Angular Components

Goal

In this lab, you will:

- Display a list of movies or the selected movie details
- Use Bootstrap and some custom CSS to style the application
- Update the movies details

Your mission

This is the lab you will update a movie list applications. The application you start with contains the movie data and displays a simple list of movie titles. You will enhance this by adding Bootstrap and using it to display more movie details. There will also be a button to switch to a details view of a movie that lets you update movie details.

Type it out by hand?

> Typing it drills it into your brain much better than simply copying and pasting it. You're forming new neuron pathways. Those pathways are going to help you in the future. Help them out now.

Create a list of movies

In this section, you will display a list of movies

1. Open the **begin/MoviesApp** folder of the lab in the terminal window and start the application using the following command.

```
ng serve
```

2. Open the application in the browser at http://localhost:4200/.

3. Add a new **movies** module.

```
ng generate module movies
```

4. Add two new components **MovieList** and **MovieListItem** to **MoviesModule**.

```
cd src/app/movies
ng generate component MovieList
ng generate component MovieListItem
```

5. Open **movies.module.ts** and export **MovieListComponent** from the module.

```
@NgModule({
  imports: [
    CommonModule
  ],
  declarations: [
    MovieListComponent,
    MovieListItemComponent
  ],
  exports:[
    MovieListComponent
  ]
})
export class MoviesModule { }
```

6. Open **app.module.ts** and add the **MoviesModule** to imports of **AppModule**. You can remove the **FormsModule** and **HttpModule** as they are not needed in the **AppModule**.

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

import { AppComponent } from './app.component';
import { MoviesModule } from './movies/movies.module';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    MoviesModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

7. Render the **MovieListComponent** from the **AppComponent**. Also update **AppComponent title** property to "Popular movies".

```
<h1>
  {{title}}
</h1>

<app-movie-list></app-movie-list>
```

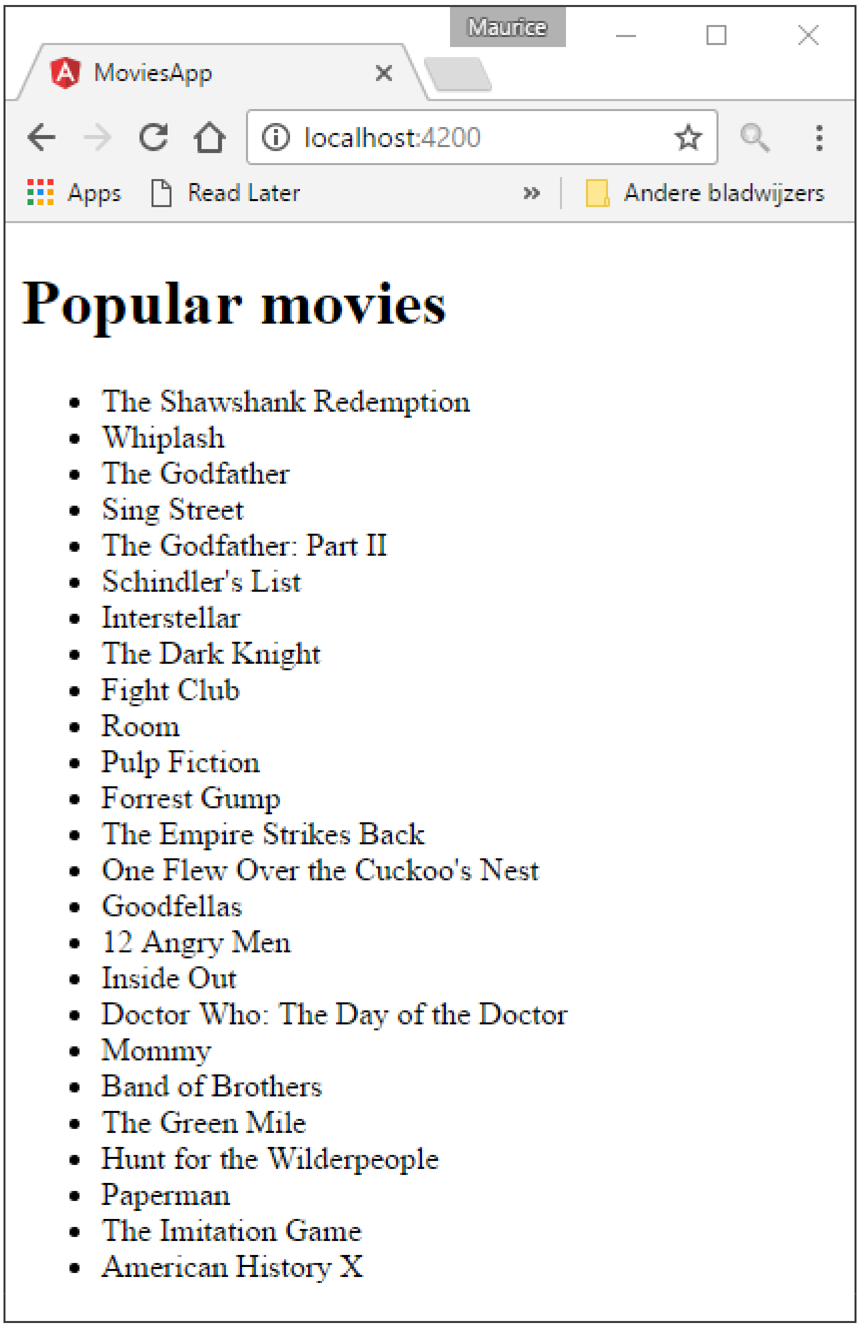
8. Expose the movies collection from the **MovieListComponent**.

```
import { Component } from '@angular/core';
import movies from '../../../../../movies';

@Component({
  selector: 'app-movie-list',
  templateUrl: './movie-list.component.html',
  styleUrls: ['./movie-list.component.css']
})
export class MovieListComponent {

  private movies = movies;
}
```

