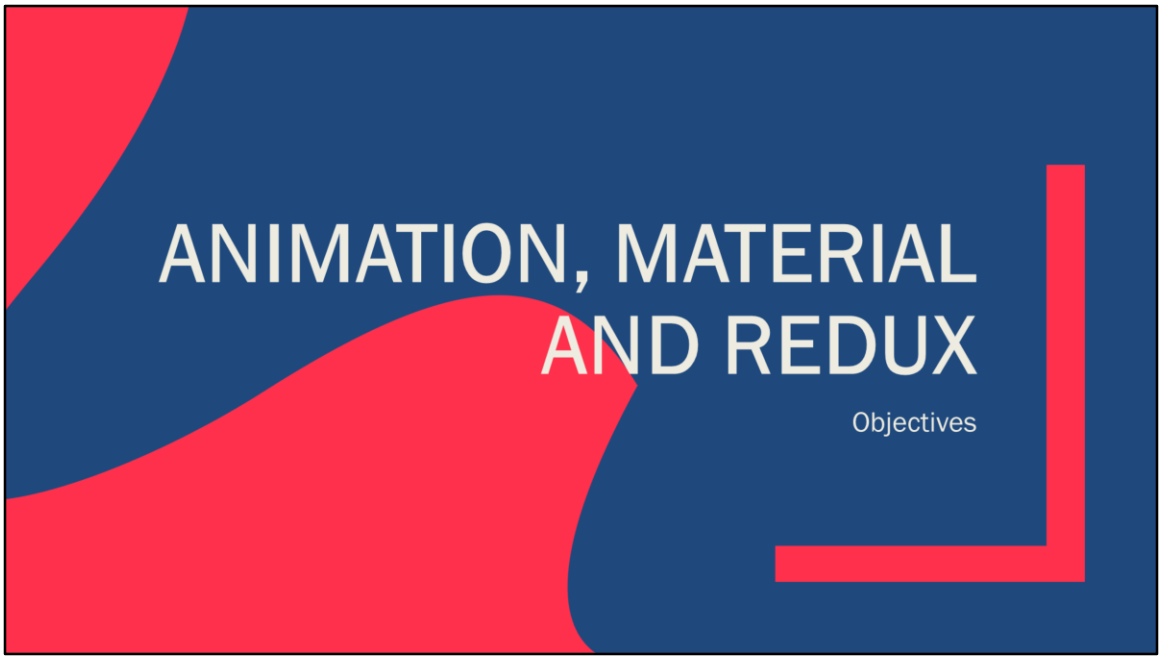
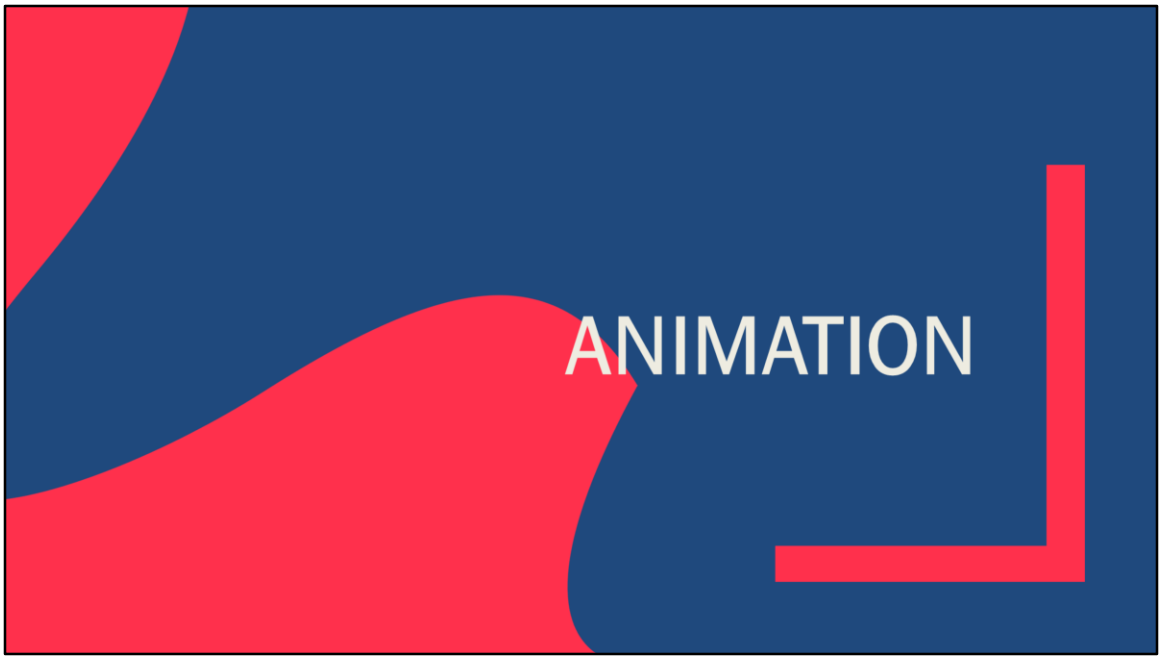

ANGULAR 6
Lesson 10


ANIMATION, MATERIAL AND REDUX
Objectives

Capgemini

Animation, Material, Redux

- Angular Animation
- Material Design Bootstrap with Angular
- New features in angular cli
  - *ng update*
  - *ng add*
- Angular Elements
- CLI workspaces
- RxJs 6 Support
- Redux introduction, Installing redux
- Building Blocks of Redux
- Working with Actions, Reducers


ANIMATION

Capgemi

Angular Animation

- Motion is an important aspect in the design of modern web applications.
- Good user interfaces transition smoothly between states with engaging animations that call at where it's needed.
- Well-designed animations can make a UI not only more fun but also easier to use.
- Angular's animation system lets you build animations that run with the same kind of native performance found in pure CSS animations.

SETTING UP THE ANGULAR 6 ANIMATION PROJECT

Capgemi

- To get started with a practical Angular 6 Animations project, we'll need to set up an Angular pr first. Therefore we'll be using Angular CLI:

  > *ng new angularanimation*

- This commend is creating a new project directory angularanimation and installing the Angular default project inside that newly created folder. Now let's change into that folder:

  > cd angularanimation

- and test if everything is working correctly by starting up the development web server:

  > ng serve

SETTING UP THE ANGULAR 6 ANIMATION PROJECT

Capgemi

- If you now access URL http://localhost:4200/ in the browser you should be able to see the fol output:

SETTING UP ANGULAR ANIMATIONS

Capgemi

- Now that the Angular 6 project is ready we're able to set up Angular Animations for this project first step is to add the following import statement in app.module.ts:

  import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

- Next we need to add BrowserAnimationsModule to the array which is assigned to the import pr of the @NgModule decorator:

  imports: [
  BrowserModule,
  BrowserAnimationsModule
  ],

- This is making the content of BrowserAnimationsModule available to our application, so that w able to import animations in our components.
- Let's create a new component first:

  > ng g c animate

  This is creating the following four new files in your project:

  - animate.component.ts
  - animate.component.spec.ts
  - animate.component.html
  - animate.component.css

SETTING UP ANGULAR ANIMATIONS

Capgemi

- The new component AnimateComponent is automatically added to the main application modu in app.module.ts and therewith it is made available in your application:

```
import { BrowserModule } from '@angular/platform-browser';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { AnimateComponent } from './animate/animate.component';
@NgModule({
  declarations: [    AppComponent,    AnimateComponent  ],
  imports: [    BrowserModule,    BrowserAnimationsModule  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Capgemi

MANAGE STATE CHANGES IN APPCOMPONENT

- The application should trigger animations based on state changes which are invoked by pres buttons. To manage the state of the application we need to introduce a class member variab in AppComponent:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'angularanimation01';
  toState = 'state1';
  changeState(state: any) {
    this.toState = state;
  }
}
```

