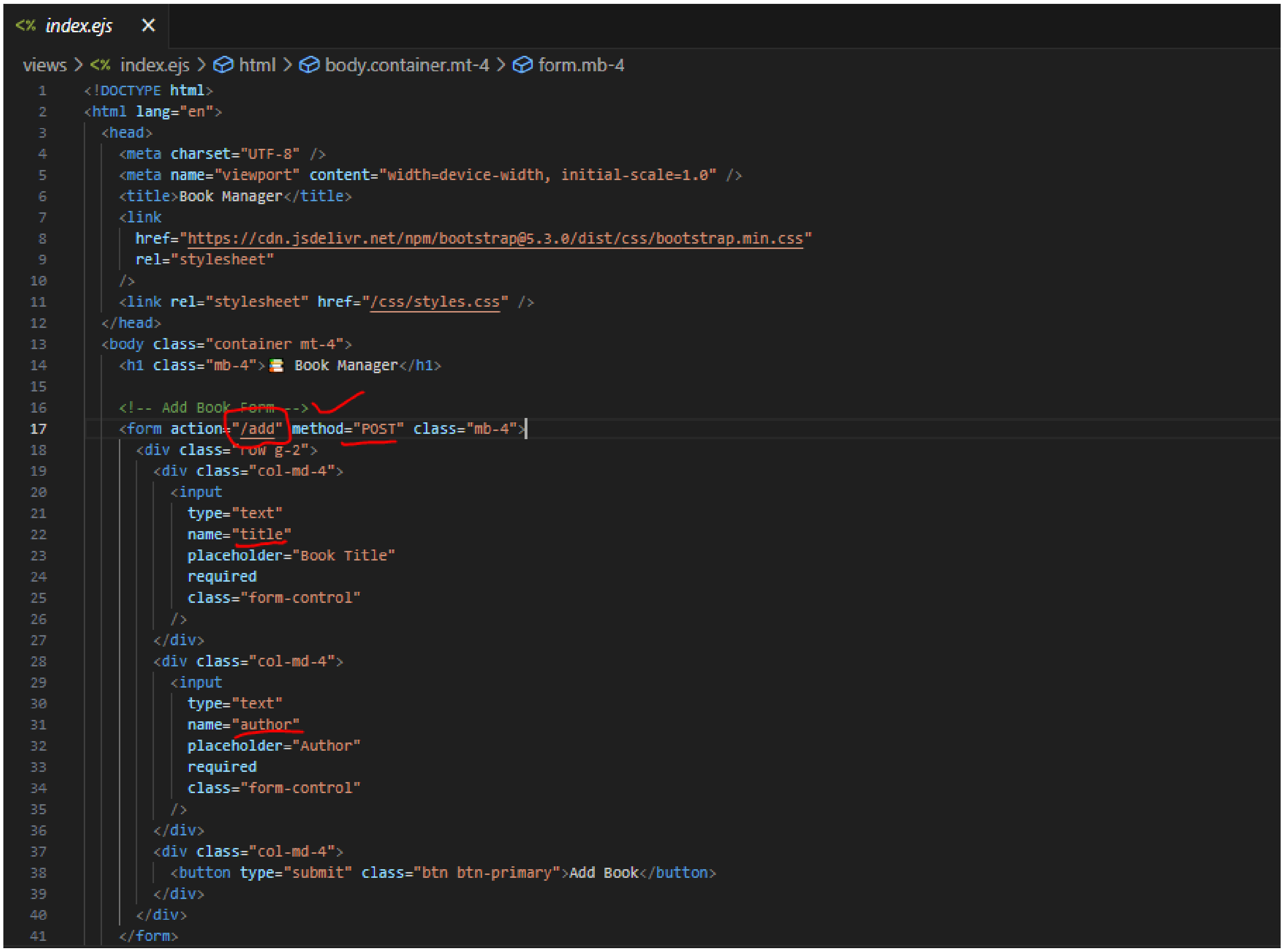
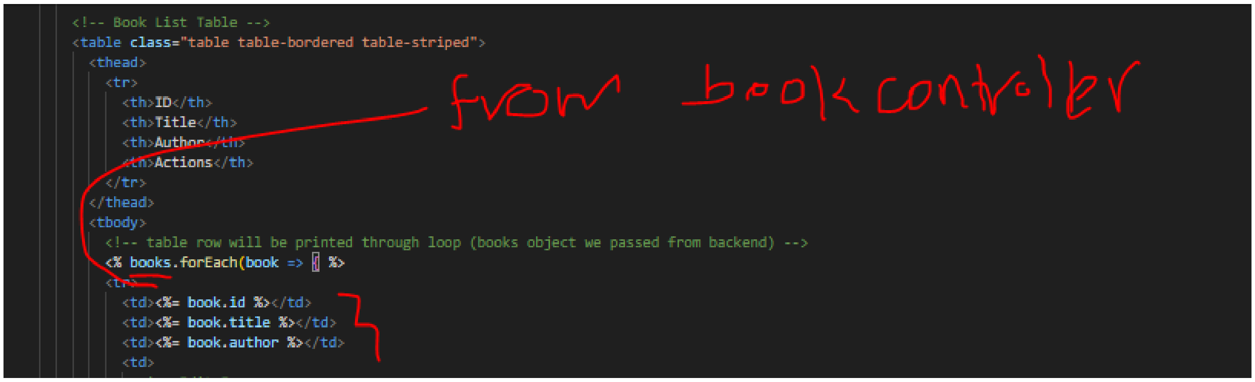
# Books CRUD using Express MVC Folder Structure Project

