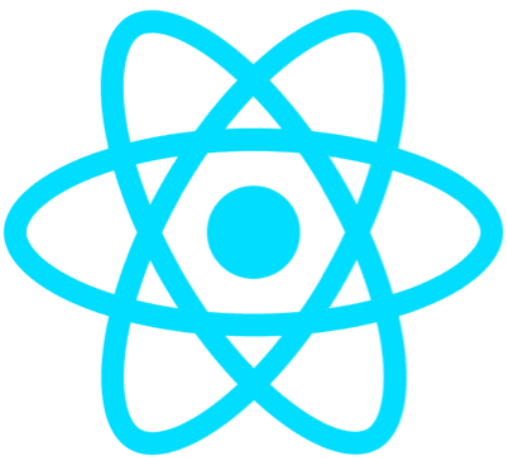
React Workshop

JumpStart to Best Practices



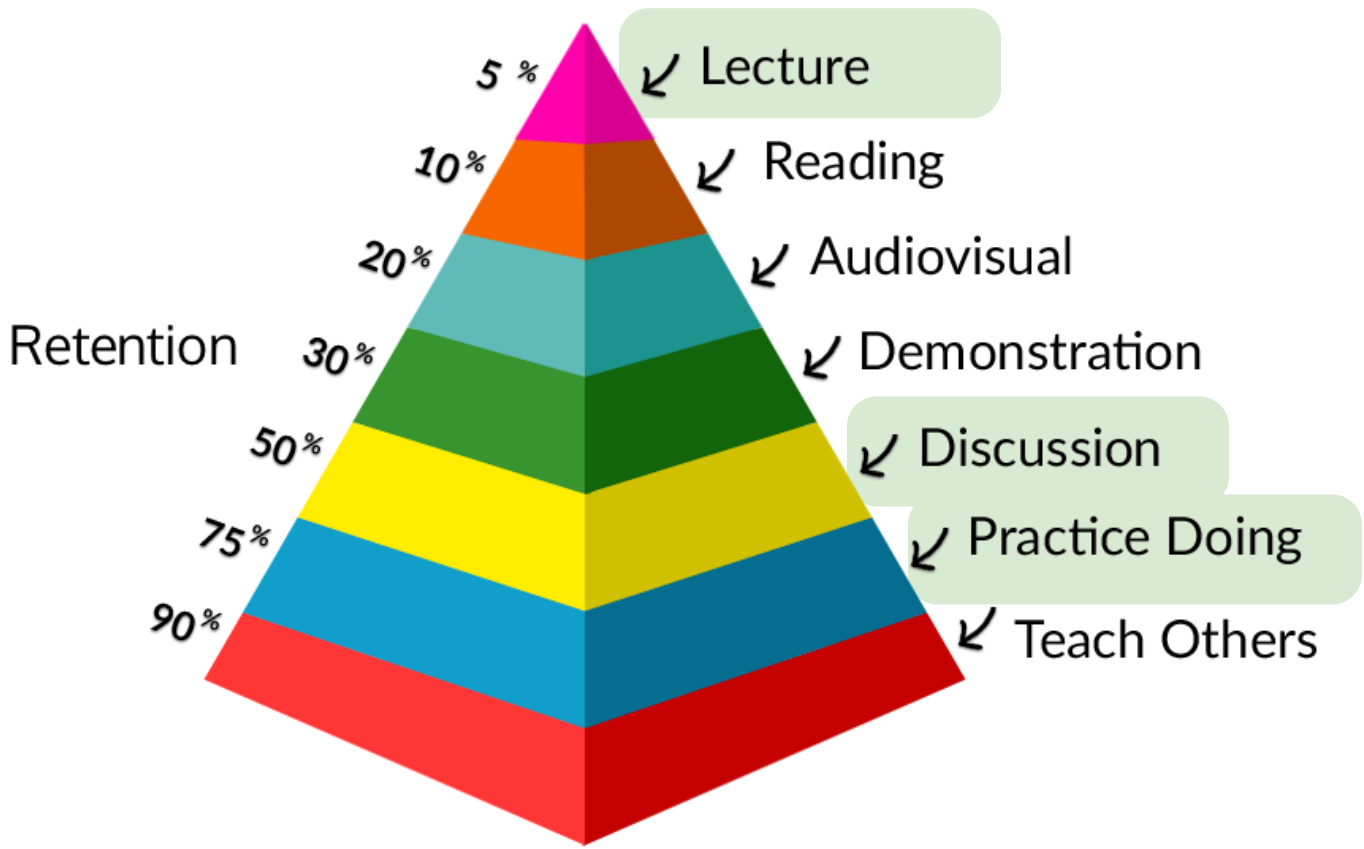
Standards for 2020 and beyond...

Speaker notes

9 Hands-on Coding Labs

React Workshop

Lectures + Coding Labs



React Workshop

Best practices on using *Monorepo*
Best practices on using *React + Typescript*
Patterns for *Separation of Concerns*

***Standards** on React Components*

This is **not** a beginner tutorial on *ES6*
This is **not** a beginner tutorial on *React*

A Brief Review

- Types (Explicit, Implicit, Custom)
- Interfaces
- Generics
- Tuples

- Destructuring
- Object Literals
- Spread Operator
- Rest Operator

Explicit vs. **Implicit** Types

Type information can be specified for variables

```
1 const age  : number = 16;
2 const name : string = 'Thomas';
3
4 function upperCase(name: string): string {
5   return name.toUpperCase();
6 }
```

Use implicit types is a best practice:

```
1 const age  = 16;
2 const name = 'Thomas';
3
4 function upperCase(name = ''): string {
5   return name.toUpperCase();
6 }
```

Explicit vs. **Implicit** Types

Explicit type required for empty array initialization:

```
1 // internal cache to be populated later
2 let contacts: Contact[] = [];
```

Custom Types

```
1  function fetchContacts(params, options) {
2    const name = (params && params.userName) ? params.userName : '';
3    const url = `${options.apiEndPoint}&name=${name}`;
4
5    console.log(`GET ${url}`);
6
7    return Promise.resolve(CONTACTS);
8  }
9
```

Improve by specifying **types**:

```
1  function fetchContacts(params: FetchParams, options: FetchOptions): Promise<Contact[]> {
2
3    const name = params.userName ? params.userName : '';
4    const url = `${options.apiEndPoint}&name=${name}`;
5
6    console.log(`GET ${url}`);
7
8    return Promise.resolve(CONTACTS);
9  }
10
```

Using **Interfaces**

```
1  export interface QueryParams  { userName: string; title      ?: string };
2  export interface QueryOptions { apiKey   : string; apiEndPoint : string };
3
4  function fetchContacts(
5       params : QueryParams,
6       options: QueryOptions): Promise<Contact[]> {
7
8    let results;
9    //....
10
11   return Promise.resolve(results || []);
12 }
```

Here ^ we clearly define the fields expected (or optional) and the types of each field

Function **Parameter Defaults**

```
getContacts(useCache, params: QueryParams): [] { ... }


getContacts(useCache = false, params: QueryParams = null): [] { ... }
```

```
getContacts(useCache = false, params: QueryParams): [] { ... }


 // This ^ is dangerous....

const params = {};
const contacts = getContacts(params);  // ????
```

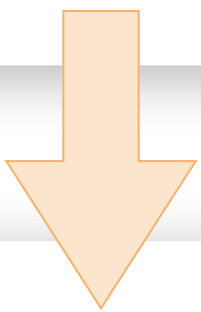
Using **Generics**

```
1   /**
2    * Load a list from the REST service...
3    */
4   getContacts(useCache = false, params: QueryParams = null): Promise {
5     let results;
6     //....
7
8     return Promise.resolve(results || []);
9   }
```

```
1
2   getContacts(useCache = false, params: QueryParams = null): Promise<Contact[]> {
3     let results;
4     //....
5
6     return Promise.resolve(results || []);
7   }
```

Using **Generics**

```
1 export const ContactsList: React.FC = () => {
2   const [service] = useState<ContactsService>() => new ContactsService());
3
4   return (...);
5 }
```



```
1 export const ContactsList: React.FC = () => {
2   const [service] = useState(() => new ContactsService());
3
4   return (...);
5 }
```

Speaker notes

Talk about why React.FC() is better than FC()
Talk about () => new ContactsService()
Talk about why using generic useState(...) is not needed

Using **Tuples**

```
1  export type ContactDetailsResult = [Contact, H.History<H.LocationState>];
2
3  /**
4   * Custom React Hook useful to load Contact details
5   */
6  export function useContactDetailHook(): ContactDetailsResult {
7    const { id } = useParams();
8    const history = useHistory();
9    const [service] = useState(() => new ContactsService());
10   const [contact, setContact] = useState<Contact>({} as Contact);
11
12   useEffect(() => {
13     service.getContactById(id).then(setContact);
14   }, [id]);
15
16   return [contact, history];
17 }
```

Using **Tuples**

Thomas Burleson
Jan 19 · 5 min read

The Power of Tuples

Learn about Typescript Tuples for React and Angular applications.

34

Destructuring

Destructure **Objects** to get values desired

```
1 const person = {
2  name: 'Harry',
3  city: 'Minneapolis', state: 'MN',
4  age: 75,
5  status: 'married'
6 };
7
8 const { name, age } = person;
```

Can do this with local **variables** or **function arguments**!

```
1 function fetchContacts(
2     { userName }: QueryParams,
3     options     : QueryOptions  ): Contact[] {
4
5  //...
6
7 }
```

Destructuring

Destructure **Arrays** to get values desired

```
1  const useHook = () => {
2    const first = 'thomas', last = 'burleson';
3    const updateName = (first, last) => {};
4
5    return ['Thomas', 'Burleson', updateName];
6  }
7
8  const results = useHook();
9  const last = results[1], updateFn = results[2];
10
11 // Or destructure
12
13 const [, last, updateFn] = useHook();
```

Notice how to skip an element during array destructuring.

Using Object Literals

ES5 Approach (old school)

```
interface Laptop { make: string; model: string, year: string; };

function getLaptop(make = '', model= '', year = ''): Laptop {
  return {
    make  : make,
    model : model,
    year  : year
  };
}
```

Using Object Literals

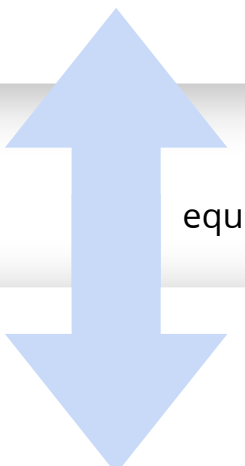
ES6 Approach (new school)

```
1 interface Laptop { make: string; model: string, year: string; };
2
3 function getLaptop(make = '', model= '', year = ''): Laptop {
4   return { make, model, year };
5 }
```

```
1 private loadContacts(params: QueryParams): Promise<Contact[]> {
2     const apiKey      = API_HEADERS;
3     const apiEndPoint = API_ENDPOINT;
4     return fetchContacts<Contact[]>(params, { apiEndPoint, apiKey });
5   }
```

Destructing + Object Literals

```
1 export class ContactsService {
2
3   constructor(private apiEndPoint: string, private apiKey: string) {}
4
5 }
```



equivalent

```
 1 export class ContactsService {
 2   private apiEndPoint: string;
 3   private apiKey: string;
 4
 5   constructor(apiEndPoint: string, apiKey: string) {
 6     this.apiEndPoint = apiEndPoint;
 7     this.apiKey      = apiKey;
 8   }
 9
10 }
```

Destructing + Object Literals

```
export class ContactsService {

  constructor(private apiEndPoint: string, private apiKey: string) {}


  /**
   * internal wrapper to fetch function...
   */
  private loadContacts(params?: QueryParams): Promise<Contact[]> {
    const { apiEndPoint, apiKey } = this;

    return fetchContacts<Contact[]>(
      params || ({} as QueryParams),
      { apiEndPoint, apiKey }
    );
  }
}
```

Lab 1: Adding Types

```
9   /**
10   * Internal state for singleton ContactsService
11   */
12  let contacts: Contact[] = [];
13
    You, 43 minutes ago | 1 author (You)
14  export interface QueryParams {
15    userName: string;
16    title?: string;
17  }
    You, 43 minutes ago | 1 author (You)
18  export interface QueryOptions {
19    apiKey: string;
20    apiEndPoint: string;
21  }
22
23  function fetchContacts({ userName }: QueryParams, options: QueryOptions): Promise<Contact[]> {
24    console.log(`GET ${options.apiEndPoint}&name=${userName || ''}`);
25
26    return Promise.resolve(CONTACTS);
27  }
28
    You, an hour ago | 1 author (You)
29  export class ContactsService {
30    constructor(private apiEndPoint: string, private apiKey: string) {}
31
32    /**
33     * Load a list from the REST service ...
34     */
35    getContacts(useCache = false, params: QueryParams = null): Promise<Contact[]> {
36      const goFetch = () => {
37        params = params || ({} as QueryParams);
38        return this.loadContacts(params).then(list => {
39          return (contacts = assignUIDs(list));
40        });
41      };
```

contacts.service.ts

Mono-Repository

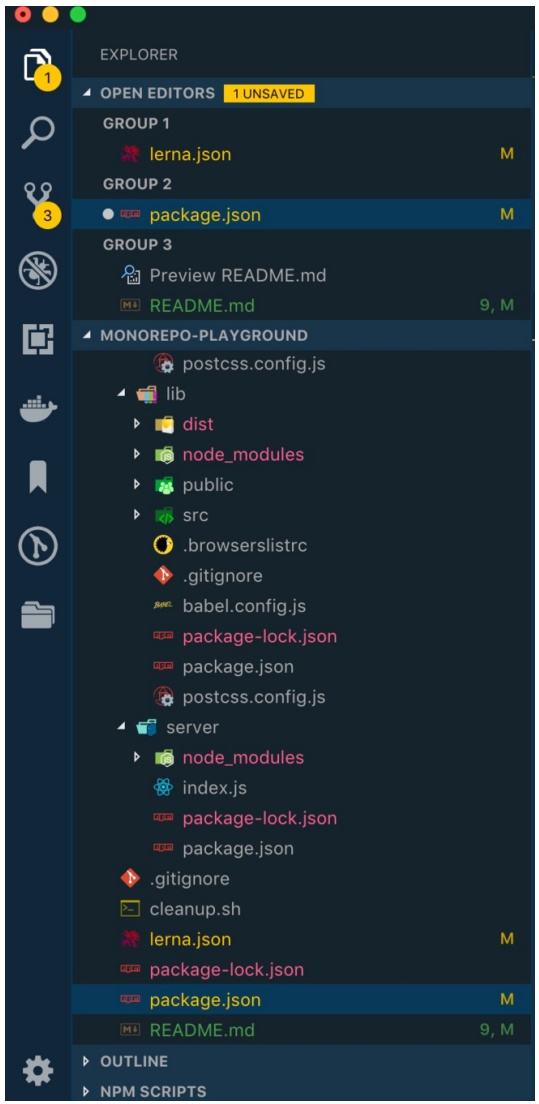
- Monorepo is a **single** Git **repository** using a workspace
- Workspace **provides** tools and conventions
- Tools **include** compilers, code generators, linters, testing, etc...
- Monorepo **contains** multiple projects (libs, apps, server, etc)

Confusion

Developers can have many types of mono-repos:

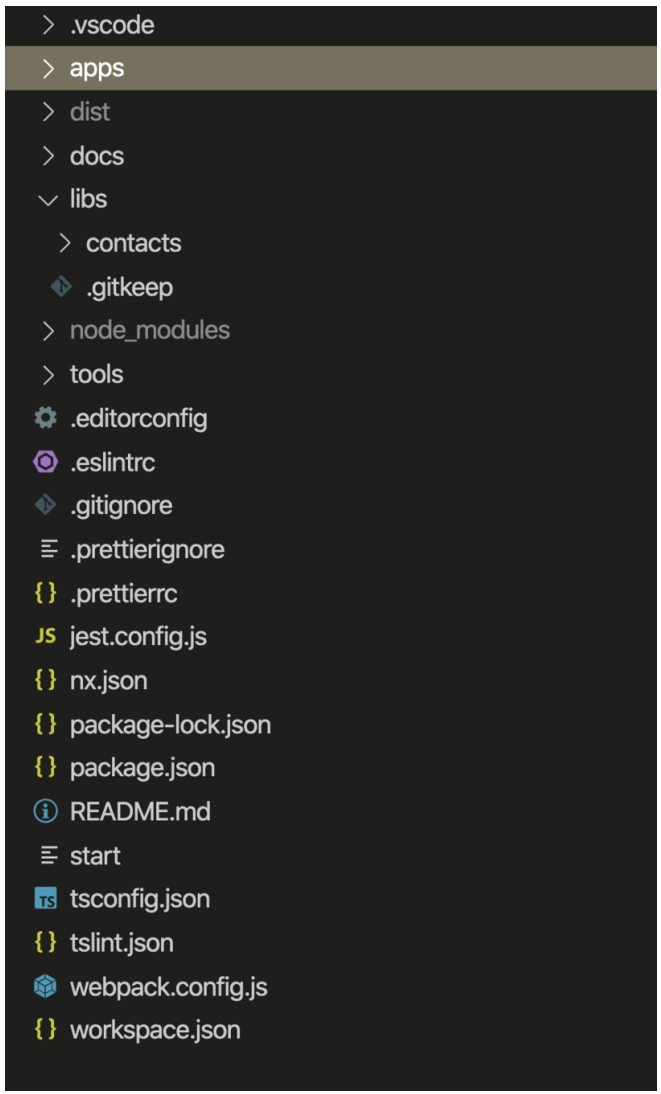
- **Lerna** Workspaces
- ~~Angular~~ Workspaces
- **Nx** Workspaces

Lerna Workspace



1. Hosts one or more projects or **packages**.
2. These packages are "m*ini-Repos*" that can be versioned, built, and published **independently**.
3. Every package contains its own ***package.json*** file due to the fact that every package is a full-fledged project on its own.
4. Packages may have **dependency relations** between each other.
5. Managing these dependencies are implemented by **symlinks**.

Nx Workspace



1. Hosts 1..n **libraries**, **applications**, and NodeJS **server**.
2. One (1) single ***package.json*** for entire repo. Same version for all apps and builds.
3. Apps and special libs can be built, tested, and published **independently**.
4. Can include **Cypress**, **Jest**, and **Storybook** tools

Developers get

More with Nx Workspaces....

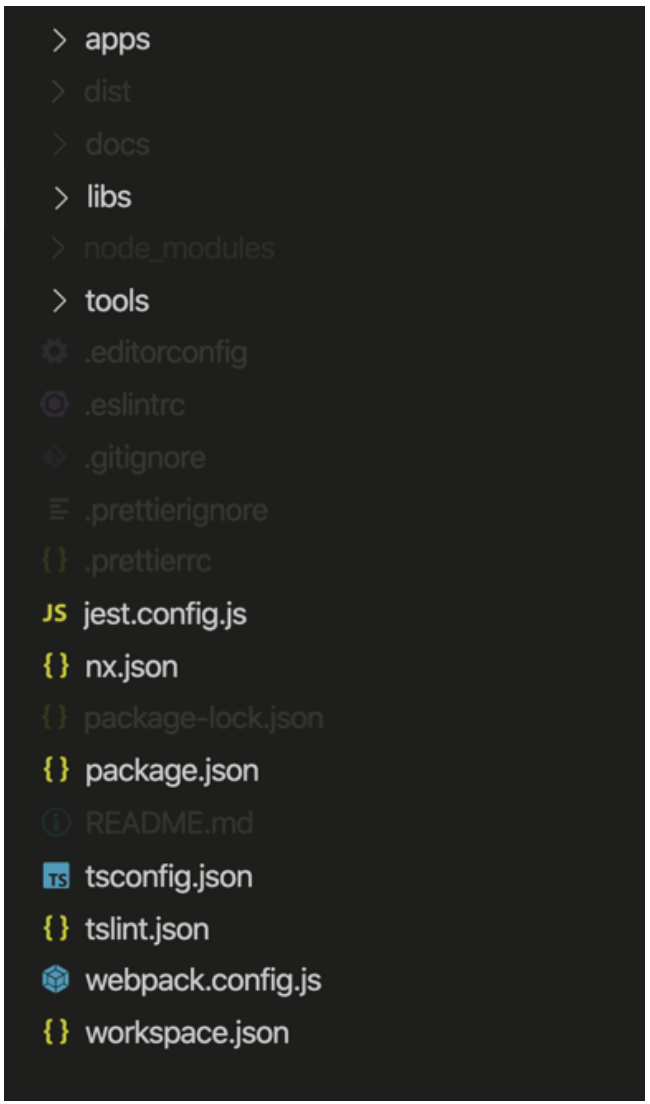
- Angular
- **React**
- **StoryBook**
- NodeJS APIs (NestJS)

- **Jest** Unit Testing
- **Cypress** e2e

- Angular CLI, Nx CLI
- GraphQL
- TypeScript or JavaScript

Creating a Workspace

```
1 npx create-nx-workspace <workspace-name>
```



How is **Nx** special ?

