Angular HTTP Requests

Goal

In this lab, you will:

- Request the list of movies using an Ajax call
- Request an individual movie using an Ajax call
- Update an individual movie using an Ajax call

Your mission

In this lab, you will load the movie list using an Ajax call to the server. You will also fetch the individual movie the users want to edit from the server and update this using separate Ajax calls.

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.

Load the list of movies using an Ajax request

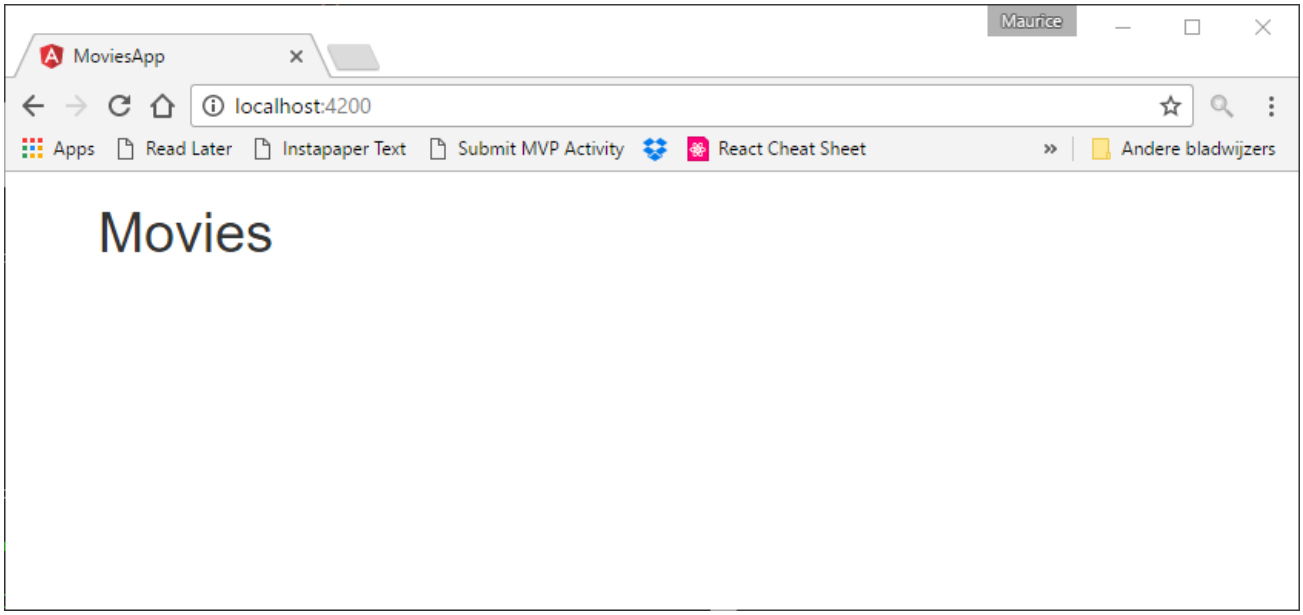
In this section, you will use the HTTP service to load all movies and display them as a list.

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

> Note: Make sure not to use **ng serve** to start the application because you will also use a movies API server in this lab.

```
npm start
```

2. Open the application in the browser at http://localhost:4200/. You should just see the title of the page.



3. Use the Angular CLI to create a Movie interface type.

```
ng generate interface Movie
```

4. Open **movies.ts** and add the properties you will be using. These are the **id** of type number and **title**, **overview** and **poster_path** of type string, and finally **genres** as type string array.

```
export interface Movie {
    id: number;
    title: string;
    overview: string;
    poster_path: string;
    genres: string[];
}
```

5. Use the Angular CLI to create a **MoviesService**.

```
ng generate service Movies
```

6. Open **movies.service.ts**. Inject the Angular **Http** service into the constructor and save this as a private property. Add a function named **getMovies()** and use it to execute an HTTP **get** request to the endpoint **/api/movies**. Define the result of the **getMovies()** function as **Observable<Movie[]>**. Add the rx **map** operator and map the HTTP response to the **json** payload in the **map()** function. Make sure the import all the required types.

```
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';
```

```
import { Observable } from 'rxjs/Observable';

import 'rxjs/add/operator/map';

import { Movie } from './movie';

@Injectable()
export class MoviesService {

  constructor(private http: Http) { }

  getMovies(): Observable<Movie[]>  {
    return this.http.get('/api/movies')
      .map(resp => resp.json());
  }
}
```

