* First step in any project is (to build folder structure and instal [main] libraries)
* Because the team lead is the only one responsible, so he builds this structure and push it on (main branch) and then all team members drive there branches with the same structure.
* Put project Layouts (MainLayout, AuthLayout, ..)

Notes:-

* Remove (strict mode) from the main.tsx file
* Because it makes the (useEffect) executes many times
* Open a new terminal from frontend part -> cd .. –> cd Backend
* Folder that contains more than 1 component
* Starts with a (lowercase letter)
* [important] (vite-plugin-svgr) is used to allow (svg) to be imported as a component inside vite projects
* (import + configure it) in the (vite config file)

Like => import svgr from "vite-plugin-svgr";

 plugins: [react(), svgr()],

// vite.config.ts

import { defineConfig } from "vite";

import react from "@vitejs/plugin-react";

import svgr from "vite-plugin-svgr"; //this

// https://vite.dev/config/

export default defineConfig({

  plugins: [react(), svgr()], //this

});

* configure in (tsconfig.app.json)

Like => "types": ["vite-plugin-svgr/client"]

// tsconfig.app.json

{

  "compilerOptions": {

    //...

    "types": ["vite-plugin-svgr/client"]

    // ...

  }

}

* (import "../../assets/logo.svg?react")

Put (?react) at the end of the import

import Logo from "../../../assets/svg/cart.svg?react";

* <Navbar> <li></li> <li></li> </Navbar>
* This is a custom header called navbar which can be used in multiple page (But each page has it own links)
* So I send links as (children) to the cusom navbar to render them there.
* <Nav className="me-auto">
* {children links array}
* </Nav>
* Here I mean that I can make a component for the list of links appear in my app, and make this component dynamic by accepting diff list each time (according to user permissions)
* Instead of make the parent component flex and align items there, I can align child component from inside it’s styles by (align-self: flex-end)
* \*\*\* To make (Footer) appears at the bottom of it’s parent component

1. Parent component =>

(display: flex; flex-direction: cloumn)

1. Footer (Child) => (margin-top: auto)

* vite-tsconfig-paths => Allows us to type (@)in paths instead of ../ or ./

(configure it like vite-plugin-svgr)

* All plugins installed on vite project have same

congiuration in (vite.config.ts, tsconfig.app.json)

((( but here tsconfig I put the routes I want )))

And (imported in the same way)

* < Nav.Link as={NavLink} to="/about-us">
* About
* </Nav.Link>

This code here means that (Nav.Link) is a component in (Reactstrap) and I want it (to act as [NavLik] component) in react-router-dom

* <NavLink

to=”/about-us”

key={index}

className={(navClass) =>

navClass.isActive ? "active\_\_menu" : ""

}

>

About Us

</NavLink>

(This (NavLink) component give (active) class for (the active route) (current route) and I can give this class some custom styles in my css file (or in global styles file and import this global styles file in the main.ts file )

global.css

.active {

  color: red !important;

  text-decoration: underline;

}

Here I gave the active class some custome styles (provided for links in <NavLink /> Component)

* <Link to="/" replace={true}></Link>

Replace here make me when I click on back button on browser to not return to this page

\*\*(Because browser makes a stack of page LIFO so if I am in the page 3 on the stack and click back this returns me to page 2, and if I clicked forward i will return to page 3, here If I put {replace} on the link this removes page 3 from the stack to i can to return to this page)

(used usually with error pages)

* {
* path: "products/:prefix",
* element: <Products />,
* loader: ({ params }) => {
* if (
* typeof params.prefix !== "string" ||
* !/^[a-z]+$/i.test(params.prefix)
* ) {
* throw new Response("Bad Request", {
* statusText: "Category not found",
* status: 400,
* });
* }
* return true; //Means continue and load the next page
* },
* // Condition to confirm that the category prefix is a pure string, because there is not category with name of any other characters
* // So here we reduce backend calls to make the app more efficient
* // throw new Response() => This is how we return errors in react-router-dom (to navigate to dynamic error page)
* }
* if (
* typeof params.prefix !== "string" ||
* !/^[a-z]+$/i.test(params.prefix)
* )
* {
* ...
* }
* // Here I put a guard in typescript (typeof params.prefix !== "string")
* // because in typescript (params.prefix) could be a string or undefined
* In eCommerce apps, it is not logic to call categories api every time I visit categories page  
  (Because it is not usual that categories are changing every day or every hour)

