Emily Buckley

First Catering API documentation

# Project Overview

This project is to build a RESTful API that will query a database and provide information to the kiosk terminals. Users will be using cards that are assigned to their unique employee ID, this will allow them to purchase food and drinks through this contactless system. These cards are already existing and not to be implemented by the new catering company; when a card is tapped onto the kiosk terminals a unique 16-digit alphanumeric Id number will be fed into the system. This 16-digit number will be already assigned to a user or it will be unassigned. If the card is assigned, then the API will respond with a welcome message that includes the users name. If the card is unregistered then the API will respond with unregistered card which will then tell the kiosk to query a different path on the API, this will allow the user to set up their card using their unique employee ID, name, email and mobile number, this will be saved to a database so that if the card is tapped again then the system will recognise them.

When the card is tapped for a second time the API will respond with ‘Goodbye’, a goodbye will also be issued after several minutes of inactivity.

## Assumptions & Changes

It is generally accepted that new style API’s should reply in JSON format, and there should be no transient state between requests and each request is separate to all others.

As part of the welcome message I will be including, the time of day to make it feel more interactive, this will include:

* `Good morning {name}, please enter in your pin. `
* `Good afternoon {name}, please enter in your pin. `

I have also decided to include the user updating the amount they have on their card, I had considered including a fake request to a bank that would validate if the user had enough money to update their card automatically however this was outside of the scope of this project and would have taken too much of the development time.

## Use case diagram

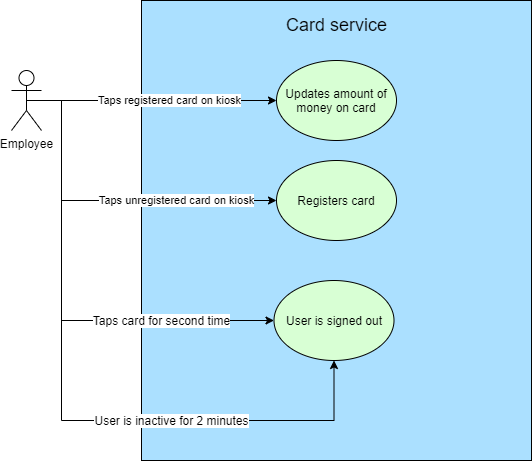


Figure Use case diagram

*Figure 1* shows a use case diagram which shows the different ways that the user can use the API. If a user taps a registered card onto the kiosk the database will be queried with their Card ID, if there is an entry for this then there will be a personalised response from the service which will depend on the current time of day:

* `Good morning {name}, please enter in your pin. `
* `Good afternoon {name}, please enter in your pin. `

When they enter their pin into the kiosk’s GUI (Graphical User Interface) a different endpoint will be queried with the user’s Card ID and pin, the API will respond with `{name}, you currently have {value} in your account. How much would you like to top up? ` they will be prompted by the GUI to enter in an amount they would like to top up their card by. The database will then be updated to top up their card by that amount.

If the card is not registered to a user then another endpoint will be queried, and the user will be prompted to enter in their:

* Unique employee ID
* Users Name
* Email
* Mobile number
* Pin

The default account balance is 0 pounds, they will need to top-up to be able to use it. Once they have entered in this information, they will then be directed back to the main endpoint and asked to enter in their pin.

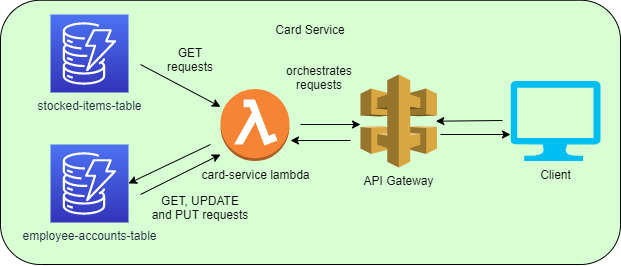
If the user taps their card onto the kiosk again or if the API is not queried for 2 minutes then the API will sign the user out by returning to a default endpoint.

## Flow Diagram

Diagram, engineering drawing

Description automatically generated

## Architecture Diagram



There will be two databases, one to store employee data and another to store the items the employees are allowed to purchase. This will be queried by a lambda which is essentially like a serverless function in the cloud. API gateway will allow me to set up different endpoints and have different methods on each endpoint. It will also deploy my API into the cloud meaning that it can be tested from anywhere.

## Endpoints

There will be 5 endpoints:

* **Default endpoint** ‘/’

*Responses:*

* + `Good morning, please tap your card to begin` or
  + `Good afternoon, please tap your card to begin`
* **Login endpoint** has one required query parameter `/login?cardId={cardId}`

*Responses:*

* + `Good morning {users name}, please enter in your pin. `
  + `Good afternoon {users name}, please enter in your pin. `
* **Account endpoint** has two required query parameters `/account?cardId={cardId}&pin={pin}` will return, if you do not include the pin it will return you to the login endpoint, if you do not have either it will return you to the default endpoint.

*Responses:*

* + `{name}, you currently have {accountBalance} in your account. What would you like to do? top-up or purchase food`
  + If the user wants to top up they press top-up and are asked how much they want to top up by, a post request is sent to `/account/topup?cardId={cardId}&pin={pin}` Body: `updateBalance:{amount added to balance}` this will update the database.
  + They will be directed back to the main account endpoint. The database will be queried again to show the current balance.
  + They can then purchase food by querying the `/account/purchase?cardId={cardId}&pin={pin}` which will return all food that is currently in stock and their prices, ie [{sandwich: 2.99}, {burger: 5.99}],
  + When the user enters in a purchase the purchase endpoint will be called again with the float value in the body. `/account/topup?cardId={cardId}&pin={pin}` Body: ` {food they would like to purchase and price}` this will be taken away from their current balance and update the database if they have enough money in their account.
  + They will be directed back to the main account endpoint the database will be queried again to show the current balance.
  + The user can then tap their card again to log out or leave the kiosk and once the 2-minute timer is up they will be directed to the Goodbye endpoint.
* **Goodbye endpoint** has no required parameters, but you can send one name parameter for a personalised response `/goodbye?name={name}` or `/goodbye`

Responses:

* + `Goodbye {name}` or
  + `Goodbye`
  + This will timeout after 10 seconds and be diverted back to the default endpoint.
* **Signup endpoint** this has a required parameter of cardId `/signup?cardId={cardId}`

*Responses:*

* + `Your card isn’t registered, please enter in your EmployeeID, Name, Email, mobile number and secret pin. Please remember your pin as you will need it for logging in`
* **Signup endpoint** `/signup?cardId={cardId}` with above employee details as the body These details will be sent to the database with the cardId as the primary key.