9. Add the list of movies to the **MovieListComponent** using HTML `<ul>` and `<li>` elements.

```
<ul>
  <li *ngFor="let movie of movies">
    {{movie.title}}
  </li>
</ul>
```

10. The application in the browser should now look like this.



Popular movies

- The Shawshank Redemption
- Whiplash
- The Godfather
- Sing Street
- The Godfather: Part II
- Schindler's List
- Interstellar
- The Dark Knight
- Fight Club
- Room
- Pulp Fiction
- Forrest Gump
- The Empire Strikes Back
- One Flew Over the Cuckoo's Nest
- Goodfellas
- 12 Angry Men
- Inside Out
- Doctor Who: The Day of the Doctor
- Mommy
- Band of Brothers
- The Green Mile
- Hunt for the Wilderpeople
- Paperman
- The Imitation Game
- American History X

Add Bootstrap to the project

In this section, you will use add the Bootstrap CSS framework to the project and add it to the rendered page.

1. Add the NPM package for the Bootstrap CSS project.

```
npm install bootstrap --save
```

2. Open **angular-cli.json** and add **bootstrap.min.css** to the styles section. This will ensure that the **bootstrap.min.css** is included in the **index.html** that is generated by Webpack.

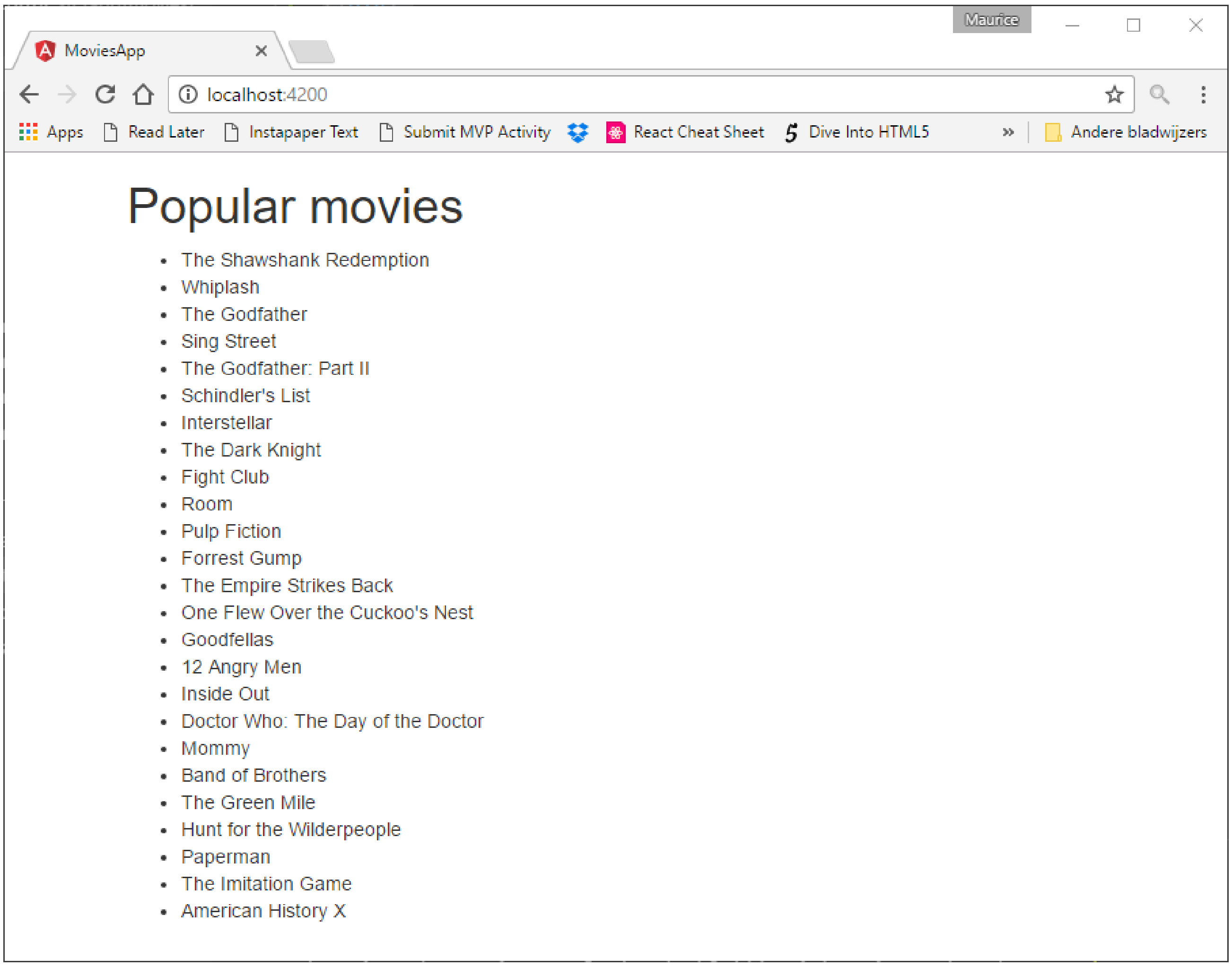
> Note: You will need to restart **ng serve** for this change to take effect.

```
"styles": [
    "../node_modules/bootstrap/dist/css/bootstrap.min.css",
    "styles.css"
],
```

