



Expertise
and insight
for the future

Dung Luu

Comparison of State Management Solutions for Angular Application

Metropolia University of Applied Sciences

Bachelor of Engineering

Software Engineering

Bachelor's Thesis

20 November 2019

Author Title	Dung Luu Comparison of State Management Solutions for Angular Application
Number of Pages Date	30 pages 20 November 2019
Degree	Bachelor of Engineering
Degree Programme	Software Engineering
Professional Major	Software Engineering
Instructors	Janne Salonen, Head of Department
<p>By 2019, there are estimated 4.39 billion people access to internet. Companies spend a record amount of investment in e-commerce and social media. Web applications are getting more complex day-by-day, evolving constantly to allow millions of users to interact and upload data upstream concurrently. The demand for ever increasingly complicated web applications had pushed new solutions to be created none-stop. One of the newest solutions is called state management. This paper takes a look at one of the most popular web development frameworks, Angular, and how it works with three state-management libraries: NgRx, Redux, Mobx.</p>	
Keywords	State Management, Web Development, Angular, NgRx, Redux, Mobx

Contents

List of Abbreviations

1	Introduction	1
2	Technology Overview	3
2.1	Angular	3
2.1.1	Architecture	3
2.1.2	Module	4
2.1.3	Component	5
2.1.4	Templates, Directives and Data Binding	7
2.2	Common Definitions and Practices	7
2.2.1	Application State Definition	7
2.2.2	Reducers for Changing State	8
2.2.3	Application State in One Place	8
2.3	Redux	9
2.3.1	Three Fundamental Principles	9
2.4	NgRx/store	11
2.5	Mobx	13
2.5.1	Core Concepts	13
3	Method	15
3.1	Comparison Criteria	15
3.1.1	Learning Curve	15
3.1.2	Availability of Training Material	15
3.1.3	Compatibility and Bugs	16
3.2	Evaluation Scope	16
3.3	Project Requirement	16
3.4	Functional Requirement	16
3.5	Technical Requirements	17
4	Solution	18
4.1	Project Description	18
4.2	Project State Management Libraries	18
4.3	Project Architects and Implementation	19

4.4	Project Front-end	20
4.4.1	Redux Solution	20
4.4.2	NgRx Solution	23
4.4.3	Mobx Solution	24
5	Evaluation	26
5.1	Learning Curve	26
5.2	Availability of Learning Materials	26
5.3	Compatibility	26
5.4	Limitation	27
5.4.1	Complexity of Projects	28
5.4.2	Experience Level of Author	28
5.4.3	Arbitrary Factor	28
6	Conclusion	29
	References	30

List of Abbreviations

HTML	Hypertext Markup Language. The standard markup language for documents designed to be displayed in a web browser.
CSS	Cascading Style Sheets. A stylesheet language used to describe the presentation of a document written in HTML or XML.
SCSS	Sassy CSS
SASS	Syntactically Awesome Style Sheets
CERN	The European Organization for Nuclear Research
MVC	Model-View-Controller
CLI	Command Line Interface
DOM	Document Object Model
UI	User Interface

1 Introduction

In the 1980s, Tim Berners-Lee at CERN developed a solution to share information across long distance between scientists. What he created, unexpectedly evolved into something much more powerful, that did not simply make the scientists work easier but changed even the foundation of communication, relationship and lifestyle for everyone today, the World Wide Web, or in a broader term: the internet.

Nowadays the convenience of the internet is taken for granted. It has altered the way people communicate to each other. Sending an email to another continent takes only seconds comparing to days or even months of sending a paper letter. Making a duplicate picture to share with loved ones in another city is virtually free and effortless while the quality of pictures never degrades.

In the early days of the internet, web applications were nothing but static web pages which displayed some information to the user. The information was written and edited directly with HTML code. This web 1.0 era applications were static, did not retain session information. Every request was treated as a separate independent transaction, all the connection between the browser and server was lost when the transaction ended.

By mid 1990s, there were only about one-hundred thousand websites. The technology for web application was evolving quickly, and soon the web 2.0 and 3.0 periods were entered. Websites were richer in content and more interactive with the user. The browser would retain some information about the browsing history in files called cookies. Technology for developing web application was not just HTML but contained multiple programming languages: HTML for the structure, CSS for format and JavaScript for logic. Websites were no longer read-only but the user could send back information to the server. Applications such as e-mail or social media platforms had users sending information back to data server at constant rate.

Recently, due to the increasing complexity in web applications, more and more solutions are developed every day to propose a way to deal with the issue and simplify such complex applications. One of those solutions is state management such as Redux,

Mobx, and NgRx. This thesis provides an overview about these technologies and compares them in a front-end application developed with Angular.

Due to the wide coverage of different libraries and practices, the project is simplified. The focus of this study is on the implementation of the different technologies into Angular eco-system.

2 Technology Overview

Angular is one of the most popular web development frameworks and it is being updated every six months. It is versatile, robust and can work with any state management libraries existing. This section is an overview of Angular and three state management solutions: Redux, NgRx and Mobx.

2.1 Angular

AngularJS was first released in 2010 by a team of developers at Google as a side project and was soon received full support from Google as it gains popularity and later became one of the most widely used platform for web application development. AngularJS applied the Model-View-Controller (MVC) which was traditionally used for developing desktop graphical user interfaces, for developing a single page web application. The MVC pattern separates data, presentation and logic into separate components for better management. The bigger and more complex projects are, the more useful MVC is.

In 2014, the developer team implement a fundamental change to the platform. Now it is modular with sub-components that are in MVC model. They also decided to change the naming convention to Angular and update a major version change twice a year, with the latest being Angular 7 released in October 2018.

