# Node Rest api

Firstly, we install morgan, helmet, express, nodemon, mongoose, dotenv using npm.

## Mongoose

* Helps us to create mongo models using schema.

## Dotenv

* Gives us secret url which includes our database, username, server, password.
* We hide the details.

## Helmet

* Helps us secure our express apps by setting various HTTP headers.

## Morgan

* It is an HTTP request level middleware which logs the requests along with other information depending upon its configuration and the presets used. It proves to be very helpful while debugging and also if you want to create log files.

# Setting up the backend

* First we import everything using const = require syntax.
* To use, **dotenv**, we need to write **dotenv.config()** and then make a file.
* Connecting to mongoDB, we need to use **process.env.MONGO\_URL** in order to connect with mongoose.

## Middlewares

Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application’s request-response cycle. It basically means anything you put in the middle of one layer of the software and another.

Middleware used: -

* app.use(express.json())
* app.use(helmet())
* app.use(morgan("common"));

## Routes

Inside the users.js, every routes related to CRUD will be created and exported which will be directly used in the index.js of the REST Api.

## Models

* then, we make the userSchema and export it as a model

# Authentication routes

## Register User

* First, we create a post request in the thunderclient
* Then, we retrieve all the data from body of the request into our model
* Then the model is saved in the database, which is wrapped inside a try and catch block.

## Password should not be exposed, rather secured

* We use bcrypt for this.(npm i bcrypt);
* So, we hash the password by adding the generated salt with the original password.

## Login User

* First, we create a post request in thunderclient.
* The request will only accept email and password from the user.
* We use User.findOne as there’s only one document with the same email in the database.
* For checking the password, we use bcrypt compare function.

# user routes

## Update user

* So, we user put request for updating a user.
* First we check the user id and its admin information, to verify that the person trying to update the information is really the person.
* For updating the password, we need to write a separate code as we need to bcrypt the new password too.
* For other information, we can just send the body of the request to the information edited by the user using **$set: req.body**

## Delete user

* We simply use delete method of js in order to delete the user if the user’s credentials are valid.

## Get User

const { password, updatedAt, ...other } = user.\_doc;

        res.status(200).json(other);

* The above code specifies that we don’t show the properties like password and updatedAt in while fetching the user from the user document

## Following and unfollowing a user

* Since we are updating a kind of list here, we use PUT request.
* First, we check whether the person we are trying to follow is not ourselves.
* Then, if the user id of the person does not match with our own user id (which means we are not trying to follow ourselves), we find the id of the user to be followed.
* We push while following the user and pull while unfollowing the user. The code happens to be fairly similar.

*And after this, all of the user requests have been made.*

# posts

## Creating the model

* A post which will be uploaded by a user will have title, description, image etc.

## Creating the routes

* After adding the posts.js route as postRoute in the index.js file, we start to create all the required requests of the posts.

# Frontend