

Be part of a better internet. [Get 20% off membership for a limited time](#)

# React: Building a Multi-Step Form with Wizard Pattern

Step-by-Step Implementation of a Multi-Step Registration with Wizard Pattern in React JS



Ishara Amarasekera · [Follow](#)

Published in Stackademic · 12 min read · May 13, 2024



144



6



**Step 1**

Email

Enter your email

Password

Choose a password

Re-enter Password

Re-enter your password

Get Started

**Step 2**

Organization

Enter organization name

Phone Number

Enter phone number

Your Message

Type your message here

Create Account Back

**Step 3**

Check your email

We sent you an email with further instructions

Open Email Back

## REACT MULTI-STEP FORM

with Wizard Pattern

TUTORIAL

## What is a Multi-Step Form ?

Multi-step forms, also known as wizards, segment the user interface flow into sequential stages. Each stage focuses on collecting specific information from the user, such as personal details, contact information, or preferences. By breaking down the registration process into smaller, more digestible chunks, multi-step forms enhance user experience and streamline data collection.

Using a multi-step form or wizard pattern can be considered a best practice in many cases, especially for complex processes or forms that involve multiple steps or stages. Here are some reasons why it's often considered a best practice:

- 1. Improved User Experience:** Breaking down a complex process into smaller, sequential steps can make it easier for users to understand and complete the task. Users are guided through each step, reducing cognitive overload and increasing the likelihood of completion.
- 2. Clear Progress Indication:** By displaying the current step and indicating progress, users have a clear understanding of where they are in the process and how much is left to complete.
- 3. Validation and Error Handling:** Each step provides an opportunity to validate user input and handle errors before proceeding to the next step. This helps prevent users from encountering issues later in the process and provides immediate feedback.
- 4. Flexibility and Adaptability:** The modular nature of the pattern allows for flexibility and adaptability. Steps can be added, removed, or rearranged as needed, making it easy to iterate on the process and accommodate changes in requirements or user feedback.

**5. Reusable Components:** Each step component can be designed to be reusable across different processes or forms, promoting code reusability and maintainability.

In this guide, we'll explore how to leverage Next.js, a popular React framework, to implement a registration flow using the wizard design pattern.

## Setting Up the React Project

### Step1 : Choose Your Development Framework

Depending on your preference and project requirements, you can choose to set up your project using either Next.js or React.js.

- **Option 1: Next.js :**

**Install Next.js and TypeScript:** Install Next.js and TypeScript along with the necessary types for React.

```
npx create-next-app@latest my-registration-app --ts
```

- **Option 2: React.js:**

**Install React.js using Create React App**

```
npx create-react-app my-registration-app
```

## Step 2 : Navigate to Your Project Directory

Use the following command to navigate to your project folder

```
cd my-registration-app
```

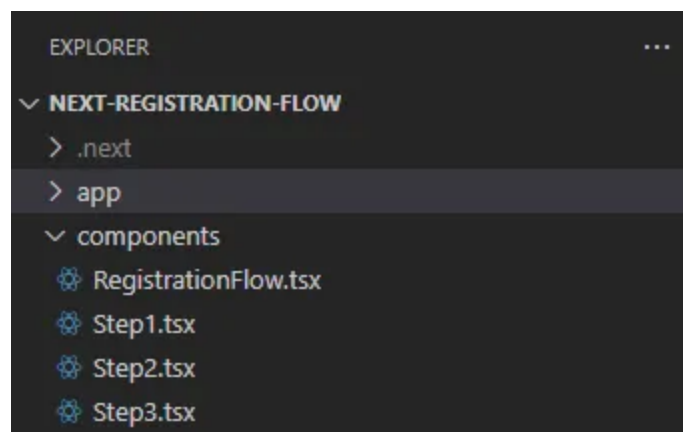
## Setting Up the Component Structure

### Step 1 : Create Components Directory

Create a `components` directory inside the `src` directory. If you don't have a `src` directory, you may create it in the root.

