Gamification of Data

***Design Document***

**Senior Design Project - CIS 4951 - Fall 2019**

**Version No. 1.0**

Team Members:

Ryan Ross

Cameron Kozan

Zaid Alsafi

Alex Navarre

**Project Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version Number | Date | Revising Author | Description of Revision |
| 1.0 | 2/12/2019 | Ryan Ross | Document Created |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **Table of Contents** |

[**1.0 Introduction**](#_83ik91ktkl1p) **7**

[1.1 Goals and Objectives](#_orgkxpxhw7vb) 7

[1.2 Statement of Scope](#_ungjh4hatje0) 7

[1.3 Software Context](#_rdgqjr5hrkau) 7

[1.4 Major Constraints](#_he46pyfvih) 7

[**2.0 Data Design**](#_6r77i7146eai) **7**

[2.1 Internal Software Data Structure](#_603luj9271tf) 7

[2.2 Global Data Structure](#_cb7o4hmg8for) 9

[**2.3 Temporary Data Structure**](#_o2vi9lrt3imc) **10**

[2.4 Database Description](#_fgr9jnoaqbk6) 10

[**3.0 Architectural and Component-Level Design**](#_mftvunn53dp6) **10**

[3.1 Program Structure](#_9zce3jy0f4eb) 10

[3.1.1 Architecture Diagram](#_u85flulxmj2h) 10

[3.1.2 Alternatives](#_kqsyzd3cyqnu) 10

[3.2a Description for component manage team](#_j5542z2ibruj) 11

[3.2a.1 Processing Narrative (PSPEC) for Component Team List](#_dcfwriw2u3kz) 11

[3.2a.2 Component Interface Description](#_c874w9bv03if) 11

[3.2a.3 Sub-Component Add Team Processing Detail](#_yirmvn5rn9xy) 11

[3.2a.3.1 Interface Description](#_bfean3cgf82) 11

[3.2a.3.2 Algorithmic Model (e.g., PDL)](#_6cxxirt9e6ur) 11

[3.2a.3.3 Restrictions/Limitations](#_2f3flwf0mtt0) 11

[3.2a.3.4 Local Data Structures](#_ki3dmjz0k4ea) 12

[3.2a.3.5 Performance Issues](#_49b98cndxv6) 12

[3.2a.3.6 Design Constraints](#_vu2p4e8tjhqf) 12

[3.2b Description for component team detail](#_bhn66dvdh0lh) 12

[3.2b.1 Processing Narrative (PSPEC) for Component Team Detail](#_bex9hldzparz) 12

[3.2b.2 Component Team Detail Interface Description](#_ghu57vj5hgg3) 12

[3.2b.3 Sub-Component Add Team Processing Detail](#_mppq0kneae8q) 12

[3.2.3.1 Interface Description](#_gyqod6k02ct5) 12

[3.2.3.2 Algorithmic Model (e.g., PDL)](#_fexhkerk10de) 13

[3.2.3.3 Restrictions/Limitations](#_hrmyecfy4y24) 13

[3.2.3.4 Local Data Structures](#_ps5fc0rhaoly) 13

[3.2.3.5 Performance Issues](#_vwgfu9ogf5hk) 13

[3.2.3.6 Design Constraints](#_9hs0je9wfs88) 13

[3.2 Description for component Manage Employees](#_1tb0m12mzs0a) 13

[3.2.1 Processing narrative (PSPEC) for component Manage Employees](#_wpr9zyh8bwox) 13

[3.2.2 Component Manage Employee Interface Description](#_pxqt1ijtkl8u) 14

[3.2.3 Sub-Component n.m Processing Detail](#_79a27f4e3dyb) 14

[3.2.3.1 Interface Description](#_q6ahbwx7y8wa) 14

[3.2.3.2 Algorithmic Model (e.g., PDL)](#_1mnpirmv5nye) 14

[3.2.3.3 Restrictions/Limitations](#_jgqrgol5lnqu) 14

[3.2.3.4 Local Data Structures](#_xokkzx35u2ey) 14

[3.2.3.5 Performance Issues](#_g6bnlnelcez1) 14

[3.2.3.6 Design Constraints](#_4euduxrkvdyu) 14

[3.2 Description for component Manage Teams](#_kniy5xy2768g) 15

[3.2.1 Processing narrative (PSPEC) for component Manage Teams](#_we036txhg98p) 15

[3.2.2 Component Manage Teams Interface Description](#_5w62cbougph) 15

[3.2.3 Sub-Component n.m Processing Detail](#_uiwqyu59t216) 15

[3.2.3.1 Interface Description](#_23m4igif9tah) 15

[3.2.3.2 Algorithmic Model (e.g., PDL)](#_fns8k65xrg8q) 15

[3.2.3.3 Restrictions/Limitations](#_jq6e0p189f9k) 15

[3.2.3.4 Local Data Structures](#_mf5fnug5xq7t) 15

[3.2.3.5 Performance Issues](#_ddnfvc1i5d8a) 15

[3.2.3.6 Design Constraints](#_c0n9g1oih5u0) 15

[3.2 Description for component Manage Assets](#_izm2g0h8tu2m) 16

[3.2.1 Processing narrative (PSPEC) for component Manage Assets](#_s3y3df8tn9z9) 16

[3.2.2 Component Manage Assets Interface Description](#_aknl78i1qzb1) 16

[3.2.3 Sub-Component n.m Processing Detail](#_r5jkwwjjjgg1) 16

[3.2.3.1 Interface Description](#_ds4schttvrw4) 16

[3.2.3.2 Algorithmic Model (e.g., PDL)](#_93kcpkfhijp1) 16

[3.2.3.3 Restrictions/Limitations](#_fjmx2ru3nnyj) 16

[3.2.3.4 Local Data Structures](#_nut032iizbju) 16

[3.2.3.5 Performance Issues](#_wdmb8qtt87zq) 16

[3.2.3.6 Design Constraints](#_ipnqqcknefig) 16

[3.2 Description for component Admin Dashboard](#_lymijief2xfs) 17

[3.2.1 Processing narrative (PSPEC) for component Admin Dashboard](#_fi48546m9liz) 17

[3.2.2 Component Admin Dashboard Interface Description](#_lo7pma9p7oge) 17

[3.2.3 Sub-Component n.m Processing Detail](#_h7igzxsn2qcr) 17

[3.2.3.1 Interface Description](#_izuanbr3ddx) 17

[3.2.3.2 Algorithmic Model (e.g., PDL)](#_hhj2x3iy5cji) 17

[3.2.3.3 Restrictions/Limitations](#_ozkzjgo8k8zy) 17

[3.2.3.4 Local Data Structures](#_zdvfijs7dz0m) 17

[3.2.3.5 Performance Issues](#_y2x2enp0gu52) 17

[3.2.3.6 Design Constraints](#_loqagjinwotu) 17

[3.2 Description for View Profile Component](#_cloq8v5q5btd) 17

[3.2.1 Processing Narrative (PSPEC) for Component View Profile](#_cvodmyyuy87l) 17

[3.2.2 View Profile Component Description](#_u7e2mtepu6at) 18

[3.2.3 Sub-Component n.m Processing Detail](#_89xvdv4qtkri) 18

[3.2.3.1 Interface Description](#_somvlok9poib) 18

[3.2.3.2 Algorithmic Model (e.g., PDL)](#_b5a7jbum7c4z) 18

[3.2.3.3 Restrictions/Limitations](#_mod7k4q4p0q6) 18

[3.2.3.4 Local data Structures](#_1y0twvdwvw3) 18

[3.2.3.5 Performance Issues](#_estajw8a3ol) 18

[3.2.3.6 Design Constraints](#_hj9vzzd14zja) 18

[3.2 Description for Edit Profile Component](#_c2idw64eug2g) 18

[3.2.1 Processing Narrative (PSPEC) for Component Edit Profile](#_gc2t8w6kaldr) 18

[3.2.2 Edit Profile Component Description](#_b678h3jqq70r) 19

[3.2.3 Sub-Component n.m Processing Detail](#_g5we8733edcm) 19

[3.2.3.1 Interface Description](#_2441j4cdz2re) 19

[3.2.3.2 Algorithmic Model (e.g., PDL)](#_e1qylaw8qejy) 19

[3.2.3.3 Restrictions/Limitations](#_botldg2p9ljp) 19

[3.2.3.4 Local data Structures](#_q1kzm9vxre7o) 19

[3.2.3.5 Performance Issues](#_ulf0vgo5mqge) 19

[3.2.3.6 Design Constraints](#_gz3ycl4824vk) 19

[3.2b Description for component Team dashboard](#_yd4s55v9875o) 19

[3.2b.1 Processing Narrative (PSPEC) for Component Team Dashboard](#_6nmzrolqhhop) 19

[3.2b.2 Component Team Detail Interface Description](#_l2bw5lzh0wvj) 19

[3.2b.3 Sub-Component Team Graph](#_vdt2qvyc7alt) 20

[3.2.3.1 Interface Description](#_xuo6dogpzye8) 20

[3.2.3.2 Algorithmic Model (e.g., PDL)](#_rk8qnbz2z0t5) 20

[3.2.3.3 Restrictions/Limitations](#_bk1victppexu) 20

[3.2.3.4 Local Data Structures](#_to7f99c7av9r) 20

[3.2.3.5 Performance Issues](#_b3xm67kf1yxl) 20

[3.2.3.6 Design Constraints](#_38jd3jzg4kbk) 20

[3.3 Software Interface Description](#_pvgd25hrmgjb) 20

[3.3.1 External Machine Interfaces](#_aam3cig9xiu) 20

[3.3.2 External System Interfaces](#_gzl3o45hgbp7) 20

[3.3.3 Human Interface](#_yrutt0xwtxq7) 21

[**4.0 User Interface Design**](#_a4gveaecrrq) **21**

[4.1 Description of the User Interface](#_2f47wrac8j3t) 21

[4.1.1 Screen Images](#_hp8gla5zke06) 21

[4.1.1.1 Login Screen](#_m2ldcnn5nhnw) 21

[4.1.1.2 Dashboard](#_rmqlhj6ub8i7) 21

[4.1.1.3 View Profile](#_6q4a6jthcntq) 22

[4.1.1.4 Edit Profile](#_c4k3lh93kno5) 23

[4.1.1.5 Manage Teams](#_xx281an7geir) 23

[4.1.1.6 Manage Employees](#_ntcq6697wqik) 24

[4.1.1.7 Manage Assets](#_t8sqr5fs7dyq) 24

[4.1.2 Objects and Actions](#_787mkxoshwqk) 24

[4.2 Interface Design rules](#_osel0aiy7lw5) 24

[4.3 Components Available](#_imm0qe5bz7wk) 25

[4.4 UIDS Descriptions](#_yn9nywxnaicr) 25

[**5.0 Restrictions, Limitations, and Constraints**](#_lt00q2d7ql4r) **25**

[**6.0 Testing Issues**](#_g1tisua7yo8o) **25**

[6.1 Classes of Tests](#_1ht1n2ew2p5r) 25

[6.2 Expected Software Response](#_ywkgzv1y0z3o) 27

[6.3 Performance Bounds](#_w55hqq7k0wn2) 28

[6.4 Identification of Critical Components](#_835rnvqs0hwd) 28

[**7.0 Appendices**](#_dhgezg977xeb) **28**

[7.1 Requirements Traceability Matrix](#_ti7zjpdp4ugf) 29

[7.2 Packaging and Installation Issues](#_xaok3i3llsi1) 29

[7.3 Design Metrics to be Used](#_ar35t5qeu0a6) 32

[7.4 Supplementary Information](#_jq9u8pnop7gz) 32

# 1.0 Introduction

## 1.1 Goals and Objectives

The goal of this project is to develop a gamified web application. The gamified web application will allow for the workers to be more motivated along with increasing productivity. The end goal of this project is for a user to go into the web application see how they rank among their peers, look at achievements, along with how all their teams are doing as well.