In the present project an Angular 7 application was built using Angular CLI which utilizes command lines to install Angular, generate application, components, services, etc. To create a new application, one would use the command line *ng new [Application Name]*.

2.1.1 Architecture

Angular is a modular system with the building blocks called NgModules, see Figure 1.

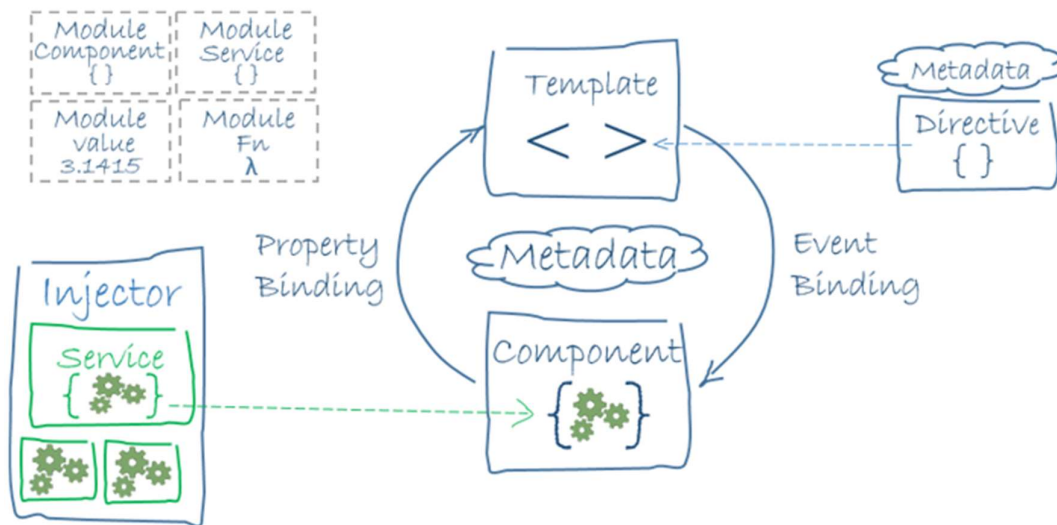


Figure 1. Visual architecture of an Angular application. (Source: <https://angular.io/guide/architecture>)

The template and component together compose the UI of the Angular applications. The template provides an HTML format while the component provides the logic behind it. (Angular architecture overview n.d.)

2.1.2 Module

Modules are the building blocks of an Angular application. They contain components, services and other meta data to be used by components within the module. Modules can also export and import functionalities to each other (Angular introduction to modules n.d).

Every Angular application has a root module. This module is automatically generated when the application is created, named AppModule. Figure 2 below is the visualization of module architectures.

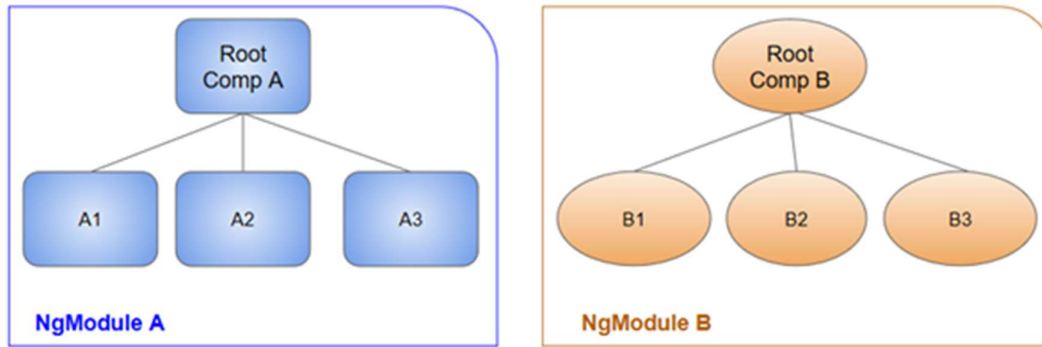


Figure 2. Modules architectures (Source: <https://angular.io/guide/architecture>)

Angular has built-in libraries with modules ready to be imported and used in the application. Figure 3 shows the architecture of a built-in module, developers have the option to only use some specific feature (i.e a component or a service) of a module.

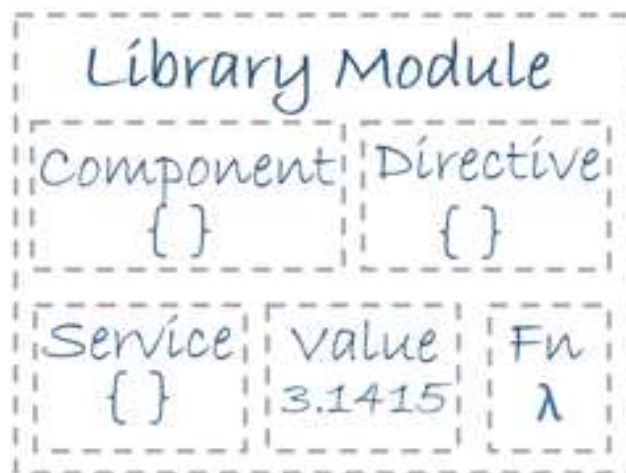


Figure 3. Architecture of a built-in module (Source: <https://angular.io/guide/architecture>)

These built-in libraries has @angular prefix and provide many useful functionalities.

