# WebAfrica

# June 09, 2022

### Business Requirements

1. Requirements for the WebAfrica api are as follows:
   1. Add employee to the system through web page and console.
   2. List projects in web page
   3. Add additional value to the cost of the project based on the employee job title
   4. Read a json text file. Deserialize and bulk insert into sql server

### Target Audience

WebAfrica hiring manager

### Name of project

WebAfrica Employee and Project Management

### Technical Specs

1. Server-Side Language: C#
2. Framework: .net Core 5.0
3. Unit test: xUnit
4. IDE: Visual Studio 2019
5. Frontend – Angular 12
6. UI CSS Toolkit: Bootstrap / PrimeNg

Solution

The WebAfrica solution consists of several projects with the goal of providing a clean architecture. This makes testing and maintenance easier as different teams can work on different projects.

### Project 1 – WA.Api

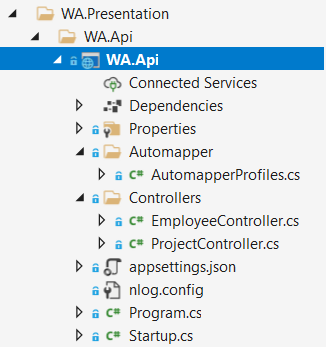


Fig. 1 - WA.Api

This project consists of the Startup.cs file and the Project and Employee Controllers.

**Project controller** - This controller serves as a way of displaying a list of projects. Additional functionality is added by the IRecalc and IEmployeeMappingService to calculate the cost of the project based on the employee’s job title. This is implemented by using interfaces for loose coupling. The logic of this functionality can be unit tested and also swopped for another implementation/

**Employee controller** - This controller is used to add an employee to the database.

The CRUD functionality is performed by delegating the logic to services. This allows for a small controller footprint and reuse of code. Two additional patterns could be followed:

1. Repository pattern – this pattern allows loose coupling and cleaner code.
2. CQRS – for larger applications, the cqrs pattern can be used which separates the crud operation into commands and queries. These then be further optimized.

### Project 2. WA.Contracts

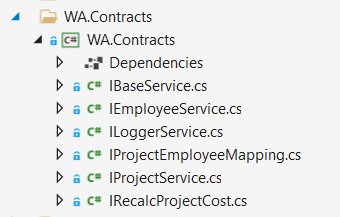


Fig 2. WA.Contracts

Four contracts are used in this project.

1. IBaseService is implemented as a base interface for all common operations shared between all the services.
2. IEmployeeService is implemented as a service to add an employee to the database. Implements the IBaseService.
3. ILoggerService is used to implement the NLog logger package.
4. IProjectEmployeeMapping is implemented as a helper service. It is injected into the ProjectService to facilitate the additional costing based on the job title requirement. It has one method only and it is to map the query results from the database to the ProjectEmployeeDto. Please note that the IRecalcService is injected into the IProjectEmployeeMapping to facilitate the actual calculation. This allows for easy maintenance and testing.
5. IProjectService is implemented as a service to allow the query and display of projects in both the WA.Api and WA.ConsoleOutput.
6. IRecalcProjectCost is implemented as a helper service with the only function to calculate the additional cost based on the employee job title and is injected into the ProjectEmployeeMapping. The concrete implemented can be easily swopped out for another implementation. See point 4.

These interfaces are injected through dependency injection and makes it easier to perform testing. Implementation can be easily swopped out just by defining another concrete implementation in the DI container.

### Project 3 - WA.IoC

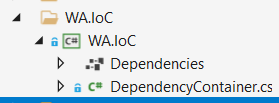


Fig 3. WA.IoC

The Di project registers all the interfaces for dependency injection. This function is normally done inside the Startup.cs class but externalizing this into a separate project, it keeps the Startup.cs cleaner.

### Project 4 - WA.Services

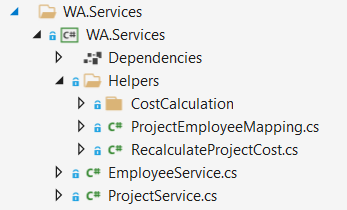


Fig 4. WA.Service

The service project is used to separate the different functions into separate logical components. The service classes implement the corresponding interfaces in WA.Contracts project which is then injected into the controllers. See project two above for a full description of the services.

### Project 5 - WA.Test

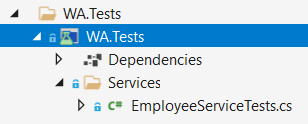


Fig 5. WA.Test

The last project is an Xunit test project for running unit tests.

