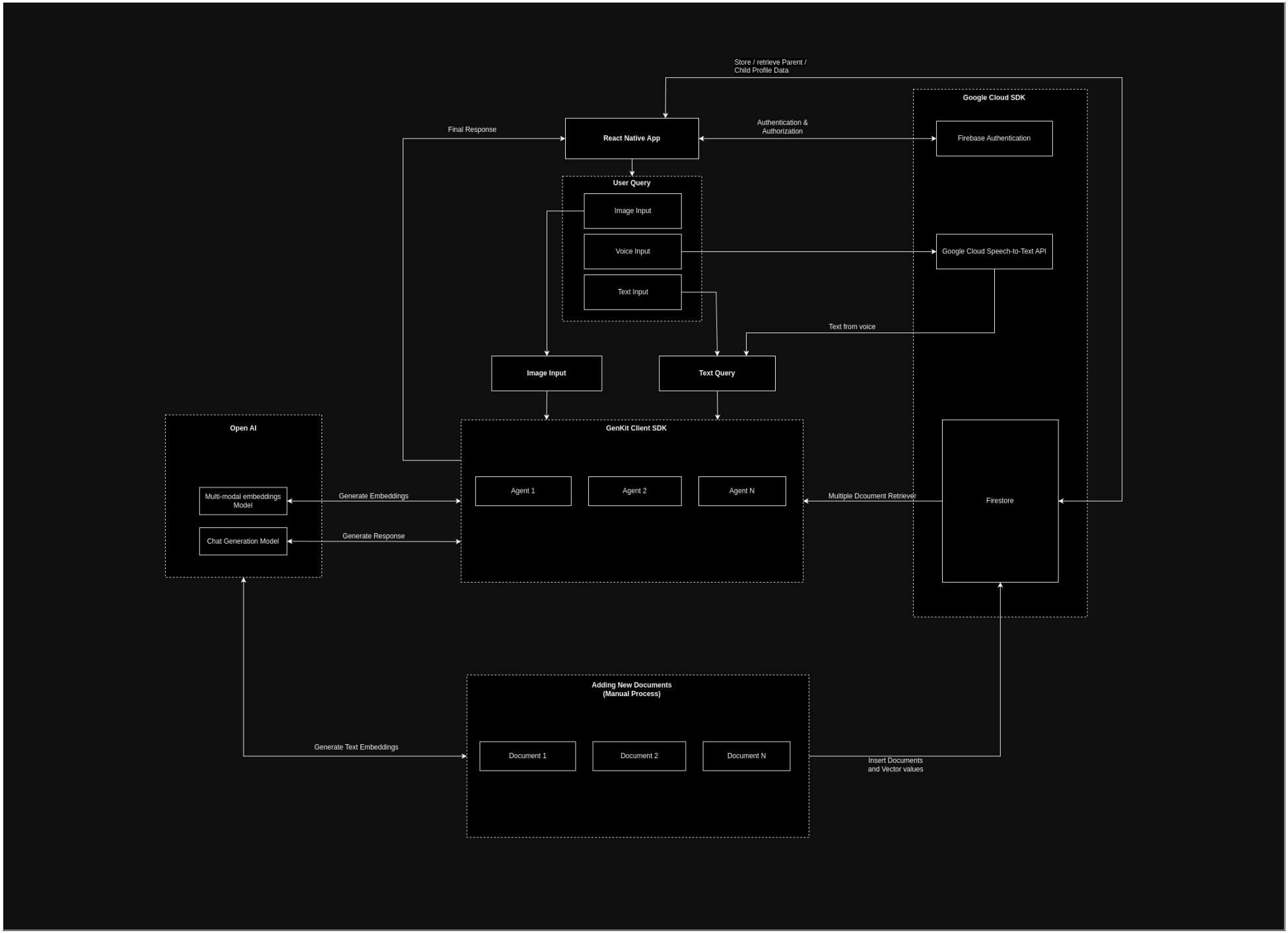