## 1: Frontend

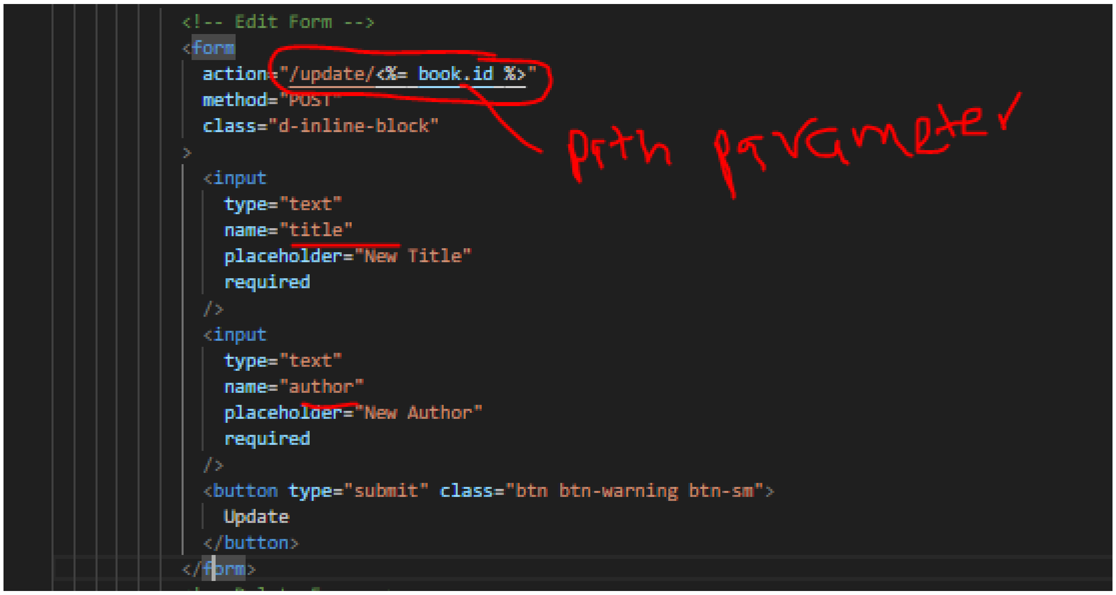


```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Book Manager</title>
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
      rel="stylesheet"
    />
    <link rel="stylesheet" href="/css/styles.css" />
  </head>
  <body class="container mt-4">
    <h1 class="mb-4">📚 Book Manager</h1>

    <!-- Add Book Form -->
    <form action="/add" method="POST" class="mb-4">
      <div class="row g-2">
        <div class="col-md-4">
          <input
            type="text"
            name="title"
            placeholder="Book Title"
            required
            class="form-control"
          />
        </div>
        <div class="col-md-4">
          <input
            type="text"
            name="author"
            placeholder="Author"
            required
            class="form-control"
          />
        </div>
        <div class="col-md-4">
          <button type="submit" class="btn btn-primary">Add Book</button>
        </div>
      </div>
    </form>
```



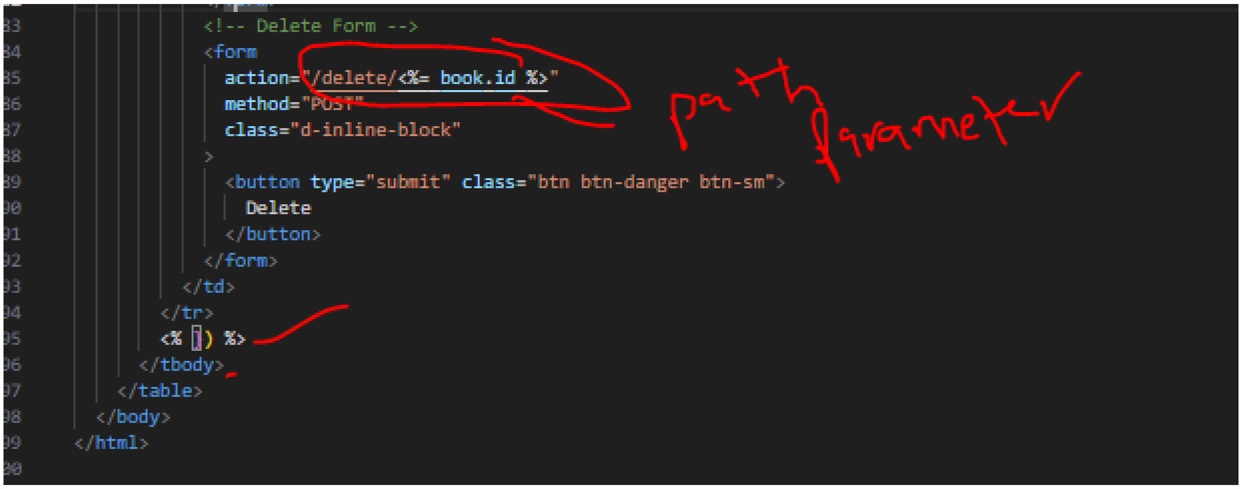
```
<!-- Book List Table -->
<table class="table table-bordered table-striped">
  <thead>
    <tr>
      <th>ID</th>
      <th>Title</th>
      <th>Author</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    <!-- table row will be printed through loop (books object we passed from backend) -->
    <% books.forEach(book => { %>
    <tr>
      <td><%= book.id %></td>
      <td><%= book.title %></td>
      <td><%= book.author %></td>
      <td>
```

from bookcontroler



```
<!-- Edit Form -->
<form
  action="/update/<%= book.id %>"
  method="POST"
  class="d-inline-block"
>
  <input
    type="text"
    name="title"
    placeholder="New Title"
    required
  />
  <input
    type="text"
    name="author"
    placeholder="New Author"
    required
  />
  <button type="submit" class="btn btn-warning btn-sm">
    Update
  </button>
</form>
```

