Vinodh Kumar
Posted on Aug 15 • Updated on Aug 19

Create a monorepo using PNPM workspace

#javascript #monorepo #typescript

Objective

To create a mono repo using the PNPM package manager and its workspace feature.

The main advantage of the PNPM workspace when compared to the yarn workspace is common packages are not hoisted to the root directory thereby making all the workspace packages completely isolated.

Technologies/Features used

The mono repo we are going to build will have the following features. Again this is my set of tools feel free to change it based on your preference.

Feature	Technology used
Package manager	PNPM
Programming language	Typescript
Basic linting	ESLint
Code formatting	Prettier
Pre-commit hook validator	Husky
Linting only staged files	lint-staged
Lint git commit subject	commitlint

Prerequisites

Tools

You will need the following things properly installed on your computer.

- Git
- Node.js

PNPM install

- If you have installed the latest v16.x or greater node version in your system, then enable the pnpm using the below cmd

```
corepack enable
corepack prepare pnpm@latest --activate
```

- If you are using a lower version of the node in your local system then check this page for additional installation methods https://pnpm.io/installation

Repo basic setup

- Initialize git if you want and enforce the node version with some info in the README.md.

```
mkdir pnpm-monorepo
cd pnpm-monorepo
pnpm init
git init
echo -e "node_modules" > .gitignore
npm pkg set engines.node=">=18.16.1" // Use the same node version you installed
npm pkg set type="module"
echo "#PNPM monorepo" > README.md
```

Code formatter

I'm going with Prettier to format the code. Formatting helps us to keep our code uniform for every developer.

Installation

- Let's install the plugin and set some defaults. Here I'm setting the single quote to be true, update it according to your preference.

```
pnpm add -D prettier
echo '{\n  "singleQuote": true\n}' > .prettierrc.json
echo -e "coverage\npublic\ndist\npnpm-lock.yaml\npnpm-workspace.yaml" > .prettierignore
```

VS Code plugin

- If you are using VS Code, then navigate to the `Extensions` and search for `Prettier - Code formatter` and install the extension.

Extension link: https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode

- Let's update the workspace to use the prettier as the default formatter and automatically format the file on save.
- Create the VS Code workspace settings JSON and update it with the following content.

```
mkdir .vscode && touch .vscode/settings.json
```

```
{
  "editor.formatOnSave": true,
  "editor.defaultFormatter": "esbenp.prettier-vscode"
}
```

Linting

Linter statically analyses your code to quickly find problems. ESLint is the most preferred tool for linting the Javascript code.

ESLint

```
pnpm create @eslint/config
```

- The ESLint will ask you a set of questions to set up the linter as per your needs. This is the configuration I've chosen for this project.

```
? How would you like to use ESLint? …
  To check syntax only
❯ To check syntax and find problems
  To check syntax, find problems, and enforce code style

? What type of modules does your project use? …
❯ JavaScript modules (import/export)
  CommonJS (require/exports)
  None of these

? Which framework does your project use? …
  React
  Vue.js
❯ None of these

Does your project use TypeScript? › No / Yes
- Yes
```

```
Where does your code run?
✓ Browser
  Node

? What format do you want your config file to be in? …
❯ JavaScript
  YAML
  JSON

The config that you've selected requires the following dependencies:
@typescript-eslint/eslint-plugin@latest @typescript-eslint/parser@latest

? Would you like to install them now? › No / Yes
- Yes

? Which package manager do you want to use? …
  npm
  yarn
❯ pnpm
```

- Set the root property to true in the ESLint config. This will make sure the linting config bubbling is stopped here and also rename the `.eslintrc.js` to `.eslintrc.cjs`.

```
module.exports = {
  root: true,
  ...
}
```

- Create the eslintignore file to let the ESLint know which files to not format.

```
touch .eslintignore
echo -e "coverage\npublic\ndist\npnpm-lock.yaml\npnpm-workspace.yaml" > .eslintignore
```

Integrating Prettier with ESLint

Linters usually contain not only code quality rules but also stylistic rules. Most stylistic rules are unnecessary when using Prettier, but worse – they might conflict with Prettier!

We are going to use Prettier for code formatting concerns, and linters for code-quality concerns. So let's make the linter run the stylistic rules of 'Prettier' instead.

- Install the necessary plugins

```
pnpm add -D eslint-config-prettier eslint-plugin-prettier
```

- Add the `plugin:prettier/recommended` as the **last element** in the extends property in eslintrc.js

```
module.exports = {
  extends: [..., 'plugin:prettier/recommended'],
}
```

For more info on this: https://prettier.io/docs/en/integrating-with-linters.html

- Let's create scripts for running the linter and prettier in the package.json file.

```
npm pkg set scripts.lint="eslint ."
npm pkg set scripts.format="prettier --write ."
```

- Run the `pnpm lint` cmd to run the ESLint and `pnpm format` cmd to format the files.

Pre-commit hook validation

Even if we added all these linter and formatter mechanisms to maintain the code quality, we can't expect all the developers to use the same editor and execute the `lint` and `format` command whenever they are pushing their code.

To automate that we need some kind of pre-commit hook validation. That's where husky and lint-staged plugins come in handy let's install and set them up.