*Responses:*

* + `You’ve been signed up! You will now be redirected to the login page where you will be prompted with your pin` and the user will be directed to the *Login endpoint*.

## Validation

The main value that needs to be validated is the input for the top-up as this needs to be a float or integer value in order for the system to work correctly, all other values could be validated using regex however it is not as important as the value does not need to be updated once the account has been made.

# Data model for API paths

This is all the endpoints, what the requests that can be made, examples of data that is being exchanged.

## Path = ‘/’ DEFAULT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METHOD | INPUT | | OUTPUT | |
| GET | Property Name | N/A | Property Name | message |
| Type | N/A | Type | String |
| Example | N/A | Example | {“message”: “Good morning, please tap your card to begin”} |

## Path = ‘/Login’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METHOD | INPUT (QUERY STRING PARAMETERS) | | OUTPUT (JSON) | |
| GET | Property Name | cardId | Property Name | message |
| Type | String | Type | String |
| Example | “r7jTG7dqBy5wGO4L” | Example | {“message”: “Good morning James Smith, please enter in your pin.”} |

## Path = ‘/Account’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METHOD | INPUT (QUERY STRING PARAMETERS) | | OUTPUT (JSON) | |
| GET | Property Name | cardId | Property Name | message |
| Type | String | Type | String |
| Example | “r7jTG7dqBy5wGO4L” | Example | {“message”: “James Smith, you currently have £10.70 in your account. What would you like to do? top-up or purchase food”} |
| Property Name | pin |  |  |
| Type | String |  |  |
| Example | “1234” |  |  |

## Path = ‘/Account/topup’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METHOD | INPUT (QUERY STRING PARAMETERS) | | OUTPUT (JSON) | |
| POST | Property Name | cardId | Property Name | message |
| Type | String | Type | String |
| Example | “r7jTG7dqBy5wGO4L” | Example | {“message”: “Your account has now been updated to £23.00, you will now be directed back to the main account”} |
| Property Name | pin |  |  |
| Type | String |  |  |
| Example | “1234” |  |  |
| **INPUT (BODY)** | |  |  |
| Property Name | updateBalance |  |  |
| Type | Object |  |  |
| Example | {“updateBalance”: 20.00} |  |  |

The user is then returned to the main account path.

## Path = ‘/Account/Purchase’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METHOD | INPUT (QUERY STRING PARAMETERS) | | OUTPUT (JSON) | |
| GET | Property Name | cardId | Property Name | Food Items |
| Type | String | Type | Array |
| Example | “r7jTG7dqBy5wGO4L” | Example |  |
| Property Name | pin |  |  |
| Type | String |  |  |
| Example | “1234” |  |  |
| POST | Property Name | cardId | Property Name | message |
| Type | String | Type | String |
| Example | “r7jTG7dqBy5wGO4L” | Example | {“message”: “You have now paid, your balance has been updated, you will now be directed back to the account page”} |
| Property Name | pin |  |  |
| Type | String |  |  |
| Example | “1234” |  |  |
| **INPUT (BODY)** | |  |  |
| Property Name | burger |  |  |
|  |  |  |  |
| Type | Object |  |  |
| Example | {“burger”: 4.99} |  |  |

The user is then directed back to the main account path.

## Path = ‘/goodbye?name={name}’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METHOD | INPUT | | OUTPUT | |
| GET | Property Name | name | Property Name | message |
| Type | string | Type | String |
| Example | “James Smith” | Example | {“message”: “Goodbye James Smith”} |

This endpoint will time out after 10 seconds, it will then divert you back to the default endpoint.

## Path = ‘/signup?cardId={cardid}’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| METHOD | INPUT | | OUTPUT | |
| GET | Property Name | cardId | Property Name | message |
| Type | string | Type | String |
| Example | “r7jTG7dqBy5wGO4L” | Example | {“message”: “Your card isn’t registered, please enter in your EmployeeID, Name, Email, mobile number and secret pin. Please remember your pin as you will need it for logging in”} |
| POST | Property Name | cardId | Property Name | message |
| Type | String | Type | String |
| Example | “r7jTG7dqBy5wGO4L” | Example | {“message”: “You’ve been signed up! You will now be redirected to the login page where you will be prompted with your pin”} |
| **INPUT (BODY)** | |  |  |
| Property Name | employee details |  |  |
| Type | Object |  |  |
| Example **(BODY)** |  |  |  |

