

Backend Overview: Node.js MongoDB Rest CRUD API

First, we start with an Express web server. Next, we add configuration for MongoDB database. Then we create models with Mongoose. After that, we write the controller and as the last step we define routes for handling all CRUD operations.

Overview of the Rest APIs that will be exported:

Methods	URLs	Actions
GET	api/tasks	get all Tasks
GET	api/tasks/:id	get Task by id
POST	api/tasks	add new Task
PUT	api/tasks/:id	update Task by id
DELETE	api/tasks/:id	remove Task by id
DELETE	api/tasks	remove all Tasks

Create Node.js App

First, we create a folder:

```
$ mkdir todo-backend
```

```
$ cd todo-backend
```

Next, we initialize the Node.js App with a package.json file:

```
$ npm init
```

Now we install necessary packages: express, mongoose & cors

```
$ npm install express mongoose
```

Note: We want to allow only requests from <http://localhost:3030> to access the endpoint at <http://localhost:3031>

Note: It is necessary to have MongoDB installed on our machine. Here is the official guide for Ubuntu:
<https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/>

Setup Express Web Server

In the root folder, we create a new *server.js* file

```
const express = require("express");
const cors = require("cors");
const app = express();

const corsOptions = {
  origin: "http://localhost:3031",
};

app.use(cors(corsOptions));
// parse requests of content-type - application/json
app.use(express.json());
// parse requests of content-type - application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));
// simple route for testing
app.get("/", (req, res) => {
  res.json({ message: "Welcome to ToDo Application" });
});
// set port, listen for requests
app.listen(3030, () => {
  console.log("Server is running on port 3030")
});
```

So, what we did here is:

- imported express & cors modules. Express is for building Rest APIs and cors provides Express middleware to enable CORS with various options.
- created an Express app, added body-parser and cors middleware using app.use() method. We set origin: <http://localhost:3031>
- defined a simple GET route for testing
- listen on port 3030 for incoming requests

Now, let's run the app with command `$ node server` and open the browser with url <http://localhost:3030>. We should see our json with message.

The first step is done! Next, we're going to work with MongoDB and Mongoose.

Configure MongoDB

First, we create a new *app* folder in our root folder. In this *app* folder we create a *config* folder and inside it a new file called *db.config.js*

```
module.exports = {
  url: "mongodb://127.0.0.1:27017/todo_db",
};
```

Define Mongoose & Connect to Database

First, we create a new *models* folder inside *app* folder. In this *models* folder we create a new file *index.js* with following code:

```
const dbConfig = require("../config/db.config");
const mongoose = require("mongoose");

const db = {};
db.mongoose = mongoose;
db.url = dbConfig.url;
db.tasks = require("./task.js");

module.exports = db;
```

In our folder *models* we're going to create a new file *task.js*, which for now is going to be empty.

Next, we want to call `connect()` method in our `server.js`, so we add the following code:

```
// connect to db
const db = require("./app/models");
db.mongoose
  .connect(db.url)
  .then(() => {
    console.log("Connected to the database");
  })
  .catch((err) => {
    console.log("Cannot connect to the database!", err);
    process.exit();
  });
```

Now, if we run `$ node server` in our console we should get the message: Connected to the database

Define the Mongoose Model

We're going back to our `task.js` file and write the following code:

```
const mongoose = require("mongoose");
const { Schema } = mongoose;

const taskSchema = new Schema({
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: false,
  },
  priority: {
    type: String,
    required: false,
  },
  isComplete: {
    type: Boolean,
    required: false,
  },
  timestamps: true
});

const Task = mongoose.model("task", taskSchema);
module.exports = Task;
```

Now, this Mongoose Model represents **tasks collection** in MongoDB database. These fields will be generated automatically for each **Task document**: `_id`, `title`, `description`, `priority`, `isComplete`, `createdAt`, `updatedAt`, `_v`

After finishing the steps above, we don't need to write CRUD functions because Mongoose Model supports all of them and these functions will be used in our Controller.

Create the Controller

Inside *app* we create a new folder *controllers*. Inside *controllers* we will create a *taskController.js* file with these CRUD functions:

- `task_index`
- `task_details`
- `task_create`
- `task_delete`
- `task_update`
- `task_delete_all`

Our *taskController.js* looks like this:

```
const db = require("../models");
const Task = db.tasks;

// create and save a new Task
const task_create = (req, res) => {};

// retrieve all Tasks from the database
const task_index = (req, res) => {};

// find a single Task
const task_details = (req, res) => {}

// update a single Task
const task_update = (req, res) => {}

// delete a single Task
const task_delete = (req, res) => {}

// delete all Tasks from the database
const task_delete_all = (req, res) => {}

// find all completed Tasks
const task_completed_all_get = (req, res) => {}
```

Now, we need to implement this functions! The next step will be to define the routes.

