Angular Routes

Miti Bhat

Routes

- Routes are definitions (objects) comprised from at least a path and a component (or a redirectTo path) attributes. The path refers to the part of the URL that determines a unique view that should be displayed, and component refers to the Angular component that needs to be associated with a path.

Route Definition

- Based on a route definition that we provide (via a static RouterModule.forRoot(routes) method), the Router is able to navigate the user to a specific view.
- Each Route maps a URL path to a component.
- The path can be empty which denotes the default path of an application and it’s usually the start of the application.

**

- The path can take a wildcard string (**).
- The router will select this route if the requested URL doesn’t match any paths for the defined routes. This can be used for displaying a “Not Found” view or redirecting to a specific view if no match is found.

A simple path

- { path: 'contacts', component: ContactListComponent}
- If this route definition is provided to the Router configuration, the router will render ContactListComponent when the browser URL for the web application becomes /contacts.

Prefix & Full

```
{ path:  'contacts', component:  ContactListComponent}
```

Could be also written as:

```
{ path:  'contacts',pathMatch: 'prefix', component:  ContactListComponent}
```

The pathMatch attribute specifies the matching strategy. In this case, it’s prefix which is the default.

The second matching strategy is full. When it’s specified for a route, the router will check if the the path is exactly equal to the path of the current browser’s URL:

```
{ path:  'contacts',pathMatch: 'full', component:  ContactListComponent}
```

ROUTE PARAMS

Creating routes with parameters is a common feature in web apps. Angular Router allows you to access parameters in different ways:

- Using the ActivatedRoute service,
- Using the ParamMap observable available starting with v4.

Route Parameters

You can create a route parameter using the colon syntax. This is an example route with an id parameter:

```
{ path:  'contacts/:id', component:  ContactDetailComponent}
```

ROUTE GUARDS

A route guard is a feature of the Angular Router that allows developers to run some logic when a route is requested, and based on that logic, it allows or denies the user access to the route.

It’s commonly used to check if a user is logged in and has the authorization before he can access a page.

Route Guards

You can add a route guard by implementing the CanActivate interface available from the @angular/router package and extends the canActivate() method which holds the logic to allow or deny access to the route. For example, the following guard will always allow access to a route:

CanActivate

```
class MyGuard implements CanActivate {
  canActivate() {
 if(role==‘admin’)
 return true;
Else{
  return false;
  }
}

{ path:  'contacts/:id, canActivate:[MyGuard], component:  ContactDetailComponent}
```

NAVIGATION DIRECTIVE

```
<a [routerLink]="'/contacts'">Contacts</a>
```

MULTIPLE OUTLETS AND AUXILIARY ROUTES

Angular Router supports multiple outlets in the same application.

A component has one associated primary route and can have auxiliary routes. Auxiliary routes enable developers to navigate multiple routes at the same time.

To create an auxiliary route, you'll need a named router outlet where the component associated with the auxiliary route will be displayed.

```
<router-outlet></router-outlet>
```

```
<router-outlet  name="outlet1"></router-outlet>
```

