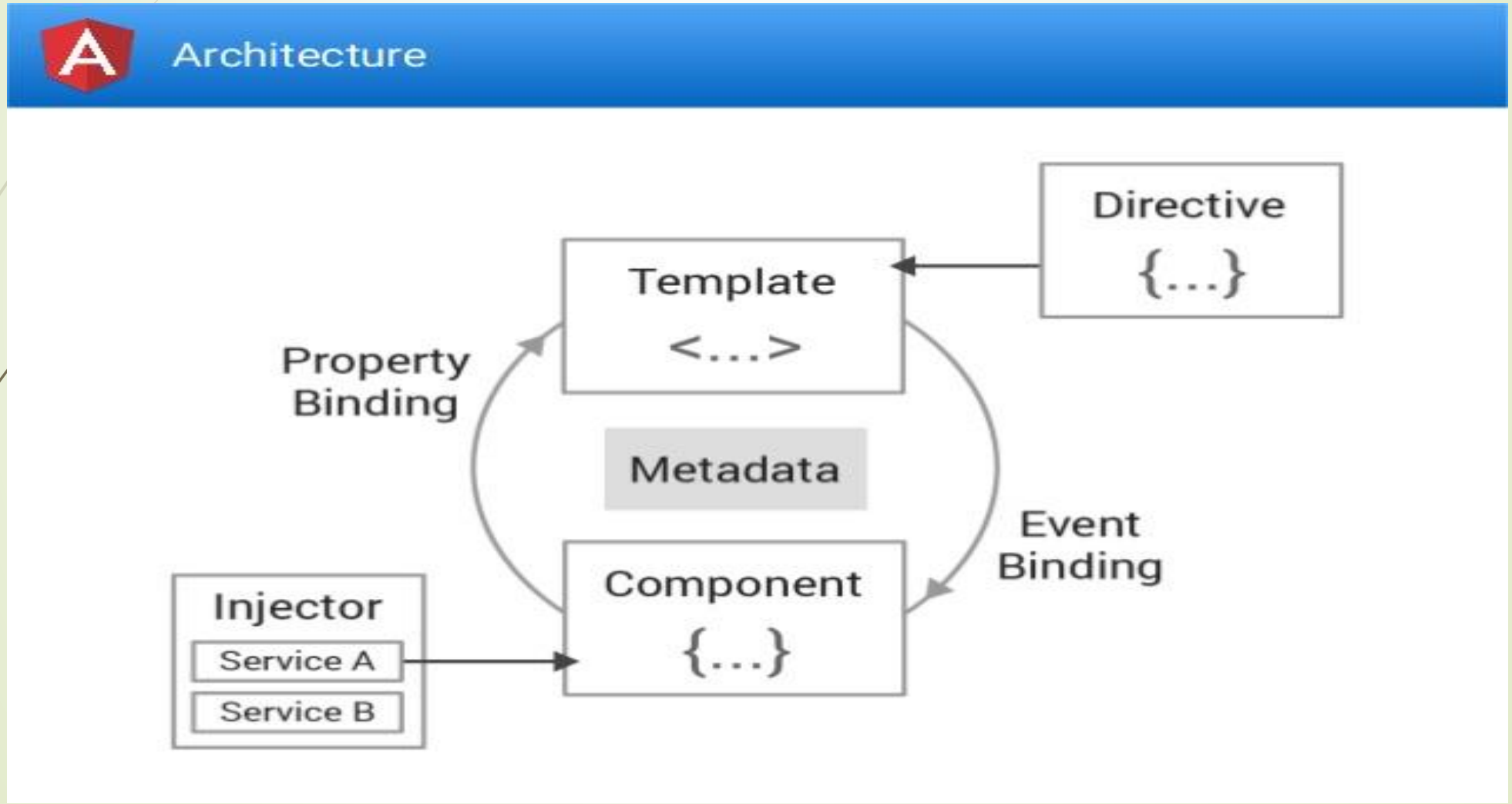
CakePHP3.x With Angular2+

