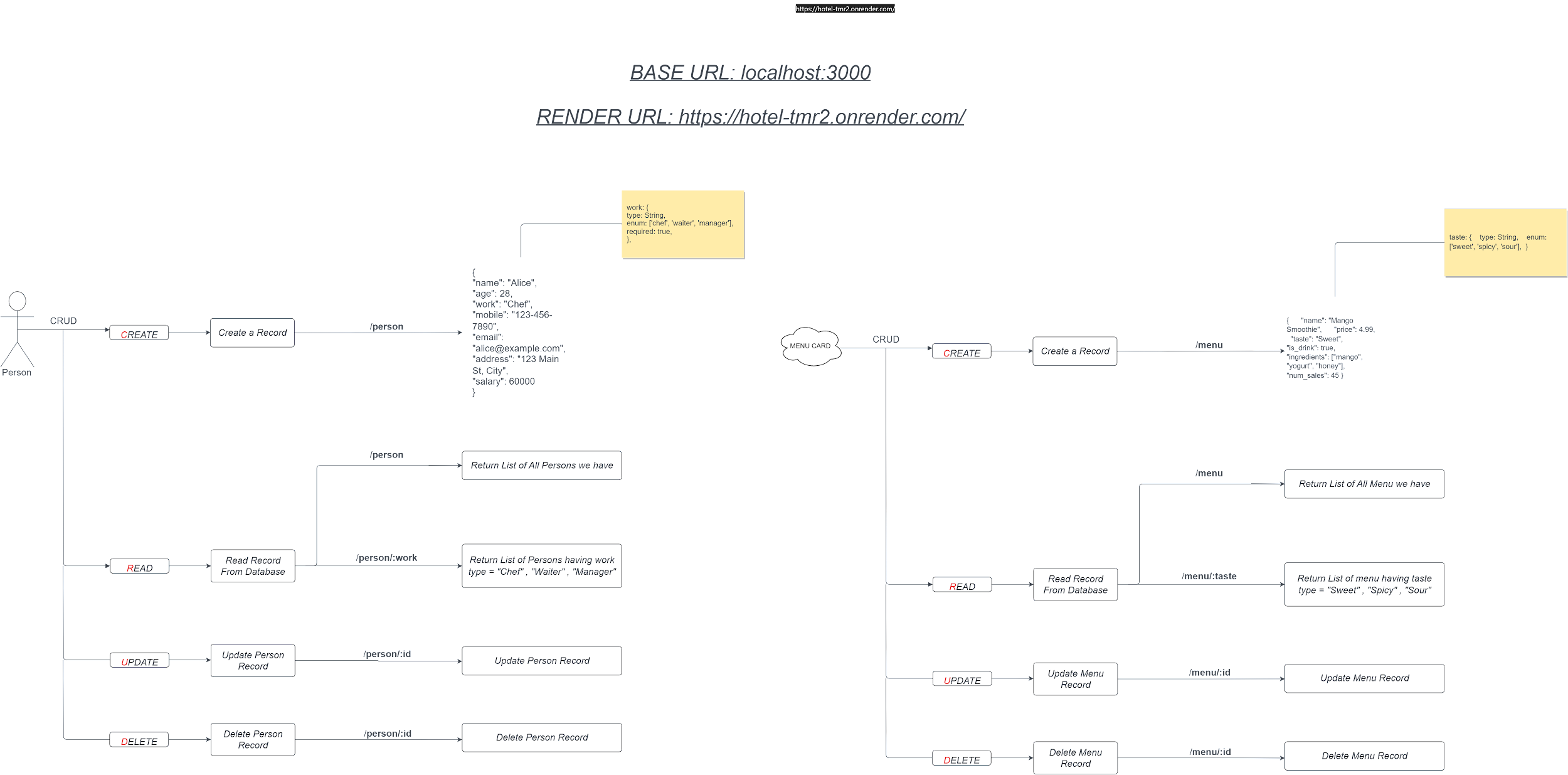
**DAY 6**

### *Homework Update for Menu API*

* **Task To create POST /menu and GET /menu**
* We are now creating a POST method to save menu details and it’s similar to person details and the same for the GET method

