Karthik Gotrala

52 Followers About Follow

Monorepo Pattern: Setting up Angular workspace for multiple applications in one single repository

Introduction:

Karthik Gotrala May 20, 2020 · 11 min read

In this article, I would like to discuss my experience while setting up an Angular project to use the Monorepo pattern. The Monorepo, as the name suggests *mono* (single) and *repo* (repository of the codebase) is a single source of truth for the entire organization code base. Monorepo as such is a broad topic, in this article, my focus would be primarily on creating an angular workspace that uses monorepo pattern.

Pre-requisites:

1. **Angular Framework:** Basic understanding of Framework and its associated architecture.

2. **Typescript:** Primary language is used by the Angular Framework.

3. **NodeJS:** As a development server and as a dependency manager via Node package manager (NPM).

5. **Visual Studio Code:** Text editor (or any other popular editor WebStorm, Atom, Sublime Text, Brackets, etc.,)

> ***Thank you to the Angular Team/Community and all supporting tools that make developers life easier and easier day by day***

Scope of the Article:

As mentioned in the introduction, the scope of this article is limited to setting up an angular workspace for building a web-based UI/UX application. We will try to understand how the files related to multiple web applications can be grouped together into one single repository which allows us to re-use without creating standalone libraries which may be added as a dependency in package.json. I will walk through a list of angular CLI commands I used to set up the workspace with multiple sub-projects and re-usable shared libraries.

Also, this is not a one fit architecture solution for all web-based applications as each one is different and needs to be handled case by case. This would be one of many such use cases which may definitely have some flaws but my main intention is to give some idea. Again, to be clear my intention is not to discuss why monorepo? but how to monorepo?.

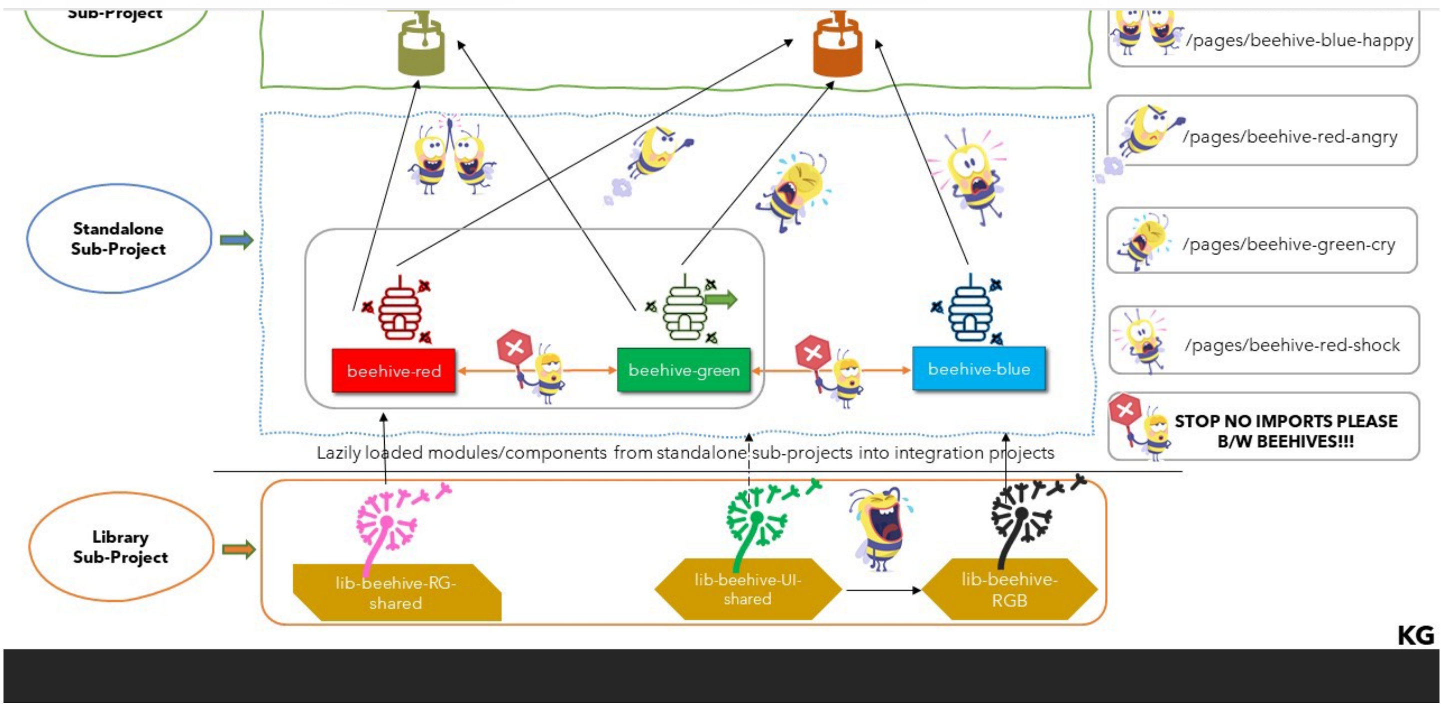
Main Characters of our made-up Beehive Story

Made-up Beehive Story:

Once upon a time, there was a **Queen honey-bee** named "Lucky" who tasked to prepare colorful Honey bottle(s) mixed with emotions(happy, Angry, Shock, Sad) as flavors. To achieve this, lucky decided to use RED, GREEN, BLUE **Beehive(s) colonies** (standalone sub-projects) that hold **horizontal cells** (modules/components) for storing honey with emotions as flavors. **Honey bottle(s)** (Integration Sub-Projects) are used to mix and match different flavors of honey from the beehive colonies(components/modules standalone sub-projects) with some spices as **pollen** extracted from **flowers** (library sub-project).

```
Beehive(s) Colonies ===> Stadalone sub-projects

Horizontal cells ===>modules/components

Honey Bottles ===> Integration sub-projects

Flowers==>pollens==> library sub-project
```

I know it's too much imagination. Well, I tried :-). Also, truth to be told, this story partly inspired by hearing from my 4-year-old son who might have learned it from his lot of TV time he is getting these days. Hopefully, the below picture gives a better idea?



A pictorial illustration of Angular Workspace which uses Monorepo pattern

The `ng-beehive-monorepo` Angular Workspace can be divided into sub-projects which are categorized into:

1. **Standalone Sub-project(s):** These are standalone projects which hold modules/components falling under one set of business functionality or domain. These can be used to launch the application independently or The modules defined in this project are exposed to be lazy-loaded into the integration sub-integration. Also, code sharing between standalone sub-projects should be avoided in order to avoid circular dependencies. See picture: `beehive-red, beehive-green, beehive-red`

2. **Integration Sub-Project(s):** These primarily work as integration project which consolidates bits and pieces from standalone and library projects and serves it as a web-application.
   See picture: `beehive-rgb, beehive-rg`

3. **Library Sub-Project(s):** The library sub-projects hold any components/modules/directive/pipes/interceptors that may be used in more than on sub-project or integration project.
   see the picture: `lib-beehive-RG-shared, lib-beehive-UI-shared, lib-beehive-RGB`

Time to get our hands dirty:

Since we got some background on what our goals, now let us go ahead and start creating the angular workspace:

Source code for our sample beehive made up beehive project can be found below:

kgotgit/ng-beehive-monorepo

This project was generated with Angular CLI version 9.1.6. Run ng serve for a dev server. Navigate to...

