**CS673 Software Engineering** 

**Team 2 - CareerForge**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Gopi Rayini | Configuration Lead | *GR* | 9/19/25 |
| Stacey Burns | QA Lead | *SB* | 09/19/2025 |
| Yongxiang Chen | Security Lead | *YC* | 09/19/2025 |
| Qi Chen | Requirement Lead | *QC* | 09/20/2025 |
| Pedro Ramirez | Design and implementation | *PR* | 09/21/2025 |
| James Rose | Database Design | *JR* | 9/23/2025 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1.0** | **Team** | **09/19/2025** | **Initial Draft** |
|  |  |  |  |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_3ipvmjgn6clp)

[Class Diagram](#_ky60nv8suxxm)

[UI Design (if applicable)](#_7ucksmkf6rzx)

[Database Design (if applicable)](#_tcmuor4nl1kz)

[Security Design](#_x18fj36s1121)

[Business Logic and/or Key Algorithms](#_mtfbusfb0eq3)

[Design Patterns](#_9zvwkmc4luo5)

[Any Additional Topics you would like to include.](#_15tmymhipvdv)

[References](#_50ojo9i46ytq)

[Glossary](#_8n34lvocupub)

# Introduction

This Software Design Document (SDD) describes the architecture, components, and design decisions of **CareerForge**, a web-based job tracker application. The document provides a blueprint for system implementation, detailing the frontend, backend, database schema, user interface, business logic, security, and design patterns. It ensures that developers, stakeholders, and maintainers share a clear and consistent understanding of how the system is structured and how it should operate.

### Design Goals

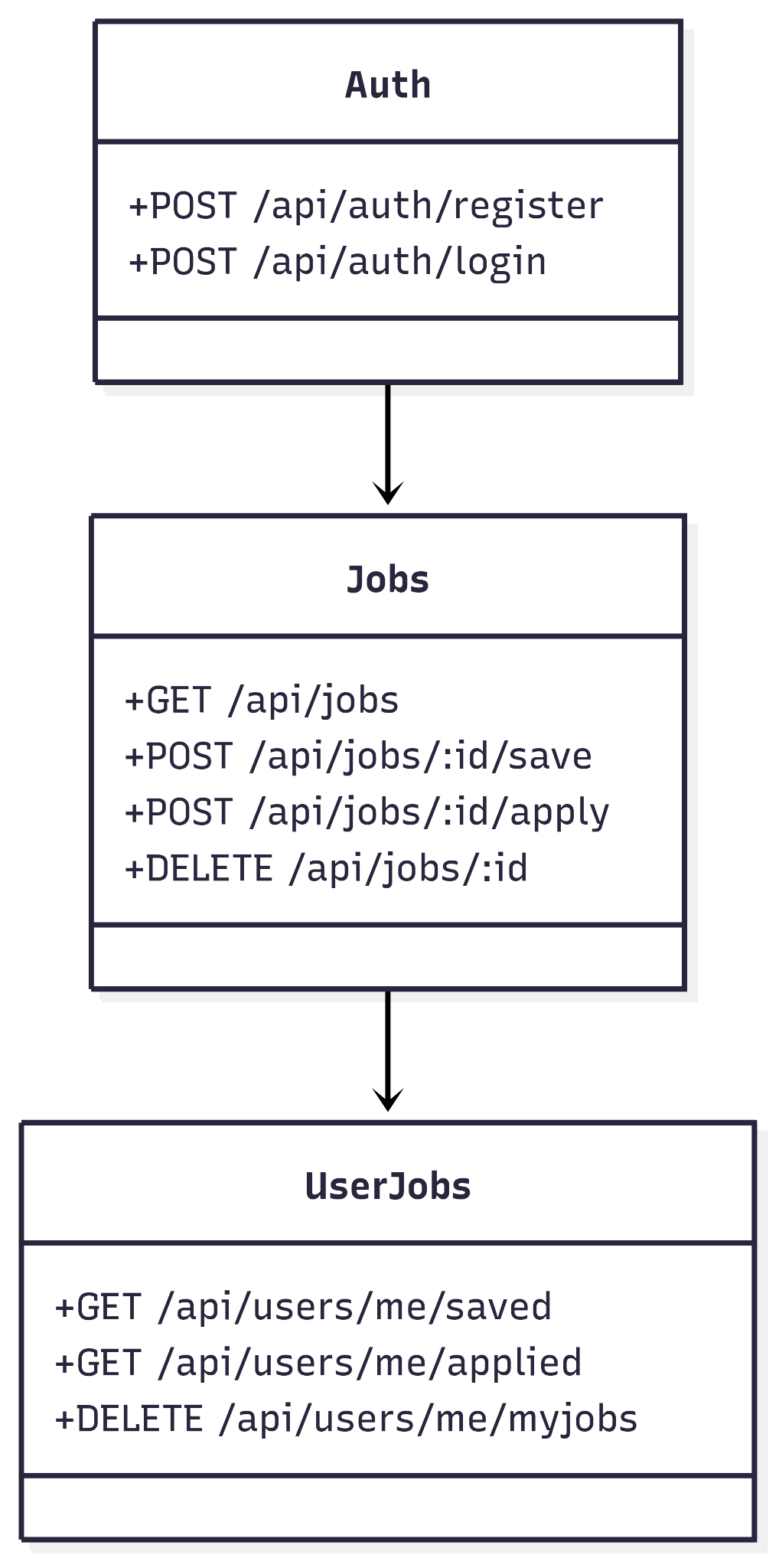
The design of CareerForge is guided by the following goals:

