Search for dubinc/dub there the monorepo concept is used

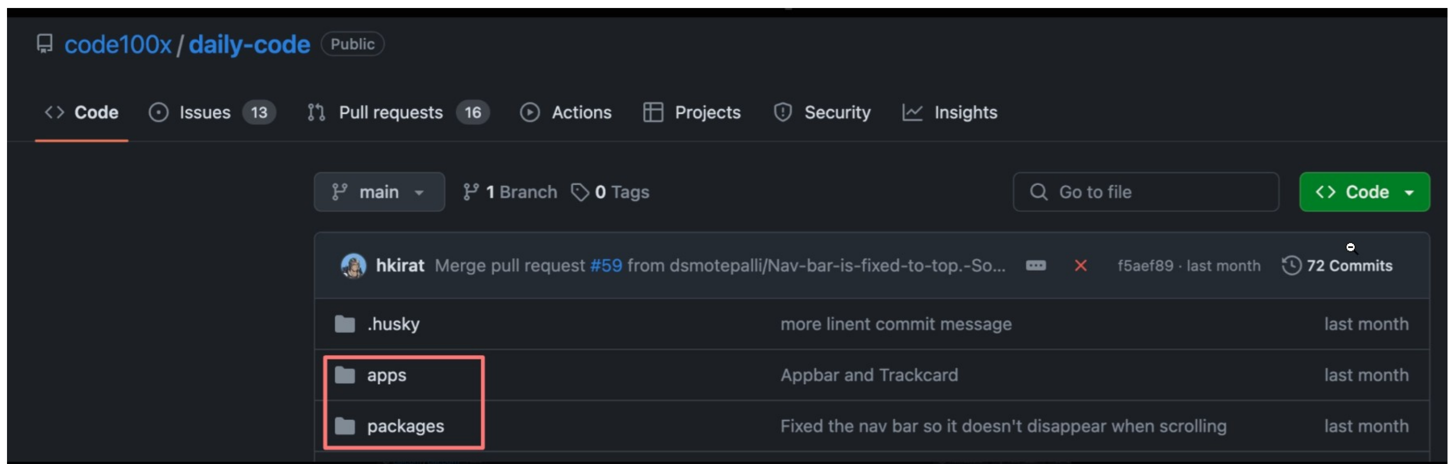
Its like weired way of structuring the project

