QuickBytes System Architecture

System Requirements

Operating Environment:

- The system is assumed to run on a modern server OS (e.g. Linux, Windows) capable of running a Node.js backend and any OS that supports modern web browsers for the frontend.
- It is also assumed that there will be a real-time database, Firebase, which will be used for data storage and retrieval.
- For the network configuration, it is assumed that the system has a stand internet connection for communication between frontend and backend, and between backend and the Firebase services (such as authentication).
- A network connection between the client and Google Maps API is required.

Programming Language and Environment:

- The language primarily used is TypeScript
- The environment assumes a NodeJS runtime

Database:

- The backend relies on Firebase Realtime Database
- Assumes the database is configured and accessible in the environment upon deployment.

Firebase Services:

- Admin SDK credentials should be configured properly in firebase-config.ts for server side authentication and DB management.
- Client side credentials should be configured properly in firebaseConfig.ts for management and authentication on the frontend.

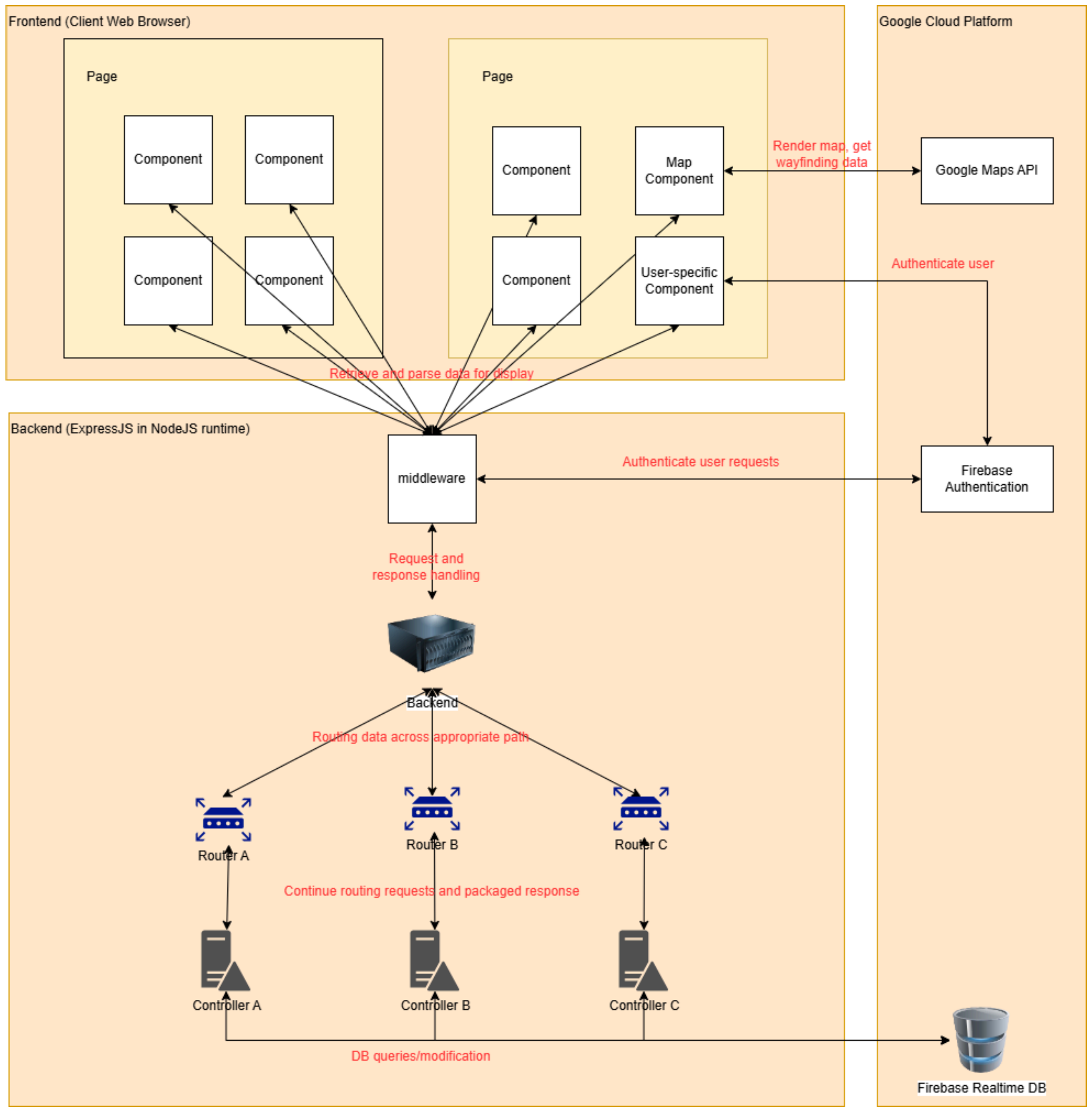
Network Configurations:

- The server should allow HTTPS requests to Firebase services and Google Maps API

Frameworks and Libraries:

- Backend includes: Express.js for handling HTTP requests, Firebase Admin for authentication and database interactions.
- Frontend includes: React for building user interfaces, Material UI for UI components and themes, React Query for data fetching and synchronization, React Router for routing in the react application.

System Diagram



System Decomposition

-- Client-side --

Components:

- **(General) component**: A regular React component which sends requests to backend and parses the response to render dynamic data for the user interface.
- **Maps component**: A special React component but sends requests to Google Maps API to load an interactive map and show wayfinding information.
- **User-specific component**: A React component which may send requests to Firebase authentication service for client-side session and authentication information.

Pages:

- **Pages**: A page combines multiple React components into a single entity that can be interacted with by the user as a coherent web page.

-- Server-side --

Backend Request Handling:

- **Middleware**: Abstracts backend functionalities to streamline frontend interactions.
- **Backend**: Processes frontend requests and communicates with Firebase.
- **Routers**: Ensures correct routing of requests to controllers based on their type.
- **Controllers**: Perform business logic by handling data retrieval/updates in Firebase.

continued…

Error and Exception Handling

- **Invalid user inputs**:
    - Frontend: Inputs are validated by restricting type and regex, and input that does not pass client-side validation will disable form submission.
    - Backend: In the case of inputs to the backend referring to non-existent entries in the database, a 400 or 404 error code should be returned. If sent from the frontend, the frontend should reflect this error with error text, pop ups, dialogs and/or animations.
- **System failures of API**:
    - Backend: Unexpected errors and exceptions such as type errors and bugs will return a response with the appropriate error status (500 - 599, but usually 500).
    - Frontend: If the error was during the processing of a request sent from the frontend, then the frontend should show error messages in the form of error text, pop ups, dialogs and/or animations
- **External System Failures**:
    - Firebase Downtime: Show cached data or appropriate error message
    - Google Maps API Error: Provide appropriate error message
- **Authentication Failures**:
    - Frontend: Redirects users to the login page and prevents unauthorized users from accessing protected pages.
    - Backend: Use tokens to validate authentication

CRC Card

Please refer to doc/sprint1/CRCcards.png