A Survey of JavaScript UI Frameworks: Angular, React, Angular 2, Vue

Group 1 Linda Kolb, Markus Lienbacher, Matthias Stefan and Eva Ulbrich,

706.057 Information Visualisation SS 2016
Graz University of Technology

22 Nov 2016

Abstract

During the last five years the web has developed to a high extend to dynamic and nicely visualized homepages. Building Pages in components has become a new way of thinking. JavaScript has become the language that support those features. To overcome some restrictions and problems in usability, JavaScript UI Frameworks has developed.

In this paper four of them are compared: Angular, Angular 2, React and Vue. They make it easy to program Apps, which change the DOM automatically, gives some API and in some cases support extra programming languages that get compiled to JS. Component based View, html templates and ShadowDOM are features that at least Angular 2 and React support.

Angular 2 is the newest and the time will tell us where it's gonna end, AngularJS (Angular 1) was the most uncomfortable one, little outdated, VueJS is recommended for small projects because it's easy to include and React is good for including in existing projects and supports many features but gives the developer also many liberties.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

This survey examines web components and how they can be realised in four different JS UI Frameworks. The frameworks chosen for analysis are AngularJS, Angular 2, React and Vue. The can in a way be viewed as frameworks aiding in the fast development of parts of applications at the client side with the focus on modularity and reusability. The survey includes a short description of the various frameworks as well as a comparison. A small example project was implemented in all four frameworks. In addition, the survey examines the design principle of atomic design which answers the question “Which parts of an interface can be a component?”.

This survey is divided into the following chapters: Chapter 2 gives more insight on why web components can improve web application performance. Chapter 3 give a brief overview of atomic design and how modularity in terms of UI elements can be grouped. Chapter 4 contains a comparison of the JavaScript UI frameworks AngularJS, Angular 2, React and Vue. Furthermore, the realisation of the example project is added to each one of the frameworks.

Chapter 2

Web Components

With web projects becoming larger, it is getting harder to maintain integrity, and keeping elements from each other, so that they do not interfere with each other. One Solution for this are Web Components, which is a standard currently developed by the W3C [*Web Components Current Status - W3C* 2016]. It consists of mainly 4 Elements: A Shadow DOM, Custom Elements, HTML Import and Templates.

Shadow DOM Tree

The Document Object Model represents all the information and data of a homepage. This information is usually accessible by JavaScript to make changes, retrieve information or generating new nodes. Browser developers started to use html internally for common objects like sliders, buttons, etc. The reason for this is not to have hardcoded look and behaviour in the browser, but instead use the tools that are already available. To hide that abstraction, this inner elements do appear in the rendering tree, but not in the DOM tree. [*What the Heck is Shadow DOM?* 2016] Web Developers saw that you could reach the shadow dom with JavaScript, and this also could be used for encapsulation and making some sort of components. With web components this now becomes standard behaviour.

In Comparison to the Web Components Standard, some frameworks also implement a so called virtuall DOM. Rendering the DOM can be very time consuming, so they keep a copy of the DOM with all the changes, and can inject only the changes that are needed to the DOM for rerendering.

Custom Elements

With Custom Elements, can introduce new HTML Elements and tell the parser how to build them.

HTML Import

While earlier, other HTML pages had to be included in, for example, iFrames, there is now a possibility to include other HTML files directly.

Templates

Templates enable developers to declare DOM Subtrees or fragments with identical content, that can be inserted and used anywhere in the HTML document.

Chapter 3

Atomic Design

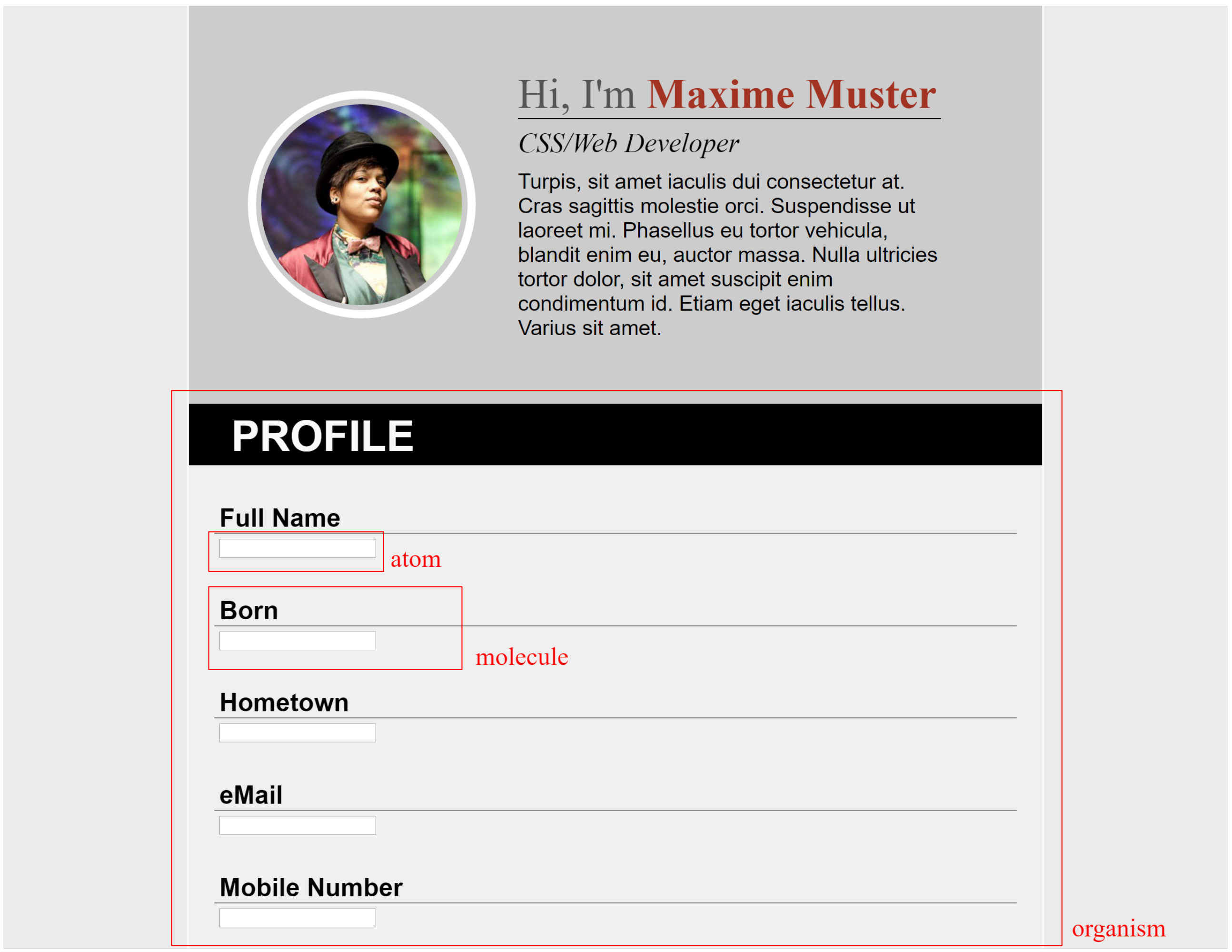
Atomic Design was introduced in 2013 by front-end designer Brad Frost. It is a design principle based on the idea of dividing a web application's interface into elements. It facilitates project collaboration, design consistency and reusability. The bottom-up approach brings structure into the development of a web project, especially regarding responsiveness. Splitting web projects into smaller parts and then combining them to more complex units makes it easier to create responsive web applications. It facilitates client meetings since example pages of the web application can already be shown at an early stage of the development phase. In addition, prototyping is easy and fast since the developer only needs to combine the already defined atoms, molecules and organisms to a template.