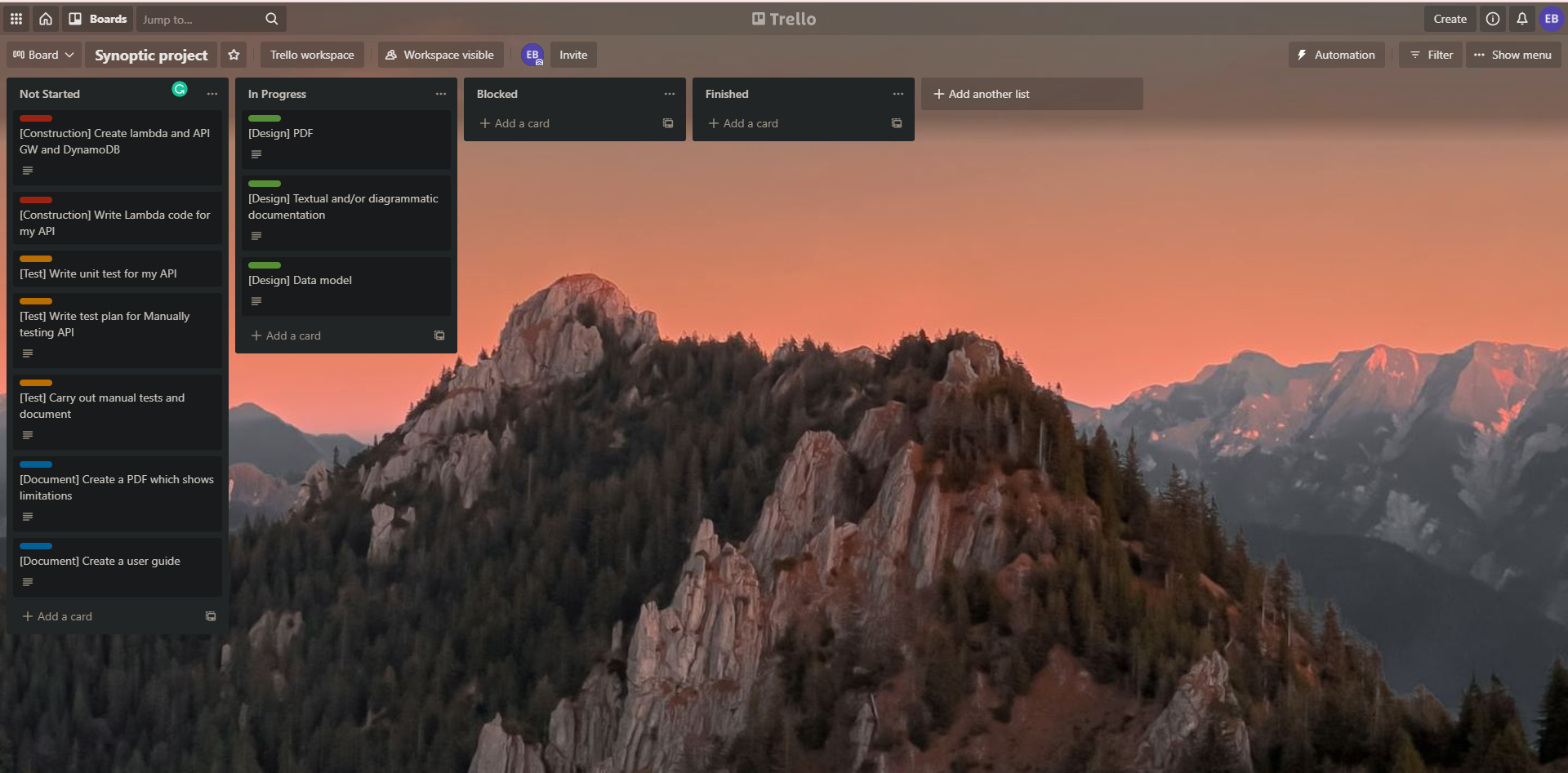
They will then be directed to the signup endpoint.

