[**1.0 Introduction**](#_heading=h.3znysh7) **5**

[**1.1 Goals and objectives**](#_heading=h.2et92p0) **[5](#_heading=h.2et92p0)**

[**1.2 Statement of scope**](#_heading=h.3dy6vkm) **[5](#_heading=h.3dy6vkm)**

[**1.3 Software context**](#_heading=h.1t3h5sf) **[5](#_heading=h.1t3h5sf)**

[**1.4 Major constraints**](#_heading=h.4d34og8) **[5](#_heading=h.4d34og8)**

[**2.0 Data design**](#_heading=h.2s8eyo1) **[6](#_heading=h.2s8eyo1)**

[**2.1 Internal software data structure**](#_heading=h.17dp8vu) **[6](#_heading=h.17dp8vu)**

[**2.2 Global data structure**](#_heading=h.3rdcrjn) **[6](#_heading=h.3rdcrjn)**

[**2.3 Temporary data structure**](#_heading=h.26in1rg) **[7](#_heading=h.26in1rg)**

[**2.4 Database description**](#_heading=h.lnxbz9) **[7](#_heading=h.lnxbz9)**

[**3.0 Architectural and component-level design**](#_heading=h.35nkun2) **[7](#_heading=h.35nkun2)**

[**3.1 Program Structure**](#_heading=h.1ksv4uv) **[7](#_heading=h.1ksv4uv)**

[**3.1.1 Architecture diagram**](#_heading=h.44sinio) **[8](#_heading=h.44sinio)**

[**3.1.2 Alternatives**](#_heading=h.2jxsxqh) **[8](#_heading=h.2jxsxqh)**

[**3.2 Description for Component manage team**](#_heading=h.z337ya) **[9](#_heading=h.z337ya)**

[**3.2.2 Component interface description.**](#_heading=h.3j2qqm3) **[9](#_heading=h.3j2qqm3)**

[**3.2.3 Sub-Component add team processing detail**](#_heading=h.1y810tw) **[9](#_heading=h.1y810tw)**

[**3.2.3.1 Interface description**](#_heading=h.4i7ojhp) **[9](#_heading=h.4i7ojhp)**

[**3.2.3.2 Algorithmic model (e.g., PDL)**](#_heading=h.2xcytpi) **[9](#_heading=h.2xcytpi)**

[**3.2.3.3 Restrictions/limitations**](#_heading=h.1ci93xb) **[9](#_heading=h.1ci93xb)**

[**3.2.3.4 Local data structures**](#_heading=h.3whwml4) **[9](#_heading=h.3whwml4)**

[**3.2.3.5 Performance issues**](#_heading=h.2bn6wsx) **[9](#_heading=h.2bn6wsx)**

[**3.2.3.6 Design constraints**](#_heading=h.qsh70q) **[10](#_heading=h.qsh70q)**

[**3.2 Description for Component team detail**](#_heading=h.3as4poj) **[10](#_heading=h.3as4poj)**

[**3.2.1 Processing narrative (PSPEC) for component team detail**](#_heading=h.1pxezwc) **[10](#_heading=h.1pxezwc)**

[**3.2.2 Component interface description.**](#_heading=h.49x2ik5) **[10](#_heading=h.49x2ik5)**

[**3.2.3 Sub-Component add team processing detail**](#_heading=h.2p2csry) **[10](#_heading=h.2p2csry)**

[**3.2.3.1 Interface description**](#_heading=h.147n2zr) **[10](#_heading=h.147n2zr)**

[**3.2.3.2 Algorithmic model (e.g., PDL)**](#_heading=h.3o7alnk) **[10](#_heading=h.3o7alnk)**

[**3.2.3.3 Restrictions/limitations**](#_heading=h.23ckvvd) **[10](#_heading=h.23ckvvd)**

[**3.2.3.4 Local data structures**](#_heading=h.ihv636) **[10](#_heading=h.ihv636)**

[**3.2.3.5 Performance issues**](#_heading=h.32hioqz) **[10](#_heading=h.32hioqz)**

[**3.2.3.6 Design constraints**](#_heading=h.1hmsyys) **[10](#_heading=h.1hmsyys)**

[**3.2 Description for Component manage employees**](#_heading=h.41mghml) **[11](#_heading=h.41mghml)**

[**3.2.1 Processing narrative (PSPEC) for component manage employees**](#_heading=h.2grqrue) **[11](#_heading=h.2grqrue)**

[**3.2.2 Component manage employees interface description.**](#_heading=h.vx1227) **[11](#_heading=h.vx1227)**

[**3.2.3 Sub-Component n.m processing detail**](#_heading=h.3fwokq0) **[11](#_heading=h.3fwokq0)**

[**3.2.3.1 Interface description for Add Employee**](#_heading=h.1v1yuxt) **[11](#_heading=h.1v1yuxt)**

[**3.2.3.2 Algorithmic model (e.g., PDL)**](#_heading=h.4f1mdlm) **[11](#_heading=h.4f1mdlm)**

[**3.2.3.3 Restrictions/limitations**](#_heading=h.2u6wntf) **[11](#_heading=h.2u6wntf)**

[**3.2.3.4 Local data structures**](#_heading=h.19c6y18) **[11](#_heading=h.19c6y18)**

[**3.2.3.5 Performance issues**](#_heading=h.3tbugp1) **[11](#_heading=h.3tbugp1)**

[**3.2.3.6 Design constraints**](#_heading=h.28h4qwu) **[11](#_heading=h.28h4qwu)**

[**3.2.3.1 Interface description for Remove Employee**](#_heading=h.nmf14n) **[11](#_heading=h.nmf14n)**

[**3.2.3.2 Algorithmic model (e.g., PDL)**](#_heading=h.37m2jsg) **[11](#_heading=h.37m2jsg)**

[**3.2.3.3 Restrictions/limitations**](#_heading=h.1mrcu09) **[12](#_heading=h.1mrcu09)**

[**3.2.3.4 Local data structures**](#_heading=h.46r0co2) **[12](#_heading=h.46r0co2)**

[**3.2.3.5 Performance issues**](#_heading=h.2lwamvv) **[12](#_heading=h.2lwamvv)**

[**3.2.3.6 Design constraints**](#_heading=h.111kx3o) **[12](#_heading=h.111kx3o)**

[**3.2.3.1 Interface description for Edit Employee**](#_heading=h.3l18frh) **[12](#_heading=h.3l18frh)**

[**3.2.3.2 Algorithmic model (e.g., PDL)**](#_heading=h.206ipza) **[12](#_heading=h.206ipza)**

[**3.2.3.3 Restrictions/limitations**](#_heading=h.4k668n3) **[12](#_heading=h.4k668n3)**

[**3.2.3.4 Local data structures**](#_heading=h.2zbgiuw) **[12](#_heading=h.2zbgiuw)**

[**3.2.3.5 Performance issues**](#_heading=h.1egqt2p) **[12](#_heading=h.1egqt2p)**

[**3.2.3.6 Design constraints**](#_heading=h.3ygebqi) **[12](#_heading=h.3ygebqi)**

[**3.2.1 Processing narrative (PSPEC) for component main-page**](#_heading=h.2dlolyb) **[13](#_heading=h.2dlolyb)**

[**3.2.2 Component main-page interface description.**](#_heading=h.sqyw64) **[13](#_heading=h.sqyw64)**

[**3.2.3 Sub-Component main-page.table processing detail**](#_heading=h.3cqmetx) **[13](#_heading=h.3cqmetx)**

[**3.2.3.1 Interface description**](#_heading=h.1rvwp1q) **[13](#_heading=h.1rvwp1q)**

[**3.2.3.2 Algorithmic model (e.g., PDL)**](#_heading=h.4bvk7pj) **[13](#_heading=h.4bvk7pj)**

[**3.2.3.3 Restrictions/limitations**](#_heading=h.2r0uhxc) **[13](#_heading=h.2r0uhxc)**

[**3.2.3.4 Local data structures**](#_heading=h.1664s55) **[13](#_heading=h.1664s55)**

[**3.2.3.5 Performance issues**](#_heading=h.3q5sasy) **[13](#_heading=h.3q5sasy)**

[**3.2.3.6 Design constraints**](#_heading=h.25b2l0r) **[13](#_heading=h.25b2l0r)**

[**3.2.1 Processing narrative (PSPEC) for component team-rank-history**](#_heading=h.kgcv8k) **[14](#_heading=h.kgcv8k)**

