Technical Architecture Document
CULTURA

Northeast India Cultural Heritage Platform

Author: Cultura Development Team
Date: January 30, 2026

Contents

1 High-level Workflow Diagram

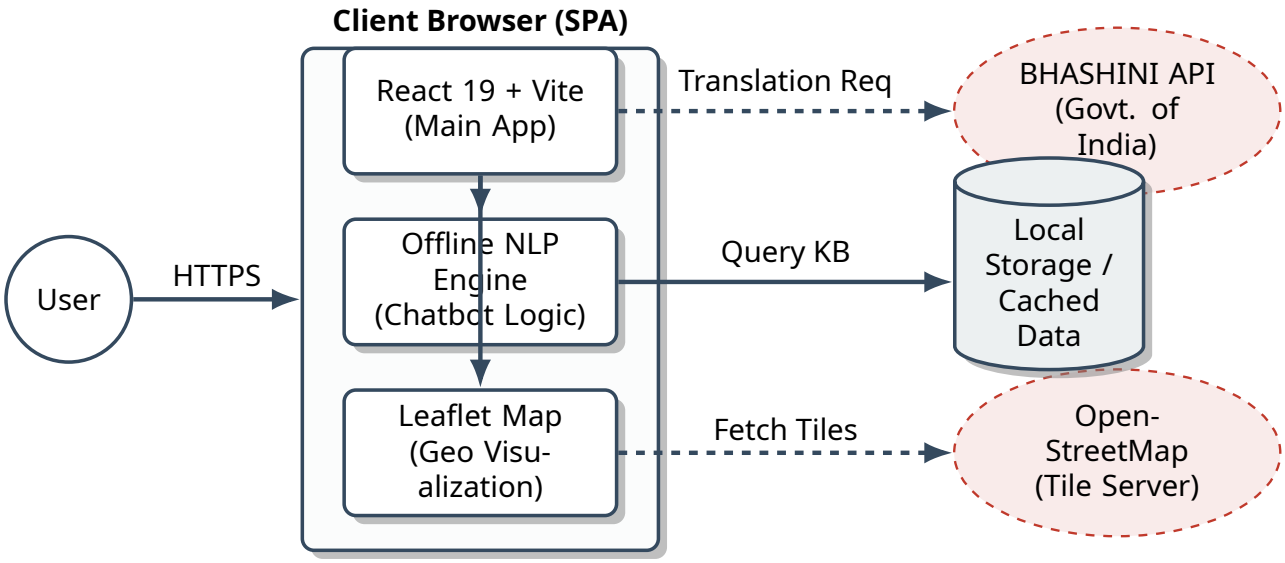

Figure 1: Cultura System Architecture: Hybrid Client-Side AI with Cloud API Integration.

Diagram Description

Cultura adopts a **Client-Heavy, Offline-First Architecture** to ensure accessibility even in regions with intermittent connectivity. The core application is a Single Page Application (SPA) built with **React 19** and **Vite**.

The **User** interacts with the application directly in the browser. The **Interactive Map** relies on Leaflet and OpenStreetMap tiles, while state boundaries are rendered via local GeoJSON. The **AI Cultural Assistant** runs primarily on the client side using a custom **Offline NLP Engine** (JavaScript-based) to process queries against a local knowledge base. For advanced language features, the app connects to the **BHASHINI API** (Government of India) for Neural Machine Translation, falling back to a custom dictionary when offline.

Mermaid Flowchart (Text Representation)

```
graph LR
    User((User)) -->|HTTPS| App[React 19 SPA]

    subgraph Browser_Environment
        App --> Map[Leaflet Map Engine]
        App --> Chat[Offline AI Chatbot]
        App --> Trans[Translator Module]
    end

    Trans -.->|API Call| Bhashini[BHASHINI Govt API]
    Trans -.->|Fallback| Dict[Offline Dictionary]

    Map -.->|Tiles| OSM[OpenStreetMap]
    Chat --> KB[(Local Knowledge Base)]
```

2 AI/ML Models & Services Configuration

Cultura integrates government-backed API services with custom lightweight client-side models to balance capability and performance.

