Angular Modules (21)

Session_21_Modules(optimizations-feature-shared-module)
263 Undersanding App module
266 Creating Shared Module
270. Components selectors + vs Routing
271. Common Gotcha
Session_21_Modules(optimizations-lazy-loading)
274 Lazy Loading
Session_21_Modules(optimizations-core-module)
275 .Protecting Lazy Loaded Routes with canLoad
276. Modules and services
277. Core Module
278. Creating a Core module
Session_21_Modules(optimizations-core-preloading)
280 Ahead of time Compilation
281. Using Ahead of time compilation
282. Preload LazyLoading

263 Undersanding App module

Import is a typeScript feature, you need to tell where the data, information is stored
Declarations: Which components will be used in this module
Imports: Will import the the modules, and all of it's exported class, information
providers: Will register all of the used services, 1 global instances (valid for the ap module)
bootstrap: You have to define the root app component, where all of the other components are located

```
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

import { AuthService } from './auth/auth.service';

import { AuthGuard } from './auth/auth-guard.service';


@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
  ],
  providers: [ShoppingListService, RecipeService, DataStorageService, AuthService, AuthGuard],
  bootstrap: [AppComponent]
```

```
})
export class AppModule { }
```

265 Creating a seperate module for the Recipes Feature module

1. Creating a seperate **recipes.module.ts** module + including the only used components + the only used modules (ReactiveFormsModule)
+ including the the created modile in the app.module ts + deleting the imported components to avoid duplication.

```
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';

import { RecipesComponent } from './recipes.component';
import { RecipeStartComponent } from './recipe-start/recipe-start.component';
import { RecipeListComponent } from './recipe-list/recipe-list.component';
import { RecipeEditComponent } from './recipe-edit/recipe-edit.component';
import { RecipeDetailComponent } from './recipe-detail/recipe-detail.component';
import { RecipeItemComponent } from './recipe-list/recipe-item/recipe-item.component';
import { RecipesRoutingModule } from './recipes-routing.module';
import { SharedModule } from '../shared/shared.module';

@NgModule({
  declarations: [
    RecipesComponent,
    RecipeStartComponent,
    RecipeListComponent,
    RecipeEditComponent,
    RecipeDetailComponent,
    RecipeItemComponent
  ],
```

```
  imports: [
    CommonModule,                       //1.In every feature module, you have to add this modul
e, it will give you access to common module directives, *ngIf ..etc.
    ReactiveFormsModule,
    RecipesRoutingModule,               //2.Creating a seperate routing module for the recipes
    SharedModule
  ]
})
export class RecipesModule {}
```

2. Creating a seperate routing module

- If you are in the app.module- you can use **RouterModule.forRoot()**
- But for every other routing you have to use **RouterModule.forChild()**

recipe-routing.module.ts

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

import { AuthGuard } from '../auth/auth-guard.service';
import { RecipeEditComponent } from './recipe-edit/recipe-edit.component';
import { RecipeDetailComponent } from './recipe-detail/recipe-detail.component';
import { RecipeStartComponent } from './recipe-start/recipe-start.component';
import { RecipesComponent } from './recipes.component';

const recipesRoutes: Routes = [
  { path: 'recipes', component: RecipesComponent, children: [
    { path: '', component: RecipeStartComponent },
    { path: 'new', component: RecipeEditComponent, canActivate: [AuthGuard] },
    { path: ':id', component: RecipeDetailComponent },
    { path: ':id/edit', component: RecipeEditComponent, canActivate: [AuthGuard] },
  ] },
];
```

```
@NgModule({
  imports: [
    RouterModule.forChild(recipesRoutes)
  ],
  exports: [RouterModule]
})
export class RecipesRoutingModule {}
```

266 Creating Shared Module

1. Create a **shared.module.ts** for the modules to be shared.
(Here we are using the **DropDownDirective** as a shared one)
We are **exporting** the imported modules, so we can import this whole **SharedModule** module class

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { DropdownDirective } from './dropdown.directive';

@NgModule({
  declarations: [
    DropdownDirective
  ],
  exports: [
    CommonModule,
    DropdownDirective
  ]
})
export class SharedModule {}
```

