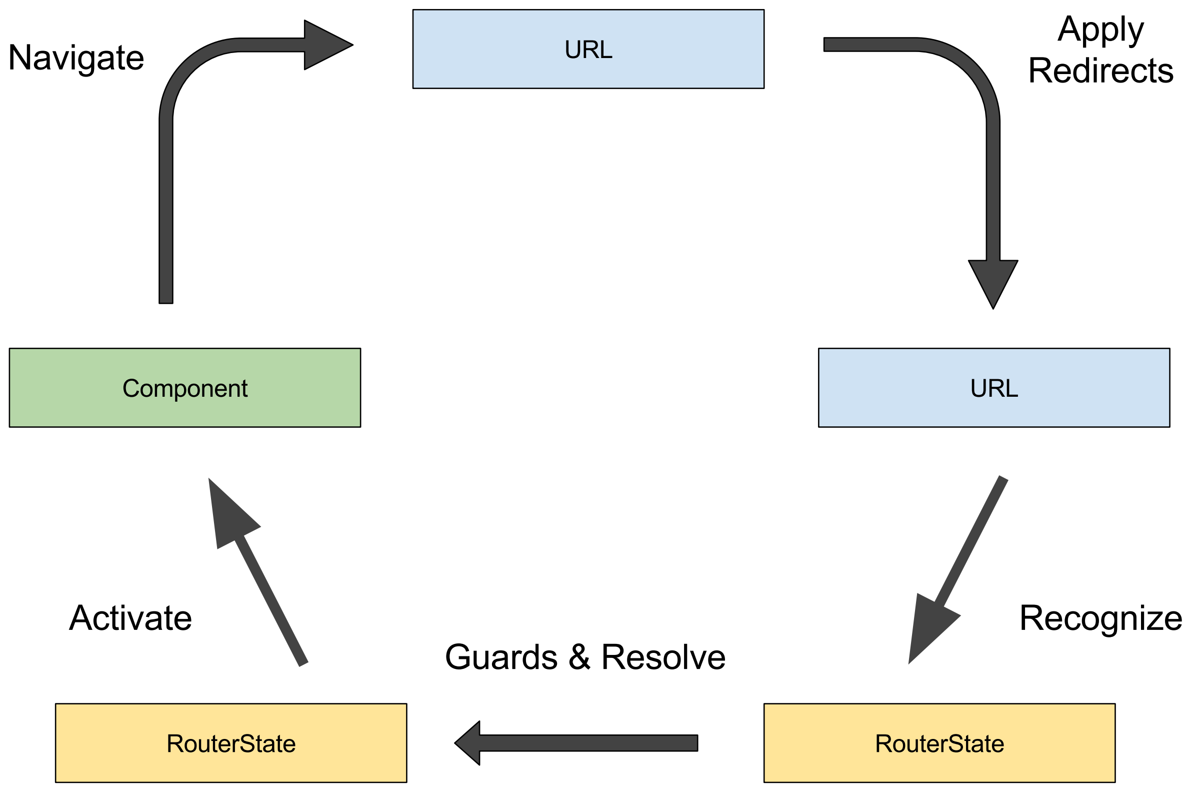
Angular Master Class

Routing

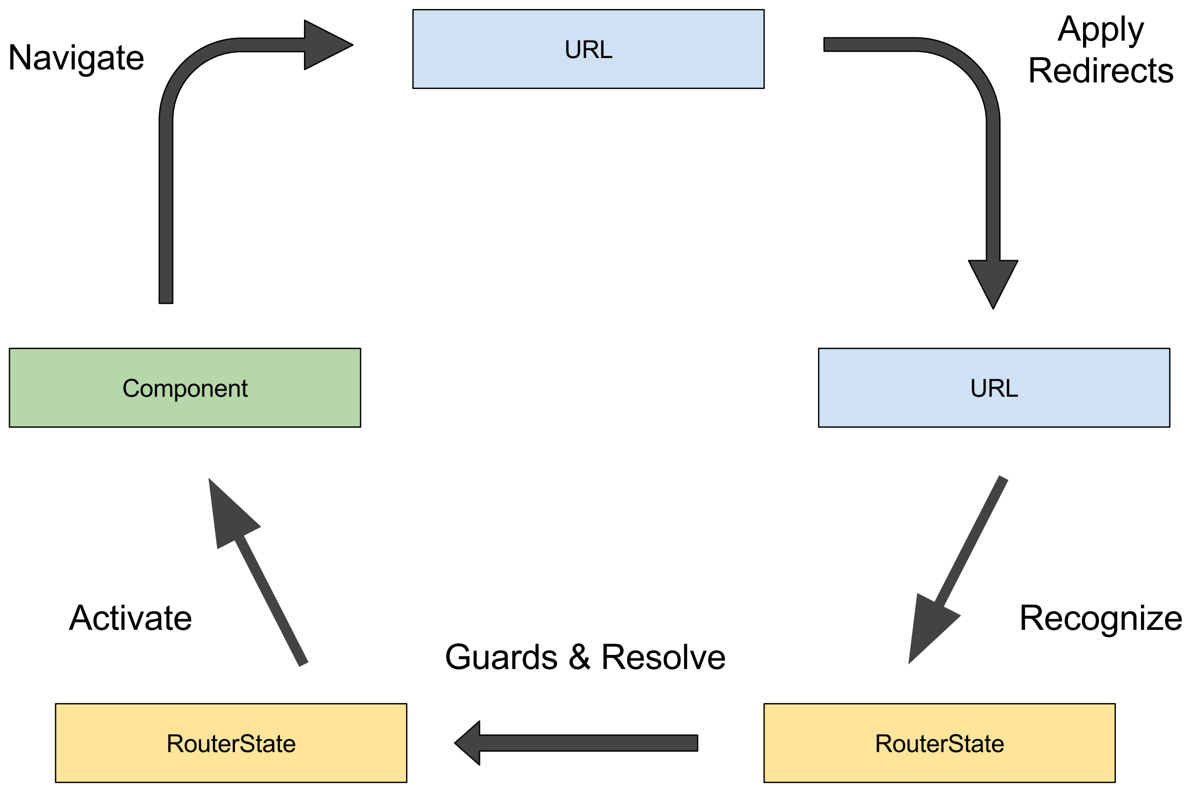
Understanding the Router

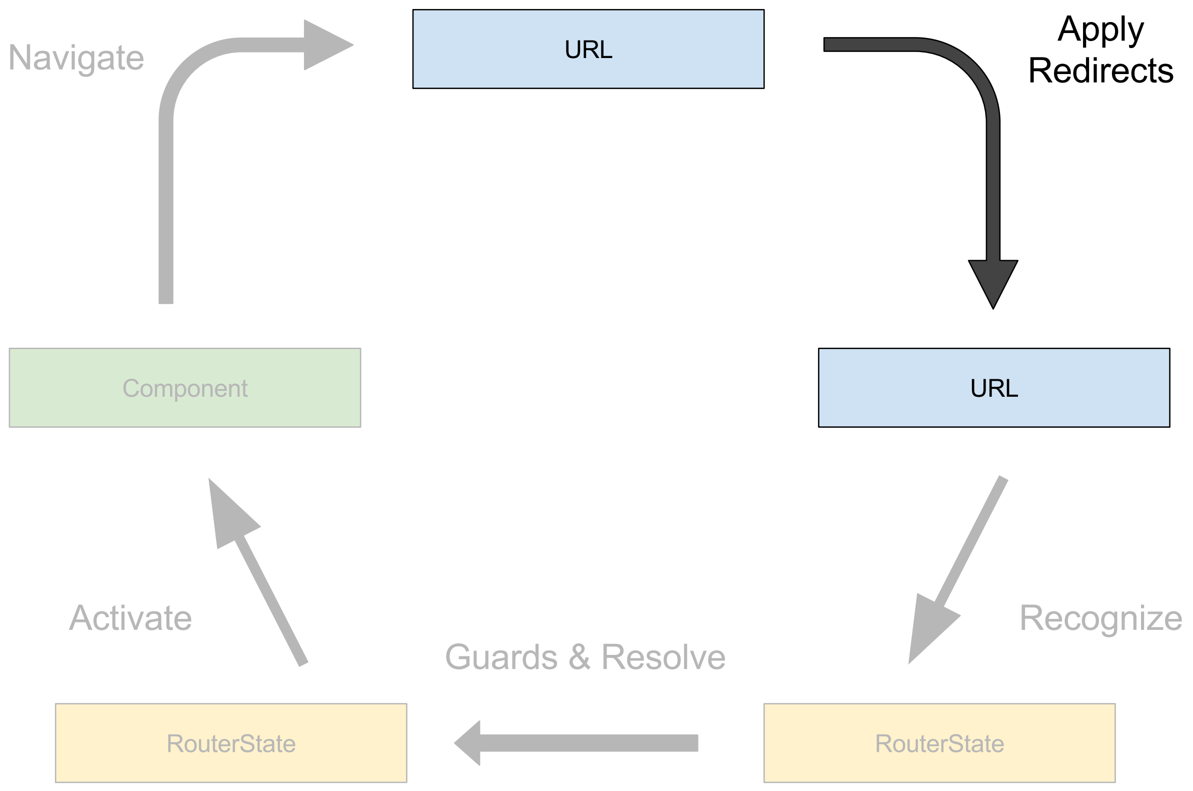

Navigate
URL
Apply Redirects
Component
URL
Activate
Recognize
Guards & Resolve
RouterState
RouterState

