**EAD 2022 – CA2 Mini Project – May 12th**

*Project Title: Chemical Management MVC*

**Team:**

*Johnathan Sheehan - X00083305*

*Martyna Pacula - X00118313*

**Project Links to Website and Source Code**

|  |  |
| --- | --- |
| Azure Website | <https://chemical-management.azurewebsites.net/> |
| MVC Project Repo (Source) | <https://github.com/MartynaP14/Chemical-Management> |
| Client Code Repo | <https://github.com/JohnnySheehan/Chemical-Management-Client> |

**Project Goals and Functionality**

* To create an MVC ASP.NET Core Application to allow CRUD operations across Lab, Reagent and Reagent Supply models, to form a chemical management system.
* The chemical management system can only be used by authorized and authenticated users.
* The App should be deployed on an Azure App Service and thus should be accessible to the users via a client.
* The App will host and maintain its data in a cloud database (Azure SQL).

**Table of Contents**

1. Project Plan: Pages 1 – 2
2. Database Design: Pages 2 - 6
3. URI Addressing Scheme: Page 7 - 8
4. Unit Testing: Page 8
5. MVC App and Console App Screenshots: Page 9 - 12
6. **Project Plan:**

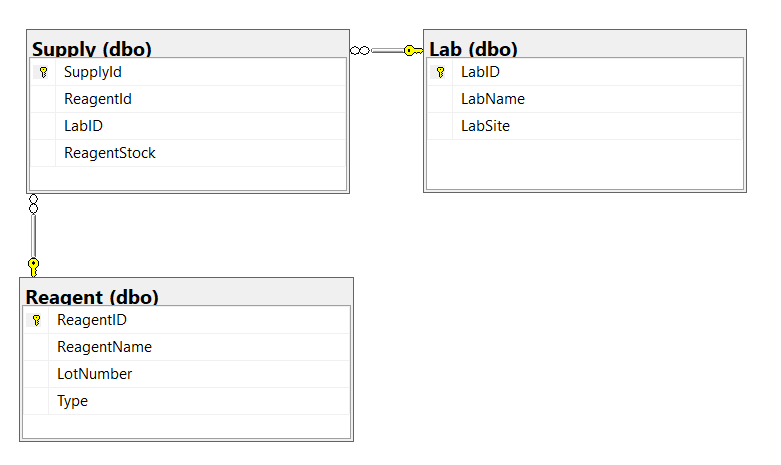
For CA2, we agreed to adopt an iterative software development lifecycle. During Scrum meetings, we would do planning around our sprint tasks. The goal of this was to thoroughly understand how to represent our requirements in code before starting development.

After coding, we would user test the code representing our requirements and take the learnings from our testing to refactor our code. This helped us adjust and in one case refactor our project scope to achieve a successful running ASP.NET Core MVC app for a chemical management system.

|  |  |  |
| --- | --- | --- |
| *Development Schedule* | | |
| Scrums | Sprint Tasks | Outcome |
| Team Creation: 04/04/22  Scrum 1: 08/04/22  Scrum 2: 15/04/22  Scrum 3: 22/04/22  Scrum 4: 26/04/22  Scrum 5: 30/04/22  Scrum 6: 02/05/22  Scrum 7: 07/05/22  Scrum 8: 11/05/22 | * Project Proposal * Set Up GitHub Repo * Create Models * Create Controllers * User test Models * User Test Controllers * Set Up Azure Resource Group * Add SQL Server to Resource Group with shared IPs * Request permission to refactor project scope * Refactor project (Models, Controllers, Views) * Add Cookie based Login Service * User test new MVC * Drop ‘Type’ MVC + DB set * Final MVC Development Iteration * Move from Local DBs to Production Database on Azure SQL Server. * Deploy App Service & Publish Project * Bootstrap CSS theme and Navbar * Unit Testing * Project Documentation * RESTFUL Web Service and Http Client Console App * Seed Data via SSMS * Add Icons, Pictures, Carousel. * DEMO Practice * Final Front-End Clean Up * Final Publish to App Service * Finalise Documentation | Team Assigned by Dermot  Granted by Dermot: Scope meets requirements  Completed  Completed  Completed  Completed  Completed  Completed  Completed  Granted by Dermot  Refactor Completed **(Milestone: ASP.NET Core App completed locally)**  Completed  Completed  Completed  Completed  Completed **(Milestone: Azure SQL Database with three tables)**  Completed **(Milestone: Website hosted in Azure cloud)**  Completed  Completed **(Milestone: Unit Testing Completed)**  Ongoing  Completed **(Milestone: RESTFUL Web Service and Client)**  Completed  Completed **(Milestone: Nice intuitive front-end for website)**  Completed  Completed  Completed **(Milestone: Finalised ASP.NET Core App Completed)**  Completed **(Milestone: Doc Completed)** |