2.1.3 Component

While NgModules are the building blocks of an Angular application, components are the building blocks of the modules. Each component is by itself an MVC structure with an

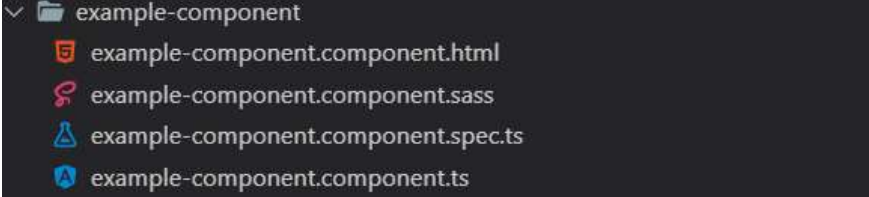
HTML and CSS file for the view, a typescript file for the logic controller. Every Angular application has at least one component, the root component that connects a component hierarchy with the page document object model (DOM). (Angular introduction to components n.d.)

With Angular CLI, new components are created by navigating to designated folder and using with command: `ng generate component [Component Name]` or short cut: `ng g c [Component Name]`, see Figure 4.

```
PS D:\Google Drive\AngularPractice\mean-course> ng g c ExampleComponent
CREATE src/app/example-component/example-component.component.html (32 bytes)
CREATE src/app/example-component/example-component.component.spec.ts (699 bytes)
CREATE src/app/example-component/example-component.component.ts (313 bytes)
CREATE src/app/example-component/example-component.component.sass (0 bytes)
UPDATE src/app/app.module.ts (1293 bytes)
```

Figure 4. Example of creating a component using Angular CLI

After running the command and letting the process finished as depicted in Figure 4 above, a new folder will be created containing all the auto generated files for the component. They include html file for template, style sheet, typescript file for logic, and a specification file, see Figure 5 below.



```
example-component
├── example-component.component.html
├── example-component.component.sass
├── example-component.component.spec.ts
└── example-component.component.ts
```

Figure 5. Component folder and files auto created after the generate command

Depending on the project setting, the style sheet can be in CSS, SASS or SCSS. In the example above, the project setting dictated the auto generated style sheet to be a SASS. However, this could be changed to any other format without creating any problem.

2.1.4 Templates, Directives and Data Binding

A template combines HTML with Angular markup that can modify HTML elements before they are displayed. Template directives provide program logic, and binding markup connects your application data and the DOM. There are two types of data binding:

- Event binding lets the app respond to user input in the target environment by updating your application data.
- Property binding enables interpolating values that are computed from the application data into the HTML.

These binding types help to create highly responsive websites. However, it should be noted that over utilized bindings can impact the performance of the application.

2.2 Common Definitions and Practices

There are many libraries or frameworks for State Management in the web application, but they are often inspired by one another, therefore common straits and practices exist. The sections below answer the questions what the state of an application is, how it is often managed, and what kind of actions are used to change the state.

2.2.1 Application State Definition

State is the information application stored about the user, their current interaction and location within the application, and other global information of the application such as configuration setting, which information the user has received, et cetera.

Browser activities in the form of HTTP request are inherently stateless. Each time the user's browser sends a request, the server would send a response, and the request-response activities are treated as separate connection. There is no awareness of the previous response and the current "state" of the browser. For example, if the user is viewing a page with information A and B now performs an action that needs information A, B, C and D. The browser sends a request to the browser asking for all A, B, C and D

and the server will therefore respond with all the information including resending A and B. With state management, the browser would only ask for information C and D.

Using cookies is one way of keeping information in the browser, but there is a limitation of size. With JavaScript single page applications getting more complicate, it is better to use a framework such as Redux or NgRx.

2.2.2 Reducers for Changing State

Actions used for changing state are called reducers. They are pure functions which take the previous state and an action and return the next state. For a function to be pure, it must not have any side effect, given the same input, it always returns the same consistent output. It is a simple rule but is commonly violated due to the practice of using other readily made functions in the evaluation. When the functions are changed by another programmer, it could alter the evaluation and alter the return output.

This practice is not applied to Mobx in this study. Mobx applies a more loose approach toward the problem and does not have strict requirements.

2.2.3 Application State in One Place

Managing the state of a large and complex application can be a complicated task. It is easy to lose track of data and mismanage them. Data location can be forgotten or duplicated, causing the application to be messy and hard to maintain.

To simplify the problem, state management solutions tend to keep all the of the application state data in one place. Redux calls this practice "Single source of truth." This approach keeps the application clean and easy to maintain. It eliminates the chance of the application data to be corrupted, duplicated or lose tracked.

2.3 Redux

Redux is a JavaScript library developed for using with React technology. It was developed for a state management library for React, but it could be used with any other platforms such as Angular in this project (Redux introduction n.d.)

2.3.1 Three Fundamental Principles

Redux principles are straightforward and easy to understand. Its simplicity means newcomers quickly learn and adapt to any existing team or project (Redux three-principles n.d.)

- **Single source of truth:** The state of the whole application is stored in a single object. This principle helps to relieve the responsibility of coding on server side and makes it easier to maintain and debug the application
- **State is read-only:** It limits the way to change state down only by an action. All changes should be centralized and happen in strict order.
- **Actions are implemented by pure reducers:** This is to eliminate side effect, and to have the state in strict control.

Listing 1 below shows an example of a pure reducer:

```
function todos(state = [], action) {
  switch (action.type) {
    case 'ADD_TODO':
      return [
        ...state,
        {
          text: action.text,
          completed: false
        }
      ]
    case 'COMPLETE_TODO':
      return state.map((todo, index) => {
        if (index === action.index) {
          return Object.assign({}, todo, {
            completed: true
          })
        }
        return todo
      })
    default:
```

(30)

```
        return state
    }
}
```

Listing 1. Example of a pure reducer

The function above takes two inputs: state and action. It then carries out computation and return a new state. For it to be “pure function,” it must always return the same result given the same input, which is called “no side-effect.” One example of impurity is calling other functions during the computation, if those functions are changed over time, it may alter the result of the computation.