What are monorepos

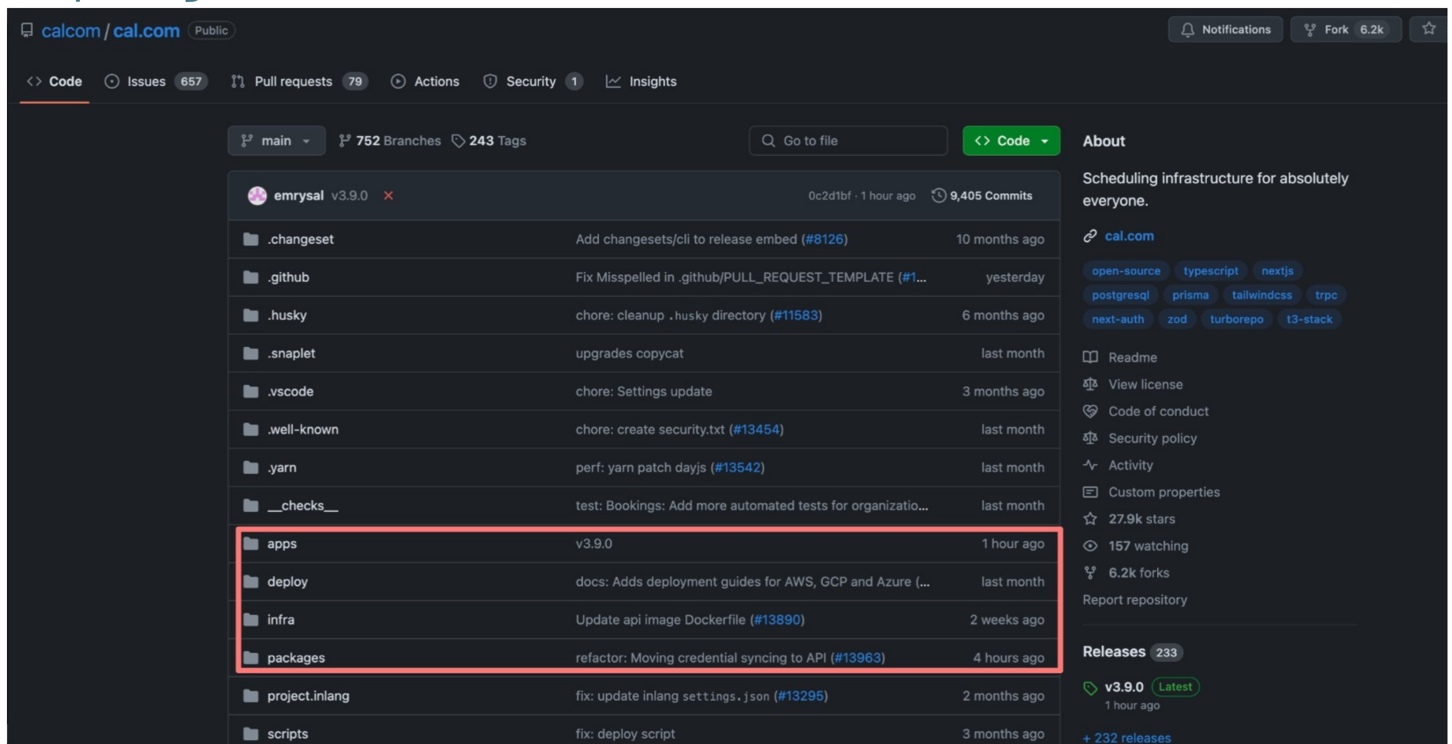
As the name suggests, a single repository (on github lets say) that holds all your frontend, backend, devops code.

Few repos that use monorepos are -

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



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



Do you need to know them very well as a full stack engineer?

Not exactly. Most of the times they are setup in the project already by the dev tools guy and you just need to follow the right practises

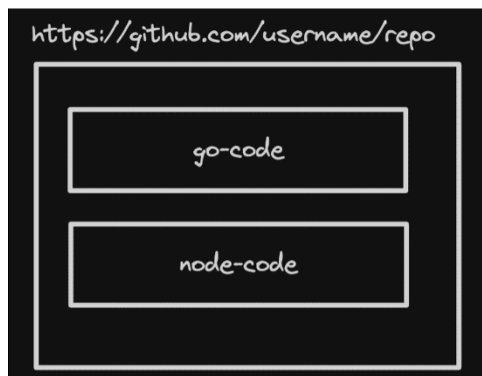
Good to know how to set one up from scratch though

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

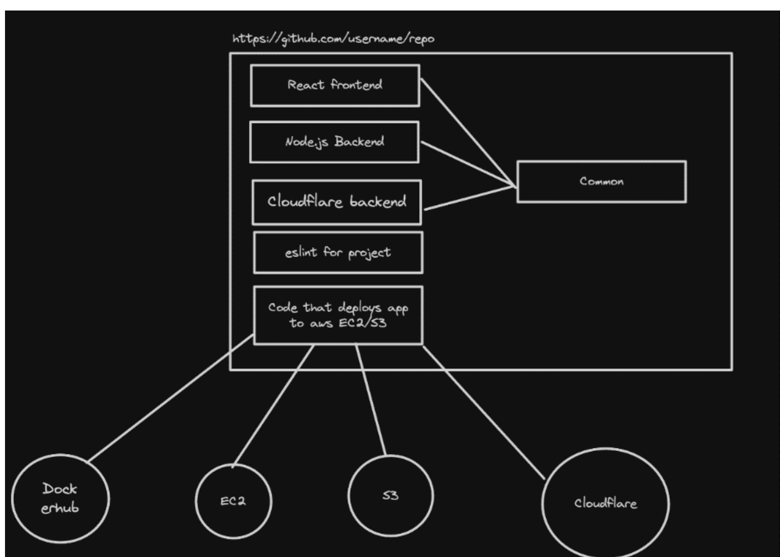


1. Services are highly decoupled (dont share any code) //which means the frontend in next js and backend in rust they both are totally not related to each other so good to keep them different