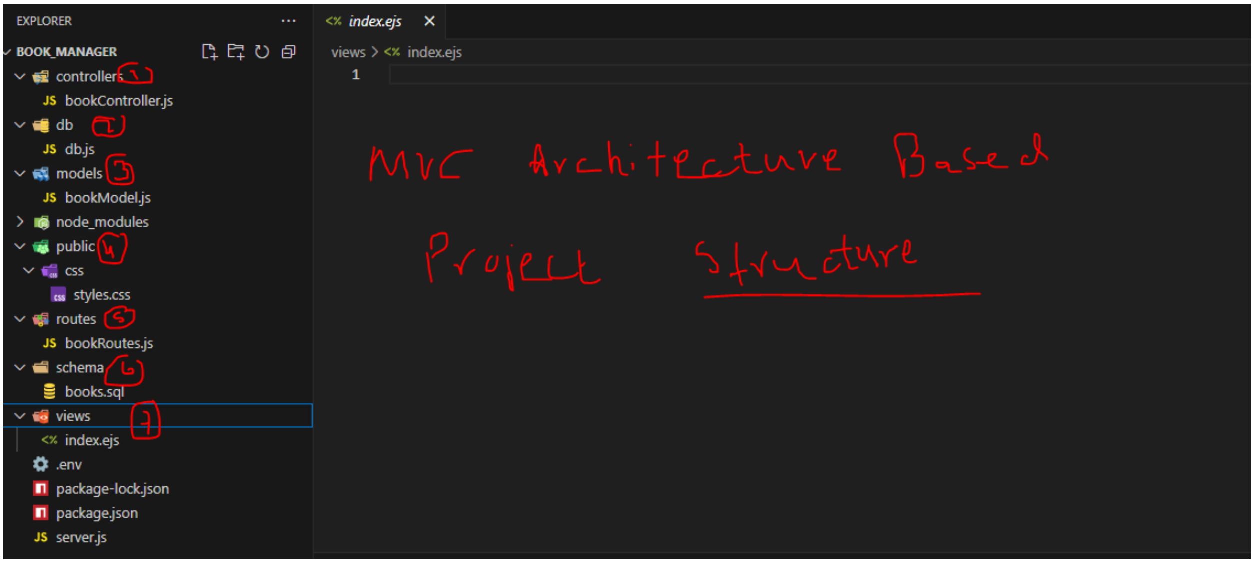
path parameter



```
          <!-- Delete Form -->
          <form
            action="/delete/<%= book.id %>"
            method="POST"
            class="d-inline-block"
          >
            <button type="submit" class="btn btn-danger btn-sm">
              Delete
            </button>
          </form>
        </td>
      </tr>
      <% }) %>
    </tbody>
  </table>
</body>
</html>
```

path parameter

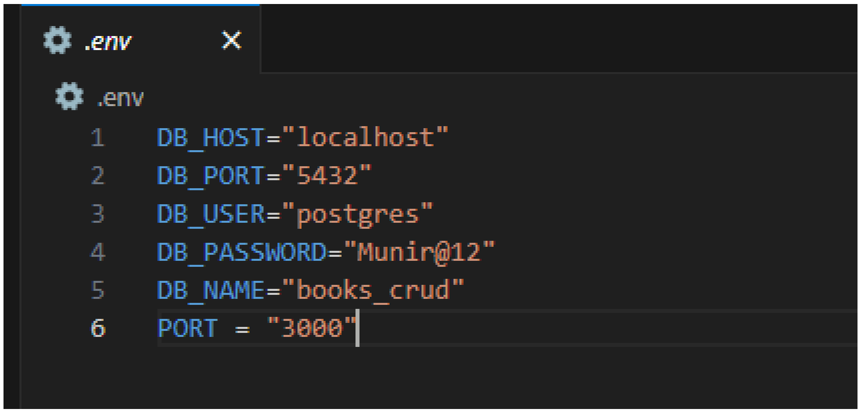
## 2: Backend



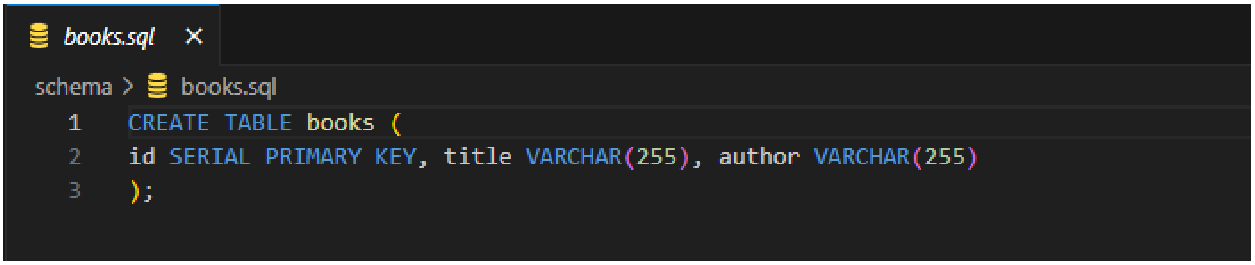
```
BOOK_MANAGER
  controllers 1
    bookController.js
  db 2
    db.js
  models 3
    bookModel.js
  node_modules
  public 4
    css
      styles.css
  routes 5
    bookRoutes.js
  schema 6
    books.sql
  views 7
    index.ejs
  .env
  package-lock.json
  package.json
  server.js
```