- Install the husky, commitlint, and lint-staged NPM package and initialize it as shown below,

```
pnpm add -D @commitlint/cli @commitlint/config-conventional
echo -e "export default { extends: ['@commitlint/config-conventional'] };" > commitlint.config.js
pnpm add -D husky lint-staged
npx husky install
npx husky add .husky/pre-commit "pnpm lint-staged"
npx husky add .husky/commit-msg  'npx --no -- commitlint --edit ${1}'
npm pkg set scripts.prepare="husky install"
```

- Update the package.json file and include the following property. This will run the ESLint on all the script files and Prettier on the other files.

```
"lint-staged": {
  "**/*.{js,ts,tsx}": [
    "eslint --fix"
  ],
  "**/*": "prettier --write --ignore-unknown"
},
```

Workspace config

- Create `pnpm-workspace.yaml` file and add the following content

```
touch pnpm-workspace.yaml
```

```
packages:
  - 'apps/*'
  - 'packages/*'
```

- Create the apps and packages directories in the root.

```
mkdir apps packages
```

Sample package - Common

- Create a sample package that can be used in the workspace apps.

```
cd packages
pnpm create vite common --template vanilla-ts
cd ../
pnpm install
npm pkg set scripts.common="pnpm --filter common"
```

- Update the main.ts file with the following content to create a simple isBlank util.

```
/* eslint-disable @typescript-eslint/no-explicit-any */
export const isEmpty = (data: any) => data === null || data === undefined

export const isObject = (data: any) => data && typeof data === 'object'

export const isBlank = (data: any) =>
  isEmpty(data) ||
  (Array.isArray(data) && data.length === 0) ||
  (isObject(data) && Object.keys(data).length === 0) ||
  (typeof data === 'string' && data.trim().length === 0)
```

- Delete the sample files

```
cd packages/common
rm -rf src/style.css src/counter.ts
```

Library mode

Vite by default builds the assets in app mode with index.html as the entry file. But we want our app to expose our main.ts file as the entry file, so let's update the Vite config to support that.

- Before that let's install the Vite package to auto-generate the type definitions from the library.

```
pnpm common add -D vite-plugin-dts
```

- Create the vite.config.ts file and update it like this,

```
 touch vite.config.ts
```

```
import { defineConfig } from 'vite'
import { resolve } from 'path'
import dts from 'vite-plugin-dts'

// https://vitejs.dev/config/
export default defineConfig({
  build: { lib: { entry: resolve(__dirname, 'src/main.ts'), formats: ['es'] } },
  resolve: { alias: { src: resolve('src/') } },
  plugins: [dts()],
})
```

The `resolve` property helps us to use absolute import paths instead of relative ones. For example:

```
import { add } from 'src/utils/arithmetic'
```

- Update the common package `package.json` file with the entry file for our script as well as the typings.

```
{
 ...,
 "main": "./dist/common.js",
 "types": "./dist/main.d.ts",
}
```

Sample app - Web app

- Create a sample app that can make use of the workspace package `common`.

```
cd apps
pnpm create vite web-app --template react-ts
cd ../
pnpm install
npm pkg set scripts.app="pnpm --filter web-app"
```

- Install the common package as a dependency in our web app by updating the web-app package.json.

```
"dependencies": {
 "common": "workspace:*",
 ...,
 }
```

- Run `pnpm install` again so that 'web-app' can symlink the common package present in the workspace
- Run `pnpm common build` so that the common package can be found by the web-app server.
- Update the `App.tsx` like below,

```
import { isBlank } from 'common'

const App = () => {
  return (
    <>
      <p>undefined isBlank - {isBlank(undefined) ? 'true' : 'false'}</p>
      <p>false isBlank - {isBlank(false) ? 'true' : 'false'}</p>
      <p>true isBlank - {isBlank(true) ? 'true' : 'false'}</p>
      <p>Empty object isBlank - {isBlank({}) ? 'true' : 'false'}</p>
    </>
  )
}

export default App
```

- Run `pnpm app dev` and check whether the `common` package util is successfully linked to the app.

That's it. We have successfully created a PNPM mono repo from scratch with typescript support.

Dev mode

- Most of the time, you just need to build the `common` package once and use it in the repo apps. But if you are actively making changes in your `common` package and want to see that in the 'web-app' immediately you can't build the `common` app again and again for every change.

To avoid this, let's run the `common` package in watch mode so that any change in the code will rebuild automatically and reflect in the 'web-app' in real-time.

- Run these commands in different terminals.

```
pnpm common build --watch
pnpm web-app dev
```

Advantages:

- All your code will be in one single repo with proper isolation.
- Only a one-time effort is needed to set up the repo with proper linting, formatting, and pre-commit hook validations which will be extended by the workspace packages.
- All the packages will have a similar setup, look and feel.

Tips:

- Check out my blog on creating a TS Util library and React app for creating repo packages with all the bells and whistles. Ignore the prettier, pre-commit hook validations in those packages as they are already handled in the root workspace of this mono repo.
- For linter alone, if you are good with the basic linting present in the root workspace you don't have to do anything special in the package. However, for apps like React, we will have some more plugins dedicated to lint the React library.

Ex:

```
module.exports = {
  root: true,
  env: { browser: true, es2020: true },
  extends: [
    'eslint:recommended',
    'plugin:@typescript-eslint/recommended',
    'plugin:react-hooks/recommended',
  ],
  ignorePatterns: ['dist', '.eslintrc.cjs'],
  parser: '@typescript-eslint/parser',
  plugins: ['react-refresh'],
  rules: {
    'react-refresh/only-export-components': ['warn', { allowConstantExport: true }],
```

```
  },
}
```

You can keep it like this itself or else you can make it extend the root by simply removing the root property and remove the duplicates.

```
module.exports = {
  extends: ['plugin:react-hooks/recommended'],
  plugins: ['react-refresh'],
  rules: {
    'react-refresh/only-export-components': ['warn', { allowConstantExport: true }],
  },
}
```

Sample repo

The code for this post is hosted in Github here

Please take a look at the Github repo and let me know your feedback, and queries in the comments section.

Top comments (3)

Wolz • Oct 18

Amazinnngggg, I learned a lot

Garen • Aug 15

awesome article ! I learn how to config a monorepo project

Dev.Casamia • Oct 2

I really thank you. Very clean and well detailed article.