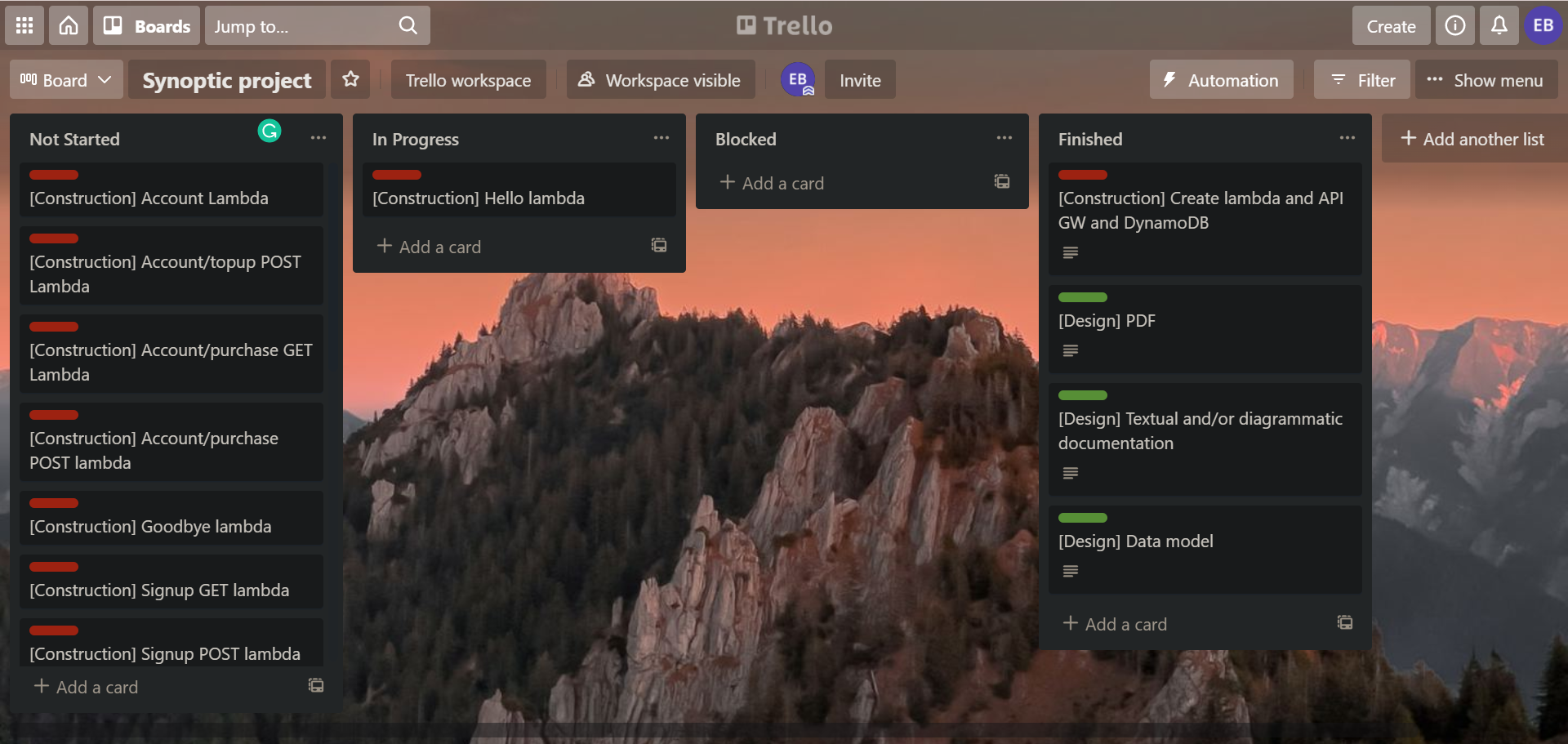
# Data model database tables

|  |  |  |  |
| --- | --- | --- | --- |
| TABLE NAME | PROPERTY NAME | TYPE | EXAMPLE |
| employee-accounts-table | cardId **(PRIMARY KEY)** | String | “r7jTG7dqBy5wGO4L” |
| employeeId | String | “1a2b3c” |
| name | String | “John Smith” |
| email | String | “john.smith@f1.com” |
| mobileNo | String | “07858423012” |
| pin | String | “1234” |
| accountBalance | Float | 9.76 |
| stocked-items-table | productId **(PRIMARY KEY)** | String | “1” |
| name | String | “burger” |
| price | Float | 4.99 |