1. **Database Design**

*Below is a diagram of our database captured from SSMS.*



*Our Database Logic*

The Reagent table is needed to hold information (properties) about this object.

The Supply Table is needed, as it is a child table of Reagent and Lab but has the unique property of ‘Reagent Stock.’ This breaks a many-to-many two-table relationship (Reagent-Lab) into a three-table relationship consisting of a one-to-many relationship between Reagent (1) to Supply (M) and a many-to-one relationship between Supply (M) and Lab (1). Therefore, each reagent can have many supplies and many Supplies are stored in a Lab.

*Code First Design (Entity Framework Core)*

To help our go-to object relationship mapper (EF Core) understand our models and their relationships, we need to add constraint properties to certain models. These properties represent Foreign Keys.

These Foreign Key act as a reference to a parent table/model within a child table/model. In our project, the Supply model needed to contain a reference to both the Reagent and the Lab objects to create the ‘one’ side of the entity relationship. This was completed by adding a virtual class reference and Foreign Key Identifiers for both Reagent and Lab.

Likewise, our Reagent and Lab classes needed an ‘ICollection’ reference for Supply to model the ‘many’ sides of the relationship.

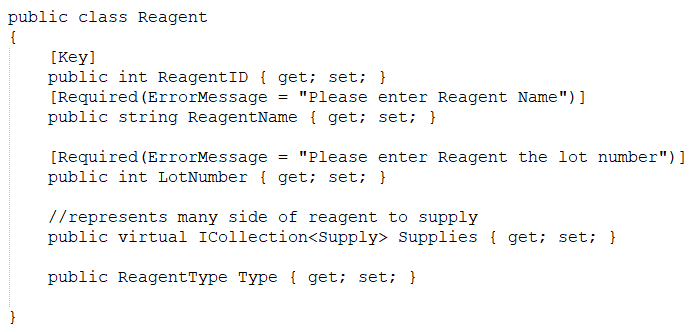
*Database Constraints*

Our models incorporated Primary Keys as unique identifiers for each model (INTs that increment in 1) and Foreign Keys to link related tables.

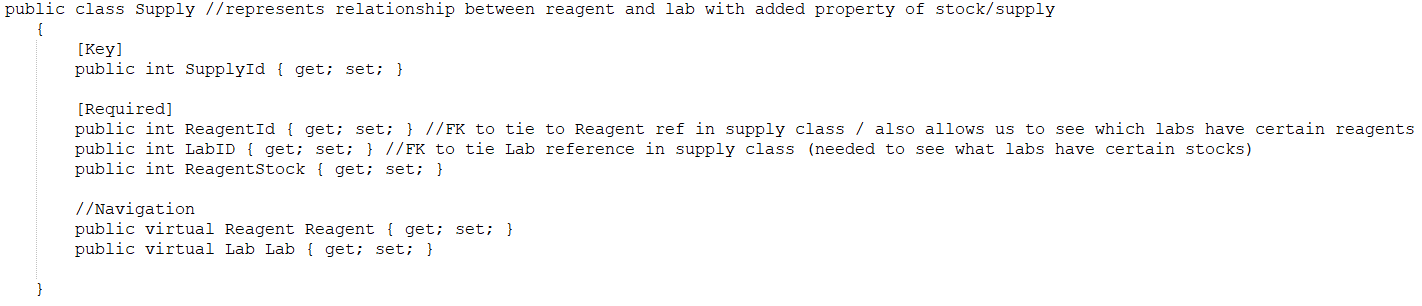
Using Data Annotations from the component model data annotation namespace allowed us to enforce required entry for model properties. This allowed us to make model properties non-nullable meaning that user input is required for the creation of the three object types in our models. This solves the challenge of our model relationship design for Supply, as it must have a reagent and lab to be constructed (Foreign Key enforcement).