Capgemi

MANAGE STATE CHANGES IN APPCOMPONENT

- The member variable is named toState and is changed by the method changeState. This me used as an event handler method for the click event of the buttons. Let's take a look at the corresponding HTML template code which needs to be inserted into app.component.html:

```
<div class="container p-4">
  <div class="row">
    <div class="col text-center">
      <a (click)="changeState('state2')" class="btn btn-danger">Change To State 2</a>
    </div>
    <div class="col text-center">
      <a (click)="changeState('state1')" class="btn btn-info">Change To State 1</a>
    </div>
  </div>
  <div class="row justify-content-center align-items-center">
    <app-animate [currentState]="toState"></app-animate>
  </div>
</div>
```

Capgemi

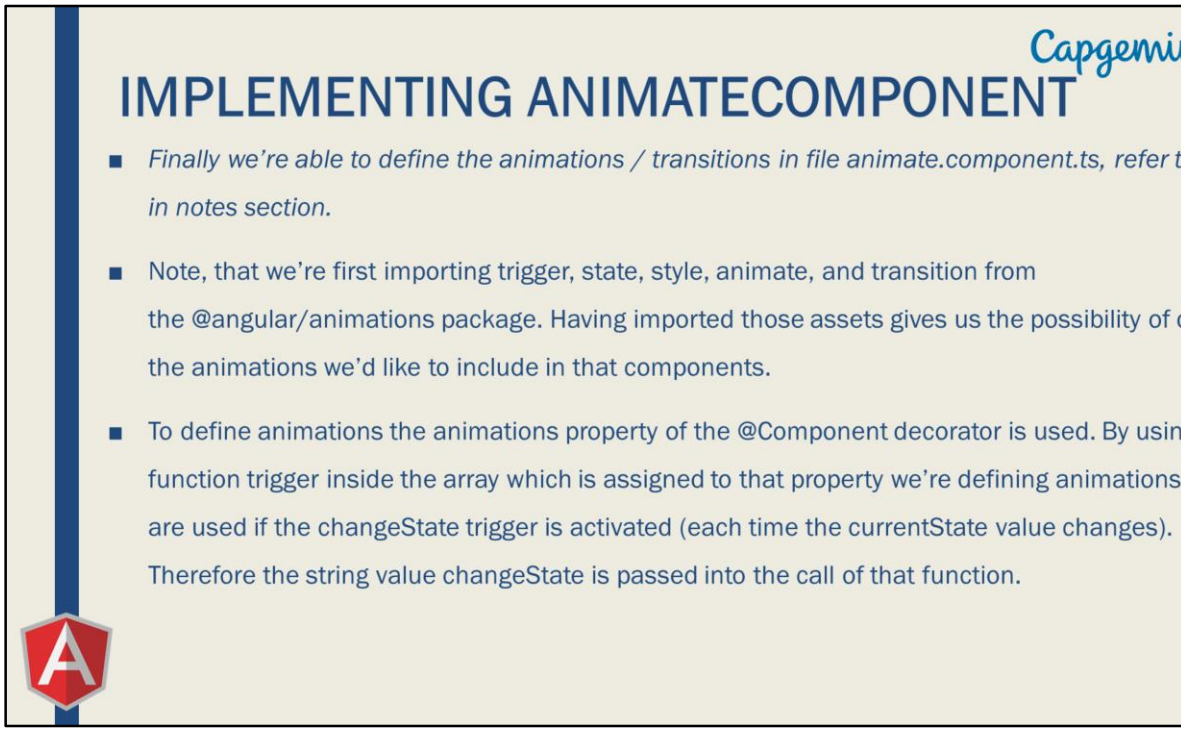
MANAGE STATE CHANGES IN APPCOMPONENT

- We're making use of Bootstrap 4 CSS classes in this template code. This requires us to inclu Bootstrap CSS file in our application. The easierst way to do so, is to include Bootstrap from in index.html by adding the following line of code to the head section:
- <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">

IMPLEMENTING ANIMATECOMPONENT

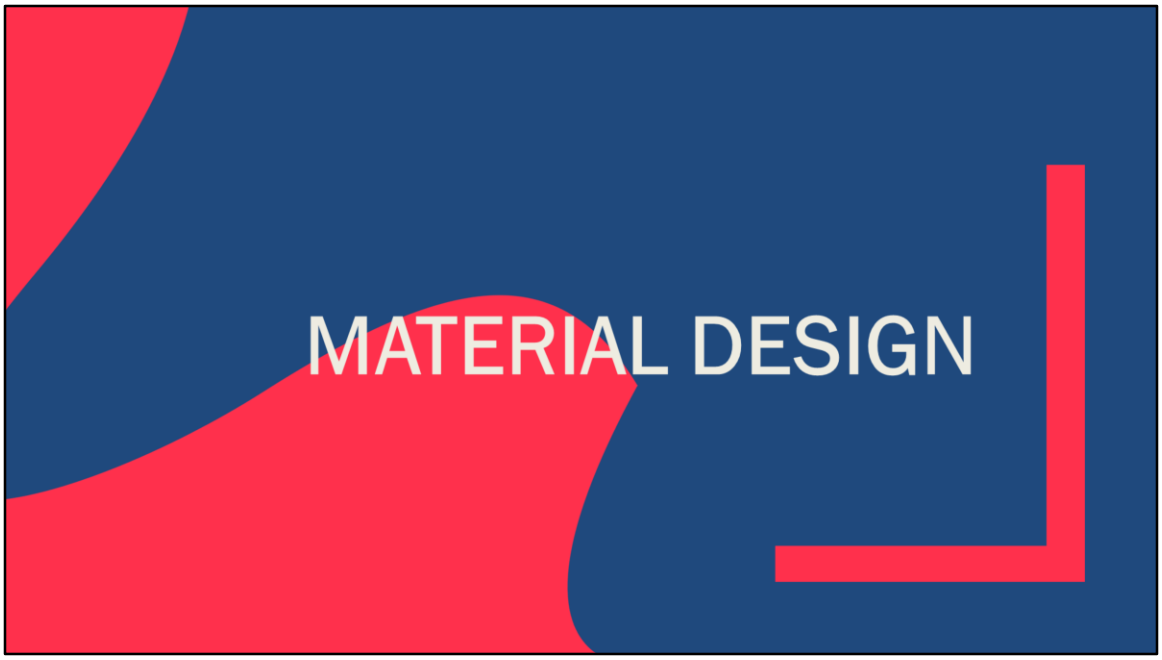
- Having prepared AppComponent we can now continue with the implementation of AnimateComponent. First, we need to include a div element in animate.component.html w displaying the rectangle which is being animated.

```
<div [@changeState]="currentState" class="myblock mx-auto"></div>
The corresponding CSS code for the myblock class needs to be inserted
into animate.component.css:
.myblock {
    background-color: green;
    width: 300px;
    height: 250px;
    border-radius: 5px;
    margin: 5rem;
}
```




```
import { Component, OnInit, Input } from '@angular/core';
import { trigger, state, style, animate, transition } from '@angular/animations';
@Component({
  selector: 'app-animate',
  templateUrl: './animate.component.html',
  styleUrls: ['./animate.component.css'],
  animations: [
    trigger('changeState', [
      state('state1', style({
        backgroundColor: 'green',
        transform: 'scale(1)'
      })),
      state('state2', style({
        backgroundColor: 'red',
        transform: 'scale(1.5)'
      })),
      transition('*=>state1', animate('300ms')),
      transition('*=>state2', animate('2000ms'))
    ])
```

```
  ]
})
export class AnimateComponent implements OnInit {
  @Input() currentState;
  constructor() { }
  ngOnInit() {
  }
}
```


MATERIAL DESIGN

Capgemini

ANGULAR 6 MATERIAL DESIGN

- Angular Material is a collection of Material Design components for Angular.
- By using these components you can apply Material Design very easily.
- With the release of Angular 6 the usage of Angular Material has become easier as well.
- The Angular Material website can be found at https://material.angular.io/.

