The Creation of a Guide on Building a Large-Scale Application using the Flutter Framework

Documentation of Practical Project for a Course in the
Bachelor of Science Program “Computer Science and Media Technology”.
Taught at the Faculty of Computer Science and Engineering Science of
Technical University Cologne.

Presented by	Sebastian Faust
In Collaboration with	Capgemini Cologne
Overseen by	Prof. Dr. Christian Kohls

Gummersbach, 17.09.2019

Contents

1. Introduction

One central problem has plagued mobile developers for the last decade: Different mobile platforms requiring different codebases. A native application built for the Android Operating System[1] cannot run on IOS[2] and vice versa. Maintaining several separate codebases multiplies the work of a given development team. Because this is such a central problem, there have been many approaches to mitigate it over the last decade[3].
Embedded Web Applications, for example, where the first approach to enable so-called "*Cross-Platform Mobile Development*"[4]. This approach allows developers to build their applications with web technologies once and then access their applications from any given mobile operating system through an embedded browser. This way developers can cut down on development time by only maintaining one singular codebase. But this approach makes sacrifices in performance and stability to enable that convenience[5].
Embedded Web Applications were followed by a multitude of other approaches with their own drawbacks and advantages[6].
The newest approach to tackle this problem is called "*Native Cross-Platform Development"*. This approach is promising near-native performance and stability, while still only needing a singular codebase[7]. Native Cross-Platform has gained a lot of attention over the last few years. Searches on Google for the biggest Native Cross-Platform Framework "React Native"[8] have been steadily rising since its release in 2015.

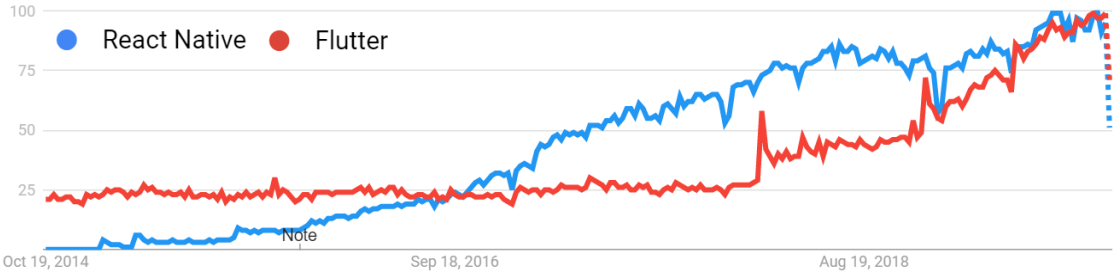

Figure 1 Google Trends: Flutter & React Native [9]

[1] Google LLC, *Android SDK*.

[2] Apple, *IOS SDK*.

[3] Latif et al., 'Review of Mobile Cross Platform and Research Orientations'.

[4] Adinugroho, Reina, and Gautama, 'Review of Multi-Platform Mobile Application Development Using WebView'.

[5] Google LLC, *How Is Flutter Different for App Development*; Adinugroho, Reina, and Gautama, 'Review of Multi-Platform Mobile Application Development Using WebView'; Ebone, Tan, and Jia, 'A Performance Evaluation of Cross-Platform Mobile Application Development Approaches'.

[6] Ebone, Tan, and Jia, 'A Performance Evaluation of Cross-Platform Mobile Application Development Approaches'; Latif et al., 'Review of Mobile Cross Platform and Research Orientations'; Biørn-Hansen, Grønli, and Ghinea, 'Research Challenges in Cross-Platform Mobile'; Chioino et al., 'Designing a Decision Tree for Cross-Device Communication Technology Aimed at IOS and Android Developers'; Sommer and Krusche, *Evaluation of Cross-Platform Frameworks for Mobile Applications*.

[7] Google LLC, *How Is Flutter Different for App Development*; Moore and Nystrom, 'Dart'; Leler, 'What's Revolutionary about Flutter'.

[8] Facebook, *React Native Framework*.

[9] Google LLC, 'Google Trends: Flutter & React Native'.

With Native Cross-Platform Frameworks, developers only need to write their application once and then the written code is compiled into native IOS and Android applications. In 2018 Google has decided to join this new trend by creating its own Native Cross-Platform Framework: "*Flutter*"[10].
Since Flutter is still such a new framework, there has not yet been a clear guideline or consensus on how to tackle a large-scale project using the framework. Many different approaches are split out over blog posts, articles, and documentations. This project aims to change that fact. The result of this project will be a single document that can be used to bridge the gap between understanding the basics of the Flutter Framework and an understanding of one possible way to tackle a large-scale Flutter application.

Chapter two of this documentation will outline the creation process. It will describe how an example application was developed, how the guide was designed on a conceptual level, and how the guide was then actually realized. The third chapter is a personal evaluation of the strengths and weaknesses of the guide. The fourth chapter is a conclusion about the writing process. It outlines which new knowledge was generated by this project.

This documentation will not cover any technical description of how to build an application using the Flutter Framework. That topic is covered by the guide itself.

1.1 Creation Context

The guide[11] was written by a student in the Bachelor of Science Program "Computer Science and Media Technology" at Technical University Cologne[12]. The project was conducted in collaboration with Capgemini Cologne[13]. The finished guide will serve as a deliverable for one of the modules in the Bachelor of Science Program and as a guideline for Capgemini employees that plan on using Flutter in one of their projects. Capgemini released a guide on building an application with Angular in May of 2019 [14], the guide is meant to be the Flutter version of that.

[10] Flutter Dev Team, *The Flutter Framework*.

[11] Faust, 'Flutter Guide'.

[12] Technical University Cologne, 'Technical University Cologne'.

[13] Capgemini, 'Capgemini'.

[14] Ambuludi, Linares, and Contributors, 'Capgemini Angular Guide'.

2. Creation of the Guide

This chapter outlines the creation of the guide[15]. The first section explains how and why an example application was built alongside the writing of the guide. The second section showcases the conceptual creation of the guide and what steps were taken before the writing process began. The third section showcases how the guide was written, why it was written the way it was, which problems arose during the writing process, and how those problems were tackled.

2.1 Building a Reference Application

Wisgen[16] is an open-source example application that was built during the creation of the guide. It displays an infinite list of “wisdom” that it obtains from a variety of sources. Those wisdoms are then displayed with related stock images from the “Unsplash Source API”[17].