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

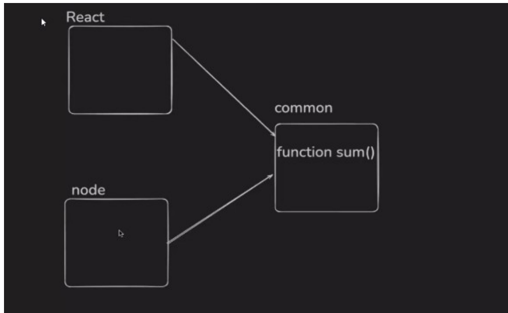
2. Enhanced Collaboration

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

THIS ONE LIKE SIMPLY MEANS WHEN THE PROJECT IS BUILDED LIKE BY USING NPM RUN DEV THAT TIME IT WILL TRY TO SAVE LIKE IF IN FRONTEND THE REACT CODE HAS NOT CHANGE THEN IT WILL RETURN THE

SAME FILE WHICH WILL HELP IN REDUCING BUILD TIME AND IS GOOD FOR A LARGE PROJECT

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. EG TSCONFIG FILES CAN BE SHARED TO STORE



This is like the usecase when there are dependency on same module used by both frontend and backend

Common monorepo framework in Node.js(HELPS TO SHARE CODE BETWEEN THE STACKS)

Lerna - https://lerna.js.org/

nx - https://github.com/nrwl/nx

Turborepo - https://turbo.build/ — Not exactly a monorepo framework

Yarn/npm workspaces - https://classic.yarnpkg.com/lang/en/docs/workspaces/

We’ll be going through turborepo since it’s the most relevant one today and provides more things (like build optimisations) that others don’t

History of Turborepo

Created by Jared Palmer

In December 2021 Acquired/aqui-hired by Vercel

Mild speculation/came from a random source - Pretty hefty dealp

They’ve built a bunch of products, Turborepo is the most used one

SOME JARGONS

Build System(VITE, next build)

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.

THIS IS NORMALLY A BUILDER LIKE IN C++ THERE IS GCC AND IN TS THERE IS TSC

Build System Orchestrator(turborepo which guides who to get build first and last)

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

THIS IS LIKE WHEN WE RUN NPM RUN BUILD THIS SYSTEM TELLS LIKE WHICH THING SHOULD BUILD FIRST IMAGINE THERE ARE 5 CODES INCLUDING FRONTEND, BACKEND SO WHICH SHOULD GET BUILD FIRST AND LAST IT DECIDES THAT

ALSO LIKE SEE IF SOMETHING IS CACHED IT WILL NOT RE BUILD THAT CODE

JUST TELLS OTHER BUILDERS THAT TU YEH KARDE TU YEH KARDE LIKE THAT

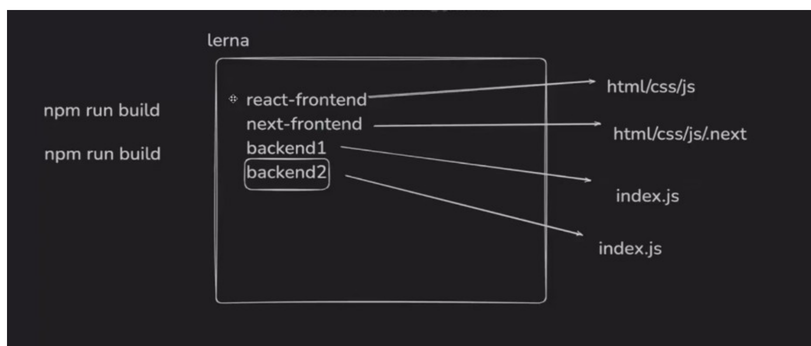
Monorepo Framework

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.

