Lappeenranta University of Technology
School of Engineering Science
Erasmus Mundus Master in Pervasive Computing and Communications for Sustainable Development (PERCCOM)

Mukhammadjon Jalolov

Analyzing JavaScript frameworks and Dart for Front-end Development in Building Automation

2018

Examiners:  Professor Eric Rondeau (University of Lorraine)
Professor Jari Porras (Lappeenranta University of Technology)
Professor Karl Andersson (Luleå University of Technology)

Supervisor:  Professor Olaf Droegehorn (Harz University of Applied Sciences)
This thesis is prepared as part of an European Erasmus Mundus programme PERCCOM – Pervasive Computing & COMmunications for sustainable development.

This thesis has been accepted by partner institutions of the consortium (cf. UDL-DAJ, n°1524, 2012 PERCCOM agreement).

Successful defence of this thesis is obligatory for graduation with the following national diplomas:

- Master in Complex Systems Engineering (University of Lorraine)
- Master of Science in Technology (Lappeenranta University of Technology)
- Master of Science in Computer Science and Engineering, Specialisation: Pervasive Computing and Communications for Sustainable Development (Luleå University of Technology)
ABSTRACT

Lappeenranta University of Technology
School of Engineering Science
Erasmus Mundus Master Program PERCCOM

Mukhammadjon Jalolov

Analyzing JavaScript frameworks and Dart for Front-end Development in Building Automation

Master’s Thesis
72 pages, 38 figures, 8 tables, 6 formulas, 1 appendix
Examiners: Professor Eric Rondeau (University of Lorraine)
          Professor Jari Porras (Lappeenranta University of Technology)
          Professor Karl Andersson (Luleå University of Technology)

Keywords: JavaScript Frameworks, Dart, Home Automation, Frontend, Maintainability

Building or Home automation systems (BAS), (HAS) can inarguably reduce energy consumption. However, the widespread adoption of BAS is still inferior due to sophisticated user interfaces (UI). In fact, JavaScript (JS) is the primary tool for web applications with dozens of frameworks (JSf). Such an abundant number of JSf require empirical comparison to select the optimal JSf for building highly acceptable BAS UIs with less complexity and efforts. Therefore, based on open source HAS – FHEM, this study analyzes JSfs and Dart language (Dart) in the field of BAS using code analysis tools to extract metrics on software maintainability in different UI scenarios in order to guide the BAS UI developers towards the convenient JSfs which require less efforts to build a UI. The study also studies the MVC design pattern support. The qualitative data analysis is done to identify the distinctions of JSfs and quality attributes. The research has selected AngularJS, Angular2, Ember and React JSfs. Online survey is also done for better results. The results show that each framework comes with their different advantages.
ACKNOWLEDGEMENTS

This thesis work is funded by and is a part of the European Erasmus Mundus programme (Pervasive Computing and COMmunications for sustainable development) [52]. The authors are grateful to the Erasmus+ programme of the European Union, academia, companies, and associate partners of the PERCCOM Consortium.

First of all, I would like to thank sincerely to the PERCCOM committee for giving me a great opportunity for participating in the program which literally turned me into a person with aspirations towards sustainability. The last two years were more than just a study abroad for me. PERCCOM program contributed my immense development in the field of ICT.

My deep gratitude to my supervisor Prof. Dr. Olaf Droegehorn for guiding me through the hardship of the research work with his advises and helps. The contribution of Professor Josef Hallberg to my thesis work accomplishment is priceless.

I would also express my special thanks to professors – Eric Rondeau, Jean-Phillipe Georges, Jari Porras, Ah Lian Kor, Karl Andersson, Collin Pattinson, Alexandra Klimova and Penzenstadler Birgit Peters. Moreover, my warm thanks to PhD student – Maria Victoria Palasin Silva for her guidance during my second semester.

The completion of this research work would not be possible without the support of my dearest people in life – parents and siblings. They have always been given me strength encouraging me to be patient during difficulties, motivating to be stronger, inspiring for higher achievements. Thank you for everything my family! My greetings to all of my PERCCOM classmates, together with them PERCCOM was fun and happy journey.
# TABLE OF CONTENTS

1 INTRODUCTION ................................................................................................................. 4
   1.1 BACKGROUND ............................................................................................................. 4
   1.2 MOTIVATION ............................................................................................................... 5
   1.3 RESEARCH OBJECTIVES AND QUESTIONS ............................................................... 6
   1.4 DELIMITATIONS .......................................................................................................... 6
   1.5 RESEARCH METHODS ............................................................................................... 7
   1.6 THESIS CONTRIBUTIONS ......................................................................................... 8
   1.7 STRUCTURE OF THE THESIS .................................................................................. 8

2 LITERATURE REVIEW ........................................................................................................ 9
   2.1 OVERVIEW ............................................................................................................... 9
      2.1.1 Building Automation Systems ........................................................................... 9
      2.1.2 JavaScript, JavaScript frameworks and Dart ..................................................... 12
      2.1.3 Software metrics and Maintainability ............................................................... 17
   2.2 RELATED WORK .................................................................................................. 22
      2.2.1 Home automation frontends design ................................................................. 22
      2.2.2 JavaScript Frameworks Comparison ................................................................. 25
      2.2.3 Measuring Software Maintainability ............................................................... 32
   2.3 SUMMARY OF THE LITERATURE REVIEW ............................................................ 36

3 RESEARCH METHOD ....................................................................................................... 37
   3.1 QUALITATIVE DATA ANALYSIS APPROACH ......................................................... 39
      3.1.1 Data collection .................................................................................................. 39
      3.1.2 Data analysis ...................................................................................................... 40
   3.2 HOME AUTOMATION FRONTEND SCENARIOS .................................................. 40
   3.3 ONLINE SURVEY DESIGN .................................................................................... 41
   3.4 SUMMARY OF THE RESEARCH METHOD ............................................................ 42

4 IMPLEMENTATION .......................................................................................................... 43
   4.1 SYSTEM ARCHITECTURE ....................................................................................... 43
   4.2 DEVELOPED USER INTERFACES .......................................................................... 44
   4.3 SOFTWARE METRICS EVALUATION ...................................................................... 51
   4.4 SUMMARY OF THE IMPLEMENTATION ............................................................... 52
**LIST OF SYMBOLS AND ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAS</td>
<td>Building Automation System</td>
</tr>
<tr>
<td>CC</td>
<td>Cyclomatic Complexity</td>
</tr>
<tr>
<td>Dart</td>
<td>Google Dart Language</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>ECA</td>
<td>Event-Condition-Action</td>
</tr>
<tr>
<td>GeSi</td>
<td>Global e-Sustainability Initiative</td>
</tr>
<tr>
<td>HAS</td>
<td>Home Automation System</td>
</tr>
<tr>
<td>IEEE</td>
<td>The Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>JS</td>
<td>JavaScript</td>
</tr>
<tr>
<td>JSf</td>
<td>JavaScript framework</td>
</tr>
<tr>
<td>JSX</td>
<td>JavaScript XML</td>
</tr>
<tr>
<td>MI</td>
<td>Maintainability Index</td>
</tr>
<tr>
<td>SPA</td>
<td>Single page application</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UX</td>
<td>User experience</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

This work carries out a research on the most popular frameworks and Dart language in the field of home automation systems. The differences in software architecture, the support of design patterns and the scope of framework capabilities will be analyzed. This chapter presents the introduction to the research work.

1.1 Background

According to Smarter2030 report by GeSi [1], buildings consume over 40% of the total energy worldwide and it is believed that smart building solutions can alleviate the problem. In the EU, CO$_2$ emissions by buildings represent 36% out of the total greenhouse gas emissions [2]. The energy consumption of buildings can be reduced by using building automation systems (BAS) as a smart building component, while at the same time these smart systems can also bring comfort, climate control, security and others [3].

However, regardless of abounding benefits, there are several issues pertaining to large-scale adoption and utilization of BAS by households. Inhabitants without technical skills have huge challenges to manage, troubleshoot and eventually cannot take an advantage from the Home Automation Systems (HAS) [4]. Difficulty of customization of home automation systems (HAS), the complexity of their user interfaces and their prices are the main barriers of adoption these systems [5]. However more commercial smart home technologies are being released and available over time which is also influencing their prices and their costs are declining [6]. Ultimately, the main limitations of the HAS are their user interfaces. Smart home frontends face challenges to incorporate all features about navigation, conveying large amount of information in understandable format, supporting easy interaction for users and others [4].

Usually BAS are controlled by the UIs made of browser-based web applications. JavaScript is amongst the main languages for web technologies and it stands in solid top 10 of most used programming languages [7]. Nevertheless, JavaScript frameworks (JSf) have been created to ease programmers’ tasks and the number of these JSf grew dramatically counting for hundreds [8]. Only few studies compare JSf considering source code
maintainability, ease of programming, ease of design. Therefore, this thesis carries out an empirical analysis of JSFs and Google Dart programming language (Dart) for intuitive, widely acceptable BAS frontends. This leads BAS frontend developers to the choice of an optimal framework for their needs with a help of experimental documentation.

1.2 Motivation

Around half of the total produced energy is consumed by buildings, houses and almost 40 percent of the carbon dioxide comes from the buildings sector [1]. In fact, carbon dioxide is known as the greenhouse gas that is responsible for temperature rises. Today we can see that changes in the global temperature even for few centigrade may end up with catastrophic consequences raising sea levels, climate shifts, and unpredictable weather conditions. These facts can show the essence and urgency of taking actions to reduce these emissions via reducing energy consumption.

Energy consumption of buildings could be reduced using home automation systems. There have been many facts about the energy saving potentials of smart homes [3]. However, any new technology may face challenges in its early adoption phase and sometimes even resistance to use it. However, in case of home automation technologies the primary problems are their complex user interfaces that are hard to use [5].

Therefore, this thesis work aims at exploring JavaScript frameworks and Dart language as the promising and widely accepted technologies worldwide to build home automation user interfaces. This research primarily investigates the ease of development, maintainability and software design pattern aspects of JS frameworks and Dart. It is acknowledged fact that maintaining costs of a software product escalates with the increase of efforts needed to maintain a software component. Hence, the overall budget of the software product increases as well [9]. Thus, the main goal of carrying out this thesis work is to reduce the development and maintenance efforts of developers to build HAS frontends and by this to contribute to the popularity of home automation systems. The contribution from this study to the energy savings may not be immediate however even a few number installations of home automation systems by people or reducing efforts to build a HAS user interface will bring about positive enabling effects.
1.3 Research objectives and questions

The objectives of this thesis work are the following:

1. Assist home automation frontend developers to select a particular JavaScript framework or Dart language through providing comparative analysis of these frameworks.
2. Investigate the support and benefits of the MVC/MVP design patterns in different JavaScript frameworks and Dart to identify their relevance for programming home automation frontends.
3. Compare maintainability, design aspects of JavaScript frameworks and Dart in different HA scenarios in order to reduce HAS frontend development efforts.

The following research questions are addressed within this study:

1. What are the main differences of JS frameworks as well as Dart?
2. How are the MVC/MVP design patterns supported by the selected JS frameworks and Dart?
3. Which maintainability metrics can be applied and compared from JS frameworks/Dart?
4. In which HAS scenarios JS frameworks/Dart to be compared?

1.4 Delimitations

Many home automation systems exist in the market such as Fhem, Calaos, Home Assistant and others. Developers can also choose a preferred building automation protocols from the set of many well-known ones such as KNX, ZWave, M-Bus, EnOcean, HomeMatic and many others. This thesis does not evaluate these protocols and focuses on frontend software source codes. Also, this work is completed adopting existing open source Home Automation (HA) server – FHEM as a platform for the experiments. The FHEM server is written in the Perl language and supports a huge number of HA protocols and additionally, up-to-date modern technologies like Alexa, iTunes, TV sets and many more. Worth
mentioning that, FHEM system has a quite primitive UI and a complex configuration process.

In addition, the scope of the communication means from designed user interfaces and to FHEM home automation server is only Websockets connection. Due to convenience and availability of Websocket libraries to almost all JS frameworks and programming languages, this thesis work utilizes Websockets. Moreover, worth to mention that Websockets are one of the only few existing connection means to FHEM platform developed by the FHEM contributors. Therefore, other technologies such as REST API are not reviewed and the thesis scope is limited to Websocket usage. Lastly, the styling aspect of the frontends is also not considered as a part of the main research process and only JavaScript as well as Dart source codes are the main focus of the thesis.

