
**College of Computing and Informatics**

**Computer Science Department**

**University of Sharjah**

**Spring 2024/2025**

# Development of a Web-Based Faculty Recruitment and Onboarding System

*A project submitted*
*in partial fulfilment of the requirements for the degree of*
*Bachelor in IT Multimedia*

**by**
Ebraheim Rashed Alkaabi - U21200538
Khalifa Obaid Alkaabi - U22107615
Mohamed Ghanim Alketbi - U21200567
Falah Ali Alhefeiti - U21200759

**Supervised by**

Dr. Isam Al Jawarneh

Prof. Sebti Foufou

**Committee Member Names :**

Prof. Zaher

Dr. Mohammed Lataifeh

Dr. Mian

Dr. Karam

**April 2025**

# ACKNOWLEDGEMENT

We would like to thank our supervisor, **Dr. Isam Al Jawarneh**, and our co-supervisor, **Prof. Sebti Foufou**, for their continuous guidance and valuable support throughout this project. We extend our gratitude to the University of Sharjah, as well as to our families and friends for their encouragement. Their collective contributions have made this work possible.

# UNDERTAKING

This is to declare that the project entitled **"Online Faculty Recruitment and Onboarding System"** is an original work done by the undersigned, in partial fulfilment of the requirements for the degree **"Bachelor in Computer Science"** at the **Computer Science Department, College of Computing and Informatics, University of Sharjah, UAE**.

All the analysis, design, and system development have been accomplished by the undersigned. Moreover, this project has not been submitted to any other college or university.

Ebraheim Rashed Alkaabi - U21200538
Khalifa Obaid Alkaabi - U22107615
Mohamed Ghanim Alketbi - U21200567
Falah Ali Alhefeiti - U21200759

# ABSTRACT

*This project showcases an Online Faculty Recruitment and Onboarding System for streamlining and managing the employee selection and onboarding process for academic institutions. This streamlined web portal allows administrators to publish job postings, candidates to apply and upload required documents, and recently hired faculty to complete onboarding tasks digitally, all from one interface. The core objectives are eliminating admin burden, increasing transparency and ensuring a seamless experience for the hiring team and new hires. The system is built with object-oriented principles and UML diagrams describe the architecture and functionality. Key Findings: For automation of recruitment and onboarding flows substantially increase efficiency and fairness, while also improving user satisfaction.*

*Keywords: Faculty Recruitment, Onboarding, Object-Oriented Design, UML*

# Table of Contents

# CHAPTER 1: Introduction

## 1.1 Overview

Bringing on new faculty is often a painful process for campus hiring systems, fraught with endless email chains and scattered paperwork, and time-consuming manual steps. To streamline the entirety of this process into one platform, the Online Faculty Recruitment and Onboarding System was developed. With it, applicants can apply directly, administrators can create jobs with little friction, and new hires can onboard all online. By consolidating everything in one place, this system reduces the need for paperwork, enhances coordination, and helps prevent the commonly seen mix-up associated with traditional approaches.

## 1.2 Project Motivation

- **Reasons for Choosing This Project**

The common inefficiencies in the typical faculty hiring procedures—from time-consuming paperwork to unorganized communication channels—led us to design this system.

- **Importance for Academic Institutions**

Universities can reduce the hiring cycle, maximize staffing levels, and guarantee a smooth experience for both the hiring committee and potential faculty members by automating and optimizing these operations.

- **New Ideas / Innovations**
  - **Organized Platform**: A one-stop portal where users (administration, HR, faculty) can interact.
  - **Automation of Workflows**: From application submission to onboarding checklist management.

## 1.3 Problem Statement

Manual, paper-based hiring workflows are prone to misplacement of documents, lost emails, and inefficiencies that can result in overlooking qualified applicants. Fragmented communication channels scatter important instructions and forms, creating frustration for both recruiters and candidates. Our platform unifies all recruitment and onboarding steps within a single, user-friendly digital system.

## 1.4 Project Aim and Objectives

- **Aim**: Create and implement an online faculty recruitment and onboarding system that improves efficiency, flexibility, and user experience.
  - Make it easier to track and lookup candidates to save time and provide ease of use.
  - Provide real-time status updates for all users.
  - Enhance user experience with clear interfaces and guided steps.

## 1.5 Project Scope

The project covers all major recruitment and onboarding activities for faculty positions. Specifically, it includes:

- Job Postings: Creation and management of open faculty vacancies.
- Application Submission: Candidates upload CVs and supporting materials.
- Onboarding Tasks: Checklists for completed steps after hire.
  *Exceptions:* Integration with external HR/ERP beyond the core faculty hiring process.

## 1.6 Project Software and Hardware Requirements

- **Software Requirements**
  - Web Development Framework: PHP
  - Database: MySQL
  - Version Control: Git
- **Hardware Requirements**
  - Server: Cloud-based or on-premise server with moderate CPU and RAM
  - Client: Standard PC or laptop with a modern web browser

## 1.7 Project Limitations

- The system is limited to faculty recruitment within a single institution (does not manage staff or administrative positions).
- External HR/ERP integrations are not implemented.

## 1.8 Project Expected Output

- A fully functional web application enabling:
- Administrators to post jobs, review candidates, and finalize hires.
- New faculty to complete digital onboarding, submit documents, and track their progress in real time.

