Architecture requirements

AR (Architectural Requirements)

- **AR1 System Architecture Style**
    - AR1.1 **Progressive Web App (PWA) Architecture:** The system must be designed as a PWA to support core features like offline functionality and installability.
    - AR1.2 **Client-Server Architecture:** The system will likely follow a client-server model, with the React/TypeScript frontend acting as the client and the Python backend serving as the server.
    - AR1.3 (Consideration) **Modular Design:** The architecture must be modular to enable seamless integration of future datasets, features, and to allow for independent development and scaling of components. (Also listed as NFR2.2)
- **AR2 Backend Architecture**
    - AR2.1 **Scalable Backend:** The backend system must be designed for scalability to handle growth in users, data, and request loads.
        - AR2.1.1 The project proposal suggests using a cloud-based system (e.g., Firebase, Supabase) for centralized data storage and backend services. This choice should be confirmed and detailed.
    - AR2.2 **API Design:**
        - AR2.2.1 If APIs are exposed (e.g., for NLP researchers or future integrations), they should be designed following RESTful principles.
        - AR2.2.2 APIs should be well-documented.
- **AR3 Frontend Architecture**
    - AR3.1 **Component-Based Architecture:** Given the use of React, the frontend will follow a component-based architecture, promoting reusability and maintainability.

    - AR3.2 **State Management:** A clear strategy for state management (e.g., React Context, Redux, Zustand) must be chosen and documented.
- **AR4 Data Architecture**
    - AR4.1 **Data Storage:** Specify the chosen database solution (e.g., NoSQL like Firebase Firestore, SQL like PostgreSQL via Supabase, or other, based on project needs for structured/unstructured data and search).
    - AR4.2 **Data Flow:** Define how data flows between the frontend, backend, database, and any external services (e.g., for PWA offline storage, synchronization).
    - AR4.3 **Indexing for Search:** The architecture must support efficient indexing mechanisms for fast and effective searching across multilingual lexicons. (Also listed as NFR1.1)
    - AR4.4 **Data Synchronization Mechanism:** Define the architectural approach for synchronizing offline data with the central repository when the user is online.
- **AR5 Offline Support Architecture**
    - AR5.1 **Service Workers:** Utilize service workers for caching application assets and data to enable offline functionality.
    - AR5.2 **Local Storage/IndexedDB:** Specify how browser storage (e.g., IndexedDB) will be used for storing downloaded lexicons and user data for offline access.
- **AR6 Security Architecture**
    - AR6.1 **Authentication and Authorization:** Define the mechanisms for user authentication and how authorization will be handled to protect resources and user data.
    - AR6.2 **Data Security:** Outline measures for securing data both in transit (e.g., HTTPS) and at rest. (Also NFR5.1)
- **AR7 Architectural Patterns**
    - AR7.1 TODO
- **AR8 Design Patterns (to be documented in SRS as per Demo 1 Instructions)**

    - AR8.1 TODO
- **AR9 Quality Requirements (to be documented in SRS as per Demo 1 Instructions - these heavily influence architecture)**
    - AR9.1 **Performance:** The architecture must support fast load times, quick search responses, and efficient data handling.
    - AR9.2 **Reliability:** The system, especially data synchronization and offline access, must be reliable.
    - AR9.3 **Scalability:** The architecture must be able to scale to accommodate more users, data, and features over time.
    - AR9.4 **Security:** The architecture must incorporate security best practices to protect data and system integrity.
    - AR9.5 **Maintainability:** The architecture should promote ease of understanding, modification, and extension by current and future developers.