## 1.2 Statement of Scope

The Gamification software will use the client’s API, Invisitag, which will allow us to see all of the teams’ assets. From their api, each asset contains a cost and a weight, which we have incorporated in our website. Our website will allow admins to edit the cost and the weight of each asset. The employees will each have a score per every time they check in their assets. This score is comprised of the asset weight multiplied by if it’s stolen, lost or it’s their own asset. From there, the user will see a dashboard with all of the statistics, such as how they rank among their teammates, their score over time graph, along with many more metrics. The admins will assign employees to their respective teams, which then the employee’s performance will be based on the team performance.

## 1.3 Software Context

This software is going to be used by employees and managers of a company. Employees will be grouped into teams, where the employee’s score will be an aggregate score of their teams. The managers and employees will be able to visually see their progress and how they can improve their performance.

## 1.4 Major Constraints

* Web application front end, back end and service must all run on EC2
* Database must be hosted on AWS RDS
* Must be done by April 2020

# 2.0 Data Design

## 2.1 Internal Software Data Structure

* Data structures that are passed among components the software are described.
  + **User**
  + The user table consists of the following attributes.
  + There will be two levels of security – User (Employee) and Admin (Owner/Manager).
    - **Id** (Primary Key) – The Id representing the user.
    - **CompanyId** (Foreign Key) – The Id representing the company the user works for.
    - **FirstName** – First name of the user.
    - **LastName** – Last name of the user.
    - **Email** – Work email for the user. This will be used as the username at login.
    - **Password** – Password for the user to login. This will be encrypted.
    - **Role** - Represents the user’s access role
    - **EXP** - How much EXP the user has earned, total
    - **Rank** - The current users’ ranking in the company
    - **StartDate** - Date that the user started working at the company
  + **Team**
  + The team table consists of the following attributes.
    - **Id** (Primary Id) – The Id representing the Team working on a task.
    - **CompanyId** (Foreign Key) – The Id representing the company the team works for.
    - **Name** – The name of the team.
  + **Rank**
  + The ranking table consists of the following attributes.
    - **Id** (Primary Key) – The Id representing the rank of the team.
    - **AssetId** (Foreign Key) – The Id representing the asset that a team has.
    - **TeamId** (Foreign Key) – The Id representing the team the rank belongs to.
    - **Week** - The week that the asset was scanned
    - **Detected** - A boolean if the asset has been detected or not
    - **ExtraTag** - A boolean if the asset has been stolen or not
  + **Assets**
  + The assets table consists of the following attributes.
    - **Id** (Primary Key) – The Id representing the asset.
    - **Name** – The name of the asset.
    - **Weight** – The weight of the asset. This is what is used when calculating the actual score for a teams rank.
    - **Cost** – The cost of the asset. If no weight is specified by the admin, the cost by default will be used when calculating the actual score for a teams rank.
    - **RFID** - The RFID tag associated with the asset
  + **Badges**
  + The badge table consists of the following attributes.
    - **Id** (Primary Key) – The Id representing the badge.
    - **Name** - The name of the badge
    - **Icon** - A link to the icon image
  + **Challenges**
  + The challenges table consists of the following attributes
    - **Id** (Primary Key) – The Id representing the challenge.
    - **Name** - The name of the challenge
    - **Description** - Description of the challenge and how to get it.
    - **AchievedTime** - Time the challenge was completed
    - **Reward** - Reward for completing the challenge
  + **Newsfeed**
  + The newsfeed table consists of the following attributes
    - **Id** (Primary Key) – The Id representing the news feed activity.
    - **EmployeeId** - The EmployeeId who completed the activity.
    - **Description** - Description of the activity and how to get it.
    - **AchievedTime** - Time the activity was completed
    - **Reward** - Reward for completing the challenge

## 2.2 Global Data Structure

* Data structured that are available to major portions of the architecture are described.
  + **Shard Manager**
  + The shard manager will be the only global data structure used.
    - **Company**
    - The company table consists of the following attributes.
      * **Id** (Primary Key) – The Id representing the company that uses the Invisi-tag software.
      * **ShardId** (Foreign Key) – The shard name that belongs to the company the user works for.
      * **Name** – Name of the company that uses the Invisi-tag software.
    - **Shard**
    - The shard consists of the following attributes.
      * **Id** (Primary Key) – The Id representing the shard for a specific company.
      * **Name** – The name of the shard.
  + **UserLogin**
  + The UserLogin table consists of the following attributes.
  + This table consists of the same attributes as User table, however this table will consist of all users for all companies.
    - **Id** (Primary Key) – The Id representing the user.
    - **CompanyId** (Foreign Key) – The Id representing the company the user works for.
    - **FirstName** – First name of the user.
    - **LastName** – Last name of the user.
    - **Email** – Work email for the user. This will be used as the username at login.
    - **Password** – Password for the user to login. This will be encrypted.
    - **Role** - Represents the user’s access role
    - **EXP** - How much EXP the user has earned, total
    - **Rank** - The current users’ ranking in the company
    - **StartDate** - Date that the user started working at the company

# 2.3 Temporary Data Structure

The only temporary data structures that are used are for session management. Sessions are managed with JWT tokens. They are created once someone logs in and deactivated once the back end shuts off.

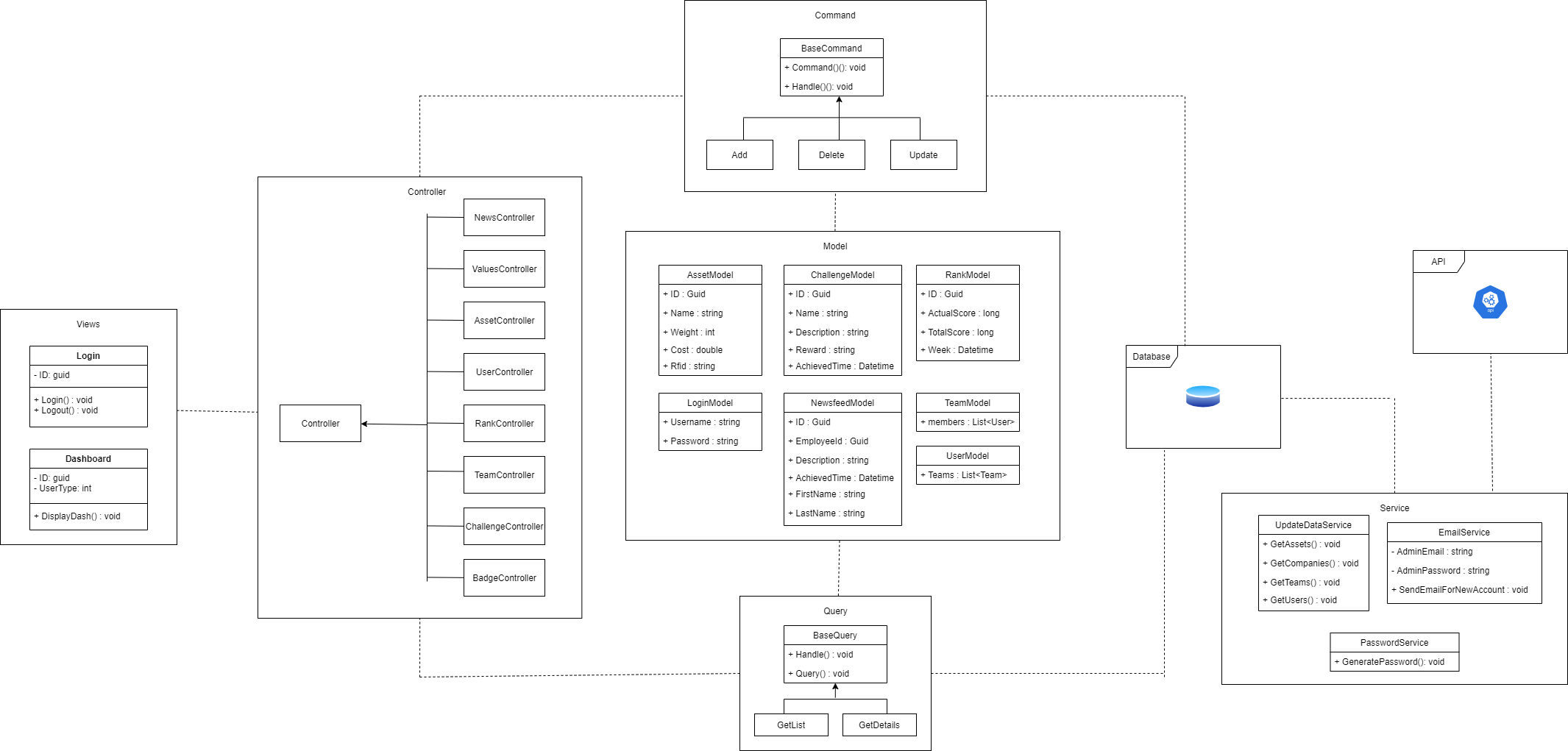
## 2.4 Database Description

We have a few different databases. All of our databases are hosted in Amazon’s RDS. There is one main database which manages all of the others called ShardManager. ShardManager contains all of the shards, which are individual databases. These individual databases contain information about one specific company.

# 3.0 Architectural and Component-Level Design

## 3.1 Program Structure

### 3.1.1 Architecture Diagram



### 3.1.2 Alternatives

○ Reason for architecture in 3.1.1:

■ We are using the MVC architecture design pattern. The website will act as the view, the view will connect to the controller when retrieving or sending information to the database. The controllers are also acting as a facade they will connect to the classes that will interact with the database, the database operations are broken up into 2 categories: Commands and queries this is following the CQRS(command and query responsibility segregation) the commands will update the database and the queries will retrieve information. Both commands and queries will use the models

■ The service application will run on a time-based trigger. Rank service will pull new reports from the client Api and populate the ranking records in the database.

■ UpdateDataService will focus on pulling new company, employee, team and asset data from the client Api.

The project is a continuation for another senior design group as such, we were given the backend setup with this architecture and built on it.

## 3.2a Description for component manage team

### 3.2a.1 Processing Narrative (PSPEC) for Component Team List

Users clicks manage teams, a get request will be sent to the team controller, for the get teams api. The mediator will the invoke the query GetTeams and retrieve the team data from the database.

