VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE PRO VYTVÁŘENÍ ROZŠÍŘENÝCH UŽIVATELSKÝCH ROZHRANÍ POMOCÍ PLOVOUCÍCH OKEN

APPLICATION FOR CREATING EXTENDED USER INTERFACES USING FLOATING WINDOWS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB ŠEDIBA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JIŘÍ HYNEK, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

154350

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Šediba Jakub**
Program:	Informační technologie
Název:	**Aplikace pro vytváření rozšířených uživatelských rozhraní pomocí plovoucích oken**
Kategorie:	Webové aplikace
Akademický rok:	2023/24

Zadání:

1. Prostudujte problematiku tvorby multiplaformních desktopových aplikací a technologie určené pro tento účel.
2. Nastudujte teorii spojenou s problematikou plovoucích oken a vytváření externích uživatelských rozhraní pomocí plovoucích oken.
3. Analyzujte požadavky na vytvoření multiplatformní aplikace pro export pokročilých uživatelských rozhraní v plovoucích oknech s možností automatizace úloh.
4. Navrhněte multiplatformní aplikaci pro požadavky z bodu 3.
5. Po konzultaci s vedoucím navrženou aplikaci implementujte.
6. Otestujte implementované řešení vytvořením několika modulů uživatelského rozhraní.

Literatura:

- Johnson, J. (2010). *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. Morgan Kaufmann Publishers/Elsevier.
- Scoccia, G. L., & Autili, M. (2020). Web frameworks for desktop apps: an exploratory study. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (s. 1-6).
- Tauri Contributors. (2023). Tauri Guide. Dostupné z: https://tauri.app/v1/guides/

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Hynek Jiří, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	30.10.2023

Abstrakt

Táto práca rieši problémy spojené s potrebou prepínania aktívnych okien pri práci vyžadujúcej informácie z externých zdrojov. Tieto problémy sú adresované vytvorením modulárnej multiplatformovej aplikácie pre operačné systémy Windows a Linux, ktorá umožňuje používateľom vytvárať rozšírené používateľské rozhrania zobrazené v plávajúcich oknách. Používatelia aplikácie potom môžu použiť kombináciu týchto modulov na vytvorenie rozšíreného používateľského rozhrania. Vytvorená aplikácia ponúka možnosť vytvárať moduly pozostávajúce z niekoľkých okien, v ktorých sú zobrazené požadované informácie. Okrem tejto funkcionality vytvorená aplikácia ponúka úroveň automatizácie dosiahnutú podporou vytvárania globálnych klávesových skratiek, monitorovania súborového systému a možnosti simulovať vstupy myši a klávesnice. Vyvinutá aplikácie používa aplikačný rámec Tauri a kombináciu webových technológií s programovacím jazykom Rust.

Abstract

This thesis aims to address problems related to the necessity to often switch active windows to access information from external sources while working. These problems are addressed by creating a modular cross-platform application for Windows and Linux operating systems. This application allows users to develop multi-window modules which can then be combined to create a desired extended user interface overlaid above other windows. Outside of this functionality, access to automation through global keyboard shortcuts, file-system monitoring and input device simulation is provided. The developed application uses the Tauri framework and a combination of web technologies with the Rust programming language.

Kľúčové slová

Multiplatformová aplikácia, rozšírené používateľské rozhrania, modulárna aplikácia, desktopová webová aplikácia, plávajúce okná, automatizácia, webové technológie, Tauri, Rust

Keywords

Cross-platform Application, Extended User Interfaces, Modular Appliaction, Desktop Web Application, Floating Windows, Automation, Web Technologies, Tauri, Rust

Citácia

ŠEDIBA, Jakub. *Aplikace pro vytváření rozšířených uživatelských rozhraní pomocí plovoucích oken*Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Aplikace pro vytváření rozšířených uživatelských rozhraní pomocí plovoucích oken

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jiřího Hynka, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Jakub Šediba
7. mája 2024

Poďakovanie

Rád by som poďakoval môjmu vedúcemu práce pánovi Ing. Jiřímu Hynkovi, Ph.D., za odborné vedenie mojej práce, všetky cenné rady a tipy, ktoré mi poskytol a neustálu dostupnosť a ochotu konzultovať so mnou problémy, na ktoré som pri tejto práci naraził. Taktiež by som chcel poďakovať Andrejovi Ľuptákovi, Kristiánovi Kičinkovi, Janovi Swiatkowskému a Václavovi Valentovi za mentálnu podporu pri písaní tejto bakalárskej práci.

Obsah

Kapitola 1

Úvod

Pri práci vyžadujúcej informácie z externých zdrojov je často potrebné použitie viacerých monitorov alebo časté prepínanie medzi viacerými oknami, čo spôsobuje rozdeľovanie pozornosti. Hlavnou motiváciou tejto bakalárskej práce je umožnenie odstránenia potreby takéhoto prepínania medzi oknami a zjednodušenie práce používateľov.

Na účely takéhoto zjednodušenia práce táto bakalárska práca navrhuje vytvorenie multiplatformovej aplikácie zameranej na zvýšenie produktivity pri práci požadujúcej využitie informácií z externých zdrojov. Cieľovými platformami tejto aplikácie sú operačné systémy Windows[1] a Linux[2]. Podstatou tejto aplikácie je umožnenie vytvorenia modulárneho rozšíreného používateľského rozhrania, ktoré bude umožňovať používateľovi tieto externé zdroje zobraziť v plávajúcich oknách nad primárnou aplikáciou.

Samotné moduly budú vytvárané používateľmi. Tieto moduly budú poskytovať možnosť využitia dodatočnej funkcionality, akou je napríklad možnosť nastavenia klávesových skratiek pre automatické kopírovanie informácií z externého zdroja alebo automatizované ovládanie myši a klávesnice. Využitie tejto aplikácie bude v podstate obmedzené len kreativitou autorov samostatných modulov. Konkrétnym použitím môže byť napríklad čiastočná automatizácia pri vyplňovaní obsahu tabuľky dátami z webu, získavanie stratégií počas hrania počítačových hier alebo možnosť spísania úloh a postupného značenia splnenia ich častí.

Ďalšou podstatnou časťou tejto aplikácie je možnosť vytvárania prednastavených rozložení modulov pre rôzne aktivity. Toto umožní používateľovi prednastaviť si rozloženie a funkcionalitu externého používateľského rozhrania pre rôzne typy práce a jednoducho prepínať medzi týmito rozloženiami. Problematika vytvárania takejto aplikácie je spojená hlavne s problematikou vytvárania multiplatformových aplikácií, problematikou vykresľovania okien a problematikou správcov okien.

Kapitola 2 rozoberá tvorbu multiplatformových aplikácií a rôzne prístupy k nej. Kapitola 3 je zameraná na vykresľovanie grafických elementov aplikácií a ich interakciu s rôznymi typmi správcov okien. Kapitola 4 obsahuje analýzu existujúcich riešení a častí potrebných pre vytvorenie požadovanej aplikácie. Kapitola 5 popisuje návrh požadovanej aplikácie. Kapitola 6 hovorí o spôsoboch, ktorými boli implementované jednotlivé časti z návrhu aplikácie. Kapitola 7 približuje spôsob, ktorým bola implementovaná aplikácia testovaná.

[1] https://www.microsoft.com/en-us/windows
[2] https://www.linux.org/

Kapitola 2

Multiplatformové aplikácie

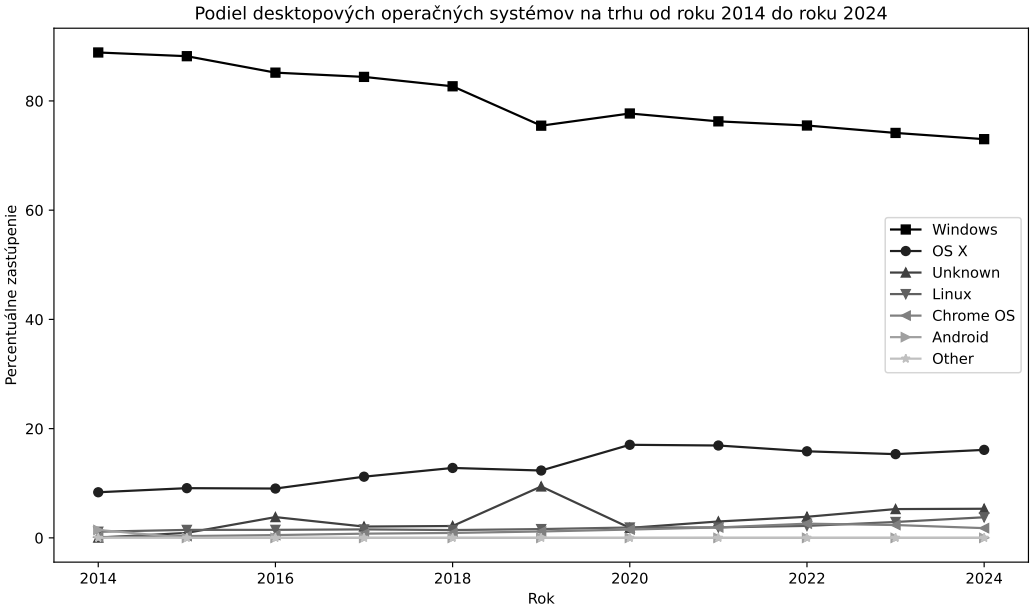
Multiplatformový softvér je podľa [10] definovaný ako softvér vyvinutý pre viac ako jeden operačný systém. Niektoré zdroje predchádzajúcu definíciu mierne upravujú, a ako multiplatformový softvér označujú softvér, ktorý je možné spustiť na dvoch alebo viacerých hardvérových platformách. Takto upravené kritériá však nie sú bežné a ako multiplatformovú aplikáciu je teda možné považovať aplikáciu vyvinutú pre viacero operačných systémov.

Historicky bola problematika vytvárania softvéru pre viacero platforiem podstatná už na prelome sedemdesiatych a osemdesiatych rokov [15]. V tejto dobe bolo často potrebné špecificky modifikovať softvér pre každý systém. Tento prístup bol však veľmi náročný z viacerých uhľov pohľadu. Neskôr sa s rastom popularity programovacích jazykov vyššej úrovne, ako napríklad C++[1], začali objavovať nové prístupy ako písať kód pre viaceré platformy jednoduchšie. Obrovským míľnikom vo vývoji aplikácií pre viacero platforiem bol vznik programovacieho jazyka Java[2] v roku 1995. Podľa [2] jazyk Java vznikol na myšlienke „Write once, run anywhere“ (v preklade „napíš raz, spusti kdekoľvek“), čo bolo dosiahnuté prekladom programu na sadu inštrukcií (*Bytecode*), ktoré sú následne interpretované virtuálnym strojom JVM.

V modernej dobe sú v istých prípadoch podobné prístupy k vytváraniu multiplatformových aplikácií stále používané. Vytváranie natívnej verzie aplikácie pre každú podporovanú platformu je stále používané najmä pri aplikáciách požadujúcich veľmi vysoký výkon na úkor vyššej ceny vývoja. V druhej dekáde dvadsiateho prvého storočia však vznikol čoraz viac populárny prístup ku vývoju multiplatformových aplikácií. Jedná sa o použitie kombinácie behového prostredia (*runtime environment*) ako napríklad Node.js[3] a knižníc ako React[4], čo umožňuje vytváranie aplikácií pre viacero platforiem použitím webových technológií. Tieto nástroje kladú dôraz na jednoduchú prenositeľnosť a vývoj na úkor nižšieho výkonu oproti natívnym aplikáciám.

So zmenami podielu zastúpenia desktopových operačných systémov na trhu v posledných rokoch rastie potreba pre vývoj aplikácií pre viacero platforiem. Z grafu zobrazenom na obrázku 2.1 je viditeľná rastúca popularita predtým menej používaných operačných systémov. Týmto zvýšením diverzity dochádza ku situácii, kedy používateľ často nemôže jeho preferované aplikácie použiť na jeho preferovanom operačnom systéme. Pri vývoji nových aplikácií je teda veľmi dôležité zohľadniť túto skutočnosť a vynaložiť snahu o umožnenie používania aplikácie čo najväčšiemu množstvu ľudí.

[1] https://isocpp.org/
[2] https://www.java.com/en/
[3] https://nodejs.org/
[4] https://react.dev/



Obr. 2.1: Podiel desktopových operačných systémov na trhu v rokoch 2014 až 2024 ukazuje pokles popularity operačného systému Windows oproti alternatívam. Najvyššie zastúpenie majú operačné systémy Windows, OS X, Chrome OS a Linux. Dáta sú z januára konkrétnych rokov a sú boli získané z webovej stránky StatCounter[6].

2.1 Vývoj multiplatformových aplikácií

Pri štandardnom vývoji aplikácií je zvolená cieľová platforma, pre ktorú je aplikácia vyvíjaná. Tento spôsob vývoja je často nazývaný aj vývoj *natívnych* aplikácií. Vývoj natívnych aplikácií umožňuje úzku spoluprácu s prostriedkami dostupnými na danej platforme, často pomocou aplikačno-programovacích rozhraní (*API*). Tieto API poskytujú vývojárom priamočiary spôsob, ktorým môžu použiť zdroje platformy. Veľké množstvo API je však závislé na špecifickej cieľovej platforme.

Použitie platformovo špecifických API spôsobí problém, ak sa vývojári rozhodnú preniesť vytvorenú aplikáciu na inú platformu. Publikácia [3] diskutuje prechod od vývoja natívnych aplikácií ku vývoju multiplatformových aplikácií. Autori tejto publikácie sa zamerali na spôsoby, ktorými je možné zjednodušiť vývoj aplikácií pre rôzne platformy. Ako podstatnú časť problému prenositeľnosti aplikácií uvádzajú tvorbu grafických užívateľských rozhraní (*GUI*), ktorá pri tvorbe sady natívnych aplikácií pre rôzne platformy často spôsobuje potrebu udržiavania viacerých verzií zdrojového kódu. Pre zníženie množstva platformovo špecifického kódu autori odporúčajú jasne definovať rozdelenie logickej a zobrazovacej časti aplikácie. Toto vedie k umožneniu znovupoužitia čo najväčšej časti kódu. Existujú 3 hlavné prístupy k vývoju multiplatformových aplikácií a to: *natívny prístup*, *webový prístup* a *vývoj desktopových webových aplikácií*.

[6] https://gs.statcounter.com/

2.1.1 Natívny prístup

Tento prístup k vývoju multiplatformových aplikácií vyplýva zo štandardného postupu vývoja aplikácií pre špecifickú platformu. Publikácia [19] hovorí, že prostriedky pre vývoj multiplatformových desktopových aplikácií sú oproti iným sféram multiplatformového vývoja pomerne vyspelé. Existuje veľké množstvo kompilátorov rôznych programovacích jazykov pre rozličné populárne operačné systémy. Toto umožňuje aplikácie napísané v jednom programovacom jazyku kompilovať na aplikácie spustiteľné na rôznych operačných systémoch. Problém nastáva pri potrebe použitia platformovo špecifických API pre vývoj GUI.