But it is logic to call products api each time because products are updated eventually

* Generics:-

Instead of building a cutom fuction that each time I pass the type of argument I pass (in ts)  
- I can build a gentric function that knows the type of argument (on executing not on building)

* // EX 1
* const getargs = <T,>(arg: T) => {
* return arg;
* };
* getargs<number>(5);
* getargs(5);
* // true also, because typescript is smart enough
* // to know the type of argument i pass
* // EX 2
* const getGenericArray = <T,>(list: T[]) => {
* return list;
* };
* getGenericArray(["string1", "string2"]);
* // EX 3
* type TList<T> = T[]; //type here (accepts an argument type)
* const getGenericArray = <T,>(list: TList<T>) => {
* return list;
* };
* // EX 4
* type TObject<T> = {arg: T}; //type here (accepts an argument type)

We type (T) here not (Type) because community used to do that, we type comma here <T,> to not make an error

* Extending an object:-
* type TObj = { id?: number; name: string; age: number };
* type hasID = { id?: number };
* const returnObject = <T extends hasID>(obj: T) => {
* return obj.id;
* };
* returnObject<TObj>({ id: 1, name: "testName", age: 30 });
* // if I remove (extends hasID) will be an error here , because it can not access and see (id) in object
* If I have a function in parent component, I want to execute it based on event (click) happens a child component
* // parent:-
* const parent = () => {
* const deleteHandler = (id) => {
* // ....
* }
* <User delete={deleteHandler} />
* }
* // child:-
* const User = ({delete}) => {
* <div onClick={() => delete(id)}>delete</div>
* }
* Render props pattern:-

Is the same as last example, but (the function in the parent invokes based on iteration of each child item [not based on an event])

// parent:-

const parent = () => {

const renderHandler = (element) => {

  // ....

  console.log('Current element in iteration')

  console.log(element)

}

<User records={records} renderHandler={renderHandler} />

}

// child:-

<User records={records} renderHandler={renderHandler} />

const User = ({records, renderHandler}) => {

  records.map(element => renderHandler(element))

}

* Cart slice
* interface ICartState {
* items: { [key: string]: string }; // id: quantity
* productFullInfo: TProduct[]; // Comes from backend each time I visit
* shopping cart page
* }
* // Beacuse may product info change (price -image - ...) we only store it's (id - quantity)
* // And each time I visit cart page => I make a call to backend with (ids) here in my state
* // Backend response with the (full info) of these products
* // I merge these full info with the quantity I have here , then show results to the user.
* We put the functionality of (add to cart) [inside]

The product component, because we will use (the same component with the same functionality in many place)

{{{

If we use the same component many time but with (different) functionalities => [send the function as a (callback) from parent

}}}

* const onj = {1: 'test', 2: 'test 2'}
* // state.items[id] => Check if (id) exists in the (obj)
* const cartItems = useAppSelector((state) => state.cart.items);
* const cartItemsQuantity = Object.values(cartItems);
* const totalQuantity = cartItemsQuantity.reduce(
* (accumulator, currentValue) => {
* return accumulator + currentValue;
* },
* 0
* );
* // This is how I get the some of all value from an object
* ({'test': 5, 'test2': 11, ...})
* // reduce() => Used in summition of many values from an array (takes 0 as an initial value)
* Store:-

When ever any action happens in an (slice) all of it’s (selectors) will work

Action1:- (store init) => Will fire all selectors of all slices

Action2: (Dispatch of a slice action)

Will fire also all selectors of this slice, But

(Selector has a level of optimization)

So the selector see if it receives a new value from the slice so => It will work

But if it receives the same value it has => It will (not) fire

[when the selector fire => All actions of it’s slice invoke]

* Here I get a function from the slice use selector
* const totalQuantity = useAppSelector(getCartTotalQuantity);

Here I have a problem, since this function is inside the slice, so it executes each time this slice has any any call, so kind of (optimization)

To go to this function in the slice and make it’s body do not execute except (when it’s argument changed) [ this done using ((( createSelector ))) ]

* These are the same (because JS sends the state automatically here)
* const totalQuantity = useAppSelector(getCartTotalQuantity);
* const totalQuantity = useAppSelector(state => getCartTotalQuantity(state));
* createSelector():-

- used inside redux store(in slices)

-checkes if argument value changed or not

- if it’s not changed, it stopes the execution of it’s function body and returns to the select the old value it had

const getCartTotalQuantity = createSelector(

  (state: RootState) => state.cart.items,

  (items) => {

    // items => The return of [ (state) => state.cart.items ]

    const totalQuantity = Object.values(items).reduce(

      (accumulator, currentValue) => {

        return accumulator + currentValue;

      },

      0

    );

    return totalQuantity;

  }

);

* In many eCommerce apps there is something called (Fair Distribution Policy), which is that user is only allowed to add specific amount of each to cart or to shopping list (EX:- 5 items of each product), and if he exists that amount I must prevent him.

* [Nullish ***coalescing***](https://www.google.com/search?sca_esv=f1335be37f2cdbad&rlz=1C1GCEA_enEG1098EG1098&sxsrf=ADLYWIILdCaBIksmFVzzm2EgNmdfU-OSsA:1729464679239&q=what+is+Nullish+coalescing&spell=1&sa=X&ved=2ahUKEwiR9-3-hZ6JAxWTVaQEHUizHwgQkeECKAB6BAgKEAE)
* const currentRemainingQuantity = max - (quantity ?? 0);
* // Nullish coalescing
* // ((( quantity ?? 0 ))) ===>>> take quantity value, but if it is (null | undefined) make it equals (zero)

( || ) or is the same here but it deels with

(empty strings) as well

* In products page when ever a specific variable changes in it, it reloads, (and it’s child (product component) reloads also).

To prevent that use (memo) inside the child component

* I must do (separation of concerns) so for every part I make 2 components (container component) to contain logic and (presentational component) to render view data

EX:- (presentational) is here is a for loop on the presintaion component

* items: { [key: string]: number };
* // id: quantity (key in object => Js always transform it to a (string))
* Optimization
* const CartItem = memo(
* ({
* id,
* title,
* price,
* img,
* max,
* quantity,
* changeQuantityHandler,
* }: CartItemProps) => {
* ...
* }

Here in this part I used (memo) for optimization for (cashing all props values) and only render when a prop change.

But it was not enough, because here I also pass a function reference (and each time it pass a new reference even for for the same cartItem)

So the solution here is also to use (uceCallBack) on the prop fuction with using (memo)

  const changeQuantityHandler = useCallback(

    (id: number, quantity: number) => {

      dispatch(cartItemChangeQuantity({ id, quantity }));

    },

    [dispatch]

)

(Encapsulated Component):-

It is the component that has it’s own logic and also view or represents data inside it.

((( Container and view component ))):- (Default)

(This is a technique) in which I have two components

First (parent component) which has all logic

Second (child component) which view the data and uses logic passed from the parent.

* In most cases we use (Container & view)

But I also should use (encapsulated) sometimes for more (optimization),

Like now I have the (HeaderBasket) get data from selector, this is data is called in Header component (parent) so the whole header will be reloaded (navigation – logo - …)

So the best case here to use (Encapsulated) so every change happens in headerbasket data only affects the headerbasket component only

* Apply actions based on response
* dispatch(actLikeToggle(id))
* .unwrap()
* .then(() => setIsLoading(false)) // for success response
* .catch(() => setIsLoading(false)); // for network errors
* Take in consideration that while (loading) I should prevent its action
* const likeToggleHandler = () => {
* if (!isLoading) {
* setIsLoading(true);
* dispatch(actLikeToggle(id))
* .unwrap()
* .then(() => setIsLoading(false)) // for success response
* .catch(() => setIsLoading(false)); // for network errors
* }
* };

Only do this logic if it is not loading

* Apply an action (on leaving page)
* useEffect(() => {
* return () => {
* dispatch(productsFullInfoCleanUp());
* };
* }, [dispatch]);
* On leaving a page (clear the state) that only used in this page.

(Because if I left the state in the global state => That affect this app memory)

* Optimization:-

1)clean state on leaving it’s page

2) prevent components (extra rerenders )

Using (

1. memo
2. useCallback
3. sending params as [props] not [children]

)

<Heading title="Your Wishlist" /> // Right

      <Heading>Your Wishlist</Heading> // Not optimized

3) Lazy loading

* (Memo – useCallback) => (Cash) the inputs and props, and each time it receives new props or inputs (it compares them to the cashed ones it has), if they are the same it never executes again and if they are different it executes again.
* (Lazy loading) => Is important because when I enter a page, it’s JS starts to load, and if the network is slow or there is any other barrier JS loads slower, so user then see a (white empty screen), so lazy is important then + (suspense) to show him a text or anything until page loads.
* (any [vs] unknown): in Ts it is better to use (unknown) because {if we use guards with it) this makes us can access (give choices) methods and properties by clicking (.) after variable name
* error: unknown
* error.
* I should stop execution of a code will navigating between page, like I am in home then navigated to categories page but before categories appear on the screen I went to products page, so then I should stop execution of getting categories from backend.
* useEffect(() => {
* const promise = dispatch(actGetProductsByCatPrefix());
* return () => {
* dispatch(cleanUpProductsRecords());
* promise.abort()
* };
* }, [dispatch, params]);

Add (signal) here while calling the api

const response = await axios.get<TResponse>(

        `/products?cat\_prefix=${payload}`,

        {

          signal,

        }

      );

I should not use this for all cases, only for examples like mentioned one.

* (types/node) => This is a package released in Ts latest updates that can be used in (give path aliases) without using other packages.
* Not there is 2 ways to use type aliases in

(Ts + vite) 🡺 1)vite-plugin-svgr , 2) types/node

* To use (types/node)

1. Vite.config.ts
2. import { defineConfig } from "vite";
3. import react from "@vitejs/plugin-react";
4. import svgr from "vite-plugin-svgr";
5. import path from "path";
6. // https://vitejs.dev/config/
7. export default defineConfig({
8. resolve: {
9. alias: {
10. "@assets": path.resolve(\_\_dirname, "./src/assets"),
11. "@components": path.resolve(\_\_dirname, "./src/components"),
12. },
13. },
14. plugins: [react(), svgr()],
15. });
16. tsConfig.app.josn
17. "paths": {
18. "@assets/\*": ["src/assets/\*"],
19. "@components/\*": ["src/components/\*"],
20. },

