Rank Interactive Technical Specs Overview

# February 28, 2022

### Business Requirements

1. Requirements for the Rank Interactive credit card api are as follows:
   1. Allow a user to submit a valid credit card.
   2. Validation is applied on card submission.
      1. Minimum card length greater than zero characters
      2. Maximum card length not exceeding threshold value.
      3. Card must include part of the card provider id number (Check if card is Visa or Mastercard).
   3. Card should not be added twice.
   4. If card is valid, store it somewhere.
   5. Display valid credit cards.
   6. Create a list of credit card providers.
2. Extra features
   1. Ability to delete credit cards from the system.
   2. Ability to log into the system and then provide a JWT token for further requests to the backend.

### Target Audience

Rank Interactive employees

### Name of project

RCCM – Rank Interactive Credit Card Management

### Technical Specs

1. Server-Side Language: C#
2. Framework: .net Core 5.0
3. ORM: EF Core
4. Database: Sql Server 2016 Express
5. Unit test: xUnit
6. IDE: Visusal Studio 2019
7. Frontend – Angular 12
8. UI CSS Toolkit: Bootstrap / Primeng

### Solution

The Rank Interactive 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 – RCCM.Api

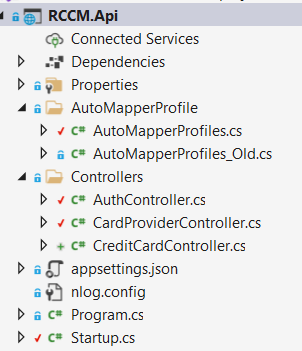


Fig. 1 - RCCM.Api

This project consists of the Startup.cs file and the Auth, CardProvider and Credit Card Controllers.

Auth controller – used for authentication. When user provides a password, it is hashed and then compared to a hashed version stored in the database. A seed data method is used to populate the user’s table. Once the user is verified, a jwt token is sent back to the front-end where it is stored in localstorage for later use in the auth guards to ensure the user is allowed to activate routes. The jwt token is then injected into subsequent requests.

CardProvider controller. This controller serves as the entry point for CRUD operations. A card provider service is injected into the controller through the Di container. This service is then used to create, list and delete providers.

The CreditCard controller is used to add, validate and delete valid credit cards. Please note that most of the validation occurs on the client side. A credit card service is injected into the controller to provide access to CRUD operations.

Project 2. RCCM.Contracts

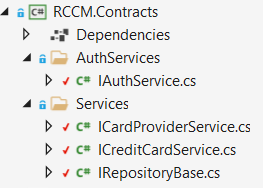


Fig 2. RCCM.Contracts

Four contracts are used in this project.

1. IAuthService is implemented as a service to perform any authorisation. See the RCCM.Service project.
2. IRepositoryBase is used to contain common and generic methods that can be inherited by other repository services. In this example only the SaveChangesAsync is reused in other services. In a more complex project methods such as Create<T>, Update<T>, Delete<T>, GetAll<T> could be include in the IRepositoryBase.
3. ICardProvider and ICreditCardService are two interfaces dedicated to Credit Card and Credit Card Provider CRUD operations.
4. All these interfaces are implemented in the RCCM.Services project. This is then added to the dependency injection container of .net core 5.

RCCM.Data

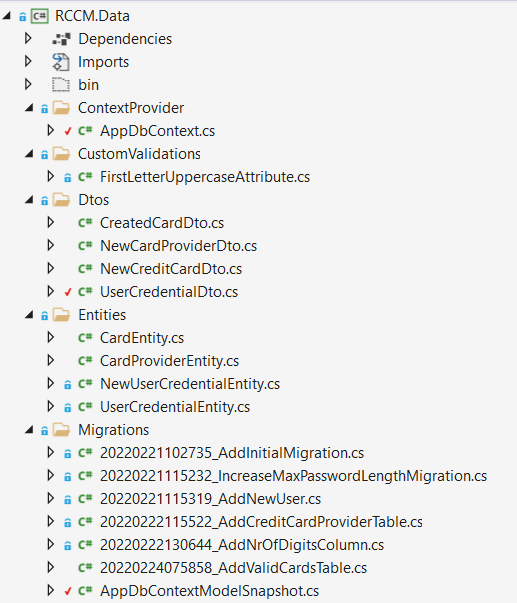


Fig 3. RCCM.Data

This project is used to communicate with the database with the help of EF Core. Implementing this function in its own project allows the solution to migrate to a different database vendors or type without having to rewrite other portions of the solution. This also makes testing the different layers easier as the db can be mocked.