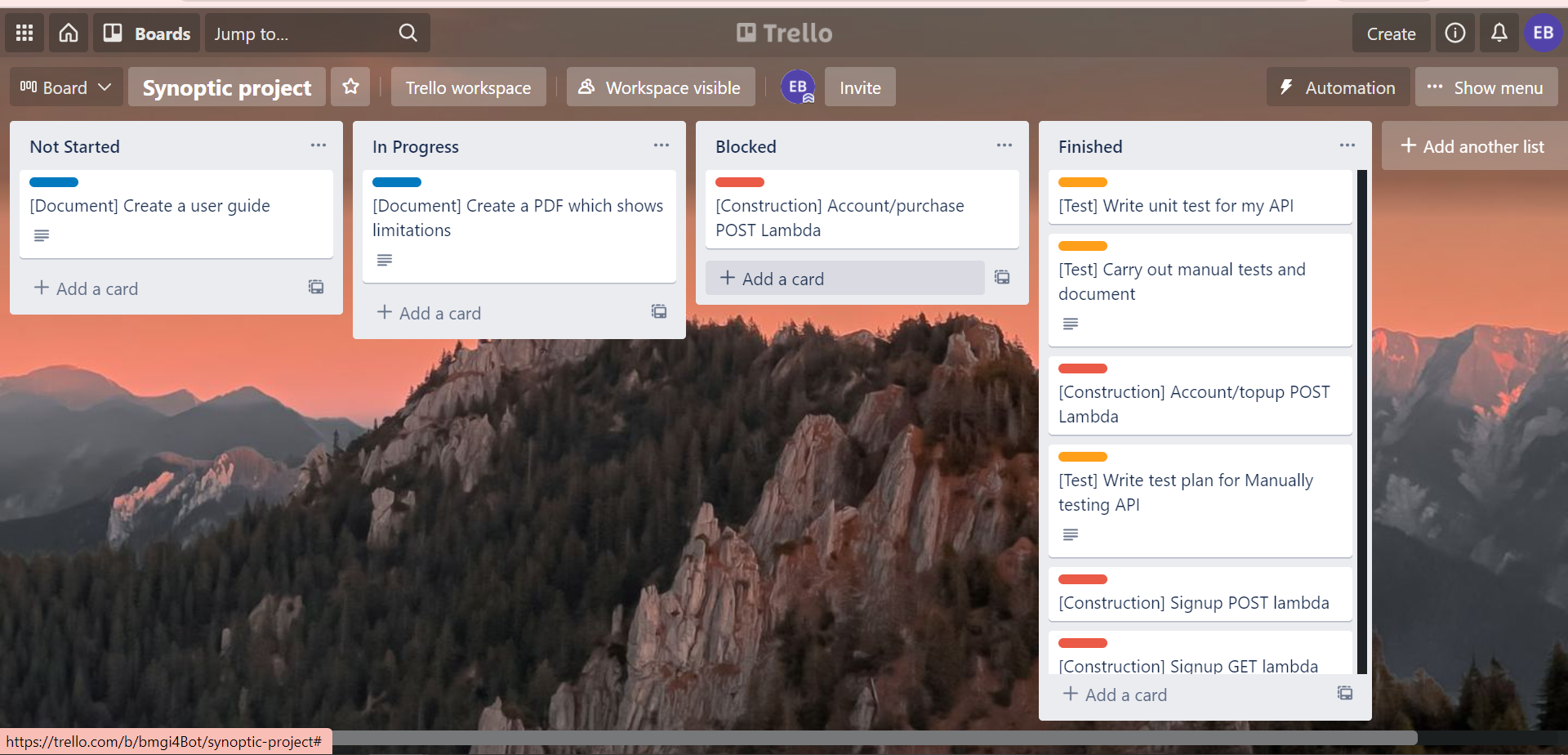
# Planning

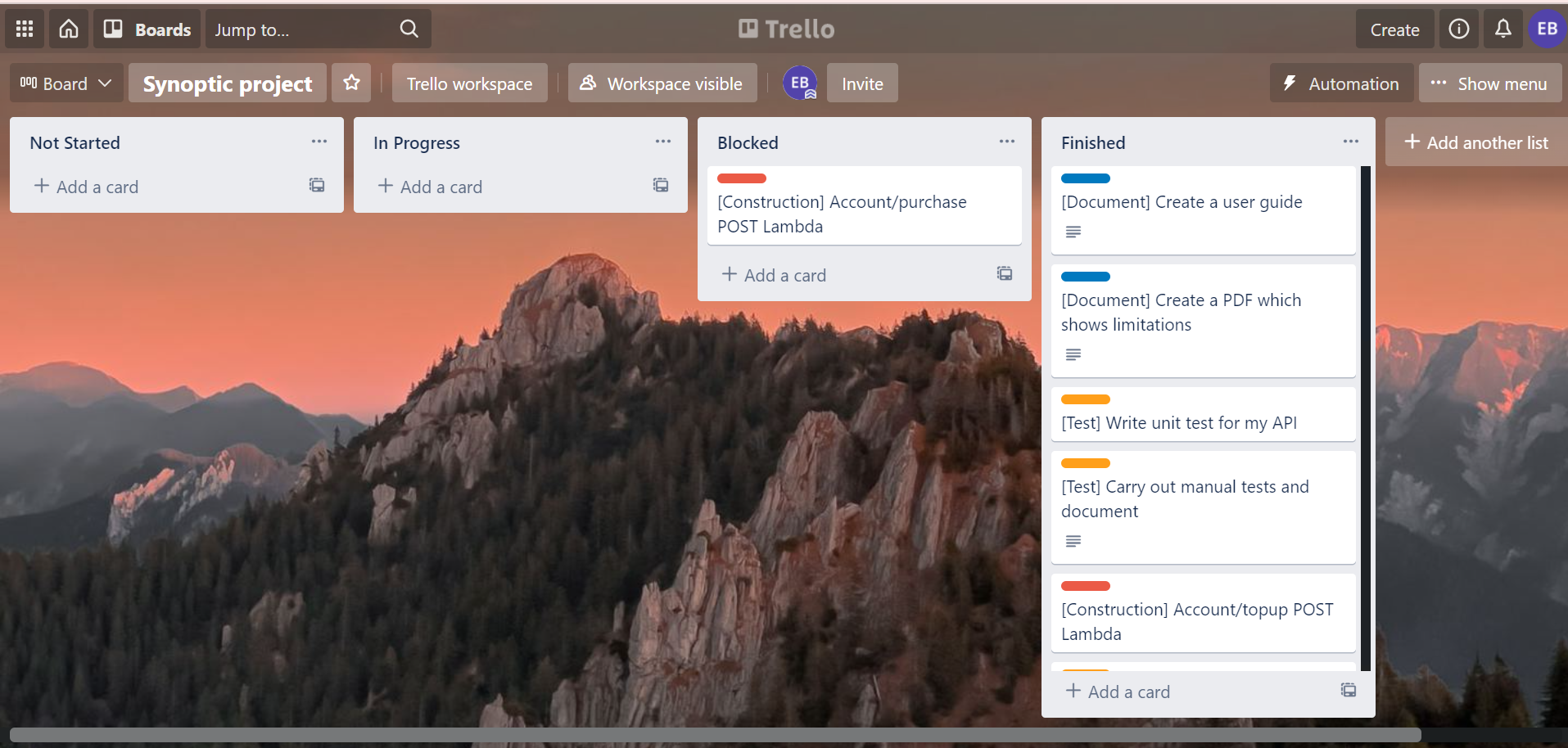
I created a Trello board, this allowed me to plan my project more effectively and break down each stage into tickets, this allowed me to be more focused when completing the project as I knew which elements I needed to complete next.





Once I had finished the design portion I knew what I wanted to build so I added additional tickets so that I could track the progress I was making and not get confused with the amount of work.

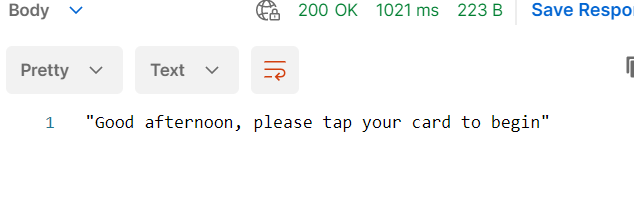
Once I had completed all of the development apart from the account/purchase POST response I started to work on the limitations.



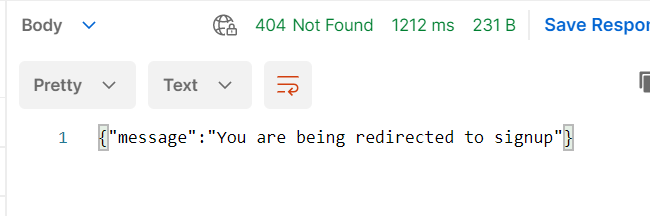
Finished project!