### 3.2a.2 Component Interface Description

* A table displays all the team names and the teams active status shows beside it.
* A search bar will be on top of the table that filters the team based on what the user searches
* A button with the label Add will take the user to the add team sub component

### 3.2a.3 Sub-Component Add Team Processing Detail

#### 3.2a.3.1 Interface Description

A list of teams from the clients api are displayed. The teams are not in the gamification in the database at this point. There is an add button near

#### 3.2a.3.2 Algorithmic Model (e.g., PDL)

* Get teams from api
* Check teams retrieved
* If team exist in database
  + Remove from list
* Return teams to front end
* If admin clicks add
  + Add team to list
* If admin clicks save
  + send teams to the add team api
* For each team in list
  + Insert team into database

#### 3.2a.3.3 Restrictions/Limitations

* The team list page must work and allow the user to click the add button to open this sub component
* The front end service that make api calls
* The team controller must be working properly
* The client api must allow requests to be made and results to return

#### 3.2a.3.4 Local Data Structures

* Team
  + Id: guid
  + Name: string
  + Is active
* User
  + FirstName: string
  + LastName: string
  + Email: string
  + Image: binary
* Front end
  + Array of team objects

#### 3.2a.3.5 Performance Issues

Retrieving teams from the clients api - making and waiting for a response may cause a delay.

#### 3.2a.3.6 Design Constraints

When requesting teams from the clients api a list of teams are retrieved, then the teams that are in the database are removed. This is a constraint because all the teams are sent with each request, even if there is only one team in the database.

## 3.2b Description for component team detail

### 3.2b.1 Processing Narrative (PSPEC) for Component Team Detail

Users selects a team from the team list, a get request will be sent to the team controller, for the get team api with the team id. The mediator will invoke the query ‘GetTeam’ and retrieve the team data from the database.

### 3.2b.2 Component Team Detail Interface Description

* A list of team members with a remove button to the right
* A search bar that allows the admin to filter the team
* The team name and the active status are on

### 3.2b.3 Sub-Component Add Team Processing Detail

#### 3.2.3.1 Interface Description

A list of employees in the database are displayed. The list does not contain any members from the team.

#### 3.2.3.2 Algorithmic Model (e.g., PDL)

* Admin selects employees to add to team
* Admin saves
* List is added to team
* Team is sent to the update team api
* The mediator invokes the command update team

#### 3.2.3.3 Restrictions/Limitations

* The team detail page must work and allow the user to click the add button to open this sub cpm
* The front end service that makes the api call
* The team controller must be working properly

#### 3.2.3.4 Local Data Structures

* Team
  + ID: guid
  + Name: string
  + Is Active: boolean
* User
  + FristName: string
  + LastName: string
  + Email: string
  + Profile Picture: base 64
  + Start Date: datetime
* Frontend
  + Array of Users

#### 3.2.3.5 Performance Issues

* Loading time for larger pictures is not ideal

#### 3.2.3.6 Design Constraints

* A user must save before they opening up dialog
* If the request fails, the user is then skipped and the error message is thrown on completion

## 3.2 Description for component Manage Employees

### 3.2.1 Processing narrative (PSPEC) for component Manage Employees

* The user selects ‘Manage Employees.’ A request is sent to the employee controller to retrieve the list of all employees and is listed as a table view. The mediator then invokes the getEmployee query to retrieve the information from the database.

### 3.2.2 Component Manage Employee Interface Description

* On login, an Admin has menu access to the ‘Manage Employee’ option on the navigation bar. A list of employees is displayed in table format. A menu will be displayed with the options to add, edit, or delete an employee.

### 3.2.3 Sub-Component n.m Processing Detail

#### 3.2.3.1 Interface Description

* ‘Add Employee’ is selected by the Admin user. The Admin will then enter all the information regarding the new user: First & Last Name, Phone Number, Email, and Start Date. On completion, a request command will be sent and the user will be inserted into the database.

#### 3.2.3.2 Algorithmic Model (e.g., PDL)

* Admin enters employee information
* Admin submits employee information upon completion
* The employee’s password is randomly generated and then emailed to the listed email
* Finally, the mediator invokes the command to insert the employee into the database

#### 3.2.3.3 Restrictions/Limitations

* The user must have an internet connection
* The connection to the database must be established
* The ‘Manage Employee’ page must be loaded for the admin to access

#### 3.2.3.4 Local Data Structures

* FirstName: string
* LastName: string
* Email: string
* StartDate: Datetime
* ProfilePicture: base 64

#### 3.2.3.5 Performance Issues

* Load time of the data, could cause issues if the list is too big

#### 3.2.3.6 Design Constraints

* The user must click submit for the employee to be added to the database. There is no auto-update feature.
* The user will encounter an error message if there is invalid data entered into their respective fields.

## 3.2 Description for component Manage Teams

### 3.2.1 Processing narrative (PSPEC) for component Manage Teams

* The user selects ‘Manage Teams.’ A request is sent to the team controller to retrieve the list of all teams and is listed as a table view. The mediator then invokes the getTeams query to retrieve the information from the database.

### 3.2.2 Component Manage Teams Interface Description

* On login, an Admin has menu access to the ‘Manage Teams’ option on the navigation bar. A list of teams is displayed in table format. A menu will be displayed with the options to add, edit, or delete teams .

### 3.2.3 Sub-Component n.m Processing Detail

#### 3.2.3.1 Interface Description

* ‘Add Team’ is selected by the Admin user. The Admin will get an option based on teams from the client’s API. On completion, a request command will be sent and the team will be inserted into the database.

#### 3.2.3.2 Algorithmic Model (e.g., PDL)

* Admin hits add new team
* Admin adds a user to the team
* Admin submits the request, which inserts it into the database

#### 3.2.3.3 Restrictions/Limitations

* The user must have an internet connection
* The connection to the database must be established
* The ‘Manage Team’ page must be loaded for the admin to access

#### 3.2.3.4 Local Data Structures

TeamDto

#### 3.2.3.5 Performance Issues

* Load time of the data, could cause issues if the list is too big

#### 3.2.3.6 Design Constraints

* The user must click submit for the team to be added to the database. There is no auto-update feature.

## 3.2 Description for component Manage Assets

### 3.2.1 Processing narrative (PSPEC) for component Manage Assets

* The user selects ‘Manage Assets.’ A request is sent to the asset controller to retrieve the list of all assets and is listed as a table view. The mediator then invokes the getAssets query to retrieve the information from the database.

### 3.2.2 Component Manage Assets Interface Description

* On login, an Admin has menu access to the ‘Manage Assets’ option on the navigation bar. A list of assets is displayed in table format. A menu will be displayed with the options to add, edit, or delete an asset. Along with a dropdown menu from the Client’s API which shows all of the assets of a company.

### 3.2.3 Sub-Component n.m Processing Detail

#### 3.2.3.1 Interface Description

* ‘Add Employee’ is selected by the Admin user. The Admin will then enter all the information regarding the new user: First & Last Name, Phone Number, Email, and Start Date. On completion, a request command will be sent and the user will be inserted into the database.

#### 3.2.3.2 Algorithmic Model (e.g., PDL)

* Admin selects a number of assets from the dropdown
* Admin hits “Save”
* Assets selected are now in the database.

#### 3.2.3.3 Restrictions/Limitations

* The user must have an internet connection
* The connection to the database must be established
* The ‘Manage Assets’ page must be loaded for the admin to access

#### 3.2.3.4 Local Data Structures

AssetDto

#### 3.2.3.5 Performance Issues

* Load time of the data, could cause issues if the list is too big

#### 3.2.3.6 Design Constraints

* The user must click submit for the asset to be added to the database. There is no auto-update feature.

## 3.2 Description for component Admin Dashboard

### 3.2.1 Processing narrative (PSPEC) for component Admin Dashboard

* The admin selects the ‘Home’ page. This is also the default view once an admin logs in.

### 3.2.2 Component Admin Dashboard Interface Description

* There should be a graph showing the team’s performance over time, to make sure they are doing positively over time. This graph will be the #1 ranking team by default.
* There will be a leaderboard and how the team is placed on the leaderboard. The #1 team will be selected by default again.
* There will be a challenges table below the graphs and leaderboard, where the admin can set the challenges that a team has.

### 3.2.3 Sub-Component n.m Processing Detail

#### 3.2.3.1 Interface Description

A graph showing the teams performance over time will be shown, the default being the rank #1 team, along with their ranking among their peers. Along with that, it will show a leaderboard and all of the teams and their rankings. There will also be a table of challenges for each individual team, where the admin can set challenges.

#### 3.2.3.2 Algorithmic Model (e.g., PDL)

* The front send service calls the API for the first team, which after you click on a team,, it’ll make subsequent API calls.
* Like the first note, the challenges are handled in the same fashion.

#### 3.2.3.3 Restrictions/Limitations

* All the teams must have data

#### 3.2.3.4 Local Data Structures

RankDto, TeamDto

#### 3.2.3.5 Performance Issues

* API calls may take longer than expected

#### 3.2.3.6 Design Constraints

* The dashboards must show gamification of some sort.

## 3.2 Description for View Profile Component

### 3.2.1 Processing Narrative (PSPEC) for Component View Profile

This page will be able to be navigated to from the navigation bar. The label will be “View profile”

### 3.2.2 View Profile Component Description

This is the page that the user will be able to see all of their statistics. This page will have their level, rank, the current challenges they have, along with all of the badges they’ve earned and their recent activity. Their recent activity will be comprised of if they’ve gained any ranks, if they’ve leveled up, if they have gained any badges, or completed any challenges.

### 3.2.3 Sub-Component n.m Processing Detail

#### 3.2.3.1 Interface Description

At the top left of the page, there will be the users’ profile picture, rank, level and EXP to the next level along with a progress bar to visually see how far they are from the next level. Below that the user should see all of the challenges that they are working towards. Once a user completes a challenge, the challenge will disappear from the challenge table. Below that will be an aggregate of their lost tools, stolen tools and how much they’ve cost the company.

#### 3.2.3.2 Algorithmic Model (e.g., PDL)

Pull all of the items that the team has gotten. Multiply that item by the weight. Add all of the asset \* weight calculations together, then that is that week's score.

#### 3.2.3.3 Restrictions/Limitations

The backend and the database must be running in order for this page to work.

#### 3.2.3.4 Local data Structures

RankDto, ChallengeDto, NewsfeedDto, UserDto

#### 3.2.3.5 Performance Issues

There will be a few performance issues on this page .There are 4 API calls to the back end. The User API call, which gets all of the information about the user, such as level, rank, name and exp. The second API call is to get all of the challenges for the user. The third is to get all of the badges associated with the user, and the last one is to get the top 10 recent activities for the user.

The performance issue with this page is that there are a few ajax calls. If one doesn’t come back, the page will be sitting at the loading screen until it comes back.

#### 3.2.3.6 Design Constraints

When designing our UI we don't know our general customer and this affects us because we don't know what kind of styling we should approach the UI and because of that we have to give our best guess about a design. Another Constraint we have to handle is how would the UI present the data and is it possible to create an easy presentation of the data.

## 3.2 Description for Edit Profile Component

### 3.2.1 Processing Narrative (PSPEC) for Component Edit Profile

This page will be able to be navigated to from the View profile page. Once a user is at their own View profile page, if they click on “Manage account” this page will show up.

### 3.2.2 Edit Profile Component Description

This is the page that the user will be able to edit their name, email and phone number along with their profile picture.

