# Project – High Level Design

# on

# Non-Profit QuizBot

Course Name: GenAI

***Institution Name:*** Medicaps University – Datagami Skill Based Course

*Submitted by:*

| Sr no | Student Name | Enrolment Number |
|---|---|---|
| 1. | Anushka Kashyap | EN22CS301180 |
| 2. | Ananya Subramanya Rao | EN22CS301120 |
| 3. | Anirudh Vyas | EN22CS301131 |
| 4. | Atharv Chaturvedi | EN22CS301228 |
| 5. | Harshit Panchal | EN23CS3L1010 |

*Group Name: Group 03D2*

*Project Number: GAI-15*

*Industry Mentor Name:*

*University Mentor Name: Hemlata Patel*

*Academic Year: 2026*

# Table of Contents

# 1. Introduction

The non-profit sector relies heavily on donor relationships, fundraising strategies, and community engagement. Professionals and volunteers working in this space need to continuously update their knowledge of best practices — from donor stewardship to impact reporting and ethical fundraising.

However, traditional learning methods lack personalization, contextual relevance, and immediate feedback. There is a clear gap between the information available in day-to-day donor communications and the ability to convert that information into structured learning experiences.

**QuizBot** bridges this gap by transforming real-world donor emails into intelligent, AI-powered assessments that test comprehension, reinforce learning, and track progress over time. **QuizBot** is an AI-powered web application built specifically for the **non-profit sector**. It takes donor emails as input and automatically converts them into intelligent multiple-choice quizzes using Google Gemini AI. Users take the quiz, get evaluated, receive detailed explanations, and track their learning progress over time.

**Problem Statement**

*Develop an interactive AI-driven educational bot for the Non-Profit domain that engages users in targeted assessments based on donor emails, evaluates their responses using a Large Language Model, and provides deep contextual explanations for incorrect answers — creating a personalized learning loop that bridges knowledge gaps and reinforces industry-specific concepts.*

**Project Objectives**

- Accept donor email content as input and generate contextual MCQ-based quizzes automatically
- Use Google Gemini LLM to generate questions, evaluate answers, and produce personalized feedback
- Store donor email embeddings in a Vector Database (ChromaDB/FAISS) for semantic search and retrieval
- Provide detailed AI-generated explanations for each correct and incorrect answer
- Track user quiz history, scores, and learning progress through Firebase Realtime Database
- Display analytics dashboards showing performance trends, accuracy rates, and achievements
- Implement secure user authentication via Firebase Authentication

## 1.1 Scope of the document

This High-Level Design (HLD) document describes the architectural design and system structure of **QuizBot** — an AI-driven educational assessment platform for the non-profit domain. The document covers:

- Overall system architecture and component interactions
- Technology stack and design decisions
- Data flow between frontend, backend, and third-party services
- Integration design for AI (Google Gemini), Vector Database (ChromaDB/FAISS), and Firebase
- Security, authentication, and API design at a high level

This document does not cover low-level implementation details such as individual function logic, database query optimization, or UI component internals — those are addressed in the LLD document.

## 1.2 Intended Audience

| Audience | Purpose |
|---|---|
| **Software Architects** | To review and validate the overall system design |
| **Backend Developers** | To understand service boundaries, API contracts, and data flow |
| **Frontend Developers** | To understand integration points with backend APIs and Firebase |
| **Project Managers** | To understand system scope, components, and dependencies |
| **QA Engineers** | To understand system boundaries for testing strategy |
| **Academic Evaluators** | To assess technical depth and design quality of the project |

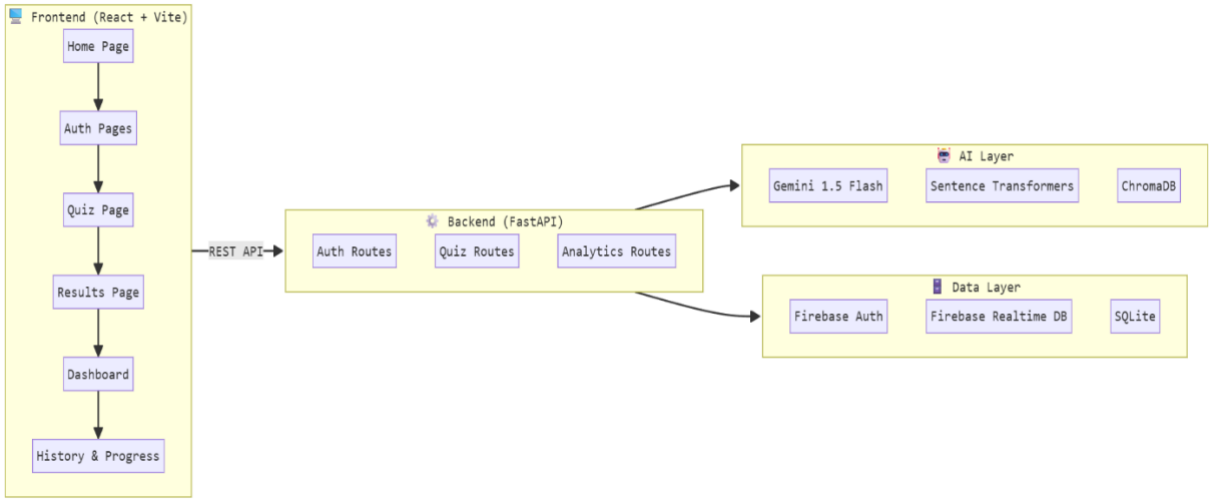
## 1.3 System overview

**QuizBot** is a full-stack, AI-powered web application designed for non-profit sector education. It allows users to paste donor emails and automatically generates contextual multiple-choice quizzes using a Large Language Model. The system evaluates responses, provides detailed AI-generated explanations, and tracks user learning progress over time.

**Core capabilities:**

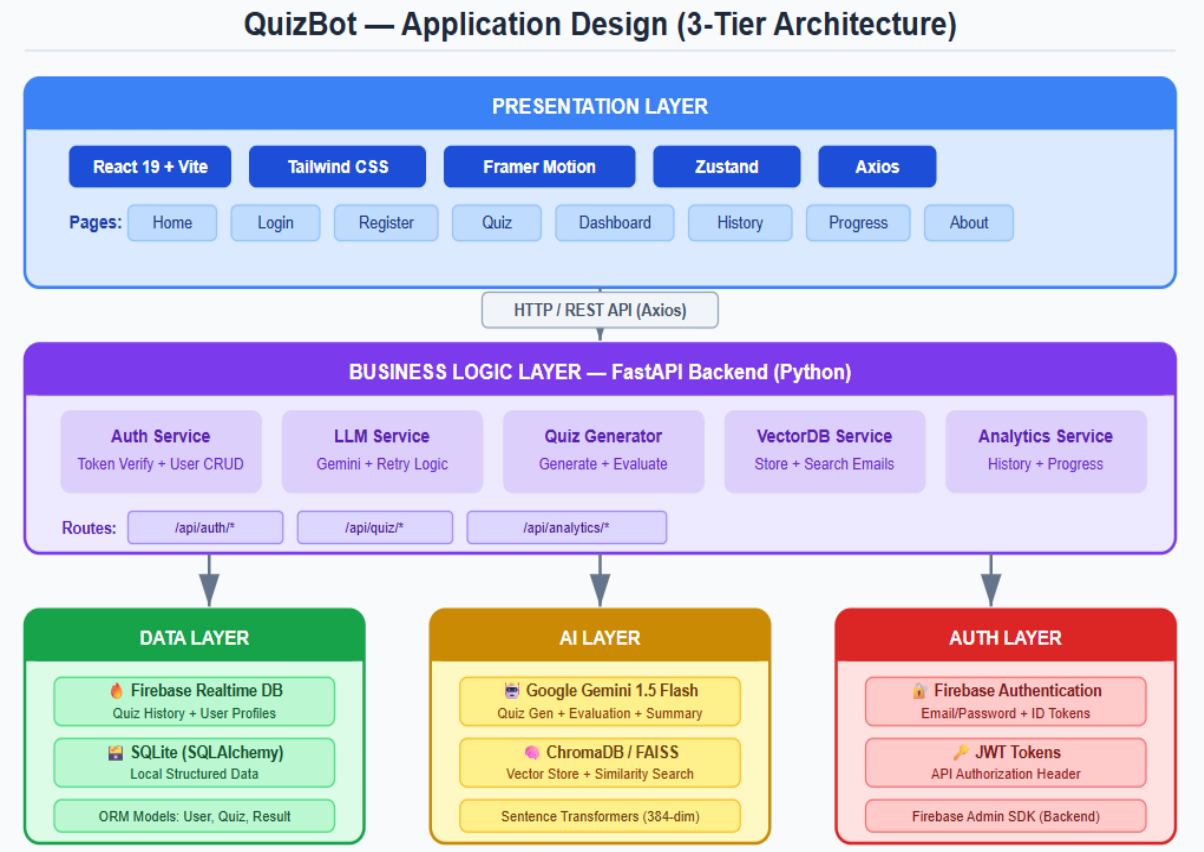
- **AI Quiz Generation** — Google Gemini API processes donor email content and generates 3–10 contextual MCQs
- **Vector Semantic Search** — ChromaDB stores email embeddings using Sentence Transformers for similarity matching
- **Smart Evaluation** — LLM evaluates each answer and generates personalized explanations
- **User Authentication** — Firebase Authentication handles secure email/password login and registration
- **Progress Tracking** — Firebase Realtime Database stores quiz history, scores, and analytics per user
- **Analytics Dashboard** — Users see performance trends, accuracy rates, and achievements