2. Included it in the **app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
```

```
import { HttpModule } from '@angular/http';


import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { ShoppingListService } from './shopping-list/shopping-list.service';
import { AppRoutingModule } from './app-routing.module';
import { RecipeService } from './recipes/recipe.service';
import { DataStorageService } from './shared/data-storage.service';
import { AuthService } from './auth/auth.service';
import { AuthGuard } from './auth/auth-guard.service';
import { RecipesModule } from './recipes/recipes.modules';
import { SharedModule } from './shared/shared.module';
import { ShoppingListModule } from './shopping-list/shopping-list.module';
import { AuthModule } from './auth/auth.module';


@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent
  ],
  imports: [
    BrowserModule,
    HttpModule,
    AppRoutingModule,
    RecipesModule,
    SharedModule,
    ShoppingListModule,
    AuthModule
  ],
  providers: [ShoppingListService, RecipeService, DataStorageService, AuthService, AuthGuard],
  bootstrap: [AppComponent]
})
```

```
export class AppModule { }
```

3. Now you can import to the custom featured modules **recipes.module.ts**

```
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';
import { SharedModule } from '../shared/shared.module';


@NgModule({
  declarations: [    RecipeItemComponent  ],
  imports: [
    CommonModule,                       //1.In every feature module, you have to add this modul
e, it will give you access to common module directives, *ngIf ..etc.
    ReactiveFormsModule,
    RecipesRoutingModule,               //2.Creating a seperate routing module for the recipes
    SharedModule                        //3. importing shared modules
  ]
})
export class RecipesModule {}
```

270. Components selectors + vs Routing

1.When you have to use the componenet selector like **<app-component>** you have to declare it in the app.module.ts in order to let angular know about where and how it was declared.
1.5 Or when you are using shared module, the modules are defined and exported as well to the app.module.ts so they are ready to use.
2. When you are using a component for routing then the only important thing is to **import in the routing module**, **or import it somewhere before using the routing.**

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


import { ShoppingListComponent } from './shopping-list/shopping-list.component';
```

```
const appRoutes: Routes = [
  { path: '', redirectTo: '/recipes', pathMatch: 'full' },
  { path: 'shopping-list', component: ShoppingListComponent }
];

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

}
```

271. Common Gotcha

When creating modules you have to check the following
- Check all of the **component's** used by the module
- Include the **commonModule** or **any** other **required modules**, httpModules or anything.
- Create the **routingModules,** include them if necessary
- Check if other component's are using a shared module if yes, create one and export it in the sahred module and import this in the app.module.ts
- CHeck out the dependencies.

274 Lazy Loading

1. Create a home page, so we can make the other component's loading optional.
2. Remove the optional module from app.module.ts since, if we leave it there webpack will gather the data prior first load anyway.

3. Use at the routing file `loadchildren` **('filePath#ExportedClassName')** . IF we include a component, it will gather the data, but if we include a string, it will only download the data once the router is activated!
app-routing.module.ts

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

import { ShoppingListComponent } from './shopping-list/shopping-list.component';
import { HomeComponent } from './home/home.component';
```

```
const appRoutes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'recipes', loadChildren: './recipes/recipes.module#RecipesModule'},
  { path: 'shopping-list', component: ShoppingListComponent }
];

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

}
```

4. Recalculate the outsourced **recipes.routing.module.ts** since we alredy redirecting at / recipes to the recipeComponenet.
Instead of **'/recipes'**, only **'/'** has to be included