Základným spôsobom multiplatformového vývoja aplikácií použitím natívneho prístupu je vytváranie samostatných rozdielnych verzií aplikácie pre rôzne cieľové platformy. Toto spôsobuje veľké množstvo duplicitného kódu, a preto je vhodné použiť separáciu logiky a zobrazovania odporúčanú v publikácií [3]. Pri použití takejto separácie je možné vytvoriť jednu centrálnu časť kódu obsahujúcu logiku aplikácie a pomocou definovaného rozhrania dodať rozdielny kód GUI pre rôzne platformy. Kombinácie týchto kódov platformovo špecifických GUI a zdieľaného kódu logiky aplikácie sú potom kompilované pomocou rôznych kompilátorov na niekoľko verzií aplikácií, z ktorých každá funguje práve na špecifickej platforme, pre ktorú bol použitý kód GUI určený.

Ďalším krokom k odstráneniu platformovo špecifických častí kódu je použitie medziplatformových sád nástrojov pre vývoj grafických rozhraní (*GUI Toolkit*) ako napríklad GTK[7] a Qt[8]. Tieto sady nástrojov poskytujú abstrakciu nad prácou s grafickými primitívami závislou na cieľovej platforme. Ďalšou časťou týchto sád nástrojov sú implementácie pre špecifické platformy, ktoré zaručujú samotné vykresľovanie takto napísaného kódu. Zatiaľ čo použitím tejto technológie odpadá nutnosť udržiavania viacerých verzií aplikácie, voľba správnej sady sa stáva kritickým problémom pri vývoji. Je totižto nutné zvoliť sadu ktorá podporuje vývojármi preferovaný programovací jazyk a poskytuje implementácie pre všetky cieľové platformy.

Ku podobnej redukcii redundantného platformovo závislého kódu vedie použitie interpretovaných jazykov ako napríklad Java. Takéto programovacie jazyky poskytujú časti API, ktoré pracujú priamo s abstrakciou platformovo závislých častí zariadení. Samotné vykonávanie týchto platformovo závislých častí potom prebieha vo virtuálnom zariadení. Programy napísané takýmto spôsobom sú interpretované virtuálnym strojom, čo okrem možnosti spúšťania programov na všetkých platformách, pre ktoré je daný virtuálny stroj vyvinutý, poskytuje aj vyššiu bezpečnosť výsledných aplikácií. Nevýhodou sa ale stáva potreba použitia špecifických programovacích jazykov a znížený výkon spôsobený samotnou interpretáciou kódu spomenutými virtuálnymi strojmi.

Natívny prístup ku vývoju aplikácií sa dá teda rozdeliť na dve hlavné metodiky. Prvou z nich je vývoj a kompilácia samostatných verzií aplikácie pre všetky cieľové platformy, s možnosťou použitia multiplatformovej sady nástrojov pre tvorbu grafických rozhraní. Druhou metodikou je použitie kombinácie interpretovaného programovacieho jazyka a virtuálneho interpreta. Pri tejto metodike je samotný kód aplikácie preložený na prechodový kód (v prípade jazyka Java sa jedná o Bytecode), ktorý obsahuje inštrukcie pre virtuálne zariadenie použité na interpretáciu výslednej aplikácie.

[7] https://www.gtk.org/
[8] https://www.qt.io/

2.1.2 Webový prístup

Webový prístup spočíva v použití webových technológií na vývoj multiplatformových aplikácií. Takéto aplikácie sú pre používateľov dostupné na rozličných typoch zariadení a nevyžadujú inštaláciu. Aplikácie tohto typu používajú webový prehliadač. Webové aplikácie používajú model klient-server a sieťovú komunikáciu pomocou protokolov HTTP alebo HTTPs. Požadovaná aplikácia je teda dostupná len v prípade, že používateľ má prístup na internet. Možnosti práce so zdrojmi zariadenia sú obmedzené na zdroje, ktoré sú prístupné cez webový prehliadač.

Model klient-server spočíva v použití dvoch typov zariadení: *serveru* a *klienta*. Komunikácia medzi týmito zariadeniami prebieha cez sieť. Štandardne sa na túto komunikáciu používajú protokoly HTTP alebo HTTPs, pri ktorých klient posiela požiadavky na server a server na tieto požiadavky odpovedá.

- Server je zariadenie zvyčajne pod správou autorov aplikácie. Toto zariadenie má prístup k prostriedkom potrebným pre funkčnosť aplikácie. Jedná sa napríklad o prístup do databáze. Server spracúva požiadavky získané od klientov a odosiela odpovede na ne. Server často vykonáva veľkú časť výpočtov potrebných pre používanie aplikácie.

- Klient je zariadenie, ktoré pristupuje k aplikácii. Klientské zariadenie získava dáta od serveru, tieto dáta zobrazuje a umožňuje používateľovi pracovať s nimi. Toto zariadenie posiela serveru požiadavky na akcie, ktoré majú byť vykonané a čaká na odpovede od servera.

Architektúru webových aplikácií je možné rozdeliť na dve hlavné časti: *backend* a *frontend*. Tieto dve časti aplikácie reflektujú model klient-server. Backend je vykonávaný serverom, zatiaľ čo frontend je vykonávaný klientským zariadením.

- Backend je podľa [4] definovaný ako časť webovej stránky alebo softvéru, ktorý používateľ nevidí. Backend a frontend spolu komunikujú a ich kombináciou vzniká výsledný systém alebo webová stránka. Backend spracúva vstupy, ktoré sú od používateľa získavané časťou fronted.

- Frontend je podľa [5] definovaný ako časť webovej stránky alebo softvéru, s ktorou používateľ pracuje. Z pohľadu používateľa môže byť frontend synonymom s používateľským rozhraním. Frontend používateľovi zobrazuje dáta a získava od neho vstupy.

Okolo roku 2015 sa začal používať pojem *progresívne webové aplikácie* [11]. Tento pojem označuje nový typ webových aplikácií, ktoré poskytujú rozsiahlejšiu funkcionalitu. Takéto aplikácie umožňujú prístup k viacerým hardvérovým zdrojom klientských zariadení, možnosť používať aplikáciu aj bez prístupu k internetu a mnohé ďalšie možnosti. Progresívne webové aplikácie môžu fungovať vo webovom prehliadači ako aj štandardné webové aplikácie, ale taktiež je možné ich nainštalovať na cieľovom zariadení. Spoločnosti ako Microsoft[9] sa snažia rozširovať množstvo hardvérových zdrojov prístupných pre tento typ aplikácií. Nové verzie webového prehliadača Microsoft Edge[10] poskytujú napríklad prístup k externým zariadeniam pomocou technológií Bluetooth a USB, možnosť práce so súbormi a hardvérovú akceleráciu grafických výpočtov [9].

[9] https://www.microsoft.com/
[10] https://www.microsoft.com/en-us/edge

2.1.3 Desktopové webové aplikácie

Vývoj desktopových webových aplikácií spočíva v kombinácií aspektov webového a natívneho prístupu. Aplikácie vyvíjané takýmto spôsobom používajú na vývoj webové technológie v kombinácií s použitím aplikačného rámca ako Electron[11], NW.js[12] alebo Tauri[13]. Tieto aplikačné rámce umožňujú vyvíjať multiplatformové aplikácie podobne ako pri použití webového prístupu, ale časti backend aj frontend sú vykonávané jedným zariadením. Toto umožňuje presun všetkých výpočtov na cieľové zariadenie a odstraňuje potrebu existencie centrálneho servera. Použitie týchto aplikačných rámcov taktiež umožňuje plné využitie hardvérových zdrojov cieľového zariadenia.

V publikácii [14] bola skúmaná sada webových desktopových aplikácií. Autori zistili, že týmto spôsobom sú najčastejšie vytvárané vývojové nástroje a aplikácie zamerané na produktivitu. Okrem tohto autori poznamenali aj to, že v porovnaní s natívnym prístupom k vývoju sa vývojové tímy takýchto aplikácií zvyčajne skladajú z menšieho počtu vývojárov.

Pre priblíženie tohto prístupu je vhodné sa pozrieť na architektúru spomenutých aplikačných rámcov. Zaujímavý je hlavne model procesov, ktoré tieto rámce používajú. Táto sekcia sa bude venovať architektúre aplikačného rámca Tauri, ktorý používa kombináciu behového prostredia vyvinutého v programovacom jazyku Rust a jadier natívnych webových prehliadačov cieľových operačných systémov. Použité jadrá webových prehliadačov sú WebView2[14] pre operačný systém Windows a WebKit[15] pre operačné systémy Linux, macOS a iOS [18].

Model procesov rámca Tauri je popísaný v dokumentácii [16]. Použitie jednoprocesovej architektúry by viedlo k problémom, v prípade že by tento proces vykonával časovo náročný výpočet. V tomto prípade by pri použití jednoprocesovej architektúry bolo grafické rozhranie neresponzívne a prípadné zlyhanie tohto procesu by spôsobilo zlyhanie celej aplikácie. Z tohto dôvodu Tauri používa viacprocesovú architektúru podobnú architektúre moderných webových prehliadačov. Aplikačný rámec Electron, ktorý je v súčasnej dobe na tvorbu webových desktopových aplikácií využívaný najviac, používa podobnú viacprocesovú architektúru. Ďalšou výhodou viacprocesovej architektúry je možnosť obmedzenia práv jednotlivých procesov, čo vedie k zvýšeniu bezpečnosti výslednej aplikácie. Aplikačný rámec Tauri používa dva typy procesov: *hlavný proces* a *procesy WebView*. Medzi týmito dvoma typmi procesov prebieha medziprocesová komunikácia.

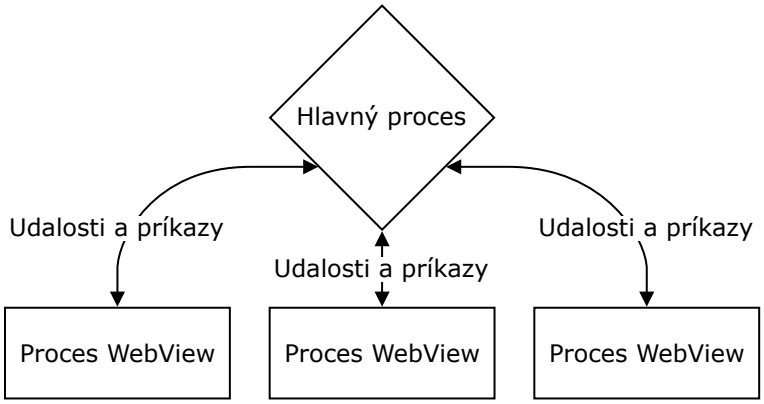
Každá aplikácia má práve jeden hlavný proces, ktorý slúži ako vstupný bod do nej. Tento proces má ako jediný plný prístup k operačnému systému. Úlohou tohto procesu je vytváranie a spravovanie aplikačných okien, ponúk systémovej lišty a notifikácií. Tauri používa na implementáciu hlavného procesu programovací jazyk Rust kvôli jeho konceptu vlastníctva, ktorý zaručuje bezpečnosť pamäte. Tento proces by mal taktiež ako jediný pracovať so senzitívnymi dátami, ktorými sú napríklad prístupové údaje do databáze. Hlavný proces môže definovať špeciálne označené funkcie nazývané Tauri príkaz (*Tauri command*), ktoré môžu byť spúšťané z procesov WebView. Tento proces nevykresľuje samotné grafické používateľské rozhrania. Hlavný proces predstavuje backend výslednej aplikácie.

Procesy WebView slúžia práve na vykresľovanie grafických používateľských rozhraní. Tento typ procesu používa prostredie podobné štandardným webovým prehliadačom, ktoré pracuje s jazykmi HTML, CSS a JavaScript. Toto umožňuje vytváranie grafických rozhraní

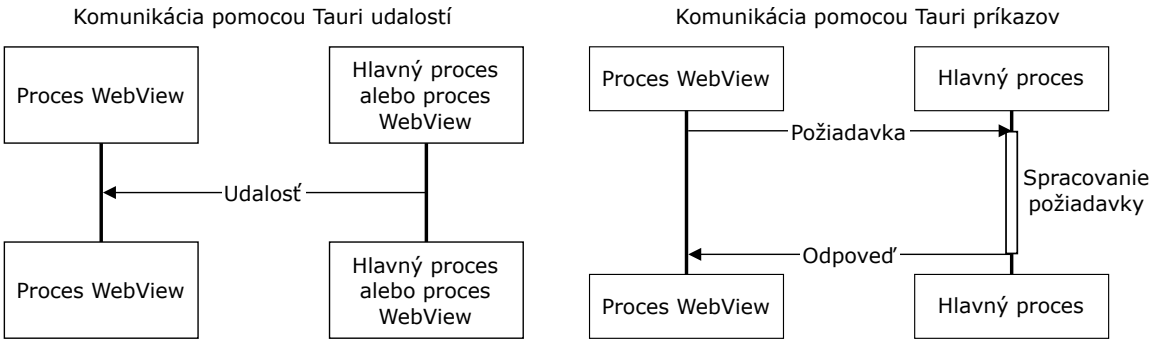
[11] https://www.electronjs.org/
[12] https://nwjs.io/
[13] https://tauri.app/
[14] https://developer.microsoft.com/en-us/microsoft-edge/webview2/
[15] https://webkit.org/

pomocou prístupov používaných pri tvorbe webových aplikácií. Pri tvorbe grafických užívateľských rozhraní je teda možné použiť napríklad knižnicu React a zväzovač (*bundler*) Vite[16]. Procesy WebView predstavujú frontend výslednej aplikácie.

Medziprocesová komunikácia v aplikačnom rámci Tauri prebieha dvoma spôsobmi. Prvým spôsobom sú už spomenuté Tauri príkazy. Rozhranie pre používanie Tauri príkazov je inšpirované rozhraním `fetch`[17]. Hlavný proces získa požiadavku na vykonanie Tauri príkazu s možnosťou použitia argumentov, túto požiadavku spracuje a následne informuje proces, ktorý požiadavku zaslal o výsledku. Toto zaisťuje komunikáciu inicializovanú procesmi WebView. Hlavný process taktiež umožňuje zasielať Tauri udalosti (*Tauri event*) procesom typu WebView, čo vedie v kombinácií s Tauri príkazmi ku obojsmernej komunikácií zobrazenej na obrázku 2.2. Tauri udalosti je možné odosielať aj z procesov WebView. Rozdielny priebeh komunikácie pomocou Tauri príkazov a Tauri udalostí je zobrazený na obrázku 2.3.



Obr. 2.2: Zjednodušená štruktúra procesov aplikačného rámca Tauri znázorňujúca obojstrannú komunikáciu medzi hlavným procesom a procesmi WebView. Obrázok je prevzatý z [16] a preložený do slovenského jazyka.



Obr. 2.3: Priebeh komunikácie v rámci aplikačného rámca Tauri. Zobrazené sú priebehy komunikácie inicializovanej hlavným procesom alebo procesom WebView pomocou Tauri udalostí a komunikácie inicializovanej procesmi WebView pomocou Tauri príkazov. Obrázok je prevzatý z [17] a preložený do slovenského jazyka.

[16] https://vitejs.dev/
[17] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Kapitola 3

Problematika plávajúcich okien

Problematika vykresľovania aplikácií v plávajúcich oknách je veľmi úzko spojená s zobrazovaním okien. Zobrazovacie systémy hrajú hlavnú rolu v spôsobe, akým sú okná vykresľované a ako s nimi používateľ pracuje. Na pochopenie princípu vykresľovania okien a možnosti vykresľovania plávajúcich okien, teda okien, ktoré sa zobrazujú čiastočne nad inými oknami je teda dôležité porozumieť tomu, ako zobrazovacie systémy fungujú a ich spojitosť so správcami okien.

