



Efficient RESTful API Development with NodeJS, Express, and PostgreSQL

Bhargav Bachina

ABSTRACT

This paper explores the construction of REST APIs using NodeJS and PostgreSQL, guided by the principles of REpresentational State Transfer (REST). REST is a popular architectural style for designing networked applications, due to its stateless communication and standard operations, which enhance system interoperability. The core of our investigation focuses on leveraging the Express Framework to initiate and structure a RESTful API, subsequently integrating PostgreSQL to manage database operations. We commence by outlining the necessary prerequisites for building a REST API and progress through the creation of an example project to demonstrate practical implementation. The discussion extends to setting up the project structure, installing PostgreSQL on a local machine, and utilizing PGAdmin for database management. Critical to the development process is the establishment of a database schema and the integration of PostgreSQL within the Express application to facilitate data interactions. Further, we delve into the specifics of implementing CRUD (Create, Read, Update, Delete) operations, essential for any RESTful service. The paper also addresses the importance of logging for debugging and monitoring purposes, and the integration of Swagger for API documentation and endpoint testing. By providing a step-by-step guide, from environment setup to application configuration, this document aims to equip developers with the knowledge to build efficient, scalable, and maintainable REST APIs using NodeJS and PostgreSQL, contributing to the broader field of web application development.

Key words: Programming, Web Development, Nodejs, Software Development, Software Engineering

INTRODUCTION



REST, short for Representational State Transfer, is a set of principles and guidelines that aid in the design and development of scalable and reliable systems for intercommunication over the web. This architectural style streamlines the interaction between client and server through stateless operations, ensuring a standardized approach for transmitting data across various systems. In this detailed exploration, we aim to demonstrate the process of constructing a RESTful API utilizing NodeJS and PostgreSQL for backend data management. The journey begins with an introduction to the Express Framework, a powerful and flexible Node.js web application

framework that provides a robust set of features to develop web and mobile applications. We will guide you through the initial setup process, illustrating how to structure a project effectively to support RESTful architecture

Following the setup, the discussion transitions to the integration of PostgreSQL, a sophisticated open-source relational database system known for its reliability, feature robustness, and performance. We will walk through the steps necessary to initiate a PostgreSQL database, including installation, configuration, and the creation of database tables essential for storing and managing data. The final segment of our exploration involves bridging the NodeJS application with the PostgreSQL database. This includes configuring the NodeJS environment to communicate seamlessly with the database, thereby enabling the retrieval, manipulation, and storage of data through the constructed REST API. By following this comprehensive guide, developers will gain a deeper understanding of REST principles and acquire the skills needed to develop a fully functional REST API with NodeJS and PostgreSQL, enhancing their capabilities in web services development and database integration.

- Prerequisites
- Example Project
- Project Structure
- Install PostgreSQL on Local Machine
- Install PGAdmin Tool
- Create a Database Table
- Configure PostgreSQL in Express App
- CRUD Operations
- Logging
- Swagger
- Summary
- Conclusion

PREREQUISITES

There are some prerequisites for this post. You need to have a NodeJS installed on your machine and some other tools that are required to complete this project.

- NodeJS (<https://nodejs.org/en/>)
- Express Framework (<https://expressjs.com/>)
- PGAdmin (<https://www.pgadmin.org/>)
- PostgreSQL (<https://www.postgresql.org/>)
- Postgresapp (<https://postgresapp.com/>)
- Node-Postgres (<https://www.npmjs.com/package/pg>)
- VSCode (<https://code.visualstudio.com/>)
- Swagger (<https://swagger.io/>)
- Postman (<https://www.postman.com/>)
- Nodemon (<https://nodemon.io/>)
- Dotenv (<https://www.npmjs.com/package/dotenv>)

NodeJS: As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications.

Express Framework: Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

PGAdmin: pgAdmin is an Open-Source administration and development platform for PostgreSQL.

PostgreSQL: Open-Source relational Database

Node-Postgres: Non-blocking Postgresql Client for NodeJS

VSCode: The editor we are using for the project. It's open-source and you can download it here.

Swagger: API documentation

Postman: Manual testing your APIs

Nodemon: To speed up the development.

