

# **Technical Solution Approach**

## **Contents**

### **1 Introduction. 2**

#### **1.1 About this document 2**

##### **1.1.1 Purpose & Scope of Document 2**

### **2 Component Design. 2**

#### **2.1 Component Design Diagram.. 2**

##### **2.1.1 Overall Workflow.. 2**

##### **2.1.2 Low-level Design. 2**

### **3 Technology & Frameworks are to be used. 3**

### **4 Solution Approach. 3**

# **1.Introduction**

## **1.1. About this document**

It specifies the approach of implementing the entire workflow of a Pluggable component in a Dynamic UI Framework.

### **1.1.1. Purpose & Scope of the Document**

- The objective is to collect and analyze all assorted ideas that have come up to define the Pluggable Components in a Dynamic UI Framework.
- It provides a detailed overview of the rendering of those feature-based pluggable components that are mutable(by database entries).
- It explains how the features of a component through server configuration can be changed.
- Specifies the functionality and reusability of a pluggable component.

## **2.Component Design**

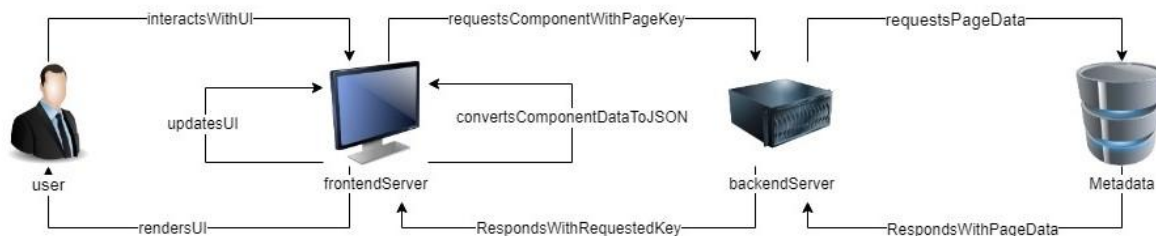
### **2.1. Component Design Diagram**

To enable interactivity on a web page, metadata is stored in a database. This metadata is used to retrieve or fetch data and build the feature-based components. Once the components have been built, the

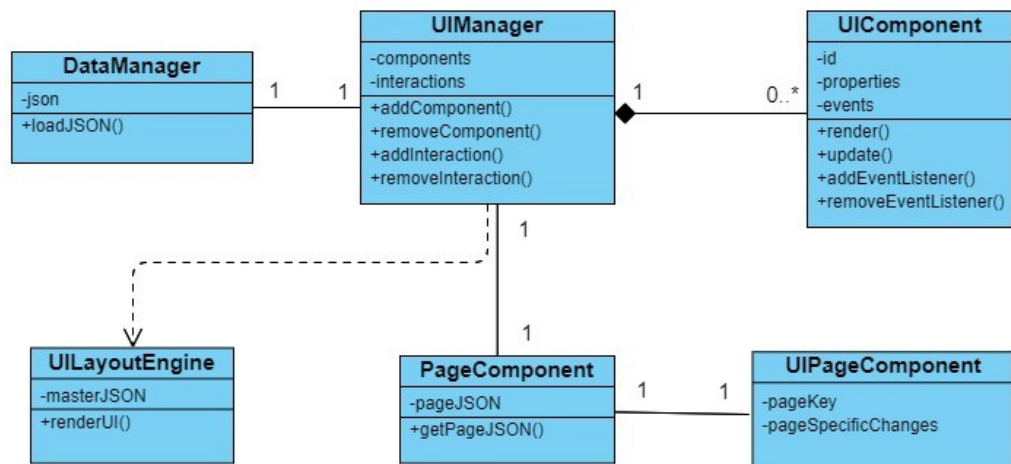
page key is passed to render the component using React. React is a JavaScript library that simplifies the process of building user interfaces. It allows for efficient updates to be made to the web page without requiring a complete refresh. By rendering the component in React, users can experience seamless and interactive web pages, improving the overall user experience.

