



React Signup Page with Firebase Integration

A Beginner-Friendly Step-by-Step Implementation Guide

This guide will help you convert your signup page to React while keeping the rest of your site as HTML/CSS/JS. Each step is explained in detail with clear indications of what code you need to modify versus what is provided as an example.

 **BEGINNER NOTE:** Sections marked with this icon provide extra explanations for beginners.

 **REQUIRED INPUT:** Sections marked with this icon indicate where you need to add your own custom information.



 **EXAMPLE CODE:** Sections marked with this icon are example code that you can use as a reference, but may need to customize for your specific needs.

Table of Contents

1. [Before You Begin - Prerequisites](#)
2. [Firebase Console Setup](#)
3. [Environment Setup](#)
4. [Project Structure](#)
5. [Webpack Configuration](#)
6. [Firebase Configuration](#)
7. [HTML Integration](#)
8. [React Components](#)
9. [Styling](#)
10. [Building and Testing](#)
11. [Troubleshooting Common Issues](#)
12. [Next Steps](#)

Before You Begin - Prerequisites

 **BEGINNER NOTE:** This guide assumes you already have a website with HTML, CSS, and JavaScript. You'll be adding React to your existing website, not creating an entirely new React application. This "hybrid approach" lets you modernize your site gradually.

Before starting, make sure you have:

1. Basic knowledge of HTML, CSS, and JavaScript
2. An existing website with a signup page
3. A Firebase project already set up (we'll verify this in the next section)
4. A text editor (like Visual Studio Code) installed

Firebase Console Setup

Before writing any code, you need to ensure your Firebase project is properly configured.

1. Login to Firebase Console

1. Go to <https://console.firebase.google.com/>
2. Sign in with your Google account
3. Select your existing project from the dashboard

2. Verify Authentication is Enabled

1. In the Firebase console, click on "Authentication" in the left sidebar
2. Click on the "Sign-in method" tab
3. Make sure "Email/Password" is enabled (should show as "Enabled" in green)
 - If not enabled, click on "Email/Password", toggle the switch to "Enable", and save

 Firebase Authentication Setup (Example image - your screen will look similar)


3. Set Up Firestore Database

1. In the Firebase console, click on "Firestore Database" in the left sidebar
2. If you haven't created a database yet, click "Create database"
 - Choose "Start in production mode"
 - Select a location closest to your users (usually the default is fine)
 - Click "Enable"
3. Set up the initial security rules for your database:
 - Click on the "Rules" tab in Firestore
 - Update your rules to allow authenticated users to access their own data:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
  }
}
```

4. Click "Publish" to save these rules

4. Verify Your Firebase Config

1. Click on the gear icon  next to "Project Overview" in the left sidebar
2. Select "Project settings"
3. Scroll down to the "Your apps" section
4. If you already have a web app, click on the web app (`</>` icon)

- If not, click "Add app" and select the web platform (</> icon)
- Enter a nickname for your app (e.g., "My Fitness Website")
- Check "Also set up Firebase Hosting" if you plan to use it
- Click "Register app"


5. You'll see your Firebase configuration. It looks like this:

```
const firebaseConfig = {  
  apiKey: "YOUR-API-KEY-HERE",  
  authDomain: "your-project-id.firebaseio.com",  
  projectId: "your-project-id",  
  storageBucket: "your-project-id.appspot.com",  
  messagingSenderId: "YOUR-MESSAGING-SENDER-ID",  
  appId: "YOUR-APP-ID",  
  measurementId: "YOUR-MEASUREMENT-ID"  
};
```

! **REQUIRED INPUT:** Make note of this configuration - you'll need it later when updating your code.


Environment Setup

Install Node.js and NPM

 **BEGINNER NOTE:** Node.js is a JavaScript runtime that allows you to run JavaScript on your computer outside of a browser. NPM (Node Package Manager) is used to install JavaScript libraries that your project will use.

1. Download and install Node.js from <https://nodejs.org/> (choose the LTS version)
2. Open a command prompt (Windows) or terminal (Mac/Linux):
 - **Windows:** Press Win+R, type "cmd" and press Enter
 - **Mac:** Open the Terminal app from Applications > Utilities
 - **Linux:** Use your distribution's terminal application
3. Navigate to your website's directory using the `cd` command:

```
cd path/to/your/website
```

 **BEGINNER NOTE:** Replace "path/to/your/website" with the actual file path to your website folder. For example:

- Windows: `cd C:\Users\YourName\Documents\MyWebsite`
- Mac/Linux: `cd /Users/YourName/Documents/MyWebsite`

4. Verify Node.js and NPM are installed correctly by typing:

```
node -v  
npm -v
```


You should see version numbers displayed, such as:

```
v18.16.0  
9.5.1
```

Initialize NPM in Your Project

This creates a `package.json` file that tracks your project's dependencies.