If you are new to NodeJS and don't know how to build REST API with it, I would recommend going through the below article. We used the project from this article as a basis for this post.

How to write production-ready Node.js Rest API — Javascript version (<https://medium.com/bb-tutorials-and-thoughts/how-to-write-production-ready-node-js-rest-api-javascript-version-db64d3941106>)

EXAMPLE PROJECT

Here is the Github link for the example project you can just clone and run it on your machine.

// clone the project git clone <https://github.com/bbachi/nodejs-restapi-postgresql.git>

Here are the main and controller classes. The server.js file is the main file and starting file of the API and the controller is called from here.

<https://gist.github.com/bbachi/a90d4016c0bea72105b135f775eaa6b5#file-server-js>

Here is the controller file which has four methods.

<https://gist.github.com/bbachi/ee9f4fb39572ed2ecf02a579aba3bc33#file-task-controller-js>

We are using nodemon for the development phase that speeds up your development. You just need to run this command after installing all dependencies.

// install dependencies npm install

// start the server in development phase npm run dev

https://miro.medium.com/v2/resize:fit:720/1*4f-1Pggz90HI-b49jRa0IQ.gif

PROJECT STRUCTURE

Let's understand the project structure that we have here. The starting point of the application is *server.js* and we have all the scripts, dependencies, etc in the *package.json*.



Figure 1: Project Structure

We have a controller, service, and repository in place. All the logging-related configuration goes into *api.logger.js* under the folder logger. We have the *.env* file for all the environment-related configurations. We have *swagger.json* and *swagger.css* for the swagger docs. We have a *db.config.js* file for all the database configurations.

INSTALL POSTGRESQL ON LOCAL MACHINE

There are so many ways to install PostgreSQL on your local machine from the below link. The Postgres.app is the easiest and fastest one.

<https://www.postgresql.org/download/macosx/>

You can click on the Postgres.app and download the app from that page.

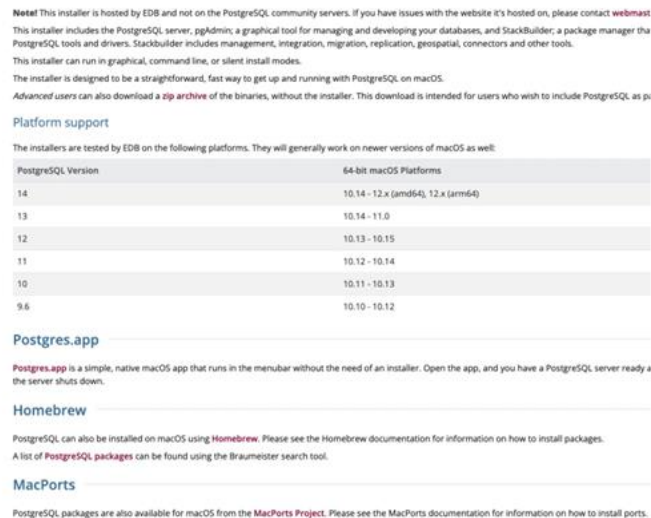


Figure 2: Postgres.app

You can go through the below installation steps and initialize the Database.



Figure 3: Download and Install Instructions

If everything is successful, you can see the below screen with the database named after the user name on the machine.

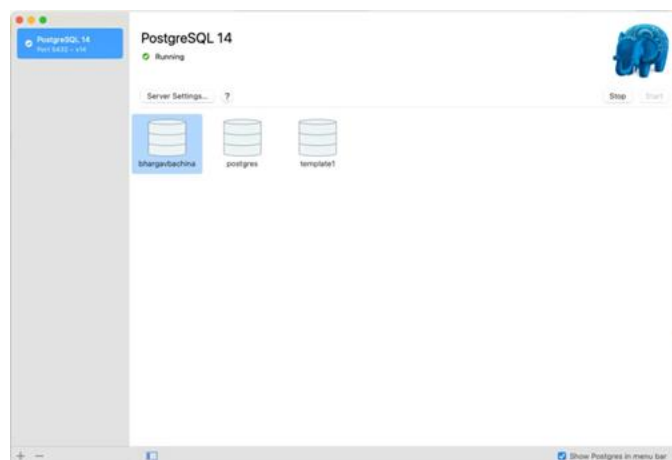


Figure 4: Postgres.app

INSTALL PGADMIN TOOL

The pgAdmin tool is the open-source administration and development platform for PostgreSQL. You can install this tool from the following location.