```
const recipesRoutes: Routes = [
  { path: '', component: RecipesComponent, children: [
    { path: '', component: RecipeStartComponent },
    { path: 'new', component: RecipeEditComponent, canActivate: [AuthGuard] },
    { path: ':id', component: RecipeDetailComponent },
    { path: ':id/edit', component: RecipeEditComponent, canActivate: [AuthGuard] },
  ] },
];
```

275 .Protecting Lazy Loaded Routes with canLoad

What if you want to use route protection (canActivate to be precise) on lazily loaded routes?

You can add canActivate to the lazy loaded routes but that of course means, that you might load code which in the end can't get accessed anyways. It would be better to check that BEFORE loading the code.

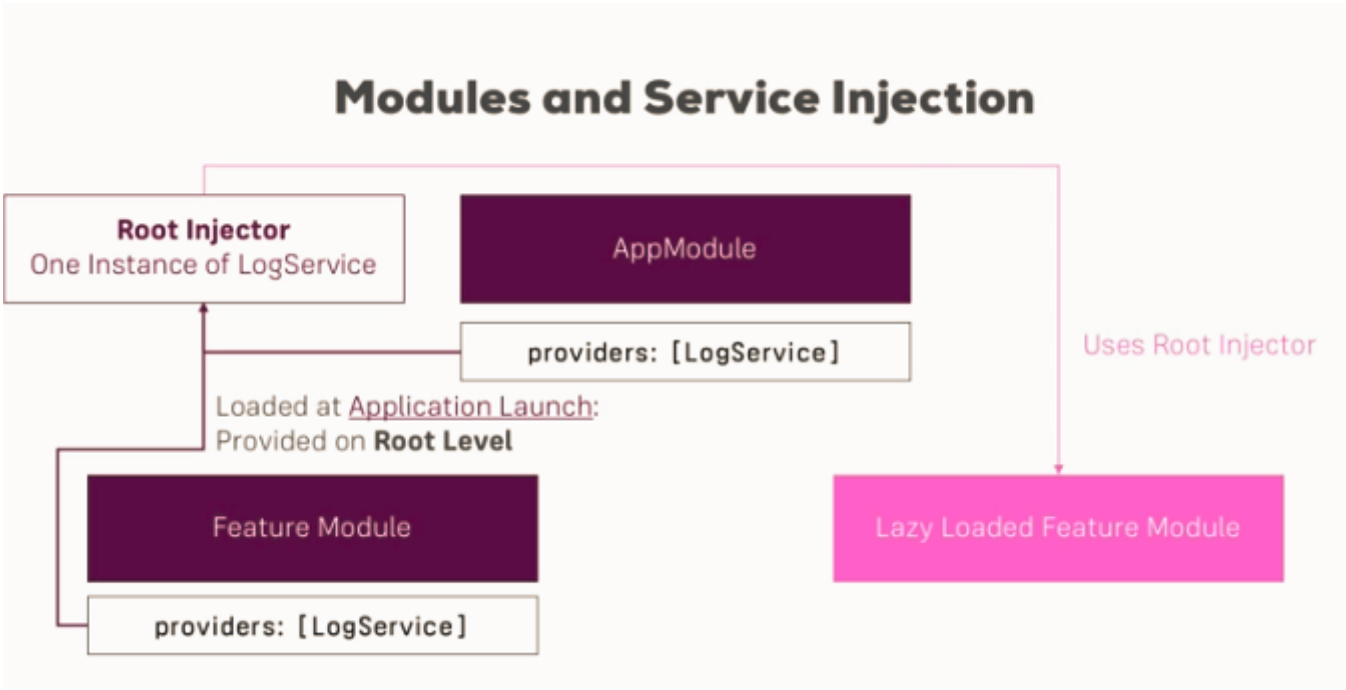
You can enforce this behavior by adding the canLoad guard to the route which points to the lazily loaded module:

