Low Level Design Document

Introduction

This Low Level Design (LLD) document outlines the core implementation details for **MentorMatch - AI Career Guidance Platform**. The platform leverages generative AI to match students with mentors, generate personalized mentorship plans, and facilitate secure, collaborative career development workflows.

---

1. System Components

Component	Key Responsibilities
User Service	User registration, authentication, profile management
Mentor Matching Engine	AI-driven mentor-mentee matching, plan generation
Mentorship Module	Plan scheduling, progress tracking, communication templates
Collaboration Tools	Messaging, file sharing, team-based features
Data Layer	MongoDB schemas, data access, persistence
API Gateway	RESTful API endpoints, request routing, authentication middleware
Frontend (React)	UI for all user roles, real-time updates, dashboard

---

2. Class/Interface Overview

Class/Interface	Relationships	Key Methods/Attributes
`User`	Base for Student/Mentor	`id`, `name`, `email`, `role`, `profile`
`Student` (extends User)	Mentorships, Progress	`interests`, `goals`, `skillGaps`
`Mentor` (extends User)	Mentorships	`expertise`, `availability`
`MentorshipPlan`	Linked to Student/Mentor	`planId`, `schedule`, `tasks`, `status`
`Message`	Linked to MentorshipPlan	`senderId`, `content`, `timestamp`
`MatchingEngine`	Uses AI Model	`match(student)`, `generatePlan(student, mentor)`

Key Methods Example:

```
class MatchingEngine {
  match(student) => mentor
```

```
  generatePlan(student, mentor) => MentorshipPlan
}
```

---

3. Data Structure Overview

Model	Fields
User	id, name, email, passwordHash, role, profile
StudentProfile	userId, interests, goals, skillGaps, progress
MentorProfile	userId, expertise, bio, availability
MentorshipPlan	planId, studentId, mentorId, schedule, tasks, status
Message	messageId, planId, senderId, content, timestamp
Progress	planId, completedTasks, feedback, lastUpdated

---

4. Algorithms/Logic

Mentor Matching Flow (Pseudocode):

```
function match_student_to_mentor(student):
    candidates = fetch_available_mentors()
    scores = []
    for mentor in candidates:
        score = ai_model.evaluate(student.profile, mentor.profile)
        scores.append((mentor, score))
    return mentor with highest score

function generate_mentorship_plan(student, mentor):
    plan = ai_model.create_plan(student.goals, student.skillGaps, mentor.experti
    save(plan)
    return plan
```

---

5. Error Handling

Scenario	Handling Approach
Invalid login/authentication	Return 401 Unauthorized, log attempt
Data validation failure	Return 400 Bad Request with error details
AI service unavailable	Fallback to rule-based matching, notify user
Mentor/Student not found	Return 404 Not Found
Database errors	Return 500 Internal Server Error, log incident
Unauthorized resource access	Return 403 Forbidden

End of Document