3.1 Zobrazovacie systémy

Zobrazovací systém je softvér, ktorý poskytuje grafický model vhodný pre vytváranie a prezentovanie grafických používateľských rozhraní. Hlavnou úlohou zobrazovacieho systému je prideľovanie miesta obrazovky grafickým častiam spustených aplikácií. Tieto miesta sú najčastejšie prideľované ako obdĺžniky nazývané *okná* s možnosťou zmeny veľkosti a polohy na obrazovke. Aplikácie potom používajú tieto okná na vykreslenie svojich grafických používateľských rozhraní. Zobrazovací systém taktiež pracuje s jadrom operačného systému na získavanie vstupov zo vstupných zariadení a ich smerovanie do správnych aplikácií.

V súčasnej dobe je v populárnych desktopových operačných systémoch používaných niekoľko zobrazovacích systémov. Operačné systémy podobné Unix najčastejšie používajú zobrazovacie systémy X11 alebo Wayland. Operačný systém Windows používa od verzie Windows Vista zobrazovací systém založený na spolupráci grafického ovládača Windows Display Driver Model (WDDM) a kompozičného správcu okien Desktop Window Manager (DWM). XOS používa zobrazovací systém XQuartz založený na zobrazovacom systéme X11.

Zobrazovací systém X je popísaný v publikácií [13]. Zobrazovací systém X, nazývaný aj X11, vznikol v júne 1987. Tento zobrazovací systém bol pôvodne vyvinutý pre univerzitné prostredie. Autori uvádzajú 9 podmienok, ktoré by mal zobrazovací systém pre univerzitné prostredie spĺňovať. Tieto podmienky sú pri zobrazovacích systémoch používaných v prostrediach s veľkým množstvom spolupracujúcich zariadení aplikovateľné aj v dnešnej dobe. Zatiaľ čo niektoré z týchto podmienok boli stanovené pred začiatkom práce na tomto systéme, iné boli stanovené až počas vývoja. Dodržanie týchto podmienok má viesť k vytvoreniu rozsiahleho univerzálneho zobrazovacieho systému. Tieto podmienky sú nasledovné:

- Systém musí byť implementovateľný pre rôzne typy zobrazovacích zariadení. Táto podmienka je zameraná na univerzálnosť použitia zobrazovacieho systému. Systém by mal fungovať na skoro každom displeji používajúcom mapu bitov na zobrazovanie obsahu. Táto podmienka je taktiež rozšírená o nutnosť podpory rozličných vstup-

ných zariadení, ktorými sú napríklad myši, klávesnice, tablety, dotykové obrazovky a svetelné perá.

- Aplikácie musia byť nezávislé na zariadení. Kvôli veľkému množstvu a rôznorodosti hardvérových zariadení je potrebné, aby aplikácie nebolo nutné programovať, kompilovať a linkovať pre každý hardvérový displej osobitne. Toto by totiž mohlo viesť k nekonzistentnému zobrazeniu grafického používateľského rozhrania medzi rôznymi typmi displejov a viedlo by k potrebe hlbokej analýzy každej aplikácie. Táto podmienka taktiež špecifikuje, že zobrazovanie na farebných a čierno-bielych displejoch by malo používať rovnaké riadenie.

- Systém musí byť sieťovo transparentný. To znamená, že aplikácia spustená na jednom zariadení by mala mať možnosť použitia displejov iných zariadení aj v prípade, že tieto zariadenia majú inú architektúru alebo operačný systém. Takáto komunikácia po sieti nesmie blokovať iné komunikácie zariadení. V prostredí s veľkou rôznorodosťou používaných zariadení je možné, že nie všetky používané zariadenia budú podporovať špecifický programovací jazyk. Toto by viedlo k potrebe vytvárania viacerých interaktívnych používateľských rozhraní. Použitie sieťovo transparentného prenosu túto potrebu odstráni.

- Systém musí podporovať viaceré aplikácie zobrazujúce obsah súčasne. Táto podmienka je pomerne priamočiara. Jedná sa o to, že beh jednej aplikácie by nemal blokovať behy ostatných aplikácií a teda by malo byť možné aktualizovať obsah okien viacerých aplikácií súčasne.

- Systém by mal byť schopný podpory rôznych aplikácií a správcovských rozhraní. Potreba tejto podmienky spočíva v rozdielnych preferenciách jednotlivých používateľov. Rozliční používatelia môžu preferovať rozličné správcovské rozhrania a zobrazovací systém by nemal obmedzovať ich voľbu. Na splnenie tejto podmienky je teda potrebné, aby systém poskytoval mechanizmy na prepojenie s rôznymi správcovskými rozhraniami. Aplikácie by mali na zmeny reagovať a nie ich spravovať. Napríklad ak je zmenená veľkosť zobrazovacieho okna, používateľské rozhranie by malo byť prekreslené bez toho, aby toto prekreslenie musela inicializovať samotná aplikácia.

- Systém musí podporovať prekrývajúce sa okná vrátane podpory zmeny obsahu čiastočne prekrytých okien. Aj keď nie všetky používateľské rozhrania podporujú prekrývanie okien, typicky podporujú aspoň nejakú formu zobrazovania ponúk, ktoré môžu prekrývať hlavné okná aplikácie. Keďže tieto ponuky sú taktiež vykresľované ako okná, je nutnosť takejto podpory samozrejmá.

- Systém by mal podporovať hierarchiu okien s možnosťou zmeny veľkosti a aplikácia by mala mať možnosť používať viacero okien naraz. Niektoré aplikácie používajú vlastný systém okien nad hlavným zobrazovacím systémom používaným operačným systémom. Výsledný zobrazovací systém by mal podporovať hierarchickú organizáciu okien, aby takto vykreslené aplikačné okná mohli existovať ako skutočné okná z pohľadu zobrazovacieho systému. Jedná sa o úroveň abstrakcie, ktorá spôsobí to, že obsah, ktorý sa na vyššej úrovni tvári ako obsah jedného okna, môže byť na nižšej úrovni zobrazený vo viacerých podoknách.

- Systém by mal podporovať vysoko-výkonnú a kvalitnú podporu pre text, 2D syntetickú grafiku a obrázky. Zatiaľ čo špecifikáciu elementu požadovaného na zobrazenie

poskytne aplikácia, zodpovednosť za presné a rýchle vykreslenie daného elementu spadá na zobrazovací systém. Podpora 3D grafiky nie je v tomto bode vyžadovaná, ale je pripomenutá jej dôležitosť. Ako dôvody jej absencie autori uviedli nedostatok času a odbornej znalosti.

- Systém by mal byť rozšíriteľný. Jedná sa o vlastnosť, ktorá by mala poskytnúť komunitám možnosť vytvárať rozšírenia pre výsledný systém bez spolupráce s autormi systému. Príkladom takéhoto rozšírenia môže byť napríklad podpora 3D grafiky spomenutá v predchádzajúcej podmienke. Takto vytvorené rozšírenia by mali byť jednoducho spojiteľné so zvyškom systému bez potreby zmien v iných častiach systému.

Zobrazovací systém X je založený na architektúre klient-server. Pre každý fyzický displej existuje ovládací server (takzvaný *X server*). Klientská aplikácia (takzvaný *X klient*) komunikuje s X serverom buď pomocou prúdovej sieťovej komunikácie v prípade vzdialeného displeja alebo pomocou medziprocesovej komunikácie v prípade, že X klient aj X server sú spustené na tom istom zariadení. K jednému X serveru môžu byť naraz pripojený viacerí X klienti a zároveň každý X klient môže byť súčasne pripojený k viacerým X serverom. Ilustráciu systémovej štruktúry zobrazovacieho systému X11 je možné vidieť na obrázku 3.1.

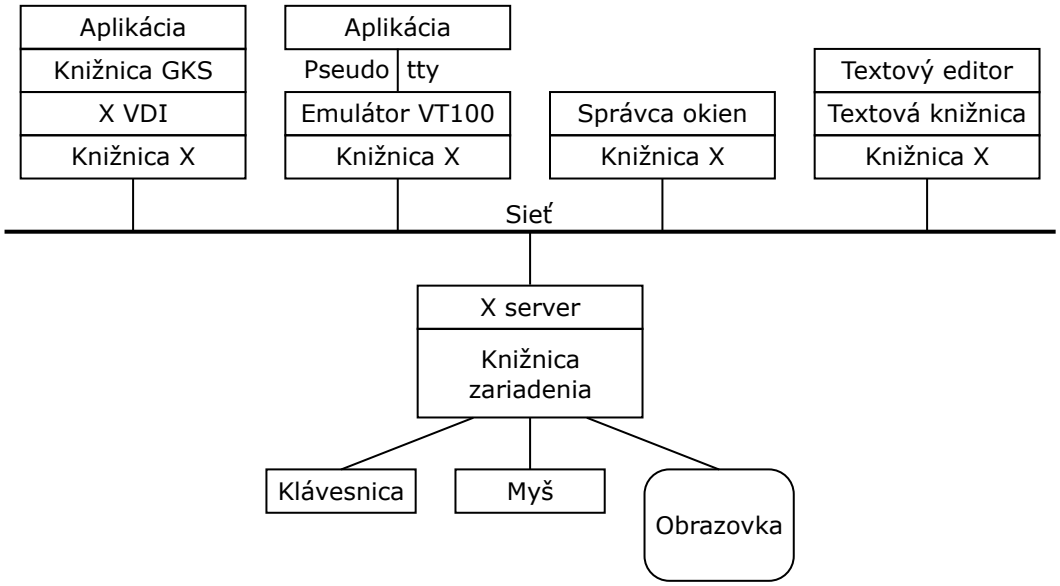
X server spravuje zobrazovacie a vstupné zariadenia, a je to jediná časť X11, ktorá komunikuje s hardvérom cez jadro systému. X klient komunikuje s X serverom pomocou špecifického komunikačného protokolu (takzvaného *X protokolu*). Správca okien zabezpečuje samotné zobrazovanie okien. X klient si od X serveru môže vyžiadať získavanie vstupov zo špecifický vstupných zariadení a X server mu potom tieto vstupy preposiela pomocou X protokolu. X klient spravuje hierarchickú štruktúru okien a používa správcu okien na vykreslenie okien na obrazovke.

Wayland je zobrazovací protokol, ktorý bol vyvinutý ako náhrada zobrazovacieho systému X11 a je popísaný v dokumentácií [6]. Architektúra systému Wayland je podobná architektúre systému X11, avšak nie je v nej používaný správca okien. Serverová časť systému Wayland, nazývaná *Wayland Compositor*, preberá úlohu správcu úloh. Wayland Compositor komunikuje s jadrom a získava informácie o vstupe, ktoré potom posiela cieľovým oknám, ktoré spravuje. Táto časť systému v sebe uchováva grafy scén, pomocou ktorých si uchováva informácie o tom, čo je zobrazované na displejoch. Wayland Compositor potom preposiela informácie o udalostiach klientskej časti systému Wayland. Klientská časť v prípade zmien zobrazenia prekresluje potrebné okná.

Zobrazovací systém používaný v operačnom systéme Windows od verzie Windows Vista funguje veľmi rozdielnym spôsobom, popísaným v dokumentáciách [7, 8]. Tento systém je založený na kompozičnom správcovi okien Windows Desktop Manager (DWM), ktorý spolupracuje s grafickým ovládačom Windows Display Driver Model (WDDM). DWM používa kompozítor Media Integration Layer (MIL) [12], ktorý komunikuje priamo s DirectX na získanie grafických primitív. MIL je použité na udržiavanie kompozičných stromov jednotlivých okien a tieto okná sú potom premietané na jednu grafickú plochu.

3.2 Správcovia okien

Správca okien je softvér, ktorý pracuje so zobrazovacím systémom a preberá kontrolu nad zobrazovacou plochou. Správcovia okien často poskytujú používateľovi rozšírené možnosti práce s oknami ako napríklad maximalizovanie a minimalizovanie okien, presun okien a precíznu zmenu veľkosti okien. Okrem týchto vlastností, niektoré z týchto systémov poskytujú



Obr. 3.1: Štruktúra zobrazovacieho systému X11 prevzatá z [13] a preložená do slovenského jazyka. Na tomto obrázku sú zobrazené rôzne spôsobmi, akými môžu aplikácie pracovať so zobrazovacím systémom X a aké zdroje sú pre túto komunikáciu používané.

používateľovi aj ďalšiu funkcionalitu ako napríklad stavové lišty, možnosť zmeny pozadia plochy a možnosť umiestnenia ikon na plochu. Ako bolo spomenuté vyššie, niektoré zobrazovacie systémy používajú pevne integrovaného správcu okien, zatiaľ čo iné, ako napríklad zobrazovací systém X11, umožňujú použitie ľubovolného správcu okien, ktorý správne pracuje s rozhraniami daného zobrazovacieho systému. Je možné aj súčasné použitie niekoľkých správcov okien, ale kvôli konfliktom táto vlastnosť nie je často používaná. Správcovia okien často poskytujú rozšírenú konfiguráciu pre konkrétne okná. Jedná sa napríklad o možnosť udržať okno na vrchu obrazovky, ktorá je kritická pre prácu s plávajúcimi oknami.

Potreby a preferencie rôznych používateľov môžu byť výrazne rozdielne, čo vedie k existencií rôznych typov správcov okien. Každý z týchto typov správcov okien funguje iným spôsobom a používateľovi poskytuje iné možnosti zobrazenia. Stránky distribúcie Arch Linux [1] delia správcov okien na 3 typy: *správcov plávajúcich okien*, *správcov dlaždicových okien* a *dynamických správcov okien*. Skupina správcov plávajúcich okien zahrňuje aj podskupinu *kompozičných správcov okien*.

3.2.1 Správcovia plávajúcich okien

Tento typ správcov okien je založený na koncepte voľne presúvateľných okien s možnosťou zmeny ich veľkosti. Podstatnou vlastnosťou tohto typu správcov okien je možnosť prekrývania okien. Správcovia plávajúcich okien nepoužívajú žiadne preddefinované rozloženie okien. Okná takýchto správcov okien je možné metaforicky porovnať s listami papieru na stole. Zatiaľ čo okná sa môžu prekrývať, len najvyššie umiestnené okno je plne viditeľné (ak neprechádza za hranice obrazovky).

Správcovia plávajúcich okien používajú jednu vyrovnávaciu pamäť na ukladanie obsahu celej obrazovky. Na vykresľovanie okien na plochu je typicky používaný maliarov algoritmus, ktorý simuluje prístup používaný maliarmi pri tvorbe obrazov. Podstata tohto algoritmu spočíva vo vykreslení okien umiestnených nižšie pred oknami umiestnenými vyššie. Toto

vedie k prekresleniu častí nižších okien časťami vyšších okien. Použitie jednej vyrovnávacej pamäte spôsobuje nutnosť prekreslenia obsahu okna aplikácie pri zmene polohy okien. Toto vedie k zníženému výkonu tohto typu správcov okien. Príkladmi správcov okien tohto typu sú Openbox[1] a Fvwm[2].

3.2.2 Kompoziční správcovia okien

Kompoziční správcovia okien, nazývaní aj *kompozítory*, rozširujú architektúru správcov plávajúcich okien o osobitnú vyrovnávaciu pamäť pre každé okno. Tento typ správcov okien predstavuje podskupinu správcov plávajúcich okien a teda podobne umožňuje voľné presúvanie okien s možnosťou prekrývania a zmeny veľkosti okien.