Router Lifecycle

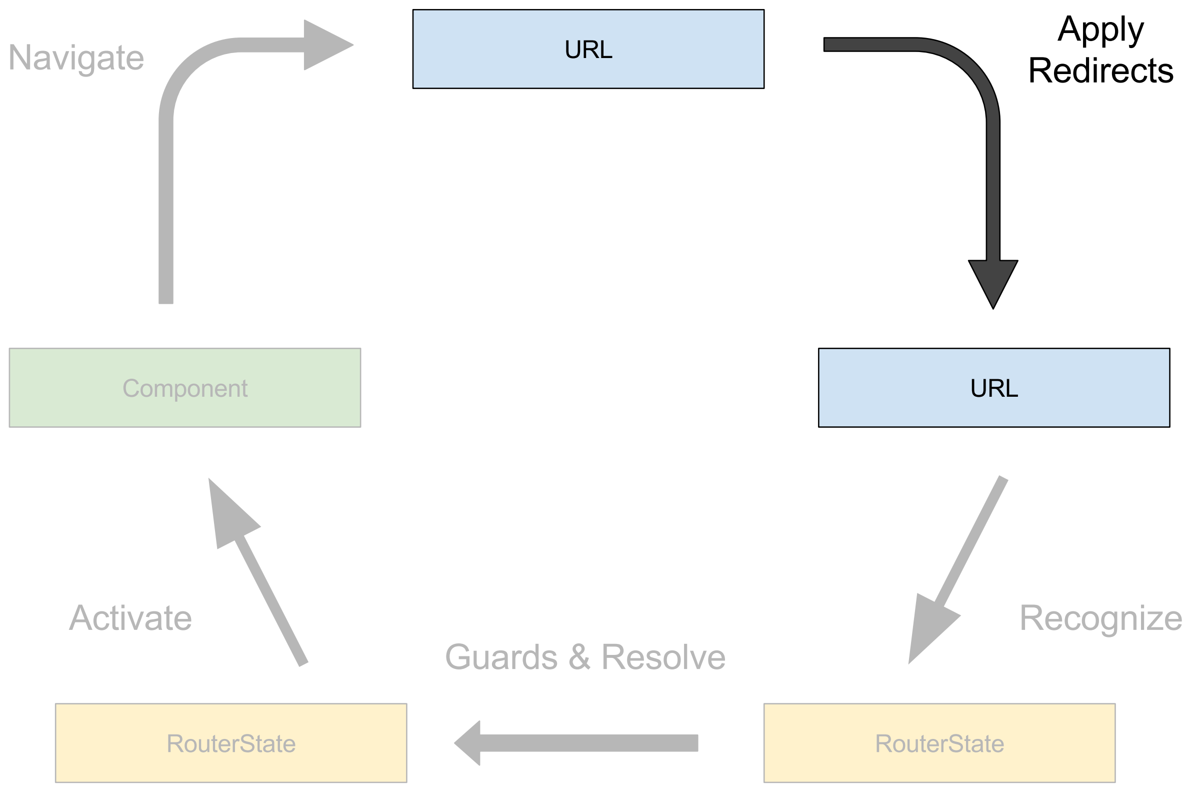
On a high level, the router takes a URL, then:

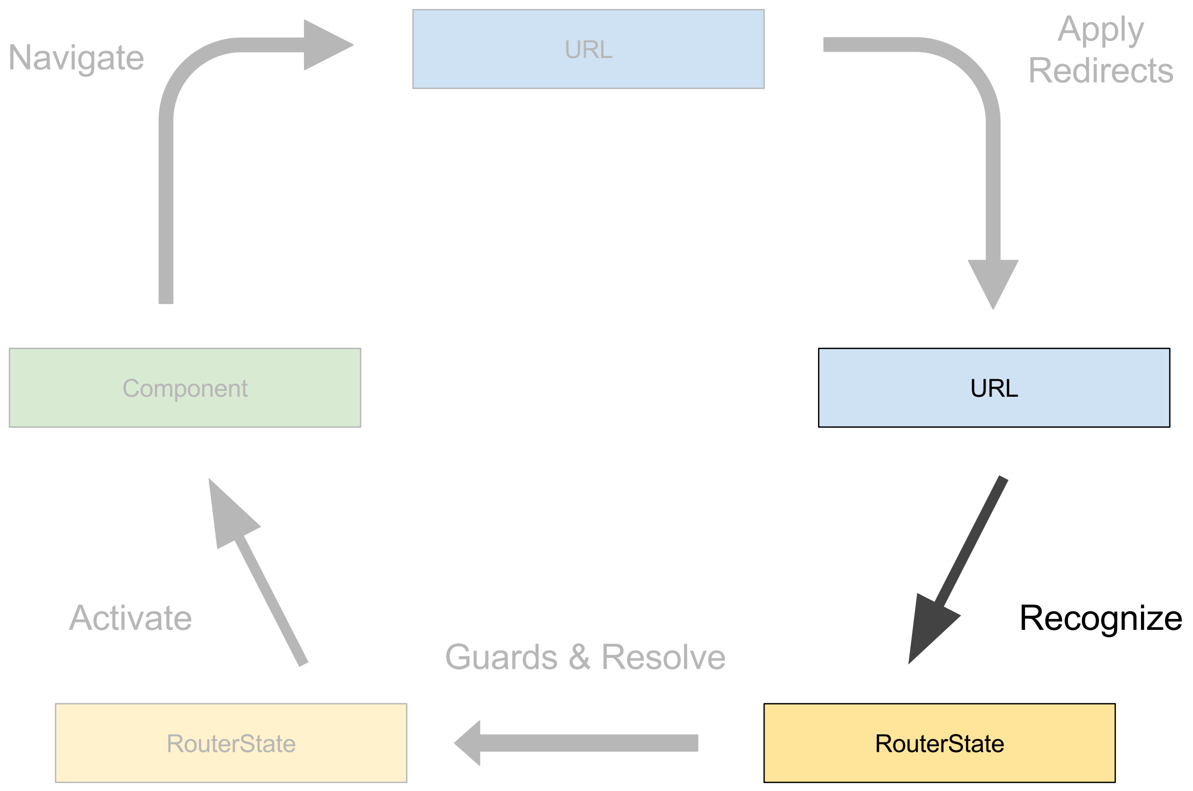
1. Applies redirects
2. Recognizes router states
3. Runs guards and resolves data
4. Activates all the needed components
5. Manages the navigation

