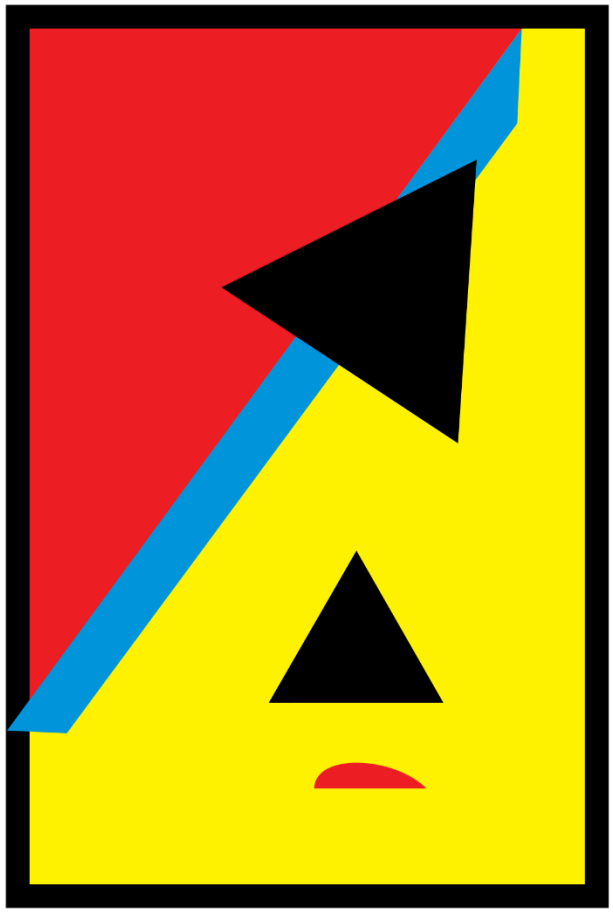
Technical Report

Referral Application Recruitment



ALTEN

Contents

Document guide 4
Version 4
Authors 4
Data model 5
  User 5
  Profile 5
  Referral 5
  Scoreboard 6
  Task 6
  Candidate 6
  Reward 6
Flow diagrams 7
  Flow legend 7
  Login 8
  Register 9
  Create scoreboard 10
  Create task 11
  Assign task 12
  Assign reward 13
Implemented technologies 14
  Code structure 14
    Web application 14
    API 14
    Database 14
  Version control 15
  Documentation 15
Application suite 15
Code performance 15
Code maintainability 16
Test plan 16
  Unit tests 16
    Web application 16
    API 16

Real-world test 17
Results 17
References 18

Document guide

This document provides a technical insight of the Alten recruitment system. As this document is written with the intent of going deeper into the technical side of the project it is advised to only be read by people with some knowledge into the technical sides of a project, this will include data structuring.