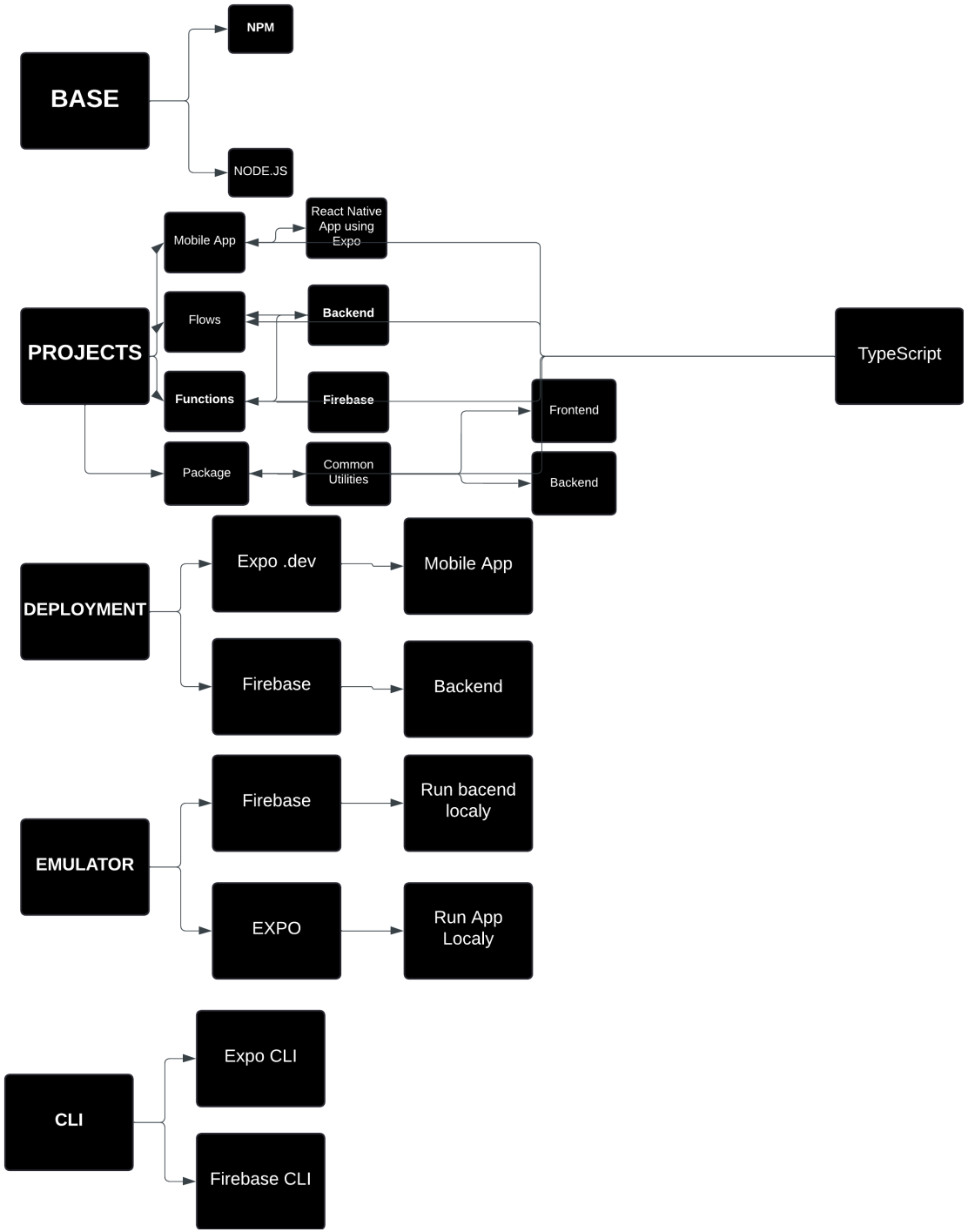

