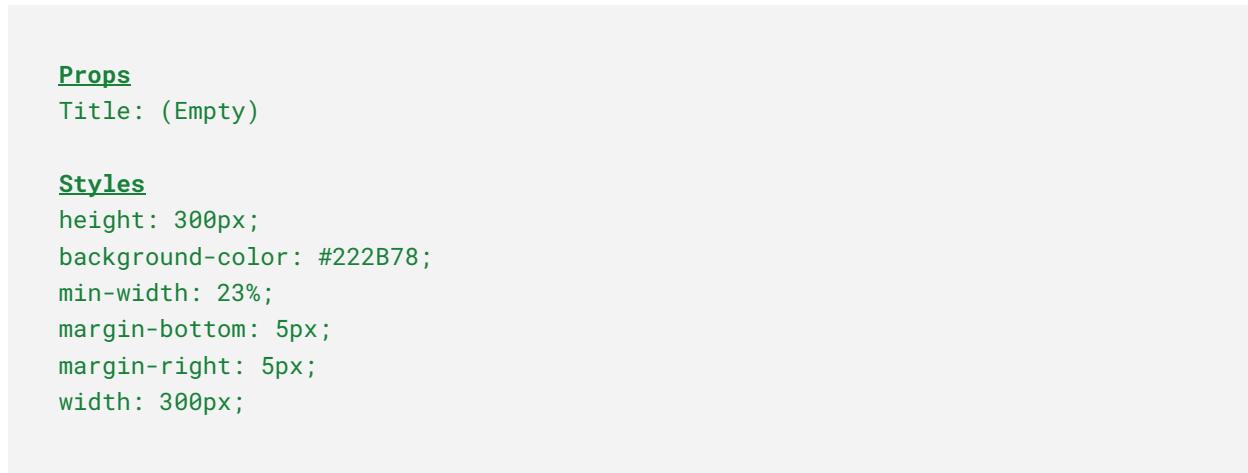
- In Component Library:
 - Search for “Card”



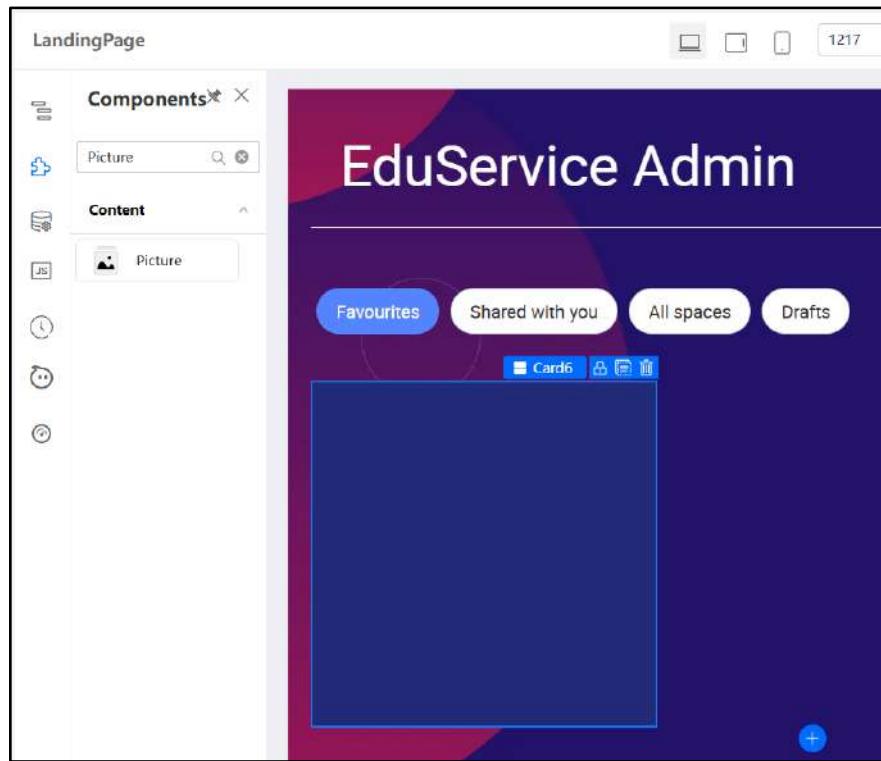
- Drag into Cell (Last cell shown in screenshot below)



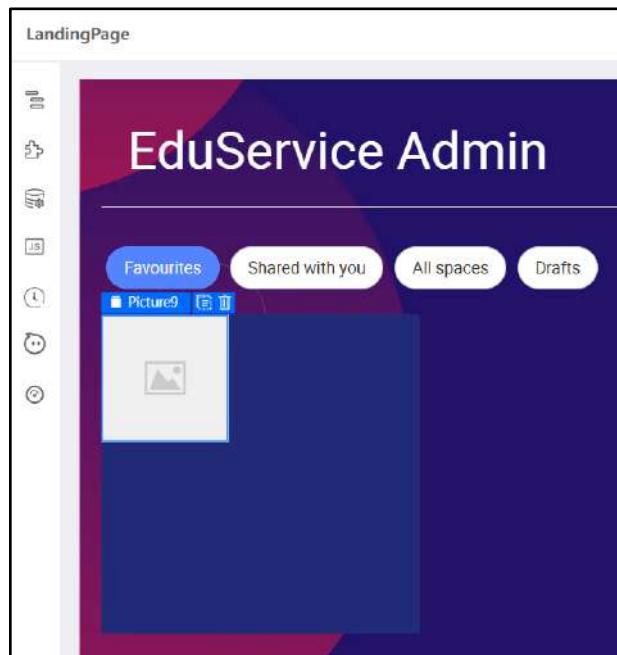
- Set the Card with the following



- In Component Library:
 - Search the “Picture” component



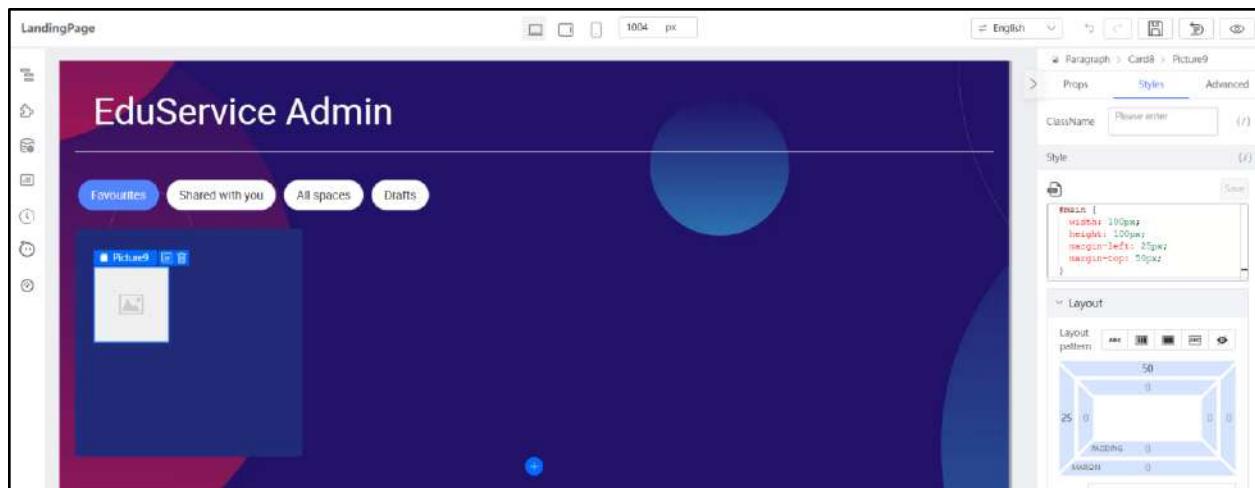
- Drag it into the **Card** component (**Blue background as shown in screenshot below**)



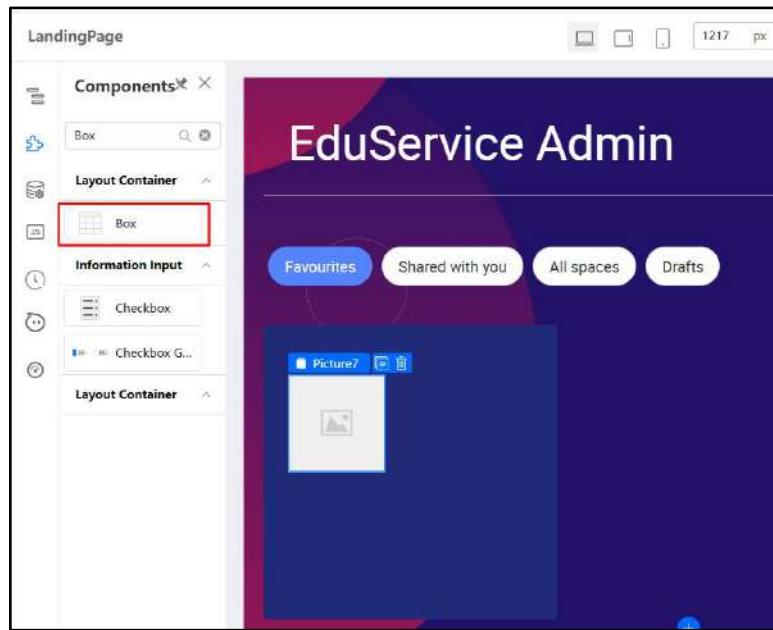
- Set the Picture component with the following:

Styles

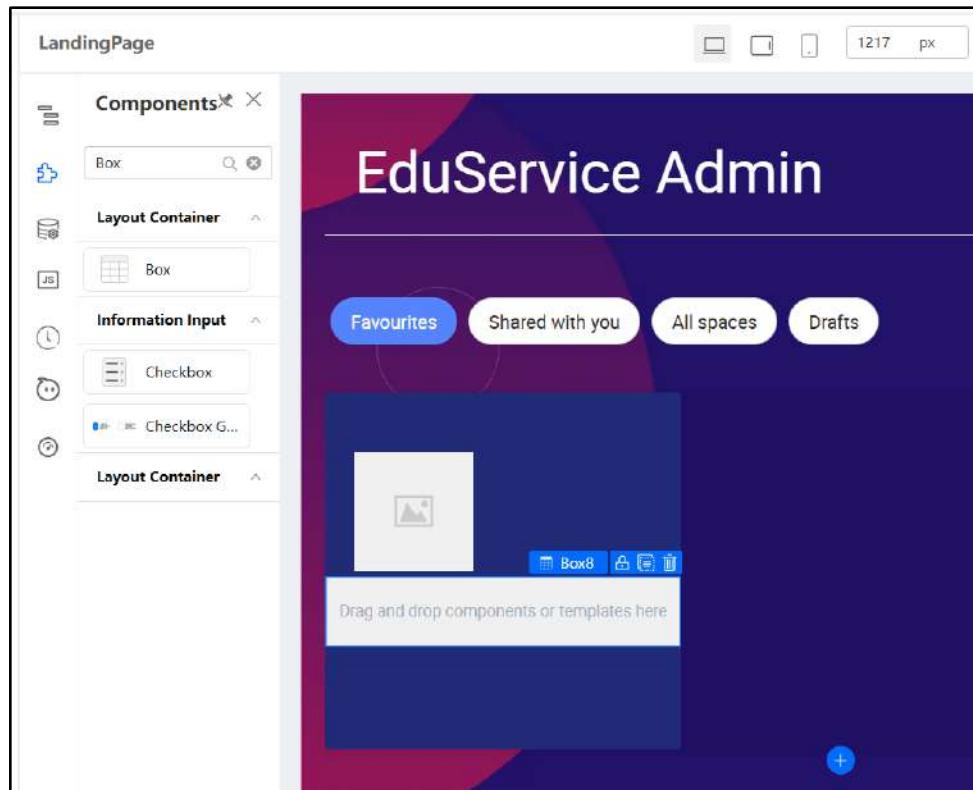
```
width: 100px;  
height: 100px;  
margin-left: 25px;  
margin-top: 50px;
```



- In Components Library:
 - Search for **Box** component



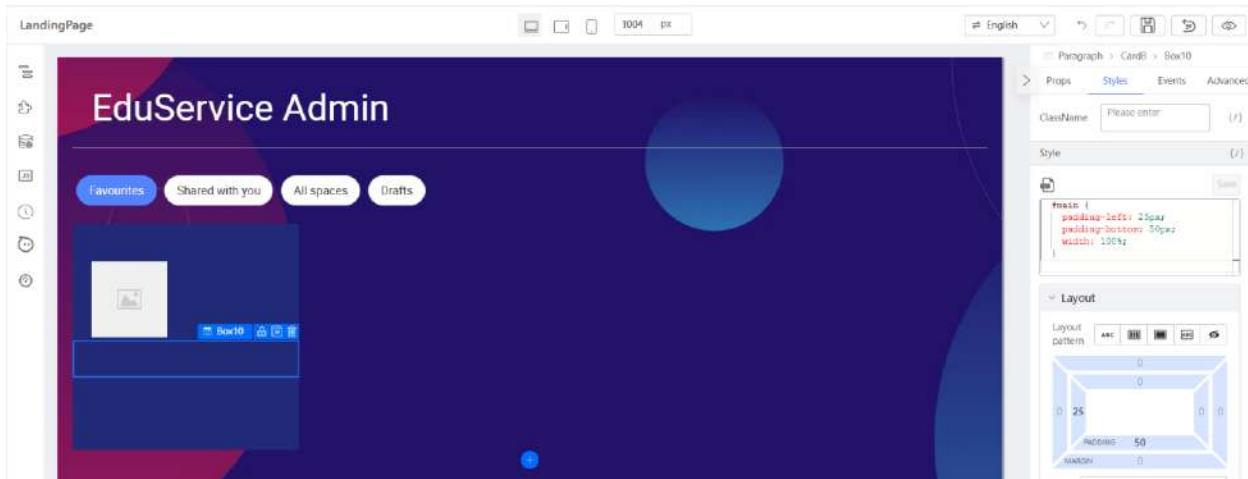
- Drag a **Box** below the Picture component



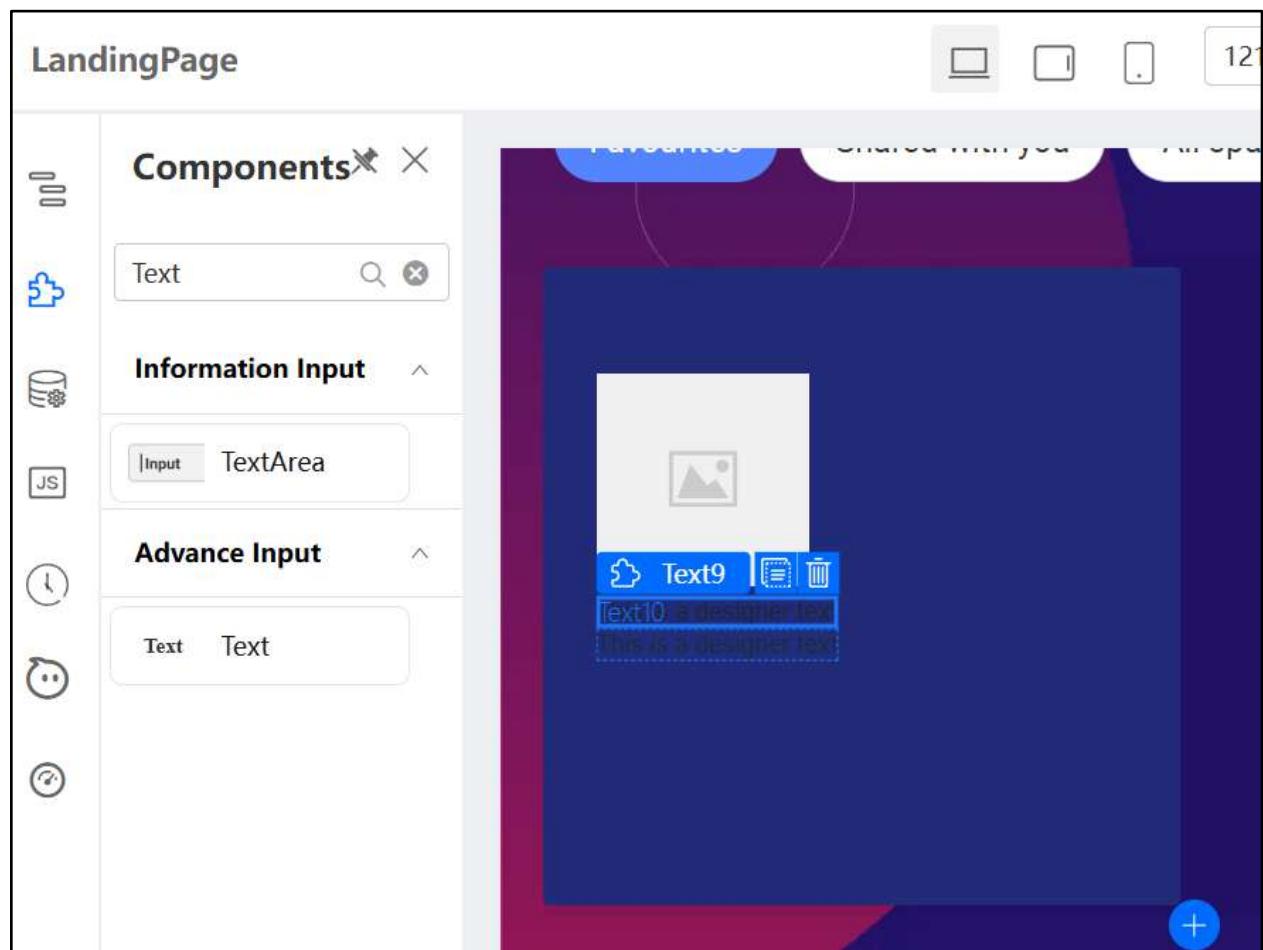
- Set the Box with the following:

```
Props
Justify: flex-start
Align: flex-start

Styles
padding-left: 25px;
padding-bottom: 50px;
width: 100%;
```



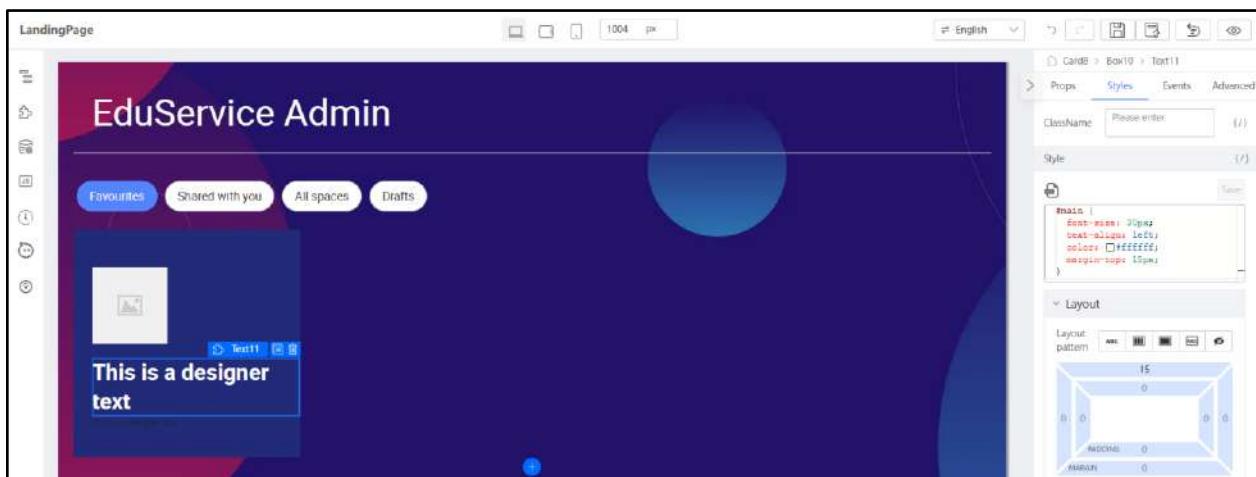
- Drag **Text** component (2 times) into the **Box** (illustrated below)



- Set the **first Text** with the following:

```
Props
Strong: true

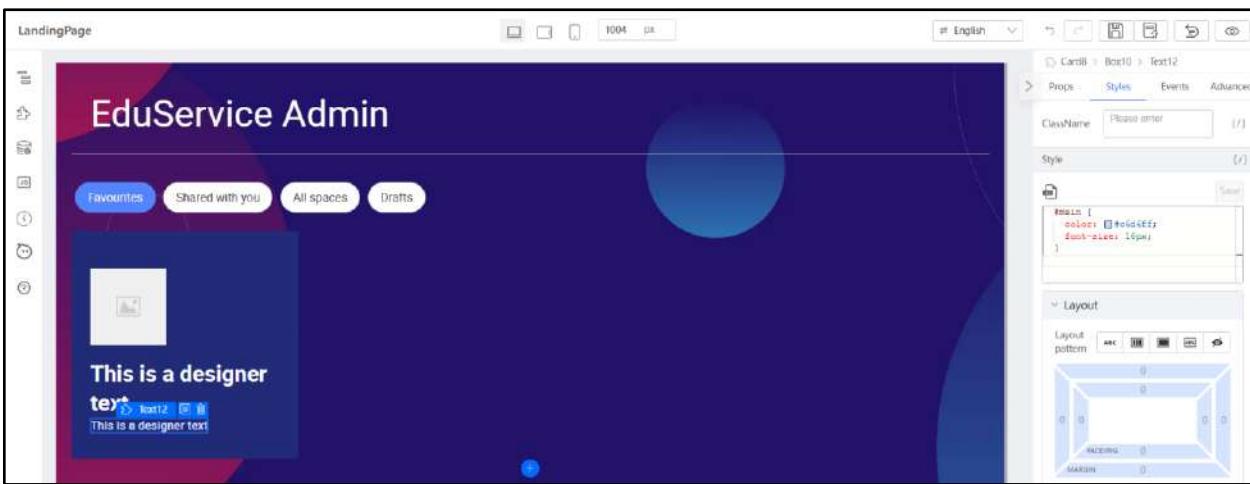
Styles
font-size: 30px;
text-align: left;
color: #ffffff;
margin-top: 15px;
```



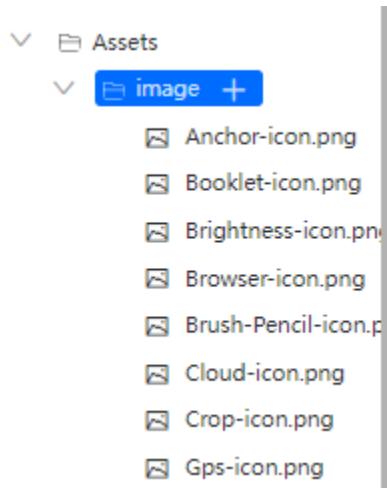
- Set the **second Text** with the following:

```
Props
Strong: true

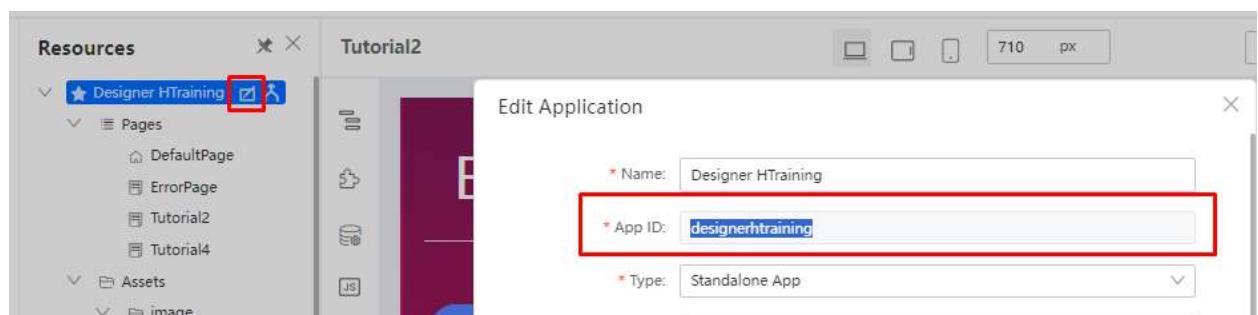
Styles
color: #c6d6ff;
font-size: 16px;
```



- **Download** these 8 images from this folder: [Tutorial2-images](#)
- **Upload** these images into asset in the resources panel



- **Download** [card-json-data.txt](#), edit all the image path to **your app id** as located here.



- Edit the image path

E.g

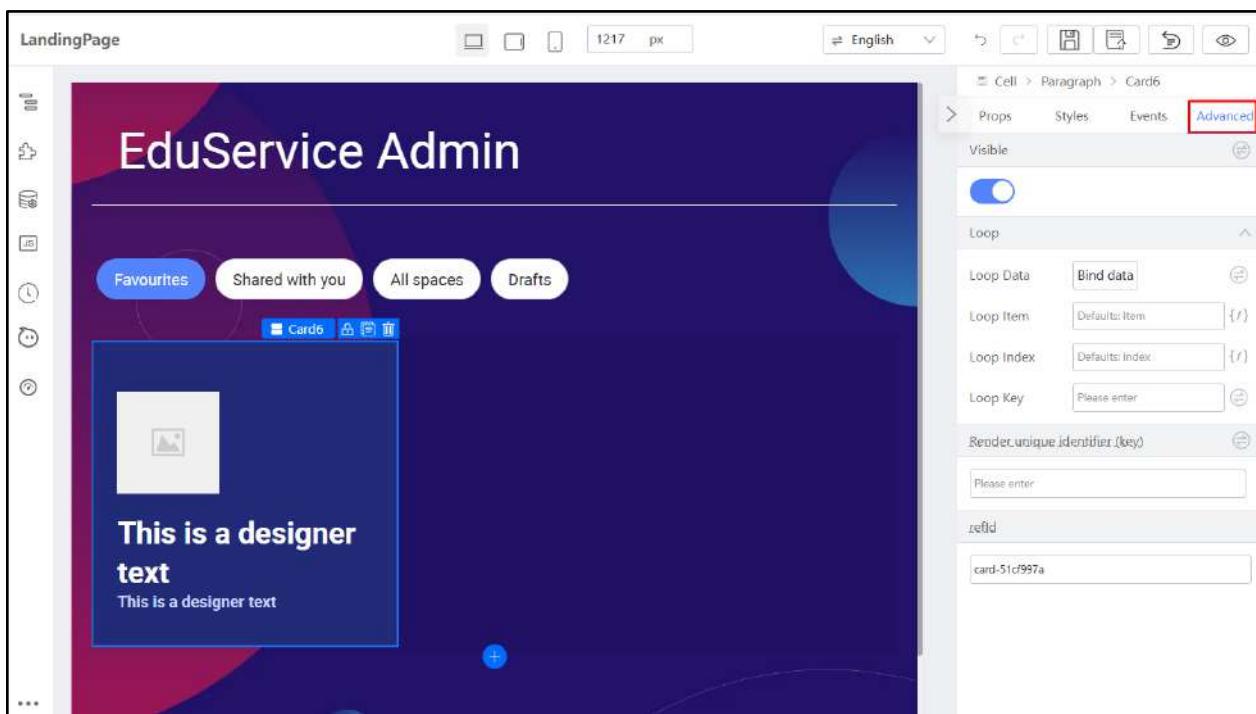
/gateway/console/api/v1/asset/{yourapp-id}/assets/image/Anchor-icon.png?branchName=main

```

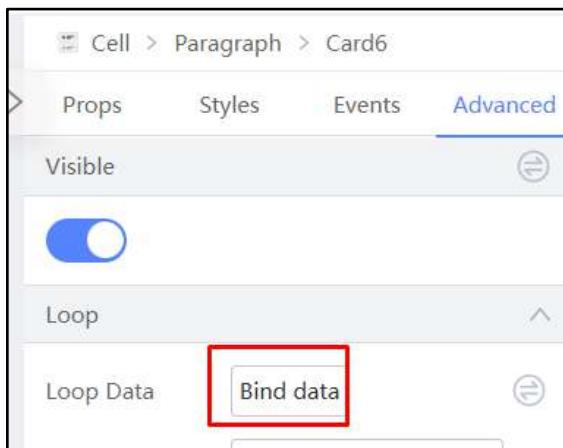
[{"imageUrl": "/gateway/console/api/v1/asset/username/assets/image/Anchor-icon.png?branchName=main", "heading": "Student Enrollment", "subHeading": "Course Registration"}, {"imageUrl": "/gateway/console/api/v1/asset/username/assets/image/Grade Management", "heading": "Grade Management", "subHeading": "Performance Tracking"}, {"imageUrl": "/gateway/console/api/v1/asset/username/assets/image/Class Scheduling", "heading": "Class Scheduling", "subHeading": "Timetable Management"}, {"imageUrl": "/gateway/console/api/v1/asset/username/assets/image/Resource Allocation", "heading": "Resource Allocation", "subHeading": "Classroom and Material Assignment"}, {"imageUrl": "/gateway/console/api/v1/asset/username/assets/image/Cloud-icon.png?branchName=main", "heading": "Financial Administration", "subHeading": "Tuition and Fee Management"}, {"imageUrl": "/gateway/console/api/v1/asset/username/assets/image/Crop-icon.png?branchName=main", "heading": "Communication Hub", "subHeading": "Announcements and Updates"}, {"imageUrl": "/gateway/console/api/v1/asset/username/assets/image/Gps-icon.png?branchName=main", "heading": "Staff Management", "subHeading": "Teacher and Administrator Roles"}, {"imageUrl": "/gateway/console/api/v1/asset/username/assets/image/Browser-icon.png?branchName=main", "heading": "Assessment and Evaluation", "subHeading": "Exam and Grading System"}]

```

- Select the **Card component** and navigate to **Advanced (Top Right Corner)** tab



- Click on Bind data



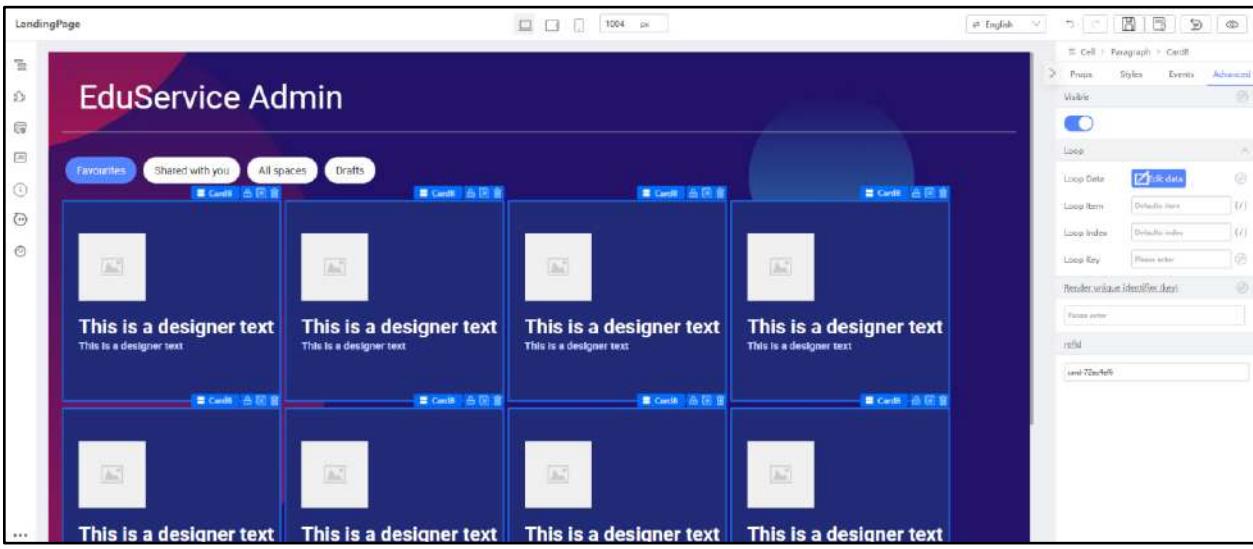
- In the pop-up, update the data with the following value (**card-json-data.txt** below):

```

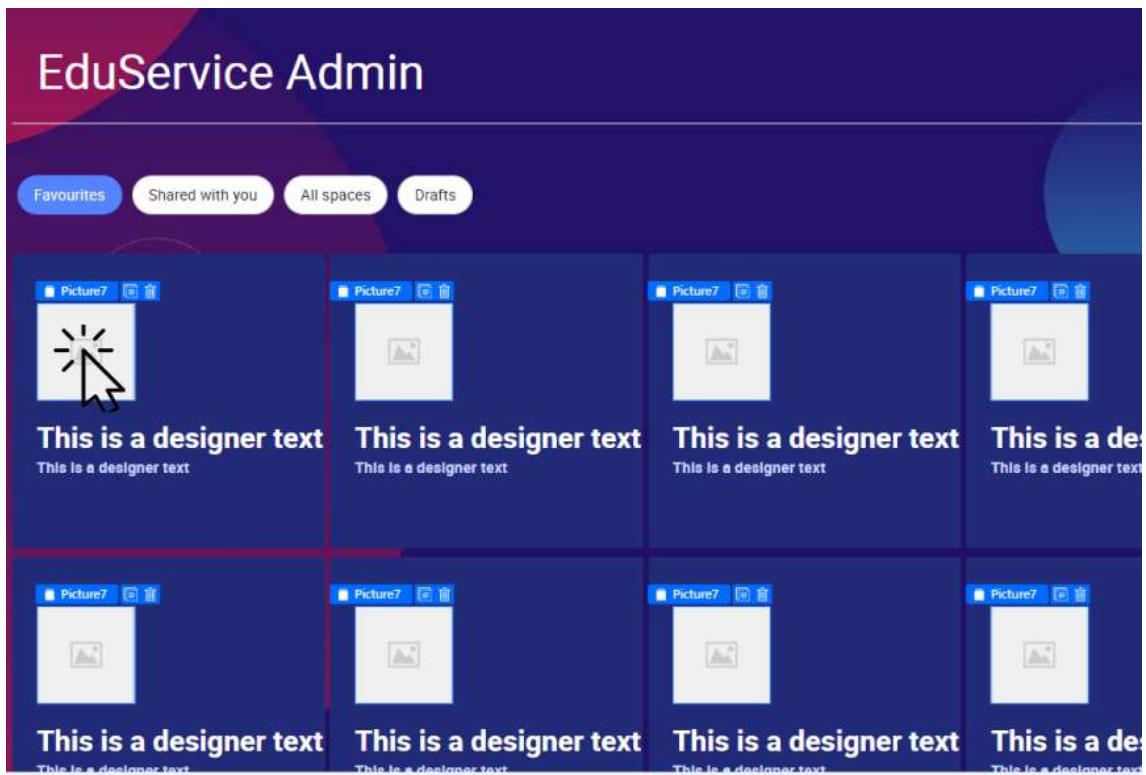
19   "heading": "Resource Allocation",
20   "subHeading": "Classroom and Material Assignment"
21 },
22 {
23   "imageUrl": "https://icons.iconarchive.com/icons/elegant-
24   "heading": "Financial Administration",
25   "subHeading": "Fees and Fee Management"
26 },
27 {
28   "imageUrl": "https://icons.iconarchive.com/icons/elegant-
29   "heading": "Communication Hub",
30   "subHeading": "Announcements and Updates"
31 },
32 {
33   "imageUrl": "https://icons.iconarchive.com/icons/elegant-
34   "heading": "Staff Management",
35   "subHeading": "Teacher and Administrator Roles"
36 },
37 {
38   "imageUrl": "https://icons.iconarchive.com/icons/elegant-
39   "heading": "Assessment and Evaluation",
40   "subHeading": "Exam and Grading System"
41 },
42

```

The expected result will be:



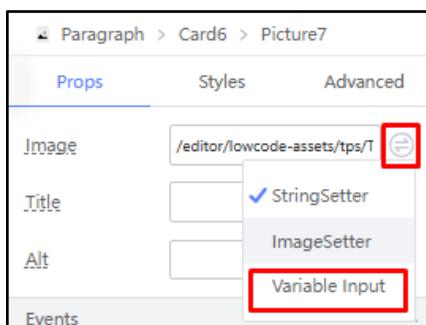
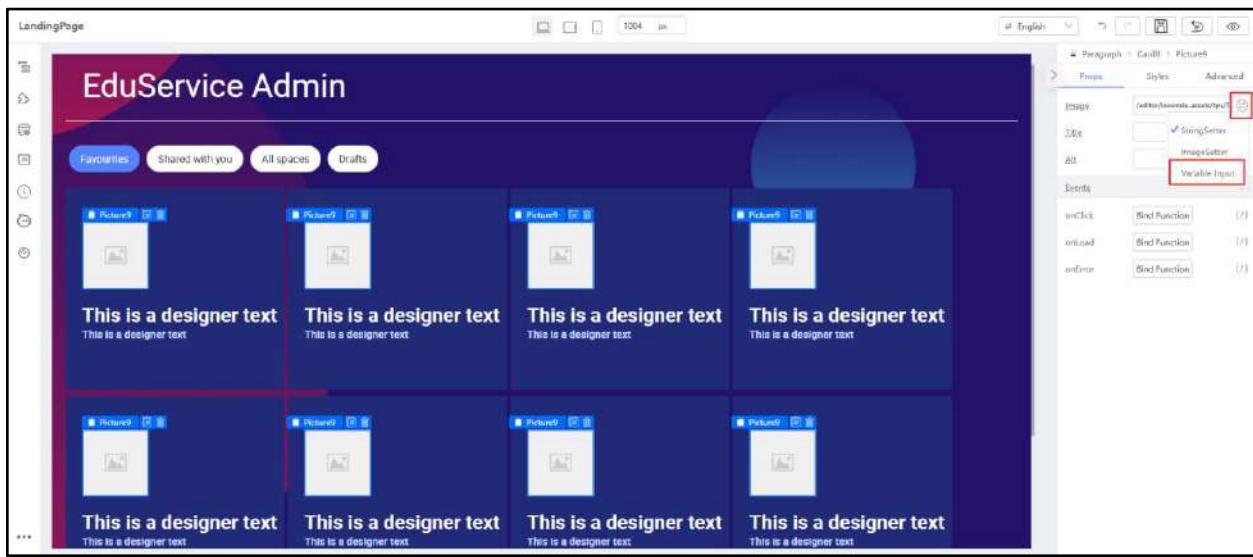
- Select **Picture component** by clicking on it



- Under **Image** property, click on **Switch Setter icon** 



- In the dropdown, select **Variable Input** to bind data:

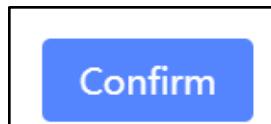
- In the popup box, update the value “`this.item.imageUrl`” under **Bind variables**

Props

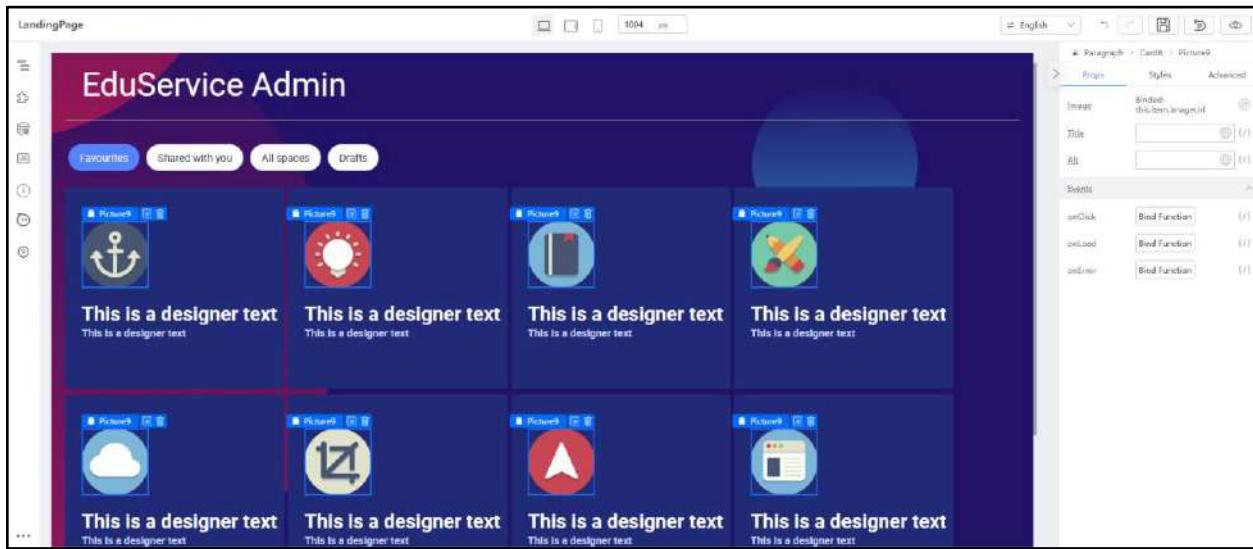
Image (**Variable binding**): `this.item.imageUrl`

The screenshot shows the EduService Admin interface with a dark blue header. Below it, there's a sidebar with 'Favourites' and 'Shared' buttons. The main area displays a document thumbnail labeled 'Picture7' with a small image icon. Below the thumbnail, the text 'This is a doc' is visible. On the right, a 'Variable Binding' dialog is open. It has a 'Variable List' section with tabs for 'State attribute', 'Customized func', and 'Datasource', and a search bar. The 'Bind variables' section contains a code editor with the line '1 `this.item.imageUrl`'. Below the code editor is a 'Usage' section with explanatory text and two variable descriptions: 'this: The instance object of the current block' and 'this.state: Data object state of page, block, widget component'.

- Click **Confirm** to update the value



The expected result will be:



- Select the **Heading Textbox** and click on **Variable Binding**

The screenshot shows a user interface for a UI builder or design tool. At the top, there is a toolbar with icons for saving, undo, redo, and delete. Below the toolbar, a header bar displays the path: Card6 > Box8 > Text9. The main area contains a dark blue card with a circular icon containing two pencils. Below the card, there is another text component labeled "Text9" with a similar toolbar. The text "This is a designer text" is displayed twice. To the right of the card, a sidebar titled "Props" is open, showing various styling options. A tab labeled "Variable Binding" is selected, indicated by a red box around its border. The "Text" field in the props panel also has a red box around its border. Other visible settings include "Font size: Default", "Highlight: Off", "Code inline: Off", "Strikethrough: Off", "Underline: Off", and "Strong: On".

- Set the following property

Props
Text (Variable binding): `this.item.heading`

Variable Binding

Variable List

Bind variables

State attribute Search

Customized func

Datasource

Usage

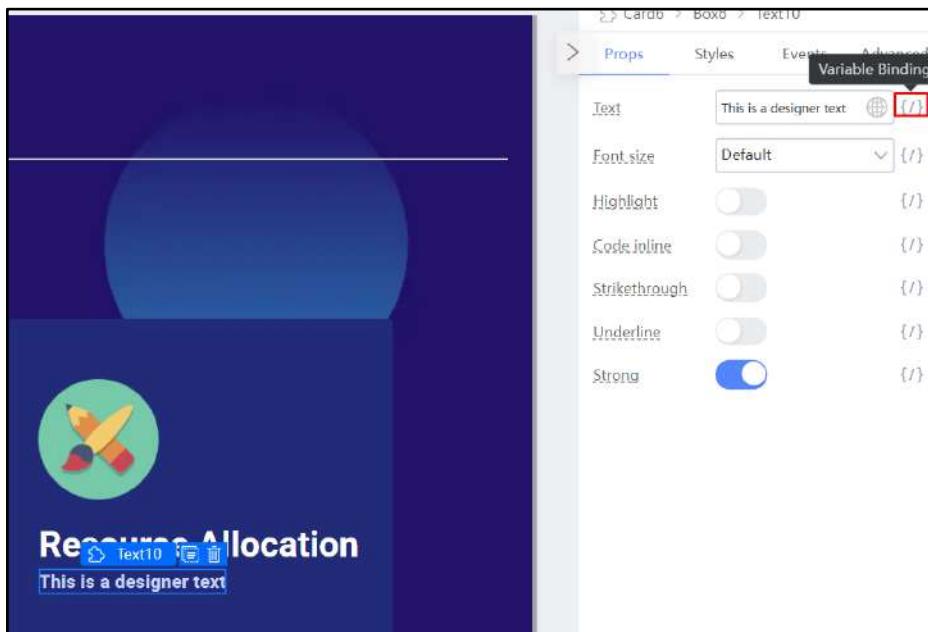
You can bind variables or functions by clicking on the left area or entering expressions above. The input box supports variables and uses JS syntax.

this: The instance object of the current block
this.state: Data object state of page, block, widget component

Props Styles Events Advanced
Binded: this.item.heading

This is a designer text

- Select the sub-heading Textbox and set the following:



Props

Text (Variable binding): this.item.subHeading

Variable Binding

Bind variables

```
1 this.item.subHeading
```

Usage

You can bind variables or functions by clicking on the left area or entering expressions above. The input box supports variables and uses JS syntax.

this: The instance object of the current block
this.state: Data object state of page, block, widget component

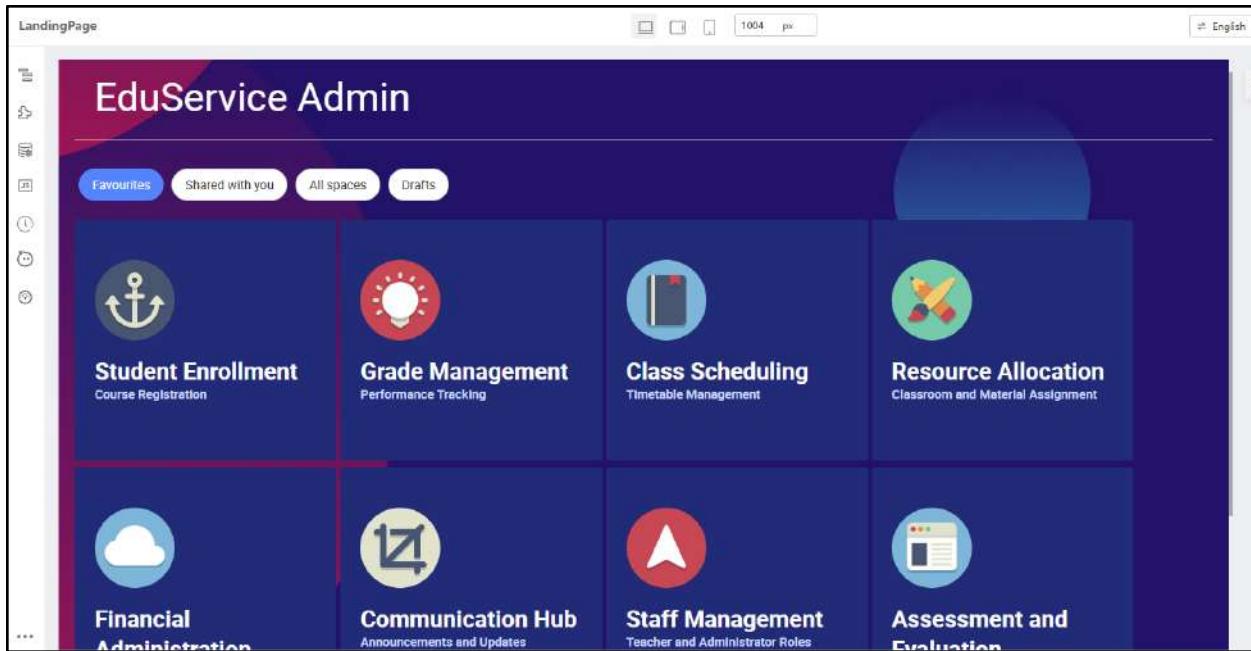
Props

Binded: this.item.subHeading

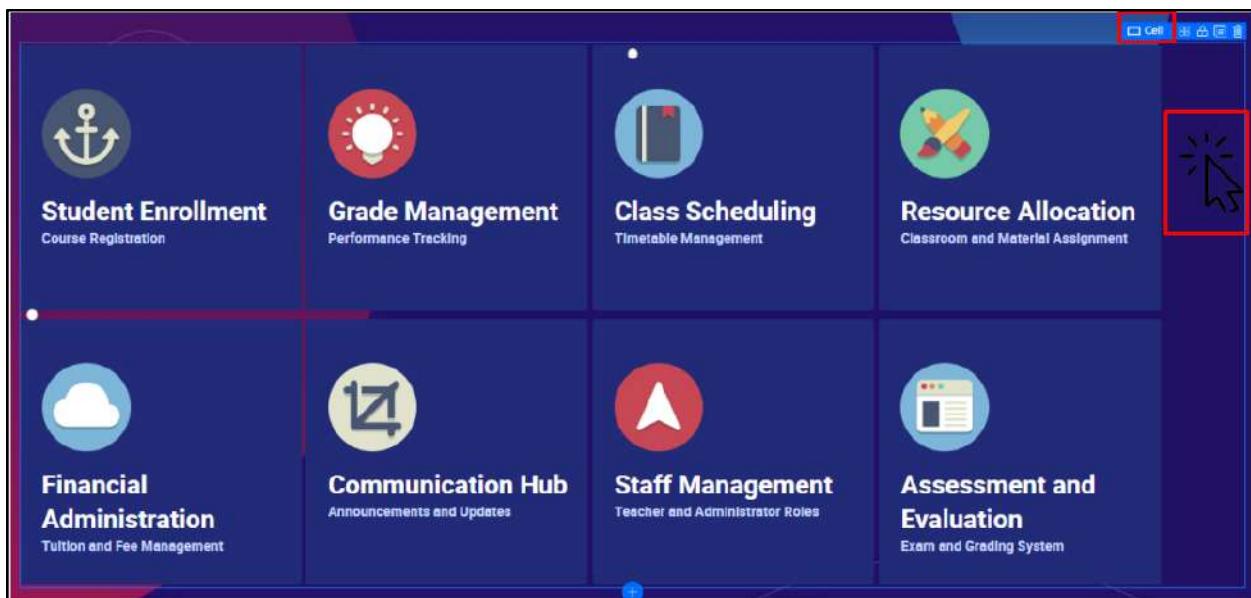
Text

Remove Binding

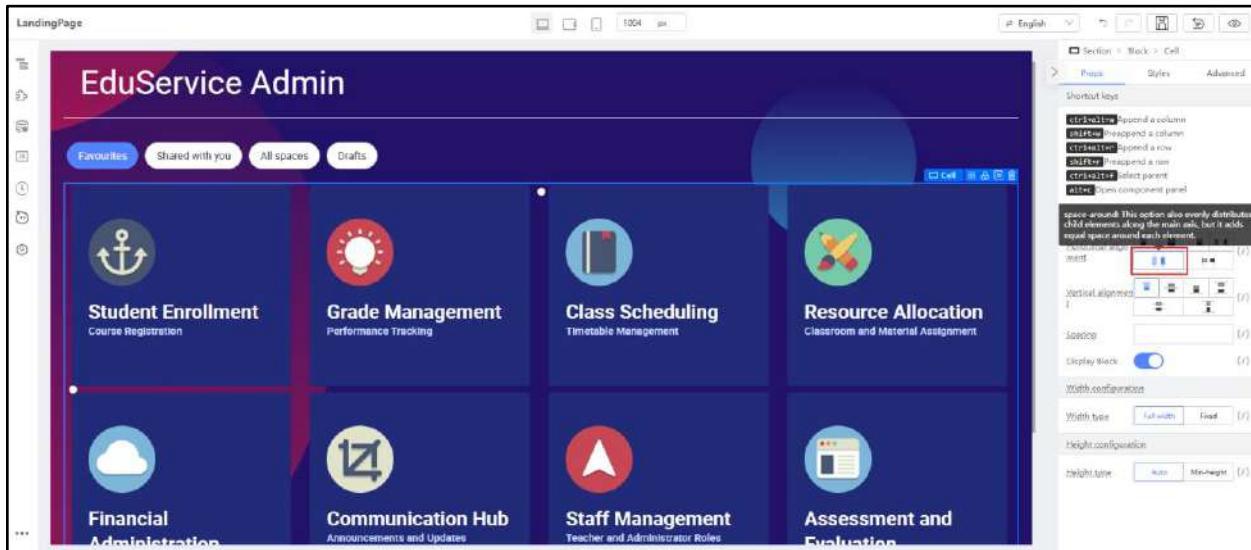
The expected result will be:



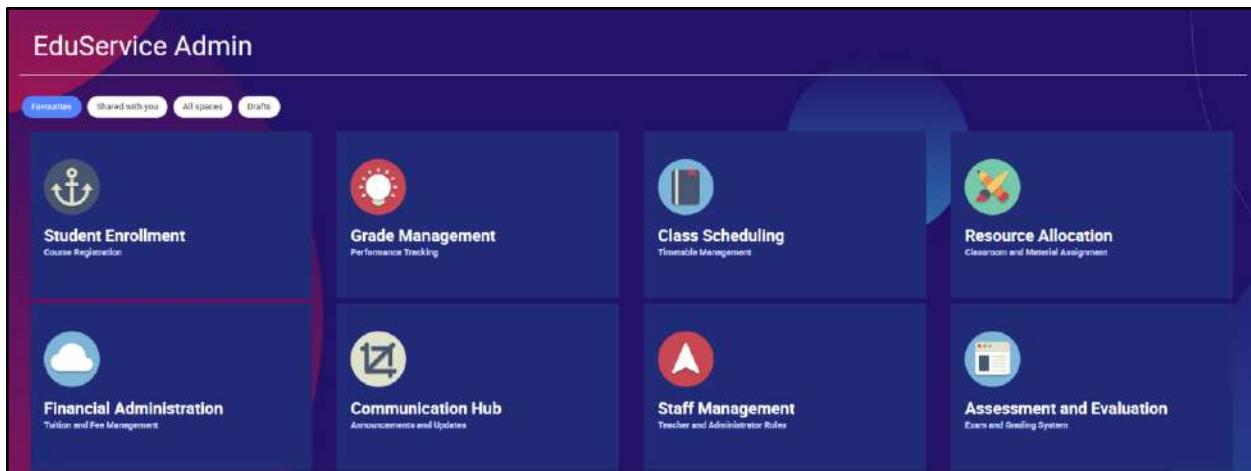
- Click on the Container Area and ensure Cell is selected



- Set the following properties



The expected result will be:



Tutorial 3: Creating the Listing Page (Optional)

This tutorial covers the following Learning Objectives:

- Understand how to design and implement a basic listing page using KAIZEN, utilizing the table component to display data effectively.
- Learn how to structure and customize the table to organize and present information in a clear and user-friendly manner.

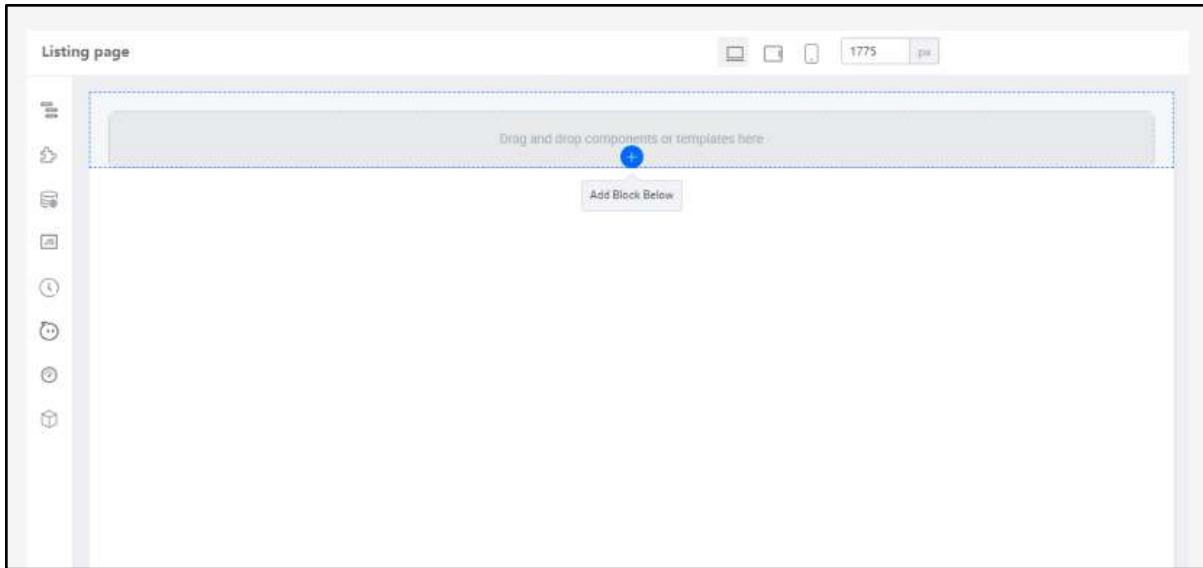
In this tutorial, we will guide you through the process of building a listing page using KAIZEN. By utilizing the table component, you will learn how to effectively display and organize data in a table. This tutorial emphasizes creating a user-friendly, dynamic listing page that allows users to view and interact with large sets of data efficiently.

The screenshot displays a user interface for a learning platform. At the top, there is a navigation bar with the text "Dashboard - Course Listing". Below the navigation bar, there are several search and filter options: "Search Name", "Filter by course type", "Status", and "Rating/Review". A "Create New Course" button is also visible. The main content area is titled "Course Listing" and shows a grid of three course cards. Each card includes a thumbnail image, the course title, a brief description, a rating, and two buttons: "Enroll Now" and "View Details".

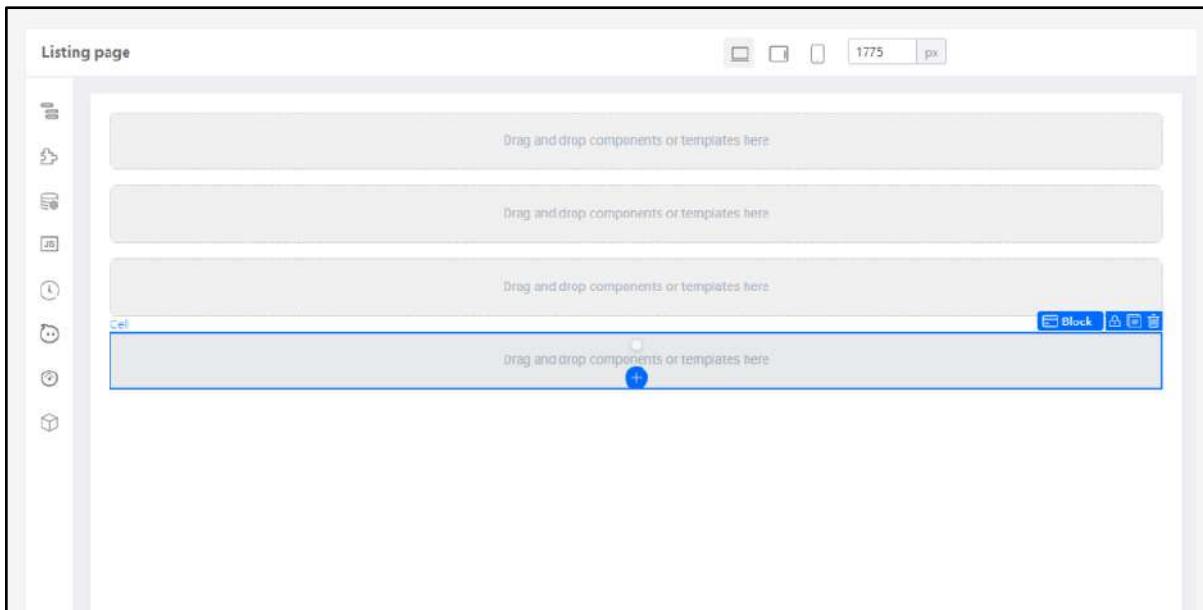
Course Title	Description	Price	Action
Introduction to Programming	Learn the basics of coding.	\$49.99	Enroll Now
Data Science Fundamentals	Explore the world of data analysis.	\$79.99	View Details
The Complete Web Development Bootcamp	Build dynamic web applications.	\$199.99	View Details

Practical 3.1: Create Initial Layout

- Create a new page
- Click “Add Block Below” 3 times to add 3 more blocks

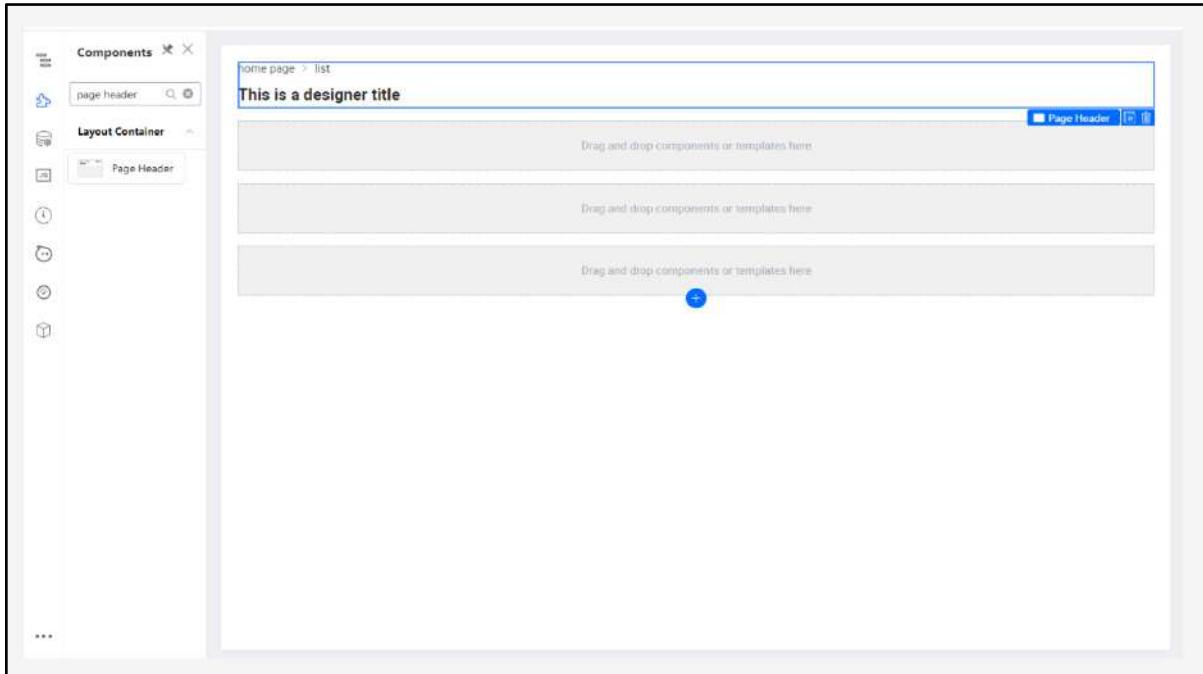


The expected result will be:

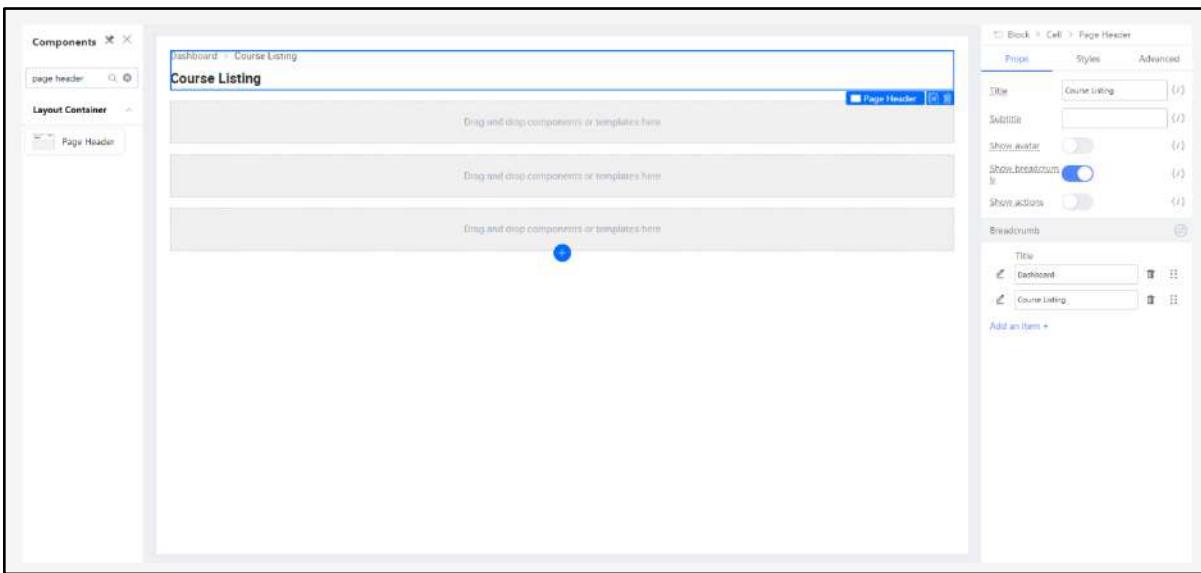


Practical 3.2: Create Heading and Breadcrumb

- Select “Page Header” from “Component Library” and drag into the first block



- Set the following



Practical 3.3: Create Search form

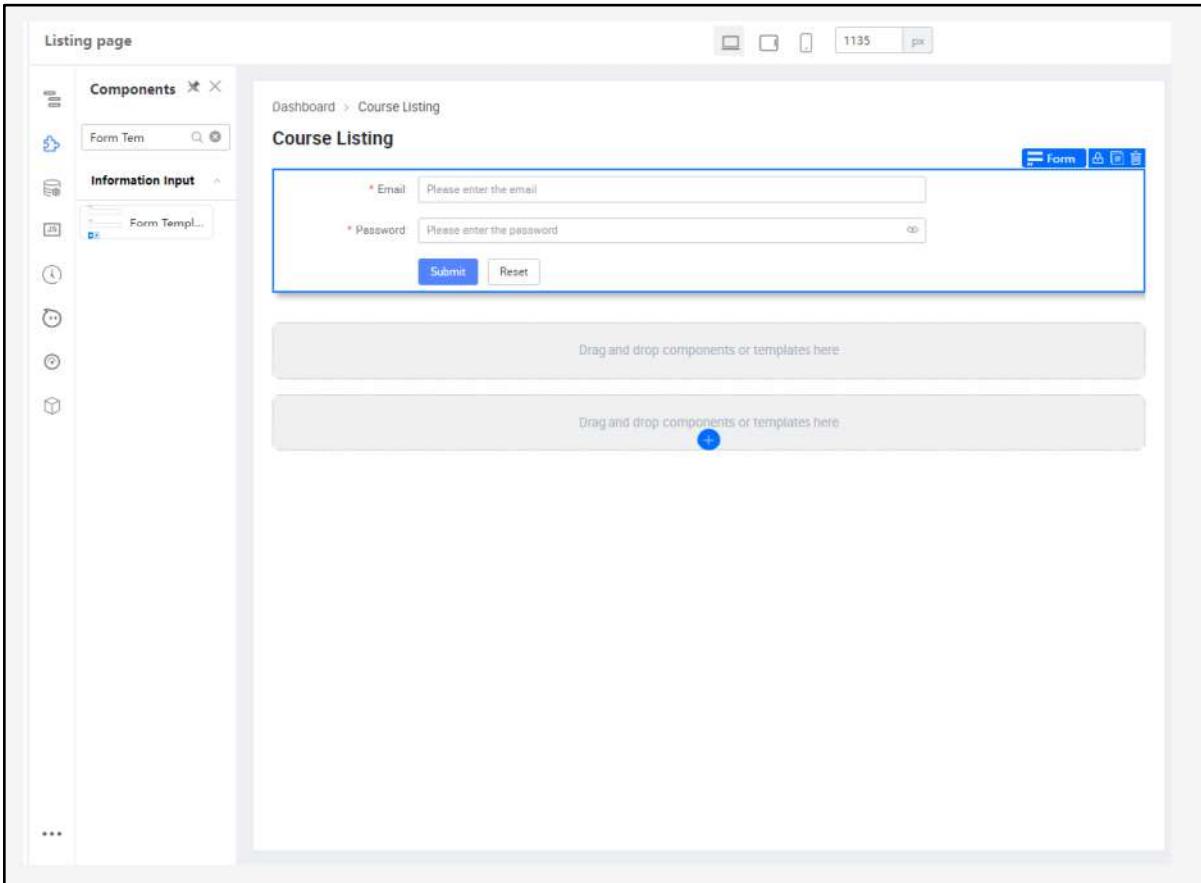
- Select the second Cell and set the following:

The screenshot shows a web editor interface with a styles panel open. The panel has tabs for 'Props', 'Styles' (which is selected), and 'Advanced'. Under 'Styles', there is a code editor containing the following CSS:

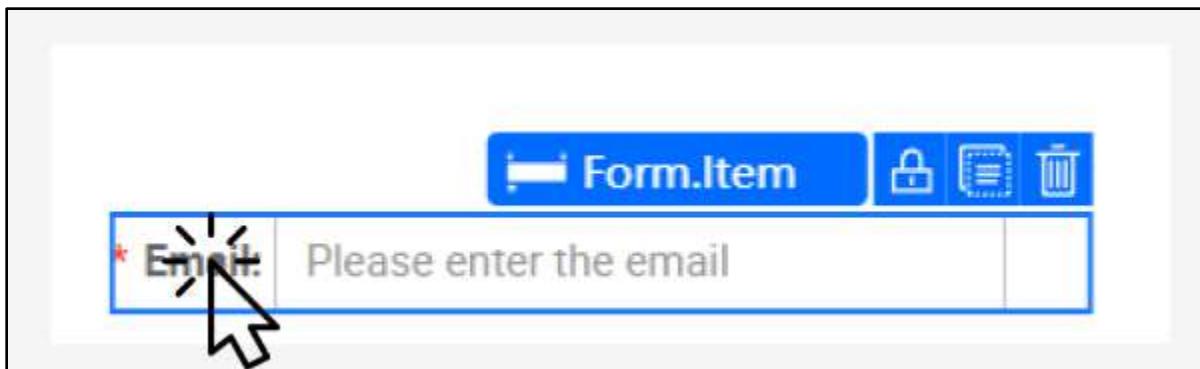
```
background-color: #ffffff;
margin-bottom: 15px;
box-shadow: 5px 5px 5px 0px #c0c0c0;
```

The screenshot shows a 'Course Listing' page in a web editor. On the left, there's a sidebar with icons for file operations. The main area has three 'Drag and drop components or templates here' boxes. The second box from the top is selected, indicated by a blue border. On the right, the 'Style' tab of the 'Cell' properties panel is active, showing the same CSS as in the previous screenshot. The 'Layout' tab is also visible, showing a 2x2 grid layout with margins and widths. The 'Font' tab is partially visible at the bottom.

- Select “Form Template” from “Component Library” and drag into the second “Cell”



- Select the first “Form Item” by clicking on the label



- In the first form item, set the following:

Props

Label: Course Name :

Size: Large

Mandatory validation: false

Layout

Col Span: 8

Label Column

Span: 3

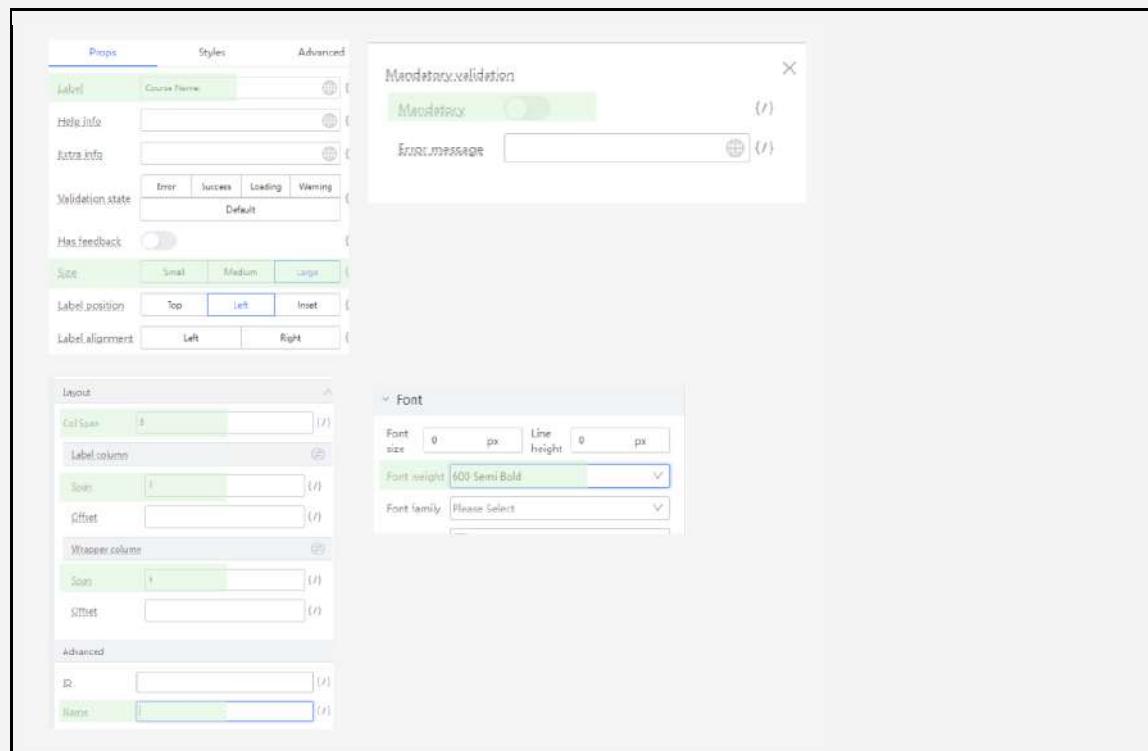
Wrapper Column

Span: 6

Name: (Empty)

Styles

font-weight: 600;



The screenshot shows the FormBuilder interface with a 'Course Listing' form. The left side displays the form structure with two input fields: 'Course Name' and 'Password'. The right side shows the properties panel for the first input field, specifically the 'Props' tab. The 'Placeholder' field is set to 'Please enter the course name'. Other visible properties include 'Label' (Course Name), 'Type' (Input), and various styling and validation options.

- In the Input, set the following:

Props

Placeholder: Please enter the course name

This screenshot shows the 'Course Name' input field selected in the form. The properties panel on the right is open, and the 'Placeholder' field is clearly visible and set to 'Please enter the course name'. The 'Props' tab is active, and other properties like 'Label' and 'Type' are also visible.

- Remove the Password Form.Item

The screenshot shows a 'Course Listing' page with a sidebar containing icons for dashboard, course, user, clock, and others. The main content area has a heading 'Course Listing'. Below it is a 'Form' section with two fields: 'Course Name:' and 'Password'. The 'Password' field is highlighted with a red box and has a tooltip 'Please enter the password'. To the right of the 'Password' field is a toolbar with icons for 'Form.Item', 'Cell', 'Row', 'Column', and 'Remove'. A callout bubble points to the 'Remove' icon. Below the form are two grey boxes with the placeholder text 'Drag and drop components or templates here'.

- Remove the associated Cell

This screenshot shows the same 'Course Listing' page as the previous one, but the 'Password' field has been removed. The 'Form' section now only contains the 'Course Name:' field, which is also highlighted with a red box and has the same tooltip. The 'Remove' icon in the toolbar is still present. The rest of the page, including the sidebar and the two grey drag-and-drop boxes, remains unchanged.

- Click “Duplicate” 3 more times. Please take note to copy the Cell item, not the Form item.

The screenshot shows a "Course Listing" section with four identical input fields for "Course Name". Each field has the placeholder "Please enter the course name". A "Duplicate" button is visible in the top right corner of the input area.

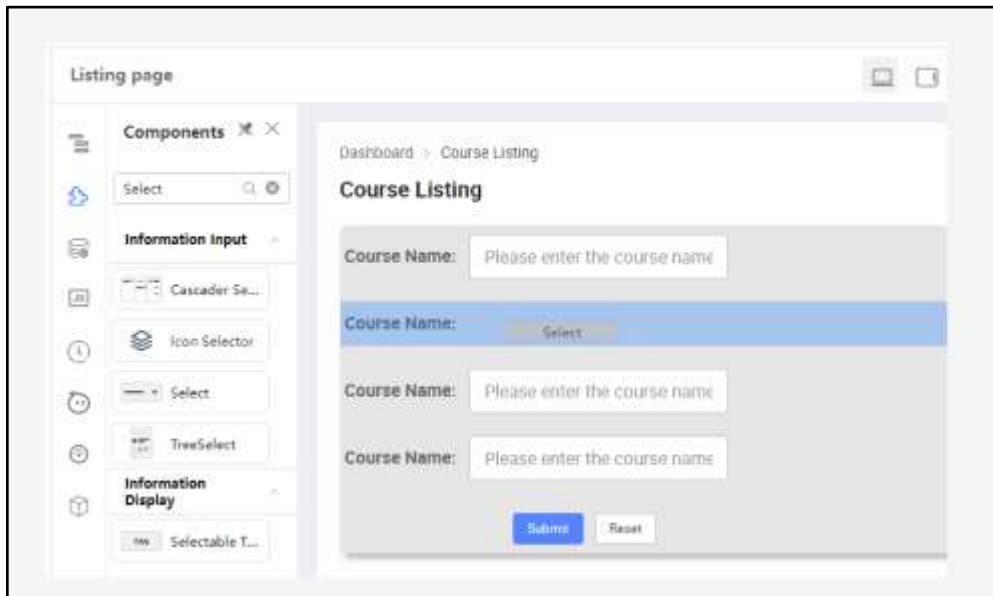
Course Name	Please enter the course name
Course Name	Please enter the course name
Course Name	Please enter the course name
Course Name	Please enter the course name

- Remove the second “Input”

The screenshot shows the same "Course Listing" section, but the second input field has been removed. Only three input fields remain, each with the placeholder "Please enter the course name". A "Remove" button is visible next to the second input field.

Course Name	Please enter the course name
Course Name	Please enter the course name
Course Name	Please enter the course name
Course Name	Please enter the course name

- Replace the Input with a “Select” component



- Set the Form.Item and Select component to the following:

Form.Item

Props

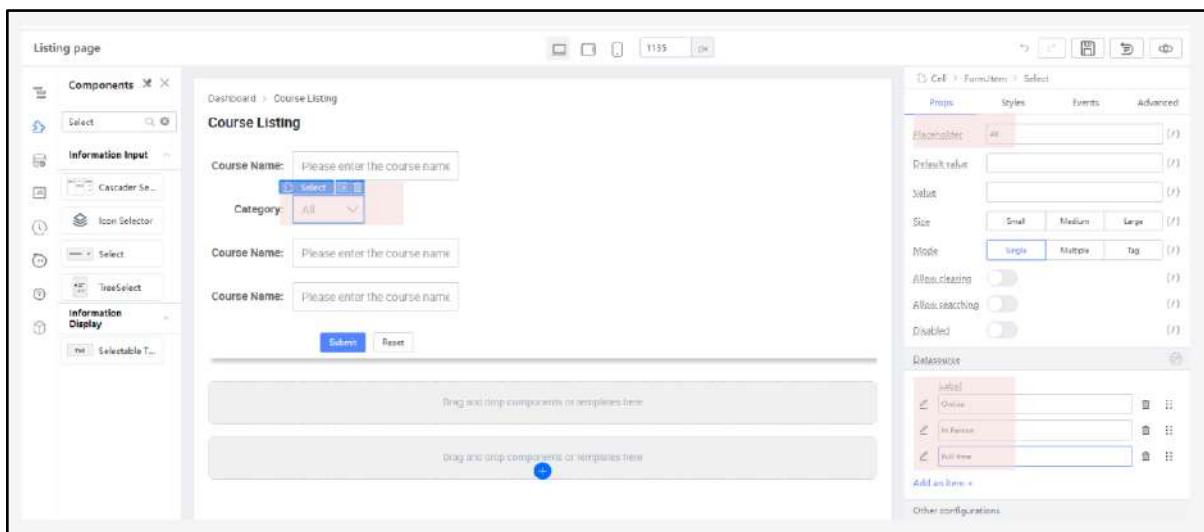
Label: Category:

Select

Props

Placeholder: All

Datasource: Online, In Person, Full time, All



- Delete Input and replace it with a rating component

The screenshot shows the Listing page interface. On the left, there is a sidebar titled "Components" with various icons and a search bar. Below the search bar, under "Information Input", there is a "Rating" component icon. The main area displays a "Course Listing" form. The form has three "Course Name" input fields. The second "Course Name" field is highlighted with a blue background and contains the text "Rating". Below the form are "Submit" and "Reset" buttons.

- Set the Form.Item and Rating component to the following:

The screenshot shows a code editor with the following content:

```

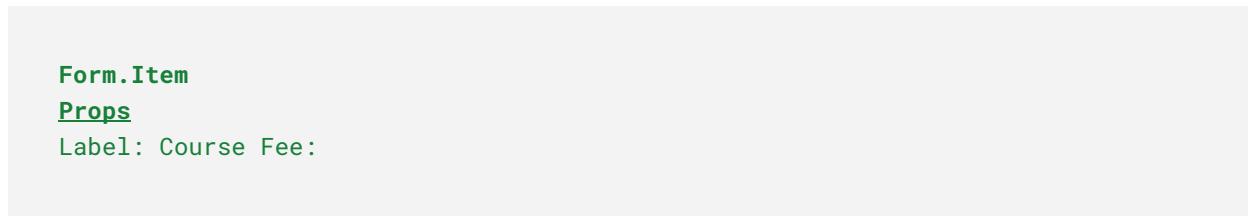
Form.Item
Props
Label: Rating (&up):

Rating
Props
Current value: 3
  
```

The screenshot shows the Listing page interface. A "Rating" component is selected in the form. A context menu is open over the component, showing "Props", "Styles", "Events", and "Advanced" tabs. The "Props" tab is active, showing the following configuration:

Current value	1
Default value	0
Size	Small Medium Large
Max rating	5
Allow half stars	<input checked="" type="checkbox"/>
Show rating	<input checked="" type="checkbox"/>
Customize rating display width	(f)
Bind Function	(f)

- Update Form.Item property and replace the fourth “Input” with a “3 Columns” component



This screenshot shows a form editor interface for a "Course Listing" page. On the left, there's a sidebar titled "Components" with a search bar and a "Layout Container" section containing a "3 Columns" component. The main area has a breadcrumb "Dashboard > Course Listing" and a title "Course Listing". It contains fields for "Course Name" (placeholder: "Please enter the course name") and "Category" (dropdown menu showing "All"). Below these are "Rating (&up):" followed by a 5-star rating icon (3 stars filled, 2 gray). A "Course Fee:" field is present, with three adjacent input boxes labeled "Drag and drop components or templates here" each. At the bottom are "Submit" and "Reset" buttons.

- In the “3 Columns” component, insert a “Number Input” to the first and third column. Delete the “Form Item” Label and set the Size to Large.

ListingPage

Dashboard > Course Listing

Course Listing

Course Name: Please enter the course name

Category: All

Rating (&up): ★★★★☆

Course Fee: Form Item: Number Input34

The 'Number Input' component has a tooltip: "Drag and drop components or templates here".

Props	Styles	Advanced
Label	Form Item: <input type="text"/>	{/}
Help.info	<input type="text"/>	{/}
Extra.info	<input type="text"/>	{/}
Validation.state	Error Success Loading Warning Default	{/}
Has feedback	<input checked="" type="checkbox"/>	{/}
Size	Small <input checked="" type="radio"/> Medium <input type="radio"/> Large	{/}

Props	Styles	Advanced
Label	<input type="text"/>	{/}
Help.info	<input type="text"/>	{/}
Extra.info	<input type="text"/>	{/}
Validation.state	Error Success Loading Warning Default	{/}
Has feedback	<input checked="" type="checkbox"/>	{/}
Size	Small <input type="radio"/> Medium <input checked="" type="radio"/> Large	{/}

- Add a “Text” component to the second column. Similarly, click on its “Form Item” parent component and delete the Label.

The screenshot shows the 'Course Listing' form builder. On the left, the 'Components' sidebar is open, displaying various components like 'text', 'Information Input', 'Basic Elements', and 'Text'. A 'Text' component is currently selected and being drag-and-dropped into the second column of a grid. A tooltip indicates: 'Drag and drop components or templates here'. The main area shows fields for 'Course Name', 'Category', 'Rating (&up)', and 'Course Fee'. Below the grid are 'Submit' and 'Reset' buttons.

This screenshot shows the properties panel for the 'Text' component. The path 'Col19 > Form Item > Text37' is visible at the top. The 'Props' tab is selected. The 'Text' field contains the placeholder 'This is a designer text'. Other props include 'Font.size' set to 'Default', and various styling options like 'Highlight', 'Code.inline', 'Strikethrough', 'Underline', and 'Strong' all turned off.

This screenshot shows the 'Form Item' properties panel. The 'Label' field is highlighted with a red box. The 'Props' tab is selected, showing the 'Label' field with a placeholder 'Form Item'.

This screenshot shows the properties panel for the 'Form Item' component. The path 'Row17 > Col19 > Form Item' is visible at the top. The 'Props' tab is selected. The 'Label' field is empty. Other props include 'Help.info', 'Extra.info', 'Validation.state' (set to 'Default'), 'Has.feedback' (unchecked), 'Size' (set to 'Medium'), and 'Label.position' (set to 'Inset').

- Set the Text component to the following:

```
Text  
Props  
Text: to
```

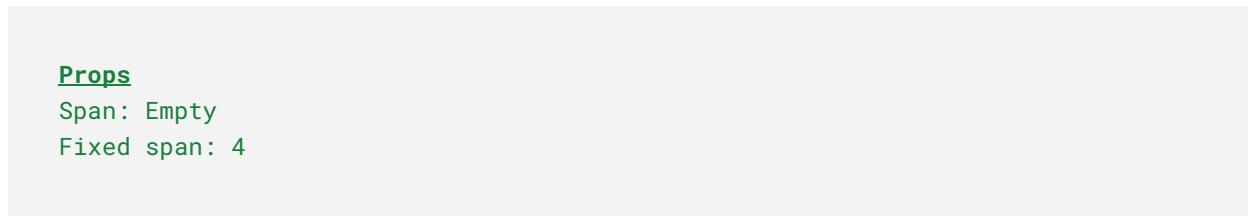
```
Styles  
font-size: 16px;
```



The screenshot shows a user interface for a 'Course Listing' search. At the top left, there is a breadcrumb navigation: 'Dashboard > Course Listing'. Below it is a section titled 'Course Listing'. The form contains several input fields and controls:

- A text input field labeled 'Course Name:' with the placeholder 'Please enter the course name'.
- A dropdown menu labeled 'Category:' with the option 'All' selected.
- A rating scale labeled 'Rating (&up):' with five stars, where the first three are filled blue and the last two are white.
- A text input field labeled 'Course Fee:' with a small icon and the identifier 'Text34' above it.
- Two empty rectangular input fields side-by-side.
- At the bottom right are two buttons: a blue 'Submit' button and a white 'Reset' button.

- Set the Column (With Number Input) to the following



The screenshot shows a 'Course Listing' form. On the left, there is a form with fields for 'Course Name' (text input), 'Category' (dropdown menu), 'Rating (&up:) (star rating)', 'Course Fee' (two input fields with a 'col29' label between them), and 'Submit/Reset' buttons. On the right, a detailed configuration panel for the 'col29' column is shown, mirroring the 'Props' settings from the previous screenshot. The 'Span' field is empty, 'Fixed span' is set to 4, and 'Align' is set to 'Please Select'.

- Set the Column (With text) to the following:

Props

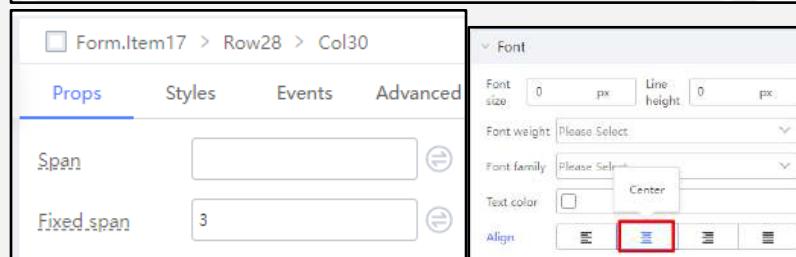
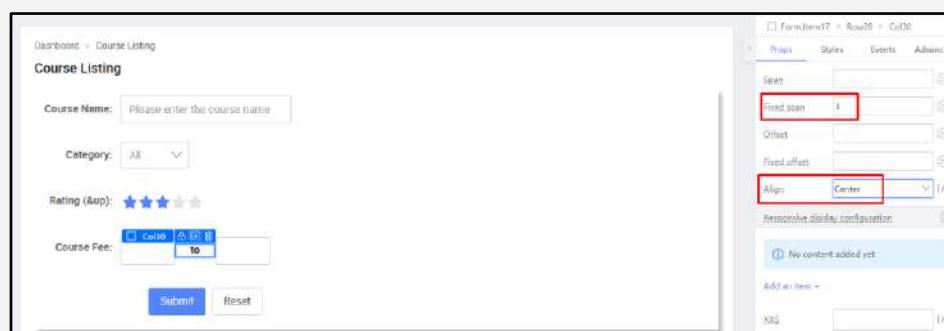
Span: Empty

Fixed span: 3

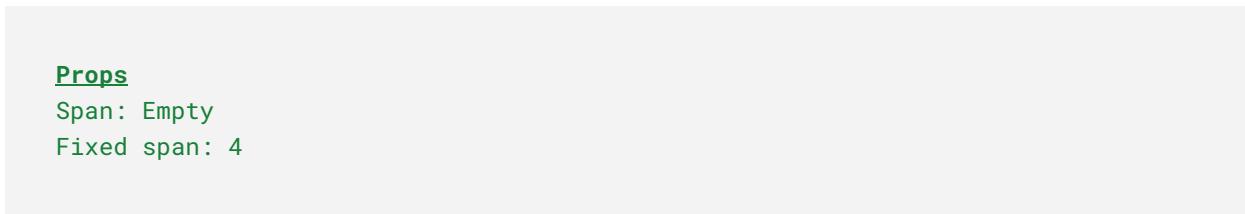
Align: Center

Styles

`text-align: center;`



- Set the Column (Last number input) to the following:



Dashboard > Course Listing
Course Listing

Course Name: Please enter the course name

Category: All

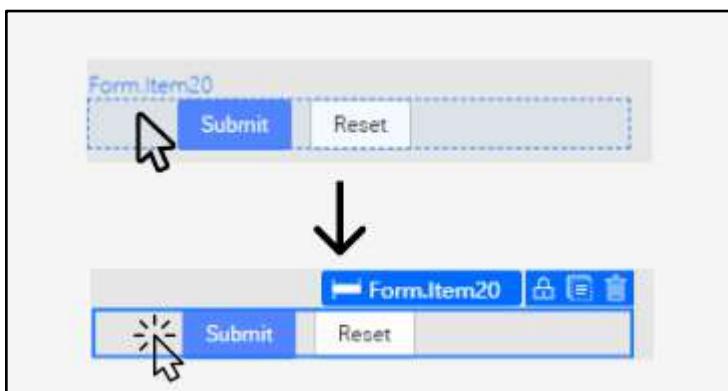
Rating (&up): ★★☆☆☆

Course Fee: [] to []

Submit Reset

Props Tab Properties:
Span: Empty
Fixed span: 4
Offsets: 0
Fixed offset: 0
Align: None
Responsive display configuration: No content added yet
Add an item +
XXS

- Click on last form.item (Containing area)



- In the Form.Item (With Submit Button), set the following:

Props

Size: Large

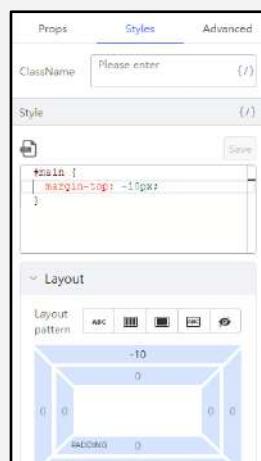
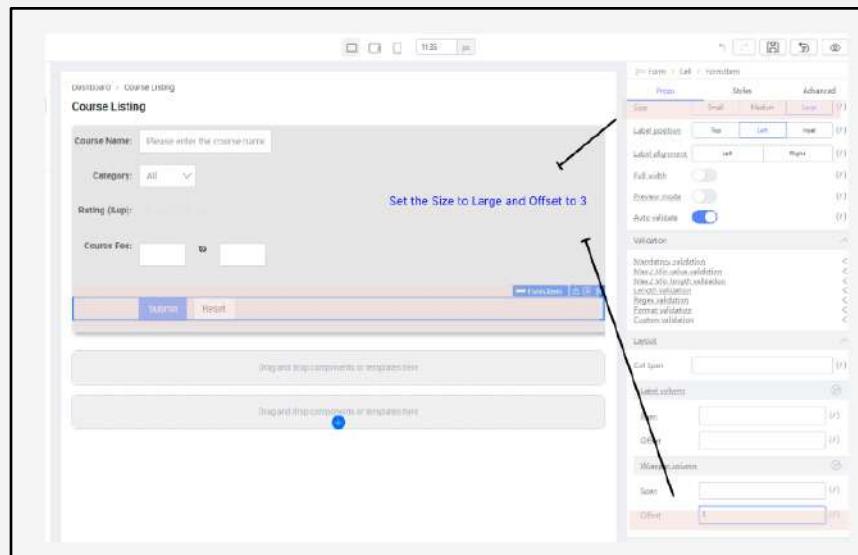
Wrapper Column

Span: 0

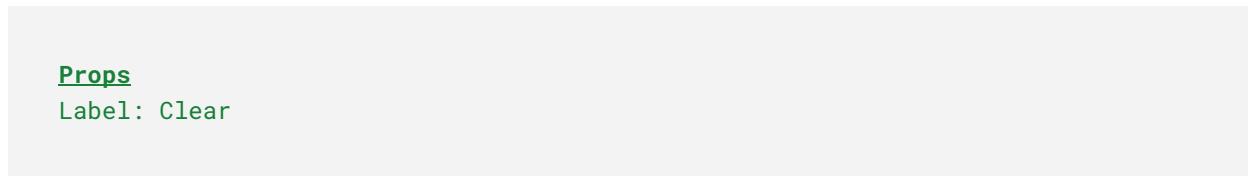
Offset: 3

Styles

```
margin-top : -10px;
```



- Set the “Reset” button label with the following



The screenshot displays a "Course Listing" form. At the top left, it says "Dashboard > Course Listing". The form includes fields for "Course Name" (with placeholder "Please enter the course name"), "Category" (set to "All"), and "Rating (&up;)". Below these are two input fields for "Course Fee" with operators "to" and "From". At the bottom right of the form is a blue "Submit" button and a white "Clear" button. To the right of the form is a "Props" panel with various configuration options, including "Label" set to "Clear".

Practical 3.4: Create List header

- Drag a “Button” to the third “Block”

The screenshot shows a "Listing page" interface with a sidebar titled "Components". The sidebar contains categories like "General" and items such as "Button", "Form", "Image", "Text", "Table", "List", and "Panel". A "Button Group" item is currently selected, indicated by a blue border.

The main area displays a "Course Listing" form with the following fields:

- Course Name:**
- Category:**
- Rating (&up):** ★★★☆☆
- Course Fee:** to
- Buttons:**

Below the form, there is a toolbar with icons for "Button", "Form", "Image", and "Text", and a "cancel" button. At the bottom, a placeholder text "Drag and drop components or templates here" is followed by a blue plus sign (+).

- Set the following

Props
Content: Create New Course

Styles
`height: 50px;
font-size: 20px;`

Dashboard > Course Listing

Course Listing

Course Name:

Category:

Rating (&up): 

Course Fee: to

Drag and drop components or templates here

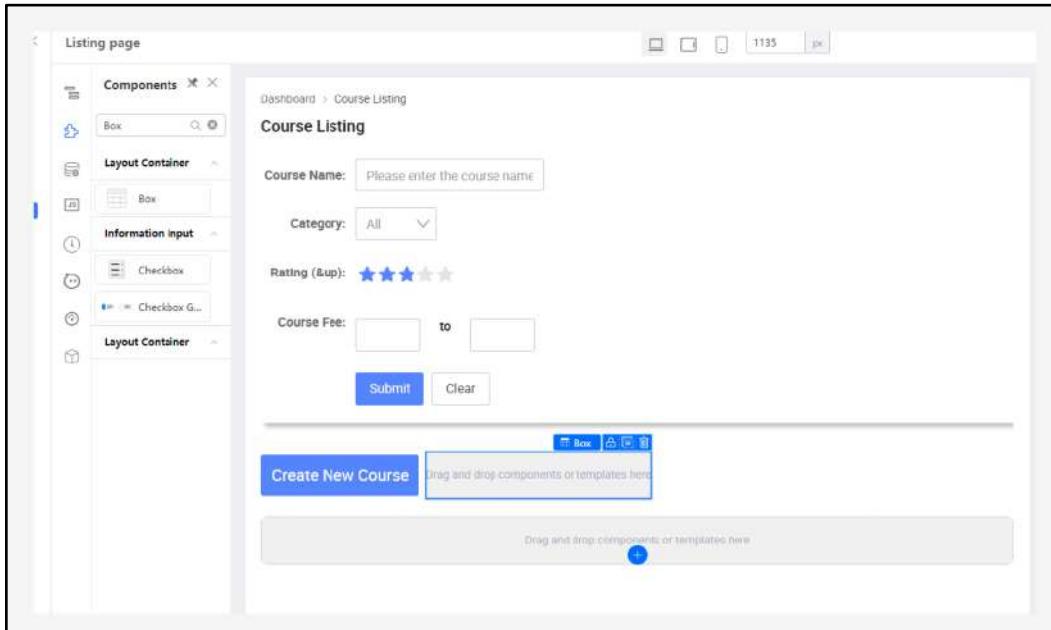
Props
ClassName: Please select

Style
`#main {
height: 50px;
font-size: 20px;
}`

Layout
Layout pattern: ABC

Width: 126 px
Height: 50 px

- Drag a “Box” beside the Button



- Set the Box with the following:

Props

Direction: Row

Justify: flex-start

Styles

```
background-color: #5584ff;
padding-top: 10px;
padding-left: 10px;
padding-bottom: 10px;
padding-right: 10px;
width: 100%;
```

The screenshot shows a "Course Listing" form on a dashboard. The form includes fields for Course Name, Category, Rating, Course Fee, and buttons for Submit and Clear. Below the form is a "Create New Course" button and a placeholder for dragging components. A style editor overlay is open, showing the CSS for a "Box" component and a layout editor with margin settings.

```
background-color: #e6f2ff;
padding-top: 10px;
padding-left: 10px;
padding-bottom: 10px;
padding-right: 10px;
width: 100%;
```

Layout pattern: ABC

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

MARGIN: 0 0 0 0

Width: 278 px

- Drag a “Text” component into the “Box”

The screenshot shows the component library on the left with the "Text" component selected. The main area displays the "Course Listing" form. A blue selection bar highlights the "Text" component in the library, indicating it has been dragged into the "Box" area.

- Set the Text with following:

```
Props  
Text: Course Listing  
  
Styles  
font-size: 24px;  
color: #ffffff;
```

The screenshot shows a user interface for building a web page. On the left, there is a preview area displaying a form titled "Course Listing". The form includes fields for "Course Name" (with placeholder "Please enter the course name"), "Category" (set to "All"), and "Rating (&up;)". It also has a "Course Fee" range selector and "Submit" and "Clear" buttons. Below the form is a "Create New Course" button. At the bottom, there is a "Course Listing" section with a "Drag and drop components or templates here" placeholder and a plus sign icon. On the right, there is a sidebar titled "Paragraph > Box > Text" with tabs for "Props", "Styles", and "Events". The "Styles" tab is active, showing a CSS snippet:

```
*main {  
    font-size: 24px;  
    color: #ffffff;
```

Below the styles are sections for "Layout" (with a visual grid editor) and "Font" (with width and height settings).

Practical 3.5: Create Table

- Add “Table” to the last “Block”

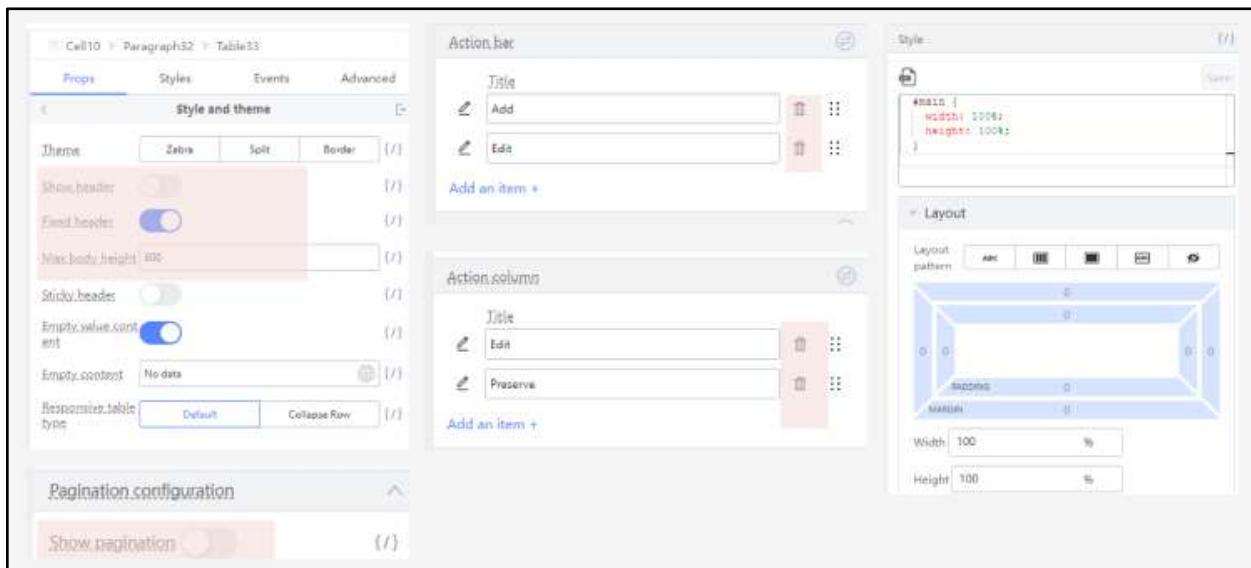
The screenshot shows a web-based application interface. On the left, there is a sidebar titled "Components" containing various UI element icons. The main area has a header "Listing page". Below the header, there is a search form with fields for "Course Name" (placeholder: Please enter the course name), "Category" (dropdown: All), "Rating (Avg)": ★★★★☆, "Course Fee": (input boxes for min and max values), and buttons for "Submit" and "Clear". Below the search form is a blue button labeled "Create New Course". Underneath it is a section titled "Course Listing" with a table. The table has columns: "Name", "Age", "Responsibility", and "Operation". It contains three rows of data:

Name	Age	Responsibility	Operation
Wang Xian	15000	developer	Edit Remove
Wang Zheng	25000	product	Edit Remove
Wang De	35000	design	Edit Remove

- In “Table” properties, set the following

```
Props
Pagination configuration
  'Show pagination': false
Style and theme
  'Show header': false
  'Fixed header': true
  'Max body height': 800
Action bar
  delete "Add" and "Edit"
Action column
  delete "Edit" and "Preserve"

Styles
height: 100%;
width: 100%;
```



The expected result will be:

The screenshot shows a user interface for a "Course Listing" page. On the left, there's a sidebar titled "Components" with various UI element icons like table, input, and dropdown. The main area has a header "Course Listing". Below it, there are input fields for "Course Name" (placeholder: "Please enter the course name"), "Category" (dropdown menu showing "All"), and "Rating (8up)" (stars). There's also a "Course Fee" field with a dropdown menu containing "10". At the bottom of this section are "Submit" and "Clear" buttons. A blue button labeled "Create New Course" is positioned below the input fields. To the right, there's a table titled "Course Listing" with three rows of data. The columns are "Name", "Fee", and "Category". The data is as follows:

Name	Fee	Category
Wang Xue	15000	develop
Wang Zhong	25000	product
Wang Da	35000	design

On the far right, there are several panels: "Style" (containing CSS code for a main element), "Layout" (with a grid pattern and margin settings), and "Font" (with font size, weight, and family options).

- In “Table” properties, under Data Column, set the following properties to all data columns

Name	Age	Responsibility
Width: 700 Data type: None Render: enabled, index, record	Width: 200 Data type: None Render: enabled, index, record	Width: 200 Data type: None Render: enabled, index, record

Example with the “Age” Data Column:

The screenshot shows the Application Manager interface with the following details:

- Left Panel (Item 1 Properties):**
 - Title: Age
 - Align: Center
 - Data key: age
 - Width: 200
 - Data type:** None (highlighted with a red arrow)
 - Lock: Left, Right, None (highlighted with a red arrow)
 - Allow sorting: Off
 - Cell: Bind Function
 - Enable: On
 - Render: Params: enabled, index, record (highlighted with a red arrow)
- Right Panel (Table Properties):**
 - Props tab is selected.
 - Tree table section: No content added yet.
 - Action bar section: No content added yet.
 - Add an item +
 - Link bar section: No content added yet.
 - Add an item +
 - Top actions section: No content added yet.
 - Data column section:
 - Title: Name
 - Age (highlighted with a red box)
 - Responsibility

Props

Direction: Row

Justify: flex-start

Align: center

Spacing: 45

The screenshot shows a UI builder interface with a 'Table' component selected. The 'Props' tab is active, displaying the following properties:

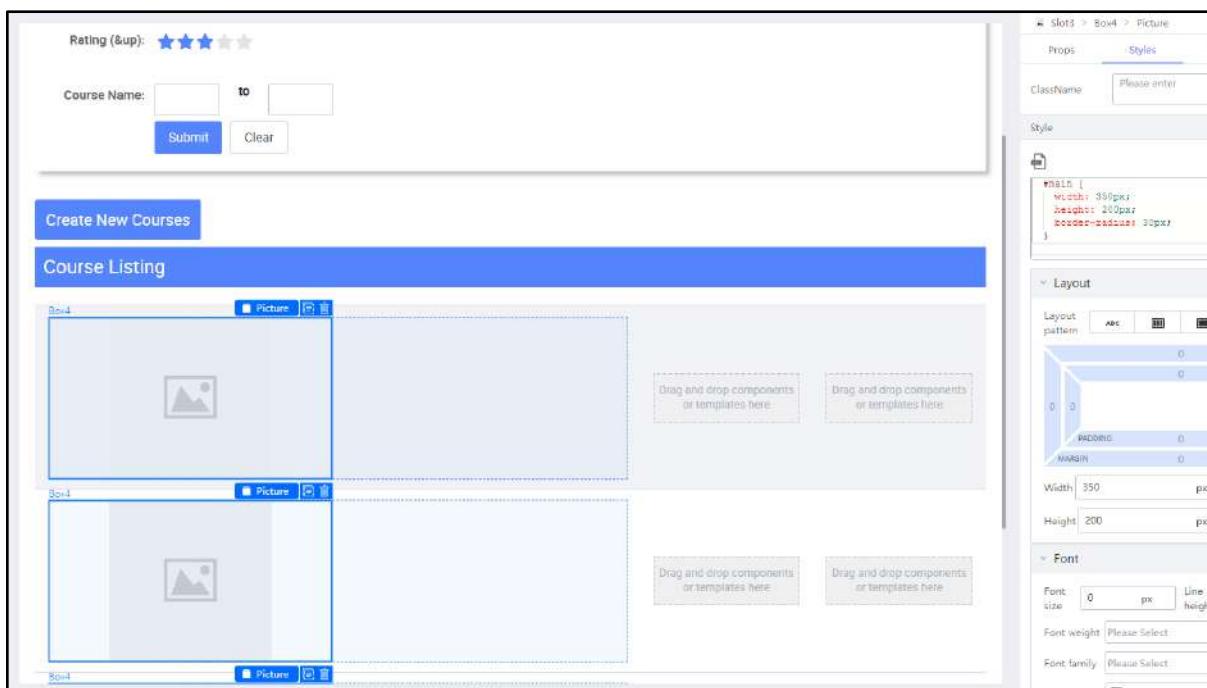
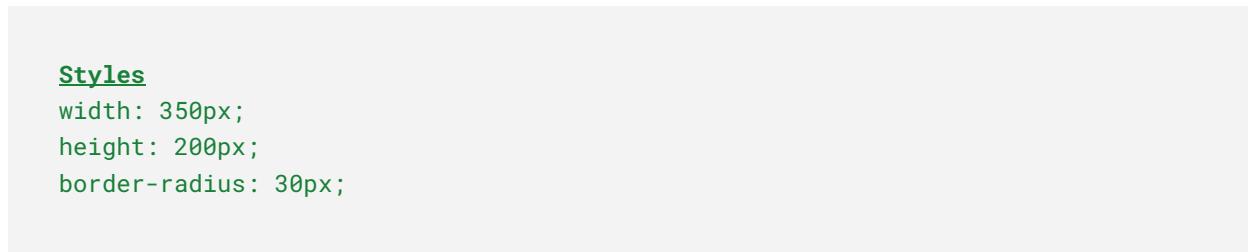
- Direction: Row
- Justify: flex-start
- Align: center
- Spacing: 45

The 'Course Listing' section contains three rows of course cards, each with a 'Picture' icon and a 'Drag and drop components or templates here' placeholder.

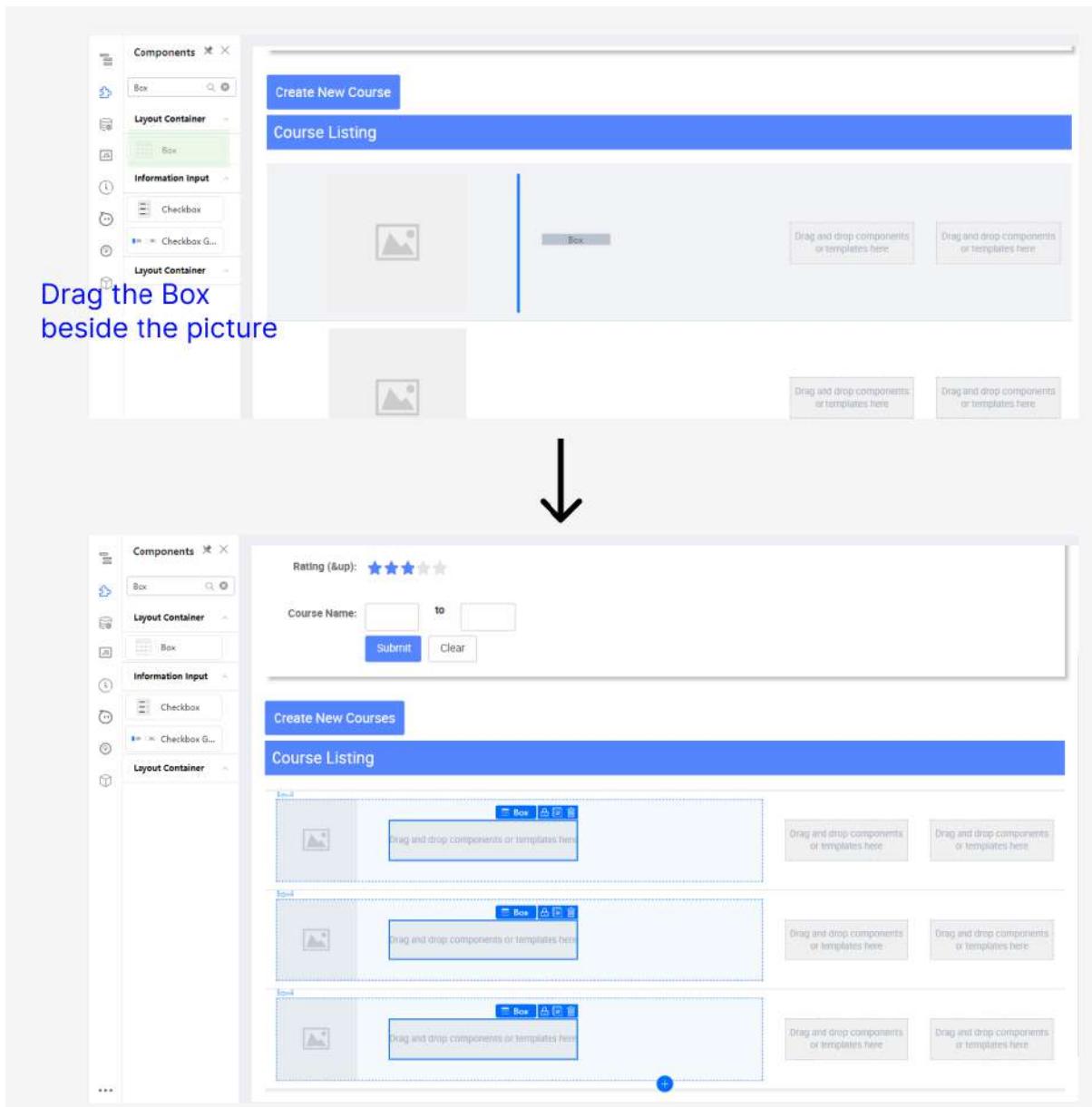
- Add a “Picture” to the table first column ‘Box’

The screenshot shows the same UI builder interface after adding 'Picture' components to the first column of the table rows. The 'Components' sidebar shows a 'Picture' component selected. The 'Course Listing' section now displays three rows, each with a 'Picture' icon in the first column and a 'Drag and drop components or templates here' placeholder in the subsequent columns.

- Set the following styles



- Add a “Box” to the first column beside Picture



- Set the following styles

The screenshot shows a web-based visual editor interface for creating and styling user interface components. On the left, there is a preview area displaying a form for rating courses and a course listing section. The course listing section contains three items, each with a thumbnail, a title, and a 'Drag and drop components or templates here' placeholder. On the right, there is a vertical toolbar with several tabs: 'Props', 'Styles', and 'Advanced'. The 'Props' tab is currently selected. Under 'Props', there are four main sections: 'Direction' (with 'Row' selected), 'Justify' (with 'flex-start' selected), 'Align' (with 'flex-start' selected), and 'Spacing' (set to '25'). The 'Styles' and 'Advanced' tabs are also visible at the top of the sidebar.

- Add a “Text” to the “Box” from the previous step

Drag the Text inside the Box

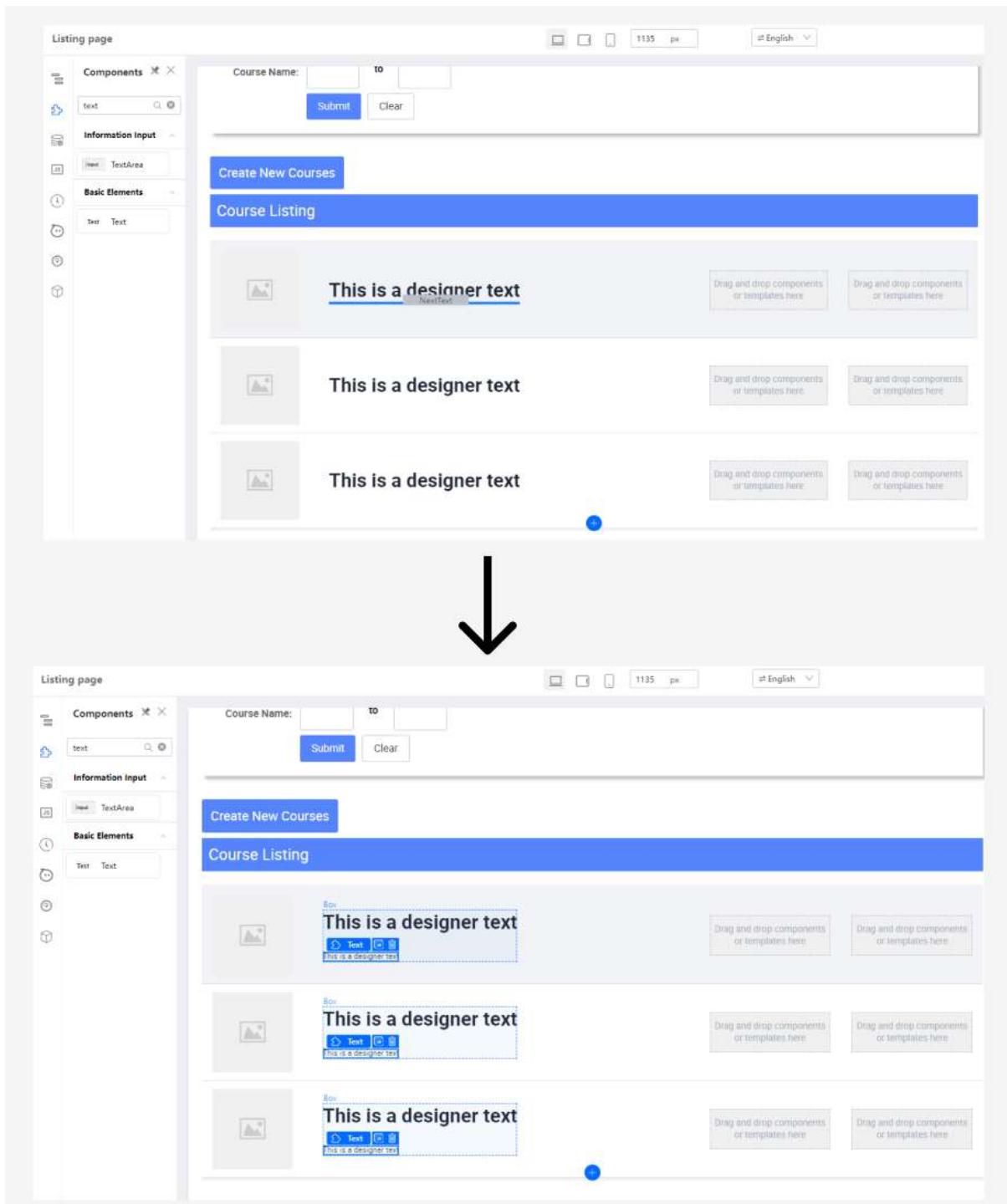
↓

- Set the following to the “Text” component

The screenshot shows a UI builder interface with a sidebar on the left and a main canvas on the right. In the sidebar, under the 'Props' tab, the 'Strong' prop is set to 'true'. Under the 'Styles' tab, the 'font-size' style is set to '30px;'. The main canvas displays a 'Course Listing' section with three identical text components, each containing the text 'This is a designer text'. Each text component has a 'Text' icon with a dropdown arrow next to it. To the right of the canvas is a vertical panel titled 'Box4 > Box > Text' which contains tabs for 'Props' and 'Styles'. The 'Styles' tab is active, showing the CSS rule '#main { font-size: 30px; }'. Below this are sections for 'Layout' (with a preview of a blue box) and 'Font' (with a font size of 30px selected).

This screenshot shows a more complex view of the UI builder. On the left, there's a form with fields for 'Course Name' and buttons for 'Submit' and 'Clear'. Below this is a 'Create New Courses' button. The main area is titled 'Course Listing' and contains three rows of course cards. Each card features a thumbnail image, the text 'This is a designer text', and two 'Drag and drop components or templates here' slots. To the right of the cards is a vertical panel for styling the entire 'Box4 > Box > Text' component. It includes tabs for 'Props' and 'Styles', with 'Styles' being the active tab. The 'Style' panel shows the CSS rule '#main { font-size: 30px; }'. It also includes sections for 'Layout' (with a preview of a blue box), 'Font' (with a font size of 30px selected), and 'Margin' (set to 0). A 'Font weight' dropdown is also present.

- Add a second “Text” to the “Box



The image shows two screenshots of a web application interface, likely a CMS or design tool, illustrating the process of adding multiple text components.

Top Screenshot (Initial State):

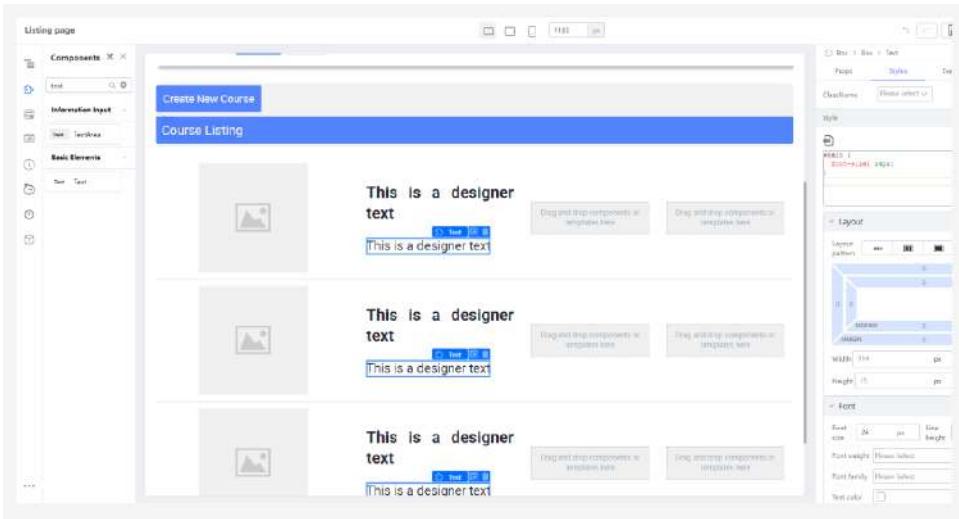
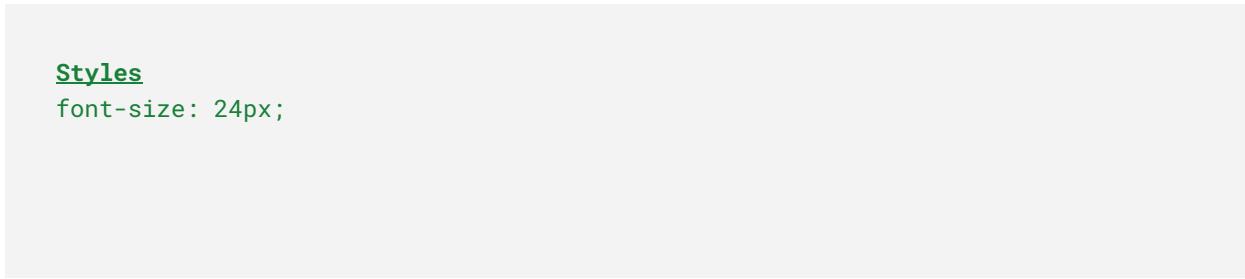
- The interface is titled "Listing page".
- A sidebar on the left contains a "Components" panel with a "text" item selected.
- The main content area features a search bar with fields for "Course Name" and "TO", and buttons for "Submit" and "Clear".
- A blue button labeled "Create New Courses" is visible.
- The main content area has three rows of text components, each containing the placeholder text "This is a designer text".
- Each row includes a small icon and a "NextText" link.
- On the right, there are two "Drag and drop components or templates here" boxes.

Bottom Screenshot (Final State):

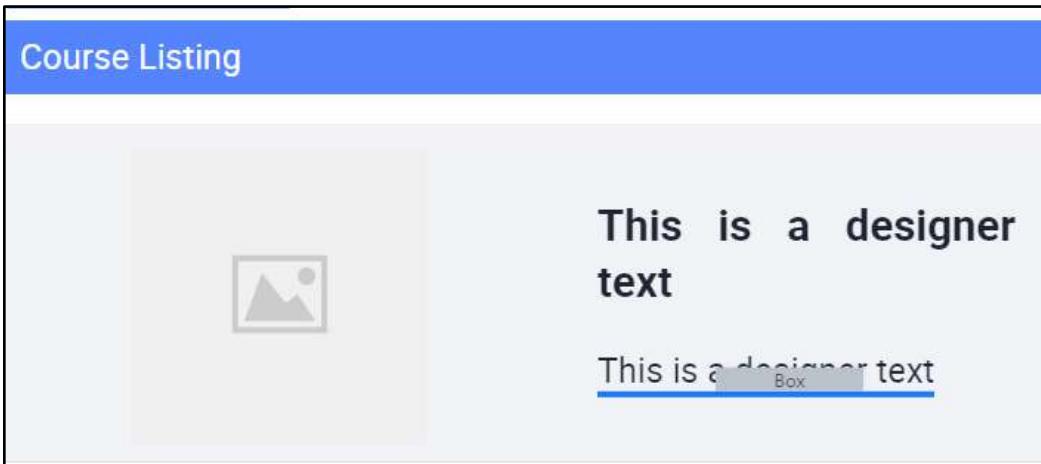
- The interface remains titled "Listing page".
- The sidebar and search bar are identical to the top screenshot.
- The main content area now shows three rows of text components, each enclosed in a dashed border indicating they are selected.
- Each selected row contains the text "This is a designer text" and includes a "Box" icon above it.
- Each selected row also includes a "Text" icon and a "NextText" link.
- On the right, there are two "Drag and drop components or templates here" boxes.

A large black arrow points downwards from the top screenshot to the bottom screenshot, indicating the progression of the task.