# How the implemented service should work with the client

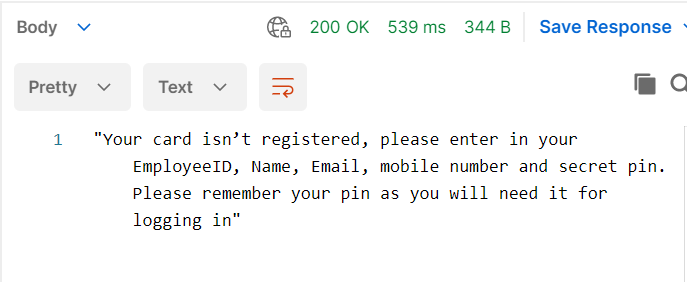
The user will be presented with a hello message from the kiosk:



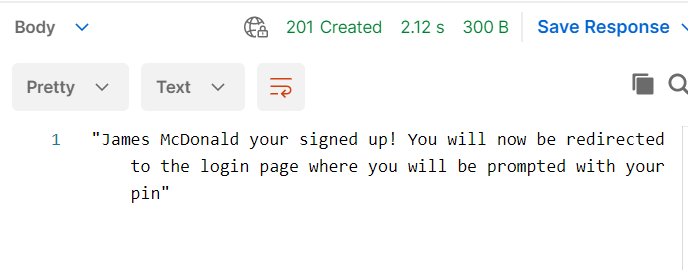
They will then tap their card onto the kiosk, this will collect the cardId and add it as a query string parameter onto the login endpoint:



The API will respond with a 404, this lets the GUI of the kiosk know that the user’s card is not registered, the message will be displayed as a new API request is made to the signup endpoint:

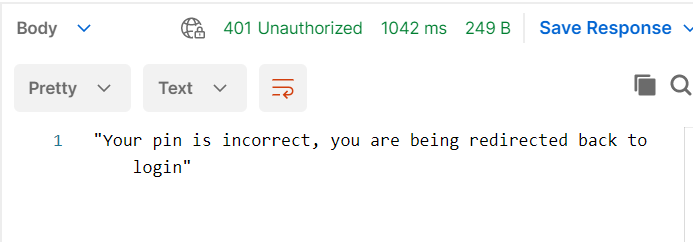


The service responds saying that the users card isn’t registered and they should enter in their employeeId, name, email, mobile number and pin number to get registered. This information will then be added to the body of the POST request to the signup endpoint.



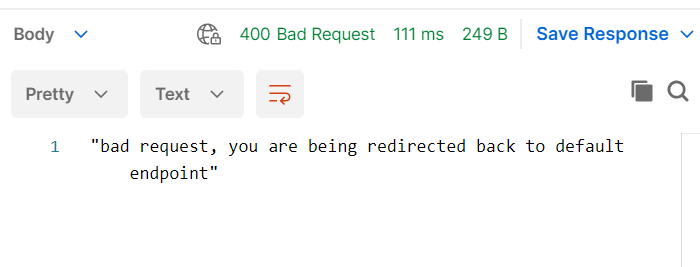
Once this request has been made the API will respond with a 201 created response, this will tell the GUI to divert back to the login page so that the user can log in.

If they enter in an incorrect pin they will get this response:

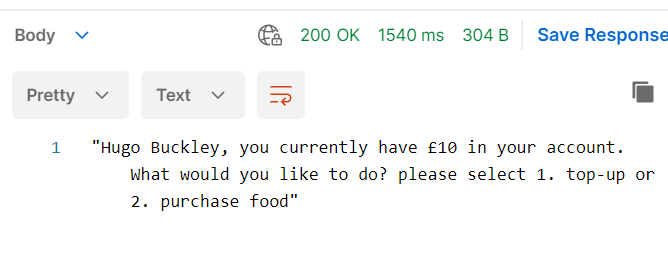


When the GUI receives a 401 response it will know to return to the login stage.

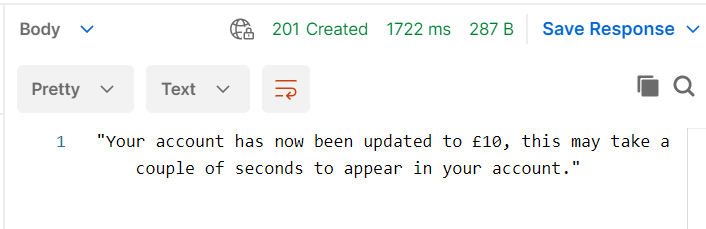
If the user doesn’t enter a pin or their request is malformed and doesn’t have a cardId then it will respond with 400, this will tell the GUI to return to the default endpoint.



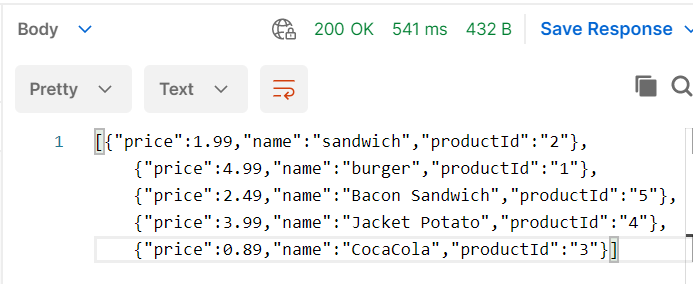
If they enter in the correct pin at signup they will get this response:



If they select the first option to topup they will be asked to enter in a value that they want to top up by, a POST request will then be made to the account/topup endpoint with the amount in the body and the cardId and pin of the account that should be updated. The endpoint should reply with a 201 response and a successful message. If this happens then the user will be redirected back to the account endpoint.

