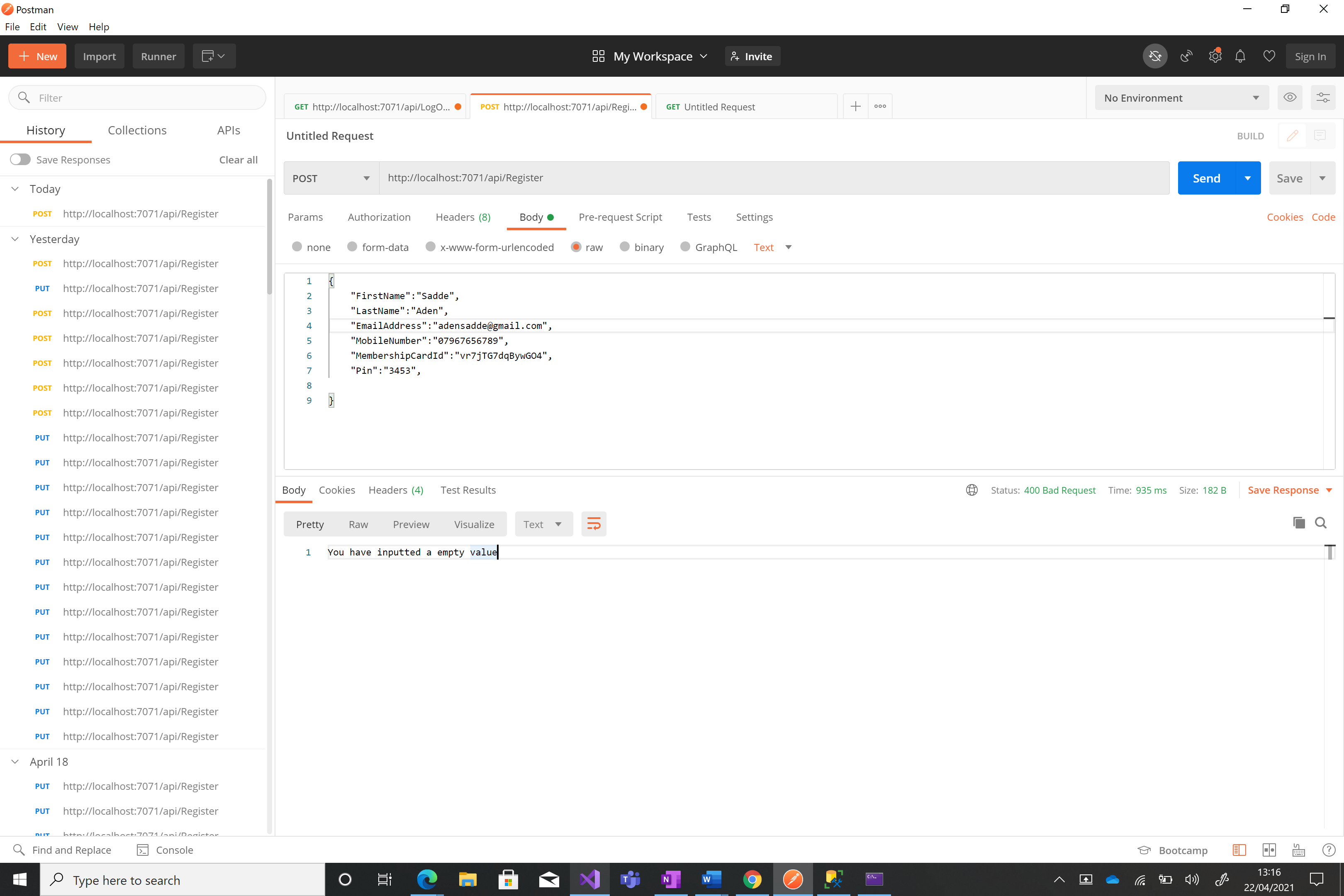
## User Guide

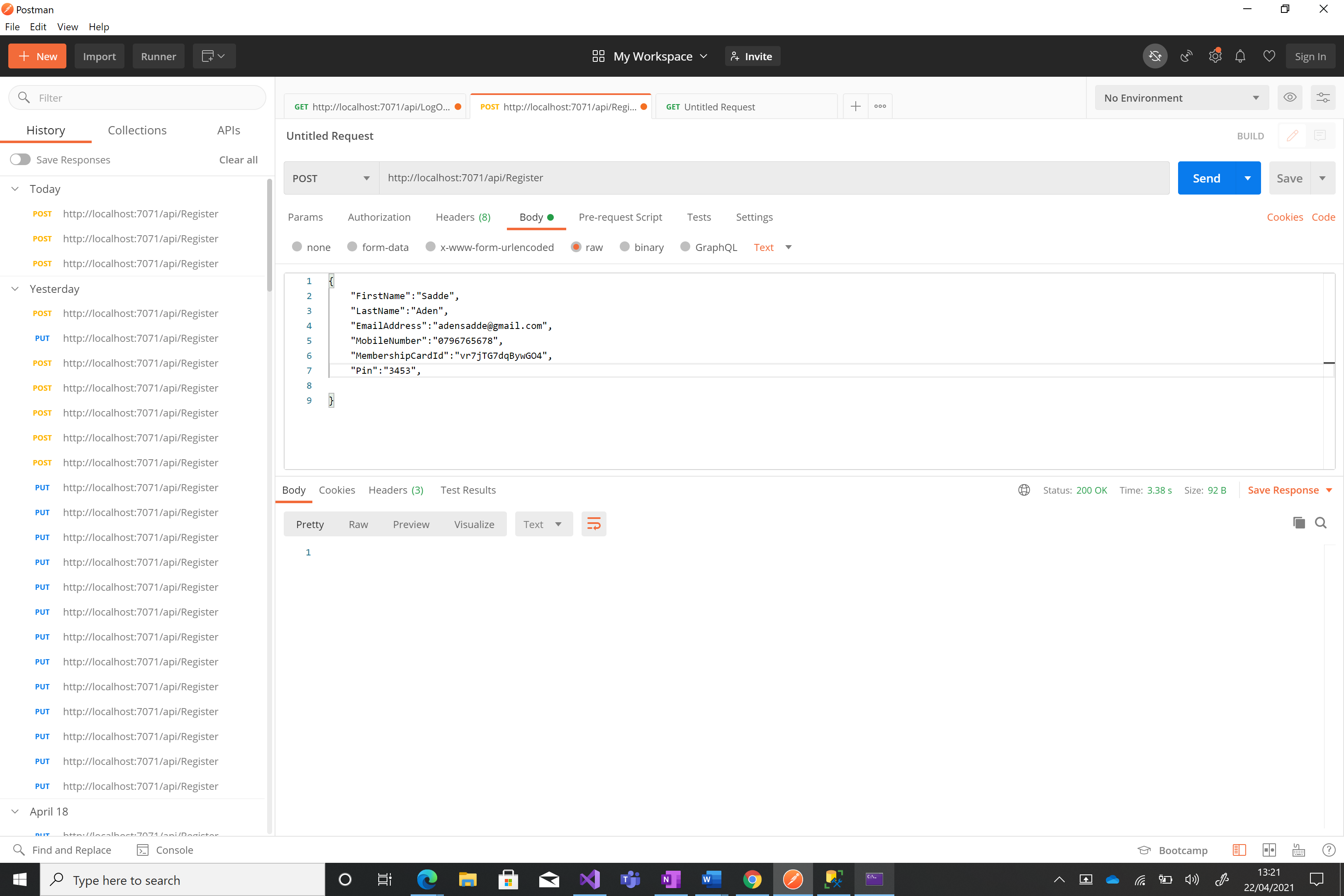
### Installation

The project is written in C# and needs visual studio to run, it also needs Microsoft SQL server. I have hard coded the connection string directly into the database context class in the repository so the user will have to alter the connection string to match the server name of the SQL server on their computer.