[**3.2.2 Component team-rank-history interface description.**](#_heading=h.34g0dwd) **[14](#_heading=h.34g0dwd)**

[**3.2.3 Sub-Component Line-chart processing detail**](#_heading=h.1jlao46) **[14](#_heading=h.1jlao46)**

[**3.2.3.1 Interface description**](#_heading=h.43ky6rz) **[14](#_heading=h.43ky6rz)**

[**3.2.3.2 Algorithmic model (e.g., PDL)**](#_heading=h.2iq8gzs) **[14](#_heading=h.2iq8gzs)**

[**3.2.3.3 Restrictions/limitations**](#_heading=h.xvir7l) **[14](#_heading=h.xvir7l)**

[**3.2.3.4 Local data structures**](#_heading=h.3hv69ve) **[14](#_heading=h.3hv69ve)**

[**3.2.3.5 Performance issues**](#_heading=h.1x0gk37) **[14](#_heading=h.1x0gk37)**

[**3.2.3.6 Design constraints**](#_heading=h.4h042r0) **[15](#_heading=h.4h042r0)**

[**3.2.1 Processing narrative (PSPEC) for component n**](#_heading=h.2w5ecyt) **[15](#_heading=h.2w5ecyt)**

[**3.2.2 Component n interface description.**](#_heading=h.1baon6m) **[15](#_heading=h.1baon6m)**

[**3.2.3 Sub-Component n.m processing detail**](#_heading=h.3vac5uf) **[15](#_heading=h.3vac5uf)**

[**3.2.3.1 Interface description**](#_heading=h.2afmg28) **[15](#_heading=h.2afmg28)**

[**3.2.3.2 Algorithmic model (e.g., PDL)**](#_heading=h.pkwqa1) **[15](#_heading=h.pkwqa1)**

[**3.2.3.3 Restrictions/limitations**](#_heading=h.39kk8xu) **[15](#_heading=h.39kk8xu)**

[**3.2.3.4 Local data structures**](#_heading=h.1opuj5n) **[15](#_heading=h.1opuj5n)**

[**3.2.3.5 Performance issues**](#_heading=h.48pi1tg) **[15](#_heading=h.48pi1tg)**

[**3.2.3.6 Design constraints**](#_heading=h.2nusc19) **[15](#_heading=h.2nusc19)**

[**3.3 Software Interface Description**](#_heading=h.1302m92) **[15](#_heading=h.1302m92)**

[**3.3.1 External machine interfaces**](#_heading=h.3mzq4wv) **[15](#_heading=h.3mzq4wv)**

[**3.3.2 External system interfaces**](#_heading=h.2250f4o) **[16](#_heading=h.2250f4o)**

[**3.3.3 Human interface**](#_heading=h.haapch) **[16](#_heading=h.haapch)**

[**4.0 User interface design**](#_heading=h.319y80a) **[16](#_heading=h.319y80a)**

[**4.1 Description of the user interface**](#_heading=h.1gf8i83) **[17](#_heading=h.1gf8i83)**

[**4.1.1 Screen images**](#_heading=h.40ew0vw) **[17](#_heading=h.40ew0vw)**

[**4.1.2 Objects and actions**](#_heading=h.2fk6b3p) **[19](#_heading=h.2fk6b3p)**

[**4.2 Interface design rules**](#_heading=h.upglbi) **[20](#_heading=h.upglbi)**

[**4.3 Components available**](#_heading=h.3ep43zb) **[20](#_heading=h.3ep43zb)**

[**5.0 Restrictions, limitations, and constraints**](#_heading=h.4du1wux) **21**

[**6.0 Testing Issues**](#_heading=h.2szc72q) **[21](#_heading=h.2szc72q)**

[**6.1 Classes of tests**](#_heading=h.184mhaj) **[21](#_heading=h.184mhaj)**

[**6.2 Expected software response**](#_heading=h.3s49zyc) **[23](#_heading=h.3s49zyc)**

[**6.3 Performance bounds**](#_heading=h.279ka65) **[25](#_heading=h.279ka65)**

[**6.4 Identification of critical components**](#_heading=h.meukdy) **[25](#_heading=h.meukdy)**

[**7.0 Appendices**](#_heading=h.36ei31r) **[25](#_heading=h.36ei31r)**

[**7.1 Requirements traceability matrix**](#_heading=h.1ljsd9k) **[26](#_heading=h.1ljsd9k)**

[**7.2 Packaging and installation issues**](#_heading=h.2koq656) **[26](#_heading=h.2koq656)**

[**7.3 Design metrics to be used**](#_heading=h.zu0gcz) **[26](#_heading=h.zu0gcz)**

[**7.4 Supplementary information (as required)**](#_heading=h.3jtnz0s) **27**

### 

### 

### 

### 1.0 Introduction

This section provides an overview of the entire design document. This document describes all data, architectural, interface and component-level design for the software.

### 1.1 Goals and objectives

This project is to develop a web application for the gamification of big data. The data that will be analyzed is data received from an inventory software, which keeps inventory of equipment that is used by rescue teams. These teams should utilize the inventory software whenever they have jobs. The data will be monitored to check if the software is being utilized, and if inventory is being returned. The data will be analyzed, and a scoring system will rank the teams, based on how much they utilized the software and if their equipment was returned. This will increase productivity.

### 1.2 Statement of scope

For the Gamification software we are using data from the client’s API to create a gamification website. We are using metrics as input to measure performance. Metrics included scanning out all the appropriate assets, and scanning them back in. We will use the Invisi-tags API to pull data about crews and jobs they have done. The gamification will give admins of the website the ability to edit values for each metric used, to better balance the score according to their priorities. The admins will also be able to create teams and employee accounts. Also Assign members to those teams. The software will calculate the performance and output the scores onto the website. The output will be a well-crafted leaderboard that will inform the users how well the crews operate compared to other crews and if they did well over all.

### 1.3 Software context

This software is going to be used by employees that are divided into teams. the objective is to boost the teams’ productivity by making them compete against each other to rise to the top of the ranking. After solving the problem of designing a software that is able to track and measure productivity accurately, Incentives could be put in place as an additional motivation to become more productive. By creating this software and seeing its effect, the goal is to ultimately be able to create software’s of similar nature that would solve a large variety of similar problems by implementing the gamification method. The client has customers that will use this software to manage their teams.

### 1.4 Major constraints

* + Time: meetings are only done twice a week due to individual schedules.
  + Finding work around due to client api
  + Database sharding adding another complexity during logging, pulling data from api for each customer of the clients, and making new users

### 2.0 Data design

A description of all data structures including internal, global, and temporary data structures.

### 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.
  + **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.
  + **Ranking**
  + The ranking table consists of the following attributes.
    - **Id** (Primary Key) – The Id representing the rank of the team.
    - **CompanyId** (Foreign Key) – The Id representing the company the team works for.
    - **TeamId** (Foreign Key) – The Id representing the team the rank belongs to.
    - **Week** – The week of the rank to be viewed.
    - **ActualScore** – The score the team earns in a specific week.
    - **TotalScore** – The highest score the team can earn for the week.
  + **Assets**
  + The assets table consists of the following attributes.
    - **Id** (Primary Key) – The Id representing the asset.
    - **CompanyId** (Foreign Key) – The Id representing the company the asset belongs to.
    - **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.

### 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)
    - **CompanyId** (Foreign Key)
    - **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.

### 2.3 Temporary data structure

* Files created for interim use are described.
  + There will be no temporary data structures used.

### 2.4 Database description

* Database(s) created as part of the application is(are) described.
  + The database we are using is a Microsoft SQL Database. Microsoft SQL Server Management Studio is used to create the database. It will contain all data inputted during development. It will also contain any information added from the user end.