- Set the following for the second “Text”



- Add another “Box” below the “Text” component



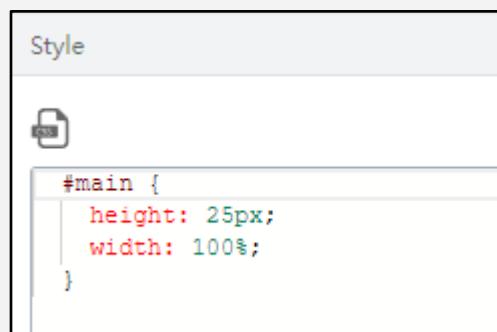
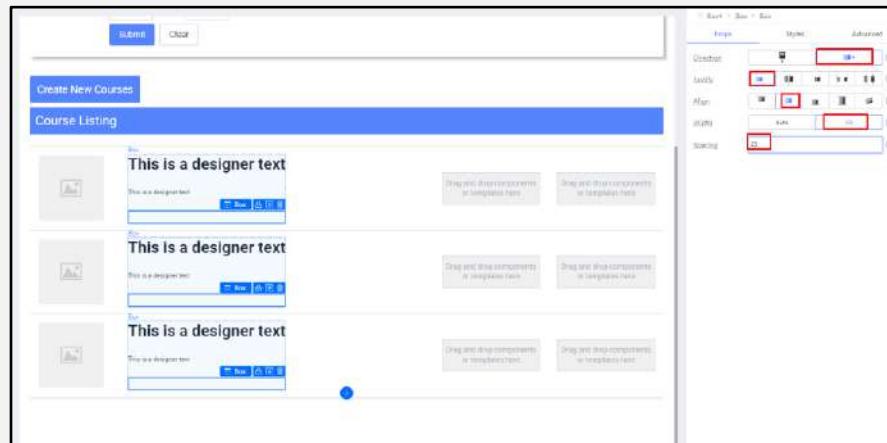
- Set the following

Props

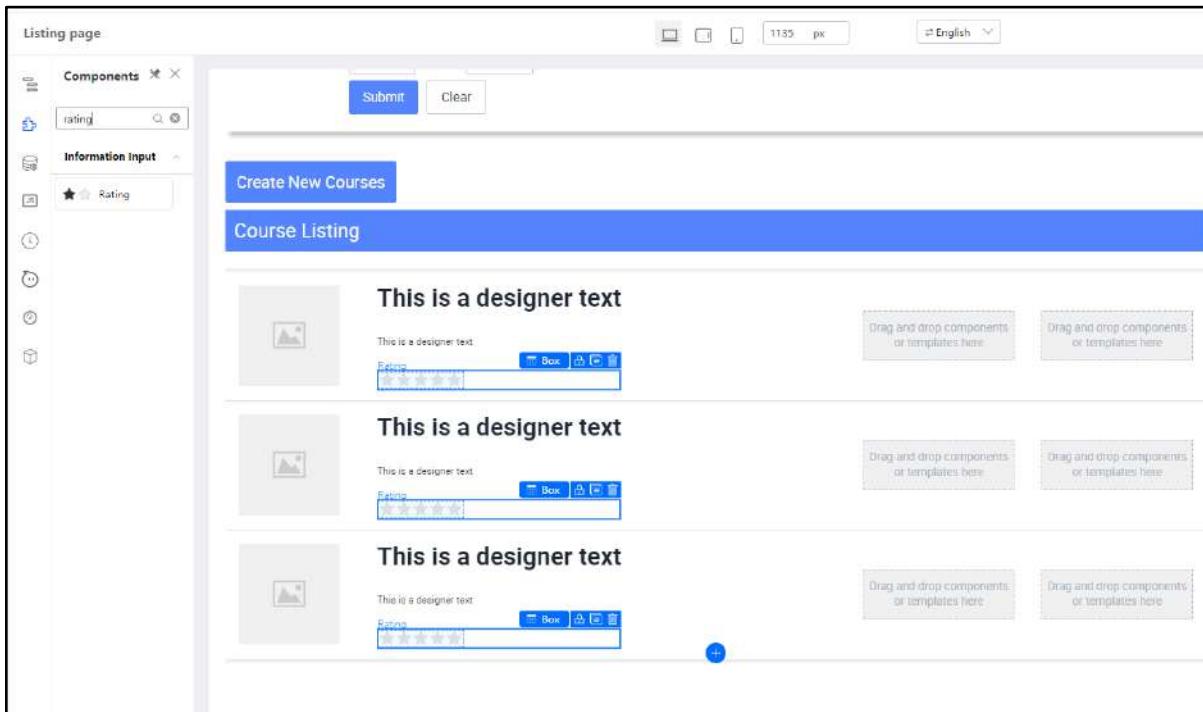
Direction: Row
Justify: flex-start
Align: center
Width: Fill
Spacing: 25

Styles

height: 25px;
width: 100%;

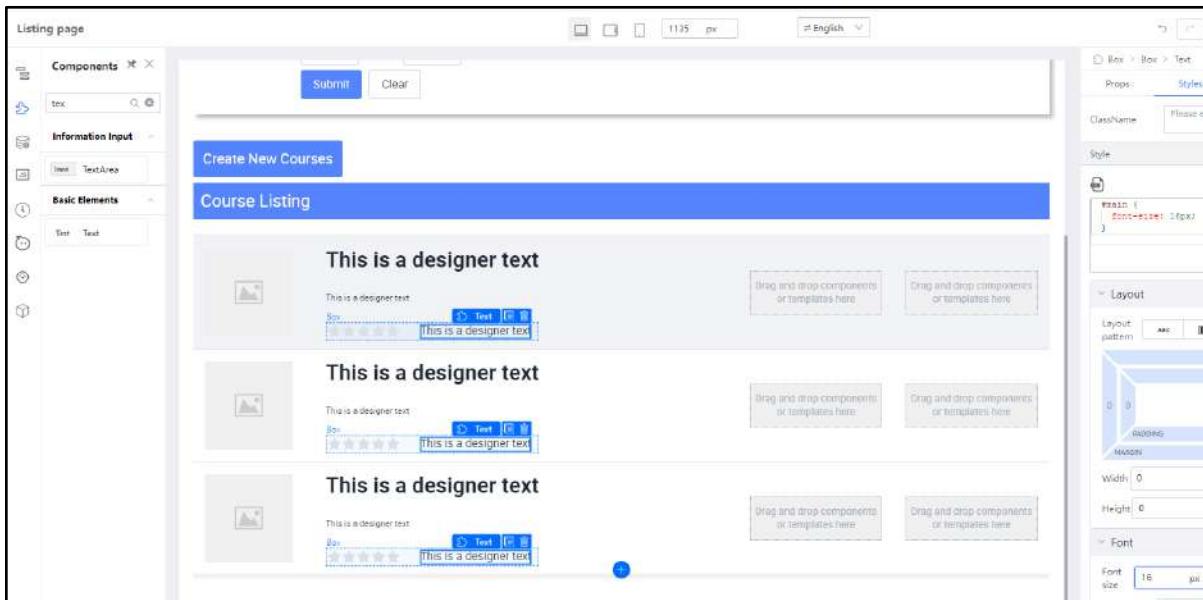


- Add a **Rating** component to the “Box” inserted in previous step



- Add a “Text” beside Rating and set the following

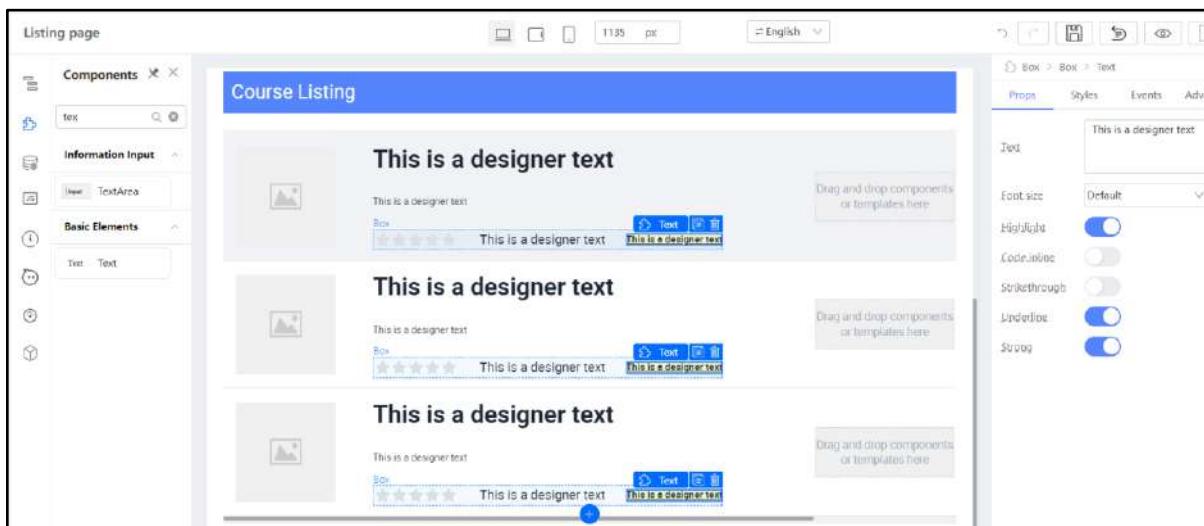
Styles
font-size: 16px;



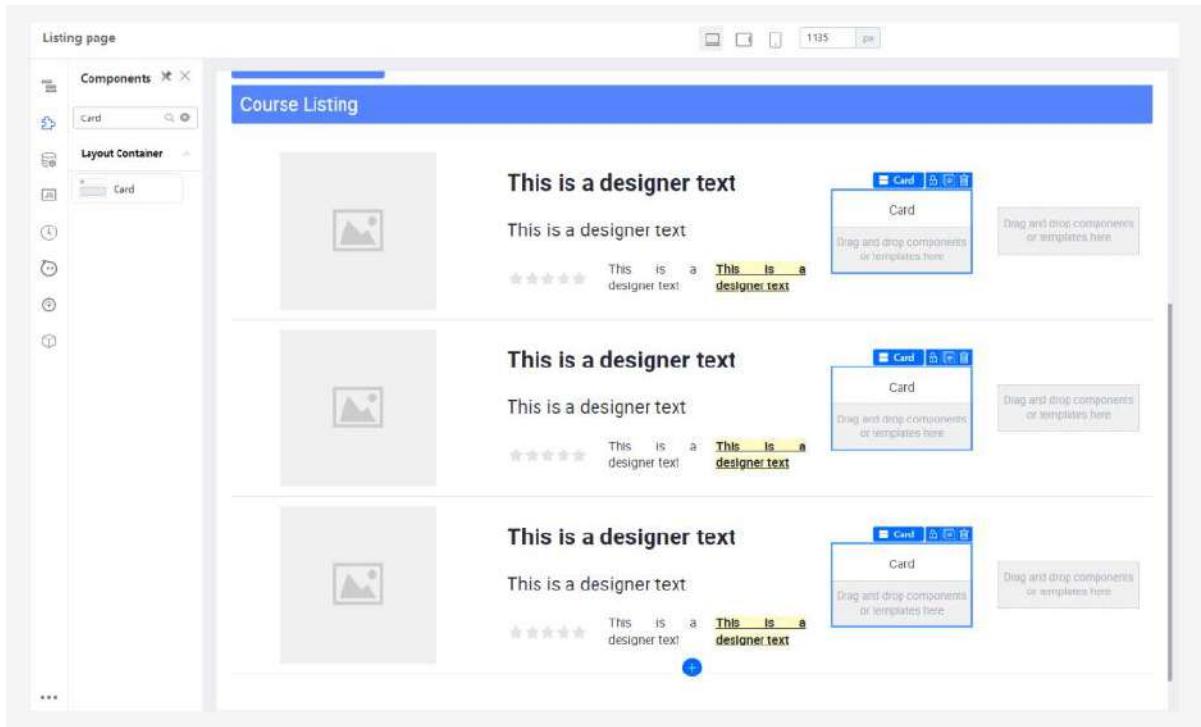
- Add another “Text” beside Rating and set the following

```
Props
Highlight: true
Underline: true
Strong: true

Styles
font-size: 16px;
```



- Add a “Card” to the second column



- Set the Card with the following

Props

Has border: true
Title: (Empty)

Styles

```
min-width: 50%;  
height: 100%;  
border-color: #479eea;  
border-width: 2px;  
padding-left: 25px;  
padding-right: 25px;  
text-align: center;  
display: flex;  
flex-direction: column;  
justify-content: center;  
align-items: center;  
border-radius: 13px;  
padding-bottom: 15px;  
padding-top: 15px;
```

The expected result will be:

The screenshot displays a 'Course Listing' interface within a UI design tool. It features three identical card components arranged vertically. Each card includes a placeholder image, a main title ('This is a designer text'), a subtitle ('This is a designer text'), a star rating, and a detailed description. A sidebar on the right provides configuration options for the 'Card' component.

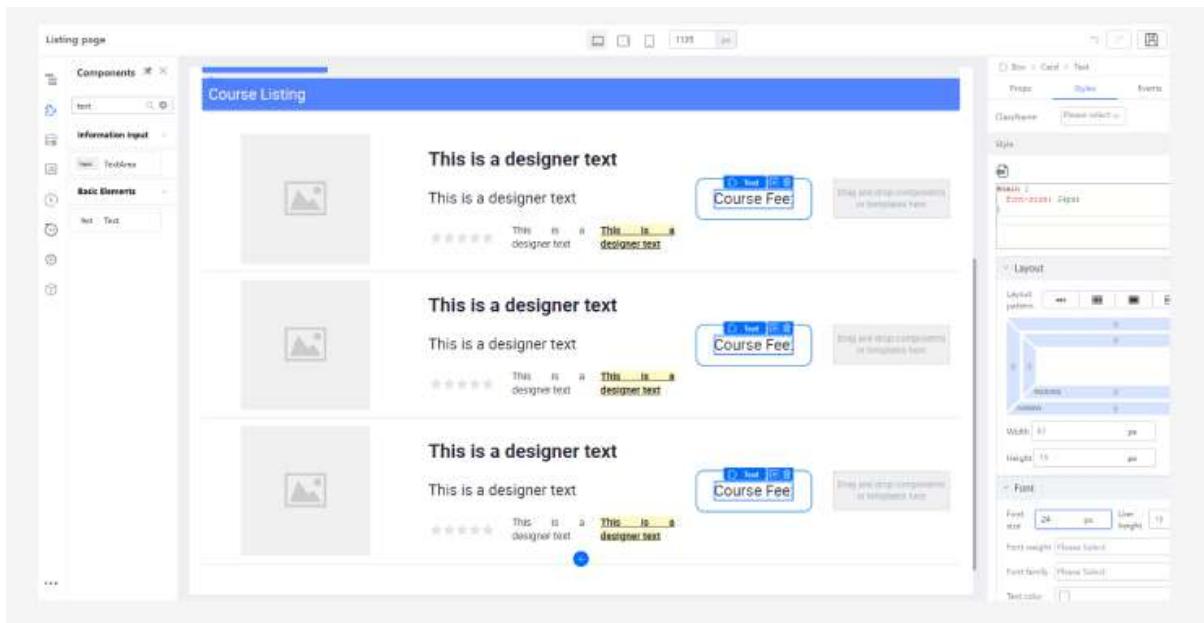
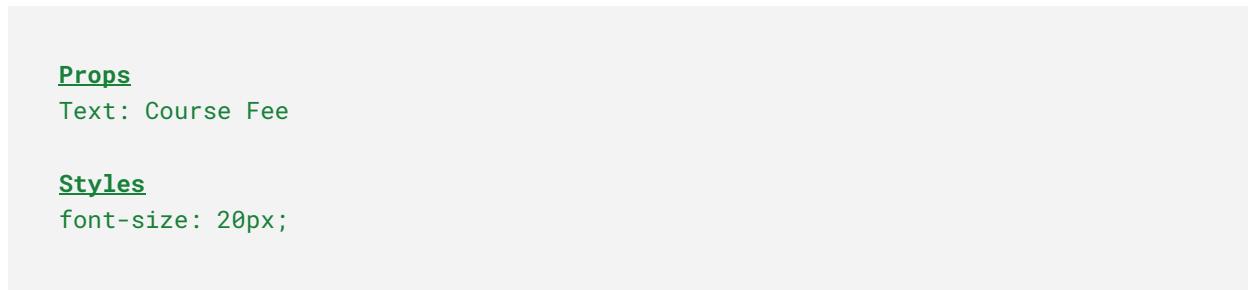
Component Settings (Sidebar):

- Mode: Card
- Free mode: On
- Show symbol: Off
- Show separator: Off
- Has border: On
- Title: (empty)
- Subtitle: (empty)
- Custom content: (empty)
- Context: bright

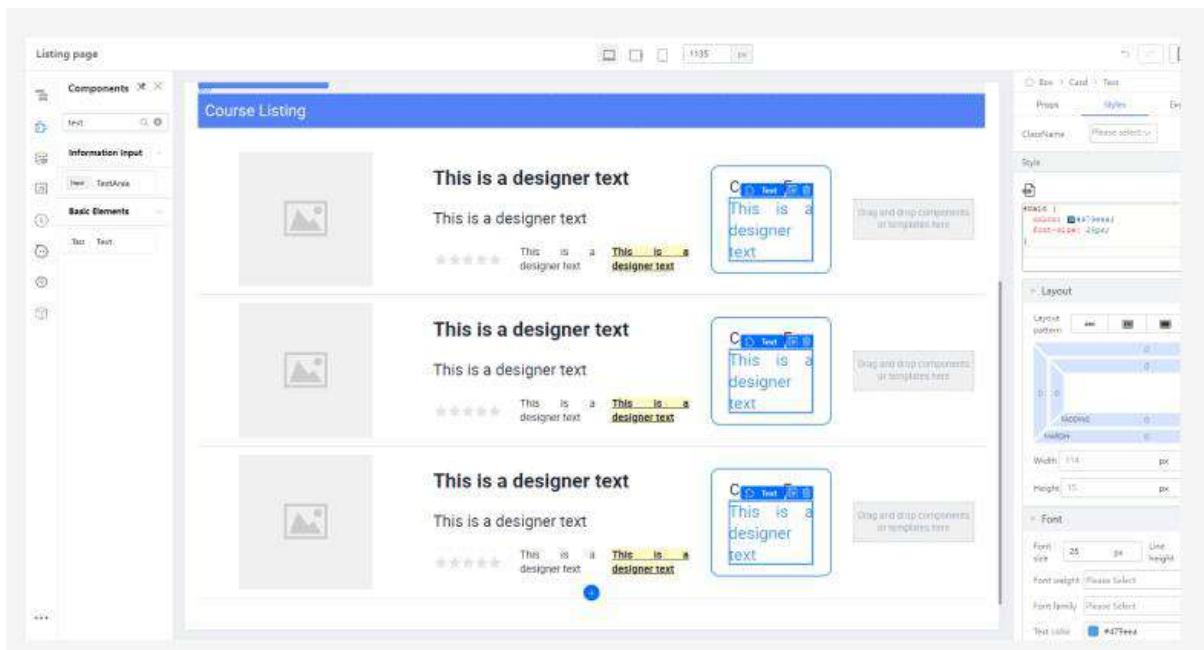
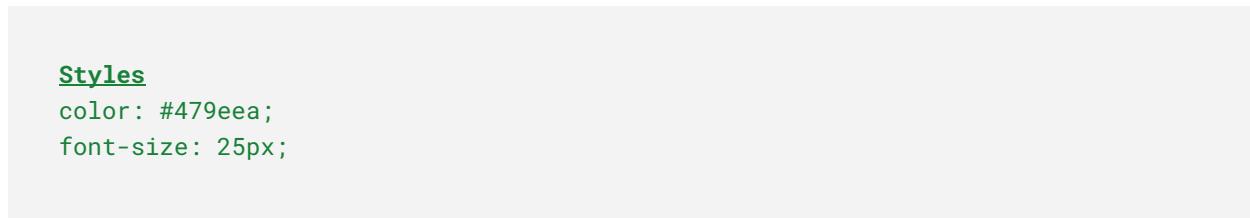
Card Content (Repeating Structure):

- Image:** Placeholder image icon.
- Title:** This is a designer text
- SubTitle:** This is a designer text
- Rating:** ★★★★☆
- Description:** This is a designer text. This is a designer text. This is a designer text.

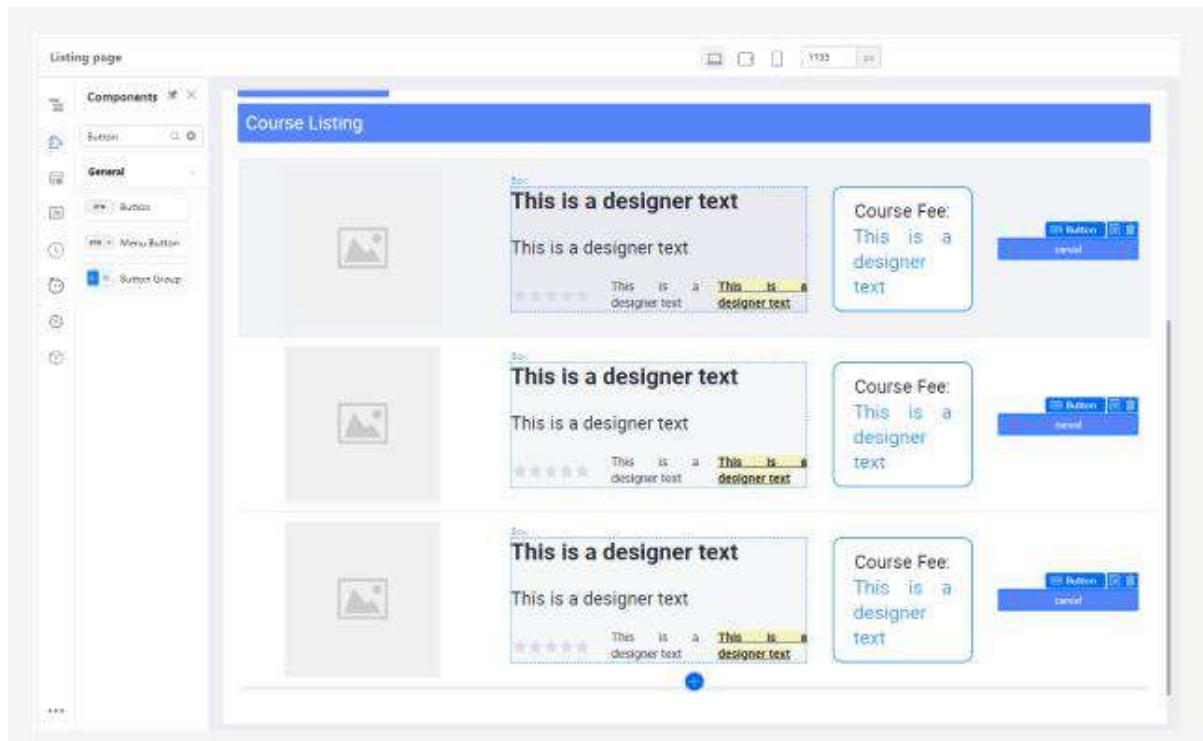
- Add a “Text” and set the following



- Add a second Text and set the following



- Add a Button to the last column



- Set the following:

Props

Content: View Courses

Styles

```
border-radius: 10px;
height: 50px;
font-size: 20px;
width: 150px;
```

Course Listing

This is a designer text

This is a designer text

★★★★★ This is a **This is a** designer text

Course Fee: This is a designer text

View courses

This is a designer text

This is a designer text

★★★★★ This is a **This is a** designer text

Course Fee: This is a designer text

View courses

This is a designer text

This is a designer text

★★★★★ This is a **This is a** designer text

Course Fee: This is a designer text

View courses

Slot Box Button

Props

ClassName: Please select

Style

```
#main {
  border-radius: 10px;
  height: 50px;
  font-size: 20px;
  width: 150px;
}
```

Layout

Width: 150

Height: 50

Font

Font size: 20 px

Font weight: Please Select

- Edit the table datasource with the following json data:

Data column

Title

- Name**
- Age**
- Responsibility**

Add an item +

Data source

Edit data (highlighted with a red box)

Sort icons desc

descending

Sort icons asc

ascending

listing-table-json.txt

Note: Remember to change the appID path on the courseImage

Upload these images to Asset->images: [elearning-education-internet-lessons-online.jpg](#), [data-analysis.jpeg](#), [Teachable.png](#)

Course Listing

Data editing

```

6   "rating": 3,
7   "ratingDescription": "Good",
8   "courseType": "Online",
9   "courseFee": "$499.99"
10 },
11 },
12 "courseImage": "https://encrypted-tbn0.gstatic.com/s
13 "title": "Data Science Fundamentals",
14 "subtitle": "Explore the world of data analysis",
15 "rating": 3,
16 "ratingDescription": "Good",
17 "courseType": "In-person",
18 "courseFee": "$399.99"
19 },
20 },
21 "courseImage": "https://process.fs.teachablecdn.com/
22 "title": "Web Development Bootcamp",
23 "subtitle": "Build dynamic and responsive websites",
24 "rating": 4,
25 "ratingDescription": "Good",
26 "courseType": "Online",
27 "courseFee": "$999.99"
28 }
29 }
```

This is a designer text

This is a designer text

View courses

OK Cancel

This is a designer text

View courses

Table Properties

Add an item +

Link bar

Add an item +

Top actions

Data column

Title

Name

Age

Responsibility

Add an item +

Data source

Sort icons desc

descending

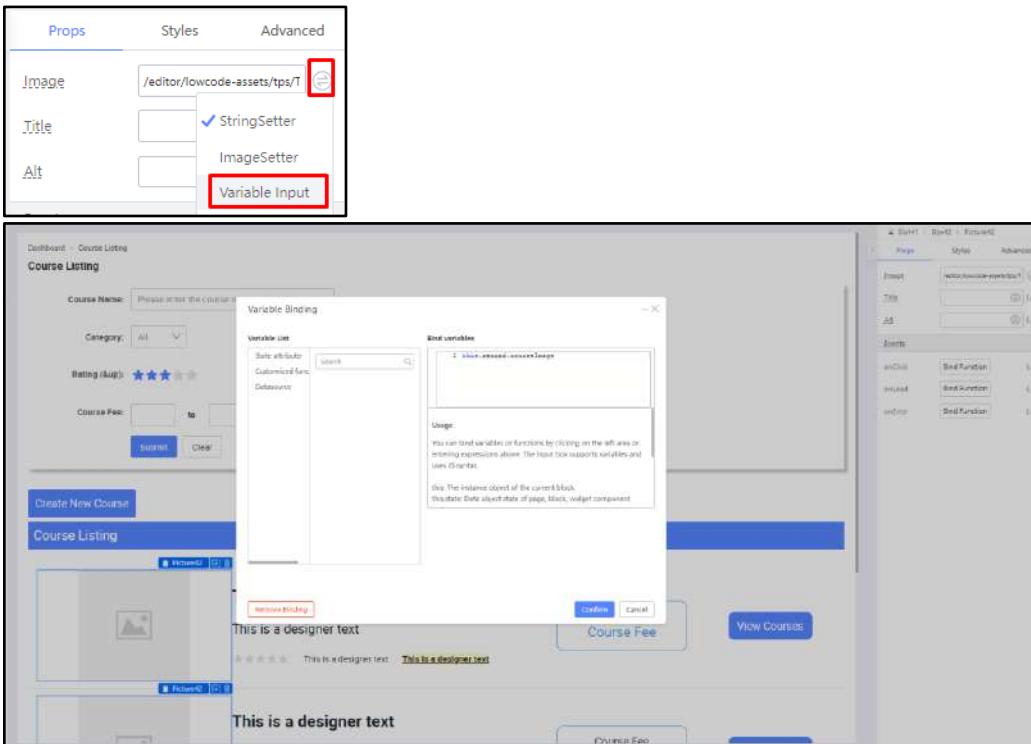
Sort icons asc

ascending

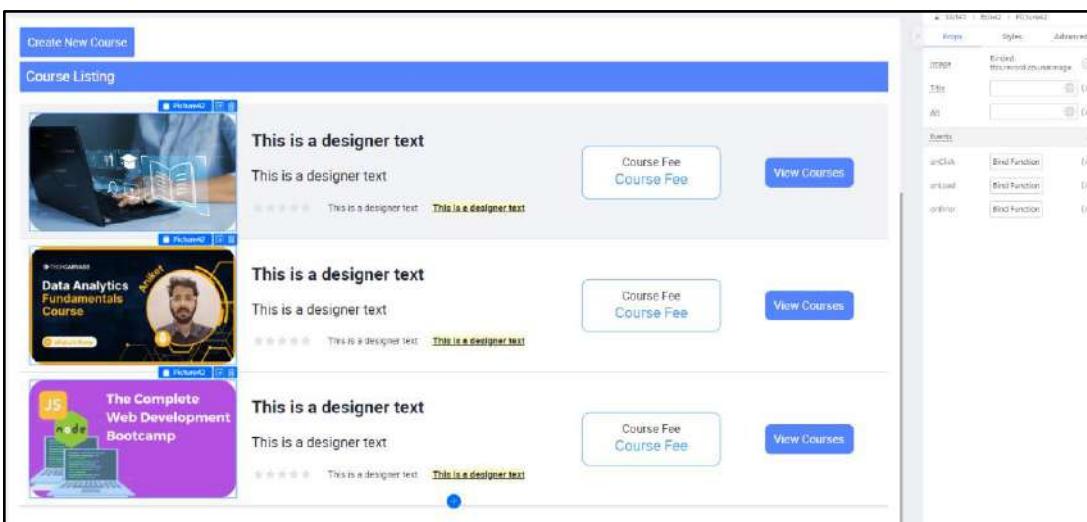
Action column

Add an item +

- In the Picture, select “Variable Input” in Switch Setter and set the following to bind variable



The expected result will be:



- Based on the numbers in the image below, variable bind these values to the following component in their **Props** tab:

No.	Field	Value to variable bind
1	Text	this.record.title
2	Text	this.record.subTitle
3	Current Value	this.record.rating
4	Text	this.record.ratingsDescription
5	Text	this.record.courseType
6	Text	this.record.courseFee



The expected result will be:

The screenshot shows a course listing interface with a blue header bar containing a 'Create New Course' button and a 'Course Listing' title. Below the header, there are three course cards, each with a thumbnail image, title, description, ratings, and a 'View Courses' button.

- Introduction to Programming**
Learn the basics of coding
★ ★ ★ ★ Good Online
Course Fee \$49.99 View Courses
- Data Science Fundamentals**
Explore the world of data analysis
★ ★ ★ ★ Good In-person
Course Fee \$79.99 View Courses
- The Complete Web Development Bootcamp**
Build dynamic and responsive websites
★ ★ ★ ★ Good Online
Course Fee \$99.99 View Courses

Tutorial 4: Creating a Form with Steps

This tutorial covers the following Learning Objectives:

- Understand how to design a multi-step form using KAIZEN.
- Learn how to break down complex forms into manageable steps for a smoother user experience.
- Explore how KAIZEN simplifies form creation by providing step-by-step navigation and validation between steps.

In this tutorial, you will learn how to design and build a multi-step form using KAIZEN. By breaking down a complex form into smaller, more manageable steps, you'll create a smoother and more engaging user experience. KAIZEN's tools will guide you through adding step-by-step navigation, validation, and data handling, ensuring that users can easily complete the form without confusion.

There will be 4 Forms in total:

Course Info

The form is a multi-step process titled "Course Info". It consists of four steps: Step 1 (Course Info), Step 2 (Instructor Particulars), Step 3 (Class Location), and Step 4 (Class Schedule). The current step is Step 1. The form fields include:

- * Course Title: Text input field
- * Course Code: Text input field
- * Enrollment Type: Drop-down menu with placeholder "Please select"
- * Subject Type: Drop-down menu with placeholder "Please select"
- * Course Fee: Text input field with a dollar sign prefix (\$)
- Description: Text area for entering course details

At the bottom of the form are "Previous" and "Next" buttons, and a small blue circular icon with a plus sign (+) in the center.

Instructor Particulars

Step 1 Course Info Step 2 Instructor Particulars Step 3 Class Location Step 4 Class Schedule

Instructor Particulars

[Retrieve Myinfo with Singpass](#)

* Salutation: Please select

* Name:

* NRIC:

* Email:

* Contact No.:

[Previous](#) [Next](#)

Class Location

Step 1 Course Info Step 2 Instructor Particulars Step 3 Class Location Step 4 Class Schedule

Class Location

* Postal Code:

* Address:

* Floor / Unit No.: -

[Previous](#) [Next](#)

and **Class Schedule**.

The screenshot shows a progress bar at the top with four steps:

- Step 1**: Course Info (checkmark)
- Step 2**: Instructor Particulars (checkmark)
- Step 3**: Class Location (checkmark)
- Step 4**: Class Schedule (blue outline, currently active)

Class Schedule

* Start Date: Select date

* End Date: Select date

Day	Start Time	End Time
Please select	Please select time <input type="button" value="Select time"/>	Please select time <input type="button" value="Select time"/>

+ Add Schedule

Previous

Introduction: Form Fields

Forms are components that accept inputs from various Form fields and then, through user interaction (usually a button press), submit the field information from the form to the backend for processing.

Previously, we encountered forms in Tutorial 1. A refresher on what the form looked like:

The form consists of a rectangular container with a thin black border. Inside, there are two input fields. The first is labeled "Email:" with a red asterisk, containing the placeholder text "name@example.com". The second is labeled "Password:" with a red asterisk, featuring a standard text input field and a small eye icon to its right for password visibility. At the bottom left of the form area, there is a link "Don't have an account? Sign up" in a smaller font. At the bottom right, there is a prominent blue rectangular button with the word "Login" in white text.

As you can see, the form contains an Email Field, a Password Field and a Login Button.

The Email Field is known as an **Input Field**, which accepts open-ended values of text you type from the keyboard. The Password field is a special Input Field that does what the Email field does, but also masks the input so that others cannot see what is being typed on screen.

Finally, the **Login button** submits the values that the user keyed in for each field, possibly to an Authentication Server.

While this is sufficient for a Login Form, modern web languages also support a multitude of other form fields, like radio buttons, dropdown selects, date pickers and more. Fortunately, Kaizen supports a good number of these.

In order to access them in the Kaizen App Designer, you can go to the Component Library and search for them.

We will get to use them when we get to the practicals later on.

Introduction: Form Validation

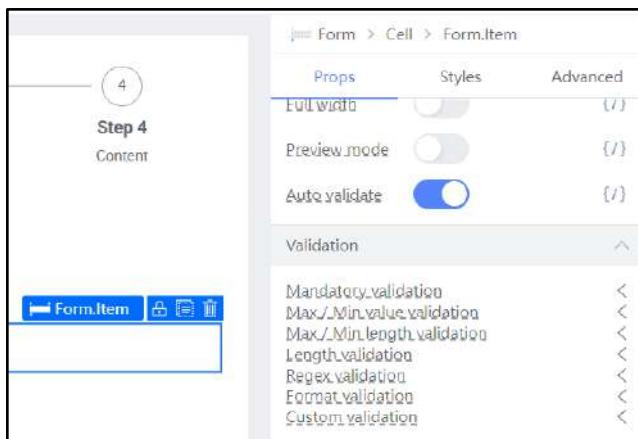
Validation (or Form Control) is a set of formatting rules defined for the browser to check. These validation rules usually act as an input format sanity check or as the first layer of defense against user-submitted dirty data, by allowing the browser to know when to warn and restrict the user before submission. The users are usually then allowed to change their inputs based on the warnings given by the browser.

You can see this in action, if you ever played around with the Login Form above, and tried to submit without any values typed into the form:

The form contains two fields: 'Email' and 'Password'. Both fields have red borders and the placeholder text 'Required Value' below them. The 'Email' field has the value 'name@example.com'. Below the form are links for 'Don't have an account?' and 'Sign up', and a blue 'Login' button.

This is because the Form Template that is in App Designer's Component Library has Mandatory Validation enabled by default for both its form fields. Mandatory Validation on means that the field in question has to have a value in it for the form to treat it as a valid submission.

Of course, Mandatory Validation is the simplest validation. There are other more complex validation types that might find more use in more complex form fields. In Kaizen's App Designer, the Validations can be found in the Props tab for a Form.Item component.



Practical 4.1: Setting up the Form

4.1.1 Creating the Form skeleton

Before the Tutorial begins, please do the following steps. This is to set up the form for the rest of the Practicals. Most of these should be familiar if you have done the Tutorials before, but still, try and closely follow the following steps:

- **Create a new Form Page.**

The image consists of two screenshots of a software interface for creating a new page.

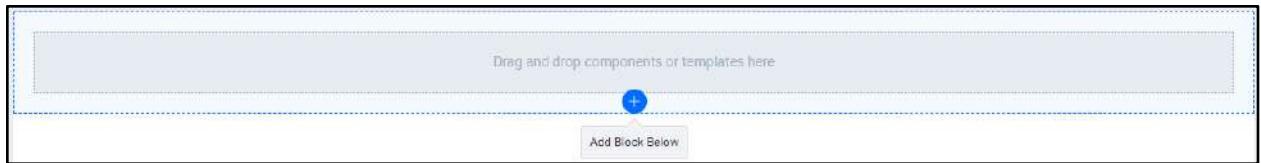
The top screenshot shows a "Resources" panel with a "Pages" section. A context menu is open over the "Pages" button, with the "Create Page" option highlighted and surrounded by a red box.

The bottom screenshot shows a "Create Page" dialog box with the following fields filled out:

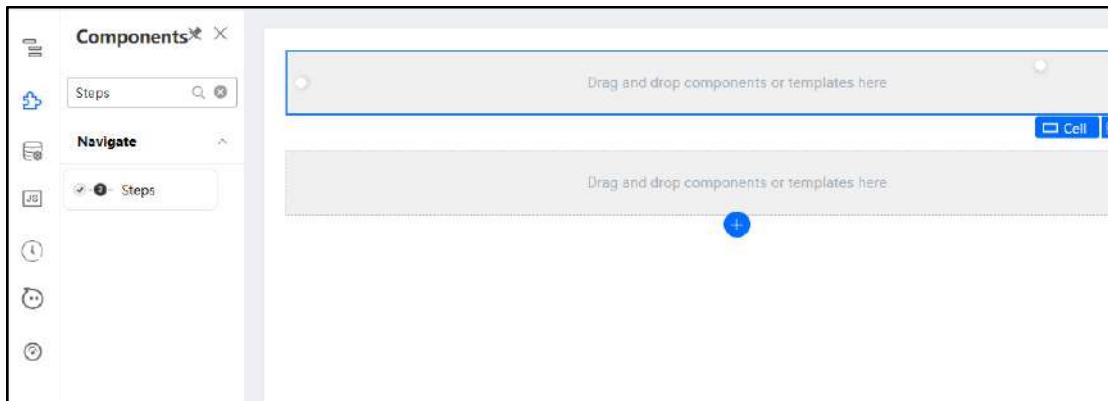
- * Application: DesignerTraining
- * Name: FormPage
- * Page ID: formpage
- * Type: Page
- Description: (empty)
- * Status: Active (radio button selected)
- * Default Page: No (radio button selected)
- * Is Protected: No (radio button selected)
- Privileges: Default domain/Anonymous Resource

At the bottom of the dialog box are "Cancel" and "Save" buttons.

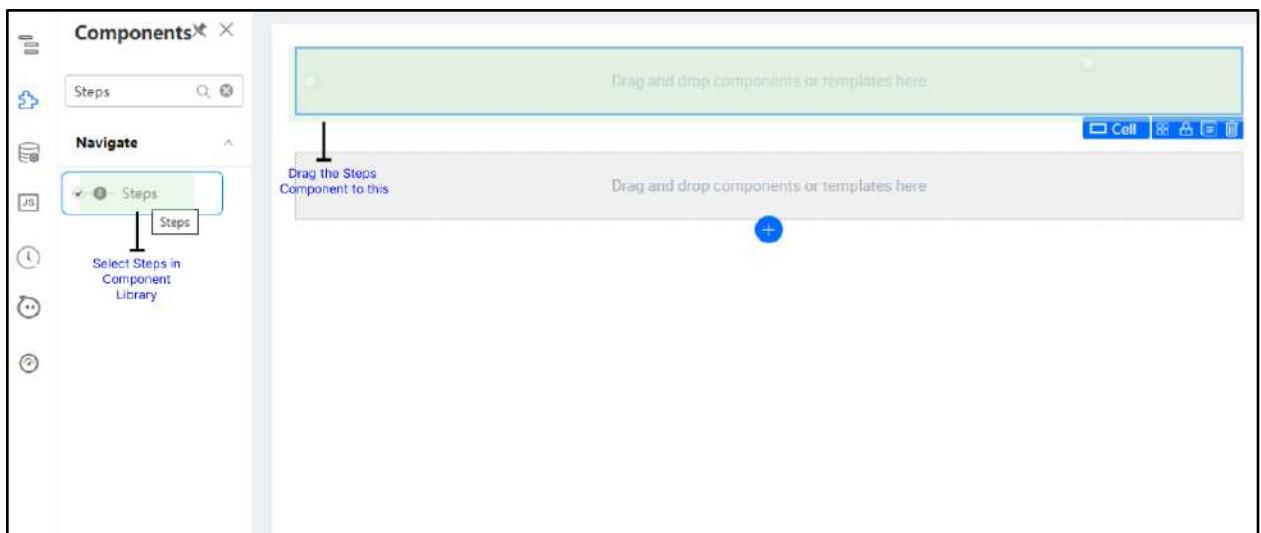
- Click on **Add Block Below** at the bottom.



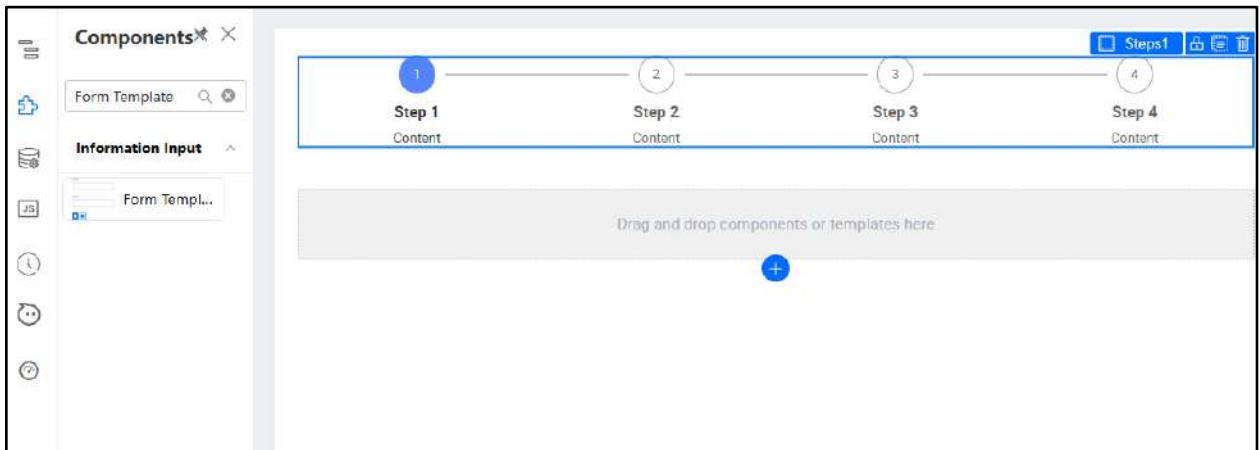
- **Search for Steps component**



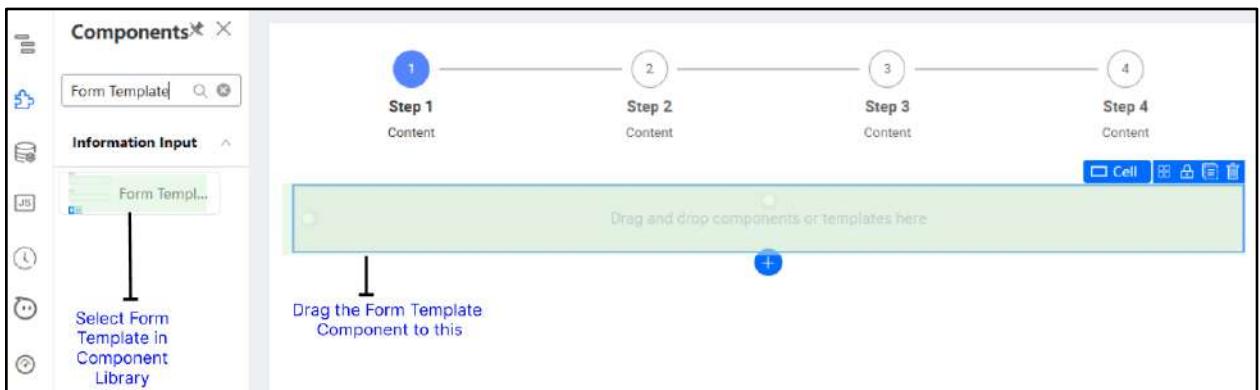
- **Drag the Steps component to the top empty Cell.**



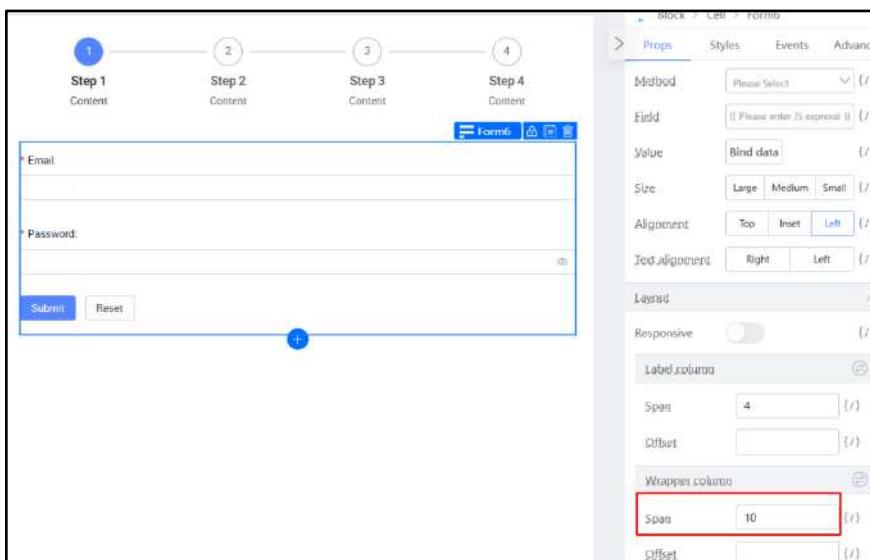
- Search for **Form Template** component



- Drag the **Form Template** component to the bottom empty Cell.

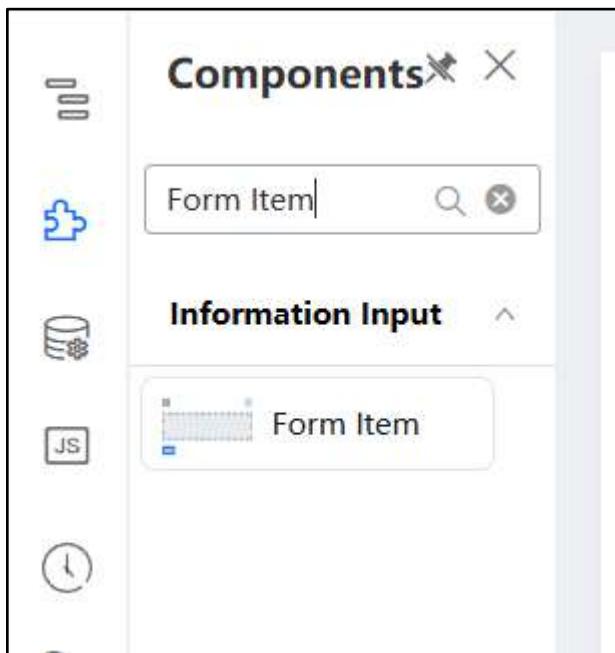


- Select Form component, under **Properties (Props)** of **Form Component**, set the following:

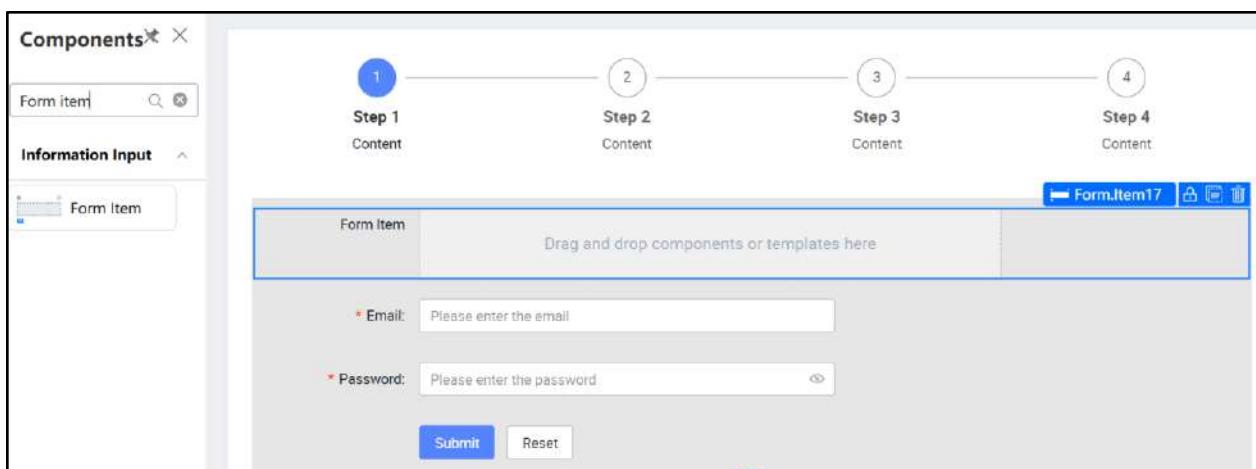




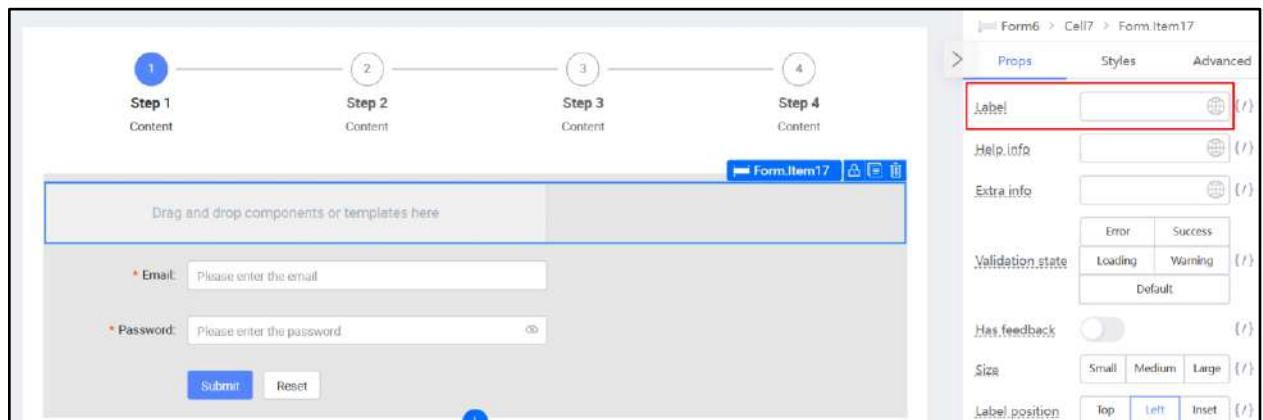
- Search for **Form Item** component



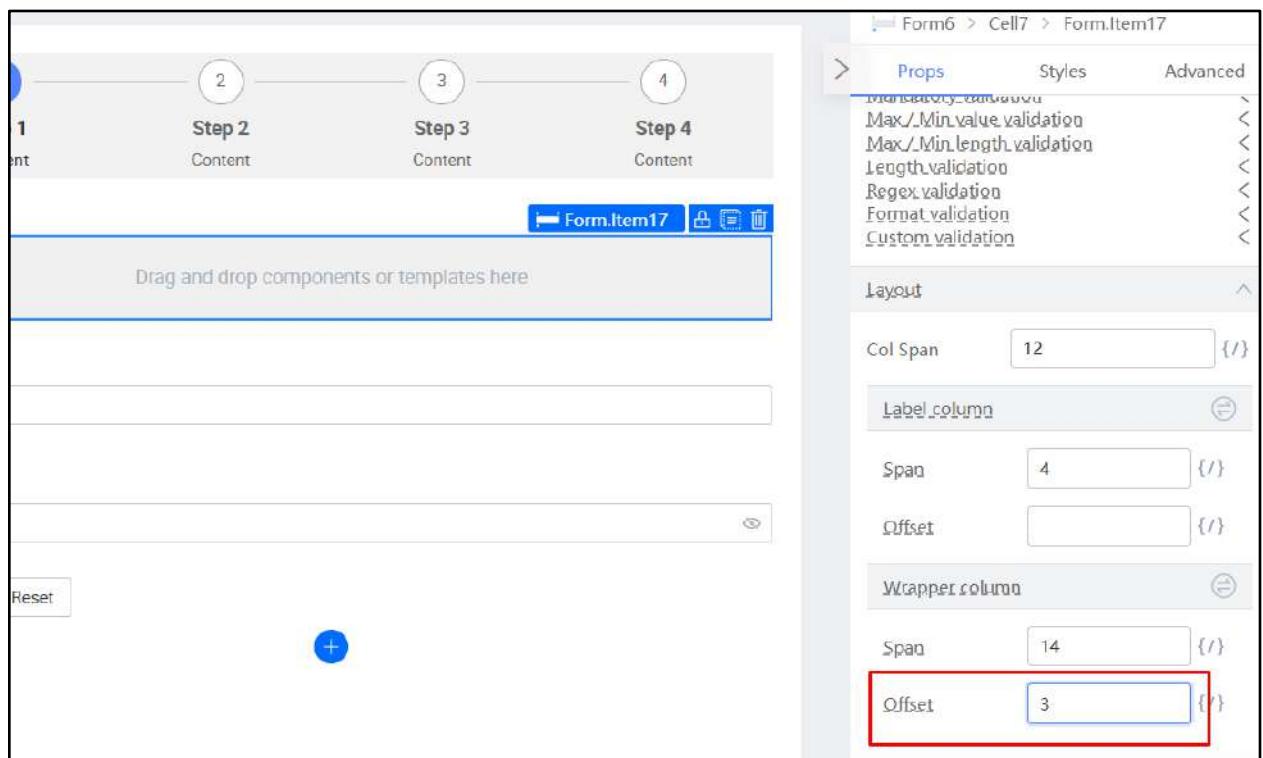
- Drag the **Form Item** component above the Email field.



- **Clear the Label field**



- **Change the Wrapper column's Offset to 3.**



- Drag a **Text component** to the top of the Form Template. **Change** the Styling and Properties of the text component to the following:

The screenshot shows a 'Components' panel on the left containing a 'text' component under 'Information Input'. The main area displays a four-step form template. Step 1 contains a 'Text' component labeled 'Text19' with the placeholder 'This is a designer text'. Step 2, Step 3, and Step 4 each contain a 'Content' section. Below the steps are two input fields: 'Email' (placeholder 'Please enter the email') and 'Password' (placeholder 'Please enter the password'). At the bottom are 'Submit' and 'Reset' buttons.

Props

Text: Course Info

Styles

```
font-size: 24px;
font-weight: 600;
font-family: Arial;
```

The expected output will be

The screenshot shows the final expected output of the form template. It features a blue header bar with icons for 'FDPage', 'Import', and 'Delete'. Below the header is a four-step flow: Step 1 (Content), Step 2 (Content), Step 3 (Content), and Step 4 (Content). A large 'Course Info' heading is centered above the input fields. The 'Email' and 'Password' input fields have the same placeholder as in the template. At the bottom are 'Submit' and 'Reset' buttons.

- Click on the Form Item containing area (as illustrated in the image below)



- Change the Wrapper column's Offset to 4.

A screenshot of a UI builder interface showing a step-by-step form. Step 4 is selected. The right panel displays the properties for 'Form.Item14' under the 'Layout' tab. The 'Offset' field is highlighted with a red border, set to 4. Other visible settings include 'Col Span' (12), 'Label.column' (1), 'Span' (10), and 'Wrapper.column' (1).

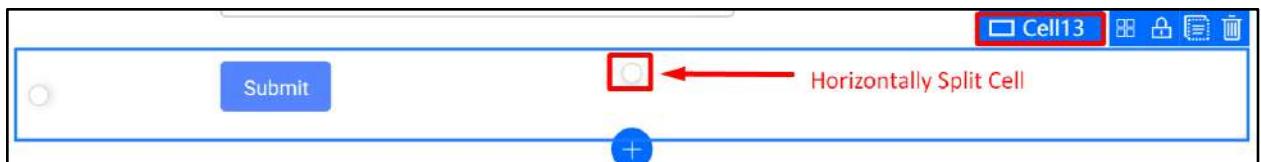
- Remove the Reset button

A screenshot of a UI builder interface showing the same step-by-step form. The 'Reset' button has been removed from the buttons bar at the bottom of the form. The right panel shows the properties for 'Form.Reset16', which is now listed as a component in the buttons bar.

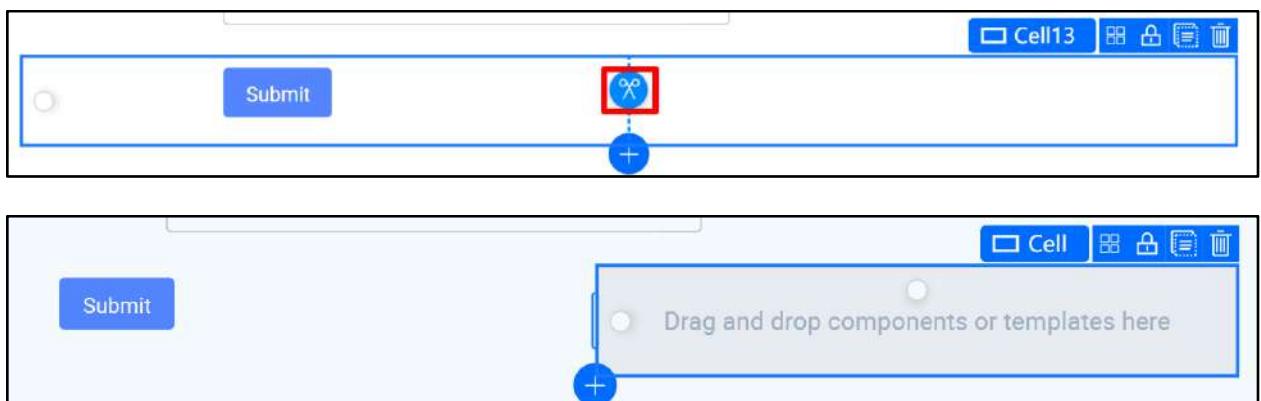
- Click on the Form Item containing area



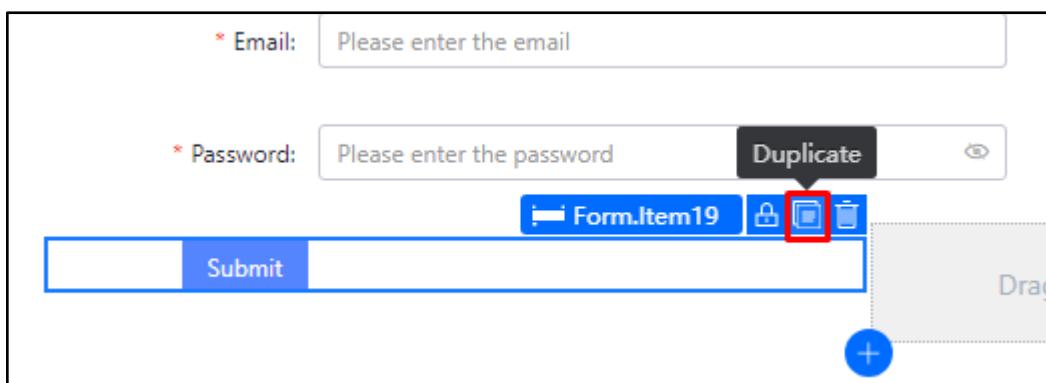
- Ensure **Cell** is selected (Note you also see a white circle, which is a split cell icon)



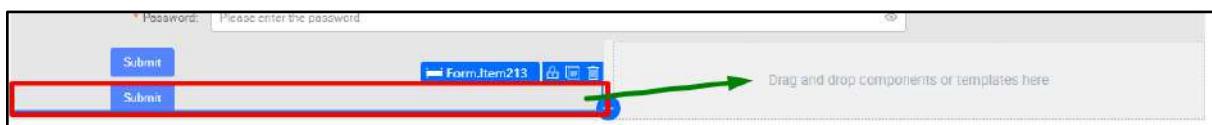
- Click the **split cell** icon to split the cell into two



- Select the Form Item with the submit button on the left side and click **Duplicate**.



- Drag the duplicated Form Item with the submit button to the right side.



The expected output will be:

The diagram illustrates a four-step form process. At the top, four circular steps are arranged horizontally, labeled 1 through 4. Step 1 is highlighted with a blue background and contains the text "Step 1 Content". Steps 2, 3, and 4 are white circles with black outlines, each containing the text "Step 2 Content", "Step 3 Content", and "Step 4 Content" respectively. Below this header is a large gray rectangular area titled "Course Info" in bold black font. Inside this area are two input fields: one for "Email" and one for "Password". Both fields have a red asterisk icon to their left indicating they are required. Below the input fields is a horizontal button bar. On the left side of the bar is a blue button labeled "Submit". To the right of the bar is a blue header labeled "Form.Item14" followed by three small icons: a lock, a copy symbol, and a trash can. On the far right of the bar is another blue "Submit" button.

- Select the Form Item on the right (illustrated below)



- Set the properties of Form Item to below

Props

Validation

Layout

Property	Value
Col Span	4
Label column	0
Span	14
Offset	10

Button's Form.Item:

Props

Size: Large

Layout

Label Column

Span: 0

Offset: 0

Wrapper Column

Span: 14

Offset: 10

Size	Small	Medium
	Large	

Layout

Col Span: 4 {/}

Label column:

Span: 0 {/}

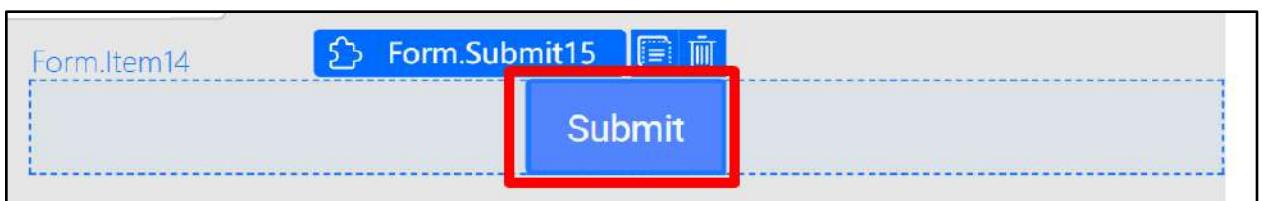
Offset: 0 {/}

Wrapper column:

Span: 14 {/}

Offset: 10 {/}

- Select the Submit button



- Under Properties Label, change **Submit** to **Next**.

Button:

Props

Label: Next

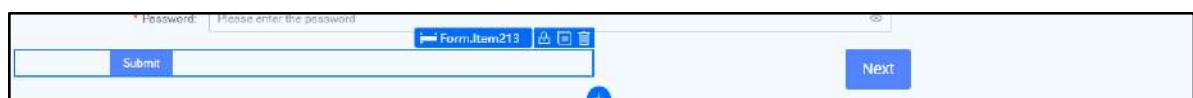
You should expect to see something like this:



- Style and configure the left button into a 'Previous' button.

Select the Form.Item. Under the Props tag, change Size to Large. Under Layout, change the Wrapper span to 14 and the offset to 5.

Select the Button. Under Content, change 'Submit' to 'Previous', Button Type to Secondary.



Button's Form.Item:

Props

Size: Large

Layout

Label Column

Span: 0

Offset: 0

Wrapper Column

Span: 14

Offset: 5

Button:

Props

Label: Previous

Button Type: Secondary

And the button should look like this:



- Select the Email Form Item Label by clicking on the label (illustrated below)

Step 1 Content

Step 2 Content

Step 3 Content

Step 4 Content

Course Info

* Email

* Password:

Previous Next

- Change the Styling and Properties of the Email Form Item.

Email Form.Item Props

Size: Large
(Advanced) Name: (Empty, clear the property 'email')

Email Form.Item Styles

```
font-weight: 500;
```

Email Input Component Props

Clear the Placeholder

- Select the Input component

Form.Item8

* Email:

Input9

- Clear the Placeholder under properties

Step 1 Content Step 2 Content Step 3 Content Step 4 Content

Course Info

* Email:

* Password:

Previous Next

Props Styles Events Advance

Label: Placeholder: Value: Default:

Default value: State: Error Success Warning Loading

Size: Small Medium Large

Email Input Component Props

Clear the Placeholder

- Select the Cell containing area (illustrated below)

Course Info

* Email:

Cell10

* Password:

Form.item11

- Ensure the Cell is selected

Cell10

* Password:

- Remove the Cell containing Password Form Item

Course Info

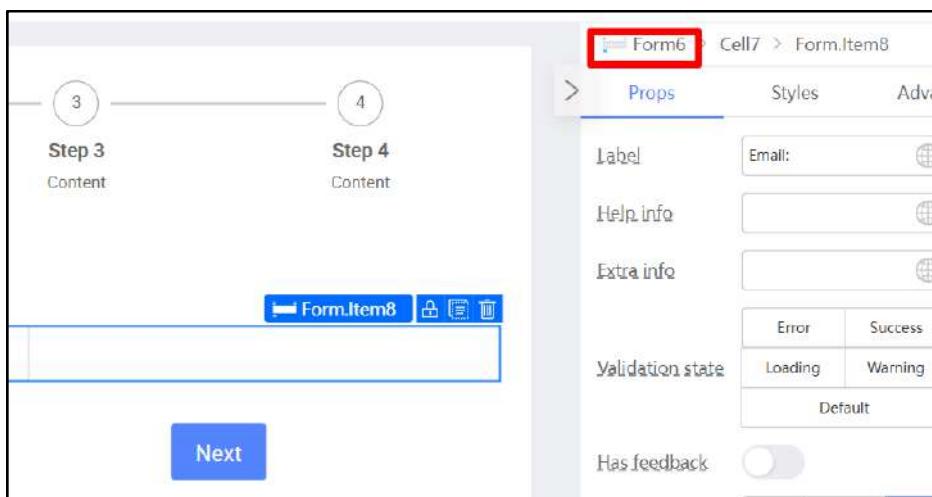
* Email:

* Password:  

 Cell10   

[Previous](#) [Next](#)

- Select the Form component (illustrated below)



Form6 > Cell7 > Form.Item8

Props Styles Advanced

Label: Email 

Help info:

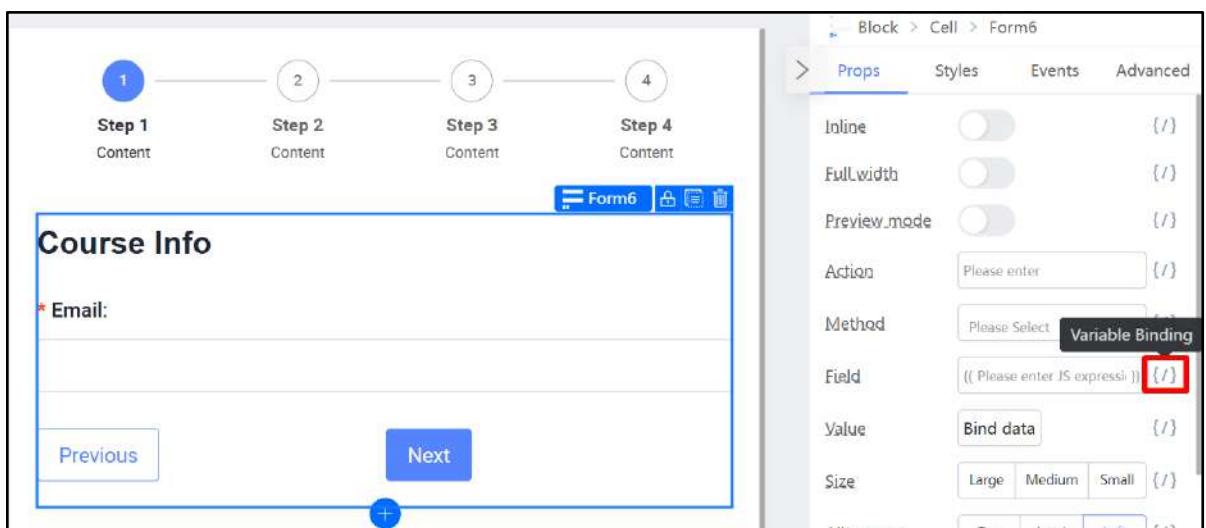
Extra info:

Validation state:

Error	Success
Loading	Warning
Default	

Has feedback: 

- Under the Props tab, click on the Field - Variable Binding (looks like this {/}).



Block > Cell > Form6

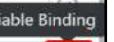
Props Styles Events Advanced

Inline:  (/)

Fullwidth:  (/)

Preview mode:  (/)

Action: Please enter 

Method: Please Select  Variable Binding

Field:  (/)

Value: Bind data 

Size: Large Medium Small 

- Under the Bind variables text area, key in **this.formField**. We will go more into this variable later. For now, just click the Confirm button after typing it in.

Variable Binding

- X

Variable List

State attribute
Customized func
Datasource

Search

Bind variables

```
1 this.formField
```

Usage

You can bind variables or functions by clicking on the left area or entering expressions above. The input box supports variables and uses JS syntax.

this: The instance object of the current block

this.state: Data object state of page, block, widget component instance

Remove Binding

Confirm

Cancel

- **Duplicate the Course Info Form 3 times.** There should be 4 ‘Course Info’ Forms now.

The screenshot shows a horizontal navigation bar with four steps: Step 1 (Content), Step 2 (Content), Step 3 (Content), and Step 4 (Content). Below the bar is a form titled "Course Info" with a single field labeled "Email". At the bottom are "Previous" and "Next" buttons. In the top right corner of the form area, there is a blue button labeled "Form6" with a red box drawn around it. A blue arrow points from the text "Duplicate this until there are 4 copies" to this button.

- Change the text in the bottom 3 Forms to ‘Instructor Particulars’, ‘Class Location’, ‘Class Schedule’ in order.

The screenshot shows the same four-step form process. The steps are now labeled: Step 1 (Course Info), Step 2 (Instructor Particulars), Step 3 (Class Location), and Step 4 (Class Schedule). Each step has a "Course Info" label above it. The "Email" field in each step is identical. The "Previous" and "Next" buttons are at the bottom of each step. The top right corner of the form area shows standard file icons (File, Print, etc.) and a blue arrow pointing to the text "Duplicate this until there are 4 copies".

- Change the Step sub-headers to their respective Step names:

Step 1: Course Info

Step 2: Instructor Particulars

Step 3: Class Location

Step 4: Class Schedule



The expected output will be:

1 Step 1 Course Info
2 Step 2 Instructor Particulars
3 Step 3 Class Location
4 Step 4 Class Schedule

Course Info
* Email:

Previous Next

Instructor Particulars
* Email:

Previous Next

Class Location
* Email:

Previous Next

Class Schedule
* Email:

4.1.2 Form Setup Complete

The image shows a step-by-step form setup interface. At the top, there is a navigation bar with four steps: Step 1 (Course Info), Step 2 (Instructor Particulars), Step 3 (Class Location), and Step 4 (Class Schedule). Each step has a corresponding input field for an email address and a 'Next' button. A 'Previous' button is also present for steps 2, 3, and 4. A blue circular progress indicator is located at the bottom center of the form.

Step 1
Course Info

Step 2
Instructor
Particulars

Step 3
Class Location

Step 4
Class Schedule

Course Info

* Email:

[Previous](#) [Next](#)

Instructor Particulars

* Email:

[Previous](#) [Next](#)

Class Location

* Email:

[Previous](#) [Next](#)

Class Schedule

* Email:

[Previous](#) [Next](#)

⊕

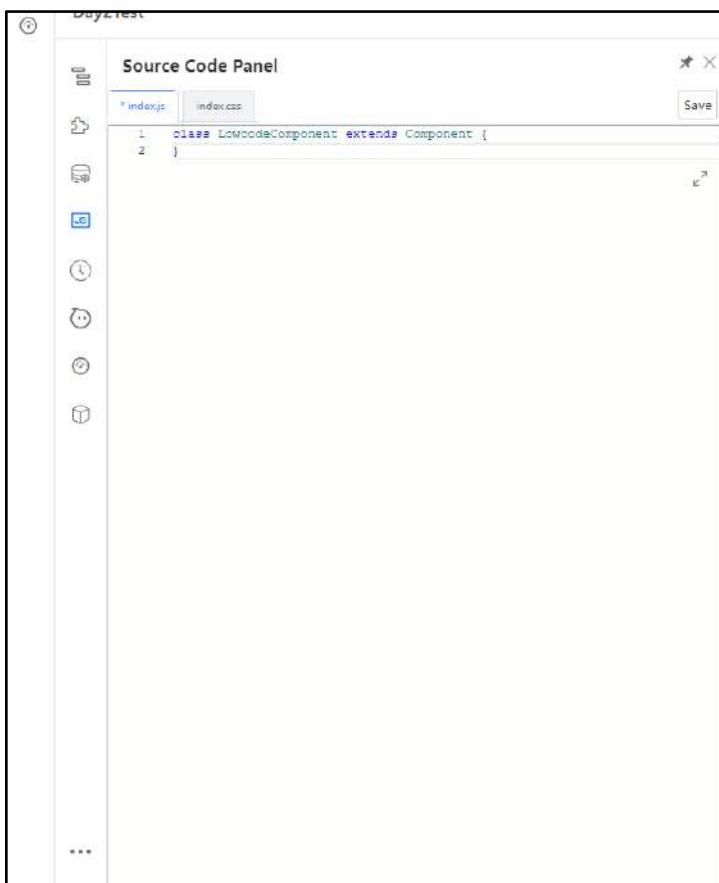
4.1.3 The Source Code Panel

Let's bring over the code we will be using for today's tutorial to the Source Code Panel:

- In the Kaizen App Designer, open the Source Code Panel on the left.



- You should see the **Source Code Panel** appear on the left of the canvas:



- Click on the gray button below to open a text file:
 - [Tutorial4-Code.txt](#)

```

class LowcodeComponent extends Component {
  /** State Variables. */
  // This stores global variables for the page
  // which can be used in this code or for variable binding in Components.
  state = {
    "currentStep": 0,
    // "startError": '',
    // "endError": '',
    // "weekly": [],
    // "day": undefined,
    // "start": undefined,
    // "end": undefined
    // ],
    // "enrollmentSubject": [],
    // "enrollmentId": 1,
    // "subjectId": ["Accounting", "Business", "Computing"],
    // ],
    // "enrollmentId": 2,
    // "subjectId": ["Accounting", "Business", "Computing", "Design", "Education"]
    // ],
    // "enrollmentId": 3,
    // "subjectId": ["Computing", "Education"]
    // ],
    // "subjectTypeVisible": false,
    // "subjectTypeArray": {}
    // "showAll": true
  }

  /**
   * Constructor. */
  // This function is called when the page is initialized.
  // Next.Field is a library in Kalem that allows the programmer to set and get values/errorcode in forms.
  constructor() {
    this.formField = new Next.Field(this);
  }

  /**
   * 4.5.4 */
  // Function to check if the Start Date is before the End Date.
  // Sets error if the Start Date is after the End Date.
  // checkDates(rule, value, callback) {
  //   if (this.formField.getValue("startDate") === undefined || this.formField.getValue("endDate") === undefined) {
  //     if (moment(this.formField.getValue("startDate"), 'YYYY-MM-DD').isAfter(this.formField.getValue("endDate")))) {
  //       this.setState({
  //         startError: 'error',
  //         endError: 'error'
  //       })
  //       callback("Start Date cannot be after End Date!");
  //     }
  //   }
}

```

- Copy everything in this file to the Text Area in the Source Code Panel, like so:

```

Source Code Panel
index.js index.css Save

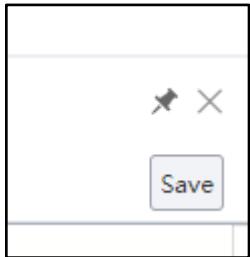
class LowcodeComponent extends Component {
  /** State Variables. */
  // This stores global variables for the page
  // which can be used in this code or for variable binding in Components.
  state = {
    "currentStep": 0,
    // "startError": '',
    // "endError": '',
    // "weekly": [],
    // "day": undefined,
    // "start": undefined,
    // "end": undefined
    // ],
    // "enrollmentSubject": [],
    // "enrollmentId": 1,
    // "subjectId": ["Accounting", "Business", "Computing"],
    // ],
    // "enrollmentId": 2,
    // "subjectId": ["Accounting", "Business", "Computing", "Design", "Education"]
    // ],
    // "enrollmentId": 3,
    // "subjectId": ["Computing", "Education"]
    // ],
    // "subjectTypeVisible": false,
    // "subjectTypeArray": {}
    // "showAll": true
  }

  /**
   * Constructor. */
  // This function is called when the page is initialized.
  // Next.Field is a library in Kalem that allows the programmer to set and get values/errorcode in forms.
  constructor() {
    this.formField = new Next.Field(this);
  }

  /**
   * 4.5.4 */
  // Function to check if the Start Date is before the End Date.
  // Sets error if the Start Date is after the End Date.
  // checkDates(rule, value, callback) {
  //   if (this.formField.getValue("startDate") === undefined || this.formField.getValue("endDate") === undefined) {
  //     if (moment(this.formField.getValue("startDate"), 'YYYY-MM-DD').isAfter(this.formField.getValue("endDate")))) {
  //       this.setState({
  //         startError: 'error',
  //         endError: 'error'
  //       })
  //       callback("Start Date cannot be after End Date!");
  //     }
  //   }
}

```

- Click the **Save** button on the top right of the Source Code Panel.



In this tutorial, the Source Code Panel will be used to handle some of the more complex logic for the forms not covered by the UI. **If you are already familiar with JS and coding in general, you can skip to the next section (Practical 4.2).**

Note that you will not be required to actually code. However, you will be required to do some uncommenting of the code. The following steps will guide you through the process:

- **Highlight** a section of a commented code (text in green). A commented section of code will not be executed by the application. Essentially, it will be ignored.

```
// // Function to check if the Start Date is before the End Date.
// // Sets error if the Start Date is after the End Date.
// checkDates(rule, value, callback) {
//   if (this.formField.getValue("startDate") !== undefined && this.formField.getValue("endDate") !== undefined) {
//     if (moment(this.formField.getValue("startDate"), 'YYYY-MM-DD').isAfter(moment(this.formField.getValue("endDate"), 'YYYY-MM-DD'))) {
//       this.setState({
//         startError: 'error',
//         endError: 'error'
//       })
//       callback("Start Date cannot be after End Date!");
//     }
//     else {
//       this.setState({
//         startError: 'default',
//         endError: 'default'
//       })
//       callback();
//     }
//   }
//   return;
// }
```

- Press **Ctrl + /** to **uncomment** the code.

```
// // Function to check if the Start Date is before the End Date.  
// // Sets error if the Start Date is after the End Date.  
checkDates(rule, value, callback) {  
  if (this.formField.getValue("startDate") !== undefined && this.formField  
    if (moment(this.formField.getValue("startDate"), 'YYYY-MM-DD').isAfter(  
      this.setState({  
        startError: 'error',  
        endError: 'error'  
      })  
      callback("Start Date cannot be after End Date!");  
    }  
    else {  
      this.setState({  
        startError: 'default',  
        endError: 'default'  
      })  
      callback();  
    }  
  }  
  return;  
}  
|
```

There is an easy way to tell through the editor if the code is commented. If the text is **green**, it is commented. If the text is **not green**, it is not commented.

Make sure to **save** your changes.



Practical 4.2: Form Fields and Form Validation

4.2.1 Context

The **1st Step: Course Info**, is a form for filling in the basic Course information for students to refer to. By the end of this Practical, this is the form you will be creating:

Course Info

* Course Title

* Course Code

* Enrollment Type Please select

* Subject Type Please select

* Course Fee \$

Description

Before we start: this Practical will only cover the ‘Course Info’ Form. Only work within this Form:

The image shows a sequential form with four steps: Step 1 Course Info, Step 2 Instructor Particulars, Step 3 Class Location, and Step 4 Class Schedule. The current step is Step 1. Each step contains a single input field labeled 'Email:' and a 'Next' button. Step 4 includes a 'Previous' button. A blue plus sign is located at the bottom center of the form area.

Step 1
Course Info

Step 2
Instructor
Particulars

Step 3
Class Location

Step 4
Class Schedule

Course Info

* Email:

Previous Next

Instructor Particulars

* Email:

Previous Next

Class Location

* Email:

Previous Next

Class Schedule

* Email:

Previous Next

+

4.2.2 Objective

Learn about:

- Creating a basic form
- Mandatory, Length and Number Format validation

4.2.3 Basic Form Setup

- Select the Email Form Item

Course Info

Form.Item8

* Email:

Previous Next



- Duplicate it 5 times. You should now have 6 Form Items in this form, like so:

Particulars

Course Info

Duplicate

Form.Item8

* Email:

Course Info

* Email:

* Email:

* Email:

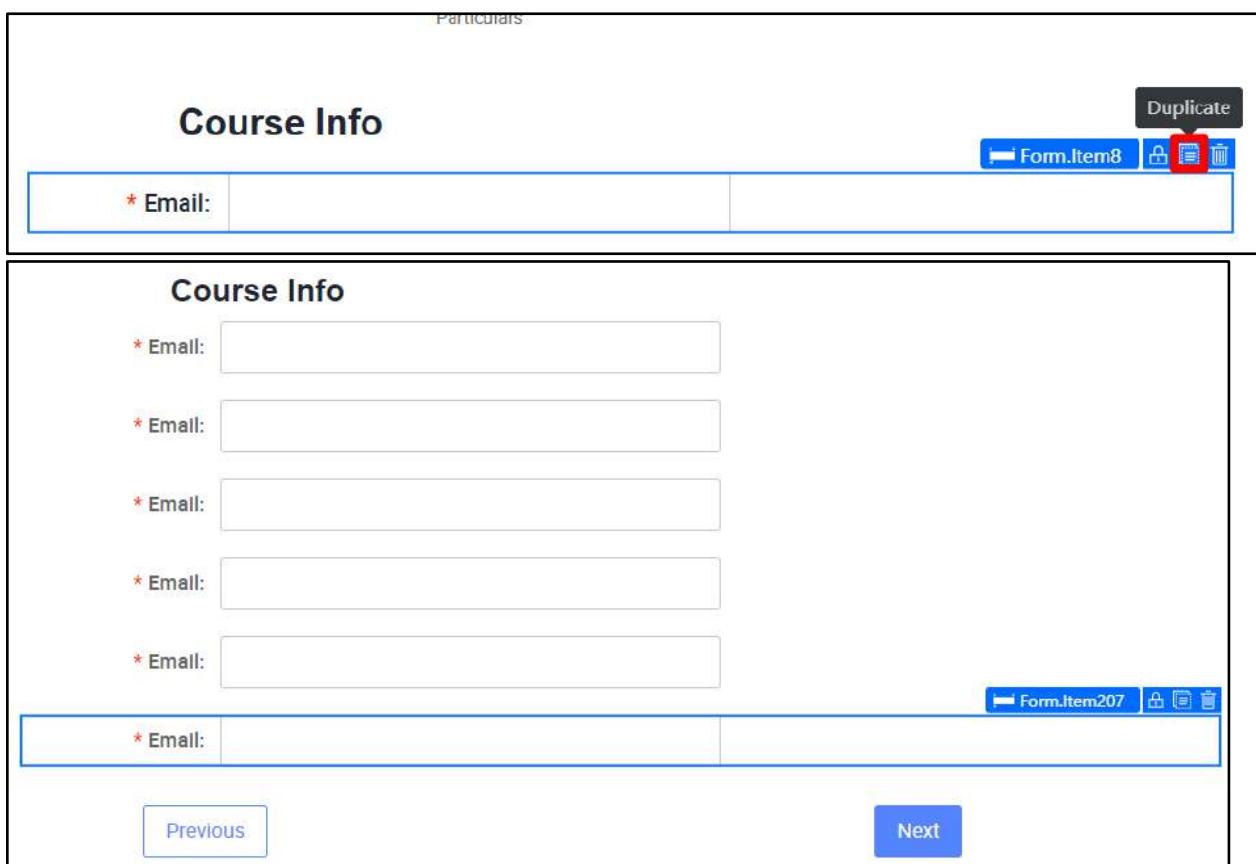
* Email:

* Email:

Form.Item207

* Email:

Previous Next



- Select the first Email Form Item and change the Label to Course Title.

The screenshot shows a 'Course Info' form with a single input field for 'Course Title'. To the right is a properties panel with tabs for 'Props', 'Styles', and 'Advanced'. The 'Label' field in the 'Props' tab is highlighted with a red box and contains the value 'Course Title'. Other fields in the panel include 'Help info', 'Extra info', 'Validation state' (with options for Error, Success, Loading, Warning), 'Has feedback', 'Size' (Small, Medium, Large), 'Label position' (Top, Left, Inset), 'Label alignment' (Left, Right), 'Full width', 'Preview mode', and 'Auto validate'.

- Then, with the Form Item still selected, change the Name Property to courseTitle

This screenshot shows the same 'Course Info' form, but now it includes two additional sections: 'Instructor Particulars' and 'Class Location'. The 'Course Title' field has been moved to the 'Instructor Particulars' section. The properties panel on the right shows the 'Name' field in the 'Props' tab highlighted with a red box and containing the value 'courseTitle'. The 'Name' field is located under the 'Advanced' tab as well.

- Now, perform the same steps for each of the **next 5 Form Items** but with the following info instead

Form Item Label	Form Item Name Property
Course Code	courseCode
Enrollment Type	enrollmentType
Subject Type	subjectType
Course Fee	courseFee
Description	description

After entering the info, the form should now look something like this:

Course Info

* Course Title	<input type="text"/>
* Course Code	<input type="text"/>
* Enrollment Type	<input type="text"/>
* Subject Type	<input type="text"/>
* Course Fee	<input type="text"/>
* Description	<input type="text"/>

Previous
Next

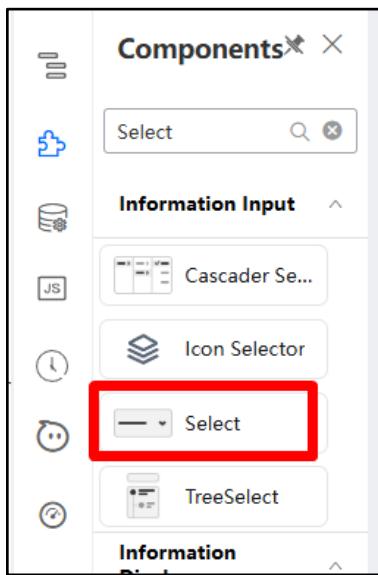
- **Ctrl + Click** the **Enrollment Type**, **Subject Type** and **Description** Input Components.
 - Press the **Delete/Backspace** key to **delete** them.

The screenshot shows a software interface with four input fields. Each field is preceded by a red asterisk (*) and has a blue 'Input208' button and a trash can icon to its right. The fields are labeled: 'Enrollment Type', 'Subject Type', 'Course Fee', and 'Description'. The background is light gray, and the input fields have a white background with a thin blue border.

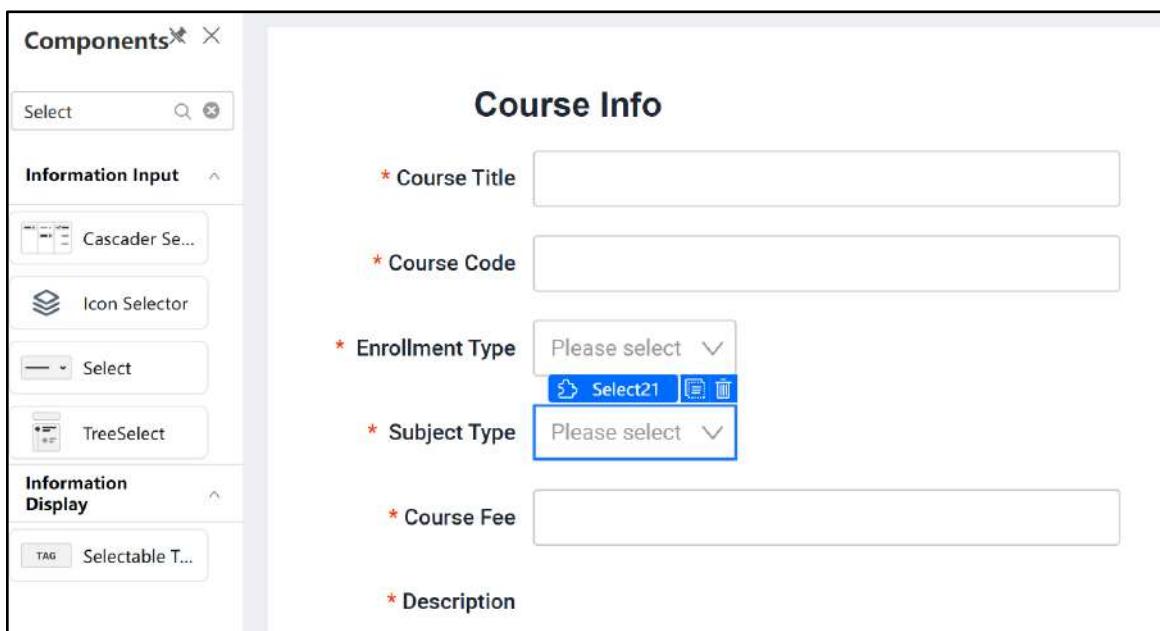
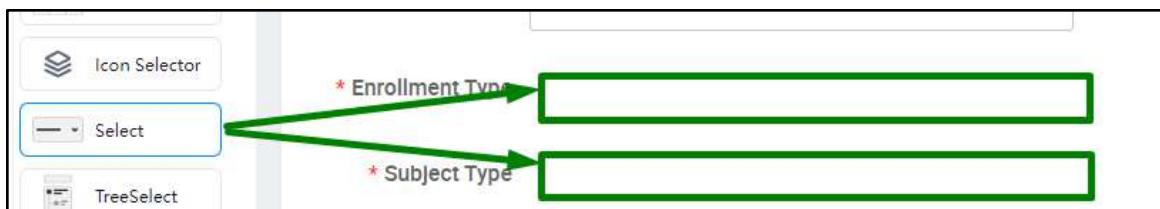
The expected output will be:

The screenshot shows a software interface with six input fields. The first two fields have red asterisks (*). The last four fields have blue asterisks (*). There are no 'Input208' buttons or trash can icons next to the fields. The fields are labeled: 'Course Title', 'Course Code', 'Enrollment Type', 'Subject Type', 'Course Fee', and 'Description'. The background is light gray, and the input fields have a white background with a thin blue border.

- In the Component Library, search the **Select** component



- Drag the Select Component to the now-empty **Enrollment Type** and **Subject Type** Form Item.



- **Ctrl + Click** both Select components and In the Styles tab, change their **width** to **100%**.

The screenshot shows a 'Course Info' form with several input fields. On the right, the 'Styles' tab of the developer tools is open for a 'Select' component. The 'ClassName' field contains 'Please enter'. The 'Style' panel shows a CSS rule: '#main f { width: 100%; }'. The 'Layout' panel shows a grid layout with a width of 100%.

- **Click on the Enrollment Type Select Component.** Under the Props tab, change the 3 Labels under **Datasource** to:
 - Part-time
 - Full-time
 - Online/Remote

The screenshot shows the 'Props' tab for a 'Select' component. The 'Datasource' section is highlighted with a red box. It contains three options labeled 'Option 1', 'Option 2', and 'Option 3'.

- Click on the **Subject Type Select Component**. Under the Props tab, change the Labels under **Datasource** to:

- Accounting
- Business
- Computing
- Design
- Education

The screenshot shows a UI component on the left and its corresponding Datasource panel on the right.

UI Component (Left):

- * Subject Type: A dropdown menu labeled "Select31".
- * Course Fee: An input field.
- Description: A text area.

Datasource Panel (Right):

Label
Accounting
Business
Computing
Design
Education

- Click the pencil icon beside **Design** and **Education**. Fill in the values for **Design** and **Education** to 4 and 5 respectively.

The Datasource panel displays five items with edit icons. The "Design" and "Education" items have their edit icons highlighted with a red box.

Label
Accounting
Business
Computing
Design
Education

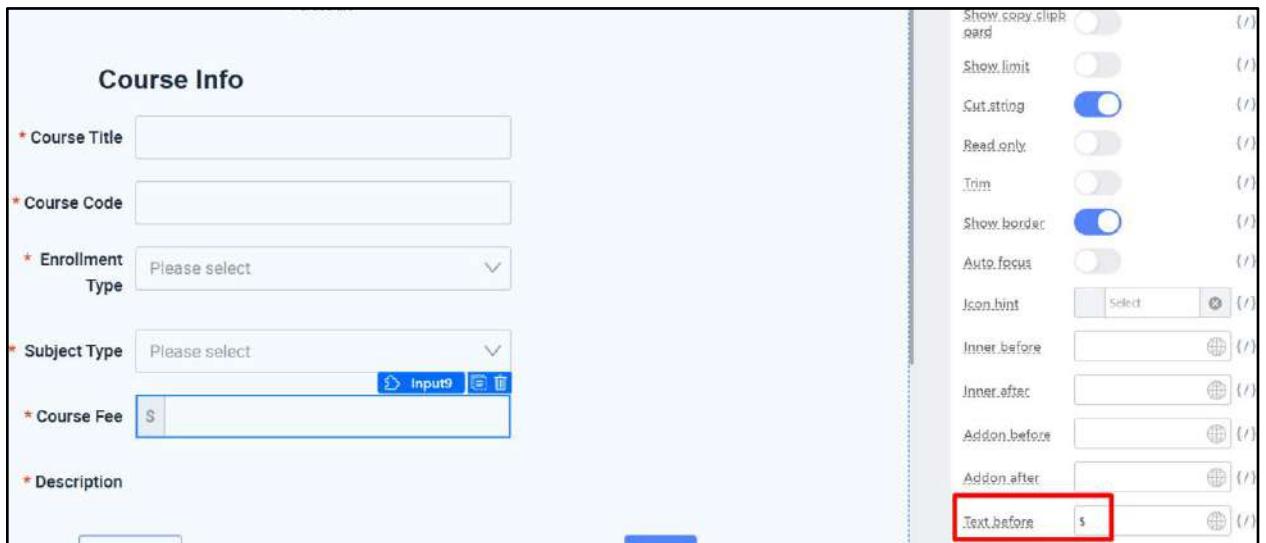
The dialog shows an item labeled "Item 3". It has two fields: "Label" (Design) and "Value" (4). The "Value" field is highlighted with a red box.

The dialog shows an item labeled "Item 4". It has two fields: "Label" (Education) and "Value" (5). The "Value" field is highlighted with a red box.

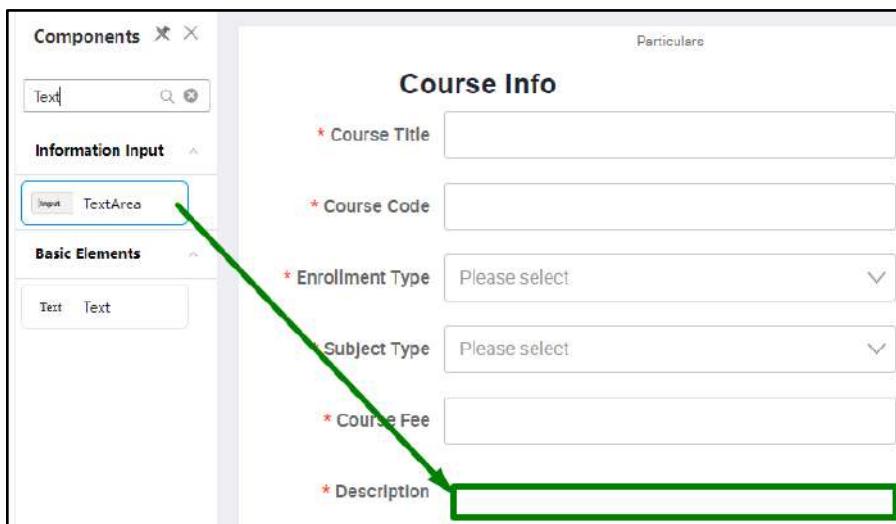
- Click on the Input Component in Course Fee



- Add a \$ sign in Text before field



- From the Component Library, drag a TextArea Component to the now-empty Description Form Item.



4.2.4 Course Info Form Validation

- Select the **Course Code** Form Item. Under Props > Validation, click on **Length Validation**.

The screenshot shows a form builder interface with four steps: Step 1 (Course Info), Step 2 (Instructor Particulars), Step 3 (Class Location), and Step 4 (Class Schedule). The 'Course Info' step is currently active. On the right, the 'Props' panel is open, showing 'Validation' settings. The 'Length validation' option is selected, indicated by a red box.

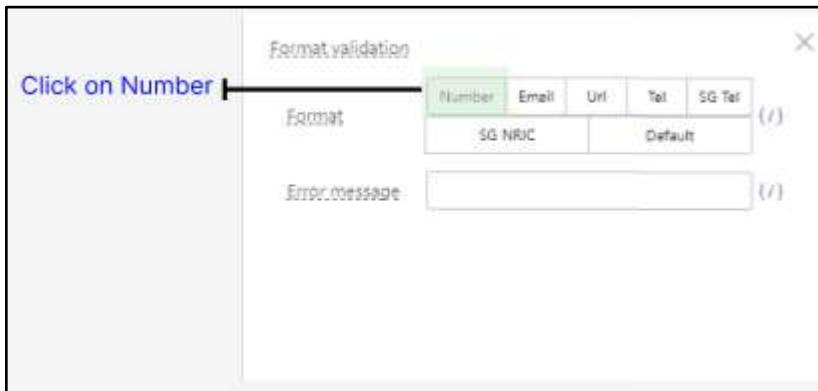
- Set the Length to 4.

The screenshot shows the same form builder interface after setting the length validation to 4. A tooltip 'Set Length to 4' points to the 'Length' field in the validation panel. The 'Course Code' input field now has a green border, indicating it is valid.

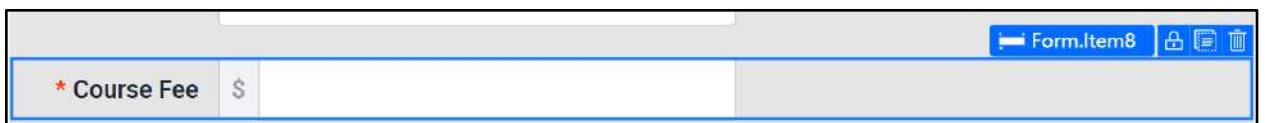
- Under Props > Validation, click on **Format Validation**.

The screenshot shows the validation panel with the 'Format validation' option highlighted, indicated by a red box. A tooltip 'Click on Format Validation' points to this option.

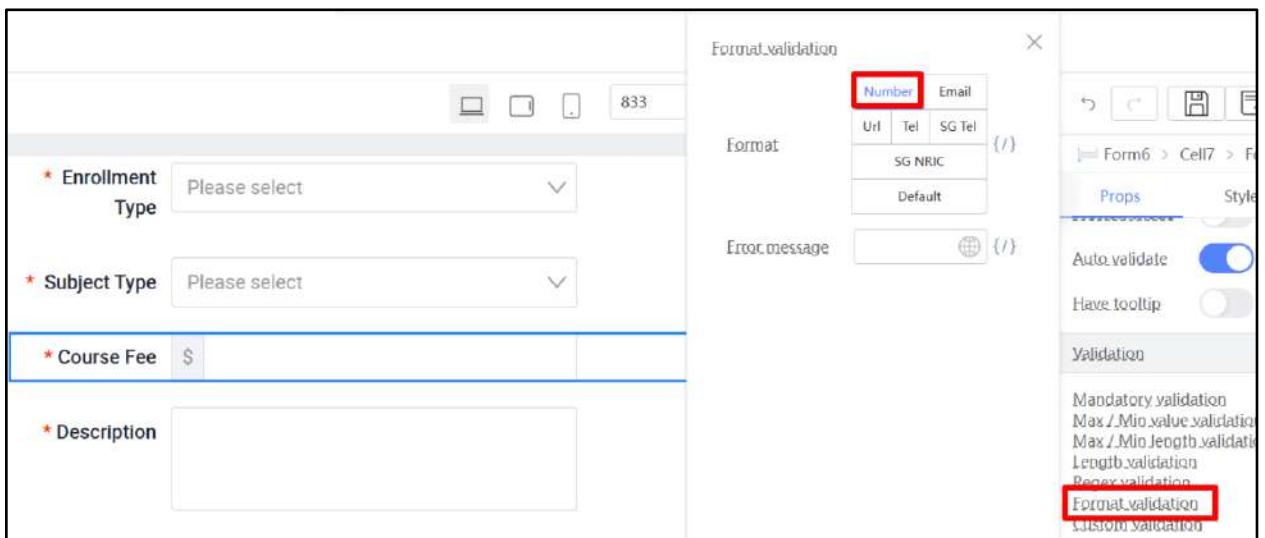
- Click on the Number Button (This ensures Course Code can only of type number)



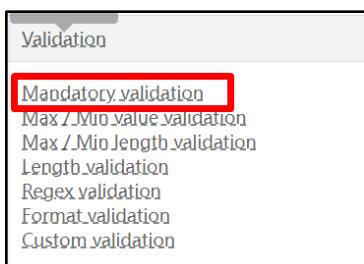
- Select the Course Fee Form Item



- Set the Course Fee Form Item Validation. Format Validation: Number



- Under Mandatory Validation



- Turn off 'Description' Form Item Mandatory Validation.



4.2.5 Form Completion

Once you are done with everything, the first form should look like this:

Course Info

* Course Title

* Course Code

* Enrollment Type

* Subject Type

* Course Fee

Description

[Previous](#) [Next](#)

Practical 4.3: Create the 'Instructor Particulars' Form

4.3.1 Context

The **2nd Step: Instructor Particulars**, is a form for filling in the Instructor Personal Particulars for the system to record and verify.

Before we start: this Practical will only cover the 'Instructor Particulars' Form. Only work within this Form:

The image shows a sequential form titled 'Instructor Particulars' divided into four steps:

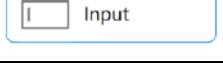
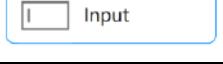
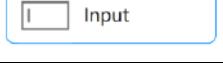
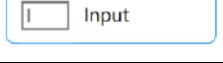
- Step 1 Course Info:** Contains a single input field for 'Email' with a red asterisk. A 'Previous' button is to the left and a 'Next' button is to the right.
- Step 2 Instructor Particulars:** Contains a single input field for 'Email' with a red asterisk. A 'Previous' button is to the left and a 'Next' button is to the right. This is the active step.
- Step 3 Class Location:** Contains a single input field for 'Email' with a red asterisk. A 'Previous' button is to the left and a 'Next' button is to the right.
- Step 4 Class Schedule:** Contains a single input field for 'Email' with a red asterisk. A 'Previous' button is to the left and a 'Next' button is to the right. A 'Finish' button is located at the bottom right of this step.

Each step has a header with a circular progress indicator (1, 2, 3, 4) and a sub-header describing the step. The entire form is contained within a large rectangular frame.

4.3.2 Objective

- Draw the page in the above figure using the concepts that you learnt in: **Form Fields and Form Validation**.

4.3.3 Requirements

Label Name	Name Property	Form Field Type	Validation	Remarks
Salutation	salutation	Select 	Mandatory	Options: Mr, Mrs, Ms, Dr
Name	name	Input 	Mandatory	
NRIC	nric	Input 	Mandatory, SG NRIC Format	
Email	email	Input 	Mandatory, Email Format	
Contact No.	contactNo	Input 	Mandatory, SG Tel Format	

4.3.4 Hints

- For this exercise, there is no need to do complex regex validations. Click and look through all the Validations - you might find what you are looking for inside.
- Do not forget your Property Names in the Form.Items!

4.3.5 Expected Result

Instructor Particulars

* Salutation

* Name

* NRIC

* Email:

* Contact No.

[Previous](#) [Next](#)

Practical 4.4: 'Class Location' Form

4.4.1 Context

The **3rd Step: Class Location**, is a form for filling in the Address where the classes for the course will take place.

Before we start: this Practical will only cover the 'Class Location' Form. Only work within this Form:

The image shows a sequential form with four steps:

- Step 1 Course Info:** Contains an email input field and 'Previous' and 'Next' buttons.
- Step 2 Instructor Particulars:** Contains an email input field and 'Previous' and 'Next' buttons.
- Step 3 Class Location:** Contains an email input field and 'Previous' and 'Next' buttons. This step is highlighted with a blue border.
- Step 4 Class Schedule:** Contains an email input field and 'Previous' and 'Next' buttons.

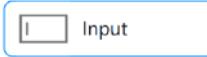
Each step also has a small circular icon with a number (1, 2, 3, 4) above it. A 'Form' button is located at the bottom right of the third step's content area.

4.4.2 Objective

Learn about:

- Creating 2 fields in one Form Item
- Validation of both fields individually
- Using Regular Expression Validation
- Conditional Form Fields

4.4.3 Requirements

Label Name	Name Property	Form Field Type	Validation	Remarks
Postal Code	postalCode	Input 	Mandatory, Length validation: 6 digits Numbers only	
Address	address	Input 	Mandatory	
Floor / Unit No.		Input 		Refer to 4.4.4

Note that for the **highlighted part in yellow** in the table, you should just have the Floor / Unit No. Form field ready. You will be taught how to create the 2 inputs in the next section.

For now, with the info above, you should have something like this:

Class Location

* Postal Code

* Address

* Floor / Unit No.

Previous
Next

4.4.4 Creating a Field with 2 inputs

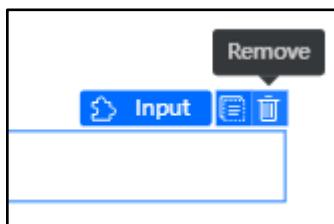
Sometimes you will need to create a form field with 2 fields. The example we used in this scenario is a floor/unit no (eg #02-16) field.

Here is the revised table with the Floor and Unit No. separate.

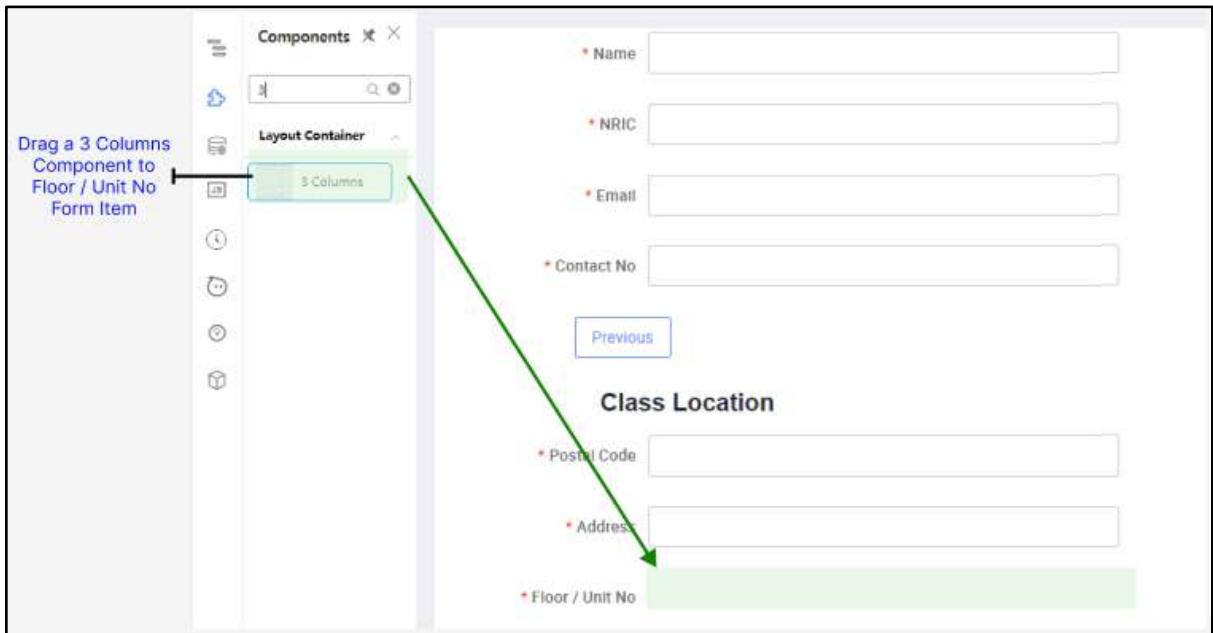
Label Name	Name Property	Form Field Type	Validation	Remarks
Postal Code	postalCode	Input	Mandatory, Length validation: 6 digits Numbers only	
Address	address	Input	Mandatory	
Floor / Unit No.	floorNo	Input	Mandatory. Must be number. Max length 2.	Must be in the same row or form item
	unitNo	Input	Mandatory. 2-3 numbers at the start and an optional uppercase letter at the end.	

It is possible to handle this in the Designer by using the 2 Columns or 3 Columns Component.

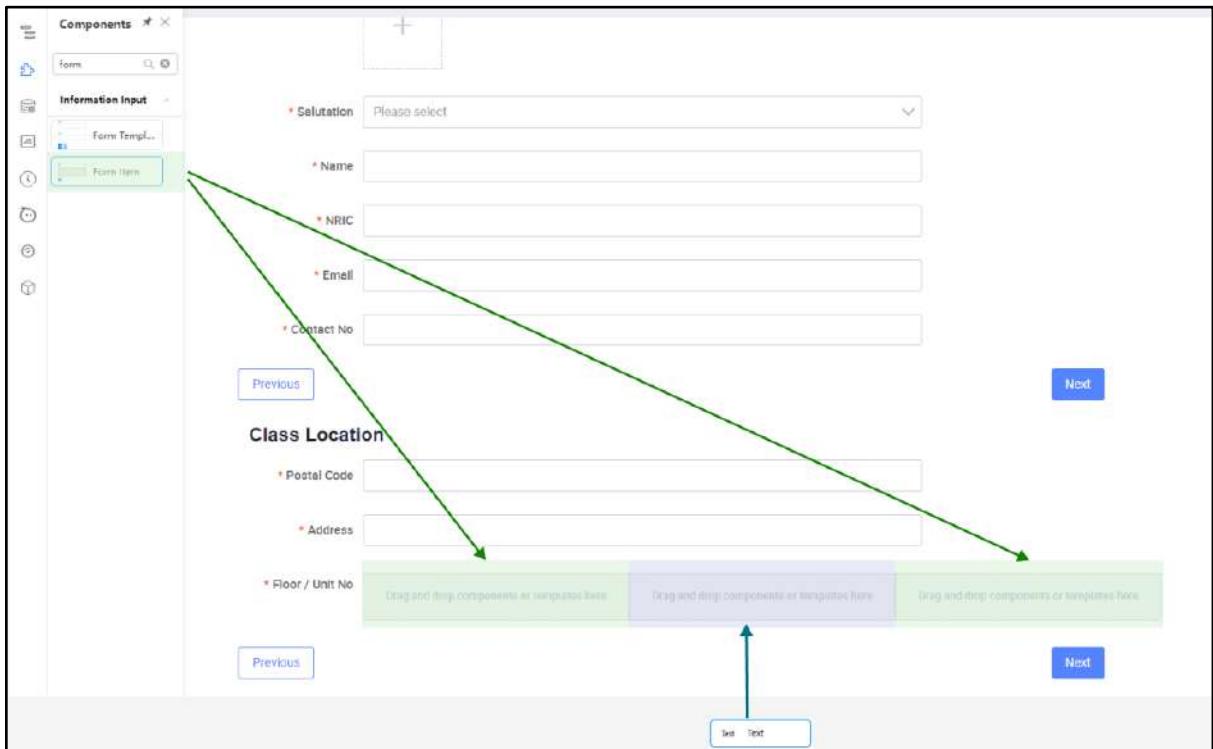
- Remove the Input component within the Floor / Unit No. Form Item.



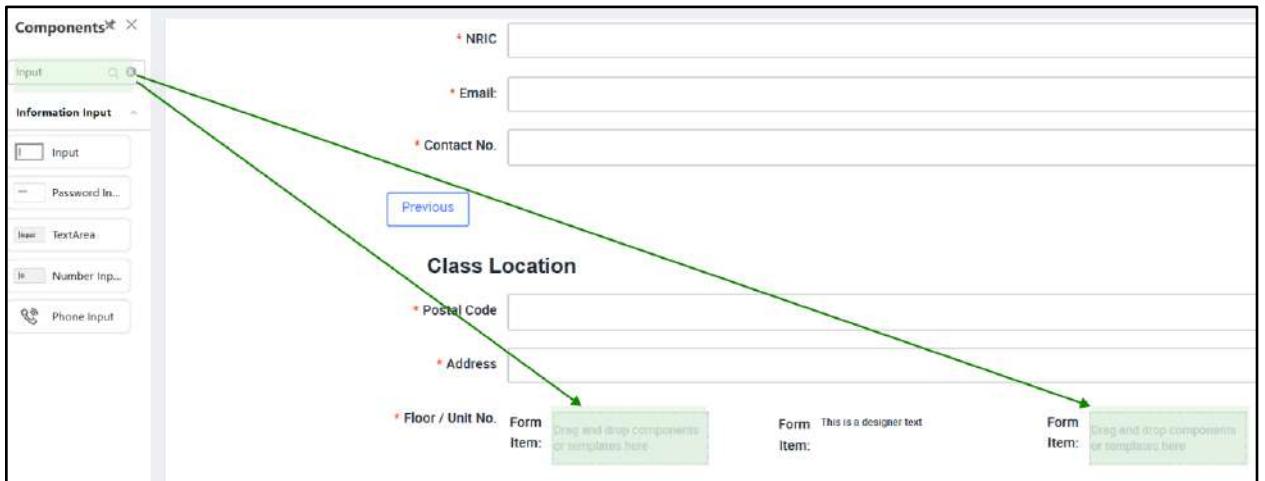
- **Drag a 3 Columns Component to the Floor / Unit No Form Item.**



- **Drag a Form Item Component each for the left and right columns. Drag a Text Component into the Center column.**



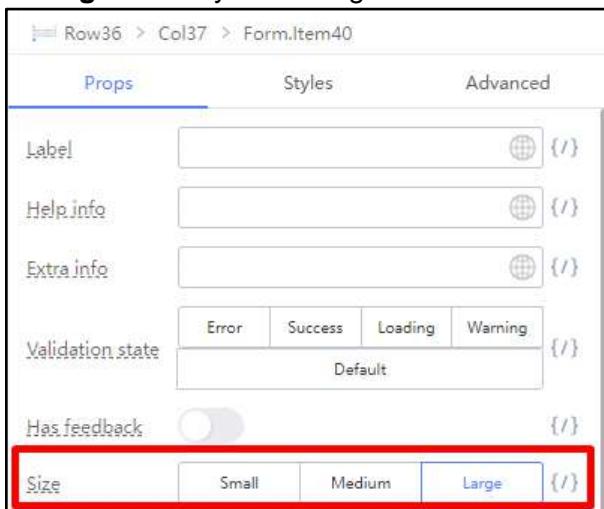
- Drag **Input Components** into the left and right Form Items.



- **Ctrl + Click** to select both Form Item Components.



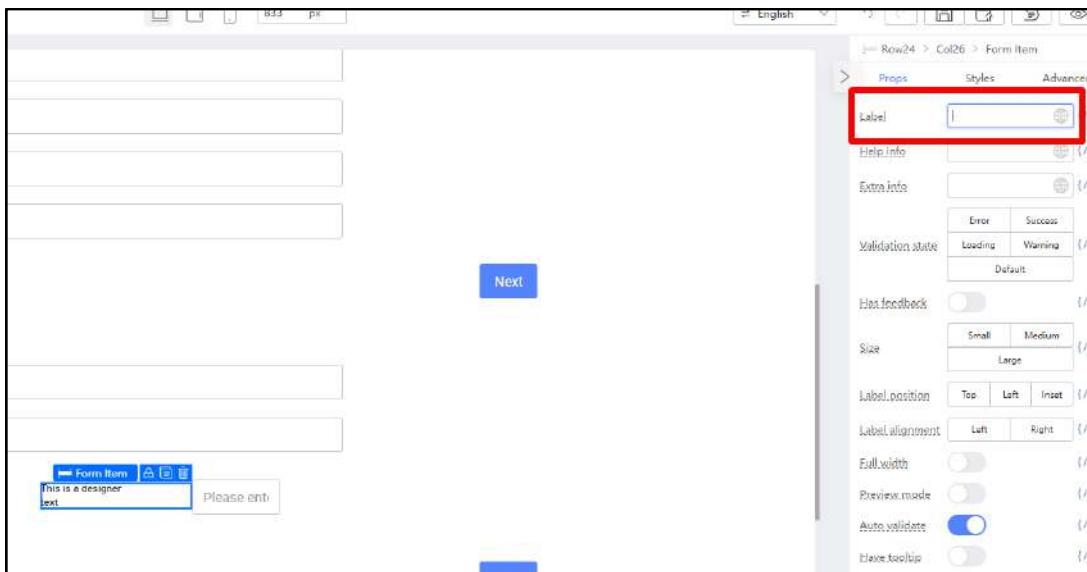
- **Change** their Styles to Large.



- Select the left Form Item. Clear its Label and change its Property Name to **floorNo**
- Select the right Form Item. Clear its Label and change its Property Name to **unitNo**

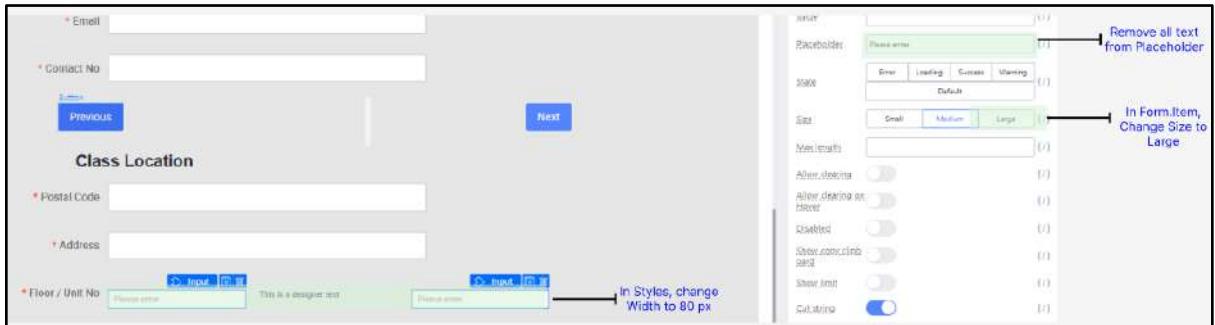
floorNo	unitNo
<p>Row36 > Col37 > Form.Item40</p> <p>Props Styles Advanced</p> <p>Label <input type="text"/> {/}</p> <p>Help.info <input type="text"/> {/}</p>	<p>Row1 > Col4 > Form.Item7</p> <p>Props Styles Advanced</p> <p>Label <input type="text"/> {/}</p> <p>Help.info <input type="text"/> {/}</p>
<p>Advanced</p> <p>ID <input type="text"/> {/}</p> <p>Name <input type="text" value="floorNo"/> {/}</p>	<p>Advanced</p> <p>ID <input type="text"/> {/}</p> <p>Name <input type="text" value="unitNo"/> {/}</p>

- Delete the **Label** of the center Form Item component

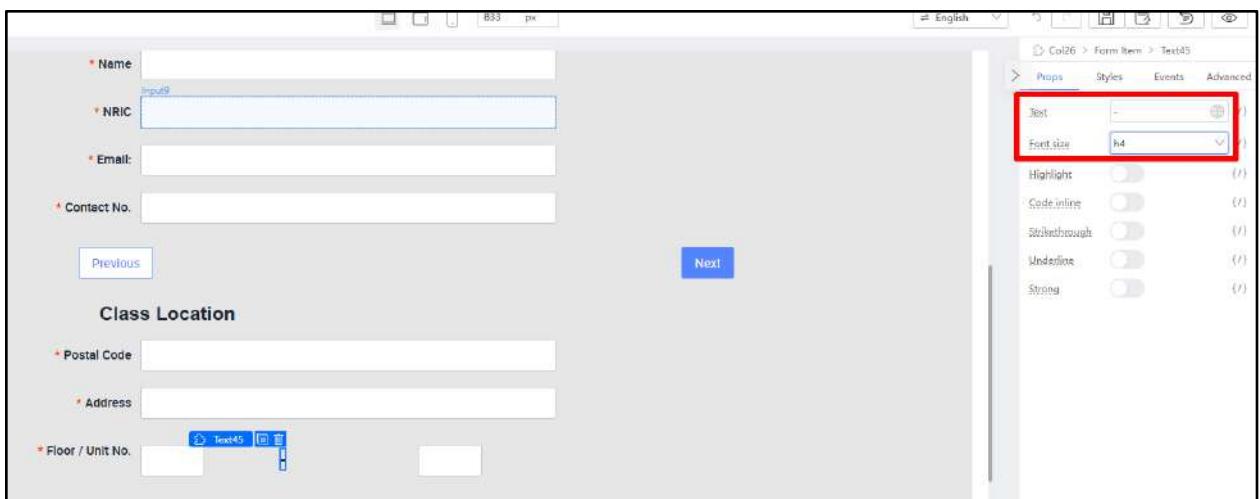


4.4.5 Styling for the Inputs

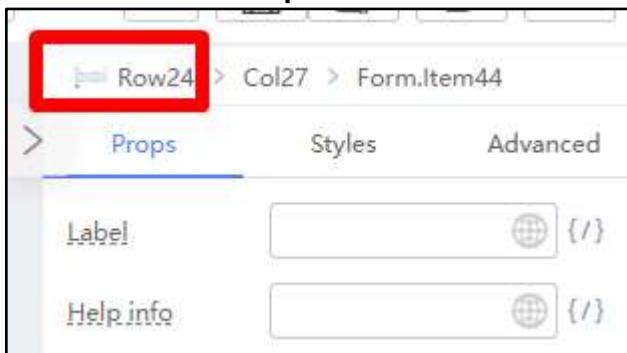
- **Ctrl + Select** both Input Components. **Delete** their **Placeholders**. In Styles, change their **width** to **80px**.



- **Change** the center **Text** Component to be a Dash (ie. '-'). **Change** **Font Size** to **h4**.



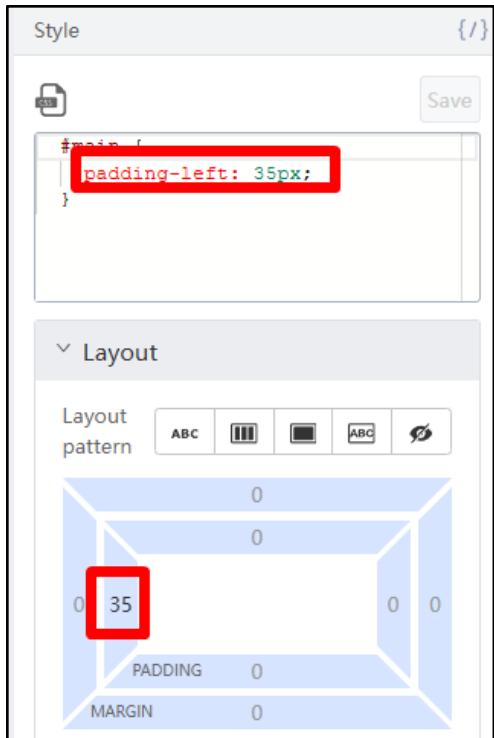
- **Select** the **Row Component** inside the **Floor / Unit No** **Form.Item**.



- Change the width of the row to 200px.

- Select the Center Column Component.