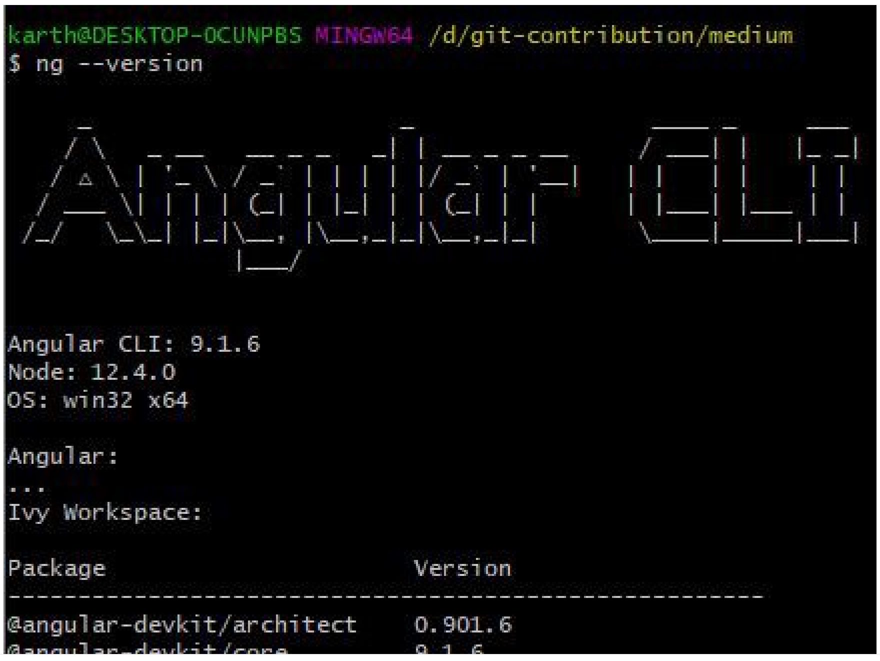
github.com

1. Install Angular CLI using NPM

```
npm i @angular/cli -g
```

2. Confirm Angular CLI is installed successfully

```
ng --version
```

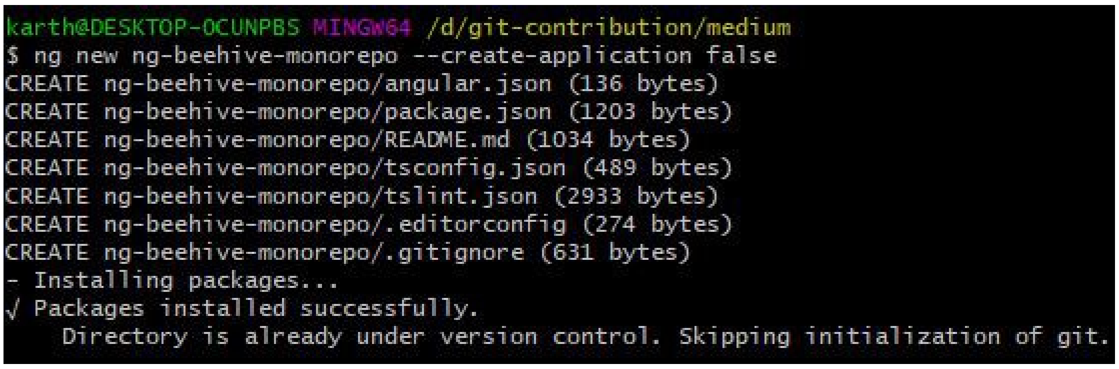


```
karth@DESKTOP-OCUNPBS MINGW64 /d/git-contribution/medium
$ ng --version

     _                      _                 ____ _     ___
    / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
   / △ \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
  / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
                |___/


Angular CLI: 9.1.6
Node: 12.4.0
OS: win32 x64

Angular:
...
Ivy Workspace:

Package                      Version
------------------------------------------------------------
@angular-devkit/architect    0.901.6
@angular-devkit/core         9.1.6
```

Successfully installed Angular CLI

3. Create Angular Workspace `ng-beehive-monorepo`

```
ng new ng-beehive-monorepo --create-application false
```

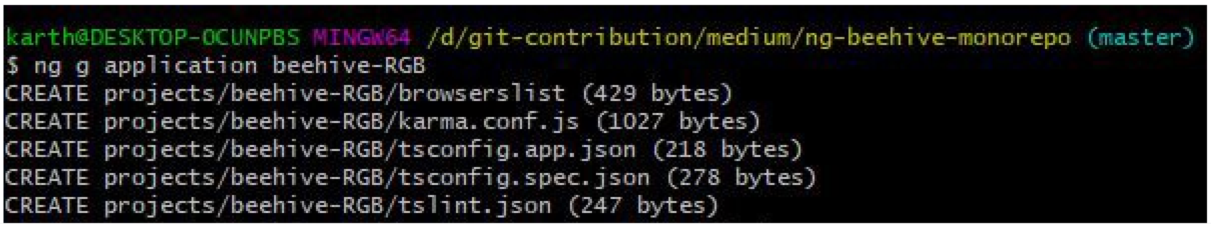


```
karth@DESKTOP-OCUNPBS MINGW64 /d/git-contribution/medium
$ ng new ng-beehive-monorepo --create-application false
CREATE ng-beehive-monorepo/angular.json (136 bytes)
CREATE ng-beehive-monorepo/package.json (1203 bytes)
CREATE ng-beehive-monorepo/README.md (1034 bytes)
CREATE ng-beehive-monorepo/tsconfig.json (489 bytes)
CREATE ng-beehive-monorepo/tslint.json (2933 bytes)
CREATE ng-beehive-monorepo/.editorconfig (274 bytes)
CREATE ng-beehive-monorepo/.gitignore (631 bytes)
- Installing packages...
√ Packages installed successfully.
    Directory is already under version control. Skipping initialization of git.
```

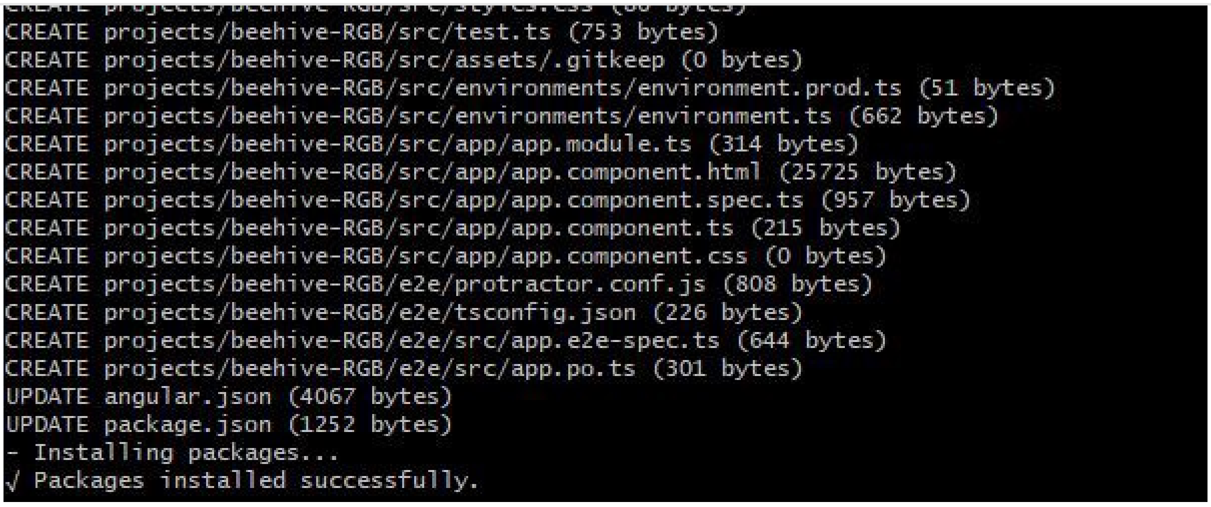
setting up ng-beehive-monorepo angular workspace

4. Create all sub-application(s)

```
//create beehive-RGB sub-applciation
ng g application beehive-RGB --routing --style=scss

//create beehive-RG sub-applciation
ng g application beehive-RG --routing --style=scss

//create beehive-red sub-applciation
ng g application beehive-red --routing --style=scss

//create beehive-green sub-applciation
ng g application beehive-green --routing --style=scss

//create beehive-blue sub-applciation
ng g application beehive-blue --routing --style=scss
```



```
karth@DESKTOP-OCUNPBS MINGW64 /d/git-contribution/medium/ng-beehive-monorepo (master)
$ ng g application beehive-RGB
CREATE projects/beehive-RGB/browserslist (429 bytes)
CREATE projects/beehive-RGB/karma.conf.js (1027 bytes)
CREATE projects/beehive-RGB/tsconfig.app.json (218 bytes)
CREATE projects/beehive-RGB/tsconfig.spec.json (278 bytes)
CREATE projects/beehive-RGB/tslint.json (247 bytes)
```