Store / retrieve Parent / Child Profile Data
Google Cloud SDK
Final Response
React Native App
Authentication & Authorization
Firebase Authentication
User Query
Image Input
Voice Input
Text Input
Google Cloud Speech-to-Text API
Text from voice
Image Input
Text Query
Open AI
GenKit Client SDK
Multi-modal embeddings Model
Generate Embeddings
Agent 1
Agent 2
Agent N
Multiple Dcoument Retriever
Firestore
Chat Generation Model
Generate Response
Adding New Documents (Manual Process)
Generate Text Embeddings
Document 1
Document 2
Document N
Insert Documents and Vector values


BASE
NPM
NODE.JS
PROJECTS
Mobile App
React Native App using Expo
Flows
Backend
Functions
Firebase
Package
Common Utilities
Frontend
Backend
TypeScript
DEPLOYMENT
Expo .dev
Mobile App
Firebase
Backend
EMULATOR
Firebase
Run bacend localy
EXPO
Run App Localy
CLI
Expo CLI
Firebase CLI

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

Frontend

- Typescript
- React Native
- Expo Framework
- React Native Reusables UI Library
- Vite for easy setup/build

Backend

- Nodejs
- Firebase
- Google Cloud
- Genkit
- OpenAI

DevOps

- GitHub Actions

Textual explanation of System Architecture - Diagram of Runtime Components

Overview:

The system uses React Native with Firebase for Authentication and document storage. Firebase GenKit handles AI queries with multiple modalities. For text generation and embedding, the system leverages OpenAI.

1. React Native
   Our App allows the user to speak and chat with an AI model that generates personalized play suggestions, helping parents engage children under 5 in their current activities.

2. Firestore
   Important data like information about the parent or a profile of the child, which includes its name, age and other relevant information are stored in Firestore. This information will be part of every chat generation query to OpenAI to further improve and personalize the results.

3. Processing the Input
   First, the user query is pre-processed by handing the voice input to Google Cloud Speech-to-Text API. The text input and any image input are then passed to Firebase GenKit. We then use a multimodal embeddings model to create embeddings (vector representations) for the combined input. The embeddings are then used to search a vector database for relevant documents. The found documents, together with the question, are then passed to the Chat Generation Model, and the result is returned to our React Native App.

Textual explanation of System Architecture - Diagram of Code Components

Our system architecture consists of multiple interconnected projects.

The core component is a React Native Expo application, developed using Node.js and NPM as the development environment.

Firebase GenKit Flows handle all LLM-related operations, while other backend operations are managed through Firebase Functions. We've chosen to implement Firebase Functions as an intermediary layer between the React Native App and Firestore to enhance data security and access control.

For deployment, we utilize expo.dev, a comprehensive platform for React Native Expo application development. This platform generates both Android and iOS applications from a single React Native codebase. The Firebase Functions and GenKit Flows are deployed separately on the Firebase infrastructure.

For local development, we employ Android and iOS emulators alongside a local Firebase environment.

1. TypeScript
2. Core
    a. Node.js
    b. Expo CLI
    c. Firebase CLI
    d. Emulators
3. Project Setup
    a. Turborepo - monorepo tool to set multiple js/ts projects
    b. Firebase
        i. Auth - for auth
        ii. Firestore - for data storage
        iii. Functions - for server side logics (if need)
        iv. Genkit - for AI related server side logic
    c. Biome.js
        i. Lint and formatting of code
    d. GCP
        i. APIs
    e. React Native reusables

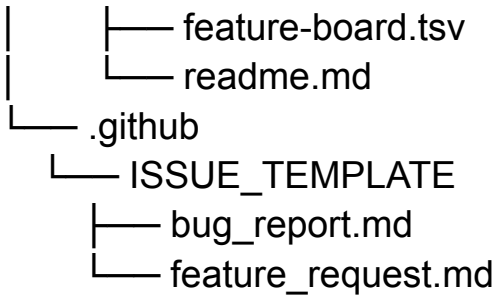
Repo Structure

- .github
    - Workflow to check lint and style
    - eas build and submit to expo.go build
    - Write release note auto
- .turbo
- apps
    - mobile
    - functions
    - flows
- packages
    - package-1
    - package-2
- scrapers
    - If needed
- .env

- Firebase files
- package.json (main package json file)
- Turbo.json (turborepo config file)
- biome.json
- Other required files

Deepview of Project

```
├── .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