```
npm init -y
```

 **BEGINNER NOTE:** The `-y` flag automatically answers "yes" to all questions during initialization. This creates a basic `package.json` file with default values that you can edit later if needed.

Install Required Dependencies

Now you'll install the necessary libraries for your React and Firebase integration:


```
npm install firebase@12.1.0 react react-dom
```

 **BEGINNER NOTE:** This command installs three packages:

- `firebase@12.1.0`: The Firebase SDK with version 12.1.0 specified
- `react`: The core React library
- `react-dom`: The package that helps React work with the browser's DOM

Next, install the development tools needed to build your React code:

```
npm install --save-dev webpack webpack-cli babel-loader @babel/core @babel/preset-env @babel/preset-react css-loader style-loader
```

 **BEGINNER NOTE:** These packages help convert ("transpile") your modern JavaScript and React code into code that runs in all browsers:

- `webpack`: Bundles all your code and assets together
- `babel-loader`, `@babel/core`, etc.: Converts modern JavaScript and JSX (React's syntax) into compatible code
- `css-loader`, `style-loader`: Helps include CSS in your bundle

Install Firebase CLI (Optional)

If you plan to deploy your site to Firebase Hosting, you'll need the Firebase Command Line Interface:

```
npm install -g firebase-tools@14.11.1
```

BEGINNER NOTE: The `-g` flag installs this package globally on your computer, so you can use it from any project.

After installation, authenticate your Firebase account:

```
firebase login
```

This will open a browser window asking you to sign in with your Google account that has access to your Firebase project.

Project Structure

You'll need to create several folders and files for your React components. This structure helps organize your code logically.

BEGINNER NOTE: This folder structure separates your React code from the rest of your website. The `src` folder will contain your source code, and webpack will compile it into a bundle that your HTML page can use.

Create the following folder structure:

```
src/
├── react/
│   └── signup/
│       ├── components/
│       │   ├── AccountInfoStep.jsx
│       │   ├── ServiceTierStep.jsx
│       │   ├── PaymentStep.jsx
│       │   └── ConfirmationStep.jsx
│       ├── SignupForm.jsx
│       ├── index.js
│       └── signup.css
└── firebase-config.js
webpack.config.js
```


You can create these folders and files manually using your file explorer or with command line:

Windows Command Prompt:

```
mkdir src
mkdir src\react
mkdir src\react\signup
mkdir src\react\signup\components
type nul > src\react\signup\components\AccountInfoStep.jsx
type nul > src\react\signup\components\ServiceTierStep.jsx
type nul > src\react\signup\components\PaymentStep.jsx
type nul > src\react\signup\components\ConfirmationStep.jsx
type nul > src\react\signup\SignupForm.jsx
type nul > src\react\signup\index.js
type nul > src\react\signup\signup.css
type nul > src\firebase-config.js
type nul > webpack.config.js
```

Mac/Linux Terminal:

```
mkdir -p src/react/signup/components
touch src/react/signup/components/AccountInfoStep.jsx
touch src/react/signup/components/ServiceTierStep.jsx
touch src/react/signup/components/PaymentStep.jsx
touch src/react/signup/components/ConfirmationStep.jsx
touch src/react/signup/SignupForm.jsx
touch src/react/signup/index.js
touch src/react/signup/signup.css
touch src/firebase-config.js
touch webpack.config.js
```

 **BEGINNER NOTE:** These commands create empty files with the correct names in the right directories. You'll add content to them in the following sections.

Webpack Configuration


Webpack is a tool that bundles your JavaScript files together so they can be used in a browser.

Create a `webpack.config.js` file in your project root with this content:

```
// webpack.config.js
const path = require('path');

module.exports = {
  mode: 'development', // Change to 'production' for production builds
  entry: './src/react/signup/index.js',
  output: {
    filename: 'signup-bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
  module: {
    rules: [
```


```
{
  test: /\. (js|jsx) $/,
  exclude: /node_modules/,
  use: {
    loader: 'babel-loader',
    options: {
      presets: ['@babel/preset-env', '@babel/preset-react']
    }
  },
},
{
  test: /\.css$/,
  use: ['style-loader', 'css-loader']
}
]
},
resolve: {
  extensions: ['.js', '.jsx']
}
};
```

 **BEGINNER NOTE:** This configuration file tells webpack:

- Where to find your main React code (**entry**)
- Where to output the bundled JavaScript file (**output**)
- How to process different types of files (JavaScript, JSX, and CSS)
- Webpack will create a file called **signup-bundle.js** in a **dist** folder

Next, open your **package.json** file and add build scripts in the "scripts" section:


```
"scripts": {
  "build": "webpack --mode production",
  "dev": "webpack --mode development --watch"
}
```

 **BEGINNER NOTE:** These scripts allow you to:

- **npm run build:** Create an optimized production version of your code
- **npm run dev:** Create a development version and automatically rebuild when you make changes

Firestore Configuration

Now you'll create a shared Firestore configuration file that both your React components and existing code can use.

 **BEGINNER NOTE:** This file sets up Firestore for your React components using the latest and recommended practices. It creates functions to handle user authentication and store user data in Firestore.

Edit the **src/firestore-config.js** file:

```
// src/firebase-config.js

// CUSTOMIZE: Use your Firebase config values from the Firebase console
const firebaseConfig = {
  apiKey: "YOUR-API-KEY-HERE",
  authDomain: "your-project-id.firebaseio.com",
  projectId: "your-project-id",
  storageBucket: "your-project-id.appspot.com",
  messagingSenderId: "YOUR-MESSAGING-SENDER-ID",
  appId: "YOUR-APP-ID",
  measurementId: "YOUR-MEASUREMENT-ID"
};

// Use modern modular Firebase SDK
import { initializeApp } from 'firebase/app';
import {
  getAuth,
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  signInWithPopup,
  GoogleAuthProvider
} from 'firebase/auth';
import {
  getFirestore,
  doc,
  setDoc,
  serverTimestamp
} from 'firebase/firestore';

