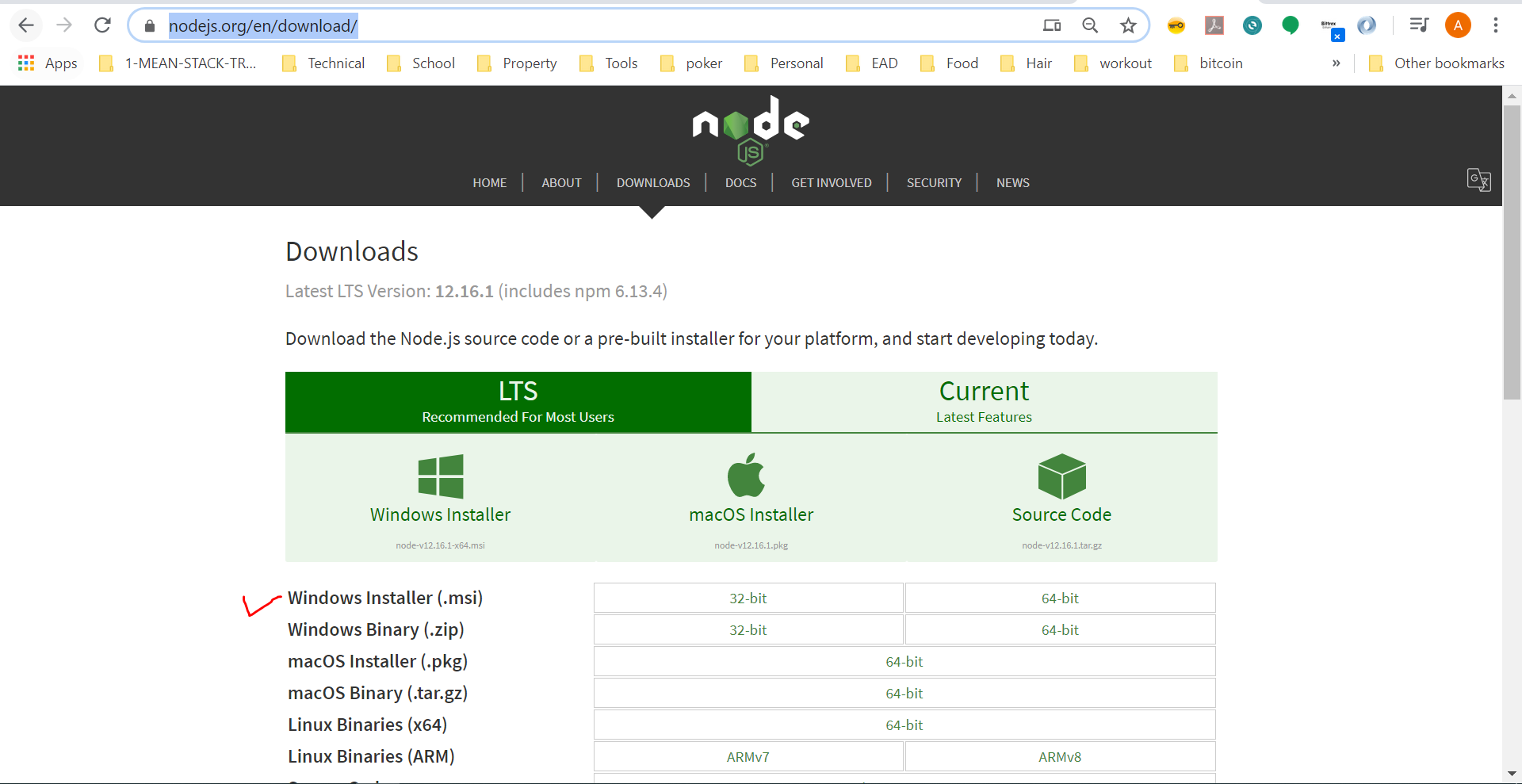
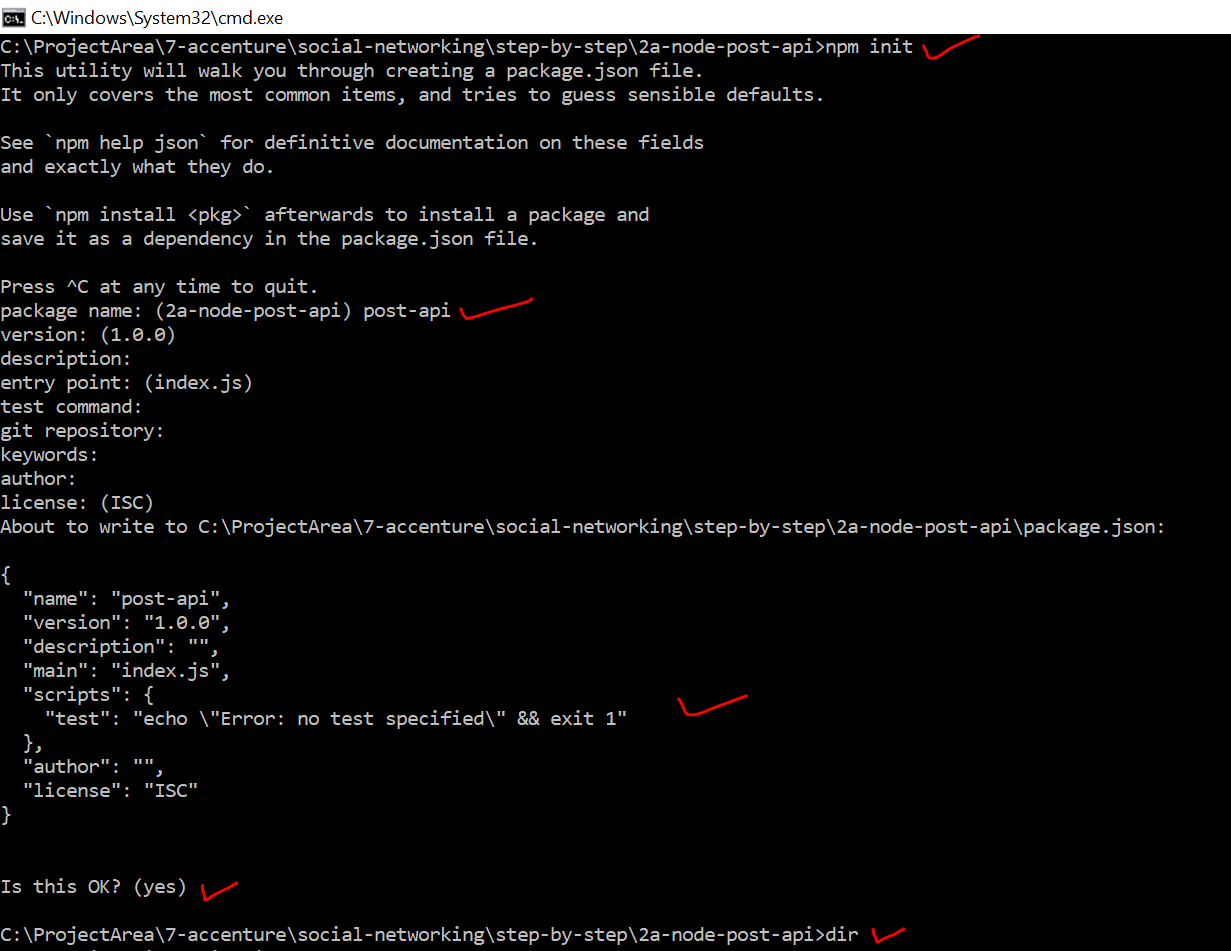
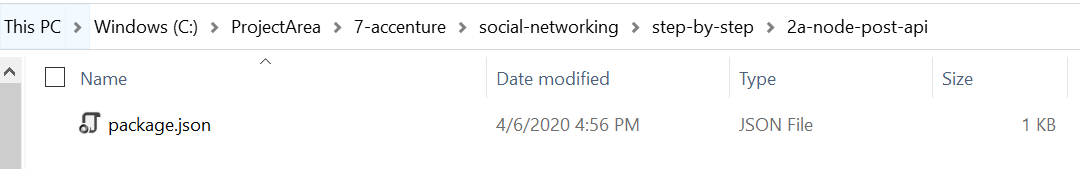
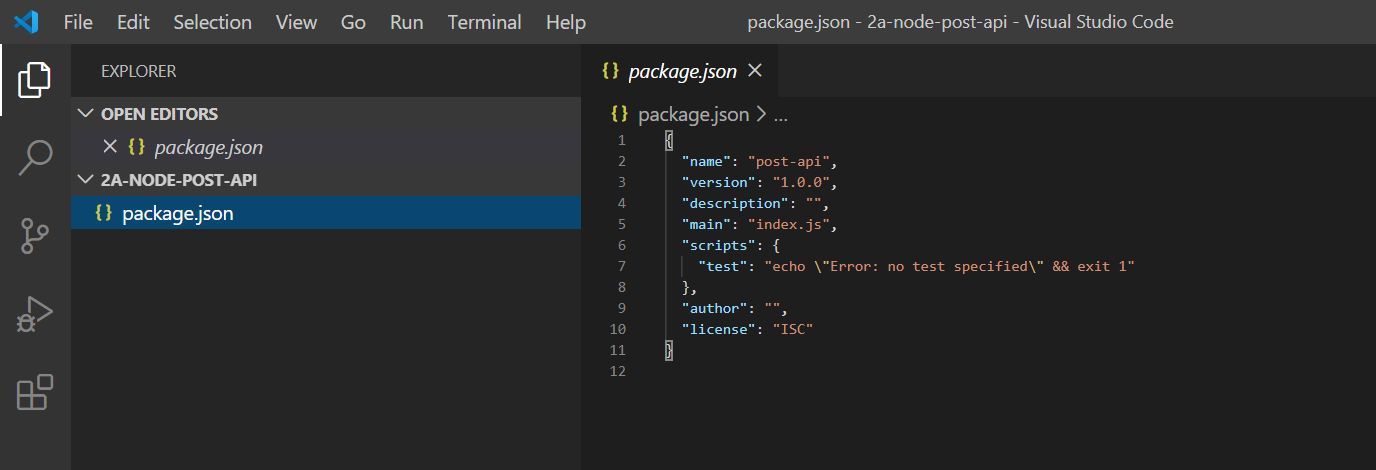
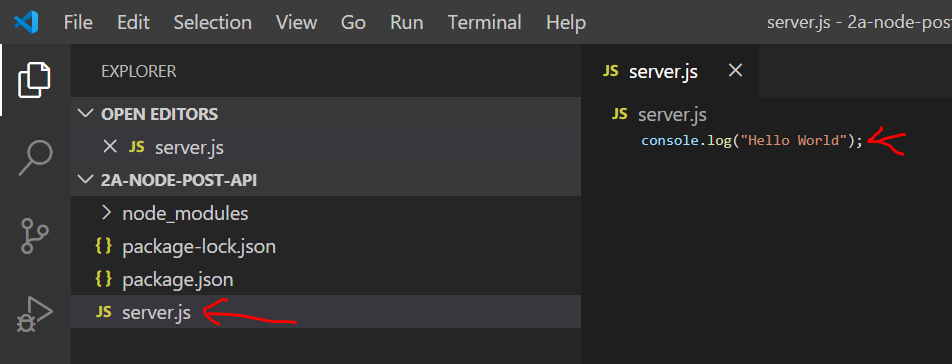
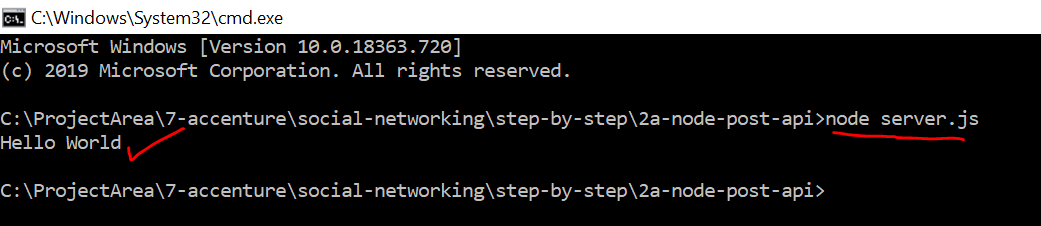
Installation

