Monorepo

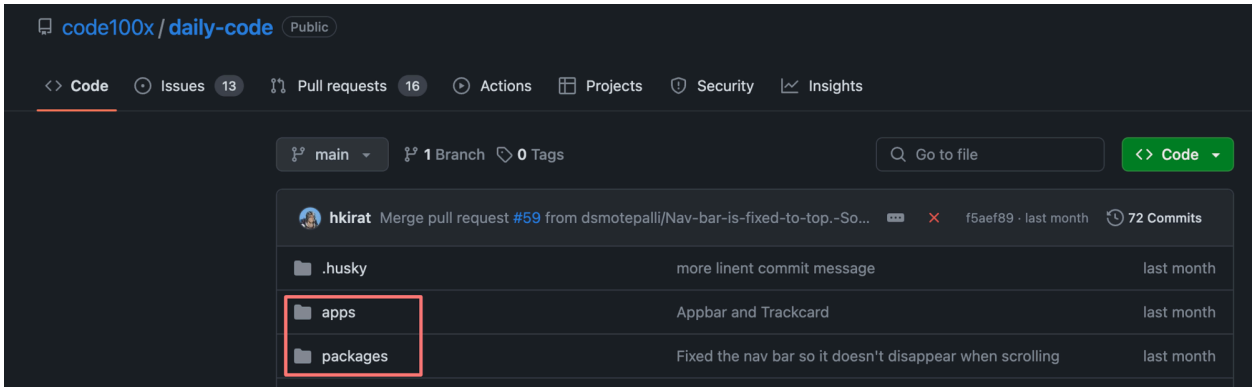
(https://projects.100xdevs.com/tracks/monorepo/monorepo-1)

What are monorepos

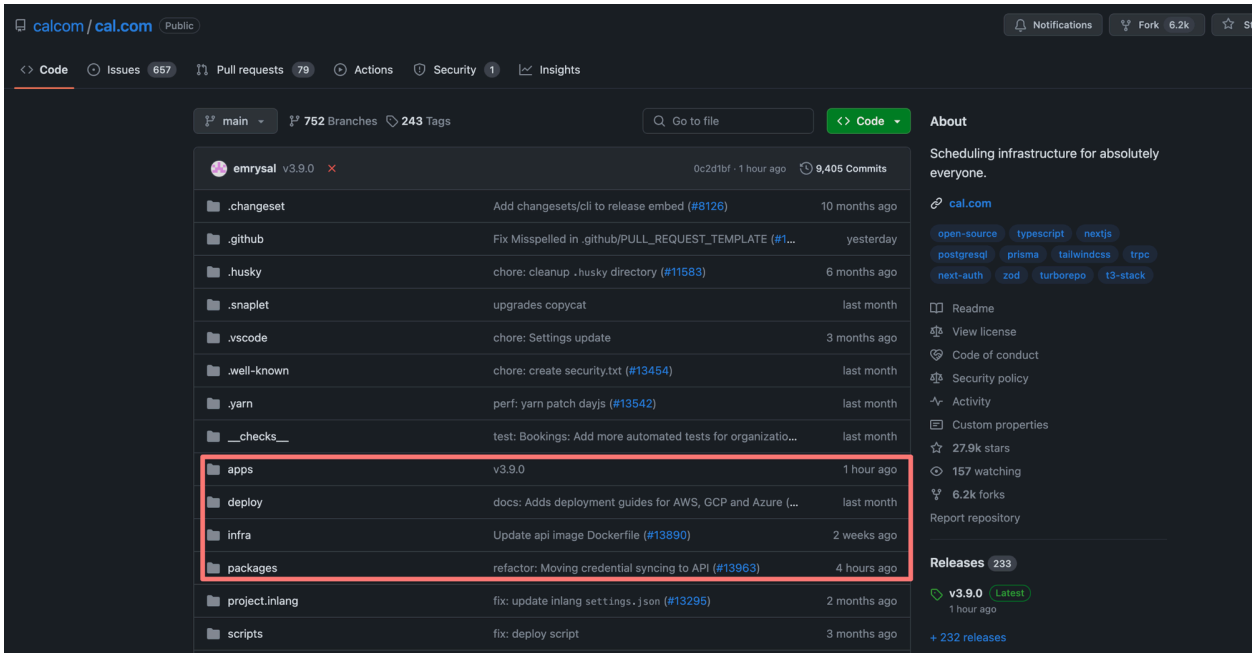
A single repository (on github lets say) that holds all your frontend , backend and devops code.

Few repositories that use monorepos are –

1. https://github.com/code100x/daily-code



2. https://github.com/calcom/cal.com



apps and packages are popular folders and will have our top level code , frontend next js code , backend node js code.

OBJ

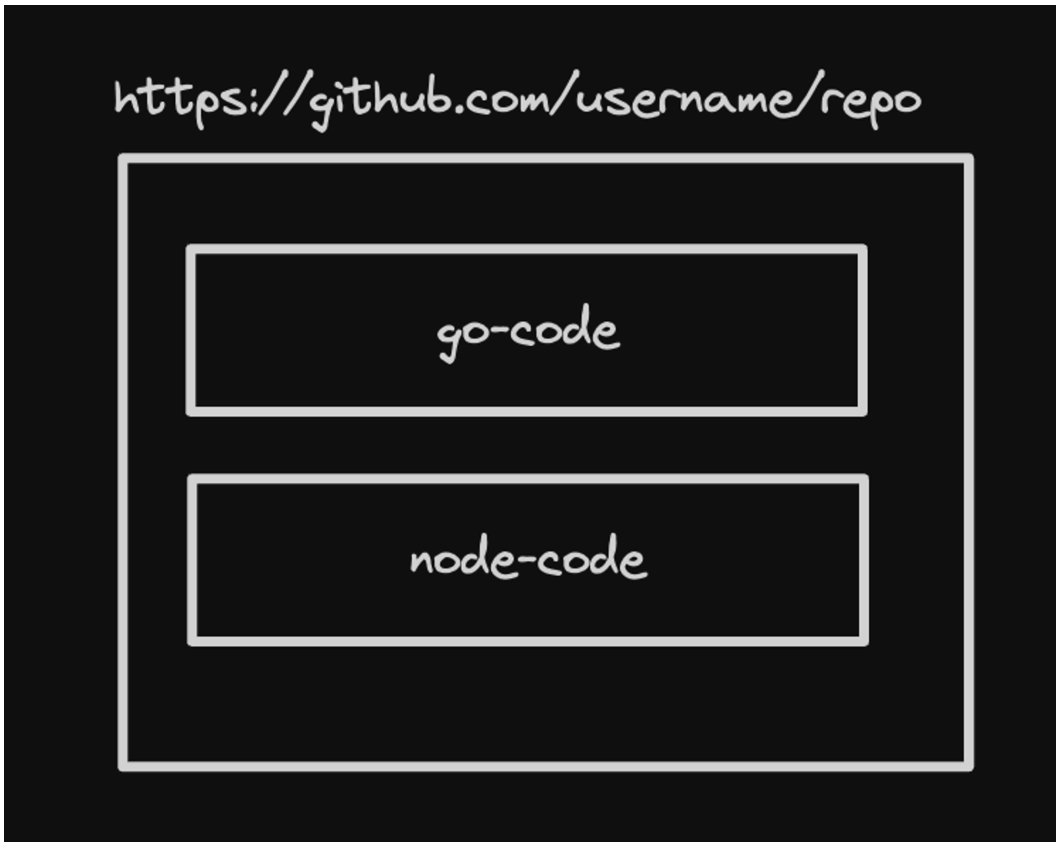
Why Monorepos??

Why not simple folders?

Why cant I just store services (backend, frontend etc) in various top level folders? You can, and you should if your

1. Services are highly decoupled (dont share any code)
2. Services don't depend on each other.

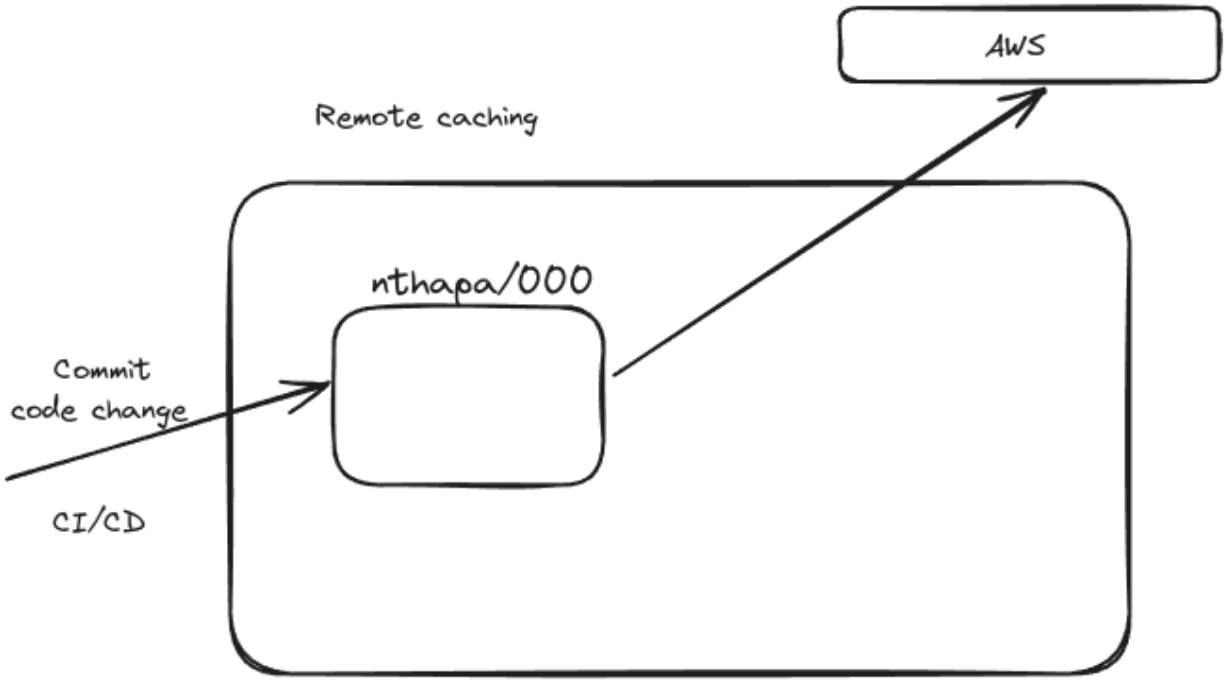
For eg - A codebase which has a Golang service and a JS service