1.5 Research Methods

In order to answer the research questions one and three – RQ1 and RQ3, a comprehensive literature review is done. This forms a solid blueprint for the following steps of the research and identifies missing gaps in JS frameworks comparison works. Specifically for the RQ3, it is important to find out which exact software metrics to employ in this study and by which tools we can cover as many frameworks at once to gain standard results. In addition, answering the RQ1 is done through deeper systematic approach via data collection and mapping study. The reason for that is comparative works on JS frameworks are highly distributed over the articles and research papers where if some frameworks are compared to each other, others are not included and vice versa. Analyzing MVC design pattern support as outlined in RQ2 requires writing the scripts in real projects. Moreover, online surveys are intended to present developers point of view in this study. Lastly, for the RQ4, few home automation scenarios have been designed which take into account and alleviate the main issues the end users face nowadays to adopt HAS. These obstacles are already mentioned and include complexity of the UIs and configuration, navigation in the application, energy monitoring and others. The designed UIs consider these issues and are represented as the solved UIs themselves.
1.6 Thesis contributions

The main contribution of this thesis work is to provide practical guidance for home automation user interface developers, supplying social and technical aspects of the sustainability with valuable documentation. Due to high popularity of JavaScript frameworks in the modern and interactive user interface development in the world, the comparative exploration of these technologies would facilitate the acceptance and expansion of home automation frontends built using JavaScript frameworks. The large-scale utilization of home automation technologies is in fact a tremendous driver for energy savings in the building sector.

1.7 Structure of the thesis

This thesis work is structured as follows:

- Chapter 2 highlights the scope of the thesis. It presents all important concepts of home automation systems, JavaScript frameworks, User interface requirements, software maintainability aspects as well as related works in the domain of JS frameworks comparison and methods of measuring software maintainability. The methods to compare JS frameworks are also presented in this chapter.

- Chapter 3 shows the research process. The thesis methodology, the online survey approach, the architecture of the experimental system and implementation of the home automation frontend source codes are summarized in this chapter.

- Chapter 4 describes the experimental platform on which the UIs are tested and software metrics are obtained.

- Chapter 5 presents the results from qualitative data analysis and software metrics obtained from the JS frameworks and Dart codes, online survey results and the MVC design pattern support and benefits by the frameworks.

- Chapter 6 discusses all results collected from the experiments.

- Chapter 7 summarizes the thesis work and introduces possible future work in the field.
2 LITERATURE REVIEW

This chapter presents the base concepts of home automation systems, architecture and the background of the selected JavaScript (JS) frameworks, software maintainability measurement methods and the previous works done in this field of JavaScript frameworks comparison. In addition to JS frameworks, the general concepts of the Dart language are also reviewed in this chapter. This chapter plays a role of a foundation for the research process.

The literature review showed that many existing works on the JSf comparison were not performed for the home automation frontends development or for a specific application design. The previous comparison studies are involved in JS features comparison without considering particular use cases.

2.1 Overview

This section highlights the generic information on building automation systems, JavaScript and its frameworks and the main concepts of the Dart language. The FHEM platform, its add-ons and communication interfaces are discussed as well. Moreover, software maintainability definition, its importance in software development processes, factors that may impact it and metrics to measure maintainability are reviewed in this section.

2.1.1 Building Automation Systems

Building automation systems (BAS) perform control and monitoring of temperature, lights, ventilation, security and other home appliances. Tasks can be automated with the use of BAS. Besides control and automation ability, these systems have a large potential to reduce energy consumption [3] [10]. BAS technologies are not a novation and many standards have been developed such as Modbus, BACnet, KNX, ZigBee [10] [11].

HAS frontends usually display a list of rooms or devices. Alternatively these UIs may also visualize the house in the form of floor and sometimes room plans for better accessibility to the appliances [12] [4].
In [13] and [6] two ways of interaction with HAS is shown – user-control and rule-based automation. Any tasks such as switching appliances on and off is performed via control panel in a user-control type of HAS. In a rule-based HAS applications, users are not expected to write a complex code for rules although some HAS may require this such as FHEM. Rule-based automation is achieved with the support of conditional commands such as “if”, “or”, “and”. For example, rules can be expressed in the way similar to – (“if” the door “opens”, then “activate” alarm). In addition this type of automation can be expanded as trigger-constraint-action which is also referred as Event-Condition-Action – ECA [6].

ISO 16484-5 standard [14] defines a standard model for any type of BAS layering it into three composite levels. Fig. 1 illustrates three levels of BAS. The lowest, Field level is composed of all hardware elements – sensors, actuators and each physical interaction with the environment belongs to this level. The second, intermediate level – the automation level is responsible for controlling devices from Field level that is managed by controllers. The last and the upper level is the management level where the whole BAS is monitored and controlled. This top level includes web servers, databases, protocol translating servers and UIs. This thesis performs research within this Management level of the BAS model not going into deep investigations on the other low levels.

According to Sánchez, et al. [15], at the present time developers of HA applications mainly use software tools provided either by the device manufacturer or based on their own personal experience. Moreover, BAS are usually developed by experts of the field who programs the system, selects and installs the needed hardware after gathering user requirements, inclining towards a particular technology of the manufacturer. The usability of HAS frontends is however, often neglected.
One of the well maintained BAS platforms is FHEM. FHEM is an open-source home automation server platform written in Perl language. FHEM can be controlled through its built-in frontends, telnet connection or via TCP/IP. It maintains many home automation protocols, media devices, weather service, Google calendars and other appliances. The supported popular protocols are the following: KNX, ZWave, EnOcean, Philipps HUE, FRITZ!Dect, Intertechno, HomeEasy, X10 and others. FHEM’s documentation includes detailed command list with description Commandref¹, Wiki Page² (in German), Forum³ (in German) and YouTube tutorials (in German).

In order to configure, program devices or send commands to them, one of the following command types can be issued to FHEM:

- FHEM specific commands – e.g. set lamp off
- Shell commands – "fhem.pl 7072 "set lamp off"" or
- Perl expressions – {fhem("set lamp off")}

The built-in frontend of FHEM is not easy to configure and not intuitive which is illustrated in Fig. 2. By default, only the style of the UI can be changed in FHEM which is change of background color and template for desktop/mobile.

---

¹ https://fhem.de/commandref.html
² https://wiki.fhem.de/wiki/Hauptseite
³ https://forum.fhem.de/

---

Fig. 2. The default frontends of FHEM
However, FHEM is a modular program, meaning that different devices belong to different modules. Some of the modules which can significantly improve the UI are Floorplan and InfoPanel. Moreover, the FHEM developers have created the following supplementary UIs and helper developments such as:

- Fronthem module – interface between FHEM and smartVISU frontend
- FHEM Tablet UI illustrated in Fig. 3.
- Charting Frontend
- Yet Another Frontend

As a communication means to FHEM:

- FHEM.JS is a Websocket connection implementation to FHEM, based on NodeJS server.

The fronthem module as an interface between FHEM and smartVISU frontend is achieved via websockets interface. Tablet UI is based on jQuery library, font-awesome library and other dependencies. Charting frontend requires FHEM’s DbLog module and MySQL, Postgres or similar database already installed. Yet Another Frontend YAF is based on JavaScript frameworks JQuery and JQuery UI.

There have been some efforts of building UIs for FHEM based on AngularJS with Websockets which is still under development and improvement. However, no projects exist in FHEM community with user interfaces written in Dart, Ember or React framework.

### 2.1.2 JavaScript, JavaScript frameworks and Dart

JavaScript is a prototype-based high level scripting language. This language has a high flexibility as functions in JavaScript can be objects or may have properties and nested

---

4 [http://www.smartvisu.de/](http://www.smartvisu.de/)
5 [https://github.com/knowthelist/fhem-tablet-ui](https://github.com/knowthelist/fhem-tablet-ui)
6 [https://wiki.fhem.de/wiki/Neues_Charting_Frontend](https://wiki.fhem.de/wiki/Neues_Charting_Frontend)
7 [https://wiki.fhem.de/wiki/YAF](https://wiki.fhem.de/wiki/YAF)
8 [https://github.com/winne27/fhem.js](https://github.com/winne27/fhem.js)
9 [https://github.com/alecosborne/neohome](https://github.com/alecosborne/neohome)
function closures, can be assigned to other objects and they can be stored in variables, objects or in arrays. They can also be passed as arguments to functions and functions can return functions [16]. Moreover, JavaScript allows prototypes, first-class functions and closures. The interaction with the browser is performed via a complex event-based mechanism. However, writing in JavaScript and maintaining its code is hard due to such high flexibility [16].

JavaScript is widely used for building interactive web applications by developers. Because of its high popularity, dozens of JS frameworks have been developed to ease the code writing process. A software framework is a collection of libraries that comes with its software architecture and principles for designing applications. Many popular JS frameworks (JSf) are built in MV* software design pattern where M-model, data for the application, V-stands for view which is how the template is displayed for users. The star character can be C-controller, P-presenter or VM-view-model. [17] MVC pattern is widely used for building user interfaces and helps separate the source code logic into three parts – model, view and controller [18]. The simple architectural view of MVC pattern is described in Figure 2. In AngularJS, Ember and in other JS frameworks the View is defined as a template in HTML while Model and Controller is defined in a code written in JavaScript [19]. AngularJS and Ember have strong template features for easing data display and manipulation. One of them is Data Binding feature which is automatic way of updating the View in case of any changes in the Model [20].

AngularJS is a Javascript Framework which is released in 2009 and the current latest version is 1.6.7 as for 24 November 2017. AngularJS is built on MVVM (Model-View-ViewModel) design pattern. The main features of AngularJS are the following. Scope in AngularJS is an intermediary object between a Controller and View and it is referred as a Model. Template is an HTML file extended with AngularJS directives that add behavior and attributes to HTML elements. Processed and rendered template is called as the View. Controllers are JavaScript functions that belong to a particular scope object and they are responsible for application functionality and logic. AngularJS also composed of Expressions, Compiler, Filter, Components, Module and Services. Automatic data-binding,
Routing, Dependency Injection, Animations and other features are supported in AngularJS. The architecture of AngularJS is shown in Fig. 3.

![AngularJS Architecture](image)

**Fig. 3. AngularJS architecture**

Angular (Angular2+ versions) is a JavaScript framework which was released in September of 2016 and is maintained by the Google company. Since Angular2, other versions were also introduced over time – Angular 4 (no Angular3 version), Angular5 and the latest version Angular6 is available from March 2018 [21] [22].

One of the most distinct features of the Angular is that Angular2+ applications are written in TypeScript language as opposed to old AngularJS versions which are JavaScript based. Although TypeScript can be a learning obstacle for developers, it supports all the features existent in JavaScript language. Most importantly TypeScript is an object-oriented language [22]. Another noticeable change that took place in Angular2+ is the shift from the MVC to component based architecture. The previous concepts like controllers and scopes have been replaced by components and directives as the main building blocks. In this case components are separate application units that can be reused with adding them to other applications [22]. A component might be a nested or child component and every

---

10 [https://angularjs.org/](https://angularjs.org/)
component is represented in 3 files [23]. Moreover, the modularity has increased in Angular2. Unnecessary modules for a project can be excluded from installation. For example, a programmer may not need to use Angular forms module and therefore is free not to include it in a project. However, similarly to Angular 1.x (AngularJS), HTML remains as a placeholder for templates in Angular2+ applications [22].

Ember is one of the oldest MVC-based JS framework as the AngularJS and it is also considered as one of the very difficult frameworks to learn. The first version was released in 2011 and the recent version is 3.0.0 from February 2018. Likewise Angular, React and Vue, Ember utilizes a component-based architecture. The general architecture of Ember can be seen in Fig. 4.

Ember consists of adapters, components, templates, controllers, models, routes, services and others [23]. Components in Ember are responsible for the view logic and behavior. Any Component is composed of its own Template and a JavaScript file that defines the behavior of the Template. Obviously, Templates are user interface views of an Ember application. Controllers operate very similarly to a Component in Ember and interestingly, Components are more preferred over Controllers since last versions of Ember.\(^{12}\) Route renders a particular Template and loads the model for the Template. Model is a class representing the data for an application and is responsible for retrieving data from a database. Ember command line interface CLI is considered as one of the best feature in Ember. This feature allows creating all Ember composites like controllers, routes and installing dependencies via terminal commands. In fact, many of the newer JS frameworks were inspired by the Ember and many Ember features were adopted by these frameworks. A Router feature in VueJS frameworks is an example of such contribution by Ember.

\(^{12}\) [https://emberigniter.com/should-we-use-ember-controllers/](https://emberigniter.com/should-we-use-ember-controllers/)
React is a JavaScript library for creating web application which was first released in March 2013. According to [23], React framework uses reactive approach and functional programming paradigm. React has component based architecture and uses unidirectional data flow. Likewise Angular, a React application consists of a single or multiple components. A component is responsible for business logic and a template. React templates are JSX templates which stands for JavaScript XML (JSX). The inter component communication can be established via Flux or other supporting libraries. Moreover, React offers special objects – so called state and props. These objects allow passing data from a component to the template or to a child component. One of the main reasons for React’s popularity is the Virtual Document Object Model – (DOM) which re-renders only a small piece of the changed view instead of the whole elements of the template saving time for app building [22]. However, the development in React is considered as the challenging task for developers – React uses a bunch of extra tools such as Flux, MobX, Redux, Fluxy, Fluxible, RefluxJS, jQuery AJAX, fetch API, Superagent, Axios and other to achieve flexibility [23].