1. Install node runtime from nodejs.org - <https://nodejs.org/en/download/>
   1. 
2. **Create a node project folder by name “post-api”** 
   1. **Inside the folder execute a command “npm init” to start a new node project**
   2. 
3. After the above step we have single file called package.json created to begin with
   1. 
4. Open the project folder in visual studio code and observe the package.json
   1. 
5. As node can execute a JavaScript code , lets write a simple hello world JavaScript program and execute inside node runtime. Add a file called server.js and write a simple code as shown below. NOTE : Please note this is not a server yet
   1. 
   2. Run command -> node server.js
   3. Observe the output
   4. 
6. Let’s convert this to a server by importing http package
   1. Copy the below code to server.js file and execute command -> node server.js

// create a server variable

const http = require("http");

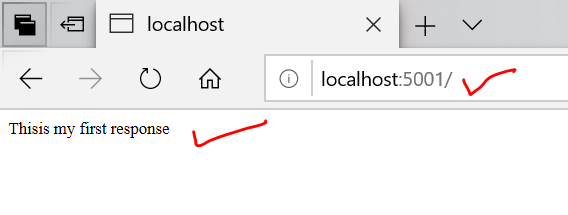
const server = http.createServer((req, res) => {

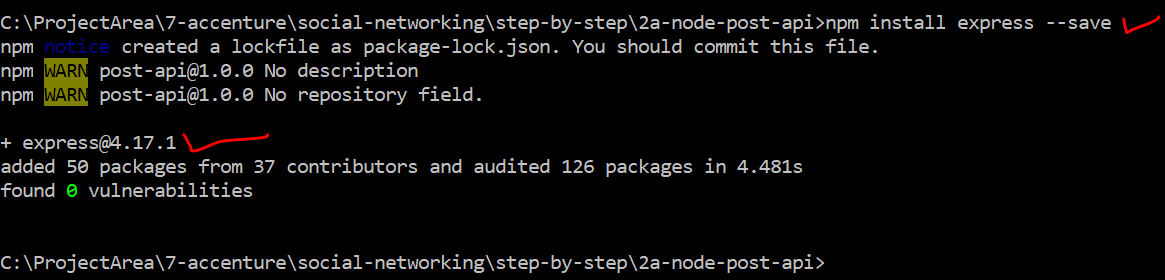
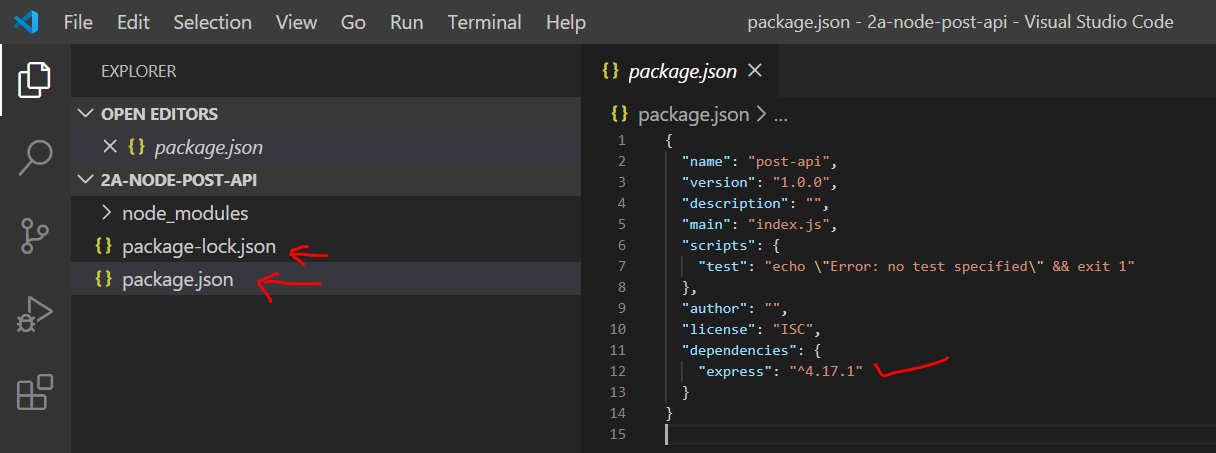
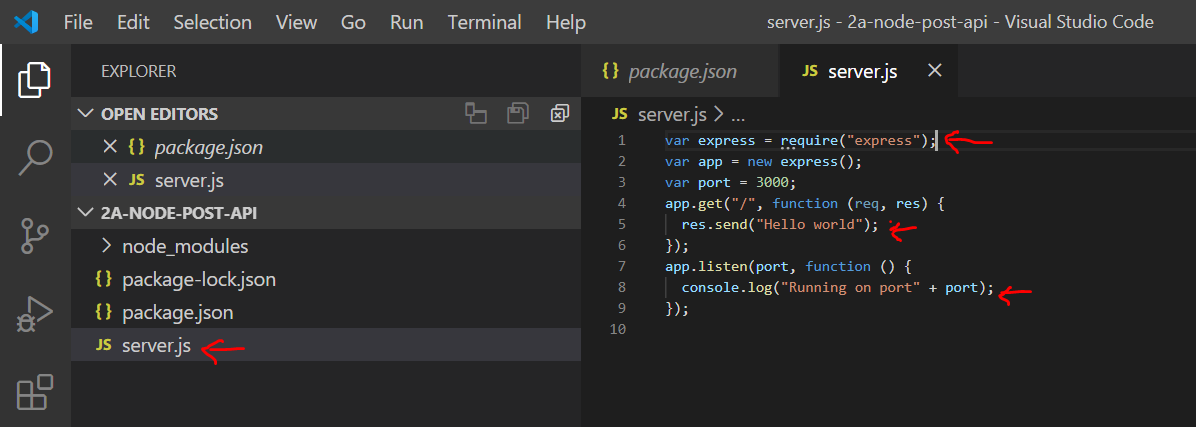
  res.end("This is my first response");

});

// listen at a port

server.listen(5001);

* 1. Observe the response in browser
     1. 

1. Before we move on to next step understand what are RESP APIs
   1. REST API
   2. <https://restfulapi.net/>
2. As server needs to listen to http request and understand GET, POST, PATCH, DELTE request verbs and parse request data, we need to write a lot of code in nodejs to achieve this
3. This is made easy by using another package called express
4. Let’s listen to the client request using express package
5. **Install Express by running command “npm install express --save”**
   1. 
6. **Open the code is editor and notice package.json , express dependency is installed**
   1. 
7. Simple ‘Hello World’ using express
   1. Make below changes to server.js and observer the highlighted areas
   2. 
   3. Include the below Code in server.js

var express = require("express");

var app = new express();

var port = 5000;

app.get("/", function (req, res) {

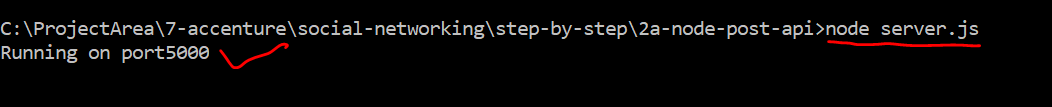
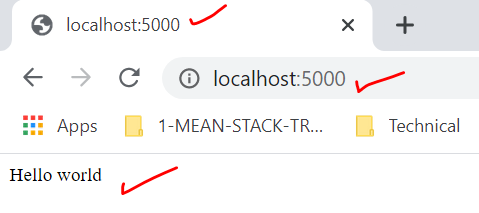
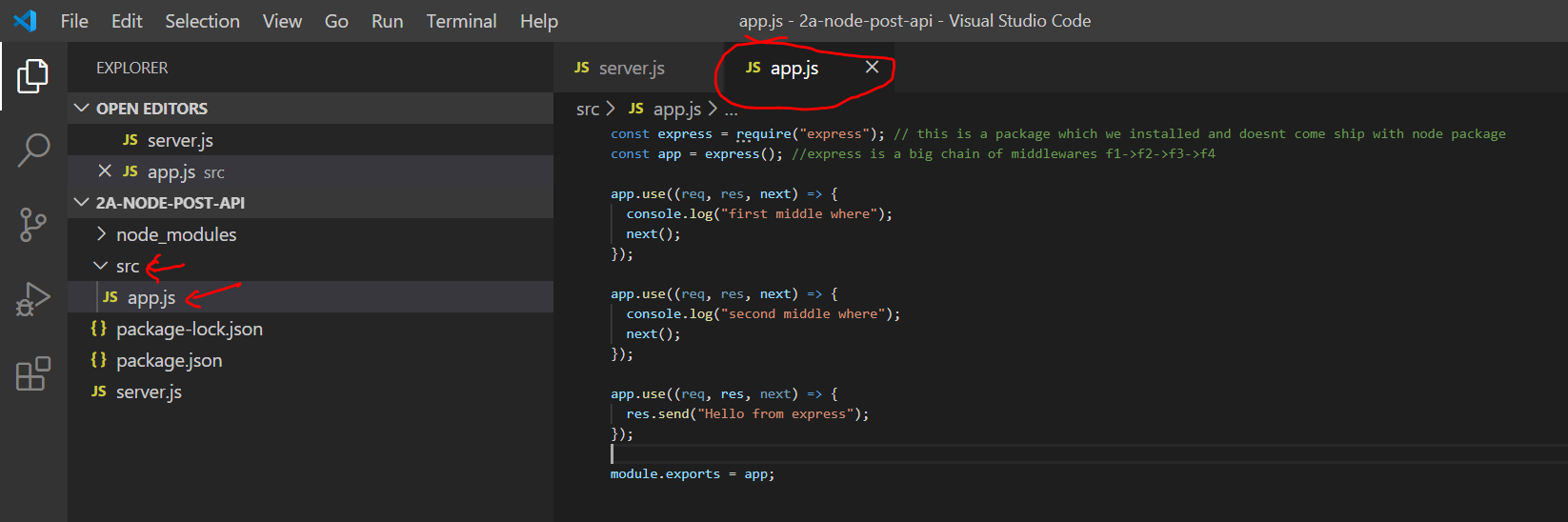
  res.send("Hello world");

});

app.listen(port, function () {

  console.log("Running on port" + port);

});

1. Run the server and check the output
   * 1. Run command -> “node server.js”
     2. 
2. Type localhost:5000 is chrome(browser) to get the message
   1. 
3. Middleware concept using express – express is nothing but a chain of middleware’s
   1. Create a new folder called src
   2. Create app.js file and include below code
   3. 
   4. Code – include in app.js

const express = require("express"); // this is a package which we installed and doesnt come ship with node package

const app = express(); //express is a big chain of middlewares f1->f2->f3->f4

app.use((req, res, next) => {

  console.log("first middle where");

  next();

});

app.use((req, res, next) => {

  console.log("second middle where");

  next();

});

app.use((req, res, next) => {

  res.send("Hello from express");

});

module.exports = app;

* 1. Include the above exported app object into server.js file with below code

const http = require("http");

