JavaScript Frameworks Cheatsheet

A comprehensive cheat sheet covering essential JavaScript frameworks, their core concepts, syntax, and usage examples. This guide helps you navigate the landscape of modern web development with frameworks like React, Angular, and Vue.js.

React Basics

Core Concepts

Components: Reusable UI building blocks. Can be functional or class-based.
JSX: JavaScript XML, a syntax extension that allows writing HTML-like code in JavaScript.
State: Data that can change over time and trigger re-renders. Managed within components.
Props: Data passed from parent to child components. Read-only from the child's perspective.

Functional Components

```
import React from 'react';

function MyComponent(props) {
  return (
    <h1>Hello, {props.name}</h1>
  );
}

export default MyComponent;
```

Class Components

```
import React from 'react';

class MyComponent extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

export default MyComponent;
```

State Management (useState Hook)

```
import React, { useState } from 'react';

function MyComponent() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count +
1)}>Increment</button>
    </div>
  );
}

export default MyComponent;
```

Props Example

```
// Parent Component
import React from 'react';
import ChildComponent from './ChildComponent';

function ParentComponent() {
  return (
    <ChildComponent name="John" />
  );
}

export default ParentComponent;

// ChildComponent.js
import React from 'react';

function ChildComponent(props) {
  return (
    <h1>Hello, {props.name}</h1>
  );
}

export default ChildComponent;
```

Angular Essentials

Key Concepts

Modules: Organize application functionality.
Components: Building blocks of the UI, consisting of a template, class, and metadata.
Templates: HTML-like code that defines the component's view.
Services: Reusable logic providers that can be injected into components.
Directives: Extend HTML with custom behavior.

Component Example

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

@Component({
  selector: 'app-my-component',
  templateUrl: './my-
component.component.html',
  styleUrls: ['./my-component.component.css']
})
export class MyComponent {
  name: string = 'Angular';
}
```

Template Example

```
<h1>Hello, {{ name }}</h1>
<p>Current time: {{ currentTime |
date:'medium' }}</p>
```

Services Example

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

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

// In component:
import { DataService } from './data.service';

constructor(private dataService: DataService)
{}

ngOnInit() {
  this.data = this.dataService.getData();
}
```

Directives

Structural Directives	*ngIf, *ngFor, *ngSwitch
	`<div *ngIf="isVisible">Show this</div>`
	`<li *ngFor="let item of items">{{ item }}</li>`

Attribute Directives	Used to change the appearance or behavior of an element.
	`import { Directive, ElementRef, Input } from '@angular/core';`
	`@Directive({ selector: '[appHighlight]' })`
	`export class HighlightDirective { constructor(private el: ElementRef) {} @Input('appHighlight') highlightColor: string; ngOnInit() { this.el.nativeElement.style.backgroundColor = this.highlightColor; } }`

Vue.js Fundamentals

Core Concepts

Components: Reusable Vue instances with a template, script, and style.
Templates: HTML-based templates that can bind data to the DOM.
Data Binding: Synchronizing data between the component and the view.
Directives: Special attributes that start with `v-` to add reactive behavior to the DOM.
Computed Properties: Dynamically calculated properties based on reactive data.

Component Example

```
Vue.component('my-component', {
  template: '<div>Hello, {{ name }}</div>',
  data() {
    return {
      name: 'Vue.js'
    }
  }
})

// Usage in HTML:
<my-component></my-component>
```

Data Binding

v-model: Two-way data binding for form inputs.
{{ }} (Mustache syntax): Text interpolation for displaying data.
v-bind: Dynamically bind attributes to expressions.

```
<input v-model="message">
<p>{{ message }}</p>
<img v-bind:src="imageURL">
```

Directives

v-if, v-else-if, v-else: Conditional rendering.
v-for: Loop through arrays or objects.
v-on: Listen to DOM events.
v-show: Toggle visibility based on a condition.

```
<div v-if="show">Visible</div>
<li v-for="item in items" :key="item.id">{{ item.name }}</li>
<button v-on:click="handleClick">Click me</button>
```

Computed Properties

```
computed: {
  fullName() {
    return this.firstName + ' ' + this.lastName;
  }
}

// Usage in template:
<p>{{ fullName }}</p>
```

Comparing Frameworks

Feature Comparison

Feature	React	Angular
Language	JavaScript/JSX	TypeScript
Data Binding	One-way	Two-way
Component Structure	Functional or Class	Component-based
Virtual DOM	Yes	Yes

Learning Curve

Framework	Pros	Cons
React	Relatively simple core concepts, large community, flexible.	Requires additional libraries for routing and state management.
Angular	Comprehensive framework, strong tooling, and clear structure.	Steeper learning curve, more verbose, opinionated.
Vue.js	Easy to learn, versatile, good for single-page applications.	Smaller community compared to React and Angular.