* **Functionality** – Enable users to register, log in, browse job postings, and manage applications efficiently.
* **Modularity** – Keep the system organized into clear layers (frontend, backend, database) to simplify updates and maintenance.
* **Scalability & Performance** – Support growth in users and job data while remaining responsive.
* **Security** – Protect user data with authentication, authorization, and secure communication.
* **Usability** – Provide a clean, intuitive, and responsive interface for job tracking on both desktop and mobile.
* **Maintainability** – Apply design patterns (MVC, Repository, Facade, Factory Method) to keep code organized, reusable, and easy to test.

# Software Architecture

## Overview

CareerForge is a job tracker web app with a React + TypeScript frontend, a Spring Boot backend, and a MySQL database. The frontend handles UI and API requests, the backend manages authentication and business logic, and MySQL stores users, jobs, and application data.



## Frontend

* Pages: Login.tsx, Register.tsx, Content.tsx, MyJobs.tsx.
* Components: JobsViewList.tsx, JobCard.tsx, filters (Aside.tsx, Field.tsx, etc.), Header.tsx, Footer.tsx.
* Theme: ThemeContext.tsx for light/dark mode.
* API Client: fetch wrapper for all REST calls.

## Backend

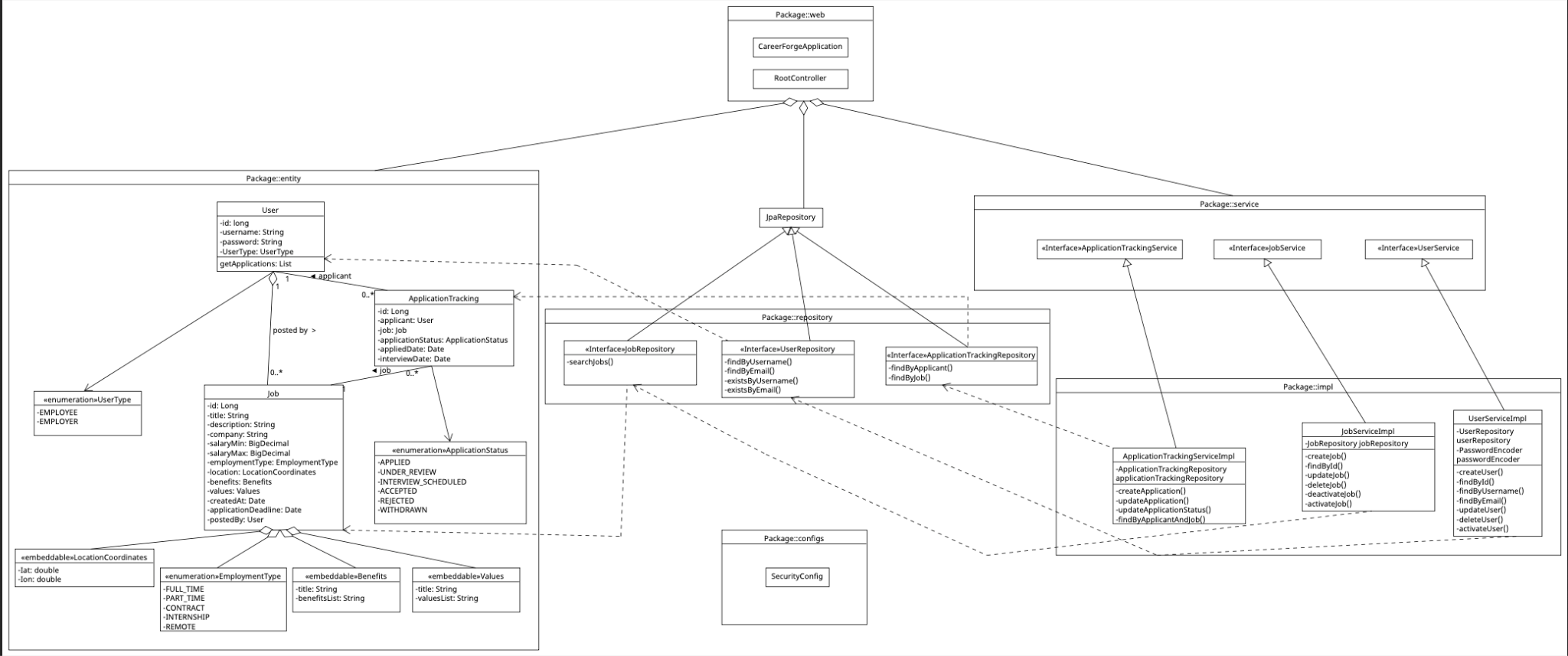
* Controllers (web/): handle REST endpoints (/auth, /jobs).
* Services (service/): business logic (auth, save/apply jobs).
* Repositories (repository/): Spring Data JPA to MySQL.
* Entities (entity/): User, Job, ApplicationTracking, SavedJob.
* Configs (configs/): Spring Security, JWT, CORS.