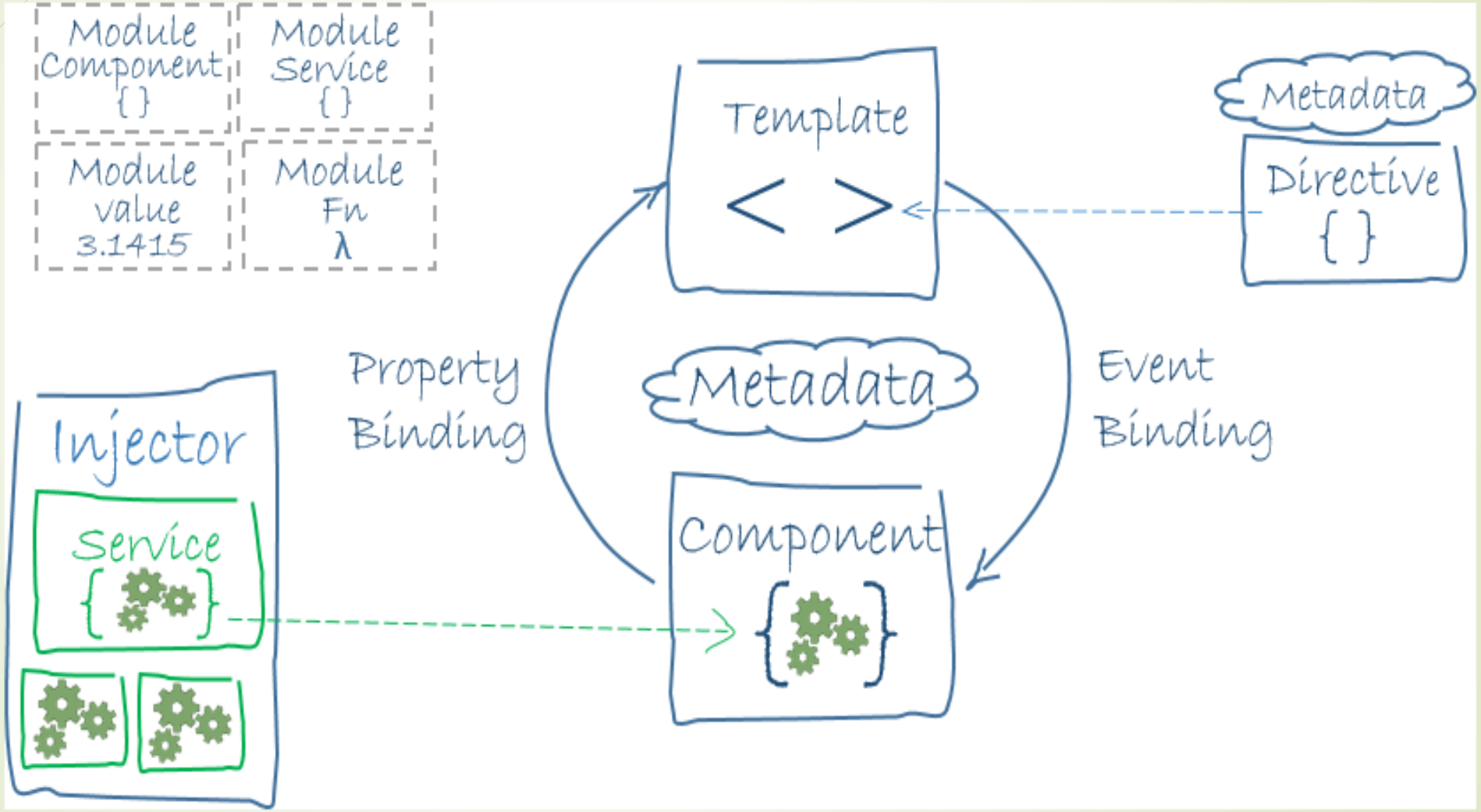
CAKEPHP Version 3.x backend with Angular2+ for frontend