<https://www.pgadmin.org/>



Figure 5: pgAdmin Tool

Once installed, you can open that and connect to the PostgreSQL server with the following credentials. It changes based on your user name folder.

// name of the servername: local (You can name anything)

// Hostnamehost name: localhost

// User Nameusername: <user name based on the above postgres.app>

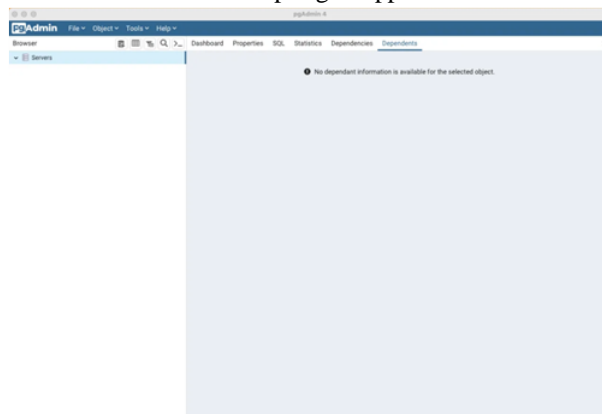


Figure 6: pgAdmin Tool

Let's connect to the server by clicking on the register as below.

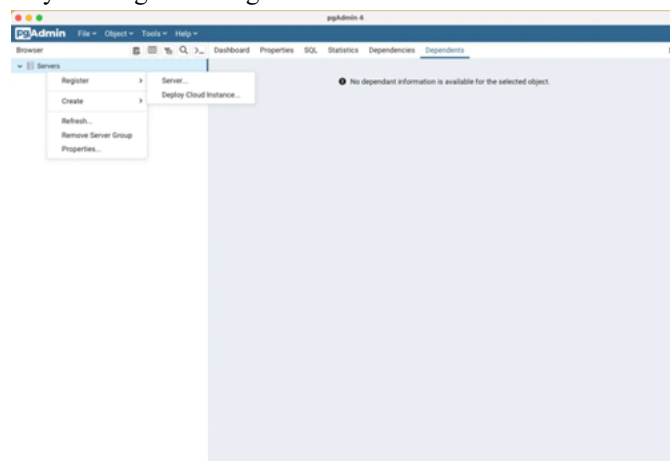


Figure 7: Register Server

The server name can be anything that you give for your server such as local, dev, test, etc.

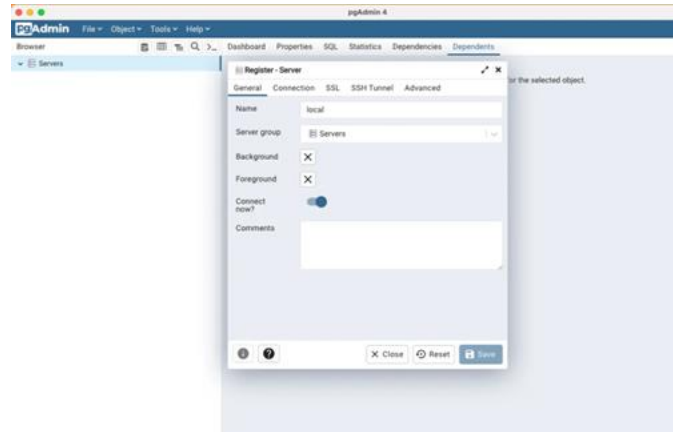


Figure 8: Server Name

Let's give all the details such as HostName, port, username, etc under the connection tab.

