CS186
BACKEND ENGINEERING
PROJECT

Presented By:
Kartik Arora

Introduction

In this presentation I’m going to present my Phonebook project project for my Backend Implementation skills. I’ve made RESTful APIs and defined endpoints for the app where required along with a frontend in react along with error handling at both ends of the application to provide user with a feedback about when an error has occured and why exactly, providing apt feedbacks and allowing for a decent functionality with a simplistic design making it functional at even low end devices.

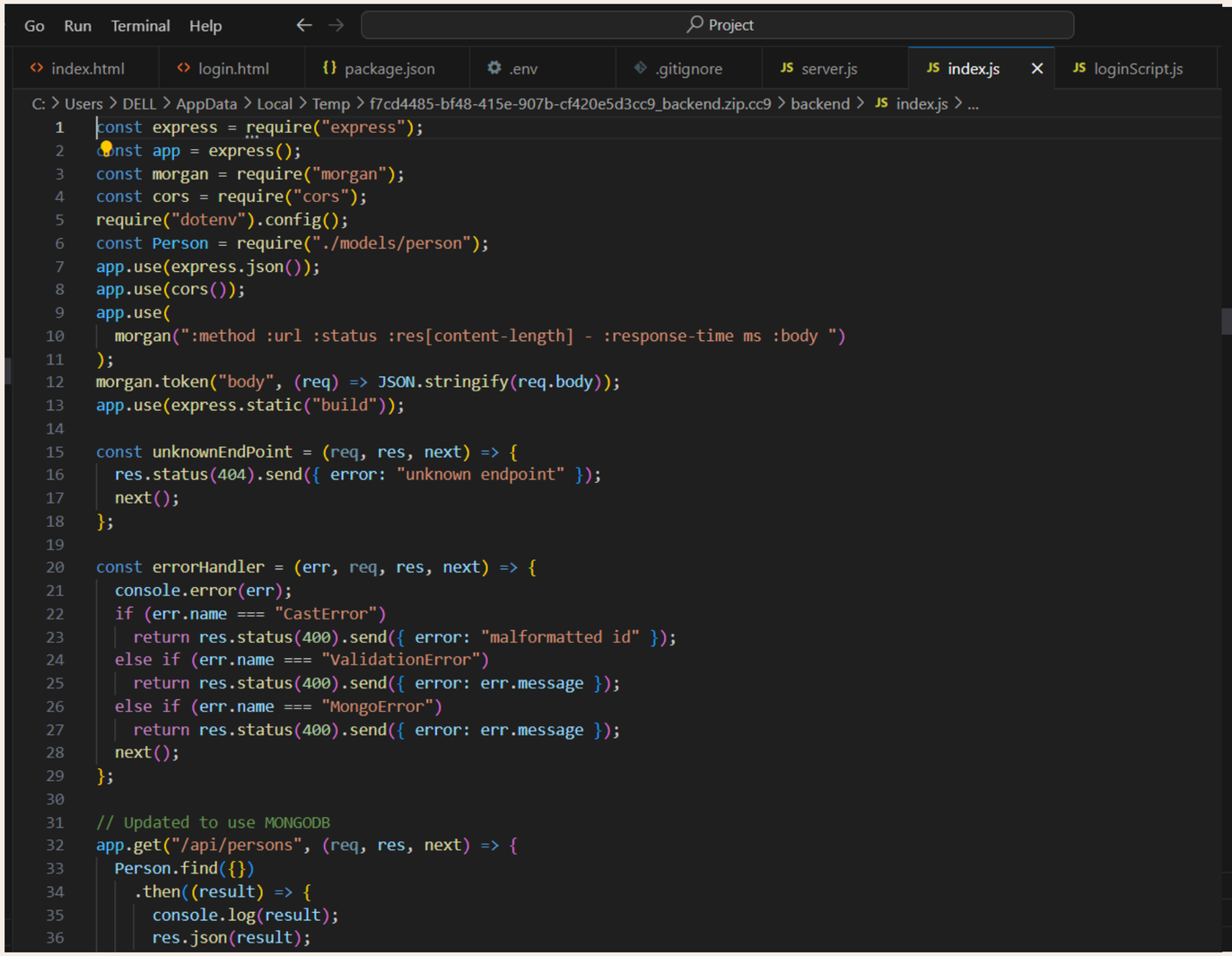
Framework Utilized

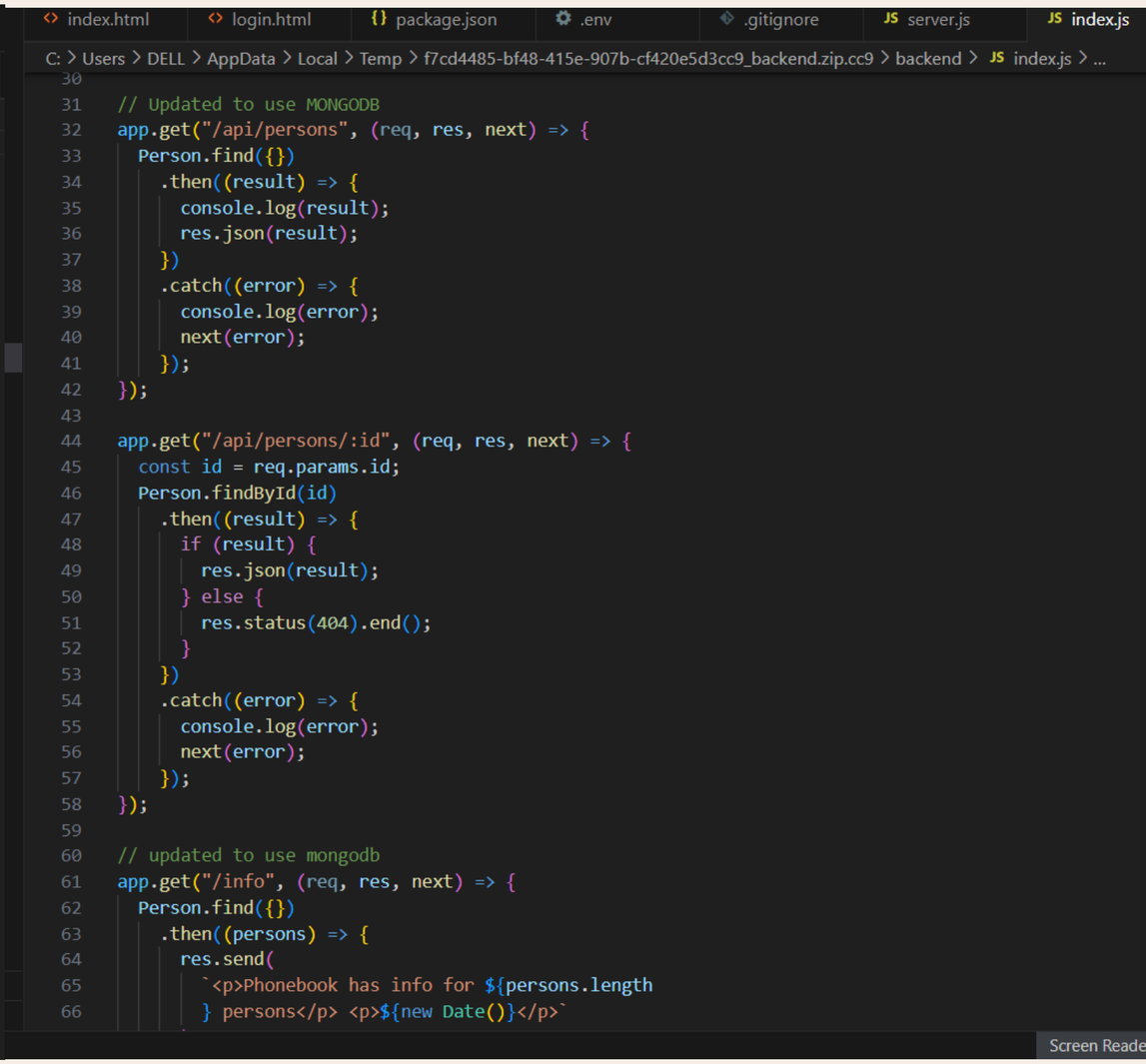
Express.js is a lightweight Node.js web framework simplifying web app/API development. Its minimalist structure aids in handling HTTP requests, middleware integration, and routing. With a flexible and robust API, it's favored for scalable apps, offering simplicity and extensive community support.