# 2. System Design



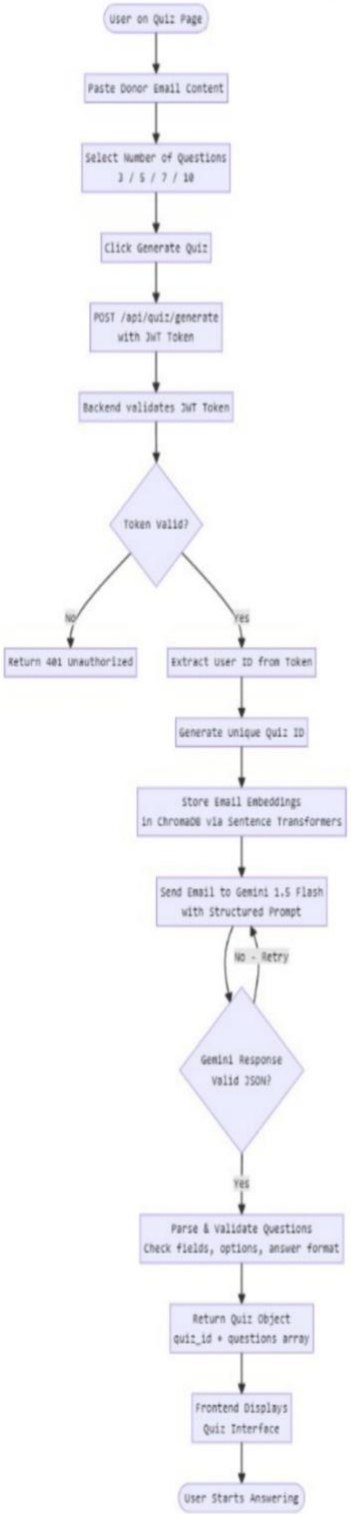

## 2.1 Application Design

QuizBot follows a 3-Tier Client-Server Architecture with clear separation between Presentation, Business Logic, and Data layers.

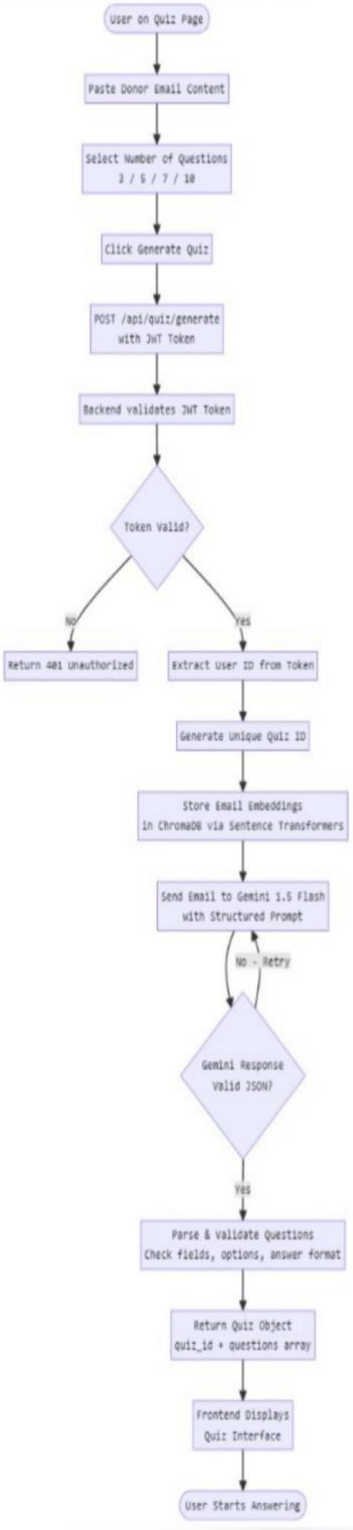


## 2.2 Process Flow

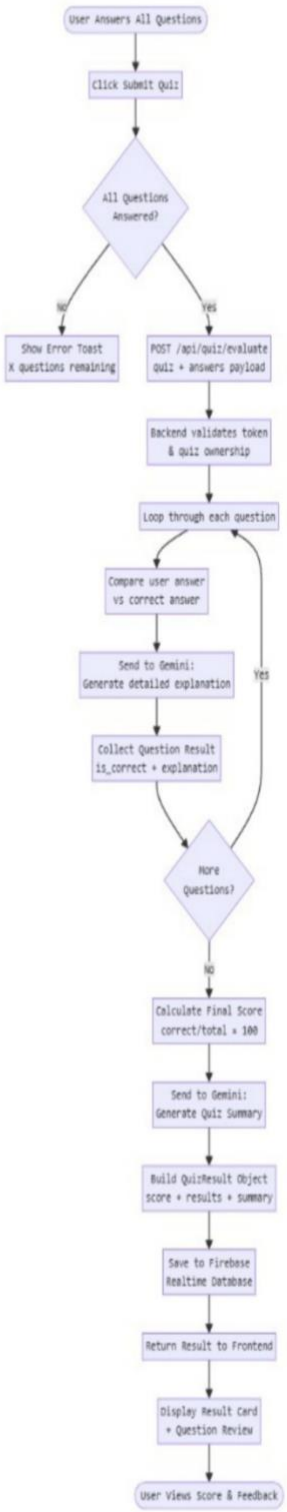
### 2.2.1 User Registration & Login Flow