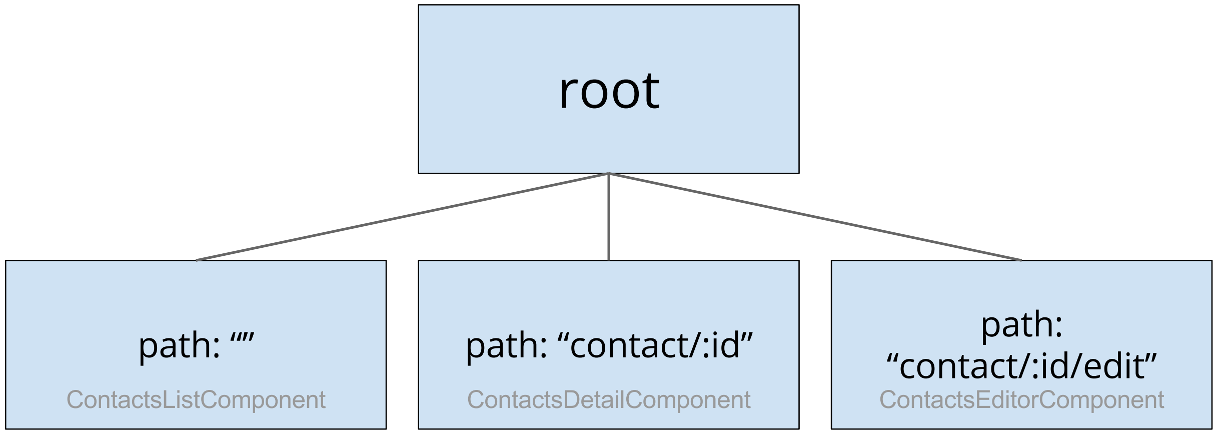

Navigate
URL
Apply Redirects
URL
Recognize
RouterState
Guards & Resolve
RouterState
Activate
Component

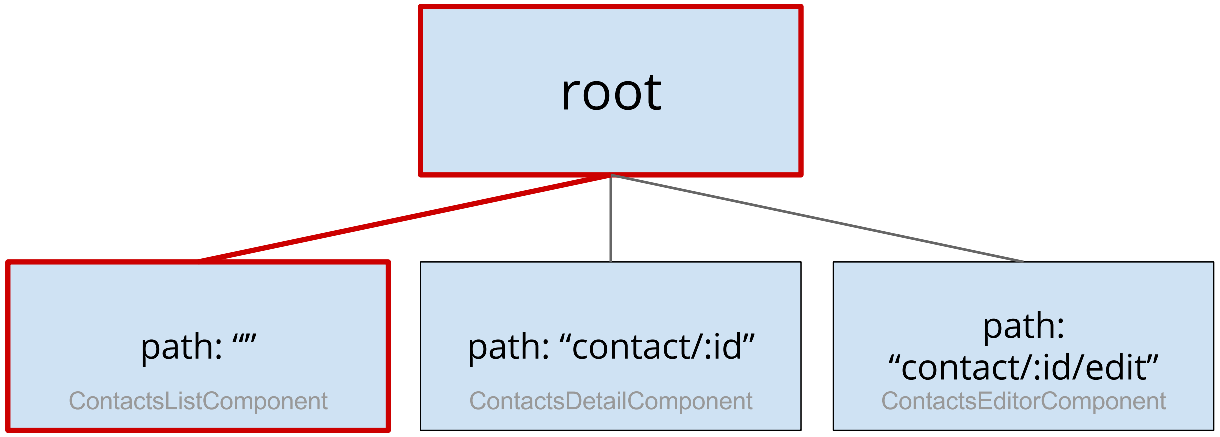

Navigate
URL
Apply Redirects
Component
URL
Activate
Recognize
Guards & Resolve
RouterState
RouterState

```
export const APP_ROUTES: Routes = [
  { path: '', component: ContactsListComponent },
  { path: 'contact/:id', component: ContactsDetailComponent },
  { path: 'contact/:id/edit', component: ContactsEditorComponent },
  { path: '**', redirectTo: '/' }
];
```

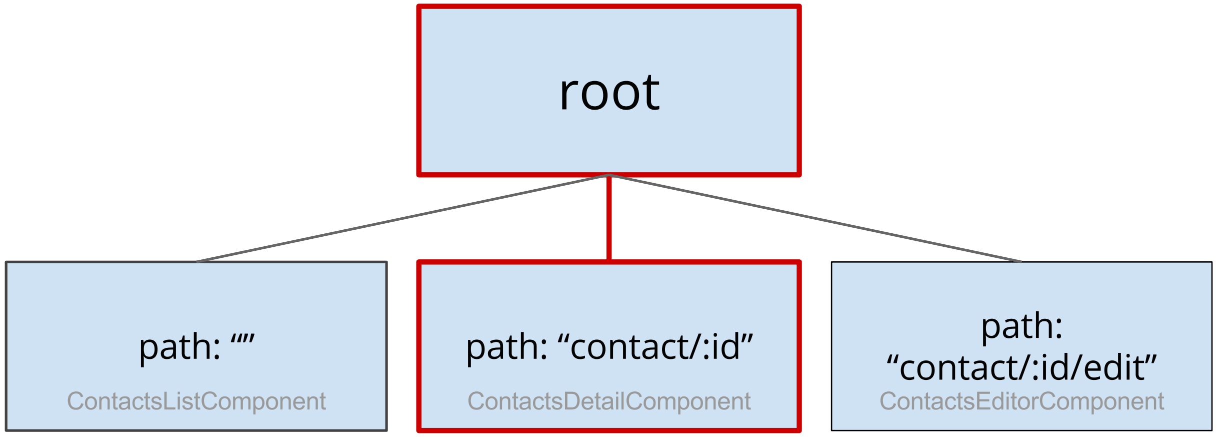

Navigate
URL
Apply Redirects
URL
Component
Activate
Recognize
Guards & Resolve
RouterState
RouterState


Navigate
URL
Apply Redirects
URL
Recognize
RouterState
Guards & Resolve
RouterState
Activate
Component