Create by PAKGON R&D Team

Angular2+ Architecture



Angular2+ Architecture



Components

- Components are the main classes where most of our code is
- **Template**:
  - Mostly HTML with some additional instructions
  - Styles are scoped to the component (similar to Web Components)
- **Component Class**:
  - Methods and properties that can be accessed and invoked from the rendered page
- Metadata specified in **Decorators:**
  - Additional instructions like pipes and directives

Template expressions

- **Model to view** binding for values and properties

```
<span class="badge">{{question.votes}}</span>

<div [class.active]="question.state == 'active'">
```

- **View to model** binding

```
<div (click)="onClick($event)"></div>
```

- View to model & model to view (**2-way-binding**)

```
<input [(ngModel)]="value"></input>
<!-- short hand for -->
<input [ngModel]="value" (ngModel)="value = $event"></input>
```

Structural Directives *ngIf, *ngFor

- ***ngIf** conditionally renders a template

```
<!-- *ngIf paragraph -->
<div *ngIf="questions">
  We have some questions
</div>

<!-- [ngIf] with template -->
<template [ngIf]="questions ">
  <div>
    We have some questions
  </div>
</template>
```

- ***ngFor** loops over a collection

```
<div *ngFor="let question of questions">{{question.question}}</div>
```

Forms

- **ngModel** adds two-way binding between model and input value

```
<input type="text" [(ngModel)]="newQuestion" name="question" required>
```

- **ngSubmit** handles form submissions

```
<form (ngSubmit)="ask(newQuestion)">
```

- Access to the form controller **ngForm**

```
<form #questionForm="ngForm" (ngSubmit)="ask(questionForm.value)">
  ...
  <button type="submit" [disabled]="!questionForm.valid">Ask</button>
</form>
```

Services

- Services are useful to share common code between different controllers.
- Services are **injectable**, so they can be used in any Component / Pipe / Directive ...

```
@Injectable()
export class StorageService {
```

- Services are created by Angular when they are first requested and are treated as **singletons**

```
export class TalkComponent{
  //StorageService is injected to the component by angular
  constructor(private storageService:StorageService) {}
```

Pipes

- Pipes are template helpers to transform values

```
<small>{{question.date | date:'shortTime'}}</small>
```

- Pipes are *pure* by default
  - A pure pipe is only invoked if its primitive input changed or the reference of an object changed
- *Impure* Pipes
  - Needed if the pipe must be reevaluated when properties of an object changed or elements of an array are updated
  - Are always evaluated (expensive)

Router

- Routes are defined as **URL patterns** and handled by a target component

```
const routes: Routes = [
  { path: 'talk/:id', component: TalkComponent }
];
```

- Matching **route parameters** can be accesses by injecting the *ActivatedRoute*

```
export class TalkComponent {
  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    let id = this.route.snapshot.params['id'];
  }
}
```

Router Navigation

- The **routerLink** directive can be used to navigate to a another route

```
<a routerLink="/login">Login</a>
```

- The router can be used to navigate programmatically

```
constructor(private router: Router)

navigate(talk) {
  this.router.navigate(['/talk', talk.id]);
}
```

- Highlight active route links by setting a class

```
<a routerLink="/login" routerLinkActive="active">Login</a>
```

Route Subscription

- You can also **subscribe** to route parameter changes
  - Prevents recreating and redrawing the component when only a parameter changes

```
ngOnInit() {
  this.sub = this.route.params.subscribe(params => {
    let id = params['id'];
  });
}

ngOnDestroy() {
  this.sub.unsubscribe();
}
```

RouterModule and Routes

- RouterModule is a separate module in angular that provides required services and directives to use routing and navigation in angular application. Routes defines and array of roots that map a path to a component. Paths are configured in module to make available it globally. To use RouterModule and Routes in module

- const routes: Routes = [
- { path: 'manage-book', component: ManageBookComponent },
- { path: 'update-book/:id', component: UpdateBookComponent },
- { path: '', redirectTo: '/manage-book ', pathMatch: 'full' },
- { path: '**', component: PageNotFoundComponent }
-]

- imports: [RouterModule.forRoot(routes)]

Router

- It is used to navigate from one component to another component.