(30)

2.4 NgRx/store

NgRx is a group of libraries for building a reactive Angular application. In it, @ngrx/store is the state management solution and is inspired by Redux. Like Redux, it uses pure functions call reducers to change old state to new state according to action. Figure 4 shows the lifecycle of NgRx state management. (NgRx introduction n.d)

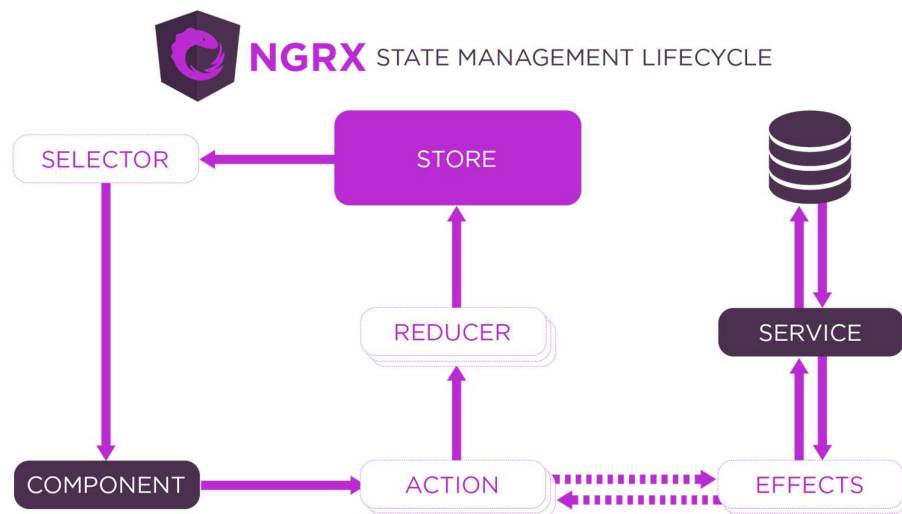


Figure 6. General flow of application state of NgRx (Source: <https://ngrx.io/guide/store>)

The architecture of NgRx is based on the following three building blocks:

- **Actions:** are inputs and outputs and help to handle events in application. An Action in NgRx is a simple interface that defines type and properties of the actions (NgRx action n.d.)
- **Reducers:** are pure functions used to change state to the next state. This concept is discussed in the section above (NgRx reducer n.d.)
- **Selectors:** are pure functions used for obtaining slices of store state. There are a few helper functions to help optimizing by offering features such as: portable, memorization, composition, testable, type-safe. With the use of pure functions, the same argument will always provide the same result. Therefore, by keeping

(30)

track of the latest arguments, there is performance benefits by avoiding carrying out the same computations again. This practice is called memorization (NgRx selector n.d.)

Listing 2 below is the interface of an NgRx Action:

```
interface Action {  
  type: string;  
}
```

Listing 2. NgRx Action interface

As to how to create an action, please see the example in Listing 3.

```
import { createAction, props } from '@ngrx/store';  
  
export const login = createAction(  
  '[Login Page] Login',  
  props<{ username: string; password: string }>()  
);
```

Listing 3. Example of creating action in NgRx

Since NgRx is inspired by Redux, it is not surprising to see the familiar of the two solutions in term of principle, architect and practices.

(30)

2.5 Mobx

"Anything that can be derived from the application state, should be derived. Automatically." That is the philosophy behind Mobx. It uses a reactive virtual dependency state graph and it is only updated when strictly needed to optimize the synchronization application state with the front-end framework (Mobx one-page summary n.d.)

One thing to be noted is that Mobx is not a state container nor an architecture. Mobx web page itself stated it is just a library for a technical problem and should not be used as an alternative to other state management solutions (Mobx one-page summary n.d.)

2.5.1 Core Concepts

Similar to previous two solutions, Mobx also has a few simple and straightforward concepts.

Observable state

Mobx set the application state to observable objects which make them a spreadsheet but each cell could hold wide range of data (Mobx one-page summary n.d.)

To set them to be observable, the user simply needs to import the feature from mobx library and use `@observable` decorator for class properties as in the example below (Listing 4):

```
import { observable } from 'mobx'

export class Student {

  @observable name = ""
  @observable id = 0

}
```

Listing 4. Example of setting properties with `@observable` decorator in Mobx

Compute values

(30)

With the decorator `@computed`, mobx will help to track and update automatically the property when data is changed (Mobx one-page summary n.d.) In the example below (Listing 5):

```
export class StudentList {  
  @observable studentList[ ]  
  @computed get numberOfStudents () {  
    return this.studentList.length  
  }  
}
```

Listing 5. Example of using `@computed` decorator in Mobx

Reactions

" MobX reacts to any existing observable property that is read during the execution of a tracked function." (Mobx one-page summary n.d.)

With the derive anything that could be from the application state, Mobx would handle the state change and notice all the affected components.

Compared to Redux and NgRx, Mobx has much more flexible requirements. There is no requirement for a strict architecture type or the use of pure function for actions.

3 Method

It is impossible to compare the solutions objectively without a well defined approach before starting the projects. This section discusses the criteria, scope and different type of requirements that were applied and observed during the implementation.

3.1 Comparison Criteria

For an Angular developer to quickly adapt the new technologies, the criteria would be the learning curve, the availability of training material, and how compatible the new technologies would be with Angular.

3.1.1 Learning Curve

The learning curve is judged from a point of view of an Angular developer. The factors to be considered are the coding styles, syntax, architectures, complexity of applying these libraries and the availability of tutorial material.

- **Coding style and syntax:** This paper evaluates the intuitiveness of utilizing these technologies when it comes to Angular developers. It tries to answer questions such as how familiar of the coding style to Angular, how seamless it fits into the existing code.
- **Architectures:** Angular has certain structures and practices. The level of difference to these from the being-tested libraries would contribute to the learning and the adaptation required from Angular developers.