3. Open **app.component.html** and wrap the markup in a `<div>` element with the **container** CSS class.

```
<div class="container">
  <h1>
    {{title}}
  </h1>

  <app-movie-list></app-movie-list>
</div>
```

4. Check the browser. The list of movies should be displayed with a different font and somewhat offset from the page border.



Popular movies

- The Shawshank Redemption
- Whiplash
- The Godfather
- Sing Street
- The Godfather: Part II
- Schindler's List
- Interstellar
- The Dark Knight
- Fight Club
- Room
- Pulp Fiction
- Forrest Gump
- The Empire Strikes Back
- One Flew Over the Cuckoo's Nest
- Goodfellas
- 12 Angry Men
- Inside Out
- Doctor Who: The Day of the Doctor
- Mommy
- Band of Brothers
- The Green Mile
- Hunt for the Wilderpeople
- Paperman
- The Imitation Game
- American History X

Create a Bootstrap styles movie list

In this section, you replace the simple list of movie titles by a list that is better looking.

1. Open the **movie-list-item.component.ts** and add an input property named **movie**.

```
import { Component, OnInit, Input } from '@angular/core';

@Component({
  selector: 'app-movie-list-item',
  templateUrl: './movie-list-item.component.html',
  styleUrls: ['./movie-list-item.component.css']
})
export class MovieListItemComponent implements OnInit {

  @Input() movie;

  constructor() { }

  ngOnInit() {
  }
}
```

2. Open **movie-list.component.html** and render the **MovieListItem** for each movie.

```
<h2>
  Movie list
</h2>
<app-movie-list-item *ngFor="let m of movies"
                     [movie]="m">
</app-movie-list-item>
```