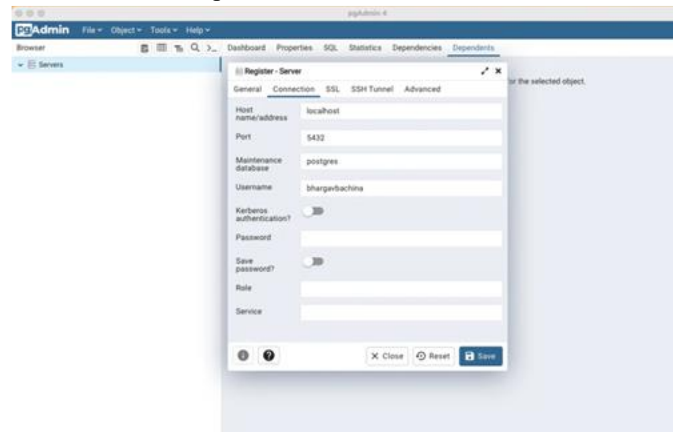


Figure 9: Connection Details

Once connected, you can see the details below.

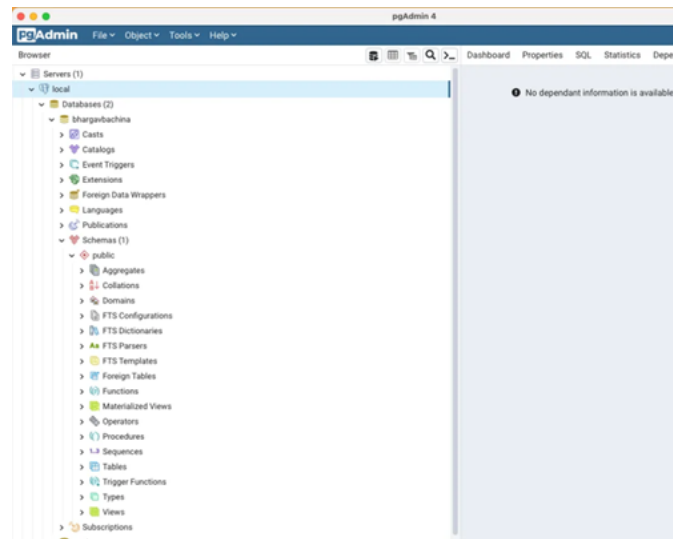


Figure 10: Connected

CREATE A DATABASE TABLE

Let's create a table by clicking on the Query Tool as below.

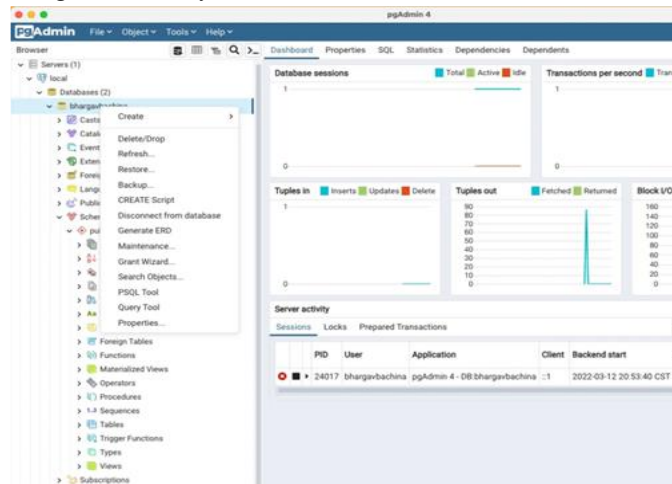


Figure 11: Query Tool

Let's run the following query to create the database table.

<https://gist.github.com/bbachi/9dbcc0530954f4a8320eda5ecd34b5c3#file-database-sql>

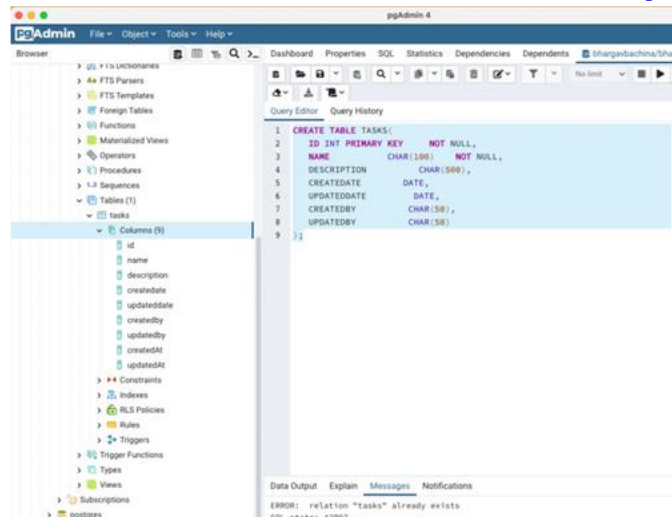


Figure 12: Table Created

CONFIGURE POSTGRESQL IN EXPRESS APP

Let's configure the pg Client from our application. The first thing we need to do is to get the connection string or connection details. You can get it from the properties as below.