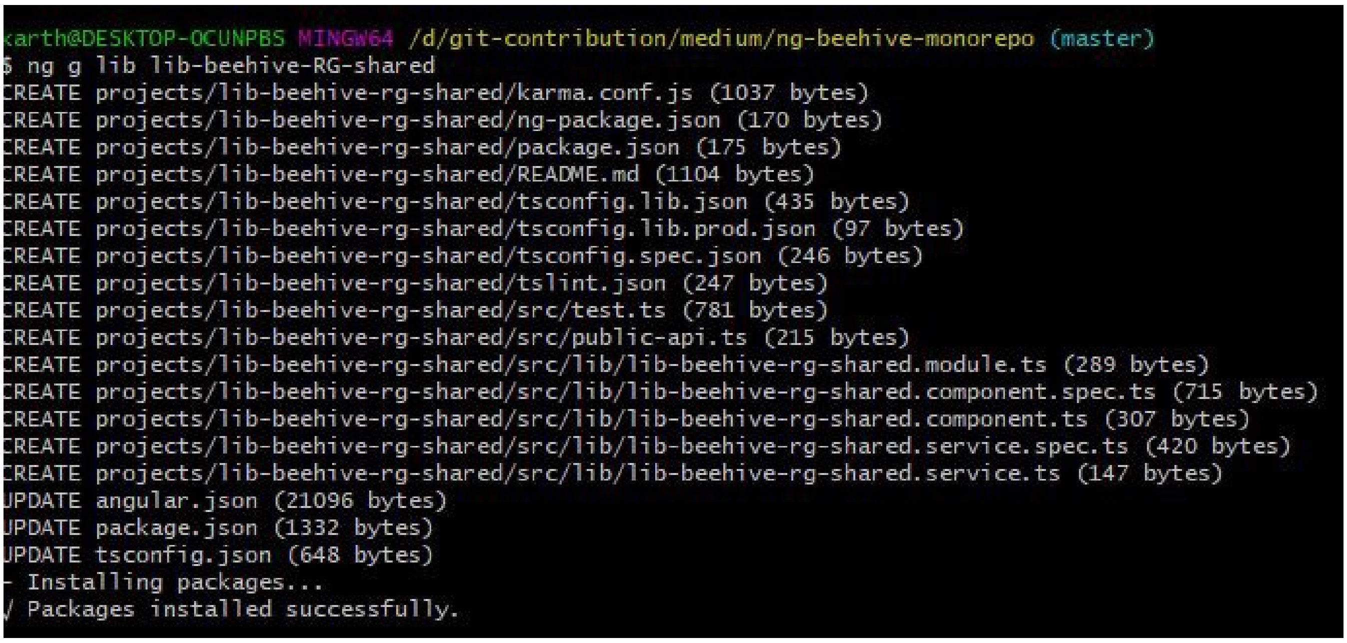


```
CREATE projects/beehive-RGB/src/test.ts (753 bytes)
CREATE projects/beehive-RGB/src/assets/.gitkeep (0 bytes)
CREATE projects/beehive-RGB/src/environments/environment.prod.ts (51 bytes)
CREATE projects/beehive-RGB/src/environments/environment.ts (662 bytes)
CREATE projects/beehive-RGB/src/app/app.module.ts (314 bytes)
CREATE projects/beehive-RGB/src/app/app.component.html (25725 bytes)
CREATE projects/beehive-RGB/src/app/app.component.spec.ts (957 bytes)
CREATE projects/beehive-RGB/src/app/app.component.ts (215 bytes)
CREATE projects/beehive-RGB/src/app/app.component.css (0 bytes)
CREATE projects/beehive-RGB/e2e/protractor.conf.js (808 bytes)
CREATE projects/beehive-RGB/e2e/tsconfig.json (226 bytes)
CREATE projects/beehive-RGB/e2e/src/app.e2e-spec.ts (644 bytes)
CREATE projects/beehive-RGB/e2e/src/app.po.ts (301 bytes)
UPDATE angular.json (4067 bytes)
UPDATE package.json (1252 bytes)
- Installing packages...
√ Packages installed successfully.
```

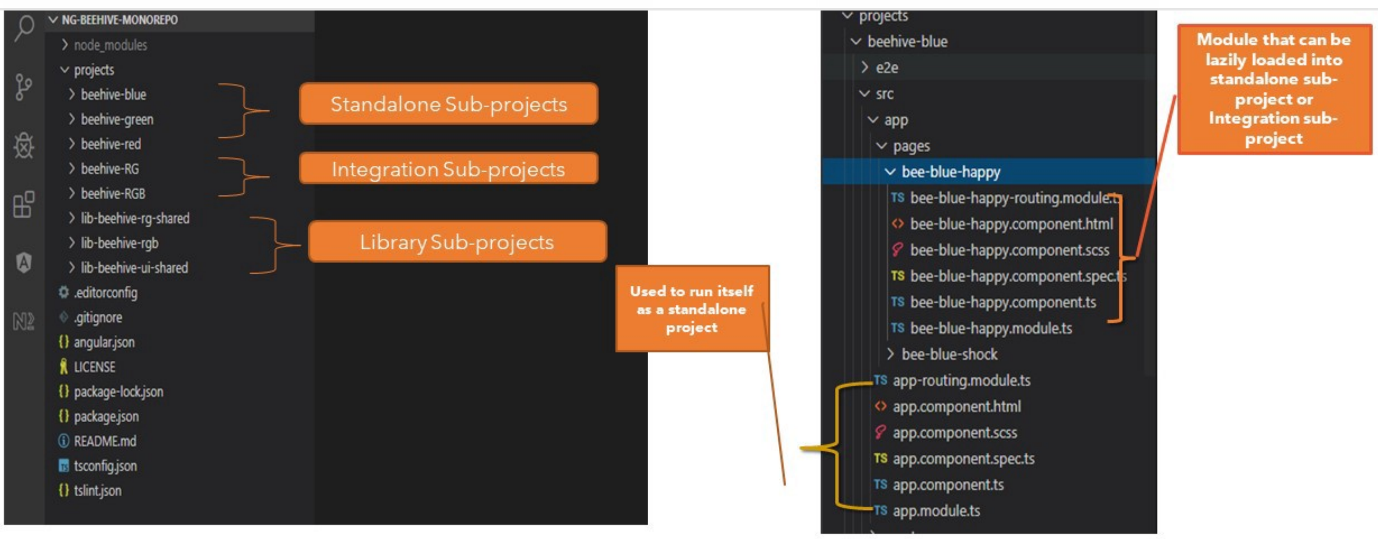
Sample output in the CLI when we create our first sub-application

5. Create Library project(s):

```
//Create library project lib-beehive-RG-shared
ng g lib lib-beehive-RG-shared

//Create library project lib-beehive-UI-shared
ng g lib lib-beehive-UI-shared

//Create library project lib-beehive-RGB-shared
ng g lib lib-beehive-RGB
```



```
karth@DESKTOP-OCUNPBS MINGW64 /d/git-contribution/medium/ng-beehive-monorepo (master)
$ ng g lib lib-beehive-RG-shared
CREATE projects/lib-beehive-rg-shared/karma.conf.js (1037 bytes)
CREATE projects/lib-beehive-rg-shared/ng-package.json (170 bytes)
CREATE projects/lib-beehive-rg-shared/package.json (175 bytes)
CREATE projects/lib-beehive-rg-shared/README.md (1104 bytes)
CREATE projects/lib-beehive-rg-shared/tsconfig.lib.json (435 bytes)
CREATE projects/lib-beehive-rg-shared/tsconfig.lib.prod.json (97 bytes)
CREATE projects/lib-beehive-rg-shared/tsconfig.spec.json (246 bytes)
CREATE projects/lib-beehive-rg-shared/tslint.json (247 bytes)
CREATE projects/lib-beehive-rg-shared/src/test.ts (781 bytes)
CREATE projects/lib-beehive-rg-shared/src/public-api.ts (215 bytes)
CREATE projects/lib-beehive-rg-shared/src/lib/lib-beehive-rg-shared.module.ts (289 bytes)
CREATE projects/lib-beehive-rg-shared/src/lib/lib-beehive-rg-shared.component.spec.ts (715 bytes)
CREATE projects/lib-beehive-rg-shared/src/lib/lib-beehive-rg-shared.component.ts (307 bytes)
CREATE projects/lib-beehive-rg-shared/src/lib/lib-beehive-rg-shared.service.spec.ts (420 bytes)
CREATE projects/lib-beehive-rg-shared/src/lib/lib-beehive-rg-shared.service.ts (147 bytes)
UPDATE angular.json (21096 bytes)
UPDATE package.json (1332 bytes)
UPDATE tsconfig.json (648 bytes)
- Installing packages...
√ Packages installed successfully.
```



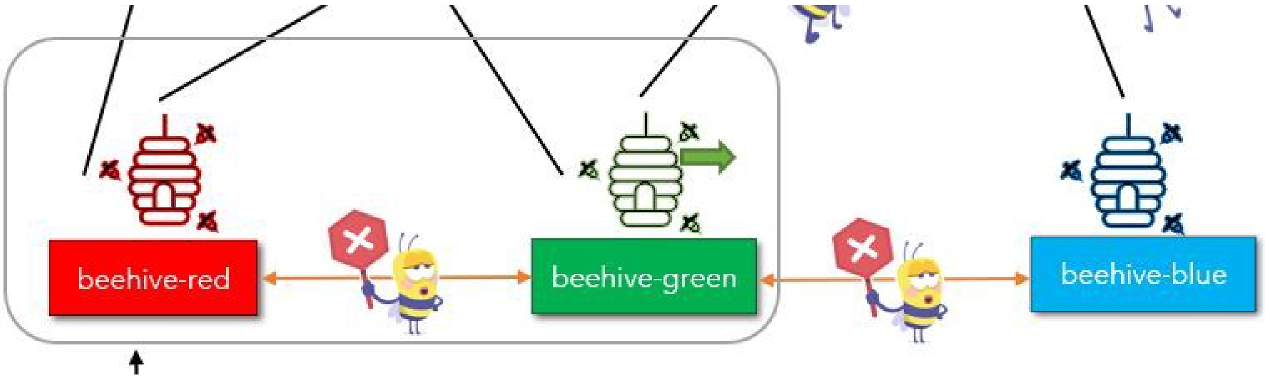
ng-beehive-monorepo workspace after executing all the above-mentioned angular CLI commands

Now that we have successfully created an angular workspace with multiple sub-projects before we proceed any further, lets us quickly go through some quick rules associated dependency hierarchy.

