ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

Scope of Work: Intern-Employer Matchmaking Platform

1. Project Overview

The project involves developing an AI-powered internship matchmaking platform that connects undergraduate and graduate students with employers offering internship opportunities. The platform will incorporate advanced AI features for improved matching, including resume parsing, career prediction, skill gap analysis, and interview pre-screening. This document outlines the comprehensive scope, components, and implementation details for the entire project.

2. Target Audience

2.1 Students

- **Demographics**: Undergraduate and graduate students, age 18-30
- **Academic Fields**: Computer Science, Information Technology, Engineering, Business, Economics, Finance, Commerce, Design, and other disciplines
- **Technical Proficiency**: Varying technical skills, from beginner to advanced
- **Needs**: Finding relevant internships, showcasing skills effectively, understanding career pathways

2.2 Employers

- **Organization Types**: Financial Institutions, Startups, SMEs, large corporations, non-profit organizations, government agencies
- **Industries**: Technology, finance, healthcare, manufacturing, retail, education, etc.
- **Needs**: Finding qualified interns, efficient screening process, evaluating potential candidates

2.3 Educational Institutions

- **Types**: Universities, colleges, vocational training institutes
- **Needs**: Placement tracking, industry partnerships, curriculum alignment with market needs
- **Campus Placements: where college profiles, student profiles and placements controlled by colleges**

3. Functional Requirements

3.1 User Authentication and Onboarding

3.1.1 Student Registration and Onboarding

- Email as user Id and email verification process not required
- Social login integration (Google, LinkedIn)
- Resume upload and parsing with manual data entry option
- Multi-step profile creation wizard

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

- Skill self-assessment questionnaire (optional)
- Career objectives and preferences survey (optional)
- Portfolio and project links collection (optional)
- Personality assessment mini-quiz (optional)

3.1.2 Employer Registration and Onboarding

- Company verification process (LinkedIn, business email domain)
- Social login integration (Google, LinkedIn)
- Company profile creation or link to their relevant website page
- Internship program details or link to their relevant website page
- Team and mentor profiles or link to their relevant website page
- Work environment and culture description or link to their relevant website page
- Technology stack information or link to their relevant website page
- Internship structure template or link to their relevant website page

3.1.3 Administrator Accounts

- Super admin dashboard
- Limited access rights to employees
- Content moderator accounts
- Analytics access profiles
- Support agent accounts
- Auto-email
- Social Media
- Campaigns with links
- Discount Codes
- Payment Gateway Integration
- Billings and Invoices

3.2 Student User Features

3.2.1 Profile Management

- Resume upload, parsing, and updating
- LinkedIn Page URL
- Portfolio management
- Skill matrix with proficiency indicators
- Education and certification tracking – verifiable links, information on LinkedIn etc
- Project showcase with descriptions and links
- Profile completion gamification
- Profile visibility controls
- Custom URL for public profile sharing

3.2.2 Internship Discovery

- Personalized job recommendations
- Advanced search with multiple filters
- Saved searches and alerts
- Role-based browsing
- Location-based discovery
- Skill-match indicator
- Student Readiness Score display
- Career Path Alignment indicator/ Graph
- Job bookmark and collections creation

3.2.3 Application Management

- One-click apply functionality
- Application status tracking
- Application history and analytics
- Document management for submissions
- Follow-up reminder system
- Interview scheduling
- Pre-interview preparation resources

3.2.4 Career Development

- Skill gap analysis with learning recommendations
- Career path visualization
- Industry insights and salary data
- Learning resource recommendations
- Certification pathway suggestions
- Mentor matching functionality
- Peer success stories

3.2.5 Communication Tools

- In-app messaging with employers
- Interview request handling
- Notification center and preferences
- Email integration and alerts
- Calendar integration for scheduling

Email id:avani@zetheta.com
Contact number: +91-91362 49369

3.3 Employer User Features

3.3.1 Company Profile Management

- Company information and branding
- Team member management
- Internship program description
- Photo and video gallery
- Company culture showcase
- Technology stack information
- Previous intern testimonials
- Diversity and inclusion policy section

3.3.2 Internship Posting Management

- Structured job posting form
- Skill requirement definition
- Automated matching criteria generation
- Job post templates and saving
- Job post analytics
- Duplicate and edit functionality
- Multiple location support
- Remote work option settings

3.3.3 Candidate Discovery

- AI-powered candidate recommendations
- Advanced search with multiple filters
- Saved candidate lists
- Skill-match filtering
- Education-based filtering
- Project experience filtering
- Readiness score filtering
- Batch actions for candidates

3.3.4 Application Review

- Application inbox with status management
- Candidate comparison view
- Resume and portfolio viewer
- Application rating system
- Team collaboration for reviews
- Interview scheduling assistant
- Rejection and acceptance template messages
- Candidate pipeline analytics

Email id:avani@zetheta.com
Contact number: +91-91362 49369

3.3.5 Communication Tools

- In-app messaging with candidates
- Bulk messaging to selected candidates
- Automated response scheduling
- Email templates management
- Interview request scheduler
- Calendar integration
- Video meeting integration

3.4 Administrator Features

3.4.1 User Management

- User account management
- Role and permission management
- User activity monitoring
- Account verification and approval
- Suspension and ban functionalities
- Password reset handling
- User feedback handling

3.4.2 Content Management

- Job posting approval and moderation
- Profile content moderation
- Message monitoring for policy violations
- Reported content handling
- Content quality guidelines management

3.4.3 Platform Analytics

- User acquisition and engagement metrics
- Matching success analytics
- Feature usage statistics
- Performance monitoring dashboards
- A/B testing management
- Conversion funnel analysis
- Retention and churn analysis

3.4.4 System Configuration

- Email notification settings
- Integration management

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- AI feature tuning parameters
- Security settings
- Backup and restoration tools
- Maintenance mode control

3.5 AI Features

3.5.1 Resume Parsing System

- Document format support (PDF, DOCX, TXT)
- Text extraction and cleaning
- Named entity recognition for:
  - Personal information
  - Education history
  - Work experience
  - Skills and technologies
  - Certifications
  - Projects
- Skill taxonomy mapping
- Confidence scoring for extracted data
- Manual correction interface
- Batch processing capabilities

3.5.2 Job Readiness Scoring Engine

- Skill match calculation
- Education relevance assessment
- Experience evaluation
- Project relevance scoring
- Certification value calculation
- Weighted scoring algorithm
- Explainable AI interface for score breakdown
- Improvement recommendations
- Benchmark comparisons with peer groups

Algorithm: Weighted Feature Scoring with Normalized Output
Input: Student profile data (skills, college, academic discipline, education, experience, projects)
Output: Readiness score (0-100)

1. Define feature weights:
   - Technical Skills Match: 35%
   - Education Relevance: 20%
   - Project Experience: 25%
   - Soft Skills Assessment: 15%
   - Certifications: 5%

Email id:avani@zetheta.com
Contact number: +91-91362 49369

2. For each job posting:
   a. Extract required skills, education, experience
   b. Calculate Technical Skills Match:
      - Use TF-IDF vectorization on job requirements and student skills
      - Calculate cosine similarity between vectors
      - Normalize to 0-100 scale

   c. Calculate Education Relevance:
      - Map degree relevance to job field (0-100)
      - Apply multiplier based on degree level (Bachelor's: 0.8, Master's: 1.0, PhD: 1.2)

   d. Calculate Project Experience:
      - Count relevant projects using keyword matching
      - Weight by project complexity (determined by description length, technologies used)
      - Normalize to 0-100 scale

   e. Calculate Soft Skills Assessment:
      - Extract soft skills mentions from resume
      - Match against job soft skill requirements
      - Normalize to 0-100 scale

   f. Calculate Certification Score:
      - Count relevant certifications
      - Weight by recency and relevance
      - Normalize to 0-100 scale

3. Calculate final score:
   Readiness Score = (0.35 × Technical Skills) + (0.20 × Education) + (0.25 × Projects) +
   (0.15 × Soft Skills) + (0.05 × Certifications)

4. Apply normalization to handle outliers

Implementation: Use scikit-learn for vectorization and similarity calculations,
NLTK for text processing, and custom Python functions for scoring logic.

ALTERNATIVE ALGORITHM could also be used to simplify to be based without job matching (see code section later on). Ideally we work on both algorithms to check which works better and achieve Minimum Viability Plan (MVP) with a simpler algorithm.

3.5.3 Career Growth Prediction System

- Historical career data analysis
- Industry trend integration
- Education pathway impact

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Skill acquisition modeling
- Salary progression forecasting
- Role transition probability calculation
- Career path visualization
- Alternative career option exploration
- Personalized development recommendations

Algorithm: Hybrid Career Path Prediction (HCPP)
Input: Student profile, historical career progression data
Output: Predicted career path, estimated salary progression

1. Data Preparation:
   a. Collect historical career progression data from similar profiles
   b. Extract features: education level, field of study, technical skills, internship experiences, job roles, salary progression

2. Clustering Phase:
   a. Use K-means clustering to group similar career profiles
   b. Features used: education, skills, early career choices
   c. Output: Career cluster assignment for the student

3. Sequence Modeling:
   a. For each cluster, build a Markov Chain model of career transitions
   b. Calculate transition probabilities between job roles
   c. Identify most common paths within the cluster

4. Regression for Salary Prediction:
   a. Train a Gradient Boosting Regressor on historical data
   b. Features: education, skills, experience, job role, industry, location
   c. Target: Salary at different career stages (1-year, 3-year, 5-year)

5. Generate Predictions:
   a. Identify the student's career cluster
   b. Use Markov Chain to generate top 3 most likely career paths
   c. For each path, predict salary progression using the regression model
   d. Calculate confidence intervals for predictions

Implementation: Use scikit-learn for clustering and regression, NetworkX for Markov Chain implementation, and pandas for data manipulation.

3.5.4 Skill Gap Analysis

- Required skill extraction from job postings
- Student skill proficiency assessment

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Gap identification algorithm
- Critical vs. optional skill classification
- Learning resource mapping
- Development timeline estimation
- Prioritized action plan generation
- Progress tracking functionality
- Industry benchmark comparison

Algorithm: Dynamic Skill Gap Analysis
Input: Student's current skills, target job requirements
Output: Identified skill gaps, learning recommendations

1. Skill Extraction:
   a. Extract skills from student resume using NER and keyword extraction
   b. Normalize skills (handle synonyms, abbreviations)
   c. Categorize skills (technical, soft, domain-specific)

2. Job Requirement Analysis:
   a. Extract and normalize required skills from job postings
   b. Determine skill importance using TF-IDF scoring
   c. Categorize required skills

3. Gap Identification:
   a. Compare student skills with required skills
   b. Identify missing skills
   c. Calculate importance score for each missing skill
   d. Rank skill gaps by importance score

4. Learning Path Generation:
   a. For each skill gap:
      - Query learning resource database
      - Filter by quality rating, estimated completion time
      - Sort by relevance and effectiveness
   b. Group related skills for learning efficiency
   c. Generate step-by-step learning path with estimated timelines

Implementation: Use spaCy for NER, scikit-learn for TF-IDF, custom Python functions for matching and recommendation logic.

3.5.5 AI Interview Pre-screening

- Video response recording
- Audio transcription
- Natural language understanding
- Sentiment analysis
- Technical answer evaluation

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Communication skill assessment
- Behavioral trait analysis
- Standardized scoring system
- Interview analytics for employers
- Practice mode for students

Algorithm: Multi-modal Interview Assessment
Input: Video interview responses
Output: Competency assessment report

1. Speech Processing:
   a. Transcribe audio using speech-to-text API
   b. Extract speech features: pace, tone, clarity, filler words

2. Text Analysis:
   a. Perform sentiment analysis on responses
   b. Analyze complexity and coherence of answers
   c. Extract key concepts and relevance to questions
   d. Check for technical accuracy in responses

3. Visual Analysis:
   a. Detect facial expressions and emotions
   b. Analyze eye contact and engagement
   c. Evaluate body language and posture
   d. Track confidence indicators

4. Competency Scoring:
   a. Define competency framework (communication, technical knowledge, problem-solving, etc.)
   b. Map multimodal features to competencies
   c. Calculate weighted scores for each competency
   d. Generate percentile rankings compared to peer responses

5. Report Generation:
   a. Summarize strengths and areas for improvement
   b. Provide specific examples from the interview
   c. Generate final competency scores
   d. Include confidence level in assessment

Implementation: Use Google Cloud Speech-to-Text API for transcription, Hugging Face Transformers for text analysis, MediaPipe for facial expression analysis, and custom Python functions for scoring and report generation.

3.5.6 Matchmaking Algorithm

- Multi-factor compatibility scoring

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Two-way preference matching
- Contextual relevance calculation
- Behavioral data integration
- Application success prediction
- Mutual interest detection
- Diversity and inclusion parameters
- Geographic matching optimization
- Feedback-based learning

Hybrid Recommendation System
Algorithm: Hybrid Recommendation System for Intern-Employer Matching
Input: Student profiles, job listings, interaction history
Output: Ranked matches for students and employers

1. Content-Based Filtering:
   a. Create feature vectors for students and jobs using:
      - Skills (weighted by proficiency)
      - Education (field, degree level)
      - Location preferences
      - Industry interests
      - Job requirements
   b. Calculate cosine similarity between student and job vectors
   c. Rank jobs by similarity score for each student

2. Collaborative Filtering:
   a. Create user-item interaction matrix from application history
   b. Use matrix factorization (SVD) to identify latent factors
   c. Generate recommendations based on similar students' applications and outcomes

3. Contextual Filtering:
   a. Apply filters based on:
      - Geographic constraints
      - Timing requirements (availability, start dates)
      - Salary expectations
      - Work arrangement preferences (remote, hybrid, on-site)
   b. Adjust similarity scores based on context

4. Hybrid Ranking:
   a. Combine scores from content-based and collaborative filtering:
      Final Score = α × ContentScore + (1-α) × CollaborativeScore
      where α is adaptively determined based on data availability
   b. Apply contextual filters
   c. Generate final ranked list of recommendations

Implementation: Use scikit-learn for vectorization and similarity calculation, Surprise library for collaborative filtering, and custom Python functions for hybrid scoring and ranking.

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Two-Way Preference Matching

Algorithm: Stable Matching Allocation for Mutual Preference
Input: Student preferences, employer preferences
Output: Stable matching assignments

1. Preference Collection:
   a. Students rank employers based on interest level (explicit) and recommendation score (implicit)
   b. Employers rank students based on qualification assessment and fit

2. Initialize Algorithm:
   a. All students start unmatched
   b. Each student has a list of preferred employers

3. Matching Process (Gale-Shapley algorithm):
   a. While there are unmatched students with remaining preferences:
      - Current student proposes to their highest-ranked unproposed employer
      - If employer is unmatched, accept proposal
      - If employer is matched but prefers current student to current match, accept new proposal and release previous match
      - If employer is matched and prefers current match, reject proposal

4. Match Refinement:
   a. Calculate satisfaction score for each match
   b. Identify potentially unstable matches
   c. Suggest alternative matches where mutual benefit is likely higher

Implementation: Use custom Python implementation of Gale-Shapley algorithm with modifications for partial preferences and preference strength.

3.6 Communication System

3.6.1 In-app Messaging

- Real-time chat functionality
- Message read receipts
- Typing indicators
- File attachment support
- Template message library
- Message search functionality
- Conversation archiving
- Message scheduling
- Group messaging for teams

Email id:avani@zetheta.com
Contact number: +91-91362 49369

3.6.2 Notification System

- Push notifications
- Email notifications
- SMS alerts (optional)
- Notification preference management
- Critical alert settings
- Event-based notification rules
- Notification read tracking
- Batch notification management

3.6.3 Calendar Integration

- Interview scheduling
- Availability sharing
- Calendar sync with external systems
- Reminder settings
- Time zone handling
- Recurring event support
- Calendar export options

3.7 Reporting and Analytics

3.7.1 Student Analytics

- Application success rate
- Profile view statistics
- Skill competitiveness analysis
- Career progress tracking
- Engagement metrics
- Learning progress analytics
- Recommendation relevance feedback

3.7.2 Employer Analytics

- Applicant funnel metrics
- Job posting performance
- Candidate quality metrics
- Time-to-fill analysis
- Source effectiveness
- Team activity reports
- Cost-per-hire tracking
- Diversity metrics

Email id:avani@zetheta.com
Contact number: +91-91362 49369

3.7.3 System Analytics

- User engagement metrics
- Feature usage statistics
- Performance monitoring
- Matching algorithm effectiveness
- AI accuracy metrics
- User satisfaction indicators
- Retention analysis

3.7.3 Optional input additions:

Additional inputs sought to be collected for improving the AI features:

3.1 From Students:

1. **Preference Questionnaire**:
   - Work environment preferences (corporate, startup, non-profit)
   - Industry interests (ranked)
   - Remote/hybrid/in-person preferences
   - Preferred company size
   - Salary expectations
2. **Skill Self-Assessment**:
   - Technical skills with proficiency levels (1-5)
   - Soft skills with examples
   - Languages spoken and proficiency
3. **Portfolio Links**:
   - GitHub/GitLab repositories
   - Personal websites/blogs
   - Project demonstrations
   - Design portfolios (for design roles)
4. **Career Objectives**:
   - Short-term goals (0-1 year)
   - Medium-term goals (1-3 years)
   - Long-term aspirations (3-5+ years)
   - Preferred job roles
5. **Availability and Constraints**:
   - Start/end date availability
   - Hours per week available
   - Geographic constraints
   - Travel capabilities
6. **Personality Assessment**:
   - Brief personality questionnaire (Big Five traits)
   - Work style preferences
   - Team role tendencies

Email id:avani@zetheta.com
Contact number: +91-91362 49369

7. **Learning Preferences**:
    - Preferred learning methods
    - Areas willing to develop
    - Mentorship expectations

3.2 From Employers:

1. **Company Culture Profile**:
    - Work environment description
    - Team dynamics
    - Management style
    - Core values
2. **Internship Structure Details**:
    - Project descriptions
    - Mentor availability
    - Performance evaluation methods
    - Learning opportunities
3. **Technology Stack**:
    - Required technologies
    - Nice-to-have technologies
    - Development methodologies
    - Tools and platforms used
4. **Growth Opportunities**:
    - Potential for full-time conversion
    - Career progression examples
    - Training programs offered
5. **Non-Technical Requirements**:
    - Soft skills importance rankings
    - Communication expectations
    - Team collaboration requirements
6. **Flexibility Parameters**:
    - Remote work policies
    - Schedule flexibility
    - Location requirements

4. Non-Functional Requirements

4.1 Performance Requirements

- Page load time under 2 seconds
- Support for 500+ concurrent users
- AI feature response time under 5 seconds
- Database query optimization for large datasets
- Efficient handling of file uploads/downloads
- Mobile responsiveness with minimal latency

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- API response time under 300ms for critical endpoints

4.2 Scalability Requirements

- Horizontal scaling capability
- Database sharding preparation
- Microservice architecture for AI components
- Caching system for frequently accessed data
- Resource utilization optimization
- Batch processing for intensive operations
- Auto-scaling configuration for cloud deployment

4.3 Security Requirements

- HTTPS encryption for all communications
- Data encryption at rest
- Secure password policy enforcement
- Two-factor authentication option
- Role-based access control
- Input validation and sanitization
- Protection against SQL injection
- XSS and CSRF protection
- Regular security audits (OWASP guidelines)
- Privacy by design implementation
- Data retention and deletion policies
- Security logging and monitoring

4.4 Usability Requirements

- Intuitive navigation structure
- Responsive design for all devices
- Accessibility compliance (WCAG 2.1 AA)
- Consistent UI patterns
- Clear error messaging
- Progressive enhancement approach
- Help documentation and tooltips
- Multi-language support preparation
- User feedback collection mechanisms
- Streamlined workflows with minimal steps

4.5 Reliability Requirements

- 99.9% uptime target
- Graceful degradation of features

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Comprehensive error handling
- Automated backup system
- Disaster recovery plan
- Transaction integrity assurance
- Monitoring and alerting system
- Fault tolerance for critical components

4.6 Maintainability Requirements

- Modular codebase organization
- Comprehensive documentation
- Code quality standards compliance
- Automated testing framework
- Continuous integration/deployment setup
- Version control best practices
- Dependency management strategy
- Technical debt monitoring
- Knowledge transfer procedures

5. Technical Architecture

5.1 Frontend Architecture

- Framework: Next.js 14.x with Pages Router, TypeScript
- State Management: React Query for server state (data fetching and caching), React Context API for local state, Redux only if justified for complex global state requirements
- Styling: Tailwind CSS for responsive design with custom theming
- Headless UI + custom components
- UI Component Library: Custom components with Tailwind CSS
- Responsive Design: Mobile-first approach with breakpoints for tablet and desktop
- Asset Pipeline: Next.js built-in asset pipeline.
- Client-side Validation: Formik with Yup schema validation
- API Communication: Next.js built-in fetch() with React Query (alternative: Axios with interceptors if needed)
- Authentication: JWT implementation with NextAuth.js, tokens stored in httpOnly cookies
  Analytics: Custom event tracking
- Accessibility: ARIA compliance and keyboard navigation
- Error Handling: Global error boundary with fallback UI

5.2 Backend Architecture

- **Primary Framework: Laravel 11.x** for core services and API endpoints
- **Secondary Framework: FastAPI for Python-based AI microservices** (self-hosted on free platforms).
- **API Design:** RESTful principles with versioning, orchestrated through Laravel Gateway

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- **Authentication:** Laravel Sanctum with Supabase Auth integration
- **Application Structure: Controllers:** RESTful resource controllers for CRUD operations, **Services Layer:** Business logic separated from controllers, **Middleware:** Request validation, authentication, logging
- **Form Validation:** Laravel's built-in validation system
- **Background Processing:** Database-based queue for resource-intensive tasks
- **Authorization:** Role-Based Access Control (RBAC) with permission inheritance
- **Rate Limiting:** Tiered approach based on endpoint sensitivity
- **Caching Strategy:** File-based caching to reduce database queries. Redis for session data and frequent queries
- **Notification Service:** Event-driven architecture
- **Logging:** Structured logging with severity levels

5.3 Database Architecture

- Primary Database: Supabase PostgreSQL (500 MB free tier) with pgvector extension for AI embeddings (aslo for Vector Database)
- Secondary Database: SQLite for backup and local development
- NoSQL Database: MongoDB for unstructured data and document storage (if needed)
- Caching Layer: Redis for session data, frequent queries, for caching and real-time features
- Schema Design: Normalized structure to minimize redundancy
- Connection: Supabase JavaScript/PHP client, , REST API access via Supabase client
- Data Models: Eloquent ORM with relationship mapping
- Database Access: Repository pattern with query builders
- Migration Strategy: Versioned migrations with rollback capability
- Backup Strategy: Automated daily backups with point-in-time recovery
- Data Partitioning: Vertical partitioning by feature domain
- Query Optimization: Prepared statements and eager loading
- Data Models: Eloquent ORM with relationship mapping
- NoSQL Database: MongoDB for unstructured data and document storage

5.4 AI Service Architecture

- CV Parsing: - Apache Tika 2.9.1 (text extraction), PyPDF2 3.0.0 (PDF-specific processing), Unstructured.io (open-source alternative for complex docs), OCR (if needed): Tesseract OCR (scanned PDFs), if needed
- Model Serving: FastAPI endpoints for model inference
- Data Pipeline: ETL processes for model training data
- Vector Embeddings: Sentence transformers for text similarity (all-MiniLM-L6-v2)
- NLP Processing: spaCy with small/medium models for text processing and named entity recognition
- Machine Learning: Scikit-learn for traditional ML algorithms
- Deep Learning: Hugging Face transformers for NLP tasks (via free inference API)
- Model Management: Version control for models and datasets
- Feature Store: Centralized repository for ML features

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Inference Optimization: Model quantization and caching
- LLM: Primary - Mistral 7B (locally run with llama.cpp), fall back to OpenAI GPT-3.5-turbo (free tier with limits)
- Hugging Face models via free inference API
- Model Management: Version control for models and datasets
- Feature Store: Centralized repository for ML features
- Inference Optimization: Model quantization and caching
- Open-source alternatives for commercial AI services

5.5 Integration Architecture

- API Gateway: API management and documentation
- Service Communication: Synchronous: HTTP/REST, Asynchronous: Message queue, Facade pattern for internal services
- External Services: Supabase client for database/storage/authentication
- Event System: Laravel's event broadcasting for real-time features
- File Handling: Supabase storage integration for document uploads
- Email Integration: SMTP connection to free email services
- Webhook Support: For third-party integrations
- Monitoring Integration: Prometheus for metrics collection
- Error Tracking: Sentry for exception monitoring
- Analytics Pipeline: Event collection and processing system

5.6 Deployment Architecture

- Containerization: Docker for application components
- Container Orchestration: Kubernetes for production (future scaling)
- Environment Strategy: Development, Staging, Production
- Infrastructure as Code: Terraform for cloud resources
- Configuration Management: Environment variables with secret management
- CDN Integration: For static assets
- Database Deployment: Managed database services
- Hosting: Vercel/Netlify (free tier) for frontend deployment, InfinityFree/000webhost for PHP backend hosting, Deta.sh/Koyeb for hosting FastAPI microservices (free tier), GitHub Pages for documentation and static assets, GitHub Actions for CI/CD automation (free for public repositories)
- Version Control: GitHub for source code management
- Database Changes: Migration scripts for schema updates
- Backup Strategy: Scheduled exports to local storage
- Failover Plan: SQLite database as backup if Supabase limits are reached
- Monitoring: Basic logging to text files for error tracking
- Monitoring Stack: Prometheus and Grafana for metrics, Sentry (free tier) for error tracking, Basic logging to text files as fallback
- Documentation: GitHub Wiki

Email id:avani@zetheta.com
Contact number: +91-91362 49369

5.7 External Services

- **Email**: MailTrap (free tier) for development / Brevo (free tier) for production
- **File Upload**: Direct upload to hosting/local storage
- **Search:** PostgreSQL full-text search

6. UI/UX Design

6.1 Design System

- Color Palette: Primary, secondary, accent colors with light/dark mode variants
- Typography: Hierarchical type system with responsive scaling
- Spacing System: Consistent spacing scale based on 4px increment
- Component Library: Reusable UI components with props for variations
- Animation Guidelines: Duration, easing, and purpose-driven motion
- Icon System: Consistent icon style with semantic meaning
- Accessibility Standards: Color contrast requirements, focus states
- Responsive Breakpoints: Mobile, tablet, desktop, large desktop

6.2 User Flows

- Student Registration and Onboarding Flow
- Employer Registration and Onboarding Flow
- Job Search and Application Flow
- Candidate Discovery and Assessment Flow
- Communication Initiation Flow
- Interview Scheduling Flow
- Feedback and Rating Flow
- Profile Completion and Enhancement Flow

6.3 Screen Designs

- Detailed wireframes for all core screens
- Interactive prototypes for key user journeys
- Responsive variants for mobile, tablet, and desktop
- State variations (empty, loading, error, success)
- Accessibility annotations for complex components
- Animation specifications for transitions

7. Implementation Phases

7.1 Phase 1: Foundation

- Project setup and repository initialization

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Authentication system implementation
- Basic user management
- Database schema setup
- Core API endpoints
- Frontend scaffolding
- CI/CD pipeline setup
- Development environment configuration

7.2 Phase 2: Core Features

- Student profile management
- Employer profile management
- Basic job posting functionality
- Simple search and filtering
- Resume upload and basic parsing
- Messaging system implementation
- Notification system setup
- Admin panel foundation

7.3 Phase 3: AI Integration

- Advanced resume parsing implementation
- Job readiness scoring algorithm
- Basic matchmaking algorithm
- Skill gap analysis foundation
- AI service infrastructure setup
- Vector database integration
- Model training pipeline setup
- AI feature API endpoints

7.4 Phase 4: Advanced Features

- Career prediction algorithm implementation
- Interview pre-screening system
- Enhanced matching algorithm
- Learning recommendation engine
- Advanced analytics dashboard
- Performance optimization
- Security hardening
- User experience refinement

7.5 Phase 5: Testing & Deployment

- Comprehensive testing (unit, integration, system)

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- User acceptance testing
- Performance testing and optimization
- Security audit and remediation
- Documentation finalization
- Production deployment preparation
- Initial user onboarding
- Monitoring setup and verification

8. Testing Strategy

8.1 Unit Testing

- Framework: PHPUnit for Laravel, Jest for Next.js, pytest for FastAPI
- Coverage Target: 70% for critical components
- Test Types: Function-level tests, component tests
- Mocking Strategy: External dependencies and services
- Automation: Test runs on each commit

8.2 Integration Testing

- API Testing: Postman collections for endpoint verification
- Database Testing: Test transactions and rollbacks
- Service Integration Tests: Communication between services
- Authentication Flow Testing: Complete auth scenarios

8.3 System Testing

- End-to-End Testing: Cypress for critical user journeys
- Load Testing: Artillery for performance under load
- Security Testing: OWASP ZAP for vulnerability scanning
- Cross-browser Testing: Major browser compatibility