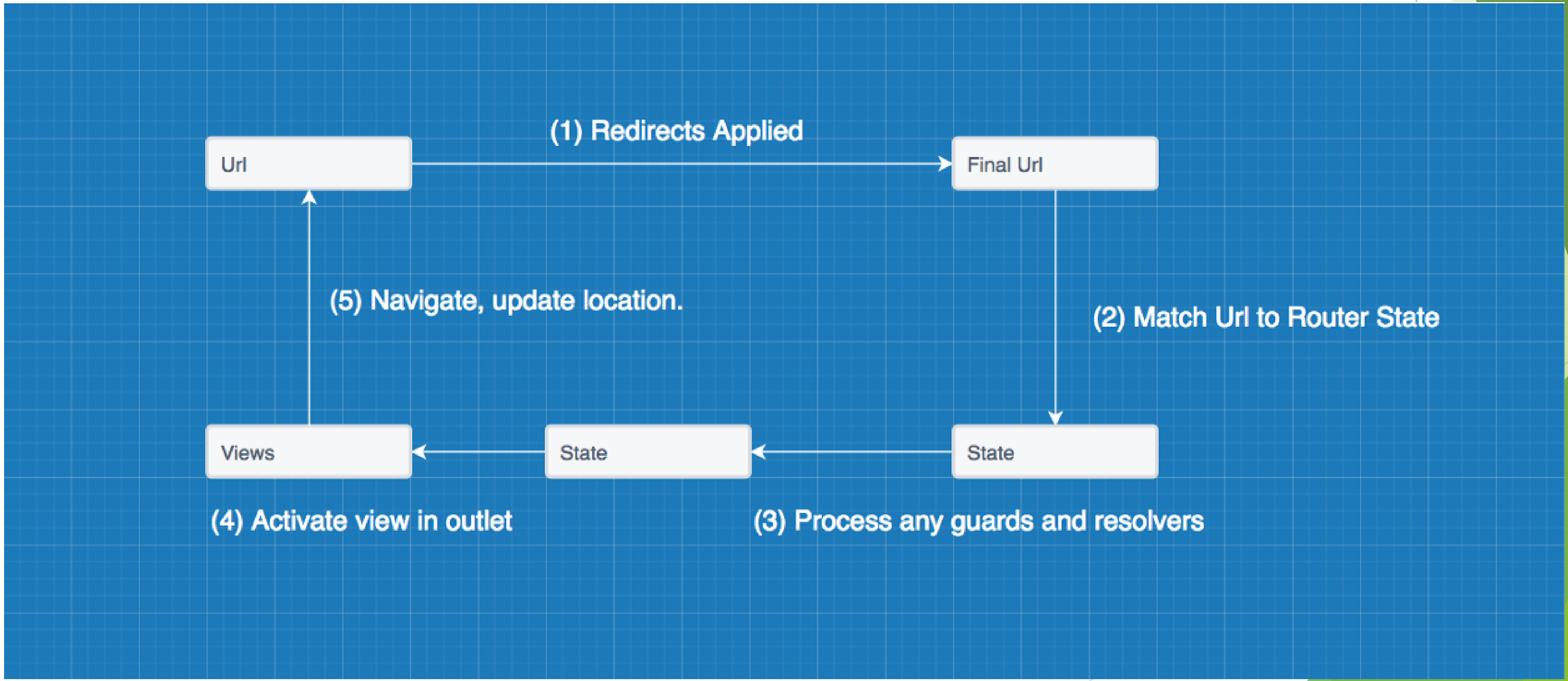
- The outlet with no name is the primary outlet.
- All outlets should have a name except for the primary outlet.

Router outlets

You can then specify the outlet where you want to render your component using the outlet attribute:

```
{ path: "contacts", component: ContactListComponent, outlet: "outlet1" }
```

Routing


Url
(1) Redirects Applied
Final Url
(5) Navigate, update location.
(2) Match Url to Router State
Views
State
State
(4) Activate view in outlet
(3) Process any guards and resolvers

Navigation Cycle

During this navigation cycle, the router emits a series of events. The Router service provides an observable for listening to router events, which can be used to define logic, such as running a loading animation, as well as aiding in debugging routing. Some noteworthy events during this cycle are:

NavigationStart: Represents the start of a navigation cycle.

NavigationCancel: For instance, a guard refuses to navigate to a route.
RoutesRecognized: When a url has been matched to a route.

NavigationEnd: Triggered when navigation ends successfully.

More Router Events

https://github.com/angular/angular/blob/master/packages/router/src/events.ts?source=post_page---------------------------#L45

Routing Events

Given the above configuration, let’s consider what happens when given the url http://localhost:4200/notes/42.The overview is as follows.

- First, any redirects must be processed, since there is no sense in trying to match a url to a router state until we have a finalized version of that url. Since there are no redirects in this case, the url stays as is and is unchanged.
- Next, the router uses a first-match-wins-with-backtracking strategy to match the url to a router state defined in the configuration. In this case, it will match path: ‘notes', and then path:’:id'. The NoteComponent is associated with this route.
- Since a matching router state was found, the router then checks if there are any guards associated with that router state, which might prevent navigation. For instance, maybe only users who are logged in can view notes. In this example, there are no guards. We’re not using any resolvers to prefetch data for this route either, so the router proceeds with the navigation.
- The router then activates the component associated with this route state.
- The router finishes navigation. Then it waits for another change to the router state/url, and repeats the process all over again.

