Get started

1. Open vscode, from terminal create a directory server and move into it npm init
2. Create .gitignore and name all files that we don't want to be part of git repository (/node_modules and .env)
3. Install the packages within server dirctory, type on terminal npm install express nodemon dotenv express-async-handler mongoose bcrypt jsonwebtoken and add the following in package.json from now on we start the server by typing npm start on the terminal

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

4. Create .env file to store sensitive data, store the port value, later on we add database infos

```
PORT=5001
```

5. Create server.js file and use express and connect to the server

```
const express = require("express");
const dotenv = require("dotenv").config();
const errorHandler = require("./middleware/errorHandler"); //when creating errorHandler
const connectDb = require("./config/dbConnection"); //when connecting DB


// Create an Express application
const app = express();
const port = process.env.PORT || 5000;
connectDb()

//to access the body being parsed req.body (for HTTP POST later)
app.use(express.json());

// use the app as middleware for routes
app.use("/api/contacts", require("./routes/contact"));

// use the app as middleware for error handler
app.use(errorHandler)

// Start the server and listen on the specified port
app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

6. In server directory create constants.js file containing standard HTTP status codes, which are used to represent different types of errors

```
exports.constants = {
  VALIDATION_ERROR: 400,
  UNAUTHORIZED: 401,
  FORBIDDEN: 403,
  NOT_FOUND: 404,
  SERVER_ERROR: 500,
};
```

Then create a middleware directory and within it create errorHandler.js file, and import the constants file to use in errorHandler which uses the defined statusCode or sets it to 500 in order to use it in switch statement

```
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;

    case constants.NOT_FOUND:
      res.json({
        title: "Not Found",
        message: err.message,
        stackTrace: err.stack,
      });
      break;

    case constants.UNAUTHORIZED:
      res.json({
        title: "Unauthorized",
        message: err.message,
        stackTrace: err.stack,
      });
      break;

    case constants.FORBIDDEN:
      res.json({
        title: "Forbidden",
        message: err.message,
        stackTrace: err.stack,
      });
      break;

    case constants.SERVER_ERROR:
      res.json({
        title: "Server Error",
        message: err.message,
        stackTrace: err.stack,
      });
      break;

    default:
      console.log("No Error, All good!")
      break;
  }
};

module.exports = errorHandler;
```

7. Create a cluster from mongoose website, choose connect MongoDB compass, then from VSCode mongoose extension, create new connection and paste the connection string

Again choose connect MongoDB Drivers and save the connection string in the .env file as variable CONNECTION_STRING (replace password)

In order to connect the app to our MongoDB database, create config directory and within it create a file dbConnections.js and import the library and write the connection function.

```
// Import the Mongoose library for working with MongoDB
const mongoose = require("mongoose");

// Function to establish a connection to the MongoDB database
const connectDb = async () => {
  try {
    // Attempt to connect to the database using the connection string
    const connect = await mongoose.connect(process.env.CONNECTION_STRING);

    // Log a success message with the host and database name
    console.log(
      "Database connected: ",
      connect.connection.host, // Host of the connected database
      connect.connection.name // Name of the connected database
    );
  } catch (err) {
    // Log any errors that occur during the connection attempt
    console.log(err);

    // Exit the process with a failure status code (1) if the connection fails
    process.exit(1);
  }
};

// Export the function to be used in other parts of the application
module.exports = connectDb;
```

8. Write the schema for the database, create models directory and within it create contactModel.js file

```
// Import Mongoose library to define schema and interact with MongoDB
const mongoose = require("mongoose");

// Define the schema for the Contact model
const contactSchema = mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, "Please add the contact name"],
    },
    email: {
      type: String,
      required: [true, "Please add the contact email address"],
    },
    phone: {
      type: String,
      required: [true, "Please add the contact phone number"],
    },
  },
  {
    // Automatically add createdAt and updatedAt timestamps
    timestamps: true,
  }
);

// Export the Contact model based on the defined schema
module.exports = mongoose.model("Contact", contactSchema);
```

9. Create the controllers directory and within it create contactController.js file to contain the logic for HTTP request methods, and export and which will connect to our database

```
// with asyncHandler we don't have to write try catch blocks, whenever an exception is
called, it passes it to the error handler
const Contact = require("../models/contactModel")
const asyncHandler = require("express-async-handler");

// @desc Get all contacts
// @route GET /api/contacts
// @access public
const getContacts = asyncHandler(async (req, res) => {
  const contacts = await Contact.find();
  res.status(200).json(contacts);
});

// @desc Get contact
// @route GET /api/contacts/:id
// @access public
const getContact = asyncHandler(async (req, res) => {
  const contact = await Contact.findById(req.params.id); // Find contact by ID
  if (!contact) {
    res.status(404); // Set status to 404 if not found
    throw new Error("Contact not found");
  }
  res.status(200).json(contact);
});

// @desc Create new contact
// @route POST /api/contacts
// @access Public
const createContact = asyncHandler(async (req, res) => {
  console.log("The request body is :", req.body);
  const { name, email, phone } = req.body; // Destructure the request body
  if (!name || !email || !phone) {
    res.status(400);
    throw new Error("All fields are mandatory !");
  }
  const contact = await Contact.create({ name, email, phone }); // Create new contact
  res.status(201).json(contact);
});

// @desc Update contact
// @route PUT /api/contacts/:id
// @access public
const updateContact = asyncHandler(async (req, res) => {
  const contact = await Contact.findById(req.params.id);
  if (!contact) {
    res.status(404);
    throw new Error("Contact not found");
  }
  const updatedContact = await Contact.findByIdAndUpdate(
    req.params.id,
    req.body, // Update with request body
    { new: true } // Return the updated document
  );
  res.status(200).json(updatedContact);
});
```

