UE22CS341A: Software Engineering
Case Study

Team Members:	**Suhit Hegde**	**PES2UG22CS591**
	Tanistha Hota	**PES2UG22CS618**

A Project Plan document for an **Automated Skill Matching and Gap Analysis Tool using Agile.**

1. Project Lifecycle Model

- **Lifecycle Model: Agile Methodology**
- **Justification:**
    - **Flexibility:** Agile's iterative approach allows for continuous updates and adjustments, making it well-suited for projects that need to respond quickly to changes, like skill requirements and evolving job roles.
    - **Phased Development:** Key components of your tool, such as skill parsing, gap analysis, and learning recommendation, can be developed and refined in phases.
    - **Regular Testing:** Agile allows for consistent feedback loops, crucial for ensuring accurate skill matching and gap analysis as job descriptions and skills may vary.

2. Tools for Project Lifecycle

- **Planning Tool:** Use Notion to track tasks, manage progress, and facilitate sprint planning.
- **Design Tools:** Utilize Draw.io for database schema and workflow diagrams.
- **Version Control:** GitHub for repository management and team collaboration.
- **Development Tools:**
    - **Backend:** Use Node.js for API development and Express.js to handle requests.
    - **Database:** Use MySQL for relational database management of candidate profiles and job requirements.
- **Bug Tracking:** Implement tools like Jira or GitHub Issues to track bugs and manage the backlog.
- **Testing Tools:** Use Postman for API testing and PyTest for unit testing.

3. Deliverables and Categorization

- **Reuse Components:**
    - **Parsing Libraries:** Leverage pre-existing libraries for parsing resumes **and**

job descriptions.
    - **Learning Recommendation API:** Use established APIs for course recommendations based on skill gaps.
- **Build Components:**
    - **Skill Matching and Gap Analysis Logic:** Develop the custom algorithms for comparing skills and identifying gaps.
    - **Custom Dashboards and Visualizations:** Implement a dashboard that provides visual insights into skill matches and gaps.
    - **User Interface for Document Upload:** Create an intuitive front-end interface for users to upload resumes and job descriptions.

4. Work Breakdown Structure (WBS)

Phase 1: Project Setup

- Define requirements, set up the development environment, and identify key features.

Phase 2: Backend Development

- Implement APIs for handling resume and job description uploads, skill matching, and gap analysis.

Phase 3: Frontend Development

- Build the user interface for resume/job description uploads and visualize results.

Phase 4: Testing and Security

- Perform unit testing, integration testing, and ensure secure data handling.

Phase 5: Documentation and Finalization

- Document system architecture, user guides, and technical specifications.

