**Main Function and Implementation   
Frontend Module**

1. **Web Design & User Interface (UI/UX Design & Frontend UI)**
   1. **Requirements Research & Information Architecture Design**
      * **Collect user requirements and competitive research.**
      * **Create user journey maps and wireframes.**
      * **Define different user roles (job seekers, HR, training managers, etc.) and their main functional modules.**
   2. **Visual & UI Design**
      * **Produce high-fidelity mockups using tools such as Figma or Adobe XD.**
      * **Design brand visual elements (logo, color schemes, fonts, icons, etc.).**
      * **Determine page layout, navigation structure, and content hierarchy.**
   3. **Frontend Platform & Tool Configuration**
      * **Choose a frontend framework (e.g., React).**
      * **Configure the frontend project with AWS Amplify and integrate with AWS services such as Cognito, API Gateway, S3, and CloudFront.**
      * **Host frontend code (using AWS CodeCommit or GitHub) and set up a CI/CD pipeline (AWS CodePipeline).**
   4. **Core Page Development & Integration**
      * **Develop pages for registration, login, job seeker dashboard, HR enterprise portal, Capsule simulation, and task display.**
      * **Integrate AWS Cognito login components and use Amplify Auth for registration, login, password resets, etc.**
      * **Implement data interaction between the frontend and backend APIs.**
   5. **Responsive Design & Cross-Platform Adaptation**
      * **Develop responsive layouts to ensure a good user experience on desktop, tablet, and mobile devices.**
      * **Accelerate development using the AWS Amplify UI component library or other UI libraries.**
      * **Test multi-device compatibility using AWS Device Farm.**
   6. **Frontend Performance Optimization**
      * **Deploy static resources to AWS S3 and use AWS CloudFront as a CDN for acceleration.**
      * **Implement code splitting, lazy loading, and image compression optimization.**
      * **Configure caching strategies to improve load speeds.**
   7. **Frontend Testing & Debugging**
      * **Write unit tests, integration tests, and end-to-end tests.**
      * **Utilize AWS Amplify’s tools for automated testing and deployment.**
      * **Use Amazon CloudWatch to monitor frontend request performance and user experience metrics.**
   8. **Deployment & Iterative Improvement**
      * **Deploy the static website via AWS S3 and configure custom domains with HTTPS on AWS Route 53 (using AWS Certificate Manager).**
      * **Continuously collect user feedback and usage data for periodic UI/UX adjustments and feature iterations.**
   9. **Technologies & Tools: AWS Amplify, S3, CloudFront, Route 53, Certificate Manager, and related development tools.**
2. **Frontend Interaction & Data Visualization**
   1. **Requirements Analysis & Data Structure Design**
      * **Identify key data to display (e.g., user activity, evaluation results, rankings, real-time updates).**
      * **Design API response data formats and data flow diagrams to ensure stable data interaction between the frontend and backend.**
   2. **Implementing Frontend Data Interaction**
      * **Use AWS Amplify or AWS AppSync to connect to the backend API.**
      * **Implement REST or GraphQL interfaces for data requests and response handling.**
      * **Configure frontend state management (e.g., Redux, MobX, or React Context) to ensure consistent data flow among components.**
   3. **Real-Time Data Updates & Interaction**
      * **Integrate WebSocket or AWS AppSync subscription features for real-time data push (e.g., rankings, notifications).**
      * **Design a real-time interaction module so users can instantly see data changes.**
   4. **Data Visualization Component Development**
      * **Choose suitable chart libraries (e.g., Chart.js, D3.js, Recharts) for graphical data display.**
      * **Develop various charts (bar charts, line charts, pie charts, scatter plots) to show statistical data, evaluation results, and matching scenarios.**
      * **Enhance component reusability and responsiveness using the AWS Amplify UI component library or custom components.**
   5. **Dashboard & Comprehensive Data Display**
      * **Design a dashboard page to centralize the display of key platform metrics (e.g., user activity, task completion rate, recruitment match rate).**
      * **Use AWS QuickSight (if advanced data visualization and BI capabilities are needed) for data analysis and reporting.**
      * **Implement features such as filtering, sorting, and dynamic refreshing to improve user experience.**
   6. **Frontend Performance & Loading Optimization**
      * **Optimize data loading and component rendering using code splitting, lazy loading, and caching strategies.**
      * **Deploy static resources to AWS S3 and accelerate global access with AWS CloudFront.**
   7. **Testing & Monitoring**
      * **Write frontend unit tests, integration tests, and end-to-end tests to ensure proper data interaction and correct chart display.**
      * **Configure AWS CloudWatch (or built-in Amplify monitoring tools) to monitor API call performance, frontend errors, and user interaction data.**
   8. **Documentation & User Feedback Collection**
      * **Prepare documentation and technical guides for the data interaction and visualization modules to facilitate future maintenance and iteration.**
      * **Use A/B testing and user surveys to continuously optimize data display and interaction.**
3. **Frontend Integration**
   1. **Project Environment Configuration**
      * **Set up the frontend project (using frameworks like React or Vue).**
      * **Initialize project configuration with AWS Amplify CLI.**