```
// @desc Delete contact
// @route DELETE /api/contacts/:id
// @access public
const deleteContact = asyncHandler(async (req, res) => {
  const contact = await Contact.findById(req.params.id);
  if (!contact) {
    res.status(404);
    throw new Error("Contact not found");
  }

  await Contact.deleteOne(contact);
  res.status(200).json(contact);
});

// export the functions to be used in contactRoutes
module.exports = {getContacts, getContact, createContact,updateContact,deleteContact};
```

10. Create route dir and within it create contactRoute.js file and create all routes

```
// Import the Express library
const express = require("express");

// Create a new router instance
const router = express.Router();

// Import controllers functions from the contactController file for handling the routes
const {
  getContacts, getContact, createContact, updateContact,  deleteContact, } =
require("../controllers/contact");

// combine multiple HTTP methods for the same route
router.route("/").get(getContacts).post(createContact);
router.route("/:id").get(getContact).put(updateContact).delete(deleteContact);

// Export the router to use it in other parts of the application
module.exports = router;
```

11. To make authentication, create userModel.js in models to make the schema of creating users

```
const mongoose = require("mongoose");
const userSchema = mongoose.Schema(
  {
    username: {
      type: String,
      required: [true, "Please add the user name"],
    },
    email: {
      type: String,
      required: [true, "Please add the user email address"],
      unique: [true, "Email address already taken"],
    },
    password: {
      type: String,
      required: [true, "Please add the user password"],
    },
  },
  {
    timestamps: true,
  }
);

module.exports = mongoose.model("User", userSchema);
```

To get access into private routes, we should use the returned access token (pass it in Auth Bearer or as Headers value) so we need a middleware to validate this token. In middleware create validateTokenHandler.js And add it to the private method in userRoutes.js and import it as const.

```
const asyncHandler = require("express-async-handler");
const jwt = require("jsonwebtoken");

const validateToken = asyncHandler(async (req, res, next) => {
  let token;
  let authHeader = req.headers.Authorization || req.headers.authorization;
  if (authHeader && authHeader.startsWith("Bearer")) {
    token = authHeader.split(" ")[1];
    jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, decoded) => {
      if (err) {
        res.status(401);
        throw new Error("User is not authorized");
      }
        req.user = decoded.user;
        next();
    });
      if (!token) {
          res.status(401);
          throw new Error("User is not authorized or token is missing");
      }
  }
});

module.exports = validateToken;
```

Create userController.js in controllers and at the end export the created methods and add ACCESS_TOKEN_SECRET in .env file,

```
const asyncHandler = require("express-async-handler");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
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!");
  }

  //Hash password
    const hashedPassword = await bcrypt.hash(password, 10);
    console.log("Hashed Password: ", hashedPassword);
    const user = await User.create({
     username,
     email,
     password: hashedPassword,
    });

    console.log(`User created ${user}`);
    if (user) {
      res.status(201).json({ _id: user.id, email: user.email });
    } else {
      res.status(400);
      throw new Error("User data us not valid");
    }
    res.json({ message: "Register the user" });
});

// @desc Login user
// @route POST /api/users/login
// @access public
const loginUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body;
  if (!email || !password) {
    res.status(400);
    throw new Error("All fields are mandatory!");
  }
  const user = await User.findOne({ email });
  //compare password with hashedPassword
  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_SECRET,
      { expiresIn: "1m" }
    );
    res.status(200).json({ accessToken });
  } else {
    res.status(401);
    throw new Error("email or password is not valid");
  }
});

// @desc Current user info
// @route POST /api/users/current
// @access private
const currentUser = asyncHandler(async (req, res) => {
  res.json(req.user);
});

module.exports = { registerUser, loginUser, currentUser };
```

We have to provide some points for users to register and login. First in index.js provide a point for users

```
app.use("/api/users", require("./route/userRoute"));
```

Then we create userRoute.js file

```
const express = require("express");
const {
  registerUser,
  currentUser,
  loginUser,
} = require("../controllers/userController");
const validateToken = require("../middleware/validateTokenHandler");
const router = express.Router();

router.post("/register", registerUser);
router.post("/login", loginUser);
router.get("/current", validateToken, currentUser);

module.exports = router;
```

We want to protect the content related to each user, so only the login user can create, delete, update and read their content, for that we need to associate the content with the user id who created that content, for that we add to contactModel.js

```
    user_id: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "User",
    },
```

And we will use the validateToken as a middleware in all routes that we want to make private

```
const validateToken = require("../middleware/validateTokenHandler");
router.use(validateToken);
// Define a route to get all contacts
router.route("/").get(getContacts).post(createContact);
router.route("/:id").get(getContact).put(updateContact).delete(deleteContact);
```

this way is used if we want to make all our routs private, otherwise like we did in /current route
And make @access public as @access private

In order to GET all the contact related to the loggedin user, for that in contactController.js we find with user_id which was created in contactModel.js

```
  const contacts = await Contact.find({user_id: req.user.id});
```

In order to POST a contact for a loggedin user, we create with user_id

```
  const contact = await Contact.create({ name, email, phone, user_id: req.user.id });
```

In order to PUT a contact with a specific contact_id, after checking if the contact exists, we check if this contact is related to the user, if so it's not then the code will continue to delete the contact otherwise it throw an error.

```
  if (contact.user_id.toString() !== req.user.id) {
    res.status(403);
    throw new Error("User doesn't have permission to update other user contacts");
  }
```

We do the same for DELETE a contact,