### 3.0 Architectural and component-level design

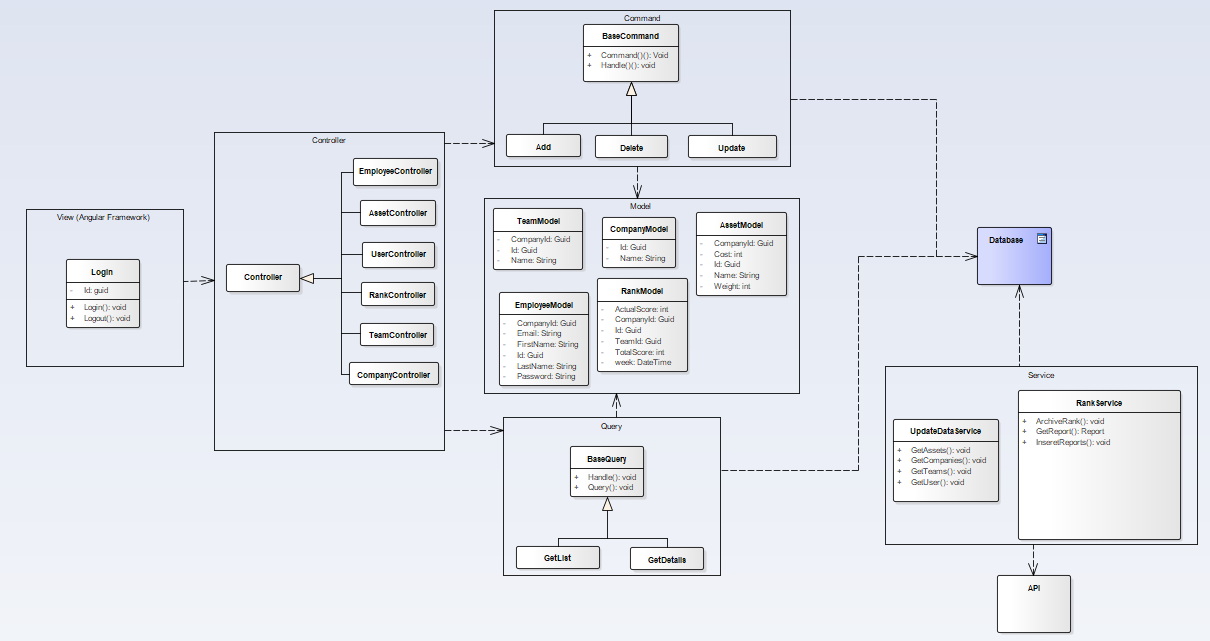
A description of the program architecture is presented.

### 3.1 Program Structure

* A detailed description the program structure chosen for the application is presented.

### 3.1.1 Architecture diagram

### 3.1.2 Alternatives

* + N/A
  + 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.

|  |  |
| --- | --- |
| 3.2 Description for Component manage team  * + **3.2.1 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.  3.2.2 Component interface description.  * + - A table displays all the team name 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  3.2.3 Sub-Component add team processing detail  |  | | --- | | 3.2.3.1 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  3.2.3.2 Algorithmic model (e.g., PDL)  * + Get teams from api   + Check teams retrieved   + If team exist in database     - remove from list   + Return teams to the 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  3.2.3.3 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 makes api calls   + The team controller must be working properly   + The client api must be allow request to be made and results to be returned  3.2.3.4 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  3.2.3.5 Performance issues  * + Retrieving teams from the clients api - making and waiting for a response may cause a delay  3.2.3.6 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. | |
| 3.2 Description for Component team detail3.2.1 Processing narrative (PSPEC) for component team detail  * + - Users selects a team from team list, a get request will be sent to the team controller, for the get team api with the team id. The mediator will the invoke the query GetTeam and retrieve the team data from the database.  3.2.2 Component 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 statu are on top  3.2.3 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 component   + The front end service that makes api calls   + The team controller must be working properly  3.2.3.4 Local data structures  * + Team     - Id: guid     - Name: string     - Is active   + User     - FirstName: string     - LastName: string     - Email: string     - Image: binary   + Front end     - Array of User  3.2.3.5 Performance issues  * + N/A  3.2.3.6 Design constraints  * + When user removes members he must save before opening this dialog   + If the request fails or an error is encountered that user is skipped and the rest are saved and an error is then thrown after processing all the users. | |
| 3.2 Description for Component manage employees3.2.1 Processing narrative (PSPEC) for component manage employees  * + - User selects manage employees. A request is sent to the employee controller to get all employees listed in the user table. The mediator then invokes the get employees query and retrieves the employee data from the database.  3.2.2 Component manage employees interface description.  * + - When a admin logs in they have a menu selection. The admin will then select Manage Employees. A list of employees will be displayed. A menu will be displayed with the options to add remove or edit employees.  3.2.3 Sub-Component n.m processing detail  * + A detailed algorithmic description for each sub-component within the component n is presented. Section 3.2.3 is repeated for each of the m sub-components of component n.  |  | | --- | | 3.2.3.1 Interface description for Add Employee  * + Add employee is selected. Admin inputs employee information and submits data. The employee controller will send a request to command. Command will then insert into the user table in the database.  3.2.3.2 Algorithmic model (e.g., PDL)  * + Admin inputs employee information.   + Admin submits employee information.   + Employee password is generated and emailed to them.   + The mediator invokes the command insert employee.  3.2.3.3 Restrictions/limitations  * + There must be an internet connection.   + A connection to the database must be established.   + The manage employee page must load for the admin to be able to select add employee.  3.2.3.4 Local data structures  * + User     - FirstName: string     - LastName: string     - Email: string     - Image: binary  3.2.3.5 Performance issues  * + N/A  3.2.3.6 Design constraints  * + When and admin enters in the employee information he must submit for the employee to be inserted into the database.   + If invalid information is filled when the admin submits, an error message is displayed and the employee is not inserted. | | 3.2.3.1 Interface description for Remove Employee  * + Remove employee is selected. Admin confirms removal of employee. The employee controller will send a request to command. Command will then delete the user from the user table in the database.  3.2.3.2 Algorithmic model (e.g., PDL)  * + Admin selects remove employee.   + Admin confirms removal of employee.   + The mediator invokes the command delete employee.  3.2.3.3 Restrictions/limitations  * + There must be an internet connection.   + A connection to the database must be established.   + The manage employee page must load for the admin to be able to select remove employee.  3.2.3.4 Local data structures  * + User     - FirstName: string     - LastName: string     - Email: string     - Image: binary  3.2.3.5 Performance issues  * + N/A  3.2.3.6 Design constraints  * + When and admin removes an employee they must confirm the removal of that employee in order for them to be deleted from the database.   + If admin does not confirm removal of the employee, the employee is not deleted from the database. | | 3.2.3.1 Interface description for Edit Employee  * + Edit employee is selected. Admin updates employees information and submits data. The employee controller will send a request to command. Command will then update the user table in the database.  3.2.3.2 Algorithmic model (e.g., PDL)  * + Admin updates employees information.   + Admin submits changes.   + The mediator invokes the command update employee.  3.2.3.3 Restrictions/limitations  * + There must be an internet connection.   + A connection to the database must be established.   + The manage employee page must load for the admin to be able to select update employee.  3.2.3.4 Local data structures  * + User     - FirstName: string     - LastName: string     - Email: string     - Image: binary  3.2.3.5 Performance issues  * + N/A  3.2.3.6 Design constraints  * + When and admin updates the employees information he must submit for the employees information to be updated in the database.   + If invalid information is filled when the admin submits, an error message is displayed and the employees information is not updated. | |
| 3.2.1 Processing narrative (PSPEC) for component main-page User is redirected from the login page to the main-page. The main-page is responsible for displaying all the teams and their ranking for the current week.  3.2.2 Component main-page interface description. At the very top a main page header is displayed indicating to the user that they are indeed on the main page. Underneath a ranking header is displayed and a field titled filter is shown allowing the user to enter the team name they desire and the filter would filter out the results displayed in the team list. A table is used to display all the teams and their ranking. The table pulls the information from the backend and displays the data to the user. 3.2.3 Sub-Component main-page.table processing detail  |  | | --- | | 3.2.3.1 Interface description  * Input for table is company id and the output is the team names and ranks for all the teams in that company.  3.2.3.2 Algorithmic model (e.g., PDL) Pull company id from token, search team and ranking table,  Calculate total ranking of the week for each team, sort the table so the team with the most ranking would rise to the top 3.2.3.3 Restrictions/limitations Connection to the database must be established for the information to be displayed. 3.2.3.4 Local data structures Rank {  id: string;  teamName: string;  currentRank: number;  previousRank: number;  } 3.2.3.5 Performance issues3.2.3.6 Design constraints  * Attributes of the overall software design (including data structures, OS features, I/O, and interoperable systems) that constrain the design of this sub-component are presented. | |  | |
| 3.2.1 Processing narrative (PSPEC) for component team-rank-history From the main-page.tableRanking if the user clicks on a team’s name they would get redirected to the team-rank-history component. Where they would see a chart displaying the rank history of a team. 3.2.2 Component team-rank-history interface description. The component takes in a team id as input and returns a list of all rankings related to that specific team. 3.2.3 Sub-Component Line-chart processing detail A line chart component is used to display the data to the user. The component contains all the ranks displayed on the y axis vs all the weeks listed in the x axis.   |  | | --- | | 3.2.3.1 Interface description Charts is made of an x and y axis displaying ranks and the corresponding weeks. 3.2.3.2 Algorithmic model (e.g., PDL) Send team id to backend,  Search rank table and return all ranks,  Display ranks to user in the line-chart format 3.2.3.3 Restrictions/limitations Connection to the database must be established for the information to be displayed. 3.2.3.4 Local data structures TeamRanks {  id: string;  week: Date;  rankScore: number;  } 3.2.3.5 Performance issues  * Information on topics that may affect the run-time performance, security, or computational accuracy of this sub-component are presented.  3.2.3.6 Design constraints  * Attributes of the overall software design (including data structures, OS features, I/O, and interoperable systems) that constrain the design of this sub-component are presented. | |
| 3.2.1 Processing narrative (PSPEC) for component n3.2.2 Component n interface description.  * + A detailed description of the input and output interfaces for the component is presented.  3.2.3 Sub-Component n.m processing detail  * + A detailed algorithmic description for each sub-component within the component n is presented. Section 3.2.3 is repeated for each of the m sub-components of component n.  |  | | --- | | 3.2.3.1 Interface description  * A description of sub-component m inputs and outputs is presented.  3.2.3.2 Algorithmic model (e.g., PDL)  * The pseudocode listing for sub-component m is presented.  3.2.3.3 Restrictions/limitations  * The external environment and/or infrastructure that must exist for sub-component m to operate correctly is provided.  3.2.3.4 Local data structures  * The data structures used within sub-component m are presented.  3.2.3.5 Performance issues  * Information on topics that may affect the run-time performance, security, or computational accuracy of this sub-component are presented.  3.2.3.6 Design constraints  * Attributes of the overall software design (including data structures, OS features, I/O, and interoperable systems) that constrain the design of this sub-component are presented. * Software/database will be used for multiple companies meaning there may be cross over in names. teams names and asset names will have to be partially unique, some attribute values must be unique for a company but other companies may use that name: ex team1 can only be used by ford once but cheavy could also use the name team1 once. * Due to the large data sets that can cause delayed response times when pulling data from the database, we will be using horizontal partitioning which is a database design principle called sharding. * Software must pull data from the client api once a day. | |

