# **Create a REST API backend service using NodeJs and Express**

## Create a backend service for a post sharing application like instagram, the user should be able to add posts, edit and delete them as required

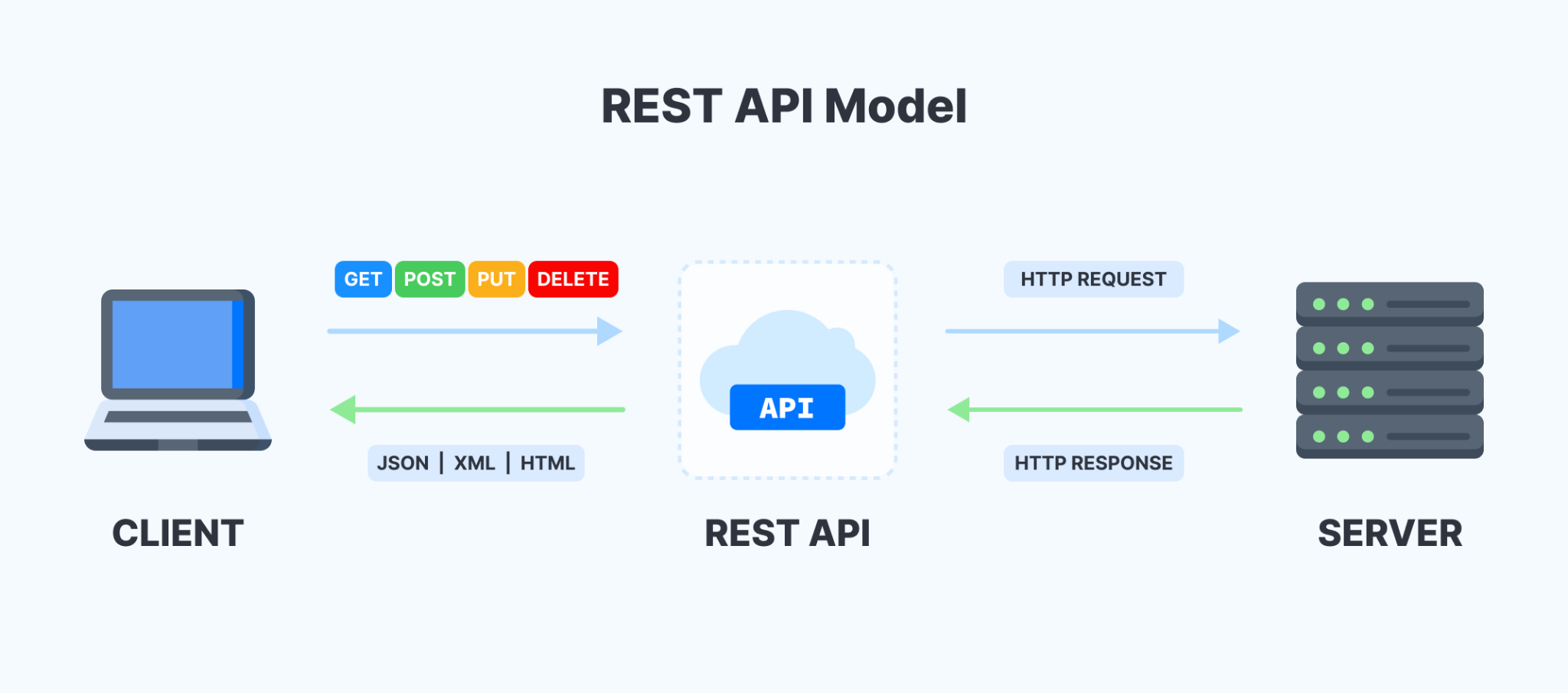
**Background**

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

Representational state transfer (REST) is a software architectural style that was created to guide the design and development of the architecture for the World Wide Web. REST defines a set of constraints for how the architecture of an Internet-scale distributed hypermedia system, such as the Web, should behave. The REST architectural style emphasizes the scalability of interactions between components, uniform interfaces, independent deployment of components, and the creation of a layered architecture to facilitate caching components to reduce user-perceived latency, enforce security, and encapsulate legacy systems.

A REST API is an API implementation that adheres to the REST architectural constraints. It acts as an interface. The communication between the client and the server happens over HTTP. A REST API takes advantage of the HTTP methodologies to establish communication between the client and the server. REST also enables servers to cache the response that improves the application’s performance.

The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are **POST, GET, PUT, PATCH, and DELETE**. These correspond to create, read, update, and delete (or CRUD) operations, respectively.