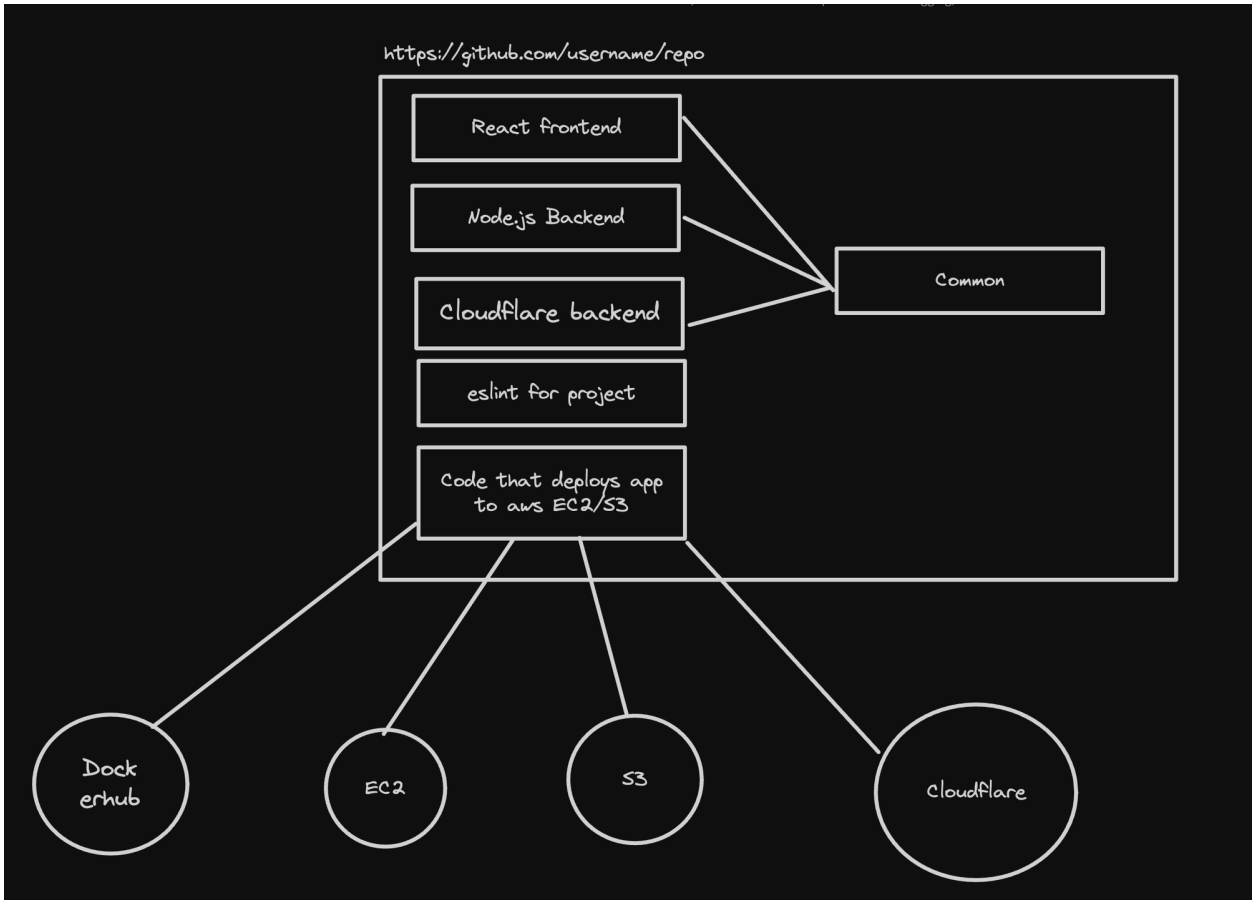
Why monorepos??

1. **Shared Code Reuse**

2. **Enhanced Collaboration**
3. **Optimized Builds and CI/CD:** Tools like TurboRepo offer smart caching and task execution strategies that can significantly reduce build and testing times.



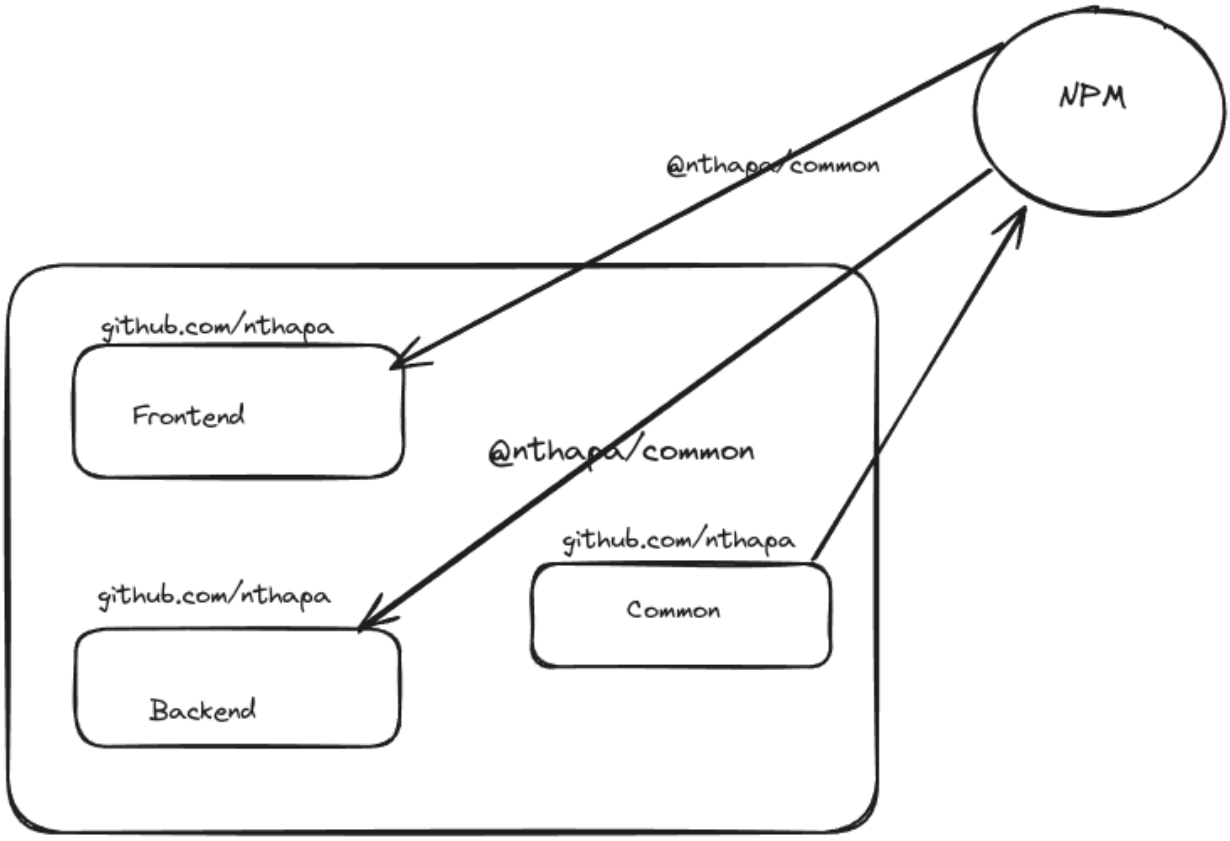
4. **Centralized Tooling and Configuration:** Managing build tools, linters, formatters, and other configurations is simpler in a monorepo because you can have a single set of tools for the entire project.



High probability that they need to share something example zod types

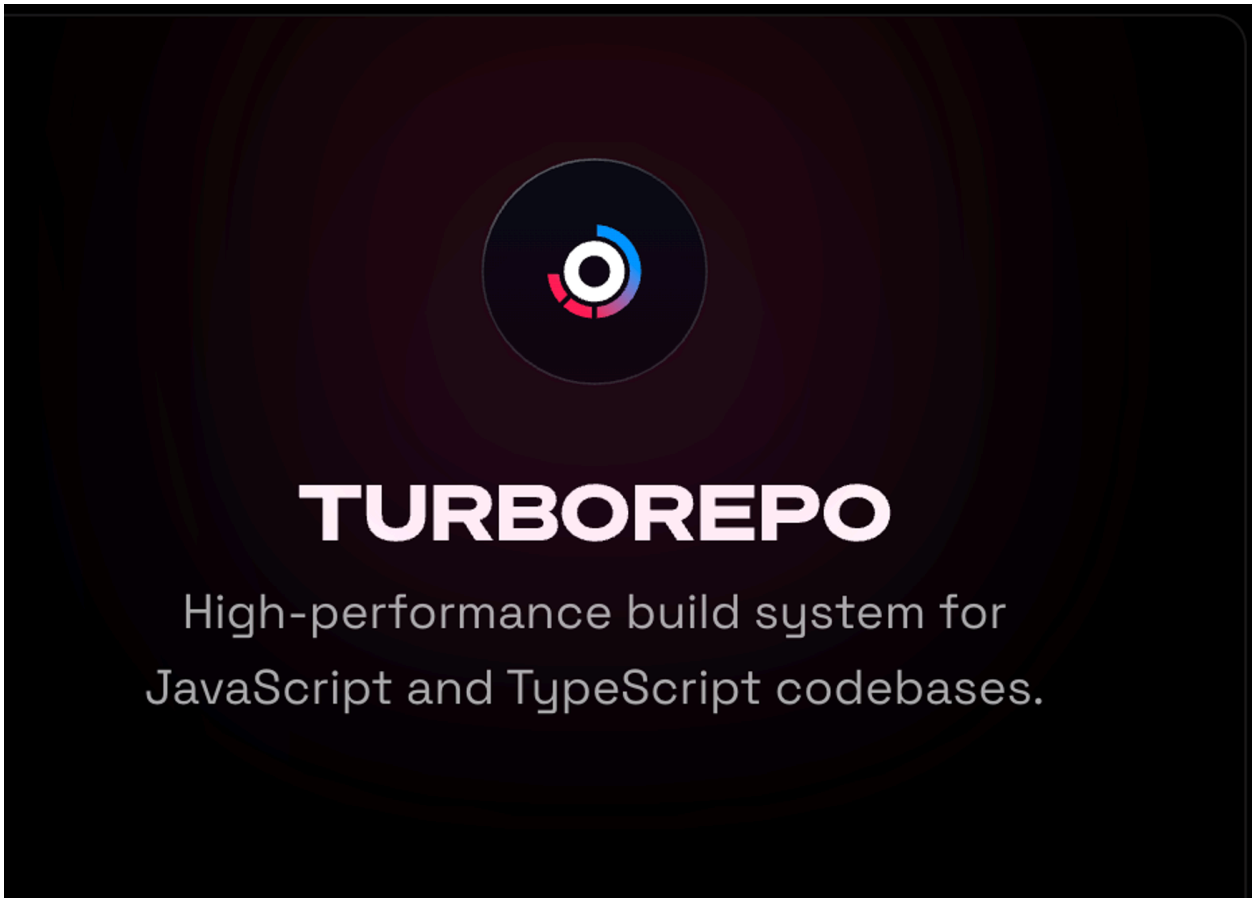
Monorepos is what lets us share the code and import modules.

