Contact Management App (MyContacts)

Technical Guide

</talentlabs>

Before you start:

Before starting the assignment please spend some time familiarize & understanding the below concepts

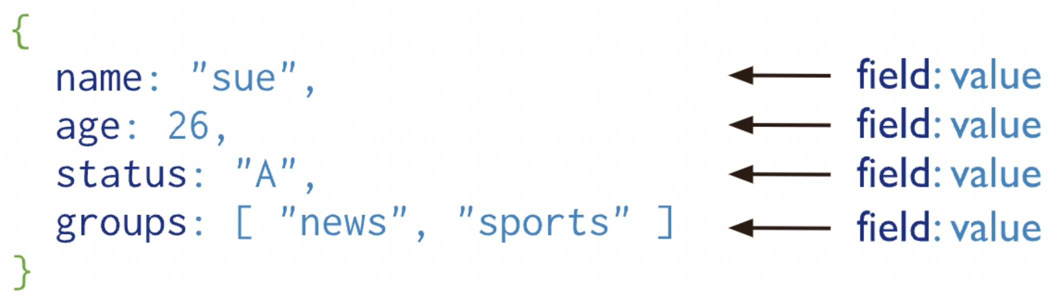
Nodejs (core module), Express framework, MongoDB and Async JavaScript : As these will be used throughout the project.

Tools & Libraries to use:

- **Node.js :** (module: http, path, fs)
  https://nodejs.dev/en/
  https://nodejs.dev/en/api/v18/documentation/
- **Express :** Use to get robust features to build web application using node.js (router, body parser, request params)
  https://expressjs.com/
  https://www.npmjs.com/package/express
- **Bcrypt :** Used for hashing the user password and compare the hash password during login
  https://www.npmjs.com/package/bcrypt
- **Dotenv :** Used to access the environment variables via node.js process core module into our files
  https://www.npmjs.com/package/dotenv
- **Express-async-handler :** Middleware used to handle exceptions inside of async express routes and passing them to your express error handlers
  https://www.npmjs.com/package/express-async-handler
- **Jsonwebtoken :** Used to add authentication to the apis by signing and verifying the access token.
  https://www.npmjs.com/package/jsonwebtoken

- **MongoDB :** It is a document based NO-SQL Database which stores the data in the form of JSON like objects into collections which can be termed as tables in SQL databases. The record in a mongodb is a document which is a data structure composed of field and value. The values of fields may include other documents, arrays and arrays of documents.

Refer to sample as shown below :



```
{
   name: "sue",
   age: 26,
   status: "A",
   groups: [ "news", "sports" ]
}
```

MongoDB supports building applications faster with flexible document data models, scales better with increase in customers request and provides high availability with MongoDB atlas which is hosted in the cloud. Few links related to MongoDb can help you in