MVC Architecture Based

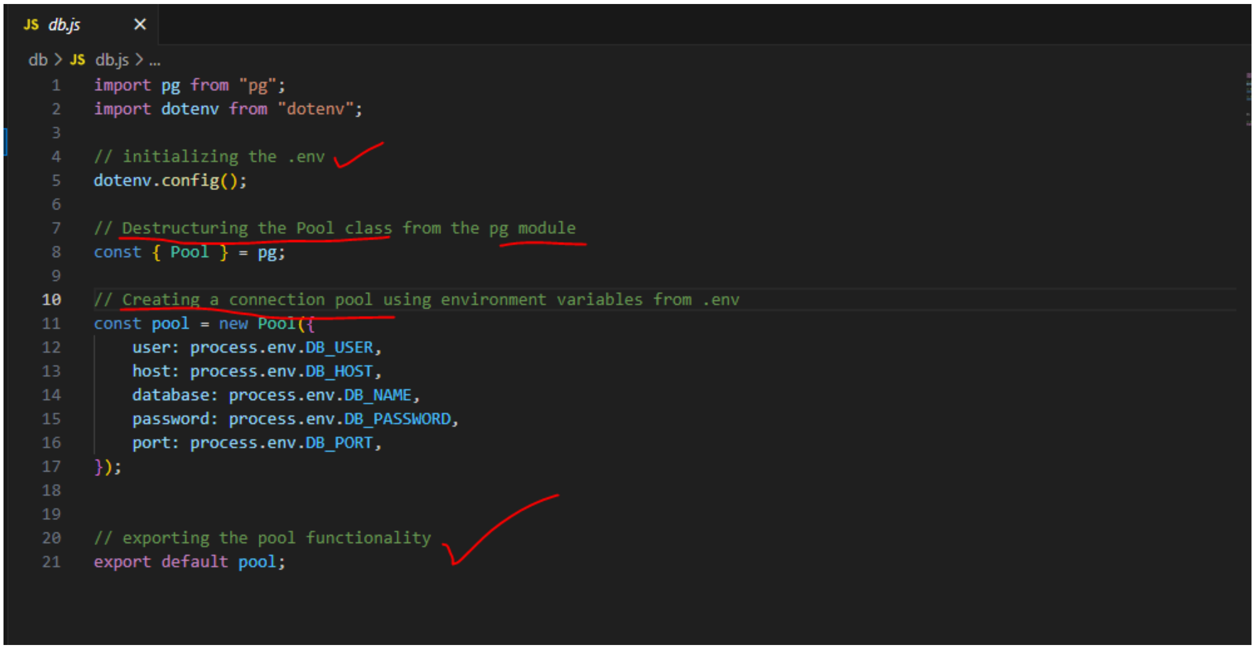
Project Structure



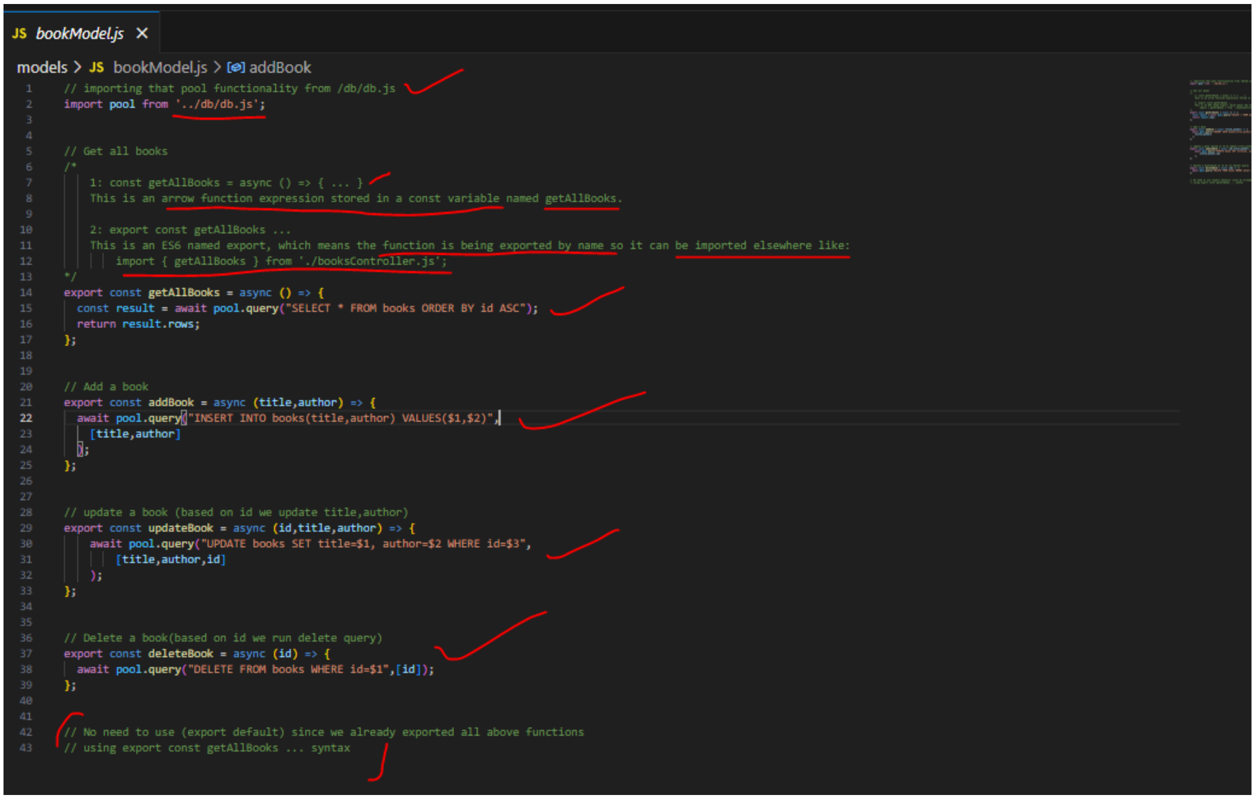
```
DB_HOST="localhost"
DB_PORT="5432"
DB_USER="postgres"
DB_PASSWORD="Munir@12"
DB_NAME="books_crud"
PORT = "3000"
```