3.1.2 Availability of Training Material

Learning a new technology is not simply reading documentation and API libraries. Reading and practicing with example codes is an excellent way to quickly learn and apply new coding practices.

(30)

During the implementation of the projects, the availability of training and tutoring materials of each library is put to comparison. While there may be a wide range of materials online, this paper only considers of materials that help to incorporate the technologies to Angular applications specifically.

3.1.3 Compatibility and Bugs

Compatibility is an important aspect to software development. It is critical for all technologies to work with each other seamlessly, not just during development phase but also maintaining and improving phases. The projects test and report any problem in terms of compatibility with the latest versions.

3.2 Evaluation Scope

All Redux, NgRx, and Mobx are complex and rich libraries with many functionalities. The scope of this study is limited to only the core state management functionality. Therefore, the main scope of testing would only revolve around Redux/store, NgRx/store and basic Mobx function.

3.3 Project Requirement

Front-end projects were developed with Angular to demonstrate the use of the above state management solutions. The projects would have the same state structures but implement different technology to compare the difficulty of how the technologies could be integrated into an Angular application.

3.4 Functional Requirement

Due to the multiple projects and technologies applied, the functional requirement was reduced as much as possible to have a feasible workload. Below is the minimum functional requirements for the projects:

(30)

- Use of state management technologies: Three front-end applications with state management application were to be developed. The applications have an application state managed by the methods according to the philosophy of the utilized technologies used in each application.
- Demonstrate the modification action of the application state: The projects have at least two actions that carrying out the modification of application states. One action is adding data to current state, and the other action is removing designated data from the current application state.

This minimal requirement should be noted as a limitation during evaluation since it would not be possible to encounter any highly sophisticated problems during the implementation.

3.5 Technical Requirements

To be able to carry out the evaluation needed, below are the minimum technical requirements for the projects:

- Front-end: The projects used the latest stable version of Angular framework. Because of the frequent update of Angular, the projects can use different Angular version depending on the created date.
- State management solutions: The projects implemented Redux, NgRx, Mobx solutions with their latest version. If there were any problems with compatibility or stability, they were rolled back to older and stable versions.

Redux and Mobx are designed with the purpose to be applied in React applications. To reduce boiler plate codes, there are many support libraries that help bridging them to Angular and reduce the effort of developer. To have a better comparison between Redux, NgRx and Mobx implementations, these support libraries were in the projects.

(30)

4 Solution

Based on the requirement set above, the projects were developed. The development had some time gap and was inspired by a multiple source of tutorials and example sources. These factors had an impact on the final product such as UI and the project folder structures, but has minimal impact on the evaluation of the thesis.

4.1 Project Description

The project is a student list management app in which the user can add or remove student data. The project applied state management technologies to demonstrate how to apply. There is one area for data inserting to the state application, and one area for displaying the current application state data.

In the input area, there are three input fields to take student data: name, Id and email and a button to carry out the action of uploading the data to the application state. In the display area, there is a table displaying all the students and their data in three columns. Each student data row has a Delete button to remove that data item from the application state

4.2 Project State Management Libraries

The state management libraries used for the projects were:

- Redux solution: Redux 4.0.1, angular-redux 10.0.0
- NgRx solution: NgRx 8.4.0
- Mobx solution: Mobx 5.14.2, mobx-angular 3.0.3

The assisting libraries, angular-redux and mobx-angular, are not required. However, they help to reduce the overall boilerplate codes needed for Redux and Mobx projects,

(30)

therefore incorporating them into the solutions provides a broader view for the evaluation.

4.3 Project Architects and Implementation

Redux and NgRx projects have very similar philosophy and architecture, especially when Redux is combined with angular-redux libraries. The implementation of the two solutions require minimal change and the formats are very similar. The same files and data structures could be used for both solutions. However, this is not the case with Mobx with a very different and more relaxed philosophy. Figure 7 below shows the architecture of the NgRx and Redux solutions:

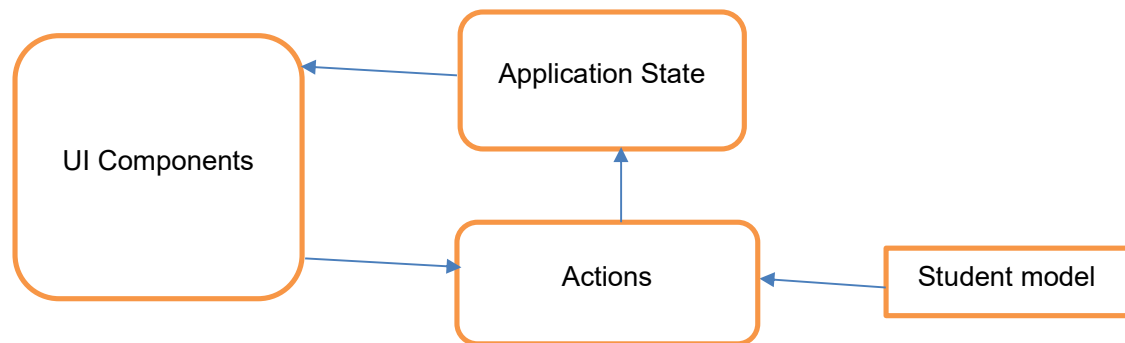


Figure 7. Architecture of NgRx and Redux solutions

The biggest different in Mobx is that the actions in Mobx are not carried out by pure functions as in the previous solutions but just need a method with the selector @action (see Figure 8).