      * **Link the Amplify environment with the AWS account.**
   2. **AWS Services Integration**
      * **Integrate AWS Cognito: Use Amplify Auth to connect to the Cognito user pool for registration, login, password resets, etc.**
      * **API Integration: Connect to backend REST/GraphQL APIs using Amplify API or AppSync.**
      * **Storage Integration: Configure AWS S3 for static resource storage and manage file uploads using Amplify Storage.**
   3. **Frontend & Backend Data Interaction**
      * **Write frontend service functions to call interfaces provided by API Gateway or AppSync.**
      * **Configure Amplify DataStore (if offline support is needed) to manage data synchronization between local storage and the cloud.**
   4. **Frontend Routing & State Management**
      * **Design a single-page application routing structure using React Router or Vue Router.**
      * **Use Redux, Context API, or Vuex for global state management to ensure smooth transfer of user sessions, authentication states, and data.**
   5. **CI/CD & Automated Deployment**
      * **Utilize AWS Amplify Console for automated build and deployment of the frontend application.**
      * **Configure Git repositories (such as AWS CodeCommit or GitHub) to integrate with Amplify Console and automatically trigger build, test, and deployment on code updates.**
      * **Use AWS CodePipeline/CodeBuild for continuous integration of the frontend code.**
   6. **Static Resource Deployment & CDN Acceleration**
      * **Deploy frontend static files to AWS S3.**
      * **Configure AWS CloudFront as the CDN to provide fast global access.**
   7. **Frontend Monitoring & Log Collection**
      * **Use monitoring functions provided by AWS Amplify Console to track the frontend application's status.**
      * **Configure CloudWatch Logs to monitor frontend errors and API call performance.**
   8. **Summary: By combining AWS Amplify, Cognito, API Gateway/AppSync, S3, CloudFront, CodePipeline, and other AWS services, the frontend integration ensures efficient connectivity with the backend, forming a stable and scalable application.**

**Backend Module**

1. **Task Generation Module**
   1. **Requirements Research & Data Collection**
      * **Job Description Data Collection:**
        + **Design and deploy questionnaires or forms to collect job description data from enterprises, including required fields (job title, job requirements, skills needed, responsibilities, industry background, etc.) and optional supplementary information.**
        + **Establish a data storage solution using AWS RDS or DynamoDB to store the raw Job Description data, ensuring a consistent and structured format.**
      * **Literature Review:**
        + **Investigate the latest advances of LLMs in text generation and task automation. Reference papers such as “Language Models are Few-Shot Learners” (Brown et al., 2020) and related literature on multi-agent collaborative generation and the application of Generative Adversarial Networks (GANs) in task generation.**
        + **Identify relevant frameworks (e.g., Hugging Face Transformers, OpenAI API) and multi-agent scheduling mechanisms to provide theoretical support for subsequent algorithm design.**
   2. **Algorithm Design & Model Selection**
      * **Text Generation Model:**
        + **Use an LLM (such as GPT‑4) as the main generation engine and design prompts to enable it to generate Hackathon-style tasks based on the Job Description.**
        + **Explore incorporating GANs or Variational Autoencoders (VAEs) to supplement task diversity and innovation.**
      * **Multi-Agent Collaborative Generation:**
        + **Evaluate the use of a multi-agent system where each agent is responsible for parsing the Job Description, generating an initial draft of the task, and providing iterative feedback.**
        + **Build a collaborative framework that uses collaborative filtering, voting mechanisms, or reinforcement learning strategies to determine the final generated task.**
      * **Model Tuning & Evaluation:**
        + **Design evaluation metrics (such as the reasonableness of generated tasks, task coverage, innovation, and operability) and assess the results with expert scoring and automated metrics (BLEU, ROUGE).**
   3. **System Architecture & Backend Implementation**
      * **System Architecture Design:**
        + **Construct a serverless architecture based on AWS. Use AWS Lambda for the task generation service and API Gateway as the external interface.**
        + **Employ AWS SageMaker for LLM calls, training, and inference, combined with AWS Step Functions to orchestrate multi-step workflows.**
      * **Data Preprocessing & Feature Extraction:**
        + **Write preprocessing scripts to clean and parse Job Description texts and extract key information (keywords, skills, responsibilities, etc.).**
        + **Use natural language processing tools (e.g., NLTK, spaCy) for tokenization, entity recognition, and semantic embedding to provide structured context for the LLM prompt.**
      * **API Development:**
        + **Develop RESTful APIs to allow the frontend to upload Job Descriptions, request task generation, and retrieve results.**
        + **Utilize AWS API Gateway, Lambda, and DynamoDB to store generated task records.**
   4. **Development & Debugging Process**
      * **Unit & Integration Testing:**
        + **Write unit tests for each Lambda function and API endpoint to ensure that text preprocessing, model calls, and feedback loops function properly.**
        + **Design integration test scenarios using realistic Job Description data to validate the task generation logic and multi-agent collaboration.**
