Guidebook
The Javascript Monorepo

Welcome to **aGuideHub**

Sharing common code between multiple projects, we have three ways and Monorepo is one of them. This guidebook explains monorepo with practical examples of sharing and using code in multiple projects.

Github Project Source - https://github.com/infinitbility/the-javascript-monorepo

Let's start with the index

PUBLISHED BY aguidehub.com

WHAT IS MONOREPO?

Monorepo is a nickname that means "using one repository for the source code management version control system".

- A monorepo architecture means using one repository, rather than multiple repositories.
- For example, a monorepo can use one repo that contains a directory for a web app project, a directory for a mobile app project, and a directory for a server app project.
- Monorepo is also known as one-repo or uni-repo.

> **Note: The shared library can be maintained in a separate repository or the same repository as your other projects.**

PROPONENTS OF MONOREPO

If components need to release together, then use a monorepo

If you think components might need to release together then they should go in the same repo, because you can in fact pretty easily manage projects with different release schedules from the same repo if you really need to.

On the other hand, if you've got a whole bunch of components in different repos which need to be released together it suddenly becomes a real pain.

If components need to share a common code, then use a monorepo

If you have components that will never need to release together, then of course you can stick them in different repositories-- but if you do this and you want to share common code among the repositories, then you will need to manage that code with some sort of robust versioning system, and robust versioning systems are hard. Only do something

like that when the value is high enough to justify the overhead. If you're in a startup, chances are very good that the value is not high enough.

I've found monorepos to be extremely valuable in a less-mature, high-churn codebase

Need to change a function signature or interface? Cool, global find & replace.

At some point, a monorepo outgrows its usefulness. The sheer amount of files in something that's 10K+ LOC (not that large, I know) warrants breaking apart the codebase into packages.

Still, I almost err on the side of monorepos because of the convenience that editors like vscode offer: autocomplete, auto-updating imports, etc.

A common mission

I find it helpful to think of a company as a group of people engaged in a common mission. The company pursues its mission through multiple subprojects, and every decision taken and every code change introduced is a step towards its primary goal. The codebase is a chunk of the company's institutional knowledge about its overarching goal and means to that end.

Looking at it from this perspective, a monorepo can be seen as the most natural expression of the fact that all team members are engaged in a single if multi-faceted, enterprise.

SO, IS A MONOREPO A GOOD IDEA?

Using a mono repository is a good idea for many companies. You can keep all of the source code (and other files/digital assets) from every team in one repository. This makes it easier to share with everyone. And it helps you maintain a single source of truth.

After any commit, the new code is visible to and usable by all of your developers. This helps avoid painful merges that are prevalent with trunk-based development.

Here are some reasons why you should consider using a mono repository:

- You want a single source of truth.
- You want to share and reuse code easily.
- You want visibility to manage dependencies (e.g., if you make a change, what else will be impacted?).
- You want to make atomic changes (e.g., one operation to make a change across multiple projects).
- You want teams to collaborate more.
- You want to make large-scale changes (e.g., code refactoring).

If you decide to go the monorepo route, you'll be in good company. Leading companies — like Google — use a monorepo.

BEST PROP & CONS

Pros

Any changes you make to the library project will be immediately available in the other local projects that depend on it. This option is the most convenient method for local development.

Cons

Other developers who work on these projects will have to go through specific steps to set it up. This option is the most *inconvenient* method for collaborating with other developers, especially if you are not using a monorepo.

MONOREPO EXAMPLE

Well, first we will create sample applications in node.js and react.js. After the application sample setup is done we will create one more project for common code and we will use this project as a package in node.js and react.js sample application without publishing it on NPM.

Created a Monorepo folder, where we can set up all our projects.

Create node.js application

I have created sample-node projects in the monorepo folder, where we set up node applications. Let's start with `npm init` .

> **npm init**

As we know, for creating a node application we have to enter the `npm init` command. After initialization is successful we have to install the express package `` `npm i -s express` ``.

> **npm i -s express**

- Add start command in your package.json script key.

```
  "scripts": {
    "start": "node index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

- Add index.js file with below code.

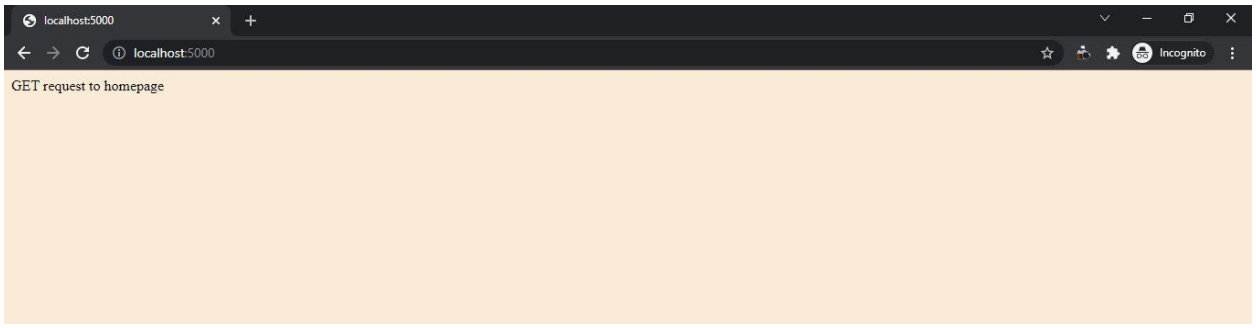
```
const express = require('express'); //Import the express dependency
const app = express();
const port = 5000;
app.get('/', (req, res) => {
    res.send('GET request to homepage');
});
app.listen(port, () => {
```