Workspaces

nx.json

- Defines library dependencies
- Defines npm scope alias

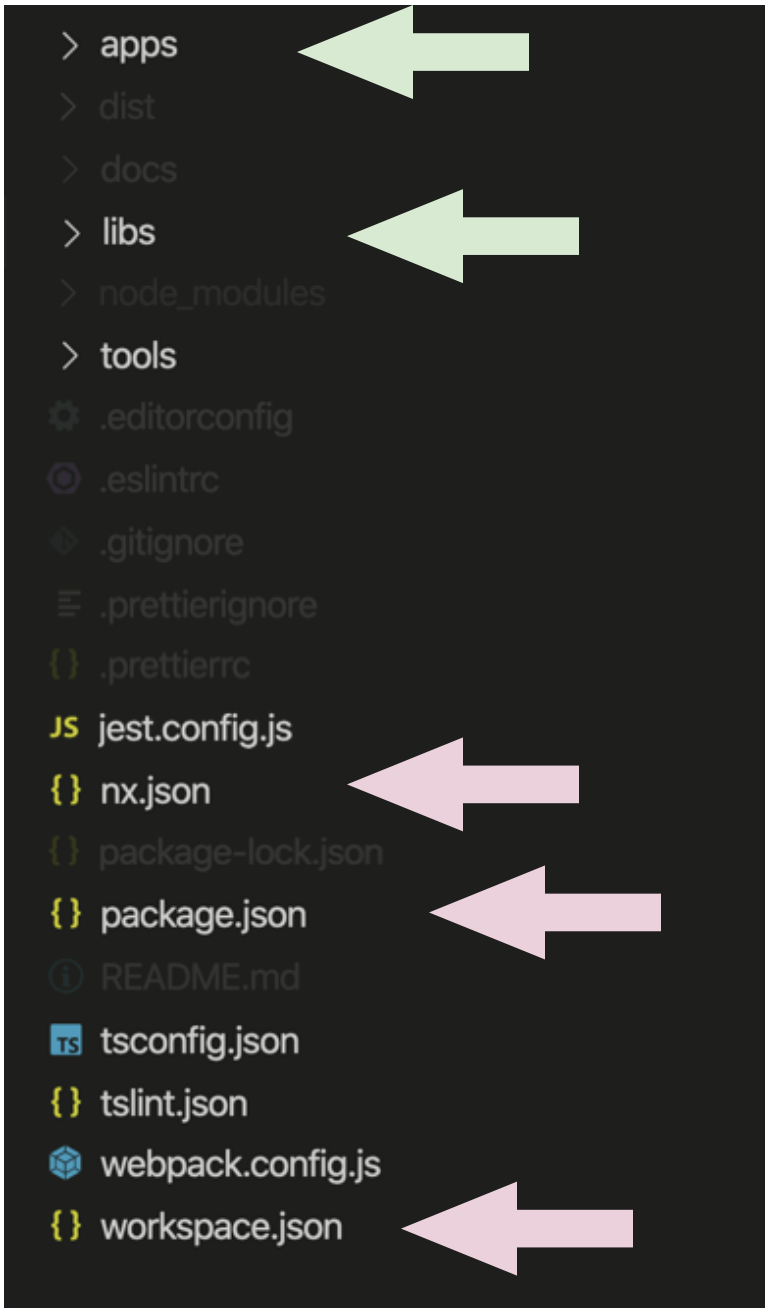
package.json

- Defines **global** package dependencies
- **Single** version for all libs and apps
- Defines version of builds

workspace.json

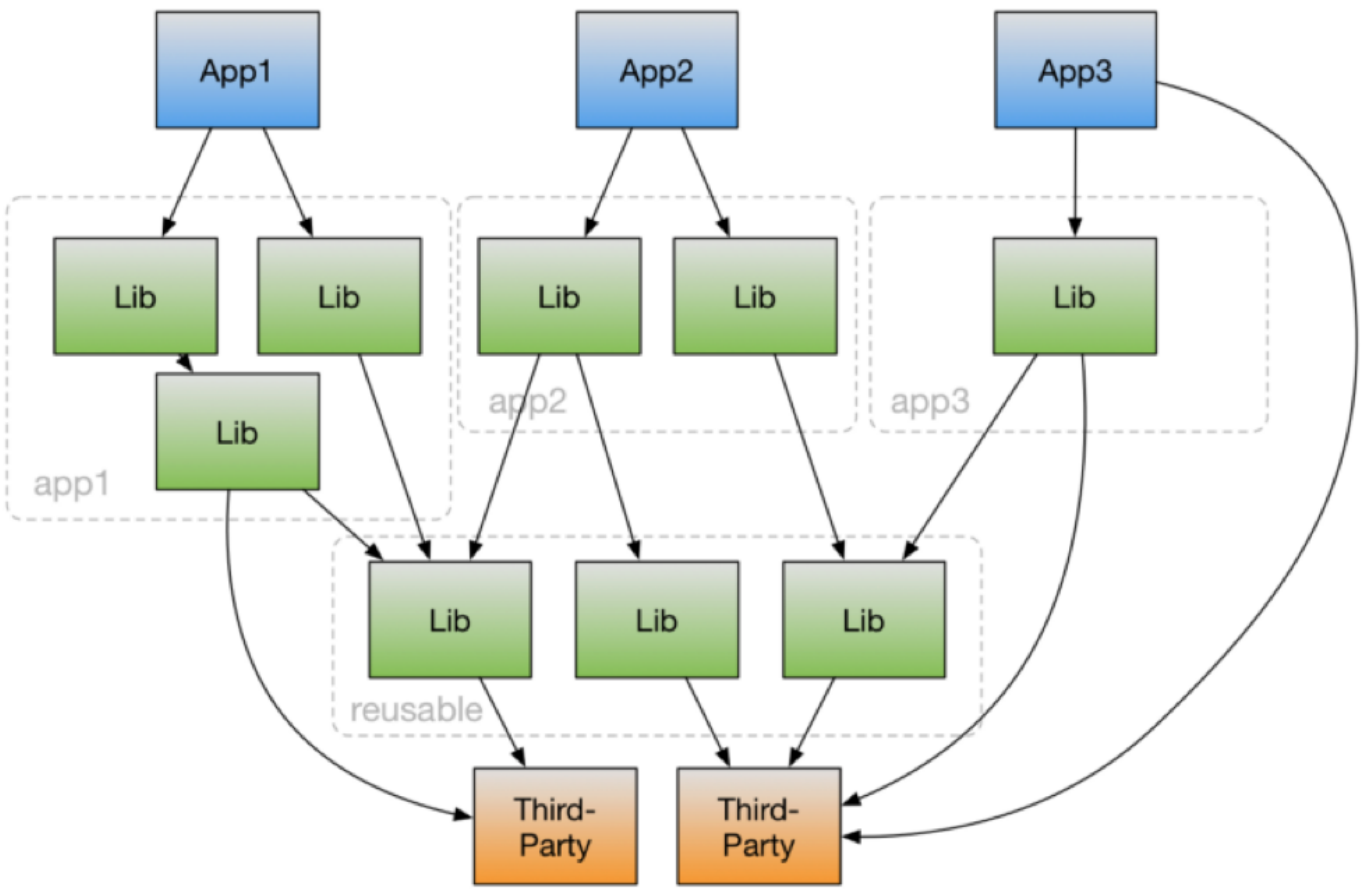
- Defines projects
- Configures tools to build, lint, and test

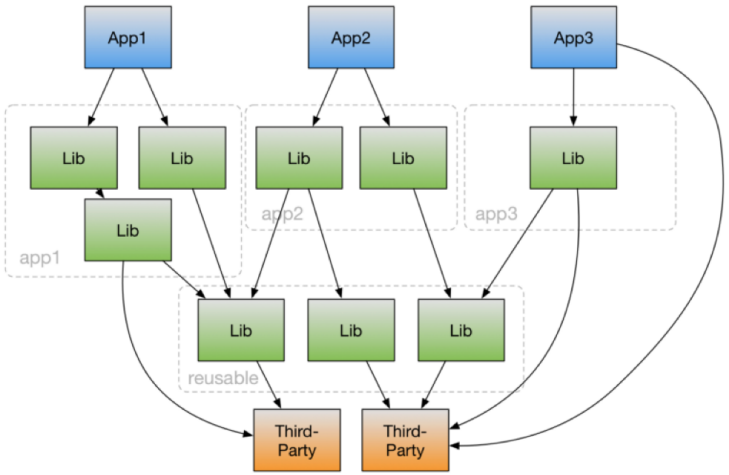
... includes **Typescript**, **Jest**, **Eslint**, **Prettier**.



Workspaces

Composable Libraries





Apps

- Are deployable
- Are responsible for arranging and bootstrapping library code
- Live in the **`apps/`** directory
- Are not importable
- Should do very little

Libs

- Consumed by apps and libs
- Have an explicit public API defined in **index.ts**
- Live in the **`libs/`** directory
- Can be used for functionality shared by many apps, or a single app

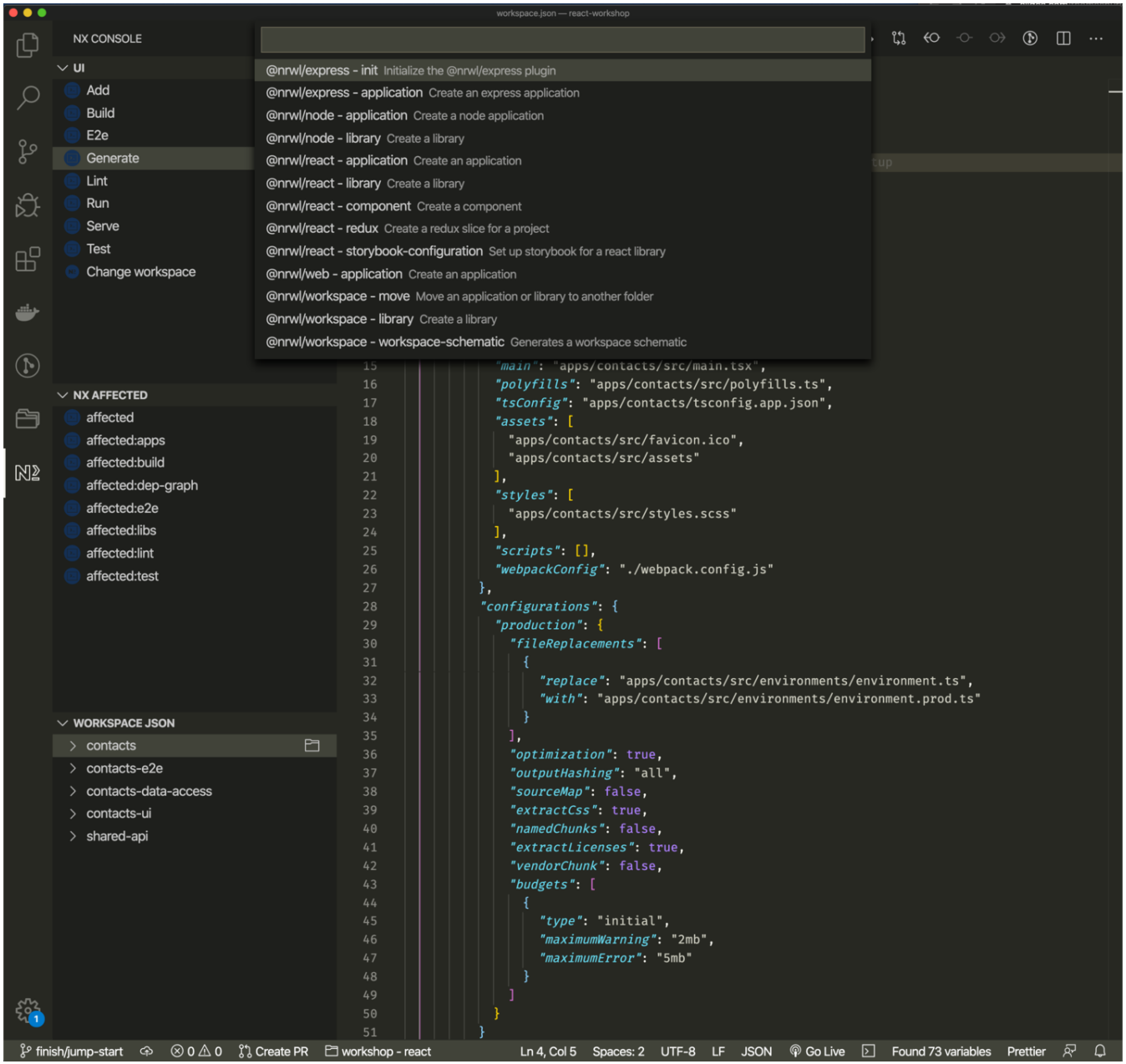
Libs can also be published directly to npm

Speaker notes

Apps are not importable!
You can create other folders, say a folder for your backend Node api app, etc
Apps and libs dirs are targeted by Nx schematics, but nothing stopping you from putting other types of apps/libs in there.


```
nextgen-poc [master]: nx help
Extensible Dev Tools for Monorepos

Commands:
  nx run
  [project][:target][:configuration]       Run a target for a project
  [options, ...]                           (e.g., nx run
                                           myapp:serve:production).

                                           You can also use the infix notation
                                           to run a target:
                                           (e.g., nx serve myapp
                                           --configuration=production)

  nx generate
  [schematic-collection:][schematic]       Generate code
  [options, ...]                           (e.g., nx generate @nrwl/web:app
                                           myapp).

  nx affected                              Run task for affected projects
  nx run-many                              Run task for multiple projects
  nx affected:apps                         Print applications affected by
                                           changes
  nx affected:libs                         Print libraries affected by changes
  nx affected:build                        Build applications and publishable
                                           libraries affected by changes
  nx affected:test                         Test projects affected by changes
  nx affected:e2e                          Run e2e tests for the applications
                                           affected by changes
  nx affected:dep-graph                    Graph dependencies affected by
                                           changes
  nx print-affected                        Graph execution plan
  nx affected:lint                         Lint projects affected by changes
  nx dep-graph                             Graph dependencies within workspace
  nx format:check                          Check for un-formatted files
  nx format:write                          Overwrite un-formatted files
  nx workspace-lint [files..]              Lint workspace or list of files
  nx workspace-schematic [name]            Runs a workspace schematic from the
                                           tools/schematics directory
  nx migrate                               Creates a migrations file or runs
                                           migrations from the migrations file.
                                           - Migrate packages and create
                                           migrations.json (e.g., nx migrate
                                           @nrwl/workspace@latest)
                                           - Run migrations (e.g., nx migrate
                                           --run-migrations=migrations.json)

  nx report                                Reports useful version numbers to
                                           copy into the Nx issue template
  nx list [plugin]                         Lists installed plugins,
                                           capabilities of installed plugins
                                           and other available plugins.

Options:
  --help     Show help                                         [boolean]
  --version  Show version number                               [boolean]
```

Nx Console


workspace.json — react-workshop
NX CONSOLE
UI
Add
Build
E2e
Generate
Lint
Run
Serve
Test
Change workspace
@nrwl/express - init Initialize the @nrwl/express plugin
@nrwl/express - application Create an express application
@nrwl/node - application Create a node application
@nrwl/node - library Create a library
@nrwl/react - application Create an application
@nrwl/react - library Create a library
@nrwl/react - component Create a component
@nrwl/react - redux Create a redux slice for a project
@nrwl/react - storybook-configuration Set up storybook for a react library
@nrwl/web - application Create an application
@nrwl/workspace - move Move an application or library to another folder
@nrwl/workspace - library Create a library
@nrwl/workspace - workspace-schematic Generates a workspace schematic
NX AFFECTED
affected
affected:apps
affected:build
affected:dep-graph
affected:e2e
affected:libs
affected:lint
affected:test
WORKSPACE JSON
contacts
contacts-e2e
contacts-data-access
contacts-ui
shared-api
"main": "apps/contacts/src/main.tsx",
"polyfills": "apps/contacts/src/polyfills.ts",
"tsConfig": "apps/contacts/tsconfig.app.json",
"assets": [
"apps/contacts/src/favicon.ico",
"apps/contacts/src/assets"
],
"styles": [
"apps/contacts/src/styles.scss"
],
"scripts": [],
"webpackConfig": "./webpack.config.js"
},
"configurations": {
"production": {
"fileReplacements": [
{
"replace": "apps/contacts/src/environments/environment.ts",
"with": "apps/contacts/src/environments/environment.prod.ts"
}
],
"optimization": true,
"outputHashing": "all",
"sourceMap": false,
"extractCss": true,
"namedChunks": false,
"extractLicenses": true,
"vendorChunk": false,
"budgets": [
{
"type": "initial",
"maximumWarning": "2mb",
"maximumError": "5mb"
}
]
}
}
finish/jump-start
Create PR
workshop - react
Ln 4, Col 5
Spaces: 2
UTF-8
LF
JSON
Go Live
Found 73 variables
Prettier

