WEEK 2: Deep Candidate Screening + Scoring Foundation

🗓️ Days 1-2: Resume Parser + GitHub Link Extractor

🧠 Objectives

Develop a high-performance resume parsing system that extracts and structures candidate data with particular focus on technical profiles, GitHub repositories, and certification credentials.

Detailed Team Responsibilities

🔧 **Fullstack Developers**

- **Frontend (React/Next.js):**
    - **File Upload Component**:
        - Implement drag-and-drop zone with animated border feedback
        - Add file type validation for `.pdf`, `.docx`, `.doc` with custom error messages
        - Create size limit validation (max 10MB) with user-friendly alerts
        - Design progress tracking UI with percentage indicator and animated spinner
        - Implement error handling with retry functionality
        - Add preview capability for uploaded documents
    - **UI State Management**:
        - Set up Redux store with specific slices for upload status, parsing progress, and results
        - Implement optimistic UI updates with proper loading states
        - Create notification system for success/failure events

- **Backend (Node.js/Python - FastAPI):**
    - **API Endpoints**:
        - `/upload-resume`:
            - Multipart form data handler with stream processing
            - Virus scanning integration before storage
            - Generate unique file identifier with timestamp and user reference
        - `/parsing-status/:id`: Endpoint for checking parse status
        - `/parsed-results/:id`: Endpoint for retrieving parsing results
    - **Storage Configuration**:
        - Implement Google Cloud Storage with proper IAM role configuration
        - Set up file lifecycle policies (30-day retention)
        - Create fallback local storage for development environments

Database Schema Implementation:

```
{
 candidateId: ObjectId,
 name: String,
 email: String,
 phone: String,
 skills: [
  {
   name: String,
   yearsExperience: Number,
   context: String, // where in resume it was mentioned
   confidence: Number // NLP confidence score
  }
 ],
 github_links: [
  {
   url: String,
   username: String,
   repositoryCount: Number,
   profileCreatedAt: Date,
   extractedFrom: String // section of resume
  }
 ],
```

```
  cert_links: [
    {
      platform: String, // "Coursera", "Udemy", etc.
      url: String,
      courseName: String,
      issueDate: Date,
      verificationStatus: String // "Pending", "Verified", "Failed"
    }
  ],
  leetcode_profile: String,
  linkedIn_profile: String,
  resumeGCSLocation: String,
  originalFilename: String,
  processingStatus: String, // "Queued", "Processing", "Completed", "Failed"
  createdAt: Date,
  updatedAt: Date
}
```

- **Queue System**:
  - Set up Google Cloud Pub/Sub for asynchronous processing
  - Implement job priority levels based on subscription tier
  - Create retry mechanism with exponential backoff (max 3 attempts)

🤖 AI/ML Engineers

- **Document Processing Pipeline**:
    - **PDF Processing**:
        - Implement PyMuPDF for text extraction with layout preservation
        - Add OCR capability (Tesseract) for scanned documents
        - Create section identification model (Education, Experience, Projects)
        - Build table/list detection for structured data extraction
    - **DOCX Processing**:
        - Use python-docx with custom parsers for maintaining structure
        - Extract embedded images and tables with structural context
        - Preserve formatting information for confidence scoring
    - **NLP Entity Extraction**:
        - Fine-tune spaCy model for technical resume terminology with custom entity types
        - Create skill taxonomy matcher with 5000+ technical skills and synonyms
        - Implement candidate name extraction with >95% accuracy using custom NER model
        - Design regex patterns for all common contact information formats
        - Create named entity linking for technologies to standardized terms
    - **Link Detection System**:
        - Implement regex patterns for all variations of GitHub URLs:
            - `github.com/username`
            - `github.com/username/repository`
            - `www.github.com/username`
            - `https://github.com/username`
        - Create verification system for GitHub usernames against API
        - Build context-aware link classification (project links vs. reference links)
        - Implement similar patterns for LeetCode, Coursera, LinkedIn, etc.