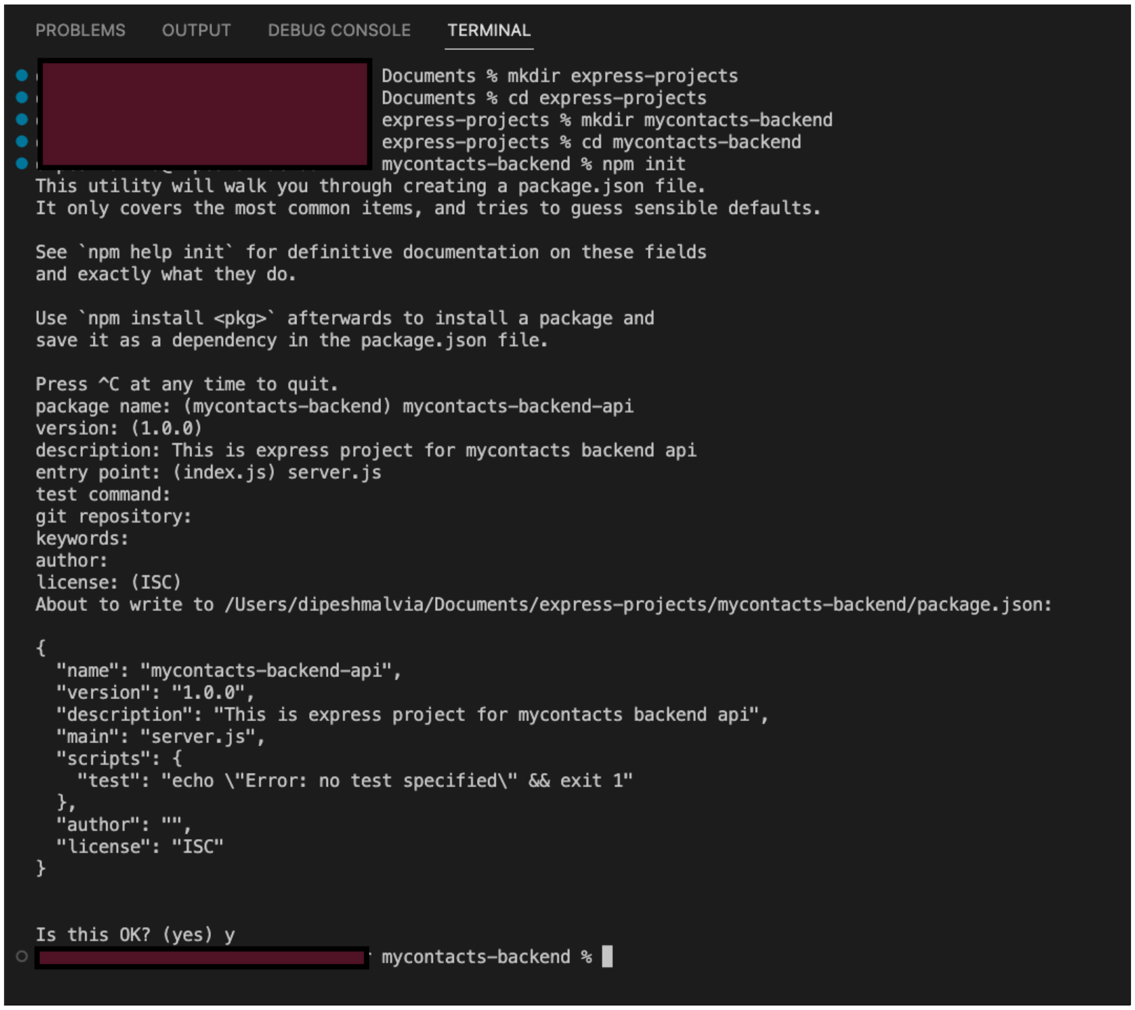
- https://www.mongodb.com/
- https://www.mongodb.com/docs/
- https://www.mongodb.com/docs/manual/tutorial/getting-started/
- **Mongoose :** Used to create object model design schema for our entities like (Contacts, Users) and provide access to methods on the entities to communicate with MongoDB.
  https://www.npmjs.com/package/mongoose
- **Thunder Client** : Used to test our API from VScode. It is a lightweight Rest API client.
  https://www.thunderclient.com/

- **Nodemon** : Used to detect the changes in file and restart the server automatically, especially used for development purposes. https://www.npmjs.com/package/nodemon

Steps to create the application:

- Open in Visual Studio Code (VS Code) terminal and create a directory with express-projects and inside that create mycontacts-backend directory.
- Create a package json file using and provide all the inputs for initial package.json file

npm init



```
[redacted] Documents % mkdir express-projects
           Documents % cd express-projects
           express-projects % mkdir mycontacts-backend
           express-projects % cd mycontacts-backend
           mycontacts-backend % npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (mycontacts-backend) mycontacts-backend-api
version: (1.0.0)
description: This is express project for mycontacts backend api
entry point: (index.js) server.js
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/dipeshmalvia/Documents/express-projects/mycontacts-backend/package.json:

{
  "name": "mycontacts-backend-api",
  "version": "1.0.0",
  "description": "This is express project for mycontacts backend api",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes) y
[redacted] mycontacts-backend % 
```

- Open the folder mycontacts-backend and create .gitignore file and server.js as mentioned in our app entry point.