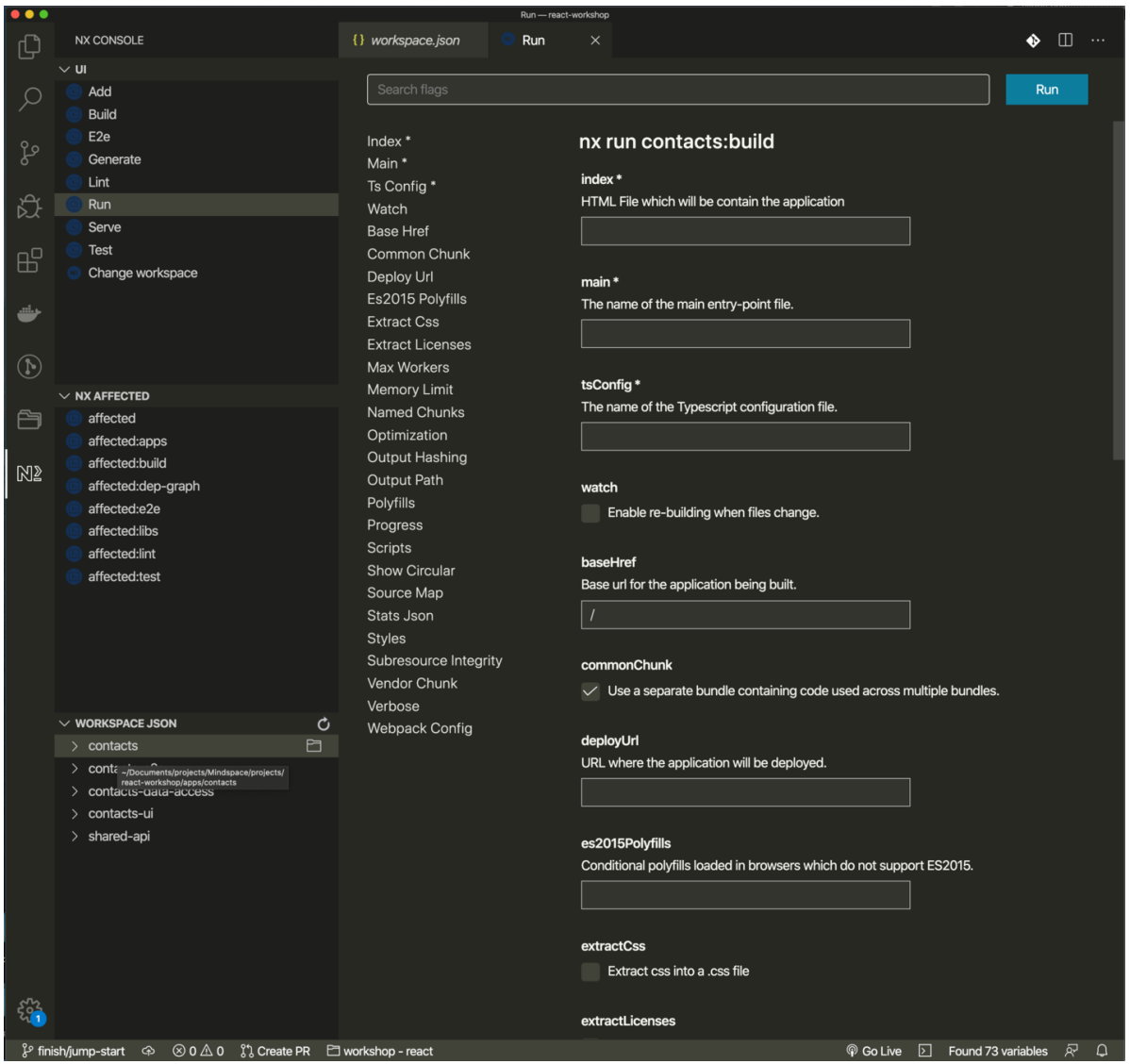
Build a React Application

Workspaces

```
nx run contacts:build --prod
nx build contacts --prod
```

```
nx run contacts:build [options,...]

Options:
  --main                  The name of the main entry-point file.
  --tsConfig              The name of the Typescript configuration file.
  --outputPath            The output path of the generated files.
  --watch                 Enable re-building when files change.
  --baseHref              Base url for the application being built. (default: /)
  --deployUrl             URL where the application will be deployed.
  --vendorChunk           Use a separate bundle containing only vendor libraries. (default: true)
  --commonChunk           Use a separate bundle containing code used across multiple bundles. (default: true)
  --sourceMap             Output sourcemaps. (default: true)
  --progress              Log progress to the console while building.
  --assets                List of static application assets. (default: )
  --index                 HTML File which will be contain the application
  --scripts               External Scripts which will be included before the main application entry
  --styles                External Styles which will be included with the application
  --budgets               Budget thresholds to ensure parts of your application stay within boundaries which you set. (de
fault: )
  --namedChunks           Names the produced bundles according to their entry file (default: true)
  --outputHashing         Define the output filename cache-busting hashing mode. (default: none)
  --stylePreprocessorOptio Options to pass to style preprocessors.
  --optimization          Enables optimization of the build output.
  --extractCss            Extract css into a .css file
  --es2015Polyfills       Conditional polyfills loaded in browsers which do not support ES2015.
  --subresourceIntegrity  Enables the use of subresource integrity validation.
  --polyfills             Polyfills to load before application
  --verbose               Emits verbose output
  --statsJson             Generates a 'stats.json' file which can be analyzed using tools such as: #webpack-bundle-analyz
er' or https://webpack.github.io/analyse.
  --extractLicenses       Extract all licenses in a separate file, in the case of production builds only.
  --showCircularDependenci Show circular dependency warnings on builds. (default: true)
  --memoryLimit           Memory limit for type checking service process in MB. (defaults to 2048)
  --maxWorkers            Number of workers to use for type checking. (defaults to # of CPUS - 2)
  --fileReplacements      Replace files with other files in the build. (default: )
  --webpackConfig         Path to a function which takes a webpack config, some context and returns the resulting webpack
```

Build a React Application



Differential Builds for React

Workspaces

```
-rw-r--r--+   1 thomasburleson  staff    43146 Feb  7 15:04 index-69c37885-js.js
-rw-r--r--+   1 thomasburleson  staff    50219 Feb  7 15:04 index-69c37885-js.js.map
-rw-r--r--+   1 thomasburleson  staff      812 Feb  8 06:50 index.html
-rw-r--r--+   1 thomasburleson  staff    14031 Feb  7 15:04 input-shims-a4fc53ac-js.js
-rw-r--r--+   1 thomasburleson  staff    15221 Feb  7 15:04 input-shims-a4fc53ac-js.js.map
-rw-r--r--+   1 thomasburleson  staff    27470 Feb  7 15:04 ios-transition-071bd421-js.js
-rw-r--r--+   1 thomasburleson  staff    27941 Feb  7 15:04 ios-transition-071bd421-js.js.map
-rw-r--r--+   1 thomasburleson  staff    27470 Jan 28 13:47 ios-transition-504cdd09-js.js
-rw-r--r--+   1 thomasburleson  staff    27941 Jan 28 13:47 ios-transition-504cdd09-js.js.map
-rw-r--r--+   1 thomasburleson  staff   248076 Feb  8 06:50 main.376593810d98c1126ffd.es5.js
-rw-r--r--+   1 thomasburleson  staff   248076 Feb  8 06:50 main.922518368a8476a07c0b.esm.js
-rw-r--r--+   1 thomasburleson  staff   108733 Feb  7 15:04 main.js
-rw-r--r--+   1 thomasburleson  staff   116229 Feb  7 15:04 main.js.map
-rw-r--r--+   1 thomasburleson  staff     4270 Feb  7 15:04 md-transition-15a81b08-js.js
-rw-r--r--+   1 thomasburleson  staff     2884 Feb  7 15:04 md-transition-15a81b08-js.js.map
-rw-r--r--+   1 thomasburleson  staff     4270 Jan 28 13:47 md-transition-fea2bbfb-js.js
-rw-r--r--+   1 thomasburleson  staff     2884 Jan 28 13:47 md-transition-fea2bbfb-js.js.map
-rw-r--r--+   1 thomasburleson  staff    97284 Feb  8 06:50 polyfills.92e03bd7972c1a154060.es5.js
-rw-r--r--+   1 thomasburleson  staff    81568 Feb  8 06:50 polyfills.adba5a71ea1890eb3207.esm.js
-rw-r--r--+   1 thomasburleson  staff   437545 Feb  7 15:04 polyfills.js
-rw-r--r--+   1 thomasburleson  staff   329909 Feb  7 15:04 polyfills.js.map
-rw-r--r--+   1 thomasburleson  staff     4812 Feb  8 06:50 runtime.10867e855fcba5ca9254.js
-rw-r--r--+   1 thomasburleson  staff     4812 Feb  8 06:50 runtime.50df1d11b074e0954570.js
-rw-r--r--+   1 thomasburleson  staff    10151 Feb  7 15:04 runtime.js
-rw-r--r--+   1 thomasburleson  staff    10354 Feb  7 15:04 runtime.js.map
```

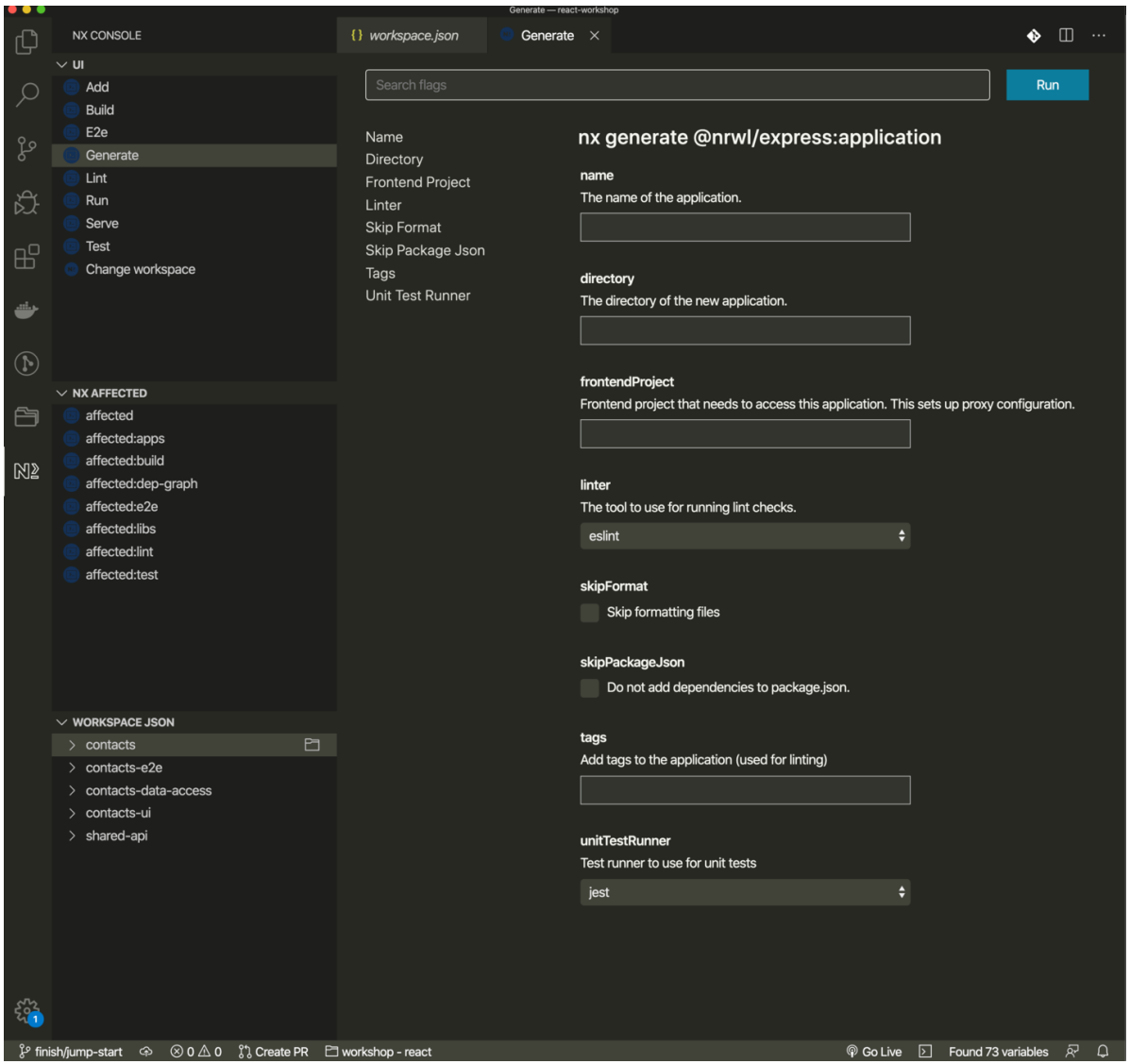
Creating a React Application

Workspaces

```
nx g @nrwl/react:app <app-name>
```

```
react-workshop [master]: nx g @nrwl/react:app --help
nx generate @nrwl/react:application [name] [options,...]

Options:
  --name                  The name of the application.
  --directory             The directory of the new application.
  --style                 The file extension to be used for style files. (default: css)
  --linter                The tool to use for running lint checks. (default: tslint)
  --routing               Generate application with routes
  --skipFormat            Skip formatting files
  --skipWorkspaceJson     Skip updating workspace.json with default schematic options based
on values provided to this app (e.g. babel, style)
  --unitTestRunner        Test runner to use for unit tests (default: jest)
  --e2eTestRunner         Test runner to use for end to end (e2e) tests (default: cypress)
  --tags                  Add tags to the application (used for linting)
  --pascalCaseFiles       Use pascal case component file name (e.g. App.tsx)
  --classComponent        Use class components instead of functional component
  --js                    Generate JavaScript files rather than TypeScript files
  --dryRun                undefined
  --help                  Show available options for project target.
```

Creating a React Application



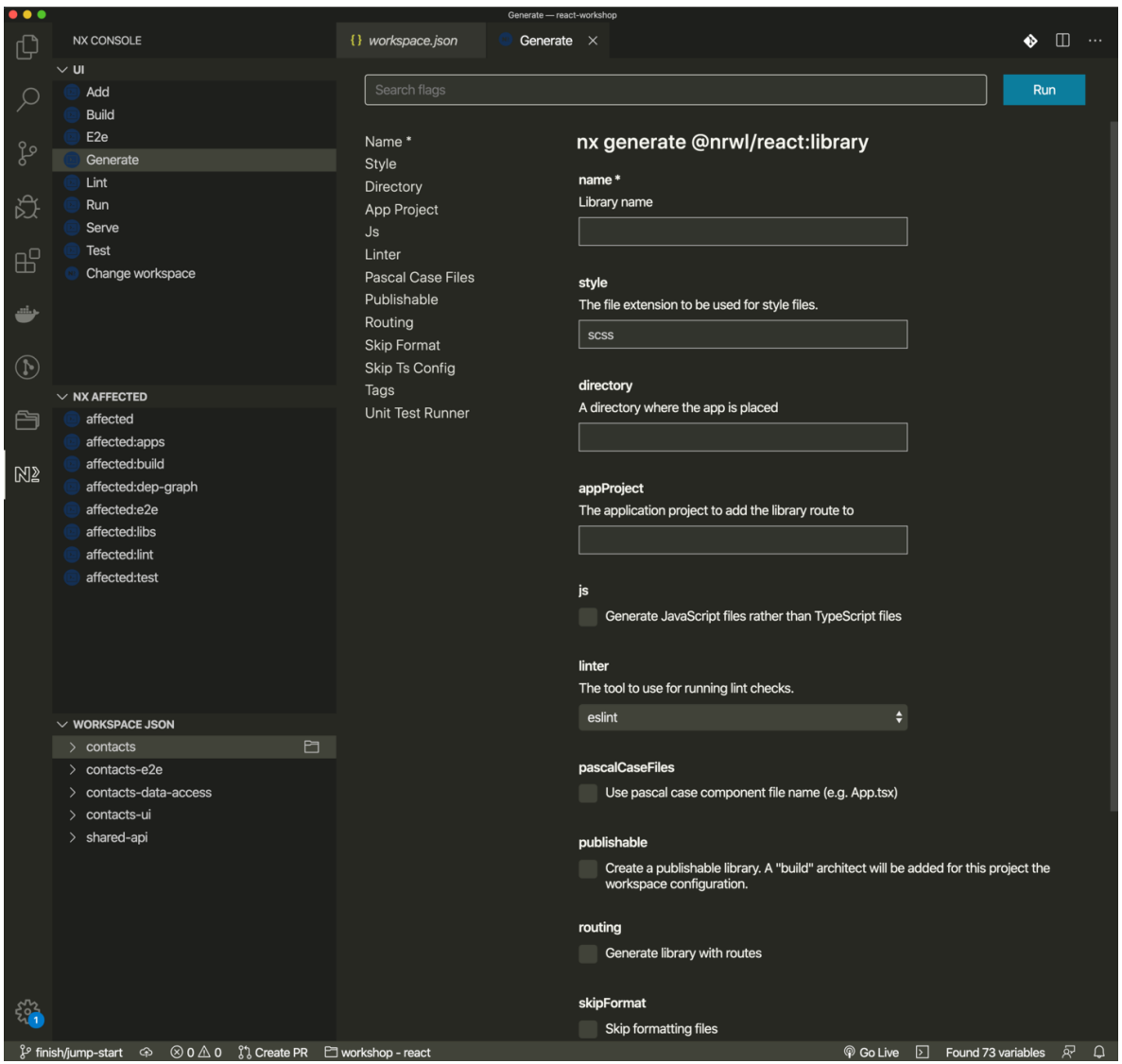
Creating a React Library

Workspaces

```
nx g @nrwl/react:lib --directory=contacts --dryRun  data-access
```

```
react-workshop [(HEAD detached at f45f4b7)]: nx g @nrwl/react:lib --directory=contacts --dryRun  data-access
CREATE libs/contacts/data-access/.eslintrc (6848 bytes)
CREATE libs/contacts/data-access/README.md (194 bytes)
CREATE libs/contacts/data-access/tsconfig.json (414 bytes)
CREATE libs/contacts/data-access/tsconfig.lib.json (227 bytes)
CREATE libs/contacts/data-access/src/index.ts (44 bytes)
CREATE libs/contacts/data-access/tsconfig.spec.json (252 bytes)
CREATE libs/contacts/data-access/jest.config.js (270 bytes)
CREATE libs/contacts/data-access/src/lib/contacts-data-access.scss (0 bytes)
CREATE libs/contacts/data-access/src/lib/contacts-data-access.spec.tsx (326 bytes)
CREATE libs/contacts/data-access/src/lib/contacts-data-access.tsx (349 bytes)
UPDATE tsconfig.json (864 bytes)
UPDATE workspace.json (5301 bytes)
UPDATE nx.json (505 bytes)

NOTE: The "dryRun" flag means no changes were made.
```

Creating a React Library



Define Public API

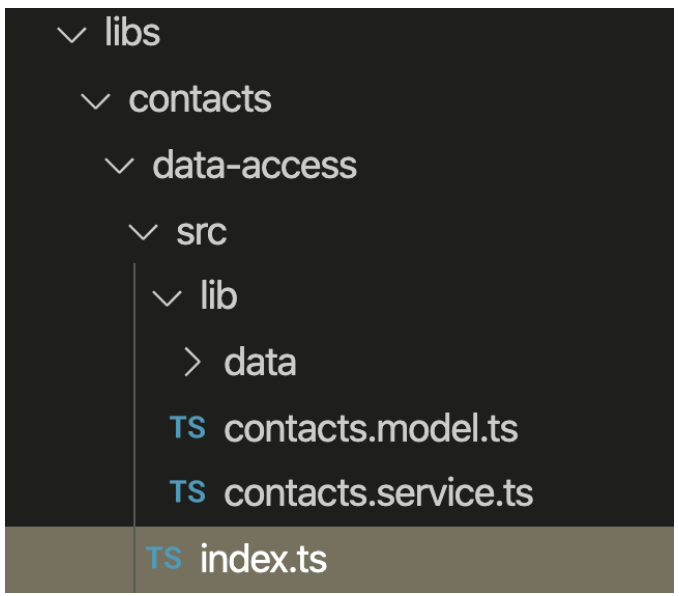
Developers need to be **explicit** with Public APIs

- Write code in **libs**
- Control what you **expose**

Nx helps you do just that!

Define Public API

Use **index.ts** to define what you expose as **public**




```
export * from './lib/contacts.model';
export * from './lib/contacts.service';
```

Using the Library Public API

Import from a library public API:

```
import { ContactsService } from '@workshop/contacts/data-access';
```

See **tsconfig.ts** that defines the import path:

```
{
  "compilerOptions": {
    "paths": {
      "@workshop/contacts/data-access": [
        "libs/contacts/data-access/src/index.ts"
      ]
    }
  },
}
```

Bypassing the Public API

Nx prevents developers doing this:

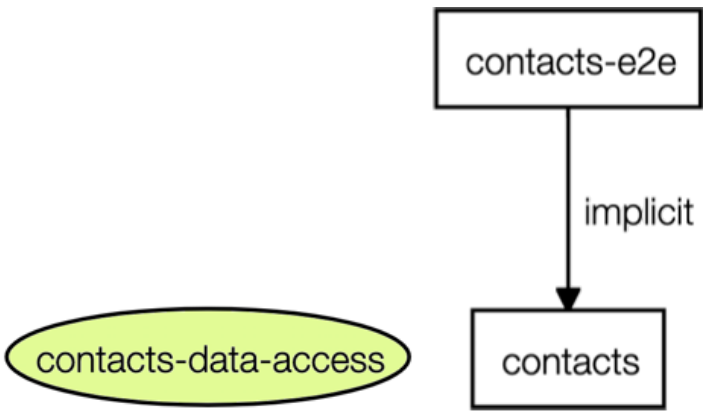
```
import { ContactsService } from '@workshop/contacts/data-access/src/lib/contacts.service';
```

Bypassing the API...

... with Nx, they should see this **lint error**:

```
deep imports into libraries are forbidden (nx-enforce-module-boundaries) tslint(1)
Peek Problem    Quick Fix...
import { ContactsService } from '@workshop/contacts/data-access/src/lib/contacts.service';
```

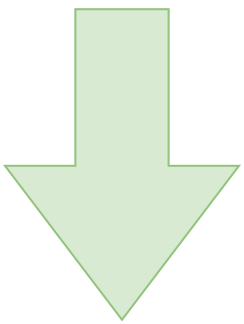
Lab 2a: Data-Access Libraries



- Generate a "data-access" library
- Move the REST service: *ContactsService*
- Configure the public API
- Test the import

Shared API

What if multiple FE apps + Node APIs want to share the **Contact** interface?