    - **Edge Case Handling**:
        - Build handling for partially valid or broken links
        - Create recovery system for links in image-based resumes
        - Implement confidence scoring for ambiguous extractions
        - Design fallback extraction for non-standard resume formats
- **Metadata Enhancement Module**:
    - **GitHub Profile Enrichment**:
        - Implement GitHub API client with rate limit handling
        - Extract public profile statistics (stars, followers, repositories)
        - Create repository categorization by language and topic
        - Calculate activity metrics (commit frequency, recency)
    - **Context Analysis**:
        - Extract surrounding text context for links (50 chars before/after)
        - Identify claimed contribution levels through semantic analysis
        - Detect project ownership claims vs. contribution claims
        - Score relevance of repositories to claimed skills

📊 **Data Scientist**

- **Schema Design & Optimization**:

GitHub Metrics Schema:

```
{
 repoId: String,
 name: String,
 ownerUsername: String,
 candidateUsername: String,
 isOriginal: Boolean, // not a fork
 stars: Number,
 forks: Number,
 issues: Number,
 createdAt: Date,
 lastUpdated: Date,
 languages: [
  {
   name: String,
   bytesOfCode: Number,
   percentage: Number
  }
 ],
 testCoverage: Number,
 commitCount: Number,
```

```
  candidateCommitCount: Number,
  complexity: {
    cyclomaticMedian: Number,
    cyclomaticMax: Number,
    linesOfCode: Number,
    functionsCount: Number,
    classesCount: Number,
    fileCount: Number,
    documentationRatio: Number
  }
}
```

LeetCode Performance Schema:

```
 {
  username: String,
  profileUrl: String,
  problemsSolved: {
    easy: Number,
    medium: Number,
    hard: Number,
    total: Number
  },
  contestRating: Number,
  badges: [String],
  submissionStats: {
    acceptanceRate: Number,
    topLanguages: [
      {
        language: String,
        problemCount: Number
      }
    ]
  },
  recentActivity: {
    lastActive: Date,
    averageSolvePerWeek: Number
  }
}
```

Certification Schema:

```
 {
  platform: String,
  courseName: String,
  courseUrl: String,
  issueDate: Date,
```

expiryDate: Date,
credentialId: String,
skills: [String],
verificationStatus: String,
credentialMatchScore: Number, // 0-1 score for name match
relevanceScore: Number // 0-1 score for relevance to job }

- **Preliminary Scoring Algorithms**:
  - Define normalization techniques for heterogeneous data sources
  - Create weighted scoring formulas with configurable parameters
  - Design confidence interval calculations for unreliable data points
  - Implement feature extraction pipelines for future ML scoring models

Day 1-2 Technical Implementation Details

1. **Development Environment Setup**:
   - Configure Docker containers for each microservice
   - Set up shared MongoDB Atlas cluster with separate collections
   - Create CI pipeline with GitHub Actions for automated testing

2. **GCP Infrastructure Preparation**:
   - Provision Google Cloud Storage bucket with proper CORS and security policies
   - Set up Cloud CDN for edge caching where appropriate
   - Configure monitoring with Cloud Monitoring and Cloud Logging

3. **API Contract Definition**:
   - Create OpenAPI/Swagger documentation for all endpoints
   - Define standard error responses and status codes
   - Implement rate limiting and security headers

4. **Integration Testing Strategy**:
   - Create test dataset with 50+ sample resumes of varying formats
   - Implement integration test suite with Jest and Supertest
   - Set up automated testing environment with mocked services

Day 1-2 Deliverables

- Functional file upload system with comprehensive validation
- Resume parser with >90% accuracy for standard formats
- Complete database schema implemented in MongoDB
- GitHub profile extraction with username verification
- Integration tests covering critical paths
- API documentation with Swagger/OpenAPI

🗓️ Days 3-4: GitHub Code Analyzer + Certification Link Verifier

Detailed Technical Specifications

🤖 AI/ML Engineers

- **GitHub Repository Analysis System**:
    - **Repository Acquisition Pipeline**:
        - Create secure GitHub API client with OAuth integration
        - Implement repository cloning with depth limiting for large repos
        - Set up temporary storage with automatic cleanup
        - Add detection for private/invalid repositories with graceful handling
    - **Code Structure Analysis**:
        - Implement directory depth analyzer with statistics generation
        - Create language-specific parsers for popular languages:
            - Python: AST-based analysis with `ast` module
            - JavaScript: Esprima/Babel parser integration
            - Java: JavaParser implementation
            - Go: Custom parser for Go modules
        - Calculate directory structure complexity metrics:
            - Max/avg directory depth
            - File count distribution
            - Directory branching factor
    - **Code Quality Assessment**:
        - **Test Coverage Detection**:
            - Identify test files across frameworks:
                - Python: `test_*.py`, `*_test.py`, pytest fixtures
                - JavaScript: `*.test.js`, `*.spec.js`, Jest/Mocha patterns
                - Java: JUnit patterns, testNG patterns
            - Calculate test-to-production code ratio
            - Detect presence of CI configuration files (GitHub Actions, Jenkins, Travis)

