

The background is a solid teal color with various white line drawings and sketches overlaid. These include architectural elements like a brick wall, a staircase, and a building facade. There are also mechanical or technical drawings, such as a circular component with internal lines and a ladder-like structure. A small crescent moon is visible in the upper right corner.

Reactive Angular met RxJS

Werken met modules

Peter Kassenaar –
info@kassenaar.com



Multiple modules

Splitting your application into separate, reusable modules

Default application – 1 module

The image displays the Angular CLI welcome interface and the file structure of a default application.

Angular CLI Welcome Screen:

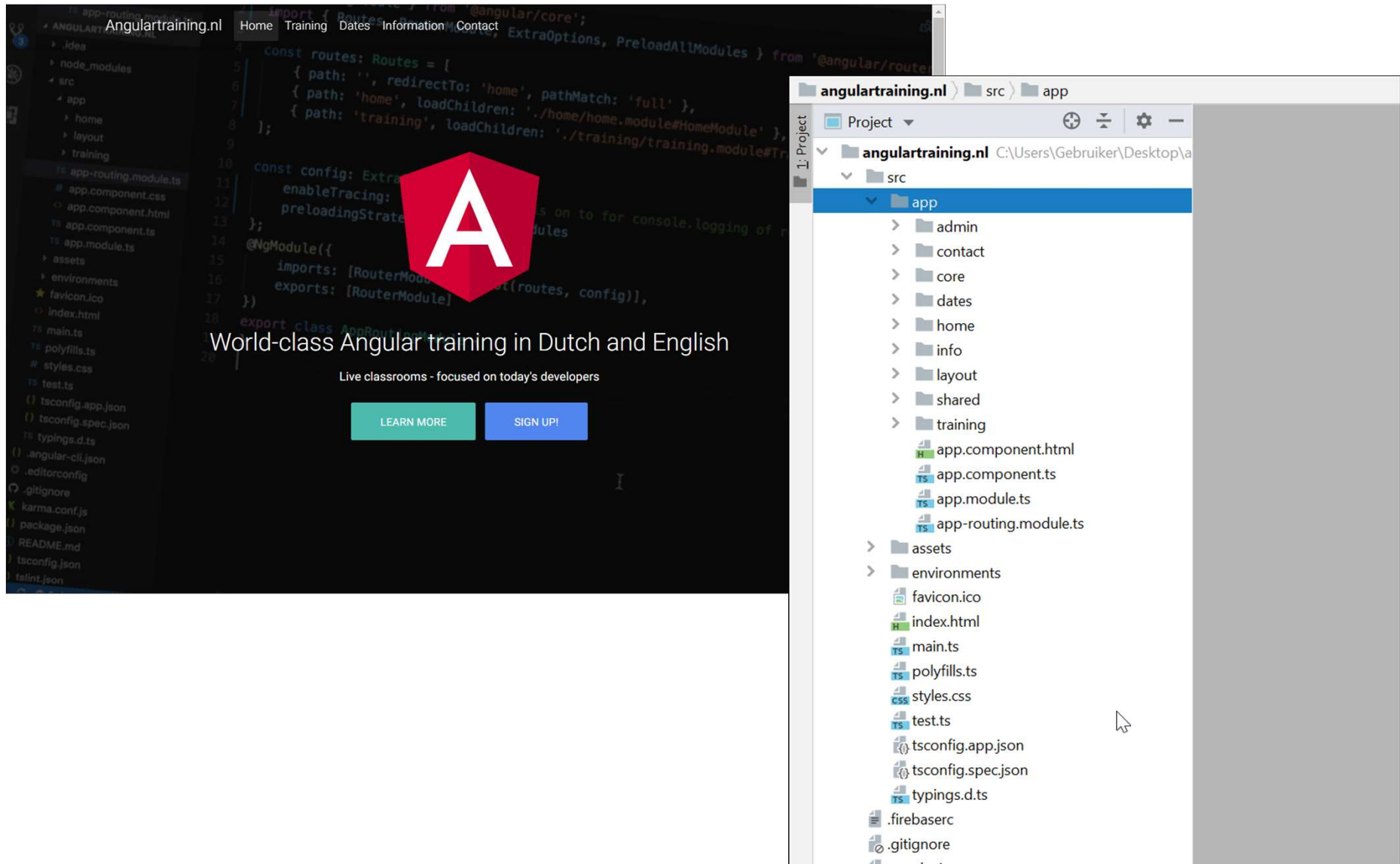
- Header:** "Welcome" with the Angular logo and a Twitter icon.
- Status:** A red rocket icon and a blue box stating "multiple-modules app is running!".
- Resources:** "Here are some links to help you get started:" with buttons for "Learn Angular >", "<> CLI Documentation >", and "Angular B".
- Next Steps:** "What do you want to do next with your app?" with buttons for "New Component", "Angular Material", "Add Dependency", "Build for Production", and a "+" button.
- Terminal:** A black terminal window showing the command `ng generate component xyz`.
- Footer:** Social media icons and a link to "Love Angular? Give our repo a star." with a "★ Star" button.