### 2.2.2 Quiz Generation Flow

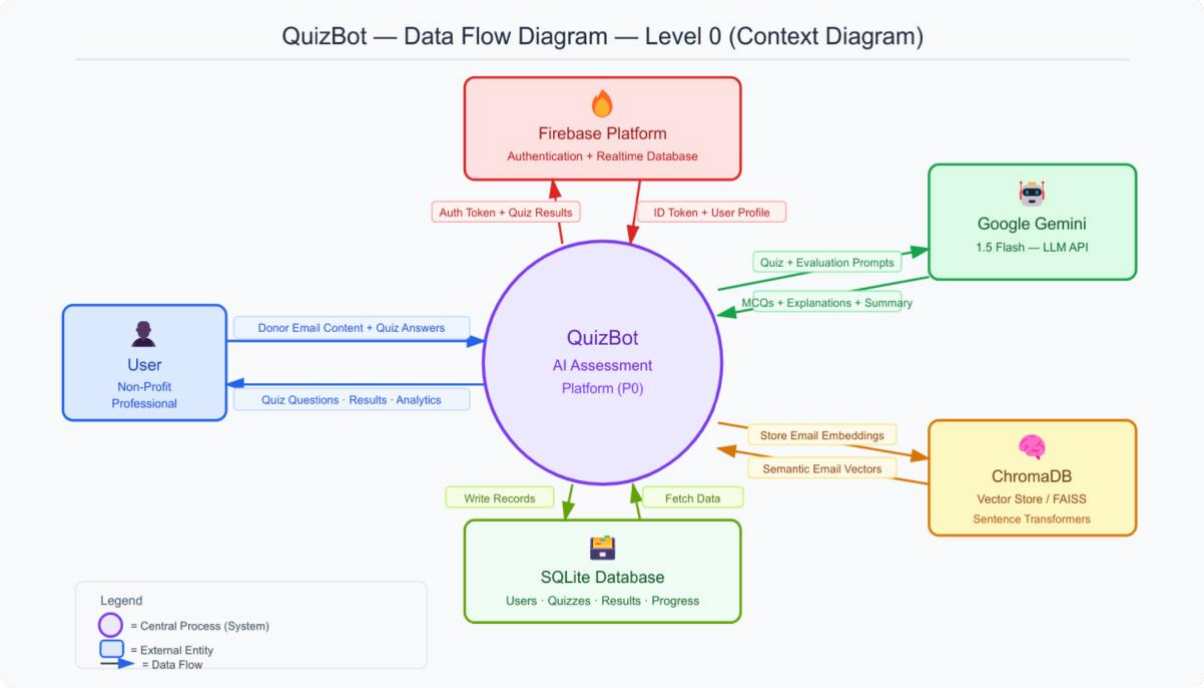



### 2.2.3 Quiz Submission & Evaluation Flow

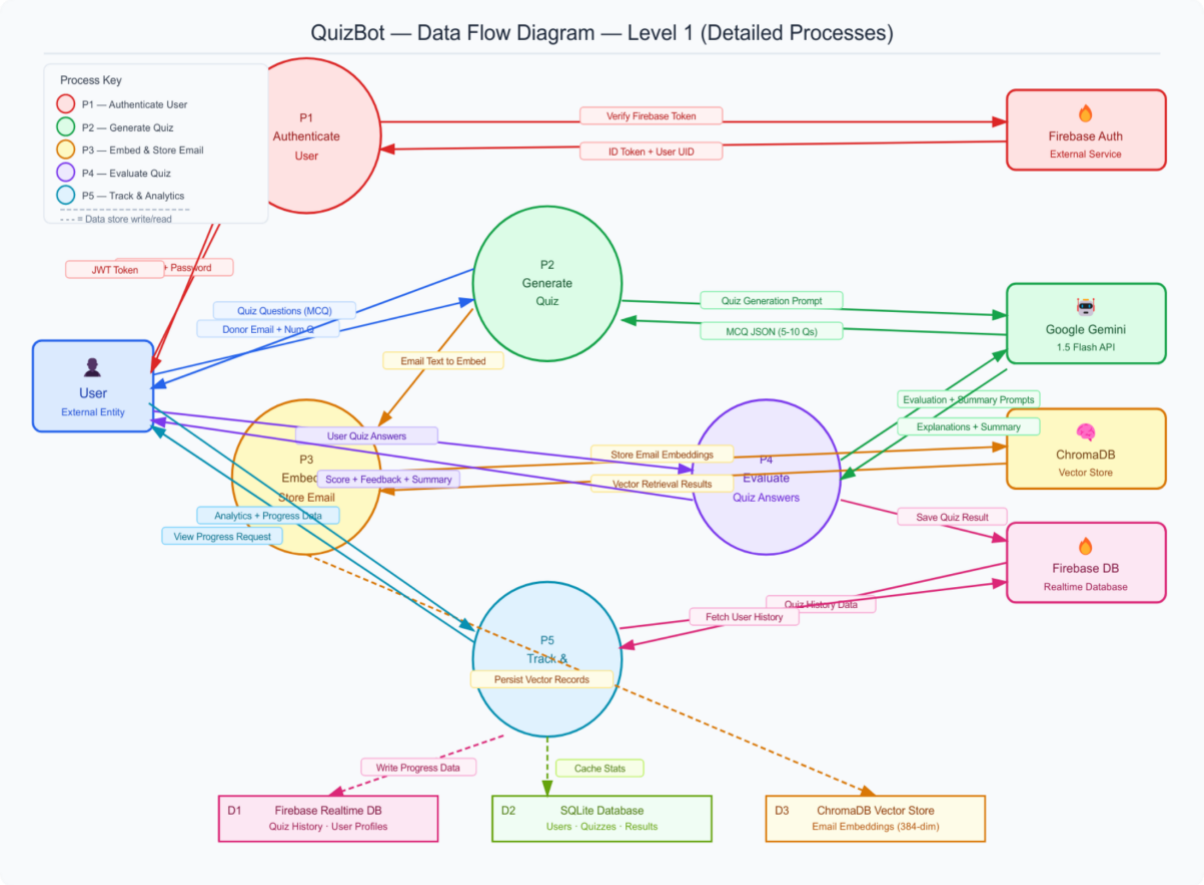


## 2.3 Information Flow

### 2.3.1 Data Flow Diagram — Level 0 (Context)




### 2.3.2 Data Flow Diagram — Level 1 (Detailed)

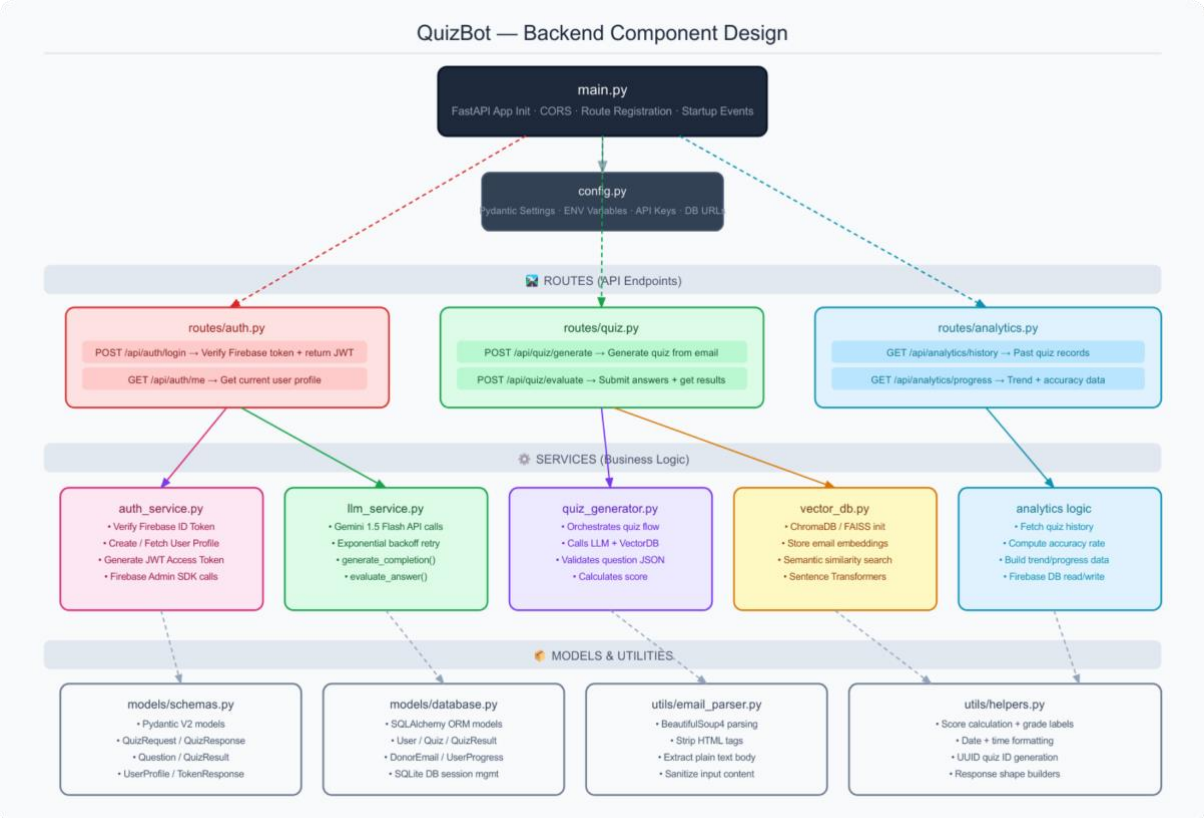


### 2.3.3 Information Flow Summary Table

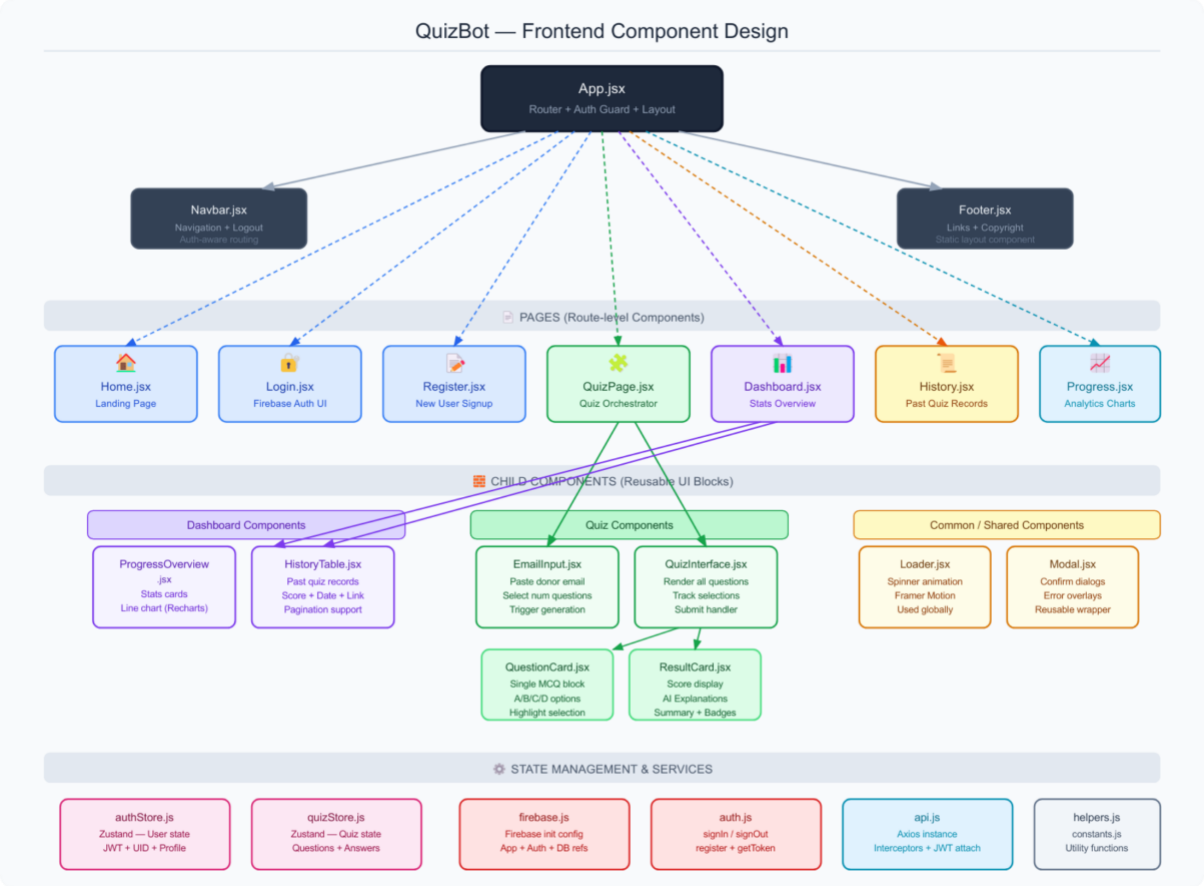
| Data | Source | Destination | Format | When |
|---|---|---|---|---|
| Donor Email Text | User Input | Backend → ChromaDB + Gemini | Plain Text | Quiz Generation |
| Email Embeddings | Sentence Transformers | ChromaDB | 384-dim Vector | Quiz Generation |
| Quiz Questions | Google Gemini | Frontend | JSON Array | After Generation |
| User Answers | Frontend | Backend | JSON Array | Quiz Submission |
| Answer Explanations | Google Gemini | Frontend | Plain Text | After Submission |
| Quiz Results | Backend | Firebase Realtime DB | JSON Object | After Evaluation |
| Auth Token | Firebase Auth | Frontend → Backend | JWT String | Login/Register |
| User Profile | Firebase Realtime DB | Backend → Frontend | JSON Object | Dashboard Load |
| Progress Analytics | Firebase Realtime DB | Backend → Frontend | JSON Object | Progress Page |

## 2.4 Components Design

### 2.4.1 Backend Components Design



### 2.4.2 Frontend Components Design




## 2.5 Key Design Considerations

**Scalability** — Grow without redesign
- Stateless FastAPI backend enables horizontal scaling
- Firebase Auth & Realtime DB auto-scale with demand
- Uvicorn multi-worker deployment ready
- ChromaDB isolated; replaceable with managed vector DB
- SQLite suitable for v1.0; migration path to PostgreSQL

**Reliability** — Resilient under failure
- Exponential backoff retry (up to 5 attempts) with jitter on Gemini calls
- Pre-built fallback responses if all retries fail
- Structured JSON validation via Pydantic models
- Graceful error responses with descriptive HTTP codes
- Silent Firebase token refresh before expiry

**Security** — Defence in depth
- Firebase-managed password hashing & storage