### 3.3 Software Interface Description

* The software's interface(s) to the outside world are described.

### 3.3.1 External machine interfaces

* + The interface for the system is going to be a web application that can be accessed on any device that can browse the web.

### 3.3.2 External system interfaces

* + The system interacts with an api to fetch the metrics which would be used to calculate the rankings of the teams. The link for the api’s interface is provided in the Supplementary information section labeled Client’s Api.

### 3.3.3 Human interface

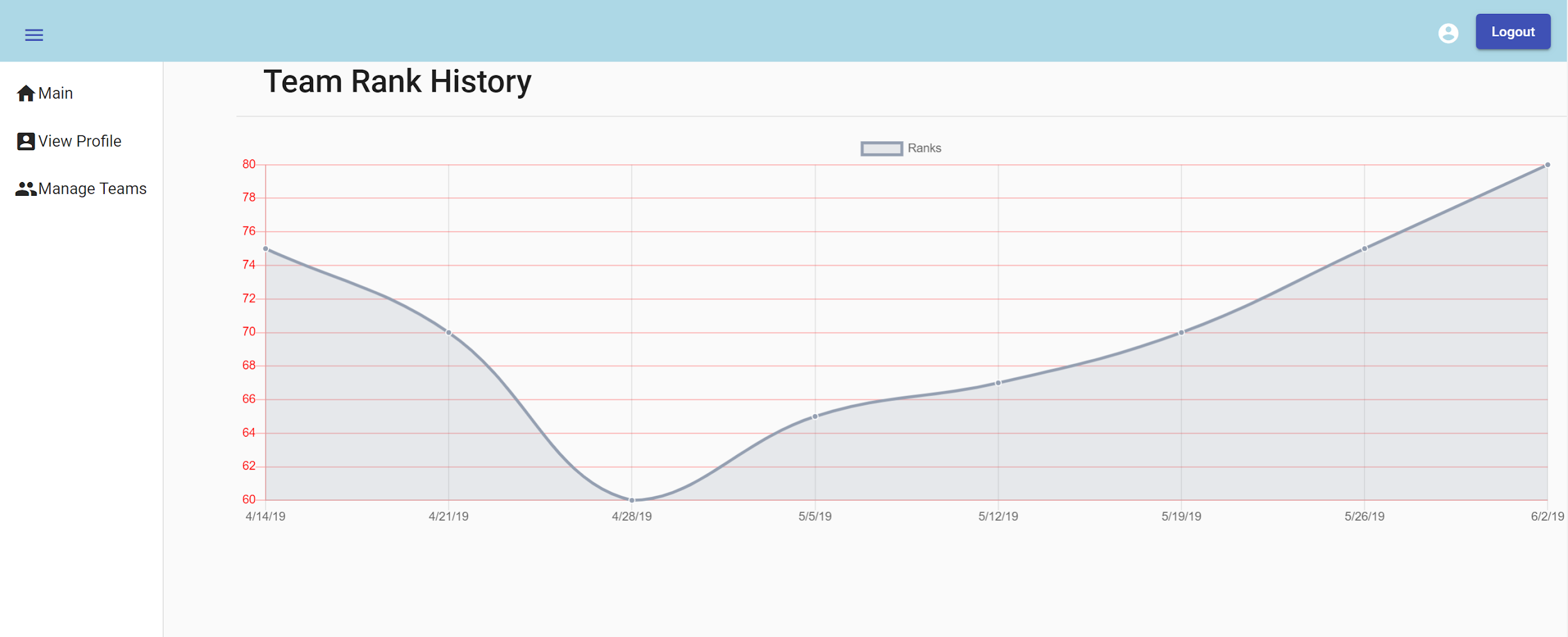
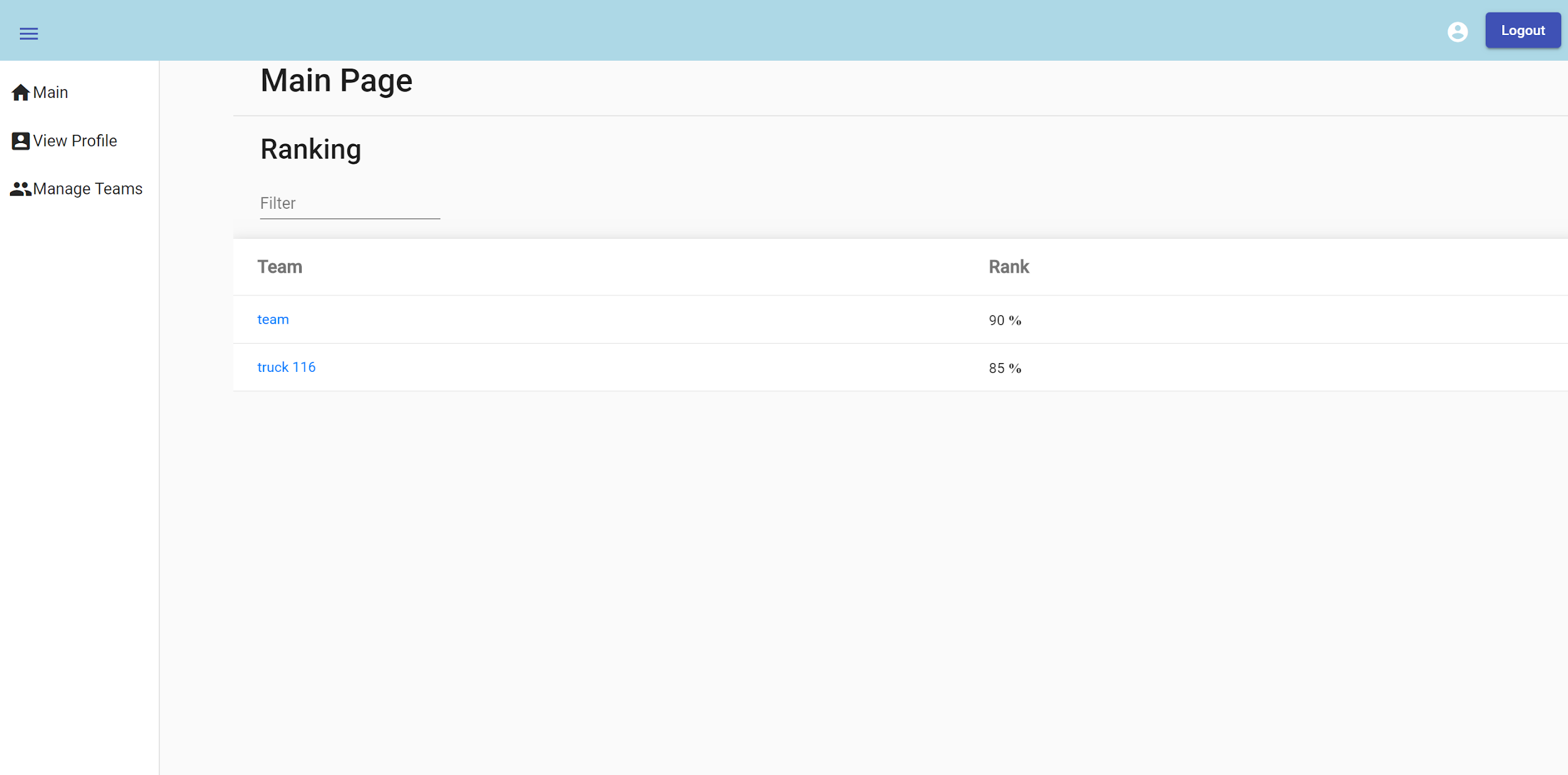
* + An overview of any human interfaces to be designed for the software is presented. See Section 4.0 for additional detail.
  + The interface that we chose will be designed in Html, CSS, and Typescript. The interface will have many controls found in basic web applications. The interface will provide users a way to quickly and easily check their rankings amongst other teams. For a more detailed description of the interface, view section 4.0.