Použitie osobitnej vyrovnávacej pamäte pre každé okno v kombinácií s vyrovnávacou pamäťou pre obrazovku umožňuje tejto skupine správcov okien odstrániť potrebu opätovného vykreslenia jednotlivých okien pri presúvaní. Obsah jednotlivých okien je už uložený vo vyrovnávacej pamäti daného okna a pri presune okien stačí len skopírovať obsahy týchto vyrovnávacích pamätí na správnu pozíciu vo vyrovnávacej pamäti obrazovky.

Oddelenie pamäte obsahu okien a pamäte obrazovky taktiež umožňuje tomuto typu správcov okien využívať pokročilejšie grafické efekty pri zobrazovaní okien. Jedná sa napríklad o animácie pri otváraní okien, čiastočnú priehľadnosť okien a možnosť priblíženia časti obrazovky. Okrem týchto grafických efektov sú podporované aj funkcie ako duplikácia okna na viacero obrazoviek. Príkladmi správcov okien tohto typu sú Compiz[3] a xfwm[4].

3.2.3 Dlaždicoví správcovia okien

Tento typ správcov okien používa iný prístup k rozloženiu okien na obrazovke. Tento prístup je založený na špecifických vzoroch, podľa ktorých sú okná pri otvorení rozmiestnené. V čisto dlaždicových správcoch okien nie je možné okná prekrývať a voľne presúvať. Cieľom dlaždicových správcov okien je maximalizovanie využitia zobrazovacej plochy bez prekrytia potenciálne dôležitých informácií. Príkladmi správcov okien tohto typu sú i3[5] a Notion[6].

Existuje veľké množstvo rôznych dlaždicových správcov okien, z ktorých každý môže pracovať iným spôsobom. Štandardne sú týmto typom správcov okien používané dve hlavné techniky dlaždicovania. Jedná sa o dlaždicovanie založené na zozname a dlždicovanie založené na stromovej štruktúre.

Dlaždicovanie založené na zozname používa pre prácu s oknami usporiadaný zoznam. Každému oknu je pridelené unikátne číslo podľa jeho pozície v tomto zozname. Okná sú na obrazovke rozložené na základe vzoru práve podľa týchto unikátnych čísel. Používateľovi je umožnené poradie okien v tomto zozname meniť, čo spôsobuje zmenu rozloženia okien na obrazovke.

Dlaždicovanie založené na stromovej štruktúre funguje na podobnom princípe. Okná sú ukladané v listoch stromovej štruktúry, zatiaľ čo koreň a vnútorné uzli tejto štruktúry reprezentujú delenie obrazovky. Toto delenie môže byť buď vertikálne alebo horizontálne. Štandardne tieto dva typy delenia alternujú medzi úrovňami stromovej štruktúry, ale toto nie je podmienkou. Podobne ako pri dlaždicovaní založenom na zozname, používateľ môže

[1] http://openbox.org/
[2] https://www.fvwm.org/
[3] https://launchpad.net/compiz
[4] https://docs.xfce.org/xfce/xfwm4/start
[5] https://i3wm.org/
[6] https://notionwm.net/

presúvať okná medzi listami stromu, čím mení ich usporiadanie na obrazovke. Výhodou použitia stromovej štruktúry oproti zoznamu je možnosť viac špecifickej úpravy vzoru, podľa ktorého sa obrazovka delí.

Keďže pre použitie plávajúcich okien je nutná podpora čiastočného prekrývania okien inými oknami, aplikácie pracujúce s nimi nie je možné použiť v kombinácií s čisto dlaždicovými správcami okien.

3.2.4 Dynamický správcovia okien

Tento typ správcov okien je založený na kombinácií vlastností správcov plávajúcich okien a dlaždicových správcov okien. Táto kombinácia môže byť dosiahnutá dvoma spôsobmi. Prvým z nich je možnosť prepínania medzi režimami, v ktorých správca okien funguje buď ako správca plávajúcich okien alebo ako dlaždicový správca okien. Druhým spôsobom je použitie prístupov pre automatické rozmiestnenie okien používaných dlaždicovými správcami okien pri prvotnom otvorení okna, ale následné umožnenie presúvania týchto okien s podporou prekrývania. Príkladmi správcov okien tohto typu sú xmonad[7] a dwm[8].

[7] https://xmonad.org/
[8] https://dwm.suckless.org/

Kapitola 4

Analýza

Cieľom tejto bakalárskej práce je poskytnúť cieľovým používateľom možnosť zobrazovať informácie z externých zdrojov v oknách zobrazených nad aktuálne používaným oknom a umožniť istú úroveň automatizácie práce. Toto zníži potrebu prepínania medzi aktívnymi oknami pri práci vyžadujúcej externé informácie. Tento problém je možné adresovať vytvorením aplikácie, ktorá bude takéto informácie zobrazovať a umožňovať automatizáciu práce. Pri analýze potrieb, ktoré musí takáto aplikácia splňovať je nutné analyzovať typy používateľov, pre ktorých je aplikácia určená. Toto umožní formuláciu podmienok, ktoré by výsledné riešenie malo splňovať. Taktiež je potrebné analyzovať existujúce riešenia a definovať nedostatky týchto existujúcich riešení.

4.1 Typy používateľov aplikácie

Používateľov, pre ktorých je aplikácie určená, je možné rozdeliť do dvoch hlavných skupín. Jedná sa o bežných používateľov a vývojárov modulov. Tieto dva typy používateľov nie sú disjunktívne, pretože vývojári modulov často budú aplikáciu používať aj ako bežný používatelia. Delenie na tieto dve skupiny je podstatné hlavne z dôvodu oddelenia používateľov, ktorý nemusia mať veľké skúsenosti v IT sfére.

- *Bežný používatelia* aplikácie sú ľudia, ktorý používajú desktopové zariadenia na prácu alebo rekreáciu. Jedná sa o používateľov, ktorý nevytvárajú nové moduly používateľských rozhraní, ale používajú už vytvorené moduly a ich kombinácie. Takýto používatelia nemusia mať skúsenosti s programovaním a vytváraním softvéru. Potreby týchto používateľov spočívajú v jednoduchosti používania aplikácie a funkcionalite, ktorú budú už vytvorené moduly rozhraní poskytovať.

  Jedna s hlavných potrieb týchto používateľov je spojená so samotným cieľom aplikácie. Jedná sa o možnosť zobrazovania informácií z externých zdrojov v oknách priamo nad hlavnou používanou aplikáciou. Táto možnosť zníži potrebu prepínania aktívnych okien hlavne pri použití nižšieho počtu monitorov. Malo by byť možné vytvoriť a uložiť niekoľko rozložení okien pre rôzne aktivity.

  Druhou významnou potrebou týchto používateľov je možnosť automatizovania istých úkonov, ktoré často vykonávajú. Táto potreba hovorí o možnosti simulovania vstupov klávesnice a myši, automatickej úpravy súborov, a možnosti používania klávesových skratiek na vykonávanie často používaných akcií. Kvôli rozdielnym preferenciám používateľov je tiež potrebné, aby výsledná aplikácia bola použiteľná na viacerých desktopových operačných systémoch.

- *Vývojári modulov* sú ľudia so skúsenosťami v programovaní. Tento typ používateľov je zameraný na samotných tvorcov modulov rozhraní. Potreby takýchto používateľov sú spojené s možnosťami, ktoré aplikácie poskytuje pri vývoji modulov. Výsledná aplikácia by mala uľahčiť vytváranie častí grafických rozhraní zobrazených v plávajúcich oknách. Jedná sa o nahradenie potreby vytvárania komplexných riešení grafických rozhraní, klávesových skratiek a simulácie vstupných zariadení. Vytváranie modulov bude prebiehať programovaním logiky a zobrazenia okien s použitím aplikačno-programovacieho rozhrania poskytujúceho prístup k funkcionalitám aplikácie. Aby bolo vytváranie modulov prístupné čo najväčšiemu množstvu programátorov, malo byť možné použiť štandardné prístupy k programovaniu a nemala by byť vyžadovaná veľká investícia času na učenie sa neznámych programovacích jazykov a návrhových vzorov. Aplikácia by mala umožňovať vytváranie modulov bez zásahu do už existujúceho zdrojového kódu aplikácie.

Na základe tejto definície cieľových používateľov je možné určiť 10 požiadaviek, ktoré by mala cieľová aplikácia splňovať:

1. Jednoduché použitie aplikácie aj pre ľudí bez skúseností s programovaním.
2. Dostupnosť na viacerých operačných systémoch.
3. Možnosť vytvárania modulov bez vstupov od autorov aplikácie.
4. Voľnosť v téme vytváraných modulov.
5. Poskytnutie vývojových nástrojov pre uľahčenie vytvárania modulov.
6. Použitie štandardných známych programovacích jazykov a návrhových vzorov pri vývoji modulov.
7. Možnosť použitia viacerých modulov súčasne.
8. Podpora globálnych klávesových skratiek.
9. Podpora pre ukladanie viacerých *profilov* (rozložení okien modulov a nastavení klávesových skratiek) a možnosť jednoduchého prepínania medzi nimi.
10. Možnosť automatizácie úkonov simulovaním vstupných zariadení a práce so súborovým systémom.

4.2 Existujúce riešenia

Rozšírené používateľské rozhrania zobrazené v plávajúcich oknách je možné v súčasnosti vytvoriť kombináciou aplikácie s grafickým používateľským rozhraním a použitím schopnosti správcov okien udržiavať špecifické okná nad ostatnými. Použitie tohto prístup umožňuje vytvoriť aplikáciu použitím ľubovolných technológií a následne zobraziť okná grafického rozhrania tejto aplikácie ako plávajúce okná. Ako existujúce riešenia tohto problému by teda mohli byť uvedené skoro všetky prístupy k tvorbe aplikácií. Je vhodné sa však zamerať na nástroje, ktoré sú určené pre podobné typy cieľových používateľov a poskytujú aspoň časť požadovanej funkcionality. V tejto sekcii je popísaná existujúca platforma zameraná na vývoj rozšírených používateľských rozhraní pre počítačové hry a softvér zameraný na automatizáciu a simuláciu vstupov s možnosťou vytvárania grafických používateľských rozhraní.

4.2.1 Overwolf

V súčasnosti časť trhu zameranú na hráčov počítačových hier dominuje platforma Overwolf[1]. Táto platforma poskytuje množstvo rozšírených používateľských rozhraní pre rôzne počítačové hry. Aplikácie pre túto platformu sú vytvárané v jazykoch HTML, CSS a JavaScript s použitím upravenej verzie rámca Electron nazývanej Overwolf Electron[2]. V súčasnej dobe je táto platforma dostupná len pre operačný systém Windows. Táto platforma spolupracuje s hernými štúdiami, čo umožňuje vývojárom aplikácií pridať reakcie na udalosti, ktoré nastali v samotnej hre. Pre vytváranie rozšírených používateľských rozhraní pre špecifické počítačové hry táto platforma teda poskytuje veľmi rozsiahle možnosti. Nevýhodou však môže byť nutnosť priamej spolupráce s poskytovateľmi tejto platformy pri vývoji aplikácií pre ňu. Samotný návrh aplikácie musí byť oficiálne schválený poskytovateľmi, čo môže viesť k obmedzeniu rôznorodosti rozhraní vyvinutých pomocou tejto platformy. Druhou nevýhodou je samotné zameranie priamo na počítačové hry, ktoré limituje bázu používateľov aplikácie. Po porovnaní s definovanými podmienkami na cieľovú aplikáciu, toto riešenie nesplňuje body 2, 3, 4 a 9.

4.2.2 AutoHotkey

AutoHotkey[3] je softvér s voľne dostupným zdrojovým kódom pre operačný systém Windows. Tento softvér je zameraný na automatizáciu úloh s podporou globálnych klávesových skratiek. AutoHotkey používa vlastný skriptovací jazyk s možnosťou vytvárania grafických používateľských rozhraní. Skripty vytvorené v tomto jazyku môžu byť spúšťané pomocou interpreta alebo kompilované na osobitne spustiteľné súbory. Kombinácia širokej podpory automatizácie a možnosti vytvárania grafických rozhraní vedie k splneniu veľkej časti požiadaviek cieľových používateľov. Nevýhodou použitia tohto riešenia je, že aplikácie musia byť vyvíjané v skriptovacom jazyku, ktorý tento softvér používa. Unikátnosť tohto skriptovacieho jazyka obmedzuje množstvo vývojárov skúsených s jeho používaním a teda často spôsobuje potrebu investície času na naučenie sa písania kódu v novom jazyku pred začiatkom vývoja. Toto riešenie nesplňuje podmienky 2, 6 a 9 definované pre výslednú aplikáciu.

4.2.3 Zhrnutie

Vytváranie používateľských rozhraní pomocou webových technológií použitých v platforme Overwolf sa javí ako vhodné riešenie pre cieľovú aplikáciu. Podobne je vhodné rozdelenie rôznych aplikácií na balíčky. Funkcionalita automatizácie, ktorú poskytuje softvér AutoHotkey, pokrýva veľkú časť podmienok, ktoré má riešenie analyzovaného problému adresovať. Pri návrhu riešenia je teda vhodné sa inšpirovať návrhovým vzorom, ktorý používa platforma Overwolf a možnosťami automatizácie poskytovanými softvérom AutoHotkey. Analyzované riešenia nepodporujú viaceré operačné systémy, čo je jednou zo stanovených podmienok. Pri návrhu je potrebné túto podporu pridať.

[1] https://www.overwolf.com/
[2] https://www.npmjs.com/package/@overwolf/ow-electron
[3] https://www.autohotkey.com/

Kapitola 5

Návrh aplikácie

Analyzované problémy je možné riešiť viacerými spôsobmi. Zvoleným spôsobom bolo vytvorenie modulárnej multiplatformovej desktopovej aplikácie. Tento spôsob umožní vyvíjanie rozšírených používateľských rozhraní v ucelených moduloch, ktoré sú následne kombinované. Tieto moduly potom môžu byť ovládané z jedného okna aplikácie. Je potrebné klásť vysoký dôraz na možnosť vytvárania modulov a umožniť používateľom tieto modely vytvárať čo možno najjednoduchšie. Toto vedie k možnosti vytvorenia kombinácie samotnej aplikácie pre spúšťanie modulov a aplikačno-programovacieho rozhrania (API) pre vytváranie modulov.