- Add a left padding of about 35px. The dash should be roughly aligned with the input boxes like this:



* Floor / Unit No -

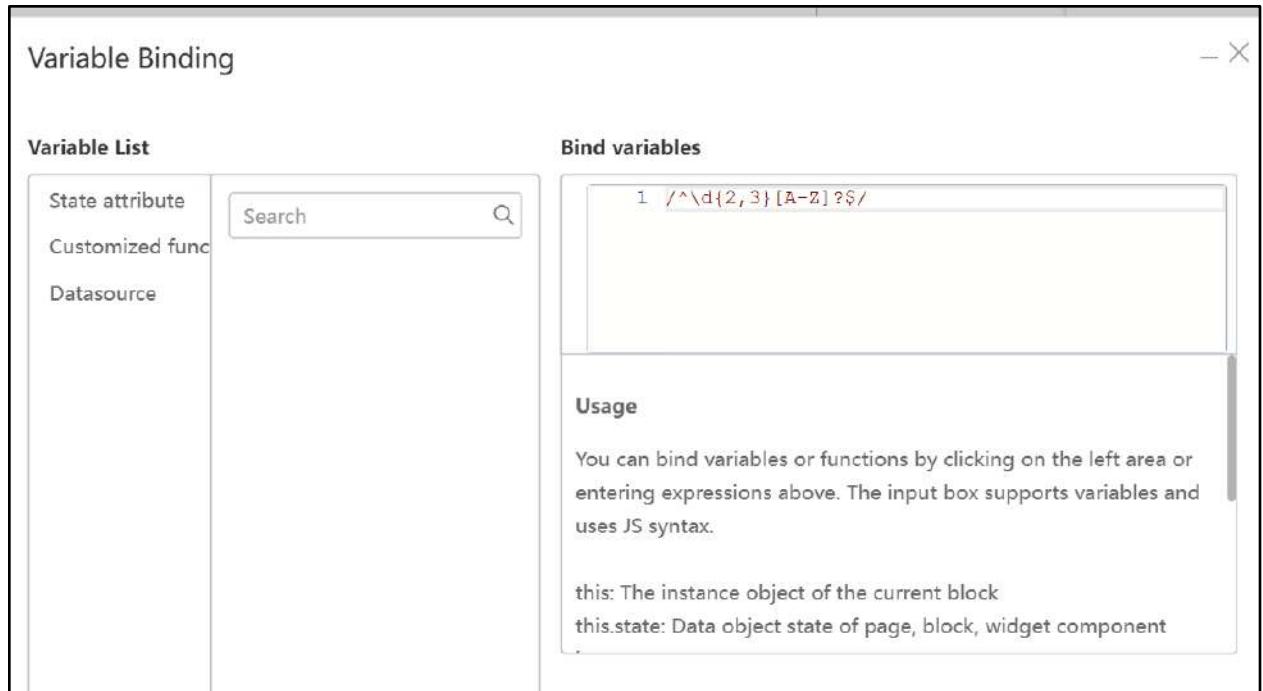
4.4.6 Validation for both input fields

- Select the left Form Item and **add Mandatory and Number Validation**.



- Select the right Form Item and **add Mandatory and Regex Validation** with this value:
`/^\d{2,3} [A-Z] ?$/`





- Add an Error message for this Regex Validation with this value:
`unitNo is 2-3 digits + optional A-Z`

Go into preview mode and click Next to see that both fields are now being validated correctly.

4.4.8 Form Completion

The final form should look something like this:

Class Location

* Postal Code

* Address

* Floor / Unit No. -

[Previous](#) [Next](#)

Practical 4.5: Class Schedule Form

4.5.1 Context

The **4th Step: Class Schedule**, is a form for filling in the start and end date which the classes will take place and also the schedule every week for this date period.

Before we start: this Practical will only cover the 'Class Schedule' Form. Only work within this Form:

The image shows a step-by-step form titled "Class Schedule". It consists of four sections: "Course Info", "Instructor Particulars", "Class Location", and "Class Schedule". Each section has an "Email:" field with a red asterisk indicating it is required. Navigation buttons "Previous" and "Next" are located between sections. A "Form" icon is in the top right corner. A progress bar at the bottom shows Step 1 is active.

4.5.2 Objective

Learn about:

- Custom Validation for the DatePicker to ensure Start Date is always after End Date
- Doing a simple dynamic row adder

4.5.3 Requirements

Label Name	Name Property	Form Field Type	Validation	Remarks
Start Date	startDate	DatePicker 	Mandatory. This value must be before End Date.	
End Date	endDate	DatePicker 	Mandatory. This value must be after Start Date.	
Weekly Schedule		Table 	Non-Mandatory	Refer to 4.5.5 for more info on how to do.

For the Weekly Schedule:

- Each schedule contains the **Day** of the week, **Start Time** and **End Time**.
- The user must be able to create new rows representing a new schedule.

Class Schedule

* Start Date 

* End Date 

Weekly Schedule	Day	Start Time	End Time
	<input type="text" value="Please select"/> 	<input type="text" value="Please select time"/> 	<input type="text" value="Please select time"/> 

+ Add Schedule

4.5.4 Custom Date Validation

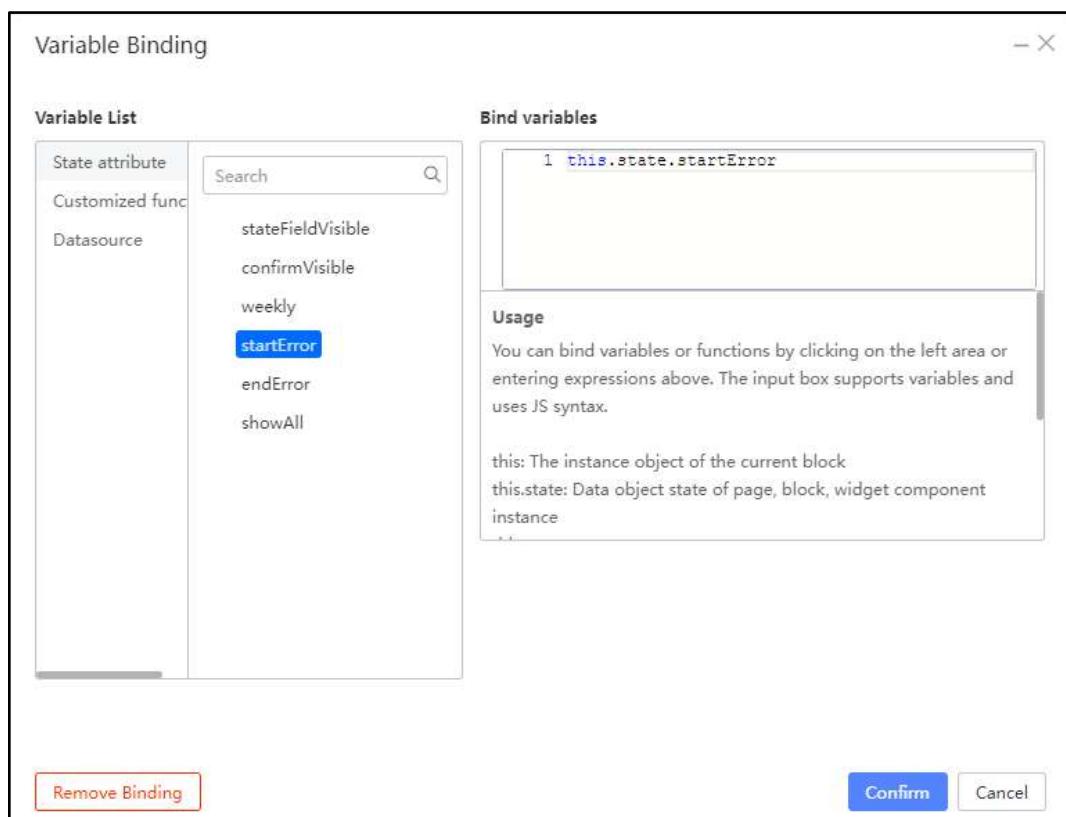
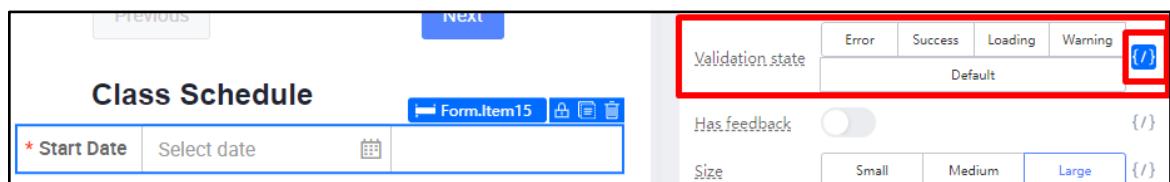
- Open the Source Code Panel and uncomment lines 7 - 8.

```
7 | // "startError": '',
8 | // "endError": '',
9 | // "rule": ''
```

- Uncomment the section under (lines 39 - 57). Then click the Save button.

```
36 | // // Function to check if the Start Date is before the End Date.
37 | // // Sets error if the Start Date is after the End Date.
38 | // checkDates(rule, value, callback) {
39 | //   if (this.formField.getValue("startDate") !== undefined && this.form
40 | //     if (moment(this.formField.getValue("startDate"), 'YYYY-MM-DD').is
41 | //       this.setState({
42 | //         startError: 'error',
43 | //         endError: 'error'
44 | //       })
45 | //       callback("Start Date cannot be after End Date!");
46 | //     }
47 | //     else {
48 | //       this.setState({
49 | //         startError: 'default',
50 | //         endError: 'default'
51 | //       })
52 | //       callback();
53 | //     }
54 | //   }
55 | //   return;
56 | // }
```

- Select the Start Date Form Item. Variable Bind the validation state to this.state.startError



- Select the End Date Form Item. Variable Bind the validation state to this.state.endError



The screenshot shows the 'Class Schedule' form editor. On the right, the validation state configuration for the 'End Date' field is displayed. A red box highlights the 'Validation state' section, which includes tabs for Error, Success, Loading, and Warning, with 'Default' selected. Below this are sections for 'Has feedback', 'Size' (set to Large), and 'Label position' (set to Inset).

Variable Binding

Variable List

- State attribute
- Customized func
- Datasource

Bind variables

```
1 this.state.endError
```

Usage

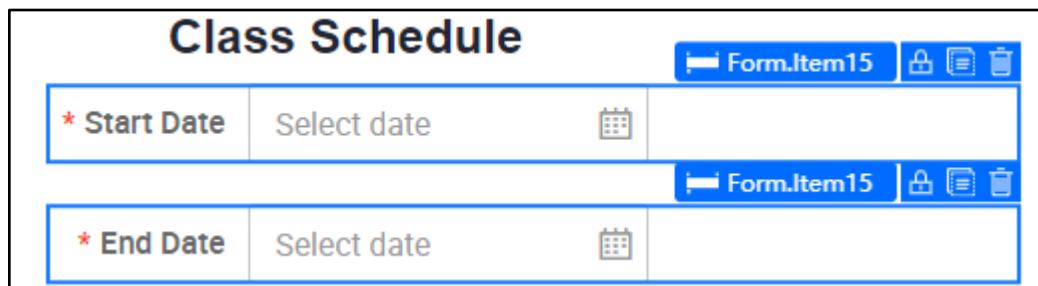
You can bind variables or functions by clicking on the left area or entering expressions above. The input box supports variables and uses JS syntax.

this: The instance object of the current block
this.state: Data object state of page, block, widget component instance

Buttons

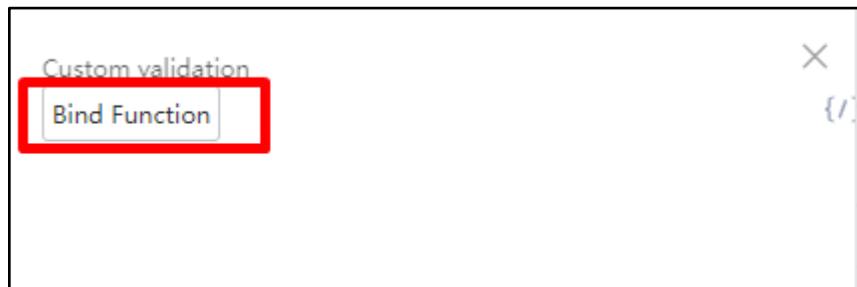
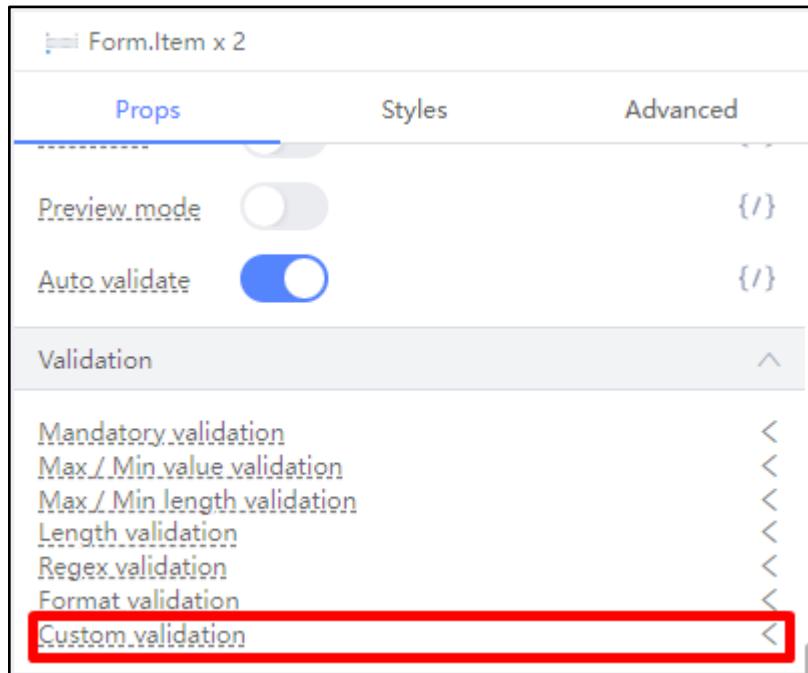
- Remove Binding
- Confirm
- Cancel

- Ctrl + Click to select both the Form Items of Start Date and End Date.

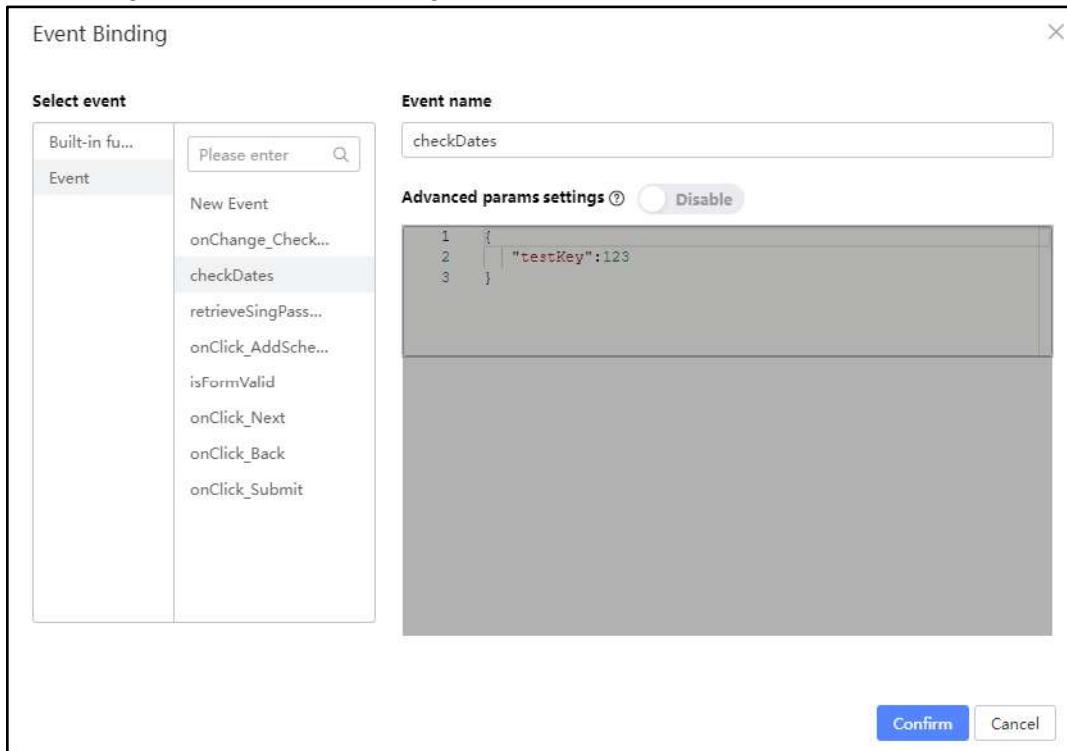


The screenshot shows the 'Class Schedule' form editor with both the 'Start Date' and 'End Date' fields selected. Both fields have a blue border and are labeled with 'Form.Item15'. The 'Start Date' field has a red asterisk indicating it is required.

- Go to the Props tab, and click on **Custom validation**. Then, click on the **Bind Function** button.



- Under Select event, click on Event, then click on **checkDates**. You should get something like this before clicking on the Confirm button:



- Go to Preview and check if the validation is working as intended by selecting the dates for both datepickers and deliberately choosing invalid values.

Class Schedule

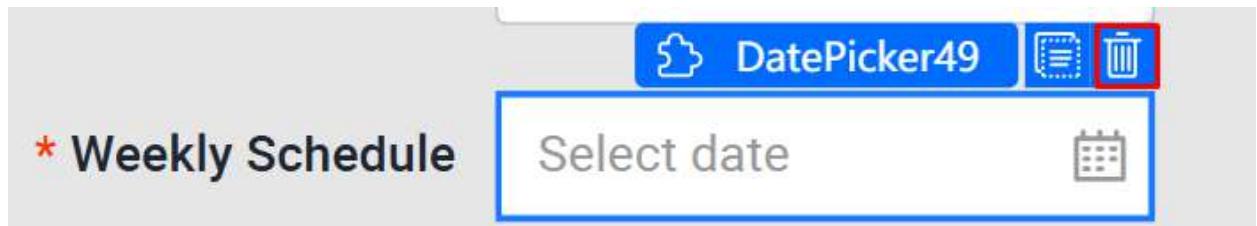
* Start Date	2024-03-06	
* End Date	2024-03-27	

Class Schedule

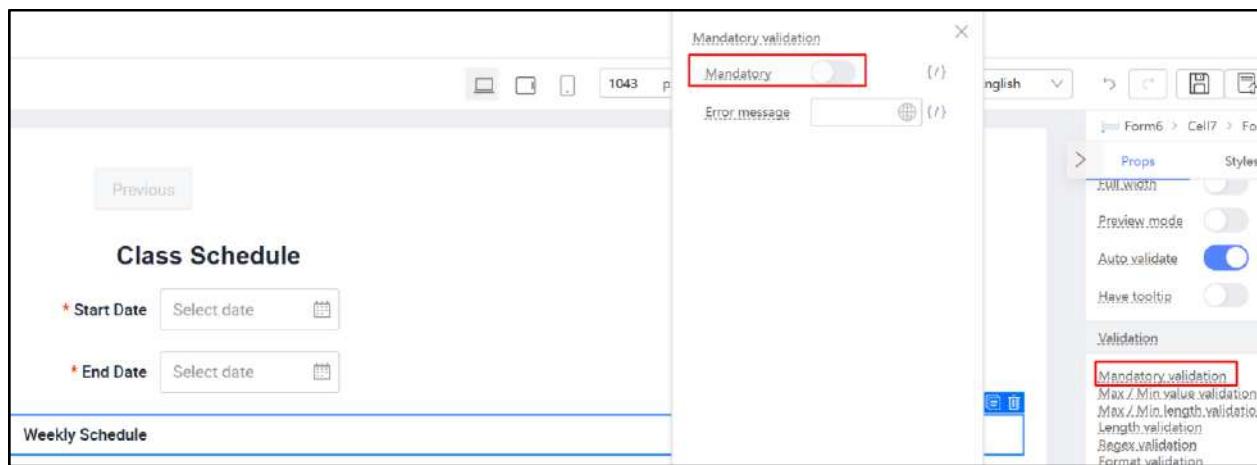
* Start Date	2024-03-13	
Start Date cannot be after End Date!		
* End Date	2024-03-06	

4.5.5 Creating the Form field that can add new Rows

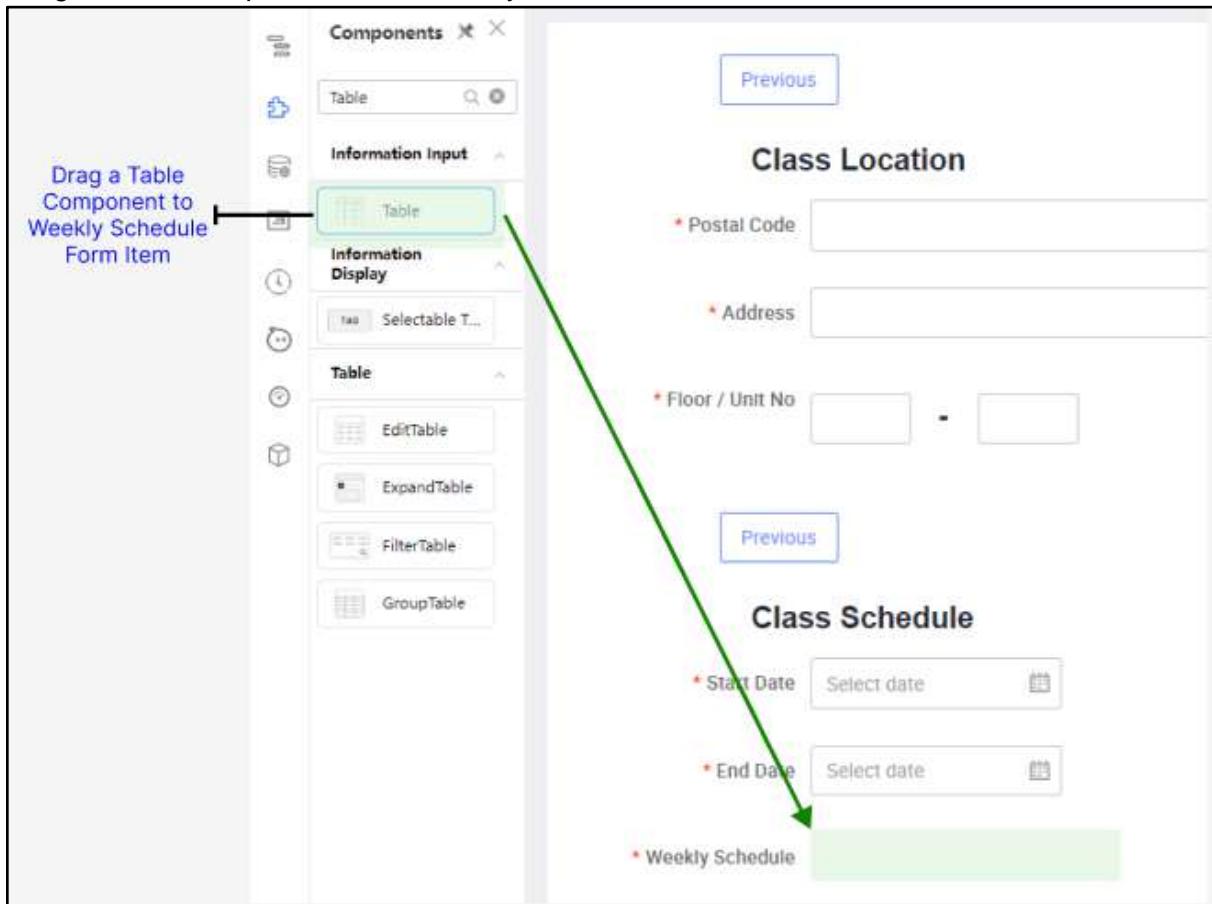
- Delete the **DatePicker** component under **Weekly Schedule** field



- Remove the Mandatory Validation



- Drag a Table Component to the Weekly Schedule Form Item.



- Select the Table Component. Under the Props tab,
 - Turn off pagination.
 - Remove the 'Add' and 'Edit' entries in the Action Bar.
 - Remove the 'Edit' and 'Preserve' entries in the Action Column.

The screenshot shows the 'Class Schedule' form item with its properties panel open. The form item displays a table with columns: Name, Age, Responsibility, and Operation. The table contains three rows with data: Wang Xiao (Age 15000, Responsibility develop), Wang Zhong (Age 25000, Responsibility product), and Wang Da (Age 35000, Responsibility design). The properties panel on the right is divided into several sections:

- Pagination configuration:** Shows a toggle switch for 'Show pagination' and a slider for 'Turn off pagination'.
- Action bar:** Shows a 'Title' section with 'Add' and 'Edit' buttons, with a note 'Remove Add and Edit' next to it.
- Action column:** Shows a 'Title' section with 'Edit' and 'Preserve' buttons, with a note 'Remove Edit and Preserve' next to it.

 The 'Action bar' and 'Action column' sections have 'Add an item +' buttons.

- Add another Cell below the Weekly Schedule Table Form Item.

The image displays two screenshots of a form builder interface, likely from a tool like AEM Forms or a similar platform. Both screenshots show a form titled "Class Schedule" with two date input fields ("Start Date" and "End Date") and a "Weekly Schedule" table.

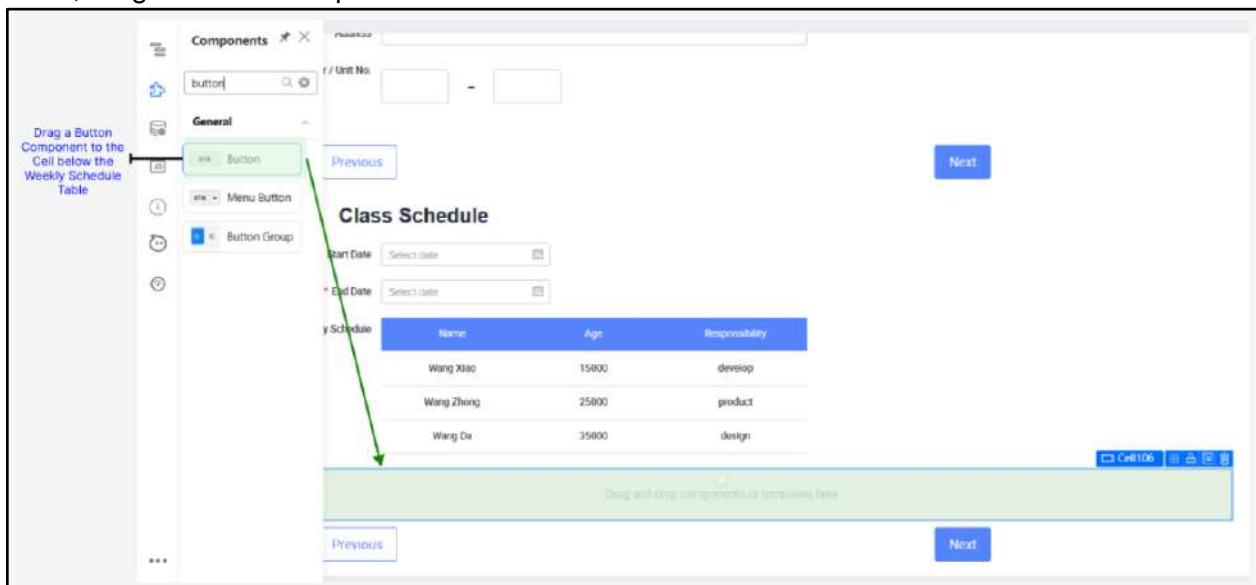
Top Screenshot: This screenshot shows the "Components" panel on the left, which includes a "Cell" component under "Layout Container". The main area displays the "Class Schedule" form. Below the table, there is a horizontal bar with a blue segment labeled "FDCell", indicating where a new component was placed.

	Name	Age	Responsibility
Wang Xiao	15000	develop	
Wang Zhong	25000	product	
Wang Da	35000	design	

Bottom Screenshot: This screenshot shows the same "Class Schedule" form, but the "Components" panel is not visible. The "Weekly Schedule" table is present, and directly below it is a large, empty rectangular area with a placeholder text "Drag and drop components or templates here". In the top right corner of this area, there is a toolbar with icons for "Cell53", "Align", "Size", and "Format".

	Name	Age	Responsibility
Wang Xiao	15000	develop	
Wang Zhong	25000	product	
Wang Da	35000	design	

- Then, drag a Button Component into it.



- In the Props Tab for the button's Form Item, change the following
 - Clear its Label**
 - Size: Large**
 - Wrapper Column offset to 4** so that it aligns with the table.
 - Change the button's text to **Add Schedule**.

The screenshot shows the 'Props Tab' for a button component. On the left, there are sections for 'Label', 'Size', and 'Wrapper column offset'. The 'Label' field contains the text 'Add Schedule'. The 'Size' dropdown is set to 'Large'. The 'Wrapper column offset' dropdown has an arrow pointing to the value 'Set to 4'. To the right, a preview of the UI is shown, featuring a table with three rows and a blue 'Add Schedule' button at the bottom. A callout box on the left of the preview area also points to the 'Add Schedule' button with the same instructions as the props tab.

- Add a + icon to the button

The screenshot shows a table component with three rows of data: Wang Xiao (Age: 15000, Responsibility: develop), Wang Zhong (Age: 25000, Responsibility: product), and Wang Da (Age: 35000, Responsibility: design). Below the table is a blue button labeled '+ Add Schedule'. To the right is a 'Props' panel with the following settings:

- Icon:** A red box highlights the 'icon-add' icon.
- Theme:** Ghost, Normal, Light, Dark (Dark is selected).
- Content:** Add Schedule.
- Button.type:** Primary, Secondary (Secondary is selected).
- Normal:**

- Select the Table Component again. Change the Data Columns to ‘Day’, ‘Start Time’ and ‘End Time’.

The screenshot shows the same table component as before. A note 'Change Columns to Day, Start Time and End Time' is placed over the 'Props' panel. The 'Props' panel includes:

- Data column:** Title, Name, Age, Responsibility.
- Data source:** edit data.
- Sort icon desc:** descending.
- Sort icon asc:** ascending.
- Action column:**

- Edit each column by clicking on the pen icon on the left of each label. Change the Data Keys of Day, Start Time and End Time to ‘day’, ‘start’ and ‘end’ respectively. Set all their Data Types to ‘None’.

- Repeat the above steps for **Start Time** and **End Time**

The screenshot shows the properties panel for a Table component named Table50. The left side lists various column properties, and the right side shows the configuration for the three columns: Day, Start Time, and End Time.

Properties (Left Side):

- Title: Day
- Align: Center
- Data key: day
- Width: 200 px
- Data type: None
- Lock: None
- Allow sorting: Off
- Cell: Bind Function
- Render: Enable (checked), Params: value,index,record

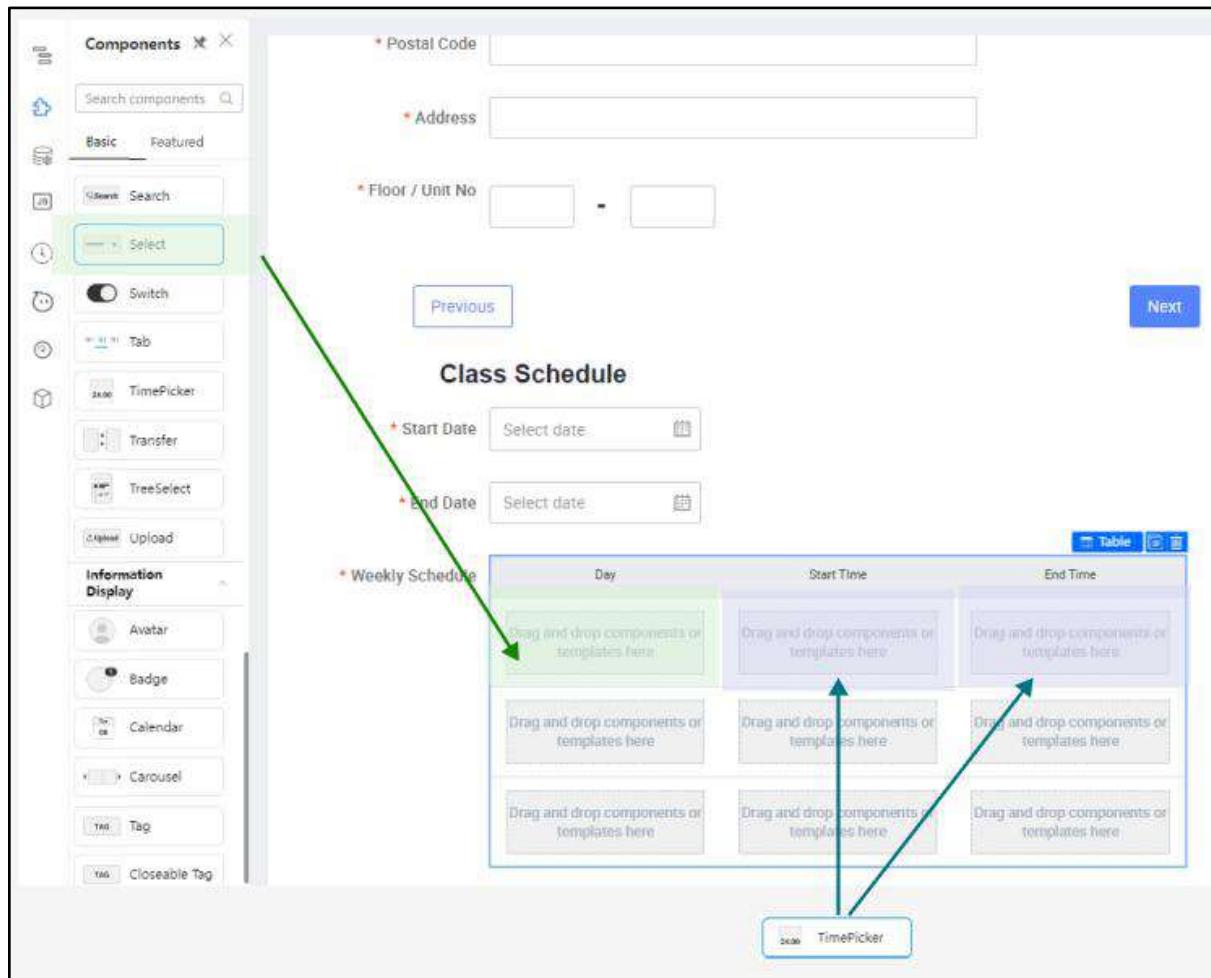
Columns (Right Side):

- Day:** Title: Day, Data key: day, Data type: None
- Start Time:** Title: Start Time, Data key: start, Data type: None
- End Time:** Title: End Time, Data key: end, Data type: None

Actions:

- Add an item +
- Data source: Edit data (checked)
- Row.Props: Bind Function
- Sort icons.desc: Descending
- Sort icons.asc: Ascending

- Drag a **Select** Component into the Date Column. Drag a **Timepicker** Component into the Start Time and End Time Columns.



- Select one of the Select Components in the Day Column and add the following Options
 - Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday' and 'Sunday'.



- Select the Timepickers in the **Start Time** and **End Time** columns. Under the Props tab,
 - Set the **Format** to **HH:mm**
 - In the Styles tab, set the **width** of the Timepickers to **150px**.

Select the Timepickers for both Columns

Props

Default value: Please select time

Time value: Please select time

Size: Small Medium Large

Format: HHmmss HHmm **HH:mm**

Hourly step: 1

Minute step: 1

Secondly step: 1

Allow clearing:

Disabled hours:

Bind function:

Disabled minutes:

Bind function:

Disabled seconds:

Bind function:

Advanced

Style

```
#main {
  width: 150px;
}
```

Save

Layout

Layout pattern: ABC

Width: 150 px

Height: 0 px

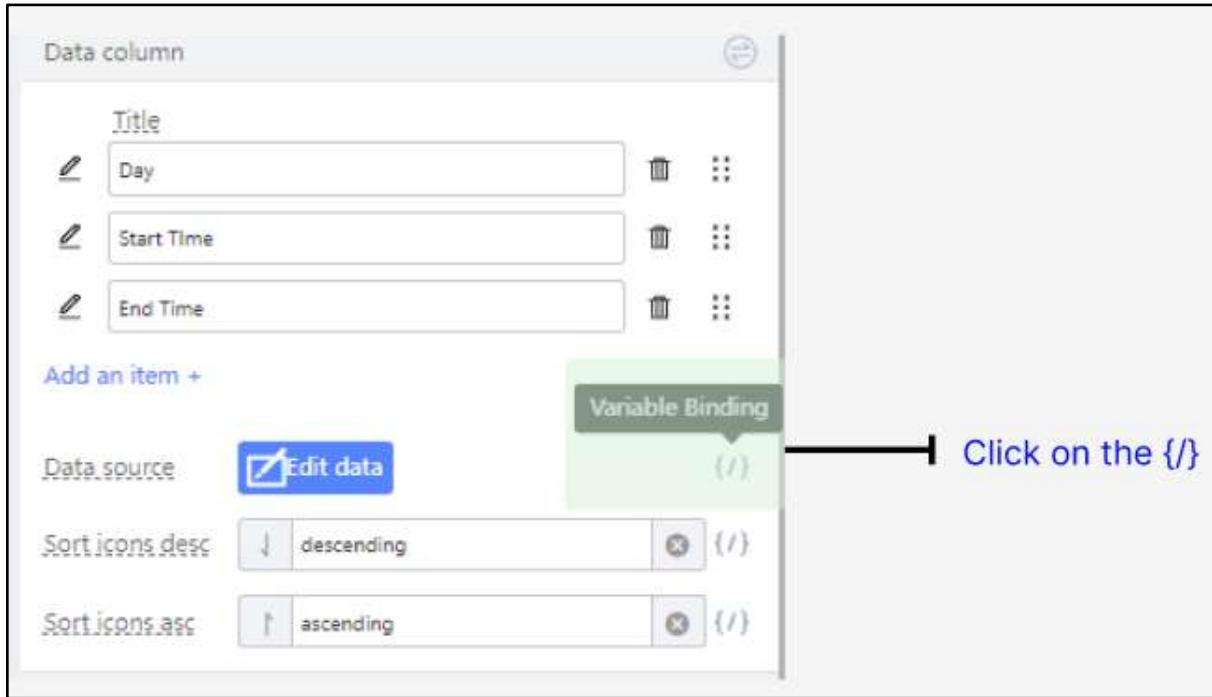
- Open the Source Code Panel and uncomment lines 9 - 13.

```
9  // "weekly": [{  
10 //   "day": undefined,  
11 //   "start": undefined,  
12 //   "end": undefined  
13 // }],
```

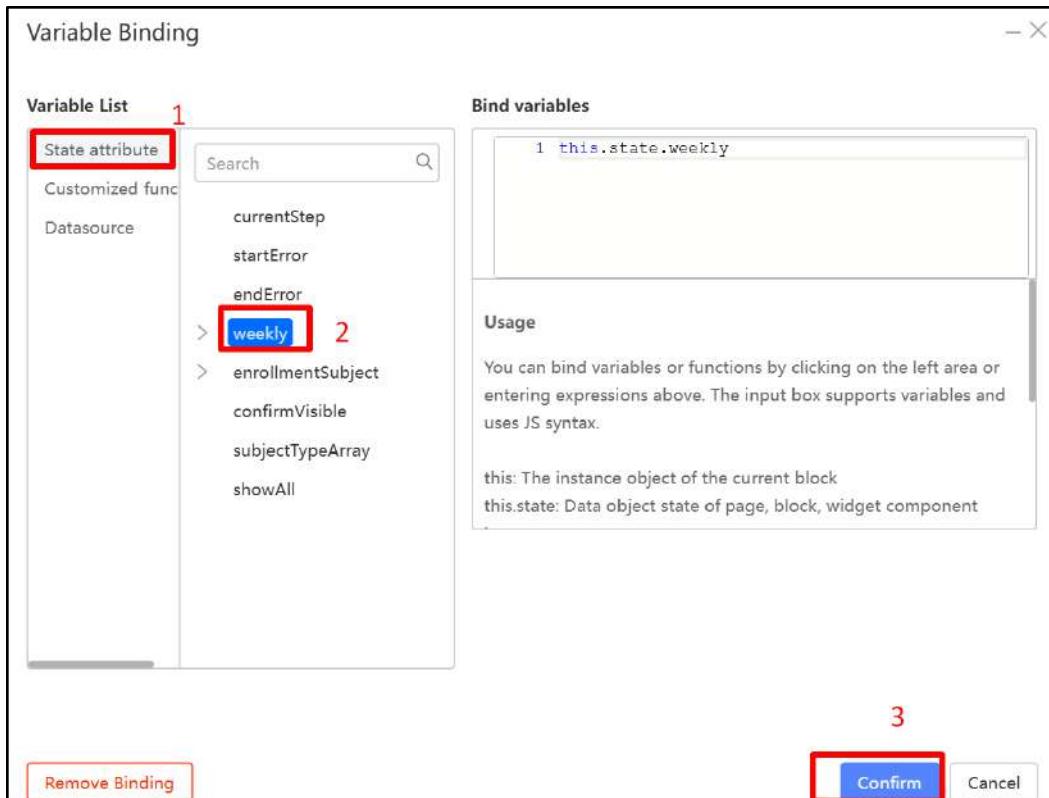
- Uncomment the section under `/** 4.5.5 */` (lines 61 - 69). Then click the Save button.

```
58  /** 4.5.5 */  
59  // // Function to add a new row every time the Add Schedule bu  
60  // onClick_AddSchedule() {  
61  //   this.setState({  
62  //     weekly: [...this.state.weekly, {  
63  //       "day": undefined,  
64  //       "start": undefined,  
65  //       "end": undefined  
66  //     }]  
67  //   })  
68  // }
```

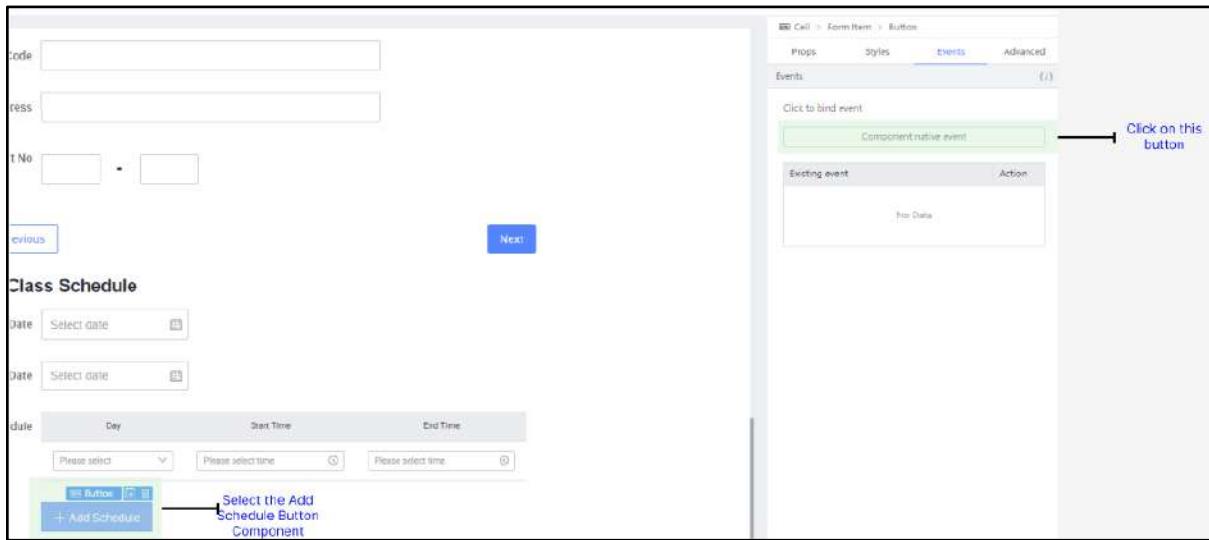
- Select the Weekly Schedule Table Component. Click on Variable Binding in Data Source.



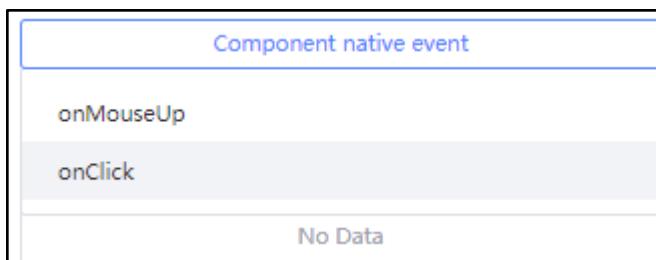
- Click on **State Attribute** under Variable List, then click on **weekly**. Then click the **Confirm** button.



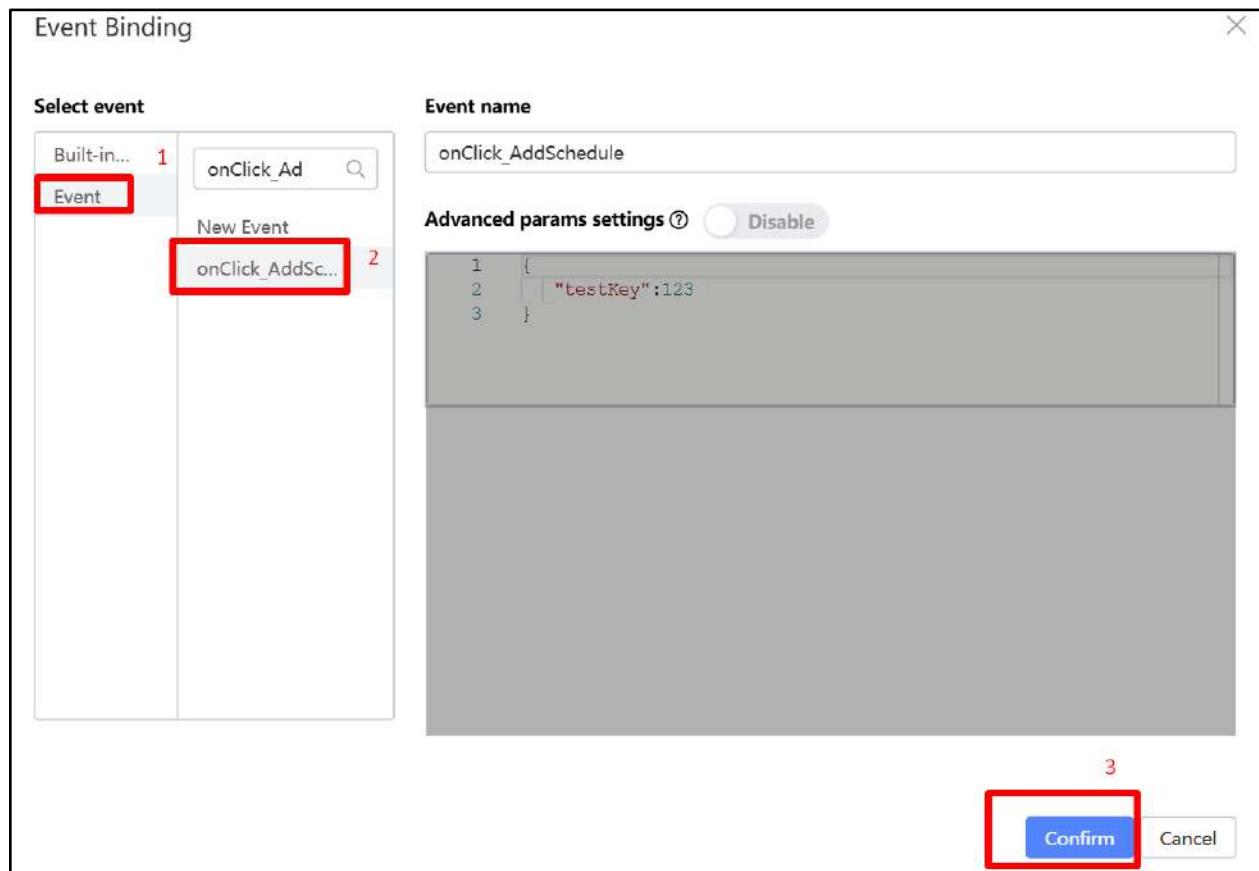
- **Click on the Add Schedule button.** Go to the **Events** tab, and click on the '**Component native event**' button.



- **Select the onClick event from the dropdown.**



- Under 'Select event', click on **Event**, then click on **onClick_AddSchedule**. Then Click the **Confirm** button.



You can now add extra schedules!

Weekly Schedule	Day	Start Time	End Time
	Monday	18:00	20:00
	Tuesday	19:00	21:00
	Saturday	10:00	12:00
+ Add Schedule			

4.5.6 Form Completion

Your finished Form 4 should look like this:

Class Schedule

* Start Date 

* End Date 

Weekly Schedule

Day	Start Time	End Time
Please select	Please select time 	Please select time 

+ Add Schedule

[Previous](#) [Next](#)

Practical 4.6: Cascade Dropdown

4.6.1 Context

Recall the 1st form, Course Info. There were 2 Select dropdown fields in that form:

- Enrollment Type - which is values like Part-time, Full-time and Online/Remote.
- Subject Type - The subject of the course (eg. Business, Computing etc.)

Now, let's say there was a change in requirements: the Part-time, Full-time and Online/Remote courses can only have certain subject types. What we will need to do is to implement some conditional logic to handle that.

4.6.2 Requirements

In the 1st Form, Course Info,

- The Select dropdown from Enrollment Type should populate a different set of values of Subject Type:

Enrollment Type	Subject Type
Part-time	Accounting, Business, Computing
Full-time	Accounting, Business, Computing, Design, Education
Online/Remote	Computing, Education

4.6.3 Adding the Logic

- Open the Source Code Panel and **uncomment lines 14 - 25**.

```
14 // "enrollmentSubject": [{  
15 //   "enrollmentId": 1,  
16 //   "subjectId": ["Accounting", "Business", "Computing"]  
17 // }, {  
18 //   "enrollmentId": 2,  
19 //   "subjectId": ["Accounting", "Business", "Computing", "Design", "  
20 //   "enrollmentId": 3,  
21 //   "subjectId": ["Computing", "Education"]  
22 // }],  
23 // "subjectTypeArray": {},  
24 // }
```

- **Uncomment** the section under `/** 4.6.3 **/` (lines 76 - 81). Then click the Save button.

```

70  /** 4.6.3 */
71  // Function to implement a cascading dropdown.
72  // Some enrollment types now only have certain subject types available.
73  // When subjectTypeArray is bound to the Subject Type Datasource,
74  // it allows switching enrollment type to only show its specific subjects.
75  // onChange_EnrollmentType(...args) {
76  //   const subjects = this.state.enrollmentSubject.find(x => x.enrollmentType === args);
77  //   this.setState({
78  //     subjectTypeArray: subjects
79  //   });
80  //}

```

- Click on the Select Component of the **Enrollment Type** Form Item. In the **Events** tab, click on the **Component native event** button and then click **onChange** from the dropdown.

The screenshot shows the 'Course Info' form on the left and the 'Events' tab for the 'Enrollment Type' select component on the right.

Course Info Form:

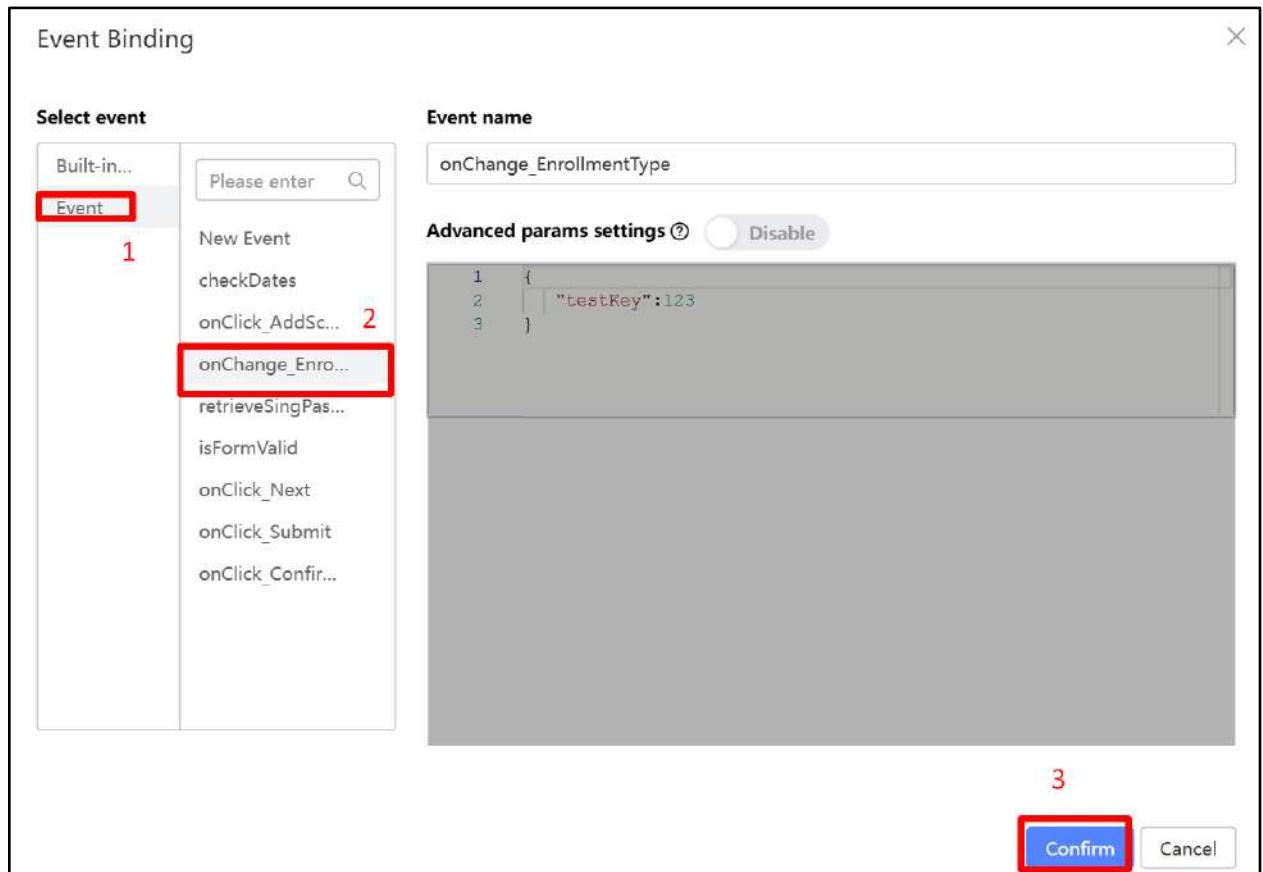
- Fields: Course Title, Course Code, Enrollment Type (highlighted with a red box), Subject Type, Course Fee, Description.

Events Tab:

- Event Type: Component native event (highlighted with a red box).
- Available Events:

onChange	Value changes
onVisibleChange	
onRemove	Tag deleted
onSearch	search

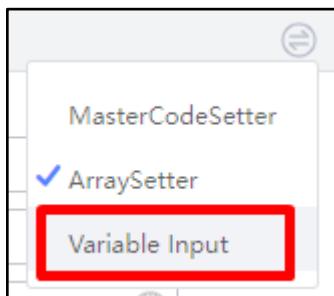
- Under Select event, click on **Event**, then click on **onChange_EnrollmentType**. Once done, click on the **Confirm** button.



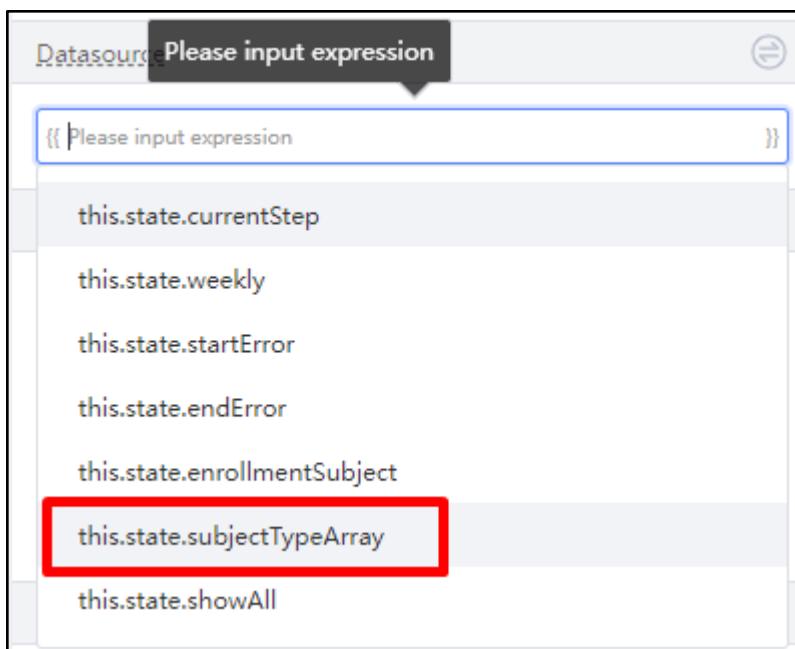
- Click on the Select Component of the Subject Type Form Item. Click on the **Switch Setter** on the top right of the **Datasource** panel.



- Select the Variable Input from the dropdown.



- Click on the “Please input expression” box and select **this.state.subjectTypeArray**



You can now click preview and check out the result:

Part-time:

* Enrollment Type	Part-time
* Subject Type	<input type="button" value="^"/>
* Course Fee	Accounting Business Computing
Description	

Full-time:

* Enrollment Type	Full-time
* Subject Type	<input type="button" value="^"/>
* Course Fee	Accounting Business Computing Design Education
Description	

Online/Remote:

* Enrollment Type	Online/Remote
* Subject Type	<input type="button" value="^"/>
* Course Fee	Computing Education

Practical 4.7: Setting field values using Form Variables (Optional)

4.7.1 Context

Before we go to the next Practical, let's say now that we want to enhance this form a little bit. Maybe the Form can also, in practice, take in the personal particulars of this instructor with MyInfo. Of course, we are not going to actually connect to MyInfo to get actual data - that's the job of the backend developers.

But we can still populate the form in the Kaizen App Designer, sort of like a proof-of-concept for the actual form. We can do this by using a function of the Next.Field library, called `setValue()`.

4.7.2 Requirements

Add the following to the 2nd Form: Instructor Particulars.

- A button at the top of the form, that when clicked, populates all the fields (except Salutation) with fixed data.

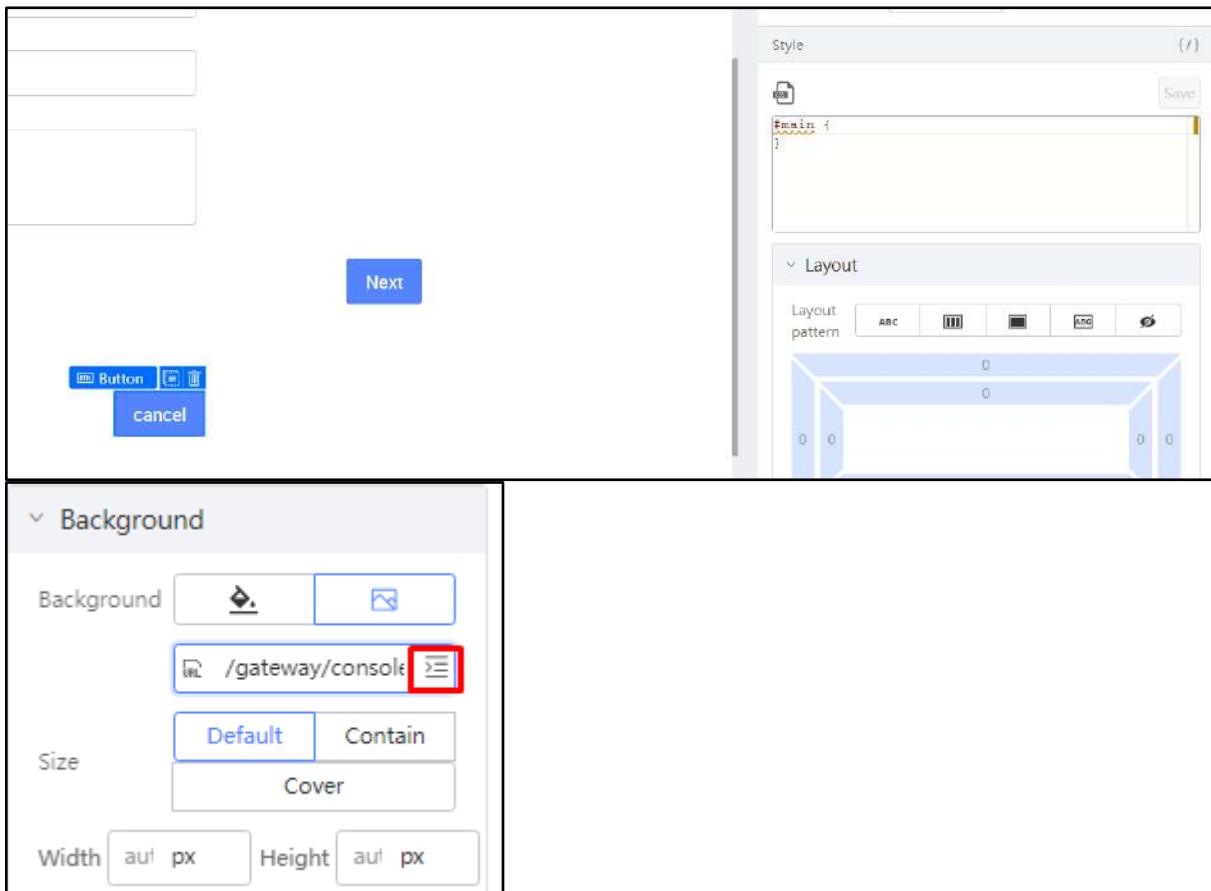
4.7.3 Adding the 'MyInfo' Button

- Scroll to the Instructor Particulars Form.
- Add a button to the top of the Form below the title. Remove the Form Item Label. Under the button's Form Item Styles tab, Shift button to the right.

The screenshot shows the 'Instructor Particulars' form. At the top left is a 'Form Item:' dropdown set to 'cancel'. Below it are two fields: 'Salutation' (dropdown menu) and 'Name' (text input). A modal dialog is open over the form, titled 'Form Item Styles'. It contains settings for 'Font weight' (500 Medium), 'Font family' (Please Select), 'Text color' (white), 'Align' (Right, highlighted in green), and 'Background' (background image and opacity slider). A tooltip 'text-align: right' points to the 'Align' setting.

- Download the sample MyInfo button image here [MyinfoBtn.png](#)
- Upload the MyinfoBtn.png image to your assets -> image

- Under the button's Styles tab, change the button background image to MyInfo.



Button Style

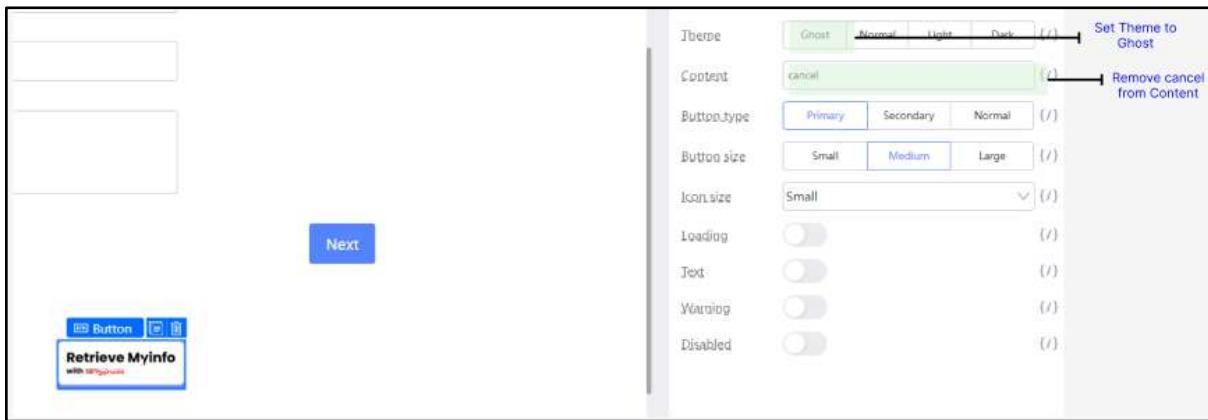
```

background-image:
url(/gateway/console/api/v1/asset/designerTraining/assets/image/MyinfoBtn.png?b
ranchName=main);
background-size: contain;
width: 150px;
height: 55px;
background-repeat: no-repeat;

```

- Go to the Button's Props tab and set the Theme to 'Ghost'. Remove 'cancel' from the

Content field.



Button Props

Theme: Ghost

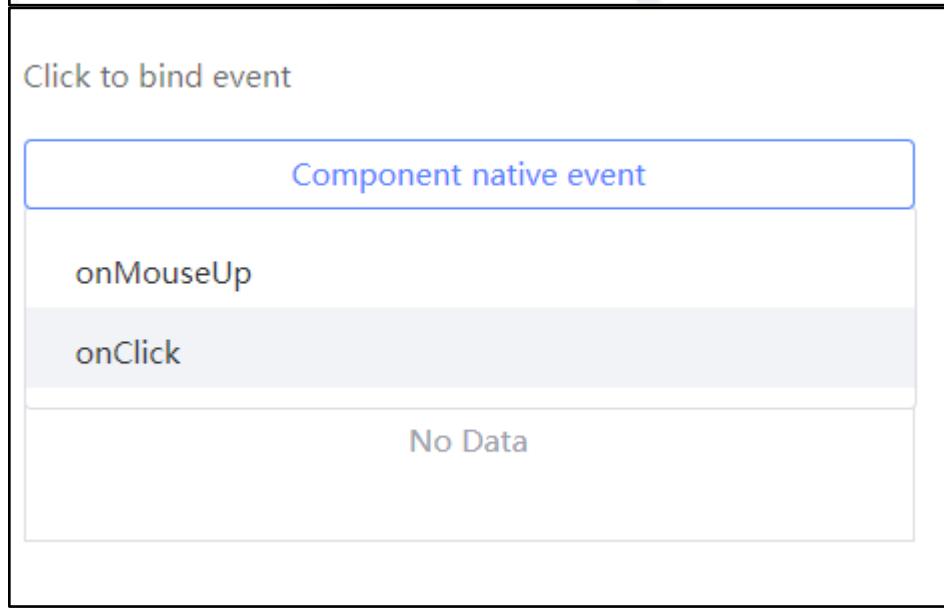
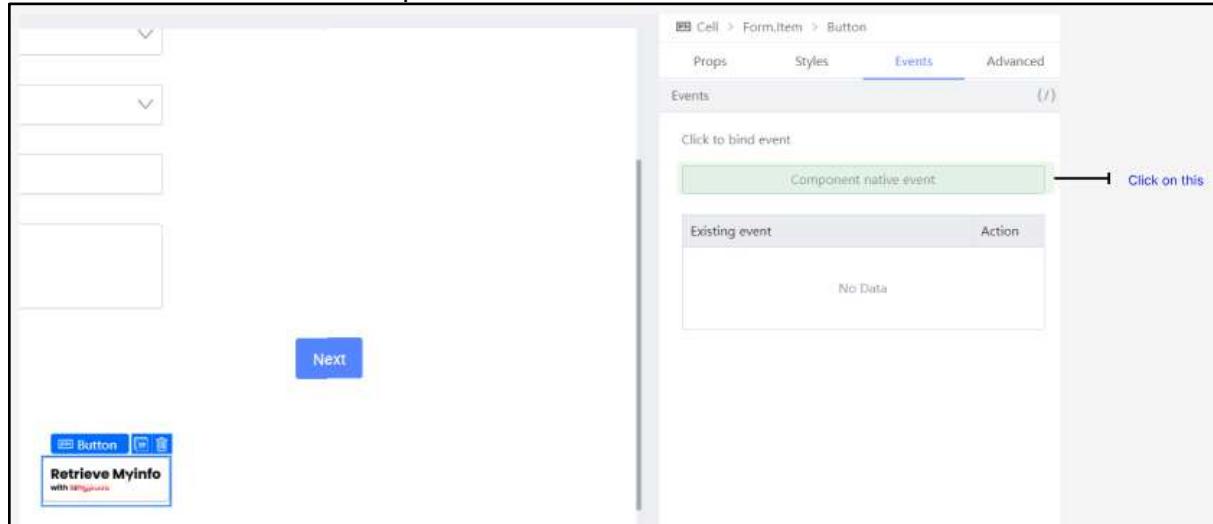
Content: (Clear this field)

4.7.4 Adding the logic

- **Uncomment** the section under `/** 4.7.4 **/` (lines 85 - 90). Then click the Save button.

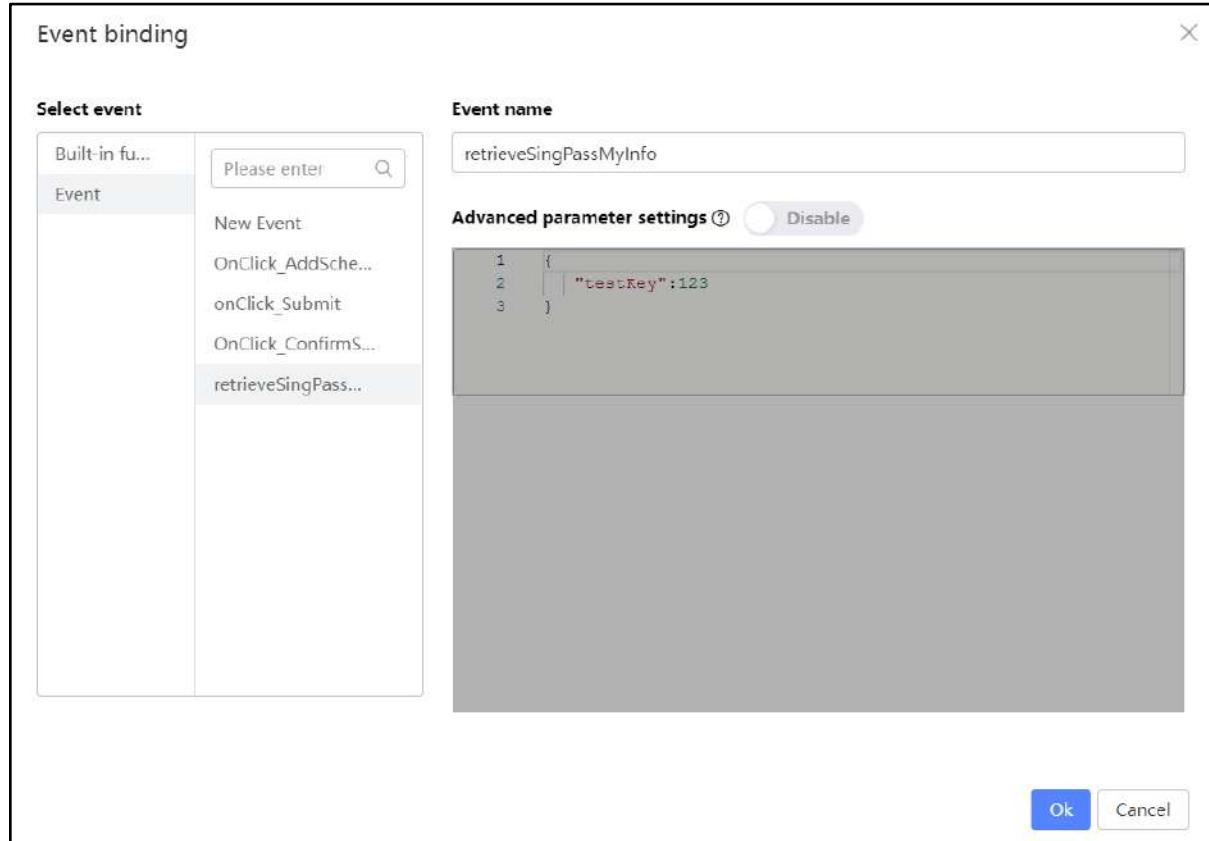
```
82  /** 4.7.4 **/
83  // // Function to set values in the fields for the Instructor Particular
84  // retrieveSingPassMyInfo() {
85  //   this.formField.setValue("name", "Samuel Ong Jin Hai");
86  //   this.formField.setValue("nruc", "S3052824G");
87  //   this.formField.setValue("email", "ongjinhai2015@gmail.com");
88  //   this.formField.setValue("contactNo", "99120384");
89  // }
```

- With the MyInfo button selected, go to **Events** tab and click **Component native event** and select '**onClick**' in the dropdown.



- Select **Event** under 'Select Event' and type in **retrieveSingPassMyInfo** under 'Event'

Name'. Click the 'Ok' button.



- Preview and check if the fields are populated correctly by clicking on the MyInfo button.

The screenshot shows a form preview with five fields:

- * Salutation: A dropdown menu showing 'Please select'.
- * Name: An input field containing 'Samuel Ong Jin Hai'.
- * NRIC: An input field containing 'S3052824G'.
- * Email: An input field containing 'ongjinhai2015@gmail.com'.
- * Contact No: An input field containing '99120384'.

Below the form are 'Previous' and 'Next' navigation buttons.

Practical 4.8: Binding it all together

Now that we have done the content, we are finally going to make the Step Form itself work.

4.8.1 Adding the code

- Uncomment the section under `/** 4.8.1 **/` (lines 94 - 103, 106 - 113). Then click the Save button.

```
91  /** 4.8.1 */
92  // // Function to check if the current form has any errors.
93  // isFormValid() {
94  //   var errors = this.formField.getErrors();
95
96  //   for (const value of Object.values(errors)) {
97  //     if (value !== null) {
98  //       return false;
99  //     }
100 //   }
101 //   return true;
102 //}
103
104 // // Function to progress the form to the next step.
105 // onClick_Next(e, v) {
106 //   if (this.isFormValid()) {
107 //     nextStep = this.state.currentStep + 1;
108 //     this.setState({
109 //       currentStep: nextStep
110 //     });
111 //   }
112 // }
```

4.8.2 Bind currentStep variable to Step Component

- Select the Steps Component at the top. Bind the variable `this.state.currentStep` to the Step Component.

The screenshot shows the Figma interface with a 'Steps' component at the top and a 'Variable binding' dialog open.

Steps Component: A horizontal navigation bar with four steps: Step 1 (Course Info), Step 2 (Instructor Particulars), Step 3 (Class Location), and Step 4 (Class Schedule). The 'Props' panel on the right shows the 'Step item' section with four items labeled Step 1 through Step 4.

Variable binding Dialog:

- Variable List:** A sidebar with three tabs: State attribute (selected), Custom function, and Datasource. The State attribute tab shows a search bar with the word "weekly".
- Bind variables:** A main area containing the binding expression:

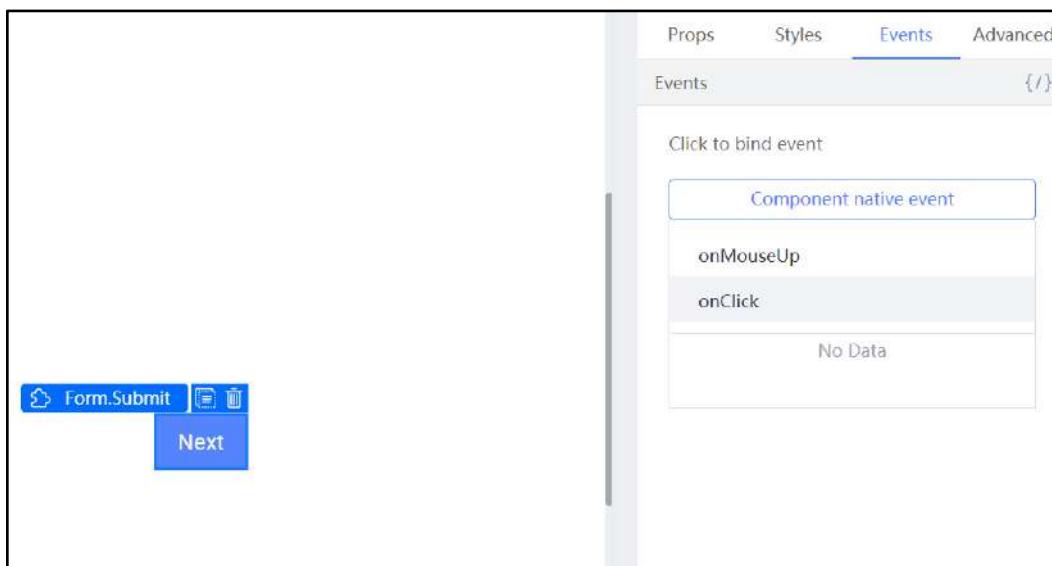
```
1 this.state.currentStep
```
- Usage:** A description of how to bind variables or functions, mentioning the 'this' instance object and the 'this.state' data object.
- Buttons:** At the bottom are 'Remove Binding' (red button), 'Confirm' (blue button), and 'Cancel' (white button).

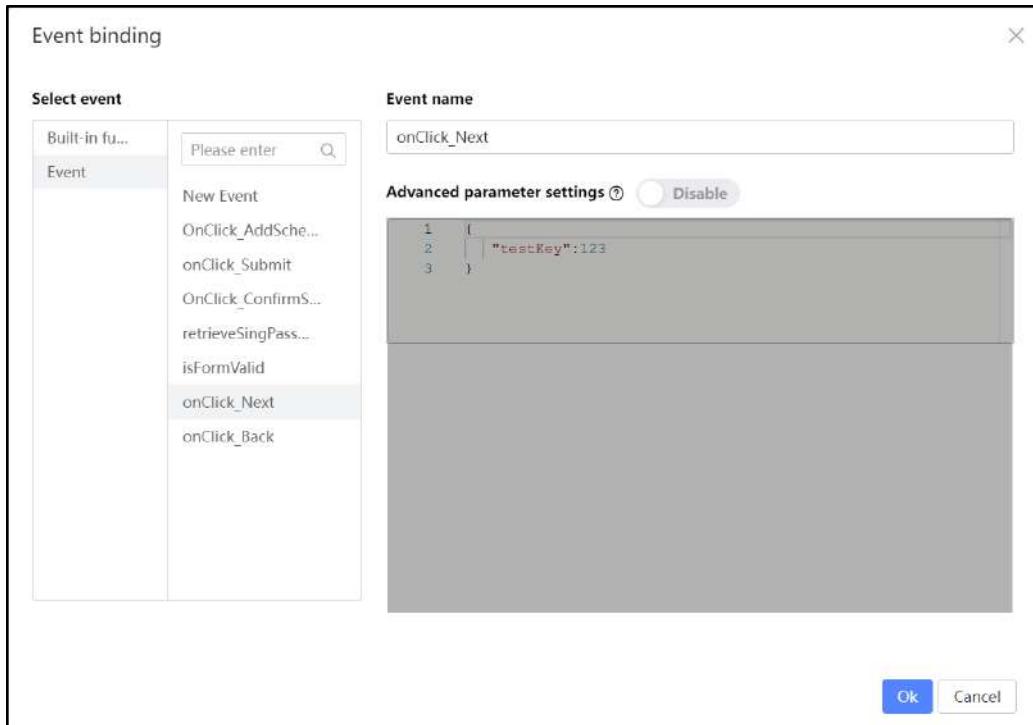
4.8.3 OnClick Events for the Next buttons

- Select all the **Previous Buttons** in the Forms and **disable** them.

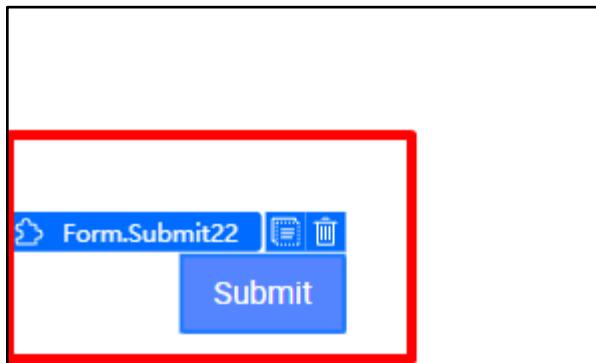
The screenshot shows a four-step form process. Step 1 is titled 'Course Info'. It contains fields for Course Title, Course Code, Enrollment Type, Subject Type, and Course Fee, each with validation messages. Below these is a 'Description' text area. At the bottom left is a 'Previous' button with a red border and a 'Next' button. To the right is a sidebar titled 'Disabled' with a toggle switch that is currently on. Other settings in the sidebar include 'Label' (set to 'Course'), 'Type' (set to 'Text'), 'Format' (set to 'Normal'), and various validation and loading options.

- Select the **Next Button** in the **Course Info Form** and go to the **Events** tab. Bind **onClick** to the **onClick_Next** event.



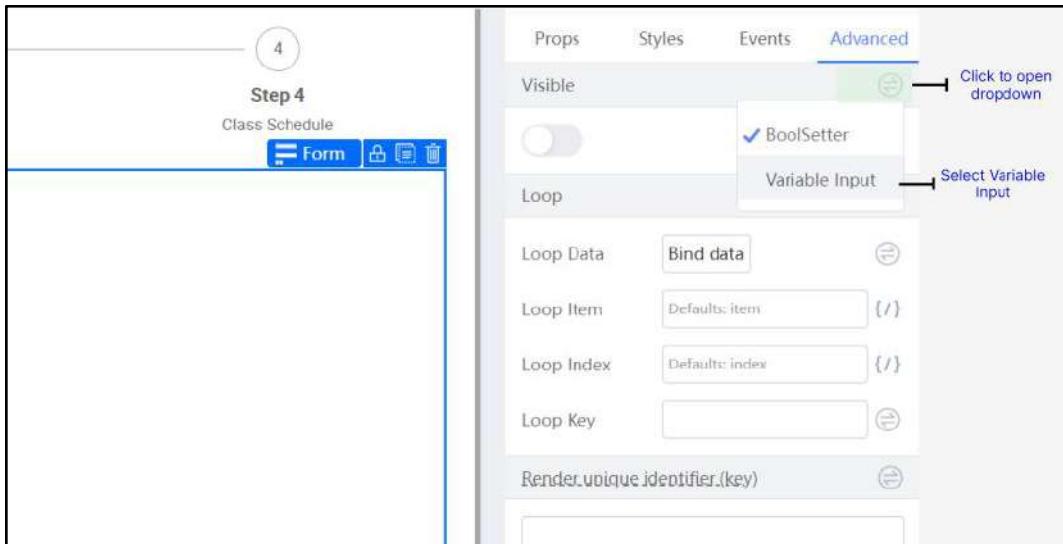


- Do the same for the 'Instructor Particulars' (2nd Form) and 'Class Location' Forms (3rd Form).
- For the 'Class Schedule' Form (4th Form), change the Label of the Next button to 'Submit'.



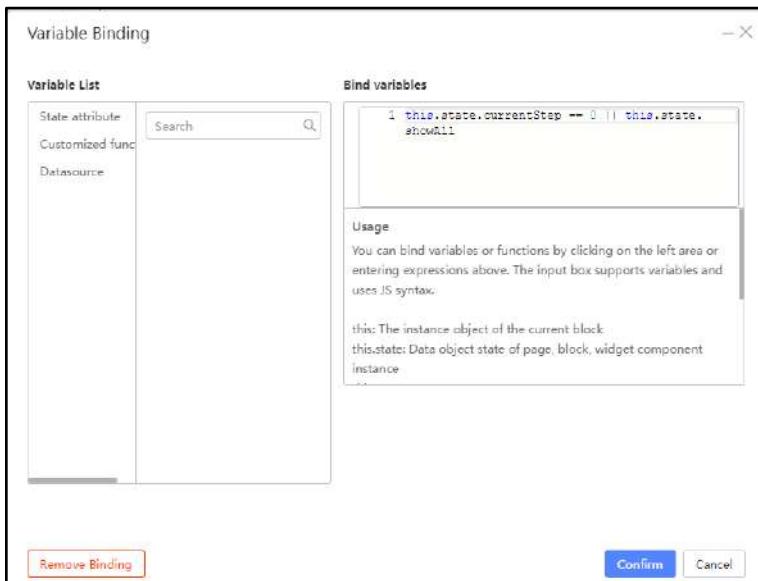
4.8.4 Set conditions for the Forms to be visible

- Select the **Course Info Form**. In the **Advanced** tab, click on the top-right for the 'Visible' field and select **Variable Input**.



- Type this in the Bind Variables box:

```
this.state.currentStep == 0 || this.state.showAll
```

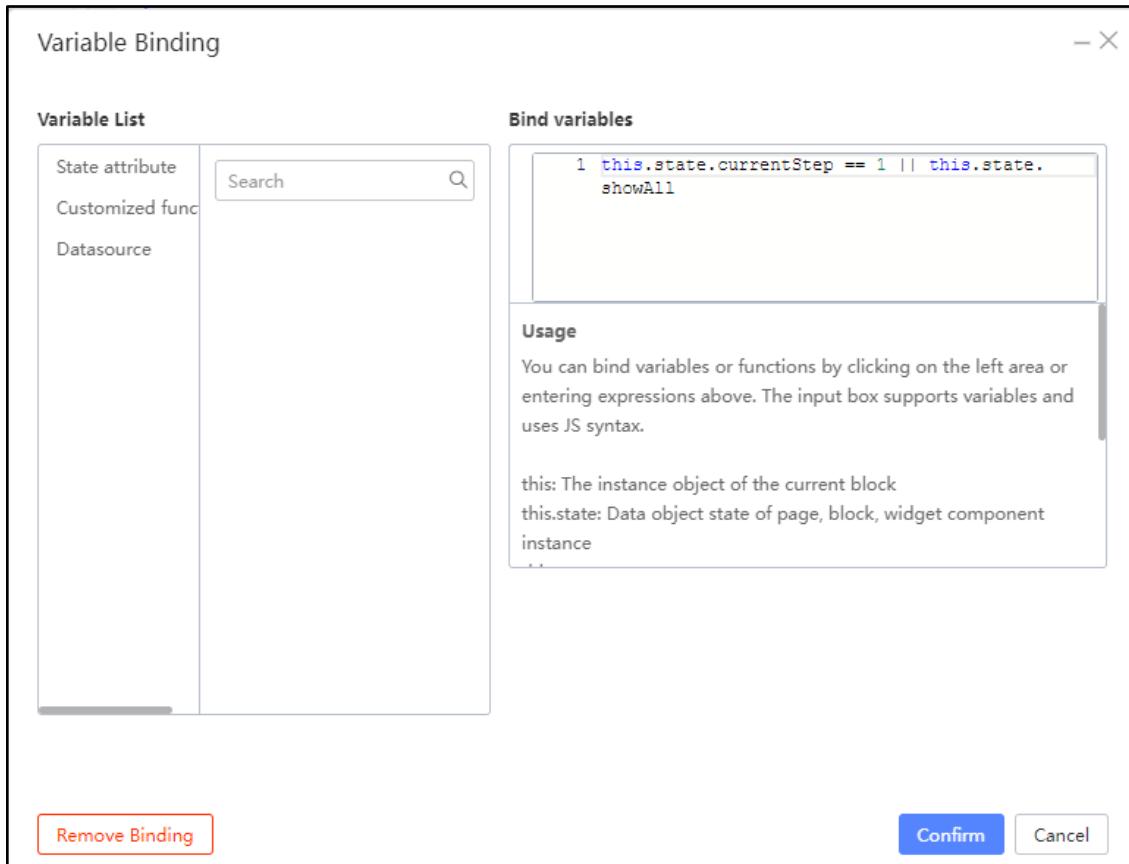


Then click the 'Confirm' button.

- For 'Instructor Particulars' Form bind this:

```
this.state.currentStep == 1 || this.state.showAll
```

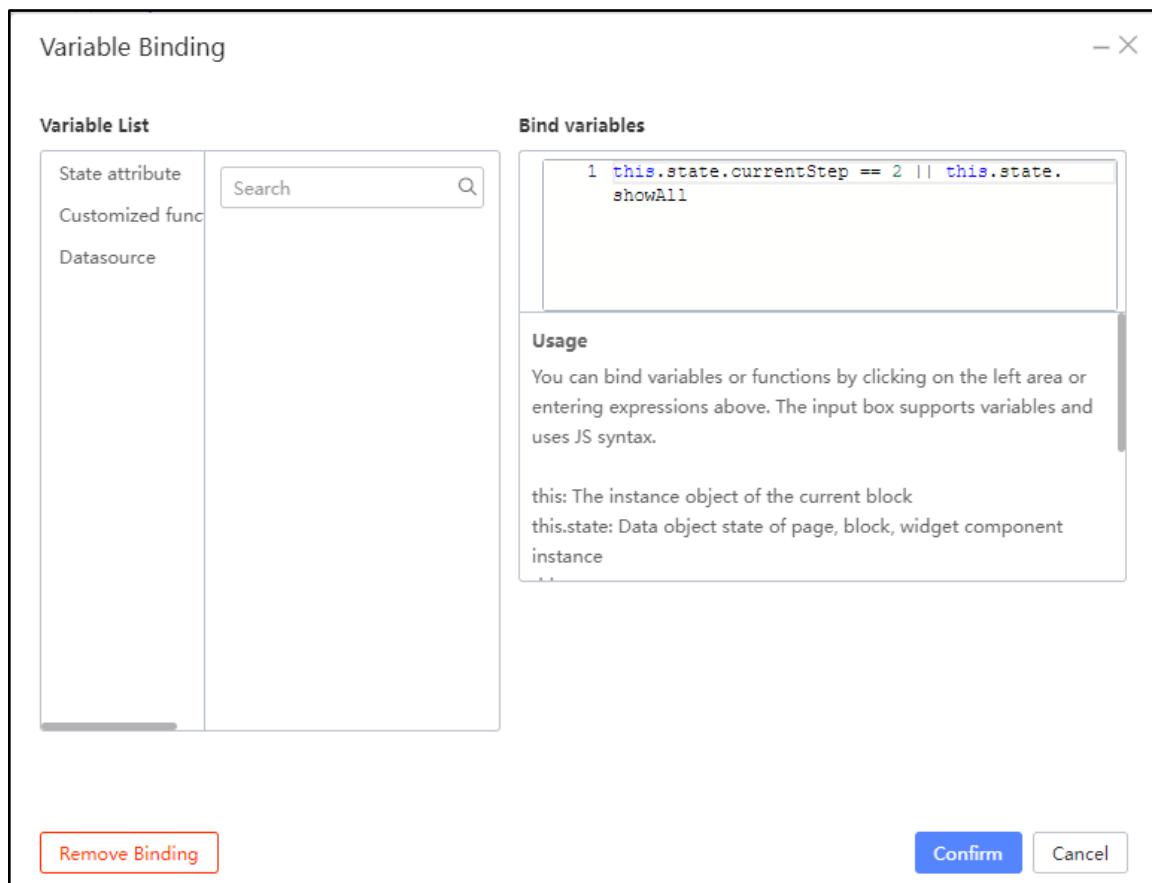
to the 'Visible' Property in the Advanced tab.



- For 'Class Location' Form bind this:

```
this.state.currentStep == 2 || this.state.showAll
```

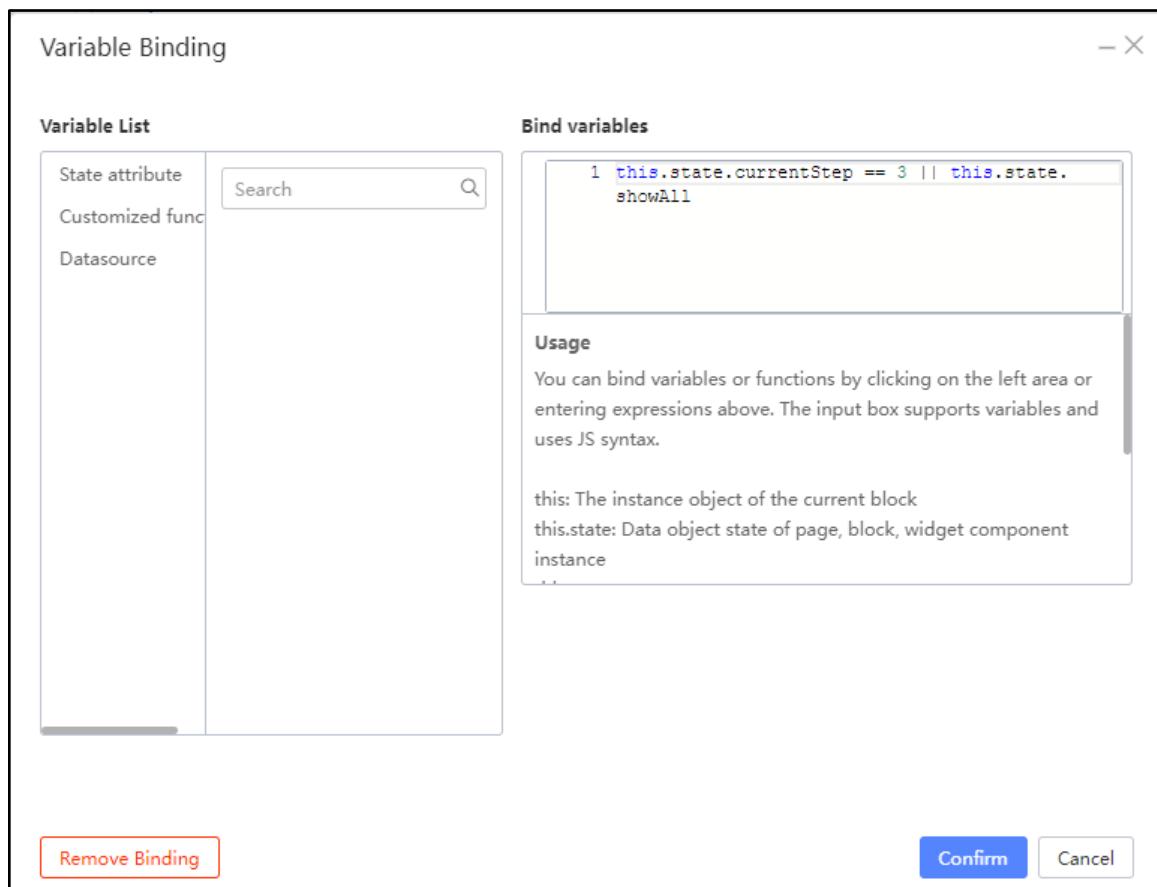
to the 'Visible' Property in the Advanced tab.



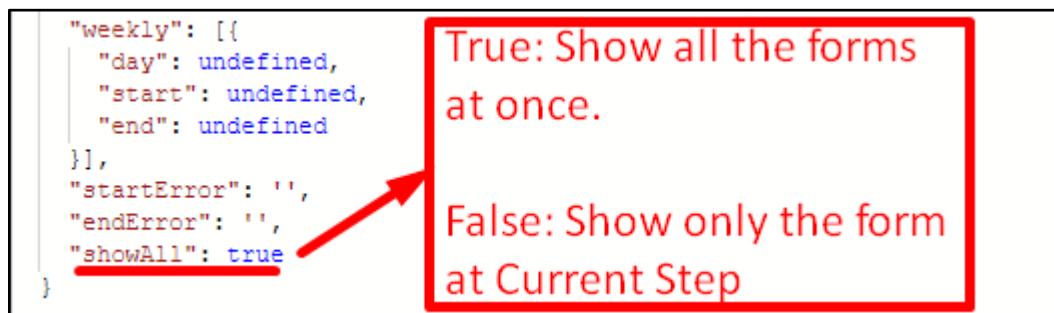
- For 'Class Schedule' Form bind this:

```
this.state.currentStep == 3 || this.state.showAll
```

to the 'Visible' Property in the Advanced tab.



Note: If you ever want to view all the forms again later, you can simply set the `showAll` state variable (line 13) to true.

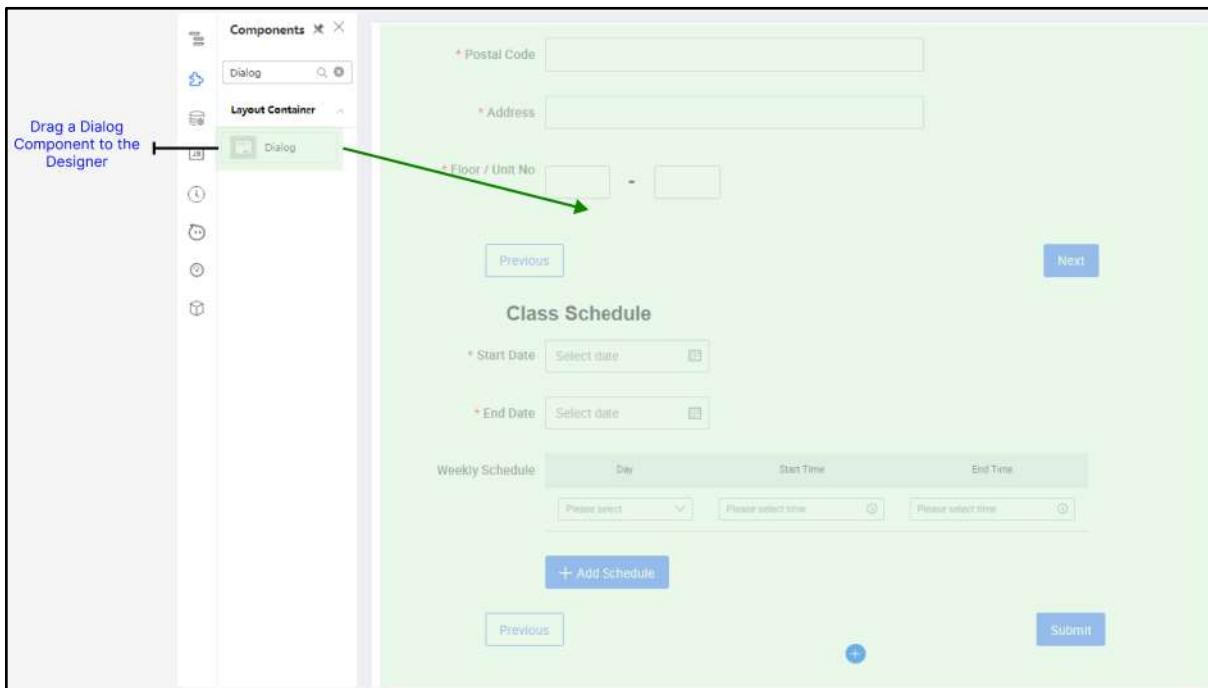


Practical 4.9: Modal Dialog Box

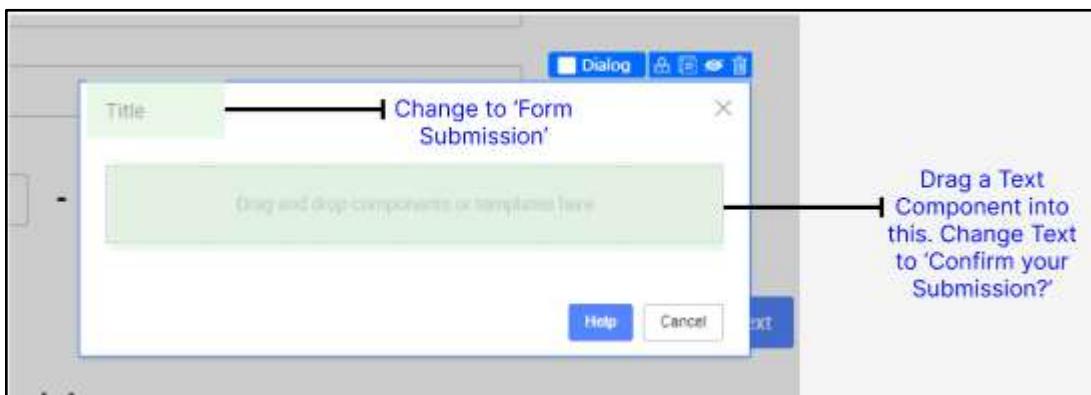
When we do a Submit, we might want a popup to appear asking if the user would want to confirm their submission before sending the information to the database. We can use a Modal dialog box for this purpose.

4.9.1 Steps for Creating a Modal Dialog

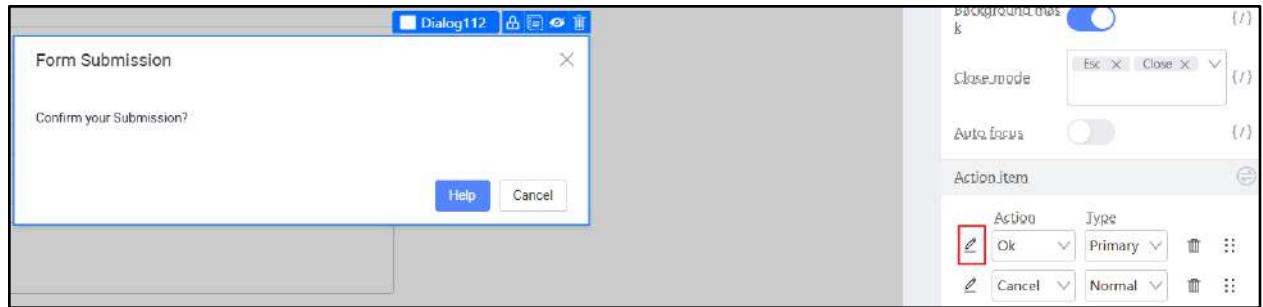
- **Drag a Dialog Component to the Designer.** Anywhere in the workspace will do.



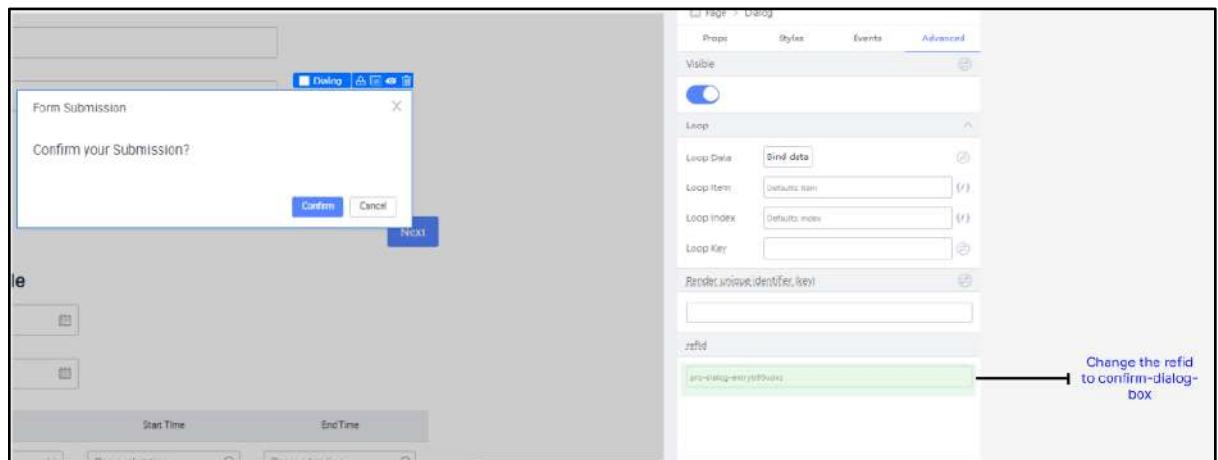
- Change the Title to '**Form Submission**' and add a **Text Component** to the Cell in the middle of the Modal Dialog box. Change the Text to '**Confirm your Submission?**'.



- Click on the Dialog Component and under the Props Tab, change the blue button's Label (Help) to 'Confirm' under the Dialog Component's Prop tab.



- Go to the **Advanced** Tab and change the **refId** to **confirm-dialog-box**

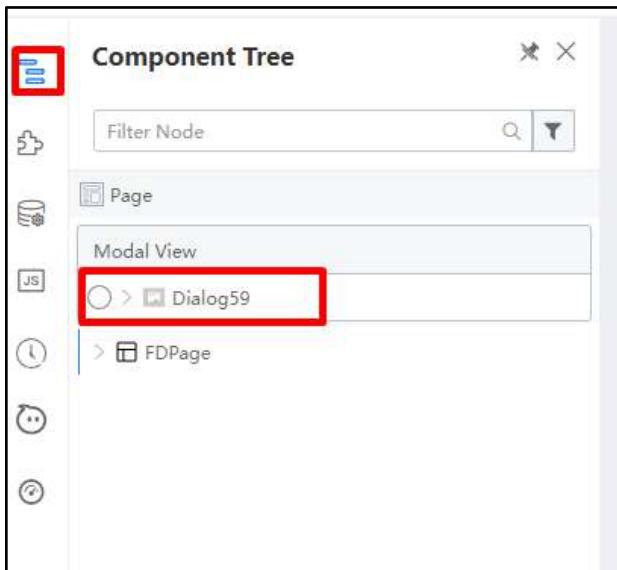


4.9.2 Adding the logic for the Modal Dialog

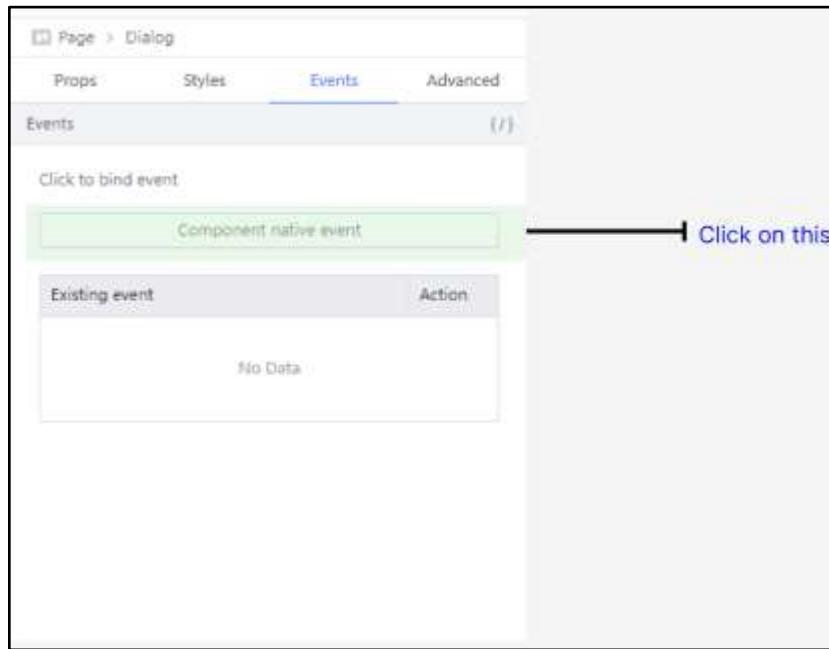
- Open the Source Code Panel and **uncomment lines 118 - 123 and 127 - 129**. And click **Save**.

```
114  /** 4.9.2 */
115  // // Function open a dialog box when the form is being submitted.
116  // // Dialog will ask for user confirmation of submission.
117  onClick_Submit(e, v) {
118      this.setState({
119          | confirmVisible: true
120      });
121      this.$("confirm-dialog-box").open();
122  }
123
124  // // Function to return to the Course Listing page
125  // // (simulating a form submission)
126  onClick_ConfirmSubmit() {
127      | this.utils.navigateTo('justToTest', 'listing');
128  }
```

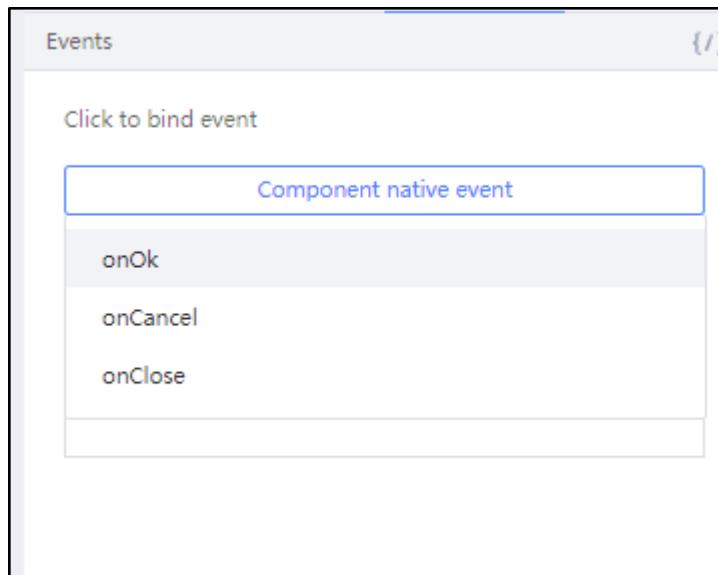
- Select the Dialog Component (Under Component Tree)



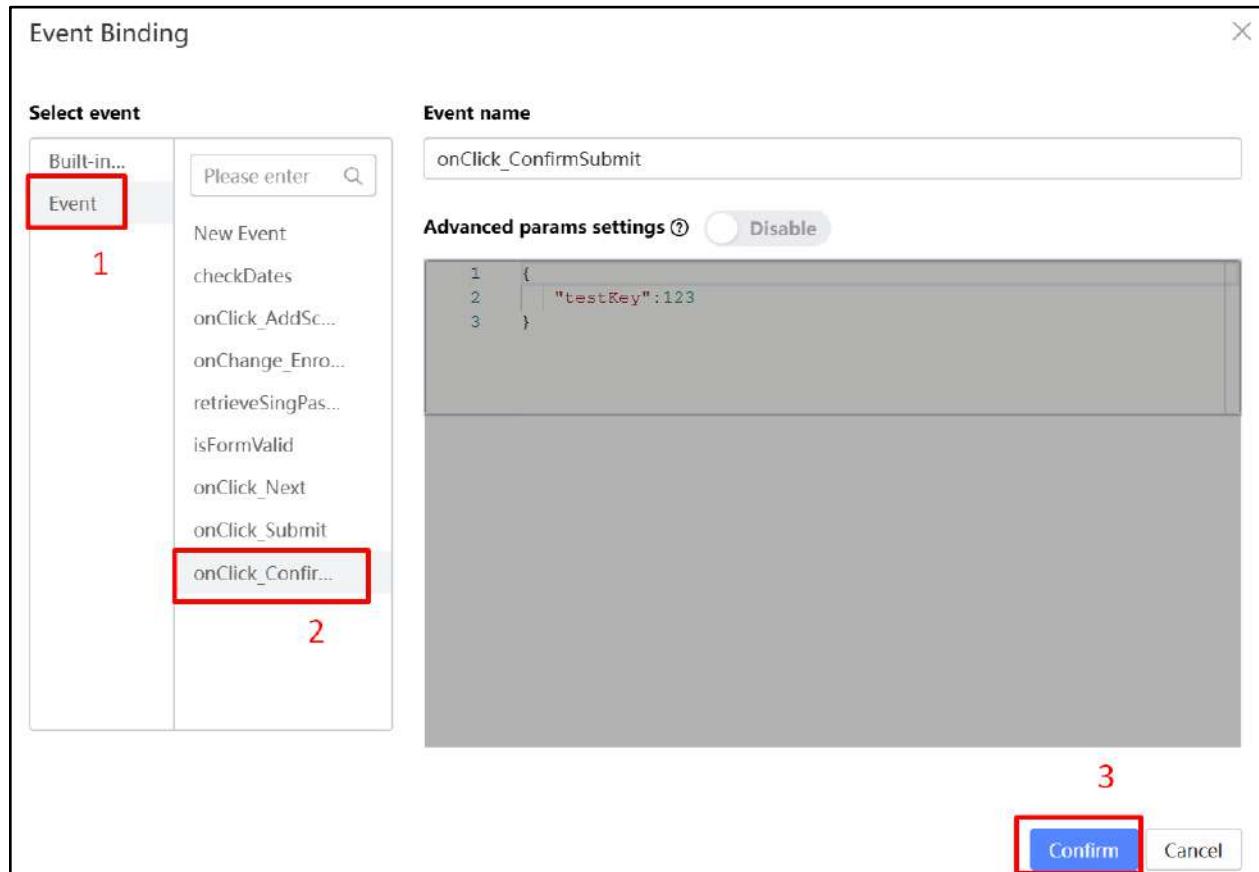
- Under the **Events** tab and click on the **Component native event** button.



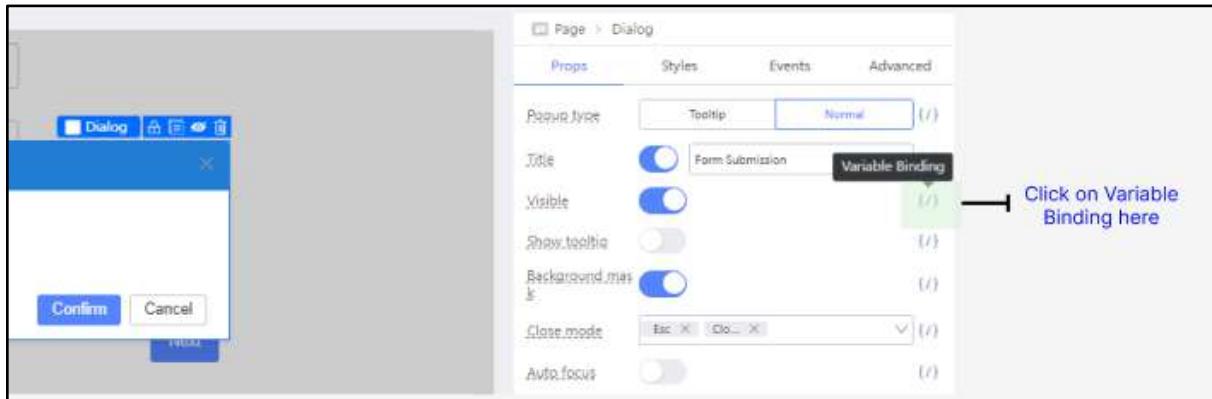
- Select the **onOk** event from the dropdown that appears.



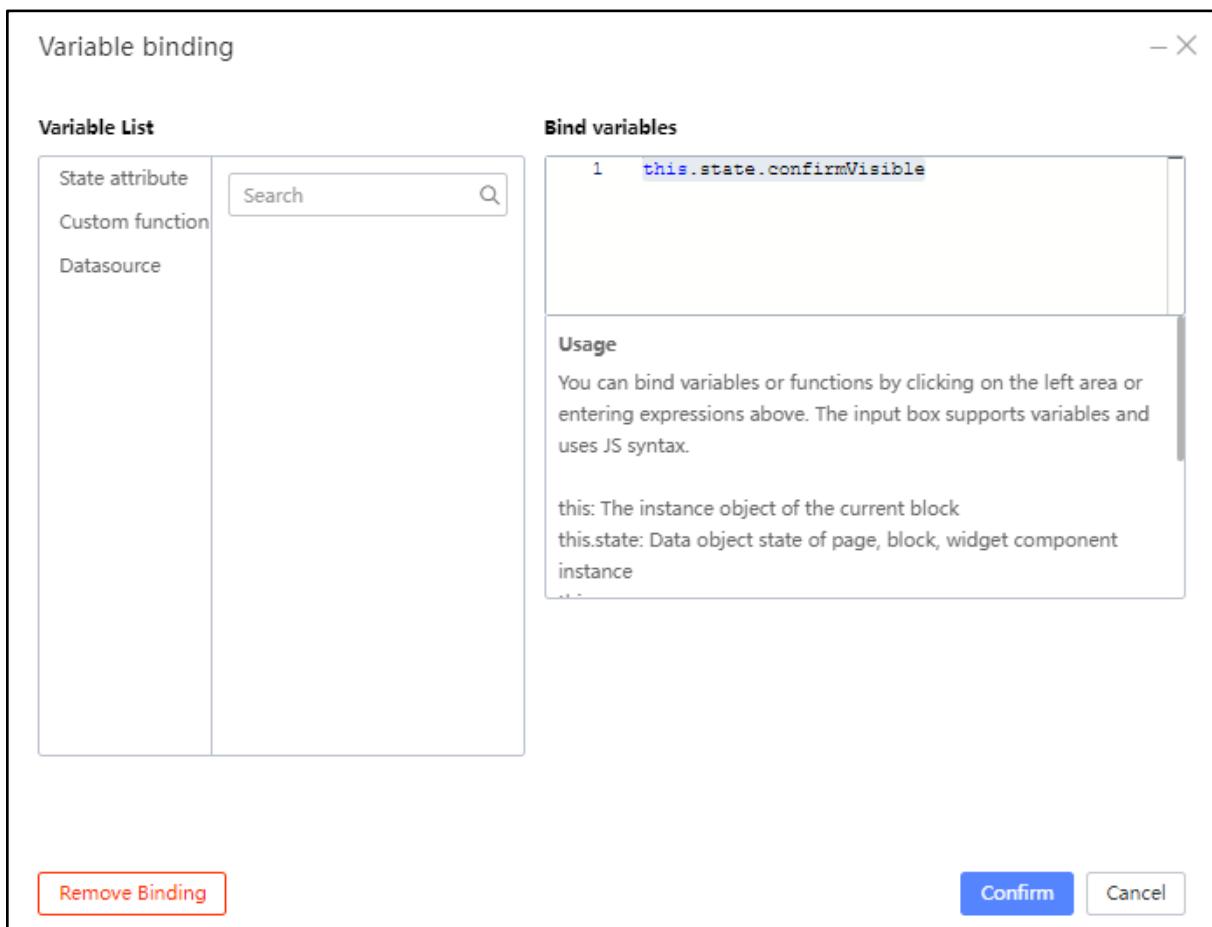
- Under **Event** and then click '**OnClick_ConfirmSubmit**'.



- Go to the Props tab for this Dialog Component. For the 'Visible' field, click on the Variable Binding {}.



- Under Bind Variables, enter the following: **this.state.confirmVisible**. Once done, click Confirm.



- Select the Submit button on the ‘Class Schedule’ Form, and go to the Events tab. Click on the ‘Component native event’ button.
- Temporary changes the **showAll** flag to true

Source Code Panel

index.js index.css Save

```

1 class LowcodeComponent extends Component {
2   /** State Variables. */
3   // This stores global variables for the page
4   // which can be used in this code or for variable binding in Components
5   state = {
6     "currentStep": 0,
7     "startError": '',
8     "endError": '',
9     "weekly": [
10       {
11         "day": undefined,
12         "start": undefined,
13         "end": undefined
14       },
15       {
16         "enrollmentSubject": [
17           {
18             "enrollmentId": 1,
19             "subjectId": ["Accounting", "Business", "Computing"]
20           },
21           {
22             "enrollmentId": 2,
23             "subjectId": ["Accounting", "Business", "Computing", "Design", "Education"]
24           },
25           {
26             "enrollmentId": 3,
27             "subjectId": ["Computing", "Education"]
28           }
29         ],
30         "confirmVisible": false,
31         "subjectTypeArray": {},
32         "showAll": true
33       }
34     ]
35   }
36 }

```

Cell > FormItem > Form.Submit

Props Styles Events Advanced

Events Click to bind event Component native event Click on this

Existing event	Action
No Data	

Next

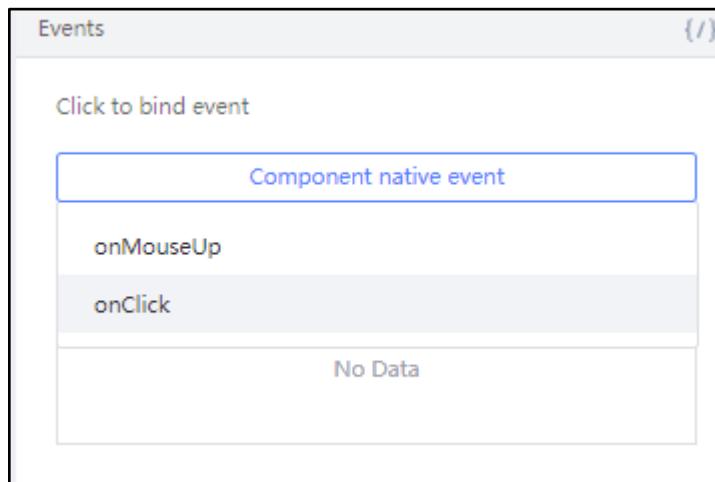
End Time

ct time

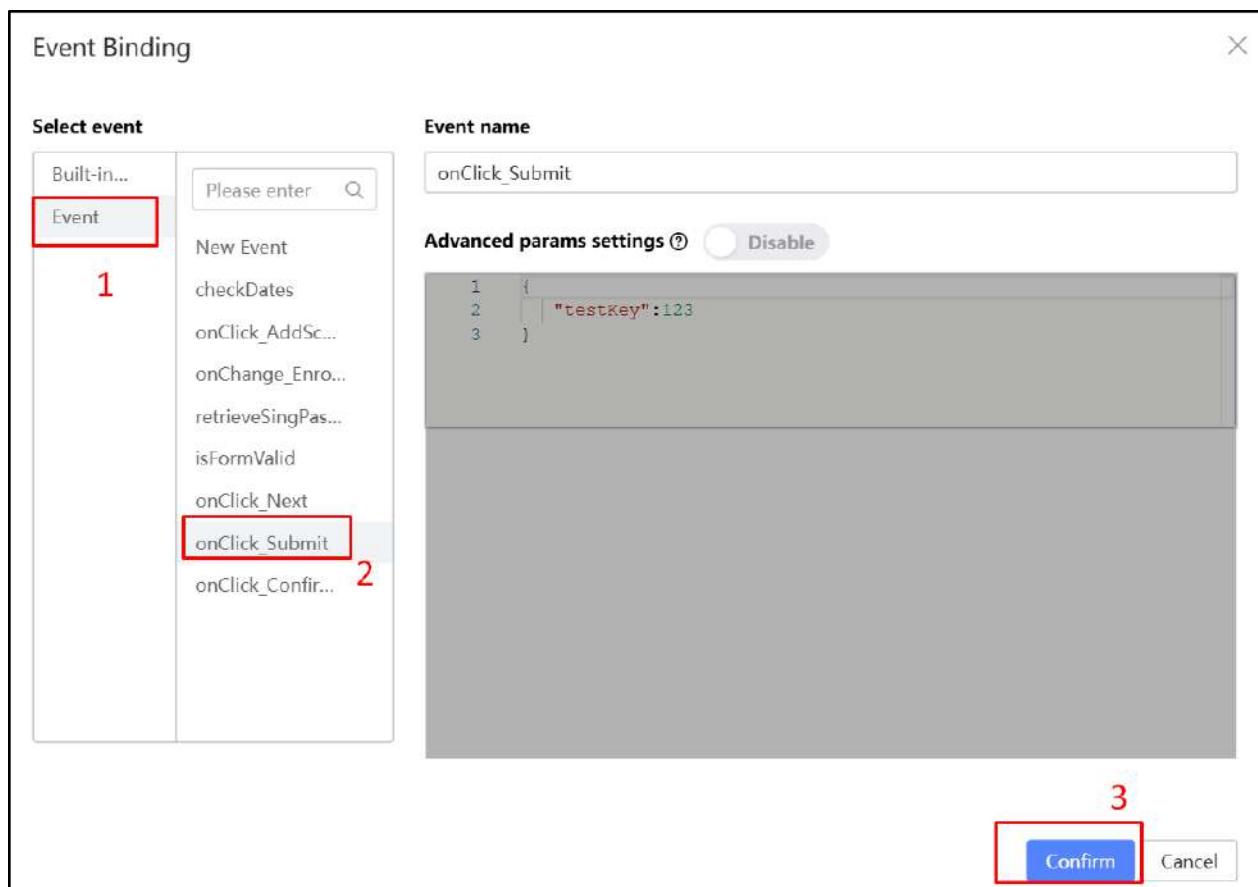
Form.Submit [≡]

Submit

- Select ‘onClick’ in the dropdown that appears.



- Under ‘Select event’, click on ‘Event’, and then click on ‘onClick_Submit’.



- Now go back to your Source Code Panel, change back the flag to false

```

    "confirmVisible": false,
    "subjectTypeArray": {},
    "showAll": false
}

```

- Change your **OnClick_ConfirmSubmit** function to navigate back to your course listing page, by changing the `navigateTo` in [utils¹](#) function to use the correct App ID and Page ID.

If you have not completed tutorial 3 in creating your course listing page, you may navigate to your landing page instead for now.

```

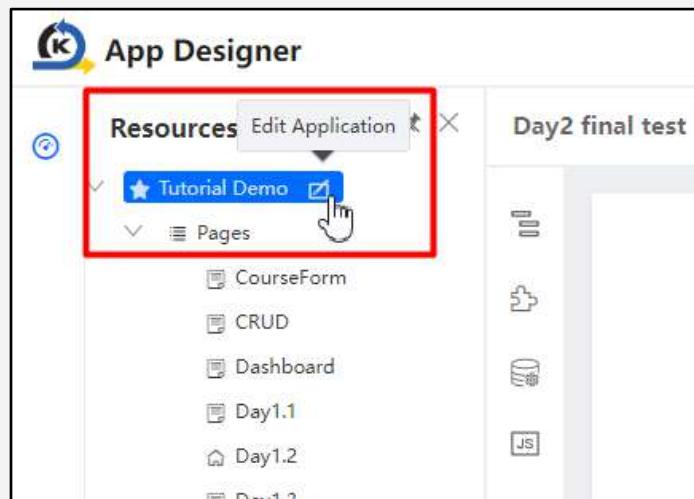
64   onClick_Submit(e, v) {
65     // if (this.isValidForm()) {
66     this.setState({
67       confirmVisible: true
68     });
69     this.$("confirm-dialog-box").open();
70   }
71 }
72
73   onClick_ConfirmSubmit() {
74     this.utils.navigateTo('tutorialdemo', 'listing');
75   }
76

```

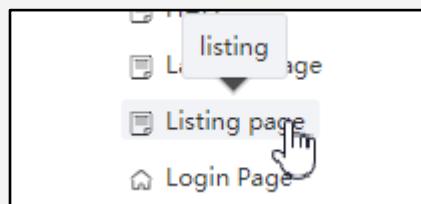
The code shows a call to `this.utils.navigateTo('tutorialdemo', 'listing');`. Two red arrows point from the text "App ID" and "Page ID" to the strings "tutorialdemo" and "listing" respectively, indicating they are the parameters for the `navigateTo` function.