        - **Modularity Analysis**:
            - Compute average function size in lines/statements
            - Calculate function parameter counts and complexity
            - Measure class/file cohesion using semantic grouping
            - Generate dependency graphs for modules

        - **Complexity Metrics**:
            - Implement cyclomatic complexity calculator for all languages
            - Calculate Halstead metrics for volume/difficulty
            - Generate maintainability index scores
            - Create weighted aggregate quality score based on industry benchmarks
    - **Code Originality Assessment**:

        - **Fingerprinting Algorithm**:
            - Generate normalized code fingerprints resistant to formatting changes
            - Create file-level hashes based on AST structure
            - Implement fuzzy matching for near-duplicate detection
        - **Repository Comparison**:
            - Build comparison database of popular public repositories
            - Implement locality-sensitive hashing for efficient comparison
            - Calculate originality score based on unique code percentage

- **Certification Verification System**:
    - **Platform-specific Crawlers**:
        - **Coursera**:
            - Implement headless browser automation with Playwright
            - Navigate certificate verification flows with multiple retry strategies
            - Extract certificate metadata with structured selectors
        - **Udemy**:
            - Create certificate URL parser for Udemy's verification system
            - Implement token extraction for certificate API access
            - Design fallback screen capture for direct verification
        - **AWS/Microsoft/Google Certifications**:
            - Build certificate ID extractors for standardized formats
            - Implement verification API clients where available
            - Create OCR-based extraction for image-only certificates
    - **Data Extraction Pipeline**:
        - Implement name extraction with normalization (handling prefixes, suffixes)
        - Create course title parser with standardization to master database
        - Extract date information with multiple format support
        - Capture certification level and specialization details

- **Verification Algorithms**:
  - **Name Matching**:
    - Implement Levenshtein distance calculator with configurable thresholds
    - Create name normalization (handling initials, order variations)
    - Design fuzzy name matching with confidence scoring

  - **Course Relevance Assessment**:
    - Build skill-to-course mapping database (10,000+ courses)
    - Implement embedding-based relevance scoring using sentence transformers
    - Calculate relevance score between certification and candidate's claimed skills

🔧 **Fullstack Developers**

- **GitHub Analysis UI Components**:

  - **Repository Overview Component**:
    - Create expandable repository cards with:
      - Repository name, description, stars, and fork count
      - Language breakdown with color-coded visualization
      - Last activity timestamp with relative time display
    - **Metrics Visualization**:
      - Implement complexity badge (A/B/C) with tooltip explanation
      - Create radar chart for code quality dimensions
      - Design progress bars for test coverage visualization
    - **Code Quality Tags**:
      - "Tested": Green checkmark for repositories with >20% test files
      - "Modular": Badge for code with good function/class distribution
      - "Original": Indicator for code with high originality score
      - "Active": Badge for repositories updated in last 3 months
  - **GitHub Profile Summary Component**:
    - Create contribution calendar visualization (similar to GitHub)
    - Implement language proficiency chart based on repository analysis
    - Design activity timeline with commit frequency visualization

- **Certification Verification UI**:
    - **Certificate Card Component**:
        - Create card layout with:
            - Platform logo with proper sizing and positioning
            - Course name with truncation handling for long titles
            - Issue date with proper formatting
            - Verification status badge with color coding

        - **Verification Status Indicators**:
            - "Verified": Green checkmark with tooltip showing match confidence
            - "Partial Match": Yellow warning with explanation (name or date mismatch)
            - "Not Verified": Red X with specific failure reason
            - "Pending": Gray clock icon for in-progress verification

        - **Detailed Certificate View**:
            - Expandable card with full certificate details
            - Skill tags extracted from course content
            - Platform credibility indicator

📊 Data Scientist

- **GitHub Quality Score Algorithm**:
    - Define weighted formula:
        - Code Quality: 30%
            - Test coverage: 10%
            - Modularity: 10%
            - Complexity: 10%
        - Code Originality: 40%
            - Unique code percentage: 25%
            - Novel implementation approaches: 15%
        - Contribution Metrics: 30%
            - Commit frequency: 10%
            - Commit volume: 10%
            - Project longevity: 10%
    - Create adaptive weighting system based on:
        - Repository age and activity
        - Language and domain context
        - Repository size and complexity
    - Implement score normalization techniques:
        - Z-score normalization across candidate pool
        - Percentile ranking within domain groups
        - Min-max scaling with outlier handling