Chalin in [24] introduces Dart language or DartLang as an open source, object-oriented language which can also be compiled into JavaScript code. It is a dynamically typed, class-based language and supports single inheritance. The initial version of Dart became available in 2011 and the latest version was released in June, 2017. Dart language has adopted Angular2 framework as a Dart version of the Angular2 framework. This

\[13 \text{https://blog.repositive.io/building-apps-with-ember/} \]
framework is called as AngularDart and is basically Angular2 framework in Dart language [25]. The architecture of Angular2 as well its Dart version – AngularDart framework is illustrated in Fig. 5. AngularDart framework is intended for developing web application. Another Dart platform is Flutter which is an SDK (software development kit) for creating mobile apps.14

![Image](image.png)

**Fig. 5.** The architecture of Angular2 & AngularDart framework

### 2.1.3 Software metrics and Maintainability

Many studies have shown that post-delivery maintenance of software is highly resource consuming in terms of time and developer efforts. It is estimated that around 40-70% of the whole software development process is allocated on maintenance activities. Up to 75% of the project budget may be spent at this phase of the software development life cycle making it the costliest stage. This makes maintainability critical to concede because of the major expenses that can arise due to making changes or fixes in a software system [26] [27] [28]. Therefore, it is essential for developers to estimate the future changes in a software project in advance [28].

In fact, software maintainability is widely acknowledged because of its significant economic impacts. The ability of software companies to adapt to and compete in unpredictable and changing business environment is affected by the time and resources for maintenance work and how efficiently they can upgrade their existing software products.

---

14 [https://www.dartlang.org/guides/platforms](https://www.dartlang.org/guides/platforms)
Therefore, the main burden for organizations comes from software maintenance activities rather than by the development of new software [28].

Software maintainability is defined as the efforts required for making modifications to the software [28]. In other words according to standard definition [29] the term maintainability is given as “the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment”. Once a software product is released for usage, some customer reasons may arise that require changes to the software. These requests include adding new functionality to the software, technology changes or implementation of a new hardware into a system. A need for a change can also arise because of operation failure, prevention of possible bugs, new requirements and others [26]. Changes made to the software are a part of maintenance activities [28].

Maintenance activities were classified into categories by Pearse & Oman [30] and [31] According to which the following the types of maintenance activities on a software system or a component may occur:

- Corrective: Maintenance activity performed to correct faults in hardware or software.
- Adaptive: Software maintenance performed to make a program usable in a changed environment.
- Perfective: Software maintenance performed to improve the performance, maintainability, or other attributes of a program.

Software metrics are one of the main elements in software measurement process that are measured throughout the whole software development process [32]. A systematic mapping study was conducted by [32] for the period from 2010 to 2015 on selected 226 studies and systematized the majority of software metrics found in the scientific research works. In total of 255 software metrics have been extracted from the sources. Over 85% of the reviewed works covered in the study pertains to object oriented software paradigm (OOP) metrics. Table 1 illustrates the top 10 most frequently used OOP metrics sorted in the research study. Allocation of the OOP metrics by category of studies falls into the
following distribution – Quality (15), Complexity (14), Change (9), Cohesion (10), Maintainability (8), Refactoring (8), Coupling (4) and Reusability (4). The research states that the most applied method for metric extraction is using software metrics tools and the top 10 identified OOP metrics were extracted with the means of metric extraction tools. However it was mentioned that the tools are dependent on the metrics sets and the accepted programming languages. The results show that the overwhelmingly high percentage of the software metrics are intended for measuring object oriented software code.

Table 1. The most used OOP metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Total occurrences per metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weighted Methods per Class (WMC)</td>
<td>89</td>
</tr>
<tr>
<td>Coupling Between Objects (CBO)</td>
<td>89</td>
</tr>
<tr>
<td>Lack of Cohesion in Methods (LCOM)</td>
<td>86</td>
</tr>
<tr>
<td>Depth of Inheritance Tree (DIT)</td>
<td>81</td>
</tr>
<tr>
<td>Lines of Code (LOC)</td>
<td>79</td>
</tr>
<tr>
<td>Number of Children (NOC)</td>
<td>77</td>
</tr>
<tr>
<td>Response for a Class (RFC)</td>
<td>72</td>
</tr>
<tr>
<td>Number of Methods (NOM)</td>
<td>57</td>
</tr>
<tr>
<td>Cyclomatic Complexity (V(G))</td>
<td>55</td>
</tr>
<tr>
<td>Number of Attributes (NOA)</td>
<td>43</td>
</tr>
</tbody>
</table>

Maintainability prediction metrics are primarily considered for object oriented software and the short list of these metrics include – Halstead volume, Cyclomatic complexity, Lines of code, Maintainability index, Coupling between objects, Number of constructors, Class size, Number of public methods and many others [33]. Since, JavaScript language partly supports OOP programming paradigm (prototype-based – a classless style of OOP), some of the metrics regarding software maintainability, code complexity found in the literature can be applied to JavaScript code. We consider only those metrics that can be applied for measuring maintainability and only those that can be applied for JavaScript source code.

According to [34] Cyclomatic Complexity (CC) is a software metric that measures the number of linearly dependent paths in a source code. This metric is broadly used in software industry. The lower value of the CC metric indicates less complexity of the
project source code. Thus, this metric can be a meaningful predictor of the development effort [35]. The CC value is calculated with a formula in Eq. 1. Where $V(G)$ is Cyclomatic complexity value, $e$ – the number of edges, $n$ – vertices (nodes) and $p$ is the number of components. “$p$” equals to 1 for one component. The vertices (nodes) are represented as blocks of code and the edges denote decision-points which connect a node to another [36]. The visual representation of the CC metric calculation example is shown in Fig. 6.

$$V(G) = e - n + 2p$$ \hspace{1cm} (1)

![Fig. 6. CC measurement example for a code with 3 nodes and 4 edges [35]](image)

Halstead metrics are another set of software metrics that measures software complexity from the operators and operands point of view [37]. Halstead metrics are considered as the best metric to measure program complexity [38]. Fig. 7 describes the main advantages of Halstead metrics [38].

<table>
<thead>
<tr>
<th>Halstead Metrics</th>
<th>Predicts error rate.</th>
<th>Predicts maintenance effort.</th>
<th>Simple to calculate.</th>
<th>Measure overall quality.</th>
<th>Used for any language</th>
</tr>
</thead>
</table>

![Fig. 7. The advantages of using Halstead metrics](image)

Halstead metrics are computed from the following figures of the software program:

- $n1$ = number of unique or distinct operators
- $n2$ = number of unique or distinct operands
- $N1$ = total number of occurrences of operators
- $N2$ = total number of occurrences of operands
Based on the above digits, Halstead Volume – V is measured with the following formula in Eq. 2:

$$V = (N_1 + N_2) \log_2 (n_1 + n_2)$$  \hspace{1cm} (2)

Difficulty of the program is calculated with a Halstead difficulty metric – D, Eq. 3:

$$D = \frac{2}{n_1} \times \frac{n_2}{N_2}$$ \hspace{1cm} (3)

Halstead effort – a metric which measures the efforts to implement or understand a program Eq. 4:

$$E = D \times V$$ \hspace{1cm} (4)

Maintainability index (MI) is a numerical metric that objectively measures maintainability level of a software project and its formula is composed of few other metrics. Eq. 5 represents the formula of (MI) which incorporates Halstead Volume (HV), Cyclomatic Complexity (CC), average number of lines of code per module (LOC) and percentage of comment lines per module (COM) desirably. The definition of the MI asserts that a system is more maintainable when the MI number is higher [39].

$$171 - 5.2ln(HV) - 0.23CC - 16.2ln(LOC) + 50.0sin\sqrt{2.46 \times COM}$$ \hspace{1cm} (5)

Although many software metrics are widely used for maintainability measurement, maintainability cannot be limited to only software characteristics and other factors should also be taken into account. Even though software metric may be an objective measure of the software quality, it may not correlate with maintainability [40]. For example, maintenance activities highly depend on developers who are doing the actual maintenance tasks. These factors include skills of developers, turnover rate, documentation, availability of supporting tools such as debuggers, visualization and configuration tools. [41].
Moreover, maintainability is itself composed of other factors that are analyzability, changeability, stability, testability and maintainability conformance [39].

From the literature it can be seen that defining maintainability is a complex procedure which involves using software metrics along with other external factors. There are also three types of maintenance activities that require different approach and tools to fulfill. The home automation platform and hardware are not changed in this study and therefore we focus on the perfective maintenance activities on the frontend codes.

2.2 Related work

This section presents previous works in JavaScript frameworks comparison, as well as proposed suggested and utilized metrics and approaches for comparison. Research works performed around other programming languages and not applicable software metrics for JavaScript language are not examined. However, OOP metrics usable for JS frameworks and Dart language are investigated in detail.

2.2.1 Home automation frontends design

In order for home automation systems accepted by the households, the usability aspect of such technologies must be considered. As mentioned in [5], the main issue for the end users is the complexity and low usability of home automation frontends. Therefore, this problem has to be taken into account for designing HAS user interfaces. The usability and user experience of HAS frontends is explored in [13]. The research identified the main problems of HAS UIs using qualitative data analysis method and validated the obtained results via prototyping a JavaScript based UI and through usability tests with the users. Other UI requirements such as neat layout of the presented information and the ease of modification of the UI in case of changes in the HAS have been examined.

It has been identified the importance of visualizing data and energy usage from the different devices in the form of understandable and meaningful information. Making users aware of their energy usage can be a critical assistance for influencing the user behavior towards more efficient energy consumption. According to [42] this can save around 10
percent of the electricity. Therefore it can be conclude that lower energy consumption can be attained through the feedback information to users. Eventually, the energy management feature on HAS frontends has a significant potential for achieving sustainability.

The Table 2 displays a set of the highly important HAS UI features that would contribute for higher user acceptance of HAS according to [13].

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Historical data access</td>
<td>Ability to visualize the past, present and future events</td>
</tr>
<tr>
<td>Energy monitoring</td>
<td>Displaying energy consumption per device to identify energy-hungry appliances at home</td>
</tr>
<tr>
<td>Automation feature</td>
<td>Schedule device automation events and the possibility to cancel it before it happens</td>
</tr>
<tr>
<td>Understandability</td>
<td>Using colors, easy-to-perceive device status</td>
</tr>
<tr>
<td>Energy analytics</td>
<td>Displaying the energy consumption within a given time period</td>
</tr>
<tr>
<td>Configuration</td>
<td>Possibility to configure home automation system easily</td>
</tr>
</tbody>
</table>

A study in [4] designs a user interface prototype named “Casalendar” for HAS focusing on the visualization aspect. Written in a JavaScript library jQuery, the user interface Casalendar displays the past, current and future automation events on the screen. To achieve illustrating this information, the prototype used two sort of temporal metaphors – calendar and clock. However, a clock metaphor was not accepted by the survey participants due to its ambiguity. The calendar type of user interface evinced higher interest among the surveyees. The research only evaluates the acceptance of temporal metaphors for visualizing the automation events. Other tasks such as generating the automation commands are not analyzed on the paper. The Fig. 8 shows the calendar based UI snippet where the events are placed on the calendar-alike template.
Fig. 8. the UI of the calendar based HAS events designed in [4]

End user programming of HAS is explored in [6] and provided vital guidelines regarding to designing HAS user interfaces that allows programming home automation system devices via setting rules. The guidelines from the research are in the following Table 3:

Table 3. UI guidelines for rule based HAS

<table>
<thead>
<tr>
<th>The guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Using Event-Condition-Action format for setting rules in a visual form</td>
</tr>
<tr>
<td>2 Not using the words “if” or “when”. This can be confusing for some users</td>
</tr>
<tr>
<td>3 Simplistic visualization of rules to attract users</td>
</tr>
<tr>
<td>4 Using tangible and non-tangible interaction</td>
</tr>
<tr>
<td>5 Providing small guidance or suggestions for users to better understand the</td>
</tr>
<tr>
<td>operation mechanism of the frontend</td>
</tr>
<tr>
<td>6 Separating time related properties. For example “at 10 p.m.” or “for 10 mins”.</td>
</tr>
<tr>
<td>7 Avoiding small screens for rule creation</td>
</tr>
<tr>
<td>8 Using simple interactions such as click, touch and avoiding complex ones</td>
</tr>
<tr>
<td>like drag-and drop</td>
</tr>
<tr>
<td>9 Complement visualization with sound or other feedback methods</td>
</tr>
<tr>
<td>10 Rigorous testing of the frontend for user evaluation</td>
</tr>
</tbody>
</table>

The authors have developed a frontend application for rule-based HAS considering some of the generated guideline rules. The UI application contains three areas which are Device Area – has all devices that can be selected for automation, Composition Area – the
placeholder for creating rules and lastly a Choice Area that incorporates actions, events and conditions to be used for automation. A final user interface from the study is given in Fig. 9. The frontend collects the data about existing home appliances via Dog Websockets API. Any created rules are sent to the HA server as JSON format. The findings from the research show that users prefer simplified interaction mechanism with the user interface – single touch, click. Cognitively this feature is found to be not requiring much effort.

