In our work, we need to create a NodeJS server by applying ExpressJS. We import the needed libraries:

ExpressJS: handles server connections.

Mongoose: initializes MongoDB’s database connection.

CORS: allows cross-origin communication in client-server interaction.

Body-parser: handles request body data like JSON.

Before setting up the server, we need to structure Models-Controllers-Routes.

Firstly, we structure a model schema, a template for our database. The fields are defined including name, created\_date and status. Name is required field, which means that if this field is not contained in requests, the error is detected, and the status code is returned 400 (bad request). Both created\_date and status have default value that are automatically generated, even if they are not contained.

Secondly, we create a controller for logically handling the request. The controller manages all the necessary logic. Each function asynchronously handles corresponding request and returns a json format.

Lastly, we generate the routes for defining requests.

After defining model schema, we set up server-side. The Express() is a top-level application to apply all the server processes. Our default port is 3000. After that, the middlewares are structured by using app.use() function. We clarify the request data type using bodyParser.json and bodyParser.urlencoded and enable cross-origin using cors. The connection to the database must be set in the right path. In our case, localhost:27017 is used. Mongoose.connect notifies the output of connection status, which can be successfully if no error occurs, or failed if yes. Finally, we listen to the defined port and activate the server.

Once the server is successfully started, we process the POST request to add some data to the database. We also observe the change happening in the database via Studio3T.