Lazy Loading Feature Modules

- As an application grows over time, more and more of its functionality will be encapsulated in separate feature modules.
- **Chances are, not all of this data will be displayed when the application first loads, so there is no reason to include all of it in the main bundle**.
- It will only bloat that file, and cause longer download times when loading the application. It is better to load these modules on demand whenever a user navigates to them, and it is through lazy loading that the Angular router achieves this.

```
// from the Angular docs https://angular.io/guide/lazy-loading-ngmodules#routes-at-the-app-level

{

  path: 'customers',

  loadChildren: 'app/customers/customers.module#CustomersModule'

}
```

The router will start fetching any lazily loaded modules during the apply redirects / url matching phase of the navigation cycle:

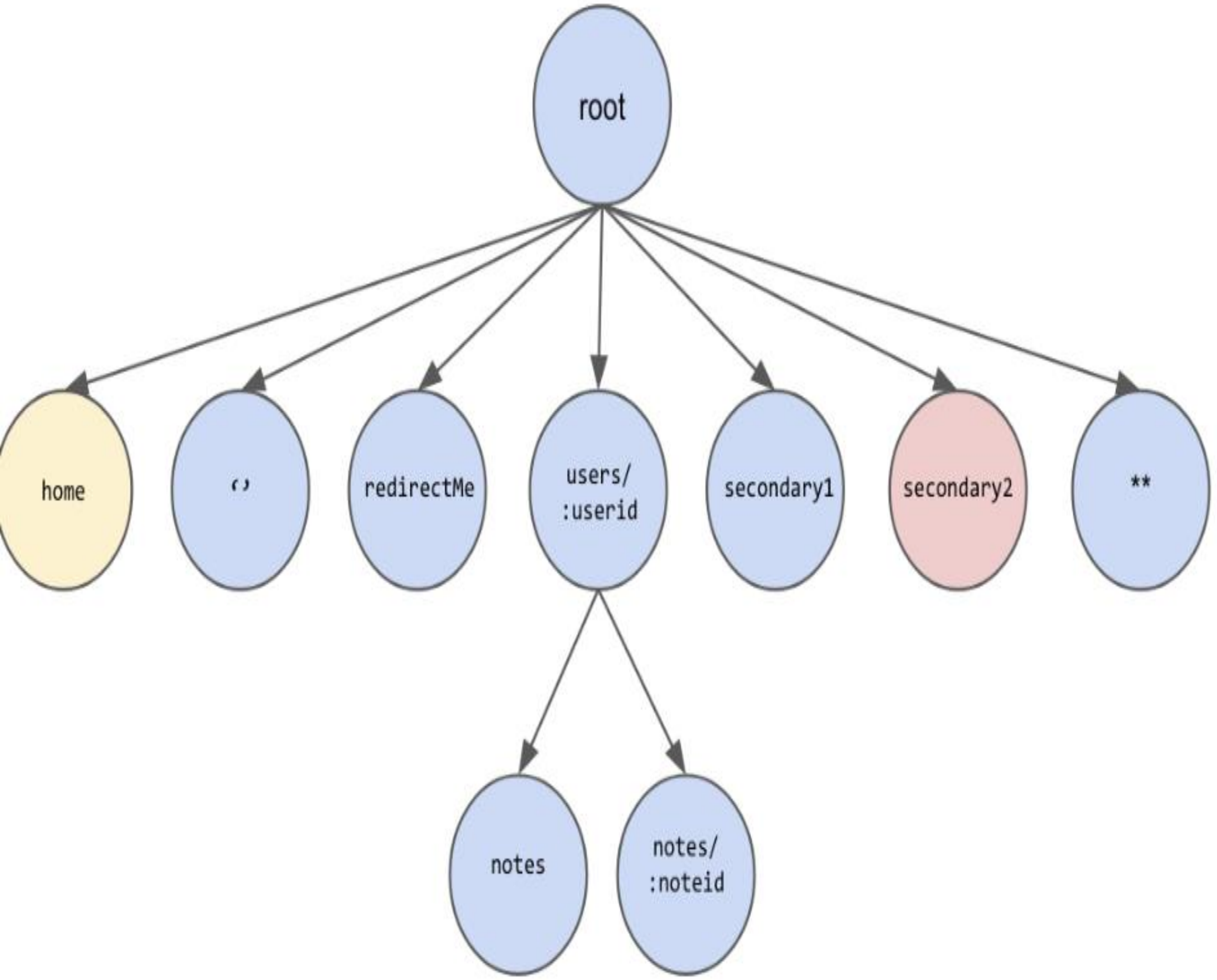
The router will use registered NgModuleFactoryLoader to fetch an NgModule associated with the loadChildren string. Then it will extract the set of routes defined in that NgModule, and will transparently add those routes to the main configuration.