**Fig. 9.** A rule based automation frontend

### 2.2.2 JavaScript Frameworks Comparison

The increase of JavaScript frameworks in the past years has led developers to a confusion to select a specific framework, since all of them can perform the same tasks equally well. Moreover, only few research works provide detailed comparison of the JS frameworks in real use cases. Existing comparative studies on JS frameworks does not give explicit distinctions for practitioners [43] and maintainability aspect is rarely considered. We review the works dedicated on the JSfs comparison where a few of them compare maintainability of these frameworks applying software metrics.
Alexander Svensson [44] measures the speed of MVC JS frameworks in milliseconds to manipulate DOM elements and display them on the screen. The experiment evaluates the speed of Angular/ Angular2.0, Aurelia, Backbone, Ember, Knockout, Mithril and Vue frameworks in creating, deleting and updating HTML elements (thousand rows table with six columns). Using Google Chrome TimeLine tool, they found Angular2.0 JSf got the highest results in all tests. However, the research claims that with the high speed of modern browsers these differences in milliseconds do not seemingly affect the visual performance on the screens. Additionally, the research only compared JSf in terms of performance without considering any software metrics or code maintainability.

Evaluation of AngularJS, Ember and Backbone is published online by [45] guided to a choice of the frameworks. The source mentions that to achieve data binding, AngularJS does not need getters and setters. Additionally, AngularJS code is shorter, meaning it takes less lines of code for a project, thus – less complex. The author also states that AngularJS and Ember are close to each other with the only advantage of a Router feature in Ember which is no longer the case since AngularJS upgraded this feature as well. On the other hand, Backbone significantly differs from Angular being a widget builder that creates HTML elements with JavaScript.

A book by McKeachie [46] mentions the importance of Routing feature in single page applications (SPA) that it enables switching between pages on the frontend. The comparative work demonstrated exemplar code differences of routing from the above JS frameworks. Code snippets in Fig. 10 show a simple routing example of switching between the two pages “Home” and “About”. This clearly shows the differences in terms of code size for routing, where Ember has the fewest lines of codes thanks to its in-built routing convenience.

<table>
<thead>
<tr>
<th>BackboneJS</th>
<th>AngularJS</th>
<th>Ember</th>
</tr>
</thead>
</table>
| var HomeView = Backbone.View.extend({
  template: '<h1>Home</h1>',
  initialize: function () {
    this.render();
  },
  render: function () {
    this.$el.html(this.template);
  }
}); | var routingExpl = angular.module('FunnyAnt.Examples.Routing', []);
routingExpl.controller('HomeController', function ($scope) {});
routingExpl.controller('AboutController', function ($scope) {});
routingExample.config(function ($routeProvider) {
  App = Ember.Application.create();
  App.Router.map(function () {
    // first parameter refers to the template
    this.route('home', { path: '/' });
    this.route('about');
  });
}); |
The authors also highlighted the main benefits and drawbacks of the above frameworks that are shown in Table 4.

**Table 4.** Advantages and disadvantages of three frameworks according to study in [46]

<table>
<thead>
<tr>
<th>Framework</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>AngularJS</td>
<td>Built-in dependency injection frees a dependency from being hard coded into a component. Also it is desirable to provide mock dependencies for test isolation.</td>
<td>Built-in router in AngularJS does not isolate the code for different views to be displayed on a single page.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AngularUI framework enables using named views for separation of a page into navigation, header, footer or sidebars parts with their own controller and scope.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>However, AngularUI needs to be</td>
</tr>
</tbody>
</table>

**Fig. 10.** Simple Routing Example to switch between two pages.
downloaded and loaded into source code as a dependency:
<script src="angular-ui-router.js"></script>

**Directives**
Many developers find it hard to write directives.

<table>
<thead>
<tr>
<th>Ember</th>
<th><strong>Router</strong></th>
<th>Powerful routing mechanism allows using named views with isolated code. Built-in router is as strong as AngularUI library for AngularJS.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Component Model</strong></td>
<td>Simpler components than AngularJS directives. Cleaner and more understandable API</td>
</tr>
<tr>
<td></td>
<td><strong>Testing</strong></td>
<td>Using getters and setters makes debugging more difficult than in AngularJS. No dirty checking in Ember as AngularJS does.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BackboneJS</th>
<th><strong>Lightweight</strong></th>
<th>Small in size, more library than a framework.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Performance</strong></td>
<td>More control on performance</td>
</tr>
<tr>
<td></td>
<td><strong>Productivity</strong></td>
<td>Backbone is more library alike – writing more code by the coder. Absence of 2-way data binding.</td>
</tr>
<tr>
<td></td>
<td><strong>Architecture</strong></td>
<td>Not clear architecture in some cases</td>
</tr>
</tbody>
</table>

In contrast to server-side applications which combine templates with data on the server and send to client, in single-page applications (SPA) server sends static templates and dynamic data that are combined on the client using JavaScript [46]. According to this study the efficient way to adopt and compare JavaScript MV* frameworks quickly is to breaking down them into a set of features. The author counts the following main features of MV* JS frameworks:

- Routing
- Data Binding
- Templates/Views
- Models
- Data Storage

The most important feature of the JavaScript MV* frameworks from the above list is data binding. Data binding convenience automatically updates any changes occurring in the
model data to the view or changes in the view is automatically reflected in the model as well. There are two types of data binding – One-way and two-way data binding where in the former, changes are spread from the model to the view only and in the latter, in both directions. Data binding is an essential aid for real time applications where any changes in a model need to be immediately displayed on the view and vice versa.

The study in [47] performed a survey with 460 AngularJS developers and identified the most valued and the most problematic features of AngularJS. Fig. 11 illustrates feedbacks from the responders on the most valuable features of AngularJS where custom components, dependency injection and two-way data binding got the highest scores respectively.

![Bar chart showing the feedbacks on the most valuable features of AngularJS]

**Fig. 11.** Primary features of AngularJS

As regards bad features that may cause possible issues in AngularJS, the research found out the following:

- Silent failures – using undefined functions or objects are used in AngularJS, it does not trigger exceptions, leading to silent application break down.
- Code is difficult in debugging – AngularJS code is not traditional “pure” JavaScript.
- Readability is low – in AngularJS application code can be spread throughout the HTML page impacting readability for developers.
- Separation of Concerns and Modularity – inserting too much JavaScript code in HTML code is a bad smell [48].

Interestingly the survey results on these features showed very low negative replies, with around 20% stated them as critical problems. Although the paper does not compare the similar features of the other popular MVC frameworks, such as Ember and React, the research method can be applied in this thesis for further comparative analysis of other JS frameworks.

Pano et al. [49] proposed a model of desirable JSf factors including usability, cost, efficiency and functionality via a qualitative interpretative study, interviewing 18 participants. Although the paper does not compare JS frameworks, it assesses the main implications developers tend to ponder on choosing a specific JSf. For example, developers prefer fewer lines of codes of a framework, readability of code, modularity, automatized event handling and DOM manipulation, browser support, precise documentation and others. These results compel us to use other detailed attributes to compare JSf.

A study in [43] mentions insufficiency of research that deploys software metrics on JavaScript frameworks and at the same time it notes that developers are induced by different motives to choose a JS framework. The study proposes a comparison framework Fig. 12 to assist a JSf differentiation by developers, the methodology which is based on applying software metrics on the same project using different JSf on the one hand, and on
the other – interviews with developers to find out practitioner inducements when choosing a JSf. After surveying four frontend developers it has been inferred that they had little knowledge on software metrics suggested by [49] and low interest on performance measurements too. All of them noted other criteria to select a JSf namely – documentation, community, so called “code less, do more”, updates and years of existence of frameworks. The comparison framework of the paper implies collecting software metric results and supplementing them with practitioner concerns in a database for a fast choice of JSf.

A significant contribution was made to the scope of JavaScript frameworks benchmarking comparison by [50]. The research compared AngularJS, BackboneJS and React frameworks on their performance differences, and similarly to [49], the following software complexity metrics – Lines of Code (LOC), Cyclomatic Complexity (CC), Halstead Complexity and Maintainability Index (MI). All measurements were conducted on the same TodoMVC\textsuperscript{15} applications written in 3 selected frameworks. The Todo application consists of the following UI elements:

- Input Field
- Task List where all items are displayed
- Footer which contains buttons “All”, “Active”, “Completed” and “Clear Completed”

For the performance comparison, the following tasks were executed on each JSf: adding 100 items, marking all 100 items, clear completed button activity which clears the list. This test results were obtained in milliseconds. For the second part of the experiment, the study extracted the metrics with the tool named complexity-report\textsuperscript{16} which needs to be installed on the Node.JS server first. Retrieving all software metrics was done per function in the source code and the following outcomes were gathered in Fig. 13:

\textsuperscript{15} http://todomvc.com/
\textsuperscript{16} https://www.npmjs.com/package/complexity-report
Fig. 13. The results of software metrics comparison for React, BackboneJS and AngularJS

2.2.3 Measuring Software Maintainability

A research in [33] measures maintainability of four object oriented software projects written in Java language using OOP metrics. The research considered a change made to the selected software as a maintenance activity. Addition, deletion and modification of a line in a source code were taken as change units. Interestingly, addition and deletion of a line of code is counted as one change whereas modification – as two changes. The maintainability prediction model utilized in the study consists of analyzing a change in software metrics before and after a modification is done to a software project. The results of the experiment show that Halstead bugs, Comments lines of code, Number of commands, Number of inner classes revealed more accurate prediction of the maintainability. However, this approach can be applied to only OOP software projects, for example written in Java, C# or C++. 
A study in [40] suggests a two-dimensional quality model for defining maintainability. According to the author, the following methods can be used to evaluate software maintainability – Metric-based approach, quality modeling and lastly process and process models. In a metric-based software maintainability definition approach, some internal attributes of software is measured by well-known software metrics such as lines of code, Halstead volume, Cyclomatic complexity and the Maintainability index which combines the followed ones. These attributes are used as objective indicators of software maintainability. However, these approaches can obscurely determine software maintainability and have substantial limitations [40]. In the proposed 2-dimensional maintainability definition model, activities involved in maintenance process are also included. In this case, the activities are mapped into the standard maintenance processes defined by the IEEE 1219 standard as illustrated in Fig. 14. However, this model represents relations between activities and factors in a Boolean relation. Which means that either the factors impact activities or does not impact at all. Therefore, a relative weights need to be used that lay within 0-1 interval to present the degree of factors' influence to maintenance activities more accurately.

![Maintainability matrix according to [40]](image)

**Fig. 14.** Maintainability matrix according to [40]

The research in [26] analyzed design decisions that influence maintainability. They investigated change effort as maintainability measure. Change effort includes total analysis and programming efforts spent to make a change on the software. There are also other
attributes that impact change efforts such as documentation, communication, deployment and testing.

The study identified the following factors that impact software maintainability:

Software properties:
- Specification-level properties, e.g. correctness or functional size
- Design-level properties, e.g. modularity, coupling
- Code-level properties, e.g. code size, complexity, maturity, decay, duplication

Maintenance process properties:
- Frequency of changes
- Maintenance procedures
- Testing process properties, e.g. test coverage

Resource properties:
- Team properties, e.g. activity rate, communication structure, personnel turnover
- Team member properties, e.g. skill level, familiarity with the system
- Maintenance infrastructure, e.g. development environment, tools

Change properties:
- Change functional properties, e.g. functional size, change type, functional complexity, performance sensitivity
- Change implementation properties, e.g. change size, change span, fault potential

Based on the whole aspects mentioned above, the maintainability can be seen as a combination of the software and a group of maintaining developers who is involved in the maintenance process. Since software has its internal attributes such as complexity, a team of developers has also its attributes that might be years of experience or working hours on a project. Eventually, software maintainability solely cannot be regarded as a software quality since it is influenced by a personal who maintains a software project. As an example, we can assume that a developer with higher experience on the particular software system can maintain it much better than others who have little awareness on the same system [26].

A study in [9] analyzed the maintainability of the different versions of a software product named “NUnit” via varied metrics. The software versions in the study differ from each
other having new functionalities as bigger changes, or with insignificant modifications as minor changes. The authors identified and measured the following factors that affect software component maintainability:

Size:
- The larger size of a software component makes it more difficult to maintain, takes longer time to understand. As a result, the maintaining efforts increase.

Complexity:
- This factor declines software maintainability and positively correlates with the size of software.

Compliance:
- This factor is measured by determining if the software was developed complying with the standard rules or not.

Maintainability index:
- A numerical value that measures source code maintenance ease.

Misra & Cafer [51] introduced a new JavaScript complexity measure metric – JavaScript Cognitive Complexity Measure (JCCM) as an indicator of JavaScript code maintainability. The paper indicates that size of lines of codes, number of Arbitrarily Named Distinct Variables (possibility for variable names to be taken arbitrarily), Number of Meaningfully Named Variables, Cognitive Weights of Basic Control Structures (advanced building blocks of a software code – condition, loop, function etc, each of which have a weight from 1 to 3). The calculation formula for JCCM is given in Eq. 6.

\[
\text{JCCM} = \sum_{i=1}^{n} \sum_{j=1}^{m_i} (((4 \times ANDV + MNV + Operators) \times CWJ_{ij})
\]

According to JCCM formula, JCCM equals to the sum of modules $i$ with $m_i$ line of code. As a demonstration of formula usage, Misra et al. [51] exemplified three JavaScript code
and calculated JCCM. To empirically validate the formula appropriateness, they conducted another experiment with 30 source codes of different projects. After analysis they observed that JCCM correlated with Cyclomatic Complexity, Logical Lines of Code and Halstead software metrics.