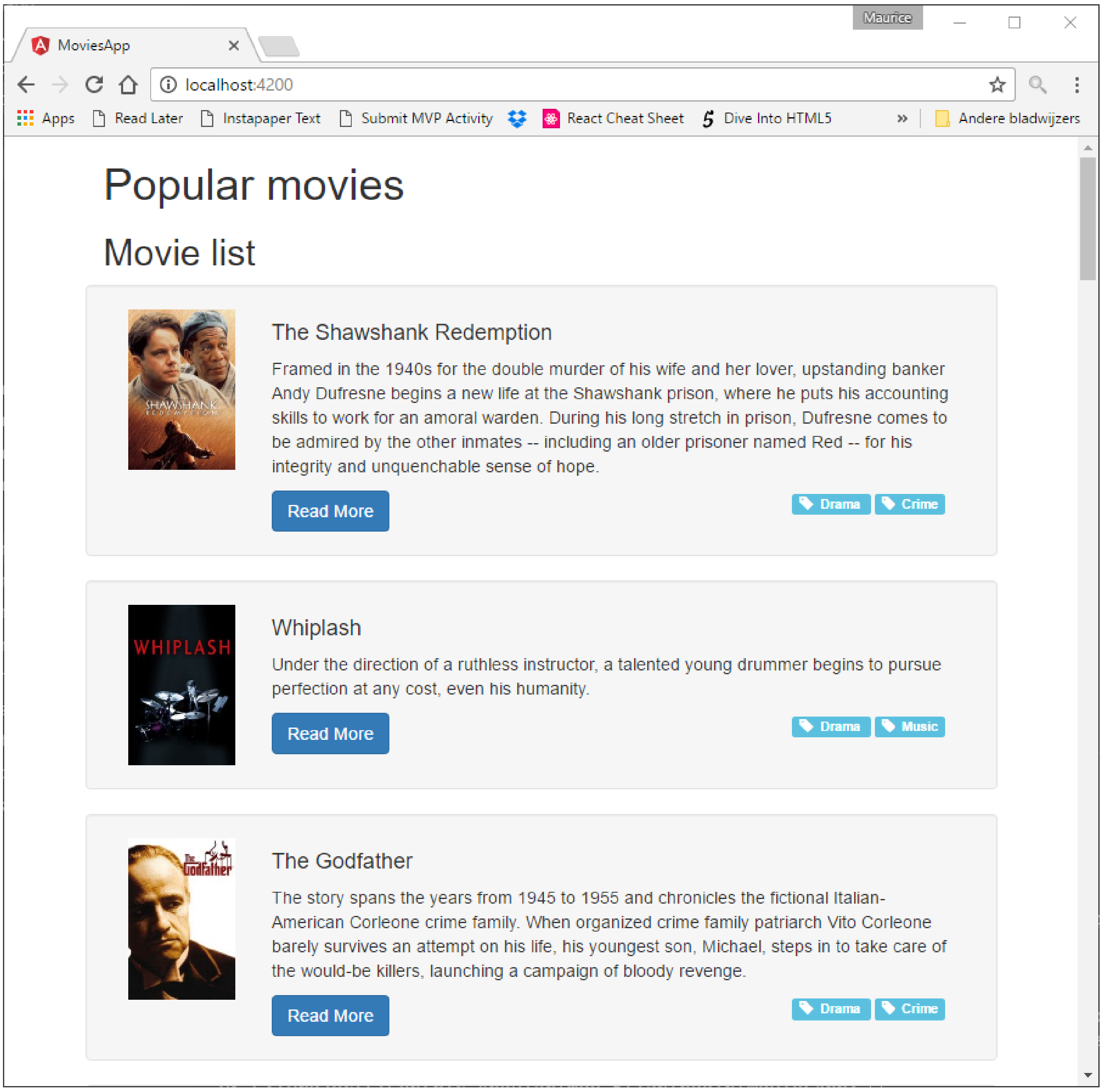
3. Open **movie-list-item.component.html** and add the following Bootstrap styled markup to display the movie details.

```
<div class="row well">
  <div class="col-sm-2">
    <img class="img-responsive" src="http://image.tmdb.org/t/
p/w154{{movie.poster_path}}">
  </div>
  <div class="col-sm-10">
    <h4>{{movie.title}}</h4>
    <p>{{movie.overview}}</p>
    <span class="pull-left">
      <button class="btn btn-primary">
        Read More
      </button>
    </span>
    <span class="pull-right">
      <span *ngFor="let genre of movie.genres" class="label la
bel-info genre">
        <span class="glyphicon glyphicon-tag"></span>
        {{genre}}
      </span>
    </span>
  </div>
  <div class="clearfix"></div>
</div>
```

4. Open movie-list-item.component.css and add a two CSS classes to add a bit of spacing to the genre tags. Note that these styles are scoped to the MovieListItem component so updating the style for the standard Bootstrap glyphicon-tag is not a global change.

```
.genre, .glyphicon-tag {
  margin-right: 3px;
}
```

5. The application in the browser should now look like this. Note that the Read More button isn't functional yet.



Popular movies

Movie list

The Shawshank Redemption

Framed in the 1940s for the double murder of his wife and her lover, upstanding banker Andy Dufresne begins a new life at the Shawshank prison, where he puts his accounting skills to work for an amoral warden. During his long stretch in prison, Dufresne comes to be admired by the other inmates -- including an older prisoner named Red -- for his integrity and unquenchable sense of hope.

Read More | Drama | Crime

Whiplash

Under the direction of a ruthless instructor, a talented young drummer begins to pursue perfection at any cost, even his humanity.

Read More | Drama | Music

The Godfather

The story spans the years from 1945 to 1955 and chronicles the fictional Italian-American Corleone crime family. When organized crime family patriarch Vito Corleone barely survives an attempt on his life, his youngest son, Michael, steps in to take care of the would-be killers, launching a campaign of bloody revenge.

Read More | Drama | Crime

Updating movie details

In this section, you will add a movie editor allowing you to update a selected movie.