Version

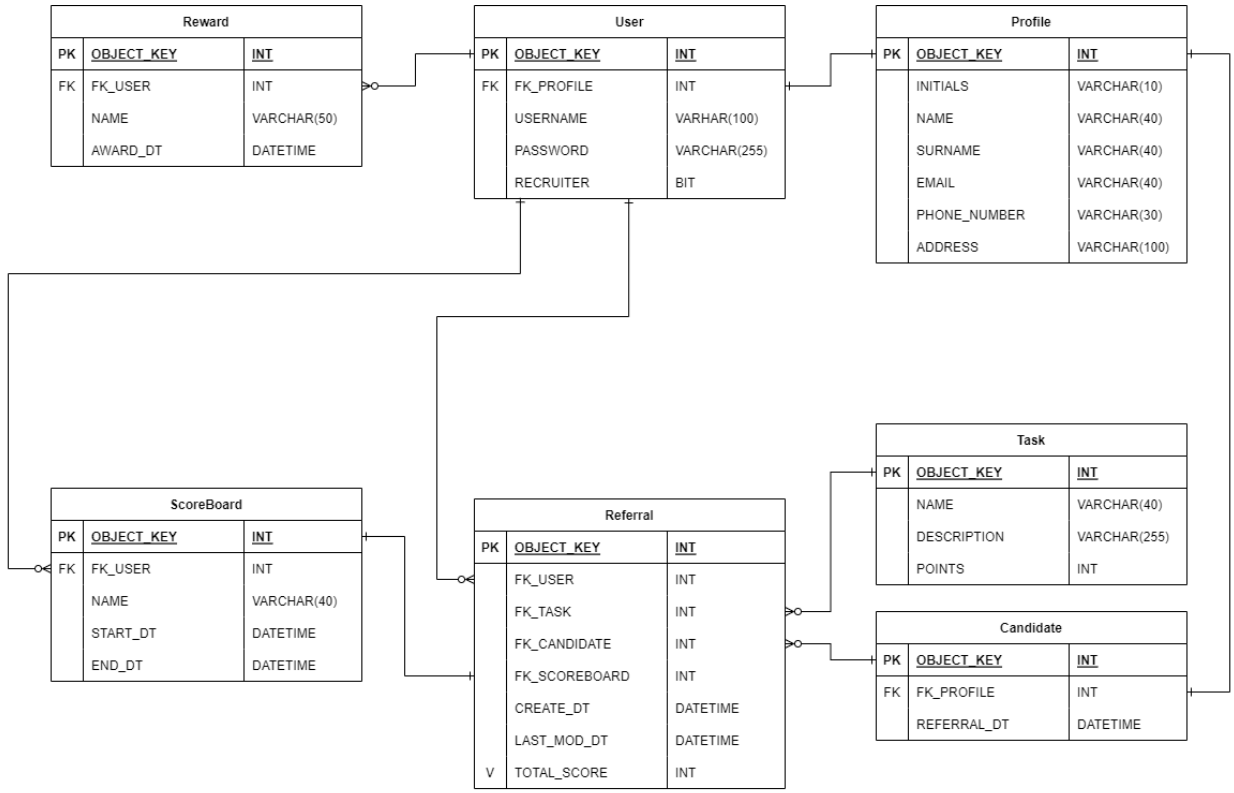
Version	Date	Description
1.0	12-09-2022	Document creation
1.1	26-09-2022	Added datamodel
1.2	30-09-2022	Updatet datamodel
1.3	05-10-2022	Added technologies and references chapters
1.4	06-10-2022	Added test plan chapter and reviewed document
1.5	10-10-2022	Updated based on sprint 1 review
1.6	13-10-2022	Added flow diagrams
1.7	14-10-2022	Reviewed document
1.8	19-01-2023	Reviewed and updated specifications based on current product and feedback
1.9	06-02-2023	Reviewed and updated for deliverable

Authors

Name	Student number
Jeremey Sas	1028294
Michael Heerkens	1028285
Marchano Gopal	1038165
Tobias Roessingh	1042623

Data model

In the UML down below, we have the Entity Relation model designed for this project.



User

A user is an entity used by Alten personal to log into the application. This user has a profile assigned to it containing information that could be useful for contacting the user.

Profile

The profile is an entity created to store personal data. This entity is separate from a user as it will also be used to store personal data for a candidate.

Referral

The referral entity will be used to track a user's progress with referring new candidates. This entity will be linked to a user, the task that was performed by the user, the candidate that was brought in and to one or more scoreboards in which the user is partaking.

Scoreboard
The scoreboard entity is used to track the progress of all users linked to a specific scoreboard and to allocate the correct position to the correct user based on the total score of a user partaking in the scoreboard.

Task
The task entity is used to configure what action will reward what kind of points, for example: if a user manages to get a candidate on a phone call to be recruited, the user will receive 5 points.

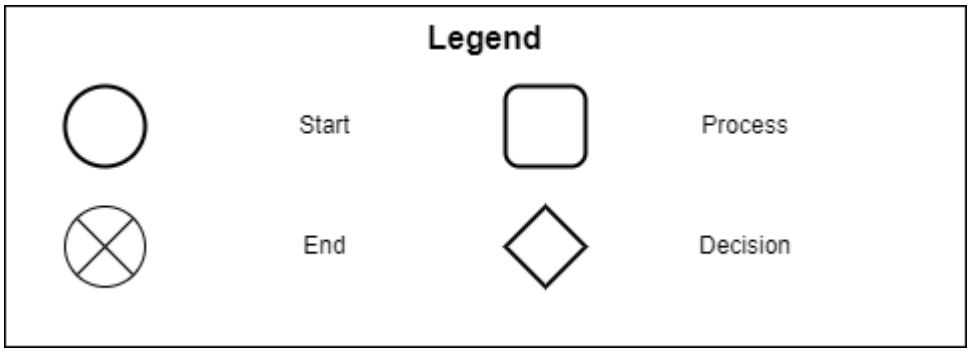
Candidate
The candidate entity is used to track who has been referred and what his/her contact information is, this could be used for keeping in touch with the recruited candidates.