## API Flows

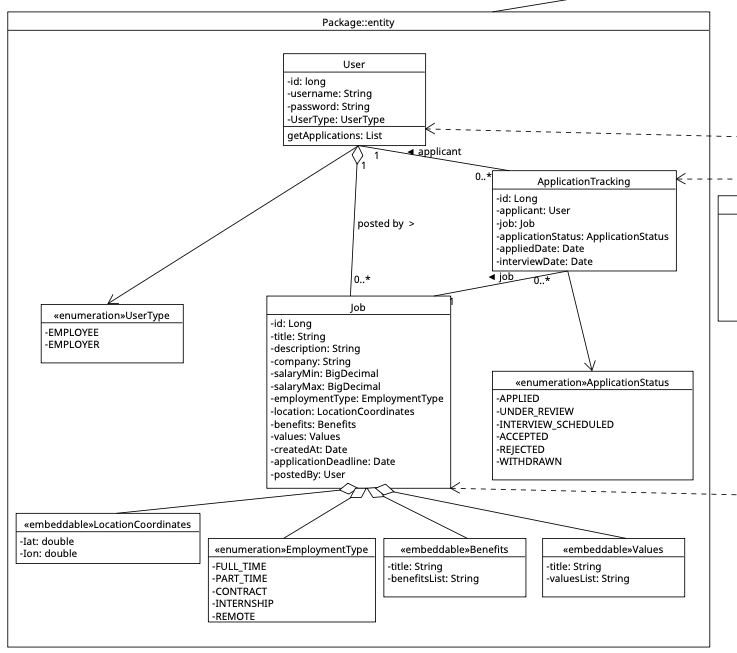
* Auth:  
  + POST /api/auth/register → create user, return JWT.
  + POST /api/auth/login → validate, return JWT.
* Jobs:  
  + GET /api/jobs → list jobs.
  + POST /api/jobs/:id/save → save job.
  + POST /api/jobs/:id/apply → apply to job.
  + GET /api/users/me/saved / applied → saved/applied jobs.
  + DELETE /api/jobs/:id → delete one saved/applied job.
  + DELETE /api/users/me/myjobs → delete all saved/applied jobs.

Frameworks: React 18 + TypeScript, Vite, Bootstrap (frontend); Spring Boot, Spring Security, Spring Data JPA (backend).

