2026 Volume: 02 Issue: 01 www.ijrpa.com ISSN 2456-9995 Review Article

International Journal Research Publication Analysis

Page: 01-34

AI BASED JOB RECOMMENDATION APP

***Hemachandran S. B.**

Bachelor of Technology in Artificial Intellegence and Machine learning Sri Shakthi Institute of Engineering and Technology, Anna University: Chennai 600025.

Article Received: 05 December 2025
Article Revised: 25 December 2025
Published on: 13 January 2026

***Corresponding Author: Hemachandran S. B.**
Bachelor of Technology in Artificial Intellegence and Machine Learning Sri Shakthi Institute of Engineering and Technology Anna University: Chennai 600025.

DOI: https://doi-doi.org/101555/ijrpa.4295

ABSTRACT

The AI-Driven Career Intelligence and Job Recommendation Platform is a mobile-first, cloud-enabled system that transforms how job seekers discover opportunities and close skill gaps through personalized, AI-powered guidance. Traditional job search tools surface generic listings and provide limited insight into how a candidate's background aligns with role requirements. This platform addresses those limitations by combining sophisticated AI search, natural language understanding, and resume analysis to deliver contextualized job matches and actionable career advice. Job opportunities are discovered and indexed using an AI search layer that normalizes and enriches listings with semantic metadata—skills, experience levels, salary bands, and role specializations—enabling accurate matching against user profiles. Users create rich profiles and upload resumes (PDF/DOCX); an LLM-driven resume analyzer extracts skills, projects, education, and experience, storing structured representations for fast comparison. Bookmarked jobs act as a curated portfolio: the system performs a multi-job skill-gap analysis to compute match scores, prioritize missing competencies, and generate targeted learning recommendations with estimated effort, resources, and project ideas. Architecturally, the platform follows modular service principles: a Node.js/Express backend exposes RESTful APIs for profile management, job indexing, bookmarking, and AI analysis; Supabase/Postgres provides persistent storage (including JSONB fields for flexible resume analysis), while the React Native frontend (Expo) delivers a responsive, offline-friendly UX using AsyncStorage and intelligent caching. Security and reliability are enforced via authenticated endpoints, input validation, rate limiting for AI calls, and resilient error handling. By integrating AI-driven discovery with personalized, resume-aware guidance, the platform moves beyond passive job listings to a proactive career

development tool—helping candidates discover relevant roles, understand precise gaps, and follow prioritized learning paths that improve employability and accelerate career progression.

CHAPTER 1 INTRODUCTION

In contemporary job markets, job discovery and career development remain fragmented across multiple platforms that surface generic listings with limited personalization. Job seekers often face difficulty finding roles that align with their precise skills, experience, and career goals, while employers struggle to connect with suitably qualified candidates. This conventional approach produces inefficiencies, missed opportunities, and an unclear path for upskilling—creating demand for an intelligent system that unifies role discovery with personalized career guidance.

The **AI-Based Job Recommendation Platform** addresses these gaps by providing a unified, mobile-first solution that couples contextual job discovery with resume-aware, AI-powered career recommendations. Rather than relying on simple keyword matching, the system uses an AI search layer to normalize and enrich job listings with semantic metadata (skills, experience bands, specialties, salary ranges) and a resume analysis pipeline to extract structured candidate profiles from uploaded PDFs/DOCX. Bookmarked jobs serve as a curated set for multi-role comparison: the platform computes match scores, identifies missing competencies, and generates prioritized, actionable learning recommendations—complete with estimated effort, curated resources, and practical project ideas.

Architecturally, the platform adopts a modular, service-oriented design to ensure scalability and maintainability. A **Node.js/Express** backend exposes RESTful APIs for user/profile management, job indexing, bookmarking, and AI analysis, while **Supabase (Postgres)** provides persistent storage with JSONB fields for AI-derived resume data. The **React Native frontend** (Expo) delivers an offline-friendly, responsive experience using AsyncStorage and intelligent caching. AI capabilities are provided via an **LLM-based service** (Gemini), used for resume parsing, skill extraction, and skill-gap analysis. Security, reliability, and performance are enforced through authenticated endpoints, input validation, rate limiting for AI calls, database indexing, and multi-level caching strategies.

By integrating AI-driven discovery with resume-aware guidance and persistent bookmarking, the platform transforms the job search from passive browsing into proactive career planning. It empowers users to discover relevant roles, understand precise gaps, and follow prioritized learning paths that increase employability—bridging the gap between current capabilities and

target positions through data-driven, personalized recommendations.

CHAPTER 2 LITERATURE REVIEW

The literature on intelligent job recommendation and career-assistance platforms highlights persistent shortcomings of conventional job search systems—keyword-based matching, fragmented candidate signals (resume text, skills lists, application history), and limited guidance on how candidates can bridge gaps. Early recommender systems applied collaborative filtering and simple content-based matching; more recent work emphasizes hybrid approaches that combine semantic content models with behavioral signals to improve relevance and personalization.

Semantic search and embedding-based retrieval have become central to modern job discovery. Studies show that dense vector embeddings (sentence/document encoders) and nearest-neighbor search outperform lexical matching for role-to-profile similarity by capturing synonyms, paraphrases, and implicit skill mentions. Research also demonstrates value in enriching job and candidate representations with extracted structured attributes (skills, seniority, technologies) before matching—an approach reflected in this project's resume parsing and job enrichment pipeline

Large language models (LLMs) and specialized NLP pipelines are increasingly used for resume parsing, skill extraction, and generation of actionable guidance. Empirical work finds LLMs effective at extracting entities and generating human-readable recommendations, but warns about hallucinations and inconsistent formats; robust prompt design, JSON-constrained outputs, and validation layers mitigate these risks. This project follows those best practices by using templated prompts and JSON parsing in the AI controller and exposing AI endpoints consumed by the frontend

Skill-gap analysis research often couples per-job parsing (requirements extraction) with candidate profile embeddings to compute match scores and prioritize missing competencies. Evaluation studies report that multi-job aggregation (analyzing bookmarked sets) yields more actionable learning priorities than single-role comparisons because frequency and cross-role importance surface naturally. The project implements multi-job comparison and prioritized recommendations in the bookmarks AI workflow

System design literature stresses modular, service-oriented architectures for AI-driven platforms. Separating concerns—indexing/search, profile management, AI analysis, and UI—enables independent scaling, reproducible pipelines, and safer experimentation. Techniques such as JSONB storage for flexible AI outputs, background jobs for long-running AI tasks,

rate limiting for model calls, and multi-level caching (frontend AsyncStorage + backend cache) are recommended; these patterns appear in the project architecture (profile storage and AI endpoints in the backend, frontend caching in AsyncStorage).

Privacy, fairness, and robustness are recurring themes. Studies document risks from biased training data and recommend transparency (explainable recommendations), user control (resume edits, opt-outs), and secure handling of sensitive PII (encrypted transport, least privilege keys). The project's design accounts for these concerns via authenticated endpoints, file-type/size checks on resume uploads, and controlled AI prompts—areas to monitor and harden further during deployment

For AI guidance, human expert validation is advised to calibrate model suggestions and ensure recommended learning resources are realistic and high quality. Incorporating these evaluation loops will strengthen the platform's reliability and user trust.

In summary, the convergent findings across the literature support the platform's hybrid approach: semantic AI search and embeddings for discovery, structured resume parsing and JSON-safe AI outputs for analysis, multi-job aggregation for prioritized learning recommendations, and modular engineering patterns for scalability and safety. Continued attention to evaluation, bias mitigation, and secure handling of resume data will be critical for real-world impact.

CHAPTER 3

SYSTEM ANALYSIS

3.1. EXSISTING SYSTEM:

The existing job recommendation and career-assistance landscape is fragmented and largely dominated by disparate portals and minimalistic matching engines that rely on keyword or rule-based filters. Candidate signals—resumes, skill lists, application history, and bookmarks—are often stored and processed in isolated silos or as unstructured files, preventing robust, profile-aware matching and cross-role analysis. Resume handling is typically superficial: files are saved as blobs or URLs without structured parsing, so comparisons between a candidate's qualifications and job requirements remain shallow and error-prone. Recommendation logic is mostly lexical (keyword matching) or simple heuristics, resulting in low relevance, duplicate listings, and poor prioritization of skills for upskilling.