- Install the required dependencies for building the project

```
npm install express dotenv bycrypt express-async-handler
npm install jsonwebtoken mongoose
npm install --save-dev nodemon
```

- Update the scripts in package.json to run the application

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node server.js",
  "dev": "nodemon server.js"
 }
```

- Add console log in the server.js and run the application as

```
npm run dev
```

- Create the server using express and listen it to port defined in .env or static port

```
const express = require("express");
const dotenv = require("dotenv").config();
const app = express();
const port = process.env.PORT || 5001
app.listen(port, () => {
 console.log(`Server running on port ${port}`);
});
```

- Create routes directory Define api routes for users and contacts

```
app.use("/api/contacts", require("./routes/contactRoutes"));
app.use("/api/users", require("./routes/userRoutes"));
```

- **userRoutes**
  - » POST: /api/users/register (Public route)
  - » POST: /api/users/login (Public route)
  - » GET: /api/users/current (Private route)

```
router.post("/register", registerUser);
router.post("/login", loginUser);
router.post("/current", validateJwtToken, currentUser);
```

- **contactsRoutes**
  - » GET: /api/contacts (Private route)
  - » POST: /api/contacts (Private route)
  - » GET: /api/contacts/:id (Private route)
  - » PUT: /api/contacts/:id (Private route)
  - » DELETE: /api/contacts/:id (Private route)

```
router.route("/").get(getContacts).post(createContact);
router.route("/:id").get(getContact).put(updateContact).delete(deleteCo
ntact);
```

- Create controller directory and define the api logic functions and communicate to data object model build using mongoose
  - userController
    - registerUser

```
const User = require("../models/userModel");
//@desc Register a user
//@route POST /api/users/register
//@access public
const registerUser = asyncHandler(async (req, res) => {
 const { username, email, password } = req.body;
 if (!username || !email || !password) {
   res.status(400);
   throw new Error("All fields are mandatory!");
 }
 const userAvailable = await User.findOne({ email });
 if (userAvailable) {
   res.status(400);
   throw new Error("User already registered!");
 }
```

    - loginUser
    - currentuser

- **contactController**
  - getContacts

```
const Contact = require("../models/contactModel");
//@desc Get all contacts
//@route GET /api/contacts
//@access private
const getContacts = asyncHandler(async (req, res) => {
 const contacts = await Contact.find({ user_id: req.user.id });
 res.status(200).json(contacts);
});
```

  - createContact
  - getContact
  - updateContact
  - deleteContact

- Create data models for userSchema and contactSchema using mongoose
  - **userModel**

Attribute	**Type**	**Required**
username	String	True “Please add the username”
email	String	True “Please add the user email address”
password	String	True “Please add the user password”
timestamp	date	true

  - **contactModel**

Attribute	**Type**	**Required**
User (usermodel)	mongoose.Schema. Types.ObjectId	True
name	String	True “Please add the contact name”
email	String	True “Please add the contact email address”
phone	String	True “Please add the contact phone number”
timetamp	Date	True

- Setup MongoDB Database
  - Create free account - https://www.mongodb.com/
  - Setup the cluster and create cluster user with permission as this is offered by MongoDB Atlas UI and to create the database and collections please follow the guide - https://www.mongodb.com/basics/create-database
  - Once we have the database we can connect it with our local application via a connection string as mentioned in the above step.
  - Create a config directory and have dbConnection.js to establish the connection with Mongodb using Mongoose.

```
const connect = await
mongoose.connect(process.env.CONNECTION_STRING);
  console.log(
    "Database connected: ",
    connect.connection.host,
    connect.connection.name
  );
```

- User operations with Mongoose
  - Register User (Public route function)
    - » Accepts the client request and create a hash password of the raw password provided by the user.
    - » Use the bcrypt package to hash the password as shown

```
const hashedPassword = await bcrypt.hash(password, 10);
```

    - » Create a new user and store it in Users collection via User model

```
const user = await User.create({
username,
  email,
  password: hashedPassword,
 });
```

– Login User (Public route function)

» Accepts the client request with user credentials (email and password) and compare the password with the hashed password stored in the database.

» Retrieve the stored user via email.

```
const user = await User.findOne({ email });
```

» Compare the password using bcrypt and if the password matched generate access token using jsonwebtoken package.

```
if (user && (await bcrypt.compare(password, user.password))) {
  const accessToken = jwt.sign(
   {
    user: {
      username: user.username,
      email: user.email,
      id: user.id,
    },
   },
   process.env.ACCESS_TOKEN_SECERT,
   { expiresIn: "15m" }
  );
  res.status(200).json({ accessToken });
} else {
  res.status(401);
  throw new Error("email or password is not valid");
}
```

» Read more about jsonwebtoken and it's different parts and payload - https://jwt.io/

- Middlewares
  - ErrorHandler
    - » Express handles errors by default but the response of the is in html we can use ErrorHandler as middleware and change the response into json format to handle all the thrown errors in the application.