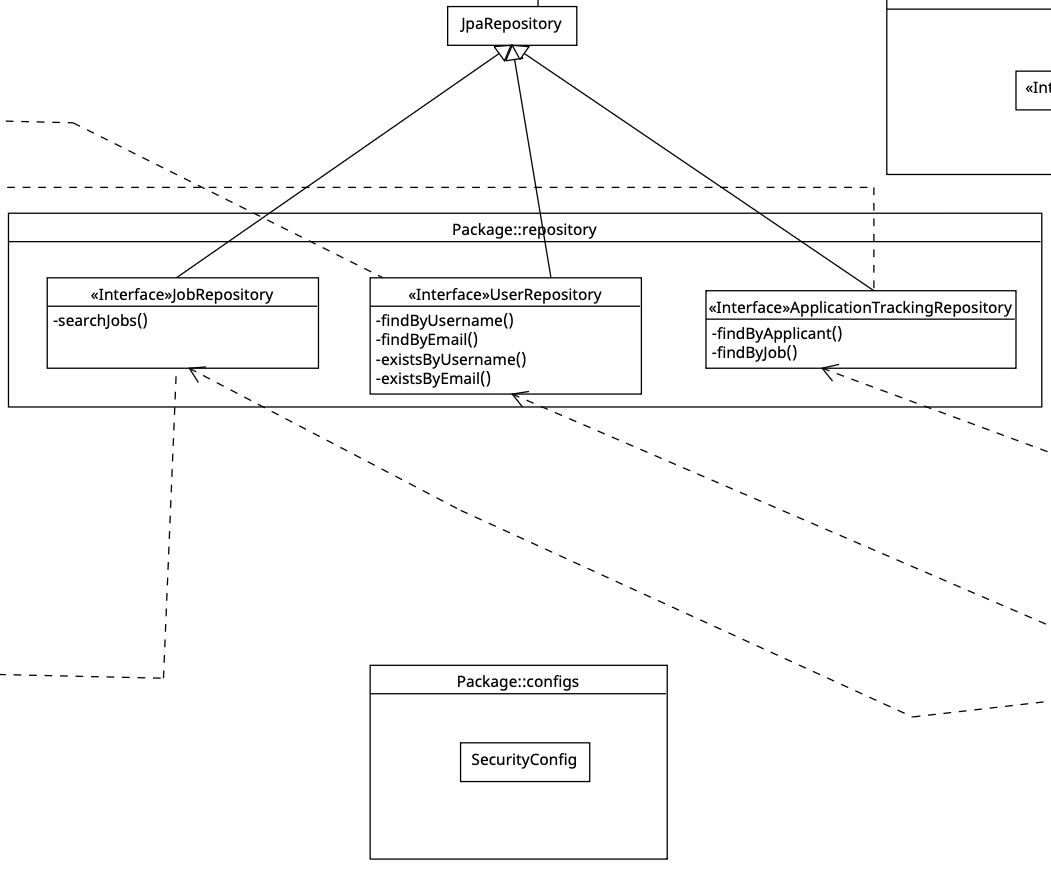
# Class Diagram

Below is an overview of our class diagram. Since it’s a little hard to read, we have broken it down further into package view.