// Initialize Firebase for React components
let app;
try {
  app = initializeApp(firebaseConfig);
} catch (error) {
  // Handle the duplicate app initialization
  if (error.code === 'app/duplicate-app') {
    console.info('Firebase already initialized, using existing instance');
    app = initializeApp(undefined, 'default');
  } else {
    console.error("Firebase initialization error:", error);
  }
}

export const auth = getAuth(app);
export const db = getFirestore(app);

// Function to create a user with tier information (modern approach)
export async function createUserWithTier(email, password, name, phone, tier) {
  try {
    // Use modern Firebase SDK
    const userCredential = await createUserWithEmailAndPassword(auth, email,
password);
```



```
    const userId = userCredential.user.uid;

    await setDoc(doc(db, 'users', userId), {
      name: name,
      email: email,
      phone: phone || null,
      tier: tier,
      role: 'client',
      createdAt: serverTimestamp()
    });

    return { success: true, userId, user: userCredential.user };
  } catch (error) {
    console.error('Error creating user:', error);
    return {
      success: false,
      error: error
    };
  }
}

// Function to sign in a user (modern approach)
export async function signInUser(email, password) {
  try {
    const userCredential = await signInWithEmailAndPassword(auth, email,
password);
    return {
      success: true,
      user: userCredential.user
    };
  } catch (error) {
    console.error('Sign in error:', error);
    return {
      success: false,
      error: error
    };
  }
}

// Helper for Google sign-in
export async function signInWithGoogleAuth() {
  const provider = new GoogleAuthProvider();
  try {
    const result = await signInWithPopup(auth, provider);
    return result;
  } catch (error) {
    console.error('Google sign-in error:', error);
    throw error;
  }
}
```

! **REQUIRED INPUT:** Replace the `firebaseConfig` object with your own Firebase configuration that you copied from the Firebase console in the earlier step. It should look like this:

```
const firebaseConfig = {
  apiKey: "AIzaSyADF9yuram-pvlzjg6kBtdCk7LuK0M65tk", // Your actual API key will
  be different
  authDomain: "shreyfitweb.firebaseio.com",           // Your actual domain will
  be different
  projectId: "shreyfitweb",                             // Your actual project ID
  will be different
  storageBucket: "shreyfitweb.firebaseio.com",         // Your actual storage
  bucket will be different
  messagingSenderId: "1076359633281",                 // Your actual sender ID
  will be different
  appId: "1:1076359633281:web:3687e1675c9e185f0ab080", // Your actual app ID will
  be different
  measurementId: "G-5GBP19SXBW"                       // Your actual measurement
  ID will be different
};
```

 **BEGINNER NOTE:** This file:

1. Initializes Firebase with your configuration
2. Sets up authentication and Firestore database services
3. Provides helper functions to create users with their service tier selection
4. Handles sign-in functionality

You don't need to understand every line of code here, but it's important to include your own Firebase config values.

HTML Integration

Now you'll update your existing signup page to include a container for the React signup form.

Find your existing `signup.html` file and add a container div where React will mount:

! **REQUIRED INPUT:** Modify your existing `signup.html` file by adding the React container and bundle script:

```
<!-- Keep your existing header and structure -->
<section class="signup-section">
  <div class="container">
    <div class="section-header">
      <h2>Join SHREY.FIT</h2>
      <p>Create your account to start your fitness journey</p>
    </div>

    <!-- This is where React will mount - ADD THIS DIV -->
    <div id="react-signup-form"></div>
  </div>
</section>
```

```
<!-- Keep your existing Firebase scripts -->
<script src="https://www.gstatic.com/firebasejs/9.22.0/firebase-app-compat.js">
</script>
<script src="https://www.gstatic.com/firebasejs/9.22.0/firebase-auth-compat.js">
</script>
<script src="https://www.gstatic.com/firebasejs/9.22.0/firebase-firestore-
compat.js"></script>

<!-- Keep your existing auth.js script -->
<script src="js/auth.js"></script>

<!-- ADD THIS LINE at the end of your body section -->
<script src="dist/signup-bundle.js"></script>
```

BEGINNER NOTE: You're making two key changes:

1. Adding a div with id="react-signup-form" where your React component will appear
2. Adding a script tag to load your React bundle at the end of the file

React Components

Now you'll create the React components that make up your signup form. Each component handles a different step in the signup process.

Entry Point

Edit the `src/react/signup/index.js` file:

```
// src/react/signup/index.js
import React from 'react';
import ReactDOM from 'react-dom';
import { SignupForm } from './SignupForm';
import './signup.css';

// Wait for DOM to be ready, then mount React
document.addEventListener('DOMContentLoaded', () => {
  const container = document.getElementById('react-signup-form');
  if (container) {
    ReactDOM.render(<SignupForm />, container);
  } else {
    console.error('Could not find react-signup-form container element');
  }
});
```

BEGINNER NOTE: This file:

1. Imports React, ReactDOM, your main SignupForm component, and CSS
2. Waits for the HTML page to fully load
3. Finds the container div you added in your HTML

4. Renders the SignupForm component inside that div

Main Signup Form Component

Edit the `src/react/signup/SignupForm.jsx` file:

```
// src/react/signup/SignupForm.jsx
import React, { useState } from 'react';
import { createUserWithTier } from '../../firebase-config';
import AccountInfoStep from '../components/AccountInfoStep';
import ServiceTierStep from '../components/ServiceTierStep';
import PaymentStep from '../components/PaymentStep';
import ConfirmationStep from '../components/ConfirmationStep';

export function SignupForm() {
  const [currentStep, setCurrentStep] = useState(1);
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    phone: '',
    password: '',
    confirmPassword: '',
    tier: null,
    paymentInfo: null
  });
  const [error, setError] = useState('');
  const [isSubmitting, setIsSubmitting] = useState(false);

  const updateFormData = (data) => {
    setFormData({...formData, ...data});
    // Clear errors when form data changes
    setError('');
  };

  const nextStep = () => setCurrentStep(currentStep + 1);
  const prevStep = () => setCurrentStep(currentStep - 1);

  const handleSubmit = async () => {
    setIsSubmitting(true);
    setError('');

    try {
      // Create user with tier info
      const result = await createUserWithTier(
        formData.email,
        formData.password,
        formData.name,
        formData.phone,
        formData.tier
      );
    }

    if (!result.success) {
```

```
        throw new Error(result.error?.message || 'Failed to create account');
    }

    // Dispatch the same success event your current code looks for
    const signupSuccessEvent = new CustomEvent('signupSuccess', {
        detail: {
            email: formData.email,
            name: formData.name,
            autoLogin: true,
            tier: formData.tier
        }
    });
    document.dispatchEvent(signupSuccessEvent);

} catch (error) {
    console.error('Signup error:', error);
    setError(error.message);
    setIsSubmitting(false);
}
};

// Render the current step
const renderStep = () => {
    switch(currentStep) {
        case 1:
            return (
                <AccountInfoStep
                    formData={formData}
                    updateFormData={updateFormData}
                    nextStep={nextStep}
                    error={error}
                />
            );
        case 2:
            return (
                <ServiceTierStep
                    formData={formData}
                    updateFormData={updateFormData}
                    nextStep={nextStep}
                    prevStep={prevStep}
                    error={error}
                />
            );
        case 3:
            return (
                <PaymentStep
                    formData={formData}
                    updateFormData={updateFormData}
                    nextStep={nextStep}
                    prevStep={prevStep}
                    error={error}
                />
            );
        case 4:
```

```

        return (
          <ConfirmationStep
            formData={formData}
            handleSubmit={handleSubmit}
            prevStep={prevStep}
            isSubmitting={isSubmitting}
            error={error}
          />
        );
      default:
        return <AccountInfoStep
          formData={formData}
          updateFormData={updateFormData}
          nextStep={nextStep}
          error={error}
        />;
    }
  };

  return (
    <div className="signup-form-container">
      {/* Progress indicator */}
      <div className="progress-steps">
        <div className={`step ${currentStep} >= 1 ? 'active' : ''}`>
          <div className="step-number">1</div>
          <div className="step-label">Account</div>
        </div>
        <div className={`step-connector ${currentStep} >= 2 ? 'active' : ''}`>
</div>
        <div className={`step ${currentStep} >= 2 ? 'active' : ''}`>
          <div className="step-number">2</div>
          <div className="step-label">Select Plan</div>
        </div>
        <div className={`step-connector ${currentStep} >= 3 ? 'active' : ''}`>
</div>
        <div className={`step ${currentStep} >= 3 ? 'active' : ''}`>
          <div className="step-number">3</div>
          <div className="step-label">Payment</div>
        </div>
        <div className={`step-connector ${currentStep} >= 4 ? 'active' : ''}`>
</div>
        <div className={`step ${currentStep} >= 4 ? 'active' : ''}`>
          <div className="step-number">4</div>
          <div className="step-label">Confirm</div>
        </div>
      </div>

      {/* Current step */}
      {renderStep()}
    </div>
  );
}

export default SignupForm;

```

📖 BEGINNER NOTE: This file:

1. Manages the overall state of the signup form (which step is active, form data)
2. Handles form submission to Firebase
3. Shows a progress indicator for the multi-step process
4. Renders different components based on the current step

Step Components

Now you'll create each of the step components that make up your form.

Account Info Step

Edit the `src/react/signup/components/AccountInfoStep.jsx` file:

```
// src/react/signup/components/AccountInfoStep.jsx
import React from 'react';

function AccountInfoStep({ formData, updateFormData, nextStep, error }) {
  const handleChange = (e) => {
    updateFormData({ [e.target.name]: e.target.value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();

    // Form validation
    if (!formData.name || !formData.email || !formData.password ||
    !formData.confirmPassword) {
      updateFormData({ error: 'Please fill in all required fields' });
      return;
    }

    if (formData.password !== formData.confirmPassword) {
      updateFormData({ error: 'Passwords do not match' });
      return;
    }

    // Proceed to next step
    nextStep();
  };

  return (
    <form className="signup-form" onSubmit={handleSubmit}>
      <h3>Create Your Account</h3>

      {error && <div className="error-message">{error}</div>}

      <div className="form-group">
        <label htmlFor="fullname">
```

```

        <i className="fas fa-user"></i> Full Name <span className="required">*
</span>
    </label>
    <input
      type="text"
      id="fullname"
      name="name"
      value={formData.name}
      onChange={handleChange}
      placeholder="Enter your full name"
      required
    />
  </div>

  <div className="form-group">
    <label htmlFor="email">
      <i className="fas fa-envelope"></i> Email Address <span
className="required">*</span>
    </label>
    <input
      type="email"
      id="email"
      name="email"
      value={formData.email}
      onChange={handleChange}
      placeholder="Enter your email address"
      required
    />
  </div>