File Explorer (Project Structure):

- Project:** `customProject` (C:\Users\Peter Kassenaar\Desktop\custo)
- Project Files:**
 - `e2e`
 - `node_modules` (library root)
 - `src`
 - `app`
 - `app.component.css`
 - `app.component.html`
 - `app.component.spec.ts`
 - `app.component.ts`
 - `app.module.ts`
 - `assets`
 - `.gitkeep`
 - `environments`
 - `environment.prod.ts`
 - `environment.ts`
 - `favicon.ico`
 - `index.html`
 - `main.ts`
 - `polyfills.ts`
 - `styles.css`
 - `test.ts`
 - `tsconfig.app.json`
 - `tsconfig.spec.json`
 - `typings.d.ts`
 - `.angular-cli.json`
 - `.editorconfig`
 - `.gitignore`
 - `karma.conf.js`
 - `package.json`
 - `protractor.conf.js`
 - `README.md`
 - `tsconfig.json`
 - `tslint.json`
 - `yarn.lock`
 - `External Libraries`

(228 MB)

Bigger applications – multiple modules



The image displays a screenshot of an Angular application project structure and code. On the left, a file explorer shows the project layout, including folders like `src`, `app`, `home`, `layout`, and `training`. The `app` folder contains files such as `app-routing.module.ts`, `app.component.css`, `app.component.html`, `app.component.ts`, `app.module.ts`, `assets`, `environments`, `favicon.ico`, `index.html`, `main.ts`, `polyfills.ts`, `styles.css`, `test.ts`, `tsconfig.app.json`, `tsconfig.spec.json`, `typings.d.ts`, `.angular-cli.json`, `.editorconfig`, `.gitignore`, `karma.conf.js`, `package.json`, `README.md`, `tsconfig.json`, and `tslint.json`.

The main area shows the `app.module.ts` file, which defines the root module. It includes the following code:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.module#HomeModule';
import { TrainingComponent } from './training/training.module#TrainingModule';

const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', loadChildren: './home/home.module#HomeModule' },
  { path: 'training', loadChildren: './training/training.module#TrainingModule' },
];

const config: ExtraOptions = {
  enableTracing: true,
  preloadingStrategy: PreloadAllModules,
};

@NgModule({
  imports: [BrowserModule, AppRoutingModule],
  exports: [RouterModule],
  providers: [],
  declarations: [AppComponent],
})
export class AppModule { }

