Electron and *Tauri*: Comparison of web-frameworks for building cross-platform desktop applications

Kevin Lüttge
University of Kassel

ABSTRACT

Web-frameworks like Electron provide the possibility to easily develop cross-platform desktop applications by using common web-technologies such as Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. The increasing popularity of those frameworks is shown by various day-to-day applications namely WhatsApp, Discord or Visual Studio Code that are built with Electron. Two years ago Tauri came up, whose developers claimed it to be faster, smaller and more focused on security than other alternative frameworks. This paper will discuss the differences and similarities between Electron and Tauri, in terms like architecture, performance or security. Therefore a counter application is implemented for each framework to analyze and compare the development process.

1. INTRODUCTION

Cross-platform frameworks like Electron, Tauri, Flutter or Proton are providing the possibility of using web-technologies like HTML, CSS and JavaScript or whole web-frameworks like Angular or React to develop or migrate classic desktop applications to web applications. The classic approach of implementing desktop applications needs the developer to consider different Application Programming Interface (API) or environments of the major operating systems Operating System (OS) Windows, Mac and Linux. In fact each application has to be implemented multiple times to adapt OS-specific requirements.

In context of consumer applications this allows companies and product owners to develop new target groups, but also increases the pool of potential applicants, since those frameworks provide web developers to implement desktop applications without the knowledge of standard programming languages that are used for desktop applications like C/C++ or Java. But also institutions or universities and their students benefit of it, because due to the fact of cross-platform development, applications can be used by many devices and decrease costs since they do not need particular hardware.

1.1 Background

Over the past years frameworks for building desktop applications have get more and more attention. This is owed to the fact of fundamental advances at web technologies like TypeScript or Frontend Frameworks like Angular or React [5], which lead to a wider use of such technologies for various scenarios. A lot of every day applications used by computer scientists like Visual Studio Code or GitHub Desktop, but also applications with usage spread over industry sectors like Microsoft Teams, Skype or Discord and even Social Media applications like WhatsApp or Twitch are implemented with such frameworks. The trend of using those applications has even grow up since the outbreak of SARS-CoV-2 in early 2020 [3] and the increased number of employees working from home as a result of lockdowns all over the world. This lead to an increasing number of cross-platform web-frameworks with different approaches in context of building, security or performance.

1.2 Motivation

As mentioned in Chapter 1.1 various cross-platform web-frameworks came up using different technologies or languages. One of the oldest and mostly used frameworks is Electron, which is backed up by several popular applications implemented with Electron like Visual Studio Code, Discord or Twitch. Therefore, Electron has become a standard in context of cross-platform development of desktop applications and almost every new framework invented is benchmarked against it like [4] or [2]. Two years ago a new framework called Tauri was introduced which is designed to improve several aspects of Electron especially in case of performance, memory usage and security [1]. Since these statements are made by the developers this paper aims to provide an objective overview to the reader of both frameworks Electron and Tauri. Therefor fundamental architecture as well as the frontend and backend core of each framework is explained in detail at chapter 2 for Electron and at chapter 3 for Tauri. This technological knowledge is applied by implementing a basic application which will be described in detail at chapter 4 and analyzes different aspects. To obtain an objective statement of the advantages and disadvantages of each framework chapter 5 compares the results of previous sections to isolate advantages and disadvantages to the reader. At the end of this paper the results of previous chapters are contrasted against the statement of the Tauri Developers in [1] and being discussed to obtain an objective comparison.

1.3 Related Work

An exploratory study of [7] has worked out, that applications developed with cross-platform web-frameworks are made for various kinds of usage. The authors also empirically documented that those frameworks are mostly used by developer teams with a median size of 1, which is a direct result of the amount resources classical desktop development consumes. Nevertheless, they also discovered disadvantages like a high number of used libraries due to the fact of compensating lack of features provided by web-frameworks, but also a high ratio of platform-related issues to all issues of 20%

The increasing popularity of using web-technologies has been shown by [5]. Since they identified their native Java Swing desktop application as a bottleneck, the authors decided to replace it with a technology stack providing long term sustainability. Therefor they used several web-technologies like AngularJS or Typescript to implement "a web-based tool for configuring experiments on the National Ignition Factory". But they also came in touch with typical problems of the JavaScript ecosystem like exit or replacement of widely used technologies bringing them to be forced to migrate AngularJS to Angular. It has to be mentioned that the authors emphasized the community support especially in case of migration by providing tools those cases.

Electron as a standard benchmark for cross-platform web-frameworks is shown by [2] or [4]. Both authors compared different web-frameworks against Electron. Although the benchmark of the authors of [2] was made with different Integrated Development Environment (IDE)s and the comparison between Flutter and Electron by [4] was based on beta version of Flutter both found out that at the one hand Electron has better performance in case of execution time and CPU consumption and on the other that it provides more features than the compared frameworks.