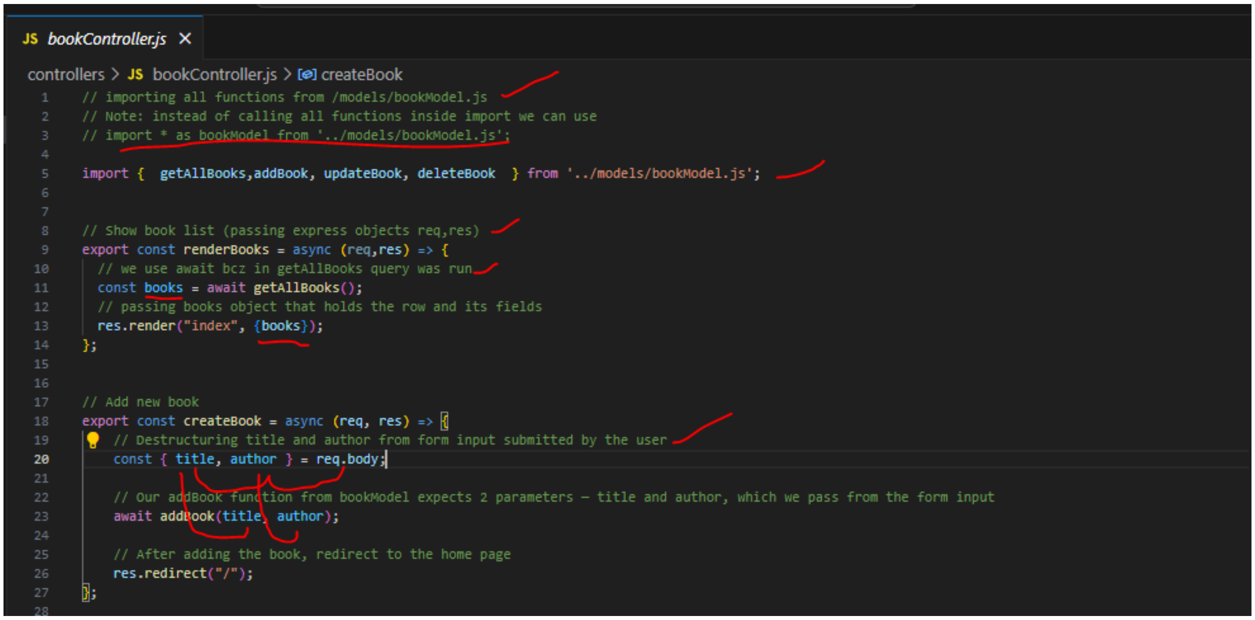
```sql
CREATE TABLE books (
id SERIAL PRIMARY KEY, title VARCHAR(255), author VARCHAR(255)
);
```



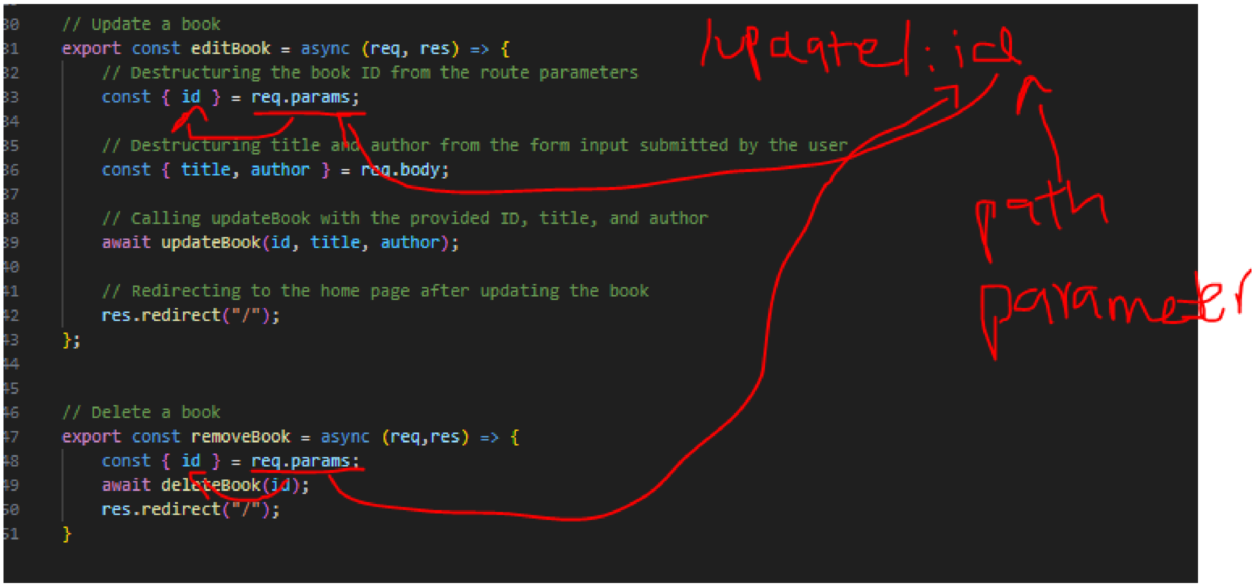
```
import pg from "pg";
import dotenv from "dotenv";

// initializing the .env
dotenv.config();

// Destructuring the Pool class from the pg module
const { Pool } = pg;

// Creating a connection pool using environment variables from .env
const pool = new Pool({
    user: process.env.DB_USER,
    host: process.env.DB_HOST,
    database: process.env.DB_NAME,
    password: process.env.DB_PASSWORD,
    port: process.env.DB_PORT,
});


// exporting the pool functionality
export default pool;
```