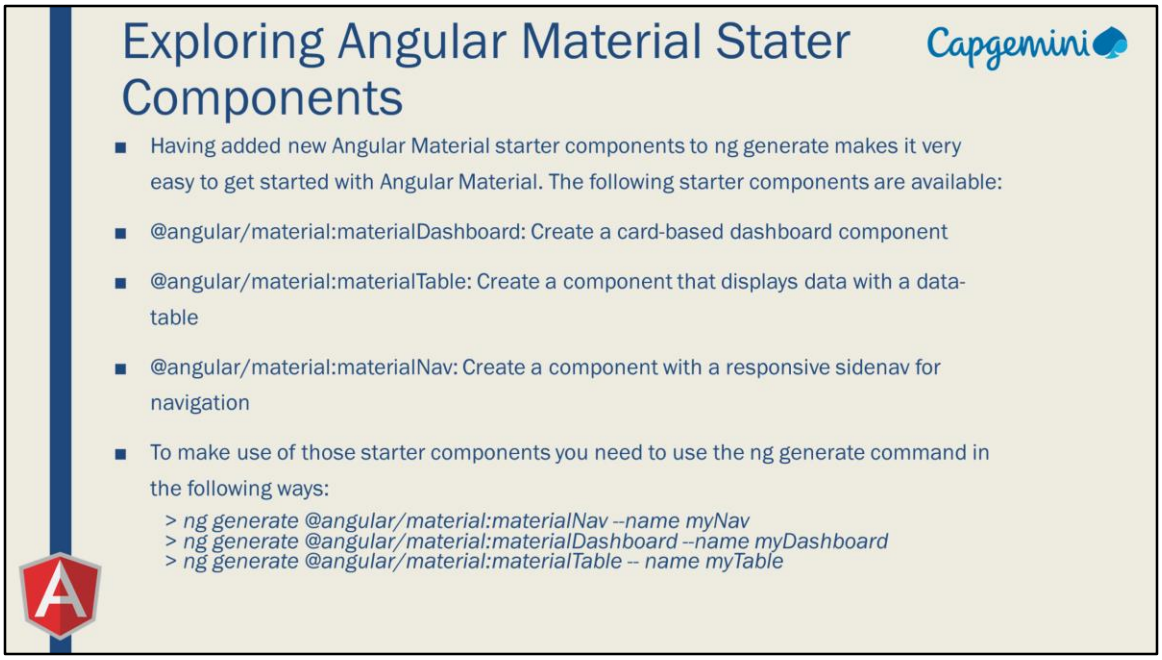
SETTING UP THE ANGULAR 6 PROJECT FOR MATERIAL DESIGN

- To get start we first need to setup the Angular 6 project. This is done by using Angular CLI (https://cli.angular.io/). If you have not installed Angular CLI on your system yet you first need to follow the steps from the project's website to make the command line interface available on your system.
- Once Angular CLI has been installed successfully you can initiate the new project by using the NG command in the following way:
- > ng new ngMat01
- In this example NGMAT01 is the name of the new project. A new project folder (with that name) is created, the Angular project template is downloaded and the needed dependencies are installed.

USING NG ADD TO ADD ANGULAR MATERIAL

Capgemini

- With the release of Angular 6 the new ng add command is available which makes it easy to add new capabilities to the project. This command will use the package manager to download new dependencies and invoke corresponding installation scripts. This is making sure that the project is updated with dependencies, configuration changes and that package-specific initialization code is executed.
- In the following we'll use the ng add command to add Angular Material to the previously created Angular 6 application:
  - *> ng add @angular/material*
- By executing this command we're installing Angular Material and the corresponding theming into the project. Furthermore new starter components are registered into NG GENERATE.

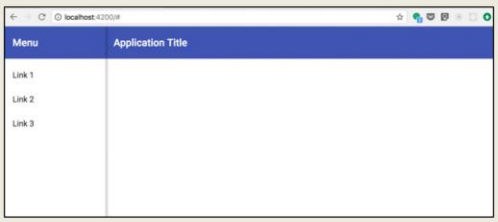


E.g. if you use the first command to generate a new myNav component you should be able to see the following output on the command line:
CREATE src/app/my-nav/my-nav.component.css (110 bytes)
CREATE src/app/my-nav/my-nav.component.html (945 bytes)
CREATE src/app/my-nav/my-nav.component.spec.ts (605 bytes)
CREATE src/app/my-nav/my-nav.component.ts (481 bytes)
UPDATE src/app/app.module.ts (795 bytes)
Four new files have been added to the project. Those files contain the implementation of the Angular Material navigation component. MyNavComponent has been added to the declarations array of the @NgModuledecorator in AppModule, so that the component can be used in our Angular application.

Exploring Angular Material Stater Components

Capgemini

- To make it visible to the user delete the default content of file app.component.html and just insert the following element:
  - *<my-nav></my-nav>*
- This is the element which is used to include the output of MyNavComponent in the output which is presented in the browser.
- Having started the development web server with
  - *> ng server –open*
- you should be able to see the following result:



E.g. if you use the first command to generate a new myNav component you should be able to see the following output on the command line:
CREATE src/app/my-nav/my-nav.component.css (110 bytes)
CREATE src/app/my-nav/my-nav.component.html (945 bytes)
CREATE src/app/my-nav/my-nav.component.spec.ts (605 bytes)
CREATE src/app/my-nav/my-nav.component.ts (481 bytes)
UPDATE src/app/app.module.ts (795 bytes)
Four new files have been added to the project. Those files contain the implementation of the Angular Material navigation component. MyNavComponent has been added to the declarations array of the @NgModuledecorator in AppModule, so that the component can be used in our Angular application.

ADDING THE ROUTER

Capgemini

- Now that the navigation layout is available we're able to add Angular Router functionality easily. In app.module.ts add the following import statement to import RouterModule and Routes:
- import { RouterModule, Routes } from '@angular/router';
- Next add a router configuration array in the same file:

```
const appRoutes: Routes = [
  { path: 'first-page', component: FirstPageComponent },
  { path: 'second-page', component: SecondPageComponent },
  { path: 'third-page', component: ThirdPageComponent }
];
```

- Of course, FirstPageComponent, SecondPageComponent, and ThirdPageComponent are not available yet. We're going to add those components in the following steps.