Project Dependency and Code sharing rules between projects:

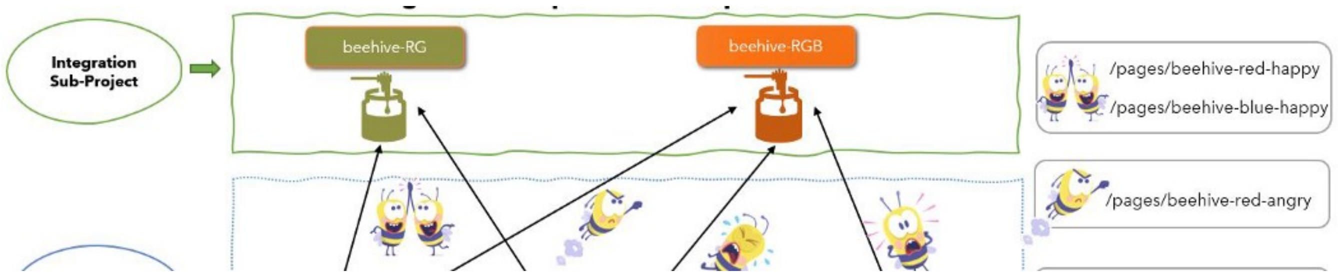
As mentioned previously, the Monorepo pattern allows us to keep multiple applications code into one single repository which has its own pros and cons. Although it is a really big topic to discuss, I would highly recommend reading the Angular Enterprise Monorepo pattern by Victor Savkin and his team which details advantages and limitations. They also developed an awesome library called NX which allows us to apply advanced techniques to build large scale enterprise-level angular applications that adhere to monorepo patterns. That being said, in our made-up beehive project, here are some of the dependency hierarchy rules that I came up:

1. Components/modules/directives/pipes etc., that are declared in standalone projects `beehive-red`, `beehive-blue`, `beehive-green` shouldn't be imported directly instead rely on the lib-projects which allows us to avoid circular dependency. This may be completely defiant from the whole concept of monorepo pattern but I felt it makes life a little easier if we have certain rules and standards set in place from making the applications dependency hierarchy over-complicated over the period of time. Also, isolating the standalone sub-projects from each other allows multiple



Avoid direct importing of files between these standalone projects.

2. Use integration sub-project to combine functionality from the various standalone project by leveraging the Angular Router to lazy load the modules or webpack import() for components respectively.



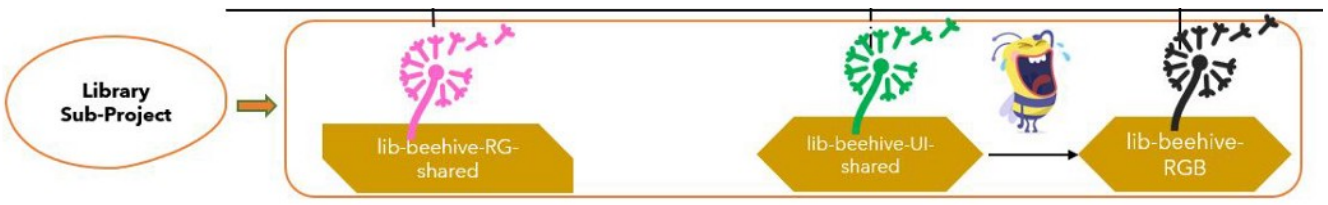
Use Angular Router or webpack import option to lazyload modules or components into integration projects.

3. All the UI components which are primarily a presentational component(eg., Multi-select Combobox, Grids, reusable form elements, etc.,) and don't have business functionality (a.ka. dumb components) should be added as part of `lib-beehive-UI-shared`

4. Any of the core app-level functionality which may be used in all standalone or integrational projects like loggers, base classes, directives, functional feature components, which may be useful should be part of `lib-beehive-RGB-shared`.

5. Any external libraries eg., Angular Material, ng-bootstrap, or ag-grid may be added as part of `lib-beehive-RGB` and exported via a module defined in `lib-beehive-RGB` . It may not happen quite often but in a situation where the support for the library is no longer available or there is a reason to migrate to a different library, it becomes a nightmare to identify all the references and change accordingly if all sub-projects

instead of directly referencing. This allows us to migrate to a newer or different library without impacting enter application(s). However, writing wrappers is not always a feasible option but its a design decision to be made as a team.



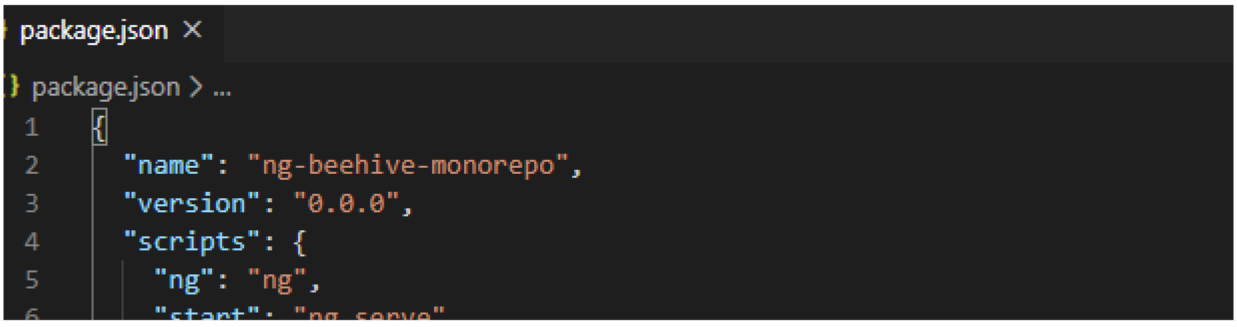
UI Modules/components defined in lib-beehive-UI-shared are imported into lib-beehive-RGB's module and exported further to be used in standalone or integration projects. lib-beehive-RG-shared as business-specific functionalities that are be shared only between beehive-red and beehive-blue

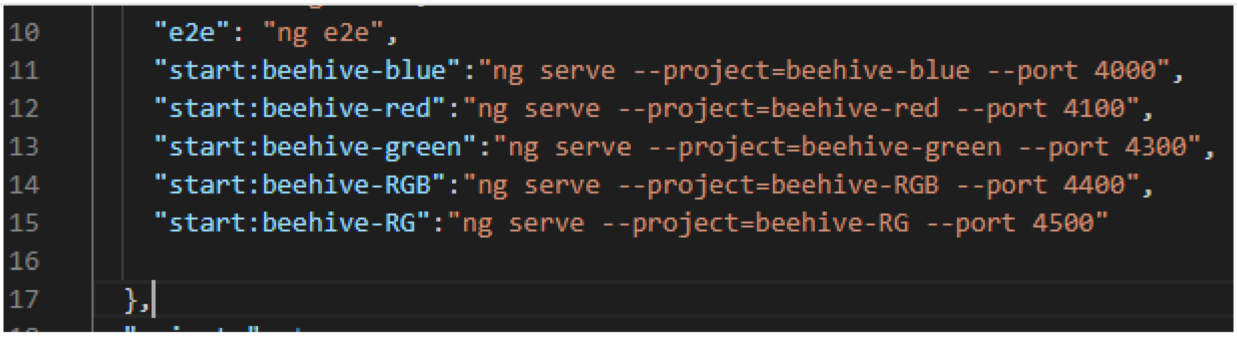
6. If there are any business-specific functionalities that are needed to be shared **only** between a subset of sub-projects (e.g., beehive-green and beehive-red) then we can use a Library project similar to `lib-beehive-RG-shared`

Now that we got some idea about the dependency hierarchy and rules to follow, let's go ahead and see how we can run these projects independently:

update the package.json to add these commands:

```
"start:beehive-blue":"ng serve --project=beehive-blue --port 4000",

"start:beehive-red":"ng serve --project=beehive-red --port 4100",

"start:beehive-green":"ng serve --project=beehive-green --port
4300",

"start:beehive-RGB":"ng serve --project=beehive-RGB --port 4400",

"start:beehive-RG":"ng serve --project=beehive-RG --port 4500"
```


```
package.json ×
{} package.json > ...
{
  "name": "ng-beehive-monorepo",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
```