      * **Logging & Monitoring:**
        + **Configure Amazon CloudWatch Logs to collect logs from all modules, facilitating debugging during the generation process.**
        + **Set up CloudWatch Alarms to monitor key processes (e.g., model call latency, generation failure rate).**
      * **User Feedback Loop:**
        + **Establish an internal feedback mechanism by inviting some enterprise users to evaluate the generated tasks for reasonableness and innovation, and use the feedback for model tuning.**
        + **Continuously adjust the prompt design and multi-agent collaboration strategy based on expert and user feedback.**
   5. **Supporting Papers & Frameworks**
      * **Reference Papers:**
        + **“Language Models are Few-Shot Learners” – Brown et al. (2020)**
        + **“Multi-Agent Communication and Coordination for Task Generation” – [Example reference, further research needed for the latest papers]**
        + **“Generative Adversarial Networks for Text Generation” – Related survey papers on GANs in text generation.**
      * **Frameworks & Tools:**
        + **Hugging Face Transformers (for GPT‑4 integration)**
        + **AWS SageMaker (for model deployment and inference)**
        + **AWS Lambda and API Gateway (for serverless API implementation)**
        + **AWS Step Functions (for workflow orchestration)**
   6. **Additional Considerations**
      * **Feedback & Continuous Improvement Mechanism**
        + **Set up channels for expert and user feedback to regularly collect evaluations and improvement suggestions for the generated tasks.**
        + **Use A/B testing and automated evaluation metrics (e.g., BLEU, ROUGE, task coverage) to quantitatively assess generation results and feed them back into the model tuning process.**
      * **Data Preprocessing & Quality Control**
        + **Design strict data cleaning and standardization procedures to address format inconsistencies and noise in Job Description data.**
        + **Automatically extract key fields and semantic features to ensure high-quality input prompts and improve task generation accuracy.**
      * **Multi-Agent Collaborative Scheduling Strategy**
        + **Clearly define the roles, communication protocols, and collaboration mechanisms if using a multi-agent system.**
        + **Establish clear voting or weighting rules so that outputs from multiple agents can be fused into a final task proposal.**
      * **Documentation & Version Management**
        + **Write detailed API documentation and system architecture guides to ensure clear reference for team collaboration and future maintenance.**
        + **Use version control systems (such as Git) and integrate CI/CD pipelines to ensure code quality and stable deployment.**
      * **Workflow Automation**
        + **Use AWS Step Functions to orchestrate calls between multiple Lambda functions, ensuring automation of the task generation, evaluation, and feedback process.**
        + **Configure automated testing, log collection (CloudWatch), and monitoring to promptly capture anomalies and performance bottlenecks.**
      * **Model & Algorithm Updates**
        + **Regularly review the task generation effectiveness and adjust prompt templates and generation strategies based on different job types and industry scenarios.**
        + **Stay updated with cutting-edge research and technical advancements to timely incorporate new generation models or multi-agent collaborative algorithms and maintain system competitiveness.**
      * **Summary: With the steps above, a Task Generation module based on the AWS platform is built—from Job Description data collection and preprocessing, to model and algorithm design, multi-agent collaborative generation, API development, debugging/testing, and feedback optimization. The entire process leverages AWS services such as Lambda, SageMaker, API Gateway, and Step Functions to achieve efficient, automated, and intelligent task generation for Hackathon-style Capsule challenges.**
2. **Solution Generation & Evaluation Module**
   1. **DS Agent Optimization:**
      * **Prompt Optimization:**
        + **Refine and detail input prompts to improve the accuracy and targeting of generated tasks.**
        + **Regularly update prompt templates and fine-tune them using expert feedback and user data.**
      * **Multi-Agent Collaboration:**
        + **Introduce a multi-agent mechanism where different agents are responsible for parsing, generating, and revising task texts.**
        + **Use voting or weighted fusion strategies to combine outputs from multiple agents, enhancing robustness and diversity.**
      * **Generation Model Improvement:**
        + **Explore integrating GANs or VAEs to enhance the innovation and diversity of generated results.**
        + **Optimize task generation quality through adversarial training.**
      * **Feedback Loop Optimization:**
        + **Establish an automated evaluation module that uses quantitative metrics (e.g., BLEU, ROUGE) and expert scoring to comprehensively assess generated tasks.**
        + **Feed evaluation results back into the generation module for continuous adjustment of generation strategies and model parameters.**
      * **Continuous Model Tuning:**
        + **Deploy and manage the LLM model using AWS SageMaker to enable online inference and ongoing tuning.**
        + **Regularly collect logs and monitoring data, analyze performance bottlenecks using CloudWatch, and update the model as needed.**
      * **Data Preprocessing & Quality Control:**
        + **Enhance the cleaning, standardization, and feature extraction of Job Description data to ensure high input quality and improve task generation outcomes.**