### 2.1.1. Overall Workflow

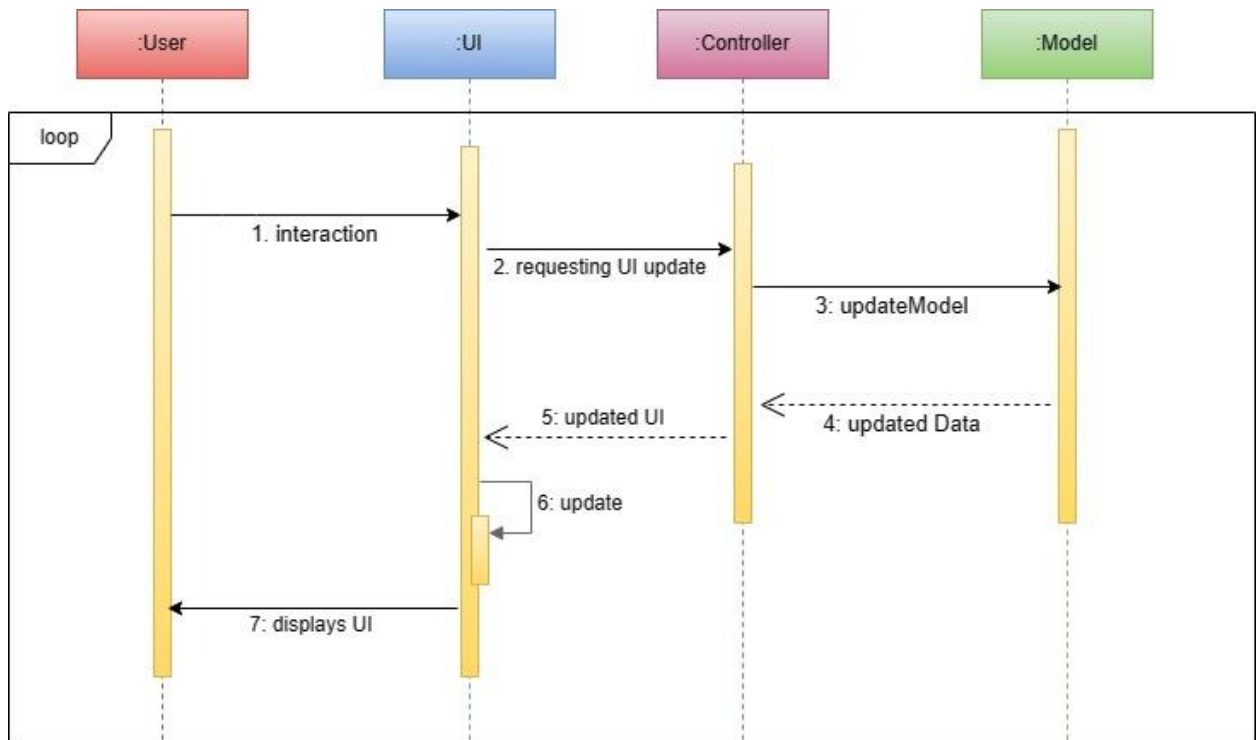
This section should explain the details of end-to-end workflow with the method level details. Below is the sample diagram just for reference.



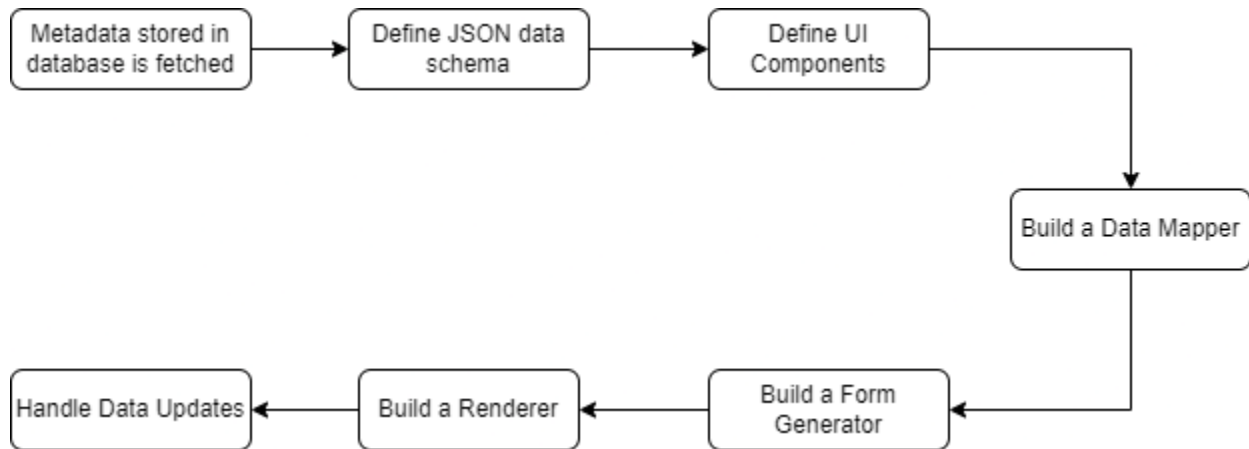
## CLASS DIAGRAM :