Database Utilized

Mongoose is a popular Node.js library simplifying MongoDB interactions. It provides a straightforward way to model data with schemas, aiding in database operations like querying, validation, and more. As an Object Data Modeling (ODM) tool, Mongoose streamlines code for MongoDB, ensuring easy data manipulation and schema-based structuring, beneficial for developers in building and managing databases efficiently.

Code Screenshots





HTTP REQUESTS

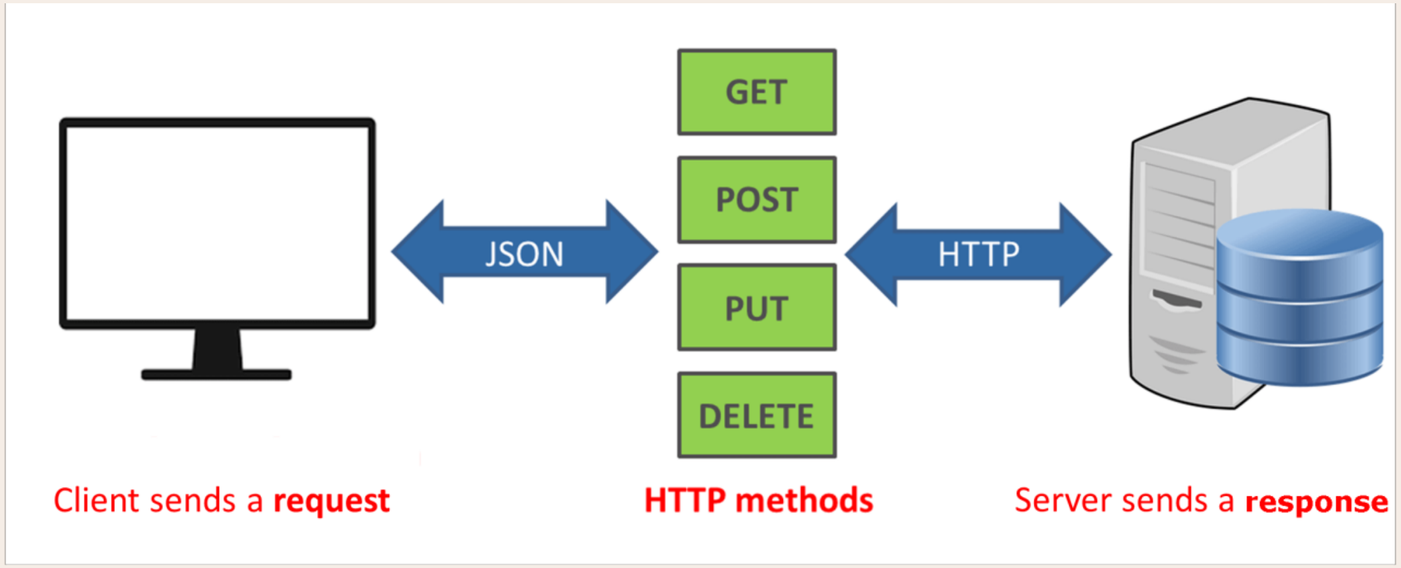
POST

POST

PUT

DELETE

GET



API’s Utilized

An API (Application Programming Interface) acts as a bridge allowing different software applications to communicate and interact. It defines the rules for how applications can request and share data or functionalities. APIs enable developers to access specific features or data from external sources, facilitating seamless integration between diverse systems, fostering innovation, and enhancing software capabilities.

APIs serve as digital messengers enabling software programs to talk to each other. They define protocols, allowing apps to request and exchange data, perform tasks, or access services. By providing a standardized way for systems to connect and share information, APIs streamline development, foster collaboration between applications, and empower developers to create versatile and interconnected software solutions.