ADDING THE ROUTER

Capgemini

- In order to activate the router configuration for our Angular application we need to make sure to add RouterModule to the imports-Array of the @NgModule in the following way:

```
imports: [
  ...
  RouterModule.forRoot(appRoutes),
  ...
],
```

- Next we need to add the router outlet (the place where the content of the route component is inserted) inside the <mat-sidenav-content>-Element in file my-nav.component.html:
  - *<router-outlet></router-outlet>*

ADDING THE ROUTER

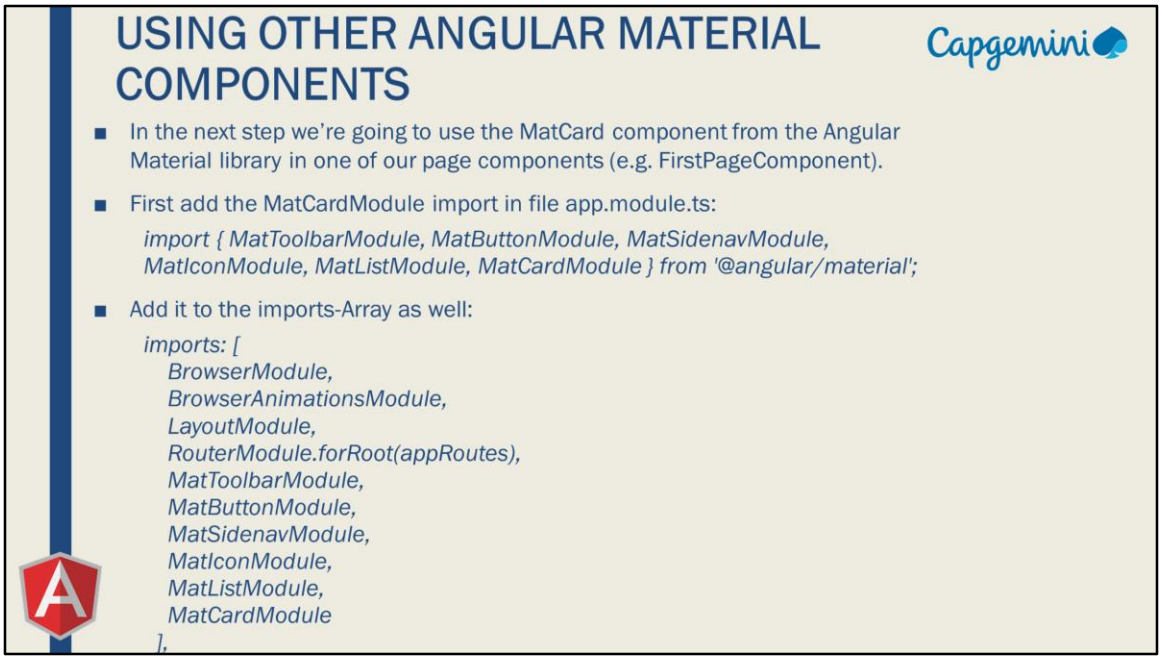
Capgemini

- Furthermore we need to update the links from the sidebar menu and use the routerLink directive to point to the respective routes:

```
<mat-nav-list>
    <a mat-list-item routerLink="/first-page">First Page</a>
    <a mat-list-item routerLink="/second-page">Second Page</a>
    <a mat-list-item routerLink="/third-page">Third Page</a>
</mat-nav-list>
```

- Finally, to make the router configuration work, add the three components by using the following commands:

```
> ng generate component FirstPage
> ng generate component SecondPage
> ng generate component ThirdPage
```



We've used the navigation starter component in our application so far. You can also make use of any of the other Angular Material components. To get an overview of available components take look at https://material.angular.io/components/categories.

USING OTHER ANGULAR MATERIAL COMPONENTS

Capgemini

- Now we're ready to use components of MatCardModule in the template code (e.g. first-page.component.html):

```
<mat-card class="example-card">
  <mat-card-header>
    <div mat-card-avatar class="example-header-image"></div>
    <mat-card-title>Shiba Inu</mat-card-title>  <mat-card-subtitle>Dog Breed</mat-card-subtitle>
  </mat-card-header>
  <img mat-card-image src="https://material.angular.io/assets/img/examples/shiba2.jpg" alt="Photo of a Shiba Inu">  <mat-card-content>
    <p>
      The Shiba Inu is the smallest of the six original and distinct spitz breeds of dog from Japan. A small, agile dog that copes very well with mountainous terrain, the Shiba Inu was originally bred for hunting.
    </p>
  </mat-card-content>  <mat-card-actions>
    <button mat-button>LIKE</button>    <button mat-button>SHARE</button>
  </mat-card-actions></mat-card>
```

We've used the navigation starter component in our application so far. You can also make use of any of the other Angular Material components. To get an overview of available components take look at https://material.angular.io/components/categories.

USING OTHER ANGULAR MATERIAL COMPONENTS

Capgemini

- Insert the CSS code in first-page.component.css as well:

```
.example-card {
  max-width: 400px;
}
.example-header-image {
  background-image: url('https://material.angular.io/assets/img/examples/shiba1.jpg');
  background-size: cover;
}
```

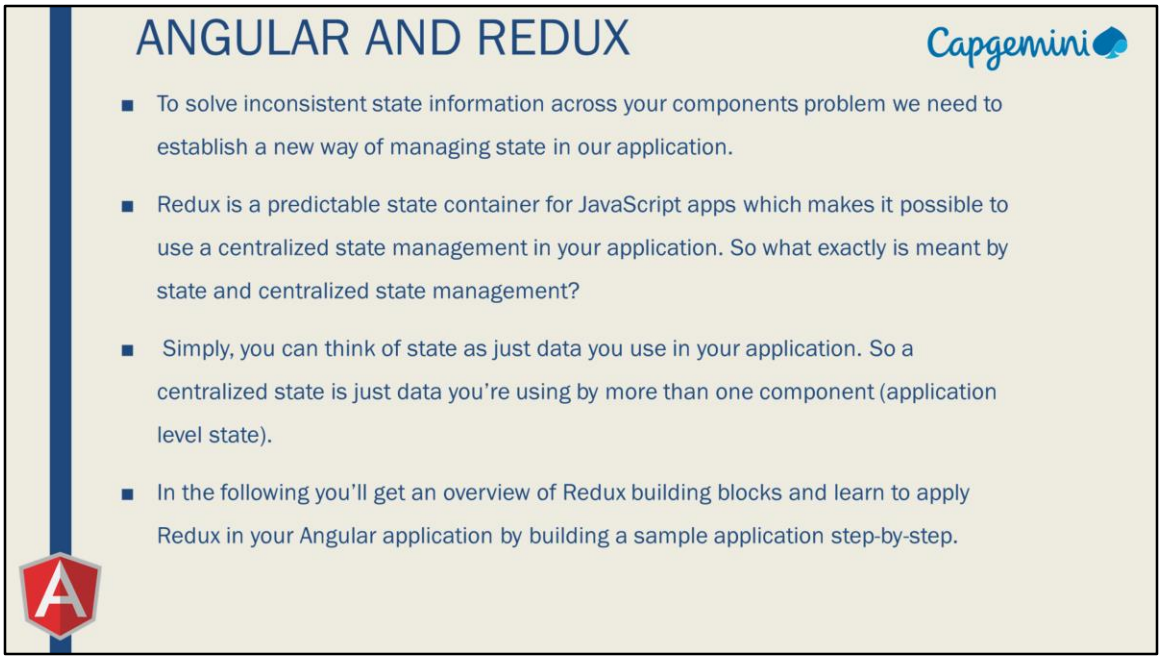
All other Material Design components from the Angular Material library can be used in the same way.