- **Certificate Scoring Algorithm**:
    - **Platform Credibility Weighting**:
        - Tier 1 (Highest): AWS, Google, Microsoft official certs
        - Tier 2: Coursera, edX, LinkedIn Learning
        - Tier 3: Udemy, Pluralsight, FreeCodeCamp
        - Tier 4: Lesser-known platforms
    - **Verification Confidence Impact**:
        - Perfect name match: 100% of available points
        - Partial name match: 50-90% based on similarity score
        - Failed verification: 0-10% based on cause
    - **Comprehensive Formula**:
        - `Certificate Score = Platform Weight * Verification Confidence * Relevance Score * Recency Factor`

Day 3-4 Integration Points

1. **Data Flow Architecture**:
    - GitHub analyzer receives repositories from resume parser
    - Certificate verifier receives URLs from resume parser
    - Score calculator receives inputs from both analyzers
2. **Shared Services**:
    - Implement shared logging service with structured JSON output
    - Create centralized error handling with categorization
    - Set up shared caching layer for API responses

Day 3-4 Deliverables

- Functional GitHub repository analyzer with language-specific parsing
- Comprehensive code quality assessment system
- Certificate verification system supporting major platforms
- UI components for displaying GitHub and certification data
- Initial scoring models with normalization techniques
- Integration tests for all components

🗓️ Days 5-6: External Platform Integration (LeetCode, Figma, etc.)

Comprehensive Technical Specifications

🤖 AI/ML Engineers

- **LeetCode Profile Analyzer**:
    - **Data Acquisition**:
        - Implement web scraping with anti-detection measures
        - Create GraphQL client for LeetCode API (where applicable)
        - Build session management for authenticated scraping
        - Set up IP rotation for rate limit avoidance
    - **Profile Data Extraction**:
        - Extract problem-solving statistics with category breakdown
        - Capture contest history and rating progression
        - Calculate percentile ranking within LeetCode community
        - Retrieve recent activity patterns and solve streaks
    - **Problem Analysis**:
        - Categorize problems by difficulty and topic
        - Map problems to relevant technical skills
        - Calculate topic-specific proficiency scores
        - Analyze solution quality metrics where available

- **Design Platform Integration**:
    - **Figma Integration**:
        - Implement OAuth flow for Figma API access
        - Create project inventory extraction with metadata
        - Build component/style analysis for skill assessment
        - Implement interaction complexity analyzer
    - **Dribbble Portfolio Analyzer**:
        - Create public portfolio scraper with pagination handling
        - Extract project statistics (likes, views, comments)
        - Implement image analysis for style classification
        - Calculate engagement metrics and community standing
    - **Design Skill Assessment**:
        - Extract tags and categories from projects
        - Map tags to standardized design skill taxonomy
        - Implement frequency analysis for skill specialization
        - Calculate skill depth based on project complexity

- **LinkedIn Profile Analyzer**:
    - **Profile Data Extraction**:
        - Implement profile scraper with section detection
        - Extract work experience with duration calculation
        - Capture education history and credentials
        - Retrieve endorsements and skill validations
    - **Activity Analysis**:
        - Calculate post frequency and engagement metrics
        - Analyze content relevance to claimed expertise
        - Measure network size and quality
        - Assess profile completeness and professionalism
    - **Validation Algorithms**:
        - Cross-reference resume claims with LinkedIn data
        - Calculate consistency score between platforms
        - Flag discrepancies in employment history
        - Generate trust score based on profile verification

🔧 **Fullstack Developers**

- **Platform-Specific UI Components**:
    - **LeetCode Profile Section**:
        - Create problem-solving statistics card with:
            - Donut chart for problem difficulty distribution
            - Line chart for solving progress over time
            - Badge showcase with tooltip explanations
            - Contest rating with percentile indicator
        - **Topic Proficiency Visualization**:
            - Radar chart for algorithm category strengths
            - Progress bars for language-specific problem counts
            - Visual indicator for highest difficulty solved
    - **Design Platform Showcase**:
        - **Figma/Dribbble Gallery**:
            - Masonry layout for project thumbnails
            - Hover states with engagement statistics
            - Modal for detailed project view
            - Tag cloud for design specialization
        - **Metrics Dashboard**:
            - Engagement statistics with benchmark comparisons
            - Community standing visualization
            - Growth chart for account activity

