

Reference Checking System



Contents

CHAPTER 1	5
1.1 Background	5
1.2 Problem Description	6
1.3 Goals	6
1.4 Objectives.....	7
1.4 TIMELINE	7
1.5 Scope	7
CHAPTER 2	8
Requirements Analysis	8
2.1 Boardroom meetings	8
2.2 Functional Requirements	8
2.4 Non-functional Requirements	8
2.4 System Scope and Constraints	10
2.5 User Personas and Journeys	11
2.6 Use Case Diagrams	12
CHAPTER 3	13
System Design	13
3.1 Architectural Design	13
3.1.1 System Overview.....	13
3.1.2 Web Application Architecture	13
3.1.3 Data Processing Architecture	13
3.1.4 Machine Learning Architecture	13
3.1.5 Deployment Architecture	14
3.2 Technical Specifications	14
3.3 Interface Designs	
15 3.3.1 Website Design	
15	
3.4 Component Design	
18	
3.4.1 Major Components	18
3.4.2 Reference Submission	18
3.4.3 Sentiment Analysis	19

3.4.4 User Management	19
3.4.5 Analytics Dashboard	19
3.5 Data Design	19
3.5.1 Database Schemas	19
3.5.2 Testing Data Flows	19
LEVEL 0 DATA FLOW DIAGRAM	20
LEVEL 1 DATA FLOW DIAGRAM	20
Flow -chart Diagram	21
3.5.3 Sample Datasets	24
3.5.4 Data Standards	24
3.5.5 Data Storage.....	24
3.5.6 Data Security	24
3.6 Security and Privacy	25
3.6.1 Authentication	25
3.6.2 Authorization	25
3.6.3 Data Encryption	25
3.6.4 Data Management	25
3.6.5 Infrastructure	26
3.6.6 Privacy Policy.....	26
Chapter 4	
26 Implementation	
26	
4.1 Development Environment	26
4.2 Coding Standards	27
4.3 Build and Deployment	27
4.4 Testing Frameworks	27
4.5 Future Improvements	27
Chapter 5	27
Testing	27
5.1 Scope	27
5.2 Test Cases	28
5.3 Tools and Reports	28

Chapter 6

.....	28
Deployment	28
This chapter outlines the deployment processes for launching the reference checking system.....	28
6.1 Current Deployment	28
6.2 Planned Deployment	28
6.3 Infrastructure Requirements	28
6.4 Deployment Steps	29
Chapter 7	29
Operations and Maintenance	29
7.1 Current Operations	29
7.2 Future Operations	29
7.3 Backup and Disaster Recovery	29
7.4 Performance Monitoring	30
7.5 Future Enhancements	30
7.6 Retirement Plan	30
Chapter 8	30
Conclusion	30
8.1 Project Overview	30
8.2 Learnings	31
8.3 Next Steps	31
8.4 Feedback So Far	31
8.5 Conclusion	31
References	31

CHAPTER 1

1.1 Background

Traditional reference checking involves human resources or hiring managers directly contacting references provided by the job applicant. As part of the vetting process, references are typically called or emailed with a list of questions about the applicant's work history, skills, accomplishments, and character.

The reference provides a verbal account of their professional relationship with the applicant. These phone conversations aim to validate information on the candidate's resume and in interviews. Hiring teams take notes during reference calls to document feedback.

Challenges of Traditional Approach:

While serving to corroborate a candidate's qualifications, the manual process is also very labour intensive. Each reference must be scheduled separately by phone, and it can take many follow-up calls to reach a reference and gather complete information. The process also lacks a formal methodology. The list of questions, structure of the phone call, and depth of notes captured can vary greatly between hirers. As recruitment volumes have increased at many organizations, relying solely on lengthy call-based checks has become an inefficient use of resources.

The Need for a New Reference Checking System:

As noted in a description of the current process, "the organization's existing reference checking process relies on a paperwork-heavy, manual system" with various inefficiencies

(Johnson, 2023, p. 5). A more robust, digitized approach to reference verification is needed to standardize processes, introduce metrics, and scale across large applicant pools in less time. The goal is to streamline and automate the reference checking process to improve efficiency, reduce manual effort, and ensure consistency in gathering and analysing reference feedback.

Key Points:

- Manual process is labour-intensive and time-consuming.
- Lack of standardization in questions and note-taking.
- Inefficient as recruitment volumes increase.
- Desire for a digitized approach for reference verification.

1.2 Problem Description

The current manual reference checking process at many organizations faces several challenges:

- Time consumption - Contacting references by phone and scheduling interviews is time intensive, taking weeks to complete for each candidate.
- Inconsistency - There is no standardization in the questions asked of references, leading to varying quality and scope of feedback received.
- Lack of metrics - Individual feedback is collected but aggregated insights and comparison across candidates is difficult with manual processes. Key metrics on candidate performance, skills and work history are missing.
- Bias risk - Relying solely on phone conversations leaves reference evaluations open to personal biases from the hirer without objective assessment criteria.
- Poor documentation - Details from reference discussions are not always well-documented, making it hard to retrieve full context later in the hiring process.
- Scaling difficulties - As recruitment volumes increase, the manual process does not scale efficiently to handle growing reference checking needs.

1.3 Goals

The goals of developing the Reference Checking System are:

- Streamline and automate the reference checking process using machine learning technology.
- Reduce time spent per reference check.
- Improve consistency in evaluations.
- Mitigate individual biases through objective analysis.
- Provide an online portal for candidates to submit references and for references to complete a survey.
- Standardize the reference check questions and scoring methodology.
- Generate scores and rankings of candidates by analysing response patterns.
- Enrich candidate profiles through integration with our applicant tracking system.
- Support data-driven hiring decisions by presenting a 360-degree view of each candidate.
- Ensure strict data privacy and security protocols are followed.