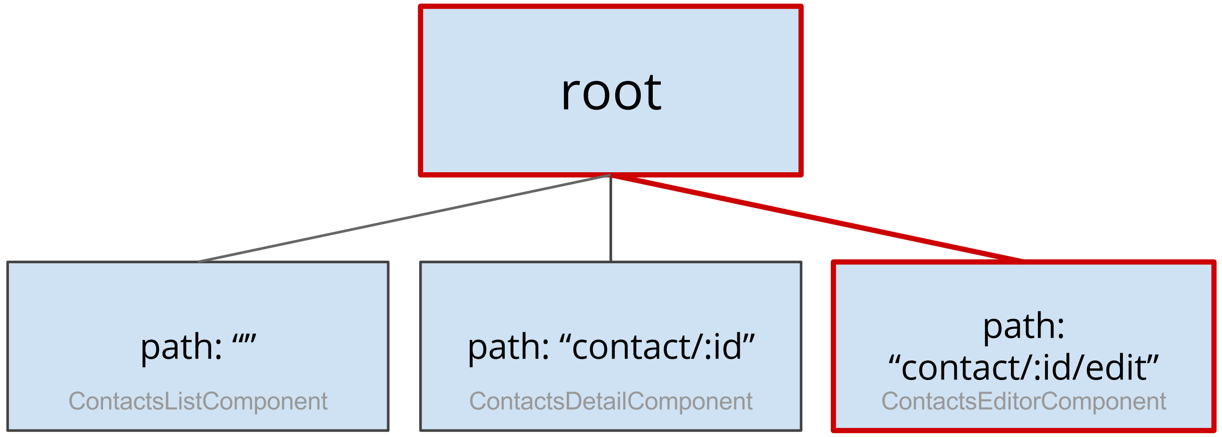

root
path: “”
ContactsListComponent
path: “contact/:id”
ContactsDetailComponent
path: “contact/:id/edit”
ContactsEditorComponent



http://localhost:4200

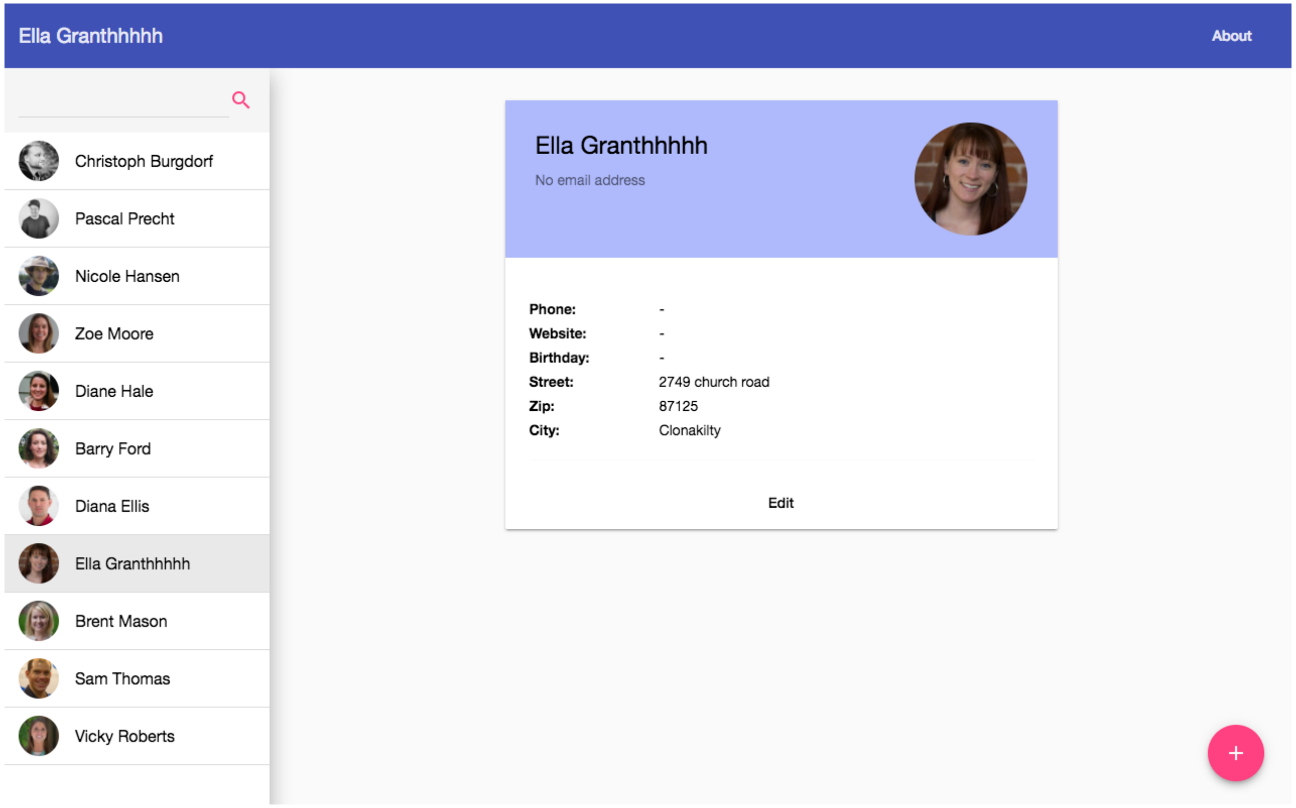


http://localhost:4200/3



http://localhost:4200/3/edit

Children Routes



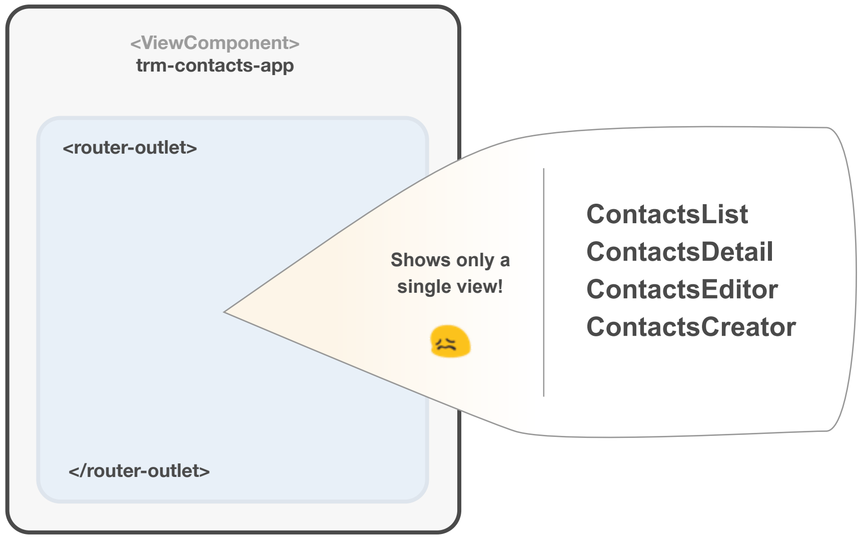
- Christoph Burgdorf
- Pascal Precht
- Nicole Hansen
- Zoe Moore
- Diane Hale
- Barry Ford
- Diana Ellis
- Ella Granthhhhh
- Brent Mason
- Sam Thomas
- Vicky Roberts