### Project 6 – WA.Infrastructure

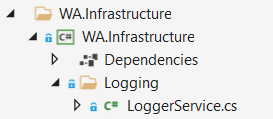


Fig 6 – WA.Infrastructure

This project includes all the cross-cutting concerns that are shared across the solution. As noted previously, the LoggerService is implemented by using NLog but can be changed to use a custom solution. The email service is used by the CartService to send payment confirmation email to the user but can also be used from any other component for a different purpose.

### Project 7 – WA.Common

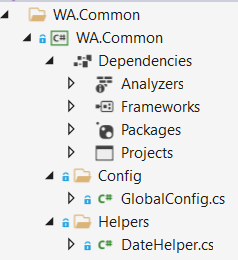


Fig 7. WA.Common

WA.Common contains different functionality that can also be used throughout the solution such as JWT configuration, any common to all projects helper classes.

Project 8

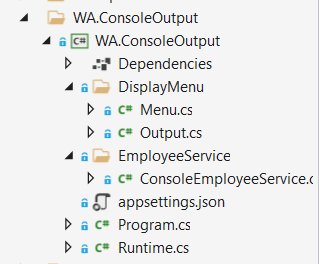


Fig 8 WA.ConsoleOutput

This project is as the name suggests, a console application where the user can type in the employee details and it is saved to the database using the same service as the WA.Api project.

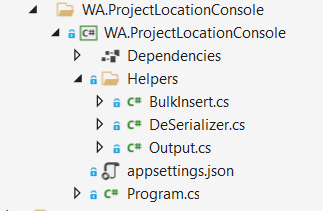


Fig 9 WA.ProjectLocationConsole

This project is used to perform the bulk insert functionality and consists of 3 classes. The Output class displays the results to the console. The Deserialize class assists with reading the json file and transforming the content into classes. This is then taken by the Bulkinsert class to insert it into the database.

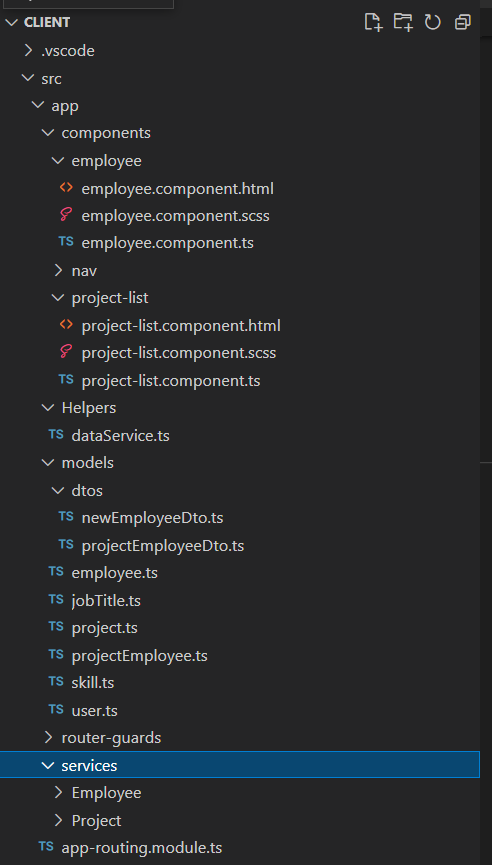


Fig 10 – Angular project structure.

As mentioned, the frontend is done in Angular 12. No specific version 12 commands were used so this should work in earlier versions of angular.

The Fig 10 shows the logical structure of the frontend. Logic is kept to the minimum and only consists of components, services and models.

1. Components: This houses the different components used in the project.
2. Services: there are two services in the Services folder that communicates with the WA.Api project by sending data in the form of parameters.
3. Models: contains the entities used in the project.

