

# Iteration 1 - ResumeAI

Summary of Progress

---

Team Members: Amruth, Jaindra,  
Faizan, Tushar, Shubh, Hemant

---

# Summary of Program in Iteration 1

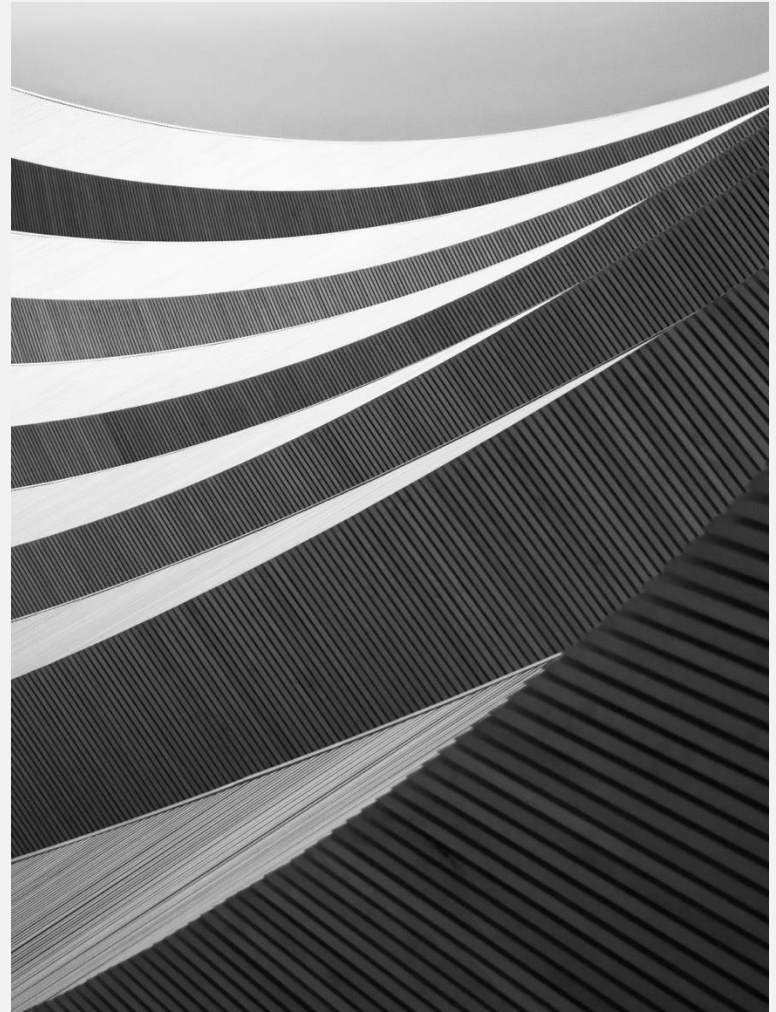
---

## Key Achievements:

- Setup of GitHub repository, Docker, and initial backend structure.
- Progress on signup, login, and resume upload features.
- Challenges faced with integration and unfamiliar technologies.

# User Stories (Features) Worked On

- Signup & Login
- Resume Upload & Matching Score Calculation
- Tool Used: Jira
- Prioritized and tracked each story with story points and statuses.
- The user stories we worked on include **Signup & Login** and **Resume Upload & Matching Score Calculation**, both essential features as outlined in the project plan. These stories are tracked using **Jira** and are prioritized with story points to ensure steady progress through the iteration.



