Step by Step Guide - Ecommerce Website using React and Django

SECTION 1 - STORE FRONT

1. Create new django project

- Install latest python version or 3.11.5
- Create New Folder called Django React Ecommerce
- In the folder, create two new folders called, backend and frontend
- CD into backend and create a virtual environment - `python -m venv venv`
- Activate the virtual environment - `venv\Scripts\activate or venv\bin\activate or source venv/bin/activate`
- Install Django - `python -m pip install Django==4.2`
- Create a Django project in current folder - `django-admin startproject backend .`
- Install Required Packages - `pip install djangorestframework==3.14.0 django-cors-headers==3.14.0 djangorestframework-simplejwt==5.2.2 PyJWT==2.6.0`
- Create requirements.txt - `pip freeze > requirements.txt or pip freeze -> requirements.txt`
- Create new app userauths, store, vendor, customer - `python manage.py startapp app_name`
- Create a .gitignore file and add the files to be ignored
- Install Apps in settings.py
- Run Python Manage.py Runsever

2. Setup Django Admin

- Install Django Jazzmin
- Add Jazzmin to settings.py INSTALLED apps Section
- Configure Jazzmin Settings
- Configure Jazzmin UI Tweaks
- Create Superuser
- Migrate Database
- Login to the admin section.

3. Configure Templates and Static Files

- Configure template configuration in TEMPLATES Section settings.py
- Create a new folder under Templates.
- Configure the static and media files in settings.py and urls.py
- Create media and static folders.
- Copy All Static Files and Add to static folder

4. Setup React

- Install Node
- Create new react app using vite.js (for better performance and load time)
- Install primary react packages/dependencies (Check package.json)
- Spin up the dev server - Tada!!

5. Create Django Custom User Model and Profile

- Create Custom user and profile models in core.models
- Register Model in Admin and Add filters, Displays, etc.
- Run Migration Commands
- Test Model in Django Admin Panel

6. Custom User Model and Profile Serializer

- Create Custom user and profile models in core.models
- Register Model in Admin and Add filters, Displays, etc.
- Run Migration Commands
- Test Model in Django Admin Panel

7. Discuss About Django JWT Authentication

- Discuss JWT Authentication
- Setup Settings.py jwt

8. Setup Login Serializer

- Create MyTokenObtainPairSerailizer Class to generate and refresh token

9. Setup Registration Serializer

- Create Registration Serializer Class to register users.

10. Token Obtain Pair and Registrations View

- Create token and registration views
- Register Views in URLS.py
- Test In Browser

11. Generate Robust Documentation using DRF-Yasg

- Generate docs using drf-yasg
- `pip install drf-yasg`
- Add `drf_yasg` to installed apps
- In main urls.py import the snippets for creating the yasg

12. Setting Up React.JS - Login, Register and Logout

Requirments

1. Make sure you have node.js installed
2. Yarn is recommended as package manager. You can simply type `npm install --global yarn` to install yarn
3. Any IDE, VS Code recommended
4. Git version control (optional but recommended)

- CD into frontend folder
- Create new project using vite.js

Why Vite and Not Create React App?

Using **Vite** to set up a React project is often a better choice than Create React App (CRA) for a simpler reason – speed and efficiency. Vite's development server is notably faster, meaning you spend less time waiting for your changes to take effect. It also creates smaller, more efficient code bundles, which results in faster page loads for your website visitors. Vite's configuration is straightforward, and it doesn't include unnecessary code for older browsers, which can make your site faster and more modern. Additionally, Vite is designed to work well with both Vue and React, offering you a fast and versatile tool for building web applications. Ultimately, it's about choosing a tool that makes your development process faster and smoother.

```
npm install --global yarn
```

```
yarn create vite . --template react

yarn

yarn add axios dayjs jwt-decode js-cookie
react-router-dom@6.10.0 zustand

yarn add -D simple-zustand-devtools prettier

yarn dev
```

Let’s Continue:

It's convenient when user information is saved in a place, like a storage space, so we can easily get it whenever we want. This creates a single, reliable source of truth. For example, if we look for a user and find nothing, it means the user is logged out. If we find the user's information, it means they're logged in. A package called **Zustang** will helps us do this easily.

- For creating a **zustang** user store. Create a folder in src named **store** & create a file named **auth.js** inside it.

  This store keeps track of user info, such as whether they're logged in or not. You can use a user function to find out the ID and username of the logged-in user. Once everything is set up, you can use DevTools to check the store's info in real-time.

Now, let's create utility functions that we will require for authentication purpose.
We will create **utils** folder inside **src** folder.
We will create 4 files, auth.js axios.js constants.js useAxios.js

Let's simplify this code in auth.js:
Login: This part needs a username and password. If the user exists in the database and the credentials are correct, they get logged in. We also save access and refresh tokens in a cookie.

Register: Here, we need a username, password1, and password2. It's how a user signs up. We check if the username is unique, and the passwords match on the server. If everything goes well, the user is automatically logged in.

Logout: This one simply logs the user out and erases their cookie.

Other Changes: When things like auth tokens and loading state change, the user's status is updated. We use useEffect to make this happen. The jwt_decode function is used to decode an access token.

Using useAxios.js for Managing Access Tokens
Issue: Access tokens have a short lifespan, meaning they're valid for only a brief period. After they expire, users can't access private parts of the application.

Solution: To address this problem, we need a way to check if the access token is still valid before sending a request to the server. If it's not valid, we should request a new token using a refresh token. Once we get a new access token, we can use it for API requests to private routes. If the token is valid, we can simply use it for the request.

How to Implement: We can solve this issue by using the axios library, which provides interceptors. These interceptors allow us to examine and modify requests before they are sent to the server. Therefore, we should use axios when making calls to private APIs. Additionally, if we obtain a new access token, we need to update our application's state. To streamline this, we can utilize a React Custom Hook.

Our Application will have 4 routes

`/login`

`/register`
`/`
`/protected` — Private Route
`/logout` — Log user out

If the user is logged in, they only should be able to access the private route, otherwise they should be redirected to the Login Page. We require a private route component that will make this possible.
Create a folder at frontend root, named `layouts` add these 2 files in it.
`MainWrapper.jsx` & `PrivateRoute.jsx`

The MainWrapper is like a big container that holds the whole app together. It's the one that tells the app who the user is.

PrivateRoute is like a security guard. It checks if the user is logged in. If they are, it lets them go to the page they want. If not, it sends them to the login page.

Now that we've done the tough part, we need to use this in the app. We need to make pages and parts of the app.

13. Password Reset (API)

- Create API to fetch user and send password email
- Configure API in url

14. Password Reset (Client)

- Create component to send email to server
- Create onChange and onSubmit handlers

15. Create Password (API)

- Create API to fetch password and related auth data
- Verify is user exists and token is valid
- Reset password
- Configure API in url

16. Create Password (Client)

- Create component to create new password
- Formdata and send it to bak

17. Configure Template In React

- Add Bootstrap CDN to index page
- Create Header and Footer
- Imort the Home page
- Override the register, login and logout page

18. Create Store Model In Django

- Create Store, CartOrder, CartOrderItems, Wishlist and Other required
- Register Models In Admin
- Test In Model

19. Serialize Store Model In Django

- Serialize Models In Admin

20. Create API to list Category, Product and Product Detail
    - Write View to list all published products from the database
    - Write another view to get and show product detail
    - Configure views in URLs

21. Create React Function for Addon
    - On the HomePage list all products via rest api

22. List Product using React
    - On the HomePage list all products via rest api

23. Get Product Detail using React
    - On the HomePage list all products via rest api

24. Add Product to Cart (API)
    - Create new django api to add product to cart
    - Configure API in URLs

25. Get User Country and Location
    - Create a new folder in view called plugin
    - THen create a new file called UserCountry.jsx
    - Write and export function to get user country

26. Get Logged In User Data
    - Create a new folder in view called plugin
    - THen create a new file called UserCountry.jsx
    - Write and export function to get user country

27. Create and Set User Cart ID
    - Create a new folder in view called plugin
    - THen create a new file called UserCountry.jsx
    - Write and export function to get user country

28. Add Product to Cart (Client Product Detail)
    - Write function to handle color, size and input change
    - Create new function to addToCart()

29. Add Product to Cart (Client Product List)
    - Write function to handle when a color, size or input button is clicked or changed
    - Update the state with the required values
    - Send the data to the addToCart() data

30. Fetch Cart Total Items Count (API)
    -

31. Fetch Cart Total Items Count (Client)
    - Write function to handle when a color, size or input button is clicked or changed
    - Update the state with the required values
    - Send the data to the addToCart() data

32. Cart List View (Client)

- In src>views>shop create a new file called Cart.jsx
- Import cart.html template to the new Cart.jsx
- Import needed plugins and packages
- Write code to fetch cat data from the api and display it in template

33. Update Cart

- In Cart.jsx write code run to get cart data
- Update cart by sending data back to the addToCart() function

34. Fetch Catch Total (API)

- In store.views, write an API view to get cart total
- Register this views in urls
- Test API

35. Fetch Catch Total (Clients)

- In Cart.jsx, fetch the cart total via the CartDetail Api View
- Update the jsx with the new data

36. Remove From Cart (API)

- Write API to delete item from cart
- Check if user is authenticated or not
- Configure the url
- Test with RestMan

37. Shipping Details (Client)

- Write a function to get all shipping address and set it to the state

38. Create Order View (API)

- Write a function to get payload from frontend and create a new order
- Calculate the required informations and validate data

39. Create Order (Client)

- Using react, create a new function to create order
- Make all shipping and personal informations required

40. Checkout Page (API)

- Create a new checkout view api to list get the user order based on the order oid
- COnfigure the URL

41. Checkout Page (Client)

- Create a new checkout page in react, configure the checkout page url
- Fetch the order details from the checkout view API
- Populate the template with the required Informations

42. Coupon (API)

- Write a function to get vendor coupon and remove the discount from the item
- Configure URL

43. Coupon (Client)

- Using react, apply coupo discount

44. Stripe Payment (Client and API)

- Using react, make a stripe payment on the frontend and send the data to backend for validation.
- Implement Webhook to get event data in real-time
- Validate payment and set order status if payment is successful

45. Success Page (API)

- Write a view to fetch completed and paid order
- Configure the API

46. Success Page (Client)

- Using React, create a success page after payment
- Configure the URL
- Fetch the order data from API
- Populate the JSX Template

47. Invoice (Client)

- Using React, create an invoice page after payment
- Configure the URL
- Fetch the order data from API
- Populate the JSX Template

48. Send Order Email (API)

- **Setup Mailing System**
- **Send order confirmation and summary to buyer**
- **Send Order Items Email to Vendor**
- **COnfirm that email is sent**
- **Test :)**

49. Paypal Payment (Client and API)

- Using react, make a flutterwave payment on the frontend and send the data to backend for validation.
- Implement Webhook to get event data in real-time
- Validate payment and set order status if payment is successfull

50. Create Review (API)

- Write API to get data from payload
- Use data to get user and product
- Create a new review using the payload data
- Test API

51. Create Review (Client)

- Create a createReview State
- Create a handleRevieChange
- Configure Input Fields, Forms and button
- Create new formdata, append the value and send to api

52. List Review (API)

- Create a view ListAPIView to fetch reviews based on product id
- Configure view in URLs
- Test In Browser

53. List Review (Client)

- Fetch product review via ListAPIView
- Set the values to the state
- Iterate over the review and render the data in the jsx template
- Test ;)

54. Search (API)

- Create a new function to search products based on query
- Configure new view in urls.py

55. Search (CLient)

- Create a new search component
- In storeheader, create search handlers, append query in url
- Send data to backend and return data to frontend.

SECTION 2 - BUYER DASHBOARD

56. Account Page

- Configure Customer Template and Components
- Fetch User Profile and Show Data in Sidebar
- Configure the Account Page

57. Order Page (API)

- Create new api to list orders
- Configure the API endpoint

58. Order Page (Client)

- Fetch Orders From API
- Configure the Account Page

59. Order Detail Page (API)

- Create new api to get single order
- Configure the API endpoint

60. Order Detail Page (Client)

- Fetch Orders Detail From API
- Configure the Order Detail Page

61. Add To Wishlist (API)

- Create new api to get create a wishlist
- Configure the API endpoint

62. Add To Wishlist (Client)

- Create a new function in plugin directory for addToWishlist
- Write the function to add items to wishlist
- Configure the required button to call the function

63. List Wishlist (API)

- Create new api to get list items from wishlist for a user
- Configure the API endpoint

64. List Wishlist (Client)

- Create a new wishlist component
- Fetch Wishlist Items from API
- Map through fetched items and display data in jsx template

65. Create Notification (API)

- Create a new function to send notifications

66. Customer List Notification (API)

- Create a new class to list customer notification
- Configure URL endpoints

67. Customer List Notification (Client)

- Create a new class to list customer notification
- Configure URL endpoints

68. Update Customer Account Detail (API)

- Create a new class to list customer notification
- Configure URL endpoints

69. Update Customer Account Detail (Client)

- Create a new class to list customer notification
- Configure URL endpoints

SECTION 3 - VENDOR DASHBOARD

70. Dashboard Page (API)

- Write API to fetch Vendor Stats
- Write API to fetch Orders and Products
- Configure API Endpoint in URLs.py
- Test In Browser

71. Dashboard Page (Client)

- Configure Customer Template and Components
- Fetch Stats, Products and Orders From API

72. Dashboard Charts.JS (API)

- Create API to fetch generate chart data for products and order
- Configure endpoint

73. Dashboard Charts.JS (Client)

- Fetch Chartdata for orders and products
- Populate Line chart with data

74. Products Page (API)

- Write API to fetch vendors product
- Configure URL

75. Products Page (Client)

- Create product page and register url in App.js
- Fetch Producs from API
- Display items in jsx template

76. Add New Product (Client)

- Creae a new component for add product form
- Register Component in App.js
- Write code to create fields dynamically
- Write code to remove dynamic fields
- Compile data to be sent to backend
- Create new formdata and appends necessary data
- Post to backend API

77. Add New Product (API)

- Create a new CreateView APi
- Overide the perform_create method
- Parse data received from frontend payload
- Loop through items and save necessary items to the database
- Overide the save nested data to save the new nested data

78. Update Product (Client)

- Creae a new component for update product form
- Register Component in App.js
- Import most re-usesable code from AddProduct.js
- Compile data to be sent to backend for update
- Create new formdata and appends necessary data
- Post to backend API

79. Update Product (API)

- Create a new UpdateView APi
- Overide the update method
- Delete other nested data in the database
- Loop through new or old items and save necessary items to the database
- Overide the save nested data to save the new nested data

80. Delete Product (API)

- Create a new DestroyView APi
- Fetch Product that is to be delete
- Delete the product using delete method

81. Delete Product (Client)

- Create a new DestroyView APi
- Fetch Product that is to be delete
- Delete the product using delete method

82. Filter Product (API)

- Create a new function to filter product
- Get Param from url that was sent from react
- Use Param to filter product
- Return the filtered product

83. Filter Product (Client)

- Create a new function to handle Product Filter
- Pass the url param dynamically
- Make a get request to an API servert

84. Order List (API - Already Exists)

- Create a new Order List api for vendor or used existing API
- Configure the view in the url
- Test ;)

85. Order List (Client)

- Fetch the new orders via the orderlist api view
- Set the values to the state
- Iterate over the orders and render the data in the jsx template
- Test ;)

86. Order Detail(API)

- Create a new Order RetrieveApiView
- Configure the view in the url
- Test ;)

87. Order Detail(Client)

- Create a new component for vendor order details
- Use the customer order detail page
- Get and set order item in state
- Render Order
- Iterate over the orderitems and render the data in the jsx template
- Test ;)

88. Earning(API)

- Create a new Earning ListApiView
- Configure the view in the url
- Test ;)

89. Monthly Earning(API)

- Create a new Monthly Earning List (fbv so we can use it for chart)
- Configure the view in the url
- Test ;)

90. Earning(Client)

- Fetch the new earning data via the earning list api view
- Set the values to the state
- Render Required Data in the frontend

91. Review List (API)

- Create new LIst API View to fetch all vendors reviews
- Register this view in the url

92. Review Detail (API)

- Create new Retrieve API View to fetch specific vendor review
- Register this view in the url

93. Review List (Client)

- Create a new component to fetch all vendors reviews
- Set review to state
- Render review in the temmplate

94. Review Detail (API)

- Create a new component to fetch all vendors reviews
- Set review to state
- Render review in the template

95. Review Reply (Client)

- Create a new handlers to change reply and submit
- Create a new form to submit reply
- Attach the reply function to form

96. List Coupon (API)

- Create a new view to list vendor created coupon
- Register View in URL

97. Create Coupon (API)

- Create new view to get data from frontend and create new coupon
- Register View in URL

98. Detail Coupon (API)

- Create a new view to get vendor single coupon
- Register View in URL

99. Coupon Stats (API)

- Create new view to get coupon stats
- Create new Coupon Summary Serializer
- Register view in URL

100. Coupon List (Client)

- Create a new component to list coupon
- Fetch coupons via CouponList api endpoint
- Set fetched coupon data to state
- Render coupons in template

101. Coupon Create (Client)

- In the coupon list component, create a new modal to creat coupon.
- Create coupon handle
- Create formdata
- Post data to coupon create api then fetched updated data

102. Coupon delete (Client)

- In the coupon list component, create a new handler for coupon delete
- Create a delete handler
- Send a delete request to the detail view to delete coupon
- Fetch updated data

103. Coupon Update(Client)

- Create a new component for updating coupon.
- Get param from url
- Make a get request to the coupon detail view
- Fetch and set coupon to state
- Update the input field with the updated coupon data
- Make a patch request to the coupon detail view and fetch updated data

104. Notification Unseen (API)

- Create a new List View to fetch all vendor unseen notifications
- Configure view in URL

105. Notification Seen (API)

- Create a new List View to fetch all vendor seen notifications
- Configure view in URL

106. Notification Summary Stats(API)

- Create a new List View to fetch all vendor notifications summary
- Configure view in URL

107. Notification Unseen List (Client)

- Create a new component to list notifications
- Fetch notifications via NotificationsList api endpoint
- Set fetched notifications data to state
- Render notifications in template

108. Notification Seen List (Client)

- Create a new modal to list seen notifications
- Fetch notifications via NotificationsList api endpoint
- Set fetched notifications data to state
- Render notifications in template

109. Mark Notification As Seen (API)

- Create a new function to mark notification as seen
- Make a get request to the endpoint in the server that marks notification as seen.
- Configure thebuttton that marks notifications as seen.

110. Vendor Settings (API)

- Create a new function to mark notification as seen
- Make a get request to the endpoint in the server that marks notification as seen.
- Configure thebuttton that marks notifications as seen.