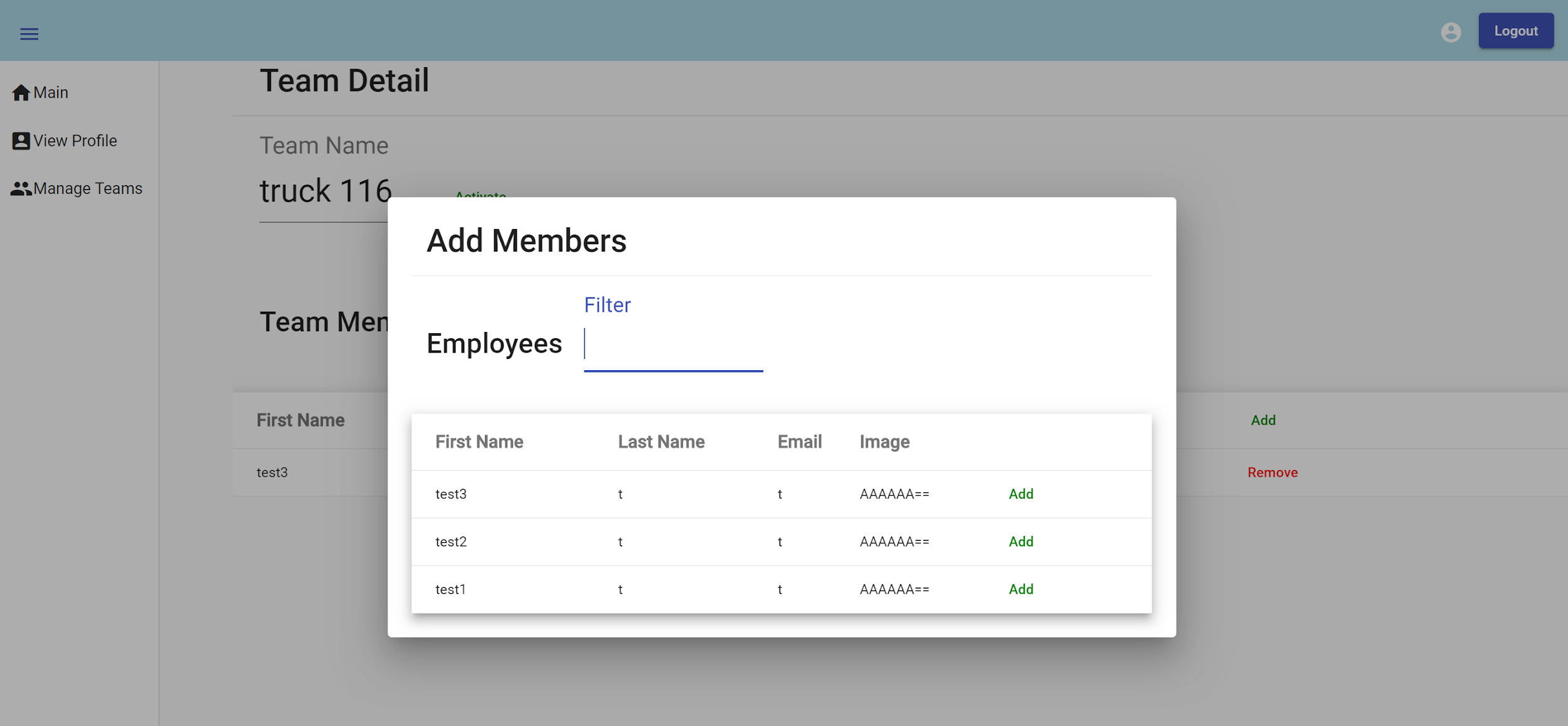
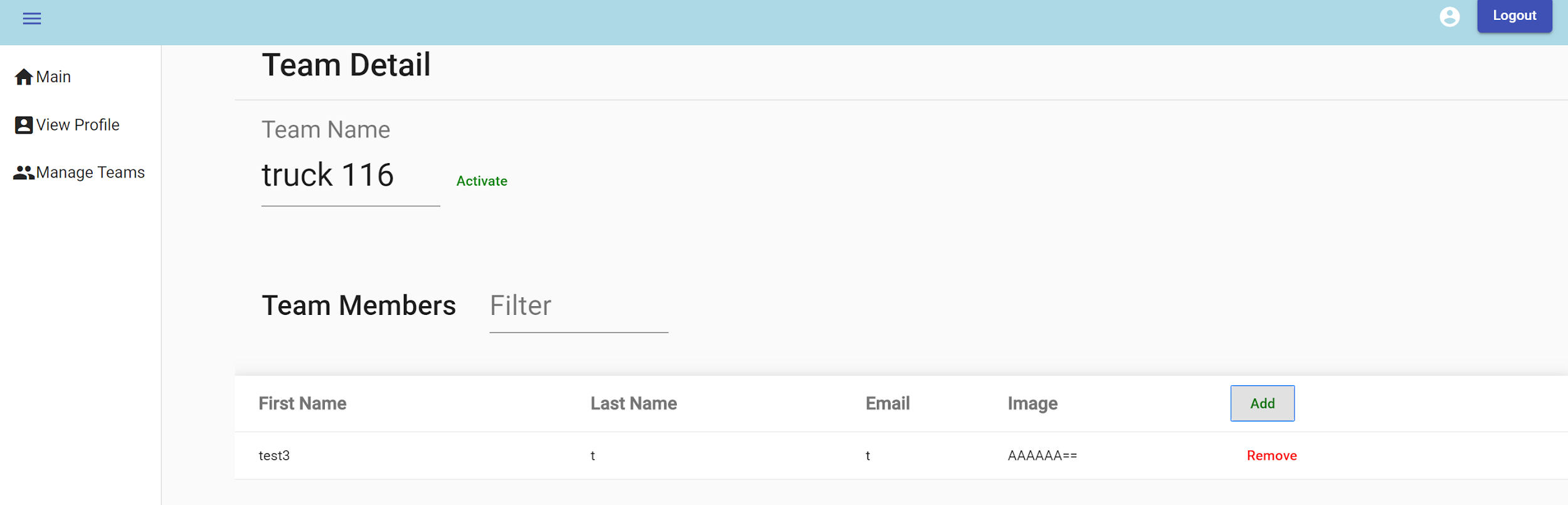
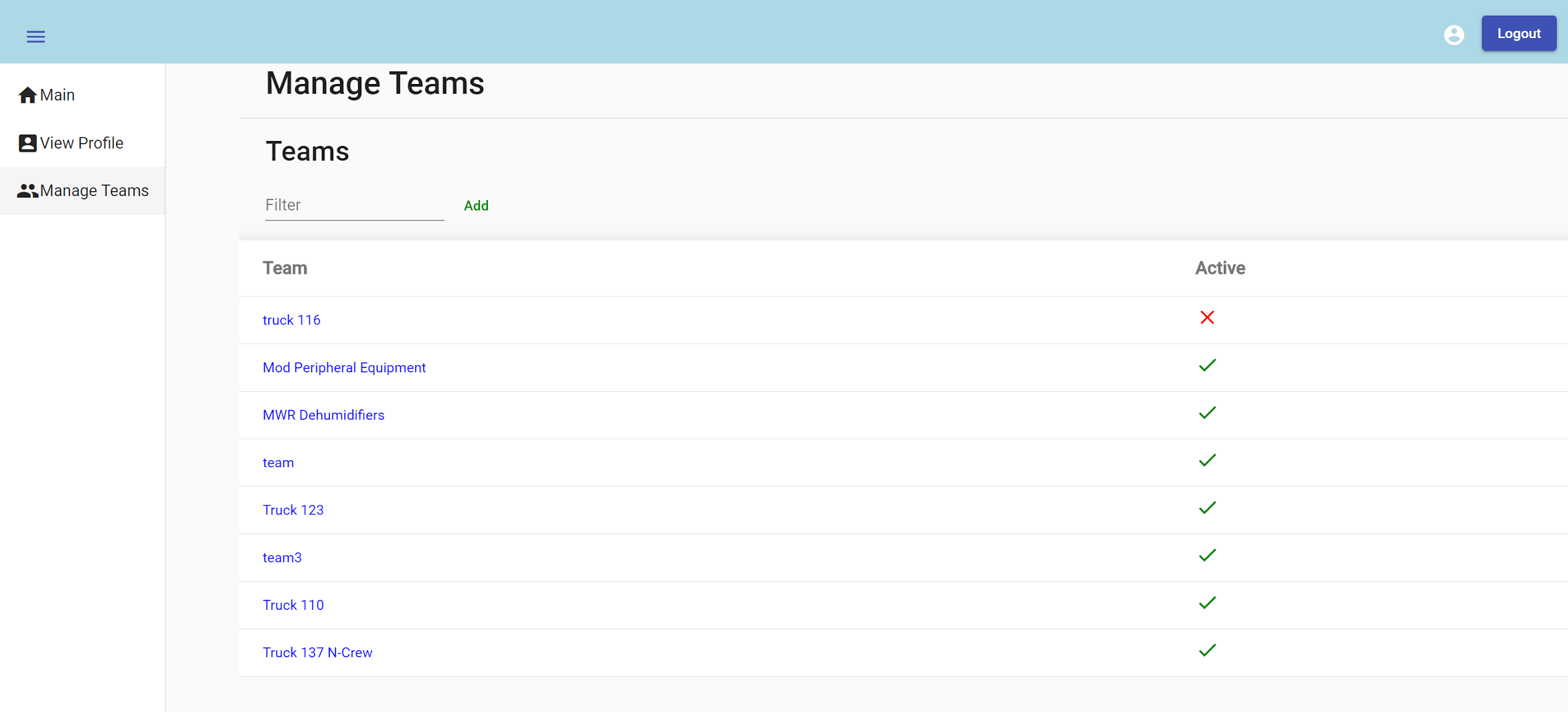
