# **Introduction**

## Background information

For my synoptic project I chose to pick the membership system API project to complete. The scenario is that the customer is a catering business which supplies food and drinks to large scale businesses. Currently they provide their own cards to customers which they can use to top up and pay for things while at work however they have been given a new contract which requires them to adapt their service. The new company has requested that they allow the use of previous employee cards and use their existing kiosk terminals so they will need to create a web service API which can be run on these existing kiosks. I will have to build a RESTful web service API to complete this.

# **Design**

## Business requirements

### Functional Requirements

From the brief I have inferred that the requirements for the system are API’s that a kiosk can interact with for the user to login, register their card, log out, buy items and add money onto their balance. For the Functional Requirements I have written out the behaviour the function will follow in gherkin.

#### Logging in

Given the employees card is registered

When the user logs in

Then they should be able to log in successfully

Given the employees card is not registered

When the user logs in

Then they should receive a bad request result

Given the employee enters an incorrect card id

Then they should be told the card does not exist

Given the employees card is registered

And they enter an incorrect pin

Then they should be told their login details are incorrect

#### Registering

Given the employees card is not registered

When the user registers with valid inputs

Then they should receive a ok object result

Given the employees card is not registered

When the user registers with invalid inputs

Then they should be told which is incorrect

#### Adding money

Given the user is logged in

When they attempted to add £5

Then Their balance should increase by £5

And they should receive a ok result

Given the user is logged in

And they have not done anything on the system for over 10 mins

When they attempted to add £5

Then They should receive a timeout error

#### Buying something

Given the user is logged in

And they have £5 in their account

When they attempt to buy something for £2

Then their balance should decrease by £5

And they should receive a ok result

Given the user is not logged in

And they have £5 in their account

When they attempt to buy something for £2

Then they should receive a bad request result

Given the user is logged in

And they have £1 in their account

When they attempt to buy something for £2

Then they should receive a bad request result

Given the user is logged in

And they have not done anything on the system for over 10 mins

When they attempted to buy something for £5

Then They should receive a timeout error

#### Logging off

Given the employees is logged in

When the user logs off

Then they should receive a ok result

Given the employees is not logged in

When the user logs off

Then they should receive a bad request result

Given the employees is logged in

And gives the wrong pin

Then They should receive a authentication error

### Non-functional requirements

These are many non-functional requirements I inferred from the requirements that the system should confirm to industry standards.

Timing constraints will also be in place as the system will need to able to perform quickly without keeping the user waiting for an unreasonable amount of time. To do this is validate all user inputs before querying the database. This saves time as looking for things in a database takes considerably more time than quick validations in C#. I also use session tokens to validate requests. This speeds up the process of validating a user because only the tables holding session data and employee pin numbers are accessed instead of accessing a further third membership card table shaving time off execution. This also saves resources which will save money for the company running the server. This would also speed up the time taken by each employee speeding up the service and being able to accommodate more staff per kiosk.

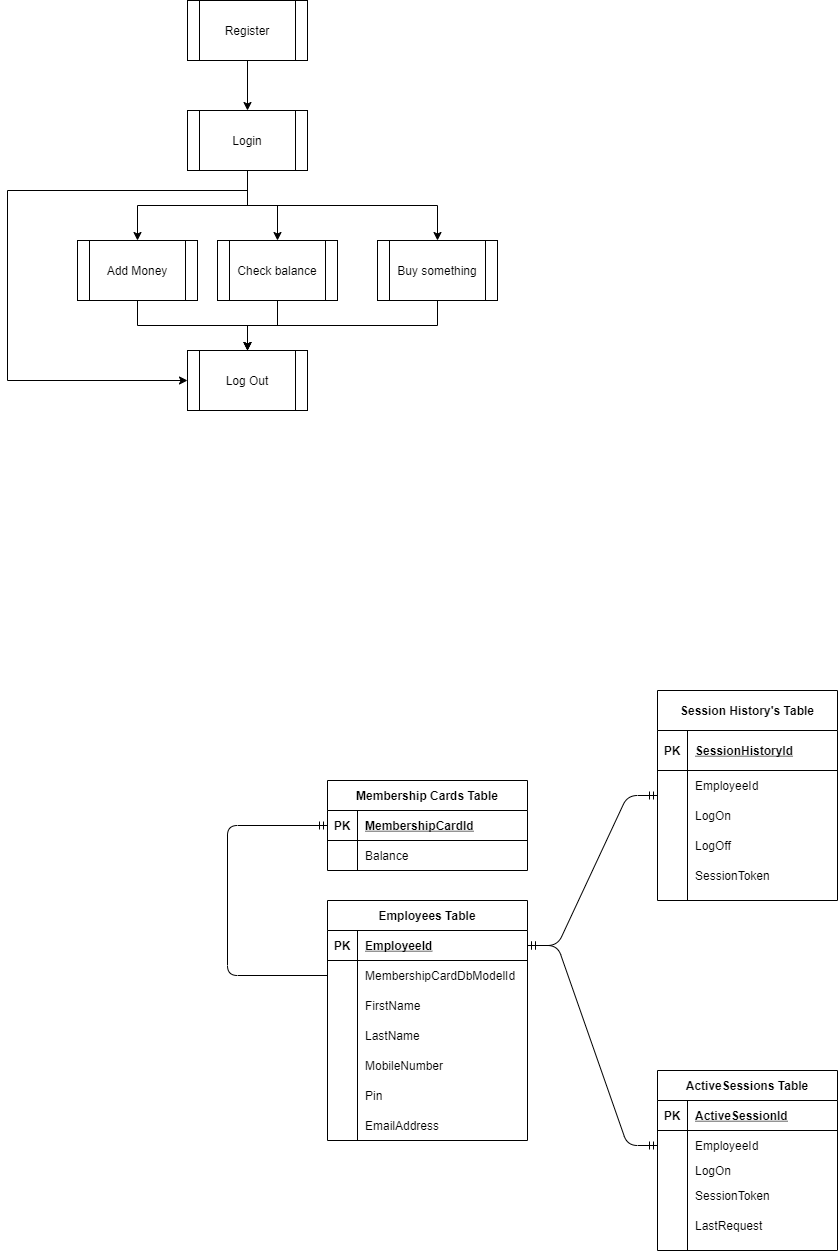
The system will be created as simply as possible to make it as easily maintainable as possible by any of the staff that will need to deal with any issues that may arise and will log errors so they can be caught by analytics.

## Assumed requirements

I assumed a few requirements for the system. The first was that the kiosk should be prompted on what the issue wit a request is so for most errors there is a message returned of the error. The next was that I assumed the kiosk would need a function to be able to query the balance of a card and display this. This would be for if a user needed to double check their balance before purchasing items.

## Data model

There is a one to one relationship between the membership card and employees table on the MemberberCardDBModel ID. There is also one to one relationships between the employee table and the session tables however this constraint isn’t applied to the database.



## Activity diagram flow charts

Here is a higher level overview of how the user will use the system. After the registration the user will then have to log in. This is where the kiosk will be sent a session token by the login API which will then be used until the user logs out. This session token will be used to add money, buy an item, check the balance of employee’s card or log out. This session token coupled with the user’s pin is what will be used to make requests to the system by the kiosk. I designed it like this because it increases security since there wont be multiple requests with both pin number and membership card ID so it is less likely to be intercepted. The timeout function isn’t included in this because the kiosk doesn’t interact with it and its handled on the server side.