Capgemini

New features in angular cli

- ng update
- ng add

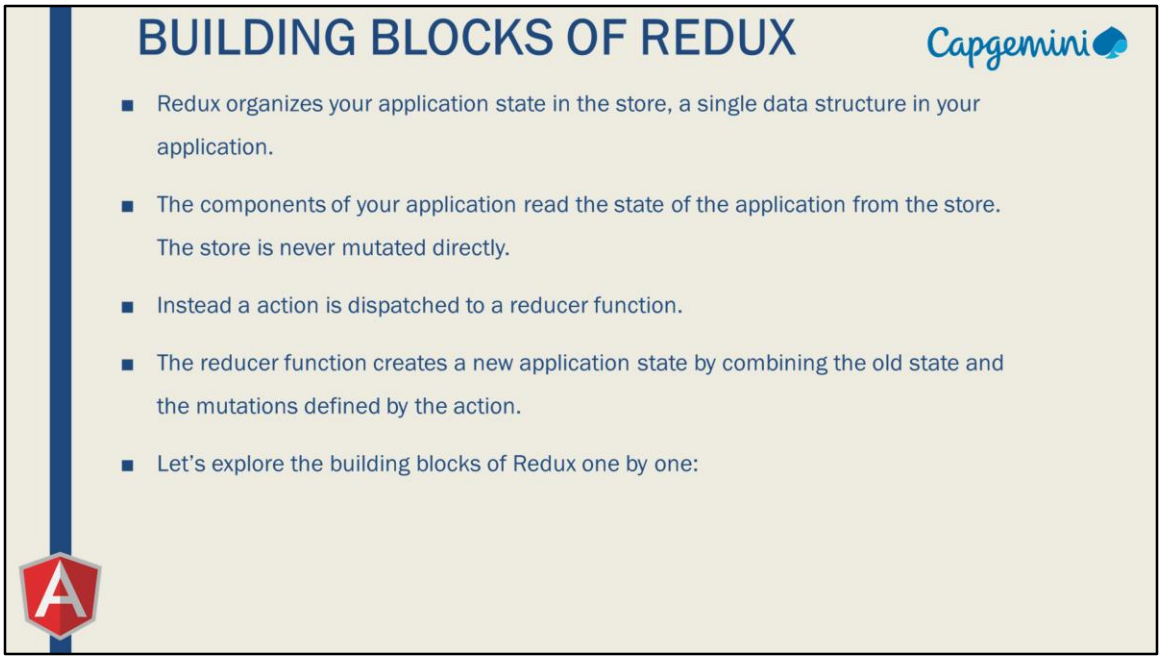

REDUX



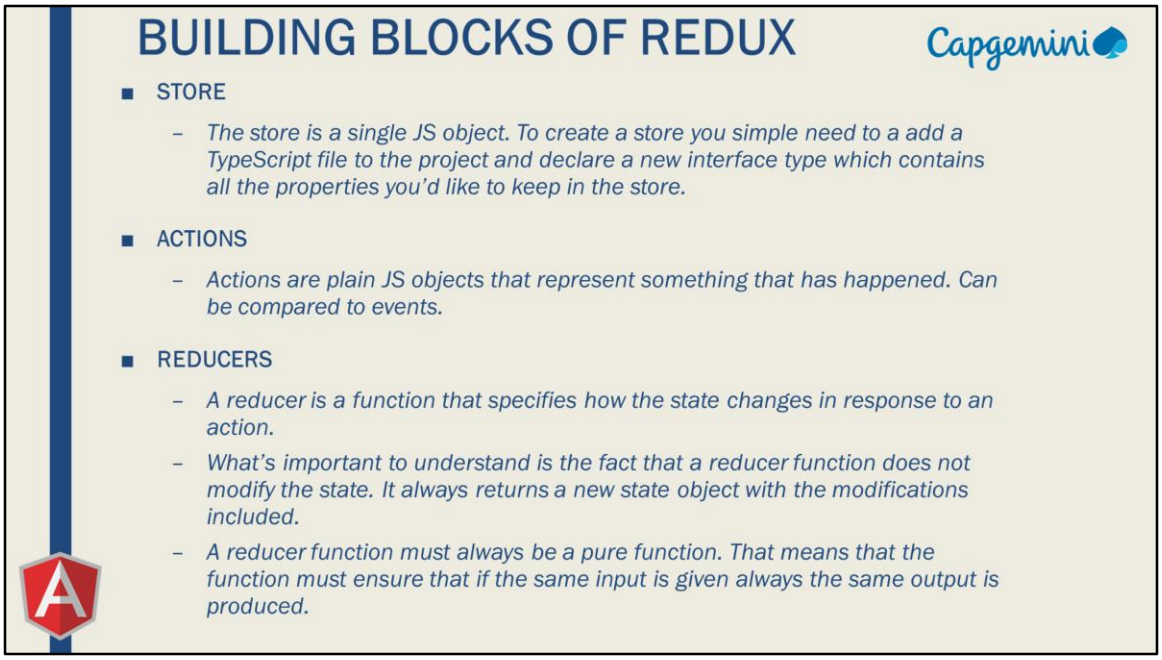
The concept of Single Page Web Applications is great for building modern web-based applications. However as more and more applications are being transferred into the browser the complexity and the amount of data which needs to be managed is consistently growing.
Many modern web frameworks, like Angular, are using a component-based approach to divide the application into smaller units. This approach is great because using components helps to better structure your project, keep the overview and make code reusable.
Implementing a component in Angular 4 means that the component is also managing it's own state (it's own data). E.g. if a component wants to display data from a service, corresponding service methods are called and the returned data is stored in properties of the component class. In the component template you can then access and embed those properties, so that the property values are displayed in the HTML output.
As the complexity of your application is increasing you'll be using more and more components and data which must be shared across components is passed down the component tree, so that the state of every component is always updated with the relevant data.
This approach is feasible if you're working with just a very few components. However,

if the number of components in your application is increasing this way of managing state becomes cumbersome and error-prone.

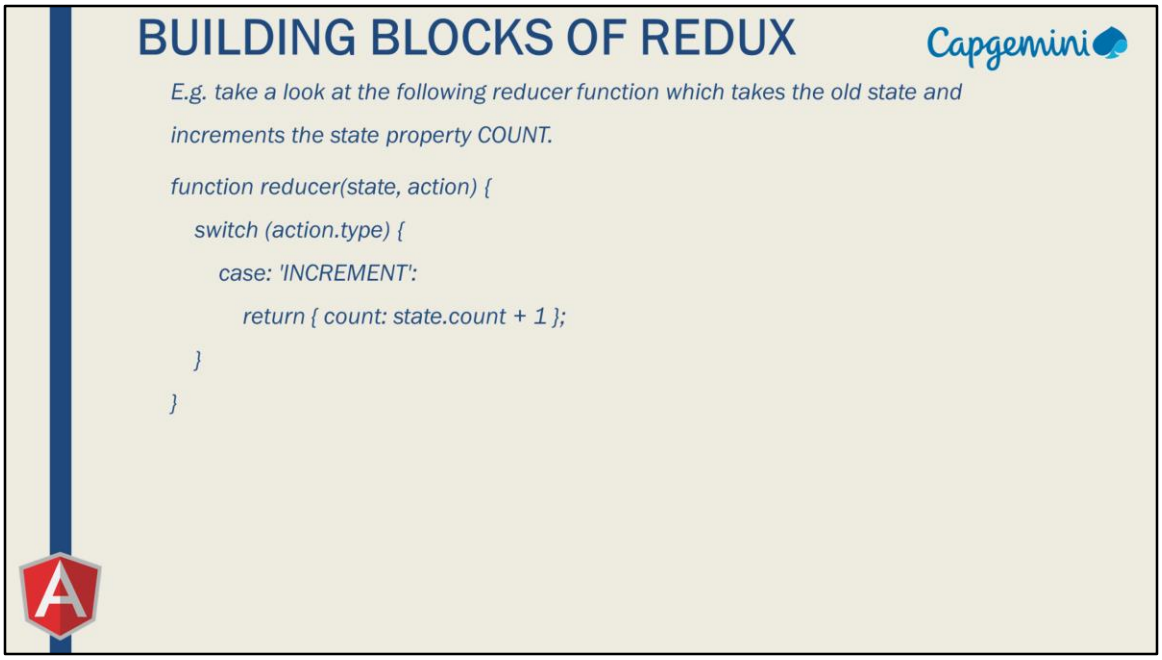
The shortcomings are as follows:
Usage of input properties for passing data down the component tree: In order to pass from one component to another, we've to use input properties to pass data down the component tree. This means that data needs to be passed to components in between as well. These components do not make use of this data, so this approach is inefficient.
Defining lots of input properties to pass data between components, makes components inflexible. Because components rely on those input properties, they cannot be reused somewhere else.
If the state is changed within one component you need to notify all other components which makes use of the same data manually.
All these shortcomings can lead to a complex application architecture and lead to inconsistent state information accross your components.