Reward
The reward entity is used to track what rewards have been sent, when they have been sent and to whom they have been sent.

Flow diagrams

In the following headings we will describe each functionality at its core and per functionality show the designed flow in a flow diagram.

Flow legend



The Start symbol is an empty circle, this indicates the start of the flow, this symbol will only occur once in a flow chart and is used to show if, and if so, what functionalities can be interacted with.

The End symbol is a circle with a cross, this indicates the end of a stream from the flow, this symbol could occur once or multiple times, depending on the different flows available in the functionality.

The Process symbol is a rounded rectangle, this indicates an action that can has to be performed by either the user or the program, for example: 'The user requires to input a value'. This symbol is used when any sort of input or output is required.

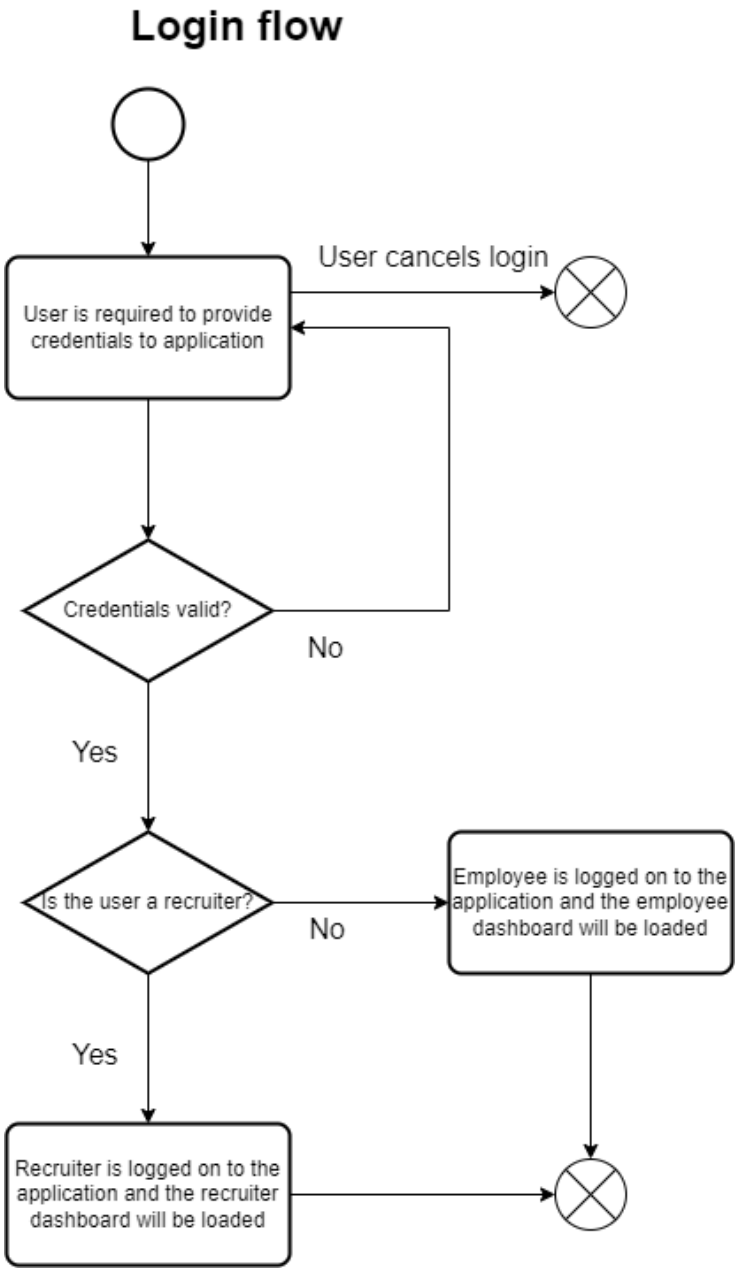
The Decision symbol is the diamond shape, this indicates a decision that either the user or the application has to make, for example: 'The user is an administrator'. From a decision there are usually two routes, yes or no, based on the decision the application could progress into a functionality differently or even end the flow of the functionality.