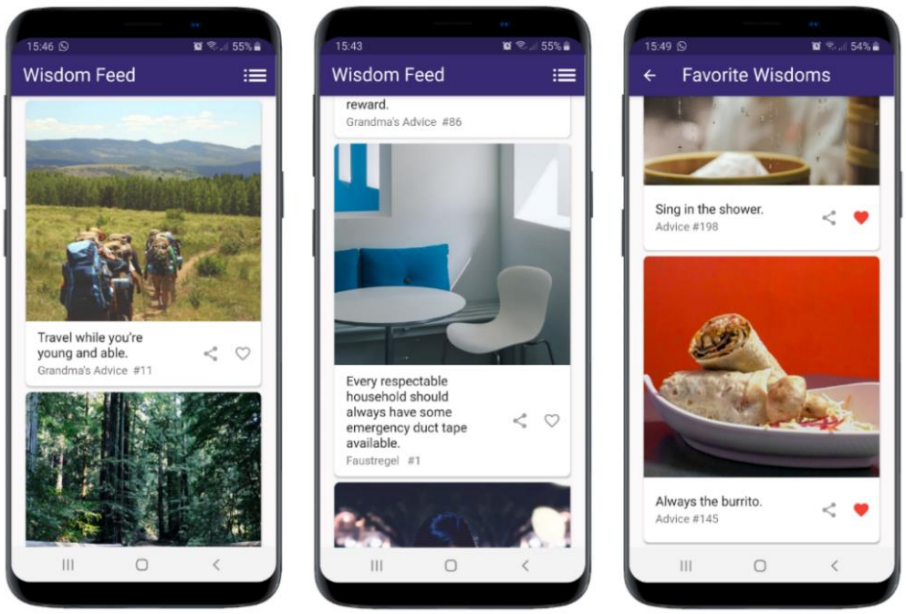

Figure 2 Wisgen application screens [18]

The Wisgen application was built for three reasons: Firstly, to get a better understanding of how to use the Flutter Framework to realize a small project. Secondly, having one example application like Wisgen to draw code examples from, adds to the continuity of the guide. And Lastly, Wisgen showcases all the current best-practices, conventions, and implements a well-defined architectural pattern so that the readers of the guide can look at it as an example.

Wisgen went through multiple iterations during its lifetime.
The first iteration did not have an architecture[19] or a State Management Solution[20].

[15] Faust, ‘Flutter Guide’.

[16] Faust, *Wisgen*.

[17] Unsplash, ‘Unsplash Source API’.

[18] Faust, *Wisgen*.

[19] ISO, ‘ISO/IEC/IEEE 42010’.

[20] Flutter Dev Team, ‘Flutter State’.

The development followed the recommendations laid out by the Flutter team for a first project[21]. This version mainly served to get a basic understanding of the framework. The resulting application was still unsatisfactory, as it was unclear how such an application could be scaled up to anything bigger. Wisgen is a relatively small application and it was already difficult to keep track of State changes.
The second iteration implemented the Provider Package[22] as a State Management Solution. This was one of the recommended approaches by the Flutter team[23]. That version of Wisgen was a clear improvement from the first iteration. State distribution was easier and more predictable. But the lack of architectural rules and no pattern to follow made this solution unsatisfactory as well.
The third and final iteration of Wisgen implemented the BLoC Pattern[24] and a four-layered architecture[25]. This approach was chosen because:

1. BLoC was publicly endorsed by the Flutter Team on multiple occasions[26].
2. BLoC has clear architectural rules[27].
3. BLoC was developed by one of Flutters engineers[28].
4. BLoC has a long list of architectural advantages that are covered in the guide in detail and thus will not be listed here again[29].

After the BLoC pattern was implemented, Wisgen was refactored to follow current Flutter and Dart best-practices and conventions[30]. Unit-Tests were implemented using advice from multiple relevant sources[31]. Wisgen was then documented using the recommendations by the Dart Team[32]. Wisgen's architecture was described using multiple diagrams that were then added to GitHub[33] repository.
This effort did not go unnoticed by the Flutter community. As of the writing of this documentation, Wisgen has been published by three different Flutter community websites as a reference project [34].

[21] Flutter Dev Team, 'Write Your First Flutter App'.

[22] Rousselet and Flutter Dev Team, 'Provider | Flutter Package'.

[23] Hracek and Sullivan, *Pragmatic State Management Using Provider*; Sullivan and Hracek, 'Pragmatic State Management in Flutter'.

[24] Soares, 'Flutter / AngularDart – Code Sharing, Better Together'.

[25] Suri, 'Architect Your Flutter Project Using BLOC Pattern'.

[26] Soares, 'Flutter / AngularDart – Code Sharing, Better Together'; Flutter Dev Team, 'Flutter State'; Sullivan and Hracek, *Technical Debt and Streams/BLoC*; Hracek and Sullivan, *Pragmatic State Management Using Provider*; Sullivan and Hracek, 'Build Reactive Mobile Apps with Flutter'.

[27] Soares, 'Flutter / AngularDart – Code Sharing, Better Together'.

[28] Soares.

[29] Faust, 'Flutter Guide'.

[30] Dart Team, 'Performance Best Practices'; Dart Team, 'Effective Dart'; Krankka, 'Putting Build Methods on a Diet'; Flutter Dev Team, 'Style Guide for Flutter Repo'.

[31] Angelov, 'Unit Testing with "Bloc"'; Dart Team, 'Asynchronous-Tests with the Test Dart Package'; Flutter Dev Team, 'Testing Flutter Apps'; Hracek and Robledo, *Testing Flutter Apps - Making Sure Your Code Works*.

[32] Dart Team, 'Effective Dart'.

[33] GitHub Inc., 'Github'.

[34] Faust, *Wisgen*.

2.2 Conceptualization

This section outlines how the guide was created on a conceptual level. The first part will describe what requirements were set to satisfy both parties involved with the guide and how those requirements where found. The second part highlights what kind of sources the guide is based on and the final part outlines the planning that was conducted before the writing process started.

2.2.1 Requirements

Capgemini's vision for this project was to generate a Flutter version of their existing guide on building a large-scale application with Angular[35]. But they did not yet have a clear set of requirements to follow. The first step of the conceptualization was to generate a list of requirements that both Capgemini and the Technical University of Cologne[36] would be satisfied with. To achieve this, the original Angular guide was analysed and all the requirements it fulfilled, that would be relevant for the Flutter guide, where extracted:

Requirement Type	Requirement Scope	Requirement
Formal	Publication Medium	GitHub Wiki[37] in Markdown file format[38]
	Language	English
		Informal tone
	Page	Has its own table of contents
		Uses Hypermedia to link to documentation and related websites
		Some but not all figures have captions
	License	CC BY-ND[39]
	References	*One reference list at the bottom of each page (with no fixed citation style)*
Functional	Introduction	Has a section describing the authors intent
		Has a section describing the focus group and requirements for understanding the guide
	Guide	*Describes specific architecture that the team developed*
		Includes code examples
		Describes a set of relevant framework features with code examples

Table 1 Capgemini Angular guide requirements

[35] Ambuludi, Linares, and Contributors, 'Capgemini Angular Guide'.