1. The Entities folder contains the classes that correspond with the database tables.
2. The context folder contains the connection to the database.
3. Migrations folder contains all the migrations of EF core that were applied.
4. The dtos folder contains classes that map request parameter properties. Having multiple dtos for the same entity (class that relates to the db table), allows for data-shaping that correspond with a specific request type (put, post, delete, get).
5. CustomValidations is used for custom attribute validation.

RCCM.Di

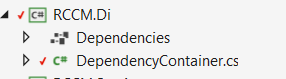


Fig 5. RCCM.Di

The Di project registers all the interfaces and ef core db context classes 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.

RCCM.Service

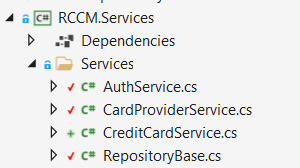


Fig 6. RCCM.Service

The service project is used to separate the different functions into separate logical files. The service classes implement the corresponding interfaces in RCCM.Contracts project which is then injected into the controllers.

RCCM.Test

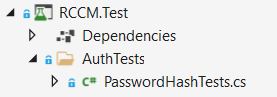


Fig. 7 RCCM.Test

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

RCCM.UI

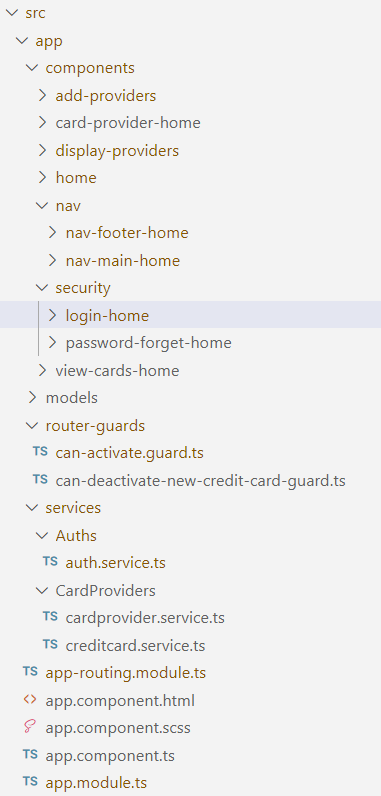


Fig 8.

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 8 shows the logical structure of the frontend –

1. Components: This houses the different components used in the project.
2. Services: there are three services in the Services folder that communicates with the RCCM.Api project by sending data in the form of parameters.
3. Models: contains the classes used in the project.
4. The router guard folder stores the CanActivate and CanDeactivate router guards. Injecting them into the routers prevent unauthorize use of the frontend application.