## SEQUENCE DIAGRAM:



### 2.1.2. Low-level Design



- Define JSON data schema: we would first need to define a JSON data schema that describes the structure and properties of the data that will be used to render the UI. This could include data types, field names, and validation rules for each property.
- Define UI components: Using React, we would define the UI components that will be used to render the JSON data. You could use tools like React-Bootstrap or Material UI to build reusable UI components that can be easily customized quickly.
- Build a Data Mapper: we would then build a data mapper that maps the JSON data to the UI components. The data mapper would take the JSON data and dynamically generate the corresponding UI components based on the schema you defined in Step 1.
- Build a Form Generator: we would build a form generator that allows users to create and edit JSON data using the same dynamic UI components. The

form generator would take the JSON schema and dynamically generate a form that allows users to create or edit the data in a user-friendly way.

- **Build a Renderer:** we would build a renderer that takes the JSON data and renders it as a read-only view using the same dynamic UI components. This would allow you to display the JSON data to users in a way that is easy to read and understand.
- **Handle Data Updates:** Finally, we would handle data updates in real time by using state management tools like Redux or React Context. This would allow you to update the UI components in real time as users interact with the data, providing a responsive and dynamic user experience.

### **3. Technology & Frameworks to be used**

React Js.

JSON

### **4. Solution Approach**

- 1) In React, the createElement() function is used to create and return a new React element. It is one of the fundamental building blocks of React applications.
- 2) Using the createElement() function we created our own react component
- 3) A separate JSON file with all the data is created to render into our component
- 4) Our react component basically accepts a tag, text, type, children, and properties if any
- 5) Then based on conditional rendering HTML tags with data and properties are passed from JSON files into this component
- 6) Page key is also passed to dynamically change the UI framework

## **Advantages of createElement:**

Dynamic element creation: With createElement, you can dynamically create elements based on certain conditions or user actions. This can be particularly useful when building complex and interactive applications.

Improved performance: createElement creates a virtual DOM element, which can be more performant than creating a real DOM element directly. This is because React can optimize the rendering process by only updating the parts of the DOM that have changed, rather than re-rendering the entire page.

Cleaner code: Using createElement can help make your code cleaner and more readable, particularly if you're using JSX. This is because JSX can sometimes make the code harder to read, particularly when working with complex nested elements.

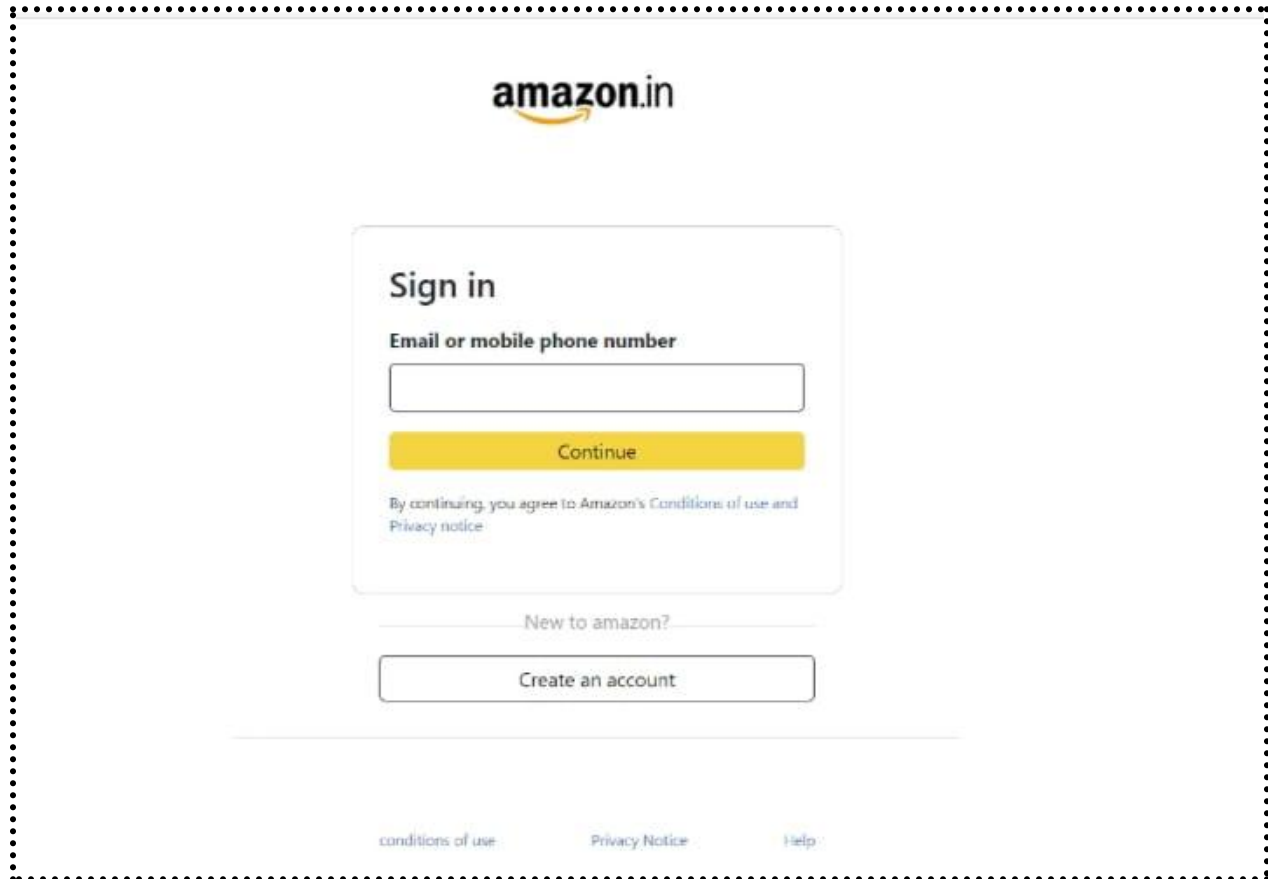
Flexibility: createElement gives you more flexibility and control over the structure and behavior of your components. For example, you can use createElement to add custom attributes or event handlers to your elements or to create custom components that are not available in the React library.