- Dual-token model: Firebase ID Token + short-lived JWT
- JWT verified on every protected API request
- CORS restricted to trusted frontend origin
- All secrets in environment variables, never in code

**Performance** — Fast by design
- Singleton services avoid repeated initialisation overhead
- Async FastAPI endpoints — non-blocking I/O throughout
- Firebase token caching reduces re-authentication calls
- Vite code splitting & tree-shaking minimise bundle size
- Zustand selective subscriptions prevent unnecessary re-renders

**Maintainability** — Easy to evolve
- Layered architecture: routes → services → data layer
- Pydantic models enforce strict request/response contracts
- Zustand stores provide clear, isolated frontend state
- Environment-based configuration for all deployments
- Modular React components with single responsibilities

**AI Integration** — Smart & cost-aware
- Gemini 1.5 Flash — balanced cost vs. capability for education
- Structured prompts with explicit JSON output format
- ChromaDB vector search grounds prompts with relevant context
- all-MiniLM-L6-v2 embeddings: fast (<100 ms) & lightweight
- Implicit prompt caching where Gemini API supports it

## 2.6 API Catalogue

**Base URL:** http://localhost:8000

**Authentication APIs - /api/auth**

| Method | Endpoint | Description | Auth Required |
|---|---|---|---|
| POST | /api/auth/login | Verify Firebase token, return JWT + user profile | No |
| GET | /api/auth/me | Get current authenticated user profile | Yes |

**Quiz APIs — /api/quiz**

| Method | Endpoint | Description | Auth Required |
|---|---|---|---|
| POST | /api/quiz/generate | Generate quiz from donor email | Yes |
| POST | /api/quiz/evaluate | Submit answers and get results | Yes |

**Analytics APIs — /api/analytics**

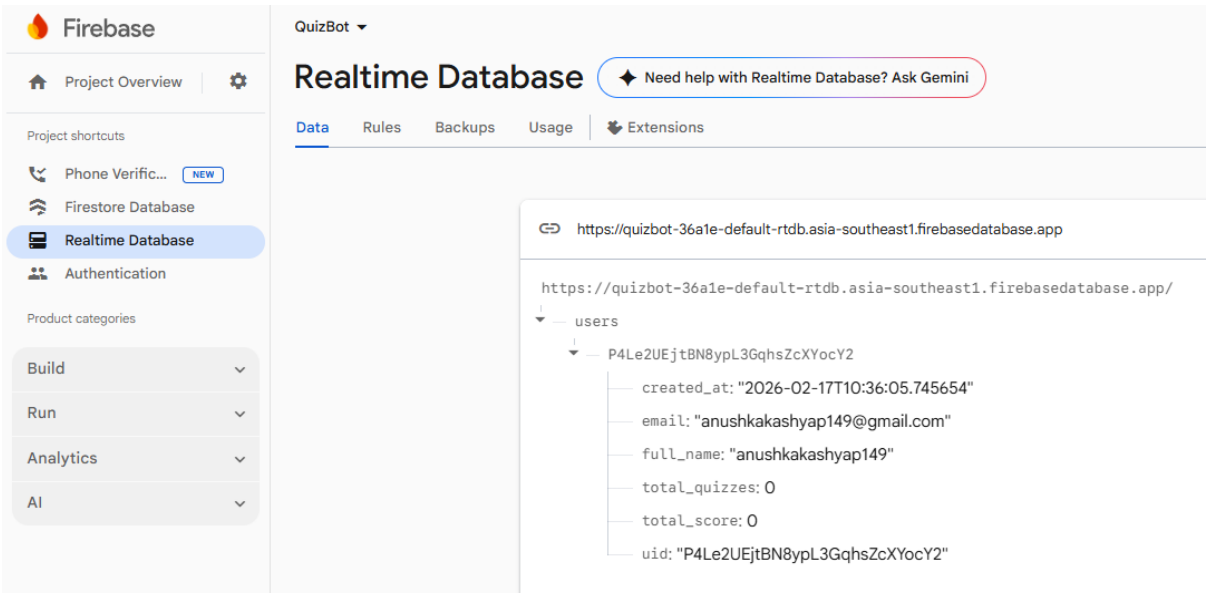
| Method | Endpoint | Description | Auth Required |
|---|---|---|---|
| GET | /api/analytics/history | Get all past quiz records for user | Yes |
| GET | /api/analytics/progress | Get progress analytics and trend data | Yes |
| GET | /api/analytics/stats | Get combined user stats + analytics | Yes |