```
    console.log(`Now listening on port ${port}`);
});
```

- For verify your node.js application working or not

```
cd sample-node && npm start
```

Open your browser and enter http://localhost:5000



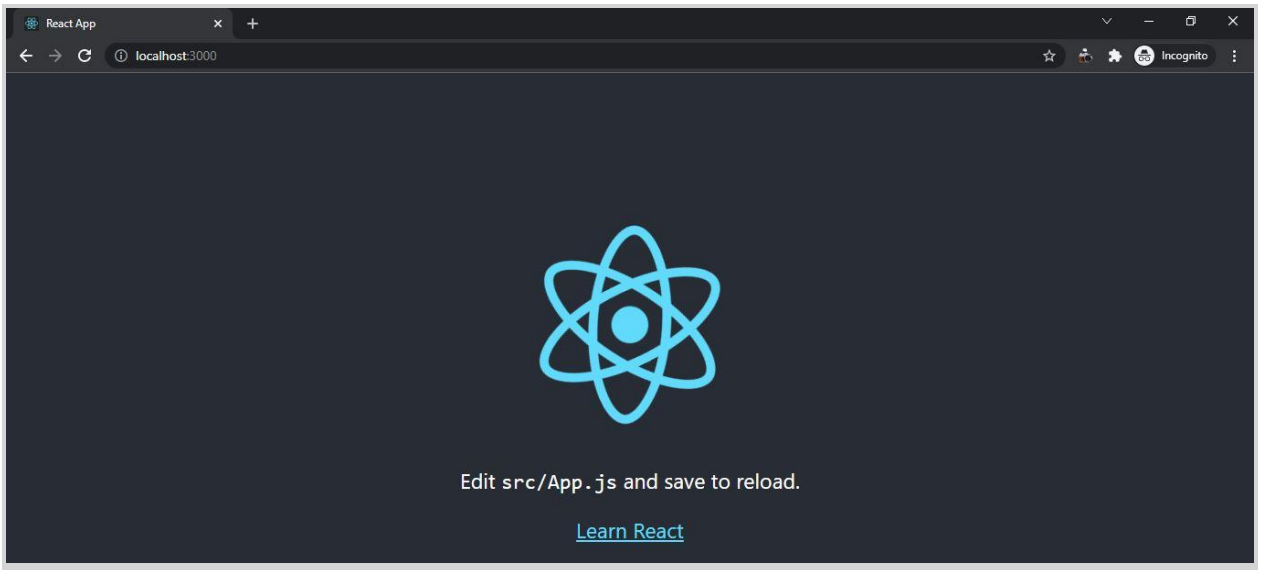
Create react.js application

For creating a react application you have to just enter `npx create-react-app sample-react` .

```
npx create-react-app sample-react
```

After your react application process is done just verify your application is working or not.

```
cd sample-react && npm start
```



Create shareable project

For creating shareable project, we have to do same like node.js project,'

1. Create folder
2. Enter npm init
3. Write what you want to share to all projects

Here, I'm going to create a **utility** folder, where I will put my constant and common operation functions.

Now, I've created an index.js file with sample const and function, which I'm going to use on my sample-node and sample-react project.

index.js

```
const STATUS = "RUNNING";

const getMyName = () => {
    return "Infinitbility";
}

module.exports = {
    STATUS,
    getMyName
}
```

Now, for making shareable code we are going to use Symbolic **Link.**

Let use the npm link command to make this project available to install on other projects. Like the image below.

> **npm link**

```
E:\>cd Monorepo\utility

E:\Monorepo\utility>npm link
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN utility@1.0.0 No repository field.

up to date in 0.929s
found 0 vulnerabilities

C:\Users\User\AppData\Roaming\npm\node_modules\utility -> E:\Monorepo\utility

E:\Monorepo\utility>
```

Now this project is ready to use in our other projects.

Let's go to our both projects and run `npm link utility`.

> **npm link utility**

```
E:\Monorepo>cd sample-node

E:\Monorepo\sample-node>npm link utility
E:\Monorepo\sample-node\node_modules\utility -> C:\Users\User\AppData\Roaming\npm\node_modules\utility -> E:\Monorepo\utility

E:\Monorepo\sample-node>cd ..