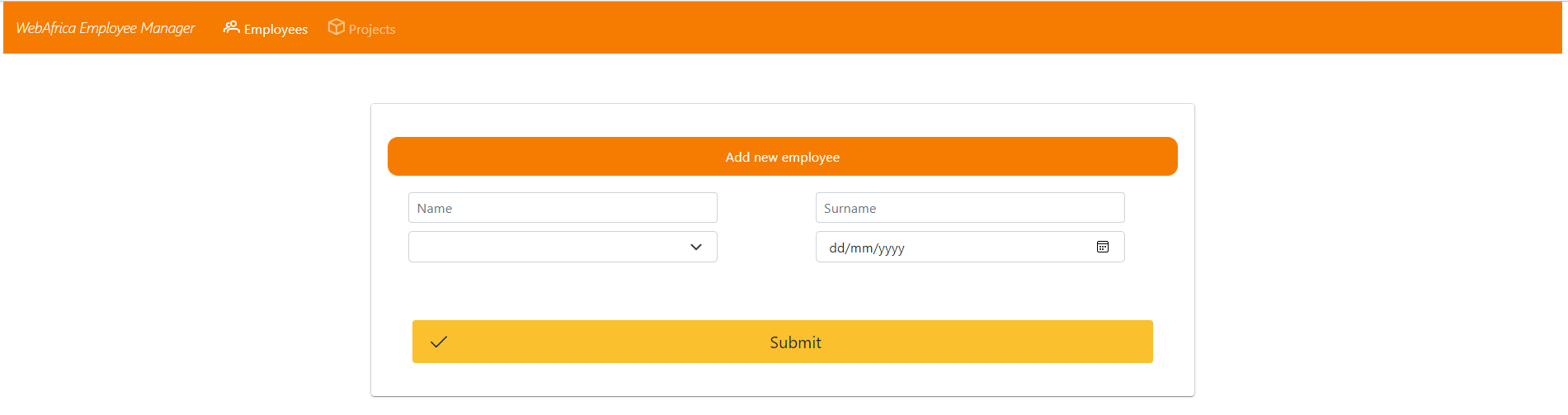


Fig 11. New Employee

Fig 11 shows the view that users can use to log into the application. Communication is asynchronous.

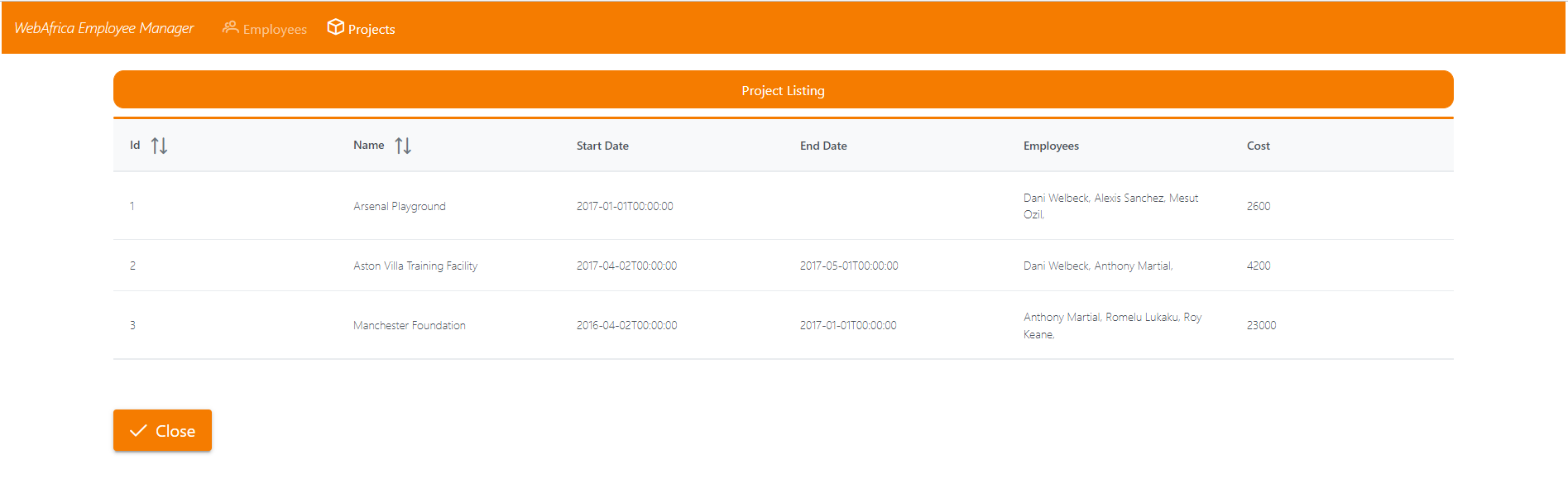


Fig 12. Project listing

Fig 12 displays the projects with the additional costs calculated as well as grouped listing with all the employees as a string.