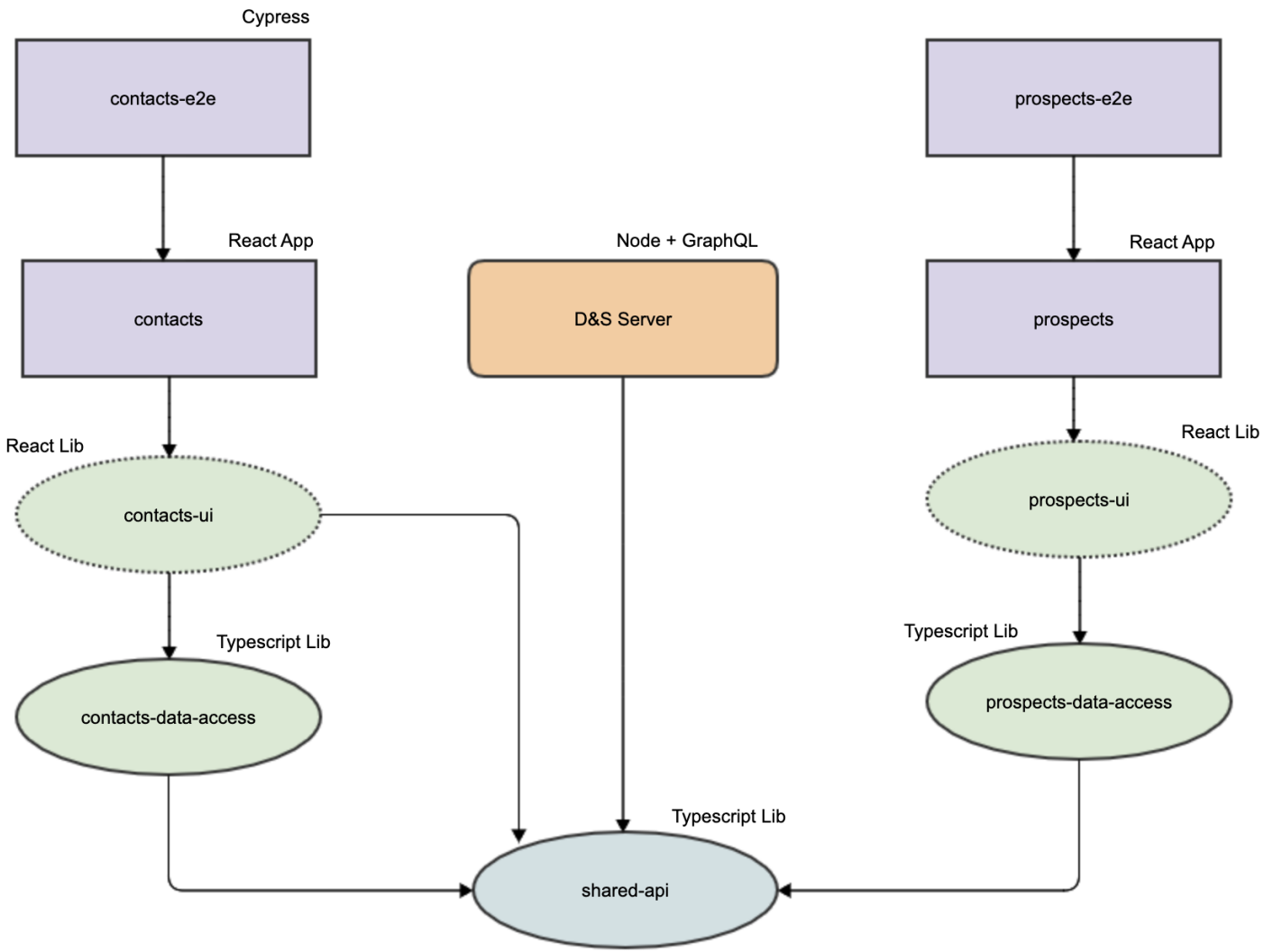
Create a shared TS library that exports `***contacts.model.ts***`

Speaker notes

How many times have your front-end and back-end fallen out of sync? Either the front-end didn't live up to it's responsibility or the back-end failed to deliver on the contract you agreed to. That would never happen at Ameriprise, of course, but imagine a place where backend teams and frontend teams are completely siloed ;)

And as a result, you may not know you are busted locally out you're busted in Dev or QA? Or PROD?

Dependency Graph



Speaker notes

When we colocate our apps and APIs in a monorepo, we get the benefit of testing them together AND promoting them together as a part of the same atomic commit. With Typescript, we get the added sanity of ensuring our front-end and back-end contracts remain in sync by sharing the EXACT same types. The shape of our payloads and responses are defined in a single, shared type.

Now we aren't focusing on backend APIs today but shared types are extremely valuable even if all you have is

Creating a Shared Library

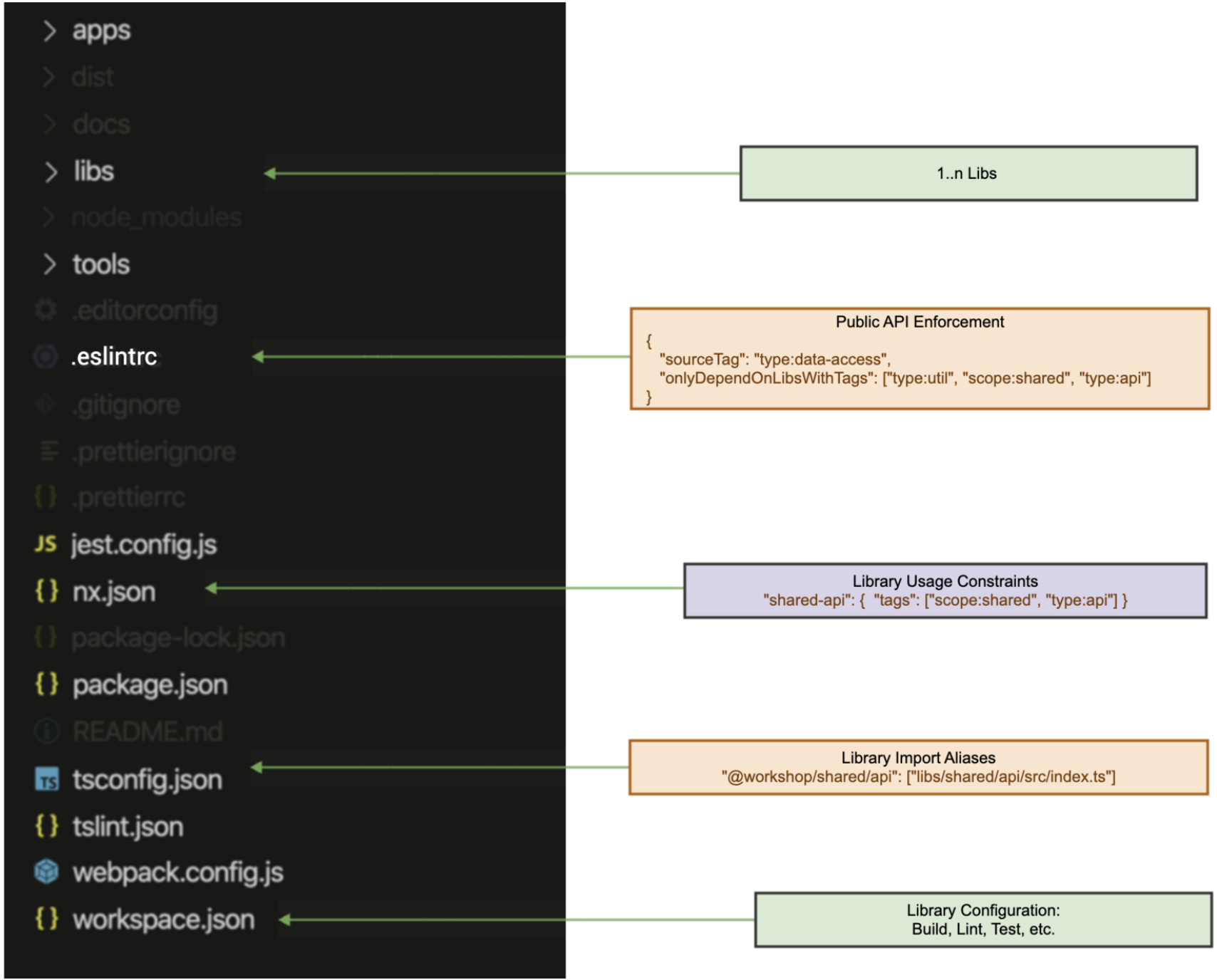
Workspaces

```
nx g @nrwl/web:lib api \
     --directory=shared \
     --tags="scope:shared,type:api" \
     --dryRun
```

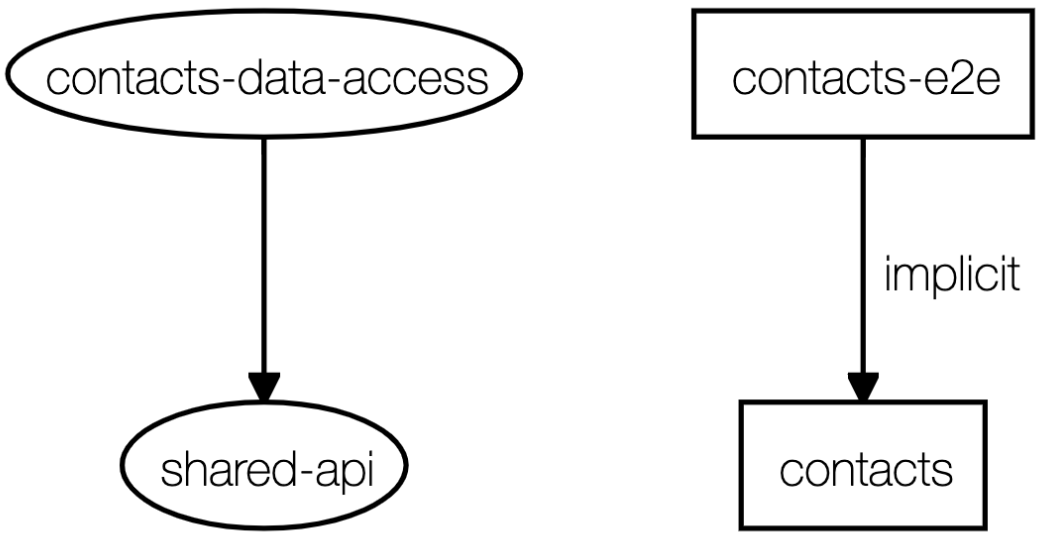
```
CREATE libs/shared/api/tslint.json (51 bytes)
CREATE libs/shared/api/README.md (176 bytes)
CREATE libs/shared/api/tsconfig.json (126 bytes)
CREATE libs/shared/api/tsconfig.lib.json (175 bytes)
CREATE libs/shared/api/src/index.ts (35 bytes)
CREATE libs/shared/api/src/lib/shared-api2.ts (0 bytes)
CREATE libs/shared/api/tsconfig.spec.json (252 bytes)
CREATE libs/shared/api/jest.config.js (252 bytes)
UPDATE tsconfig.json (994 bytes)
UPDATE workspace.json (6723 bytes)
UPDATE nx.json (698 bytes)

NOTE: The "dryRun" flag means no changes were made.
```

Speaker notes

Here's a command to generate our library along with a dry run of the output we can expect.


apps
dist
docs
libs
node_modules
tools
.editorconfig
.eslintrc
.gitignore
.prettierignore
.prettierrc
jest.config.js
nx.json
package-lock.json
package.json
README.md
tsconfig.json
tslint.json
webpack.config.js
workspace.json
1..n Libs
Public API Enforcement
{
"sourceTag": "type:data-access",
"onlyDependOnLibsWithTags": ["type:util", "scope:shared", "type:api"]
}
Library Usage Constraints
"shared-api": { "tags": ["scope:shared", "type:api"] }
Library Import Aliases
"@workshop/shared/api": ["libs/shared/api/src/index.ts"]
Library Configuration:
Build, Lint, Test, etc.

Lab 2b: Shared API Library



- Generate the "api" TS library
- Move `contacts.model.ts`
- Configure the public API
- Test the import and Constraints

Creating a UI Library

Workspaces

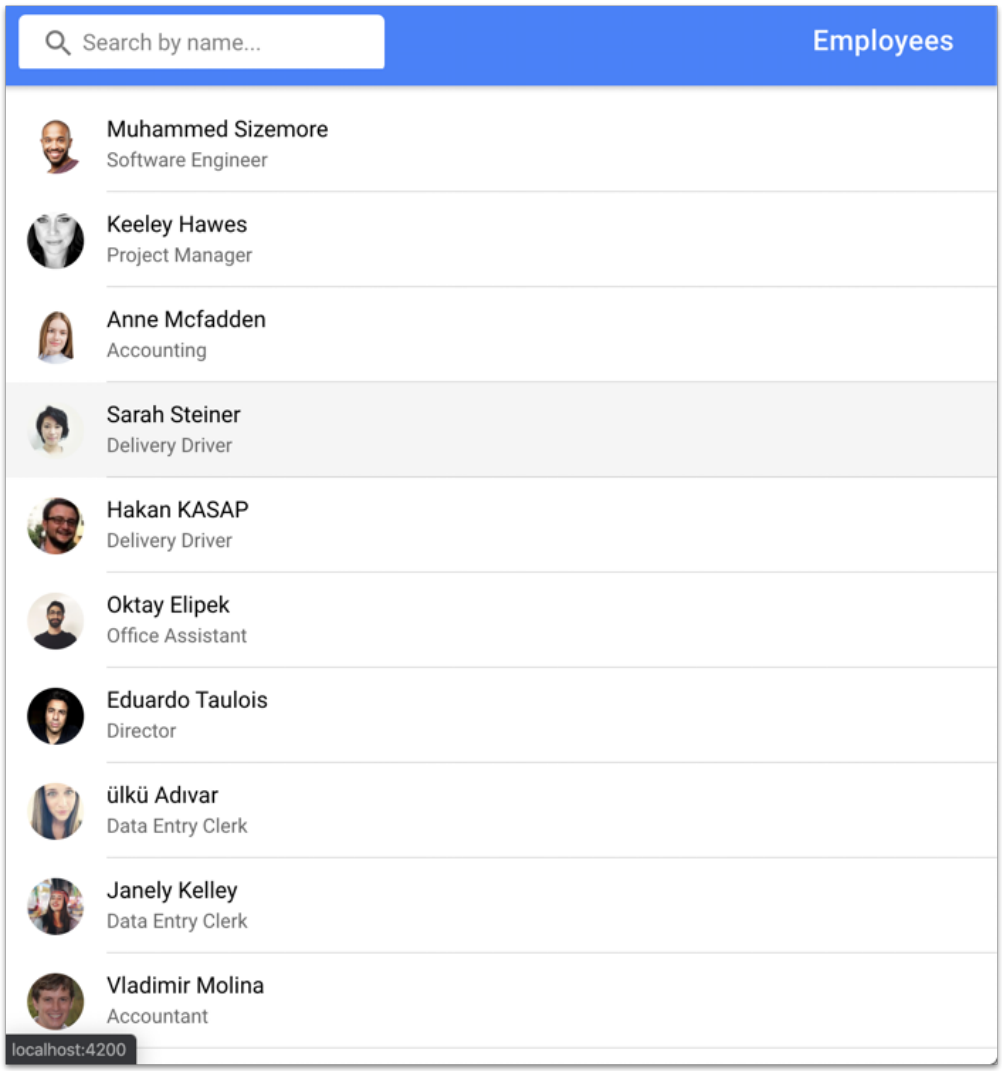
```
nx g @nrwl/react:lib ui \
     --directory=contacts \
     --tags="scope:contacts,type:feature" \
     --dryRun
```

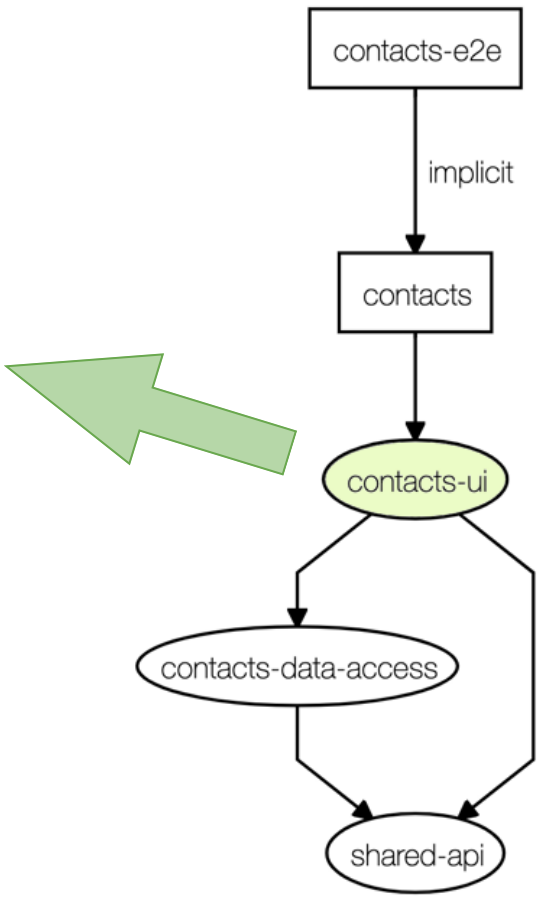
```
react-workshop [(HEAD detached at 1870038)]: nx g @nrwl/react:lib ui --directory=contacts --tags="scope:contac
ts,type:feature" --dryRun
CREATE libs/contacts/ui/.eslintrc (6848 bytes)
CREATE libs/contacts/ui/README.md (176 bytes)
CREATE libs/contacts/ui/tsconfig.json (414 bytes)
CREATE libs/contacts/ui/tsconfig.lib.json (227 bytes)
CREATE libs/contacts/ui/src/index.ts (35 bytes)
CREATE libs/contacts/ui/tsconfig.spec.json (252 bytes)
CREATE libs/contacts/ui/jest.config.js (252 bytes)
CREATE libs/contacts/ui/src/lib/contacts-ui.scss (0 bytes)
CREATE libs/contacts/ui/src/lib/contacts-ui.spec.tsx (293 bytes)
CREATE libs/contacts/ui/src/lib/contacts-ui.tsx (299 bytes)
UPDATE tsconfig.json (994 bytes)
UPDATE workspace.json (6788 bytes)
UPDATE nx.json (704 bytes)

NOTE: The "dryRun" flag means no changes were made.
```

Speaker notes

Notice that an component has also been generated.
We will rename and update the component soon...

Creating a UI Library: **contacts-ui**

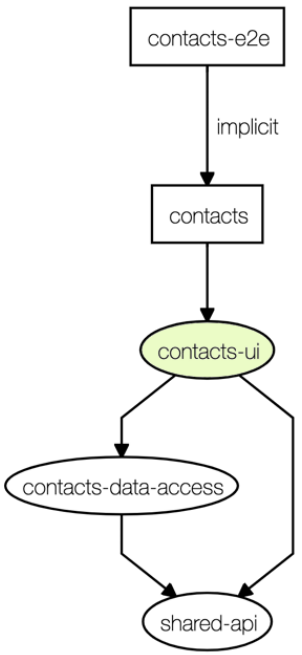




libs/contacts/ui/src/lib/**contacts-list.tsx**

apps/contacts/src/app/**app.tsx**

```
1 import { ContactsList } from '@workshop/contacts/ui';
```



libs/contacts/ui/src/lib/**contacts-list.tsx**

```
1 import { Contact } from '@workshop/shared/api';
2 import { ContactsService } from '@workshop/contacts/data-access';
```

UI Component LifeCycles

life cycle effects

triggers code when...

component mounts

props or state updates

component unmounts

Class-based UI Component

libs/contacts/ui/src/lib/**contacts-list.tsx**

```
export class ContactsList extends Component {
  private service = new ContactsService();

  constructor(props) {
    super(props);
    this.state = { people: [] };
  }

  componentDidMount() {
    this.service.getContacts().then(list => {
      this.setState({ people: list });
    });
  }

  render() {
    const doSearch = criteria => {
      this.service.searchBy(criteria).then(list => {
        this.setState({ people: list });
      });
    };

    return (
      ...
    );
  }
}
```

... types are missing!