Another study in [49] attempted to choose the most proper software maintainability metrics that could be applied in JavaScript and JSf. It evaluated lines of code (LOC), number of statements, comment lines and comment lines to lines of code ratio as size metrics. For the complexity metrics Cyclomatic Complexity (CC), Branches and Depth have been chosen. Most importantly, the study opted for Halstead metrics – Halstead volume and Halstead level of a program as well as the Maintainability Index (MI). These mentioned metrics were calculated using JSmeter (jsmeter.info), Cloc (cloc.sourceforge.net) and Understand (scitools.com). However, the paper investigated those metrics on ExtJS, MooTools, Prototype, YUI, Dojo and jQuery, last three of which are JS libraries rather frameworks. The results show that jQuery and YUI libraries have relatively low maintainability since they had higher CC and MI.

2.3 Summary of the literature review

As it can be seen from this chapter, analysis of the JavaScript frameworks and their maintainability, relevant software metrics and correctly applying them in the field of home automation user interfaces require comprehensive and multi-dimensional approach. This chapter has analysed home automation systems operation principle, in which of its level we can do investigations and home automation system example – FHEM platform. In addition a holistic overview to the JS frameworks have been done.

The literature review shows that only few software metrics can be applied to JS frameworks. In addition, in the related works section we analyzed the HAS user interface requirements complied to which the later UI scenarios will be designed.
3 RESEARCH METHOD

This chapter presents the research process and methods employed in this study. The research methodology of the thesis represents mixed type of qualitative data analysis methodology and quantitative research methods. The overview of the research steps is illustrated in Fig. 18. Ready-to-deploy frontend solutions for FHEM technology will also be available in the end of the study.

In the first step of the research process shown in the Fig. 15, a qualitative data analysis methodology is employed in order to answer the research questions 1 and 4. The main tool applied in this phase is the document review for collecting and systematizing the previous works on the topic. Although none of the preceding papers investigated JS frameworks comparison for the home automation frontends, they contain valuable information on JSFs differences and qualities.

However, the quality attributes of any software product are also measured and evaluated via software metrics. Therefore, a quantitative analysis of the UI scripts written in different JS frameworks is done in the next step of the research process. This would allow the thesis work to express the comparison results numerically using software metrics. Metrics measurement process is achieved using the usable metric analysis tools.

To proper way to compare JS frameworks is de facto, producing the scripts from different JSFs on the same application where these projects have the same functionality and visually identical. This approach is described in a comparison framework suggested by [43]. Hence, to avoid bias and to achieve objective comparison, this thesis performs code metrics evaluation on JSFs in the same UI scenarios but using the selected JS frameworks and Dart. The scenarios design is described in the following sections.

JavaScript frameworks are based on specific design patterns such as MVC, MVP or MVVM. This thesis work also explores the support of the main pattern – MVC by the selected JS frameworks. Moreover, the benefits of them for HAS frontend applications need to be identified. For this purpose we test and evaluate these patterns through
scenarios. The ultimate aim from this phase is to show which pattern is implemented easier and in which JS framework or Dart.

In the last stage, two online surveys are also conducted with developers of JavaScript frameworks. The experience level of the developers varies in years. The main purpose of this questionnaire is to explore the maintainability of JS frameworks. The results from the qualitative data analysis and software metrics comparison serve as supplementary attributes for investigating maintainability.

![Research Process Workflow Diagram]

**Fig. 15.** The research process workflow
3.1 Qualitative data analysis approach

The method used in the first step is Qualitative Data Analysis (QDA) with inductive content analysis approach. The main goal of inductive approach in QDA is to produce new knowledge, in the form of summary from the raw textual data. In inductive data analysis approach so called “categories” are created through the coding process that emerges from the set of raw textual materials. 17

The final outcomes from the inductive QDA approach in this work are delivered in the form of external software qualities of the JS frameworks and Dart. We know, software maintainability cannot be blindly measured using software metrics only and therefore, comparison of other attributes of JSf and Dart is essential to determine their maintainability.

3.1.1 Data collection

In the first stage of the thesis research method data from the previous studies were collected in order to obtain comparative information about different JavaScript frameworks and Dart. Few of the published research papers on JS frameworks comparison are available on the most popular e-databases such as IEEE, ACM and Springer. However, the majority of comparative studies are found on the online blogs, developers` support websites and online documentation of the JS frameworks. Therefore, the data collection process covers all available materials, dedicated on the topic including personal blogs of developers. In total twenty three sources have been reviewed.

Preliminary data analysis showed that the distribution of the specific comparison material is spread around the sources. Each of the sources gives various comparisons of the different frameworks. For example, a source may contain qualitative data about AngularJS, Angular2 and Ember frameworks, missing others.

3.1.2 Data analysis

After the data collection process is completed, the obtained information was analyzed using the open coding technique. The open coding technique involves reading and re-reading process through the collected data iteratively where it is necessary and breaking the whole data down into more specific pieces of meaningful information. 18 19

The observations on the results obtained from the qualitative data analysis show that the JavaScript frameworks are compared by the different external and internal software qualities. The compared internal software qualities are the framework source code size, lines of code etc. Other attributes include community support, framework popularity, the quantity of the developed projects using a framework.

3.2 Home automation frontend scenarios

The scenarios for prototyping the user interfaces are based on the specific usability requirements in order to meet the challenges mentioned in the previous chapters about the obstacles for accepting HAS. First of all, the design guidelines investigated by [13] will be taken into consideration and other approaches for information visualization in HAS are included. The best practices for the rule-based automation user interface primarily a study by [6] is applied in the UI scenarios. Eventually, the purpose for designing UI scenarios is to achieve the main thesis goal – differentiating the most popular JS frameworks and Dart in terms of maintainability in different use cases.

The UI scripts will be tested with FHEM home automation platform and evaluated with code metrics and design patterns support. In the end, all results are summarized for three main scenarios defined previously in the form of tables and diagrams. The UI scenarios along with their descriptions are given in Table 5. However, three main scenarios have been fully analyzed.

19 https://www.projectguru.in/publications/qualitative-content-analysis-research/
### Table 5. The proposed scenarios for the user interfaces

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic control</td>
<td>All devices can be turned on and off immediately. Device status is visible.</td>
</tr>
<tr>
<td>Advanced control</td>
<td>“All lights off” features, room-based control, different modes</td>
</tr>
<tr>
<td>Basic automation</td>
<td>All devices can be scheduled for turn on and off in a timely, daily manner</td>
</tr>
<tr>
<td>Advanced automation</td>
<td>All devices can be scheduled in daily basis using the rule-based complex automation composition mechanism. This method is described in [6] and by event modes (vacation mode, security mode, savings mode)</td>
</tr>
<tr>
<td>Basic monitoring</td>
<td>The energy consumption of home appliances are visible in real-time.</td>
</tr>
<tr>
<td>Advanced monitoring</td>
<td>Any device readings as well as their energy consumption from the current and past time frame can be displayed alone or combined.</td>
</tr>
</tbody>
</table>

### 3.3 Online survey design

Developers often prefer to choose a particular JS framework by practical criteria such as quality and availability of documentation. They claim to have little understanding about existing software metrics for JS frameworks [8]. Therefore based on comparison frameworks suggested by [43] and [47], an online survey was designed to gather JS frameworks differences from the developers’ point of view. Survey participants were found amongst Stackoverflow platform[^20] which contains questions and answers in computer science, using the tool Stack Exchange API[^21]. This tool extracts users profile links from Stackoverflow who answered questions with a particular keyword like “angularjs”. Next, in order to find users email addresses, their profiles have been checked

[^20]: [https://stackoverflow.com/](https://stackoverflow.com/)
[^21]: [https://api.stackexchange.com/](https://api.stackexchange.com/)
for the existence of a github\textsuperscript{22} page. If their github pages contained a public email address, it was added to participant list. In total 319 emails were collected.

The survey questions are split into 3 sections. First, years of experience in JavaScript, a JS framework they work on and familiarity about home automation was asked. The second section has questions about design patterns, the convenience of the most important features in JS frameworks and a preferred framework to create a home automation frontends. The last section asks about the software metrics importance, most essential metrics.

3.4 Summary of the research method

This chapter has described the main research methodology which comprises of mixed type of qualitative and quantitative research methods. The qualitative research method has been chosen because of JS frameworks analysis cannot be solely relied on software metrics and developers often prefer looking into other qualities of the JS frameworks. In addition, the literature review would require much effort to only cover all information about JS frameworks differences or similarities. Therefore, a Qualitative Data Analysis method has been used in the first part of the research methodology.

Following this, relying on quantitative data analysis methods, the software metrics will be investigated and presented in the form of pure numbers of correlations between JS frameworks will be identified. The online survey also will contribute to the qualitatively and numerically to the final results.

\textsuperscript{22} \url{https://github.com/}
4 IMPLEMENTATION

This chapter presents the architecture of the home automation system in which frontends codes have been tested, the process of software metrics measurement and the developed user interfaces for the pre-defined scenarios.

4.1 System architecture

The architecture for the implementation is depicted in Fig. 16. Home appliances including actuators, sensors are connected to the FHEM server. Communication to the FHEM is established via telnet connection using FHEM.js proxy server written in Node.js. On the other hand, frontends written in JS frameworks and Dart communicates to FHEM.js via Websockets connection. The data about home appliances is loaded by the frontends in the form of JSON string. Important to notice that the received JSON string also contains FHEM server configuration information and is filtered to extract a payload – a list of devices and their status, as well as possible sets and attributes of the devices in the system.

The frontend codes as a client side utilize socket.io library and there is a specific socket.io client library developed for each JS framework or Dart by their contributors. The syntax and the working principle of these libraries is the same though. The FHEM server and intermediary FHEM.js proxy server are both installed on a single-board computer Raspberry Pi. The frontends are hosted by the FHEM.js proxy itself. Other popular servers such as Apache HTTP Server or Nginx can be used to serve the frontends. However, thanks to sufficient hosting capabilities of the FHEM.js other alternatives are not used. Software metrics extraction process for AngularJS, Ember and React is done using the software measurement tool Plato.
4.2 Developed User Interfaces

The frontends were developed for three main scenarios in which the frameworks are evaluated. These user interfaces are illustrated in Fig. 17, 18, 20 and 21. All frontends written by the selected JS frameworks have the same functionality within the same scenario. In order to achieve displaying up-to-date status of home appliances two-way data binding feature is widely used. This feature is supported by many JS frameworks by default. In addition, other useful features such as routing, dynamic rendering, component-based architecture, ChartJS libraries are used in order to achieve full-featured functionality of these web-applications.

As can be seen in Fig. 17, a basic control frontend allows users to observe the real-time status of all existing home appliances. At any given time they can easily turn the devices on or off via input elements such as switches, buttons and range sliders. However, this frontend does not include any scheduling, automating or energy monitoring possibilities. In
order to include these mentioned features, a new user interface need to be added into the application where users should easily switch into it.

In a basic automation frontend depicted in Fig. 18, households can perform basic scheduling actions on home appliances allowing them to create future events. The working principle and the main stages of this scenario is the following:

- A user conveniently selects devices from a drop-down list;
- Then chooses an event that need to occur with the selected device;
- If a time for the schedule is specified, this event command is generated and saved in FHEM;
- Any saved events can be cancelled before the event happens which is displayed on the top of event composition area.

It is important to note that in this scenario a device events are composed separately following one another. Thus, complex events such as what should happen in case of some other events take place cannot be formed. In terms of programming aspect, all devices in a drop-down menu and in events menu are filled dynamically without relying on hard-coded way. Which means the devices and their actions are taken from the model received from the FHEM when the application booted.
Since the basic scheduling frontend does not meet the challenges of a full home automation application demands, a frontend with a complex rule-based automation feature must be also examined using JS frameworks. Therefore, another more advanced frontend is designed which helps to create complex home automation commands. The user interface for this advanced home automation scenario is illustrated in Fig. 19 and 20.

This frontend is written in AngularJS, Ember, React, Angular2+ versions and Dart language. The main advantage of such UI is it fully includes the rule-based home automation guidelines emplyed in [6]. From the Fig. 19 it can be seen that a user can compose many miniactions that should happen if the main event happens. In Fig. 19. a user wants to set a Lamp on, dim Bed dimmer to 50%, and Foyerlamp on IF a door is open/opens (on) at 2 AM late night as the home reaction to a potential intruder. Similarly to this, any complex event-condition-action type of events can be easily programmed by this frontend. For example, the user can reduce energy consumption by turning off the devices during work hours and when nobody is at home. In this case however, the home may also be configured in a way to set alarms, keeping security and optimal energy levels.
Configuring your home:

**Fig. 19.** Complex event configuration process based on Event-Condition Action (ECA)

**Fig. 20.** Advanced automation

**Fig. 21.** Basic monitoring
As mentioned in the thesis goals, the main objective for this thesis here is to explore how different frontends can be built by the different JS frameworks. More clearly, how the MVC pattern and framework features are supported with less efforts for UI developers by different frameworks.

