**MERN STACK WEB DEVELOPMENT DOCUMENTATION**

**TESTING**

**1.Jest(frontend):**A JavaScript testing framework developed by Facebook, primarily used for testing React applications but can be used with other frameworks as well.

**2.Mocha(backend):** A feature-rich JavaScript test framework running on Node.js, making asynchronous testing simple and fun.

**3.Chai:** A BDD (Behavior-Driven Development) / TDD (Test-Driven Development) assertion library for Node.js that can be paired with any testing framework.

**4.Supertest:** A library used for testing HTTP servers, built on top of SuperAgent.

* **Required Packages To Install:**
* npm init
* npm install express
* npm install dotenv
* npm install --save-dev nodemon
* npm install mongoose
* npm install bcrypt
* npm install body-parser
* npm install joi
* npm install jsonwebtoken
* npm install express-fileupload
* npm install multer
* npm install react-router-dom
* npm install cors (in backend)
* npm install react-toastify

**BACKEND**

**1. Initial Setup:**

1. Create a Folder – Backend

2. Open This Foler In Vs Code

3. Initialize Npm(Node Package Manager) as = Npm init

4. Create Index.Js File In Root

5. Copy Paste Api From Npmjs Website And Install Express Library (Npm Install Express)

Test this if it is working:

Const Express = Require('Express')

Const App = Express()

App.Get('/', Function (Req, Res) {

Res.Send('Hello Mern!')

})

Const Port = 3000

App.Listen(Port, () => {

Console.Log('Server Is Running On Port '+Port)

})

6. Install Dotenv To Create Virtual Environment (Npm Install Dotenv)

7. Create .Env File In Root And Write

Port =3000

8. Copy Paste Code From Npmjs – Dotenv In Index.Js And .Env And Process Env In Index.Js

9. To Run Server Type Node Index.Js Or Npm Run Script\_Name -> In Pacackage.Json

10. Install Nodemon It Is A Watcher Or Restarts Server Automatically After Any Changes In Code

11. Npm Install --Save-Dev Nodemon -> It Installs In Other Dependencies

12. Create Some Folders In Root -> Controllers, Helper, Middlewares, Models, Routes

13. Install Mongoose To Connect Backend Database (Npm I Mongoose)

**2. Backend Connection:**

14. Connect Db Using This Code And Specify Mongodbcompass Url In .Env File

Const Mongoose = Require('Mongoose');

// Connect To Mongodb Database Using Mongo Atlas

Mongoose.Connect(Process.Env.Db\_Url)

.Then(()=>{

Console.Log('Connected To Db');

})

.Catch((Error)=>{

Console.Log('Error In Connecting:',Error);

})

**3. Models Creation:**

15. Create Models Inside Models Folder And Create Schemas For Every .Js Files Below

a) User.Js => [Roll, Phno, Email, Pic, Role, Pass]

b) Product.Js => [Name, Description, Image, Category, Type, Priceperquantity]

c) Category.Js => [Name, Logo]

d) Cart.Js => [User, Products(Product, Quantity, Price)]

e)Order.Js => [User, Products(Product, Quantity, Price), Paymentype, Firstname, Secondname, Address, Address2, City, State, Zip, Status]

**4. Creating API Structure In Draw.Io:**

16. Create API route structure, method, and status app.drawings.net

A diagram of a software company

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

**5. Creating API’s For Different Functions:**

**a. Sign Up API:**

17. Creating user sign up api function in controller

18. Install bcrypt, encrypts the given password by user and hash it for privacy (npm install bcrypt)

19. Create user.js file in controller folder

20. Import required packages and create a signup function

21. Create a user.js file in routes folder and create a router post

22. Specify all the modules in index.js

24. Install body-parser to parse json files from user request(npm install body-parser)

25. Create validation.js file in helper folder

26. Install joi to validate the input fields should be required(npm install joi)

27. Import joi and create a schema of fields

**b. Postman For API Testing:**

23. Open and Create a workspace in postman to test api

**c. Login API:**

28. Create a login api function as signup api in controller

29. Specify login api route or post in routes.js file and test in postman

30. Create a login schema in validation file for email and password

31. Create a auth.js in middleware folder

32. Create a middleware authentication for login

33. Create a authMiddleware function in auth file to verify token and fetch user info from database and attach it to req.user variable in request object of next function module.exports = { authMiddleware } function to create token for user for 24 hours expiry time and attach it to user token field in database