### *Flow Diagram of API*

[](https://app.diagrams.net/?page-id=nwkyMnnYAZ6rW4phBCeg&scale=auto#G1TswAyCgfsa04Hp6f4OP-Umg_GVkdW4eQ)

<https://drive.google.com/file/d/1TswAyCgfsa04Hp6f4OP-Umg_GVkdW4eQ/view?usp=sharing>

### *Parametrised API calls*

* Now if someone told you to give a list of people who are only waiters
* Then we can create an endpoint like this
* **/person/chef**
* **/person/waiter**
* **/person/manager**
* But this is not the correct method to create as many functions Here we can use parametrized endpoints
* It can be dynamically inserted into the URL when making a request to the API.
* **localhost:3000/person/:work**

→ work = [ “chef”, “waiter”, “manager” ]

| app.get('/person/:work', async (req, res) => {  try {  const workType = req.params.work; // Extract the work type from the URL parameter   // Assuming you already have a Person model and MongoDB connection set up  const persons = await Person.find({ work: workType });   // Send the list of persons with the specified work type as a JSON response  res.json(persons);  } catch (error) {  console.error('Error fetching persons:', error);  res.status(500).json({ error: 'Internal server error' });  } }); |
| --- |

### *Express Router*

* We have lots of Endpoints in a single file server.js
* This makes bad experience in code readability as well as code handling
* Express Router is a way to modularize and organize your route handling code in an Express.js application.
* So let’s create a separate file to manage endpoints /person and /menu
* Express Router is like a traffic cop for your web server
* Express Router helps you organize and manage these pages or endpoints in your web application. It's like creating separate folders for different types of tasks.
* Create a folder routes → personRoutes.js

| const express = require('express'); const router = express.Router();  // Define routes for /person router.get('/', (req, res) => {  // Handle GET /person });  router.post('/', (req, res) => {  // Handle POST /person });  module.exports = router; |
| --- |

* Now in **server.js,** we will use this personRoutes

| // Import the router files const personRoutes = require('./routes/personRoutes');  // Use the routers app.use('/person', personRoutes); |
| --- |

### *Update Operation*

* We will update our person Records, and for that, we will create an endpoint from where we are able to update the record
* For Updation, we need two things
  + Which record we want to update?
  + What exactly do we want to update?
* For update, we will use the ***PUT*** method to create an endpoint
* *What is a unique identifier in a document in a collection?*
* It’s **\_id** which Mongodb itself gives, We will use this to find the particular record that we want to update
* —> And now we will send the data the same as we did in the POST method.

| app.put('/person/:id', async (req, res) => {  try {  const personId = req.params.id; // Extract the person's ID from the URL parameter  const updatedPersonData = req.body; // Updated data for the person   // Assuming you have a Person model  const updatedPerson = await Person.findByIdAndUpdate(personId, updatedPersonData, {  new: true, // Return the updated document  runValidators: true, // Run Mongoose validation  });   if (!updatedPerson) {  return res.status(404).json({ error: 'Person not found' });  }   // Send the updated person data as a JSON response  res.json(updatedPerson);  } catch (error) {  console.error('Error updating person:', error);  res.status(500).json({ error: 'Internal server error' });  } }); |
| --- |

### *Delete Operation*

* We will **Delete** our person Records, and for that we will create an endpoint from where we are able to delete the record
* For Deletion, we need one thing
  + Which record we want to update?
* For deletion, we will use the ***DELETE*** method to create an endpoint
* *What is a unique identifier in a document in a collection?*
* It’s **\_id** which Mongodb itself gives, We will use this to find the particular record that we want to delete

| app.delete('/person/:id', async (req, res) => {  try {  const personId = req.params.id; // Extract the person's ID from the URL parameter   // Assuming you have a Person model  const deletedPerson = await Person.findByIdAndRemove(personId);   if (!deletedPerson) {  return res.status(404).json({ error: 'Person not found' });  }   // Send a success message as a JSON response  res.json({ message: 'Person deleted successfully' });  } catch (error) {  console.error('Error deleting person:', error);  res.status(500).json({ error: 'Internal server error' });  } }); |
| --- |