## **Instructions**

* Create a folder (eg. **Assignment**) for storing the files related to the assignment,
* Create a database connection to mongodb on your localhost with the **database name as assignment**
* Create an express server that listens for request on Port 3000
* Create user and post schema with { name, email, password } for user and { title, body, image, user } for post

Create login and registration API for users to log into the app and register a new user, store the data in the mongodb

| **Register New User**   * **POST - /register** - Accepts name, email and password in the body, the email Id should be unique i.e no two users can have the same email id * **POST - /login** - Accepts email and password in request body, verify the email and password and respond with correct http status code in case of success or failure     Return token in the response after successful login (Example Response - { token: <JWT Token> } ) |
| --- |

* Add CRUD routes for Posts create, read, update, and delete operations

| * **GET -**  **/posts** - Get all the posts in the db, the response should be in format (Example - { posts: [] } ) * **POST - /posts** - Create a new post (accept title, body, image) and store the info in the db with the user reference, return the post created in the response (Example - { \_id, name, title, body, user } ) * **PUT - /posts/:postId** - Edit/Update a Post with the id provided * **DELETE - /posts/:postId** - Delete a Post with the id provided |
| --- |

* Add a middleware to perform authentication and authorization for users (only logged in users can access the posts apis - use JWT for authentication)
* Only authorized users can perform Edit/Delete Operation for the posts, userA should not be able to edit/delete userB's posts
* Use appropriate status code for different operations

Example - Use 200 http status for successful request, use 4xx status code for unauthorized, forbidden response

**Evaluation**

| Test case number | Steps | Example |
| --- | --- | --- |
| 1 | 1. Open command line 2. Do cd to the directory where your folder is present 3. Run the command **npm run dev** 4. It should print **“server is running”** |  |
| 2 | 1. Download Postman App 2. Create a new collection **users** in it 3. Add a new **POST** request registerUser in it 4. Enter URL ([**http://localhost:3000/register**](http://localhost:3000/register)) 5. Inside body provide **name, email and password** for registering the user   Eg. {“name”: “Raja Rai”, “email”: “[rajarai@gmail.com](mailto:rajarai@gmail.com)”, “password”: “123456789”}   1. Click on Send button to create a request 2. It should show “status”: “success” along with the data entered in the body (refer the image) |  |
| 3 | 1. Add a new **POST** request loginUser in the collection, 2. Enter URL ([**http://localhost:3000/login**](http://localhost:3000/login)), 3. Inside body provide email and password to log in the registered user   Eg.{ “email”: “[rajarai@gmail.com](mailto:rajarai@gmail.com)”, “password”: “123456789”},   1. Click on Send button to log in, 2. It should show “status”: “success” and generate a token (refer the image). |  |
| 4 | 1. Add a new **POST** request createPosts in the collection, 2. Enter URL ([**http://localhost:3000/posts**](http://localhost:3000/posts)), 3. Use the generated token inside **Headers** for Authorization, 4. Inside body provide title, body and image to create a post   Eg. {“title” : “This is Raja Rai”, “body” : “Do whatever you like”, “image”: “<https://cdn.pixabay.com/photo/2015/04/23/22/00/tree-736885__480.jpg>”} etc.,   1. Click on Send button to create a post, 2. It should show “status”: “post created” along with the data entered in the body (refer the images). |  |
| 5 | 1. Add a new **PUT**  request loginUsers in the collection, 2. Enter URL ([http://localhost:3000/posts/:postId](http://localhost:3000//posts/:postId)), 3. Use the generated token inside **Headers** for Authorization, 4. Inside body edit whatever data you want title, body or image to edit post   Eg.{ “title”: “Best view”},{“body”: “Let’s do it”} etc.   1. Click on Send button to edit the post, 2. It should show “status”: “success” and check in the GET request as fetchPosts in the collection (refer the image). |  |
| 6 | 1. Add a new **DELETE**  request deleteUsers in the collection, 2. Enter URL ([http://localhost:3000/posts/:postId](http://localhost:3000//posts/:postId)), 3. Use the generated token inside **Headers** for Authorization, 4. Click on Send button to delete the post, 5. It should show “status”: “Successfully deleted” and check in the GET request as fetchPosts in the collection whether the post with the used id is there or not (refer the image). |  |

Submission

1. Create a repository in GitHub for assignment-2.
2. Create a new branch from main/master branch (say my-assignment-2) and write code in new branch. After completing the assignment, push this (my-assignment-2 branch) into github and raise PR and submit the github link.