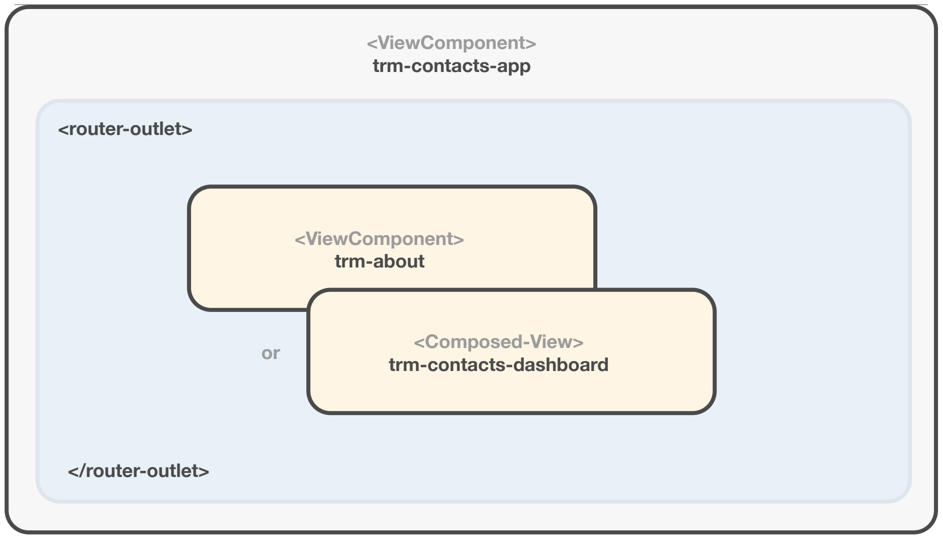
Ella Granthhhhh

No email address

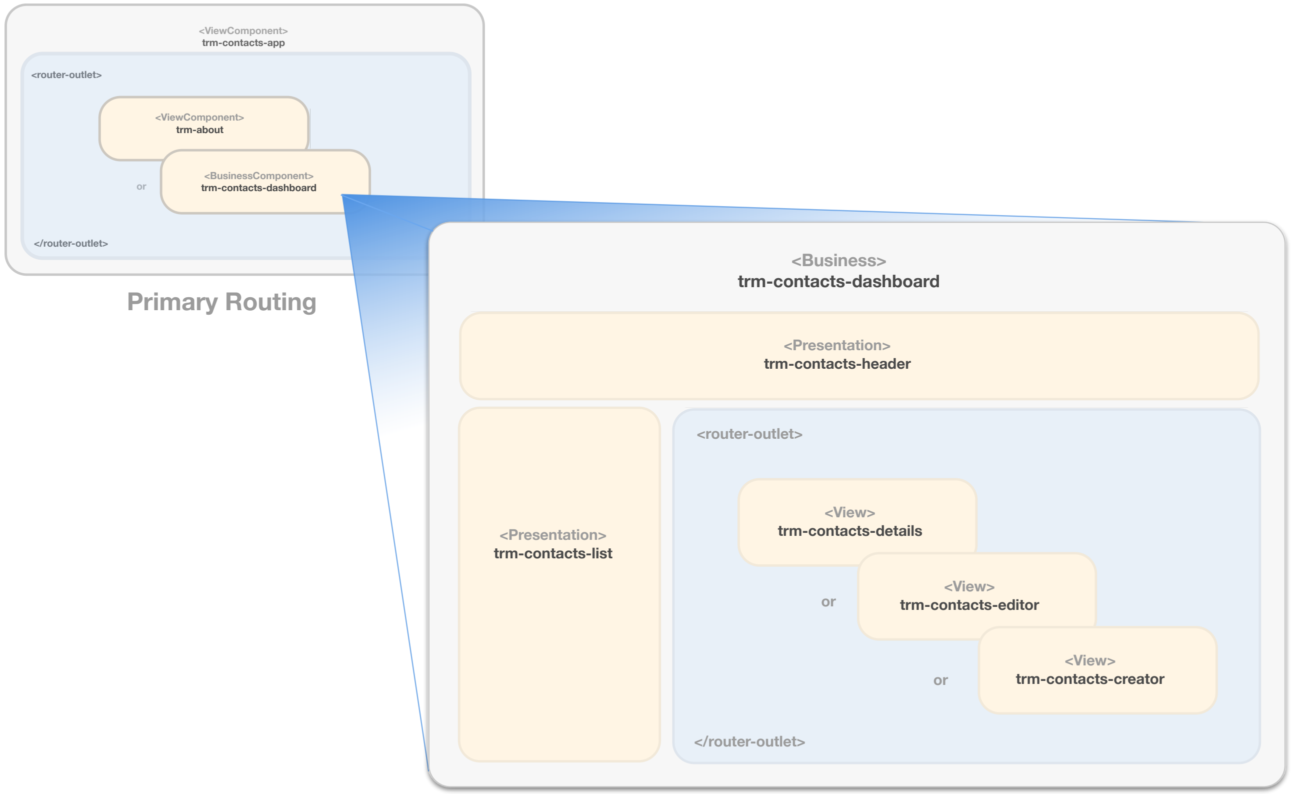
Phone:	-
Website:	-
Birthday:	-
Street:	2749 church road
Zip:	87125
City:	Clonakilty

Our current application has a problem:





Primary Routing



Child Routing in Nested Views 😉

Children Routes

We define children rountes using the route config's **children** property

```
{
  path: '',
  component: ContactsDashboardComponent,
  children: [

    { path: 'contact/:id', component: ContactsDetailComponent },
    { path: 'contact/:id/edit', component: ContactsEditorComponent }
  ]
}
```

Children Routes

We define children rountes using the route config's **children** property

```
{
  path: '',
  component: ContactsDashboardComponent,
  children: [

    { path: 'contact/:id', component: ContactsDetailComponent },
    { path: 'contact/:id/edit', component: ContactsEditorComponent }
  ]
}
```

Redirect Routes

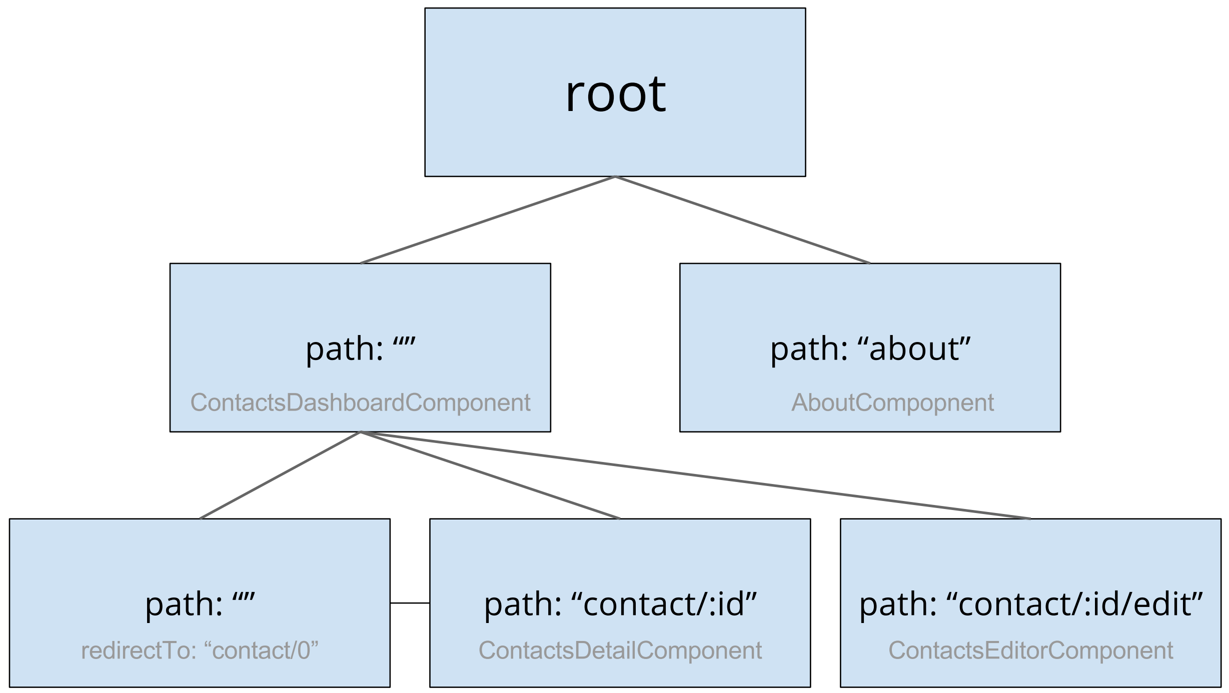
We can easily redirect to predefined routes using the **redirectTo** property.

