ADMIRE M. KHULUMO

19001366

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.   
When you’re ready to add your content, just click here and start typing.]

Research Management System

Software Documentation - COMP 342 Project

Table of Contents

[Abstract 2](#_Toc61741697)

[Introduction 2](#_Toc61741698)

[Database 2](#_Toc61741699)

[Entities 2](#_Toc61741700)

[Entity relationships 4](#_Toc61741701)

[ER Diagram 7](#_Toc61741702)

[WEB API 7](#_Toc61741703)

[Models 7](#_Toc61741704)

[DB Class 9](#_Toc61741705)

[Controllers 9](#_Toc61741706)

[Client Web App 9](#_Toc61741707)

[Models 9](#_Toc61741708)

[APIRequest Class 9](#_Toc61741709)

[Controllers 9](#_Toc61741710)

[Security 9](#_Toc61741711)

[Authentication 9](#_Toc61741712)

[Authorisation 9](#_Toc61741713)

[Report 10](#_Toc61741714)

[Appendices 10](#_Toc61741715)

[Appendix 1: ER DiAgram 10](#_Toc61741716)

# 1Abstract

Stuff

# Introduction

Stuffs

# Database

## Entities

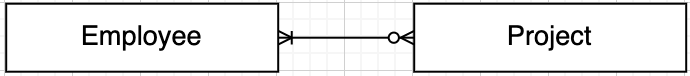
Note: The blue highlight represents an assumption – either the attribute is not specified in the question or its use/meaning was assumed.

|  |  |  |  |
| --- | --- | --- | --- |
| ENTITY | ATTRIBUTE | Data Type | Description |
| Employee – the BIUST employee who conducts research. | empId (PK) | int | Employee ID number |
| fname | varchar | First name - |
| sname | varchar | Surname |
| department | varchar | Department |
| position | varchar | Position held |
| qualification | varchar | Highest qualification and institution obtained from |
| interests | varchar | Research interests/areas |
| phone | varchar | Employee’s work phone number |
| email | varchar | Employee’s email address, same one used to login to the client web app. |
| photo | varchar | Url for the employee’s phot ID |
| Project – the research project undertaken by the employee. | pId (PK) | int | Project ID |
| title | varchar | Project title |
| leader (FK) | int | Project leader, references Employee.empId |
| fsource | varchar | Funding source (BIUST, Private, Self) |
| famount | double | Funding amount |
| BookChapter – a book chapter written by an employee when working on a project. | pId (FK) | int | Project ID – each book chapter belongs to one project. Links a book chapter to the project it belongs to. |
| chTitle (SK) | varchar | Chapter title – secondary key that helps identify a book chapter, not unique. |
| bkTitle (SK) | varchar | Book Title – secondary key that helps identify a book chapter, not unique. |
| publisher | varchar | Book publisher |
| pages | int | Number of pages |
| year | int | Year of publication |
|  |  |  |
| JArticle – a research article published in an accredited journal as part of the project. | pId (FK) | int | Project ID – each journal article belongs to one project. Links a journal article to the project it belongs to. |
| title (SK) | varchar | Article’s title - secondary key that helps identify a JArticle, not unique. |
| jname (SK) | varchar | Name of the journal - secondary key that helps identify a JArticle, not unique. |
| volume | int | Journal volume number |
| issue | int | Journal issue number |
| pages | int | Number of pages |
| year | int | Year of publication |
| CArticle – a research article in accredited conference proceedings as part of the project. | pId (FK) | int | Project ID – each conference article belongs to one project. Links a conference article to the project it belongs to. |
| title (SK) | varchar | Article’s title - secondary key that helps identify a CArticle, not unique. |
| cname (SK) | varchar | Name of conference - secondary key that helps identify a CArticle, not unique. |
| pages | int | Number of pages |
| publisher | varchar | Article’s publisher |
| location | varchar | Location of the conference |
| year | int | Year when the conference was held |
| Editorial – a review activity carried out by an employee. It is not part of a project. | date | varchar | Date of the editorial/review |
| activity (SK) | varchar | Details of review activity - - secondary key that helps identify an editorial, not unique. |
| publication (SK) | varchar | Organisation, journal or conference - - secondary key that helps identify an editorial, not unique. |
| url | varchar | URL for the organisation or publication |
| Funder – an individual providing financial support for the research project. That individual may represent themselves or their organization. | fId | int | Funder’s ID |
| name | varchar | Funder’s full name |
| email | varchar | Funder’s contact email |
| organisation | varchar | Organisation that the funder represents (could be self) |
|  |  |  |