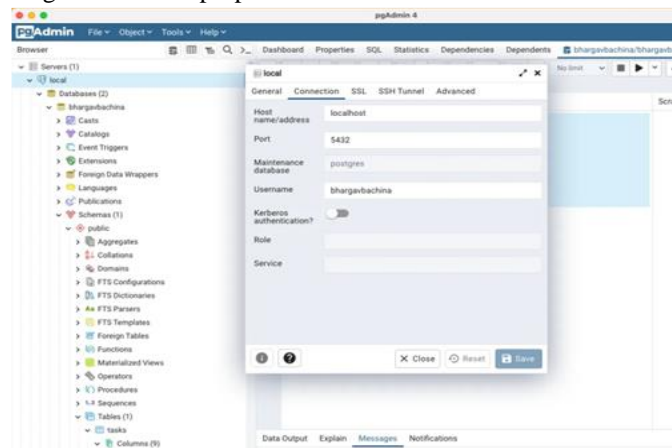


Figure 13: Connection Details

The next thing is to install the pg client with the following command.

```
// install client and sequelize
npm install pg npm install pgnpm install sequelize
```

Let's place the connection string and database name in the application properties file as below. You have to URL encode the password if you have any special characters in the password.

We need to use the dotenv library for environment-specific things. Dotenv is a zero-dependency module that loads environment variables from a `.env` file into `process.env` (https://nodejs.org/docs/latest/api/process.html#process_process_env). Storing configuration in the environment separate from code is based on **The Twelve-Factor App** (<http://12factor.net/config>) methodology.

The first step is to install this library `npm install dotenv` and put the `.env` file at the root location of the project <https://gist.github.com/bbachi/84a2d8470359a5d618a7e2b6a60d7b75#file-env>

We just need to put this line `require('dotenv').config()` as early as possible in the application code as in the `server.js` file.

<https://gist.github.com/bbachi/1a29181bbce1029e85af3802504a778f#file-server-js>

Let's define the configuration class where it creates a connection with the connection details from the properties. We are using pg client to connect with PostgreSQL for all the queries. This client makes it easy for you to interact with PostgreSQL. We are fetching the connection details with the dotenv library and connecting it to PostgreSQL with pg client. We are exposing one function from this file connect.

Sequelize is a promise based NodeJS ORM tool for many relational databases such as Postgres, MYSQL, etc.

<https://gist.github.com/bbachi/d2a5ec615c9a8477b5b3369e07bddfb3#file-db-config-js>

We need to define a model for our collection as below. We need to define the schema for the collection and then you need to pass that schema to the model and export it as a module.

<https://gist.github.com/bbachi/58dce9e582edaa51699338c249087363#file-task-model-js>

Finally, here is the repository class where it uses the above model for all the CRUD operations.

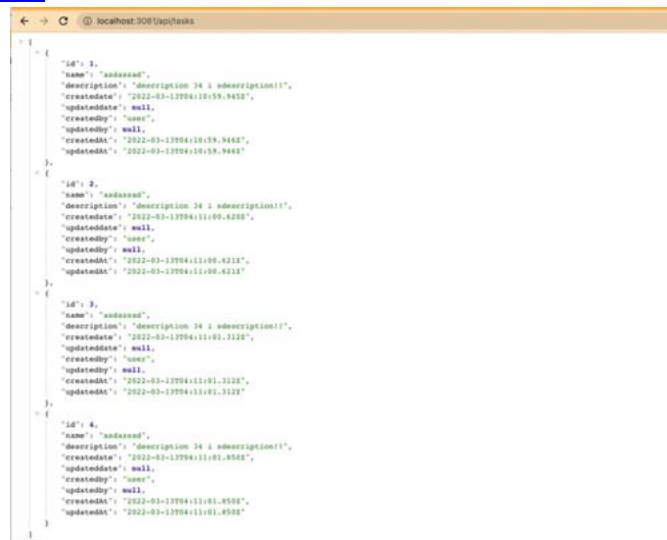
<https://gist.github.com/bbachi/a5d0c2cdb190971fb0bbfde835633264#file-task-repository-js>

We have this service file in between the controller and repository for any data manipulation if needed.

