1. Main Components of the System (3 pts)

1.1 Frontend

- **Framework/Language**: React.js (or Angular/Vue as alternatives)
- **Reasoning**:

  For the frontend, we chose React.js because it allows us to build a responsive and accessible interface that works well across different devices. Since many ALICE users may rely on older phones or shared devices, accessibility and performance were major priorities. React's component-based structure makes it easier for us to maintain the project and add new features later without needing to rewrite everything. Another big advantage is the strong ecosystem around React, including libraries like Material UI and Tailwind CSS, which save us development time while still letting us customize the interface. While Angular or Vue could also work, we decided on React because it has the largest community support and resources available.

1.2 Backend

- **Framework/Language**: Node.js with Express.js
- **Reasoning**:

  For the backend, we decided on Node.js with Express.js. This combination is lightweight but powerful, making it a good fit for handling our API requests, user profile data, and eligibility logic. Node.js is especially useful for applications that need to scale and respond quickly to multiple users at once, which is important since our system will connect people to resources in real time. Express.js adds simple tools for managing routes and requests without unnecessary complexity. We also liked that Node.js has a large community, which means we can find solutions and integrations more easily, especially when connecting to external services like transit data or nonprofit resource directories.

1.3 Database

- **Database**: Firebase
- **Reasoning**:

We chose Firebase for our database because it gives us the flexibility we need for this type of project. Nonprofit and government resource data often comes in different formats, and Firebase's NoSQL structure makes it easier to store and manage without forcing us into a rigid design. Another important factor is Firebase's offline caching feature, which allows users with unstable internet connections to still complete surveys and view their results. This is especially important for the ALICE population, who may not always have reliable access to broadband. Looking ahead, Firebase also gives us room to grow, such as saving user histories, generating recommendations, or syncing across multiple devices.

1.4 Authentication & Profiles

- **Tool**: Firebase Authentication (or Auth0, Passport.js)
- **Reasoning**:

  For authentication, we went with Firebase Authentication because it's simple to set up and secure. It lets users create persistent profiles so they can come back later and pick up where they left off. Firebase also supports different login methods, such as email or phone number, which makes it more flexible for users who may not always have the same technology available. While other tools like Auth0 or Passport.js are also strong options, Firebase stood out to us because it integrates smoothly with the database and other Firebase services, which keeps our system more consistent and easier to manage.

1.5 External Integrations

- **APIs/Connections**: Links to nonprofit directories, government resource databases, and local groups.
- **Reasoning**: Ensures data accuracy and provides real-world utility.

---

2. Use Case Diagram

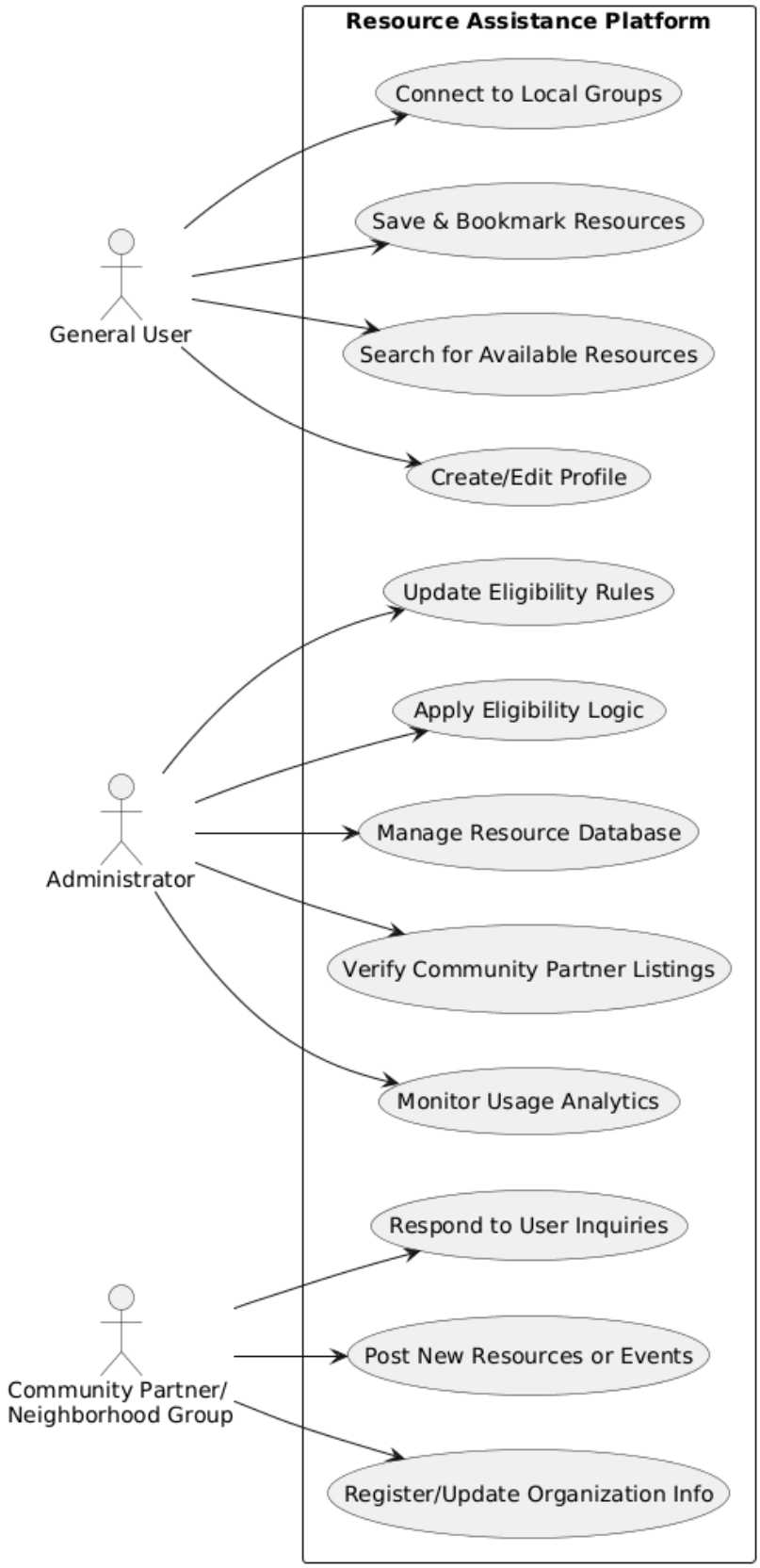
User Roles:

- **General User (low-income individual/family)**
- **Administrator**

- **Community Partner/Neighborhood Group**

Tasks/Use Cases:

- **General User:**
    - Create/Edit Profile
    - Search for Available Resources
    - Save & Bookmark Resources
    - Connect to Local Groups
- **Administrator:**
    - Manage Resource Database
    -
    - Apply Eligibility Logic
    - Update Eligibility Rules
    - Monitor Usage Analytics
    - Verify Community Partner Listings
- **Community Partner:**
    - Register/Update Organization Info
    - Post New Resources or Events
    - Respond to User Inquiries


Resource Assistance Platform
Connect to Local Groups
Save & Bookmark Resources
Search for Available Resources
Create/Edit Profile
Update Eligibility Rules
Apply Eligibility Logic
Manage Resource Database
Verify Community Partner Listings
Monitor Usage Analytics
Respond to User Inquiries
Post New Resources or Events
Register/Update Organization Info
General User
Administrator
Community Partner/
Neighborhood Group

Functional Requirements

1. **User Registration & Authentication**

- The system shall allow users to create an account and log in securely.
- The system shall authenticate users before allowing them to access saved resources or personal profiles.

2. **User Profiles**

- The system shall allow users to create and update profiles with information such as:
    - Location (city/ZIP code)
    - Transportation (status)
    - Disability (status)
    - Income range
    - Household size
    - Student status (optional)
- The system shall store this data in the database for use in eligibility checks.

3. **Eligibility Matching**

    - The system should match users with resources they are eligible for based on stored profile information (e.g, income and family size).
    - The system shall hide resources that the user is not eligible for.

4. **Resource Finder**

- The system shall allow users to search and filter resources by type (e.g., food, housing, nonprofit aid).

5. **Saved Resources**

    - The system shall allow users to bookmark resources to their profile for later access.

    - The system shall allow users to view and manage their saved resource list.

6. **Notifications**

    - The system shall notify users when new resources or programs become available in their area.
    - The system shall notify users of technical issues and emergency notifications if they happen to mention them immediately.

    - Notifications may be delivered through in-app alerts (and optionally email/SMS in the future).

7. **Multilingual Support**

    - The application will support multiple languages, initially starting with English in the beta phase.

8. **Administrator Functions**

    - The system shall allow administrators to add, update, and remove resources (e.g., food banks, housing programs, nonprofit aid) in the database.

    - The system shall allow administrators to define and update eligibility criteria for each resource, such as:

        i. Income thresholds

        ii. Household size requirements

        iii. Student or residency status

    - The system shall ensure that only administrators have permission to perform these operations.

9. **Database**

- The system must use a database to store user profiles and resource information.

Non-Functional Requirements

1. Usability

- The frontend will use **React** to provide a simple, mobile-friendly interface with large buttons and bilingual support (English for the beta phase for now).

2. Performance

- Backend will use Node.js with Express.js
- Database queries in **Firebase Firestore** shall be optimized to support growth in the number of stored resources while maintaining smooth performance.

3. Scalability

- The system shall be able to support a growing number of users as more families and organizations join.
- The database design should allow expansion for future resources (e.g., healthcare, childcare).

4. Security

- All user data (income, household size, location) shall be stored securely in the database.
- Authentication will be required for accessing user profiles and saved resources using Firebase.
- Communication between the frontend and backend shall be encrypted (HTTPS).

5. Reliability/Availability

- The system should be available the majority of the time (**high uptime**) to ensure users can access resources when needed.

- Data should remain consistent and not be lost in case of failures.

6. Maintainability

- The system shall be built in a modular structure so components (frontend, backend, database) can be updated independently.
- Code and database rules will be written clearly to allow future developers to maintain and extend the application.

7. Accessibility

- The application shall follow basic accessibility standards (readable fonts, contrast, clear navigation).
- Support for multiple languages will be added, starting with English and extendable to others.

---

4. Group Contributions

- **Diana Avila:** Designed system architecture and selected frontend/backend technologies.
- **Sangit Gaire:** Drafted functional and non-functional requirements.
- **Kobe Leverett:** Created Use Case diagram and defined user roles.
- **Puru Sahuja:** Compiled final document, cross-checked formatting, and prepared submission.