```
const { constants } = require("../constants");
const errorHandler = (err, req, res, next) => {
 const statusCode = res.statusCode ? res.statusCode : 500;
 switch (statusCode) {
  case constants.VALIDATION_ERROR:
    res.json({
      title: "Validation Failed",
      message: err.message,
      stackTrace: err.stack,
    });
    break;
```

- **ValidateTokenHandler**
  - » The validate token Handler will be used as middleware to validate the access token that is attached in the request as Authentication Bearer token in the request for private routes.

```
if (authHeader && authHeader.startsWith("Bearer")) {
  token = authHeader.split(" ")[1];
  jwt.verify(token, process.env.ACCESS_TOKEN_SECERT, (err, decoded) =>
{
    if (err) {
      res.status(401);
      throw new Error("User is not authorized");
    }
    req.user = decoded.user;
    next();
  });
```

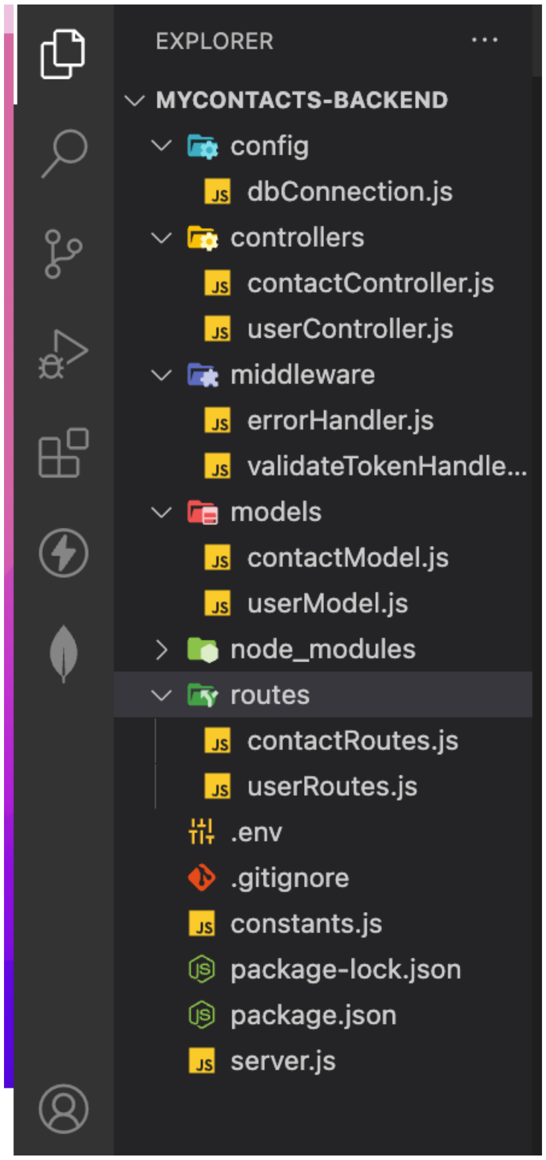
- The Handler can use used in routes to protect them for unauthorized access of contact crud operations