Login

To gain access to the application, a user has to log into the system with valid credentials.

If the user provides valid credentials the system will check if the user is either a recruiter or an employee, based on this the system will provide the corresponding dashboard.

The user can cancel the login at any time, this will stop the user from accessing the system.



Register

When a new user wants to access the system, the user will need to provide unique credentials.

The user can always back out of creating a new user.

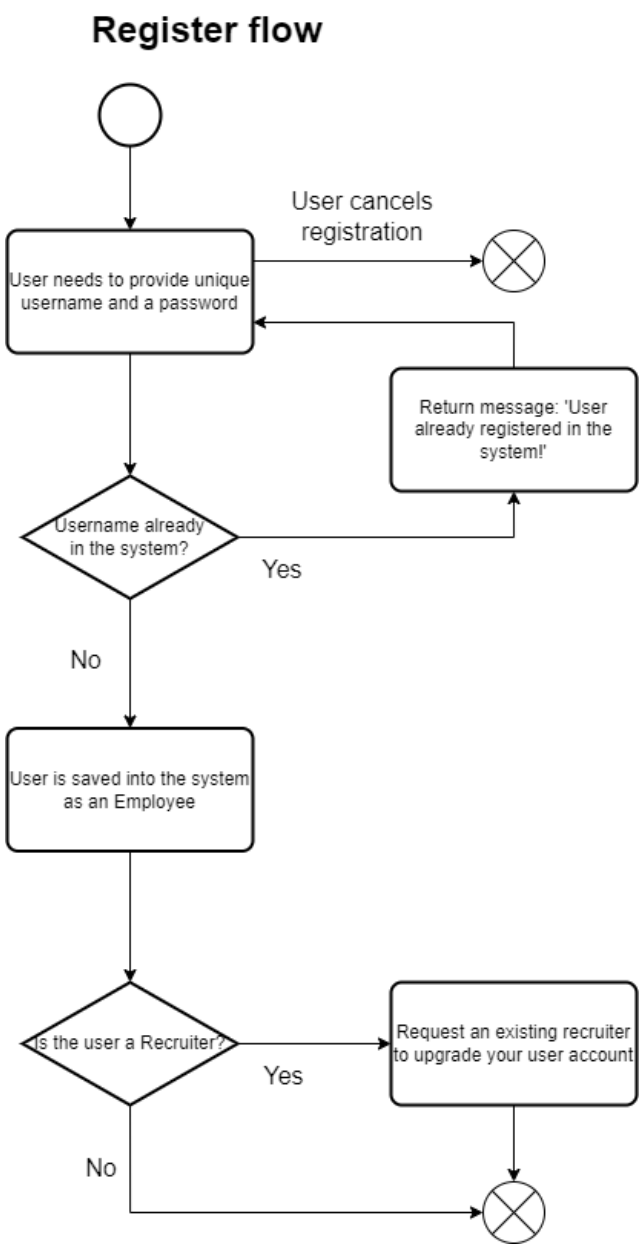
When a new user is created, by default this is set as an employee user, this will ensure the user can only read the data linked to this user.

If a user needs the Recruiter role an existing Recruiter needs to upgrade the role of this user through the system.

The recruiter can always remove users from the system to challenge users with malicious intents.

The initial Recruiter needs to be the System Admin and needs to also have write access to the database, as to become the initial Recruiter the user needs to register its credentials and manually toggle the user to be a Recruiter.

See manual for more information to regards to initial system setup.