### Register

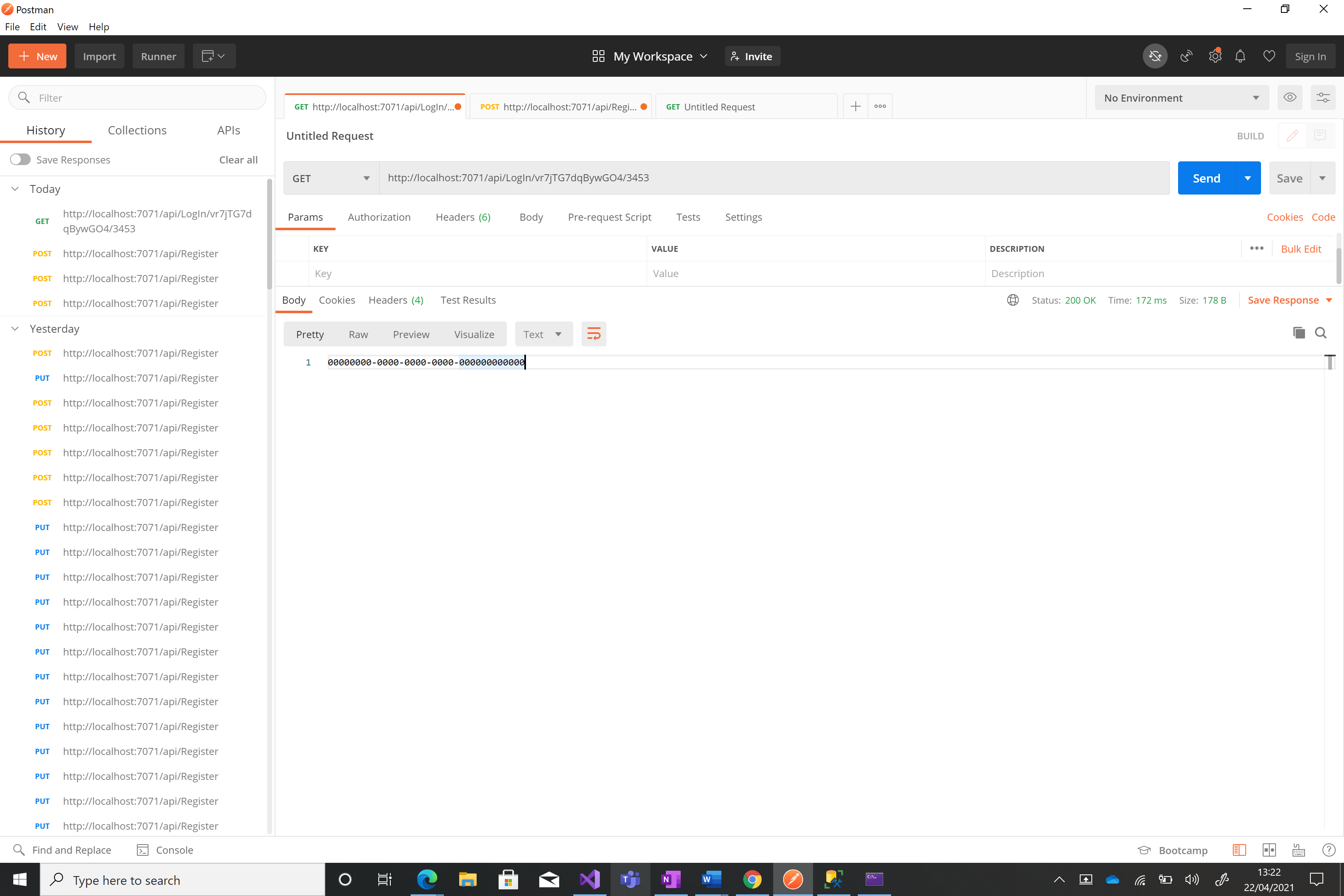
The registration function takes in a JSON object which will need to be posted to the API. Every value will need to be populated to retrieve a ok response and the kiosk is prompted on what is incorrect. This meets the requirement that the user needs to be able to register and log in using an existing card. There are validations on any of the values being null, the email address being a correct email format and the membership card must be a unique string of 16 characters which only contain numbers or letters.