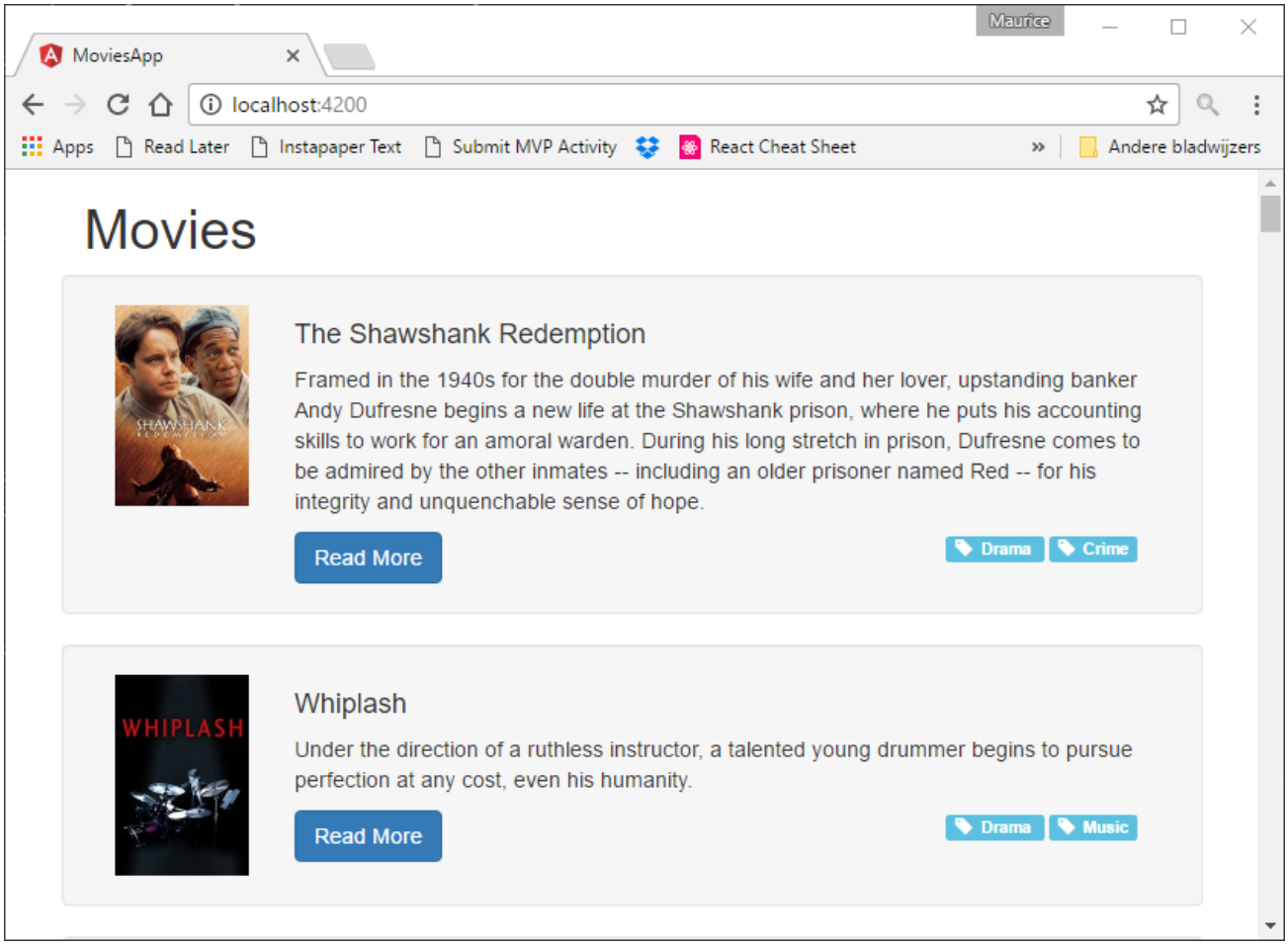
7. Open **app.module.ts** and add the Angular **HttpModule** to the **imports** section and the **MoviesService** you just created to the **providers** section.

```
import { HttpModule } from '@angular/http';
import { MoviesService } from './movies.service';

@NgModule({
  declarations: [
    // Existing components
  ],
  imports: [
    // Existing imports
    HttpModule
  ],
  providers: [MoviesService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

8. Open **movie-list-item.component.ts**. Add a **movies** property of type **Movie[]** and initialize it as an empty array. Import the **MoviesService** and inject this into the constructor saving it as a private property. Use the **ngOnInit()** lifecycle function to request the movies data using the **getMovies()** function. Update the **movies** property when the movie data is loaded.

