Introduction

Welcome to the Design Documentation for the MUMBI Project!

This document provides an in-depth look at the architecture, runtime components, and technical setup of our AI-powered family play companion. Our goal is to create a seamless, adaptive play experience that integrates neuroscience, child development principles, and cutting-edge AI.

Below, you'll find detailed information on our system architecture, runtime flow, tech stack, and project structure to help you understand how MUMBI works and how it’s being developed.

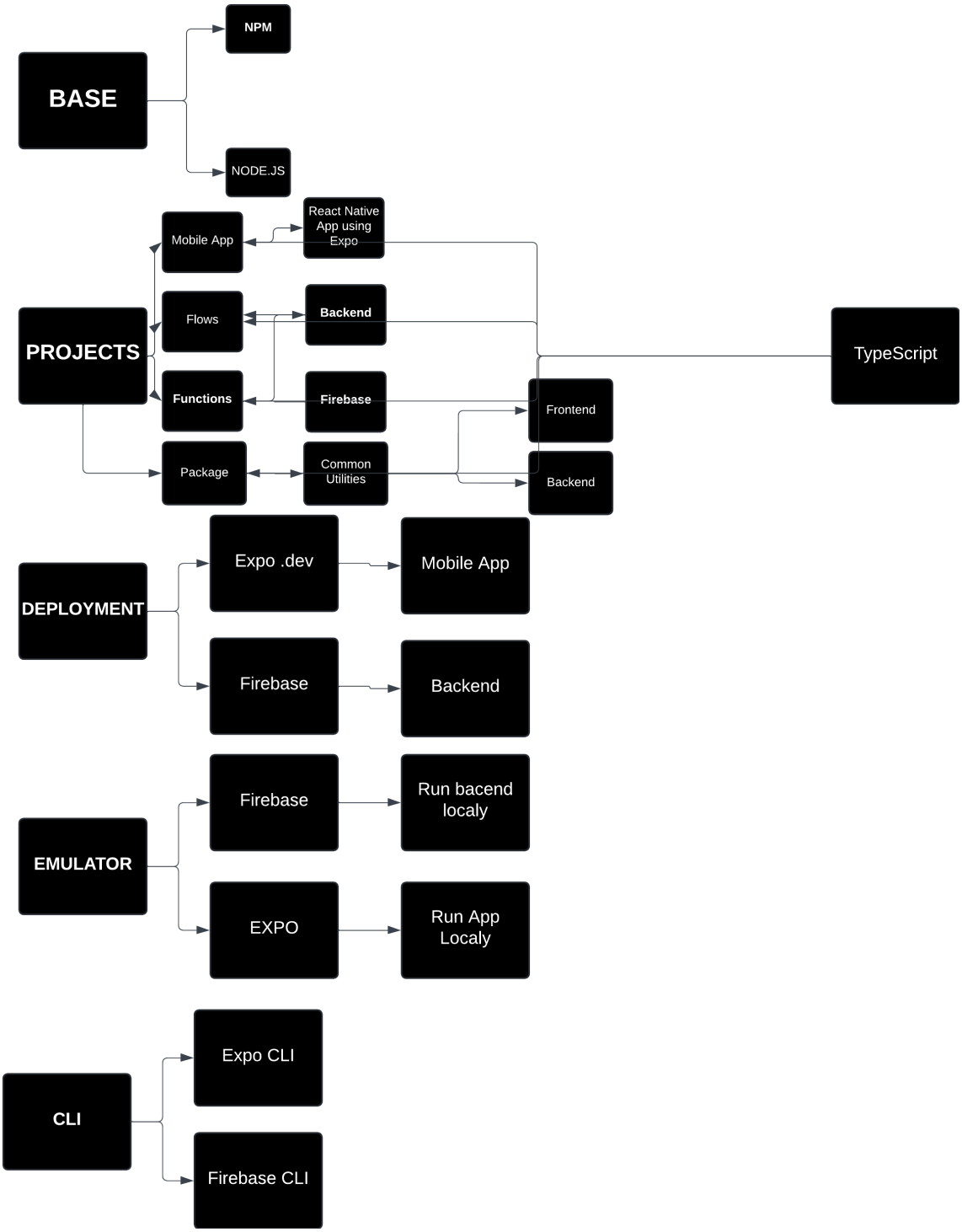
Code Components

Our system architecture consists of multiple interconnected projects , with the core built on React Native Expo, supported by Node.js and NPM for development.

Key Components

1. **React Native Expo Application**: Our cross-platform app for Android and iOS, powered by a single React Native codebase.
2. **Firebase GenKit Flows**: Handles all LLM-related operations, including text and multimodal AI queries.
3. **Firebase Functions**: Acts as a secure intermediary layer between the app and Firestore to enhance data security and access control.
4. **Deployment**:
    - Mobile apps are built and deployed via `expo.dev`.
    - Backend operations (Firebase Functions and GenKit Flows) are deployed separately on Firebase infrastructure.
    - Development is supported by Android and iOS emulators with a local Firebase environment.