# 3. Data Design

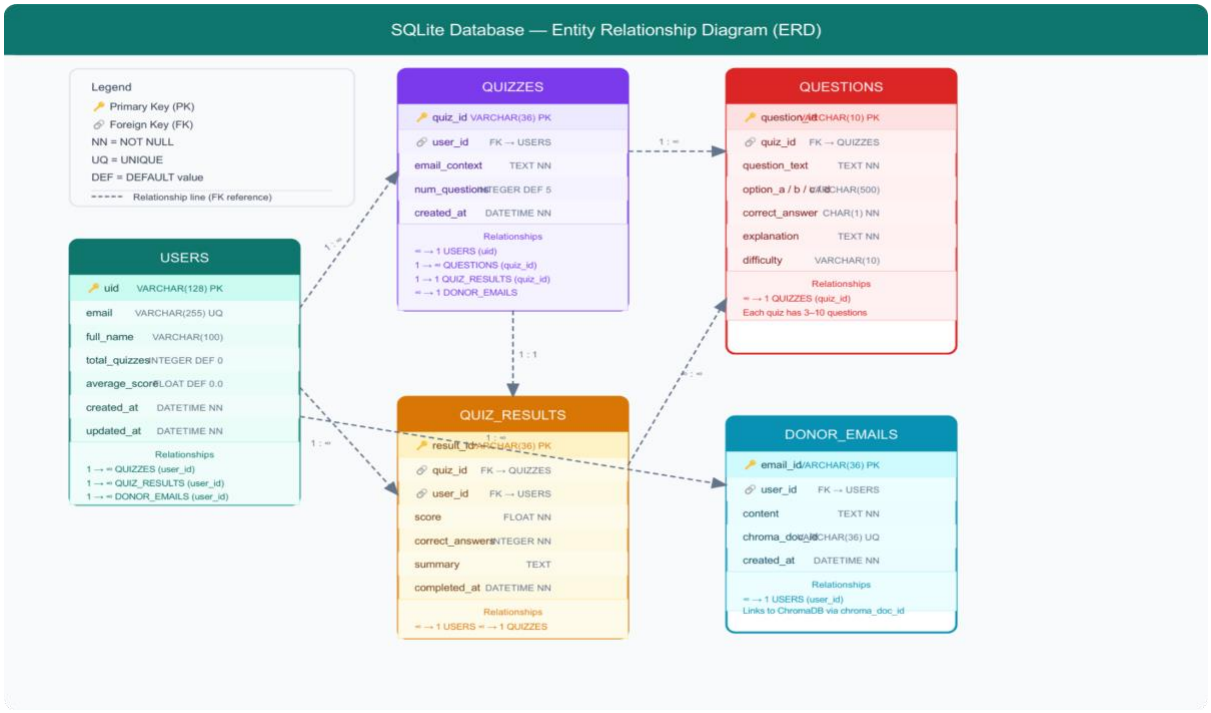
## 3.1 Data Model

**QuizBot** uses a **multi-database strategy** - Firebase Realtime Database for cloud-synced user and quiz data, SQLite for local structured storage, and ChromaDB for vector embeddings. Each database is chosen for its specific strengths.

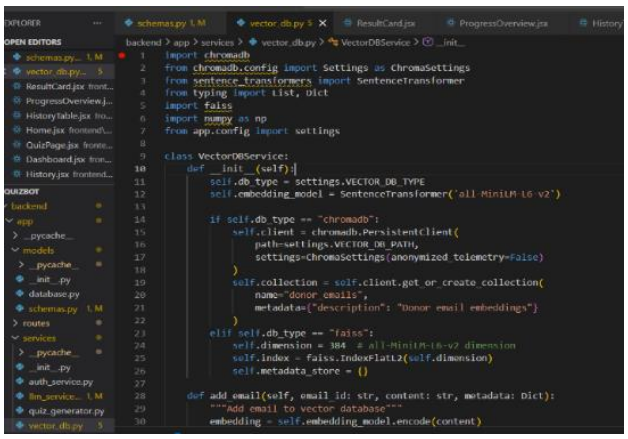
3.1.1 **Firebase Realtime Database Schema** Firebase stores data as a hierarchical JSON tree. QuizBot uses two root nodes - users and quiz_history.



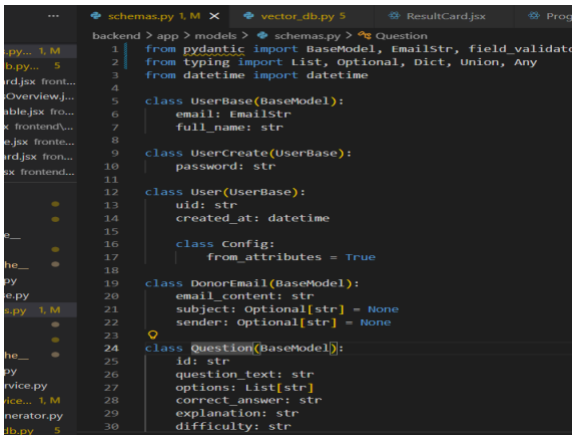
### 3.1.2 SQLite Database Schema



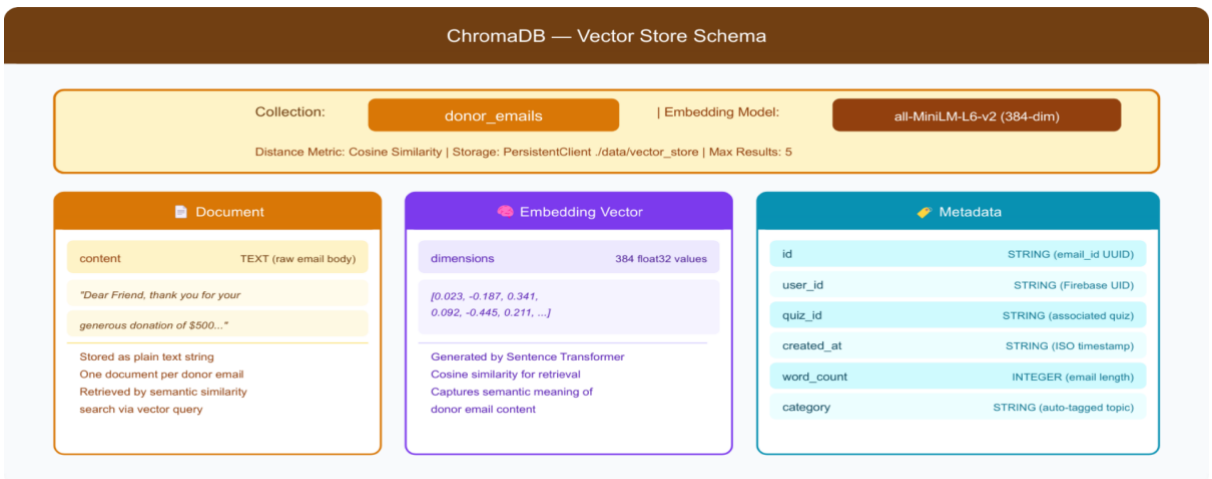

### 3.1.3 ChromaDB Vector Store Schema



```python
import chromadb
from chromadb.config import Settings as ChromaSettings
from sentence_transformers import SentenceTransformer
from typing import List, Dict
import faiss
import numpy as np
from app.config import settings

class VectorDBService:
    def __init__(self):
        self.db_type = settings.VECTOR_DB_TYPE
        self.embedding_model = SentenceTransformer('all-MiniLM-L6-v2')

        if self.db_type == "chromadb":
            self.client = chromadb.PersistentClient(
                path=settings.VECTOR_DB_PATH,
                settings=ChromaSettings(anonymized_telemetry=False)
            )
            self.collection = self.client.get_or_create_collection(
                name="donor_emails",
                metadata={"description": "Donor email embeddings"}
            )
        elif self.db_type == "faiss":
            self.dimension = 384  # all-MiniLM-L6-v2 dimension
            self.index = faiss.IndexFlatL2(self.dimension)
            self.metadata_store = {}

    def add_email(self, email_id: str, content: str, metadata: Dict):
        """Add email to vector database"""
        embedding = self.embedding_model.encode(content)
```

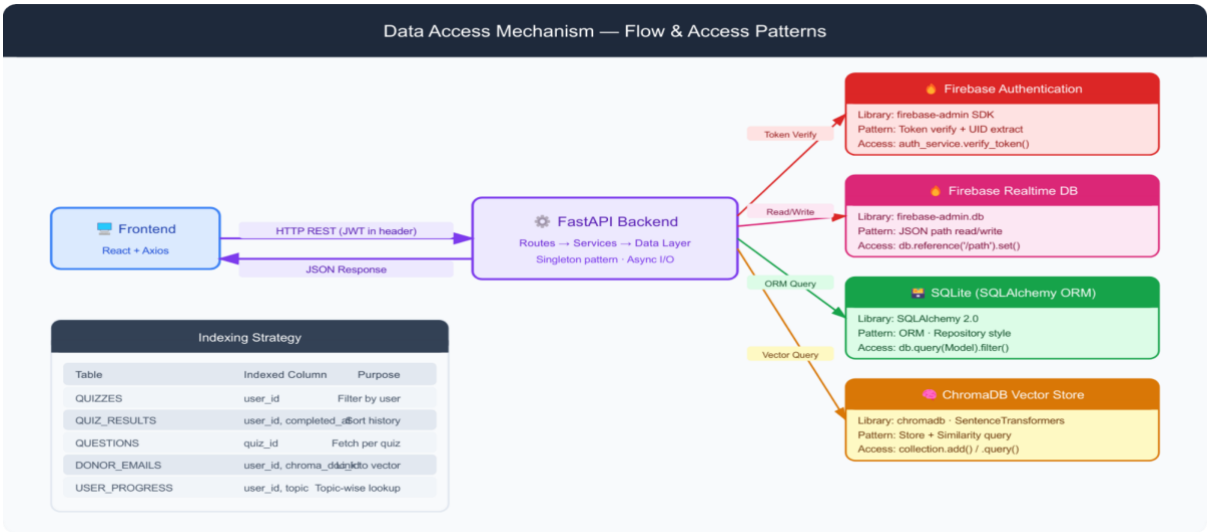


```python
from pydantic import BaseModel, EmailStr, field_validat
from typing import List, Optional, Dict, Union, Any
from datetime import datetime

class UserBase(BaseModel):
    email: EmailStr
    full_name: str

class UserCreate(UserBase):
    password: str

class User(UserBase):
    uid: str
    created_at: datetime

    class Config:
        from_attributes = True

class DonorEmail(BaseModel):
    email_content: str
    subject: Optional[str] = None
    sender: Optional[str] = None

class Question(BaseModel):
    id: str
    question_text: str
    options: List[str]
    correct_answer: str
    explanation: str
    difficulty: str
```