```
{ path: 'recipes', loadChildren: './recipes/recipes.module#RecipesModule', canLoad: [AuthGuard]
 }
```

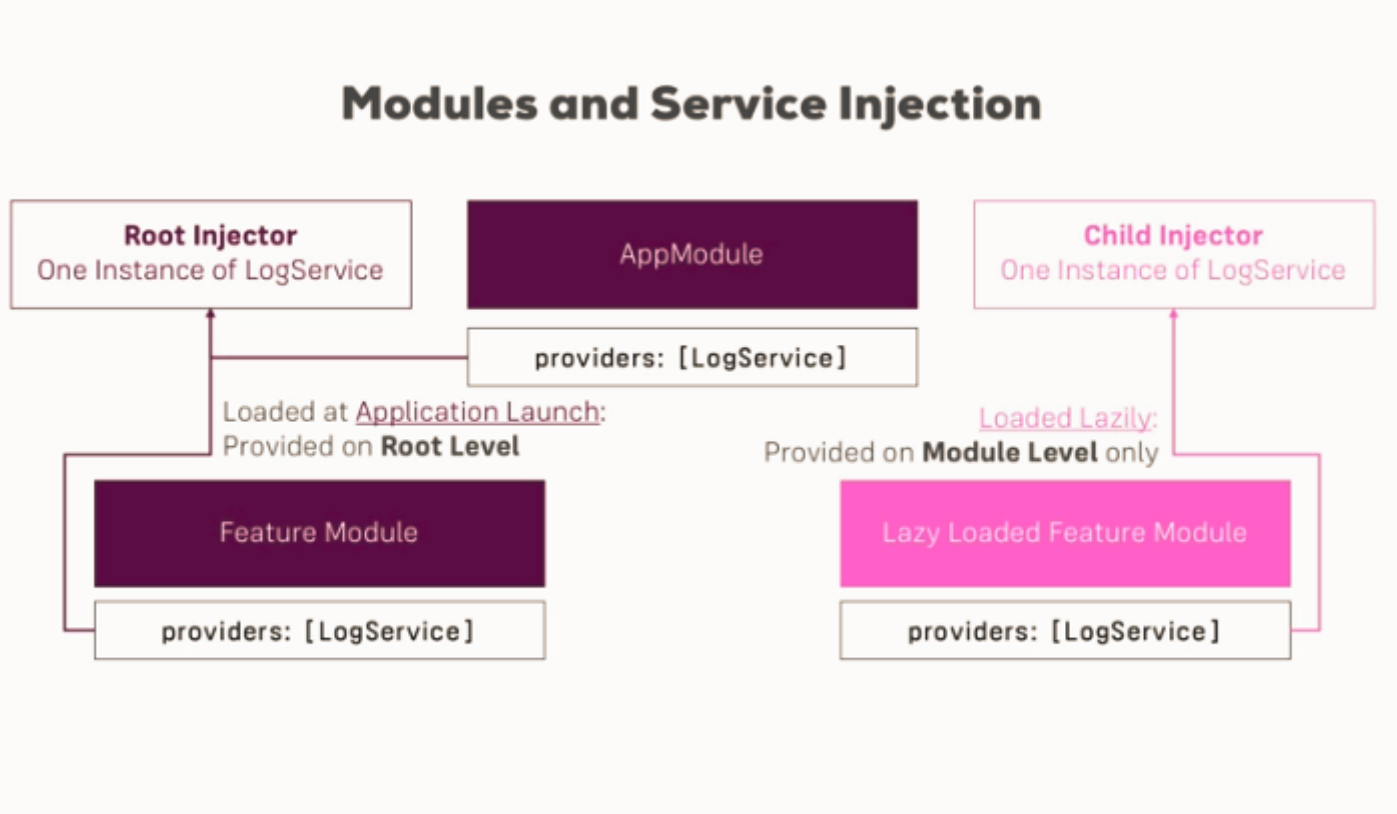
In this example, the AuthGuard should implement the `CanLoad` interface.

276. Modules and services

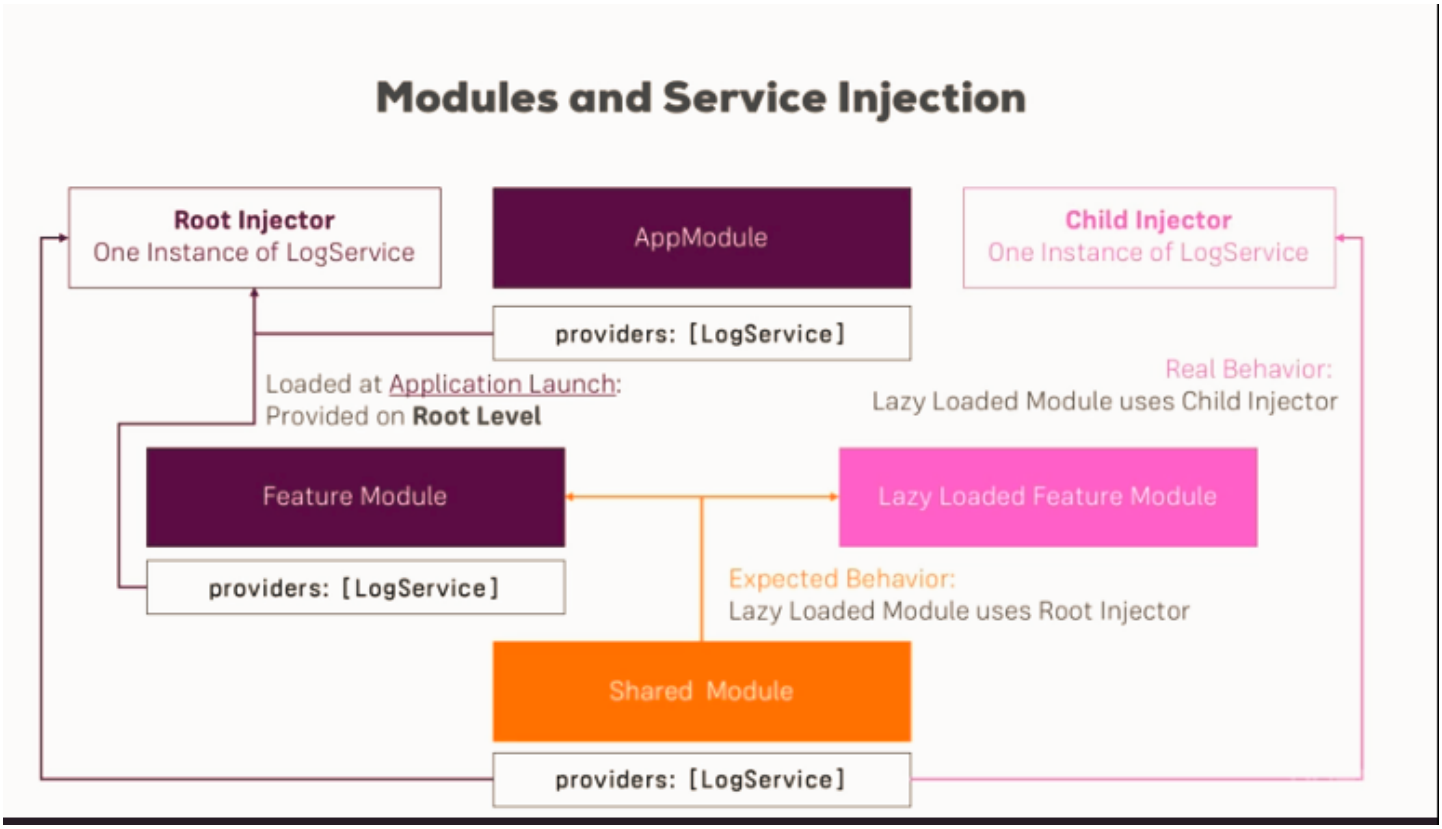
1. IF you provide the service in the **app.module.ts** at the **Root** you will get 1 instance in every component, including the lazyloaded ones.



2. If you provide it at the modular level, in the component's provider:, you will have hierarchical inheritance of the service instance, so the appmodule and all of the components under it will have the same instance, but the seperately lazy lodade component will have a different instance.