8.4 User Acceptance Testing

- Test Scenarios: Real-world use cases
- Test Users: Representative sample of target audience
- Feedback Collection: Structured questionnaires
- Iteration: Refinement based on feedback

9. Maintenance and Support

9.1 Post-Launch Support

- Bug tracking and prioritization process

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Feature request handling procedure
- Security update protocol
- Regular maintenance schedule
- User feedback analysis

9.2 Monitoring and Alerting

- Performance monitoring setup
- Error tracking configuration
- Security monitoring system
- Usage analytics collection
- Automated alerting thresholds

9.3 Documentation

- System architecture documentation
- API documentation
- User guides for all user types
- Administrator documentation
- Development guides for future contributors
- Deployment and configuration documentation

10. Risks and Mitigations

10.1 Technical Risks

- Using both Laravel and FAST API increases complexity.
    - Mitigation: If the project is not running on schedule, consider consolidation to primarily Laravel.
- AI algorithm accuracy limitations
    - Mitigation: Progressive improvement approach, clear confidence indicators
- Scalability challenges with free-tier services
    - Mitigation: Efficient resource usage, caching, optimization
- Security vulnerabilities
    - Mitigation: Regular security audits, follow OWASP guidelines

10.2 Project Risks

- Limited experience of student interns
    - Mitigation: Clear task breakdown, detailed documentation, peer programming
- Scope creep
    - Mitigation: Strict prioritization, MVP definition, change management process
- Integration challenges
    - Mitigation: Early prototyping of critical integrations, fallback options

Email id:avani@zetheta.com
Contact number: +91-91362 49369

10.3 Business Risks

- Low user adoption
  - Mitigation: Early user feedback, focus on core value proposition
- Data quality issues
  - Mitigation: Data validation, cleaning processes, manual verification options
- Ethical considerations in AI matching
  - Mitigation: Transparency in algorithms, bias detection, human oversight

11. Success Criteria

11.1 Technical Success Metrics

- System uptime: 99.9%
- Average page load time: < 2 seconds
- API response time: < 300ms for 95% of requests
- AI feature accuracy: > 85% for recommendations
- Test coverage: > 70% for critical components
- Security: Zero high or critical vulnerabilities

11.2 User Success Metrics

- Student profile completion rate: > 80%
- Employer job posting completion rate: > 90%
- Application success rate improvement: > 25% vs. traditional methods
- User satisfaction rating: > 4/5 stars
- Return visitor rate: > 60%
- Time-to-match: < 5 days average

11.3 Business Success Metrics

- Active user growth: 20% month-over-month
- User retention: > 70% after 3 months
- Feature adoption: > 60% of users using AI features
- Platform reliability: < 5 critical bugs per month

Implementation Plan

Team Structure and Roles

For effective execution, assign specialized roles to each intern based on their strengths:

1. **Frontend Development**
   - Responsible for Next.js implementation

Email id:avani@zetheta.com
Contact number: +91-91362 49369

   - UI/UX implementation
   - Mobile responsiveness
   - Client-side validation
2. **Backend Development - Laravel**
   - API development
   - Database operations
   - Authentication system
   - Business logic implementation
3. **AI/ML ChatBot Engineers**
   - Algorithms
   - FastAPI microservices
   - ML model implementation
   - Integration with main application
   - Model training and evaluation
4. **Full-stack Developer**
   - Cross-functional implementation
   - Integration between frontend and backend
   - Testing and debugging
   - DevOps and deployment
5. **Project Lead**
   - Project coordination
   - Overall implementation
   - UI/UX design
   - Documentation
   - Testing and quality assurance

8-Week Development Timeline

Week 1: Project Setup and Foundation

- **Days 1-2: Project Initialization**
  - Set up Git repository and branch strategy
  - Initialize Next.js project structure
  - Initialize Laravel project structure
  - Set up FastAPI project structure
  - Configure Docker development environment
- **Days 3-5: Core Architecture**
  - Design and implement database schema
  - Set up authentication system (Laravel)
  - Create basic API structure
  - Set up frontend routing
  - Implement CI/CD pipeline with GitHub Actions

Email id:avani@zetheta.com
Contact number: +91-91362 49369

Week 2: Core User Functionality

- **Days 1-3: User Management**
  - Implement user registration and login
  - Create user profile components
  - Implement role-based access control
  - Design and implement account settings
- **Days 4-5: Profile Management**
  - Create student profile form components
  - Create employer profile form components
  - Implement profile update functionality
  - Create resume upload component

Week 3: Resume Parsing and Basic Matching

- **Days 1-3: Resume Parsing**
  - Implement document upload and storage
  - Configure NLP pipeline for text extraction
  - Develop entity recognition for skills, education, experience
  - Create parsed resume viewer component
- **Days 4-5: Basic Matching**
  - Implement simple keyword matching algorithm
  - Create job posting form components
  - Design and implement job search functionality
  - Create job recommendation component

Week 4: Advanced UI and Communication

- **Days 1-3: Enhanced UI Components**
  - Implement responsive dashboard for students
  - Implement responsive dashboard for employers
  - Create data visualization components
  - Implement notification system
- **Days 4-5: Messaging System**
  - Design and implement chat interface
  - Set up real-time messaging with Laravel Echo
  - Create message notification system
  - Implement file sharing in messages

Week 5: AI Feature Implementation (Part 1)

- **Days 1-3: Readiness Score System**
  - Implement skill extraction and matching algorithm
  - Create scoring model and training pipeline

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

    - Develop API endpoints for readiness calculation
    - Create readiness score visualization components
- **Days 4-5: Skill Gap Analysis**
    - Implement skill gap identification algorithm
    - Create learning resource recommendation system
    - Develop API endpoints for skill gap analysis
    - Create skill gap visualization components

Week 6: AI Feature Implementation (Part 2)

- **Days 1-3: Career Prediction**
    - Implement career path prediction algorithm
    - Set up salary estimation model
    - Develop API endpoints for career prediction
    - Create career path visualization components
- **Days 4-5: Interview Bot**
    - Implement basic interview question system
    - Create interview recording and processing system
    - Develop API endpoints for interview analysis
    - Create interview feedback visualization components

Week 7: Advanced Matching and Integration

- **Days 1-3: Advanced Recommendation System**
    - Implement hybrid recommendation algorithm
    - Create personalization features
    - Optimize matching performance
    - Develop A/B testing framework for algorithm comparison
- **Days 4-5: System Integration and Optimization**
    - Integrate all AI services with main application
    - Optimize database queries and caching
    - Implement rate limiting and API security
    - Create comprehensive error handling system

Week 8: Testing, Optimization, and Deployment

- **Days 1-3: Testing and Bug Fixes**
    - Conduct comprehensive testing (unit, integration, E2E)
    - Fix identified bugs and issues
    - Optimize frontend performance
    - Conduct security audit and fix vulnerabilities
- **Days 4-5: Deployment and Documentation**
    - Set up production environment
    - Deploy application to cloud services
    - Create user documentation

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

- Create technical documentation
- Conduct final review and handover

Development Guidelines

Code Quality Standards

- Follow PSR-12 coding standard for PHP
- Use Airbnb JavaScript style guide for JS/React
- Write meaningful comments and documentation
- Use typed interfaces in TypeScript
- Create reusable components with clear props interfaces

Git Workflow

- Use feature branches for all new features
- Create pull requests for code review
- Require at least one review before merging
- Write descriptive commit messages
- Regularly pull from main to avoid merge conflicts

Testing Requirements

- Write unit tests for critical business logic
- Implement integration tests for API endpoints
- Create E2E tests for critical user flows
- Achieve at least 70% code coverage for core functionality
- Test on multiple browsers and screen sizes

Documentation Requirements

- Document all API endpoints using Swagger/OpenAPI
- Create component documentation with PropTypes/TypeScript
- Write setup instructions for local development
- Create user guides for key features
- Document database schema and relationships

Simplified Implementation for Short Timeline

To ensure completion within 2 months, simplify implementation by:

1. **Prioritize Core Features:**
   - Focus on essential user journeys first
   - Defer advanced analytics and reporting features

Email id:avani@zetheta.com
Contact number: +91-91362 49369

   - Start with simple matching before implementing advanced algorithms
2. **Use Ready-made Components:**
   - Leverage Tailwind UI components
   - Use pre-built React components
   - Incorporate open-source ML models instead of training from scratch
3. **Implement Incremental AI Features:**
   - Begin with rule-based matching before ML implementation
   - Use pre-trained NLP models
   - Implement simpler versions of algorithms first, then enhance
4. **Optimize Development Workflow:**
   - Use pair programming for complex features
   - Hold daily standup meetings (15 min max)
   - Weekly code reviews
   - Bi-weekly retrospectives
5. **Technical Shortcuts (When Necessary):**
   - Use existing libraries for complex functionality
   - Implement server-side rendering only for critical pages
   - Begin with monolithic architecture before microservices
   - Use managed services instead of self-hosted solutions

Progress Tracking and Quality Control

1. **Weekly Deliverables Schedule:**
   - End of Week 1: Project setup and authentication
   - End of Week 2: User profiles and basic job posting
   - End of Week 3: Resume parsing and job search
   - End of Week 4: Messaging and notification system
   - End of Week 5: Readiness score and skill gap analysis
   - End of Week 6: Career prediction and interview system
   - End of Week 7: Advanced matching and system integration
   - End of Week 8: Testing and deployment
2. **Code Review Process:**
   - Mandatory review for all PRs
   - Automated linting and testing
   - Weekly code quality audits
   - Performance benchmarking
3. **Testing Milestones:**
   - Unit tests: 70% coverage for core services
   - Integration tests: All critical API endpoints
   - E2E tests: Main user journeys
   - Performance tests: Response time < 300ms for key endpoints

Technical Resources and Learning Materials

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

Frontend Development

- Next.js Documentation
- React Hooks Guide
- Tailwind CSS Documentation
- Redux Toolkit Guide

Backend Development

- Laravel Documentation
- Laravel API Development
- Eloquent ORM Guide
- Laravel Authentication

AI/ML Development

- FastAPI Documentation
- Scikit-learn Tutorials
- Hugging Face Transformers
- spaCy Documentation

DevOps and Deployment

- GitHub Actions Documentation
- Docker for Development
- Vercel Deployment Guide
- AWS Free Tier Services

Laravel & PHP Resources

1. **Laravel Official Documentation**
    - Laravel Documentation - Comprehensive guide to all Laravel features
    - Laravel Blade Templates - Learn Blade templating system
2. **PHP Fundamentals**
    - PHP Manual - Official PHP documentation
    - PHP: The Right Way - Modern PHP best practices
3. **Free Laravel Courses**
    - Laracasts - Laravel From Scratch - Excellent beginner series
    - Laravel Daily YouTube Channel - Practical tutorials

Supabase Resources

1. **Supabase Documentation**
    - Supabase Docs - Official documentation

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

   - Supabase JavaScript Client - Client library
2. **Supabase Integration Tutorials**
   - Supabase with Laravel - Integration guide
   - Supabase Auth - Authentication setup
3. **Free Supabase Learning Resources**
   - Supabase YouTube Channel - Official tutorials
   - FreeCodeCamp Supabase Course - Comprehensive tutorial

Tailwind CSS Resources

1. **Official Documentation**
   - Tailwind CSS Docs - Complete reference
   - Tailwind CSS with Laravel - Integration guide
2. **Free Tailwind Tutorials**
   - Tailwind CSS Crash Course - Quick introduction
   - Tailwind CSS Components - Free component examples

Hosting Resources

1. **InfinityFree Guides**
   - InfinityFree Documentation - Official documentation
   - Deploying Laravel on InfinityFree - Step-by-step guide
2. **000WebHost Resources**
   - 000WebHost Tutorials - Official guides
   - Laravel on 000WebHost - Deployment tutorial

Database Design Resources

1. **PostgreSQL Learning**
   - PostgreSQL Tutorial - Free comprehensive guide
   - Supabase PostgreSQL Features - Supabase-specific features
2. **Database Optimization**
   - Database Normalization Guide - Normalization basics
   - PostgreSQL Performance Tips - Query optimization

Algorithm Implementation Resources

1. **PHP Algorithm Implementation**
   - PHP Data Structures and Algorithms - Open-source implementations
   - Regex Tutorial for PHP - Pattern matching
2. **Text Processing in PHP**
   - Text Processing with PHP - String manipulation functions
   - Regular Expressions in PHP - Pattern matching documentation

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

Version Control and Deployment

1. **GitHub Fundamentals**
   - GitHub Learning Lab - Interactive tutorials
   - Git Handbook - Core concepts
2. **Laravel Deployment**
   - Laravel Deployment Best Practices - Official guide
   - GitHub Actions for Laravel - Automation examples

Development Tools

1. **Free Code Editors**
   - Visual Studio Code - Popular code editor with extensions
   - VSCode Laravel Extensions - Laravel-specific tools
2. **Local Development**
   - XAMPP - Free local development stack
   - Laravel Sail Documentation - Docker development environment

Project Management

1. **Agile Development**
   - Trello - Free kanban board for task management
   - Agile for Students - Free course introduction
2. **Collaboration Tools**
   - GitHub Project Boards - Free project tracking
   - Discord - Free communication platform

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

Fallback Plans and Risk Mitigation

1. **Technical Challenges:**
   - Keep a running list of blockers in project management system
   - Create a knowledge sharing system for solutions to common problems
   - Establish a protocol for seeking external help
2. **Timeline Slippage:**
   - Identify MVP features vs. nice-to-have features
   - Prepare simplified versions of complex features
   - Have biweekly timeline reassessment meetings
3. **Integration Issues:**
   - Use contract-first API development
   - Create interface mocks for dependent services
   - Implement integration tests early
4. **Resource Limitations:**
   - Use serverless architecture when possible
   - Implement resource monitoring and alerts
   - Have scaling plans ready for high-demand periods

Post-Development Activities

1. **User Acceptance Testing**
   - Create test scripts for user evaluation
   - Implement feedback collection mechanism
   - Prioritize critical bug fixes
2. **Documentation Finalization**
   - Complete API documentation
   - Create user guides and tutorials
   - Document system architecture
3. **Knowledge Transfer**
   - Create handover documentation
   - Record walkthrough videos for complex features
   - Schedule knowledge transfer sessions
4. **Deployment and Monitoring**
   - Set up production monitoring
   - Implement automated backups
   - Create incident response plan

Detailed Tech Stack Configuration

This document provides in-depth information about the technology stack for the Intern-Employer Matchmaking Platform:

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

Frontend Technologies

Next.js Framework

- **Version**: 14.x (Latest stable)
- **Configuration**:
  - TypeScript support
  - Server-side rendering for critical pages
  - Static generation for content-heavy pages
  - API routes for backend communication
  - File-based routing
  - Environment variables management

UI Framework

- **Primary**: Tailwind CSS 3.x
  - Configuration: Custom theming with extended color palette
  - Plugins: Forms, Typography, Aspect Ratio, Line Clamp
  - Optimization: PurgeCSS for production builds
- **Component Framework**: Headless UI for accessible components
- **Icon Library**: Heroicons (free)

State Management

- **Local State**: React hooks (useState, useReducer)
- **Global State**: Context API for small-scale state
- **Complex State**: Redux Toolkit with RTK Query for data fetching

Form Handling

- **Form Library**: React Hook Form
- **Validation**: Yup schema validation
- **File Upload**: Custom hooks with progress tracking

Data Visualization

- **Charts**: Chart.js with React wrapper
- **Tabular Data**: React Table
- **Interactive Elements**: Framer Motion for animations

Development Tools

- **Linting**: ESLint with Airbnb config
- **Formatting**: Prettier

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- **Testing**: Jest + React Testing Library
- **Build Optimization**: Webpack Bundle Analyzer

Backend Technologies

Laravel Framework

- **Version**: 11.x (Latest stable)
- **Configuration**:
  - API mode configuration
  - CORS middleware setup
  - Rate limiting middleware
  - Custom validation rules
  - Resource-based controllers
  - API versioning

Authentication

- **System**: Laravel Sanctum for SPA authentication
- **Social Login**: Optional integration with Socialite
- **Password Management**: Custom password policies
- **Two-Factor Authentication**: Optional for enhanced security

Database Access

- **ORM**: Laravel Eloquent
- **Query Builder**: Laravel Query Builder for complex queries
- **Database Migrations**: Versioned migration system
- **Seeding**: Factory-based seeders for development data

API Design

- **Structure**: RESTful API design
- **Documentation**: OpenAPI/Swagger using l5-swagger
- **Responses**: API Resource transformations
- **Error Handling**: Consistent error response format

Background Processing

- **Queue System**: Laravel Queue with Database driver
- **Scheduling**: Laravel Task Scheduler for recurring tasks
- **Email Sending**: Queue-based email dispatch
- **Heavy Processing**: Chunked processing for data-intensive tasks

Email id:avani@zetheta.com
Contact number: +91-91362 49369

Development Tools

- **Linting**: PHP_CodeSniffer with PSR-12
- **Testing**: PHPUnit with parallel testing
- **API Testing**: Laravel HTTP Tests
- **Documentation**: PHPDoc standard

AI/ML Technologies (FastAPI)

FastAPI Framework

- **Version**: 0.104.x (Latest stable)
- **Configuration**:
  - ASGI server (Uvicorn)
  - API documentation with Swagger UI
  - Dependency injection system
  - Pydantic models for validation
  - Background tasks

NLP Tools

- **Core Library**: spaCy 3.x
  - Pre-trained English language model (medium size)
  - Custom named entity recognition for resume parsing
  - Entity rulers for skill extraction
  - Similarity matching with word vectors
- **Text Processing**: NLTK for supplementary functions
- **Vectorization**: Scikit-learn TF-IDF, Word2Vec

Machine Learning

- **Framework**: Scikit-learn
  - Pipelines for data processing
  - Model serialization with joblib
  - Cross-validation tools
  - Hyperparameter optimization
- **Algorithms**:
  - Classification: RandomForest, Gradient Boosting
  - Regression: Linear Regression, SVR
  - Clustering: K-Means, DBSCAN
  - Dimensionality Reduction: PCA, t-SNE

Email id:avani@zetheta.com
Contact number: +91-91362 49369

Deep Learning (Optional)

- **Framework**: PyTorch or TensorFlow Lite
- **Pre-trained Models**: HuggingFace Transformers
  - BERT for text classification
  - DistilBERT for faster inference
  - Sentence-BERT for embeddings

Vector Database

- **Primary Option**: Supabase or ChromaDB (open-source)
  - In-memory vector store for development
  - Persistent storage for production
  - Similarity search capabilities
- **Alternative**: FAISS (Facebook AI Similarity Search)

Development Tools

- **Linting**: Flake8, Black
- **Testing**: Pytest
- **Documentation**: Sphinx with Google docstring format

Database Technologies

Supabase PostgreSQL

Connection Method: REST API access via Supabase client
ORM: Laravel's Eloquent ORM for database interactions
Backup/Fallback: SQLite for local development and as a potential fallback if Supabase limits are reached
If you're hosting on InfinityFree or 000WebHost, they also offer MySQL databases as part of their free hosting package, which could be used as an additional fallback option if needed.

- **Extensions**:
  - pg_trgm for text similarity
  - pgvector for vector operations (if needed)
  - btree_gin for faster indexing
- **Configuration**:
  - Connection pooling
  - Optimized query planning
  - Regular vacuum scheduling
  - Appropriate indexing strategy

MongoDB

- **Version**: 7.x

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- **Collections Structure**:
  - Resumes collection
  - Job descriptions collection
  - Chat messages collection
  - User activities collection
- **Configuration**:
  - Document validation schemas
  - Compound indexes for common queries
  - TTL indexes for temporary data

Redis

- **Version**: 7.x
- **Use Cases**:
  - Session storage
  - Cache layer
  - Real-time features (pub/sub)
  - Rate limiting
  - Job queues
- **Configuration**:
  - Appropriate eviction policies
  - Data persistence settings
  - Memory optimization

DevOps & Infrastructure

Version Control

- **System**: Git
- **Hosting**: GitHub
- **Workflow**:
  - Feature branching
  - Pull request reviews
  - Semantic versioning
  - Automated changelog

CI/CD

- **Platform**: GitHub Actions
- **Pipelines**:

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Continuous Integration: Linting, testing, security scanning
- Continuous Deployment: Building, packaging, deployment
- Scheduled tasks: Dependency updates, security checks

Deployment Environments

- **Development**: Local Docker setup
- **Staging**: Vercel/Netlify for frontend, Render for backend
- **Production**: AWS Free Tier
  - Frontend: Vercel (free tier)
  - Backend: AWS Elastic Beanstalk or EC2 (free tier)
  - Database: AWS RDS free tier or MongoDB Atlas free tier

Containerization

- **Technology**: Docker
- **Services**:
  - Web service (Next.js)
  - API service (Laravel)
  - AI services (FastAPI)
  - Worker services (Queue processing)
- **Local Development**: Docker Compose

Monitoring & Logging

- **Logging**: ELK Stack (limited setup for essential logs)
  - Application logs
  - Error tracking
  - Performance metrics
- **Monitoring**: Prometheus + Grafana (basic setup)
  - System metrics
  - Application metrics
  - Endpoint health

Security Implementation

Authentication Security

- **Password Storage**: Bcrypt hashing with appropriate work factor
- **Token Management**: JWT with short expiration and refresh tokens
- **CSRF Protection**: Laravel built-in CSRF tokens
- **XSS Protection**: Content Security Policy, input sanitization
- **Rate Limiting**: IP-based and user-based rate limiting
- **Session Security**: Secure and HttpOnly cookies, session timeout

Email id:avani@zetheta.com
Contact number: +91-91362 49369

Data Security

- **Encryption**: AES-256 for sensitive data at rest
- **Transport Security**: TLS 1.3 for all communications
- **Input Validation**: Server-side validation for all inputs
- **Output Encoding**: Context-specific output encoding
- **SQL Injection Prevention**: Parameterized queries, ORM usage
- **File Upload Security**: Type validation, size limits, malware scanning

Infrastructure Security

- **Environment Isolation**: Separate development and production environments
- **Principle of Least Privilege**: Minimal permissions for service accounts
- **Secrets Management**: Environment variables, .env files (git-ignored)
- **Dependency Security**: Regular security audits, Dependabot alerts
- **Code Security**: Static analysis tools, secure coding practices
- **API Security**: API keys for service-to-service communication, throttling

Third-Party Integrations

Email Service

- **Provider**: SendGrid (free tier)
- **Features**:
    - Transactional email templates
    - Email verification
    - Delivery tracking
    - Event webhooks
- **Implementation**: Laravel mail driver with queue

File Storage

- **Provider**: AWS S3 (free tier) or local storage for development
- **Features**:
    - Document storage for resumes
    - Image storage for user avatars and company logos
    - Temporary storage for processing
- **Implementation**: Laravel Filesystem with driver abstraction

Analytics

- **Provider**: Matomo (self-hosted, open-source)
- **Features**:
    - User behavior tracking

Email id:avani@zetheta.com
Contact number: +91-91362 49369

    - Conversion tracking
    - Event tracking
    - Privacy-focused analytics
- **Implementation**: Client-side JavaScript with consent management

Social Login

- **Providers**: Google, LinkedIn
- **Implementation**: Laravel Socialite package
- **Data Storage**: Minimal data collection for authentication only

Development Environment Setup

Local Development

- **Prerequisites**:
    - Docker and Docker Compose
    - Git
    - Node.js and npm/yarn
    - Composer
    - Python 3.9+
- **Setup Process**:
    - Repository cloning
    - Environment configuration
    - Docker container initialization
    - Database seeding
    - Development server startup

Development Tools

- **IDE Recommendations**: VS Code with extension pack
    - ESLint/Prettier integration
    - PHP CodeSniffer integration
    - Docker extension
    - Database extension
- **API Testing**: Postman/Insomnia collections
- **Database Management**: TablePlus/DBeaver

Documentation System

- **Code Documentation**:
    - PHP: PHPDoc
    - JavaScript: JSDoc
    - Python: Google-style docstrings

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- **API Documentation**: OpenAPI/Swagger
- **User Documentation**: Markdown-based wiki
- **Architecture Documentation**: C4 model diagrams

Performance Optimization

Frontend Performance

- **Loading Optimization**:
  - Code splitting
  - Tree shaking
  - Lazy loading
  - Resource prioritization
- **Rendering Optimization**:
  - Component memoization
  - Virtual list for large datasets
  - Windowing techniques
  - Skeleton loaders

Backend Performance

- **Database Optimization**:
  - Efficient indexing
  - Query optimization
  - Connection pooling
  - Eager loading for relationships
- **API Response Optimization**:
  - Response compression
  - JSON serialization optimization
  - Caching strategies
  - Resource pagination

AI Service Performance

- **Model Optimization**:
  - Model quantization
  - Batched prediction
  - Asynchronous processing
  - Caching prediction results
- **Inference Optimization**:
  - Pre-computed embeddings
  - Approximate nearest neighbor search
  - Feature selection optimization
  - Parallelized processing

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

Testing Strategy

Unit Testing

- **Frontend**: Jest with React Testing Library
- **Backend**: PHPUnit for Laravel
- **AI Services**: Pytest for FastAPI

Integration Testing

- **API Testing**: Laravel HTTP Tests
- **Service Communication**: Mock service integration
- **Database Integration**: Test database with transactions

End-to-End Testing

- **Technology**: Cypress for critical user journeys
- **Coverage**:
    - Authentication flows
    - Profile creation
    - Job application process
    - Messaging system
    - Search functionality

Performance Testing

- **Load Testing**: Artillery.io for simulated load
- **Endpoint Performance**: Response time benchmarking
- **Database Performance**: Query execution plan analysis

Deployment Strategy

Staging Deployment

- **Process**:
    - Automated builds from staging branch
    - Database migration automation
    - Test data seeding
    - Integration testing
- **Environment**: Vercel preview deployments, Render free tier

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

Production Deployment

- **Process**:
  - Automated builds from main branch
  - Zero-downtime deployment
  - Database backup before migration
  - Phased rollout strategy
- **Environment**: Vercel for frontend, AWS free tier for backend

Post-Deployment

- **Verification**:
  - Smoke tests
  - Health check monitoring
  - Error rate monitoring
  - Performance baseline comparison
- **Rollback Plan**:
  - Automated rollback triggers
  - Database restore procedure
  - Traffic routing fallback

Maintenance and Support

Monitoring Plan

- **System Health**:
  - Service uptime monitoring
  - Resource utilization tracking
  - Error rate monitoring
  - Response time tracking
- **Application Health**:
  - User session monitoring
  - Conversion funnel tracking
  - Feature usage analytics
  - AI model performance monitoring

Update Strategy

- **Dependency Updates**:
  - Scheduled security updates
  - Major version migration planning
  - Compatibility testing process
- **Feature Updates**:
  - Feature flagging system

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- A/B testing framework
- Phased rollout strategy

Backup Strategy

- **Database Backups**:
  - Daily automated backups
  - Point-in-time recovery capability
  - Backup verification process
- **File Backups**:
  - Incremental backup system
  - Cross-region redundancy (when possible)
  - Restoration testing procedure

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

FRONT END SCREEN SUGGESTIONS (TO BE MODIFIED)

A sample list of all UI screens the frontend team needs to develop for this project, organized by user type and functionality is as follows:

Authentication & Onboarding Screens

1. **Landing Page** - Platform introduction with login/signup options
2. **Login Screen** - Email/password login with social login options
3. **Registration Selection** - Choose between Student/Employer registration
4. **Student Registration Flow**:
   - Basic information (name, email, password)
   - Education details
   - Skills assessment
   - Resume upload
   - Profile completion progress indicator
5. **Employer Registration Flow**:
   - Company information
   - Industry selection
   - Team details
   - Job posting preferences

Student-Specific Screens

6. **Student Dashboard** - Overview with readiness score, recommended jobs, application status
7. **Profile Management**:
   - Personal information editor
   - Education history manager
   - Skills management interface
   - Projects showcase editor
   - Resume management
8. **Job Discovery**:
   - Job search with filters (location, industry, skills)
   - Job listing cards with match scores
   - Saved jobs view
   - Job recommendations
9. **Job Detail Screen** - Complete job information with apply option
10. **Application Management**:
    - Application form
    - Application status tracker
    - Application history
11. **Career Development**:
    - Readiness score dashboard
    - Readiness score detailed report
    - Skill gap analysis dashboard

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

   - Skill gap detailed report
   - Learning resources recommendations
12. **PDF Reports Viewer** - For viewing generated reports

Employer-Specific Screens

13. **Employer Dashboard** - Overview with active postings, applicant metrics
14. **Company Profile Management**:
   - Company information editor
   - Team profile management
   - Brand customization
15. **Job Posting Management**:
   - Job creation form
   - Job listing editor
   - Job post analytics
16. **Candidate Discovery**:
   - Candidate search with filters
   - Candidate cards with match scores
   - Saved candidates list
17. **Application Review**:
   - Application inbox
   - Candidate profile viewer
   - Resume viewer
   - Application rating interface
18. **Interview Management**:
   - Interview scheduling
   - Interview feedback form

Admin Screens

19. **Admin Dashboard** - Platform metrics, user statistics
20. **User Management** - List, edit, suspend users
21. **Content Moderation** - Review and approve job postings
22. **Analytics Dashboard** - Platform performance metrics

Common/Shared Screens

23. **Messaging Interface** - Simple messaging between students and employers
24. **Notification Center** - Alert management for all users
25. **Settings Page** - Account settings, notification preferences, etc.
26. **Help & Support** - FAQ, support tickets, documentation

Report/PDF Screens

Email id:avani@zetheta.com
Contact number: +91-91362 49369

27. **Readiness Score Report Template** - Design for PDF generation
28. **Skill Gap Analysis Report Template** - Design for PDF generation
29. **Career Path Prediction Report Template** - Design for PDF generation

Mobile Responsive Versions

30. **Mobile Dashboard** - Simplified mobile view of dashboard
31. **Mobile Job Browsing** - Optimized job listings for mobile
32. **Mobile Application Process** - Streamlined application flow for mobile

Development Priority Order

FIRST

- Authentication screens
- Basic dashboard views
- Profile management
- Job posting and discovery
- Application processes
- Report generation screens

SECOND

- Messaging
- Analytics
- Advanced filtering
- Recommendation interfaces

THIRD

- Admin functionality
- Advanced customization options
- Social features

Each screen should follow the design system guidelines with a focus on the Tailwind CSS implementation outlined in the scope document.

