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 HTML, 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, claiming 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 small screen cast application is implemented for each framework to analyze and compare the development process.

1. INTRODUCTION

Short introduction why using web-frameworks for building desktop applications. Advantages and Disadvantages compared to classic way. Motivation of comparing Tauri and Electron

2. ELECTRON

Electron originally was released as Atom Shell by GitHub at 15th July, 2013 [2]. The intention of the developers was "to handle the Chromium/Node.js event loop integration and native APIs" [1] for the Atom Editor. On 23rd 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 SCREEN-CAST APPLICATION

This section will include a short explanation of the screencast 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. REFERENCES

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

[2] 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].