UI Component with **Types**

libs/contacts/ui/src/lib/**contacts-list.tsx**

```
1  interface ContactsState { people: Contact[] };
2
3  export class ContactsList extends Component<{}, ContactsState> {
4
5    constructor(props) {
6      super(props);
7      this.state = { people: [] };
8    }
9
10   componentDidMount() {
11     this.service.getContacts().then(list => {
12       this.setState({ people: list });
13     });
14   }
15
16   render() {
17     const doSearch = criteria => {
18       this.service.searchBy(criteria).then(list => {
19         this.setState({ people: list });
20       });
21     };
22
23 }
```

UI Component **JSX**

```
render() {
  const doSearch = criteria => {
    this.service.searchBy(criteria).then(list => {
      this.setState({ people: list });
    });
  };

  return (
    <IonPage>
      <IonHeader>
        <IonToolbar>
          <IonItem style={inlineItem}>
            <IonIcon icon={search}></IonIcon>
            <IonInput
              clearInput
              autofocus
              style={iconOnLeft}
              onIonChange={e => doSearch((e.target as HTMLIonInputElement).value)}
              placeholder="Search by name..."
            ></IonInput>
          </IonItem>
          <IonTitle style={stickyRight}> Employees </IonTitle>
        </IonToolbar>
      </IonHeader>
      <IonContent>
        <IonList>
          {this.state.people.map((person, idx) => {
            return <ContactListItem key={idx} person={person} />;
          })}
        </IonList>
      </IonContent>
    </IonPage>
  );
```

Update the Public API

libs/contacts/ui/src/**index.ts**

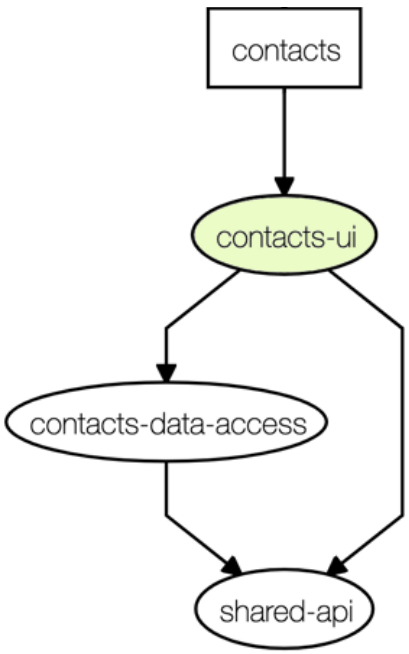
```
1 export * from './lib/contacts-list';
2 export * from './lib/contact-item';
```

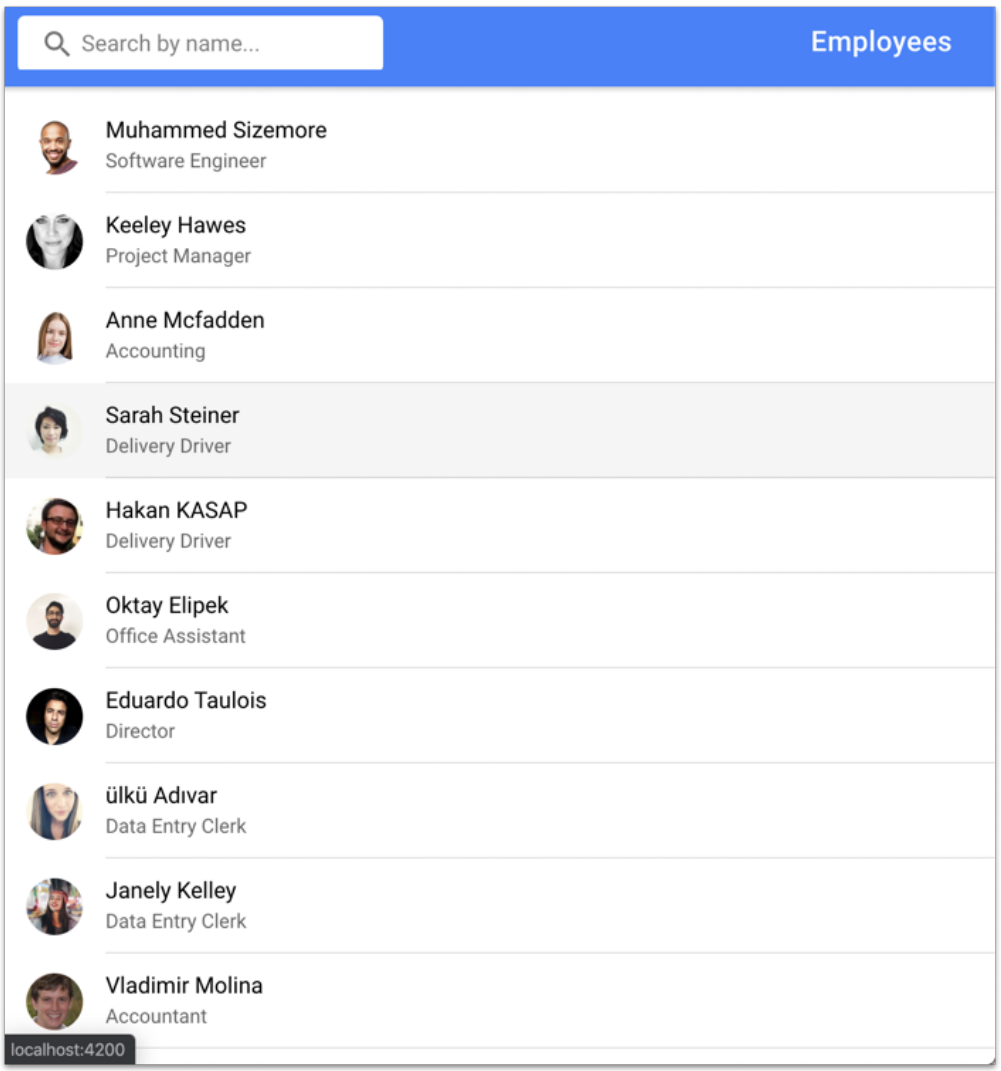
Update the Lint Rules

```
rules": {
    "@nrwl/nx/enforce-module-boundaries": [ "error",
      {
        "allow": [],
        "enforceBuildableLibDependency": true
        "depConstraints": [
          { "sourceTag": "scope:contacts", "onlyDependOnLibsWithTags": ["scope:contacts"] },
          { "sourceTag": "type:e2e",       "onlyDependOnLibsWithTags": ["type:app"] },
          { "sourceTag": "type:app",       "onlyDependOnLibsWithTags": ["type:feature", "type:util"] },
          { "sourceTag": "type:feature",   "onlyDependOnLibsWithTags": [
                                                                        "type:data-access",
                                                                        "type:ui",
                                                                        "type:util",
                                                                        "scope:shared"
                                                                        ]

          },
          { "sourceTag": "type:data-access", "onlyDependOnLibsWithTags": [
                                                                        "type:util",
                                                                        "scope:shared",
                                                                        "type:api"
                                                                ]
          }
        ],
      }
    ]
  },
```

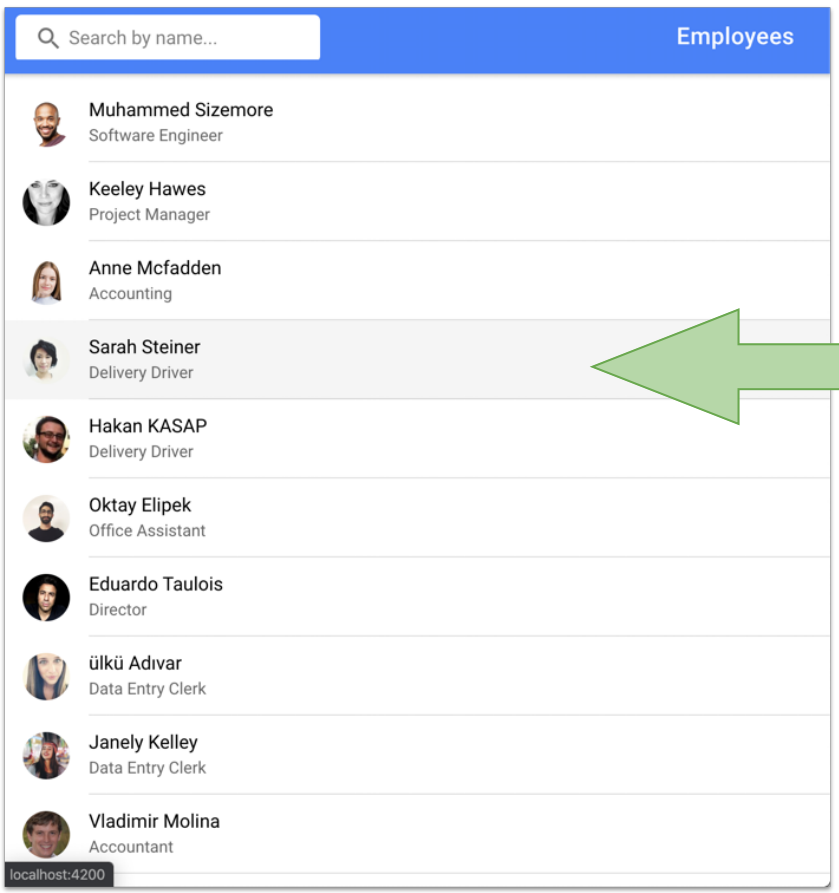
.eslintrc

Lab 3a: UI Libraries

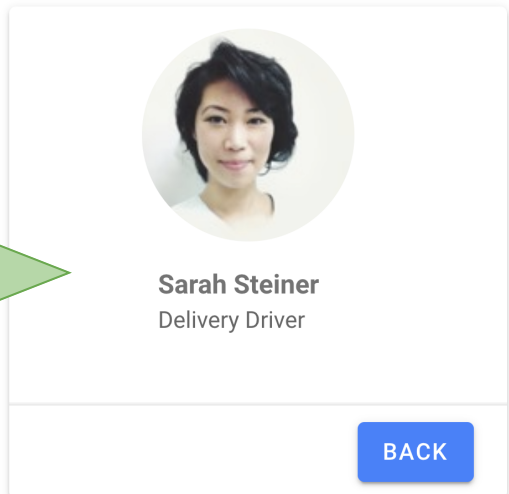




Master-Detail with **Routing**

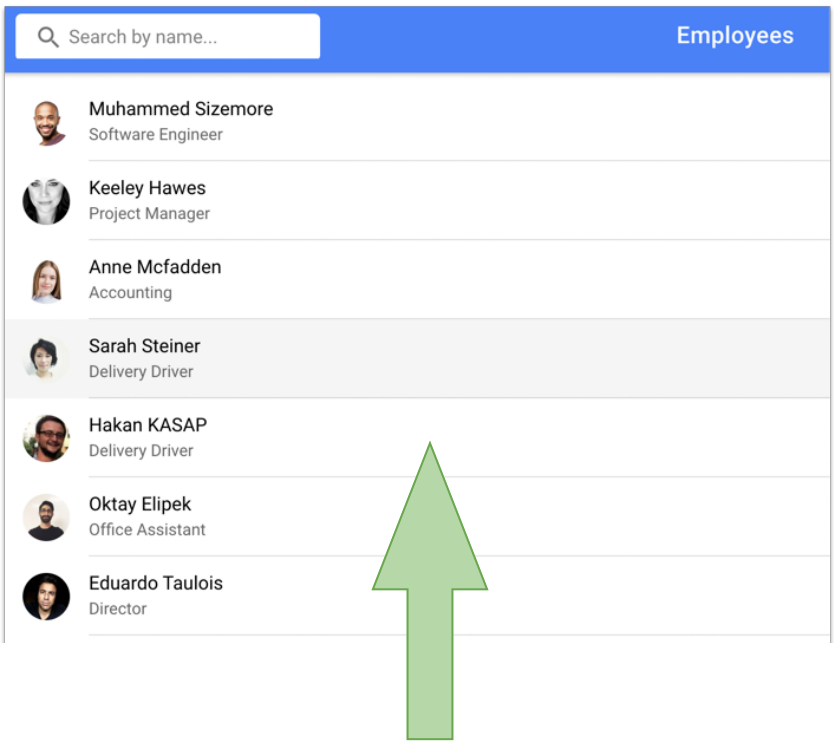


libs/contacts/ui/src/lib/**contacts-list**.**tsx**



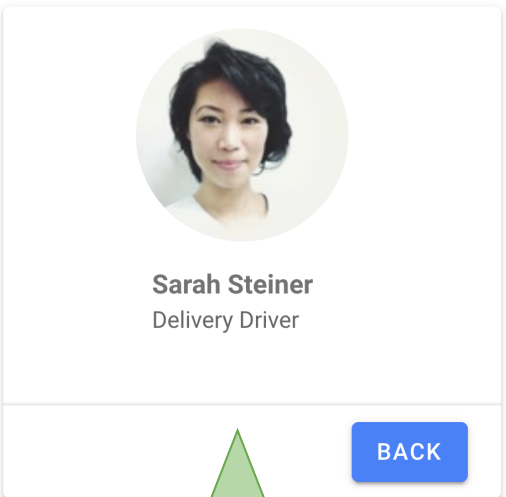
libs/contacts/ui/src/lib/**contact-detail**.**tsx**

Using the ContactsService




```
1 const service = new ContactsService();
```

Uses Router `params.id`




```
1 const service = new ContactsService();
```

Master-Detail with **Routing**

```
  <Router>
    <Switch>

      <Route path="/contacts"     exact component={ContactsList}   />
      <Route path="/contacts/:id" exact component={ContactDetails} />

      <Redirect exact from="/" to="/contacts" />

    </Switch>
  </Router>
```

```
<IonApp>
  <IonReactRouter>
    <IonRouterOutlet>

      <Route path="/contacts"     exact component={ContactsList}   />
      <Route path="/contacts/:id" exact component={ContactDetails} />

      <Redirect exact from="/" to="/contacts" />

    </IonRouterOutlet>
  </IonReactRouter>
</IonApp>
```

Speaker notes

2 routing approaches: React Router or Ionic React Router.
The IRR is a simply wrapper around the RR to support page transitions and animations (for mobile)

Class-based UI Component

libs/contacts/ui/src/lib/**contact-detail.tsx**

```
1
2  export class ContactDetails extends Component {
3
4    constructor(props) {
5      super(props);
6      this.state = { contact: {} as Contact };
7    }
8
9    componentDidMount() {
10     this.loadContact();
11   }
12
13   componentDidUpdate() {
14     const newId = this.props.match.params.id;
15     const oldId = this.state.contact ? this.state.contact.id : '';
16
17     if (newId !== oldId) {
18       this.loadContact(newId);
19     }
20   }
21
22   private loadContact(id = '') {
23     const service = new ContactsService();
24     const params  = this.props.match.params;
25     const request = service.getContactById(id || params.id);
26
27     request.then(contact => this.setState({ contact }));
28   }
29
30   render() {
31     return ( ... );
32   }
33 }
```

... types are missing!

Router `match.params`

```
<IonApp>
  <IonReactRouter>
    <IonRouterOutlet>

      <Route path="/contacts"     exact={true} component={ContactsList}   />
      <Route path="/contacts/:id" exact={true} component={ContactDetails} />

      <Redirect exact from="/" to="/contacts" />

    </IonRouterOutlet>
  </IonReactRouter>
</IonApp>
```

Navigate with **routerLink**

```
1
2  interface ListItemProps { person: Contact; }
3
4  export const ContactListItem: React.FC<ListItemProps> = ({ person }) => {
5    const url = `/contacts/${person.id}`;
6    return (
7      <IonItem routerLink={url}>...</IonItem>
8    );
9  }
```

When we '**link**' to a url, our component props gets *extra* information:

- **match**,
- location,
- history

Speaker notes

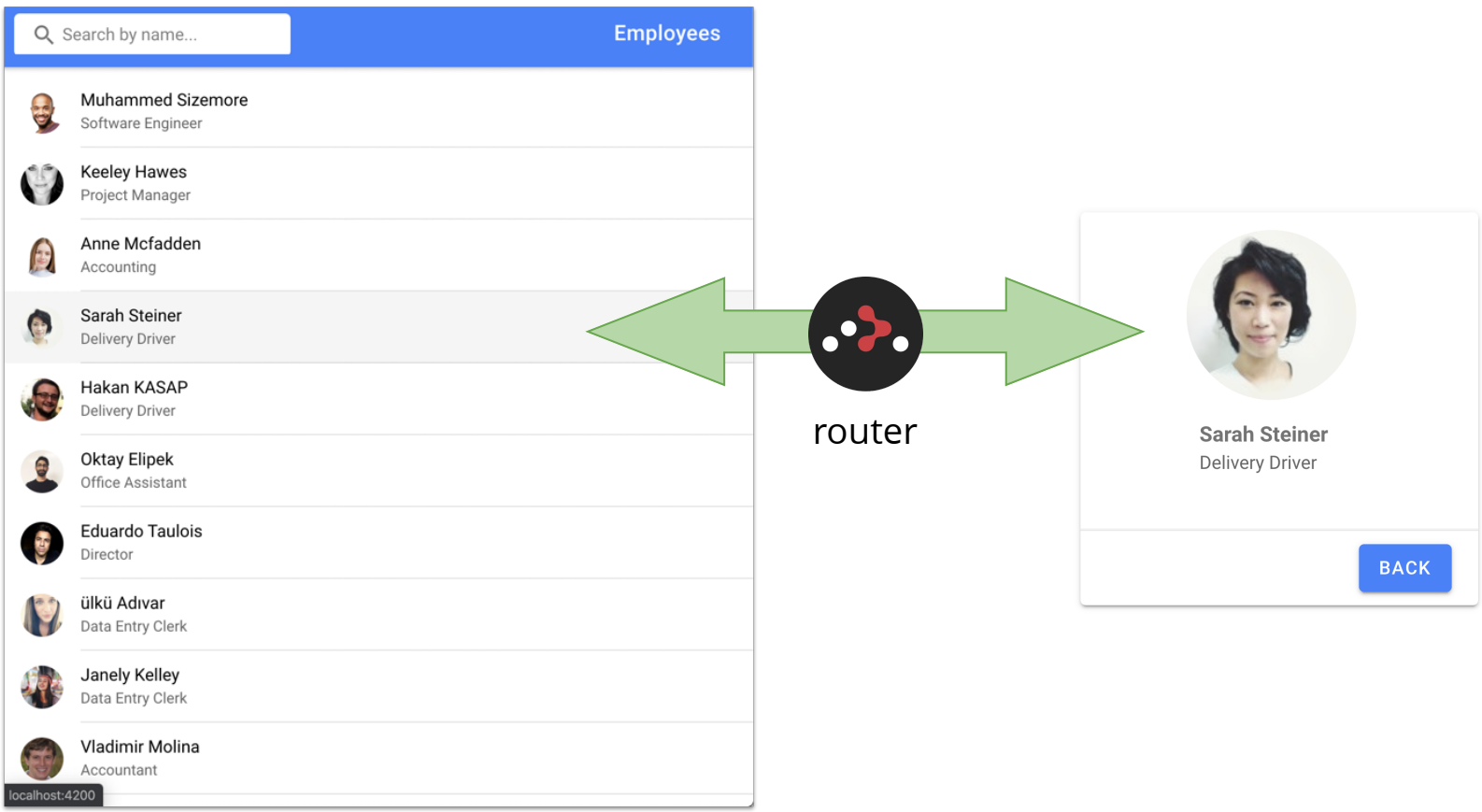
What is the 2nd parameter to React.FC
Why do we not specify it?

UI Component with **Types**

libs/contacts/ui/src/lib/**contacts-list.tsx**

```tsx
import React, { Component } from 'react';
import { Contact, ContactsService } from '@workshop/contacts/data-access';

export interface ContactDetailProps extends RouteComponentProps<{ id: string }> {}
export interface ContactDetailState { contact: Contact; }

export class ContactDetails extends Component<ContactDetailProps, ContactDetailState> {
  constructor(props) {
    super(props);
    this.state = { contact: {} as Contact };
  }

  private loadContact(id = '') {
    // Use Router param `id` to lookup
    const service = new ContactsService();
    const params = this.props.match.params;
    const request = service.getContactById(id || params.id);

    request.then(contact => this.setState({ contact }));
  }

  // ...
}
```

Lab 3b: Add UI Routing



Functional Components (React.FC)

```
// Class-based component
class Button extends React.Component {
  render(){
     return <button>{this.props.title}</button>
  }
}

// Functional component
const Button = (props) =>
  <button>{props.title}</button>
```

React Functional components are view components that can be composed into a tree of nested UI (DOM) elements.