E:\Monorepo>cd sample-react

E:\Monorepo\sample-react>npm link utility
E:\Monorepo\sample-react\node_modules\utility -> C:\Users\User\AppData\Roaming\npm\node_modules\utility -> E:\Monorepo\utility

E:\Monorepo\sample-react>
```

Let's use utility code in our sample-node project and check output.

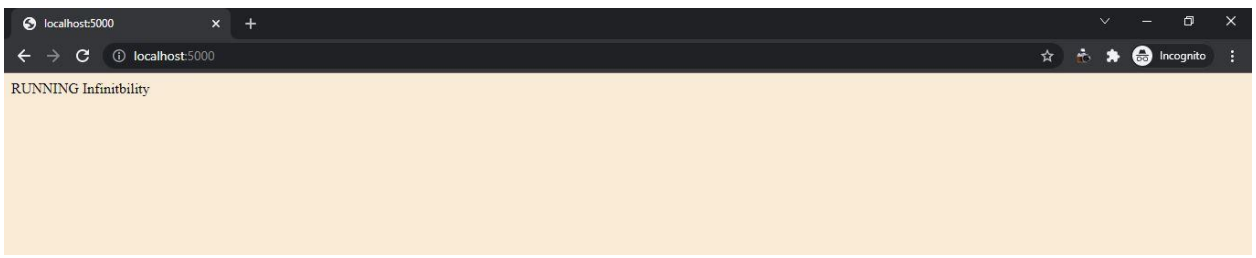
sample-node\index.js

```
const express = require('express'); //Import the express dependency
const app = express();              //Instantiate an express app, the main
work horse of this server
const port = 5000;                  //Save the port number where your
server will be listening
const utility = require('utility');

//Idiomatic expression in express to route and respond to a client request
app.get('/', (req, res) => {        //get requests to the root ("/") will
route here
    let platformName = utility.getMyName();
    res.send(`${utility.STATUS} ${platformName}`);
});

app.listen(port, () => {            //server starts listening for any
attempts from a client to connect at port: {port}
    console.log(`Now listening on port ${port}`);
});
```

sample-node output

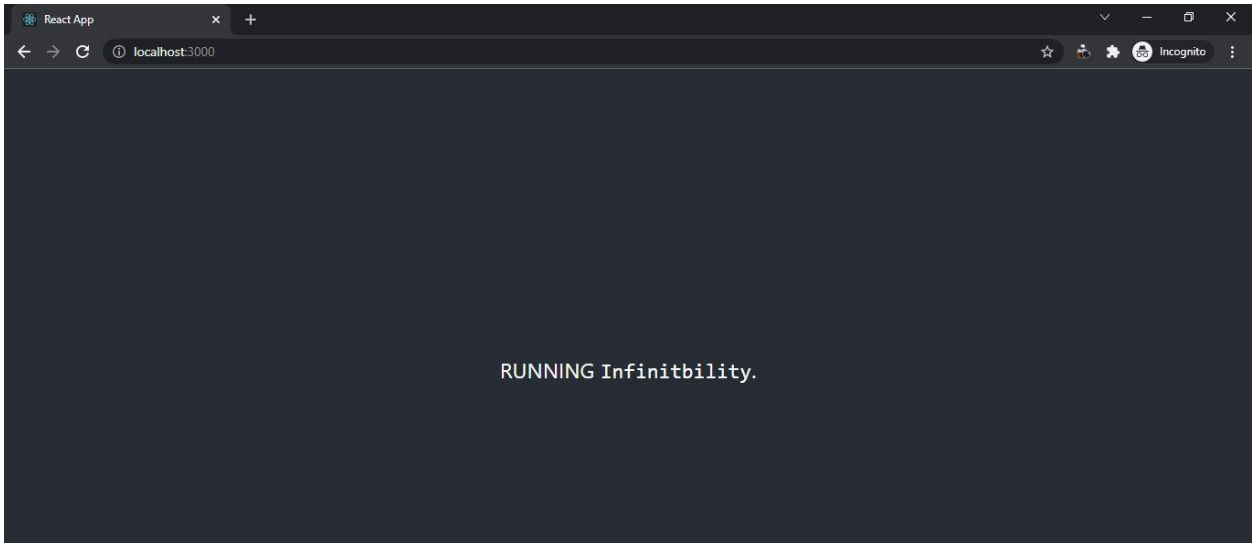


Let's use utility code in our sample-react project and check output.

```
import './App.css';
import { STATUS, getMyName } from 'utility';
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <p>
          {STATUS} <code>{getMyName()}</code>.
        </p>
      </header>
    </div>
  );
}

export default App;
```

Sample-react output



IN THIS BOOK WE ARE USING THE SYMBOLIC LINK METHOD TO CREATE MONOREPO PROJECT BUT THIS FLOW IS ALSO POPULAR FOR TESTING PACKAGES ON LOCAL.

THIS METHOD TEST WITH DEPLOY CODE ON PRODUCTION.

THANKS FOR READING.