### Step 2 : Create Step Components

Within the `components` directory, create `Step1`, `Step2`, and `Step3` components, and create a parent component to manage the registration flow.



Setting up the Project Folder Structure

# Developing the Registration Flow Components

## Step 1 : Developing a Component for Step 1

Let's develop the first step with email, password, and re-enter password fields.

- Wireframe : Step 1

### Step 1

Email

Password

Re-enter Password

Get Started

Step 1 Component Design

- Source Code : Step 1

```
// Step1.tsx
```

```
import React, { useState, ChangeEvent } from 'react';
```

```
// Define the props interface for Step1 component
```

```
interface Step1Props {
```

```
  onNext: () => void; // Function to move to the next step
```

```
  onChange: (data: { [key: string]: string }) => void; // Function to handle for
```

```

}

const Step1: React.FC<Step1Props> = ({ onNext, onChange }) => {
  // State to manage form data
  const [formData, setFormData] = useState({
    email: '',
    password: '',
    reEnterPassword: '',
  });

  // Function to handle input changes
  const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
    const { name, value } = e.target;
    // Update the form data state
    setFormData({ ...formData, [name]: value });
    // Invoke the onChange function with updated form data
    onChange({ [name]: value });
  };

  return (
    <div>
      {/* Input fields for email, password, and re-enter password */}
      <input type="email" name="email" placeholder="Email" value={formData.email} />
      <input type="password" name="password" placeholder="Password" value={formData.password} />
      <input type="password" name="reEnterPassword" placeholder="Re-enter Password" value={formData.reEnterPassword} />
      {/* Button to move to the next step */}
      <button onClick={onNext}>Get Started</button>
    </div>
  );
};

export default Step1;

```

## • Code Breakdown: Understanding Step 1 Component

The Step1 component encapsulates the UI and functionality required for collecting user input for the first step of a multi-step registration process. Let's walk through the code.

1. We start by importing necessary modules from React, including `useState` and `ChangeEvent`, which are hooks used for managing state and handling

input changes, respectively.

2. We define an interface named `Step1Props` to specify the props that `Step1` component will receive. It includes two functions: `onNext`, which is a function to move to the next step, and `onChange`, which is a function to handle form data changes.
3. The `Step1` component is defined as a functional component. It takes props of type `Step1Props`.
4. Inside the `Step1` component, we define a state variable `formData` using the `useState` hook. It initializes with an object containing three properties: `email`, `password`, and `reEnterPassword`.
5. The `handleChange` function is defined to handle input changes. It receives a `ChangeEvent` and extracts the `name` and `value` of the input field that triggered the change. It then updates the `formData` state with the new value using the spread operator (`...`). Finally, it invokes the `onChange` function passed from the parent component, along with the updated form data.
6. Within the JSX, we render input fields for email, password, and re-enter password. Each input field is bound to the corresponding property in the `formData` state and uses the `handleChange` function to update the state when the input value changes.
7. Lastly, we render a button labeled “Get Started”, which triggers the `onNext` function when clicked, moving the user to the next step in the registration flow.

## Step 2: Developing a Component for Step 2

Assume the second step requires additional details such as organization, phone number, and message fields.