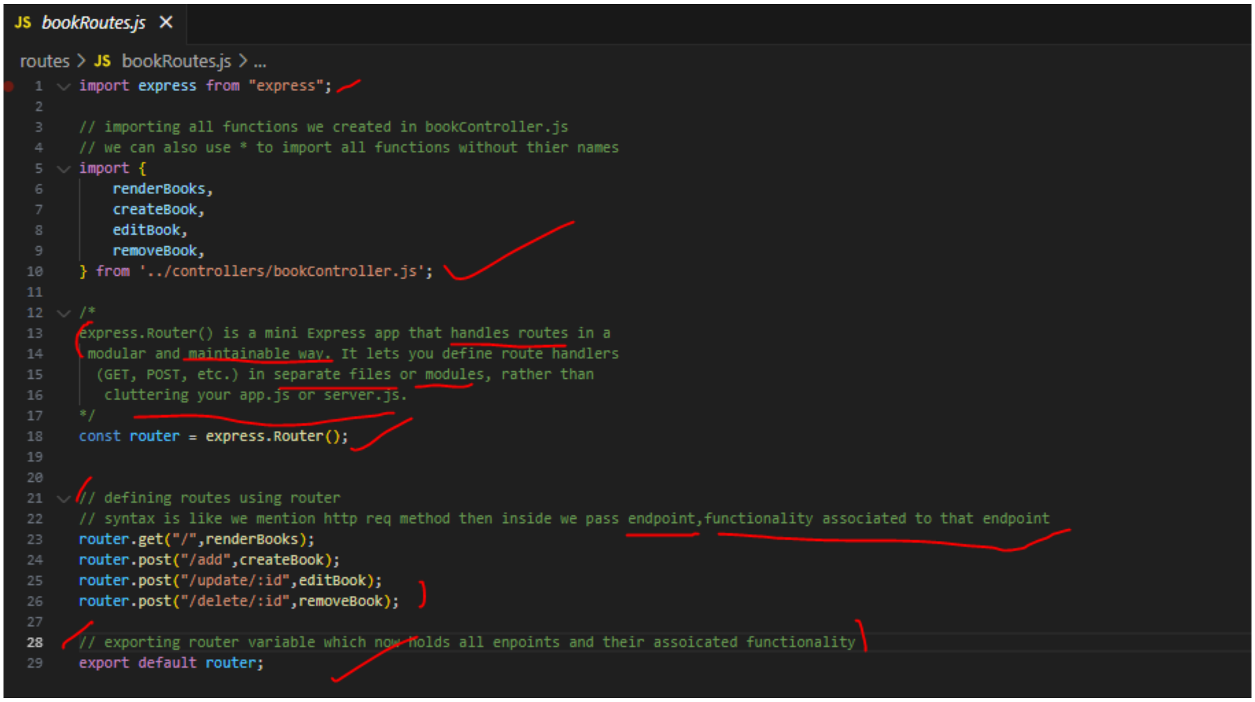
```
// importing that pool functionality from /db/db.js
import pool from '../db/db.js';


// Get all books
/*
    1: const getAllBooks = async () => { ... }
    This is an arrow function expression stored in a const variable named getAllBooks.

    2: export const getAllBooks ...
    This is an ES6 named export, which means the function is being exported by name so it can be imported elsewhere like:
        import { getAllBooks } from './booksController.js';
*/
export const getAllBooks = async () => {
  const result = await pool.query("SELECT * FROM books ORDER BY id ASC");
  return result.rows;
};


// Add a book
export const addBook = async (title,author) => {
  await pool.query("INSERT INTO books(title,author) VALUES($1,$2)",
    [title,author]
  );
};


// update a book (based on id we update title,author)
export const updateBook = async (id,title,author) => {
    await pool.query("UPDATE books SET title=$1, author=$2 WHERE id=$3",
        [title,author,id]
    );
};


// Delete a book(based on id we run delete query)
export const deleteBook = async (id) => {
  await pool.query("DELETE FROM books WHERE id=$1",[id]);
};


// No need to use (export default) since we already exported all above functions
// using export const getAllBooks ... syntax
```