      * **Automated Workflow Integration:**
        + **Use AWS Step Functions and Lambda to build an end-to-end automated process that seamlessly integrates task generation, evaluation, and feedback.**
        + **Configure a CI/CD pipeline to ensure rapid iteration and updates of both the model and the generation module.**
   2. **Requirements Analysis & Data Collection**
      * **Standardizing Task Descriptions: Collect Job Description data from enterprises, analyze its format and key information (job title, responsibilities, skill requirements, etc.), and develop standardized templates.**
      * **Literature Review: Consult recent papers (e.g., “Language Models are Few-Shot Learners” and related studies on multi-agent collaborative generation and GAN applications in text generation) to clarify model choices and prompt design strategies.**
      * **Data Preprocessing: Use NLP tools (e.g., spaCy, NLTK) to clean, tokenize, perform entity recognition, and extract features, forming structured input data.**
   3. **Algorithm Design & Model Construction**
      * **LLM Task Generation:**
        + **Use the DS-Agent framework as a foundation with GPT‑4 and other large language models to automatically generate Hackathon-style Capsule tasks from Job Descriptions via carefully designed prompts.**
        + **Analyze prompt design strategies to ensure the generated tasks are both business-relevant and challenging.**
      * **Multi-Agent Coordination:**
        + **Consider introducing a multi-agent system where multiple agents handle text parsing, initial task generation, and task content refinement.**
        + **Fuse outputs from various agents through voting or weighted fusion to generate the final result.**
      * **Generative Networks:**
        + **Consider using GAN or VAE models to supplement the traditional LLM’s generation capability, enhancing the diversity and innovation of the generated tasks.**
        + **Design a cooperative training process between the generator and discriminator to ensure the output tasks meet real business needs.**
      * **Model Tuning & Evaluation Metrics:**
        + **Design quantitative metrics (e.g., BLEU, ROUGE, task coverage) and qualitative expert scoring mechanisms to evaluate the reasonableness and operability of generated tasks.**
        + **Use AWS SageMaker for model training, tuning, and online inference to achieve automated evaluation feedback.**
   4. **System Architecture & Backend Implementation**
      * **Architecture Design:**
        + **Build the backend service using a serverless architecture with AWS Lambda for executing business logic and AWS API Gateway for exposing generation and evaluation interfaces.**
        + **Orchestrate multi-step workflows for task generation, evaluation, and feedback using AWS Step Functions.**
      * **Model Invocation & Integration:**
        + **Integrate the DS-Agent as the core module in the backend and call GPT‑4 and auxiliary generation models via AWS SageMaker Endpoints.**
        + **Design API interfaces so that when the frontend submits a Job Description, a Lambda function triggers DS-Agent to return the generated task proposal.**
      * **Feedback Mechanism & Automated Evaluation:**
        + **Develop an automated evaluation module to score the generated tasks, feeding the evaluation results back into the generation module for continuous tuning.**
        + **Monitor the entire generation process with CloudWatch Logs and CloudWatch Metrics to ensure system stability.**
   5. **Development & Debugging Process**
      * **Unit Testing: Write unit tests for text preprocessing, model calls, task generation, and evaluation feedback submodules to ensure each part works correctly.**
      * **Integration Testing: Simulate real Job Description data for end-to-end testing to verify the entire generation and evaluation process.**
      * **Logging & Monitoring: Configure Amazon CloudWatch to collect logs from Lambda functions and SageMaker Endpoints, setting alarms to respond to exceptions.**
      * **User Feedback Collection: Deploy a trial version, invite enterprise users and experts to score and provide feedback on generated tasks, and fine-tune prompts and model parameters through A/B testing.**
   6. **Supporting Papers & Frameworks**
      * **Reference Papers:**
        + **“Language Models are Few-Shot Learners” – Brown et al. (2020)**
        + **“Multi-Agent Communication and Coordination for Task Generation” – related research (further investigation required for the latest papers)**
        + **“Generative Adversarial Networks for Text Generation” – application of GANs in text generation.**
      * **Development Frameworks & Tools:**
        + **Hugging Face Transformers for invoking GPT‑4.**
        + **AWS SageMaker for model deployment, tuning, and inference.**
        + **AWS Lambda and API Gateway for serverless API implementation.**
        + **AWS Step Functions for workflow orchestration.**
        + **NLP tools (spaCy, NLTK) for data preprocessing and monitoring tools (CloudWatch Logs/Metrics).**
   7. **Technical Challenges & Key Issues**
      * **Multi-Agent Coordination Mechanism: Clearly define each agent’s role and coordination strategy to ensure the overall quality of generated tasks.**
      * **Diversity & Innovation of Generated Tasks: Address the limitations of a single LLM by leveraging GAN/VAE techniques and designing an effective adversarial training scheme.**
      * **Feedback Loop Mechanism: Build a set of quantitative and qualitative evaluation metrics to effectively translate expert feedback into model optimization parameters for continuous improvement.**