[36] Technical University Cologne, 'Technical University Cologne'.

[37] GitHub Inc., 'Github'.

[38] Gruber and Swartz, 'Markdown'.

[39] Creative Commons, 'Creative Commons CC BY-ND'.

The entries in *red* are requirements that were changed for the final requirement list of the Flutter guide. The first two, because the Technical University of Cologne wanted the guide to conform to more traditional guidelines of scientific writing. The last one because creating a new architecture would have been to work-intensive for this project.

Requirement Type	Requirement Scope	Requirement
Formal	Page	All figures and code examples are labelled
	References	All references follow one consistent citation style
		Work that was influenced or copied from other sources is marked as such and the original work is referenced through a citation

Table 2 Formal requirement additions by the Technical University of Cologne

Using the edited list as a base, two important steps were left to create a clear list of requirements. Defining a target group and deciding on a list of specific topics to cover. As this guide is meant to showcase one approach for building a large-scale Flutter application and not as an introduction to the framework, it was decided to define the target group as: Developers with a basic understanding of Cross-Platform Development and the Flutter Framework. The topics covered by the guide were defined in close cooperation with Capgemini. It was made sure that every information needed to create a large-scale application would at least be touched upon.

Requirement Type	Requirement Scope	Requirement
Functional	The Flutter Framework	How it functions on a technical level
		Best-practices and conventions
	Concepts in Flutter	The Widget Tree
		State
		Immutability
	How to realize a specific functionality in Flutter	Asynchrony
		Testing
		Communication with the Web
	The architecture of a Flutter application	Architecture recommendation
	Evaluation	Personal evaluation of the current state of the Flutter Framework

Table 3 Functional requirements for the guide

It was decided to put the focus of the guide on the architecture recommendation, as this was also the central focus of the Angular guide.

2.2.2 Sources

After the requirements were set, the next step was to acquire high-quality sources to base the guide on. The sources used for the guide can generally be put into four categories: *Scientific*, *official*, *community* and *organizational*.
Scientific sources are publications by highly regarded publishing houses like IEEE[40] or ACM[41]. *Official* sources are any resources that have been published by Flutter[42], Dart[43] or Google. *Community* sources, as the name suggests, are all sources that have been generated by the Flutter, Dart or Computer Science Community. These include blog articles, guides, tutorials, and talks. And lastly, *organizational* sources are items like *Capgemini,* the *Technical University of Cologne*, and the *Android SDK*[44]. Because *organisational* sources do not add to the quality of the guide, they will be ignored for the rest of this section.
The optimal scenario would have been to have an equal three-way split between the first three categories. But as Flutter is still such a new framework, next to no scientific research has so far been conducted on it. As of the writing of this guide, there has not been a single publication on the Flutter Framework through IEEE or ACM. Thus, the scientific sources of the guide are limited to research on Cross-Platform Mobile Development in general, research on Software Architecture, and other broader topics.

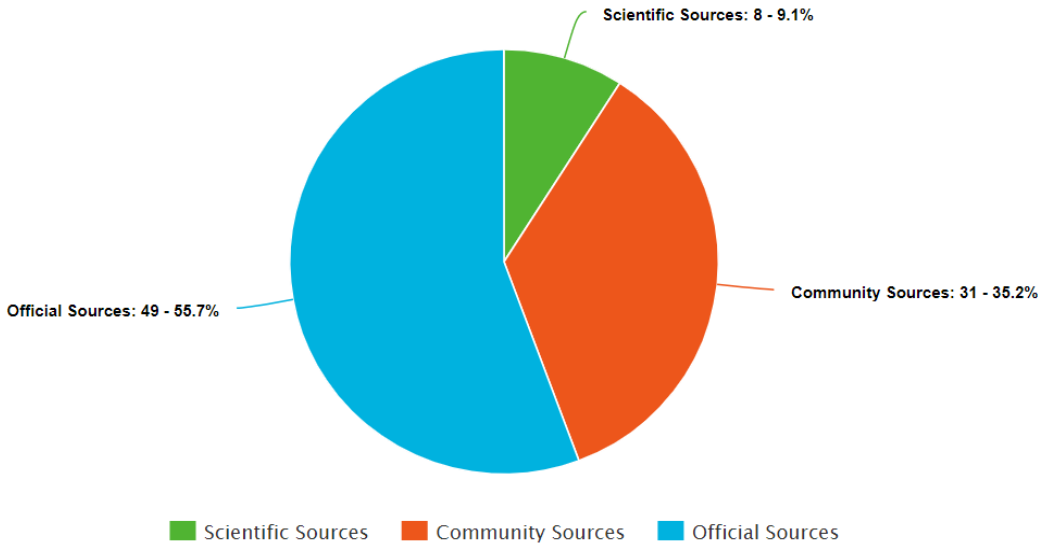

Figure 3 Source category distribution

Not all sources were collected before the writing of the guide started. The need for most of the sources only became obvious during the writing process.

[40] IEEE, 'IEEE Xplore Digital Library'.

[41] ACM, 'ACM Digital Library'.

[42] Flutter Dev Team, *The Flutter Framework*.

[43] Dart Team, 'Dart Programming Language'.

[44] Google LLC, *Android SDK*.

2.2.3 Planning the Writing Process

Once several sources where collected and the Wisgen application was in its second iteration, it was time to outline the guide. It was decided to have the guide consist of six chapters. The following table will showcase which of the chapters is responsible for which of the requirements laid out in section *2.2.1*.