5. Effort Estimation and Gantt Chart

1. Requirement Analysis (Week 1 - Week 2):

    Effort: 0.5 person-months

2. System Design (Week 2 - Week 4):

    Effort: 1 person-month

3. Database Development (Week 4 - Week 6):

    Effort: 1.5 person-months

4. UI Development (Week 5 - Week 6):

    Effort: 2 person-months

5. Integrating Machine Learning Algorithms (Week 6- Week 7)

Effort: 0.5 person-months

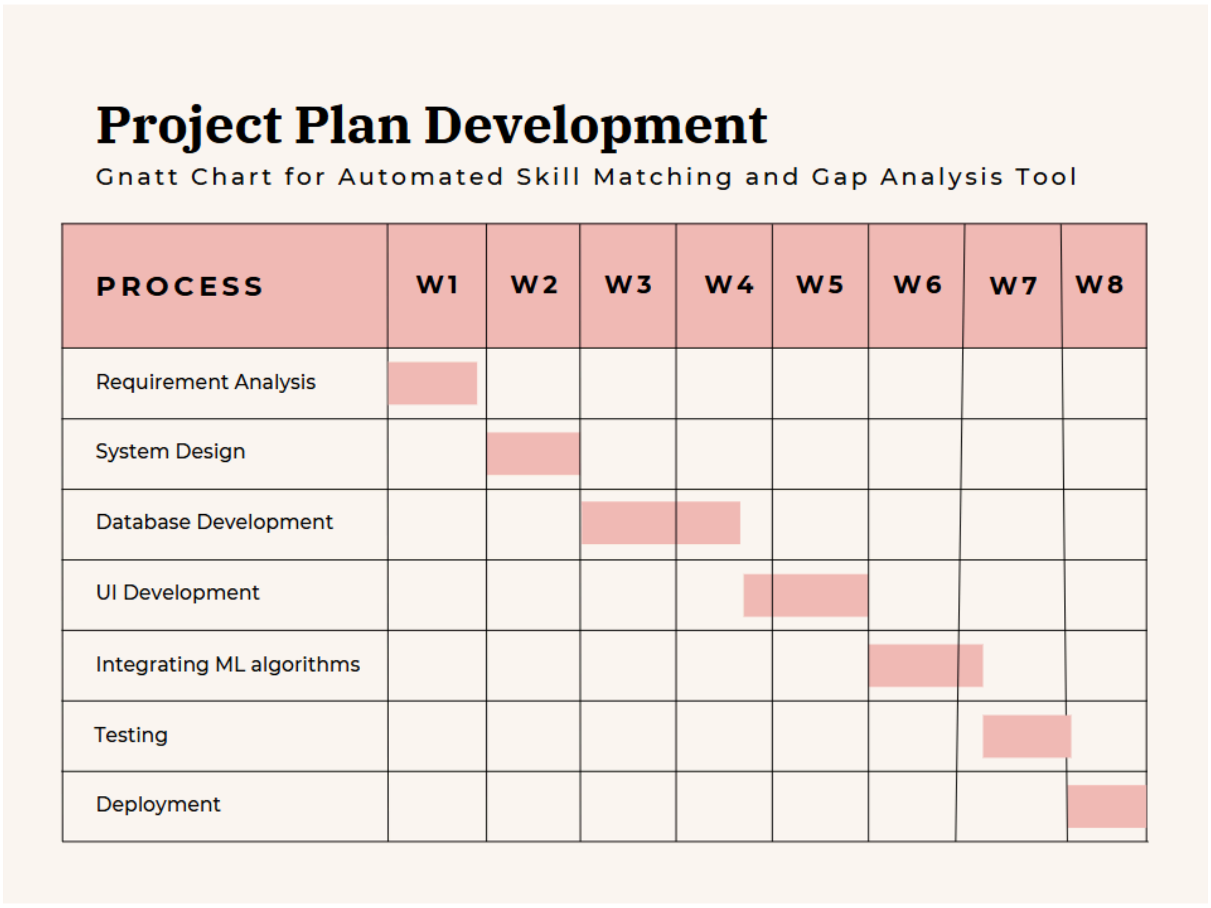
6. Testing (Week 7 - Week 8):

Effort: 0.5 person-months

7. Deployment (Week 8):

Effort: 0.25 person-months

Total Effort Estimate: 5.75 person-months



Project Plan Development

Gnatt Chart for Automated Skill Matching and Gap Analysis Tool

PROCESS	W1	W2	W3	W4	W5	W6	W7	W8
Requirement Analysis	▨							
System Design		▨						
Database Development			▨	▨				
UI Development				▨	▨			
Integrating ML algorithms						▨	▨	
Testing							▨	
Deployment								▨

6. Coding Details

Languages:

- Backend: Python for API and MySQL database operations.
- Frontend: JavaScript with frameworks like React.js for dynamic and responsive UIs
- Frameworks: Flask/Django for backend services and REST API.
- React.js for frontend development.
- Database: MySQL for storing entities.
- Integration: Use REST APIs for interaction between frontend and backend. Code

Structure:

- Machine Learning- Fuzzy algorithm for String Matching
- MVC (Model-View-Controller) architecture to separate concerns.
- Modular code with reusable components for data handling, visualization, and user management.