    - **LinkedIn Integration UI**:
        - Profile summary card with verification status
        - Experience timeline with position details
        - Endorsement visualization with skill breakdown
        - Network quality indicator with industry benchmarks

- **Platform Status Indicators**:
    - Create unified status badge system:
        - "Not Found": Gray icon with tooltip for missing profiles
        - "Private Profile": Lock icon for inaccessible data
        - "Limited Access": Partial data indicator
        - "Verified": Green checkmark for complete access
    - **Score Visualization**:
        - Implement circular progress indicators for platform scores
        - Create comparative bar charts across platforms
        - Design detailed score breakdown panels
        - Add peer comparison visualizations where relevant

📊 Data Scientist

- **Multi-Source Scoring Engine**:
    - **Technical Skill Score**:
        - Formula: `Tech Score = (GitHub 50%) + (LeetCode 50%)`
        - GitHub components:
            - Code quality: 20%
            - Project relevance: 15%
            - Contribution history: 15%
        - LeetCode components:
            - Problem difficulty distribution: 20%
            - Topic coverage: 15%
            - Contest performance: 15%
    - **Creative Skill Score**:
        - Formula: `Creative Score = (Figma 60%) + (Dribbble 40%)`
        - Figma components:
            - Design complexity: 20%
            - Component structure: 20%
            - Interaction design: 20%
        - Dribbble components:
            - Visual appeal: 15%
            - Community engagement: 10%
            - Portfolio diversity: 15%

- **Professional Presence Score**:
  - Formula: `Social Score = LinkedIn Activity + Profile completeness`
  - Activity components:
    - Post frequency: 15%
    - Engagement metrics: 15%
    - Relevance to expertise: 20%
  - Profile components:
    - Completeness: 15%
    - Recommendation quality: 15%
    - Network relevance: 20%

- **Normalization Pipeline**:
  - Implement min-max scaling for each score category
  - Create percentile ranking system across candidate pool
  - Design outlier detection and handling
  - Calculate confidence intervals for each score

Day 5-6 Integration Requirements

1. **Authentication Management**:
   - Implement secure credential storage for API access
   - Create rate limiting and backoff strategies
   - Design error recovery with alternative data sources
2. **Data Consistency Enforcement**:
   - Create unified candidate profile from multiple sources
   - Implement conflict resolution for contradictory data
   - Design data freshness tracking and update triggers

Day 5-6 Deliverables

- Functional platform-specific crawlers for all targeted sites
- Comprehensive UI components for displaying platform data
- Multi-dimensional scoring engine with normalized outputs
- Advanced analytics framework for candidate evaluation
- Integration tests for data flow across components
- Privacy-compliant data handling system

🗓️ Day 7: Candidate Scoring & Leaderboard MVP

Comprehensive Technical Blueprint

🤖 AI/ML Engineers + Data Scientist

- **Aggregate Scoring Pipeline Architecture**:
  - **Data Integration Layer**:
    - Create unified candidate profile from all sources
    - Implement data quality assessment with completeness metrics
    - Design conflict resolution strategies with source priority hierarchy
  - **Feature Engineering Framework**:
    - Implement domain-specific feature extractors
    - Create cross-platform correlation features
    - Design temporal feature extraction for activity recency
  - **Comprehensive Scoring System**:
    - **Primary Score Components**:
      - Technical Ability: 40%
        - Code quality (GitHub): 15%
        - Problem-solving (LeetCode): 15%
        - Technical breadth: 10%
      - Verification Confidence: 30%
        - Identity consistency: 10%
        - Claim verification: 10%
        - Experience validation: 10%
      - Platform-specific Excellence: 30%
        - GitHub quality metrics: 10%
        - Certification portfolio: 10%
        - Design/creative indicators: 10%
    - **Secondary Adjustment Factors**:
      - Activity recency multiplier: 0.8-1.2
      - Profile completeness bonus: 0-10%
      - Inconsistency penalty: 0-20%
  - **Output Generation**:
    - Create standardized candidate profile with detailed scoring breakdown
    - Generate detailed reports highlighting strengths and improvement areas