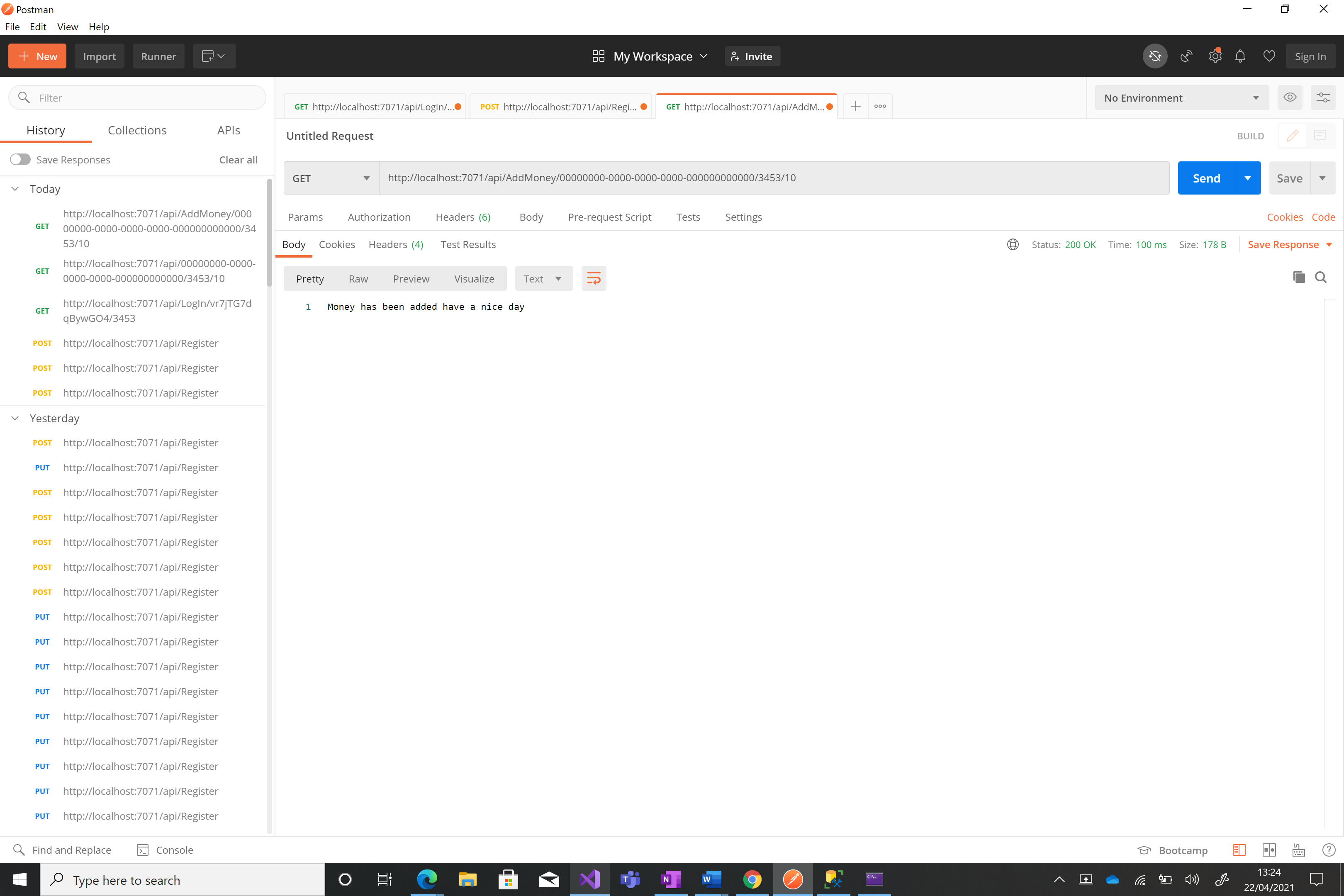
### Login

The kiosk will have to send the membership card followed by a / and the pin in the request string. This will return a session token which will be used for the following API’s until the user logs out. This will meet the requirement of the employee being able to login and use the system. There are validations on the guid type imputed for the session token and the pin number to be a correct type and to be correct to avoid an authentication error.