01.

Retrieval

GET /api/persons:
Retrieves a list of all persons in the database.

GET /api/persons/:id:
Retrieves details of a specific person based on the provided ID.

GET /info:
Returns information about the number of persons in the phonebook.

02.

Deletion

DELETE /api/persons/:id:

Deletes a person from the database based on the provided ID.

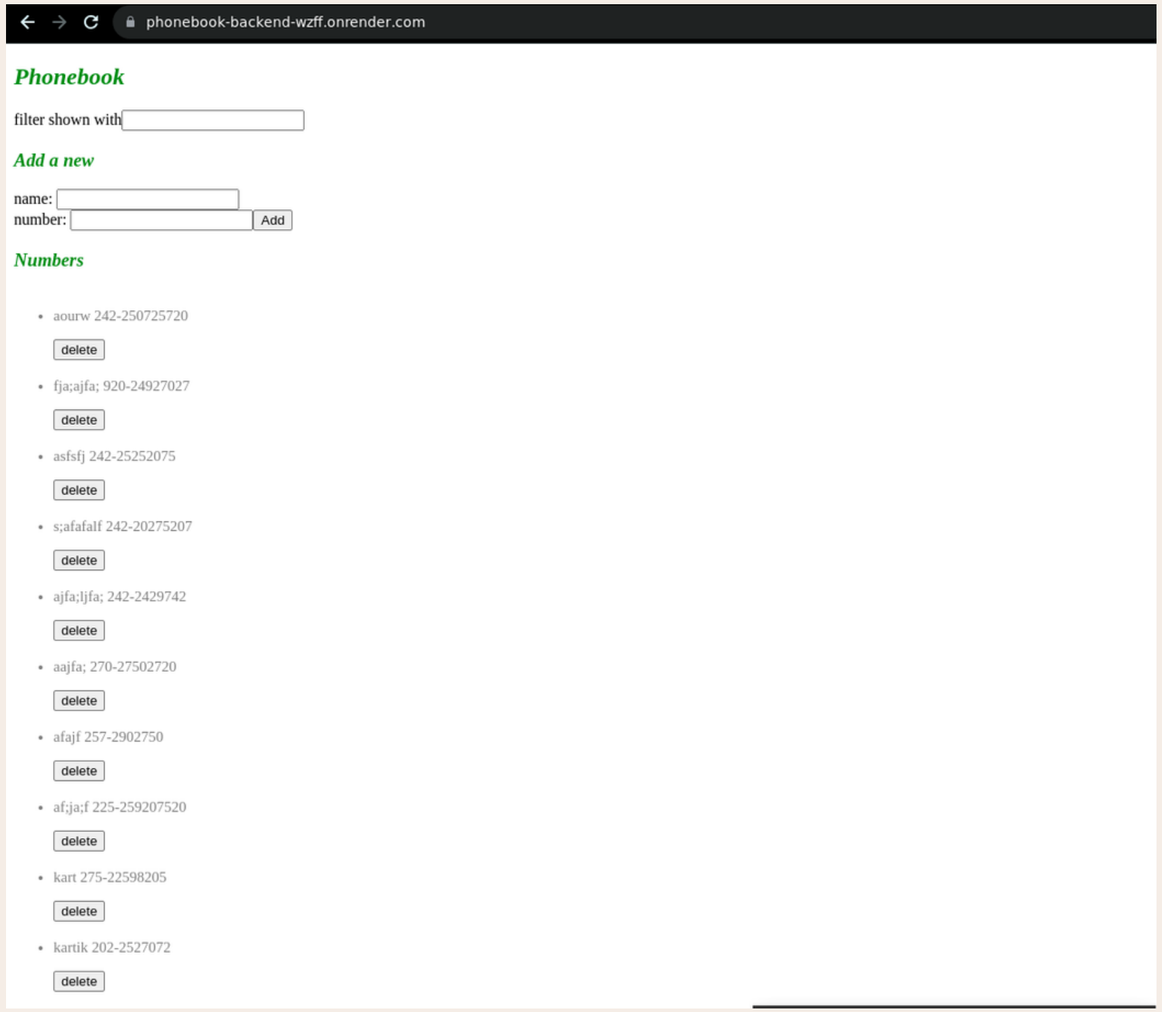
03.