# SCRUM Task Progress - Iteration 1

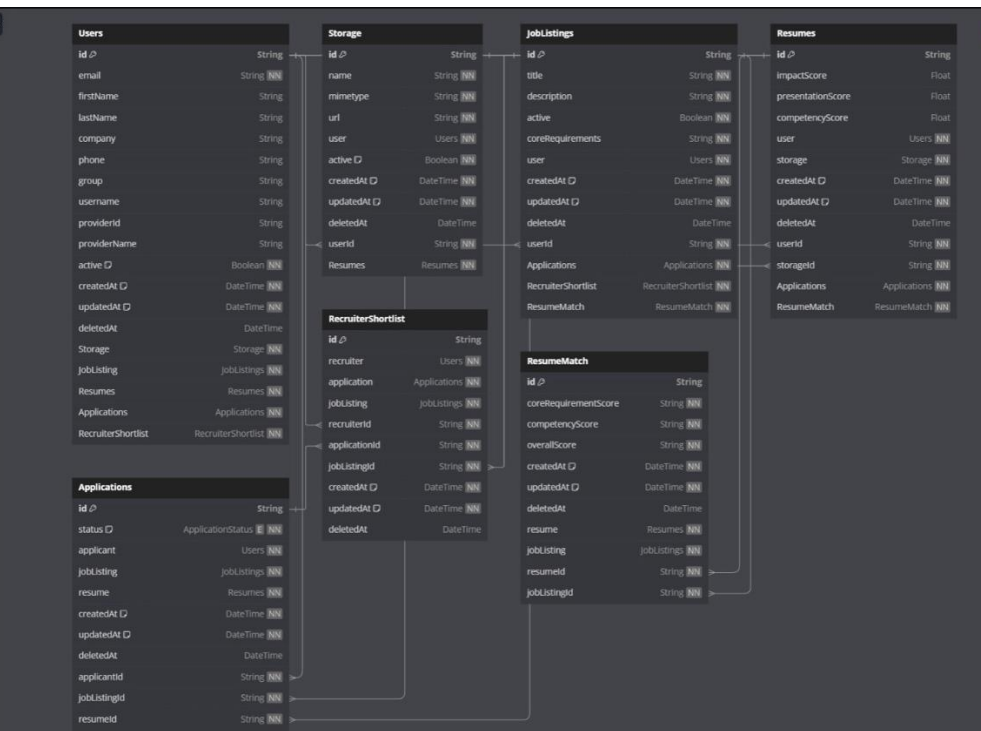
| Issue ID   | Description                                    | Status      | Story Points | Assignee |
|------------|--|-------------|--------------|----------|
| SCRUM-56   | Integrate React                                | DONE        | 2            | FA       |
| SCRUM-57   | Integrate J                                    | DONE        | 1            | FA       |
| SCRUM-59   | Create Layout                                  | TO DO       | 1            | FA       |
| SCRUM-60   | Create table                                   | PR REVIEW   | 2            | TK       |
| SCRUM-33   | Bulk Resume Upload                             | IN PROGRESS | 5            | TK       |
| SCRUM-49   | Recruiter Upload                               | IN PROGRESS | 3            | TK       |
| SCRUM-40   | Reset Password                                 | TO DO       | -            | TK       |
| SCRUM-69   | Integrate React router - Landing page          | In progress | 2            | JP       |
| SCRUM-66   | Figma component                                | DONE        | 3            | JP       |
| SCRUM - 67 | Figma pages                                    | DONE        |              | HK       |
| SCRUM - 70 | Integrate React router - Login page            | IN PROGRESS |              | jp       |
| SCRUM - 71 | Integrate React Router - App.tsx               | IN progress |              | JP       |
| SCRUM - 72 | Integrate React router - signup page           | IN progress |              | HK       |
| SCRUM - 73 | Integreate React Router - Forgotpasswpord page | IN PROGRESS |              | HK       |