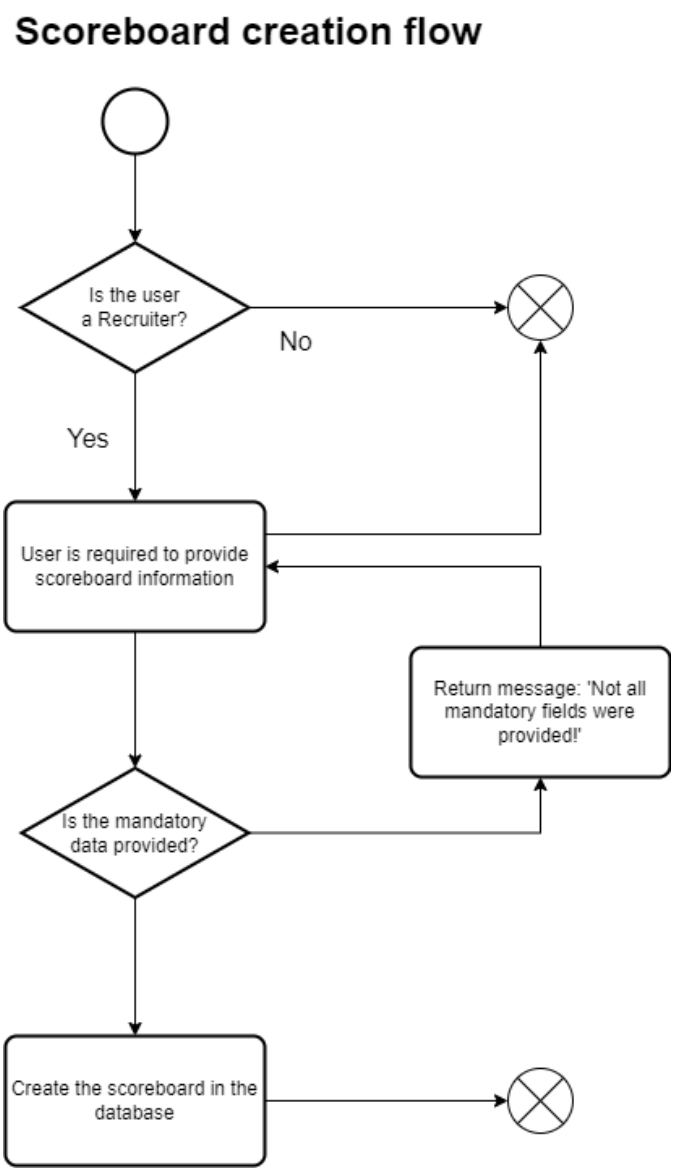
Create scoreboard

To create a scoreboard a Recruiter is required to provide the mandatory data to the API.

See manual for more information on endpoints and the data that is mandatory and optional to provide to an endpoint.

If a user that is not a Recruiter tries to access this endpoint, through custom scripts or tools like Postman, the endpoint will always return in an error status code.

Only Recruiter users are allowed to create scoreboards into the system.