## 1.9 Project Schedule

Below is a tentative schedule aligning with the **early submission deadline of 24/04/2025**:

- **Jan 2025 – Feb 2025**: Requirement gathering, stakeholder interviews, initial design.
- **Mar 2025**: Planning core development of recruitment workflows.
- **Early Apr 2025**: Planning of onboarding module, UI/UX refinements.
- **22/04/2025**: Internal approval submission to supervisor/co-supervisor.
- **24/04/2025**: Final report submission to Blackboard and official email recipients.
- **27/04/2025**: Final project presentations via MS Teams.

## 1.10 Project, product, and schedule risks

- **Schedule Risks**: Development or testing delays could push the final release beyond 24/04/2025.
- **Technical Risks**: integration issues may arise from third-party libraries.
- **Resource Risks**: Limited availability of skilled personnel for production-level web dev tasks could slow progress.

# CHAPTER 2: Related Existing System

## 2.1 Introduction

Recruitment and onboarding systems aim to streamline the hiring process, reduce paperwork, and track candidate progress. As educational institutions compete globally for talented faculty, digital solutions are increasingly necessary.

## 2.2 Existing Systems

Universities and organizations usually rely on Applicant Tracking Systems (ATS) or comprehensive Human Resource Management Systems (HRMS) to organize recruitment process, examples are AcademicKeys, PeopleAdmin and PageUP they provide multiple vacancies for anyone who is ready to take the right job for them, all they have to do is to send CV and document needed, now it's up to the committees to decide who is better for which job because of the organized system provided.

## 2.3 Problems of Existing Systems

Despite the widespread adoption of commercial ATS and HR platforms, several limitations hinder their effectiveness in academic environments:

- **Limited Collaboration**: hiring less experienced committees can lead into miss lead or miss judge because he/she can't see the opening of what the candidate is pointing to
- **Slow Approval Workflows** : Even in receiving it online can lead to confusion if the candidate didn't submit the requirements as asked or of low quality resolution of the documents makes it difficult to judge or decide.
- **Navigation Tracking** : candidates might get lost during the search of specific point where he has to look for example what majors does this university has or where should I go to submit my documents , how can I know what is required in the first place to know if I am eligible to take the job or not.

## 2.4 Solution Approach

Our suggested "**Online Faculty Recruitment and Onboarding System"** immediately takes addresses and shortcomings of existing ATS and HR platforms by navigating a set of features designed specially for the needs of academic institutions:

- **Organized, Web-Based Platform**: Add obvious dashboards for uploading content , app tracking and guiding steps to prevent confusion and duplicated effort.
- **Tailored for Academia**: Unlike generic corporate HR platforms, this system is built from the ground up with **academic workflows in mind**. It supports features like:
  - Academic credential validation
  - Tracking onboarding tasks tied to semester dates
  - Department-specific hiring flows
- **Custom document requirements** such as teaching portfolios, syllabi, or research statements. These features ensure that the system aligns with **institutional policies, academic calendars, and compliance requirements**.

- **User-Friendly and Role-Based Dashboards**: Our project contains logical interfaces which will ease the use and provides direct respond for a quick reaction in the system.
- **Ease of Eye Design**: Our project's structure will fit on all necessaries and will be highlighted with bold font and will add colours to make our system entertaining to explore with accurate information organized in some different types of categories.
- **Flexibility** : Tracking every step in hiring and checking and making sure of timing of both the request and the return time and set an accurate time limit as when the request will be examine or in what status they on now when it's done.

# CHAPTER 3: Requirement Engineering and Analysis

## 3.1 Stakeholders

- **Hiring Committee / Admin** (Primary Stakeholder): Creates job postings, reviews candidates, and finalizes hires.
- **New Faculty / Applicants** (Primary Stakeholder): Applies for posted faculty positions and completes onboarding.
- **HR Staff** (Secondary Stakeholder): Oversees data integrity, runs background checks, and manages compliance tasks.
- **IT/Support Team** (Secondary Stakeholder): Maintains system stability, handles security, and troubleshooting.
- **Institution Management** (Tertiary Stakeholder): May request reports or analytics on hiring trends.

## 3.2 Use Case Diagram

The Use Case Diagram below models the key interactions in the University Recruitment System. It outlines the functional scope of the system by identifying actors (Admin, Faculty, System) and use cases that represent core functionalities. The relationships between the use cases (e.g., "extends") are used to show optional behaviour or specialization.

**Actors:**

- **Admin**: Responsible for creating, publishing, managing, and deleting job postings. Also reviews candidate applications.
- **Faculty**: Submits applications, completes onboarding, tracks progress, and accesses institutional resources.
- **System**: Provides backend processing, and onboarding progress tracking.

