Backend NodeJS technical details

Contents

[Overview 1](#_Toc104466187)

[Endpoints 1](#_Toc104466188)

[Routes 2](#_Toc104466189)

[Controllers 3](#_Toc104466190)

[Error.js 3](#_Toc104466191)

[Models 3](#_Toc104466192)

[Receipt.js 4](#_Toc104466193)

[Index.js 5](#_Toc104466194)

[OCR output, MySQL parsing and recommendation logic 5](#_Toc104466195)

[Uploads 5](#_Toc104466196)

[Views 6](#_Toc104466197)

[Util 6](#_Toc104466198)

[Node modules 6](#_Toc104466199)

[Config 7](#_Toc104466200)

[Future steps 8](#_Toc104466201)

# Overview

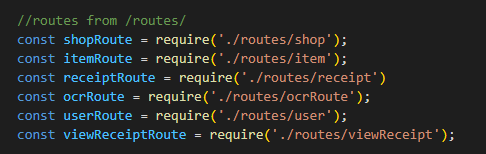
The Backend NodeJS server was designed with the MVC model in mind, this follows a model, view and controller for each database table or component that is going to be used in an API.

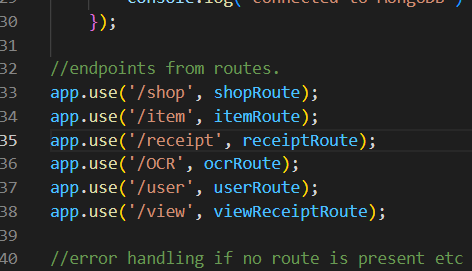
The model defines the class which should match the MySQL database table, the view should display information to the client and the controller should interact with the database.

Please refer to the backend install instructions in the GitHub documentation folder to get the backend system running.

# Endpoints

Currently there are endpoints for /shop, /item, /receipt, /OCR, /user, and /view, these all correspond to various related API through the routes scripts. The API can be seen in /dm\_app/documentation/ Backend API documentation.docx. These are called in index.js, which is the main file that is run when starting the system with “node index.js” in the terminal.

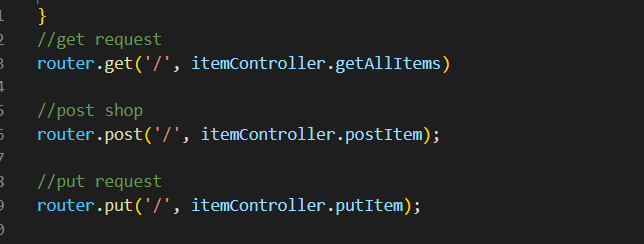


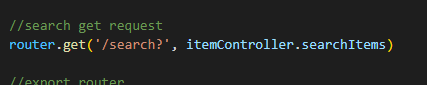


# Routes

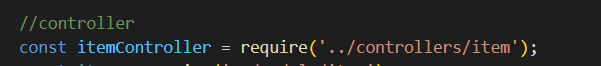
The routes are called from the endpoints mentioned above and are responsible for calling Get Post and Put requests for each API implemented, these are split into, item, ocrRoute, receipt, shop, user and viewreceipt.

This is an example of the item route script and how it calls exported functions from the associated item controller script.





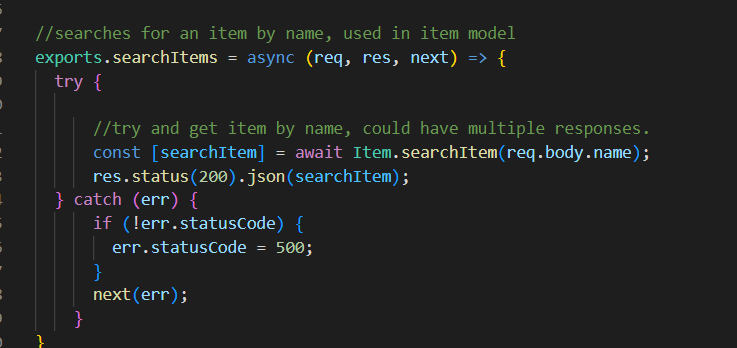
These are called by requiring the itemcontroller at the top of the script.