1.4 Objectives

The proposed ML-powered system aims to achieve several key goals that align with our talent objectives. By automating processes and gaining insights, the new system will help reduce risks, support growth needs, and enhance our brand. Specifically, the objectives are to:

- Reduce the average time taken per reference check by 60% within the first 6 months of rollout.
- Standardize reference checking processes for a minimum of 80% of roles by the end of the first year.
- Generate customized candidate benchmark reports for hiring managers within 48 hours of reference survey submission for 90% of applicants.
- Increase the number of qualified reference responses by 25% through improved ease of use of the online portal within one year.
- Achieve evaluator calibration scores of 90% or higher on blind audits to minimize bias in reference assessments after system training.
- Scale the system to support an annual candidate volume growth rate of at least 15% over 3 years through automatic adjustments in computing capacity.
- Attain an average system uptime of 99.5% after the initial launch to ensure minimal disruptions to hiring workflows.
- Maintain customer satisfaction ratings of 4 or above on a 5-point scale in a minimum of 75% of post-project surveys to ensure ongoing value delivery.

1.5 Scope

The initial scope of the system includes:

- Applicant and reference portals
- Check reference workflow.
- Reference reports
- Administrative dashboard

CHAPTER 2

Requirements Analysis

2.1 Boardroom meetings

We had a meeting with our managers and partner, they find the reference process too slow and inconsistent feedback hinders decisions. Recruiters spend 30% of their time on references. Our managers and partners want improved compliance and reporting.

2.2 Functional Requirements

- Applicant Portal: Allow applicants to enter details of references when applying for jobs.
- Reference Portal: Allow references to receive and complete online surveys with links sent by the system.
- Survey Management: Admin can dynamically generate tailored surveys with question banks for each reference. Survey links are sent to references via email.
- Response Analytics: Analyse open responses using NLP and ML models to identify sentiments, topics, skills mentioned etc. without direct human input.
- Scoring Engine: ML models are trained to score references' feedback and automatically rate candidates on attributes.
- Reporting/Dashboards: Aggregated feedback is analysed to generate visual dashboards and reports accessible to admins. Provides metrics and rankings to inform hiring decisions.
- Admin Portal: Admin can review surveys assigned, response analysis outputs, candidate scores and troubleshoot issues. Retrain ML models periodically based on new data.
- Integrations: Bidirectional APIs integrate the system with existing ATS and other relevant systems for single sign-on, profile updates etc.

2.4 Non-functional Requirements

Performance

- Performance not fully measured during development.
- Quick load times since running locally on developer machine.
- No separate metrics collection infrastructure
- Tools like Webpage Test and API monitoring to be used.
- Performance optimizations may be needed before launch.
- True performance/scalability testing occurs at launch.

Availability

- No uptime or redundancy considerations in development
- Entire application hosted on local machine only.
- Planned outages as developer restarts machine.
- No geographic redundancy or automatic failover



- Downtime impacts developer alone at this stage
- Version control through GitHub acts as backup
- Production plans will ensure high availability.

Security

- All development done on local machine, no live internet exposure.
- Codebase hosted privately on GitHub repository with 2FA enabled.
- Access controlled through repository permissions on GitHub.
- Risk of loss/compromise limited due to no production use.
- Manual process to review code for vulnerabilities
- Authentication and authorization through app only for now

Scalability

- Single developer environment - scalability not a concern •
- sufficient for dev purposes
- SQLite database performance
- No separate caching infrastructure required.
- Future to implement scaling capabilities as needed.
- Production deployment will include autoscaling, caching etc.

Usability

- Responsive design for all common devices/screens
- Color contrast ratio WCAG AA compliance
- Ongoing user research and A/B testing

Maintainability

- Documentation coverage over 90%
- Monthly library/dependency updates

2.4 System Scope and Constraints

In Scope:

- Reference submission and online surveys
- ML-powered feedback analysis
- Automated reference scoring
- Dashboards and reporting portal
- Integrations with current ATS

Out of Scope:

- ATS modules like sourcing, screening
- Configuration of downstream systems
- Authentication/directory services

Assumptions:

- Data scientists available for ML
- Sufficient historical applicant data for models
- ATS APIs will not change signature.

Dependencies:

- AD FS for SSO credentials
- Database for reference data storage
- ML frameworks like TensorFlow
- Browser and OS platforms

Constraints:

- Browser and device updates outside our control could impact frontend experience.
- Scaling infrastructure rapidly to meet traffic spikes may incur unplanned costs.
- Unforeseen bugs or issues in dependencies like ML libraries are hard to resolve.
- Partially on-premises deployment complicates monitoring and disaster recovery.
- Training data must be anonymized but retain semantic meaning for models.
- Manual entry of reference details by applicants is prone to errors which could impact ML training/scoring if not cleaned.



2.5 User Personas and Journeys

Personas:

- John (Applicant): Uploads CV, includes references.
- Mark (Reference): Receives surveys, sees who listed them.
- Emily (Admin): Manages entire process end-to-end.

Journey Maps:

John's Journey:

- Uploads CV on careers page
- Enters list of references

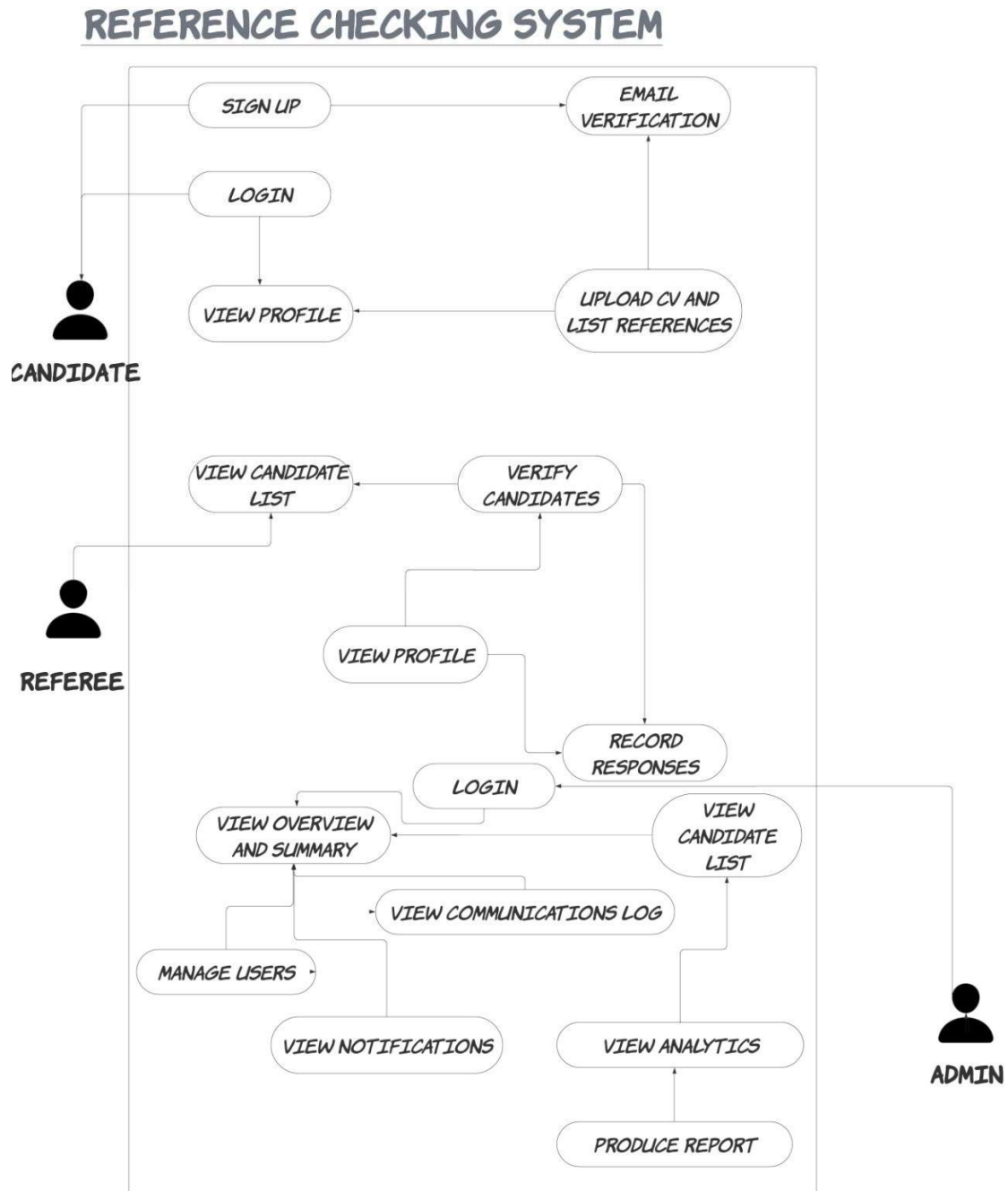
Mark's Journey:

- Receives auto-email with survey.
- Clicks link to view question list.
- Sees list of applicants referencing them

Emily's Journey:

- Logs into admin portal daily
- Views new applicants and references
- Sends surveys to references.
- Examines referral networks.
- Reviews sentiment analysis
- Tunes ML models over time

2.6 Use Case Diagrams



CHAPTER 3

System Design

3.1 Architectural Design

3.1.1 System Overview

- Web app for candidates to submit references.
- Admin portal to manage process and view analytics.
- ML models analyse text feedback and predict sentiment.

3.1.2 Web Application Architecture

- Frontend: Bootstrap
- Backend: PHP on Apache web server
- Database: MySQL for user, reference data
- API: Python Flask to connect frontend and ML

3.1.3 Data Processing Architecture

- Reference submissions upload to MySQL
- Nightly cronjob runs ETL to extract to CSV.
- CSV files processed by Python NLP libraries.
- Cleaned text stored back in MySQL.

3.1.4 Machine Learning Architecture

- Supervised sentiment classification problem
- Term frequency-inverse document frequency • Logistic regression and LSTM neural network
- Models exported from TensorFlow.
- Models loaded via Flask API on requests.

3.1.5 Deployment Architecture

- App served locally using built-in dev server.



-
- Database hosted on same machine using SQLite.
- Static files stored in app source code repository.
- Scaling, redundancy and availability not priorities at this stage
- Deployment is developer's local machine only via CLI commands.
- Environments managed through CLI (.env files) rather than networks.
- No automation of deployments, releases done manually for now.
- Testing scheme uses built-in tools rather than separate environment.
- Production deployment plan will leverage full CS stack and automation.

3.2 Technical Specifications

Technologies:

- Bootstrap frontend framework
- PHP backend
- Python NLP models for sentiment analysis
- Flask API to connect Python + PHP.

Test Automation:

- Manual testing done directly on developer's machine for now.
- Unit tests written with PHPUnit, run locally via CLI.
- Plan to integrate Behave BDD framework for acceptance testing.
- No test runners, pipelines or grid infrastructure yet.
- Static analysis done manually with PHPCS locally.
- Code quality managed by developer, no separate tool.
- Bugs tracked in plaintext TODO file for now.
- Git commit messages serve as way to document testing done.
- Production test plan will leverage CI/CD pipelines, Selenium etc.