<https://gist.github.com/bbachi/64f3228af4e9084194f218c8c6303986#file-task-service-js>

With all the above files in place, we can hit the following URL.

<http://localhost:3081/api/tasks>



```

[
  {
    "id": 1,
    "name": "task1",
    "description": "description 1",
    "createdate": "2022-03-13T04:10:59.945Z",
    "updatedate": null,
    "createdby": "user",
    "updatedby": null,
    "createdat": "2022-03-13T04:10:59.945Z",
    "updatedat": "2022-03-13T04:10:59.945Z"
  },
  {
    "id": 2,
    "name": "task2",
    "description": "description 2",
    "createdate": "2022-03-13T04:11:09.628Z",
    "updatedate": null,
    "createdby": "user",
    "updatedby": null,
    "createdat": "2022-03-13T04:11:09.628Z",
    "updatedat": "2022-03-13T04:11:09.628Z"
  },
  {
    "id": 3,
    "name": "task3",
    "description": "description 3",
    "createdate": "2022-03-13T04:11:01.312Z",
    "updatedate": null,
    "createdby": "user",
    "updatedby": null,
    "createdat": "2022-03-13T04:11:01.312Z",
    "updatedat": "2022-03-13T04:11:01.312Z"
  },
  {
    "id": 4,
    "name": "task4",
    "description": "description 4",
    "createdate": "2022-03-13T04:11:01.858Z",
    "updatedate": null,
    "createdby": "user",
    "updatedby": null,
    "createdat": "2022-03-13T04:11:01.858Z",
    "updatedat": "2022-03-13T04:11:01.858Z"
  }
]

```

Figure 14: API GET URL

CRUD OPERATIONS

Let's do some CRUD Operations.

A. Create Task

Create Task is the post-call that takes the request body and saves that into the todos collection.

URL: <http://localhost:3081/api/task>

Request Body: { "task" { "name": "asdassad", "description": "description 34 i sdescription!!", "createdby": "user" } }

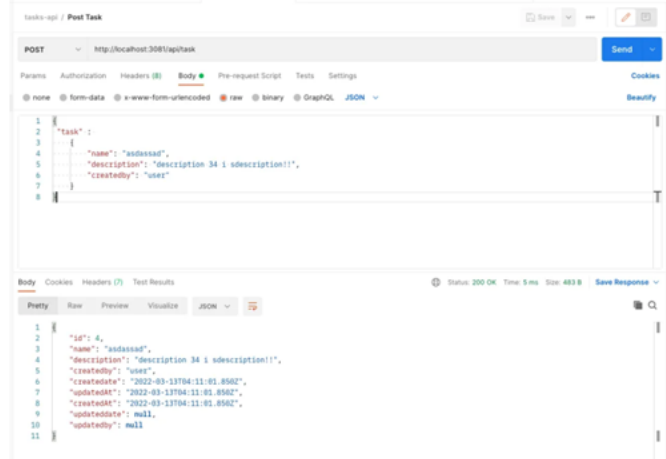


Figure 15: Post Call

B. Update Task

Update Task is the put call that takes the request body and updates that into the todos collection with the same id.

URL: <http://localhost:3081/api/task>

Request Body: { "task": { "id": "1", "name": "name 1", "description": "description 1 ", "updatedby": "user" } }

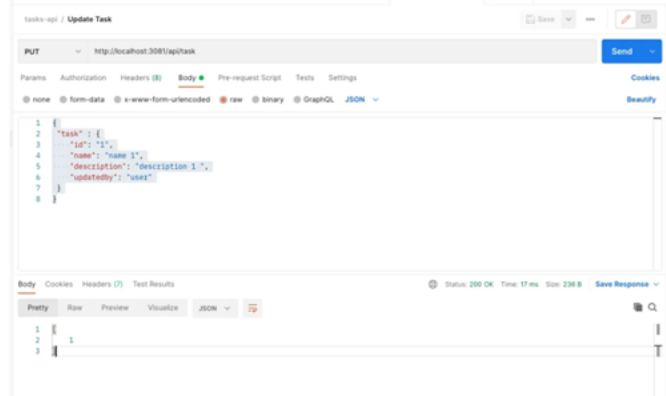


Figure 16: PUT Call

C. Delete Task

Delete Task is the delete call that takes the id as a path variable and deletes it from the todos collection.