As it has been discussed in home automation frontends design section, the tasks users want to achieve must be convenient and intuitive. Therefore an important functionality is added into basic automation and advanced rule-based automation frontends. This feature is providing only necessary and essential commands for home appliances. For example a simple remote switch of HomeMatic vendor contains the following possible sets – commands can be send:


Since, the selection of a particular home device from a dropdown list opens all possible commands for this specific device in the next selection section, some devices can have overwhelmingly large and complex commands to be chosen. Therefore, this vast number of visually not intuitive and misleading commands can sharply decrease UX and mislead a user from choosing the right task he/she wants to achieve. Therefore, the similar operation – providing only necessary commands out of the large PossibleSets is done on the UIs in all JS frameworks. This operation is simply filtering out device Possiblesets and populating the options with hard coded fewer ones which exactly match the selected device Possiblesets. The Fig. 22 describes the initial problem for our frontends.
The following scenario is intended to explore how the charting is done by the JS frameworks. For this case a popular charting library – ChartJS was utilized to display energy consumption by different home appliances in real-time. Again this scenario is based on MVC software design pattern where the View is bound to the Model and displays any changes that take place in the Model instantly. The Controller is responsible for keeping up-to-date the Model and updating it when new readings from the devices arrive in the application. The UI design for this scenario is displayed in Fig. 21. The descriptive sequence diagrams for the scenarios are provided in Fig. 23 and 24.
Fig. 23. Basic control scenario sequence diagram

Fig. 24. Basic and Advanced rule-based automation scenarios sequence diagram
4.3 **Software metrics evaluation**

After the user interface prototypes for each scenario have been written and tested in real-life environment using FHEM platform, the source code software metrics were extracted using the tool – Plato. From the set of tools identified in the literature the following ones – Jsmeter, JSHint and JSLint does not identify AngularJS and Ember frameworks code syntax and throw error. The reason is these tools can only extract software metrics from the standard JavaScript code. Therefore, this stage of the experiments ended up using the tool Plato. Plato tool does not measure metrics from Dart source code neither and therefore, Dart source code metrics are extracted manually via code observation and using the corresponding software metric formulas.

After the installation of the Plato with the command `npm install –g plato`, the source codes located in directories can be evaluated using tool-specific commands from the terminal.

![Image of Command Prompt window](image.png)

**Fig. 25.** Measuring the software metrics via the tool Plato
4.4 Summary of the implementation

As can be seen from this chapter, the whole platform for performing the experiments and frameworks evaluation is consist of a single board computer Raspberry Pi, home appliances, few additional composite software elements such as intermediary fhem.js proxy server, FHEM home automation server and the tools to extract the software metrics from the UI source codes. The user interface scenarios have been created in order to evaluate the complexity changes of the different frameworks in various use cases. The metrics evaluation started from the minimal user requirements in order to interact with home. Since, all JS frameworks are written in the same programming language, the differences can not be adequate in a small software projects. Therefore, the complexity, size and the functionality of the user interfaces range from basic controlling ability into more complex HAS with end-user programming possibility. The end-users do not have to write any programing scripts to configure their home though. For this case the thesis employs the event-condition-action (ECA) mechanism to allow complex rule-setting for the home. Finally, taking into account all these UI scenarios, obtained software metrics results can indicate which JS framework or Dart brings on higher complexity for the UI developers.
5 RESULTS

The results from the qualitative data analysis, the MVC design pattern support by the JS frameworks and Dart, software maintainability metrics results and finally the outcomes from the online survey is presented in this chapter. Because of the absence of code metric analyzer tools for Dart language amongst the developer community, only the lines of code is provided in the results.

5.1 Qualitative data analysis results

The qualitative data analysis is focused on the differences of JS frameworks and Dart that can affect the project maintainability. These features comprise the internal and external software attributes such as community support and learning curve.

5.1.1 Community

Community is seen as an essential factor for a framework choice. This can be explained that larger community implies more support by other developers, larger number of questions and answers available and higher number of tutorials.

The data on the Fig. 26 from June, 2018 shows the number of questions tagged on the Stackeoverflow and the number of repositories on the GitHub platform. It clearly shows that Angular frameworks have a larger community involvement with the highest proportion of questions and the repositories on the GitHub. Surprisingly, React framework has the highest number of GitHub repositories which exceeds any other JS framework several times. However, it keeps around as many questions as Angular framework. Ember framework and newcomer – Dart have fairly big community as well with several thousands questions and GitHub repositories although they are in much lower popularity and community involvement.

These metrics however, show only the current state of the frameworks on a particular date and these figures may change over time.
5.1.2 Other attributes

There are other attributes developers look at when choosing a particular framework to develop a frontend application. For example, the maturity of a framework, learning curve, documentation, and others. These comparisons are presented in Table 6.

**Table 6. The primary differences of JS frameworks**

<table>
<thead>
<tr>
<th></th>
<th>AngularJS</th>
<th>Angular2</th>
<th>React</th>
<th>Ember</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>MVW framework</td>
<td>MVC framework</td>
<td>JavaScript library</td>
<td>MVC framework</td>
</tr>
<tr>
<td>1st release</td>
<td>2009</td>
<td>2016</td>
<td>2013</td>
<td>2011</td>
</tr>
<tr>
<td>Founded by</td>
<td>Google</td>
<td>Google</td>
<td>Facebook</td>
<td>Yehuda Katz</td>
</tr>
<tr>
<td>Licenses</td>
<td>MIT</td>
<td>MIT</td>
<td>MIT</td>
<td>BSD-3-Clause</td>
</tr>
<tr>
<td>Relative popularity</td>
<td>***</td>
<td>***</td>
<td>*****</td>
<td>*</td>
</tr>
<tr>
<td>Learning curve</td>
<td>Steep</td>
<td>Steep</td>
<td>Moderate</td>
<td>Steep</td>
</tr>
<tr>
<td>Support</td>
<td>*****</td>
<td>****</td>
<td>****</td>
<td>**</td>
</tr>
<tr>
<td>Community</td>
<td>Highest</td>
<td>High</td>
<td>Very popular</td>
<td>Average</td>
</tr>
<tr>
<td>Documentation</td>
<td>AngularJS</td>
<td>Angular2</td>
<td>React</td>
<td>Ember</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>----------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>Server-side rendering</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Popular websites that use this framework</td>
<td>Youtube, Vevo, Freelancer, Istockphoto, Weather, SkyStore</td>
<td>The Guardian, PayPal, Netflix, Hypertrack</td>
<td>Facebook, Instagram, Khan Academy, New York Times, Airbnb, Flipkart, Sony Lifeolog</td>
<td>Apple Music, Yahoo!, LinkedIn, TinderBox, Netflix, Groupon</td>
</tr>
<tr>
<td>Ideal for</td>
<td>Building highly complex and active web applications</td>
<td>Building highly complex and active web applications</td>
<td>Building complex application where data is changed very frequently</td>
<td>Dynamic SPAs</td>
</tr>
<tr>
<td>Data binding</td>
<td>Two-way</td>
<td>Two-way</td>
<td>One-way</td>
<td>Two-way</td>
</tr>
<tr>
<td>Routing</td>
<td>Yes (Limited functionality)</td>
<td>Yes</td>
<td>No (third party needed)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 5.2 Software metrics results

After the developing and testing the user interfaces on the FHEM platform. The source codes were analyzed using the tool Plato. The most important software metrics regarding to software complexity, size and maintainability have been obtained.

The result from the software metrics comparison for the basic control scenario is given in Fig. 27. It can be seen React code has as twice larger lines of code as opposed to AngularJS, Angular2 and Ember frameworks. They had similarly less lines of code. When maintainability and Halstead bugs were almost identical, the difficulty and Lint errors showed higher results for React again.

In the second scenario a new feature is added to the existing user interface code – Basic automation. The difference in lines of codes was nearly three times higher in React compared to AngularJS and almost twice than in Ember as it is shown in Fig. 28. Whereas Maintainability index, Cyclomatic Complexity and Halstead bugs were visually similar in all frameworks. Identically to Basic Automation, in the Advanced Automation scenario
where complex commands can be formed, the metrics were symmetrically similar. Where again React outperformed others with the program size. However, the lines of codes in other frameworks increased as well due to adding the new functionality Fig. 29.

**Fig. 27.** The software metrics from the Basic Control scenario

**Fig. 28.** The software metrics from the Basic Automation scenario
A closer analysis of the metrics change within the scenarios reveals clearer understandings of the frameworks differences and their impact on the software metrics. For example from the literature review it is apparent that the AngularJS framework provides shortest code for developing web applications. React on the other hand, is the component based framework which means that even single input elements can be written as a component and have a complex behavior. Components in React may also have their own state, maintaining which increases the application size significantly. This can be observed from the Fig. 30.

As the application size – lines of codes increases it becomes more difficult to make changes to it, i.e. to maintain. Therefore, it is assumed that the maintainability decreases. The diagram in Fig. 31. illustrates the overall trend in which the frontend source code maintainability reduces. Although this trend shows small shifts in the Maintainability index, the future modifications to the software, especially increases in the size, can be speculated by the developers and assist making a less complex framework choice.

**Fig. 29.** The software metrics from the Advanced Automation scenario
Fig. 30. The change of the Lines of codes by scenarios

Fig. 31. The change of the Maintainability Index by scenarios
Lastly, we know that the Cyclomatic complexity, as defined in [34] is the number of linearly dependent paths in a source code. The frontend codes in this work have identical functionality and therefore the flow in the source code is expected to be similar as well.

The comparison can be observed from the Fig. 32. It shows that the Cyclomatic Complexity in Ember and React closely match each other, although AngularJS shows deviations through the modifications. This can be explained with the following fact. In the Advanced Automation scenario in AngularJS new sub-tasks are generated using the “Add Action” button which adds new fields into the UI. These uniqueness of these fields are kept and maintained in separate and dynamically created arrays. Thus, having small extra operations on these arrays might have caused the Cyclomatic Complexity to be higher compared to React and Ember. All in all, these figures are still correlating and following the same rising trends Fig. 32.

Fig. 32. The change of the Cyclomatic complexity by scenarios

5.3 MVC design pattern support

In AngularJS, the MVC pattern support is straightforward and the Model is defined with a $scope variable and any data, functions represent the Model. An application Controller is a holder of the model. The View is represented by the HTML template. The functionality of
the View is extended using so called Directives. Directives help to bind the View with the Model in a convenient way. For example the directive ng-repeat iterates over the Model elements and display it on the View. Any changes in the Model reflect automatically without writing an extra code. Moreover, when a Routing feature is used in AngularJS, it shares the same data on any pages and Routes do not necessitate loading the data again.

Angular2 framework is utmost similar to AngularJS except its component based architecture and in fact, being a different programming language – TypeScript. In addition, Angular2 extensively relies on extending the HTML with using directives. A developer-designed directives can also be used to improve UX. However, Angular2 crisply reduced the number of built-in directives. For example, NgIf, NgFor and also NgSwitch are the main remaining built-in directives in this framework. As pertains to Routing feature, any data can be shared between different Routes via using Services.

The model name enclosed in double curly braces in HTML document will be shown whenever a model has a change. Fig. 33 shows how a model stored with a name — “device” can be displayed in AngularJS View. Angular2+ versions, React and Ember also provide the same feature. In this study the model represents a collection of datasets from devices. Most importantly, all JavaScript frameworks allow iteration over elements of a model that stores information about multiple items. This feature can also create any time of user input elements such as buttons, switch elements and range sliders repeatedly in a loop. The switch button element in the last 3 lines of code in Fig. 33 is created multiple times for each device in a model – “allDevices” from line 1. However, due to a different View logic in Dart, displaying current home devices status is achieved via programmatically removing the displayed status and adding updated ones in the Dart code. Moreover, the HTML view in of the Dart does not provide repeatedly performing operations. For example, in Fig. 33 for each device in the model – “allDevices” a new switch element is being created and added into a “statusList” container of an HTML page. Most of the View logic is defined in the Dart code unlike JS frameworks where complex displaying operations can be performed in HTML templates. However, AngularDart framework achieves the same functionality as Angular2 framework. And all features existing in Angular2 are fully available in AngularDart framework.
React on the other hand is regarded as the View layer of the MVC only. Thus, it requires using other libraries such as Redux and FLUX for Model management. In this work only React library itself was investigated without considering Redux library. In order to display the home appliances and achieve the same functionality as other frameworks, the data obtained from the Websockets is stored in an array. Updating this array when a change occur is used as the MVC pattern. Most importantly, React components communicate with each other and share the data easily and conveniently with the use of “Props”. React applications also does not use HTML template separately, instead it uses JSX – JavaScript XML. For some developers that is the main complexity for using React. This is another reason for the soared application size in React. Other frameworks retain their own templates with the extension HTML – in Angular frameworks and HBS – in Ember.

Ember framework fully supports MVC pattern although its Model need to be obtained using Adapters which retrieve the data from the external sources and store as Records. The Model simply fetches the Records and provide them as the application Model. The existing Ember Adapters are built on JSON API and their rely on GET requests to the resource.
Which means that the backend need to support this type of requests. In this study the NodeJS based proxy communicates with our UI applications via Websockets only, and therefore it does not support HTTP methods. Which means that for the Ember frontends, this work used Websockets to receive a data from the FHEM server and store it in an array. This array data then used as the application Model.