# Implementation and Testing

---

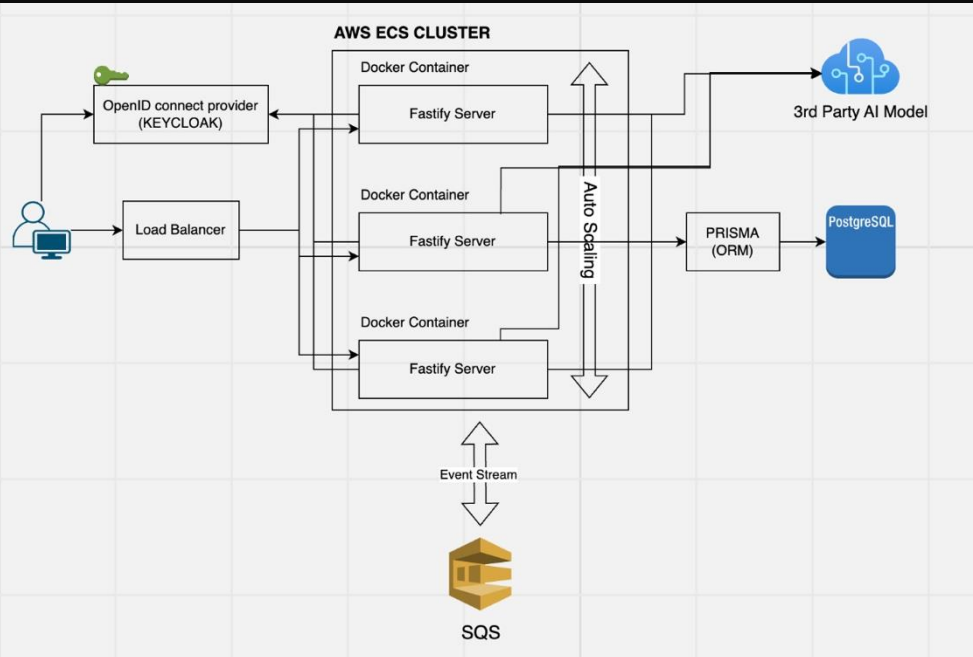
- Implemented Features:
- Resume Upload
- Testing:
- Unit testing (Jest for frontend, integration testing for backend)
- Code coverage metrics tracked
- Unit testing is still in progress, with **Jest** used for frontend testing and **integration testing** for the backend. Code coverage metrics are being actively tracked to ensure thorough testing. While testing is ongoing, no major issues have been identified so far.
- Unit testing is still in progress, with Jest used for frontend testing and integration testing for the backend. Code coverage metrics are being actively tracked to ensure thorough testing. While testing is ongoing, no major issues have been identified so far.

# DataBase Diagram



- **User-Centric Structure:** The schema is built around the **Users** table, linking users to key elements like **Resumes**, **Job Listings**, **Applications**, and **Recruiter Shortlists**, ensuring a comprehensive view of both applicants and recruiters.
- **Job and Resume Matching:** The **ResumeMatch** table connects **Resumes** and **Job Listings** by calculating scores like **coreRequirementScore** and **competencyScore**, helping recruiters match applicants to job requirements effectively.
- **Document and Application Management:** The **Storage** table manages uploaded documents (resumes), and the **Applications** table tracks the application flow, linking users, resumes, and job listings to streamline the recruitment process.

# Software Architecture



- Tech Stack: React, Fastify, Prisma, PostgreSQL, AWS (ECS, RDS)
- Mapped architecture to the code structure.



# Data Flow

- **User Interaction:**

The user interacts with the system through a web interface or client. Requests are routed through a Load Balancer to distribute traffic across available backend resources.

- **Authentication and Authorization (Keycloak):**

Keycloak handles user authentication, providing secure tokens upon login.

It communicates with both the Load Balancer and Fastify Servers to ensure authorized access.

- **AWS ECS Cluster:**

Core processing happens in an AWS ECS Cluster, which runs multiple Docker containers.

These containers host Fastify servers that handle application logic, including user requests and backend interactions.

- **Database Layer (PostgreSQL and Prisma ORM):**

The system uses PostgreSQL for data storage.

Prisma ORM facilitates smooth data interactions between Fastify servers and the database.

- **3rd Party AI Model Integration:**

For advanced AI tasks (e.g., resume matching), Fastify servers interact with a 3rd Party AI Model.

Data is processed based on business logic and returned to the server.

- **Event Streaming (AWS SQS):**

Asynchronous events, such as logs or notifications, are sent to AWS SQS for further processing.

SQS manages the event stream generated by the system.