  <div className="form-group">
    <label htmlFor="phone">
      <i className="fas fa-phone"></i> Phone Number
    </label>
    <input
      type="tel"
      id="phone"
      name="phone"
      value={formData.phone}
      onChange={handleChange}
      placeholder="Enter your phone number"
    />
  </div>

  <div className="form-group">
    <label htmlFor="password">
      <i className="fas fa-lock"></i> Password <span className="required">*
</span>
    </label>
    <input
      type="password"
      id="password"
      name="password"
      value={formData.password}

```



```

        onChange={handleChange}
        placeholder="Create a password"
        required
      />
    </div>

    <div className="form-group">
      <label htmlFor="confirmPassword">
        <i className="fas fa-lock"></i> Confirm Password <span
className="required">*</span>
      </label>
      <input
        type="password"
        id="confirmPassword"
        name="confirmPassword"
        value={formData.confirmPassword}
        onChange={handleChange}
        placeholder="Confirm your password"
        required
      />
    </div>

    <button type="submit" className="btn-primary-enhanced">
      <i className="fas fa-arrow-right"></i>
      Continue
    </button>
  </form>
);
}

export default AccountInfoStep;

```

BEGINNER NOTE: This component:

1. Collects basic user information (name, email, phone, password)
2. Validates that required fields are filled and passwords match
3. Lets the user proceed to the next step when the form is valid

Service Tier Step

Edit the `src/react/signup/components/ServiceTierStep.jsx` file:

```

// src/react/signup/components/ServiceTierStep.jsx
import React from 'react';

function ServiceTierStep({ formData, updateFormData, nextStep, prevStep, error })
{
  // CUSTOMIZE: Match your service tiers from services.html
  const tiers = [
    {
      id: 'in-person-training',

```

```

    title: 'In-Person Training',
    price: '$70/session',
    description: 'Expert in-person coaching sessions focused on technique, form,
and effective workouts.',
    details: 'Seattle Area Only'
  },
  {
    id: 'online-coaching',
    title: 'Online Coaching',
    price: '$199/month',
    description: 'Complete transformation system with custom programs, nutrition
guidance, and support.',
    details: 'Remote Coaching'
  },
  {
    id: 'complete-transformation',
    title: 'Complete Transformation',
    price: '$199/month + $60/session',
    description: 'Comprehensive coaching plus hands-on training for maximum
results.',
    details: 'Seattle Premium Experience'
  }
];

const handleTierSelect = (tier) => {
  updateFormData({ tier });
};

const handleSubmit = (e) => {
  e.preventDefault();

  if (!formData.tier) {
    updateFormData({ error: 'Please select a service tier' });
    return;
  }

  nextStep();
};

return (
  <form className="signup-form" onSubmit={handleSubmit}>
    <h3>Select Your Service Tier</h3>

    {error && <div className="error-message">{error}</div>}

    <p className="form-description">
      Choose the service that best fits your fitness goals and preferences.
    </p>

    <div className="service-tiers">
      {tiers.map(tier => (
        <div
          key={tier.id}
          className={`service-tier ${formData.tier === tier.id ? 'selected' :

```

```

    `}``}

    onClick={() => handleTierSelect(tier.id)}
  >
  <div className="tier-header">
    <input
      type="radio"
      name="tier"
      id={tier.id}
      className="tier-radio"
      checked={formData.tier === tier.id}
      onChange={() => handleTierSelect(tier.id)}
    />
    <h4 className="tier-title">{tier.title}</h4>
  </div>
  <div className="tier-price">{tier.price}</div>
  <p className="tier-description">{tier.description}</p>
  <span className="tier-details">{tier.details}</span>
</div>
  )})
</div>

<div className="button-row">
  <button
    type="button"
    className="btn-secondary"
    onClick={prevStep}
  >
    <i className="fas fa-arrow-left"></i> Back
  </button>
  <button type="submit" className="btn-primary-enhanced">
    Continue <i className="fas fa-arrow-right"></i>
  </button>
</div>
</form>
);
}

export default ServiceTierStep;

```

BEGINNER NOTE: This component:

1. Presents service tier options for the user to select
2. Lets the user choose one tier by clicking on it
3. Provides navigation buttons to go back or continue to the next step
4. **! REQUIRED INPUT:** You should update the `tiers` array with your actual service tiers, prices, and descriptions

Payment Step

Edit the `src/react/signup/components/PaymentStep.jsx` file:

```
// src/react/signup/components/PaymentStep.jsx
import React, { useState } from 'react';

function PaymentStep({ formData, updateFormData, nextStep, prevStep, error }) {
  // This is a simplified payment form
  // In production, you would integrate with Stripe or another payment processor

  const [cardInfo, setCardInfo] = useState({
    cardNumber: '',
    cardName: '',
    expiry: '',
    cvv: ''
  });

  const handleChange = (e) => {
    setCardInfo({
      ...cardInfo,
      [e.target.name]: e.target.value
    });
  };

  const handleSubmit = (e) => {
    e.preventDefault();

    // Basic validation
    if (!cardInfo.cardNumber || !cardInfo.cardName || !cardInfo.expiry ||
!cardInfo.cvv) {
      updateFormData({ error: 'Please complete all payment fields' });
      return;
    }