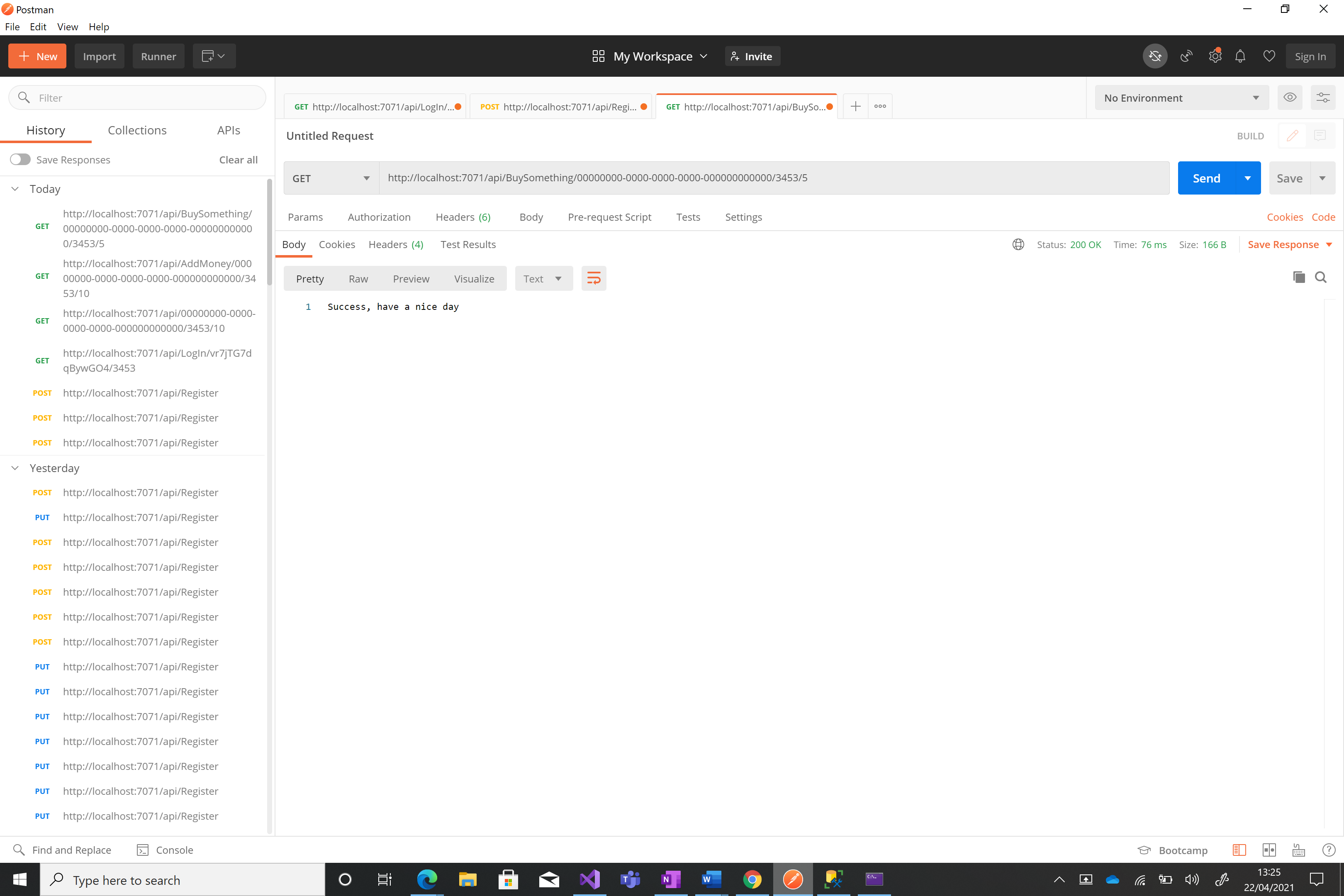
### Add Money

The kiosk will have to use the stored session token to make a request to add money. This will be followed by the pin number for the card and then the amount to add. This will meet the requirement of the employee being able to add money to their card. There are validations on the guid type imputed for the session token and the pin number to be a correct type and to be correct to avoid an authentication error.



