

WA_clone-UI Design and Routing

Agenda

- WhatsApp Demo
- Requirement Analysis
- routing in application
- firebase setup

Demo

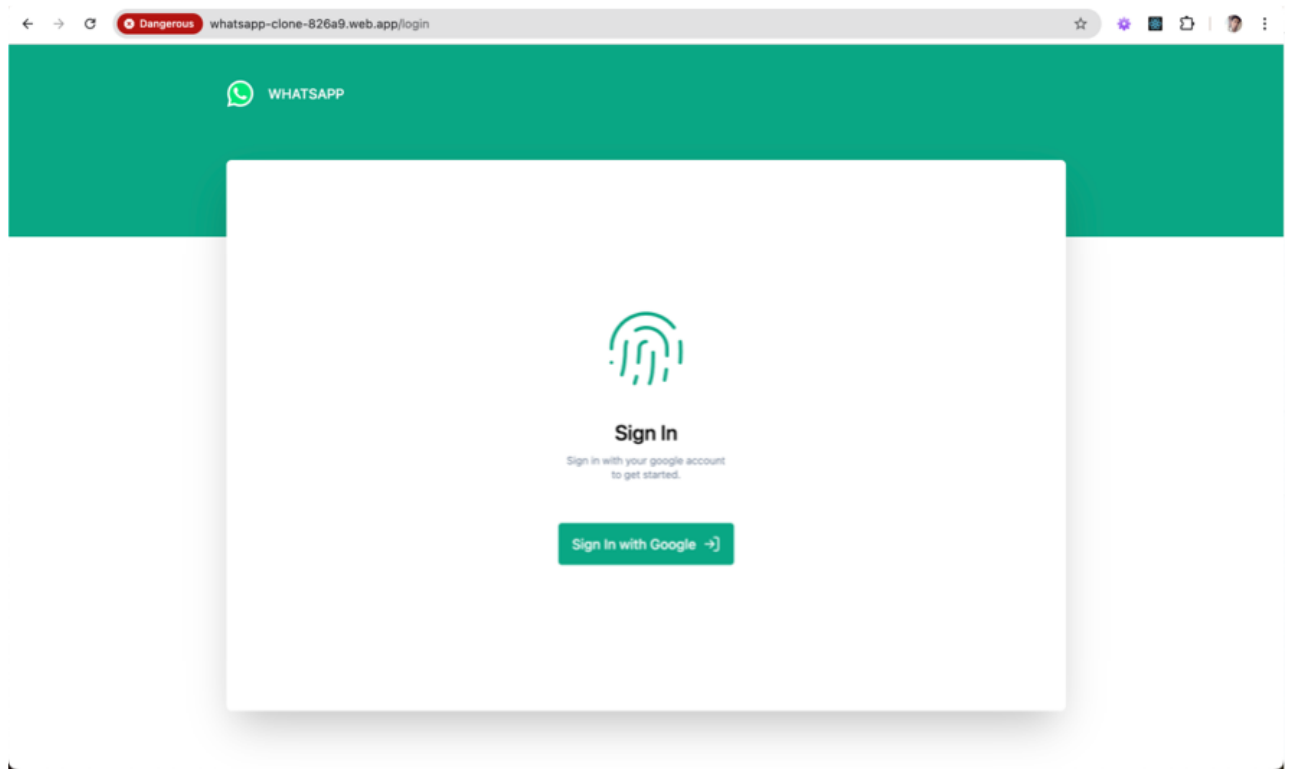
Features

We will be building Whatsapp Clone with the following features

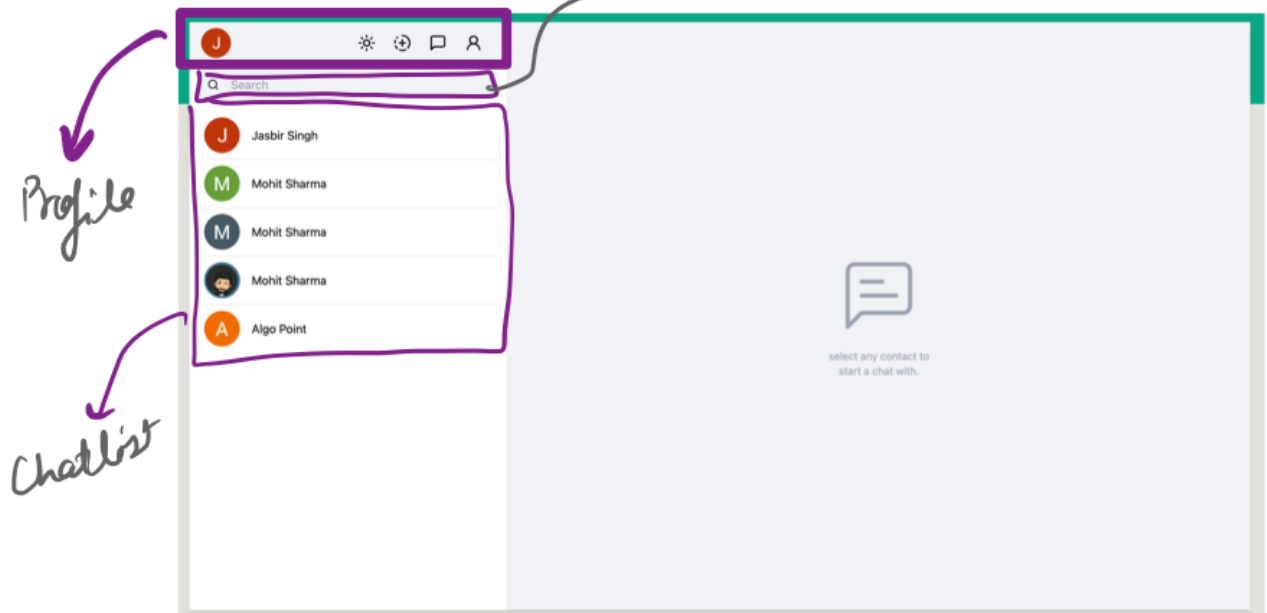
- Google login and auth
- Profile
- Chat list
- last seen
- chat with the message and the time
- UX -> user cues and Hover

Views : screens

① login




② Protected homepage search



③ Profile

[←](#) Profile



Your name

Jasbir Singh

✓

Status

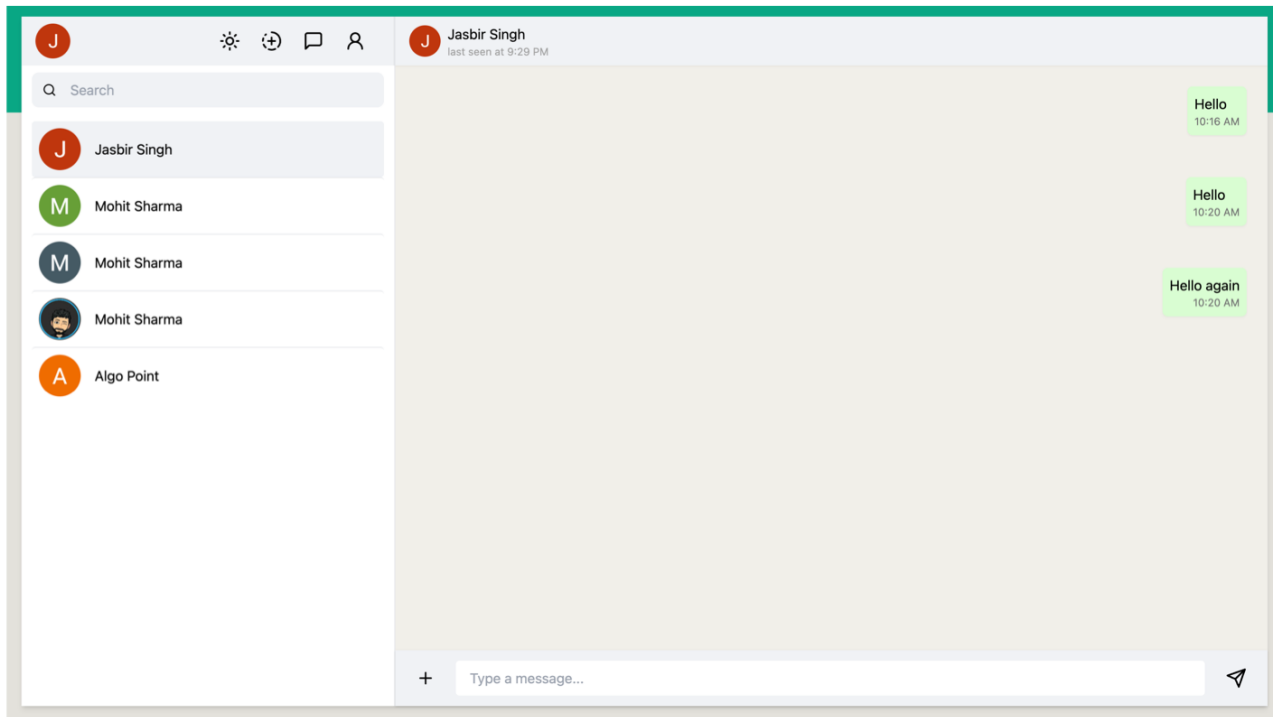
Update your status...

✓

Logout



Chat screen →



Requirements

Tech Stack

- React
- React-router-dom
- Firebase
 - Firebase auth
 - Firebase : realtime database
 - Firebase : Storage
 - Firebase hosting

User discovery : How will user discover our application

- They will have a link to the app . whenever user Login it will be shown in the left chat screen -> all the users will be visible to them and if clicked will show the history

Routes

- Login -> `/login`
- Protected Route Home -> `/` & if not loggedIN-> redirect to login
- Chat -> `/uniqueId`
- 404 -> if not the above url -> take the user to 404 page not found

Features

- Ui screens
 - ☐ Login -> google login
 - ☐ Home screen
 - ☐ Left sidebar (top section)
 - ☐ Top left profile
 - ☐ Profile page
 - ☐ details ,
 - ☐ status
 - ☐ logout button
 - ☐ DP-> (image upload)
 - ☐ Top Right section
 - ☐ 4 static icons only one is working with dark and light mode
 - ☐ Search -> name search
 - ☐ ChatList
 - ☐ user cue on -> Hover & Selected
 - ☐ User chat
 - ☐ Thumbnail
 - ☐ Name
 - ☐ Chat screen
 - ☐ Top Bar -> user chat
 - ☐ Last seen
 - ☐ Thumbnail
 - ☐ Name
 - ☐ chat screen
 - ☐ left right orientation of message
 - ☐ type the message and click to send

setup of react application : Refer the last session note to know how to setup the react application using vite

Routing

To match our view we will be having following screens

- HomePage
 - Profile
 - PageNotFound
 - chat : unique url
- out of these Login and PageNotFound is open for all
Profile , chat , HomePage should only be allowed to access when user is loggedIn.

Let's see how we can provide Routing in react application

Routing tutorial

In this tutorial, we will cover the basics of frontend routing in a React application using `react-router-dom`. Frontend routing allows you to create a single-page application with multiple views. Instead of navigating to different pages, your application can dynamically display different components based on the URL.

Setting Up React Router DOM

To get started with `react-router-dom`, you first need to install it in your React project.

Install `react-router-dom` using npm:

```
npm install react-router-dom
```

Components

Create the components that will be rendered based on the routes.

Home.jsx

```
import React from 'react';

function Home() {
  return <div>Home</div>;
}
```

```
export default Home;
```

Login.jsx

```
import React from 'react';

function Login() {
  return (
    <>
      <div>
        
        <div>WhatsApp</div>
      </div>
      <div className="white"></div>
      <div>
        <div>Sign In</div>
        <div>Sign in with your Google account to get started.</div>
        <button>Sign in with Google</button>
      </div>
    </>
  );
}

export default Login;
```

Chat.jsx

```
import React from 'react';
import { useParams } from 'react-router-dom';

function Chat() {
  const params = useParams();
  return <h2>Chat: {params.uniqueId}</h2>;
}

export default Chat;
```

PageNotFound.jsx

```
import React from 'react';

function PageNotFound() {
  return <div>Page Not Found</div>;
}

export default PageNotFound;
```

Setting Up BrowserRouter

In your `main.jsx` file, wrap the `App` component with `BrowserRouter` to enable routing in your application.

main.jsx

```
import ReactDOM from 'react-dom/client';
import App from './App.jsx';
import './index.css';
import { BrowserRouter } from 'react-router-dom';

ReactDOM.createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

Using routes to make do conditional routing

In your `App.jsx` file, define the routes for your application using `Routes` and `Route` components from `react-router-dom`.

App.jsx

```
import { Route, Routes } from 'react-router-dom';
import Login from './Components/Login';
import Chat from './Components/Chat';
import PageNotFound from './Components/PageNotFound';
import Home from './Components/Home';
```



```
function App() {
  return (
    <>
      <h1>WhatsApp Clone</h1>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/chat/:uniqueId" element={<Chat />} />
        <Route path="*" element={<PageNotFound />} />
      </Routes>
    </>
  );
}

export default App;
```

Explanation of Routes

- `<Route path="/" element={<Home />} />`: This route matches the root URL (`/`). When the user visits the root URL, the `Home` component is rendered.
- `<Route path="/login" element={<Login />} />`: This route matches the `/login` URL. When the user visits `/login`, the `Login` component is rendered.
- `<Route path="/chat/:uniqueId" element={<Chat />} />`: This route matches URLs like `/chat/123`, where `123` is a dynamic parameter. The `Chat` component is rendered, and the dynamic part of the URL can be accessed via the `useParams` hook.
- `<Route path="*" element={<PageNotFound />} />`: This route matches any URL that doesn't match any of the above routes. The `PageNotFound` component is rendered for all undefined routes, providing a fallback for unmatched URLs.

Understanding `useParams` in React Router

`useParams` is a hook provided by `react-router-dom` that allows you to access the parameters of the current route. It's particularly useful for extracting dynamic segments from the URL, which can be used to display data specific to those parameters.

Use Case for `useParams`

The primary use case for `useParams` is when you need to access dynamic URL parameters in your component. For example, in a chat application, you might have a URL structure where each chat is accessed by a unique ID.

Example: Chat Component

Let's look at the `Chat` component in your example:

```
import React from 'react';
import { useParams } from 'react-router-dom';

function Chat() {
  const params = useParams();
  console.log(params); // This will log the dynamic parameters of the current route
  return <h2>Chat: {params.uniqueId}</h2>;
}

export default Chat;
```

How `useParams` Works

1. **Route Definition:** You define a route with a dynamic segment in `App.jsx`.

```
<Route path="/chat/:uniqueId" element={<Chat />} />
```

2. ****Accessing Parameters**:** In the `Chat` component, you use the `useParams` hook to access the `uniqueId` parameter from the URL.

```
``jsx
const params = useParams();
```

3. **Using Parameters:** The `params` object will contain all the dynamic segments defined in the route. In this case, `params.uniqueId` will give you the value of `uniqueId` from the URL.

Practical Example

Let's say you navigate to the URL `http://yourapp.com/chat/12345`.

- The `Chat` component will render because it matches the route `/chat/:uniqueId`.
- The `useParams` hook will extract the `uniqueId` parameter from the URL.
- `params.uniqueId` will be `12345`.
- The component will display `Chat: 12345`.

Note: We still need to figure out how to

Firestore

Firestore is a platform developed by Google that provides a suite of tools and services to help build and grow applications. It offers a wide range of features, from database management to authentication, making it a popular choice for developers.

5 Core Firestore Services

1. Firestore Authentication

- **Purpose:** Manages user authentication, allowing users to sign in using various methods like email/password, Google, Facebook, etc.
- **Benefits:** Provides secure user management, simplifies user onboarding, and enables features like password reset and email verification.

2. Firestore Realtime Database

- **Purpose:** A cloud-hosted NoSQL database that stores and syncs data in real-time.
- **Benefits:** Enables real-time updates, offline capabilities, and easy data synchronization across devices.

3. Firestore Storage

- **Purpose:** Stores and serves user-generated content such as images, videos, and audio files.
- **Benefits:** Provides scalable storage, integrates seamlessly with other Firestore services, and offers features like file management and security rules.

4. Firebase Hosting

- **Purpose:** Deploys and hosts static and dynamic web applications.
- **Benefits:** Offers fast content delivery, global CDN, custom domains, and easy deployment workflows.

Setting Up Firebase for React Users : Watch the last section of recording