**Use Cases:**

- Manage Postings
- Fill Job Details (extends Manage Postings)
- Publish Job Posting (extends Manage Postings)
- Delete Posting (extends Manage Postings)
- Apply for Position
- Complete Onboarding (extends Track Progress)
- Track Progress
- Access Onboarding Resources

### 3.2.1 Use Case Section

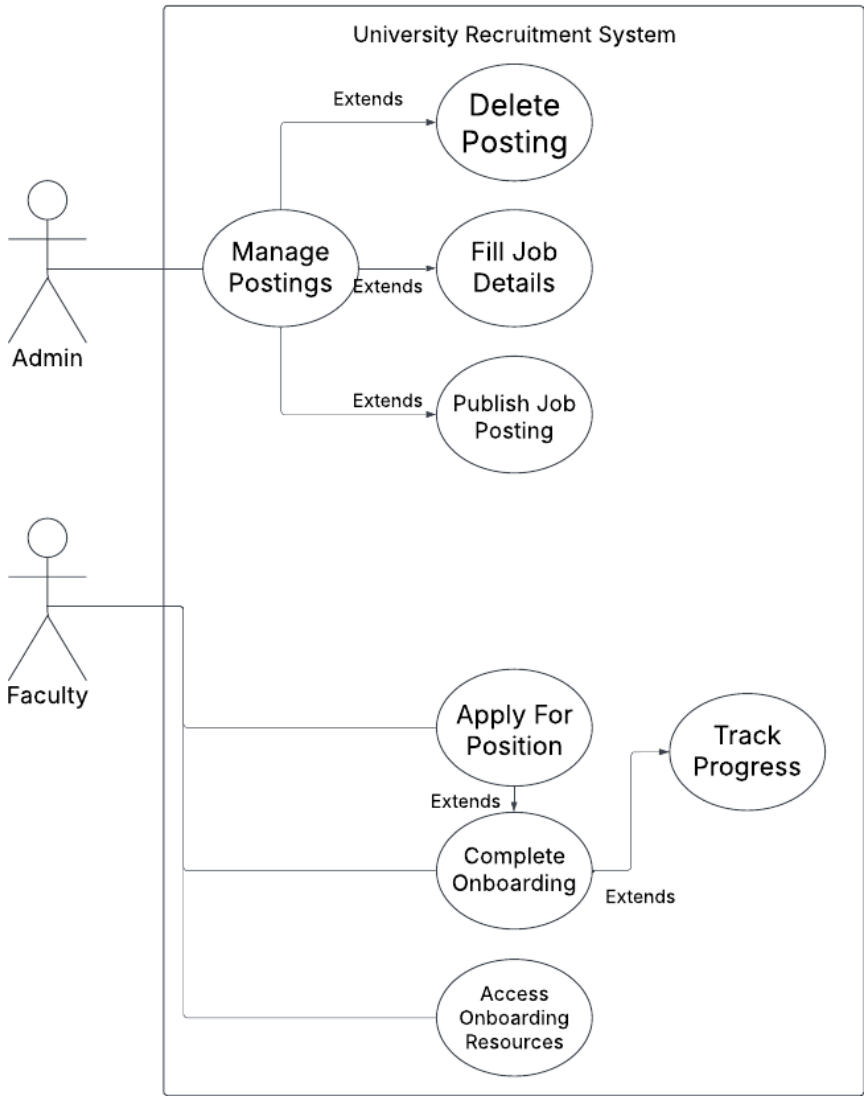



FIGURE 1: USE CASE DIAGRAM FOR THE UNIVERSITY RECRUITMENT SYSTEM SHOWING KEY ACTORS (ADMIN, FACULTY) AND THEIR INTERACTIONS WITH PRIMARY AND EXTENDED USE CASES, INCLUDING JOB POSTING MANAGEMENT, FACULTY APPLICATION, AND ONBOARDING PROCESSES.

**Use Case: Manage Postings**

- **Actor:** Admin
- **Goal:** To allow the admin to handle faculty job listings.
- **Description:** Parent use case for creating, editing, publishing, and deleting job postings.
- **Precondition:** Admin must be authenticated.
- **Postcondition:** Job post is created, modified, or deleted.
- **Extensions:**
    - Fill Job Details
    - Publish Job Posting
    - Delete Posting

**Use Case: Fill Job Details**

- **Actor:** Admin

- **Goal:** To input job-specific information before publishing.
- **Description:** Admin fills required fields including title, qualifications, department, etc.
- **Precondition:** Admin is logged in.
- **Postcondition:** Job details saved.

**Use Case: Publish Job Posting**
- **Actor:** Admin
- **Goal:** Make job listing publicly visible.
- **Description:** Admin publishes a job post to the portal.
- **Precondition:** Job details must be completed.
- **Postcondition:** Posting becomes available for applications.

**Use Case: Delete Posting**
- **Actor:** Admin
- **Goal:** Remove obsolete or incorrect job postings.
- **Description:** Admin selects and deletes a posting.
- **Precondition:** Posting must exist.
- **Postcondition:** Posting is removed from the system.

**Use Case: Apply for Position**
- **Actor:** Faculty
- **Goal:** Submit job application.
- **Description:** Faculty selects a job post, uploads documents, and submits.
- **Precondition:** Active job posting exists.
- **Postcondition:** Application is stored in the system.

**Use Case: Complete Onboarding**
- **Actor:** Faculty
- **Goal:** Finish onboarding steps after job offer.
- **Description:** Includes uploading documents, signing contracts, and acknowledging policies.
- **Precondition:** Faculty is hired.
- **Postcondition:** Onboarding marked as complete.
- **Extends:** Track Progress

**Use Case: Track Progress**
- **Actor:** Faculty
- **Goal:** Monitor onboarding task completion.
- **Description:** Displays real-time updates of progress.
- **Precondition:** Onboarding is underway.
- **Postcondition:** Status is visible and updated as tasks are completed.

