# Introduction

A small New Zealand company named ‘NZ Groceries’ has been affected by the alert level restrictions in the Country. ‘NZ Groceries’ is a small supermarket store based out of Hamilton, New Zealand. They supply a large portion of the local businesses with supermarket items, such as, Meat, Confectionary, Dairy etc.

The alert level restrictions in New Zealand currently ask NZ residents to remain home, and all ‘non-essential’ businesses to close until the alert levels drop down to a lower level, resulting NZ Groceries needing to close their doors.

NZ Groceries also supplies supermarket items to local Hamilton residents; however, NZ Groceries does not have a website that allows customers to browse and shop online.

NZ Groceries has reached out to me and asked me to develop a full-stack website for them that will:

* Allow their customers to log in to or register an account, with password hashing.
* Browse the full NZ Groceries catalog.
* Allow customers to browse and search for items.
* Each item must display the name, price and allow a customer to add to their cart.
* Checkout their cart.
* View their order.

NZ Groceries has also requested that I introduce API features, these will include API endpoints that will allow Admin to have superior control over the website, compared to a customer.

An Admin will have access to:

* Logging in as an ‘Admin’ account, with hashed passwords.
* Add new products.
* Edit existing products.
* Update product availability.
* View all current orders.
* Fulfill orders.

For this project, I have decided to use MongoDB as the database technology, Express.js as my back-end web application framework, React.js as my front-end library and Node.js as my runtime environment.

I have decided to use the ‘MERN Stack’ as it is a fairly modern approach to this project, and it will be a great way for me to learn more about each technology.

# Technical Report

In this section I will talk more about the API endpoints, as well as the routes, middleware, handlers and I will include JSON test data that can be tested using the ‘Postman’ software and changing the request type accordingly.

## API Endpoints

All API Endpoints have been specified by adding a request handler that requires ‘/api’ before each route, this will help to distinguish between API routes and public routes.

### Customer register and log in

I have created two ways to add a customer record to the MongoDB Database. The first way is using the API endpoint, using Postman, create a POST request to the address:

localhost:4000/api/customers

The POST request should also contain data, this data should be formatted in the ‘JSON’ format.

Below I have added some test data, however, this test data can only be used once as I have created rules that only allow the username and email to be used once.

{

“username”: “Ali”,

“email”: “Ali@gmail.com”,

“password”: “you choose”

}

Once the data above has been submitted, the customer account details will be added to the database and saved for later use.

Graphical user interface, text, application, email

Description automatically generatedThe other way to create an account is by navigating to the ‘sign up’ page and entering in your details to the input boxes. I have set up code in the server-side that will take your input information and transform it into a JSON object that will then be sent to my database, effectively working the same way as above, but increasing the usability for customers.

### Display Catalog

This API Endpoint will be useful when the front-end has been developed more. It will allow the website to display each instance of the ‘Products’ collection, once the code has been implemented.

Sending a GET request to:

localhost:4000/api/products/all

Will return a response containing information about the products in the Products collection currently.

### Display Customer Records

Similarly, to the ‘Display Catalog’ endpoint, Display Customer Records works in the same way. This endpoint will allow Admin accounts to fetch data about the current list of Customer accounts in the database, this will help the ‘Admin’ find accounts and help the deletion of an account.

Sending a GET request to:

http://localhost:4000/api/customers/all

Will return every Customer account instance in a JSON Format.

### Products API

I have created an API endpoint that will allow an Admin account to create and save a new product to the database, to access this API endpoint, send a POST request to:

http://localhost:4000/api/products

Below I have included test data, in JSON format:

}

"product\_name": "Ham",

"category": "Deli",

"price": 4.80

}

The Search Products API will allow a customer to search through the current catalog to find specific items and have them displayed on the frontend.

To access this API route, send a GET request to:

http://localhost:4000/api/products/615020e94ab3ee74de146215

This will return the ‘Chocolate’ item

### Admin To Update Customer Records

Another important feature that has been requested, is to allow an Admin account to update certain records within a customer’s account. I have set up an API Route that will handle this, send a PUT request to:

http://localhost:4000/api/customers/614fd4bb77fbfb246fd5f68b

The long string at the end is the customers id that is created when the account is created.

Make sure to include which field you would like to update; I have included test data below that will update the email address attached to my personal account and send a response with the account information updated.

{

"email": “Ali@gmail.com”

}

### Delete Customer Account

An Admin must have a feature that will allow the deletion of records and accounts. For this requirement, I have added an API endpoint that can be accessed by sending a DELETE request to:

http://localhost:4000/api/customers/614f02e1f80895d0105f7f24

The long string at the end is the customers id that is created when the account is created. Once you have sent the DELETE request, the server will send back a response containing information about the account that was deleted.

## Server Run Down

The webserver has been broken down into routes and handlers. Each route has a handler that will handle the specified request in a way that is set by the code I set it up with and send a response that I have set.

Diagram

Description automatically generatedTo help staff and customers understand the webserver more, I have created several diagrams that illustrate what requests are doing. Note, I have not created a diagram for each interaction between the User and the Server, I have created these diagrams to illustrate how everything is connect and how everything works.

This is a request handler diagram

Diagram

Description automatically generated

This is a simple routes diagram I have created.

Chart, diagram

Description automatically generatedDiagram

Description automatically generated

This is a basic technology diagram I have created to show what role each technology plays in this webserver

This is a simple walkthrough I have created to show where the webserver will talk to the database and why.