User experience suffers from inconsistent state persistence, limited offline support, and brittle

navigation flows that cause data reloads or empty listings when users navigate between screens. Bookmarking and saved-job workflows are ad-hoc, lacking reliable deduplication, centralized analytics, or multi-job comparison features that could produce actionable learning plans. System-level shortcomings include limited observability, minimal AI governance, and no standardized schema for enriched job or resume metadata (e.g., normalized skill taxonomies, JSONB resume analysis). Security and operational controls are often basic or missing—file uploads, AI calls, and sensitive profile data are not consistently validated, rate-limited, or audited—reducing reliability and trust.

Consequently, the current systems do not support persistent, resume-aware recommendations, prioritized skill-gap analysis, nor seamless, AI-driven career guidance; they are inadequate for delivering personalized, actionable pathways that help users close gaps and improve employability.

Proposed System:

The proposed system is an AI-driven, cloud-native career intelligence platform that unifies job discovery, resume analysis, and personalized upskilling guidance under a single interoperable framework. Built on a modular microservices architecture (Node.js/Express backend, Gemini LLM service for AI analysis, Supabase/Postgres for persistent storage with JSONB for flexible AI outputs), the system performs semantic job indexing via an AI search layer, structured resume parsing, and bookmark-driven multi-job skill-gap analysis. Advanced NLP and embedding techniques normalize job and profile metadata (skills, seniority, location, salary bands) to enable precise matching and prioritized recommendations. The mobile frontend (React Native/Expo) offers responsive, offline-friendly UX with AsyncStorage caching and focus-aware refresh; role-aware dashboards present bookmarked jobs, match scores, and AI-generated learning plans. Operational concerns—authenticated endpoints, file validation for resume uploads, rate limiting for AI calls, structured JSON outputs to avoid hallucinations, and database indexing—ensure security, reliability, and scalability. By combining AI search, resume-aware analysis, and actionable learning recommendations, the system converts passive job listings into a proactive career development tool that improves relevance, reduces friction, and accelerates employability.

System Requirements:

- **Functional Requirements:**
- User management
  - Sign up / sign in via Supabase Auth (email/password, third-party optional).
  - Profile CRUD: name, location, experience, skills, preferences.
- Resume management
  - Upload resume (PDF/DOCX), validate type/size, store secure URL in profiles.
  - Parse resume text and store structured analysis (skills, experience, education) in profiles.resume_text and profiles.resume_analysis (JSONB).
- Job ingestion & indexing (AI search)
  - Index job listings via AI search/enrichment layer (extract skills, seniority, location, salary band, description embeddings).
  - Provide endpoints to search/filter jobs by keywords, skills, location, specialization, salary.

- Bookmarks
  - Bookmark/unbookmark job (persist to bookmarks table with UNIQUE(user_id, job_id)).
  - Get bookmarks, get bookmark count, check bookmark status.
- Job details & persistence
  - Ensure job details load by job db-id (UUID) and persist state on navigation.
  - When bookmarking or opening a job not yet stored, store job in DB first and return DB UUID.
- AI skill-gap analysis
  - Multi-job analysis comparing profiles.resume_analysis vs bookmarked jobs; return per-job match %, missing skills, prioritized recommendations.
  - Use Gemini API key from .env for LLM calls.
- UI features
  - Dashboard (suggested/recent jobs), job-search, job-details, bookmarks (third tab), profile with resume upload and AI results.
  - Bookmark icon toggle persists and highlights across pages.
  - See All/View All shows full set (no arbitrary slice).
  - Search bar queries DB (not just cached data).
  - Pull-to-refresh, offline cache via AsyncStorage, focus-aware reload using useFocusEffect.
- Administrative / maintenance

- Migration scripts for DB columns and bookmarks table.
- Logging endpoints and health check.

- **Non-Functional Requirements:**
- Security & privacy
  - Authenticate all protected endpoints via Supabase JWT.
  - Enforce file validation and size limit (e.g., ≤10MB).
  - Store GEMINI_API_KEY in backend .env (do not expose to frontend).
  - Use HTTPS/TLS in production; use row-level security for Supabase where needed.
- Performance & scalability
  - API latency targets: <300ms for simple queries; AI tasks async with progress (10–30s).
  - Support concurrent users (scale Node services horizontally; DB connection pooling).
  - Caching: backend cache TTL and frontend AsyncStorage for instant loads.
- Reliability & availability
  - Graceful degradation: show cached data if AI or search is unavailable.
  - Retry/backoff for external AI calls; circuit breaker for excessive failures.

- Maintainability & observability
  - Modular services: separate AI controller/service, jobs controller, bookmarks controller.
  - Structured logs, request tracing, error reporting (Sentry or similar).
- Usability & accessibility
  - Responsive React Native UI; clear states for loading/error/empty.
  - Accessible labels for buttons/icons; readable fonts and color contrast.
- Data quality
  - Deduplicate skills (case-insensitive) on frontend and backend.
  - Validate job and resume data before storage; normalize skill taxonomy where possible.
- Costs
  - Optimize AI calls (batching, caching) to limit Gemini usage and costs.

- **Feasibility Analysis:**
  - Technical Feasibility:

Proven stack (Node.js, Express, Supabase/Postgres, React Native, Gemini LLM). Resume parsing and skill extraction achievable via LLM + regex, with prompt tuning to reduce errors. Multi-job analysis feasible using aggregated requirements and embedding similarity.

- Operational Feasibility:

Requires API key and quota management, plus background processing for long AI tasks. Supabase simplifies database and authentication management.

- Economic Feasibility:

Moderate development and hosting cost; AI usage costs can be optimized through caching, batching, and rule-based fallbacks.

- Legal & Privacy Feasibility:

Resume storage must follow privacy laws; user consent, data deletion, and export options are mandatory.

- **Problems in the Existing System:**

- Low Relevance Search: Keyword-based matching ignores context, synonyms, and semantic similarity.
- Unstructured Resume Data: Resumes stored as raw files or URLs without parsing hinder accurate job matching.
- Poor State Management: Navigation causes data loss; listings vanish due to inconsistent caching.
- Duplicate Data Issues: Duplicate skills, jobs, and bookmarks appear across UI and database.
- Weak AI Reliability: No centralized AI service, rate limiting, or proper error handling for model failures.
- Stale Backend Cache: Cached data not refreshed dynamically, requiring manual reloads.
- No Actionable Insights: Missing skill-gap and learning path recommendations based on resumes.

3.2. PROPOSED METHOD:

The proposed system is an AI-first, Career Intelligence Platform that unifies job discovery, resume-aware matching, and personalized upskilling guidance in a single interoperable framework. It relies on a modular microservices architecture (**Node.js / Express for APIs**, a dedicated **Gemini LLM service** for semantic enrichment and analysis, and **Supabase/Postgres** for persistent storage with JSONB fields for flexible AI outputs). Job opportunities are indexed by an AI search layer that generates embeddings and semantic metadata (skills, seniority, salary band, role specializations), enabling accurate, context-aware matching against structured user profiles derived from parsed resumes (PDF/DOCX). Bookmarked jobs constitute the user's curated set for multi-role skill-gap analysis: the system computes per-job match scores,

aggregates missing competencies, and generates prioritized, actionable learning recommendations (learning time, curated resources, project ideas).

The mobile frontend **(React Native / Expo)** provides an offline-friendly UX (AsyncStorage caching, focus-aware reload) with intuitive screens for Dashboard, Job Search, Bookmarks (AI recommendations), and Profile (resume upload & analysis). Security and governance include Supabase Auth (JWT), file validation and size limits for resume uploads, rate-limiting for AI endpoints, and structured JSON outputs to reduce LLM hallucinations. Operational features—background tasks for long-running AI jobs, multi-level caching, database indexing, and clear observability/logging—ensure performance and reliability. This architecture converts passive job listings into a proactive career development tool that helps users find relevant roles, understand skill deficits, and follow prioritized learning paths to improve employability.

Features of the proposed system

- **AI Search & Semantic Indexing:** LLM-driven embedding and metadata extraction for role normalization and precise matching.
- **Resume Parsing & Structured Storage:** PDF/DOCX parsing → resume_text + resume_analysis (JSONB) stored in profiles.
- **Bookmark-driven Skill-gap Analysis:** Multi-job aggregation producing match %, missing skills, prioritized learning plans.
- **Persistent, Responsive UI:** AsyncStorage caching, useFocusEffect reloads, instant navigation and offline access.
- **Secure Resume Upload:** File-type/size validation, authenticated endpoints, sensitive data controls.
- **Scalable Microservices:** Clear separation of indexing, AI, job, and profile services for independent scaling and maintenance.
- **Reliability & Cost Controls:** Background processing for AI, caching and batching to reduce LLM usage and latency.
- **Observability & Governance:** Structured logging, rate-limiting, input validation, and JSON-schema validation for AI outputs.
- **Extensible APIs:** REST endpoints for jobs, bookmarks, profiles, and AI analysis (ready for integration or future GraphQL support).

