1. Initiate an express app
2. The app’s starting point is server.js
3. Server.js contains the port number and environment variables
4. The .env file contains stuff like port number etc
5. app is the name of the express app we chose
6. app.use() is used to use any kind of middleware we’re going to use in the express app

```
7. app.use("/api/contacts",require("./routes/contacts-route"))
```

8. The above line means that the URL which begins with /api/contacts is definitely sent to the middleware (routing middleware) which is at the location specified
9. The routing middleware and the contacts controller perform the tasks together. Considering them as a single entity makes is easier to understand how the app runs

The routing middlware has

```
router.route("/").get(getAllContacts).post(createNewContact)


router.route("/:id").get(getIndividualContact).put(updateContact)
.delete(deleteContact)
```

10. Which means that whichever URL has the above specified CRUD method and parameter will perform the specified function inside the contacts-controller

The contacts controller exports small chunks of code called modules and each module is named according to the function they perform. Ex:

```
const getAllContacts = asyncHandler(
async(req,res)=>{

    // res.status(200).json({message:"Get all contacts"})
    const AllContacts =await ContactModel.find();
    res.status(200).json(AllContacts);
})
const createNewContact =asyncHandler( async(req,res)=>{
    console.log("The data recieved in request is ", req.body)
```

11. Now coming to the error-handling, if we notice in the server.js, we see that the error handling middleware is added after the routing middleware, which means, while routing, if any request is responded with a particular statuscode like 400 or 404, an appropriate message is sent to the console and as a response
12. Then comes the real CRUD operations using MongoDB
13. First a mongodb database is created and connected using the mongodb extension in VSCode
14. Mongoose is installed

Using mongoose, the db we created is connected:

```
const mongoose = require("mongoose")


const connectDB = async()=>{
    try{
        const connect =
mongoose.connect("mongodb://localhost:27017/MyContacts");
        console.log("Connected to the database")
    }catch(err){
        console.log(err);
        process.exit(1);
    }
};
```

15. We need to specify the structure of each entity we are going to store in our database, it is called as Schema
16. We again use mongoose to create and export Schema. We can find it in the contact-model file
17. Using the same Schema, we perform the CRUD operations for different types of CRUD requests specified in the routing middleware and the contact-controller

Ex:
```
const createNewContact =asyncHandler( async(req,res)=>{

    console.log("The data recieved in request is ", req.body)
    const {name, email, phone} = req.body;
    if(!name || !email || !phone){
        res.status(400)
        throw new Error("All fields are mandatory")
    }
    const contact = await ContactModel.create({
        name,
        email,
        phone
    })
    res.status(201).json({contact})
});
```

18. Now our app performs CRUD operations in the database

Users

1. In server.js, define a route which starts with /api/users and attach a routing middleware which takes care of the remaining middlewares and operations. The routing middleware is situated at the routes folder:

```
app.use("/api/users",require("./routes/users-route"))
```

2. similar to the contacts, a users-controller.js and this users-route middleware work as one entity and to take care of the users CRUD operations

3. functions to create, update and delete users are exported from the controller and used in the users-route middleware

4. A MongoDB schema is needed to store the users, it is done using mongoose and exporting the Schema and making use of it in the users-route and user-controller middlewares

Password Hashing:

5. bcrypt is used for password hashing

6. only hashed passwords need to be stored in the database.

7. it is done using bcrypt.hash(passwordString, 10)

Logging in and auth token:

8. Auth token is generated using JWT – JsonWebToken

9. An auth token is generated using jwt module and how long that authtoken is valid can be specified by the developer.
```
if (user && 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: "20m" }
    );
```

10. this auth-token is like a green-signal for the user where he can access what he is allowed to until it stays green

11. At this point, we have achieved acquiring an auth token when a user logs in.

12. Now we need to validate the auth-token for different routes

13. validateToken middleware:

What does it do?

- When a route requires the validateToken to be satisfied, it must have a header which contains an auth-token.
- validateToken() will extract the auth-token from the header and verifies it with the authtoken_secret mentioned in the .env file and decodes it into username, email and id.
- It contains only those as in the login step, where the authtoken is generated, we have only given username email and id as the payload. Giving password in the payload is like erripookthanam and we are not in for it.
- So now we have the info decoded from the authtoken in the validateToken step.

14. Now after the authoken is validated, it is routed again to the currentUser route which required this validation

User-specific routes:

- Now the goal is to let the users access, view, update and delete ONLY the contacts they have created.
- This can be done by adding a property in the contacts schema where it also stores the creator's id.
- This id is retrieved in the createContact step and the id is extracted from the authtoken generated when a user logs in . and it is extracted from the decoded info at the validateToken step and is used while creating the new contact.
- That'all.
- In the update and delete functions, the creator_id can directly be extracted from the contact targeted. This id is compared with the id of the user logged in and permission is given accordingly