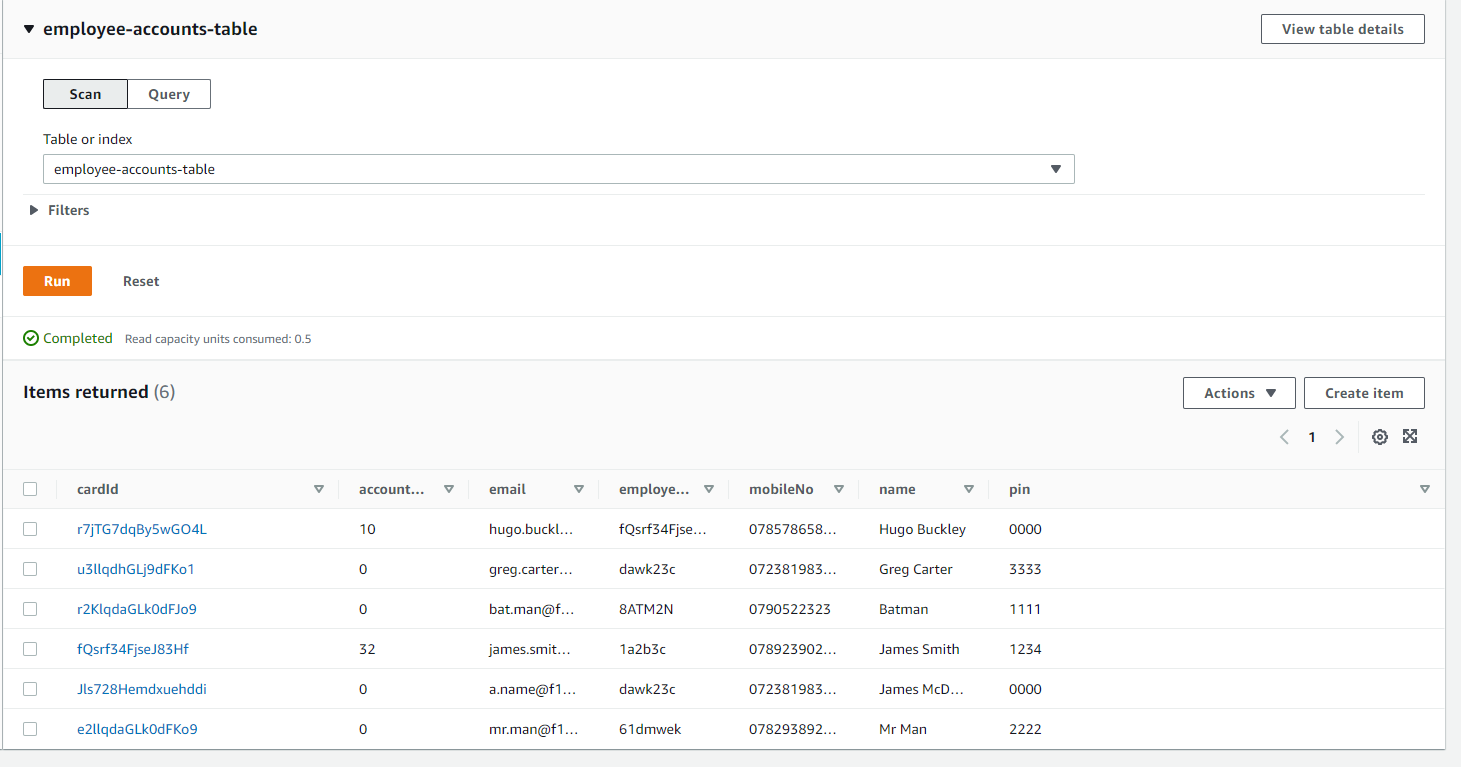


Where if they choose purchase food a GET request will be sent to the account/purchase endpoint, which will respond with a list of food items that can be purchased, the GUI can then format these into a list for the user to choose from

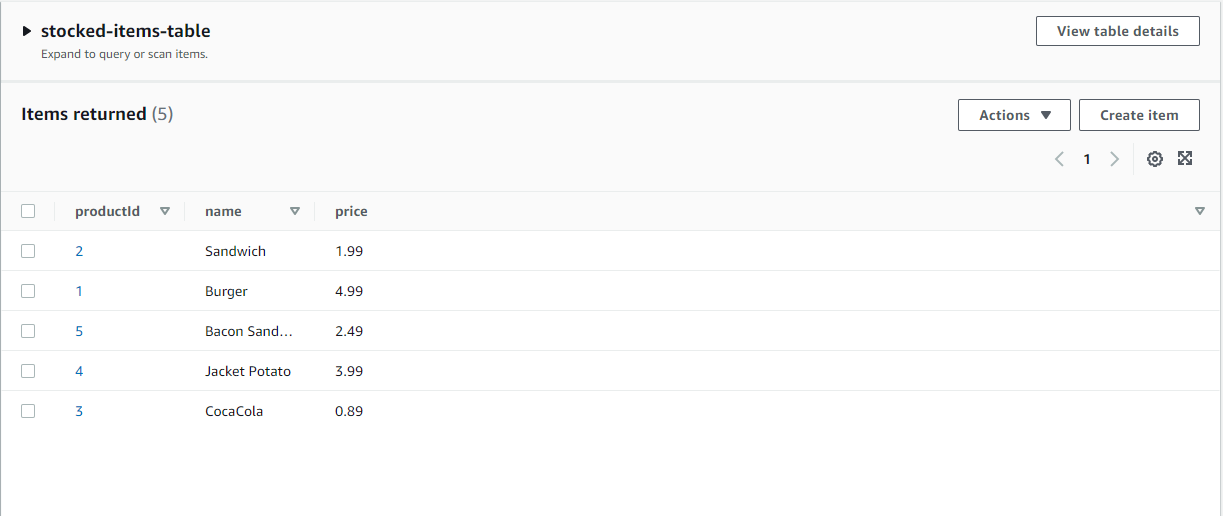


If the API was finished they would also be able to make a POST request with the item they want to purchase into the body and this would check they have enough money in their account to make the purchase as well as removing the amount from their accountBalance.

## Databases



This database stores all of the employees details.



This database stores all the products that are in stock.

# Card Service Limitations

## Limitations of design

The design of the project was incredibly thorough as I wanted to plan everything down to what type each entry into the database would be. A limitation of this design is that I was unsure when I was designing it how API gateway would route requests into my lambda, if I would need multiple lambdas or just a single one, If I had known this beforehand then I would have planned out how I would like to split up the routes into different lambdas as this ended up being the easiest way to implement my API. I would have instead added the login lambda to the account route as it uses the same method of querying the database as this would make the code more DRY (Don’t Repeat Yourself).