URL: <http://localhost:3081/api/task/1>

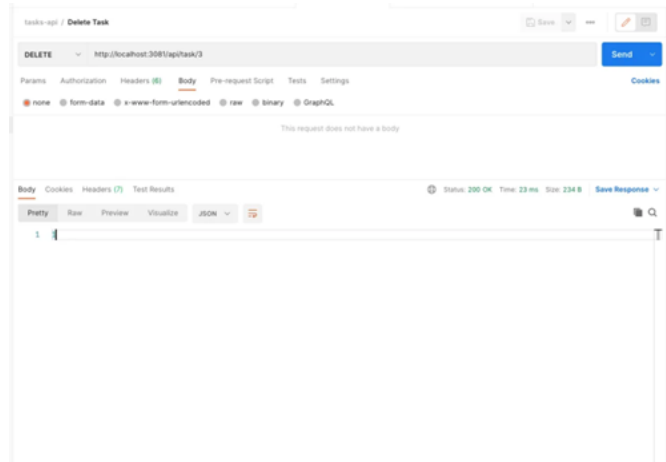


Figure 17: Delete Call

D. Get Tasks

Get Tasks is a simple GET URL that retrieves all the tasks from the todos collection.

URL: <http://localhost:3081/api/tasks>

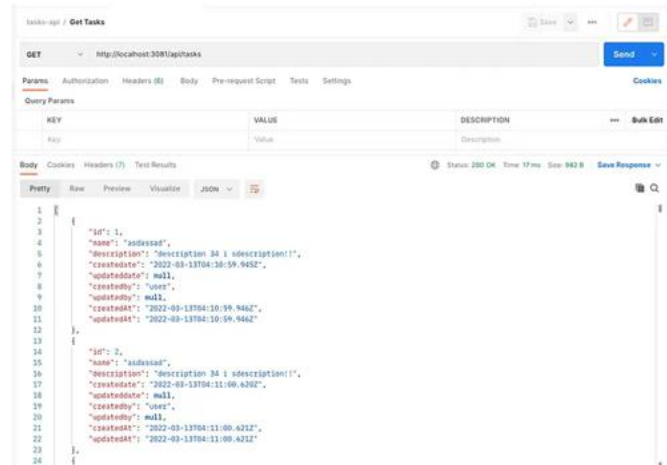


Figure 18: GET Call

LOGGING

There are so many logger libraries, and I used the pine library for the logging. It's always best practice to have a separate file for logging and importing that file everywhere. You can configure and change the underlying logging library with one file change.

I have defined the following file.

<https://gist.github.com/bbachi/1fbef2983c21950ae736a68d64c41009#file-api-logger-js>

You can import this in any file like below and use these methods.

<https://gist.github.com/bbachi/30cce808703910392e307fcd2f93015#file-task-controller-js>

```
Bhargavs-MacBook-Pro:python bhargavbachina$ helm install api-release1 python-api-chart-0.1.0.tgz
NAME: api-release1
LAST DEPLOYED: Tue Nov 2 23:18:33 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Get the application URL by running these commands:
kubect1 get svc
kubect1 cluster-info
```

Figure 19: Installed

You can see the notes on how to access the application because we have Notes.txt in the Chart folder as below. You can put any information that helps the developers who use your chart.

<https://gist.github.com/bbachi/f3273a5a13226853b6f675ebef0feec4#file-notes-txt>

SWAGGER

The Swagger is a tool that simplifies API documentation for the services. With Swagger, you can design your API and share it across teams very easily.

The first thing we need to do is to install swagger-related dependencies with the following command.

```
npm install swagger-ui-express swagger-jsdoc --save
```

There are two things you need to understand before implementing swagger docs to your REST API.