      * **Data Standardization: Ensure uniform format for Job Description data provided by enterprises to avoid noise affecting the generation quality.**
3. **Database Management**
   1. **Requirements Research & Data Model Design**
      * **Analyze the types of data that need to be stored (user information, job data, Capsule tasks, evaluation results, matching data, etc.).**
      * **Create an ER diagram and design the data structure and table schemas.**
      * **Define data fields, relationships, indexing strategies, and data access patterns.**
   2. **Technology Selection & Architecture Design**
      * **Choose a relational database (e.g., AWS RDS with PostgreSQL or Amazon Aurora).**
      * **Consider AWS DynamoDB if NoSQL is required.**
      * **For similarity search or vector data, consider integrating Amazon OpenSearch or using solutions like FAISS/Pinecone.**
      * **Design for database scalability, read/write separation, and high availability.**
   3. **Database Instance Deployment & Configuration**
      * **Create RDS/Aurora instances in the AWS console and configure instance type, storage, networking, etc.**
      * **Configure parameter groups, maintenance windows, automatic backups, and snapshots.**
      * **Set up VPC, subnets, and security groups to ensure backend services can access the database instance.**
   4. **Data Migration & Initialization**
      * **Write SQL scripts to create tables, indexes, and constraints.**
      * **Use AWS Database Migration Service (DMS) or manual data import for initial data migration.**
      * **Configure seed scripts to ensure consistency across development, testing, and production environments.**
   5. **Data Access Layer Development**
      * **Integrate ORM frameworks (such as SQLAlchemy or Django ORM) into the backend service for data operations.**
      * **Develop data access APIs to implement CRUD operations.**
      * **Optimize queries using indexes and views.**
   6. **Database Monitoring & Performance Optimization**
      * **Configure Amazon CloudWatch to monitor RDS/Aurora performance metrics (e.g., CPU, memory, IO).**
      * **Set up auto-scaling and alert mechanisms to respond to database performance bottlenecks.**
      * **Regularly perform query optimization and database maintenance.**
   7. **Data Backup & Recovery Strategy**
      * **Configure automatic backup and snapshot strategies to ensure regular data backups.**
      * **Develop a recovery process and conduct drills to ensure quick data recovery in case of failures.**
   8. **Summary: With the above tasks, utilize AWS services such as RDS, Aurora, DynamoDB, and CloudWatch to build an efficient and scalable database management system that supports the data storage and business logic needs of the AI recruitment platform.**
4. **Registration & Authentication System**
   1. **Requirements Analysis & Architecture Design**
      * **Analyze requirements for user registration, login, password reset, and account management.**
      * **Design the system architecture diagram to clarify interactions between the frontend, backend, and AWS services.**
   2. **Configure AWS Cognito User Pool**
      * **Create a user pool and define required attributes and verification methods.**
      * **Configure password policies and account recovery strategies.**
      * **Create an App Client and set up OAuth settings with redirect URIs.**
   3. **Integrate Third-Party Identity Federation**
      * **Configure identity providers such as Google and Facebook within Cognito.**
      * **Generate OAuth 2.0 credentials in the Google Cloud Console and complete the configuration in Cognito.**
   4. **Backend API Development & Encapsulation**
      * **Write AWS Lambda functions for registration, login, password reset, etc.**
      * **Expose these functions as REST/GraphQL endpoints using API Gateway.**
      * **Implement APIs for querying and updating user data.**
   5. **User Session Management**
      * **Configure the issuance and refresh mechanism for Cognito tokens.**
      * **Implement JWT-based session verification in the backend to ensure secure API calls.**
   6. **Integration & Testing**
      * **Use Postman or integrated testing tools to validate the registration and authentication APIs.**
      * **Invoke Cognito APIs to ensure user information is correctly stored and that verification emails/SMS are sent properly.**
   7. **Deployment & Automation**
      * **Deploy Lambda functions and API Gateway using AWS SAM or Serverless Framework.**
      * **Set up AWS CodePipeline/CodeBuild for continuous integration and automated deployment.**
   8. **Documentation & API Specifications**
      * **Document the APIs with detailed explanations of the registration, login, and password reset methods along with parameter formats.**
   9. **Summary: By combining AWS Cognito, Lambda, API Gateway, and automation tools, a robust registration and authentication system based on the AWS platform is implemented.**
5. **API Development**
   1. **API Requirements Analysis & Design**
      * **Define business requirements and determine the functionalities to be exposed (e.g., user registration, login, data querying, Capsule task generation, candidate matching, etc.).**
      * **Create API flow diagrams and data interaction diagrams, and develop interface specifications (e.g., using Swagger/OpenAPI standards).**
   2. **Technology Selection & Architecture Design**
      * **Choose between RESTful API or GraphQL based on business needs.**
      * **Decide on a backend implementation strategy using AWS Lambda for serverless functions combined with API Gateway for exposing interfaces.**
