Angular Cheat Sheet

A comprehensive cheat sheet for Angular, covering essential concepts, syntax, and commands for efficient development.

Core Concepts

Components

Definition: Reusable UI building blocks with a template (HTML), a class (TypeScript), and metadata.

Key Features:

- Encapsulation of data, logic, and view.
- Supports input (`@Input`) and output (`@Output`) properties for communication.
- Lifecycle hooks for managing component behavior.

Creating a Component:

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

@Component({
  selector: 'app-my-component',
  templateUrl: './my-
component.component.html',
  styleUrls: ['./my-component.component.css']
})
export class MyComponentComponent {
  @Input() data: any;
  @Output() action = new EventEmitter<any>();

  onClick() {
    this.action.emit(this.data);
  }
}
```

Using a Component:

```
<app-my-component [data]="item"
(action)="handleAction($event)"></app-my-
component>
```

Modules

Definition: Containers that group related components, directives, services, and other modules.

Key Features:

- Organize application into functional units.
- Define dependencies and providers.
- Enable lazy loading for improved performance.

Creating a Module:

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

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

Services

Definition: Reusable classes that encapsulate business logic, data access, or other functionalities.

Key Features:

- Promote code reusability and maintainability.
- Dependency injection for loose coupling.
- Singleton pattern by default.

Creating a Service:

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

@Injectable({
  providedIn: 'root',
})
export class MyService {
  getData() {
    return ['item1', 'item2', 'item3'];
  }
}
```

Using a Service:

```
import { Component, OnInit } from
'@angular/core';
import { MyService } from './my.service';

@Component({
  selector: 'app-consumer',
  template: `<div>{{ items }}</div>`,
})
export class ConsumerComponent implements
OnInit {
  items: string[] = [];
  constructor(private myService: MyService) {}

  ngOnInit() {
    this.items = this.myService.getData();
  }
}
```

Templates & Data Binding

Template Syntax

Interpolation:	`{{ expression }}` - Displays expression value in the view. **Example:** `{{ title }}`
Property Binding:	`[property]="expression"` - Sets element property to expression value. **Example:** `<img [src]="imageUrl">`
Event Binding:	`(event)="handler"` - Binds element event to a component method. **Example:** `<button (click)="onClick()">Click Me</button>`
Two-way Binding:	`[(ngModel)]="property"` - Enables two-way data binding for form elements. Requires `FormsModule`. **Example:** `<input [(ngModel)]="name">`

Directives

Structural Directives:	Modify the DOM layout by adding, removing, or replacing elements. • `*ngIf`: Conditionally includes a template. • `*ngFor`: Repeats a template for each item in a collection. • `*ngSwitch`: Conditionally displays a template based on a switch expression.
Attribute Directives:	Change the appearance or behavior of an element, component, or another directive. • `[ngClass]`: Adds or removes CSS classes. • `[ngStyle]`: Sets inline styles. • `ngModel`: Implements two-way data binding.
Example - ngIf:	`<div *ngIf="isVisible">This is visible</div>`
Example - ngFor:	`<div *ngFor="let item of items">{{ item }}</div>`

Routing & Navigation

Routing Module

Setting up the Router:

1. Import `RouterModule` and define routes.
2. Configure routes array with `path` and `component`.
3. Add `<router-outlet>` in the root component template.
4. Use `routerLink` directive for navigation.

Example:

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

const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: '', redirectTo: '/home', pathMatch: 'full' },
];

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

Navigation:

```
<a routerLink="/home" routerLinkActive="active">Home</a>
<a routerLink="/about" routerLinkActive="active">About</a>
<router-outlet></router-outlet>
```

Route Parameters

Definition:	Passing data through the URL.
Usage:	Define parameters in the route configuration. Access parameters using `ActivatedRoute`.
Example Route:	`{ path: 'product/:id', component: ProductComponent }`

Accessing Parameter:	`import { ActivatedRoute } from '@angular/router';`
	
	`constructor(private route: ActivatedRoute) {`
	`  this.route.params.subscribe(params => {`
	`  const id = params['id'];`
	`  });`
	`}`

Forms

Template-Driven Forms

Overview:

- Relies on directives like `ngModel` for data binding.
- Simpler to implement for basic forms.
- Validation is primarily done in the template.

Example:

```
<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm)">
  <input type="text" name="name" ngModel required>
  <div *ngIf="myForm.controls['name']?.invalid &&
myForm.controls['name']?.touched">
    Name is required.
  </div>
  <button type="submit" [disabled]="myForm.invalid">Submit</button>
</form>
```

Reactive Forms

Overview:

- Form structure defined in the component class.
- More control over form elements and validation.
- Suitable for complex forms and dynamic validation.

Example:

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

export class MyComponent {
  myForm = new FormGroup({
    name: new FormControl('', Validators.required),
    email: new FormControl('', [Validators.required, Validators.email])
  });

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

```
<form [formGroup]="myForm" (ngSubmit)="onSubmit()">
  <input type="text" formControlName="name">
  <div *ngIf="myForm.controls['name']?.invalid &&
myForm.controls['name']?.touched">
    Name is required.
  </div>
  <input type="email" formControlName="email">
  <div *ngIf="myForm.controls['email']?.invalid &&
myForm.controls['email']?.touched">
    Invalid email.
  </div>
  <button type="submit" [disabled]="myForm.invalid">Submit</button>
</form>
```