Edit movie details

1. Add a new **MovieEditorComponent** to **MoviesModule** using the Angular CLI.

```
cd src/app/movies
ng generate component MovieEditor
```

2. Open **movies.module.ts** and export **MovieEditorComponent** and import **FormsModule** to the **MovieModule**.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule }  from '@angular/forms';
import { MovieListComponent } from './movie-list/movie-list.co
mponent';
import { MovieListItemComponent } from './movie-list-item/movi
e-list-item.component';
import { MovieEditorComponent } from './movie-editor/movie-edi
tor.component';

@NgModule({
  imports: [
    CommonModule,
    FormsModule
  ],
  declarations: [
    MovieListComponent,
    MovieListItemComponent, MovieEditorComponent
  ],
  exports:[
    MovieListComponent,
    MovieEditorComponent
  ]
})
export class MoviesModule { }
```

3. Open **movie-editor.component.ts** and add an input property named **movie**.

```
import { Component, OnInit, Input } from '@angular/core';

@Component({
  selector: 'app-movie-editor',
  templateUrl: './movie-editor.component.html',
  styleUrls: ['./movie-editor.component.css']
})
export class MovieEditorComponent implements OnInit {

  @Input() movie;

  constructor() { }
  ngOnInit() {
  }
}
```

4. Open **movie-list-item.component.ts** and add an **movieSelected** output property and a **showMore()** function. When the **showMore()** function is called, the **movieSelected** property should emit the current movie as the new selected movie.

```
import { Component, Input, Output, EventEmitter } from '@angul
ar/core';

@Component({
  selector: 'app-movie-list-item',
  templateUrl: './movie-list-item.component.html',
  styleUrls: ['./movie-list-item.component.css']
})
export class MovieListItemComponent {

  @Input() movie;
  @Output() movieSelected = new EventEmitter();

  showMore() {
    this.movieSelected.emit(this.movie);
  }
}
```

5. Open **movie-list-item.component.html** and add a click handler to the Read More button. Call the components **showMore()** function when the button is clicked.

```
<button class="btn btn-primary" (click)="showMore()">
  Read More
</button>
```

6. Open **movie-list.component.ts** and add a **movieSelected** output property and an **onMovieSelected()** function to propagate the event.

```
export class MovieListComponent {

  @Output() movieSelected = new EventEmitter();
  private movies = movies;

  onMovieSelected(movie) {
    this.movieSelected.emit(movie);
  }
}
```

7. Open **movie-list.component.html** and wire up the event from the **MovieListItem** component.

```
<app-movie-list-item *ngFor="let m of movies"
                     [movie]="m"

(movieSelected)="onMovieSelected($event)">
</app-movie-list-item>
```

8. Open **app.component.ts** and add a function named **movieSelected()** to handle the **movieSelected** event. Save the currently selected movie in a property named **selectedMovie**.

```
selectedMovie = null;

movieSelected(movie) {
  this.selectedMovie = movie;
}
```

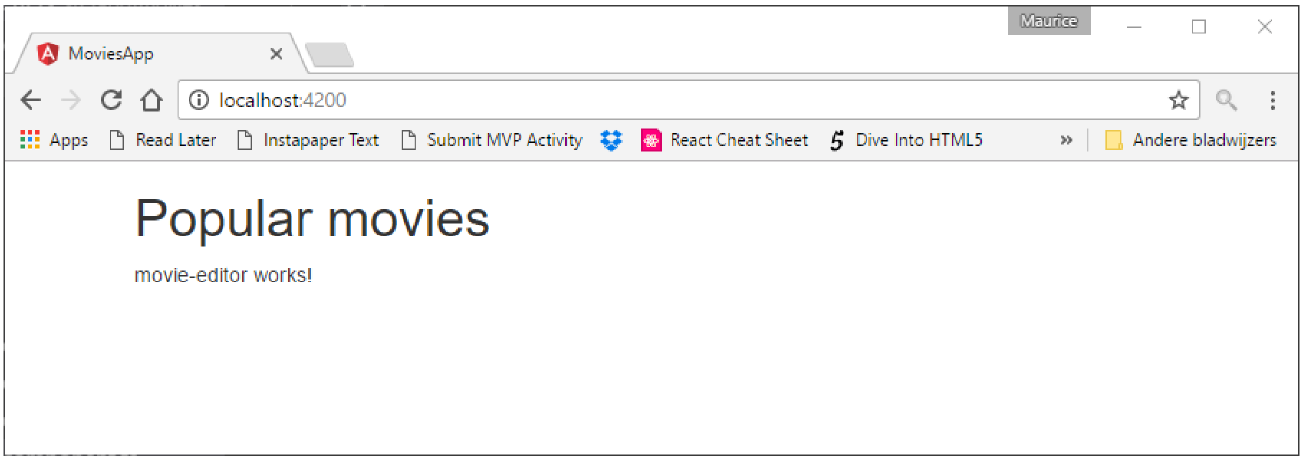
9. Open **movie-editor.component.ts** and add a **movie** input property.