Diagram of code components



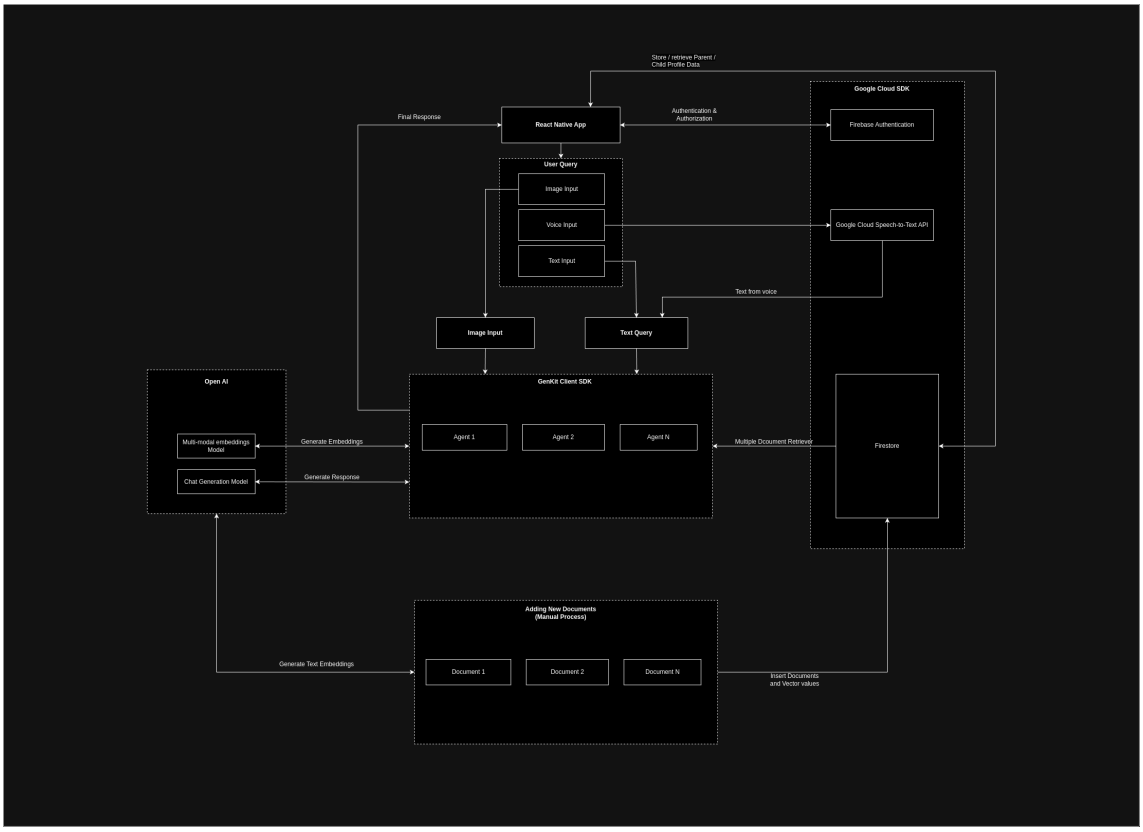
Runtime Components

System Overview

The system integrates React Native, Firebase, and OpenAI APIs to deliver personalized, interactive play experiences.

1. **React Native**: Allows users to interact with an AI that generates personalized play suggestions for parents of children under 5.
2. **Firestore**: Stores user data such as parent profiles, child information (e.g., name, age), and play preferences, enriching AI-driven suggestions.
3. **Input Processing**:
    - **Speech-to-Text**: Voice inputs are converted to text via Google Cloud Speech-to-Text API.
    - **Multimodal Embeddings**: Text and image inputs are processed using a multimodal embeddings model to generate vector representations.
    - **Document Retrieval**: Embeddings are used to search a vector database for relevant documents.
    - **Chat Generation**: Relevant documents and user queries are passed to the Chat Generation Model, with responses delivered back to the app.

Diagram of runtime components



Tech Stack Used

For our project, we are going to use the following tech stack:

Frontend

1. Typescript

2. React Native
3. Expo Framework
4. React Native Reusables UI Library
5. Vite for easy setup/build

Backend

1. Node.js
2. Firebase
3. Google Cloud
4. Firebase GenKit
5. OpenAI API

DevOps

1. GitHub Actions

Detailed Project Structure

Components

1. **Core**
    - Node.js
    - Expo CLI
    - Firebase CLI
    - Emulators
2. **Project Setup**
    - **Turborepo**: `monorepo` tool for managing multiple `js/ts` projects.
    - **Firebase services**:
        - **Auth**: For user authentication.
        - **Firestore**: For data storage.
        - **Functions**: For server-side logic.
        - **GenKit**: For AI-driven logic.
    - **Biome.js**: For linting and code formatting.
    - **Google Cloud Platform**: APIs for speech-to-text and other services.

Repository Structure

```
├── .firebaserc
├── .gitignore
├── .turbo
│   └── daemon
├── Documentation
│   └── README.md
├── README.md
├── biome.json
├── firebase.json
├── firestore.indexes.json
├── package.json
├── turbo.json
```

```
├── apps
│   ├── flows
│   │   ├── package.json
│   │   └── vite.config.ts
│   ├── functions
│   │   ├── package.json
│   │   └── vite.config.ts
│   └── mobile
│       ├── App.tsx
│       ├── app.json
│       ├── global.css
│       ├── metro.config.js
│       └── tailwind.config.js
├── Deliverables
│   ├── README.md
│   └── sprint-01
│       ├── feature-board.tsv
│       └── readme.md
└── .github
    └── ISSUE_TEMPLATE
        ├── bug_report.md
        └── feature_request.md
```