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

THE DESIGN SYSTEM GUIDELINES ARE SAMPLED BELOW

This is just a sample and multiple changes will be done during the course of this project:

1. Color Palette

Primary Colors

- **Primary Blue** (#3B82F6): Main brand color, used for primary buttons, links, and key UI elements
- **Primary Blue Light** (#93C5FD): Secondary actions, highlights, backgrounds
- **Primary Blue Dark** (#1E40AF): Hover states, text on light backgrounds

Secondary Colors

- **Accent Teal** (#0D9488): Used for highlighting important information, success states
- **Accent Purple** (#8B5CF6): Used for select UI elements to provide visual interest

Neutral Colors

- **Gray 900** (#111827): Primary text
- **Gray 700** (#374151): Secondary text
- **Gray 500** (#6B7280): Disabled text, icons
- **Gray 300** (#D1D5DB): Borders, dividers
- **Gray 100** (#F3F4F6): Backgrounds, disabled states
- **White** (#FFFFFF): Card backgrounds, primary background

Semantic Colors

- **Success** (#10B981): Success states, positive actions
- **Warning** (#F59E0B): Warning states, cautionary actions
- **Error** (#EF4444): Error states, destructive actions
- **Info** (#3B82F6): Informational states, uses primary blue

2. Typography

Font Families

- **Primary Font**: Inter (sans-serif)
- **Monospace Font**: JetBrains Mono (for code elements)

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

Font Sizes

- **xs**: 0.75rem (12px)
- **sm**: 0.875rem (14px)
- **base**: 1rem (16px)
- **lg**: 1.125rem (18px)
- **xl**: 1.25rem (20px)
- **2xl**: 1.5rem (24px)
- **3xl**: 1.875rem (30px)
- **4xl**: 2.25rem (36px)

Font Weights

- **Regular**: 400
- **Medium**: 500
- **Semibold**: 600
- **Bold**: 700

Line Heights

- **Tight**: 1.25
- **Normal**: 1.5
- **Relaxed**: 1.75

Text Styles

- **Heading 1**: 2.25rem (36px), 700 weight, 1.25 line height
- **Heading 2**: 1.875rem (30px), 700 weight, 1.25 line height
- **Heading 3**: 1.5rem (24px), 600 weight, 1.25 line height
- **Heading 4**: 1.25rem (20px), 600 weight, 1.5 line height
- **Body Large**: 1.125rem (18px), 400 weight, 1.5 line height
- **Body**: 1rem (16px), 400 weight, 1.5 line height
- **Body Small**: 0.875rem (14px), 400 weight, 1.5 line height
- **Caption**: 0.75rem (12px), 400 weight, 1.5 line height

3. Spacing System

Using a 4px baseline grid:

- **0**: 0px
- **1**: 0.25rem (4px)
- **2**: 0.5rem (8px)
- **3**: 0.75rem (12px)
- **4**: 1rem (16px)

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- **5**: 1.25rem (20px)
- **6**: 1.5rem (24px)
- **8**: 2rem (32px)
- **10**: 2.5rem (40px)
- **12**: 3rem (48px)
- **16**: 4rem (64px)
- **20**: 5rem (80px)
- **24**: 6rem (96px)

4. Border Radius

- **None**: 0px
- **Small**: 0.125rem (2px)
- **Default**: 0.25rem (4px)
- **Medium**: 0.375rem (6px)
- **Large**: 0.5rem (8px)
- **XL**: 0.75rem (12px)
- **2XL**: 1rem (16px)
- **Full**: 9999px (for circular elements)

5. Shadows

- **None**: none
- **Small**: 0 1px 2px 0 rgba(0, 0, 0, 0.05)
- **Default**: 0 1px 3px 0 rgba(0, 0, 0, 0.1), 0 1px 2px 0 rgba(0, 0, 0, 0.06)
- **Medium**: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0, 0, 0, 0.06)
- **Large**: 0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0, 0, 0, 0.05)
- **XL**: 0 20px 25px -5px rgba(0, 0, 0, 0.1), 0 10px 10px -5px rgba(0, 0, 0, 0.04)

6. Component-Specific Guidelines

Buttons

Primary Button

- Background: Primary Blue
- Text: White
- Hover: Primary Blue Dark
- Padding: 0.5rem 1rem (py-2 px-4)
- Border Radius: Default (0.25rem)
- Font Weight: Medium (500)

Secondary Button

- Background: White

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Text: Primary Blue
- Border: 1px solid Primary Blue
- Hover Background: Primary Blue Light (10% opacity)
- Padding: 0.5rem 1rem (py-2 px-4)
- Border Radius: Default (0.25rem)
- Font Weight: Medium (500)

Tertiary Button

- Background: Transparent
- Text: Gray 700
- Hover: Gray 100
- Padding: 0.5rem 1rem (py-2 px-4)
- Border Radius: Default (0.25rem)
- Font Weight: Medium (500)

Danger Button

- Background: Error
- Text: White
- Hover: Darker Error
- Padding: 0.5rem 1rem (py-2 px-4)
- Border Radius: Default (0.25rem)
- Font Weight: Medium (500)

Button Sizes

- **Small**: py-1 px-2, text-sm
- **Default**: py-2 px-4, text-base
- **Large**: py-3 px-6, text-lg

Form Elements

Input Fields

- Height: 2.5rem (40px)
- Border: 1px solid Gray 300
- Border Radius: Default (0.25rem)
- Padding: 0.5rem 0.75rem (py-2 px-3)
- Focus: Border Primary Blue, Ring 2px Primary Blue (10% opacity)
- Error: Border Error, Ring 2px Error (10% opacity)

Email id:avani@zetheta.com
Contact number: +91-91362 49369

Select Inputs

- Similar to input fields
- Add dropdown icon
- Padding-right increased to accommodate icon

Checkboxes and Radio Buttons

- Border: 1px solid Gray 300
- Checked State: Primary Blue background
- Focus: Ring 2px Primary Blue (10% opacity)

Labels

- Font Size: text-sm (14px)
- Font Weight: Medium (500)
- Color: Gray 700
- Margin Bottom: 0.25rem (mb-1)

Cards

Standard Card

- Background: White
- Border: 1px solid Gray 300
- Border Radius: Large (0.5rem)
- Padding: 1rem (p-4)
- Shadow: Default

Interactive Card

- Standard card properties
- Hover: Shadow Medium
- Transition: 150ms for shadow

Highlighted Card

- Standard card properties
- Border Left: 4px solid Primary Blue
- or
- Border Top: 4px solid Primary Blue

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

Navigation

Main Navigation

- Background: White
- Active Item: Text Primary Blue
- Inactive Item: Text Gray 700
- Hover: Text Primary Blue, Background Gray 100
- Padding: 0.5rem 1rem (py-2 px-4)
- Font Weight: Medium (500)

Sidebar Navigation

- Background: Gray 50 or White
- Active Item: Background Primary Blue Light, Text Primary Blue
- Inactive Item: Text Gray 700
- Hover: Background Gray 100
- Padding: 0.75rem 1rem (py-3 px-4)
- Border Radius: Default (0.25rem)
- Font Weight: Medium (500)

Alerts and Notifications

Info Alert

- Background: Primary Blue Light (10% opacity)
- Border Left: 4px solid Primary Blue
- Text: Gray 900
- Icon: Primary Blue

Success Alert

- Background: Success (10% opacity)
- Border Left: 4px solid Success
- Text: Gray 900
- Icon: Success

Warning Alert

- Background: Warning (10% opacity)
- Border Left: 4px solid Warning
- Text: Gray 900
- Icon: Warning

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

Error Alert

- Background: Error (10% opacity)
- Border Left: 4px solid Error
- Text: Gray 900
- Icon: Error

Data Visualization

Match Score Indicator

- High Match (80%+): Success color
- Good Match (60-79%): Primary Blue
- Medium Match (40-59%): Warning
- Low Match (<40%): Gray 500

Progress Bars

- Height: 0.5rem (8px)
- Border Radius: Full
- Background: Gray 200
- Progress: Primary color based on context

7. Responsive Breakpoints

- **SM**: 640px (Phones)
- **MD**: 768px (Large phones, tablets)
- **LG**: 1024px (Tablets, laptops)
- **XL**: 1280px (Desktops)
- **2XL**: 1536px (Large desktops)

8. Icons

- Use Heroicons (https://heroicons.com/) for consistency
- Icon sizes:
    - **Small**: 16px × 16px
    - **Medium**: 20px × 20px
    - **Large**: 24px × 24px

9. Animation Guidelines

- Use short, subtle animations
- Default Transition: 150ms ease-in-out
- Hover transitions: 100ms ease-in-out

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

- Page transitions: 300ms ease-in-out
- Use CSS transitions for simple animations
- Use Framer Motion for more complex animations

10. Accessibility Guidelines

- Maintain minimum contrast ratio of 4.5:1 for normal text
- Maintain minimum contrast ratio of 3:1 for large text
- Include focus states for all interactive elements
- Ensure all form elements have associated labels
- Use semantic HTML elements
- Include alt text for all images
- Support keyboard navigation

11. Implementation with Tailwind CSS

All design tokens should be configured in the Tailwind configuration file:

```
// tailwind.config.js
module.exports = {
  theme: {
    extend: {
      colors: {
        primary: {
          DEFAULT: '#3B82F6',
          light: '#93C5FD',
          dark: '#1E40AF',
        },
        accent: {
          teal: '#0D9488',
          purple: '#8B5CF6',
        },
        success: '#10B981',
        warning: '#F59E0B',
        error: '#EF4444',
        info: '#3B82F6',
      },
      fontFamily: {
        sans: ['Inter', 'sans-serif'],
        mono: ['JetBrains Mono', 'monospace'],
      },
      boxShadow: {
        DEFAULT: '0 1px 3px 0 rgba(0, 0, 0, 0.1), 0 1px 2px 0 rgba(0, 0, 0, 0.06)',
        medium: '0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0, 0, 0, 0.06)',
        large: '0 10px 15px -3px rgba(0, 0, 0, 0.1), 0 4px 6px -2px rgba(0, 0, 0, 0.05)',
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
      xl: '0 20px 25px -5px rgba(0, 0, 0, 0.1), 0 10px 10px -5px rgba(0, 0, 0, 0.04)',
    },
  },
},
};
```

12. Component Library

Create a component library documentation with examples of all common components:

- Buttons (all variations)
- Form inputs
- Cards
- Navigation elements
- Alerts
- Tables
- Modals
- Dropdowns
- Badges
- Avatars

This design system provides a comprehensive foundation for creating a consistent, accessible, and visually appealing interface for the Intern-Employer Matchmaking Platform. The frontend team should follow these guidelines to ensure cohesive implementation across all screens and components.

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

BACKEND TEAM IMPLEMENTATION GUIDE

Core Backend Components

1. API Endpoints Structure

```
/api
├── /auth              # Authentication endpoints
├── /users             # User management
├── /students          # Student-specific endpoints
├── /employers         # Employer-specific endpoints
├── /jobs              # Job posting endpoints
├── /applications      # Job application endpoints
├── /skills            # Skills management
├── /reports           # Report generation
└── /analytics         # Analytics endpoints
```

2. Key API Endpoints to Implement

Authentication

- POST /api/auth/register - User registration
- POST /api/auth/login - User login
- POST /api/auth/logout - User logout
- POST /api/auth/reset-password - Password reset request
- PUT /api/auth/reset-password/{token} - Password reset confirmation

User Management

- GET /api/users/me - Get current user
- PUT /api/users/me - Update user profile
- GET /api/users/{id} - Get user by ID (admin only)
- PUT /api/users/{id}/status - Update user status (admin only)

Students

- GET /api/students/{id} - Get student profile
- PUT /api/students/{id} - Update student profile
- POST /api/students/{id}/resume - Upload resume
- GET /api/students/{id}/readiness-score - Get readiness score
- GET /api/students/{id}/skill-gap - Get skill gap analysis
- GET /api/students/{id}/career-paths - Get career path predictions
- GET /api/students/{id}/recommended-jobs - Get job recommendations

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

Employers

- GET /api/employers/{id} - Get employer profile
- PUT /api/employers/{id} - Update employer profile
- GET /api/employers/{id}/job-postings - Get employer's job postings
- GET /api/employers/{id}/applications - Get applications to employer's jobs

Jobs

- GET /api/jobs - List job postings (with filtering)
- GET /api/jobs/{id} - Get job posting details
- POST /api/jobs - Create job posting
- PUT /api/jobs/{id} - Update job posting
- DELETE /api/jobs/{id} - Delete job posting
- GET /api/jobs/{id}/applicants - Get applicants for job

Applications

- POST /api/applications - Submit job application
- GET /api/applications/{id} - Get application details
- PUT /api/applications/{id}/status - Update application status
- GET /api/applications/{id}/matches - Get match details for application

Skills

- GET /api/skills - List all skills
- GET /api/skills/trending - Get trending skills
- POST /api/skills - Add new skill (admin only)

Reports

- GET /api/reports/readiness-score/{student_id} - Generate readiness score report
- GET /api/reports/skill-gap/{student_id} - Generate skill gap analysis report
- GET /api/reports/career-path/{student_id} - Generate career path report

3. Laravel Service Structure

```
app/Services/
├── AuthService.php            # Authentication logic
├── UserService.php            # User management
├── StudentService.php         # Student profile management
├── EmployerService.php        # Employer profile management
├── JobService.php             # Job posting management
├── ApplicationService.php     # Application processing
├── MatchingService.php        # Matching algorithm
```

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

```
├── ResumeParserService.php      # Resume parsing
├── ReadinessScoreService.php    # Readiness score calculation
├── SkillGapService.php          # Skill gap analysis
├── CareerPathService.php        # Career path prediction
├── RecommendationService.php    # Job recommendation
├── ReportService.php            # PDF report generation
└── NotificationService.php      # Notification handling
```

4. Controller Structure

```
app/Http/Controllers/
├── AuthController.php           # Authentication endpoints
├── UserController.php           # User management
├── StudentController.php        # Student endpoints
├── EmployerController.php       # Employer endpoints
├── JobController.php            # Job posting endpoints
├── ApplicationController.php    # Application endpoints
├── SkillController.php          # Skills endpoints
├── ReportController.php         # Report generation
└── AdminController.php          # Admin-only endpoints
```

5. Middleware Requirements

- Authenticate - Verify user is logged in
- CheckUserType - Verify user type (student/employer/admin)
- EnsureProfileComplete - Verify profile completion
- RateLimiting - Prevent API abuse
- HandleCors - Handle cross-origin requests

6. Important Libraries to Install

```
# Core functionality
composer require laravel/sanctum     # API authentication
composer require spatie/laravel-permission  # Role/permission management
composer require barryvdh/laravel-dompdf    # PDF generation

# Optional but recommended
composer require intervention/image   # Image processing
composer require maatwebsite/excel    # Excel/CSV processing
composer require spatie/laravel-backup  # Automated backups
```

7. API Response Format Standard

```
{
  "success": true,
  "message": "Operation successful",
  "data": { ... },
  "meta": {
    "pagination": {
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    "total": 100,
    "per_page": 15,
    "current_page": 1,
    "last_page": 7,
    "from": 1,
    "to": 15
   }
  }
}
```


8. Error Handling Standard


```
{
  "success": false,
  "message": "An error occurred",
  "errors": {
    "field_name": ["Error message"]
  },
  "code": "ERROR_CODE"
}
```


9. Backend Implementation Priorities

1. **FIRST**:
   - User authentication system
   - User profile management
   - Database migrations
   - Basic API endpoints structure
2. **SECOND**:
   - Job posting/listing functionality
   - Student/employer profile management
   - File upload handling (resumes)
   - Basic matching algorithm
3. **THIRD**:
   - Readiness score calculation
   - Skill gap analysis
   - PDF report generation
   - Recommendation service
4. **FOURTH**:
   - API optimization and security
   - Testing and debugging
   - Documentation
   - Deployment preparation

Database Team Implementation Guide

1. Database Schema and Relationships

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Core Tables

Users Table

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,
  first_name VARCHAR(100),
  last_name VARCHAR(100),
  phone VARCHAR(50),
  user_type ENUM('student', 'employer', 'admin') NOT NULL,
  status ENUM('active', 'inactive', 'suspended', 'pending') DEFAULT 'active',
  email_verified BOOLEAN DEFAULT FALSE,
  profile_image_url TEXT,
  reset_token VARCHAR(100),
  reset_token_expiry TIMESTAMP,
  remember_token VARCHAR(100),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Students Table

```
CREATE TABLE students (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  university VARCHAR(255),
  degree VARCHAR(100),
  major VARCHAR(100),
  graduation_year INTEGER,
  bio TEXT,
  gpa FLOAT,
  preferences JSONB DEFAULT '{}',
  career_objectives JSONB DEFAULT '{}',
  readiness_score INTEGER,
  last_active TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Employers Table

```
CREATE TABLE employers (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  company_name VARCHAR(255) NOT NULL,
  industry VARCHAR(100),
  company_size VARCHAR(50),
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
  company_description TEXT,
  website VARCHAR(255),
  logo_url TEXT,
  company_culture JSONB DEFAULT '{}',
  tech_stack JSONB DEFAULT '[]',
  verified BOOLEAN DEFAULT FALSE,
  last_active TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Job Postings Table

```
CREATE TABLE job_postings (
  id SERIAL PRIMARY KEY,
  employer_id INTEGER NOT NULL REFERENCES employers(id) ON DELETE CASCADE,
  title VARCHAR(255) NOT NULL,
  description TEXT NOT NULL,
  location VARCHAR(255),
  job_type VARCHAR(50),
  start_date DATE,
  end_date DATE,
  salary_range VARCHAR(100),
  experience_level VARCHAR(50),
  remote_allowed BOOLEAN DEFAULT FALSE,
  status VARCHAR(20) DEFAULT 'active',
  views_count INTEGER DEFAULT 0,
  applications_count INTEGER DEFAULT 0,
  positions_available INTEGER DEFAULT 1,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Relationship Tables

Applications Table

```
CREATE TABLE applications (
  id SERIAL PRIMARY KEY,
  student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE CASCADE,
  job_posting_id INTEGER NOT NULL REFERENCES job_postings(id) ON DELETE CASCADE,
  applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  status VARCHAR(50) DEFAULT 'pending',
  cover_letter TEXT,
  matching_score FLOAT,
  employer_notes TEXT,
  last_status_change TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(student_id, job_posting_id)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
);
```

Skills Table

```
CREATE TABLE skills (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL UNIQUE,
  category VARCHAR(50),
  description TEXT
);
```

Student Skills Table

```
CREATE TABLE student_skills (
  id SERIAL PRIMARY KEY,
  student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE CASCADE,
  skill_id INTEGER NOT NULL REFERENCES skills(id) ON DELETE CASCADE,
  proficiency_level INTEGER CHECK (proficiency_level BETWEEN 1 AND 10),
  is_verified BOOLEAN DEFAULT FALSE,
  UNIQUE(student_id, skill_id)
);
```

Job Skills Table

```
CREATE TABLE job_skills (
  id SERIAL PRIMARY KEY,
  job_posting_id INTEGER NOT NULL REFERENCES job_postings(id) ON DELETE CASCADE,
  skill_id INTEGER NOT NULL REFERENCES skills(id) ON DELETE CASCADE,
  type VARCHAR(20) CHECK (type IN ('required', 'preferred')),
  UNIQUE(job_posting_id, skill_id, type)
);
```

Education Table

```
CREATE TABLE education (
  id SERIAL PRIMARY KEY,
  student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE CASCADE,
  institution VARCHAR(255) NOT NULL,
  degree VARCHAR(100),
  field_of_study VARCHAR(100),
  start_date DATE,
  end_date DATE,
  gpa FLOAT,
  description TEXT,
  is_current BOOLEAN DEFAULT FALSE
);
```

Experience Table

```
CREATE TABLE experience (
  id SERIAL PRIMARY KEY,
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
  student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE CASCADE,
  company VARCHAR(255) NOT NULL,
  position VARCHAR(100) NOT NULL,
  start_date DATE NOT NULL,
  end_date DATE,
  description TEXT,
  location VARCHAR(100),
  is_current BOOLEAN DEFAULT FALSE
);
```

Projects Table

```
CREATE TABLE projects (
  id SERIAL PRIMARY KEY,
  student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE CASCADE,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  url VARCHAR(255),
  technologies JSONB DEFAULT '[]',
  start_date DATE,
  end_date DATE,
  image_url VARCHAR(255)
);
```

Reports Table

```
CREATE TABLE reports (
  id SERIAL PRIMARY KEY,
  student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE CASCADE,
  report_type VARCHAR(50) NOT NULL,
  file_url VARCHAR(255) NOT NULL,
  generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  expires_at TIMESTAMP
);
```

2. Data Migration Strategy

1. **Database Setup**:
    - Create initial database schema using Laravel migrations
    - Set up permissions and roles
2. **Seed Data**:
    - Create database seeders for:
        - Skills taxonomy
        - Industries list
        - Educational institutions
        - Sample users for testing
        - Sample job postings
3. **Data Validation**:

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Implement validators for all input data
- Create data consistency checks

3. Database Performance Optimization

1. **Indexing Strategy**:
   - Create indexes on frequently queried columns:
   - -- Frequently filtered or joined columnsCREATE INDEX idx_users_email ON users(email);CREATE INDEX idx_users_user_type ON users(user_type);CREATE INDEX idx_students_user_id ON students(user_id);CREATE INDEX idx_employers_user_id ON employers(user_id);CREATE INDEX idx_job_postings_employer_id ON job_postings(employer_id);CREATE INDEX idx_job_postings_status ON job_postings(status);CREATE INDEX idx_applications_student_id ON applications(student_id);CREATE INDEX idx_applications_job_posting_id ON applications(job_posting_id);CREATE INDEX idx_student_skills_student_id ON student_skills(student_id);CREATE INDEX idx_student_skills_skill_id ON student_skills(skill_id);CREATE INDEX idx_job_skills_job_posting_id ON job_skills(job_posting_id);CREATE INDEX idx_job_skills_skill_id ON job_skills(skill_id);
2. **Query Optimization**:
   - Use eager loading for related entities
   - Implement pagination for list endpoints
   - Use database-level filtering where possible
   - Consider caching frequent queries
3. **Supabase Specific Optimizations**:
   - Implement proper RLS (Row Level Security) policies
   - Use pgvector for any AI feature calculations
   - Optimize JSON/JSONB queries for preferences and settings

4. Data Backup Strategy

1. **Regular Backups**:
   - Daily database backups
   - Store backups in a separate location (different Supabase project)
   - Implement backup rotation policy
2. **Disaster Recovery**:
   - Document step-by-step recovery process
   - Test recovery process periodically
   - Maintain backup of schema separately from data

5. Data Security Measures

1. **Data Protection**:
   - Implement proper data encryption (password hashing)
   - Secure sensitive data in JSONB fields

Email id:avani@zetheta.com
Contact number: +91-91362 49369

- Use parameterized queries to prevent SQL injection

2. **Access Control**:
   - Implement Row Level Security policies in Supabase
   - Create appropriate database roles
   - Limit direct database access

Deployment Team Implementation Guide

1. Deployment Architecture

- Frontend (Vercel/Netlify)
- Backend API (InfinityFree)
- Database (Supabase)
- Static Assets (GitHub Pages)
- File Storage (InfinityFree)
- PDF Reports (Storj Storage)

2. Environment Configuration

Production Environment Variables

Frontend (.env.production)

```
NEXT_PUBLIC_API_URL=https://your-backend.infinityfree.com/api
NEXT_PUBLIC_SUPABASE_URL=https://your-project.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=your-anon-key
```

Backend (.env)

```
APP_NAME=InternEmployerMatchmaking
APP_ENV=production
APP_KEY=base64:your-app-key
APP_DEBUG=false
APP_URL=https://your-backend.infinityfree.com

DB_CONNECTION=pgsql
DB_HOST=db.your-project.supabase.co
DB_PORT=5432
DB_DATABASE=postgres
DB_USERNAME=postgres
DB_PASSWORD=your-db-password

QUEUE_CONNECTION=database
SESSION_DRIVER=cookie
CACHE_DRIVER=file
```

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailgun.org
MAIL_PORT=587
MAIL_USERNAME=your-username
MAIL_PASSWORD=your-password
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=noreply@yoursite.com
MAIL_FROM_NAME="${APP_NAME}"

REPORT_STORAGE_PATH=public/reports
```

3. Deployment Steps

Backend Deployment (Laravel on InfinityFree)

1. **Prepare Laravel for Production**

```
# Install dependencies without dev packages
composer install --no-dev --optimize-autoloader

# Generate key
php artisan key:generate

# Cache configuration
php artisan config:cache
php artisan route:cache
php artisan view:cache
```

2. **Configure .htaccess**

```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteRule ^(.*)$ public/$1 [L]
</IfModule>
```

3. **FTP Upload**
   - Upload prepared files to InfinityFree hosting
   - Set appropriate file permissions (755 for directories, 644 for files)
   - Create writable directories for storage, cache, and logs
4. **Database Setup**
   - Run migrations and seeders via CLI or phpMyAdmin
   - Verify database connection

Frontend Deployment (Next.js on Vercel)

1. **Build Frontend**

npm run build

2. **Configure vercel.json**

```
{
  "version": 2,
  "builds": [
    {
      "src": "package.json",
      "use": "@vercel/next"
    }
  ],
  "routes": [
    {
      "src": "/api/(.*)",
      "dest": "/api/$1"
    },
    {
      "src": "/(.*)",
      "dest": "/$1"
    }
  ]
}
```

3. **Deploy to Vercel**

vercel --prod

4. Continuous Integration/Deployment

GitHub Actions Workflow (.github/workflows/deploy.yml)

```
name: Deploy Intern-Employer Platform

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test-backend:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    runs-on: ubuntu-latest
    # Backend testing configuration...

  test-frontend:
    runs-on: ubuntu-latest
    # Frontend testing configuration...

  deploy-backend:
    needs: [test-backend, test-frontend]
    if: github.ref == 'refs/heads/main' && github.event_name == 'push'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up PHP
        uses: shivammathur/setup-php@v2
        with:
          php-version: '8.1'
      - name: Install dependencies
        run: |
          cd backend
          composer install --no-dev --optimize-autoloader
      - name: Prepare for production
        run: |
          cd backend
          cp .env.example .env
          php artisan key:generate
          php artisan config:cache
          php artisan route:cache
          php artisan view:cache
      - name: Deploy to InfinityFree
        uses: SamKirkland/FTP-Deploy-Action@4.3.0
        with:
          server: ftpupload.net
          username: ${{ secrets.FTP_USERNAME }}
          password: ${{ secrets.FTP_PASSWORD }}
          local-dir: ./backend/
          server-dir: /htdocs/

  deploy-frontend:
    needs: [test-frontend]
    if: github.ref == 'refs/heads/main' && github.event_name == 'push'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Node.js
        uses: actions/setup-node@v3
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
      with:
        node-version: '16'
    - name: Install Vercel CLI
      run: npm install -g vercel
    - name: Deploy to Vercel
      run: |
        cd frontend
        vercel deploy --prod --token ${{ secrets.VERCEL_TOKEN }}
```

5. Monitoring and Maintenance

Monitoring Setup

1. **Performance Monitoring**
   - Set up free tier New Relic or Sentry for backend monitoring
   - Implement basic logging for critical operations
   - Set up Vercel Analytics for frontend monitoring
2. **Error Tracking**
   - Implement Sentry for error reporting
   - Configure email notifications for critical errors
3. **Uptime Monitoring**
   - Use UptimeRobot (free tier) to monitor API endpoints
   - Set up alerts for downtime