```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-movie-editor',
  templateUrl: './movie-editor.component.html',
  styleUrls: ['./movie-editor.component.css']
})
export class MovieEditorComponent {

  @Input() movie;

}
```

10. Open **app.component.html** and update the markup to display the movie list if there is no selected movie. Display the **MovieEditor** component if there is a **selectedMovie**. Pass the **selectedMovie** as the **movie** property to the **MovieEditor** component.

```
<div class="container">
  <h1>
    {{title}}
  </h1>

  <div *ngIf="!selectedMovie">
    <app-movie-list (movieSelected)="movieSelected($event)">
    </app-movie-list>
  </div>
  <div *ngIf="selectedMovie">
    <app-movie-editor [movie]="selectedMovie">
    </app-movie-editor>
  </div>
</div>
```

11. Reload the application in the browser. Clicking Read More on a movie should hide all movies and display the placeholder for the **MovieEditor** component.



Popular movies

movie-editor works!

12. Open the **movie-editor.component.html** and add an HTML `<form>` tag. The form should contain an HTML `<input>` tag for the movie title property and an HTML `<textarea>` for the overview property of the movie. Add an HTML `<button>` tag to submit the form. Make sure to use the appropriate Bootstrap styling for a vertical form.

```
<h2>
  Movie editor
```

```
</h2>

<div class="row well">
  <form>
    <div class="form-group">
      <label for="title">Title</label>
      <input
        type="text"
        [(ngModel)]="movie.title"
        class="form-control"
        id="title"
        name="title"
        placeholder="Title">
    </div>
    <div class="form-group">
      <label for="overview">Overview</label>
      <textarea
        [(ngModel)]="movie.overview"
        class="form-control"
        rows="10"
        id="overview"
        name="overview"
        placeholder="Overview">
      </textarea>
    </div>
    <div class="btn-group">
      <button type="submit" class="btn btn-primary">
        Save
      </button>
      <button class="btn btn-danger">
        Cancel
      </button>
    </div>
  </form>
</div>
```

13. Open **movie-editor.component.ts** and add an **onSubmit()** function and a **movieSaved** output property to the **MovieEditor** component. Emit a **movieSaved** event when the **onSubmit()** function is called.

```
export class MovieEditorComponent implements OnInit {
  // Previous code
  @Output() movieSaved = new EventEmitter();

  onSubmit() {
    this.movieSaved.emit();
  }
}
```

14. Open the **movie-editor.component.html** and call the **onSubmit()** function when the form is submit event is triggered.