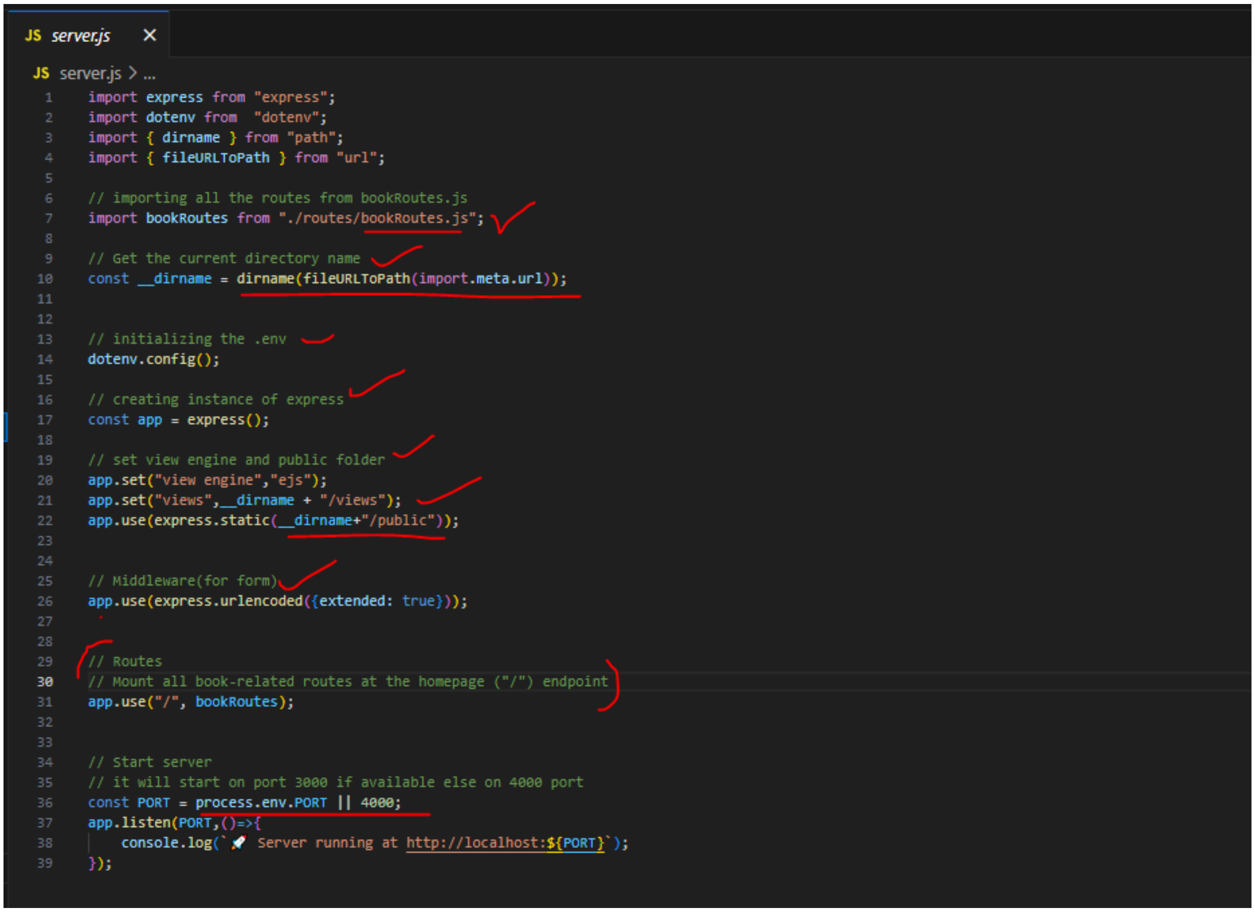
```javascript
// importing all functions from /models/bookModel.js
// Note: instead of calling all functions inside import we can use
// import * as bookModel from '../models/bookModel.js';

import {  getAllBooks,addBook, updateBook, deleteBook  } from '../models/bookModel.js';


// Show book list (passing express objects req,res)
export const renderBooks = async (req,res) => {
  // we use await bcz in getAllBooks query was run
  const books = await getAllBooks();
  // passing books object that holds the row and its fields
  res.render("index", {books});
};


// Add new book
export const createBook = async (req, res) => {
    // Destructuring title and author from form input submitted by the user
    const { title, author } = req.body;

    // Our addBook function from bookModel expects 2 parameters – title and author, which we pass from the form input
    await addBook(title, author);

    // After adding the book, redirect to the home page
    res.redirect("/");
};
```



```javascript
// Update a book
export const editBook = async (req, res) => {
    // Destructuring the book ID from the route parameters
    const { id } = req.params;

    // Destructuring title and author from the form input submitted by the user
    const { title, author } = req.body;

    // Calling updateBook with the provided ID, title, and author
    await updateBook(id, title, author);

    // Redirecting to the home page after updating the book
    res.redirect("/");
};


// Delete a book
export const removeBook = async (req,res) => {
    const { id } = req.params;
    await deleteBook(id);
    res.redirect("/");
}
```

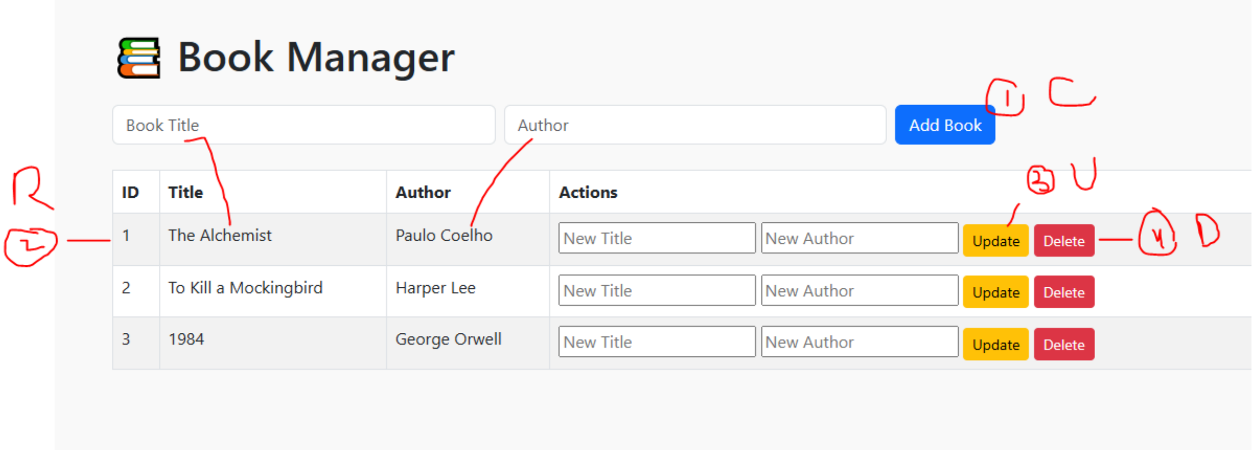


```javascript
import express from "express";

// importing all functions we created in bookController.js
// we can also use * to import all functions without thier names
import {
    renderBooks,
    createBook,
    editBook,
    removeBook,
} from '../controllers/bookController.js';

/*
express.Router() is a mini Express app that handles routes in a
 modular and maintainable way. It lets you define route handlers
  (GET, POST, etc.) in separate files or modules, rather than
   cluttering your app.js or server.js.
*/
const router = express.Router();


// defining routes using router
// syntax is like we mention http req method then inside we pass endpoint,functionality associated to that endpoint
router.get("/",renderBooks);
router.post("/add",createBook);
router.post("/update/:id",editBook);
router.post("/delete/:id",removeBook);

// exporting router variable which now holds all enpoints and their assoicated functionality
export default router;
```