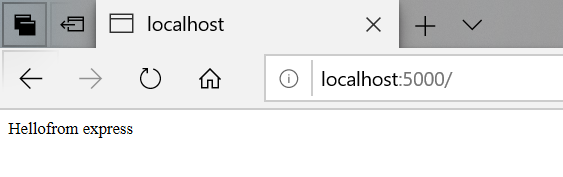
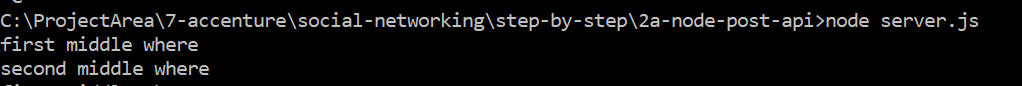
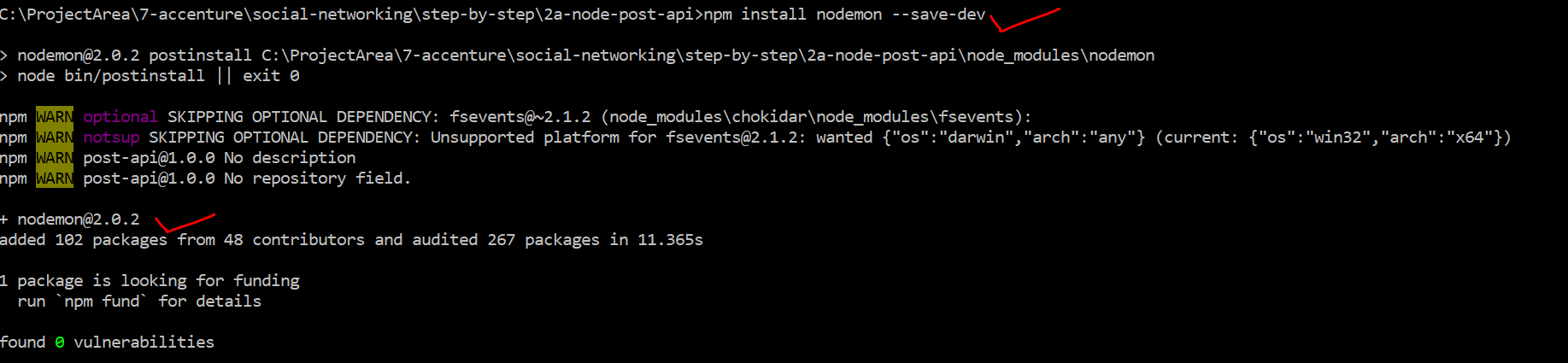
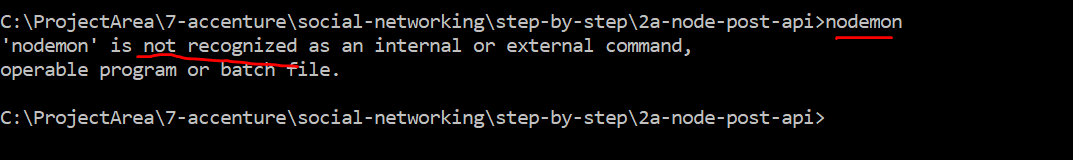
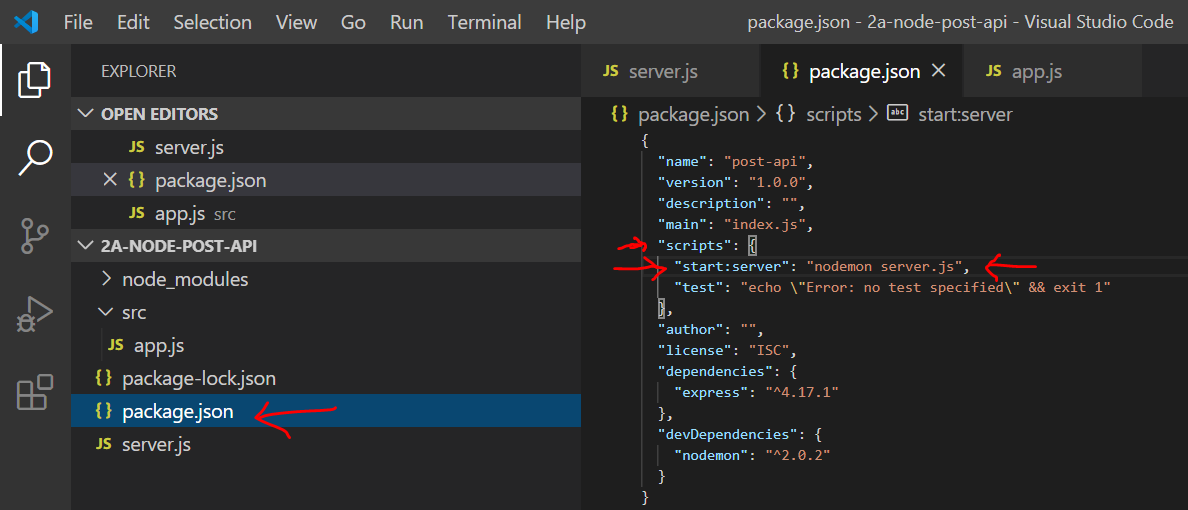
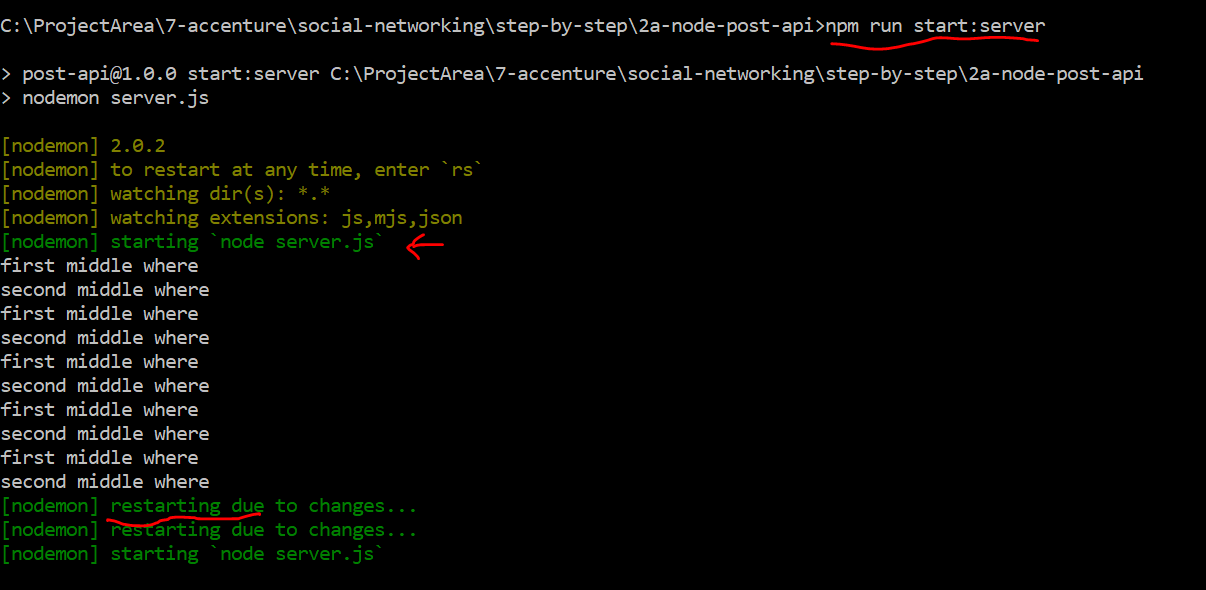
const app = require("./src/app");

const port = process.env.PORT || "5000";

app.set("port", port);

const server = http.createServer(app);

server.listen(port);

1. Execute the project and see the output in browser and console
   1. 
   2. 
2. Every time we are doing code changes, we are killing the server and restarting. If we have to make our server recompile and restart automatically whenever new code changes happen, we can do this with help of nodemon
   1. Run command -> npm install nodemon --save-dev
      1. 
   2. NOTE : nodemon is installed locally for this project only and it won’t run the command
      1. 
   3. In-order to run nodemon create a custom command as shown below in package.json under scripts section
      1. 
      2. Run the server and do some code changes and observe that server restarts automatically
         1. 
3. Create get api
   1. Paste the below code in app.js and run <http://localhost:5000/api/posts>

const express = require("express");

const app = express();

app.get("/api/posts", (req, res, next) => {

  const posts = [

    {

      id: "fadf12421l",

      title: "First server-side post",

      content: "This is coming from the server",

    },

    {

      id: "ksajflaj132",

      title: "Second server-side post",

      content: "This is coming from the server!",

    },

  ];

  res.status(200).json({

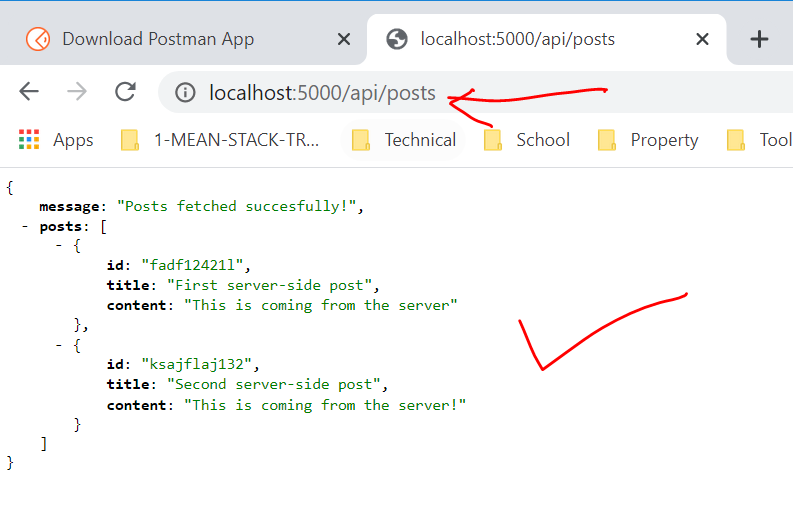
    message: "Posts fetched succesfully!",

