Overall System Architecture

1. Client Layer

- **Web App (React)**: Dashboard, resource upload, mentorship matching.
- **Mobile App (Flutter)**: Notifications, calendar sync, community discussions.

2. Backend Layer (Node.js/Express)

- **Auth Service**: JWT token generation, MFA .
- **Event Service**: Manages event creation, RSVPs, and Google Calendar sync .
- **Community Service**: Handles group creation, discussions, and mentorship tools .

3. AI Layer (Microservices)

- **Chatbot Service**: Flask API + Dialogflow (answers university queries).
- **Recommendation Service**: TensorFlow model for personalized suggestions.

4. Storage Layer

- **MongoDB Atlas**: Stores user profiles, events, and community data.
- **Elasticsearch**: Powers resource search and discovery .

5. Security Layer

- **API Gateway**: Kong/Express Gateway (rate limiting, JWT validation).
- **Encryption**: AES-256 (data at rest), TLS 1.3 (data in transit).

6. Integration Layer

- **Third-Party APIs**: Google Calendar (events), Firebase (notifications), Twilio (MFA).

Data Flow

1. Users interact via React/Flutter clients.
2. Requests pass through the API Gateway for security checks.
3. Node.js backend processes CRUD operations (e.g., event RSVPs, profile updates).
4. AI services provide real-time chatbot responses and recommendations.
5. Real-time notifications (Socket.io/FCM) and calendar sync ensure timely updates.

Why This Architecture?

- **Modularity**: Separates concerns (backend, AI, security) for scalability.
- **Agile Alignment**: Supports iterative sprints
- **Compliance**: Meets security NFRs (encryption, audits) and usability goals (responsive UI/UX).