Backup Strategy

1. **Database Backups**
   - Schedule daily backups from Supabase
   - Implement retention policy (keep 7 daily, 4 weekly, 3 monthly backups)
2. **Code Backups**
   - Maintain complete GitHub repository backups
   - Document configuration settings separately

Maintenance Procedures

1. **Regular Updates**
   - Schedule monthly dependency updates
   - Security patch process
2. **Scaling Considerations**
   - Document upgrade path to paid hosting when traffic increases
   - Identify bottlenecks and optimization opportunities

6. Fallback Procedures

Email id:avani@zetheta.com
Contact number: +91-91362 49369

1. **Database Fallback**
    - Implement SQLite fallback in case Supabase limits are reached
    - Document database restoration process
2. **API Fallback**
    - Implement graceful degradation for critical features
    - Create static fallback pages for key functionality
3. **Recovery Plan**
    - Document complete disaster recovery procedure
    - Test recovery process before launch

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

CORE IMPLEMENTATION CODE SUGGESTIONS (TO BE MODIFIED)

```
# This file contains the core implementation of AI components for the Intern-Employer Matchmaking Platform
# Each component is separated into its own class with documented methods
import pandas as pd
import numpy as np
from typing import Dict, List, Tuple, Optional, Union, Any
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.ensemble import RandomForestRegressor, GradientBoostingClassifier
from sklearn.cluster import KMeans
import spacy
import re
from datetime import datetime
import joblib
import os
from fastapi import FastAPI, HTTPException, File, UploadFile, Form, Depends, BackgroundTasks
from pydantic import BaseModel, Field

# Load NLP models
try:
    nlp = spacy.load("en_core_web_md")  # Medium model with word vectors
except:
    # Fallback to small model if medium isn't available
    nlp = spacy.load("en_core_web_sm")


# ----------------------------------
# Resume Parser Component
# ----------------------------------

class ResumeParserModel(BaseModel):
    """Pydantic model for resume parsing results"""
    personal_info: Dict = Field(default_factory=dict)
    education: List[Dict] = Field(default_factory=list)
    experience: List[Dict] = Field(default_factory=list)
    skills: List[Dict] = Field(default_factory=list)
    projects: List[Dict] = Field(default_factory=list)
    certifications: List[Dict] = Field(default_factory=list)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    languages: List[Dict] = Field(default_factory=list)
    parsed_confidence: float = 0.0

class ResumeParser:
    """
    Class to parse resume documents and extract structured information
    """

    def __init__(self):
        """Initialize the resume parser with required models and patterns"""
        # Load skill taxonomy
        self.skill_patterns = self._load_skill_patterns()

        # Add skill patterns to NLP pipeline
        ruler = nlp.add_pipe("entity_ruler", before="ner")
        ruler.add_patterns(self.skill_patterns)

        # Regex patterns for parsing
        self.email_pattern = re.compile(r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}")
        self.phone_pattern = re.compile(r"(\+\d{1,3}[ -]?)?\(?\d{3}\)?[ -]?\d{3}[ -]?\d{4}")
        self.url_pattern = re.compile(r"(https?://(?:www\.)?[a-zA-Z0-9-]+\.[a-zA-Z]{2,}(?:/[^\s]*)?)")
        self.education_keywords = ["education", "degree", "university", "college", "bachelor",
                                   "master", "phd", "diploma", "academic", "gpa"]
        self.experience_keywords = ["experience", "work", "employment", "job", "position",
                                    "career", "professional", "role"]
        self.project_keywords = ["project", "portfolio", "implementation", "developed",
                                 "created", "built", "designed", "application"]
        self.cert_keywords = ["certification", "certificate", "certified", "license",
                              "credential", "qualification"]

    def _load_skill_patterns(self) -> List[Dict]:
        """Load skill patterns from database or file"""
        # In a production environment, these would be loaded from a database
        # For demonstration, we'll use a small sample of hardcoded patterns
        skills = [
            "Python", "JavaScript", "Java", "C++", "C#", "PHP", "TypeScript", "Ruby", "Swift",
            "Kotlin", "React", "Angular", "Vue.js", "Django", "Flask", "Laravel", "Spring Boot",
            "Express.js", "Node.js", "TensorFlow", "PyTorch", "scikit-learn", "pandas", "numpy",
            "SQL", "MongoDB", "PostgreSQL", "MySQL", "SQLite", "Oracle", "AWS", "Azure", "GCP",
            "Docker", "Kubernetes", "Jenkins", "Git", "GitHub", "GitLab", "CI/CD", "Agile",
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        "Scrum", "Kanban", "Jira", "Confluence", "MS Office", "Tableau", "Power BI"
    ]

    patterns = []
    for skill in skills:
        pattern = {
            "label": "SKILL",
            "pattern": skill,
            "id": f"skill_{len(patterns)}"
        }
        patterns.append(pattern)

    return patterns

def parse_text(self, text: str) -> ResumeParserModel:
    """
    Parse resume text and extract structured information

    Args:
        text: The resume text content

    Returns:
        A structured ResumeParserModel object with extracted information
    """
    # Process the text with spaCy
    doc = nlp(text)

    # Initialize result with default structure
    result = ResumeParserModel()

    # Extract personal information
    result.personal_info = self._extract_personal_info(doc, text)

    # Extract skills
    result.skills = self._extract_skills(doc)

    # Split text into sections
    sections = self._split_into_sections(text)

    # Extract education
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        result.education = self._extract_education(sections, doc)

        # Extract work experience
        result.experience = self._extract_experience(sections, doc)

        # Extract projects
        result.projects = self._extract_projects(sections, doc)

        # Extract certifications
        result.certifications = self._extract_certifications(sections, doc)

        # Calculate confidence score based on completeness
        result.parsed_confidence = self._calculate_confidence(result)

        return result

    def _extract_personal_info(self, doc, text: str) -> Dict:
        """Extract personal information like name, email, phone, etc."""
        personal_info = {}

        # Extract email
        email_matches = self.email_pattern.findall(text)
        if email_matches:
            personal_info["email"] = email_matches[0]

        # Extract phone
        phone_matches = self.phone_pattern.findall(text)
        if phone_matches:
            personal_info["phone"] = phone_matches[0]

        # Extract URLs (portfolio, LinkedIn, etc.)
        url_matches = self.url_pattern.findall(text)
        if url_matches:
            personal_info["urls"] = url_matches

        # Extract name (assume first 2-3 tokens of the document might be a name)
        # This is a simplistic approach; in production, use more sophisticated name extraction
        potential_name = " ".join([token.text for token in list(doc)[:3]
                                   if token.is_alpha and not token.is_stop])
        if potential_name:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            personal_info["name"] = potential_name

        return personal_info

    def _extract_skills(self, doc) -> List[Dict]:
        """Extract skills from the document"""
        skills = []
        seen_skills = set()

        # Extract skills from entity recognition
        for ent in doc.ents:
            if ent.label_ == "SKILL" and ent.text not in seen_skills:
                skills.append({
                    "name": ent.text,
                    "source": "entity_recognition",
                    "confidence": 0.9
                })
                seen_skills.add(ent.text)

        # Look for skills in noun chunks (less confidence)
        for chunk in doc.noun_chunks:
            if any(token.text.lower() in [s.lower() for s in self.skill_patterns] for token in chunk):
                skill_text = chunk.text
                if skill_text not in seen_skills:
                    skills.append({
                        "name": skill_text,
                        "source": "noun_chunk",
                        "confidence": 0.7
                    })
                    seen_skills.add(skill_text)

        return skills

    def _split_into_sections(self, text: str) -> Dict[str, str]:
        """
        Split resume text into sections based on common section headers
        """
        lines = text.split('\n')
        current_section = "header"
        sections = {current_section: []}
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        for line in lines:
            line = line.strip()
            if not line:
                continue

            # Check if this line is a section header
            if line.lower() in ["education", "academic background", "academic history"]:
                current_section = "education"
                sections[current_section] = []
            elif line.lower() in ["experience", "work experience", "professional experience", "employment
history"]:
                current_section = "experience"
                sections[current_section] = []
            elif line.lower() in ["projects", "project work", "portfolio"]:
                current_section = "projects"
                sections[current_section] = []
            elif line.lower() in ["certifications", "certificates", "credentials"]:
                current_section = "certifications"
                sections[current_section] = []
            elif line.lower() in ["skills", "technical skills", "competencies"]:
                current_section = "skills"
                sections[current_section] = []
            else:
                sections[current_section].append(line)

        # Convert lists to strings
        return {k: "\n".join(v) for k, v in sections.items()}

    def _extract_education(self, sections: Dict[str, str], doc) -> List[Dict]:
        """Extract education information from education section"""
        education = []

        # If we have an education section, process it
        if "education" in sections:
            edu_text = sections["education"]
            edu_doc = nlp(edu_text)

            # Look for education entities
            degree_pattern = re.compile(r"(Bachelor|Master|PhD|B\.S\.|M\.S\.|B\.A\.|M\.A\.|Ph\.D\.|"
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
                r"Bachelor's|Master's|Doctorate|BSc|MSc|MBA|Associate)")

        # Extract degree mentions
        degree_matches = degree_pattern.finditer(edu_text)
        for match in degree_matches:
            # Get surrounding context (30 characters before and after)
            start = max(0, match.start() - 30)
            end = min(len(edu_text), match.end() + 30)
            context = edu_text[start:end]

            # Process the context to extract structured data
            edu_item = {
                "degree": match.group(0),
                "raw_text": context
            }

            # Try to extract university/institution
            for ent in edu_doc.ents:
                if ent.label_ == "ORG" and ent.start_char >= start and ent.end_char <= end:
                    edu_item["institution"] = ent.text
                    break

            # Try to extract dates
            date_pattern = re.compile(r"(19|20)\d{2}(-(19|20)\d{2})?")
            date_matches = date_pattern.finditer(context)
            for date_match in date_matches:
                date_text = date_match.group(0)
                if "-" in date_text:
                    start_year, end_year = date_text.split("-")
                    edu_item["start_date"] = start_year
                    edu_item["end_date"] = end_year
                else:
                    # Assume this is an end date
                    edu_item["end_date"] = date_text

            education.append(edu_item)

    return education

def _extract_experience(self, sections: Dict[str, str], doc) -> List[Dict]:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    """Extract work experience information"""
    experience = []

    if "experience" in sections:
        exp_text = sections["experience"]
        exp_doc = nlp(exp_text)

        # Split experience section into individual experiences
        # This is a simplified approach - in production, more sophisticated parsing is needed
        exp_entries = re.split(r"\n{2,}", exp_text)

        for entry in exp_entries:
            if not entry.strip():
                continue

            exp_item = {
                "raw_text": entry.strip()
            }

            # Try to extract company/organization
            entry_doc = nlp(entry)
            for ent in entry_doc.ents:
                if ent.label_ == "ORG":
                    exp_item["company"] = ent.text
                    break

            # Try to extract job title
            title_pattern = re.compile(r"(Engineer|Developer|Manager|Analyst|Designer|Director|"
                                       r"Intern|Coordinator|Specialist|Consultant|Associate|"
                                       r"Assistant|Lead|Senior|Junior)")
            title_matches = title_pattern.finditer(entry)
            for match in title_matches:
                # Get surrounding context
                start = max(0, match.start() - 15)
                end = min(len(entry), match.end() + 15)
                context = entry[start:end]

                # Simplified job title extraction
                exp_item["title"] = context
                break
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            # Try to extract dates
            date_pattern = re.compile(r"((?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-
z]*\.?)\s+"
                                      r"(19|20)\d{2}\s*(?:-|–|to)\s*"
                                      r"((?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-z]*\.?)\s+"
                                      r"(19|20)\d{2}|Present|Current)")
            date_matches = date_pattern.finditer(entry)
            for date_match in date_matches:
                date_text = date_match.group(0)
                exp_item["date_range"] = date_text
                break

            experience.append(exp_item)

        return experience

    def _extract_projects(self, sections: Dict[str, str], doc) -> List[Dict]:
        """Extract project information"""
        projects = []

        if "projects" in sections:
            proj_text = sections["projects"]

            # Split projects section into individual projects
            proj_entries = re.split(r"\n{2,}", proj_text)

            for entry in entry_entries:
                if not entry.strip():
                    continue

                proj_item = {
                    "raw_text": entry.strip()
                }

                # Try to extract project title
                lines = entry.split("\n")
                if lines and lines[0].strip():
                    proj_item["title"] = lines[0].strip()
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            # Try to extract technologies/skills used
            entry_doc = nlp(entry)
            technologies = []
            for ent in entry_doc.ents:
                if ent.label_ == "SKILL":
                    technologies.append(ent.text)

            if technologies:
                proj_item["technologies"] = technologies

            projects.append(proj_item)

        return projects

    def _extract_certifications(self, sections: Dict[str, str], doc) -> List[Dict]:
        """Extract certification information"""
        certifications = []

        if "certifications" in sections:
            cert_text = sections["certifications"]

            # Split certifications section into individual entries
            cert_entries = re.split(r"\n+", cert_text)

            for entry in cert_entries:
                if not entry.strip():
                    continue

                cert_item = {
                    "name": entry.strip()
                }

                # Try to extract dates
                date_pattern = re.compile(r"(19|20)\d{2}")
                date_matches = date_pattern.finditer(entry)
                for date_match in date_matches:
                    cert_item["year"] = date_match.group(0)
                    break

                certifications.append(cert_item)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        return certifications

    def _calculate_confidence(self, result: ResumeParserModel) -> float:
        """Calculate overall confidence score of the parsing"""
        # Count how many sections we were able to extract
        sections_found = 0
        max_sections = 5  # personal_info, education, experience, skills, projects

        if result.personal_info:
            sections_found += 1

        if result.education:
            sections_found += 1

        if result.experience:
            sections_found += 1

        if result.skills:
            sections_found += 1

        if result.projects:
            sections_found += 1

        # Basic confidence score based on completeness
        confidence = sections_found / max_sections

        # Adjust based on amount of data in each section
        if len(result.skills) > 5:
            confidence += 0.1

        if len(result.education) > 0:
            confidence += 0.1

        if len(result.experience) > 1:
            confidence += 0.1

        # Cap at 1.0
        return min(confidence, 1.0)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# ---------------------------------------------------------------------------
# Resume Parser Component (Alternative Advanced Option)
# ---------------------------------------------------------------------------
import re
import spacy
from pydantic import BaseModel
from typing import List, Dict, Optional
from fastapi import FastAPI, File, UploadFile, HTTPException
import PyPDF2
import io

# Load lightweight spaCy model
try:
    nlp = spacy.load("en_core_web_sm")
except:
    import spacy.cli
    spacy.cli.download("en_core_web_sm")
    nlp = spacy.load("en_core_web_sm")

# Define custom entity patterns (for skills)
ruler = nlp.add_pipe("entity_ruler", before="ner")
patterns = [
    {"label": "SKILL", "pattern": "Python"},
    {"label": "SKILL", "pattern": "JavaScript"},
    {"label": "SKILL", "pattern": "React"},
    {"label": "SKILL", "pattern": "Laravel"},
    {"label": "SKILL", "pattern": "SQL"},
    # Add more patterns for common skills
]
ruler.add_patterns(patterns)

class Education(BaseModel):
    institution: Optional[str] = None
    degree: Optional[str] = None
    field_of_study: Optional[str] = None
    start_date: Optional[str] = None
    end_date: Optional[str] = None
    description: Optional[str] = None

class Experience(BaseModel):
    company: Optional[str] = None
    title: Optional[str] = None
    start_date: Optional[str] = None
    end_date: Optional[str] = None
    description: Optional[str] = None
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
class Skill(BaseModel):
    name: str
    level: Optional[str] = None
    category: Optional[str] = None

class ResumeData(BaseModel):
    personal_info: Dict = {}
    education: List[Education] = []
    experience: List[Experience] = []
    skills: List[Skill] = []
    confidence_score: float = 0.0

app = FastAPI()

@app.post("/parse-resume", response_model=ResumeData)
async def parse_resume(file: UploadFile = File(...)):
    try:
        # Extract text from PDF
        content = await file.read()
        text = extract_text_from_pdf(content)

        # Parse the text
        result = parse_resume_text(text)
        return result
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error parsing resume: {str(e)}")

def extract_text_from_pdf(content: bytes) -> str:
    """Extract text from PDF content"""
    try:
        pdf_reader = PyPDF2.PdfReader(io.BytesIO(content))
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text() + "\n"
        return text
    except:
        # If PyPDF2 fails, return content as text (might be plaintext resume)
        return content.decode('utf-8', errors='ignore')

def parse_resume_text(text: str) -> ResumeData:
    """Parse resume text into structured data"""
    # Process with spaCy
    doc = nlp(text)

    # Initialize result
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    result = ResumeData()

    # Extract personal information
    result.personal_info = extract_personal_info(text)

    # Extract education
    result.education = extract_education(text, doc)

    # Extract experience
    result.experience = extract_experience(text, doc)

    # Extract skills
    result.skills = extract_skills(text, doc)

    # Calculate confidence score
    result.confidence_score = calculate_confidence(result)

    return result

def extract_personal_info(text: str) -> Dict:
    """Extract personal information from text"""
    info = {}

    # Extract email
    email_pattern = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
    email_matches = re.findall(email_pattern, text)
    if email_matches:
        info['email'] = email_matches[0]

    # Extract phone
    phone_pattern = r'(\+\d{1,3}[ -]?)?\(?\d{3}\)?[ -]?\d{3}[ -]?\d{4}'
    phone_matches = re.findall(phone_pattern, text)
    if phone_matches:
        info['phone'] = phone_matches[0]

    # Extract LinkedIn URL
    linkedin_pattern = r'linkedin\.com/in/[a-zA-Z0-9_-]+'
    linkedin_matches = re.findall(linkedin_pattern, text)
    if linkedin_matches:
        info['linkedin'] = linkedin_matches[0]

    return info

def extract_education(text: str, doc) -> List[Education]:
    """Extract education information"""
    education = []
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    # Find education section
    edu_section = find_section(text, ['education', 'academic', 'degree', 'university'])
    if not edu_section:
        return education

    # Find education entries using regex patterns
    degree_pattern = r'(Bachelor|Master|PhD|B\.S\.|M\.S\.|B\.A\.|M\.A\.|Ph\.D\.|Associate)'
    university_pattern = r'(University|College|Institute|School) of ([A-Z][a-z]+ ?)+|(([A-Z][a-z]+ ?)+)
(University|College|Institute|School)'
    date_pattern =
r'(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|February|March|April|May|June|July|Augus
t|September|October|November|December),? (\d{4}) ?[-——]
?(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|February|March|April|May|June|July|Augus
t|September|October|November|December),? (\d{4}|Present|Current)|(\d{4}) ?[-——]
?(\d{4}|Present|Current)'

    # Find universities
    university_matches = re.finditer(university_pattern, edu_section)
    for match in university_matches:
        # Get surrounding context
        start = max(0, match.start() - 100)
        end = min(len(edu_section), match.end() + 200)
        context = edu_section[start:end]

        # Create education entry
        entry = Education(institution=match.group(0))

        # Find degree
        degree_match = re.search(degree_pattern, context)
        if degree_match:
            entry.degree = degree_match.group(0)

        # Find dates
        date_match = re.search(date_pattern, context)
        if date_match:
            if date_match.group(2) and date_match.group(4):  # Month Year format
                entry.start_date = f"{date_match.group(1)} {date_match.group(2)}"
                entry.end_date = f"{date_match.group(3)} {date_match.group(4)}"
            elif date_match.group(5) and date_match.group(6):  # Year only format
                entry.start_date = date_match.group(5)
                entry.end_date = date_match.group(6)

        education.append(entry)

    return education
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
def extract_experience(text: str, doc) -> List[Experience]:
    """Extract work experience information"""
    experience = []

    # Find experience section
    exp_section = find_section(text, ['experience', 'employment', 'work history', 'professional'])
    if not exp_section:
        return experience

    # Find company names and positions using regex and NER
    # For this simplified version, we'll use a pattern-based approach
    company_pattern = r'([A-Z][a-z]+ ?)+, (Inc|LLC|Ltd|Corporation|Corp|Company|Co)'
    position_pattern =
r'(Engineer|Developer|Manager|Director|Intern|Associate|Analyst|Designer|Consultant|Coordinator)'
    date_pattern =
r'(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|February|March|April|May|June|July|Augus
t|September|October|November|December),? (\d{4}) ?[-——]
?(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|January|February|March|April|May|June|July|Augus
t|September|October|November|December),? (\d{4}|Present|Current)|(\d{4}) ?[-——]
?(\d{4}|Present|Current)'

    # Split into paragraphs (potential experience entries)
    paragraphs = re.split(r'\n\s*\n', exp_section)

    for para in paragraphs:
        if len(para.strip()) > 30:  # Reasonable length for experience entry
            entry = Experience()

            # Try to find company
            company_match = re.search(company_pattern, para)
            if company_match:
                entry.company = company_match.group(0)

            # Try to find position
            position_match = re.search(position_pattern, para)
            if position_match:
                entry.title = position_match.group(0)

                # Look for additional context around the position
                pos_start = position_match.start()
                context_start = max(0, pos_start - 20)
                context_end = min(len(para), pos_start + 50)
                position_context = para[context_start:context_end]

                # Try to find a more complete title (e.g., "Senior Software Engineer")
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            extended_title_match = re.search(r'([A-Z][a-z]+ )+' + position_match.group(0),
position_context)
            if extended_title_match:
                entry.title = extended_title_match.group(0)

        # Try to find dates
        date_match = re.search(date_pattern, para)
        if date_match:
            if date_match.group(2) and date_match.group(4):  # Month Year format
                entry.start_date = f"{date_match.group(1)} {date_match.group(2)}"
                entry.end_date = f"{date_match.group(3)} {date_match.group(4)}"
            elif date_match.group(5) and date_match.group(6):  # Year only format
                entry.start_date = date_match.group(5)
                entry.end_date = date_match.group(6)

        # Add description
        entry.description = para.strip()

        # Only add if we have at least company or title
        if entry.company or entry.title:
            experience.append(entry)

    return experience

def extract_skills(text: str, doc) -> List[Skill]:
    """Extract skills from text"""
    skills = []

    # Find skills section
    skills_section = find_section(text, ['skills', 'technologies', 'technical', 'proficiencies'])

    # If we found a skills section, focus on that
    if skills_section:
        text_to_process = skills_section
    else:
        text_to_process = text

    # Process with spaCy to find skill entities
    skill_doc = nlp(text_to_process)

    # Extract skills from entity recognition
    skill_names = set()
    for ent in skill_doc.ents:
        if ent.label_ == "SKILL":
            skill_name = ent.text.strip()
            if skill_name and skill_name not in skill_names:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
                skills.append(Skill(name=skill_name))
                skill_names.add(skill_name)

    # Backup: Look for common programming languages and technologies
    skill_keywords = [
        "Python", "JavaScript", "Java", "C++", "C#", "Ruby", "Go", "Swift",
        "HTML", "CSS", "SQL", "NoSQL", "React", "Angular", "Vue", "Node.js",
        "Django", "Flask", "Laravel", "Spring", "Express", "TensorFlow",
        "PyTorch", "scikit-learn", "AWS", "Azure", "GCP", "Docker", "Kubernetes",
        "Git", "REST API", "GraphQL", "Agile", "Scrum", "JIRA", "CI/CD"
    ]

    for keyword in skill_keywords:
        if keyword.lower() in text_to_process.lower() and keyword not in skill_names:
            skills.append(Skill(name=keyword))
            skill_names.add(keyword)

    return skills

def find_section(text: str, keywords: List[str]) -> str:
    """Find a section in the resume text based on keywords"""
    lines = text.split('\n')
    section_start = -1
    section_end = len(lines)

    for i, line in enumerate(lines):
        # Check if this line is a section header
        if any(keyword.lower() in line.lower() for keyword in keywords):
            section_start = i + 1
        # Check if this is the start of next section (after we found our section)
        elif section_start != -1 and line.strip() and line[0].isupper() and ':' in line:
            section_end = i
            break

    if section_start != -1:
        return '\n'.join(lines[section_start:section_end])
    return ""

def calculate_confidence(result: ResumeData) -> float:
    """Calculate confidence score of parsed results"""
    score = 0.0

    # Check for personal info
    if result.personal_info.get('email'):
        score += 0.2
    if result.personal_info.get('phone'):
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            score += 0.1

        # Check for education
        if result.education:
            score += min(0.2, len(result.education) * 0.1)

        # Check for experience
        if result.experience:
            score += min(0.3, len(result.experience) * 0.1)

        # Check for skills
        if result.skills:
            score += min(0.2, len(result.skills) * 0.02)

        return min(1.0, score)

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("app:app", host="0.0.0.0", port=8000, reload=True)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# ------------------------------------------------------------------------------
# JOB MATCHING ALGORITHM (with Laravel Implementation)
# ------------------------------------------------------------------------------
<?php

namespace App\Services;

use App\Models\JobPosting;
use App\Models\Student;
use App\Models\Skill;
use App\Models\StudentSkill;
use App\Models\Education;
use App\Models\Experience;
use App\Models\Industry;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Collection;

class MatchingService
{
    protected $skillWeighting = 0.6;
    protected $educationWeighting = 0.15;
    protected $experienceWeighting = 0.15;
    protected $preferencesWeighting = 0.1;

    /**
     * Find matching students for a job posting with detailed scoring
     *
     * @param JobPosting $jobPosting
     * @param int $limit
     * @return array
     */
    public function findMatchingStudents(JobPosting $jobPosting, $limit = 20)
    {
        // Get required and preferred skills
        $requiredSkills = $jobPosting->requiredSkills()->get();
        $preferredSkills = $jobPosting->preferredSkills()->get();

        // Get job details for matching
        $jobIndustry = $jobPosting->industry_id;
        $jobExperienceLevel = $jobPosting->experience_level;
        $jobRemoteAllowed = $jobPosting->remote_allowed;
        $jobLocation = $jobPosting->location;

        // Get all eligible students
        $students = Student::with(['skills', 'education', 'experience', 'user'])
```