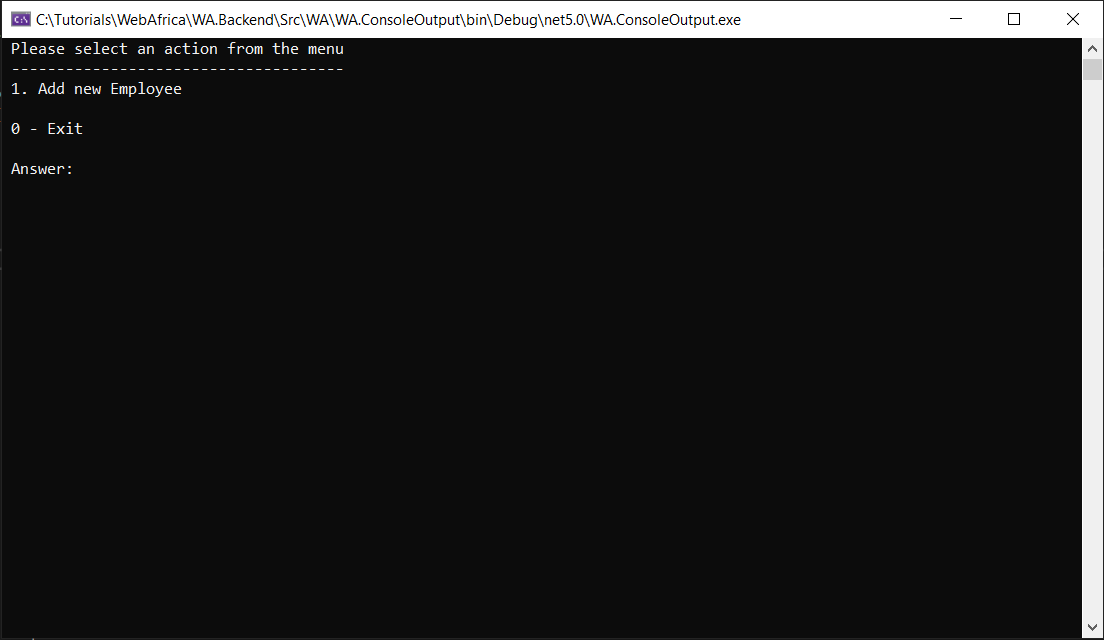


Fig 13. Add employee through the console