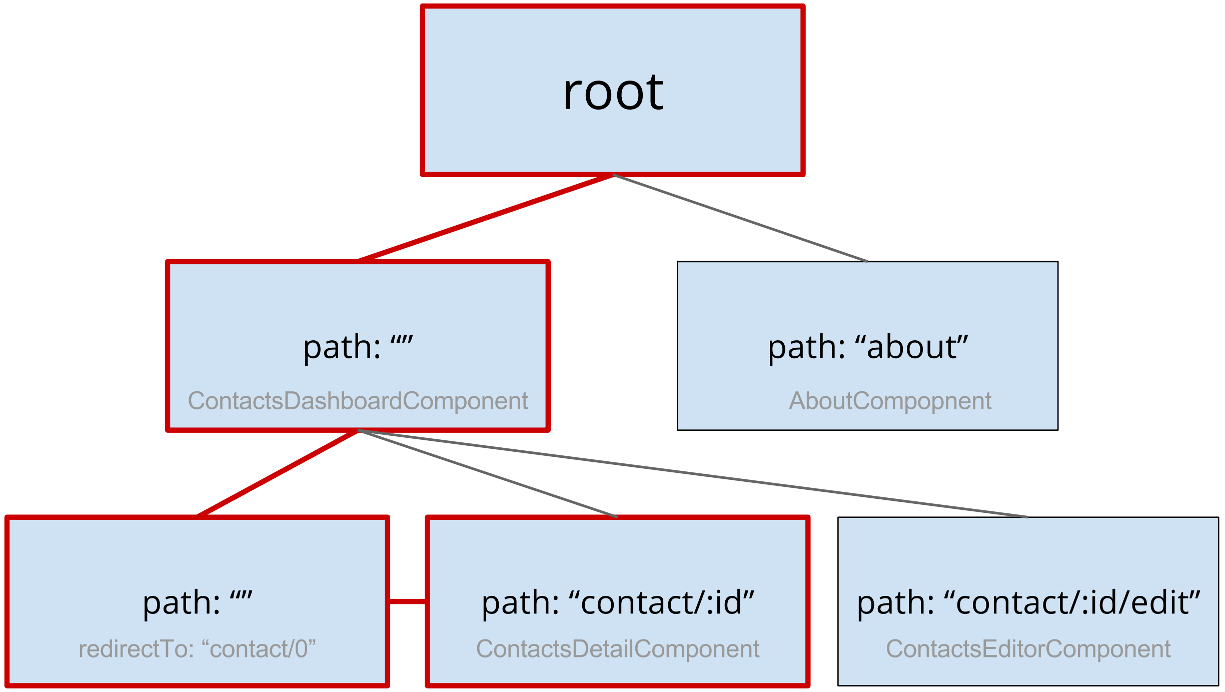
```
{
  path: '',
  component: ContactsDashboardComponent,
  children: [

    { path: 'contact/:id', component: ContactsDetailComponent },
    { path: 'contact/:id/edit', component: ContactsEditorComponent }
  ]
}
```

Redirect Routes

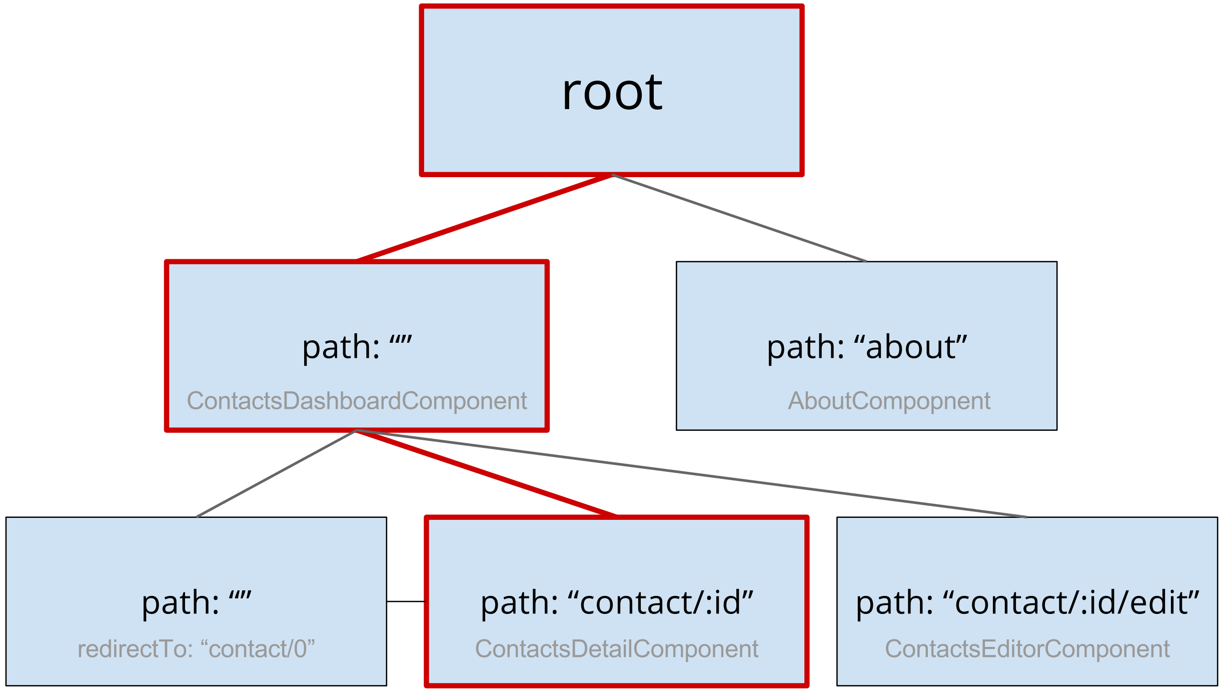
We can easily redirect to predefined routes using the **redirectTo** property.

```
{
  path: '',
  component: ContactsDashboardComponent,
  children: [
    { path: '', redirectTo: 'contact/0', pathMatch: 'full' },
    { path: 'contact/:id', component: ContactsDetailComponent },
    { path: 'contact/:id/edit', component: ContactsEditorComponent }
  ]
}
```