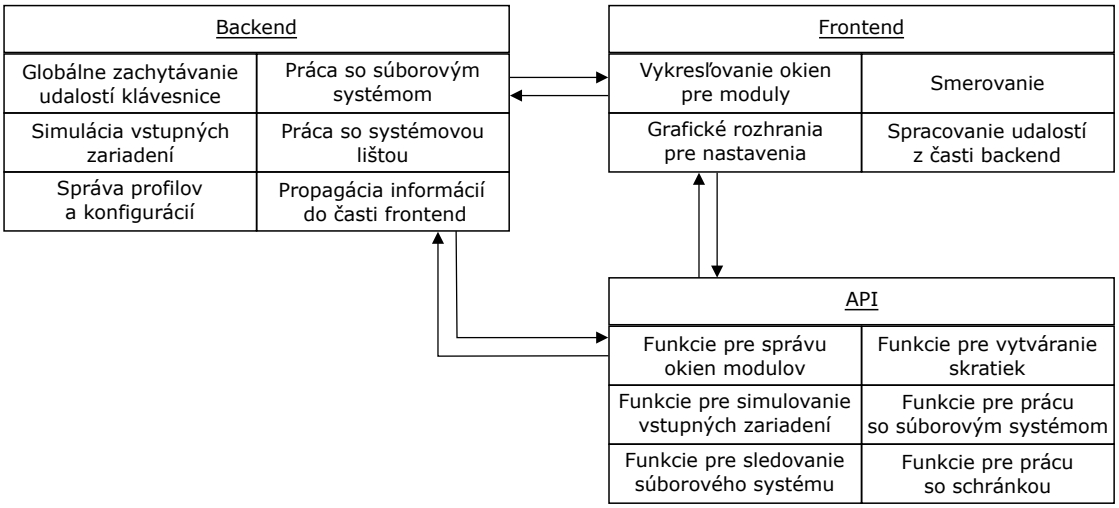
Ako prístup k vývoju tejto cieľovej multiplatformovej aplikácie bol zvolený vývoj desktopovej webovej aplikácie, ktorý je popísaný v sekcii 2.1.3. Použitie tohto prístupu spôsobí, že vývoj modulov bude prebiehať podobne ako vývoj webových aplikácií. Jazyky používané pri vývoji webových aplikácií sa často vyskytujú na vysokých umiestneniach v rebríčkoch popularity programovacích jazykov. V rebríčkoch PYPL[1] a TIOBE[2] sa jazyk JavaScript nachádza na pozíciách 3 a 6. Jazyk TypeScript sa v rebríčku TIOBE nachádza na pomerne nízkej pozícii 39, ale v rebríčku PYPL sa nachádza na pozícií 8. Voľba tohto prístupu k vývoju aplikácie teda umožní splnenie požiadaviek 2 a 6 definovaných v kapitole Analýza.

Architektúra navrhovanej aplikácie sa taktiež odvíja od architektúry webových desktopových aplikácií. Jedná sa o 3 časti aplikácie: backend, frontend a API pre vývoj modulov. Časti backend a frontend spolupracujú pri samotnom používaní aplikácie. API pre vývoj modulov je časť, ktorá poskytuje metódy pre jazyky JavaScript a TypeScript, ktoré zjednodušujú prácu s oknami a funkcionalitou, ktorú výsledná aplikácia podporuje. Návrh architektúry spolu s úlohami častí aplikácie je zobrazený na obrázku 5.1.

5.1 Backend

Táto časť aplikácie pracuje s operačným systémom a poskytuje možnosti práce s prostriedkami zariadenia. Z tejto časti je samotná aplikácia spúšťaná. Úlohy tejto časti sú globálne zachytávanie udalostí klávesnice, práca so súborovým systémom, simulovanie vstupných zariadení, správa profilov a konfigurácií, sledovanie súborového systému a propagácia udalostí do časti frontend. Časť frontend môže používať funkcionalitu časti backend pomocou volaní metód na to určených. Niektoré časti API môžu taktiež priamo používať funkcionalitu časti backend.

[1] https://pypl.github.io/PYPL.html
[2] https://www.tiobe.com/tiobe-index/



Obr. 5.1: Návrh architektúry cieľovej aplikácie. V tomto návrhu sú určené tri hlavné časti aplikácie: Backend, frontend a API pre vývoj modulov. Pri každej z týchto častí sú znázornené ich hlavné úlohy.

5.1.1 Globálne zachytávanie udalostí klávesnice

Táto úloha spočíva v sledovaní udalostí systému spojených so vstupmi z klávesnice. Problém pri práci s globálnymi klávesovými skratkami je nutnosť sledovania udalostí klávesnice, vrátane tých, ktoré neboli smerované do okien aplikácie. Na dosiahnutie takéhoto zachytávania udalostí je potrebná úzka spolupráca s operačným systémom. Udalosti takéhoto typu sú potom posielané do časti frontend, kde pomocou nich môžu byť vykonané rôzne akcie.

5.1.2 Práca so súborovým systémom

Aplikácia musí mať možnosť čítať a zapisovať dáta do súborov. Dôležité je taktiež informovanie o zmenách sledovaných súborov. Táto úloha sa skladá z dvoch častí. Prvou časťou je spracovanie požiadaviek na čítanie súborov a zapisovanie do súborov. Druhou časťou je možnosť pridania sledovaných ciest. Zmeny v súboroch a priečinkoch na danej ceste a cestách, ktoré sú jej potomkami budú potom generovať udalosti, ktoré budú propagované do časti frontend.

5.1.3 Simulácia vstupných zariadení

Podobne ako pri úlohe globálneho zachytávania udalostí klávesnice, je pri tejto úlohe potrebná spolupráca s operačným systémom. Jedná sa o spracovanie požiadaviek o simulovanie vstupu alebo sekvencie vstupov. Takto simulované vstupy musia byť zaznamenané aj v ostatných spustených aplikáciách. Táto úloha predstavuje podstatnú časť možností automatizácie.

5.1.4 Správa profilov

Táto úloha hovorí o možnosti ukladania viacerých rozložení rozšírených grafických rozhraní a možnosti rýchleho prepínania medzi nimi. Profily sú na zariadení reprezentované pomocou serializovaných dát, ktoré obsahujú informácie o otvorených oknách modulov, ich pozícií a

nastavených klávesových skratkách. Profily sú uložené na disku, aby ich nastavenia pretrvávali medzi behmi cieľovej aplikácie. Po zmene profilu sú aktuálne zobrazené okná modulov aplikácie zatvorené a okná uložené v profile sú otvorené a rozložené na nastavené pozície na obrazovke.

5.1.5 Práca so systémovou lištou

Časť backend poskytuje možnosť pridať aplikáciu na systémovú lištu. Takéto umiestnenie aplikácie na systémovú lištu umožňuje skrytie ovládacieho okna aplikácie bez jeho zavretia a straty možnosti jeho opätovného otvorenia. Časť aplikácie zobrazená na systémovej lište bude používateľovi ponúkať možnosť zobraziť alebo skryť ovládacie okno aplikácie a vypnúť aplikáciu.

5.2 Frontend

Táto časť aplikácie pracuje so zobrazovaním grafických používateľských rozhraní modulov aplikácie. Hlavnou úlohou tejto časti je vykresľovanie obsahu okien modulov a reakcia na vstupy od používateľa. Frontend je v desktopových webových aplikáciách vyvíjaný v jazykoch HTML, CSS a JavaScript (prípadne TypeScript). Úlohy tejto časti aplikácie sú: Vykresľovanie okien modulov, poskytovanie grafických rozhraní pre nastavenia častí aplikácie, správa udalostí získaných z časti backend a smerovanie adries URL na správne komponenty okien modulov. Frontend je časť aplikácie, s ktorou používateľ pracuje primárne. Niektoré akcie používateľa môžu byť vykonané priamo v rámci časti frontend, zatiaľ čo iné časti môžu používať funkcionalitu časti backend.

5.2.1 Vykresľovanie okien pre moduly

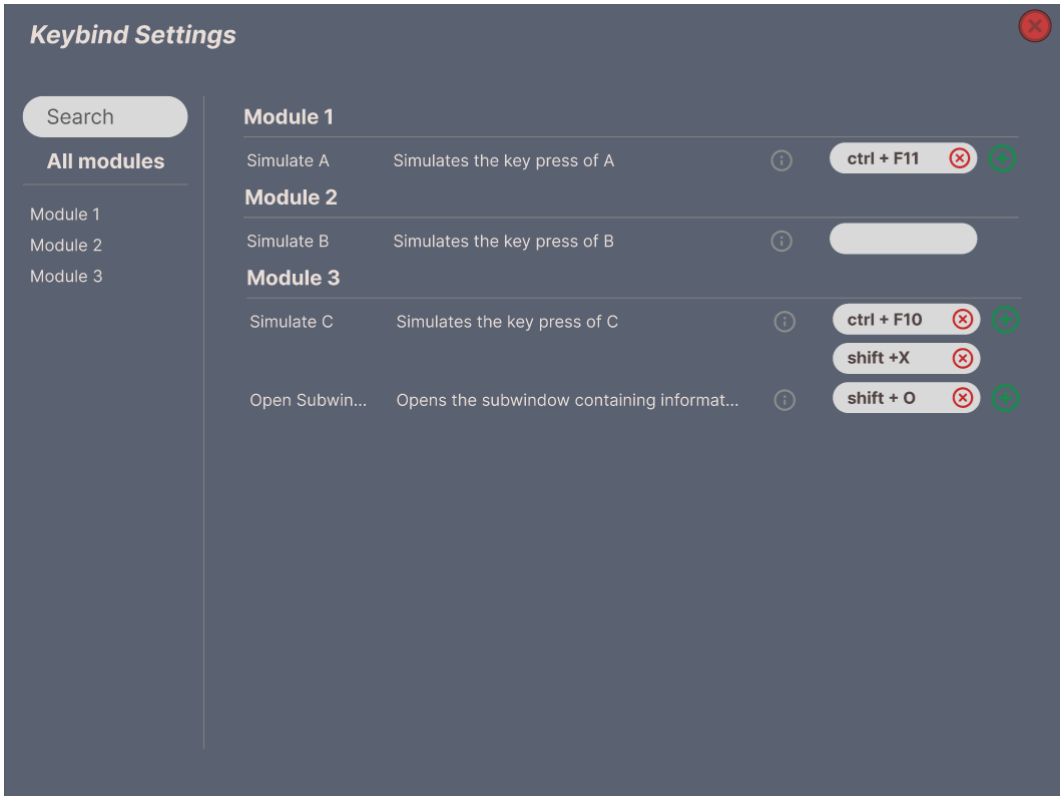
Toto je najpodstatnejšia úloha časti frontend. Jedná sa o schopnosť vytvárať okná, v ktorých je vykreslený obsah modulov aplikácie. Frontend si uchováva informácie o takýchto oknách, čím je umožnená správa okien modulov a práca s profilmi. Keďže je cieľová aplikácia vyvíjaná ako desktopová webová aplikácia, okná predstavujú inštanciu webového prehliadača s upraveným vzhľadom. Obsah okien je vykresľovaný rovnakým spôsobom ako obsah webových stránok v štandardných webových prehliadačoch.

5.2.2 Smerovanie

Prístup k vývoju desktopových webových aplikácií je podobný prístupu k vývoju štandardných webových aplikácií, pričom komunikácia cez sieť je nahradená medziprocesovou komunikáciou. O tom, aký obsah je v okne zobrazený, teda rozhoduje adresa URL, ktorá je otvorená v špecifickom okne. Je možné použiť smerovanie založeného na súborovom systéme, ktoré umožní automatické vytváranie adries URL pre jednotlivé okná modulov.

5.2.3 Poskytnutie grafických rozhraní pre nastavenia

Táto úloha hovorí o potrebe vytvorenia osobitných grafických rozhraní pre prácu s profilmi, modulmi a klávesovými skratkami. Jedná sa o grafické rozhrania poskytované priamo aplikáciou bez potreby dodatočných modulov. Nastavenia klávesových skratiek poskytujú používateľovi možnosť úpravy, pridávania a odstraňovania klávesových skratiek pre kon-



Obr. 5.2: Maketa grafického používateľského rozhrania pre nastavenia klávesových skratiek akcií modulov. Pomocou tohto grafického rozhrania môže používateľ spravovať kombinácie klávesov, ktoré aktivujú akcie v oknách modulov. Pri každej akcii spustiteľnej týmto spôsobom je zobrazený názov a popis tejto akcie. Pre každú akciu môže byť pridelený ľubovolný počet kombinácií klávesov.

krétne akcie modulov. Maketa obrazovky nastavení klávesových skratiek je zobrazená na obrázku 5.2.

Grafické rozhranie pre nastavenia modulov umožňuje používateľovi aktivovať alebo deaktivovať nainštalované moduly a pracovať s oknami týchto modulov. Pomocou rozhrania pre nastavenia profilov môže používateľ uložiť aktuálne rozloženie okien modulov ako nový profil, upraviť existujúce profily alebo existujúce profily zmazať.

5.2.4 Správa udalostí z časti backend

Táto úloha spočíva v reakcií na správy o udalostiach, ktoré časť frontend získava od časti backend. Tieto udalosti môžu signalizovať rozličné typy správ, ktoré musia byť spracované. Jedná sa napríklad o žiadosť o aktiváciu a deaktiváciu modulu, informovanie o stlačení klávesu, informovanie o zmene obsahu sledovaného súboru alebo žiadosť o zmenu aktívneho profilu.

5.3 API pre vývoj modulov

Táto časť aplikácie pracuje priamo s časťami backend a frontend. Jedná sa o aplikačno-programovacie rozhranie, ktoré vývojárom modulov poskytuje funkcie na zjednodušenie

použitia implementovanej funkcionality aplikácie. Toto rozhranie poskytuje vývojárom metódy pre 4 hlavné typy úkonov: správu okien, prácu s globálnymi klávesovými skratkami, simuláciu vstupných zariadení a prácu so súborovým systémom. Podstatou tohto API je vytvorenie úrovne abstrakcie nad konkrétnymi implementáciami funkcionality aplikácie. Cieľom je, aby pri vývoji autor modulu nemusel uvažovať nad spôsobom, akým aplikácia vnútorne funguje, ale mohol na vykonanie pokročilých funkcií použiť jednoduché rozhrania pre jazyky TypeScript a JavaScript.

5.3.1 Správa okien

Časť API pre správu okien ponúka funkcie pre otváranie, zatváranie, zobrazenie a skrytie okien modulu. Je podstatné oddelenie modulov, aby pomocou tejto časti API nebolo možné ovládať okná iného modulu.

5.3.2 Zachytávanie globálnych udalostí klávesnice

Časť API pre prácu s globálnymi klávesovými skratkami spočíva v poskytnutí funkcií pre vytvorenie akcie, ktorá môže následne byť aktivovaná stlačením nastavenej klávesovej kombinácie. Pri vytváraní takejto akcie je možné presne špecifikovať, aká funkcia má byť po aktivovaní vykonaná. Keďže klávesové kombinácie, ktoré majú jednotlivé akcie spúšťať sú nastaviteľné používateľmi aplikácie pomocou grafického rozhrania aplikácie, je potrebné každej akcii priradiť názov a popis, podľa ktorých ju bude používateľ vedieť identifikovať.

5.3.3 Simulácia vstupných zariadení

API pre simuláciu vstupných zariadení ponúka funkcie pre zostavenie sekvencie simulovaných udalostí klávesnice a myši. Jedná sa o rôzne typy stlačenia klávesu alebo tlačidla myši, rolovania kolieska myši a presunu myši. Je podstatné, aby implementácia tejto funkcionality rešpektovala poradie prvkov simulovanej sekvencie.

5.3.4 Práca so súborovým systémom

Pri práci so súborovým systémom API poskytuje funkcie na štandardné čítanie súboru a zápis do súboru. Okrem týchto funkcií, je poskytovaná možnosť vytvoriť pozorovateľa súborového systému. Tomuto pozorovateľovi je zadaná cesta, ktorá má byť sledovaná a funkcia, ktorá ma byť spustená pri zaznamenaní udalosti na zadanej ceste.