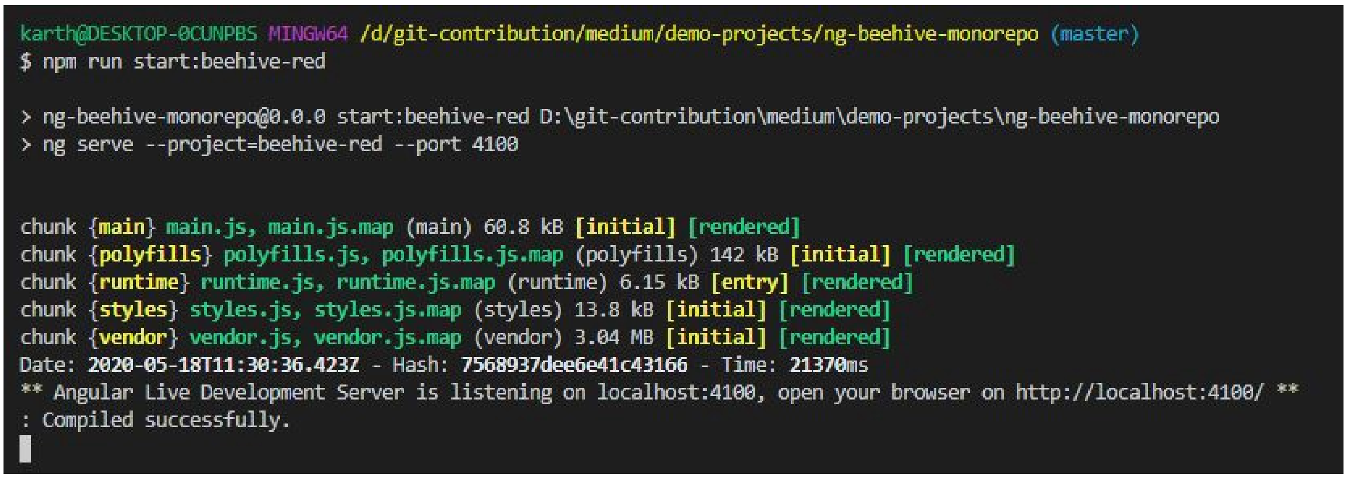


```
    "e2e": "ng e2e",
    "start:beehive-blue":"ng serve --project=beehive-blue --port 4000",
    "start:beehive-red":"ng serve --project=beehive-red --port 4100",
    "start:beehive-green":"ng serve --project=beehive-green --port 4300",
    "start:beehive-RGB":"ng serve --project=beehive-RGB --port 4400",
    "start:beehive-RG":"ng serve --project=beehive-RG --port 4500"

  },
```

Code snippet of package.json after adding custom npm commands

To start an application eg., beehive-blue and the console execute below command

```
npm run start:beehive-red
```



```
karth@DESKTOP-0CUNPBS MINGW64 /d/git-contribution/medium/demo-projects/ng-beehive-monorepo (master)
$ npm run start:beehive-red

> ng-beehive-monorepo@0.0.0 start:beehive-red D:\git-contribution\medium\demo-projects\ng-beehive-monorepo
> ng serve --project=beehive-red --port 4100


chunk {main} main.js, main.js.map (main) 60.8 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 142 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 13.8 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.04 MB [initial] [rendered]
Date: 2020-05-18T11:30:36.423Z - Hash: 7568937dee6e41c43166 - Time: 21370ms
** Angular Live Development Server is listening on localhost:4100, open your browser on http://localhost:4100/ **
: Compiled successfully.
```

Console output when beehive-red dev-server is launched.

As mentioned in the above console logs, you can use http://localhost:4100 to launch the application in the browser. At this moment you will see the static default template content that angular ships when creating the application project via CLI.

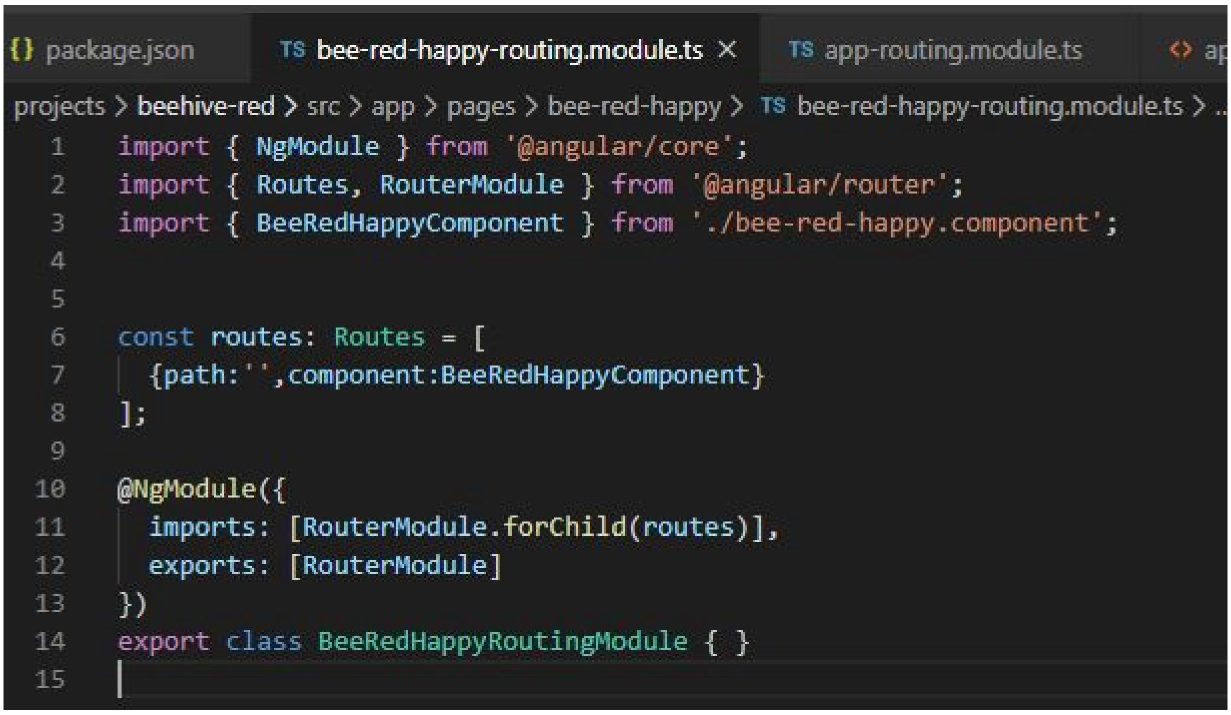
Let's go ahead and add some modules and routable components in beehive-red project:

```
//create module bee-red-happy along with routing
ng g m pages/bee-red-happy --routing --project=beehive-red

//console ouput
CREATE projects/beehive-red/src/app/pages/bee-red-happy/bee-red-
happy-routing.module.ts (255 bytes)
```

```
//create bee-blue-happy component
ng g c pages/bee-red-happy --project=beehive-red

//console output
CREATE projects/beehive-red/src/app/pages/bee-red-happy/bee-red-
happy.component.html (28 bytes)
CREATE projects/beehive-red/src/app/pages/bee-red-happy/bee-red-
happy.component.spec.ts (665 bytes)
CREATE projects/beehive-red/src/app/pages/bee-red-happy/bee-red-
happy.component.ts (302 bytes)
CREATE projects/beehive-red/src/app/pages/bee-red-happy/bee-red-
happy.component.scss (0 bytes)
UPDATE projects/beehive-red/src/app/pages/bee-red-happy/bee-red-
happy.module.ts (388 bytes)
```

Update the routes to add a mapping between /beehive-red-happy path to BeeRedHappyComponent

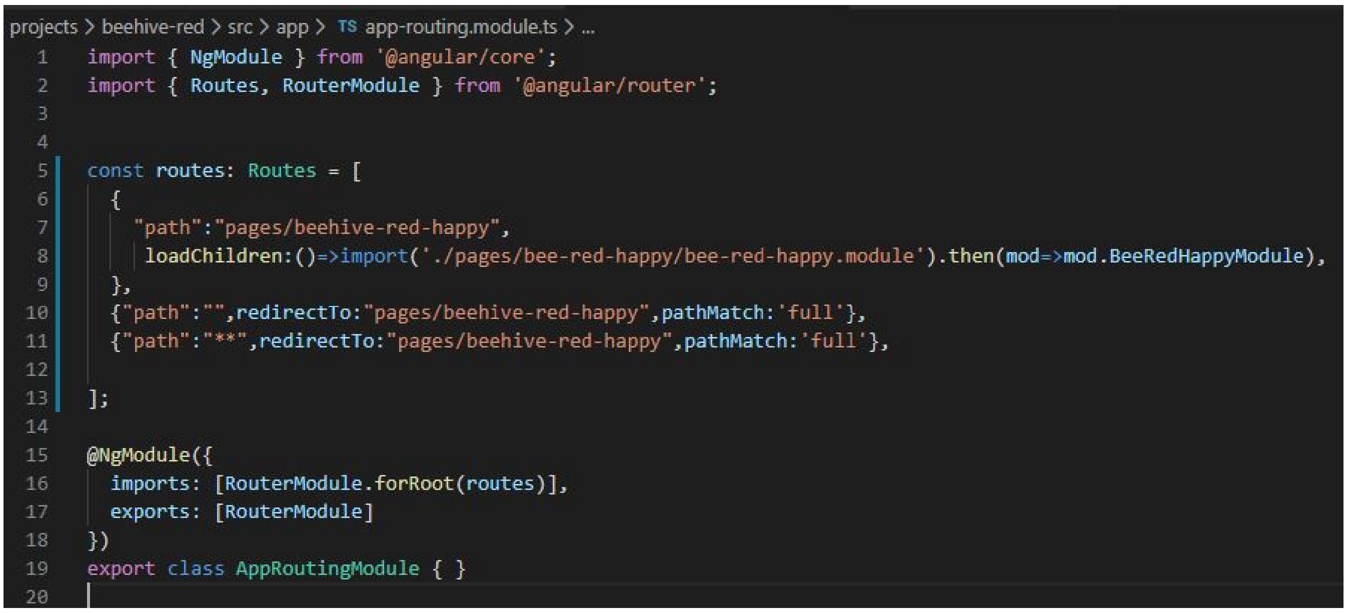
```
const routes: Routes = [
  {path:'',component:BeeRedHappyComponent}
];
```


