LearnovateX – Full Project Documentation

Repository: shaik-asif0/finalyearProject
Project Type: Final Year Project (Full-Stack + AI)
Date: February 3, 2026

---

1) Abstract

LearnovateX is an AI-powered learning and career readiness platform built as a full-stack web application. It supports multiple user roles (Student, Job Seeker, Company, College Admin) and provides modules for AI tutoring, coding practice with evaluation, resume analysis, mock interviews, career readiness dashboards, analytics, and administrative workflows.

The system is designed to run locally with minimal setup using **FastAPI + SQLite** on the backend and **React** on the frontend. AI features support **demo mode** for offline/demo usage and **Azure OpenAI mode** for real responses when credentials are configured.

---

2) Problem Statement

Students and job seekers often struggle with:

- Limited access to personalized guidance
- Slow feedback loops for coding practice
- Unclear resume quality and ATS compatibility
- Low-confidence interview preparation
- Lack of data-driven progress tracking

Institutions and recruiters also need:

- Batch/student progress visibility
- Faster candidate screening and ranking
- Simple tools for assessments and tracking

---

3) Objectives

- Provide a 24/7 AI tutor experience (demo/real AI)
- Evaluate code submissions with structured feedback (correctness, complexity, score)
- Analyze resumes and provide credibility scoring and improvement suggestions
- Provide mock interview practice and scoring
- Build dashboards for progress + career readiness
- Support role-based modules for students, companies, and colleges

---

4) Tech Stack

Frontend

- React 18 (Create React App + CRACO)
- React Router v7
- Tailwind CSS + shadcn/ui (Radix primitives)
- Axios client with offline GET caching
- Monaco editor for coding arena

- Recharts for charts

Backend

- Python 3.11+
- FastAPI (REST API)
- SQLite (local DB file)
- JWT auth + bcrypt password hashing
- File uploads + static serving under `/uploads`
- PDF parsing via PyPDF2
- ATS resume PDF generation via ReportLab
- Azure OpenAI via the OpenAI Python SDK (optional)

---

5) Project Structure

```
.
├── backend/
│   ├── server.py
│   ├── schema.sql
│   ├── requirements.txt
│   ├── .env / .env.example
│   ├── uploads/
│   └── web.config
├── frontend/
│   ├── src/
│   ├── package.json
│   └── staticwebapp.config.json
├── README.md
├── PRESENTATION.md
└── PROJECT_DOCUMENTATION.md
```

---

6) High-Level Architecture

Logical Components

- **React Frontend**: UI, routing, offline caching, feature pages
- **FastAPI Backend**: auth, business logic, AI orchestration, storage
- **SQLite**: persistence for users, learning history, evaluations, tracking
- **Azure OpenAI (optional)**: real AI outputs when enabled

Data Flow

1. Frontend calls backend REST endpoints under `/api/*`
2. Backend validates JWT (protected routes)
3. Backend reads/writes SQLite tables
4. AI modules build prompts and call AI wrapper (demo/azure)
5. Backend returns structured JSON to frontend

---

7) User Roles & Access

- **Student**: tutor, coding, resume, interview, dashboard, achievements, leaderboard, learning paths

- **Job Seeker**: similar to student + company-facing profile scoring
- **Company**: candidate listing + stats, assessments/tests, job postings, candidate actions
- **College Admin**: student tracking, analytics, announcements, messaging

Role enforcement is implemented in the backend (e.g., company-only endpoints) and in the frontend routing guards.

---

8) Backend Setup & Configuration

8.1 Prerequisites

- Python 3.11+
- Node.js 18+

8.2 Backend Environment Variables

Backend loads `backend/.env`.

Minimal (demo mode):

```
AI_MODE=demo
JWT_SECRET=change-me
```

Azure OpenAI (real AI):

```
AI_MODE=azure
AZURE_OPENAI_API_KEY=your_key
AZURE_OPENAI_ENDPOINT=https://your-resource.openai.azure.com/
AZURE_OPENAI_DEPLOYMENT=gpt-4
```

Other useful settings (optional):

- `CORS_ORIGINS` / `CORS_ORIGIN_REGEX`
- `MAX_FILE_SIZE_MB`
- `UPLOAD_DIR`, `RESUME_UPLOAD_DIR`, `CODE_UPLOAD_DIR`
- SMTP vars for password reset OTP email

8.3 Running the Backend (Local)

From the repo root:

```
cd backend
pip install -r requirements.txt
uvicorn server:app --reload --host 0.0.0.0 --port 8000
```

API docs (Swagger):

- http://localhost:8000/docs

---

9) Frontend Setup & Configuration

9.1 Frontend Environment Variables

Frontend uses `REACT_APP_API_BASE_URL` (base URL without `/api`).

```
REACT_APP_API_BASE_URL=http://localhost:8000
```

9.2 Running the Frontend (Local)

```
cd frontend
npm install
npm start
```

Frontend:

- http://localhost:3000

---

10) Core Backend Modules (What the API Does)

This section documents the key backend modules implemented in `backend/server.py`.