5.4 Online survey results

The results from the online survey is presented here. In this survey, the total 565 participant emails to whom the survey was sent, 15 were not reachable due to an error of not existing email. Only 30 developers filled the survey by end August 2018, which comprises around 5.4 percent of the emails who received the survey successfully.

All developers have 3 and more years of experience in JavaScript and they consider Maintainability Index, Cyclomatic Complexity, Lines of Codes and community as the most important metrics to use as shown in Fig. 35. Slightly over half of them – 53.3% stated that they would or might use software metrics to select a JS framework, whereas only around 13.3% of them responded firmly yes. Moreover, the majority of respondents mentioned that it is hard to make useful comparisons using software metrics and it could help only in specific use cases. This again shows the mild interest in using only software metrics to select a framework amongst developers. On the other hand, programmers who prefer using software metrics are more interested in Maintainability index, Cyclomatic Complexity, Lines of Codes and Community support. They also commented that the metrics may help to the decision making process but can not be fully relied on.

As mentioned earlier, JavaScript is a classless type of Object oriented programming language and does not fully support OOP approach of programming. However JavaScript allows using objects even without classes. The chart in Fig. 36. shows the efforts that might be needed to write a code with object-oriented approach in a particular JS framework or Dart. It can be seen that Angular2, Dart and Ember could be easier for implementing OOP paradigm. In fact, Angular2 is based on TypeScript language while Dart is truly object oriented programming language. Ember on the other hand, supports
creating classes and objects via its **Ember.Object** package. The benefits of using OOP paradigm can be useful for defining home appliances as independent objects with their own methods as their specific commands. However, in case of this thesis, the data received from the FHEM is converted into JSON objects and stored as Model without creating JavaScript objects. In other cases, for example in different home automation platforms deploying OOP approach could be useful to define home appliances.

**Fig. 35.** Most important software metrics for developers

**Fig. 36.** Support of the Object Oriented Programming by the frameworks
In terms of other highly essential features of JSFs and Dart, we can make conclusions from the responses in Fig. 37. about how well the specific feature can be implemented by the frameworks. It can be seen that the Data binding feature has collected nearly equal votes for all JS frameworks except Dart language. Indeed, as the experiments from the scenarios show, data binding is supported equally well by the JSFs. Only in React few extra lines of codes is needed to bind a View to the Model data. As for Routing and Services, Ember and Angular2 frameworks outperformed others. Interestingly, Ember is claimed to have a strong Routing feature by many online JSf comparison blogs.

For achieving high UX and UI design, it is important to deploy design libraries like MaterialUI. According to the developers who already have experience over 3 years, React and Angular2 have the best design libraries support compared to other frameworks. Another feature – using Component-Template architecture, far better frameworks are Ember, Angular2 and React for the JS programmers.

The Community support and Conditional rendering variances were similar, although Community Support for Dart has recorded the minimal endorsement. The largest share in the figure by the React again confirms its biggest pool of community. Angular2, React also have much better dependencies which is probably due to their high popularity today.

![Fig. 37. Support of different features by the frameworks](image-url)
Lastly, the online survey has also asked developers comments on the reasons to choose a particular JS framework for building home automation user interface and the most difficult aspects of these JSFs. As it is presented in Table 7 and Table 8, not all developers have commented to these questions. Only few of them fully completed their responses. Although these tables contain small amount of information, these are fully credible comparison from the experienced programmers.

**Table 7.** The reasons why developers would choose this framework for HAS UI

<table>
<thead>
<tr>
<th>JS framework/Dart</th>
<th>Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>AngularJS</td>
<td>Lightweight and not opinionated;</td>
</tr>
<tr>
<td>Ember</td>
<td>Has a solid and complete toolset; ease of setup; standard tooling and ecosystem; good for scalability; conventions over configuration and routing focused. Easy to keep updated;</td>
</tr>
<tr>
<td>Dart</td>
<td>AngularDart is highly productive and provides a type system; because of AngularDart framework of Dart;</td>
</tr>
<tr>
<td>Angular2</td>
<td>Provides far more features as standard; cross platform support of Angular2;</td>
</tr>
<tr>
<td>React</td>
<td>Easier to learn; Not based on DOM; because of React native framework;</td>
</tr>
</tbody>
</table>

**Table 8.** The hardest features of JSFs according to developers

<table>
<thead>
<tr>
<th>JS framework/Dart</th>
<th>Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>AngularJS</td>
<td>Communication between controllers; scopes; state management; server side rendering;</td>
</tr>
<tr>
<td>Ember</td>
<td>Dependencies; obscure debugging; state management; server side rendering;</td>
</tr>
<tr>
<td>Dart</td>
<td>AngularDart Requires importing even common directives like NgIf</td>
</tr>
<tr>
<td>Angular2</td>
<td>Learning difficulty; RxJS library and streams; state management; server side rendering;</td>
</tr>
<tr>
<td>React</td>
<td>Dependency on many libraries; state management; server side rendering; mixture of JS, HTML and CSS;</td>
</tr>
</tbody>
</table>
5.5 Summary of the results

This chapter presented the results from the qualitative data analysis, software metrics as well as the feedbacks from the online survey. The experiments on the frameworks have been done on three main user interfaces scenarios. Each of the following UI scenario incorporates the functionalities of the previous one. Following this approach we can observe how the code implementation in a particular framework changes. The metrics affecting the developer efforts, coding difficulty, source code application complexity have been analyzed. The outcomes show valuable differential information and are giving insight on the frameworks’ capabilities for implementing home automation UIs.

However, as the previous studies indicate, the maintainability of a software cannot be determined using the software metrics only. Other external factors such as documentation, community size and support, the availability of the already existing applications or projects and personal factor must also be examined. Therefore, the qualitative data is also supplied along with metrics. This chapter presents the above highlighted factors in an organized manner.

Lastly, the opinions of "real" developers matter. In other words, JS developers feedbacks on this JS framework comparison area is also involved in the thesis results. The online survey shows the reliable information on which HAS UI developers can rely. In this work, this data affirms the importance of metrics for the developers for choosing a JSf. Moreover, code readability, documentation quality and other important features was obtained from the survey. All these results will be closely discussed in the next chapter.
6 ANALYSIS AND DISCUSSION

This research demonstrated the essence of JS frameworks comparison in the domain of home automation. Although JS family frameworks share more similarities than distinctions, noticeable contrasts come into place when we deploy them in a specific field under certain requirements.

It has been shown that a community behind a framework may determine the success of the framework popularity and adoption for projects by developers. The number of existing projects and answered questions implies better support and easier maintainability of frameworks. Therefore, we can see that React framework has by far larger community support than any other framework or Dart. After React, AngularJS has also the second largest number of projects on Github platform. Lastly, AngularJS shares the largest question-answer ratio and Angular with React are the second most questioned frameworks.

The experiments show that AngularJS framework can provide shortest size in terms of Lines of codes for building the same HAS frontends than other frameworks. This allows developers to complete HAS web-applications much faster. For the same application, Ember and Angular2 bring about more lines of code therefore the larger is the size of the code and higher complexity. However, it is important to note that Angular is maintaining its component based architecture with equal UX while with less code than React. HTML template extension is supported well likewise the previous AngularJS framework. This is also taking much of the complexity away from the JS source code.

The Dart language framework – AngularDart was identical to Angular2 in terms of software metrics. Finally, the size of React projects showed quick expansion with the advancement of application functionalities and the largest lines of code than any other framework. This can lead to declined maintainability as well reducing bug fixing and modification efforts.

The maintainability index declines with the increase of the lines of codes in an application. In fact, JS frameworks also presented this trend in the scenarios. However, Angular2 and
React frameworks showed quite better results in Maintainability index. With the advancement of the application the maintainability index has even improved. AngularJS and Ember however, had lower indications.

As for MVC design pattern, all frameworks can support this pattern well as the results show. However, in React framework the essential element of MVC pattern – two way data binding requires updating application state to display the updated Model on the View by writing extra code or by additional libraries. In larger applications this could add up in the overall code size. AngularJS, Ember and Angular2+ frameworks support this feature without extra piece of code therefore, MVC pattern is supported easier.

Although Dart language has not evolved usable code analysis tools, its loose coupling with an HTML template decreases the risk of erroneous mistakes for developers. On the other hand, Dart cannot compete with AngularJS, Ember or similar JSf in data binding which has negative effects on BAS frontend where a real-time device status displaying is essential. However, the optimal solution to use Dart for HAS UI could be using its AngularDart framework which is identical to Angular2 JavaScript framework. The only difference is the programming language – Dart.

Even the feedbacks on design patterns were dispersed and biased amongst developers, their responses on software metrics help choose a particular JS framework for home automation frontend developers. Because the results of the survey showed the importance of Maintainability index and Lines of codes software metrics. Therefore, the metric results would assist reducing user interface creation costs and efforts for programmers through a higher awareness about JS frameworks. As a result, home automation systems would have better frontends written with less effort that may make them popular.

The online survey also shows that Angular2, Ember and Dart are the most suitable frameworks for Object Oriented programming paradigm. With AngularJS and React, this OOP approach is harder according to developers. Ember and Angular2 were found to be the most proper frameworks for implementing Routing feature. These two frameworks – Angular2, Ember and React are also most optimal for component-template architecture.
Lastly, for making UIs with better visually attractive design, Angular2 and React are preferred by the developers.

### 6.1 Sustainability analysis

Although the sustainability benefits from this work cannot be expressed in kgs of CO2 reduced a year, it can contribute the technical aspect of the sustainability via software sustainability. This work showed the possibilities of developing a HAS UI by different JS frameworks in detail. More importantly, this thesis evaluated the different JS frameworks on the same home automation UI and how feasible it is for these JSFs.

As discussed by C.U.Becker in [53], the science and technology are the influencing factors on the sustainability along with the economy. As these dimensions are under proximate inter-relations, the improvements in one of them will impact the sustainability as well. Therefore, it can be stressed that the sustainability from technical contribution is imminent. In addition, it has been highlighted in every and each study regarding to home automation systems that the main obstacle for HAS adoption is their complex frontends. Yet with the countless number of JS frameworks existing and increasing by time, another issue arises which is using a very JS framework which provides lower efforts to develop a UI. The problem is obvious – assisting JS developers with a meaningful documentation which could aid them reducing programming efforts to produce "good" and convenient UIs. This could also prevent wasting manpower, for example, with using a JS framework which has a poorest documentation or requires complex programming.

Assuming this multi-dimensional factors, the thesis has carried out investigations in home automation user interface best practises, the meaningful and useful software metrics, external factors that affect software maintainability and lastly survey with JS developers. All these results are summarized in the results chapter.

The sustainability impacts from this thesis work is presented in Fig. 38. From this figure we can see that the main contribution from this thesis is to the developer community in home automation field. By providing to them this documentation, the developers efforts to
build a HAS frontends could be reduced which has following effects on other aspects of the sustainability. One of them is economic impacts, as it has been discussed in the beginning of this work, most developer efforts is spent on making changes to a software. And making changes to already built software is considered to be costly activity. Therefore, choosing the right JS framework from the beginning in case of HAS is important. As it prevents developers from using difficult and less maintainable framework. Other enabling and structural effects are popularity of home automation applications, improved comfort and definitely, reduced energy consumption due to more popular HAS.

![Sustainability analysis of the research](image)

**Fig. 38.** Sustainability analysis of the research
Admittedly, the thesis has also its prominent limitations. These limitations might be utilizing the only home automation system technology – FHEM which comes with its own pros and cons. So this thesis used Websockets connection alone not involving popular HTTP methods on the resource. Another limitation could be limited number of user interface types. Some end-users might want having a fancy UI with more interactive functionalities instead of easy configuring the home appliances. For example, animations and little graphic content analysis on JS frameworks would be a big advantage for this thesis work.

In the end, any software developer will definitely rely on his or her own programming experience in the first place. However, availability of a such practical documentation in hand can undoubtedly guide them towards more sustainable and programmer friendly JS frameworks. Eventually, this would be the beginning for more popular home automation systems with JS-based UI which leads to less consumed energy and finally, to a greener world.
7 CONCLUSION AND FUTURE WORK

In this thesis, a comparison research on JavaScript frameworks and Dart language for creating user interfaces in home automation is presented. In the beginning of the study, a comprehensive analysis has been carried out in home automation user interface usability aspects. The reason is to meet the requirements users often claim about – the complex design and low usability of HAS frontends. Taking into account these considerations in order to contribute to the increased acceptance of the user interfaces, the thesis reviewed the main factors that influence software maintainability.

The main challenge of the thesis is the unique similarities of all JavaScript frameworks for the first sight. However, the Qualitative Data Analysis and the online survey identified the main distinctions of these JSFs and the developers` main preferences to differentiate them. In the next step software metrics that are highly important for determining any software project maintainability and difficulty have been identified. Only few software metrics have been selected as possible sets to apply them to JS source code. Using metric extraction tools, the variations of them in different frameworks have been compared.