   3. **Development Environment & Tool Configuration**
      * **Use AWS SAM, Serverless Framework, or AWS CDK to build and manage the API deployment process.**
      * **Utilize Git for version management and integrate with AWS CodeCommit or GitHub for code hosting and collaboration.**
   4. **API Implementation & Functionality**
      * **Develop Lambda functions for various business scenarios such as registration, login, data querying, Capsule task generation, and matching recommendations.**
      * **Integrate database access (using RDS/Aurora or DynamoDB) and external service calls (e.g., invoking AWS Cognito for authentication).**
      * **Configure API Gateway endpoints, traffic control, and CORS settings.**
   5. **Security & Access Control**
      * **Configure access permissions in API Gateway using IAM roles, Cognito Authorizers, etc.**
      * **Set up rate limiting and monitor API performance using AWS CloudWatch.**
   6. **Testing & Debugging**
      * **Use Postman or the AWS API Gateway test console to perform functional and unit testing for each endpoint.**
      * **Write unit tests for Lambda functions to ensure proper logic.**
   7. **Automated Deployment & Continuous Integration**
      * **Configure CI/CD pipelines with AWS CodePipeline/CodeBuild for automatic building, testing, and deployment of APIs.**
      * **Use AWS SAM or Serverless Framework for Infrastructure as Code (IaC) management.**
   8. **Documentation & Version Management**
      * **Generate API documentation (e.g., via Swagger/OpenAPI) to ensure correct API usage by frontend developers and third parties.**
      * **Manage API versions and plan for iterative updates.**
   9. **Summary: Using AWS Lambda, API Gateway, SAM/Serverless Framework, and AWS CodePipeline, a robust and scalable API development and deployment process is established to meet various business requirements.**
6. **Matching Algorithm Module**
   1. **Requirements Analysis & Data Modeling**
      * **Define candidate features, job descriptions, and Capsule performance data structures.**
      * **Create data flow and relationship diagrams, and determine matching metrics and weights.**
   2. **Technology Selection & Algorithm Design**
      * **Select suitable matching algorithms (such as collaborative filtering, content-based recommendation, hybrid recommendation, or machine learning models).**
      * **Define similarity calculation methods (e.g., cosine similarity, Euclidean distance).**
      * **Use AWS SageMaker for model training and tuning (if deep learning algorithms are needed).**
   3. **Data Preprocessing & Feature Extraction**
      * **Use AWS Glue or Lambda for data ETL processing.**
      * **Extract features from user behavior and Capsule task data.**
      * **Process structured data and store vectors (integrating AWS DynamoDB, Amazon OpenSearch, or Pinecone as needed).**
   4. **Matching Algorithm Implementation**
      * **Write matching algorithm services using AWS Lambda.**
      * **Call AWS SageMaker endpoints for online prediction or batch matching.**
      * **Integrate machine learning models to compute similarity between candidates and job positions in real time.**
   5. **Interface Design & Integration**
      * **Design API interfaces that accept query requests and return matching results.**
      * **Expose the matching service via API Gateway.**
   6. **Performance Optimization & Scalability**
      * **Deploy the algorithm as microservices using AWS Lambda for serverless scalability.**
      * **Configure caching strategies (using AWS ElastiCache) to reduce repeated computation overhead.**
      * **Monitor matching service performance using Amazon CloudWatch and adjust resource allocation accordingly.**
   7. **Testing & Validation**
      * **Write unit tests and integration tests to ensure the accuracy and stability of the matching algorithm.**
      * **Conduct A/B testing to validate recommendation effectiveness and user satisfaction.**
   8. **Summary: Utilizing AWS SageMaker, Lambda, API Gateway, Glue, ElastiCache, and CloudWatch, a highly efficient and scalable matching algorithm module is built to support precise matching among talent, job positions, and Capsule performance.**
7. **Backend Infrastructure & Deployment**
   1. **Infrastructure Architecture Planning**
      * **Create an overall system architecture diagram.**
      * **Define microservice decomposition and serverless deployment strategies.**
      * **Clarify dependencies and data flows among services.**
   2. **Infrastructure as Code (IaC)**
      * **Choose AWS CloudFormation, AWS CDK, or Serverless Framework.**
      * **Write templates/code to describe backend service resources (Lambda, API Gateway, RDS/Aurora, DynamoDB, S3, etc.).**
      * **Configure resource version management and environment variables.**
   3. **Serverless Computing Resource Deployment**
      * **Develop and deploy AWS Lambda functions as business logic processing units.**
      * **Use API Gateway to expose RESTful or GraphQL interfaces.**
      * **Configure integrations between Lambda and databases or other AWS services.**
   4. **Database & Storage Configuration**
      * **Deploy relational databases using AWS RDS or Aurora.**
      * **Configure DynamoDB for NoSQL storage.**
      * **Set up S3 buckets for static resource and file storage.**
   5. **CI/CD Pipeline Construction**
      * **Use AWS CodePipeline/CodeBuild or GitHub Actions for continuous integration and deployment.**
      * **Configure automated testing, build, packaging, and deployment tasks.**
      * **Integrate version control and rollback strategies.**