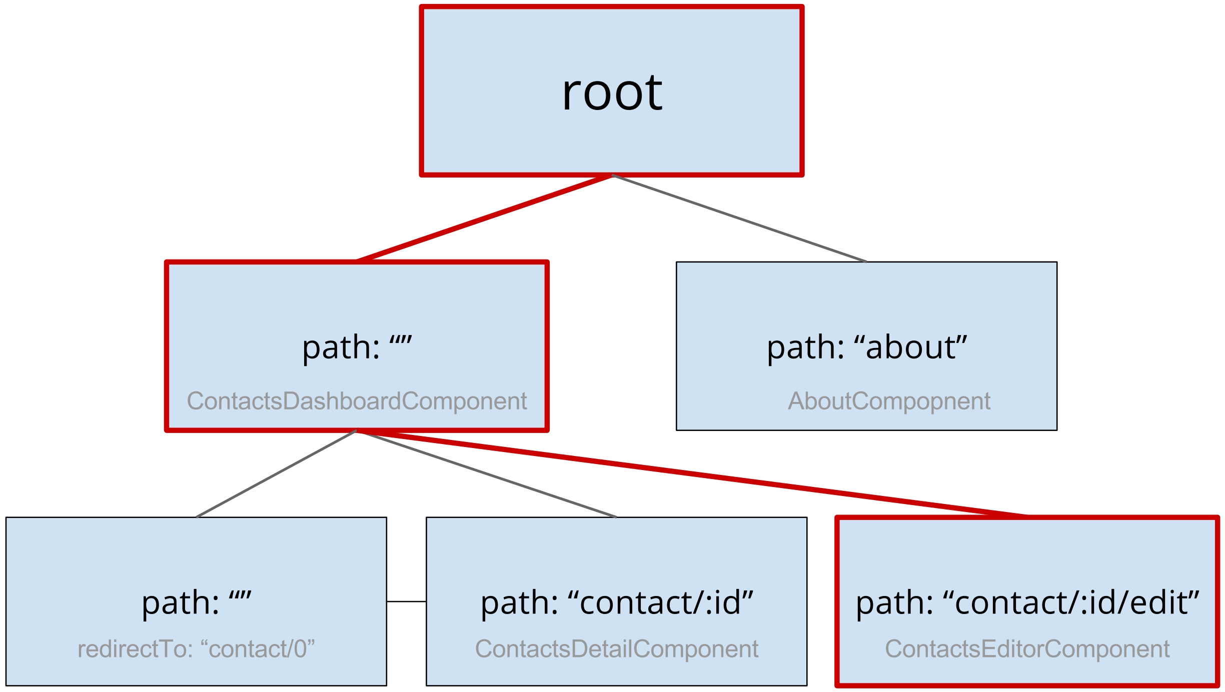

root
path: “”
ContactsDashboardComponent
path: “about”
AboutCompopnent
path: “”
redirectTo: “contact/0”
path: “contact/:id”
ContactsDetailComponent
path: “contact/:id/edit”
ContactsEditorComponent



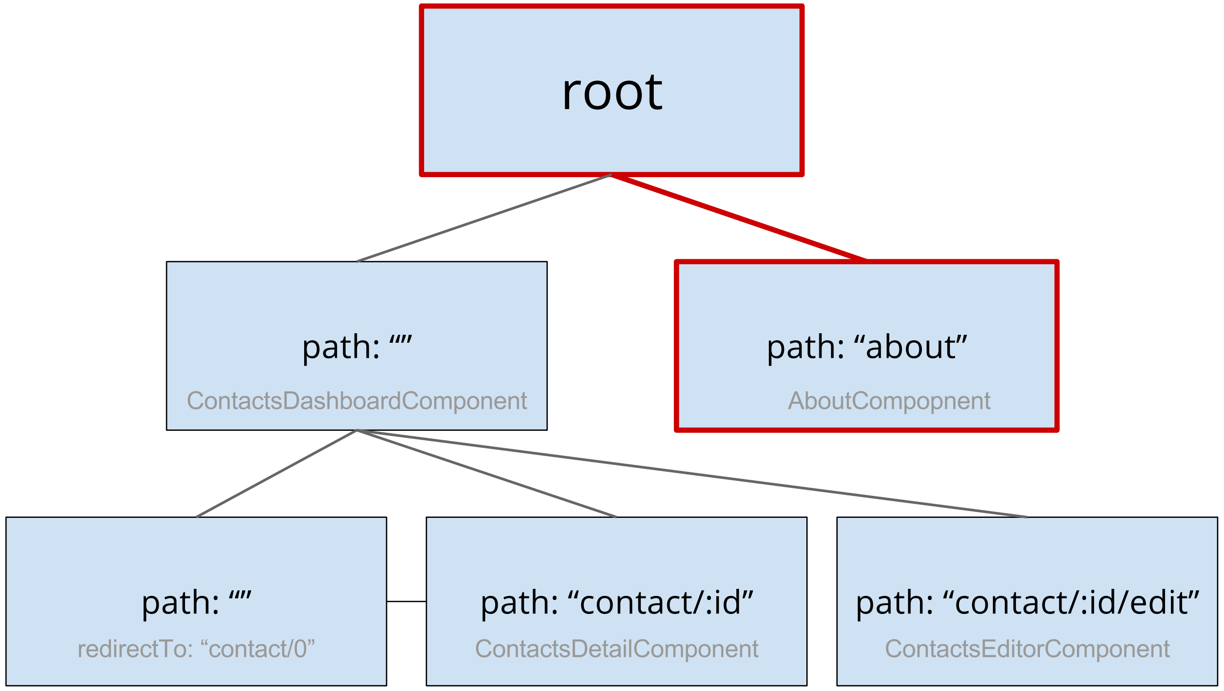
http://localhost:4200



http://localhost:4200/contact/3



http://localhost:4200/contact/3/edit



http://localhost:4200/about

```
@Component({
  selector: 'trm-contacts-dashboard',
  template: `
    <mat-drawer-container>
      <mat-drawer mode="side" opened="true">

      </mat-drawer>
      <div class="main-content">

      </div>
    </mat-drawer-container>
  `,
})
export class ContactsDashboardComponent {}
```

```
@Component({
  selector: 'trm-contacts-dashboard',
  template: `
    <mat-drawer-container>
      <mat-drawer mode="side" opened="true">

      </mat-drawer>
      <div class="main-content">

      </div>
    </mat-drawer-container>
  `,
})
export class ContactsDashboardComponent {}
```

```
@Component({
  selector: 'trm-contacts-dashboard',
  template: `
    <mat-drawer-container>
      <mat-drawer mode="side" opened="true">
        <trm-contacts-list></trm-contacts-list>
      </mat-drawer>
      <div class="main-content">
        <router-outlet></router-outlet>
      </div>
    </mat-drawer-container>
  `,
})
export class ContactsDashboardComponent {}
```

Parameter changes

We can subscribe to route parameter changes when Components are reused.

```
@Component()
export class ContactsDetailComponent {
  ...
  ngOnInit() {
    this.route.params.subscribe(params => {
      // do something with params
    })
  }
}
```

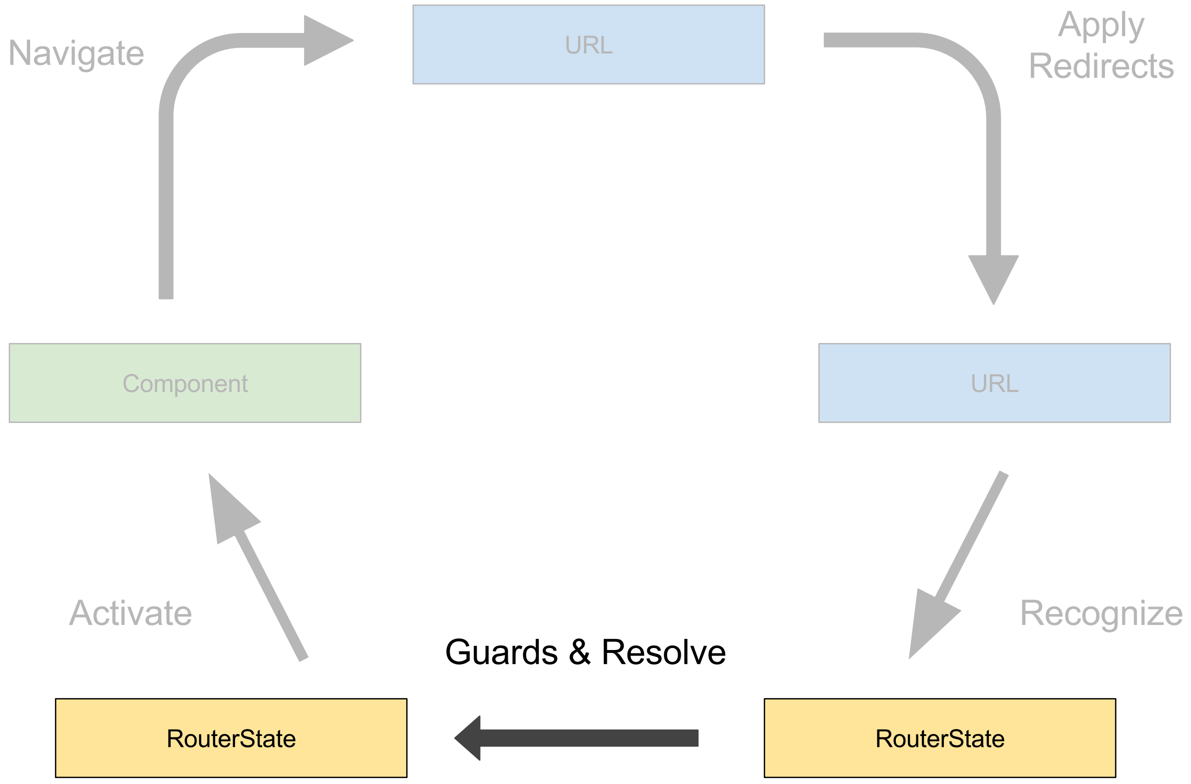
Parameter changes

We can subscribe to route parameter changes when Components are reused.

```
@Component()
export class ContactsDetailComponent {
  ...
  ngOnInit() {
    this.route.params.subscribe(params => {
      // do something with params
    })
  }
}
```