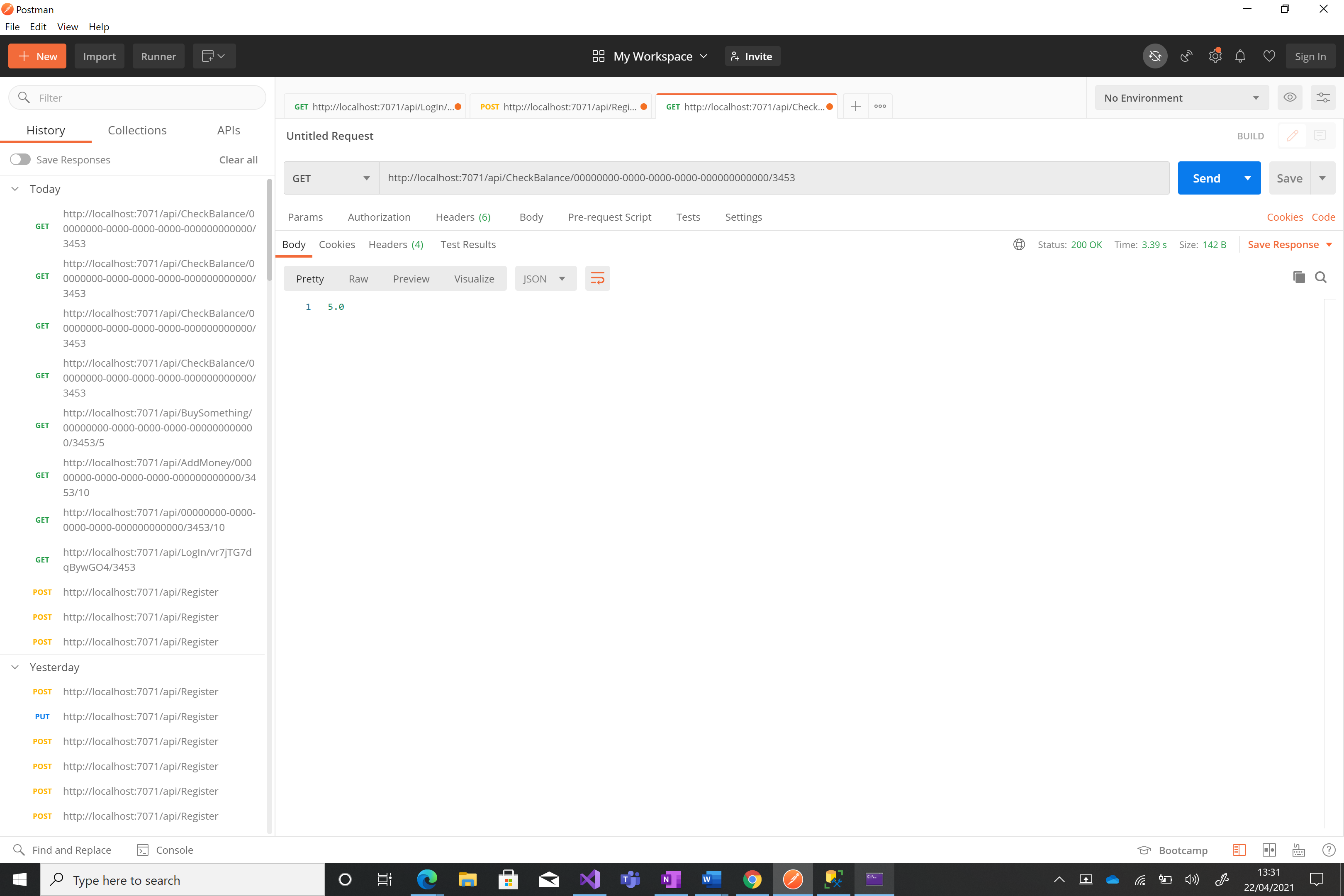
### Buy Something

The kiosk will have to use the stored session token to make a request to buy something. This will be followed by the pin number for the card and then the amount to add. This will meet the requirement of the employee being able to buy items using their employee card. There are validations on the guid type imputed for the session token and the pin number to be a correct type and to be correct to avoid an authentication error.