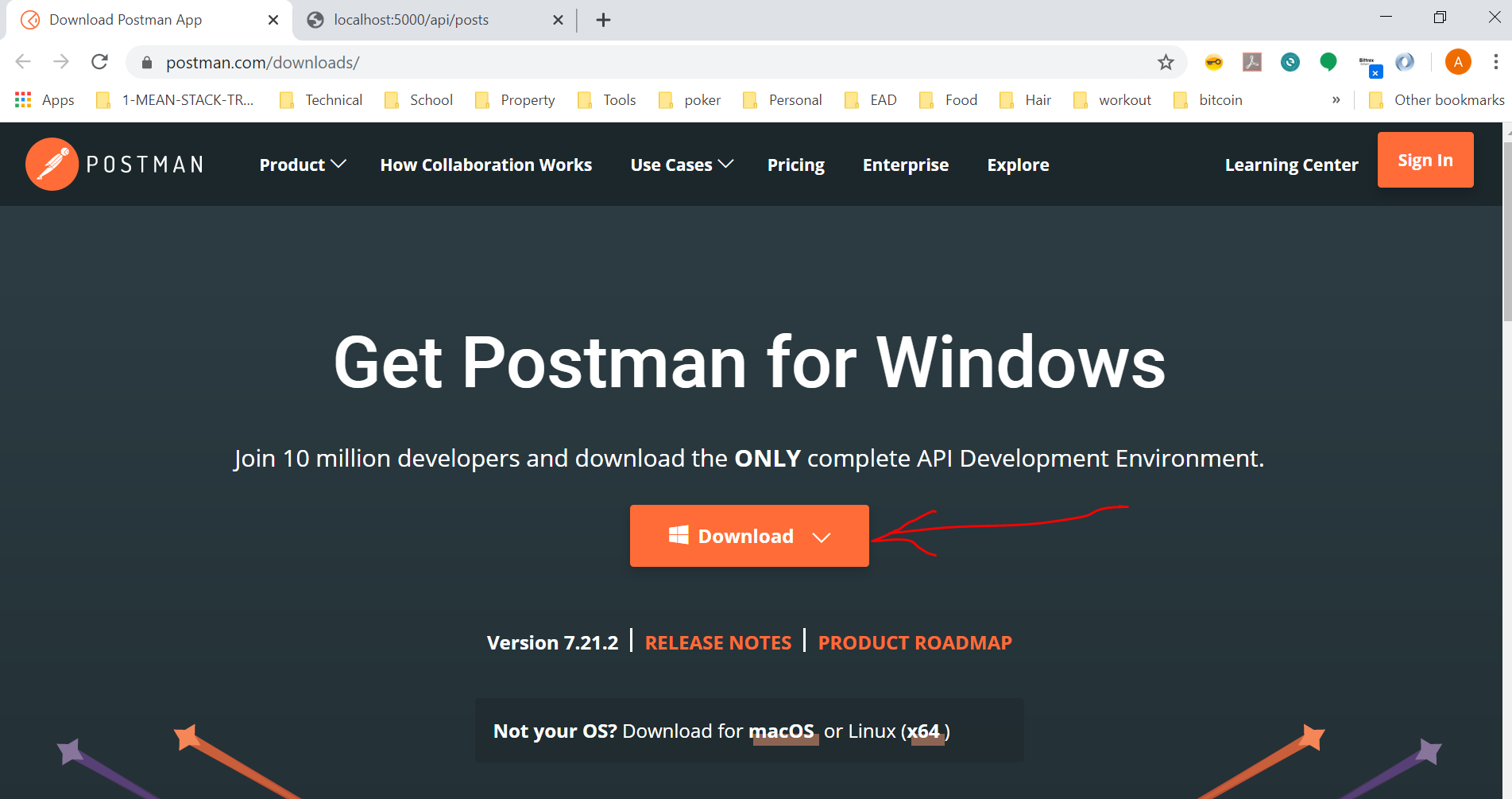
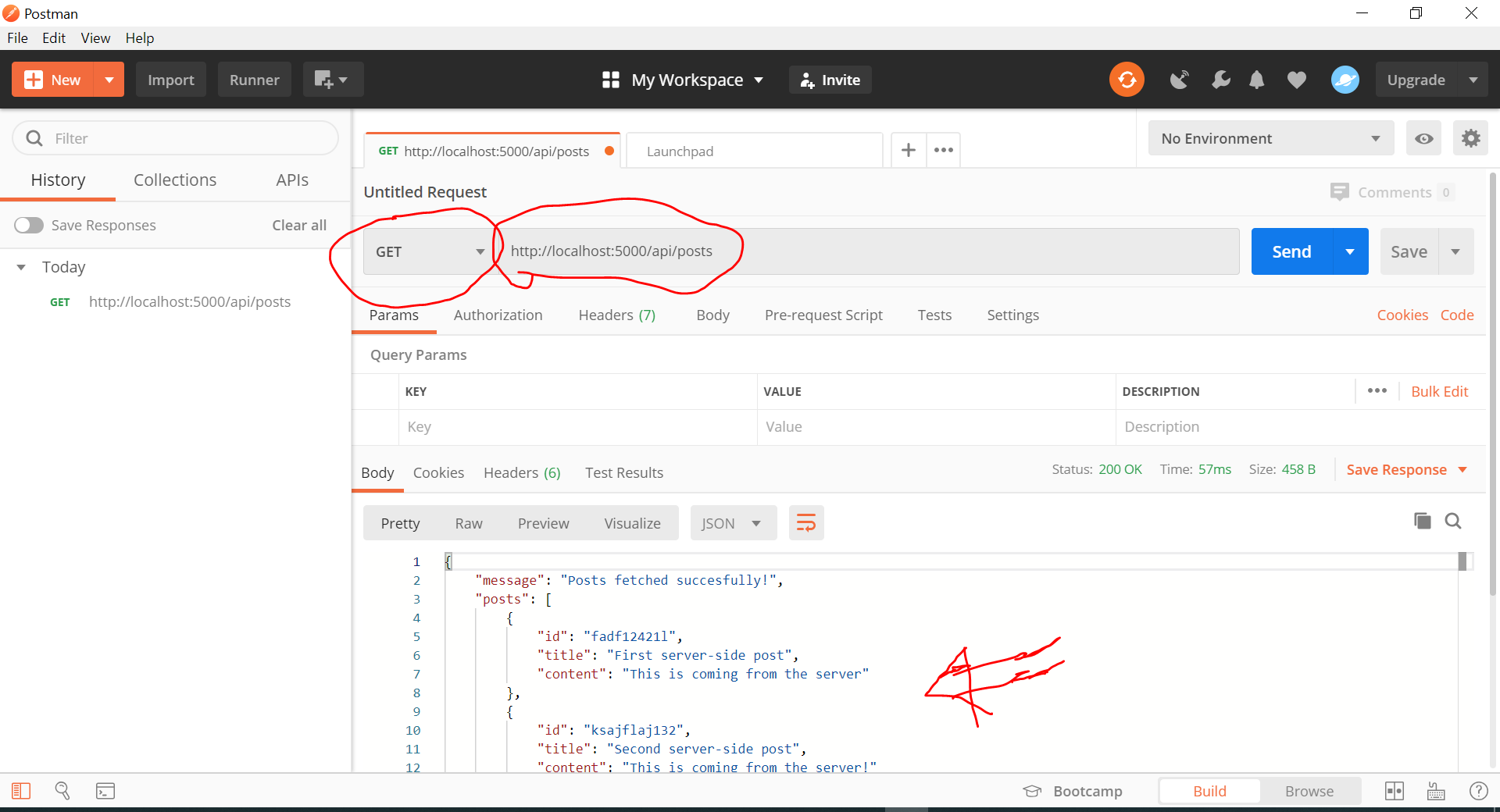
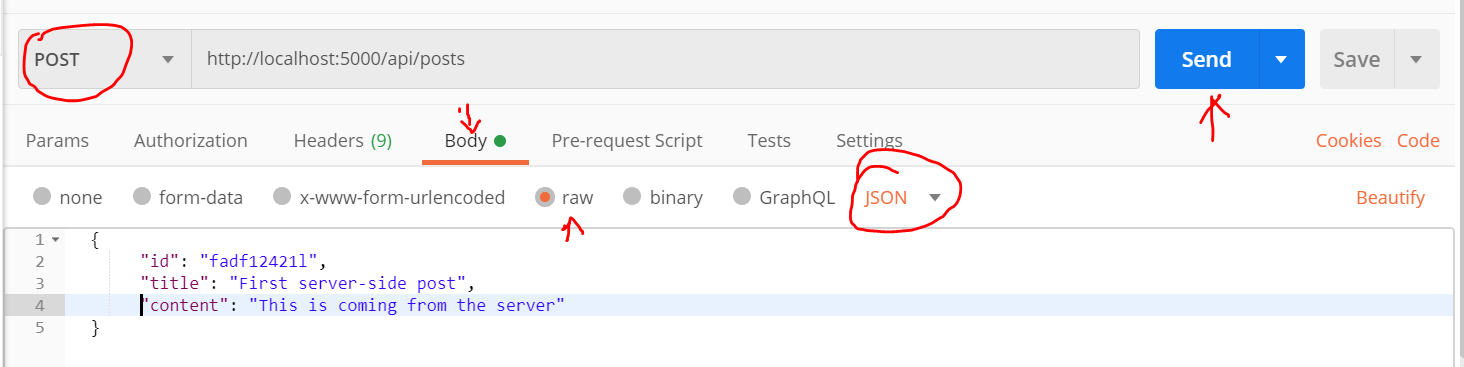
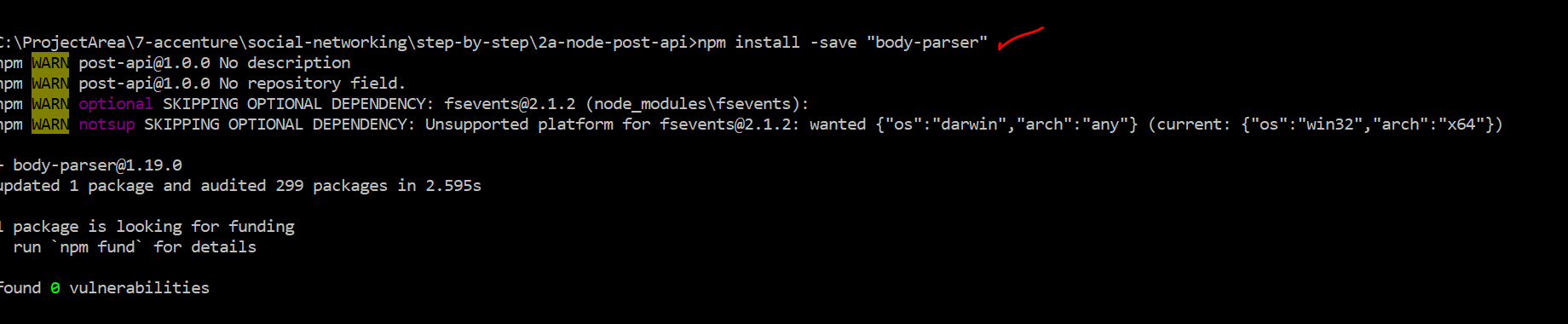
    posts: posts,

  });

});

module.exports = app;

* 1. Observer the output
  2. 

1. Install POSTMAN tool from below site- <https://www.postman.com/downloads/>
   1. Postman tool will be used perform all types http requests
   2. 
   3. For example: At this point we have GET API and that can be executed in POSTMAN as below
      1. 
2. Next step is to Create Post API – This API is responsible for creating out POST records
   1. NOTE: User can initiate only HTTP GET requests using browser, can not do POST or PUT or DELETE requests.
   2. Our requirement is to initiate a HTTP POST request using POSTMAN tool as shown below and get a response saying the record was created successfully
      1. 
   3. We have not created POST API in server yet
   4. POST API should accept data from POST http request body
   5. The POST http request body data need to be extracted from the request and parsed made available in request variable at the server side. This is performed by a middleware package called “body-parser”. Let’s install “body-parser” with below command
      1. npm install –save “body-parser
      2. 
      3. This is a node express package , which is used as middlerware.
   6. Lets start building the POST API with below code
      1. Import the package in app.js
         1. const bodyParser = require("body-parser");
      2. Lets attach this middleware into our request pipeline with below code

// Lets attach the body-parser middleware

// bodyParser.json() -> this will tell only to process json type data from the request body

app.use(bodyParser.json());

//another example showing body-parser can process other types of body other than json

app.use(bodyParser.urlencoded({ extended: false }));

* + 1. Add below post API code

app.post("/api/posts", (req, res, next) => {

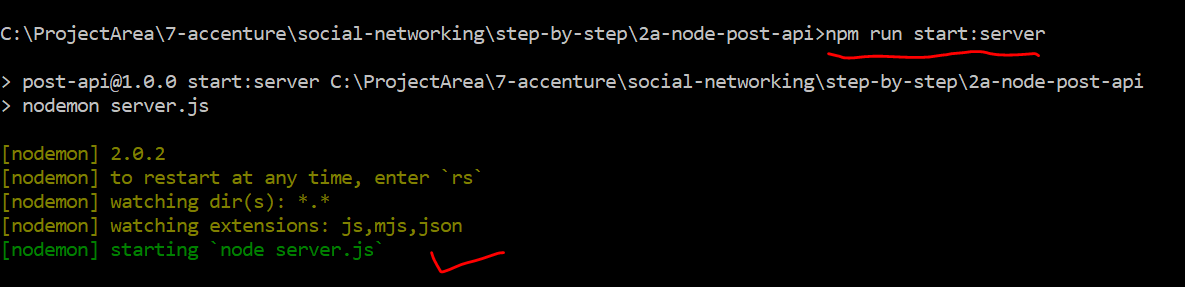
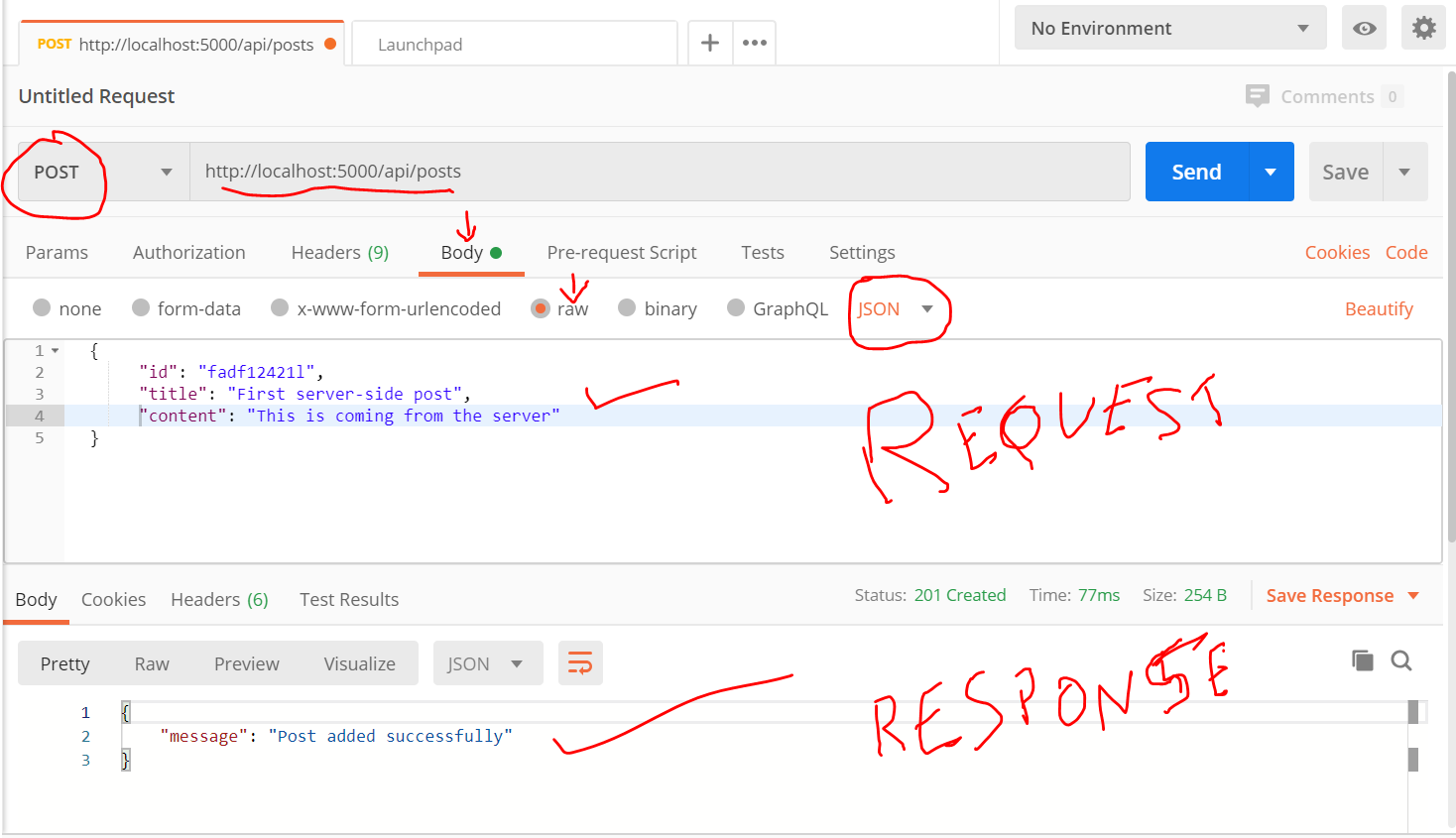
  const post = req.body;