🔧 **Fullstack Developers**

- **Advanced Leaderboard UI System**:
    - **Candidate Card Component**:
        - Create responsive card layout with candidate info and score visualization
        - Implement platform badges with verification status
        - Add quick action buttons (view profile, save, flag)
    - **Filtering and Search System**:
        - Implement advanced filtering by role, tech stack, score threshold
        - Build full-text search across candidate data
        - Create typeahead suggestions based on available candidates
    - **Sorting and Organization**:
        - Create flexible sorting options by various score components
        - Implement grouping capabilities by skill cluster and experience level
    - **Pagination and Performance**:
        - Implement virtual scrolling for large candidate pools
        - Create server-side pagination with customizable page size
        - Design skeleton loaders for improved UX during data fetching

- **Detailed Candidate Profile**:
    - **Profile Overview Section**:
        - Create comprehensive candidate summary with contact info and skills
        - Build score dashboard with interactive charts
    - **Platform Data Integration**:
        - Implement tabbed interface for platform-specific data
        - Create expandable evidence panels for claims

☁️ **DevOps Engineer (GCP Focus)**

- **Infrastructure Architecture**:
    - **Containerization Strategy**:
        - Create Docker images for each microservice
        - Configure Google Kubernetes Engine for production deployment
    - **Scaling Configuration**:
        - Design horizontal scaling policies based on queue length
        - Implement caching strategies using Memorystore
        - Configure resource limits for all containers
    - **Network Security**:
        - Set up VPC Service Controls for secure communication
        - Implement IAM policies for service authentication
        - Configure Cloud Armor for public-facing components

- **Monitoring Framework**:
    - Implement comprehensive logging with structured format
    - Create custom metrics for pipeline performance
    - Design alerting system with severity levels
    - Set up Cloud Monitoring dashboards for key metrics
- **API Gateway Configuration**:
    - Create unified API gateway using Cloud Endpoints
    - Generate API documentation with OpenAPI/Swagger
    - Implement authentication and authorization

Day 7 Deliverables

- Fully operational scoring pipeline with component weights
- Interactive leaderboard with filtering and pagination
- Comprehensive API documentation with examples
- Monitoring and alerting system for operational issues
- Integration test suite covering end-to-end flows

⚠️ CRITICAL COLLABORATION MANDATE

ATTENTION ALL TEAM MEMBERS: Week 2 development establishes our core product functionality and is CRITICAL to project success. The following collaboration requirements are NON-NEGOTIABLE:

Communication Protocol

- **Continuous Presence**: All team members must be communicating and collaborating
- **Status Updates**: Post component status updates every day
- **Stand-ups**: Mandatory 15 to 30-minute stand-ups at discord with NO EXCEPTIONS

Cross-Team Integration Requirements

- **Frontend + Backend**: API contract must be finalized by Day 1 EOD with mock endpoints deployed
- **AI/ML + Data Science**: Model interfaces must be defined by end of Day 2 with test harnesses
- **All Teams**: Integration testing sessions required at 4 PM daily

Code Quality Gates

- **Pre-commit Testing**: All commits must pass automated tests before PR submission
- **Code Reviews**: Maximum 4-hour turnaround for all PRs with at least two reviewers from different teams
- **Test Coverage**: Minimum 80% code coverage for new features, 70% for bugfixes

Dependency Management

- **Frontend Dependencies**: Must clear packages with DevOps before installation
- **Backend Services**: Any new service must be containerized immediately
- **Third-party APIs**: All external service integration requires fallback implementation

🎯 Week 2 Success Criteria

1. **Resume Parser Accuracy**:
    - 85% accuracy in extracting GitHub links
    - 90% accuracy in identifying candidate name and contact info
    - Processing time <30 seconds per document

2. **GitHub Analysis Quality**:
    - Code quality metrics align with manual assessment in >80% of cases
    - Repository fetching success rate >95% for public repositories
    - Language detection accuracy >90%

3. **Certification Verification Performance**:
    - 80% success rate in accessing certification platforms
    - 90% accuracy in extracting certificate metadata
    - Name matching achieves >85% accuracy including partial matches

4. **External Platform Integration**:
    - Successful data retrieval from at least 3 platforms (LeetCode, LinkedIn, Figma/Dribbble)
    - 90% profile match rate when correct username is provided
    - Graceful handling of private profiles and rate limits

5. **Scoring System Reliability**:
    - Score calculation completes in <5 seconds per candidate
    - Consistent results with <5% variation on repeated scoring
    - Score components demonstrably correlate with manual assessment

6. **Overall System Performance**:
    - End-to-end processing (upload to leaderboard) <3 minutes per candidate
    - API endpoints maintain <200ms response time under load
    - System handles 50+ concurrent users without degradation