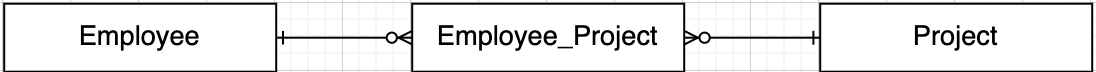
## Entity relationships

1. **Employee and Project**

* An Employee does a Project, and a Project is done by an Employee.
* One Employee can do zero or many Projects, and one Project can be done by one or many Employees. Hence, this forms the many-to-many relationship shown below:

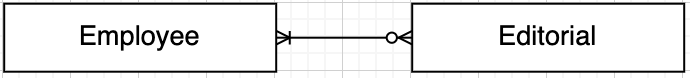


* This can be normalised by using an intermediary table, Employee\_Project containing empId and pId as foreign keys from the Employee and Project tables respectively. This is diagrammed below:

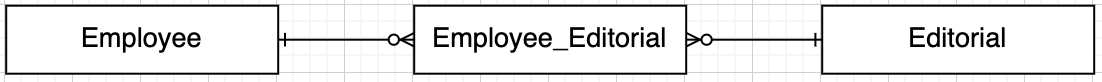


1. **Employee and Editorial**

* An Employee does an Editorial, and an Editorial is done by an Employee.
* One Employee can do zero or many Editorials, and one Editorial can be done by one or many Employees. Hence, this forms the many-to-many relationship shown below:

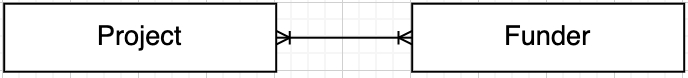


* This can be normalised by using an intermediary table, Employee\_Editorial containing empId and (activity, publication) as foreign keys from the Employee and Editorial tables respectively. This is diagrammed below:

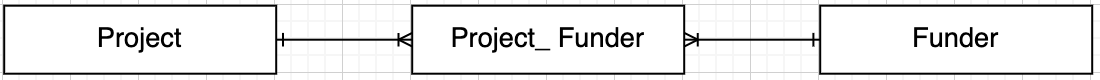


1. **Project and Funder**

* A Project has a Funder, and a Funder funds a Project.
* One Project has at least one or many Funders, and one Funder can fund at least one or many Projects. Hence, this forms the many-to-many relationship shown below:

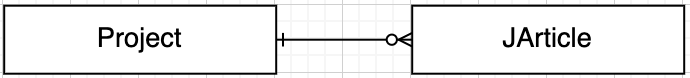


* This can be normalised by using an intermediary table, Project\_Funder containing pId and fId as foreign keys from the Project and Funder tables respectively. This is diagrammed below:



1. **Project and JArticle**

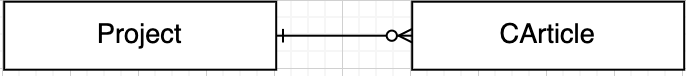
* A Project contains a JArticle, and a JArticle belongs to a Project.
* One Project has zero or many JArticles, and one JArticle belongs to only one project. This forms a one-to-many relationship demonstrated below:



* The JArticle table has a pId field that references the project ID of the Project to which it belongs.

1. **Project and CArticle**

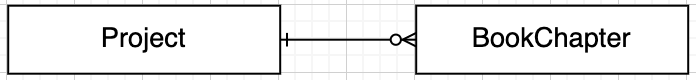
* A Project contains a CArticle, and a CArticle belongs to a Project.
* One Project has zero or many CArticles, and one CArticle belongs to only one project. This forms a one-to-many relationship demonstrated below:



* The CArticle table has a pId field that references the project ID of the Project to which it belongs.

1. **Project and BookChapter**

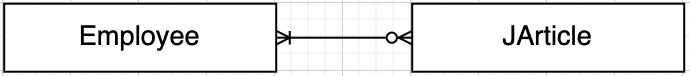
* A Project contains a BookChapter, and a BookChapter belongs to a Project.
* One Project has zero or many BookChapters, and one BookChapter belongs to only one project. This forms a one-to-many relationship demonstrated below:

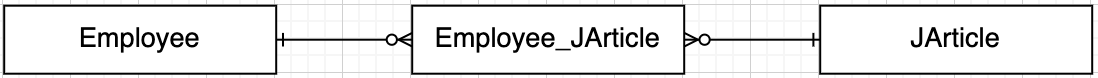


* The BookChapter table has a pId field that references the project ID of the Project to which it belongs.