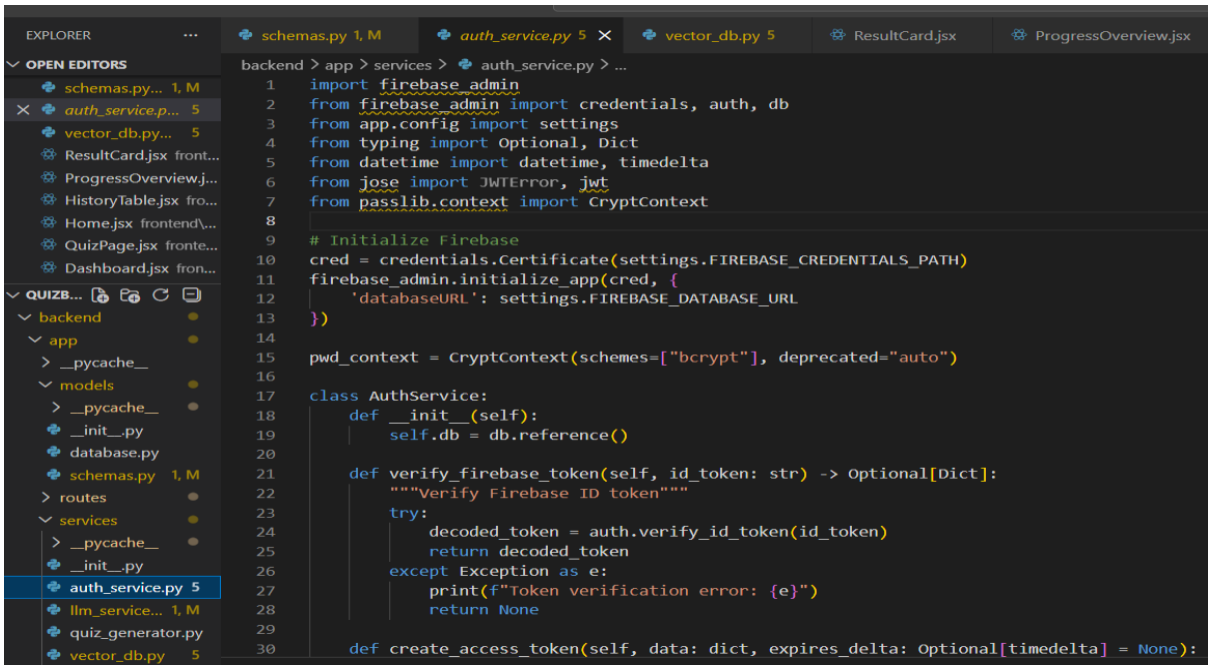


## 3.2 Data Access Mechanism

**QuizBot** accesses data through distinct mechanisms per database, all orchestrated by the FastAPI backend services.



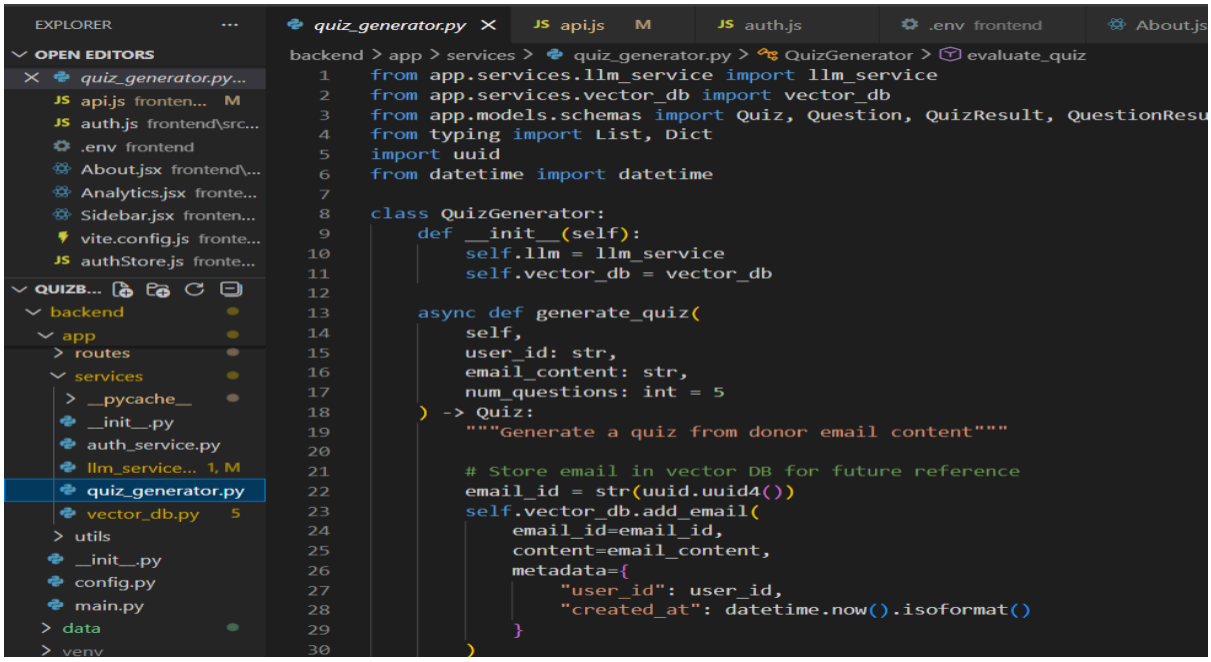

### 3.2.1 Firebase Realtime Database Access




**Access Rules:**

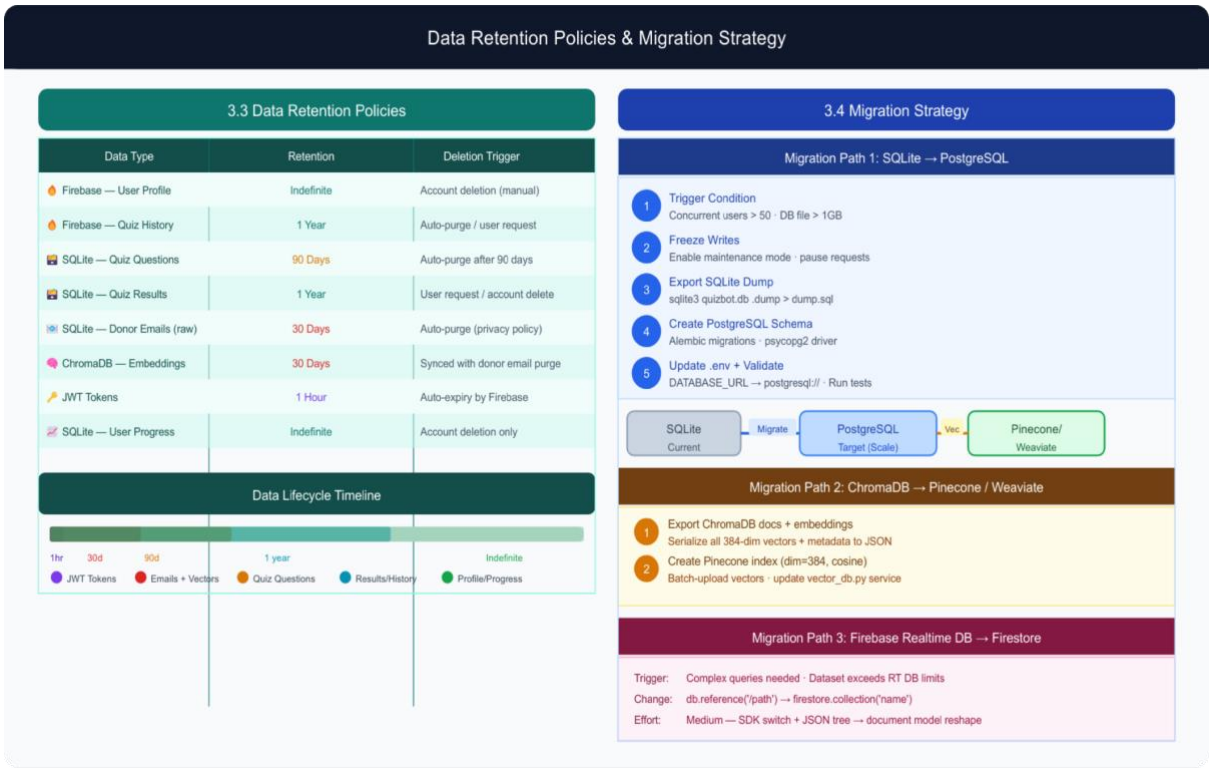
- All reads/writes are server-side only via Firebase Admin SDK
- Frontend never directly reads/writes the Realtime DB — always goes through FastAPI
- Firebase Security Rules set to deny all direct client access

### 3.2.2 ChromaDB Vector Access




## 3.3 Data Retention Policies

## 3.4 Data Migration



# 4. Interfaces

## 4.1 User Interfaces (UI)

**QuizBot's** frontend is a Single Page Application (SPA) built with React 19 and Vite. All interfaces are responsive, accessible, and transition-animated via Framer Motion.