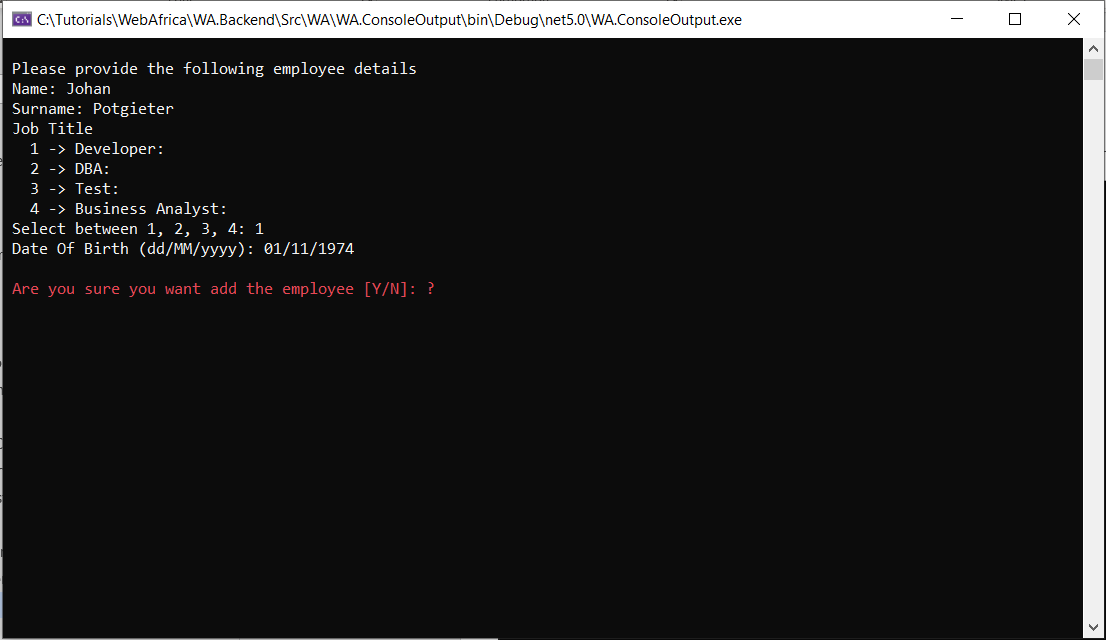
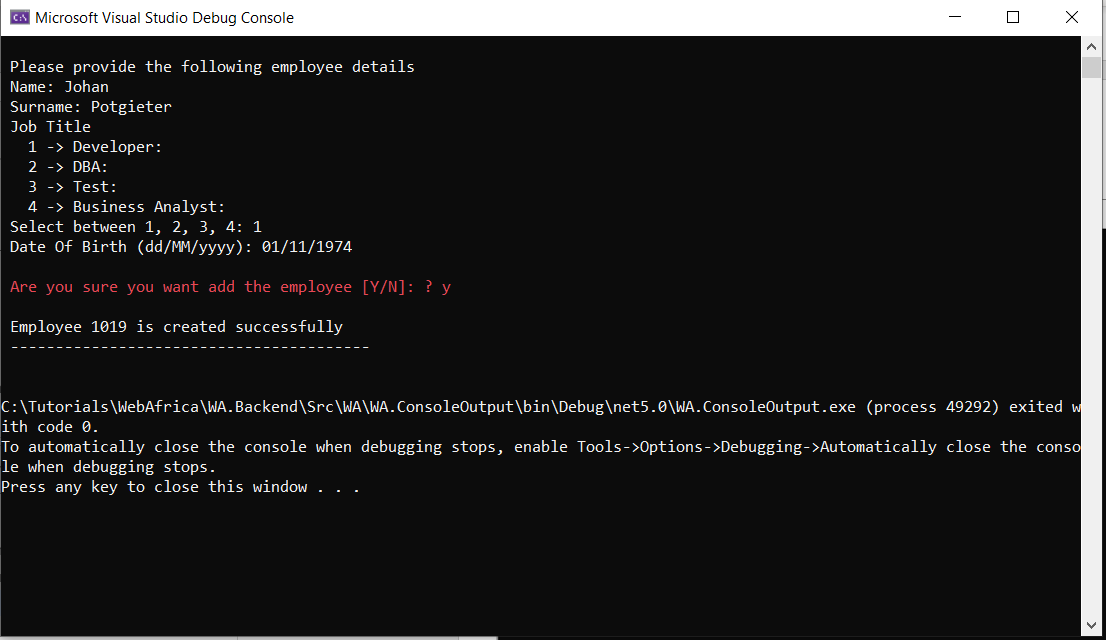