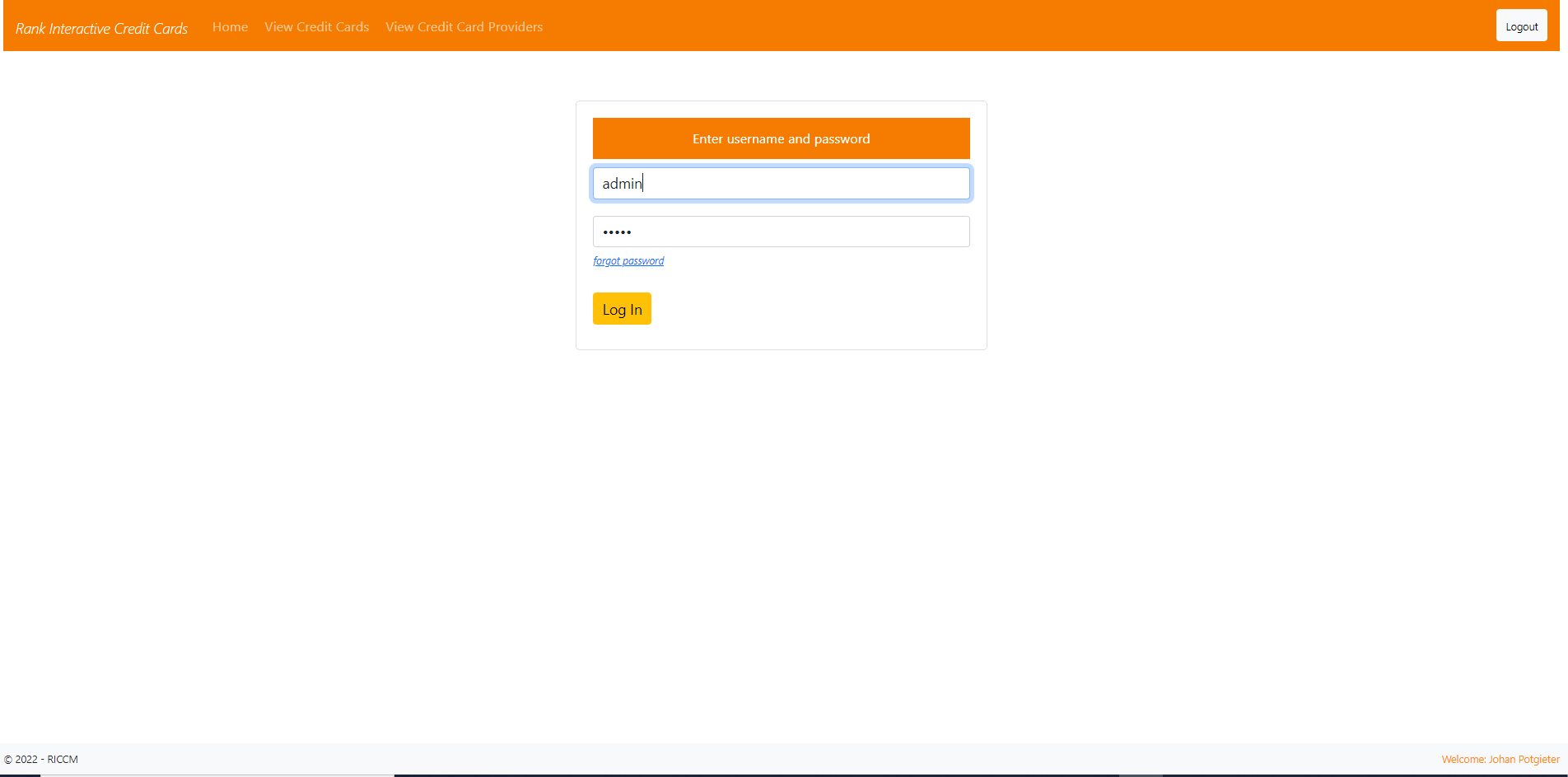


Fig. 9.

Fig 9 shows the login screen. Current username and password are admin and admin. The password is initially hashed on the server side and stored in the database (Done with seed methods). Any further login attempts will hash the entered password with a hashed version stored in the database. A jwt token is generated during the login process which is send to the client. The token is stored in the localstorage for later use. As mentioned earlier, the token is used in the CanActivate auth guard. The token is also injected into the headers of subsequent requests.