Kapitola 6

Implementácia aplikácie

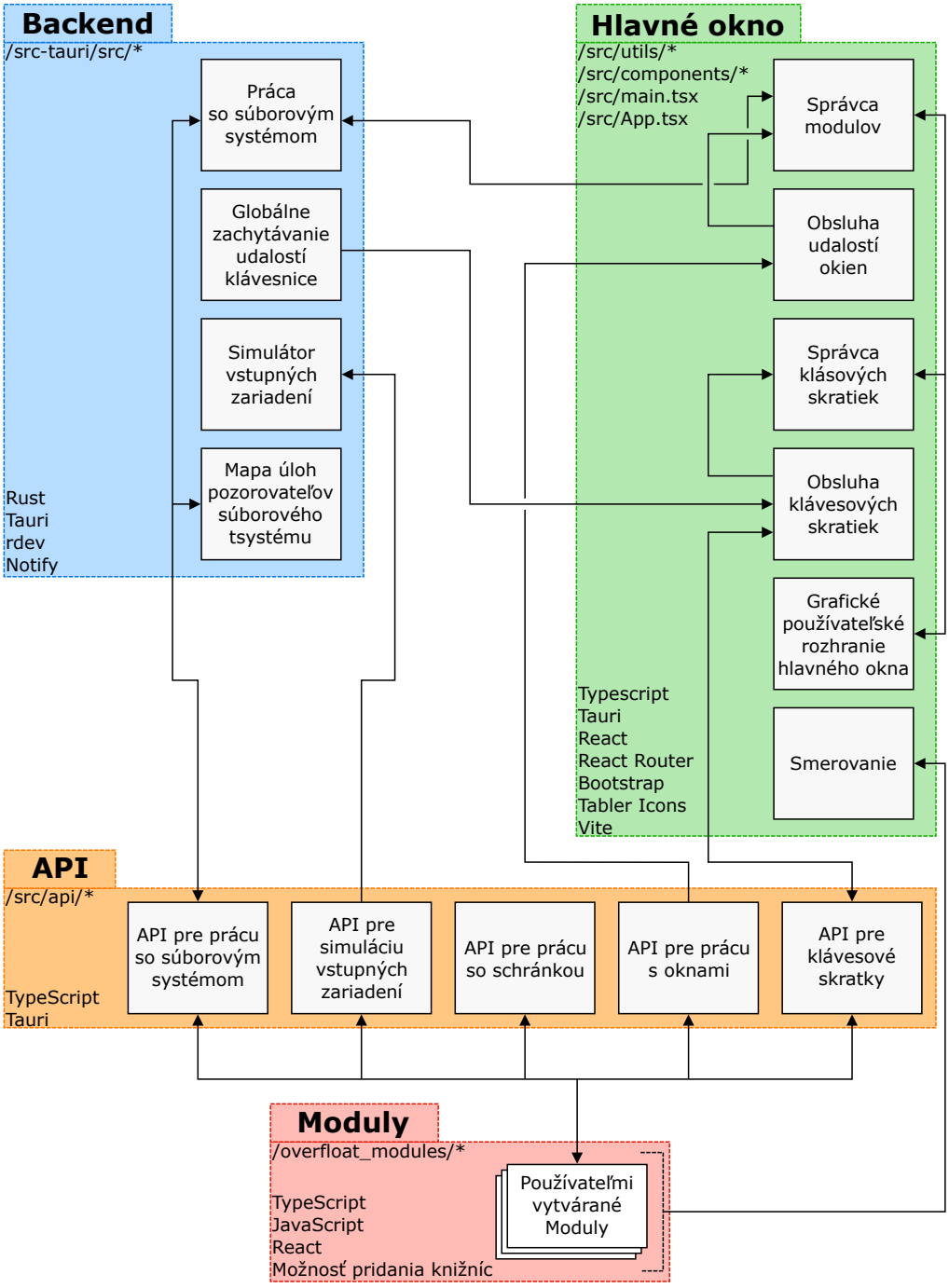
Ako už bolo spomenuté v kapitole 5, aplikácia pre túto bakalársku prácu bola vyvinutá ako modulárna multiplatformová desktopová webová aplikácia. Prvým krokom vývoja bola voľba aplikačného rámca, ktorý bude následne pri vývoji použitý. Z aplikačných rámcov používaných pri tomto prístupe k vývoju spomenutých v sekcii 2.1.3 je aktuálne najpoužívanejší rámec Electron.js. Tento aplikačný rámec by ponúkal všetku funkcionalitu potrebnú pre vytvorenie požadovanej aplikácie, ale nakoniec nebol zvolený. Zvolený bol aplikačný rámec Tauri, pretože ponúka oproti rámcu Electron.js vyšší výkon a nižšiu veľkosť výsledných aplikácií.

Vyšší výkon je spôsobený použitím behového prostredia vytvoreného v programovacom jazyku Rust. Toto behové prostredie však v porovnaní s behovým prostredím Node.js, ktoré používa aplikačný rámec Electron, poskytuje menší počet rozširujúcich balíčkov s funkcionalitou. Tento nedostatok balíčkov môže viesť k nutnosti vytvorenia vlastnej implementácie niektorej funkcionality, ktorú poskytujú balíčky pre prostredie Node.js. Samotné balíčky pre jazyk Rust sú nazývané *crate*. Správca balíčkov pre programovací jazyk Rust sa nazýva Cargo[1] a s jeho pomocou je možné použiť zbierku používateľmi vytvorených balíčkov Crates.io[2].

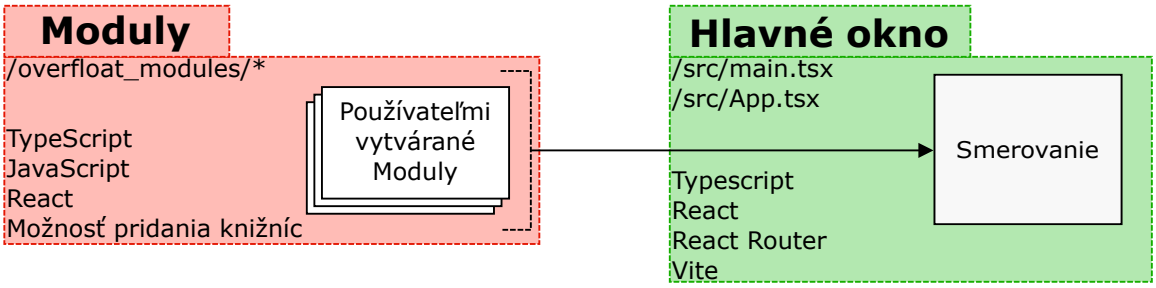
Nižšia veľkosť výsledných aplikácií je spôsobená tým, že Tauri na rozdiel od Electron.js používa natívne webové prehliadače rôznych operačných systémov a teda nie je potrebné dodávať jadro webového prehliadača Chromium do všetkých spustiteľných súborov ako pri použití aplikačného rámca Electron.js. Samotné použitie rôznych webových prehliadačov na rozdielnych cieľových platformách môže spôsobiť inkonzistencie vo výsledných grafických prvkoch aplikácie, ale k tomuto by nemalo dôjsť pri použití správnych techník pre vývoj grafických používateľských rozhraní pre webové aplikácie.

Architektúra výslednej aplikácie je zobrazená na obrázku 6.1. Táto architektúra spočíva v implementácií častí, ktoré pracujú s operačným systémom v jazyku Rust, implementácií častí spravujúcich jednotlivé moduly a profily v jazyku TypeScript, vytvorení grafického rozhrania pre ovládanie aplikácie v jazykoch HTML, CSS a TypeScript s knižnicou React, a vytvorení aplikačno-programovacieho rozhrania (API) umožňujúceho jednoduchú implementáciu funkcionality aplikácie do používateľmi vytvorených modulov. Celková architektúra tejto aplikácie je pomerne komplexná a nasledujúce sekcie budú zjednodušene približovať architektúru implementácie konkrétnych častí funkcionality.

[1] https://doc.rust-lang.org/cargo/
[2] https://crates.io/



Obr. 6.1: Architektúra vytvorenej aplikácie. Hlavné časti architektúry sú znázornené odlišnými farbami. Modrá časť znázorňuje časť backend, zelená časť znázorňuje hlavné okno aplikácie pre správu modulov, profilov a klávesových skratiek, oranžová časť znázorňuje aplikačno-programovacie rozhranie pre vývoj modulov a červená časť znázorňuje používateľmi vytvorené moduly. V ľavom hornom rohu týchto častí sú uvedené súbory, v ktorých sú dané časti implementované. V ľavom dolnom rohu týchto častí sú uvedené technológie, ktoré dané časti používajú.



Obr. 6.2: Časť architektúry predstavujúca používateľmi vytvorené moduly a smerovanie adries URL na správne komponenty okien modulov. Je použité smerovanie na základe súborového systému. Toto smerovanie bolo vytvorené pomocou kombinácie knižnice React, balíčku React Router a zväzovača Vite.

6.1 Používateľmi vytvárané moduly

Samotné používateľmi vytvorené moduly sú časti, z ktorých sa skladajú výsledné rozšírené používateľské rozhrania. Jedná sa o celky pozostávajúce z jedného alebo viacerých okien. Tieto moduly sú vývojármi modulov vytvárané v jazykoch TypeScript alebo JavaScript s použitím knižnice React a poskytnutého aplikačno-programovacieho rozhrania popísaného v nasledujúcich sekciách tejto kapitoly.

Vytváranie modulov prebieha vytvorením nového priečinka s názvom modulu v priečinku `overfloat_modules`. Tento priečinok musí mať v koreni práve jeden TSX alebo JSX súbor, ktorý reprezentuje hlavné okno modulu. Ostatné okná sú podoknami, ktoré musia byť reprezentované TSX alebo JSX súbormi v pod-priečinku `subwindows`. Názvy modulu a okien môžu obsahovať len alfanumerické znaky, znak `-` (ASCII hodnota 45) a znak `_` (ASCII hodnota 95). Predvoleným exportom súborov, ktoré implementujú okná, musí byť funkčný komponent knižnice React. Tento komponent reprezentuje samotné okno a teda návratová hodnota tohto komponentu definuje grafické rozhranie okna.

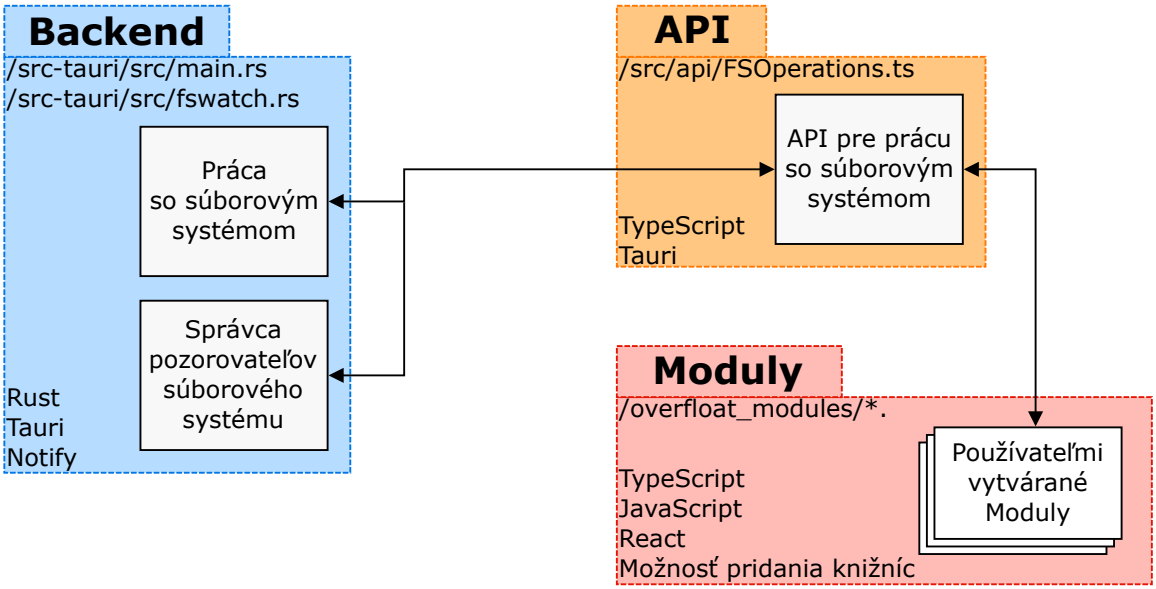
Okrem týchto pod-priečinkov je možné vytvoriť aj pod-priečinok `icons`, v ktorom sú umiestnené ikony okien modulov vo formáte PNG. Tieto ikony sú potom použité na reprezentáciu okien v lište modulov hlavného okna aplikácie. Názov ikony hlavného okna musí byť `icon.png` a názvy ikon podokien musia byť zhodné s názvami súborov reprezentujúcich dané podokná. Formát ostatných pod-priečinkov nie je z pohľadu funkčnosti aplikácie podstatný.

Použitie takéhoto špecifického formátu priečinka pri vytváraní modulov je potrebné preto, že aplikácia používa automatické smerovanie na základe súborového systému. Pre implementáciu samotného smerovania bola použitá kombinácia balíčka React Router[3] a zväzovača Vite. Smerovanie na základe súborového systému bolo zvolené preto, že umožňuje vytváranie modulov bez zásahu do vnútorného kódu aplikácie. Obrázok 6.2 ilustruje časti architektúry, ktoré sa podieľajú sa podieľajú na smerovaní.

6.2 Práca so súborovým systémom

Táto časť implementácie pozostáva z poskytnutia schopnosti čítania dát zo súborov a zápisu dát do súborov, a možnosti pridávania automatických reakcií na udalosti, ktoré nastanú

[3] https://www.npmjs.com/package/react-router



Obr. 6.3: Časť architektúry zabezpečujúca umožnenie práce so súborovým systémom. Samotný prístup k súborom prebieha vrámci časť backend s použitím programovacieho jazyka Rust. Na monitorovanie súborového systému je použitá crate Notify. Komunikácia smerom z časti API prebieha pomocou Tauri príkazov a komunikácia smerzom z časti backend pomocou Tauri udalostí.

v určitých častiach súborového systému. Časti architektúry, ktoré sa na tejto časti funkcionality podieľajú sú ilustrované na obrázku 6.3. Aplikačný rámec Tauri poskytuje vstavané funkcie pre čítanie zo súborov a zápis do súborov. Pôvodným plánom bolo použitie týchto funkcií, ale ich povolenie a sprístupnenie celého súborového systému spôsobovalo problémy pri spúšťaní aplikácie na operačnom systéme Linux. Z tohto dôvodu neboli nakoniec tieto funkcie z aplikačného rámca použité. Namiesto nich boli v súbore `src-tauri/src/main.rs` vytvorené Tauri príkazy na zápis a čítanie.

Na rozšírenie možností automatizácie bola pridaná podpora monitorovania súborového systému. Táto funkcionalita umožňuje sledovať určitý priečinok alebo súbor, pričom zmeny v tomto priečinku alebo súbore spustia nastavenú funkciu. Táto funkcionalita bola implementovaná v súbore `src-tauri/src/fswatch.rs` pomocou crate Notify[4]. Notify umožňuje vytvoriť takzvaných pozorovateľov (*watchers*), ktorý po nastavení cesty sledujú zmeny súborového systému na danej ceste a cestách, ktoré sú jej potomkami. Takéto zmeny potom vyvolávajú udalosti (takzvané *Notify udalosti*), ktoré obsahujú informácie o zmenách. Implementácia pozorovateľov je rozdielna pre operačné systémy Linux a Windows, a teda bolo potrebné vytvoriť 2 rozdielne funkcie, ktoré Notify udalosti spracúvajú. Výber funkcie, ktorá bude na spracovanie Notify udalostí použitá prebieha pri kompilácií aplikácie na základe cieľovej platformy.