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

```
        ->where('status', 'active')
        ->get();

    $matchResults = [];

    foreach ($students as $student) {
        // Skip if student has already applied
        if ($jobPosting->applications()->where('student_id', $student->id)->exists()) {
            continue;
        }

        // Calculate skill match score
        $skillScore = $this->calculateSkillMatchScore(
            $student,
            $requiredSkills,
            $preferredSkills
        );

        // Calculate education match score
        $educationScore = $this->calculateEducationMatchScore(
            $student,
            $jobPosting
        );

        // Calculate experience match score
        $experienceScore = $this->calculateExperienceMatchScore(
            $student,
            $jobPosting
        );

        // Calculate preferences match score
        $preferencesScore = $this->calculatePreferencesMatchScore(
            $student,
            $jobPosting
        );

        // Calculate weighted total score
        $totalScore =
            ($this->skillWeighting * $skillScore) +
            ($this->educationWeighting * $educationScore) +
            ($this->experienceWeighting * $experienceScore) +
            ($this->preferencesWeighting * $preferencesScore);

        // Only include if above threshold
        if ($totalScore >= 0.3) {
            $matchResults[] = [
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
                'student' => $student,
                'total_score' => $totalScore,
                'skill_score' => $skillScore,
                'education_score' => $educationScore,
                'experience_score' => $experienceScore,
                'preferences_score' => $preferencesScore,
                'match_details' => [
                    'required_skills_matched' => $this
                'match_details' => [
                    'required_skills_matched' => $this->getMatchedSkills($student, $requiredSkills),
                    'preferred_skills_matched' => $this->getMatchedSkills($student, $preferredSkills),
                    'missing_skills' => $this->getMissingSkills($student, $requiredSkills),
                    'education_match_factors' => $this->getEducationMatchFactors($student, $jobPosting),
                    'experience_match_factors' => $this->getExperienceMatchFactors($student, $jobPosting),
                ]
            ];
        }
    }

    // Sort results by total score (descending)
    usort($matchResults, function($a, $b) {
        return $b['total_score'] <=> $a['total_score'];
    });

    // Limit to requested number
    return array_slice($matchResults, 0, $limit);
}

/**
 * Find matching jobs for a student with detailed scoring
 *
 * @param Student $student
 * @param int $limit
 * @return array
 */
public function findMatchingJobs(Student $student, $limit = 20)
{
    // Get student skills
    $studentSkills = $student->skills()->get();

    // Get active job postings
    $jobPostings = JobPosting::with(['requiredSkills', 'preferredSkills', 'employer', 'industry'])
        ->where('status', 'active')
        ->where('end_date', '>=', now())
        ->get();
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
$matchResults = [];

foreach ($jobPostings as $jobPosting) {
    // Skip if student has already applied
    if ($jobPosting->applications()->where('student_id', $student->id)->exists()) {
        continue;
    }

    // Calculate skill match score
    $skillScore = $this->calculateSkillMatchScore(
        $student,
        $jobPosting->requiredSkills,
        $jobPosting->preferredSkills
    );

    // Calculate education match score
    $educationScore = $this->calculateEducationMatchScore(
        $student,
        $jobPosting
    );

    // Calculate experience match score
    $experienceScore = $this->calculateExperienceMatchScore(
        $student,
        $jobPosting
    );

    // Calculate preferences match score
    $preferencesScore = $this->calculatePreferencesMatchScore(
        $student,
        $jobPosting
    );

    // Calculate weighted total score
    $totalScore =
        ($this->skillWeighting * $skillScore) +
        ($this->educationWeighting * $educationScore) +
        ($this->experienceWeighting * $experienceScore) +
        ($this->preferencesWeighting * $preferencesScore);

    // Only include if above threshold
    if ($totalScore >= 0.3) {
        $matchResults[] = [
            'job' => $jobPosting,
            'total_score' => $totalScore,
            'skill_score' => $skillScore,
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
                'education_score' => $educationScore,
                'experience_score' => $experienceScore,
                'preferences_score' => $preferencesScore,
                'match_details' => [
                    'required_skills_matched' => $this->getMatchedSkills($student, $jobPosting->requiredSkills),
                    'preferred_skills_matched' => $this->getMatchedSkills($student, $jobPosting->preferredSkills),
                    'missing_skills' => $this->getMissingSkills($student, $jobPosting->requiredSkills),
                    'education_match_factors' => $this->getEducationMatchFactors($student, $jobPosting),
                    'experience_match_factors' => $this->getExperienceMatchFactors($student, $jobPosting),
                ]
            ];
        }
    }

    // Sort results by total score (descending)
    usort($matchResults, function($a, $b) {
        return $b['total_score'] <=> $a['total_score'];
    });

    // Limit to requested number
    return array_slice($matchResults, 0, $limit);
}

/**
 * Calculate skill match score between student and job skills
 */
protected function calculateSkillMatchScore(Student $student, $requiredSkills, $preferredSkills)
{
    $studentSkillIds = $student->skills->pluck('id')->toArray();

    // Handle empty required skills
    if ($requiredSkills->isEmpty()) {
        return 0.5; // Neutral score if no required skills
    }

    // Count matched required skills
    $matchedRequiredCount = 0;
    foreach ($requiredSkills as $skill) {
        if (in_array($skill->id, $studentSkillIds)) {
            $matchedRequiredCount++;
        }
    }

    // Calculate required skills match ratio
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        $requiredSkillsCount = $requiredSkills->count();
        $requiredMatchRatio = $requiredSkillsCount > 0 ?
            $matchedRequiredCount / $requiredSkillsCount : 0;

        // Count matched preferred skills
        $matchedPreferredCount = 0;
        foreach ($preferredSkills as $skill) {
            if (in_array($skill->id, $studentSkillIds)) {
                $matchedPreferredCount++;
            }
        }

        // Calculate preferred skills match ratio
        $preferredSkillsCount = $preferredSkills->count();
        $preferredMatchRatio = $preferredSkillsCount > 0 ?
            $matchedPreferredCount / $preferredSkillsCount : 0;

        // Calculate weighted score (required skills are more important)
        if ($preferredSkillsCount > 0) {
            return (0.7 * $requiredMatchRatio) + (0.3 * $preferredMatchRatio);
        } else {
            return $requiredMatchRatio;
        }
    }

    /**
     * Calculate education match score
     */
    protected function calculateEducationMatchScore(Student $student, JobPosting $jobPosting)
    {
        // Define education level scores
        $educationLevels = [
            'high school' => 0.3,
            'associate' => 0.5,
            'bachelor' => 0.7,
            'master' => 0.9,
            'phd' => 1.0,
            'doctorate' => 1.0
        ];

        // Get student education
        $studentEducation = $student->education()->get();

        if ($studentEducation->isEmpty()) {
            return 0.0;
        }
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    // Find highest education level
    $highestEduScore = 0.0;
    foreach ($studentEducation as $edu) {
        $degree = strtolower($edu->degree ?? '');

        // Check each education level keyword
        foreach ($educationLevels as $level => $score) {
            if (strpos($degree, $level) !== false) {
                $highestEduScore = max($highestEduScore, $score);
                break;
            }
        }
    }

    // Get job-required education level
    $jobEduLevel = strtolower($jobPosting->education_level ?? '');
    $jobEduScore = 0.5; // Default middle score

    foreach ($educationLevels as $level => $score) {
        if (strpos($jobEduLevel, $level) !== false) {
            $jobEduScore = $score;
            break;
        }
    }

    // Calculate match based on how student education compares to job requirements
    if ($highestEduScore >= $jobEduScore) {
        return 1.0; // Perfect match or over-qualified
    } else {
        // Partial match - higher score as they get closer to required level
        return $highestEduScore / $jobEduScore;
    }
}

/**
 * Calculate experience match score
 */
protected function calculateExperienceMatchScore(Student $student, JobPosting $jobPosting)
{
    // Calculate total months of student experience
    $studentExperience = $student->experience()->get();
    $totalMonths = 0;

    foreach ($studentExperience as $exp) {
        $startDate = $exp->start_date;
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            $endDate = $exp->is_current ? now() : $exp->end_date;

            if ($startDate && $endDate) {
                $months = $startDate->diffInMonths($endDate);
                $totalMonths += $months;
            }
        }

        // Map job experience level to expected months
        $expectedMonths = [
            'entry_level' => 0,
            'internship' => 0,
            'junior' => 12,
            'mid_level' => 36,
            'senior' => 60,
            'executive' => 120
        ];

        // Get expected months for the job level
        $jobLevel = strtolower($jobPosting->experience_level ?? 'entry_level');
        $targetMonths = $expectedMonths[$jobLevel] ?? 0;

        // Calculate experience ratio (cap at 1.5)
        $experienceRatio = min(1.5, $totalMonths / max(1, $targetMonths));

        // Normalize to 0-1 range
        return min(1.0, $experienceRatio);
    }

    /**
     * Calculate preferences match score
     */
    protected function calculatePreferencesMatchScore(Student $student, JobPosting $jobPosting)
    {
        $score = 0.0;
        $factors = 0;

        // Location preference
        if ($student->preferences && isset($student->preferences['preferred_locations'])) {
            $factors++;
            $studentLocations = $student->preferences['preferred_locations'];

            if (is_array($studentLocations) && in_array($jobPosting->location, $studentLocations)) {
                $score += 1.0;
            } elseif (is_string($studentLocations) && $studentLocations == $jobPosting->location) {
                $score += 1.0;
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            }
        }

        // Remote work preference
        if ($student->preferences && isset($student->preferences['remote_work'])) {
            $factors++;
            $prefersRemote = $student->preferences['remote_work'];

            if ($prefersRemote && $jobPosting->remote_allowed) {
                $score += 1.0;
            } elseif (!$prefersRemote && !$jobPosting->remote_allowed) {
                $score += 1.0;
            }
        }

        // Industry preference
        if ($student->preferences && isset($student->preferences['preferred_industries'])) {
            $factors++;
            $preferredIndustries = $student->preferences['preferred_industries'];

            if (is_array($preferredIndustries) && in_array($jobPosting->industry_id, $preferredIndustries)) {
                $score += 1.0;
            } elseif (is_numeric($preferredIndustries) && $preferredIndustries == $jobPosting->industry_id)
{
                $score += 1.0;
            }
        }

        // Salary range preference
        if ($student->preferences && isset($student->preferences['salary_expectation']) && $jobPosting-
>salary_range) {
            $factors++;
            $expectedSalary = $student->preferences['salary_expectation'];

            // Parse salary range (assuming format like "50000-70000")
            $salaryRange = explode('-', $jobPosting->salary_range);
            if (count($salaryRange) == 2) {
                $minSalary = (int) $salaryRange[0];
                $maxSalary = (int) $salaryRange[1];

                if ($expectedSalary <= $maxSalary) {
                    // Full match if within range, partial if close
                    if ($expectedSalary >= $minSalary) {
                        $score += 1.0;
                    } else {
                        // Partial score if within 20% below range
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
                    $buffer = $minSalary * 0.2;
                    if ($expectedSalary >= ($minSalary - $buffer)) {
                        $score += 0.5;
                    }
                }
            }
        }
    }

    // Return average score across all factors
    return $factors > 0 ? $score / $factors : 0.5;
}

/**
 * Get matched skills between student and job
 */
protected function getMatchedSkills(Student $student, $skills)
{
    $studentSkillIds = $student->skills->pluck('id')->toArray();
    return $skills->filter(function($skill) use ($studentSkillIds) {
        return in_array($skill->id, $studentSkillIds);
    })->values();
}

/**
 * Get missing skills between student and job
 */
protected function getMissingSkills(Student $student, $skills)
{
    $studentSkillIds = $student->skills->pluck('id')->toArray();
    return $skills->filter(function($skill) use ($studentSkillIds) {
        return !in_array($skill->id, $studentSkillIds);
    })->values();
}

/**
 * Get education match factors
 */
protected function getEducationMatchFactors(Student $student, JobPosting $jobPosting)
{
    $studentEducation = $student->education()->get();
    $jobEduLevel = strtolower($jobPosting->education_level ?? '');

    $factors = [
        'student_highest_education' => null,
        'job_required_education' => $jobEduLevel,
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            'is_match' => false
        ];

        // Define education level hierarchy
        $eduLevels = [
            'high school' => 1,
            'associate' => 2,
            'bachelor' => 3,
            'master' => 4,
            'phd' => 5,
            'doctorate' => 5
        ];

        // Find student's highest education level
        $highestLevel = 0;
        $highestDegree = null;

        foreach ($studentEducation as $edu) {
            $degree = strtolower($edu->degree ?? '');

            foreach ($eduLevels as $level => $value) {
                if (strpos($degree, $level) !== false && $value > $highestLevel) {
                    $highestLevel = $value;
                    $highestDegree = $edu->degree;
                }
            }
        }

        $factors['student_highest_education'] = $highestDegree;

        // Determine job required level
        $jobLevel = 0;
        foreach ($eduLevels as $level => $value) {
            if (strpos($jobEduLevel, $level) !== false && $value > $jobLevel) {
                $jobLevel = $value;
            }
        }

        // Determine if match
        $factors['is_match'] = $highestLevel >= $jobLevel;

        return $factors;
    }

    /**
     * Get experience match factors
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
     */
    protected function getExperienceMatchFactors(Student $student, JobPosting $jobPosting)
    {
        // Calculate total months of student experience
        $studentExperience = $student->experience()->get();
        $totalMonths = 0;

        foreach ($studentExperience as $exp) {
            $startDate = $exp->start_date;
            $endDate = $exp->is_current ? now() : $exp->end_date;

            if ($startDate && $endDate) {
                $months = $startDate->diffInMonths($endDate);
                $totalMonths += $months;
            }
        }

        // Map job experience level to expected months
        $expectedMonths = [
            'entry_level' => 0,
            'internship' => 0,
            'junior' => 12,
            'mid_level' => 36,
            'senior' => 60,
            'executive' => 120
        ];

        // Get expected months for the job level
        $jobLevel = strtolower($jobPosting->experience_level ?? 'entry_level');
        $targetMonths = $expectedMonths[$jobLevel] ?? 0;

        return [
            'student_experience_months' => $totalMonths,
            'job_required_months' => $targetMonths,
            'experience_ratio' => $targetMonths > 0 ? $totalMonths / $targetMonths : 1.0,
            'is_match' => $totalMonths >= $targetMonths
        ];
    }
}
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# ------------------------------------------------------------------------------------------
# SKILL GAP ANALYSIS (Advanced Implementation)
# ------------------------------------------------------------------------------------------
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from typing import List, Dict, Optional
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import spacy

app = FastAPI()

# Load lightweight spaCy model
try:
    nlp = spacy.load("en_core_web_sm")
except:
    import spacy.cli
    spacy.cli.download("en_core_web_sm")
    nlp = spacy.load("en_core_web_sm")

class Skill(BaseModel):
    id: Optional[int] = None
    name: str
    proficiency: Optional[int] = None
    category: Optional[str] = None

class SkillGapInput(BaseModel):
    student_id: int
    job_posting_id: Optional[int] = None
    student_skills: List[Skill] = Field(default_factory=list)
    target_skills: List[str] = Field(default_factory=list)
    career_goal: Optional[str] = None

class LearningResource(BaseModel):
    title: str
    url: str
    provider: str
    duration_hours: float
    is_free: bool
    skill_name: str
    description: Optional[str] = None

class MissingSkill(BaseModel):
    name: str
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    importance: float
    difficulty: float
    related_skills: List[str] = Field(default_factory=list)

class SkillGapOutput(BaseModel):
    missing_skills: List[MissingSkill] = Field(default_factory=list)
    skill_strengths: List[Dict] = Field(default_factory=list)
    learning_resources: List[LearningResource] = Field(default_factory=list)
    career_path_alignment: Optional[float] = None
    development_timeline: Optional[Dict] = None

# Skill importance dictionary (mock data)
SKILL_IMPORTANCE = {
    "python": 0.95,
    "javascript": 0.92,
    "react": 0.88,
    "node.js": 0.85,
    "sql": 0.87,
    "machine learning": 0.82,
    "data analysis": 0.80,
    "aws": 0.83,
    "docker": 0.78,
    "kubernetes": 0.74,
    "git": 0.89,
    "agile": 0.76,
    "communication": 0.90,
    "teamwork": 0.85,
    "problem solving": 0.92
}

# Skill relationships graph (mock data)
SKILL_GRAPH = {
    "python": ["data analysis", "machine learning", "django", "flask"],
    "javascript": ["react", "node.js", "angular", "vue.js"],
    "react": ["javascript", "redux", "react native"],
    "node.js": ["javascript", "express.js", "mongodb"],
    "sql": ["database design", "postgresql", "mysql"],
    "machine learning": ["python", "tensorflow", "scikit-learn"],
    "data analysis": ["python", "pandas", "sql", "tableau"],
    "aws": ["cloud computing", "devops", "lambda"],
    "docker": ["containerization", "kubernetes", "devops"],
    "kubernetes": ["docker", "devops", "container orchestration"],
    "git": ["version control", "github", "gitlab"],
    "agile": ["scrum", "kanban", "project management"]
}
```

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

```
# Learning resources (mock data)
LEARNING_RESOURCES = {
    "python": [
        {
            "title": "Learn Python Programming",
            "provider": "Codecademy",
            "url": "https://www.codecademy.com/learn/learn-python-3",
            "duration_hours": 30,
            "is_free": True,
            "description": "Comprehensive Python course for beginners"
        },
        {
            "title": "Python for Everybody",
            "provider": "Coursera",
            "url": "https://www.coursera.org/specializations/python",
            "duration_hours": 76,
            "is_free": False,
            "description": "Specialization teaching Python from the ground up"
        }
    ],
    "javascript": [
        {
            "title": "JavaScript Basics",
            "provider": "MDN Web Docs",
            "url": "https://developer.mozilla.org/en-US/docs/Learn/JavaScript",
            "duration_hours": 15,
            "is_free": True,
            "description": "The definitive guide to JavaScript by Mozilla"
        },
        {
            "title": "JavaScript: Understanding the Weird Parts",
            "provider": "Udemy",
            "url": "https://www.udemy.com/course/understand-javascript/",
            "duration_hours": 12,
            "is_free": False,
            "description": "Deep dive into JavaScript internals"
        }
    ]
    # Add more resources for other skills
}

@app.post("/skill-gap-analysis", response_model=SkillGapOutput)
async def analyze_skill_gap(input_data: SkillGapInput):
    """Analyze skill gap between student skills and target skills"""
    try:
        # Extract student skills
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
student_skill_dict = {skill.name.lower(): skill.proficiency or 5 for skill in input_data.student_skills}

# Process target skills
target_skills_lower = [skill.lower() for skill in input_data.target_skills]

# Initialize result
result = SkillGapOutput()

# Identify missing skills
for skill in target_skills_lower:
    if skill not in student_skill_dict:
        # Get importance score (default 0.5 if not in our dictionary)
        importance = SKILL_IMPORTANCE.get(skill, 0.5)

        # Get related skills
        related_skills = SKILL_GRAPH.get(skill, [])

        # Check if student has any related skills
        related_skill_proficiency = 0
        for related in related_skills:
            if related in student_skill_dict:
                related_skill_proficiency = max(related_skill_proficiency, student_skill_dict[related])

        # Calculate difficulty (lower if student has related skills)
        difficulty = 1.0 - (related_skill_proficiency / 10) * 0.5

        # Add to missing skills
        result.missing_skills.append(MissingSkill(
            name=skill,
            importance=importance,
            difficulty=difficulty,
            related_skills=related_skills
        ))

# Sort missing skills by importance
result.missing_skills.sort(key=lambda x: x.importance, reverse=True)

# Identify skill strengths
for skill, proficiency in student_skill_dict.items():
    if skill in target_skills_lower and proficiency >= 7:  # High proficiency
        result.skill_strengths.append({
            "name": skill,
            "proficiency": proficiency,
            "importance": SKILL_IMPORTANCE.get(skill, 0.5)
        })
```

