# Requirements

The requirements mainly revolve around members being able to authenticate and access the internal system. Since a database is required to store personal details and a PIN, they will need to be encrypted to protect the cardholder’s personal information.

I will be providing the REST API using an HTTP web service, providing various endpoints and using HTTP status codes to send responses to the end user. For example, in an authentication request, if the user is not found, sending back a 404 will indicate the user was not found and needs to register.

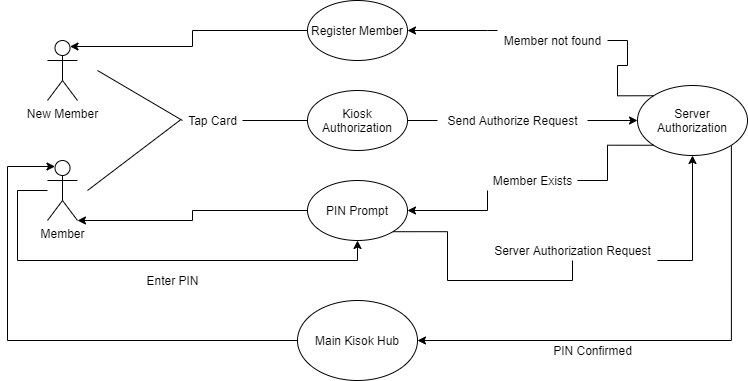
Although it is envisioned that the user will tap the card a second time to log out, the Login/Logout endpoints will be separated to conform with HTTP standards and allow for potential alternative logout methods in the future such as a logout button, thus it will be the responsibility of the Kiosk developer(s) to implement the “double card tap” feature. I will also be using HTTP Authorization headers using Basic Authentication as an additional security feature.

Some requirements that aren’t listed that would be useful include:

* Ability to update/edit user information
* Ability to delete user so the card can be used by someone else
  + This could be done by an administrator member

# Diagrams

An example use case for the REST API would be the following for members new and old going through the authorization process:



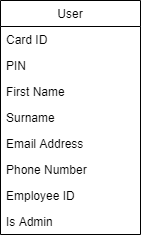
The authorization process can also be represented in detail by the following flow chart:

Diagram

Description automatically generated

## Class Diagrams

Since the core of this API revolves around memberships, a User class will be required as the business logic. Consider the following class:



This class details the main fields that will likely be required for the User class. Since I’m using HTTP, the login request and the register request, while similar, will not require all of the information to be sent. Since I plan to make a separate endpoint for creating admin users, the register endpoint won’t need the “Is Admin” field. Similarly, the Login endpoint will only need the Card ID and PIN.

I can therefore break down the User class into separate classes to create an inheritance hierarchy:

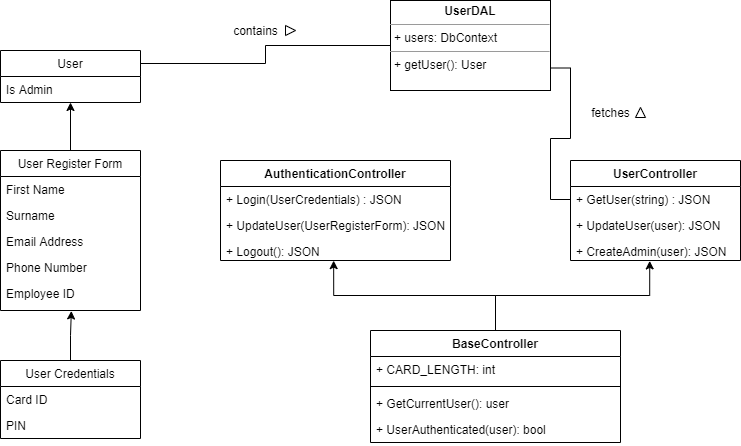


The Login endpoint will accept the “User Credentials” model, and the “User Register Form” will be sent to the Register endpoint.

Other classes I will need include:

* Controllers (ASP.NET endpoints)
* Data Access Layer (DAL) classes

Both of these types of classes have some form of “responsibility” i.e. the controllers are responsible for receiving HTTP requests and sending a response to the user. They do not make any database calls, this is the job of the DAL classes. Consider the following class diagram:



Here we can see a mockup relationship between the controllers, the DAL and the User model. Note that the UserController and the AuthenticationController inherit from a BaseController. The BaseController will be an abstract class containing common methods and fields, such as the character limit of the Card ID in the form of a constant, or a getter for the current user.

The GetUser() method also makes a call to the DAL to “fetch” the requested User from the database, and the DAL “contains” many User objects.

# Technologies/Software

To develop this REST API, I will be using the following programs and technologies:

* C# .NET Core 3.1 as the API backend
* MongoDB as the database
* Git for version control
* Github to host the source code and utilize continuous integration technologies
* Postman for manual testing
* Sandcastle to generate technical documentation