### 3.2.3 Sub-Component n.m Processing Detail

#### 3.2.3.1 Interface Description

In the top left there will be the First Name, Top Right will be Last name, Below first name is going to be the phone number and below the last name will be their email account. Below the phone number will be the disabled start date field. Then there will be a submit and cancel button below all of that, and near the buttons will be an upload image button for the user to upload a new profile picture.

#### 3.2.3.2 Algorithmic Model (e.g., PDL)

Uploading the new profile to the database

#### 3.2.3.3 Restrictions/Limitations

The backend and the database must be running in order for this page to work.

#### 3.2.3.4 Local data Structures

UserDto

#### 3.2.3.5 Performance Issues

The only performance issue that this page could face is the API call failing when attempting to save the profile.

#### 3.2.3.6 Design Constraints

The design constraints for this were all of the fields in the User table in the database.

## 3.2b Description for component Team dashboard

### 3.2b.1 Processing Narrative (PSPEC) for Component Team Dashboard

This page will be accessed from the nav bar under ‘Team Dashboard’

### 3.2b.2 Component Team Detail Interface Description

* There should be a graph showing the team’s performance over time, to make sure they are doing positively over time.
* There will be a list of employees that are on the team.
* There will be a ranking of the team on the page
* There will be a leaderboard and how the team is placed on the leaderboard.
* There will be challenges for the team on the team view page.

### 3.2b.3 Sub-Component Team Graph

#### 3.2.3.1 Interface Description

A graph showing the teams performance over time will be shown, along with their ranking among their peers. Along with that, it will show the users on the team along with the challenges for the team.

#### 3.2.3.2 Algorithmic Model (e.g., PDL)

* Admin selects employees to add to team
* Admin saves
* List is added to team
* Team is sent to the update team api
* The mediator invokes the command update team

#### 3.2.3.3 Restrictions/Limitations

* The front end service that makes the api call
* The team controller must be working properly

#### 3.2.3.4 Local Data Structures

* Rank
  + Id: GUID
  + ActualScore : long
  + TotalScore : long
  + Week : datetime

#### 3.2.3.5 Performance Issues

* API calls may take longer than expected

#### 3.2.3.6 Design Constraints

* We will show the top 3 teams along with where the current team is placed.
* If the request fails, the user is then skipped and the error message is thrown on completion

## 3.3 Software Interface Description

### 3.3.1 External Machine Interfaces

The external machines the application will interface with are any devices connected to the web. Our main target of the web page are people with PC’s rather than mobile devices.

### 3.3.2 External System Interfaces

There are a few external systems that this application interfaces with. The first one is Amazon RDS, where the database is hosted. Secondly, the web application is hosted on Amazon EC2.

The web application checks the Client’s API daily to see if there are any more jobs that have been done for that day, and then adds them to the database.

The application also communicates with a SMTP server, which is used when an employee resets’ there passwords or an admin creates an account. The SMTP server emails the said user their new email and password.

### 3.3.3 Human Interface

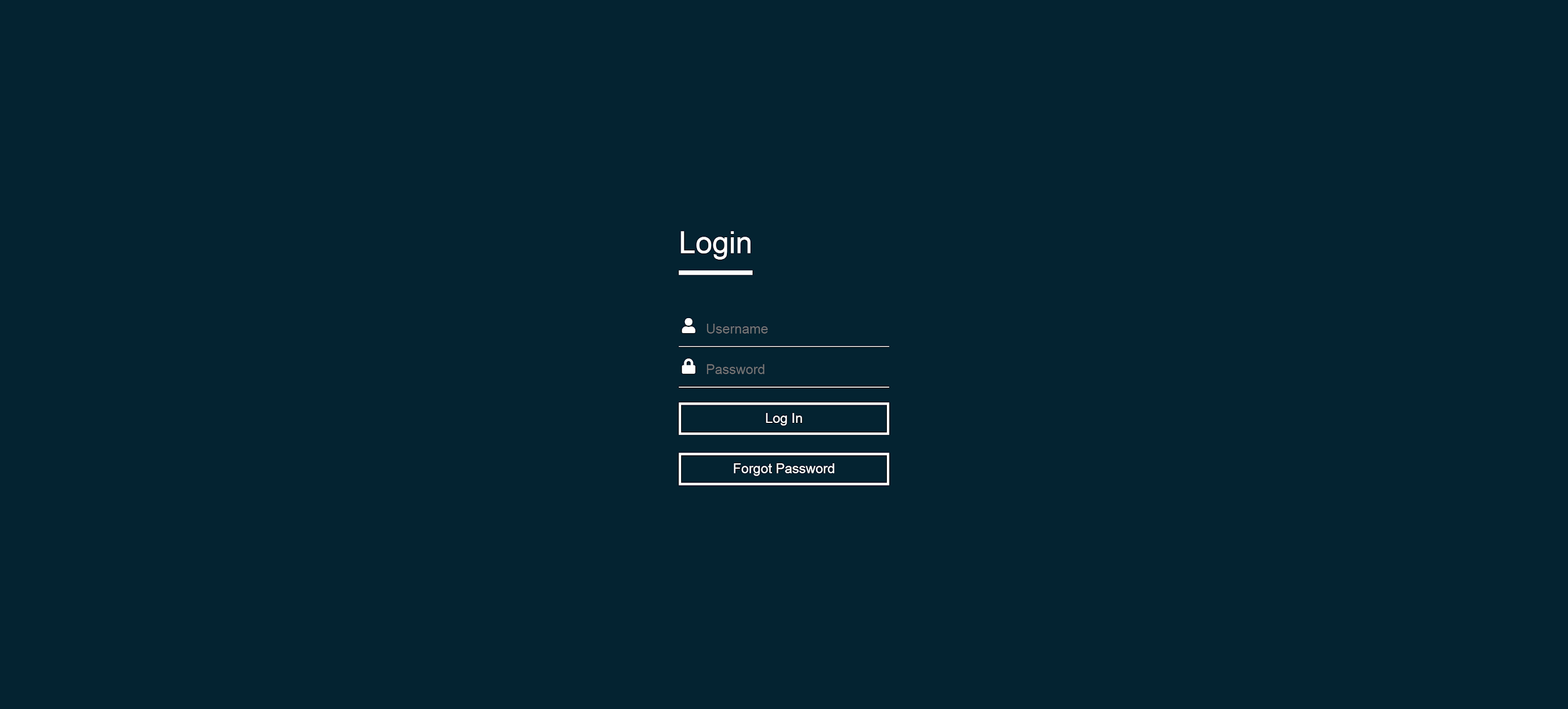
* An overview of all the screens can be found in Section 4.
* The web application is created using AngularJS, Typescript along with Javascript, HTML and CSS. The web application will have a gamified look and feel to it, so the employees will feel motivated and promote competition among employees.

# 4.0 User Interface Design

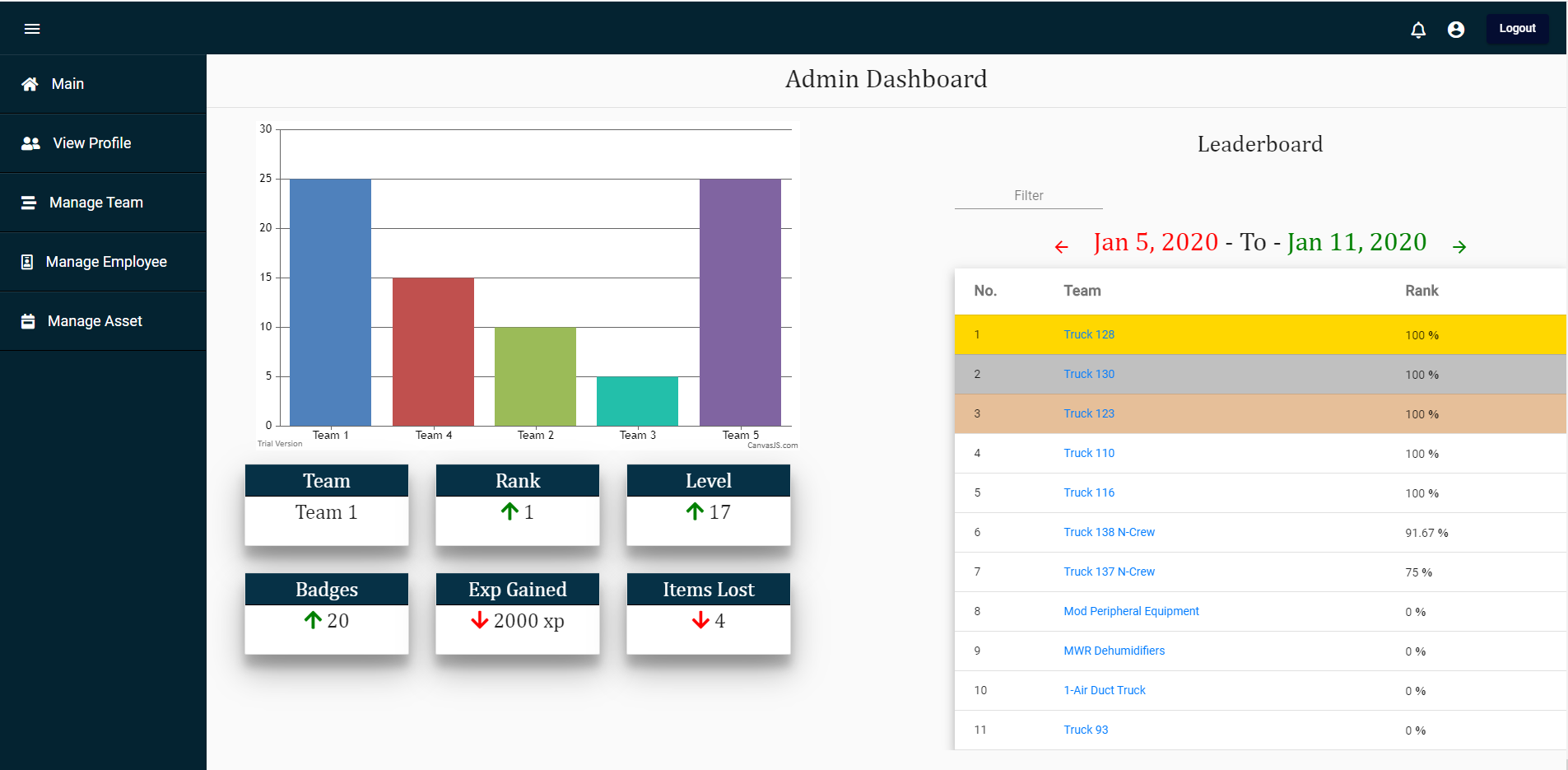
## 4.1 Description of the User Interface