- import { Router } from '@angular/router';
- ....
- constructor(private router: Router) { }
- ...
- this.router.navigate(['/update-book', id]);

Location

- Location is a service that is used to interact with browser URL for example navigating back and forward. Location has methods such as go(), forward() and back() etc.

- import { Location } from '@angular/common';
- ...
- constructor(private location: Location) { }
- ...
- this.location.back();

RouterLink and RouterLinkActive

- RouterLink is a directive that is used to bind a route with clickable HTML element. RouterLinkActive is a directive that is used to add or remove CSS classes. When the HTML element with RouterLink binding is clicked then the CSS classes bound with RouterLinkActive will be active.

  - <a routerLink="/manage-book" routerLinkActive="active-link">Manage Book
  - <a [routerLink]="['/view-detail', book.id]">View Detail

- In the first link routerLink is bound with a route and routerLinkActive is bound with a CSS class. When this linked will be clicked then the associated CSS class will be activated. In the second link we are binding only routerLink with a parameter.

RouterOutlet

- RouterOutlet is a directive that is used as <router-outlet>. The role of <router-outlet> is to mark where the router displays a view.

- <nav [ngClass] = '"menu"'>
- <a routerLink="/home" routerLinkActive="active-link">Home |
- <a routerLink="/add-book" routerLinkActive="active-link">Add Book |
- <a routerLink="/manage-book" routerLinkActive="active-link">Manage Book
- </nav>
- <router-outlet></router-outlet>

- We have created menu items in the above code using RouterOutlet. They will be shown in every view where we navigate using the route binding with routerLink.

Service

- Angular services are injectable and injector can inject it in any component in our angular application. When we add our service in providers metadata of @NgModule in module file then the service becomes available globally in the application. To get instance of service in our component, we need to create a constructor with arguments of our service types in private scope. When we use @Injectable() decorator in service at class level then angular injector considers the service available to inject. A service contains methods that can be used by components.

- @Injectable()
- export class ItemService {}

- @NgModule({
- ...
- providers: [ItemService],
- ...
- })
- export class AppModule { }

HTTP Class

- Http performs HTTP request using XMLHttpRequest as default backend. Http is injectable. For any request it returns the instance of Observable. Http has following methods.
- post: Performs HTTP POST request.
- get: Performs HTTP GET request.
- put: Performs HTTP PUT request.
- delete: Performs HTTP DELETE request.
- patch: Performs HTTP PATCH request.
- head: Performs HTTP HEAD request.
- options: Performs HTTP OPTIONS request.
- request: Performs any type of HTTP request.

CRUD Operation using Http

- Now we will perform CREATE, READ, UPDATE and DELETE (CRUD) operation using Http. We will perform
- 1. CREATE operation using Http.post method.
- 2. READ operation using Http.get method.
- 3. UPDATE operation using Http.put method.
- 4. DELETE operation using Http.delete method.

HTTP POST

- post(url: string, body: any, options?: RequestOptionsArgs) : Observable<Response>

- createArticle(article: Article):Observable<number> {
- let cpHeaders = new Headers({ 'Content-Type': 'application/json' });
- let options = new RequestOptions({ headers: cpHeaders });
- return this.http.post(this.articleUrl, article, options)
- .map(success => success.status)
- .catch(this.handleError);
- }

We are passing header Content-Type as application/json. After successful operation we are returning status code as an instance of Observable. Headers is the angular class that is used to configure request headers.

HTTP GET

- get(url: string, options?: RequestOptionsArgs) : Observable<Response>

- getArticleById(articleId: string): Observable<Article> {
- let cpHeaders = new Headers({ 'Content-Type': 'application/json' });
- let cpParams = new URLSearchParams();
- cpParams.set('id', articleId);
- let options = new RequestOptions({ headers: cpHeaders, params: cpParams });
- return this.http.get(this.articleUrl, options)
- .map(this.extractData)
- .catch(this.handleError);
- }

HTTP PUT

- put(url: string, body: any, options?: RequestOptionsArgs) : Observable<Response>