**Interface Inventory**

| Interface | Route | Access | Purpose |
|---|---|---|---|
| **Home Page** | /home | Public | Landing page, product overview, CTA |
| **Login** | /login | Public | Firebase email/password login |
| **Register** | /register | Public | New user account creation |
| **Quiz Page** | /quiz | Protected | Email input + quiz generation + taking |
| **Dashboard** | /dashboard | Protected | Stats overview + recent quizzes |
| **History** | /history | Protected | All past quiz records |
| **Progress** | /progress | Protected | Analytics charts + achievements |

**UI Component Interfaces**

**EmailInput Component**

- Input: Large textarea for donor email paste
- Config: Dropdown to select number of questions (3 / 5 / 7 / 10)
- Action: "Generate Quiz" button → triggers POST /api/quiz/generate
- Output: Passes quiz object to QuizInterface

**QuizInterface Component**

- Input: Quiz object (quiz_id + questions array)
- Renders: Each question via QuestionCard
- Tracks: Selected answers in local state
- Action: "Submit Quiz" button → validates all answered → triggers POST /api/quiz/evaluate

**QuestionCard Component**

- Input: Single question object (text + 4 options + difficulty)
- Output: Selected option (A/B/C/D) emitted to parent QuizInterface
- Visual: Highlights selected option, shows difficulty badge

**ResultCard Component**

- Input: QuizResult object (score + results array + summary)
- Renders: Score percentage, AI summary, per-question explanation accordion
- Action: "Try Another Quiz" → resets quizStore and navigates to /quiz

**Dashboard Components**

- ProgressOverview: Displays total quizzes, average score, accuracy rate as stat cards + Recharts line graph
- HistoryTable: Paginated table of past quizzes with score, date, and view link

## 4.2 API Interface (Backend ↔ Frontend)

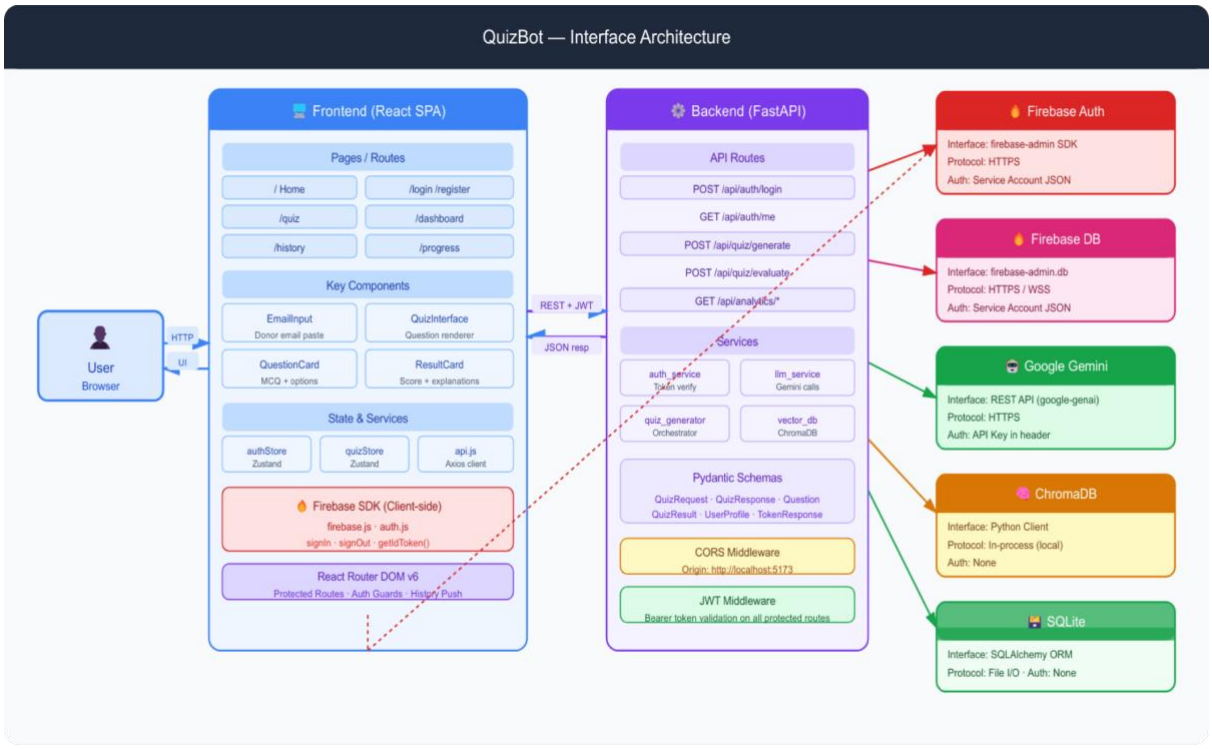
All API communication follows REST principles. Full catalogue covered in Section 2.6. Summary:

| Category | Base Path | Methods | Auth |
|---|---|---|---|
| Authentication | /api/auth | POST, GET | Partial |
| Quiz | /api/quiz | POST | Required |
| Analytics | /api/analytics | GET | Required |

## 4.3 External Service Interfaces

| Service | Interface Type | Protocol | Auth Method |
|---|---|---|---|
| **Google Gemini AI** | REST API | HTTPS | API Key in header |
| **Firebase Authentication** | SDK + REST | HTTPS | Service Account JSON |
| **Firebase Realtime DB** | SDK | HTTPS/WSS | Service Account JSON |
| **ChromaDB** | Python Client | In-process | None (local) |
| **SQLite** | SQLAlchemy ORM | File I/O | None (local file) |

### 4.4 Interface Flow Diagram




# 5. State and Session Management

## 5.1 Frontend State Management

QuizBot uses **Zustand** for global client-side state — lightweight, boilerplate-free, and React-hook based.

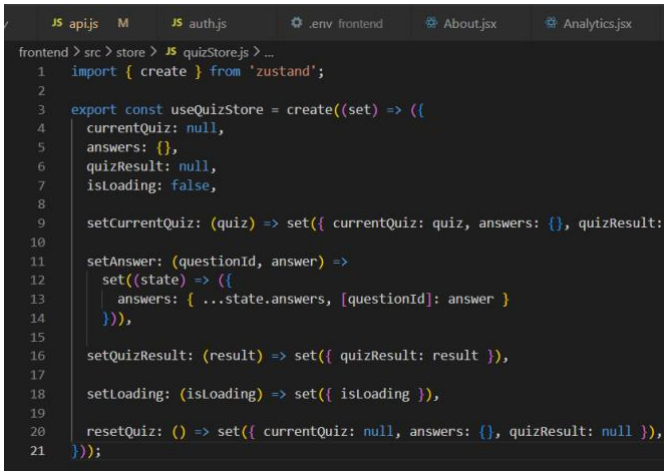
```
frontend > src > store > JS authStore.js > ...
1   import { create } from 'zustand';
2   import { persist } from 'zustand/middleware';
3
4   export const useAuthStore = create(
5     persist(
6       (set) => ({
7         user: null,
8         isAuthenticated: false,
9         isLoading: true,
10
11        setUser: (user) => set({ user, isAuthenticated: !!user,
12
13        logout: () => set({ user: null, isAuthenticated: false
14
15        setLoading: (isLoading) => set({ isLoading }),
16      }),
17      {
18        name: 'auth-storage',
19      }
20    )
21  );
```

**authStore.js**

Manages the authenticated user session on the frontend.

| State Field | Type | Description |
|---|---|---|
| user | Object | Firebase user object (uid, email, displayName) |
| token | String | JWT access token from backend |
| isAuthenticated | Boolean | True when logged in |
| isLoading | Boolean | Auth check in progress |

- **Actions:** setUser(), setToken(), logout(), checkAuth()
- **Persistence:** JWT stored in localStorage. On app load, checkAuth() reads the stored token and re-validates with Firebase before restoring the session.