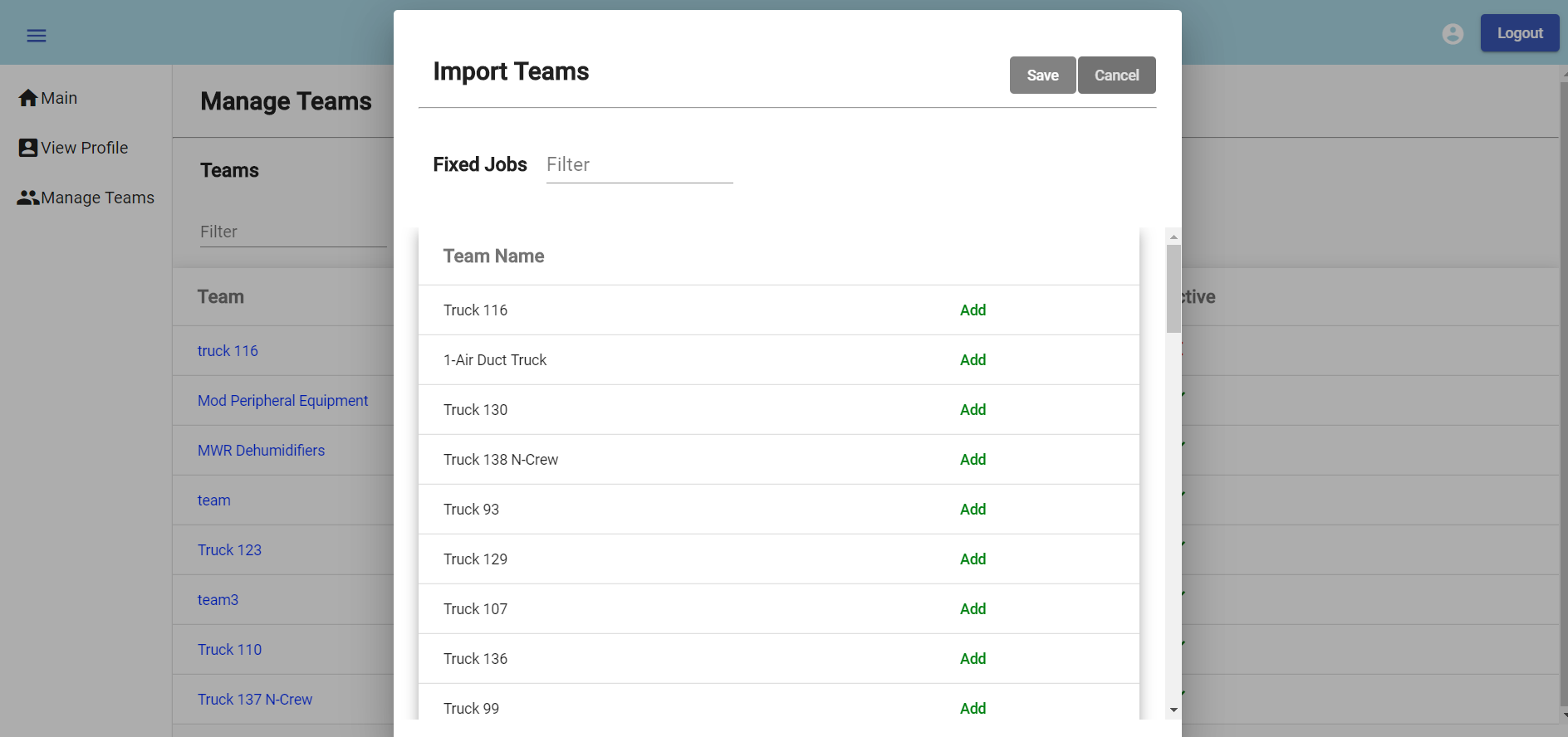
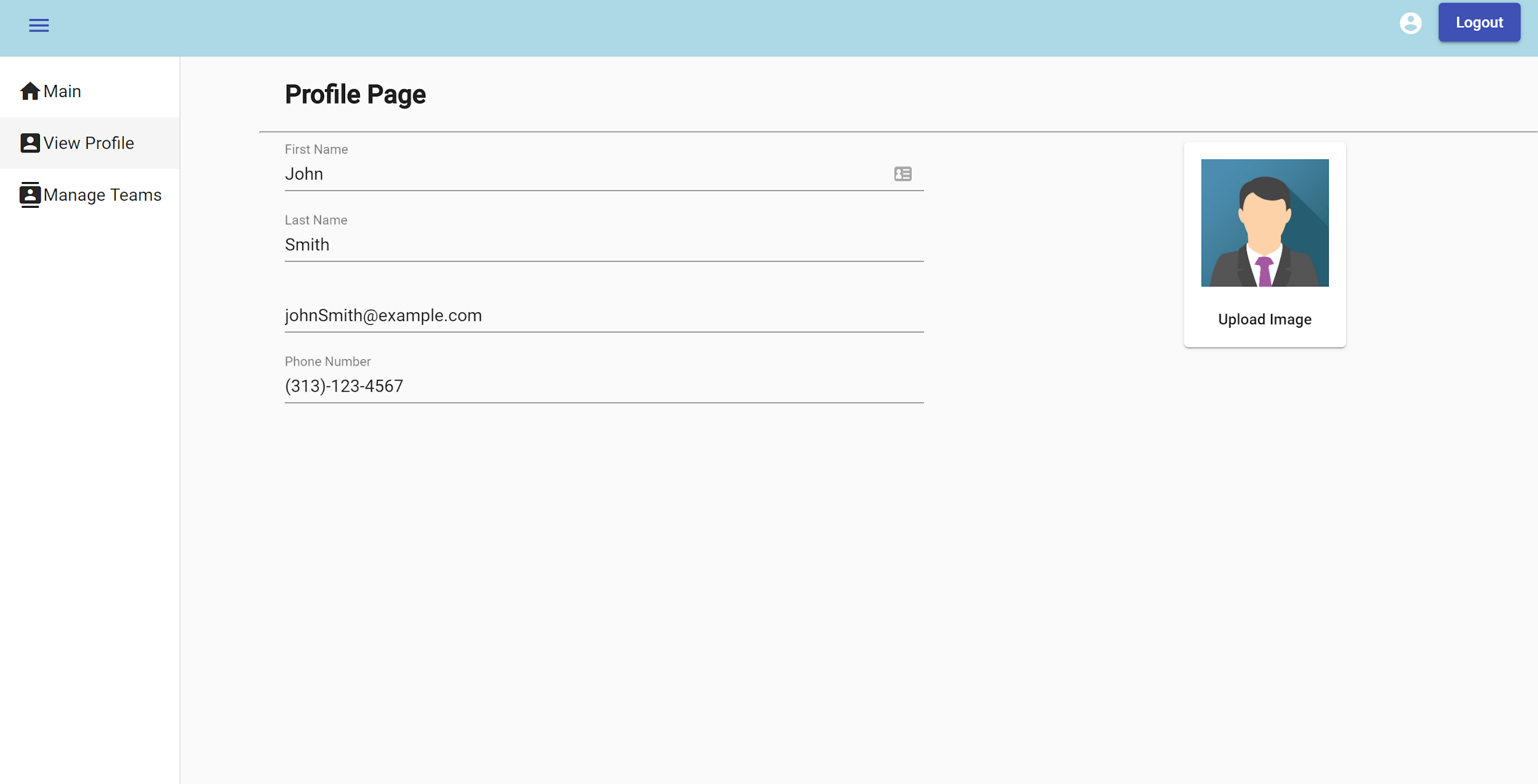
### 4.0 User interface design