3.3. ADVANTAGES OVER THE EXISTING SYSTEM

Unified Candidate and Job Representation

Unlike fragmented job portals, the proposed platform consolidates user profiles, resumes, bookmarks, and job metadata into a single standardized representation, enabling consistent, accurate comparisons and eliminating siloed data workflows.

AI-Driven Resume Parsing and Enrichment

Automated resume analysis extracts structured skills, projects, and experience from uploaded documents and normalizes job requirements, replacing manual parsing and superficial keyword matching used in legacy systems.

Contextual, Multi-Job Skill-Gap Analysis

The system performs aggregated comparisons across a user's bookmarked roles to compute match scores, surface recurring missing competencies, and produce prioritized learning plans—moving beyond single-role heuristics to actionable career guidance.

Personalized, Semantic Matching

Semantic matching (embeddings and contextual understanding) yields more relevant job recommendations than lexical filters, capturing synonyms, implied skills, and seniority nuances that keyword systems miss.

Automated, Actionable Recommendations

Instead of passive lists, the platform generates prioritized learning recommendations with estimated effort, curated resources, and project ideas—helping users close gaps and track progress toward roles.

Improved Data Quality and Deduplication

Centralized validation and deduplication (skills, bookmarks, job entries) prevent inflated counts and inconsistent records, ensuring analytics and counts reflect true user intent.

Scalable, Cost-Conscious AI Usage

AI operations are batched, cached, and performed asynchronously where needed to balance responsiveness and cost, avoiding per-request overhead that makes LLM-driven features impractical in naive implementations.

Extensibility and Maintainability

A modular design separates indexing, profile management, bookmarking, and AI analysis so features can evolve independently, enabling incremental improvements without disrupting user workflows. In short, the proposed solution replaces ad-hoc, keyword-centric job discovery with a unified, resume-aware, AI-driven career guidance platform that delivers higher relevance, clearer actionability, and a more reliable user experience.

CHAPTER 4 SYSTEM SPECIFICATION

This chapter outlines the technical and functional specifications of the **AI-Based Job Recommendation Platform**. It details the architectural design, core modules, system requirements, and technologies used to implement resume-aware matching, bookmark-driven skill-gap analysis, and AI search.

User Authentication & Access Control

- **Roles:** End User (job seeker), Admin, Support/Moderator, AI Worker
- **Authentication: Supabase Auth (JWT)** for user sign-in / sign-up; token validation for API access
- **Session & Secrets:** Secure session handling, short-lived tokens, Gemini API key and service keys stored in backend and not exposed to client
- **Privacy controls:** User controls for resume upload/delete and consent for AI analysis

Data Ingestion & Processing Pipeline

- **Job discovery:** AI search/enrichment layer accepts job feeds or API input, generates semantic metadata and embeddings (no direct scraping asserted)
- **Job storage zones:**
  - Raw / Ingested: raw job payloads (audit trail)
  - Enriched: normalized job metadata (skills, seniority, location, salary-band, embeddings)
  - Indexed: search/embedding index for fast retrieval
- **Message broker & background tasks:**
  - Redis + BullMQ (or RabbitMQ) for background jobs (resume parsing, async AI analysis, indexing)
  - Background workers perform long-running LLM calls and DB writes
- **Caching:**
  - Backend cache (Redis) for frequent queries and AI results (TTL configurable)

- Frontend cache (AsyncStorage) for instant UX and offline resilience

AI / ML Module

- **LLM service:**

- Uses Gemini (API key) for resume parsing, skill extraction, and skill-gap recommendation
- Prompt templates, JSON-constrained outputs, and schema validation to reduce hallucinations

- **Embeddings & semantic search:**

- Generate embeddings for jobs and profile summaries; use vector index (e.g., Redis Vector, Milvus, or Pinecone) for similarity search

- **Skill-gap algorithms:**

- Compare resume_extracted_skills vs job_required_skills
- Compute per-job match percentage, aggregate missing skills, rank by frequency/priority
- Generate actionable recommendations (learning time, resources, projects)

- **Safety & governance:**

- Output validation with JSON Schema
- Rate limiting, retry/backoff for external LLM calls
- Optional human-in-loop validation for recommendations

Core Modules & Endpoints

- Profiles
  - **CRUD** profile, skills deduplication, resume upload endpoint
  - Columns: resume_url, resume_text (text), resume_analysis (JSONB)
- Jobs
  - Store jobs with normalized metadata and embeddings
- Observability & Admin
  - Health check, metrics, admin analytics endpoints

Metadata & Storage Management

- Primary DB: **PostgreSQL (Supabase)** with JSONB fields for flexible AI outputs
- Search & vector store: Redis Vector / Milvus / Pinecone for embeddings
- File storage: secure object store (Supabase storage or S3/MinIO) for resumes and logos
- Indexes: user_id, job_id, created_at, and relevant GIN indexes on JSONB fields for fast

queries

- Backups & migration: migration scripts for schema changes (resume columns, bookmarks table), regular DB backups

Visualization & Client Access

- Frontend (mobile): **React Native (Expo)** for iOS/Android apps
  - Screens: Dashboard, Job Search, Job Details, Bookmarks (AI recommendations), Profile (resume upload)
  - UX: pull-to-refresh, focus-aware reloads **(useFocusEffect)**, offline cache **(AsyncStorage)**
- API access:
  - RESTful endpoints for all core functions
  - Optional future **GraphQL** layer for aggregated queries and client efficiency

Security & Governance

- Transport: HTTPS/TLS for all endpoints
- Secrets: Gemini api key and service role keys kept server-side only
- File validation: accept only PDF/DOCX, enforce size limit (e.g., 10MB)
- Data protection: access controls for resume text, user-initiated delete/export options
- Audit & logging: audit trails for uploads, AI analysis requests, bookmark actions
- Rate limiting & quotas: per-user and per-endpoint throttling for AI calls
- Compliance: configurable data retention and deletion policies to support regulatory needs

Maintainability & Scalability

- Architecture:
  - Modular microservices (indexing/search service, profile service, AI worker service, jobs service)
  - Containerized with Docker; deployable to Kubernetes for horizontal scaling
- CI/CD: GitHub Actions for build/test
- Observability: Prometheus metrics, Grafana dashboards, centralized logs (ELK/Sentry), distributed tracing (Jaeger)
- Testing: unit, integration, and e2e tests; mock AI responses for CI stability

System Requirements

Server Configuration (example baseline)

- CPU: 4 vCPUs (min), 8+ vCPUs recommended for production AI workers
- RAM: 8 GB (min), 16–32 GB recommended for workers and search nodes
- Disk: 200 GB SSD (min) + object storage for resumes/media
- Network: 1 Gbps for backend services

Client Configuration

- Device: Modern smartphone (iOS/Android) or tablet
- OS: iOS 13+ / Android 9+
- Network: Recommended stable internet for AI features; offline features available for cached data

Software Requirements

- Backend: Node.js (14+), Express.js, BullMQ (Redis), Supabase/Postgres
- AI: Gemini API (LLM), vector store (Redis Vector / Milvus / Pinecone)
- Frontend: React Native (Expo), AsyncStorage, react-navigation
- DevOps: Docker, Kubernetes (optional), GitHub Actions, Helm
- Monitoring: Prometheus, Grafana, Jaeger, Sentry
- Databases & Storage: PostgreSQL (Supabase), Redis, Object store (S3/MinIO or Supabase storage)

Performance & Reliability Targets

- API latency: <300ms for simple queries; AI analysis handled asynchronously (expected 10–30s)
- Cache TTLs: configurable (default 30 minutes) for AI responses and popular searches
- Failure modes: degrade gracefully to cached results if AI service unavailable

This system specification provides the foundation to build a secure, scalable, and maintainable AI-driven career intelligence platform that unifies job discovery, resume analysis, and actionable upskilling guidance while ensuring data quality, privacy, and operational resilience.

