Cheat Sheet for comprehensive Angular

Angular CLI Commands

- **ng new**

- `ng new <project-name>`: Creates a new Angular project.
- `ng new <project-name> --routing`: Adds routing to the project.
- `ng new <project-name> --style=scss`: Sets SCSS as the default style.

- **ng generate**

- `ng generate component <component-name>`: Generates a new component.
- `ng generate directive <directive-name>`: Generates a new directive.
- `ng generate service <service-name>`: Generates a new service.
- `ng generate pipe <pipe-name>`: Generates a new pipe.
- `ng generate module <module-name>`: Generates a new module.
- `ng generate class <class-name>`: Generates a new class.
- `ng generate interface <interface-name>`: Generates a new interface.
- `ng generate enum <enum-name>`: Generates a new enum.

- **ng serve**

- `ng serve`: Starts the development server.
- `ng serve --open`: Starts the server and opens the app in the default browser.
- `ng serve --port <port-number>`: Specifies a custom port.

- **ng build**

- `ng build`: Builds the project for production.
- `ng build --prod`: Builds the project with production optimizations.
- `ng build --configuration=<config-name>`: Builds the project with a specific configuration.

- **ng test**

- `ng test`: Runs unit tests.
- `ng test --code-coverage`: Generates code coverage report.
- `ng test --watch=false`: Runs tests once without watching for changes.

- **ng e2e**

- `ng e2e`: Runs end-to-end tests.
- `ng e2e --protractor-config=<config-file>`: Specifies a custom Protractor configuration file.

Component Lifecycle Hooks

- ngOnChanges

- Called after a bound input property changes.
- Example:

```
ngOnChanges(changes: SimpleChanges) {
  console.log(changes);
}
```

- ngOnInit

- Called once, after the first `ngOnChanges`.
- Example:

```
ngOnInit() {
  console.log('Component initialized');
}
```

- ngDoCheck

- Called during every change detection run.
- Example:

```
ngDoCheck() {
  console.log('Change detection run');
}
```

- ngAfterContentInit

- Called after content (ng-content) has been projected into the view.
- Example:

```
ngAfterContentInit() {
  console.log('Content initialized');
}
```

- ngAfterContentChecked

- Called every time the projected content has been checked.
- Example:

```
ngAfterContentChecked() {
  console.log('Content checked');
}
```

- ngAfterViewInit

- Called after the component's view (and child views) has been initialized.
- Example:

```
ngAfterViewInit() {
  console.log('View initialized');
}
```

- ngAfterViewChecked

- Called every time the view (and child views) has been checked.
- Example:

```
ngAfterViewChecked() {
  console.log('View checked');
}
```

- ngOnDestroy

- Called once, before the instance is destroyed.
- Example:

```
ngOnDestroy() {
  console.log('Component destroyed');
}
```

Data Binding

- Interpolation

- Syntax: `{{ expression }}`
- Example:

```
<p>{{ message }}</p>
```

- Property Binding

- Syntax: `[property]="expression"`
- Example:

```
<img [src]="imageUrl">
```

- **Event Binding**

- Syntax: `(event)="statement"`
- Example:

```
<button (click)="onButtonClick()">Click me</button>
```

- **Two-Way Data Binding**

- Syntax: `[(ngModel)]="property"`
- Example:

```
<input [(ngModel)]="name">
```

Directives

- **Structural Directives**

- `*ngIf`
- Example:

```
<div *ngIf="isVisible">Content</div>
```

- `*ngFor`
- Example:

```
<ul>
  <li *ngFor="let item of items">{{ item }}</li>
</ul>
```

- `*ngSwitch`
- Example:

```
<div [ngSwitch]="condition">
  <p *ngSwitchCase="'A'">Case A</p>
  <p *ngSwitchCase="'B'">Case B</p>
  <p *ngSwitchDefault>Default</p>
</div>
```

- Attribute Directives

- `ngClass`
- Example:

```
      <div [ngClass]="{'active': isActive, 'disabled':
isDisabled}">Content</div>
```

- `ngStyle`
- Example:

```
      <div [ngStyle]="{'color': textColor, 'font-size': fontSize +
'px'}">Content</div>
```

Pipes

- Built-in Pipes

- `date`
- Example:

```
<p>{{ today | date:'shortDate' }}</p>
```

- `uppercase`
- Example:

```
<p>{{ message | uppercase }}</p>
```

- `lowercase`
- Example:

```
<p>{{ message | lowercase }}</p>
```

- `currency`
- Example:

```
<p>{{ amount | currency:'USD' }}</p>
```

- `percent`
- Example:

```
<p>{{ rate | percent }}</p>
```

- **Custom Pipes**

- Example:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'reverse'
})
export class ReversePipe implements PipeTransform {
  transform(value: string): string {
    return value.split('').reverse().join('');
  }
}
```

Services and Dependency Injection

- **Creating a Service**

- Example:

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class DataService {
  getData(): string {
    return 'Data from service';
  }
}
```