## **Component explanation:**

- 1) When our component is called, it accepts a tag, text, type, children, and props
- 2) Based on the conditional rendering approach firstly it checks for 'text'
- 3) If it passes then it returns the tag, props, and text
- 4) If that fails it checks for 'key'
- 5) If the key exists , it checks the 'type' of the key by passing the 'type' into it and our component is called recursively
- 6) Then it checks for 'children', if that exist, it maps each child, and the component is again called based on a recursive approach finally it returns the children with their parent tag
- 7) If all these fails then it returns only tag and props

## 5. Implementation:

### PAGEKEY 1



The image shows the Amazon India (amazon.in) sign-in page. At the top center is the Amazon logo. Below it is a white rounded rectangle containing the 'Sign in' heading. Under the heading is a label 'Email or mobile phone number' followed by a text input field. Below the input field is a yellow 'Continue' button. Under the button is a line of text: 'By continuing, you agree to Amazon's Conditions of use and Privacy notice'. Below this white box is a horizontal line, followed by the text 'New to amazon?'. Under this text is a white rounded rectangle with a black border containing the text 'Create an account'. At the bottom of the page, there is a horizontal line, and below it are three links: 'conditions of use', 'Privacy Notice', and 'Help'.

amazon.in

Sign in

Email or mobile phone number

Continue

By continuing, you agree to Amazon's Conditions of use and Privacy notice

New to amazon?

Create an account

[conditions of use](#) [Privacy Notice](#) [Help](#)



## PAGEKEY 2

amazon.in

Sign in

+91 9361455974

Password

Forgot password

Continue

contact for help

New to amazon?

Create an account

conditions of use

Privacy Notice

Help

## PAGEKEY 1

Component 1

Child Component 2

### Best Seller

Best Seller



### Exclusive offers inside!

Take a look into our brand new Wool sweater, from our best seller

What if you could use 100% of your brain?