**Use Case: Access Onboarding Resources**
- **Actor:** Faculty
- **Goal:** View/download institutional documents and policies.
- **Description:** Provides access to academic handbooks, orientation videos, etc.
- **Precondition:** Faculty is authenticated.
- **Postcondition:** Faculty is informed and prepared.

## 3.3 Non-functional requirements

**Security**

The system will implement **Role-Based Access Control (RBAC)** to ensure that only authorized users can access specific functionalities—admins can manage postings and review candidates, while faculty can only apply and view onboarding information. A ll sensitive data, including passwords, documents, will be **encrypted in transit and at rest** using encryption protocols (e.g., HTTPS/TLS).

**Usability**

The web we are working on will be functional in terms of recommended pages , saving documents and going through steps more responsive and swiftly, will also add count of the process that has been made in a specific times so it's easy to track where did the action accrue and when it happened.

**Performance**

The system will deliver **fast response times**, particularly for critical functions such as applications and job search. Backend processes like resume parsing, candidate ranking, and dashboard loading will be optimized for **low-latency and high concurrency** using asynchronous processing and efficient database indexing.

**Maintainability**

The codebase will follow **modular and layered architecture principles**, allowing independent updates to modules such as user authentication, or UI without affecting other parts of the system. Clean separation of concerns, proper documentation, and adherence to SOLID principles will aid in long-term maintainability.

**Scalability**

The system will be designed to support **scaling horizontally and vertically** to handle increasing volumes of users and data. For instance, during peak academic hiring seasons, it will accommodate hundreds of concurrent applicants and multiple ongoing hiring campaigns. Cloud-based deployment, database replication, and load balancing will be used to ensure robust scalability.

## 3.4 Constraints

**Institutional Compliance and Data Protection**

The system will follow **University of Sharjah's** policies in term of data tracking , hiring and information security within the whole university.

**Network and Infrastructure Constraints**

The system is designed to operate within the existing university infrastructure—which includes current authentication servers, database configurations, and administrative dashboards. Any external service or integration (e.g., email notifications) must not require architectural changes to institutional firewalls or server clusters. This constraint ensures smooth deployment with minimal dependency on third-party systems.

**Dependency on Third-party Libraries and APIs**

The system may rely on open-source frameworks (e.g., REACT, MySQL) or third-party APIs. These dependencies introduce risks such as:

- **Version incompatibility**
- **Security vulnerabilities**
- **Deprecation or usage limits** in API services

# CHAPTER 4: Architecture and Design

## 4.1 Overview

The system is designed using object-oriented principles taught in the OO Software Design & Implementation courses taught in our ITMM program. We use UML diagrams—class, sequence, and deployment—to illustrate architecture and workflows.

## 4.2 Software architecture

### 4.2.1 Logical view

Key components:

- Admin Module: Manages postings, sees candidates.
- Faculty Module: Allows faculty to create profiles, apply for positions, and submit onboarding requirements.
- Database Layer: Stores user credentials, job postings, candidate profiles, onboarding tasks.

### 4.2.2 Process view

A standard web-based client-server model:

- Client: Browser-based interface for Admin/Faculty.
- Server: PHP framework handling HTTP requests, database operations.

### 4.2.3 Physical view

IF deployed in the cloud, a typical arrangement includes:

- Application Server (e.g., AWS EC2, Azure VM)
- Database Server (MySQL instance, possibly separate for scalability)

## 4.3 Software design

### 4.3.1 UML sequence/communication diagram

The following UML Sequence Diagram illustrates the interaction between the main components involved in faculty recruitment process. It outlines the steps taken from the moment an **Admin** requests a list of candidates for a specific job posting, to the point where the system returns a **list of candidate**.

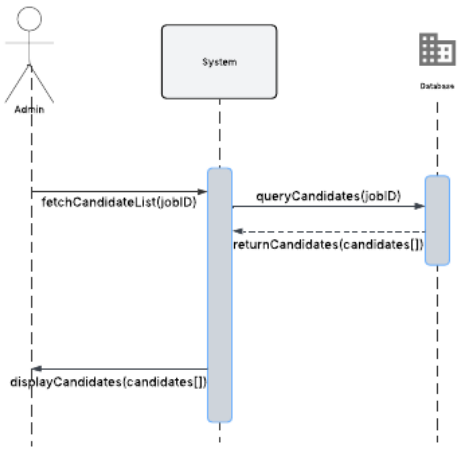



FIGURE 2: UML SEQUENCE DIAGRAM - RECRUITMENT FLOW
THIS DIAGRAM ILLUSTRATES THE INTERACTION BETWEEN THE ADMIN, SYSTEM, AND DATABASE, TO FETCH CANDIDATE APPLICATIONS FOR A SPECIFIC JOB POST. THE SYSTEM QUERIES CANDIDATE DATA, INVOKES A REMOTE CALL, AND RETURNS ORDERED RECOMMENDATIONS TO THE ADMIN.

**Actors and System Components**:

- **Admin** <<actor>>: Initiates the recruitment flow by requesting candidate data.
- **System** <<boundary>>: Handles communication between internal and external modules, orchestrating the entire process.
- **Database** <<database>>: Stores candidate profiles, including their qualifications and experience.