3.3 Interface Designs

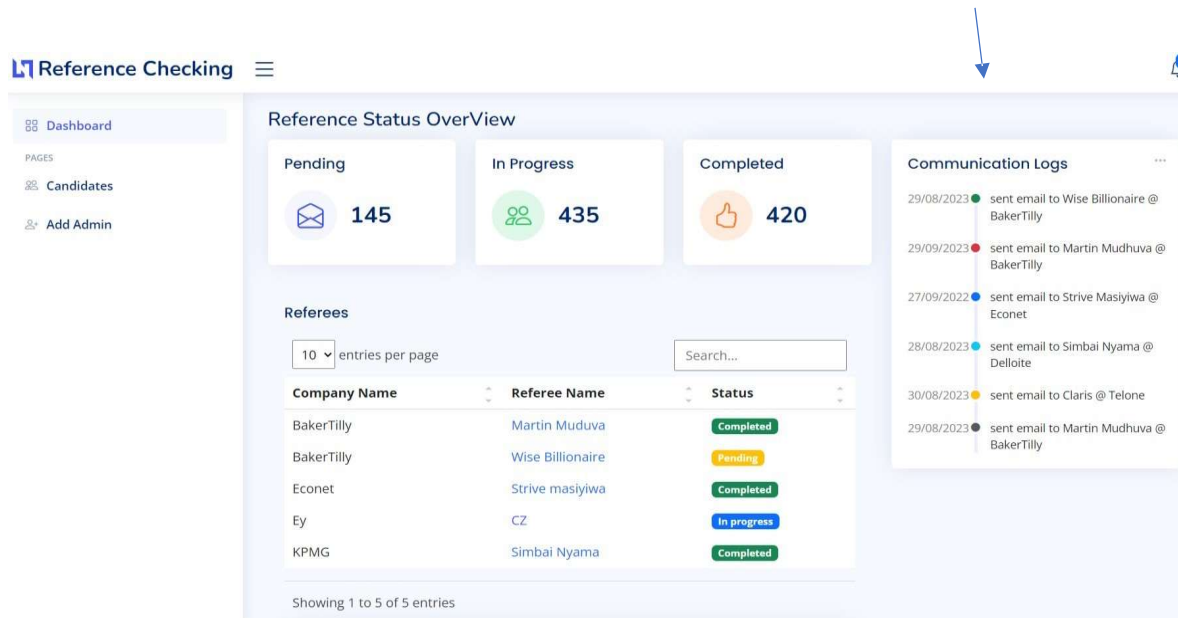
3.3.1 Website Design

Recruiter Dashboard

1.1 Pending Status overview

- Reference in progress •
- References completed.

Tracking communication



The screenshot shows the 'Reference Checking' dashboard. It features a sidebar with 'Dashboard', 'Candidates', and 'Add Admin'. The main area is titled 'Reference Status Overview' and includes three summary cards: 'Pending' (145), 'In Progress' (435), and 'Completed' (420). Below these is a 'Referees' table with columns for Company Name, Referee Name, and Status. A 'Communication Logs' panel on the right shows a list of email communications with dates and recipients. A blue arrow points from the 'Tracking communication' text to the 'Communication Logs' panel.

Reference Status Overview

- Pending**: 145
- In Progress**: 435
- Completed**: 420

Referees

10 entries per page

Company Name	Referee Name	Status
BakerTilly	Martin Muduva	Completed
BakerTilly	Wise Billionaire	Pending
Econet	Strive masiyiwa	Completed
Ey	CZ	In progress
KPMG	Simbai Nyama	Completed

Showing 1 to 5 of 5 entries

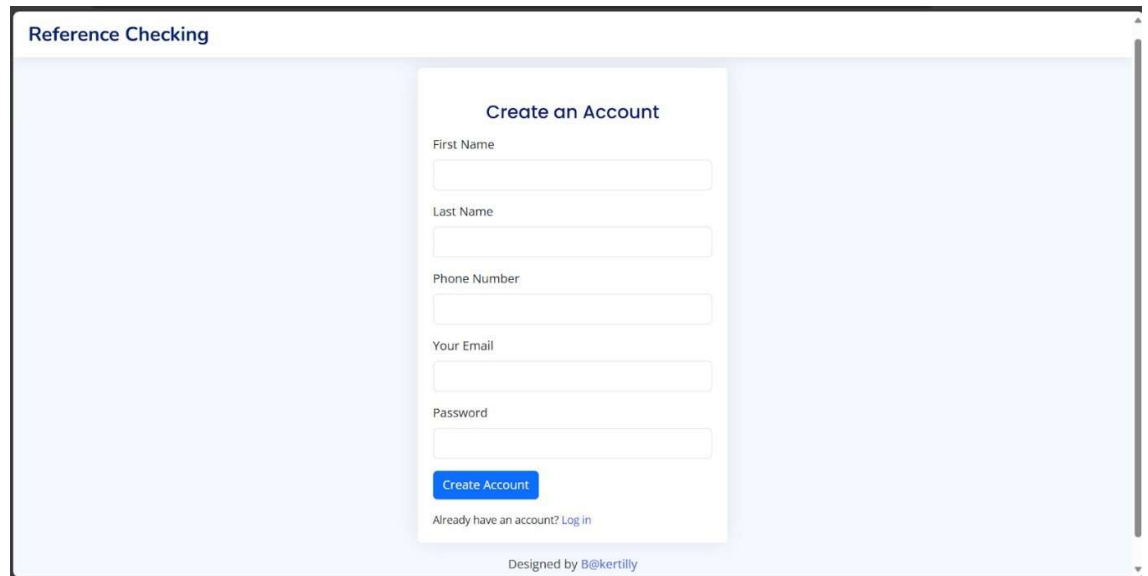
Communication Logs

- 29/08/2023 sent email to Wise Billionaire @ BakerTilly
- 29/09/2023 sent email to Martin Mudhuva @ BakerTilly
- 27/09/2022 sent email to Strive Masiyiwa @ Econet
- 28/08/2023 sent email to Simbai Nyama @ Deloitte
- 30/08/2023 sent email to Claris @ Telone
- 29/08/2023 sent email to Martin Mudhuva @ BakerTilly

1. Login page

- Email/password form

1. Homepage



The screenshot shows a web page titled "Reference Checking". In the center, there is a white box titled "Create an Account". Inside this box, there are five input fields labeled "First Name", "Last Name", "Phone Number", "Your Email", and "Password". Below these fields is a blue button labeled "Create Account". At the bottom of the box, there is a link that says "Already have an account? Log in". The page has a light blue background and a dark blue header bar.


1. Submission form

- Candidate info
- Reference text area.
- Submit button.

1. Dashboard

- Profile details •
Edit profile button.