According to Frost [2016] , atomic design splits user interface development into the following elements which are oriented toward chemistry:

- Atoms: Atoms are basically HTML tags, such as an input field or a button. An atom could be realised as a very basic web component.
- Molecules: A module is a combination of multiple atoms. An example would be the search function of a website which consists of an input field and a button. This search function could be implemented as a simple web component.
- Organisms: Organisms consist of various molecules, thus a more complex web component.
- Templates: A template describes the layout of a web page. Placeholders and filler texts are used to show how atoms, molecules and organisms are combined to one system.
- Pages: Pages are actual instances of the templates. Placeholders and filler texts are replaced with the actual content of the web page.

Figure 3.1 visualises a template following atomic design. A single input field would be an atom, a input field with a label would be a molecule and all input fields and labels inside the profile section can be combined to an organism.

To maintain the elements of a web application, style guides can be used. A style guide is basically a documentation of all components used for a specific web application. It can be realised with a static HTML site showing examples for atoms, molecules and organisms used in the web application. The elements of the web application can even be presented with the associated code. Using style guides helps with consistency, extension and maintenance of the web application.



Hi, I'm **Maxime Muster**

CSS/Web Developer

Turpis, sit amet iaculis dui consectetur at. Cras sagittis molestie orci. Suspendisse ut laoreet mi. Phasellus eu tortor vehicula, blandit enim eu, auctor massa. Nulla ultricies tortor dolor, sit amet suscipit enim condimentum id. Etiam eget iaculis tellus. Varius sit amet.

PROFILE

Full Name

Born

Hometown

eMail

Mobile Number

Figure 3.1: An example showing atoms, molecules and organisms according to atomic design. The image as a whole would be a template consisting of various organisms.

Chapter 4

JS UI Frameworks for Web Components

This survey examines four libraries aiding in creating front ends. They follow the mindset that a browser is not only a tool to display document data but also the front end of a more complex application like a webshop or a word processor or even more complex applications shifting the focus of maintainability of software away from users but moving the rendering and input processing to front ends. This shifts the traditional roles of users using software on their computers to servers as well as servers preprocessing and then delivering content to users. The frameworks in this survey can in a way be viewed as frameworks aiding in the fast development of parts of applications at the client side with the focus on modularity and reusability. The role of script languages became more and more important with the rise of interactivity of web sites. Web sites turned into web applications with rising complexities. JS does not always provide enough flexibility and reusability to enable fast development and often JS ends up in being an island solution to a single usecase that can not be tested efficiently and thus is error prone. The presented JS frameworks were developed with the Model View Controller (MVC) philosophy in mind allowing to use a well known script language and more modularity and especially reusability during coding[Seshadri and Green, 2014].

In 1990, HTML was developed to display document content to share ideas. HTML at that time was not capable of interactions with and input from users it but only able to organise a document structure. Three years later with the introduction of the Common Gateway Interface the content became more dynamic and since 1999 scripting languages like Flash and Silverlight as well as JS interpret interactions with websites and even build HTML tags[Branas, 2014].

The increase of the use of images and JS due to a change from simple linked HTML pages to entire web applications caused a rise of website file sizes as well as communication with webservers. Especially interactions with users require the processing of input and the display of the results of their actions that can result in complex processes. One example where one website is affected is the use of a login form at the checkout of an eCommerce marketplace where name and address as well as billing information has to be validated as well as the ordered goods and the order itself stored. Another example affecting not only the interaction with one website but even more complex could be the use of a form to insert information in social media portals including sending information who the users friends are and who should see that information plus what kind of advertisement should be shown to both the user and the user's friends. The advertisement is based on the user's initial input and other information like the browser history stored from search engines.

One approach to tackle the challenge of increasing script transfer at at one website and thus waiting time until scripts loaded and changed what users see on their screens is the development of so called single page web applications. Web applications can be built by using JS frameworks and some of them specialize in taking care of data-binding or templates.

The presented frameworks in this survey were developed to realize single web applications to allow asynchronous data loading to allow generating and rendering a Document Object Model on the client side thus reducing traffic instead of hyperlinking and fetching content from servers. Traditionally, rendering of a DOM is a one way process where data from a server and a design template result in a HTML structure showing users the state of a website. Traffic between a client and at least one server is triggered in case users change content or interact with the website resulting in a change of the DOM structure. The resulting DOM combined with the template needs to be re-rendered. The frameworks we focus on in this survey reflect the changes immediately and reduce unnecessary traffic[Kozlowski and Darwin, 2013].

4.1 AngularJS

The JS framework AngularJS was created by Misko Hevery worked his way through Adobe, Sun Microsystems, Intel, Xerox, and currently works at Google as a coach for automated testing. His involvement in open source projects lead to him developing frameworks like AngularJS and JSTestDriver[*About Miško Hevery* 2016]. Misko Hevery started to work on AngularJS in 2009 and it then was first introduced in 2012 where it was already backed by Google allowing Misko to develop AngularJS full time with a team. AngularJS is thus licensed by Google with an open source MIT license[Kozlowski and Darwin, 2013].