```
import { Component, OnInit, Output, EventEmitter } from '@angu
lar/core';

import { MoviesService } from '../movies.service';
import { Movie } from '../movie';

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

  private movies: Movie[] = [];
  @Output() movieSelected = new EventEmitter<number>();

  constructor(private moviesService: MoviesService) { }

  ngOnInit() {
    this.moviesService.getMovies()
      .subscribe(movies => this.movies = movies);
  }

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

9. The application in the browser should now look like this:



Loading the movie details

In this section, you will load the individual movie when a user clicks Read More.

1. Open **movies.service.ts** and add a function **getMovie()** that takes a movie **id** as its parameter and returns an **Observable**. This function should do an Ajax request to the **/api/movies/${id}** endpoint to load an individual movie object.

```
getMovie(id: number): Observable<Movie> {
  return this.http.get(`/api/movies/${id}`)
    .map(resp => resp.json());
}
```

2. Open **movie-editor.component.ts** and add import statements for the **Movie** interface and the **MoviesService**.

```
import { MoviesService } from '../movies.service';
import { Movie } from '../movie';
```

3. Inject the **MoviesService** into the constructor saving it as a private property. Use the **MoviesService** in the **ngOnInit()** lifecycle function to

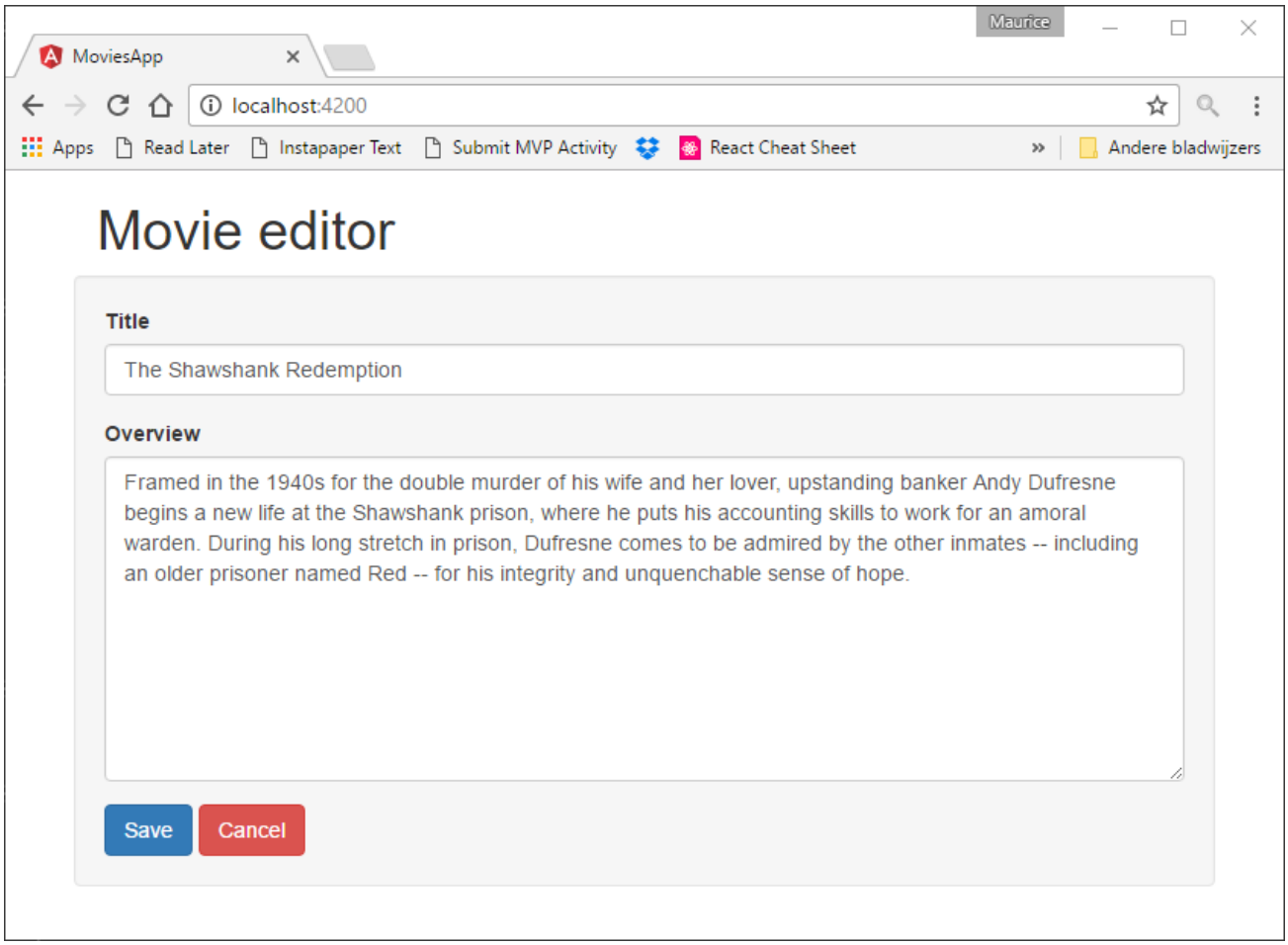
request the movie data. The movie **id** can be found in the **movieId** input property. Add a **movie** property of type **Movie** and use that to store the movie object that was loaded with the Ajax request.

```
movie: Movie = null;

constructor(private moviesService: MoviesService) { }

ngOnInit() {
  this.moviesService.getMovie(this.movieId)
    .subscribe(movie => this.movie = movie);
}
```