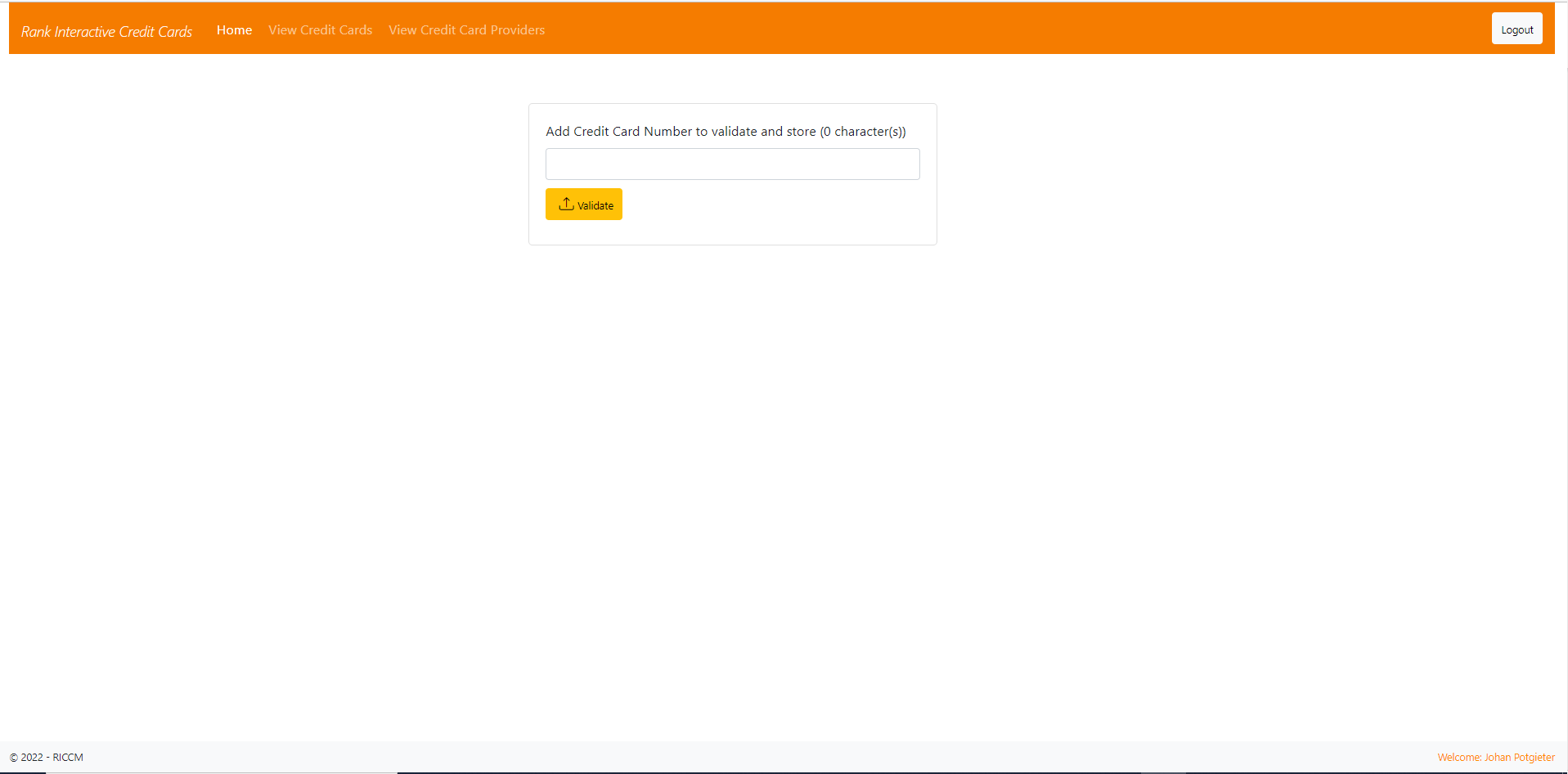


Fig. 10 Home Component

After successful login, the user will be redirected to the home component where a credit card can be entered. The card number is subject to validation as stipulated in the Business Requirements section.