CHAPTER 5 PROJECT DESCRIPTION

An AI-driven mobile career intelligence platform that combines resume-aware matching, semantic job search, and bookmark-driven skill-gap analysis to help users discover relevant

roles and close competency gaps. Users upload resumes (PDF/DOCX) which are parsed and stored as structured profiles; job listings are enriched with semantic metadata and embeddings. Bookmarked jobs are analyzed against the user's resume by an LLM (Gemini) to compute match scores, surface missing skills, and generate prioritized learning recommendations with resources and project ideas.

Key capabilities:

- Resume upload and AI parsing → structured resume_analysis (JSONB)
- Semantic job indexing and personalized job feeds
- Bookmark persistence with deduplication and count metrics
- Multi-job skill-gap analysis producing prioritized recommendations
- Secure, scalable backend (Node.js + Supabase/Postgres), React Native frontend with offline caching, and LLM integration (Gemini) for AI tasks

The platform converts passive job discovery into actionable career planning by delivering personalized insights that guide learning and improve employability.

Objectives:

- **To enable secure and structured resume management** — allowing users to upload resumes (PDF/DOCX), parse content, and store normalized resume analyses with extracted skills, education, and experience.
- **To develop an AI-driven skill extraction and job indexing system** — leveraging Gemini LLM for semantic parsing, metadata generation, and embedding-based job search.
- **To implement intelligent job recommendation and skill-gap analysis** — comparing user resumes against bookmarked jobs to compute match percentages, identify missing skills, and suggest personalized learning paths.
- **To ensure seamless and responsive mobile experience** — delivering an optimized React Native interface with offline caching, instant bookmark toggles, and persistent user state.
- **To maintain high data quality and system security** — through skill deduplication, strict input validation, Supabase JWT authentication, and secure key management.

- **To optimize performance, scalability, and cost-efficiency** — using caching, asynchronous AI processing, and batch request handling to reduce latency and API expenses.
- **To enable observability, maintainability, and continuous improvement** — with

structured logging, performance metrics, automated migrations, and CI/CD readiness for ongoing evaluation and iteration.

System Overview:

The system is an AI-first, mobile-centric Career Intelligence Platform that integrates resume parsing, semantic job indexing, bookmarking, and multi-job skill-gap analysis into a cohesive, production-ready architecture.

Data Ingestion Layer

- AI Search Ingest: Accepts job feeds, partner APIs, and manual job posts; an AI enrichment layer normalizes descriptions and extracts semantic metadata (skills, seniority, location, salary band).
- Resume Ingest: Secure file upload (PDF/DOCX) pipeline that forwards files to resume parsing workers.

Processing & Standardization Layer

- NLP & ETL pipelines transform raw job text and resume text into normalized artifacts: canonical skill tokens, experience-level tags, and structured JSONB resume_analysis.
- Validation stages remove duplicates, normalize casing, and enforce schema for AI outputs (JSON Schema).

AI/ML Analytics Layer

- Resume Parsing & Entity Extraction: LLM-based service (Gemini) converts resume text → skills, projects, experience, education.
- Embeddings & Semantic Search: Generate vector embeddings for jobs and profile summaries; vector index enables nearest-neighbor retrieval for matching.
- Skill-Gap & Recommendation Engine: Compares parsed resume vs bookmarked job requirements to compute match%, missing skills, and prioritized learning recommendations.
- Safety Controls: Prompt templates, JSON constraints, schema validation, retry/backoff, and optional human-in-loop verification.

Metadata & Storage Layer

- Primary DB: PostgreSQL (Supabase) with JSONB fields for resume_analysis and job enrichment.
- Vector Store: Redis Vector / Milvus / Pinecone for embedding search.
- Object Storage: Supabase storage / S3 / MinIO for resume files and logos.
- Caching: Redis for backend caches; AsyncStorage for frontend offline cache.
- Indexes: GIN on JSONB and B-tree on user/job keys for performant queries.

Visualization & Access Layer

- Mobile Frontend: React Native (Expo) delivering Dashboard, Job Search, Job Details, Bookmarks (AI recommendations), and Profile (resume upload).
- UX Features: Pull-to-refresh, focus-aware reloads, offline caching, and instant bookmark toggles.
- APIs: REST endpoints for jobs, profiles, bookmarks, and AI workflows; optional GraphQL for aggregated queries.

Modules Description

Authentication & Access Control Module

- Supabase Auth (JWT) for authentication; role-based controls for admin and support operations; secure server-side storage of GEMINI_API_KEY and service keys.

Data Quality Control (QC) Module

- Deduplication, schema validation, and data hygiene rules applied to skills and job metadata; QC metadata persisted with versioning.

AI/ML Processing Module

- Dedicated AI worker services for resume parsing, embedding generation, and skill-gap analysis; manages async jobs with Redis/BullMQ workers and exposes status endpoints.

Data Governance Module

- JSONB provenance, audit logs for uploads and AI calls, configurable retention/deletion policies, and export/delete options for user data.

Visualization & Reporting Module

- Mobile UI components for aggregated insights and per-job breakdowns; exportable recommendations and simple progress tracking for learned skills.

API & Integration Module

- REST endpoints, webhook support for ingestion partners, and background indexing pipelines for enrichment and embedding updates.

Technologies Used

- Backend: Node.js, Express, Redis, BullMQ
- Frontend: React Native (Expo), AsyncStorage
- AI/LLM: Gemini (LLM), embeddings via vector store (Redis Vector / Milvus / Pinecone)
- Database & Storage: PostgreSQL (Supabase) + JSONB, Supabase/S3 for object storage
- Caching & Queueing: Redis, BullMQ

- DevOps: Docker, Kubernetes (optional), GitHub Actions
- Observability: Prometheus/Grafana, Sentry/Jaeger

Technologies Used:

Backend	**Node.js, Express**
Frontend (mobile)	React Native (Expo), AsyncStorage
AI / LLM	Google Gemini (via API), prompt templates, JSON-schema validation
Vector store / Semantic Search	Redis Vector / Milvus / Pinecone (one of these as vector index)
Database & Storage	Supabase (Postgres + JSONB), Supabase Storage for resumes
Caching	Redis (backend cache), AsyncStorage (frontend cache)
Authentication & Authorization	Supabase Auth (JWT), role-based access controls
File handling & Validation	PDF/DOCX validation, server-side file upload handling (object store)
API & Integration	RESTful APIs (jobs, profiles, bookmarks, AI), webhooks for partners
Dev utilities	dotenv (.env), migration scripts, JSON Schema for AI outputs

System Architecture

- Presentation Layer: Mobile client (React Native) consumes REST APIs and caches results locally.
- Application Layer: Modular microservices (profiles, jobs, bookmarks, AI workers) expose clear REST endpoints and run background tasks for long AI jobs.
- Data Layer: Relational store for authoritative records, JSONB for flexible AI outputs, and a vector store for semantic search.
- This separation enables independent scaling, isolated failure domains, and targeted optimization of AI workloads.

Security Features

- TLS everywhere; server-side secret management for GEMINI_API_KEY.
- Authenticated endpoints, RBAC for admin actions, file-type/size validation on uploads.
- Rate limiting and quotas on AI endpoints; audit logs for AI requests and resume access.
- JSON Schema validation for AI responses to prevent malformed or misleading outputs.

Scalability and Deployment

- Containerized services (Docker) deployable to Kubernetes for horizontal scaling.
- Background workers scale independently to handle CPU/latency-intensive AI tasks.

- Multi-level caching (frontend AsyncStorage, Redis backend) reduces latency and LLM call volume.
- CI/CD pipelines for safe rollouts and automated migrations for DB schema changes.

This overview captures the end-to-end design: secure resume ingestion → AI parsing → semantic job enrichment → bookmark-driven analysis → actionable recommendations delivered via a resilient mobile UX.

5.1. Working

The AI-Driven Career Intelligence Platform operates through a layered workflow that automates end-to-end job discovery, resume ingestion and parsing, and personalized skill-gap recommendations. An AI search layer normalizes and enriches job listings, secure resume uploads are parsed by LLM-based workers to produce structured profiles, and a vectorized matching + recommendation engine compares bookmarked roles against the user's resume to generate prioritized learning plans. Background workers handle long-running AI tasks, multi-level caching preserves fast and persistent UX, and authenticated APIs provide traceability and auditability—ensuring seamless, transparent, and scalable career guidance.