Having trouble finding the **App ID** and **Page ID**?

- You can find the **App ID** under resources. Click on the App, then click on Edit Application.



- Similarly, you can find the **Page ID** under resources as well. All you have to do is to hover over your page created in Day 2 and you will get the Page ID. You may also click on edit page to view the Page ID.



And Finally...

Let's go through the form together. Click Preview.

Step 1: Course Info

The form is titled "Course Info". It contains the following fields:

- * Course Title: Business Foundations
- * Course Code: 1224
- * Enrollment Type: Part-time
- * Subject Type: Business
- * Course Fee: \$ 1000
- Description: (Text area)

At the bottom are "Previous" and "Next" buttons.

Step 2: Instructor Particulars

The form is titled "Instructor Particulars". It contains the following fields:

- * Salutation: Mr
- * Name: Samuel Ong Jin Hai
- * NRIC: S3052824G
- * Email: ongjinhai2015@gmail.com
- * Contact No.: 99120384

There is a "Retrieve Myinfo" button with "with Singpass" text. At the bottom are "Previous" and "Next" buttons.

Step 3: Class Schedule

The form is titled "Class Location". It contains the following fields:

- * Postal Code: 643527
- * Address: 527 Jurong West Street 52
- * Floor / Unit No.: 10 - 224A

At the bottom are "Previous" and "Next" buttons.

Step 4: Class Location

Step 1 Course Info Step 2 Instructor Participants Step 3 Class Location Step 4 Class Schedule

Class Schedule

* Start Date: 2024-03-30

* End Date: 2024-04-20

Weekly Schedule Day Start Time End Time

Day	Start Time	End Time
Monday	16:00	17:00
Tuesday	17:00	18:00
Wednesday	16:00	17:00

+ Add Schedule

Form Submission: Redirect back to Course Listing

Dashboard > Course Listing

Course Listing

Course Name:

Category:

Rating (Avg):

Course Fees: to

Course Listing

Introduction to Programming
Learn the basics of coding
 Good Online
Course Fee: \$49.99

Data Science Fundamentals
Explore the world of data analysis
Course Fee: \$79.99

Data Analytics Fundamentals Course

Additional Information

1. Utils functions summary

Data Storage Utilities

- `this.utils.dataStore`: Stores and retrieves data for page references.
 - Methods: `getData(key)`, `setData(key, value)`, `clearData()`.
- `this.utils.context`: Passes data between pages.
 - Methods: `getDataMap(object)`, `setData(object, key, data)`,
`getData(object, key)`.

Third-Party Libraries

- `this.utils.FileSaver`: Uses the **file-saver** library for file operations.
- `this.utils.dateFormat`: Uses the **dateformat** library for formatting dates.

Navigation Utilities

- `this.utils.navigateTo(appId, pageId, data, object)`: Navigates to a specific page within an app (`object` should be `this`).
- `this.utils.navigateToCurrAppPages(pageId, data, object)`: Navigates within the current app pages.
- `this.utils.getNavigatedData(key, object)`: Retrieves data passed between pages.

Security and Role Management

- `this.utils.RSAUtil`: Encrypts passwords.
 - Methods: `convertJwkToPem(jwk)`, `encrypt(text, pubKey, keyFormat)`.
- `this.utils.RolesUtils`: Manages user roles for filtering navigators in preview mode.
 - Methods: `delete(role, userDomain)`, `set(role, userDomain)`,
`getRoles()`, `clear()`, `saveRoles(roles, userDomain)`.

Utility Functions

- `this.utils.getUrlParams()`: Retrieves URL parameters.
- `this.utils.hasSuperAdminPrivilege()`: Checks if the current user is a super admin.
- `this.utils.getAssetsUrl(assetsName, fileName)`: Retrieves asset file URLs.
- `this.utils.handleProfile`: Used to do virtual login with profile in preview, only work in preview and will not generate in runtime.
 - Methods: `handleProfile(data: {
 userDomainCode: string;
 userAccountId: string;
 vault: string })`.

Note: * indicates different implementations between **designer mode** and **runtime mode**.

Tutorial 5: Tables, Sorting and Pagination

This tutorial covers the following Learning Objectives:

- Understand how to create and structure tables in KAIZEN to display data effectively.
- Learn how to implement client-side sorting functionality to allow users to organize data by different columns.
- Explore how to add client-side pagination to tables to manage large data sets efficiently.
- Discover how KAIZEN simplifies the integration of sorting and pagination features to enhance user interaction.

In this tutorial, you will learn how to create and manage tables in KAIZEN to display data clearly and efficiently. We'll guide you through implementing sorting and pagination features, allowing users to interact with and organize large sets of data easily. By the end of this tutorial, you'll be able to enhance your application's data presentation with dynamic, user-friendly tables.

For this example, we will be doing a registered course list table, containing the Course Name, Course Code, Subject, Enrollment Type, Duration, Schedule, Price and the number of Students enrolled.

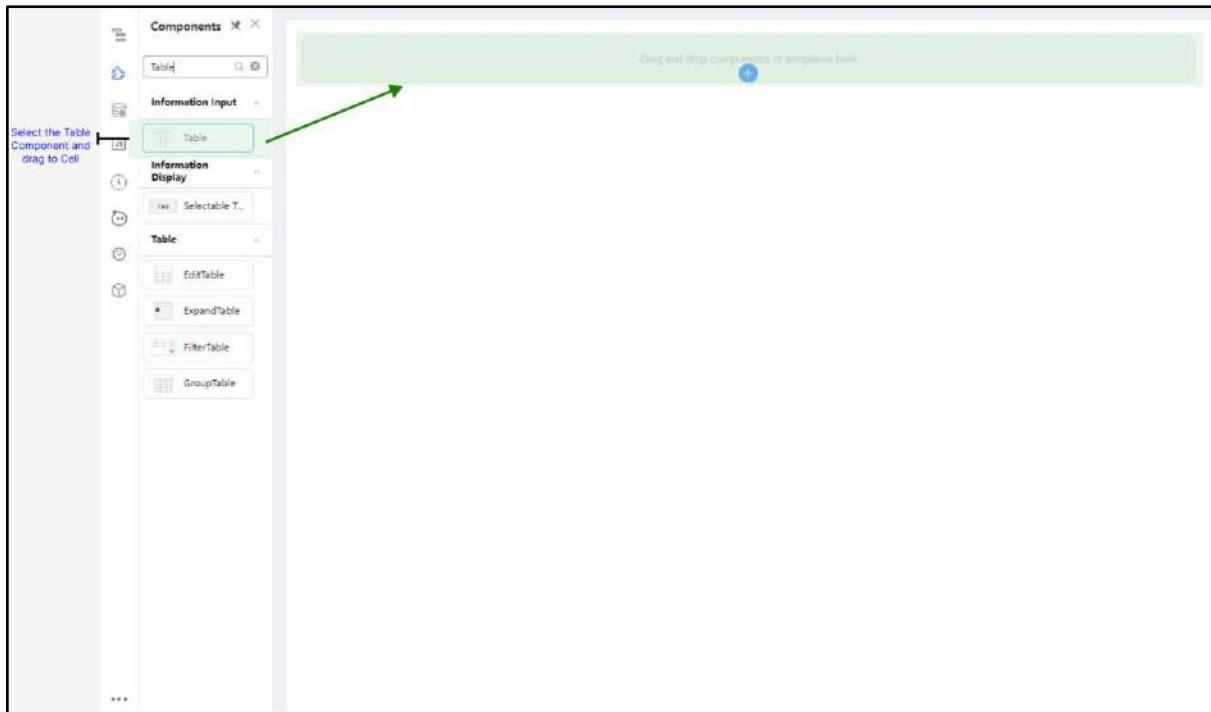
Course Name	Course Code	Subject	Enrollment Type	Duration	Schedule	Price	Students
Financial Reporting and Analysis	4156	Accounting	Full-time	4 months	weekly	2200	1325
Business Information Systems	7234	Computing	Part-time	3 months	bi-weekly	850	235
Strategic Design Thinking	6621	Design	Full-time	6 months	weekly	15000	245
Educational Leadership and Administration	4785	Education	Part-time	2 months	bi-weekly	3000	20
Business Ethics and Corporate Responsibility	8932	Business	Full-time	1 month	daily	2000	180
Management Accounting	2678	Accounting	Part-time	5 months	weekly	11000	125
Data Science for Business	5412	Computing	Full-time	4 months	bi-weekly	20000	150
Graphic Design Principles	6890	Design	Part-time	3 months	weekly	2000	90
Teaching Strategies for Diverse Learners	3245	Education	Full-time	2 months	bi-weekly	1500	20
International Business Management	9123	Business	Part-time	1 month	daily	10000	150
Total: 22							
Previous 1 2 3 Next >							

Practical 5.1: The Table Component

The Table Component is a template that you can use to create tables for your projects with relative ease. You can find it in the Component Library as a component called 'Table'. Simply drag it out into the design space as you would any other component and should work.

5.1.1 Setting up the Table Component

- Create a new Course Table Page.
- From the Component Library, **drag** out the **Table Component** into the Cell in the Design space.



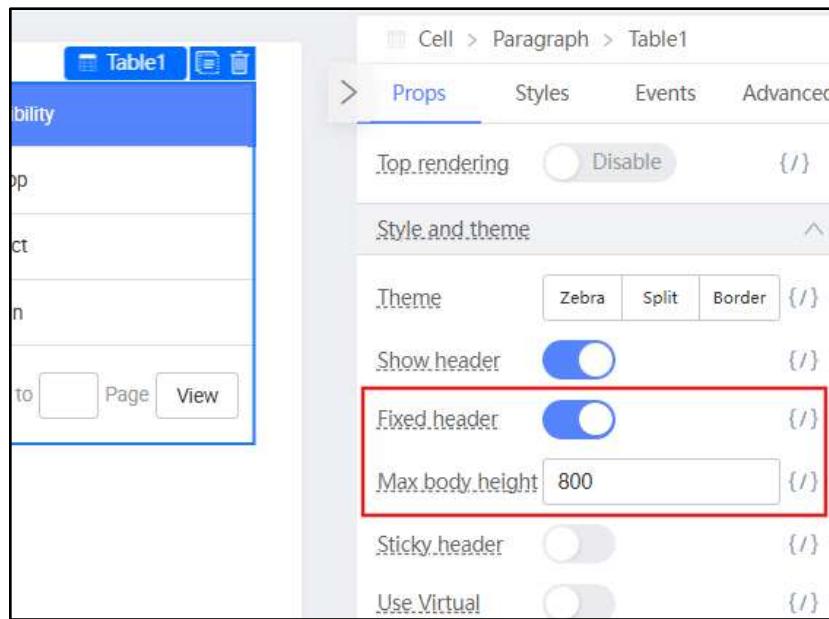
- Under Props:
 - Remove the **Add** and **Edit** buttons by deleting them under the **Action Bar**
 - Delete the **Edit** and **Preserve** links under the **Action Column** as well

The screenshot shows the 'Props' tab of the WYSIWYG editor for a table component. The table has three rows with columns for Name, Age, Responsibility, and Operation. The 'Action Bar' section shows 'Add' and 'Edit' buttons, which are highlighted with a red box and labeled 'Select the Table'. The 'Action Column' section shows 'Edit' and 'Preserve' links, also highlighted with a red box and labeled 'Click to delete both items'. The 'Props' tab includes sections for Title, Data columns, Data source, and Action column.

- Under Styles, change the width of the Table to 100%.

The screenshot shows the 'Styles' tab of the WYSIWYG editor for a table component. The table remains selected, indicated by a red box and the label 'Table remains selected'. The 'Layout' section shows a 'Layout pattern' grid with a green border around the table. The 'Width' input field is set to '100%', highlighted with a red box and labeled 'Change Width to 100%'. The 'Styles' tab includes sections for Style, Layout, Font, and other styling options.

- Go to the Props tab and go to **Style and theme**.
 - Turn on the **Fixed Header** switch and then
 - Set the **Max body height** to **800px**.



5.1.2 Setting the Table Headers

The configuration for changing the headers are located in the Props tab of the Table itself under **Data Column**. You can change these values and it will reflect on the Design space. Add and/or remove the headers accordingly with the buttons in the Data Column menu.

The screenshot shows a Figma workspace with a table component on the left and its properties panel on the right. The table has three columns: Name, Age, and Responsibility. The data rows are Wang Xiao (Age 15000, Responsibility develop), Wang Zhong (Age 25000, Responsibility product), and Wang Da (Age 35000, Responsibility design). The properties panel is titled 'Table' and contains tabs for Cell, Paragraph, Table, and Advanced. Under the Table tab, there are sections for Action column, Link, and Data column. The Data column section is expanded, showing a list of columns with their current titles: Name, Age, and Responsibility. Each column entry includes a pencil icon for editing and a three-dot menu icon. A tooltip 'Change the header labels in here' points to the edit icons, and another tooltip 'Use this to edit a column header' points to the 'Add an item' button. Other settings in the Data column include a checkbox for 'Data source' (set to 'No data'), sort options for 'Sort icon dev' (descending) and 'Sort icon eng' (ascending), and an 'Action column' section with a tooltip 'No content added yet'.

Recall that we will need to add headers for:

- Course Name
- Course Code
- Subject
- Enrollment Type
- Duration
- Schedule
- Price
- Students

You should have something like this:

The screenshot shows a table editor interface with a table containing three rows of course data. The columns are labeled: Course Name, Course Code, Subject, Enrollment Type, Duration, Schedule, Price, and Students. The rows contain the following data:

Course Name	Course Code	Subject	Enrollment Type	Duration	Schedule	Price	Students
Wang Xiao	15900	develop					
Wang Zhong	25900	product					
Wang Da	35900	design					

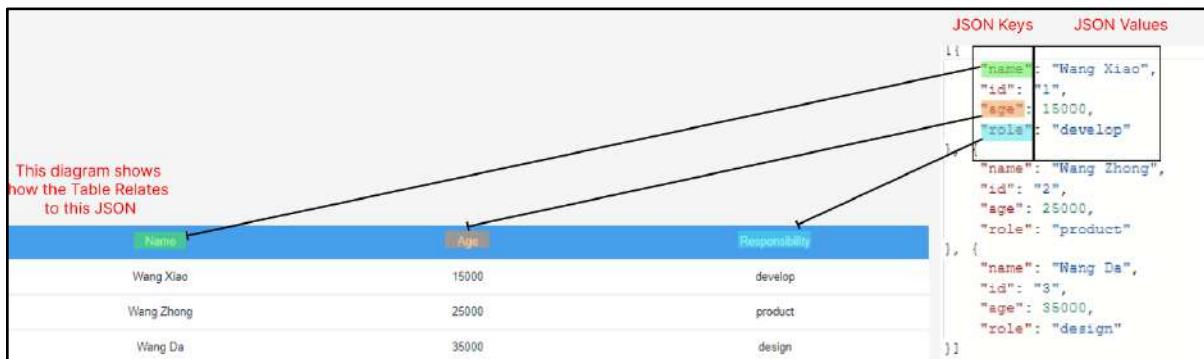
On the right side of the interface, there are several panels:

- Props**: Shows "No content added yet".
- Styles**: Shows "Add an item +".
- Events**: Shows "Link bar".
- Advanced**: Shows "No content added yet".
- Add an item +**: A button to add a new row.
- Data column**: A list of columns with edit icons and three-dot menus:
 - Course Name
 - Course Code
 - Subject
 - Enrollment Type
 - Duration
 - Schedule
 - Price
 - Students
- Edit data**: A button to edit the data source.
- Sort (course class)**: Set to "descending".
- Sort (course acc)**: Set to "descending".
- Action column**: A button to manage actions.

5.1.3 Adding a Static Dataset to the Table

Now that the table is set up, it is time to add the content to each row for testing.

The Table Component parses JSON and uses it to generate the data it displays on the rows. Each Key in a JSON object will correspond to the header in the table and each Value in a JSON object will be the row value.



We can add the row values through 3 different ways under the Props tab > Data Column of the Table Component:

- Click on the 'Edit Data' button. Modify the **JSON** accordingly.
- Add the JSON data** into the **Source Code Panel** in the Designer, then variable bind it.
- Retrieve** the data from an **external datasource**, like a backend server, then apply **variable binding** to it.

For today, we will only be covering (b) in detail:

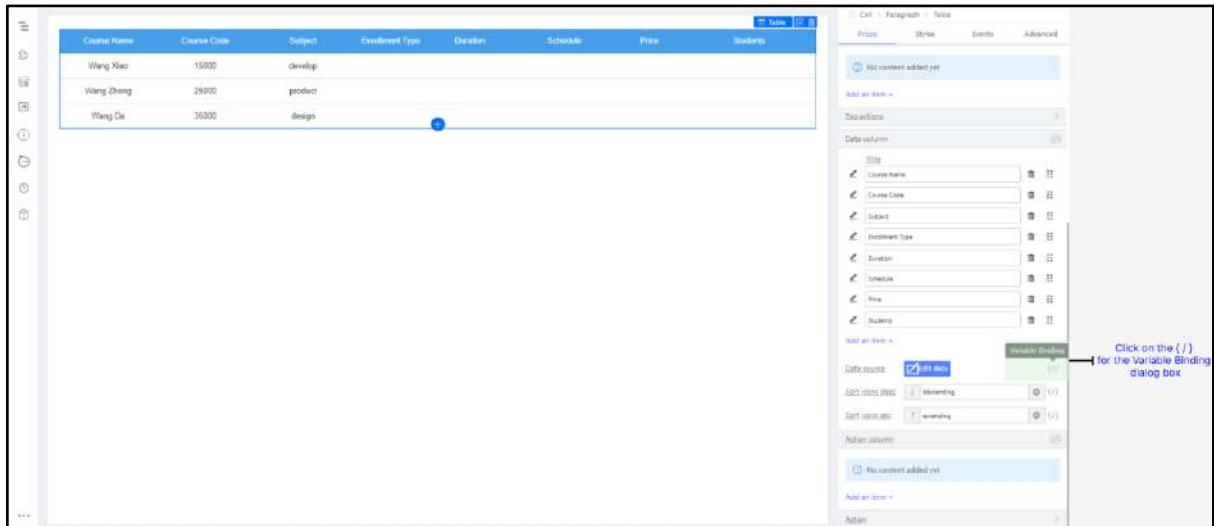
- Open the **Source Code Panel**. Copy the entire file from below:
[Tutorial5-code.txt](#)
- Click **Save** on the top-right of the Source Code Panel.

The Source Code Panel shows the `index.js` file with the following content:

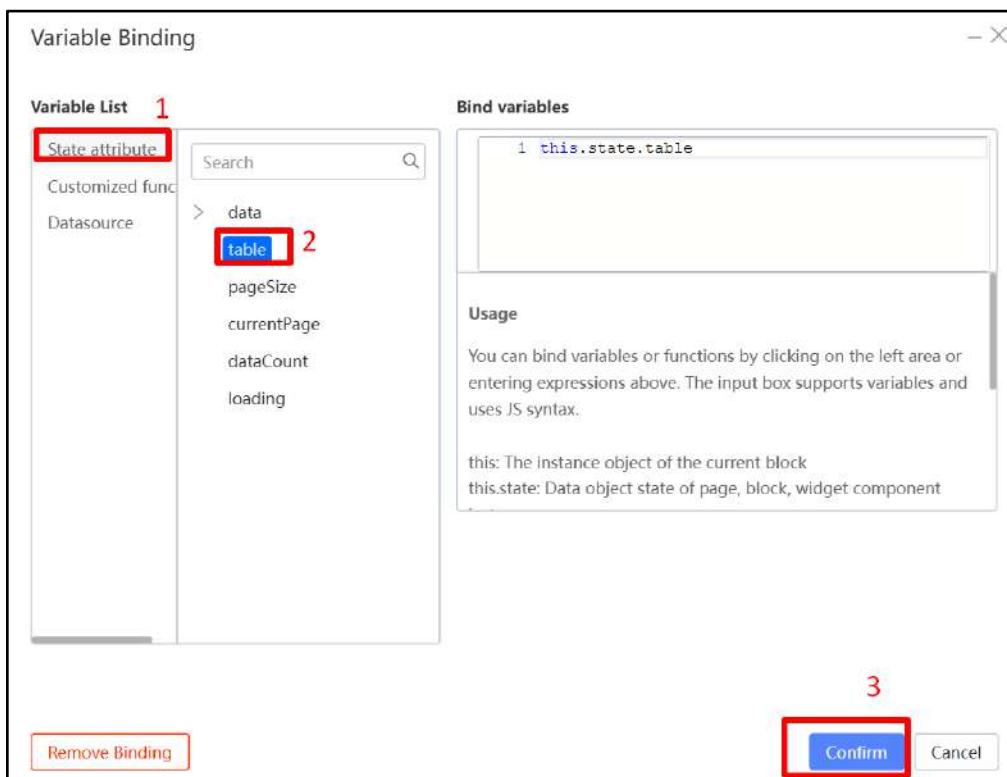
```
226  "pageSize": 10,
227  "currentPage": 1,
228  "dataCount": 100,
229  "loading": false
230
231
232 // ComponentDidMount is a function that is invoked after this page is lo
```

A red box highlights the **Save** button in the top-right corner of the panel.

- Select the Table Component and under Props tab > Data Column, click on the Variable Binding (the {}) for Data Source.



- Under the Variable List, select **State Attribute > table**. Click on **Confirm**



Notice the data in the table has changed.

The screenshot shows a table component editor interface. On the left is a table with columns: Course Name, Course Code, Subject, Enrollment Type, Duration, Schedule, Price, and Students. The rows contain course names like 'Financial Reporting and Analysis', 'Business Information Systems', etc. On the right is a sidebar with tabs for 'Props', 'Styles', 'Events', and 'Advanced'. Under 'Props' > 'Data column', there is a list of columns with edit icons: Course Name, Course Code, Subject, Enrollment Type, Duration, Schedule, Price, and Students. Below this is a 'Data source' section with an 'edit data' button and dropdowns for 'Sort items desc' (set to 'descending') and 'Sort items asc' (set to 'ascending').

You can see that the headers are there, but the data it is displaying doesn't look right. Let's figure out what's wrong:

- Select the Table Component and under Props tab > Data Column, Click on the Edit button on the 'Course Name' header.

The screenshot shows the configuration dialog for the 'Course Name' data column. It includes fields for Title (set to 'Course Name'), Data key (set to 'name'), Width (set to '200'), Data type (set to 'Text'), Lock (set to 'None'), Allow sorting (disabled), Cell (set to 'Bind function'), and Render (set to 'Param separated by commas').

Look at the **Data Key** field. This field takes the values of the key in a JSON object to put in the Table. So, this field must have exactly the same name as whatever the key is in our dataset.

Let's examine one of the entries in the data we put in the Source Code Panel earlier:

```
{  
  "id": "1",  
  "name": "Financial Reporting and Analysis",  
  "code": 4156,  
  "subject": "Accounting",  
  "enrollment-type": "Full-time",  
  "duration": "4 months",  
  "schedule-type": "weekly",  
  "price": 22000,  
  "student-no": 1325  
}
```

We can see that the key, 'name', in this JSON, corresponds with 'name' in our own 'Course Name Header'. So that explains why the Table has those rows under 'Course Name' populated even though we have not done anything yet.

Now that we have figured it out, all that's left to do is to change the rest of the headers.

- On Props tab > Data Column, edit each header's Data Key:

Header Title	Data Key
Course Name	name
Course Code	code
Subject	subject
Enrollment Type	enrollment-type
Duration	duration
Schedule	schedule-type
Price	price
Students	student-no

Click to bring up more options

Change the Data Key

Course Name	Course Code	Subject	Enrollment Type	Duration	Schedule	Price	Students
Financial Reporting and Analysis	4156	Accounting	Full-time	4 months	weekly	22000	1325
Business Information Systems	7234	Computing	Part-time	3 months	bi-weekly	8500	235
Strategic Design Thinking	6521	Design	Full-time	6 months	weekly	15000	245
Educational Leadership and Administration	4785	Education	Part-time	2 months	bi-weekly	3000	20
Business Ethics and Corporate Responsibility	8932	Business	Full-time	1 month	daily	2000	180
Management Accounting	2678	Accounting	Part-time	5 months	weekly	11000	125
Data Science for Business	5412	Computing	Full-time	4 months	bi-weekly	28000	150
Graphic Design Principles	6890	Design	Part-time	3 months	weekly	2000	90
Teaching Strategies for Diverse Learners	3245	Education	Full-time	2 months	bi-weekly	1500	20
International Business Management	9123	Business	Part-time	1 month	daily	10000	150

Once done, you should have a table like this:

Course Name	Course Code	Subject	Enrollment Type	Duration	Schedule	Price	Students
Financial Reporting and Analysis	4156	Accounting	Full-time	4 months	weekly	22000	1325
Business Information Systems	7234	Computing	Part-time	3 months	bi-weekly	8500	235
Strategic Design Thinking	6521	Design	Full-time	6 months	weekly	15000	245
Educational Leadership and Administration	4785	Education	Part-time	2 months	bi-weekly	3000	20
Business Ethics and Corporate Responsibility	8932	Business	Full-time	1 month	daily	2000	180
Management Accounting	2678	Accounting	Part-time	5 months	weekly	11000	125
Data Science for Business	5412	Computing	Full-time	4 months	bi-weekly	28000	150
Graphic Design Principles	6890	Design	Part-time	3 months	weekly	2000	90
Teaching Strategies for Diverse Learners	3245	Education	Full-time	2 months	bi-weekly	1500	20
International Business Management	9123	Business	Part-time	1 month	daily	10000	150

Practical 5.2: Table Sorting & Pagination

Now that we have the table set up, it's time to look at some other features we can add to this table, namely table sorting and pagination.

5.2.1 Setup

- Open the Source Code Panel. Uncomment the lines 253 - 273.

```
251  /** 5.2.1 **/  
252  // // Function to sort the column  
253  // onSort(dataIndex, order) {  
254  //   var dataSorted = this.state.data.sort(function (a, b) {  
255  //     var a;  
256  //     var b;  
257  //     if (typeof a[dataIndex] === "number") {  
258  //       a = a[dataIndex];  
259  //       b = b[dataIndex];  
260  //     }  
261  //     else {  
262  //       a = a[dataIndex].toString().toLowerCase();  
263  //       b = b[dataIndex].toString().toLowerCase();  
264  //     }  
265  //     return (order === 'asc') ? (a > b ? 1 : -1) : (a > b ? -1 : 1);  
266  //   });  
267  //   this.setState({  
268  //     data: dataSorted  
269  //   });  
270  //   this.loadTable(this.state.currentPage, this.state.pageSize);  
271  // }  
272  // }  
273  // }
```

The code block that you uncommented is for table sorting.

5.2.2 Sorting

In order to sort a table, we will first need to determine which data columns will be sortable and then set them as such in the Designer.

- The 4 columns that will be sortable are:
 - Course Name
 - Duration
 - Price
 - Students
- Under **Data Column**, Click on the **Edit** button beside the **Course Name** header.

The screenshot shows the Oracle Application Manager Designer interface. On the left, there is a list of items with frequencies: 1348, Schedule (highlighted), weekly, bi-weekly, weekly, bi-weekly, daily, weekly, bi-weekly, weekly, bi-weekly, daily. The 'Schedule' item is selected. On the right, the 'Props' tab is active, showing properties for the selected item:

- Title: Course Name
- Data key: name
- Width: 200
- Data type: Text
- Lock: Left, Right, None
- Allow sorting: A green switch is turned on, with the text "Enable Sorting here" displayed below it.
- Cell: Bind function
- Render: Enable (switch is on), Param separated by commas

A blue callout box with the text "Click to bring up more options" points to the "More Options" icon (three dots) next to the Render setting.

On the far right, under "Data column", there is a list of columns with edit icons and delete icons:

- Course Name
- Course Code
- Subject
- Enrollment Type
- Duration
- Schedule
- Price
- Students

At the bottom right of the interface, there are buttons for "Edit data" and a refresh icon.

- Enable **Allow Sorting**.
- Repeat the same steps for **Duration**, **Price** and **Students**

Note in the Table in the Designer's display area has changed the column to have an up/down arrow beside the headers you have changed. Clicking on it does nothing for now. We will need to do some event binding.

- Under the **Events** tab, click on the **Component native event** button.

The screenshot shows the Figma Designer interface. On the left, there is a table component with two columns: "Price" and "Students". The table has ten rows of data. On the right, the "Table" component is selected in the "Cell" > "Paragraph" > "Table" hierarchy. The "Events" tab is active in the top navigation bar. In the sidebar, under the "Events" section, there is a button labeled "Component native event". A blue callout with the text "Click on this button" points to this button. Below it, there is a table with two columns: "Existing event" and "Action", which currently displays "No Data".

Price	Students
22000	1325
8500	235
15000	245
3000	20
2000	180
11000	125
28000	150
2000	90
1500	20
10000	150

Cell > Paragraph > Table

Props Styles Events Advanced {/}

Click to bind event

Component native event

Existing event Action

No Data

Click on this button

- In the dropdown, **select** the **onSort** event.

The screenshot shows a UI builder interface with a table component. The table has two columns: "Price" and "Students". The "Price" column is sorted in descending order (indicated by a downward arrow), and the "Students" column is sorted in ascending order (indicated by an upward arrow). The table contains ten rows of data. To the right of the table, there is a sidebar with the following structure:

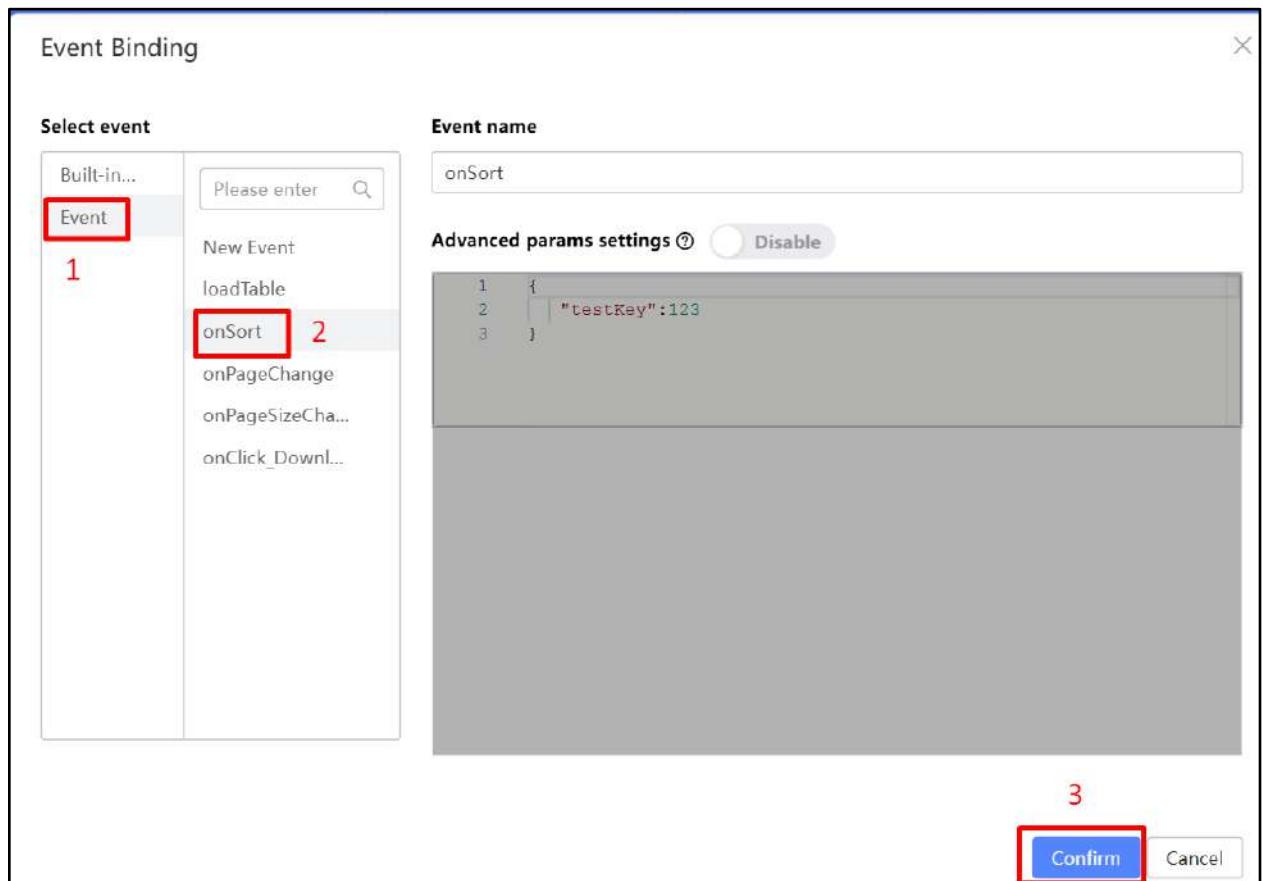
- Cell > Paragraph > Table
- Props
- Styles
- Events** (selected)
- Advanced

The "Events" section contains a list of native component events:

- onFetchData
- onSelect**
- onSelectAll**
- onSort** (highlighted with a green background and a blue arrow pointing to it)
- onRowClick
- onRowMouseEnter
- onRowMouseLeave
- onRowOpen

Price	Students
22000	1325
8500	235
15000	245
3000	20
2000	180
11000	125
28000	150
2000	90
1500	20
10000	150

- In the popup, select **Event**, click on **onSort**. Then click on the **Confirm** button.



We should now be able to sort by clicking on the headers in each column.

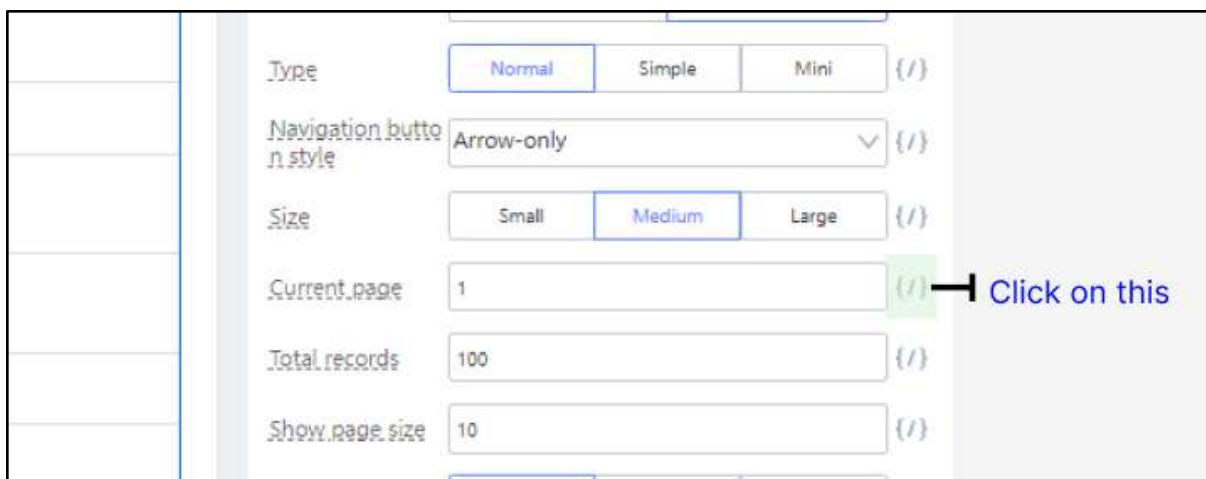
5.2.3 Pagination

Something you probably noticed is that the table is only displaying 10 entries even though the dataset has 22 entries in total. That's because the Table and the code placed in the Source Code Panel has catered for pages in the table. However, there is no way to navigate the pages or change the page size. Let's change this.

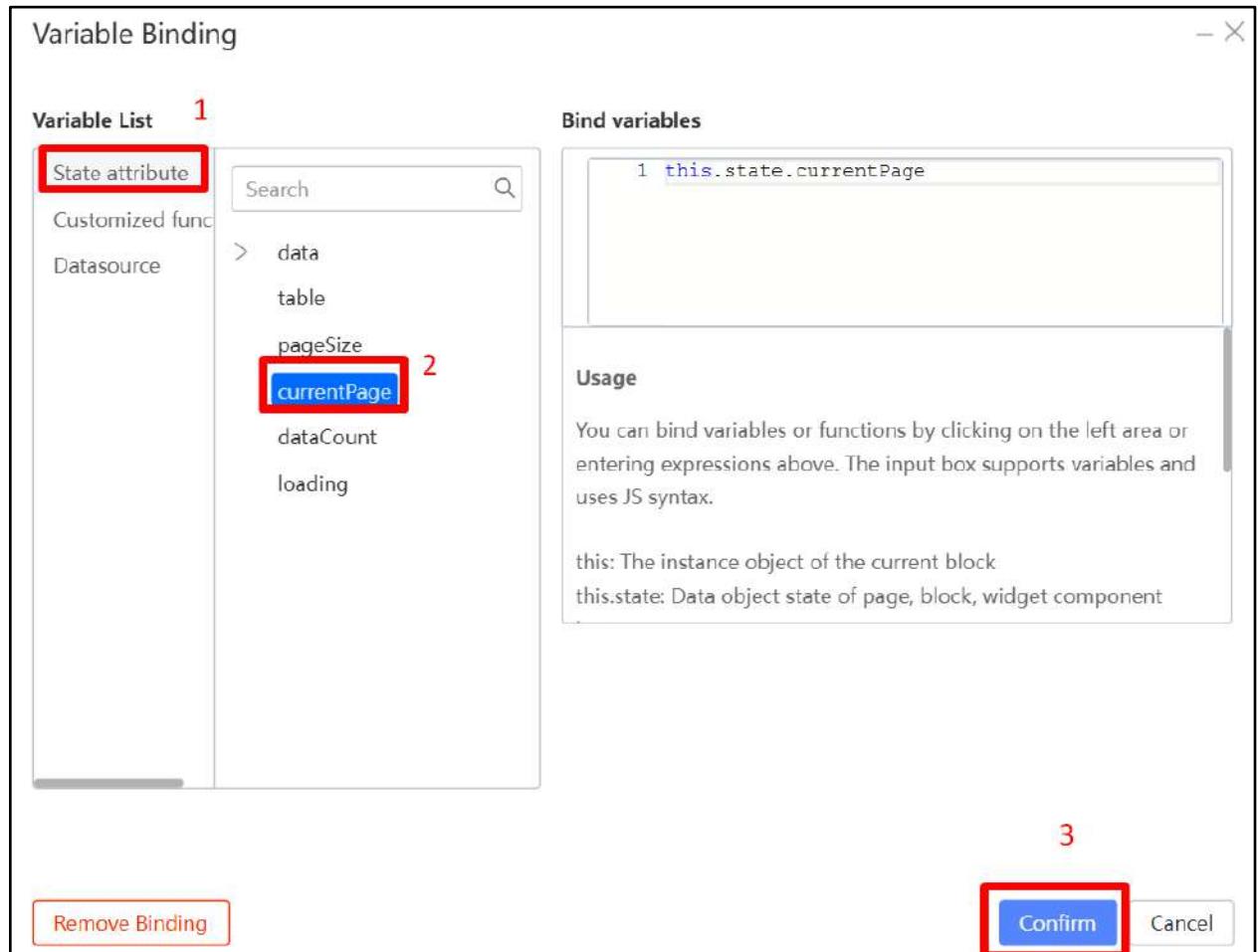
- Open the Source Code Panel. Uncomment lines 277 - 282

```
275  /** 5.2.3 **/  
276  // Function to change the page of the table  
277  // onPageChange(curPage) {  
278  //   this.loadTable(curPage, this.state.pageSize);  
279  //   this.setState({  
280  //     currentPage: curPage  
281  //   });  
282  // }  
283
```

- Under the **Pagination Configuration**, click on the **Variable Binding** for field **Current Page**



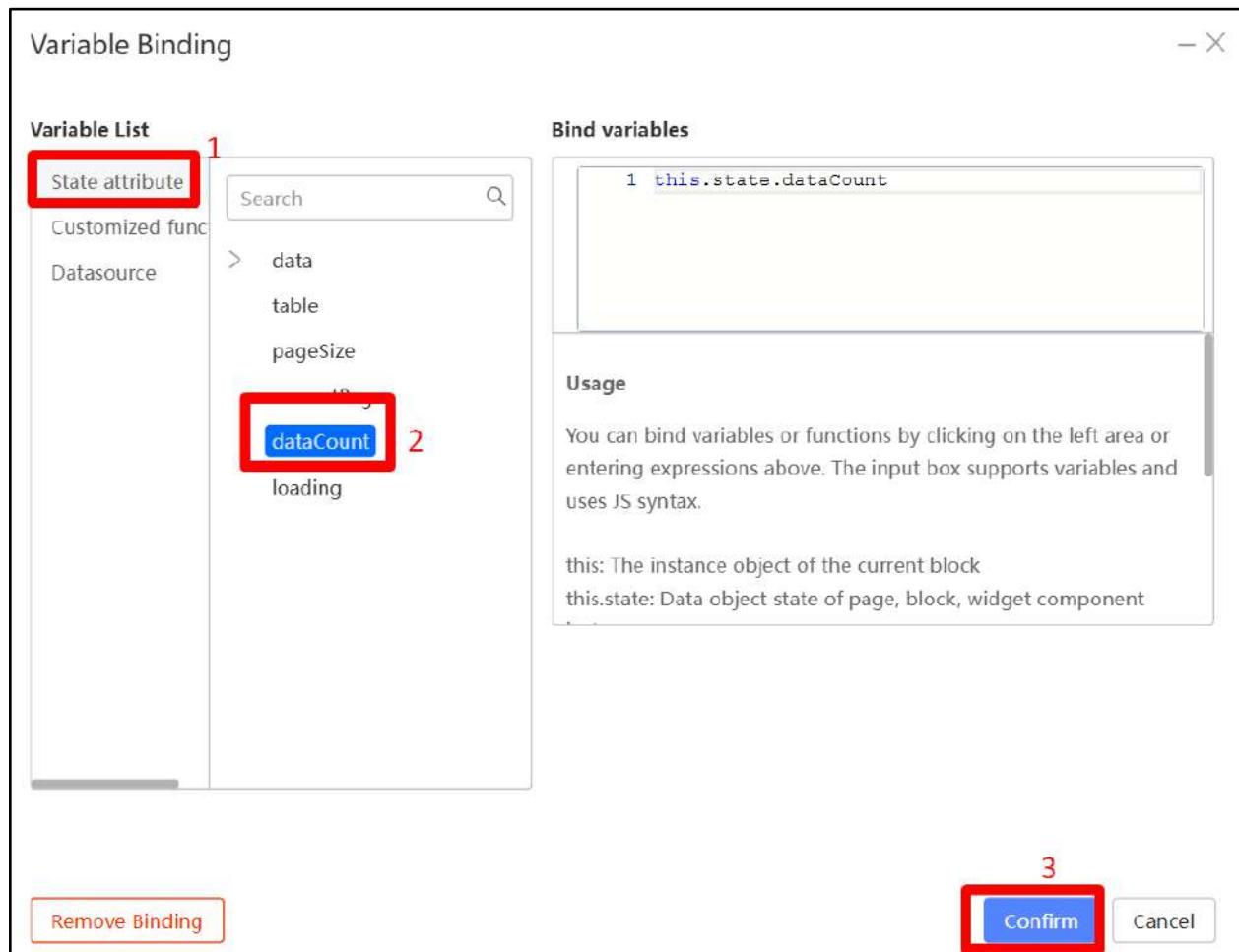
- Under Variable List, Select 'State attribute' then select 'currentPage'.



- Still under the Pagination Settings, click on the Variable Binding for ‘Total records’.

<div style="border: 1px solid #ccc; padding: 5px; height: 300px; overflow-y: auto;"> <p>235</p> <p>1480</p> <p>150</p> <p>20</p> <p>70</p> <p>350</p> <p>1325</p> <p>90</p> <p>290</p> </div>	<p>Pagination settings</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Alignment <input type="button" value="Left"/> <input checked="" type="button" value="Right"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Type <input checked="" type="button" value="Normal"/> <input type="button" value="Simple"/> <input type="button" value="Mini"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Navigation button style <input type="button" value="Arrow-only"/> <input type="button" value="Page numbers"/> <input type="button" value="Both"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Size <input type="button" value="Small"/> <input checked="" type="button" value="Medium"/> <input type="button" value="Large"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Current page <input type="text" value="1"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Total records <input type="text" value="100"/> <input type="button" value="Bind function"/> Click on this </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Show page size <input type="text" value="10"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Selector type <input checked="" type="button" value="None"/> <input type="button" value="Filter"/> <input type="button" value="Dropdown"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Page size list <input type="text" value="5,10,20"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Page size alignment <input type="button" value="Start"/> <input checked="" type="button" value="End"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Single page hide <input checked="" type="checkbox"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Show jump <input checked="" type="checkbox"/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> Jump link <input type="text" value=""/> <input type="button" value="Bind function"/> </div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> OnChange <input type="button" value="Bind function"/> </div>
---	---

- Under Variable List, Select 'State attribute' then select 'dataCount'.



- Still under the Pagination Settings, click on the Variable Binding for ‘Show page size’.

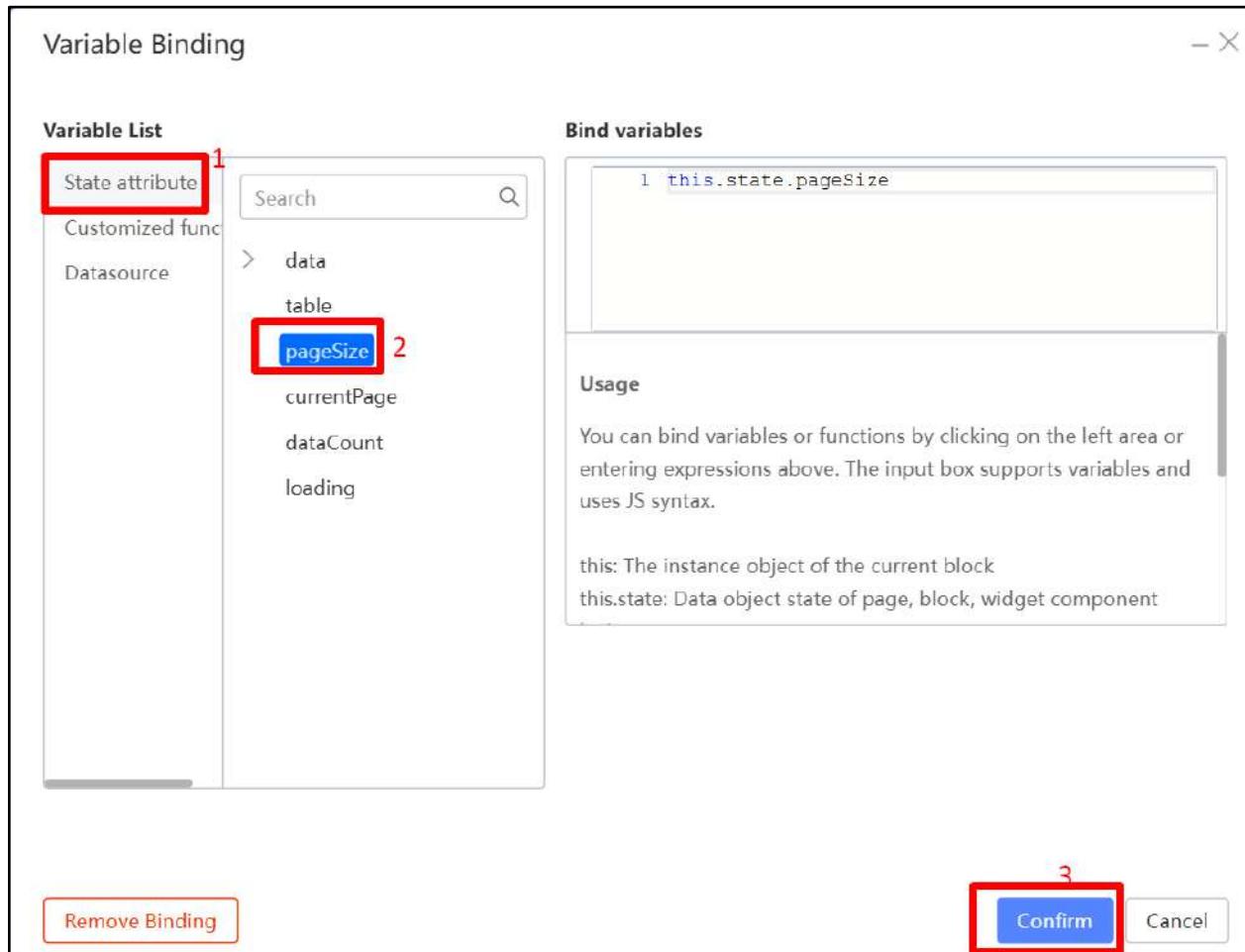
The screenshot shows the configuration interface for the Paginate component. On the left, there is a sidebar with the following values listed vertically:

- 1480
- 150
- 20
- 70
- 350
- 1325
- 90
- 290

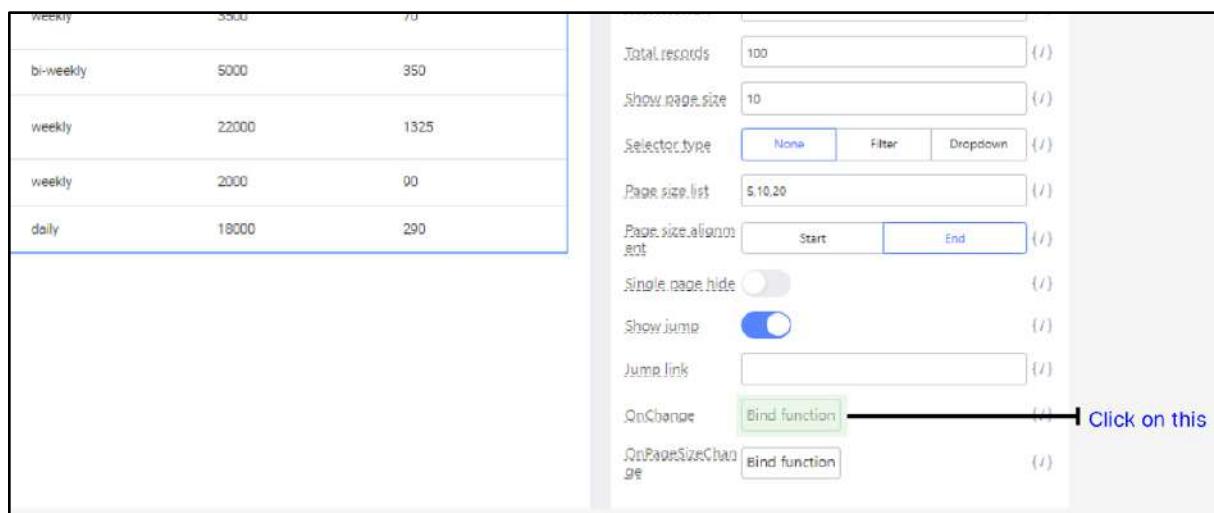
The main configuration area contains the following settings:

- Type: Normal (selected)
- Navigation button style: Arrow-only
- Size: Medium (selected)
- Current page: 1
- Total records: 100
- Show page size: 10 (highlighted with a green background and a blue arrow pointing to it)
- Selector type: None (selected)
- Page size list: 5,10,20
- Page size alignment: Start
- Single page hide: Off
- Show jump: On
- Jump link: (empty input)
- OnChange: Bind function
- OnPageSizeChange: Bind function

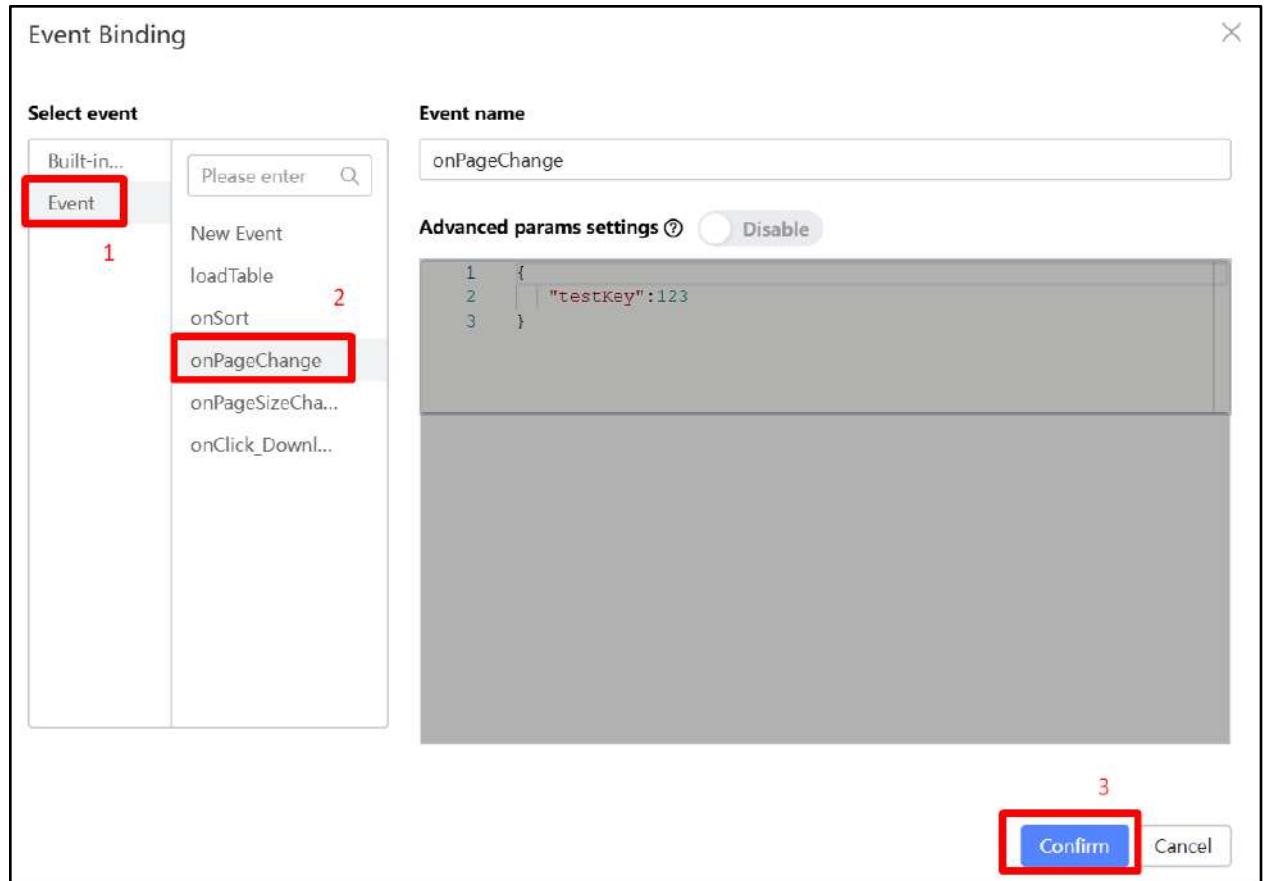
- Under Variable List, Select 'State attribute' then select 'pageSize'.



- In the **OnChange** field, click on the **Bind Function** button.



- Under **Event**, click on the **onPageChange** event.



In **Preview** mode, now you can click on the pages at the bottom-right of the table and each page should display each page set of data correctly.

Course Name	Course Code	Subject	Enrollment Type	Duration	Schedule	Price	Students
Leadership in Higher Education	7891	Education	Fulltime	2 months	weekly	10000	40
Human Resource Law	4562	Business	Part-time	1 month	daily	18000	290

5.2.4 Page Sizing

Let's continue to add a Page Size feature.

- Open the Source Code Panel. Uncomment lines 286 - 293.

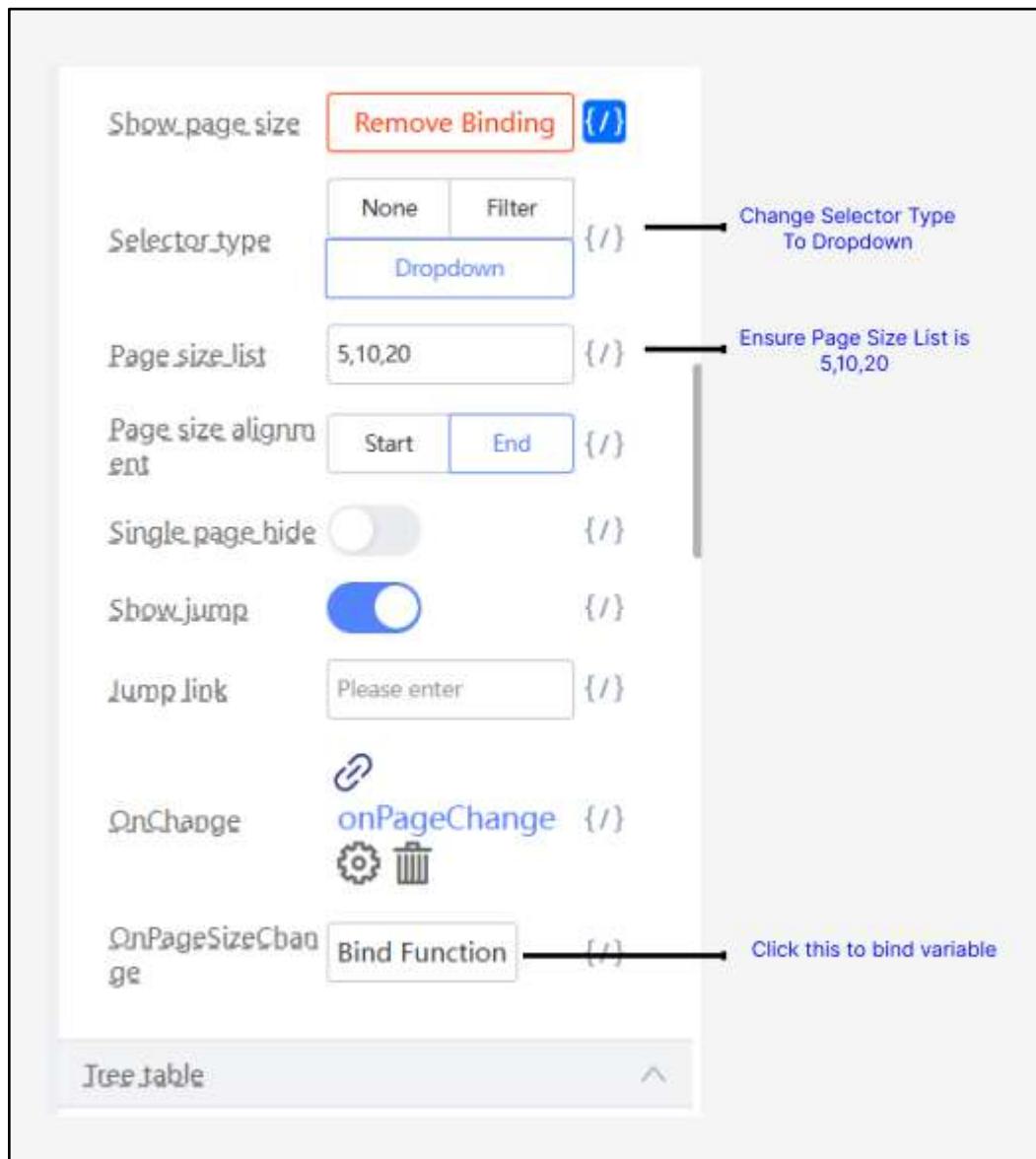
```
284  /** 5.2.4 **/
285  // Function to change the page size of the table
286  // onPageSizeChange(size) {
287  //   // reset to page 1
288  //   this.loadTable(1, size);
289  //   this.setState({
290  //     currentPage: 1,
291  //     pageSize: size
292  //   });
293  // }
```

- In the **Selector type** field, choose ‘**Dropdown**’.

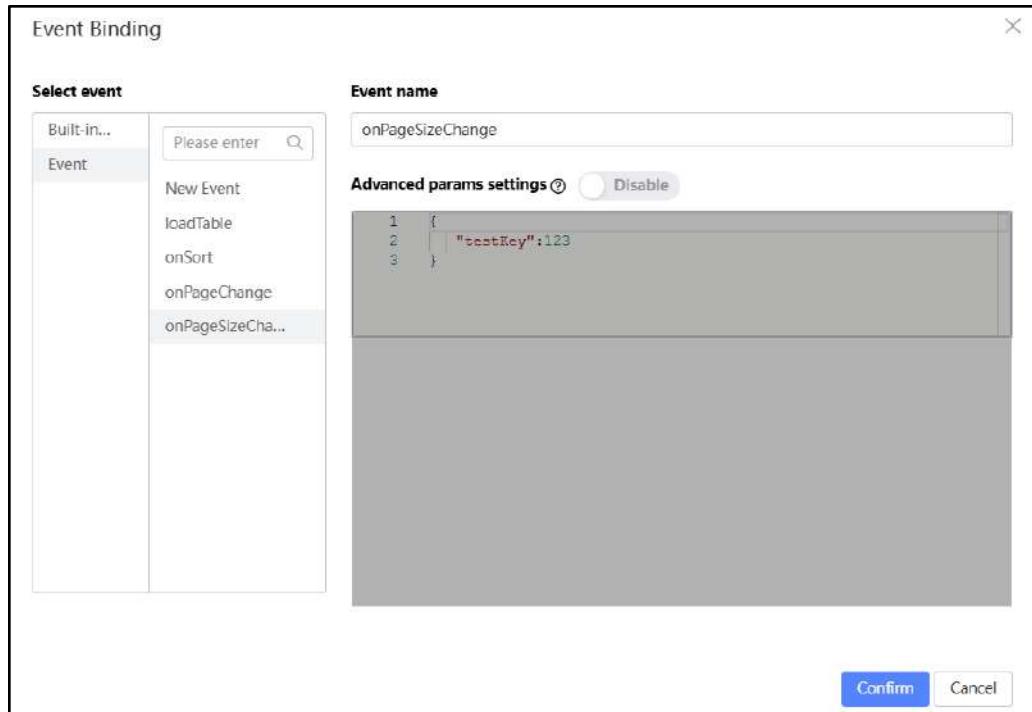
The screenshot shows the configuration interface for a component named 'Table2'. On the left, there is a preview of the component displaying a table with two columns and several rows of data. The first column contains values '24000' and '32000'. The second column is labeled 'Enrollment Type' and contains alternating 'Full-time' and 'Part-time' entries. On the right, the 'Props' tab of the configuration panel is selected, showing various settings for the component. One of these settings is 'Selector type', which has a dropdown menu open, with the option 'Dropdown' highlighted and surrounded by a red box. Other options in the dropdown include 'None' and 'Filter'. The other settings visible in the props panel include Alignment (Left or Right), Type (Normal, Simple, or Mini), Navigation button style (Arrow-only), Size (Small, Medium, or Large), Current page, Total records, Total text, Show page size, Selector type, Page size list (containing '5,10,20'), and Page size alignment (Start or End).

- Page size list should be ‘5,10,20’

- In the OnPageSizeChange field, click on the ‘Bind Function’ button.



- Under ‘Select event’, Click on ‘Event’, then click on ‘OnPageSizeChange’.



- Click on the '**Confirm**' button.

These are the options you should have:

Cell4 > Paragraph5 > Table6

Props	Styles	Events	Advanced
Pagination configuration			
Show pagination	<input checked="" type="checkbox"/>	{/}	
Pagination settings			
Alignment	Left	Right	{/}
Type	Normal	Simple	Mini
Navigation button style	Arrow-only	{/}	
Size	Small	Medium	Large
Current page	1	{/}	
Total records	100	{/}	
Show page size	10	{/}	
Selector type	None	Filter	Dropdown
Page size list	5,10,20	{/}	
Page size alignment	Start	End	{/}
Single page hide	<input type="checkbox"/>	{/}	
Show jump	<input checked="" type="checkbox"/>	{/}	
Jump link	{/}		
OnChange	onPageChange	⚙️	✖️
OnPageSizeChange	onPageSizeChange	⚙️	✖️

Now you can Preview to check all the changes that we added.

Practical 5.3: Download functionality (Optional)

Sometimes there is a need for the user to download files or data from your end. It is possible to do this with the Designer's Source Code Panel. In this Practical, we will add a button that allows users to download the JSON data of the table we did in the previous Practical.

5.3.1 Add a download button

- Add a block to the bottom of the table.

ng	Part-time	5 months	weekly
ng	Full-time	4 months	bi-weekly
	Part-time	3 months	weekly
n	Full-time	2 months	bi-weekly
s	Part-time	1 month	daily

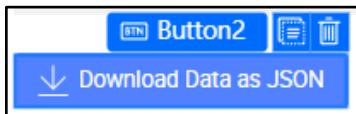
+
Add Block Below

- Drag a Button Component to Cell.

The screenshot shows a course management application interface. On the left, there is a sidebar titled 'Button' with a tree view. The 'General' node is expanded, showing 'My Button' (selected), 'Menu Button', and 'Button Group'. A green arrow points from the 'My Button' node to a table cell in the bottom right corner. The main area displays a table of courses with columns: Course Name, Course Code, Subject, Enrollment Type, Duration, Schedule, Price, and Students. The table has 22 rows. At the bottom right of the table, there is a toolbar with icons for 'Cell', 'Row', and 'Column' operations, and a message: 'Drag and drop components or templates here'.

Course Name	Course Code	Subject	Enrollment Type	Duration	Schedule	Price	Students
Financial Reporting and Analysis	4156	Accounting	Full-time	4 months	weekly	22000	1325
Business Information Systems	7234	Computing	Part-time	3 months	bi-weekly	8500	235
Strategic Design Thinking	6521	Design	Full-time	6 months	weekly	15000	245
Educational Leadership and Administration	4785	Education	Part-time	2 months	bi-weekly	9000	20
Business Ethics and Corporate Responsibility	8932	Business	Full-time	1 month	daily	2000	180
Management Accounting	2678	Accounting	Part-time	5 months	weekly	11000	125
Data Science for Business	5412	Computing	Full-time	4 months	bi-weekly	28000	150
Graphic Design Principles	6890	Design	Part-time	3 months	weekly	2000	90
Teaching Strategies for Diverse Learners	3245	Education	Full-time	2 months	bi-weekly	1500	20
International Business Management	9123	Business	Part-time	1 month	daily	10000	150

- Do some styling for the button and rename it to ‘Download Data as JSON’.



5.3.2 Add the download code

- Open the Source Code Panel. Uncomment the lines 297 - 301.

```

295  /**
296   * Function to download the table data as a json file
297   */
298   // onClick_Download() {
299   //   var payload = JSON.stringify(this.state.data, null, 2);
300   //   var fileName = "Courses.json";
301   //   this.utils.FileSaver.saveAs(new Blob([payload], { type: 'application/json' }), fileName)
302 }

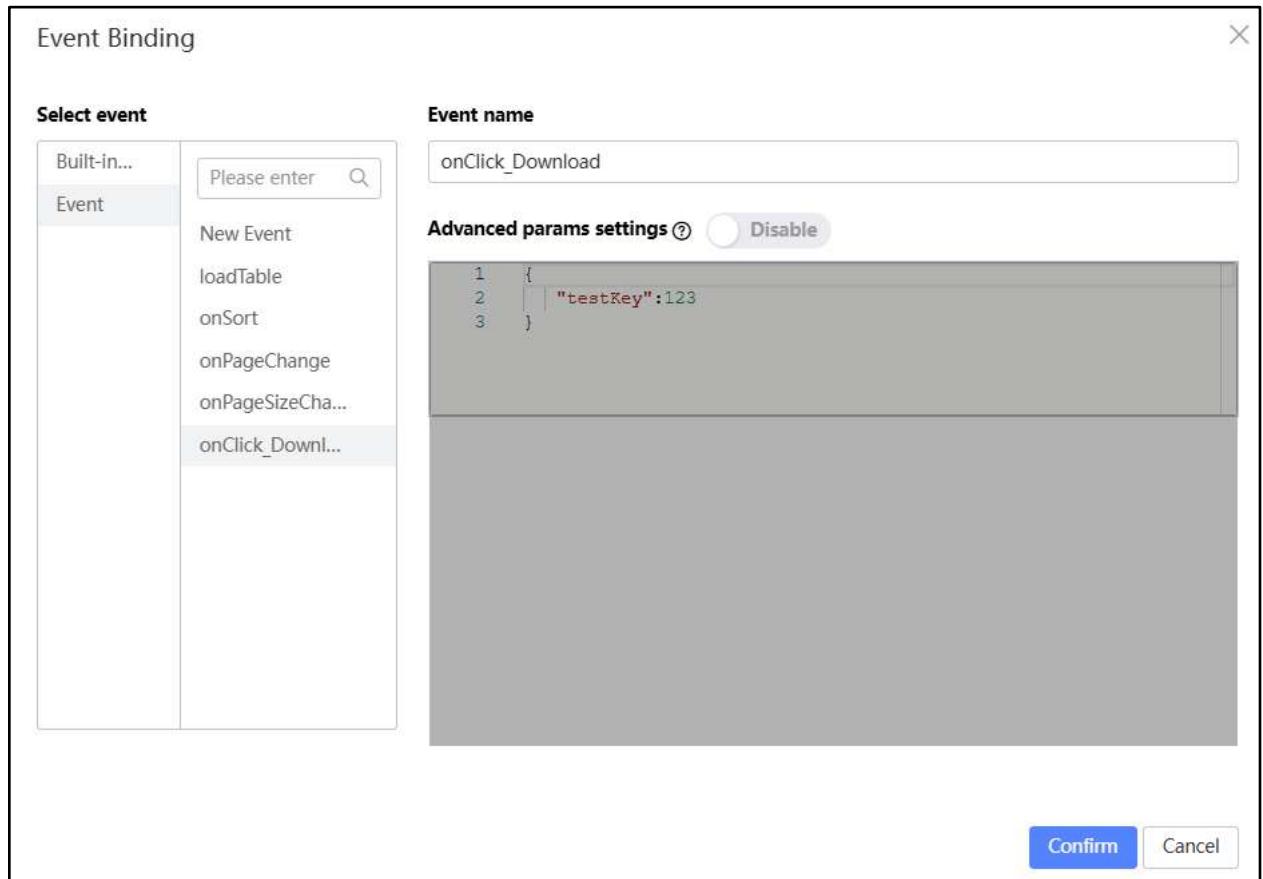
```

5.3.3 Event bind the download button

- Click on the Button Component and go to the Events tab and click on the ‘Component native event’ button. In the dropdown, click on ‘OnClick’.

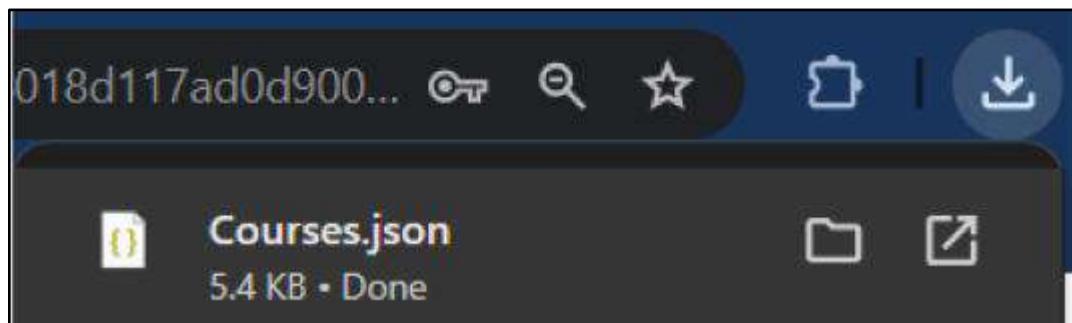
Course Name	Course Code	Subject	Enrollment Type	Duration	Schedule	Price	Students
Financial Reporting and Analysis	4133	Accounting	Full-time	4 months	weekly	22000	122
Business Information Systems	7234	Computer	Part-time	3 months	bi-weekly	8000	231
Strategic Design Thinking	3321	Design	Full-time	6 months	weekly	15000	345
Educational Leadership and Administration	4716	Education	Part-time	2 months	bi-monthly	3000	30
Business Ethics and Corporate Responsibility	8472	Business	Full-time	1 month	daily	2000	180
Management Accounting	2478	Accounting	Part-time	5 months	weekly	11000	125
Data Science for Business	5412	Computer	Hybrid	4 months	bi-weekly	25000	190
Graphic Design Principles	5880	Design	Part-time	3 months	weekly	2000	90
Teaching Strategies For Diverse Learners	3245	Education	Ad-hoc	2 months	bi-monthly	1500	20
International Business Management	9121	Business	Part-time	1 month	daily	18000	190

- Under 'Select event', Click on 'Event', then click on 'OnClick_Download'.



- Click on the 'Ok' button.

Now, whenever you click on the Download button, you will start downloading a JSON file with the table's data.



Tutorial 6: Charts and Graphs

This tutorial covers the following Learning Objectives:

- Understand how to integrate and display various types of charts and graphs in KAIZEN.
- Learn how to visualize data effectively using different chart types like bar and pie charts.
- Explore how to bind data dynamically to charts for real-time updates.

In this tutorial, you will learn how to create and display charts and graphs in KAIZEN to visualize data effectively. We'll guide you through selecting the right chart types and binding them to dynamic data, enabling real-time updates. By the end of this tutorial, you'll be able to enhance your application with informative and interactive visualizations that help users interpret data quickly and easily.

Practical 6.1: Adding Graph Components

6.1.1 Setup Steps

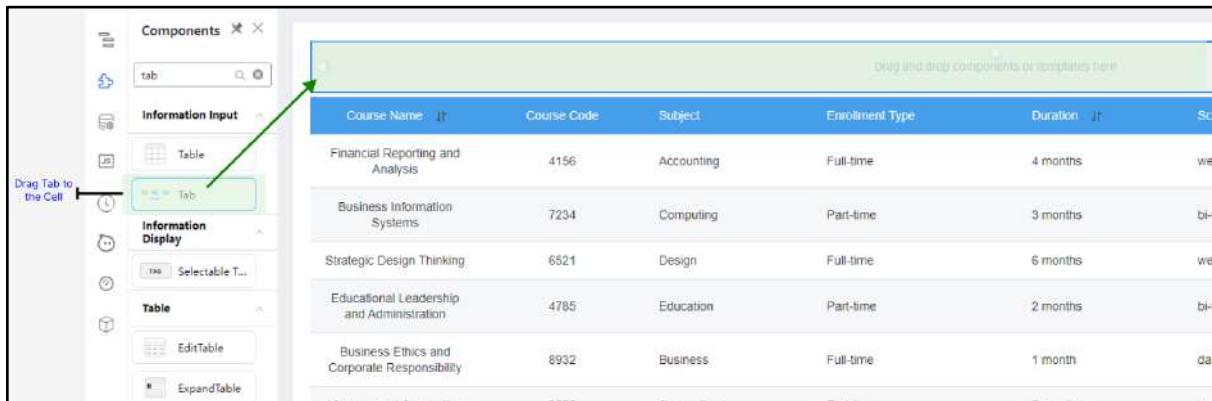
- From the Component Library, drag a Cell Component to the top of the Table in the Course Table Page that you have created from Tutorial 5.



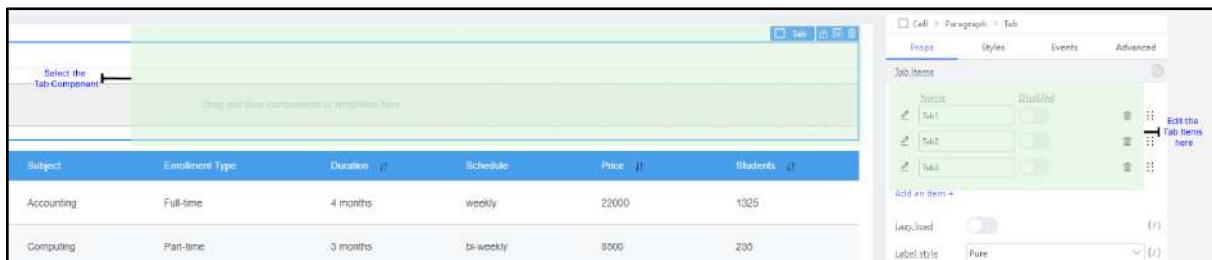
The screenshot shows the KAIZEN interface. On the left, the 'Components' panel is open, displaying a list of components including 'Cell', 'Layout Container', and others. A green arrow points from the 'Cell' component icon in the library to the top row of a table on the main workspace. The table has columns for Course Name, Course Code, Subject, Enrollment Type, Duration, and Schedule. The first few rows list courses such as 'Financial Reporting and Analysis', 'Business Information Systems', 'Strategic Design Thinking', etc. A tooltip 'Drag the Cell above the Items per Page' is visible near the component library.

Course Name	Course Code	Subject	Enrollment Type	Duration	Schedule
Financial Reporting and Analysis	4156	Accounting	Full-time	4 months	week
Business Information Systems	7234	Computing	Part-time	3 months	bi-weekly
Strategic Design Thinking	6521	Design	Full-time	6 months	week
Educational Leadership and Administration	4788	Education	Part-time	2 months	bi-weekly
Business Ethics and Corporate Responsibility	8932	Business	Full-time	1 month	daily
Management Accounting	2678	Accounting	Part-time	5 months	week
Data Science for Business	5412	Computing	Full-time	4 months	bi-weekly
Graphic Design Principles	6890	Design	Part-time	3 months	week
Teaching Strategies for Diverse Learners	3245	Education	Full-time	2 months	bi-weekly
International Business					

- Drag a Tab Component into the Cell Component.



- Rename the tabs (and remove any extra) so that there are only these 2 tabs: 'Price Comparison' and 'Enrollment'.



Once you are done, you should have something like this:



6.1.2 Adding a Bar Graph

- Select the 'Price Comparison' tab in the Tab Component.
- From the Component Library, search for 'chart' and drag a Bar Chart Component (aka Column Component) into the Price Comparison Tab.

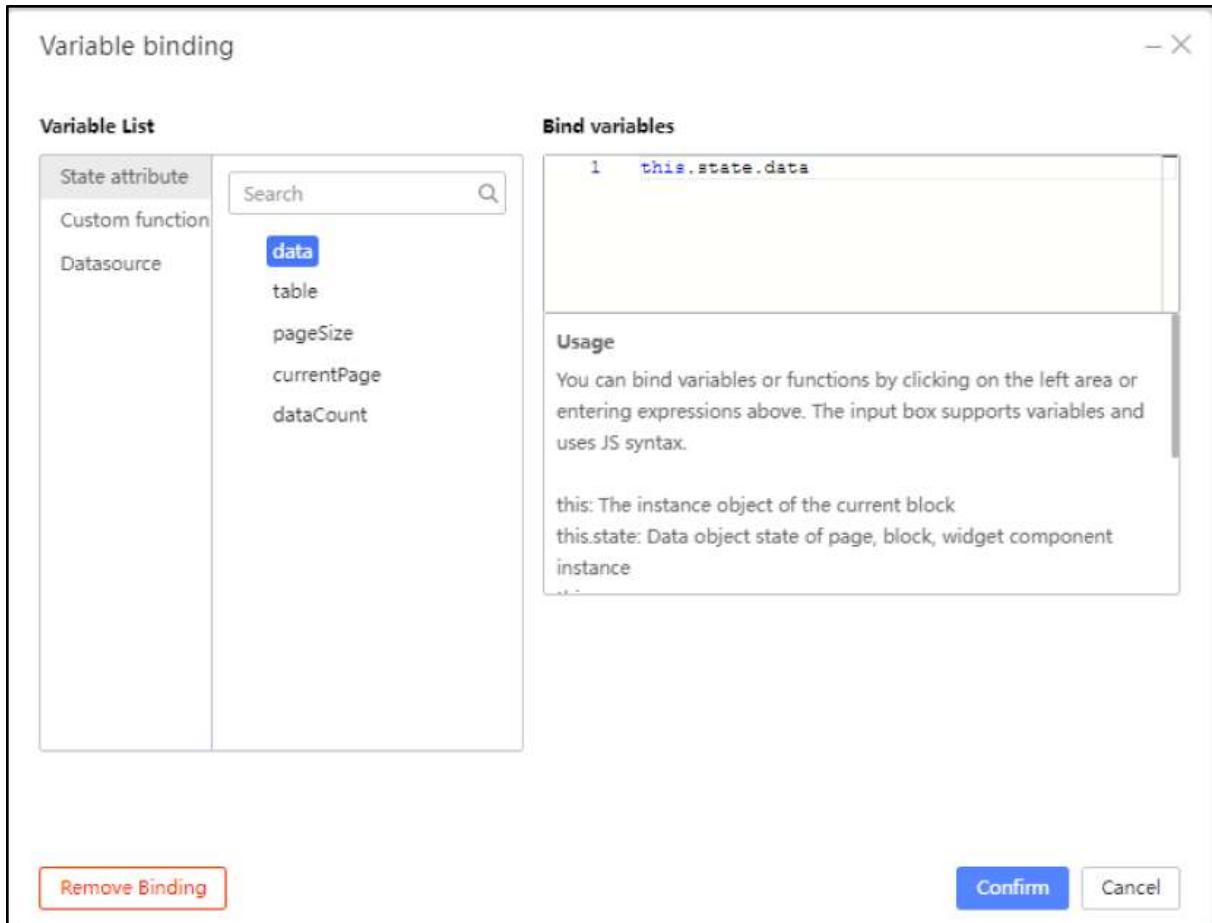
The screenshot shows the DataGrip interface. On the left, the 'Components' palette is open, displaying various chart types: Bar, Line, Area, and Scatter. A green arrow points from the 'Bar' component towards the main workspace. The main workspace contains a table titled 'Enrolment' with columns: ID, Course Code, Subject, Enrollment Type, Duration, Schedule, Price, and Student Count. The table has 11 rows of data. At the bottom right of the table, there are navigation buttons for pages 1 through 10 and a 'Print current page' button.

6.1.3 Sending Data to the Graph

- Select the Column Component and under the Props Tab, click on Variable Binding.

The screenshot shows the DataGrip interface with a bar chart titled 'Bar16' displayed. The chart has blue bars representing data for years 1991 to 2000. The x-axis ranges from 0 to 80,000,000. The y-axis lists the years. The chart's properties panel on the right is open, showing the 'Props' tab selected. Under 'Chart data', there is a 'Edit data' button and a 'Click this' link. The 'x-axisLabel' is set to 'value' and 'y-axisLabel' is set to 'year'. The 'Color' is set to '#0079F2'. The 'Display' section has a toggle switch turned on and a 'Label' dropdown with options 'Left' and 'Middle'.

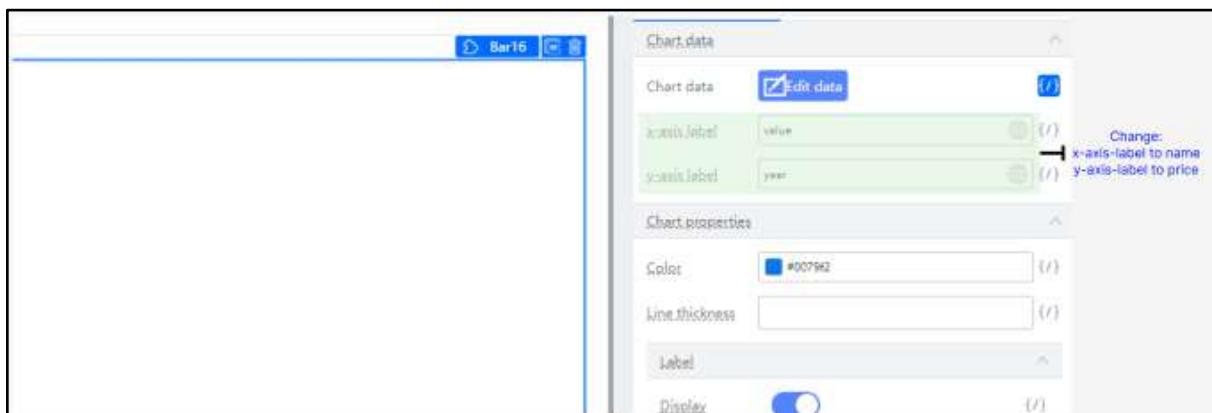
- Under 'Variable List', Select 'State attribute', then select 'data'.



- Click the 'Confirm' button.

Note the graph has disappeared.

- Change the 'x-axis label' to **price** and the 'y-axis label' to **name**.



- With the Bar Chart Component still selected, go to the Styles tab and change the Height to 600px.

If you click on Preview, you can hover over each column to see their specific x-axis labels.

6.1.4 Adding a Pie Chart

- Select the ‘Enrollment’ tab in the Tab Component.
- From the Component Library, search for ‘chart’ and drag a Pie Chart Component into the Enrollment Tab.

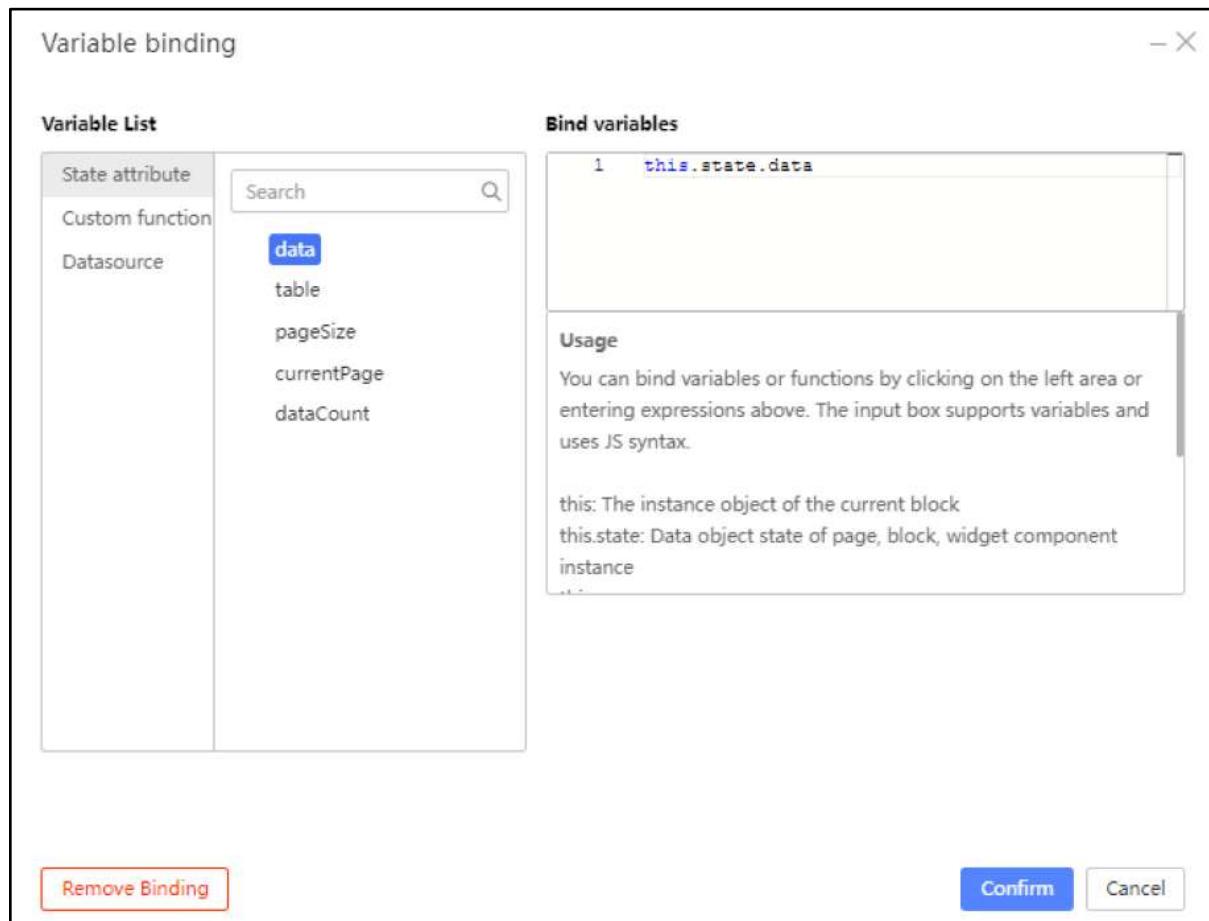
The screenshot shows the Marmoset interface. On the left, there's a sidebar titled 'Components' with a search bar and a list of chart types: Area, Bar, Column, Donut, and Line. Below these is a 'Pie' icon. A green arrow points from this icon towards a table component in the main area. The table has columns for Course Name, Course Code, Subject, Enrollment Type, Duration, Schedule, Price, and Students. The table lists various courses like 'Machine Learning for Business', 'Data Science for Business', etc., with their respective details.

6.1.5 Sending Data to the Graph

- Select the Pie Chart Component and under the Props Tab, click on Variable Binding.

The screenshot shows the Marmoset interface with a pie chart component. The chart has several slices with labels: 72345678, 4321132, 331211123, 45227721, 432122519, 6322121, 78312213, 2193212, 6213352, and 1183212. To the right is the 'Props' tab of the component's properties. Under the 'Chart data' section, there is a 'Edit data' button and fields for 'source.field.name' (set to 'value') and 'Category.field.name' (set to 'year'). A green arrow points to the 'Edit data' button with the text 'Click this'.

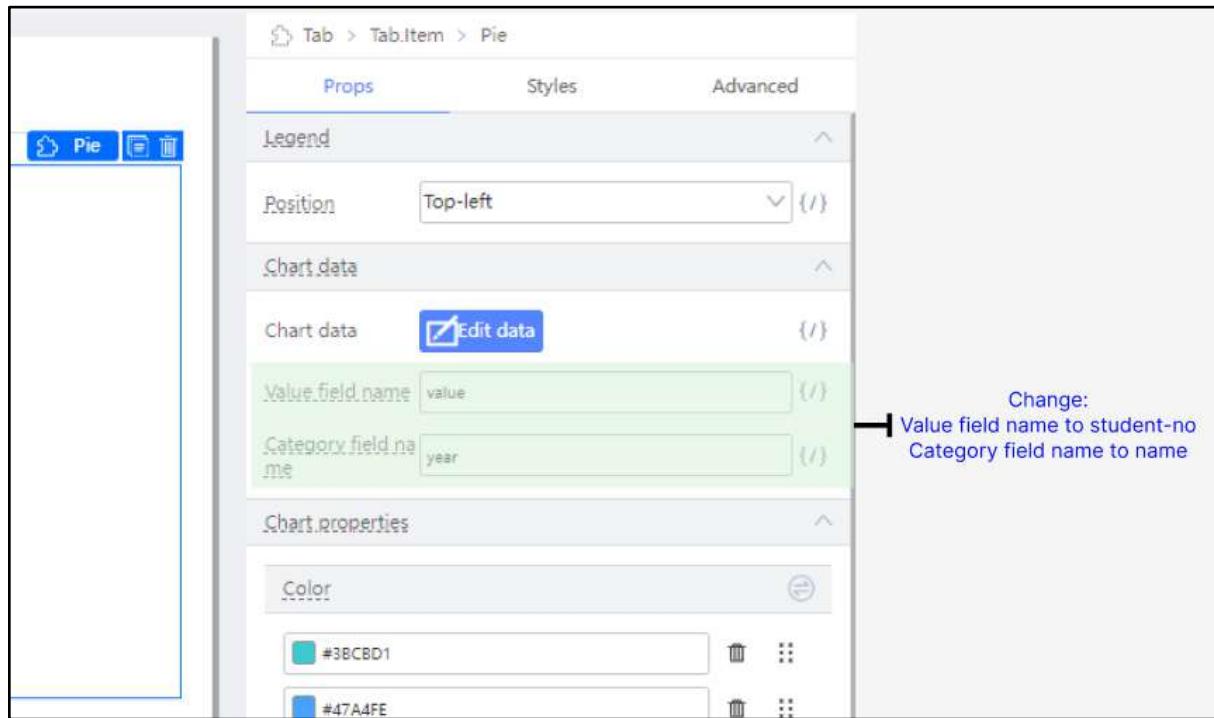
- Under 'Variable List', Select 'State attribute', then select 'data'.



- Click the 'Confirm' button.

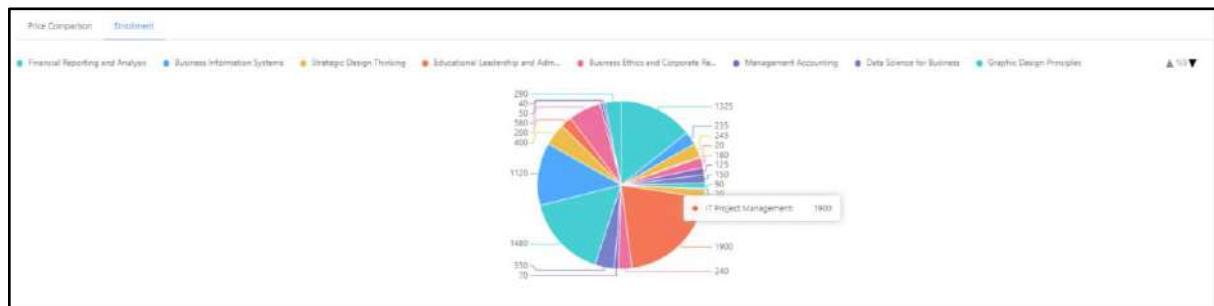
Note the graph has disappeared.

- Change the 'Value field name' to **student-no** and the 'Category field name' to **name**.



- With the Pie Chart Component still selected, go to the Styles tab and change the Height to 600px.

If you click on Preview, you can hover over each column to see their specific category names. You can also hover over each item in the legend to highlight their position in the pie chart.



In the Legend, you can also click on an item to deselect it and prevent it from showing up in the Pie Chart.

Tutorial 7: Using External JS Libraries

This tutorial covers the following Learning Objectives:

- Understand how to integrate external JavaScript libraries into your KAIZEN application.
- Learn how to enhance application functionality by leveraging popular libraries like `uuid.js`, and others.
- Explore the process of importing and utilizing external libraries to extend the capabilities of your application.

In this tutorial, you will learn how to integrate external JavaScript libraries into your KAIZEN application to enhance its functionality. We'll guide you through the process of importing and using libraries like `uuid.js` to add features such as advanced data transformation and interactivity. By the end of this tutorial, you'll be equipped to extend your application's capabilities by seamlessly integrating third-party JavaScript libraries.

Practical 7.1: Importing a UUID JS Library (Demonstration)

We will be returning to the Step Form page we created before to demonstrate how to use this feature.

Let's say, now, the Instructor has to submit a special Registration code that was provided to them before to confirm their identity. This Registration code will be in a UUIDv4 format. When working on the form, we would like to generate the UUID to make it more convenient for ourselves.

However, by default, the Kaizen App Designer does not have a library to generate UUIDs. Fortunately, Kaizen is capable of supporting external JS libraries to help with this.

7.1.1 Setup Steps

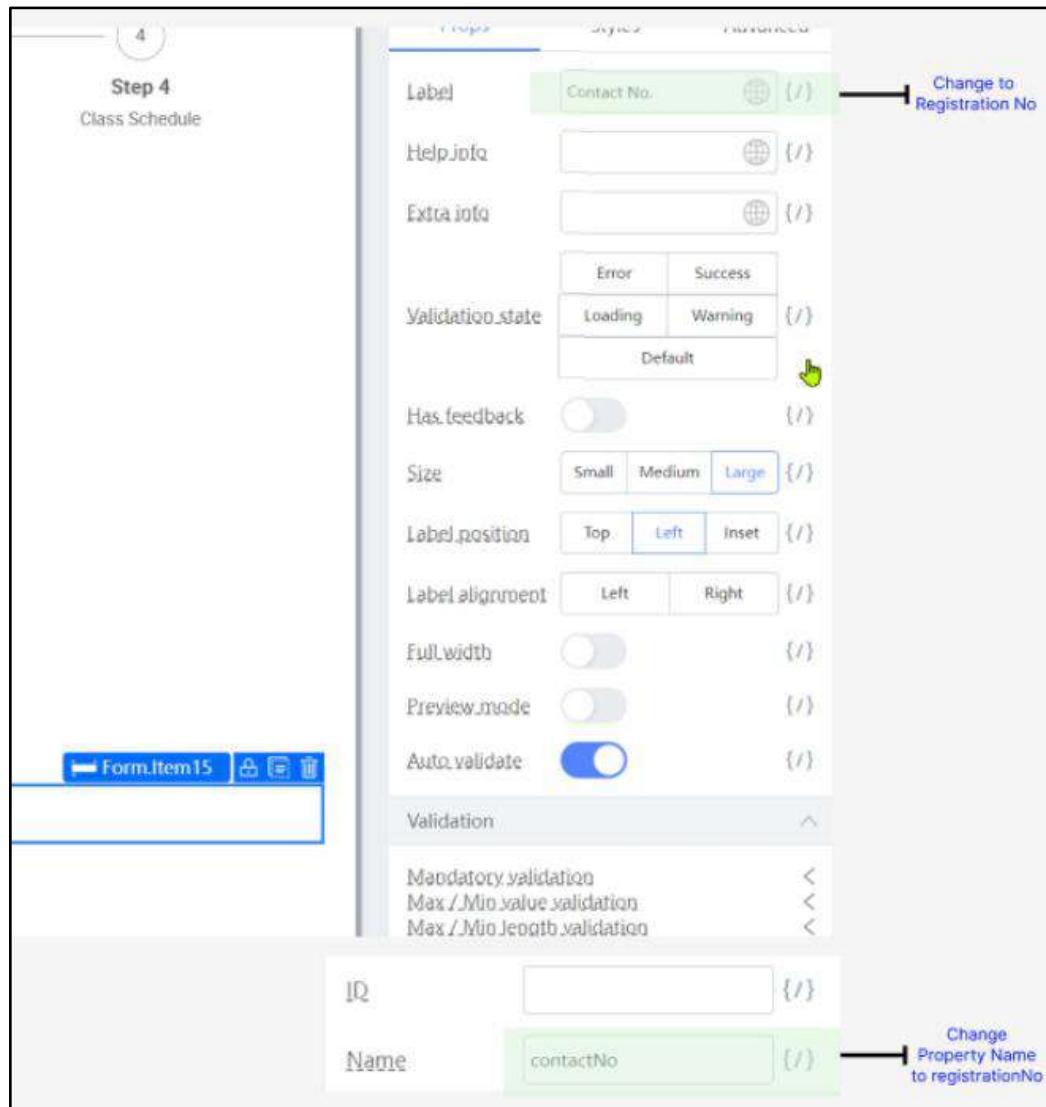
- Go to the 2nd Step (Open the Source Code Panel and change the “currentStep” to 1, then save the Source Code).
- Duplicate the Cell of the Contact No Form Item.

Instructor Particulars

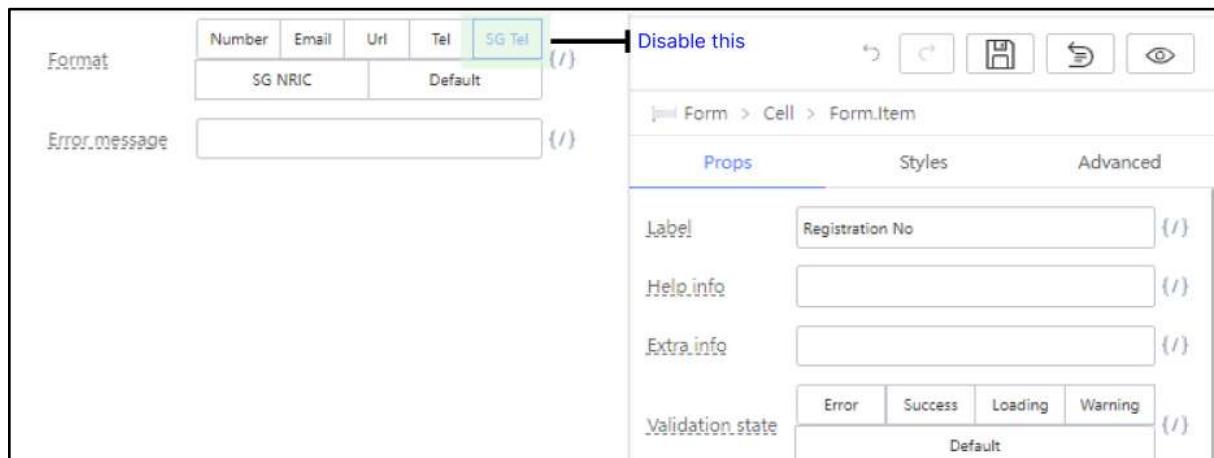
Retrieve Myinfo
with singpass

* Salutation	Please select
* Name	
* NRIC	
* Email	
Duplicate	
Form.Item80	
* Contact No.	

- Select the duplicated form item and change the Label to 'Registration No' and Property Name to 'registrationNo'.

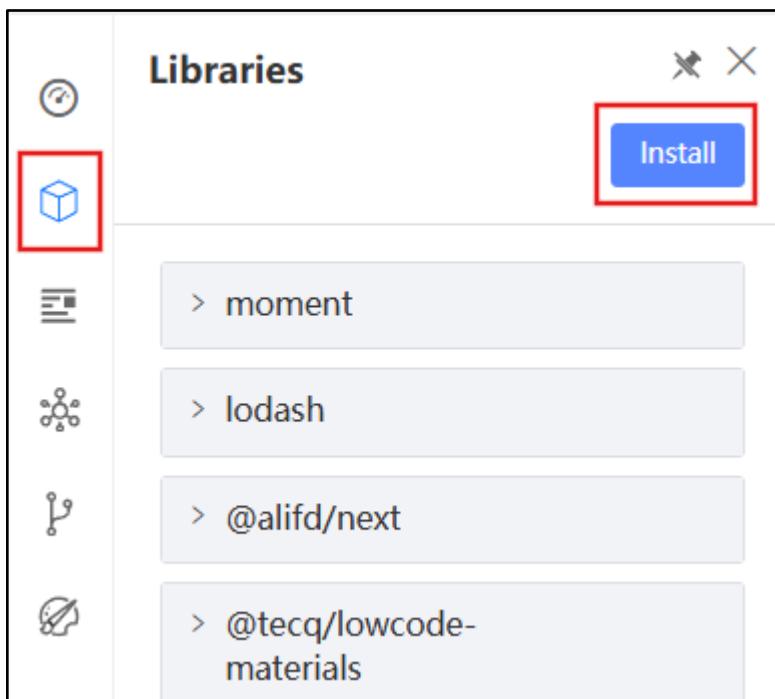


- Select the Registration No Form Item and select Default for the Format Validation.

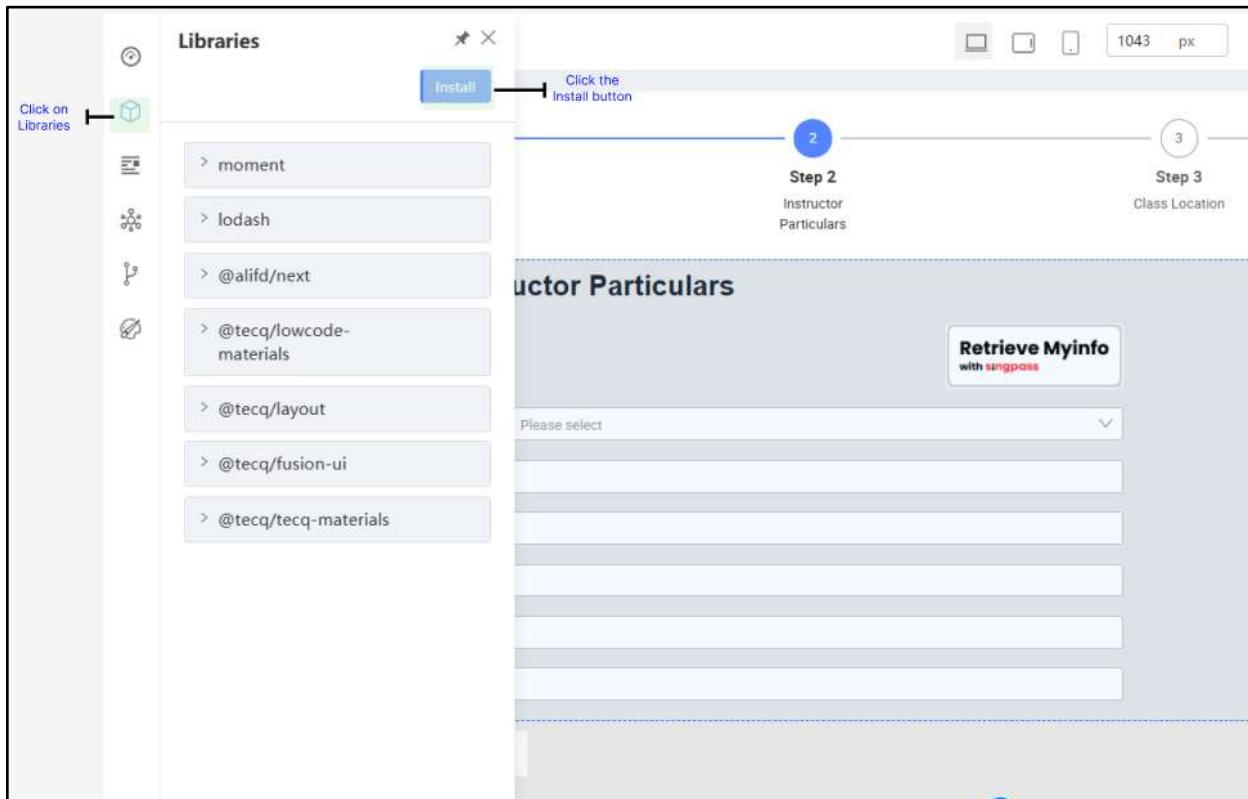


7.1.2 Adding the Library

- Select the **Libraries** option from the left sidebar. Click on the ‘Install’ button.



- Fill in the info for the ‘Add Library’ form:



```

Package: uuid
Version: 8.3.1
URL: https://unpkg.com/uuid@8.3.1/dist/umd/uuid.min.js
Library: uuid
  
```

- Click on the ‘Save’ button.
- Reload the Page.

With the `uuid` library installed, we can now use it in our Source Code. You can look at the possible functions here: <https://www.npmjs.com/package/uuid/v/8.3.1>.

7.1.3 Using the library

- Before using the library, make sure that Tutorial [4.7.3 Adding the 'MyInfo' Button](#) is done so there is the button to trigger the below method. If not, do proceed to create a button in the Instructor Particulars form component under the FormPage.
- In the Source Code Panel, add the following line in the retrieveSingPassMyInfo() function:

```
85     retrieveSingPassMyInfo() {  
86         this.formField.setValue("name", "Samuel Ong Jin Hai");  
87         this.formField.setValue("nrict", "S3052824G");  
88         this.formField.setValue("email", "ongjinhai2015@gmail.com");  
89         this.formField.setValue("contactNo", "99120384");  
90         this.formField.setValue("registrationNo", uuid.v4());  
91     }
```

```
this.formField.setValue("registrationNo", uuid.v4());
```

- Preview and click on the MyInfo button to check if the UUID is generated correctly.

The screenshot shows a four-step form titled 'Instructor Particulars'. Step 1 is completed, Step 2 is in progress, Step 3 is completed, and Step 4 is pending. The 'Retrieve Myinfo with singpass' button is located in Step 3. The form fields include Email (with a plus icon), Salutations (dropdown menu), Name (Samuel Ong Jin Hai), NRIC (S3052824G), Email (ongjinhai2015@gmail.com), Contact No (99120384), and Registration No (48267de4-33cf-4ee1-ae70-ca45ca0155eb). Navigation buttons 'Previous' and 'Next' are at the bottom.

Tutorial 8: Theme Designer

This tutorial covers the following Learning Objectives:

- Understand the purpose of the Theme Designer feature in KAIZEN
- Customize application themes by modifying color schemes and layouts without advanced knowledge of CSS
- Apply consistent branding across multiple applications using the Theme Designer
- Explore best practices for theme customization to enhance user interfaces

In modern application development, user interface (UI) and user experience (UX) are critical elements that influence how users interact with and perceive a product. KAIZEN streamline the development process by providing visual tools to design and deploy applications quickly. One of the key features in KAIZEN is the Theme Designer, which allows developers and administrators to customize the look and feel of applications without the need for in-depth front-end coding expertise.

In this tutorial, we'll explore how the Theme Designer enables customization of UI elements, helping users maintain consistent branding and visual appeal across multiple applications. This flexibility allows teams to create cohesive user experiences while reducing development time and effort.

Practical 8.1: Customize new theme

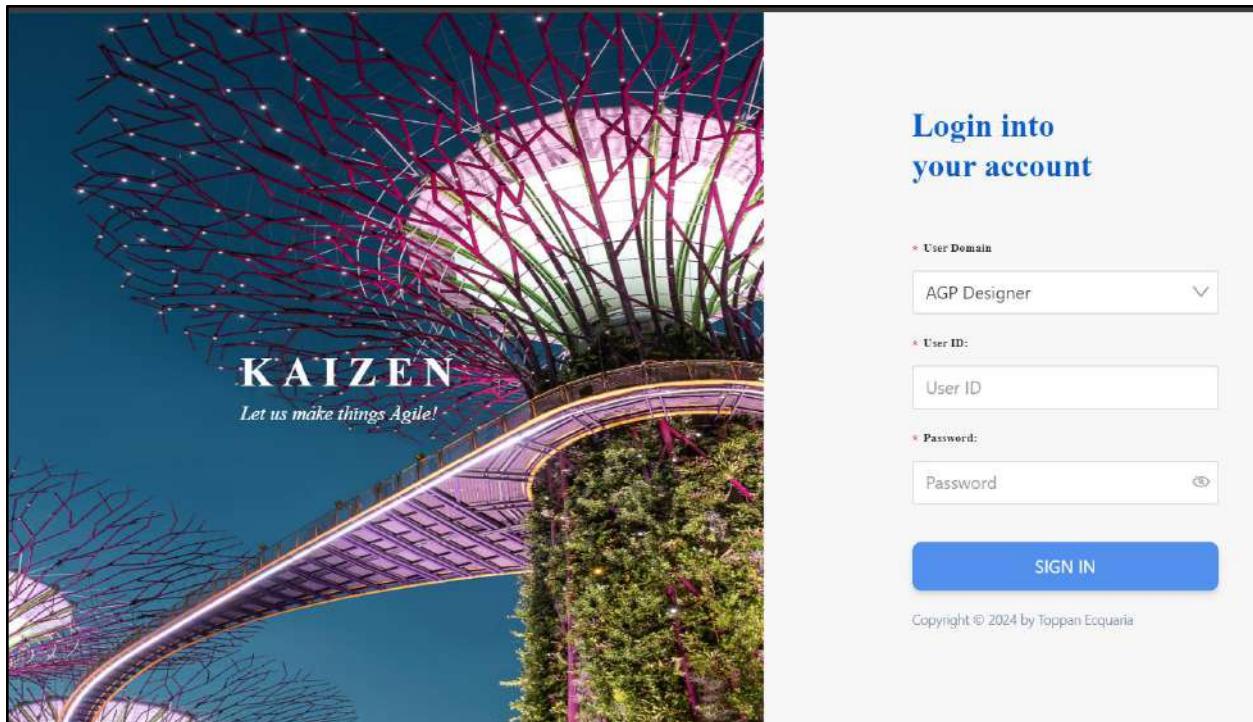
This tutorial will guide you through the customization of application themes using the KAIZEN Theme Designer, enabling you to maintain consistent branding and improve UI/UX with minimal coding.

- Open your browser, go to the Theme Designer URL, and log in using your credentials.

URL: <https://kaizen-daas.toppaecquaria.com/theme-editor/#/login>

Username: <username>

Password: <password>



- After login, you will see a list of themes available and published history panel
 - Edit: Users can further customize their selected theme by clicking the **Edit** button
 - Publish History: Users can access the **Publish History** feature to view the previously published versions of the selected theme.
 - Setting: Additional setting to edit/export/delete theme
 - Create (+): User can start to create a customize theme
 - Import: You can click the **Import** button to bring existing themes into the designer

KAIZEN Theme Designer

Search Theme [Search icon] [New icon] [Import icon]

Create Theme Import Theme

Recent Published Themes

Theme Name	Author	Version	Published
@tecq/gre...	admin	24.10.0	Published
@tecq/kai...	admin	24.10.0	Published
@tecq/ali...	admin	24.10.0	Published
@tecq/yo...	fooyongli		Published
@tecq/def...	admin	24.10.0	Published

Edit Button **Publish History** **Edit/Export/Delete Function**

Publish History

X

Basic Management

Theme Name: @tecq/yongli-test

Version: 1.0.0

Description:

Publish History

Version	Publisher	Publish time	Comment	Status
1.0.0	fooyongli	12 Dec 2024 01:38 PM	test	Success

Refresh

Now, let's start to create new project for our new theme:

The screenshot shows the KAIZEN Theme Designer interface. At the top, there is a search bar labeled "Search Theme". Below the search bar, there is a grid of four theme cards. Each card displays a color palette, the theme name, author, version, and a "Published" status. The fourth card from the top-right has a red arrow pointing to its "Published" status.

Theme Name	Author	Version	Status
@tecq/gre...	admin	24.10.0	Published
@tecq/kai...	admin	24.10.0	Published
@tecq/yo...	fooyongli		Published
@tecq/ali...	admin	24.10.0	Published

- Click on **Create Theme**, and follow the naming convention `@tecq/` to ensure proper integration.
 - **Theme Name:** For standard name convention, The theme name must start with `@tecq/<username>`. This is required for now as all js libraries integration
 - **Version:** We can add a version number to control and track updates. The recommended format should be Major.Minor.Patch (`1.0.0`).
 - **Inherit:** For the Inherit option, you can choose which theme you can copy from. So the new theme will be created based on the inherited theme.

The screenshot shows the "Create Theme" dialog box. It contains fields for "Theme Name" (set to `@tecq/amandalam`), "Version" (set to `1.0.0`), "Author" (set to `amandalam`), and "Inherit" (set to `@tecq/ali-orange-24.10.1`). There is also a "Description" field and two buttons at the bottom: "Cancel" and "Save".

Field	Value
Theme Name	@tecq/amandalam
Version	1.0.0
Author	amandalam
Inherit	@tecq/ali-orange-24.10.1

For the inherit option, we can either inherit from the KAIZEN default theme or inherit from the customized theme.

For this case we will inherit from `@tecq/ali-orange-24.10.1`.

Configure as per the above with your theme name prefixed with your username '`@tecq/<username>`' (E.g. `@tecq/amandalam`). After configure, click on the **Save** button to save your theme.

- Once the theme is created, you may access it by clicking on the design icon .



Before we go any further, let's take a brief aside to talk about the components of the Theme Designer.

Header Pane:

The Header Pane serves as a central control point for theme management, offering users the ability to **save** or **publish** their theme customizations.

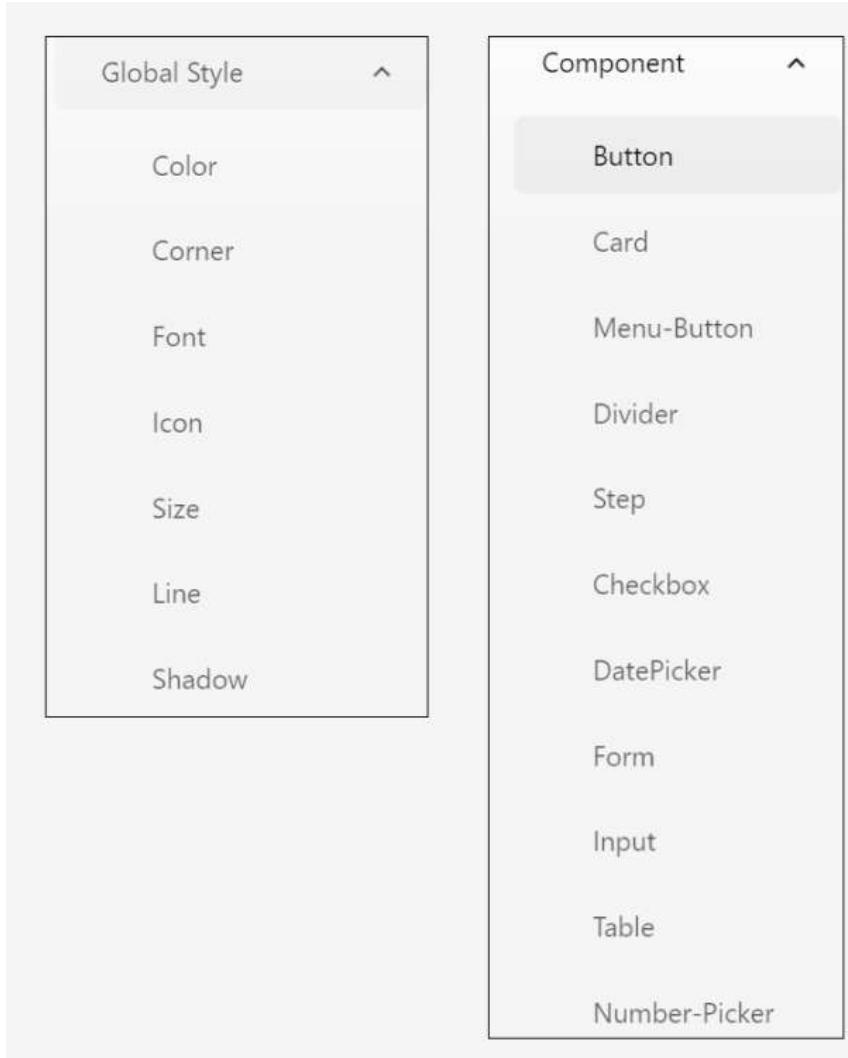


- **Save Functionality:** Users can save their theme progress at any point, ensuring that any adjustments made to colors, layouts, fonts, or other design elements are securely stored. This allows users to work incrementally without the risk of losing their changes.
- **Publish Functionality:** Once the theme is finalized, users can use the **Publish** feature to make their customized theme live across KAIZEN App Designer . This function ensures that all users or viewers will see and apply the newly updated theme.

Designer Pane:

The Designer Pane plays a crucial role in offering **real-time visibility** to the changes users apply while customizing their theme or design elements. As users make adjustments—whether it's altering the layout, modifying colors, or changing fonts—these changes are instantly reflected in the Designer Pane. This real-time feedback allows users to see how their design evolves without needing to reload or switch views, ensuring a smooth and uninterrupted workflow.

Component Pane:



Global Styles

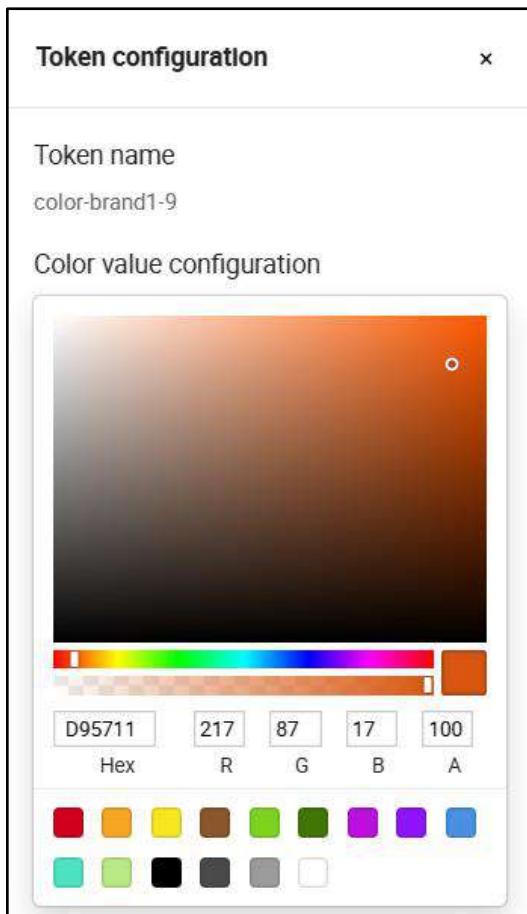
Global styles are settings that apply to the entire application or design system, ensuring consistency and scalability. They are used to configure the styling of your components.

Component

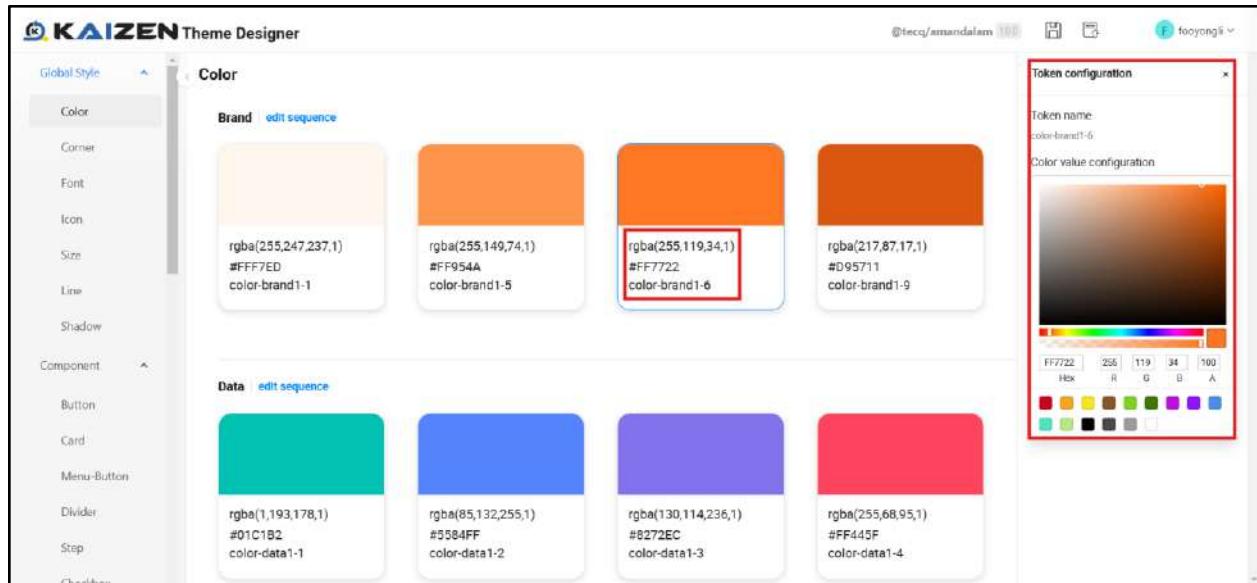
Components are individual, reusable UI elements that follow the global style guidelines. They are building blocks of the interface and can be customized with predefined properties.

Token Configuration Pane:

Once user click on the color, user will see the token Token Configuration Pane on the right hand panel. This pane enables users to adjust a wide range of design properties, such as borders, padding, text size, and colors for specific UI components (in this case, a button). It allows for detailed and precise styling adjustments, ensuring components align with the overall design language of the theme.



- Let's try to **design a new theme**. We will first customize a new color scheme for your brand.
- Click on Global Style > Color, you can see 3 available color codes for **Brand**, let's edit the second **color brand1-6** code. The color selection will appear once you click the color code.



- Let's try to edit **color-brand1-6** code with this hex color **#7F3618**.



- Once finished the **color-brand1-6** will immediately change its color display as below.

Color

Brand | [edit sequence](#)

rgba(255,247,237,1) #FFF7ED color-brand1-1	rgba(255,149,74,1) #FF954A color-brand1-5	rgba(127, 54, 24, 1) #7F3618 color-brand1-6	rgba(217,87,17,1) #D95711 color-brand1-9
--	---	---	--

- Next, you can **apply this new color code to Components**. When you click on the "Button" menu under the "Component" section, you will immediately see the effects for the "Active" color states for buttons

KAIZEN Theme Designer

Global Style

- Color
- Corner
- Font
- Icon
- Size
- Line
- Shadow

Component

- Button
- Card
- Menu-Button
- Divider
- Step
- Checkbox

Primary

L	M	S
Normal	Button (dark brown)	Button (brown)
Hover	Button (orange)	Button (orange)
Active	Button (orange)	Button (orange)
Disabled	Button (light gray)	Button (light gray)

Statement

- normal
- background
- border color
- border style
- shadow
- text
- hover
- background

Color palette on the right side of the interface.