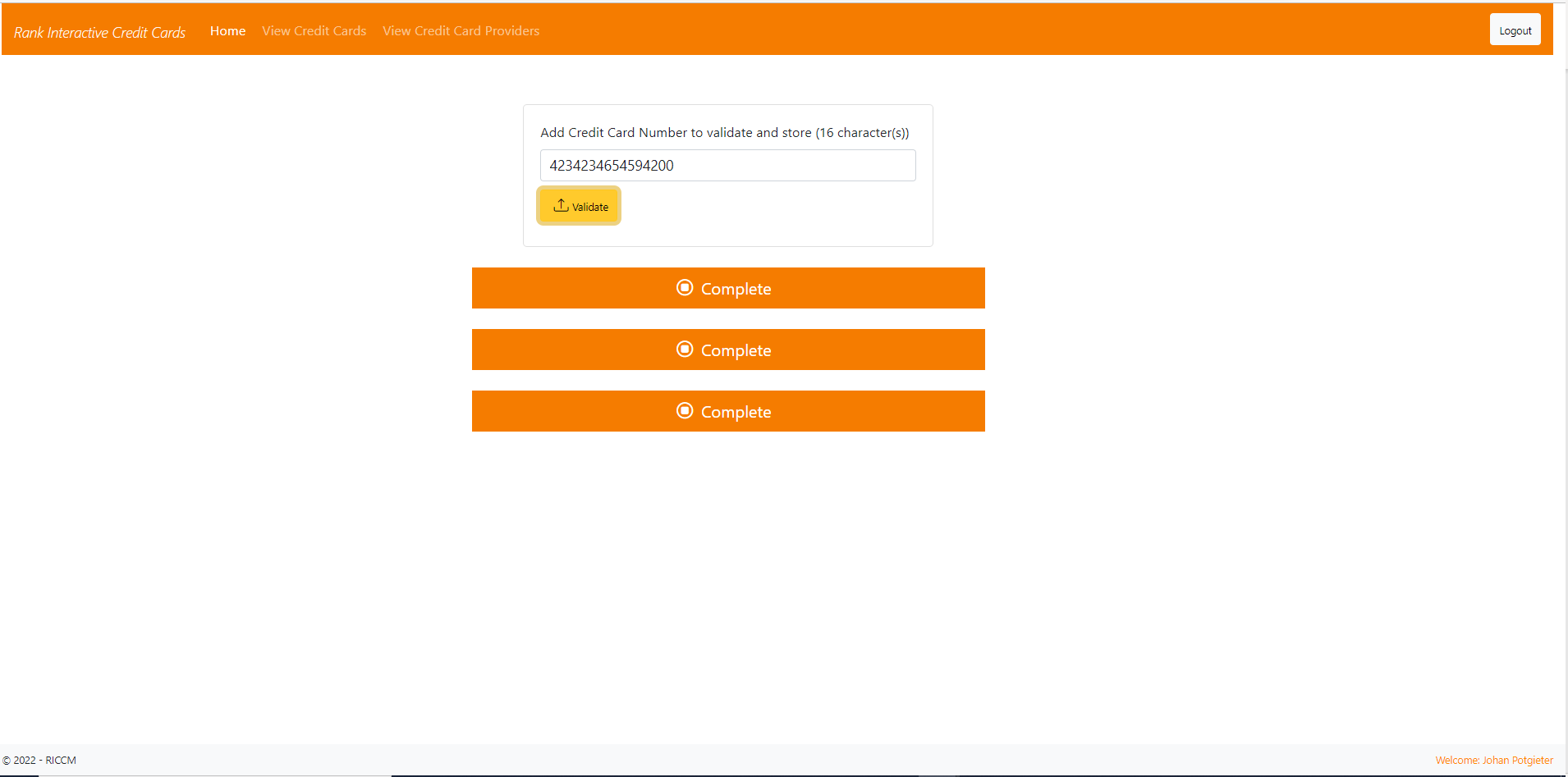


Fig 11.

Once the credit card number passed the validation (see Fig 11), it is stored in the database. To see a listing of the card numbers, see Fig 12.