4. Open **movie-editor.component.html** and use the **ngIf** directive to only render the movie editor if there is a movie loaded.

```
<div class="row well" *ngIf="movie">
```

5. Test the application to see of everything is working as expected. You should now be able to open the movie details screen.



Saving changes to the movie details

In this section, you will implement the save action in the movie editor.

1. Open **movies.service.ts** and add an **updateMovie()** function. This function takes a movie object as parameters and uses an HTTP put request to update the move resource on the server.

```
updateMovie(movie: Movie): Observable<Response> {
  return this.http.put(`/api/movies/${movie.id}`, movie);
}
```

2. Open **movie-editor.component.ts** and add an **onSubmit()** function. Using this function, you should call the **moviesService.updateMovie()** function passing in the current movie. Emit using the **showList** output property when saving of movie is successful. This will show the movie list, with the updated data, again.

```
onSubmit() {
  this.moviesService.updateMovie(this.movie)
    .subscribe(() => this.showList.emit());
}
```

3. Open **movie-editor.component.html** and call the components **onSubmit()** function when the HTML form is submitted using the **ngSubmit** directive.

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

4. Test the application to see if everything is working as expected. You should now be able to open the movie details screen, make changes, and see those changes reflected on the main screen.

Add validation

Right now you can save the movie even if you deleted all data. As a movie without title and description makes little sense you should add validation to both input fields that they are required. The save button should only be enabled when the input data is valid.

1. Open **movie-editor.component.html** and add the **required** attribute to both the **title** input and the **overview** text area tags.

```
<input
  type="text"
```

```
  [(ngModel)]="movie.title"
  required
  class="form-control"
  id="title"
  name="title"
  #title="ngModel"
  placeholder="Title">
```

2. Add a template reference variable named **movieForm** for the form object and store the **ngForm** directive in the variable.