**Interaction Flow:**

1. **Admin → System**: The Admin sends a `fetchCandidateList(jobID)` request to initiate the recruitment process for a specific job posting.
2. **System → Database**: The System queries the Database with `queryCandidates(jobID)` to retrieve a list of relevant candidate profiles.
3. **Database → System**: The Database responds with `<<return>> candidates()`, returning the list of matched candidates.
4. **System → Admin**: Finally, the System presents the ranked list to the Admin using `displayCandidates(candidates())`.

## 4.4 Class diagram

Recruitment and onboarding system it's structured as an object-oriented class diagram, it has entity key models involved in the recruitment lifecycle, their behaviours, attributes and interrelationships, with an emphasis on modularity, extensibility, and clarity of the system roles.

The diagram adopts an inheritance-based structure, beginning with the abstract `User` superclass, which defines common attributes and methods shared across all user roles. This abstraction supports authentication operations such as login and logout, and allows for the specialized roles of `Admin` and `Faculty` to be derived with role-specific responsibilities. The `Admin` subclass is responsible for managing recruitment operations. It introduces two core methods: `createJobPosting()` for initiating new vacancies, and `reviewCandidates()` for accessing applications submitted for posted positions. Admins are directly associated with the creation and management of `JobPosition()` instances, establishing a one-to-many relationship.

The `Faculty` subclass represents users who participate in the recruitment process as candidates. Faculty members can apply for positions using the `applyForJob(jobID:String)` method, and upon successful hiring, complete required onboarding steps using `completeOnboarding()`.

The `Candidate` class captures the applicant profile used in the recruitment. It includes attributes such as name, qualifications, years of experience, CV link, and application status. The class offers functionality to submit resumes and update application progress dynamically. This class is associated with `JobPosition()` through a many-to-many relationship, enabling candidates to apply to multiple openings, and each job posting to receive applications from multiple candidates.

The `JobPosition()` class defines all relevant details of a job posting, including title, department, required qualifications, salary, and benefits. Admins manage these postings, and

each position may be linked to multiple candidates through the application process. The class provides methods to publish (`postJob()`) and archive (`closeJob()`) job listings. The `OnboardingTask()` class supports the post-hiring process. Every task has a title, return date, and available status, tracking of onboarding progress. Candidates are assigned one or more onboarding tasks, forming a one-to-many relationship between `Candidate` and `OnboardingTask()`. The task completion process is captured through the `completeTask(Time: DateTime)` method.