Define Routes

When a client sends request for an endpoint using HTTP request (GET, POST, PUT, DELETE) we need to determine how the server will response by setting up the routes.

These are our routes:

- `/api/tasks:GET,POST,DELETE`
- `/api/tasks/:id: GET, PUT, DELETE`
- `/api/tasks/completed: GET`

Inside *app* folder we're going to create a new *routes* folder and inside this new folder we'll create a new *taskRoutes.js* file. We'll use Express Router to make our code modular and we're going to scope our routes.

The *taskRoutes.js* will look like this:

```

const express = require("express");
const controller = require("../controllers/taskController");

const router = express.Router();

router.post("/", controller.task_create);
router.get("/", controller.task_index);
router.get("/completed", controller.task_completed_all_get);
router.get("/:id", controller.task_details);
router.put("/:id", controller.task_update);
router.delete("/:id", controller.task_delete);
router.delete("/", controller.task_delete_all);

module.exports = router;

```

Next, we need to include these routes in *server.js* (right before *app.listen()*), using these two lines:

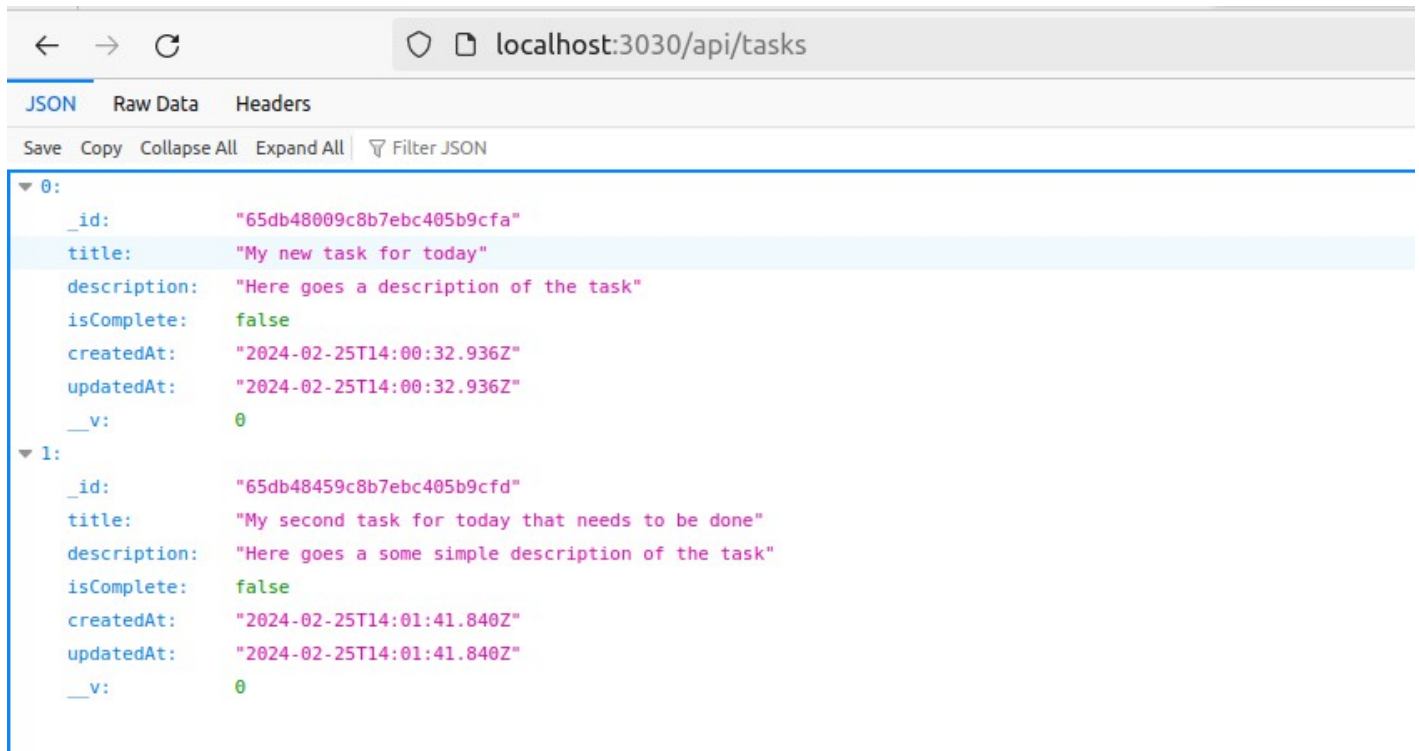
```

...
const taskRoutes = require("../app/routes/taskRoutes");
// task routes
app.use("/api/tasks", taskRoutes)
...

```

Test the APIs

- We're going to use Postman to test our APIs. We added two documents to our tasks collection.