Chapter	Sub Chapter	Responsibility
Introduction	-	Showcase intent of the guide; Explain the focus group; Explain the requirements for understanding the guide
The Flutter Framework	Under the Hood	Explain how Flutter functions on a technical level
	Thinking Declaratively	Explain the concept of State; *Show the difference in programming style to other frameworks*
	The Widget Tree	Explain the concept of Widgets; Explain the concept of the Widget Tree; Explain Statelessness
	Asynchronous Flutter	Showcase Asynchronous Programming in Flutter; Explain communication with the Web in Flutter
Architecting a Flutter app	State-Management Alternatives	*Showcase State-Management Solutions that are not covered in detail by the guide*
	BLoC	Give an architecture recommendation
Testing	-	Showcase how to test a Flutter application
Conventions	-	Highlight current best-practices and conventions
Conclusion	-	Give a personal evaluation of the Flutter Framework; *Reflect on the guides strengths and weaknesses*

Table 4 Chapter responsibility

Responsibilities in *violet* do not strictly cover any of the official requirements. They are additions made during the planning process. The first addition was made because it is very difficult to understand the concept of "*State*"[45] without understanding "*Declarative Programming*"[46]. The addition to the chapter "*Architecting a Flutter app*" was made to give a reference for why BLoC[47] was chosen as an architectural pattern and not one of the many other approaches[48]. The "Conclusion" was extended to point out a few things that would have been done differently when rewriting the guide.

2.3 Realisation

This section showcases how the guide was written and why it was written the way it was. The first part of this section analyses how a chapter is structured on an abstract level and why chapters were chosen to be structured in this way. The second part briefly highlights how Hypermedia was used in the guide. The third part looks at how

[45] Flutter Dev Team, 'Flutter State'.

[46] Flutter Dev Team, 'Introduction to Declarative UI'.

[47] Soares, 'Flutter / AngularDart – Code Sharing, Better Together'.

[48] Egan and Contributors, 'Flutter Architecture Samples'.

the formal requirements listen in section *2.2.1* affected the writing style of the guide. The last part outlines one of the biggest hurdles during the writing process, the handling of citations in a GitHub Wiki[49].

2.3.1 Macrostructure of a Chapter

Most of the chapters in the guide were written using the same *Macrostructure. Macrostructure* in this context refers to how the sections of a given chapter are ordered, and which section is responsible for conveying which information. The common Macrostructure looks like this:

Section	Subsection	Responsibility
Introduction	→	Summaries all *blocks of content* in this chapter; Gives the reader the option to skip a chapter, or sections of it if they are already familiar with the content
Block of content	↴	Conveys one idea; One chapter has multiple *blocks of content* related to an overarching topic
	Abstract concept	Beginning of a *block of content;* The information that will be conveyed on an abstract level
	Worked-out example	Gives an example of the *abstract concept;* Mostly code examples
	Visual aid *(where possible)*	If the concept can be visualized, this graphic helps to understand it
	Short summary *(where possible)*	A condensed version of the information contained in this *block of content;* One or two sentences long; Labelled with "*TLDR*"; Helps readers that are only skimming the chapter

Table 5 Macrostructure of a chapter

This structure was chosen based on two scientific studies conducted on the topic of learning.
The first one being a study conducted by Alexander Renkl in 1997[50]. Renkl observed, that providing concert examples for complex concepts helped his subject in understanding them. This structure aims to achieve the same effect. Renkle also coined the term "*worked-out examples*", for an abstract concept being displayed in a relevant, complete example.
The second source for this Macrostructure is a journal article on visual learning in technical fields by McGraths and Browns published in 2005[51]. They found that providing a visual aid for complex technical concepts improves student engagement and understanding. The guide aims to achieve those two effects by providing as much visual aid as possible.

This next figure showcases how this Macrostructure can be observed in the chapter "*Widget Tree*"[52]. All relevant sections described in table five are highlighted.

[49] GitHub Inc., 'Github'.

[50] Renkl, 'Learning from Worked-out Examples'.

[51] McGrath and Brown, 'Visual Learning for Science and Engineering'.

[52] Faust, 'Flutter Guide'.

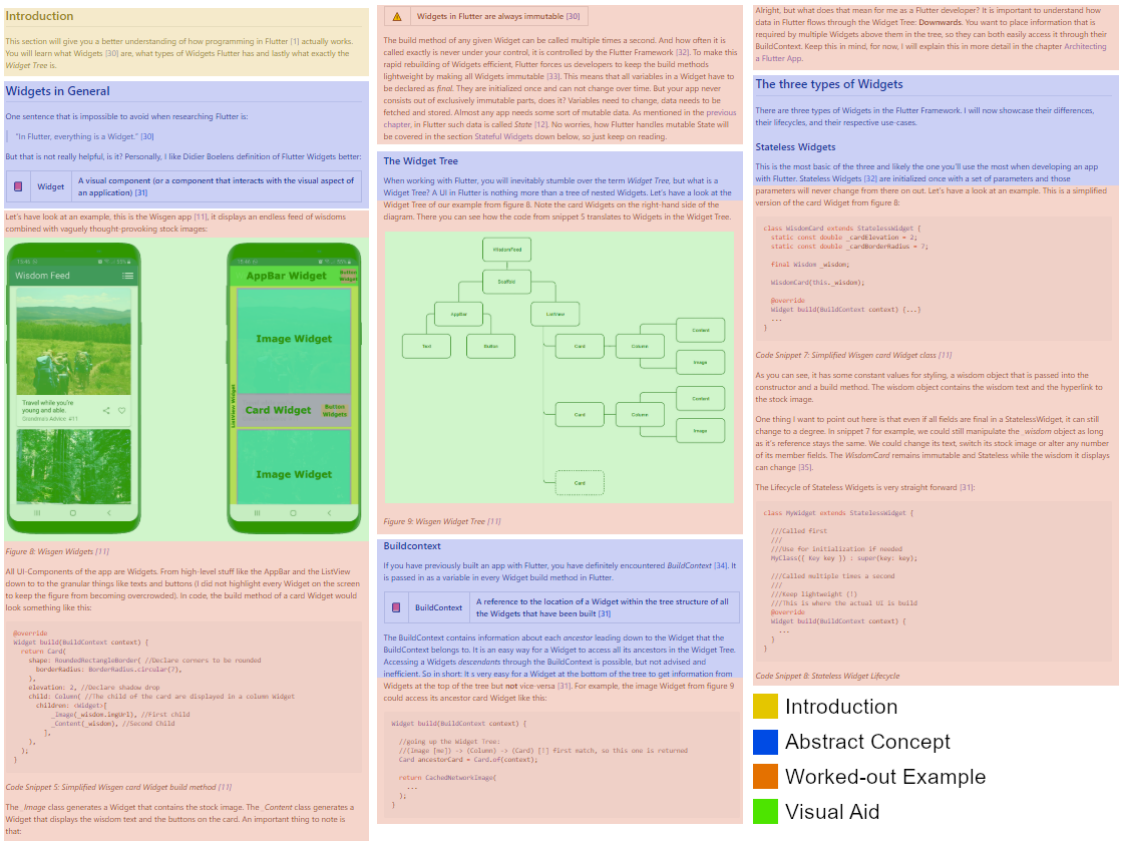

Figure 4 Macrostructure of chapter "*Widget Tree*" [53]

In this chapter, as in most chapters, all visual aids and all worked-out examples relate to the Wisgen application. This way the continuity within the guide is improved and we achieve one "*Thread*"[54] for the reader to hold onto.