Turborepo as 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.

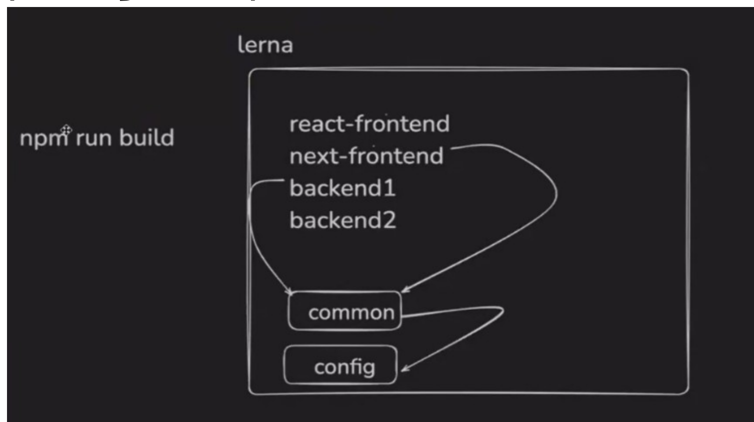


IN THIS LIKE MONOREPOS LIKE LERNA WHAT USED TO HAPPEN IS IF YOU RUN NPM RUN DEV IT WILL LIKE BUILD THE WHOLE PROJECT BUT IF YOU BUILD THE PROJECT AGAIN AND VERY LITTLE CHANGES IS DONE IT WILL THEN ALSO BUILD THE WHOLE PROJECT AGAIN

SO TO FIX THAT TURBOREPO CAME AND FIXED THAT USING **CACHING** AND THEY_STORE THE BUILDS IN THE **CLOUD SERVER** IMPROVES PERFORMANCE IN **CONTINOUS INTEGRATION ENV**

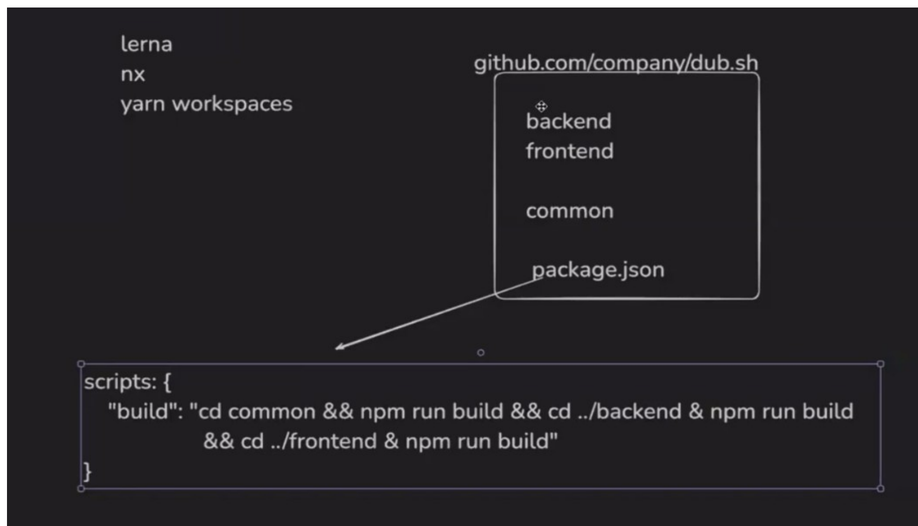
2. Parallelization: It can run independent tasks in parallel, making efficient use of your machine's resources. This reduces the overall time needed to complete all tasks in your project.

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.