Description of Token Configuration

Menu	Section	Usage
bounding	border width	Thickness of the border
	height	Height of the border
	x	Padding of the border
text	size	Size of the text
normal	corner	Style of the corner
Statement - normal,hover,active	background	Background color
	border color	Border color
	border style	Border style
	shadow	Shadow style
	text	Text color

- Once complete, click on **Save** to store your personalized theme.



Save draft

* Theme Name: @tecq/amandalam

* Version: 1.0.0

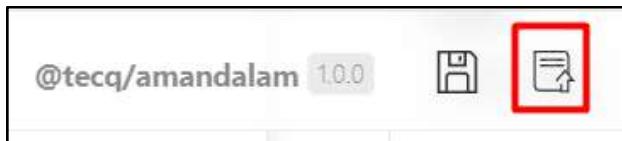
* Author: Foo Yong Li

* Comment: save

Save **Reset**

Save success

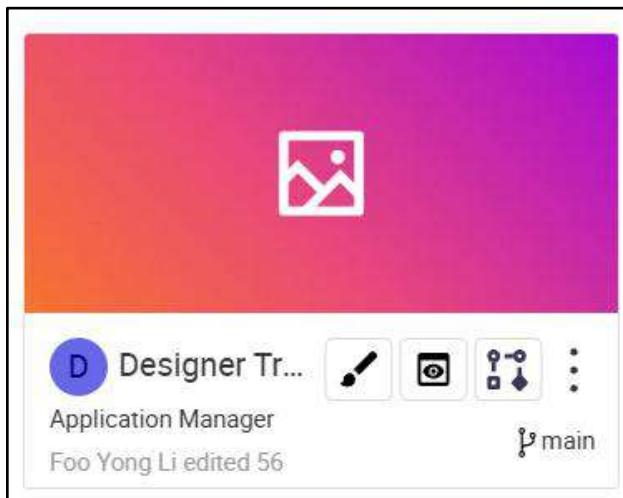
- This button here is to publish the theme to the Nexus repository. Note that publishing a theme to a Nexus repository does indeed take some processing time, typically around **5-10 minutes**. In the interest of time, we will skip this step for the training.



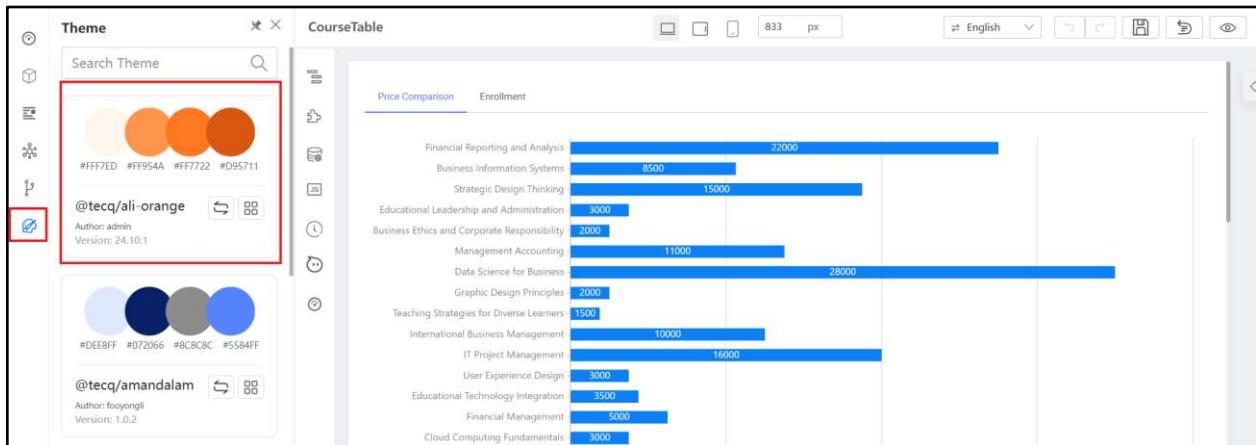
- Once the theme is published, you will be able to start applying the published theme in the app Designer. Once a theme is published, it becomes publicly accessible and can be viewed and used globally.

Practical 8.2 Apply theme to your application

- We have pre-created a theme to be applied onto your Designer Training application. First, login to [KAIZEN](#) and access the App Designer of your application. Navigate to the Course Table page.



- Click on the theme design button and select the **@tecq/ali-orange** theme to apply, then click **Ok**.



- The page will be reloaded for the changes to take effect. After that, you will see the colour and style of the components of your application has changed according to this newly applied theme.

The screenshot displays a dashboard application with two main panels. On the left, a 'Theme' sidebar lists available themes: '@tecq/ali-orange' (selected) and '@tecq/amandalam'. The main panel, titled 'CourseTable', contains a 'Price Comparison' section with a bar chart and a 'Enrollment' section with a pie chart. A modal dialog titled 'Apply Theme' asks 'Are you sure you want to apply?' with 'Ok' and 'Cancel' buttons. The 'Ok' button is highlighted with a red box.

Price Comparison

Course Name	Price
Business Information Systems	8500
Strategic Design Thinking	15000
Educational Leadership and Administration	3000
Business Ethics and Corporate Responsibility	2000
Management Accounting	11000
Data Science for Business	28000
Graphic Design Principles	2000
Teaching Strategies for Diverse Learners	1500
International Business Management	10000
IT Project Management	16000
User Experience Design	3000
Educational Technology Integration	3500
Financial Management	5000
Client Communication Fundamentals	3000

Enrollment

A pie chart illustrating student enrollment distribution across different courses. The segments are labeled with their respective counts:

- Financial Reporting and Analysis: 1120
- Business Information Systems: 1460
- Strategic Design Thinking: 1120
- Educational Leadership and Administration: 1900
- Business Ethics and Corporate Responsibility: 240
- Management Accounting: 200
- Data Science for Business: 150
- Graphic Design Principles: 120
- Teaching Strategies for Diverse Learners: 150
- International Business Management: 200
- IT Project Management: 120
- User Experience Design: 150
- Educational Technology Integration: 100
- Financial Management: 150
- Client Communication Fundamentals: 100

CourseTable

Course Name	Course Code	Subject	Enrollment Type	Duration	Schedule	Price	Students
Financial Reporting and Analysis	4156	Accounting	Full-time	4 months	weekly	22000	1325
Business Information Systems	7224	Computing	Part-time	3 months	bi-weekly	8500	235
Strategic Design Thinking	6521	Design	Full-time	6 months	weekly	15000	245
Educational Leadership and Administration	4765	Education	Part-time	2 months	bi-weekly	9000	20
Business Ethics and Corporate Responsibility	8932	Business	Full-time	1 month	daily	2000	180
Management Accounting	2678	Accounting	Part-time	5 months	weekly	11000	125
Data Science for Business	5412	Computing	Full-time	4 months	bi-weekly	28000	150
Graphic Design Principles	6899	Design	Part-time	3 months	weekly	2000	90
Teaching Strategies for Diverse Learners	3245	Education	Full-time	2 months	bi-weekly	1500	20
International Business Management	9123	Business	Part-time	1 month	daily	10000	150

Total: 22 [1](#) [2](#) [3](#) [4](#) Items per page: 10

[Download Data as JSON](#)

Additional Information

Export

You can click export theme if you wish to share the theme design.



Import

You also click on the import button to import existing themes into the designer.

The screenshot shows the KAIZEN Theme Designer interface. At the top, there is a search bar labeled "Search Theme" with a magnifying glass icon, a "+" button, and a red-bordered "Import" button. Below the search bar, there are four theme cards, each representing a different color scheme:

- Theme 1:** Four circles in light blue, dark blue, grey, and dark blue. Hex codes: #DEEBFF, #479EEA, #8C8C8C, #5584FF. Author: admin. Version: 24.10.1. Status: Published.
- Theme 2:** Four circles in light blue, cyan, light blue, and cyan. Hex codes: #E6FDFE, #2B05FF, #03C1FD, #0090D6. Author: admin. Version: 24.10.1. Status: Published.
- Theme 3:** Four circles in light orange, orange, orange, and dark orange. Hex codes: #FFF7ED, #FF954A, #FF7722, #D95711. Author: admin. Version: 24.10.1. Status: Published.
- Theme 4:** Four circles in light green, green, light green, and dark green. Hex codes: #A9FFED, #C1E1C1, #9FE2BF, #008080. Author: amandalam. Version: 24.10.1. Status: Published.

Tutorial 9: Preview Mode

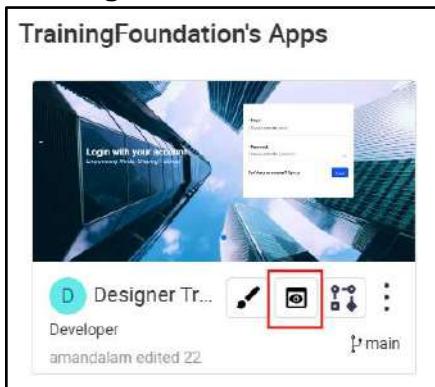
Practical 9.1: Preview Mode and Collaborative Commenting

This practical covers the following Learning Objectives:

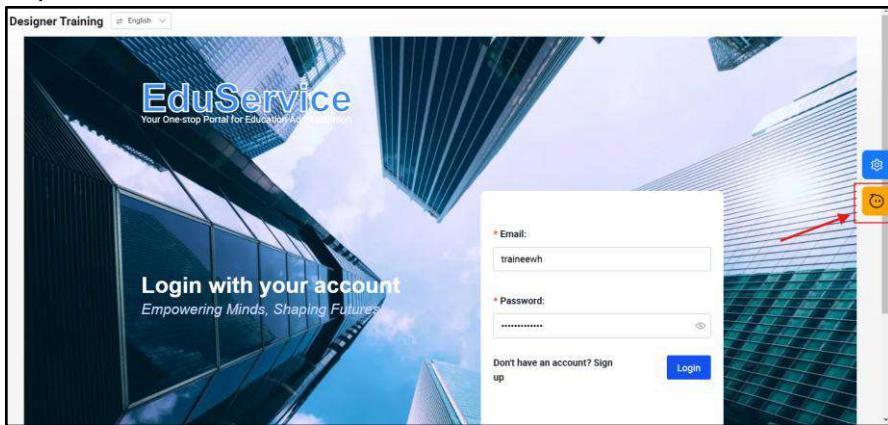
- Learn how to enable preview mode in your application for testing and review purposes.
- Understand how to implement collaborative commenting to allow team members to provide feedback on the application.
- Enhance team collaboration by providing a seamless workflow for reviewing and improving your application before publishing.

In this practical, you will learn how to enable preview mode in your application, allowing you and your team to test and review changes in real-time before finalizing them. Additionally, we will guide you through implementing collaborative commenting, so team members can easily provide feedback and suggestions. This feature enhances collaboration and ensures that all stakeholders can participate in the review process to improve the application before it goes live.

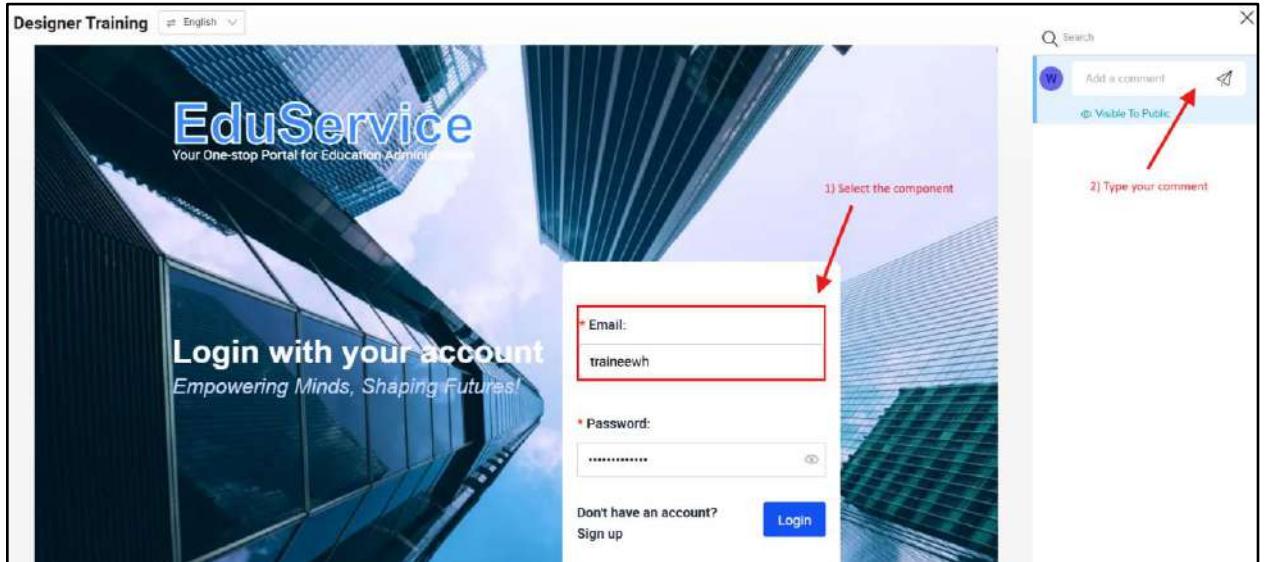
- Navigating back to the shared project that all trainees can access - **TrainingFoundation**, click on the 'Preview' button the application.



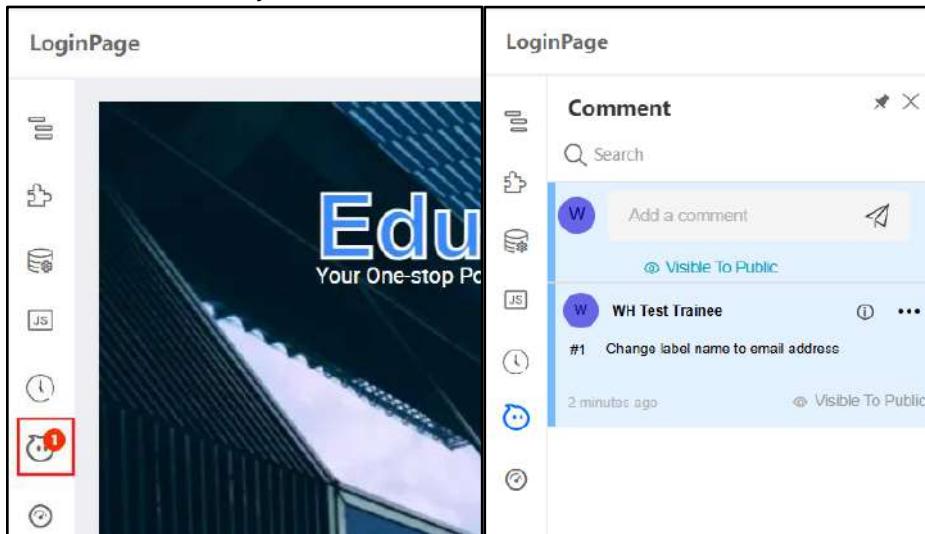
- In preview mode, click on the Comments icon near the sidebar.



- Select a component, and type the comment in the comment box.
 - Note that you can toggle the visibility of your comment to either “Visible to Public” or “Visible to Internal” by clicking on the text below the comment box. This is useful in the event that a project member wants to leave a comment to internal team members, which should not be visible to the customers when they preview this page.



- Navigating back to this page in the App Designer, click on the Comments icon to bring up the list of comments relating to this page. You will be able to access and see the comments added by all users.



Practical 9.2: Navigator Creation / Page Linking

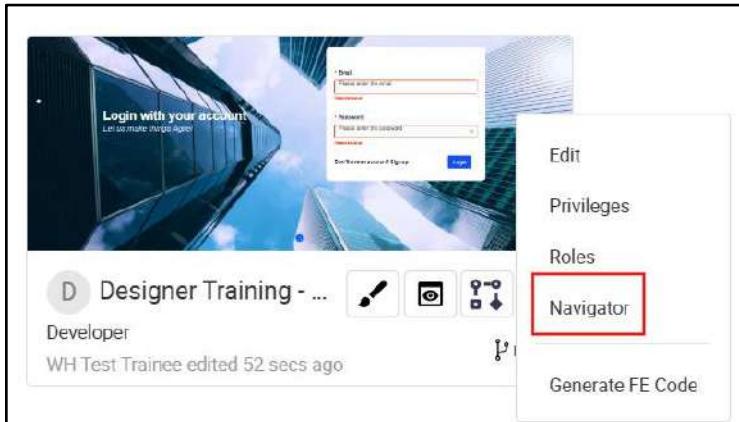
This practical covers the following Learning Objectives:

- Learn how to create and organize navigators within your application.
- Explore how to enhance user experience by designing intuitive and easy-to-use navigation menus.
- Learn how to create a custom navigator and menu within your application.
- Customize the user experience by creating tailored navigation flows.
- Understand how to link pages efficiently to ensure smooth navigation for users.

In this practical, you will learn how to create navigators for your application, enabling users to easily navigate between pages. We'll guide you through the process of linking pages effectively, ensuring that the navigation is intuitive and user-friendly. By the end of this tutorial, you will be able to design and implement an organized, seamless navigation system that improves the overall user experience within your application.

9.2.1 Navigator Creation

- Navigate back to your own project and application.
- Click on **Navigator** to edit the setting



```
User Domain: Default domain
Name: Landing page
Parent: (Empty)
Path: /landing
Target: Landing Page
Open Method: Current Tab
Priority: 1
Icon: List
```

Create Navigator

* User Domain: Default domain

* Name: Landing Page

Parent: Please Select

* Path: /landing

Target: LandingPage

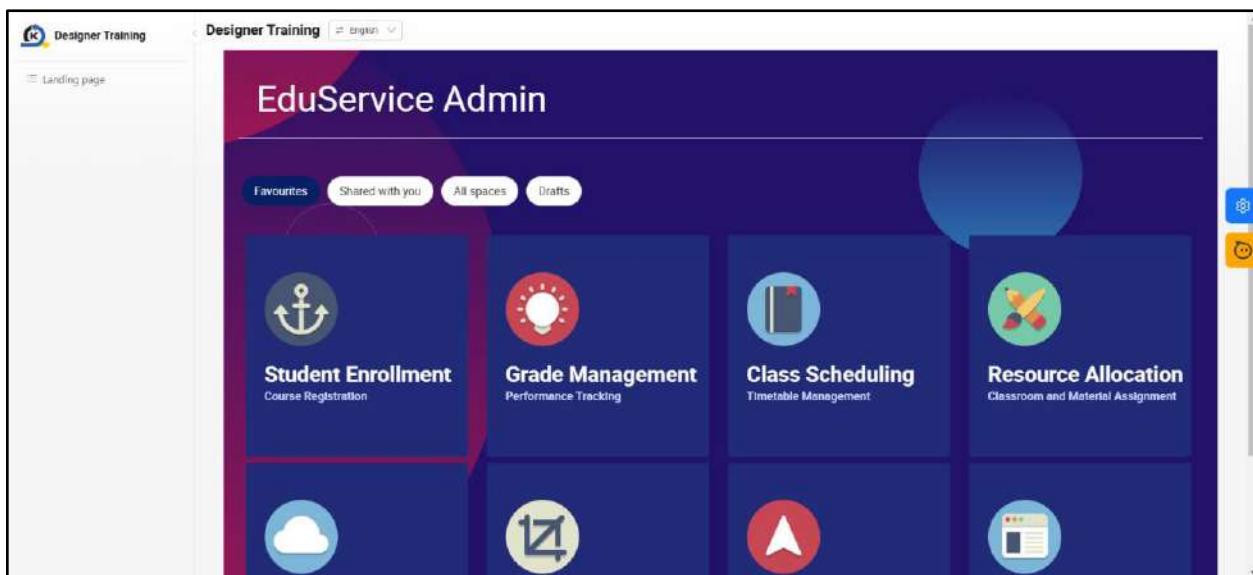
* Open Method: Current Tab

* Priority: 1

Privileges: Please Select

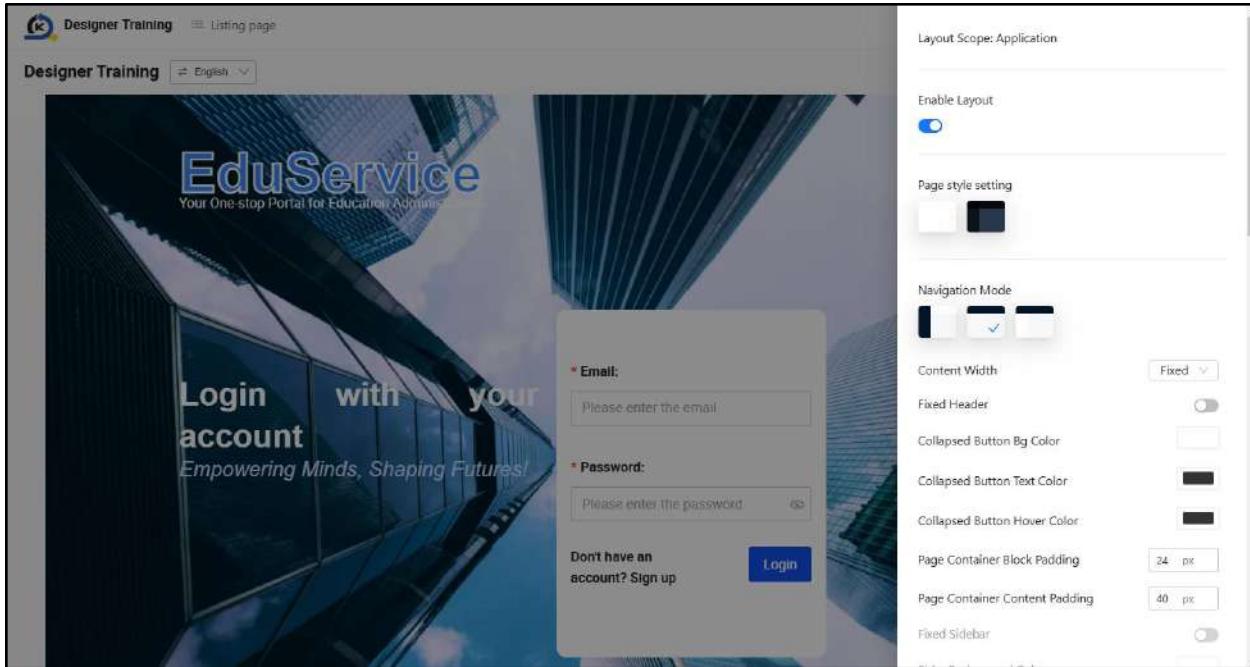
Icon:

- Now, when you preview your application, you will be able to see your newly created navigator, which brings you to the Landing Page upon click.

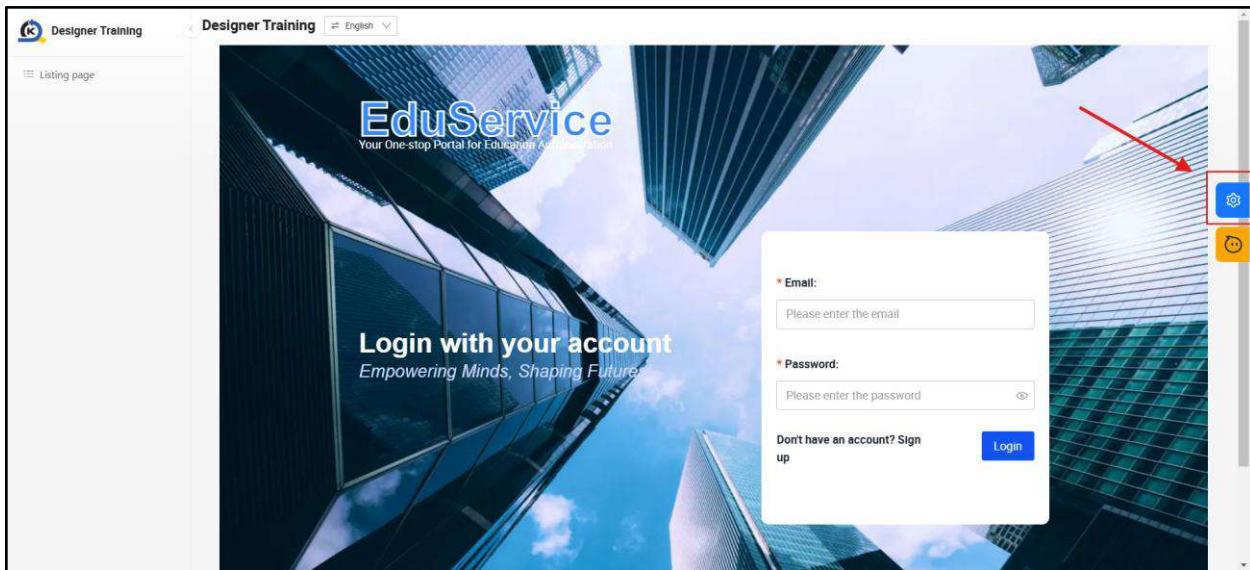


9.2.2 Customize Navigator Menu

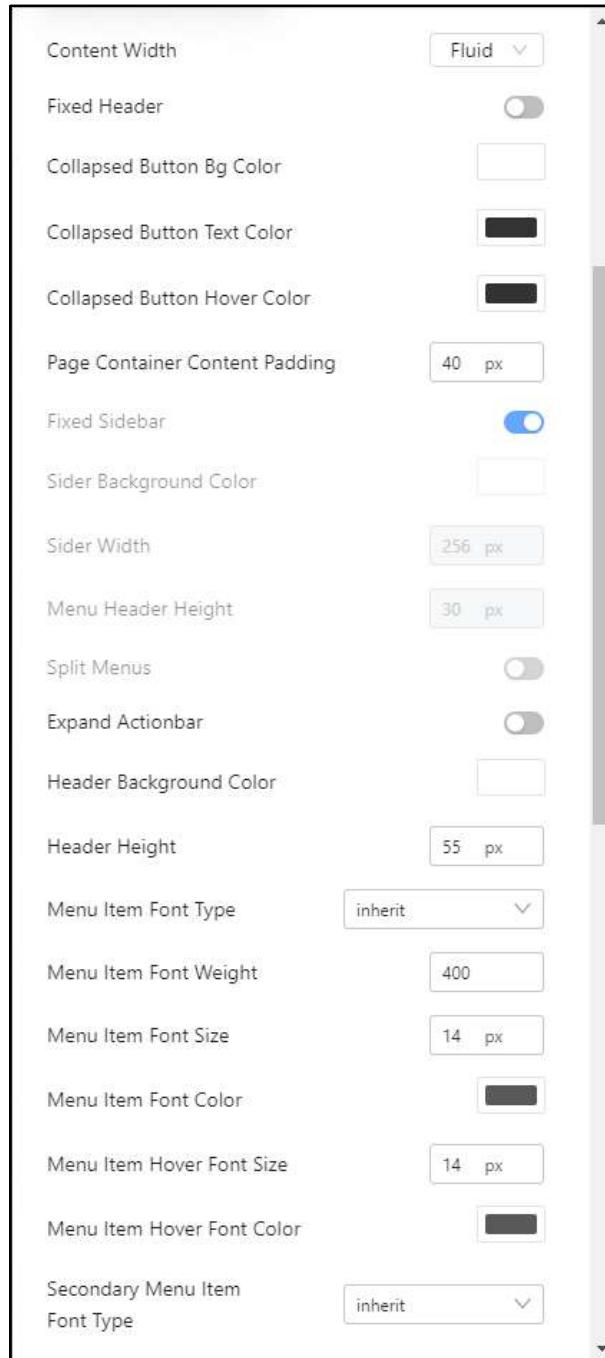
After creating your Navigator Menu, you will learn how to customize the navigators and menus for your application, providing a personalized user experience.



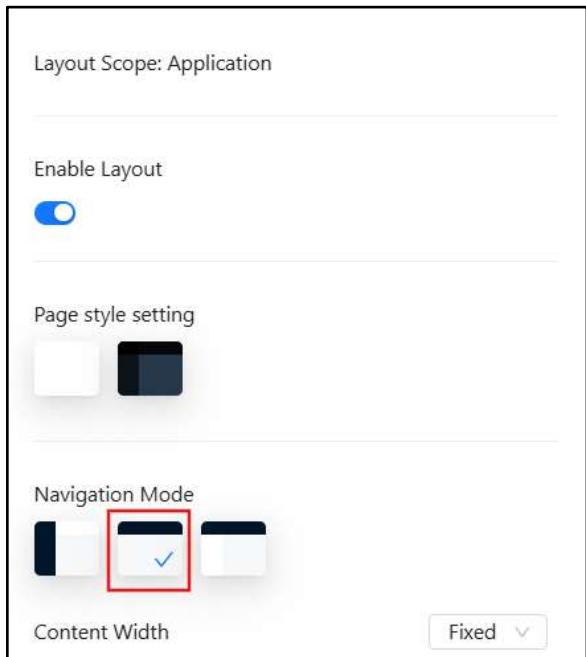
- Preview your application.
- Click on the Setting icon in order to start customizing your navigator menu. We will now proceed to edit the menu color.



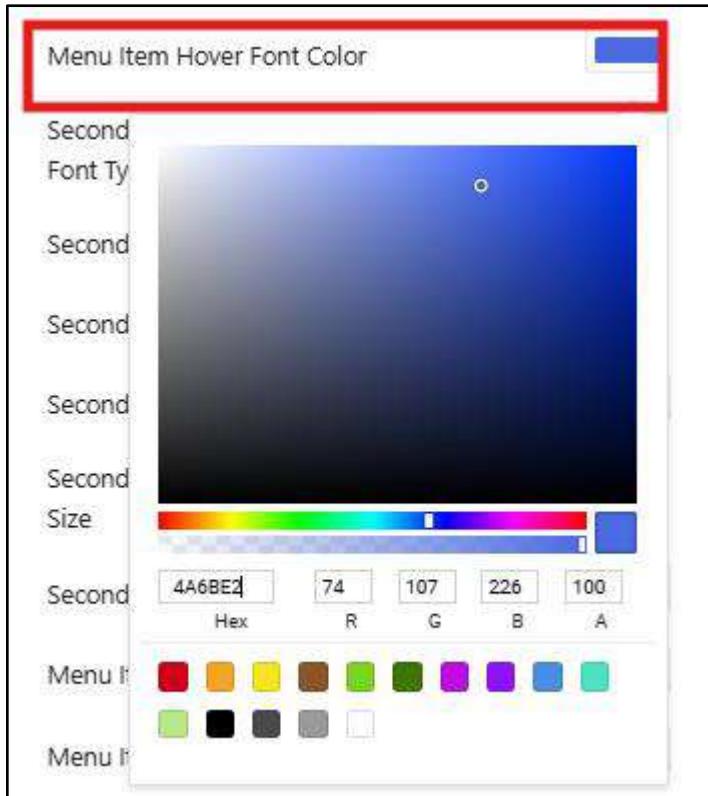
- In the settings, note that there are many CSS properties to change navigator menus

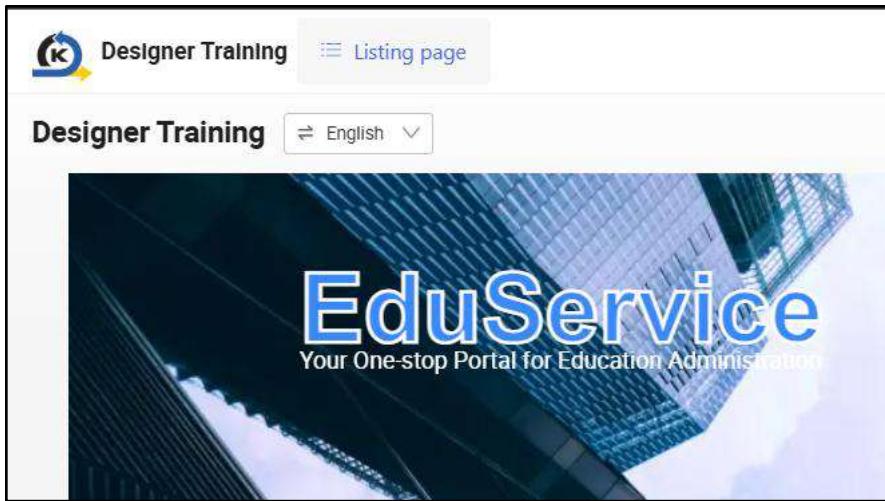


- Change the Navigation Mode to ‘Top Menu Layout’

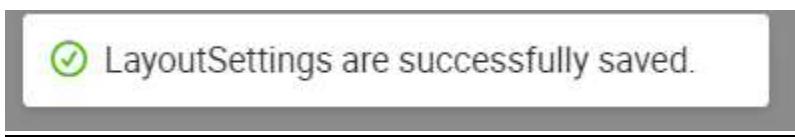
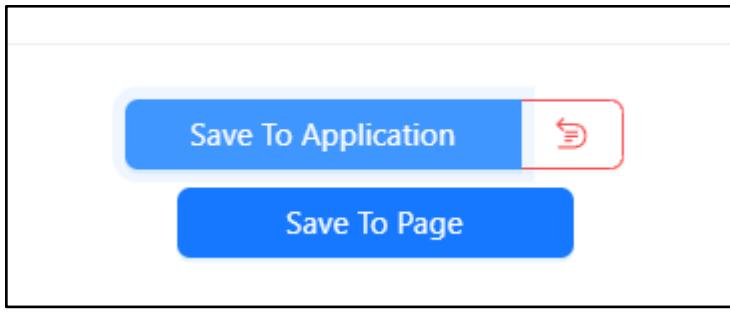


- Edit “Menu Item Hover Font Color” to change menu color on hover





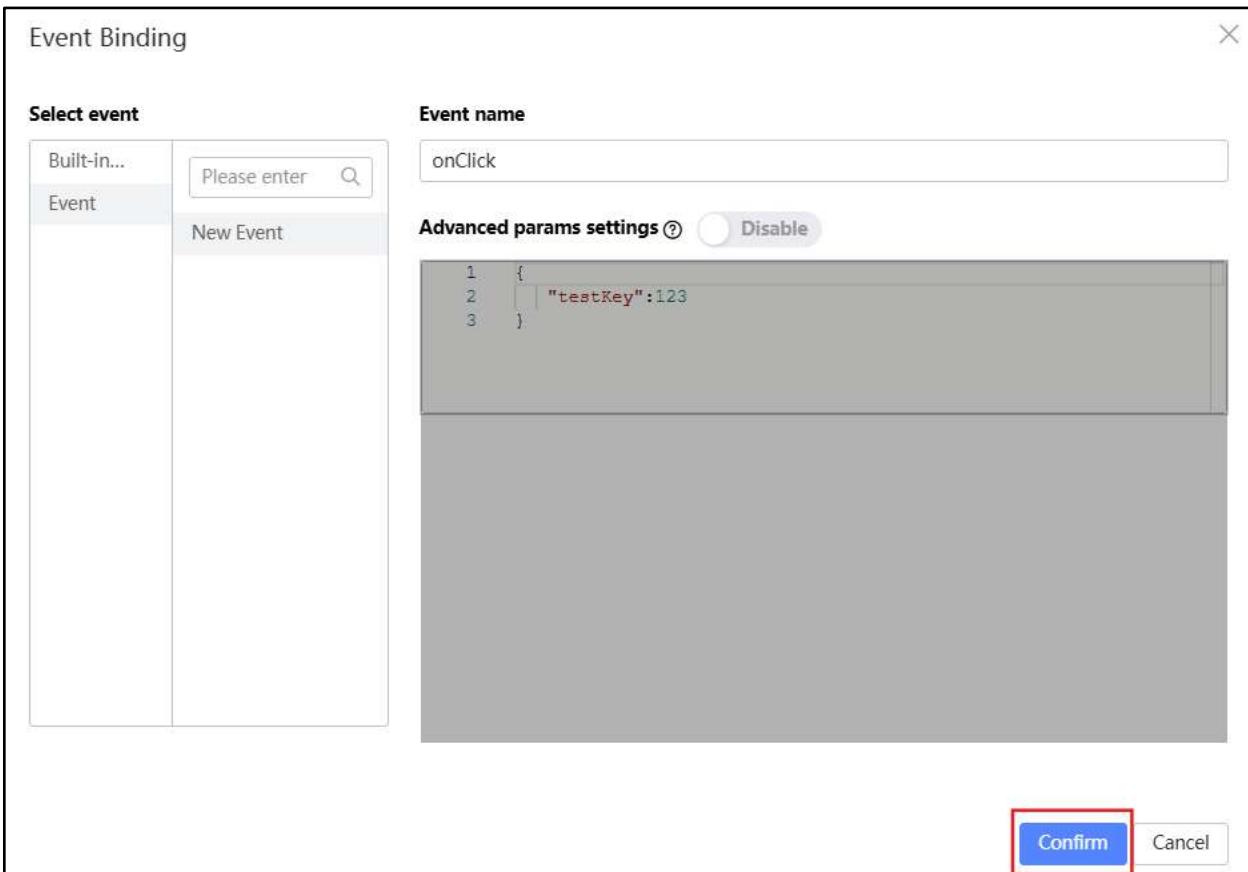
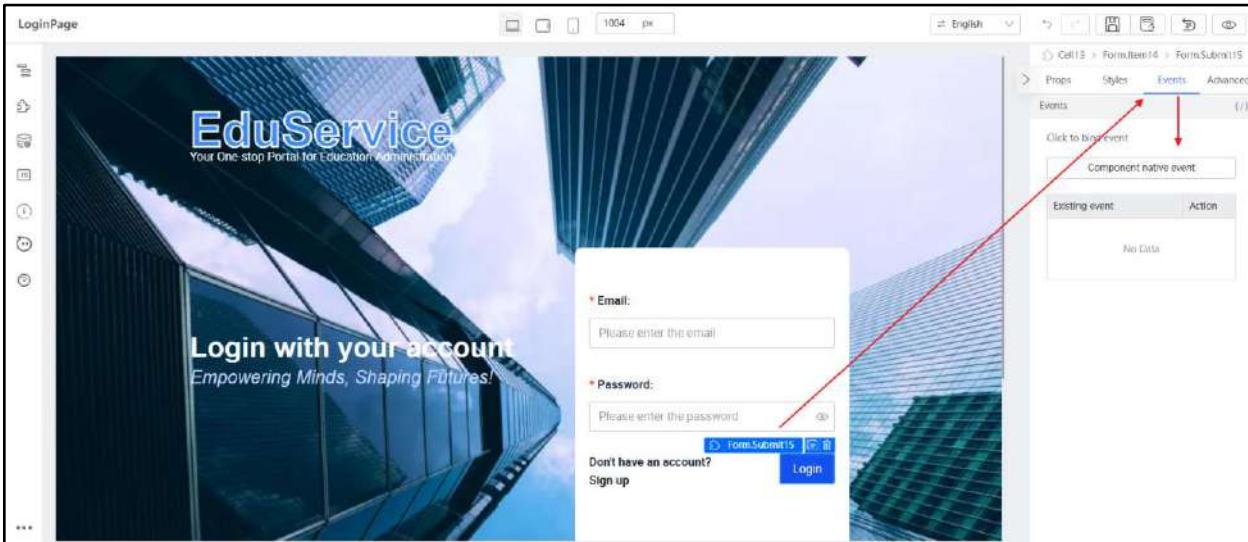
- Click on “Save To Application”. This will save your changes across all pages to your entire application. If you only want to customize at page level, you can click on “Save To Page” instead.



9.2.3 Page Linking

Once all your pages and Navigator Menus are created, we can connect them to ensure a seamless experience by binding the onClick event.

- In your Login Page, bind the onClick event.



- Add the following code in the JS Source Code Panel.
The **navigateTo** function takes in two parameters, the “App ID” and “Page ID”. Replace with your application IDs accordingly.

```
onClick(){
    this.utils.navigateToCurrAppPages('Page ID');
}
```



- Save your changes and try to login in preview mode! Your page should navigate over to the landing page successfully.

Practical 9.3: Page Tagging for External Review (Demonstration)

This practical covers the following Learning Objectives:

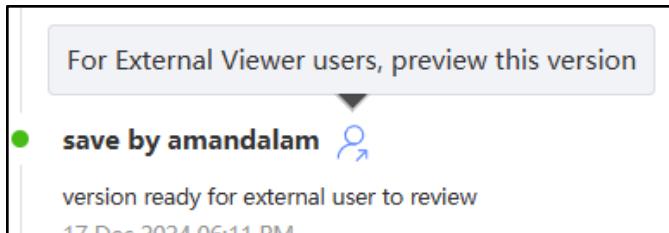
- Learn how to implement page tagging for external review, enabling stakeholders to view specific versions of your application.
- Understand how to manage and control which updates are visible to external reviewers while hiding upcoming revisions.
- Explore how to tag pages for review, ensuring customers can only view the current version and not unfinished updates.

In this practical, you will learn how to use page tagging for external review, which allows you to mark specific versions of your application for stakeholder review. By tagging pages, you ensure that external reviewers can only see the current version of the application, while any ongoing revisions remain hidden. This feature helps to maintain a clear and organized review process, ensuring that external feedback is based on a consistent, unaltered version of the product.

- We will be using the Designer Training application in our TrainingFoundation project.
- Under the revision history of each page, you can click on the button for external viewers to preview that particular published version. Subsequent versions with further enhancements by the project team will not be visible to external viewers when they preview the application.

The screenshot shows a 'LoginPage' interface with a 'Revision' history panel. The panel lists five revisions made by 'amandalam'. The second revision, which is highlighted with a blue circle and a red box around its 'Review' column, is described as 'version ready for external user to review' and was made on 17 Dec 2024 06:11 PM. Other revisions include saving features, changing text, publishing pictures, and a save operation by PageCreate.

Revision	Details
save by amandalam	feature enhancement 17 Dec 2024 06:12 PM
save by amandalam	version ready for external user to review. 17 Dec 2024 06:11 PM
save by amandalam	change text 17 Dec 2024 06:05 PM
save by amandalam	publish picture 17 Dec 2024 06:04 PM
save by amandalam	Save By PageCreate 17 Dec 2024 06:03 PM



- We have assigned a customer user account with an external viewer role assigned for this application as seen to demonstrate this.

Assign User

User Domain	Account ID	User Name	Role	Actions
AGP Designer	amandalam	amandalam	Application Manager	Remove
AGP Designer	weihan	weihan	Application Manager	Remove
AGP Designer	fooyongli	Foo Yong Li	Application Manager	Remove
AGP Designer	yongheng	yongheng	Application Manager	Remove
AGP Designer	haryantoluo	Haryanto Luo	Internal Viewer	Remove
AGP Designer	traineewh	WH Test Trainee	Developer	Remove
AGP Designer	customer	customer	External Viewer	Remove

- Now as a customer, the customer will only be able to see this version tagged for them to review and not subsequent versions when previewing this application.

Tutorial 10: Page Improvement

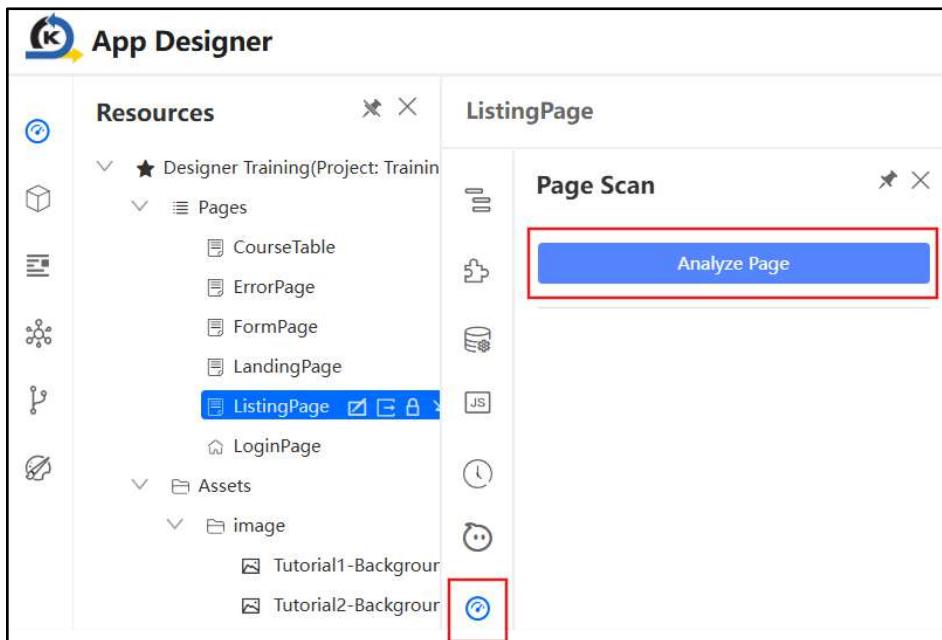
Practical 10.1: Page Analysis with Page Scan (Demonstration)

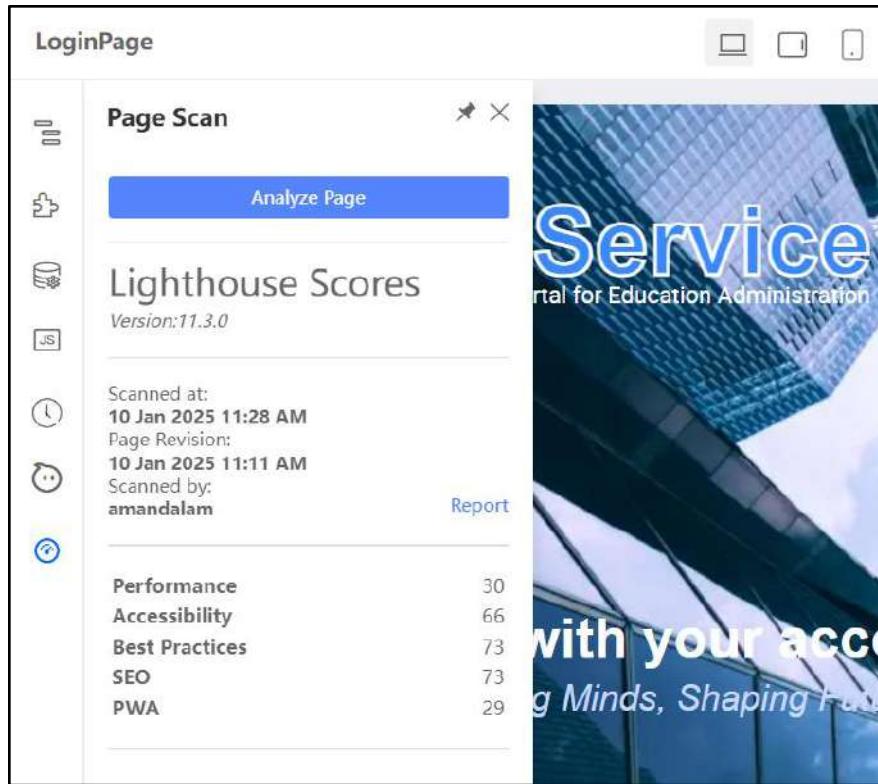
This practical covers the following Learning Objectives:

- Understand how to perform a comprehensive page scan using the platform's built-in tool which analyzes web apps and web pages, collecting modern performance metrics and insights on developer best practices.
- Analyze key metrics such as Performance, Accessibility, Best Practices, SEO, and Progressive Web App (PWA) compliance to identify areas for improvement.

In this practical, you will learn how to use the platform's page scanning capabilities to evaluate your pages against industry-standard metrics, optimize them for better performance and accessibility, and ensure they meet the latest web development best practices.

- Select the “Page Scan” plugin and click ‘Analyze Page’.





- Upon scanning, you will also be able to download the report of the page scan which will also give you further insights such as recommended changes to your page.

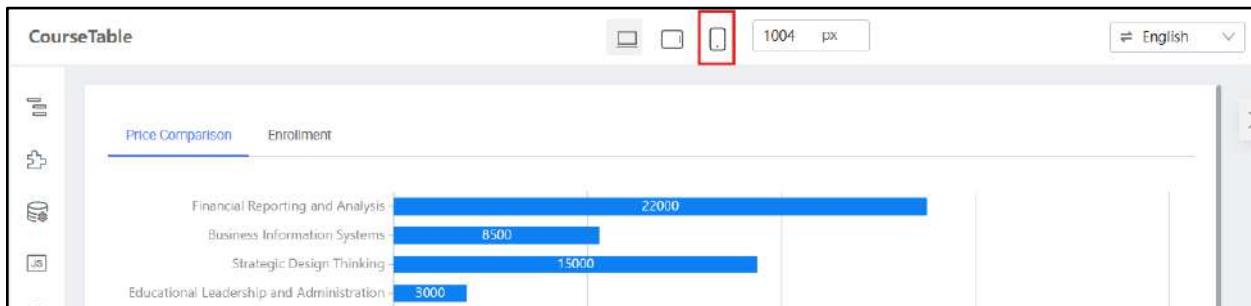
Practical 10.2: Mobile Responsive UI (Demonstration)

This practical covers the following Learning Objectives:

- Learn to configure table rows to collapse for optimal display on smaller screen sizes.
- Explore adaptive design techniques to ensure usability across various device types.

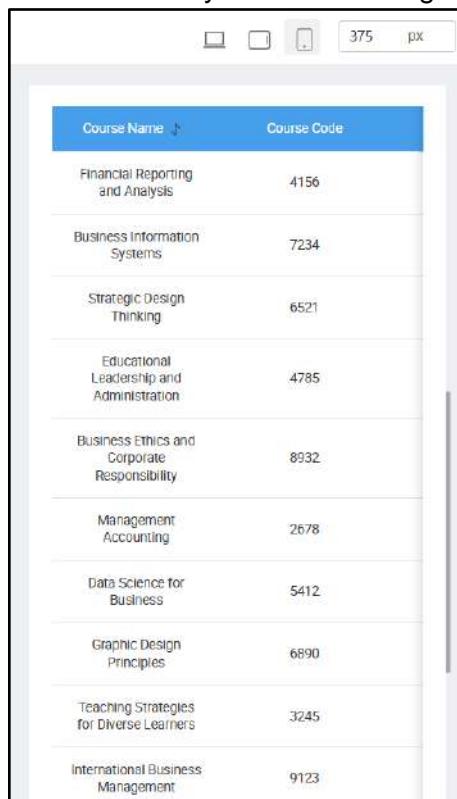
In this practical, you will learn how to apply the collapse row feature to optimize table data presentation for small screens, which is one of the mobile responsive features provided.

- Navigate to the Course Table page and click on the ‘mobile’ view icon.



The screenshot shows a desktop browser window with a title bar 'CourseTable'. Below the title bar are icons for desktop, tablet, and mobile devices, with the mobile icon highlighted by a red box. To the right of these icons is a resolution setting '1004 px' and a language dropdown set to 'English'. The main content area displays a horizontal bar chart titled 'Price Comparison' with categories 'Enrollment' and 'Financial Reporting and Analysis' at the top. Below the chart, there are four bars representing different courses: Financial Reporting and Analysis (22000), Business Information Systems (8500), Strategic Design Thinking (15000), and Educational Leadership and Administration (3000). A vertical scrollbar is visible on the right side of the content area.

- When you scroll down to the table, notice that its view is not user friendly when in mobile screen size as you need to navigate via horizontal scrolling.



The screenshot shows a mobile browser window with a title bar 'CourseTable' and icons for desktop, tablet, and mobile devices, with the mobile icon selected. Below the title bar is a resolution setting '375 px'. The main content area displays a table with two columns: 'Course Name' and 'Course Code'. The table lists ten courses with their respective codes. The table is scrollable both horizontally and vertically. The first few rows of the table are visible, showing 'Financial Reporting and Analysis' (4156), 'Business Information Systems' (7234), 'Strategic Design Thinking' (6521), 'Educational Leadership and Administration' (4785), 'Business Ethics and Corporate Responsibility' (8932), 'Management Accounting' (2678), 'Data Science for Business' (5412), 'Graphic Design Principles' (6890), 'Teaching Strategies for Diverse Learners' (3245), and 'International Business Management' (9123).

Course Name	Course Code
Financial Reporting and Analysis	4156
Business Information Systems	7234
Strategic Design Thinking	6521
Educational Leadership and Administration	4785
Business Ethics and Corporate Responsibility	8932
Management Accounting	2678
Data Science for Business	5412
Graphic Design Principles	6890
Teaching Strategies for Diverse Learners	3245
International Business Management	9123

- Click on the table. In props, select 'Collapse Row' for responsive table type.

Notice that your table will now be collapsed on mobile view, which is a more intuitive and user friendly interface.

Cell > Paragraph > Table1

Props Styles Events Advanced

Custom.toolbar.rendering.configuration

Custom.rendering Disable {/}

Top-right.rendering Disable {/}

Top.rendering Disable {/}

Style and theme

Theme Zebra Split Border {/}

Show.header {/}

Fixed.header {/}

Max.body.height 800 {/}

Sticky.header {/}

Use.Virtual {/}

Empty.value.content {/}

Empty.content No data {/}

Responsive.table.type Default Collapse Row {/}

Responsive.break.point.(px) 500 {/}

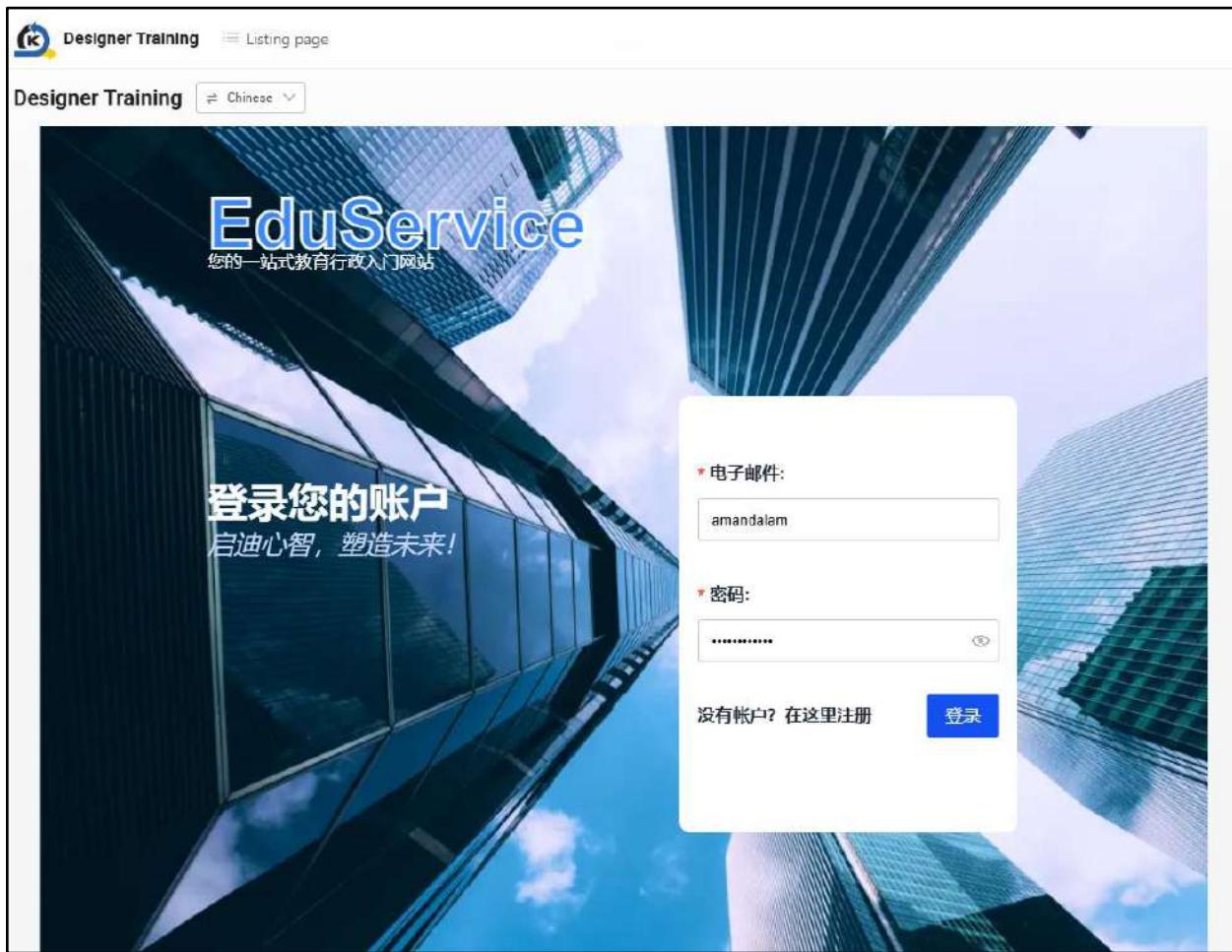
Table1	
Course Name	Financial Reporting Analysis
Course Code	4156
Subject	Accounting
Enrollment Type	Full-time
Duration	4 months
Schedule	weekly
Price	22000
Students	1325
Course Name	Business Information Systems
Course Code	7234
Subject	Computing
Enrollment Type	Part-time
Duration	3 months
Schedule	bi-weekly

Practical 10.3: Translation using i18n (Demonstration)

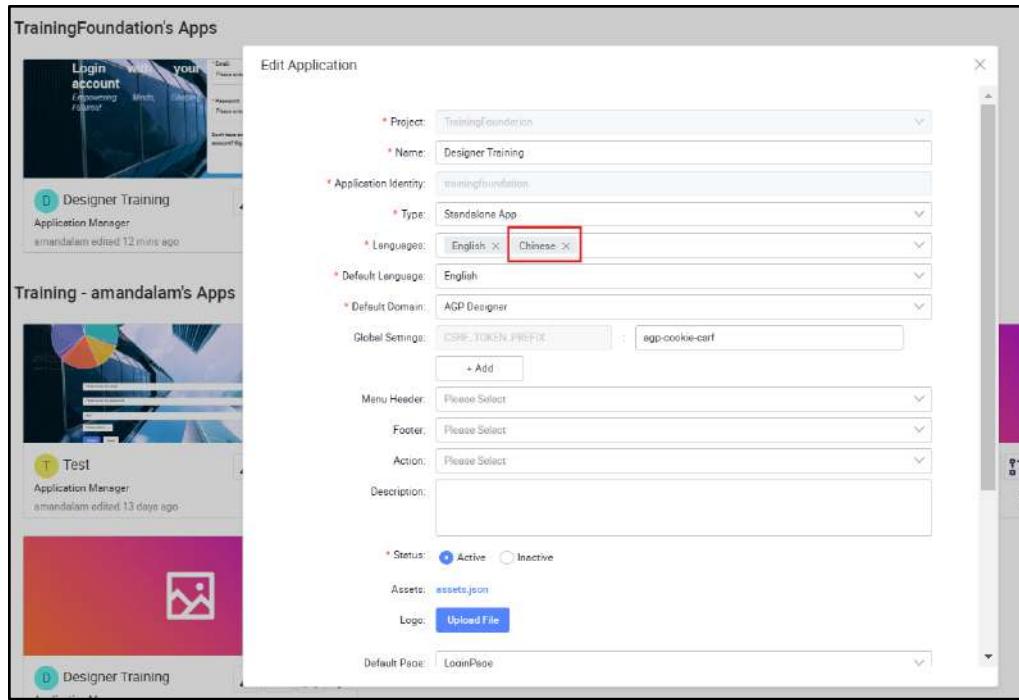
This practical covers the following Learning Objectives:

- Learn how to add multi-language switching functionality to your application using i18n.
- Enhance user accessibility by supporting global audiences with multiple language options.

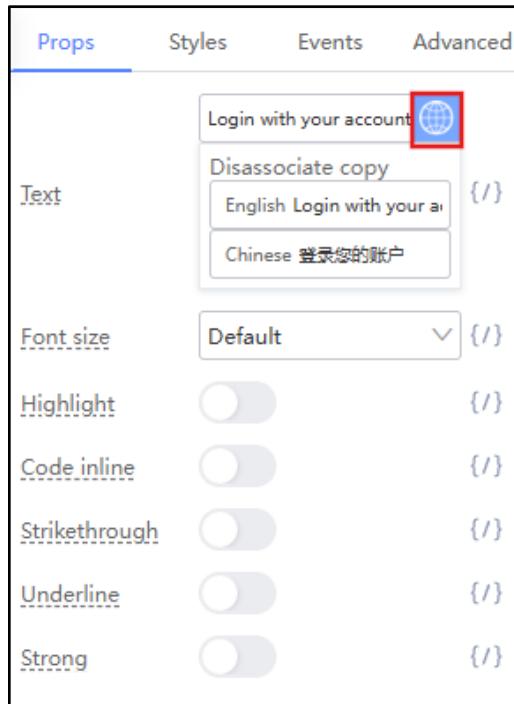
In this practical, you will learn how to integrate translation functionality into your application using the i18n standard. This feature allows users to switch between different languages seamlessly, making your application accessible to a wider, global audience. You'll explore how to add and configure language files, making your product user-friendly across different regions.



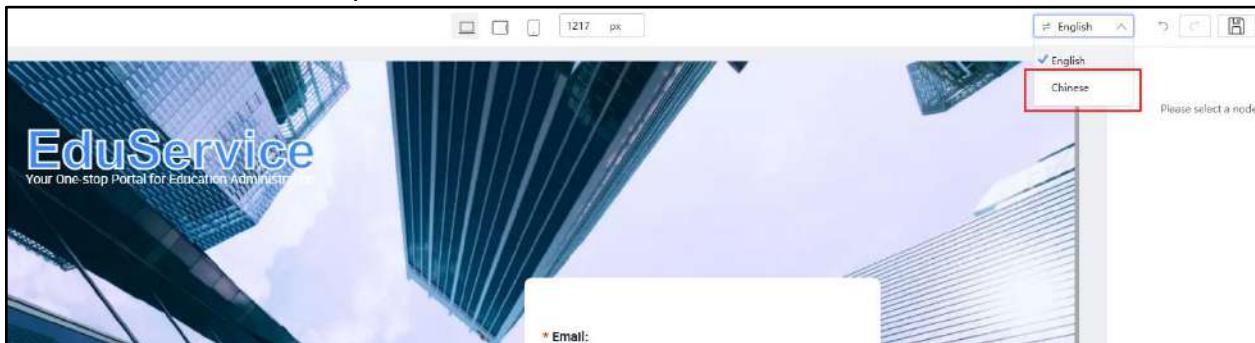
- In the TrainingFoundation project, the “Chinese” language has already been added into the application settings in Languages.



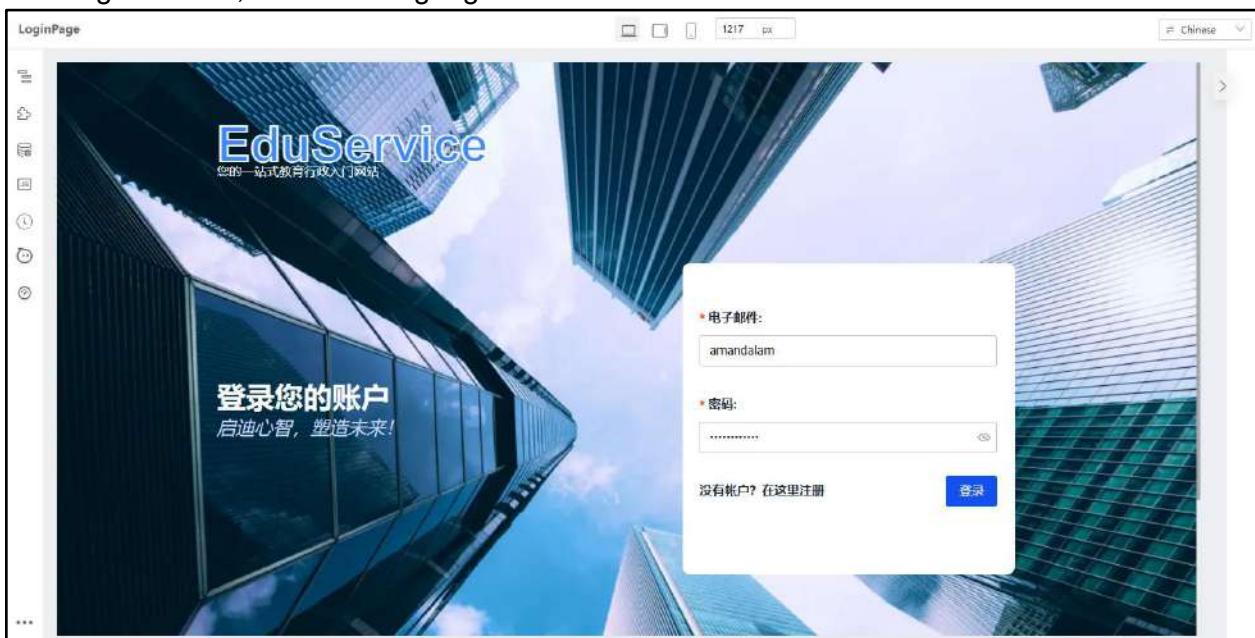
- Each text's Chinese translation has already been pre-filled as well.



- Select “Chinese” in the dropdown



- In Designer mode, note that language is switched to “Chinese”.



- Similarly in Preview mode, note that the page's language can also be switched to “Chinese”.

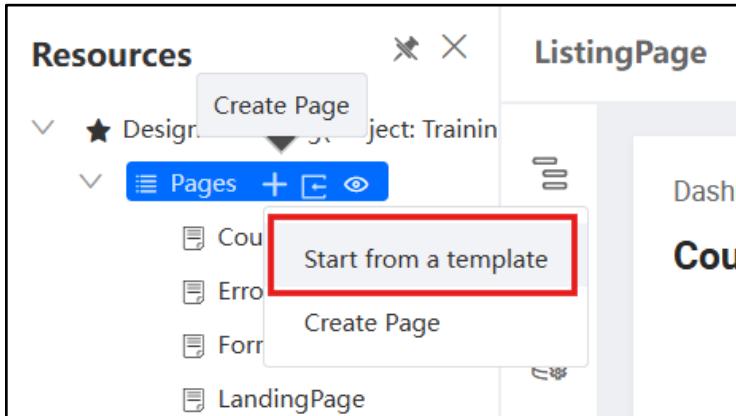
Tutorial 11: Page Import From Template

This tutorial covers the following Learning Objectives:

- Learn how to import pages from pre-designed templates to speed up application development.
- Understand how to customize imported templates to fit your specific application needs.
- Explore the benefits of using templates for consistency and efficiency in your design process.

In this practical, you will learn how to import pre-designed pages from templates, allowing you to quickly start building your application with a solid foundation. We'll guide you through the process of customizing these templates to meet your project's requirements, saving you time and effort in the design phase. By the end of this tutorial, you will be able to leverage templates to create consistent, professional-looking pages in your application while ensuring they align with your specific needs.

- Create an application from a template.



- Select the CRUD Template and select its page.

Create Page from Template

* Category: Website

* Template: CRUD

* Pages:

Cancel Next

- Fill in the necessary details and click **Save**.

Create Page

* Application: Designer Training

* Name: ImportedPage

* Page ID: crud

* Type: Page

Description:

* Status: Active Inactive

* Default Page: Yes No

* Is Protected: Yes No

Privileges: AGP Designer/Anonymous Resource

Cancel Save

- Just like that, your new page is created by using the selected template.

gFoundation)

ImportedPage

Page "ImportedPage" created successfully

Application Management > Job Submission

Job Submission

Job ID: Search Job ID

Submitter: Please select

Description: Description or keywords

<input type="checkbox"/>	Job ID	Email	Submitter	Feedback	Actions
<input type="checkbox"/>	JOBID8312938	sophie.walker@example.com	Sophie Walker	Impressed with the new features!	
<input type="checkbox"/>	JOBID7218945	jack.turner@example.com	Jack Turner	Smooth experience using the platform.	
<input type="checkbox"/>	JOBID5820137	ella.harris@example.com	Ella Harris	User-friendly interface, kudos!	
...					

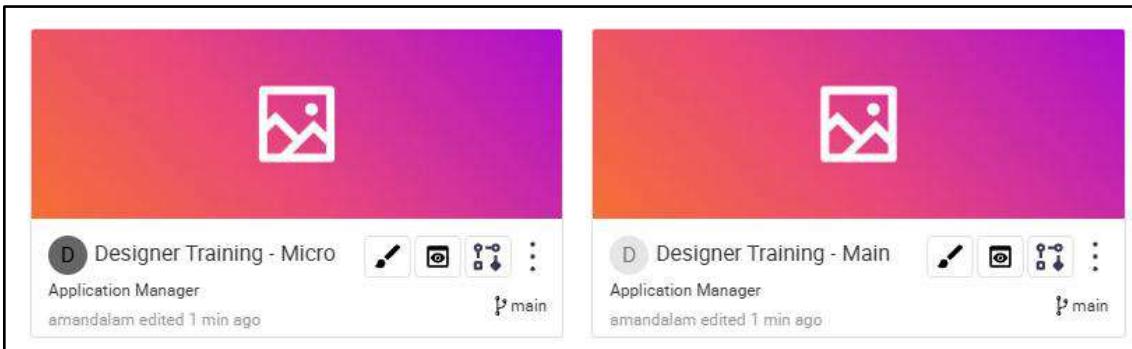
Tutorial 12: Main and Micro Applications

This tutorial covers the following Learning Objectives:

- Understand how to set up and manage a main application and micro-application architecture.
- Learn the advantages of micro-application architecture for scalability and flexibility.

In this tutorial, you will learn how to set up a main application with micro-applications. Micro-application architecture allows you to split your application into smaller, more manageable components, which enhances scalability, flexibility, and performance—an ideal solution for rapidly evolving environments. This aligns with the micro-frontend framework.

- These two Main and Micro applications have been created for you. Your current Designer Training Application that you have been working on is a “Standalone” application type.

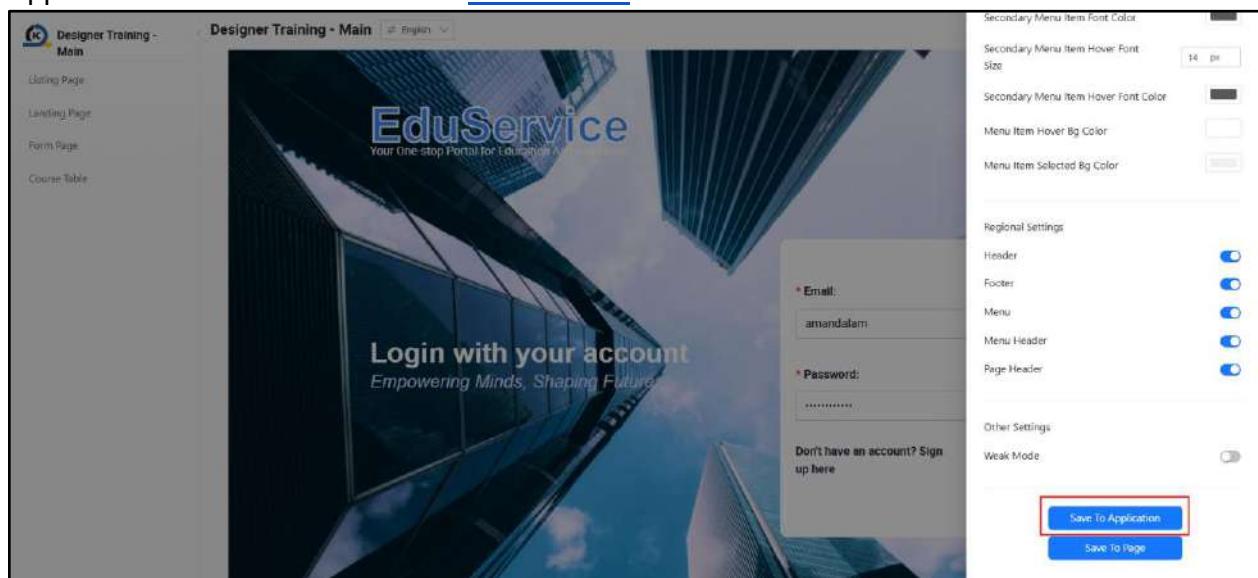


- Generally, the Main application will contain the menu and it will load child Micro application(s). Note that one Main application can be tied to multiple micro applications.

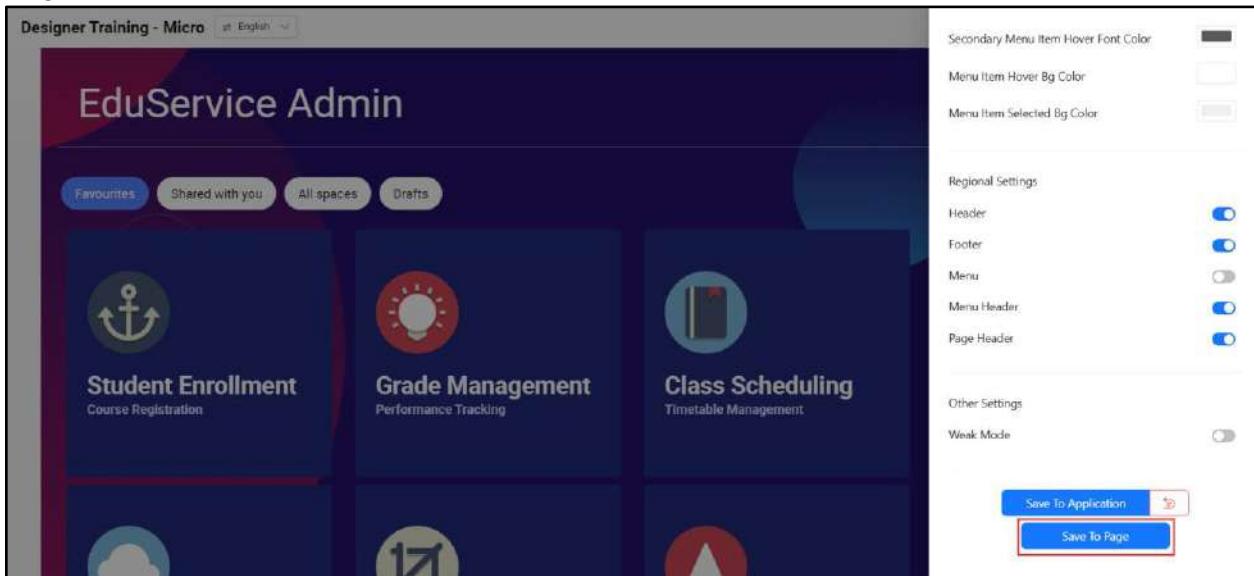
Edit Application

* Project:	TrainingFoundation
* Name:	Designer Training - Main
* Application Identity:	trainingFoundationMain
* Type:	Main App
Micro Applications:	Designer Training - Micro X
* Languages:	English X Chinese X Japanese X
* Default Language:	English
* Default Domain:	AGP Designer
Global Settings:	CSRF_TOKEN_PREFIX : agp-cookie-csrf + Add
Menu Header:	Please Select
Footer:	Please Select

- Navigator should be done in the Main application. Refer back to [Practical 9.2](#) for more details.
- Customization of global layout should be done in the Main application via 'Save to Application' function. Refer back to [Practical 9.2](#) for more details.



- Customization of individual layout should be done in the Micro application via 'Save to Page' function. Refer back to [Practical 9.2](#) for more details.



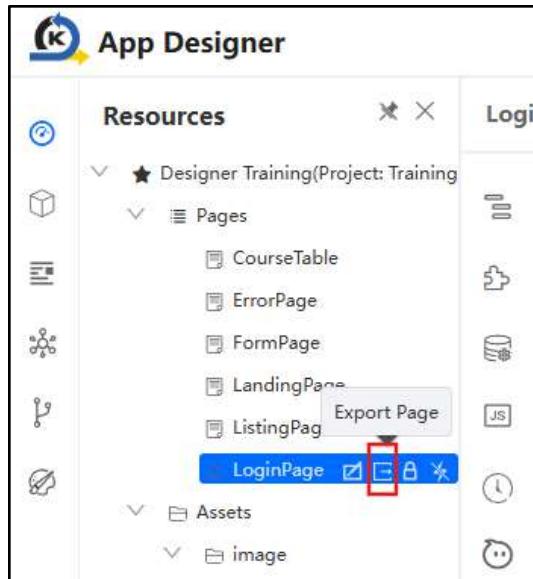
Practical 12.1 Export and Import Pages to Main/Micro Apps (Optional)

This practical covers the following Learning Objectives:

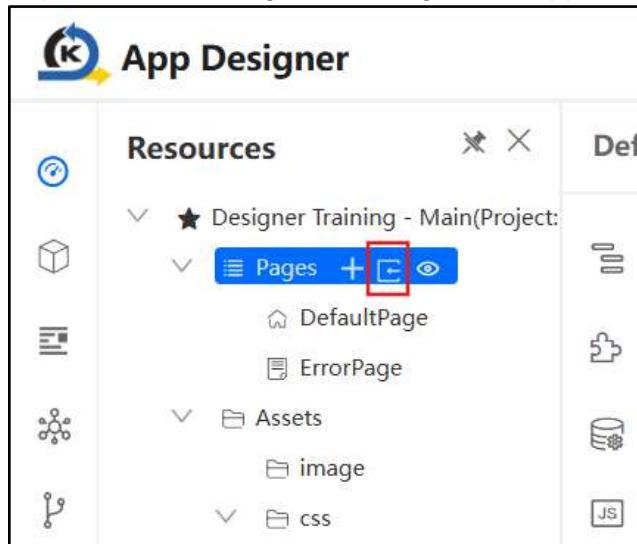
- Understand how to export and import pages in the App Designer.

In this practical, you will learn how to export and import your pages for use. This will be required in development when you need to either duplicate your existing pages in your application for ease of development, or in the case of this tutorial, to import pages into other applications.

- In your Designer Training application, we will first export the login page and import it over to the **Main** application.
 - Export Login Page from your original Designer Training application



- Import into the Designer Training - Main application



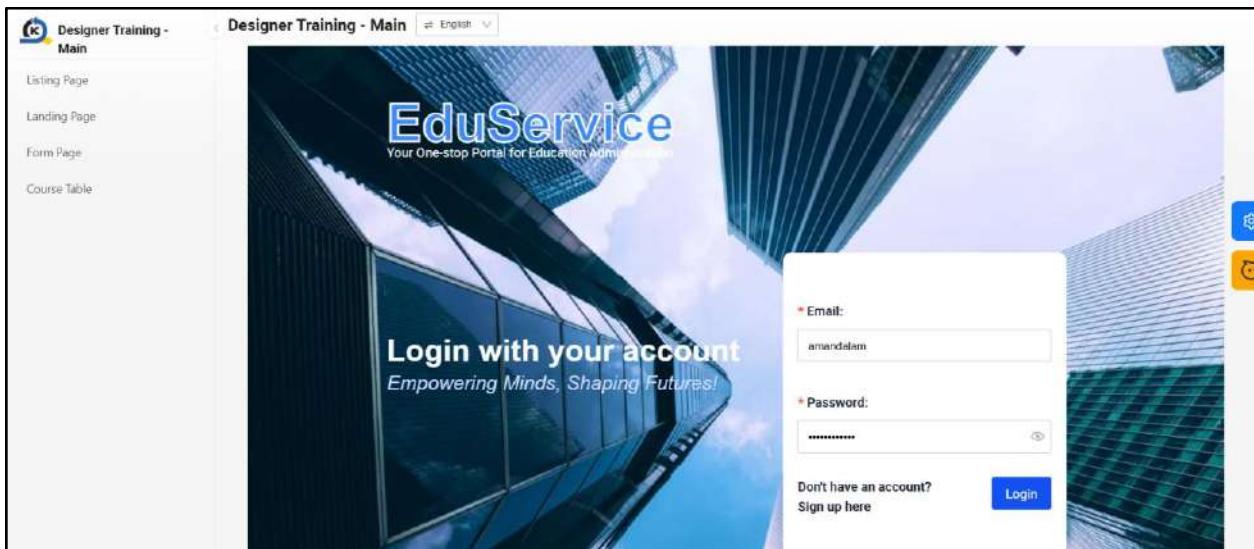
- Do the same for the rest of the pages and import it accordingly as follows. Ensure to save and publish your applications once all imports are done.
 - Designer Training - Main
 - Login Page
 - Designer Training - Micro
 - Listing Page
 - Landing Page
 - Form Page
 - Course Table

- Create the navigator in your main app to point to all 4 pages in the micro app.

Config Navigators								
User Domain	Name	Path	Target	Open Method	Icon	Priority	Privileges	Action
AGP Designer	Listing Page	/listing	/trainingFoundat...	current_tab		1		Edit Delete
AGP Designer	Landing Page	/landing	/trainingFoundat...	current_tab		2		Edit Delete
AGP Designer	Form Page	/form	/trainingFoundat...	current_tab		3		Edit Delete
AGP Designer	Course Table	/course	/trainingFoundat...	current_tab		4		Edit Delete

[Create](#) [Export](#)

- Once you preview your main app, you will be able to navigate to the different pages via the menu bar.

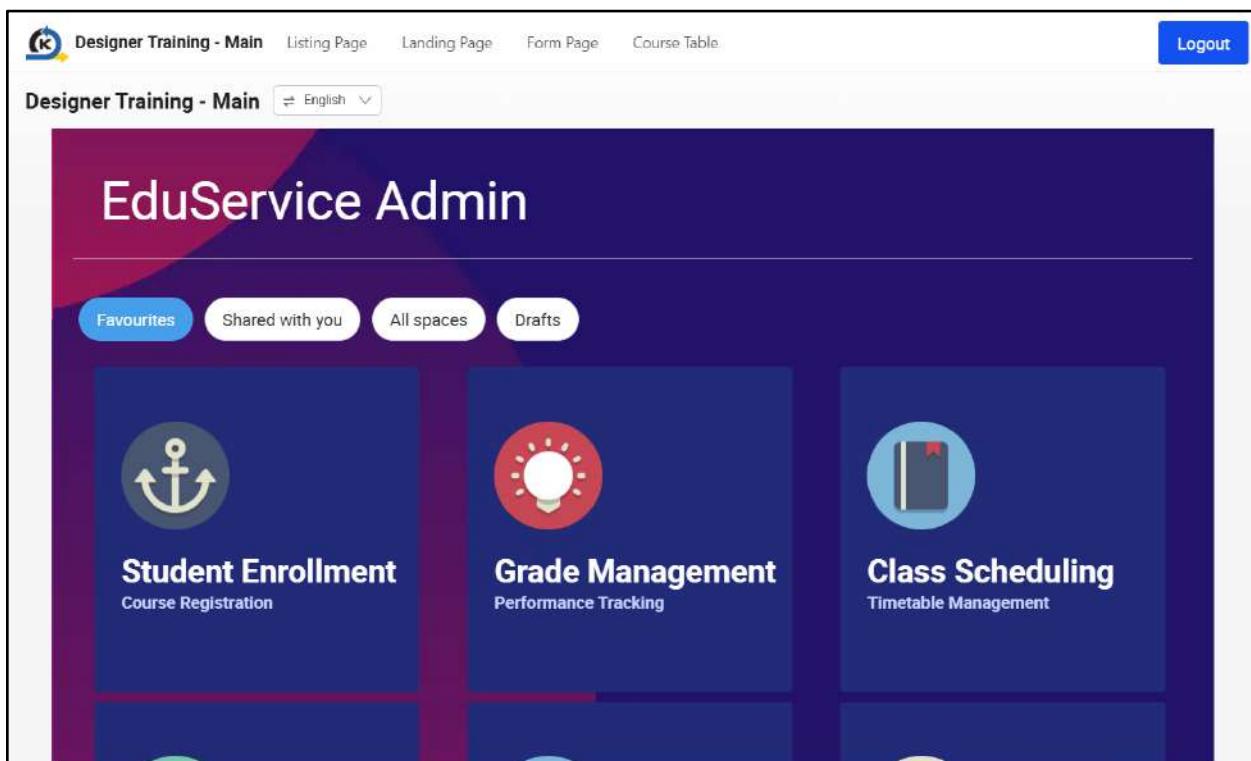


Practical 12.2 Add Action Page Type to Main App (Optional)

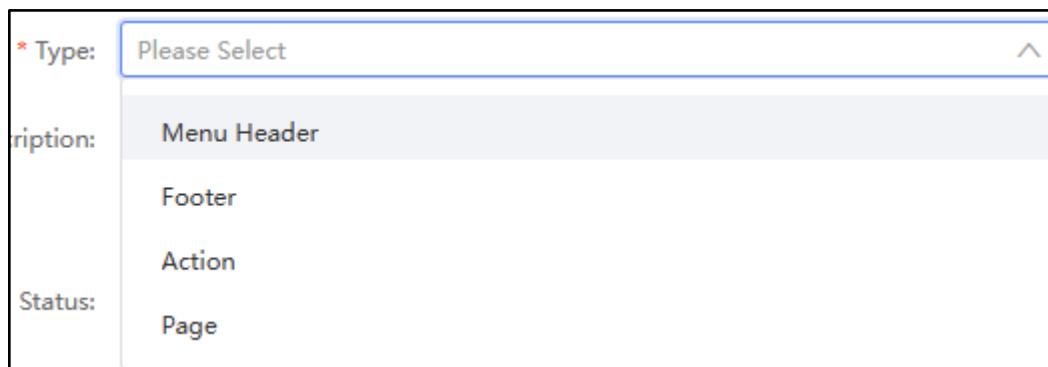
This practical covers the following Learning Objectives:

- Learn how to utilise the ‘Action’ type page in your applications.

In this practical, you will learn how to create a ‘logout’ button which will persist across all your application pages. In order to avoid having to duplicate global components that you want to persist on all pages (e.g. profile, logout, language toggle), you can create an ‘Action’ type page in your main application. In addition to action components, you can also create ‘Menu Header’ and ‘Footer’ type pages to persist the design of headers and footers across your application respectively.



- There are a few types of pages you can create for different purposes. In previous tutorials, you have only been using the standard 'Page' type.
 - **Menu Header:** A navigation element used to group related pages under a common section in the application's menu. It helps organize and structure the navigation for better usability.
 - **Footer:** A section that appears at the bottom of the application's interface, typically used for displaying copyright information, quick links, or additional navigation options.
 - **Action:** A page type designed for action components that persist across all pages in the application, such as profile settings, logout, and language toggle. These components remain accessible regardless of the user's navigation.
 - **Page:** The standard content page where users can design and display UI components, input forms, data tables, and other interactive elements. This is the primary type used for building application interfaces.



- For this tutorial, create a new ‘Action’ page in your main app.

Create Page

* Application: Designer Training - Main

* Name: Action

* Page ID: action

* Type: Action

Description:

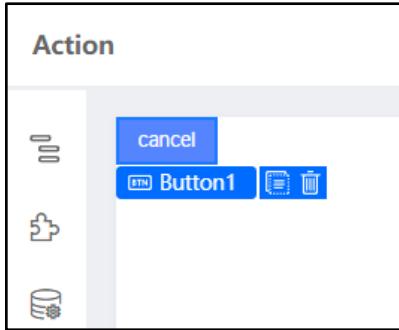
* Status: Active Inactive

* Default Page: Yes No

* Is Protected: Yes No

Privileges: Anonymous Resource X

- Drag a Button component into the page.



- Set the button with the following styles and code. Save and publish your page changes.



Props

Content: Logout

Button size: Large

Styles

background-color: #1352f1;

Events

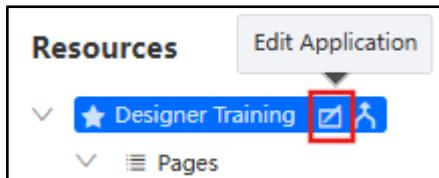
Component native event: OnClick

```

onClick(){
  this.utils.navigateTo('trainingFoundationMain', 'LoginPage');
}

```

- In the Resource panel, click on the 'edit' icon of your main app.



- Under Action, select the 'Action' page that you have just created and click 'Save'.

Edit Application

* Project: TrainingFoundation

* Name: Designer Training - Main

* Application Identity: trainingFoundationMain

* Type: Main App

Micro Applications: Designer Training - Micro X

* Languages: English X Chinese X Japanese X

* Default Language: English

* Default Domain: AGP Designer

Global Settings: CSRF_TOKEN_PREFIX : agp-cookie-csrf

+ Add

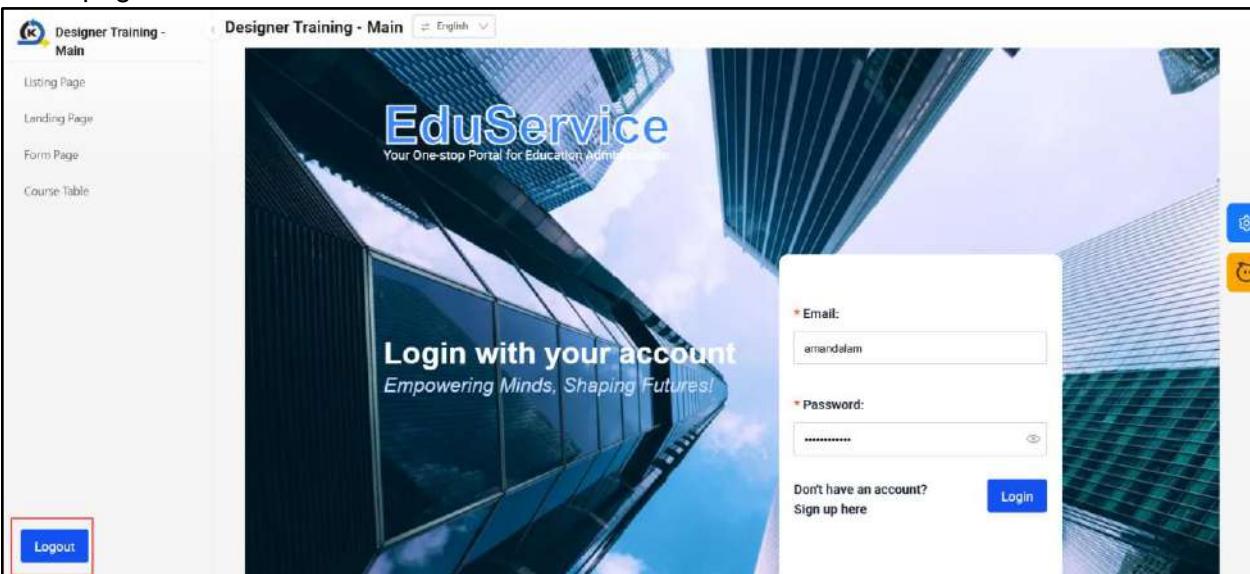
Menu Header: Please Select

Footer: Please Select

Action: Action

 Application "Designer Training - Main" updated successfully

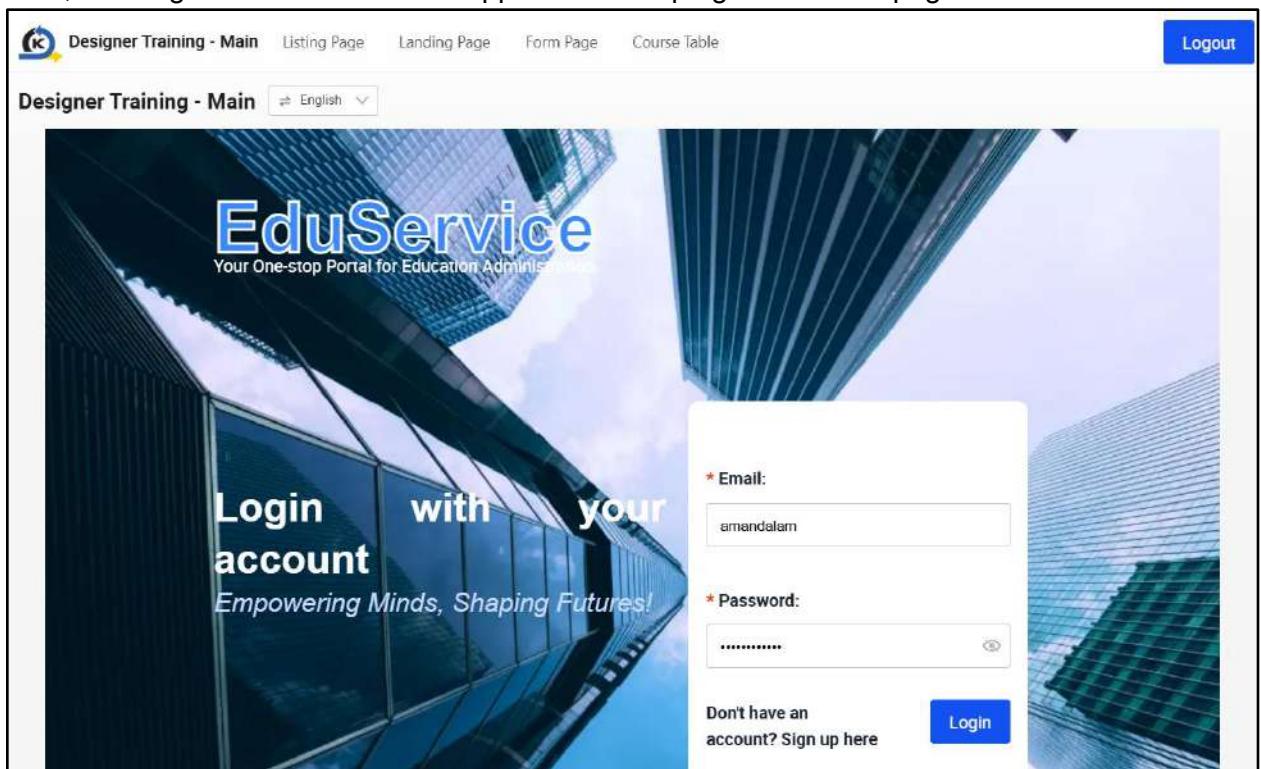
- In preview mode of your main app, notice that the 'Logout' button added now forms part of the page header.



- Click on the Setting icon to customize your navigator menu. Select 'Top Menu Layout' for the Navigation Mode and click 'Save To Application'.



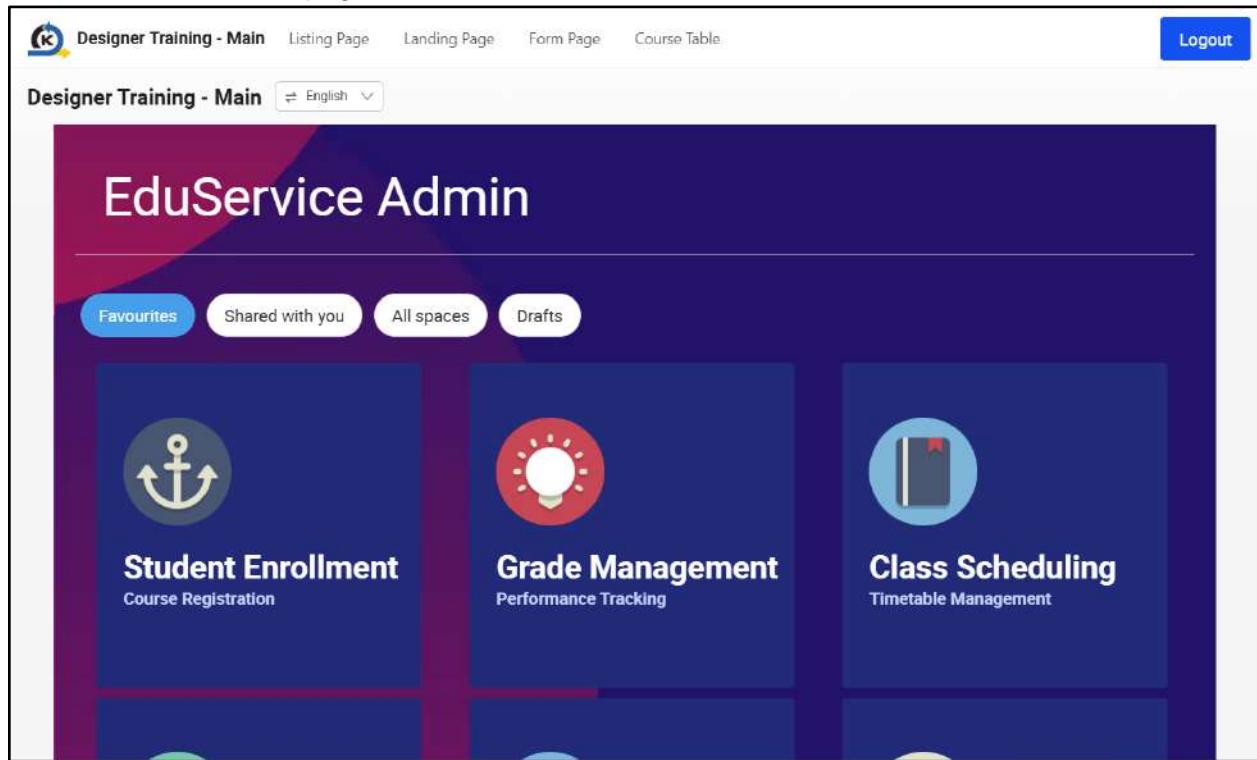
- Now, the 'Logout' action button will appear at the top right across all pages.



- However, you may not want your action buttons to appear on all pages. In this case, it would not make sense to have this button (or access to the other pages via the menu bar) on the first Login page. Hence, you can choose to customize your navigator menu further by disabling the layout for the login page and clicking on ‘Save To Page’.

The screenshot illustrates the process of saving a page layout. At the top, the navigation bar includes links for 'Designer Training - Main', 'Listing Page', 'Landing Page', 'Form Page', and 'Course Table'. Below the navigation, the main content area displays the 'EduService' logo and tagline 'Your One-stop Portal for Education Administration'. On the right side of the screen, there is a configuration panel titled 'Layout Scope: Application' with two settings: 'Enable Layout' (which is turned off, indicated by a red box around the toggle switch) and 'Page style setting' (with a dark blue square icon). Below this panel, a large button labeled 'Save To Page' is visible. The bottom half of the screenshot shows the resulting login page, which has a similar header and background image. The login form contains fields for 'Email' (with the value 'amandalam') and 'Password', along with a 'Login' button and a link for users who don't have an account.

- Once you have logged in, you will be able to see the navigator menu and ‘Logout’ action button across all other pages.



Tutorial 13: Build screen skeleton

This tutorial covers the following Learning Objectives:

- Learn how to build the skeleton structure of a screen within your application.
- Understand how to create a layout framework that can be easily customized and populated with content.
- Explore the process of designing a flexible and responsive screen skeleton that ensures scalability and consistency.

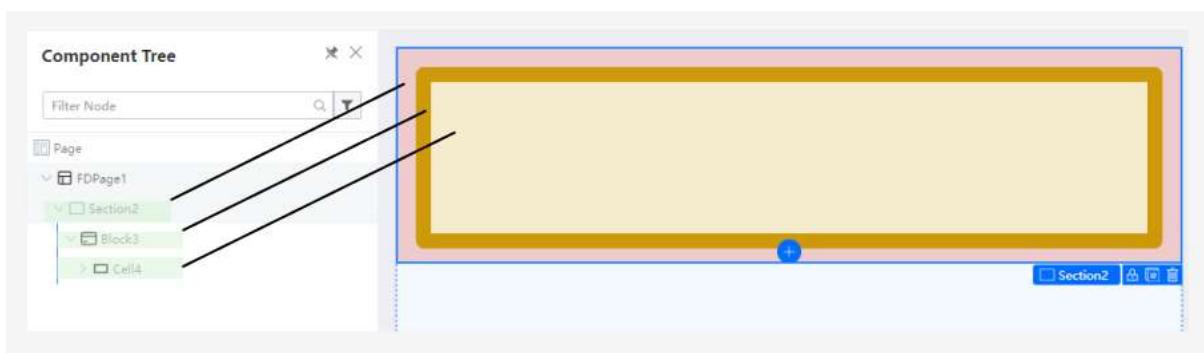
In this tutorial, you will learn how to build the basic framework, or skeleton, of a screen within your application. The screen skeleton serves as the foundation upon which you will add content and functionality. We will guide you through the process of structuring the layout, ensuring it is flexible and can easily adapt to different screen sizes. By the end of this tutorial, you'll be able to create a consistent and scalable screen layout that forms the core of your application's user interface.

Key Concepts

We will start by introducing the hierarchy structure of Section, Block and Cell ([Parent-Child relationship](#)).

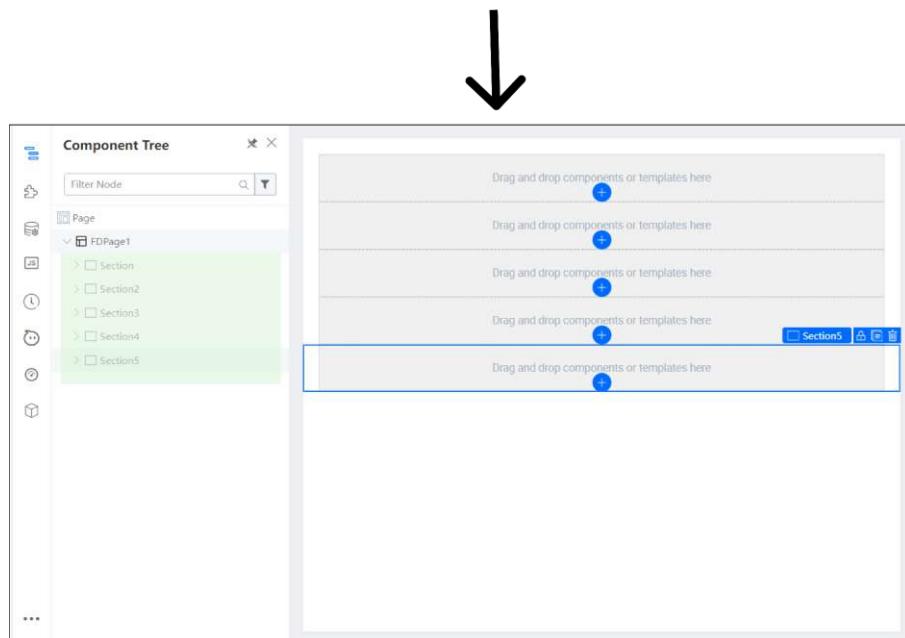
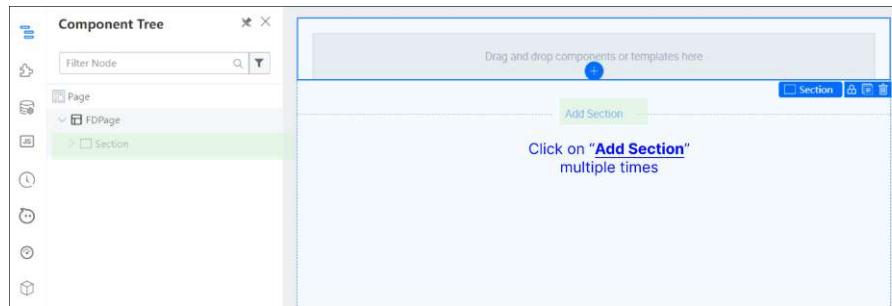
- Section can contain Block
- Block can contain Cell, but cannot contain Section
- Cell cannot contain Section and Block

Hierarchy:



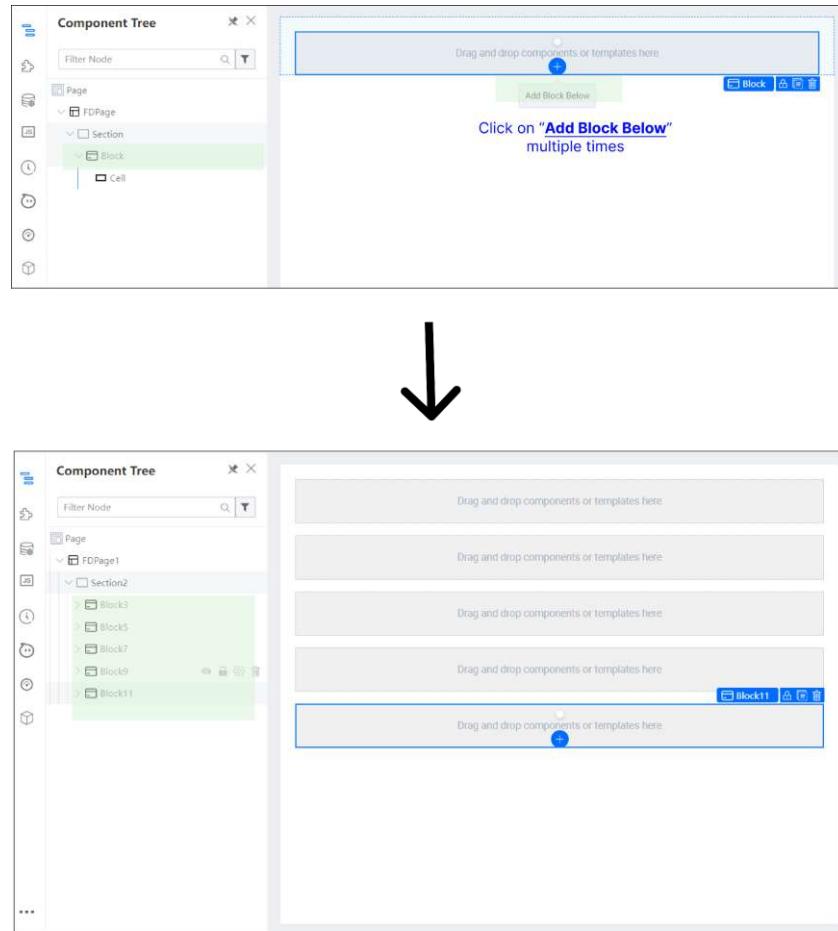
Layout: Section vs Block

Section:



- **Organization:** Takes up the **entire width (100%)** and provides a clear and structured way to organize content. Each section can represent a distinct topic or category, making it easier for users to navigate and understand the layout.
- **Customization:** Sections allow for more **customization** options in terms of layout, styling, and functionality. You can apply **different designs, backgrounds, and behaviors to each section** as needed.
- **Scalability:** Sections offer scalability, allowing you to easily add or remove content sections without affecting the overall layout. This flexibility is particularly useful for websites or applications with dynamic or evolving content.
- **Readability:** By dividing content into sections, you can **improve readability and comprehension** for users. Clear demarcations between sections help users understand the flow of information and locate relevant content quickly.

Block:



- **Granularity:** Using **multiple blocks** allows for a more granular approach to content management. Each block can represent a smaller unit of content or functionality, giving you finer control over the layout and presentation.
- **Flexibility:** Blocks offer **flexibility in arranging content** within a section. You can easily rearrange blocks or add new ones without restructuring the entire layout, providing greater adaptability to changing requirements or user preferences.

Example (Using Section)

Using the <https://amazon.sg> website as an example, we will explain why it is preferable to use Sections over Blocks

The screenshot shows the homepage of Amazon Singapore. At the top, there's a banner for a promotion: "Get S\$10 off min. spend S\$100 Up to 70% off on Home & Kitchen Ready set Raya". Below this, the page is organized into several sections:

- Get S\$10 promo credit with S\$120 eGift Card purchase**: Shows three gift cards with various designs.
- Today's Top Deal**: Features a UGREEN Nexode 100W USB C Charger Plug 4-Port with a 10% discount from \$48.99 to \$44.99.
- Shop for your home essentials**: Categories include Cleaning Tools & Vacuums, Home Storage, Home Decor, and Bedding.
- Toys on clearance**: Shows a pink playhouse toy.
- Shop 'buy 1, get 1 free' deals at Watsons**: Features products from Kiehl's LAB and Watsons.
- Items for Baby Wishlist**: Categories include Nursery, Clothing, Prams & Strollers, and Toys.
- Discover these luxury beauty products for you**: Categories include Skincare, Makeup, Nails, and Fragrances.
- Enjoy 2hr same-day delivery from Little Farms**: Categories include Little Farms, Fruits & Vegetables, Meats, and Farm-fresh.
- All the ways to save at Amazon**: Icons for Today's Deals, Lightning Deals, 50% off or more, Bank Promotions, Deals under S\$25, Deals from Amazon US, and Deals from Amazon JP.

At the bottom, there's a footer with links to "Get to Know Us", "Connect with Us", "Make Money with Us", and "Let Us Help You". It also includes links to various Amazon services like AWS, Kindle Direct Publishing, Goodreads, IMDb, and Shopshop. The footer ends with a copyright notice: "Conditions of Use Privacy Notice Interest-Based Ads © 1996-2024, Amazon.com, Inc. or its affiliates".

Heading Section

SubHeading Section

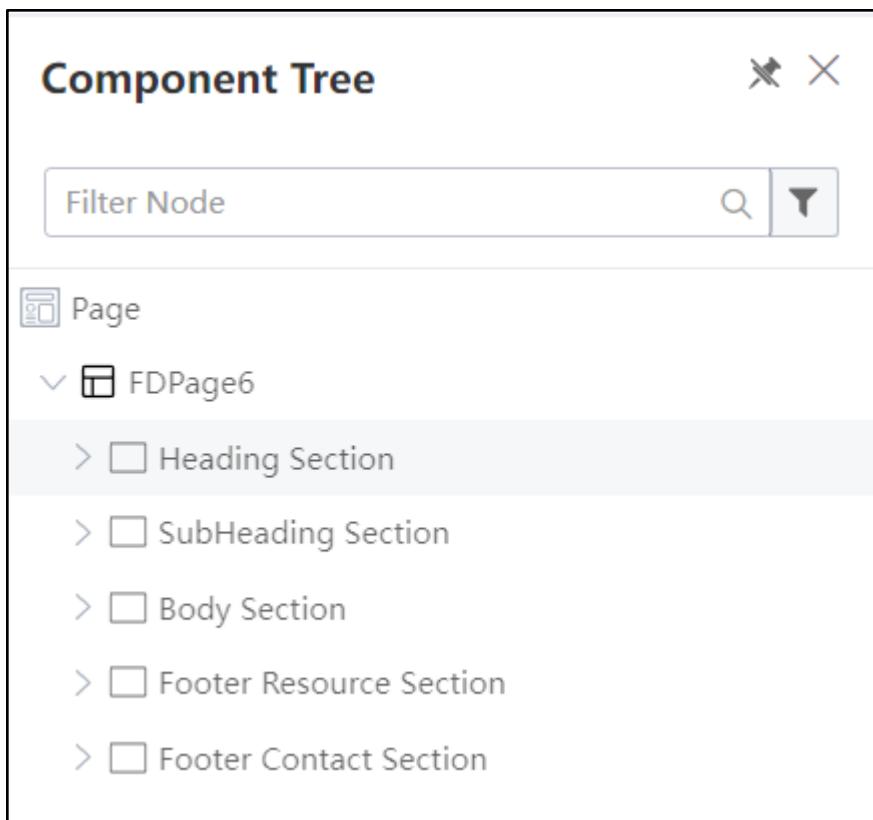
Body Section

Footer Resource Section

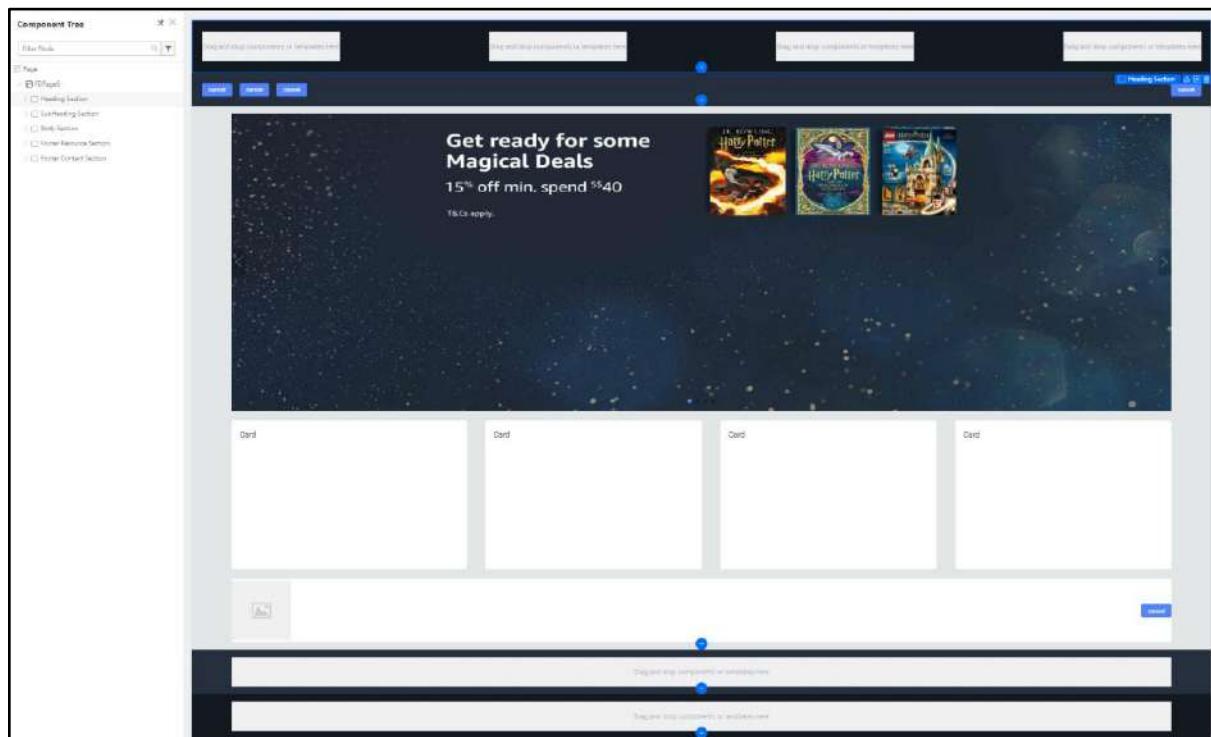
Footer Contact Section

The image shows the Amazon.in homepage with several sections highlighted by red boxes and labeled for A/B testing:

- Heading Section:** The top navigation bar.
- SubHeading Section:** The main search bar area.
- Body Section:** The main content area featuring promotional banners for eGift cards, a power adapter, home essentials, and clearance toys.
- Footer Resource Section:** The footer navigation bar with links to About Us, Connect with Us, Make Money with Us, and Let Us Help You.
- Footer Contact Section:** The footer contact information section with links to Amazon Web Services, Amazon Fresh, Goodreads, and Kindle.

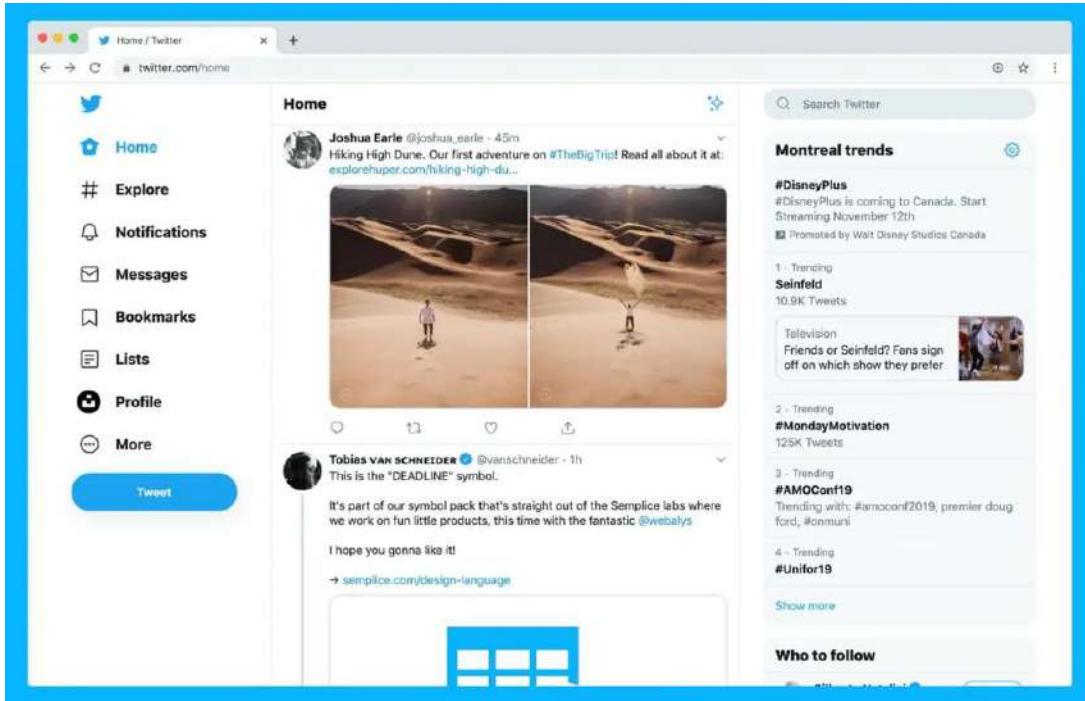


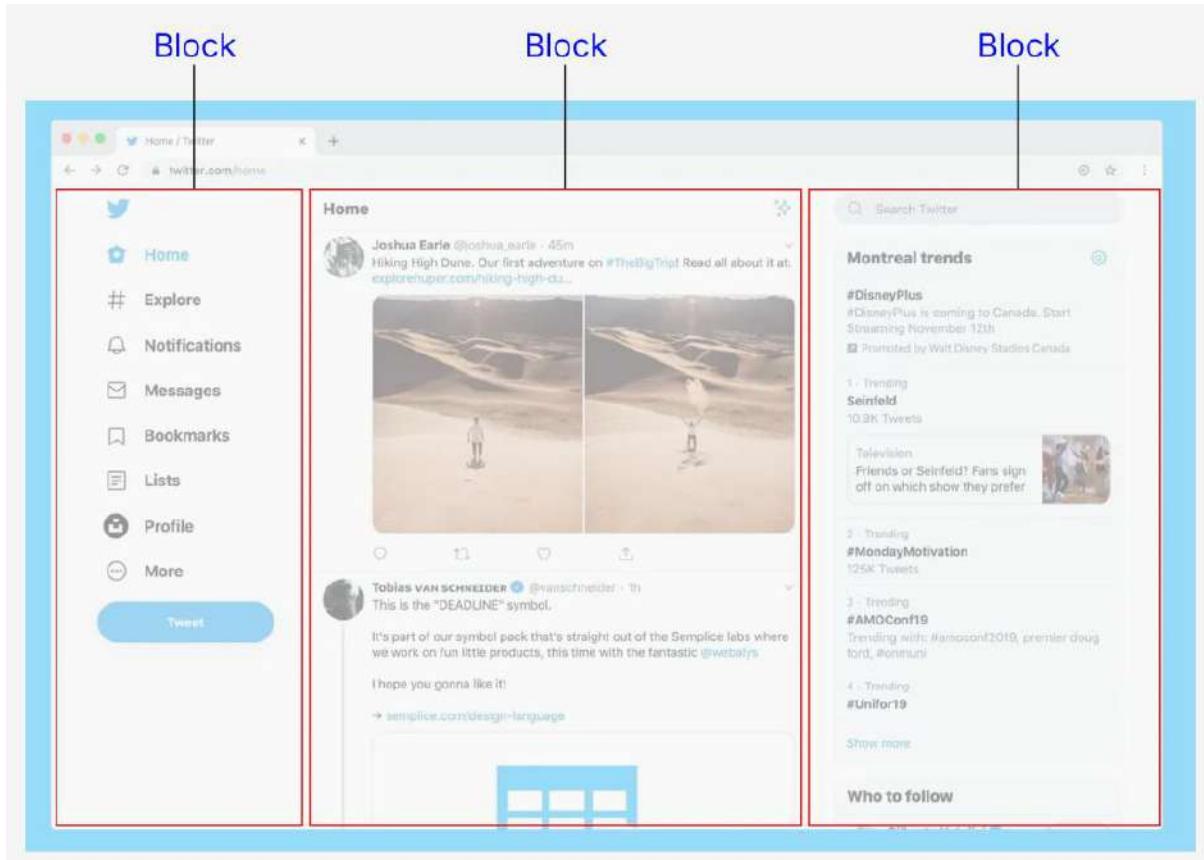
Creating the skeleton of amazon.sg website in KAIZEN



Example (Using Block)

Using the Twitter (<https://twitter.com>) website as an example, we will explain why it is preferable to use Blocks over Sections





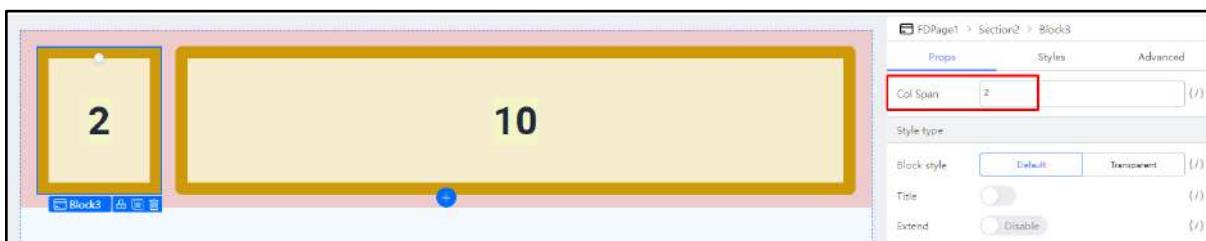
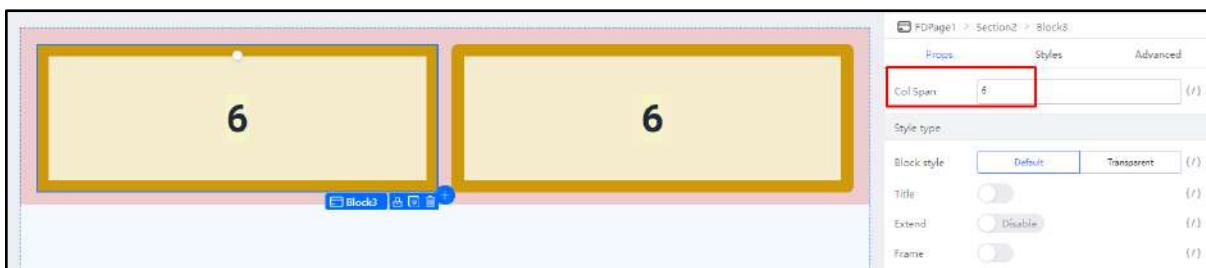
Layout: Block vs Cell

Block:

- Supports Vertical Cut option



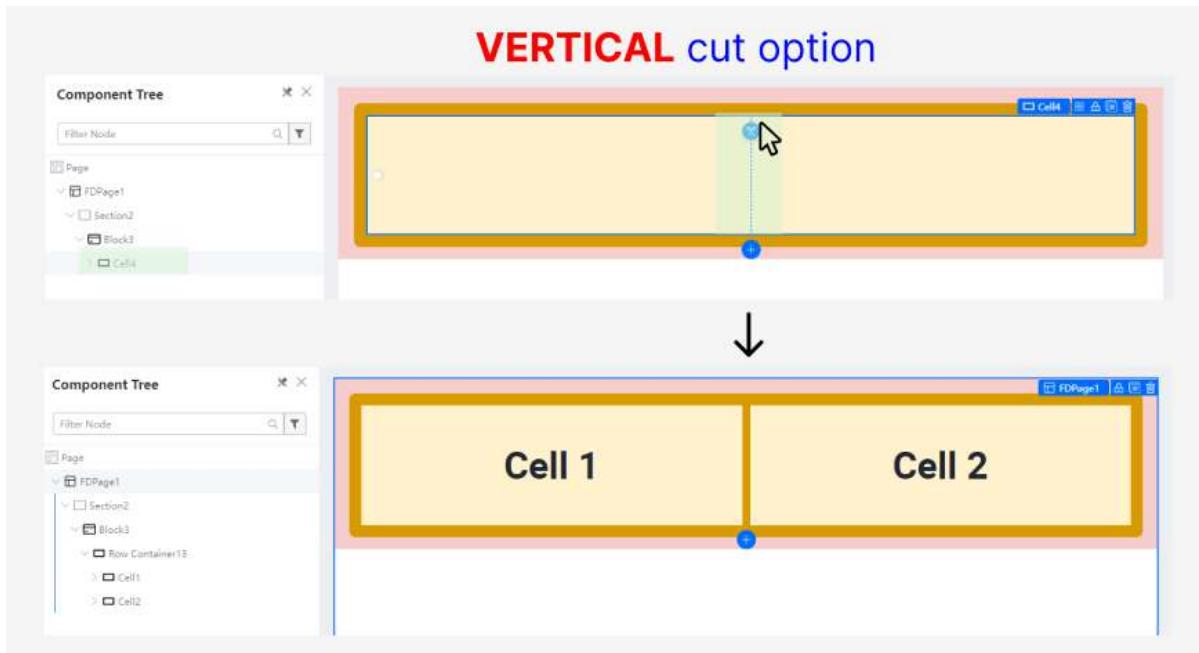
- Uses structured **grid layout** through configuring “Col Span” property



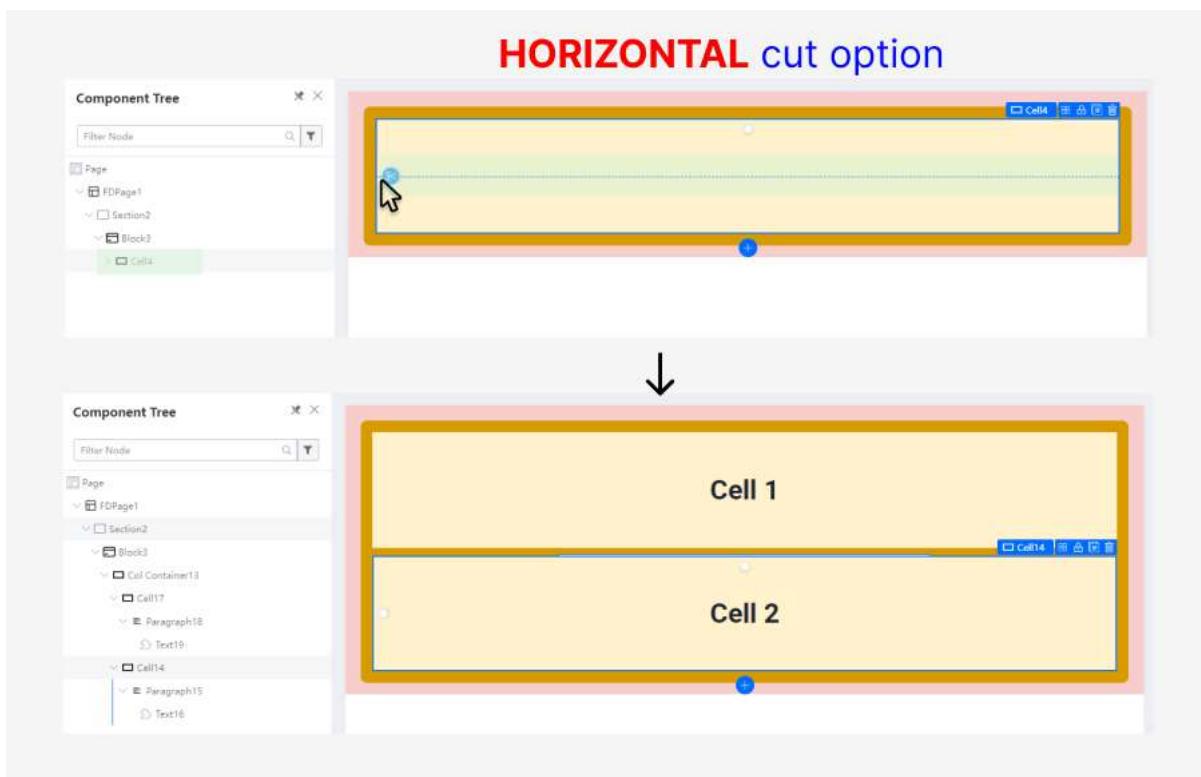
- **Granularity:** Using **multiple blocks** allows for a more granular approach to content management. Each block can represent a smaller unit of content or functionality, giving you finer control over the layout and presentation.
- **Flexibility:** Blocks offer **flexibility in arranging content** within a section. You can easily rearrange blocks or add new ones without restructuring the entire layout, providing greater adaptability to changing requirements or user preferences.