The Redux website can be found at http://redux.js.org/. Redux can be used with any modern JavaScript-based web frameworks. Before starting to build our Angular Redux sample application let's first clarify the core concept of Redux.



The Redux website can be found at http://redux.js.org/. Redux can be used with any modern JavaScript-based web frameworks. Before starting to build our Angular Redux sample application let's first clarify the core concept of Redux.



The Redux website can be found at http://redux.js.org/. Redux can be used with any modern JavaScript-based web frameworks. Before starting to build our Angular Redux sample application let's first clarify the core concept of Redux.

SETTING UP A NEW ANGULAR PROJECT WITH ANGULAR CLI

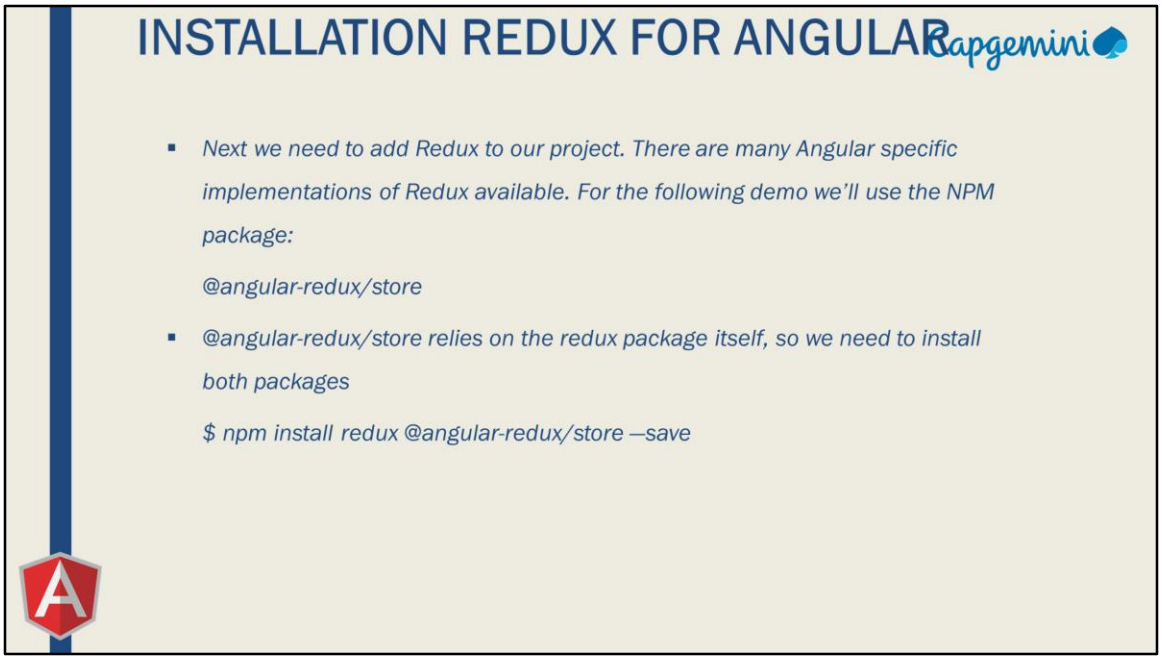
Capgemini

- *Now that you have a basic understanding of Redux, let's start with the second part of this tutorial and create a new Angular 4 application with Redux from scratch.*
- *To initiate a new Angular 4 project we can use Angular CLI:*

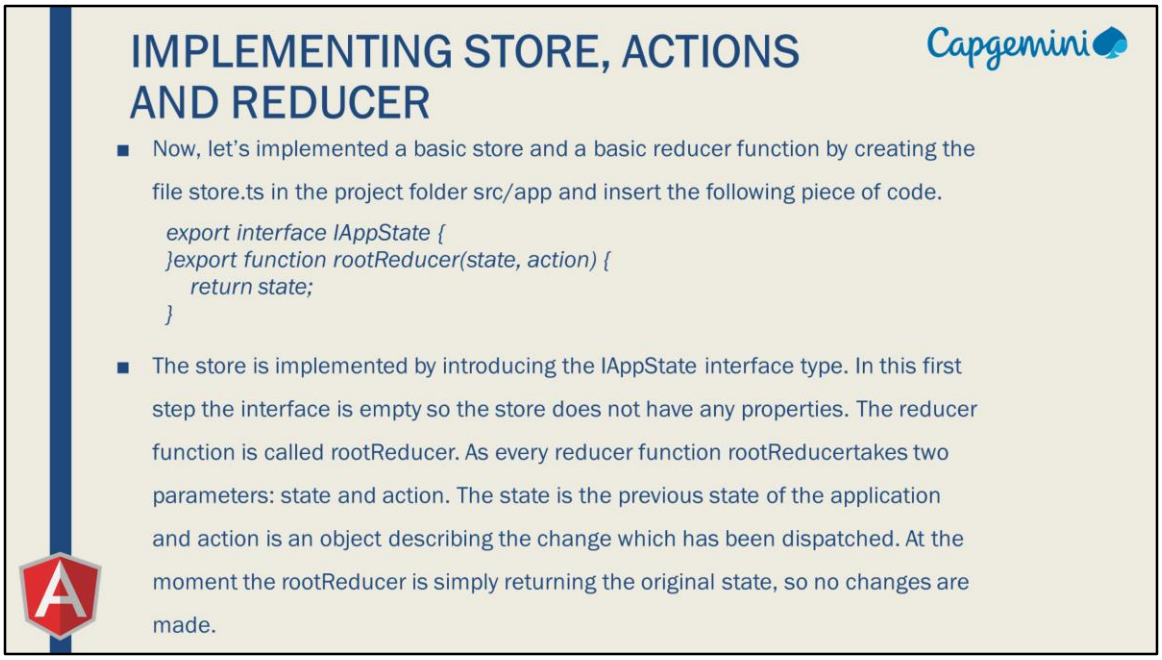
  $ ng new angularedux-todo
- *This creates a new folder angularredux-todo and within that folder you can find the initial Angular starter project. The live-reloading development web server is started by using the following command in this directory:*

  $ ng serve

The Redux website can be found at http://redux.js.org/. Redux can be used with any modern JavaScript-based web frameworks. Before starting to build our Angular Redux sample application let's first clarify the core concept of Redux.



The Redux website can be found at http://redux.js.org/. Redux can be used with any modern JavaScript-based web frameworks. Before starting to build our Angular Redux sample application let's first clarify the core concept of Redux.



The Redux website can be found at http://redux.js.org/. Redux can be used with any modern JavaScript-based web frameworks. Before starting to build our Angular Redux sample application let's first clarify the core concept of Redux.