**quizStore.js**

Manages the active quiz session on the frontend.

| State Field | Type | Description |
|---|---|---|
| currentQuiz | Object | Full quiz object (quiz_id + questions) |
| selectedAnswers | Object | Map of question_id → selected option |
| quizResult | Object | Evaluation result after submission |
| isGenerating | Boolean | Quiz generation in progress |
| isSubmitting | Boolean | Evaluation in progress |

**Actions:** setQuiz(), setAnswer(), setResult(), resetQuiz()

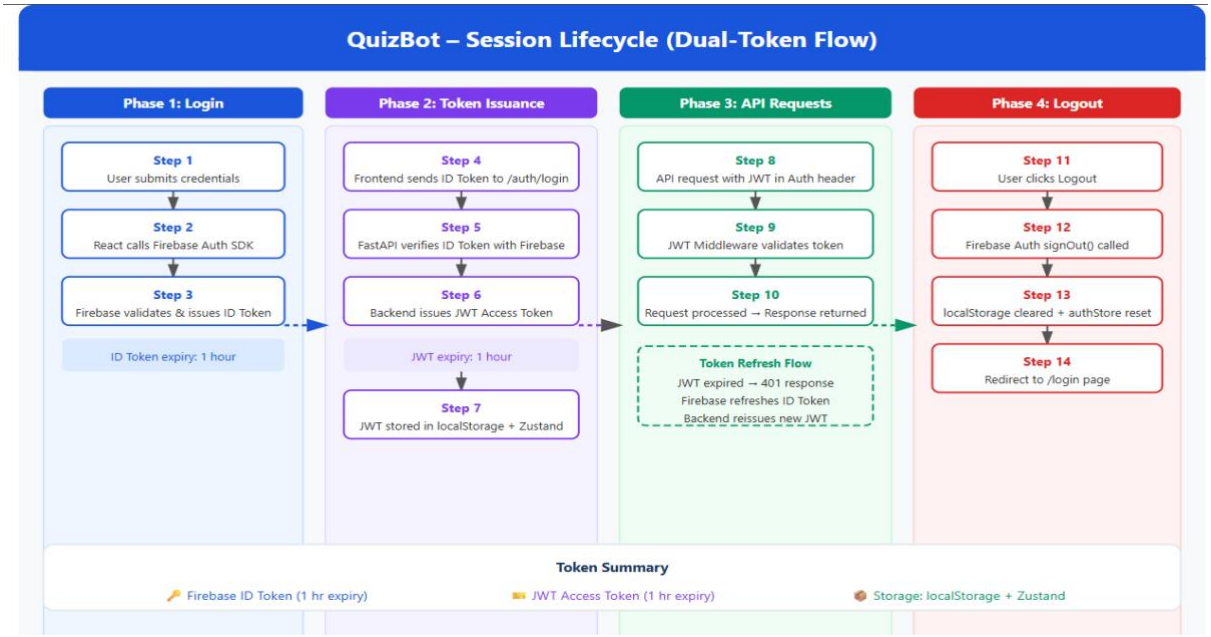
## 5.2 Session Management

QuizBot uses a **dual-token session model** — Firebase ID Token for authentication and a backend JWT for API authorization.

**Session Security Rules**

- JWT tokens expire after **1 hour** — matching Firebase ID token lifetime
- Tokens are **never stored in cookies** — localStorage only (no CSRF risk)
- Backend verifies **every request** — no session state held server-side (stateless)
- Frontend **Axios interceptor** automatically attaches the latest valid token to every outgoing request
- If token refresh fails — user is silently logged out and redirected to /login

**Session Lifecycle**




# 6. Caching

QuizBot is a **v1.0 application** with no dedicated caching layer implemented yet. However, the following caching mechanisms exist implicitly and are documented for future planning.

## 6.1 Current Implicit Caching

| Cache Type | Location | What Is Cached | Duration |
|---|---|---|---|
| **JWT Token** | localStorage | Auth token | 1 hour (token lifetime) |
| **Quiz State** | Zustand (memory) | Current quiz + answers | Session duration |
| **Auth User** | Zustand (memory) | User profile + UID | Session duration |
| **Vite Build Cache** | Browser cache | Static JS/CSS bundles | Until new deploy |
| **Firebase SDK Cache** | Browser memory | Firebase app instance | Session duration |
| **Singleton Services** | Backend memory | llm_service, vector_db | App lifetime |

## 6.2 Backend Singleton Caching

FastAPI initialises the following services **once at startup** and reuses them across all requests - acting as in-memory singletons:

| Singleton | Initialized In | What It Holds |
|---|---|---|
| llm_service | main.py startup | Gemini client + config |
| vector_db | main.py startup | ChromaDB client + collection reference |
| auth_service | main.py startup | Firebase Admin app instance |
| SessionLocal | database.py | SQLAlchemy connection pool |

# 7. Non-Functional Requirements

## 7.1 Security Aspects

### 7.1.1 Authentication Security

| Requirement | Implementation | Status |
|---|---|---|
| Secure login | Firebase Authentication (email + password) | Implemented |
| Token-based auth | JWT via python-jose (1hr expiry) | Implemented |
| Server-side verification | Firebase Admin SDK token validation on every request | Implemented |
| Token refresh | Frontend auto-refresh via getIdToken(true) | Implemented |
| Logout cleanup | localStorage cleared + Zustand reset | Implemented |
| Password hashing | Managed by Firebase Auth (bcrypt internally) | Implemented |

### 7.1.2 API Security

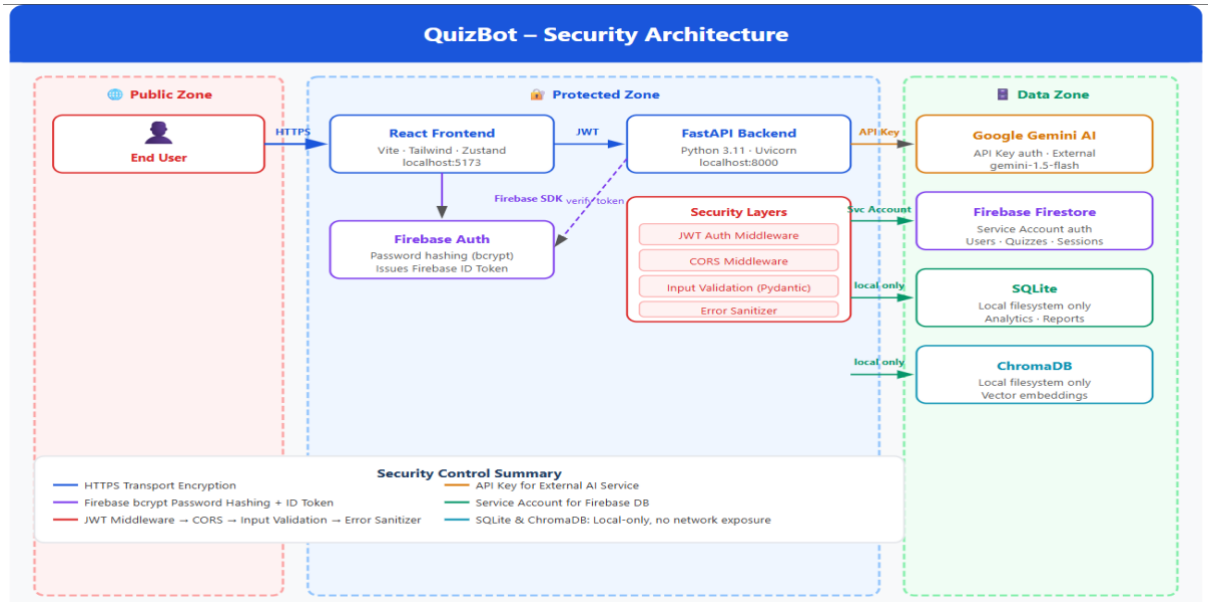
| Requirement | Implementation | Status |
|---|---|---|
| CORS policy | Backend allows only http://localhost:5173 | Implemented |
| Protected routes | JWT middleware on all non-public endpoints | Implemented |
| Input validation | Pydantic V2 schema validation on all request bodies | Implemented |
| Rate limiting | Not yet implemented | Planned |
| HTTPS enforcement | Required in production deployment | Production |

### 7.1.3 Data Security

| Requirement | Implementation |
|---|---|
| API key protection | All keys stored in .env files, never hardcoded |

| | |
|---|---|
| Firebase credentials | Service account JSON stored server-side only |
| Email content privacy | Donor emails retained for 30 days then purged |
| No PII in vectors | ChromaDB stores only non-reversible embedding vectors |
| DB access control | SQLite and ChromaDB are local — no external exposure |
| Firebase DB rules | Configured to deny all direct client access (server-only) |

### 7.1.4 Security Architecture Diagram




## 7.2 Performance Aspects