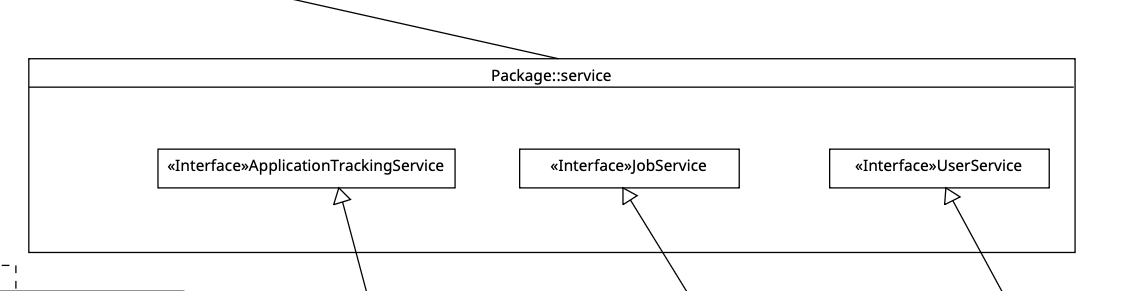
* **Package: Entity**



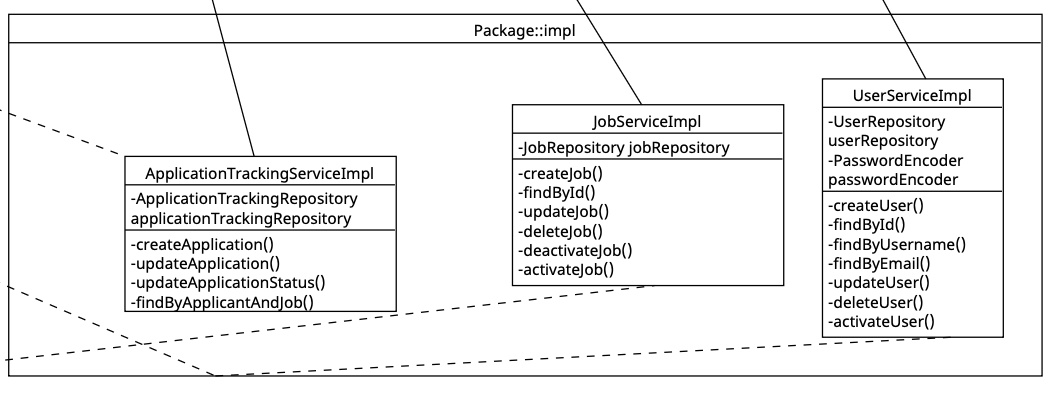
* + **Package: Repository**



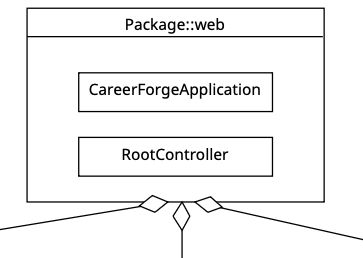
* **Package: Service**



* **Package: impl**

****

* **Package: web**



# UI Design (if applicable)

The user interface of CareerForge is designed to be simple, modern, and

intuitive, ensuring that users can easily register, log in, and manage their

job applications.

1. Login Page

- Minimalist card layout centered on the screen.

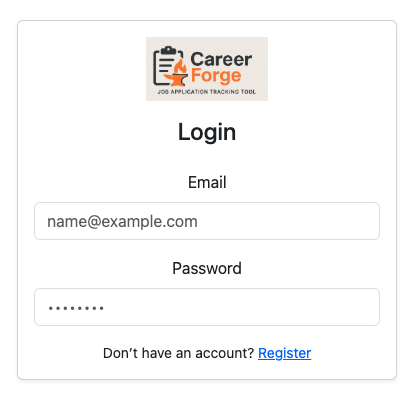
- CareerForge logo at the top for branding.

- Input fields for email and password.

- A link to navigate to the registration page if the user does not

already have an account.

- Clean use of spacing and a large “Login” heading for clarity.



2. Register Page

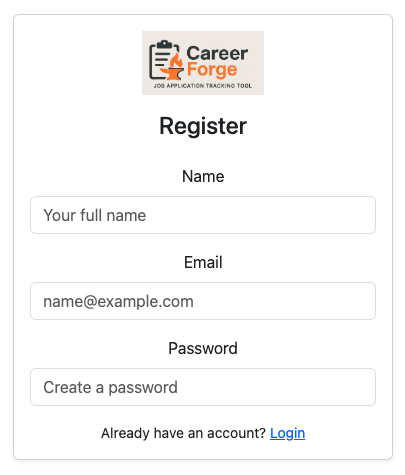
- Similar card layout to the login page for consistency.

- Fields for name, email, and password to create a new account.

- Clear labels for each input.

- A link to navigate back to the login page for returning users.

- Maintains consistent branding with the logo at the top.



3. Job Listings Page

- Main workspace once the user is logged in.

- Two-column layout:

• Left panel shows a scrollable list of job postings with company

logos, titles, and key tags (e.g., level, type, location).

• Right panel displays the full job details when a posting is selected.