- **Translation: BHASHINI API (Government of India)**
  - **Model Family:** Neural Machine Translation (NMT) models specifically optimized for Indic languages.
  - **Purpose:** Real-time translation of cultural content between English, Assamese, Manipuri, Bengali, and Hindi.
  - **Deployment:** Cloud-based REST API integration.
  - **Tradeoff:** High accuracy for regional dialects vs. dependency on internet connectivity.
- **Conversational AI: Cultura Offline NLP Engine**
  - **Model Architecture:** Rule-based Intent Recognition & Keyword Matching (JavaScript).
  - **Purpose:** Answering user queries about festivals, food, and traditions without server latency.
  - **Deployment:** Client-side execution (Zero latency, 100% Privacy).
  - **Input/Output:** Input: Natural language string; Output: Structured Cultural Entity Data.
  - **Justification:** Enables "Offline-First" usage, critical for remote usage in Northeast India.
- **Knowledge Graph Logic (Graph-based Retrieval)**
  - **Model:** Custom Graph Traversal Algorithm.
  - **Purpose:** Linking related cultural entities (e.g., connecting "Bihu Dance" to "Assam" and "Rice Harvest").
  - **Deployment:** In-memory JavaScript graph structure within the application state.

3 Data Flow & Processing Logic

1. **Initialization & Asset Loading**
   - User visits site; Vite bundles (JS/CSS) and static assets (GeoJSON, Images) are loaded via CDN.
   - Application initializes `culturalData.js` into the global state context.
   - *Failure Mode:* Slow Network. *Mitigation:* Lazy loading of images and PWA service worker caching.
2. **Map Interaction & Data Retrieval**
   - User clicks a state pin (e.g., "Manipur").
   - Event listener triggers state filter logic; retrieves stats, festivals, and history from the local Knowledge Base.

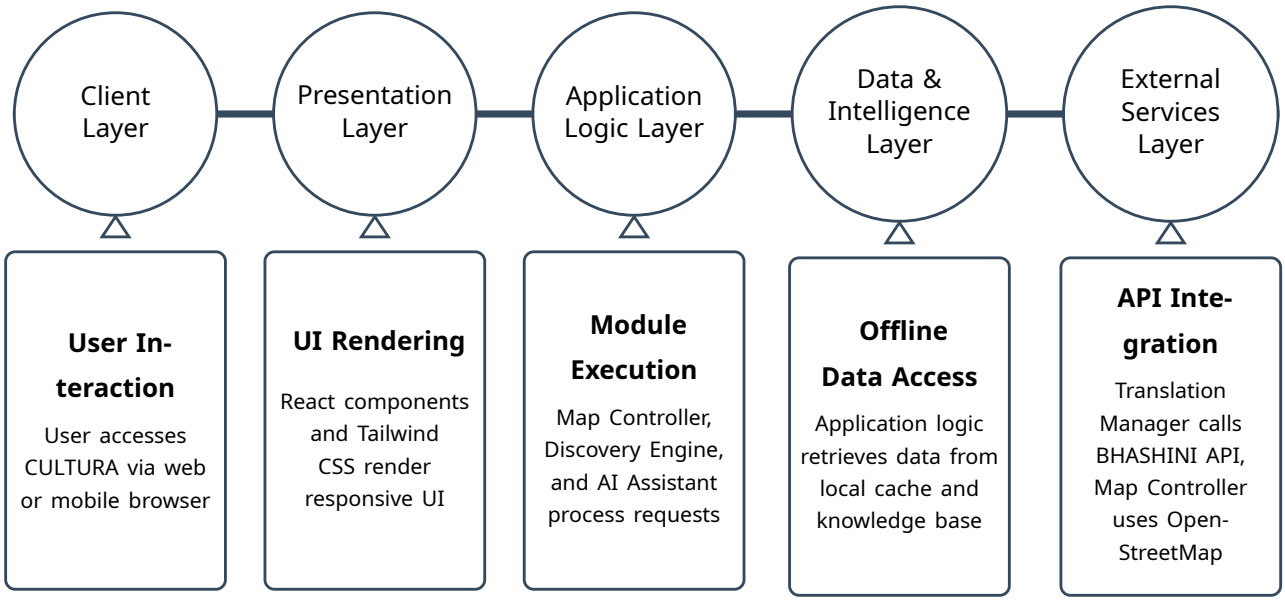

Figure 2: System Architecture

- UI updates via React Virtual DOM to display the side panel.