Data Ingestion & Acquisition

- Job data enters via partner feeds, APIs, and manual postings; an AI search/enrichment layer normalizes incoming descriptions and extracts semantic metadata (skills, seniority, location, salary band).
- Users upload resumes (PDF/DOCX) through a secured endpoint; files are stored in object storage and queued for parsing by background workers.

Data Standardization & Transformation

- Resume and job text are transformed by NLP pipelines into canonical tokens: normalized skills, experience levels, project items, and structured JSONB records.
- Deduplication, case normalization, and schema validation are applied; normalized records and embeddings are written to the enriched/indexed storage tier for fast retrieval.

AI/ML-Driven Processing & Analysis

- LLM-based services (Gemini) perform resume parsing, skill extraction, requirement parsing, and generation of JSON-constrained recommendations using prompt templates and schema validation.
- Embeddings are generated for jobs and profile summaries and stored in a vector index; the skill-gap engine compares resume_extracted_skills against bookmarked job requirements

to compute match percentages, missing skills, and prioritized learning plans.

- Long-running AI tasks run asynchronously via a job queue with status endpoints and retry/backoff logic to handle transient failures.

Metadata & Governance Management

- Structured AI outputs, provenance metadata, and audit logs are persisted in Postgres (JSONB) and linked to user records; bookmarks are stored with UNIQUE(user_id, job_id) to prevent duplicates.
- Access control uses JWT-based authentication; sensitive keys (GEMINI_API_KEY) are server-side only, and user data deletion/export is supported for privacy compliance.

Visualization & Analytics

- Mobile UI (React Native) renders dashboards for suggested jobs, bookmarked jobs, per-job match scores, and learning recommendations; UI indicates bookmark state persistently across pages.
- Visual elements include match percentage bars, prioritized skill lists, and resource/project suggestions; progress and analysis statuses are shown for async AI jobs.

Data Access & Interoperability

- REST APIs expose profile, jobs, bookmarks, and AI analysis endpoints; AI endpoints accept jobId arrays for multi-job aggregation and return JSON-validated recommendations.
- Webhooks or partner API integrations enable external ingestion and result delivery; vector search endpoints support semantic queries from the client.

Monitoring & Logging

- System metrics, job-queue status, and AI call usage are tracked (Prometheus/Grafana or hosted equivalents); structured logs record resume uploads, AI requests, and bookmark operations for audit and debugging.
- Rate limiting and quota monitoring protect LLM usage; failures degrade gracefully to cached results.

Deployment & Scalability

- Services are modular (API, AI worker, indexing, background queue) and containerized for horizontal scaling; background workers scale independently for CPU-intensive AI tasks.
- Multi-level caching (frontend AsyncStorage, Redis backend) reduces latency and LLM calls; CI/CD pipelines and migration scripts manage safe rollouts.

User Interaction & Output

- Users sign in, upload a resume, browse suggested jobs, and bookmark roles; bookmarking

persists immediately and highlights until manually toggled off.

- From Bookmarks, users can trigger to run aggregated skill-gap analysis; results present match scores, missing/prioritized skills, estimated learning effort, curated resources, and suggested practice projects.

- Auth / Sign in & Sign up
  - Purpose: Authenticate users and obtain JWT for protected API calls.
  - Key components: Supabase Auth flow, token storage, auth gate for app.
  - APIs: Supabase Auth endpoints; local token refresh.
  - UX notes: Simple email/password with persistence; redirect to Dashboard after sign-in.
- Dashboard (Home)
  - Purpose: Landing view with personalized job suggestions, recent activity, and quick actions.
  - Key components: Suggested jobs feed, recent searches, bookmark count badge.
  - UX notes: Loads cached data from AsyncStorage instantly, refreshes in background.
- Job Search
  - Purpose: Discover roles via query, filters, and sorting with semantic relevance.
  - Key components: Search bar (debounced), filters (location, experience, specialization), paginated list.
  - UX notes: Debounced queries, infinite scroll, bookmark toggle on cards.

- Job Details
  - Purpose: Show full job description, requirements, and actions (bookmark, apply).
  - Key components: Description, extracted skills, company info, salary, apply/bookmark buttons.
  - UX notes: Ensure job persisted before bookmarking; optimistic UI for bookmark toggle.
- Bookmarks
  - Purpose: Central place for saved jobs and AI-driven fit analysis for those jobs.
  - Key components: Bookmarked list, total count, “Analyze My Fit” button, AI Recommendations panel.
  - UX notes: Async analysis with progress UI; results show per-job match %, missing skills, prioritized suggestions.
- Profile
  - Purpose: User profile management and resume upload / AI analysis center.

- Key components: Profile fields (name, location, skills), resume upload (PDF/DOCX), parsed resume summary.
- UX notes: Validate file type/size client-side, show parsed skills and allow re-run of analysis.

- Specializations / Explore
  - Purpose: Browse role categories or domains (Data Science, DevOps, Design) to discover focused feeds.
  - Key components: Category cards, curated lists, filter by specialization.
  - UX notes: Useful for targeted job discovery and bookmarking.

5.2: Block Diagram and Explanation

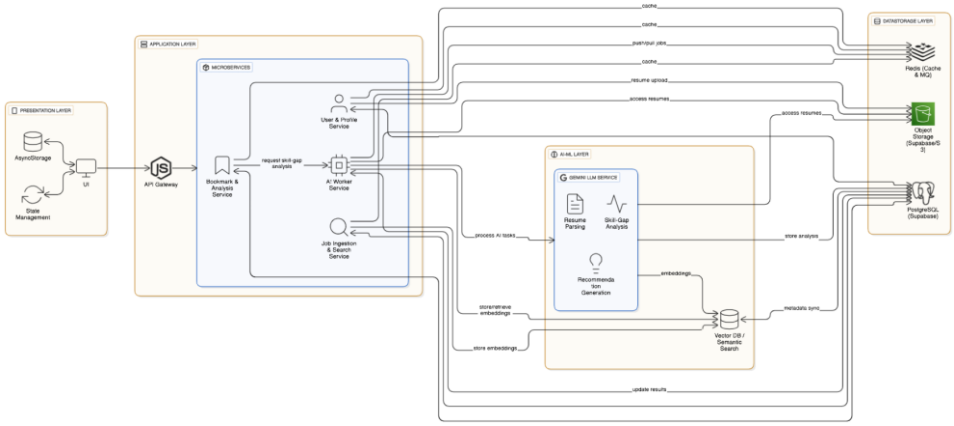

Figure 5.2.1: Block Diagram.

5.2.2 ONBOARDING PAGE

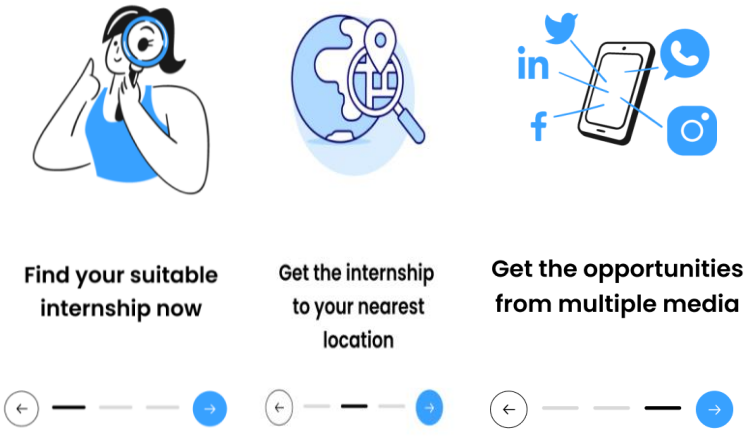

Figure 5.2.2: Onboarding Page

5.2.3 LOGIN PAGE

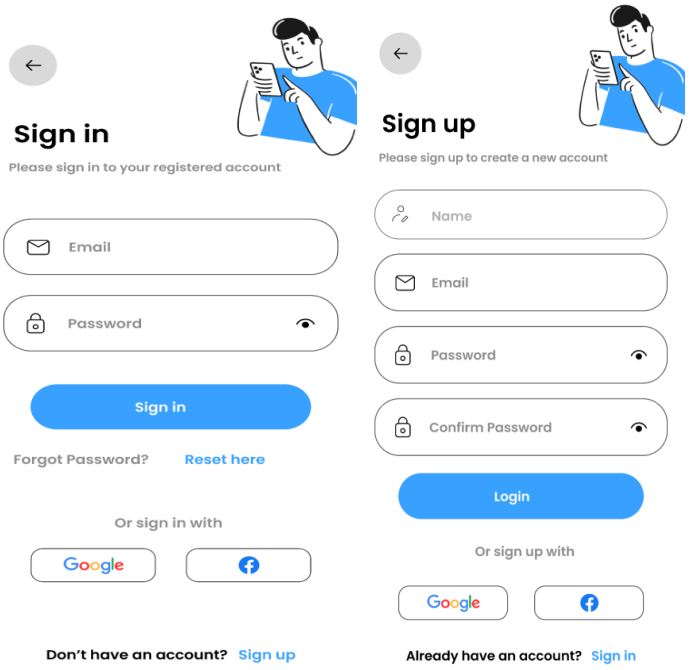

Figure 5.2.3: Login Page.