Let see how we did before Monorepo



Common monorepo framework in Node.js

1. Lerna - https://lerna.js.org/
2. nx - https://github.com/nrwl/nx 1,2,4 are monorepo framework
3. Turborepo - https://turbo.build/ — Not exactly a monorepo framework, monorepos on steroids
4. Yarn/npm workspaces - https://classic.yarnpkg.com/lang/en/docs/workspaces/



History of Turborepo

1. Created by Jared Palmer
2. In December 2021 Acquired/aqui-hired by Vercel
3. Mild speculation/came from a random source - Pretty hefty dealp
4. They’ve built a bunch of products, Turborepo is the most used one

Build system vs Build system orchestrator vs Monorepo framework

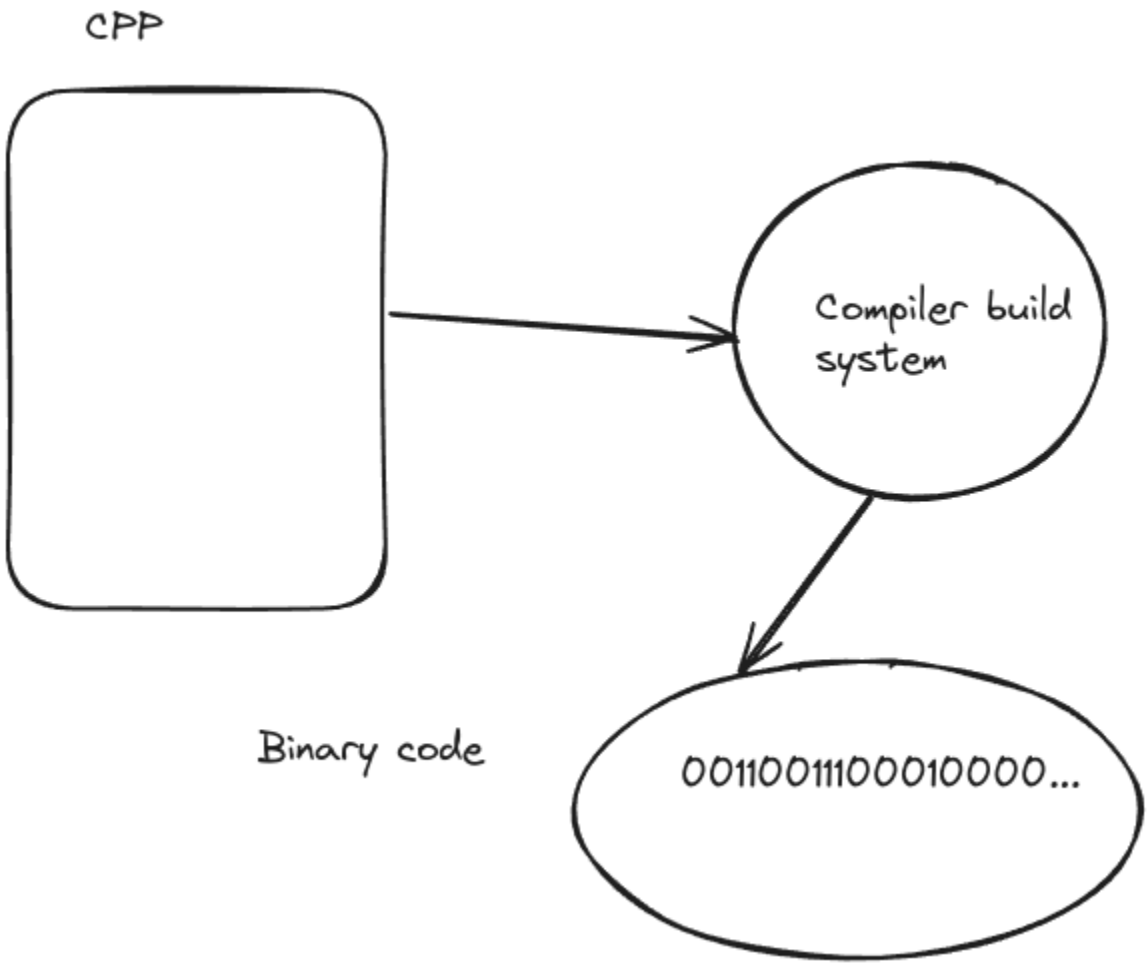
We will be able to differentiate between turborepo and monorepo framework

Build System

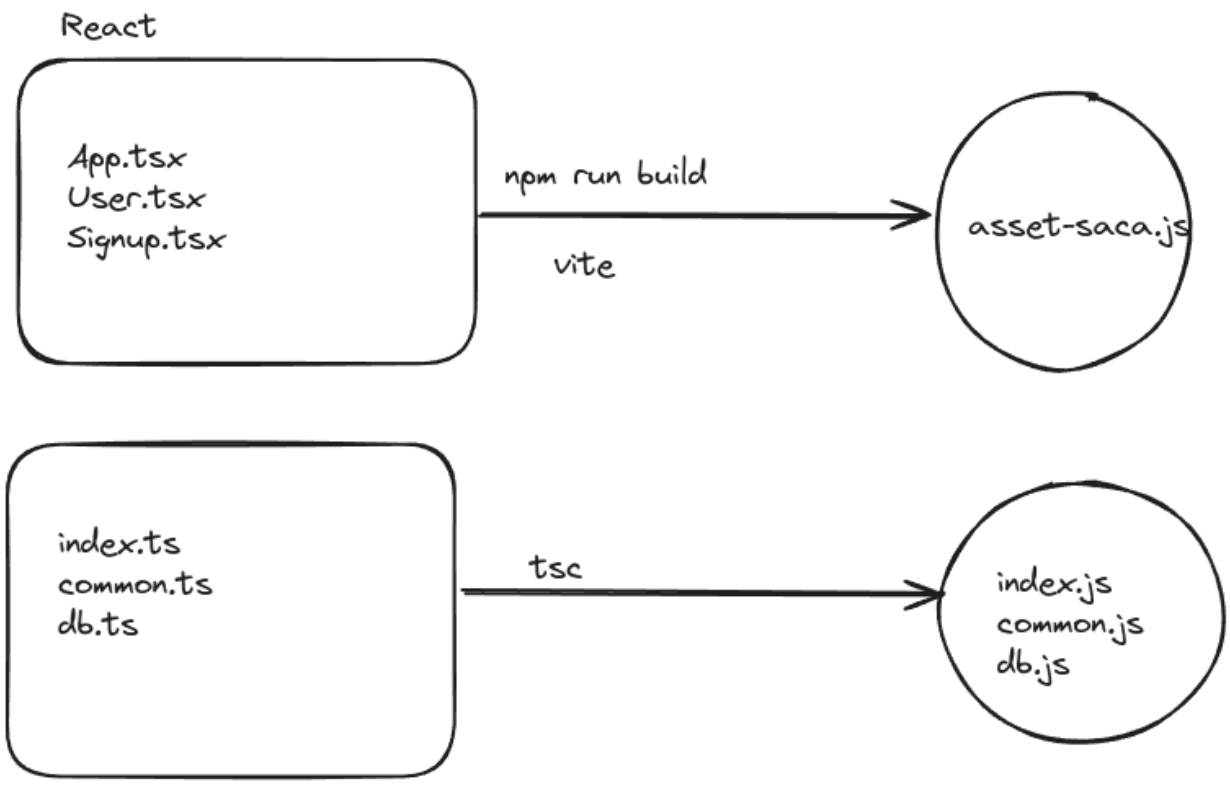
A build system automates the process of transforming source code written by developers into binary code that can be executed by a computer. For JavaScript

and TypeScript projects, this process can include transpilation (converting TS to JS), bundling (combining multiple files into fewer files), minification (reducing file size), and more. A build system might also handle running tests, linting, and deploying applications.

Let take example of C++:: g++ is good example of build system



Lets see for Javascript



Build System Orchestrator

TurboRepo acts more like a build system orchestrator rather than a direct build system itself. **It doesn't directly perform tasks like transpilation, bundling, minification, or running tests.** Instead, TurboRepo **allows you to define tasks in your monorepo that call other tools** (which are the actual build systems) to perform these actions.

These tools can include anything from **tsc, vite** etc

Dependency management during builds , Caching during builds. If backend hasn’t changed since last build it will cache it from the last build and this will decrease the net build time.

Monorepo Framework

A monorepo framework provides tools and conventions for managing projects that contain multiple packages or applications within a single repository

(monorepo). This includes **dependency management** between packages, workspace configuration.

Monorepo framework has nothing to do with build system

import {zodType} from “common”

Turborepo as a build system orchestrator

Turborepo is a **build system orchestrator** .

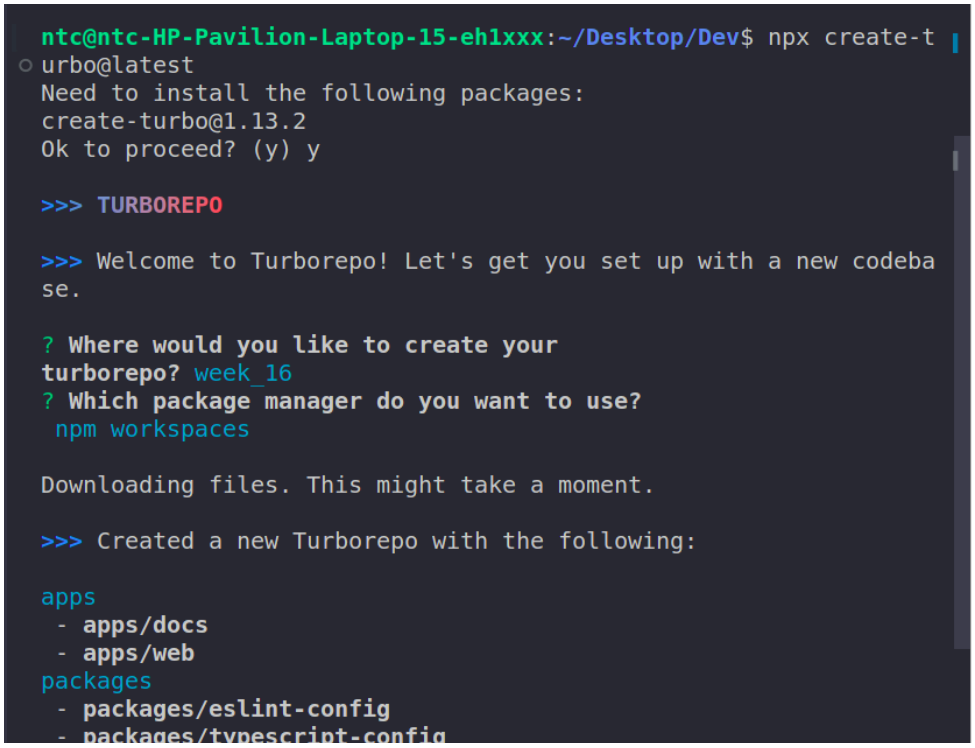
The key feature of TurboRepo is its ability to manage and optimize the execution of these tasks across your monorepo. It does this through:

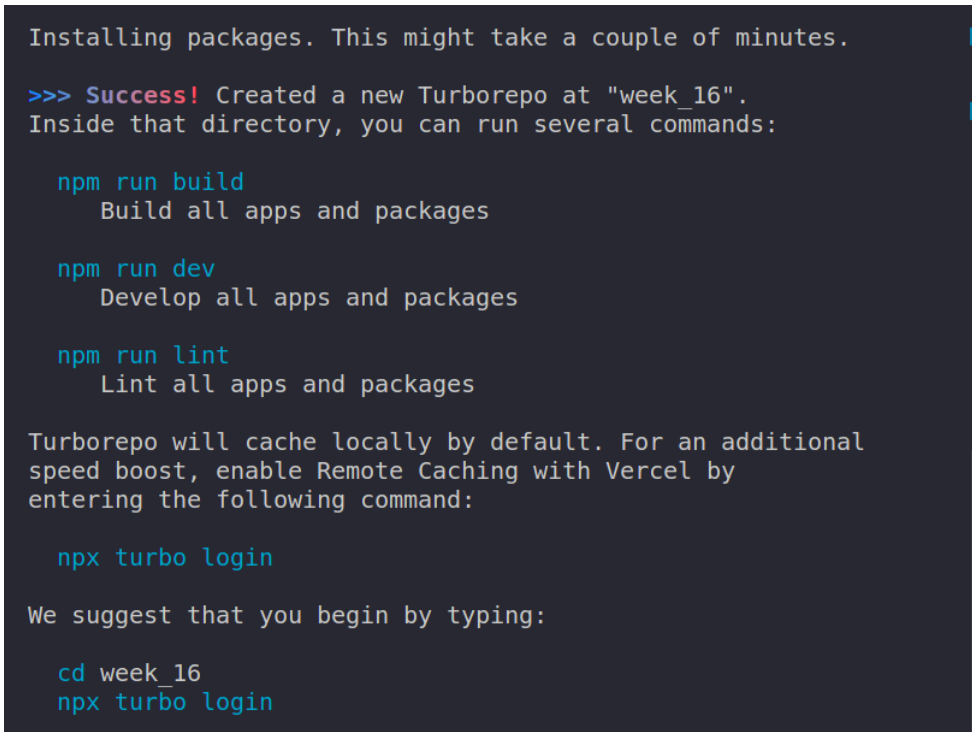
1. **Caching**: TurboRepo caches the outputs of tasks, so if you run a task and then run it again without changing any of the inputs (source files, dependencies, configuration), TurboRepo can skip the actual execution and provide the output from the cache. This can significantly speed up build times, especially in continuous integration environments.
2. **Parallelization**: It can run independent tasks in parallel, making efficient use of your machine's resources. This reduces the overall time needed to complete all tasks in your project.
3. **Dependency Graph Awareness**: TurboRepo understands the dependency graph of your monorepo. This means it knows which packages depend on each other and can ensure tasks are run in the correct order.

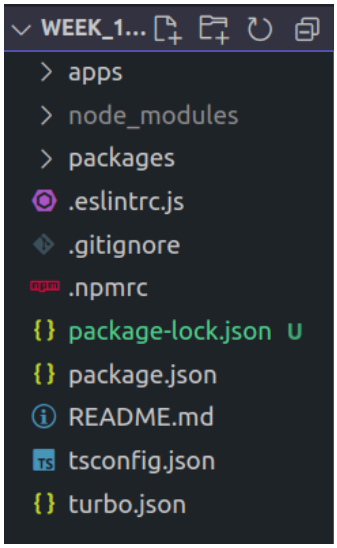
Let’s initialize a simple Turborepo

https://turbo.build/repo/docs

npx create-turbo@latest

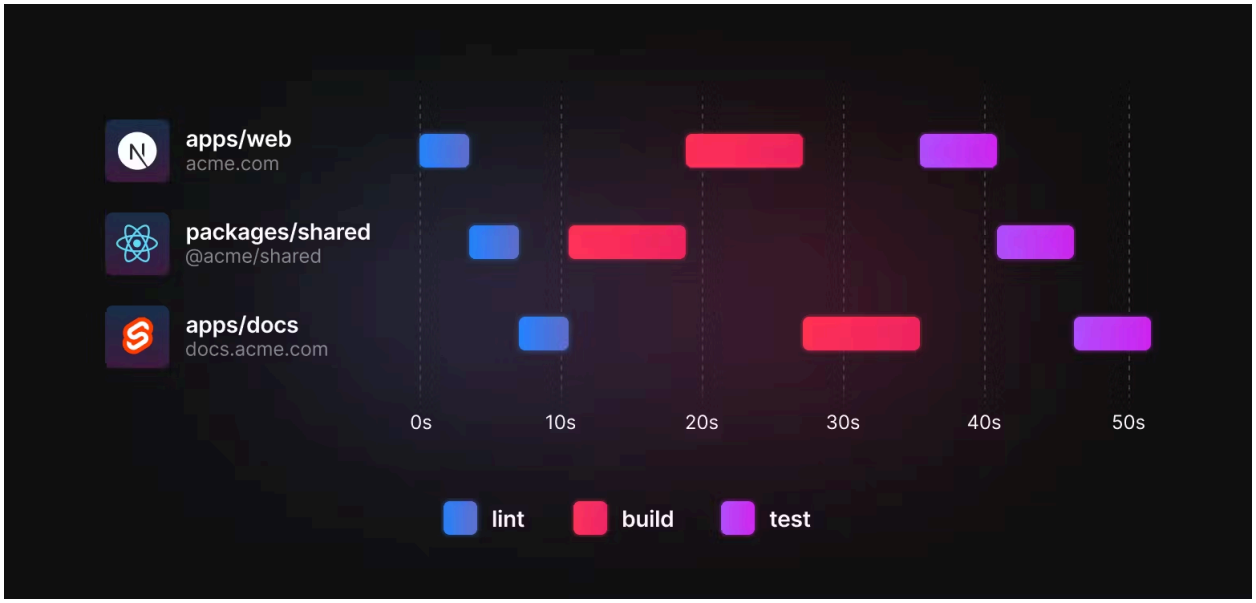






Turbo repo with NextJs and express node make sense.

Running in conventional way



This is the slowest possible way to run these tasks. Each task needs to wait for the previous one to finish before it can start. To improve on this, we'll need a tool that can multitask.

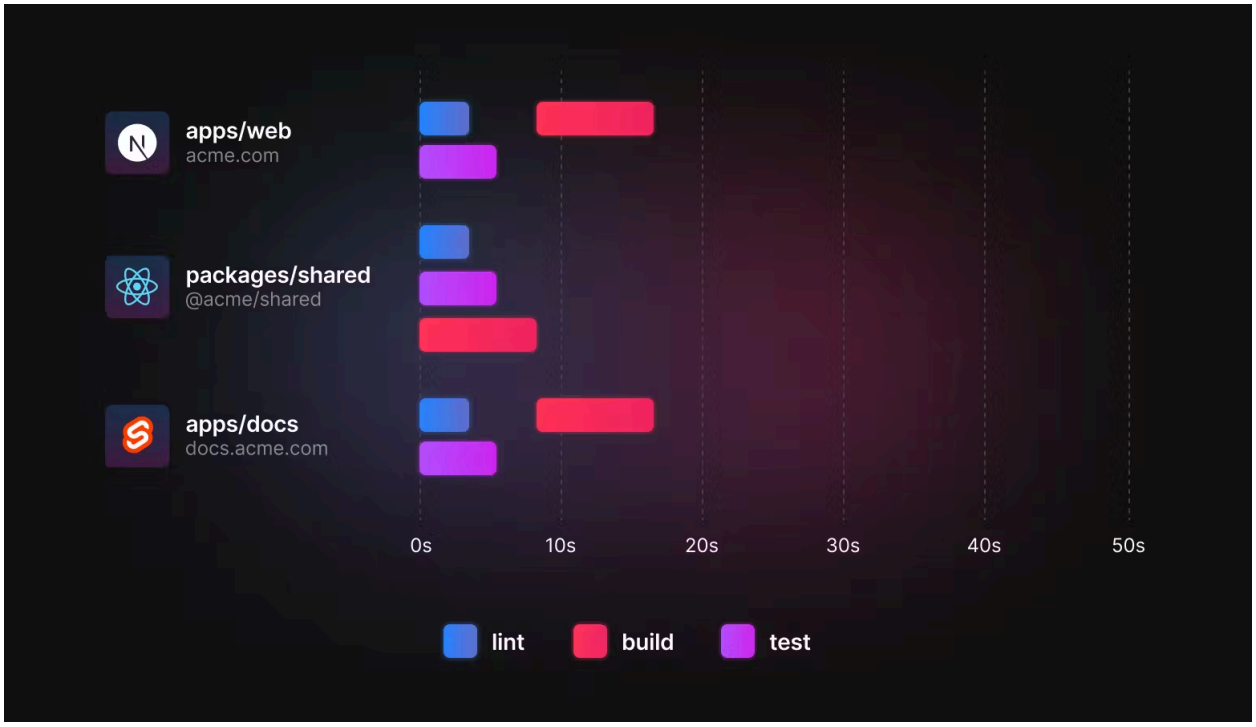
Turborepo will multitask

Turborepo can schedule our tasks for maximum speed by understanding the dependencies between our tasks.

turbo.json

```
{
 "$schema": "https://turbo.build/schema.json",
 "globalDependencies": ["**/.env.*local"],
 "pipeline": {
   "build": {
     "dependsOn": ["^build"],
     "outputs": [".next/**", "!.next/cache/**"]
   },
   "lint": {
     "dependsOn": ["^lint"]
   },
   "dev": {
     "cache": false,
     "persistent": true
   }
 }
}
```

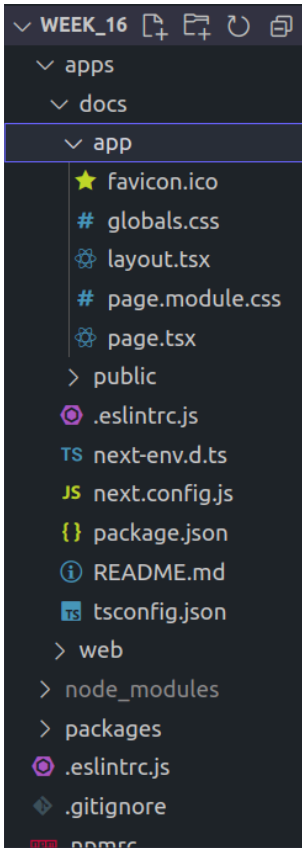
When we run it , Turborepo will multitask as many tests as possible over all available CPU’s meaning our task runs like this:



Both `lint` and `test` run immediately, because they have no `dependsOn` specified in `turbo.json`.

The `build` task inside `shared` completes first, then `web` and `docs` build afterwards.

Folder Structure



This is Next JS application , we can add our express server here.

There are 5 modules in our project

End user apps (websites/core backend)

1. apps/web - A Next.js website
2. apps/docs - A Docs website that has all the documentation related to your project

Helper packages

1. packages/ui - UI packages (common folder in medium website, example type of api endpoint , Small mini functionalities, sharing react component using ui folder

   type UserSignup(){
   username: string;
   password: string;

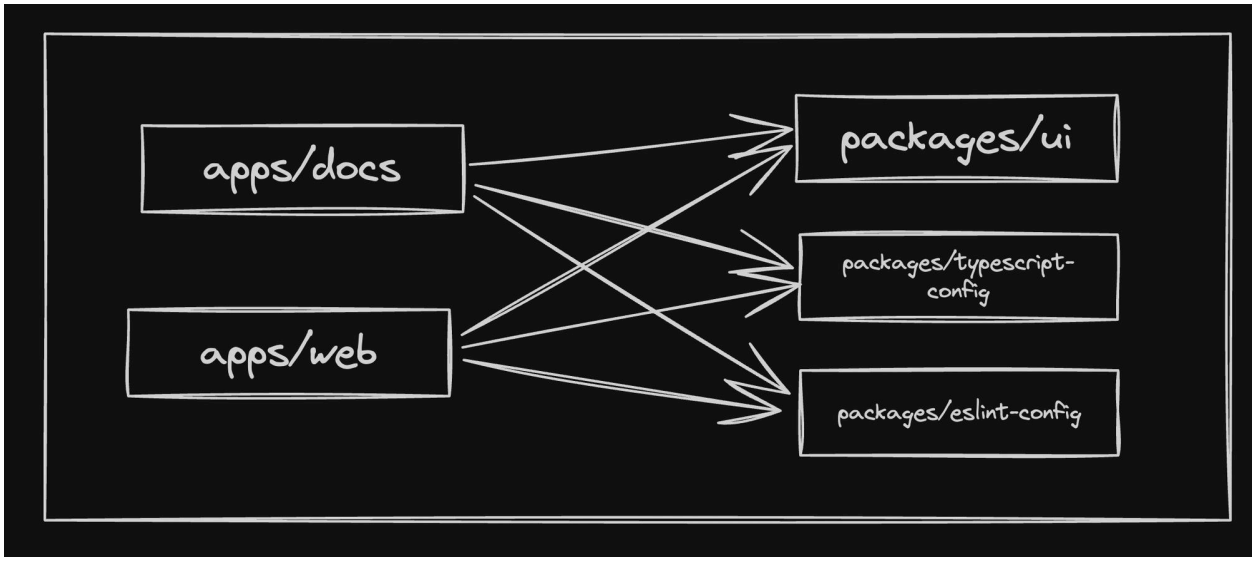
}

app.post(“/signup”,(req,res)=> {

}))

Has component which will be shared eg button component, it is our own ui library Example **dub.sh github**

2. packages/typescript-config - Shareable TS configuration
3. packages/eslint-config - Shareable ESLine configuration



Run the project

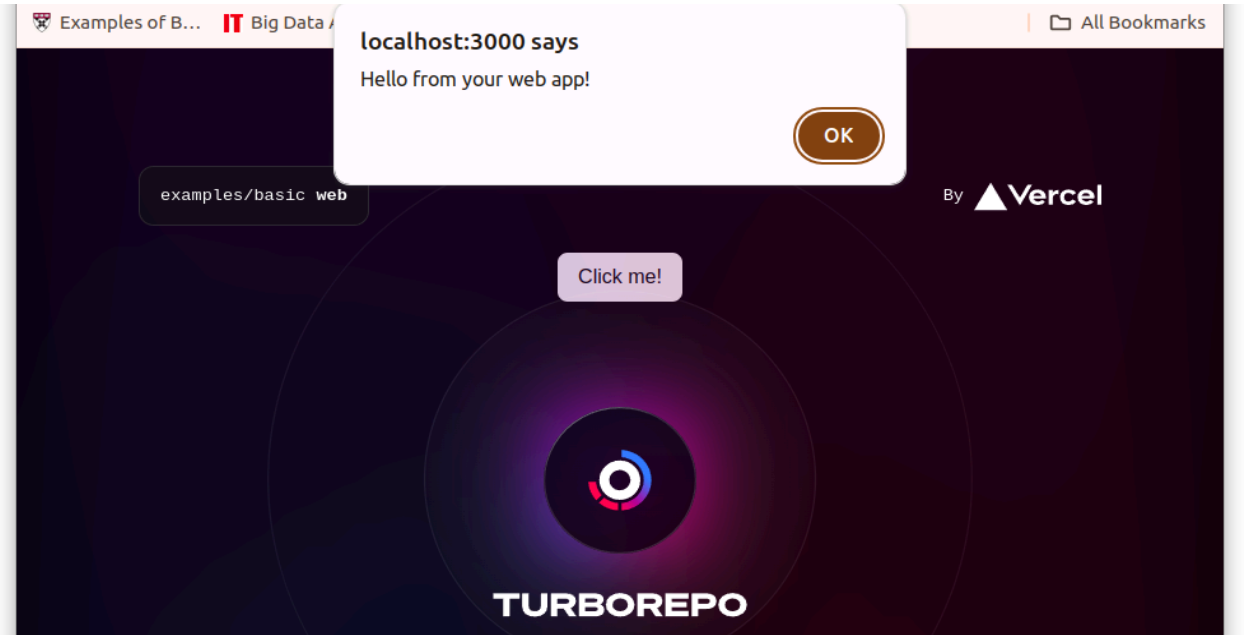
Root folder **, npm run dev**

Upgrade node.js version

```
web:dev:
web:dev: You are using Node.js 18.1.0. For Next.js, Node.js version >= v18.17.0 is required.
docs:dev: You are using Node.js 18.1.0. For Next.js, Node.js version >= v18.17.0 is required.
web:dev: ERROR: command finished with error: command (/Users/kirat/Projects/project/apps/web)
```

You will notice two websites running on

Localhost:3000



This is a NextJs application

localhost:3001

This means we have a single repo which has multiple projects which share code from packages/ui

npm run dev

Lets see the package.json file in root directory and dev key inside it

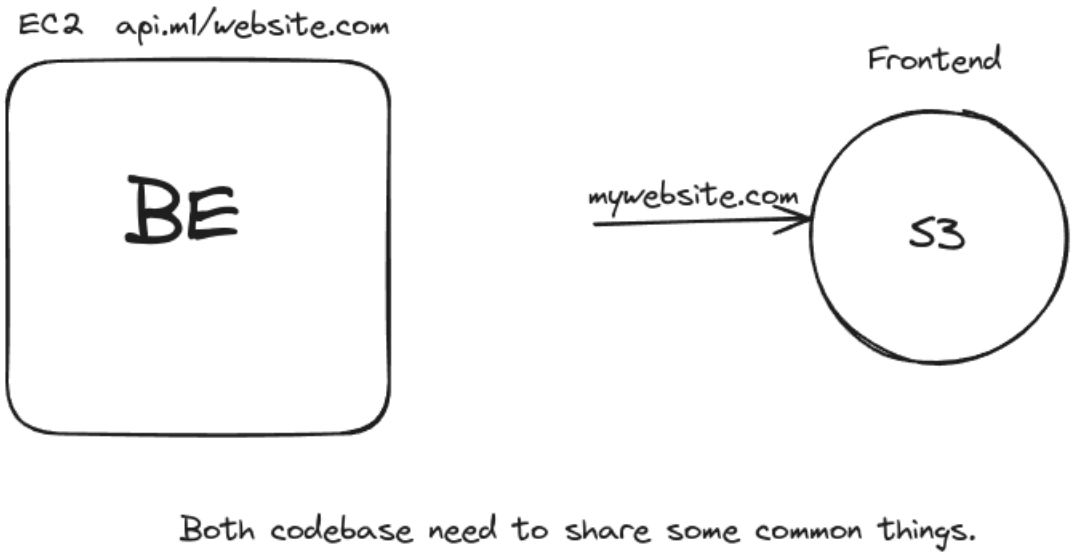
{} package.json > {} scripts

```
{
  "name": "week_16",
  "private": true,
  ▷ Debug
  "scripts": {
    "build": "turbo build",
    "dev": "turbo dev",
    "lint": "turbo lint",
    "format": "prettier --write \"**/*.{ts,tsx,md}\""
  },
```

Now in each folder of app and check its package. Json for dev.

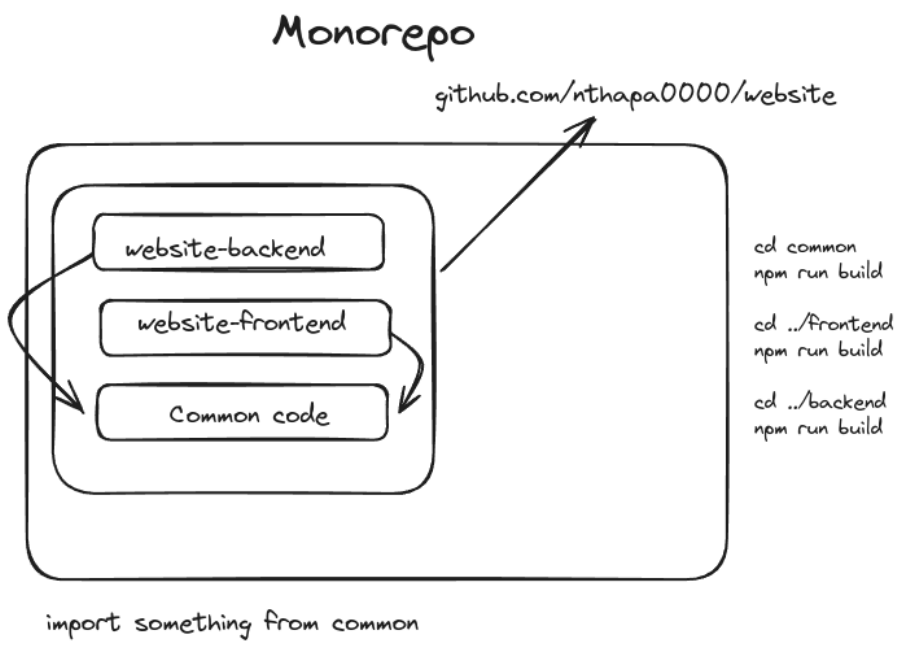
turbo is build system orchestrator.

Make sure all the individual app run on seperate ports



Both codebase need to share some common things.

Monorepo



Lets see how Turborepo will do

turbo build

(will build things faster)

Exploring root package.json

scripts

This represents what command runs when you run

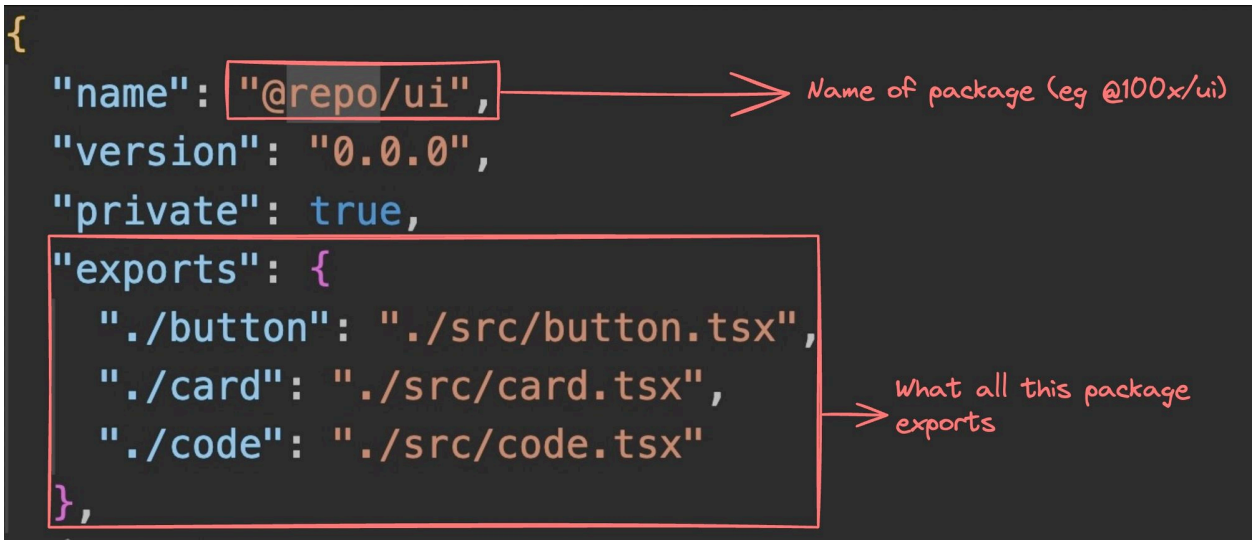
1. npm run build
2. npm run dev
3. npm run lint

turbo build goes into all packages and apps and runs **npm run build** inside them (provided they have it)

Same for dev and lint

Exploring packages/ui

1. package.json



In case we publish to npm / prefixed by username

If someone import from @repo/ui/button they will get the button.tsx component

We can also do is something like this

“./”:”./src/index.ts”

index.ts

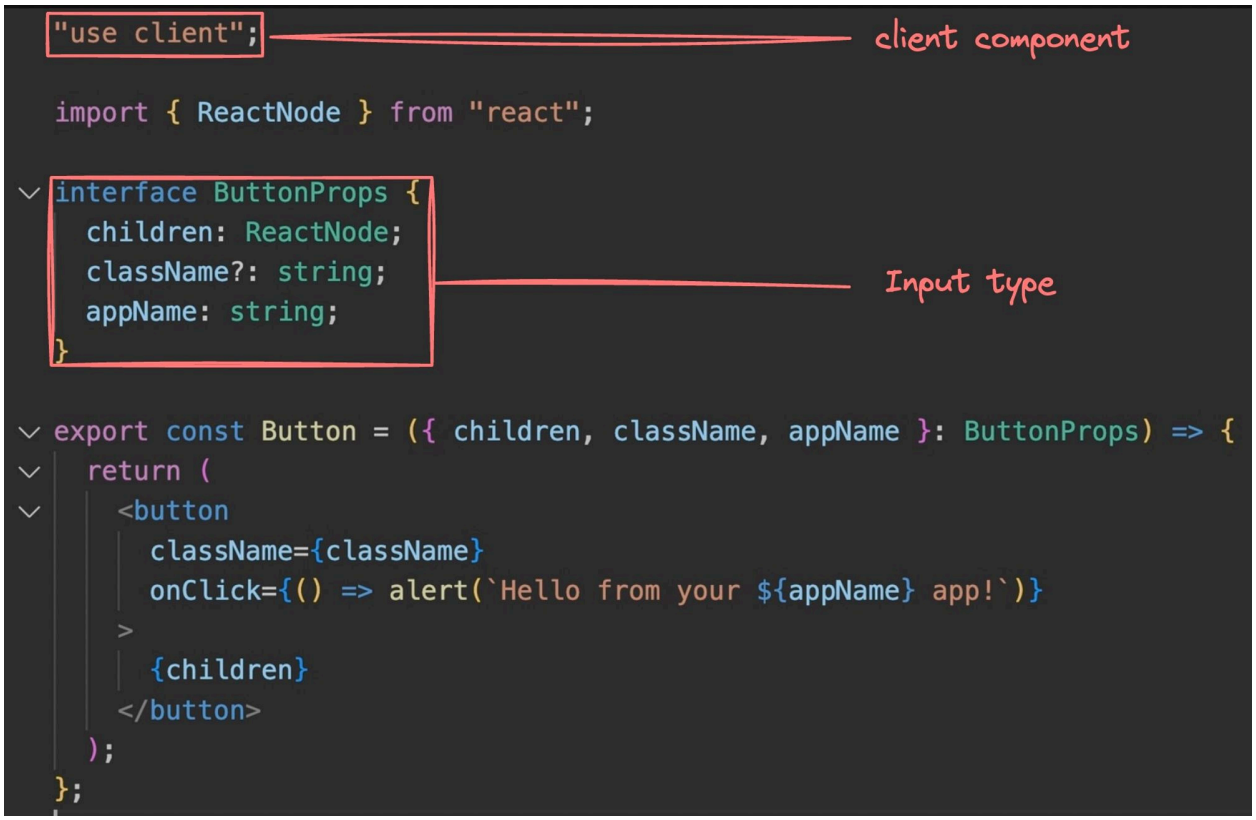
export * from ‘./card’