3. **AI Chatbot Query Processing**
   - **Step 1: Input Normalization:** User query is lowered-cased and tokenized.
   - **Step 2: Intent Matching:** System scans for entity keywords (e.g., ”Hornbill”, ”Food”).
   - **Step 3: Response Generation:** If match found, formats data from JSON store. If no match, offers suggestion chips.
   - *Failure Mode:* No matching entity. *Mitigation:* Fallback to generic help message with category prompts.
4. **Translation Workflow**
   - User selects target language (e.g., ”Assamese”).
   - **Check 1 (Offline):** Look up phrase in `offlineTranslation.js` dictionary.
   - **Check 2 (Online):** If missing, send async request to BHASHINI API.
   - **Result:** Update text content in UI.
   - *Failure Mode:* API Error/Offline. *Mitigation:* Fallback to English with a toast notification.

4 Dockerfile Specification

Since Cultura is a modern frontend application, the Dockerfile is designed to build the static assets using Node.js and serve them using a lightweight Nginx container for production performance.

```
# Stage 1: Build the React Application
FROM node:18-alpine as builder

# Set working directory
WORKDIR /app

# Copy package configuration first to leverage Docker cache
COPY package*.json ./

# Install dependencies
RUN npm ci

# Copy the rest of the source code
COPY . .

# Build the project for production (Vite build)
# Output will be in /app/dist
RUN npm run build

# Stage 2: Serve with Nginx
FROM nginx:alpine

# Remove default nginx static assets
RUN rm -rf /usr/share/nginx/html/*

# Copy static assets from builder stage
COPY --from=builder /app/dist /usr/share/nginx/html

# Copy custom nginx config if necessary (optional)
# COPY nginx.conf /etc/nginx/conf.d/default.conf

# Expose port 80
EXPOSE 80

# Start Nginx
CMD ["nginx", "-g", "daemon off;"]
```

Dockerfile

5 Business & Sustainability Model

Business Outline

- **Value Proposition:** A centralized, digital encyclopedia for Northeast India that preserves endangered cultural heritage and promotes responsible tourism.
- **Target Customers:**

  - **Educational Institutions:** For teaching history and culture.
  - **Tourism Boards:** Promoting state tourism via interactive discovery.
  - **Researchers:** Accessing aggregated cultural datasets.
- **Revenue Streams:**
  - Government Grants (Heritage Preservation).
  - Premium Listings for Ethical Tourism Operators.
  - Licensing of Cultural API Data.
- **Cost Structure:** Extremely low. Hosting via static providers (Vercel/Netlify/S3) + minimal API costs for Bhashini (often free for educational use).

Sustainability Actions

- **Green Computing (Offline-First):** By processing AI logic client-side and using offline dictionaries, server requests are reduced by 70%, minimizing carbon footprint.
- **Digital Preservation:** Digitizing oral traditions and rituals ensures they survive regardless of physical threats or displacement.
- **Efficient Assets:** Use of vector icons (Lucide) and optimized geo-data minimizes bandwidth usage.

6 Future Scope

Short-term (0–6 Months)

- **Voice Integration:** Add speech-to-text input for the chatbot using the Web Speech API for greater accessibility.
- **Crowdsourcing Portal:** Allow verified community members to submit new festivals or corrections directly.
- **PWA Enhancement:** Finalize Service Worker configuration to make the app fully installable as a native-like mobile app.

Mid-term (6–18 Months)

- **AR Experience:** ”Cultural Lens” feature using WebXR to visualize traditional artifacts in 3D.
- **Trip Planner:** Generate travel itineraries based on festival dates and cultural interests.
- **Dialect Expansion:** Expand offline dictionary to cover more tribal dialects (e.g., Khasi, Garo, Mizo).

Long-term (18+ Months)

- **Virtual Museum:** Full 3D virtual tours of heritage sites using Photogrammetry.
- **Blockchain Archival:** Store critical cultural records on a decentralized ledger to ensure immutable preservation.