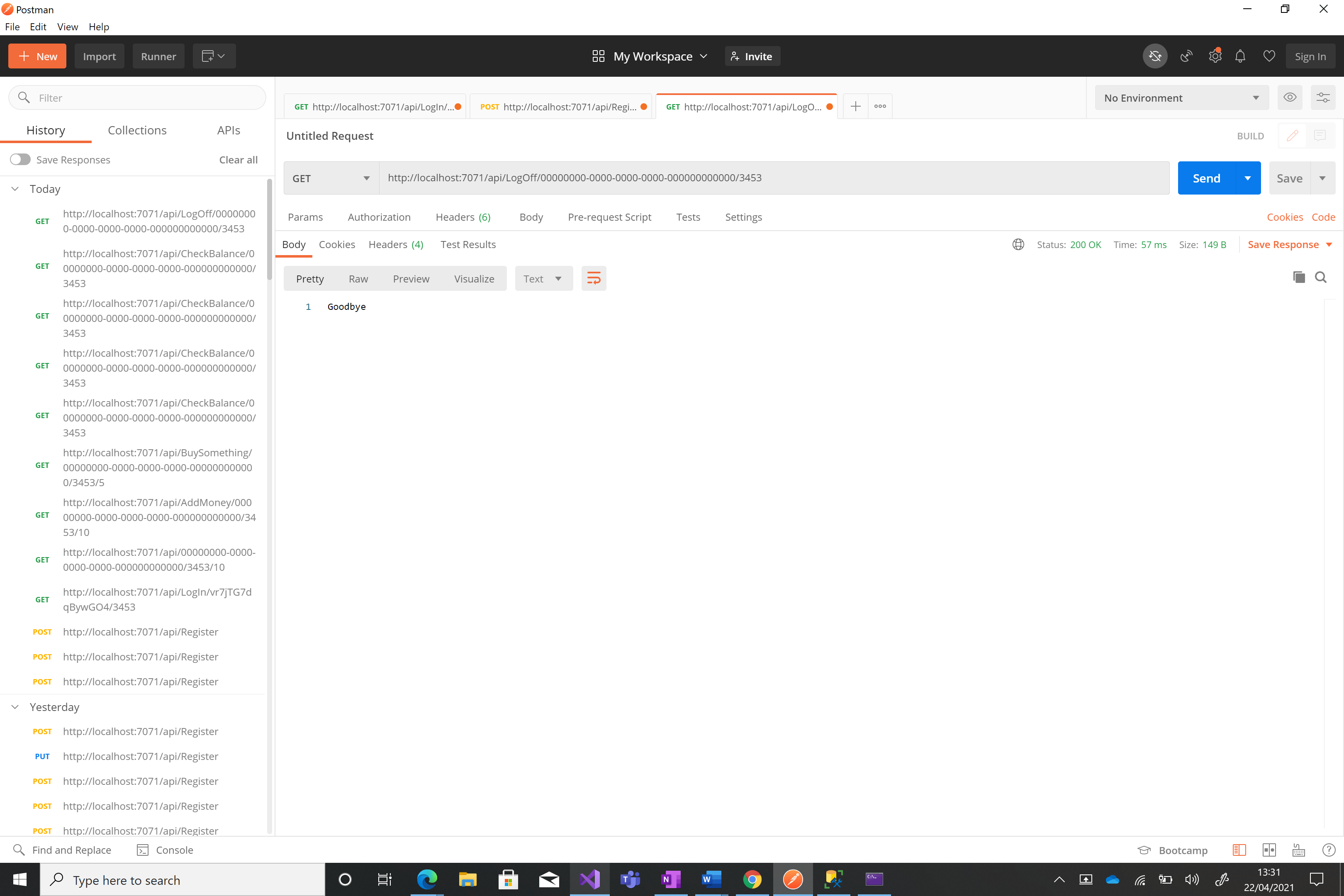
### Check Balance

The kiosk will have to use the stored session token to make a request to check their balance. This will be followed by the pin number for the card. The balance of the card is returned. This will meet the requirement of the employee being able to check their balance. There are validations on the guid type imputed for the session token and the pin number to be a correct type and to be correct to avoid an authentication error.