- **Injecting a Service**

- Example:

```
import { Component } from '@angular/core';
import { DataService } from './data.service';

@Component({
  selector: 'app-root',
  template: `<p>{{ data }}</p>`
})
export class AppComponent {
  data: string;

  constructor(private dataService: DataService) {
    this.data = this.dataService.getData();
  }
}
```

Routing

- Basic Routing

- Example:

```
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

- Route Parameters

- Example:

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

- Accessing Parameters:

```
import { ActivatedRoute } from '@angular/router';

constructor(private route: ActivatedRoute) {
  this.route.params.subscribe(params => {
    console.log(params['id']);
  });
}
```

- **Navigation**

- Example:

```
import { Router } from '@angular/router';

constructor(private router: Router) {}

navigateToAbout() {
  this.router.navigate(['/about']);
}
```

Forms

- **Template-Driven Forms**

- Example:

```
<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm)">
  <input name="name" ngModel required>
  <button type="submit">Submit</button>
</form>
```

- **Reactive Forms**

- Example:

```
    import { FormBuilder, FormGroup, Validators } from
'@angular/forms';

    export class MyComponent {
      myForm: FormGroup;

      constructor(private fb: FormBuilder) {
        this.myForm = this.fb.group({
          name: ['', Validators.required]
        });
      }

      onSubmit() {
        console.log(this.myForm.value);
      }
    }
```

Observables and RxJS

- Creating an Observable

- Example:

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

const myObservable = new Observable(observer => {
  observer.next('Hello');
  observer.next('World');
  observer.complete();
});
```

- Subscribing to an Observable

- Example:

```
myObservable.subscribe(
  value => console.log(value),
  error => console.error(error),
  () => console.log('Completed')
);
```

- Common RxJS Operators

- `map`
- Example:

```
import { of } from 'rxjs';
import { map } from 'rxjs/operators';

of(1, 2, 3).pipe(map(x => x * 2)).subscribe(console.log);
```

- `filter`
- Example:

```
of(1, 2, 3).pipe(filter(x => x > 1)).subscribe(console.log);
```

- `switchMap`
- Example:

```
import { fromEvent } from 'rxjs';
import { switchMap, map } from 'rxjs/operators';

fromEvent(document, 'click').pipe(
  switchMap(() => interval(1000).pipe(map(x => x * 2)))
).subscribe(console.log);
```

Angular Modules

- AppModule

- Example:

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

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

- Feature Modules

- Example:

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { UserComponent } from './user.component';

@NgModule({
  declarations: [UserComponent],
  imports: [CommonModule],
  exports: [UserComponent]
})
export class UserModule {}
```

Testing

- Unit Testing with Jasmine and Karma

- Example:

```
import { TestBed } from '@angular/core/testing';
import { MyService } from './my.service';

describe('MyService', () => {
  let service: MyService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(MyService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

- End-to-End Testing with Protractor

- Example:

```
describe('Protractor Demo App', function() {
  it('should have a title', function() {
    browser.get('http://juliemr.github.io/protractor-demo/');
    expect(browser.getTitle()).toEqual('Super Calculator');
  });
});
```

Tips and Tricks

- Lazy Loading Modules

- Example:

```
const routes: Routes = [
  { path: 'lazy', loadChildren: () =>
import('./lazy/lazy.module').then(m => m.LazyModule) }
];
```

- AOT Compilation

- Use `--aot` flag with `ng build` or `ng serve` for Ahead-of-Time compilation.

- **Environment Configuration**

- Use `environment.ts` and `environment.prod.ts` for different configurations.

- **Angular Material**

- Install Angular Material:

```
ng add @angular/material
```

- Example:

```
import { MatButtonModule } from '@angular/material/button';

@NgModule({
  imports: [MatButtonModule]
})
export class AppModule {}
```

- **Performance Optimization**

- Use `trackBy` in `*ngFor` to improve performance:

```
<div *ngFor="let item of items; trackBy: trackByFn">
  {{ item }}
</div>
```

- Example:

```
trackByFn(index, item) {
  return item.id;
}
```

- **Error Handling**

- Use `catchError` operator in RxJS:

```
import { catchError } from 'rxjs/operators';
import { of } from 'rxjs';

myObservable.pipe(
  catchError(error => {
    console.error(error);
    return of([]);
  })
).subscribe();
```

- Debugging

- Use Angular DevTools for debugging Angular applications.
- Use `ng.probe` in the browser console for debugging components.

Conclusion

This cheat sheet covers the essential features, commands, and best practices for working with Angular. Use these tips and tricks to build robust, scalable, and maintainable Angular applications.

By Ahmed Baheeg Khorshid

ver 1.0