```
<form (submit)="onSubmit()">
```

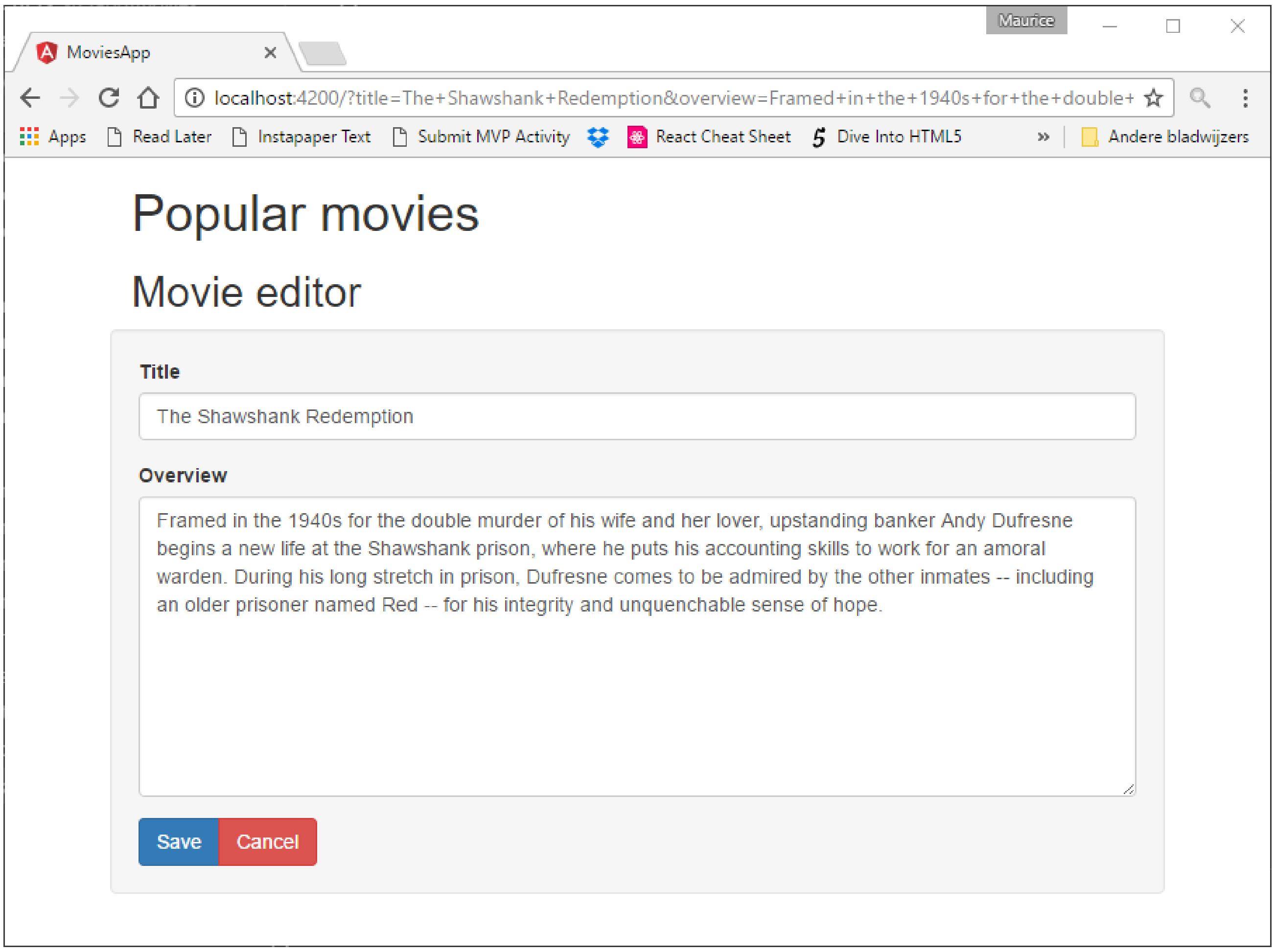
15. Open **app.component.ts** and add a **movieSaved()** function that is called after the **movieSaved** event is emitted. Set the **selectedMovie** property to null in this function so the movie list is displayed again.

```
movieSaved() {
  this.selectedMovie = null;
}
```

16. Open **app.component.html** and add an event handler to the **MovieEditor** component **movieSaved** event to call the **moviesSaved()** function on the component. Note you are not actually doing anything about the movie properties. This is because you are data binding to the same object used in the movie list. This means the list data is updated as soon as you make a change to the input form. These changes only become visible when you click Save and the movie list is displayed again.

```
<app-movie-editor [movie]="selectedMovie"
                  (movieSaved)="movieSaved()">
</app-movie-editor>
```

17. Open the application in the browser and make sure you can update movie details.



Popular movies

Movie editor

Title

The Shawshank Redemption

Overview

Framed in the 1940s for the double murder of his wife and her lover, upstanding banker Andy Dufresne begins a new life at the Shawshank prison, where he puts his accounting skills to work for an amoral warden. During his long stretch in prison, Dufresne comes to be admired by the other inmates -- including an older prisoner named Red -- for his integrity and unquenchable sense of hope.

Add a Cancel edit action

Currently, you can update movies but you can't abort changes after you made them. In this section, you will add a Cancel button and edit a copy of the movie so changes are not final until the Save button is clicked.

1. Open **movie-editor.component.ts** and add an **onCancel()** function and a **cancelEdit** output **EventEmitter** property. Make sure to prevent the default form action when the cancel button is clicked.

```
  @Output() cancelEdit = new EventEmitter();

  onCancel(e) {
    e.preventDefault();
    this.cancelEdit.emit();
  }
```

2. Open **movie-editor.component.html** and update the **Cancel** button. Call the **onCancel()** function when the button is clicked.

