# Fazit Angular-Prototyp

## Einleitung

Im April 2019 wurde ein Prototyp mit Angular entwickelt. Eine lauffähige Version befindet sich auf meiner Github Seite und implementiert mit 2 UI Frameworks ( Elements UI,  Angular Material).

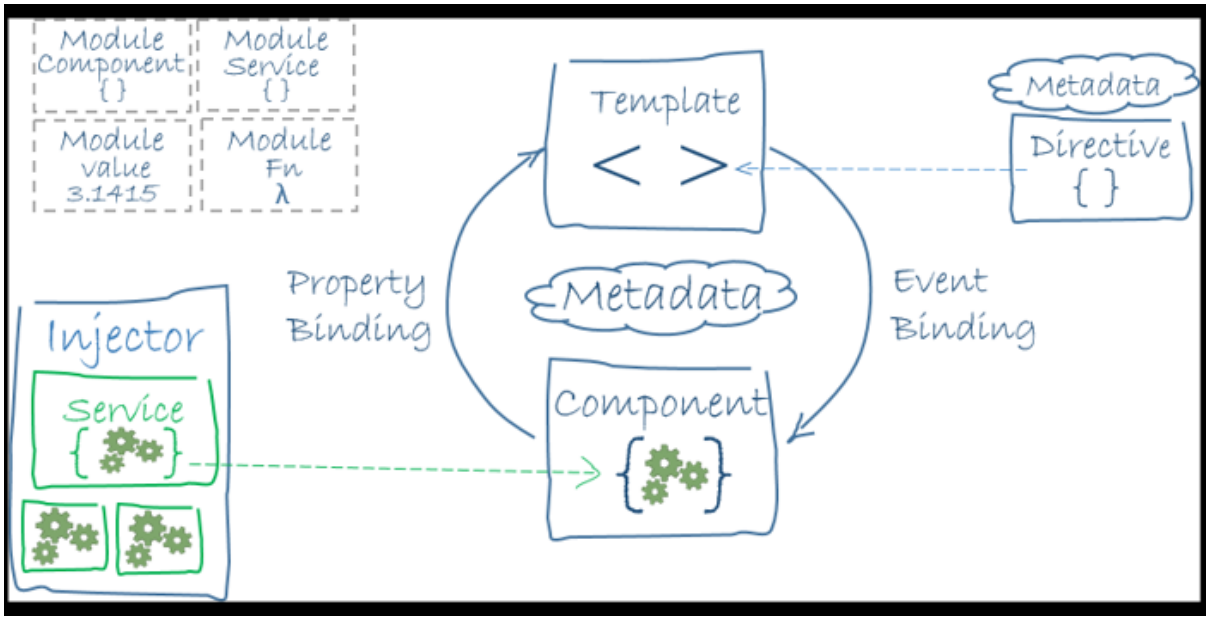
## Allgemeines

Angular ist ein Framework, da es Ihnen einen guten Einstieg in die Erstellung einer Anwendung mit dem vollständigen Setup bietet. Wir müssen uns nicht mit Bibliotheken, Routing-Lösungen und der Struktur befassen. Wir können einfach anfangen zu bauen.

Angular ist eine Plattform zum Erstellen von Clientanwendungen in HTML und **TypeScript**.

Die Grundbausteine einer Angular-Anwendung sind **NgModules**, die einen Kompilierungskontext für Komponenten bereitstellen. NgModules sammelt verwandten Code in Funktionssätzen.

Eine Angular-App wird durch eine Reihe von NgModules definiert. Eine App verfügt immer über mindestens ein **Root-Modul,** das das Bootstrapping ermöglicht, und normalerweise über viel mehr Funktionsmodule.




**Angular hat folgende basis Funktionen:**

1. Basis-Komponente, Klassen und Decorators(ähnlich Annotationen)
2. Property- und Event-Binding
3. Zwei-Wege-Datenbindung
4. Expressions
5. Direktiven und Komponenten
6. Schleifen mit *ngFor. IF Statement mit *ngIf
7. Pipes(Filter)
8. Services
9. Dependency injection
10. Observables
11. Component-Lifecycle
12. Interfaces
13. Routing

Ähnlich wie Vue.js hat Angular Built-In Directives, wie ***ngFor,  *ngIf, [ngClass], [ngStyle], [ngSwitch].** Und 2 way binding **[(ngModel)]**.