Why Functional Components?

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

Functional Components:

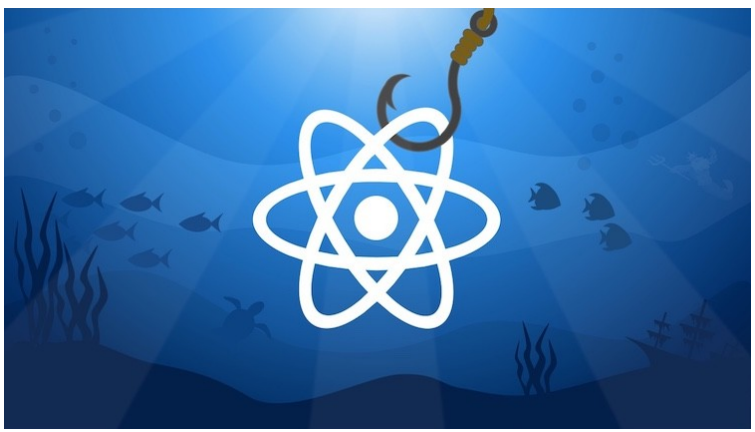
- accepts **props** from parent,
- do not have **lifecycle** methods,
- do not have **state**,
- centralize or **externalize** logic,
- are essentially a UI **renderer**,
- returns a React element (JSX)
- recommended **standard** (by Facebook),

Functional Components + **State**

React Hooks let you use **state** and other React features without writing a class.

5 React Hooks:

- useState()
- useEffect(),
- useParams()
- useContext()
- useRef()



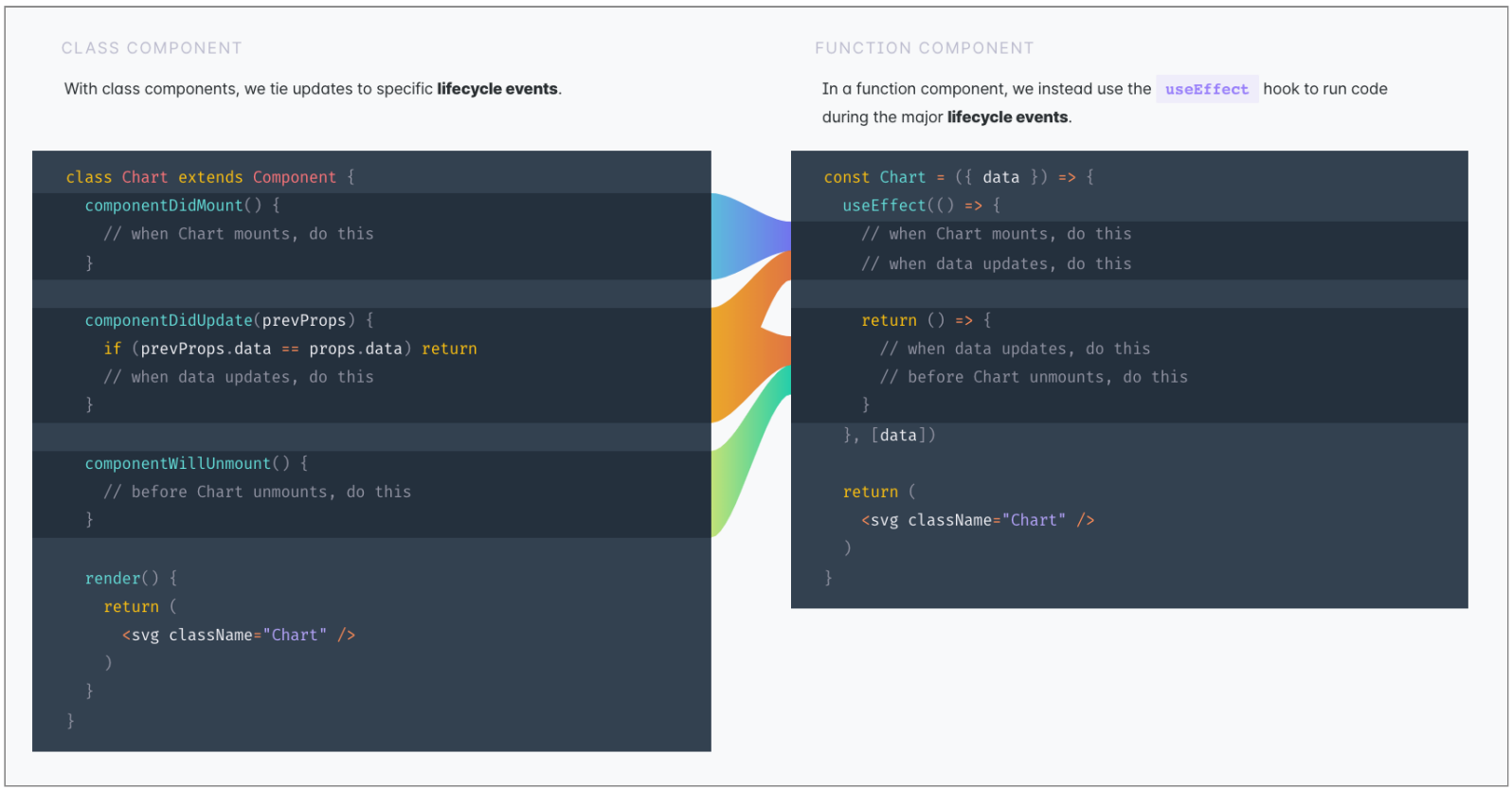
React Hooks lets developers separate JSX rendering from

(a) state construction, and

(b) change management.

Great React developers use Hooks!

Thinking in **React** Hooks



https://wattenberger.com/blog/react-hooks

Rules of Hooks

Only Call Hooks at the Top Level

Don't call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function.

```
export const ContactsList: React.FC = () => {
  const [people, setPeople] = useState([]);
  const [service]           = useState(() => new ContactsService());

  useEffect(() => {
    service.getContacts().then(setPeople);
  }, [service, setPeople]);

  return (
    ... JSX here
  );
}
```

Rules of Hooks

```tsx
export const ContactsList: React.FC = () => {
  const [people, setPeople] = useState([]);
  const [service]           = useState(() => new ContactsService());

  useEffect(() => {
    service.getContacts().then(setPeople);
  }, [service, setPeople]);

  return (
    ... JSX here
  );
}
```

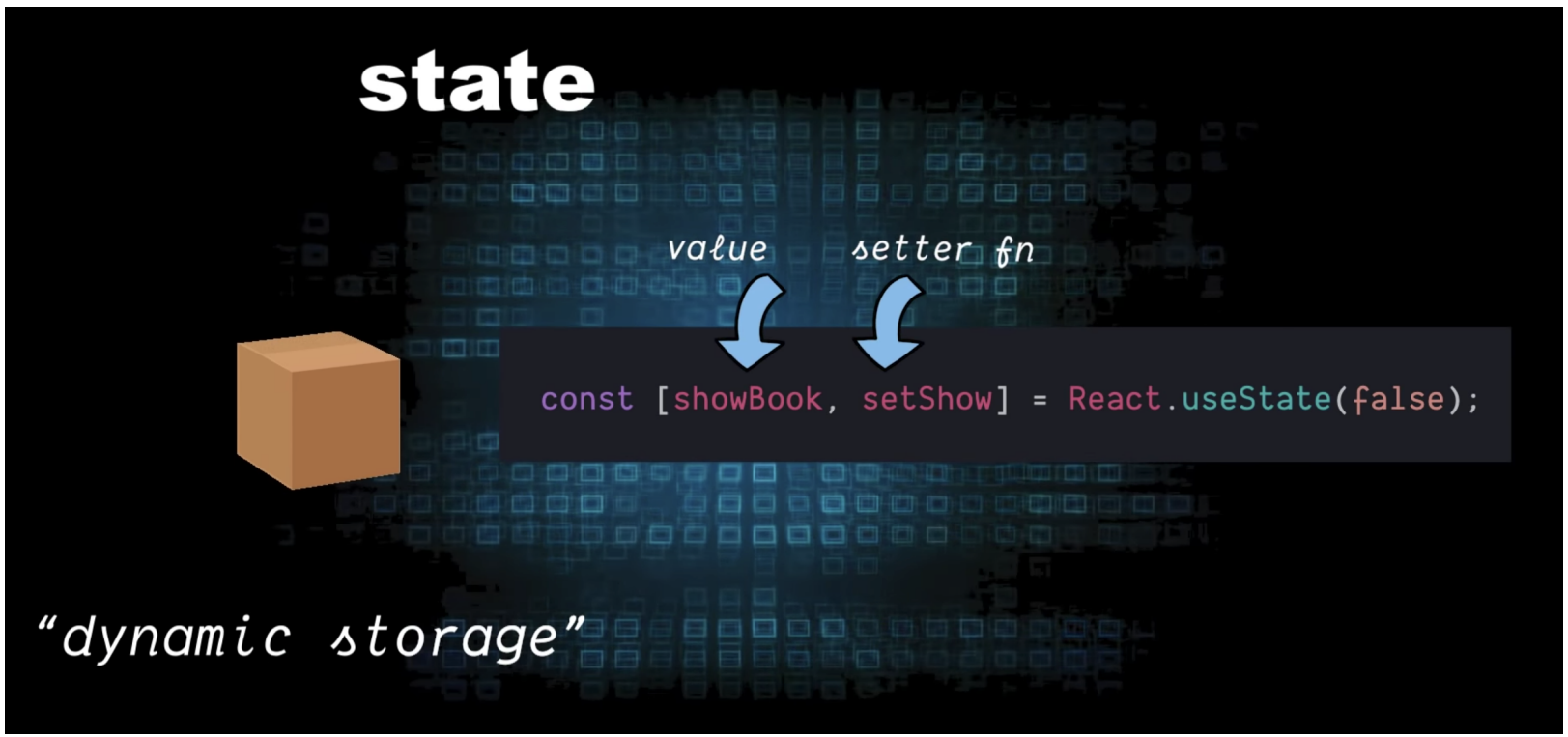
Only Call Hooks from React Functions

Don't call Hooks from regular JavaScript functions. Instead, you can:

- ✅ Call Hooks from React function components.
- ✅ Call Hooks from custom Hooks (we'll learn about them on the next page).

By following this rule, you ensure that all stateful logic in a component is clearly visible from its source code.

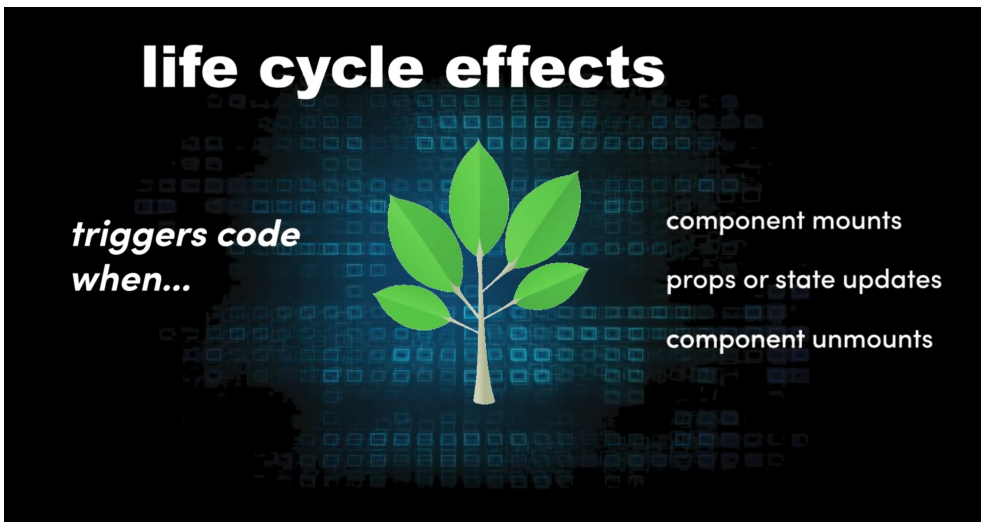
useState() Hook



Calling the setter fn **setShow** will automatically queue an *potential* render update with the latest boolean value.

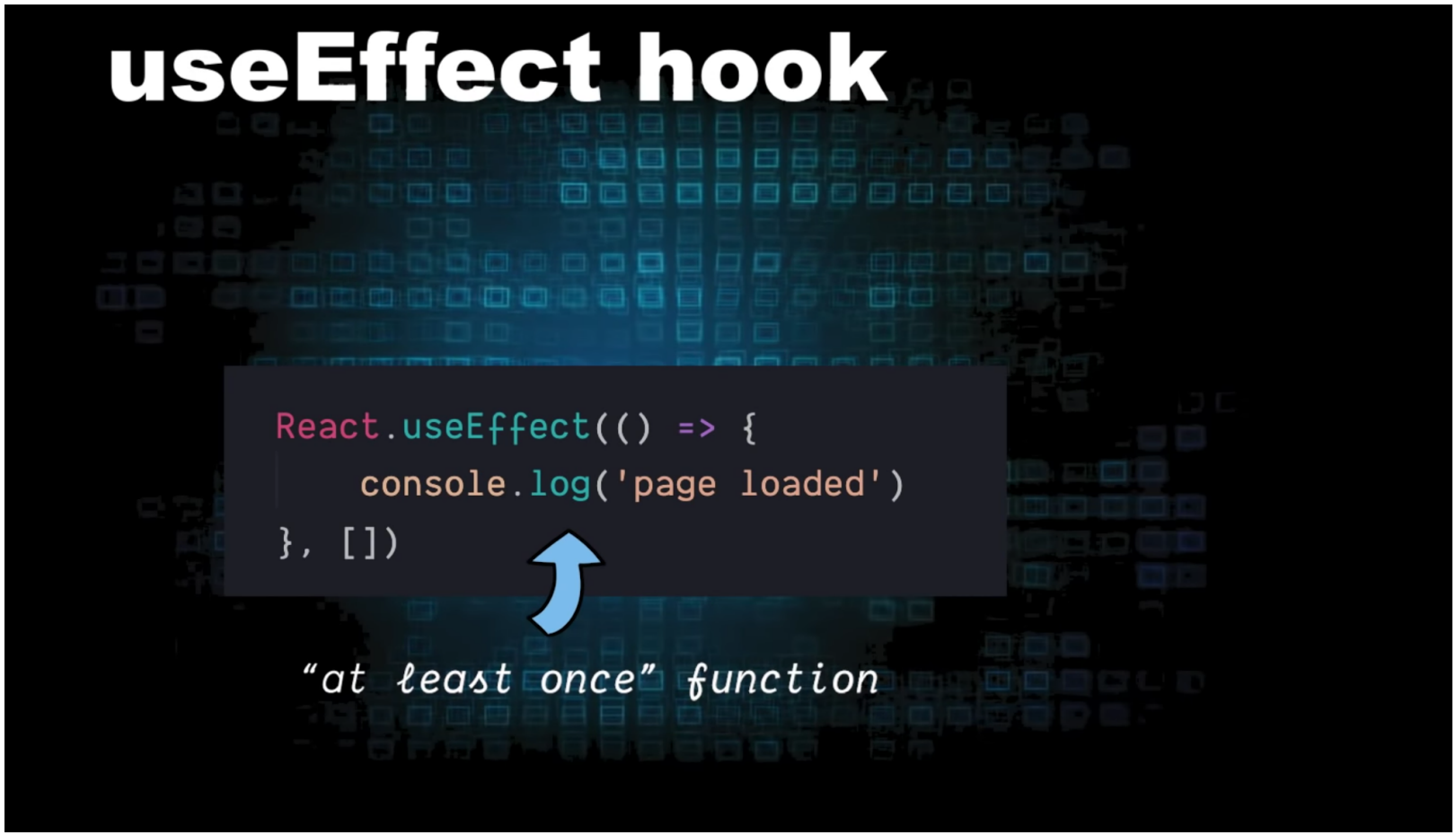
useEffect() Hook

You've likely performed data fetching, subscriptions, or manually changing the DOM from React components before. We call these operations "side effects" (or "effects" for short) because they can affect other components and can't be done during rendering.



The Effect Hook, `useEffect`, adds the ability to perform side effects from a function component. It serves the same purpose as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` in React classes, but unified into a single API. (We'll show examples comparing `useEffect` to these methods in Using the Effect Hook.)

useEffect() Hook



useEffect() Hook

Speaker notes

The function may be called >1x dependent upon the trigger variables specified.

useEffect() Hook

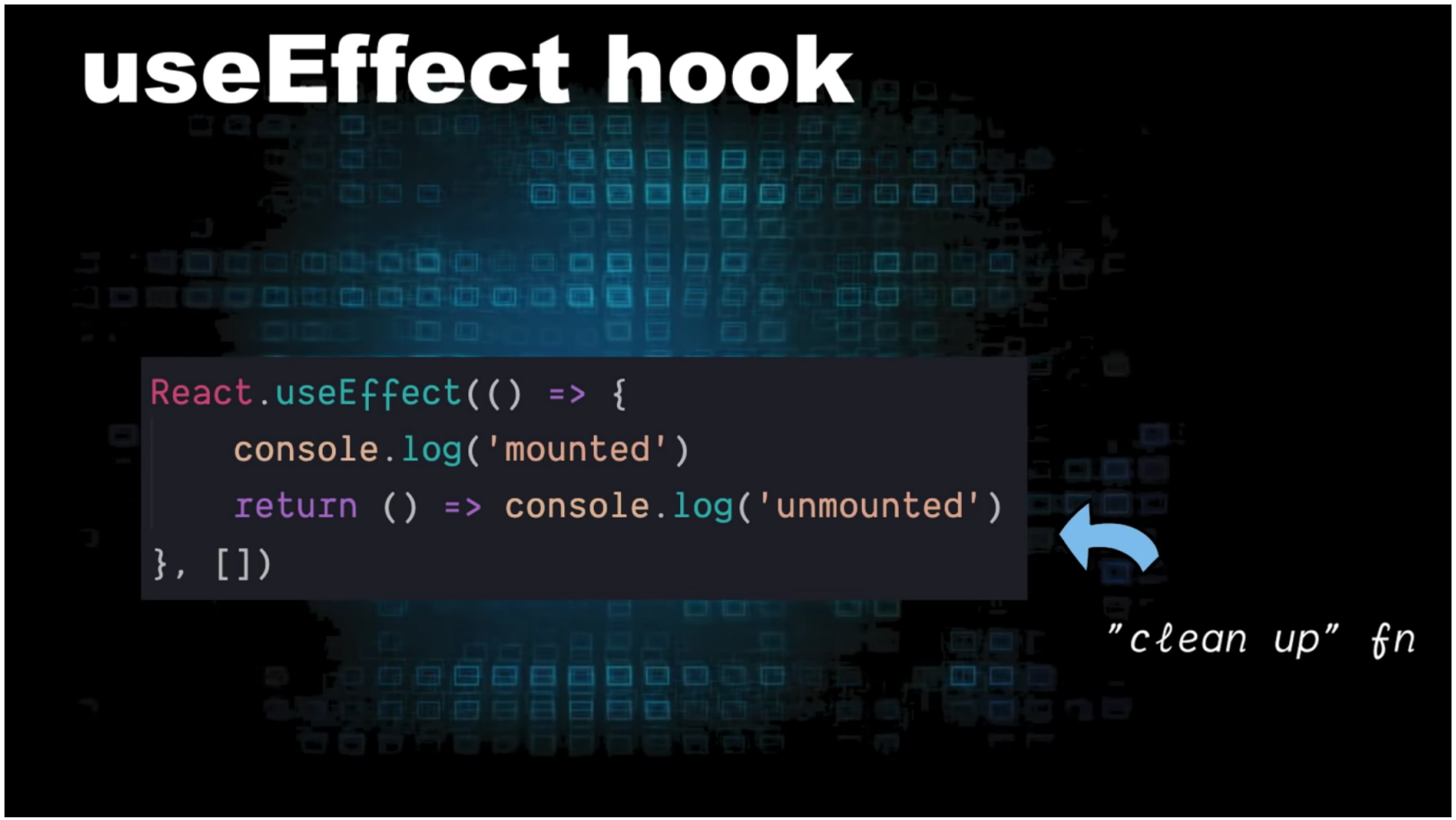
```
1  export const ContactsList: React.FC = () => {
2    const [people, setPeople] = useState([]);
3    const [service]           = useState(() => new ContactsService());
4
5    useEffect(() => {
6      service.getContacts().then(setPeople);
7    }, [service, setPeople]);
8
9    return (
10     ... JSX here
11   );
12 }
```

Run the effect function only when the `**service**` or `**setPeople**` function reference changes.