export class AppRoutingModule { }
```

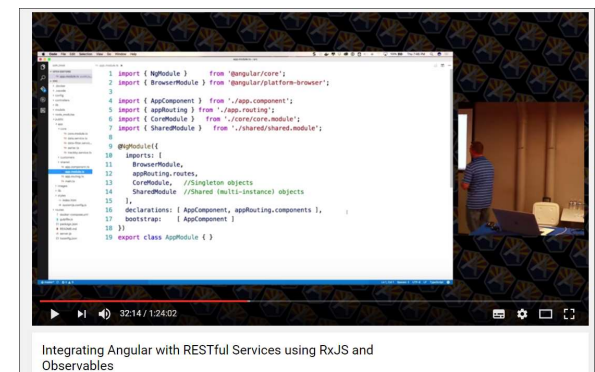
Below the code, the text "World-class Angular training in Dutch and English" is displayed, along with the tagline "Live classrooms - focused on today's developers". Two buttons, "LEARN MORE" and "SIGN UP!", are visible.

On the right, a project structure view shows the `angulartraining.nl` project, with the `src` folder expanded to show the `app` folder. The `app` folder contains the following files and folders:

- `admin`
- `contact`
- `core`
- `dates`
- `home`
- `info`
- `layout`
- `shared`
- `training`
- `app.component.html`
- `app.component.ts`
- `app.module.ts`
- `app-routing.module.ts`
- `assets`
- `environments`
- `favicon.ico`
- `index.html`
- `main.ts`
- `polyfills.ts`
- `styles.css`
- `test.ts`
- `tsconfig.app.json`
- `tsconfig.spec.json`
- `typings.d.ts`
- `.firebaserc`
- `.gitignore`

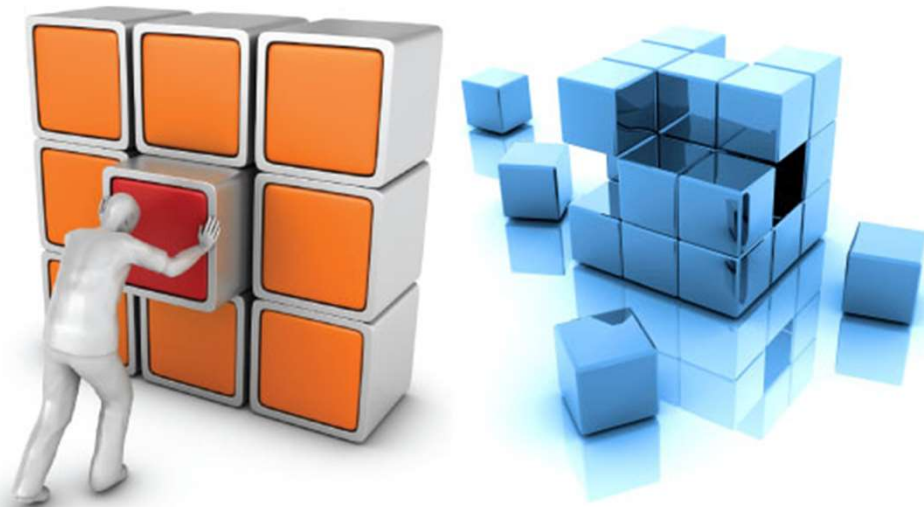
Angular Modules

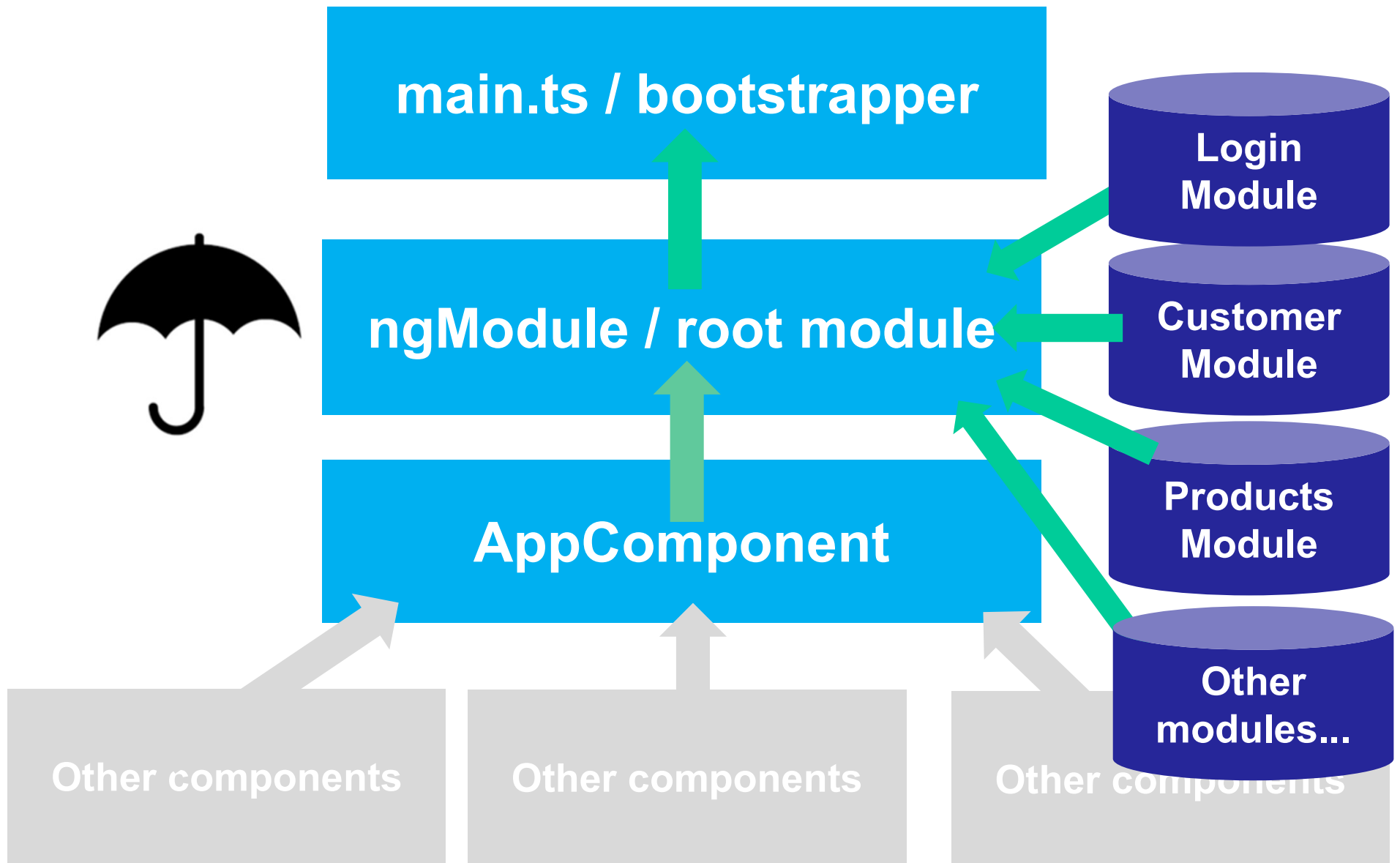
- Divide your app into *logical* and often *reusable* pieces of code
- Keyword : **code organization**
- Use one AppModule - the root of your app
- Use one CoreModule - containing all *singletons* in your app
- Use one SharedModule - containing all shared resources, possible multiple instances
- Use additional modules *per feature*
- <https://www.youtube.com/watch?v=YxK4UW4UfCk>



Application – multiple Modules – why?

- *Reuse* of Components, Pipes, Routes and Services etc. over different apps
- *Wrap* each set of logical related components, services, etc. in its own module.





Steps

1. Create a new module

- Optional: test first with `--dry-run`
- `ng generate module customers --dry-run`

2. Create component(s) inside that module

- Again: test first with `--dry-run`
- `ng generate component customers --module customers --dry-run`

3. Apply UI, logic, etc. to your component