**Wichtige Class field decorators** für directives und components:

| | |
|---|---|
| **@Input() myProperty;** | **Deklariert eine Eingabeeigenschaft, die Sie über die Eigenschaftsbindung aktualisieren können** |

| | |
|---|---|
| @**Output()** myEvent = new EventEmitter(); | Deklariert eine Ausgabeeigenschaft, die Ereignisse auslöst, die Sie mit einer Ereignisbindung abonnieren können |
| @**ViewChild(myPredicate)** myChildComponent; | Bindet das erste Ergebnis der Komponentenansichtsabfrage (myPredicate) an eine Eigenschaft (myChildComponent) der Klasse. Nicht für directives verfügbar. |

**Angular Lifecycles in Reihnfolge:**

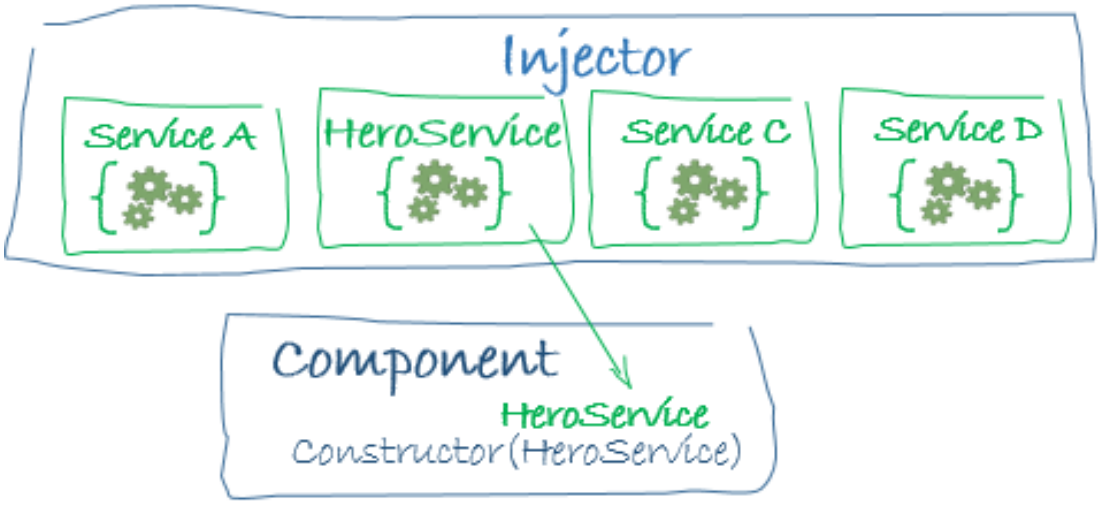

**Services and dependency injection:**

Service ist eine breite Kategorie, die alle Werte, Funktionen oder Features umfasst, die eine App benötigt. Ein Service ist normalerweise eine Klasse mit einem engen, genau definierten Zweck.

**Angular unterscheidet Komponenten von Services**, um die Modularität und Wiederverwendbarkeit zu erhöhen. Indem die ansichtsbezogenen Funktionen einer Komponente von anderen Verarbeitungsarten zu trennen.

Eine Komponente kann bestimmte Aufgaben an Dienste delegieren, z. B. das Abrufen von Daten vom Server, die Überprüfung von Benutzereingaben oder die direkte Protokollierung an der Konsole.

Indem man solche Verarbeitungsaufgaben in einer injectable service class definieren, stellt man diese Aufgaben jeder Komponente zur Verfügung.




**Testen von Komponenten mit Dependency**

Das Entwerfen einer Klasse mit Abhängigkeitsinjektion erleichtert das Testen der Klasse. Das Auflisten von Abhängigkeiten als Konstruktorparameter ist möglicherweise alles, was wir zum effektiven Testen von Anwendungsteilen benötigen.

Wir können beispielsweise eine neue HeroListComponent mit einem mock service erstellen, den wir im Test bearbeiten können.

