Write requirements for a nostr web app that can be clearly understood by Replit's AI coding agent (which uses Claude Sonnet 3.7). The app has the following functionality:

Three personas: 1. A credential issuing authority who has an administrator view. They can log in using their nostr keys (in npub/nsec format) and issue a NIP-58 badge as a credential for a healthcare provider (radiologist, cardiologist, acupuncturist, psychotherapist, nutritionist, etc)
2. A healthcare provider, who can log in using their nostr keys (npub/nsec format) and update their demographic information, request a credential (NIP-58 badge) if they don't have one
3. A user searching for doctors - they can log in using their nostr keys (npub/nsec format) and search for healthcare providers, click on their profiles, view their demographic information, and view their credentials to ensure they are indeed real doctors

The web app is essentially a directory for healthcare providers, where patients can search for the doctor they are looking for

Requirements for the Nostr Web App

Below is a detailed and clear set of requirements for the Nostr web app, tailored for Replit's AI coding agent (Claude Sonnet 3.7). The app will support three personas: Credential Issuing Authority, Healthcare Providers, and Users (patients). It will leverage Nostr keys (npub/nsec format) for authentication and implement NIP-58 badges as credentials.

General Requirements

1. **Authentication**:
    - All users (Credential Issuing Authority, Healthcare Providers, and Users) must log in using their Nostr keys (npub/nsec format).
    - Authentication should securely handle private keys (nsec) without exposing them to unauthorized parties.
2. **Database**:
    - A database to store:
        - User profiles (Credential Issuing Authority, Healthcare Providers, Users)
        - Healthcare provider demographic information
        - Issued credentials (NIP-58 badges)

    - Use a relational database (e.g., PostgreSQL) or a NoSQL database (e.g., MongoDB), depending on scalability needs.
3. **Frontend**:
    - A responsive user interface built with a modern framework like React or Vue.js.
    - Ensure accessibility and ease of use for all personas.
4. **Backend**:
    - A backend API built with Python (e.g., Flask or FastAPI) or Node.js to handle business logic.
    - Implement NIP-58 badge issuance and verification logic.
    - Use libraries/tools to interact with the Nostr protocol.
5. **Search Functionality**:
    - Allow users to search healthcare providers by name, specialty, or location.
    - Use a search library like Elasticsearch or implement full-text search in the database.
6. **Security**:
    - Encrypt sensitive data in transit and at rest.
    - Implement rate limiting and other measures to prevent abuse of the system.
    - Follow best practices for handling cryptographic keys.

Persona-Specific Requirements

1. Credential Issuing Authority

- **Login**:
    - Must log in using their Nostr keys (npub/nsec format).
- **Administrator Dashboard**:
    - View a list of all registered healthcare providers.
    - Search for healthcare providers by name or specialty.
    - Issue NIP-58 badges as credentials for healthcare providers.
        - Select the provider from a list or search by npub key.
        - Assign a specialty credential (e.g., radiologist, cardiologist).
        - Store the issued badge in the database and associate it with the provider's profile.
    - Revoke credentials if necessary.
- **Audit Logs**:
    - View logs of all issued/revoked credentials.

2. Healthcare Provider

- **Login**:
    - Must log in using their Nostr keys (npub/nsec format).
- **Profile Management**:
    - Update demographic information such as:
        - Name
        - Specialty
        - Location
        - Contact details
    - Upload profile pictures or documents if needed.
- **Credential Request**:
    - Request an NIP-58 badge if they do not have one.
        - Submit their specialty and any required details for verification.
        - View the status of their credential request (pending/approved/rejected).
- **View Credentials**:
    - See their issued credentials (NIP-58 badges) on their profile.

3. User Searching for Doctors

- **Login**:
    - Must log in using their Nostr keys (npub/nsec format).
- **Search for Healthcare Providers**:
    - Search by name, specialty, or location.
    - Filter results by specialty or proximity to a specific location.
- **View Profiles**:
    - Click on a healthcare provider's profile to view their details:
        - Demographic information
        - Credentials (NIP-58 badges)
        - Contact information
- **Credential Verification**:
    - Ensure that displayed credentials are valid and issued by the Credential Issuing Authority.

Technical Features

1. **NIP-58 Badge Implementation**:
   - Integrate functionality to issue and verify NIP-58 badges using the Nostr protocol.
   - Ensure badges are cryptographically signed by the Credential Issuing Authority's private key.
2. **Profile Privacy Settings**:
   - Allow healthcare providers to control what demographic information is visible to users.
3. **Search Optimization**:
   - Implement efficient indexing for fast search queries.
   - Support autocomplete suggestions for specialties and locations.
4. **User Notifications**:
   - Notify healthcare providers when their credential request is approved or rejected.
   - Notify users when new doctors are added in their area of interest.
5. **Audit Trails**:
   - Maintain logs of all actions performed by the Credential Issuing Authority for transparency.
6. **Scalability**:
   - Design the app to handle a growing number of users and healthcare providers efficiently.
7. **Error Handling & Validation**:
   - Validate all user inputs on both frontend and backend.
   - Provide meaningful error messages for invalid actions or inputs.

Suggested Tech Stack

1. Frontend: React.js or Vue.js
2. Backend: Python with FastAPI or Flask, or Node.js with Express
3. Database: PostgreSQL or MongoDB
4. Authentication: Integration with Nostr protocol libraries
5. Search: Elasticsearch or database-native full-text search

This specification provides clear instructions to build the desired functionality while ensuring security, usability, and scalability.