Táto funkcionalita nepoužíva hlavné okno aplikácie ale vyžaduje komunikáciu medzi hlavným procesom, ktorý predstavuje časť backend a procesom WebView okna modulu. Ako je popísané v sekcii 2.1.3, komunikácia medzi týmito dvoma typmi okien zvyčajne prebieha použitím Tauri príkazov. Tauri príkazy sú však použiteľné, len v prípade, že komunikáciu inicializoval proces typu WebView. Tento typ komunikácie je použitý pri zapínaní

[4] https://crates.io/crates/notify

monitorovania súborového systému, ale po zachytení Notify udalosti ho nie je možné použiť na poskytnutie informácií o udalosti procesu okna. Na komunikáciu týmto smerom časť backend po zachytení Notify udalosti spracuje relevantné informácie z nej a vyšle Tauri udalosť obsahujúcu tieto informácie procesu okna.

Pre zjednodušenie práce s monitorovaním súborového systému, zápisom do súborov a čítaním súborov bola vytvorená časť API v súbore `src/api/FSOperations.ts`. Táto časť API sprístupňuje spomenuté Tauri príkazy na zápis a čítanie súborov a okrem nich implementuje triedu pre správcu pozorovateľov. Táto trieda je implementovaná návrhovým vzorom jedináčik, pričom je využité to, že každé okno je spustené v osobitnej inštancii webového prehliadača. V API je vytvorená inštancia tejto triedy, ktorá je následne exportovaná pod názvom `WatchManager`. Toto spôsobuje, že každé okno má práve jednu inštanciu triedy správcu pozorovateľov. Pomocou metód tejto triedy môžu vývojári modulov pracovať s monitorovaním súborového systému. Pri vytvorení nového pozorovateľa súborového systému je zvolená cesta a funkcia, ktorá sa má vykonať v prípade zmeny na danej ceste. Pre automatické spúšťanie zvolenej funkcie, je pri vytvorení pozorovateľa zapnuté zachytávanie Tauri udalostí generovaných časťou backend pri zmene na sledovanej ceste.

Kvôli potrebe podpory viacerých pozorovateľov v jednom okne bola v časti backend vytvorená štruktúra, ktorá mapuje názvy modulov, označenia okien a identifikátory pozorovateľov na špecifické úlohy (*task*), ktoré zaisťujú monitorovanie. Táto štruktúra je implementovaná v súbore `src-tauri/src/fswatch.rs`. Toto umožňuje odstrániť problém viacnásobného vytvorenia rovnakého pozorovateľa, a teda viacnásobného spracovania rovnakej udalosti, pri obnovení okna modulu aj pri podpore viacerých pozorovateľov v jednom okne.

API pre prácu so súborovým systémom poskytuje nasledovné dátové typy, funkcie a metódy:

- `FSResult` – dátový typ reprezentujúci výsledok operácie čítania súboru alebo zápisu do súboru. Tento typ má nasledujúce polia:
  - `successful: boolean` – informácia o tom, či operácia prebehla úspešne.
  - `path: string` – cesta k súboru, na ktorom bola operácia vykonávaná.
  - `message: string` - pole obsahujúce chybový výpis v prípade neúspechu operácie. V prípade úspechu operácie čítania je v tomto poli obsah získaný z prečítaného súboru. V prípade úspechu operácie zápisu je toto pole prázdne.
- `FSEventKind` – enumeorvaný dátový typ predstavujúci druh udalosti súborového systému. Tento typ môže nadobúdať nasledujúce hodnoty:
  - `Create` – hodnota predstavujúca vytvorenie súboru alebo priečinka.
  - `Remove` – hodnota predstavujúca zmazanie súboru alebo priečinka.
  - `Modify` – hodnota predstavujúca zmenu obsahu súboru alebo priečinka.
  - `Rename` – hodnota predstavujúca premenovanie alebo presunutie súboru alebo priečinka.
- `FSEvent` – dátový typ predstavujúci udalosť súborového systému. Tento dátový typ obsahuje nasledujúce polia:
  - `kind: FSEventKind` – druh udalosti súborového systému.
  - `isDir: boolean` – informácia o tom, či sa udalosť týka priečinka alebo súboru.

  - `path: string` – cesta k súboru alebo priečinku v ktorom zmena nastala.
  - `pathOld: string` – pôvodná cesta k súboru alebo priečinku v prípade udalosti premenovania alebo presunu. V ostatných typoch udalostí je hodnota tohto poľa rovnaká ako hodnota poľa `path`.

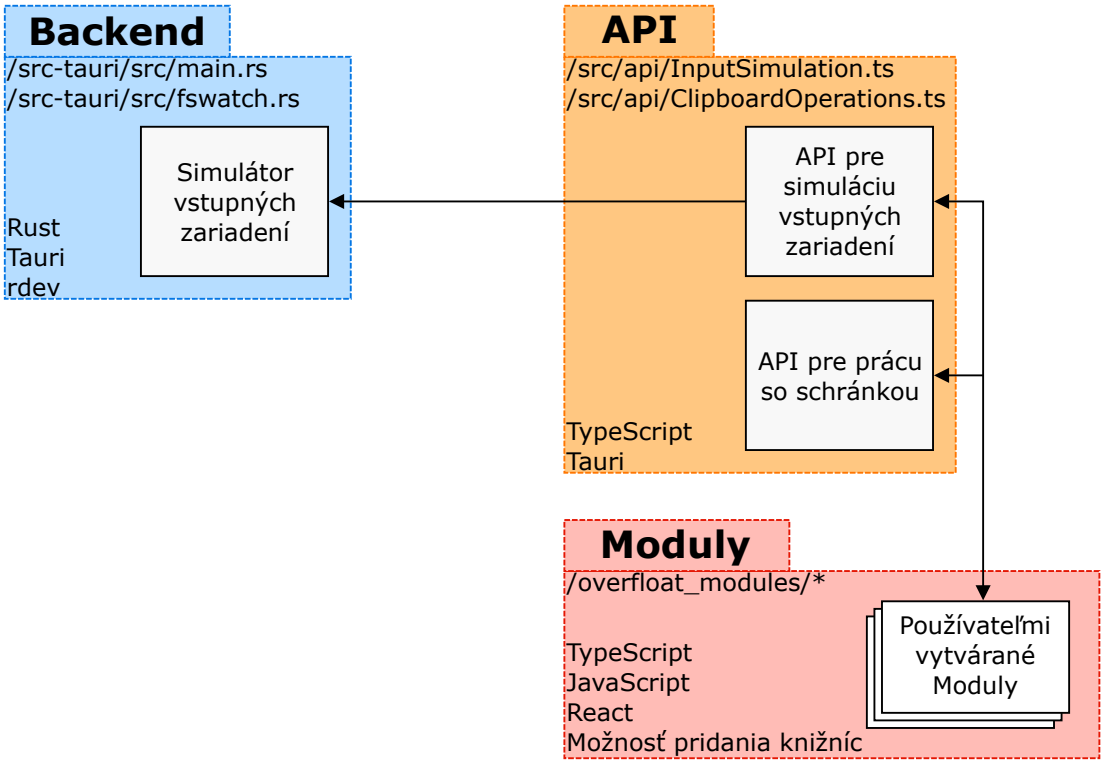
- `writeFile(content, path, useRelativePath, appendMode): FsResult` – asynchrónna funkcia pre zápis textu do súboru. Táto funkcia vracia výsledok operácie zápisu a má nasledovné parametre:
  - `content: string` – text, ktorý ma do súboru byť zapísaný.
  - `path: string` – cesta k súboru, do ktorého má text byť zapísaný.
  - `useRelativePath: boolean` – informácia o tom, či má byť cesta použitá ako absolútna cesta alebo ako relatívna cesta. V prípade že je použitá relatívna cesta, táto cesta začína v priečinku `$1/overfloat_modules/$2`, kde `$1` je priečinok, v ktorom je aplikácia nainštalovaná a `$2` je názov modulu, z ktorého bola táto funkcia zavolaná.
  - `appendMode: boolean` – informácia o tom, či má byť pôvodný obsah súboru ponechaný a text pripojený na koniec, alebo zmazaný a nahradený novým textom.

- `readFile(path, useRelativePath): FSResult` – asynchrónna funkcia pre získanie obsahu zo súboru. Táto funkcia vracia výsledok operácie čítania a má nasledovné parametre:
  - `path: string` – cesta k súboru, z ktorého má byť obsah získaný.
  - `useRelativePath: boolean` – informácia o tom, či má byť cesta použitá ako absolútna cesta alebo ako relatívna cesta. V prípade že je použitá relatívna cesta, táto cesta začína v rovnakom priečinku ako pri funkcii `writeFile()`.

- `watchPath(id, path, callback)` – metóda inštancie `WatchManager`, ktorá spustí sledovanie súborového systému na zadanej ceste.
  - `id: string` – identifikátor nového pozorovateľa. Tento identifikátor musí byť v okne unikátny.
  - `path: string` – absolútna cesta, na ktorej má nový pozorovateľ sledovať súborový systém.
  - `callback: (FSEvent) => void` – funkcia, ktorá má byť spustená keď na sledovanej ceste nastane udalosť súborového systému. Táto funkcia musí mať jeden parameter typu `FSEvent`.

- `stopWatching(id)` – metóda inštancie `WatchManager`, ktorá ukončí sledovanie so zvoleným identifikátorom pozorovateľa. Táto metóda má nasledovný parameter:
  - `id: string` – identifikátor pozorovateľa súborového systému, ktorý má byť zastavený.

- `stopAll()` – metóda inštancie `WatchManager`, ktorá ukončí všetkých sledovateľov okna, z ktorého je zavolaná.



Obr. 6.4: Časť architektúry poskytujúca možnosť simulovania vstupných zariadení. Simulátor vstupných zariadení pozostáva z funkcií používajúcich funkcionalitu crate rdev. Pre zjednodušenie automatického vkladania textu bola táto časť rozšírená o prácu so schránkou.

6.3 Simulácia vstupných zariadení

Ako ďalšia funkcionalita bola pridaná možnosť simulovania vstupných zariadení. Časti architektúry, ktoré sú touto funkcionalitou používané sú ilustrované na obrázku 6.4. Na samotné simulovanie udalostí klávesnice a myši bol vytvorený v časti backend simulátor vstupných zariadení, ktorý použitá crate rdev[5]. Tento simulátor je implementovaný formou sady funkcií v súbore `src-tauri/src/inputsim.rs`. Pri simulácii klávesnice je podporovaná simulácia stlačenia klávesu bez jeho uvoľnenia, simulácia uvoľnenia klávesu a simulácia rýchleho stlačenia klávesu nasledovaná jeho uvoľnením. Pri simulácii myši je podporovaná simulácia stlačenia a uvoľnenia ľavého, stredného a pravého tlačidla myši, podobná ako pri simulácii klávesnice. Okrem toho je podporovaná simulácia presunu ukazovateľa myši na špecifikované súradnice a simulácia rolovania kolieska myši. Simulovanie vstupných zariadení prebieha postupným vykonávaním zostavenej sekvencie krokov simulácie. Pre sprístupnenie tejto funkcionality bola v súbore `src/api/InputSimulation.ts` vytvorená ďalšia časť API. Táto časť API poskytuje dátový typ `SimulationStep` použitý na interné reprezentovanie kroku simulácie a enumerované dátové typy reprezentujúce klávesy, modifikačné klávesy, tlačidlá myši a smer rolovania kolieska myši.

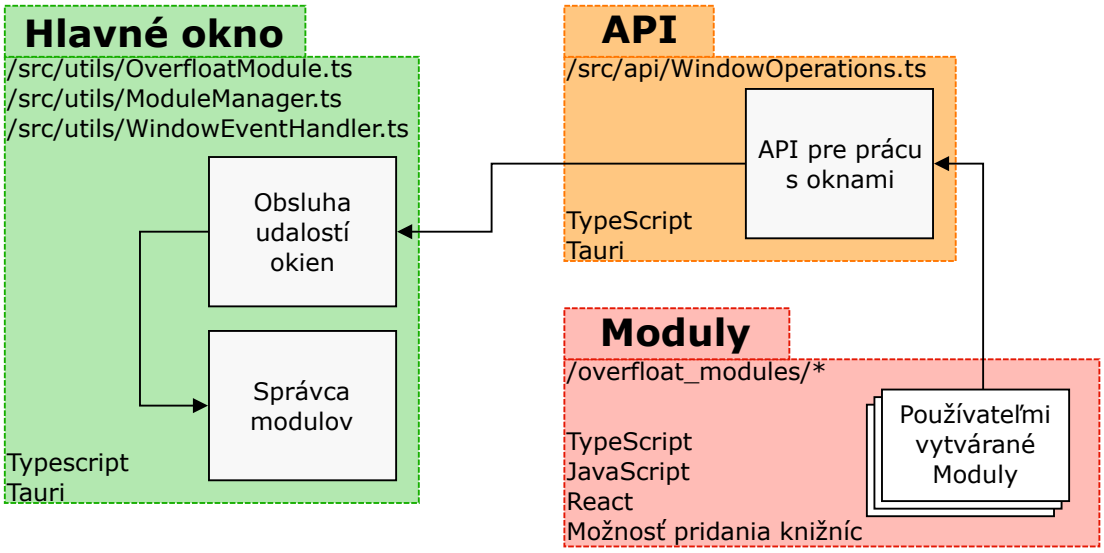
Jedným z prípadov použitia simulácie vstupných zariadení je automatické písanie textu. Pri dlhšom texte by zostavenie sekvencie krokov, ktoré je potrebné vykonať na napísanie tohto textu bolo časovo náročné. Preto bola táto časť funkcionality rozšírená o možnosť

[5] https://crates.io/crates/rdev

práce so schránkou. Toto umožní vložiť už zostavený text do schránky, z ktorej je ho následne možné vložiť do zameraného textového poľa. Vloženie textu do cieľového textového poľa môže byť vykonané simulovaním stlačenia klávesovej skratky `Ctrl + V`. Aplikačný rámec Tauri ponúka funkcie na manipuláciu so schránkou. Tieto funkcie boli obalené funkciami s vhodnejším názvom v súbore `src/api/ClipboardOperations.ts`.

