# Day 6

* Home-work 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 diagaram of API***

**<https://drive.google.com/file/d/1TswAyCgfsa04Hp6f4OP-Umg_GVkdW4eQ/view>**

**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

1. /person/chef
2. /person/waiter
3. /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”];

**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***

***\* Update Operation***

\* We will update our person Record, for that we will create an endpoint from where we are able to update record

\* For Updation we need two things

\* Which record we want to update?

\* What exactly we want to update?

\* For update we will use PUT method to create a endpoint

\* **What is a unique identifier in a document in a collection?**

\* It's \_id which is given by mongodb itself, we will use this to find the particular record which we want to update

\* --> And now we will send the data as same like we did in POST method.

*//Update*

router**.**put('/:id'**,** *async* (**req,** **res**) **=>** {

**try** {

    const personId **=** req**.***params***.***id***;**

    const updatedData **=** req**.***body***;**

    const response **=** **await** Person**.**findByIdAndUpdate(personId**,** updatedData**,** {

      new**:** true**,**  *//Return the updated document*

      runValidators**:** true  *//Run Mongoose validation*

    })**;**

**if** (**!**response) {

**return** res**.**status(404)**.**json({ error**:** 'person not found!! ' })

    }

    console**.**log("Data Updated!! ")**;**

    res**.**status(200)**.**json(response)**;**

  } **catch** (error) {

    console**.**log("Error!!")**;**

    res**.**status(500)**.**json({ error**:** "Internal server Error!! " })

  }

})

**\* Delete Operation**

**\*** We will Delete our person Record, for that we will create an endpoint from where we are able to delete record

\* For Deletion we need one things

\* *Which record we want to update?*

\* For delete we will use **DELETE** method to create a endpoint

\* What is a unique identifier in a document in a collection?

\* It's \_id which is given by mongodb itself, we will use this to find the particular record which we want to delete:

*// DELETE Function:*

router**.**delete("/:id"**,** *async* (**req,** **res**) **=>** {

**try** {

    const personId **=** req**.***params***.***id***;**

    const response **=** **await** Person**.**findByIdAndDelete(personId)**;**

*//Assuming we have a person model:*

**if** (**!**response) {

**return** res**.**status(404)**.**json({ error**:** 'person not found!! ' })

    }

    console**.**log("Data Delted!! ")**;**

    res**.**status(200)**.**json({ message**:** 'person Deleted Successfully' })**;**

  } **catch** (error) {

    console**.**log("Error!!")**;**

    res**.**status(500)**.**json({ error**:** "Internal server Error!! " }) } })

***Homeworks***

**Task:**

Must be completed before starting today’s namaste JS series.

Create parameterized routes for menu based on tasks.

***Do the same things for menu as done for person.***