Node API documentation

Code setup for project

* Install an IDE (I had used VScode).
* Install Node js and check for installation using node –v command.
* Install express js framework to develop back-end Api application on node js.
* Install postman for request validation.
* Install MongoDB for storing data into a database.
* Install mongoose to connect node application with database.

Start Developing

* Create server.js file.
* Create http methods (get, post, put, delete) and respective response status.
* Use async/await for asynchronous request handling.
* Use try catch blocks to catch errors.
* Connect the server.js file to the database using mongoose.
* Use connect method and provide the MongoDB URI to complete the connection.
* Install nodemon to auto start the server.
* Listen to the server and provide the port number to see the result on the browser.
* Keep checking the collection in the database also.

Creating HTTPS methods

GET ALL

* Create GET method to fetch all the tasks.
* Use find({}) from mongoose to get all the records from database.
* Send response from the server 200 that response fetched correctly.
* If not, then catch error in the catch block and send response status as 500.

POST

* Create POST method to create new tasks in the database.
* Use the create method from mongoose to create a new task.
* Send response 200 if a new task is created.
* If not, then catch error in the catch block and send response status as 500.

GET BY ID

* Create GET method to fetch task by using id as a parameter.
* Use findById method from mongoose to get the task from database.
* Send response 200 if a task is fetched according to its id mentioned.
* If that id is not present, then send response as 404 id not found.
* In case of error use catch to handle the error and send response as 500.

PATCH BY ID

* Create PATCH method to update the existing task with respect to the id.
* Use findByIdAndUpdate method from mongoose to update the task.
* Send response 200 if a task is updated with respect to id mentioned.
* If that id is not present, then send response as 404 id not found.
* I In case of error use catch to handle the error and send response as 500.

DELETE BY ID

* Create DELETE method to delete the existing task with respect to the id.
* Use findByIdAndDelete method from mongoose to delete the task.
* Send response 200 if a task is deleted with respect to id mentioned.
* If that id is not present, then send response as 404 id not found.
* I In case of error use catch to handle the error and send response as 500.

Note: Keep checking database after performing every operation because with every operation there is a simultaneous change in the database as well. If no changes are found, then there are some errors or exceptions occurring in the code itself.

Refactor and use MVC Architecture

Model

* Create a new folder called models and create one file called taskModel.js into it.
* Add all the necessary requirements and parameters of the task model such as task title, task description and task due date.
* Set some required status for validation purpose.
* Import the file in server.js file.

Routes

* Create one new folder called routes and create one file called taskRoutes.js into it.
* Declare all the http methods into it and import taskModel functionality and validations by importing that file.
* Now import taskRoutes.js into server.js to implement all the routes in the mainstream.

Controllers

* Create one new folder called controllers and create one file called taskController.js into it.
* Shift the http method CRUD logics to taskController.js to handle the CRUD logics.
* Now import taskModel.js file to taskController.js so that schema gets called.
* Import taskController.js file into taskRoutes.js file to call the http functions that we have created in the controller file.
* Finally save the MVC changes and run the application.

Note: Keep a good track of all the modules and import them into correct files to follow the MVC architecture. Once it's achieved check whether all the operations are happening or not and simultaneously check the database to see whether all the changes are being reflected into it or not.

Advantages of MVC architecture: -

* Increases code readability.
* Makes application modular.
* Makes applications easy to test.
* Makes Asynchronization seamless.
* Makes code debugging easy.

Error Handling Middleware

* Create a middleware function to handle development and production time errors.
* Import that file in server.js.
* Use the middleware function inside the application.
* Check for clean errors and null stack for an incomplete response in production environment.
* Development and production time runtime response errors are handled successfully.