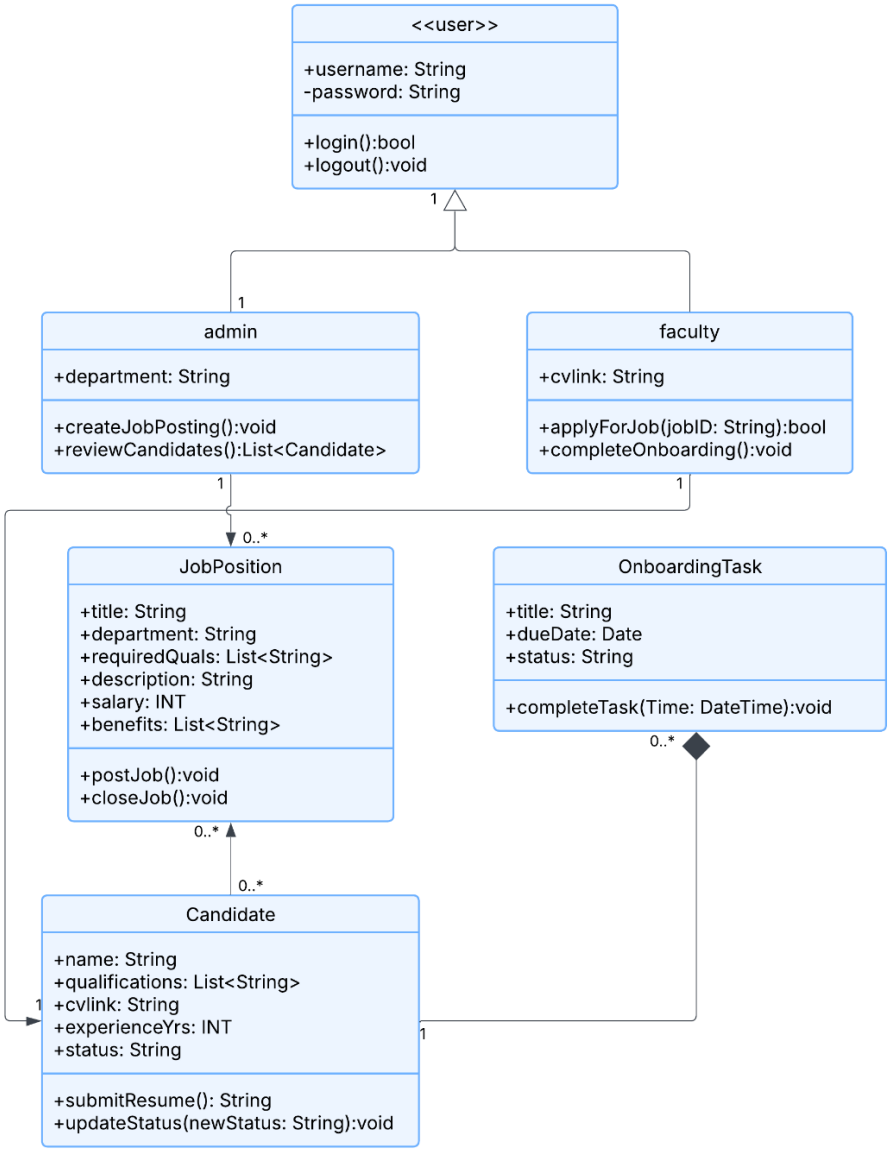
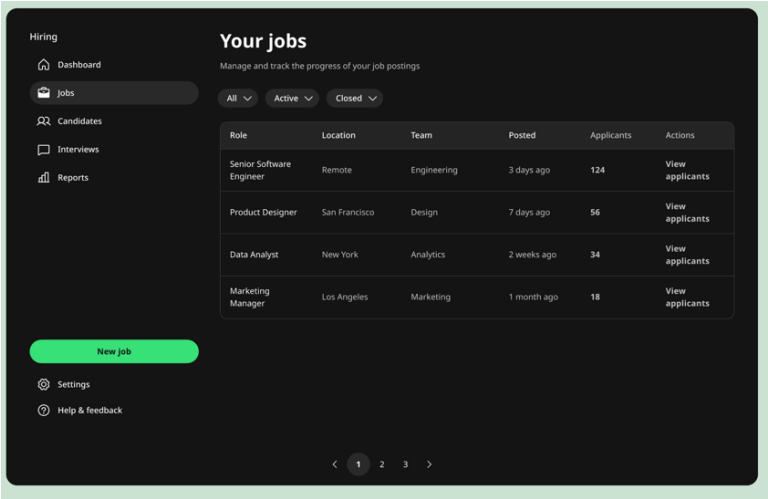
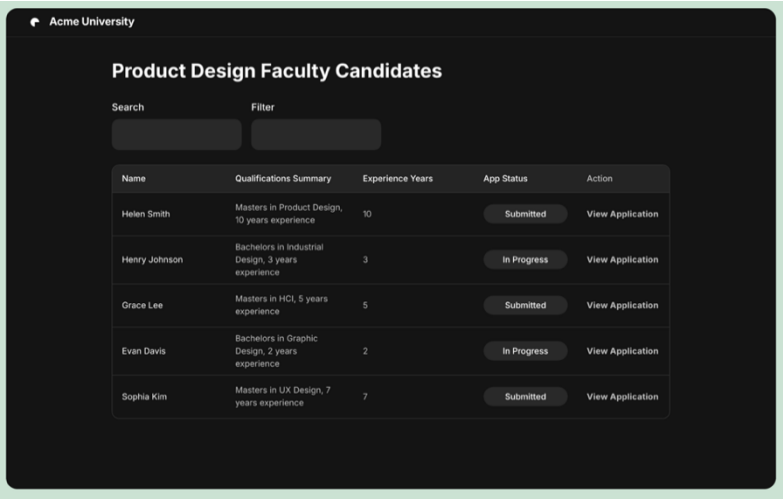
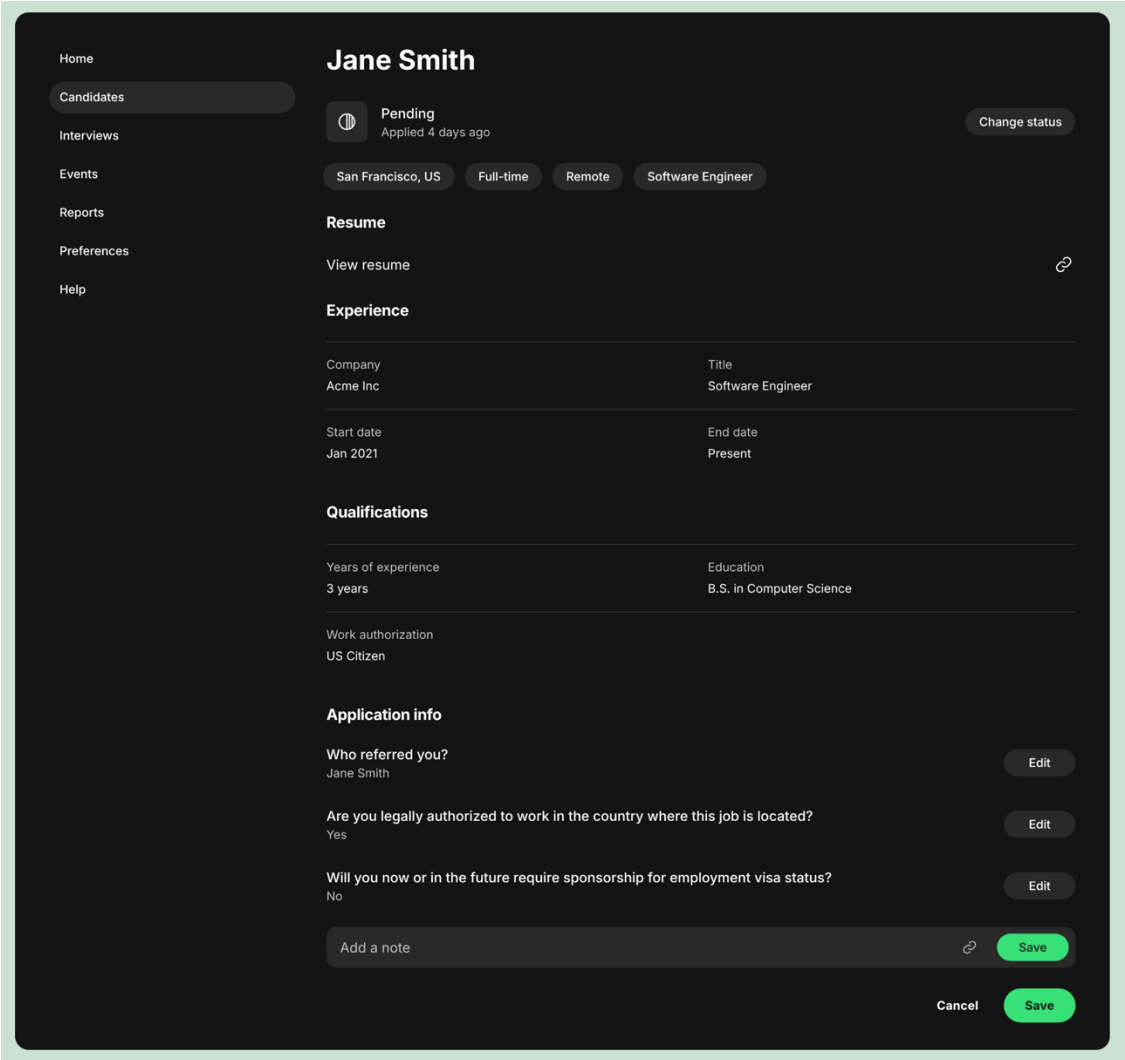