Cells:

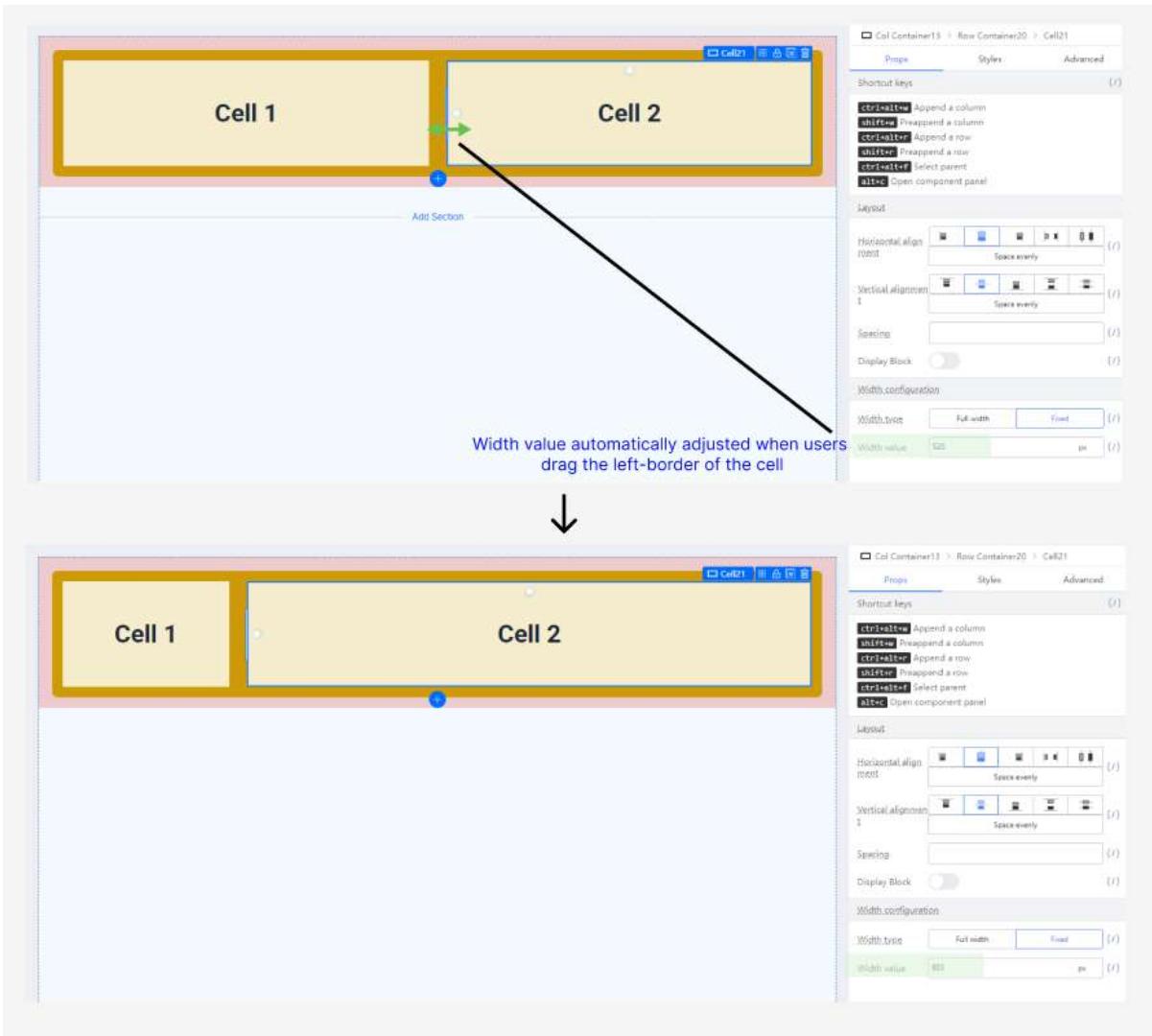
- Supports Vertical and Horizontal Cut option



HORIZONTAL cut option



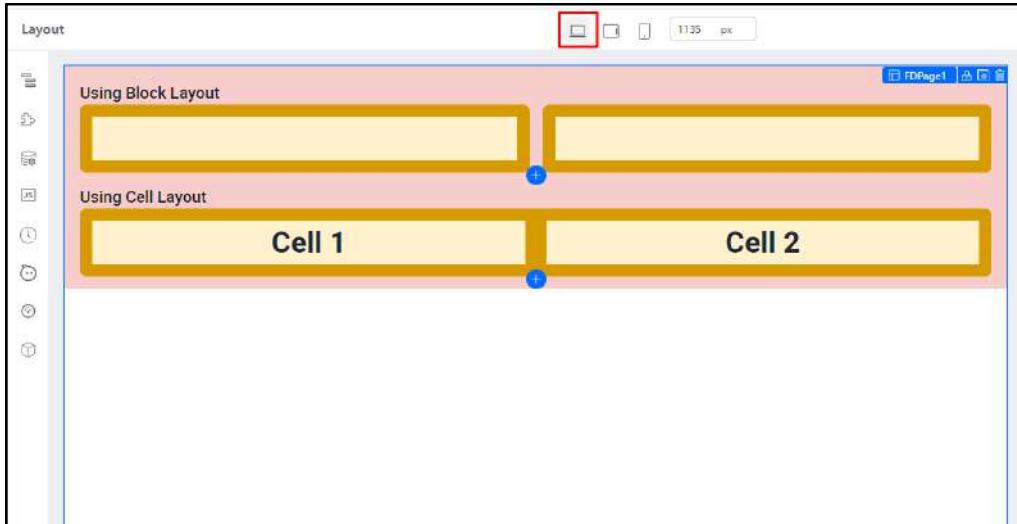
- Uses **flex layout** through adjusting the span (width)



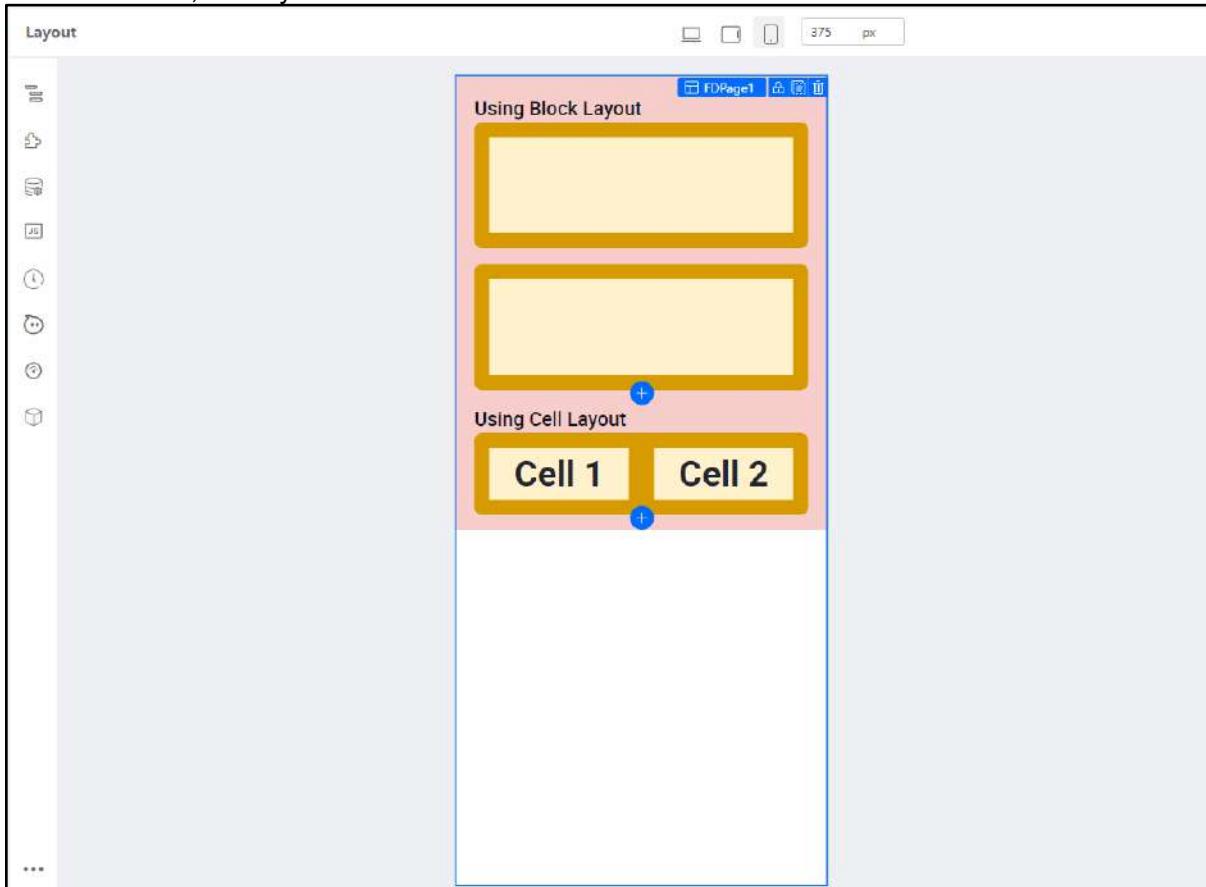
- **Modular Layouts:** Cells allow for modular design by segmenting content into smaller units within a Block.
- **Flexible Styling:** Provides flexibility in styling individual cells, including formatting, borders, padding, and other visual properties.
- **Tabular Data Presentation:** Ideal for presenting tabular data with rows and columns, enhancing readability and organization.

Block VS Cell

In Desktop view, the layout structure for Block and Cell looks like this:



In Mobile view, the layout structure looks like this:



By default, using Block will have a responsive layout wrap behavior, while Cell does not.

Example: Webpage walkthrough example

We will use the following webpage as an example to walk you through the methodology when creating page skeleton:

The screenshot displays a web-based application for managing tasks and communications. On the left, there's a sidebar with a purple header containing a hand holding a smartphone icon and the text "Task Left 4". Below this is a button labeled "Buy More Task". To the right of the sidebar are three status indicators: "Done" (12), "In Progress" (4), and "In Queue" (2). The main content area features a table listing tasks assigned to various designers. The columns are: Designer, Task, Status, Priority, and Last Update. The tasks listed are:

Designer	Task	Status	Priority	Last Update
Alana Zeinski	Landing Page Design	Done	High	Yesterday
Aminah Sulistyo	Instagram Post Design	In Queue	Low	Yesterday
David Cooper	Mobile App Design	On Review	High	1 hour ago
Franklin Gothic	Custom Font Design	In Progress	Medium	Yesterday
Garfield Houston	Developing Landing Pages	Done	High	Apr 1, 2022
Jarjit Singh	Pitch Deck Design	On Review	Low	2 hours ago
Norman Nolan	Logo Design	On Review	High	2 hours ago
Alana Zeinski	Landing Page Design Phase 2	In Queue	Medium	6 hours ago
Jarjit Singh	Digital Banner Design	In Queue	Low	Apr 1, 2022
Franklin Gothic	Logo Design	In Queue	Low	Apr 1, 2022

On the right side, there are three sections for "On Review" tasks:

- Mobile App Design** by David Cooper: Hi Andrew, I've set the figma file, and you can check my first preview! [Figma link](#)
- Home Preview.jpg** by David Cooper: 1.1 MB [Download](#) 1 hour ago
- Pitch Deck Design** by Jarjit Singh: Hi Andrew, let's check my first design [Lokotrop pitch deck v1.pptx](#) 12.1 MB 2 hours ago
- Logo Design** by Norman Nolan: Hi Andrew, let's check my first alternative [Healthcare Logo Design](#) Figma link 2 hours ago

Step 1: Create layout

*To illustrate the example clearer, I will apply background color to Section, Block and Cell

The screenshot shows a digital workspace interface. On the left, there is a section titled "Block" containing a dashboard with a user profile picture, task counts (4, 12, 4, 2), and a "Buy More Task" button. Below this is a list of tasks assigned to various users. On the right, another section labeled "Block" contains a "On Review" board with several cards. A callout box highlights the "On Review" board area.

- Cut Block into two
- Set column span to 9 and 3 respectively

A screenshot of the digital workspace showing the "On Review" board from the previous step. The board has a yellow border around its content area. A callout box highlights this yellow border.

Step 2: Divide cell horizontally for left block

The screenshot shows the "Left Block" section. It has been divided into two horizontal cells. The top cell contains a dashboard with a user profile picture, task counts (4, 12, 4, 2), and a "Buy More Task" button. The bottom cell contains a list of tasks assigned to various users. A callout box highlights the top cell area.

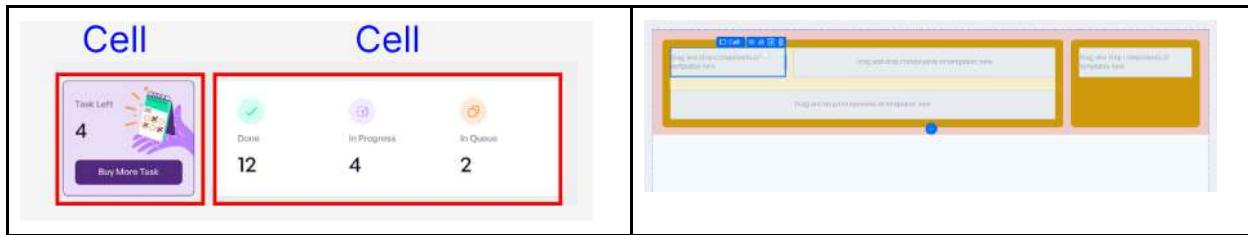
- Cut the Cell horizontally
- Apply a spacing of 25 between the cell

A screenshot of the digital workspace showing the "Left Block" section. The top cell has a yellow border. A callout box highlights this yellow border.

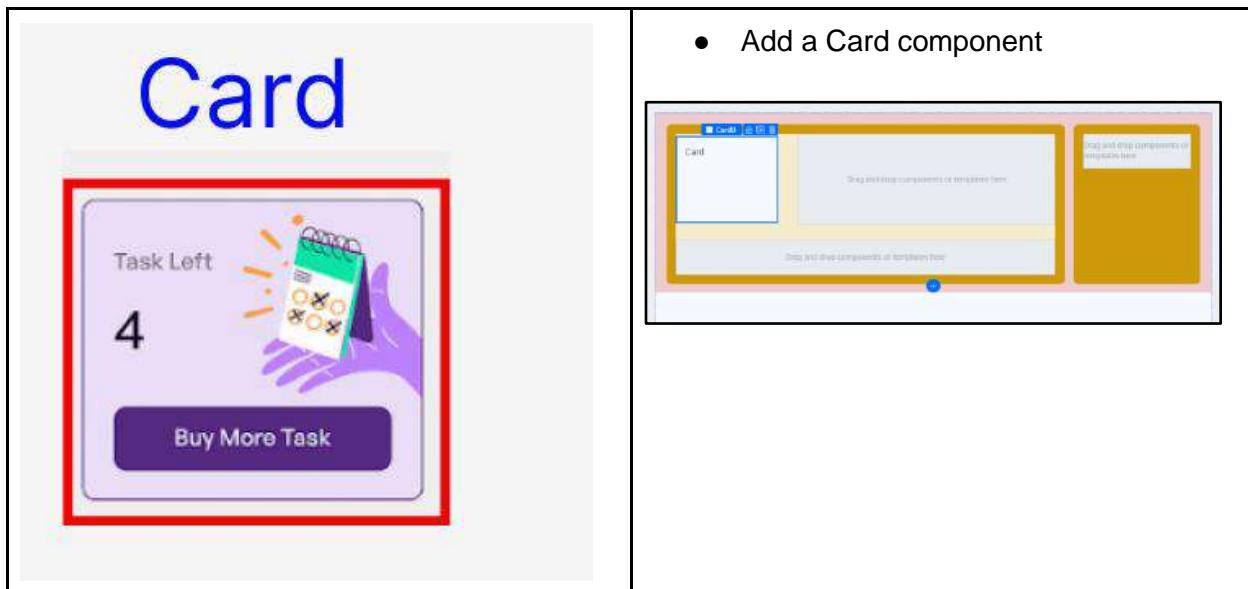
Step 3: Split cell into two

The screenshot shows the "Left Block" section. The top cell has been split into two vertical cells. The left cell contains a dashboard with a user profile picture, task counts (4, 12, 4, 2), and a "Buy More Task" button. The right cell contains a list of tasks assigned to various users. A callout box highlights the left cell area.

- Cut the Cell vertically
- Set the width to 30%



Step 4: Add a card component



Step 5: Add 3 Card components



Step 6: Add a table component

Table

Designer	Task	Status	Priority	Last Update
Ariana Zelenka	Landing Page Design	Done	High	Yesterday
Aminah Sulisyo	Instagram Post Design	In Queue	Medium	Yesterday
David Cooper	Mobile App Design	On Review	High	3 hours ago
Franklin Ortiz	Custom Font Design	In Progress	Medium	Yesterday
Garfield Houston	Developing Landing Pages	Done	Medium	Apr 1, 2023
Jayit Singh	Pitch Deck Design	On Review	Medium	2 hours ago
Norman Nolan	Logo Design	On Review	High	2 hours ago
Ariana Zelenka	Landing Page Design Phase 2	In Queue	Medium	2 hours ago
Jayit Singh	Digital Banner Design	In Queue	Medium	Apr 1, 2023
Franklin Ortiz	Logo Design	In Queue	Medium	Apr 1, 2023

- Add a table component

The screenshot shows a user interface element consisting of a table and a search bar. The table has five columns: Designer, Task, Status, Priority, and Last update. The data in the table is identical to the one shown in the previous screenshot. Below the table is a search bar with the placeholder text "Search".

Step 7: Add components to the right block

Right Block

On Review 3

Mobile App Design
David Cooper
Hi Andrew, I've set the figma file, and you can check my first preview ✨
Healthcare Mobile App (figma file)
Home Preview.jpg (1.1MB)
1 hour ago

Pitch Deck Design
Jaij Singh
Hi Andrew, let's check my first design
Lokotrek pitch deck v1.pptx (51.1MB)
2 hour ago

Logo Design
Norman Nolan
Hi Andrew, let's check my first alternative
Healthcare Logo Design (figma file)
2 hour ago

- Add a Box component
- Followed by 3 Card components

** We have included few other examples below to demonstrate the drawing of screen skeleton

Example

Building Screen skeleton examples

Component Tree

Filter Node

Page FORpage18 Section19 Section22 Section22

FOR INDIVIDUALS

Drag and drop components or templates here

Component Tree

Filter Node

Page FORpage18 Section19 Section22 Section22

ANIMAL & VETERINARY MEDICAL BOARD OF SINGAPORE

A one-stop personalised portal to manage your dog licence(s.)

FOR INDIVIDUALS

Login with Singpass

Login with PALS Account

First time? Create New Profile

Renew License without Login

FOR ORGANISATIONS

Pet Shops/Farms, Government Agencies

Login with Singpass

Login with PALS Account

Contact Us Feedback User Guide Feedback

Report Vulnerability Privacy Statement Terms of Use Rate this e-Service Site-Map

©2024 Government of Singapore

Developer Materials

In the upcoming tutorials, we will shift our focus to backend features that KAIZEN offers to support the development of backend services and APIs for your application. To explore these capabilities, we will use another application called “BETraining” which is designed to guide you through backend-centric functionalities.

Prerequisites (in folder [Software Installation](#))

For Windows:

1. **Node.js Version 18 (Preferably 18.20.4)**
 - o Download and install [node-v18.20.4-x64.msi](#)
2. **Nginx**
 - o Download [nginx.zip](#), unzip and place it in your C:\ drive (C:/nginx).
3. **Java**
 - o Download and install [OpenJDK17U-jdk_x86-32_windows_hotspot_17.0.12_7.msi](#)
4. **Maven**
 - o Download [apache-maven-3.9.9-bin.zip](#) Maven Zip File and Extract
 - o Install via this guide from Step 2 onwards - <https://phoenixnap.com/kb/install-maven-windows>

For MacOS:

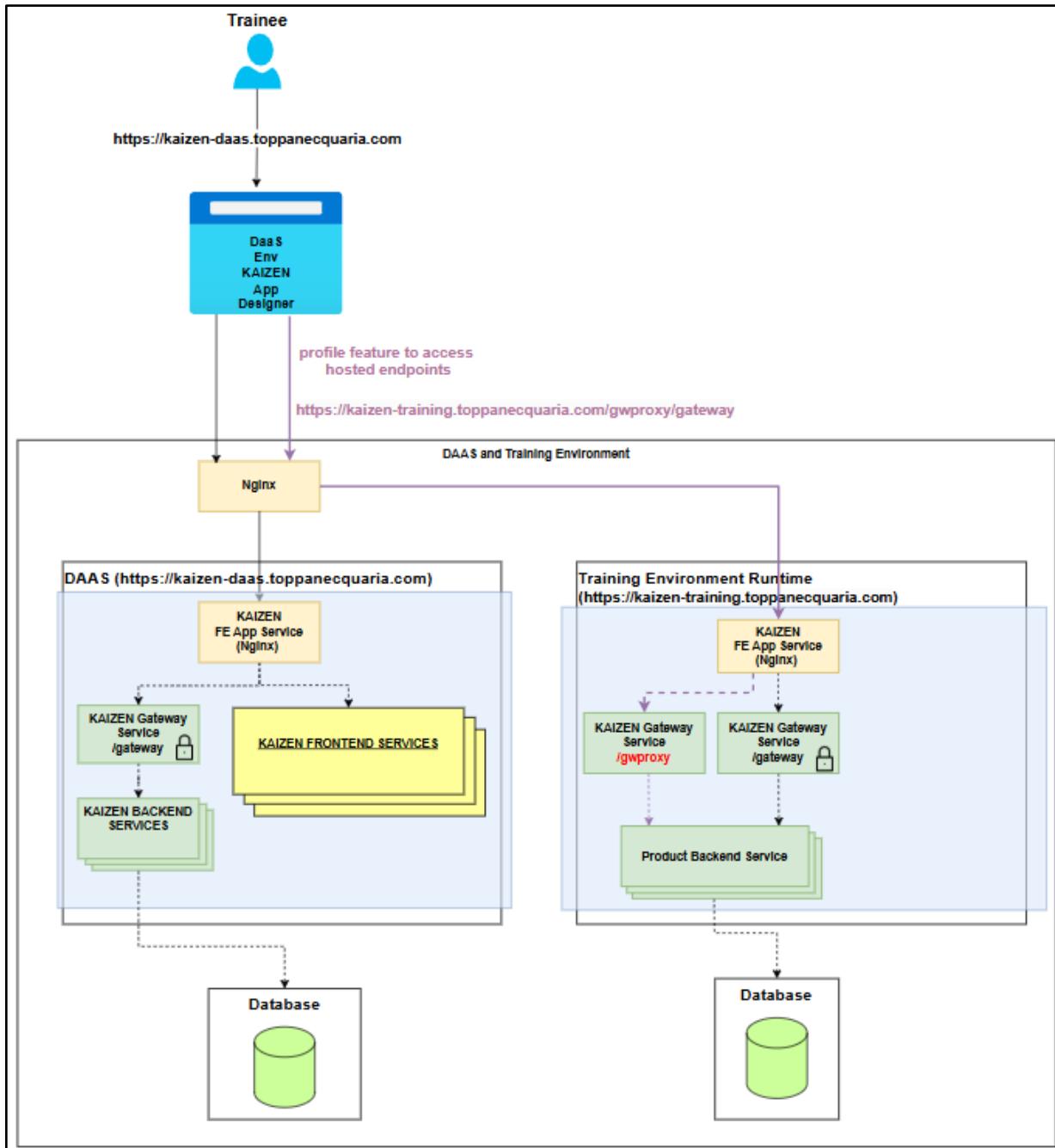
5. **Node.js Version 18 (Preferably 18.20.4)**
 - o Download this package [node-v18.20.4.pkg](#)
Or <https://nodejs.org/dist/v18.20.4/node-v18.20.4.pkg>
6. **Nginx**
 - o brew install nginx
7. **Java**
 - o Download and install
[OpenJDK17U-jdk_x64_mac_hotspot_17.0.12_7.pkg](#)
8. **Maven**
 - o Download Maven: [apache-maven-3.9.9-bin.tar.gz](#)
Or download from URL <https://dlcdn.apache.org/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz>
 - o Open the Terminal where your file being downloaded and Run this command on terminal

```
$ tar -xvf apache-maven-3.9.9-bin.tar.gz
$ sudo mv apache-maven-3.9.9 /opt/
$ export M2_HOME=/opt/apache-maven-3.9.9
$ export PATH=$M2_HOME/bin:$PATH
```

```
$ mvn -version (make sure it is v3.9.9 and its using JDK 17)
```

Architecture

A simple network architecture of the training environment is provided below.



Full Diagram:

[KAIZEN Deployment Architecture_v8.0\(CloudDaaS\)-KAIZEN Training.drawio.png](#)

Creation of Profile

Add to create the “Training Environment Profile” in your App Designer. This will connect your application to the training environment which we have pre-created.

This is the runtime environment set up for this BETraining application, which we are connecting your application to via the App Designer profile feature directly.

We will be going through more on the profile feature in [Tutorial 23](#).

The screenshot shows the App Designer interface with the 'Profiles' screen open. A specific profile named 'Training Environment' is selected. An 'Edit Profile' dialog is overlaid on the main screen, containing fields for Name, Url, User Domain, Account Id, and Password. The 'Name' field is set to 'Training Environment'. The 'Url' field contains the value 'https://kaizen-training.toppanecquaria.com/gwproxy'. The 'User Domain' field is set to 'training_product'. The 'Account Id' field contains 'amandalam' with a red link 'Change to your username'. The 'Password' field is obscured by dots. At the bottom of the dialog are 'Test Connection', 'Cancel', and 'Save' buttons.

Name	Training Environment
Url	https://kaizen-training.toppanecquaria.com/gwproxy
User Domain	training_product
Account	<username>
Password	Password\$1234

Ensure it is checked and **refresh the page**



Tutorial 14: Git Setup & Creation of New Branch

This tutorial covers the following Learning Objectives:

- Understand how to set up Gitlab in KAIZEN for version control.
- Creation of new branch

In this tutorial, we'll learn how to connect a project to a GitLab repository for efficient version control. This Git integration feature allows users to seamlessly connect their projects to a Git repository, enabling direct version control operations from within the application in KAIZEN. We will also learn how to initialize new repositories directly from KAIZEN. This integration enhances the CI/CD pipeline by automating the process of committing and pushing changes, thus improving efficiency and collaboration. Additionally, we will also learn how to create a new branch in order to work on feature enhancements or bug fixes in a project setting.

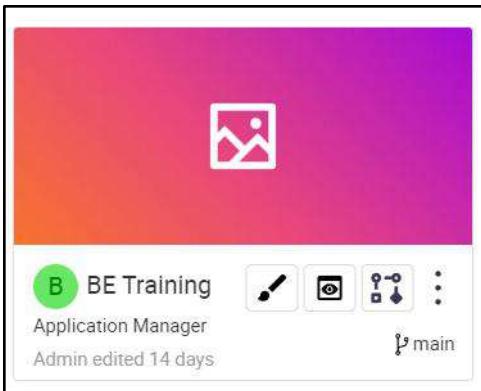
Practical 14.1: Connect with Gitlab

In this [GitLab](#) environment, a group “KAIZEN-Training” has already been created for the purpose of this training. In addition, a sub-group has also been created for each trainee prefixed with your username ‘kaizen-training-<username>’ (E.g. kaizen-training-amandalam).

The screenshot shows a GitLab group page for 'KAIZEN-Training'. At the top, there's a header with the group name. Below it, there's a large 'K' icon followed by the group name 'KAIZEN-Training' and a lock icon. A navigation bar below the header includes 'Subgroups and projects', 'Shared projects', and 'Inactive'. There's a search bar with a placeholder 'Search (3 character minimum)'. At the bottom, there's a list of subgroups, with one subgroup named 'kaizen-training-amandalam' highlighted in orange, showing its status as 'Owner'.

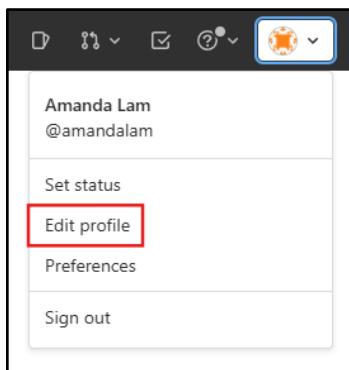
Similarly, in your project's context, you may create subgroups for different applications required in your project (E.g. FE internet portal, BE intranet portal), and each subgroup will then be initialized with your application repositories. After which, you may then connect to Git in KAIZEN by configuring the respective group.

Navigate back to your KAIZEN studio console, you will notice that the current branch for your application is on ‘main’. We will go through a later tutorial on Git Branch Management to allow you to push your changes onto your repositories.

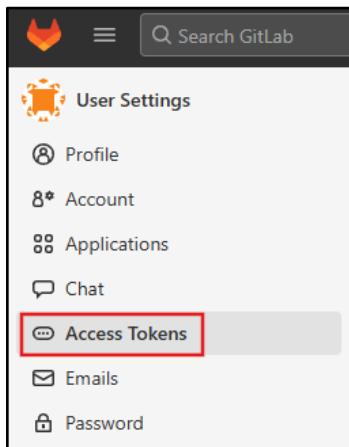


14.1.1 Generate Gitlab Personal Access Token

- Login to your gitlab account as follows and reset your password
 - **URL:** <http://alb-kaizen-daasez-001-673358367.ap-southeast-1.elb.amazonaws.com/>
 - **Username:** <username>
 - **Password:** ahdoh6shaechai0seidu1ahnailoogaeF8Ainie
- Navigate to your gitlab account profile, click **Edit Profile**



- Click on **Access Tokens**



- **Create** your personal access token
 - **Token name:** access_token
 - **Expiration date:** leave as default
 - **scope:** api, read_api, read_repository, write_repository

Add a personal access token

Token name
access_token

For example, the application using the token or the purpose of the token.

Expiration date
2025-03-31

Select scopes

Scopes set the permission levels granted to the token. [Learn more](#).

api
Grants complete read/write access to the API, including all groups and projects, the container registry, the dependency proxy, and the package registry.

read_api
Grants read access to the API, including all groups and projects, the container registry, and the package registry.

read_user
Grants read-only access to your profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

create_runner
Grants create access to the runners.

manage_runner
Grants access to manage the runners.

k8s_proxy
Grants permission to perform Kubernetes API calls using the agent for Kubernetes.

read_repository
Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.

write_repository
Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).

ai_features
Grants access to GitLab Duo related API endpoints.

- **Copy** the generated token to store it for later use

Your new personal access token

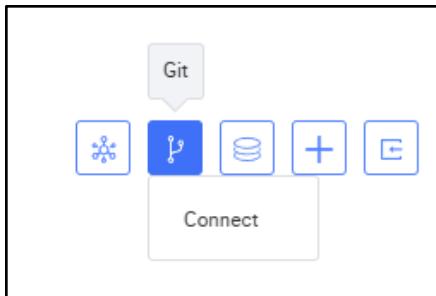
glpat-6-YqyHsf11...

Copy personal access token

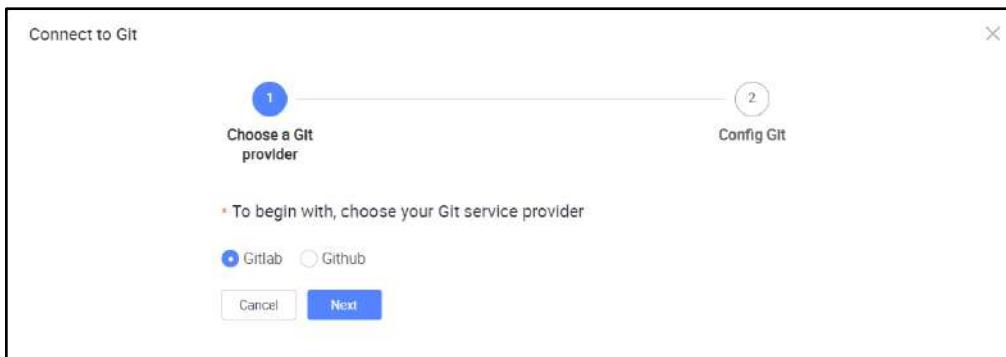
Make sure you save it - you won't be able to access it again.

14.1.2 Connect to Git

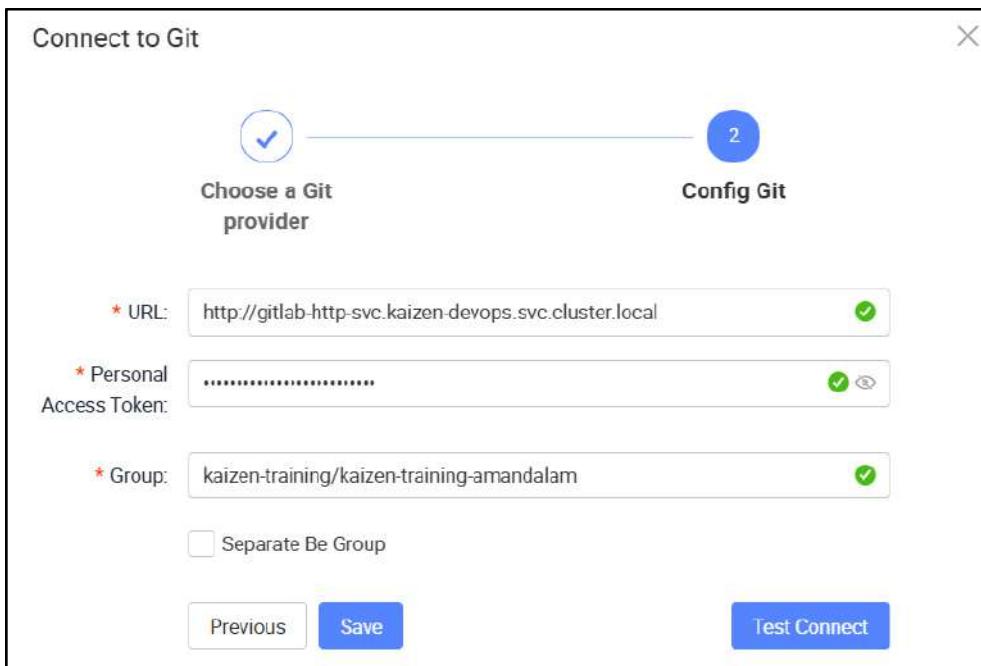
- On the KAIZEN studio console, click on the Git button of your BE Training project and click **Connect**.



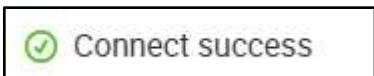
- Select **Gitlab** as the service provider and click **Next**



- Enter the following details to connect to Gitlab
 - **URL:** <http://gitlab-http-svc.kaizen-devops.svc.cluster.local>
 - **Personal Access Token:** <token-string>
 - **Group:** kaizen-training/kaizen-training-<username>
- Click on **Test Connect** to test your connection



- The group to specify is the url path of your group and subsequent subgroups where you will be initializing your project.
- Note that if the “Separate BE Group” checkbox is selected, then “Group” will specify the group for your Frontend Repository and “BE Group” will be for your Backend Repository. If it is not selected, then both FE and BE will be connected to the same Git group. This is to help aid in development such that you will not need to always re-connect to another group path in the event you want to push your code in your BE repository.
- Once successful, click on **Save** to connect.



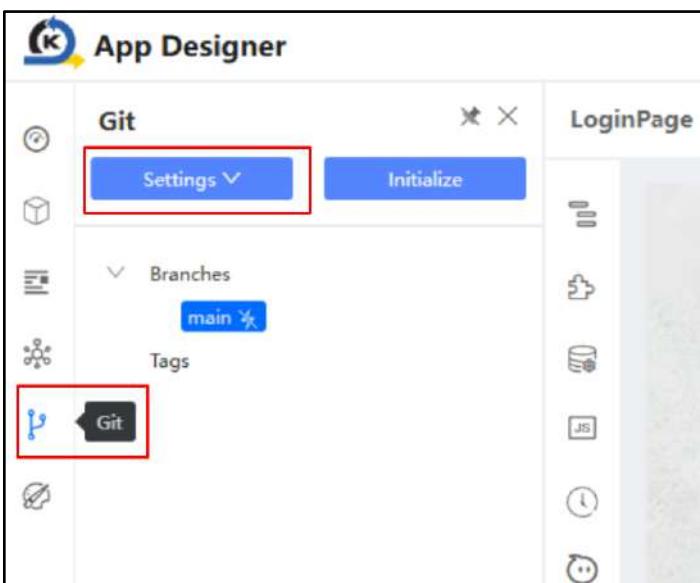
Practical 14.2: Initialize Git Repository

14.2.1 Frontend - Initialize Git Repository

- Click on the **Design icon** of your application. It will navigate to the **App designer** of your application



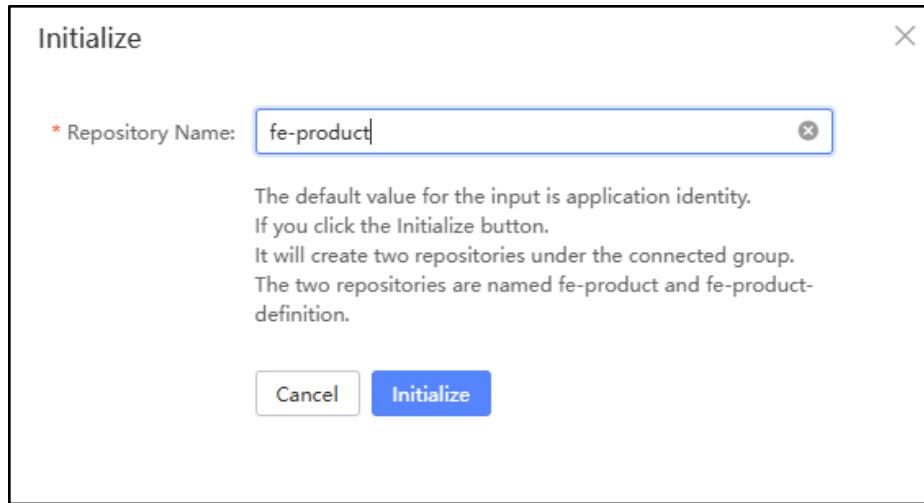
- In the left panel area, click on the **Git icon**. Notice that you are already connected to Git. You may click on **Settings** to verify.



- Click on the button **Initialize**



- In the popup, input the following and click on **Initialize** to create the gitlab repository:
 - **Repository name:** fe-product



✅ application initialization to git was successful,these new repositories are created successfully.

- Verify the following initialized frontend repositories in your gitlab project.

Repository	Created	Stars
fe-product	just now	0
fe-product-definition	just now	0

- Currently, these initialized repositories are empty. In the later tutorial we will show you how you can generate and push your code into these defined repositories.

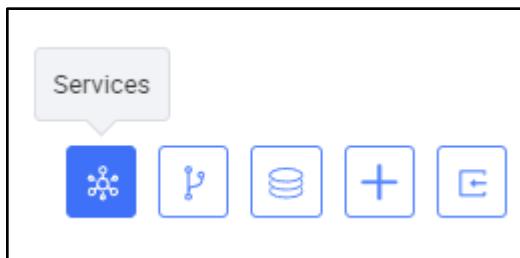
14.2.2 Backend - Initialize Git Repository

Backend - Create Service (Prerequisite)

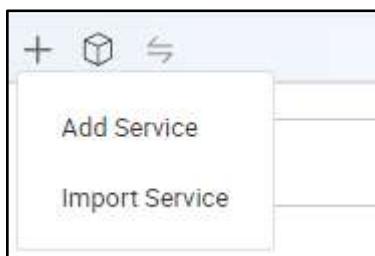
You may initialize your BE repository through the Service Designer after the creation of your microservice(s).

Let us first create an empty Product Service first so that the BE repository can be initialized. Note that a microservice for your application needs to be created first before you can initialize your BE Git repository. Not to worry as we will cover more details on the Service Designer tutorial below.

- Click **Services** to launch the user interface.



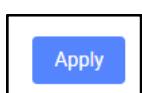
- Select **Add Service** in the Service API Designer.



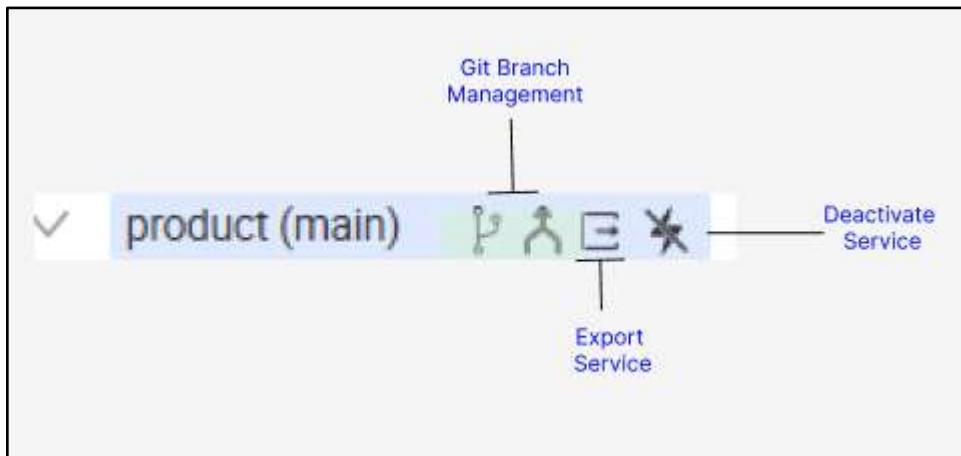
- In the configuration panel, name your microservice to represent its function (e.g., "Product" for Product microservice). Click **Apply**.

The configuration panel for adding a service. It has three input fields:

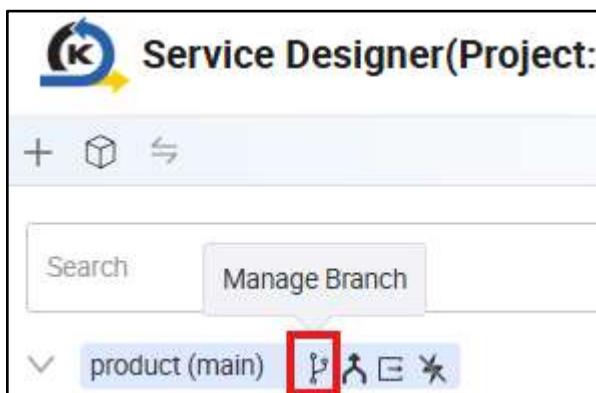
- * Name: Product
- Service Package Path: com.ecquaria.lowcode
- Controller Package: (empty)



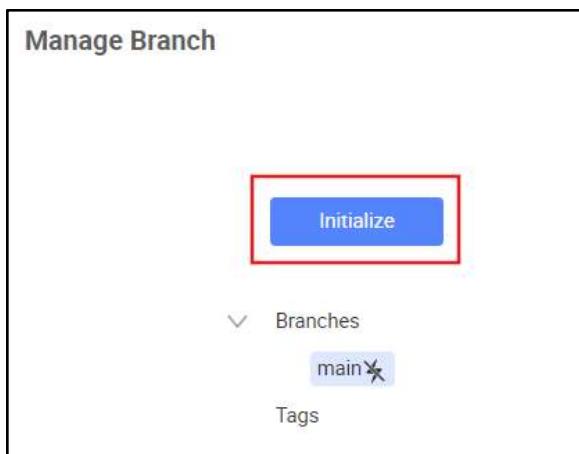
- In the Service Designer, the Product Service has a couple of features which we will explore further in later tutorials.



- In your Service Designer, click on the **Git icon (Manage Branch)** on your Service.



- Click **Initialize** to set up your BE repository.





service initialization to git was successful, these new repositories are created successfully.

- You should be able to see this newly created repository reflected on Git as well.

Subgroups and projects	Shared projects	Inactive
<input type="button" value="Search (3 character minimum)"/> Q	Name ▾	
be-product	★ 0	just now
fe-product	★ 0	2 minutes ago
fe-product-definition	★ 0	2 minutes ago

- KAIZEN will create these 3 repositories under the defined group.

Project Name	Description
fe-product	FE generated react code
fe-product-definition	FE screen pages which store the JSON Schema (JS definitions etc). This schema is rendered during runtime and on preview to load the page as well.
be-product	BE generated code

Note:

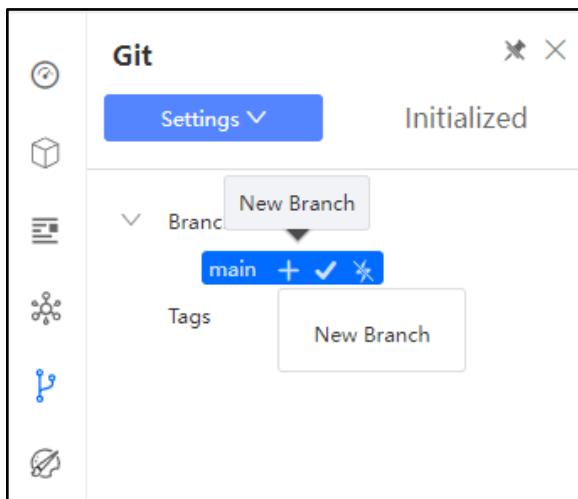
If you connect to git again with the repositories already present and initialized, clicking on the initialized button will not create new repositories. The success message will flag out accordingly that this repository with this name has already been initialized when connecting.

Practical 14.3: Create and Switch to new branch

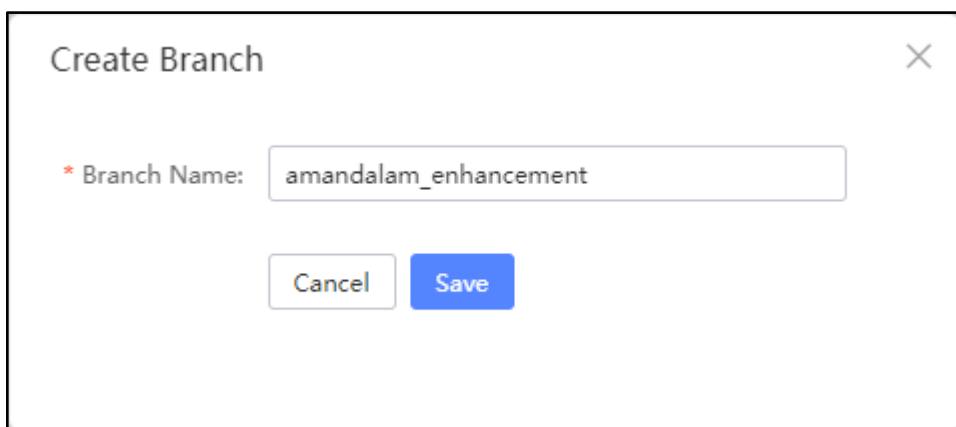
In KAIZEN, frontend Git branch management is essential for maintaining an organized and efficient workflow. By creating separate branches for UI changes, such as theme updates, new features or bug fixes, developers can isolate modifications, avoid conflicts, and ensure smooth collaboration across teams. This approach supports parallel development, minimizes disruption to the main codebase, and allows for better testing and integration of frontend updates, ultimately enhancing project quality and stability.

14.3.1 Frontend Git Branch Management

- In your App Designer, click on the **Git icon** menu item on the left panel. Then, click on the **+** icon to create a new branch



- In the pop-up window, enter the following details
 - **Branch Name:** <username>_enhancement (e.g. amandalam_enhancement).



This will create a new branch based on the current branch that you are on, in this case, 'main'. You can then work on your feature enhancement and make the relevant changes to this newly created branch.

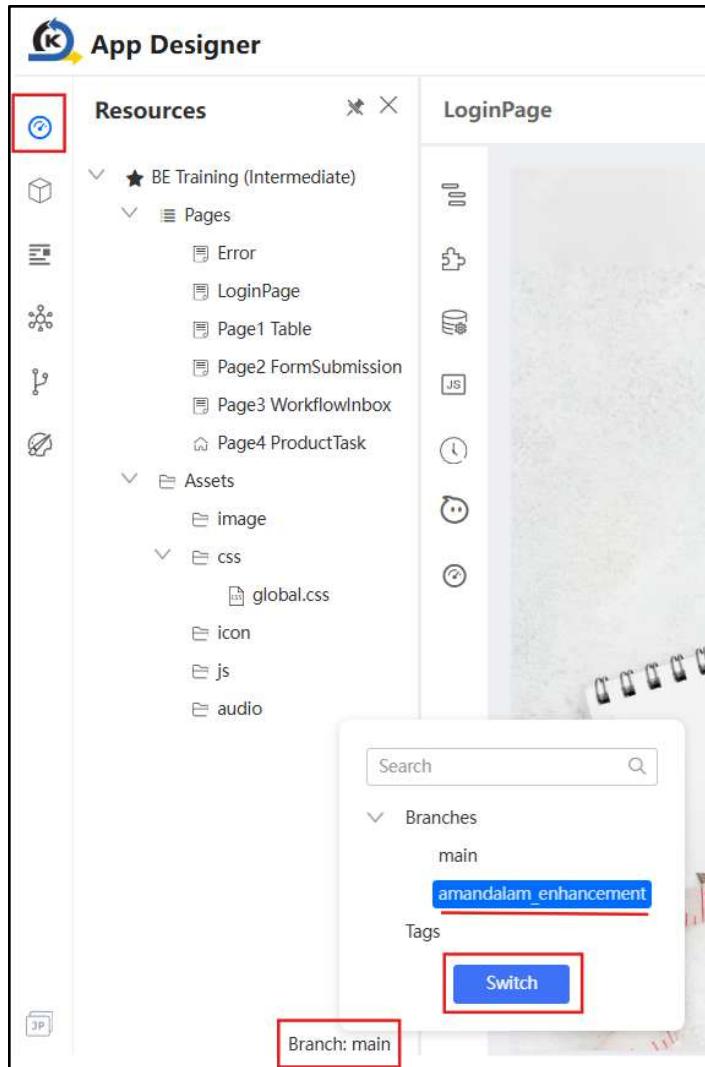
 Application Branch "amandalam_enhancement" created successfully

You should be able to see this newly created branch reflected on Git as well.

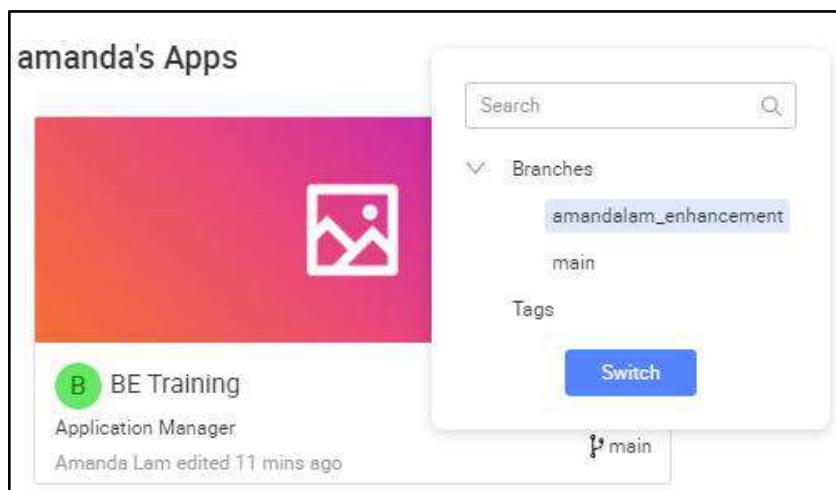
- Navigate back to your application's resources by clicking the **Resources** icon



- In the app designer. You may switch to the new branch by clicking on your application's current branch as shown below.



Alternatively, you may also switch to the new branch by clicking the Git branch on your application from the studio console.



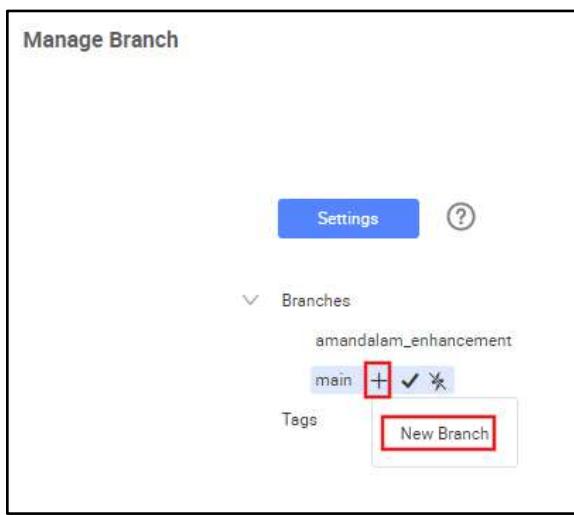
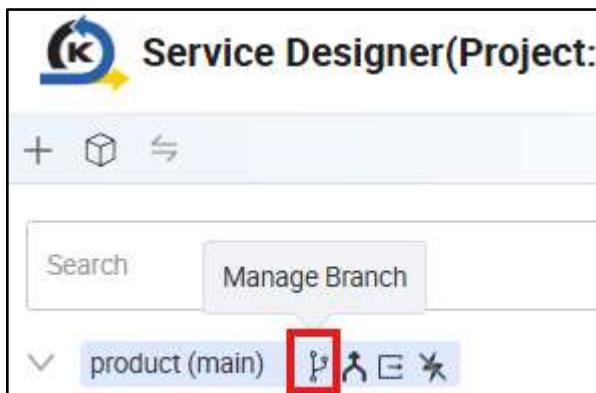
Note that switching of branches on KAIZEN is a personal setting and is tied to individual users. Hence, if another project member switches branches, it will not be reflected on your application.

- Ensure that your 'Training Environment' profile is checked in the new branch as well.



14.3.2 Backend Git Branch Management (Optional)

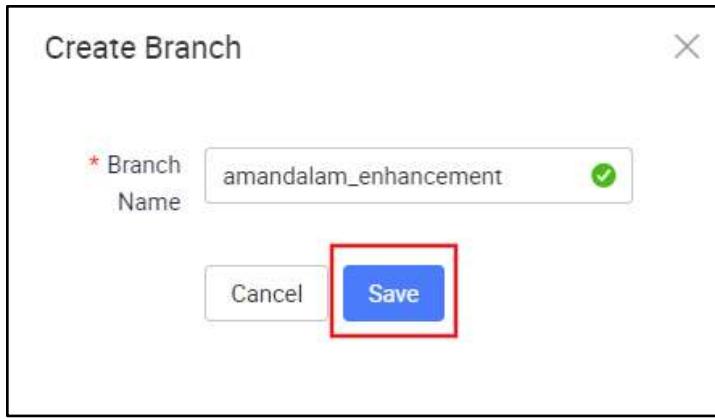
- Similarly for BE, we will proceed to create a new branch in the Service Designer. This will allow us to work on our new feature enhancements before merging into the main branch.



Create Branch

* Branch Name amandalam_enhancement ✓

Cancel Save



- You should be able to see this newly created branch reflected on Git as well.
- Switch to the new branch from the service designer. Select the branch to switch to in the dropdown and click 'Apply'.

Service Designer

+ ⌂ 

Name	Branch
ProductService	amandalam_enhancement

Apply

 Switch Successfully

Product (amandalam_enhancement) + ⌂ ⌂ ⌂ ⌂



Tutorial 15: Master Code Integration

This tutorial covers the following Learning Objectives:

- Understand how to integrate Master Code functionality within a low-code application.
- Learn to bind specific master codes, such as location codes, dynamically to different entities.
- Enhance application flexibility by allowing the dynamic assignment of master codes.

In this tutorial, you will learn how to integrate Master Code functionality into your application. Master Codes allow users to assign predefined codes to different entities, such as location or category codes. This integration improves the flexibility of your application, enabling dynamic assignment of codes that enhance data management and workflow precision.

Practical 15.1: Integrating Master Code

- (Note) Two Master Codes “Category” and “Location” were already created

The screenshot shows the 'Master Code' section of the System Configuration interface. At the top, there are input fields for 'User Domain' (Training Product), 'Master Code Name' (Please enter), 'Status' (Please select), and 'Description' (BE). Below these are 'Search' and 'Clear' buttons. A navigation bar at the bottom includes 'Create', 'Import', 'Export', and 'Export All' buttons. The main area displays a table of master codes:

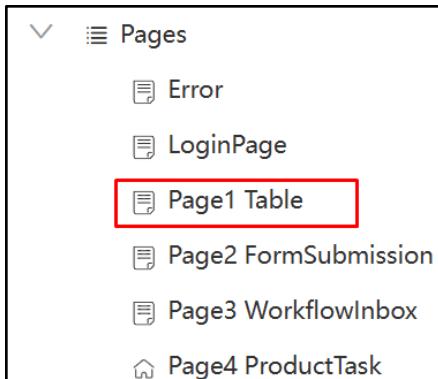
No.	User Domain	Master Code Name	Status	Description	Actions
1	Training Product	Category	Active	Category for BE...	View Edit More
2	Training Product	Location	Active	Location for BE...	View Edit More

Two side-by-side 'Edit Master Code' dialog boxes are shown. Both dialogs have 'Master Code Name' set to 'Category' and 'Status' set to 'Active'. The left dialog has a description of 'Category for BE Training' and the right one has 'Location for BE Training'. Both dialogs include a 'Create Item' button and a table of items:

No.	Item Code	Short Description	Priority	Status	Operation
1	Clothing	Clothing	0	Active	Edit Delete
2	Electronics	Electronics	0	Active	Edit Delete
3	Home	Home	0	Active	Edit Delete

At the bottom of each dialog are 'Cancel' and 'Save' buttons.

- In App Designer, click on **Page1 Table** page

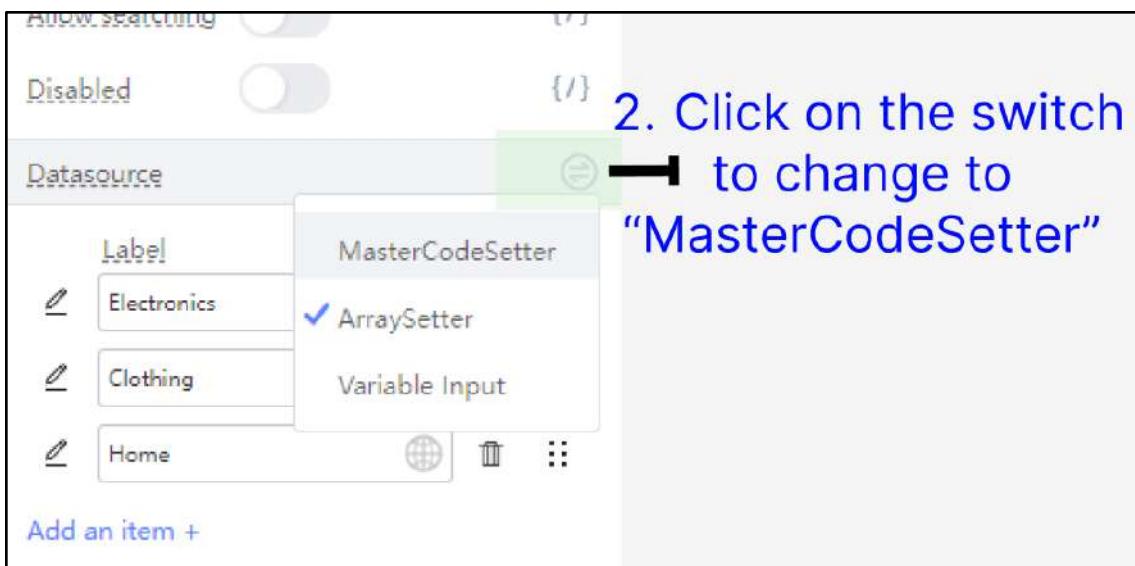


- **Click on Category search field**

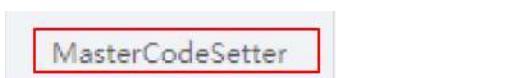
Click on the Select component

- (Note) Notice that the current values for this dropdown are static and hardcoded under **Datasource** section

- Click on the switch icon to switch to MasterCodeSetter in order to dynamically populate this field

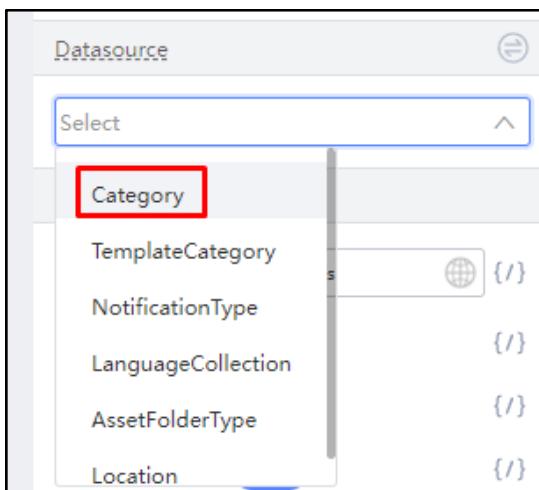


- Select **MasterCodeSetter**



Note: MasterCodeSetter supports multiple components such as Select, Cascader, CascaderSelect, PhoneInput.

- In the dropdown, select **Category**



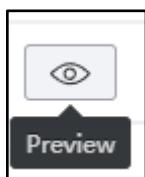
- Similarly, select **Location** from search field. Under **Datasource**, click on **switch icon** and change to **MasterCodeSetter**

The screenshot shows a web-based application for managing products. At the top, there is a search bar with fields for 'Product Name' (placeholder: Please enter), 'Category' (placeholder: Please select), and 'Location' (placeholder: Please select). Below the search bar is a table listing eight products with columns: Id, Product Name, Category, Brand, Color, Price, Location, Status, and Action (with a 'Download PDF' button). To the right of the table is a configuration panel for a 'Select' component. The configuration panel includes tabs for 'Preview', 'Styles', 'Events', and 'Advanced'. Under the 'Preview' tab, there are settings for 'Placeholder' (Please select), 'Default value' (empty), 'Status' (Medium), 'Size' (Large), 'Mode' (Single), and 'Allow creation' (disabled). Under the 'Datasource' section, there is a dropdown menu with items: 'Label' (MasterCodeSetter), 'Singapore' (checked), 'Malaysia' (unchecked), and 'Variable input'. Other configuration options include 'Empty value placeholder' (No option), 'Placeholder' (empty), 'Autosize' (disabled), and 'Show arrow' (disabled). A callout box labeled '1. Click on the Select component' points to the 'Select' component in the preview area. Another callout box labeled '2. Click on the switch to change to "MasterCodeSetter"' points to the 'MasterCodeSetter' checkbox in the datasource dropdown.

- In the dropdown, select **Location**

This screenshot shows the 'Datasource' configuration for the 'Select' component. It lists several options under the 'Select' dropdown: 'Category', 'TemplateCategory', 'NotificationType', 'LanguageCollection', 'AssetFolderType', and 'Location'. The 'Location' option is highlighted with a red rectangle. To the right of each option is a placeholder field ending in '{/}'.

- Preview the page



- Perform the Searching function. Note that it still works as intended using Master Code values fetched dynamically, instead of hardcoded values

The screenshot shows a browser developer tools Network tab. A request to `https://kaizen-training.toppaenequaia.com/gw/proxy/gateway/common/api/v1/masterCodes/masterCodeStructure?masterCodeName=Location` is listed. The request method is GET, and the status code is 200 OK. The response headers include `Access-Control-Allow-Credentials: true`, `Access-Control-Allow-Origin: https://kaizen-datas.toppaenequaia.com`, and `Cache-Control: no-cache, no-store, max-age=0, must-revalidate`.

Practical 15.2: Multi language support (Optional)

Master code feature has a multi-language support. This enhancement will allow users to view and select master codes, such as location codes, in their preferred language, improving the accessibility and usability of your application for a global audience.

- Master code translation to Chinese have already been created

The screenshot shows the 'Edit Master Code Item' dialog box. The item code is 'Clothing'. The short description is 'Clothing'. The translation table has one row:

No.	Language	Value	Operation
1	Chinese	衣服	Edit Delete

Priority: 0
 Status: Active
 Icon: Click to select icon
 Start Time: Select date
 End Time: Select date
 Buttons: Cancel, Save

- Click on “Edit Application”. Note that multi-language support “Chinese” is enabled.

- Click on “Edit Application”

The screenshot shows the 'Edit Application' dialog box. In the 'Languages' section, both 'English' and 'Chinese' are listed with a checkmark. The 'Chinese' option is highlighted with a red box. Other settings include:

- Name:** BE Training
- App ID:** backend
- Type:** Standalone App
- Default Language:** English, Chinese (both checked)
- Status:** Active
- Assets:** assets.json (6.52KB)
- Logos:** Upload File
- Default Page:** DefaultPage

- Preview the page and select “Chinese” as the language

The screenshot shows a preview of the application. A dropdown menu labeled 'BE Training' has 'Chinese' selected, indicated by a red box. The dropdown also contains 'English' with a red box around it. The page title is 'BE Training'.

- In the dropdown select, noted that master code is using chinese translation

The screenshot shows the application interface with two dropdown menus:

- Category:** The dropdown shows 'Please select' and '衣服' (Clothing).
- Location:** The dropdown shows 'Please select' and '马来西亚' (Malaysia).

BE Training Table Form Submission Workflow Inbox

BE Training zh - Chinese

Product Name: Please enter

Category: 电子

Location: 衣服
✓ 电子
家庭用品

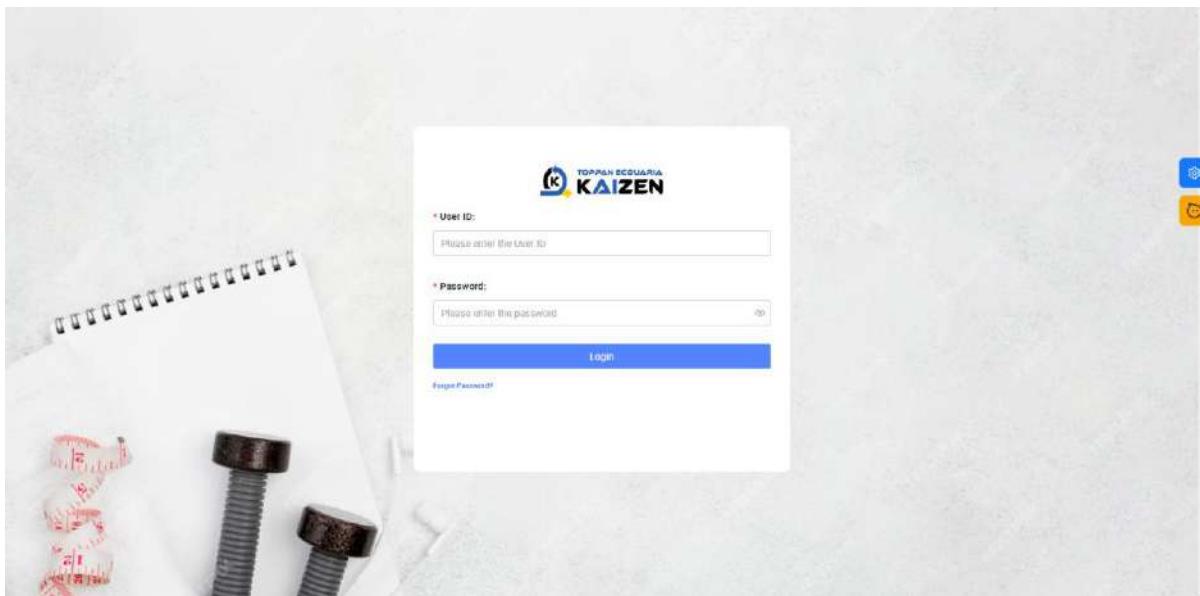
Id	Product Name	Category	Brand	Color	Price	Location
1	Product A	Electronics	Brand X	Black	499.99	Singapore
3	Product C	Electronics	Brand Z	Silver	899.99	Singapore
5	Product E	Electronics	Brand Y	Black	99.99	Malaysia
8	Product H	Electronics	Brand X	White	399.99	Singapore

Tutorial 16: Custom Login Page Integration

This tutorial covers the following Learning Objectives:

- Learn to integrate a custom login page for your application.
- Understand how to connect your custom login page with IAM (Identity and Access Management) service.
- Secure your application with enhanced authentication mechanisms.

In this tutorial, you will learn how to customize and integrate a login page into your application, backed by the IAM service in KAIZEN. This allows you to provide a secure and user-friendly login experience while ensuring that user authentication aligns with your organization's identity management policies.



Practical 16.1: Login Page Integration

All necessary backend APIs for authentication have already been configured, allowing you to focus on designing and implementing the front-end aspects of the login page.

- **Update** the path with your **username** in the Source Code
 - betraining_amandalam

LoginPage

Source Code Panel

* index.js index.css Save

```
1 class LowcodeComponent extends Component {
2
3     state = {
4         publicKey:'',
5         loginForm:{},
6         userId: '',
7         password: ''
8     }
9
10    componentDidMount() {
11    }
12
13    onChangeUserId(...val) {
14        this.setState({userId: val[0]})
15    }
16
17    onChangePwd(...val) {
18        this.setState({password: val[0] })
19    }
20
21    onClick(){
22        const {userId, password} = this.state;
23        this.dataSourceMap['Publckey'].load().then(res => {
24            this.setState({
25                publicKey: res.data,
26            }, () => {
27                console.log(this.state.publicKey)
28                console.log(userId, password);
29                const encryptedPassword = this.utils.RSAUtil.encrypt(password, JSON.parse(res.data));
30                console.log(encryptedPassword);
31                this.setState({
32                    loginForm: {
33                        userAccountId: userId,
34                        userDomainCode: 'designer',
35                        vault: encryptedPassword
36                    }
37                }, () => {
38                    this.dataSourceMap['LoginAPI'].load({ ...this.state.loginForm })
39                    if (res.success == true) {
40                        localStorage.setItem('token', res.data);
41                        //update this to your username (ex. betraining_amandalap)
42                        this.utils.navigateTo('betraining_traineeName/#/table', '');
43                    } else {
44                        console.log(res.data)
45                    }
46                })
47            })
48        });
49    });
50});
```

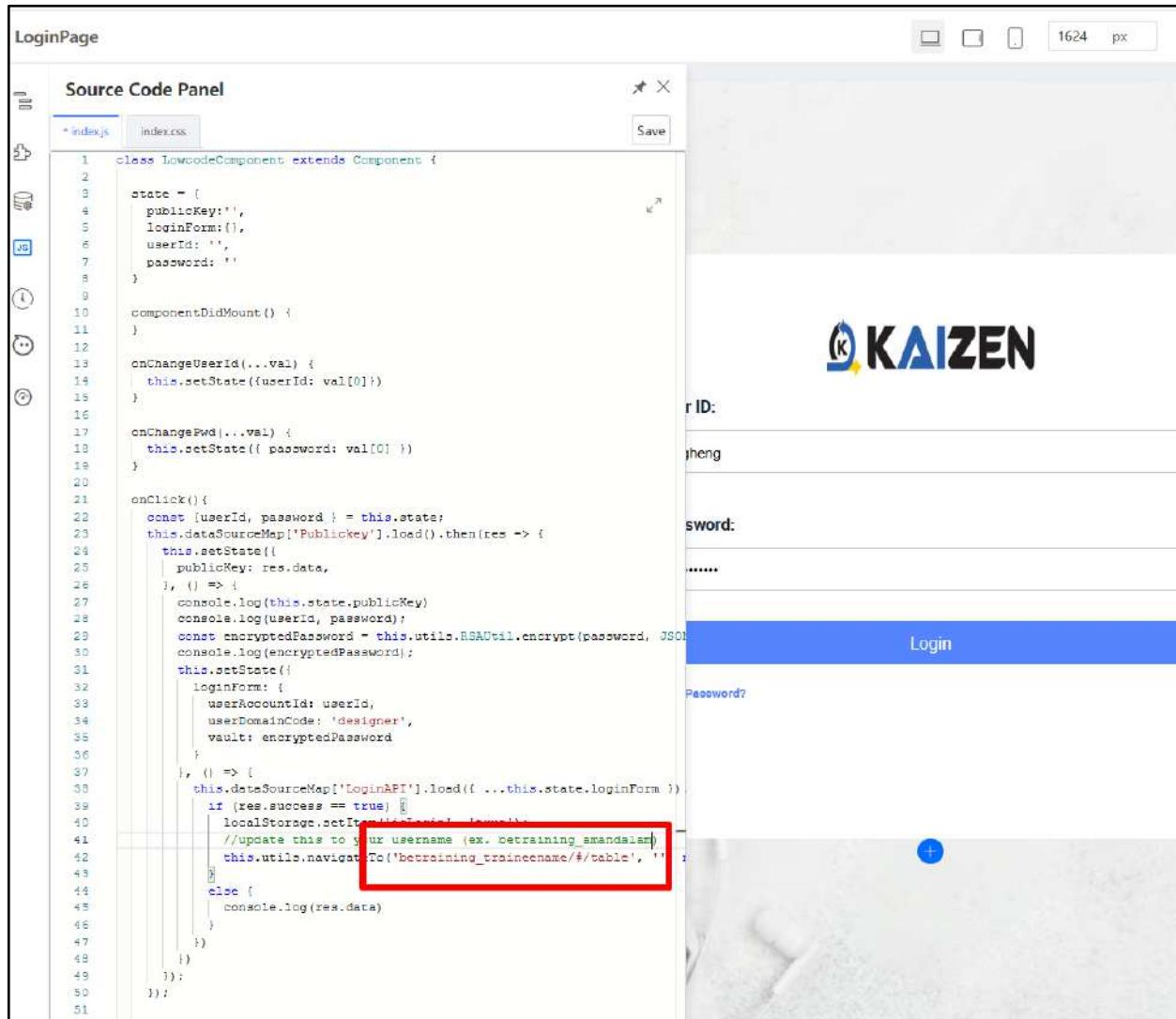
KAIZEN

UserID: pheng

sword: *****

Login

Password?



Tutorial 17: Database Designer

This tutorial covers the following Learning Objectives:

- Understand how to create, modify, and manage database schemas using the Database Designer within the low-code platform.
- Learn to automate the generation of Database Access Objects (DAOs) by configuring them through the Database Designer.
- Effectively manage your database scripts within a centralized platform for improved organization and accessibility.

In this tutorial, you'll explore the Database Designer feature within KAIZEN, which allows you to design and manage your application's database with ease. Instead of manually writing SQL queries, the designer provides a user-friendly interface to create tables, define relationships, and structure your data model. This feature helps streamline database management and ensures that your data structure is aligned with your application's requirements.

Practical 17.1: Setup Database and tables

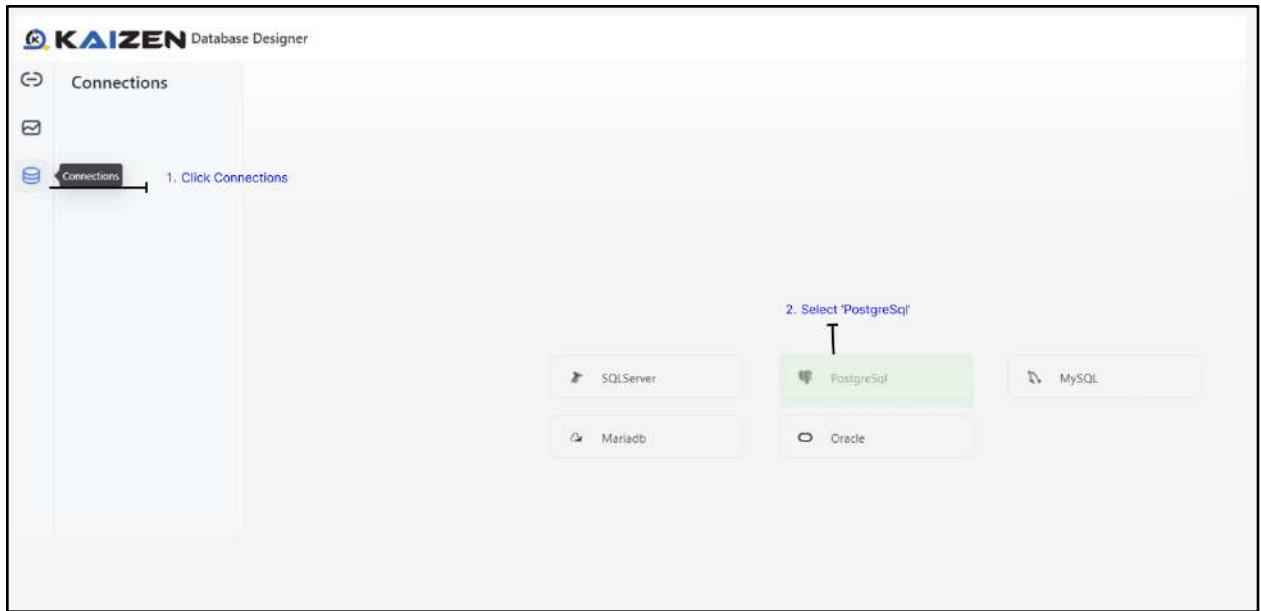
Each trainee is provided with a central PostgreSQL database for the practical. Ensure you have the correct database credentials (hostname, username, password, and database name) before proceeding.

Connect to the PostgreSQL Server

- In the studio console, go to your project and click **Database Designer** to launch the user interface



- Click **Connections** and select **PostgreSQL**



- Enter the following configuration details:
 - **Host:** 172.20.0.141
 - **Port:** Use the default PostgreSQL port (usually 5432)
 - **User:** <username>
 - **Password:** tr@ining2025
 - **Database:** trg_single

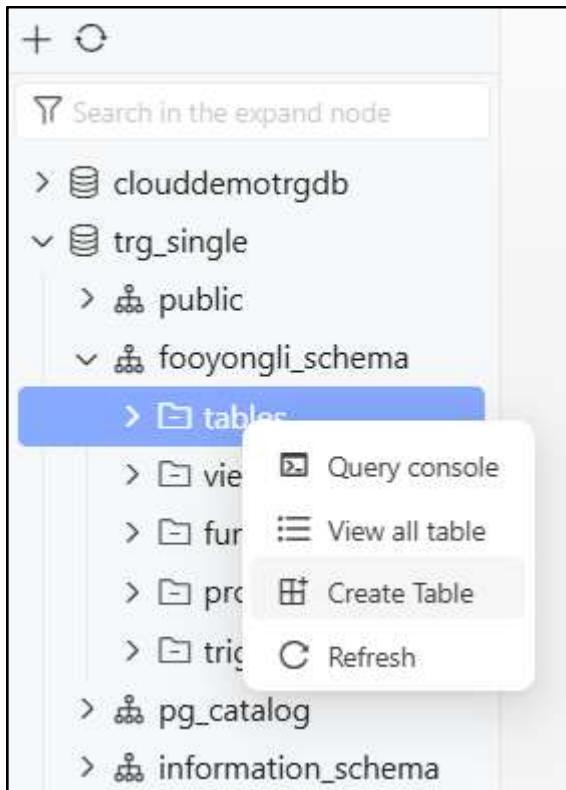
The screenshot shows the 'PostgreSQL' connection configuration dialog. The 'Name' field is set to '@172.20.0.141'. The 'Host' field is '172.20.0.141' and the 'Port' field is '5432'. Under 'Authentication', 'User&Password' is selected. The 'User' field is 'trainee1' and the 'Password' field contains redacted text. The 'Database' field is 'trg_single'. The 'URL' field displays the JDBC URL: 'jdbc:postgresql://172.20.0.141:5432/trg_single'. At the bottom, there are 'Test', 'Cancel', and 'Modify' buttons. A tooltip 'Advanced Configuration' is visible near the bottom left.

- **Test** the connection. Once successful, click on **Save**



Creating a New Table

- To create a table, right-click on the Tables section under the created schema and select **Create Table**.



- In the right-panel section under **Basic**, enter the table name **product**

Create Table X +

Basic Column Index Foreign Key

* Table name:

product

Comment:

This screenshot shows the 'Create Table' dialog box. The 'Basic' tab is selected. A required field 'Table name:' contains the value 'product'. There is also a 'Comment:' field which is currently empty.

- Under Column, click on **Add column** to add new columns

Create Table X +

Basic Column Index Foreign Key

Name	Type	Size	Nullable	Key	Comment
No data					

[Add new column](#) [+ Add column](#)

This screenshot shows the 'Column' tab of the 'Create Table' dialog box. It displays a table structure with columns for Name, Type, Size, Nullable, Key, and Comment. Below the table, there is a message 'No data'. At the bottom, there are two buttons: 'Add new column' and '+ Add column'.

- Define the following columns for the product table

Create Table X +

Basic Column Index Foreign Key

Name	Type	Size	Nullable	Key	Comment
id	SERIAL		<input type="checkbox"/>		¹
name	VARCHAR	64	<input checked="" type="checkbox"/>		
price	VARCHAR	64	<input checked="" type="checkbox"/>		
category	VARCHAR	64	<input checked="" type="checkbox"/>		
status	VARCHAR	64	<input checked="" type="checkbox"/>		
brand	VARCHAR	64	<input checked="" type="checkbox"/>		
color	VARCHAR	64	<input checked="" type="checkbox"/>		
location	VARCHAR	64	<input checked="" type="checkbox"/>		
pdf_data	BYTEA		<input checked="" type="checkbox"/>		

Column Name	Data Type	Size	Additional Constraint
id	SERIAL	-	Key: Primary Key
name	VARCHAR	64	Nullable: true
price	VARCHAR	64	Nullable: true
category	VARCHAR	64	Nullable: true
status	VARCHAR	64	Nullable: true
brand	VARCHAR	64	Nullable: true
color	VARCHAR	64	Nullable: true
location	VARCHAR	64	Nullable: true
pdf_data	BYTEA	-	Nullable: true

- Click Save

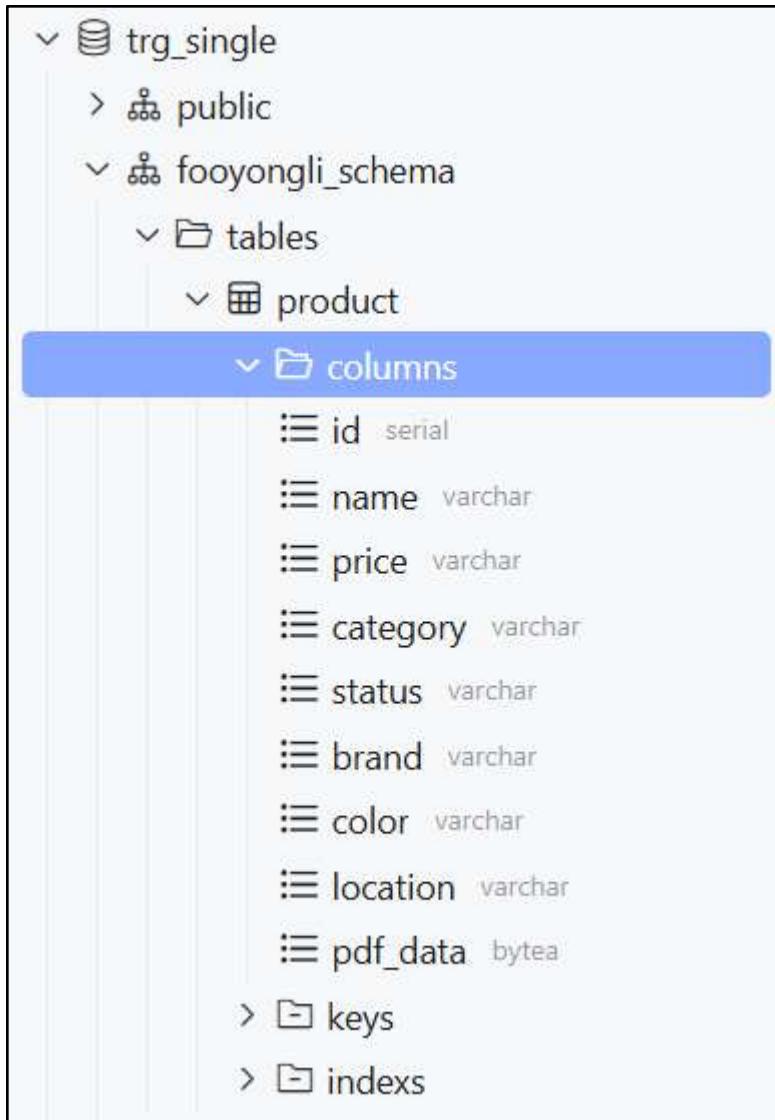


- Click **Run** in SQL preview



```
□CREATE TABLE "product" (
    "id" SERIAL NOT NULL ,
    "name" VARCHAR(64) NULL ,
    "price" VARCHAR(64) NULL ,
    "category" VARCHAR(64) NULL ,
    "status" VARCHAR(64) NULL ,
    "brand" VARCHAR(64) NULL ,
    "color" VARCHAR(64) NULL ,
    "location" VARCHAR(64) NULL ,
    "pdf_data" BYTEA NULL ,
    PRIMARY KEY ("id")
);
```

- □After creating the table, expand the Tables section under the schema to view the newly created table.



Adding Rows to Table

In a development environment, it is often necessary to quickly add data to a PostgreSQL table for testing purposes. There are two main ways to add data to a PostgreSQL table: manually through the user interface or by running SQL scripts. Below are the steps for both approaches.

- Right-click on the table and select **Open Table**.

The screenshot shows a database navigation tree. At the top level is 'trg_single'. Below it is 'public'. Under 'public' is 'fooyongli_schema'. Inside 'fooyongli_schema' is a folder named 'tables'. A blue bar highlights the 'product' table. A context menu is open over the table, listing 'Open Table', 'Query console', and 'Pin'. The table itself has columns: id, name, price, category, status, brand, color, location, and pdf_data.

- In the right panel, select **Add Row (+)**

The screenshot shows the 'product' table interface. At the top right, there is a button labeled 'Add Row' with a red box around it. Below the table header, there are buttons for navigating between rows and a search bar. The table has columns: id, name, price, category, status, brand, color, location, and pdf_data. The 'id' column is currently sorted by ascending order.

- Enter the following details to the new row for the first record
 - id:** 1
 - name:** Product A
 - price:** 499.99
 - category:** Electronics
 - status:** Approved
 - brand:** Brand X
 - color:** Black
 - location:** Singapore
 - pdf_data:** <null>

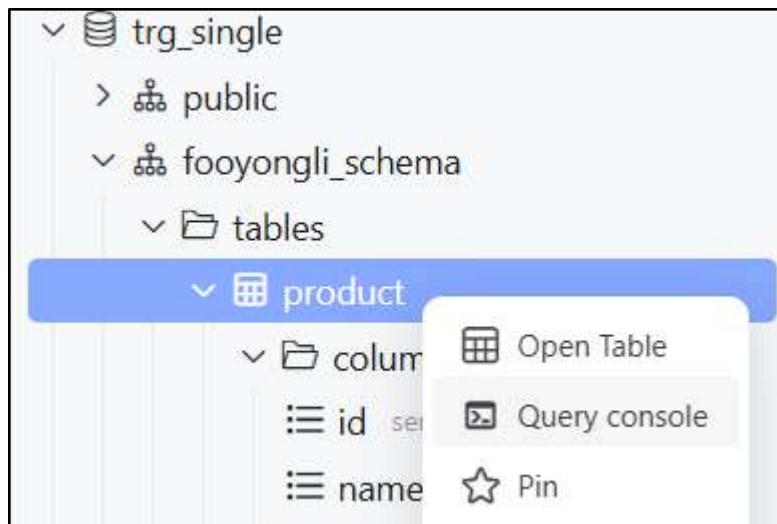
The screenshot shows the 'product' table after a new row has been added. The table now has a total of 201 rows. The newly added row is the first one, with id 1, name 'Product A', price 499.99, category 'Electronics', status 'Approved', brand 'Brand X', color 'Black', location 'Singapore', and pdf_data '<null>'. The 'ORDER BY' dropdown is set to 'id'.

- Click **Submit** to save the new row

The screenshot shows a database interface with a table named 'product'. The table has columns: id, name, price, category, status, brand, color, location, pdf_data, created_at, and updated_at. There is one row with values: 1, Product A, 499.99, Electronics, Approved, Brand X, Black, Singapore, <null>, 2024-10-12 00:00:00, and 2024-10-12 00:00:00. A 'Successful Execution' message is displayed at the top right.

	id	name	price	category	status	brand	color	location	pdf_data	created_at	updated_at
1	1	Product A	499.99	Electronics	Approved	Brand X	Black	Singapore	<null>	2024-10-12 00:00:00	2024-10-12 00:00:00

- Alternatively, we can select the **Query console** to run SQL scripts. Right click on table and select **Query console**



- Paste the SQL script below in the console and click **Run**

The screenshot shows a MySQL Workbench session titled 'trg_single (@172.20.0.141)'. The editor contains the following SQL code:

```

1 INSERT INTO product (id, name, category, brand, color, status, price, location) VALUES
2 (2, 'Product B', 'Clothing', 'Brand Y', 'Blue', 'Approved', '29.99', 'Singapore'),
3 (3, 'Product C', 'Electronics', 'Brand Z', 'Silver', 'Approved', '899.99', 'Singapore'),
4 (4, 'Product D', 'Clothing', 'Brand X', 'Blue', 'Approved', '49.99', 'Singapore'),
5 (5, 'Product E', 'Electronics', 'Brand Y', 'Black', 'Approved', '99.99', 'Singapore'),
6 (6, 'Product F', 'Home', 'Brand Z', 'Brown', 'Approved', '599.99', 'Singapore'),
7 (7, 'Product G', 'Clothing', 'Brand Y', 'Red', 'Approved', '39.99', 'Singapore'),
8 (8, 'Product H', 'Electronics', 'Brand X', 'White', 'Approved', '399.99', 'Singapore'),
9 (9, 'Product I', 'Clothing', 'Brand Z', 'Gray', 'Approved', '59.99', 'Singapore'),
10 (10, 'Product J', 'Electronics', 'Brand Y', 'Silver', 'Approved', '149.99', 'Singapore'),
11 (11, 'Product 404', 'Electronics', 'Brand Y', 'Silver', 'Approved', '149.99', 'Singapore'),
12 (12, 'Product 400', 'Electronics', 'Brand Y', 'Silver', 'Approved', '149.99', 'Singapore'),
13 (13, 'Product 500', 'Electronics', 'Brand Y', 'Silver', 'Approved', '149.99', 'Singapore');

```

The 'Run' button is highlighted with a red box. Below the editor, a 'Result 1' tab is open, and at the bottom right, it says 'Affected rows: 12'.

```

□INSERT INTO product (id, name, category, brand, color, status, price, location)
VALUES
(2, 'Product B', 'Clothing', 'Brand Y', 'Blue', 'Approved', '29.99', 'Singapore'),
(3, 'Product C', 'Electronics', 'Brand Z', 'Silver', 'Approved', '899.99',
'Singapore'),
(4, 'Product D', 'Clothing', 'Brand X', 'Blue', 'Approved', '49.99', 'Singapore'),
(5, 'Product E', 'Electronics', 'Brand Y', 'Black', 'Approved', '99.99', 'Singapore'),
(6, 'Product F', 'Home', 'Brand Z', 'Brown', 'Approved', '599.99', 'Singapore'),
(7, 'Product G', 'Clothing', 'Brand Y', 'Red', 'Approved', '39.99', 'Singapore'),
(8, 'Product H', 'Electronics', 'Brand X', 'White', 'Approved', '399.99',
'Singapore'),
(9, 'Product I', 'Clothing', 'Brand Z', 'Gray', 'Approved', '59.99', 'Singapore'),
(10, 'Product J', 'Electronics', 'Brand Y', 'Silver', 'Approved', '149.99',
'Singapore'),
(11, 'Product 404', 'Electronics', 'Brand Y', 'Silver', 'Approved', '149.99',
'Singapore'),
(12, 'Product 400', 'Electronics', 'Brand Y', 'Silver', 'Approved', '149.99',
'Singapore'),
(13, 'Product 500', 'Electronics', 'Brand Y', 'Silver', 'Approved', '149.99',
'Singapore');

```



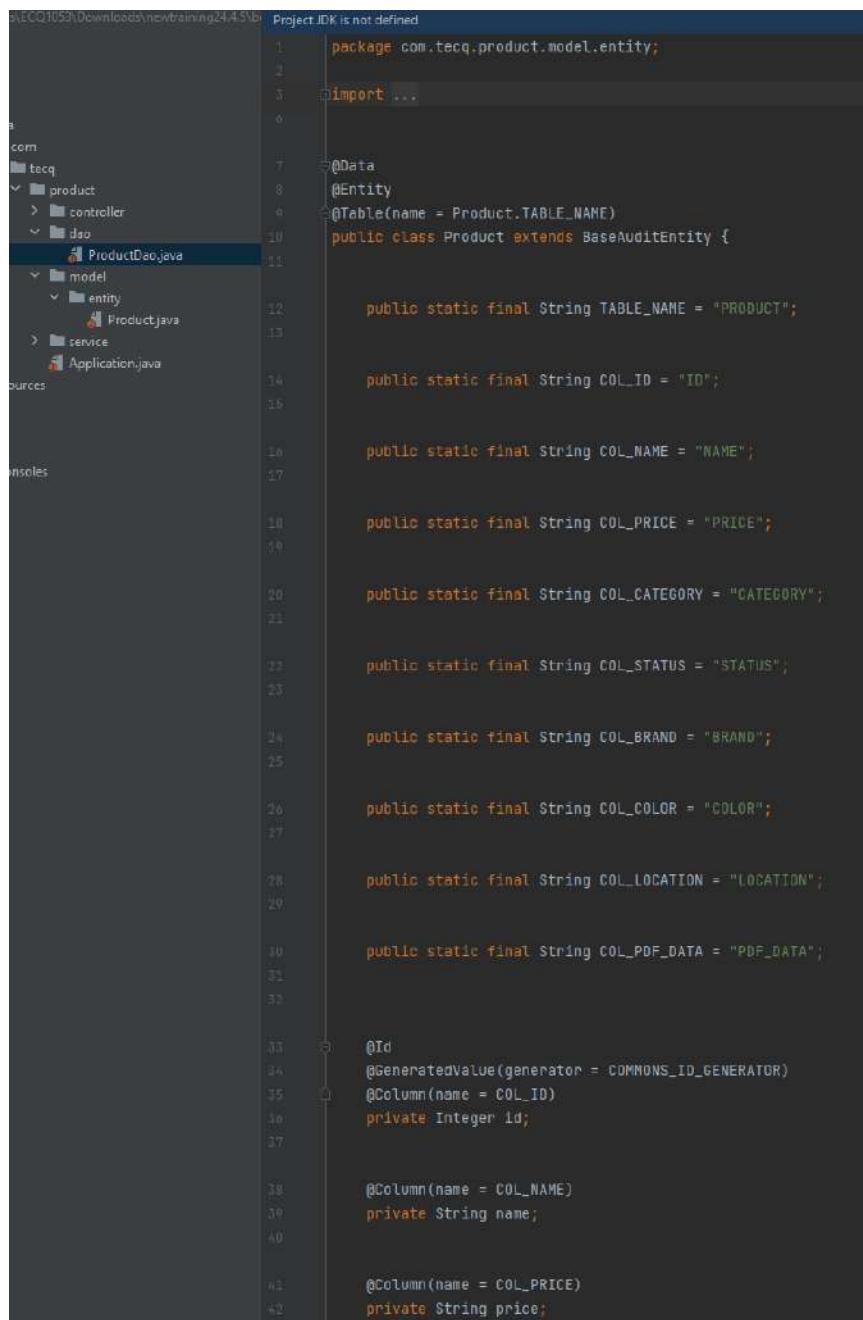
- Back to the Table tab, we click **Refresh** and see the updated list of rows in the table

	id	name	price	category	status	brand	color	location	pdf_data
1	1	Product A	499.99	Electronics	Approved	Brand X	Black	Singapore	<null>
2	2	Product B	29.99	Clothing	Approved	Brand Y	Blue	Singapore	<null>
3	3	Product C	899.99	Electronics	Approved	Brand Z	Silver	Singapore	<null>
4	4	Product D	49.99	Clothing	Approved	Brand X	Blue	Singapore	<null>
5	5	Product E	99.99	Electronics	Approved	Brand Y	Black	Singapore	<null>
6	6	Product F	599.99	Home	Approved	Brand Z	Brown	Singapore	<null>
7	7	Product G	39.99	Clothing	Approved	Brand Y	Red	Singapore	<null>
8	8	Product H	399.99	Electronics	Approved	Brand X	White	Singapore	<null>
9	9	Product I	59.99	Clothing	Approved	Brand Z	Gray	Singapore	<null>
10	10	Product J	149.99	Electronics	Approved	Brand Y	Silver	Singapore	<null>
11	11	Product 404	149.99	Electronics	Approved	Brand Y	Silver	Singapore	<null>
12	12	Product 400	149.99	Electronics	Approved	Brand Y	Silver	Singapore	<null>
13	13	Product 500	149.99	Electronics	Approved	Brand Y	Silver	Singapore	<null>

Benefits of Database designer

The Database Designer tool supports automatic generation of Data Access Objects (DAO) during code generation. This feature allows developers to quickly generate Java classes that map to database tables and handle CRUD operations, reducing the need for manual coding. We will explore this feature in greater detail in later tutorials.

In addition, the Database Designer also supports the automated creation of **Entity Relationship (ER) Diagram** and **Data Dictionary** for your application's Databases.



The screenshot shows a Java code editor with the file `Product.java` open. The code defines a class `Product` that extends `BaseAuditEntity`. The class includes static final variables for table name, column names, and price. It also contains annotations for `@Id`, `@GeneratedValue`, `@Column`, and `private` fields for `id`, `name`, and `price`. The code editor interface includes a project tree on the left and line numbers on the right.

```
1 package com.tecq.product.model.entity;
2
3 import ...
4
5 @Data
6 @Entity
7 @Table(name = Product.TABLE_NAME)
8 public class Product extends BaseAuditEntity {
9
10     public static final String TABLE_NAME = "PRODUCT";
11
12     public static final String COL_ID = "ID";
13
14     public static final String COL_NAME = "NAME";
15
16     public static final String COL_PRICE = "PRICE";
17
18     public static final String COL_CATEGORY = "CATEGORY";
19
20     public static final String COL_STATUS = "STATUS";
21
22     public static final String COL_BRAND = "BRAND";
23
24     public static final String COL_COLOR = "COLOR";
25
26     public static final String COL_LOCATION = "LOCATION";
27
28     public static final String COL_PDF_DATA = "PDF_DATA";
29
30
31     @Id
32     @GeneratedValue(generator = COMMONS_ID_GENERATOR)
33     @Column(name = COL_ID)
34     private Integer id;
35
36
37     @Column(name = COL_NAME)
38     private String name;
39
40
41     @Column(name = COL_PRICE)
42     private String price;
```

Tutorial 18: Service / API Designer

This tutorial covers the following Learning Objectives:

- Understand how to design and configure the service layer for APIs using the Service Designer feature within the low-code platform.
- Learn to define and manage essential API parameters and configurations using the API Designer.
- Gain insights into best practices for creating scalable and maintainable services and APIs to support application functionality.

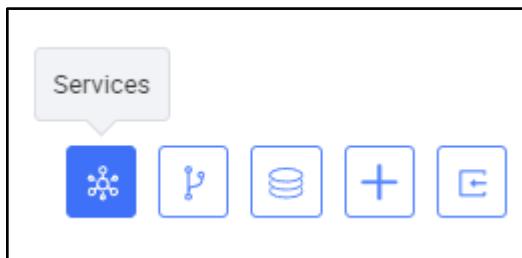
In this tutorial, you'll explore the Service and API Designer features of KAIZEN, which allow you to build and configure the service layer for your applications seamlessly. The Service Designer enables you to design the architecture of your service layer, ensuring that it meets the requirements of your application. Meanwhile, the API Designer allows you to specify the necessary parameters and configurations for your APIs, ensuring effective communication between different components of your application. By mastering these tools, you will create robust and efficient service and API layers that enhance your application's performance and scalability.

Practical 18.1: Creating Service and Controller

The Service API Designer in development enables users to visually create and manage APIs with minimal coding effort. It simplifies the process of defining API endpoints, adding controllers, configuring datasource, and handling data types through an intuitive drag-and-drop interface. With built-in import/export features and code generation capabilities, the tool allows developers to efficiently design and integrate APIs into their applications. This feature streamlines API management, making it accessible to both technical and non-technical users, while ensuring that services are well-organized and maintainable.

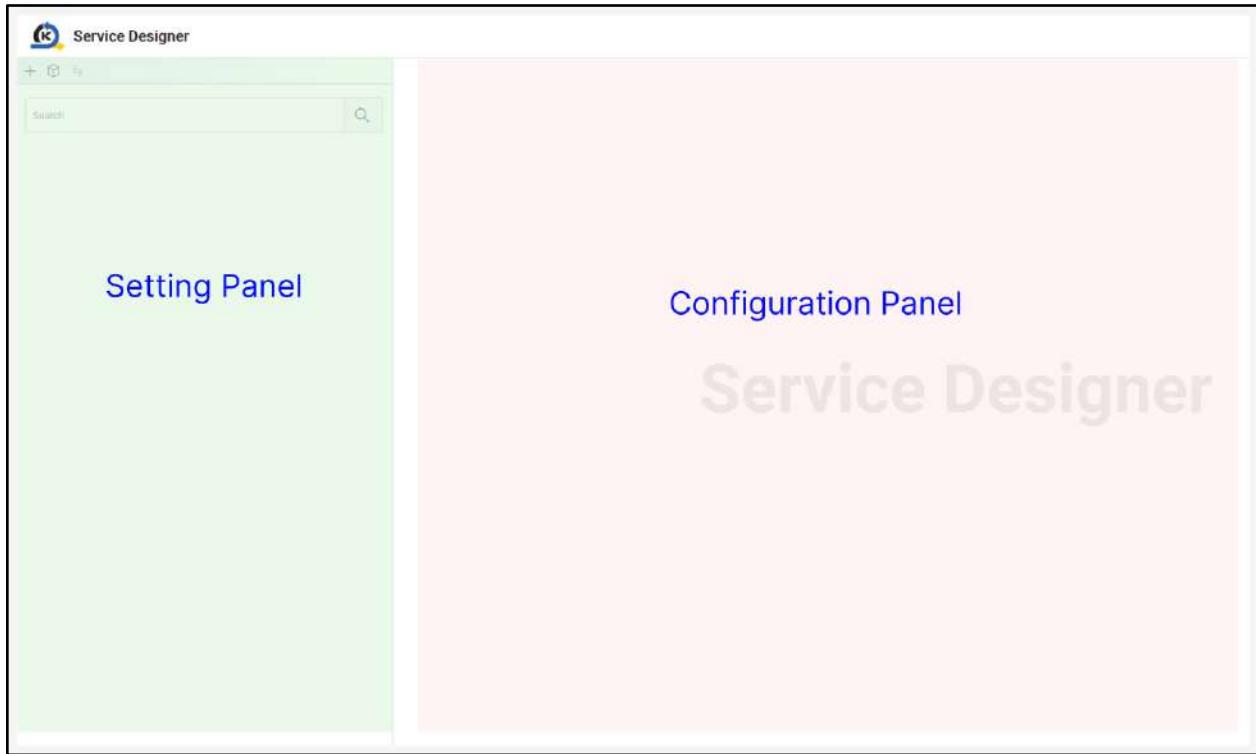
Navigating the Service / API Designer

- Click **Services** to launch the user interface



- Familiarize yourself with the interface layout

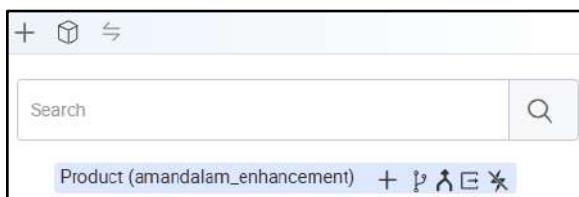
- **Settings Panel (Left):** Contains options like add/import service, code generation, switching branch.
- **Configuration Panel (Right):** For configuring controllers, datasources, and more.



Adding a Service

The Service API Designer allows you to create and manage service (microservices), which serve as the backbone of a modular application architecture. Microservices enable scalable and independent components for handling different application functions. Microservices offer flexibility in development by breaking down a monolithic application into smaller, independent services that can be developed, deployed, and scaled individually. This ensures that each microservice focuses on a specific business capability, allowing for faster updates, better fault isolation, and easier scaling.

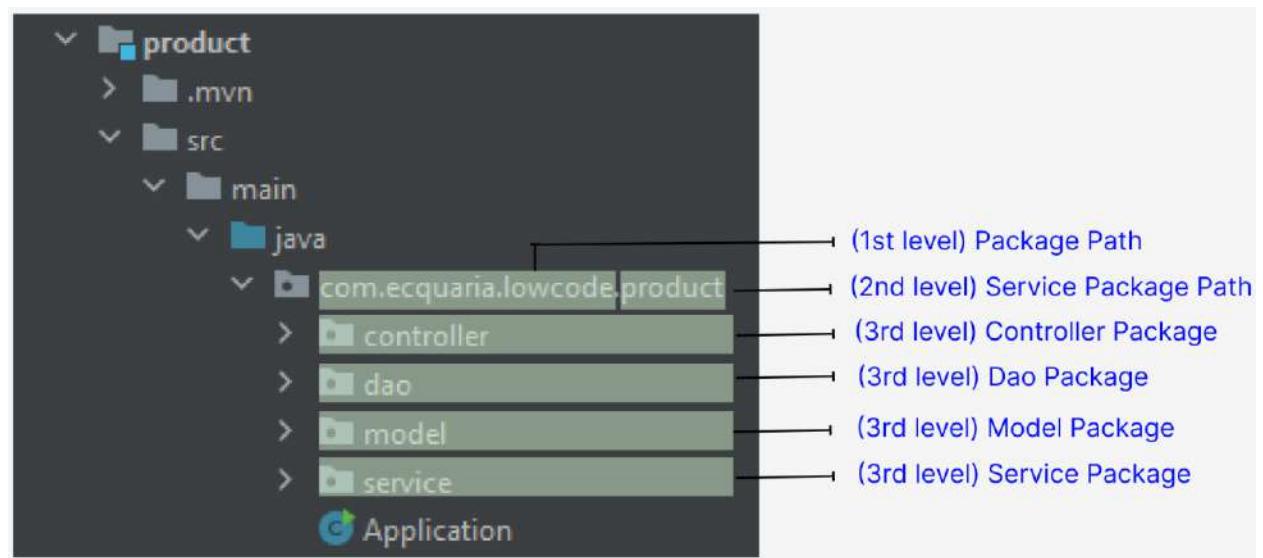
- In Tutorial 11, you would have already added an empty ‘Product’ microservice in order to initialize your Git Repository. Click on it in the Service API Designer to edit the service.



- In the configuration panel, name your microservice to represent its function (e.g., "Product" for Product microservice).
 - **Name:** Product
This is the name of the service being created. In this case, "Product" represents a microservice responsible for managing product-related functionalities in your application.
 - **(1st level) Package Path:** com.ecquaria.lowcode
This specifies the root path at project level. Only project admin has the privilege to edit this value.
 - **(2nd level) Service Package Path:** product
This specifies the base package path where all service-related components will reside. It organizes the project structure under product, ensuring that the service logic is well-organized.
 - **(3rd level) Controller Package:** controller (Default value: controller)
The package where the controllers will be placed. Controllers handle incoming HTTP requests and route them to the appropriate service methods, providing endpoints for the API.
 - **(3rd level) Service Package:** service (Default value: service)
This defines the location for service classes. The service layer contains business logic that performs operations related to the domain (e.g., product management), sitting between the controller and DAO layers.
 - **(3rd level) Model Package:** model (Default value: model)
This package holds the data models or entities representing your domain objects. In this case, the Product model would define the structure of product data in the system.
 - **(3rd level) Dao Package:** dao (Default value: dao)
DAO (Data Access Object) package contains classes that interact with the database, executing queries and managing persistence for your models (e.g., fetching or saving product data).
 - **(3rd level) Dao Package:** vo (Default value: vo)
VO (Value Object) package represents a simple object that holds data and defines an attribute or a concept in the domain.
 - **Priority:** 0
The priority defines the importance or order of this service relative to others. A lower number (like 0) could indicate higher priority, ensuring this service is loaded or accessed first when needed.

* Name:	product
Service Package Path:	com.ecquaria.lowcode product
Controller Package:	product controller <input type="button" value="X"/>
Service Package:	product service
Model Package:	product model
Dao Package:	product dao
Vo Package:	product vo
KAIZEN Version:	Please Select <input type="button" value="▼"/>
<input checked="" type="checkbox"/> Generate service layer classes	

Example of generated code structure based on the value entered:



Note the **Package Path** value is specified when creating the project and the value can be edited. Only project admin can edit this value. For this training, the value of **com.ecquaria.lowcode** is being used.

The screenshot shows a software interface for managing projects. At the top, there are four buttons: 'Create', 'Import', 'Export', and 'Export All'. Below this is a table with columns: 'No.', 'Project Name', 'Package Path', and 'Description'. A single row is selected, showing '1', 'Training - amandalam', and 'com.ecquaria.lowcode' in the 'Package Path' column, which is highlighted with a green border. Below the table, there is a dropdown for 'Items per page' set to 10. To the right, an 'Edit Project' dialog is open. It contains fields for 'Project Name' (Training - amandalam), 'Package Path' (com.ecquaria.lowcode, also highlighted with a green border), 'Status' (Active, selected), and 'Description'. At the bottom of the dialog are 'Cancel' and 'Save' buttons.

Adding a Database

Configuring a database is a crucial step to enable data storage and retrieval within the application. Integrating a new database configuration allows you to specify connection details, select tables, and set up data sources that your app can interact with directly. By configuring the database, your application can seamlessly manage and access data as part of its automated workflows and user interactions.

- Click **Add Database** to add database configuration

The screenshot shows a database configuration interface. On the left, it says 'Databases:' followed by a list of tabs: 'Connection', 'Database', 'Schema', 'Package Path', and 'Operation'. The 'Connection' tab is active and highlighted with a blue background. Below the tabs, the text 'No Data' is displayed. At the bottom, there is a blue button labeled 'Add Database'.

- Enter the following configuration details and click **Save**:
 - **Connection:** @172.20.0.141
 - **Database:** trg_single
 - **Schema:** <username>_schema (e.g. amandalam_schema)
 - **Dao Package Path:** <BLANK> (Default)
 - **Tables:** product
 - Each of the tables in 'Assigned' will generate the DAO in the code.
 - Usually in a project setting, you might not want to generate the DAO for all the tables in your project database for this particular microservice. In this case, you will select the relevant 'product' database table created.

Add Database

* Connection: @172.20.0.141

* Database: trg_single

* Schema: amandalam_schema

Dao Package Path: dao

Tables	Pending assign	Assigned
<input type="text" value="Please Input"/>	<input type="text" value="Please Input"/>	<input checked="" type="checkbox"/> product
Not Found		
<input type="checkbox"/> 0 item		<input checked="" type="checkbox"/> 1/1 item

> <

Cancel Save

- Click **Apply** to create the service

* Name: product

Service Package Path: com.ecquaria.lowcode.product

Controller Package: product.controller

Service Package: product.service

Model Package: product.model

Dao Package: product.dao

Vo Package: product.vo

KAIZEN Version: Please Select

Generate service layer classes

Priority: 0

Databases:	Connection	Database	Schema	Package Path	Operation
	@172.20.0.141	trg_single	amandalam_schema		Edit Delete
	Add Database				Apply

Adding a Controller

Controllers in the Service API Designer act as an interface between the client (front end) and the service logic. They receive incoming requests, process them, and return appropriate responses. Controllers play a critical role in organizing API endpoints for different services. They manage the flow of data between the client and the service layer, ensuring that requests are routed correctly and data is processed efficiently. Properly structuring your controllers helps maintain clarity and scalability as your application grows.

- Click **Add Controller**



- Enter the following:

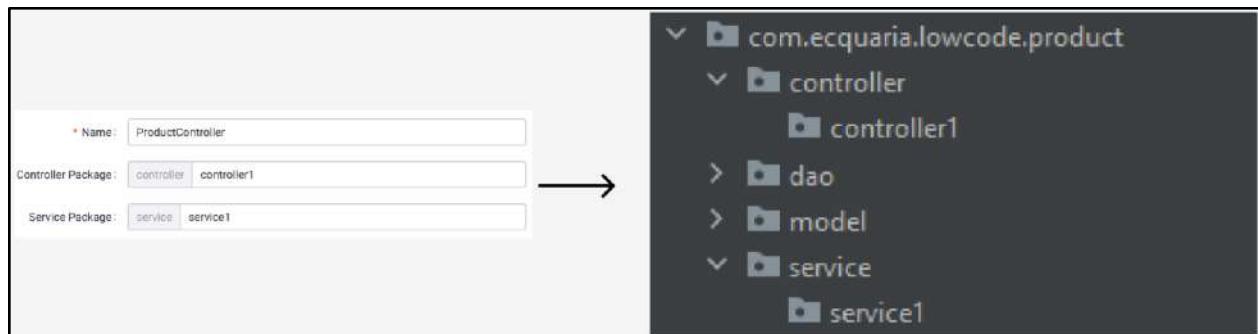
- **Name:** ProductController
A descriptive name that reflects the functionality it controls (e.g., "ProductController" for product-related operations or "OrderController" for order-related tasks).
- **(4th level) Controller Package:** <BLANK> (Default value: <BLANK>)
The package where the controllers interface will be placed. Controllers handle incoming HTTP requests and route them to the appropriate service methods, providing endpoints for the API.
- **(4th level) Service Package:** <BLANK> (Default value: <BLANK>)
This defines the location for service classes. The service layer contains business logic that performs operations related to the domain (e.g., product management), sitting between the controller and DAO layers.

The screenshot shows a configuration dialog for adding a controller. It has three input fields: 'Name' (ProductController), 'Controller Package' (controller), and 'Service Package' (service). A blue 'Apply' button is at the bottom right.

DataPoolController "ProductController" created successfully

For illustration purpose only

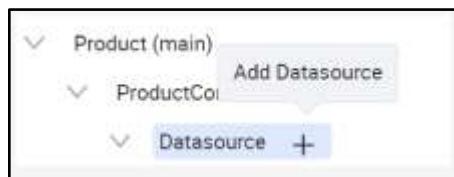
In certain scenarios where we need complex application designs in controllers and services, adding a 4th layer controllers and services help make our code more maintainable. Below is an example of generated code output if 4th layer controller and service are specified in the configuration.



Adding Datasource & Configure API Endpoints

In KAIZEN, adding a datasource and configuring API endpoints allows developers to quickly connect design interfaces with live APIs. This feature enables seamless interaction between the front-end design and back-end services, supporting various HTTP methods (e.g., GET, POST) to retrieve or send data. By integrating datasource into your project, you can bind the API to the design interface, ensuring real-time communication between the user interface and actual service APIs, streamlining development without requiring extensive manual coding.

- Select **Add Datasource** under controller



- In the configuration panel, a list of configuration items are as follows:
 - **Datasource ID:** A unique identifier for the datasource, used to reference it within your service.
 - **Identity:** A unique identifier, typically a key or ID, used to distinguish and manage the specific datasource connection.
 - **Location URL:** The endpoint URL where the API is hosted, used for sending requests.
 - **Version:** The version of the API being accessed, useful for managing compatibility.
 - **Params:** Parameters passed in the URL or body of the request, such as filters or identifiers.

- **Query/Body:** Defines where the parameters are sent, either in the query string (URL) or request body.
- **Encode Data:** Option to encode the data before sending it, typically used for secure or structured data formats like JSON.
- **Response Type:** Specifies the format of the response data (e.g., JSON, Text) expected from the API.
- **Method:** The HTTP method (e.g., GET, POST, PUT, DELETE) used to interact with the API.
- **Timeout (ms):** The maximum time in milliseconds to wait for a response before the request times out.
- **Request Headers:** Key-value pairs sent with the API request, often used for authentication or content-type definitions.
- **Add Function:** Allows the creation of custom functions to manipulate data or process the response after receiving it from the API.

The screenshot shows a configuration interface for a datasource. The fields include:

- * Datasource ID: [Text input]
- * Identity: [Text input]
- * Location URL: [Text input]
- * Version: [Text input] (with a dropdown arrow)
- Params: [Dropdown menu: Please Select] [Single (dropdown)] [List item: {/}]
- Query Body
- * Encode Data: [Switch button (on)]
- Response Type: [Dropdown menu: Please Select]
- * Method: [Dropdown menu: Please Select]
- * Timeout(ms): [Text input: 5000]
- Request Headers: [Text input: +Add]
- Add Function: [Text input: Choose to Add ▾]

Apply

- Create a **getProducts** datasource as follows

Datasource ID:

Identity:

Location URL:

Version:

Params	Type	Description	Operations
pageNo	Single	Integer	<input type="button" value="Delete"/> <input type="button" value="Edit"/>
pageSize	Single	Integer	<input type="button" value="Delete"/> <input type="button" value="Edit"/>
name	Single	String	<input type="button" value="Delete"/> <input type="button" value="Edit"/>
category	Single	String	<input type="button" value="Delete"/> <input type="button" value="Edit"/>
location	Single	String	<input type="button" value="Delete"/> <input type="button" value="Edit"/>
sortBy	Single	String	<input type="button" value="Delete"/> <input type="button" value="Edit"/>
sortDir	Single	String	<input type="button" value="Delete"/> <input type="button" value="Edit"/>

Query Body

Encode Data:

Response Type:

Method:

Timeout(ms):

Request Headers:

Add Function:

Note that **Params** allows toggling by clicking on **slash-icon (/)** and **+Add**. We will explore other Params feature in later tutorial

Params:

Params:

Params: :

The details are as follows:

Field	Value
Datasource ID	getProducts

Identity	getProducts
Location URL	/gateway/product/api/v1/betraining/products
Version	1
Params	pageNo: Single (Integer)
	pageSize: Single (Integer)
	name: Single (String)
	category: Single (String)
	location: Single (String)
	sortBy: Single (String)
	sortDir: Single (String)
Query	True
Encode Data	False
Response Type	<Default>
Method	GET
Timeout (ms)	5000
Request Headers	<Default>
Add Function	<Default>

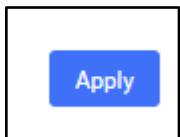
The "**Location URL**" parameter must follow the format `/gateway/<service>/api/v1`, where each segment plays a specific role in routing requests to the appropriate backend microservice:

- **/gateway/**: Serves as the main entry point, directing the request to the routing layer.
- **<service>**: This placeholder should be replaced with the unique name of the target microservice. It allows the gateway to identify and forward the request to the correct backend service.
- **/api/v1**: Specifies the API version to ensure backward compatibility and structured request handling.

This URL structure enables consistent routing within the microservice architecture, directing calls from the gateway to the designated backend services. Note that only requests formatted according to this convention will be correctly routed by the system

Note: The "Location URL" parameter currently does not support external API calls. Please ensure that all provided URLs are internal or self-hosted resources within the supported environment. External API functionality may be considered for future updates.

- Click **Apply** to add the datasource



Tutorial 19: Datasource API Binding

This tutorial covers the following Learning Objectives:

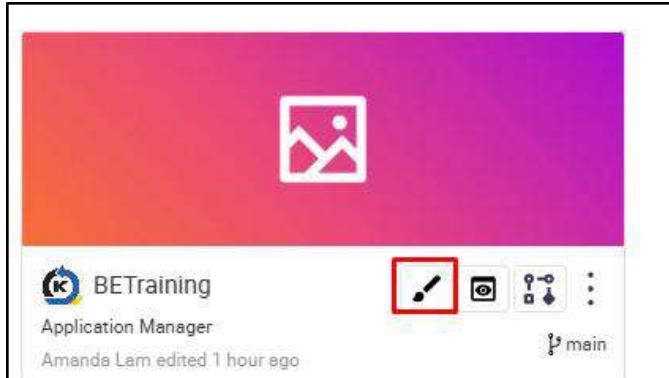
- Understand how to bind datasources to APIs using HTTP GET and POST methods within a low-code platform.
- Learn to handle and process various data types, datasources during API binding.
-

In this tutorial, you'll discover the Datasource API Binding feature of KAIZEN, which allows you to seamlessly connect and interact with external datasources. This feature enables you to bind APIs through GET and POST methods, retrieve and send data in diverse formats, and configure datasources to suit your application needs. By mastering these capabilities, you'll create dynamic and responsive applications that effectively manage data from multiple sources, enhancing their functionality and scalability.

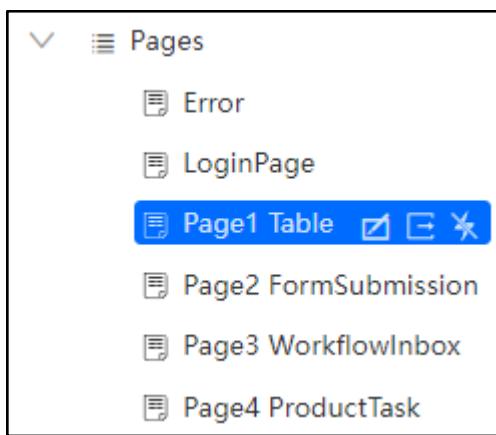
Practical 19.1: Binding API to Table (GET)

Binding an API to a frontend design enables seamless interaction between the user interface and backend services. This process allows developers to easily connect UI components to RESTful APIs, facilitating real-time data retrieval and manipulation with minimal coding. KAIZEN uses intuitive drag-and-drop interfaces, making it accessible for users with varying technical skills. This approach accelerates development, enhances maintainability, and allows teams to focus on business logic and user experience, leading to quicker deployment and improved application quality.