2.3.2 Hypermedia

Using Hypermedia to link to relevant sources was one of the requirements for the guide. The general rule during the writing process was to use as much Hypermedia as possible. All citations that have an online origin are linked to that origin. Cross-references within the guide are also linked. All chapters begin with a table of contents that links to the different sections of that chapter. Every chapter ends with a "back to top" navigation and a "next chapter" navigation. The guide is published as an online resource and the goal was to use the potential of that medium to its fullest.

2.3.3 Writing Style

The text of the guide follows one common *Writing Style*. *Writing Style* in this context refers to how the text was written on a sentence to sentence level. Topics like tone and sentence structure.
As per the requirement listed in section *2.2.1*, the guide was written in a personal, informal tone. This was realized through the use of the pronoun "*you*", and informal phrases.

[53] Faust.

[54] Merriam-Webster, 'Definition of Thread'.

Sentences of the guide were generally kept as short as possible. A study by Ikuo Kitagaki shows that short sentences in educational material can improve the ability to recall information significantly[55].

The following figure is an excerpt from the chapter "*BLoC*"[56]. The pronoun you, informal phrasing and instances of Hypermedia are highlighted.

Introduction

BLoC is an architectural pattern that functions as a State Management Solution. It was designed by Paolo Soares in 2018 [7]. Its original purpose was to enable code sharing between Flutter [1] and Angular Dart [72] applications. This chapter will give you an in-depth understanding of what the BLoC Pattern is and how it works. You will learn how to implement it using the BLoC Package [39] by Felix Angelov. And Finally, you will learn how to use the BLoC Pattern to achieve a scalable four-layered architecture for your application.

Intro to BLoC

When Soares designed the BLoC Pattern, he was working on applications in both Flutter and Angular Dart. He wanted a pattern that enabled him to hook up the same business logic to both Flutter and Angular Dart apps. His idea was to remove business logic from the UI as much as possible and extract it into its own classes, into BLoCs (Business Logic Components). The UI should only send *Events* to BLoCs and display the UI based on the *State* of the BLoCs. Soares defined, that UI and BLoCs should only communicate through Streams [40]. This way the developer would not need to worry about manually telling the UI to redraw. The UI can simply subscribe to a Stream of State [12] emitted by a BLoC and change based on the incoming State [7], [48], [53], [70].

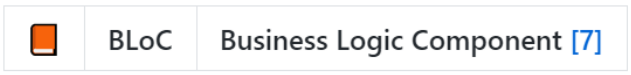

Figure 5 Writing style of the chapter "*BLoC*" [57]

2.3.4 References in a GitHub Wiki

Handling references and citations by hand can be a time-consuming process. Especially when handling a long list of references like the one for the Flutter guide (102 references)[58]. To minimize this work overhead, many digital citation-tools have been released over the last decade[59]. The one used during the writing of the guide is Zotero[60]. One problem when writing a citation heavy text in Markdown[61] is that Zotero or any of the other current citation tools do not support that text format natively.

[55] Kitagaki, 'Effect of English Short Sentences Memorization on the Speaking Skill and the E-Learning of English'.

[56] Faust.

[57] Faust.

[58] Faust.

[59] Homol, 'Web-Based Citation Management Tools'.

[60] Corporation for Digital Scholarship, 'Zotero'.

[61] Gruber and Swartz, 'Markdown'.

The first approach was to only use Zotero as a database for all the references and manually copy a given citation into the Markdown file as plaintext. Used references were marked with a label in Zotero. At the end of the writing process, all labelled references could then have been turned into a reference list.
This approach came with multiple problems. The citation style had to be chosen once and could not easily be changed. The labelling and un-labelling of references added additional overhead. Adding a new citation in the guide might require editing the citations that came before or after it as the citation numbers had to be tracked manually.

After finishing the second chapter of the guide, the mental overhead of handling ~80 citations became too much and the search for a better solution began. This search led to the discovery of a Zotero extension called “Better BibTex”[62], which adds two functionalities to Zotero: Exporting citations as unique, plaintext keys and exporting a Zotero library as a BibTex file[63]. BibTex is a standardized file format for reference-lists that is traditionally used in combination with Latex[64]. With this Zotero extension, a unique key could be inserted wherever a citation was needed in the Markdown file. This unique key could then be mapped to the relevant citation in the BibTex file.

```
## The Goal of this Guide
This guide aims to bridge the gap between the absolute Flutter
[[@flutterdevteamFlutterFramework2018]](https://flutter.dev/)
basics and clean, structured Flutter development.
```

Figure 7 Markdown citation using a unique key generated by Better BibTex

```
@software{flutterdevteamFlutterFramework2018,
  title = {The {{Flutter Framework}}},
  url = {https://flutter.dev/},
  organization = {{Google LLC}},
  urldate = {2019-09-20},
  date = {2018},
  author = {{Flutter Dev Team}}
}
```

Figure 6 BibTex entry

The next step was to automate the mapping and generate a reference list out of all mapped citations.

[62] Heyns, ‘Better BibTeX for Zotero’.

[63] Feder, ‘BibTeX’.

[64] Lamport and Contributors, ‘LaTeX’.

Pandoc[65] is a command-line tool used for transforming one file format to another. Pandoc has an extension called Cite-Proc[66]. This extension takes in an input text document, a file defining a citation style, and a BibTex file and then outputs a document with mapped citations and a reference list. Crucially for this use case, Pandoc and Cite-Proc support the transformation of one Markdown file into another Markdown file.
There was still one last problem with this approach. Pandoc and Cite-Proc were only designed to support a single document input and a single document output. The GitHub Wiki, however, consists of many pages and thus of many Markdown files. To use Pandoc and Cite-Proc effectively, the decision was made to have one large *master file* that contains the entire guide. This *master file* would be transformed using Pandoc and Cite-Proc and then split into the actual pages of the Wiki. A Dart[67] script was written to make this splitting automatic. This script also preserved the table of contents at top of each page and the navigation at the bottom of each page. That way the *master file* only contains the content of the guide and not the navigational overhead. The resulting workflow looked like this:

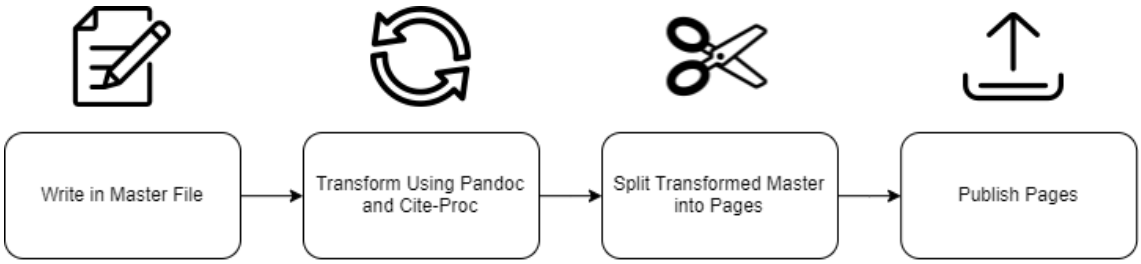

Figure 8 Writing workflow

With this workflow, the citation overhead was essentially removed. Keys could be inserted during the writing and all other work was done by Zotero, Pandoc, Cite-Proc, and the Splitting-Scrips.

[65] MacFarlane, *Pandoc*.

[66] MacFarlane, *Citeproc*.

[67] Dart Team, 'Dart Programming Language'.

3. Evaluation

This chapter is going to be my personal evaluation of the guide. I will highlight some of the guide's strengths and some of the guide's weaknesses.

I am satisfied with how the guide turned out. I managed to find a diverse range of sources for most of the topics I wanted to cover. I kept close to the informal style of the original Angular guide[68] and I think the guide is easier to read because of this choice. The guide did end up being longer than I planned (~50 A4 pages without figures and references). If I would write it again, I would choose a more narrow and clear scope for what the guide should include. For example, I might have excluded or shortened the chapter "The Flutter Framework" and instead focused even more on the chapter "Architecting a Flutter App". I would have also liked to include more scientific sources on Flutter. But as mentioned in section *2.2.2* of this documentation, next to no scientific research has so far been conducted on it.

I think I fulfilled the goal I set myself when starting this guide. To bridge the gap between the basics of Flutter and clean, structured Flutter development that can work in a large-scale project. The result is a resource I would have liked to have when bridging that gap myself.

[68] Ambuludi, Linares, and Contributors, 'Capgemini Angular Guide'.

4. Conclusion

This project started with the development of an example application[69]. This application was further refined and iterated during the course of the project. After the second iteration of the application was finished, the requirements for the guide were defined. Then the planning process and collection of sources began. Once a sufficient number of sources were collected and a plan for the guide was laid out, the writing began. One common writing structure was defined and followed for each chapter. During the writing, the manual handling of citations became more and more difficult and an alternative approach was found and used from there on out.

During the wiring of the guide, a deep understanding of how the Flutter Framework[70] functions and how one might develop a large-scale application with it, was obtained. The knowledge was condensed into a publicly available document. This document contains a concrete recommendation on how to approach a large-scale project using the Flutter Framework. This resource can be used by new Flutter developers from here on out.

Additionally, an example Flutter application that follows all the current best-practices, conventions, and implements a well-defined architectural pattern was created. This application can be used as an aid for peers developing their own applications with Flutter.

Furthermore, a generic structure for online guides was developed. This structure is based on scientific research in the field of learning. It is laid out in detail in section *2.3*. Peers can use this structure as an aid to develop their own guides.

Lastly, one possible way to use Zotero[71] in combination with a GitHub Wiki[72] was found. This approach can be useful for peers aiming to create a guide or project documentation with a long list of citations.

4.1 Future of this Project

The guide was commissioned by Capgemini Cologne[73]. It is supposed to aid their mobile developers when building large-scale applications using the Flutter Framework. It will also be the basis for the Bachelor Thesis I am starting this November. In my Thesis, I will create and document a large-scale Flutter application for Capgemini. It is supposed to be an example project that other developers can use as a guideline for creating their own large-scale Flutter applications. More specifically, Capgemini has used the “My Thai Star” application[74] as a reference project for Angular[75]. My Thesis will be the creation of a Flutter version of that application.

[69] Faust, *Wisgen*.

[70] Flutter Dev Team, *The Flutter Framework*.

[71] Corporation for Digital Scholarship, ‘Zotero’.

[72] GitHub Inc., ‘Github’.

[73] Capgemini, ‘Capgemini’.

[74] Linares, Gonzalez, and Contributors, *My Thai Star*.

[75] Google LLC, ‘Angular’.

4.2 Other Future Work

I am planning on publishing a more detailed guide on how to handle citations in GitHub Wikis[76]. I obtained a lot of knowledge on that topic during the writing process and I plan on sharing it with the GitHub community. I am also planning on releasing a generic version of the Splitting Script described in section *2.3.4*.

One thing I and peers in the mobile development community[77] would like to see is more scientific research on the Flutter Framework: Large-scale benchmarking, usability tests, scalability tests, and architecture evaluations.

[76] GitHub Inc., ‘Github’.

[77] Biørn-Hansen, Grønli, and Ghinea, ‘Research Challenges in Cross-Platform Mobile’.

References

ACM. ‘ACM Digital Library’. Not-for-profit Membership Corporation, 1947. https://dl.acm.org/.

Adinugroho, Timothy Yudi, Reina, and Josef Bernadi Gautama. ‘Review of Multi-Platform Mobile Application Development Using WebView: Learning Management System on Mobile Platform’. *Procedia Computer Science*, International Conference on Computer Science and Computational Intelligence (ICCSCI 2015), 59 (1 January 2015): 291–97. https://doi.org/10.1016/j.procs.2015.07.568.

Ambuludi, Juan Andrés, Santos Jiménez Linares, and Contributors. ‘Capgemini Angular Guide’. Guide. GitHub, 2019. https://github.com/devonfw/devon4ng.

Angelov, Felix. ‘Unit Testing with “Bloc”’. Tutorial. Medium, 2019. https://medium.com/flutter-community/unit-testing-with-bloc-b94de9655d86.

Apple. *IOS SDK* (version 13). Cross-platform, Swift. Apple, 2010. https://developer.apple.com/ios/.

Biørn-Hansen, Andreas, Tor-Morten Grønli, and Gheorghita Ghinea. ‘A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development’. *ACM Comput. Surv.* 51, no. 5 (November 2018): 108:1–108:34. https://doi.org/10.1145/3241739.

Capgemini. ‘Capgemini - Home Page’. Home Page, 2019. https://www.capgemini.com/us-en/.