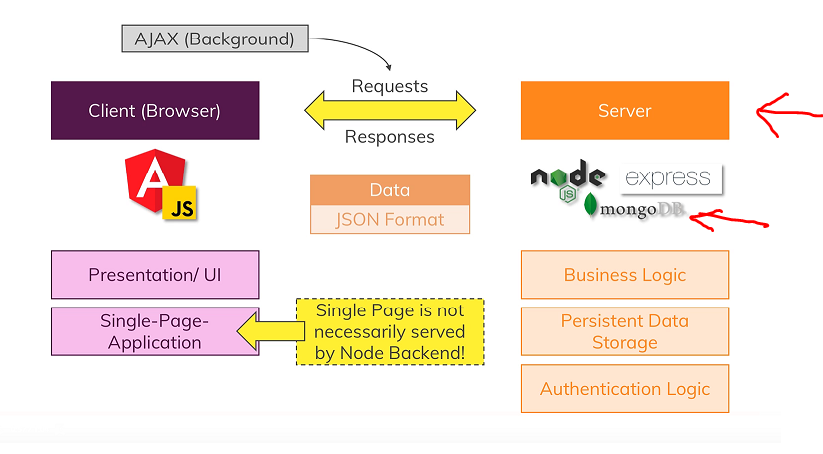
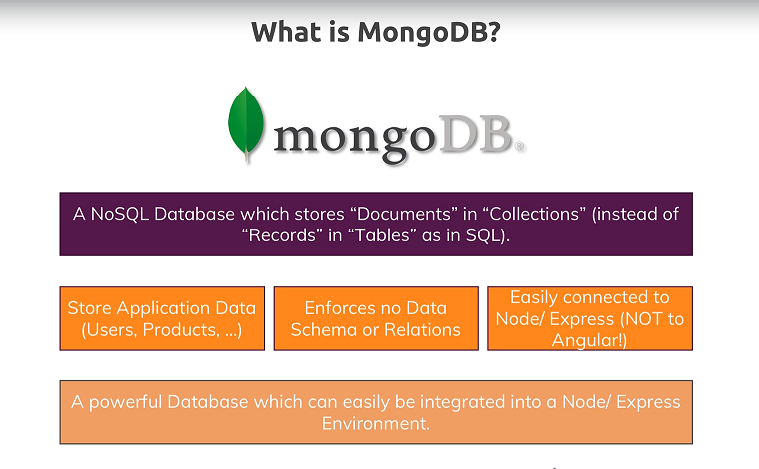
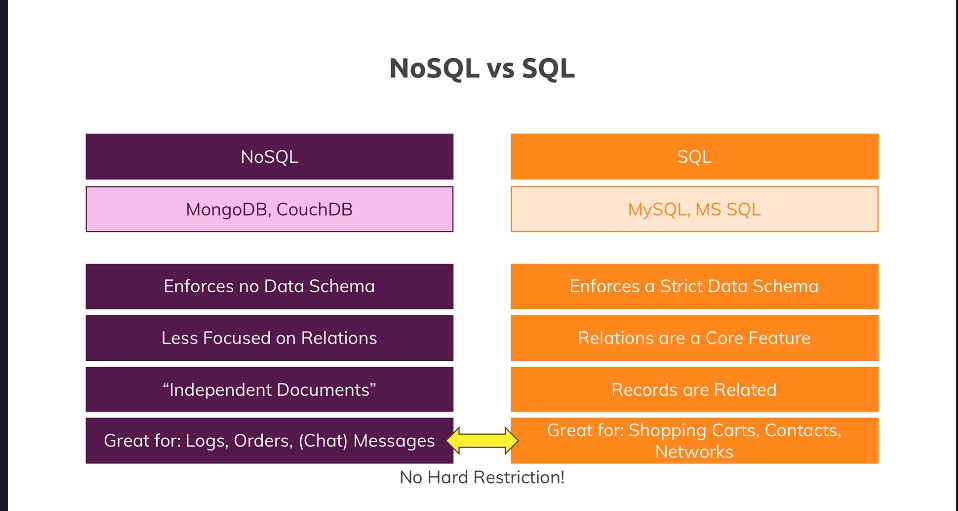
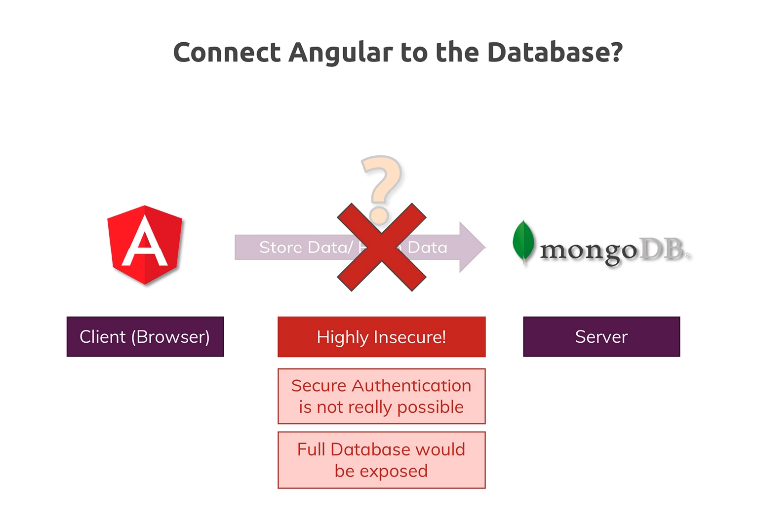
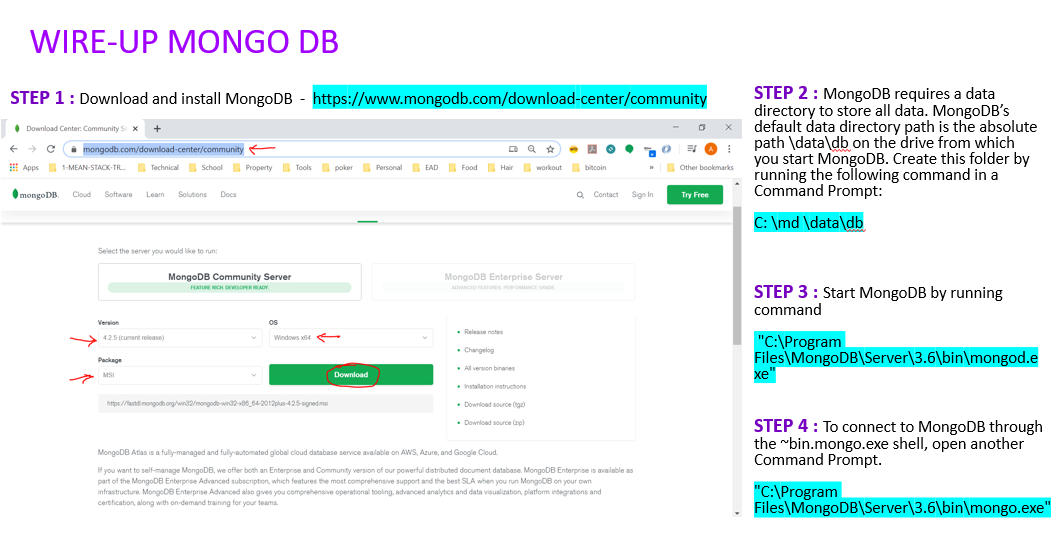
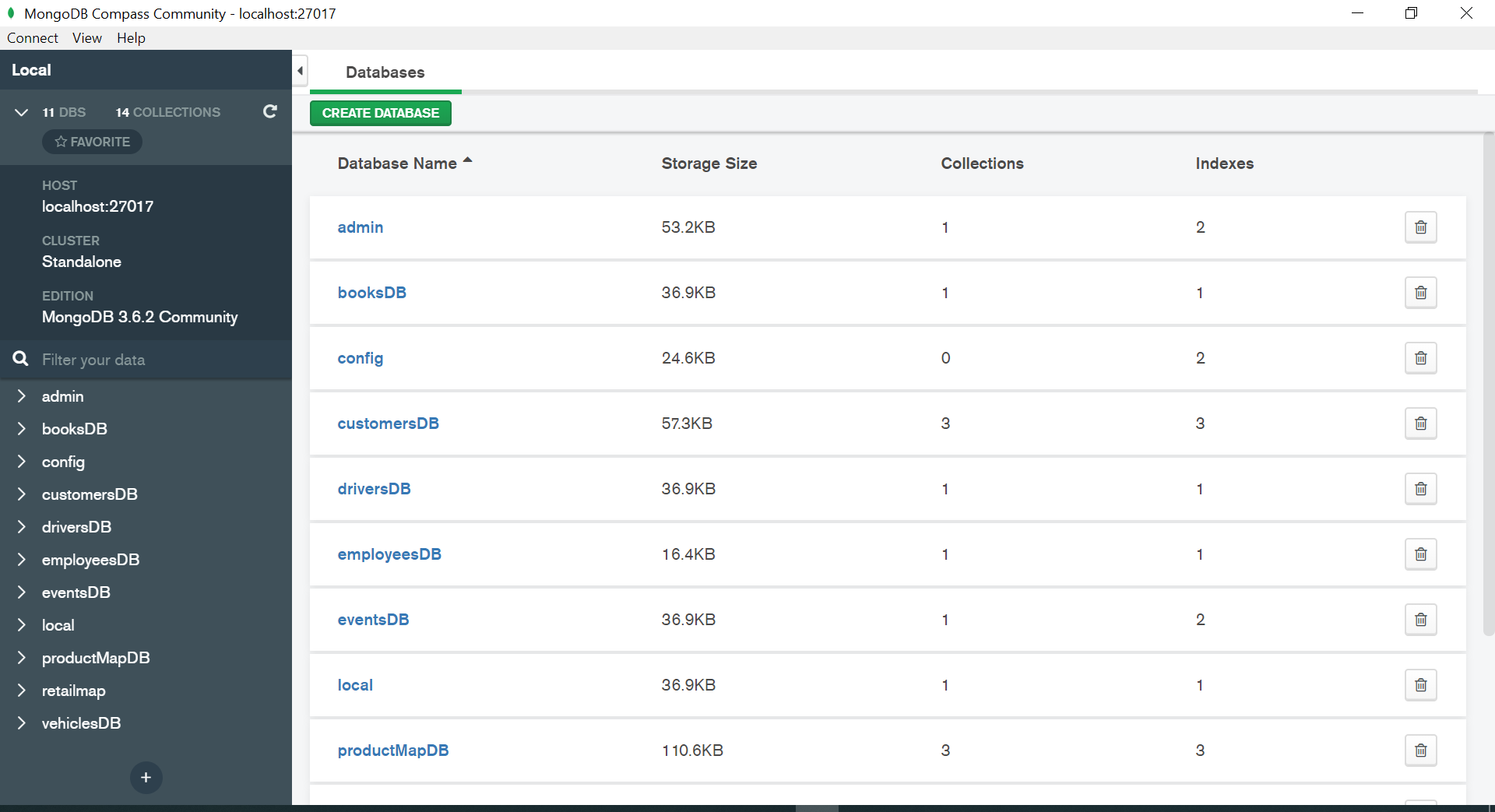
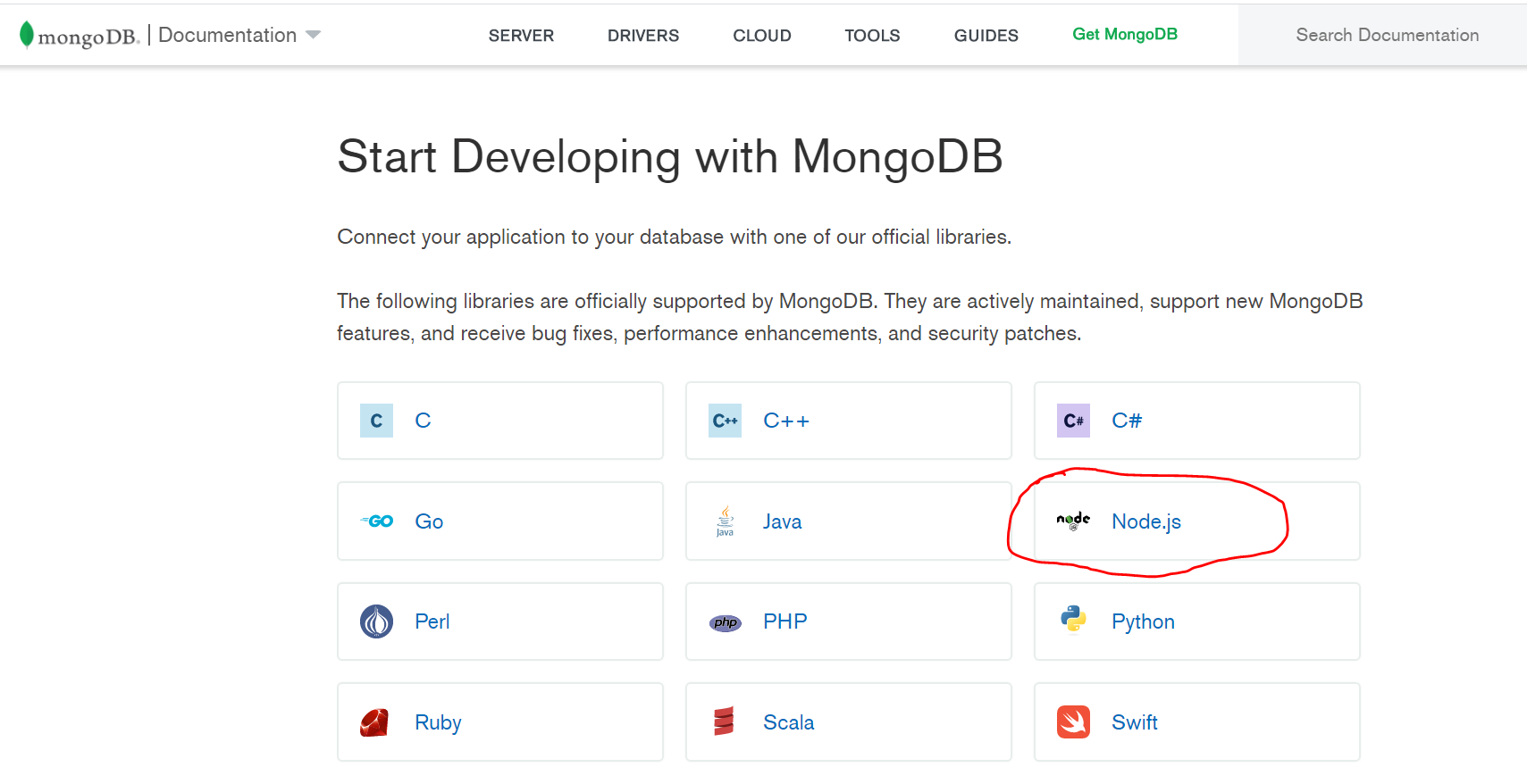
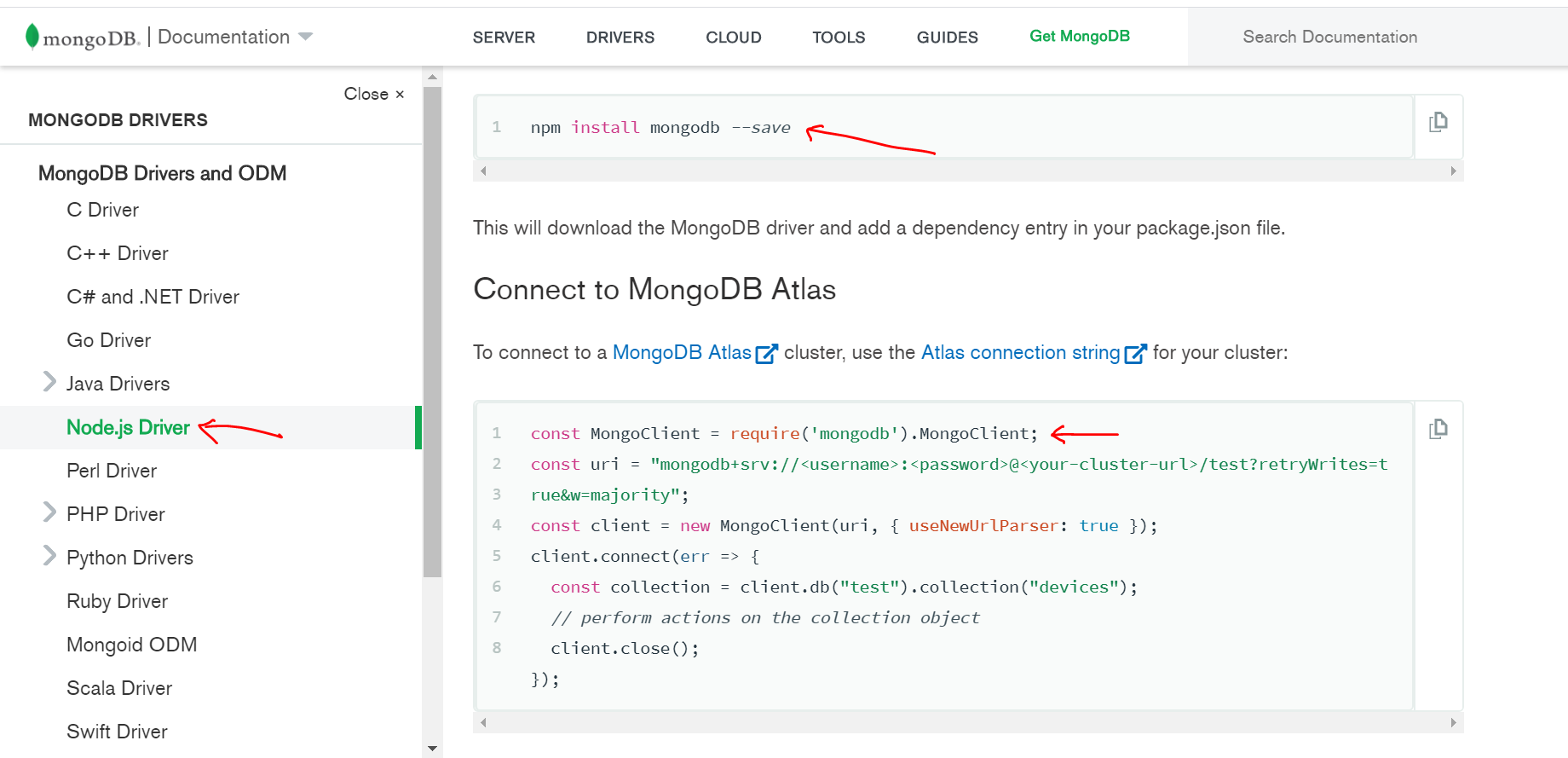
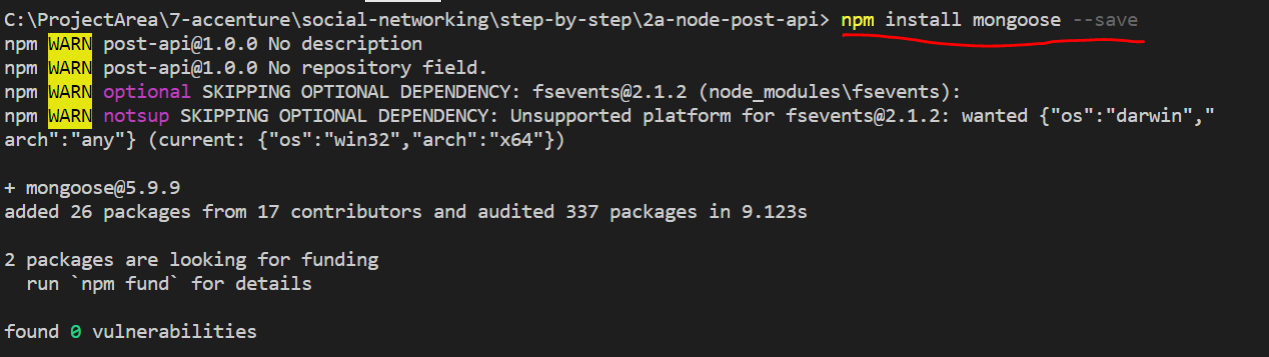
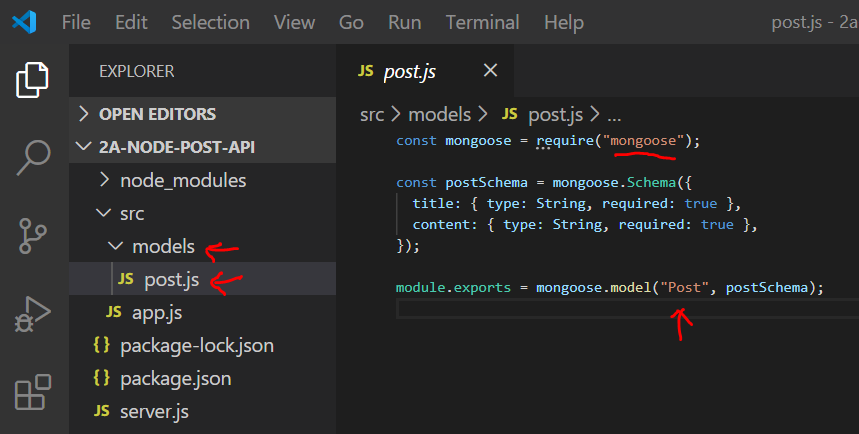
  res.status(201).json({