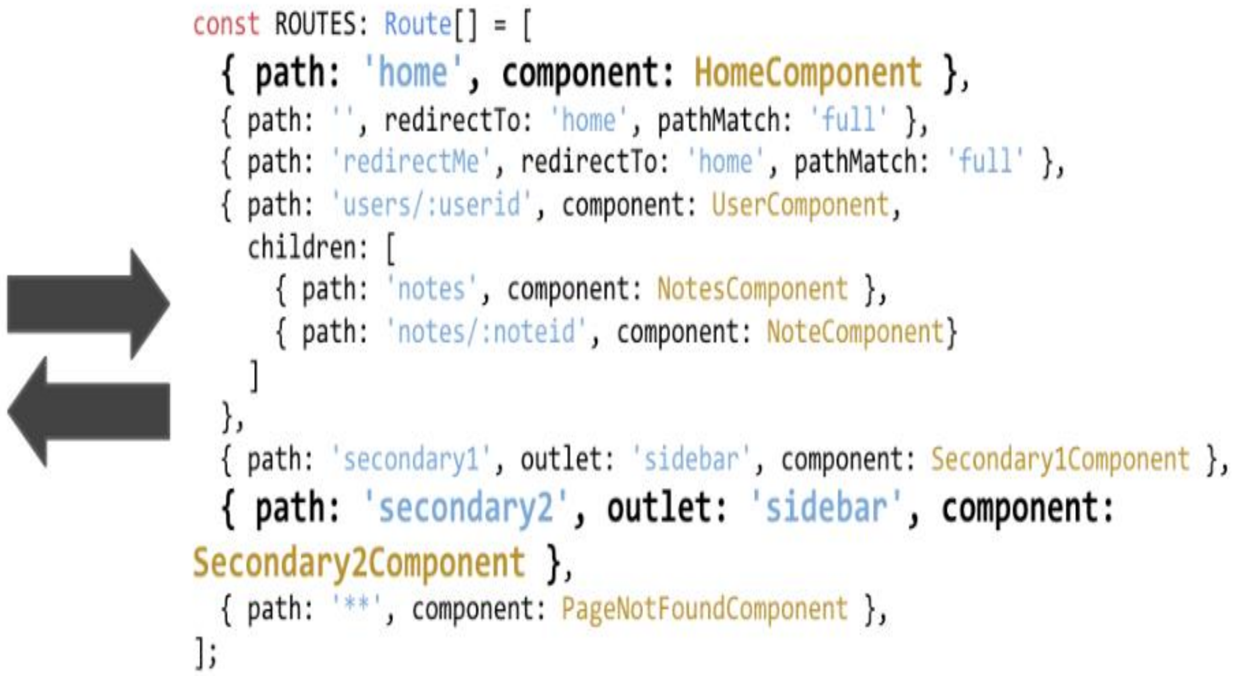
So the routes defined in the lazily loaded module's configuration will be loaded into the main configuration, and can then be matched against and routed to.

A Tree Of States

The router views the routable portions of an application as a tree of *router states*

```
const ROUTES: Route[] = [
  { path: 'home', component: HomeComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'redirectMe', redirectTo: 'home', pathMatch: 'full' },
  { path: 'users/:userid', component: UserComponent,
    children: [
      { path: 'notes', component: NotesComponent },
      { path: 'notes/:noteid', component: NoteComponent}
    ]
  },
  { path: 'secondary1', outlet: 'sidebar', component: Secondary1Component },
  { path: 'secondary2', outlet: 'sidebar', component: Secondary2Component },
  { path: '**', component: PageNotFoundComponent },
];
```