- **Wireframe: Step 2**

## Step 2

### Organization

### Phone Number

### Your Message

Create AccountBack

Step 2 Component Design

- **Source Code : Step 2**

```
// Step2.tsx

import React, { useState, ChangeEvent } from 'react';

// Define the props interface for Step2 component
interface Step2Props {
  onNext: () => void; // Function to move to the next step
  onChange: (data: { [key: string]: string }) => void; // Function to handle form data
}

const Step2: React.FC<Step2Props> = ({ onNext, onChange }) => {
  // State to manage form data
  const [formData, setFormData] = useState({
    organization: '',
    phoneNumber: '',
    message: '',
  });

  const handleNext = () => {
    onNext();
  };

  const handleChange = (e: ChangeEvent<HTMLFormElement>) => {
    const { name, value } = e.target;
    setFormData({
      ...formData,
      [name]: value,
    });
  };

  return (
    <div>
      <div>
        <input type="text" value={formData.organization} />
      </div>
      <div>
        <input type="text" value={formData.phoneNumber} />
      </div>
      <div>
        <input type="text" value={formData.message} />
      </div>
      <div>
        <button type="button" value="Create Account" />
        <button type="button" value="Back" />
      </div>
    </div>
  );
};
```



```
// Function to handle input changes
const handleChange = (e: ChangeEvent<HTMLInputElement | HTMLTextAreaElement>)
  const { name, value } = e.target;
  // Update the form data state
  setFormData({ ...formData, [name]: value });
  // Invoke the onChange function with updated form data
  onChange({ [name]: value });
};

return (
  <div>
    {/* Input fields for organization, phone number, and message */}
    <input type="text" name="organization" placeholder="Organization" value={f
    <input type="tel" name="phoneNumber" placeholder="Phone Number" value={for
    <textarea name="message" placeholder="Your Message" value={formData.messag
    {/* Button to move to the next step */}
    <button onClick={onNext}>Create Account</button>
  </div>
);
};

export default Step2;
```

## • Code Breakdown: Understanding Step 2 Component

The `Step2` component encapsulates the UI and functionality required for collecting additional user information in the second step of a multi-step registration process.

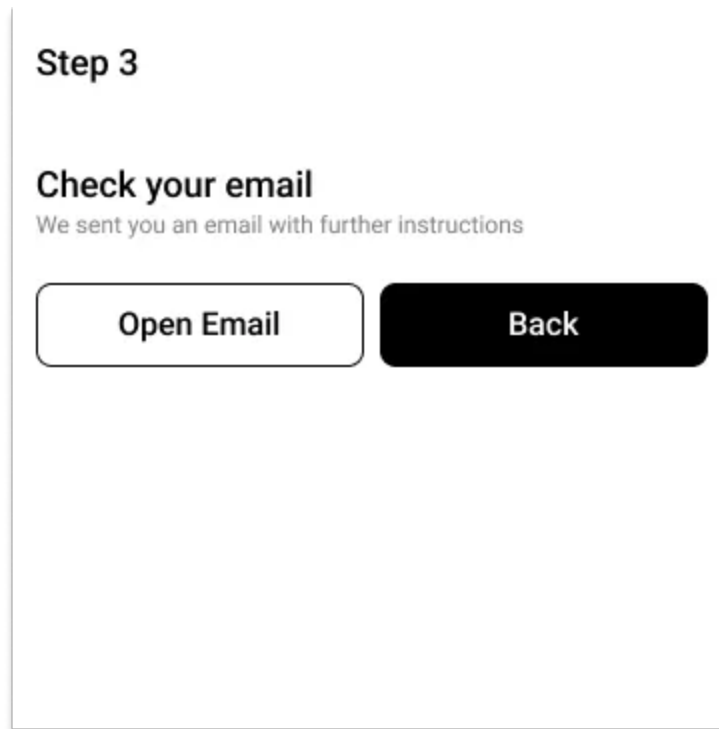
1. Again, we start by importing necessary modules from React, including `useState` and `ChangeEvent`, which are hooks used for managing state and handling input changes, respectively.
2. Then, we define an interface named `Step2Props` to specify the props that `Step2` component will receive. It includes two functions: `onNext`, which is a function to move to the next step, and `onChange`, which is a function to handle form data changes.

3. The `Step2` component is defined as a functional component. It takes props of type `Step2Props`.
4. Inside the `Step2` component, we define a state variable `formData` using the `useState` hook. It initializes with an object containing three properties: `organization`, `phoneNumber`, and `message`.
5. The `handleChange` function is defined to handle input changes. It receives a `ChangeEvent` and extracts the `name` and `value` of the input field that triggered the change. It then updates the `formData` state with the new value using the spread operator (`...`). Finally, it invokes the `onChange` function passed from the parent component, along with the updated form data.
6. Within the JSX, we render input fields for organization and phone number, along with a textarea for the message. Each input field is bound to the corresponding property in the `formData` state and uses the `handleChange` function to update the state when the input value changes.
7. Lastly, we render a button labeled “Create Account”, which triggers the `onNext` function when clicked, moving the user to the next step in the registration flow.