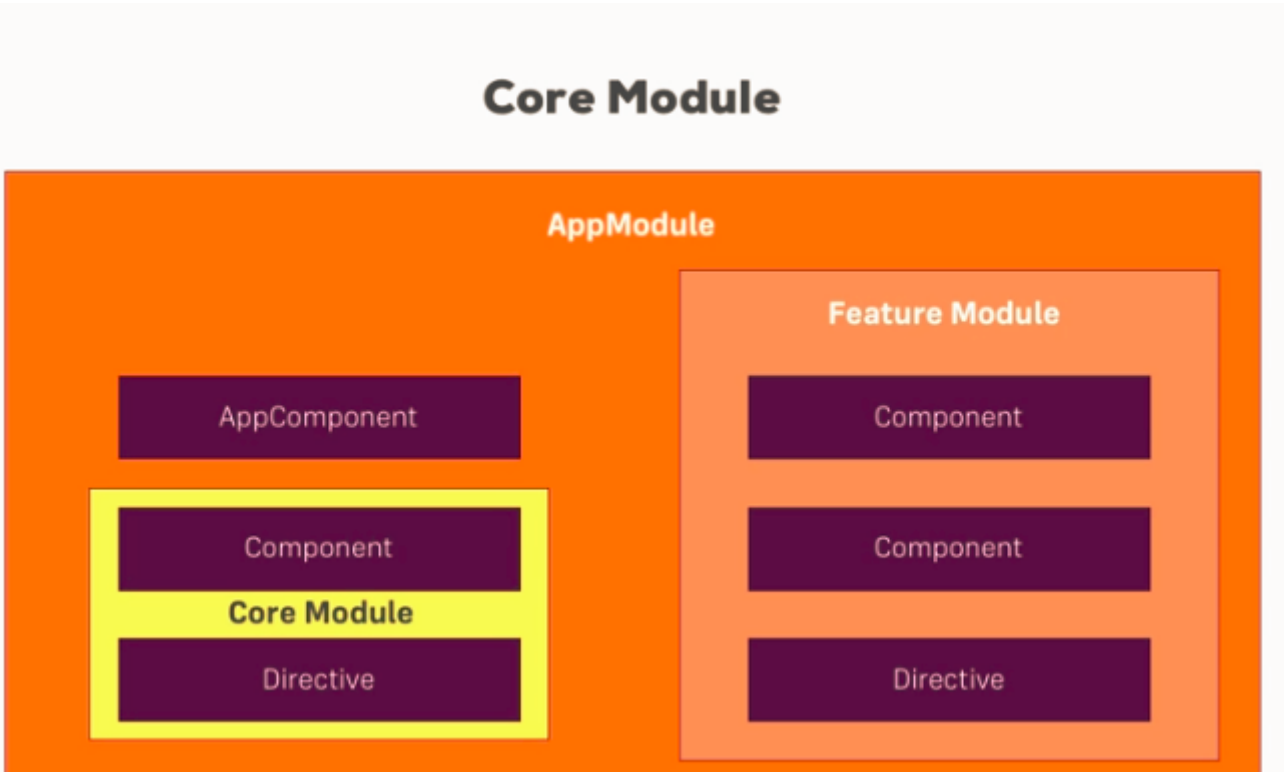


3. Definitely avoid! is to provide services in the Shared Modules! SInce it will create different instances, but you have less understanding about what is going on.



277. Core Module

Everything component, directive, service which is essential for the app. should be inlcuded in the **core module.**



278. Creating a Core module

We imports all of the required modules, component's to make the core module funcitonal, and also since we provide the CoreModule early in the app.module.ts we can **provide here the services** as well.