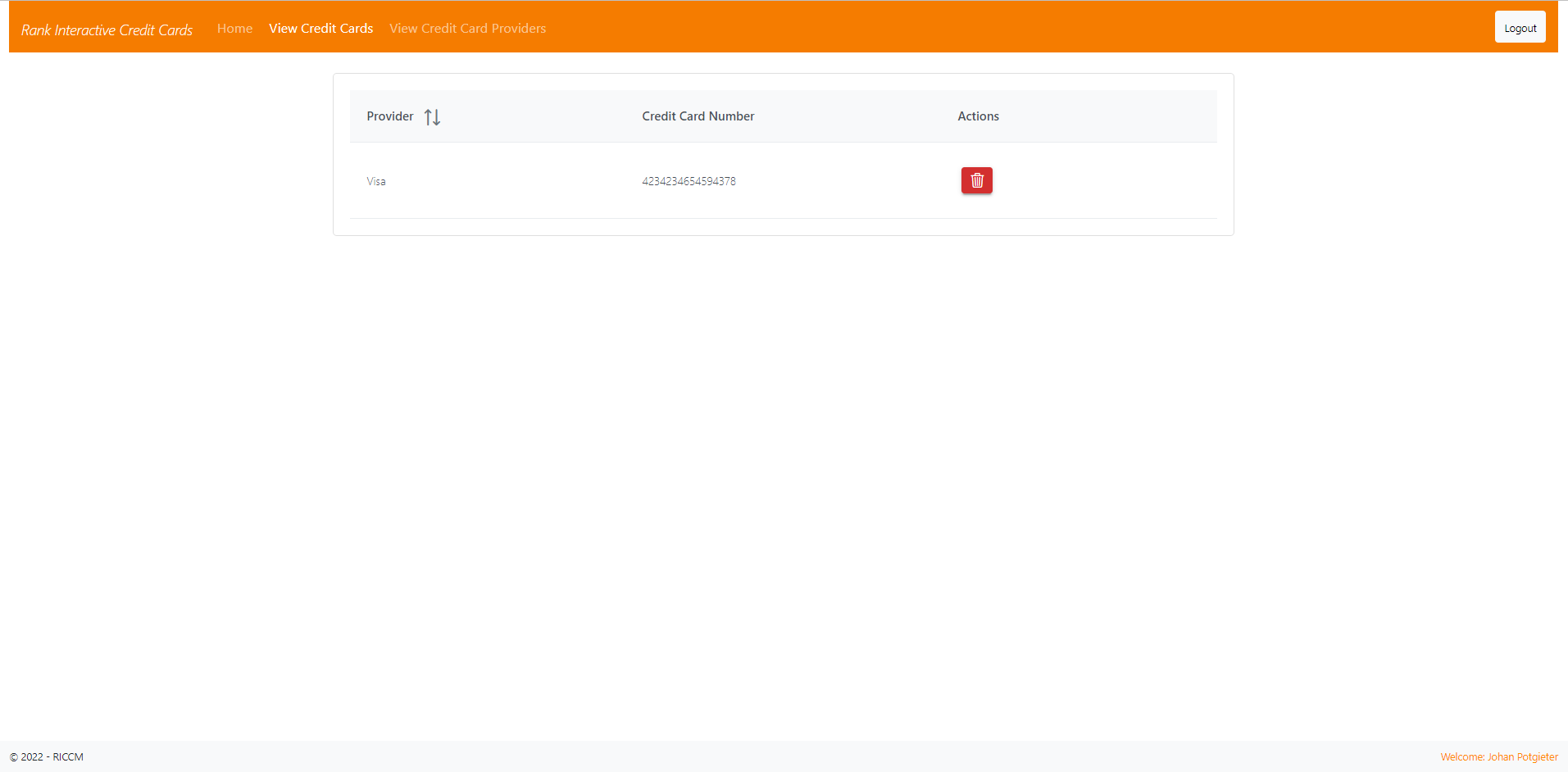


Fig 12.