   6. **Containerization & Microservices Deployment (if applicable)**
      * **Build service images using Docker and deploy to AWS ECS or EKS.**
      * **Configure load balancers (e.g., AWS ALB) and auto-scaling groups for high availability.**
   7. **Log Collection & Monitoring**
      * **Configure Amazon CloudWatch Logs to collect logs from Lambda and API Gateway.**
      * **Use CloudWatch Metrics to monitor system performance (response time, error rate, resource utilization).**
      * **Set alert strategies to respond promptly to abnormal situations.**
   8. **Deployment Testing & Environment Isolation**
      * **Set up development, testing, pre-production, and production environments to ensure resource isolation.**
      * **Implement blue-green deployments or canary releases to reduce deployment risk.**
   9. **Documentation & Operations Manual**
      * **Write deployment documentation, operations manuals, and resource management strategies.**
      * **Configure automated backup and recovery solutions to ensure data security and system stability.**
   10. **Summary: Using AWS CloudFormation/CDK, Lambda, API Gateway, RDS/Aurora, DynamoDB, S3, CodePipeline, CloudWatch, etc., a robust, automated, and scalable backend infrastructure and deployment system is built to ensure the stable operation and rapid iteration of all microservices.**
8. **Notification System**
   1. **Requirements Definition & Event Triggers**
      * **Identify business scenarios requiring notifications (e.g., registration success, task submission, evaluation updates, matching alerts).**
      * **Define trigger conditions and notification content formats for different event types.**
   2. **Technology Selection**
      * **Choose AWS SNS (Simple Notification Service) as the primary notification service for email, SMS, and push notifications.**
      * **For more advanced notification channels and customization, consider integrating Amazon Pinpoint.**
   3. **Notification Topic Configuration**
      * **Create corresponding notification topics in SNS.**
      * **Configure subscriptions for different notification types (e.g., email, SMS, mobile push).**
   4. **Backend Notification Module Development**
      * **Develop Lambda functions as event handlers that call the SNS API to send notifications upon receiving specific business events.**
      * **Integrate with API Gateway or directly call Lambda within the business logic to trigger the notification process.**
   5. **User Notification Preference Management**
      * **Develop API endpoints that allow users to set and modify their notification preferences (selecting channels, notification frequency, etc.).**
      * **Store user preference data in the database and filter notifications accordingly.**
   6. **Integration & Testing**
      * **Test each notification topic and subscription via the AWS SNS console.**
      * **Use Postman or local simulations to trigger events and verify the integration between Lambda and SNS.**
      * **Check the accuracy of notification content and monitor sending delays.**
   7. **Monitoring & Log Collection**
      * **Configure Amazon CloudWatch Logs to collect logs from Lambda and SNS.**
      * **Set CloudWatch Alarms to alert on notification failures, delays, and other anomalies.**

**Project Timeline**

**Phase 1 (1–3 Months): Research & Requirements Modeling**

* **Requirements Research & User Interviews**
  + **Collect and analyze sample Job Description data provided by enterprises.**
  + **Research current industry recruitment, training, and talent matching processes to clarify the platform’s core features (job seeking, training, Capsule challenges, headhunting matching).**
  + **Complete user role definitions, user journey mapping, and preliminary system architecture sketches.**
* **Literature Review & Technology Selection**
  + **Study the latest advances in LLMs for task generation and automated evaluation (e.g., GPT‑4, Few-Shot Learning).**
  + **Research multi-agent collaborative generation and GAN/VAE applications in text generation; collect relevant papers (e.g., “Language Models are Few-Shot Learners”).**
  + **Identify key AWS services: Amplify, Cognito, Lambda, API Gateway, SageMaker, Step Functions, RDS/Aurora, DynamoDB, SNS, CloudFront, CloudWatch.**
* **Data Modeling & Preprocessing Process Design**
  + **Develop a standardized Job Description template and design data cleaning, tokenization, entity recognition, and semantic embedding processes (using spaCy, NLTK, etc.).**
  + **Create an ER diagram to define database structures, table schemas, indexing strategies, and data access patterns; select RDS/Aurora or DynamoDB as the primary storage.**

**Phase 2 (4–6 Months): System Development & Core Module Implementation**

* **Frontend Module Development**
  + **Set up a frontend project based on React/Vue and initialize it using AWS Amplify CLI.**
  + **Develop pages for registration, login, job seeker dashboard, HR enterprise portal, Capsule simulation, and task display.**
  + **Integrate AWS Cognito (Amplify Auth), API Gateway, S3, and CloudFront to implement user authentication, data interaction, and responsive design.**
  + **Implement frontend interaction and data visualization components to support real-time data display for rankings, notifications, etc.**
* **Task Generation Module Development**
  + **Use the DS-Agent framework with GPT‑4 and Hugging Face Transformers to design task generation prompts.**
  + **Explore multi-agent collaborative generation: build multiple agents for text parsing, initial task drafting, and task refinement using voting or weighted fusion strategies.**
  + **Incorporate GAN or VAE models for adversarial training to improve task diversity and innovation.**