The screenshot shows the Postman interface with a GET request to `localhost:3030/api/tasks`. The response is in JSON format and contains an array of two task documents. The first document has an ID of `"65db48009c8b7ebc405b9cfa"` and a title of `"My new task for today"`. The second document has an ID of `"65db48459c8b7ebc405b9cfd"` and a title of `"My second task for today that needs to be done"`. Both documents have a `description` field, an `isComplete` field set to `false`, and `createdAt` and `updatedAt` timestamps.

```

{
  "_id": "65db48009c8b7ebc405b9cfa",
  "title": "My new task for today",
  "description": "Here goes a description of the task",
  "isComplete": false,
  "createdAt": "2024-02-25T14:00:32.936Z",
  "updatedAt": "2024-02-25T14:00:32.936Z",
  "__v": 0
},
{
  "_id": "65db48459c8b7ebc405b9cfd",
  "title": "My second task for today that needs to be done",
  "description": "Here goes a some simple description of the task",
  "isComplete": false,
  "createdAt": "2024-02-25T14:01:41.840Z",
  "updatedAt": "2024-02-25T14:01:41.840Z",
  "__v": 0
}

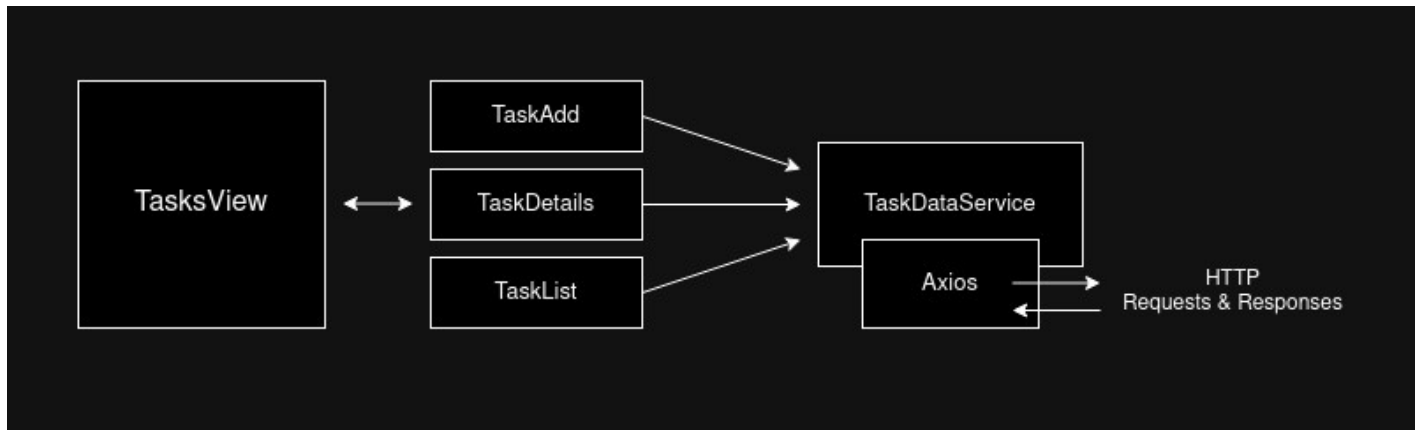
```

Frontend Overview: Vue.js 3 Options API, Vue router & Axios

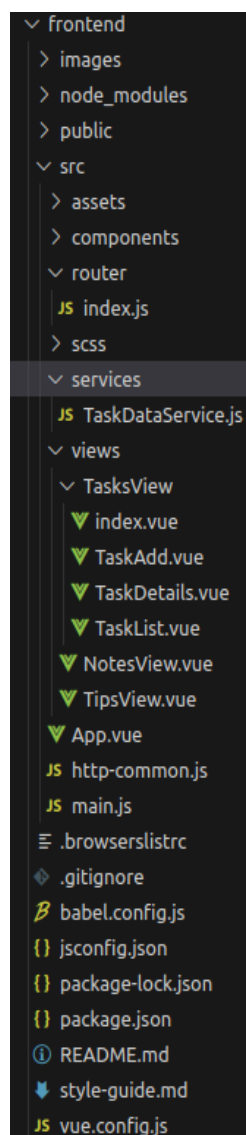
The App component is a container with router-view. It has sidebar with navigation that links to route paths: TasksView, NotesView and TipsView. For now, we're only going to work on TasksView.

The idea is that TasksView communicate only with its components: TaskAdd, TaskDetails & TaskList via sending props, receiving custom events etc. These components call TaskDataService methods which use axios to make HTTP requests and receive responses. This way we have some kind of a structure.

Simple Diagram:



Project Structure



Briefly explanation:

- *package.json* contains three main modules: vue, vue-router and axios
- *TasksView* has three components: TaskAdd, TaskDetails and TaskList
- *router* folder with *index.js* file defines routes for View components
- *http-common.js* initializes axios with HTTP base URL and headers
- *vue.config.js* configures port for this Vue Client. In this case it is set to 3031.