10.1 Health / Status

- `GET /api/` – basic API response
- `GET /api/health` – health checks
- `GET /api/status` – status info

10.2 Authentication

- `POST /api/auth/register` – register user (role-based)
- `POST /api/auth/login` – login and receive JWT
- `GET /api/auth/me` – current user profile

Password reset flow:

- `POST /api/auth/forgot-password`
- `POST /api/auth/verify-otp`
- `POST /api/auth/reset-password`

10.3 Profile

- `GET /api/profile`
- `PUT /api/profile`
- `POST /api/profile/avatar` – avatar upload
- `GET /api/profile/resume/ats` – generate/download ATS resume PDF

10.4 AI Tutor

- `POST /api/tutor/chat`

Stores conversations into `learning_history`.

10.5 Coding Arena Evaluation

- `POST /api/code/evaluate`
- `GET /api/code/submissions`

Stores evaluations into `code_evaluations`.

10.6 Resume Analyzer

- `POST /api/resume/analyze` (PDF upload)
- `GET /api/resume/history`
- `GET /api/resume/latest`

Uploaded resumes are saved under `backend/uploads/…` and served from the backend under `/uploads/...`.

10.7 Mock Interview

- `POST /api/interview/start`
- `POST /api/interview/evaluate`
- `GET /api/interview/history`

Stores evaluations into `interview_evaluations`.

10.8 Dashboards

- `GET /api/dashboard/stats` – quick statistics
- `GET /api/career/readiness` – aggregated career readiness dashboard payload

10.9 Activity Tracking

- `POST /api/activity/event` – records lightweight client analytics (page views, time spent)
- `GET /api/activity/heatmap` – activity heatmap style data

10.10 Apply Tracker

- `GET /api/career/apply-tracker`
- `POST /api/career/apply-tracker`
- `PATCH /api/career/apply-tracker/{item_id}`
- `DELETE /api/career/apply-tracker/{item_id}`

10.11 Achievements & Leaderboard

- `GET /api/achievements`
- `GET /api/leaderboard`

10.12 Company Portal

Companies can:

- Create/list tests or assessments
- Manage job postings
- View candidates and perform actions

Representative endpoints:

- `POST /api/company/tests`
- `GET /api/company/tests`
- `GET /api/company/candidates`
- `GET /api/company/analytics`
- `POST /api/company/jobs`
- `GET /api/company/jobs`
- `PUT /api/company/jobs/{job_id}`
- `DELETE /api/company/jobs/{job_id}`
- `POST /api/company/candidates/{candidate_id}/action`

- `GET /api/company/candidates/{candidate_id}/actions`

10.13 College Admin

- `GET /api/college/students`
- `GET /api/college/analytics`
- `POST /api/college/announcements`
- `GET /api/college/announcements`
- `DELETE /api/college/announcements/{announcement_id}`
- `POST /api/college/students/{student_id}/message`
- `GET /api/college/students/{student_id}/details`

10.14 Premium Courses / Internships

- `POST /api/premium/enroll-course` (multipart form + optional screenshot)
- `POST /api/premium/apply-internship`
- `GET /api/premium/my-enrollments`
- `GET /api/premium/my-applications`

Admin approval endpoints:

- `GET /api/admin/enrollments`
- `PUT /api/admin/enrollments/{enrollment_id}/status`
- `GET /api/admin/applications`
- `PUT /api/admin/applications/{application_id}/status`

---

11) Database Design (SQLite)

SQLite DB file is stored in `backend/learnovatex.db`.

The schema is defined in `backend/schema.sql`.

Key tables:

- `users` – accounts, roles, profile data, avatar
- `learning_history` – tutor chats
- `code_evaluations` – code submissions + scoring
- `resume_analyses` – resume analysis history
- `interview_evaluations` – interview sessions + scoring
- `career_readiness_snapshots` – day-wise readiness
- `activity_events` – page view/time spent events
- `apply_tracker` – job application tracking
- `tests`, `assessments`, `job_postings`, `candidate_actions` – company portal
- `announcements`, `messages`, `student_messages` – college admin messaging
- `course_enrollments`, `internship_applications` – premium flows
- `password_reset_otps` – OTP storage for password reset

---

12) Security & Limits

- JWT-based auth using Bearer token
- Password hashing using bcrypt
- CORS allowlist + regex for preview/static web app domains
- Upload size enforced using `MAX_FILE_SIZE_MB`

- Resume file type allowlist via `ALLOWED_RESUME_FORMATS`

---

13) Offline / Demo Behavior

- Frontend caches successful `GET` responses into localStorage for up to 24 hours and can serve cached data when offline.
- AI endpoints support demo mode (`AI_MODE=demo`) so the app can be demonstrated without cloud keys.

---

14) Deployment Notes (Azure)

Backend

- Can be deployed to Azure App Service.
- `backend/web.config` shows a typical IIS/httpPlatformHandler configuration for running Uvicorn.

Frontend

- Can be deployed to Azure Static Web Apps.
- `frontend/staticwebapp.config.json` includes a production `REACT_APP_API_BASE_URL` value for the deployed backend.

---

15) How to Generate This Documentation as PDF

This repo includes a Markdown version of the full documentation (`PROJECT_DOCUMENTATION.md`).

A PDF version can be generated automatically using Node tooling (Puppeteer-based renderers) such as `md-to-pdf` .

---

16) Appendix – Quick Local Run (Windows)

Backend:

```
cd backend
python -m venv venv
venv\Scripts\Activate.ps1
pip install -r requirements.txt
uvicorn server:app --reload --port 8000
```

Frontend:

```
cd frontend
npm install
$env:REACT_APP_API_BASE_URL="http://localhost:8000"
npm start
```