A. Swagger.json

The first thing is to add the file `swagger.json` to define the operations. You can define schemas for the request and response objects, you can define parameters and body and descriptions of each Http operation, etc.

<https://gist.github.com/bbachi/2f053a32dc096b8c999a23d50f905abf#file-swagger-json>

B. Custom CSS

You can have your own custom CSS for your swagger docs Rest API. You need to define the `swagger.css` file where you can put all your CSS that can be applied to the Swagger page.

<https://gist.github.com/bbachi/53187853801960d536a48aaa52287482#file-swagger-css>

Once you have these files `swagger.json` and `swagger.css` in place, it's time to add the swagger-related code in the `server.js`.

```
// import library and files
const swaggerUi = require('swagger-ui-express');const swaggerDocument = require('./swagger.json');const customCss = fs.readFileSync((process.cwd()+"/swagger.css"), 'utf8');
// let express to use thisapp.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument, {customCss}));
```

Here is the complete `server.js` file

<https://gist.github.com/bbachi/0be92f6b1dc3543f07adfc5663a1d77#file-server-js>

You can start the project in the development environment with the following command and access the

```
// start the projectnpm run dev
```

```
// Access the swagger docs herehttp://localhost:3081/api-docs/#/
```

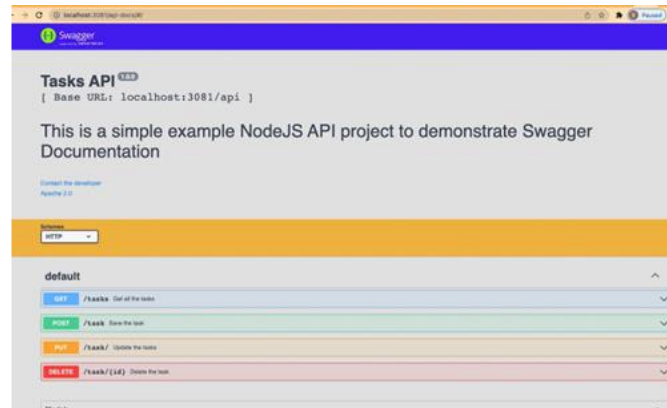


Figure 20: Swagger UI

SUMMARY

- REST is an acronym for REpresentational State Transfer. Rest follows some guidelines to facilitate communication between systems.
- You can get the connection string and configure the NodeJS application to talk to PostgreSQL with pg client, etc.
- The Swagger is a tool that simplifies API documentation for the services. With Swagger, you can design your API and share it across teams very easily.
- Node-Postgres: Non-blocking Postgresql Client for NodeJS
- PGAdmin: pgAdmin is an Open-Source administration and development platform for PostgreSQL.
- The Swagger is a tool that simplifies API documentation for the services. With Swagger, you can design your API and share it across teams very easily.

CONCLUSION

In conclusion, this paper has provided a comprehensive exploration of developing RESTful APIs, adhering to the principles of Representational State Transfer (REST), which simplifies and standardizes communication between systems. We detailed the process of integrating a NodeJS application with a PostgreSQL database using the pg client, facilitating seamless data interaction and management within a RESTful architecture. Furthermore, we highlighted the significance of API documentation and design, particularly through the use of Swagger, a tool that not only simplifies the creation of API documentation but also enhances collaboration across development teams by providing a clear and interactive interface for API endpoints.

Additionally, the utilization of Node-Postgres, a non-blocking PostgreSQL client for NodeJS, ensures efficient database interactions, while PGAdmin serves as a robust tool for database administration and development, enhancing the overall development workflow. By combining these technologies and tools, developers can create robust, efficient, and scalable web services, thus advancing the field of web development and promoting best practices in API design and database integration. This paper aims to serve as a guide for developers looking to leverage REST principles and the PERN stack for efficient full-stack application development

REFERENCES

- [1]. JavaScript Documentation <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- [2]. PostgreSQL Documentation <https://www.postgresql.org/docs/>
- [3]. Express.js Documentation <https://expressjs.com/>
- [4]. NodeJS Documentation <https://nodejs.org/docs/latest/api/>