BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2024
http://excel.fit.vutbr.cz

Overfloat - an Overlay System for Windows and Linux

Jakub Šediba*

Abstract

While using the computer, many use cases depend on or would greatly benefit from easy access to information from external sources. This often leads to a lack of desktop real estate and the necessity to constantly switch between active windows, especially if only a single monitor is used. My thesis aims to combat this problem, while adding partial automation support, by providing a system for the creation of modular overlay-based extended user interfaces. This system, which is useable on Windows and Linux operating systems, consists of an API that exposes the features to module creators and an app that controls the modules. The created system allows module creators to develop multi-window modules using TypeScript and the React library. The main distinguishing features are the ability to bind actions to hotkeys, which can be triggered even when the module window isn't focused and the ability to monitor folders or files while triggering a callback when changes in them occur. With this system, users can create widget-like applications using web technologies, but not necessarily requiring internet access. All of this can be achieved without having to deal with direct interactions with the operating system.

*xsedib00@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Several different computer use cases greatly benefit from **access to information from external sources** and **automation**. The one that was my biggest inspiration when developing this system was video games. Over the recent years overlay applications offering this kind of functionality, such as Overwolf[1], gained quite a bit of popularity.

My goal was to develop a more approachable system that would allow users to create **multi-window modules** to present information and provide a certain level of automation. These modules can then be freely combined to provide an **extended user interface** displayed in windows floating above the main active window.

The market for these kinds of systems is mainly dominated by the earlier-mentioned Overwolf, which includes a store[2] with a plethora of overlays for many different games. I was however dissatisfied with the lack of a Linux version of this program and the instability and performance issues caused by its use of Electron.js[3]. This led to my choice of developing a cross-platform alternative.

With a system like this, window management support is a necessity. Other than this, I've decided to implement several other features, that make this system useable in more cases. These features include the support for global hotkeys, manipulation of the filesystem and clipboard, input simulation, and monitoring of the filesystem. A profile system that allows users to save multiple window layouts and keybind settings is also included.

2. Architecture

Using a hybrid approach to cross-platform development makes it possible for the modules to be developed using JavaScript and TypeScript, which rank highly on the programming language popularity indexes [1, 2]. This combined with the need for cross-platform compatibility, led me to choose Tauri[4] as a framework for this system.

While Tauri is simlar to Electron.js in principle, it re-

[1] https://www.overwolf.com/pages/homepage/
[2] https://www.overwolf.com/apps/
[3] https://www.electronjs.org/
[4] https://tauri.app/

places of the traditional Node.js[5] runtime environment with a custom Rust-based one [3], which should lead to better performance. The other benefit of Tauri is that it uses platform native browser engines [4] instead of bundling the Chromium core to every installer [5].

While Rust allows developers to use of the Cargo package manager and its library of Crates[6], the comparatively younger ecosystem lacks implementations of key functioanlty that are easily found in Node.js.

The final architecture of the system is displayed in Figure 1. It consists of four main parts: the Rust backend, the main app, the API, and the user-made modules.

The Rust backend is the part that interacts with the operating system. It implements the capturing of global keypresses, filesystem manipulation, input simulation and filesystem monitoring.

The main app handles the routing and profile management. It also holds information about active modules, their windows and shortcuts, that can be bound to hotkeys. The app provides a graphical user interface that allows users to start and stop specific modules, hide or show windows, change profiles and configure bound keys.

The API provides methods to implement the features of this system into modules. Some of these methods interact directly with the Rust backend, while others interact with the module and keybind managers in the app window.

This provides the support foundation that lets the last part of the architecture, the **user-created modules**, shine. These modules are developed in structured directories and consist of a main window and any number of subwindows. The structure of a module directory is illustrated on Figure 2. The windows of a module are represented by React[7] components, which have to be the default exports of the files they are implemented in.

The root of the module directory must contain specifically one TSX file representing the main window. Subwindows are represented by TSX files located in the subwindows subdirectory inside the module directory. All other TSX files must be located in different subdirectories inside the module directory.

This structure allows for **filesystem-based routing** to be used, which means that the creation of modules does not necessitate changes in the code outside of the module itself.

3. Module Examples

Multiple modules were developed to showcase the features of this system. Two of them are shown in Figure 3 and Figure 4.

Figure 3 shows a directory monitoring module. This module is designed to display all of the changes that happen in a selected directory using the filesystem monitoring feature. In this example, a test run of an image-generating program is shown. The notifications are displayed in chronological order from the most to the least recent.

Icons and colours are used to differentiate between different types of events. In this example, you can see file and folder creation and deletion events, file content modification events and file relocation events.

Figure 4 shows a module designed to assist the user with trading in the online game Path of Exile[8]. This module uses the filesystem monitoring feature to get information about incoming trade requests from the chat log of the game. These trades are then displayed in the cards showing the requested items and the items offered in exchange.

These cards also contain buttons to quickly invite the other person into the user's party, open the trade window with them, or cancel the trade request. The user can also bind hotkeys to the actions tied to these buttons for an even smoother experience.

4. Conclusions

Overfloat is an accessible solution for your overlay needs on Windows and Linux operating systems. From displaying the contents of a website in a modified way to automation and filesystem monitoring, I believe a lot of people will find a use for this system. If this system interests you, you can find out more on the GitHub repository[9].

Acknowledgements

I would like to sincerely thank my supervisor Ing. Jiří Hynek Ph.D. for all their help. Their unending willingness to discuss my problems helped me make this system as good as I could.

[5] https://nodejs.org/en
[6] https://crates.io/
[7] https://react.dev/
[8] https://www.pathofexile.com/
[9] https://github.com/jsediba/overfloat

References

[1] PYPL. Pypl index. https://pypl.github.io/PYPL.html, 2024. Accessed on the 19th of June 2024.

[2] TIOBE Software. Tiobe index. https://www.tiobe.com/tiobe-index/, 2024. Accessed on the 19th of June 2024.

[3] Tauri Contributors. Architecture. https://tauri.app/v1/references/architecture/, 2024. Accessed on the 19th of June 2024.

[4] Tauri Contributors. Webview versions. https://tauri.app/v1/references/webview-versions, 2023. Accessed on the 19th of June 2024.

[5] Electron. Electron documentation. https://www.electronjs.org/docs/latest/. Accessed on the 19th of June 2024.