### Step 3 : Developing a Component for Step 3

Let's consider this as the last step with a title, description, and button.

- Wireframe: Step 3



Step 3 Component Design

- Source Code : Step 3

```
// Step3.tsx

import React from 'react';

// Define the props interface for Step3 component
interface Step3Props {
  onPrevious: () => void; // Function to move to the previous step
}

const Step3: React.FC<Step3Props> = ({ onPrevious }) => {
  return (
    <div>
      {/* Heading and message */}
      <h2>Check Your Email</h2>
      <p>We sent you an email with further instructions.</p>
      {/* Button to go back to the previous step */}
      <button onClick={onPrevious}>Back</button>
      {/* Button to open email app */}
      <button>Open Email App</button>
    </div>
  );
};
```

```
};  
  
export default Step3;
```

- **Code Breakdown: Understanding Step 3 Component**

The `Step3` component encapsulates the UI and functionality required for informing the user about the next steps after completing the registration process and providing options to navigate back or proceed further.

1. We define an interface named `Step3Props` to specify the props that `Step3` component will receive. It includes one function: `onPrevious`, which is a function to move to the previous step.
2. The `Step3` component is defined as a functional component. It takes props of type `Step3Props`.
3. Within the JSX, we render a heading “Check Your Email” and a message “We sent you an email with further instructions.” to inform the user about the next steps.
4. We render a button labeled “Back”, which triggers the `onPrevious` function when clicked, moving the user back to the previous step in the registration flow.
5. Additionally, we render a button labeled “Open Email App”, which presumably opens the user’s email application. However, the functionality to open the email app is not implemented in the provided code.

## **Step 4: Developing a Registration Flow Component:**

Finally, create a parent component to manage the registration flow.

- Source Code : Registration Flow Component

```
// RegistrationFlow.tsx

import React, { useState } from 'react';
import Step1 from './Step1';
import Step2 from './Step2';
import Step3 from './Step3';

const RegistrationFlow = () => {
  // State to track the current step of the registration flow
  const [step, setStep] = useState(1);
  // State to store form data across steps
  const [formData, setFormData] = useState({});

  // Function to move to the next step
  const handleNextStep = () => {
    setStep(step + 1);
  };

  // Function to move to the previous step
  const handlePreviousStep = () => {
    setStep(step - 1);
  };

  // Function to handle form data changes across steps
  const handleFormDataChange = (data: { [key: string]: string }) => {
    setFormData({ ...formData, ...data });
  };

  return (
    <div>
      {/* Render Step1 component if step is 1 */}
      {step === 1 && <Step1 onNext={handleNextStep} onChange={handleFormDataChange} />}
      {/* Render Step2 component if step is 2 */}
      {step === 2 && <Step2 onNext={handleNextStep} onChange={handleFormDataChange} />}
      {/* Render Step3 component if step is 3 */}
      {step === 3 && <Step3 onPrevious={handlePreviousStep} />}
    </div>
  );
};

export default RegistrationFlow;
```

- **Code Breakdown for the Registration Flow Component**

The `RegistrationFlow` component manages the overall registration process by controlling the flow between different steps (`Step1`, `Step2`, `Step3`). It keeps track of the current step, stores form data, and provides functions to move between steps and handle form data changes.

This is how it works.

1. We import necessary modules from React, including `useState`, which is a hook used for managing state.
2. We import the `Step1`, `Step2`, and `Step3` components from their respective files. These components represent the individual steps in the registration flow.
3. The `RegistrationFlow` component is defined as a functional component.
4. Inside the `RegistrationFlow` component, we define two state variables using the `useState` hook: `step` to track the current step of the registration flow, initialized to 1, and `formData` to store form data across steps, initialized as an empty object.
5. We define three functions:
  - `handleNextStep`: This function increments the `step` state by 1, moving the user to the next step in the registration flow.
  - `handlePreviousStep`: This function decrements the `step` state by 1, moving the user back to the previous step in the registration flow.
  - `handleFormChange`: This function takes an object `data` as input, which represents the form data changes. It merges the new form data with the

existing `formData` state using the spread operator ( `...` ), updating the `formData` state.

