# Abstract

In modern frontend web development, most developers use frameworks like Angular, React or Vue. On a basic level, all of these frameworks use so called web components: bundles of html, CSS and Typescript which are independent of the rest of the code and each represented by their own custom html tag. The idea is to pack chunks of code which are frequently used throughout a project (i.e. Product cards in a webshop) into reusable components. This reduces the amount of code needed, eliminates the necessety for copy-pasting code segements throughout a web page and promotes “dry” code.

However, since not every project will be built using the same framework and all these frameworks use components as their basic building blocks, it would be usefull to be able to build framework agnostic web components that can then be used by any framework. This way, components would only need to be created once instead of creating them in each framework separately.

The subject of this Thesis is therefore to answer the question of whether it is possible to create framework agnostic web components and use them in multiple frameworks without the need for considerable amounts of extra code and to shed some light on how this universal compatibility is achieved. As an example, StencilJS will be used to create web components and the frameworks Angular, React and Vue will be used to demonstrate the universal compatibility.

Apart from small example pages that contain Components created in StencilJs, a part of this Thesis is also going to be a real-world project; A tablet app for Therapists that displays essential patient data and helps with managing patients.

# Introduction

In der Modernen Web-Frontendentwicklung verwenden die meisten Entwickler Frameworks, wie zum Beispiel Angular, React, oder Vue. Grundsätzlich verwenden diese Frameworks alle sogenannte Web-Komponenten: voneinander unabhängige Blöcke von html, CSS und Typescript, welche einen eigenen html-tag bekommen. Die Idee dahinter ist es, Codeblöcke, die mehrmals in einem Projekt verwendet werden (z.B. Produktkarten in einem Webshop) in wiederverwendbaren Komponenten zusammenzufassen. Das reduziert die Menge an Code in einem Projekt und erleichtert das Schreiben von „dry“ Code.

Da man Allerdings nicht jedes Projekt mit dem selben Framework umsetzen wird, und alle diese Frameworks Komponenten als grundlegende Bausteine verwenden, wäre es nützlich, Frameworkagnostische Komopnenten erstellen zu können, welche dann in jedem beliebigen Framework verwendet werden könnten. Dadurch müsste man Web-Komponenten nur einmal erstellen, anstatt für jedes Framework einzeln.

Das Ziel dieser Arbeit ist es daher, festzustellen, ob es möglich ist, Frameworkagnostische Webkomponenten zu erstellen und diese in verschiedenen Frameworks ohne viel Boiler Plate Code zu verwenden.Als Beispiel wird Stenciljs verwendet, um Komponenten zu erstellen und die Frameworks Angular, React und Vue werden dienen Dazu, die Universale Verwendbarkeit der Komponenten zu demonstrieren.

Neben einigen Testseiten, die die in Stenciljs erstellten Componenten beinhalten wird auch ein Projekt Teil dieser Arbeit sein. Es geht dabei um eine Tablet-app für Therapeuten, welche Essenzielle Patientendaten schnell und einfach zugänglich macht und das Patientenmanagement unterstützt.

# Fundamentals and related work

For context it is important to elaborate some basics about web components and the frameworks they are used in. This means going into the basic architecture of each framework, comparing the structure of their web components and showing similarities between the framework specific components created in each of the frameworks and the framework agnostic components created in Stenciljs

## What is angular?

Angular was created by Google in 2010 and is described in the official documentation as a Typescript based platform that includes a framework to build web applications, as well as many helpful libraries and tools to streamline the entire process of developing and maintaining a web application [1]. This means that Angular is not just a framework, but also has a large number of tools around the framework itsself.

### Basic concepts

* **Components:** A component is basically a container for view. Views are sets of html, Typescript and CSS that together define sets of UI elements like buttons, paragraphs, labels, pictures, etc that can be altered, replaced, or removed entirely depending on the application’s logic. Components are the basic building blocks of an Angular application
* **Services:** A service is a part of a component that defines a specific behaviour or functionality and is usually written purely in Typescript. Services can be injected into components to provide functionality. This helps to make the code more modular and reuseable.
* **NgModules:** A module in Angular represents a collection of components and services that share a certain context.
* **Decorators:** An angular component is implemented as a Typescript class which can contain decorators with a certain type which provides metadata. This metadata in turn tells the angular compiler what how to use the decorators (for example @NgComponent to tell the compiler that the following class is a component).
* **Metadata:** as mentioned above, metadata tells the compiler what to do with a certain piece of code. To give a more specific example, the @NgComponent decorator’s metadata contains the location of the component’s html and CSS files. This way all files can be linked to a single component.
* **Templates:** Templates are an enhancement of html featured in Angular that allows a developer to inline some functionality like hiding UI-Elements, which would normally take several additional lines of Typescript and CSS. It works by placing html code in a <template> tag. The UI element(s) can then be altered using event binding.
* **Event binding:** Event binding is a way of responding to DOM events inside the html code. A goot example for this is the click event. By using it inside an html element like this:

<button (Click)="onClick()">Do something</button>

the button will call the onClick function when it is clicked. The function would be implemented in the component’s main class. It is worth mentioning that the data flow of event binding only goes one way; from the html to the typescript class.

* **Property binding:** Similar to event binding, but in this case a property or content of an html element to a property of the corresponding Typescript class. For example the path of an image can be given to an img tag with the following code:
  + html element: <img [src]="source">
  + inside the class: source:string = '../assets/image.jpg'

the data flow of property binding, just like event binding goes only one way, but in the opposit direction: from the typescript class to the html.

* **Directives:** Directives are also Typescript classes that are defined with the @Directive decorator and can be attached to DOM elements in order to apply a certain behaviour like changing the background color of a button while it is clicked. Directives are also modular and can be used multiple times

The graph below describes how these concepts work together.

Text

Description automatically generated with low confidence

Figure : Architecture of an Angular application [2]

### Structure of an Angular Component

In Angular a component is divided into four files:

* A Typescript file for the component’s class (i.e. myComp.component.ts)
* An html file for the visual repersentation (i.e. myComp.component.html)
* an SCSS file for styling (i.e. myComp.component.scss)
* a spec.ts (Typescript) file for testing purposes (i.e. myComp.spec.ts)

These files are linked together in the component.ts file.

# Bibliography

[1]

Author: Google  
Date Accessed: 31.10.2022  
Title: What is Angular?  
URL: <https://angular.io/guide/what-is-angular>

[2]

Author: Google   
Date Accessed: 31.10.2022  
Title: Introduction to Angular Concepts  
URL: <https://angular.io/guide/architecture>