THIS LINES CAN BE ASSUMED AS GRAPH LIKE COMMON CANNOT WORKS WITHOUT CONFIG AND FRONTEND AND BACKEND CANNOT WORKS WITHOUT COMMON SO IT KNOWS THE ORDER IN WHICH THE CODE SHOULD BE BUILD

SIMPLE THING WHAT TURBOREPO DO IS



like in this we defining the script manually turborepo is smart enough to do that and also manage the order who will be build first so we can just do **turbo build**

Let's initialize a simple Turborepo

Ref https://turbo.build/repo/docs

Initialize a Turborepo

npx create-turbo@latest

```
Application packages
 - apps/docs
 - apps/web
Library packages
 - packages/eslint-config
 - packages/typescript-config
 - packages/ui

>>> Success! Created your Turborepo at week-21-turborepo
```

the apps thing is the user sees

the packages is the code that the apps uses as a common code

Understanding the Project Structure

```json
{
  "name": "week-21-monorepo",
  "private": true,
  "scripts": {
    "build": "turbo run build",
    "dev": "turbo run dev",
    "lint": "turbo run lint",
    "format": "prettier --write \"**/*.{ts,tsx,md}\"",
    "check-types": "turbo run check-types"
  },
  "devDependencies": {
    "prettier": "^3.7.4",
    "turbo": "^2.8.10",
    "typescript": "5.9.2"
  },
  "engines": {
    "node": ">=18"
  },
  "packageManager": "npm@11.8.0",
  "workspaces": [
    "apps/*",
    "packages/*"
  ]
```

The apps folder contains two similar nextjs project docs and web

Packages folder contains the components which are used in both the pages and when we use npm run dev it runs both the nextjs projects

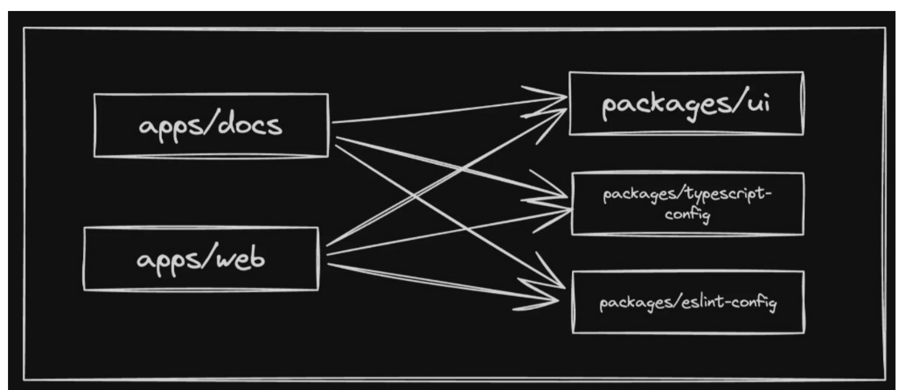
Imagine you have these two folders and both are build first and later you made changes to the web folder and then again rebuild then the docs project will be taken from cache and and the web folder will be rebuild

CI / CD PIPELINE = Continuous Integration, Continuous Deployment during this procedure the turborepo cache things helps a lot builds are fast and highly optimized

Explore the folder structure

There are 5 modules in our project

<u>End user apps (websites/core backend)</u>



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

<u>Helper packages</u>

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

Let's try to run the project

You will notice two websites running on

1. localhost:3000
2. localhost:3001

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

Exploring root package.json

```json
{
  "name": "project",
  "private": true,
  "scripts": {
    "build": "turbo build",
    "dev": "turbo dev",
    "lint": "turbo lint",
    "format": "prettier --write \"**/*.{ts,tsx,md}\""
  },
  "devDependencies": {
    "@repo/eslint-config": "*",
    "@repo/typescript-config": "*",
    "prettier": "^3.2.5",
    "turbo": "latest"
  },
  "engines": {
    "node": ">=18"
  },
  "packageManager": "npm@7.24.2",
  "workspaces": [
    "apps/*",
    "packages/*"
  ]
}
```

turbo build system

npm workspaces

the npm workspaces folder is the folder which contains the code which is shared between the next js projects