Speaker notes

Best rule of practice: if the effect function uses variables, register them in the trigger array.

useEffect() Hook

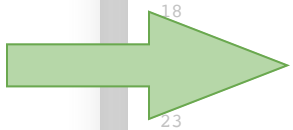


The optional, *clean-up fn* is great when resources **inside** the effect function should be cleaned up!

Convert UI Components?

Class

```
export class ContactsList extends Component {

  private service = new ContactsService();

  constructor(props) {
    super(props);
    this.state = { people: [] };
  }

  componentDidMount() {
    this.service.getContacts().then(list => {
      this.setState({ people: list });
    });
  }

  render() {
    const onSearch = doSearch(this.service, this.setState);

    return (
      ...
    );
  }
}

function doSearch(service, setState) {
  return (e: Event) => {
    const input = (e.target as HTMLIonInputElement);
    const criteria = input.value;
     service.searchBy(criteria).then(list => {
        setState({ people: list });
     });
  };
}
```



Functional

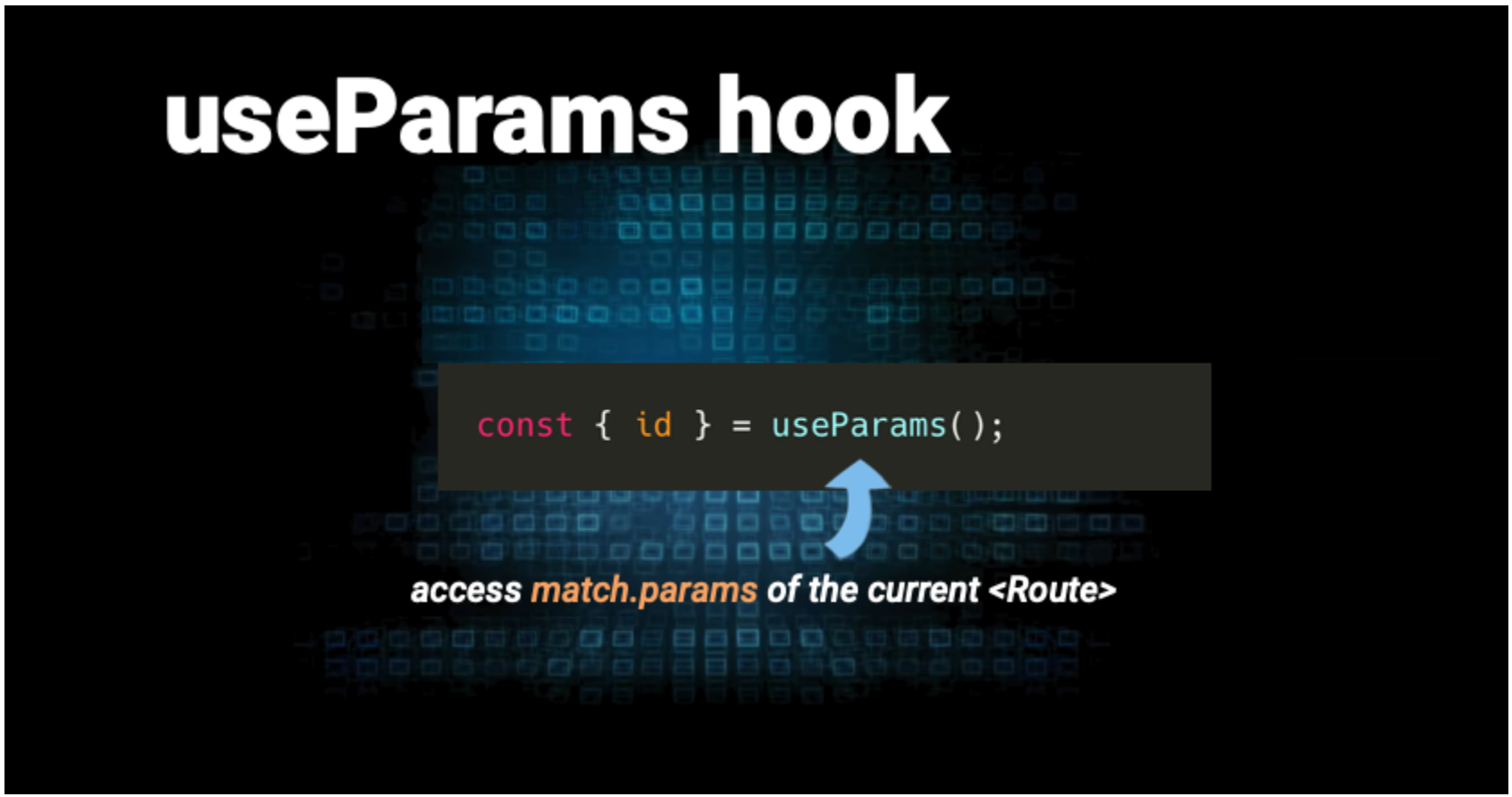
```
export const ContactsList: React.FC = () => {

  const [people, setPeople] = useState([]);
  const [service]           = useState(() => new ContactsService());
  const onSearch            = doSearch(service, setPeople);

  useEffect(() => {
    service.getContacts().then(setPeople);
  }, [service, setPeople]);


  return (
    ...
  );
};



function doSearch(service: ContactsService, setPeople) {
  return (e: Event) => {
    const input = (e.target as HTMLIonInputElement);
    const criteria = input.value;

    service.searchBy(criteria).then(setPeople);
  };
}
```

libs/contacts/ui/src/lib/**contacts-list.tsx**

Speaker notes

Wait... how many instances of ContactsService will be created?
Answer: Only 1; since we are using a deferral function.

useParams() Hook



useParams returns an object of key/value pairs of URL parameters.

Speaker notes

useParams returns an object of key/value pairs of URL parameters.

Convert UI Components?

Class

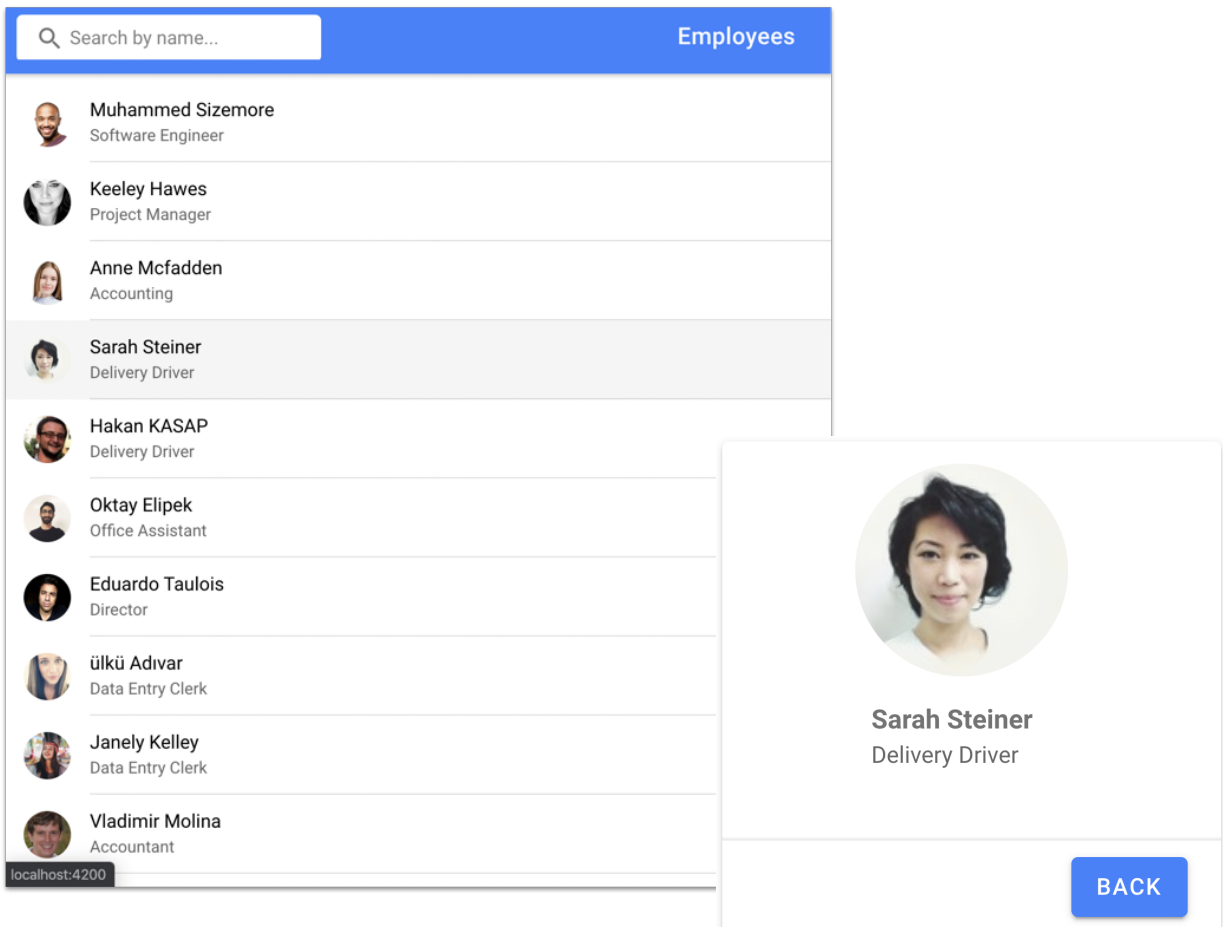
```
export class ContactDetails extends Component {

  constructor(props) {
    super(props);
    this.state = { contact: {} as Contact };
  }

  componentDidMount() {
    this.loadContact();
  }

  componentDidUpdate() {
    const newId = this.props.match.params.id;
    const oldId = this.state.contact ?
                  this.state.contact.id : '';

    if (newId !== oldId) {
      this.loadContact(newId);
    }
  }

  private loadContact(id = '') {
    const service = new ContactsService();
    const params  = this.props.match.params;
    const request = service.getContactById(id || params.id);

    request.then(contact => this.setState({ contact }));
  }

  render() {
    const contact = this.state.contact;
    return (
      ...
    );
  }
}
```

Functional

```
import { Redirect, useParams } from 'react-router';

export const ContactDetails: React.FC = () => {

  const { id }                = useParams();
  const [contact, setContact] = useState<Contact>({} as Contact);

  useEffect(() => {
    const service = new ContactsService();
    const request = service.getContactById(id);

    request.then(who => setContact(who));
  }, [id]);


  return contact ? ( ... ) : (
    <Redirect to="/contacts" />
  );

};
```

libs/contacts/ui/src/lib/**contact-detail.tsx**

Lab 4: Use Functional Components



Why Custom Hooks?

Our Functional Components use React hooks.
But still has **too much** logic.

This is what the **JSX** needs:

ContactsList

- renders a list of **contact**(s)
- uses `setCriteria()` to filter/search for new contacts

ContactDetails

- renders a **contact**
- uses `**history**` to navigate on Esc

Custom Hooks?

Encapsulates all logic and state changes

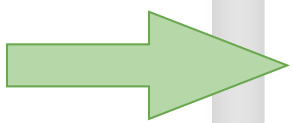
Reduces JSX to **presentational** UI.

Easy **testing** of logic and state changes

Convert UI Components?

React Hooks

```tsx
export const ContactsList: React.FC = () => {

  const [people, setPeople] = useState([]);
  const [service]           = useState(() => new ContactsService());
  const onSearch            = doSearch(service, setPeople);

  useEffect(() => {
    service.getContacts().then(setPeople);
  }, [service, setPeople]);


  return (
    ...
  );
};









function doSearch(service: ContactsService, setPeople) {
  return (e: Event) => {
    const input = (e.target as HTMLIonInputElement);
    const criteria = input.value;

    service.searchBy(criteria).then(setPeople);
  };
}
```



Custom Hook

```tsx
export const ContactsList: React.FC = () => {

  const [people, doSearch] = useContactsHook();

  return (
    ...
  );
};









function doSearch(service: ContactsService, setPeople) {
  return (e: Event) => {
    const input = (e.target as HTMLIonInputElement);
    const criteria = input.value;

    service.searchBy(criteria).then(setPeople);
  };
}
```

libs/contacts/ui/src/lib/**contacts-list.tsx**

Custom Hook

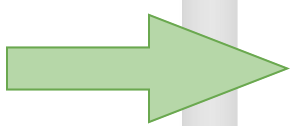
```tsx
/**
 * Define tuple types
 */
export type ContactHookResults = [Contact[], (criteria: string) => void];


/**
 * Custom React Hook useful to search and load Contacts
 */
export function useContactsHook(): ContactHookResults {
  const [service] = useState(() => new ContactsService());
  const [criteria, setCriteria] = useState<string>('');
  const [people, setPeople] = useState<Contact[]>([]);

  useEffect(() => {
    service.searchBy(criteria).then(setPeople);
  }, [criteria, service, setPeople]);

  return [people, setCriteria];
}
```

libs/contacts/ui/src/lib/**contacts.hooks.ts**

Convert UI Components?

React Hooks

```tsx
import { Redirect, useParams } from 'react-router';

export const ContactDetails: React.FC = () => {

  const { id }                = useParams();
  const [contact, setContact] = useState<Contact>({} as Contact);

  useEffect(() => {
    const service = new ContactsService();
    const request = service.getContactById(id);

    request.then(who => setContact(who));
  }, [id]);


  return contact ? ( ... ) : (
    <Redirect to="/contacts" />
  );

};
```



Custom Hook

```tsx
import { Redirect, useParams } from 'react-router';

export const ContactDetails: React.FC = () => {

  const [contact, history] = useContactDetailHook();

  return contact ? ( ... ) : (
    <Redirect to="/contacts" />
  );

};
```

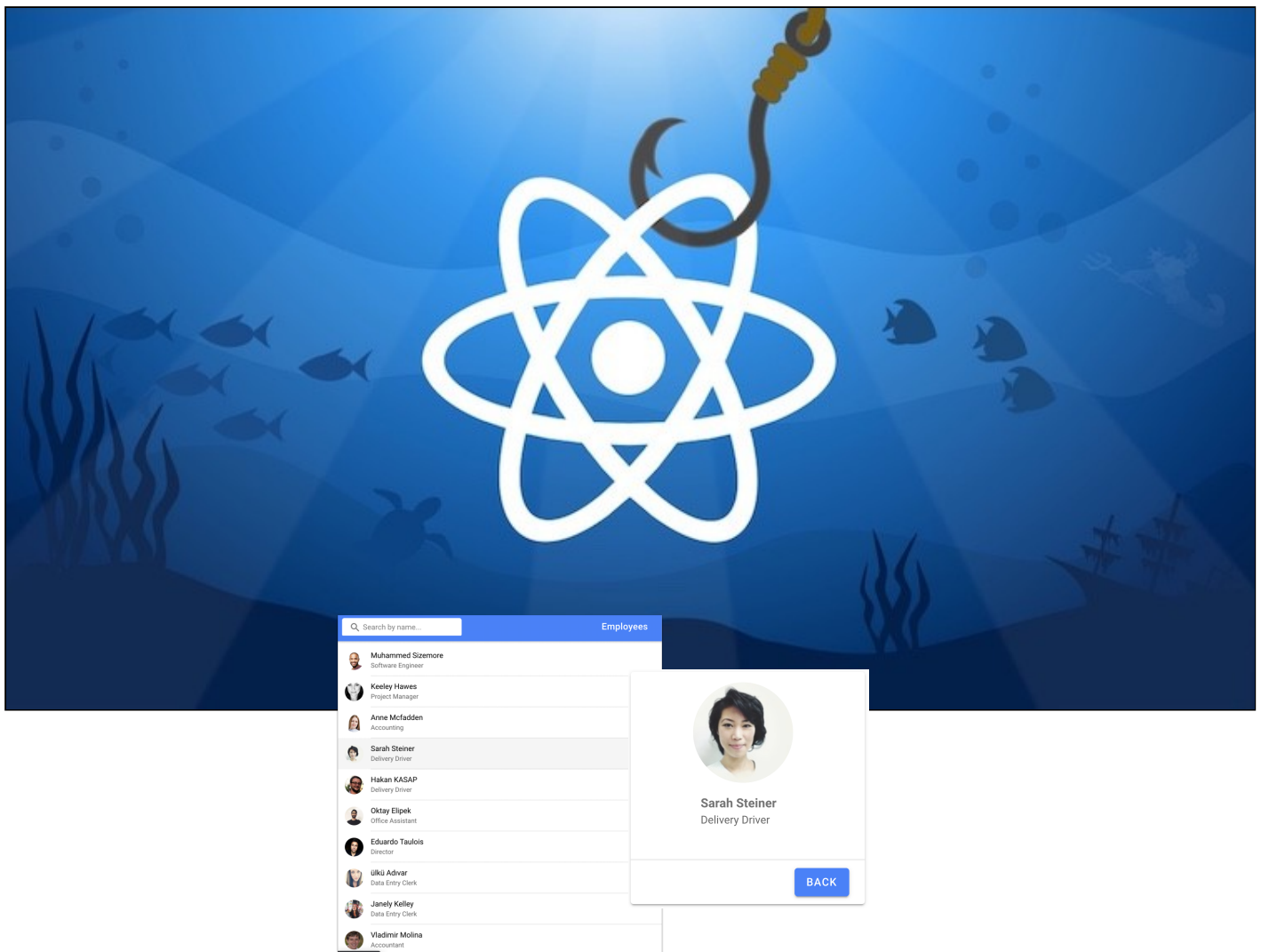
libs/contacts/ui/src/lib/**contact-detail.tsx**

Custom Hook

```tsx
/**
 * Define tuple types
 */
export type ContactDetailsResult = [Contact, H.History<H.LocationState>];

/**
 * Custom React Hook useful to load Contact details
 */
export function useContactDetailHook(): ContactDetailsResult {
  const { id } = useParams();
  const history = useHistory();
  const [service] = useState(() => new ContactsService());
  const [contact, setContact] = useState<Contact>({} as Contact);

  useEffect(() => {
    service.getContactById(id).then(setContact);
  }, [id]);

  return [contact, history];
}
```

libs/contacts/ui/src/lib/**contacts.hooks.ts**

Lab 5: Use Custom Hooks



Our **Custom** Hook

```
1
2  /**
3   * Custom hook: load and filter Contacts
4   */
5  export function useContactsHook(): ContactHookResults {
6    const [criteria, setCriteria] = useState<string>('');
7    const [people, setPeople]     = useState<Contact[]>([]);
8
9    const [service] = useState(() => new ContactsService());
10
11   useEffect(() => {
12     service.searchBy(criteria).then(setPeople);
13   }, [criteria, service, setPeople]);
14
15   return [people, setCriteria];
16 }
```

libs/contacts/ui/src/lib/**contacts.hooks.ts**

Speaker notes

Can we share the service instance within the entire application ?

React **Context** Hook

Share **services** and **data** in the view hierarchy... without prop drilling.