The success of AngularJS was based on a broad community generating mailing lists, IRC channels, a specific tag at stackoverflow, and of course a Google+ community. The developing team created an AngularJS blog and is present in social media channels like Google+ and Twitter and community meetups give users the chance to exchange ideas. A collection of applications built with AngularJS can be found in at [**.**] [Kozlowski and Darwin, 2013] provides examples as well as recordings from previous events.

To start to develop a web application with AngularJS developers need to set up an environment to develop web sites. It is recommended to either use a XAMPP server locally or rely on a development environment using Node.js. The AngularJS library can be imported by the common <script> tag from `https://ajax.googleapis.com/ajax/libs/angularjs/1.6.0-rc.2/angular.js` or installed by using 'npm install angular@1.6.0-rc.2'. In contrast to other scripts, the AngularJS script needs to be inserted in the header or the rest of the HTML will not be rendered as an AngularJS app. Otherwise it would simply be displayed as HTML not rendering the input.

Not everything from this fully fledged framework is available out of the box. However, developers support each other by providing modules that tackle complex issues like mobile device support or simple tasks like the drag and drop feature we use in our survey. But these extra modules often use other JS libraries and have to be downloaded and made available by using npm - 'npm i angular-file-upload' - to make a file upload available in one certain development scope and the libraries the modules are written in have of course to be imported by the <script> tag. Often, jQuery is used but sometimes the libraries get more exotic and the usage of many modules in complex application can lead to a <head> tag full of script imports of script that are barely used throughout the web application.

4.1.1 Directives

An example of a very simple interaction of a user resulting in a change of displayed data could be inserting an email in some form. Here, the inner HTML of a UI element is changed. The function would have to be called in case a user logged in. The email email address of the user has to be displayed somewhere to help the user understand that the user is logged in or not. Another use case would be if the user wants to change the email address.

But this would not be the only tags content that needed input to display content personalised to a user. Perhaps the user is not only shown the email address but also the user's first and last name. Every change in any of those tags contents would normally result in sending data to a server and back as well as a recalculation and rerendering of displayed content.

AngularJS does not send content but binds the data directly to the HTML tag and updates it whenever necessary. In AngularJS, all code needed would be entirely within the HTML.

The content of the HTML tag would be updated whenever necessary without any JS code necessary since the HTML tags content itself would take care of displaying whatever email address "email" would have. Moreover, in case someone would turn off JS and had more complex HTML half written in the HTML document and the other half generated by JS on the fly, only good documentation would give a hint on what really

```
Email address: <span id="email"></span>

<script>
  var updateEmail = function(email) {
    $('#email').text(email);
  };
</script>
```

Listing 4.1: Email Update

```
Email address: <span>{{email}}</span>
```

Listing 4.2: Email Update in AngularJS

should have been displayed. In AngularJS the HTML has the focus and thus information about what should be displayed.

Furthermore, this example shows the reason for the frameworks name: AngularJS has its name from the angular brackets around variables. The information which value something should have and where and how HTML tags should act like AngularJS tags is declared in so called directives. An AngularJS directive can be recognized by the prefix ng- and is put directly into a HTML tag. One special directive marks parts of HTML as AngularJS applications: ng-app put into any tag within the body marks the part of the HTML or in the HTML tag itself marks the whole HTML document.

Important directives are not only the ng-app directive but also:

1. ng-init initializes application data.
2. ng-bind="name" binds the value of "‘name"’ to a tag - similar to name.
3. ng-module allows a two way information exchange between HTML tags and code within modules.
4. ng-controller creates a controller for data of a module.

All directives that come with the library can be found at `https://docs.angularjs.org/api/ng/directive` but since directives can also be defined manually many more can be found at the internet.

4.1.2 Modularity

As mentioned in the previous subsection the ng-module directive allows to exchange values of data between a module and a HTML tag. Whenever a user interacts with a tag and thus has the current scope on that tag a controller loads or reads data as stated in the regarding module. A controller is declared within a view and its behavior is written in the JS part of the HTML document. That way, a behavior can simply be reproduced by anything using the same controller. In case something went wrong with the JS the HTML would still put "’Welcome helloTo.title!"’ into the text thus at least explaining what value a user should expect.

```
<body ng-app = "myapp">
  <div ng-controller = "welcomeController" >
     <h2>Welcome {{helloTo.title}}!</h2>
  </div>
</body>

<script>
   angular.module("myapp", [])

   .controller("welcomeController", function($scope) {
      $scope.helloTo = {};
      $scope.helloTo.title = "AngularJS";
   });
</script>
```

Listing 4.3: Controller putting the "model" value named helloTo.title to the HTML

```
<div id="profile"  ng-init="content2=[
    {profile_header:'TU Graz',profile_text:'seit 201x'},
    {profile_header:'HTL Grammatneusiedl',profile_text:'Maschinenbau'}]">
    <span class="header">Education</span>
    <div ng-repeat="x in content2">
      <span class="profile_header"> {{ x.profile_header}} </span>
      <br>
      <span class="profile_text"> {{x.profile_text }} </span>
    </div>
</div>
```

Listing 4.4: Much logic within HTML directly causing a vast HTML document.

4.1.3 Template example

Rewriting the template using HTML that is boosted in its capabilities with AngularJS was easy since the support to write basic examples and how to handle directives is vast. Moreover, AngularJS did not require to learn a new scripting language. While changing parts of the CSS and values of HTML tags was fairly easy and could be done within the HTML using built in directives and expressions, more complexity required either a lot of logic within the HTML itself or coding JS.