    // In a real app, you would process payment here
    // For this example, we'll just collect the info and store it in formData
    updateFormData({
      paymentInfo: {
        ...cardInfo,
        // Mask card number for security in state
        cardNumber: '**** *' + cardInfo.cardNumber.slice(-4)
      }
    });

    nextStep();
  };

  // Calculate pricing based on selected tier
  const getTierPricing = () => {
    switch(formData.tier) {
      case 'in-person-training':
        return {
          name: 'In-Person Training',
          price: '$70.00',
          recurring: false,
          frequency: 'per session'
        }
    }
  }
}
```

```

    };
    case 'online-coaching':
      return {
        name: 'Online Coaching',
        price: '$199.00',
        recurring: true,
        frequency: 'monthly'
      };
    case 'complete-transformation':
      return {
        name: 'Complete Transformation',
        price: '$199.00',
        recurring: true,
        frequency: 'monthly + $60 per session'
      };
    default:
      return {
        name: 'Selected Plan',
        price: '$0.00',
        recurring: false,
        frequency: ''
      };
  }
};

const pricingInfo = getTierPricing();

return (
  <form className="signup-form" onSubmit={handleSubmit}>
    <h3>Payment Information</h3>

    {error && <div className="error-message">{error}</div>}

    <div className="payment-summary">
      <h4>Order Summary</h4>
      <div className="summary-item">
        <span className="item-name">{pricingInfo.name}</span>
        <span className="item-price">{pricingInfo.price}</span>
      </div>
      {pricingInfo.recurring && (
        <p className="recurring-notice">
          You will be charged {pricingInfo.price} {pricingInfo.frequency}
        </p>
      )}
    </div>

    <div className="form-group">
      <label htmlFor="cardName">
        <i className="fas fa-user"></i> Name on Card <span
className="required">*</span>
      </label>
      <input
        type="text"
        id="cardName"

```

```

        name="cardName"
        value={cardInfo.cardName}
        onChange={handleChange}
        placeholder="Name as it appears on card"
        required
      />
    </div>

    <div className="form-group">
      <label htmlFor="cardNumber">
        <i className="fas fa-credit-card"></i> Card Number <span
className="required">*</span>
      </label>
      <input
        type="text"
        id="cardNumber"
        name="cardNumber"
        value={cardInfo.cardNumber}
        onChange={handleChange}
        placeholder="XXXX XXXX XXXX XXXX"
        maxLength={19}
        required
      />
    </div>

    <div className="form-row">
      <div className="form-group half">
        <label htmlFor="expiry">
          <i className="fas fa-calendar"></i> Expiry Date <span
className="required">*</span>
        </label>
        <input
          type="text"
          id="expiry"
          name="expiry"
          value={cardInfo.expiry}
          onChange={handleChange}
          placeholder="MM/YY"
          maxLength={5}
          required
        />
      </div>
      <div className="form-group half">
        <label htmlFor="cvv">
          <i className="fas fa-lock"></i> CVV <span className="required">*
</span>
        </label>
        <input
          type="text"
          id="cvv"
          name="cvv"
          value={cardInfo.cvv}
          onChange={handleChange}
          placeholder="123"

```

```

        maxLength={4}
        required
      />
    </div>
  </div>

  <div className="button-row">
    <button
      type="button"
      className="btn-secondary"
      onClick={prevStep}
    >
      <i className="fas fa-arrow-left"></i> Back
    </button>
    <button type="submit" className="btn-primary-enhanced">
      Review Order <i className="fas fa-arrow-right"></i>
    </button>
  </div>
</form>
);
}

export default PaymentStep;

```

BEGINNER NOTE: This component:

1. Shows a summary of the selected plan and price
2. Collects credit card information
3. Validates that all required payment fields are filled
4. In a real application, you would integrate with a payment processor like Stripe
5. **! REQUIRED INPUT:** You should update the `getTierPricing` function to match your actual pricing structure

Confirmation Step

Edit the `src/react/signup/components/ConfirmationStep.jsx` file:

```

// src/react/signup/components/ConfirmationStep.jsx
import React from 'react';

function ConfirmationStep({ formData, handleSubmit, prevStep, isSubmitting, error }) {
  // Format selected tier for display
  const formatTier = (tierId) => {
    switch(tierId) {
      case 'in-person-training':
        return {
          name: 'In-Person Training',
          price: '$70 per session'
        };
      case 'online-coaching':

```

```

    return {
      name: 'Online Coaching',
      price: '$199 per month'
    };
  case 'complete-transformation':
    return {
      name: 'Complete Transformation',
      price: '$199 per month + $60 per training session'
    };
  default:
    return {
      name: 'Unknown tier',
      price: 'Price not available'
    };
}
};

const tierInfo = formatTier(formData.tier);

return (
  <div className="signup-form">
    <h3>Review and Confirm</h3>

    {error && <div className="error-message">{error}</div>}

    <div className="review-section">
      <h4>Account Information</h4>
      <div className="review-item">
        <span className="item-label">Name:</span>
        <span className="item-value">{formData.name}</span>
      </div>
      <div className="review-item">
        <span className="item-label">Email:</span>
        <span className="item-value">{formData.email}</span>
      </div>
      {formData.phone && (
        <div className="review-item">
          <span className="item-label">Phone:</span>
          <span className="item-value">{formData.phone}</span>
        </div>
      )}
    </div>