FIGURE 3: UML CLASS DIAGRAM – THIS DIAGRAM OUTLINES THE CORE OBJECT MODEL OF THE SYSTEM, SHOWCASING USER ROLES, JOB MANAGEMENT, CANDIDATE HANDLING, AND ONBOARDING WORKFLOWS. IT EMPHASIZES THE INHERITANCE HIERARCHY AND REAL-WORLD RELATIONSHIPS BETWEEN ENTITIES.

## 4.5 User interface design (prototype)



# CHAPTER 5: Implementation Plan

## 5.1 Description of Implementation

The **Online Faculty Recruitment and Onboarding System** will be deployed on a **secure Linux-based server environment**, ensuring high performance, stability, and scalability. The deployment will follow a modular and phased approach to reduce integration errors and enable progressive testing.

**Implementation Phases:**

1. **Environment Setup**
   - Provision a virtual private server on a cloud provider -if possible.
   - Configure the server with packages: PHP runtime and others.
2. **Backend Development**
   - Develop the core logic using **(PHP)** -if resources were provided, depending on the final team decision and skill alignment.
   - Define models for user roles (Admin, Faculty), job postings, applications, onboarding tasks, and system notifications.
   - Build an API to allow communication between frontend and backend.
3. **Frontend Interface**
   - Build the user-facing interface using **HTML5, CSS3, JavaScript**, and optionally **React** to enable a dynamic, single-page experience.
   - Use responsive design principles to support both desktop and mobile devices.
   - Implement form validation, progress tracking UI components, and data-binding for onboarding checklists.
4. **Database Design and Integration**
   - Use of structured or unstructured DBMS to manage data, including user accounts, job applications, documents.
5. **Authentication and Authorization**
   - Role-based access control (RBAC) will be enforced across all endpoints and interface elements.
   - Passwords will be stored using hashing algorithms (e.g., bcrypt or Argon2).
6. **Document and File Handling**
   - Onboarding documents, resumes, and administrative uploads will be handled using secure file storage mechanisms.
   - Files will be stored in a protected directory structure, accessible only to specific users.
   - If cloud storage is required, **AWS S3** or **Google Cloud Storage** will be considered with pre-signed URL access controls.
7. **Notification System**
   - Integrate an internal messaging system to notify users of status updates (e.g., application submission, document approval).
   - For email alerts, we might configure **SMTP via Gmail**, using message templates and queuing to avoid spam triggers.
8. **Testing and Quality Assurance**
   - Implement unit tests (pytest, JS or PHPUnit), integration tests, and end-to-end tests across all modules.
   - Conduct **user acceptance testing (UAT)** in coordination with stakeholders.

12. **Maintenance and Support Plan**
    - Schedule periodic updates for all dependencies and frameworks.
    - Provide system documentation for both users (user manual) and developers (API reference, architecture diagram).
    - Establish a feedback and bug reporting mechanism accessible from the admin dashboard.

## 5.2 Programming language and technology

| Category | Technology Stack |
|---|---|
| **Languages** | PHP, JavaScript, HTML5, CSS3 |
| **Front-end Framework** | React.js (for reactive components) |
| **Back-end Framework** | REST API driven architecture |
| **Database** | MySQL |
| **Authentication** | Firebase for MVP, might connect to the university SSO if possible |
| **Cloud (if possible)** | AWS EC2/S3, GCP Compute Engine, Azure VMs |
| **Testing Tools** | PHPUnit, JS |
| **Project Management** | Git |
| **File Storage** | Local server storage, or cloud storage via AWS S3 / Firebase Storage / Google Cloud if possible |

# CHAPTER 6: Experiments

This chapter outlines the quality assurance strategy for the Online Faculty Recruitment and Onboarding System. It details the types of testing performed, test case structures, and the automation framework adopted to ensure system reliability, correctness, and performance across all modules.