- updateArticle(article: Article):Observable<number> {
- let cpHeaders = new Headers({ 'Content-Type': 'application/json' });
- let options = new RequestOptions({ headers: cpHeaders });
- return this.http.put(this.articleUrl +"/"+ article.id, article, options)
- .map(success => success.status)
- .catch(this.handleError);
- }

HTTP DELETE

- delete(url: string, options?: RequestOptionsArgs) : Observable<Response>

- deleteArticleById(articleId: string): Observable<number> {
- let cpHeaders = new Headers({ 'Content-Type': 'application/json' });
- let options = new RequestOptions({ headers: cpHeaders });
- return this.http.delete(this.articleUrl +"/"+ articleId)
- .map(success => success.status)
- .catch(this.handleError);
- }

HTTP Header

- Headers is the angular class that is used to configure request headers.

- **append(name: string, value: string):** Appends a header to existing list of header values for a given header name.
- **set(name: string, value: string | string[]):** Sets or overrides header value for given name. delete(name: string): Deletes all header values for the given name.
- **get(name: string):** string: Returns first header that matches given name.
- **getAll(name: string):** string[]: Returns list of header values for a given name.

HTTP Header Ex.

- let myHeaders = new Headers();
- let myHeaders = new Headers({ 'Content-Type': 'application/json', 'Cache-Control': 'no-cache' });
-
- myHeaders.append('Accept', 'text/plain');
- myHeaders.append('Accept', ' application/xhtml+xml ');
- myHeaders.delete('Accept');
-
- let acceptHeader = myHeaders.get('Accept');
- let acceptHeaders = myHeaders.getAll ('Accept');
-
- myHeaders.set('Content-Type', 'application/json');
- myHeaders.set('Accept', 'text/plain');

URLSearchParams

- **URLSearchParams** creates the query string in the URL. It is a map-like representation of URL search parameters. Find its constructor syntax.

- **constructor(rawParams?: string, queryEncoder?: QueryEncoder)** Both arguments in the constructor are optional. Angular queryEncoder parameter is used to pass any custom QueryEncoder to encode key and value of the query string. By default QueryEncoder encodes keys and values of parameter using JavaScript encodeURIComponent() method.

URLSearchParams

- **append(param: string, val: string):** void:

  Appends parameter value to existing list of parameter values for a given parameter name.

  It is used to add values in multi-value fields or arrays in query string.

- **set(param: string, val: string):**

  Sets or overrides parameter value for given parameter name.

- **delete(param: string):** void:

  Deletes all parameter values for the given parameter name.

- **get(param: string):** string:

  In case of multi-value fields, it returns the first value for given parameter name.

- **getAll(param: string):** string[]:

  Returns list of values for a given parameter name.

URLSearchParams Ex.

- let myParams = new URLSearchParams();
- myParams.append('names', 'John');
- myParams.append('names', 'David');
-
- //?names[]=John&names[]=David //Output
- myParams.set('names', 'Bob');
- let namesParam = myParams.getAll('names');

RequestOptionsArgs and RequestOptions

- **RequestOptionsArgs** is an interface that is used to construct a RequestOptions. The fields of RequestOptionsArgs are url, method, search, params, headers, body, withCredentials, responseType.

- **RequestOptions** is used to create request option. It is instantiated using RequestOptionsArgs. It contains all the fields of the RequestOptionsArgs interface. Now find the constructor of RequestOptions class.

HTTP Request / Header / Option Ex.

```
getBooksAfterFilter(catg: string, wtr: string): Observable < Book[] > {
  let myHeaders = new Headers();
  myHeaders.set('Content-Type', 'application/json');
  myHeaders.set('Accept', 'text/plain');
  let myParams = new URLSearchParams();
  myParams.set('category', catg);
  myParams.set('writer', wtr);
  let options = new RequestOptions({ headers: myHeaders, params: myParams });
  return this.http.get(this.url, options)
    .map(this.extractData)
    .catch(this.handleError);
}
```

RxJS and Observable

- Observable is a RxJS API. Observable is a representation of any set of values over any amount of time. All angular Http methods return instance of Observable. Find some of its operators.
- map: It applies a function to each value emitted by source Observable and returns finally an instance of Observable.
- catch: It is called when an error is occurred. catch also returns Observable.