*Model Code*

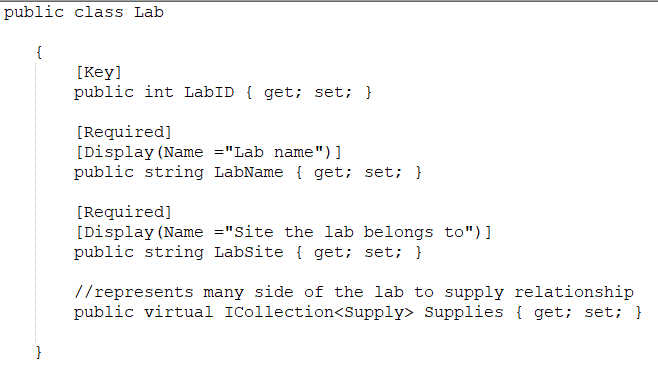
Reagent Model:



Supply Model:



Lab Model:



*Adding Tables and Data Seeding our Database*

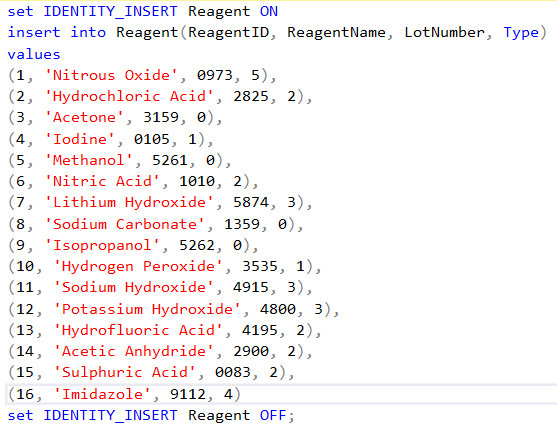
We decided to create our models using a code-first approach. The database tables for each model were created by using a database context and model database sets using EF Core.

This code first method gave us the initial benefit of database version control (locally) while testing our models in the early stages of the project design. In hindsight, this was key to the outcome of our final database design, as we refactored our model design to a more refined yet interconnected one.

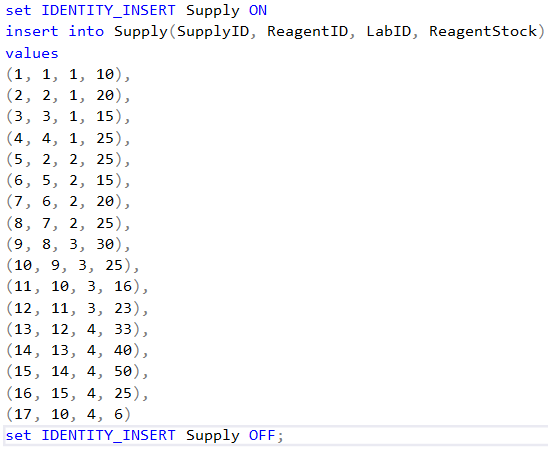
Although, we created our database with a code first approach; for our database seeding, we decided to add data to our models using SQL statements in SQL Server Management Studio. Our logic behind this decision was to prove our database context class within our application was working correctly (executing the SQL statements in SSMS to see the data appear on web application views).