(30)

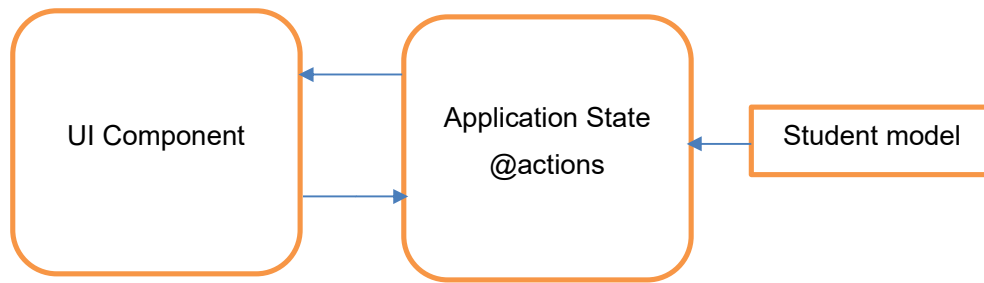


Figure 8. Architecture of Mobx solutions

The architecture of Mobx is less clear, less defined. In a small project such as this, it is easy to maintain, but for a larger project with multiple developers, it would be more difficult to manage.

4.4 Project Front-end

The front-end of the project contains a form for the user to fill in a new student. There is an option to add that new student data. This demonstrates the adding modification of the application state.

The application state is displayed as a list of students. Each student item has a method to remove it from the list, hence demonstrating the removing action to the application state.

4.4.1 Redux Solution

In the Redux solution, everything is contained in one UI Angular component. There is a form to fill in the student data and an Add button to input the data to the application state. Listing 6 below is the UI code for this solution and Figure 9 is a screen short and explanation of how the solution works.

(30)

```

<h2>Add students</h2>
<form (ngSubmit)="onAdd()" #studentForm="ngForm">
  <div class="form-row">
    <div class="col-auto">
      <input
        type="text"
        class="form-control"
        placeholder="Name"
        id="name"
        [(ngModel)]="student.name"
        name="name"
        #name="ngModel">
    </div>
    <div class="col-auto">
      <input
        type="number"
        class="form-control"
        placeholder="Id"
        id="id"
        [(ngModel)]="student.id"
        name="id"
        #id="ngModel">
    </div>
    <div class="col-auto">
      <input
        type="text"
        class="form-control"
        placeholder="email"
        id="email"
        [(ngModel)]="student.email"
        name="email"
        #email="ngModel">
    </div>
    <div class="col-auto">
      <button type="submit" class="btn btn-primary">Add</button>
    </div>
  </div>
</form>
<br />
<h2>Student List</h2>
<div *ngIf="(studentList | async)?.length==0">
  Student list is empty
</div>
<div *ngIf="(studentList | async)?.length!=0">
<table class="table">
  <thead class="thead-inverse">
    <tr>
      <th>Name</th>
      <th>Id</th>
      <th>Email</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let student of studentList | async">
      <td><span>{{ student.name }}</span></td>
      <td><span>{{ student.id }}</span></td>
      <td><span>{{ student.email }}</span></td>
      <td><button class="btn btn-primary" (click)="onRemove(student)">De-
lete</button></td>
    </tr>
  </tbody>
</table>
</div>

```

(30)

Listing 6. HTML code for Redux UI solution

The HTML code above (Listing 6) has a form to which the user can fill in data and a table that lists all items in the state. See Figure 9 below for more explanation.