The interface for the system is going to be a web application developed using angular framework. Angular is a TypeScript-based open-source web application framework that is used by many enterprises in the industry for its professional look and code reusability. Upon calling the website’s link, the user would be redirected to the login page where he/she would have to enter their credentials. If approved they would be redirected to the main page. There will be 3 main pages as of now. in addition to few sub-pages. the main page would hold information regarding the ranking of the team. and it will hold options that would enable the user to be redirected to the other pages. there will be a separate page to view the user’s account information, and a page that would have detailed data and graphs explaining each team’s ranking.

### 4.1 Description of the user interface

### 4.1.1 Screen images

****

****

### 4.1.2 Objects and actions

1. Login: The Login page would be displayed to the user for authentication. the user must enter the credentials (Username & password) to access the website’s information. the checkbox “remember my login” saves the user’s credentials for fast login. The user has the option “Forget password?” in case the forget their password it would email them a secure link to reset password. After user clicks the login button their credentials would get verified, upon verification they would be redirected to the main page. there will be two different views for main page one for admin and one for employee. the user would be redirected to the appropriate main page automatically.
2. Main page: The main page would have a table containing the ranking of the teams. As an admin user, the user would have the options on the right side nav bar as follows, View profile, Manage Teams and Manage Employees. by clicking on the icons, the user would be redirected to the appropriate page. when the user clicks on a team’s name they would be redirected to a different page which would display a detailed analysis and history of that specific team.
3. Manage Teams: on the Manage teams page, the admin has the ability to add/edit/remove teams. you add a new team by pressing the add team button. the team table would expand by a new row with editable text fields where the user would enter the team’s information. by clicking edit the selected row would turn into an editable text field where the user would enter the updated information and clicks save to save them. Finally, by clicking the remove button, the selected row would be removed from the database.
4. Manage Employees: on the Manage Employees page, the admin has the ability to add/edit/remove Employees. you add a new Employee by pressing the add Employee button. the Employees table would expand by a new row with editable text fields where the user would enter the Employee’s information. by clicking edit the selected row would turn into an editable text field where the user would enter the updated information and clicks save to save them. finally by clicking the remove button, the selected row would be removed from the database.
5. View Profile: The View profile page displays all of the user’s information such as first name, last name, email ….etc. by clicking the edit button the user would be able to edit their information. the user can also upload a picture for their profile.
6. Team Rank History: The team rank history page displays a graph with data points representing the rank of that specific team throughout the weeks. The user will use that as a way to track progress and overall performance to detect progress or regression.

### 4.2 Interface design rules

1. Design rules were mainly focused towards the user. Giving the user a friendly intuitive interface
2. Kept everything consistent through the design, if the header of a row is clickable in one table it must behave the same way for all the other tables in the website. That way the user will not be confused or left guessing on the use of a functionality.

### 4.3 Components available

We are using Angular as a framework and importing many libraries to use the parts of which to complete the design of the website, we will be listing some of them below:-

1. Angular material: Angular material provides a variety of components used as parts injected inside the website’s interface.
   1. Mat-table: mat-table is used for all the tables that are listed on the website
   2. Mat-dialog: all the dialog boxes that popup when user clicks on a feature that require them to select users to be added to a team
   3. Mat-drawer and mat-navbar: used to set up the overall structure of the website placing different page navigation on the left side and having a toolbar at the top displaying important information for the user like logout and the name of the application.
   4. Mat-input
   5. Mat-dropdown
2. Bootstrap: bootstrap is used to provide the graphs and the charts for the website as well as some additional front-end design components.
3. Charts.js: is a library containing many different variety of charts that are minpulatibale and can dynamically changed. We are using the line chart to display ranking for teams.

### 

### 5.0 Restrictions, limitations, and constraints

Restrictions:

The software will need to work with future browser updates meaning that we cannot use 3rd party feature like angular modals and sticking strictly to angular vanilla. That will restrict the front-end design to a more static web pages and with few dynamic features.

Limitations:

We are creating the gamification web-application to fit the client's current software. Using the clients Api we can pull information, but we are limited to how the client’s software is designed. There software was not initially created with this feature in mind. The client only allows you to retrieve job reports for a start date not an end date, the jobReport/tags(assets used on a report) api does not have the tagUUID instead it has a tag RFID meaning that during creation we will have to save both creating an extra unused column. The other IDS for entities work understandably as of now. ~~Ex tags in the clients Api don't belong to any one company, everyone has the same set of tags. but in the gamification website the client can change the weight for the asset which means each company can modify their assets(tag). There are no team IDs or team tables, just reports that hold the team names. We will need to find clever solutions to overcome this initial design flaw.~~

Constraints:

~~Software/database will be used for multiple companies meaning there may be cross over in names. teams’ names and asset names will have to be partially unique, some attribute values must be unique for a company, but other companies may use that name: ex team1 can only be used by ford once but Chevy could also use the name team1 once.~~

database should be sharded, each shard should contain only one company. The shards are completely independent of each other. This will remove previous set constraints but add new ones. The shard manager should direct users and the system to the correct database shard.

Software must pull data from the client Api once a day.

### 6.0 Testing Issues

Test strategy and preliminary test case specification are presented in this section.