1. **Employee and JArticle**

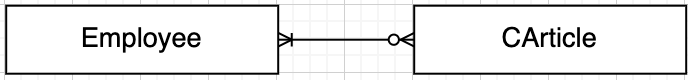
* An Employee authors a JArticle, and a JArticle is authored by an Employee.
* One Employee can author zero or many JArticles, and one JArticle can be authored by at least one or many Employees. Hence, this forms the many-to-many relationship shown below:

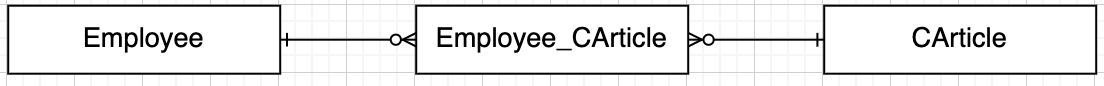


* This can be normalised by using an intermediary table, Employee\_JArticle containing empId and (jname, title) as foreign keys from the Employee and JArticle tables respectively. This is diagrammed below:

1. **Employee and CArticle**

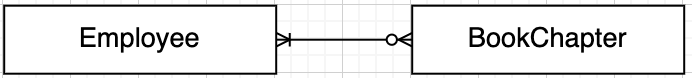
* An Employee authors a CArticle, and a CArticle is authored by an Employee.
* One Employee can author zero or many CArticles, and one CArticle can be authored by at least one or many Employees. Hence, this forms the many-to-many relationship shown below:

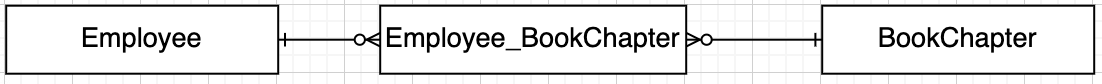


* This can be normalised by using an intermediary table, Employee\_ CArticle containing empId and (cname, title) as foreign keys from the Employee and CArticle tables respectively. This is diagrammed below

1. **Employee and BookChapter**

* An Employee authors a BookChapter, and a BookChapter is authored by an Employee.
* One Employee can author zero or many BookChapters, and one BookChapter can be authored by at least one or many Employees. Hence, this forms the many-to-many relationship shown below:



* This can be normalised by using an intermediary table, Employee\_ BookChapter containing empId and (bkTitle, chTitle) as foreign keys from the Employee and BookChapter tables respectively. This is diagrammed below

## ER Diagram

* See appendix 1.

# WEB API

## Models

* The location of each of these model classes is “RMS/RMS\_API/Models”
* Each file is a .cs class that does not inherit from any other class.
* Each class has:
  + a default constructor,
  + a parameterised constructor,
  + properties with accessors and modifiers for each member variable, and
  + a ToString method used to print the object’s details.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Description | Member Variables |  |  |
|  |  | Identifier | Data Type | Description |
| Employee.cs | Models an employee or research staff member | empId | int | Employee’s unique employee ID |
| fname | string | First name |
| sname | string | Surname |
| department | string | Department in which the employee works |
| position | string | Employee’s role, e.g lecturer, professor etc |
| qualification | string | Highest academic qualification and institution obtained from |
| interests | string | Research interests |
| phone | string | Contact phone number |
| email | string | Contact email address |
| photo | string | Url to employee’s photo ID |
| Project.cs | Models a project undertaken by the employee | pId | int | Project’s unique ID |
| title | string | Title of the project |
| leader | string | Employee ID of the project’s leader |
| famount | double | Funding amount availed to project |
| fsource | string | Source of project’s funding – BIUST, Private or public |
| BookChapter.cs | Models a book chapter written by an employee | pId | int | Project ID to identify the project to which it belongs |
| chTitle | string | Book chapter’s title |
| bkTitle | string | Book’s title |
| publisher | string | Book’s publisher |
| pages | int | Number of pages written |
| year | int | Year of publication |
| CArticle.cs | Models details of a conference article paper written by an employee | pId | int | Project ID to identify the project to which it belongs |
| title | string | Title of the article |
| cname | string | Name of the conference where the article was presented |
| publisher | string | Article’s publisher |
| location | string | Location of the conference |
| pages | int | Number of pages of the article |
| year | int | Year of publication |
| JArticle.cs | Models details of a journal article paper written by an employee | pId | int | Project ID to identify the project to which it belongs |
| title | string | Title of the article |
| jname | string | Name of the journal in which the article was published |
| volume | int | Journal’s volume number |
| issue | int | Journal’s issue number |
| pages | int | Number of pages of the article |
| year | int | Year of publication |
| Editorial.cs | Models an editorial or review done by an employee | activity | string | Details editorial activity |
| publication | string | Organisation/conference/journal article associated with the editorial |
| url | string | URL of the editorial or organisation |
| date | string | Date on which the editorial occurred |
| Funder.cs | Models an individual who funds a project | fId | int | Unique identifier of each funder in the database |
| name | string | Full name of the funder (or representative of the firm) |
| organisation | string | Name of organisation providing funding |
| email | string | Contact email of funder |