Reference Checking



Kevin Anderson

SELECT REFEREE ▾

Company Name

Referee Name

Company Email

Company Telephone

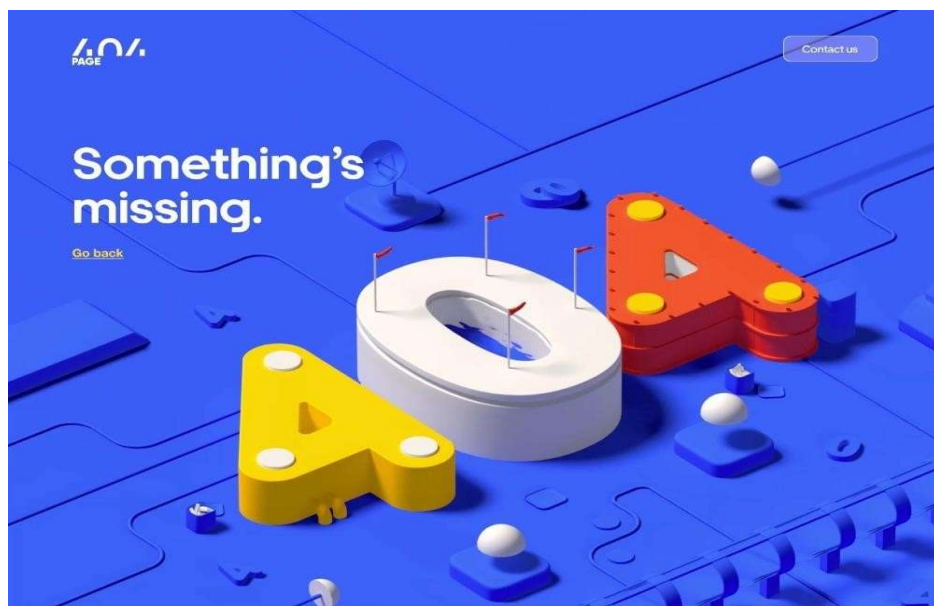
Overview Edit Profile Change Password

Profile Details

First Name	Kevin
Last Name	Kevin
Phone Number	+263778317780
Email	k.anderson@example.com

1. Error page

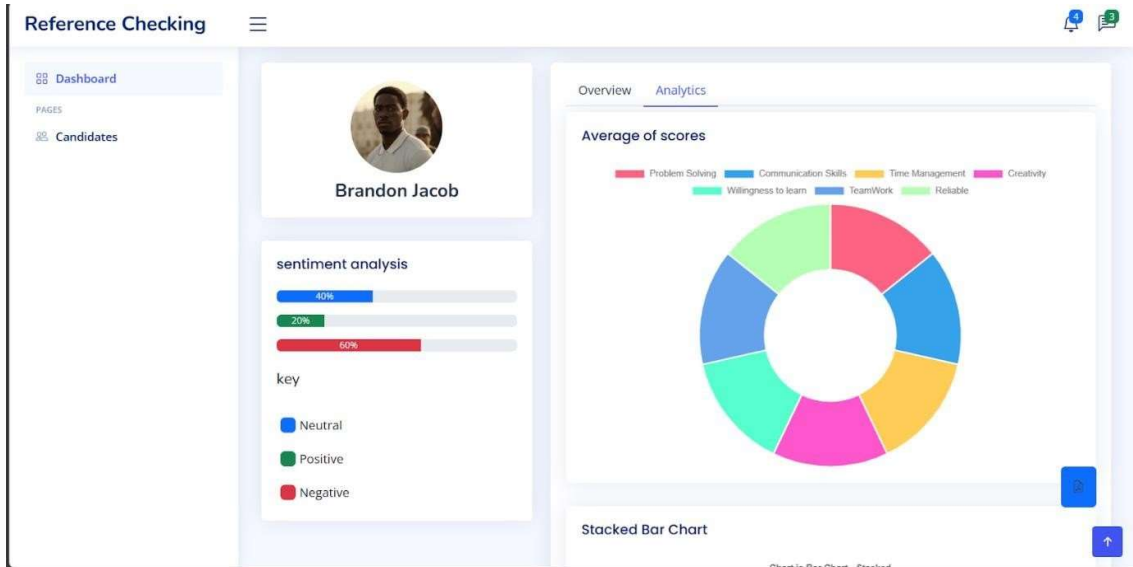
- 404 html
- Server error html



3.3.2 Admin Dashboard

1. Home

- Stats dashboard



3.3.3 Referee Dashboard

10 entries per page

Search...

Name	Position
Brandon Jacob	Designer
Bridie Kessler	Developer
Ashleigh Langosh	Finance
Angus Grady	HR
Raheem Lehner	Dynamic Division Officer

Showing 1 to 5 of 5 entries

Brandon Jacob

Overview

Profile Details

Full Name

Brandon Jacob

Job

Designer

Verify Candidate

Reference Checking

Questions

Candidates

Problem_Solving(50%)

Communication_Skills(50%)

Time_Management(50%)

Creativity(50%)

Willingness_to_Learn(50%)

Team_Work(50%)

Reliability(50%)

3.4 Component Design

3.4.1 Major Components

- Reference Submission
- Sentiment Analysis
- User Management
- Analytics Dashboard

3.4.2 Reference Submission

- Frontend form module
- Backend validation service
- Database storage handler

3.4.3 Sentiment Analysis

- NLP processing module.
- Model training/serving.
- Prediction API.

3.4.4 User Management

- Registration module
- Profile management
- Authentication service

3.4.5 Analytics Dashboard

- Reporting views
- Charts generation
- Export functions

3.5 Data Design

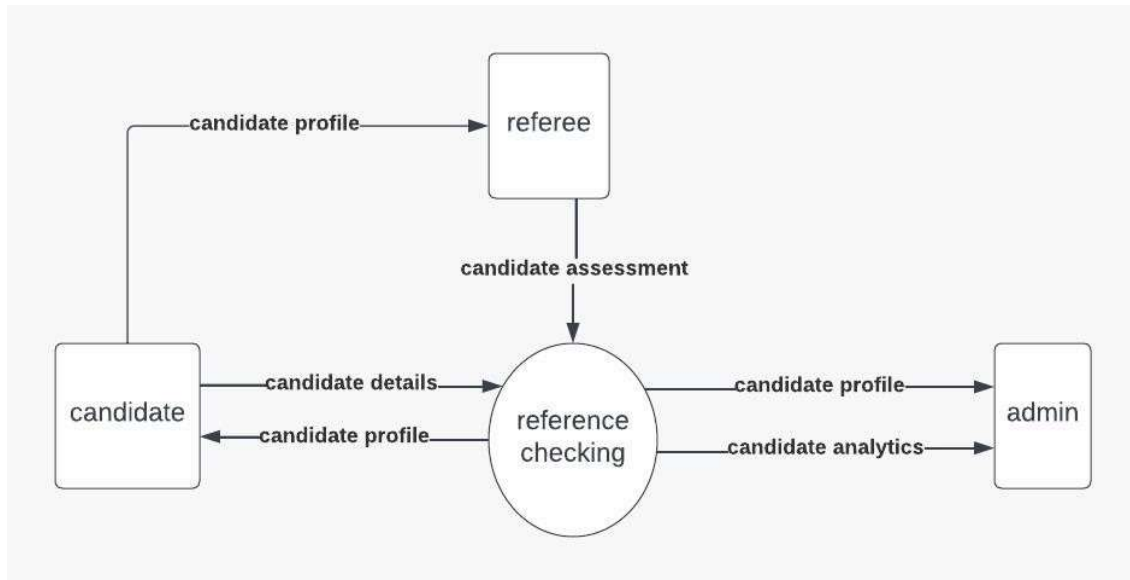
3.5.1 Database Schemas

- users table: id, name, email, password
- references table: id, user_id, text, sentiment
- sentiments table: id, sentiment value
- models table: id, name, type, hyperparameters