In contrast to other frameworks evaluated in this survey the logic of the language is within the HTML tags and does not output HTML tags as a result of building modules. This causes a much longer HTML document than at the other libraries we looked at. As an example, the repetition of a content structure initialized within a <div> containing profile information about a person using the directive ng-repeat would take a variables value derived from the initialized content and repeat each value throughout the content. In other libraries this could be a custom tag having exactly the function to repeat variables only displaying the <custom> tag in the HTML. Using modules and controllers moves the logic away from the HTML into JS. Another way to repeat content could be the bind directive ng-bind and using a controller sending and receiving data to and from JS.

```
<div id="profile" ng-app="myApp" ng-controller="myCtrl">
  <span class="header">Profile</span>
  <div ng-repeat="x in content">
    <span class="profile_header" ng-bind="x.profile_header" ng-click="removeItem2($index)
        "> </span>
    <br>
    <span class="profile_text" ng-bind="x.profile_text"> </span>
  </div>
</div>

<script>
var app = angular.module('myApp', []);
  app.controller("myCtrl", function($scope) {
    $scope.content = [{
      profile_header : 'TU Graz',
      profile_text : 'seit 201xr'
    }, {
      profile_header : 'HTL Grammatneusiedl',
      profile_text : 'Maschinenbau'
    }];

    $scope.addItem2 = function() {
      $scope.content.push({
        profile_header : $scope.addMe,
        profile_text : $scope.addMe2
      });
    };
      $scope.removeItem2 = function(x) {
        $scope.content.splice(x, 1);
      };
  });
</script>
```

Listing 4.5: Using Controllers to move the logic away from HTML.

Even more complex than the issue of simply repeating entries from a source was the matter of recreating the part of the file upload for the picture of a user. Since there is no built in drag and drop directive, there were two ways to implement the function:

1. Create a module using JS allowing to pick up, drag, drop, and upload pictures,
2. or hope to not be alone with the problem and find an already existing solution from another programer.

The first thing that was tried for this survey was to find an already existing solution which there are a few of. The found solutions were model based and written in not only different languages but also different complexity. Amongst the three different solutions tried for this survey, some were only able to upload per drag and drop. The chosen solution allowed the upload and immediate display of pictures per drag and drop and by clicking a button opening a dialog to choose a file from a computer.

For the chosen upload model, it was necessary to define an area where to put the upload button and to drag and drop the image to as well as tell the image section that it should display an image by binding the data to the <image> tag. The HTML part was short but the JS part of the model was vast. No two controllers can easily control the data within the same model if the HTML tags are not distinct since the controller and model logic is bound to HTML logic. The drop area of the pictures source was in an entirely different section of the HTML than the image was. That meant putting all JS within one huge model taking care of mainly the whole application. A situation like this renders the modularity of AngularJS to a minimum. To avoid that, it would have been necessary to rewrite all HTML sections to have a clean structure.

Since the used drag and drop module was written in JS, namely in jQuery, the specific jQuery version had to be imported. If a web application would use many different models, many more libraries would have to be imported. And since AngularJS claims to be in some way independent of the scripting language used to create models, not only libraries but also running for example Ruby or PHP could have been possible bringing with them all the positive and negative sides the language has into the web application a programer wants to create

In our opinion, the openness of AngularJS can be both a nice feature and dangerously error prone in the hands of beginners. There is no clean line between the logic within directives and modules. Directives keep the code part short and put the logic into the HTML while modules put the logic inside of a controller serving as an interface to JS. This leaves the door open for custom results that could be not modular at all.

4.2 Angular 2

Angular 2 [*Angular 2 Framework* 2016] is an open source, cross-platform framework for building web applications. This framework was released in 2016. Angular 2 is the successor of AngularJS, but it has major differences to its predecessor. It uses TypeScript. TypeScript is "a strict superset of JS that adds optional static typing and support for interfaces and decorators" [Deeleman, 2016, page 18] . It compiles to plain JS and offers type checkingwhich makes it less prone for producing run time errors.

Since Angular 2 was released earlier this year, the online community support is less established yet. Searching for documentation and help online can be confusing with two different Angular versions out there. Angular 2 takes time to learn. Since the older Angular version is an entirely different thing, using AngularJS code in Angular 2 is not possible and migration is a complex procedure. Angular 2 needs Node.js (a server-side JS enviroment) and npm (a package manager for JS). Angular 2 focuses on mobile apps and uses server side rendering for fast views on mobile which makes it a great framework for progressive web apps. Additionally, the code structure and modularity of Angular 2 compared to AngularJS is noteworthy. Angular 2 uses virtual DOM which allows users to manipulate a DOM tree directly in JS. AngularJS and 2 both use dependency injection.

Angular 2 is entirely component-based, it is not possible to create web applications without creating at least one web component. The whole application needs to be nested into the root component. The idea is to extend HTML itself with custom HTML tags. A directive in Angular 2 adds behaviour to an existing element in the DOM. Angular 2 components are directives with an associated template which contains HTML. Web components in Angular 2 combine model, view and controller, therefore it is much more object-oriented than AngularJS.

```html
<div id="content">
    <div id="header">
        <portrait></portrait>
        <introduction></introduction>
    </div>
    <cv-form></cv-form>
    <div id="footer">
        <quote></quote>
    </div>

</div>
```

Listing 4.6: This file (`app.component.html`) includes the HTML template of the root component called AppComponent of the Angular 2 example. There are various web components nested inside the root component.

To demonstrate how a component is implemented in Angular 2, a demo application was created. The root component consists of different other components, one of them is a form where users can enter personal information. It follows the model-view-controller design pattern, which is already set by the framework's structure. The root component called AppComponent consists of multiple nested components, each with their own costum HTML tag (see Listing 4.6).

In Listing 4.7 the class CV is implemented in TypeScript. It has different member variables to represent the model for the form. Listing 4.7 shows the actual component of the form. The `selector` denotes the name of the new HTML tag that is created with this component. The developer can then add the following HTML tag into any other web component template:

```html
<cv-form></cv-form>
```

The CV class is imported. The view (an external HTML file) is added via `templateUrl`. The content of the HTML file is shown in Listing 4.9.

4.3 React

React is a JS Framework that is built from Facebook. It us useful for rich and fast Webcomponents. It makes the compononts way of seeing the basis for all. The Apps are written in modularized JSX code. It is like program in HTML code directly.