6. Within the JSX, we conditionally render the `Step1`, `Step2`, and `Step3` components based on the current value of the `step` state:

- If `step` is 1, we render the `Step1` component and pass the `handleNextStep` and `handleFormChange` functions as props.
- If `step` is 2, we render the `Step2` component and pass the `handleNextStep` and `handleFormChange` functions as props.
- If `step` is 3, we render the `Step3` component and pass the `handlePreviousStep` function as a prop.

## Step 5 : Rendering the Registration Flow Component in a Page

Create a Page: Create a page where you want to integrate the registration flow. For example, you can create a `RegistrationPage.tsx` file under the `pages` directory.

- Source Code : Registration Page

```
// RegistrationPage.tsx

import React from 'react';
import RegistrationFlow from '../components/RegistrationFlow';

const RegistrationPage = () => {
```

Open in app ↗

Medium

Search

Write



```
);
};
```

```
export default RegistrationPage;
```

- Code Breakdown for the Registration Page

The `RegistrationPage` component serves as a container for the `RegistrationFlow` component, providing a structured layout for displaying the registration process.

1. We import the necessary modules from React and define the `RegistrationPage` component as a functional component.
2. Within the `RegistrationPage` component, we return JSX that represents the content of the registration page.
3. The JSX consists of a `div` element with the classes "flex justify-center items-center h-screen". This CSS styling ensures that the content of the registration page is centered both horizontally and vertically on the screen, regardless of the screen size.
4. Inside the `div` element, we render the `RegistrationFlow` component. This component represents the entire registration flow, including all the individual steps (Step1, Step2, and Step3).
5. The `RegistrationFlow` component handles the logic for navigating through the registration steps and collecting user input, while the `RegistrationPage` component serves as the container for displaying the registration flow on the actual webpage.

## Running the Application

To run the application and verify its functionality locally, follow these steps:



## Step 1: Start the Development Server

Run the development server by executing the following command in your terminal:

- If you're using a Next.js project

```
npm run dev
```

- If you're using a React.js project

```
npm start
```

These commands will start the development server, and your project will be hosted at <http://localhost:3000>.

## Step 2 : Navigate to the Registration Page

Open your web browser and enter the URL corresponding to the registration page. For example, if your registration page is located at `/registration`, navigate to <http://localhost:3000/registration>.

Once you reach the registration page, you can view and interact with the multi-step registration flow to ensure that it works as expected.

## Conclusion

In this guide, we explored implementing a multi-step form with the wizard pattern in React.js, enhancing user experience and simplifying data collection. Leveraging components like Step1, Step2, and Step3, along with a parent RegistrationFlow component, we built an intuitive registration flow.

Using this pattern offers advantages like improved user experience, clear progress indication, and code reusability. Whether it's for a registration flow, checkout process, or any multi-step form, this approach simplifies complex tasks and guides users seamlessly.

To access the complete code for this tutorial and try it out yourself, you can find it in this GitHub repository: [next-registration-flow](#)

For a live demo, you can check out the application here: [Registration Flow Demo](#)

Now that you have a solid understanding of how to build a multi-step form with the wizard pattern in React.js, feel free to customize and expand upon the concepts discussed here to suit your specific project needs.

Happy coding!

[Reactjs](#)[Multi Step Form](#)[Nextjs](#)[Tutorial](#)[Software Development](#)



## Written by Ishara Amarasekera

79 Followers · Writer for Stackademic

Tech Lead | Software Consultant

Follow



### More from Ishara Amarasekera and Stackademic



Ishara Amarasekera in Level Up Coding

## Building an Authenticator App with Redux Toolkit: A Beginner's Guide

Learn how to efficiently manage authentication state in a React application...