## DB Class

* This file is located in “RMS/RMS\_API/DB.cs”
* Purpose: This class is used as a central point to establish and close connections to the database, as well as contain functions that perform CRUD operations against the database. These functions are described below.
* The advantage here is that there is no need to re-establish a connection to the database in each controller action/method every time. When querying, an object of this DB class is instantiated and the relevant function is called using the dot (.) operator. For example:

DB db = new DB();

List<Employee> employees = db.getEmployees();

### Instance Variables

* All instance variables are private to the class.

|  |  |  |
| --- | --- | --- |
| Type | Identifier | Description |
| string | sql | Used to store an SQL string. Prevents having to re-declare in each function. |
| MySqlConnection | conn | Database connection object for establishing a connection to the database. |
| MySqlCommand | cmd | Database command object for running queries against the connection. |

### Functions

* Unless stated otherwise, all functions are public.
* Each function opens/closes the connection individually as needed.
* All database related code is enclosed in a try-catch block that catches and reports any SQL exceptions.
* Any programming statements are displayed in blue text.
* For functions that return a Boolean value, assume that true is returned when the operation is successful, and false when the operation is not successful.
* Optional parameters are shown with an assignment operator denoting the default value as follows: email = “”
* Detailed information on the code of each function is specified as comments.
* No deleteEmployee() method exists as the purpose of this project is to manage research projects not employees.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | IDENTIFIER | RETURN TYPE | PARAMETERS | | FUNCTIONALITY |
| **TYPE** | **IDENTIFIER** |
| 1. | DB | None | None | - | ⚬ Class constructor  ⚬ Initialises the connection object using a connection string. |
| 2. | openConn | bool | None | - | ⚬ Tries to open a connection to the database by running conn.Open()  ⚬ Returns true if connection is opened successfully, otherwise false. |
| 3. | closeConn | bool | None | - | ⚬ Tries to close the database connection by running conn.Close()  ⚬ Returns true if connection is closed successfully, otherwise false. |
| **EMPLOYEE** | | | | | |
| 4. | getEmployees | List<Employee> | string | email = “” | ⚬ Returns a list of all employees in the database if no arguments are passed.  ⚬ Returns a list with a single Employee object if email or empId are passed. This is the employee matching the value passed. Empty list is returned if no match is found. |
| int | empId = 0 |
| 5. | addEmployee | bool | Employee | e | ⚬ Adds an employee record to the employee table in the database. |
| 6. | editEmployee | bool | Employee | e | ⚬ Edits employee details on the database, except for empId which is a primary key. |
| 7. | getAuthors | List<Employee> | string | type | ⚬ Returns a list of authors of a publication. An author is of Employee type.  ⚬ type specifies the type of publication as jArticle, cArticle or bookChapter  ⚬ id1 and id2 represent the two keys that make up the primary key of the publication. |
| string | id1 |
| string | id2 |
| 8. | addAuthor | bool | int | empId | ⚬ Associates an author to a publication.  ⚬ empId is used to identify the employee.  ⚬ type, id1 and id2 are the same as above. |
| string | type |
| string | id1 |
| string | id2 |
| 9. | getEditors | List<Employee> | string | activity | ⚬ Returns employees who are editors/reviewers of an editorial  ⚬ activity and publication are used to uniquely identify the editorial |
| string | publication |
| **PROJECT** | | | | | |
| 10. | getProjects | List<Project> | int | empId = 0 | ⚬ Returns a list of all projects in the database.  ⚬ If empId is passed, all projects by a certain employee are returned. Might be empty if employee has done no projects. |
| 11. | addProject | bool | Project | p | ⚬ Adds a project into the database.  ⚬ empId is used to associate the project to the employee doing it.  ⚬ fId is used to associate the first project funder to that project. |
| int | empId |
| int | fId |
| 12. | editProject | bool | Project | p | ⚬ Edits a project’s details. |
| 13. | deleteProject | bool | int | pId | ⚬ Deletes a project from the project table in the database.  ⚬ Associations in the employee\_project table are also deleted. |
| **JOURNAL ARTICLE** | | | | | |