{} package.json
TS bee-red-happy-routing.module.ts ×
TS app-routing.module.ts
<> ap
projects > beehive-red > src > app > pages > bee-red-happy > TS bee-red-happy-routing.module.ts > ...

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { BeeRedHappyComponent } from './bee-red-happy.component';


const routes: Routes = [
  {path:'',component:BeeRedHappyComponent}
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class BeeRedHappyRoutingModule { }

```

Code snippet for BeeRedHappyRoutingModule

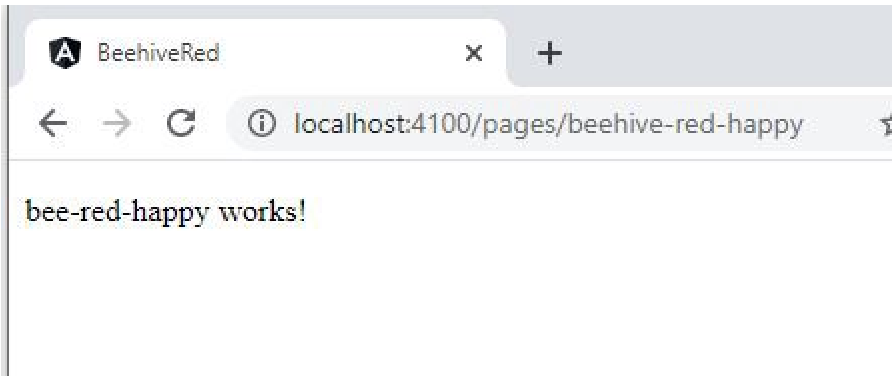
```
const routes: Routes = [

{"path":"pages/beehive-red-happy",loadChildren:
()=>import('./pages/bee-red-happy/bee-red-
happy.module').then(mod=>mod.BeeRedHappyModule)},

{"path":"",redirectTo:"pages/beehive-red-happy",pathMatch:'full'},

{"path":"**",redirectTo:"pages/beehive-red-happy",pathMatch:'full'}
];
```



projects > beehive-red > src > app > TS app-routing.module.ts > ...

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';


const routes: Routes = [
  {
    "path":"pages/beehive-red-happy",
    loadChildren:()=>import('./pages/bee-red-happy/bee-red-happy.module').then(mod=>mod.BeeRedHappyModule),
  },
  {"path":"",redirectTo:"pages/beehive-red-happy",pathMatch:'full'},
  {"path":"**",redirectTo:"pages/beehive-red-happy",pathMatch:'full'},

];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

AppRoutingModule of beehive-red project is updated to lazily load beehive-red-happy page. This allows us to run the application as a standalone project.



bee-red-happy works!

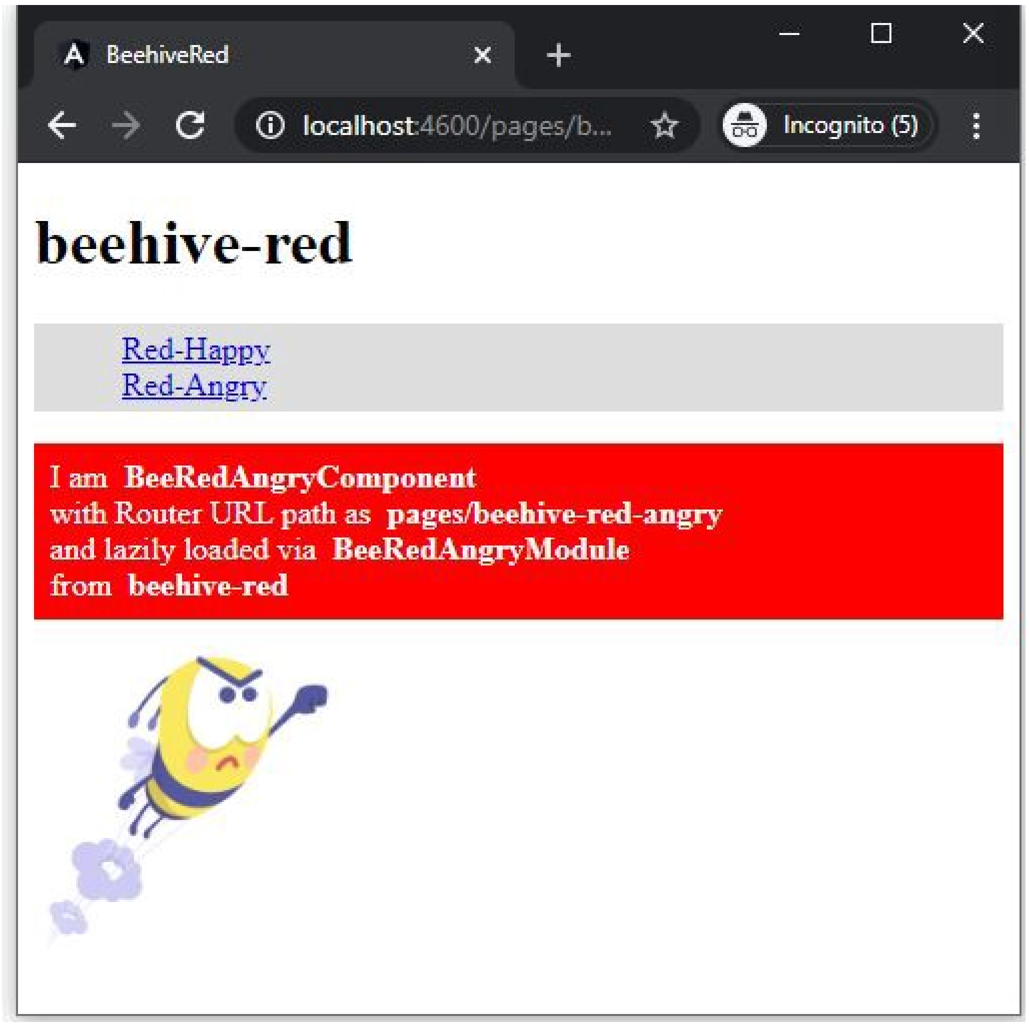
When we launch http://localhost:4100, beehive-red standalone project is running and we can see the component is loaded appropriately.

be re-used in almost all projects within the `ng-beehive-monorepo`, we can add this as part of `lib-beehive-ui-shared`

Here's what our presentation component takes in as the input and renders a simple HTML content:

```
Input:

Component Name

Router URL path

Name of the Module that the component belongs to

Name of the project that component name and module it belongs

Output:

Hello,

I am {{ComponentName}} with Router URL path as {{urlPath}} and
belong to {{module}} of {{project}}

//CLI Command:
ng g m features/component-identifier --project=lib-beehive-UI-shared

//Console output
CREATE projects/lib-beehive-ui-shared/src/lib/features/component-
identifier/component-identifier.module.ts (205 bytes)

//CLI Command:
ng g c features/component-identifier --project=lib-beehive-UI-shared