```
import { NgModule } from '@angular/core';


import { HeaderComponent } from './header/header.component';
import { HomeComponent } from './home/home.component';
import { SharedModule } from '../shared/shared.module';
import { AppRoutingModule } from '../app-routing.module';
import { AuthService } from '../auth/auth.service';
import { DataStorageService } from '../shared/data-storage.service';
import { RecipeService } from '../recipes/recipe.service';
import { ShoppingListService } from '../shopping-list/shopping-list.service';


@NgModule({
  declarations: [                    //These are the used components
    HeaderComponent,
    HomeComponent
```

```
  ],
  imports: [
    SharedModule,                       //Using the dropdown directve
    AppRoutingModule                    //Need to have the routing functionality
  ],
  exports: [
    AppRoutingModule,
    HeaderComponent                  // We are using the <header> selector
  ],
  //Since we have loaded eagerly the coremodule in the app.module.ts we can include the service
s and have 1 instance
  providers: [
    ShoppingListService,
    RecipeService,
    DataStorageService,
    AuthService
  ]
})
export class CoreModule {}
```

Eearly included core module in the app.module to provide 1 instance of the services

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';
import { SharedModule } from './shared/shared.module';
import { ShoppingListModule } from './shopping-list/shopping-list.module';
import { AuthModule } from './auth/auth.module'
import { CoreModule } from './core/core.module';
```

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpModule,
    AppRoutingModule,
    SharedModule,
    ShoppingListModule,
    AuthModule,
    CoreModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