| 14. | getJArticles | List<JArticle> | int | pId = 0 | ⚬ Returns a list of all journal articles in the database.  ⚬ If pId is supplied, list of journal articles belonging to a certain project is returned. List can be empty. |
| 15. | addJArticle | bool | JArticle | ja | ⚬ Adds a journal article into the database. |
| 16. | editJArticle | bool | JArticle | ja | ⚬ Edits a journal article’s details in the database.  ⚬ old\_title and old\_jname are used to identify the existing record before changing to the new values.  ⚬ Edits are also applied on the employee\_jArticle table. |
| string | old\_jname |
| string | old\_title |
| 17. | deleteJArticle | bool | string | jname | ⚬ Deletes a journal article from the database.  ⚬ Author associations are also deleted on the employee\_jArticle table.  ⚬ jname and title are used to uniquely identify the journal article. |
| string | title |
| **CONFERENCE ARTICLE** | | | | | |
| 18. | getCArticles | List<CArticle> | int | pId = 0 | ⚬ Returns a list of all conference articles in the database.  ⚬ If pId is supplied, list of conference articles belonging to a certain project is returned. List can be empty. |
| 19. | addCArticle | bool | CArticle | ca | ⚬ Adds a conference article into the database. |
| 20. | editCArticle | bool | CArticle | ca | ⚬ Edits a journal article’s details in the database.  ⚬ old\_title and old\_cname are used to identify the existing record before changing to the new values.  ⚬ Edits are also applied on the employee\_cArticle table. |
| string | old\_cname |
| string | old\_title |
| 21. | deleteCArticle | bool | string | cname | ⚬ Deletes a conference article from the database.  ⚬ Author associations are also deleted on the employee\_cArticle table.  ⚬ cname and title are used to uniquely identify the conference article. |
| string | title |
| **BOOK CHAPTER** | | | | | |
| 22. | getBookChapters | List<BookChapter> | int | pId = 0 | ⚬ Returns a list of all book chapters in the database.  ⚬ If pId is supplied, a list of book chapters belonging to a certain project is returned. List can be empty. |
| 23. | addBookChapter | bool | BookChapter | bk | ⚬ Adds a book chapter into the database. |
| 24. | editBookChapter | bool | BookChapter | bk | ⚬ Edits a book chapter’s details in the database.  ⚬ old\_chTitle and old\_bkTitle are used to identify the existing record before changing to the new values.  ⚬ Edits are also applied on the employee\_bookChapter table. |
| string | old\_bkTitle |
| string | old\_chTitle |
| 25. | deleteBookChapter | bool | string | bkTitle | ⚬ Deletes a book chapter from the database.  ⚬ Author associations are also deleted on the employee\_bookChapter table.  ⚬ bkTitle and chTitle are used to uniquely identify the book chapter. |
| string | chTitle |
| **EDITORIAL** | | | | | |
| 26. | getEditorials | List<Editorial> | int | empId = 0 | ⚬ Returns a list of all editorials in the database.  ⚬ If empId is supplied, a list of editorials done by that employee is returned. List can be empty. |
| 27. | addEditorial | bool | Editorial | ed | ⚬ Adds an editorial into the database.  ⚬ Association is also added on the employee\_editorial table to denote the first employee who did that editorial. addEditor can be called to add more employees as editors. |
| int | empId |
| 28. | editEditorial | bool | Editorial | ed | ⚬ Edits an editorial’s details in the database.  ⚬ old\_activity and old\_publication are used to identify the existing record before changing to the new values.  ⚬ Edits are also applied on the employee\_editorial table. |
| string | old\_activity |
| string | old\_publication |
| 29. | deleteEditorial | bool | string | activity | ⚬ Deletes an editorial from the database.  ⚬ Author associations are also deleted on the employee\_editorial table.  ⚬ old\_activity and old\_publication are used to uniquely identify the editorial. |
| string | publication |
| **FUNDER** | | | | | |
| 30. | getFunders | List<Funder> | int | pId = 0 | ⚬ Returns a list of all funders in the database.  ⚬ If pId is supplied, a list of funders of that project is returned. List can be empty. |
| 31. | addFunder | bool | Funder | fu | ⚬ Adds a funder into the database.  ⚬ pId is used to associate the funder to that project in the project\_funder table. |
| int | pId |
| 32. | editFunder | bool | Funder | fu | ⚬ Edits a funder’s details on the database. |
| 33. | deleteFunder | bool | int | pId | ⚬ Deletes a funder from a project by removing the association in the project\_funder table.  ⚬ pId is used to identify the project, and fId identifies the funder to remove. |
| int | fId |