React **Context** Hook

1. **Create** a Context object
2. **Assign** Context value
3. **Use** Context value

```
1 import { createContext } from 'react';
2 import { ContactsService } from './contacts.service';
3
4 /**
5  * Used to provide ContactsService in the JSX tree
6  * or a `useContext()` hook
7  */
8 export const ContactsContext = createContext<ContactsService>(null);
```

React **Context** Hook

1. **Create** a Context object
2. **Assign** Context value
3. **Use** Context value

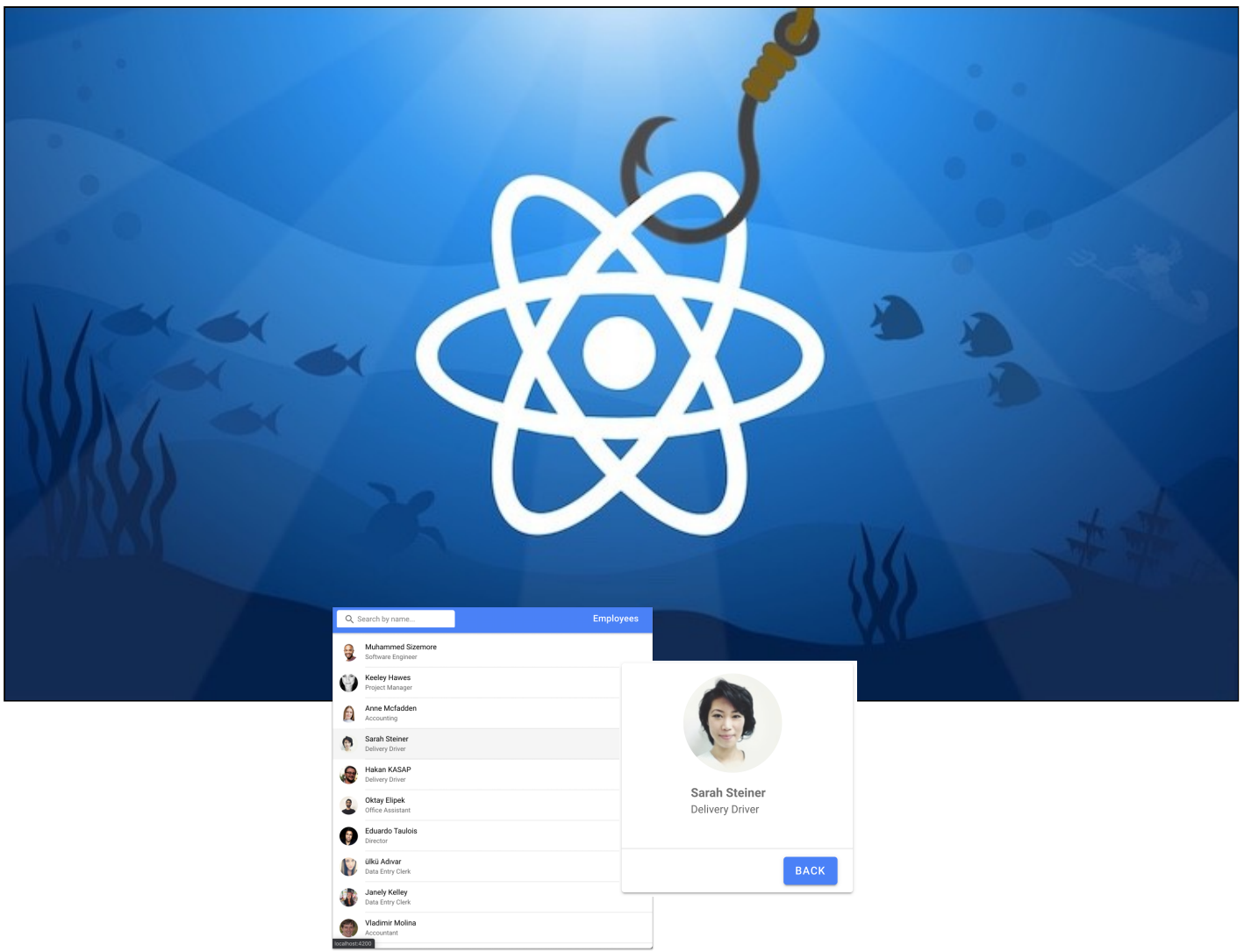
```
1  import { ContactsList } from './contacts-list';
2  import { ContactDetails } from './contact-detail';
3  import {
4    ContactsService,
5    ContactsContext
6  } from '@workshop/contacts/data-access';
7
8
9  export const ContactsDashboard: React.FC = () => {
10   return (
11     <ContactsContext.Provider value={new ContactsService()}>
12       <Router>
13           <Route path="/contacts"     exact component={ContactsList} />
14           <Route path="/contacts/:id" exact component={ContactDetails} />
15           <Redirect exact from="/" to="/contacts" />
16       </Router>
17     </ContactsContext.Provider>
18   );
19 };
```

React **Context** Hook

1. **Create** a Context object
2. **Assign** Context value
3. **Use** Context value

```
1  import { ContactsContext } from './contacts.injector';
2  import { ContactsService } from './contacts.service';
3
4  /**
5   * Custom React Hook useful to search and load Contacts
6   */
7  export function useContactsHook(): ContactHookResults {
8    const [service] = useState(() => new ContactsService());
9    const [criteria, setCriteria] = useState<string>('');
10   const [people, setPeople] = useState<Contact[]>([]);
11   const service = useContext<ContactsService>(ContactsContext);
12
13   useEffect(() => {
14     service.searchBy(criteria).then(setPeople);
15   }, [criteria, service, setPeople]);
16
17   return [people, setCriteria];
18 }
```

Lab 6a: useContext Hook



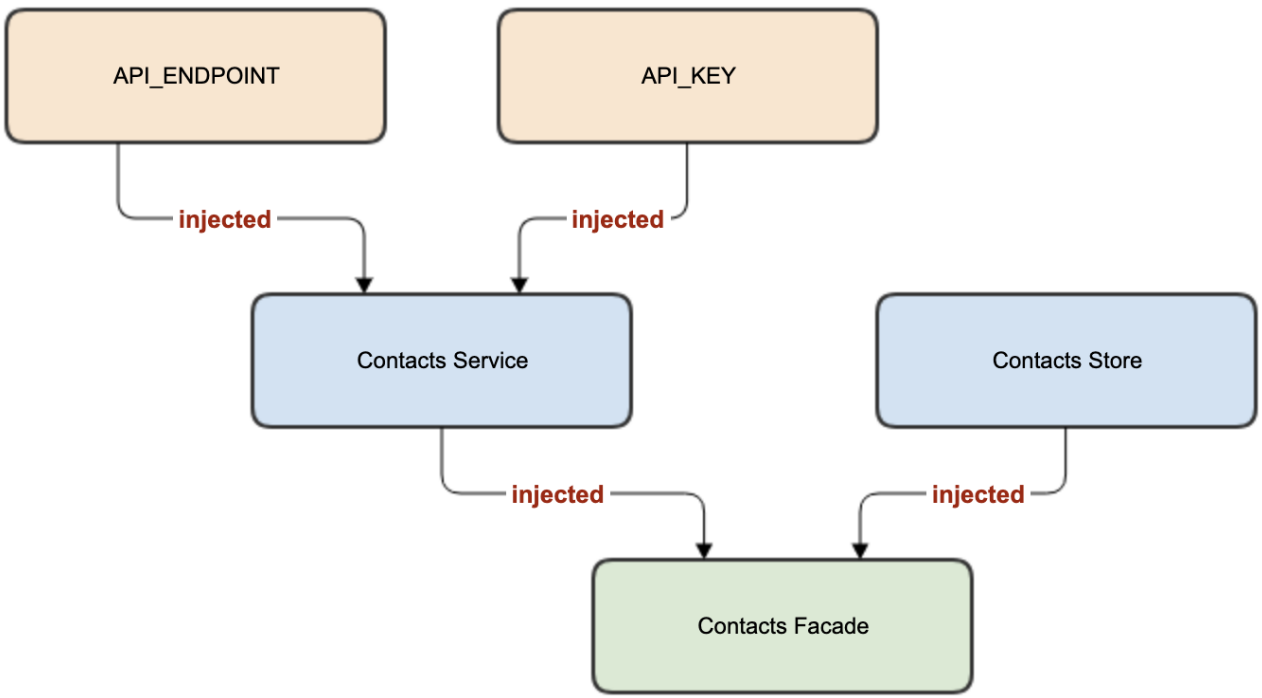
React Context is **not enough!**

Context can help us with sharing instances/data across nested view components + hooks.

- What about sharing **within non-ui** services?
- What about non-trivial **instantiation** requirements
- What about **run-time configuration** of services
  - mocks
  - environment setting
  - etc.

Dependency Injection solves these issues!

Context does not solve this:



Issue #1: **Runtime Configuration**

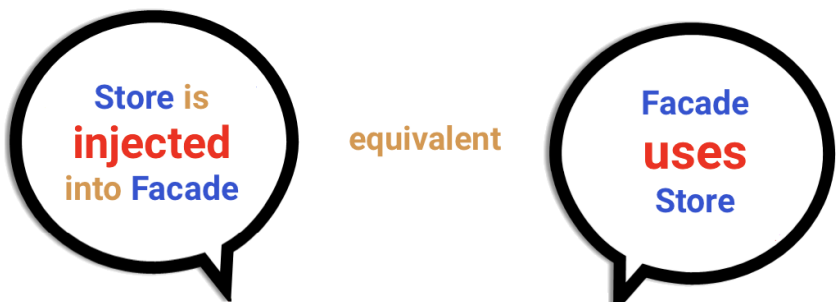
```
const API_ENDPOINT = 'https://uifaces.co/api?limit=25;';
const API_HEADERS = { headers: { 'x-API-KEY': '873771d7760b846d51d025ac5804ab' } };


async function fetchFaces<T>({ userName }: QueryParams | null): Promise<T> {
  const url = API_ENDPOINT + (userName ? `&name=${userName}` : '');
  const result = await fetch(url, API_HEADERS);
  const data = await result.json();
  return <T>data;
}

export class ContactsService {

  getContacts(skipCache = false, params: QueryParams = null): Observable<Contact[]> { }
  getContactById(id: string): Observable<Contact | undefined> { }
  updateContact(contact: Contact): Observable<Contact> { }
  searchBy(userName: string, title: string = ''): Observable<Contact[]> { }

}

```

Solution #1: Allow **External Inputs**

```
export class ContactsService implements ContactsService {

  constructor(private apiEndPoint: string, private apiKey: string) {}

}
```

Our service has dependencies on two (2) API values.
We cannot instantiate ContactsService without those values.

Dependency Injection solves these issues!



Dependency Injection

Dependency Injection (**DI**)

Dependency injection is a technique whereby a system supplies the **dependencies** instances to a target object instance.

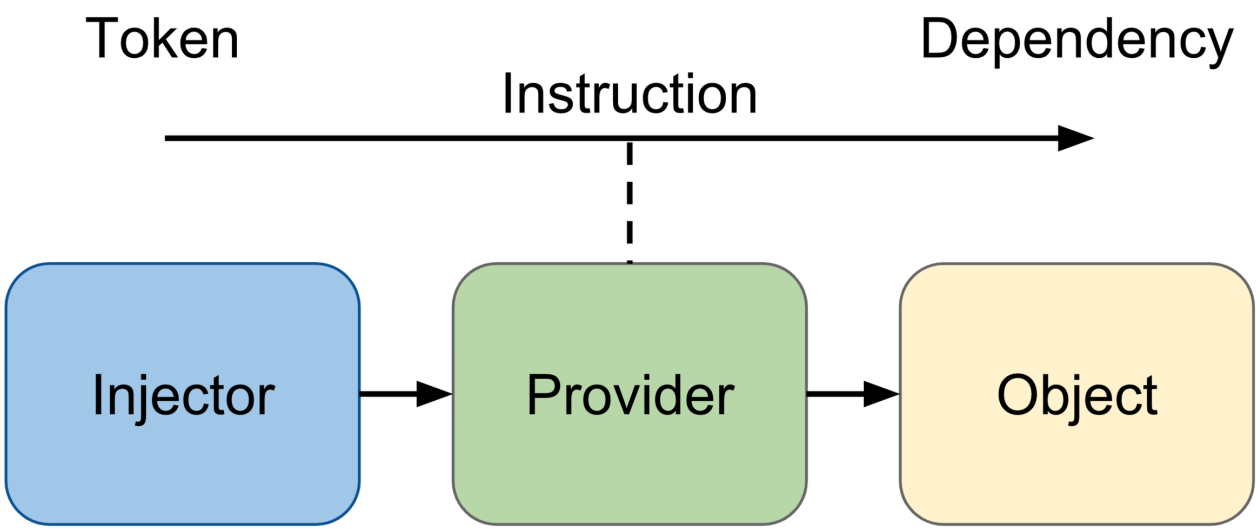
- The DI engine **manages** instantiations and caching
- The DI engine provides **easy registration** of configurations
- The DI engine provides **easy lookups** of instances
- Use DI to separate **USE** from **CONSTRUCTION**
- Enables **easy testing** with **Mocks**!
- Enables cached, shared services or state

Dependency Injection (**DI**)

- Independent of **Angular**, **React**, **Ionic**, **Vue**, or **StencilJS**
- Support construction & lookups using **tokens**
- Configure the DI system for custom application requirements
- Provide an easy way to access **singleton** (shared) instances
- Provide an easy way to create a non-shared, **localized instance**
- Provide an easy way to **extend** existing configurations
- Provide an easy way to **override** existing configurations

Inside Dependency Injection

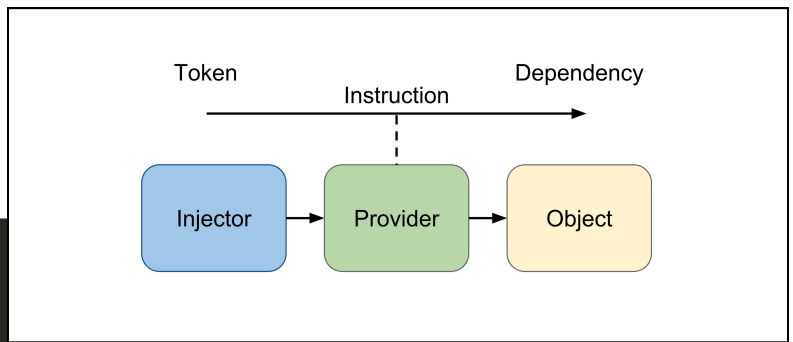
Dependency injection uses a lookup process to determine **HOW** to create an object.



Configure Dependency Injection

Dependency injection can be **configured** easily with Providers:

Provider

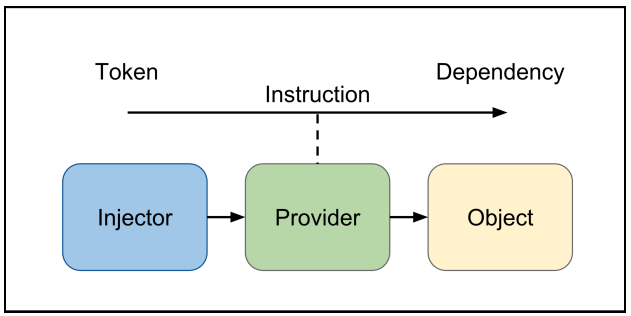



```
{

  provide     : <token>,

  useValue    ?: any,
  useClass    ?: (deps?:any[]) => any,
  useFactory  ?: (deps?:any[]) => any,

}
```

Configure Dependency Injection

What is a **Token**? This is an ID object used to lookup the cached object instance.

- Class
- string (not unique)
- InjectionToken



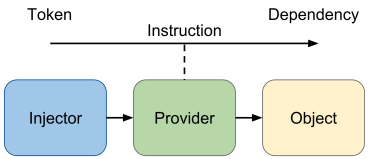
Use an `InjectionToken` whenever the type you are injecting is not reified (does not have a runtime representation) such as when injecting an interface, callable type, array or parameterized type.

Unique tokens using **InjectionToken**

```
import { InjectionToken } from '@mindspace/core';

export const API_ENDPOINT = new InjectionToken('endpoint.ui-faces.com');
export const API_KEY      = new InjectionToken('api-key.ui-faces.com');
```

Configure Dependency Injection

Dependency injection can be **configured** easily with Providers:



Each injector has 1...n providers

```
import { makeInjector, DependencyInjector } from '@mindspace/core';

import { ContactsService, API_ENDPOINT, API_KEY } from './contacts.service';
import { ContactsFacade } from './contacts.facade';

/**
 * Create a DependencyInjector for Contacts features: service + facade
 */
export const injector: DependencyInjector = makeInjector([
  { provide: API_KEY,         useValue: '873771d7760b846d51d025ac5804ab'               },
  { provide: API_ENDPOINT,    useValue: 'https://uifaces.co/api?limit=25'              },
  { provide: ContactsService, useClass: ContactsService, deps: [API_ENDPOINT, API_KEY] },
  { provide: ContactsFacade,  useClass: ContactsFacade,  deps: [ContactsService]       }
]);
```

Dependency Injection (**DI**)

- Angular has DI using **@Injectable()**
- React provides lookups using **Context.**

NOTE: this is only part of ID

View-level dependency injection using **import**

```
import React, { useState } from 'react';
import { ContactsFacade, injector } from '../../+state';

export const ContactsList: React.FC = () => {
  const [facade] = useState<ContactsFacade>( injector.get(ContactsFacade) );

  // ...
}
```

The injected instance could be:

- ContactsFacade
- Mock
- Spy

Using **Dependency Injection**

1. Update **ContactsService** to support constructor DI
2. Define and instantiate a **Contacts** injector.
3. Use injector in **Contacts** hooks
4. Remove use of **Context**

```
export const API_ENDPOINT = new InjectionToken('endpoint.ui-faces.com');
export const API_KEY = new InjectionToken('api-key.ui-faces.com');

export class ContactsService {
  constructor(private apiEndPoint: string, private apiKey: string) {}

  // ...
}
```

libs/contacts/ui/src/lib/**contacts.service.ts**

Using **Dependency Injection**

1. Update **ContactsService** to support constructor DI
2. Define and instantiate a **Contacts** injector.
3. Use injector in **Contacts** hooks
4. Remove use of **Context**

```ts
import { makeInjector, DependencyInjector } from '@mindspace-io/utils';
import { ContactsService, API_ENDPOINT, API_KEY } from './contacts.service';

/**
 * Create a DependencyInjector for Contacts features: service + facade
 */
export const injector: DependencyInjector = makeInjector([
  { provide : API_KEY,      useValue: '873771d7760b846d51d025ac5804ab' },
  { provide : API_ENDPOINT, useValue: 'https://uifaces.co/api?limit=25' },
  {
    provide : ContactsService,
    useClass: ContactsService,
    deps    : [API_ENDPOINT, API_KEY]
  }
]);
```

libs/contacts/ui/src/lib/**contacts.injector.ts**

Using **Dependency Injection**

1. Update **ContactsService** to support constructor DI
2. Define and instantiate a **Contacts** injector.
3. Use injector in **Contacts** hooks
4. Remove use of **Context**

```
1  import * as H from 'history';
2  import { useState, useEffect } from 'react';
3  import { useHistory } from 'react-router-dom';
4  import { useParams } from 'react-router';
5  import { Contact } from '@workshop/shared/api';
6
7  import { injector } from './contacts.injector';
8  import { ContactsService } from './contacts.service';
9
10
11 export function useContactsHook(): ContactHookResults {
12   const [criteria, setCriteria] = useState<string>('');
13   const [people, setPeople] = useState<Contact[]>([]);
14   const [service] = useState(injector.get(ContactsService));
15
16   useEffect(() => {
17     service.searchBy(criteria).then(setPeople);
18   }, [criteria, service, setPeople]);
19
20   return [people, setCriteria];
21 }
```

libs/contacts/ui/src/lib/**contacts.hooks.ts**

Using **Dependency Injection**

1. Update **ContactsService** to support constructor DI
2. Define and instantiate a **Contacts** injector.
3. Use injector in **Contacts** hooks
4. Remove use of **Context**

```
1  import { ContactsService, ~~ContactsContext~~ } from '@workshop/contacts/data-access';
2
3  import { ContactsList } from './contacts-list';
4  import { ContactDetails } from './contact-detail';
5
6  export const ContactsDashboard: React.FC = () => {
7    return (
8      ~~<ContactsContext.Provider value={new ContactsService()}>~~
9        <IonReactRouter>
10         <IonRouterOutlet>
11           <Route path="/contacts" exact={true} component={ContactsList} />
12           <Route path="/contacts/:id" exact={true} component={ContactDetails} />
13           <Redirect exact from="/" to="/contacts" />
14         </IonRouterOutlet>
15       </IonReactRouter>
16     ~~</ContactsContext.Provider>~~
17   );
18 };
19
```

libs/contacts/ui/src/lib/**contacts.dashboard.ts**

Lab 6b: Use Dependency Injection (DI)