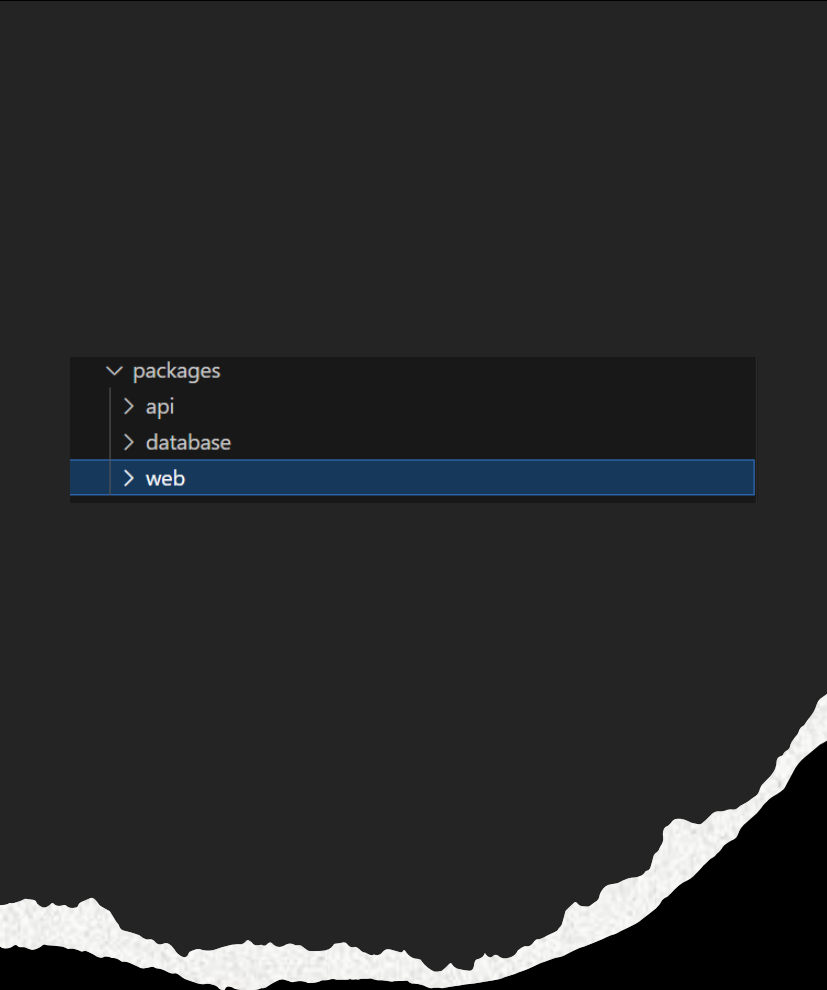


# Configuration Work

- Branching Strategy:
- GitHub branching strategy used, with individual branches for features.
- Code Review:
- Pull requests reviewed by at least 2 members before merging.
- No major bugs identified.
- CI/CD: Initial setup using GitHub Actions and AWS ECS.

# Code Setup

- We use lerna and yarn workspaces to manage aspects of the code namely:
  - API – the backend server (Fastify)
  - Database – for database design, management and prisma operations
  - Web – the React.js frontend



```
✓ packages
  > api
  > database
  > web
```



# API

- fastify is used to create backend REST APIs
- Using fastify plugins, authentication pre-handler is setup for verifying JWT token(Authorization header) of an incoming request with the SSO server
- Validations are done using 'yup'
- Appropriate error handling with custom error codes has been setup and will be expanded in future

# Database

- Prisma ORM is used for handling database configurations and operations
- New changes to database design is done via migrations
- Every database change also creates a dbml file to visualize the database

# Web

- React js is the base framework for frontend development
- Vite is used as a build tool
- Material UI is the core UI package
- Keycloak SSO has been implemented for login, logout and other functionalities
- Redux is used for state management

# API Implementation



- We always use asynchronous functions in Fastify's request handlers to ensure smooth, non-blocking operations.
- Types are defined for the request body to maintain consistency and enforce validation rules.
- The request body is verified when applicable to ensure all necessary data is valid before processing.
- The request body is verified when applicable to ensure all necessary data is valid before proceeding.
- We then perform the required database operations efficiently.
- Finally, appropriate status codes and error messages are returned to ensure clear and correct responses.

# Plan for Next Iteration

---

- Tasks:
- Complete backend setup for CRUD operations.
- Integration testing of frontend and backend.
- Implement UI/UX refinements.
- User feedback and testing for MVP features. The next
- Iteration will focus on completing the backend setup for CRUD operations, performing thorough integration testing between the frontend and backend, and refining the UI/UX. Additionally, gathering user feedback and conducting testing on MVP features will be prioritized to ensure readiness for deployment.



# DEMO