### 4.1.1 Screen Images

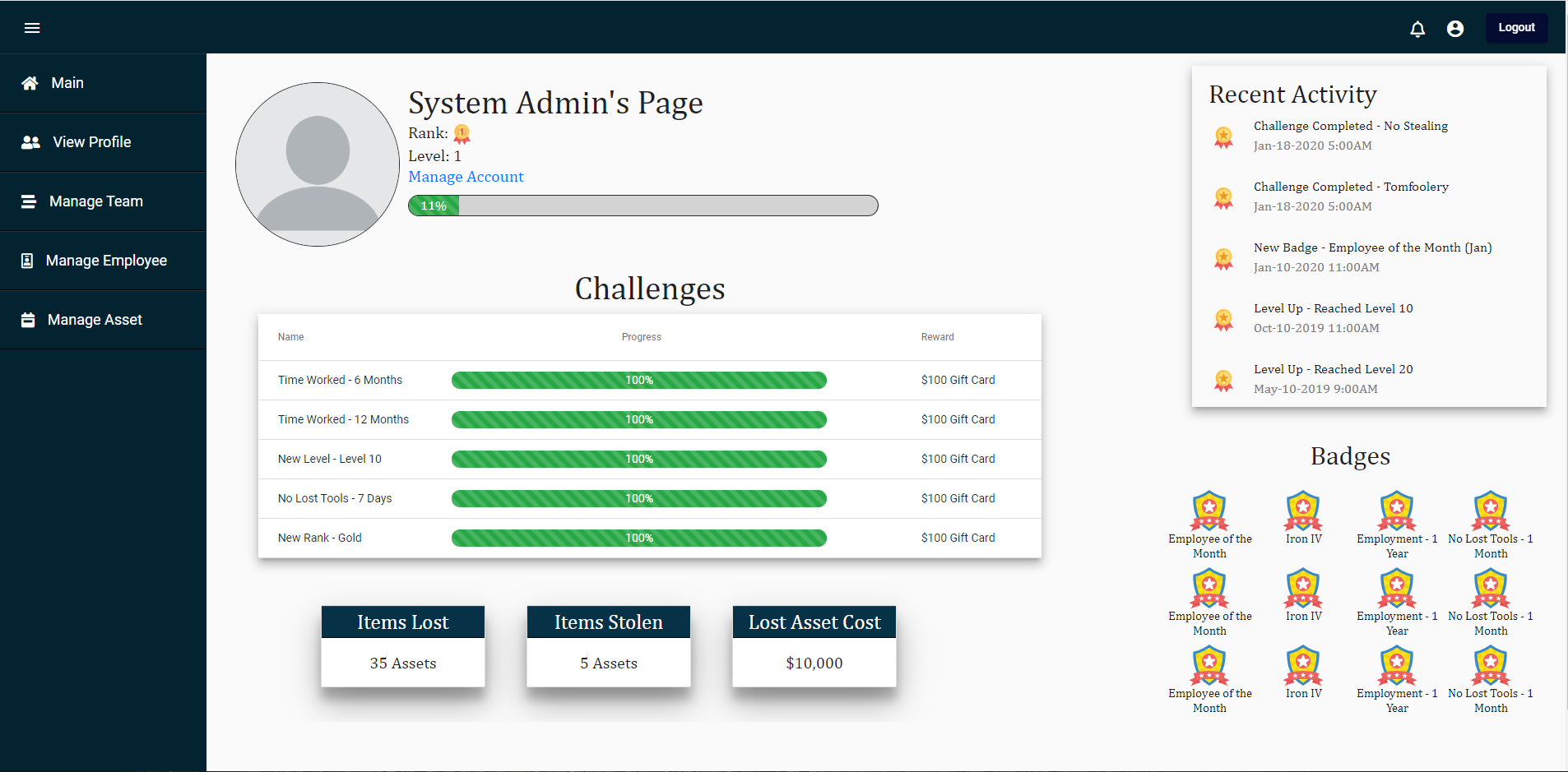
### 4.1.1.1 Login Screen



### 4.1.1.2 Dashboard

****

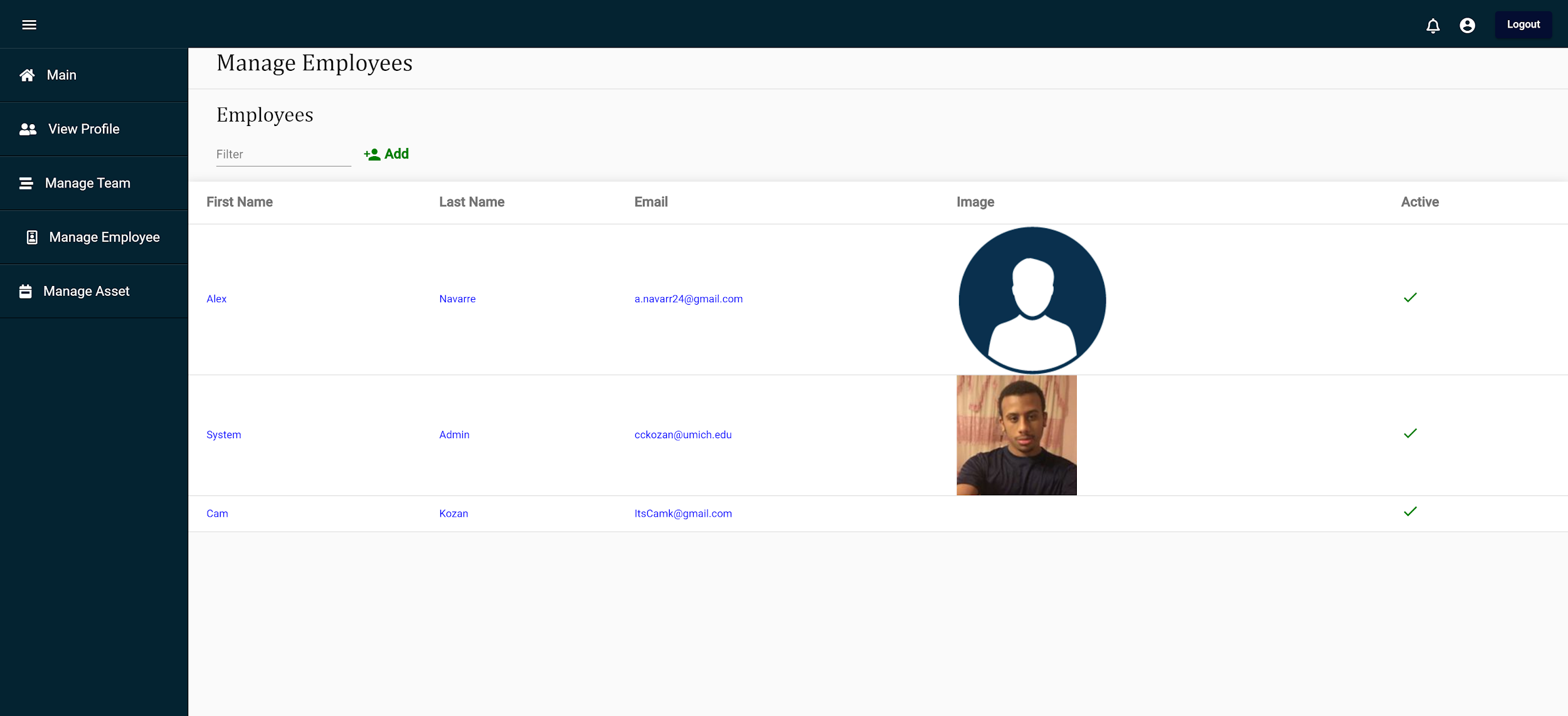
### 4.1.1.3 View Profile



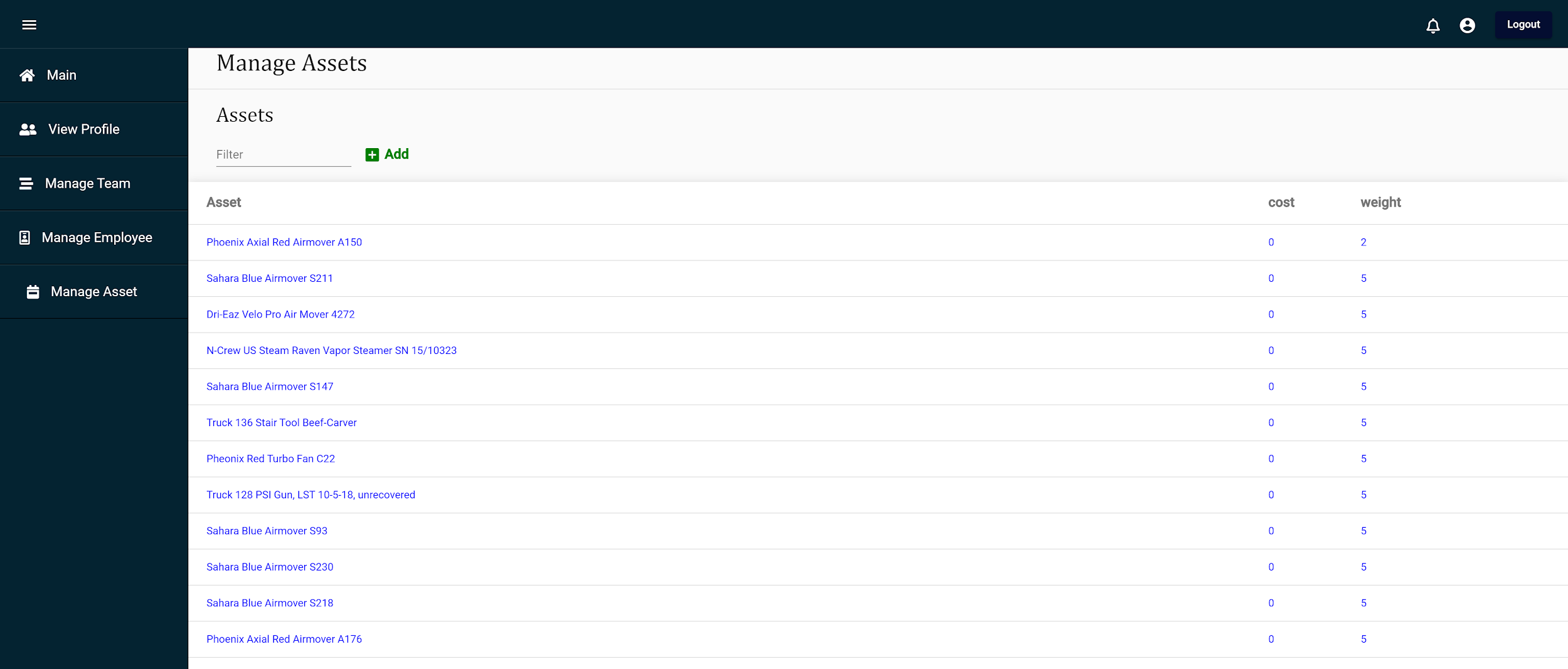
### 4.1.1.4 Edit Profile

### 4.1.1.5 Manage Teams

### 4.1.1.6 Manage Employees



### 4.1.1.7 Manage Assets



### 4.1.2 Objects and Actions

## 4.2 Interface Design rules

Consistency: Our interface shouldn't make the user wonder whether different words, situations, or actions mean the same thing. We want to keep the process of each page the same and feel consistent. We also want the terminology and appearances to a level that is simple and clean but still consistent to each page.

## 4.3 Components Available

We are using Angular.Js as a framework and importing other libraries to help us complete many of the functionality of the website

* Chart.js: Is a library containing many different varieties of charts that are minulatibable and can dynamically change. We are using many different kinds of charts and graphs to display data in a more digestible way.
* Bootstrap: bootstrap is used to provide the graphs and the charts for the website as well as some additional front-end design components.
* Angular Material: We use Angular Material to get our basic core design of the page. This allows us to create a navigation bar, top bar, and many more other things.