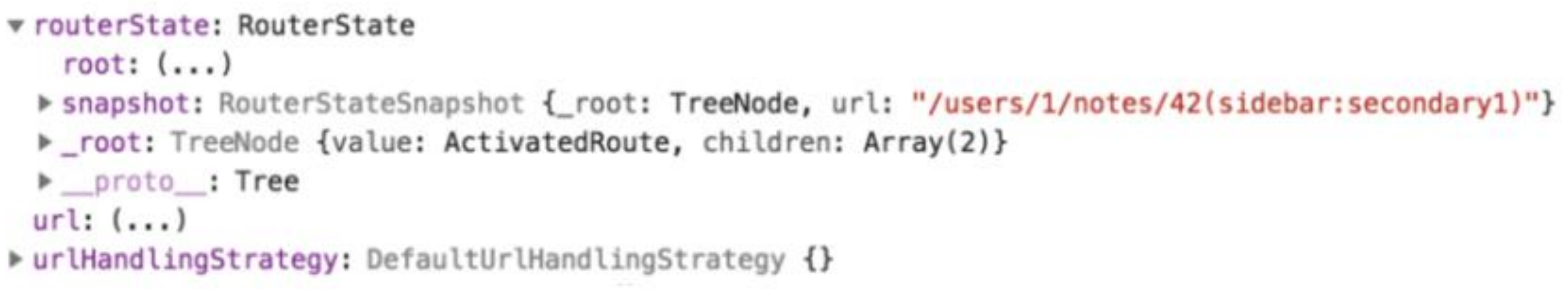


```
const ROUTES: Route[] = [
  { path: 'home', component: HomeComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'redirectMe', redirectTo: 'home', pathMatch: 'full' },
  { path: 'users/:userid', component: UserComponent,
    children: [
      { path: 'notes', component: NotesComponent },
      { path: 'notes/:noteid', component: NoteComponent}
    ]
  },
  { path: 'secondary1', outlet: 'sidebar', component: Secondary1Component },
  { path: 'secondary2', outlet: 'sidebar', component:
Secondary2Component },
  { path: '**', component: PageNotFoundComponent },
];
```

Redirects

- Redirects can happen at each level of nesting in the router config tree, but can happen only once per level.
- **This is to avoid any infinite redirect loops.**
- **Refresh any component:**

```
this.router.navigateByUrl('/RefreshComponent', { skipLocationChange: true
}).then(() => {
    this.router.navigate(['Your actualComponent']);
});
```

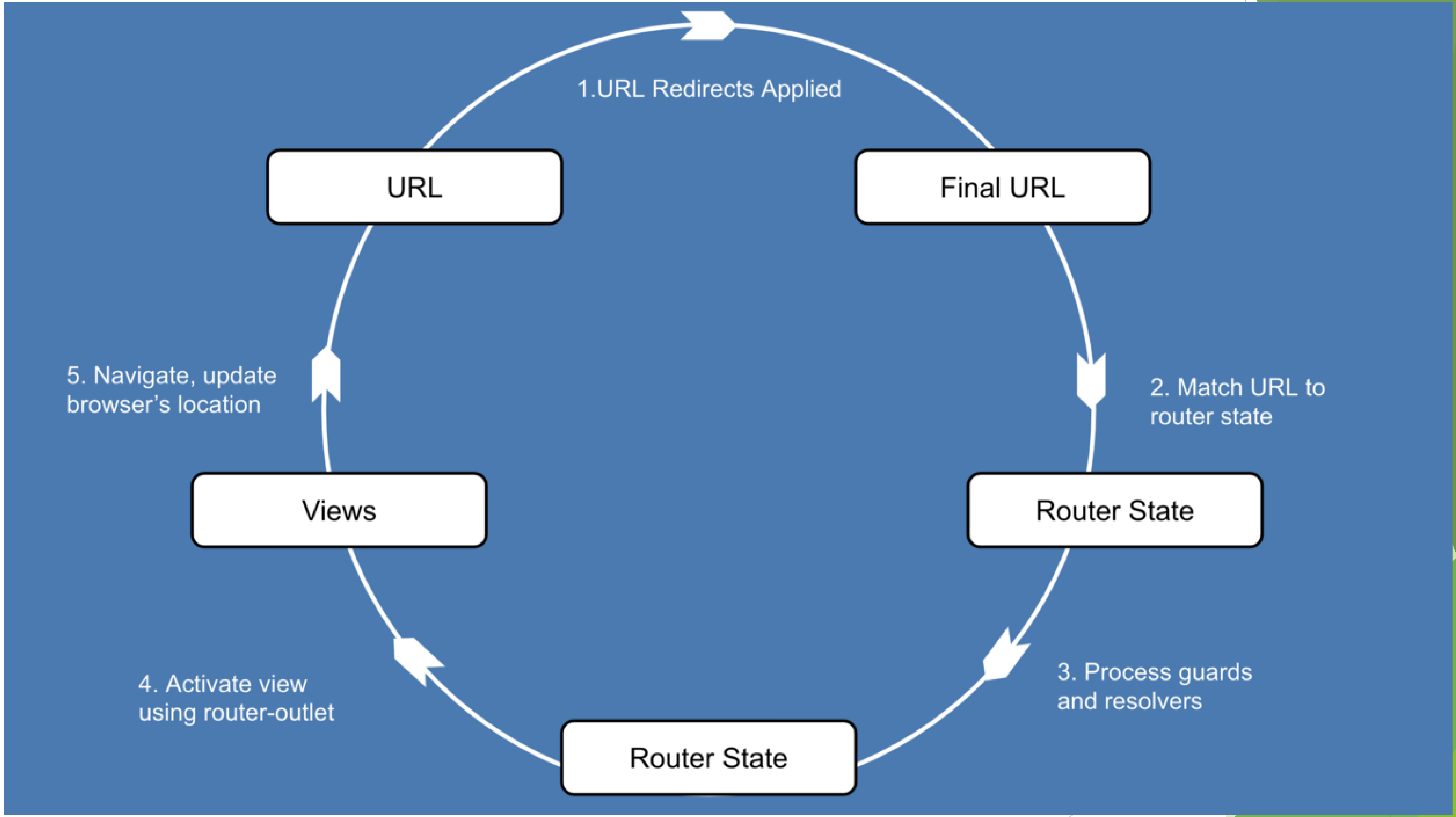


```
▼ routerState: RouterState
    root: (...)
  ▶ snapshot: RouterStateSnapshot {_root: TreeNode, url: "/users/1/notes/42(sidebar:secondary1)"}
  ▶ _root: TreeNode {value: ActivatedRoute, children: Array(2)}
  ▶ __proto__: Tree
  url: (...)
▶ urlHandlingStrategy: DefaultUrlHandlingStrategy {}
```