export * from ‘./code’

```
"@repo/ui": "*",
```

Find it somewhere here

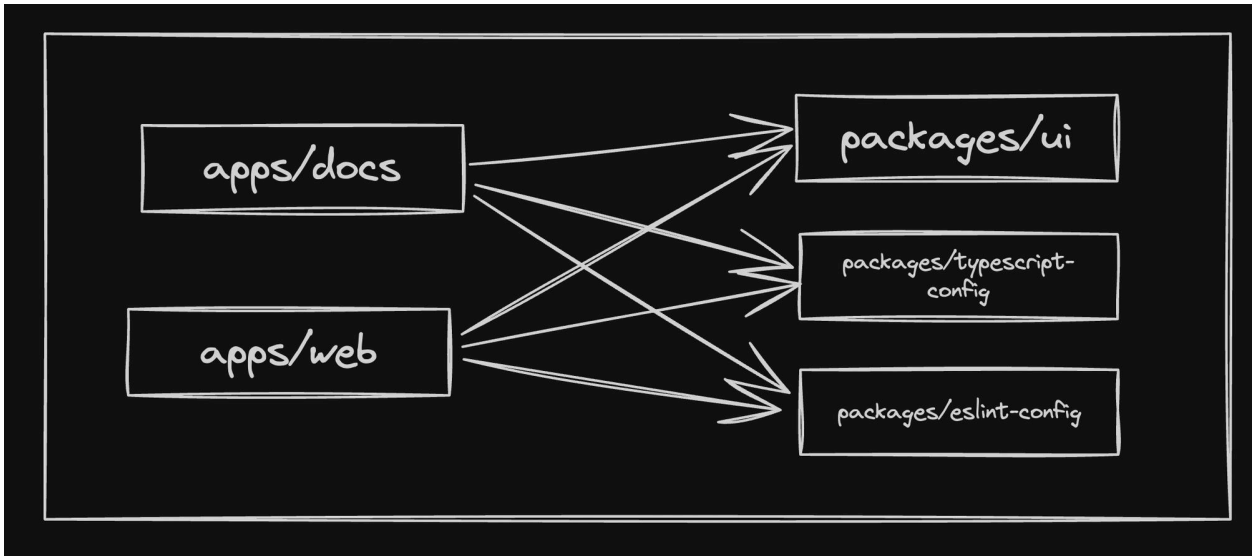
2. src/button.tsx



Exploring apps/web

1. **Dependencies**

It is a simple Next.js app. But it uses some UI components from the packages/ui module



2. **Exploring package.json**

If you explore package.json of apps/web, you will notice @repo/ui as a dependency

```
"dependencies": {
  "@repo/ui": "*",
  "next": "^14.1.1",
  "react": "^18.2.0",
  "react-dom": "^18.2.0"
},
```

3. **Exploring page.tsx**

Lets see import and usage of the button component

```
import Image from "next/image";
import { Card } from "@repo/ui/card";
import { Code } from "@repo/ui/code";
import styles from "./page.module.css";
import { Button } from "@repo/ui/button";
```

Import from packages module

```
<Button appName="web" className={styles.button}>
  Click me!
</Button>
```

The same Button component can be used by the apps/docs website as well

Let’s add a new page

Cleaning app -> web -> app -> page.tsx

```
import { Button } from "@repo/ui/button";

export default function Page(): JSX.Element {
 return (
     <div>
       <Button appName="Web app">
           Hi there
       </Button>
     </div>
 );
}
```

Adding admin.tsx in packages -> ui -> src

```
export function Admin(){
   return <div>
       Admin Page
       {/* not much reusable page */}
   </div>
}
```

Another step so that other modules can use add export in package.json

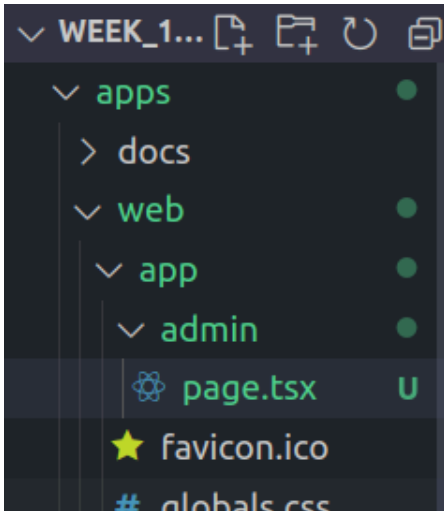
packages > ui > {} package.json > ...

```
{
  "name": "@repo/ui",
  "version": "0.0.0",
  "private": true,
  "exports": {
    "./button": "./src/button.tsx",
    "./card": "./src/card.tsx",
    "./code": "./src/code.tsx",
    "./admin":"./src/admin.tsx"
  },
```

But this looks ugly

Turbo repos has something called **generators**

File based routing for admin page




```
import { Admin } from "@repo/ui/admin";

export default function(){
   return <div>
       hi from the admin Page
       <Admin/>
   </div>
}
```

We define ui component once and reuse it in different frontend.

Lets explore turbo folder under ui

Lets see the package.json inside the ui folder

```
"scripts": {
  "lint": "eslint . --max-warnings 0",
  "generate:component": "turbo gen react-component"
},
```

This generate component basically creates a file for us in source.tsx and append to package.json

turbo -> package. json

```
import type { PlopTypes } from "@turbo/gen";

// Learn more about Turborepo Generators at
https://turbo.build/repo/docs/core-concepts/monorepos/code-generation

export default function generator(plop: PlopTypes.NodePlopAPI): void {
 // A simple generator to add a new React component to the internal UI
library
 plop.setGenerator("react-component", {
   description: "Adds a new react component",
   prompts: [
     {
       type: "input",
       name: "name",
```

```
      message: "What is the name of the component?",
    },
  ],
  actions: [
    {
      type: "add",
      path: "src/{{kebabCase name}}.tsx",
      templateFile: "templates/component.hbs",
    },
    {
      type: "append",
      path: "package.json",
      pattern: /"exports": {(?<insertion>)/g,
      template: '"./{{kebabCase name}}": "./src/{{kebabCase
name}}.tsx",',
    },
  ],
 });
}
```

We have to go to the write path and do

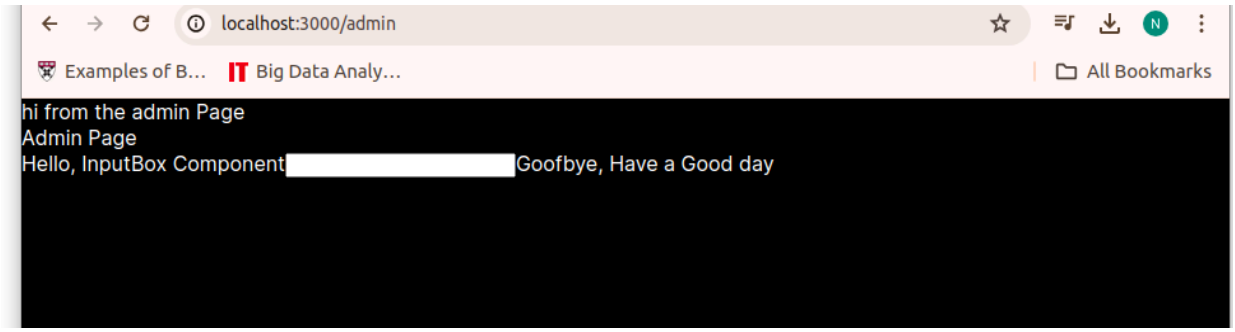
npm run generate:component

```
ntc@ntc-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Dev/week_16.2$ cd packages/ui
ntc@ntc-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Dev/week_16.2/packages/ui$ npm run generate:component

> @repo/ui@0.0.0 generate:component
> turbo gen react-component


>>> Modify "week_16.2" using custom generators

? What is the name of the component? InputBox
>>> Changes made:
  • /src/input-box.tsx (add)
  • /package.json (append)

>>> Success!
```

Any time we want to generate a component in ui we will use this command

input-box.tsx

```tsx
export const InputBox = () => {
 return (
   <div>
     Hello, InputBox Component
     <input type="text" />
     Goofbye, Have a Good day
   </div>
 );
};
```



Exploring turbo.json

Reference:

```json
{
 "$schema": "https://turbo.build/schema.json",
 "globalDependencies": ["**/.env.*local"],
 "pipeline": {
   "build": {
     "dependsOn": ["^build"],
     "outputs": [".next/**", "!.next/cache/**"]
   },
   "lint": {
     "dependsOn": ["^lint"]
   },
   "dev": {
     "cache": false,
     "persistent": true
```

```
    }
  }
}
```