4.3.1 History

JSX the actual programing language in React is based on XHP, which has been produced and distributed by Facebook before. XHP is build up on PHP and should reduce the possibility for Cross site scripting. XHP has fixed that problem but it doesn't offer a better performance when there are often connections back to the server, what is typically for dynamic web content. A developer has got the mission to play a little with XHP to increase it. The output of it has become React. [*JavaScript's History and How it Led To ReactJS* 2016]

```
export class CV {
  constructor(
    public name: string,
    public born: string,
    public hometown: string,
    public email: string,
    public phone: string
  ) {  }
}
```

Listing 4.7: This file (`cv.ts`) includes the model for the Angular 2 example component. It is a class named CV with different variables. It represents the model for the CV form component, written in TypeScript.

```
import { Component } from '@angular/core';
import { CV } from './cv';
@Component({
  moduleId: module.id,
  selector: 'cv-form',
  templateUrl: 'cv-form.component.html'
})
export class CVFormComponent {

  model = new CV('', '', '', '', '');
  submitted = false;
  onSubmit() { this.submitted = true; }

  get diagnostic() { return JSON.stringify(this.model); }

}
```

Listing 4.8: This code snippet (file `cv-form.component.ts`) implements the controller for the CV form. The associated HTML is included with templateUrl. The model class CV is imported.

```html
<form #CVForm="ngForm">
  <div id="profile">
    <span class="header">Profile</span>
    {{diagnostic}}
    <div class="form-group">
      <span class="profile_header"><label >Full Name</label></span>
      <span class="profile_text"><input type="text" class="form-control" [(ngModel)]="
          model.name" name="name" ></span>
    </div>

    <div class="form-group">
      <span class="profile_header"><label >Born</label></span>
      <span class="profile_text"><input type="text" class="form-control" [(ngModel)]="
          model.born" name="born"></span>
    </div>

    <div class="form-group">
      <span class="profile_header"><label >Hometown</label></span>
      <span class="profile_text"><input type="text" class="form-control" [(ngModel)]="
          model.hometown" name="hometown" ></span>
    </div>

    <div class="form-group">
      <span class="profile_header"><label >eMail</label></span>
      <span class="profile_text"><input type="text" class="form-control" [(ngModel)]="
          model.email" name="email"></span>
    </div>

    <div class="form-group">
      <span class="profile_header"><label >Mobile Number</label></span>
      <span class="profile_text"><input type="text" class="form-control" [(ngModel)]="
          model.phone" name="phone"></span>
    </div>

  </div>
  <button type="submit" class="btn btn-default" [disabled]="!CVForm.form.valid">Submit</
      button>
</form>
```

Listing 4.9: This HTML template file (`cv-form.component.html`) implements the view part of the CV form component.

4.3.2 Community

There are some homepages where you get support for React. But before it comes to community support there are also tutorials, documenatation ready to start configurations by facebook. That all can be found at `https://facebook.github.io/react/`. They help to understand how JSX works and how get into it.

Apart on that there are own discussion topics on Stackoverflow `http://stackoverflow.com/questions/tagged/reactjs`, a discussion forum `https://discuss.reactjs.org/` and a Live Chat `https://discord.gg/0ZcbPKXt5bZjGY5n` when someone really needs imediate support. The project is hosted on Github and it has got the project has been rated with 55103 stars by now. [*React at Github* 2016]

4.3.3 Prerequisites and dependencies

There are a few ways to set up a development and building environment. You need different tools for different steps in the compiling process:

- **Node.js**: The package manager of Node.js is used for installing the things above and keep them up to date and handle the dependencies between them.
- **Babel**:Compiles JS into JS that is executable in common client browsers. It has optimization libs for JSX and supports also the new ES6 JS Standard.
- **Browserfy**: Combines JS code from components together in one file. It detects automatically the dependencies.
- **React**: React itself consists of several librarys but for minimal usage the following are needed:
  - react: Element forr interacting with the React API and use the function it supports.
  - react-dom: It's main function is to render react elements. So Creating

  One important thing is that write down functionality and rendering is done somewhere else.

[Stefanov, 2016][4f,96f]

The most comfortable way of getting React running is to use the npm for everything you need. It downloads the software install it and makes it accessable in the code files. But you should well decide which library of react you need. Adding Libraries afterwards to npm may cause into errors. For simple use see Listing 4.10

React is actually built for running in the client part of a webapp but through Node.js it is also possible to run some code on server.

```
import React from 'react';
import ReactDOM from 'react-dom';
```

Listing 4.10: Import Listings

4.3.4 JSX

As discussed before JSX is the programming language react supports. The easiest way for understanding how it works is through looking how react builds the example of the cv for comparing the frameworks. The final Page looks like in Figure 4.1. The page consists of a header with portrait picture and a short text for opening. Under that the CV entries are starting. The elements with the actual data are combined in groups. So the CV itself is modular and corresponding to this the code is also modular. The JSX code that builds the CV is displayed in Listing 4.11.

The render function of the ReactDOM object transforms the JSX code in JS and updates the DOM of the Browser. So when the elements gets created the DOM is changed once. In the render function the codes for building the component is written down. JSX looks like writing interactive HTML. It supports the ability to write templates that maps to other react components. `<Cv name='Max Mustermann'>...</Cv>` creates the component for Max Mustermann. In another file this Cv component is defined. The values that are set here with parameters like name or the content that is inside the tags are transmitted to the Cv component via parameters:

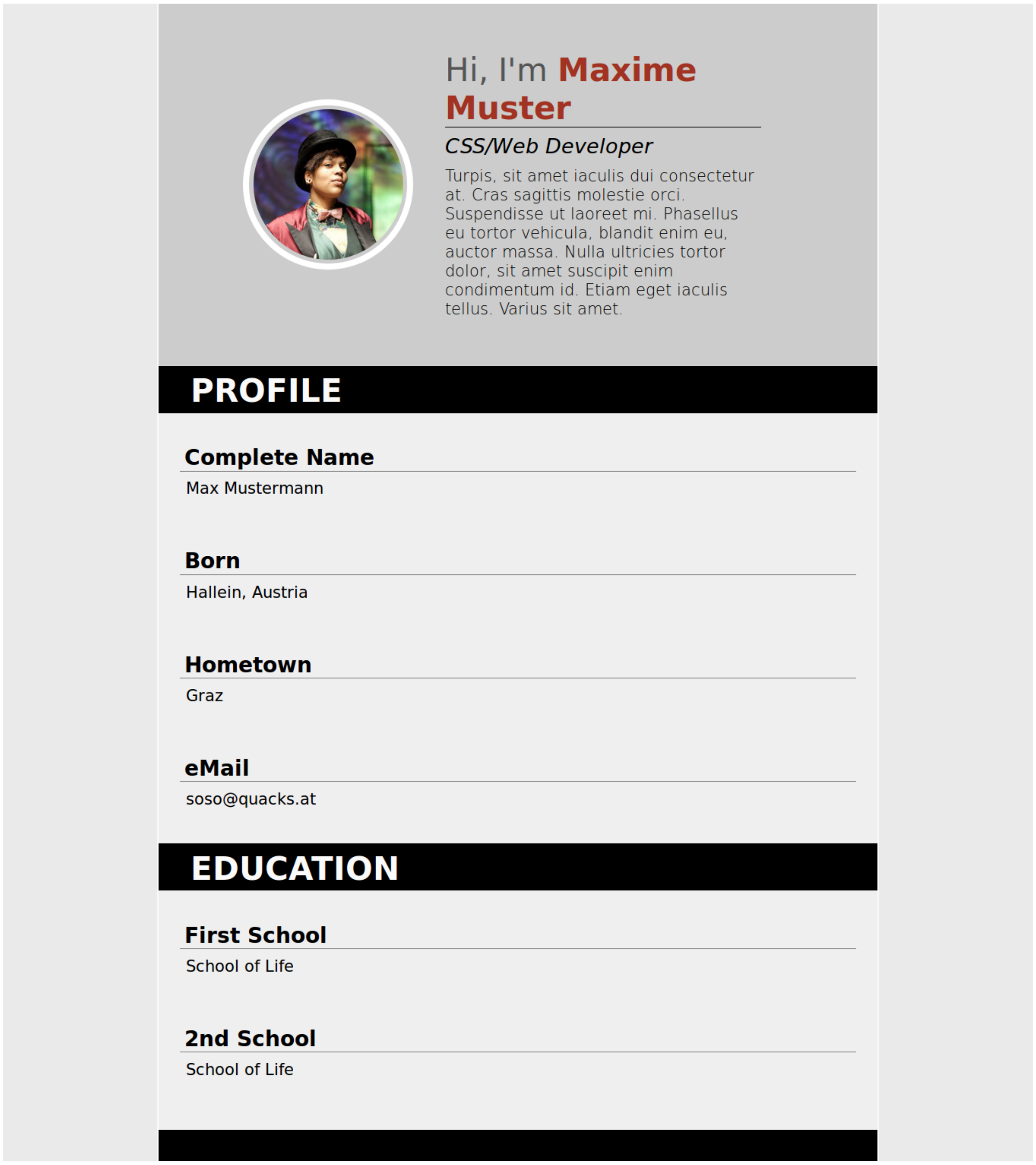
- name: `this.props.name`
- Stuff inside Cv (`<Profile ... </Profile>`): `this.props.children`

```
ReactDOM.render(
  <Cv name='Max Mustermann'>
    <Profile name='Profile'>
      <Element name='Complete Name'>Max Mustermann</Element>
      <Element name='Born'>Hallein, Austria</Element>
      <Element name='Hometown'>Graz</Element>
      <Element name='eMail'>soso@quacks.at</Element>
    </Profile>
    <Profile name='Education'>
      <Element name='First School'>School of Life</Element>
      <Element name='2nd School'>School of Life</Element>
    </Profile>
  </Cv>,
document.getElementById('root')
);
```

Listing 4.11: Building the CV in JSX

4.3.5 React Components

As described before react applications are built upon components. A component is called and parameterized via the call as html tag. Before the component has to be accessable for other files. So for example in `Cv.jsx` the app has to be exported at the end of the file, displayed at Listing 4.12. Then the internal React component Cv can be used.



Hi, I'm **Maxime Muster**

CSS/Web Developer

Turpis, sit amet iaculis dui consectetur at. Cras sagittis molestie orci. Suspendisse ut laoreet mi. Phasellus eu tortor vehicula, blandit enim eu, auctor massa. Nulla ultricies tortor dolor, sit amet suscipit enim condimentum id. Etiam eget iaculis tellus. Varius sit amet.

PROFILE

Complete Name
Max Mustermann

Born
Hallein, Austria

Hometown
Graz

eMail
soso@quacks.at

EDUCATION

First School
School of Life

2nd School
School of Life

Figure 4.1: Screenshot of CV built with React

```
import React, { Component } from 'react';
import './Cv.css';
import pic from '../img/portrait2.jpg';


class Cv extends React.Component{

  render() {
    return (
    <div id='content'>
    <div id='header'>
    <div id='portrait'>
    <img id='portrait_image' src={pic} alt='My Name'/>
    </div>
    <div id='introduction'>
    <div id='introduction_hello'>
    <span className='greeting'>Hi, Im </span>Maxime Muster
    </div>
    <div id='introduction_job'>
    CSS/Web Developer
    </div>
    <div id='introduction_abstract'>
    Turpis, sit amet iaculis dui consectetur at. Cras sagittis
        molestie orci. Suspendisse ut laoreet mi. Phasellus eu tortor
         vehicula, blandit enim eu, auctor massa. Nulla ultricies
        tortor dolor, sit amet suscipit enim condimentum id. Etiam
        eget iaculis tellus. Varius sit amet.
    </div>
    </div>
    </div>
    {this.props.children}
    <div id='footer'>
    Random Quote
    </div>
    </div>
    );
  }
}

export default Cv;
```

Listing 4.12: Cv Component in React

Int Listing 4.12 the component is designed in HTML. The Profiles them self are defined in another file called `Profile.js`. They get included by simply calling this.props.children.

```
  class Profile extends React.Component{

    render(){
      return (
      <div className='profile'>
      <span className='header'>{this.props.name}</span>
      {this.props.children}
      </div>
      );}
  }

  export default Profile;
```

Listing 4.13: Profile Component in React

```
class Element extends React.Component{

  render(){
    return (
    <div className='Element'>
    <span className='profile_header'>{this.props.name}</span>
    <span className='profile_text'>{this.props.children}</span>
    </div>
    );}
}

export default Element;
```

Listing 4.14: Element Component in React

4.3.6 Install example

This project is built with npm. The base was an easy to start package from facebook that can be found at `https://github.com/facebookincubator/create-react-app`. This is not necessary for starting the CV page. Everything that is necessary for running is distributed with this paper.

There are two different ways for running:

- Run the development version with npm
- Run the build solution

Run the development version

Just change to the directory and start the app. The app will then be running at localhost:3000

```
$ cd <Source Dir>
$ npm start
```

Apache 2 has to be installed for that.