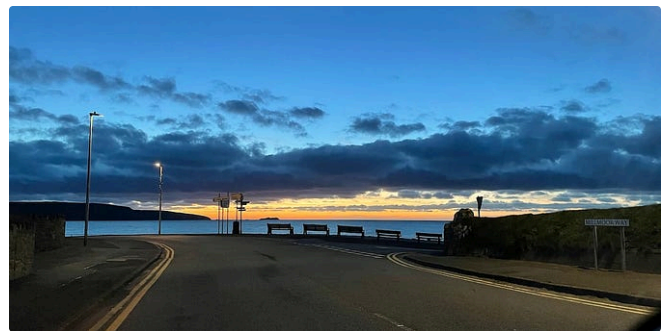
Oct 17, 2023



13



1



Peter Bunyan in Stackademic

## Who needs Redis, when Postgres will do?

May 9




594



5





 Dylan Cooper in Stackademic

## Oracle Tightens Grip on Java: Big Companies Beware!

It has been over a year since Oracle revised the pricing model for Java SE, and now their...


 Jun 13

 708

 11





 Ishara Amarasekera

## Streamlining Your Development Workflow: Managing Releases,...

Efficient software development workflow management

Sep 20, 2023

 8

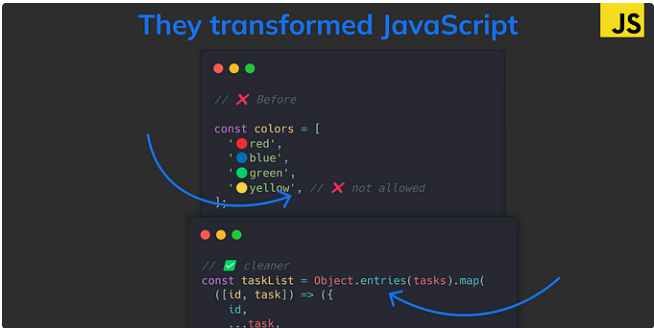




See all from Ishara Amarasekera

See all from Stackademic

## Recommended from Medium





Tari Ibaba in Coding Beauty

## The 5 most transformative JavaScript features from ES8

5 juicy ES15 features with new functionality for cleaner and shorter JavaScript code.



Jun 20



227



4



Vladimir Topolev in Numatic Ventures

## Animation of Ranking Tables with React OR again about...

Ranking tables are commonly used in different competitive events. If you enjoy...



Jun 25



11



### Lists



#### General Coding Knowledge

20 stories · 1375 saves



#### Stories to Help You Grow as a Software Developer

19 stories · 1190 saves



#### Coding & Development

11 stories · 696 saves



#### Good Product Thinking

11 stories · 628 saves

# Next.js Dynamic



Fungineer

## Next.js Dynamic (next/dynamic)

What is next/dynamic?



May 30



3



The SaaS Enthusiast

## React Hook Form Mastery: Creating Clean and Scalable Form...

Creating efficient, user-friendly forms is a pivotal task for developers. With the rise of...



Mar 14



6



1





Sviat Kuzhelev

## Mastering NextJS Architecture with TypeScript in Mind | Design...

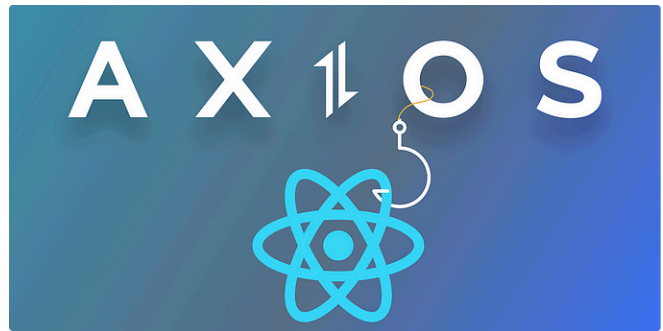
You'll never know how everything works, but you should understand the system. —by Svi...



Jun 17



6.7K



Kiran Kumal

## Implementing Refresh Token in React Using Axios, Zustand and...

Introduction

Jun 17



215

[See more recommendations](#)