Exercise: Child Routes

Navigation Guards


Navigate
URL
Apply Redirects
Component
URL
Activate
Recognize
Guards & Resolve
RouterState
RouterState

Navigation Guards

Route guards can prevent users from navigating to or from certain routes, if needed.

- **canLoad** - Determine module can be loaded
- **canActivate** - Determine if user is allowed to route to component
- **canActivateChild** - Determines if child route can be loaded
- **canDeactivate** - Determine if user is allowed to route from component

Defining Guards

Guards are functions that return either **true/false** or an **Observable<boolean>**

```
export function confirmNavigationGuard() {
  return window.confirm('Are you sure?');
}
```

Registering Guards

Every Guard needs to be registered with a provider.

```
@NgModule({
  ...
  providers: [
    ...
    {
      provide: 'ConfirmNavigationGuard',
      useValue: confirmNavigationGuard
    }
  ]
})
export class ContactsModule {}
```

Registering Guards

Every Guard needs to be registered with a provider.

```
@NgModule({
  ...
  providers: [
    ...
    {
      provide: 'ConfirmNavigationGuard',
      useValue: confirmNavigationGuard
    }
  ]
})
export class ContactsModule {}
```

Using Guards

We use guards by adding them to either **canActivate**, **canDeactivate**, **canLoad** or **canActivateChild** properties of our route config

```
// app.routes.ts
{
  path: 'contact/:id/edit',
  component: ContactsEditorComponent,
  canDeactivate: ['ConfirmNavigationGuard']
}
```

Using Guards

We use guards by adding them to either **canActivate**, **canDeactivate**, **canLoad** or **canActivateChild** properties of our route config

```
// app.routes.ts
{
  path: 'contact/:id/edit',
  component: ContactsEditorComponent,
  canDeactivate: ['ConfirmNavigationGuard']
}
```

Using Guards

We use guards by adding them to either **canActivate**, **canDeactivate**, **canLoad** or **canActivateChild** properties of our route config

```
// app.routes.ts
{
  path: 'contact/:id/edit',
  component: ContactsEditorComponent,
  canDeactivate: ['ConfirmNavigationGuard']
}
```

Guards are executed in the defined order

Exercise: CanDeactivate Guard

Resolvers

Resolvers

After guards have been executed, we can use resolvers to defer the component instantiation until certain data is loaded.

Registering Resolvers

Resolvers are defined on a route's `resolve` object property.

```
// app.routes.ts
{
  path: 'contact/:id',
  component: ContactsDetailComponent,
  resolve: {
    contact: ContactResolver
  }
}
```

Registering Resolvers

Resolvers are defined on a route's `resolve` object property.

```
// app.routes.ts
{
  path: 'contact/:id',
  component: ContactsDetailComponent,
  resolve: {
    contact: ContactResolver
  }
}
```

Defining Resolvers

A resolver is a class with a **resolve()** method that returns an **Observable<any>** or **Promise<any>**

```
@Injectable()
class ContactResolver implements Resolve<Contact> {

  constructor(private contactsService: ContactsService) {}

  resolve(route: ActivatedRouteSnapshot) {
    return this.contactsService
               .getContact(route.paramMap.get('id'));
  }
}
```

Defining Resolvers

A resolver is a class with a **resolve()** method that returns an **Observable<any>** or **Promise<any>**

```
@Injectable()
class ContactResolver implements Resolve<Contact> {

  constructor(private contactsService: ContactsService) {}

  resolve(route: ActivatedRouteSnapshot) {
    return this.contactsService
               .getContact(route.paramMap.get('id'));
  }
}
```

Defining Resolvers

A resolver is a class with a **resolve()** method that returns an **Observable<any>** or **Promise<any>**

```
@Injectable()
class ContactResolver implements Resolve<Contact> {

  constructor(private contactsService: ContactsService) {}

  resolve(route: ActivatedRouteSnapshot) {
    return this.contactsService
               .getContact(route.paramMap.get('id'));
  }
}
```

Defining Resolvers

A resolver is a class with a **resolve()** method that returns an **Observable<any>** or **Promise<any>**

```
@Injectable()
class ContactResolver implements Resolve<Contact> {

  constructor(private contactsService: ContactsService) {}

  resolve(route: ActivatedRouteSnapshot) {
    return this.contactsService
               .getContact(route.paramMap.get('id'));
  }
}
```

Receiving resolved data

ActivatedRoute exposes the resolved data on an observable **data** property

```
@Component(...)
class ContactsEditorComponent implements OnInit {
  ...
  ngOnInit() {
    this.route.data
        .pipe(map(data => data['contact']))
        .subscribe(contact => this.contact = contact);
  }
}
```

Receving resolved data

ActivatedRoute exposes the resolved data on an observable **data** property

```
@Component(...)
class ContactsEditorComponent implements OnInit {
  ...
  ngOnInit() {
    this.route.data
        .pipe(map(data => data['contact']))
        .subscribe(contact => this.contact = contact);
  }
}
```

Lazy Loading

Loading modules lazy

We can easily lazy load **NgModules** by using a route's **loadChildren** property

```
// app.routes.ts
export const APP_ROUTES = [
  ...
  {
    path: 'about',
    loadChildren: './about/about.module#AboutModule'
  }
];
```

Loading modules lazy

We can easily lazy load **NgModules** by using a route's **loadChildren** property

```
// app.routes.ts
export const APP_ROUTES = [
  ...
  {
    path: 'about',
    loadChildren: './about/about.module#AboutModule'
  }
];
```

Loading modules lazy

We can easily lazy load **NgModules** by using a route's **loadChildren** property

```
// app.routes.ts
export const APP_ROUTES = [
  ...
  {
    path: 'about',
    loadChildren: './about/about.module#AboutModule'
  }
];
```

#AboutModule specifies the module class in the lazy-loaded bundle

About Module

AboutComponent isn't visited often, maybe never.
That's why we want to lazy-load it.

```
@NgModule({
  imports: [
    ...
    RouterModule.forChild([
      { path: '', component: AboutComponent }
    ])
  ],
  declarations: [AboutComponent]
})
export class AboutModule {}
```

About Module

AboutComponent isn't visited often, maybe never.
That's why we want to lazy-load it.

```
@NgModule({
  imports: [
    ...
    RouterModule.forChild([
      { path: '', component: AboutComponent }
    ])
  ],
  declarations: [AboutComponent]
})
export class AboutModule {}
```

Exercise: Lazy Loading

THOUGHTRAM

EXTEND YOUR MEMORY

- git add .
- git commit -am "(completed) - routing"
- git tag classroom/routing