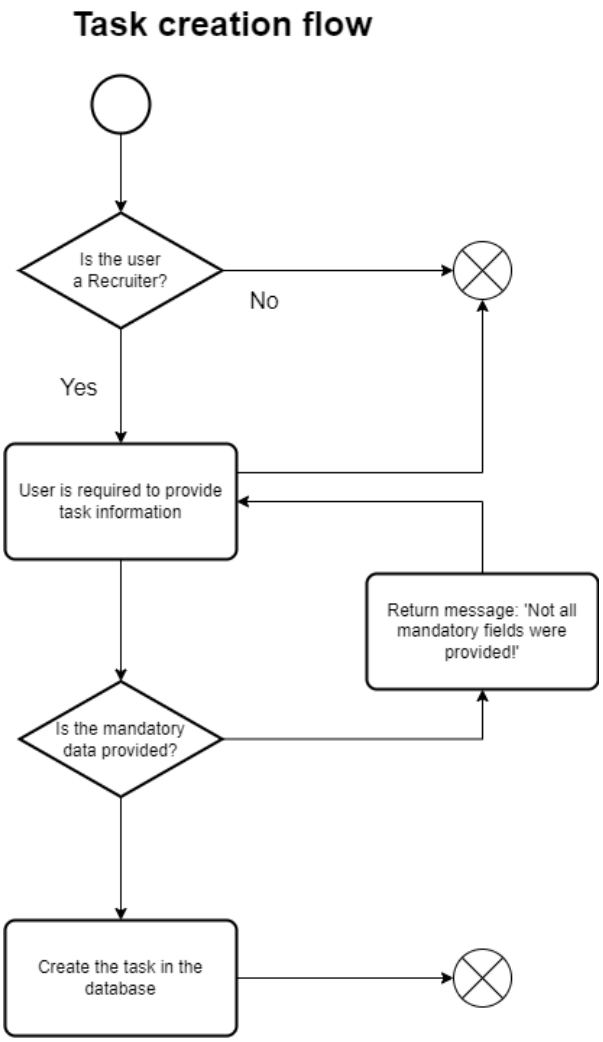
Create task

To create a task a Recruiter is required to provide the mandatory data to the API.

See manual for more information on endpoints and the data that is mandatory and optional to provide to an endpoint.

If a user that is not a Recruiter tries to access this endpoint, through custom scripts or tools like Postman, the endpoint will always return in an error status code.

Only Recruiter users are allowed to create tasks into the system.



Assign task

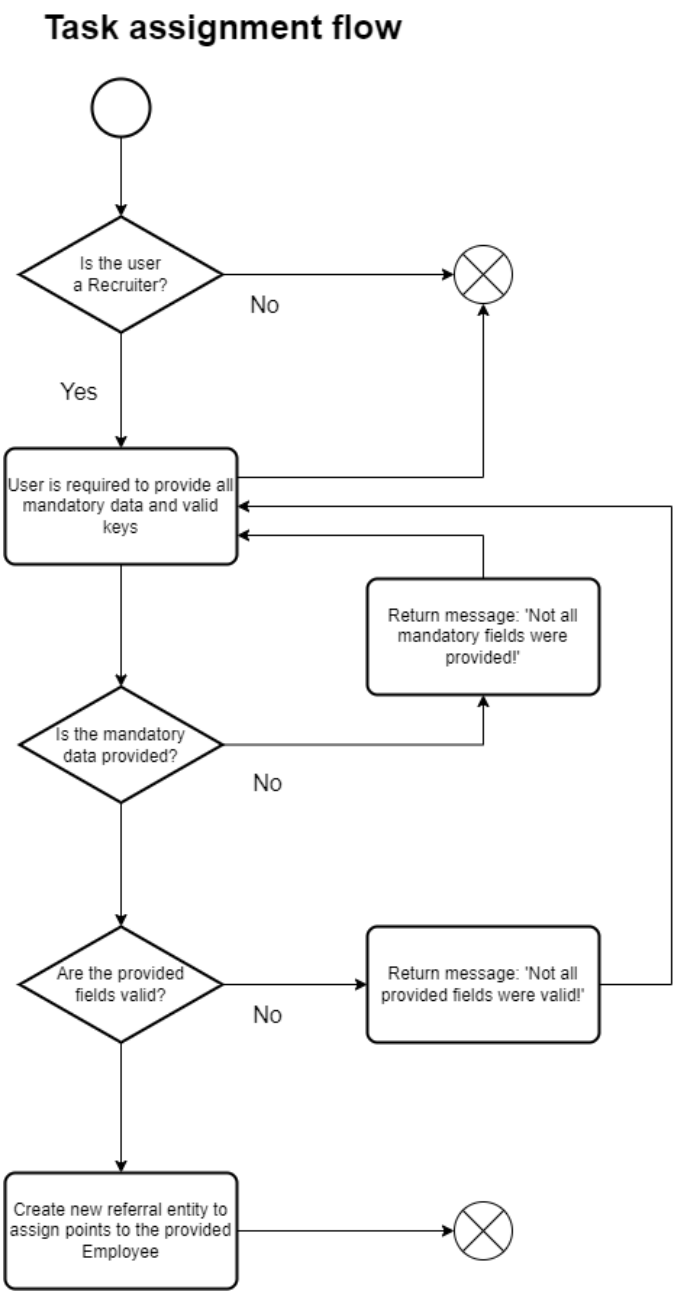
To assign a task to an Employee the Recruiter needs to provide the mandatory data to the API.