4. Export your component inside `customer.module.ts`

- `exports : [CustomerComponent],`
- Otherwise it can't be used in other components!

5. Provide new module to `app.module.ts`

- `imports: [CustomerModule]`

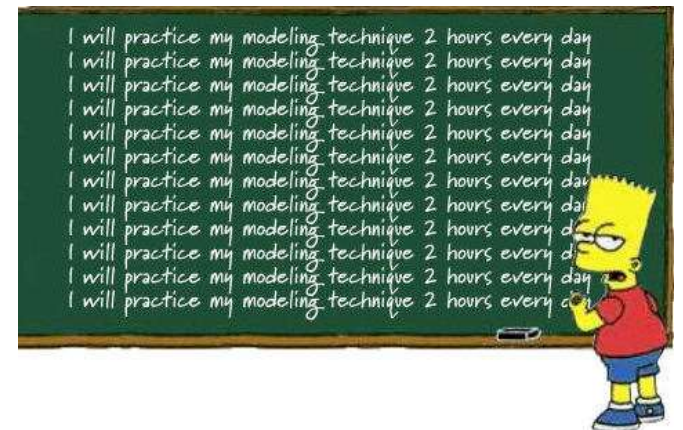
Optional : SharedModule

- Reuse components in multiple modules? Use a SharedModule
 - `ng g m shared` – shorthand notation
- Create components inside SharedModule
- Import SharedModule in other modules
- It doesn't have to be in AppModule if you don't use it directly!
- It *does* not add size to module bundles



Workshop

- Open ../100-multiple modules.
- Create a new module
- Create a new component inside this new module and give it some UI.
- Include the module in the Main Module and show it besides other modules
- Include the Search Component in your own module
- *OR:*
- Add Multiple Modules from scratch to your own application, using the steps described in this module.



How to structure feature modules



242



Why and how to structure Features in Modules in Angular

This might sound pretty basic, but I encounter these challenges over and over in customer projects and it's still an ongoing discussion internally.

A central project goal in a recent Angular project was to design features and UI components for reusability. To achieve this, we need to make sure our code is well isolated and has a simple and clear dependency model.

Prologue: Feature vs. Technical Project Structure

When building small apps and looking at common code samples in the internet a lot of devs (including myself) tend to come up with a project structure like this:

```
└─ MYAPP
  └─ src
    └─ app
      └─ components
        └─ home
          └─ home.component.html
          └─ TS home.component.ts
        └─ user
          └─ user.component.html
```

<https://medium.com/@philippbauknecht/why-and-how-to-structure-features-in-modules-in-angular-d5602c6436be>