**UNIEATS**

**divplusplus**

**API Specification**

The backend API for the UNIEats website is developed using Node.js and Express.js, providing a robust framework for building a RESTful API that seamlessly interacts with the React frontend and MongoDB database. Hosted on Azure Web Apps, this API ensures high availability and scalability, allowing for efficient handling of requests from the client-side application. The API exposes several endpoints for essential functionalities, including user authentication, menu retrieval, order placement, and feedback submission. Each endpoint follows RESTful conventions, utilising standard HTTP methods such as GET, POST to manage resources effectively. The integration with MongoDB facilitates a flexible and schema-less database structure, enabling dynamic content management and efficient data retrieval. To enhance security, JSON Web Tokens (JWT) are employed for user authentication, ensuring that sensitive data is protected during transmission. The API is designed with performance and maintainability in mind, featuring comprehensive logging and error handling mechanisms, allowing for easy debugging and monitoring of application health. This architecture provides a solid foundation for delivering a responsive and user-friendly dining services platform for the campus community.  
  
The api encapsulates all the core functionalities of the service such as menu access, fields for dietary management, meal credit system as well as voucher system, reservation and lastly feedback system.

Furthermore, all these functionalities are reflected on our mongodb database.

Some information about our api  
  
We have a few endpoints that are accessible to external systems such as   
app.get('/api/viewRestaurants', async (req, res) => {

// Retrieves all restaurants from the database using a GET METHOD

try {

const restaurants = await restaurantModel.find({}).select('name description -\_id');

res.status(200).send(restaurants);

} catch (error) {

res.status(500).send({ message: 'Server error processing the request', error: error.message });

}

});

app.get('/api/viewLocations', async (req, res) => { //takes in a restaurant and checks the menuitems table for entries from that restaurant and return all of them for that restaruant

try {

const locationItems = await locationModel.find({});

res.status(200).send(locationItems);

} catch (error) {

res.status(500).send({ message: 'Server error processing the request', error: error.message });

}

});

Allowing for other teams to integrate with our software   
  
Furthermore, we have implemented node-cron that allows the backend to automatically check if any orders are ready. Cron queries the database every minute to check wether or not there are orders that are ready for collection   
If the order is ready for collection, it will be updated to such. Moreover, we have used nodemailer to send emails to clients once there orders are ready for collection

We have also incorporated thorough error handling in our backend using response status such as 404,401 and 500  
  
  
Below is the documentation of our internal api endpoints   
  
/signUp  
This endpoint registers the users userID into our database from auth0 allowing us to keep track of user information and credits. If the user is already in the database, it does not add them again.

/viewRestaurants  
Retrieves all restaurants

/addRestaurants   
self-explanatory

/addCredits   
Takes the userID and voucher code as parameters it then checks the value for that voucher and adds that amount to the user’s credits in the user table

/viewMenuItems  
Queries menu table with respect to a certain restaurant

/addReview  
Adds a review to the review table with items like rating, comment, userID and restaurant. Given the rating of the review that rating is then included in calculating the average rating for the restaurant that gets displayed.

/completeOrder

Once order has been collected by customer, they will click the order has been collected   
And this will trigger this end point   
Since this is a mock up there is no restaurant side processing, so the order automatically gets made to ready for collection after a certain amount of time. Thereafter the user can click order collected. This is done to simulate as best as possible a real-life situation by where the restaurant will continuously update the order status from there side   
  
  
/addOrder

Adds the order to database with the status to as pending

/viewReservation

based on userID

/deleteReservation   
Based on reservation ID

/viewReviews   
Views all reviews for a particular restaurant   
  
/viewUser  
Takes userID as parameter and returns user information