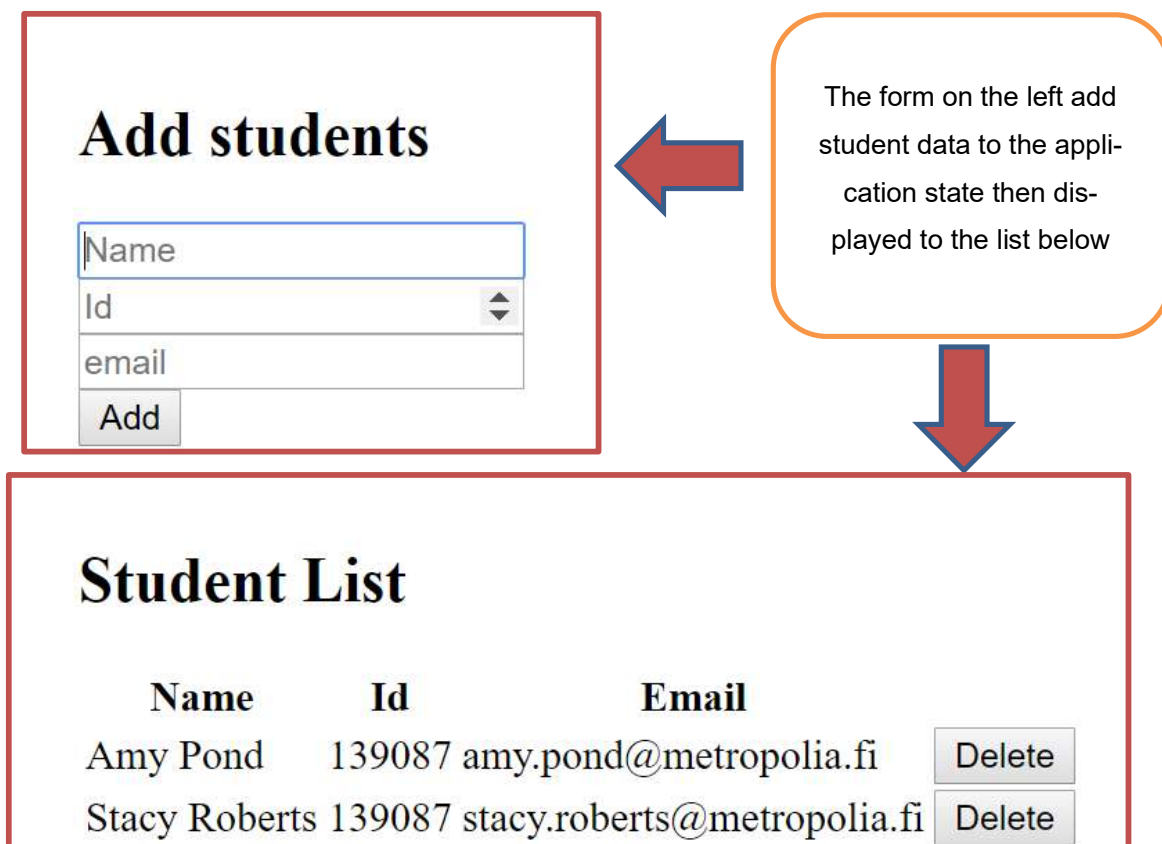


Figure 9. Screen shot and comments of how the Redux solution work

The students are displayed in the list with a remove button for each student. Clicking the button next to the student will remove it from the list and the data from the application state.

(30)

4.4.2 NgRx Solution

In this solution, there are two UI components: Create and Read. The Create component has a form to fill in student information and a button to add the data into application state. Listing 7 below shows the code of this component.

```
<div class="left">
  <input type="text" placeholder="name" #name>
  <input type="text" placeholder="id" #id>
  <input type="text" placeholder="email" #email>

  <button (click)="addStudent(name.value, id.value, email.value)">Add student</button>
</div>
```

Listing 7. HTML code for Create component of NgRx solution

The Read component displays the student list, clicking on the student removes it from the application state.

```
<div class="right" *ngIf="studentList">
  <h3>Student List</h3>
  <ul>
    <li (click)="deleteStudent(i)" *ngFor="let student of studentList | async; let I = index">
      <a target="_blank">{{ student.name }}</a>
    </li>
  </ul>
</div>
```

Listing 8. HTML code for Create component of NgRx solution

The components above formed the UI below. See Figure 10 for the screen short of NgRx solution with comments.

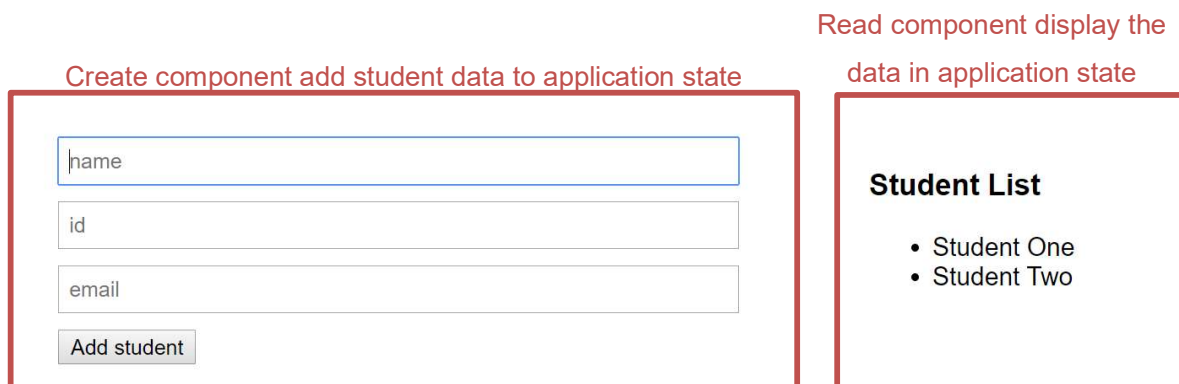


Figure 10. Screen shot and comments of how the NgRx solution work

(30)

NgRx solution was the first to be implemented, therefore it was the simplest and the student list didn't even present all three field of data. But this shortcoming had no negative impact for the purpose of comparing state management solution in the background.

4.4.3 Mobx Solution

The Mobx solution is implemented with the UI inherited from Redux solution. The UI code in Listing 9 below was taken from Redux project and modified to work with Mobx.

```

<h2>Add students</h2>
<form (ngSubmit)="onAdd(name, id, email)" #studentForm="ngForm" *mobxAutorun>
  <div class="form-row">
    <div class="col-auto">
      <input
        type="text"
        class="form-control"
        placeholder="Name"
        id="name"
        [(ngModel)]="name"
        name="name"
        #name="ngModel">
    </div>
    <div class="col-auto">
      <input
        type="number"
        class="form-control"
        placeholder="Id"
        id="id"
        [(ngModel)]="id"
        name="id"
        #id="ngModel">
    </div>
    <div class="col-auto">
      <input
        type="text"
        class="form-control"
        placeholder="email"
        id="email"
        [(ngModel)]="email"
        name="email"
        #email="ngModel">
    </div>
    <div class="col-auto">
      <button type="submit" class="btn btn-primary">Add</button>
    </div>
  </div>
</form>
<br />
<h2>Student List</h2>
<div *ngIf="(studentList | async)?.length==0">
  Student list is empty
</div>

```

(30)

```
<div *ngIf="(studentList | async)?.length!=0">
<table class="table">
  <thead class="thead-inverse">
    <tr>
      <th>Name</th>
      <th>Id</th>
      <th>Email</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let student of studentList | async">
      <td><span>{{ student.name }}</span></td>
      <td><span>{{ student.id }}</span></td>
      <td><span>{{ student.email }}</span></td>
      <td><button class="btn btn-primary" (click)="onRemove(student)">De-
lete</button></td>
    </tr>
  </tbody>
</table>
</div>
```

Listing 9. HTML code for UI component of Mobx solution

The Mobx solution had a problem with compatibility and required version rollback. Also, the differences in the architecture and coding style made it difficult to trouble shoot and the project was not working as intended.