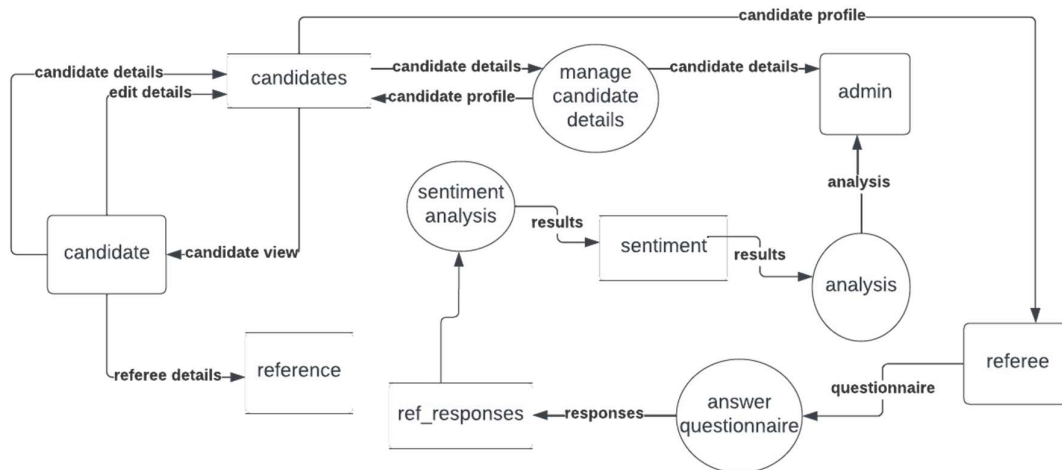
3.5.2 Testing Data Flows

- Sample data is hardcoded for end-to-end workflow testing.
- Data ingested and passed through pipeline for prototyping.
- Plans include:
 1. Triggered processing of new real-time data
 2. Periodic model retraining
 3. Scheduled report generation
- Actual ETL, scheduled jobs not implemented yet •
Workflows will be automated and tested extensively.