    <div className="review-section">
      <h4>Selected Plan</h4>
      <div className="review-item">
        <span className="item-label">Service:</span>
        <span className="item-value">{tierInfo.name}</span>
      </div>
      <div className="review-item">
        <span className="item-label">Price:</span>
        <span className="item-value">{tierInfo.price}</span>
      </div>
    </div>
  </div>

```



```

    <div className="review-section">
      <h4>Payment Information</h4>
      <div className="review-item">
        <span className="item-label">Card:</span>
        <span className="item-value">{formData.paymentInfo?.cardNumber || 'Not
provided'}</span>
      </div>
    </div>

    <div className="terms-agreement">
      <label className="checkbox-container">
        <input type="checkbox" required />
        <span className="checkmark"></span>
        I agree to the <a href="#" target="_blank">Terms of Service</a> and <a
href="#" target="_blank">Privacy Policy</a>
      </label>
    </div>

    <div className="next-steps-info">
      <h4>What happens next?</h4>
      <p>
        After completing signup, you will be directed to your dashboard where
you can schedule your 30-minute consultation to discuss your fitness goals and
create a personalized plan.
      </p>
    </div>

    <div className="button-row">
      <button
        type="button"
        className="btn-secondary"
        onClick={prevStep}
        disabled={isSubmitting}
      >
        <i className="fas fa-arrow-left"></i> Back
      </button>
      <button
        type="button"
        className="btn-primary-enhanced"
        onClick={handleSubmit}
        disabled={isSubmitting}
      >
        {isSubmitting ? 'Creating Account...' : 'Complete Signup'}
        {!isSubmitting && <i className="fas fa-check"></i>}
      </button>
    </div>
  </div>
);
}

export default ConfirmationStep;

```

BEGINNER NOTE: This component:

1. Shows a summary of all the information collected in previous steps
2. Lets the user review their account details, selected plan, and payment information
3. Includes terms and conditions agreement
4. Handles the final submission process
5. **! REQUIRED INPUT:** You should update the `formatTier` function to match your actual service tiers and pricing

Styling

Now you'll add CSS styles for your React components. These styles will only apply to the React part of your site, not affecting your existing styles.

Edit the `src/react/signup/signup.css` file:

```
/* src/react/signup/signup.css */
/* Progress steps */
.progress-steps {
  display: flex;
  align-items: center;
  margin-bottom: 2rem;
  justify-content: center;
}

.step {
  display: flex;
  flex-direction: column;
  align-items: center;
}

.step-number {
  width: 32px;
  height: 32px;
  border-radius: 50%;
  background-color: #e0e0e0;
  display: flex;
  align-items: center;
  justify-content: center;
  font-weight: 600;
  margin-bottom: 5px;
}

.step.active .step-number {
  background-color: var(--primary, #4caf50);
  color: white;
}

.step-label {
  font-size: 14px;
  color: #666;
}
```

```
.step.active .step-label {
  color: var(--primary, #4caf50);
  font-weight: 600;
}

.step-connector {
  height: 2px;
  flex-grow: 1;
  background-color: #e0e0e0;
  margin: 0 10px;
  margin-bottom: 20px;
  min-width: 40px;
}

.step-connector.active {
  background-color: var(--primary, #4caf50);
}

/* Form styling */
.signup-form {
  max-width: 600px;
  margin: 0 auto;
  padding: 2rem;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  background-color: white;
}

.service-tiers {
  display: flex;
  flex-wrap: wrap;
  gap: 1rem;
  margin: 2rem 0;
}

.service-tier {
  border: 2px solid #e0e0e0;
  border-radius: 8px;
  padding: 1.5rem;
  flex: 1;
  min-width: 250px;
  cursor: pointer;
  transition: all 0.3s ease;
}

.service-tier.selected {
  border-color: var(--primary, #4caf50);
  background-color: rgba(76, 175, 80, 0.05);
}

.tier-header {
  display: flex;
  align-items: center;
```

```
    margin-bottom: 1rem;
  }

  .tier-radio {
    margin-right: 10px;
  }

  .tier-title {
    font-weight: 600;
    font-size: 1.2rem;
    margin: 0;
  }

  .tier-price {
    font-weight: 700;
    font-size: 1.5rem;
    margin: 1rem 0;
    color: var(--primary-dark, #388e3c);
  }

  .tier-description {
    color: #666;
    margin-bottom: 1rem;
  }

  /* Button row */
  .button-row {
    display: flex;
    justify-content: space-between;
    margin-top: 2rem;
  }

  .btn-secondary {
    background-color: transparent;
    color: var(--primary, #4caf50);
    border: 1px solid var(--primary, #4caf50);
  }

  .error-message {
    background-color: #ffebee;
    color: #d32f2f;
    padding: 10px;
    border-radius: 4px;
    margin-bottom: 1rem;
    border-left: 4px solid #d32f2f;
  }

  /* Review step styling */
  .review-section {
    margin-bottom: 1.5rem;
    padding-bottom: 1rem;
    border-bottom: 1px solid #eee;
  }
```

```
.review-section h4 {
  margin-bottom: 0.5rem;
  color: #333;
}

.review-item {
  display: flex;
  justify-content: space-between;
  margin-bottom: 0.5rem;
}

.item-label {
  font-weight: 500;
  color: #666;
}

.item-value {
  color: #333;
}

.payment-summary {
  background-color: #f9f9f9;
  border-radius: 4px;
  padding: 1rem;
  margin-bottom: 1.5rem;
}

.summary-item {
  display: flex;
  justify-content: space-between;
  font-weight: 600;
}