- Navigate to the App Designer page by clicking on the **Design** icon



- Go to **Page1 Table** page by clicking on the Resource panel



- Click **Preview**  to enter the preview page, click **Confirm** using default roles

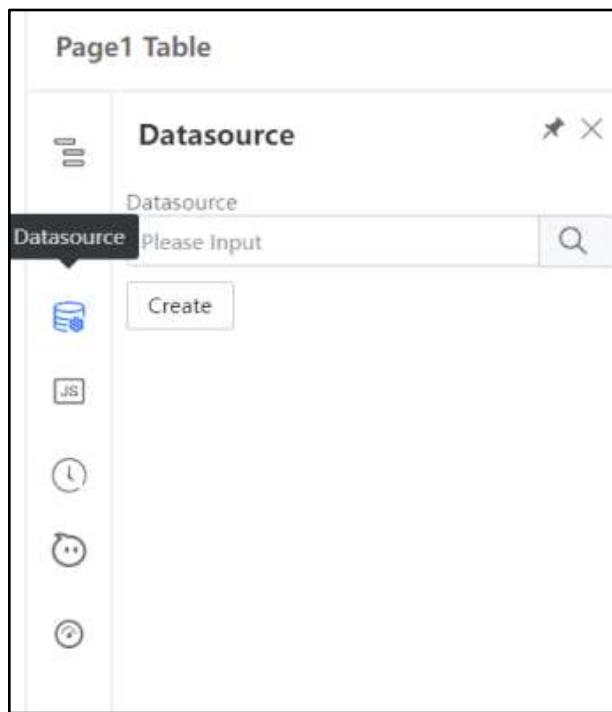


- Explore the client-side feature (Searching, Sorting, Pagination)

The screenshot shows a web-based application for searching products. At the top, there are three input fields: 'Product Name' (Please input), 'Category' (Please select), and 'Location' (Please select). Below these is a 'Sorting' section with a dropdown menu set to 'ID'. There are two buttons: 'Search' and 'Clear'. The main area is a table with the following columns: ID, Product Name, Category, Brand, Color, Price, Location, Status, and Action. The table contains 9 rows of data. Each row has a 'Download PDF' button in the 'Action' column. At the bottom right, there is a 'Pagination' section with a dropdown menu set to '10'.

ID	Product Name	Category	Brand	Color	Price	Location	Status	Action
1	Product A	Electronics	Brand X	Black	499.99	Singapore	Approved	<button>Download PDF</button>
2	Product B	Clothing	Brand Y	Blue	29.99	Malaysia	Approved	<button>Download PDF</button>
3	Product C	Electronics	Brand Z	Silver	899.99	Singapore	Approved	<button>Download PDF</button>
4	Product D	Clothing	Brand X	Blue	49.99	Singapore	Approved	<button>Download PDF</button>
5	Product E	Electronics	Brand Y	Black	99.99	Malaysia	Approved	<button>Download PDF</button>
6	Product F	Home	Brand Z	Brown	599.99	Singapore	Approved	<button>Download PDF</button>
7	Product G	Clothing	Brand Y	Red	39.99	Singapore	Approved	<button>Download PDF</button>
8	Product H	Electronics	Brand X	White	399.99	Singapore	Approved	<button>Download PDF</button>
9	Product I	Clothing	Brand Z	Gray	59.99	Malaysia	Approved	<button>Download PDF</button>

- Select **Datasource** and click **Create**



- In the pop-up window, select **getProducts** and click **Next**



- In the datasource configuration panel, click the **V** icon and select **Expression**



It will change the text box to javascript expression, allowing values to be evaluated dynamically



- Ensure all other fields are changed to **Expression**

Params:	<input type="text" value="pageNo"/> :	<input type="text" value="{{ Please enter JS expression }}"/>	<input type="button" value="⇄"/> <input checked="" type="checkbox"/>
	<input type="text" value="pageSize"/> :	<input type="text" value="{{ Please enter JS expression }}"/>	<input type="button" value="⇄"/> <input checked="" type="checkbox"/>
	<input type="text" value="name"/> :	<input type="text" value="{{ Please enter JS expression }}"/>	<input type="button" value="⇄"/> <input checked="" type="checkbox"/>
	<input type="text" value="category"/> :	<input type="text" value="{{ Please enter JS expression }}"/>	<input type="button" value="⇄"/> <input checked="" type="checkbox"/>
	<input type="text" value="location"/> :	<input type="text" value="{{ Please enter JS expression }}"/>	<input type="button" value="⇄"/> <input checked="" type="checkbox"/>
	<input type="text" value="sortBy"/> :	<input type="text" value="{{ Please enter JS expression }}"/>	<input type="button" value="⇄"/> <input checked="" type="checkbox"/>
	<input type="text" value="sortDir"/> :	<input type="text" value="{{ Please enter JS expression }}"/>	<input type="button" value="⇄"/> <input checked="" type="checkbox"/> {/}

- Enter the following details for the datasource configuration and click **Create**:

pageNo	this.state.currentPage
name	this.state.name
pageSize	this.state.pageSize
sortBy	this.state.sortBy
location	this.state.location
category	this.state.category
sortDir	this.state.sortDir

Create Datasource

[Create](#) [Cancel](#)

* Identity: [?](#)

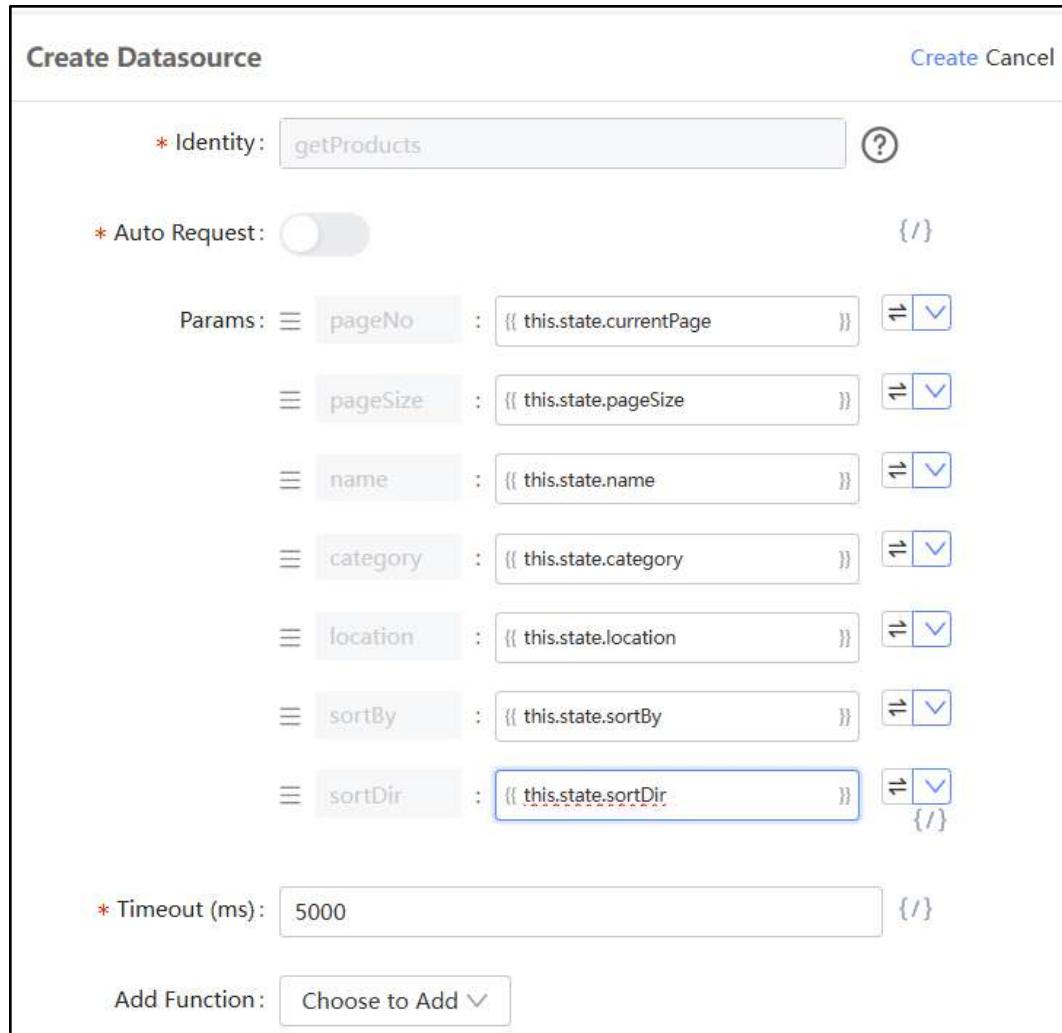
* Auto Request: [{/}](#)

Params:

pageNo	:	<code>{{ this.state.currentPage }}</code>	⤒	⤓
pageSize	:	<code>{{ this.state.pageSize }}</code>	⤒	⤓
name	:	<code>{{ this.state.name }}</code>	⤒	⤓
category	:	<code>{{ this.state.category }}</code>	⤒	⤓
location	:	<code>{{ this.state.location }}</code>	⤒	⤓
sortBy	:	<code>{{ this.state.sortBy }}</code>	⤒	⤓
sortDir	:	<code>{{ this.state.sortDir }}</code>	⤒	⤓

* Timeout (ms): [{/}](#)

Add Function: [Choose to Add ▾](#)



- Locate the **loadTable()** function in **Source Code Panel**

1. Click on the Source Code Panel

2. Locate the `loadTable()` function

```

142 // Call this function after doing changes (sort, paginate etc.) to load
143 // Client-side
144 const loadTable = () => {
145   // Filter data based on name, category, and location
146   const filteredData = data.filter(item => {
147     const itemName = item.name ?? item.name.toLowerCase(); // Check if name is present
148     const itemCategory = item.category ?? item.category.toLowerCase();
149     const itemLocation = item.location ?? item.location.toLowerCase();
150
151     const nameMatch = !name || (itemName ?? itemName.includes(name.toLowerCase()));
152     const categoryMatch = !category || (itemCategory ?? itemCategory.includes(category));
153     const locationMatch = !location || (itemLocation ?? itemLocation.includes(location));
154
155     return nameMatch ?? categoryMatch ?? locationMatch;
156   });
157
158   // Sort the filtered data based on sortBy and sortDir
159   const sortedData = filteredData.sort((a, b) => {
160     const aValue = a[sortBy];
161     const bValue = b[sortBy];
162
163     if (sortDir === 'asc') {
164       return aValue < bValue ? -1 : aValue > bValue ? 1 : 0;
165     } else {
166       return aValue > bValue ? -1 : aValue < bValue ? 1 : 0;
167     }
168   });
169
170   // Paginate the sorted data
171   const startIndex = currentPage * pageSize;
172   const endIndex = startIndex + pageSize;
173   const pageTable = sortedData.slice(startIndex, endIndex);
174
175   this.setState({
176     table: pageTable,
177     dataCount: this.state.data.length
178   });
179
180
181
182   // Server-side
183   // this.dataSourceMap('getProducts').load().then(res => {
184   //   console.log(res)
185   //   this.setState({
186   //     "table": res.data.content,
187   //     "currentPage": res.data.number,
188   //     "dataCount": res.data.totalElements,
189   //     "pageSize": res.data.size
190   //   })
191   // })
192
193 }

```

- Comment out Client-side code and Uncomment Server-side code and click **Save**
 - Shortcut key: [Ctrl + /](#)

Source Code Panel

```

143 // Call this function after doing changes (sort, paginate etc.) to load
144 // loadable()
145 // Client-side
146 // const { currentPage, pageSize, data, sortBy, sortDir, name, category,
147 // location } = filters;
148 // Filter data based on name, category, and location
149 const filteredData = data.filter(item => {
150   const itemName = item.name ?? item.name.toLowerCase(); // Check if
151   const itemCategory = item.category ?? item.category.toLowerCase();
152   const itemLocation = item.location ?? item.location.toLowerCase();
153
154   const nameMatch = (name) ? (itemName === name) : true;
155   const categoryMatch = (category) ? (itemCategory === category) : true;
156   const locationMatch = (location) ? (itemLocation === location) : true;
157
158   return nameMatch && categoryMatch && locationMatch;
159 });
160
161 // Sort the filtered data based on sortBy and sortDir
162 const sortedData = filteredData.sort((a, b) => {
163   const aValue = a[sortBy];
164   const bValue = b[sortBy];
165
166   if (sortDir === 'asc') {
167     return aValue < bValue ? -1 : aValue > bValue ? 1 : 0;
168   } else {
169     return aValue > bValue ? -1 : aValue < bValue ? 1 : 0;
170   }
171 });
172
173 // Paginate the sorted data
174 const startIndex = currentPage * pageSize;
175 const endIndex = startIndex + pageSize;
176 const paginatedTable = sortedData.slice(startIndex, endIndex);
177 this.setState({
178   table: paginatedTable,
179   dataCount: this.state.data.length
180 });
181
182 // Server-side
183 // this.dataSourceMap['getProducts'].load().then(res => {
184 //   console.log(res);
185 //   this.setState({
186 //     "table": res.data.content,
187 //     "currentPage": res.data.number,
188 //     "dataCount": res.data.totalElements,
189 //     "pageSize": res.data.size
190 //   });
191 // });
192 })

```

Comment out client-side code (Ctrl + /)

Source Code Panel

```

143 // Call this function after doing changes (sort, paginate etc.) to load
144 // loadable()
145 // Client-side
146 // const { currentPage, pageSize, data, sortBy, sortDir, name, category,
147 // location } = filters;
148 // Filter data based on name, category, and location
149 const filteredData = data.filter(item => {
150   const itemName = item.name ?? item.name.toLowerCase(); // Check if
151   const itemCategory = item.category ?? item.category.toLowerCase();
152   const itemLocation = item.location ?? item.location.toLowerCase();
153
154   const nameMatch = (name) ? (itemName === name) : true;
155   const categoryMatch = (category) ? (itemCategory === category) : true;
156   const locationMatch = (location) ? (itemLocation === location) : true;
157
158   return nameMatch && categoryMatch && locationMatch;
159 });
160
161 // Sort the filtered data based on sortBy and sortDir
162 const sortedData = filteredData.sort((a, b) => {
163   const aValue = a[sortBy];
164   const bValue = b[sortBy];
165
166   if (sortDir === 'asc') {
167     return aValue < bValue ? -1 : aValue > bValue ? 1 : 0;
168   } else {
169     return aValue > bValue ? -1 : aValue < bValue ? 1 : 0;
170   }
171 });
172
173 // Paginate the sorted data
174 const startIndex = currentPage * pageSize;
175 const endIndex = startIndex + pageSize;
176 const paginatedTable = sortedData.slice(startIndex, endIndex);
177 this.setState({
178   table: paginatedTable,
179   dataCount: this.state.data.length
180 });
181
182 // Server-side
183 this.dataSourceMap['getProducts'].load().then(res => {
184   console.log(res);
185   this.setState({
186     "table": res.data.content,
187     "currentPage": res.data.number,
188     "dataCount": res.data.totalElements,
189     "pageSize": res.data.size
190   });
191 });
192

```

Uncomment server-side code (Ctrl + /)

- **Preview** the page and observe Sorting, Searching and Pagination function are all server-side now
 - Inspect “Network tab” in developer tools to see that server side requests are firing (F12)

The screenshot shows a web application interface for searching products. At the top, there are three dropdown menus: 'ProductName' (Please enter), 'Category' (Please select), and 'Location' (Please select). Below these are two buttons: 'Search' and 'Clear'. The main area displays a table of products with columns: ID, Product Name, Category, Brand, Color, Price, Location, Status, and Action. The table contains five rows of data. To the right of the table, a network traffic monitor is visible, showing several requests to 'products?size=20&page=1'. The requests are color-coded by type: GET (blue), POST (orange), and others (green). The status column for these requests shows various HTTP status codes like 200, 201, 202, 204, and 205.

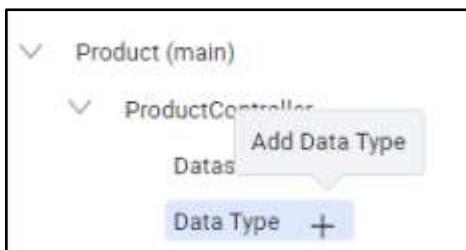
Practical 19.2: Binding API to Form submission (POST)

In this practical, you will learn how to handle form submissions using the Fetch API in JavaScript to send a POST request. This enables the collection of user input and send it to a server for processing, such as saving data to a database or initiating some backend logic.

Adding DataType & Datasource

In the API Designer, defining data types and individual query parameters serves distinct but complementary purposes in the development of a robust API. When you define a data type, you create a reusable structure that represents an object in your application, such as a product or user. This data type encapsulates multiple attributes (fields) and their types, forming a Value Object (VO) in Java. VOs help maintain a clear and consistent representation of complex data throughout the application. By leveraging data types, developers can:

1. **Promote Reusability:** Once defined, a data type can be utilized across multiple endpoints, reducing redundancy and enhancing maintainability.
 2. **Simplify Code:** Using VOs simplifies method signatures and enhances code readability by allowing developers to work with well-defined objects rather than primitive types.
- Firstly, in Service API designer, click on **Add Data Type**



- Define the Data Type with the following details:
 - **Name:** ProductVO
 - **Params:**

name	Single	String
category	Single	String
brand	Single	String
color	Single	String
price	Single	String
location	Single	String

workflowName	Single	String
--------------	--------	--------

* Name: ProductVO

Params:

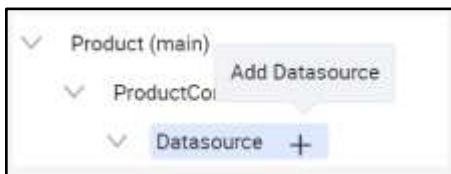
name	Single	String
category	Single	String
brand	Single	String
color	Single	String
price	Single	String
location	Single	String
workflowName	Single	String

+Add Apply

- Click **Apply** to create the Data Type



- Next, select **Add Datasource (+)** under controller



- Create a **postProduct** datasource with the following details:

Field	Value
Datasource ID	postProduct
Identity	postProduct
Location URL	/gateway/product/api/v1/betraining/products
Version	1
Params	ProductVO (Single)
Body	True

Encode Data	False
Response Type	<Default>
Method	POST
Timeout (ms)	5000
Request Headers	<Default>
Add Function	<Default>

The screenshot shows the configuration for a POST API named 'postProduct'. The configuration includes:

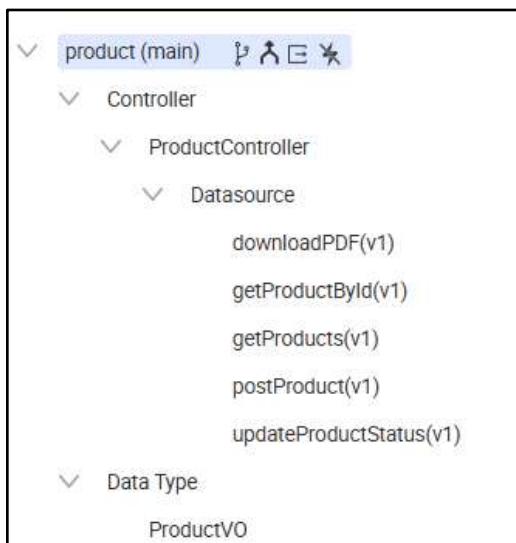
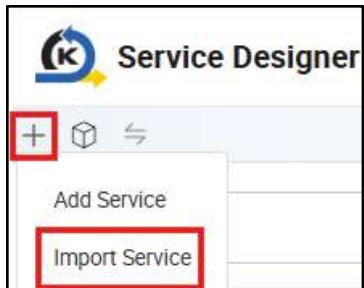
- Datasource ID:** postProduct (checked)
- Identity:** postProduct (checked)
- Location URL:** /gateway/product/api/v1/betraining/products (checked)
- Version:** v 1 (checked)
- Params:** ProductVO (dropdown) and Single (dropdown) (checked)
- Body:** Selected (checked)
- Encode Data:** Off (switch)
- Response Type:** Please Select (dropdown)
- Method:** POST (checked)
- Timeout(ms):** 5000 (checked)
- Request Headers:** +Add (button)
- Add Function:** Choose to Add (button)

An 'Apply' button is located at the bottom right.

Import Remaining APIs

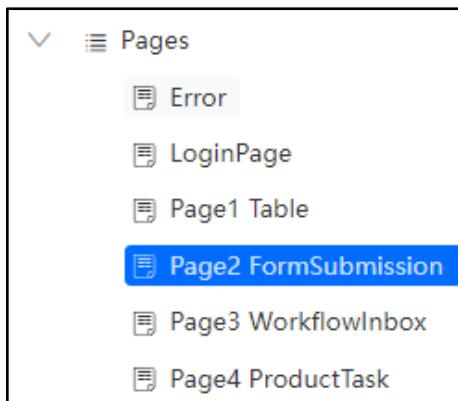
- In the Service Designer, click on 'Import Service' to import the remaining 3 APIs required for the BETraining application with the following file:

[kaizen-service\(cleaned\).xml](#)

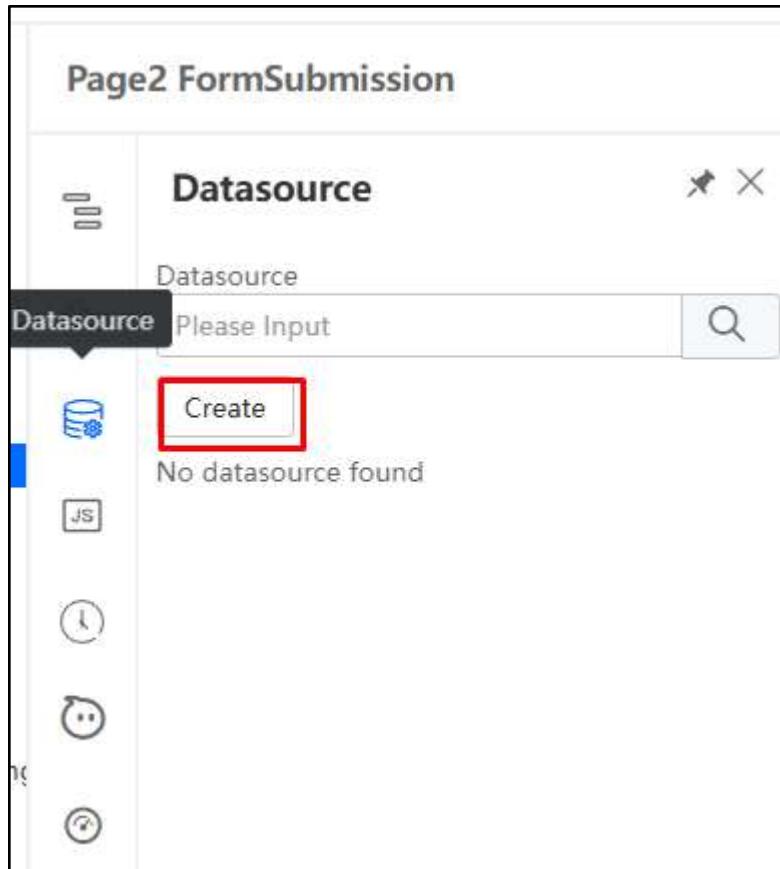


Binding to FormSubmission

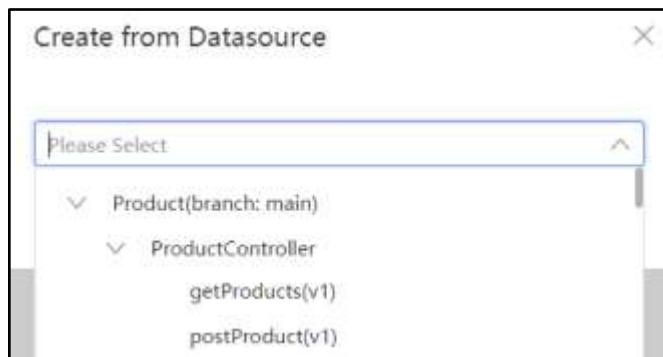
- Go back to the App Designer and navigate to the **Page2 FormSubmission** page



- Select **Datasource** and click **Create**



- In the pop-up window, select **postProduct** and click **Next**



- In the datasource configuration panel, click the **slash-icon (/) icon** to change to javascript expression.

*Note: This configuration simplifies the process by reducing redundancy, as you can handle multiple values with a single expression, rather than configuring each parameter individually.

Create Datasource

* Identity: postProduct [?](#)

* Auto Request: {/}

Params: [{/}](#)

* Timeout (ms): 5000 [{/}](#)

Add Function: Choose to Add ▾

Create Datasource

* Identity: postProduct [?](#)

* Auto Request: {/}

Params: [{/}](#)

* Timeout (ms): 5000 [{/}](#)

Add Function: Choose to Add ▾

- Enter the values:
 - **Params:** this.state.productForm

Create Datasource

* Identity: postProduct [?](#)

* Auto Request: {/}

Params: [{/}](#)

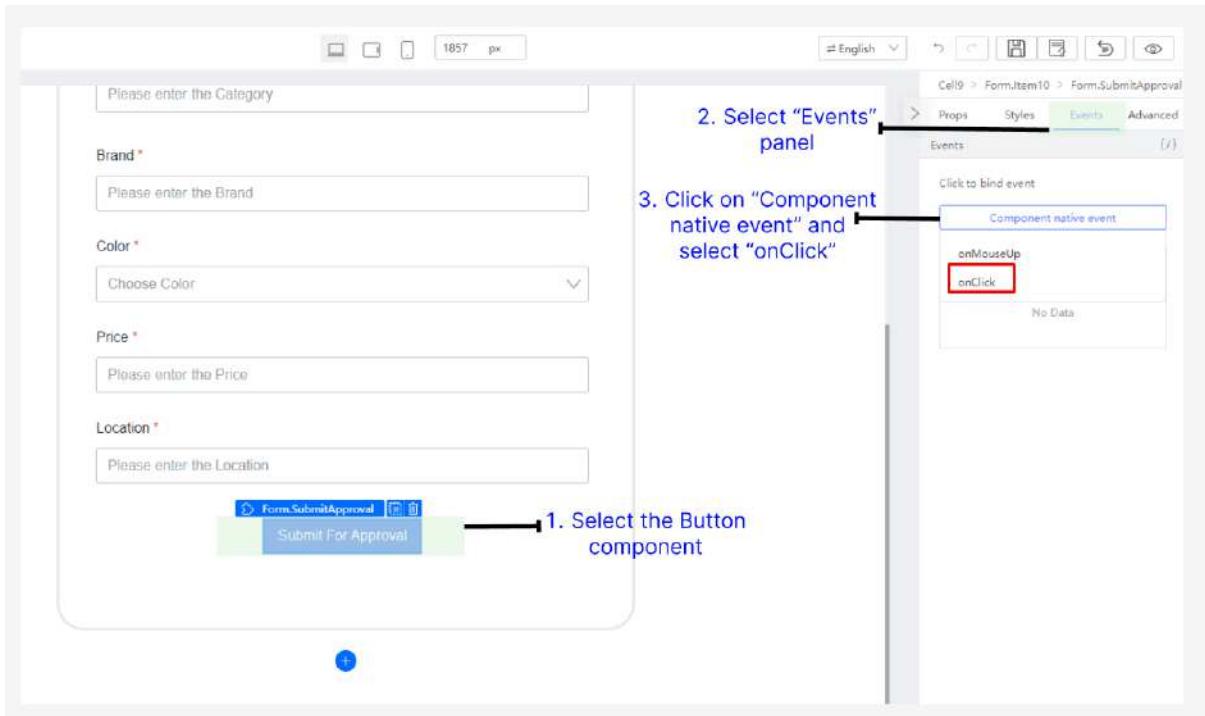
* Timeout (ms): 5000 [{/}](#)

Add Function: Choose to Add ▾

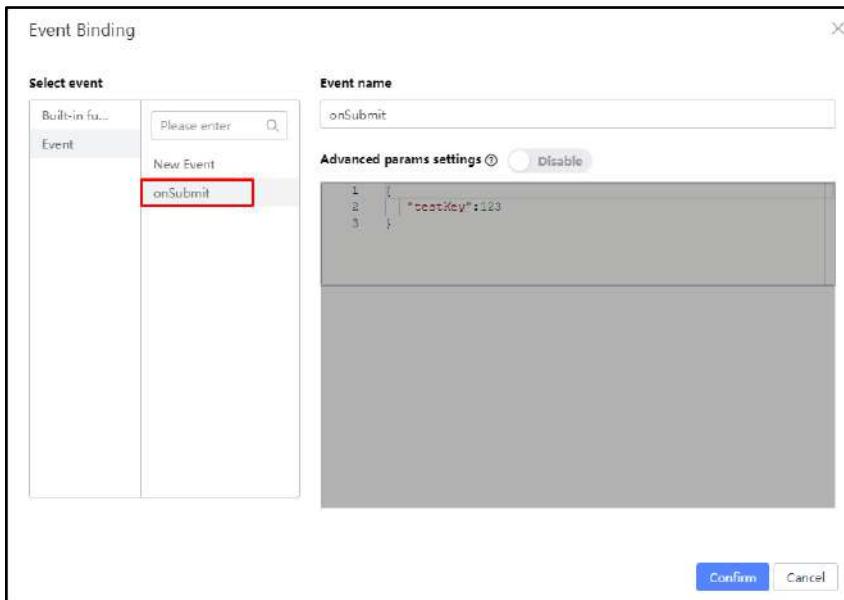
- Click **Create**

Create

- Next, Bind Button component to onClick event



- Select **onSubmit** and click **Confirm**



- Make sure to publish these changes made and click **Preview** to enter the preview page (which will also automatically publish for you)
- Open “Network tab” via developer tools to inspect the request being sent to server

The screenshot shows a web browser with the 'Add New Product' form open. The form contains fields for Product Name, Category, Brand, Color, Price, and Location. The 'Product Name' field has 'Nike Shoe' entered. The 'Category' field has 'Shoe'. The 'Brand' field has 'Nike'. The 'Color' field has 'Black'. The 'Price' field has '123'. The 'Location' field has 'Singapore'. Below the form is a blue 'Submit For Approval' button. To the right of the form is the browser's developer tools Network tab. It shows two requests named 'products'. The 'Payload' tab for the second request displays the following JSON:

```

{
  "name": "Nike Shoe",
  "category": "Nike",
  "brand": "Shoe",
  "color": "Black",
  "location": "Singapore",
  "name": "Nike Shoe",
  "price": "123",
  "workflowName": "traineeename"
}

```

The request payload name is set from the Form.Item Advanced property

The screenshot shows a browser with the 'Add New Product' form. The form fields are identical to the previous screenshot. To the right of the form is the Form editor. It shows the 'Advanced' tab for a selected item. In the 'Request Payload' section, the 'name' field is highlighted with a red box and an arrow pointing to it. The 'Request Payload' JSON is shown as follows:

```

{
  "name": "Product yongheng",
  "category": "Elec",
  "brand": "Nike",
  "category": "Electronics",
  "color": "Black",
  "location": "Singapore",
  "name": "Product yongheng",
  "price": "18"
}

```

Binding POST Form using “Encode”

To bypass security restrictions by converting special characters into a standardized format, form encoding allows data to be safely transmitted without triggering Web Application Firewall (WAF) rules. This ensures reliable communication with servers by preventing data from being blocked or misinterpreted.

- Modify the ‘postProduct’ API with the following new change:
 - **Encode Data:** true
 - **Request Headers:** Content-Type:

Datasource ID: postProduct

Identity: postProduct

Location URL: /gateway/product/api/v1/betraining/products

Version: 1

Params: ProductVO Single {/}

Query Body

*** Encode Data:**

Response Type: Please Select

Method: POST

Timeout(ms): 5000

Request Headers: Content-Type

+Add

Add Function: Choose to Add

Apply

- Fire the request, observe the payload is automatically encoded
- Recreate the Datasource by adding the headers **application/json**

Datasource

Datasource: Please Input

Create

No datasource found

Create Datasource

Identity: postProduct

Auto Request: {/}

Params: {{ this.state.productForm }} {/}

Timeout (ms): 5000 {/}

Headers: Content-Type : application/json {/}

Add Function: Choose to Add

Create Cancel

- Fire the request in preview, observe the payload is automatically encoded

X Headers **Payload** Preview Response Initiator >>

▼Request Payload [view parsed](#)

```
"eyJuYW1lIjoiUHJvZHVtblCBGb3JtIHdpdGggRGF0YSBFbmNvZGlubmNvZGluZyIsImNhdGVnb3J5IjoiQ2F0ZWdvcnkgRW5jb2RpbmcilCJicmFuZCI6IkJyW5kIEVuY29kaW5nIiwiY29sb3IiOiJCbGFjayIsInByaWN1IjoiMTAiLCJsb2NhdGlvbii6IlNpbmdhcG9yZSJ9"
```

Tutorial 20: Error Handling

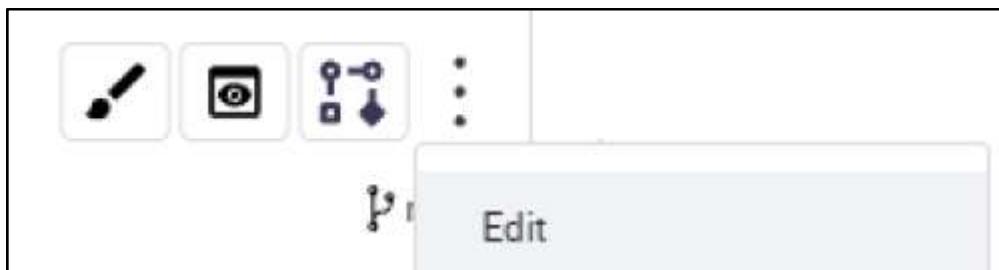
This tutorial covers the following Learning Objectives:

- Understand how to implement error handling mechanisms within the KAIZEN platform.
- Learn how to manage and resolve common errors encountered during application development.
- Improve application stability and user experience by effectively managing exceptions.

In this tutorial, you will explore error handling techniques within KAIZEN. By implementing robust error handling mechanisms, you will ensure that your application can gracefully handle and recover from errors, improving both the stability of the application and the overall user experience.

Practical 20.1: Redirect to custom error page

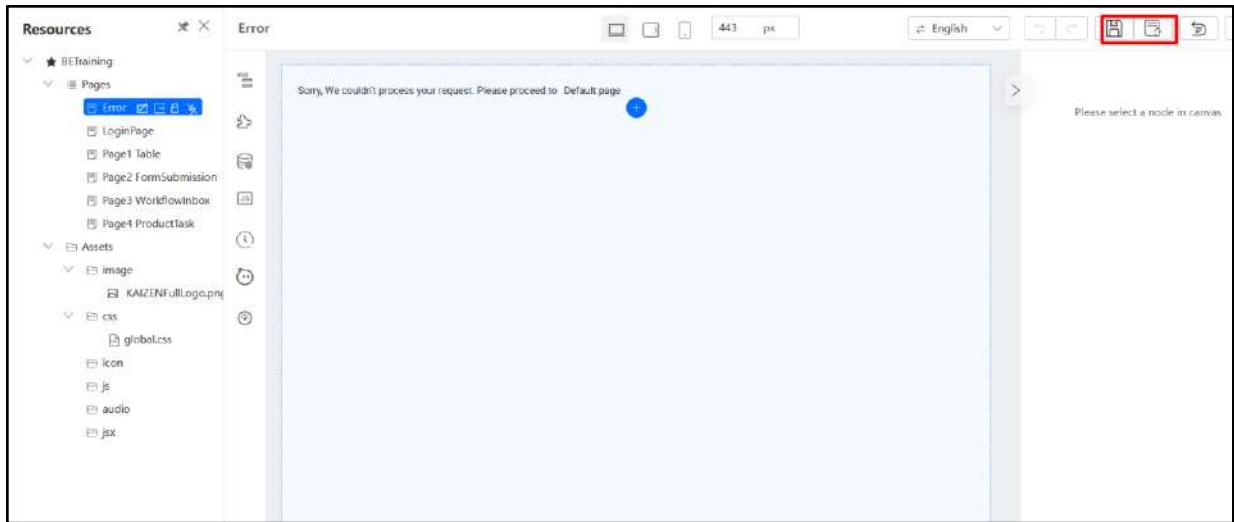
- Click on Edit Application



- (Note) View the **Error Pages** configuration:
 - **Forward Error:** Error will be forwarded to page
 - **To Error Page:** Error will be redirected to error page

Error Pages:	Error	Action	Path	Operation
	401	forwardError		Edit Delete
	4XX	toErrorPage	/error	Edit Delete
	5XX	toErrorPage	/error	Edit Delete

- Navigate to the Error page, click on **Save Draft** and **Publish Page**



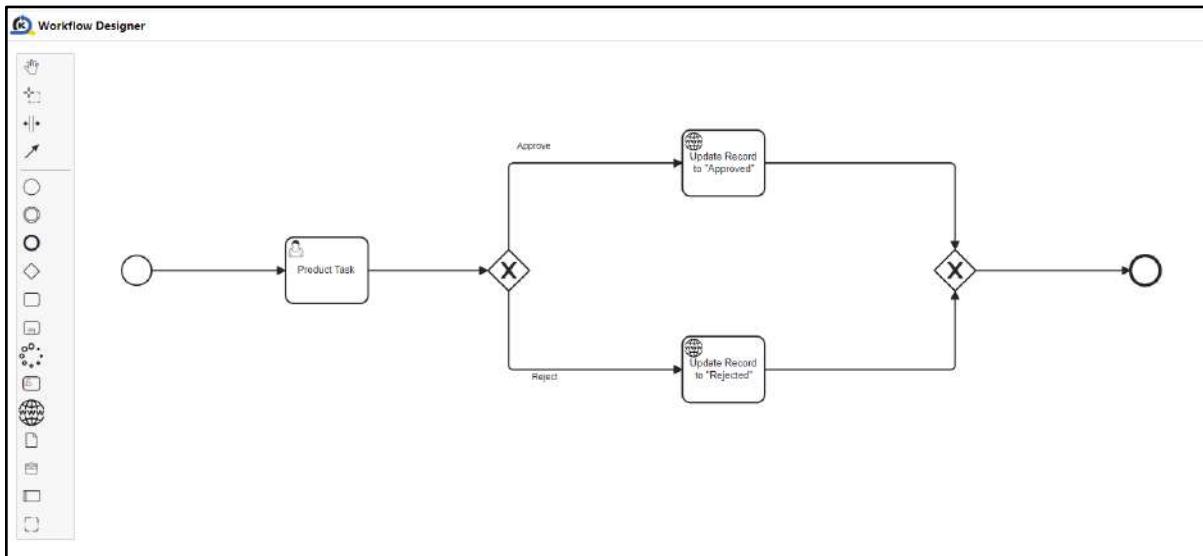
- We will go through this error handling effect in a later [tutorial on Code Generation](#) to showcase how pages will be redirected to custom error pages.

Tutorial 21: Workflow Designer

This tutorial covers the following Learning Objectives:

- Learn how to integrate a workflow system into your application to automate business processes.
- Understand how to manage and structure workflows to ensure efficient task sequencing.
- Enable automation of repetitive tasks through workflow management.

In this tutorial, you will explore how to integrate workflow features into your application, allowing for the automation of business processes. With workflows, you can define specific sequences of tasks required in your application, allowing you to design complex workflows of your business logic effortlessly.



Description of workflow:

Product Submission:

- The user submits product details.
- Product enters "Pending" state.

Product Approval / Rejection:

- If approved, the product transitions to "Approved" state.
- Approved products are visible to users.
- If rejected, the product transitions to "Rejected" state.
- Users are notified of rejection and may revise the product.

Practical 21.1: Using default workflow (Claim-based routing)

The backend APIs have been integrated with some of the workflow APIs, enabling smooth communication and data exchange between the application's backend system and the workflow management platform.

Task Claiming: When a task becomes available within the workflow, it is open for claiming by any eligible participant or user.

Claim Ownership: The participant who claims the task takes ownership and responsibility for its completion.

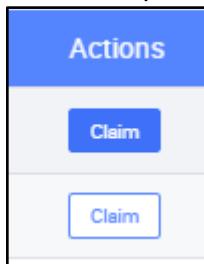
Task Completion: Once claimed, the participant is expected to complete the task within a specified timeframe or according to predefined criteria.

- Navigate to the “Workflow Inbox” page
 - **Task Claiming:** When a task becomes available within the workflow, it is open for claiming by any eligible participant or user. Hence, all users can see the task for any user.

The image shows two side-by-side browser windows. Both windows have a header "KAIZIN Studio - Console" and a tab "Workflow Inbox". The left window is labeled "Trainee A" and the right one is labeled "Trainee B". Both windows display a "Workflow Inbox" page with search and filter fields for Task Name, Group / Role Name, Status, Start Date, and Completed Date. Below these fields is a table showing two tasks. The first task is "Product Test" assigned to "traineeA" with status "Unfinished". The second task is "Product Test" assigned to "traineeB" with status "Unfinished". Each row in the table has a "Claim" button under the "Actions" column. At the bottom of each table, there is a "item per page" dropdown set to 10.

Task Name	User	Status	Suspended	Start Date	Completed Date	Actions
Product Test	traineeA	Unfinished	No	12-Jun-2024 11:21 PM		<button>Claim</button>
Product Test	traineeB	Unfinished	No	12-Jun-2024 11:21 PM		<button>Claim</button>

- Under “Actions” column, each user claim their own task by clicking on the “Claim” button
 - **Claim Ownership:** The participant who claims the task takes ownership and responsibility for its completion.



- After claiming their own task, each user is now able to see only his/her own task.
 - Task Completion:** Once claimed, the participant is expected to complete the task within a specified timeframe or according to predefined criteria.

Trainee A

Trainee B

Task Name	User	Status	Suspended	Start Date	Completed Date	Actions
Product Task	traineeA	Unfinished	No	12-Jun-2024 11:21 PM		<button>Approve / Reject</button>

- Click on **Approve/Reject** button to complete the task

Workflow Inbox

Task Name	User	Status	Suspended	Start Date	Completed Date	Actions
Product Task	traineeA	Unfinished	No	12-Jun-2024 11:21 PM		<button>Approve / Reject</button>

1. Click on "Approve/Reject" to complete the task

- According to the workflow, the task can either go through 'Approval' or 'Rejection' flow.
Click on either **Approve** or **Reject**

The screenshot shows a software interface with a header bar containing 'BE Training', 'Table', 'Form Submission', and 'Workflow Inbox'. Below this is a navigation bar with 'BE Training' and a language dropdown set to 'English'. The main content area features a rounded rectangular card titled 'Pending Approval'. Inside the card, there are six input fields with labels: 'Product Name' (value: 'TraineeA Product'), 'Category' (value: 'TraineeA Category'), 'Brand' (value: 'TraineeA Brand'), 'Color' (value: 'Black', with a dropdown arrow), 'Price' (value: '10'), and 'Location' (value: 'Singapore'). At the bottom of the card are two buttons: 'Reject' (white background) and 'Approve' (blue background). The entire card has a light gray shadow effect.

- In the workflow inbox, each user can see their own completed workflow by filtering status 'Finished'

The screenshot shows a software interface with a search bar at the top. On the left, there is a label 'Status' followed by a dropdown menu with the option 'Finished' selected. To the right of the dropdown is a small downward-pointing arrow icon. The rest of the screen is mostly blank white space.

Trainee A

Workflow Inbox

Task Name: Search Task Name
Group / Rule Name: Search Group / Rule Name
Status: Finished
Start Date: Start date End date
Completed Date: Start date End date

Task Name	User	Status	Suspended	Start Date	Completed Date	Actions
Project Task	TraineeA	Finished	No	12-Jun-2024 11:21 PM	12-Jun-2024 11:33 PM	View

Trainee B

Workflow Inbox

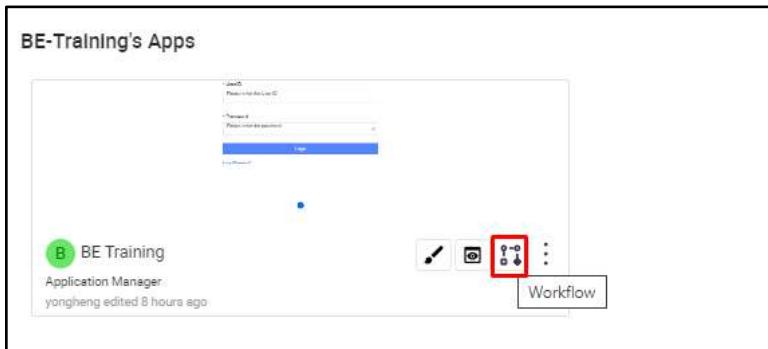
Task Name: Search Task Name
Group / Rule Name: Search Group / Rule Name
Status: Finished
Start Date: Start date End date
Completed Date: Start date End date

Task Name	User	Status	Suspended	Start Date	Completed Date	Actions
Project Task	TraineeB	Finished	No	12-Jun-2024 11:21 PM	12-Jun-2024 11:34 PM	View

Practical 21.2: Designing and Deploying custom workflow

This exercise involves designing and deploying a custom workflow by creating a sequence of automated steps to improve a business process. You'll start by analyzing the requirements, then design the workflow with tasks, conditions, and decision points. Next, implement the workflow by writing code or configuring a workflow tool.

- Click on the “Workflow” icon



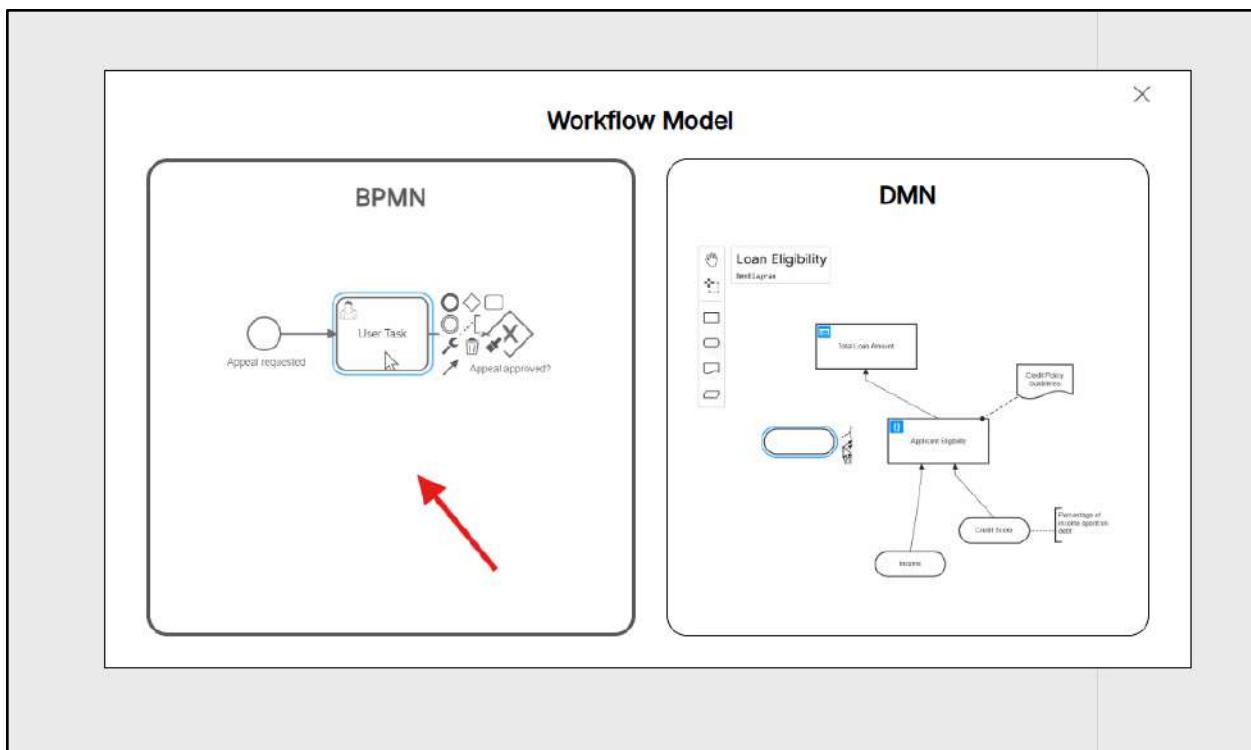
- Import the workflow
 - [BETraining_Workflow.bpmn](#)

▼

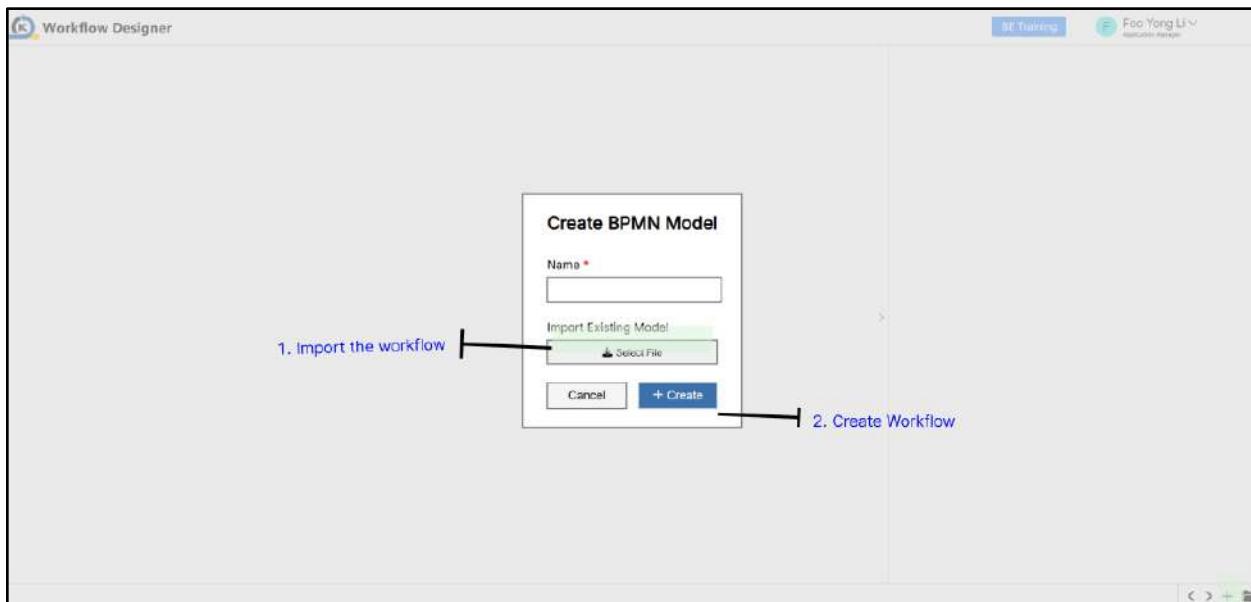
BPMN	BETrainingWorkflow	Draft 0.3
BPMN	BETrainingWorkflow	Draft 0.2
BPMN	BETrainingWorkflow	Draft 0.1
BPMN	BETrainingWorkflow_fooyongli	Draft 0.1
BPMN	BETrainingWorkflow_yongheng	Draft 0.4

+ Create Open

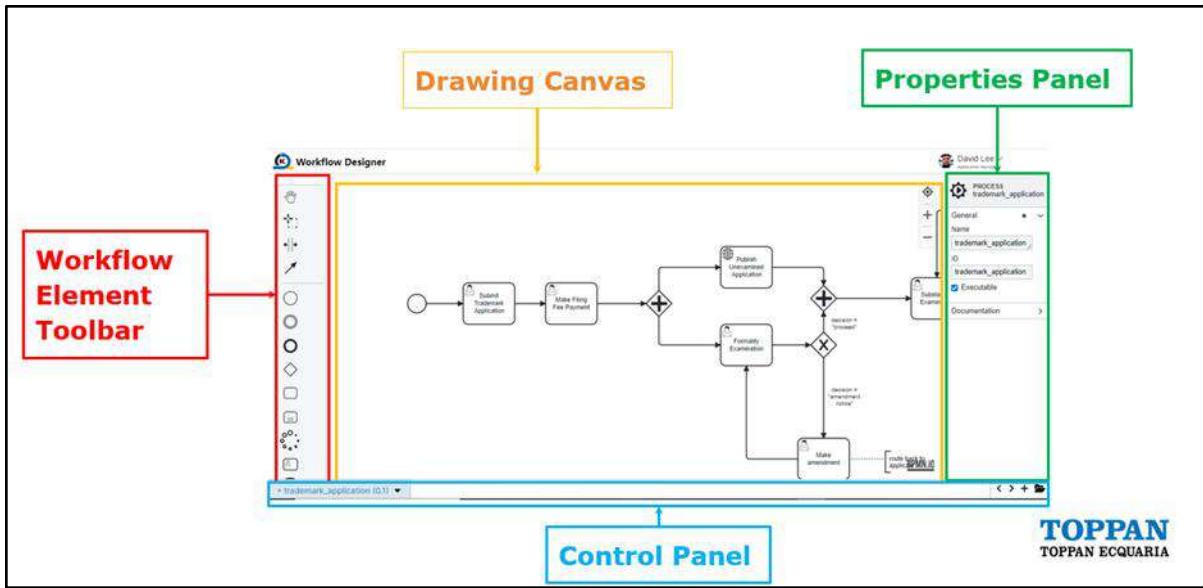
Cancel



Disclaimer: For this training , we will focus more on the BPMN workflow.

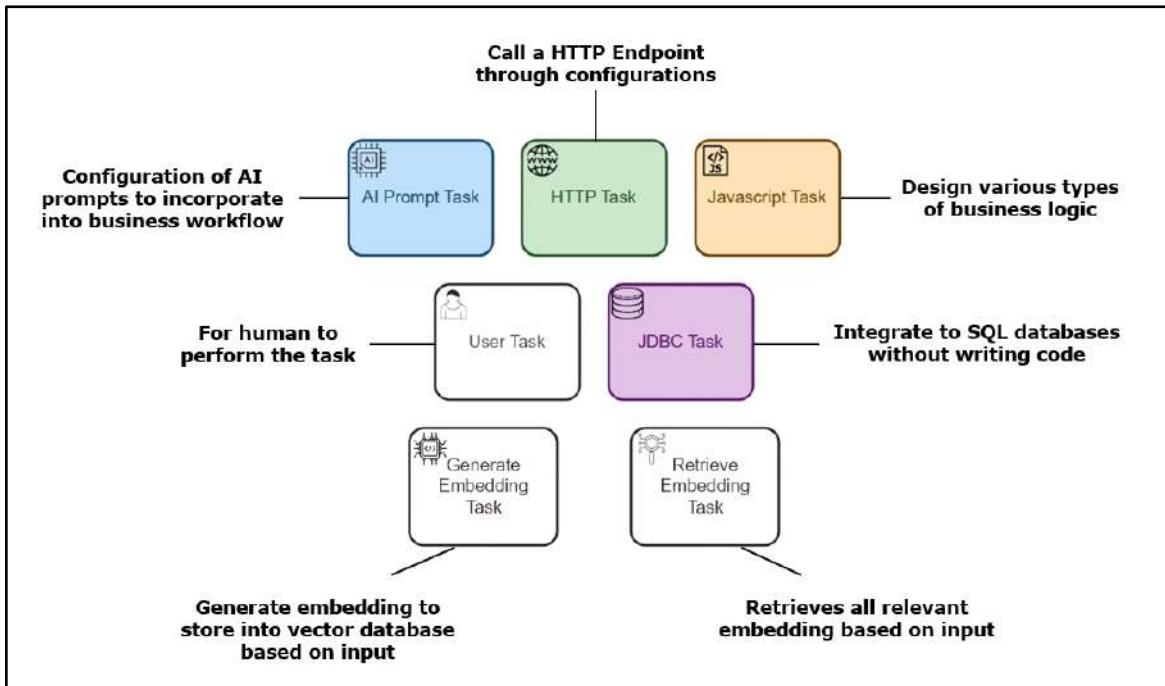


Import the downloaded workflow and press Create button.

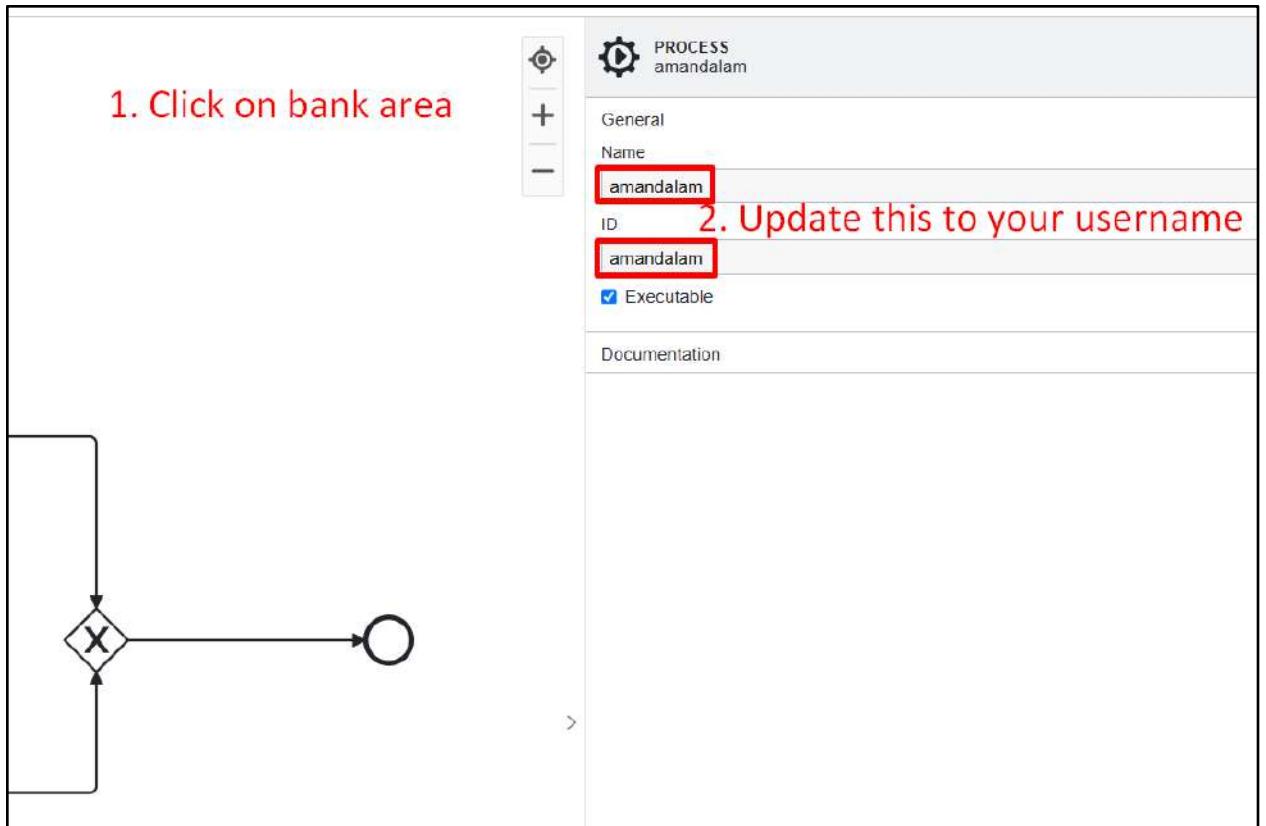


Disclaimer: Above is the sample description of workflow element, not representing the imported workflow.

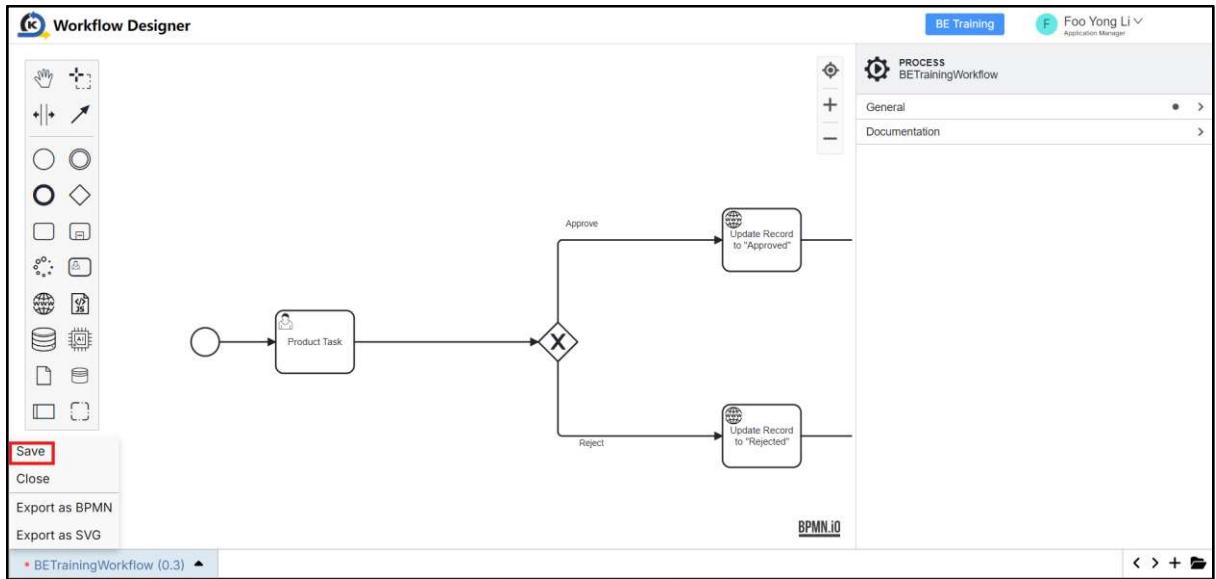
KAIZEN Workflow Task Elements



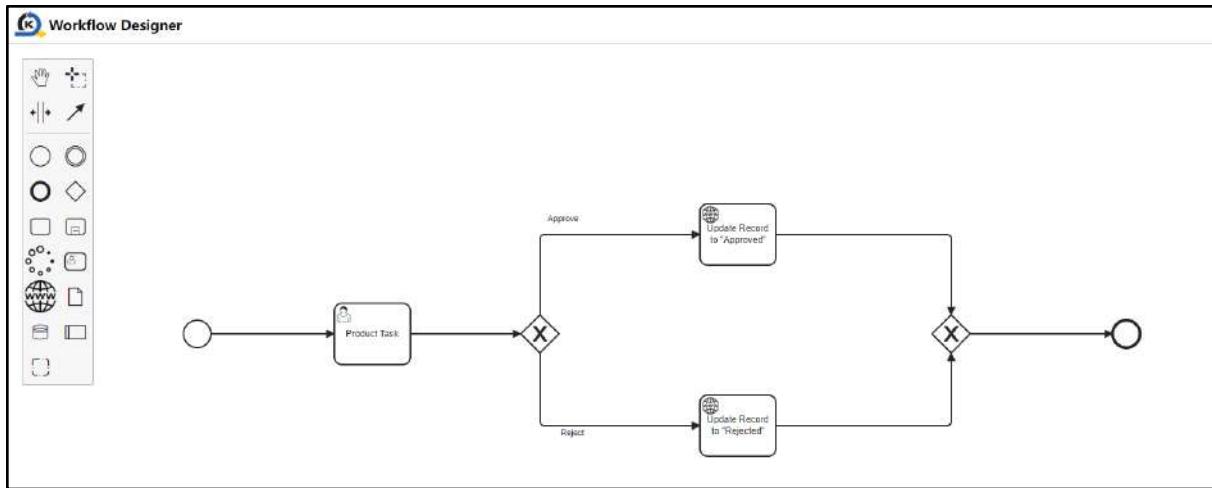
- In the imported workflow, rename it by changing the Process name to your username
 - <username>



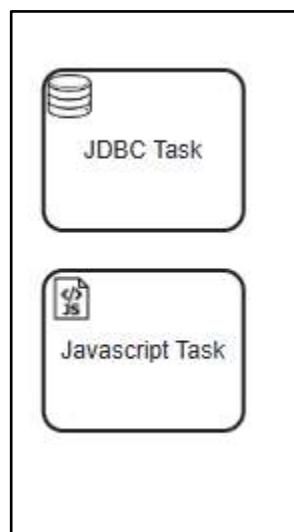
- Save the workflow



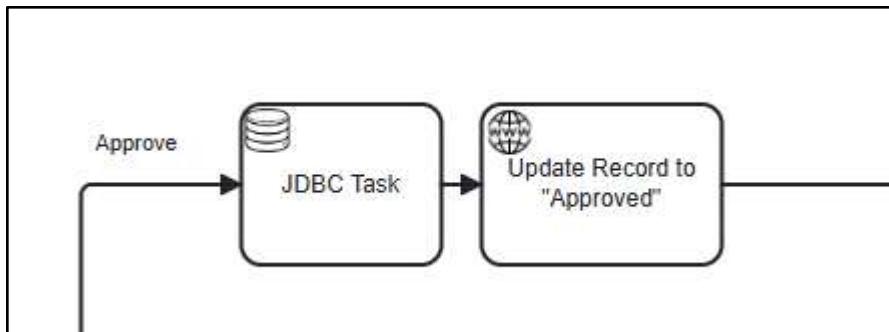
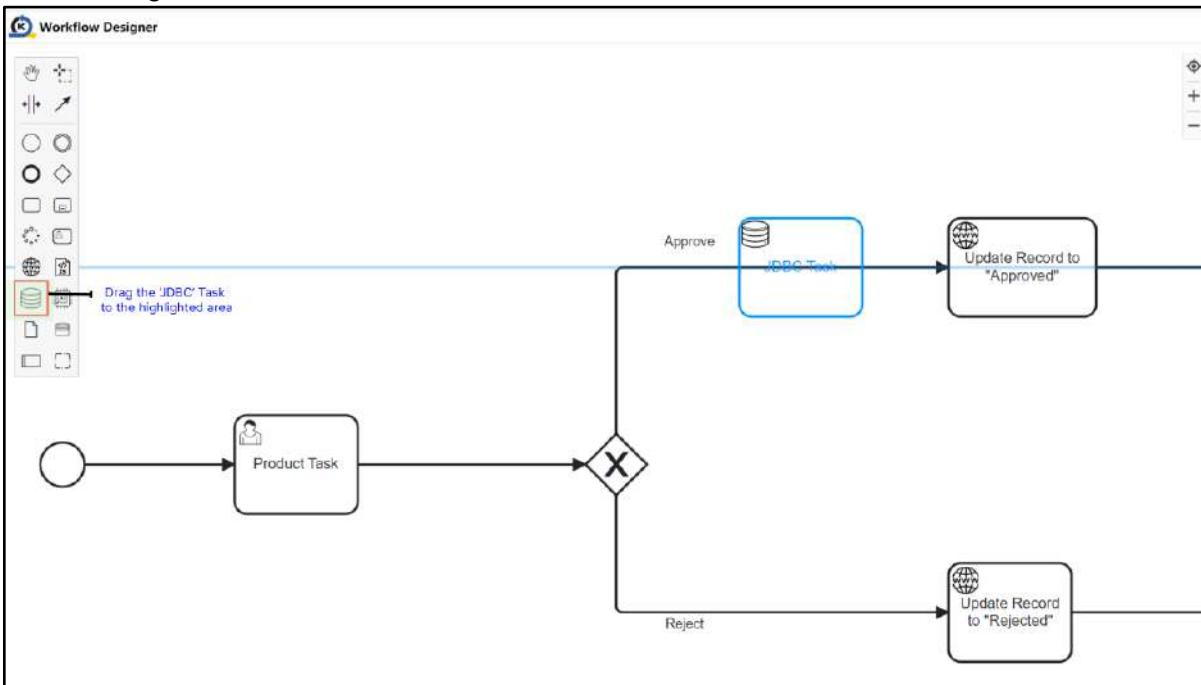
- Start designing the workflow



- We will be using the following two tasks for this training:
 - **JDBC Task**: involves connecting to a database using Java Database Connectivity (JDBC) to execute SQL queries and manage database operations. This task allows for reading, updating, and manipulating database records within a workflow, ensuring data consistency and enabling dynamic data-driven processes.
 - **Javascript Task**: involves writing and executing JavaScript code within a workflow to perform custom operations. This task can include data manipulation, making API calls, handling conditional logic, and other dynamic behaviors, allowing for flexibility and control over workflow execution.



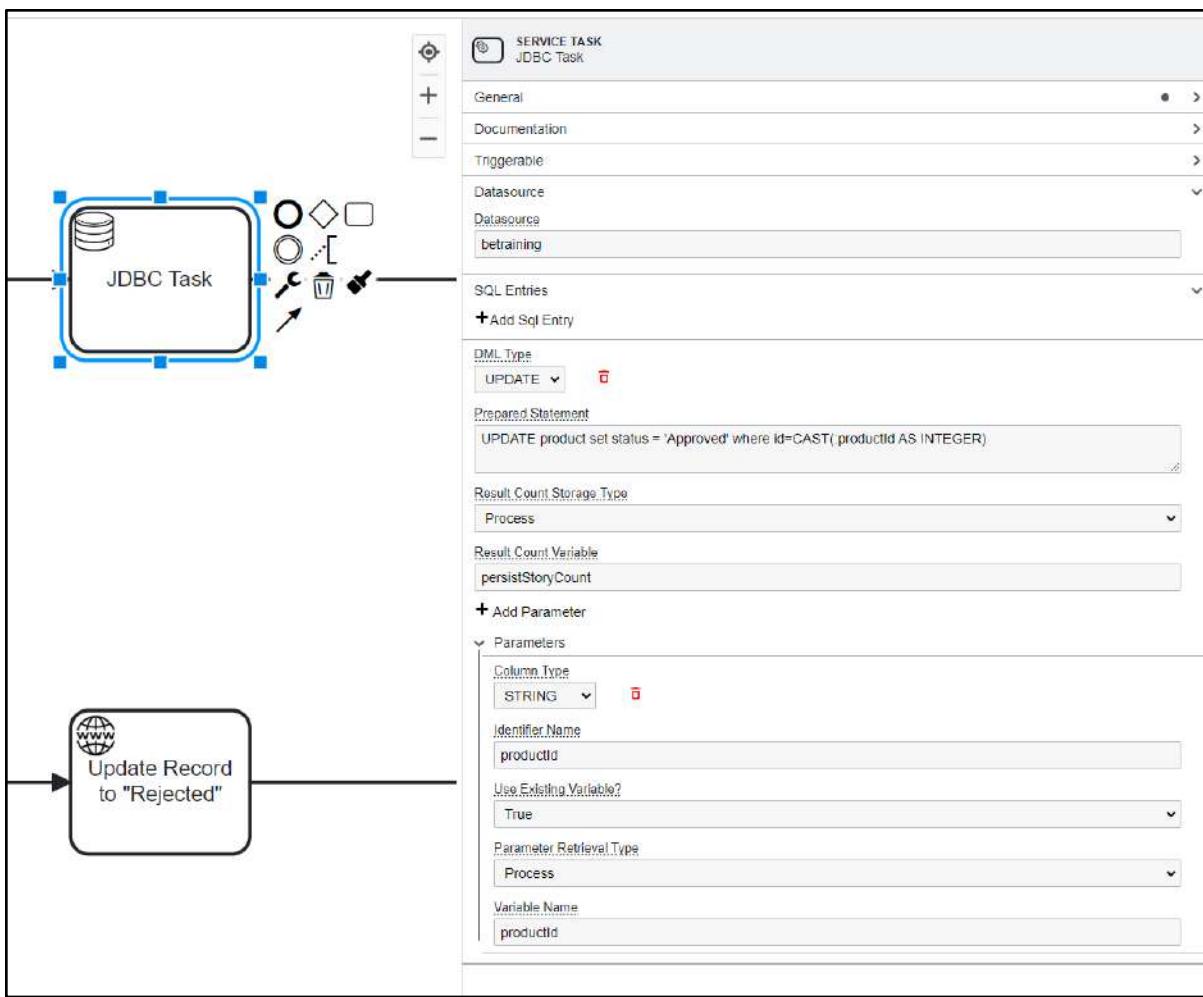
- Drag the JDBC task beside the HTTP Task



- Remove the original HTTP Task



- Update the JDBC task with the following values:

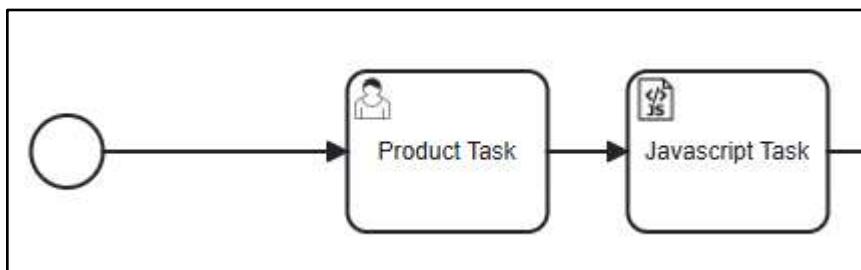
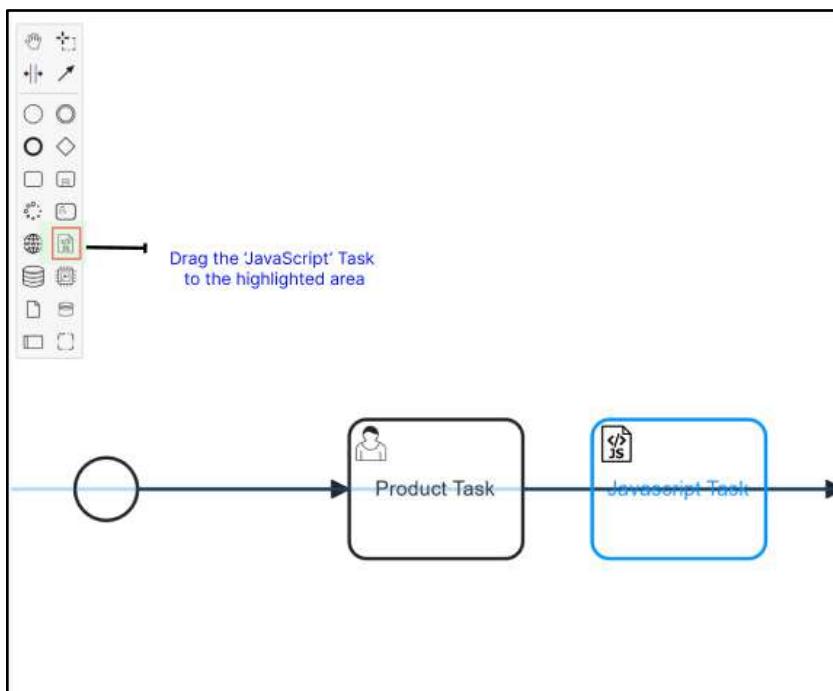


The JDBC task is trying to update the database directly by setting the status to 'Approved'. This database is the training central database that you have connected to via the Training Environment Profile earlier (the Profile feature will be explained in further details in Tutorial 23). The workflow feature is running on the central db so all trainees can see Approved and Rejected tasks.

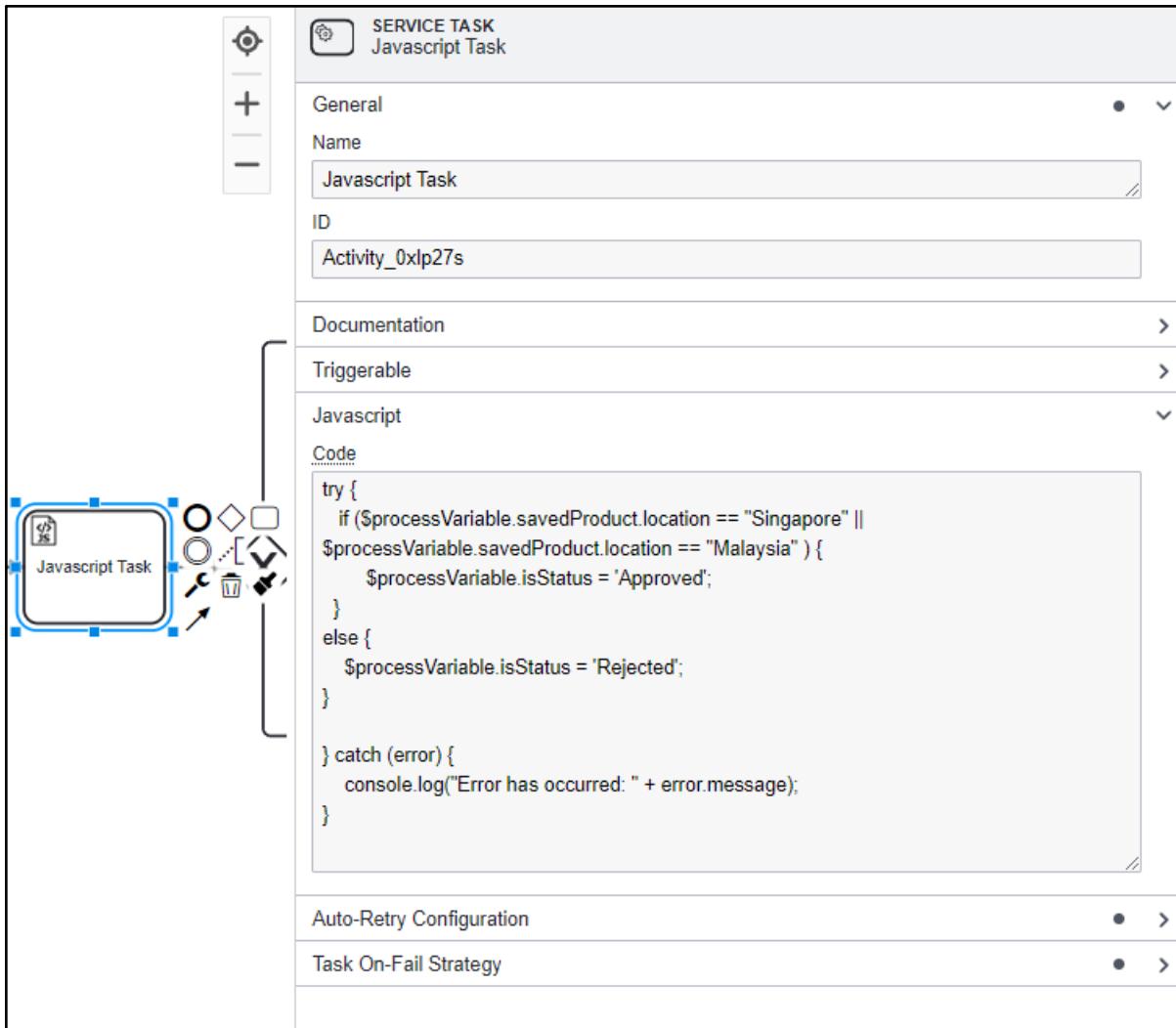
Datasource	betraining
DML Type	UPDATE
Prepared Statement	UPDATE product set status = 'Approved' where id=CAST(:productId AS INTEGER)
Result Count Storage Type	Process
Result Count Variable	persistStoryCount

Parameters	Column Type	STRING
	Identifier Name	productId
	Use Existing Variable	True
	Parameter Retrieval Type	Process
	Variable Name	productId

- Drag the Javascript Task to the workflow



- Update the Javascript task with the following javascript code:



This Javascript task ensures that the location must be 'Singapore' or 'Malaysia', else it will change the 'isStatus' to 'Rejected' and takes the Reject flow instead

```

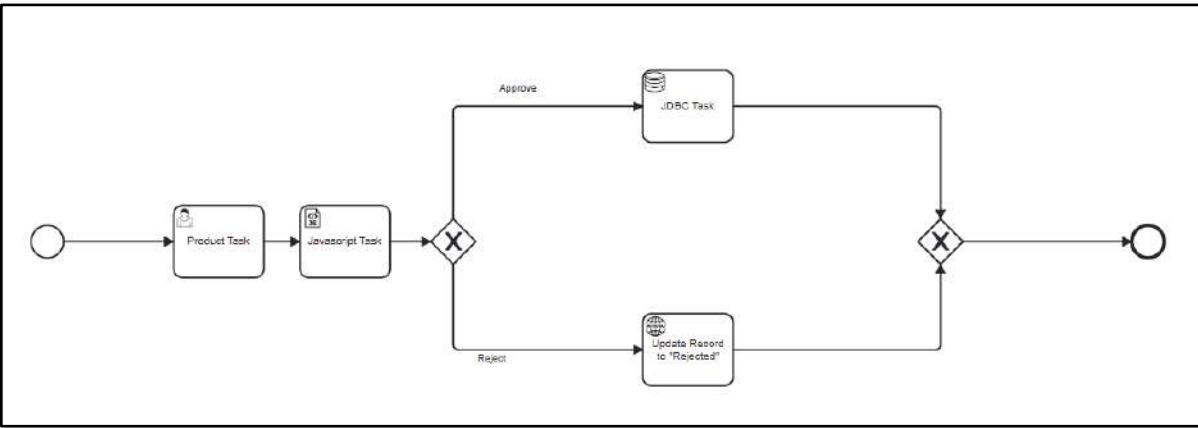
try {
    if ($processVariable.savedProduct.location == "Singapore" ||
$processVariable.savedProduct.location == "Malaysia" ) {
        $processVariable.isStatus = 'Approved';
    }
    else {
        $processVariable.isStatus = 'Rejected';
    }

} catch (error) {
    console.log("Error has occurred: " + error.message);
}

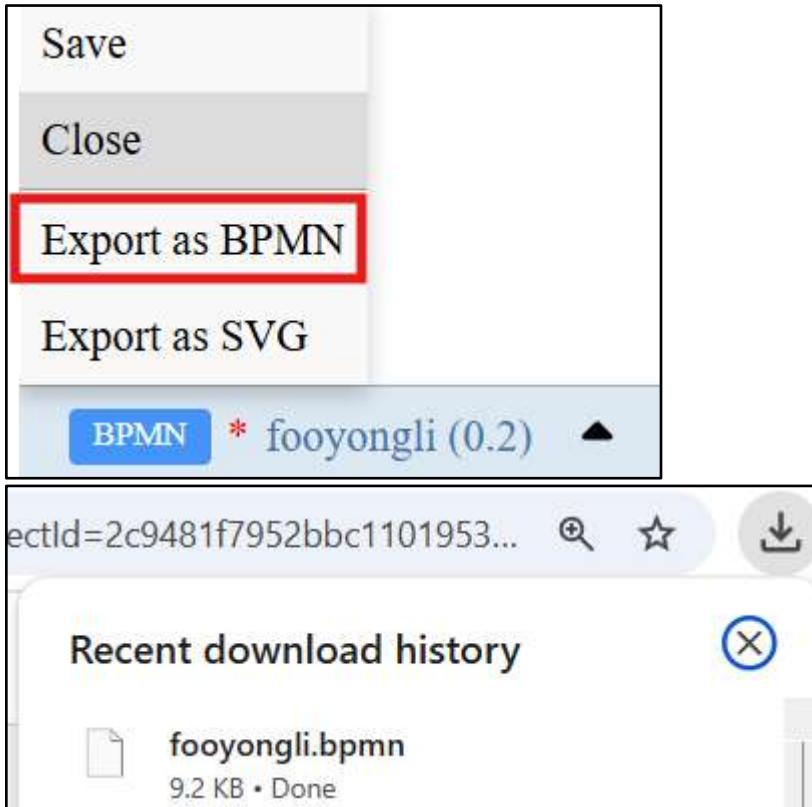
```



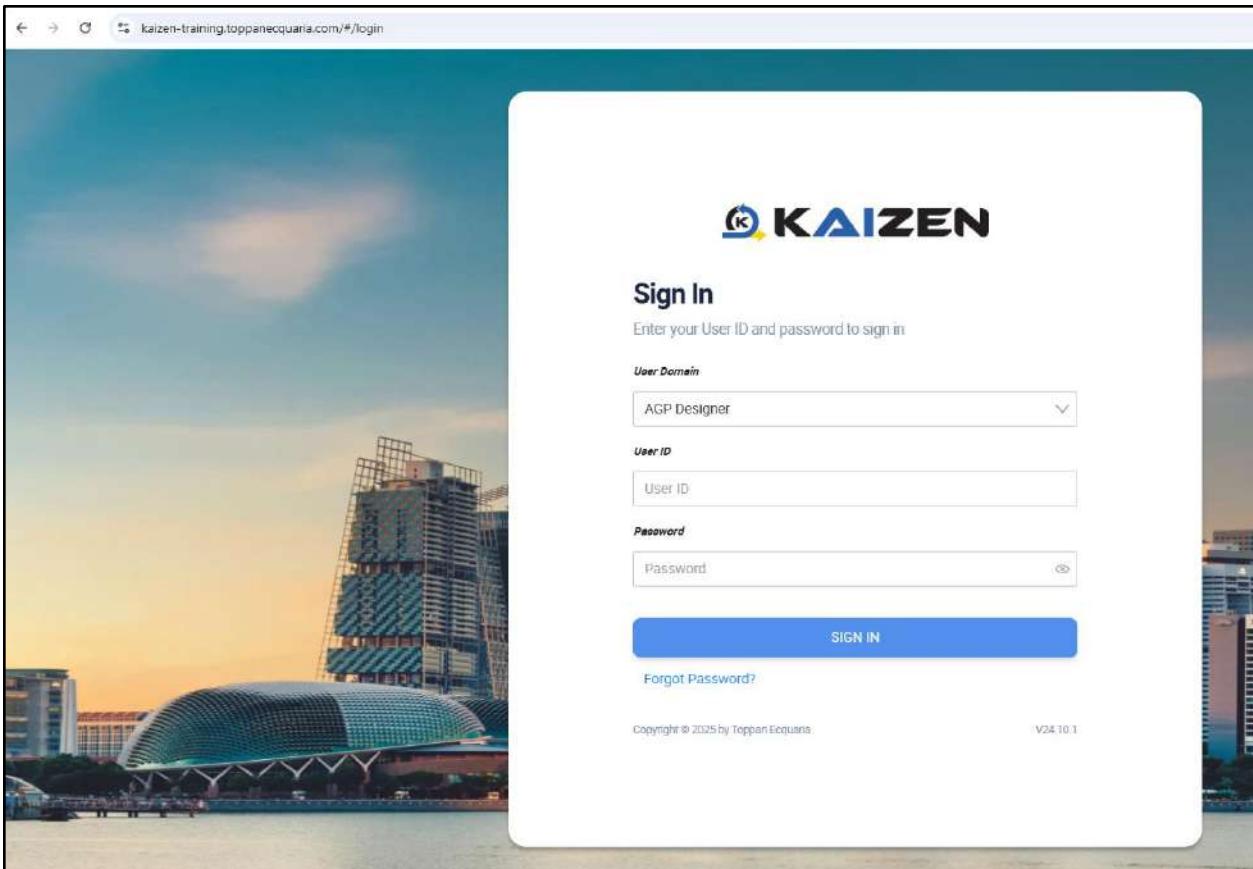
- Your complete workflow look like this



- Export your workflow



- Login to Training Environment (<https://kaizen-training.toppanecquaria.com/#/login>)
 - Username (Example: yongheng)
 - Password (Password\$1234)



- Navigate to Workflow Admin

KAIZEN Admin Console

Home Workflow/ Admin

Access Controller Job Scheduler Audit Trails Notification Management Workflow Admin

Instances Dashboard Integrations System Configuration Report

Workflow Name: Search Workflow
Model Type: Please select
Engine Version: Search Engine Version
Revision: Search Revision
Status: Please select
Deploy Date: Start date - End date
Activate Date: Start date - End date

Search Clear

- Deploy the workflow

KAIZEN Studio

Home Workflow Admin

Application Management Access Controller Job Scheduler Audit Trails Registered Client Notification Management Workflow Instances Development Tools System Configuration

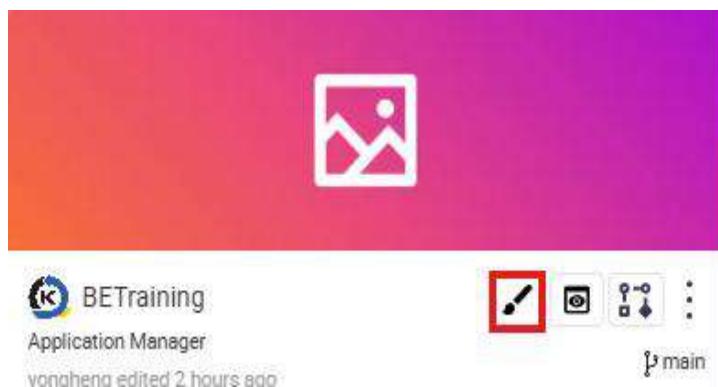
Workflow Name: Search Workflow
Engine Version: Search Engine Version
Revision: Search Revision
Status: Draft
Deploy Date: Start date - End date
Activate Date: Start date - End date

1. Deploy by uploading file 2. Or, via the action column

Workflow Name: testWorkflow Status: Draft Deploy Date: Activate Date: Actions

- To use your workflow, frontend code logic needs to be modified.

Navigate to BE-Training App



Make Sure Training Profile is selected and configured as below

A screenshot of the 'Profiles' configuration screen. On the left, there is a sidebar with various icons. The main area shows a list of profiles: 'Training Environment' (selected, highlighted in blue) and 'localhost'. The 'Training Environment' profile has its details expanded: Name: Training Environment, Url: https://kaizen-training.toppanecquaria.com/gwproxy, and a checked checkbox 'Use by viewer'. A modal window titled 'Edit Profile' is open over the list. It contains the following fields:

- * Name: Training Environment
- * Url: https://kaizen-training.toppanecquaria.com/gwproxy
- * User Domain: training_product
- * Account Id: amandalam Change to your username
- * Password: (redacted)

At the bottom of the modal are 'Test Connection', 'Cancel', and 'Save' buttons.

na

Under the Source Code Panel of the FormSubmission page, edit and replace the value with your username

Page2 FormSubmission

Source Code Panel

index.js index.css Save

```
1 class LowcodeComponent extends Component {
2     /** State Variables */
3     constructor() {
4         this.state = {
5             productForm: undefined,
6         }
7     }
8
9     onSubmit(value, errors, field) {
10        if (!errors) {
11            const updatedForm = {
12                ...value,
13                workflowName: "yongheng" // Change to your username
14            };
15
16            this.setState({
17                productForm: updatedForm,
18            }, () => {
19                console.log(this.state.productForm);
20                this.dataSourceMap['postProduct'].load().then(res => {
21                    window.Next.Message.success("Product Added Successfully!");
22                });
23            });
24        }
25    }
26}
27
28
29}
```

- Submit a form to trigger user task

Product Information

Product Name *

Category *

Brand *

Color *



Price *

Location *

Submit For Approval

- From the admin page, a new workflow instance was started.

Workflow Instances

Instance ID:	<input type="text" value="Search Instance ID"/>	<input type="button" value="Search"/>
--------------	---	---------------------------------------

Process Name:	<input type="text" value="Search Process Name"/>	<input type="button" value="Search"/>
---------------	--	---------------------------------------

Engine Version:	<input type="text" value="Search Engine Version"/>	<input type="button" value="Search"/>
-----------------	--	---------------------------------------

Revision:	<input type="text" value="Search Revision"/>	<input type="button" value="Search"/>
-----------	--	---------------------------------------

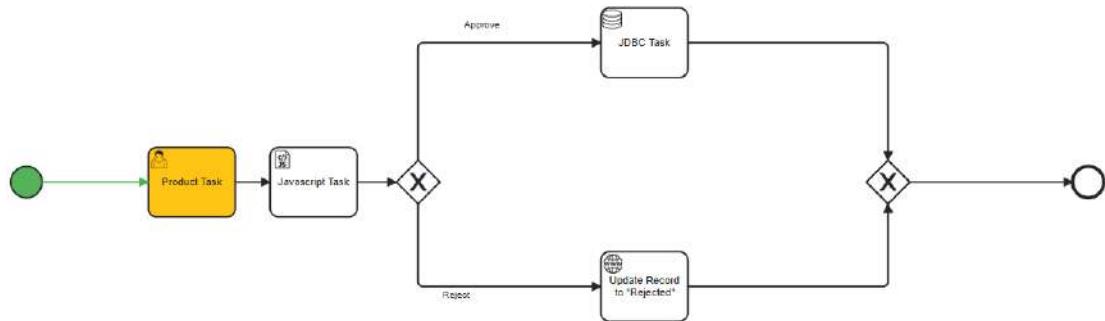
Status:	<input type="text" value="Please select"/>	<input type="button" value="Search"/>
---------	--	---------------------------------------

Start Date:	<input type="text" value="Start date"/> - <input type="text" value="End date"/>	<input type="button" value="Search"/>
-------------	---	---------------------------------------

End Date:	<input type="text" value="Start date"/> - <input type="text" value="End date"/>	<input type="button" value="Search"/>
-----------	---	---------------------------------------

Searched Instance(s):

Actions	Start Date	End Date	Status	Revision	Engine Ver.	Process Name	Instance ID
<input type="checkbox"/>	20-Feb-2025 05:41 PM		Running	3	24.10.1	yongheng	d47ac097-e10e-11ef-9be6-465ec18b311



- Open workflow inbox to claim and Reject the task that was submitted

Workflow Inbox

Task Name

Group / Role Name

Status

Start Date -

Completed Date -

Task Name	User	Status	Suspended	Start Date	Completed Date	Actions
Product Task	fooyongli	Unfinished	No	20-Feb-2025 05:41 PM		<input type="button" value="Claim"/>
Product Task	fooyongli	Unfinished	No	20-Feb-2025 05:35 PM		<input type="button" value="Claim"/>
Product Task	yongheng	Unfinished	No	20-Feb-2025 12:24 PM		<input type="button" value="Approve / Reject"/>

Item per page

1 / 1

Workflow Inbox

Task Name

Group / Role Name

Status

Start Date -

Completed Date -

Task Name	User	Status	Suspended	Start Date	Completed Date	Actions
Product Task	fooyongli	Unfinished	No	20-Feb-2025 05:41 PM		<input type="button" value="Approve / Reject"/>
Product Task	fooyongli	Unfinished	No	20-Feb-2025 05:35 PM		<input type="button" value="Claim"/>
Product Task	yongheng	Unfinished	No	20-Feb-2025 12:24 PM		<input type="button" value="Approve / Reject"/>

Item per page

1 / 1

Pending Approval

Product Name

Polo T X200

Category

Clothing

Brand

Polo

Color

White



Price

50

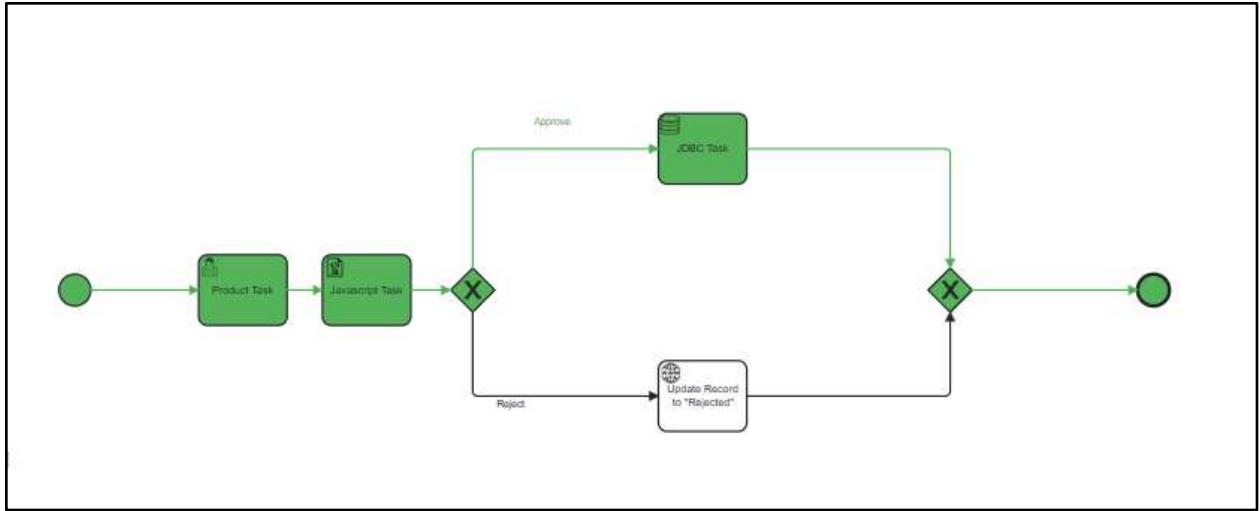
Location

Singapore

Reject

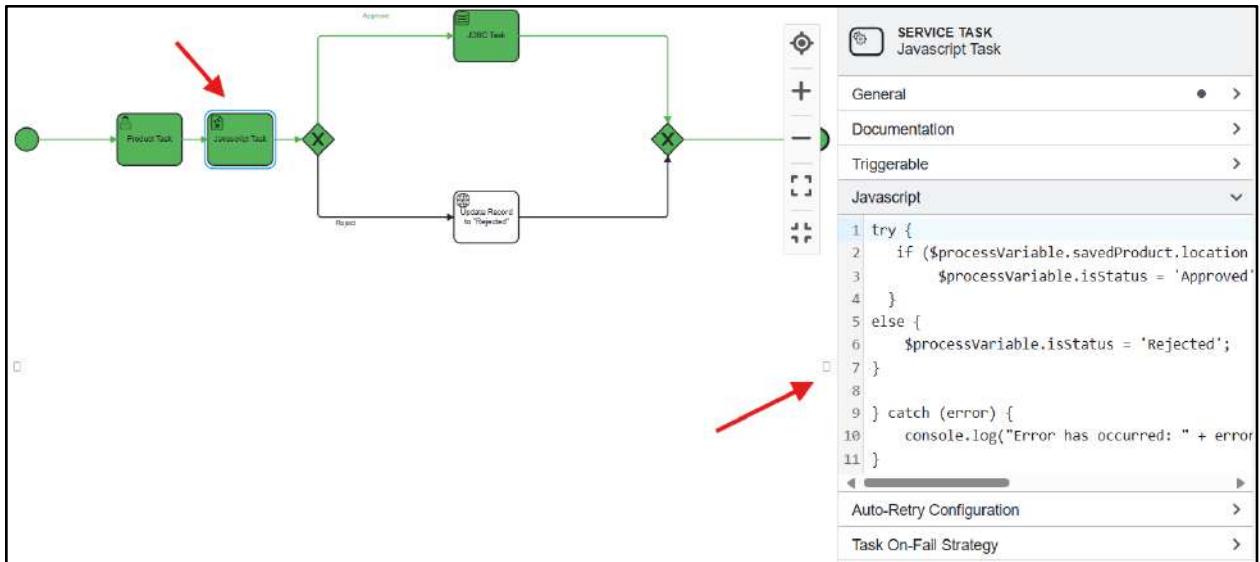
Approve

- After rejecting the task , the workflow instance has been completed as shown below.



- But why does it goes to “Approved” flow?

Previously we have added a javascript task to do location check in javascript in here. Thus if the location of product task was “Singapore” or “Malaysia”, the status will set to Approved”



The screenshot shows a Java code editor with two tabs open: `WorkflowTaskController.java` and `BeJobHandler.java`. The `WorkflowTaskController.java` tab is active, displaying the following code:

```
42  public class WorkflowTaskController {
43      public R<?> claim(@RequestBody ClaimTaskVO claimTaskVO) {
44          try {
45              workflowTaskService.claim(parseResult.result());
46          } catch (NotFoundException e) {
47              log.error("Unable to find task to claim", e);
48              return R.error(code: "404", errMsg: "Task to claim does not exist");
49          }
50
51          return R.ok();
52      }
53
54      @PostMapping("/complete")
55      public R<?> complete(@RequestBody CompleteRequestVO completeRequestVO) {
56          ParseResult<CompleteRequestDto> parseResult = completeRequestVOParser.parse(completeRequestVO);
57          if (parseResult.errors().hasErrors() || parseResult.result() == null) {
58              return R.error(code: "400", parseResult.errors());
59          }
60
61          try {
62              workflowTaskService.complete(parseResult.result());
63          } catch (NotFoundException e) {
64              log.error("Unable to find task to complete", e);
65              return R.error(code: "404", errMsg: "Task to complete does not exist");
66          } catch (StateException e) {
67              return R.error(code: "409", e.getMessage());
68          }
69      }
70  }
```

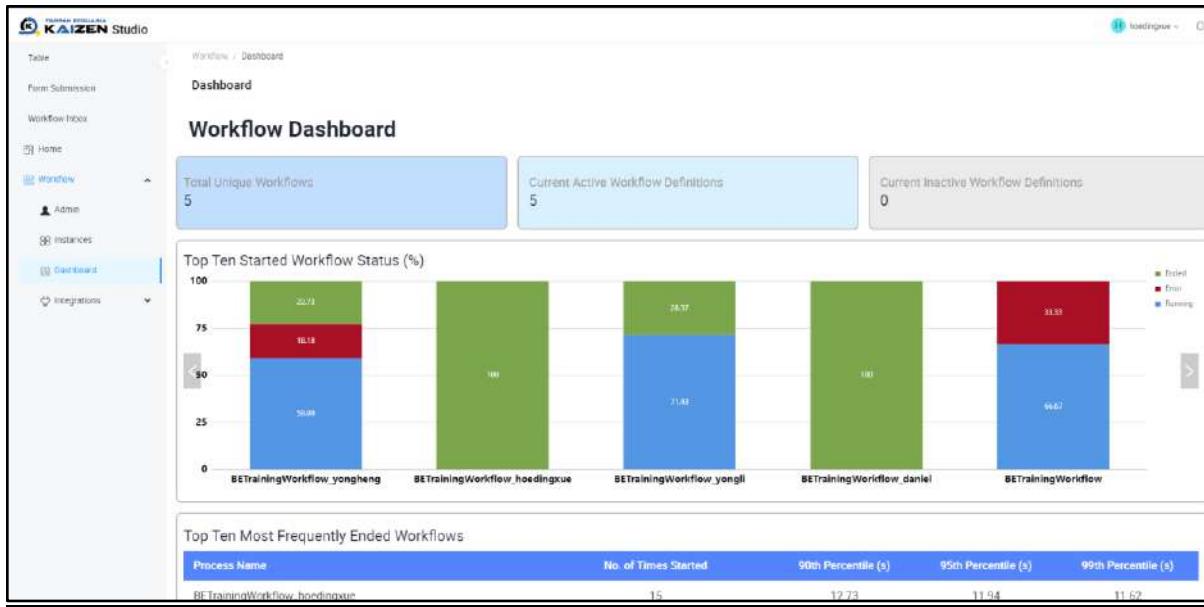
Practical 21.3: Workflow observability

In this practical 4.3, we will look into workflow observability features.

- Workflow Dashboard
- Workflow Instance Tracking
- Workflow Execution Timeline View

Workflow Dashboard

Dashboard allows users to gain overview of workflow running and status



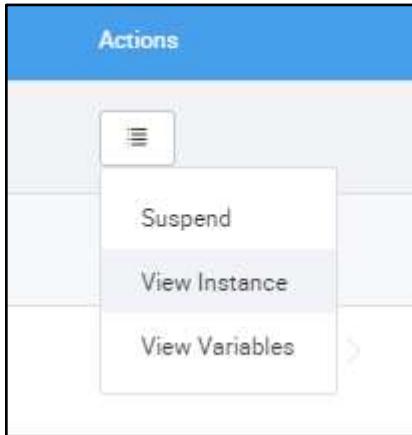
Workflow Instance Tracking

Track and monitor the live execution of your workflows visually with color-coded indicators for easy identification, allowing you to spot workflow issues at a glance. Instance variable and allows users to zoom in on specific tasks for quick diagnosis and resolution, by looking into the variables comparison passed between each workflow task. This will be useful in observing the variable transfer and changes in the workflow.

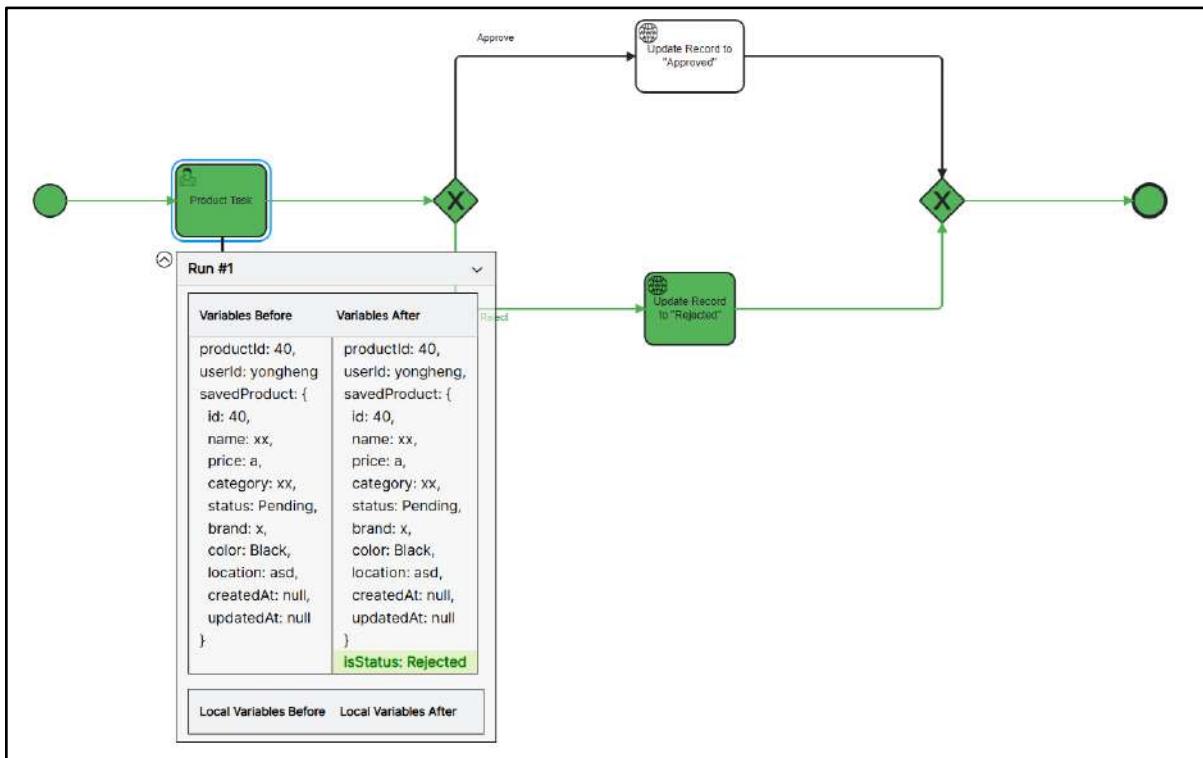
- Workflow Admin can also monitor the status of each workflow instances
 - For example, the two instances were the one created earlier

Instance ID	Process Name	Engine Version	Revision	Status	Start Date	End Date	Actions
711504e26511ef8710-69f911c3340f	BETrainingWorkflow	1.0.0	1	Running	12-Jun-2024 11:21 PM	12-Jun-2024 11:21 PM	
711504e26511ef8710-69f911c3340f	BETrainingWorkflow	1.0.0	1	Running	12-Jun-2024 11:21 PM	12-Jun-2024 11:21 PM	

- Click on “View Instance”



- By clicking on the page, we can view the instance variable values before and after the workflow task.



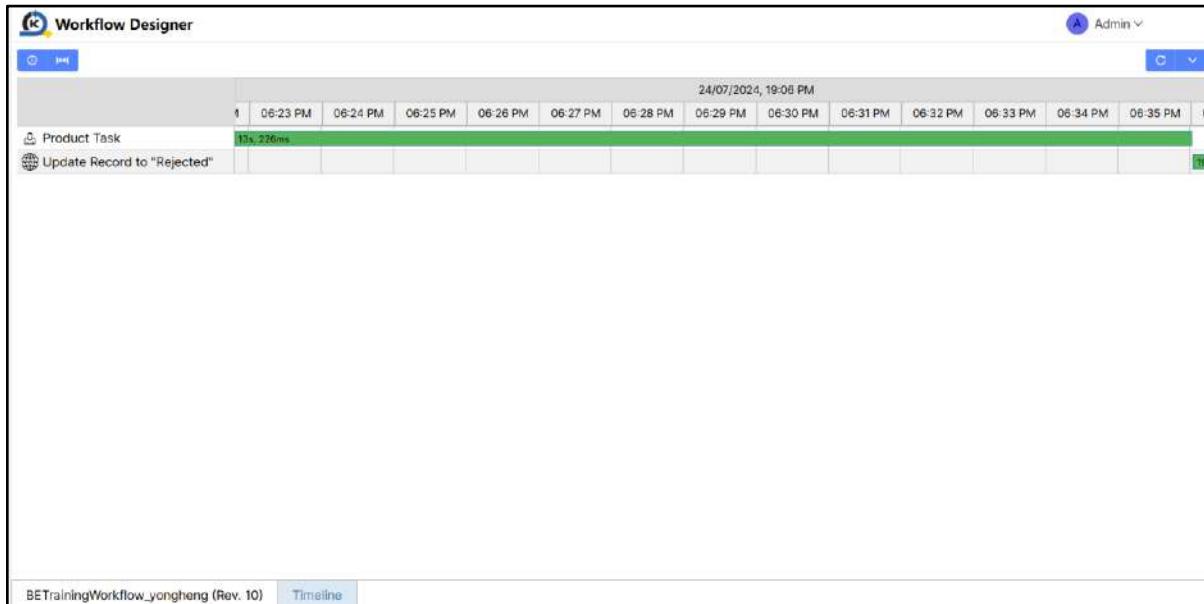
Workflow Execution Timeline View

The timeline feature allows users to look into the time execution of each workflow task. This is useful in observing any bottlenecks and deep dive into individual tasks for debugging in the event of error.

- We can also click on the ‘Timeline’ tab

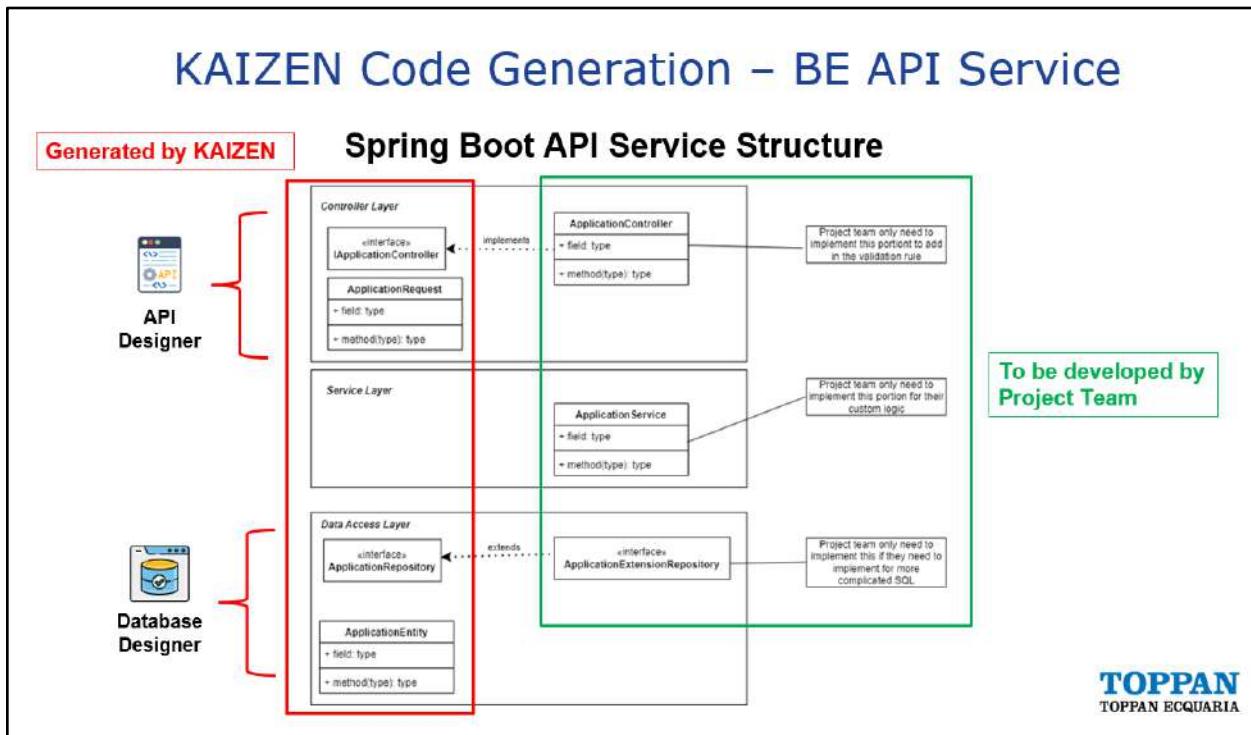


- We can see the time execution of the task



Tutorial 22: Code Generation (Backend)

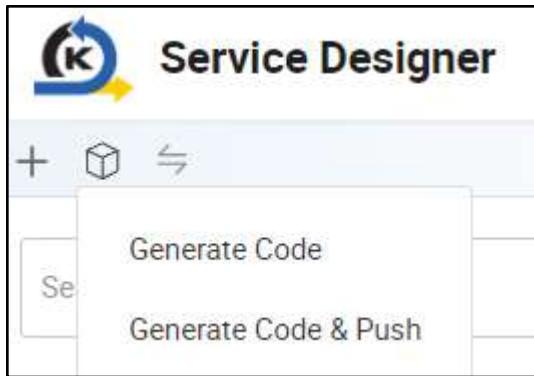
Generating backend code in KAIZEN simplifies the creation of robust server-side applications by converting user-defined configurations into structured Java code, including Value Objects (VOs) and application properties. This automated process enhances productivity by eliminating the need for manual coding, reducing the risk of errors, and ensuring adherence to best practices. Developers can easily integrate business logic, manage data access through DAOs, and configure application settings, all while maintaining a consistent architecture. By leveraging code generation tools, teams can accelerate backend development, streamline workflows, and focus on delivering high-quality applications efficiently.



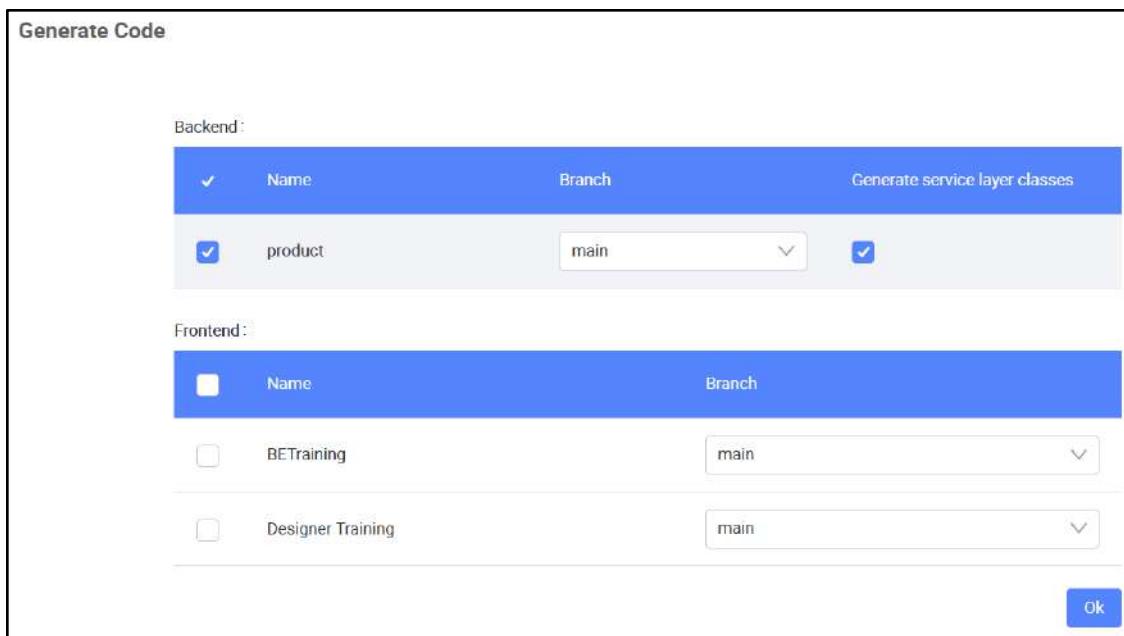
Practical 22.2: Generating and Running Backend Code

Generate Code

- In Service Designer, select **Generate Code**



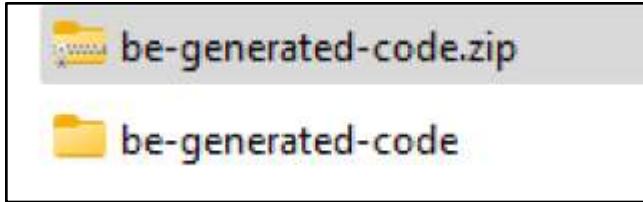
- Ensure **Backend: Product** is selected together with the **Generate service layer classes** option, and click **OK**



A zip file is downloaded



- Unzip the file



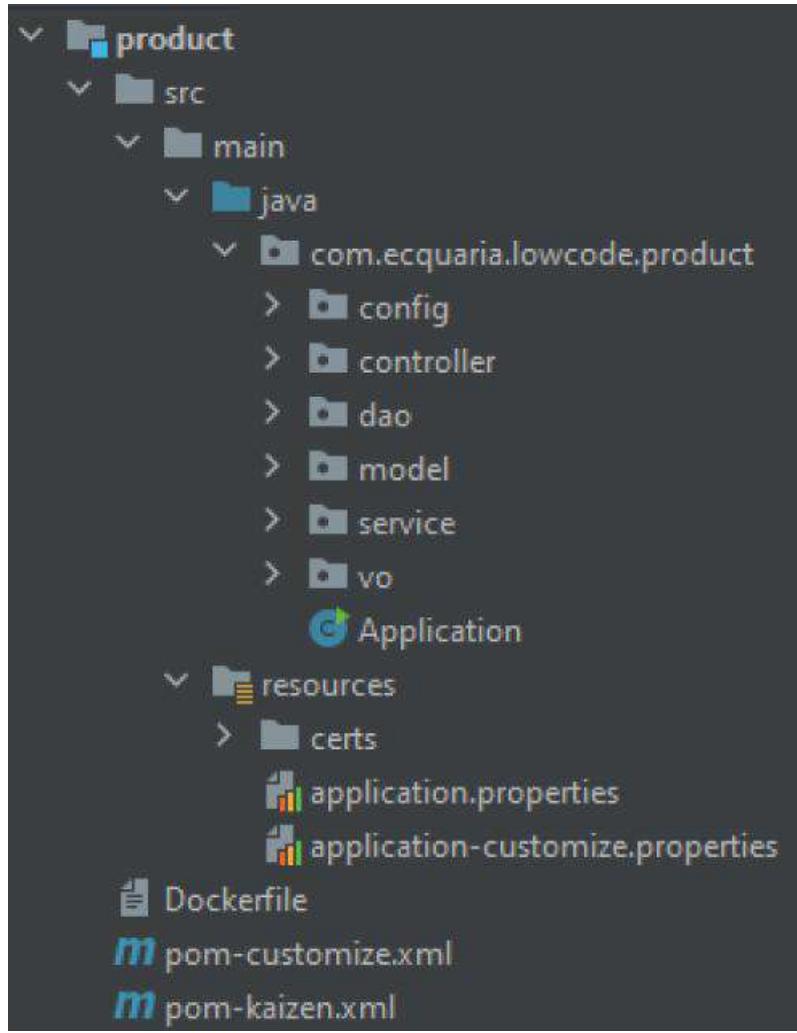
Understanding Generated Backend Code Structure

In KAIZEN, the backend code is often auto-generated to streamline development. However, understanding the structure of this generated code is crucial for customization, debugging, and integrating new features. The backend typically includes components like controllers, services, repositories, and models, each serving a specific role in handling API requests, business logic, and database interactions. By familiarizing yourself with this structure, you can make targeted modifications, optimize performance, and ensure the backend aligns with your application's requirements.

Directory Structure

The generated backend code in low-code platforms is organized into a clear directory structure to manage different functionalities efficiently. Common folders include:

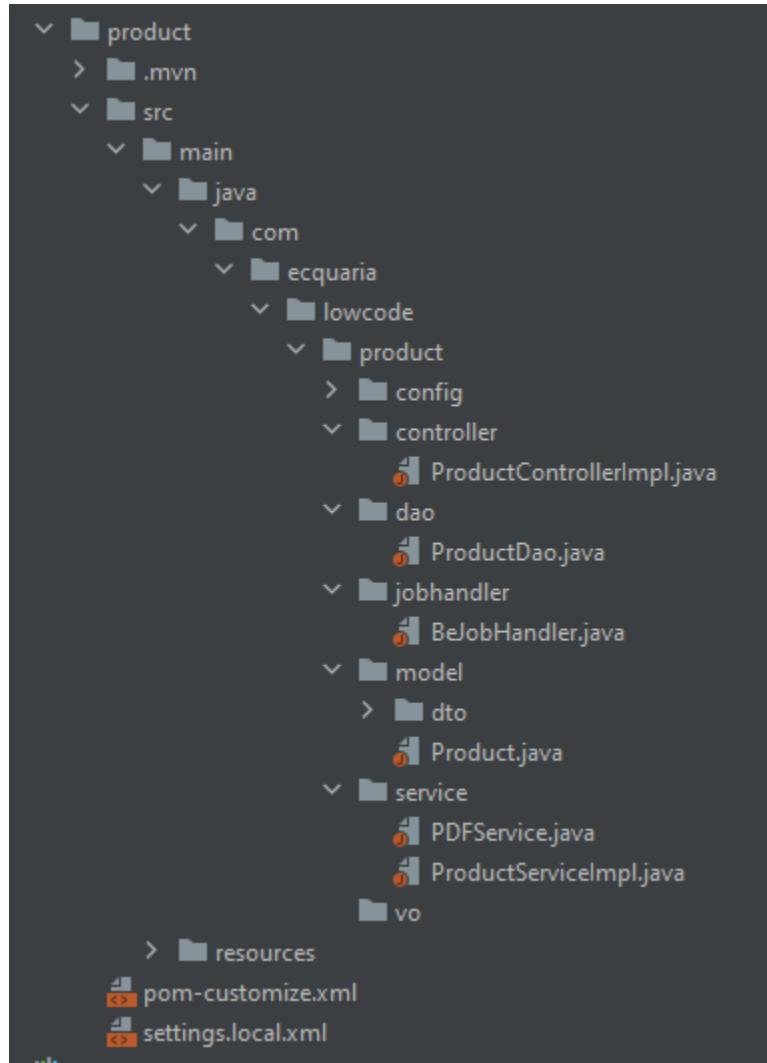
- **Controllers:** These handle incoming API requests and map interactions between the frontend and backend. Controllers can be customized to change specific API behaviors based on your application's needs.
- **Dao (Data access object):** These are responsible for managing database interactions, using auto-generated methods to access and manipulate data. Repositories can be customized to add complex queries or optimize performance.
- **Models and Entities:** Models represent your application's data and map directly to database tables. You can modify them to include additional fields or relationships, aligning with evolving business requirements.
- **Services:** Services manage the core business logic and interact with controllers. They encapsulate reusable logic and can be extended or modified to introduce additional functionality.
- **Configuration:** The configuration files in the generated backend code are essential for managing environment-specific settings like security, database connections, and API credentials



<u>application.properties</u>	Default application configuration settings.
<u>application-customize.properties</u>	A separate properties file for custom configurations without modifying defaults.
<u>pom-kaizen.xml</u>	The primary Maven configuration file for the server module.
<u>pom-customize.xml</u>	A customizable Maven file to override dependencies or configurations.

- Download the zip file below containing the custom code and environment configurations to run your product service
 - [product-custom.zip](#)

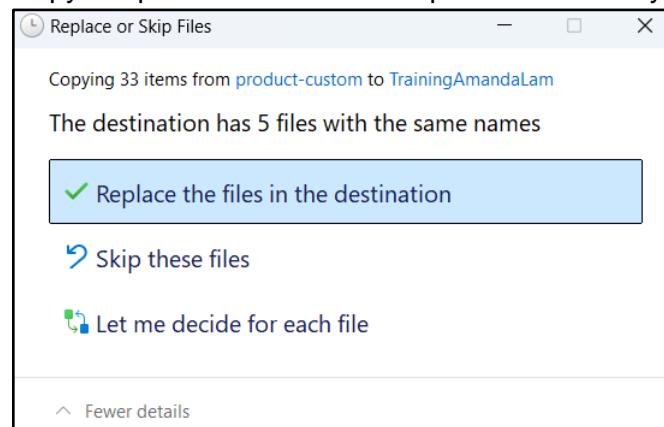
- This source code has the core business logic implemented which interacts with interfaces generated by KAIZEN.



- **Unzip and Copy** the necessary code over.