A snippet of the Router service

Both are trees representing the current router state (components that have been routed to, along with URL segments and parameters), but they differ in one key way:
Snapshot is a tree of ActivatedRouteSnapshot objects — which are static
Root is a tree of ActivatedRoute objects — which are dynamic.

Note that a routerState can have multiple trees of ActivatedRoutes at once — one for each outlet.


1.URL Redirects Applied
URL
Final URL
2. Match URL to router state
Router State
3. Process guards and resolvers
Router State
4. Activate view using router-outlet
Views
5. Navigate, update browser's location

Navigation

- In Angular, we build single page applications. This means that we don’t actually load a new page from a server whenever the URL changes. Instead, the router provides location-based navigation in the browser, which is essential for single page applications. It is what allows us to change the content the user sees, as well as the URL, all without refreshing the page.
- Navigation (routing) occurs whenever the URL changes. We need a way to navigate between views in our application, and standard anchor tags with href will not work, since those would trigger full page reloads. This is why Angular provides us with the[routerLink] directive. When clicked, it tells the router to update the URL and render content using <router-outlet> directives, without reloading the page.

Reading parameters from route

```
constructor(router: Router) {

  route.params.subscribe(
      params =>{
          this.courseId = parseInt(params['id']);
      }
   );
}
```

Using Child Routes for implementing Master Detail

```
{
    path: 'courses',
    children: [
      {
        path: ':id',
children: [
          {
            path: '',
            component: CourseLessons,
          },
```

```
{
  path: 'videos/:id',
          component: VideoLesson
        },
        {
          path: 'textlectures/:id',
          component: TextLesson
        },
        {
          path: 'quizzes/:id',
          component: QuizLesson
        },
```

```
{
              path:
'interactivelessons/:id',
              component:
InteractiveLesson
            }
          ]
        }
      ]
    }
];
```

```
export const routeConfig:Routes = [
    {
        path: 'courses',
        children: [
            {
                path: ':id',
                children: [
                    {
                        path: '',
                        component:
CourseLessons,
                    },
                    {
                        path: 'videos/:id',
                        component:
VideoLesson
                    },
```

```
                    {
                        path: 'textlectures/:id',
                        component: TextLesson
                    },
                    {
                        path: 'quizzes/:id',
                        component: QuizLesson
                    },
```

```
export const routeConfig:Routes = [
    {
        path: 'courses',
        ....
    },
    {
        path: 'playlist',
        outlet: 'aside',
        component: Playlist
    }
];
```

What does the URL look like for accessing an auxiliary route?

The Angular Router introduces a special syntax to allow to define auxiliary route URLs in the same URL as the primary route URL. Let's say that we want to trigger navigation and show AllLessons in the primary outlet and Playlist in the rightAside outlet. The URL would look like this:

```
/lessons(aside:playlist)
```

Multiple Auxiliary Routes

Note that you could have multiple auxiliary routes inside parenthesis, separated by //. for example this would define the url for a left-menu outlet:

`/lessons(aside:playlist//leftmenu:/some/path)`

Thank You