## 4.4 UIDS Descriptions

As stated above, we are using AngularJS and Typescript for the majority of the front end. For the charts and graphs, we are using some of the built in graphs and charts from Angular. Along with that we are using ng-charts-2 which allows for better looking diagrams. We are also using ChartJS, which is an additional 3rd party diagram library.

# 5.0 Restrictions, Limitations, and Constraints

Restrictions:

The software will need to run on newer web browsers so we are unable to to use specific 3rd party add on features for Angular. This will limit our capabilities when it comes to a more module approach when designing the webpage. We found it most important to keep the webpage running as long as possible in trade off for not using the most current Angular libraries.

Limitations:

A majority of our data comes from the companies already established API. The Invisitag API only allows us to retrieve job start dates and times so we had to manually calculate if a job has been completed. The jobReport/tags (assets used on a job) API calls do not have the tag UUID, instead it has an RFID tag.

Challenges, badges, experience points, and levels are all calculated from an individual's score only. This limits our ability to create extensive and interactive and elaborate challenges. The amount of challenges has been limited due to the small amount of data we have received.

Constraints:

The database requires sharding and each shard should only contain one company. The sharding technique allows us to use a single database for multiple companies. This saves space and money while allowing each company to access their own data. The shard manager should direct users and the system to the correct database shard on login.

The program must pull data from the Invisitag API once a day.

# 6.0 Testing Issues

## 6.1 Classes of Tests

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Type | Description | Input |
| 1 | Black Box Testing | Testing Login | Username: [cckozan@umich.edu](mailto:cckozan@umich.edu)  Password: Yeet123 |
| 2 | Black Box Testing | Testing if an admin can create a team | Under Manage Teams -> Add team |
| 3 | Black Box Testing | Testing if an admin can delete a team | Under Manage Teams -> Delete team |
| 4 | Black Box Testing | Testing if an admin can add employees to a team | Under Manage Teams -> Add Employee |
| 5 | Black Box Testing | Testing if an admin can remove employees to a team | Under Manage Teams -> Remove Employee |
| 6 | Black Box Testing | Testing if admin can create  employee account. | Under Manage Employees -> Add employee |
| 7 | Black Box Testing | Testing if admin can delete  employee account. | Under Manage Employees -> Delete employee |
| 8 | Black Box Testing | Testing to see if a user gets a badge once they complete a challenge | Give a user data to complete his challenge |
| 9 | Black Box Testing | Testing to see if a user levels up once they hit the required XP to level up | Give a user enough XP to level up |
| 10 | Black Box Testing | Testing to see if a user can see their teams dashboard | Navigate to the Team Dashboard page |
| 11 | Black Box Testing | Testing to see if a user can see their own stats | Navigate to the view-profile page |
| 12 | Black Box Testing | Testing to see if an admin can see all the teams dashboards | Navigate to the admin dashboard and click on each individual team |
| 13 | Black Box Testing | Testing to see if another user can see if the can see other user’s profiles | Navigate to the dashboard, click on another team on the leaderboard |
| 14 | Black Box Testing | Testing to see if a user can reset their password | At the login screen, hit reset password |
| 15 | Black Box Testing | Upload an image | Go to the view-profile page -> manage profile -> upload image -> submit |
| 16 | Black Box Testing | Change a user’s name | Go to the view-profile page -> manage profile -> change their name-> submit |
| 17 | Black Box Testing | Move a user from team to team and see if his score changes | Check the user’s profile, then remove them from team A and add them to team B and see if the score has changed. |
| 18 | Black Box Testing | Add a team to a user and see if his score changes | Check the user’s profile, add them to another team and see if the score has changed. |
| 19 | Black Box Testing | Remove a user from a single user team | Check the user’s profile, then remove them from team A and go back to the profile page and see if their score has changed. |
| 20 | Black Box Testing | Checking to make sure the styling doesnt completely mess up once the width changes | Change the dimensions of the screen and see the changes |

## 6.2 Expected Software Response

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Expected Output | Actual Output | Pass/Fail | Date Fixed (or N/A) |
| 1 | User should be logged in |  |  |  |
| 2 | Admin should have created a team |  |  |  |
| 3 | Admin should have deleted a team |  |  |  |
| 4 | Admin should have added an employee to a team |  |  |  |
| 5 | Admin should have removed an employee to a team |  |  |  |
| 6 | Admin should have created an employee account |  |  |  |
| 7 | Admin should have deleted an employee account |  |  |  |
| 8 | The user should now have the badge for the challenge they completed |  |  |  |
| 9 | The user should now be one level higher |  |  |  |
| 10 | The user should be shown their own teams dashboard |  |  |  |
| 11 | The user should be shown their own profile (Statistics) page |  |  |  |
| 12 | The admin should be able to access all of the team’s dashboards |  |  |  |
| 13 | The user should be able to see their peer’s profile |  |  |  |
| 14 | A user should be able to reset their password |  |  |  |
| 15 | The user should be able to see their newly updated profile picture |  |  |  |
| 16 | The user’s name should be changed |  |  |  |
| 17 | The users score should reflect the new team’s score, rather than the old |  |  |  |
| 18 | The users score should reflect the old team + the newly added teams score |  |  |  |
| 19 | The users score should reflect the single team they’re on |  |  |  |
| 20 | The styling should remain good looking under compression |  |  |  |

## 6.3 Performance Bounds

The web application should perform seamlessly on the EC2 servers.

If there is an error updating a user’s job one day, the score should not reflect the errors.

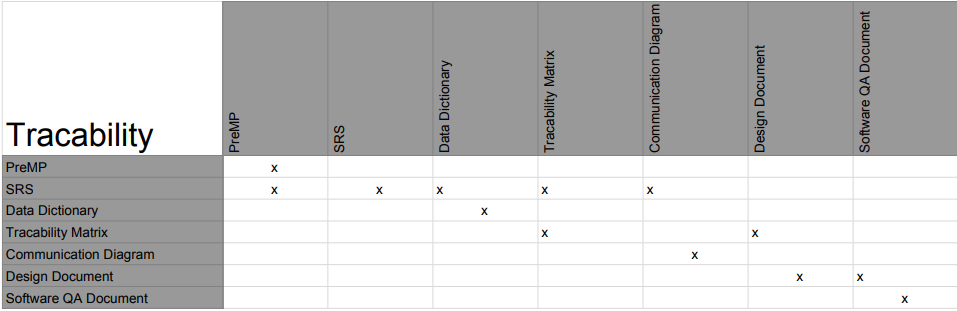
If the call fails for resetting a password, a static password will be in place.

## 6.4 Identification of Critical Components

1. The service should be tested thoroughly both with and without the client’s API.
2. The dashboards designs are critical to the project.
3. Gamification characteristics must be implemented.

# 7.0 Appendices

## 7.1 Requirements Traceability Matrix

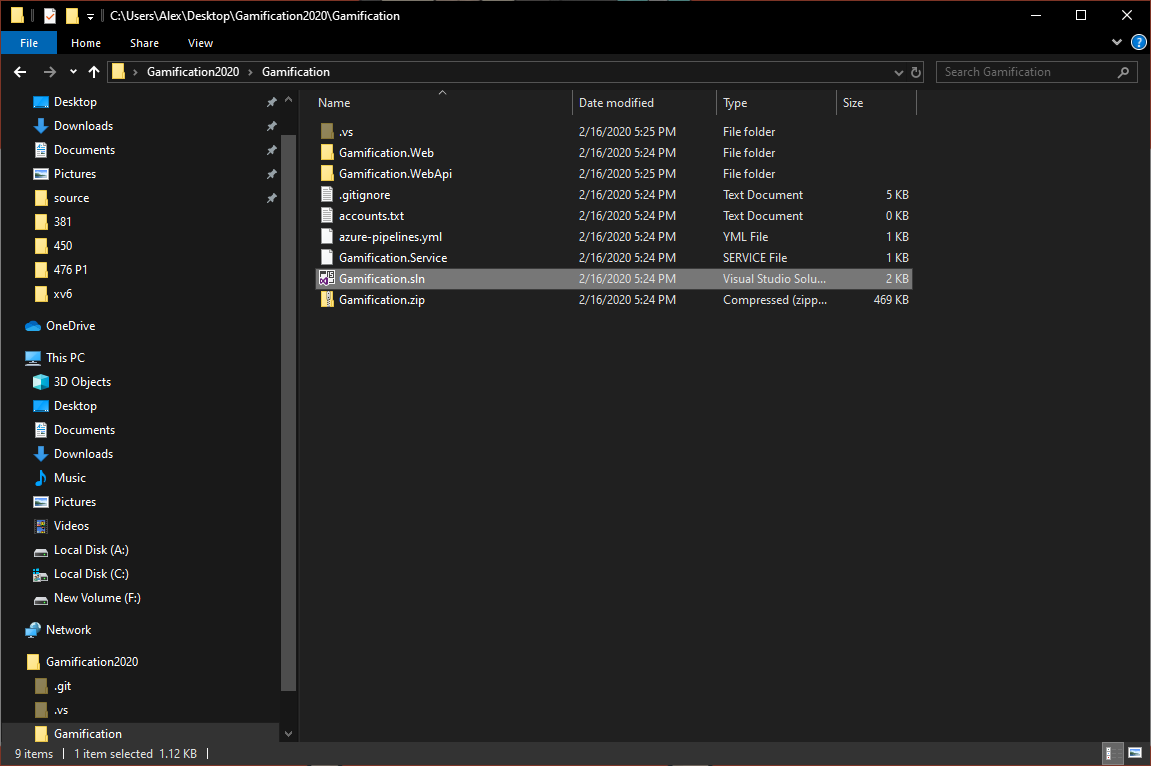


## 7.2 Packaging and Installation Issues

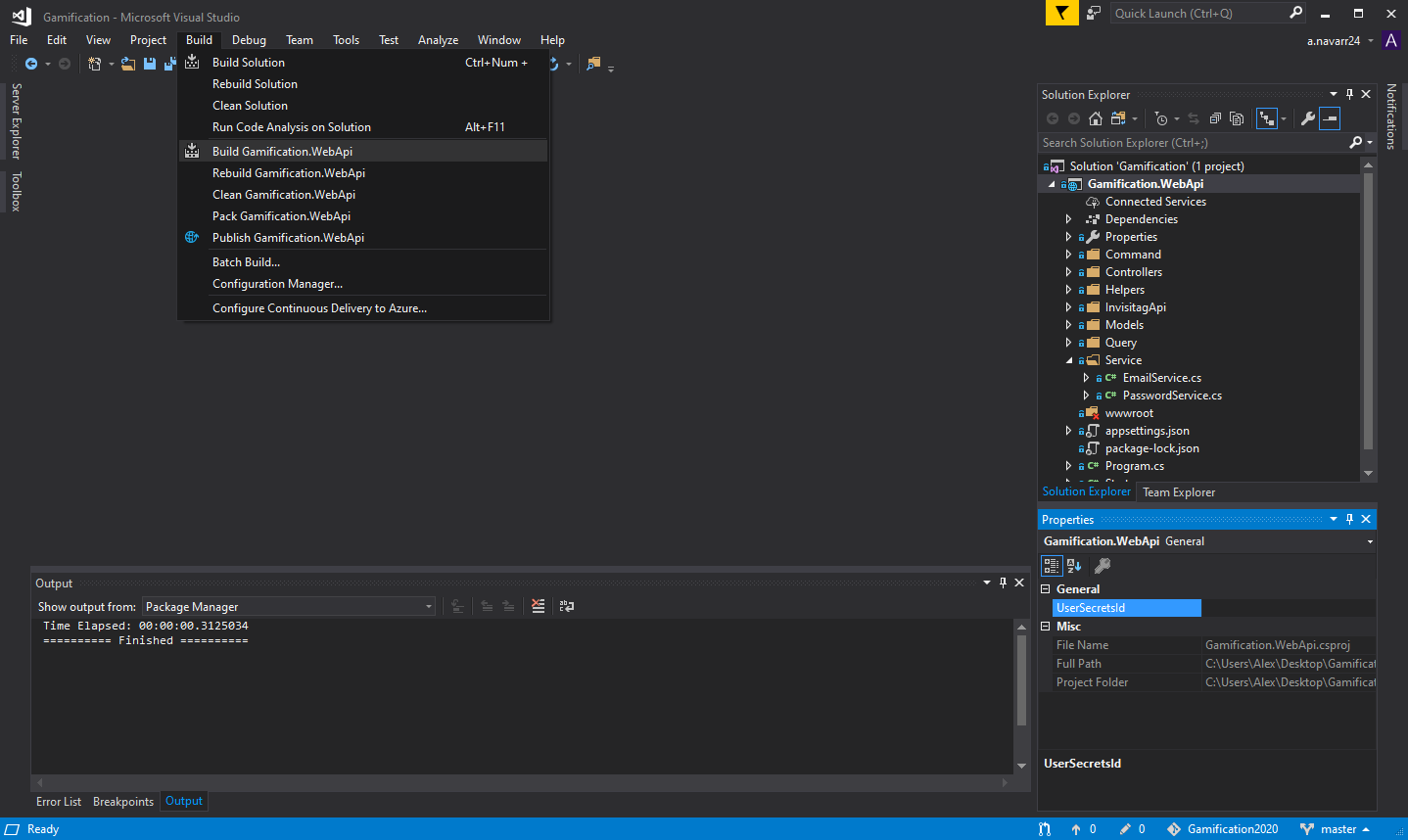
How to run the program locally:

Backend:

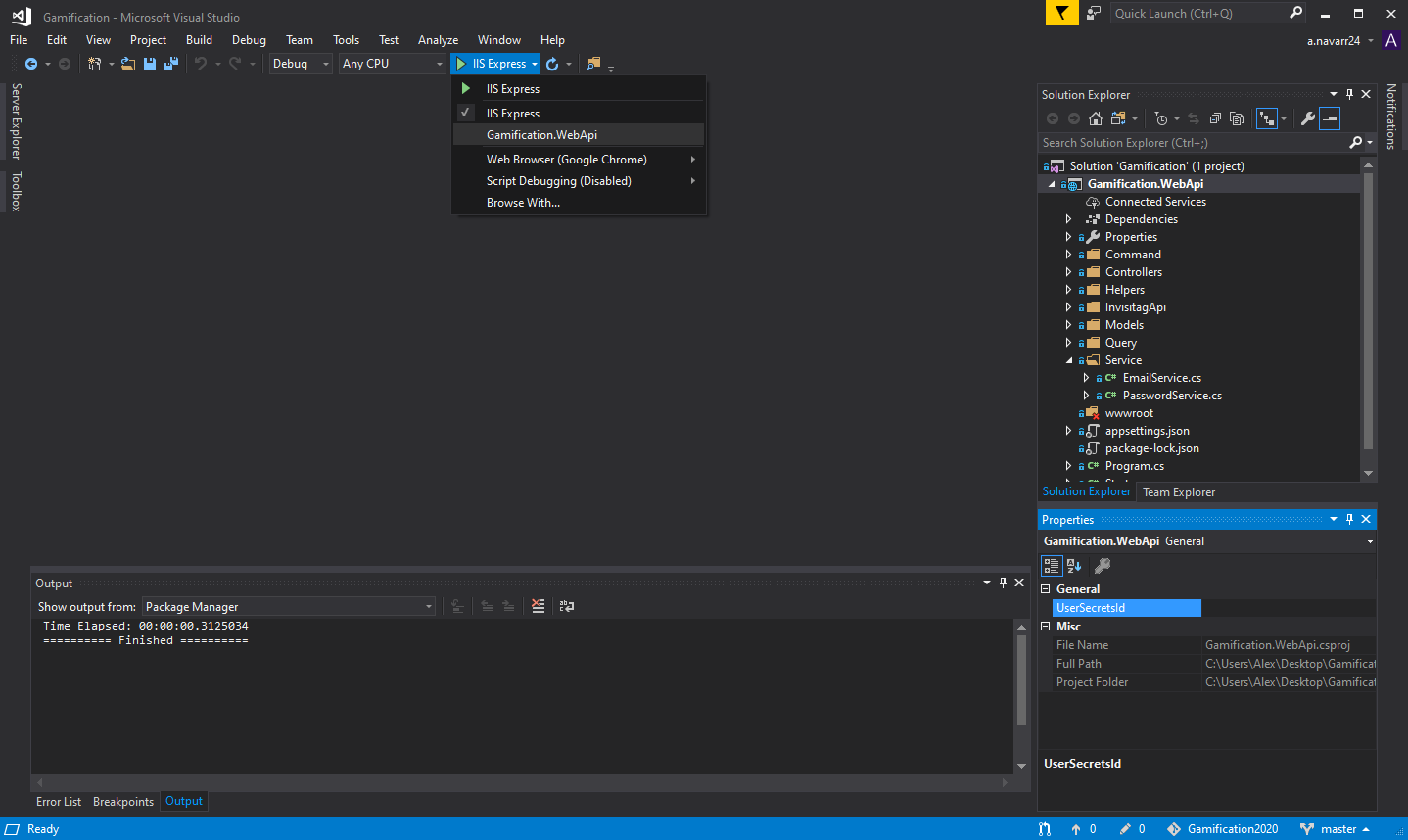
* Open ‘Gamification.sln’



* Build in Microsoft Visual Studio

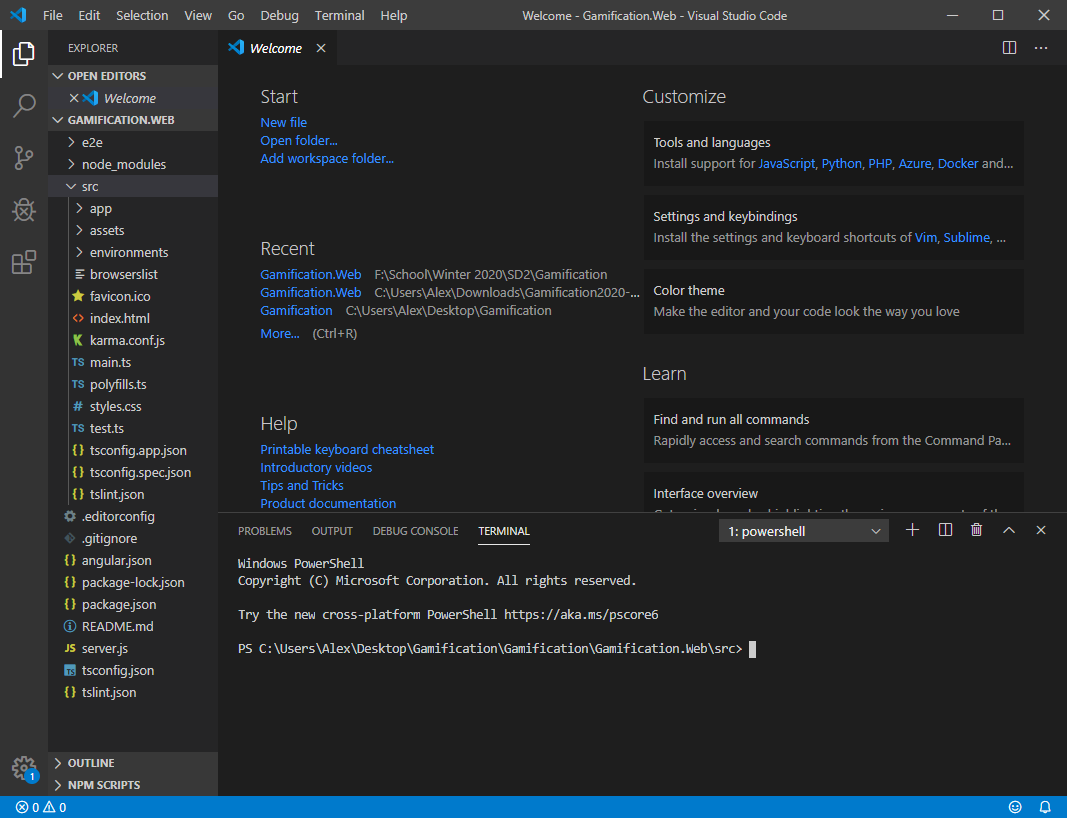


* Select ‘Gamification.WebAPI’ and run

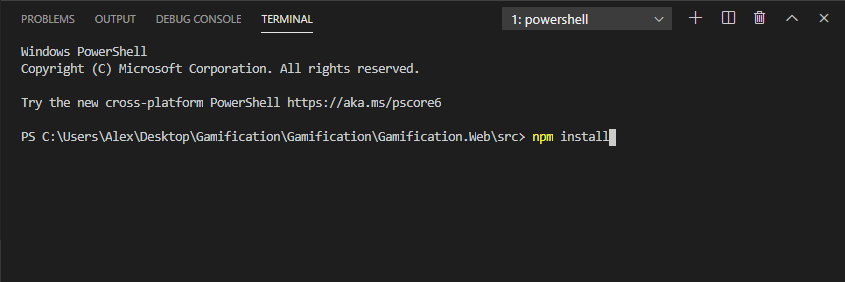


Frontend:

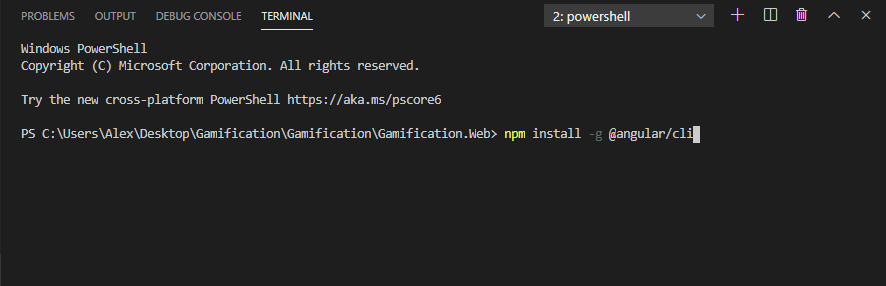
* Navigate to Gamification.Web in the terminal



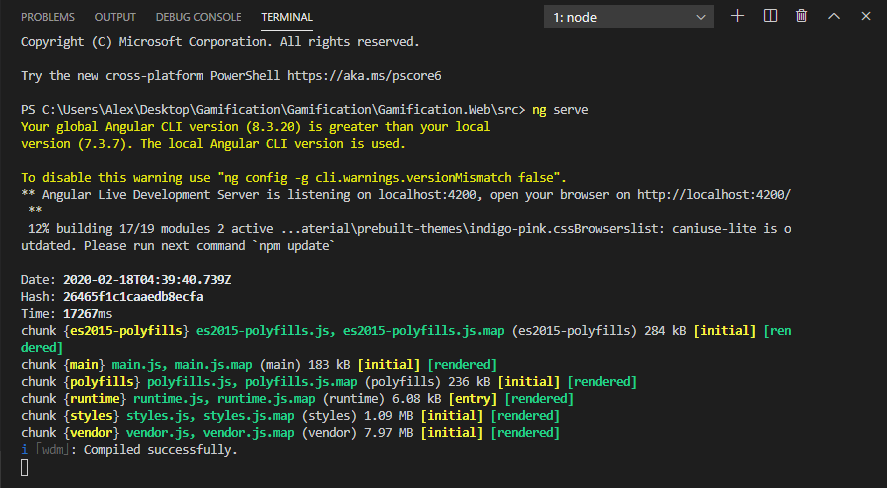
* Type ‘npm install’ to install all of the node packages



* Type ‘npm install -g @angular/cli’ to install the NuGet package



* Type ‘ng serve’ to build and deploy the front end locally or ‘ng serve --host 0.0.0.0’ to deploy the front end to the web



* Navigate to localhost:4200 or IP:4200

## 7.3 Design Metrics to be Used

The Design metrics we are using are CoCoMo and function points, with time being defined in Man-Hours/Man-Days/Man-Weeks

## 7.4 Supplementary Information

Client Api:

<http://api.invisi-tag.com/docs/#/>

Client Software demonstration:

<https://www.youtube.com/watch?v=e9aQ1T808Aw>

Virtual Hosting:

<https://aws.amazon.com/ec2/>

Database:

<https://aws.amazon.com/rds/>

Frontend:

<http://sudiptachaudhari.com/angular-7-asp-net-core/>

Graph:

<https://canvasjs.com/>

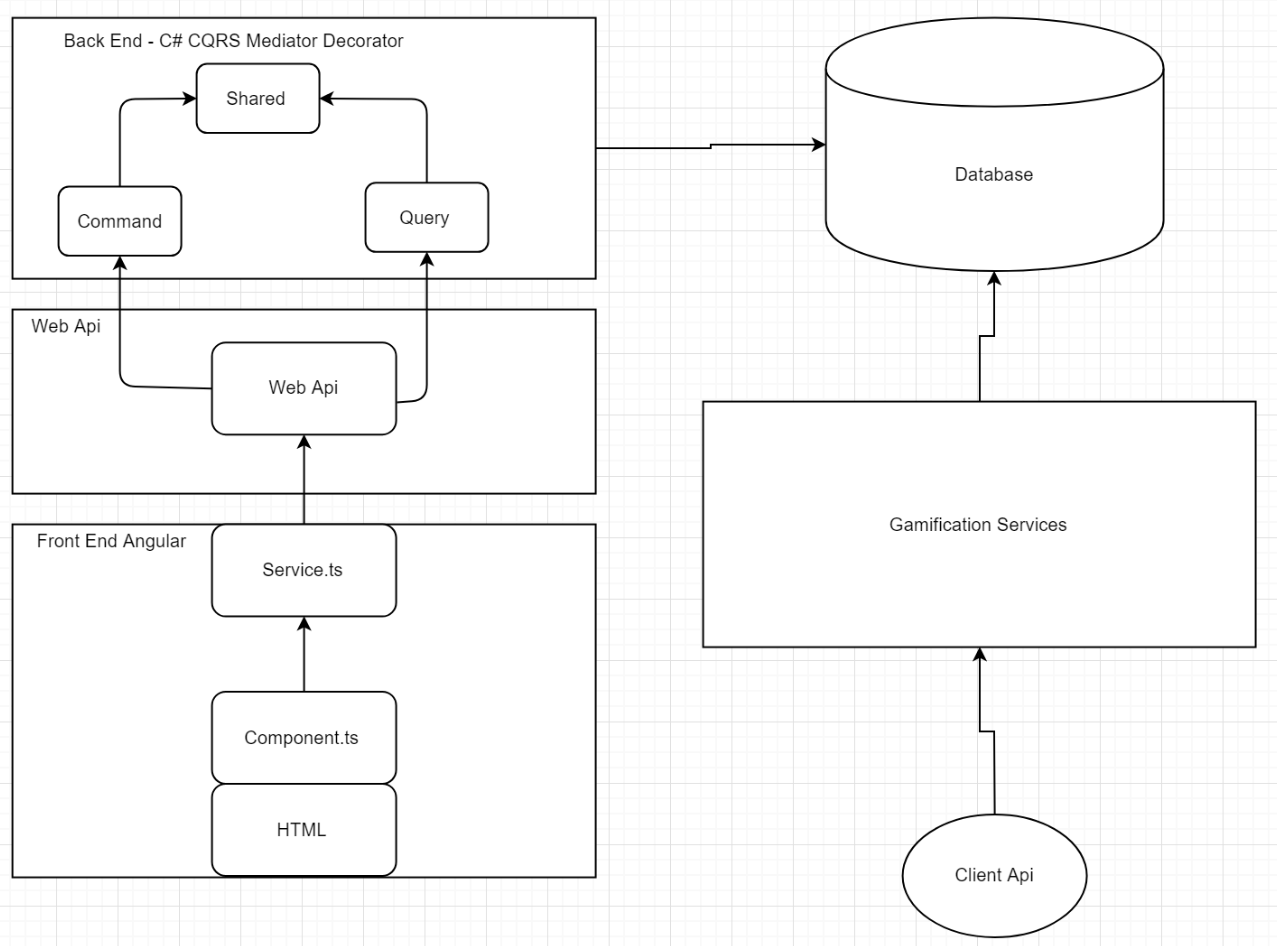
<https://www.chartjs.org/>

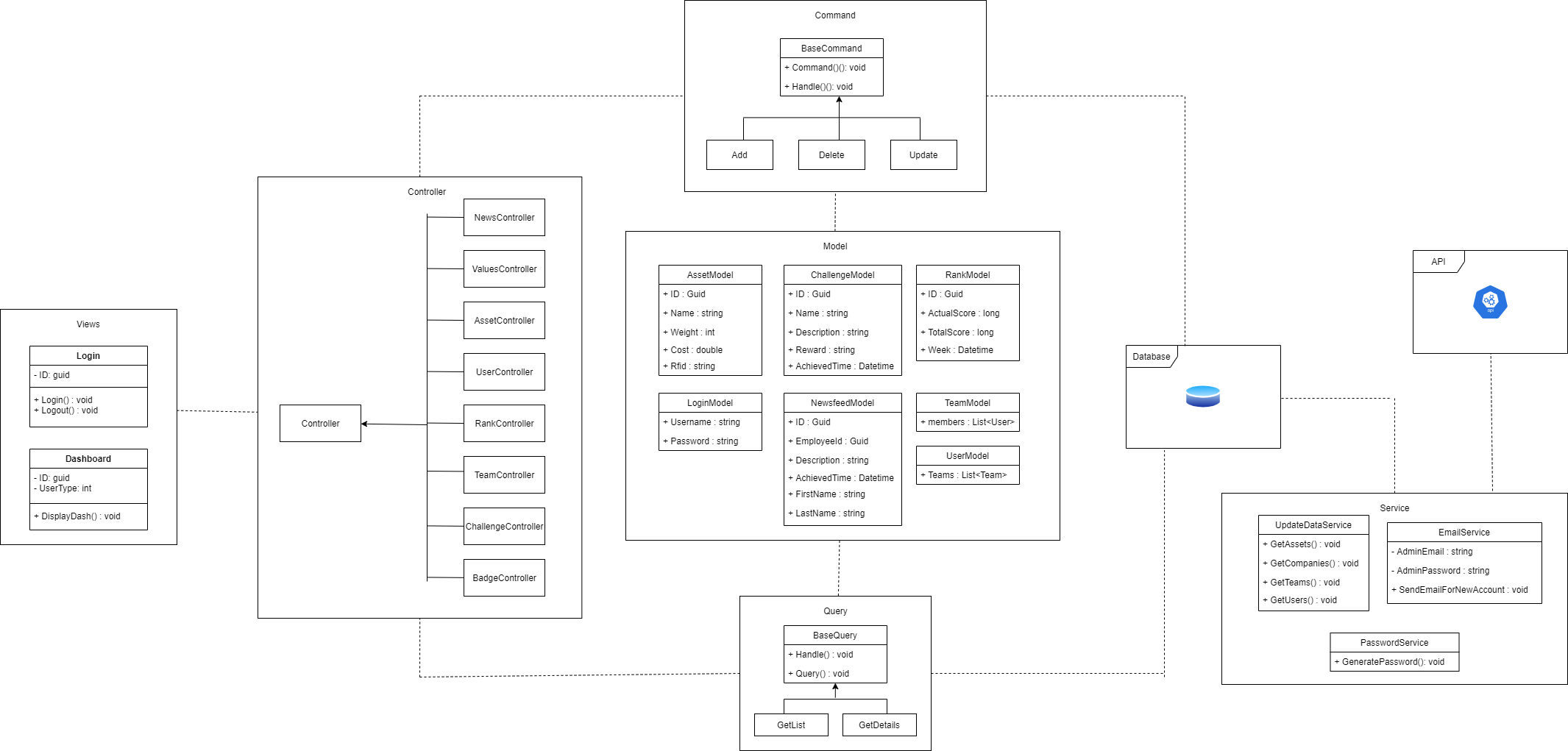
Badges:

<https://www.flaticon.com/free-icons/badge>

## 7.5 UML Diagrams

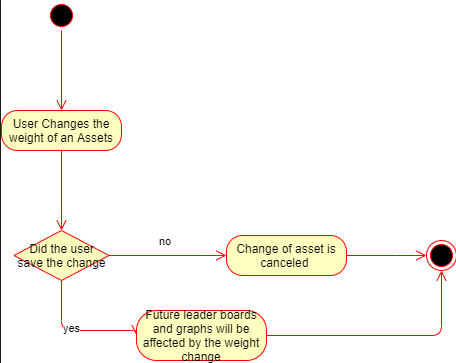
## 7.5.1 Architecture/Class diagrams

.

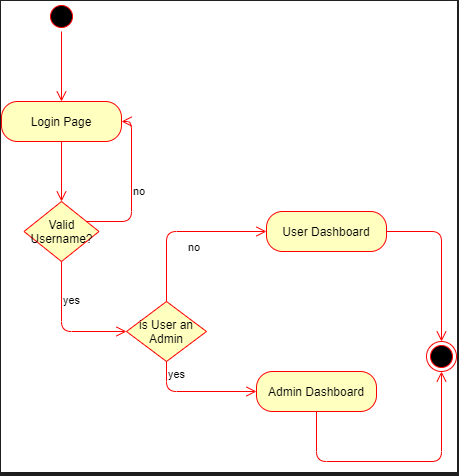


## 7.5.2 State diagrams

## 7.5.2.A Manage Assets



## 7.5.2.B Login State Diagram



## 7.5.3 Communication Diagram