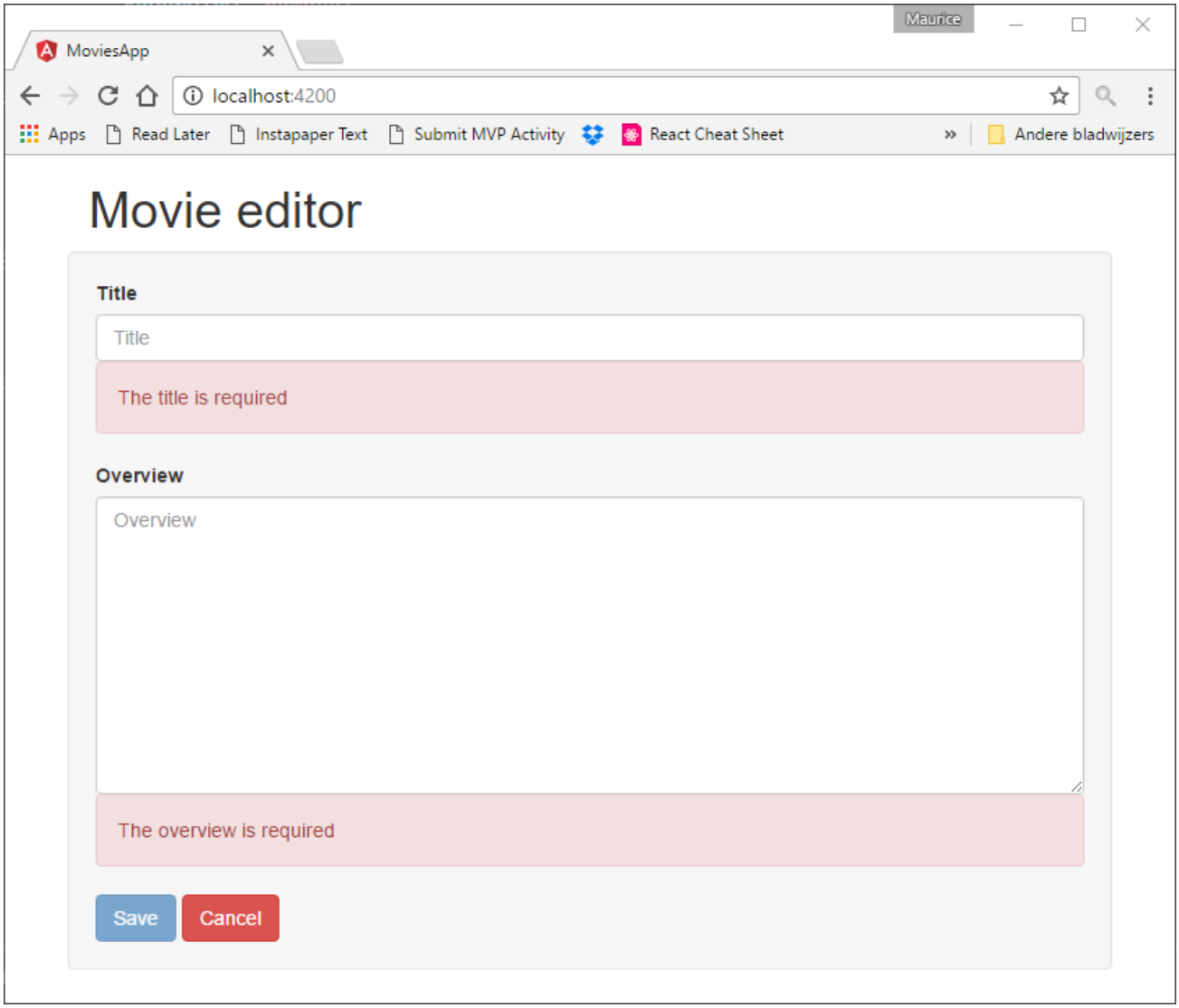
```
#movieForm="ngForm"
```

3. Add a **disabled** attribute to the **submit** button and disable the save button if the form is not valid.

```
<button type="submit" class="btn btn-primary"
        [disabled]="!movieForm.form.valid">
  Save
</button>
```

4. Add two template reference variables for the **title** and **overview** input. Add an error message that is only visible when the related input is invalid.

```
  <input
    type="text"
    [(ngModel)]="movie.title"
    required
    class="form-control"
    id="title"
    name="title"
    #title="ngModel"
    placeholder="Title">
  <div [hidden]="title.valid"
       class="alert alert-danger">
    The title is required
  </div>
```



Solution

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