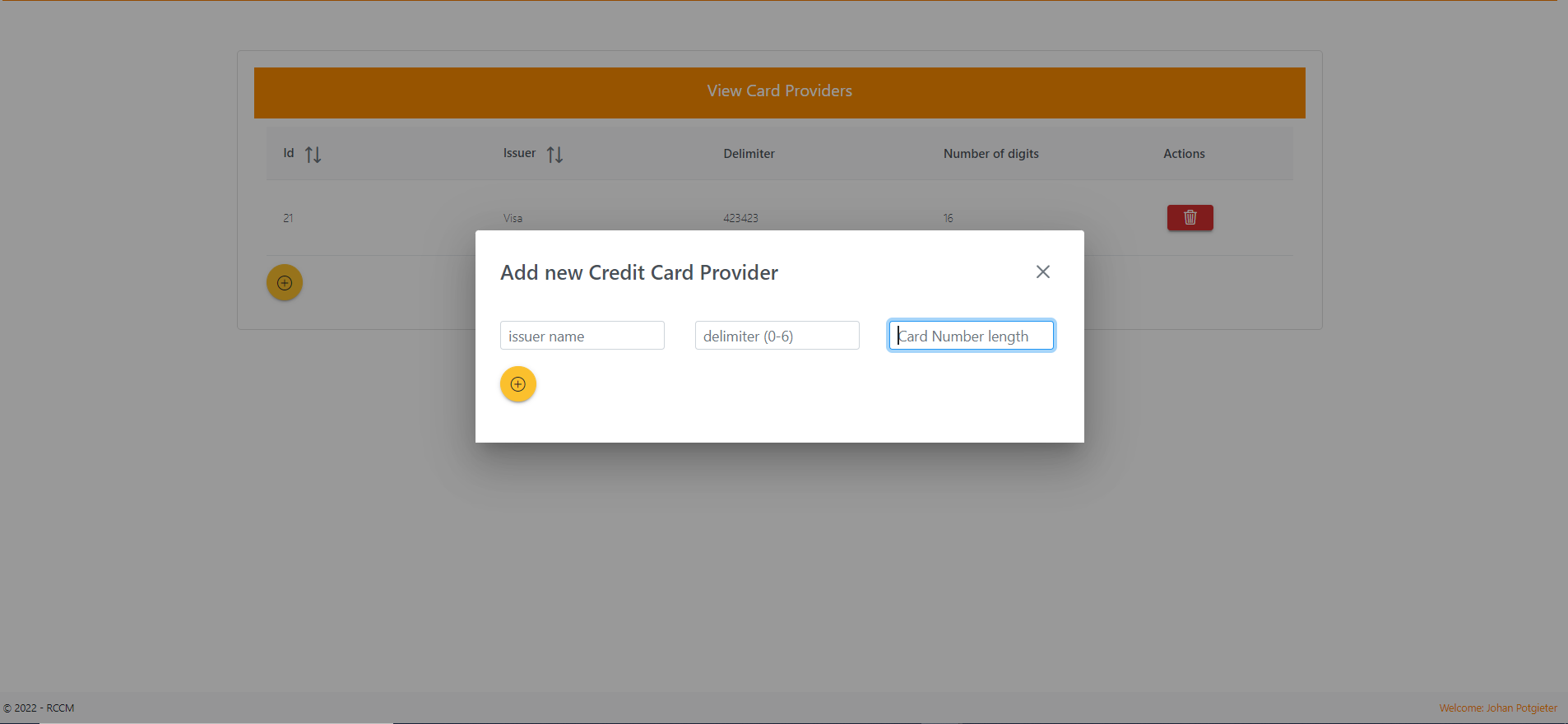


Fig 13.

To add a new card provider, click on the View Card Providers menu item. A list of existing providers can be seen and deleted. Click on the yellow button to add a new card provider.

**PS**

Issuer name - MasterCard

Delimiter – a number that acts as card provider identity. During the validation procedure on the client application, this value is compared to the first digits of the entered card number.

Card Number Length is to specify the maximum number of digits per provider. This is also used during the validation procedure.

Diagram

Description automatically generated

Fig. 14 – Login activity diagram

## 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 but there are room for improvement. 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.

To get the solution running:

1. First run the migrations or use the included sql script to create the database after changing the connection string in appSettings.json.
2. Start the api solution. The default url is <http://localhost:5000> and <https://localhost:5001> for https.
3. Run the front end with ng serve. The default url is <http://localhost:4200>.
4. [Swagger UI](https://localhost:5001/swagger/index.html) -> <https://localhost:5001/swagger>.
5. Please find the swagger endpoints in json format for import into postman in the docs folder. File -> swagger-endpoints.json