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

```
    # Sort skill strengths by proficiency
    result.skill_strengths.sort(key=lambda x: x["proficiency"], reverse=True)

    # Recommend learning resources for top missing skills
    for missing_skill in result.missing_skills[:5]:  # Top 5 missing skills
        skill_name = missing_skill.name
        resources = LEARNING_RESOURCES.get(skill_name, [])

        # If no direct resources, check related skills
        if not resources and missing_skill.related_skills:
            for related in missing_skill.related_skills:
                related_resources = LEARNING_RESOURCES.get(related, [])
                if related_resources:
                    for resource_data in related_resources:
                        resource = LearningResource(
                            skill_name=skill_name,
                            title=f"{resource_data['title']} (related to {related})",
                            url=resource_data["url"],
                            provider=resource_data["provider"],
                            duration_hours=resource_data["duration_hours"],
                            is_free=resource_data["is_free"],
                            description=resource_data.get("description")
                        )
                        result.learning_resources.append(resource)
        else:
            for resource_data in resources:
                resource = LearningResource(
                    skill_name=skill_name,
                    title=resource_data["title"],
                    url=resource_data["url"],
                    provider=resource_data["provider"],
                    duration_hours=resource_data["duration_hours"],
                    is_free=resource_data["is_free"],
                    description=resource_data.get("description")
                )
                result.learning_resources.append(resource)

    # Calculate career path alignment if career goal provided
    if input_data.career_goal:
        # Simple similarity-based alignment
        goal_doc = nlp(input_data.career_goal.lower())
        skill_text = " ".join(target_skills_lower)
        skills_doc = nlp(skill_text)

        # Calculate semantic similarity
        if goal_doc.vector_norm and skills_doc.vector_norm:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            result.career_path_alignment = goal_doc.similarity(skills_doc)
        else:
            # Fallback if vectors not available
            aligned_skills = 0
            total_target_skills = len(target_skills_lower)
            for skill in student_skill_dict:
                if skill in target_skills_lower:
                    aligned_skills += 1
            result.career_path_alignment = aligned_skills / total_target_skills if total_target_skills > 0 else
0

        # Generate development timeline
        result.development_timeline = generate_development_timeline(result.missing_skills)

        return result
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error analyzing skill gap: {str(e)}")

def generate_development_timeline(missing_skills: List[MissingSkill]) -> Dict:
    """Generate a development timeline for acquiring missing skills"""
    if not missing_skills:
        return {
            "total_weeks": 0,
            "phases": []
        }

    # Calculate learning time for each skill
    skills_with_time = []
    for skill in missing_skills:
        # Base learning time in weeks (1-12 weeks)
        base_time = skill.difficulty * 12

        # Adjust based on importance (more important = more time invested)
        adjusted_time = base_time * (0.7 + skill.importance * 0.3)

        # Reduce time if student has related skills
        has_related = len(skill.related_skills) > 0
        if has_related:
            adjusted_time *= 0.7

        skills_with_time.append({
            "name": skill.name,
            "weeks": max(1, round(adjusted_time)),
            "importance": skill.importance
        })
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    # Organize skills into phases
    phases = []
    current_phase = []
    current_phase_weeks = 0
    max_phase_weeks = 8  # Maximum weeks per phase

    for skill in skills_with_time:
        # If skill would push phase over max duration, start a new phase
        if current_phase_weeks + skill["weeks"] > max_phase_weeks and current_phase:
            phases.append({
                "skills": current_phase,
                "duration_weeks": current_phase_weeks
            })
            current_phase = []
            current_phase_weeks = 0

        current_phase.append(skill)
        current_phase_weeks += skill["weeks"]

    # Add the last phase if it has skills
    if current_phase:
        phases.append({
            "skills": current_phase,
            "duration_weeks": current_phase_weeks
        })

    # Calculate total timeline duration
    total_weeks = sum(phase["duration_weeks"] for phase in phases)

    return {
        "total_weeks": total_weeks,
        "phases": phases
    }

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("app:app", host="0.0.0.0", port=8001, reload=True)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# ----------------------------------------------------------------------------------------
# READINESS SCORE COMPONENT (Advanced Implementation)
# ----------------------------------------------------------------------------------------
class ReadinessScoreInput(BaseModel):
    """Input model for readiness score calculation"""
    student_id: int
    job_posting_id: int
    student_skills: List[Dict] = Field(default_factory=list)
    student_education: List[Dict] = Field(default_factory=list)
    student_experience: List[Dict] = Field(default_factory=list)
    student_projects: List[Dict] = Field(default_factory=list)
    student_certifications: List[Dict] = Field(default_factory=list)
    job_required_skills: List[str] = Field(default_factory=list)
    job_preferred_skills: List[str] = Field(default_factory=list)
    job_description: str = ""
    job_title: str = ""
    job_experience_level: str = ""

class ReadinessScoreOutput(BaseModel):
    """Output model for readiness score calculation"""
    overall_score: float
    skill_match_score: float
    education_score: float
    experience_score: float
    project_score: float
    certification_score: float
    breakdown: Dict[str, Any] = Field(default_factory=dict)
    recommendations: List[Dict] = Field(default_factory=list)

class ReadinessScorer:
    """
    Class to calculate job readiness score for students
    """

    def __init__(self):
        """Initialize the readiness scorer with required models and weights"""
        self.vectorizer = TfidfVectorizer(stop_words='english',
                                          max_features=5000,
                                          ngram_range=(1, 2))
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        # Define feature weights
        self.weights = {
            'skill_match': 0.35,
            'education': 0.20,
            'experience': 0.25,
            'projects': 0.15,
            'certifications': 0.05
        }

        # Experience level mapping
        self.experience_levels = {
            'entry_level': 0,
            'junior': 1,
            'mid_level': 2,
            'senior': 3,
            'executive': 4
        }

    def calculate_readiness(self, input_data: ReadinessScoreInput) -> ReadinessScoreOutput:
        """
        Calculate the overall readiness score for a student-job pair

        Args:
            input_data: The ReadinessScoreInput containing student and job information

        Returns:
            A ReadinessScoreOutput with overall and component scores
        """
        # Initialize result
        result = ReadinessScoreOutput(
            overall_score=0.0,
            skill_match_score=0.0,
            education_score=0.0,
            experience_score=0.0,
            project_score=0.0,
            certification_score=0.0,
            breakdown={},
            recommendations=[]
        )
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# Calculate skill match score
result.skill_match_score = self._calculate_skill_match(
    input_data.student_skills,
    input_data.job_required_skills,
    input_data.job_preferred_skills
)

# Calculate education score
result.education_score = self._calculate_education_score(
    input_data.student_education,
    input_data.job_title,
    input_data.job_description
)

# Calculate experience score
result.experience_score = self._calculate_experience_score(
    input_data.student_experience,
    input_data.job_experience_level,
    input_data.job_description
)

# Calculate project score
result.project_score = self._calculate_project_score(
    input_data.student_projects,
    input_data.job_required_skills,
    input_data.job_description
)

# Calculate certification score
result.certification_score = self._calculate_certification_score(
    input_data.student_certifications,
    input_data.job_required_skills
)

# Calculate overall score using weights
result.overall_score = (
    self.weights['skill_match'] * result.skill_match_score +
    self.weights['education'] * result.education_score +
    self.weights['experience'] * result.experience_score +
    self.weights['projects'] * result.project_score +
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        self.weights['certifications'] * result.certification_score
    )

    # Add detailed breakdown
    result.breakdown = {
        'skill_match': {
            'score': result.skill_match_score,
            'weight': self.weights['skill_match'],
            'contribution': result.skill_match_score * self.weights['skill_match'],
            'details': self._get_skill_match_details(
                input_data.student_skills,
                input_data.job_required_skills,
                input_data.job_preferred_skills
            )
        },
        'education': {
            'score': result.education_score,
            'weight': self.weights['education'],
            'contribution': result.education_score * self.weights['education']
        },
        'experience': {
            'score': result.experience_score,
            'weight': self.weights['experience'],
            'contribution': result.experience_score * self.weights['experience']
        },
        'projects': {
            'score': result.project_score,
            'weight': self.weights['projects'],
            'contribution': result.project_score * self.weights['projects']
        },
        'certifications': {
            'score': result.certification_score,
            'weight': self.weights['certifications'],
            'contribution': result.certification_score * self.weights['certifications']
        }
    }

    # Generate recommendations
    result.recommendations = self._generate_recommendations(
        input_data,
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            result
        )

        return result

    def _calculate_skill_match(self,
                               student_skills: List[Dict],
                               required_skills: List[str],
                               preferred_skills: List[str]) -> float:
        """Calculate the skill match score"""
        if not required_skills:
            return 0.5  # Default middle score if no required skills

        # Extract student skill names
        student_skill_names = [skill['name'].lower() for skill in student_skills]

        # Count matched required skills
        required_matches = sum(1 for skill in required_skills
                               if skill.lower() in student_skill_names)
        required_match_ratio = required_matches / len(required_skills) if required_skills else 0

        # Count matched preferred skills
        preferred_matches = sum(1 for skill in preferred_skills
                                if skill.lower() in student_skill_names)
        preferred_match_ratio = preferred_matches / len(preferred_skills) if preferred_skills else 0

        # Calculate weighted score (required skills are more important)
        if preferred_skills:
            return 0.7 * required_match_ratio + 0.3 * preferred_match_ratio
        else:
            return required_match_ratio

    def _get_skill_match_details(self,
                                 student_skills: List[Dict],
                                 required_skills: List[str],
                                 preferred_skills: List[str]) -> Dict:
        """Get detailed breakdown of skill matching"""
        student_skill_names = [skill['name'].lower() for skill in student_skills]

        required_matches = [skill for skill in required_skills
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
                    if skill.lower() in student_skill_names]
        required_misses = [skill for skill in required_skills
                           if skill.lower() not in student_skill_names]

        preferred_matches = [skill for skill in preferred_skills
                             if skill.lower() in student_skill_names]
        preferred_misses = [skill for skill in preferred_skills
                            if skill.lower() not in student_skill_names]

        return {
            'required_skills': {
                'total': len(required_skills),
                'matched': len(required_matches),
                'missing': len(required_misses),
                'match_ratio': len(required_matches) / len(required_skills) if required_skills else 0,
                'matched_skills': required_matches,
                'missing_skills': required_misses
            },
            'preferred_skills': {
                'total': len(preferred_skills),
                'matched': len(preferred_matches),
                'missing': len(preferred_misses),
                'match_ratio': len(preferred_matches) / len(preferred_skills) if preferred_skills else 0,
                'matched_skills': preferred_matches,
                'missing_skills': preferred_misses
            }
        }

    def _calculate_education_score(self,
                                   student_education: List[Dict],
                                   job_title: str,
                                   job_description: str) -> float:
        """Calculate the education score"""
        if not student_education:
            return 0.0

        # Define education level scores
        education_levels = {
            'high school': 0.3,
            'associate': 0.5,
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            'bachelor': 0.7,
            'master': 0.9,
            'phd': 1.0,
            'doctorate': 1.0
        }

        # Find highest education level
        highest_edu_score = 0.0
        for edu in student_education:
            degree = edu.get('degree', '').lower()

            # Check each education level keyword
            for level, score in education_levels.items():
                if level in degree:
                    highest_edu_score = max(highest_edu_score, score)
                    break

        # Relevance score based on field of study and job
        relevance_score = 0.0
        job_doc = nlp(f"{job_title} {job_description}")

        for edu in student_education:
            field = edu.get('field_of_study', '')
            if field:
                field_doc = nlp(field)
                similarity = field_doc.similarity(job_doc)
                relevance_score = max(relevance_score, similarity)

        # Combine education level and relevance
        return 0.6 * highest_edu_score + 0.4 * relevance_score

    def _calculate_experience_score(self,
                                    student_experience: List[Dict],
                                    job_experience_level: str,
                                    job_description: str) -> float:
        """Calculate the experience score"""
        if not student_experience:
            return 0.0

        # Calculate total months of experience
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
total_months = 0
for exp in student_experience:
    # Extract start and end dates
    start_date = exp.get('start_date')
    end_date = exp.get('end_date', 'present')

    # Convert to datetime objects for calculation
    if isinstance(start_date, str) and start_date:
        try:
            start_date = datetime.strptime(start_date, '%Y-%m-%d')
        except ValueError:
            try:
                start_date = datetime.strptime(start_date, '%Y')
            except ValueError:
                continue
    else:
        continue

    if end_date == 'present' or end_date == 'current':
        end_date = datetime.now()
    elif isinstance(end_date, str) and end_date:
        try:
            end_date = datetime.strptime(end_date, '%Y-%m-%d')
        except ValueError:
            try:
                end_date = datetime.strptime(end_date, '%Y')
            except ValueError:
                continue

    # Calculate months
    months = (end_date.year - start_date.year) * 12 + (end_date.month - start_date.month)
    total_months += max(0, months)

# Map job experience level to expected months
expected_months = {
    'entry_level': 0,
    'junior': 12,
    'mid_level': 36,
    'senior': 60,
    'executive': 120
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        }

        # Get expected months for the job level
        target_months = expected_months.get(job_experience_level.lower(), 0)

        # Calculate experience ratio (cap at 1.5)
        experience_ratio = min(1.5, total_months / max(1, target_months))

        # Normalize to 0-1 range
        experience_score = min(1.0, experience_ratio)

        # Calculate relevance of experience to job
        relevance_score = 0.0
        job_doc = nlp(job_description)

        for exp in student_experience:
            exp_text = f"{exp.get('title', '')} {exp.get('description', '')}"
            if exp_text.strip():
                exp_doc = nlp(exp_text)
                similarity = exp_doc.similarity(job_doc)
                relevance_score = max(relevance_score, similarity)

        # Combine experience length and relevance
        return 0.7 * experience_score + 0.3 * relevance_score

    def _calculate_project_score(self,
                                 student_projects: List[Dict],
                                 job_required_skills: List[str],
                                 job_description: str) -> float:
        """Calculate the project score"""
        if not student_projects:
            return 0.0

        # Calculate skill coverage in projects
        skill_coverage_score = 0.0
        if job_required_skills:
            required_skills_lower = [skill.lower() for skill in job_required_skills]
            skills_covered = set()

            for project in student_projects:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            # Extract technologies from project
            technologies = project.get('technologies', [])
            if isinstance(technologies, list):
                tech_names = [tech.lower() for tech in technologies]

                # Check which required skills are covered by project technologies
                for skill in required_skills_lower:
                    if skill in tech_names:
                        skills_covered.add(skill)

            # Also check in description
            description = project.get('description', '')
            if description:
                desc_doc = nlp(description)
                for skill in required_skills_lower:
                    if skill in description.lower():
                        skills_covered.add(skill)

        # Calculate coverage ratio
        skill_coverage_score = len(skills_covered) / len(required_skills_lower)

    # Calculate relevance of projects to job
    relevance_score = 0.0
    job_doc = nlp(job_description)

    for project in student_projects:
        project_text = f"{project.get('title', '')} {project.get('description', '')}"
        if project_text.strip():
            project_doc = nlp(project_text)
            similarity = project_doc.similarity(job_doc)
            relevance_score = max(relevance_score, similarity)

    # Factor in number of projects (more projects is better, up to a point)
    project_count_score = min(1.0, len(student_projects) / 5)

    # Combine scores
    return 0.4 * skill_coverage_score + 0.4 * relevance_score + 0.2 * project_count_score

def _calculate_certification_score(self,
                                   student_certifications: List[Dict],
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
                                job_required_skills: List[str]) -> float:
        """Calculate the certification score"""
        if not student_certifications:
            return 0.0

        # Count relevant certifications
        relevant_cert_count = 0

        # Convert required skills to lower case for matching
        required_skills_lower = [skill.lower() for skill in job_required_skills]

        for cert in student_certifications:
            cert_name = cert.get('name', '').lower()

            # Check if certification name contains any required skill
            is_relevant = any(skill in cert_name for skill in required_skills_lower)

            if is_relevant:
                relevant_cert_count += 1

        # Calculate score based on number of relevant certifications
        return min(1.0, relevant_cert_count / max(1, len(required_skills_lower) / 2))

    def _generate_recommendations(self,
                                  input_data: ReadinessScoreInput,
                                  result: ReadinessScoreOutput) -> List[Dict]:
        """Generate improvement recommendations based on scores"""
        recommendations = []

        # Extract missing required and preferred skills
        skill_breakdown = result.breakdown['skill_match']['details']
        missing_required = skill_breakdown['required_skills']['missing_skills']
        missing_preferred = skill_breakdown['preferred_skills']['missing_skills']

        # Recommend acquiring missing required skills
        if missing_required:
            recommendations.append({
                'type': 'skill_acquisition',
                'importance': 'high',
                'description': f"Acquire these required skills: {', '.join(missing_required[:3])}",
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            'details': {
                'skills': missing_required,
                'reason': 'These are required skills for the job that you don\'t have'
            }
        })

    # Recommend acquiring key preferred skills
    if missing_preferred:
        recommendations.append({
            'type': 'skill_enhancement',
            'importance': 'medium',
            'description': f"Consider learning these preferred skills: {', '.join(missing_preferred[:3])}",
            'details': {
                'skills': missing_preferred,
                'reason': 'These are preferred skills that would make you more competitive'
            }
        })

    # Education recommendations
    if result.education_score < 0.6:
        recommendations.append({
            'type': 'education',
            'importance': 'medium',
            'description': "Consider additional education or certifications relevant to this field",
            'details': {
                'current_score': result.education_score,
                'threshold': 0.6,
                'reason': 'Your educational background isn\'t strongly aligned with this position'
            }
        })

    # Experience recommendations
    if result.experience_score < 0.5:
        recommendations.append({
            'type': 'experience',
            'importance': 'high',
            'description': "Gain more relevant work experience through internships or projects",
            'details': {
                'current_score': result.experience_score,
                'threshold': 0.5,
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
                'reason': 'You need more relevant experience for this position'
            }
        })

    # Project recommendations
    if result.project_score < 0.7:
        missing_skills_for_projects = missing_required[:5] if missing_required else missing_preferred[:5]
        if missing_skills_for_projects:
            skill_list = ', '.join(missing_skills_for_projects)
            recommendations.append({
                'type': 'projects',
                'importance': 'medium',
                'description': f"Create projects that demonstrate skills in: {skill_list}",
                'details': {
                    'current_score': result.project_score,
                    'threshold': 0.7,
                    'skills_to_demonstrate': missing_skills_for_projects,
                    'reason': 'Projects help demonstrate practical skills to employers'
                }
            })

    return recommendations

# ----------------------------------------------------------------------------------------
# SKILL GAP ANALYSER COMPONENT (ADVANCED WITH AI)
# ----------------------------------------------------------------------------------------
class SkillGapInput(BaseModel):
    """Input model for skill gap analysis"""
    student_id: int
    job_posting_id: Optional[int] = None
    job_title: Optional[str] = None
    student_skills: List[Dict] = Field(default_factory=list)
    target_skills: List[str] = Field(default_factory=list)
    career_goal: Optional[str] = None

class SkillGapOutput(BaseModel):
    """Output model for skill gap analysis"""
    missing_skills: List[Dict] = Field(default_factory=list)
    skill_strengths: List[Dict] = Field(default_factory=list)
    learning_resources: List[Dict] = Field(default_factory=list)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    career_path_alignment: Optional[float] = None
    development_timeline: Optional[Dict] = None

class SkillGapAnalyzer:
    """
    Class to analyze skill gaps between student skills and job requirements
    """

    def __init__(self):
        """Initialize the skill gap analyzer with required models"""
        # Skill importance model (simplified mock version)
        self.skill_importance = self._load_skill_importance()

        # Skill relationships graph (simplified mock version)
        self.skill_graph = self._load_skill_graph()

        # Learning resource database (simplified mock version)
        self.learning_resources = self._load_learning_resources()

    def _load_skill_importance(self) -> Dict[str, float]:
        """Load skill importance model"""
        # In production, this would be loaded from a database or pre-trained model
        # Mock data for demonstration
        return {
            "python": 0.95,
            "javascript": 0.92,
            "react": 0.88,
            "node.js": 0.85,
            "sql": 0.87,
            "machine learning": 0.82,
            "data analysis": 0.80,
            "aws": 0.83,
            "docker": 0.78,
            "kubernetes": 0.74,
            "git": 0.89,
            "agile": 0.76,
            "communication": 0.90,
            "teamwork": 0.85,
            "problem solving": 0.92
        }
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
def _load_skill_graph(self) -> Dict[str, List[str]]:
    """Load skill relationship graph"""
    # In production, this would be a more sophisticated graph structure
    # Mock data for demonstration
    return {
        "python": ["data analysis", "machine learning", "django", "flask"],
        "javascript": ["react", "node.js", "angular", "vue.js"],
        "react": ["javascript", "redux", "react native"],
        "node.js": ["javascript", "express.js", "mongodb"],
        "sql": ["database design", "postgresql", "mysql"],
        "machine learning": ["python", "tensorflow", "scikit-learn"],
        "data analysis": ["python", "pandas", "sql", "tableau"],
        "aws": ["cloud computing", "devops", "lambda"],
        "docker": ["containerization", "kubernetes", "devops"],
        "kubernetes": ["docker", "devops", "container orchestration"],
        "git": ["version control", "github", "gitlab"],
        "agile": ["scrum", "kanban", "project management"]
    }

def _load_learning_resources(self) -> Dict[str, List[Dict]]:
    """Load learning resource database"""
    # In production, this would be loaded from a database
    # Mock data for demonstration
    return {
        "python": [
            {
                "title": "Learn Python Programming",
                "provider": "Codecademy",
                "url": "https://www.codecademy.com/learn/learn-python-3",
                "type": "course",
                "duration_hours": 30,
                "is_free": True,
                "rating": 4.7
            },
            {
                "title": "Python for Everybody",
                "provider": "Coursera",
                "url": "https://www.coursera.org/specializations/python",
                "type": "specialization",
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            "duration_hours": 76,
            "is_free": False,
            "rating": 4.8
        }
    ],
    "javascript": [
        {
            "title": "JavaScript Basics",
            "provider": "MDN Web Docs",
            "url": "https://developer.mozilla.org/en-US/docs/Learn/JavaScript",
            "type": "documentation",
            "duration_hours": 15,
            "is_free": True,
            "rating": 4.9
        },
        {
            "title": "JavaScript: Understanding the Weird Parts",
            "provider": "Udemy",
            "url": "https://www.udemy.com/course/understand-javascript/",
            "type": "course",
            "duration_hours": 12,
            "is_free": False,
            "rating": 4.8
        }
    ],
    "react": [
        {
            "title": "React - The Complete Guide",
            "provider": "Udemy",
            "url": "https://www.udemy.com/course/react-the-complete-guide-incl-redux/",
            "type": "course",
            "duration_hours": 40,
            "is_free": False,
            "rating": 4.7
        },
        {
            "title": "React Documentation",
            "provider": "React.js",
            "url": "https://reactjs.org/docs/getting-started.html",
            "type": "documentation",
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
                "duration_hours": 10,
                "is_free": True,
                "rating": 4.6
            }
        ]
        # Additional resources for other skills would be included here
    }

def analyze_skill_gap(self, input_data: SkillGapInput) -> SkillGapOutput:
    """
    Analyze the skill gap between student skills and target skills

    Args:
        input_data: The SkillGapInput containing student and target information

    Returns:
        A SkillGapOutput with gap analysis and recommendations
    """
    # Initialize result
    result = SkillGapOutput(
        missing_skills=[],
        skill_strengths=[],
        learning_resources=[],
        career_path_alignment=None,
        development_timeline=None
    )

    # Extract student skill names and proficiency levels
    student_skill_dict = {}
    for skill in input_data.student_skills:
        name = skill.get('name', '').lower()
        proficiency = skill.get('proficiency', 0)
        student_skill_dict[name] = proficiency

    # Process target skills
    target_skills_lower = [skill.lower() for skill in input_data.target_skills]

    # Identify missing skills
    for skill in target_skills_lower:
        if skill not in student_skill_dict:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        # Get importance score
        importance = self.skill_importance.get(skill, 0.5)

        # Get related skills
        related_skills = self.skill_graph.get(skill, [])

        # Check if student has any related skills
        related_skill_proficiency = 0
        for related in related_skills:
            if related in student_skill_dict:
                related_skill_proficiency = max(related_skill_proficiency,
                                                student_skill_dict[related])

        # Calculate difficulty (lower if student has related skills)
        difficulty = 1.0 - (related_skill_proficiency / 10) * 0.5

        # Add to missing skills
        result.missing_skills.append({
            'name': skill,
            'importance': importance,
            'difficulty': difficulty,
            'related_skills': related_skills,
            'has_related_skills': related_skill_proficiency > 0
        })

    # Sort missing skills by importance
    result.missing_skills.sort(key=lambda x: x['importance'], reverse=True)

    # Identify skill strengths (skills the student has that align with targets)
    for skill, proficiency in student_skill_dict.items():
        if skill in target_skills_lower and proficiency >= 7:  # High proficiency
            result.skill_strengths.append({
                'name': skill,
                'proficiency': proficiency,
                'importance': self.skill_importance.get(skill, 0.5)
            })

    # Sort skill strengths by proficiency
    result.skill_strengths.sort(key=lambda x: x['proficiency'], reverse=True)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# Recommend learning resources for top missing skills
for missing_skill in result.missing_skills[:5]:  # Top 5 missing skills
    skill_name = missing_skill['name']

    # Find resources for this skill
    resources = self.learning_resources.get(skill_name, [])

    # If no direct resources, check related skills
    if not resources and 'related_skills' in missing_skill:
        for related in missing_skill['related_skills']:
            related_resources = self.learning_resources.get(related, [])
            if related_resources:
                # Tag these as related resources
                for resource in related_resources:
                    resource_copy = resource.copy()
                    resource_copy['is_related'] = True
                    resource_copy['related_to'] = related
                    resources.append(resource_copy)

    # Add resources to recommendations
    for resource in resources:
        result.learning_resources.append({
            'skill_name': skill_name,
            'resource': resource
        })

# Calculate career path alignment if career goal provided
if input_data.career_goal:
    # In a real implementation, this would use a more sophisticated model
    # Simplified calculation for demonstration
    aligned_skills = 0
    total_target_skills = len(target_skills_lower)

    for skill in target_skills_lower:
        if skill in student_skill_dict:
            aligned_skills += 1

    result.career_path_alignment = aligned_skills / total_target_skills if total_target_skills > 0 else 0

# Generate development timeline
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        result.development_timeline = self._generate_development_timeline(result.missing_skills)

        return result

    def _generate_development_timeline(self, missing_skills: List[Dict]) -> Dict:
        """Generate a recommended development timeline"""
        if not missing_skills:
            return {
                'total_weeks': 0,
                'phases': []
            }

        # Calculate an estimated learning time for each skill based on difficulty and importance
        for skill in missing_skills:
            # Base learning time in weeks (1-12 weeks)
            base_time = skill['difficulty'] * 12

            # Adjust based on importance (more important = more time invested)
            adjusted_time = base_time * (0.7 + skill['importance'] * 0.3)

            # Reduce time if student has related skills
            if skill.get('has_related_skills', False):
                adjusted_time *= 0.7

            skill['learning_weeks'] = max(1, round(adjusted_time))

        # Organize skills into phases
        phases = []
        current_phase = []
        current_phase_weeks = 0
        max_phase_weeks = 8  # Maximum weeks per phase

        for skill in missing_skills:
            # If skill would push phase over max duration, start a new phase
            if current_phase_weeks + skill['learning_weeks'] > max_phase_weeks and current_phase:
                phases.append({
                    'skills': current_phase,
                    'duration_weeks': current_phase_weeks
                })
                current_phase = []
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            current_phase_weeks = 0

        current_phase.append({
            'name': skill['name'],
            'weeks': skill['learning_weeks'],
            'importance': skill['importance']
        })
        current_phase_weeks += skill['learning_weeks']

    # Add the last phase if it has skills
    if current_phase:
        phases.append({
            'skills': current_phase,
            'duration_weeks': current_phase_weeks
        })

    # Calculate total timeline duration
    total_weeks = sum(phase['duration_weeks'] for phase in phases)

    return {
        'total_weeks': total_weeks,
        'phases': phases
    }
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# ----------------------------------------------------------------------------------------
# PDF REPORT GENERATION FOR READINESS SCORE, SKILL GAP
# ----------------------------------------------------------------------------------------
# For PDF reports, install a reliable Laravel PDF package:
composer require barryvdh/laravel-dompdf
# Then create well-structured Blade templates for each report type:
// Example Blade template for readiness score report (resources/views/reports/readiness_score.blade.php)
<!DOCTYPE html>
<html>
<head>
    <title>Career Readiness Report</title>
    <style>
        /* Add styling for professional PDF reports */
        body { font-family: Arial, sans-serif; }
        .header { text-align: center; background-color: #003366; color: white; padding: 20px; }
        .section { margin: 15px 0; padding: 10px; border-bottom: 1px solid #eee; }
        .score { font-size: 24px; font-weight: bold; }
        .score-container { text-align: center; padding: 20px; }
        .progress-bar { background-color: #f3f3f3; height: 20px; border-radius: 10px; }
        .progress-value { background-color: #4CAF50; height: 20px; border-radius: 10px; }
        .recommendations { background-color: #f9f9f9; padding: 15px; }
        .recommendation-item { margin-bottom: 10px; }
        .zetheta-promo { background-color: #e6f2ff; padding: 15px; border-left: 4px solid #0066cc; margin:
20px 0; }
        .footer { text-align: center; font-size: 12px; margin-top: 50px; color: #666; }
    </style>
</head>
<body>
    <div class="header">
        <h1>Career Readiness Assessment</h1>
        <p>Generated on {{ $generated_at }}</p>
    </div>

    <div class="score-container">
        <div>Your Overall Readiness Score</div>
        <div class="score">{{ round($data['total_score']) }}%</div>
        <div class="progress-bar">
            <div class="progress-value" style="width: {{ round($data['total_score']) }}%;"></div>
        </div>
    </div>
```

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

```
    <div class="section">
        <h2>Score Breakdown</h2>
        <div>Education: {{ round($data['education_score'] * 100) }}%</div>
        <div>Skills: {{ round($data['skills_score'] * 100) }}%</div>
        <div>Experience: {{ round($data['experience_score'] * 100) }}%</div>
        <div>Projects: {{ round($data['projects_score'] * 100) }}%</div>
        <div>Certifications: {{ round($data['certifications_score'] * 100) }}%</div>
    </div>

    <div class="section recommendations">
        <h2>Recommended Actions</h2>
        @foreach($data['recommendations'] as $recommendation)
            <div class="recommendation-item">
                <h3>{{ $recommendation['title'] }}</h3>
                <p>{{ $recommendation['description'] }}</p>
                @if(isset($recommendation['courses']))
                    <ul>
                        @foreach($recommendation['courses'] as $course)
                            <li>{{ $course['title'] }} - {{ $course['url'] }}</li>
                        @endforeach
                    </ul>
                @endif
            </div>
        @endforeach
    </div>

    <div class="zetheta-promo">
        <h3>Accelerate Your Career with Zetheta</h3>
        <p>Zetheta's internship preparation program provides comprehensive training to address your
specific skill gaps and increase your employment readiness.</p>
        <p>Visit <a href="https://www.zetheta.com">www.zetheta.com</a> to learn more.</p>
    </div>

    <div class="footer">
        <p>This report was generated by the Intern-Employer Matchmaking Platform</p>
    </div>
</body>
</html>
```

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# ------------------------------------------------------------------------------
# CAREER PATH PREDICTION COMPONENT (ADVANCED)
# ------------------------------------------------------------------------------
class CareerPathInput(BaseModel):
    """Input model for career path prediction"""
    student_id: int
    education: List[Dict] = Field(default_factory=list)
    skills: List[Dict] = Field(default_factory=list)
    experience: List[Dict] = Field(default_factory=list)
    projects: List[Dict] = Field(default_factory=list)
    preferences: Dict = Field(default_factory=dict)
    personality_traits: Optional[Dict] = None

class CareerPathOutput(BaseModel):
    """Output model for career path prediction"""
    predicted_paths: List[Dict] = Field(default_factory=list)
    salary_projections: Dict = Field(default_factory=dict)
    skill_recommendations: List[Dict] = Field(default_factory=list)
    industry_outlook: Dict = Field(default_factory=dict)
    confidence_score: float = 0.0

class CareerPredictor:
    """
    Class to predict career paths and salary growth
    """

    def __init__(self):
        """Initialize the career predictor with required models"""
        # Load or initialize models
        self.career_cluster_model = self._load_career_cluster_model()
        self.salary_prediction_model = self._load_salary_prediction_model()
        self.industry_growth_data = self._load_industry_growth_data()

        # Career path transition graph
        self.career_transitions = self._load_career_transitions()

        # Initialize vectorizer for skill processing
        self.skill_vectorizer = TfidfVectorizer(stop_words='english')

    def _load_career_cluster_model(self):
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        """Load the career clustering model"""
        # In production, this would load a pre-trained model
        # For demonstration, we create a simple K-means model
        return KMeans(n_clusters=10, random_state=42)

    def _load_salary_prediction_model(self):
        """Load the salary prediction model"""
        # In production, this would load a pre-trained model
        # For demonstration, we create a simple regression model
        return RandomForestRegressor(n_estimators=100, random_state=42)

    def _load_industry_growth_data(self) -> Dict:
        """Load industry growth projections data"""
        # In production, this would be loaded from a database
        # Mock data for demonstration
        return {
            "software_development": {
                "growth_rate": 0.22,
                "outlook": "excellent",
                "hot_skills": ["cloud computing", "machine learning", "cybersecurity"]
            },
            "data_science": {
                "growth_rate": 0.31,
                "outlook": "excellent",
                "hot_skills": ["machine learning", "python", "big data"]
            },
            "web_development": {
                "growth_rate": 0.13,
                "outlook": "good",
                "hot_skills": ["javascript", "react", "node.js"]
            },
            "mobile_development": {
                "growth_rate": 0.16,
                "outlook": "good",
                "hot_skills": ["react native", "flutter", "swift"]
            },
            "cybersecurity": {
                "growth_rate": 0.33,
                "outlook": "excellent",
                "hot_skills": ["network security", "ethical hacking", "threat intelligence"]
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        },
        "devops": {
            "growth_rate": 0.21,
            "outlook": "excellent",
            "hot_skills": ["kubernetes", "docker", "ci/cd"]
        },
        "ui_ux_design": {
            "growth_rate": 0.15,
            "outlook": "good",
            "hot_skills": ["user research", "figma", "interaction design"]
        }
    }

def _load_career_transitions(self) -> Dict[str, Dict]:
    """Load career transition probability data"""
    # In production, this would be derived from historical data
    # Mock data for demonstration
    return {
        "intern": {
            "next_roles": {
                "junior_developer": 0.6,
                "junior_data_analyst": 0.2,
                "junior_designer": 0.1,
                "technical_support": 0.1
            },
            "avg_months": 6
        },
        "junior_developer": {
            "next_roles": {
                "developer": 0.7,
                "devops_engineer": 0.1,
                "technical_lead": 0.1,
                "product_manager": 0.1
            },
            "avg_months": 24
        },
        "developer": {
            "next_roles": {
                "senior_developer": 0.6,
                "technical_lead": 0.2,
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            "devops_engineer": 0.1,
            "product_manager": 0.1
          },
          "avg_months": 36
        },
        "senior_developer": {
          "next_roles": {
            "technical_lead": 0.4,
            "architect": 0.3,
            "engineering_manager": 0.2,
            "cto": 0.1
          },
          "avg_months": 48
        }
        # Additional roles would be included here
    }

def predict_career_path(self, input_data: CareerPathInput) -> CareerPathOutput:
    """
    Predict career paths based on student profile

    Args:
        input_data: The CareerPathInput containing student information

    Returns:
        A CareerPathOutput with career predictions and recommendations
    """
    # Initialize result
    result = CareerPathOutput(
        predicted_paths=[],
        salary_projections={},
        skill_recommendations=[],
        industry_outlook={},
        confidence_score=0.0
    )

    # Extract features from input data
    features = self._extract_features(input_data)

    # Determine current career stage
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        current_stage = self._determine_career_stage(input_data)

        # Generate career paths
        career_paths = self._generate_career_paths(current_stage, features, 3)  # Generate top 3 paths
        result.predicted_paths = career_paths

        # Calculate salary projections
        result.salary_projections = self._calculate_salary_projections(career_paths, features)

        # Identify skill recommendations
        result.skill_recommendations = self._generate_skill_recommendations(career_paths,
input_data.skills)

        # Provide industry outlook
        result.industry_outlook = self._get_industry_outlook(career_paths)

        # Calculate confidence score
        result.confidence_score = self._calculate_confidence_score(features, career_paths)

        return result

    def _extract_features(self, input_data: CareerPathInput) -> Dict:
        """Extract features from input data for model use"""
        features = {}

        # Education features
        highest_edu_level = 0
        edu_levels = {
            "high school": 1,
            "associate": 2,
            "bachelor": 3,
            "master": 4,
            "phd": 5,
            "doctorate": 5
        }

        for edu in input_data.education:
            degree = edu.get('degree', '').lower()
            for level_name, level_value in edu_levels.items():
                if level_name in degree:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            highest_edu_level = max(highest_edu_level, level_value)

    features['education_level'] = highest_edu_level

    # Field of study
    stem_fields = ['computer science', 'engineering', 'information technology',
                   'mathematics', 'physics', 'data science']

    is_stem = False
    for edu in input_data.education:
        field = edu.get('field_of_study', '').lower()
        if any(stem in field for stem in stem_fields):
            is_stem = True
            break

    features['is_stem'] = 1 if is_stem else 0

    # Experience features
    total_months = 0
    for exp in input_data.experience:
        # Extract duration in months (simplified)
        months = exp.get('duration_months', 0)
        total_months += months

    features['experience_months'] = total_months

    # Skills features (count by category)
    tech_skills = ['python', 'javascript', 'java', 'c++', 'sql', 'aws', 'azure',
                   'docker', 'kubernetes', 'react', 'angular', 'vue', 'node.js']

    data_skills = ['data analysis', 'machine learning', 'statistics', 'pandas',
                   'numpy', 'tensorflow', 'pytorch', 'r', 'tableau', 'power bi']

    design_skills = ['ui design', 'ux design', 'graphic design', 'figma', 'sketch',
                     'adobe', 'illustrator', 'photoshop']

    # Count skills in each category
    tech_count = 0
    data_count = 0
    design_count = 0
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        for skill in input_data.skills:
            skill_name = skill.get('name', '').lower()

            if any(tech in skill_name for tech in tech_skills):
                tech_count += 1

            if any(data in skill_name for data in data_skills):
                data_count += 1

            if any(design in skill_name for design in design_skills):
                design_count += 1

        features['tech_skill_count'] = tech_count
        features['data_skill_count'] = data_count
        features['design_skill_count'] = design_count

        # Project features
        features['project_count'] = len(input_data.projects)

        # Preferences (simplified)
        if input_data.preferences:
            features['prefers_remote'] = 1 if input_data.preferences.get('remote_work', False) else 0
            features['prefers_large_company'] = 1 if input_data.preferences.get('company_size') == 'large' else
0
        else:
            features['prefers_remote'] = 0
            features['prefers_large_company'] = 0

        return features

    def _determine_career_stage(self, input_data: CareerPathInput) -> str:
        """Determine the current career stage of the student"""
        # Calculate total experience
        total_months = 0
        for exp in input_data.experience:
            months = exp.get('duration_months', 0)
            total_months += months

        # Map experience to career stage
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        if total_months == 0:
            return "intern"
        elif total_months < 12:
            return "intern"
        elif total_months < 36:
            return "junior_developer"
        elif total_months < 60:
            return "developer"
        else:
            return "senior_developer"

    def _generate_career_paths(self, current_stage: str, features: Dict, num_paths: int) -> List[Dict]:
        """Generate potential career paths based on current stage and features"""
        paths = []

        # Check if current stage exists in our transition data
        if current_stage not in self.career_transitions:
            # Fallback to intern if stage not found
            current_stage = "intern"

        # Generate multiple paths
        for i in range(num_paths):
            path = []
            current_role = current_stage
            total_months = 0
            max_steps = 5  # Maximum 5 career moves to predict

            # Add current role as starting point
            path.append({
                "role": current_role,
                "timeframe": "Current",
                "probability": 1.0,
                "months_from_now": 0
            })

            # Generate next steps in the path
            for step in range(max_steps):
                if current_role in self.career_transitions:
                    transitions = self.career_transitions[current_role]
                    next_roles = transitions["next_roles"]
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        avg_months = transitions["avg_months"]

        # For different paths, slightly randomize the selection
        # but weight by probability
        roles = list(next_roles.keys())
        probabilities = list(next_roles.values())

        # Adjust probabilities based on features for more personalization
        adjusted_probs = self._adjust_probabilities(probabilities, roles, features, i)

        # Select next role based on adjusted probabilities
        next_role = roles[adjusted_probs.index(max(adjusted_probs))]
        next_prob = next_roles[next_role]