5 Evaluation

The evaluation based on the comparison criteria is discussed below. They were acquired during the processes of researching, learning and creating solutions for each technology. Also, during the developments, many limitations surfaced and were taken note of. They are also discussed at the latter half of this section.

5.1 Learning Curve

For anyone familiar with the Angular framework, NgRx or Redux is easy to learn and apply. Both have similar syntax, architectures and principles. Mobx has a unique syntax and a very relaxed architecture. It is more challenging to learn and apply Mobx than the previous two.

5.2 Availability of Learning Materials

Learning material and examples are abundant for NgRx and Redux. Especially for Angular, NgRx materials are easy to learn and apply. For Redux, most training and tutorial materials include another layer of dependency (i.e. angular-redux). Mobx is the most challenging solution to look for learning material to apply with Angular. For the same solution, the coding for Mobx was more complicated and required more effort.

5.3 Compatibility

NgRx is the most compatible with Angular. After all, this is the libraries developed with the primary purpose to be used with Angular.

Both Redux and Mobx solutions encountered compatibilities problems and had to roll-back to older versions. These libraries created with the usage with React, and with the usage middleware libraries such as angular-redux and mobx-angular, the applications have more dependencies that would need to be waited for update and debugged.

(30)

With the latest Redux library (version 4.0.4), the solution encountered problem such the one illustrated below in Figure 11:

```
Failed to compile.  
  
node_modules/@angular-redux/store/components/ng-redux.d.ts(10,31): error TS2420: Class  
'NgRedux<RootState>' incorrectly implements interface 'ObservableStore<RootState>'.  
Property '[Symbol.observable]' is missing in type 'NgRedux<RootState>' but required in type  
'ObservableStore<RootState>'.
```

Figure 11. Error encountered with Redux implementation

Setting Redux to the strict version 4.0.1 resolves the problem. A quick search on the internet shows this seems to be a common problem with Redux.

The latest Mobx version (5.15.0) gives the Mobx solution the compatibility problem below (see Figure 12):

```
ERROR in node_modules/mobx/lib/api/flow.d.ts(8,83): error TS2315: Type 'Generator' is not generic.  
node_modules/mobx/lib/api/flow.d.ts(8,108): error TS2304: Cannot find name 'AsyncGenerator'.
```

Figure 12. Error encountered with Mobx implementation

Rerolling Mobx to an older version 5.14.2 resolves this problem. However, the Mobx solution is currently not working due to an unknown error.

5.4 Limitation

After the development of all solutions, it became clear that there are many limitations of this study. The size and complexity of the projects were small, and the author has limited experience with all technologies utilized, so the comparison did not have a sophisticated analysis. These libraries were updated frequently, there may be problem that was universal and not Angular specific. These limitations are discussed in more detailed below.

(30)

5.4.1 Complexity of Projects

State management solutions are often reserved for large and complicated applications. The projects implemented here are very simple. The scope was limited and narrowed down to have a level view of each solution and the feasible implementations.

Therefore, this paper does not give an in-depth view of each technology, but rather an overview of how quickly an Angular developer could learn and adapt.

With deeper investigation and larger and more complex projects, there could be other factors to be revealed thereby changing the outcome of the comparison presented here.

5.4.2 Experience Level of Author

It should be noted that the author of this paper is very inexperienced with any of the technologies being used here. It is not just Redux, NgRx and Mobx that were being learnt during the implementation. The base of the comparison, Angular application, was also being learnt as well as web development in general.

5.4.3 Arbitrary Factor

With the fast-moving nature of software development, new versions of these libraries are released and updated frequently.

Bugs or incompatibility having occurred during the development of the projects could have been caused by simply bad timing. There could be a recent update with a certain library with bugs temporary not fixed while another library could be at a stable stage and yet waiting for an update to take place soon.

Given a different time, there could be more problems in other libraries and the result of such a comparison factor might be altered.

6 Conclusion

Both in terms of learning curve and compatibility, not surprisingly NgRx is the best choice since it is a solution that was created for the Angular eco-system. The application of NgRx into the existing Angular application required the least effort.

The Redux solution is also very effortless to apply into existing applications due to the simplicity and similarity with NgRx. However, with the extra layer of dependency, there are more weak points on the compatibility comparison.

Mobx is the most challenging solution due to multiple reasons. First of all, Mobx is designed to fit into the React eco-system. Every material finding on its page and most material online are about applying it into React applications. And even then, it is not made to be a replaceable alternative to Redux. Applying it into an Angular application required more training and more effort.

(30)

References

Angular architecture overview n.d., accessed 15 September 2019, <<https://angular.io/guide/architecture>>

Angular introduction to modules n.d., accessed 15 September 2019, <<https://angular.io/guide/architecture-modules>>

Angular introduction to components n.d., accessed 20 September 2019, <<https://angular.io/guide/architecture-components>>

Mobx one-page summary n.d., accessed 20 October 2019, <<https://mobx.js.org/RE-ADME.html>>

NgRx action n.d., accessed 22 September 2019, <<https://ngrx.io/guide/store/actions>>

NgRx introduction n.d., accessed 22 September 2019, <<https://ngrx.io/docs>>

NgRx reducer n.d., accessed 22 September 2019, <<https://ngrx.io/guide/store/reducers>>

NgRx selector n.d., accessed 22 September 2019, <<https://ngrx.io/guide/store/selectors>>

Redux introduction n.d., accessed 10 September 2019, <<https://redux.js.org/introduction/motivation>>

Redux three-principles n.d., accessed 10 September 2019, <<https://redux.js.org/introduction/three-principles>>

Simon Kemp 30 January 2019, accessed 02 December 2019, <<https://wearesocial.com/blog/2019/01/digital-2019-global-internet-use-accelerates>>