Another reason we chose to add data manually via SQL, is that after testing on local databases, we figured by the very nature of the project (two programmers working on separate project instances, at the same time, pulled from GitHub, and both are editing and seeding data to one cloud database) that it could cause concurrency issues. For example, programmer one executes an edit in the web app and programmer two re-seeds the data and updates before programmer one’s changes are written to the database. *However, concurrency could still happen as ANSI -> Set Implicit\_Transactions is set to off automatically in Azure SQL Server and no locks were added (therefore, more than one transaction per connection on the same data item could happen).*

Reagents Table Insert:



Supply Table Insert: Lab Table Insert:



Text, letter

Description automatically generated

1. **URI Addressing Scheme**

*To test URI, please log in with* Username: Dermot & Password: Dermot2022!

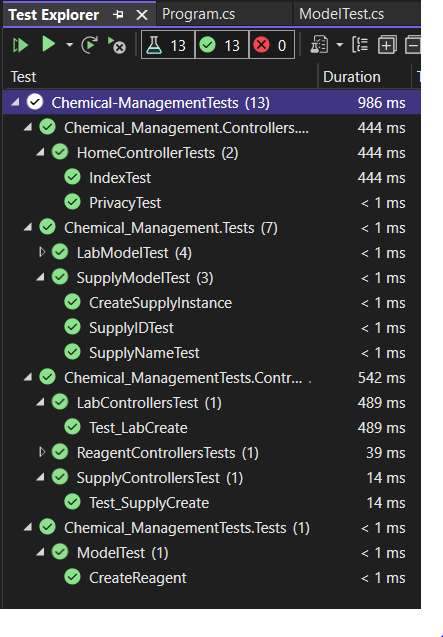
*The App uses cookie-based authentication, so options will not appear if not logged in.*

|  |  |  |
| --- | --- | --- |
| *URI Addressing Scheme* | | |
| Model / Action | Description | URI |
| User / Login / POST | Login page for hard-coded users | <https://chemical-management.azurewebsites.net/Login> |
| Reagent / GET | Returns All Reagents | <https://chemical-management.azurewebsites.net/Reagents> |
| Reagent / POST | Create Reagent | <https://chemical-management.azurewebsites.net/Reagents/Create> |
| Reagent / POST | Edit Reagent (ID = 1) | <https://chemical-management.azurewebsites.net/Reagents/Edit/1> |
| Reagent / GET | Returns details for single Reagent (ID = 1) | <https://chemical-management.azurewebsites.net/Reagents/Details/1> |
| Reagent / DELETE | Deletes Reagent (ID = 1) | <https://chemical-management.azurewebsites.net/Reagents/Delete/1> |
| Lab / GET | Returns All Labs | [https://chemical-management.azurewebsites.net/Labs](https://chemical-management.azurewebsites.net/Labs/Create) |
| Lab / POST | Create Lab | <https://chemical-management.azurewebsites.net/Labs/Create> |
| Lab / POST | Edit Lab (ID = 1) | <https://chemical-management.azurewebsites.net/Labs/Edit/1> |
| Lab / GET | Returns details for single Lab (ID = 1) | <https://chemical-management.azurewebsites.net/Labs/Details/1> |
| Lab / DELETE | Deletes Lab (ID = 1) | <https://chemical-management.azurewebsites.net/Labs/Delete/1> |
| Supply / GET | Returns all Supplies | <https://chemical-management.azurewebsites.net/Supplies> |
| Supply / POST | Creates Supply | <https://chemical-management.azurewebsites.net/Supplies/Create> |
| Supply / POST | Edit Supply (ID = 1) | <https://chemical-management.azurewebsites.net/Supplies/Edit/1> |
| Supply / GET | Returns details for single Supply (ID = 1) | <https://chemical-management.azurewebsites.net/Supplies/Details/1> |
| Supply / DELETE | Deletes Supply (ID = 1) | <https://chemical-management.azurewebsites.net/Labs/Delete/1> |

1. **Unit Testing:**

For Unit Testing, we decided the best strategy was to focus on our models and controllers. For the models, we tested using Assert to see if the models behaved in an expected manner. The first asserts given to our models were for the purpose of proving they (the models) were instances of their types, and the following asserts were used to prove their properties matched with identical types and values given.