## Controllers

* Each of the above models has its own controller. They can be found at “RMS/RMS\_API/Controllers”
* Controllers and their actions map directly to the URL routes. These, and sample queries, have been compiled through SwaggerUI (API testing tool) and attached under appendix 2.
* The detailed code and explanation of each controller is found in the code files.

# Client Web App

## Models

* All models are exactly the same as in the API described above.
* A few models have some additional members, and these are listed below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Description | Member |  |  |
|  |  | Identifier | Data Type | Description |
| Employee.cs | Models an employee or research staff member | user | Employee | Holds details of currently logged in user. |
| isAdmin() | - | Static method used to check if the logged in user is an admin or not. Returns true if user is an admin. |
| Project.cs | Models a project undertaken by the employee | leadDetails | Employee | Holds details of the project leader, who is also an employee. |
| bookChapters | List<BookChapter> | List of book chapters that belong to that project. |
| cArticles | List<CArticle> | List of conference articles that belong to that project. |
| jArticles | List<JArticle> | List of journal articles that belong to that project. |
| funders | List<Funder> | List of funders funding the project. |
| Editorial.cs | Models an editorial/review done by an employee | doers | List<Employee> | List of employees who did that editorial. |

## APIRequest Class

* This class is used to make API calls to the API endpoints detailed in the previous section.
* All methods match with those described in the DB.cs class of the API, with the same parameters, names and return types.
* Each method uses a different API endpoint to either create, read, update or delete data from the database through the API.
* try-catch blocks were used to catch any errors when forming requests through the HttpClient object.

## Controllers

## Security

### Authentication

* Form-based authentication was used.
* Except for the Login and Register pages, no other pages are accessible to anyone not logged in.
* User creates an account and their credentials are stored by the .NET environment.
* Authentication Procedure:
  1. User supplies credentials through the login page, and Forms authenticates the user.
  2. When user successfully logs in, if the user is not an admin user, his/her employee details are retrieved from the database.
  3. User’s details are stored in the static Employee user variable in the Employee model, and hence are available throughout the application (mimics a session).
* Each controller is tagged with [Microsoft.AspNetCore.Authorization] to ensure that if a user types in the URL in the address bar, they cannot bypass the Forms authentication.

### Authorisation

* A simple role-based authorisation method was implemented.
* Two roles exist:
  1. Normal Employee – staff member who manages their research work.
  2. Admin – user who is able to manage employees, view high-level data and create reports.
* Authorisation Procedure:
  1. When a user successfully logs in, their email, at least, is stored in the user variable in the Employee model.
  2. When user requests a certain page, the static method isAdmin() is used to determine if they are admin or not, and depending on the result they are either allowed to access the page, or redirected accordingly. All this is performed through the relevant controller.
* Admin Pages: URL is RMS/RMS\_Client/Views/{Controller}/{View}
  + Employee/Index.cshtml
  + Employee/AddEmployee.cshtml
  + Project/Index.cshtml
  + Report/Index.cshtml
* All other pages are for a normal employee.

## Report

* sections and outputs

# Appendices

## Appendix 1: ER Diagram

* See attached file titled “RMS\_ER\_DIAGRAM.pdf”

## Appendix 2: API Controllers and Routes

* Each controller maps directly to the routes via the controller actions. These have been compiled via SwaggerUI.
* See attached file titled “RMS\_API\_Routes.pdf” in the root directory.