Fig 14. Enter employee details

Fig 15. Employee added

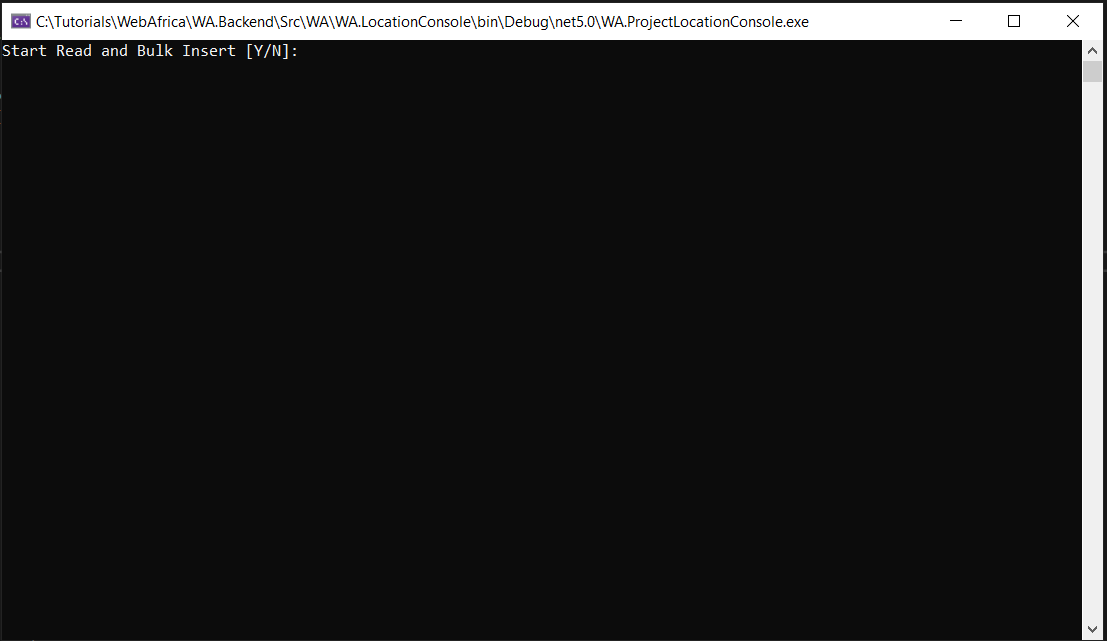


Fig 16. Bulk Insert

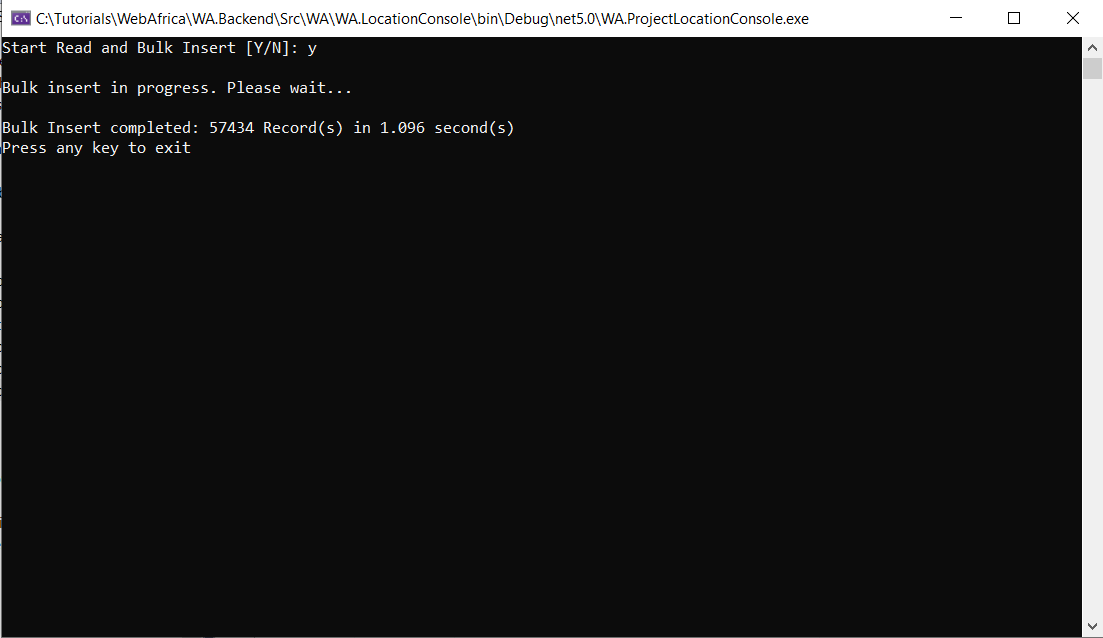


Fig 17. Bulk insert complete

Diagram

Description automatically generated

Fig 18. Project listing activity diagram

### User Stories

Diagram

Description automatically generated

Fig 19. Employee and Project user stories

Diagram

Description automatically generated

Fig 20. Composite Services

Diagram

Description automatically generated

Fig 16. Composite classes

## Conclusion

The construction of the solution is hopefully done in a way that supports loose coupling and high cohesion.

1. Class, method and variable names are given meaningful values. The only time where x is used as variable is as a loop counter variable.
2. The controllers do not contain any logic. Filters can be used to remove redundant code such as try catch.
3. Fields are declared with a leading \_ to indicate the role it plays and can be found at the top of each class.
4. Any injected dependency (services) is declared as read-only to keep it immutable.
5. Dividing the solution into separate projects, make the application more testable and maintainable.
6. Please note that all requests are handled asynchronously.
7. Base entity is created to store any common properties such as Id and are inherited by dto and entity classes where applicable.
8. Base interface is also created to store any common CRUD operations and are inherited by service interfaces where applicable.

To get the solution running:

1. Start the api solution. The default url is <http://localhost:5000> and <https://localhost:5001> for https.
2. After opening the angular solution – run the npm install command. This will install all the dependencies needed.
3. Run the front end with ng serve. The default url is <http://localhost:4200>.
4. Testing can also be performed with the use of swagger. Go to url <https://localhost:5001/swagger/index.html>
5. Alternatively – import the the swagger.json (included) into postman. This will setup all the existing endpoints in postman.