//Console output
CREATE projects/lib-beehive-ui-shared/src/lib/features/component-
identifier/component-identifier.component.html (35 bytes)
CREATE projects/lib-beehive-ui-shared/src/lib/features/component-
identifier/component-identifier.component.spec.ts (720 bytes)
CREATE projects/lib-beehive-ui-shared/src/lib/features/component-
identifier/component-identifier.component.ts (331 bytes)
CREATE projects/lib-beehive-ui-shared/src/lib/features/component-
identifier/component-identifier.component.scss (0 bytes)
UPDATE projects/lib-beehive-ui-shared/src/lib/features/component-
identifier/component-identifier.module.ts (314 bytes)
```

time on making the UI look aesthetic, as my main goal was to give some idea how multiple-subprojects grouped together and maintain the code in one single repository.



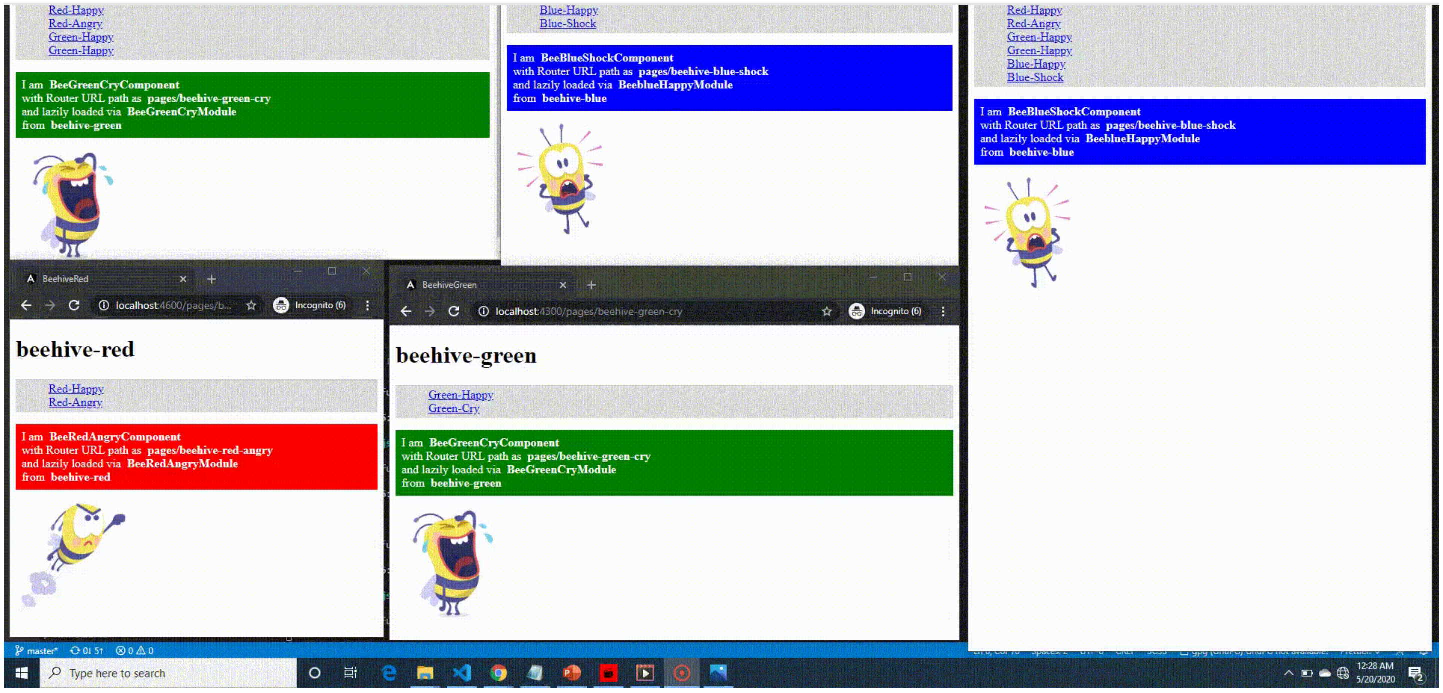
beehive-red

Red-Happy
Red-Angry

I am **BeeRedAngryComponent**
with Router URL path as **pages/beehive-red-angry**
and lazily loaded via **BeeRedAngryModule**
from **beehive-red**

Added reusable UI component into `lib-beehive-ui-shared` and used this component inside **beehive-red sub-project**

After repeating the same approach for all the standalone and integration sub-projects. Here is how it looks when you run all the sub-projects independently.



I am BeeGreenCryComponent
with Router URL path as pages/beehive-green-cry
and lazily loaded via BeeGreenCryModule
from beehive-green

I am BeeBlueShockComponent
with Router URL path as pages/beehive-blue-shock
and lazily loaded via BeeblueHappyModule
from beehive-blue

I am BeeBlueShockComponent
with Router URL path as pages/beehive-blue-shock
and lazily loaded via BeeblueHappyModule
from beehive-blue

beehive-red

I am BeeRedAngryComponent
with Router URL path as pages/beehive-red-angry
and lazily loaded via BeeRedAngryModule
from beehive-red

beehive-green

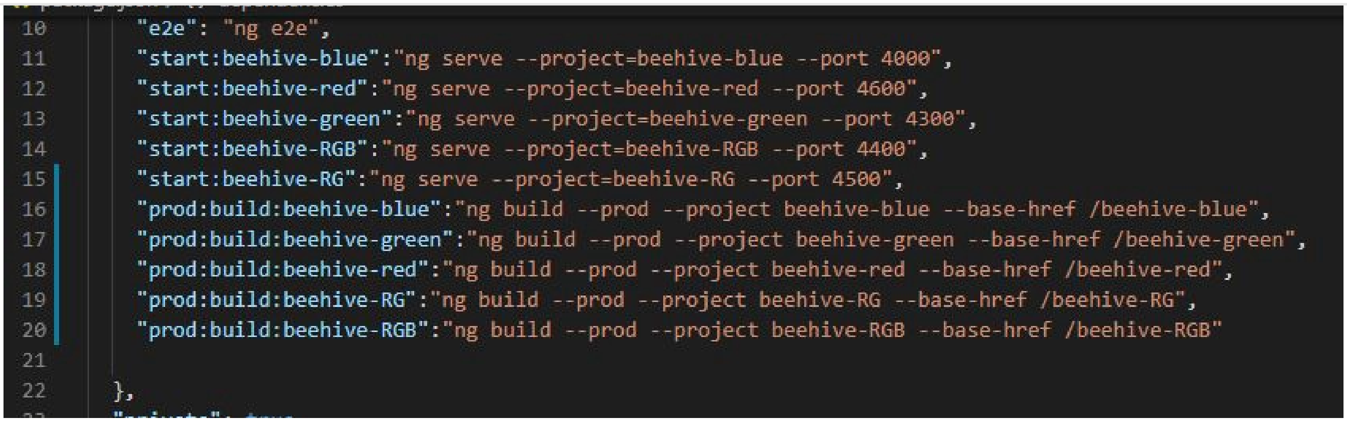
I am BeeGreenCryComponent
with Router URL path as pages/beehive-green-cry
and lazily loaded via BeeGreenCryModule
from beehive-green

Running standalone and integration sub-projects independently while all the code base is from one single Angular Workspace (monorepo).

Finally, let's see how we can make a prod build for all these sub-projects individually:

You can update package.json with below command:

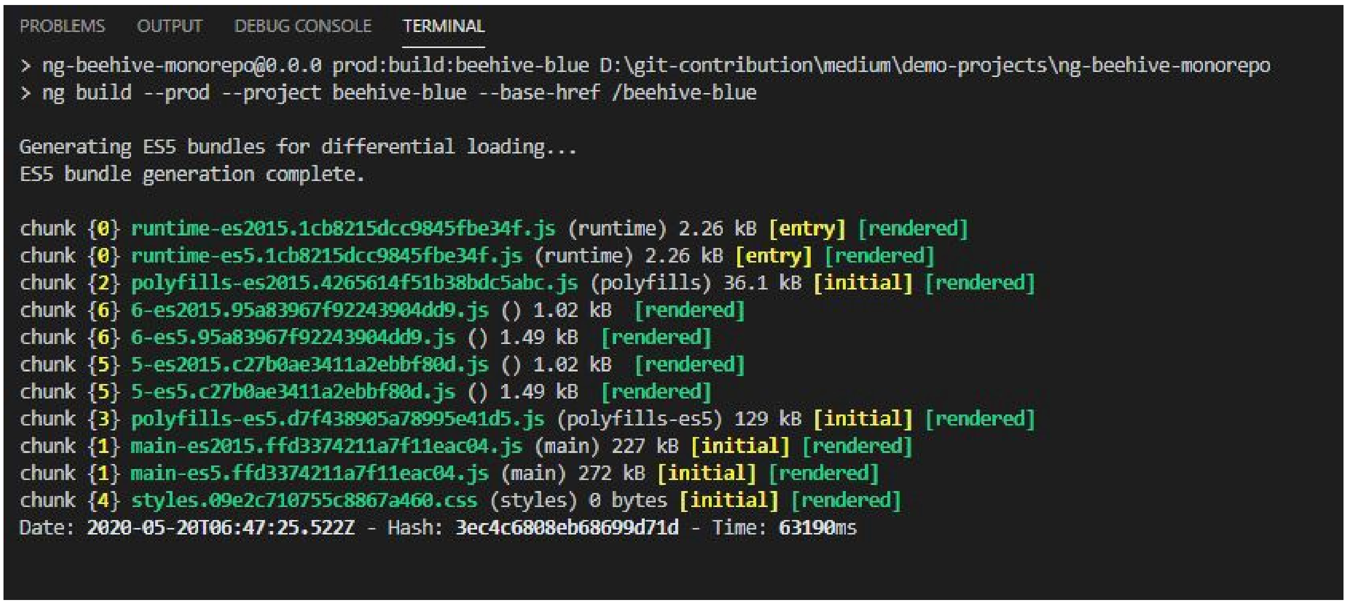
```
"prod:build:beehive-blue":"ng build --prod --project beehive-blue --
base-href /beehive-blue",