Run build solution

The files are already compiled to pure java script. It is necessary to open the file. Here for example with Firefox:

```
$ cd <Source Dir>/build
$ firefox index.html
```

This build solution can be executed with:

```
$npm run build
```

4.4 Vue.js

Vue.js[*vue.js* 2016] is a JS Framework developed by Evan You with its first release in early 2014[*First Week of Launching Vue.js* 2016]. The current version 2.1 was introduced in November 2016.

The focus of Vue is on the core library, maintaining components, reacitivity and DOM manipulation, while leaving problems like statemanagement, routing or server side rendering to other libaries/frameworks, which can be easily integrated.

4.4.1 Structure

To get up and running, nothing more than to include the vue.js file in your html page and start designing your components is needed. You have an html tag where your vue-instance is hooked on. So, your page is not built by Vue, it merely is included in your website. Of course you can see your page as one single component, but you can also have multiple instances of Vue and make multiple, or just some parts of the project reactive.

Creating a new instance is as simple as creating it in JavaScript and bind it to an DOM element:

```
new Vue({
  el: '#element'
}
```

Listing 4.15: Creating a new instance

4.4.2 Components

A Component is a self-containing entity with its own controller. It consists mainly of a template, which defines the look of the component, and the model for the data. There are also, for example, directives, mixins and computed properties to help shaping the right behaviour for the components. It is registered to the Vue instance in the following way:

```

  Vue.component('component_name',
    {
      //template,
      //data,
    //..
    })
```

Listing 4.16: Component registration

Template

The template is build upon html tags and other components already built in Vue. To help maintaining a declarative style in comparision to a imperative style, directives are used. They function like a typical attribute in an html tag, declaring behaviour in a way as "you are this and behave like that" instead of imperative forming the behaviour with JavaScript functions.

Every Template has to have one root node. List 4.17 shows a simple template consisting of one div-element as root node, having three vue components as child nodes:

```
  Vue.component('introduction_c',
    {
        template: '<div id="introduction">' +
        '           <introduction_hello_c></introduction_hello_c>' +
        '           <introduction_job_c></introduction_job_c>' +
        '           <introduction_abstract_c></introduction_abstract_c>
            ' +
        '         </div>',

    })
```

Listing 4.17: Simple template

Listing 4.18 shows a more complex template including directives like v-if and the common brackets for displaying data:

```

Vue.component('introduction_hello_c',
{
template:
  '<div id="introduction_hello"><greeting_c></greeting_c>' +
  '  <span v-if="is_text==true">{{name}}</span>' +
  '  <span v-if="is_text==false">' +
  '    <input  name="input_form" type="text" ' +
  '           v-on:keyup.enter="switch_form" ' +
  '           v-model="name" ' +
  '           :value="name"' +
  '           v-focus>' +
  '  </span>' +
  '</div>',
}
```

Listing 4.18: More complex template

Data

Vue is reactive, which means, if the model has changed, the view is getting updated aswell. The advantage for the developer is that he has not to interfere with the DOM at all. A component can either have its own data, or have data passed down from the parent via so called props. Vue follows a top-down approach, which means children can get data from their parents, but not change data from the parents. To have data from parents changed, you need to use events. In short, props down, events up.

```
data: function () { return { name: 'Max Muster', is_text: true }},
```

Listing 4.19: Data defined in a component

```
<container_c v-for="container in containers" v-bind:headername="
    container">
```

Listing 4.20: Bind data in a parent to be used by children as a prob with the v-bind directive

```
<span>{{name}}</span>
```

Listing 4.21: Using element name from the data in a template

Directives

Directives can be seen as adding attributes to various elements to maintain a declarative style. There are several built-in directives in Vue, but one can also make new directives for each component, or even for each vue instance and include them in the components via so called mixins. An example of a directive would be v-for, which is a declarative style of a for-loop, for example to display a list:

```
<container_c v-for="container in containers" v-bind:headername="
    container">
</container_c>
```

Listing 4.22: v-for directive

For every entry in containers, the v-for element makes a new container component. It is reactive aswell, meaning that everytime the model gets changed, the list is rerendered with the new entries in the model. So if you add a new element in the model, the element instantly appears in the DOM.

4.4.3 Going large-scale

Vue.js was create to be lightweight and focus on beeing reactive and DOM manipulation. But there are many tools and libraries for large-scale projects to help with things like state-management and server side rendering. For state-management for example, Vue has it's own, flux-like vuex library.

SSR

If you look at the source of a vue-made website, you mostly see some tags with names of some components. So if a users turns of javascript - nothing gets displayed. One way to avoid this, is to implement server side rendering.

Single File Components

As a projects grows, it is getting unreadable to have all the components in one single file. Vue.js supports Single File Components, where each component is separated into single files. Another advantage of this is that everything in one component file is in its own scope. That means, you can also put component-specific stylesheets and scripts in the file, and they cannot interfere with styles or scripts of other components. Also, readability especially for templates is much higher, because you can define templates like normal html and not within an javascript string, which enables better syntax highlighting. The major drawback is, as Vue.js is lightweight by itself, one needs packages like browserify and precompile all the components before deploying.

Routing

Vue.js has some simple routing options built in, but also has its own vue-router library for full-fledged single page applications.

vue-cli

There is an official command line interface for Vue called vue-cli, which offers for example hot-reloading of components.

Debugging

For Chrome there exists an official addon to have a look at the model, and also time travel between different states if vuex is used.

Awesome Vue

AwesomeVue [*vuejs/awesome-vue - A curated list of awesome things related to Vue.js* 2016] is a comprehensive resource for everything related to Vue.js, with links to libaries, examples and practice examples.

	AngularJS	**Angular2**	**React**	**Vue**
WebComponents	no, just modular	yes	no, just modular	loosely modeled after the spec
Age	deprecated, 4 years	released 2016	3 years	3 years
Community	well developed	not yet established	well developed	good
Dependencies	Node.js, npm	Node.js, npm	npm	none
Compatibility	Angular2 not compatible	cannot use code of predecessor	no previous version	largely compatibale
Language	JS	TypeScript	JS, JSX	JS
Recommended	no	yes, especially for bigger projects that have to be responsive	yes	yes, especially for small projects

Table 4.1: Framework Distinctions

4.5 Comparative Studie

In Table 4.1 all four tested frameworks are presented due to some capabilities. The oldest one is AngularJS. Vue and React have quite the same age and Angular 2 is almost the newest one.