    message: "Post added successfully",

  });

});

* + 1. Start the server using command
       1. 
    2. Initiate the POST request in POSTMAN and observe the response
       1. 

1. As we are handling data in angular app and when we close the app the data is gone, we need to persist the data in a database. The Next step is to store the data from POST API into a mongoDB
2. Observer the overall architecture and where we are going to work next and connecting to mongoDB
   1. 
3. What is Mongo DB
   1. 
4. Why are we using mongodb and why are we using NOSQL db and not SQL
5. NOSQL VS SQL
   1. 
6. Can we connect our angular application directly to mongodb
   1. Technically we can connect our angular application to mongodb, but we should not do as it will pose security issues like connection string expose to clients. User authentication not possible at monodb level.
   2. 
7. Wire up MongoDB
   1. There are 2 ways to wire up mongo dB
      1. Download mongo from below link and install locally
         1. <https://www.mongodb.com/download-center/community>
      2. Using MongoDB cloud storage - MongDB Atlas
         1. Out of scope of this training
   2. 
8. In the previous step we setup the mongodb, Below are the different ways to connect and talk to mongoDB
   1. Mongo Shell -> Command line tool
   2. MongoDB Compass tool
      1. 
      2. <https://docs.mongodb.com/compass/current/>
   3. MongoDB Drivers
      1. <https://docs.mongodb.com/drivers/>
      2. 
   4. As we are using nodejs we can use below driver
      1. <https://docs.mongodb.com/drivers/node/>
      2. 
      3. This uses mongoclient using which we can talk to mongoDB directly
         1. <http://mongodb.github.io/node-mongodb-native/3.5/tutorials/collections/>
      4. MongoDB Manual for CRUD operation
         1. <https://docs.mongodb.com/manual/crud/>
   5. For the scope of this training we are using a 3rd party package “mongoose” package which makes accessing mongodb easier and convenient
      1. <https://mongoosejs.com/docs/>
9. Install mongoose using “npm install mongoose –save”
   1. 
10. MongoDB doesn’t recommend having a schema as it is meant to store unstructured data. But in our case if we want to enforce schema, we need to do this at nodejs level. This is achieved by mongoose.
11. Mongoose is built on top of mongodb driver
12. As we are enforcing a schema for our example by using mongoose, let’s create a schema as below
    1. Create a folder called Model and create a file called posts.js
    2. 
    3. Mongoose schema takes a javascript object
    4. We can have different types of schema; schema is just a blueprint
    5. We create a model out of schema and export it for external use
    6. Code

const mongoose = require("mongoose");