See manual for more information on endpoints and the data that is mandatory and optional to provide to an endpoint.

After providing all the data one or more referral entities will be created, this is based on how many scoreboards the Recruiter has assigned the points to for the provided Employee.

For displaying the points of an Employee in a specific scoreboard a sum of the points for the Employee in a specific scoreboard can be requested from the API.

See manual for more information.



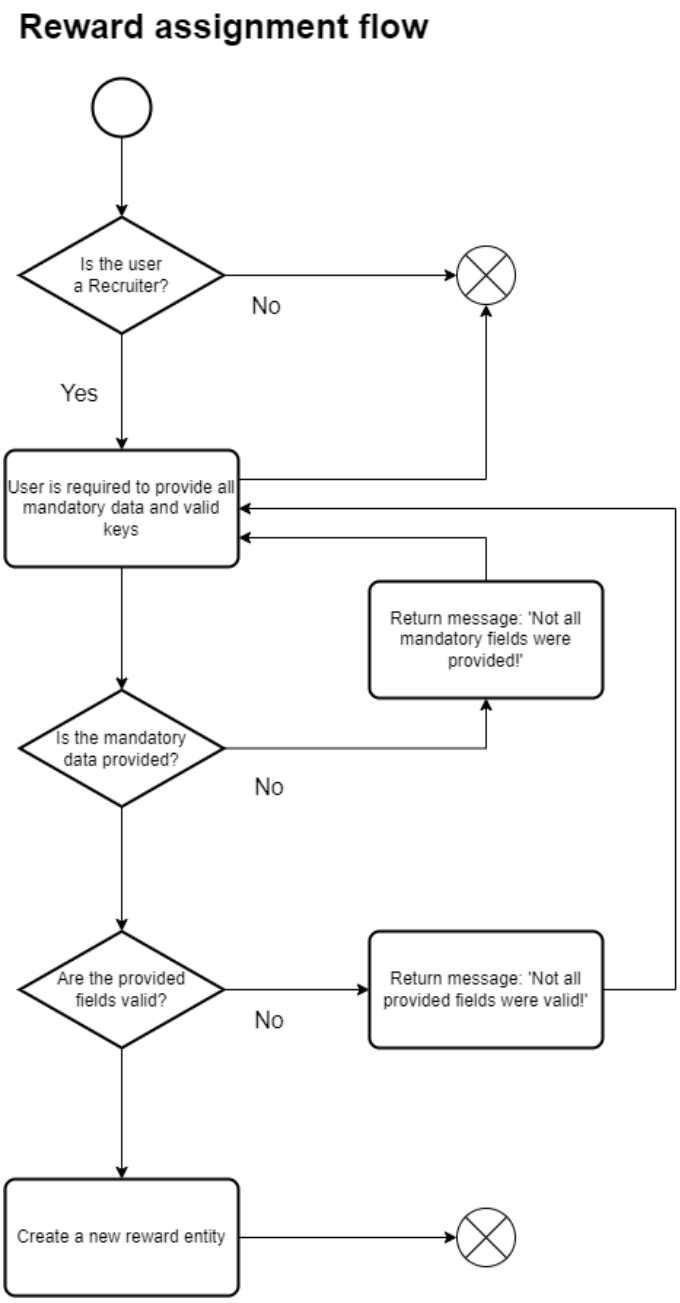
Assign reward

To assign a reward to an Employee the Recruiter needs to provide the mandatory data to the API.

See manual for more information on endpoints and the data that is mandatory and optional to provide to an endpoint.

After assigning a reward a Recruiter needs to take physical actions to ensure the reward is awarded to the provided Employee.

This follow up process is to be managed by Alten and their Employees.



Implemented technologies

Code structure

Web application

The web application will be developed in the Vue JS framework as this will allow us to setup an environment in no time, it will allow to structure and manage the code better and it will allow the code to be tested through unit tests designed to test the code from a logical stand point.