Most common used now is React but it is not decided how successful Angular 2 will be. The potential is there that they are getting big. This is a try for comparing them.

Based on what they are and support they are technically different to each other, especially their extent is different. React basically starts with less functionality than Angular 2 does. It gives more freedom to the user to write their apps. Pure JS, JSX in ES5 and in ES6 are ok and valid. Some extensions work only with certain languages [*React vs. Angular 2 - Comparison* 2016].

Where does this come from? Typescript is the language Angular 2 is using. While JSX in React is a way to write HTML like Templates, in Angular 2 string based HTML templates are supported. Typescript uses models to implement changeable data binding and React uses the state property that gets compared to the displayed value and updates it if there is a difference [*React vs. Angular 2 - Comparison* 2016].

> **"When comparing the two it is like deciding whether to buy a computer off the shelf or building a computer with off the shelf parts."**
>
> *Debugme Blog, React vs. Angular2 Comparison: A Heavyweight Bout for the JS Title,*
> `http://blog.debugme.eu/react-vs-angular2/`

The quote above describes it the best when it comes to the major difference and aim react and Angular 2 have. Angular 2 delivered with all functionality in it when starting and in React you first have to think which extensions you need and also which special compilers you need.

In comparison to Vue both, Angular 2 and React are more easy when it comes to write apps in HTML. In Vue HTML structures has to be constructed with single strings per line. This is annoying and makes it more unstructured and hard to read. But Vues Benefit is fast and lightweight setup and also supports automatic changing of the DOM.

AngularJS (or Angular 1) is not comparable to Angular 2 at all. It is quite old fashioned. But it is also the oldest one of these Frameworks. It us optimized for rendering also on mobile devices. To start a new project with AngularJS is not recommended.

The community is quite huge for React, Angular and Vue. Angular2 hasn't really a big one. For the other three frameworks many issues has already been discussed in Stackoverflow and Github. So some problem that occurs has already been discussed somewhere.

Chapter 5

Concluding Remarks

In this survey four different JS UI frameworks were analysed and compared. The chosen frameworks were AngularJS, Angular 2, React and Vue. Due to the demand for more and more complexity in web applications, the model-view-controller design pattern has shifted to web development as well. The browser is just displaying the front end of an extensive application. Thus, the need for frameworks that aid fast development and focus on modularity and reusability has risen.

To realise the requirements of reusable interface elements, web components were introduced. Web components are custom HTML tags that combine behaviour and view. Web components were created to encapsulate parts of an user interface which can be reused. It is possible to write a custom web component or import web components from a framework or library. When implementing a custom web component, the developer is confronted with the question "What parts of the interface can be assembled together to one web component?". To answer this question, atomic design gives a guideline about what could be a web component. Atomic design is a desing principle that divides parts of the interface into elements, based on chemistry. The smallest element is an atom, a single HTML tag. Atoms combined together are called a molecule. Multiple molecules are organisms. An organism would be a complex web component.

An overview about the featues of each JS UI framework was given. In addition, an example of how a web component could be realised in the four different frameworks is presented. AngularJS, being the oldest one of the compared frameworks, does not support web componens while its successor Angular 2 is entirely component-based. Nevertheless, it is possible to combine behaviour and view as modules in AngularJS. Modularity is also possible in React. Vue supports component-based development.

In the final analysis of the four different frameworks it is concluded that Vue is especially recommendable for small projects and prototyping since the app size is small and it is easy to use. Angular 2 and React are good alternatives as well, but here the developer has to take some time to learn how to use this framework. AngularJS will probably become more and more outdated since there is a completely new version of it with more advanced features.

Web components are definitely a future trend in web development. Creating complex web applications is gaining more siginicance. Users prefer using the online version of a program instead of downloading them on their computer. Moreover, the majority of the web traffic is now produced by tablets and mobile phones. Users still want to access applications, therefore online versions of those applications have an increasing demand. Thus, JS UI frameworks like the ones analysed in this survey, will have a bright future.

Bibliography

About Miško Hevery [2016]. 27th Nov 2016. `http://misko.hevery.com/about/` (cited on page 8).

Angular 2 Framework [2016]. 27th Nov 2016. `https://angular.io/` (cited on page 12).

Branas, Rodrigo [2014]. *AngularJS Essentials*. Packt Publishing, 2014. ISBN 9781783980086 (cited on page 7).

Deeleman, Pablo [2016]. *Learning Angular 2*. Packt Publishing, 2016. ISBN 9781785882074 (cited on page 12).

First Week of Launching Vue.js [2016]. 5th Dec 2016. `http://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project/` (cited on page 21).

Frost, Brad [2016]. *Atomic Design*. 27th Nov 2016. `http://bradfrost.com/blog/post/atomic-web-design/` (cited on page 5).

JavaScript's History and How it Led To ReactJS [2016]. 3rd Dec 2016. `http://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/` (cited on page 13).

Kozlowski, Pawel and Peter Bacon Darwin [2013]. *Mastering Web Application Development with AngularJS*. Packt Publishing, 2013. ISBN 9781782161820 (cited on page 8).

React at Github [2016]. 3rd Dec 2016. `https://github.com/facebook/react` (cited on page 16).

React vs. Angular 2 - Comparison [2016]. 5th Dec 2016. `http://blog.debugme.eu/react-vs-angular2/` (cited on page 25).

Seshadri, Shyam and Brad Green [2014]. *AngularJS: Up and Running: Enhanced Productivity with Structured Web Apps*. O'Reilly Media, 2014. ISBN 9781491901946 (cited on page 7).

Stefanov, Stoyan [2016]. *React - Up & Running*. 2016. ISBN 1491931825 (cited on page 16).

vue.js [2016]. 5th Dec 2016. `http://vuejs.org` (cited on page 21).

vuejs/awesome-vue - A curated list of awesome things related to Vue.js [2016]. 6th Dec 2016. `https://github.com/vuejs/awesome-vue` (cited on page 24).

Web Components Current Status - W3C [2016]. 6th Dec 2016. `https://www.w3.org/standards/techs/components` (cited on page 3).

What the Heck is Shadow DOM? [2016]. 6th Dec 2016. `https://glazkov.com/2011/01/14/what-the-heck-is-shadow-dom/` (cited on page 3).