# Controllers

Controllers are called from the routes as an exported function and get data from the model script, controllers are responsible for controlling the requests and responses to the client for the model scripts to utilize.

As seen in this screenshot the function search items in the item controller requires a body.name variable from the client for the searchItem function in the item model, then responds with a response code of 200 and with the results in JSON format.

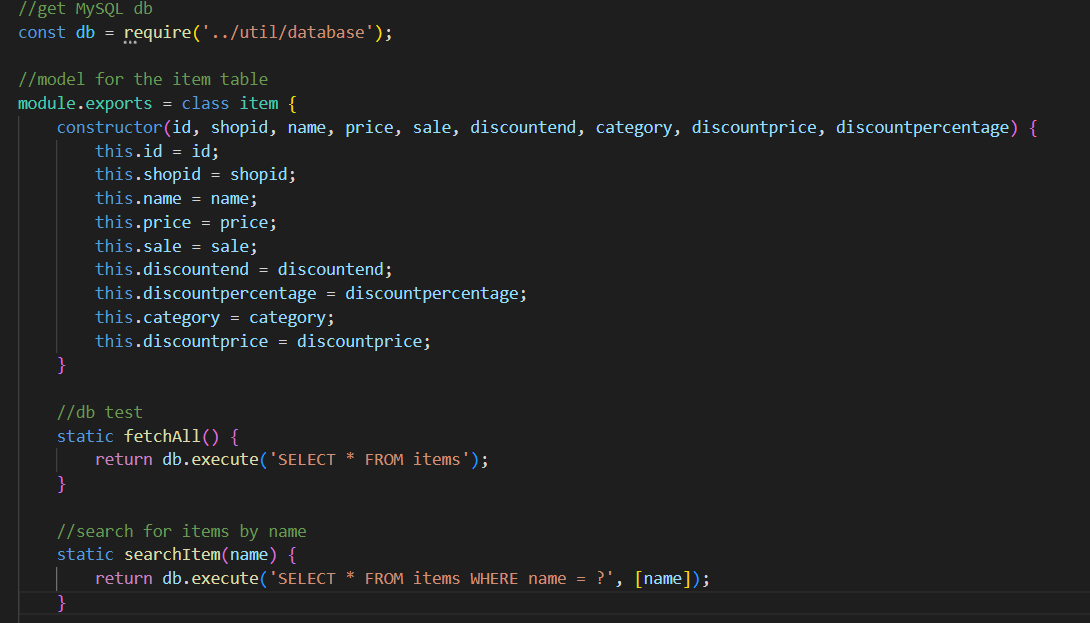


## Error.js

The /controllers/error.js file contains basic error handling when an error is not provided, this can be wrapped in a try catch block as seen in the controller example. This also provides the error message and code back to the client for debugging purposes if a response is not defined previously.

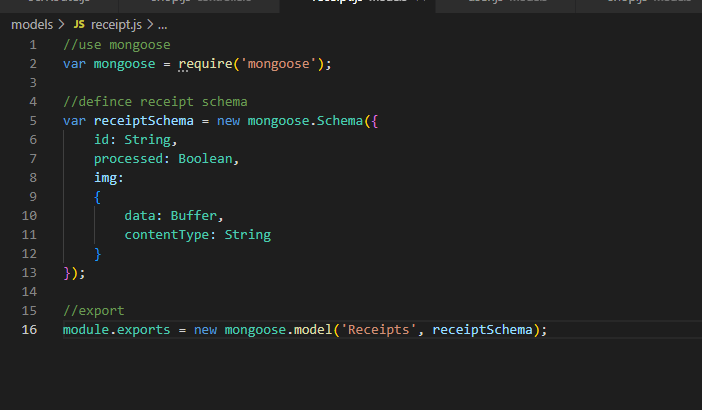
# Models

Models are called from controllers, this is where the database commands are called for the SQL database, and where the class is defined. These static functions are called asynchronously from the controller class as it takes some time to get responses from the database (notice the await Item.searchItem in the item controller) you can also pass variables from the controller to the model, in this instance body.name is used to search with a “name” variable in the MySQL database.