For the moment, the research shows that the comparison of different tools in the home automation domain is important and can reveal the weak sides of JS frameworks. The immediate impact of the study in energy consumption and environment may not be noticeable. However, this thesis is a starting point in comparing very similar technologies in a specific domain – home automation technologies and can already assist developers. Therefore, having immediate positive technical impacts, this work can inevitably promote home automation systems and bring about significant enabling effects to energy savings and to the environmental benefits afterwards.

As a future work, other HAS can be integrated into frontends written in JS frameworks. This will contribute to a more advanced comparison of JSf possibilities and limitations in the field. In addition, animations and multimedia support by the JS frameworks would also be an interesting work. This kind of study would help JS frameworks competing android or iOS applications and cause a higher interest for users to interact with their houses.
REFERENCES


12. Floorplan extension of FHEM. Documented online at https://wiki.fhem.de/wiki/FLOORPLAN


17. Hjelm, S., Andersson, J., Evaluation of JavaScript frameworks, Linnaeus University, Faculty of Technology, Thesis project, 2015., Available at: https://pdfs.semanticscholar.org/1212/8e74fc7b1f9eed482b6372224f456932ef48.pdf.


20. AngularJS official website, retrieved in April 2018 from https://angularjs.org/


29. IEEE Standard Glossary of Software Engineering Terminology, the Institute of Electrical and Electronics Engineers, Inc. New York, USA. Retrieved in May, 2018 from: 


53. Becker, C., Sustainability Ethics and Sustainability Research, LinkSpringer
APPENDIX 1.
Advantages and disadvantages of JavaScript frameworks from the Qualitative Data Analysis method

The frontend source codes are available on: [https://github.com/Mukhammadjon-Jalolov](https://github.com/Mukhammadjon-Jalolov)

<table>
<thead>
<tr>
<th>AngularJS</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Abundance of the training resources.</td>
<td>Declining popularity.</td>
</tr>
<tr>
<td></td>
<td>Good documentation.</td>
<td>Complexity of the directives API.</td>
</tr>
<tr>
<td></td>
<td>Very well established.</td>
<td>For pages with many interactive elements, Angular becomes slow.</td>
</tr>
<tr>
<td></td>
<td>Full featured.</td>
<td>Original design tends to be slow.</td>
</tr>
<tr>
<td></td>
<td>Creation of customized Document Object Model (DOM) elements.</td>
<td>Performance might be an issue due to many DOM elements.</td>
</tr>
<tr>
<td></td>
<td><strong>“Dirty checking” feature.</strong></td>
<td>Complex third-party integration.</td>
</tr>
<tr>
<td></td>
<td>Angular scans for each single rendered element on the page for any changes. A key feature for two-way data binding.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Dependency injection.</strong></td>
<td>Steep learning curve.</td>
</tr>
<tr>
<td></td>
<td>Angular has a built-in dependency injection feature which eases application development and testing. Using this feature a dependency can be injected easily to an application.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Simple routing.</td>
<td>Router is limited.</td>
</tr>
<tr>
<td></td>
<td>Easy-to-test code. Every piece of the application is easy to test.</td>
<td>Easy-to-use Scopes, however they are hard to debug.</td>
</tr>
<tr>
<td></td>
<td>The framework facilitates extension</td>
<td>Not pure MVC framework, rather</td>
</tr>
<tr>
<td>Advantage/Issue</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>of HTML syntax and creates reusable components by directives.</td>
<td>it uses MVVM – Model-View-ViewModel pattern.</td>
<td></td>
</tr>
<tr>
<td>Robust template-building solutions. Uses binding expressions in HTML attributes which maintains templating functionality.</td>
<td>Low performance for pages with lots of interactive elements.</td>
<td></td>
</tr>
<tr>
<td><strong>Data modeling</strong> is limited to the use of small data models in order to make the code simple and easy to test.</td>
<td>Problems with indexability by search engines when it comes to single page apps</td>
<td></td>
</tr>
<tr>
<td><strong>Fast</strong> when rendering static lists.</td>
<td>Complicated third-party integration</td>
<td></td>
</tr>
<tr>
<td>Great with code reuse (Angular libraries)</td>
<td>Troubles with debugging</td>
<td></td>
</tr>
<tr>
<td><strong>Cross-Browser Compliant</strong></td>
<td>Applications written in Angular framework are highly interoperable in different browsers. This is achieved by automatic code handling for various browsers.</td>
<td></td>
</tr>
<tr>
<td><strong>Directives</strong></td>
<td>Possibility to create custom HTML tags to use them as reusable components.</td>
<td></td>
</tr>
<tr>
<td>Quick code production.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large community that is ready to assist in using the framework.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### APPENDIX 1. (continues)

<table>
<thead>
<tr>
<th>Angular2</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Two-way data binding</strong></td>
<td>Faster and easier data binding with the minimum developer effort. A key feature for MVC pattern.</td>
<td><strong>Divided community</strong></td>
</tr>
<tr>
<td><strong>DOM manipulation</strong></td>
<td>Two-way data binding feature releases developers from manual manipulation of the Document Object Model (DOM). Thus, saves programming time and development efforts.</td>
<td><strong>Updates</strong></td>
</tr>
<tr>
<td>Full-featured framework with well-tested defaults.</td>
<td><strong>Initial load time</strong></td>
<td>Relatively slower page rendering especially in case of complex Angular applications and using older hardware.</td>
</tr>
<tr>
<td>TypeScript is easier to adopt for developers who has experience in object-oriented programming.</td>
<td><strong>Steep learning curve</strong></td>
<td>Newcomers to Angular framework encounter huge difficulty to acclimatize into it. This is also caused by the finite amount of the official documentation.</td>
</tr>
<tr>
<td>Strong corporate support (Google).</td>
<td>TypeScript can be an obstacle to learn for beginners.</td>
<td></td>
</tr>
<tr>
<td>Clear best practices.</td>
<td>Poor start up metrics in benchmarks.</td>
<td></td>
</tr>
</tbody>
</table>

(continues)
APPENDIX 1. (continues)

<table>
<thead>
<tr>
<th>Ember</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Convention over configuration.</strong></td>
<td>Ember helps with its self-configuration. For example in case of generating routes Ember automatically generates the code instead of requiring configurations for it.</td>
<td><strong>API Querying</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ember Data is convenient for using with standard REST API. However, for API’s that does not match to REST standards or that return data in a different way, developers have to write own AJAX requests.</td>
</tr>
<tr>
<td><strong>Ember addons</strong></td>
<td>Ember has a centralized repository for all addons. Any addons can be easily installed with only one terminal command – “ember install” (<a href="http://emberaddons.com">http://emberaddons.com</a>).</td>
<td><strong>Ported add-ons</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Many addons in ember are ports of jQuery libraries. The only minority of them are written from scratch.</td>
</tr>
<tr>
<td><strong>Built-in testing tools</strong></td>
<td>Ember has a built-in QUnit testing tool. Other tools can be utilized as well such as Mocha or Chai.</td>
<td><strong>jQuery dependency</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The tight coupling with jQuery doesn’t go down with some developers. Many prefer using axios or standard fetch for making ajax requests.</td>
</tr>
<tr>
<td><strong>Active community and support</strong></td>
<td>Ember has a strong and constant community.</td>
<td><strong>Very steep learning curve</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Good knowledge of JavaScript is very important to master Ember.</td>
</tr>
<tr>
<td></td>
<td>Clear best practices.</td>
<td><strong>Slow updates</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ember maintenance team is slow. The last versions have only minor changes.</td>
</tr>
<tr>
<td></td>
<td>Full featured.</td>
<td><strong>Declining popularity.</strong></td>
</tr>
<tr>
<td><strong>Ember Data library</strong></td>
<td>This is a very convenient library for querying API layer and getting data quickly.</td>
<td><strong>Large size.</strong></td>
</tr>
<tr>
<td><strong>Client-side rendering</strong></td>
<td>and structure to scalable web applications beyond the view layer.</td>
<td><strong>No server-side rendering</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ember does not support server side rendering.</td>
</tr>
<tr>
<td><strong>URL-support.</strong></td>
<td>Ember lacks the reuse of components at Controller level.</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Ember’s object model facilitates <strong>Key-Value Observation.</strong></td>
<td>Ember documentation has much outdated content and examples that are not usable.</td>
<td></td>
</tr>
<tr>
<td><strong>Nested UIs.</strong></td>
<td>Handlebars.js pollutes the DOM with many <code>&lt;script&gt;</code> tags, which it uses as markers to keep the templates up to date with your model.</td>
<td></td>
</tr>
<tr>
<td>High performance. Ember can operate well in massive application ecosystems.</td>
<td>Cumbersome when going outside its typical uses.</td>
<td></td>
</tr>
<tr>
<td><strong>Fully-fledged templating mechanism</strong></td>
<td>Ember’s object model implementation bloats Ember’s overall size and call stack while debugging.</td>
<td></td>
</tr>
<tr>
<td>Handlebars templating feature reduces the amount of code.</td>
<td>Ember’s data layer has a very good integration with Java.</td>
<td></td>
</tr>
<tr>
<td>Ember’s data layer has a very good integration with Java.</td>
<td>Ember is considered as the most opinionated and heavy framework.</td>
<td></td>
</tr>
<tr>
<td>Uses <strong>observers to change values</strong>, which results in only rendering the values being changed.</td>
<td>Ember is too heavy for small projects.</td>
<td></td>
</tr>
<tr>
<td><strong>Avoids “dirty checking”</strong> by using accessories.</td>
<td>Testing story seems vague/incomplete.</td>
<td></td>
</tr>
<tr>
<td>Faster boot times and inherent stability.</td>
<td>Difficulties for meeting quick changes.</td>
<td></td>
</tr>
<tr>
<td>Friendly docs and API.</td>
<td>Relatively small community and many web tutorials are outdated.</td>
<td></td>
</tr>
<tr>
<td>Two-way data binding.</td>
<td>(continues)</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX 1. (continues)

<table>
<thead>
<tr>
<th>React</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSX</td>
<td>JSX is JavaScript XML. JSX is extension syntax for the JavaScript language.</td>
<td>Poor documentation</td>
</tr>
<tr>
<td></td>
<td>JSX is compiled to JavaScript objects. Visually looks like HTML and JavaScript mixed string. JSX can be used to render subcomponents.</td>
<td>Although React is highly popular, it has moderate documentation.</td>
</tr>
<tr>
<td>Virtual DOM</td>
<td>A virtual DOM is a node tree of the UI elements similar to the actual DOM.</td>
<td>Not MVC</td>
</tr>
<tr>
<td></td>
<td>This virtual DOM and a real DOM are kept synchronized. Any change in a model is first reflected in the virtual DOM and then the real DOM is updated accordingly. Due to Virtual DOM React has better application performance than other JS frameworks.</td>
<td>React is responsible for the View layer of the MVC pattern only. For manipulating a model and state other libraries need to be used such as Redux.</td>
</tr>
<tr>
<td>Testability</td>
<td>React allows developers to use the views functions of the state (a state is an object which determines how the component will render and behave). Hence the developer can easily manipulate with state of the components passed to the React view. The developer can also monitor the output and triggered actions, events, functions, etc. Using this method makes it very simple to test and debug ReactJS apps.</td>
<td>Abundance of choice can be overwhelming at first.</td>
</tr>
<tr>
<td>Server-Side Rendering(SSR)</td>
<td>React supports server-side rendering when an HTML page is received from a server. A browser then renders the page before the JavaScript is loaded. SSR allows building isomorphic, universal applications.</td>
<td>Unclear best practices for newcomers.</td>
</tr>
<tr>
<td><strong>One-Way Data Binding</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The data is streamed in one direction, providing better control of the data flow for a developer.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity and learning can be steep for creating larger applications.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Highly popular and has a strong job market.</th>
</tr>
</thead>
<tbody>
<tr>
<td>It’s not a full framework it’s a library for the View layer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lots of training resources and third-party libraries to help accelerate development.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very complex view layer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Versatility. React is an ideal option for cross-platform development (web, mobile, desktop and other devices).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flux architectures are a different paradigm that what developers are used to.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strong corporate support (Facebook).</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSX</td>
</tr>
<tr>
<td>JSX can be unfamiliar for developers and it has steep learning curve.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Easy-to-learn API and interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficult for beginners. React has a steep learning curve.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Faster updates.</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>React creates a new virtual DOM and a patching mechanism with the most recent data and efficiently compares it against the previous version, creating a minimal list of update portions to be made to the real DOM to bring it in sync, rather than having to re-render the entire site on each change.</td>
</tr>
<tr>
<td>Troublesome configuration for integrating React to other frameworks or third party tools.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Easy import of components</strong> even though it has little dependencies.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Good code reuse.</strong></th>
</tr>
</thead>
</table>

| Great for JavaScript debugging. |
Can be used with other frameworks.

It is absolutely possible for example to augment Angular with React to improve performance of arduous components.

<table>
<thead>
<tr>
<th>Component based architecture.</th>
</tr>
</thead>
<tbody>
<tr>
<td>React Native Library.</td>
</tr>
<tr>
<td>Efficient rendering of massive data sets.</td>
</tr>
<tr>
<td>Readable component and data patterns.</td>
</tr>
</tbody>
</table>