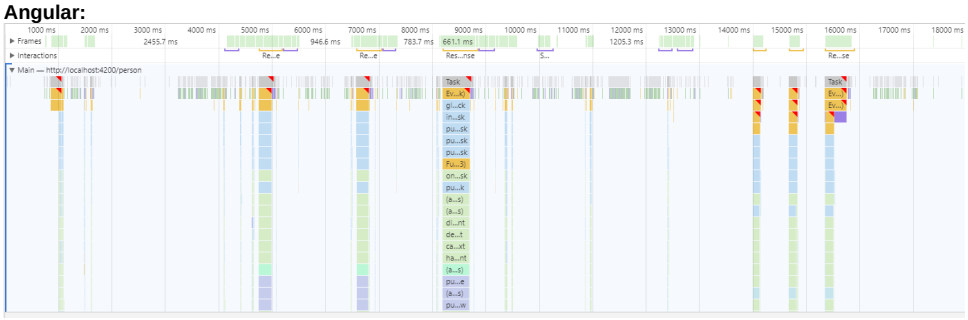
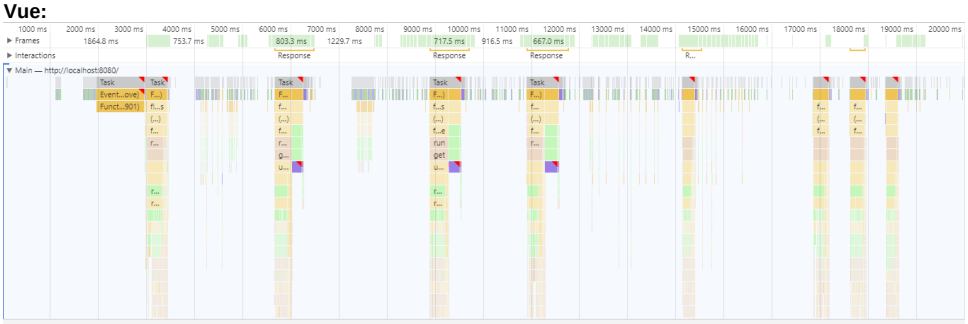
```
const expectedHeroes = [{name: 'A'}, {name: 'B'}]
const mockService = <HeroService> {getHeroes: () => expectedHeroes }

it('should have heroes when HeroListComponent created',
() => {  // Pass the mock to the constructor as the Angular injector would
 const component = new HeroListComponent(mockService);
 expect(component.heroes.length).toEqual(expectedHeroes.length);
});
```

**Angular bittet sich ganz viele CLI Commands an:**

| | | |
|---|---|---|
| add | | Adds support for an external library to your project. |
| build | b | Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from within a workspace directory. |
| config | | Retrieves or sets Angular configuration values in the angular.json file for the workspace. |
| doc | d | Opens the official Angular documentation (angular.io) in a browser, and searches for a given keyword. |
| e2e | e | Builds and serves an Angular app, then runs end-to-end tests using Protractor. |
| generate | g | Generates and/or modifies files based on a schematic. |
| help | | Lists available commands and their short descriptions. |
| lint | l | Runs linting tools on Angular app code in a given project folder. |
| new | n | Creates a new workspace and an initial Angular app. |
| run | | Runs an Architect target with an optional custom builder configuration defined in your project. |
| serve | s | Builds and serves your app, rebuilding on file changes. |
| test | t | Runs unit tests in a project. |
| update | | Updates your application and its dependencies. See https://update.angular.io/ |
| version | v | Outputs Angular CLI version. |
| xi18n | | Extracts i18n messages from source code. |

**Performance**

1. Untere Bilder sind jeweils getestet mit Angular Projekt und Vue-test Projekt (500 Einträge) under Development Mode. Allgemein ist beim Angular Projekt schneller.
2. **Angular:**
3. 
4. **Vue:**
5. 

Beim Mouce-Click von Tabelle von Angular ist besondes schneller, ist durchschnittlich 100ms, aber bei Vue-Test Projekt ist über 1000ms!

Beim Editieren von Tabellen ist auch bei Angular schneller, von Durchschnitt 200-250ms, bei Vue ist 350-400ms.

Beim Button Editiren ist auch bei Angular 200ms schneller als beim Vue.

Diese Performance Unterschiede bin ich der Meinung, dass es mit UI Framework zu tun ist. Beim Angular habe ich Material benutzt, aber beim Vue ist Element.

Material design und Angular sind vom Unternehmen Google Inc. entwickelte Designsprache. Deswegen ist beim Angular und Material mehr kompatibel und schneller.

## Zusätzliche Tools

1. Für den Prototypen wurde als Entwicklungsumgebung Visual Studio Code verwendet.

## Testing

Beim Erstellen von Angular-Projekten mit der Angular-CLI werden standardmäßig Unit-Tests mit **Jasmine und Karma** erstellt und ausgeführt. Die CLI übernimmt für Sie die Jasmine- und Karma-Konfiguration.

Die .**spec.ts-Datei** sind Komponententests für Ihre Quelldateien. Die Konvention für Angular-Anwendungen besteht darin, für jede **.ts-Datei** eine .**spec.ts-Datei** zu haben.

Sie werden mit dem Jasmine Javascript Test Framework über den Karma Test Runner (https://karma-runner.github.io/) ausgeführt, wenn man den Befehl **ng test** verwenden.

z.B nachdem ich Tests in .spec.ts-Datei geschrieben habe, ich kann einfach in Command line **ng test** laufen lassen, dann wird folgendes Test Ergebnis in meine Browser geöffnet: