LinkSphere Detailed Technical Documentation

1. Project Overview

LinkSphere is a single-page React web application that enables users to create and share a customizable public profile page, similar to a LinkTree clone. The project integrates Firebase Authentication and Firestore Database to handle user registration, login, and persistent profile data. The frontend is fully developed using React.js, deployed on Vercel for seamless continuous deployment.

2. Folder and File Structure

The project follows a modular structure for maintainability: - /src: Contains all application source code. - /components: Holds reusable UI components like Auth, LogIn, ProfileDashboard, and PublicProfile. - /config: Contains Firebase configuration and initialization files. - /assets: Stores static assets such as icons or logos. - App.js: Entry point managing routing and navigation. - index.js: Renders the root React component to the DOM. This structure ensures separation of concerns and promotes scalability.

3. Routing and Navigation

React Router DOM handles navigation between pages. The main routes include Home, Login, Signup, Dashboard, and Public Profile. Dynamic routing is used for public profile URLs in the format /profile/:uid, where each unique ID corresponds to a user's Firestore document. Navigation is managed with and elements, while useNavigate() enables redirection after authentication events. To avoid 404 errors on refresh (a common issue in SPAs deployed on Vercel), a custom rewrite rule is applied in Vercel’s configuration to redirect all routes to index.html.

4. Authentication System

Firebase Authentication is used for secure user sign-up and login via email and password. When a user signs up, Firebase creates a unique UID which serves as their identity across the system. React’s useState hook tracks authentication state locally, and useEffect ensures persistence across page reloads. After login, users are redirected to their Dashboard. Sign-out clears authentication state and redirects to the login page. This setup ensures that private routes are protected and accessible only to authenticated users.

5. Firestore Database Integration

The Firestore NoSQL database stores each user's profile in a dedicated document under the 'userProfileCards' collection. Each document is keyed by the user’s Firebase UID, ensuring unique identification. Data such as username, bio, email, and social media links are saved using Firestore’s setDoc() method, while getDoc() retrieves existing profiles. The database schema is flexible and supports future extensions like profile images or custom themes. Error handling ensures data consistency and provides user feedback if operations fail.

6. Profile Creation and Dashboard Logic

The ProfileDashboard component serves as the core workspace for users. On loading, it fetches existing profile data from Firestore using the logged-in user’s UID. Each form field is bound to React state variables, ensuring real-time synchronization with user inputs. When the user saves changes, the application updates Firestore in a single atomic operation. A sharable public link is generated in the format /profile/:uid, which users can copy via a “Copy Link” button. Conditional rendering

ensures that the link appears only after the profile is saved successfully.

7. Public Profile Page

This component displays the public-facing version of the user's profile. When someone accesses a URL like /profile/abcd1234, the component extracts the UID from the route parameters using useParams(). It then queries Firestore to fetch corresponding profile details and displays them dynamically. If the UID does not exist, a "Profile not found" message is shown. This ensures a smooth, user-friendly experience for visitors viewing shared profiles.

8. State Management

The application uses local React state (via useState and useEffect) for managing component-specific data. Since the app's data flow is straightforward, a global state management library like Redux was not required. Authentication state is preserved in memory during a session and revalidated using Firebase's onAuthStateChanged listener. This approach maintains simplicity while ensuring reliable session handling.

9. User Interface Implementation

LinkSphere's interface follows a modern, minimalist, and responsive design. All UI components use CSS modules for styling, ensuring scope isolation and maintainability. SVG icons are embedded directly for visual consistency. Layouts use Flexbox for responsiveness, and spacing and alignment are controlled through utility classes. A dark theme palette enhances visual comfort and user appeal, aligning with contemporary UI trends.

10. Firebase Configuration and Environment Setup

Firebase is initialized in the config/firebase.js file using the project-specific credentials. These include the API key, project ID, and authentication domain. While the configuration is public by design (as required for frontend SDKs), sensitive operations remain secure through Firebase's backend verification. Environment variables are managed via Vite's .env file, ensuring local and production configurations remain separate.

11. Deployment on Vercel

The application is built using the Vite bundler and deployed directly to Vercel for production hosting. Vercel automatically detects the React build configuration and sets up continuous deployment from the connected GitHub repository. A custom rewrite rule in vercel.json ensures that all client-side routes redirect to index.html, preventing 404 errors during page refreshes. This setup offers a fast, globally distributed hosting solution with instant rollbacks and version control integration.

12. Common Issues and Debugging

- **404 on Refresh:** Fixed using Vercel's SPA rewrite configuration. - **Firestore Access Issues:** Occur when security rules restrict reads/writes; resolved by updating Firestore rules. - **Auth State Delay:** Handled using Firebase's onAuthStateChanged to ensure the UI waits for initialization. - **Data Not Updating:** Usually due to stale state; resolved by calling Firestore again after save operations. Proper console logging throughout ensures quick debugging and issue identification.

13. Future Enhancements

- Add profile picture uploads using Firebase Storage.

- Introduce user-selected themes and layout customizations.
- Integrate analytics to monitor public link visits.
- Add password reset and email verification workflows.
- Include OAuth-based social logins for faster sign-up.

14. Conclusion

LinkSphere exemplifies a clean, modular, and scalable architecture that leverages the synergy between React and Firebase. Each feature — from authentication to Firestore integration — is designed with clarity and simplicity, ensuring maintainability and extensibility. This technical structure makes LinkSphere a strong demonstration of real-world React + Firebase project development, ideal for showcasing full-stack skills in interviews or portfolio presentations.