```
<button class="btn btn-danger" (click)="onCancel($event)">
  Cancel
```

```
</button>
```

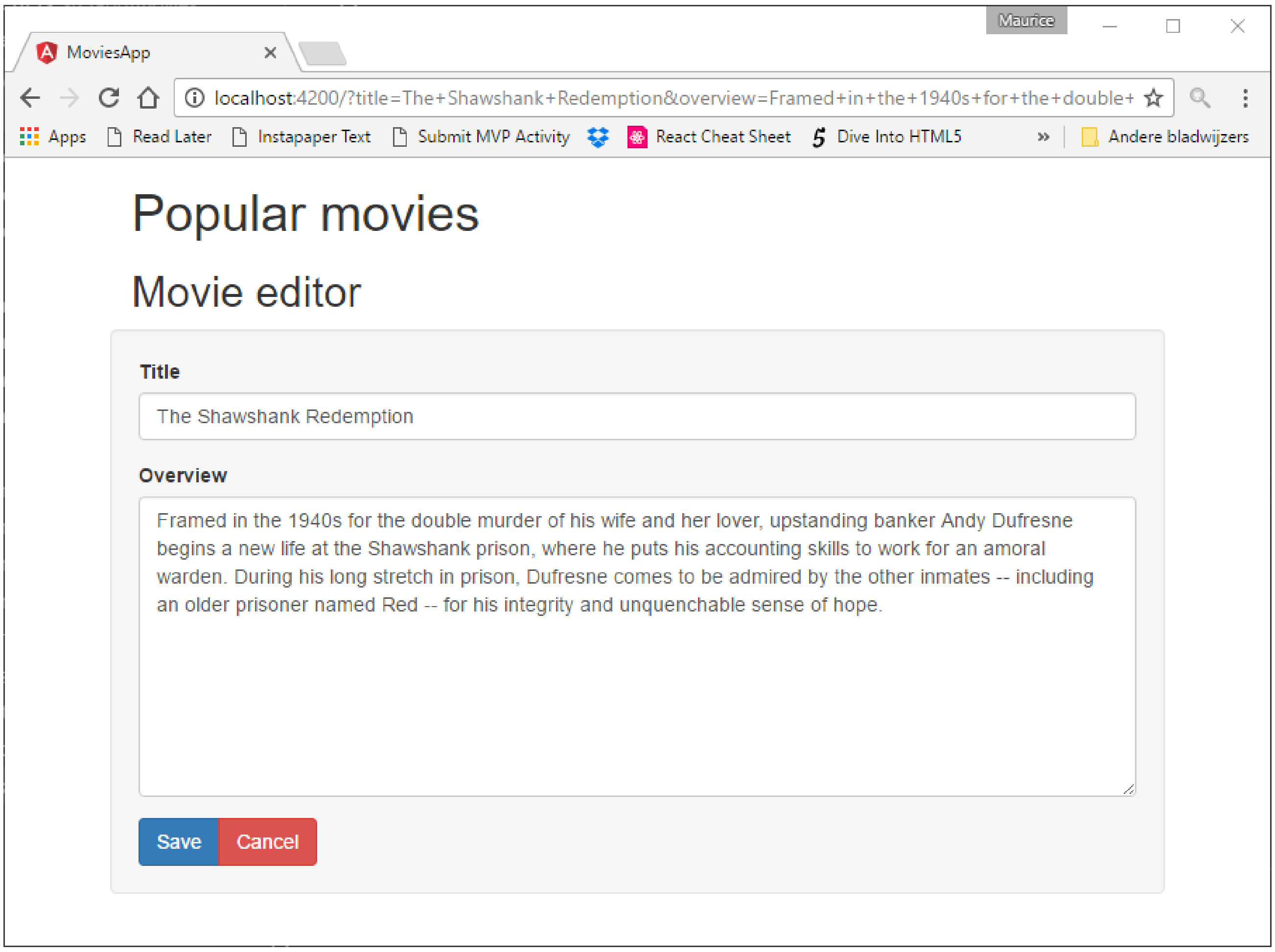
3. Open **app.component.ts** and add a **savedMovie** property. Next, update the **movieSelected()** to create a copy of the movie passed and **movieSaved()** to update the original movie object with the changes made to the copy. Additionally, add a **cancelEdit()** function to clear the **selectedMovie** and **savedMovie** properties.

```
export class AppComponent {
  movies = movies;
  selectedMovie = null;
  savedMovie = null;

  movieSelected(movie) {
    this.savedMovie = movie;
    this.selectedMovie = Object.assign({}, movie);
  }

  movieSaved() {
    Object.assign(this.savedMovie, this.selectedMovie);
    this.selectedMovie = null;
    this.savedMovie = null;
  }

  cancelEdit() {
    this.selectedMovie = null;
    this.savedMovie = null;
  }
}
```

4. Open **app.component.html** and wire up the **cancelEdit** output property of the **MovieEditor** component to call the **cancelEdit()** function.

```
<app-movie-editor [movie]="selectedMovie"
                  (movieSaved)="movieSaved()"
                  (cancelEdit)="cancelEdit()">
```

5. Test to make sure you can cancel changes made in the editor.



Popular movies

Movie editor

Title

The Shawshank Redemption

Overview

Framed in the 1940s for the double murder of his wife and her lover, upstanding banker Andy Dufresne begins a new life at the Shawshank prison, where he puts his accounting skills to work for an amoral warden. During his long stretch in prison, Dufresne comes to be admired by the other inmates -- including an older prisoner named Red -- for his integrity and unquenchable sense of hope.

Solution

The Solution can be found in **complete** folder