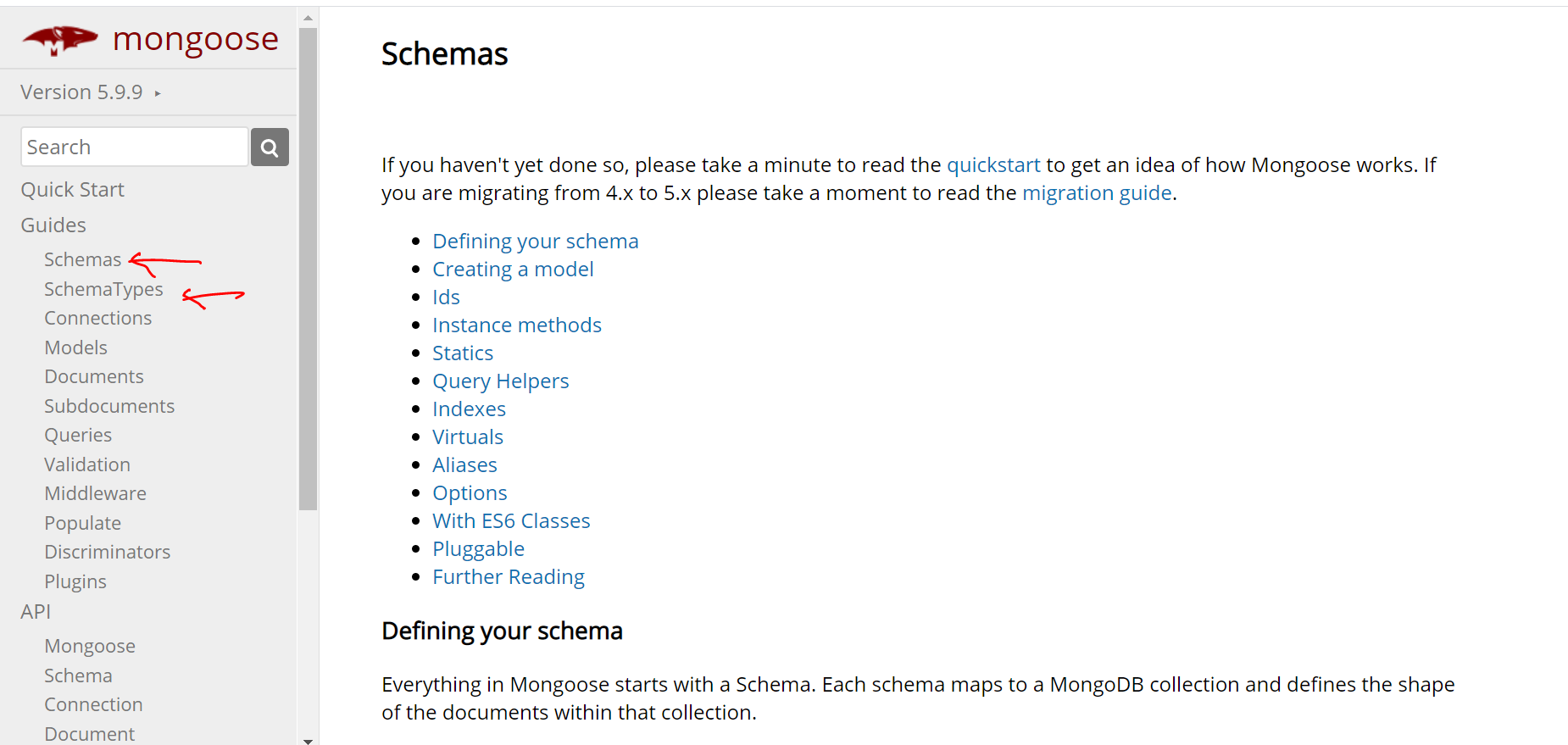
const postSchema = mongoose.Schema({

  title: { type: String, required: true },

  content: { type: String, required: true },

});

module.exports = mongoose.model("Post", postSchema);

* 1. More about mongoose schemas below - <https://mongoosejs.com/docs/guide.html>
     1. 

1. Lets create a POST API using the schema and model created in the previous step
   1. Import the “Post” Model in app.js

const Post = require("./models/post");

* 1. Change the POST API code as below
  2. The code is creating a new Post model and accepting a javascript object in the constructor with values from the http post request body

app.post("/api/posts", (req, res, next) => {

  const post = new Post({

    title: req.body.title,

    content: req.body.content,

  });

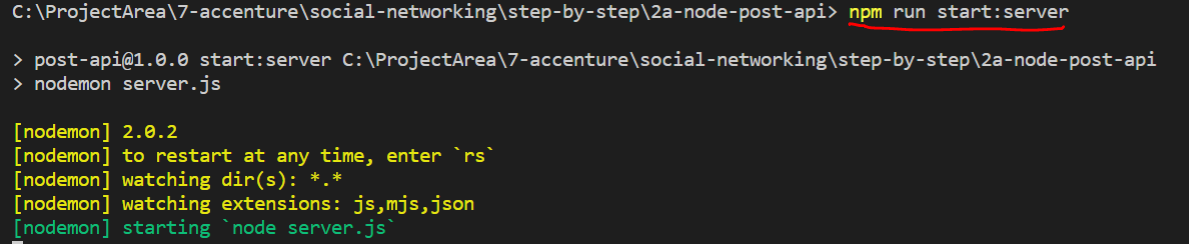
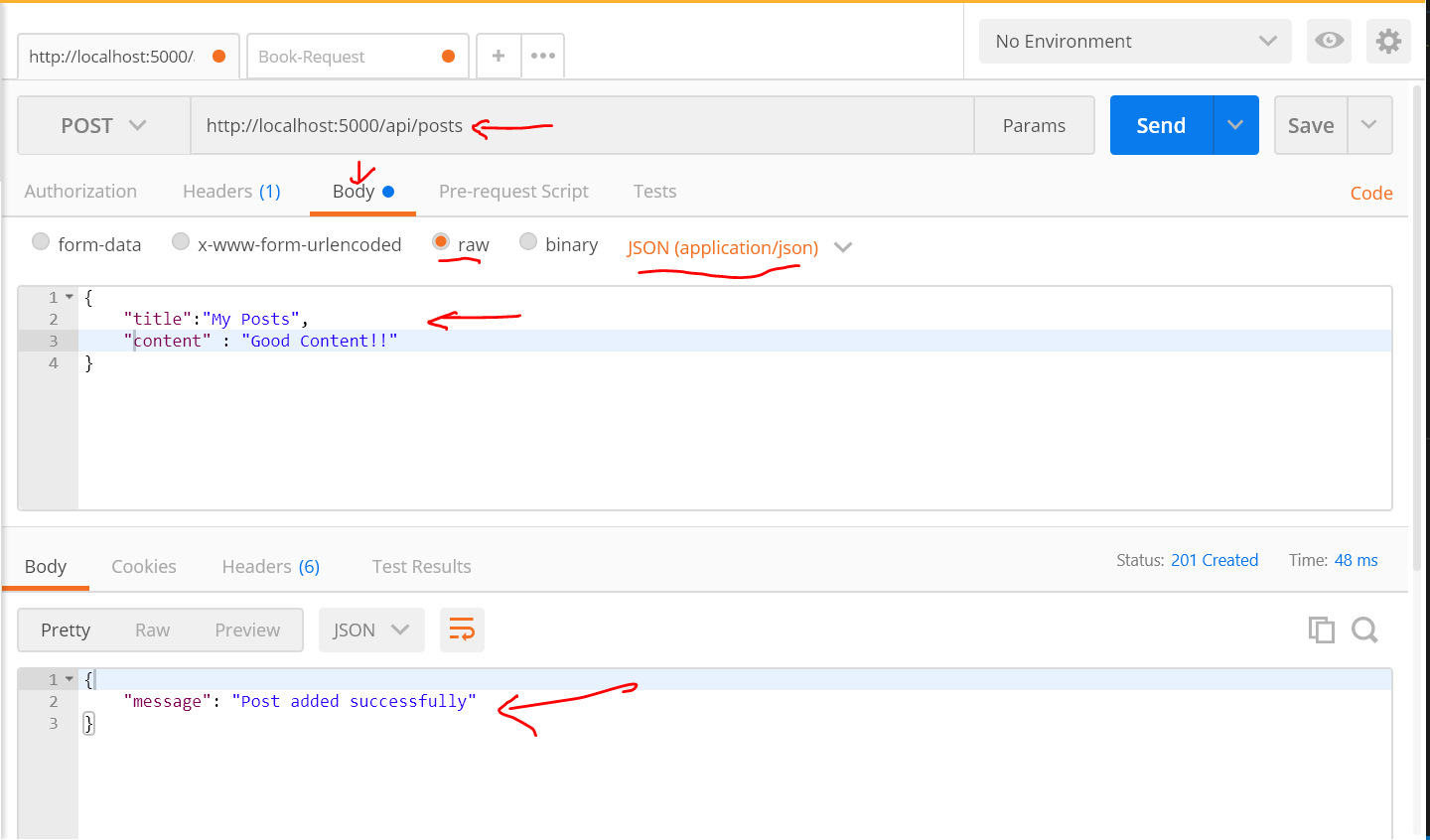
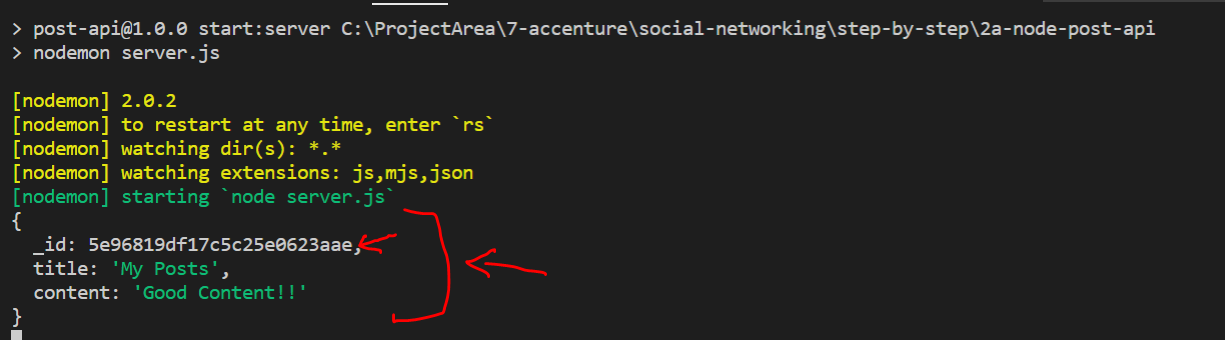
  console.log(post);

  res.status(201).json({

    message: "Post added successfully",

  });

});

* 1. Let’s try to post some data from POSTMAN
     1. Start the server if not started already
        1. 
     2. Execute http POST request from POSTMAN tool
        1. 
     3. Check the logs on the server
        1. 
     4. Notice the auto generated id

1. Let’s save the above POST data object into mongoDB
   1. In order to save the data to DB we need to connect to mongoDB using below connection sting
      1. mongodb://localhost:27017/?readPreference=primary&appname=MongoDB%20Compass%20Community&ssl=false
   2. Import mongoose in app.js and connect to DB using below code

const mongoose = require("mongoose");

mongoose

  .connect(

    "mongodb://localhost:27017/?readPreference=primary&appname=MongoDB%20Compass%20Community&ssl=false"

  )