- Filters on the right-hand side (Field, Location, Type) allow users to

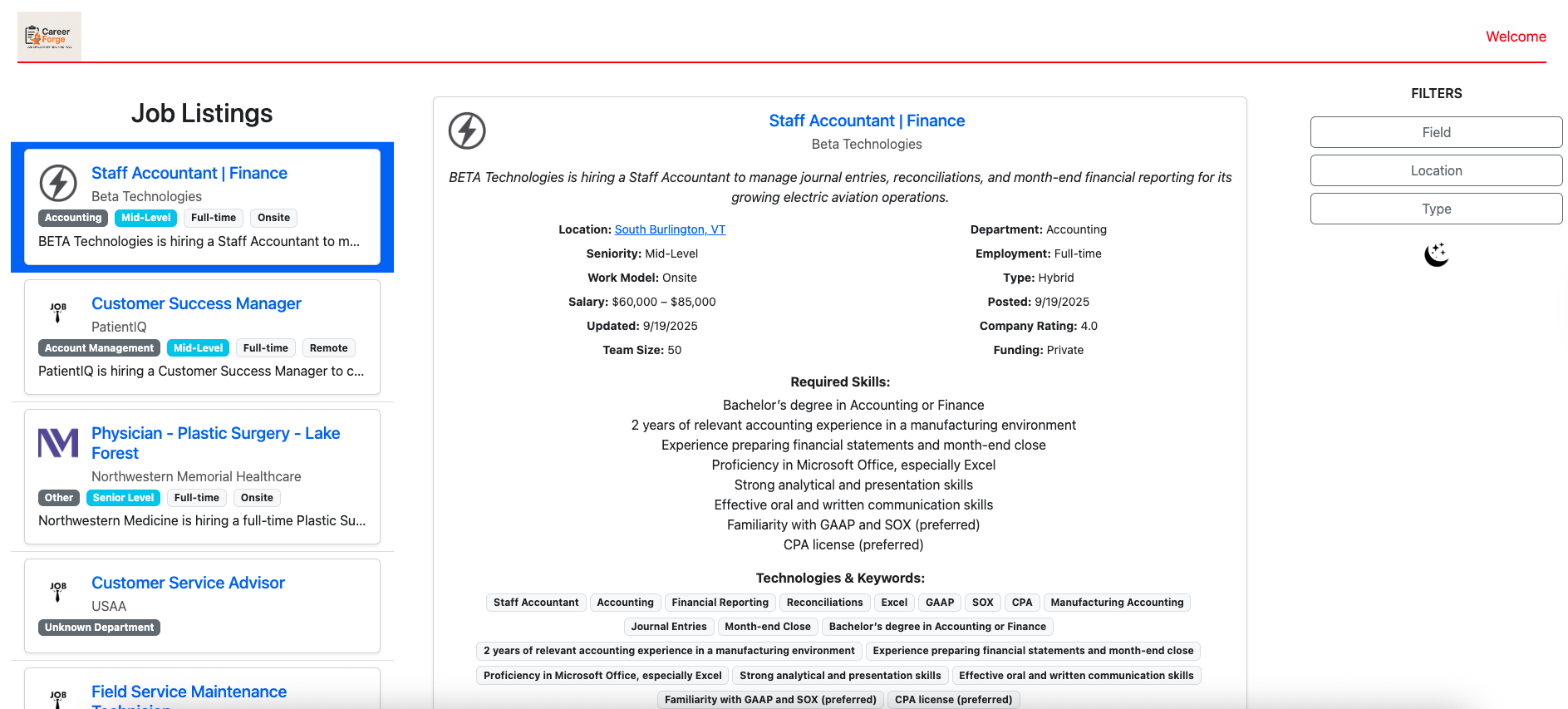
refine their job search.

- A theme toggle (light/dark mode) is included for accessibility and

user preference.

- Design emphasizes readability, with job titles and key details

highlighted using bold text and tags.



4.- MyJobs page will be created in the next iteration

Overall

- Consistent layout and branding across all pages.

- Clear navigation between login, register, and job listings.

- Responsive design ensures usability across desktop and mobile devices.

- Focus on simplicity: essential actions such as login, registration,

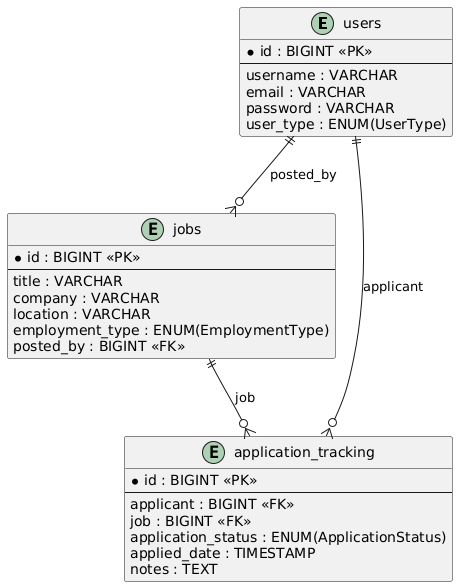
applying, and saving jobs are always easy to find.