Chioino, Jamil, Ivan Contreras, Alfredo Barrientos, and Luis Vives. ‘Designing a Decision Tree for Cross-Device Communication Technology Aimed at IOS and Android Developers’. In *Proceedings of the 2Nd International Conference on Information System and Data Mining*, 81–87. ICISDM ’18. New York, NY, USA: ACM, 2018. https://doi.org/10.1145/3206098.3206103.

Corporation for Digital Scholarship. ‘Zotero’, 2006. https://www.zotero.org/.

Creative Commons. ‘Creative Commons CC BY-ND’, 2001. https://creativecommons.org/licenses/by-nd/3.0/.

Dart Team. ‘Asynchronous-Tests with the Test Dart Package’. Guide. Dart packages, 2019. https://pub.dev/packages/test#asynchronous-tests.

———. ‘Dart Programming Language’. Documentation, 2019. https://dart.dev/.

———. ‘Effective Dart’. Guide, 2019. https://dart.dev/guides/language/effective-dart.

———. ‘Performance Best Practices’. Guide, 2018. https://flutter.dev/docs/testing/best-practices.

Ebone, A., Y. Tan, and X. Jia. ‘A Performance Evaluation of Cross-Platform Mobile Application Development Approaches’. In *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 92–93, 2018.

Egan, Brian, and Contributors. ‘Flutter Architecture Samples’. Architecture Samples. fluttersamples, 2017. https://fluttersamples.com/.

Facebook. *React Native Framework*. Cross-platform, JavaScript. Facebook, 2015. https://facebook.github.io/react-native/.

Faust, Sebastian. ‘Flutter Guide’. Guide. GitHub, 2019. https://github.com/Fasust/flutter-guide.

———. *Wisgen*. Cross-platform, Dart. Germany, 2019. https://github.com/Fasust/wisgen.

Feder, Alexander. ‘BibTeX’, 2006. http://www.bibtex.org/.

Flutter Dev Team. ‘Flutter State’. Documentation, 2019. https://flutter.dev/docs/development/data-and-backend/state-mgmt.

———. ‘Introduction to Declarative UI’. Documentation, 2019. https://flutter.dev/docs/get-started/flutter-for/declarative.

———. ‘Style Guide for Flutter Repo’. Guide. GitHub, 2018. https://github.com/flutter/flutter/wiki/Style-guide-for-Flutter-repo.

———. ‘Testing Flutter Apps’. Tutorial, 2018. https://flutter.dev/docs/testing.

———. *The Flutter Framework* (version 1.9). Cross-platform, Dart. Google LLC, 2018. https://flutter.dev/.
———. ‘Write Your First Flutter App’. Guide, 2018. https://flutter.dev/docs/get-started/codelab.
GitHub Inc. ‘Github’. GitHub, 2008. https://github.com.
Google LLC. *Android SDK* (version 10). Cross-platform, Java. Google LLC, 2008. https://developer.android.com/.
———. ‘Angular’. Documentation, 2016. https://angular.io/.
———. ‘Google Trends: Flutter & React Native’. Data Analysis. Google Trends, 2019. https://trends.google.de/trends/explore?date=today%205-y&q=React%20Native,Flutter.
———. *How Is Flutter Different for App Development*. Google Developers Official Youtube Channel, 2019. https://www.youtube.com/watch?v=I-YO9CmaSUM&feature=youtu.be.
Gruber, John, and Aaron Swartz. ‘Markdown’. Definition, 2004. https://daringfireball.net/projects/markdown/.
Heyns, Emiliano. ‘Better BibTeX for Zotero’. Documentation, 2017. http://retorque.re/zotero-better-bibtex/.
Homol, Lindley. ‘Web-Based Citation Management Tools: Comparing the Accuracy of Their Electronic Journal Citations’. *The Journal of Academic Librarianship* 40, no. 6 (1 November 2014): 552–57. https://doi.org/10.1016/j.acalib.2014.09.011.
Hracek, Filip, and Anthony Robledo. *Testing Flutter Apps - Making Sure Your Code Works*. Tutorial. Vol. Ep. 21. The Boring Flutter Development Show, 2019. https://www.youtube.com/watch?v=bj-oMYyLZEY&.
Hracek, Filip, and Matt Sullivan. *Pragmatic State Management Using Provider*. Vol. 24. The Boring Flutter Development Show, 2019. https://www.youtube.com/watch?v=HrBiNHEqSYU.
IEEE. ‘IEEE Xplore Digital Library’. Professional association, 1963. https://ieeexplore.ieee.org/Xplore/home.jsp.
ISO. ‘ISO/IEC/IEEE 42010:2011 Systems and Software Engineering — Architecture Description’. ISO, 2011. http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/05/50508.html.
Kitagaki, Ikuo. ‘Effect of English Short Sentences Memorization on the Speaking Skill and the E-Learning of English’. *Procedia - Social and Behavioral Sciences*, 13th International Educational Technology Conference, 103 (26 November 2013): 348–51. https://doi.org/10.1016/j.sbspro.2013.10.343.
Krankka, Iiro. ‘Putting Build Methods on a Diet’. Blog. iirokrankka.com, 2018. https://iirokrankka.com/2018/06/18/putting-build-methods-on-a-diet/.
Lamport, Leslie, and Contributors. ‘LaTeX’, 1984. https://www.latex-project.org/.
Latif, Mounaim, Younes Lakhrissi, El Habib Nfaoui, and Najia Es-Sbai. ‘Review of Mobile Cross Platform and Research Orientations’. In *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, 1–4, 2017. https://doi.org/10.1109/WITS.2017.7934674.
Leler, Wm. ‘What’s Revolutionary about Flutter’. Blog. *Hackernoon* (blog), 2017. https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514.
Linares, Santos Jiménez, Dario Rodriguez Gonzalez, and Contributors. *My Thai Star*. En. 2017. Reprint, devonfw, 2019. https://github.com/devonfw/my-thai-star.
MacFarlane, John. *Citeproc*. Haskell, 2019. https://github.com/jgm/pandoc-citeproc.
———. *Pandoc*. En, 2006. https://pandoc.org/.
McGrath, M.B., and J.R. Brown. ‘Visual Learning for Science and Engineering’. *IEEE Computer Graphics and Applications* 25, no. 5 (September 2005): 56–63. https://doi.org/10.1109/MCG.2005.117.
Merriam-Webster. ‘Definition of Thread’. Definition, 2019. https://www.merriam-webster.com/dictionary/thread.

Moore, Kevin, and Bob Nystrom. ‘Dart: Productive, Fast, Multi-Platform - Pick 3’. Conference Talk presented at the Google I/O’19, Mountain View, CA, 9 May 2019. https://www.youtube.com/watch?v=J5DQRPRBiFI.