IMPLEMENTING STORE, ACTIONS AND REDUCER

Capgemini

- Now let's add the properties to the IAppState interface:

```
export interface IAppState {
   todos: ITodo[];
   lastUpdate: Date;
}
export const INITIAL_STATE: IAppState = {
   todos: [],
   lastUpdate: null
}
```

- Here you can see that two properties are defined:
  - *todos as an array of type ITodo to contain all of our todo items*
  - *lastUpdate as Date type to contain the information when the todos array has been updated*

At the same time we're defining the INITIAL_STATE object of type IAppState. INITIAL_STATE is implementing the interface IAppState and initializing the properties todos with an empty array and lastUpdate with null. INITIAL_STATE will be used later on, when setting up the store of our application.

IMPLEMENTING STORE, ACTIONS AND REDUCER

Capgemini

- Next we need to implement the ITodo interface. Create the new file src/app/todo.ts and insert the following implementation:

```
export interface ITodo {
    id: number;
    description: string;
    responsible: string;
    priority: string;
    isCompleted: boolean;
}
```

- Add the following import statement to STORE.TS:

```
import { ITodo } from './todo';
```

At the same time we're defining the INITIAL_STATE object of type IAppState. INITIAL_STATE is implementing the interface IAppState and initializing the properties todos with an empty array and lastUpdate with null. INITIAL_STATE will be used later on, when setting up the store of our application.

ACTIVATING THE APPLICATION STORE

Capgemini

- Now let's activate the store for our application. First add the following import statement to the top of app.module.ts:
  import { NgRedux, NgReduxModule } from '@angular-redux/store';
- Next, add NgReduxModule to the imports array of @NgModule as well.
- We need to add one further import statement to
  import IAppState, rootReducerand INITIAL_STATE from store.ts:
  import { IAppState, rootReducer, INITIAL_STATE } from './store';
- The activation of the store is done by adding a constructor to the AppModuleclass, injecting NgRedux into that constructor and then calling the configureStore method of the NgRedux service:

```
export class AppModule {
   constructor (ngRedux: NgRedux<IAppState>) {
      ngRedux.configureStore(rootReducer, INITIAL_STATE);
   }
}
```

- The configureStore method takes two parameter. As the first parameter we're passing in our reducer function rootReducer. The second parameter is an object containing the initial state of the store. In our case we've defined INITIAL_STATE already so that we can pass in that object here.

DEFINING ACTION TYPES

Capgemini

- The reducer function should be able to handle all action types which are used in our application. Each action type is identified by a string. Create a new file src/app/actions.ts and define the following four action types:

```
export const ADD_TODO = 'ADD_TODO';
export const TOGGLE_TODO = 'TOGGLE_TODO';
export const REMOVE_TODO = 'REMOVE_TODO';
export const REMOVE_ALL_TODOS = 'REMOVE_ALL_TODOS';
```

Capgemini

USING ACTION TYPES IN THE REDUCER

- Having defined action type constants makes it easier to deal with action types in the reducer function. In store.ts add the following import statement first:

  import { ADD_TODO, TOGGLE_TODO, REMOVE_TODO, REMOVE_ALL_TODOS } from './actions';

- Finish the implementation of the the reducer function in STORE.TS and make use of the action types as you can see in the following: (Refer the code in Notes section)

```
export function rootReducer(state: IAppState, action): IAppState {
   switch (action.type) {
      case ADD_TODO:
         action.todo.id = state.todos.length + 1;
         return Object.assign({}, state, {
            todos: state.todos.concat(Object.assign({}, action.todo)),
            lastUpdate: new Date()
         })
      case TOGGLE_TODO:
         var todo = state.todos.find(t => t.id === action.id);
         var index = state.todos.indexOf(todo);
         return Object.assign({}, state, {
            todos: [
               ...state.todos.slice(0, index),
               Object.assign({}, todo, {isCompleted: !todo.isCompleted}),
               ...state.todos.slice(index+1)
            ],
            lastUpdate: new Date()
         })
```

```
    case REMOVE_TODO:
      return Object.assign({}, state, {
        todos: state.todos.filter(t => t.id !== action.id),
        lastUpdate: new Date()
      })
    case REMOVE_ALL_TODOS:
      return Object.assign({}, state, {
        todos: [],
        lastUpdate: new Date()
      })
  }
  return state;
}
```

Capgemini

USING ACTION TYPES IN THE REDUCER

- Let's explore the reducer function step by step.
  - *First you may notice that a switch statement has been added for action.type. Action type contains the action string, so that the case statements can make use of the previously defined action constants:*
  - *ADD_TODO: The ADD_TODO case uses the new todo object which is available in action.todo and creates a new state object in which the todos array is extended with that new todo element. To create a new state object the Object.assign method is used.*
  - *TOGGLE_TODO: The TOGGLE_TODO action is dispatched if the user wants to complete / uncomplete a todo entry. In that case the isCompletedproperty of the current todo element must be changed to the opposite. This means that a new state objects needs to be created and returned which contains this new value. The Object.assign method is used once again to compile this new state object.*
  - *REMOVE_TODO: With REMOVE_TODO an action is handled which is returning a new state where a specific todo entry has been removed from the previous state's todos array. REMOVE_ALL_TODOS: This actions returns a new state objects where the todos property is set to an empty array, so that all todo items are removed from the application state.*

```
export function rootReducer(state: IAppState, action): IAppState {
  switch (action.type) {
    case ADD_TODO:
      action.todo.id = state.todos.length + 1;
      return Object.assign({}, state, {
        todos: state.todos.concat(Object.assign({}, action.todo)),
        lastUpdate: new Date()
      })
    case TOGGLE_TODO:
      var todo = state.todos.find(t => t.id === action.id);
      var index = state.todos.indexOf(todo);
      return Object.assign({}, state, {
        todos: [
          ...state.todos.slice(0, index),
          Object.assign({}, todo, {isCompleted: !todo.isCompleted}),
          ...state.todos.slice(index+1)
        ],
        lastUpdate: new Date()
      })
```

```
        case REMOVE_TODO:
          return Object.assign({}, state, {
            todos: state.todos.filter(t => t.id !== action.id),
            lastUpdate: new Date()
          })
        case REMOVE_ALL_TODOS:
          return Object.assign({}, state, {
            todos: [],
            lastUpdate: new Date()
          })
      }
      return state;
}
```

IMPLEMENTING TODOOVERVIEWCOMPONENT

Capgemini

- Now that our application has implemented all relevant Redux building blocks we're ready to implement the TodoOverviewComponent and the TodoListComponent. First let's start with the Todo-Overview component. Use the following Angular CLI command to add that new component to the project:

  $ ng g component todo-overview

- The four new files are added to src/app/todo-overview:
  - *todo-overview.component.css*
  - *todo-overview.component.html*
  - *todo-overview.component.spec.ts*
  - *todo-overview.component.ts*
- Open up file todo-overview.component.ts and change the default implementation to:
- Refer the code in notes page

```
import { Component, OnInit } from '@angular/core';
import { NgRedux, select } from '@angular-redux/store';
import { IAppState } from '../store';
import { REMOVE_ALL_TODOS } from '../actions';
@Component({
  selector: 'app-todo-overview',
  templateUrl: './todo-overview.component.html',
  styleUrls: ['./todo-overview.component.css']
})
export class TodoOverviewComponent implements OnInit {
  @select() todos;
  @select() lastUpdate;
  constructor(private ngRedux: NgRedux<IAppState>) { }
  ngOnInit() {
  }
  clearTodos() {
    this.ngRedux.dispatch({type: REMOVE_ALL_TODOS});
  }
}
```