# Database Design (if applicable)

Our system uses a MySQL relational database to persist user accounts, job postings, and application records. The schema is defined in initialization.sql, which creates three primary tables: users, jobs, and application\_tracking. These tables capture the core relationships in the system: employers (from the users table) can create jobs, while employees (also in users) can apply to those jobs. The application\_tracking table connects applicants to jobs with one row per application. It records the current application\_status, relevant timestamps, and any notes. Foreign keys make sure that every application is always linked to an existing user and job.

The database design closely follows the entity classes in the backend, such as User.java, Job.java, and ApplicationTracking.java. Enums like ApplicationStatus.java, EmploymentType.java, and UserType.java are mapped to MySQL ENUM columns, making the stored data human-readable and consistent with the domain model. Data access is handled through Spring Data JPA repositories (UserRepository.java, JobRepository.java, ApplicationTrackingRepository.java), which abstract away SQL and provide a clean interface for CRUD operations. In other words, we don’t write SQL directly; instead, we use repositories that automatically handle the database operations for us in a simple, reusable way. This design ensures that the schema supports all REST API flows, from authentication and job listing to saving and applying for positions, all while maintaining strong data integrity and performance.

**Entity–Relationship:**

****

**One user → many jobs** (an employer posts many jobs).

**One user → many applications** (a candidate can apply to many jobs).

**One job → many applications** (many candidates can apply to the same job).

# Security Design

CareerForge’s security design ensures confidentiality through encryption and access controls, integrity through hashed credentials and validated inputs, and availability through isolated deployments and monitoring. By combining JWT-based authentication, Spring Security policies, encrypted data handling, and secure DevOps practices, the system protects both user and organizational assets while remaining extensible for future role-based access control and scaling.

Authentication is implemented with JSON Web Tokens signed using a secret key. These are short-lived, with refresh tokens available to reduce exposure risk. Authorization is currently implemented using protected endpoints enforced via Spring Security filter chains. The outline for role-based logic is there to be extended later for employers versus job seekers. The idea of having an administrator role has also been raised.

For data security, all communication between client and server happens over HTTPS. MySQL database stores hashed passwords using BCrypt. User details can be encrypted with AES. JWT keys, DB credentials, and API tokens are injected at runtime via environment variables. No secrets are hardcoded in the codebase.

Our application security involves strong validations on user inputs to mitigate SQL injection and XSS. Our session management involves stateless sessions with JWT tokens that expire after a set duration defined in application.properties. Our infrastructure and CI/CD security involves sensitive credentials for the database and our JWT secret and build artifacts are scanned for vulnerabilities before deployment. Services are deployed in isolated environments with network segmentation. Maven dependencies are scanned for known vulnerabilities.

# Business Logic and/or Key Algorithms

The core business logic of **CareerForge** centers on user authentication, job management (save/apply/delete), and application tracking. These flows ensure that only authorized users can interact with job data and that all actions are recorded consistently in the database.

### 1. User Registration & Authentication

Handles secure account creation and login with JWT-based authentication.

**Pseudocode – Registration Flow:**

function registerUser(name, email, password):

if UserRepository.existsByEmail(email):

return error "Email already registered"

hashedPassword = PasswordEncoder.encode(password)

newUser = User(name, email, hashedPassword)

UserRepository.save(newUser)

token = JWT.generate(newUser)

return token

**Pseudocode – Login Flow:**

function loginUser(email, password):

user = UserRepository.findByEmail(email)

if user is null:

return error "User not found"

if !PasswordEncoder.matches(password, user.hashedPassword):

return error "Invalid credentials"

token = JWT.generate(user)

return token

### 2. Saving a Job

Allows a user to bookmark jobs for later review.

**Pseudocode:**

function saveJob(userId, jobId):

user = UserRepository.findById(userId)

job = JobRepository.findById(jobId)

if job is null:

return error "Job not found"

savedJob = SavedJob(user, job)

SavedJobRepository.save(savedJob)

return success "Job saved"

### 3. Applying to a Job

Tracks when a user submits an application to a job.

**Pseudocode:**

function applyJob(userId, jobId):

if ApplicationTrackingRepository.exists(userId, jobId):

return error "Already applied"

application = ApplicationTracking(userId, jobId, status="APPLIED", timestamp=now)

ApplicationTrackingRepository.save(application)

return success "Application submitted"

### 4. Viewing User’s Saved and Applied Jobs

Fetches data based on user identity.

**Pseudocode:**

function getUserJobs(userId):

saved = SavedJobRepository.findByUserId(userId)

applied = ApplicationTrackingRepository.findByUserId(userId)

return { "saved": saved, "applied": applied }

# Design Patterns

**MVC -** A controller like UserController handles the HTTP request, turns it into a simple input object, and hands it to UserService. The service does the real work and returns a result the controller can serialize back to JSON. Persistence is not the controller’s problem—UserService leans on UserRepository for that. This keeps web concerns, business rules, and data access cleanly separated.

**Repository Pattern.** A repository such as UserRepository hides the database behind a small interface. Services call methods like findByUsername or save without writing SQL or touching EntityManager details. The repository returns domain objects and accepts them for persistence, so the service code reads like business logic instead of database plumbing. Swapping query details or the underlying store doesn’t ripple through the app.

**Facade Layer.** A service like UserService acts as the facade that controllers talk to: one entry point for “register a user,” “log in,” or “update profile.” Inside, it coordinates the repository and helpers (for example, a password encoder) so callers don’t juggle multiple components. Controllers stay thin and predictable because all rules and sequencing live behind the facade. That makes behavior easy to test and change in one place.

**Factory Method.** In configuration (e.g., SecurityConfig), @Bean methods act as factory methods that build ready-to-use singletons like SecurityFilterChain, AuthenticationManager, and PasswordEncoder. Spring calls these once, manages their lifecycle, and injects them wherever needed. Construction logic—algorithms, flags, and wiring—lives in one spot instead of being scattered across the app. Changing an implementation later is a config edit, not a code rewrite.

# Rest APIs

The API lives under a single base path (/api) and speaks JSON. Public routes are carved out explicitly (/public/\*\* and /auth/login) while everything else flows through Spring Security and can be tightened once JWT is fully on. The stack is intentionally stateless for APIs (CSRF off, sessionless), with clean separation of concerns: controllers handle HTTP, services hold business rules, repositories do persistence. Responses avoid sensitive data (no passwords), use standard HTTP codes (create vs. read vs. errors), and expose basic observability via Actuator (health, mappings).

# Any Additional Topics you would like to include.

# AI usage Log

You are allowed and even encouraged to use AI tools to help you generate the project idea, plan it and build it, but you need to clearly describe 1) What tools were used? 2) for what specific tasks and 3) Is it helpful? 4) how did you evaluate or modify AI-generated content? Additionally, you should submit the exported AI chat history as an appendix or share that with the instructor and facilitators.

| Tools | Who | Tasks | helpful | Evaluation/modification | links |
| --- | --- | --- | --- | --- | --- |
| ChatGPT | James Rose | Assist in the development of the Aside sidebar on the content page. | Very helpful | After getting a general framework of what I wanted, a bunch of tinkering had to be done to ensure the pop up windows were in the proper spot, exitable, and displayed what I wanted. While AI is very good at creating the window with features, it is not good at sizing and understanding what the user will see. | https://chatgpt.com/share/68cdcba0-1218-8006-87a6-66d632a41ec8 |
| ChatGPT | Gopi | Assist in filling out the details of API and Design Patterns based on the current code base. | Yes | Originally the response was fragmented and bulleted and I had to do some prompt work to get it to give me a response in the proper form. | <https://docs.google.com/document/d/1bn1-OdCbJgEjOeXCMfUaSKAWudefqPqm2fHJyIyxZM8/edit?usp=sharing> |
| ChatGPT | Yongxiang | Assist in filling out the intro and the business logic | Yes | I need to make my prompt clear so that GPT can give me a more detailed introduction. | <https://chatgpt.com/share/68ce1bd3-5c68-8006-869d-be45da0ccafd> |
| ChatGPT | Stacey | Assist in figuring out where to begin with implementing user authentication and create unit tests for tokens | Yes | It did a great job of laying out step by step what to do and where to begin, but it doesn’t live inside my project. It instructed me to create certain files that already existed so I had to figure out how to manipulate the existing files without causing any issues. AI was used here because I was not sure where to begin with this implementation. | <https://chatgpt.com/share/68d08706-4f98-8002-8816-d7eb68ab08d3> |
| ChatGPT | Stacey | Assist with auto generating a key for dev. | Yes | It clearly laid out what needs to be done in order to not rely on a hard coded key. It also helped me implement this change for the tests as well. | <https://chatgpt.com/share/68d08a1b-0d3c-8002-9b7d-570e0fef87a7> |

# 

# References

# Glossary