LEVEL 0 DATA FLOW DIAGRAM

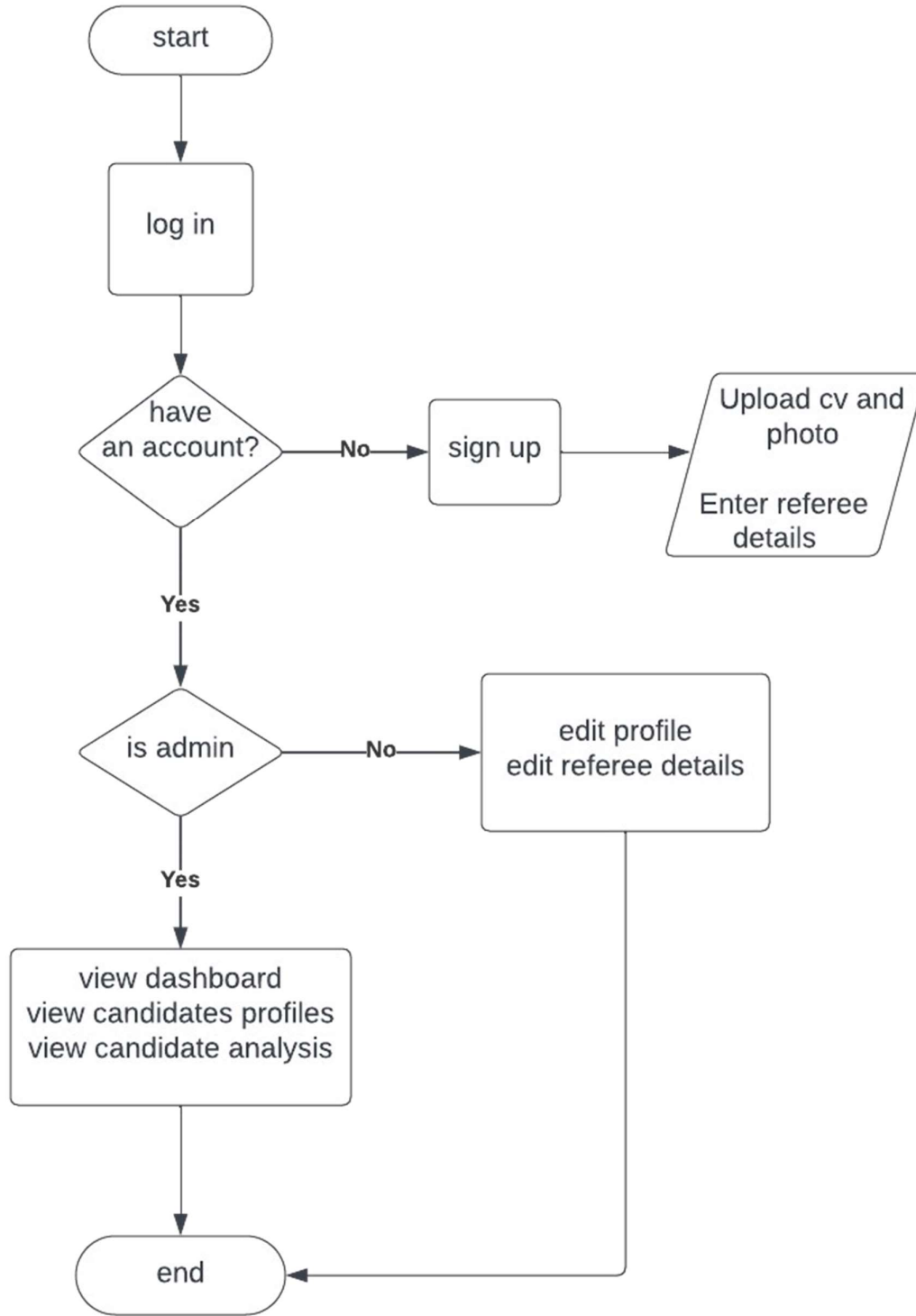


LEVEL 1 DATA FLOW DIAGRAM

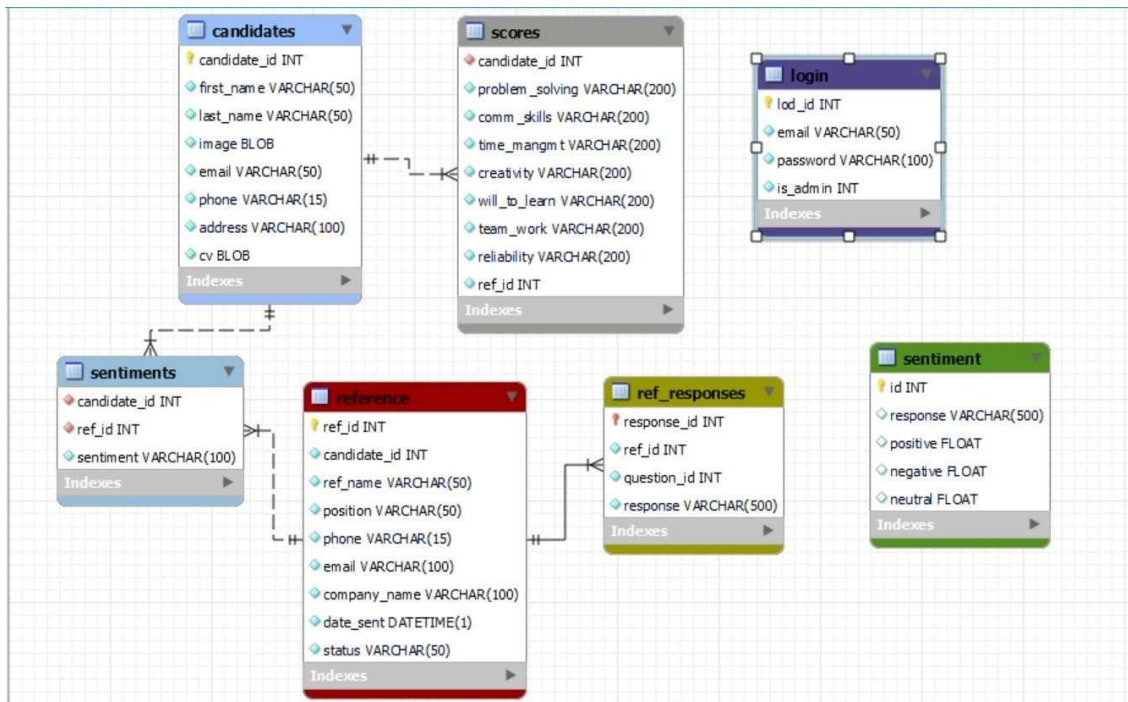




Flow -chart Diagram



ERD (entity Relation Diagram)



In the entity-relationship diagram (ERD) for the reference checking system, the following entities and relationships are represented:

- **Candidates:** Represents individuals applying for a job. (One-to-many relationship with Scores entity)
- **Scores:** Represents assessment scores assigned to candidates. (One-to-one relationship with Candidates entity)
- **Login:** Represents user login credentials for system access.
- **Sentiments:** Represents sentiments or opinions expressed by references about candidates. (One-to-one relationship with Candidates entity)
- **Reference:** Represents references provided by candidates. (One-to-many relationship with Candidates entity)
- **Reference Response:** Represents feedback received from references. (One-to-one relationship with Reference entity)



3.5.3 Sample Datasets

- User records: 12345, "Tafadzwa Ndoro", etc
- Reference text: "Good worker, on time daily"
- Sentiments: 1="positive", 2="negative"

3.5.4 Data Standards

- Sample/test data uses consistent formats for development
- Encoding uses UTF-8 for text data
- Scales like sentiments represented simply for prototyping
- Plans for production include
 1. Standards for encrypting PII at rest and in transit
 2. Supported file formats for reports/exports
 3. Data validation and schema enforcement
- Actual standards implementation and format handling will be tested extensively prior to launch

3.5.5 Data Storage

- Development database hosted using SQLite for simplified setup
- Database file stored in source code repository along with codebase
- No database backups required at this stage
- Data versions controlled through Git commits to code repository
- Data files accessed directly from development machine filesystem
- No redundancy or high availability capabilities at this point
- Data wiped and regenerated periodically as codebase evolves
- Database schema migrations managed through code commits
- Testing and seed data loaded from YAML files within codebase
- Production database migration/backup strategies TBD for future

3.5.6 Data Security

- No live user or sensitive data to secure currently
- Sample data hardcoded for testing purposes
- Credentials stored locally in configuration files • Basic role definitions implemented for access
 - Plans for production include:
 1. Encrypting credentials with strong algorithms
 2. Granular RBAC roles and permissions
 3. Anonymizing analytics data
 4. Limited retention periods for raw data



- Security controls will be implemented and tested extensively prior to launch

3.6 Security and Privacy

3.6.1 Authentication

- Basic email/password authentication implemented for testing
- Credentials are hardcoded in development configuration
- OAuth and MFA not implemented yet to integrate external services
- Authentication handled within development server temporarily
- Plans include securing credentials using JWT tokens
- Integrating services like Google/Authy before production
- Authentication workflows and security will be thoroughly tested

3.6.2 Authorization

- Basic role definitions created as application functionality is built
- Roles are hardcoded in development and seeded through migrations
- Access controls managed within application code at this stage
- Authentication is handled by the development server temporarily
- Plans to implement proper RBAC with packages pre-production
- User authorization workflows will be tested extensively
- Production plans include dedicated roles management database

3.6.3 Data Encryption

- No live website or transmitted data to encrypt currently
- Credentials stored locally in codebase only during development
- Authentication handled within development server for now
- Encryption strategies will be implemented before production
- Passwords will be securely hashed before launching publicly
- Encryption standards like AES-256 will secure user data on launch
- Deployment options continue to be evaluated