- To fetch data from instance of Observable we need to subscribe it using RxJS subscribe operator. The actual hit to server goes only when we call subscribe or use async pipe on instance of Observable. In our example we will use subscribe operator to fetch data from Observable. Find the sample code.

RxJS and Observable (Continuous)

```
this.articleService.getAllArticles()
    .subscribe(articles => {
        //Data from server has been received.
        //perform operation on articles
      }
    );
```

Form Control, Form Group, Form Array

- **FormGroup** takes part in creating **reactive form**. FormGroup is used with **FormControl** and **FormArray**. The role of FormGroup is to track the value and validation state of form control. In our example we will create a form that will include <input> and <select> element. We will see how to instantiate FormGroup and how to set and access values in FormGroup. We will also validate the form.

- **FormControl:** It is a class that tracks the value and validation state of a form control.
- **FormGroup:** It is a class that tracks the value and validity state of a group of FormControl.
- **FormArray:** It is a class that tracks the value and validity state of array of FormControl, FormGroup and FormArray.

Form Control, Form Group, Form Array (Prop)

- FormControlName: It is a directive that syncs a FormControl in an existing FormGroup to a form control by name.
- FormGroupDirective: It is a directive that binds an existing FormGroup to a DOM element.
- FormGroupName: It is a directive that syncs a nested FormGroup to a DOM element.
- FormArrayName: It is a directive that syncs a nested FormArray to a DOM element.

Reference

- (Angular CLI)
  - https://github.com/angular/angular-cli
- (Angular CLI Wiki)
  - https://github.com/angular/angular-cli/wiki
- (Angular Redux)
  - https://github.com/ivanderbu2/angular-redux
- (Type Script)
  - https://tutorialzine.com/2016/07/learn-typescript-in-30-minutes
- (JWT and Authenticate)
  - https://github.com/auth0/angular2-jwt
- (RxJS and Reactive Programing)
  - https://www.babelcoder.com/blog/posts/rxjs-observables
- (Mock server API)
  - https://github.com/typicode/json-server

Reference

- (NGX-PIPES (template Helper))
  - https://github.com/danrevah/ngx-pipes#object
- (My Datepicker (Date picker Plugin))
  - https://github.com/kekeh/mydatepicker
- (Bulma CSS Framework)
  - http://bulma.io/documentation/overview/start/
- (Font Awesome ICON)
  - http://fontawesome.io/
- (Netbean Plugin)
  - https://github.com/Everlaw/nbts/releases

Reference

- Lazy Loading
  - https://medium.com/@flyingfree/%E0%B9%80%E0%B8%9E%E0%B8%B4%E0%B9%88%E0%B8%A1%E0%B8%9B%E0%B8%A3%E0%B8%B0%E0%B8%AA%E0%B8%B4%E0%B8%97%E0%B8%98%E0%B8%B4%E0%B8%A0%E0%B8%B2%E0%B8%9E%E0%B8%81%E0%B8%B2%E0%B8%A3%E0%B8%97%E0%B8%B3%E0%B8%87%E0%B8%B2%E0%B8%99-%E0%B9%83%E0%B8%AB%E0%B9%89-angular-%E0%B8%94%E0%B9%89%E0%B8%A7%E0%B8%A2-lazy-loading-6d2fa60431ec

- NG Rx (Angular Store)
  - https://medium.com/@flyingfree/%E0%B8%8B%E0%B8%B5%E0%B8%A3%E0%B8%B5%E0%B8%A2%E0%B9%8C-%E0%B8%88%E0%B8%B1%E0%B8%94%E0%B8%81%E0%B8%B2%E0%B8%A3-state-%E0%B9%83%E0%B8%99-angular-%E0%B8%94%E0%B9%89%E0%B8%A7%E0%B8%A2-ngrx-b40f155f086a

THE END

THANK