- For Windows:

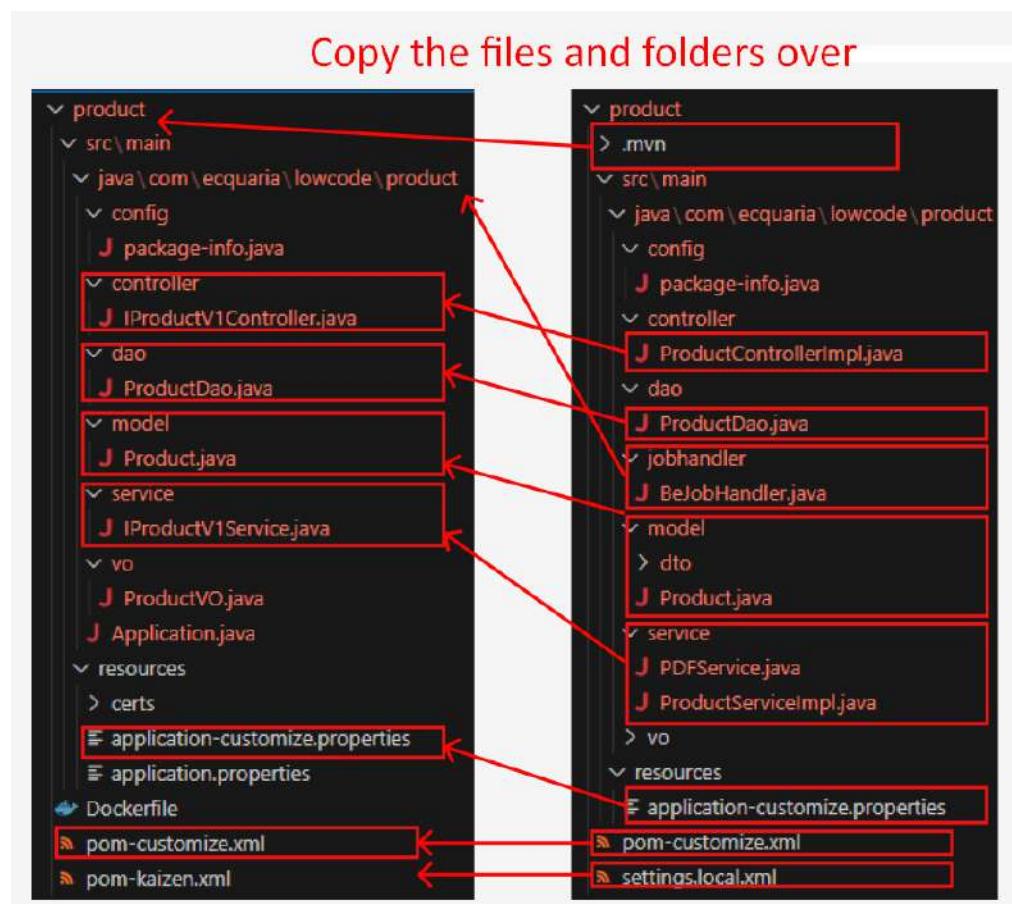
Copy the product folder into the product folder of your generated code.



- For macOS:

.mvn will be hidden, use command+shift+dot to show hidden files

```
$ rsync -av /source_folder/ /destination_folder/
```



- The result will look something like below



- Open your now updated generated code “Training<username>” in your IDE
- Update the properties configuration in your **application-customize.properties** file:
 - (`(TrainingAmandaLam\product\src\main\resources)`) to your own schema name highlighted in red below

`Spring.datasource.url: <username>_schema
(Example: amandalam_schema)`

- Add the config property “`spring.jpa.properties.hibernate.default_schema`” with the value as your username. See the area highlighted green below

`spring.jpa.properties.hibernate.default_schema: <username>_schema`

(Example: spring.jpa.properties.hibernate.default_schema: amandalam_schema)

```

src/main
  java/com/ecquaria/lowcode/product
    config
    controllers
      J ProductV1Controller.java
      J ProductControllerImpl.java
    dao
      J ProductBaseDjEntity.java
      J ProductEntity.java
    model
      J Entity.java
      J BaseProduct.java
      J Product.java
    service
      J Application.java
    resources
      application.properties
      application-customize.properties
    target
    Dockerfile

```

```

1 server.port=8092
2
3 spring.datasource.url=jdbc:postgresql://nlb-kalzen-dexez-001-001061d7c5f50d3e.elb.ap-southeast-1.amazonaws.com:5432/trg-single?currentSchema=amandalam_schema
4 spring.datasource.username="iam"
5 spring.datasource.password="trg!nning2025"
6 service.name.mesh="iam"
7 use.mesh.name=true
8 server.ssl.enabled=false
9 system.user.admin.account.id=systemsecurityuser
10
11 spring.jpa.properties.hibernate.default_schema=amandalam_schema
12
13 spring.cache.redis
14 spring.data.redis.host=nlb-kalzen-dexez-001-001061d7c5f50d3e.elb.ap-southeast-1.amazonaws.com
15 spring.data.redis.port=6379
16 spring.cache.type=redis
17 spring.data.redis.lettuce.pool.max-active=8
18 spring.data.redis.lettuce.pool.max-idle=8
19 spring.data.redis.lettuce.pool.min-idle=0
20
21 service.name.job-job
22 ### job config
23 job.executor.logpath=/usr/local/tomcat/logs/agg-job-log
24 job.executor.logretentiondays=-1
25 job.executor.upname=product
26 job.executor.ip=
27 job.schedule.upname=product

```

- Open your terminal in the product directory (Training<username>/product)
- **mvn clean package -f pom-customize.xml**

```

PS C:\Users\ECQ1031\Downloads\product\product> mvn clean package -f pom-customize.xml
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.ecquaria.lowcode.product:product >-----
[INFO] Building product 24.10.1-SNAPSHOT
[INFO]   from pom-customize.xml
[INFO] ----- [ jar ] -----

```

- **java -jar target/product.jar**

```

PS C:\Users\ECQ1031\Downloads\product\product> java -jar target/product.jar
SLF4J(W): Class path contains multiple SLF4J providers.
SLF4J(W): Found provider [ch.qos.logback.classic.spi.LogbackServiceProvider@107a0d6]
SLF4J(W): Found provider [org.slf4j.nop.NOPServiceProvider@1a9e7c8]
SLF4J(W): See https://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J(T): Actual provider is of type [ch.qos.logback.classic.spi.LogbackServiceProvider@107a0d6]
=====
=====
Welcome to Agile Gov Platform
=====
2025-02-21T10:36:50.161+08:00 INFO 20484 --- [product] [main] c.ecquaria.lowcode.product.Application : Starting Application v24.10.1-SNAPSHOT using Java 17.0.12 with PID 20484 (C:\Users\ECQ1031\Downloads\product\product\target\product.jar started by ECQ1031 in C:\Users\ECQ1031\Downloads\product\product)
2025-02-21T10:36:50.166+08:00 INFO 20484 --- [product] [main] c.ecquaria.lowcode.product.Application : The following 1 profile is active: "customized"
2025-02-21T10:36:50.380+08:00 WARN 20484 --- [product] [main] o.s.c.a.AnnotationBeanNameGenerator : Support for convention-based stereotype names is deprecated in a future version of the framework. Please annotate the 'value' attribute in @com.ecquaria.lowcode.job.handler.annotation.JobHandler with @AliasFor(annotation=Component.class) to declare an explicit alias for @Component's 'value' attribute.
2025-02-21T10:36:51.747+08:00 INFO 20484 --- [product] [main] s.d.r.c.RepositoryConfigurationDelegate : Multiple Spring Data modules found, entering strict repository configuration mode
2025-02-21T10:36:51.760+08:00 INFO 20484 --- [product] [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-02-21T10:36:51.947+08:00 INFO 20484 --- [product] [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 183 ms. Found 1 JPA repository interface.
2025-02-21T10:36:51.956+08:00 INFO 20484 --- [product] [main] s.d.r.c.RepositoryConfigurationDelegate : Multiple Spring Data modules found, entering strict repository configuration mode

```

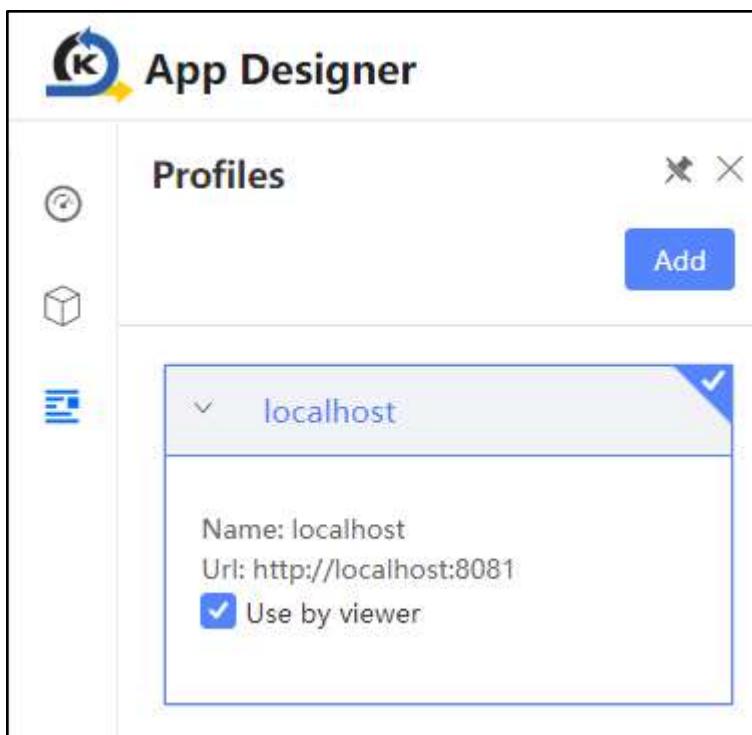

Tutorial 23: Profile for local development

This tutorial covers the following Learning Objectives:

- Learn how to configure profiles for local development within your application.
- Understand how to redirect traffic to different resources based on the environment.
- Improve local development workflows by integrating API calls with KAIZEN seamlessly.

In this tutorial, you will learn about configuring profiles that allow you to redirect traffic to different resources, enhancing local development. This feature enables smooth development of APIs in a local environment while still interacting with KAIZEN resources, improving the overall development experience.

Essentially, you can use profile to configure and connect to different environments, allowing you to easily access the APIs there from your frontend pages in the app designer. All these can be done without having to build and run your entire frontend application locally. Note that you can only connect to 1 profile at a time.



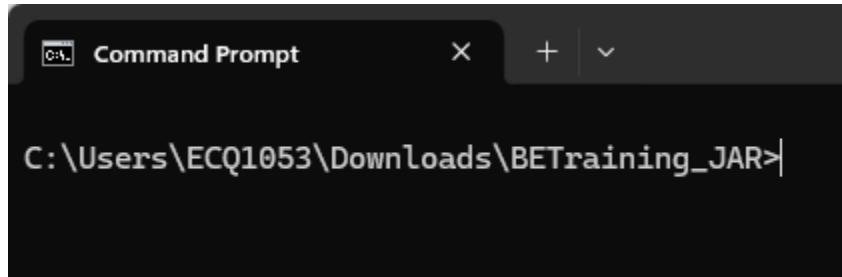
Practical 23.1: Running be-services

For local development, the application requires running multiple services to simulate the production environment effectively. Developers need to run four jar files locally, each representing a critical component of the application. These jar files work together to provide the necessary backend functionality for testing, debugging, and developing features.

- **Download** the following based on your Operating System:
 - If you are using Windows:
 - [BETraining_JAR\(CloudDaaSenv\)-Windows.zip](#)
- **Unzip** the zip file. There should be a script file included (either a .sh for MacOS or Linux or .bat for Windows)

 start.bat	30/7/2025 2:30 pm	Windows Batch File	1 KB
 be-iam-proxy	14/7/2025 6:11 pm	File folder	
 be-gateway	14/7/2025 6:11 pm	File folder	

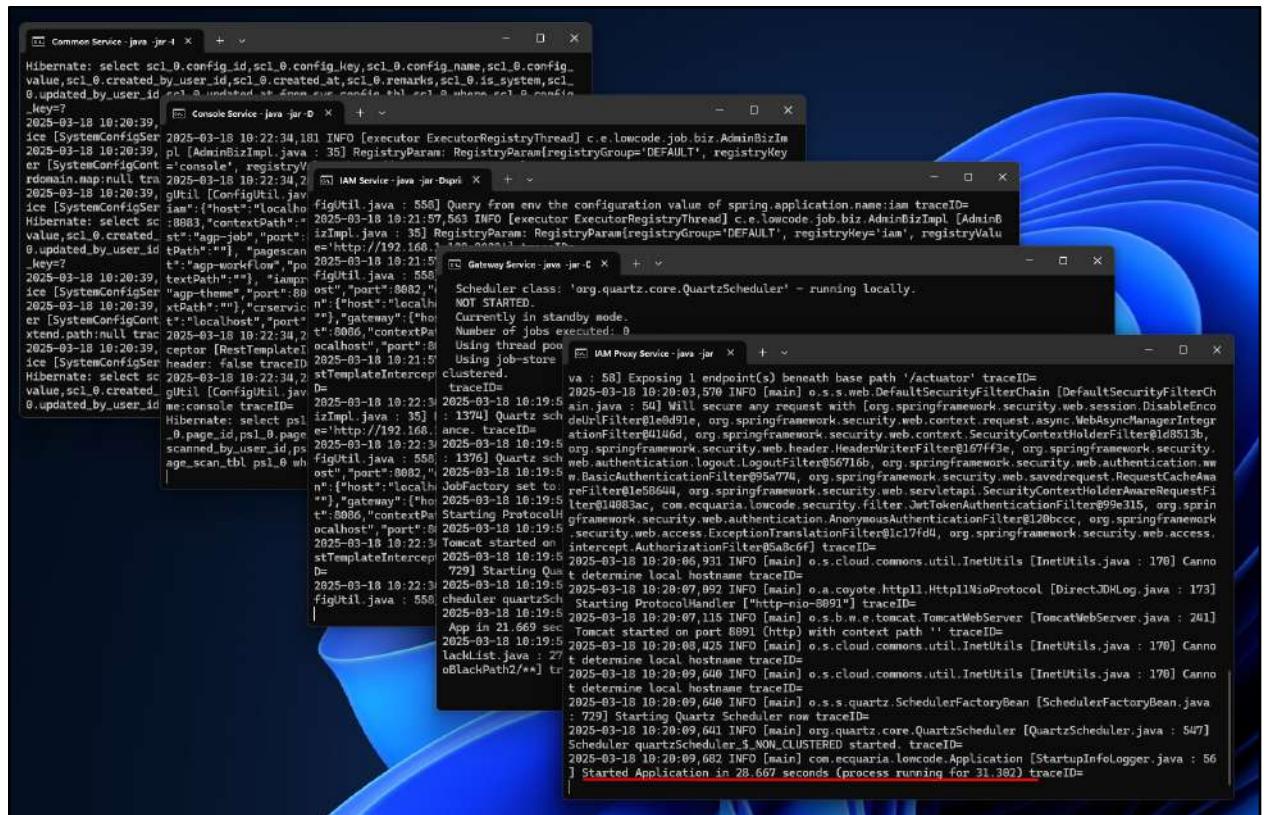
- Open command prompt in the unzipped folder



- Run the jar files through the script provided:
 - For Windows:
 - Run the script using ./start.bat

```
PS C:\Users\ECQ913\Downloads\BETraining_JAR> ./start.bat
```

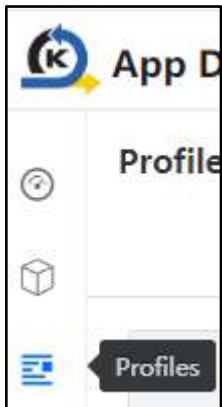
- The script should open 5 terminal windows with the following KAIZEN microservices: IAM, Common, Console, Gateway, IAM Proxy:



Ensure that the applications run with no exceptions/stack traces. Alternatively, you can look out for the default Spring Boot Application started message (ie. Started Application in xx.xxx seconds (process running for xx.xxx))

Practical 23.2: Update Profile feature

- Click “Profiles” plugin



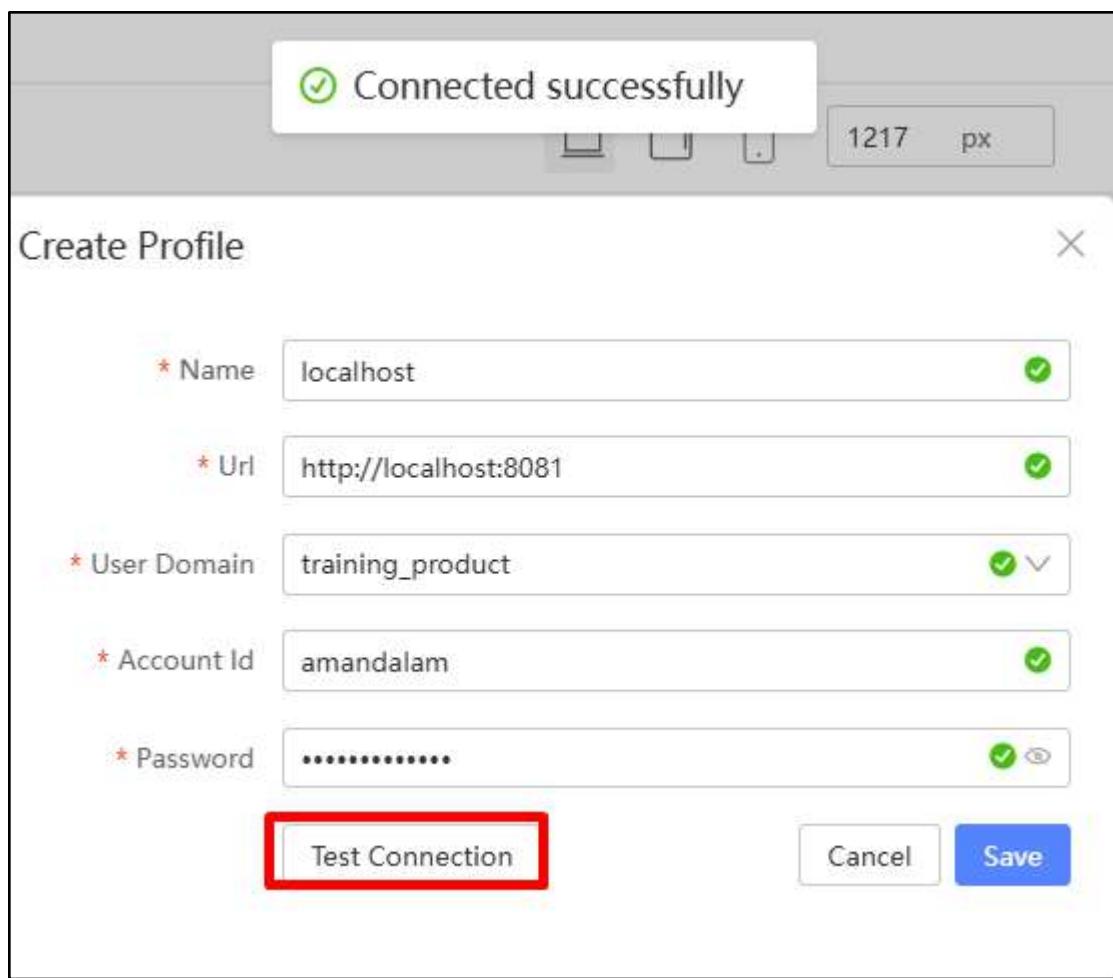
- Add a new “localhost” profile

A screenshot of a 'Create Profile' dialog box. The title bar says 'Create Profile' and has a close button 'X'. The form contains five fields:

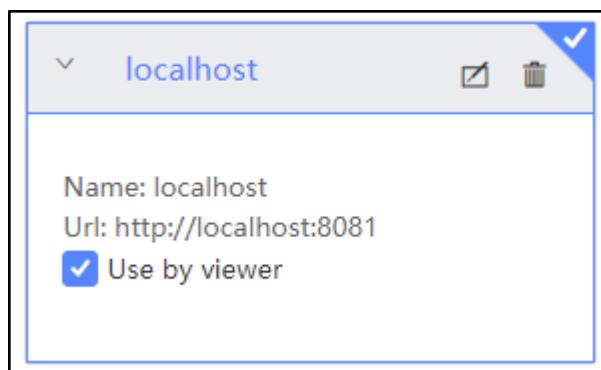
- * Name: 'localhost'
- * Url: 'http://localhost:8081'
- * User Domain: 'training_product'
- * Account Id: 'amandalam' (with a red note: 'update this to your username')
- * Password: '*****' (with an eye icon)

At the bottom are three buttons: 'Test Connection', 'Cancel', and 'Save'.

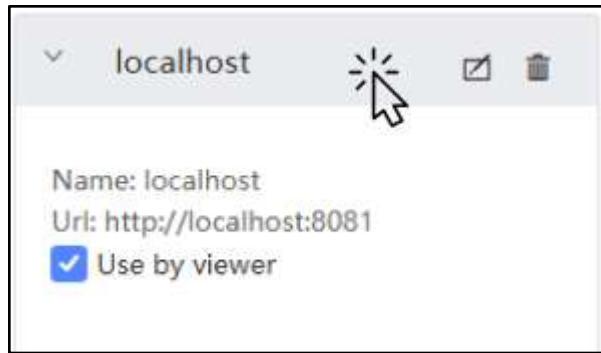
- Click on **Test Connection** to test connectivity



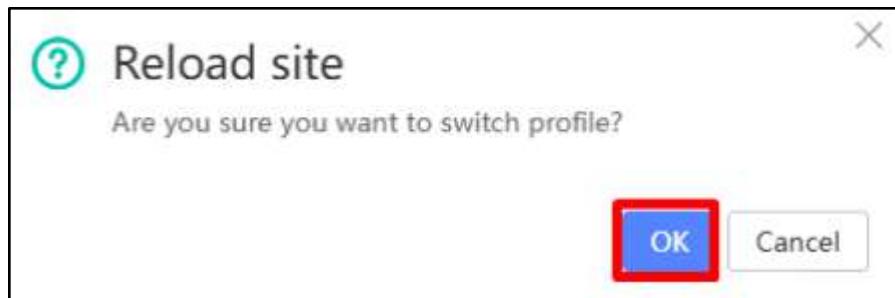
- Select “Use by viewer”. Selecting this feature will ensure that the application’s users with external or internal viewer roles will use this profile when previewing the application.



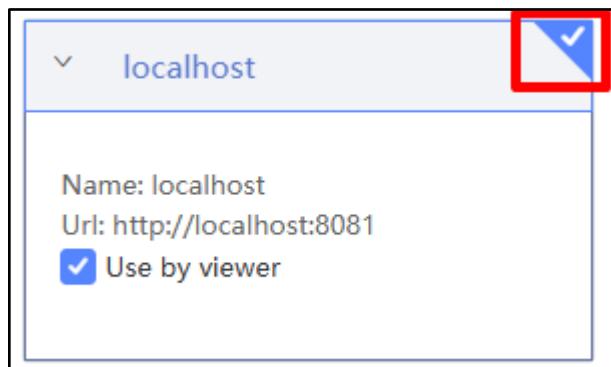
- **Click** to switch the profile



- Click OK to reload the site



- Ensure a **Tick sign** appears on the **localhost** profile



- Refresh the page, note that API calls are made using your configured profile “localhost”, which allows users to do local development.

	Headers	Payload	Preview	Response	Initiator	Timing
▼ General						
Request URL:				http://localhost:8081/gateway/product/api/v1/betraining/products?pageNo=0&name=&pageSize=10&sortBy=id&location=&category=&sortDir=asc		
Request Method:				GET		
Status Code:				200 OK		
Remote Address:				127.0.0.1:8081		
Referrer Policy:				strict-origin-when-cross-origin		

Product Management System								
Search Criteria		Product Data Table						
ID	Product Name	Category	Brand	Color	Price	Location	Status	Action
1	Product A	Clothing	Brand Y	Black	29.99	Singapore	Approved	<button>Download PDF</button>
2	Product B	Clothing	Brand Y	Blue	29.99	Singapore	Approved	<button>Download PDF</button>
3	Product C	Electronics	Brand Z	Silver	899.99	Singapore	Approved	<button>Download PDF</button>
4	Product D	Clothing	Brand X	Blue	49.99	Singapore	Approved	<button>Download PDF</button>
5	Product E	Electronics	Brand Y	Black	99.99	Singapore	Approved	<button>Download PDF</button>
6	Product F	Home	Brand Z	Brown	599.99	Singapore	Approved	<button>Download PDF</button>
7	Product G	Clothing	Brand Y	Red	39.99	Singapore	Approved	<button>Download PDF</button>
8	Product H	Electronics	Brand X	White	399.99	Singapore	Approved	<button>Download PDF</button>
9	Product I	Clothing	Brand Z	Gray	99.99	Singapore	Approved	<button>Download PDF</button>

Notice that all these data loaded on your application's 'Page1 Table' page are taking the data you have added in your schema earlier using the Database Designer Tutorial.

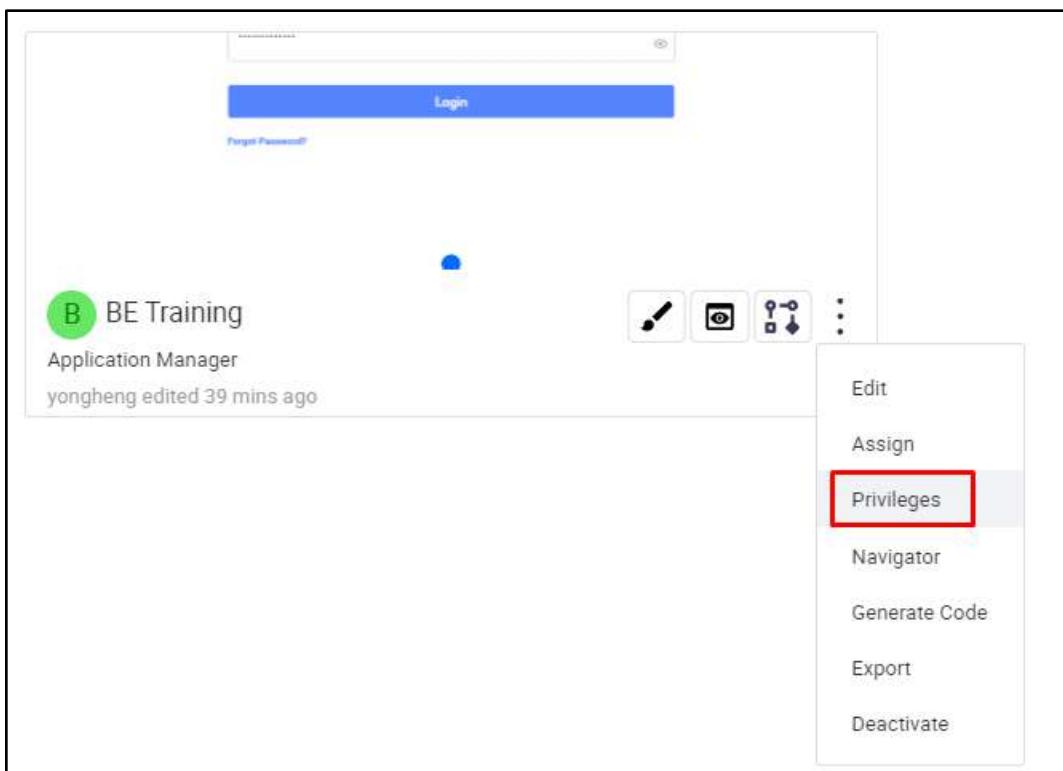
Tutorial 24: Customize Privilege for Page Permission

This tutorial covers the following Learning Objectives:

- Understand the Importance of Page Permissions in securing application pages and controlling user access.
- Learn to Define Custom Page Privileges and configure custom privileges for specific application pages to grant access.

Now that you are able to get your application successfully running on your local profile, you can easily develop and test the integration between your backend code logic and your frontend application screens. However, the concern with your current application is that any user will be able to access all pages in your application. In this tutorial, you'll explore how you can configure your pages with custom privileges to control its access privileges. We will go into more detail about all the other Identity and Access Management (IAM) features in the last tutorial.

- Click on setting and select **Privileges**



- Click **Create** to open a popup

Config Privileges

No.	User Domain	Privilege Name
1	AGP Designer	Anonymous Resourc

Items per page: 10

Create **Export** **Export By UserDomain**

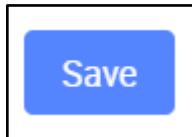
- Enter the following value:
 - **User Domain:** training_product
 - **Privilege Name:** betraining_<username> (example: betraining_yongheng)
 - **Privilege Code:** be001
 - **Subject Type:** betraining

Create Privilege

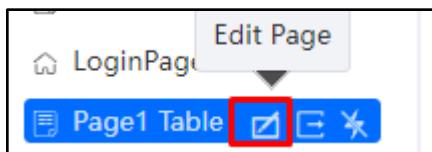
X

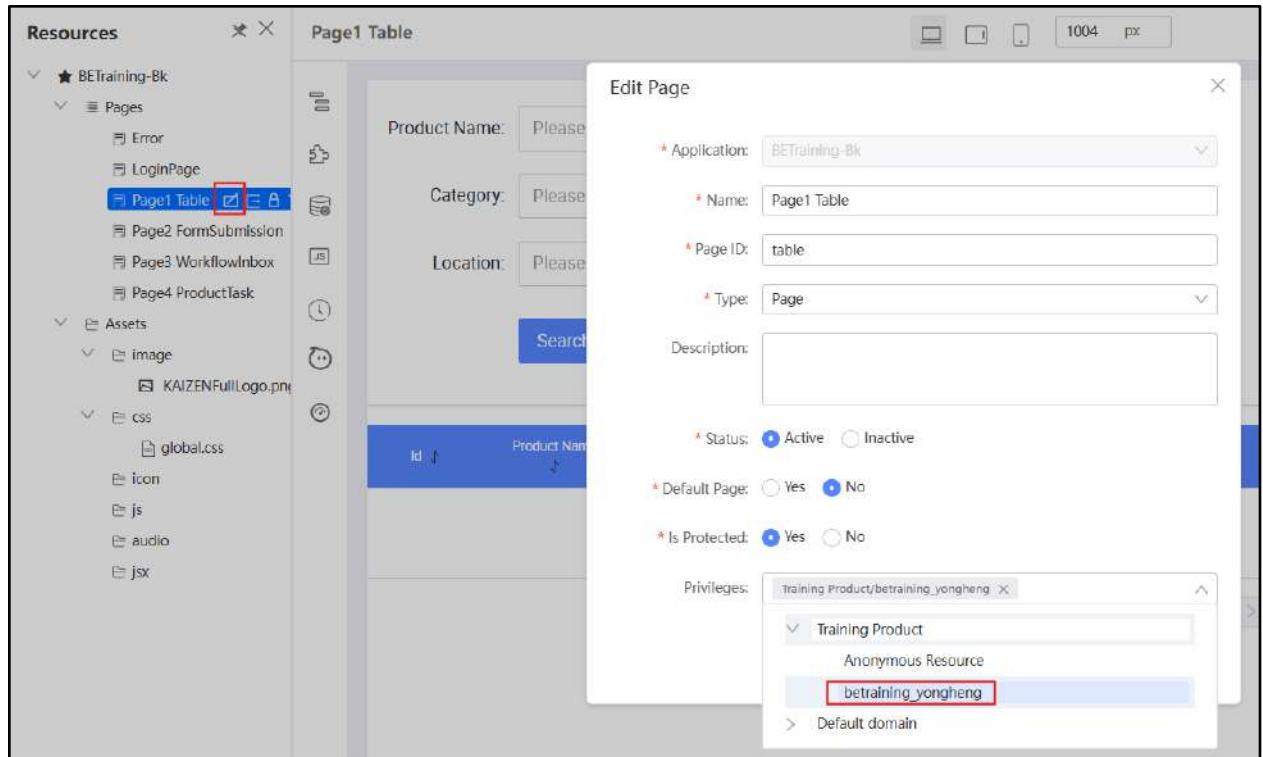
* User Domain	training_product
* Privilege Name	betraining_yongheng
* Privilege Code	be001
Subject Type	betraining
* Status	<input checked="" type="radio"/> Active <input type="radio"/> Inactive
Remarks	
<input type="button" value="Cancel"/> <input type="button" value="Save"/>	

- Click **Save**



- Edit each page to add in the necessary privileges





Page	Privileges
Error	Anonymous Resource
Login	Anonymous Resource
Table	betraining_<username>
FormSubmission	betraining_<username>
WorkflowInbox	betraining_<username>
ProductTask	betraining_<username>

The above configuration of privileges for the respective pages mean that only users with roles that have the corresponding privilege would be able to access the respective page in the application. This step will ensure that your local application would be able to run smoothly after the next tutorial on code generation for backend.

More details on the other IAM features will be covered in the last tutorial.

Tutorial 25: Code Generation (Frontend)

This tutorial covers the following Learning Objectives:

- Understand the process of code generation for both frontend and backend components using KAIZEN.
- Gain insights into the structure and organization of the generated code to facilitate easier navigation and modifications.
- Learn best practices for integrating and maintaining generated code within your development workflow to ensure consistency and scalability.

In this tutorial, you'll explore the code generation feature of KAIZEN, which allows you to automatically create frontend and backend code components tailored to your application. This feature streamlines the development process by eliminating repetitive coding tasks, enabling you to focus on business logic and functionality. You will also learn how to customize the generated code to align with your project's architecture, ensuring a seamless integration with your existing codebase and improving overall maintainability.

Practical 25.1: Generating and Running Frontend Code

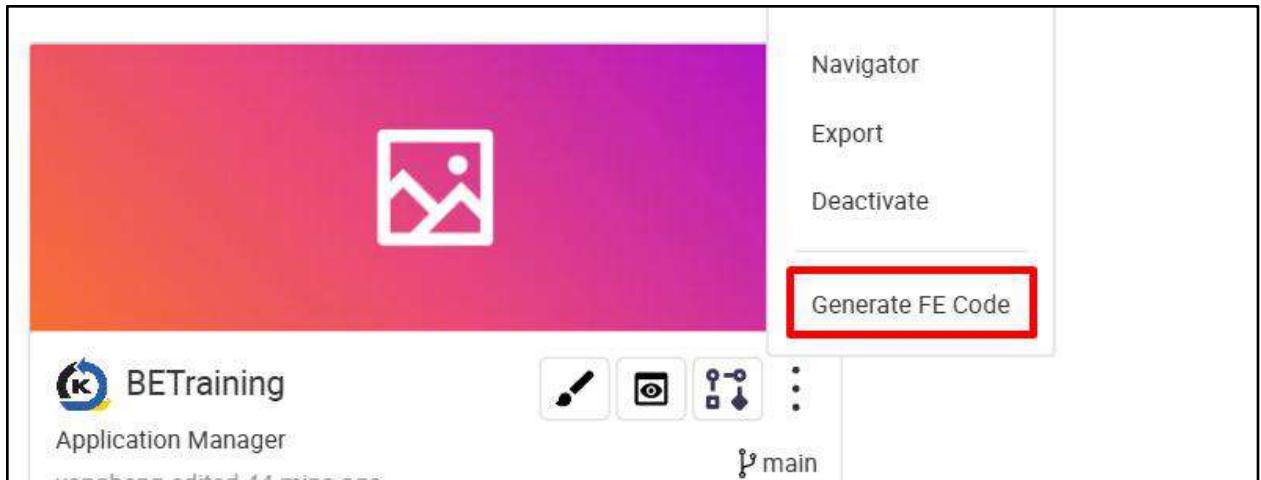
Generating frontend code in KAIZEN streamlines the creation of interactive user interfaces by transforming user-defined JSON Schemas into functional React code. This automated process accelerates development, ensures consistency, and minimizes errors, allowing developers to focus on business logic and user experience rather than repetitive coding tasks. By using visual design tools and pre-defined templates, teams can quickly prototype, iterate, and deploy applications, making KAIZEN an effective solution for efficient software delivery.

Installation of Necessary Prerequisites:

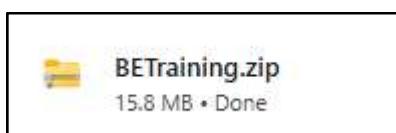
- Install Node.js (18.20.4) and Yarn to manage frontend dependencies.
- Set up Nginx

Generate Code

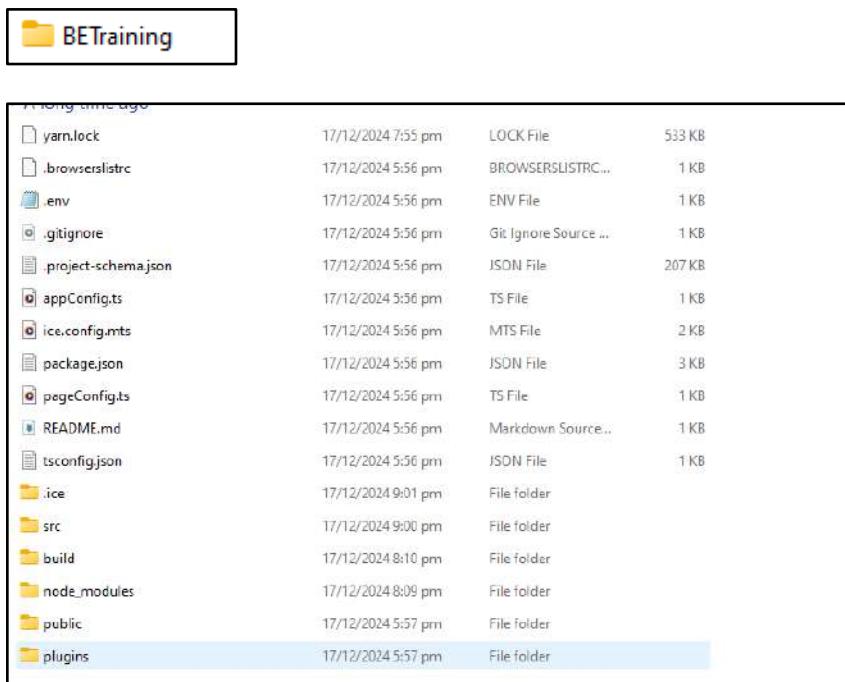
- Click setting and select **Generate FE Code**



A zip file is downloaded



- Unzip the file



- Open with your code editor or using command prompt
 - Visual Studio Code
 - Command Prompt

```
C:\Users\ECQ1053\Downloads\BETraining>code .
```

The screenshot shows a code editor with two panes. The left pane displays the directory structure of the BETraining project:

- plugins
- public
- src
- .browserslistrc
- .env
- .gitignore
- .project-schema.json
- appConfig.ts
- ice.config.mts
- iconConfig.ts
- package.json** (highlighted in blue)
- pageConfig.ts
- README.md
- tsconfig.json

The right pane shows the content of the package.json file:

```
1  {
2   "name": "BETraining",
3   "version": "0.1.5",
4   "description": "ecq template",
5   "dependencies": {
6     "@alifd/next": "1.26.4",
7     "@grafana/faro-web-tracing": "1.11.0",
8     "@tecq/lowcode-plugin-inject": "24.10.1",
9     "@tecq/lowcode-react-renderer": "24.10.1",
10    "@ant-design/pro-components": "2.8.1",
11    "@ice/runtime": "1.4.13",
12    "@ice/plugin-icespark": "1.1.1",
13    "@ice/stark-app": "1.5.0",
14    "@iceworks/spec": "1.6.0",
15    "antd": "5.21.5",
16    "file-saver": "2.0.5",
17    "file-type": "19.6.0",
18    "jwk-to-pem": "2.0.6",
19    "moment": "2.30.1",
20    "node-rsa": "1.1.1",
21    "prop-types": "15.8.1",
22    "react": "18.3.1",
23    "react-dom": "18.3.1",
24    "styled-components": "6.1.13",
25    "dateformat": "5.0.3",
```

- Ensure you're using node version of 18.20.4
 - node -v

```
>node -v
v18.20.4
```

- Change yarn registry path as some custom dependencies are hosted internally in Toppan network nexus
 - **Install yarn (Skip this step if you've already installed yarn)**
 - npm install -g yarn
 - **yarn config set registry http://alb-kaizen-daasez-001-673358367.ap-southeast-1.elb.amazonaws.com:8081/repository/npm-group**

```
PS C:\Users\ECQ1031\Downloads\BETraining-Bk4\BETraining-Bk> yarn config set registry http://alb-kaizen-daasez-001-673358367.ap-southeast-1.elb.amazonaws.com:8081/repository/npm-group
yarn config v1.22.22
```

Windows (Run as administrator):

```
java -jar be-common/tecq-be-common-24.10.1.jar --  
spring.config.location=file:be-common/application.properties
```

macOS:

```
sudo java -jar be-common/tecq-be-common-24.10.1.jar --  
spring.config.location=file:be-common/application.properties
```

```
C:\Users\ECQ1053\Downloads\BETraining_JAR>java -jar be-common\tecq-be-common-2  
4.10.1.jar --spring.config.location=file:be-common\application.properties
```

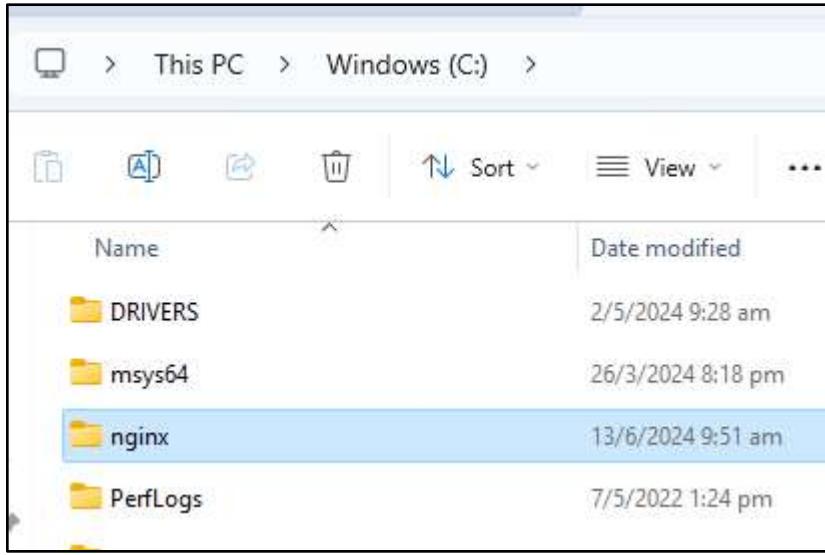
- Install the dependencies
 - **yarn install**

```
>yarn install  
yarn install v1.22.22  
warning ...\\package.json: No license field  
[1/5] Validating package.json...  
[2/5] Resolving packages...  
warning Resolution field "rollup@2.79.2" is incompatible with requested version "rollup@^0.25.8"  
success Already up-to-date.
```

- Run the application
 - **yarn run start**

```
>yarn run start  
yarn run v1.22.22  
warning ...\\package.json: No license field  
$ ice start --port 4000
```

Running nginx



- Edit the **nginx/nginx.conf** value in the nginx folder to your username
 - **betraining_trainneename** (Example: `betraining_amandalam`)

```
server {
    listen 80;
    server_name localhost;

    #charset koi8-r;

    #access_log logs/host.access.log main;

    location /betraining_trainneename/ {
        proxy_pass http://127.0.0.1:4000/;
        #root html;
        #index index.html index.htm;
    }

    location /gateway {
        proxy_pass http://localhost:8081;
        #root html;
        #index index.html index.htm;
    }

    location /login {
        proxy_pass http://localhost:8081;
        #root html;
        #index index.html index.htm;
    }

    location /logout {
        proxy_pass http://localhost:8081;
    }
}
```

New nginx conf

```

server {
    listen      80;
    server_name localhost;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

    location / {
        proxy_pass  http://localhost:3333/;
    }

    location /betraining_traineeename/ {
        #proxy_pass http://127.0.0.1:4000/;
        #root   html;
        #index  index.html index.htm;
        alias C:/Users/ECQ1031/Downloads/InterPRO 2.0 BE_gen6/InterPRO2.0BE/build;
        #index  index.html;
        if ($request_uri ~* \.(js|css|jpg|jpeg|png|gif|ico|svg|woff|woff2|eot|ttf|otf|json|webp)$) {
            proxy_pass http://localhost:4000;
            break;
        }

        rewrite ^/betraining_traineeename(/.*)$ / break;
        proxy_pass  http://localhost:4000/;
    }
}

```

For macOS:

Copy paste [nginx.conf](#) file inside /opt/homebrew/etc/nginx folder

- Double click the nginx.exe to start the service. Some helpful command for debugging:
 - start nginx
 - nginx.exe -s stop
 - tasklist /FI "IMAGENAME eq nginx.exe"

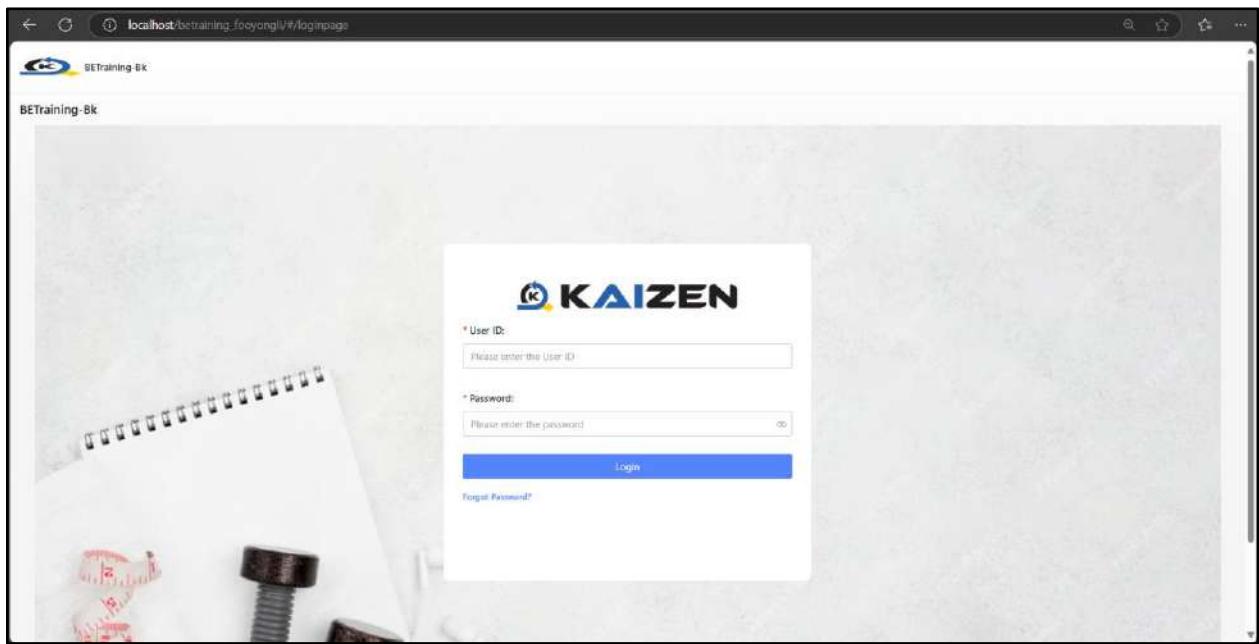
For macOS:

- sudo nginx
- sudo nginx -s stop

File Explorer			
This PC > Windows (C:) > nginx >			
		Sort	View
Name	Date modified	Type	Size
conf	13/6/2024 10:14 pm	File folder	
contrib	13/6/2024 9:51 am	File folder	
docs	13/6/2024 9:51 am	File folder	
html	13/6/2024 9:51 am	File folder	
logs	13/6/2024 8:47 pm	File folder	
temp	13/6/2024 9:54 am	File folder	
nginx.exe	13/6/2024 9:50 am	Application	4,606

Running the generated code

- Navigate to http://localhost/betraining_traineeName/#/loginpage, you should be able to see your login screen.
(Example: http://localhost/betraining_amandalam/#/loginpage)



After login to your account, you will be redirected and see the table listing

Table

The screenshot shows a search interface with three dropdown menus: 'Product Name' (Please enter), 'Category' (Please select), and 'Location' (Please select). Below the search bar are two buttons: 'Search' and 'Clear'. A table follows, with columns: Id, Product Name, Category, Brand, Color, Price, Location, Status, and Action. The data rows are:

Id	Product Name	Category	Brand	Color	Price	Location	Status	Action
1	Product A	Electronics	Brand X	Black	499.99	Singapore	Approved	<button>Download PDF</button>
2	Product B	Clothing	Brand Y	Blue	29.99	Malaysia	Approved	<button>Download PDF</button>
3	Product C	Electronics	Brand Z	Silver	899.99	Singapore	Approved	<button>Download PDF</button>
4	Product D	Clothing	Brand X	Blue	49.99	Singapore	Approved	<button>Download PDF</button>
5	Product E	Electronics	Brand Y	Black	99.99	Malaysia	Approved	<button>Download PDF</button>
6	Product F	Home	Brand Z	Brown	599.99	Singapore	Approved	<button>Download PDF</button>
7	Product G	Clothing	Brand Y	Red	39.99	Singapore	Approved	<button>Download PDF</button>

Triggering Error

- **Ctrl + C** and **stop** the running of product.jar

```
Starting the program. Press Ctrl + C to terminate.
Running... Press Ctrl + C to stop.
Running... Press Ctrl + C to stop.
Running... Press Ctrl + C to stop.
```

- **Click on Download.** As the API is no longer present, it will trigger a 500 error

The screenshot shows the same search interface and table as before. The 'Download' button for the first row (Product A) is highlighted with a red box. The data rows are identical to the previous table:

Id	Product Name	Category	Brand	Color	Price	Location	Status	Action
1	Product A	Clothing	Brand Z	Black	19.99	Singapore	Approved	<button>Download</button>
2	Product B	Clothing	Brand Y	Blue	29.99	Singapore	Approved	<button>Download</button>
3	Product C	Electronics	Brand Z	Silver	899.99	Singapore	Approved	<button>Download</button>
4	Product D	Clothing	Brand X	Blue	49.99	Singapore	Approved	<button>Download</button>
5	Product E	Electronics	Brand Y	Black	99.99	Singapore	Approved	<button>Download</button>
6	Product F	Home	Brand Z	Brown	599.99	Singapore	Approved	<button>Download</button>

- Notice your page is redirected to error page

localhost/betraining_traineeName/#/error

 BETrainingIntermed Table Form Submission Workflow Inbox

BETrainingIntermediate

Sorry, We couldn't process your request. Please proceed to [Default page](#)

Tutorial 26: Push and Merge Code to Git

This tutorial covers the following Learning Objectives:

- Managing branches and tags within KAIZEN.
- Learn to push changes to a Gitlab repository from KAIZEN for both frontend and backend code.

In earlier tutorials, we have learnt how to set up the connection to Git and switch to a new branch to work on various enhancements and test your application to run locally. Now, we'll learn how to automate the process of committing and pushing changes in KAIZEN, thus improving efficiency and collaboration. We will demonstrate how the integration with GitLab's CI/CD pipeline feature automates the testing and deployment, streamlining the release process directly from KAIZEN.

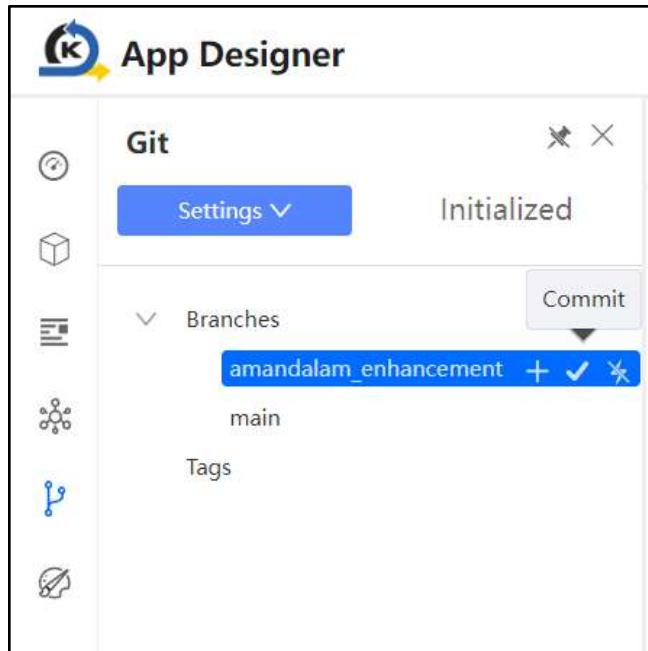
Practical 26.1: Frontend Git Management

In KAIZEN, frontend Git branch management is essential for maintaining an organized and efficient workflow. By creating separate branches for UI changes, such as theme updates or new features, developers can isolate modifications, avoid conflicts, and ensure smooth collaboration across teams. This approach supports parallel development, minimizes disruption to the main codebase, and allows for better testing and integration of frontend updates, ultimately enhancing project quality and stability.

Once you enter the design page, you will be able to see the theme design button and the published theme.

Create New Tag and Git Push

- On the App Designer, click on the commit icon of the branch you are on and proceed to commit changes, which will generate the FE code and push into Git.
- When the branch is ready to be released, as you push your code to Git, you may also tag your commit to mark a specific commit as a release version (e.g., v1.0, v2.1.3) and is immutable.



Commit Changes [?](#) [Generate Code And Push.](#) [X](#)

* Commit Message: bind APIs

* New Tag Yes No

* Tag Name: v1.0

[Cancel](#) [Commit](#)

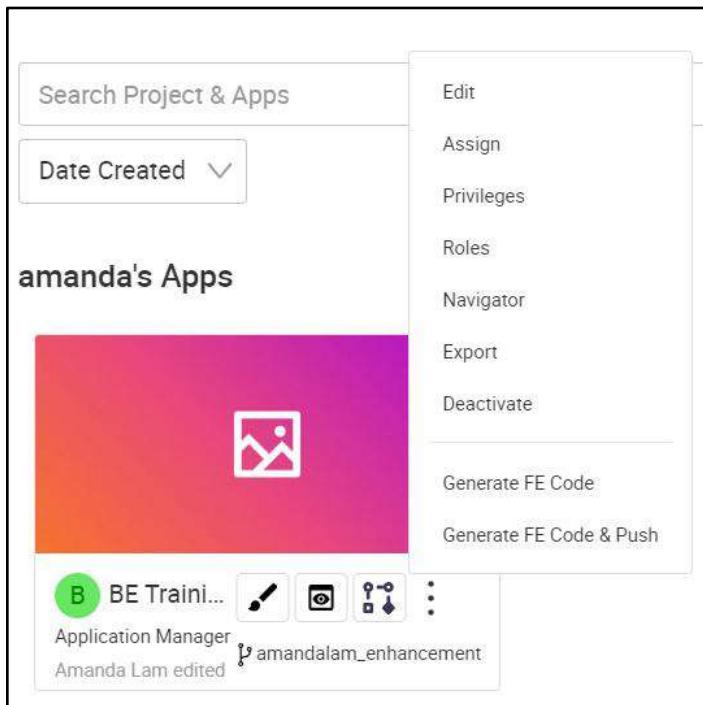
Commit and create tag successful!

- You should be able to see this code changes pushed onto the branch on Git as well.

KAIZEN-Training / kaizen-training-amandalam / fe-product / Repository

Name	Last commit	Last update
plugins	bind APIs	1 minute ago
public	bind APIs	1 minute ago
src	bind APIs	1 minute ago
.browserslistrc	bind APIs	1 minute ago
.env	bind APIs	1 minute ago
.gitignore	bind APIs	1 minute ago
.project-schema.json	bind APIs	1 minute ago
README.md	bind APIs	1 minute ago

- Another way to push FE Code onto the current branch on Git is from the project itself on the studio console by clicking **Generate FE Code & Push**.



Generate FE Code & Push

* Message:

New Tag

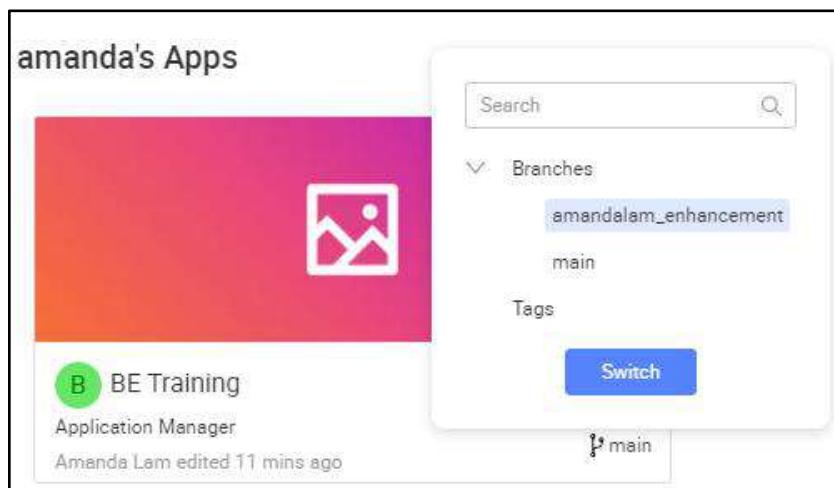
Yes No

* Tag Name

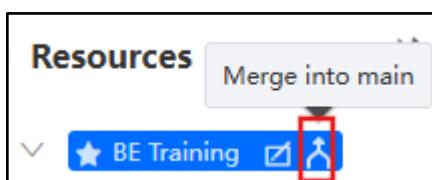
Commit

Merge Code to Main Branch + Push Main Branch

- Once you are done making relevant enhancements on your current enhancement branch, we will now proceed to merge your FE changes to the main branch.
- Switch back to the **main** branch by clicking the Git branch on your application from the studio console.



- In your app designer, click on the merge button beside the application and select your enhancement branch to merge into the main branch.





- Before you merge, notice that the new additions will be highlighted in green. You will need to specify what you want to merge in by clicking on the blue arrows at each individual addition or on top to merge in all new additions.

A screenshot of the 'Merge into main' dialog box. It shows a comparison between two branches: 'enhancement_amandalam' (left) and 'main' (right).

The 'enhancement_amandalam' branch contains the following files:

- assets file (03 Dec 2024 05:12 PM)
- pages
 - LoginPage (03 Dec 2024 05:12 PM)
 - Page2 FormSubmission (03 Dec 2024 05:12 PM)
 - Error (03 Dec 2024 05:12 PM)
 - Page1 Table (03 Dec 2024 05:12 PM)
 - Page3 WorkflowInbox (03 Dec 2024 05:12 PM)
 - Page4 ProductTask (03 Dec 2024 05:12 PM)
- assets

The 'main' branch contains the following files:

- assets file (03 Dec 2024 05:12 PM)
- pages
 - LoginPage (03 Dec 2024 05:12 PM)
 - Page2 FormSubmission (03 Dec 2024 05:12 PM)
 - Error (03 Dec 2024 05:12 PM)
 - Page1 Table (03 Dec 2024 05:12 PM)
 - Page3 WorkflowInbox (03 Dec 2024 05:12 PM)
 - Page4 ProductTask (03 Dec 2024 05:12 PM)
- assets

Below the code comparison, there is a 'Comments:' text input field and a row of buttons at the bottom: 'Cancel', 'Back', and 'Merge' (which is highlighted).

- In this case, the red colour coded indication indicates that there are changes to existing lines in the code. You may click into the file icon of each of the pages to view the

changes.

View Differences

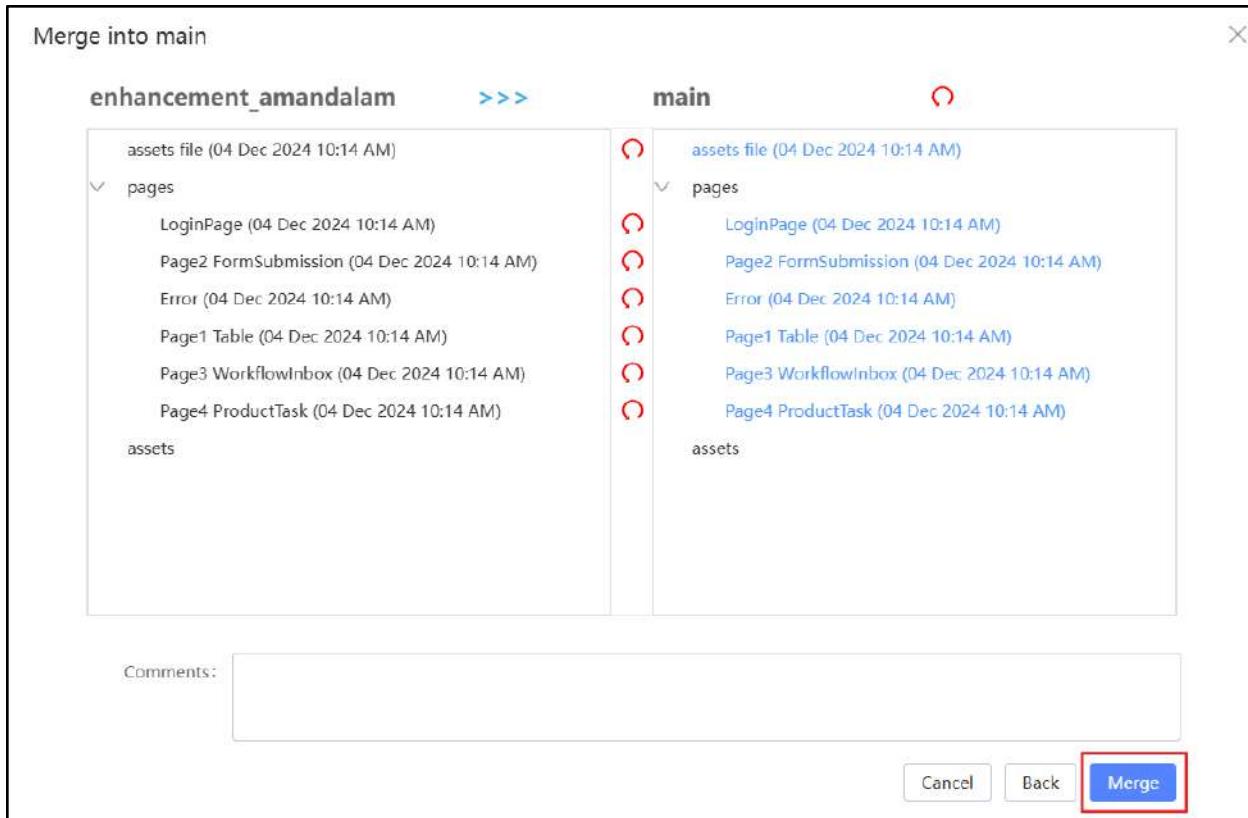
assets file(enhancement_amandalam)

```
17 ]
18 },
19 {
20   "title": "fusion components",
21   "package": "@alifd/next",
22   "version": "1.26.4",
23   "urls": [
24     "/editor/css/1.26.4/next.min.css",
25     "/editor/script/next-with-locales.min.js"
26   ],
27   "library": "Next"
28 },
29 {
30   "package": "@tecq/lowcode-materials",
31   "version": "24.4.5-ali-orange-24.4.5-XX",
32   "library": "TecqLowcodeMaterials",
33   "isMaterial": true,
34   "urls": [
35     "/unpack/@tecq/lowcode-materials@24.4.5-ali-orange-24.4.5-XX/build/lowcode
36     "/unpack/@tecq/lowcode-materials@24.4.5-ali-orange-24.4.5-XX/build/lowcode
37   ],
38   "editUrls": [
39     "/unpack/@tecq/lowcode-materials@24.4.5-ali-orange-24.4.5-XX/build/lowcode
40     "/unpack/@tecq/lowcode-materials@24.4.5-ali-orange-24.4.5-XX/build/lowcode
41   ],
42   "advancedUrls": {
43     "default": [
44       "/unpack/@tecq/lowcode-materials@24.4.5-ali-orange-24.4.5-XX/build/lowco
45       "/unpack/@tecq/lowcode-materials@24.4.5-ali-orange-24.4.5-XX/build/lowco
46     ],
47   },
48   "advancedEditUrls": []
49 },
50 {
51   "package": "@tecq/layout",
52   "version": "24.4.5",
```

assets file(main)

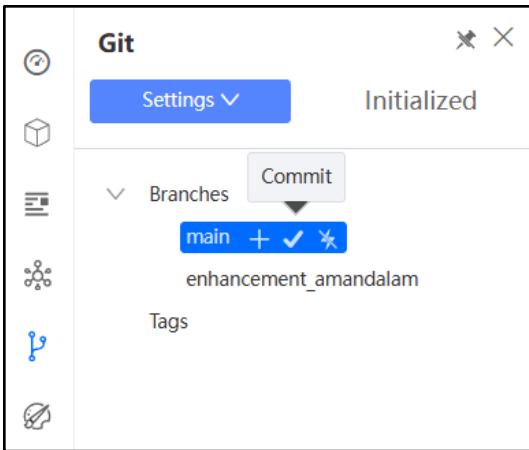
```
17 ]
18 },
19 {
20   "title": "fusion components",
21   "package": "@alifd/next",
22   "version": "1.26.4",
23   "urls": [
24     "/editor/css/1.26.4/next.min.css",
25     "/editor/script/next-with-locales.min.js"
26   ],
27   "library": "Next"
28 },
29 {
30   "package": "@tecq/lowcode-materials",
31   "version": "24.4.5",
32   "library": "TecqLowcodeMaterials",
33   "isMaterial": true,
34   "urls": [
35     "/unpack/@tecq/lowcode-materials@24.4.5/build/lowcode/render/default/view.
36     "/unpack/@tecq/lowcode-materials@24.4.5/build/lowcode/render/default/view.
37   ],
38   "editUrls": [
39     "/unpack/@tecq/lowcode-materials@24.4.5/build/lowcode/view.js",
40     "/unpack/@tecq/lowcode-materials@24.4.5/build/lowcode/view.css"
41   ],
42   "advancedUrls": {
43     "default": [
44       "/unpack/@tecq/lowcode-materials@24.4.5/build/lowcode/render/default/vic
45       "/unpack/@tecq/lowcode-materials@24.4.5/build/lowcode/render/default/vic
46     ],
47   },
48   "advancedEditUrls": []
49 },
50 {
51   "package": "@tecq/layout",
52   "version": "24.4.5",
```

- Click on the blue arrow on top to merge in all new changes made by the theme change. After which, click on the 'merge' button.



Merge Successfully

- You may now verify that your enhancements have been successfully merged and reflected on your application screens on the main branch.
- Note that KAIZEN's merge feature is not a replica of merging on Git. Refer to the [additional information](#) below for more information on KAIZEN's Git features.
- Next, we will proceed to push these changes on the main branch onto Git. As done in the earlier step, click on the commit icon of the branch you are on and proceed to commit changes, which will generate the FE code and push into Git.

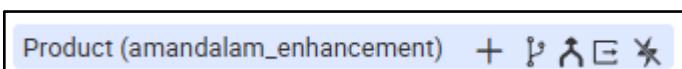


- Once again, you may verify these code changes pushed onto the main branch on Git as well.

Practical 26.2: Backend Git Management (Optional)

In KAIZEN, using Git branches for backend enhancements, such as adding a new API, is critical for maintaining clean, structured development workflows. By creating a separate branch for each enhancement, developers can work independently on new features without affecting the stability of the main application. This approach allows for safe testing, minimizes conflicts, and enables smoother collaboration among team members. Once the changes are tested and validated, the new API enhancements can be easily merged back into the main codebase, ensuring a stable and organized development process.

Create New Tag and Git Push



- Earlier you have configured your respective service and controller in the Service Designer. Now we will proceed to generate the BE code and push these changes onto this new branch.



Generate Code & Push

Backend:

Name	Branch
<input checked="" type="checkbox"/> Product	amandalam_enhancement

Frontend:

Name	Branch
<input checked="" type="checkbox"/> BE Training	amandalam_enhancement

* Comments:

add product service and controller

Create Tag

* Tag Name:

v1.0

OK

 Backend code successfully generated.

- This will successfully push both the FE and BE code onto the specified branch on Git.
- Similar to FE code push, notice that we can also create a tag for the BE code that we want to push to allow us to manage releases for features built.

Merge Code to Main Branch + Push Main Branch

- Once you are done making relevant enhancements on your current enhancement branch, we will now proceed to merge your BE changes to the main branch as well. Switch back to the main branch.

Switch Branch

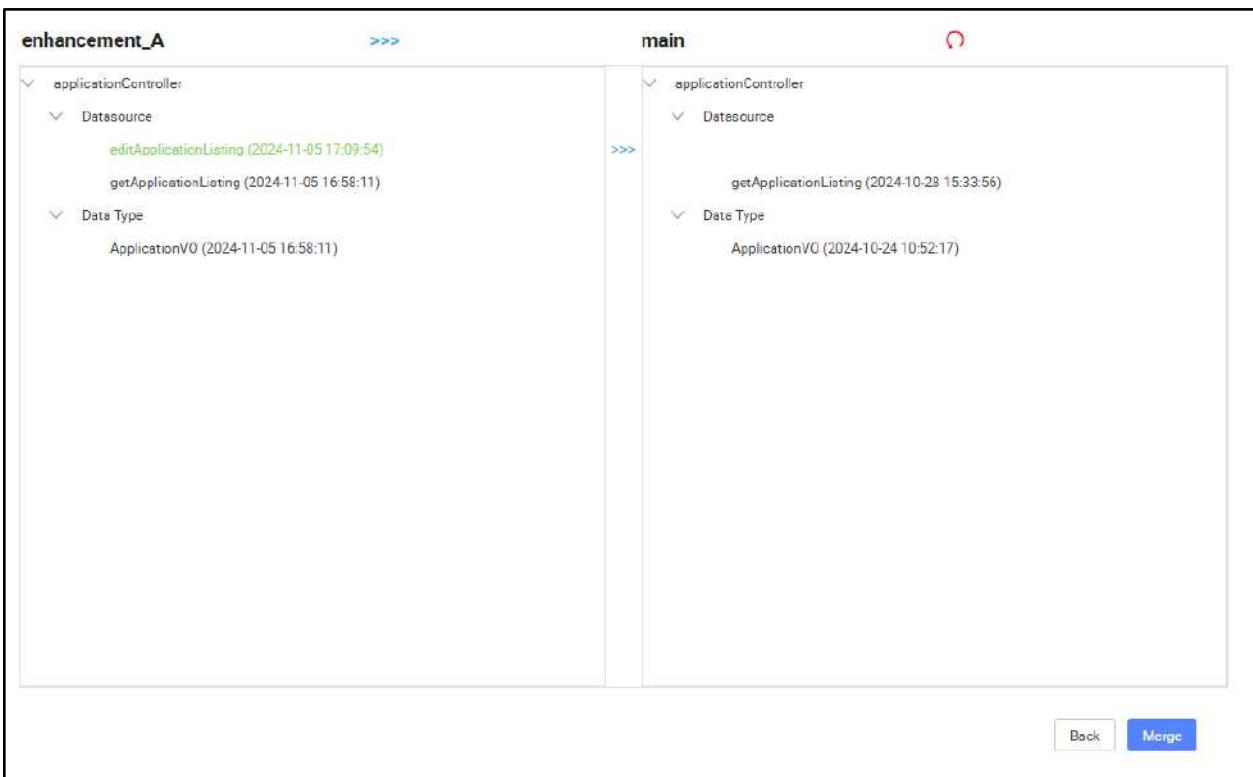
Name	Branch
Product	main

Apply

- Click on this merge button to merge into the main branch.



- Select the new branch you pushed your code to earlier to merge into the main branch.



- Before you merge, notice that the new additions will be highlighted in green. You will need to specify what you want to merge in by clicking on the blue arrows at each

individual addition or on top to merge in all new additions. After which, click on the ‘merge’ button.

 Merge Successfully

- Notice that your main branch on KAIZEN’s Service Designer now has the newly merged new datasource from the enhancement branch.
- Next, we will proceed to push these changes on the main branch onto Git. As done in the earlier step, click on the commit icon of the branch you are on and proceed to commit changes, which will generate the BE code and push into Git.



Generate Code & Push

Backend:

Name	Branch
Product	main

Frontend:

Name	Branch
BE Training	main
Demo Training	Please Select
Demo app - amandalam	Please Select
BE Training Intermediate	main
Test	Please Select

* Comments:
push main after merge

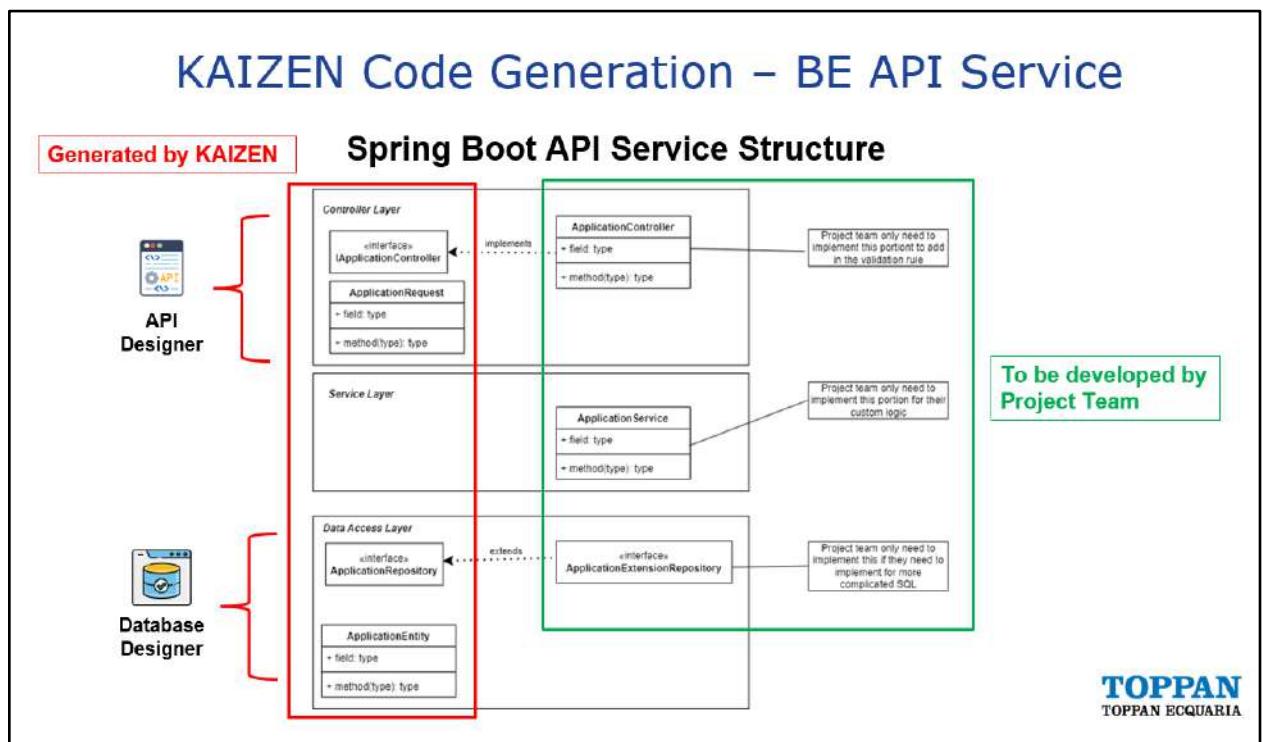
Create Tag

Ok

- You may verify these code changes pushed onto the main branch on Git as well.

Additional Information - Branch Management

- **KAIZEN Merge Feature** : For both frontend or backend development, KAIZEN's merge feature is not a replica of merging on git. It solely handles the merging for the interfaces and code generated by KAIZEN (labelled in red). As for the custom logic in the service layer, it will be developed by the project team (labelled in green). Merging of code for the custom logic by the project should still be done through Git. KAIZEN **only** supports the pushing of code onto Git.



- **Branch Switching tied to individual users:** Switching of branches on KAIZEN is a personal setting and is tied to individual users. Hence, if another project member switches branches, it will not be reflected on your application.
- **Separate FE and BE Branch Management:** Backend branch switching operates independently from frontend branch management.
- **No Automatic Switching:** Switching to a new branch on the frontend does not automatically switch the backend branch.
- **Resolve merge conflicts**
 - When you work in a team, you may come across a situation when somebody pushes changes to a file you are currently working on. If these changes do not overlap (that is, changes were made to different lines of code), the conflicting

files can be accepted and merged straight away. However, if the same lines were affected, one side cannot randomly be picked over the other, and you will need to resolve the conflict.

- Take the following scenario:
 - Dev 1 creates branch A from main
 - Dev 2 creates branch B from main
 - Dev 1 merges Branch A into main
 - Dev 2 now wants to merge Branch B into main. There is no pull function from main and we want to avoid changes being overwritten.
- In this scenario, you may continue to merge Branch B into main and perform merge conflict resolution by accepting the additions or changes made in branch B.

For more information on the behaviour performing various actions in your application design on KAIZEN, refer to the Appendix below for User Actions in the **App Designer** (Frontend Development) and **Service Designer** (Backend Development).

Appendix

App Designer (Application Level)

User Action	KAIZEN Operation	Git Operation	Remarks
Create a New App	New App is Created with a “main” branch by default	-	
Create New Page	New Page Definition is Created under the App	-	
Modify Page & Publish Page	Page Definition is Updated under the App A new Revision is Created under the current branch’s Page’s Revision History	-	Revision History is tied to KAIZEN branch
Connect to Git	Git Connection Details & Status is Saved	Git connection is Authenticated	Git connection Details & Status saved to both App Designer and Service Designer
Initialize to Git	2 Git Repo Names are Saved	2 Git Repo are Created - one for definitions, one for generated FE code	
Generate FE Code & Download	Generate FE Code FE Code Downloaded	-	

Generate FE Code & Push without New Tag	Generate FE Code	Git Commit & Git Push Definitions to Definition Git repo Git Commit & Git Push Generated FE (Reactjs) Code to code repo	
Generate FE Code & Push with New Tag	Generate FE Code New Tag is Created	Git Commit & Git Push Definitions to Definition Git repo New Tag is created with the Commit ID Git Commit & Git Push Generated FE (Reactjs) Code to code repo New Tag is created with the Commit ID	
Create New Branch	New Branch is Created based on latest revision on KAIZEN All Pages will have the 1st Revision created under the new branch	New Branch is Created in both Git Repo	Branch creation is based on the latest revision on KAIZEN and not based on the base branch pushed onto Git.
Switch to Target Branch	All Pages will be shown based on the Target Branch's revision	-	Switching of Branch will only take effect on the user account who performed this action. It will not affect another user's view in the App Designer if they want to view/work on a different branch.
Merge from Source Branch to Target Branch	Changes are merged to target branch	-	After merging to the target branch, if the user wants to commit to the 2 git repo, the user should verify in KAIZEN and then perform a commit at the target branch.

Service Designer (Application Level)

User Action	KAIZEN Operation	Git Operation	Remarks
Connect to Git	Git Connection Details & Status is Saved	Git connection is Authenticated	Git connection Details & Status saved to both App Designer and Service Designer
Create a New Service	New Service is Created with a "main" branch by default	-	
Initialize to Git	Git Repo Name is Saved	Git Repo created for Microservice	
Generate Code & Download	Generate BE Code BE Code Downloaded	-	
Generate Code & Push without New Tag	Generate BE Code	Git Commit & Git Push Definitions to Definition Git repo Git Commit & Git Push Generated BE Code to code repo	
Generate Code & Push with New Tag	Generate BE Code New Tag is Created	Git Commit & Git Push Definitions to Definition Git repo New Tag is created with the Commit ID Git Commit & Git Push Generated BE Code to code repo New Tag is created with the Commit ID	
Create New Branch	New Branch is Created All Pages will have the 1st Revision created under the new branch	New Branch is Created in both Git Repo	
Switch to Target Branch	All Pages will be shown based on the Target Branch's revision	-	Switching of Branch will only take effect on the user account who performed this action. It will not affect another user's view in the App Designer if they want to view/work on a different branch.

Merge from Source Branch to Target Branch	Changes are merged to target branch	-	After merging to the target branch, if the user wants to commit to the 2 git repo, the user should verify in KAIZEN and then perform a commit at the target branch.
---	-------------------------------------	---	---

Tutorial 27: Job Scheduler

This tutorial covers the following Learning Objectives:

- Understand how to set up and configure a job scheduler within your application.
- Learn to automate tasks at predefined intervals to improve efficiency.
- Manage task execution to ensure the smooth operation of automated processes.

In this tutorial, you will learn how to configure a job scheduler within your application. The job scheduler allows you to automate tasks, enabling predefined processes to run at specific times or intervals. This feature helps manage routine tasks more efficiently, reducing manual effort and ensuring consistent operations.

The screenshot shows the KAIZEN Studio interface with the 'Job Scheduler / Job' page selected. The left sidebar includes links for Home, Application Management (Access Controller, Job Scheduler), Executors, Job Log, Audit Trails, Notification Management, Workflow, Development Tools, and System Configuration. The main area has a search bar with fields for Executor (Please select), Job Name (Please enter), and Job Handler (Please enter). Below the search bar are 'Search' and 'Clear' buttons. A 'Create' button is located above a table. The table lists two jobs: 'BETraining.JobHandler' and 'Job Test'. The columns are No., Job Name, Executor, Job Handler, Cron, and Validity Start. The table footer shows an 'Items per page' dropdown set to 10. The overall interface is clean and modern, typical of a web-based administration tool.

No.	Job Name	Executor	Job Handler	Cron	Validity Start
1	BETraining.JobHandler	BETraining	BETraining.JobHandler	0 */* * * ?	
2	Job Test	Job	demoJobHandler	0 */* * * ?	

A JobHandler function is already prepared for the training. We will go through a demonstration of how a Job can be created and used. The Job Scheduler is only accessible to an admin role.

```

import com.ecquaria.lowcode.job.handler.IJobHandler;
import com.ecquaria.lowcode.job.handler.annotation.JobHandler;
import com.ecquaria.lowcode.job.log.JobLogger;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;

no usages
@JobHandler(value = "BETrainingJobHandler")
@Slf4j
public class BeJobHandler extends IJobHandler {

    2 usages
    private PDFService pdfService;

    no usages
    @Autowired
    public BeJobHandler(PDFService pdfService) { this.pdfService = pdfService; }

    @Override
    public R<String> execute(String param) throws Exception {
        JobLogger.log(appendLogPattern("JOB", "Hello World. BETraining"));
        log.info("JOB, Hello World. BETraining");

        try {
            pdfService.generateAndSaveApprovedProductsPDF();
            log.info("PDF generated and saved to the database successfully.");
            return SUCCESS;
        } catch (Exception e) {
            log.error("Error generating and saving PDF", e);
            return FAIL;
        }
    }
}

```

Practical 27.1: Creating and Running Job (Demonstration)

- The PDF generation of product listing has been configured to run via job scheduler. If we click the Download button, no file is downloaded because the job is not running.

ID	Product Name	Category	Brand	Color	Price	Location	Status	Action
1	Product A	Electronics	Brand X	Black	499.99	Singapore	Approved	<button>Download PDF</button>
2	Product B	Clothing	Brand Y	Blue	29.99	Malaysia	Approved	<button>Download PDF</button>
3	Product C	Electronics	Brand Z	Silver	899.99	Singapore	Approved	<button>Download PDF</button>
4	Product D	Clothing	Brand X	Blue	49.99	Singapore	Approved	<button>Download PDF</button>
5	Product E	Electronics	Brand Y	Black	99.99	Malaysia	Approved	<button>Download PDF</button>
6	Product F	Home	Brand Z	Brown	599.99	Singapore	Approved	<button>Download PDF</button>

- As an admin, you can create a job scheduler via the “Create” button.

The screenshot shows the 'Job Scheduler/ Job' interface. On the left, there's a sidebar with 'Access Controller', 'Job Scheduler' (selected), 'Executors', 'Job Log', 'Audit Trails', 'Notification Management', 'Workflow', 'System Configuration', and 'Report'. The main area has tabs for 'Job' and 'Edit Job'. In the 'Edit Job' tab, there's a form with fields: 'User Domain' (Training Product), 'Job Name' (Please enter), 'Job Handler' (Please enter), 'Executor' (Please select), 'Cron' (0 * /1 * * ?), 'Validity Start Date/Time' (Select date), 'Validity End Date/Time' (Select date), 'Job Handler' (BETrainingJobHandler), 'Route Strategy' (Round Robin), 'Block Strategy' (Serial Execution), 'Job Timeout Period' (0), and 'Job Description'. Buttons for 'Cancel' and 'Save' are at the bottom. A 'Schedule Forecast' section with a toggle switch is also visible.

<u>Field Name</u>	<u>Description</u>
Job Name	The unique identifier for the scheduled job.
Executor	The service responsible for executing the job.
Cron	The cron expression defines the job's schedule. The cron syntax is using Quartz scheduler library.
Validity Start Date/Time	The date and time when the job becomes active.
Validity End Date/Time	The date and time when the job becomes inactive.
Job Handler	The specific handler or method that executes the job logic.
Route Strategy	<p>The strategy used to route the job to an executor. Route Strategies supported:</p> <ol style="list-style-type: none"> Round Robin - Jobs are distributed sequentially to the executor across application instances. This strategy is ideal for high-availability (HA) setups with multiple instances. Sticky - A job is assigned to an executor on the first workflow instance, ensuring consistent handling. Broadcast - Jobs are executed by all executors across all application instances simultaneously. This strategy is suitable for tasks that require uniform application across all instances.
Block Strategy	The strategy to handle job execution conflicts. Block Strategies supported:

	<p>1. Serial - Waits for the current job to execute first.</p> <p>2. Discard Later - Discards incoming job and continues running the previous job in the event it has not finished running yet.</p> <p>If the job blocks and is unable to complete, that is where the Job Timeout Period comes into picture to terminate the job.</p>
Job Timeout Period	The maximum duration the job is allowed to run before timing out.
Job Description	A brief description of the job's purpose and functionality.
Schedule Forecast	Plan the timing and execution of your scheduled jobs

- Endpoint assignment is also required for JobHandler to execute.

- Run the Job function by clicking on the dropdown.

The screenshot shows the KAIZEN Admin Console interface. On the left, there's a sidebar with various menu items like Home, Access Controller, Job Scheduler, Audit Trails, etc. The 'Job' item under 'Job Scheduler' is selected and highlighted in blue. The main content area is titled 'Job Scheduler / Job' and shows a table of jobs. One job entry is visible:

No.	User Domain	Job Name	Executor	Job Handler	Open	Validity Start	Validity End	Description	Status	Actions
1	Training Product	BETraining.JobHandler	product	BETraining.JobHandler	0 * / 1 * * * ?				Success	<button>View</button> <button>Edit</button> <button>More</button> <button style="border: 2px solid red; padding: 2px;">Run</button> <button>Start</button>

Below the table, there's a dropdown for 'Items per page' set to 10. At the bottom right of the table, there are buttons for 'Run' and 'Start'.

Job "BETrainingJobHandler" run successfully

- From the “Job Log”, note that we can view the job execution success or failure.

The screenshot shows the KAIZEN Studio interface. The left sidebar has the same navigation as the Admin Console. The 'Job Log' item under 'Job Scheduler' is selected. The main content area is titled 'Job Scheduler / Job Log' and shows a table of job logs. One log entry is visible:

No.	Job Name	Executor	Trigger time	Job Handler	Handle Date	Trigger Result	Handle Result	Actions
1	BETraining.JobHandler	BETraining	2024-06-13 22:45:11.667	BETraining.JobHandler	2024-06-13 22:45:16.598	Success	Success	<button>View</button> <button>View Log</button>

Below the table, there's a dropdown for 'Items per page' set to 10. At the bottom right of the table, there is a small blue button with a white icon.

JobLogger function allows log to be written to the Job Scheduler log

```
□ JobLogger.log("JOB, Hello World. BETraining");
□
```

```

    @Override
    public R<String> execute(String param) throws Exception {
        JobLogger.log( appendLogPattern: "JOB, Hello World. BETraining");
        log.info("JOB, Hello World. BETraining");

        try {
            pdfService.generateAndSaveApprovedProductsPDF();
            log.info("PDF generated and saved to the database successfully.");
            return SUCCESS;
        } catch (Exception e) {
            log.error("Error generating and saving PDF", e);
            return FAIL;
        }
    }
}

```

Rolling Log

Job Execute Log

2024-06-13 14:45 [com.ecquaria.lowcode.job.thread.JobThread#run]-[123]-[Thread-7]
----- job execute start -----
----- Param: 2024-06-13 14:45 [com.ecquaria.lowcode.betraining.jobhandler.BeJobHandler#execute]-
[33]-[Thread-7] JOB, Hello World. BETraining 2024-06-13 14:45
[com.ecquaria.lowcode.job.thread.JobThread#run]-[159]-[Thread-7]
----- job execute end(finish) -----
----- ReturnT:com.ecquaria.lowcode.base.R@796a4c0b

Trigger Message

Job trigger type:Manual trigger
Trigger machine address:172.18.0.1
Execute address:[http://172.18.0.1:8092/betraining]
Executor route strategy:Round Robin
Block Strategy:Serial execution
Job timeout period:0

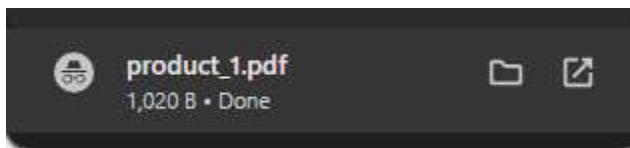
>>>>>>> Trigger Job <<<<<<<

Trigger Job:
address: http://172.18.0.1:8092/betraining
code: 200

- Back to our application, when we click the “Download” button, a PDF file is downloaded.

Table

Action	ID	Product Name	Category	Brand	Color	Price	Location	Status
<button>Download PDF</button>	1	Product A	Electronics	Brand X	Black	499.99	Singapore	Approved
<button>Download PDF</button>	2	Product B	Clothing	Brand Y	Blue	29.99	Malaysia	Approved
<button>Download PDF</button>	3	Product C	Electronics	Brand Z	Silver	899.99	Singapore	Approved
<button>Download PDF</button>	4	Product D	Clothing	Brand X	Blue	49.99	Singapore	Approved
<button>Download PDF</button>	5	Product E	Electronics	Brand Y	Black	99.99	Malaysia	Approved
<button>Download PDF</button>	6	Product F	Home	Brand Z	Brown	599.99	Singapore	Approved



C:/Users/ECQ1053/Downloads/product_1.pdf

1 / 1 | - 100% + | ↻

Product ID: 1
 Name: Product A
 Price: 499.99
 Category: Electronics
 Brand: Brand X
 Color: Black
 Location: Singapore
 Created At: 2024-05-12 00:00:00.0
 Updated At: 2024-05-12 00:00:00.0

Note:

- 1) For custom service to appear in the dropdown, it needs to be added to the "service.names.mesh" property in application.properties file.

Example:

application.properties file

```
□...
...
service.names.mesh: '{"iam":{"host":"kaizen-trg-be-iam-service.kaizen-
trg.svc.cluster.local","port":8082,"contextPath":"/iam"},"console":{"host":"kaizen-
trg-be-console-service.kaizen-
trg.svc.cluster.local","port":8083,"contextPath":"/console"},"common":{"host":"kaizen-
trg-be-common-service.kaizen-
trg.svc.cluster.local","port":8084,"contextPath":"/common"},"job":{"host":"kaizen-trg-
be-job-service.kaizen-
trg.svc.cluster.local","port":8085,"contextPath":"/job"},"gateway":{"host":"kaizen-
trg-be-gateway-service.kaizen-
trg.svc.cluster.local","port":8081,"contextPath":""},"pagescan":{"host":"kaizen-trg-
be-lighthouse-scanner-service.kaizen-
trg.svc.cluster.local","port":8086,"contextPath":""},"workflow":{"host":"kaizen-trg-
be-workflow-engine-service.kaizen-
trg.svc.cluster.local","port":8087,"contextPath":"/workflow"},"setup":{"host":"kaizen-
trg-be-setup-service.kaizen-
trg.svc.cluster.local","port":8089,"contextPath":"/setup"},"betraining":{"host":"kaize
n-trg-be-training-service.kaizen-
trg.svc.cluster.local","port":8092,"contextPath":"/betraining"},"iamproxy":{"host":"ka
izen-trg-be-iam-proxy-service.kaizen-
trg.svc.cluster.local","port":8091,"contextPath":"/iamproxy}}}'
```

...

...

□

* Service Name:	Please select
* Status:	betraining
	console
	database

Import

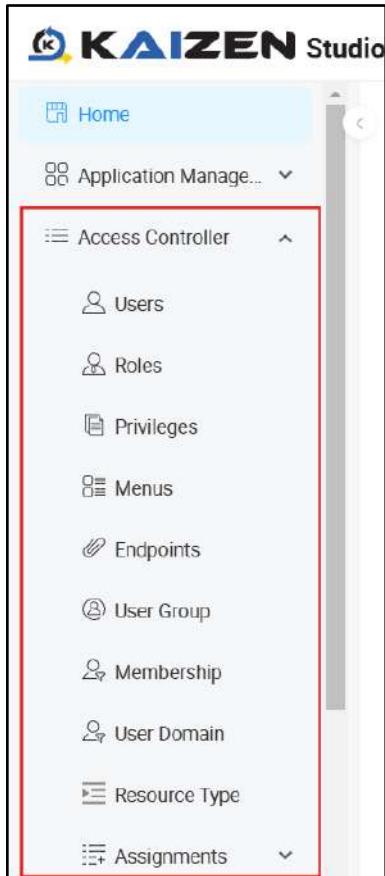
Tutorial 28: IAM

This tutorial covers the following Learning Objectives:

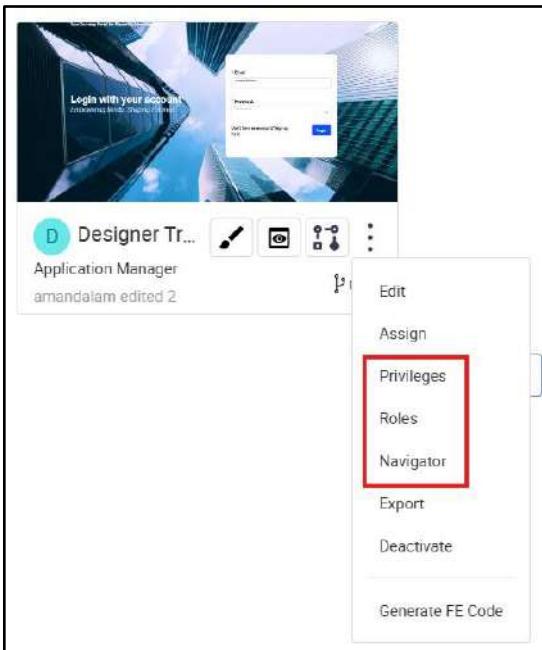
- Understand the concepts of IAM (Identity and Access Management) and Role-Based Access Control (RBAC).
- Explore the relationships between user domains, users, roles, privileges, and endpoints.
- Learn how to manage user roles, assign privileges, and secure access to endpoints in an application's domain environment.

In this tutorial, you will understand KAIZEN's IAM system with RBAC, including defining user domains, assigning roles to users, configuring privileges, and managing access to endpoint resources securely and efficiently. This structure supports dynamic role and privilege assignments, allowing administrators to fine-tune access control. Privileges and resources ensure least-privilege access, aligning with security best practices. By leveraging this RBAC model, your application can provide efficient, scalable, and secure user access management tailored to organizational needs.

KAIZEN's IAM features can be accessed here via the 'Access Controller' menu, which can be accessed only if you have the admin role assigned to your user account.



Note that these are all actual runtime IAM features configurable for your application. In addition to this, KAIZEN also allows you to create your IAM configurations during the design phase of your application in the application level from the studio console.



You may access the admin user account in the Training Playground domain to access the IAM features.

URL: <https://kaizen-training.toppanecquaria.com/#/login>

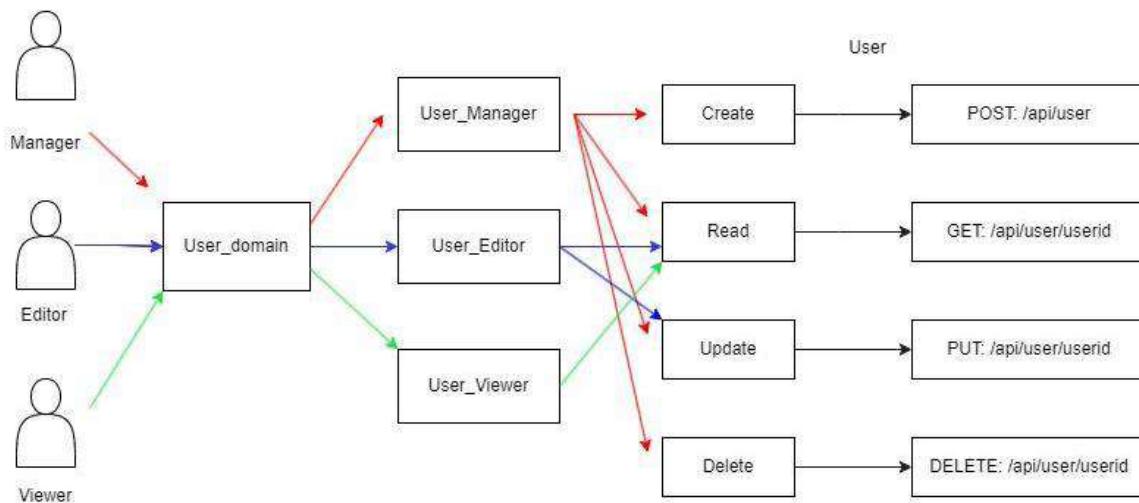
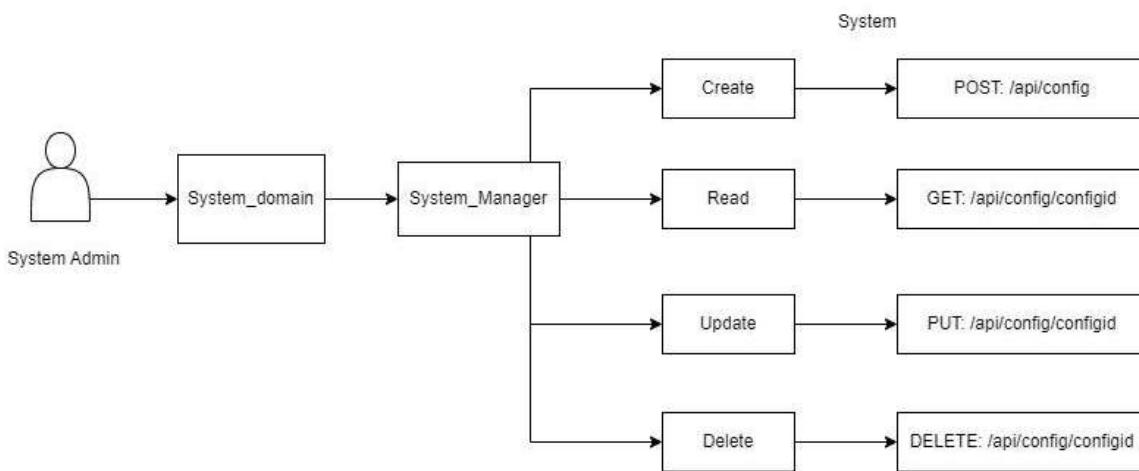
User Domain: Training Playground

Username: admin

Password: Password\$1234

Relationship: User domain ->* User ->* role ->* privileges ->* resources (API)

endpoints/menu/page)



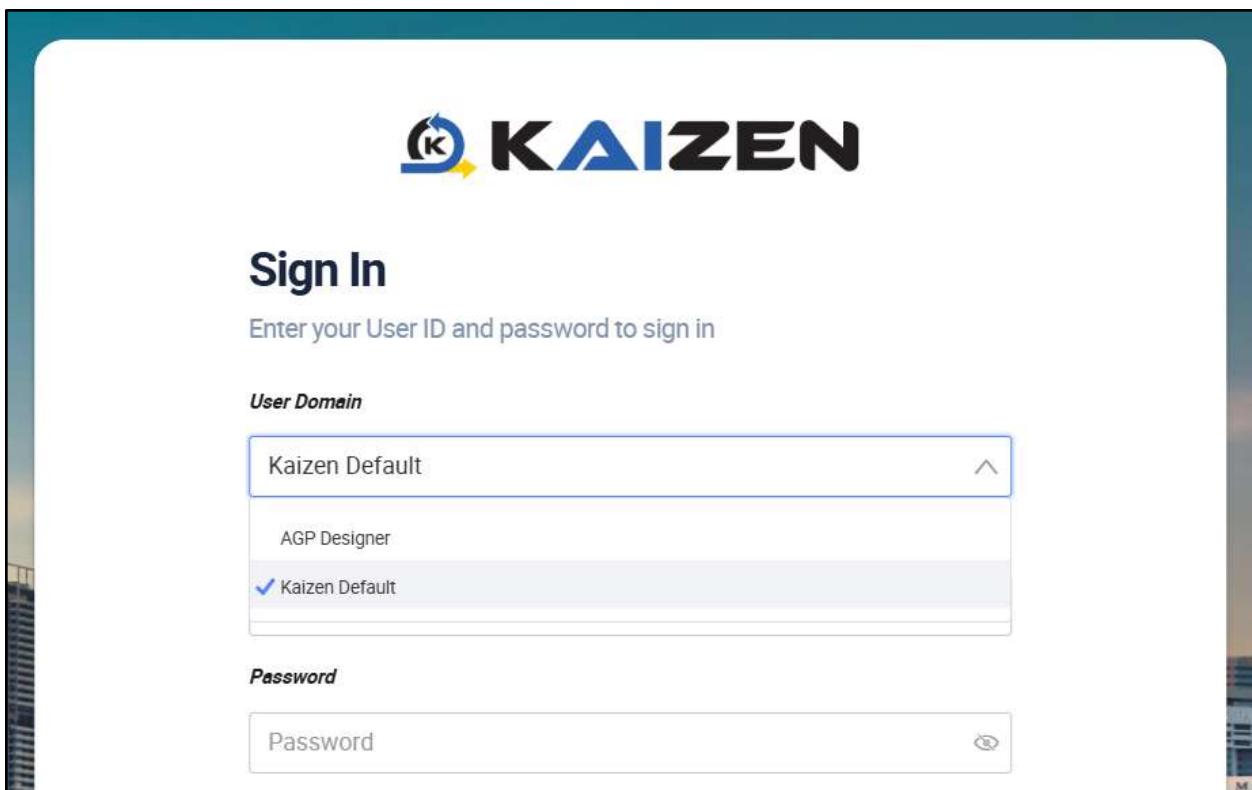
Example Flow

1. Currently there are 3 users in the "Sales" domain who are assigned to 3 roles: "Manager", "Editor" and "Viewer."
2. The Editor role has privileges to **READ** and **UPDATE** sales data endpoints.
3. The Viewer role has privileges to **READ** sales data endpoints.
4. The Manager gains combined access to **CREATE, READ, UPDATE, DELETE** sales data endpoints

1. User Domain

The User Domain acts as a boundary for user management and access control. It groups users, roles, privileges, and resources to enforce segregation of duties and multi-tenancy. Each domain defines its unique policies, ensuring users and roles within one domain do not conflict or overlap with those in another. For example, a project might have 2 applications, one being a customer facing portal (Internet) and the other being an internal staff portal (Intranet). In this case, the User Domain is introduced to maintain separation and enforce access control between the two user groups.

The user domain can only be accessed and configured by the 'super admin' role from the 'Kaizen Default' domain. You are also able to see this domain from KAIZEN's login screen.



KAIZEN Studio

Access Controller/ User Domain

User Domain

No.	User Domain Code	Display Name	Status	Actions
1	designer	AGP Designer	Active	View Edit More
2	kaizen_default	Kaizen Default	Active	View Edit More

2. User

A User represents an individual entity with a unique identity within a specific domain.

- Each user can belong to only one domain.
- A user can be assigned one or more roles, depending on their responsibilities.
- Users can inherit privileges indirectly through their assigned roles.

KAIZEN Studio

Access Controller/ Users

Users

No.	User Domain	Account ID	Display Name	Account Status	Remarks	Actions
1	AGP Designer	admin	Admin	Locked		View Edit More
2	AGP Designer	alvintay	Alvin Tay	Active		View Edit More
3	AGP Designer	amandalam	amandalam	Active		View Edit More

User Assignments

Domain: AGP Designer

Assigned Roles

No.	User Domain	Role Code	Role Name
1	AGP Designer	agp_admin	Admin
2	AGP Designer	agp_common_role	Common Role
3	AGP Designer	agp_wf	Workflow User

Remove

3. Role

A Role is a collection of privileges that represent a job function or responsibility within the domain.

- Roles define what actions a user can perform by grouping privileges.
- Users can have multiple roles, enabling flexible access control.
- Roles are reusable and can be assigned to multiple users.

KAIZEN Studio

Application Manager - Access Controller/ Roles

Users

Roles

Privileges

Menus

Endpoints

User Group

Membership

Resource Type

Assignments

Job Scheduler

Audit Trails

Notification Manager

User Domain: AGP Designer

Role Code:

Role Name:

Subject Type:

Status:

Remarks:

Create **Assign** **Export** **Export All** **Import** **Import Application Roles**

No.	User Domain	Role Code	Role Name	Subject Type	Status	Remarks	Actions
1	AGP Designer	agp_super_admin	Super Admin	General	Active	AGP Super Admin	View Edit More
2	AGP Designer	agp_project_leader	Project Leader	Application	Active	AGP Project Lead	View Edit More

In a project scenario, our application offers some predefined roles that can be assigned to users, such as **Project Manager**, **Developer**, **UI Designer**, **Internal Viewer**, and **External**

Viewer. Each role is tailored with specific access levels, enabling precise control over features and functionalities for managing both applications and the overall project.

Subject Type	Role Name
Project	Project Manager
	Maintainer
Application	Project Leader
	Application Manager
	Developer
	UI Developer
	Internal Viewer
	External Viewer

More information on Subject will be explained below.

4. Privileges

Privileges are granular permissions that define specific actions a role can perform on specific resources.

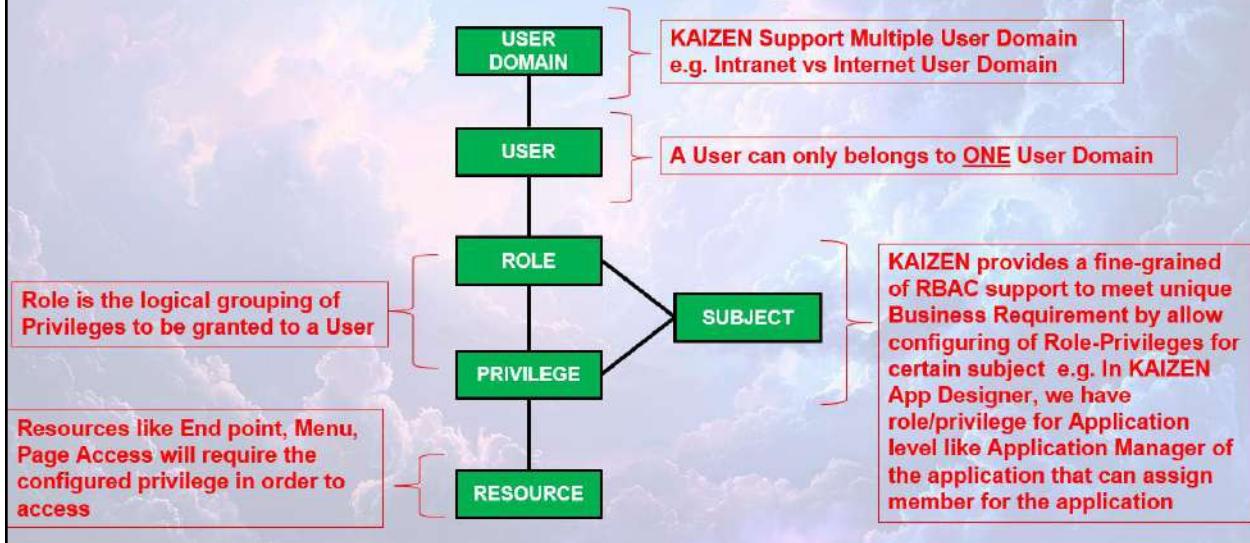
- A role can have multiple privileges, specifying the allowed operations.
- Privileges enforce fine-grained control over application features and functionalities.

5. Resources (Endpoints)

Resources refer to the endpoints or specific system assets that users can access. In our case, KAIZEN segregates access to various resources through endpoints.

- Privileges are directly linked to resources, ensuring precise control over what each user or role can access.
- Resources can include APIs, web pages, data objects, or any system element exposed through an endpoint.
- Privileges define how resources are accessed, ensuring security and preventing unauthorized actions.

KAIZEN Security Architecture - RBAC



Subject adds a contextual layer to RBAC, by ensuring that roles and privileges apply only within a defined scope.

Court Case Example: How "Subject" Works

We can use Subject to add RBAC control to the data level. In a Court Case Management system, we can use Subject to control the access rights for each court case.

Imagine a judicial system where different individuals take on roles such as Judges, Lawyers, and Clerks. Each role comes with specific privileges, but access to court cases should be restricted based on assignment.

- **Admin users** may oversee case management but should not influence case outcomes.
- **Lawyers** should only access the cases they are representing and file documents, but they cannot assign judges.
- **Judges** should have full decision-making privileges over cases they are assigned to, but they cannot access or modify cases they are not involved in.

In this scenario, the court case itself acts as the Subject. A judge is not simply assigned a role, they are assigned a role in relation to a specific case.

Practical Example: Judge Case Assignment

Consider Judge Alice and Judge Bob. They are both Judges, but their privileges are scoped to the cases they preside over.

- Judge Alice is assigned with Judge role to Case #1001. She can review evidence, issue rulings, and set hearings for this case.
- Judge Bob is assigned with Judge role to Case #1002. He has similar privileges but only within the context of his case.
- Neither Judge Alice nor Judge Bob can interfere in each other's cases, even though both are Judges.

This fine-grained control is achieved by defining the court case as a Subject, ensuring that privileges are only valid within the assigned scope.

Practical 28.1: Assignment of custom menu (Optional)

This tutorial covers the following Learning Objectives:

- Learn how to assign custom menus and pages to your application.
- Enhance navigation and user experience by providing tailored menus.
- Understand how to structure your application to meet user needs more effectively.

In this tutorial, you will explore the process of assigning custom menus and pages to your application. This customization enables a personalized user experience, where navigation and page layout are adapted to meet the specific needs of your users or organization.

Name	Display Label	Path	Priority	Status	Target	User Domain	Actions
Table	Table	/table	1	Active	/#/table	AGP Designer	View Edit More
Form Submission	Form Submission	/formSubmission	2	Active	/#/formSubmission	AGP Designer	View Edit More
Workflow Init	Workflow Init	/workflowInit	3	Active	/#/workflowInit	AGP Designer	View Edit More
Home	Home	/home	99	Active	/#/console	AGP Designer	View Edit More
> Application Management	Application Management	/mgmt	96	Active		AGP Designer	View Edit More
> Access Controller	Access Controller	/access	97	Active		AGP Designer	View Edit More
> Job Scheduler	Job Scheduler	/jobScheduler	98	Active		AGP Designer	View Edit More
> Audit Trail	Audit Trails	/auditTrail	99	Active		AGP Designer	View Edit More

Create and Assign Menu

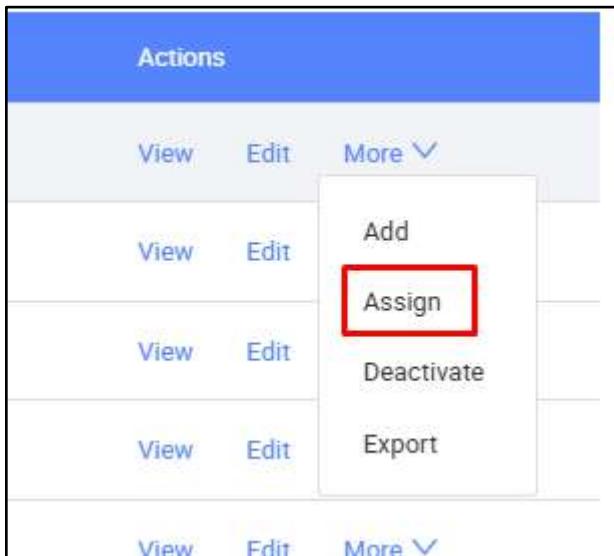
- In the menu edit/create, you can fill in different fields such as name, label, path and target.

The screenshot shows the KAIZEN Studio interface with the 'Access Controller / Menus' page selected. On the left, there's a sidebar with various navigation options like Home, Application Management, Access Controller, Users, Roles, Privileges, and so on. The 'Menus' option is highlighted. In the main area, there's a search bar and a table listing existing menus. A modal window titled 'Edit Menu' is open, allowing the creation of a new menu item named 'Table' with a path of '/table'. The table below lists other menu items such as 'Form Submission', 'Workflow Inbox', 'Home', and 'Application Management'.

- Menu also supports nested layer

	Name	Display Label	Path	Priority
<input type="checkbox"/>	Table	Table	/table	1
<input type="checkbox"/>	Form Submission	Form Submission	/formsubmission	2
<input type="checkbox"/>	Workflow Inbox	Workflow Inbox	/workflowinbox	3
<input type="checkbox"/>	Home	Home	/home	95
<input type="checkbox"/>	Application Management	Application Management	/mgmt	96
<input type="checkbox"/>	Templates	Templates	/templates	94
<input type="checkbox"/>	Projects	Projects	/projects	95
<input type="checkbox"/>	Applications	Applications	/applications	96

- Click on More > Assign



- Users can assign custom privilege to allow only authorized users to view the menus

Menu Assignment

Domain: AGP Designer

Assigned Privileges

<input type="checkbox"/>	No.	User Domain	Privilege Code	Privilege Name
<input type="checkbox"/>	1	AGP Designer	be001	BE training api

Remove

Search

Available Privileges

<input type="checkbox"/>	No.	User Domain	Privilege Code	Privilege Name	Remarks
<input type="checkbox"/>	1	AGP Designer	a10zx	AGP Generate Privilege Code	
<input type="checkbox"/>	2	AGP Designer	a1123	AGP Export Menu	

Practical 28.2: Endpoint Assignment (Optional)

This tutorial covers the following Learning Objectives:

- Understand how to assign endpoints to different functionalities within your application.
- Learn how to organize and manage API endpoints efficiently.
- Enhance application performance by structuring endpoints according to functional needs.

In this tutorial, you will learn how to assign and manage API endpoints for various functionalities within your application. Organizing endpoints ensures that each API is structured and accessible, making your backend services more efficient and easier to maintain.

The screenshot shows the KAIZEN Studio interface with the following details:

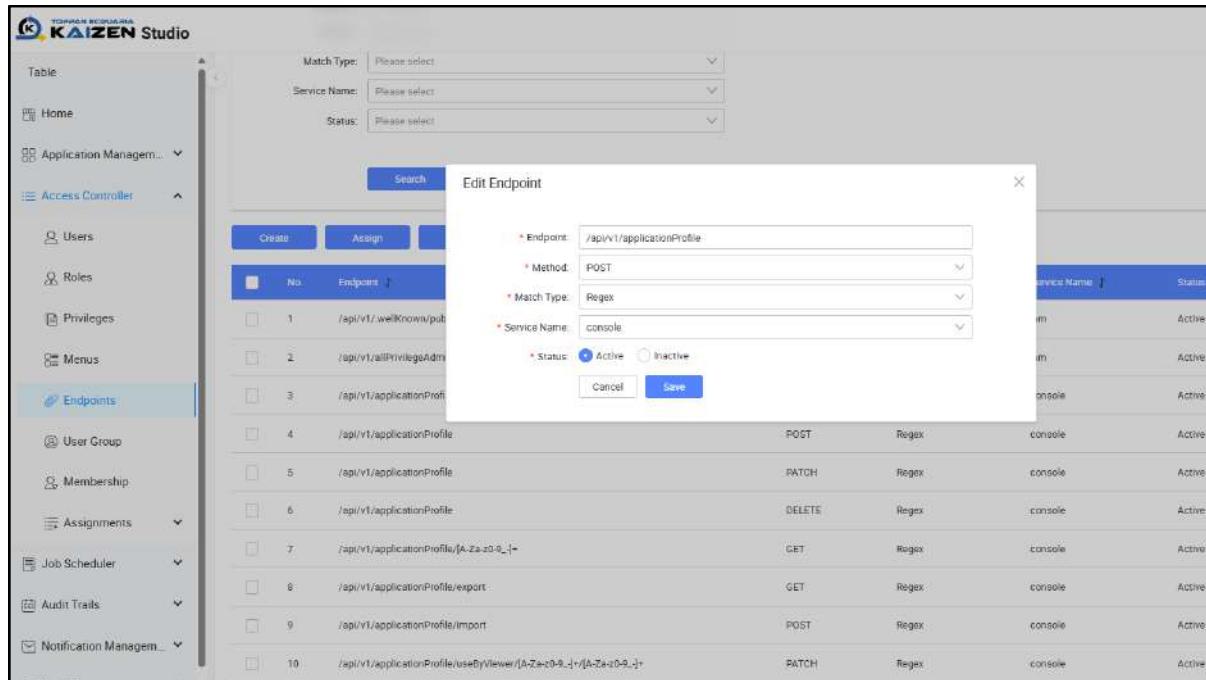
- Left Sidebar:** A navigation menu with sections like Home, Application Management, Access Controller, Users, Roles, Privileges, Menus, Endpoints (which is selected and highlighted in blue), User Group, Membership, Assignments, Job Scheduler, Audit Trails, Notification Management, Workflow, and Development Tools.
- Top Bar:** The title "Access Controller / Endpoints" and the section title "Endpoints".
- Search and Filter:** A search bar and dropdown filters for Endpoint, Method, Match Type, Service Name, and Status.
- Action Buttons:** Buttons for Create, Assign, Import, Export, and Export All.
- Table:** A list of 7 assigned endpoints with columns: No., Endpoint, Method, Match Type, and Service Name.

No.	Endpoint	Method	Match Type	Service Name
1	/api/v1/.wellKnown/publicKey	GET	Regex	iam
2	/api/v1/allPrivilegedAdmin	POST	Regex	iam
3	/api/v1/applicationProfile	GET	Regex	console
4	/api/v1/applicationProfile	POST	Regex	console
5	/api/v1/applicationProfile	PATCH	Regex	console
6	/api/v1/applicationProfile	DELETE	Regex	console
7	/api/v1/applicationProfile/{A-Za-z0-9_-}+	GET	Regex	console

Create new endpoint assignment

- In the endpoint assignment page, click on “Create” button.

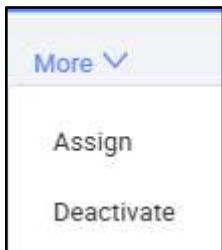
Field	Description
Endpoint	The specific URL or path where the API request is directed.
Method	The HTTP method used for the request (e.g., GET, POST, PUT, DELETE).
Match Type	The criteria used to match the request to the endpoint (e.g., exact match, prefix match, regex).
Service Name	<p>The name of the service or functionality that the endpoint is associated with.</p> <p>Note: For custom service to appear in the dropdown, it needs to be added to the “service.names.mesh” property</p>
Status	The current status of the endpoint (e.g., active, inactive).



- Endpoint also support regex expression

* Endpoint: /api/v1/applicationProfile/useByViewer/[A-Za-z0-9_-]+/[A-Za-z0-9_-]+

- After creating endpoint, click on More > Assign



- Users can also assign custom privilege to the endpoint to allow only authorized users to call the endpoint

The screenshot shows the 'Endpoint Assignment' dialog box. At the top, it displays the 'Domain: AGP Designer'. Below this, the 'Assigned Privileges' section shows a single entry: 'product - GET: /api/v1/betraining/[A-Za-z0-9_-]+'. This entry has a checkbox next to it, which is checked. The table columns are 'No.', 'User Domain', 'Privilege Code', and 'Privilege Name'. The value 'be001' is listed under 'Privilege Code' and 'BE training api' under 'Privilege Name'. A 'Remove' button is located below the table. Below this section is a search bar with the word 'Search' and a table of available privileges. The table has columns: 'No.', 'User Domain', 'Privilege Code', 'Privilege Name', and 'Remarks'. Three entries are listed:

- No. 1, User Domain AGP Designer, Privilege Code a10zx, Privilege Name AGP Generate Privilege Code
- No. 2, User Domain AGP Designer, Privilege Code a1123, Privilege Name AGP Export Menu
- No. 3, User Domain AGP Designer, Privilege Code a1b01, Privilege Name AGP Assignments

Note:

- For custom service to appear in the dropdown, it needs to be added to the "service.names.mesh" property in application.properties file.

Example:

application.properties file

```
□ ...
...
service.names.mesh: '{"iam":{"host":"kaizen-trg-be-iam-service.kaizen-
trg.svc.cluster.local","port":8082,"contextPath":"/iam"},"console":{"host":"kaizen-
trg-be-console-service.kaizen-
trg.svc.cluster.local","port":8083,"contextPath":"/console"},"common":{"host":"kaizen-
trg-be-common-service.kaizen-
trg.svc.cluster.local","port":8084,"contextPath":"/common"},"job":{"host":"kaizen-trg-
be-job-service.kaizen-
trg.svc.cluster.local","port":8085,"contextPath":"/job"},"gateway":{"host":"kaizen-
trg-be-gateway-service.kaizen-
trg.svc.cluster.local","port":8081,"contextPath":""},"pagescan":{"host":"kaizen-trg-
be-lighthouse-scanner-service.kaizen-
trg.svc.cluster.local","port":8086,"contextPath":""},"workflow":{"host":"kaizen-trg-
be-workflow-engine-service.kaizen-
trg.svc.cluster.local","port":8087,"contextPath":"/workflow"},"setup":{"host":"kaizen-
trg-be-setup-service.kaizen-
trg.svc.cluster.local","port":8089,"contextPath":"/setup"},"betraining":{"host":"kaize
n-trg-be-training-service.kaizen-
trg.svc.cluster.local","port":8092,"contextPath":"/betraining"},"iamproxy":{"host":"ka
izen-trg-be-iam-proxy-service.kaizen-
trg.svc.cluster.local","port":8091,"contextPath":"/iamproxy"}}'
```

...
...
□

The screenshot shows a user interface for selecting a service status. A dropdown menu is open, displaying three options: 'betraining', 'console', and 'database'. The 'betraining' option is highlighted with a red rectangular box. To the left of the dropdown, there is a field labeled 'Service Name:' with a placeholder 'Please select'. At the bottom left of the interface, there is a blue button labeled 'Import'.

- 2) After editing Endpoint Assignment, backend deployment needs to be restarted for the change to take effect

App	Status	Name	Kind	Namespace	Last Updated	
kaizen-be-core-proxy	● running a day	kaizen-trg-be-gateway-proxy	Deployment	kaizen-trg	a day	
	● running a day	kaizen-trg-be-iam-proxy	Deployment	kaizen-trg	a day	
	8081/TCP	kaizen-trg-be-gateway-proxy-service	Service	kaizen-trg	a day	
kaizen-trg-be-cm-config	8091/TCP	kaizen-trg-be-iam-proxy-service	Service	kaizen-trg	a day	
	Data: 20	kaizen-trg-be-cm-config	ConfigMap	kaizen-trg	a day	
	● running 4 hours	kaizen-trg-be-training	Deployment	kaizen-trg	4 hours	
kaizen-trg-be-studio-service	8092/TCP	kaizen-trg-be-training-service	Service	kaizen-trg	a day	
	8093/TCP	kaizen-trg-be-console-service	Service	kaizen-trg	2 days	
	8095/TCP	kaizen-trg-be-job-service	Service	kaizen-trg	2 days	
	8096/TCP	kaizen-trg-be-lighthouse-scanner-service	Service	kaizen-trg	2 days	
	8097/TCP	kaizen-trg-be-workflow-engine-service	Service	kaizen-trg	2 days	
kaizen-trg-be-core-service	8094/TCP	kaizen-trg-be-common-service	Service	kaizen-trg	2 days	
	8091/TCP	kaizen-trg-be-gateway-service	Service	kaizen-trg	2 days	
	8092/TCP	kaizen-trg-be-iam-service	Service	kaizen-trg	2 days	
kaizen-trg-be-studio	● running 4 hours	kaizen-trg-be-console	Deployment	kaizen-trg	4 hours	
	● running 4 hours	kaizen-trg-be-job	Deployment	kaizen-trg	4 hours	
	● running 4 hours	kaizen-trg-be-lighthouse-scanner	Deployment	kaizen-trg	4 hours	
	● running 4 hours	kaizen-trg-be-workflow-engine	Deployment	kaizen-trg	4 hours	
kaizen-trg-be-core	● running 4 hours	kaizen-trg-be-common	Deployment	kaizen-trg	4 hours	
	● running 4 hours	kaizen-trg-be-gateway	Deployment	kaizen-trg	4 hours	
	● running 4 hours	kaizen-trg-be-iam	Deployment	kaizen-trg	4 hours	