For controllers, we tested their ‘create’ functionality by creating a new instance of the database context from an in-memory database and then passing this to a new model controller instance. A new model instance was created and added to the in-memory database via the controller’s ‘create’ function and stored in a variable. This variable was then tested to assert whether it was null or not, to prove the ‘create’ function in the controller. All tests created passed.



1. Application and Console Http Client Screenshots

Chemical Management Application

Home Page:

A person standing in front of a computer screen

Description automatically generated with low confidence

A group of beakers with liquid in them

Description automatically generated with low confidence

Login Page:

A picture containing text, person, electronics, screenshot

Description automatically generated

Once Logged In (options available in navbar):

A picture containing text, indoor, hospital room, screenshot

Description automatically generated

Reagents Page:

A picture containing table

Description automatically generated

Reagent Edit: Reagent Details:

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Reagent Delete: Reagent Create:

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, application

Description automatically generated

Labs Page:

A screenshot of a computer screen

Description automatically generated

Labs Edit: Lab Details:

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Lab Delete: Lab Create:

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, application

Description automatically generated

Supply Page:

A screenshot of a computer

Description automatically generated with medium confidence

Supply Edit: Supply Details:

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Supply Delete: Supply Create:

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application

Description automatically generated

User Management (no data, as users are hardcoded in the asp.net core project)

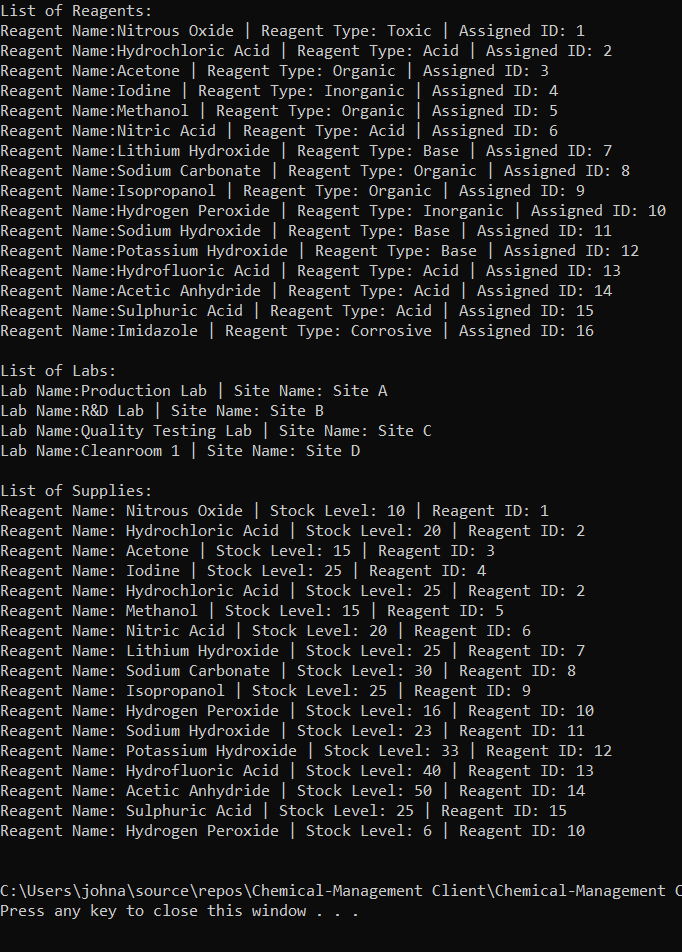
Graphical user interface, text, application

Description automatically generated

Console App output from RESTFUL web Service Interaction:

Http Client calling api/[Controller]

Three GET calls.



**Helpful Links**

*Cookie-Based Auth: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie?view=aspnetcore-6.0*

*Carousel: https://fontawesome.com/docs/apis/javascript/configuration*

*CSS Bootstrap:* https://bootswatch.com/superhero/