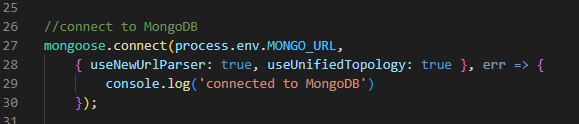
## Receipt.js

The only model that defers from this structure is the receipt.js model, which is defines the receipt schema for the MogoDB database. The schema is defined with an id, which is a string (currently is set to the date&time) The processed Boolean, which is set to true once the OCR script runs on an upload and the image which contains the data for the image and content type.



# Index.js

The Index.js is the file that the backend system start on, this contains all routes and required files, as well as where the bodyparser package is imported, the bodyparser allows the use of requiring body information from the client, for example req.body.name would require the client to provide a “name” variable from the body of a request. Mongoose is the package used to connect to the MongoDB database, this can be seen at line 27 in index.js



This uses the MongoDB connection string in the .env file to connect to MongoDB.

The application then listens on the ports provided in the “ports” variable at line 18. (from the .env file or 3000 by default)



# OCR output, MySQL parsing and recommendation logic

The OCR python script runs from the /routes/receipt.js file, this uses PythonShell to run the “t1\_2022\_ocr\_final.py” script in the /util/ folder with system arguments defined in “args” at line 387.

The first system argument is your local tesseract file path, this cannot be uploaded to GitHub as the file size is too large, the second argument is the WriteFilePath which is the temporary image (the one that has just been uploaded) that gets passed into the OCR script and the third system argument is the userID which is currently stored in the system environment file (the .env file) but will need to be updated to use the access key generated on user login.

The OCR script then runs on the image uploaded to the /receipt POST API and fills the “ocrtable” in the MySQL database.

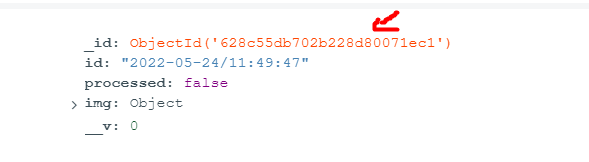
Once the OCR script has finished running it will call item.processed = true, which sets the processed Boolean in the MongoDB to true, saves then calls the FillSqlServer(); function at the top of the receipt.js file. This function is the function that inserts the data from the ocrtable in MySQL to the various other tables in the database.

# Uploads

This is where the uploads to the MongoDB are saved locally by /routes/receipt.js at the POST API call, this functionality is mostly for debugging for now and may be removed at a later stage. This folder also contains the uploaded image from /routes/receipt.js when the POST request is called to /receipt (at line 318) This image is temporary and self-deletes once the image has been processed by the OCR script and has been uploaded to the MongoDB database.

# Views

The views folder is where the debug web pages are for testing such as the test MongoDB upload which is called “index.ejs” and the “receipt.ejs” which lets you view an image from the MongoDB by referencing the ID



In this case it would be “628c55db702b228d80071ec1”

# Util

The /util/ folder contains miscellaneous scripts, including the packageInstall.py script which installs all necessary packages for the OCR python script (except for tesseract).

It also contains the generateAccessToken.js script, which uses a library called jwt to assign users access tokens, this script exports an access token and a refresh token which are both called when a user has logged in at /controllers/user.js at line 118.

And database.js which uses mysql2 to create a pool with the MySQL database and exports the pool promise.

# Node modules

The node modules folder contains all npm packages installed at the /backend/ directory. Currently we are using the following packages:



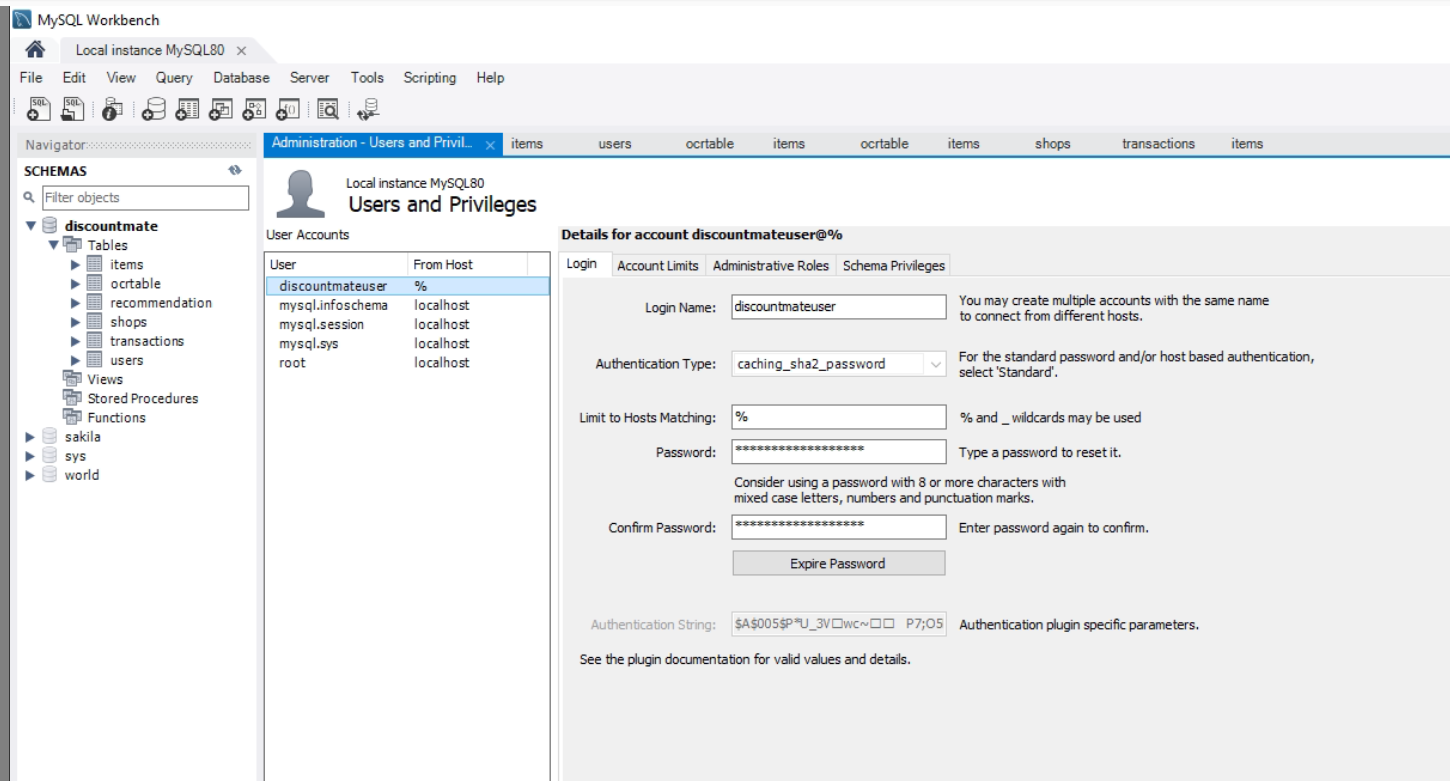
# Config

Config.json inside the config folder is where the details to connect to the MySQL server are stored, this needs to be updated to the new hosts IP address and port. They also need to ensure that there is a user set up on the MySQL database that can be accessed remotely and that the details in this file match the user details for access.

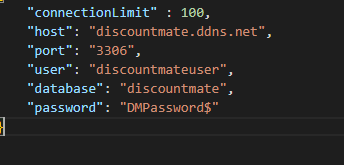
The “host” is the hosts IP address

“Port” is the port to connect to, this port will need to be port forwarded on the hosts router

“user” is the user that is set up on the MySQL database as such:



“password” which is the password for the above user.



# Future steps

1. Currently, the route for receipt.js does not follow the MVC structure and will need to be updated as all calls to the database and MySQL logic are competed in the route receipt.js instead of the controller/model.
2. The ocrRoute.js route file as well as the /OCR route may be removed as it was used to test the OCR function, but is now obsolete unless it is wanted for future testing.
3. The jwt access key should be used to obtain the current users username instead of storing it as an environment variable in the .env file as this would persist through sessions and would not be secure. This can be seen in the /controllers/user.js at line 116 and 117.
4. The jwt access key should also be used to obtain the userID which is passed to the OCR script at line 384, instead of storing it as a .env variable.
5. The logic for the MySQL table implementations will need to be updated as they are very basic.
6. The recommend GET request API in the /routes/item.js will need to be updated to use the MVC structure as well, as it’s function is called in the route js file when it should be in a controller/model.