IMPLEMENTING TODOOVERVIEWCOMPONENT

- We're adding a few import statements on top of the file. NgRedux and select is imported from the @angular-redux/store package. IAppState is imported from store.ts and the action type REMOVE_ALL_TODOS is imported from actions.ts.
- Using dependency injection again the NgRedux<IAppState> service is injected into the class. The clearTodos method is implemented to dispatch the REMOVE_ALL_TODOS action type to the store. Dispatching is done by using the dispatch method of the NgRedux service.
- Furthermore we need to access the state properties todos and lastUpdate. To define class properties which gives you access to the store properties we need to use the @select decorator.
- Now we're ready to implement the corresponding template in todo-overview.component.html:

```
<p class="text-right"><span class="badge badge-secondary">Last Update: {{ (lastUpdate | async)
| date:'mediumTime' }} | Total items: {{ (todos | async).length }}</span></p>
<button class="btn btn-primary" (click)="clearTodos()">Delete All</button>
```

- Please note, that you need to use the async pipe to include store properties as an expression statement in your template code.

```
import { Component, OnInit } from '@angular/core';
import { NgRedux, select } from '@angular-redux/store';
import { IAppState } from '../store';
import { REMOVE_ALL_TODOS } from '../actions';
@Component({
  selector: 'app-todo-overview',
  templateUrl: './todo-overview.component.html',
  styleUrls: ['./todo-overview.component.css']
})
export class TodoOverviewComponent implements OnInit {
  @select() todos;
  @select() lastUpdate;
  constructor(private ngRedux: NgRedux<IAppState>) { }
  ngOnInit() {
  }
  clearTodos() {
    this.ngRedux.dispatch({type: REMOVE_ALL_TODOS});
  }
}
```

IMPLEMENTING TODOLISTCOMPONENT

- Our second component is the TODOSLISTCOMPONENT and is added to the project by using the following command:
  $ ng g component todo-list
- As we need to implement the todo entry form as part of this component we need to add the Angular FormsModule to our application. Add the following import statement to file app.module.ts:
  import { FormsModule } from '@angular/forms';
- Add the FormsModule to the imports array as well.
- Next, let's change the default implementation of the component class in todos-list.component.ts to what you can see in the following code listing: (Refer the code in Notes section)

```
import { Component, OnInit } from '@angular/core';
import { NgRedux, select } from '@angular-redux/store';
import { IAppState } from '../store';
import { ADD_TODO, REMOVE_TODO, TOGGLE_TODO } from '../actions';
import { ITodo } from '../todo';
@Component({
  selector: 'app-todo-list',
  templateUrl: './todo-list.component.html',
  styleUrls: ['./todo-list.component.css']
})
export class TodoListComponent implements OnInit {
  @select() todos;
  model: ITodo = {
    id: 0,
    description: "",
    responsible: "",
    priority: "low",
    isCompleted: false
  };
```

```
  constructor(private ngRedux: NgRedux<IAppState>) { }
  ngOnInit() {
  }
  onSubmit() {
    this.ngRedux.dispatch({type: ADD_TODO, todo: this.model});
  }
  toggleTodo(todo) {
    this.ngRedux.dispatch({ type: TOGGLE_TODO, id: todo.id });
  }
  removeTodo(todo) {
    this.ngRedux.dispatch({type: REMOVE_TODO, id: todo.id });
  }
}
```

IMPLEMENTING TODOLISTCOMPONENT

- Now let's take a look at the template implementation in file todo-list.component.html:
  - *Refer the code in Notes session*
- This template contains the implementation of an Angular template-driven form which lets the user input new todo items. The form submit event is connected to the onSubmit event handler method, so that the ADD_TODOaction type is dispatched to the store whenever a user submits the form.
- The table output of the todos array is done by using the NgFor directive in the following form:
  - *<tr *ngFor="let t of todos | async">*
- Again, it's important to use the async pipe again to retrieve data from the store for usage in the template.
- To complete a todo item in the table the user should be able to simply click on the element (either on ID, description or responsible). Because of that we're connecting the toggleTodo event handler method with the click event of the elements which contains the text information in the table.
- Finally the user should be able to delete a single todo item from the list. To do so a button is included for each row and the click event of that button is connected to the removeTodo event handler method.

```
<h6>Create Todo:</h6>
<form (ngSubmit)="onSubmit()" #todoForm="ngForm">
 <div class="form-row">
  <div class="col-auto">
    <input
     type="text"
     class="form-control"
     placeholder="Description"
     id="description"
     [(ngModel)]="model.description"
     name="description"
     #description="ngModel">
  </div>
  <div class="col-auto">
   <input
    type="text"
    class="form-control"
    placeholder="Responsible"
    id="responsible"
```

```
      [(ngModel)]="model.responsible"
      name="responsible"
      #responsible="ngModel">
    </div>
    <div class="col-auto">
     <select
      class="form-control"
      id="priority"
      [(ngModel)]="model.priority"
      name="priority"
      #priority="ngModel">
      <option value="low">Low</option>
      <option value="medium">Medium</option>
      <option value="high">High</option>
     </select>
    </div>
    <div class="col-auto">
     <button type="submit" class="btn btn-primary">Create</button>
    </div>
  </div>
</form>
<br />
<h6>Todos List:</h6>
<div *ngIf="(todos | async)?.length!=0">
<table class="table">
  <thead class="thead-inverse">
    <tr>
      <th>#</th>
      <th>Todo Description</th>
      <th>Responsible</th>
      <th>Priority</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let t of todos | async">
      <td><span (click)="toggleTodo(t)" [class.completed]="t.isCompleted">{{ t.id
}}</span></td>
      <td><span (click)="toggleTodo(t)" [class.completed]="t.isCompleted">{{
t.description }}</span></td>
      <td><span (click)="toggleTodo(t)" [class.completed]="t.isCompleted">{{
t.responsible }}</span></td>
```

```
    <td>
      <span *ngIf="t.priority == 'low'" class="badge badge-success">Low</span>
      <span *ngIf="t.priority == 'medium'" class="badge badge-
warning">Medium</span>
      <span *ngIf="t.priority == 'high'" class="badge badge-danger">High</span>
    </td>
    <td><button class="btn btn-primary"
(click)="removeTodo(t)">Delete</button></td>
   </tr>
  </tbody>
</table>
</div>
```

IMPLEMENTING TODOLISTCOMPONENT

Capgemini

- Last but not least you can find the CSS code for class completed in the following listing. This code needs to be inserted into file todo-list.component.css:

```
.completed {
   text-decoration: line-through;
 }
```

- By adding the following attribute to the elements which contains the text values of a todo item we're applying that class only of the isCompleted property is set.

```
[class.completed]="t.isCompleted"
```

- In that case the text is crossed out, so that the user can see that this item is completed.

IMPLEMENTING APPCOMPONENT

Capgemini

- Finally we need to include both components in AppComponent by using the elements <app-todo-overview> and <app-todo-list>.

```
<div class="container" id="app">
  <br/>
    <a href="http://codingthesmartway.com/" target="_blank"><img
src="assets/images/logo.png" width="300" /></a>
  <hr>
  <div>
     <div class="card">
        <div class="card-body">
          <h3 class="card-title">Todos App</h3>
          <h6 class="card-subtitle mb-2 text-muted">Using Angular & Redux</h6>
          <app-todo-overview></app-todo-overview>
          <app-todo-list></app-todo-list>
        </div>
      </div>
  </div>
</div>
```