EVER STARTING A COMPANY BASED PROJECT ALWAYS USE LIKE MONOREPO

npx create-turbo@latest

doing style raw because ts can be complicated

packages/ui should contain the ui parts

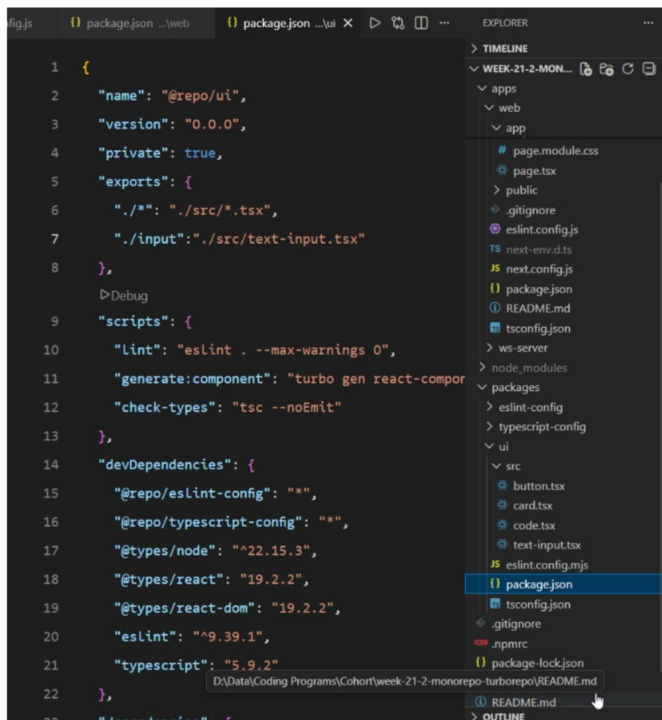
blade.razorpay.com contains all the components of the razorpay website

so you should keep the ui items in the ui packages folder

in the ui folder you firstly define interface and then used in export function

while importing the function you never put a absolute path but we add a package name

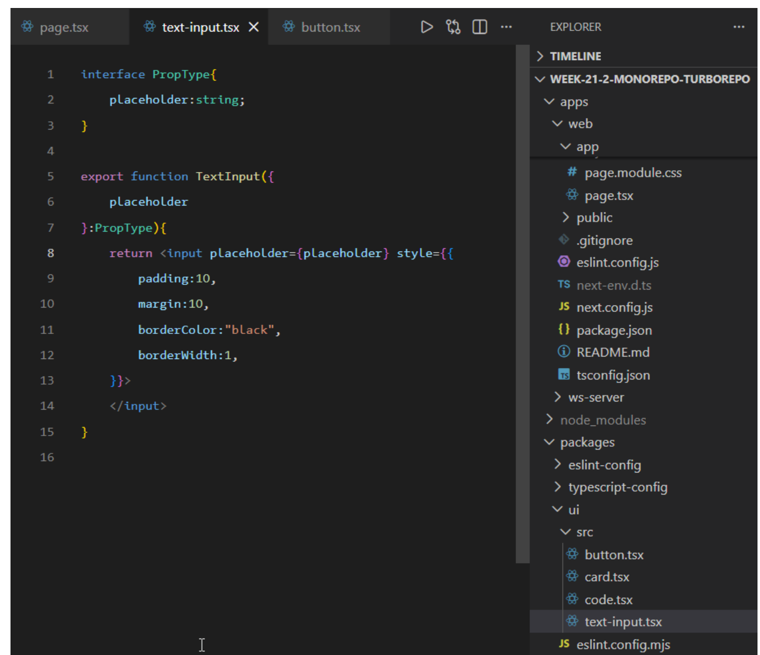
after creating a package in the ui/src folder then you have to add in package.json



to use the package you use this

import {TextInput} from "@repo/ui/input";

you can also add variant in the below placeholder



1. Initialised a monorepo
2. We learned how to create a design system/
   re-use/import things from a `ui` module.
3. We learnt about "exports" in package.json
4. We created a vary minimal frontend for our chat app

----

1. We added a ws and a http-server folder
2. We initialized package.json in both of them
3. We put the common tsconfig.json in the `typescript-configs` module
4. We added code to the express server
5. We added a dev and a build script