Here we have said that turbo has three pipeline

build

lint

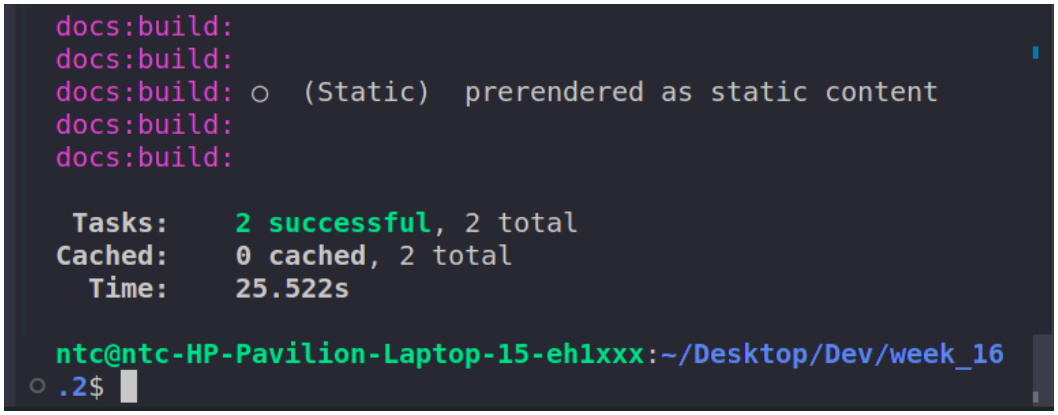
dev

This file consist of the configuration of the turborepo

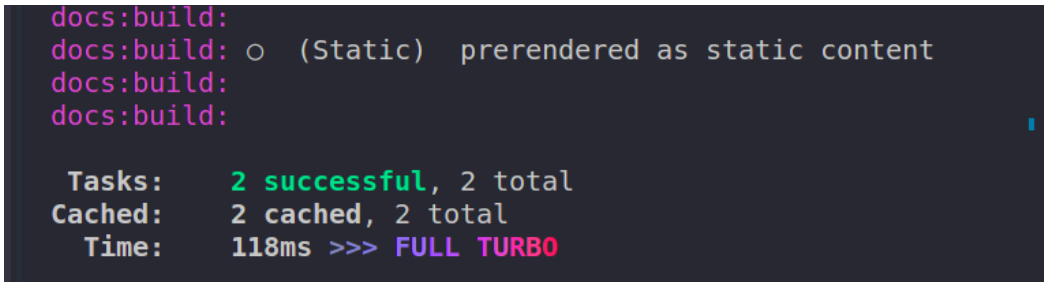
Turborepo is build orchestrator

If ui had build script then it would have build it first, so that web and docs folder can use it

Lets run npm run build and see how turborepo cache



Lets again run the same command




```
  "outputs": [".next/**", "!.next/cache/**"]
```

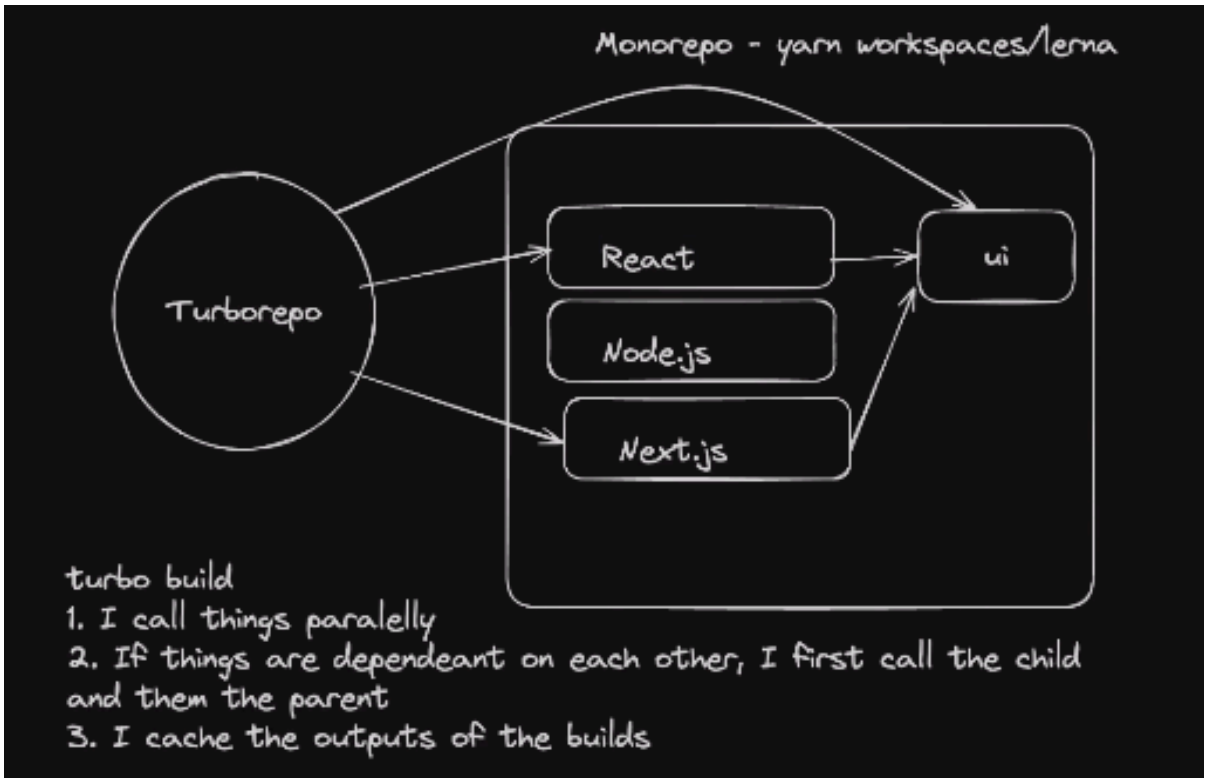
Final asset for the nextJs application is here

In dev we don’t want caching

persistent : true

This is long running process

Nothing allow to depend on docs and web



Adding React Projects

- cd apps
- Add a new react project
- npm create vite@latest
- cd ..
- npm i (global npm install)
- npm run dev

```
  web:dev:
  docs:dev:
  docs:dev: > docs@1.0.0 dev
  docs:dev: > next dev --port 3001
  docs:dev:

  react-app:dev:   VITE v5.2.8  ready in 332 ms
  react-app:dev:
  react-app:dev:   ➜  Local:   http://localhost:5173/
  react-app:dev:   ➜  Network: use --host to expose
  react-app:dev:   ➜  press h + enter to show help
  docs:dev:    ▲ Next.js 14.2.1
  web:dev:    ▲ Next.js 14.2.1
  web:dev:    - Local:        http://localhost:3000
  web:dev:
  docs:dev:    - Local:        http://localhost:3001
  docs:dev:
  web:dev:   ✓ Starting...
  docs:dev:   ✓ Starting...
  docs:dev:   ✓ Ready in 3.5s
  web:dev:   ✓ Ready in 3.5s
```

How can we use the existing ui components in the react app?

```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import {Button} from "@repo/ui/button";

function App() {
 const [count, setCount] = useState(0)

 return (
   <>
     <div>
       <Button appName='React-app'>
         Hey there
       </Button>
     </div>
   <>
)
```

We didn’t added @repo/ui in dependencies in react-app/package. json

@repo/ui already exist in root node_modules

We must add it to package.json

Adding a Node.Js application

apps -> backend

cd backend

It can be a websocket backend

npm init -y

npx tsc –init

Lets see the tsconfig.json file inside the backend

```
{
 "extends":"@repo/typescript-config/base.json",
 "compilerOptions": {
   "lib":["ES2016"],
   "module": "CommonJS",
   "outDir": "./dist",
"rootDir": "./src"
 },
 "exclude": ["node_modules"],
 "include": ["."]
}
```

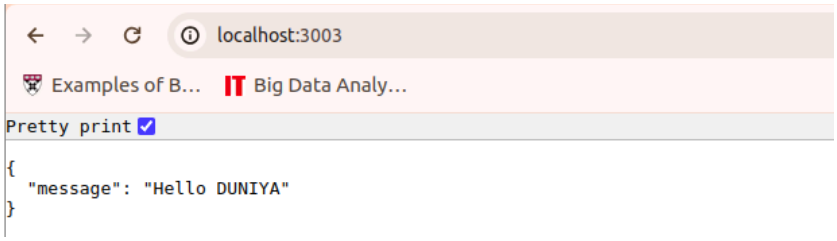
Here we can see that it has extended from the packages -> type-config

npm i express @types/express

Add your build script lint script to package. json inside backend

```
{
 ..........
 "scripts": {
   "test": "echo \"Error: no test specified\" && exit 1",
   "build":"tsc -b",
   "dev":"tsc -b && node dist/index.js"
 },
 ..........
}
```

npm run build
node dist/index.js



Common module

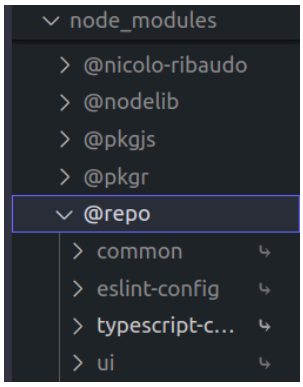
npm init -y
npm i zod
Storing zod types
Whenever you add a new package make sure to run a global npm install

package.json

```
{
 "name": "@repo/common",
 "version": "1.0.0",
 "description": "",
 "main": "index.js",
 "exports":{
   "./config":"./src/index.ts"
 },
```

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
},
"dependencies": {
  "@repo/common":"*"
},
"keywords": [],
"author": "",
"license": "ISC"
}
```

Inside the global node_module we can see the



common -> index.ts

```
export const VALUE = "nishant thapa"


export const BACKEND_URL = "https://api.google.com"
```

When we try to export the common in backend we get the error

```
/Desktop/Dev/week_16.2/apps/backend$ rm -r dist
ntc@ntc-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Dev/week_16
.2/apps/backend$ npm run dev

> backend@1.0.0 dev
> tsc -b && node dist/index.js

/home/ntc/Desktop/Dev/week_16.2/packages/common/src/index.
ts:1
export const VALUE = "nishant thapa"
^^^^^^

SyntaxError: Unexpected token 'export'
    at internalCompileFunction (node:internal/vm:128:18)
    at wrapSafe (node:internal/modules/cjs/loader:1279:20)
    at Module._compile (node:internal/modules/cjs/loader:1
331:27)
```

Solution

```
// step1: cd common
// step2: In common/package.json,
// "exports" : {
// "./config": "./dist/index.js"
// },
// step3: tsc -b (in common folder)
// step4: tsc -b (in backend folder)
// step5: node dist/index.js

// This should work!
```