# Case Study

# [Html+CSS+SASS+JavaScript/Typescript+react+NestJS]

**Abstract:**

This case study outlines the process of creating a Book shopping website using modern web development technologies such as React for the frontend and Nest JS for the backend. The project utilizes HTML, CSS, SASS, and JavaScript/TS for styling and client-side functionality, while React and Nest JS handle the frontend and backend logic, respectively.

**Introduction:**

In the digital era, e-shopping of Books has become an integral part of modern business. Creating an efficient and user-friendly e-commerce website requires leveraging cutting-edge web development technologies. In this case study, we will explore the implementation of a fictional website using React for the frontend and Nest JS for the backend. HTML, CSS, SASS, and JavaScript will be employed for styling and client-side functionality.

**Project Overview:**

The e-commerce website will consist of several key features including user authentication, product browsing, shopping cart management, and order processing. The frontend will be developed using React, a popular JavaScript/TS library for building user interfaces, while the backend will be built with Nest JS, a progressive Node.js framework.

**Implementation Steps:**

* **Setting up the Development Environment:**
  + Install Node.js
  + Set up a new React project using Create React App.
  + Initialize a new Nest JS project using the Nest CLI.
* **Designing the User Interface:**
  + Create static HTML pages for different sections of the website (e.g., Login page, Home, Book listings, Booking cart)
  + Use CSS/SASS to style the HTML elements and improve the overall design.
  + Implement responsive design to ensure compatibility with various devices and screen sizes.
  + Utilize these pages in React.
  + Reuse the above code during the below React Project requirement.

**Developing the Backend with Nest JS:**

* + Define Book RESTful API endpoints.

|  |  |
| --- | --- |
| **Method** | **Endpoint** |
| Get | <http://localhost:3100/api/books> |
| Get | <http://localhost:3100/api/book> |
| Post | <http://localhost:3100/api/book> |
| Patch/Put | <http://localhost:3100/api/book> |
| Delete | <http://localhost:3100/api/book> |

* + Set up MySQL or another database system to store user data, product information.
  + Implement JWT authentication at middleware to protect

**Delete:** <http://localhost:3100/api/book> routes that require user authentication.

* + Utilize Nest JS modules and dependency injection for modular and scalable code architecture.

**Building the Frontend with React:**

* + Convert the static HTML pages into React components.
  + Application should have a Book List page, Edit page and perform the CRUD operation.
  + Utilize React Router for client-side routing between different pages.
  + Implement state management using React Context or Redux for managing book cart and total amount.

Note*:* ***Front data of Book model and backend Book class properties must be compatible****. Make sure Book is string type.*

**Hints:**

**Backend:**

1. Create a Nest Project Application
   1. npm i -g @nestjs/cli
2. Install the necessary dependencies

"@nestjs/class-validator": "^0.13.4",

"@nestjs/common": "^10.0.0",

"@nestjs/core": "^10.0.0",

"@nestjs/platform-express": "^10.0.0",

"@nestjs/typeorm": "^10.0.1",

"class-transformer": "^0.5.1",

"class-validator": "^0.14.1",

"mysql2": "^3.9.1",

"nodemon": "^3.0.3" etc.

1. Set the port no to 3100 in main.ts
2. Test the application for app controller.
3. Setup DB Configuration in app.module.ts file

TypeOrmModule.forRoot({

type: <db>,

host: <>,

port: <port no>,

username: <username>,

password:<password>,

database: <dbname>

autoLoadEntities: true,

synchronize: true,

entities: [],

})

1. Create book-dto.ts,
   1. Install @nestjs/common
   2. Include the book properties such as id, name, price, publishing date etc.
   3. Add validators like @IsNotEmpty
2. Create a Book entity class with @Entity, @PrimaryGeneratedColumn, @Column etc.
3. Add imports: [TypeOrmModule.forFeature([Book])] in book module
4. Export the BookService and TypeORM I nbook Module
5. Add BookModule in app.module.ts
6. Create a BookRepository and decorate it with @EntityRepository
7. Create BookService class and decorate it with @Injectable
   1. Inject the BookRepository
   2. Create insertBook(….)
   3. Create fetchAllBooks()
   4. Create getByBookId(id)
   5. Create deleteBook(…)
   6. Create update(..)
8. Write a BookController with decorator @Controller
   1. Create getAllBooks(..) decorated with @Get
   2. Create getBookById(..) decorated with @Get
   3. Create changeBook(..) decorated with @Patch or @Put
   4. Create deleteBook(…) decorate with @Delete

Note: Complete one workflow for each functionality and test the each endpoint one by one.

**Front End:**

1. Create Book Add component to open a Book Form to accept the data from user.

**A screenshot of a book management app

Description automatically generated**

1. Create Book List component to list existing Books**A screenshot of a book list

   Description automatically generated**
2. Create Update book component to which allows you to change the data .
3. On click on delete button will delete the book.
4. To integrate the backend and frontend call nest rest api endpoint in react application**.**

**Optional:**

1. Implement the JWT in Backend to generate the token and test the authentication with one of the Book endpoint in Postman.