"prod:build:beehive-green":"ng build --prod --project beehive-green
--base-href /beehive-green",

"prod:build:beehive-red":"ng build --prod --project beehive-red --
base-href /beehive-red",

"prod:build:beehive-RG":"ng build --prod --project beehive-RG --
base-href /beehive-RG",

"prod:build:beehive-RGB":"ng build --prod --project beehive-RGB --
base-href /beehive-RGB"
```



```
    "e2e": "ng e2e",
    "start:beehive-blue":"ng serve --project=beehive-blue --port 4000",
    "start:beehive-red":"ng serve --project=beehive-red --port 4600",
    "start:beehive-green":"ng serve --project=beehive-green --port 4300",
    "start:beehive-RGB":"ng serve --project=beehive-RGB --port 4400",
    "start:beehive-RG":"ng serve --project=beehive-RG --port 4500",
    "prod:build:beehive-blue":"ng build --prod --project beehive-blue --base-href /beehive-blue",
    "prod:build:beehive-green":"ng build --prod --project beehive-green --base-href /beehive-green",
    "prod:build:beehive-red":"ng build --prod --project beehive-red --base-href /beehive-red",
    "prod:build:beehive-RG":"ng build --prod --project beehive-RG --base-href /beehive-RG",
    "prod:build:beehive-RGB":"ng build --prod --project beehive-RGB --base-href /beehive-RGB"

  },
```

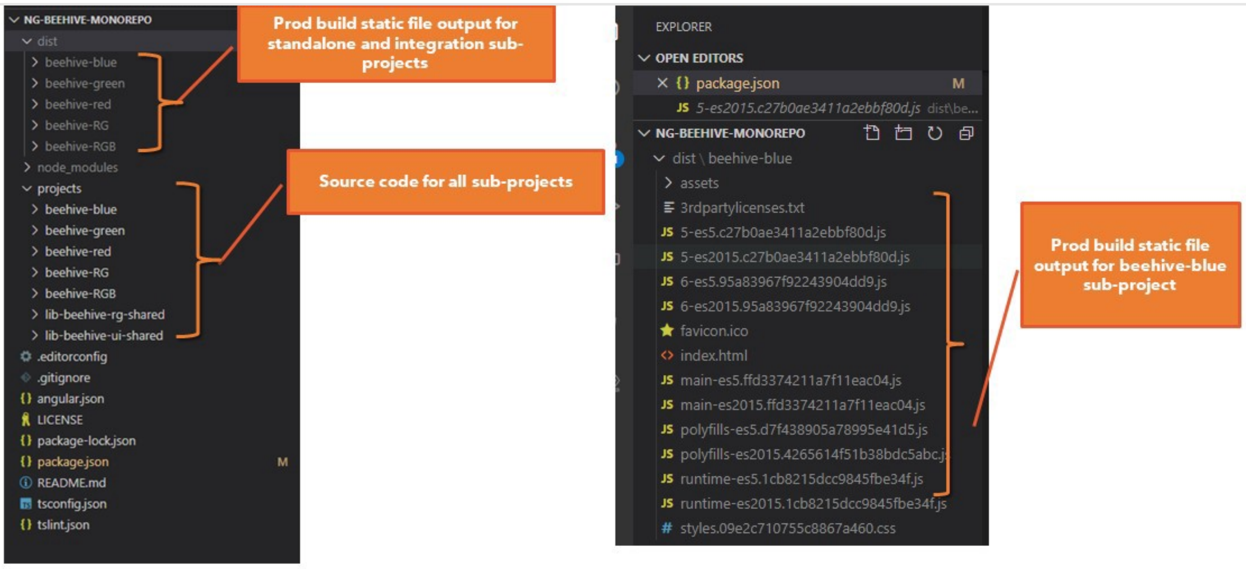
updated package.json to have prod build script

```
executing below command in the CLI will generate artificates for the
prod deployment:

npm run prod:build:beehive-blue
```



```
> ng-beehive-monorepo@0.0.0 prod:build:beehive-blue D:\git-contribution\medium\demo-projects\ng-beehive-monorepo
> ng build --prod --project beehive-blue --base-href /beehive-blue

Generating ES5 bundles for differential loading...
ES5 bundle generation complete.

chunk {0} runtime-es2015.1cb8215dcc9845fbe34f.js (runtime) 2.26 kB [entry] [rendered]
chunk {0} runtime-es5.1cb8215dcc9845fbe34f.js (runtime) 2.26 kB [entry] [rendered]
chunk {2} polyfills-es2015.4265614f51b38bdc5abc.js (polyfills) 36.1 kB [initial] [rendered]
chunk {6} 6-es2015.95a83967f92243904dd9.js () 1.02 kB  [rendered]
chunk {6} 6-es5.95a83967f92243904dd9.js () 1.49 kB  [rendered]
chunk {5} 5-es2015.c27b0ae3411a2ebbf80d.js () 1.02 kB  [rendered]
chunk {5} 5-es5.c27b0ae3411a2ebbf80d.js () 1.49 kB  [rendered]
chunk {3} polyfills-es5.d7f438905a78995e41d5.js (polyfills-es5) 129 kB [initial] [rendered]
chunk {1} main-es2015.ffd3374211a7f11eac04.js (main) 227 kB [initial] [rendered]
chunk {1} main-es5.ffd3374211a7f11eac04.js (main) 272 kB [initial] [rendered]
chunk {4} styles.09e2c710755c8867a460.css (styles) 0 bytes [initial] [rendered]
Date: 2020-05-20T06:47:25.522Z - Hash: 3ec4c6808eb68699d71d - Time: 63190ms
```

prod build console output for beehive-blue sub-project


NG-BEEHIVE-MONOREPO
dist
beehive-blue
beehive-green
beehive-red
beehive-RG
beehive-RGB
node_modules
projects
beehive-blue
beehive-green
beehive-red
beehive-RG
beehive-RGB
lib-beehive-rg-shared
lib-beehive-ui-shared
.editorconfig
.gitignore
angular.json
LICENSE
package-lock.json
package.json M
README.md
tsconfig.json
tslint.json
Prod build static file output for standalone and integration sub-projects
Source code for all sub-projects
EXPLORER
OPEN EDITORS
package.json M
5-es2015.c27b0ae3411a2ebbf80d.js dist\be...
NG-BEEHIVE-MONOREPO
dist \ beehive-blue
assets
3rdpartylicenses.txt
5-es5.c27b0ae3411a2ebbf80d.js
5-es2015.c27b0ae3411a2ebbf80d.js
6-es5.95a83967f92243904dd9.js
6-es2015.95a83967f92243904dd9.js
favicon.ico
index.html
main-es5.ffd3374211a7f11eac04.js
main-es2015.ffd3374211a7f11eac04.js
polyfills-es5.d7f438905a78995e41d5.js
polyfills-es2015.4265614f51b38bdc5abc.js
runtime-es5.1cb8215dcc9845fbe34f.js
runtime-es2015.1cb8215dcc9845fbe34f.js
styles.09e2c710755c8867a460.css
Prod build static file output for beehive-blue sub-project

Angular production build outputs for all standalone and integration sub-projects using CLI commands.

All the code for this project can be found below:

kgotgit/ng-beehive-monorepo

This project was generated with Angular CLI version 9.1.6. Run ng serve for a dev server. Navigate to...

github.com

Summary:

To summarize we walkthrough:

1. How to create an angular workspace with multiple sub-projects which are categorized into standalone, integration and library sub-projects
2. We went through the list of CLI commands to generate the sub-projects and ways to generate the reusable UI components in library projects.

4. How to run the applications independently using the same angular workspace and also CLI commands for generating production artifacts for each sub-projects separately.

This was a very long post for me, I hope you guys liked it. Please do clap and share it with your friends and colleagues. If there any questions, anything is wrong or needs modification, please do comment.

Resources:

1. All the images were generated using subscribed Microsoft PowerPoint. Icons and stickers images were part of the PowerPoint's shapes, icons.

2. https://angular.io/guide/workspace-config

3. https://go.nrwl.io/angular-enterprise-monorepo-patterns-new-book

Angular Monorepo Angular Cli Web Development Architecture