  + **Integrate AWS SageMaker to deploy the LLM model and use Lambda, API Gateway, and Step Functions to construct an end-to-end automated workflow.**
* **Solution Generation & Evaluation Module Development**
  + **Design an automated evaluation system combining BLEU, ROUGE, and expert scoring to assess task reasonableness and coverage.**
  + **Establish a feedback loop to continuously optimize prompts and generation strategies.**
  + **Utilize CloudWatch to monitor model calls, response times, and generation quality.**
* **Backend API & Database Management**
  + **Develop serverless APIs (using AWS Lambda and API Gateway) to encapsulate registration, authentication, data querying, task generation, and matching recommendation functions.**
  + **Deploy the database (RDS/Aurora or DynamoDB) and create data preprocessing scripts to ensure consistency and quality of Job Description and Capsule data.**
  + **Build the notification system and matching algorithm module using SNS, Lambda, SageMaker, and OpenSearch/Pinecone for real-time notifications and intelligent matching.**

**Phase 3 (7–9 Months): System Integration, Testing & Optimization**

* **Frontend-Backend System Integration**
  + **Seamlessly integrate frontend modules with backend APIs to ensure consistent data flow and interaction logic.**
  + **Configure frontend state management and real-time data update mechanisms for dashboards and data visualization.**
* **Automated Deployment & CI/CD Pipeline Construction**
  + **Use AWS Amplify Console and AWS CodePipeline/CodeBuild to establish continuous integration and automated deployment pipelines for both frontend and backend.**
  + **Employ AWS CloudFormation/CDK/Serverless Framework for Infrastructure as Code (IaC) management to automate the deployment of Lambda, API Gateway, databases, etc.**
* **System Testing & Debugging**
  + **Write unit tests, integration tests, and end-to-end tests covering registration, authentication, task generation, evaluation, matching, and notification modules.**
  + **Set up development, testing, and pre-production environments to conduct stress tests and performance optimizations.**
  + **Configure CloudWatch Logs/Metrics for system performance monitoring and set up alert mechanisms.**
* **User Feedback & Model Tuning**
  + **Invite enterprise users, job seekers, and experts to participate in trial runs and collect user feedback.**
  + **Use A/B testing and expert scoring to fine-tune the DS-Agent, generation model, and evaluation feedback loop.**

**Phase 4 (10–12 Months): Production Deployment, Promotion & Continuous Improvement**

* **Production Deployment & Launch**
  + **Deploy the frontend static website using AWS S3, CloudFront, and Route 53, configuring custom domains and HTTPS (via AWS Certificate Manager).**
  + **Launch backend serverless resources to ensure the stable operation of all modules.**
* **Performance Monitoring & Scalability Optimization**
  + **Continuously monitor the performance and error rates of Lambda, SageMaker, API Gateway, databases, etc., using CloudWatch.**
  + **Configure auto-scaling (e.g., Lambda concurrency, RDS auto-scaling) based on traffic demands to ensure high availability.**
* **Continuous Model Updates & Technology Follow-up**
  + **Regularly review task generation and evaluation performance, and adjust generation strategies, prompt templates, and multi-agent collaboration mechanisms based on different industries and job types.**
  + **Stay updated with cutting-edge research and incorporate new generation models and multi-agent algorithms to maintain system competitiveness.**
* **Product Promotion & User Support**
  + **Conduct user training and promotion activities, inviting enterprises and job seekers to use the platform.**
  + **Continuously collect user data and feedback, updating technical documentation and operational manuals to refine platform features and UI/UX.**

**Project Priorities & Key Technologies**

* **Core Project Focus:**
  + **Use the DS-Agent framework and LLMs (e.g., GPT‑4) to automatically generate Hackathon-style Capsule tasks based on Job Descriptions.**
  + **Build a multi-agent collaboration mechanism and supplement with generative adversarial networks (GAN/VAE) to ensure task diversity and innovation.**
  + **Develop an automated evaluation and feedback loop to continuously optimize generated tasks and enable online model tuning.**
* **Key Technologies & Methods:**
  + **AWS Platform: Amplify, Cognito, Lambda, API Gateway, SageMaker, Step Functions, RDS/Aurora, DynamoDB, SNS, S3, CloudFront, CloudWatch.**
  + **Frontend Technologies: React/Vue, AWS Amplify, state management & responsive design, data visualization (Chart.js, D3.js, Recharts).**
  + **Backend Technologies: Serverless architecture, microservices design, REST/GraphQL API, automated CI/CD, database design (RDS/Aurora, DynamoDB).**
  + **Data Processing & Machine Learning: NLP preprocessing (spaCy, NLTK), feature extraction, LLM generation, multi-agent collaboration, generative adversarial networks, and automated evaluation metrics (BLEU, ROUGE).**
* **Workflow Overview:**
  + **Frontend: Design and development, interactive data display, seamless integration with the backend, and automated deployment.**
  + **Backend: Data collection and preprocessing, task generation (DS-Agent), solution generation and automated evaluation, matching recommendations, and complete implementation of registration, authentication, and notification systems.**
  + **Overall: Utilize AWS services to achieve automated deployment, continuous integration, log monitoring, performance optimization, and rapid iterative updates.**