5.2.4 DETAILS FILLING PAGE

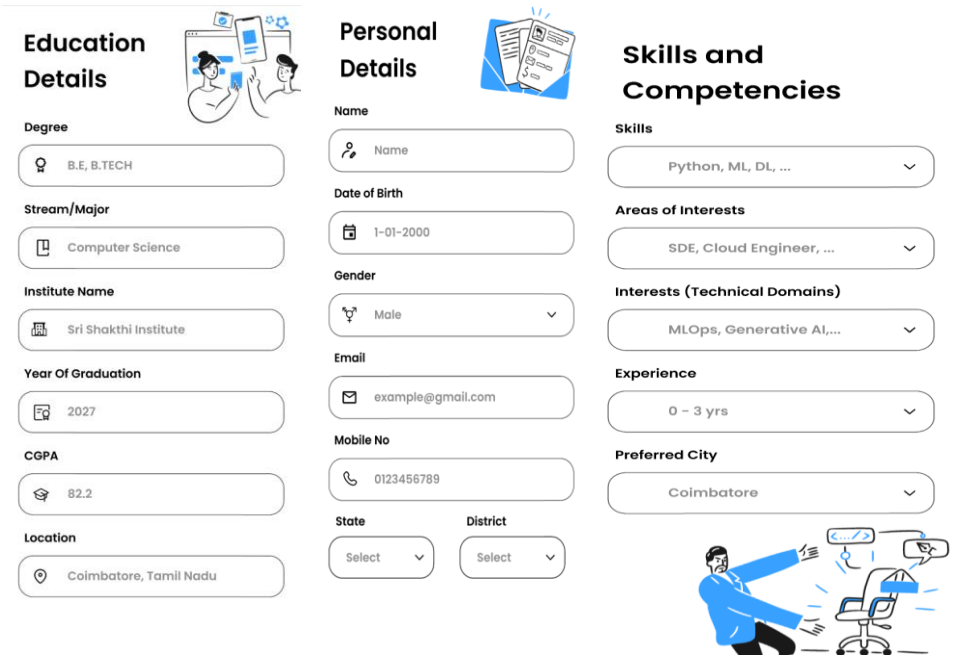

Figure 5.2.4: Details Filling Page

5.2.5 DASHBOARD PAGE

Figure 5.2.5: Dashboard Page.

5.2.6 JOB LISTING PAGE

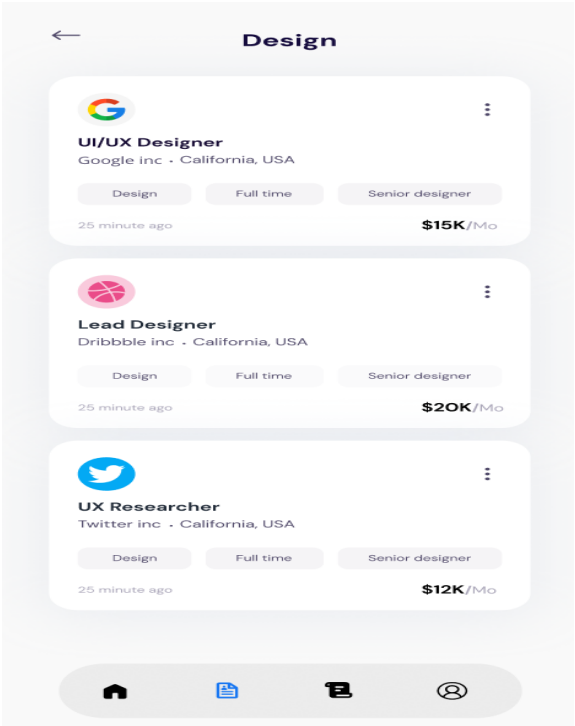

Figure 5.2.6: Job listing page

5.2.7 JOB DETAILS PAGE

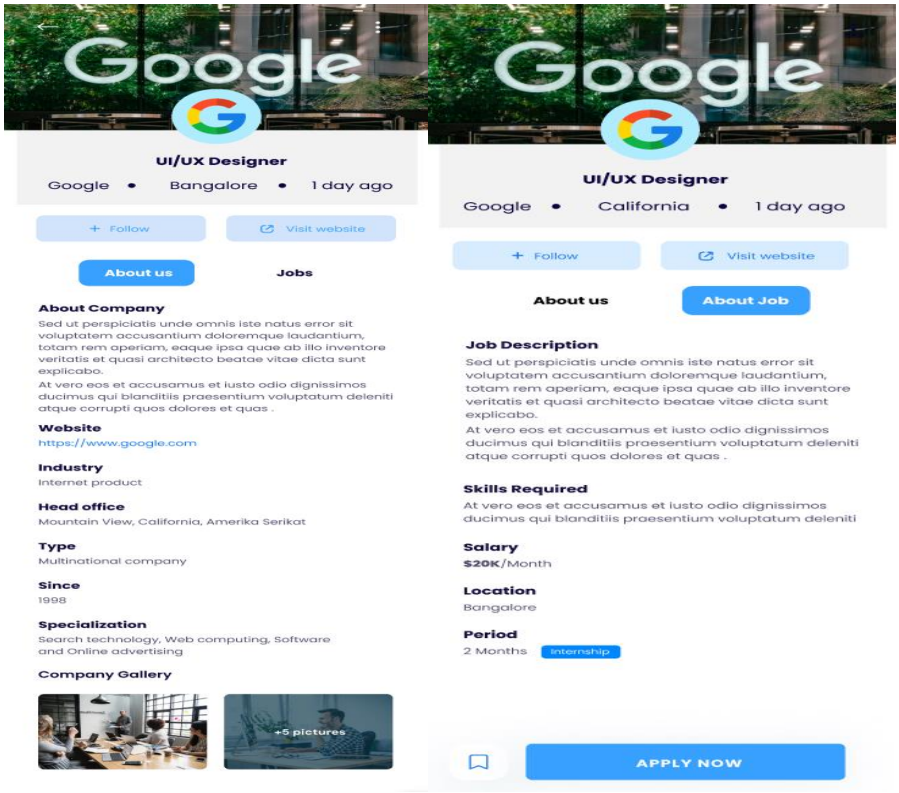

Figure 5.2.7: Job Details Page

5.2.8 BOOKMARKS PAGE

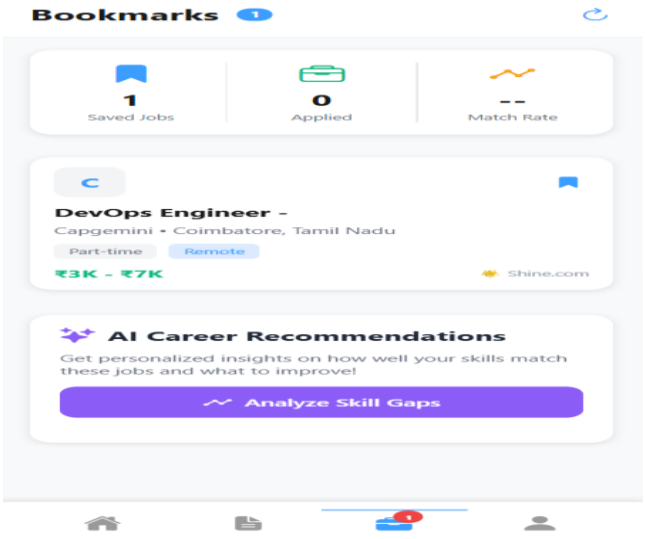

Figure 5.2.8: Bookmark Page.