2. ELECTRON

Electron originally was released as Atom Shell by GitHub at 15^{th} July, 2013 [8]. The intention of the developers was "to handle the Chromium/Node.js event loop integration and native APIs" [6] for the Atom Editor. On 23^{rd} April, 2015 it was renamed to Electron and announced that the developers want to provide a framework that allows to build desktop applications with the using web-technologies.

2.1 Architecture

This subsection takes a deeper look at the underlying features and techniques like Chromium and so on.

2.2 Frontend

This subsection deals with the front-end Electron is using, which other frontend-frameworks are supported, etc.

2.3 Backend

At this subsection we will take a look at the Node JS Backend of Electron and analyse the advantages and disadvantages of NodeJS

3. TAURI

History, why/when it was released, written in ..., general information

3.1 Architecture

Same as described for Tauri.Ref to introduction2.1

3.2 Frontend

3.3 Backend

4. IMPLEMENTATION OF A COUNTER APPLICATION

This section will include a short explanation of the counter app as an example application. It will show the features and define the requirements for an objective comparison.

4.1 Electron

This subection section will take the findings of 2 into action and analyze the development and building process as well as the performance of the executable application.

4.1.1 Development

At this subsubsection the general development process of an Electron application is discussed but also the actual development using the screencast application as an example. Are there any templates provided, which dependencies and tools are needed for implemention, debbuging and testing. Are they already provided by the installer? Also have a short look at the documentation e.g. are there any guides, community feedback.

4.1.2 Build

This subsubsection deals with the whole building process of the example Electron app e.g. Package manager, cross-platform support, build tools, build time etc.

4.1.3 Performance

At this subsubsection the actual performance of the example application is analyzed. Memory consumption, known security issues, bugs, freezes etc.

4.2 Tauri

This subsection section will take the findings of 3 into action and analyze the development and building process as well as the performance of the executable application.

4.2.1 Development

Same as described in 4.1.1 for Tauri.

4.2.2 Build

Same as described in 4.1.2 for Tauri.

4.2.3 Performance

Same as described in 4.1.3 for Tauri.

5. SUMMARY

This section will take the knowledge of 2 and 3 as well as the analysis of 4 to contrast them.

5.1 Differences

This subsection will work out the differences between the two frameworks with regard to Architecture, Frontend, Backend, Development, Build and Performance.

5.2 Similarities

This subsection will work out possible similarities between the two frameworks with regard to Architecture, Frontend, Backend, Development, Build and Performance.

6. CONCLUSIONS

At this section a conclusion is made based on the advantages/disadvantages of each framework described in 6. Are there different use-cases for each framework, are they concurrent to each other, etc.?

7. LIST OF ABBREVIATIONS

HTML Hypertext Markup Language

CSS Cascading Style Sheets

API Application Programming Interface

OS Operating System

IDE Integrated Development Environment

8. REFERENCES

[1] Tauri blog. `https://tauri.app/about/security`. [Accessed 30-July-2022].

[2] A. Alkhars and W. Mahmoud. Cross-platform desktop development (javafx vs. electron).

[3] Alexander E. Gorbalenya, Susan C. Baker, Ralph S. Baric, Raoul J. de Groot, Christian Drosten, Anastasia A. Gulyaeva, Bart L. Haagmans, Chris Lauber, Andrey M. Leontovich, Benjamin W. Neuman, Dmitry Penzar, Stanley Perlman, Leo L. M. Poon, Dmitry V. Samborskiy, Igor A. Sidorov, Isabel Sola, John Ziebuhr, and Coronaviridae Study Group of the International Committee on Taxonomy of Viruses. The species severe acute respiratory syndrome-related coronavirus: classifying 2019-ncov and naming it sars-cov-2. *Nature Microbiology*, 5(4):536–544, Apr 2020.

[4] Elias Müller. Web technologies on the desktop: an early look at flutter, 2021.

[5] E.R. Pernice et al. Application Development in the Face of Evolving Web Technologies at the National Ignition Facility. In *Proc. ICALEPCS'19*, number 17 in International Conference on Accelerator and Large Experimental Physics Control Systems, pages 1052–1056. JACoW Publishing, Geneva, Switzerland, 08 2020. https://doi.org/10.18429/JACoW-ICALEPCS2019-WEMPR006.

[6] Kevin Sawicki. Atom shell is now electron. `https://www.electronjs.org/blog/electron`, Apr 2015. [Accessed 26-May-2022].

[7] Gian Luca Scoccia and Marco Autili. Web frameworks for desktop apps: An exploratory study. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '20, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Cheng Zhao. Release v0.1.0: Update node: use node's implementation of set immediate. `https://github.com/electron/electron/releases/tag/v0.1.0`, Jul 2013. [Accessed 26-May-2022].