        # Adjust average months based on features
        adjusted_months = avg_months * (0.8 + 0.4 * (i / num_paths))

        # Add to total months
        total_months += adjusted_months

        # Add to path
        path.append({
            "role": next_role,
            "timeframe": f"{int(total_months)} months from now",
            "probability": next_prob,
            "months_from_now": int(total_months)
        })

        # Update current role for next iteration
        current_role = next_role
    else:
        # End path if no transitions available
        break

paths.append({
    "path": path,
    "total_timeframe_months": int(total_months),
    "path_probability": self._calculate_path_probability(path),
    "suitable_industries": self._identify_suitable_industries(path, features)
})
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        # Sort paths by probability
        paths.sort(key=lambda x: x["path_probability"], reverse=True)

        return paths

    def _adjust_probabilities(self, probabilities: List[float], roles: List[str],
                              features: Dict, seed: int) -> List[float]:
        """Adjust transition probabilities based on student features"""
        adjusted = probabilities.copy()

        # Use seed for slight randomization while keeping deterministic
        np.random.seed(seed)
        random_factors = np.random.normal(1.0, 0.1, len(probabilities))

        for i, role in enumerate(roles):
            # Adjust based on tech skills
            if "developer" in role or "engineer" in role:
                adjusted[i] *= 1.0 + (features['tech_skill_count'] * 0.05)

            # Adjust based on data skills
            if "data" in role or "analyst" in role or "scientist" in role:
                adjusted[i] *= 1.0 + (features['data_skill_count'] * 0.05)

            # Adjust based on design skills
            if "design" in role or "ux" in role or "ui" in role:
                adjusted[i] *= 1.0 + (features['design_skill_count'] * 0.05)

            # Adjust based on education
            if "architect" in role or "lead" in role or "manager" in role:
                adjusted[i] *= 1.0 + (features['education_level'] * 0.05)

            # Adjust based on preference matches
            if "remote" in role and features['prefers_remote'] == 1:
                adjusted[i] *= 1.2

            # Add random factor for diversity of paths
            adjusted[i] *= random_factors[i]

        # Normalize to ensure sum = 1
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        total = sum(adjusted)
        return [p / total for p in adjusted]

    def _calculate_path_probability(self, path: List[Dict]) -> float:
        """Calculate the overall probability of a career path"""
        if not path:
            return 0.0

        # Multiply individual step probabilities
        probability = 1.0
        for step in path:
            probability *= step.get("probability", 1.0)

        return probability

    def _identify_suitable_industries(self, path: List[Dict], features: Dict) -> List[str]:
        """Identify suitable industries for a career path"""
        suitable = []

        # Extract terminal role (career goal)
        if path:
            final_role = path[-1]["role"]

            # Map roles to industries
            if "developer" in final_role or "engineer" in final_role:
                suitable.append("software_development")

            if "data" in final_role or "analyst" in final_role or "scientist" in final_role:
                suitable.append("data_science")

            if "web" in final_role:
                suitable.append("web_development")

            if "mobile" in final_role or "ios" in final_role or "android" in final_role:
                suitable.append("mobile_development")

            if "security" in final_role or "cyber" in final_role:
                suitable.append("cybersecurity")

            if "devops" in final_role or "cloud" in final_role or "infrastructure" in final_role:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            suitable.append("devops")

        if "design" in final_role or "ux" in final_role or "ui" in final_role:
            suitable.append("ui_ux_design")

    # If no specific mapping, use skills to determine
    if not suitable:
        if features['tech_skill_count'] > features['data_skill_count'] and features['tech_skill_count'] >
features['design_skill_count']:
            suitable.append("software_development")

        if features['data_skill_count'] > features['tech_skill_count'] and features['data_skill_count'] >
features['design_skill_count']:
            suitable.append("data_science")

        if features['design_skill_count'] > features['tech_skill_count'] and features['design_skill_count'] >
features['data_skill_count']:
            suitable.append("ui_ux_design")

    return suitable

  def _calculate_salary_projections(self, career_paths: List[Dict], features: Dict) -> Dict:
    """Calculate salary projections for career paths"""
    projections = {
        "current_estimate": 0,
        "one_year": 0,
        "three_year": 0,
        "five_year": 0,
        "by_path": []
    }

    # Base salary estimates by role (USD)
    base_salaries = {
        "intern": 50000,
        "junior_developer": 70000,
        "developer": 90000,
        "senior_developer": 120000,
        "technical_lead": 140000,
        "architect": 160000,
        "engineering_manager": 150000,
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    "product_manager": 130000,
    "cto": 200000,
    "junior_data_analyst": 65000,
    "data_analyst": 85000,
    "data_scientist": 110000,
    "junior_designer": 60000,
    "designer": 80000,
    "senior_designer": 110000,
    "devops_engineer": 100000,
    "technical_support": 55000
}

# Education multipliers
edu_multipliers = {
    0: 0.8,  # No formal education
    1: 0.9,  # High school
    2: 0.95, # Associate
    3: 1.0,  # Bachelor's
    4: 1.1,  # Master's
    5: 1.2   # PhD/Doctorate
}

# Get education multiplier
edu_level = features.get('education_level', 3)  # Default to bachelor's
edu_multiplier = edu_multipliers.get(edu_level, 1.0)

# Experience multiplier
exp_months = features.get('experience_months', 0)
exp_years = exp_months / 12
exp_multiplier = min(1.5, 1.0 + exp_years * 0.05)  # 5% per year up to 50%

# Calculate current salary estimate
if not career_paths:
    # Default estimate if no paths available
    projections["current_estimate"] = 60000 * edu_multiplier * exp_multiplier
    return projections

# Get starting role from first path
current_role = career_paths[0]["path"][0]["role"]
base_current = base_salaries.get(current_role, 60000)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
current_estimate = base_current * edu_multiplier * exp_multiplier
projections["current_estimate"] = int(current_estimate)

# Calculate path-specific projections
for path_idx, path_data in enumerate(career_paths):
    path = path_data["path"]
    path_projections = {
        "path_index": path_idx,
        "timeline": []
    }

    # Initial values
    last_salary = current_estimate
    last_months = 0

    for step in path:
        role = step["role"]
        months = step["months_from_now"]

        # Skip current role (already accounted for)
        if months == 0:
            continue

        # Base salary for this role
        base = base_salaries.get(role, 60000)

        # Adjust for education and accumulated experience
        total_exp_years = (exp_years + months / 12)
        step_exp_multiplier = min(1.5, 1.0 + total_exp_years * 0.05)
        salary = int(base * edu_multiplier * step_exp_multiplier)

        # Add to timeline
        path_projections["timeline"].append({
            "role": role,
            "months_from_now": months,
            "estimated_salary": salary
        })

        # Update trackers
        last_salary = salary
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            last_months = months

            # Update aggregate projections at standard time points
            if months <= 12 and projections["one_year"] == 0:
                projections["one_year"] = salary
            elif months > 12 and months <= 36 and projections["three_year"] == 0:
                projections["three_year"] = salary
            elif months > 36 and months <= 60 and projections["five_year"] == 0:
                projections["five_year"] = salary

        projections["by_path"].append(path_projections)

    # Fill any missing standard projections using linear interpolation
    # if not specifically hit by a career move
    if projections["one_year"] == 0:
        projections["one_year"] = int(current_estimate * 1.1)  # 10% growth assumption

    if projections["three_year"] == 0:
        projections["three_year"] = int(current_estimate * 1.25)  # 25% growth assumption

    if projections["five_year"] == 0:
        projections["five_year"] = int(current_estimate * 1.4)  # 40% growth assumption

    return projections

def _generate_skill_recommendations(self, career_paths: List[Dict], current_skills: List[Dict]) ->
List[Dict]:
    """Generate skill recommendations based on career paths"""
    recommendations = []

    if not career_paths:
        return recommendations

    # Extract current skill names
    current_skill_names = [skill.get('name', '').lower() for skill in current_skills]

    # Collect target roles from paths
    target_roles = []
    for path in career_paths:
        for step in path["path"]:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            if step["months_from_now"] > 0 and step["months_from_now"] <= 36:  # Focus on 3-year horizon
                target_roles.append(step["role"])

        # Role to skill mapping (simplified)
        role_skills = {
            "junior_developer": ["javascript", "html", "css", "git", "basic algorithms"],
            "developer": ["react", "node.js", "python", "sql", "restful apis", "testing"],
            "senior_developer": ["system design", "architecture", "mentoring", "advanced algorithms"],
            "technical_lead": ["project management", "team leadership", "architecture", "requirements analysis"],
            "architect": ["distributed systems", "cloud architecture", "performance optimization", "security"],
            "engineering_manager": ["people management", "agile methodologies", "budgeting", "hiring"],
            "devops_engineer": ["docker", "kubernetes", "ci/cd", "infrastructure as code", "cloud platforms"],
            "junior_data_analyst": ["sql", "excel", "data visualization", "statistics"],
            "data_analyst": ["python", "r", "tableau", "statistical analysis", "machine learning basics"],
            "data_scientist": ["machine learning", "deep learning", "nlp", "big data", "research"],
            "junior_designer": ["ui basics", "figma", "design principles", "typography"],
            "designer": ["ux research", "interaction design", "prototyping", "design systems"],
            "product_manager": ["user stories", "market research", "roadmapping", "analytics"]
        }

        # Count role occurrences to prioritize
        role_counts = {}
        for role in target_roles:
            role_counts[role] = role_counts.get(role, 0) + 1

        # Sort roles by frequency
        sorted_roles = sorted(role_counts.items(), key=lambda x: x[1], reverse=True)

        # Collect and prioritize recommended skills
        recommended_skills = {}
        for role, count in sorted_roles:
            if role in role_skills:
                for skill in role_skills[role]:
                    if skill.lower() not in current_skill_names and skill not in recommended_skills:
                        # Calculate importance based on role frequency and position in skills list
                        position_factor = 1.0 - (role_skills[role].index(skill) / len(role_skills[role])) * 0.5
                        importance = count * position_factor
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
                recommended_skills[skill] = {
                    "name": skill,
                    "importance": importance,
                    "target_roles": [role],
                    "timeframe": "short-term" if count > len(target_roles) / 2 else "medium-term"
                }
            elif skill.lower() not in current_skill_names and skill in recommended_skills:
                # Update existing recommendation
                recommended_skills[skill]["target_roles"].append(role)
                recommended_skills[skill]["importance"] += count

    # Convert to list and sort by importance
    recommendations = list(recommended_skills.values())
    recommendations.sort(key=lambda x: x["importance"], reverse=True)

    # Limit to top 10 recommendations
    return recommendations[:10]

def _get_industry_outlook(self, career_paths: List[Dict]) -> Dict:
    """Provide industry outlook information based on career paths"""
    if not career_paths:
        return {}

    # Collect suitable industries from all paths
    all_industries = []
    for path in career_paths:
        all_industries.extend(path.get("suitable_industries", []))

    # Count industry occurrences
    industry_counts = {}
    for industry in all_industries:
        industry_counts[industry] = industry_counts.get(industry, 0) + 1

    # Sort industries by frequency
    sorted_industries = sorted(industry_counts.items(), key=lambda x: x[1], reverse=True)

    # Build outlook data for top industries
    outlook = {}

    for industry, count in sorted_industries[:3]:  # Top 3 industries
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            if industry in self.industry_growth_data:
                industry_data = self.industry_growth_data[industry]

                outlook[industry] = {
                    "relevance_score": count / len(career_paths),
                    "growth_rate": industry_data["growth_rate"],
                    "outlook": industry_data["outlook"],
                    "hot_skills": industry_data["hot_skills"]
                }

        return outlook

    def _calculate_confidence_score(self, features: Dict, career_paths: List[Dict]) -> float:
        """Calculate confidence score for predictions"""
        if not career_paths:
            return 0.3  # Low confidence if no paths generated

        # Base confidence score
        confidence = 0.7

        # Adjust based on available data
        data_completeness = 0.0

        # Check key features
        if features.get('education_level', 0) > 0:
            data_completeness += 0.2

        if features.get('experience_months', 0) > 0:
            data_completeness += 0.3

        if features.get('tech_skill_count', 0) + features.get('data_skill_count', 0) +
features.get('design_skill_count', 0) > 5:
            data_completeness += 0.3

        if features.get('project_count', 0) > 0:
            data_completeness += 0.2

        # Adjust confidence based on data completeness
        confidence = 0.5 + (confidence * data_completeness)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    # Further adjust based on path probabilities
    avg_probability = 0.0
    for path in career_paths:
        avg_probability += path.get("path_probability", 0)

    if career_paths:
        avg_probability /= len(career_paths)
        confidence *= (0.7 + avg_probability * 0.3)

    return min(0.95, confidence)  # Cap at 95% confidence
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# ----------------------------------------
# FAST API IMPLEMENTATION
# ----------------------------------------
app = FastAPI(title="Intern-Employer Matchmaking AI API",
          description="API for AI features of the intern-employer matchmaking platform",
          version="1.0.0")

# Initialize service components
resume_parser = ResumeParser()
readiness_scorer = ReadinessScorer()
skill_gap_analyzer = SkillGapAnalyzer()
career_predictor = CareerPredictor()

@app.post("/api/parse-resume", response_model=ResumeParserModel)
async def parse_resume(file: UploadFile = File(...)):
    """
    Parse a resume file and extract structured information
    """
    try:
        # Read file content
        content = await file.read()

        # Process based on file type
        filename = file.filename.lower()

        if filename.endswith('.pdf'):
            # In production, use a PDF parser library
            # For demonstration, we'll assume plain text extraction
            text = content.decode('utf-8', errors='ignore')
        elif filename.endswith(('.docx', '.doc')):
            # In production, use a Word document parser library
            # For demonstration, we'll assume plain text extraction
            text = content.decode('utf-8', errors='ignore')
        elif filename.endswith(('.txt', '.rtf')):
            text = content.decode('utf-8', errors='ignore')
        else:
            raise HTTPException(status_code=400, detail="Unsupported file format")

        # Parse the text
        result = resume_parser.parse_text(text)
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
        return result

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error parsing resume: {str(e)}")

@app.post("/api/calculate-readiness", response_model=ReadinessScoreOutput)
async def calculate_readiness(input_data: ReadinessScoreInput):
    """
    Calculate job readiness score for a student-job pair
    """
    try:
        result = readiness_scorer.calculate_readiness(input_data)
        return result

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error calculating readiness score: {str(e)}")

@app.post("/api/analyze-skill-gap", response_model=SkillGapOutput)
async def analyze_skill_gap(input_data: SkillGapInput):
    """
    Analyze the skill gap between student skills and target skills
    """
    try:
        result = skill_gap_analyzer.analyze_skill_gap(input_data)
        return result

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error analyzing skill gap: {str(e)}")

@app.post("/api/predict-career-path", response_model=CareerPathOutput)
async def predict_career_path(input_data: CareerPathInput):
    """
    Predict career paths based on student profile
    """
    try:
        result = career_predictor.predict_career_path(input_data)
        return result

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error predicting career path: {str(e)}")
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# Main entry point for running the API directly
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

REACT COMPONENTS FOR FRONTEND (SAMPLE)

```
#React component for the Student Dashboard that efficiently displays job recommendations:
import React, { useEffect, useState } from 'react';
import { useRouter } from 'next/router';
import { useQuery } from 'react-query';
import JobCard from '../components/jobs/JobCard';
import SkillGapWidget from '../components/dashboard/SkillGapWidget';
import ProfileCompletion from '../components/dashboard/ProfileCompletion';
import UpcomingInterviews from '../components/dashboard/UpcomingInterviews';
import { fetchRecommendedJobs, fetchProfileCompleteness, fetchUpcomingInterviews } from
'../services/api';
import LoadingSpinner from '../components/common/LoadingSpinner';
import ErrorMessage from '../components/common/ErrorMessage';

const StudentDashboard = () => {
  const router = useRouter();
  const [activeTab, setActiveTab] = useState('recommended');

  // Fetch recommended jobs with react-query
  const {
    data: recommendedJobs,
    isLoading: isLoadingJobs,
    error: jobsError
  } = useQuery('recommendedJobs', fetchRecommendedJobs);

  // Fetch profile completeness
  const {
    data: profileData,
    isLoading: isLoadingProfile
  } = useQuery('profileCompleteness', fetchProfileCompleteness);

  // Fetch upcoming interviews
  const {
    data: interviewsData,
    isLoading: isLoadingInterviews
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
  } = useQuery('upcomingInterviews', fetchUpcomingInterviews);

  // Calculate profile completeness percentage
  const profileCompleteness = profileData?.percentage || 0;

  return (
    <div className="container mx-auto px-4 py-8">
      <div className="flex flex-col md:flex-row justify-between items-start gap-6">
        {/* Main content area */}
        <div className="w-full md:w-2/3">
          <div className="bg-white rounded-lg shadow-md p-6 mb-6">
            <h1 className="text-2xl font-bold text-gray-800 mb-6">Welcome back!</h1>

            {/* Profile completion alert */}
            {profileCompleteness < 80 && (
              <div className="bg-blue-50 border-l-4 border-blue-500 p-4 mb-6">
                <div className="flex">
                  <div className="flex-shrink-0">
                    <svg className="h-5 w-5 text-blue-500" viewBox="0 0 20 20" fill="currentColor">
                      <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7-4a1 1 0 11-2 0 1 1 0
012 0zM9 9a1 1 0 000 2v3a1 1 0 001 1h1a1 1 0 100-2v-3a1 1 0 00-1-1H9z" clipRule="evenodd" />
                    </svg>
                  </div>
                  <div className="ml-3">
                    <p className="text-sm text-blue-800">
                      Complete your profile to improve your job matches!
                      <button
                        onClick={() => router.push('/profile')}
                        className="font-medium underline ml-1"
                      >
                        Update profile
                      </button>
                    </p>
                  </div>
                </div>
              </div>
            )}

            {/* Tabs navigation */}
            <div className="border-b border-gray-200 mb-6">
              <nav className="-mb-px flex space-x-6">
                <button
                  className={`
                    py-2 px-1 border-b-2 font-medium text-sm
                    ${activeTab === 'recommended'
                      ? 'border-blue-500 text-blue-600'
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            : 'border-transparent text-gray-500 hover:border-gray-300 hover:text-gray-700'}
          `}
          onClick={() => setActiveTab('recommended')}
        >
          Recommended Jobs
        </button>
        <button
          className={`
            py-2 px-1 border-b-2 font-medium text-sm
            ${activeTab === 'applied'
              ? 'border-blue-500 text-blue-600'
              : 'border-transparent text-gray-500 hover:border-gray-300 hover:text-gray-700'}
          `}
          onClick={() => setActiveTab('applied')}
        >
          Applied Jobs
        </button>
        <button
          className={`
            py-2 px-1 border-b-2 font-medium text-sm
            ${activeTab === 'saved'
              ? 'border-blue-500 text-blue-600'
              : 'border-transparent text-gray-500 hover:border-gray-300 hover:text-gray-700'}
          `}
          onClick={() => setActiveTab('saved')}
        >
          Saved Jobs
        </button>
      </nav>
    </div>

    {/* Job listings */}
    {isLoadingJobs ? (
      <LoadingSpinner message="Loading job recommendations..." />
    ) : jobsError ? (
      <ErrorMessage message="Failed to load job recommendations. Please try again later." />
    ) : recommendedJobs && recommendedJobs.length > 0 ? (
      <div className="space-y-4">
        {recommendedJobs.map((job) => (
          <JobCard
            key={job.id}
            job={job}
            matchScore={job.match_score}
            showMatchDetails={true}
          />
        ))}
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            </div>
          ) : (
            <div className="text-center py-8">
              <p className="text-gray-500">No jobs found. Update your profile to get better
recommendations.</p>
              <button
                onClick={() => router.push('/profile')}
                className="mt-4 inline-flex items-center px-4 py-2 border border-transparent text-sm font-
medium rounded-md shadow-sm text-white bg-blue-600 hover:bg-blue-700"
              >
                Update Profile
              </button>
            </div>
          )}

          {/* View more button */}
          {recommendedJobs && recommendedJobs.length > 0 && (
            <div className="mt-6 text-center">
              <button
                onClick={() => router.push('/jobs')}
                className="inline-flex items-center px-4 py-2 border border-gray-300 shadow-sm text-sm
font-medium rounded-md text-gray-700 bg-white hover:bg-gray-50"
              >
                View All Jobs
              </button>
            </div>
          )}
        </div>
      </div>

      {/* Sidebar */}
      <div className="w-full md:w-1/3 space-y-6">
        {/* Profile completion widget */}
        <ProfileCompletion
          percentage={profileCompleteness}
          isLoading={isLoadingProfile}
        />

        {/* Upcoming interviews widget */}
        <UpcomingInterviews
          interviews={interviewsData || []}
          isLoading={isLoadingInterviews}
        />

        {/* Skill gap widget */}
        <SkillGapWidget />
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
      </div>
    </div>
  </div>
 );
};

export default StudentDashboard;
```

#Corresponding JobCard component:

```
import React from 'react';
import Link from 'next/link';
import { format } from 'date-fns';

interface Skill {
 id: number;
 name: string;
}

interface Job {
 id: number;
 title: string;
 company_name: string;
 logo_url: string;
 location: string;
 job_type: string;
 salary_range: string;
 remote_allowed: boolean;
 created_at: string;
 required_skills: Skill[];
 preferred_skills: Skill[];
}

interface JobCardProps {
 job: Job;
 matchScore?: number;
 showMatchDetails?: boolean;
}

const JobCard: React.FC<JobCardProps> = ({ job, matchScore, showMatchDetails = false }) => {
 // Format date
 const postedDate = new Date(job.created_at);
 const formattedDate = format(postedDate, 'MMM dd, yyyy');

 // Determine match score color
 const getMatchScoreColor = (score: number) => {
  if (score >= 80) return 'bg-green-100 text-green-800';
  if (score >= 60) return 'bg-blue-100 text-blue-800';
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    if (score >= 40) return 'bg-yellow-100 text-yellow-800';
    return 'bg-gray-100 text-gray-800';
  };

  return (
    <div className="bg-white border border-gray-200 rounded-lg shadow-sm hover:shadow-md
transition-shadow">
      <div className="p-5">
        <div className="flex justify-between items-start">
          <div className="flex items-center space-x-4">
            {/* Company logo */}
            <div className="flex-shrink-0">
              <div className="w-12 h-12 bg-gray-100 rounded-md flex items-center justify-center overflow-
hidden">
                {job.logo_url ? (
                  <img
                    src={job.logo_url}
                    alt={`${job.company_name} logo`}
                    className="w-full h-full object-contain"
                  />
                ) : (
                  <span className="text-gray-400 text-xl font-bold">
                    {job.company_name.charAt(0)}
                  </span>
                )}
              </div>
            </div>

            {/* Job details */}
            <div>
              <h3 className="text-lg font-semibold text-gray-900">
                <Link href={`/jobs/${job.id}`} className="hover:text-blue-600">
                  {job.title}
                </Link>
              </h3>
              <p className="text-gray-600">{job.company_name}</p>
            </div>
          </div>

          {/* Match score */}
          {matchScore && (
            <div className={`px-2.5 py-1 rounded-full text-xs font-medium
${getMatchScoreColor(matchScore)}`}>
              {matchScore}% Match
            </div>
          )}
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
      </div>

      {/* Job meta information */}
      <div className="mt-4 grid grid-cols-2 gap-2">
        <div className="flex items-center text-sm text-gray-500">
          <svg className="mr-1.5 h-4 w-4 text-gray-400" fill="currentColor" viewBox="0 0 20 20">
            <path fillRule="evenodd" d="M5.05 4.05a7 7 0 119.9 9.9L10 18.9l-4.95-4.95a7 7 0 010-
9.9zM10 11a2 2 0 100-4 2 2 0 000 4z" clipRule="evenodd" />
          </svg>
          {job.location}
          {job.remote_allowed && <span className="ml-1 text-green-600">(Remote)</span>}
        </div>

        <div className="flex items-center text-sm text-gray-500">
          <svg className="mr-1.5 h-4 w-4 text-gray-400" fill="currentColor" viewBox="0 0 20 20">
            <path fillRule="evenodd" d="M6 2a1 1 0 00-1 1v1H4a2 2 0 00-2 2v10a2 2 0 002 2h12a2 2 0
002-2V6a2 2 0 00-2-2h-1V3a1 1 0 10-2 0v1H7V3a1 1 0 00-1-1zm0 5a1 1 0 000 2h8a1 1 0 100-2H6z"
clipRule="evenodd" />
          </svg>
          Posted {formattedDate}
        </div>

        <div className="flex items-center text-sm text-gray-500">
          <svg className="mr-1.5 h-4 w-4 text-gray-400" fill="currentColor" viewBox="0 0 20 20">
            <path fillRule="evenodd" d="M6 6V5a3 3 0 013-3h2a3 3 0 013 3v1h2a2 2 0 012
2v3.57A22.952 22.952 0 0110 13a22.95 22.95 0 01-8-1.43V8a2 2 0 012-2h2zm2-1a1 1 0 011-1h2a1 1 0
011 1v1H8V5zm1 5a1 1 0 011-1h.01a1 1 0 110 2H10a1 1 0 01-1-1z" clipRule="evenodd" />
          </svg>
          {job.job_type}
        </div>

        <div className="flex items-center text-sm text-gray-500">
          <svg className="mr-1.5 h-4 w-4 text-gray-400" fill="currentColor" viewBox="0 0 20 20">
            <path fillRule="evenodd" d="M4 4a2 2 0 00-2 2v4a2 2 0 002 2V6h10a2 2 0 00-2-2H4zm2 6a2 2
0 012-2h8a2 2 0 012 2v4a2 2 0 01-2 2H8a2 2 0 01-2-2v-4zm6 4a2 2 0 100-4 2 2 0 000 4z"
clipRule="evenodd" />
          </svg>
          {job.salary_range || 'Not specified'}
        </div>
      </div>

      {/* Skills */}
      <div className="mt-4">
        <div className="flex flex-wrap gap-2">
          {job.required_skills.slice(0, 3).map((skill) => (
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            <span key={skill.id} className="inline-flex items-center px-2.5 py-0.5 rounded-full text-xs
font-medium bg-blue-100 text-blue-800">
              {skill.name}
            </span>
          ))}
          {job.preferred_skills.slice(0, 2).map((skill) => (
            <span key={skill.id} className="inline-flex items-center px-2.5 py-0.5 rounded-full text-xs
font-medium bg-gray-100 text-gray-800">
              {skill.name}
            </span>
          ))}
          {(job.required_skills.length + job.preferred_skills.length > 5) && (
            <span className="inline-flex items-center px-2.5 py-0.5 rounded-full text-xs font-medium bg-
gray-100 text-gray-500">
              +{job.required_skills.length + job.preferred_skills.length - 5} more
            </span>
          )}
        </div>
      </div>

      {/* Match details */}
      {showMatchDetails && matchScore && matchScore >= 60 && (
        <div className="mt-4 pt-4 border-t border-gray-100">
          <p className="text-sm text-gray-600">
            <span className="font-medium">Why this matches:</span> Your skills in
{job.required_skills.slice(0, 2).map(s => s.name).join(', ')} align with this role's requirements.
          </p>
        </div>
      )}

      {/* Actions */}
      <div className="mt-5 flex justify-between">
        <Link href={`/jobs/${job.id}`}
          className="inline-flex items-center px-3 py-1.5 border border-gray-300 text-xs font-medium
rounded text-gray-700 bg-white hover:bg-gray-50">
          View Details
        </Link>

        <button
          className="inline-flex items-center px-3 py-1.5 border border-transparent text-xs font-medium
rounded shadow-sm text-white bg-blue-600 hover:bg-blue-700"
          onClick={() => {/* Apply function */}}
        >
          Quick Apply
        </button>
      </div>
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    </div>
   </div>
  );
};

export default JobCard;
```

CI/CD PIPELINE WITH GITHUB ACTIONS (SAMPLE)