Renkl, Alexander. ‘Learning from Worked-out Examples: A Study on Individual Differences’. *Cognitive Science* 21, no. 1 (1 January 1997): 1–29. https://doi.org/10.1016/S0364-0213(99)80017-2.

Rousselet, Remi, and Flutter Dev Team. ‘Provider | Flutter Package’. Documentation. Dart packages, 2018. https://pub.dev/packages/provider.

Soares, Paolo. ‘Flutter / AngularDart – Code Sharing, Better Together’. Conference Talk presented at the DartConf 2018, Google Campus, LA, 25 January 2018. https://www.youtube.com/watch?v=PLHln7wHgPE.

Sommer, Andreas, and Stephan Krusche. *Evaluation of Cross-Platform Frameworks for Mobile Applications*. Gesellschaft für Informatik e.V., 2013. http://dl.gi.de/handle/20.500.12116/17386.

Sullivan, Matt, and Filip Hracek. ‘Build Reactive Mobile Apps with Flutter’. Conference Talk presented at the Google I/O ’18, Mountain View, CA, 10 May 2018. https://www.youtube.com/watch?v=RS36gBEp8OI.

———. ‘Pragmatic State Management in Flutter’. Conference Talk presented at the Google I/O’19, Mountain View, CA, 9 May 2019. https://www.youtube.com/watch?v=d_m5csmrf7I.

———. *Technical Debt and Streams/BLoC*. Vol. 4. The Boring Flutter Development Show, 2018. https://www.youtube.com/watch?v=fahC3ky_zW0.

Suri, Sagar. ‘Architect Your Flutter Project Using BLOC Pattern’. Guide. *Medium* (blog), 2019. https://medium.com/flutterpub/architecting-your-flutter-project-bd04e144a8f1.

Technical University Cologne. ‘Technical University Cologne’. Home Page, 2019. https://www.th-koeln.de/en/homepage_26.php.

TU Dresden. ‘Declaration of Originality’. Document. TU Dresden. Accessed 17 September 2019. https://tu-dresden.de/bu/bauingenieurwesen/studium/im-studium/access/copy2_of_index.

Unsplash. ‘Unsplash Source API’. Documentation, 2017. https://source.unsplash.com/.

Appendix

A.1 Milestones of the writing process

Month	August (08)					September (09)		
Week Relative	2	3	4	5	6	7	8	
Week Number	33	34	35	36	37	38	39	
Milestones	**1. Prof Meeting,** **Exposé,** Wisgen 1st Iteration	Wisgen 2nd Iteration	Outline for the Guide, **Requirements,**	Summarize and collect Sources	Wisgen 3rd Iteration	**Requirements 2.0,** More Sources, Docu: Outline, Guide: Intro, Guide: Under the Hood	**2. Prof Meeting,** Guide: Thinking, Declaratively, Guide: Widget Tree 1/2	
Capgemini				Working Student				

Month	October (10)				
Week Relative	9	10	11	12	13
Week Number	40	41	42	43	44
Milestones	Guide: Widget Tree 2/2, Guide: Async, Guide: Architecture 1/2, Pandoc Citations	Guide: Architecture 2/2, Guide: Testing, Guide: Conventions, Guide: Conclusion,	Guide: Third Pass Doku: Outline, Doku: Intro, Doku: Main 1/2	**3. Prof Meeting** Doku: Main 1/2, Doku: Eval, Doku: Conclusion Guide: Proofing	Doku: Third Pass Doku: Proofing, **Hand in**
Capgemini	Working Student				

A.2 Snapshot of the stages of writing the guide

Sections		Sources	Media	Draft	Write	Mark Refs	Second Pass	Third Pass	Proofing	Split Pages	Total
000 Introduction	2	100	100	100	100	100	100	0	0	100	76
100 The Flutter Framework	1	100	100	100	100	100	100	0	0	100	76
110 Under the Hood	2	100	100	100	100	100	100	0	0	100	76
120 Thinking Declaratively	2	100	100	100	100	100	100	0	0	100	76
130 The Widget Tree	4	100	100	100	100	100	100	0	0	100	76
140 Asynchronous Flutter	4	100	100	100	100	100	100	0	0	100	76
200 Architecting a Flutter App	2	100	100	100	100	100	100	0	0	100	76
210 Stage-management Alternatives	3	100	100	100	100	100	100	0	0	100	76
220 BLoC	4	100	100	100	100	100	100	0	0	100	76
300 Testing	2	100	50	100	50	0	0	0	0	100	43
400 Conventions	2	100	0	100	0	0	0	0	0	100	21
500 Conclusion	2	100	0	100	0	0	0	0	0	100	21
	Weights	*4*	*3*	*1*	*10*	*1*	*2*	*3*	*4*	*1*	66
		Find	Code			Pandoc	One day after writing	Once everything is written	Self	Wrote tool	
		Summarize	Images			Hyperlinks			Grammarly		

A.3 Snapshot of the stages of the writing the documentation

Sections		Draft	Media	Write	Second Pass	Third Pass	Proofing	Total	Template
I Personal Statement	1	100	100	100	100	100	100	100	
1. Introduction	2	100	100	100	100	100	0	85	
1.1 Context	1	100	100	100	100	100	0	85	
2. Creating a guide to the Flutter Framework	1	100	100	100	100	100	0	85	
2.1 Building a reference application	2	100	100	100	100	100	0	85	
2.2 Conceptualization	1	100	100	100	100	100	0	85	
2.2.1 Requirements	2	100	100	100	100	100	0	85	
2.2.2 Sources	2	100	100	100	100	100	0	85	
2.2.3 Planning the writing process	2	100	100	100	100	100	0	85	
2.3 Realisation	1	100	100	100	100	100	0	85	
2.3.1 Macrostructure of a Chapter	2	100	100	100	100	0	0	70	
2.3.2 Hypermedia	1	100	100	100	100	0	0	70	From Guide
2.3.3 Writing Style	2	100	100	100	100	0	0	70	
2.3.4 References in a GitHub Wiki	4	100	100	100	100	0	0	70	
3. Evaluation	1	100	100	100	100	0	0	70	From Guide
4. Conclusion	2	100	100	100	100	0	0	70	
4.1 Future of this Project	1	100	100	100	100	0	0	70	
4.2 Other Future Work	1	100	100	100	100	0	0	70	
	Weights	*1*	*2*	*8*	*3*	*3*	*3*	80	

Images

Diagrams

One day after writing

Once everything is written

Self

Grammarly