### 6.1 Classes of tests

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Type** | **Description** | **Input** |
| 1 | Black Box Testing | Testing log in. | Username: admin  Password: password |
| 2 | Black Box Testing | Testing if admin can create a team. | Admin creates team. |
| 3 | Black Box Testing | Testing if admin can delete team. | Admin deletes team. |
| 4 | Black Box Testing | Testing if admin can add employee to team. | Admin adds employee to team. |
| 5 | Black Box Testing | Testing if admin can remove employee from team. | Admin removes employee from team. |
| 6 | Black Box Testing | Testing if admin can create employee account. | Admin creates employee account. |
| 7 | Black Box Testing | Testing if admin can deactivate employee account. | Admin deactivates employee account. |
| 8 | Black Box Testing | Testing if admin can create a team with the same name. | Admin creates team with same name. |
| 9 | Black Box Testing | Testing if admin can add the same employee to a team twice. | Admin adds the same employee to a team twice. |
| 10 | Black Box Testing | Testing if admin can change metrics. | Admin Changes weight of asset. |
| 11 | Black Box Testing | Testing if user can view team history. | User selects view team history. |
| 12 | Black Box Testing | Testing if user can view weekly ranking. | User selects view weekly ranking. |
| 13 | Black Box Testing | Testing if user can view all time ranking. | User selects view all time ranking. |
| 14 | Black Box Testing | Testing if user can change their password. | User selects change password and confirms new password. |
| 15 | Black Box Testing | Testing if user can use the same password when changing password. | User attempts to change password to existing password. |
| 16 | Black Box Testing | Testing if user can change their account information. | User makes changes to account information. |
| 17 | Black Box Testing | Testing if user can save account information if nothing was changed. | User selects change account information and saves without making any changes. |
| 18 | Black box testing | Test to see if invalid credentials are allowed | Username:invalid  password:invalid |
| 19 | Black box testing | Test to see the result of running the service function when no assets are in the table | N/A |
| 20 | Black box testing | Test to see if deactivated teams show on the leaderboard. | Deactivate team |
| 21 | Black box testing | Test to see if a deleted employee can still log in | Delete employee and then attempt to log in with the credentials |
| 22 | Black box testing | Test to see if job reports are duplicated/recounted into ranking for the week when past 12 and the service runs | Have the service run at 12, then have a jobreport be entered while its running. It should have a timestamp past 12 |

### 6.2 Expected software response

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Expected Output** | **Actual Output** | **Pass/Fail** | **Date Fixed** |
| 1 | User is logged in. |  |  |  |
| 2 | Team is created. Team is saved into database. Admin is now able to edit team. |  |  |  |
| 3 | Team is deleted. Team is removed from database. |  |  |  |
| 4 | Employee is added to team. Changes made to team are saved in database. |  |  |  |
| 5 | Employee is removed from team. Changes made to team are saved in database. |  |  |  |
| 6 | Employee account is created. Credentials are generated for employee. Credentials are saved in database. |  |  |  |
| 7 | Employee account is deactivated. Credentials are revoked. |  |  |  |
| 8 | Account is not created. |  |  |  |
| 9 | Employee is not added. |  |  |  |
| 10 | Weight of asset is changed. Metrics are updated. |  |  |  |
| 11 | Team history is displayed to user. User can view all their ranking history. |  |  |  |
| 12 | Weekly ranking is displayed to user. User can see what ranking position they are in and the ranking of all the other teams. |  |  |  |
| 13 | All time ranking is displayed to the user. user can view their all-time ranking and the all-time ranking of the other teams. |  |  |  |
| 14 | Users password is changed and saved in database. |  |  |  |
| 15 | User password is not changed. |  |  |  |
| 16 | User account information is changed and saved in database. |  |  |  |
| 17 | User account information is changed and saved in database. |  |  |  |
| 18 | Error is displayed saying the the credentials are invalid |  |  |  |
| 19 | The service should not create a rank record unless the total score is greater than 0 |  |  |  |
| 20 | Deactivated teams should show on the archived leaderboard, but not new records |  |  |  |
| 21 | The “deleted” employee should get an error that is the same as invalid credentials |  |  |  |
| 22 | The job reports that come after 12 should not be added into the score of rank but saved for the next day. |  |  |  |

### 6.3 Performance bounds

The login step with the sharded database should work seemingly.

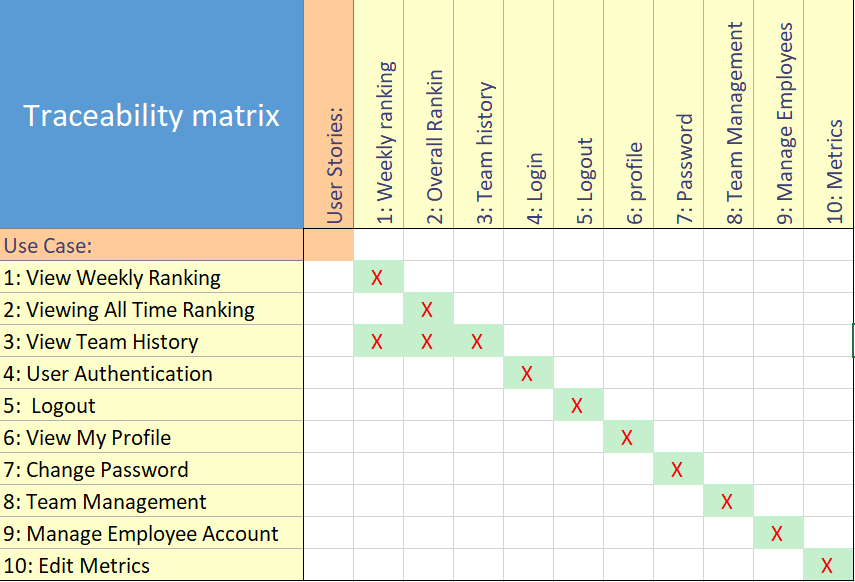
Creating a user account will require the system to connect to 2 databases and insert the rows, and if one fails it should not save in either.

### 6.4 Identification of critical components

1. Service should be tested thoroughly without using the clients API.
2. Team-management should be tested to see if everything works properly with the client API.
3. Assets-management should be tested if everything works properly with the client API.
4. The database has been sharded, this will require to check if creating users works properly with the the partitioned databases. Shard manager will need to be tested for security reasons as well.

### 7.0 Appendices

### 7.1 Requirements traceability matrix

****

### 7.2 Packaging and installation issues

The website would be hosted on an azure server, a service provided by microsoft.

The database would be hosted on an azure server as well.

### 7.3 Design metrics to be used

The Design metrics used are CoCoMo & Function points

### 7.4 Supplementary information (as required)

Client Api:

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

Client Software demonstration:

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

Gamification Definition:

<https://en.wikipedia.org/wiki/Gamification>

Gamification Software:

<https://appsource.microsoft.com/en-us/product/dynamics-365/mscrm.f6d23ec7-255c-4bd8-8c99-dc041d5cb8b3?tab=Overview>

Kpi:

<https://www.google.com/search?q=key+performance+indicators&oq=key+performance+indicators&aqs=chrome..69i57j0l5.456j0j7&sourceid=chrome&ie=UTF-8>

Data Dictionary:

<https://docs.microsoft.com/en-us/sql/t-sql/data-types/decimal-and-numeric-transact-sql?view=sql-server-2017>

<https://docs.microsoft.com/en-us/sql/t-sql/functions/date-and-time-data-types-and-functions-transact-sql?view=sql-server-2017>

[https://docs.microsoft.com/en-us/sql/t-sql/functions/date-and-time-data-types-and-functions-transact-sql?view=sql)-server-2017](https://docs.microsoft.com/en-us/sql/t-sql/functions/date-and-time-data-types-and-functions-transact-sql?view=sql-server-2017)

Database:

<https://docs.microsoft.com/en-us/azure/sql-database/sql-database-single-database-get-started>

serialize image to binary & save to database:

<http://www.nullskull.com/articles/20020929.asp>

<https://www.youtube.com/watch?v=09ftaulKUVM>

Backend:

Web Application:  
 <https://docs.microsoft.com/en-us/visualstudio/ide/quickstart-aspnet-core?view=vs-2017>

Program scheduler (time trigger):

<https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer>

<https://www.quartz-scheduler.net/>

<https://www.thewindowsclub.com/how-to-schedule-batch-file-run-automatically-windows-7>

Login Authentication:

<http://www.primaryobjects.com/2015/05/08/token-based-authentication-for-web-service-apis-in-c-mvc-net/>

REST Guidelines:

<https://tools.ietf.org/html/rfc7231#section-4.3>

Frontend:

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

Picture upload:

<https://www.academind.com/learn/angular/snippets/angular-image-upload-made-easy/>

Graph:

<https://docs.microsoft.com/en-us/power-bi/visuals/power-bi-visualization-types-for-reports-and-q-and-a>

Design patterns used for web application:

MVC:

<https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.2>:

Mediator:

<https://www.dofactory.com/net/mediator-design-pattern>

Facade:

<https://www.dofactory.com/net/facade-design-pattern>

CQRS:

<https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>