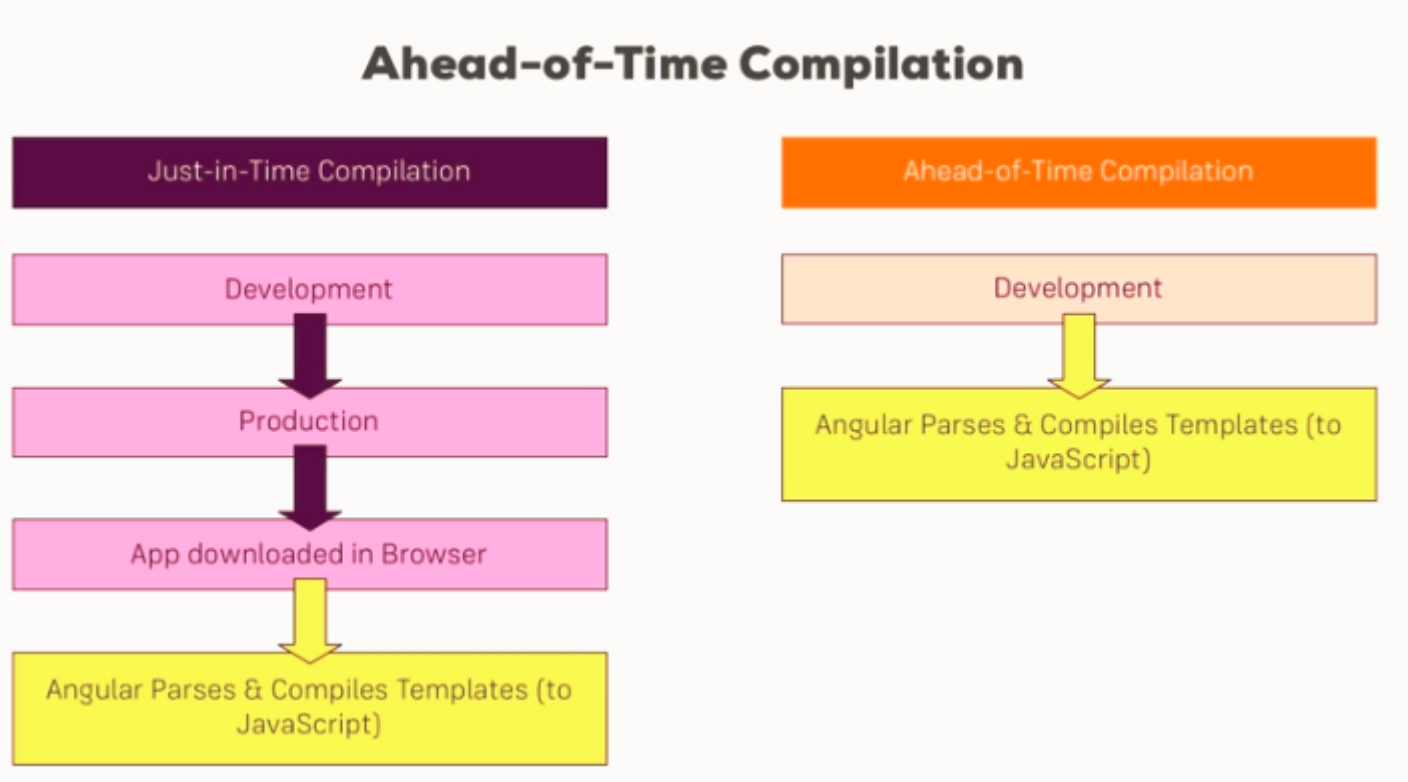
3. Only `AuthGuard` services can be provided into the routing modules, since they are the only ones who uses them. I**t can make the performance better, if we load it lazy.**
recipes-routing.module.ts

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

import { AuthGuard } from '../auth/auth-guard.service';
import { RecipeEditComponent } from './recipe-edit/recipe-edit.component';
import { RecipeDetailComponent } from './recipe-detail/recipe-detail.component';
import { RecipeStartComponent } from './recipe-start/recipe-start.component';
import { RecipesComponent } from './recipes.component';

const recipesRoutes: Routes = [
  { path: '', component: RecipesComponent, children: [
```

```
    { path: '', component: RecipeStartComponent },
    { path: 'new', component: RecipeEditComponent, canActivate: [AuthGuard] },
    { path: ':id', component: RecipeDetailComponent },
    { path: ':id/edit', component: RecipeEditComponent, canActivate: [AuthGuard] },
  ] },
];


@NgModule({
  imports: [
    RouterModule.forChild(recipesRoutes)
  ],
  exports: [RouterModule],
  providers: [
    AuthGuard
  ]
})
export class RecipesRoutingModule {}
```

280 Ahead of time Compilation

Angular compiles your html templates to javascript, since it will load faster.



Benefits

- Faster startup, since the compiling won't happen in the browser.
- Templates get checked beforehead in the developement.
- Smaller file size, since the angular compiler do not have to be included in the downloader package.

281. Using Ahead of time compilation

Type in the CLI the following
ng build --prod --aot
and it will minify your codes to be smaller in a folder called **dist, (Vendor file** is the angular compiler, which will be reduced significantly**)**

282. Preload LazyLoading

At the **app-routing.module.ts** for the forRoot include the **PreloadAllModules** strategy

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


import { ShoppingListComponent } from './shopping-list/shopping-list.component';
import { HomeComponent } from './core/home/home.component';
```

```
const appRoutes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'recipes', loadChildren: './recipes/recipes.module#RecipesModule'},
  { path: 'shopping-list', component: ShoppingListComponent }
];


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


}
```