3.6.4 Data Management

- No live user data to handle at this stage of development
- Sample/testing data is hardcoded in application • Data access is by development team only on local machines



- Plans will be made for handling production data

including:

1. Training/analytics datasets with anonymized user data
 2. Limited retention periods for certain data types
 3. Export options that de-identify fields as needed
- Actual data handling workflows will be tested prior to launch

3.6.5 Infrastructure

- Application deployed on developer's local machine for initial testing
- Project codebase stored securely in private Git repository
- Database hosted locally using SQLite for simplified development
- Local web server like Apache or Nginx used for serving application
- Windows Firewall configured for protection
- Strict access control of developer workstation for security
- Non-production data and no user-facing access at this stage

3.6.6 Privacy Policy

- Basic policy drafted to govern planned user data collection
- No live site or user data yet to which policy can apply
- Policy focuses on intentions for production stages
- Policy stored locally along with codebase for now
- Acceptance workflows not yet implemented
- Policies and governance will evolve in pre-production testing
- Full policy acceptance will be integrated before public launch

Chapter 4

Implementation

This chapter describes how the reference checking web application was developed. At this stage, the app is hosted on GitHub for version control and deployed locally on developers' machines.

4.1 Development Environment

The application is built using Python, PHP, and Bootstrap. Development environments were configured locally using:

- Python 3.8
- PHP 7.4 running on Apache web server
- MySQL 8 for the reference and user databases
- Bootstrap 5 for frontend styles and components

4.2 Coding Standards

PEP-8 style guidelines are followed for Python code quality. PHP code adheres to PSR-1 and PSR-2 standards. Variables, functions and files are named for readability and maintainability.

4.3 Build and Deployment

The app is version controlled on a GitHub repository. Developers pull the latest code and run `PHP -S localhost` to launch the stack including MySQL, Apache, and Python API. Frontend assets are compiled using Bootstrap. Code is deployed to local environments by pulling from GitHub. No automated deployment processes are in place yet.

4.4 Testing Frameworks

Unit testing of Python API code is done with `pytest`. Integration tests validate API and backend PHP code. Frontend functionality is manually tested in browser environments. Test coverage will be expanded before production release.

4.5 Future Improvements

As development continues, CI/CD pipelines will be set up to automate testing and deployment. Load testing will be conducted to measure performance. Optimization techniques may be applied based on profiling results.

Chapter 5

Testing

With the reference checking system still in development phases, testing aims to validate core functionality and catch defects early. This chapter outlines the testing processes.

5.1 Scope

We test the:



- Python sentiment analysis logic
- Flask API endpoints
- PHP backend functions
- Database interactions
- Bootstrap frontend and forms
- Integration of all components

5.2 Test Cases

- Over 100 test cases cover factors like valid/invalid inputs, edge cases, responses.
- Integration tests validate API-backend-database interactions via the frontend.
- Manual test plans include successful and error-based user journeys.

5.3 Tools and Reports

- Test results from Pytest, PHPUnit and frameworks are reported to GitHub issues.
- We use TestRail for test management including plans, runs and defects logging.
- Selenium soon automates frontend functionality testing and browser compatibility.
- Continuous integration on GitHub Actions runs the full test suite on all code changes.

Chapter 6

Deployment

This chapter outlines the deployment processes for launching the reference checking system.

6.1 Current Deployment

The codebase is version controlled on GitHub. Developers pull latest changes and run the stack locally using Docker Compose.

6.2 Planned Deployment

We plan to host the application on Heroku using their Docker deployment process. Database will be on PostgreSQL.

6.3 Infrastructure Requirements

Backend services like the API will scale on Heroku Dynos. Database requirements are estimated based on user/job projections. CDN or caching may be used.



6.4 Deployment Steps

1. Code is deployed to Heroku from GitHub via CI/CD
2. Docker containers are built and launched.
3. Database schema/data are applied.
4. Traffic is routed via a load balancer

Chapter 7

Operations and Maintenance

This outlines how the reference checking system will be supported after launch.

7.1 Current Operations

As a development project, the code is stored on GitHub. Developers troubleshoot issues locally.

7.2 Future Operations

Post-launch we will:

- Monitor hosting platforms like Heroku for uptime/issues.
- Use Github issues to track bugs and feature requests.
- Respond to user queries via email support.
- Manage deployments via Git feature branches.

7.3 Backup and Disaster Recovery

Heroku automatically handles daily database backups to AWS. We document disaster recovery procedures including steps to restore services.



7.4 Performance Monitoring

New Relic tracks site performance, database queries and API response times. Alerts within New Relic notify the team about slow queries and errors.

7.5 Future Enhancements

We maintain a public roadmap for planned features, based on user needs and market trends. Mergers of significant new features may require dedicated release cycles.

7.6 Retirement Plan

When services retire, we ensure users can extract their data. Deployed code is maintained for 12 months before archiving.

Chapter 8

Conclusion

This project established the foundations for a reference checking system and lessons learned along the way.

8.1 Project Overview

We defined requirements, designed workflows and implemented core functionality to analyze references via machine learning. Internal testing proved the technical approach is feasible.

8.2 Learnings

Thorough documentation of system architecture, APIs, databases, and configurations helped our distributed team stay organized and understand dependencies. Rigorous unit and integration testing uncovered bugs early.

8.3 Next Steps

With the Minimum Viable Product now built, priorities are user research, usability studies and feature refinement based on feedback. Automating more tests, deploying infrastructure and formalizing support processes will prepare the project for a public beta.

8.4 Feedback So Far

Our documentation supported successful code reviews and status reporting to stakeholders. Feedback focused on completion of testing, deployment planning and overall project management - all areas we're addressing before a broader launch.

8.5 Conclusion

While still in nascent development stages locally, this undertaking established a solid technical foundation and crucial organizational practices for further developing the reference checking system into a supported product. The next phase of stakeholder validation and productization can now commence.

References

Johnson, A. 2023. Challenges of Manual Recruitment Systems. In: Recruiting in the Digital Age (eds J. Smith and M. Brown), New York: Wiley, pp. 3-10.