#GitHub Actions workflow file (.github/workflows/deploy.yml) that implements a complete CI/CD pipeline at zero cost:

```
name: Deploy Intern-Employer Platform

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  # Backend tests and linting
  test-backend:
    runs-on: ubuntu-latest

    services:
      # Use PostgreSQL service container for tests
      postgres:
        image: postgres:13
        env:
          POSTGRES_USER: postgres
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: testing
        ports:
          - 5432:5432
        # Health check to ensure database is ready
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
  steps:
  - uses: actions/checkout@v3

  - name: Set up PHP
    uses: shivammathur/setup-php@v2
    with:
      php-version: '8.1'
      extensions: mbstring, zip, pdo, sqlite, pdo_sqlite, pgsql, pdo_pgsql
      coverage: xdebug

  - name: Copy .env
    run: |
      cd backend
      cp .env.example .env
      php artisan key:generate

  - name: Install Composer dependencies
    run: |
      cd backend
      composer install --prefer-dist --no-progress

  - name: Run Laravel tests
    env:
      DB_CONNECTION: pgsql
      DB_HOST: localhost
      DB_PORT: 5432
      DB_DATABASE: testing
      DB_USERNAME: postgres
      DB_PASSWORD: postgres
    run: |
      cd backend
      php artisan migrate:fresh --seed
      php artisan test

  - name: Run Laravel linting
    run: |
      cd backend
      composer require --dev friendsofphp/php-cs-fixer
      vendor/bin/php-cs-fixer fix --dry-run --diff

# Frontend tests and linting
test-frontend:
  runs-on: ubuntu-latest

  steps:
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    - uses: actions/checkout@v3

    - name: Set up Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '16'
        cache: 'npm'
        cache-dependency-path: frontend/package-lock.json

    - name: Install dependencies
      run: |
        cd frontend
        npm ci

    - name: Run ESLint
      run: |
        cd frontend
        npm run lint

    - name: Run frontend tests
      run: |
        cd frontend
        npm test

  # AI Services tests and linting
  test-ai-services:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.9'
        cache: 'pip'

    - name: Install dependencies
      run: |
        cd ai-services
        python -m pip install --upgrade pip
        pip install -r requirements.txt
        pip install pytest flake8

    - name: Run flake8
      run: |
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
      cd ai-services
      flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics

  - name: Run pytest
    run: |
      cd ai-services
      pytest

# Deploy backend to InfinityFree
deploy-backend:
  needs: [test-backend, test-frontend, test-ai-services]
  if: github.ref == 'refs/heads/main' && github.event_name == 'push'
  runs-on: ubuntu-latest

  steps:
  - uses: actions/checkout@v3

  - name: Set up PHP
    uses: shivammathur/setup-php@v2
    with:
      php-version: '8.1'
      extensions: mbstring, zip, pdo, sqlite, pdo_sqlite, pgsql, pdo_pgsql

  - name: Install Composer dependencies
    run: |
      cd backend
      composer install --prefer-dist --no-dev --optimize-autoloader

  - name: Copy .env and optimize Laravel
    run: |
      cd backend
      cp .env.example .env
      php artisan key:generate
      php artisan config:cache
      php artisan route:cache
      php artisan view:cache

  - name: Prepare deployment archive
    run: |
      cd backend
      # Create .htaccess for InfinityFree
      echo "<IfModule mod_rewrite.c>
          RewriteEngine On
          RewriteRule ^(.*)$ public/$1 [L]
      </IfModule>" > .htaccess
      # Create a deployment archive
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
      zip -r deploy.zip .

  - name: Deploy to InfinityFree
    uses: SamKirkland/FTP-Deploy-Action@4.3.0
    with:
      server: ftpupload.net
      username: ${{ secrets.FTP_USERNAME }}
      password: ${{ secrets.FTP_PASSWORD }}
      local-dir: ./backend/
      server-dir: /htdocs/

# Deploy frontend to Vercel
deploy-frontend:
  needs: [test-frontend]
  if: github.ref == 'refs/heads/main' && github.event_name == 'push'
  runs-on: ubuntu-latest

  steps:
  - uses: actions/checkout@v3

  - name: Set up Node.js
    uses: actions/setup-node@v3
    with:
      node-version: '16'

  - name: Install Vercel CLI
    run: npm install -g vercel

  - name: Deploy to Vercel
    run: |
      cd frontend
      vercel deploy --prod --token ${{ secrets.VERCEL_TOKEN }}

# Deploy AI services to Deta Space
deploy-ai-services:
  needs: [test-ai-services]
  if: github.ref == 'refs/heads/main' && github.event_name == 'push'
  runs-on: ubuntu-latest

  steps:
  - uses: actions/checkout@v3

  - name: Set up Python
    uses: actions/setup-python@v4
    with:
      python-version: '3.9'
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
  - name: Install Deta CLI
    run: |
      curl -fsSL https://get.deta.dev/cli.sh | sh

  - name: Deploy to Deta Space
    run: |
      cd ai-services
      ~/.deta/bin/deta login --api-key ${{ secrets.DETA_TOKEN }}
      ~/.deta/bin/deta deploy
```

DATABASE ARCHITECTURE AND SCHEMA

Database Models and Migrations

User model and migration for Laravel:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    protected $fillable = [
        'email',
        'password',
        'first_name',
        'last_name',
        'phone',
        'user_type',
        'status',
        'email_verified',
        'profile_image_url',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
        'reset_token',
        'reset_token_expiry',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
        'password' => 'hashed',
        'email_verified' => 'boolean',
        'created_at' => 'datetime',
        'updated_at' => 'datetime',
        'reset_token_expiry' => 'datetime',
    ];

    /**
     * Get the student profile associated with the user.
     */
    public function student()
    {
        return $this->hasOne(Student::class);
    }
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
/**
 * Get the employer profile associated with the user.
 */
public function employer()
{
    return $this->hasOne(Employer::class);
}

/**
 * Get the admin profile associated with the user.
 */
public function admin()
{
    return $this->hasOne(Admin::class);
}

/**
 * Determine if the user is a student.
 *
 * @return bool
 */
public function isStudent()
{
    return $this->user_type === 'student';
}

/**
 * Determine if the user is an employer.
 *
 * @return bool
 */
public function isEmployer()
{
    return $this->user_type === 'employer';
}

/**
 * Determine if the user is an admin.
 *
 * @return bool
 */
public function isAdmin()
{
    return $this->user_type === 'admin';
}
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
/**
 * Determine if the user is active.
 *
 * @return bool
 */
public function isActive()
{
    return $this->status === 'active';
}

/**
 * Get the user's profile based on user type.
 *
 * @return \Illuminate\Database\Eloquent\Model|null
 */
public function profile()
{
    if ($this->isStudent()) {
        return $this->student;
    } elseif ($this->isEmployer()) {
        return $this->employer;
    } elseif ($this->isAdmin()) {
        return $this->admin;
    }

    return null;
}

/**
 * Get all messages sent by the user.
 */
public function sentMessages()
{
    return $this->hasMany(Message::class, 'sender_id');
}

/**
 * Get all messages received by the user.
 */
public function receivedMessages()
{
    return $this->hasMany(Message::class, 'receiver_id');
}

/**
 * Get all notifications for the user.
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
     */
    public function notifications()
    {
        return $this->hasMany(Notification::class);
    }

    /**
     * Get all conversations that the user participates in.
     */
    public function conversations()
    {
        return $this->belongsToMany(Conversation::class, 'conversation_participants')
                    ->withPivot('joined_at', 'is_active')
                    ->withTimestamps();
    }
}
```

And here's the migration file for the User model:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('email')->unique();
            $table->string('password');
            $table->string('first_name')->nullable();
            $table->string('last_name')->nullable();
            $table->string('phone')->nullable();
            $table->enum('user_type', ['student', 'employer', 'admin'])->default('student');
            $table->enum('status', ['active', 'inactive', 'suspended', 'pending'])->default('active');
            $table->boolean('email_verified')->default(false);
            $table->string('profile_image_url')->nullable();
            $table->string('reset_token')->nullable();
            $table->timestamp('reset_token_expiry')->nullable();
            $table->rememberToken();
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
            $table->timestamps();

            // Index for performance
            $table->index('email');
            $table->index('user_type');
            $table->index('status');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('users');
    }
};
```

Database Architecture and Schema

```
erDiagram
    User {
        int id PK
        string email
        string password_hash
        string first_name
        string last_name
        string phone
        datetime created_at
        datetime updated_at
        enum user_type
        enum status
        boolean email_verified
        string profile_image_url
        string reset_token
        datetime reset_token_expiry
    }

    Student {
        int id PK
        int user_id FK
        string university
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    string degree
    string major
    int graduation_year
    jsonb skills
    text bio
    float gpa
    jsonb preferences
    jsonb personality_assessment
    jsonb career_objectives
    int readiness_score
    jsonb financial_fitness
    datetime last_active
}

Employer {
    int id PK
    int user_id FK
    string company_name
    string industry
    string company_size
    text company_description
    string website
    string logo_url
    jsonb company_culture
    jsonb tech_stack
    boolean verified
    datetime last_active
}

Resume {
    int id PK
    int student_id FK
    string file_url
    datetime uploaded_at
    jsonb parsed_data
    boolean is_current
    int parse_status
    text parse_error
}
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
Education {
   int id PK
   int student_id FK
   string institution
   string degree
   string field_of_study
   date start_date
   date end_date
   float gpa
   text description
   boolean is_current
}

Experience {
   int id PK
   int student_id FK
   string company
   string position
   date start_date
   date end_date
   text description
   string location
   boolean is_current
   string reference_name
   string reference_contact
}

Project {
   int id PK
   int student_id FK
   string title
   text description
   string url
   jsonb technologies
   date start_date
   date end_date
   string image_url
}

Certification {
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    int id PK
    int student_id FK
    string name
    string issuing_organization
    date issue_date
    date expiry_date
    string credential_id
    string credential_url
    boolean has_expiry
}

JobPosting {
    int id PK
    int employer_id FK
    string title
    text description
    jsonb required_skills
    jsonb preferred_skills
    string location
    enum job_type
    date start_date
    date end_date
    string salary_range
    enum experience_level
    boolean remote_allowed
    datetime created_at
    datetime updated_at
    enum status
    int views_count
    int applications_count
    int positions_available
}

Application {
    int id PK
    int student_id FK
    int job_posting_id FK
    datetime applied_at
    enum status
    text cover_letter
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    jsonb additional_documents
    float matching_score
    jsonb interview_results
    text employer_notes
    datetime last_status_change
}

SkillTaxonomy {
    int id PK
    string name
    string category
    string description
    int parent_id FK
    int level
}

StudentSkill {
    int id PK
    int student_id FK
    int skill_id FK
    int proficiency_level
    boolean is_verified
    jsonb endorsements
}

Message {
    int id PK
    int sender_id FK
    int receiver_id FK
    text content
    datetime sent_at
    boolean is_read
    datetime read_at
    int conversation_id FK
}

Conversation {
    int id PK
    string title
    datetime created_at
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    datetime last_message_at
}

ConversationParticipant {
    int id PK
    int conversation_id FK
    int user_id FK
    datetime joined_at
    boolean is_active
}

Notification {
    int id PK
    int user_id FK
    string title
    text content
    string link
    enum type
    boolean is_read
    datetime created_at
    datetime read_at
}

Interview {
    int id PK
    int application_id FK
    datetime scheduled_at
    enum status
    enum interview_type
    text location
    text meeting_link
    jsonb interviewer_details
    text notes
    jsonb evaluation_results
}

InterviewQuestion {
    int id PK
    int employer_id FK
    string question
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    enum category
    enum difficulty
    text expected_answer
    boolean is_technical
    boolean is_active
}

InterviewResponse {
    int id PK
    int interview_id FK
    int question_id FK
    text response
    string video_url
    jsonb ai_analysis
    int score
    text feedback
}

LearningResource {
    int id PK
    string title
    text description
    string url
    string provider
    enum resource_type
    jsonb skills_covered
    int estimated_hours
    float rating
    int reviews_count
    boolean is_free
    string price
}

StudentLearningPath {
    int id PK
    int student_id FK
    string title
    text description
    datetime created_at
    enum status
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    float completion_percentage
}

LearningPathItem {
    int id PK
    int learning_path_id FK
    int resource_id FK
    int order
    enum status
    datetime completed_at
    int user_rating
    text user_notes
}

CareerPathPrediction {
    int id PK
    int student_id FK
    jsonb predicted_paths
    jsonb salary_predictions
    datetime generated_at
    float confidence_score
    jsonb skill_recommendations
}

SkillGapAnalysis {
    int id PK
    int student_id FK
    int job_posting_id FK
    jsonb missing_skills
    jsonb skill_strengths
    datetime generated_at
    jsonb learning_recommendations
}

AdminUser {
    int id PK
    int user_id FK
    enum role
    jsonb permissions
    datetime last_login
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
}

SystemLog {
    int id PK
    datetime timestamp
    enum log_level
    string component
    text message
    jsonb context_data
    string ip_address
    int user_id FK
}

AIModel {
    int id PK
    string name
    string version
    enum model_type
    datetime trained_at
    float accuracy
    text description
    string file_path
    enum status
}

AIModelEvaluation {
    int id PK
    int model_id FK
    datetime evaluation_date
    float accuracy
    float precision
    float recall
    float f1_score
    jsonb confusion_matrix
    int sample_size
    text notes
}

User ||--o{ Student : "has_profile"
User ||--o{ Employer : "has_profile"
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
User ||--o{ AdminUser : "has_role"
Student ||--o{ Resume : "uploads"
Student ||--o{ Education : "has"
Student ||--o{ Experience : "has"
Student ||--o{ Project : "creates"
Student ||--o{ Certification : "earns"
Student ||--o{ StudentSkill : "possesses"
Student ||--o{ Application : "submits"
Student ||--o{ StudentLearningPath : "follows"
Student ||--o{ CareerPathPrediction : "receives"
Employer ||--o{ JobPosting : "creates"
JobPosting ||--o{ Application : "receives"
SkillTaxonomy ||--o{ StudentSkill : "categorizes"
SkillTaxonomy ||--o{ SkillTaxonomy : "has_parent"
Application ||--o{ Interview : "schedules"
Application ||--o{ SkillGapAnalysis : "generates"
Interview ||--o{ InterviewResponse : "collects"
InterviewQuestion ||--o{ InterviewResponse : "asks"
Employer ||--o{ InterviewQuestion : "creates"
StudentLearningPath ||--o{ LearningPathItem : "contains"
LearningResource ||--o{ LearningPathItem : "used_in"
User ||--o{ Message : "sends"
User ||--o{ Message : "receives"
Conversation ||--o{ Message : "contains"
Conversation ||--o{ ConversationParticipant : "includes"
User ||--o{ ConversationParticipant : "joins"
User ||--o{ Notification : "receives"
User ||--o{ SystemLog : "generates"
AIModel ||--o{ AIModelEvaluation : "evaluated_by"
```

System Architecture for Intern-Employer Matching

```
flowchart TB
    subgraph "Client Layer"
        WebApp["Web Application\n(Next.js)"]
        Mobile["Mobile View\n(Responsive)"]
    end

    subgraph "API Gateway"
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    APIGateway["API Gateway\n(Express.js)"]
  end

  subgraph "Core Services"
    AuthService["Authentication Service\n(Laravel)"]
    UserService["User Service\n(Laravel)"]
    MessageService["Messaging Service\n(Laravel)"]
    JobService["Job Service\n(Laravel)"]
    MatchingService["Matching Service\n(Laravel)"]
    NotificationService["Notification Service\n(Laravel)"]
  end

  subgraph "AI Services"
    ResumeParser["Resume Parser\n(FastAPI)"]
    SkillGapAnalyzer["Skill Gap Analyzer\n(FastAPI)"]
    CareerPredictor["Career Predictor\n(FastAPI)"]
    InterviewBot["Interview Bot\n(FastAPI)"]
    ReadinessScorer["Readiness Scorer\n(FastAPI)"]
    RecommendationEngine["Recommendation Engine\n(FastAPI)"]
  end

  subgraph "Data Layer"
    PostgreSQL["PostgreSQL\n(Structured Data)"]
    MongoDB["MongoDB\n(Unstructured Data)"]
    Redis["Redis\n(Cache & Real-time)"]
    VectorDB["Vector Database\n(Pinecone/ChromaDB)"]
  end

  subgraph "Infrastructure Services"
    FileStorage["File Storage\n(AWS S3/Local)"]
    EmailService["Email Service\n(SendGrid)"]
    BackgroundJobs["Background Jobs\n(Laravel Queue)"]
    MLPipeline["ML Pipeline\n(Training & Evaluation)"]
  end

  subgraph "Monitoring & Analytics"
    Logging["Logging\n(ELK Stack)"]
    Metrics["Metrics\n(Prometheus/Grafana)"]
    ErrorTracking["Error Tracking\n(Sentry)"]
  end
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

WebApp <--> APIGateway
Mobile <--> APIGateway

APIGateway <--> AuthService
APIGateway <--> UserService
APIGateway <--> MessageService
APIGateway <--> JobService
APIGateway <--> MatchingService
APIGateway <--> NotificationService

AuthService <--> PostgreSQL
UserService <--> PostgreSQL
UserService <--> MongoDB
MessageService <--> PostgreSQL
MessageService <--> Redis
JobService <--> PostgreSQL
MatchingService <--> PostgreSQL
MatchingService <--> Redis
NotificationService <--> PostgreSQL
NotificationService <--> Redis

UserService <--> ResumeParser
JobService <--> ResumeParser
MatchingService <--> RecommendationEngine
MatchingService <--> ReadinessScorer
MatchingService <--> SkillGapAnalyzer
JobService <--> CareerPredictor
JobService <--> InterviewBot

ResumeParser <--> VectorDB
ResumeParser <--> MongoDB
SkillGapAnalyzer <--> PostgreSQL
SkillGapAnalyzer <--> VectorDB
CareerPredictor <--> PostgreSQL
CareerPredictor <--> VectorDB
InterviewBot <--> MongoDB
InterviewBot <--> VectorDB
ReadinessScorer <--> PostgreSQL
ReadinessScorer <--> VectorDB

Email id:avani@zetheta.com
Contact number: +91-91362 49369

RecommendationEngine <--> PostgreSQL
RecommendationEngine <--> VectorDB
RecommendationEngine <--> Redis

UserService <--> FileStorage
MessageService <--> FileStorage
JobService <--> FileStorage
UserService <--> EmailService
NotificationService <--> EmailService
JobService <--> BackgroundJobs
MatchingService <--> BackgroundJobs
ResumeParser <--> MLPipeline
SkillGapAnalyzer <--> MLPipeline
CareerPredictor <--> MLPipeline
ReadinessScorer <--> MLPipeline
RecommendationEngine <--> MLPipeline

AuthService --> Logging
UserService --> Logging
MessageService --> Logging
JobService --> Logging
MatchingService --> Logging
NotificationService --> Logging
ResumeParser --> Logging
SkillGapAnalyzer --> Logging
CareerPredictor --> Logging
InterviewBot --> Logging
ReadinessScorer --> Logging
RecommendationEngine --> Logging

AuthService --> Metrics
UserService --> Metrics
MessageService --> Metrics
JobService --> Metrics
MatchingService --> Metrics
NotificationService --> Metrics
ResumeParser --> Metrics
SkillGapAnalyzer --> Metrics
CareerPredictor --> Metrics
InterviewBot --> Metrics

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ReadinessScorer --> Metrics
RecommendationEngine --> Metrics

AuthService --> ErrorTracking
UserService --> ErrorTracking
MessageService --> ErrorTracking
JobService --> ErrorTracking
MatchingService --> ErrorTracking
NotificationService --> ErrorTracking
ResumeParser --> ErrorTracking
SkillGapAnalyzer --> ErrorTracking
CareerPredictor --> ErrorTracking
InterviewBot --> ErrorTracking
ReadinessScorer --> ErrorTracking
RecommendationEngine --> ErrorTracking

INFRASTRUCTURE SETUP GUIDE

Frontend (Next.js) Deployment on Vercel

```
# Step 1: Prepare Next.js for production
cd frontend
npm run build

# Step 2: Configure Vercel settings
# Create vercel.json configuration
cat > vercel.json << EOF
{
  "version": 2,
  "builds": [
    {
      "src": "package.json",
      "use": "@vercel/next"
    }
  ],
  "routes": [
    {
      "src": "/api/(.*)",
      "dest": "/api/$1"
    },
    {
      "src": "/(.*)",
      "dest": "/$1"
    }
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
  ],
  "env": {
    "NEXT_PUBLIC_API_URL": "https://your-backend-url.com/api"
  }
}
EOF

# Step 3: Deploy to Vercel
# Install Vercel CLI
npm i -g vercel

# Deploy
vercel --prod
```

Backend (Laravel) Deployment on InfinityFree

```
# Step 1: Prepare Laravel for production
cd backend
composer install --optimize-autoloader --no-dev
php artisan config:cache
php artisan route:cache
php artisan view:cache

# Step 2: Configure .htaccess for shared hosting
cat > public/.htaccess << EOF
<IfModule mod_rewrite.c>
    <IfModule mod_negotiation.c>
        Options -MultiViews -Indexes
    </IfModule>

    RewriteEngine On

    # Handle Authorization Header
    RewriteCond %{HTTP:Authorization} .
    RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]

    # Redirect Trailing Slashes If Not A Folder...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_URI} (.+)/$
    RewriteRule ^ %1 [L,R=301]

    # Send Requests To Front Controller...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
EOF
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
# Step 3: Create a zip for upload
zip -r backend.zip .

# Step 4: Upload to InfinityFree via FTP
# (Use a tool like FileZilla to upload)

# Step 5: Configure environment
# Create proper .env file on the server with:
# - Database connection to Supabase
# - File paths for local storage
# - Proper app URL
```

AI Services (FastAPI) Deployment on Deta.sh (now Deta Space)

```
# Step 1: Prepare FastAPI for deployment
cd ai-services
pip install deta
pip freeze > requirements.txt

# Step 2: Configure Deta Space settings
# Create a Spacefile
cat > Spacefile << EOF
v: 0
micros:
  - name: resume-parser
    src: ./resume_parser
    engine: python3.9
    run: uvicorn app:app
    public: true
    primary: true
    presets:
      api: true
  - name: skill-gap
    src: ./skill_gap
    engine: python3.9
    run: uvicorn app:app
    public: true
    presets:
      api: true
EOF

# Step 3: Deploy to Deta Space
deta new
deta deploy
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

Database Setup on Supabase

```
-- Step 1: Setup Schema
-- Users Table
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    phone VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_type VARCHAR(20) NOT NULL CHECK (user_type IN ('student', 'employer', 'admin')),
    status VARCHAR(20) DEFAULT 'active',
    email_verified BOOLEAN DEFAULT FALSE,
    profile_image_url TEXT,
    reset_token VARCHAR(100),
    reset_token_expiry TIMESTAMP
);

-- Students Table
CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    university VARCHAR(255),
    degree VARCHAR(100),
    major VARCHAR(100),
    graduation_year INTEGER,
    skills JSONB DEFAULT '[]',
    bio TEXT,
    gpa FLOAT,
    preferences JSONB DEFAULT '{}',
    personality_assessment JSONB DEFAULT '{}',
    career_objectives JSONB DEFAULT '{}',
    readiness_score INTEGER,
    last_active TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Employers Table
CREATE TABLE employers (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    company_name VARCHAR(255) NOT NULL,
    industry VARCHAR(100),
    company_size VARCHAR(50),
    company_description TEXT,
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    website VARCHAR(255),
    logo_url TEXT,
    company_culture JSONB DEFAULT '{}',
    tech_stack JSONB DEFAULT '[]',
    verified BOOLEAN DEFAULT FALSE,
    last_active TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Job Postings Table
CREATE TABLE job_postings (
    id SERIAL PRIMARY KEY,
    employer_id INTEGER NOT NULL REFERENCES employers(id) ON DELETE CASCADE,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    required_skills JSONB DEFAULT '[]',
    preferred_skills JSONB DEFAULT '[]',
    location VARCHAR(255),
    job_type VARCHAR(50),
    start_date DATE,
    end_date DATE,
    salary_range VARCHAR(100),
    experience_level VARCHAR(50),
    remote_allowed BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) DEFAULT 'active',
    views_count INTEGER DEFAULT 0,
    applications_count INTEGER DEFAULT 0,
    positions_available INTEGER DEFAULT 1
);

-- Skills Table
CREATE TABLE skills (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    category VARCHAR(50),
    description TEXT
);

-- Student Skills Table
CREATE TABLE student_skills (
    id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE CASCADE,
    skill_id INTEGER NOT NULL REFERENCES skills(id) ON DELETE CASCADE,
    proficiency_level INTEGER CHECK (proficiency_level BETWEEN 1 AND 10),
    is_verified BOOLEAN DEFAULT FALSE,
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
    UNIQUE(student_id, skill_id)
);

-- Job Skills Table
CREATE TABLE job_skills (
    id SERIAL PRIMARY KEY,
    job_posting_id INTEGER NOT NULL REFERENCES job_postings(id) ON DELETE CASCADE,
    skill_id INTEGER NOT NULL REFERENCES skills(id) ON DELETE CASCADE,
    type VARCHAR(20) CHECK (type IN ('required', 'preferred')),
    UNIQUE(job_posting_id, skill_id, type)
);

-- Applications Table
CREATE TABLE applications (
    id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(id) ON DELETE CASCADE,
    job_posting_id INTEGER NOT NULL REFERENCES job_postings(id) ON DELETE CASCADE,
    applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(50) DEFAULT 'pending',
    cover_letter TEXT,
    additional_documents JSONB DEFAULT '[]',
    matching_score FLOAT,
    interview_results JSONB DEFAULT '{}',
    employer_notes TEXT,
    last_status_change TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(student_id, job_posting_id)
);

-- Step 2: Create RLS Policies for security
-- Example policy for users table
CREATE POLICY "Users can view their own data" ON users
    FOR SELECT USING (auth.uid() = id);

CREATE POLICY "Users can update their own data" ON users
    FOR UPDATE USING (auth.uid() = id);

-- Step 3: Enable Row Level Security
ALTER TABLE users ENABLE ROW LEVEL SECURITY;
ALTER TABLE students ENABLE ROW LEVEL SECURITY;
ALTER TABLE employers ENABLE ROW LEVEL SECURITY;
ALTER TABLE job_postings ENABLE ROW LEVEL SECURITY;
ALTER TABLE applications ENABLE ROW LEVEL SECURITY;

-- Step 4: Create pgvector extension for AI features if needed
CREATE EXTENSION IF NOT EXISTS vector;
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

```
-- Create a table for skill embeddings
CREATE TABLE skill_embeddings (
    skill_id INTEGER REFERENCES skills(id) ON DELETE CASCADE,
    embedding vector(384),
    PRIMARY KEY (skill_id)
);
```

Project Setup and Environment Configuration

```
# Clone the repository
git clone https://github.com/your-org/intern-employer-platform.git
cd intern-employer-platform

# Backend setup (Laravel)
cd backend
composer install
cp .env.example .env
# Edit .env with your database credentials
php artisan key:generate
php artisan migrate
php artisan db:seed
php artisan serve

# Frontend setup (Next.js)
cd ../frontend
npm install
cp .env.example .env.local
# Edit .env.local with your API URL
npm run dev

# AI Services setup
cd ../ai-services
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install -r requirements.txt
cd resume_parser
python -m uvicorn app:app --reload
```


Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

Project Structure and Architecture

```
intern-employer-platform/
├── backend/                    # Laravel API and main application
│   ├── app/
│   │   ├── Http/Controllers/   # API controllers
│   │   ├── Models/             # Eloquent models
│   │   ├── Services/           # Business logic services
│   │   └── Jobs/               # Background jobs
│   ├── config/                 # Configuration files
│   ├── database/               # Migrations and seeders
│   ├── routes/                 # API route definitions
│   └── tests/                  # Test suite
├── frontend/                   # Next.js frontend
│   ├── components/             # Reusable UI components
│   │   ├── common/             # Shared components
│   │   ├── students/           # Student-specific components
│   │   └── employers/          # Employer-specific components
│   ├── contexts/               # React context providers
│   ├── hooks/                  # Custom React hooks
│   ├── pages/                  # Next.js pages
│   ├── public/                 # Static assets
│   ├── services/               # API service connectors
│   └── styles/                 # Global styles and Tailwind config
├── ai-services/                # FastAPI microservices
│   ├── resume_parser/          # Resume parsing service
│   ├── skill_gap_analyzer/     # Skill gap analysis service
│   ├── career_predictor/       # Career prediction service
│   ├── readiness_scorer/       # Readiness scoring service
│   └── interview_bot/          # Interview bot service
└── docs/                       # Documentation
```

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369

ZeTheta Algorithms Private Limited
CIN No. U62012MH2023PTC410415
Registered Office Address: B3 Grd Flr Silver Valley 1 CHS Ltd
Vakola Bridge CSM Marg, Santacruz East Mumbai - 400055
Maharashtra, India.

Development Workflow

1. **Feature Development Process**:
   - Create a feature branch from develop branch
   - Implement the feature with appropriate tests
   - Create a pull request to merge back to develop
   - Have the code reviewed by at least one other team member
   - Merge to develop once approved
2. **Testing Requirements**:
   - Write unit tests for all critical business logic
   - Create feature tests for key user flows
   - Test on multiple browsers and screen sizes
   - Maintain at least 70% code coverage
3. **Coding Standards**:
   - Follow PSR-12 for PHP/Laravel
   - Use ESLint/Prettier for JavaScript/TypeScript
   - Use PEP 8 for Python code
   - Document all functions and methods
   - Create meaningful commit messages
4. **Critical Features to Implement**:
   - User management (students, employers)
   - Profile creation and management
   - Job posting and simple search
   - Basic skills-based matching
   - Readiness score calculation
   - Skill gap analysis
   - PDF report generation
5. **Features to Defer**:
   - Interview bot
   - Career prediction (if struggling on time then use simplified version)
   - Real-time messaging (if struggling on time then use email notifications instead)
   - Advanced analytics

The main focus is on PDF report generation with Zetheta recommendations that will ensure students receive actionable insights to improve their employability.

PRIVATE AND CONFIDENTIAL – TOP SECRET DOCUMENT

Email id:avani@zetheta.com
Contact number: +91-91362 49369