5.2.9 PROFILE PAGE

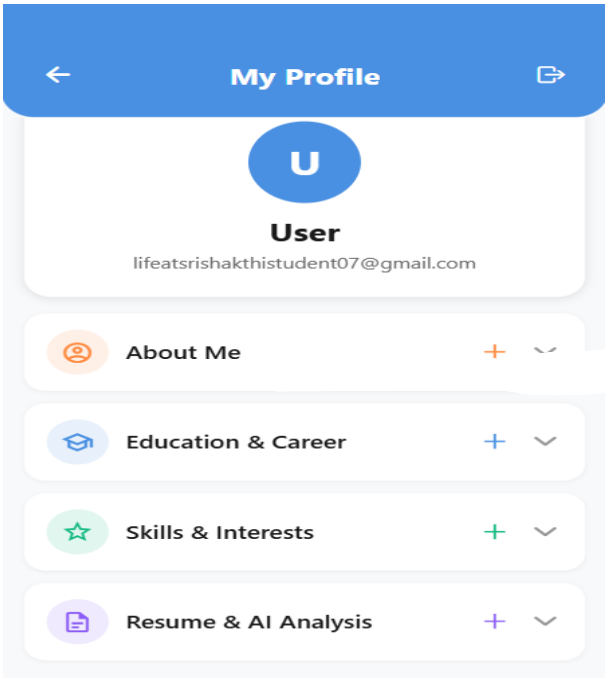

Figure 5.2.9: Profile Page.

5.2.10 CONFIRMATION AND PASSWORD MANAGEMNT PAGE

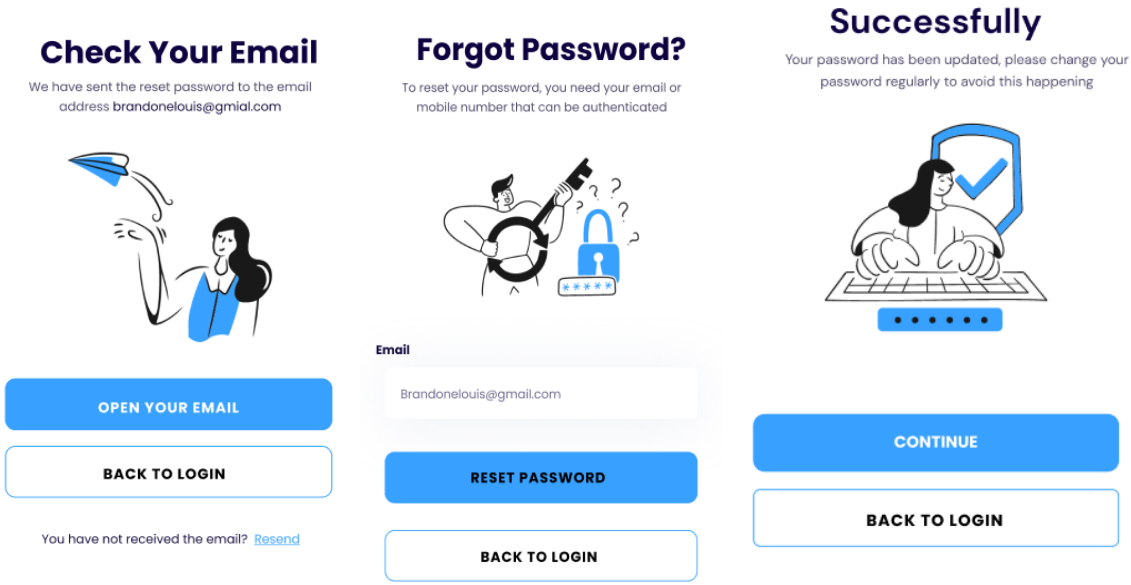

Figure 5.2.10: Confirmation and Password management Page

CHAPTER 6 VALIDATION AND TESTING

6.1 VALIDATION:

Validation and testing are integral to ensure the Career Intelligence & Job Recommendation Platform operates accurately, securely, and reliably. The validation process targets data

integrity, AI output correctness, API security, and end-user experience under realistic conditions.

Validation Techniques Used

- Data Ingestion Validation
  - All incoming job feeds and partner API payloads pass schema validation (required fields: title, description, location, postedAt, skills) before enrichment.
  - Resume uploads (PDF/DOCX) are validated for MIME type and size limits (e.g., ≤10MB); corrupted files are rejected and quarantined.
  - Checksums and server-side file validation ensure upload integrity.
- Metadata & Standardization Validation
  - Normalized job and profile metadata (canonical skill tokens, seniority tags) are validated against controlled vocabularies and JSON Schema.
  - Deduplication routines and casing normalization run client- and server-side to prevent duplicate skills or job entries.
- AI/ML Model & LLM Output Validation
  - LLM prompts enforce JSON-constrained outputs; responses are validated against JSON Schema before persistence.
  - Embedding generation and vector indexing are sanity-checked (dimension, non-NaN).
  - Automated checks detect improbable recommendations (e.g., recommending beginner resources for senior roles) and flag them for review.
- API & Access Control Validation
  - Automated API tests (Postman / supertest) verify request/response shapes, status codes, input validation, and error handling.
  - RBAC and JWT validation tests ensure endpoints restrict actions appropriately (profile edit, admin analytics).
  - Penetration checks for common API attacks (injection, malformed payloads).
- Data Pipeline & Workflow Validation
  - Background job workflows (resume parse, async AI analysis) validated end-to-end; retry and dead-letter handling tested.
  - Cache invalidation and focus-aware reload flows tested to ensure UI consistency after navigation.
- File & Object Storage Validation
  - Object storage entries validated for path integrity, access permissions, and checksum

correctness.

- Resume retrieval and reprocessing tested for idempotency.

- Session, Authentication & Encryption Validation
- Token lifecycle tests (issue, expiration, refresh) and replay-attack checks.
- All communications validated to run over TLS in staging/production.
- Secrets (GEMINI_API_KEY) confirmed server-side only.

- UI & UX Validation
- Component tests ensure bookmark toggle, optimistic UI, and highlight state behave consistently across job cards and job details.
- Resume upload, parsed summary display, and "Analyze My Fit" flows validate correct states (loading, error, completed).

- Performance & Reliability Validation
- Load tests validate API latency under concurrent users and check LLM call backpressure behavior.
- Cost-control checks validate batching/caching reduce duplicate LLM invocations.

6.2 TESTING:

A layered testing strategy verifies units, integrations, systems, models, and user acceptability.

- Unit Testing
- Backend: Jest / Mocha tests for controllers, validators, DB access (mocked).
- Frontend: Jest/React Testing Library for components (bookmark button, job card, profile forms).
- AI helpers: prompt templates, JSON Schema validators, and skill-normalization utilities covered by unit tests.

- Integration Testing
- Containerized integration tests exercise API → DB → object store flows (resume upload → analyze → store resume_analysis).
- Background worker integration: enqueue job, worker processes LLM call (mocked LLM), result persisted and status updated.

- System Testing
- End-to-end scenarios emulate user journeys (signup → upload resume → bookmark jobs → analyze bookmarks → view recommendations).
- Monitoring of latency, queue lengths, error rates via observability stack during tests.

- Model & LLM Validation

- Offline evaluation of extraction templates using labeled resumes; metrics: extraction precision/recall, entity F1.
- Skill-gap heuristics validated on held-out job/resume pairs; spot checks for hallucinations and plausibility.
- Drift detection: monitor recommendation quality and trigger retrain or prompt adjustments.

- Manual & Exploratory Testing
  - QA verifies edge cases: very long resumes, multilingual resumes, jobs with minimal descriptions, intermittent network.
  - UX reviewers validate clarity of recommendations, resource links, and privacy controls.
- User Acceptance Testing (UAT)
  - Beta testers validate usefulness of recommendations, match accuracy, and overall flow; feedback cycles used to refine prompts, UI copy, and error handling.
- Regression & CI Testing
  - CI pipeline runs unit + integration tests and lints on every PR; mocks used for LLM-dependent tests to keep CI stable.
  - Nightly regression suite covers critical user flows and AI response schema validation.

7.1 MERITS:

CHAPTER 7 MERITS AND DEMERIT

Personalized matching: resume-aware comparisons produce job recommendations tailored to a user's actual skills and experience rather than simple keyword matches.

- Actionable guidance: bookmark-driven multi-job analysis yields prioritized, practical learning steps (skills, resources, project ideas).
- Enhanced discoverability: semantic/embedding-based job indexing captures synonyms and context, improving relevance over lexical search.
- Data quality & consistency: client+server deduplication and normalized skill tokens reduce noise and duplicate records.
- Persistent, responsive UX: frontend caching and focus-aware reloads preserve state across navigation and support offline access.
- Reliable bookmark workflows: referential integrity (store job before bookmark) and UNIQUE constraints prevent duplicate bookmarks and count errors.
- Asynchronous AI handling: background workers and status endpoints keep UI

responsive while long LLM tasks run safely.

- Cost and latency control: batching, caching, and async processing minimize unnecessary LLM calls and operational cost.
- Security & privacy controls: server-side API keys, authenticated endpoints, file-type/size validation, and audit logging protect sensitive resume data.
- Extensible architecture: modular services (indexing, AI, profile, bookmarks) enable independent scaling and future feature additions.
- Observability & maintainability: structured logs, metrics, and migration scripts support debugging and safe deployments.
- Measurable impact: explicit metrics (match accuracy, analysis latency, bookmark usage) enable iterative improvement and validation.

7.2 DEMERITS

- Heavy dependency on LLM (Gemini): cost, rate limits, and availability may affect features (resume analysis, skill-gap).
- LLM hallucination risk: AI may produce incorrect or misleading recommendations; requires prompt validation and human review (backend/src/controllers/aiController.js).
- Latency for AI tasks: long running analyses can degrade UX unless reliably handled async with status updates (bookmarks analysis in frontend/app/bookmarks.tsx).
- Privacy and compliance concerns: storing parsed resume text (PII) increases regulatory and security burden; needs strict access controls and retention policies (resume fields/migration backend/migrations/add_resume_columns.sql).
- Cost and scaling of vector/embedding store and LLM calls: production scale may require additional budget and engineering (batching/caching essential).
- Partial test coverage for AI flows: LLM-dependent logic needs robust unit/integration tests and mocked AI responses for CI.
- Operational complexity: background workers, queues, and retries add maintenance overhead and failure modes (resume/job ingest + async AI).
- Data quality dependence on input text: poor or multilingual resumes and sparse job descriptions yield low-quality recommendations.
- Potential for duplicate or transient job IDs: must persist jobs before bookmarking to avoid FK errors (route/order and storage assumptions).
- Mobile performance limits: large lists, images, and frequent AI calls can increase bandwidth and battery usage; caching and pagination required.

- Security surface: server must protect GEMINI_API_KEY and ensure file validation (PDF/DOCX) and anti-abuse rate limits are enforced.
- Migration & backward compatibility: DB schema changes (resume_analysis column) require careful rollout and migration scripts to avoid downtime.

CHAPTER 8 FUTURE SCOPE

- **Recruiter & Employer Portal**

Introduce a dedicated dashboard for employers to post jobs, manage candidates, and discover anonymized talent profiles. Backend routes and admin UI can be implemented to support recruiter-specific workflows and analytics.

- **One-Click Apply & ATS Integration**

Enable seamless job applications directly from the app, automatically forwarding user data to external Applicant Tracking Systems (ATS) or employer webhooks through secured API endpoints.

- **Application Tracking System (ATS)**

Implement in-app application tracking to display job application history, current status, and updates through notifications. Extend existing job and bookmark models to support tracking and analytics.

- **Multi-Resume & Profile Versioning**

Allow users to maintain multiple resume versions and select a preferred one for each job application. Store these variations as structured JSONB arrays in the profile schema.

- **Resume Redaction & Privacy Controls**

Integrate automated redaction tools to remove personally identifiable information (PII) such as phone numbers or emails before sharing resumes with recruiters. Offer user-controlled export options for anonymized sharing.

- **Anonymized Candidate Search for Recruiters**

Provide privacy-preserving candidate discovery using vector embeddings for skill-based similarity without exposing PII. Implement this via redacted vector search endpoints and recruiter-facing interfaces.

- **Human-in-the-Loop Review Mechanism**

Introduce an admin moderation layer where AI-generated recommendations can be reviewed or approved before being shown to users, ensuring higher accuracy and ethical AI governance.

- **Explainable AI Recommendations**

Enhance transparency by including evidence snippets or extracted context behind each AI suggestion (e.g., skill extraction source lines), stored in the resume_analysis JSONB field and displayed in the UI.

- **Scheduled & Batch Bookmark Analysis**

Automate re-evaluation of bookmarked jobs at scheduled intervals using background workers (Redis + BullMQ). This ensures updated skill-gap analyses as new jobs or resume updates occur.

- **Webhooks & Partner Integrations**

Support external integrations through webhook endpoints for job feed ingestion and result delivery to partner systems, expanding the platform's ecosystem connectivity.

- **Multi-Language Resume Parsing**

Extend resume parsing capabilities to support multiple languages using enhanced Gemini LLM prompts and localized rule-based extraction fallbacks.

- **Portfolio & Document Integration**

Allow users to link GitHub repositories, personal websites, or project portfolios for richer skill profiling. Extract repository metadata such as languages, technologies, and contributions.

- **Export, Import & Reporting Features**

Provide options to export or import user data, analysis results, and bookmarks in CSV/JSON formats for compliance (GDPR) and personal recordkeeping.

- **Notifications & Scheduling**

Add push notifications and email summaries for important events such as AI analysis completion, new matching jobs, or recruiter responses, integrated with Expo push and background schedulers.

- **Operational & Cost Optimization Dashboards**

Develop monitoring dashboards to visualize LLM usage, API costs, and model fallback rates. Implement adaptive batching and caching mechanisms to balance performance with cost efficiency.

CHAPTER 9 CONCLUSION

The project delivers an AI-driven Job recommendation app that unifies resume parsing, semantic job indexing, bookmark-driven skill-gap analysis, and prioritized learning

recommendations into a single, scalable system. Implemented features include secure resume upload and JSONB resume analysis, LLM-based parsing and recommendations (Gemini), vectorized semantic matching, persistent bookmark workflows with deduplication, async AI processing, and an offline-friendly React Native client. The architecture emphasizes modularity, security, observability, and cost controls (caching, batching), making it ready for staged deployment and iterative improvement. With validated core flows and clear migration/admin procedures, the platform is positioned to improve job relevance and user employability while accommodating future enhancements such as course integrations, explainability, and enterprise-grade privacy controls.

REFERENCE

1. Google Makersuite — Generative AI & Gemini API. Google Cloud / Makersuite. https://makersuite.google.com/app/apikey
2. Supabase Documentation — Auth, Storage, and Postgres (JSONB). Supabase. https://supabase.com/docs
3. React Native (Expo) Documentation. Meta / Expo. https://reactnative.dev/ & https://docs.expo.dev/
4. Node.js & Express.js Documentation. Node.js Foundation / Express. https://nodejs.org/ & https://expressjs.com/
5. PostgreSQL Documentation (JSONB). The PostgreSQL Global Development Group. https://www.postgresql.org/docs/
6. Redis & BullMQ (background jobs). Redis Labs / OptimalBits BullMQ. https://redis.io/ & https://docs.bullmq.io/
7. Vector Search & Embeddings — Milvus / Pinecone / Redis Vector (product docs). https://milvus.io/ https://www.pinecone.io/ https://redis.io/docs/stack/search/
8. JSON Schema — Validation best practices. IETF & json-schema.org. https://json-schema.org/
9. OWASP Top Ten — Web application security guidance. OWASP Foundation. https://owasp.org/www-project-top-ten/
10. JWT (RFC 7519) — JSON Web Token specification. IETF. https://datatracker.ietf.org/doc/html/rfc7519
11. MLflow — Model lifecycle & tracking. Databricks / MLflow. https://mlflow.org/
12. Prompt Engineering & LLM Safety Guidelines — Best practices (technical blogs / papers). OpenAI & community

resources. https://platform.openai.com/docs/guides/prompting

13. React Navigation & useFocusEffect Hook (navigation lifecycle). React Navigation Docs. https://reactnavigation.org/
14. AsyncStorage (React Native) — Offline caching patterns. React Native Community. https://react-native-async-storage.github.io/async-storage/
15. Software Architecture Patterns — Microservices, Background Workers, Caching (general references). Martin Fowler & cloud provider docs. https://martinfowler.com/articles/microservices.html
16. Secure File Uploads — MIME/type and size validation practices. OWASP File Upload Cheat
Sheet. https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html
17. User Experience & Mobile Performance — Pagination, optimistic UI, and caching patterns. Platform design guides (Google, Apple) & React Native performance docs. https://developer.android.com/ & https://developer.apple.com/