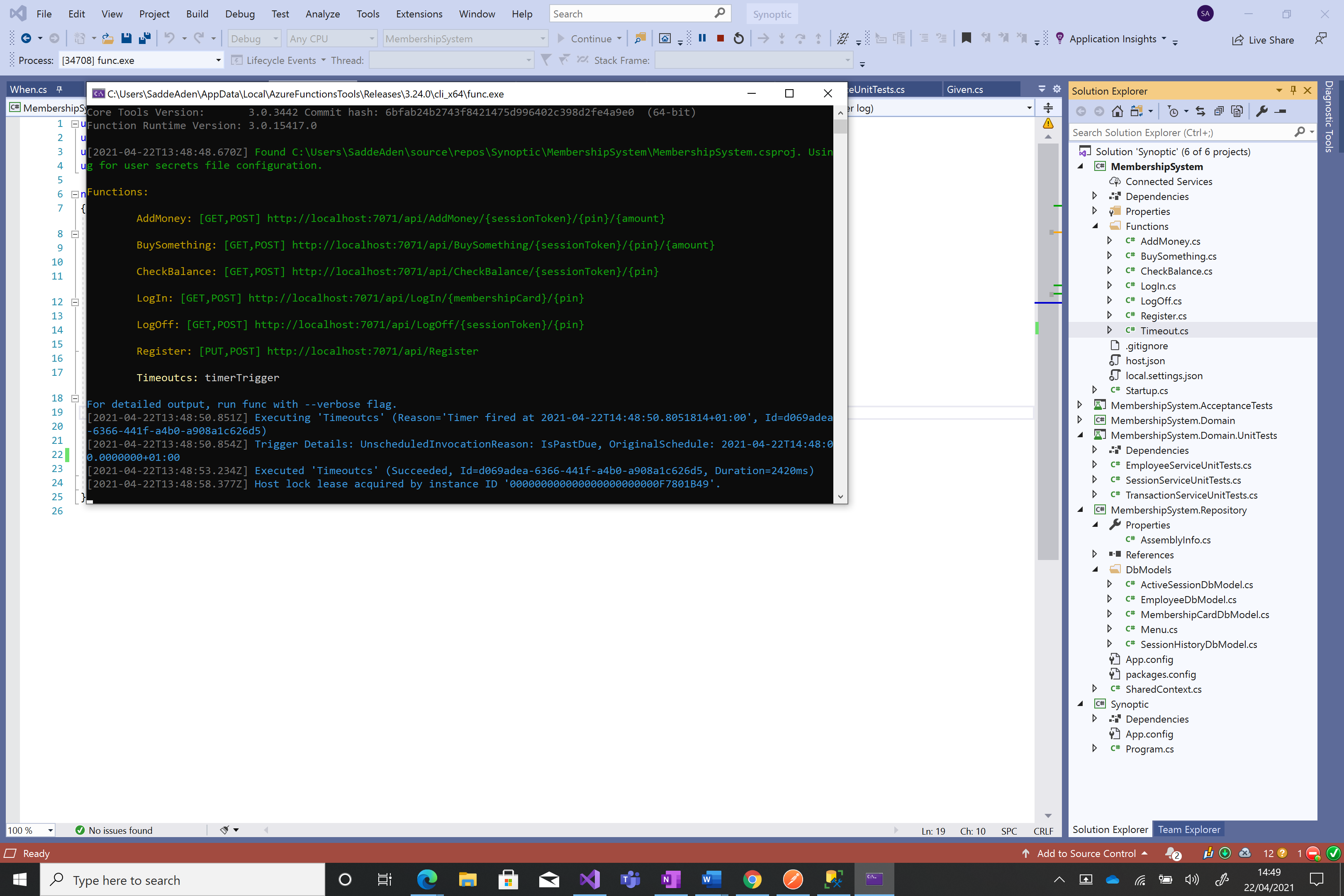
### Log off

The kiosk will have to use the stored session token to log off. This will be followed by the pin number for the card. A message is returned allowing the kiosk to know this was a success. This will meet the requirement of the employee being able to log out of the system. There are validations on the guid type imputed for the session token and the pin number to be a correct type and to be correct to avoid an authentication error.



### Timeout

The timeout function is different from all the other functions. This can not be requested and runs on a timer. Every two minutes the function is run and it checks all the active sessions in the database. It will check if they haven’t made a request for over ten minutes and if so they are logged out and the session is moved to the session history. This meets the requirement for the users session to timeout after a given time, the time I have used is 10 minutes.



# Conclusion

## What I would have added

There are many features I would have added or further improved if I had the time in this project. These include:

#### Hashing passwords

One of the biggest is hashing passwords and storing hashed passwords instead of in plain text. The system would save only the hashed password and then when the user attempts to make a request the hash would be sent instead of the real password increasing security.

#### Encryption

Another security feature I would have implemented is encrypting requests using asymmetric encryption. The kiosk to encrypt the request with a public key which can then be decrypted by the server. This would increase security and validate that the request is made by an actual kiosk and not by a hacker.

#### Sanitising the user inputs

Although entity framework does a good job of providing a layer between the SQL commands and the code I would have further validated user inputs to ensure the integrity of the database and reduce risk of malicious code being inputted.

#### Acceptance Tests

Although I have acceptance tested in this project I would have added more tests focusing on more unhappy path scenarios and the tests at the moment use entity framework to configure the database before the test however I would have liked to run the acceptance tests completely

#### Dependency injection

I attempted to use dependency injection in this project however I couldn’t figure out how to inject entity framework 6 and the database context wasn’t being assigned when the service layer was created so I had to remove it. I would have also liked to include this.

#### Locking

I would have added locking which makes sure that transactions could not be interrupted in progress.

#### Culture

I use datetime in the system and didn’t know how to use specify a uniform culture so this is something I would change if I had more time. Use one culture for the whole system to avoid issues.

#### Architecture

Due to the time limit I cut some corners I referenced domain model objects in the function when I would have liked to create separate view models which would then be auto-mapped to the domain models and referenced database models in the domain layer instead of creating data transfer objects like I should have to further loosely couple the code.

#### Transactions

I would have liked to use transaction scope to make sure that no changes are made to the database unless the whole transaction of removing the previous entity and adding the updated one is completed however was having issues implementing it using entity framework.

#### Menu

Currently my system takes in an amount an item costs when buying an item and this is subtracted from the users card balance. This would mean a employee would have to man the kiosk entering the amount to subtract from peoples cards. Ideally I would have a menu table with a pre-set menu in which a user would pick a menu option which would then be subtracted from the users balance. This would cause less issues and provide less room for error.

#### Swagger

Another change I would have loved to make was to use a swagger front end instead of azure functions to make it easier to display how to use the API’s.

#### Database

The last change I would have made was correctly setup the database. At the moment there is a relationship between two of the tables, but I would have further connected the tables and normalised the database to make it more efficient when being queried. I would have also included the database connection string in a configuration file and not in the class to make it easier to change.

## Limitations

There are a few limitations with my code. The first one is the lack of test coverage for different scenarios where money is being added and subtracted. This has led me to be unsure of if there are bugs I haven’t found in this like limits of double type or what the system does given negative numbers. Another limitation is how brittle my tests are, I had to constantly be updating my tests when I changed functionality so they weren’t as useful as I would have liked them to be.

## Summary

In conclusion I believe that I have meet all the requirements mentioned in the brief.