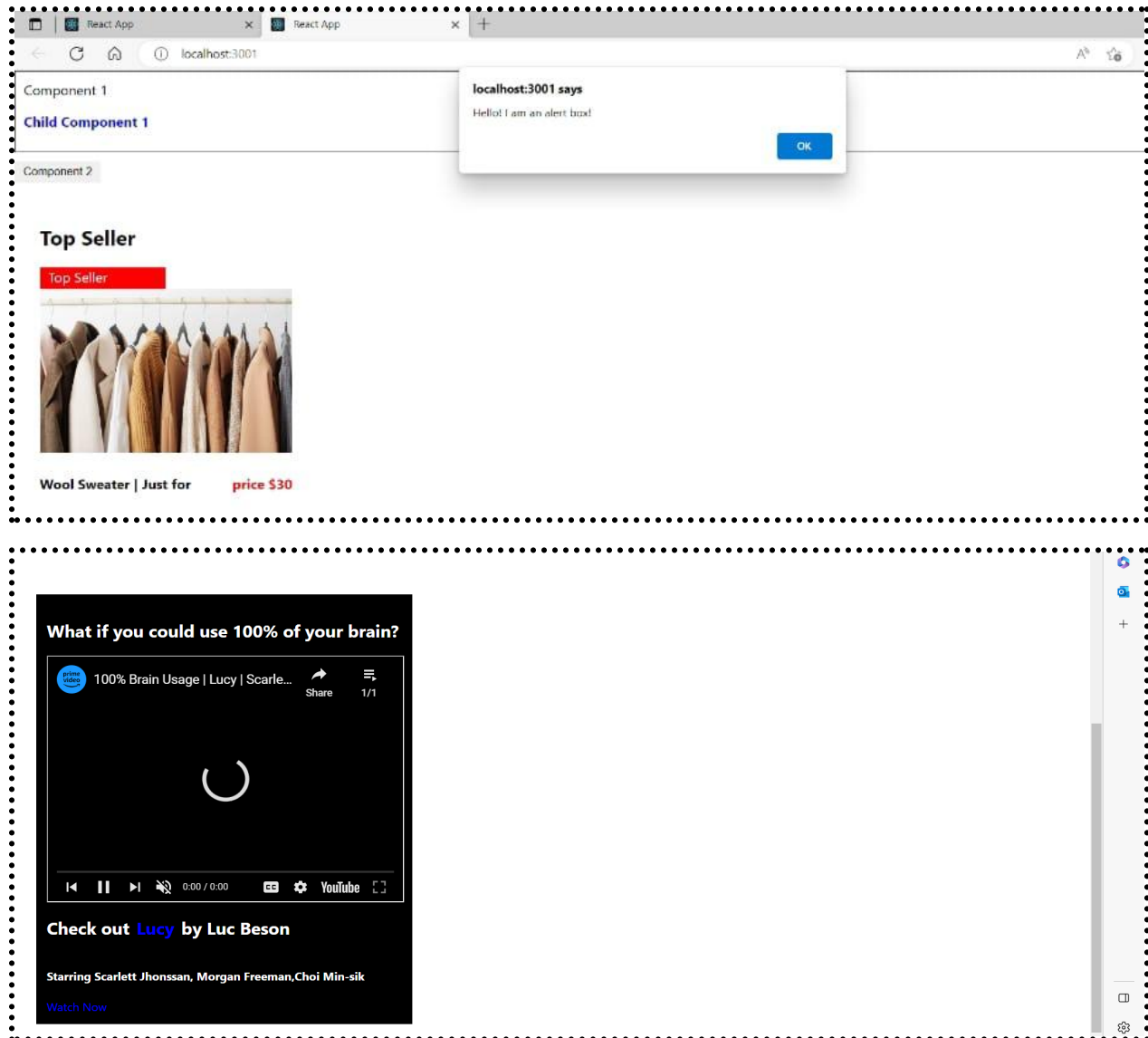


Check out **Lucy** by Luc Besson

Starring Scarlett Johansson, Morgan Freeman, Choi Min-sik

[Watch Now](#)

## PAGEKEY2



The above-mentioned samples are the actual implementation we have done for the dynamic UI framework. Basically, we have all the tags, i.e. the structure in the JSON format. We

implemented a component that accepts these structures with data from JSON, (using `createElement` our dynamic component renders the structure and data from JSON).

As mentioned in the problem statement, the proposed component should dynamically change as per our needs. Our approach is to achieve this by using page key, in the above-mentioned figures, a single component is used to render the structure from JSON, and using `pagekey` we change the UI dynamically as per our requirement.

In the first example, a Sign in page similar to Amazon is created, the first figure is created with the `pagekey1`, and the second page is the next page of sign-in, where we have shown few changes from the previous page, only a few contents were changed, and this is `pagekey2`, and all the remaining repeated UI remains the same, without repeated the code again and again.

And similarly, a card component is created by us, and the image remains the same in both page keys, so we didn't repeat the code. We just passed the page key to the places where the UI changes, the difference between the two is in flex-direction, so we minimized the minor change with the page key, without repeating the code.

The other components were created with the same approach.

## **6. Queries:**

We have designed the component with the page key and have implemented the component which calls the other functions as well, such as `onClick()`. But we have doubts about how to implement our component with stateful hooks, i.e. `useState`. So far we have implemented a few components, one of which is the sign-in page, but we have doubts on how to save the state of the input given by the user and how to pass that state or information given by them to the next page. For example, how to transfer the number given by the user as input to the next page.