.recurring-notice {
  margin-top: 0.5rem;
  font-size: 0.9rem;
  color: #666;
}

/* Form row for splitting inputs */
.form-row {
  display: flex;
  gap: 1rem;
}

.form-group.half {
  flex: 1;
}

/* Terms agreement styling */
.terms-agreement {
  margin: 1.5rem 0;
}
```

```
.checkbox-container {  
  display: flex;  
  align-items: center;  
}  
  
.next-steps-info {  
  background-color: #e8f5e9;  
  padding: 1rem;  
  border-radius: 4px;  
  margin: 1rem 0;  
}
```

BEGINNER NOTE: This CSS file:

1. Styles the multi-step progress indicator
2. Formats the service tier selection cards
3. Styles form elements, buttons, and error messages
4. Creates a consistent look for the review page

! REQUIRED INPUT: You might need to adjust some colors to match your site's theme. Look for variables like `var(--primary, #4caf50)` which use your site's CSS variables if they exist, or fall back to a default green color.

Building and Testing

Now that you've created all the necessary files, it's time to build your React signup form and test it.

Step 1: Build the React Component

Open a command prompt or terminal in your project directory and run:

```
npm run dev
```

BEGINNER NOTE: This command:

1. Starts webpack in watch mode
2. Compiles your React code
3. Creates the `dist/signup-bundle.js` file
4. Automatically updates the bundle when you make changes to your code
5. The terminal window will keep running - this is normal! It's watching for changes.

If you see any errors during the build process, review them carefully as they usually provide hints about what needs to be fixed.

Step 2: Test Your Implementation

1. Open your `signup.html` page in a web browser

- If you're using a local development server, navigate to your signup page (e.g., `http://localhost:3000/signup.html`)
 - If not, you can open the HTML file directly from your file system (e.g., `file:///C:/Users/YourName/Documents/MyWebsite/signup.html`)
2. You should see your new React-based signup form appear in the designated container
 3. Test each step of the signup process:
 - Fill out the account information form and click Continue
 - Select a service tier and click Continue
 - Enter test payment information and click Review Order
 - Review the confirmation page and click Complete Signup
 4. If everything is working correctly, your new user should be created in Firebase

BEGINNER NOTE: If your signup page appears empty where the React component should be, open your browser's developer console (F12 or right-click > Inspect) and look for any error messages that might help you troubleshoot.

Step 3: Prepare for Production

When you're ready to deploy your changes to a production environment, build an optimized version:

```
npm run build
```

This creates a minified, production-ready bundle in the `dist` folder.

Troubleshooting Common Issues

Here are solutions for common issues you might encounter:

1. "React element not appearing"

If your React component doesn't appear on the page:

- Check if the `dist/signup-bundle.js` file was created successfully
- Verify that you added the correct `<div id="react-signup-form"></div>` container in your HTML
- Check that the script tag `<script src="dist/signup-bundle.js"></script>` is included at the end of your HTML
- Open your browser console (F12) to see any JavaScript errors

2. "Firebase initialization errors"

If you see Firebase errors in the console:

- Ensure your Firebase configuration in `src/firebase-config.js` matches exactly what's in the Firebase console
- Check that you have the correct Firebase scripts included in your HTML

- Verify that the Firebase services you need (Authentication, Firestore) are enabled in the Firebase console

3. "Webpack build errors"

If webpack fails to build your bundle:

- Make sure all required dependencies are installed
- Check that your file paths are correct (case-sensitive)
- Verify that your JavaScript syntax is correct in all files
- Ensure your React components are properly structured with correct import/export statements

4. "Form submission not working"

If the form doesn't submit successfully:

- Check the Firebase security rules to ensure they allow writes to the users collection
- Make sure your user authentication is set up correctly in Firebase
- Look for specific error messages in the console when the form is submitted
- Verify that the form validation is passing correctly

5. "CSS styles not applying"

If your styles aren't showing up:

- Ensure that the CSS import in `index.js` is correct
- Check that webpack is properly processing CSS files
- Verify that your CSS selectors match the class names in your components
- Inspect the elements in your browser dev tools to see which styles are being applied

Next Steps

Now that you've successfully implemented a React signup form with Firebase integration, here are some ways to improve and extend your implementation:

1. Enhance User Experience

- Add form validation feedback as the user types
- Implement password strength indicators
- Add loading spinners during form submission
- Create smooth transitions between form steps

2. Improve Security

- Implement proper security rules in your Firebase Firestore
- Add CAPTCHA or other anti-bot measures
- Consider using environment variables for Firebase configuration

3. Extend Functionality

- Add social media login options (Google, Facebook, etc.)
- Implement email verification

- Create a "forgot password" feature
- Add analytics to track conversion rates

4. Convert More Pages to React

Once you're comfortable with this hybrid approach, consider converting other pages:

1. Login page
2. User dashboard
3. Account settings

5. Learn More React

To deepen your understanding of React:

- Explore React hooks in more depth (useEffect, useContext, etc.)
- Learn about state management solutions like Redux or Context API
- Study React Router for multi-page applications
- Practice creating reusable components

By completing this guide, you've taken a significant step toward modernizing your website while maintaining compatibility with your existing code. This hybrid approach allows you to gradually adopt React at your own pace, focusing on the components that benefit most from React's capabilities.

Remember that Firebase fully supports this incremental approach - you don't need to convert your entire application at once, and both your React components and traditional pages can work with the same Firebase project and data.