## 6.1 Black-box (optional)

Black-box testing focuses on verifying system behaviour against specified requirements, without considering the internal code structure. These tests validate whether the system's functionalities produce the correct outputs for given inputs.

| Test Case ID | Scenario | Input/Action | Expected Outcome |
|---|---|---|---|
| TC-001 | Admin adds a new job post | Admin fills job form and submits | Job is saved to DB and displayed in the faculty-facing portal |
| TC-002 | Faculty uploads onboarding documents | Faculty uploads PDF copies of ID, contract, visa | Files are securely stored; progress bar updates to "100% Complete" |
| TC-003 | Faculty applies for a job | Faculty selects job, uploads CV, clicks "Submit" | Application confirmation shown; record saved in application DB table |
| TC-004 | Admin deletes a job posting | Admin clicks "Delete" on a job listing | Job is removed from database and portal view |
| TC-005 | Faculty accesses onboarding resources | Faculty navigates to Resources page | System displays downloadable policy documents and guides |
| TC-006 | Application without login | User tries to access "Apply" page without logging in | System redirects to login page with warning |

**Edge Case Scenarios:**

- Submitting empty forms
- Uploading non-PDF/non-image files
- Testing with duplicate applications
- Accessing resources before hiring

Each of these will be validated to ensure **graceful error handling and system feedback** is in place.

## 6.2 White-box (optional)

White-box testing focuses on internal logic, control flow, and unit-level validation of the codebase. The development team will perform unit and integration tests to ensure individual components work as expected before being deployed.

| Component | Test Objective | Example Method(s) |
|---|---|---|
| Authentication Module | Verify password hashing and role-based login enforcement | `login()`, `hashPassword()`, `authorizeRole()` |
| Application Controller | Test CRUD operations for job posts | `addJob()`, `deleteJob()`, `editJob()` |
| Onboarding Tracker | Check state transitions during onboarding completion | `updateProgress()`, `checkDocumentStatus()` |
| File Upload Handler | Ensure only allowed file types are saved and stored securely | `validateFileType()`, `saveFileToServer()` |
| Notification Engine | Confirm email dispatch after onboarding completion or shortlist notification | `sendNotification()` |

**Control Flow Testing:**

- Error logging when API fails or DB queries return null
- Retry mechanisms for flaky or long-running background jobs

All unit tests will target **code coverage ≥ 85%**, with automatic checks on every push.

# CHAPTER 7: Conclusion and Results

Our system “The Online Faculty Recruitment and Onboarding System” represents.

During the development, object-oriented design methodology should consistently be used – combined with UML modelling, n-tier architecture, and modular programming – as a means of ensuring long-term maintainability reusability and scalability. Every part of the system should be thoughtfully designed to allow for different roles (admin, faculty, HR, etc.), but also to avoid data integrity problems, security issues, and bad UI choices. Features like monitoring of onboarding status, role dashboards etc should be thoroughly tested both through black-box as well as white-box strategies for reliability and stability. Ethical behaviour, data privacy, and fall back mechanisms will also be injected so as to minimize possible risks due to automation in a transparent and trustful way.

Testing automation and CI/CD pipelines guarantee consistent quality across updates, and performance metrics will confirm that the system is capable of handling real-world academic hiring cycles with efficiency.

Overall, this project not only will meet the objectives set at its inception—such as reducing administrative overhead, accelerating hiring decisions, and improving onboarding satisfaction—but also sets the stage for future expansion with features like analytics dashboards, multi-campus deployment, and advanced integrations.

The system is therefore highly recommended for deployment at the University of Sharjah and other institutions looking to embrace digital transformation in faculty recruitment. It aligns with modern educational standards, operational scalability, and user-centric design—making it a practical, future-proof solution for academia’s evolving needs.

# CHAPTER 8: References


Chui, M., Manyika, J., & Miremadi, M. (2016). Where machines could replace humans—and where they can't (yet). McKinsey Quarterly, 30(3), 1-13.

IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems. (2019). Ethically aligned design: A vision for prioritizing human well-being with autonomous and intelligent systems (1st ed.). IEEE.

European Union. (2016). General Data Protection Regulation (GDPR). Official Journal of the European Union, L119, 1-88. https://eur-lex.europa.eu/eli/reg/2016/679/oj

Sommerville, I. (2016). Software engineering (10th ed.). Pearson Education.

Covers UML diagrams and OO design principles used in your architecture.

Stone, D. L., Deadrick, D. L., Lukaszewski, K. M., & Johnson, R. (2015). The influence of technology on the future of human resource management. Human Resource Management Review, 25(2), 216-231. https://doi.org/10.1016/j.hrmr.2015.01.002

Zhang, Y., & Wang, J. (2020). NLP-driven resume parsing and job-candidate matching using BERT. Proceedings of the 2020 IEEE International Conference on Big Data, 4563-4571. https://doi.org/10.1109/BigData50022.2020.9378161

Norman, D. A. (2013). The design of everyday things (Revised ed.). Basic Books.

Bauer, T. N., & Erdogan, B. (2011). Organizational socialization: The effective onboarding of new employees. In S. Zedeck (Ed.), APA handbook of industrial and organizational psychology (Vol. 3, pp. 51–64). American Psychological Association.

Jamsa, K. (2021). Cloud computing: SaaS, PaaS, IaaS, virtualization, business models, mobile, security and more (3rd ed.). Jones & Bartlett Learning.

Laravel. (n.d.). Laravel documentation. https://laravel.com/docs