* TypeScript predicate
* interface Dog {
* name: string;
* guard: string;
* }
* interface Cat {
* name: string;
* numLives: string;
* }
* const isCat = (animalType: unknown): animalType is Cat => {
* return (animalType as Cat).numLives !== undefined;
* };
* console.log(isCat({ name: "test", numLives: "7" })); // true
* console.log(isCat({ name: "test", guard: "guard" })); // false
* // -----------------------------------------------------
* // -----------------------------------------------------
* const makeNoise = (animal: Dog | Cat) => {
* if (isCat(animal)) {
* return animal.numLives;
* } else {
* return animal.guard
* }
* };
* console.log(makeNoise({ name: "test", numLives: "7" })) // "7"
* // Here in the (isCat) function (I want to see if animal is cat or not)
* // 1) (: animalType is Cat) => This makes TS awair if animal is cat or not.
* // (not only return true or false without understanding)
* // 2) ((animalType as Cat).numLives !== undefined)
* // Here I choose the (unique key) in the cat object
* // If this key exist so it is true, otherwise it is false
* // So Here I made a function that
* // (returns animal type + the func itself is awair of animal type)
* // So Here I made a function that
* // (returns animal type + the func itself is awair of animal type)
* // \*\*\* So based on that, in conditions we replaces using of (true, false) with (Dog, Cat)
* Dynamic component pattern
* type LoadingProps = {
* loading: TLoading;
* error: string | null;
* children: React.ReactNode;
* type: "cart" | "product" | "category";
* };
* const skeletonsTypes = {
* category: CategorySkeleton,
* product: ProductSkeleton,
* cart: CartSkeleton,
* }; // keys names = (type) values
* const Loading = ({
* loading,
* children,
* type = "category",
* }: LoadingProps) => {
* const Component = skeletonsTypes[type];
* // points at a component in skeletonsTypes object
* if (loading === "pending") {
* return <Component />;
* }
* return <div>{children}</div>;
* };
* ((( Keyof type query ))) => TS property
* const skeletonsTypes = {
* category: CategorySkeleton,
* product: ProductSkeleton,
* cart: CartSkeleton,
* }; // keys names = (type) values
* type LoadingProps = {
* loading: TLoading;
* error: string | null;
* children: React.ReactNode;
* type?: keyof typeof skeletonsTypes;
* };
* // ((( keyof type query ))) => TS property
* // Here I made (type) gets values [dynamically] based on [skeletonsTypes key]
* // [[[ typeof skeletonsTypes ]]]:-
* // -------------------------------
* // const skeletonsTypes = {
* //   category: "CategorySkeleton",
* //   product: "ProductSkeleton",
* //   cart: "CartSkeleton",
* // };
* // keyof => Take (keys) and put them in an array
* // [category, product, cart]
* Use (Lotti files) + (Lotti react)
* Uses a (JSON file) so it is (light weight)
* dispatch(actAuthRegister({ firstName, lastName, email, password }))
* .unwrap()
* .then(() => {
* navigate("/login");
* });
* };
* // (unwrap()) => It makes sure to to execute (then())
* // except when dispatch action is done
* I should prevent navigation to login and register pages while user is loggedin.
* if (accessToken) {
* return <Navigate to="/" />;
* }
* If I have a reusable component that I want to use to display (two different shapes), these shapes have (similarities, differences)

{{{

1) similarities => send as (props)

2) differences => send as (children)

}}}

* Make text like (text inf…), only inside it’s parent.
* white-space: nowrap;
* overflow: hidden;
* text-overflow: ellipsis;

Error got in (catch) is the error returned from the (redux)

dispatch(actPlaceOrder(subtotal))

      .unwrap()

      .then(() => {

        dispatch(clearCartAfterPlaceOrder());

        setShowModal(false);

      })

      .catch((error) => {

        setError(error);

      })

      .finally(() => setLoading(false));

  };

Store slice:-

builder.addCase(actPlaceOrder.rejected, (state, action) => {

      state.loading = "failed";

      if (isString(action.payload)) {

        state.error = action.payload;

      }

    });

* const [showModal, setShowModal] = useState(false);
* const [selectedProduct, setSelectedProduct] = useState<TProduct[]>([]);
* const viewDetailsHandler = (id: number) => {
* const productDetails = orderList.find((order) => order.id === id);
* const newItems = productDetails?.items ?? [];
* setShowModal(true);
* setSelectedProduct((prev) => [...prev, ...newItems]);
* };
* // ################################################
* // ### Manage all logic in the parent component ###
* // ################################################
* // Manage the logic of showing the Modal of order details in the parent component
* // Manage the rendered list of order details in the parent
* // Send all that as props to the child component (order details Modal)

Commands:-

npm i typescript -g

npm create vite@latest

npm install

npm i axios

npm i react-router-dom

npm i react-redux

npm i @reduxjs/toolkit

npm i react-bootstrap

npm i bootstrap

npm i vite-plugin-svgr

npm i vite-tsconfig-paths

npm i redux-persist

npm i -D @types/node

npm i react-content-loader

npm i lottie-react

npm install react-hook-form

npm i zod

npm i @hookform/resolvers

* npm init - npm i json-server
* npm i json-server-auth

Technologies:-

React

Typescript

Axios

React-router-dom

React-redux

Redux toolkit

React bootstrap

Bootstrap

vite-plugin-svgr

vite-tscongig-paths

redux-persist

types/node

react-content-loader

lottie-react

react-hook-form

zod

@hookform/resolvers

-- json-server

-- json-server-auth

Questions:-

1. What is (SWC), what is the difference between it and (Babel)