Our choice for Vue JS came after some debate of which web framework to use, here we wanted to use either Angular JS or Vue JS, in the end we chose Vue JS as it does not have a steep learning curve as compared to Angular, which allows us to spend more time developing the web application, instead of figuring out how certain functionalities inside the framework should be used.

API

The API will be developed in C# using Visual Studio Code and Visual Studio 2022. This will allow for integration with the database structure of our choice and will also provide a way of unit testing.

Our choice to make an API and to make it in C# came from an ease of access point of view, as this would allow the Customer to slightly migrate into this recruitment system from their Excel sheets without having to go through the web application and learning how the interface is used.

This also allows us to split the front- and backend of the recruitment applications so the web application will not have to worry about the logic of scoreboards, the creation of entities into the database, authentication & authorization and staying true to the designed data model.

Database

The database used to store the data of the recruitment system is Postgre SQL, this was used because it is easily accessible on both Windows and Mac OS, this was an important issue as one of the developers uses a Mac, this eliminated some database structures as they were not as accessible.

Version control

For the version control we will be using GitHub, this because it is a web-based version control service and it can be made accessible to the people involved with the project.

Documentation

For documentation we are using Office 365 Word, this choice is made as it is easily accessible for everyone in the development team and thus ensures that everyone could work on the documentation at any time.

Application suite

This project contains the following applications:

- Vue JS web application
- C# REST API
- Postgre SQL database

These applications combined make for the projects application suite and allows the customer to take advantage of the ability to further develop each system separately to meet with future functionalities.

Code performance

The products that will be produced will work responsive and stable. This will be tested upon to ensure the produced code lives up to both the standard of speed and of reliability. The following tests will be performed to ensure the produced code is both reliable and fast:

- Fetching data should not take longer than 2 seconds per record
    - This will be achieved by using caching and other technologies
- Creating, updating and deleting data should feel responsive and should never take longer than one minute
- After creating, updating or deleting data, these changes should be both saved and represented by the displayed information
- The API needs to b able to handle multiple requests at the same time
- Handling API calls should never take longer than five minutes
- Functionalities should always behave the same, even after the API has been running for a long time

Code maintainability

To allow for code maintainability the developers creating the products will regularly make us of code comments, this will allow for descriptions of (parts of) the functionality.

The delivered code will be maintainable by developers that have common know-how of C# coding and Vue coding using the Quasar framework.

For added maintainability code commentary is put in places where it seemed required to explain the functionality/purpose of a consecutive piece of code.

Test plan

Unit tests

During the development of the application there will be designated unit tests created to ensure the functionality of a piece of code. This allows the developers to not worry about introducing unintended bugs into the system.

Web application

In Vue JS the unit test will ensure every piece of code can be reached, based on the provided parameters. If a unit test fails, the developer will be able to quickly source the piece of failing code and this will reduce time spend on bug hunting.

Creating unit tests in Vue JS will allow the application to handle some testing prospects, based on the unit tests put in place.

API

In C# the unit tests will ensure the functionality, integrity and stability of the code. For each data entity and the business logic linked to the entity a unit test will be created. This unit test will test the reading, creation, modification and removing of an entity from the database using both correct and incorrect parameters.

Doing these entity-based unit test will allow us to pin point which data entity could be generating bad behaviour and in which case/cases this could be.

Creating unit tests in the C# API will allow the application to handle logical testing and will allow us to focus on process flow, instead of logical errors.

Real-world test

The real-world test will include a checklist that is setup in a way to ensure all the requirements are met when this checklist is passed.

This real-world test will be executed at the end of the development cycle and will allow for ensuring the useability of the product and to assure the quality of the provided code.

If during this checklist any functionality point is met with a failure, it would result in the product not being as requested by the customer and with such needs to either be reworked if time allows or, in consultation with the customer, the functionality can be reduced or scrapped from the scope.

Results

With the results of the above-mentioned test implementations, we can track the progress of the application and check if the program is functioning as it is required by the customer.

When the project is delivered to the customer a document will be provided containing suggestions for a follow-up team on how to further improve the applications.

References

Name	Type	
Master checklist	Test plan	Provided in attachment
Vue documentation	Guide	Link
ERD guide	Guide	Link
Click-up	Sprint board	Link