  .then(() => {

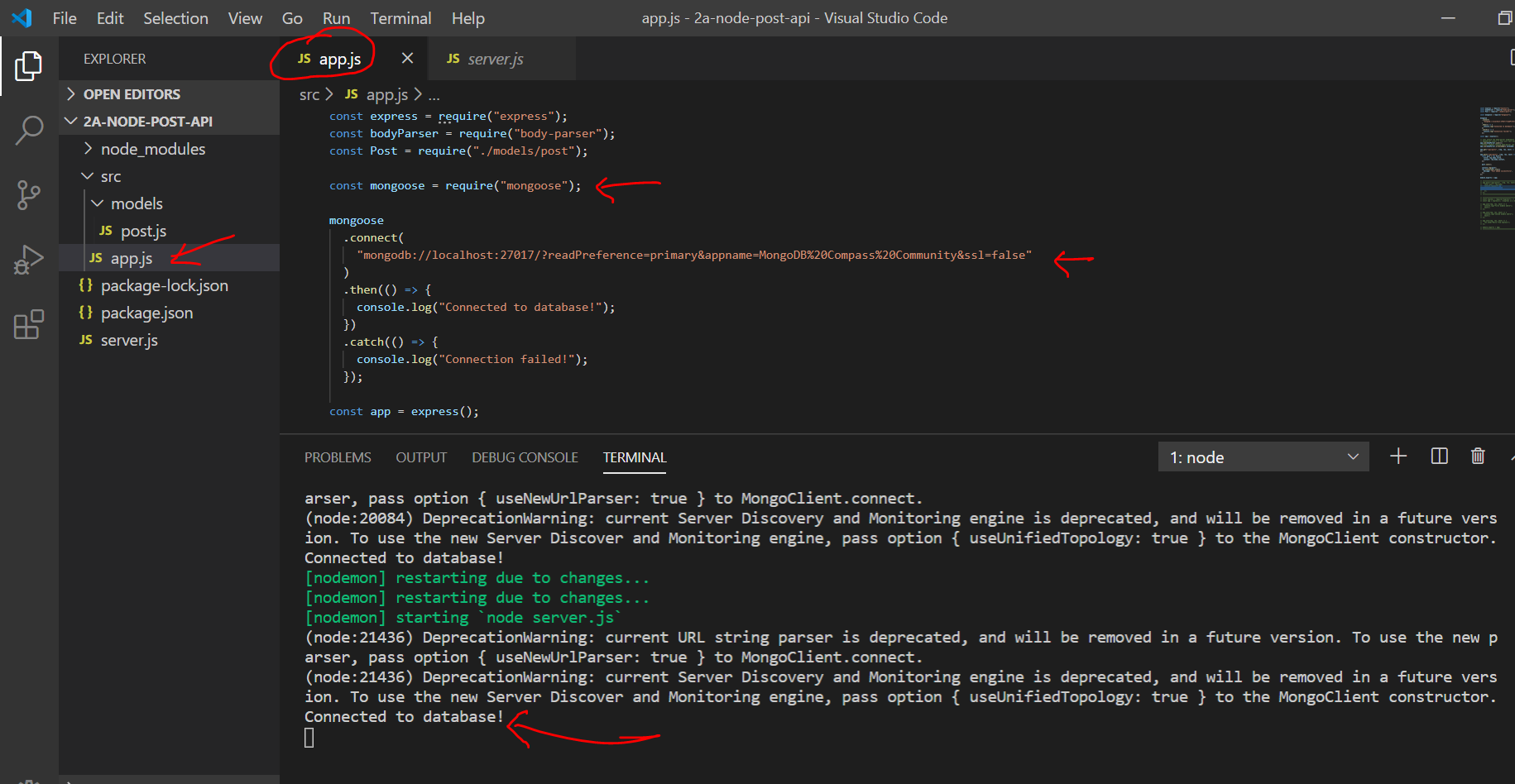
    console.log("Connected to database!");

  })

  .catch(() => {

    console.log("Connection failed!");

  });

* 1. After including above code and as soon as server is restarted notice the below highlighted message from command line saying “connection to Database”
  2. 

1. Now the final step to save the data into DB is to include the below highlighted code in POST API

app.post("/api/posts", (req, res, next) => {

  const post = new Post({

    title: req.body.title,

    content: req.body.content,

  });

  post.save();

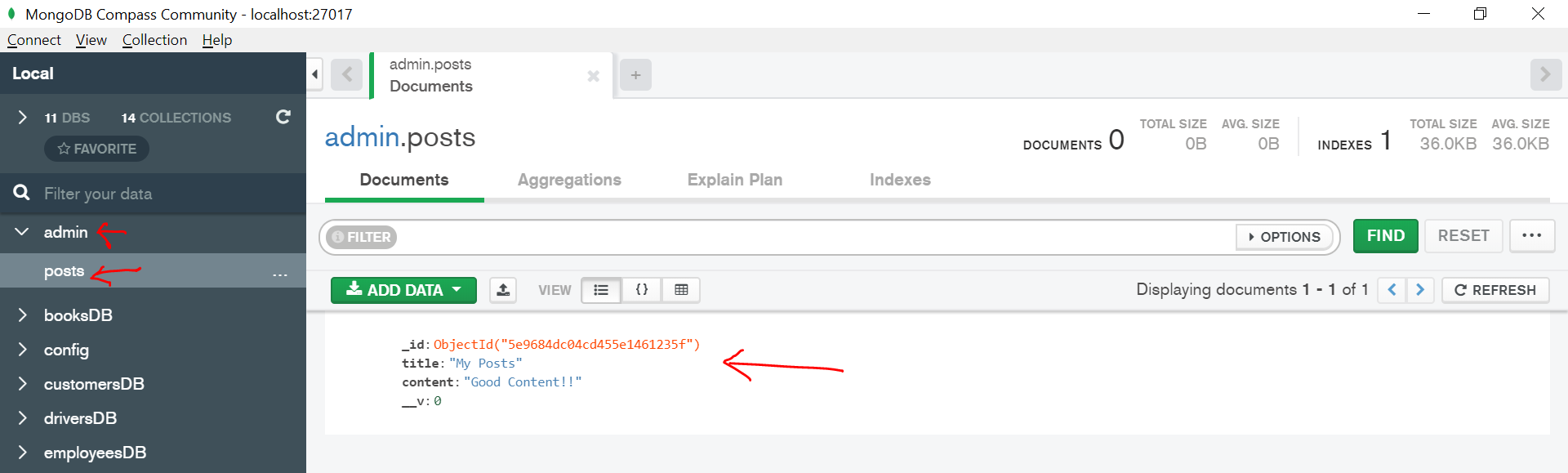
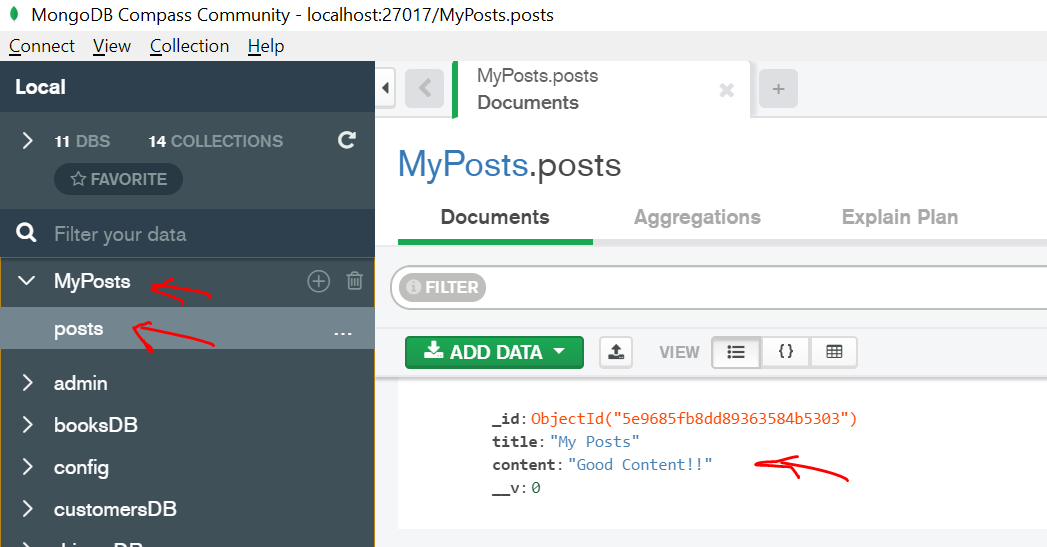
  console.log(post);

  res.status(201).json({

    message: "Post added successfully",

  });

});

1. After the code change the execute the POST API thru POSTMAN tool and observer the data being stored in DB from mongodb compass tool
   1. 
2. If we carefully observe the above the collection “posts” is getting created under “admin” db directly
3. Lets change the connection string as below to store it under “MyPosts” DB
   1. "mongodb://localhost:27017/MyPosts?readPreference=primary&appname=MongoDB%20Compass%20Community&ssl=false"
4. After the above change run the POST API again using POSTMAN tool and observe the new DB being created and “posts” collection under it
   1. 
   2. Observe that collection name is posts. This is because we created a mongoose model called “post” and the plural form of that is the name of the collection
5. Another way to find out whether data was stored in DB or not is thru shell using commands as shown below
   1. 
6. As we are able to store the data in DB , lets fetch it using GET API. Replace get api with below code.

app.get("/api/posts", (req, res, next) => {

  Post.find().then((documents) => {

    res.status(200).json({

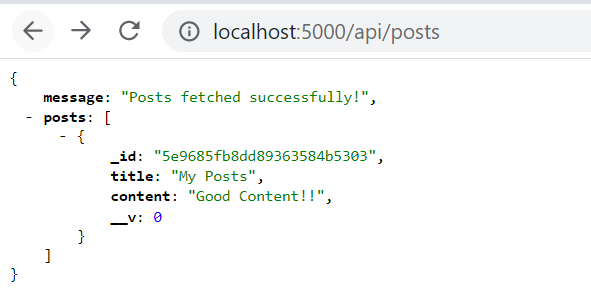
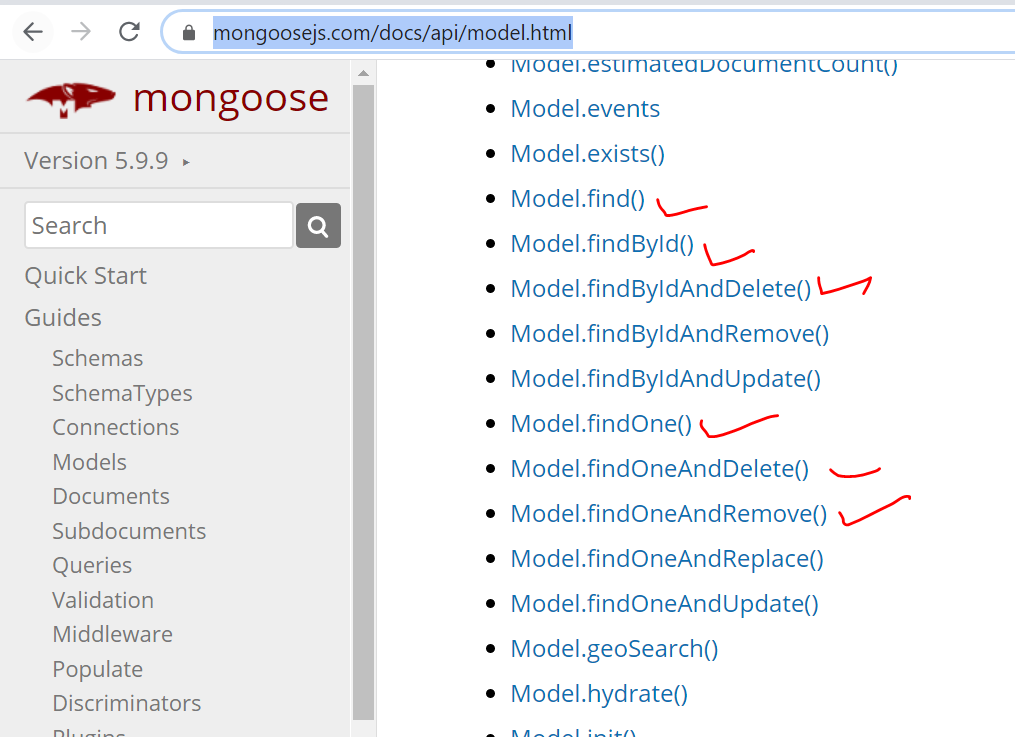
      message: "Posts fetched successfully!",

      posts: documents,

    });

  });

});

1. After the above code change, test the GET API using browser. See that the data is being fetched from database and not from hardcoded posts
   1. 
2. We can find more variations to GET APIs under mongoose documentation
   1. <https://mongoosejs.com/docs/api/model.html>
   2. 
3. Lets create DELETE API with help of below code. Include the below code in app.js

app.delete("/api/posts/:id", (req, res, next) => {

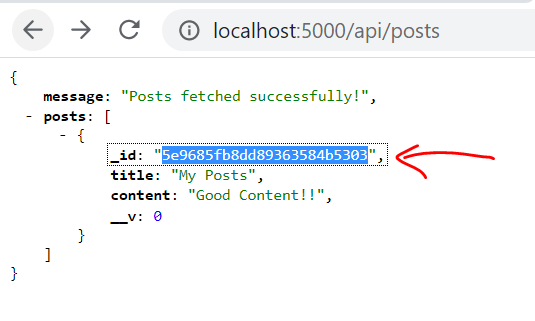
  Post.deleteOne({ \_id: req.params.id }).then((result) => {

    console.log(result);

    res.status(200).json({ message: "Post deleted!" });

  });

});

1. Execute the DELETE API in POSTMAN and observe the changes in GET API
   1. In order to delete a single post we need its ID
      1. 
   2. Pick up the ID from GET API response from above step
   3. Execute DELETE API as shown below from POSTMAN tool. The URL for DELETE API should include the id as shown below. This is as per REST API standards
      1. 