```
const validateToken = require("../middleware/validateTokenHandler");
router.use(validateToken);
router.route("/").get(getContacts).post(createContact);
router.route("/:id").get(getContact).put(updateContact).delete(deleteCo
ntact);
```

- The Handler will also be used for the user route to get the information of the current logged in user

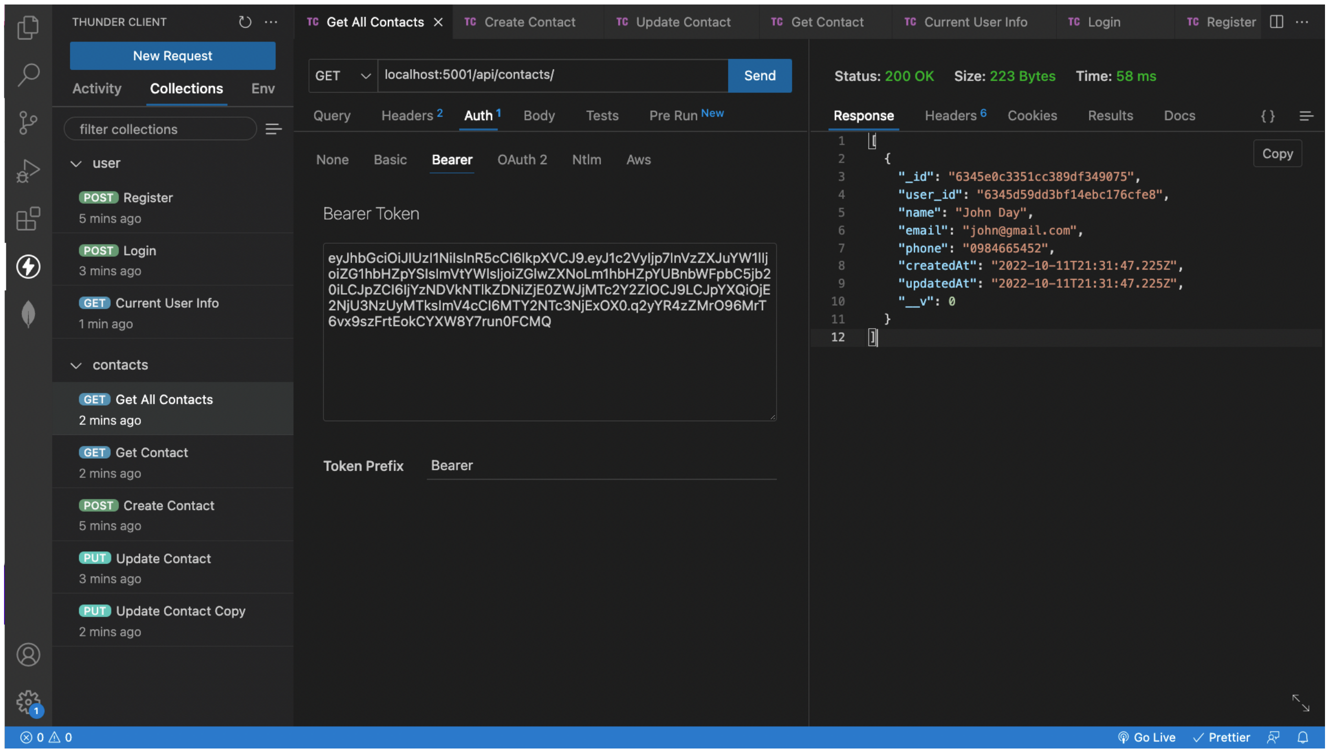
```
const validateToken = require("../middleware/validateTokenHandler");

router.get("/current", validateToken, currentUser);
```

- **Folder Structure – Should look like this**



```
EXPLORER                    ...
v MYCONTACTS-BACKEND
  v config
      dbConnection.js
  v controllers
      contactController.js
      userController.js
  v middleware
      errorHandler.js
      validateTokenHandle...
  v models
      contactModel.js
      userModel.js
  > node_modules
  v routes
      contactRoutes.js
      userRoutes.js
    .env
    .gitignore
    constants.js
    package-lock.json
    package.json
    server.js
```

- **Thunder client**
  - API testing can be done using the ThunderClient as an extension in the Visual Studio code



```
GET   localhost:5001/api/contacts/   Send
```

Bearer Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7InVzZXJuYW1lIj
oiZG1hbHZpYSIsImVtYWlsIjoiZGlwZXNoLm1hbHZpYUBnbWFpbC5jb2
0iLCJpZCI6IjYzNDVkNTlkZDNiZjE0ZWJjMTc2Y2ZlOCJ9LCJpYXQiOjE
2NjU3NzUyMTksImV4cCI6MTY2NTc3NjExOX0.q2yYR4zZMrO96MrT
6vx9szFrtEokCYXW8Y7run0FCMQ
```

Token Prefix: Bearer

Status: 200 OK   Size: 223 Bytes   Time: 58 ms

```
[
  {
    "_id": "6345e0c3351cc389df349075",
    "user_id": "6345d59dd3bf14ebc176cfe8",
    "name": "John Day",
    "email": "john@gmail.com",
    "phone": "0984665452",
    "createdAt": "2022-10-11T21:31:47.225Z",
    "updatedAt": "2022-10-11T21:31:47.225Z",
    "__v": 0
  }
]
```

Please refer to the video for more details.

Thank you.

All The Best :)