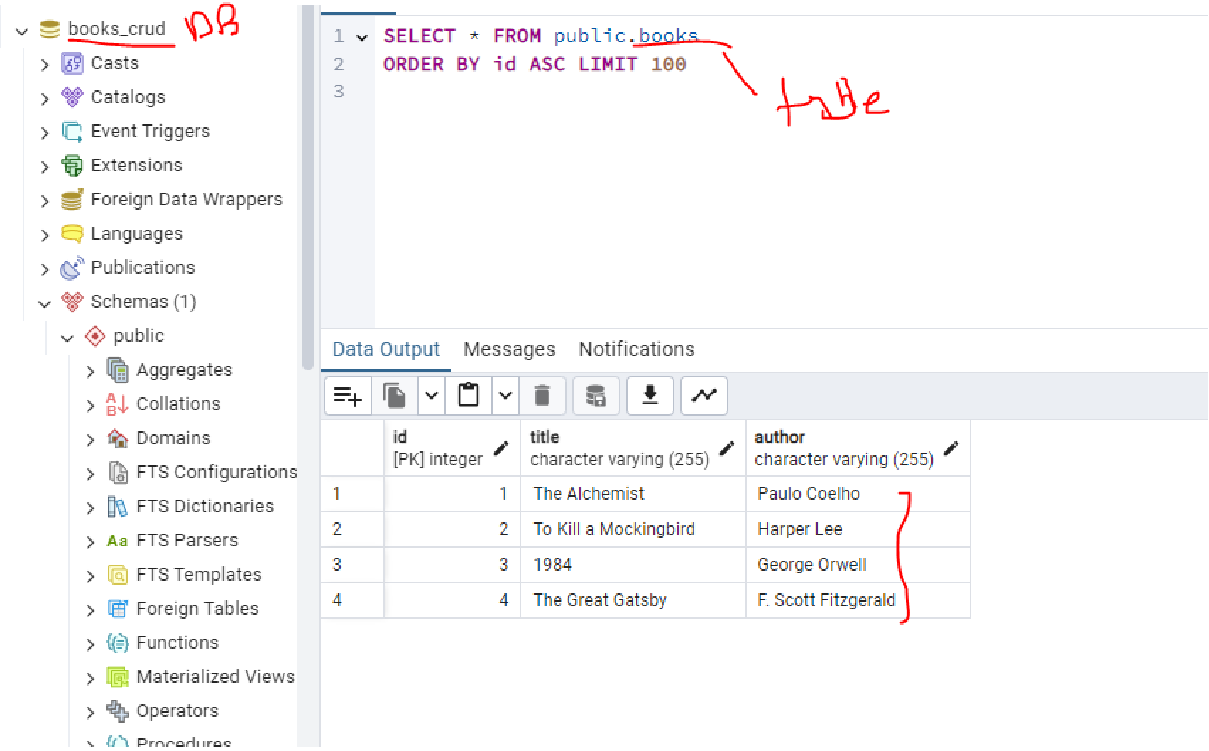


```javascript
import express from "express";
import dotenv from  "dotenv";
import { dirname } from "path";
import { fileURLToPath } from "url";

// importing all the routes from bookRoutes.js
import bookRoutes from "./routes/bookRoutes.js";

// Get the current directory name
const __dirname = dirname(fileURLToPath(import.meta.url));


// initializing the .env
dotenv.config();

// creating instance of express
const app = express();

// set view engine and public folder
app.set("view engine","ejs");
app.set("views",__dirname + "/views");
app.use(express.static(__dirname+"/public"));


// Middleware(for form)
app.use(express.urlencoded({extended: true}));


// Routes
// Mount all book-related routes at the homepage ("/") endpoint
app.use("/", bookRoutes);


// Start server
// it will start on port 3000 if available else on 4000 port
const PORT = process.env.PORT || 4000;
app.listen(PORT,()=>{
    console.log(`🚀 Server running at http://localhost:${PORT}`);
});
```

# 3: Results



| ID | Title | Author | Actions |
|---|---|---|---|
| 1 | The Alchemist | Paulo Coelho | New Title, New Author, Update, Delete |
| 2 | To Kill a Mockingbird | Harper Lee | New Title, New Author, Update, Delete |
| 3 | 1984 | George Orwell | New Title, New Author, Update, Delete |



```sql
SELECT * FROM public.books
ORDER BY id ASC LIMIT 100
```

| | id<br>[PK] integer | title<br>character varying (255) | author<br>character varying (255) |
|---|---|---|---|
| 1 | 1 | The Alchemist | Paulo Coelho |
| 2 | 2 | To Kill a Mockingbird | Harper Lee |
| 3 | 3 | 1984 | George Orwell |
| 4 | 4 | The Great Gatsby | F. Scott Fitzgerald |