34. Instal jwt (jsonwebtoken) to create tokens (npm install jsonwebtoken)

35. Create a function to create a token in controller

**d. User Info API:**

36. Userinfo is specified in authMiddleware to get user info to get this create a userinfo function in controller

37. Creating userinfo function

38. Specify user route with middleware in routes file

39. Import middleware in routes file and include between user path and userinfo

**e. Create Category API:**

40. Create a category file in controller

41. Create function to create new category and upload image

42. Create category schema in validation

43. Create category file in routes folder and specify and import category path or route

44.install express-fileupload used to upload files(npm install express-fileupload)

45. install multer(npm install multer)

46. Import all the packages in index.js

47. Test the api and upload the image and check in database

48. Make this category api as admin auth middleware

49.In auth file make a copy of authMiddleware and name it as authAdminMiddleware

50. Make changes in code

51. Export it

52. Import in category route and specify in route path

**f. Category Listing API:**

53. Create a function for category listing in category file in controller

54. After creating specify in category route as get method and test the api

**g. Product Creation API:**

55. Create product creating api only for admin by using authAdminmiddleware

56. Create product.js file in controller and create a function for creating product

57. Export this function

58. Create a product.js in router to create or specify a route to this product

59. Create a validation for this product function in validation

60. Import the route of this product function in index.js and app.use this

**h. Product Listing API:**

61. Create a product listing api in controller

62. Export and import in router as get method

**i. Product Details API:**

63. Create product details api in controller

64. Export and import in router

65. Use populate function to get the category details

**j. Products Modification API:**

66. Create a api for editing or updating products in controller

67. Export and import in router as put to update

68. Test the api by taking any product id and update it in postman by put method

69. Add authAdminMiddleware to authenticate for admin only

**k. Add To Cart API:**

70. Create a add to cart api in controller

71. Export and import in router as post and use authMiddleware for normal user

72. Import in index specify the router and app.use it

**l. Get User Cart API:**

73. Create a api to get user cart items

74. Export and import in router as get method to get the result

**m. Remove Cart Items API:**

75. Create a api to remove cart items

76. Export and import in router as delete method to remove

**n. Initiate Checkout API:**

77. Create a api to initiate checkout

78. Create a validation for initiate checkout

79. Export all and import in new checkout routes

80. Import this route in index and use

**o. Confirm Checkout API:**

81. Create a api to confirm checkout

82. Export and import in routes as put to update

**p. Order Listing API From Admin and User:**

83. Create a api to get order listing for both admin and user

84. Export and import in router

85. Import and user router in index page

**q. Order Info API:**

86. Create a api to get order info from findbyid

87. Populate only required things

88. Export and create its router

**r. Status Update API:**

89. Create api to update status of order

90. Get the order info by findbyid

91. Go to validation in helper folder and create a new schema for updating

Status and validate it in api

92. Find and update status and export and create its router

**Frontend**

**User Side:**

**1. Create React App:**

1.In MERN folder install react for frontend: npx create-react-app frontend (in vs code terminal)

2.Open this folder in vs code

3.npm start and check if it is working or not

**2. Create Pages and Folders:**

4.Create a folder pages in we will create all the pages

5.In pages folder create a Home file and write a function with html integrated

6.Create a modules folder and under this components folder

7.Create a helper folder

8.All folders should be in src folder

9.Now after writing a function in Home export it and import it in App.js within a function and run server

10.Go to bootstrap 5 and copy the very first code of css and paste it in public folder index.html under head tag

11.Create 2 files header and footer in components folder

12.Inside header file copy navbar code from bootstrap components and modify it

13.Write a small code for footer in footer

14.Create a layout file in modules and write a code and import all the header and footer and write dynamic code for body in layout

15.Go to index.css and write css for footer and change the appearance

**3. Installing Required Libraries:**

16.Install react-router-dom to render routes and pages

17.Go to react router site and copy code for routes from userroutes in App.js and modify

18.Then Import or copy browserrouter from same site from router components and copy in index.js file

19.In layout.js import outlet to render child from parent page and include in code

**4. Creating Pages and Writing Code:**

20.Create a products page like Home page in pages folder

21.Create a new Product.js in components folder

22.From bootstrap take a code of card and use it in above Product page modify and export

23.In Products page under pages folder Import the above page and write a code to display the card in products route and use array to display multiple products and remove style tags

24.Instal Axios and import axios in Products.js

25.Use useeffect and usestate in Products.js and modify the code

26.Change the Product.js code or modify it

**5. Creating .env File:**

27.Create a env in frontend

28.Go to backend and runserver and install cors and import and use app cors

**6. Login Sign Up:**

29.Go to Header.js and create two buttons for login and signup

30.Create a new file Signup.js in pages folder and create a function of SignUp and make a form and write a code in it to get the backend signup api and use usestate and modify the code like loading style and alert mssg take this from bootstrap and test it and lookup in mongodb if it is creating account successfully

31.Before testing the above go to app.js and create a router or link for signup by importing SignUp function

32.Open and run both server front and backend

33.Create a login.js in pages and write a function for login same as signup and modify the code

34.Export and import in app.js and set as route

35.Go to signup.js and add userEffect code in it and modify other stuff required

36.Go to header.js put a condition for user logged in or not and ad that in the form

37.Create a new profile.js file in pages and export and make a route of it

38.Go to app.js and add or include user logged in thing in it and pass it to routes parameter

39.Create a new checkauth function in app.js and assign it inside the element of routes

40.Test the api and data in local storage in inspect application

41.Open and run the servers and go to Headers.js and change the home link

**7. Product Page:**

42.Go to Product.js in components and create a function within Product function for add to cart working

43.Call the add to cart api, authorization, token and install react-toastify for stylish message

44.Return this function in the button as onclick function

45.Import the installed react toastify in the index.js and also in Product component and test this function in inspect, network

**8. Cart Page:**

46.Create a Cart.js in pages and write a function for getting cart listing in cart page

47.Create a table to display data also make it dynamic and write another function to remove the item from cart and assign it to a button

48.Export this and import and create a route in app.js

49.Create a cart button in header section in header.js

50.Test the both the api’s and check in database

**9. Checkout Page:**

51.Create a new page for checkout.js in pages

52.Write code to create a form and product listing into a function

53.Copy and paste the getcart api and all the table to display in checkout and modify code

54.Export the checkout and import in app.js and make a route and test it

55.Write code for initiate order and call its api and modify the code and add payment method

56.Create all the required useStates and use it and also create all the required functions and test the api

57.Change the actionCheckout function in checkout.js page

58.Now write code in the function of confirmCheckout and modify the getCarts function code in checkout and cart pages

59.import the navigator to navigate to next pages if order is confirmed and if it is empty

**Admin Side:**

**1. Profilemenu and Profile Page:**

60.Create a new page profilemenu in components and create two links for profile page and order and create a button for logout and make its function export it

61.Create a Profile page in pages and add the exported profilemenu links in this and display

62.Create a Order page in pages and create a function for order and it should include same code as profile page and also the order details or listing

63.To make admin auth route go to app.js and write a code to get userinfo and role of user

64.add that role In prewritten functions and create a new function for admin auth

65.Create a new folder inside pages and in this folder create a new page as Profile and write code inside this and export and import in app.js

66.Take the userinfo and role code to the header.js and paste in function and add the role in html code and modify the code

67. Take the userinfo and role code to the Profilemenu.js and paste in function and add the role in html code and modify the code

**2. Category Page:**

68.Create a new page category list inside admin folder and make a function to retrieve category listing api and use usestate and useeffect to store and render and write html code to display the content in the page and export the function

69.Go to app.js and import that above function and assign a route to It

70.Create a create button in categorylist page as link

71.Create a new page category form and copy all category list html into it and use usestate and input and file change handler to get input and file and create a api function to call api and add content type in headers and write remaining code and export and import it in app js and use it

also add loading animation using usestate

**3. Product Page:**

72.Create a new page for product listing in admin folder

73.Create a function for product list and use usestae and useeffect and get the api and display the content in frontend using html and before this go to backend produclist function and add populate inside that api and as category to display the category name or other stuff

74.Also use loading and at last export and import in app js and make a rout for it

75.Create a new page product form for creating product in admin folder

76.Write the html code to display the products on page and modify it

77.Validate that created form and validation in html

78.Add categories api to select different categories

79.Create a input and file change handler to capture the form data in it and locate it in html onchange

80.Create a action button to append formdata in it and call create product api in this function and add this function in html submit button as onclick and test the product creating also export and create its route

81.To edit the product go to app js and create a new rout or link for product edit

82.Go to product list and change the location of edit button

83.Now go to product form page and create a function to get the product details and call its api to display the details of product when edit is clicked and after getting the data edit the html code to display the name of the page using ternary operator also use if statement and use it in useeffect

84.Create another function to edit the product, use form data and call its api

85.Create a action submit function to assign the task either for creating or editing product by using if else and use this function or replace it In the submit button

**4. Order Page:**

86.Create a new page for order listing in admin folder and write the code for it call the api and write html code to display the content

87.Export and create a rout for this in app js

88.Create a new page for displaying order details and write html code to display it

89.Go to backend of order controller and create a new api to get details of order and export it and make a rout and test in postman

90.Come back to frontend and write a function to call that api and modify the code and display content by modifying html code and also make input change handler for status update

91.Go to backend of order controller and make new api for updating the status and export it and make a rout of it

92.Come to frontend in the same page of displaying order details and create a new function to update the status and call the api and use this as onclick in the html update button for status and test the api

**Assignment:** Show the admin or user info in the frontend