Creation/Updation

POST /api/persons:
Adds a new person to the database.

PUT /api/persons/:id:
Updates details of a person based on the provided ID.

Frontend



As it’s a backend Centric project, I’ve made a simple frontend design to illustrate the basic design of it all, the frontend consists of a search bar, a form for entering a new phone record and has error handling for the same, such as the name should be minimum of three letters and the phone number should follow a particular pattern, 2-3 digits before ‘-’ and 6 or up digits after the ‘-’

Code Explanation

Request Handlers

```
> app.get("/api/persons", (req, res, next) => {···
  });

> app.get("/api/persons/:id", (req, res, next) => {···
  });

  // updated to use mongodb
> app.get("/info", (req, res, next) => {···
  });

  // updated to use mongodb
> app.delete("/api/persons/:id", (req, res, next) => {···
  });

  // updated to use mongodb
> app.post("/api/persons", (req, res, next) => {···
  });

  // updated to use mongodb
> app.put("/api/persons/:id", (req, res, next) => {···
```

I’ve made these RESTful APIs that allows for updation, creation and deletion of the phonebook entries made using HTTP requests

Middlewares

Here, I've used the express.json(), cors, morgan and the build file from the frontend to integrate into the backend, express.json is used to convert to json and morgan is used for giving verbose details and cors is used for allowing access to cross origin sites and to secure site.

```
app.use(express.json());
app.use(cors());
app.use(
  morgan(":method :url :status :res[content-length] - :response-time ms :body ")
);
morgan.token("body", (req) => JSON.stringify(req.body));
app.use(express.static("build"));
```

```
app.use(unknownEndPoint);
app.use(errorHandler);
```

Here, I've made my custom build error handling middlewares for recoding any error occured during the app execution.,
Which helps a lot in getting to know about the exact error

Error Handling

This handles the unknown endpoints for unexpected routes that otherwise lead to a blank page, making the debugging easier.

```
const unknownEndPoint = (req, res, next) => {
  res.status(404).send({ error: "unknown endpoint" });
  next();
};
```

```
const errorHandler = (err, req, res, next) => {
  console.error(err);
  if (err.name === "CastError")
    return res.status(400).send({ error: "malformatted id" });
  else if (err.name === "ValidationError")
    return res.status(400).send({ error: err.message });
  else if (err.name === "MongoError")
    return res.status(400).send({ error: err.message });
  next();
};
```

This handles Casting, data formatting, Validation and Mongoose errors.

Database

Mongoose Schema

```
const mongoose = require('mongoose');
mongoose.set('strictQuery', false);
require('dotenv').config();

const url = process.env.MONGODB_URI;

mongoose.connect(url).then(() => {
    console.log('connected');
}).catch(err => {
    console.log(err);
})

const personSchema = new mongoose.Schema({
    name: {
        type: String,
        minLength: 3,
        required: true,
    },
    number: {
        type: String,
        validate: {
            validator: function(v) {
                return (/^\d{2,3}-\d{6,}$/).test(v)
            }
        },
        required: true,
    },
});

personSchema.set('toJSON', {
    transform: (document, returnedObject) => {
        returnedObject.id = returnedObject._id.toString()
        delete returnedObject._id
        delete returnedObject.__v
    }
})

module.exports = mongoose.model('Person', personSchema);
```

Here, I've used the mongoose library and dotenv library to handle the schema design and defined a custom function to transform the returned object and add another parameter by the name of id and added validation to the schema for minimum name length to be 3 and the phone number to be 2-3 digits before '-' and 6 or more digits after the '-'.

Dotenv

backend > .env.example

```
1    MONGODB_URI=<YOUR_TOKEN>
```

I’ve used the dotenv library to safely store and use my secret keys and use them in my application as and where required (blog model) and mongo.js file which I used to test and save the mongodb model and

Thank You