I also would have added the default/hello endpoint and the goodbye endpoint together as these were simple lambdas and using the same lambda would stop complexity.

I should have also discussed what errors were going to be thrown by each lambda and what would happen to cause the error state to occur. Error states that I handled inside my API were:

* 400 if the request they sent was malformed
* 401 if the user entered in an incorrect pin (Unauthorised)
* 404 not found requests if a user inputted a pin that wasn’t valid for their account.

## Limitations of Implementation

The limitations of what I was able to create in the time did not cover all error scenarios that could possibly happen in the API. For instance, the purchase GET response can only respond with a 200 response, the API should also be able to respond with a 404 if no data could be found in the database.

I would have liked to cover more edge cases and unhappy paths with the unit tests, this is so that I can figure out bugs earlier before release of the API and ensure that the unhappy paths are responding in the manner that I expect.

There was an endpoint that I did not get to implement as part of this project: POST account/purchase. This endpoint would have been easy to set up as I am already connected to the database, already implemented a POST request, and already have a put request to a database. The endpoint is already set up on API gateway so you are able to make this request to my API, but it responds with a message saying the endpoint hasn’t been set up yet.

In my design stage I outlined that I wanted certain endpoints to reroute to others automatically, I did look into this whilst working on this project and tried to implement it however I would have had to use a express server for routing which was taking too long to implement and would have cut into the development time for the rest of my application so I made the decision to have the client reroute using responses from the API. I unfortunately was unable to implement the timing out functionality because of this.

## Future Improvements

I would firstly implement all the elements I was not able to finish as part of my initial development. I would also like to add in a way to track the stock; I would have to add in a stock value into the stocked-items-table, and the user would need to request a certain amount of a item in their GET request. The user would also not be allowed to request more stock than the database currently has so I would need to do error handling around this.

I would also like to implement an API key into my API, this is easy to implement as part of API gateway however I did not want to stop the examiner from accessing my API. This would stop attackers from attacking the system using SQL injection or DDOSing the server.

I would also like to stop SQL injection by checking all requests for any code being entered, this would only allow through allowed values like names and emails.

I also would have liked to add in a JWT cookie to be passed around as the login details instead of passing around these parameters in the URL as these are easily viewable.

# User Documentation

## Path = ‘/’ DEFAULT

|  |  |  |  |
| --- | --- | --- | --- |
| ACTIVITY | METHOD | URL | INPUT |
| User is starting the journey | GET | https://dol4161gx7.execute-api.eu-west-1.amazonaws.com/ | No parameters |

## Path = ‘/LOGIN’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACTIVITY | METHOD | URL | INPUT (QUERY STRING PARAMETERS) | |
| User taps card on reader and the API checks if the card is registered | GET | https://dol4161gx7.execute-api.eu-west-1.amazonaws.com/login | Property Name | cardId |
| Example | fQsrf34FjseJ83Hf |

## Path = ‘/ACCOUNT’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACTIVITY | METHOD | URL | INPUT (QUERY STRING PARAMETERS) | |
| User has typed in their pin and is directed to their account page; their account balance is displayed, and they are asked if they want to top up or purchase items | GET | https://dol4161gx7.execute-api.eu-west-1.amazonaws.com/account | Property Name | cardId |
| Example | fQsrf34FjseJ83Hf |
| Property Name | pin |
| Example | 1234 |

## Path = ‘/ACCOUNT/TOPUp’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACTIVITY | METHOD | URL | INPUT (QUERY STRING PARAMETERS) | |
| User has inputted the amount they want to top up by. | POST | https://dol4161gx7.execute-api.eu-west-1.amazonaws.com/account/topup | Property Name | cardId |
| Example | fQsrf34FjseJ83Hf |
| Property Name | pin |
| Example | 1234 |
| **INPUT (BODY)** | |
| Example | {      "updateBalance": 10.00  } |

## Path = ‘/ACCOUNT/Purchase’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACTIVITY | METHOD | URL | INPUT (QUERY STRING PARAMETERS) | |
| User wants to buy an item and they want to know what items there are to purchase and how much they are | GET | https://dol4161gx7.execute-api.eu-west-1.amazonaws.com/account/purchase | Property Name | cardId |
| Example | fQsrf34FjseJ83Hf |
| Property Name | pin |
| Example | 1234 |

## Path = ‘/signup’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACTIVITY | METHOD | URL | INPUT (QUERY STRING PARAMETERS) | |
| User is not signed up; they are prompted to enter in details to sign up | GET | https://dol4161gx7.execute-api.eu-west-1.amazonaws.com/signup | Property Name | cardId |
| Example | Hh92dkaJakO82bJ9 |

## Path = ‘/SIGNUP’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACTIVITY | METHOD | URL | INPUT (QUERY STRING PARAMETERS) | |
| User has entered in the details they want to sign up with | POST | https://dol4161gx7.execute-api.eu-west-1.amazonaws.com/signup | Property Name | cardId |
| Example | Hh92dkaJakO82bJ9 |
| **INPUT (BODY)** | |
| Example | {      "employeeId": "12345",      "mobileNo": "0787216458",      "name": "BCS Examiner",      "pin":"0000",      "email": "bcs.exam@f1.com"  } |

## Path = ‘/Goodbye’ users name known

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACTIVITY | METHOD | URL | INPUT (QUERY STRING PARAMETERS) | |
| The user has timed out on their session or has tapped their card again to log out | GET | https://dol4161gx7.execute-api.eu-west-1.amazonaws.com/goodbye | Property Name | name |
| Example | Pikachu |

## Path = ‘/Goodbye’ Users name not known

|  |  |  |  |
| --- | --- | --- | --- |
| ACTIVITY | METHOD | URL | INPUT |
| The user has logged out before the user’s name is found | GET | https://dol4161gx7.execute-api.eu-west-1.amazonaws.com/goodbye | No parameters |