Časti API pre simulovanie vstupných zariadení a prácu so schránkou poskytujú nasledujúce funkcie:

- simKeyDown(key), simKeyUp(key), simKeyPress(key): SimulationStep – funkcie slúžiace na vytvorenie krokov simulácie pre prácu s klávesnicou. Tieto funkcie vracajú reprezentáciu kroku simulácie v dátovom type `SimulationStep` a majú nasledujúci parameter:
  - `key: Key|ModifierKey` – kláves, s ktorým má požadovaný krok simulácie pracovať.
- simMouseDown(button), simMouseUp(button), simMouseClick(button): SimulationStep – funkcie slúžiace na vytvorenie krokov simulácie pre prácu s tlačidlami myši. Tieto funkcie vracajú reprezentáciu kroku simulácie v dátovom type `SimulationStep` a majú nasledujúci parameter:
  - `button: MouseButton` – tlačidlo myši, s ktorým má požadovaný krok simulácie pracovať.
- `simMouseScroll(direction)` – funkcia slúžiaca na vytvorenie kroku simulácie reprezentujúceho rolovanie kolieska myši. Táto funkcia vracia reprezentáciu kroku simulácie v dátovom type `SimulationStep` a má nasledujúci parameter:
  - `direction: Direction` – smer, do ktorého má byť rolovanie kolieska myši v tomto kroku simulované.
- `simMouseMove(x,y): SimulationStep` – funkcia slúžiaca na vytvorenie kroku simulácie reprezentujúceho presun ukazovateľa myši na zadané súradnice. Táto funkcia vracia reprezentáciu kroku simulácie v dátovom type `SimulationStep` a má nasledujúce parametre:
  - `x: number` – poloha na osi X, na ktorú má byť ukazovateľ myši presunutý.
  - `y: number` – poloha na osi Y, na ktorú má byť ukazovateľ myši presunutý.
- `inputSimulation(steps)` – funkcia na vykonanie sekvencie krokov simulácie. Táto funkcia má nasledujúci parameter:
  - `steps: SimulationStep[]` – pole simulačných krokov v poradí, v akom sa majú vykonať. Tieto kroky je vhodné konštruovať pomocou ostatných funkcií poskytovaných touto časťou API.
- `clipboardRead(): string` – asynchrónna funkcia na získanie aktuálneho textu v schránke. Táto funkcia vráti text získaný zo schránky.
- `clipboardWrite(text)` – asynchrónna funkcia, ktorá vloží text do schránky. Táto funkcia má nasledovný parameter:
  - `text: string` – text, ktorý má byť do schránky vložený.



Obr. 6.5: Časť architektúry umožňujúca prácu s oknami. Okná modulov sú interne uložené v inštanciách triedy modulu. Tieto inštancie sú uložené v správcovi modulov, pomocou ktorého je možné k nim pristúpiť. Komunikácia medzi oknami modulov a správcom modulov prebieha pomocou Tauri udalostí, ktoré sú zachytávané obsluhou udalostí okien.

6.4 Práca s oknami modulov

Táto časť funkcionality spočíva v možnosti otvárať a spravovať okná a podokná modulov. Časti architektúry, ktoré sú pri poskytovaní tejto funkcionality používané sú ilustrované na obrázku 6.5. Aktívne moduly sú v aplikácií reprezentované inštanciami triedy `OverfloatModule` implementovanej v súbore `src/utils/OverfloatModule.ts`. Táto trieda poskytuje metódy pre prácu s modulom ako napríklad otváranie podokien, minimalizáciu okien a pridávanie akcií, pre ktoré je možné nastaviť klávesové skratky.

Aktívne a neaktívne moduly sú udržiavané v správcovi modulov. Správca modulov je trieda s návrhovým vzorom jedináčik, ktorá poskytuje metódy pre spúšťanie a vypínanie jednotlivých modulov a správu profilov. Inštancia tejto triedy je jednou z hlavných častí aplikácie, pretože práve v nej je uchovávaný aktuálny stav modulov. Táto inštancia je v hlavnom okne aplikácie vytvorená pri načítaní grafického rozhrania hlavného okna aplikácie. Implementácia tejto triedy sa nachádza v súbore `src/utils/ModuleManager.ts`.

Na to, aby vývojári modulov mohli pracovať s oknami modulov, je potrebné, aby boli určité metódy správcu modulov a samotných modulov sprístupnené pre vývoj modulov. Toto nie je riešiteľné priamočiaro sprístupnením týchto metód, pretože každé okno modulu je spustené v osobitnej inštancii webového prehliadača, medzi ktorými nie je možná priama komunikácia. Takáto komunikácie je však možná pomocou medziprocesovej komunikácie poskytovanej aplikačným rámcom Tauri, popísanej v sekcii 2.1.3. Keďže okná modulov aj hlavné okno aplikácie sú procesy typu WebView, je možné použiť komunikáciu pomocou Tauri udalostí.

No to, aby mohla byť takáto komunikácia použitá na aktiváciu metód inštancií tried v hlavnom okne aplikácie, je potrebný nasledovný postup:

1. vytvorenie Tauri udalosti s vhodným názvom v okne modulu,
2. dodanie parametrov, s ktorými má byť metóda spustená do vytvorenej udalosti,

3. odoslanie tejto udalosti do hlavného okna aplikácie,
4. zachytenie Tauri udalosti v hlavnom okne aplikácie,
5. spracovanie parametrov z udalosti a aktivácia metódy v hlavnom okne aplikácie.

Zachytávanie Tauri udalostí v hlavnom okne aplikácie je vyriešené vytvorením triedy pre obsluhu udalostí okien. Táto trieda používa návrhový vzor jedináčik a je implementovaná v súbore `src/utils/WindowEventHandler.ts`. V hlavnom okne aplikácie je vytvorená inštancia tejto triedy pri načítaní grafického rozhrania hlavného okna aplikácie.

Nie je vhodné, aby vývojári modulov museli pre každú operáciu s oknami ručne vytvárať Tauri udalosti a teda je potrebné tento postup zjednodušiť. Pre túto úlohu bola v súbore `src/api/WindowOperations.ts` vytvorená časť API pre prácu s oknami. Táto časť API implementuje funkcie, ktoré vytvárajú a odosielajú potrebné Tauri udalosti na základe zadaných parametrov.

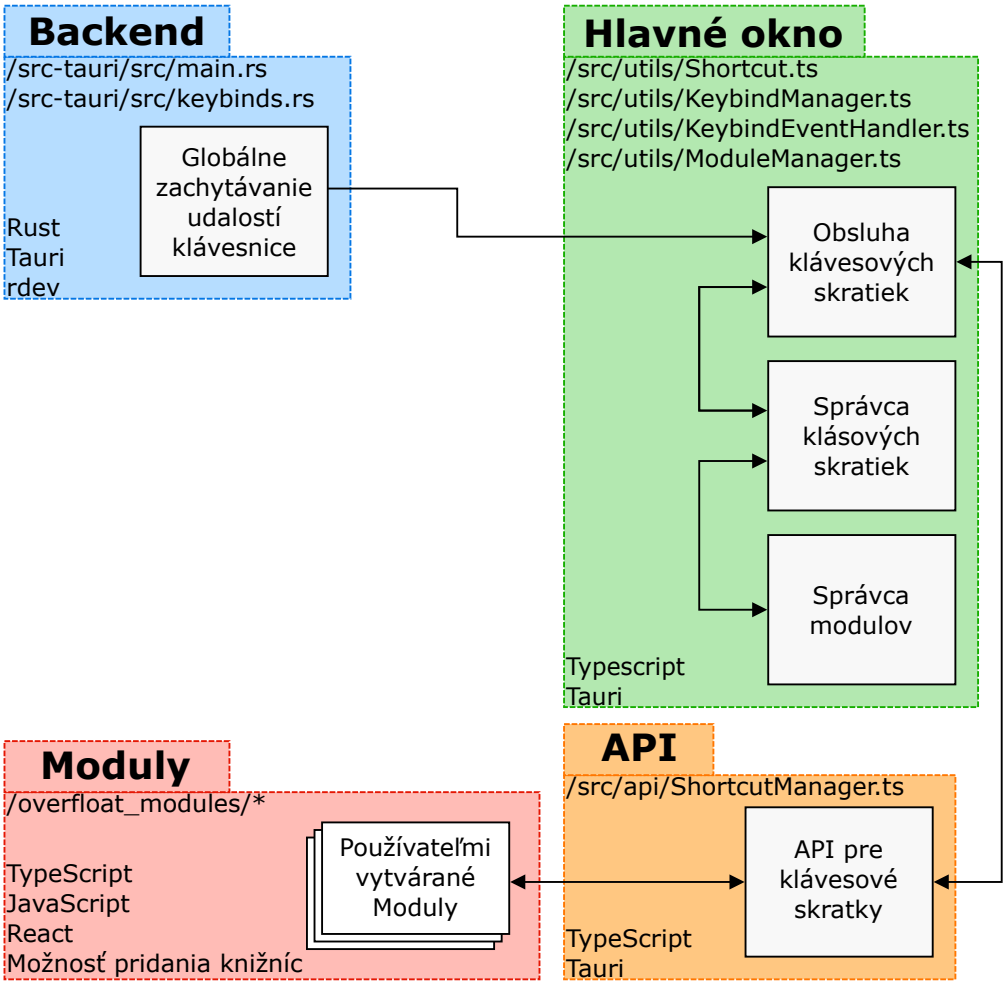
API pre prácu s oknami poskytuje nasledujúce rozhrania, dátové typy a funkcie:

- `NameValuePairs` – rozhranie pre dáta typu názov-hodnota. Toto rozhranie je použité pri otváraní nového podokna na poskytnutie dát pre toto nové podokno. Hodnoty môžu byť reťazce alebo čísla. Formát pre tento dátový typ je `{nazov1: hodnota1, nazov2: hodnota2}`.
- `WindowSettings` – dátový typ reprezentujúci nastavenia okna, používaný pri otváraní nového podokna. Tento dátový typ obsahuje nasledujúce voliteľné polia:
  - `visible: boolean` – informácia o tom, či má byť podokno po vytvorení zobrazené alebo minimalizované.
  - `transparent: boolean` – informácia o tom, či má byť podokno priehľadné.
  - `x: number` – poloha na osi X, na ktorej má byť podokno otvorené.
  - `y: number` – poloha na osi Y, na ktorej má byť podokno otvorené.
  - `height: number` – počiatočná výška okna.
  - `width: number` – počiatočná šírka okna.
- `showMainWindow(), hideMainWindow(), closeMainWindow()` – funkcie, ktoré slúžia na zobrazenie, minimalizovanie a zavretie hlavné okno modulu. Zavretie hlavného okna modulu spôsobí úplné zavretie celého modulu.
- `openSubwindow(subwindowName, title, parameters, windowSettings)` – funkcia, ktorá otvorí nové podokno. Táto funkcia má nasledujúce parametre:
  - `subwindowName: string` – názov typu podokna, ktoré má byť otvorené.
  - `title: string` – titulok pre nové okno. Tento parameter nie je povinný.
  - `parameters: NameValuePairs` – dáta pre nové okno, tieto dáta sú oknu poskytnuté cez adresu URL. Tento parameter nie je povinný.
  - `windowSettings: WindowSettings` – nastavenia pre nové okno. Tento parameter nie je povinný.
- `getParameter(name): string|undefined` – funkcia, ktorá získa hodnotu z dát poskytnutých podoknu pri jeho otváraní. Ak tieto dáta obsahovali hodnotu so zadaným názvom funkcia vráti túto hodnotu ako reťazec, inak táto funkcia vráti nedefinovanú hodnotu. Táto funkcia má nasledujúci parameter:

– `name: string` – názov hodnoty, ktorá má byť získaná.

`showWindow(label), hideWindow(label), closeWindow(label)` – funkcie, ktoré zobrazia, minimalizujú alebo zavrú zvolené okno modulu. Tieto funkcie môžu pracovať len s oknami modulu, z ktorého sú volané. Zavretie hlavného okna modulu spôsobí úplné zavretie celého modulu. Tieto funkcie majú nasledovný parameter:

– `label: string` – označenie okna, na ktorom má byť akcia vykonaná. Tento parameter je voliteľný a v prípade jeho absencie je akcia vykonaná na okne, z ktorého bola funkcia zavolaná.



Obr. 6.6: Časť architektúry zameraná na implementáciu prácu s globálnymi klávesovými skratkami. Klávesové skratky sú interne reprezentované pomocou inštancií triedy pre skratky a sú uložené v inštanciách aktívnych modulov v správcovi modulov. Komunikácia prebieha pomocou Tauri udalostí, ktorých obsluhu a smerovanie do správnych okien rieši obsluha klávesových skratiek.

6.5 Globálne klávesové skratky

Jednou z požiadaviek na výslednú aplikáciu je možnosť nastavenia klávesových skratiek pre aktiváciu určitých akcií modulov. Aplikačný rámec Tauri poskytuje funkcionalitu na nastavovanie globálnych klávesových skratiek, ale konkrétna implementácia spôsobuje to, že samotná klávesová skratka je po aktivácií zachytená a nie je ďalej propagovaná do systému. Po diskusií s niekoľkými cieľovými používateľmi bolo stanovené, že takéto správanie klávesových skratiek nie je vhodné, a teda bola vytvorená vlastná implementácia globálnych klávesových skratiek. Časti architektúry, ktoré sa podieľajú na implementácii práce s globálnymi klávesovými skratkami sú ilustrované na obrázku 6.6. Implementácia systému globálnych klávesových skratiek sa skladá z 3 hlavných častí:

- Globálne zachytávanie udalostí klávesnice a uchovávanie stavu klávesnice.
- Interná reprezentácia akcií spustiteľných klávesovými skratkami a klávesových kombinácií, ktoré ich majú spúšťať.
- Propagácia udalostí klávesnice do okien modulov a systém spúšťania cieľových akcií.

Pre globálne zachytávanie udalostí klávesnice bola rovnako ako pre simulovanie vstupných zariadení použitá crate rdev. Táto crate poskytuje možnosť zachytávania udalostí klávesnice z operačného systému. Keďže táto crate poskytuje len udalosti jednotlivých klávesov, bola v súbore `src-tauri/src/keybinds.ts` vytvorená štruktúra na reprezentovanie celkového stavu klávesnice. Pomocou uchovávania stavu klávesnice je možné možné udržanie informácií o tom, ktoré modifikačné klávesy sú aktuálne stlačené. Okrem toho, uchovávanie stavu klávesnice umožňuje filtrovať viaceré udalosti klávesnice reprezentujúce jedno dlhé stlačenie klávesu. Pri stlačení klávesu, ktorý nie je modifikačný, je do hlavného okna aplikácie zaslaná udalosť, ktorá obsahuje informácie o stlačenom klávese a stlačených modifikačných klávesoch.

Samotné klávesové skratky sú interne reprezentované inštanciami triedy `Shortcut`, ktorá je implementovaná v súbore `src/utils/Shortcut.ts`. Tieto inštancie sú uchovávané v časti inštancie modulu reprezentujúcej okno, do ktorého daná skratka patrí. Ako bolo popísané v sekcii 6.4, inštancie modulov sú udržiavané v správcovi modulov. Druhou časťou internej reprezentácie klávesových skratiek je správca klávesových skratiek. Jedná sa o triedu s návrhovým vzorom jedináčik, ktorej inštancia je spustená v hlavnom okne aplikácie. Táto trieda v sebe uchováva mapovanie klávesových kombinácií na klávesové skratky všetkých okien modulov, ktoré má daná klávesová kombinácia aktivovať. Takéto mapovanie umožňuje jednoduché vyhľadanie skratiek, ktoré majú byť pri stlačení kombinácie kláves spustené. Okrem tohto mapovania poskytuje správca klávesových skratiek aj metódy na pridávanie a spravovanie klávesových skratiek.

Propagovanie udalosti klávesnice prebieha v troch krokoch. Udalosť klávesnice je zachytená v časti backend pomocou crate rdev. V tejto časti sú ku klávesu, ktorý bol stlačený pridané informácie o stlačených modifikačných klávesoch. Takto doplnené informácie o stlačení klávesovej kombinácie sú pomocou Tauri udalosti odoslané do hlavného okna aplikácie. V hlavnom okne aplikácie túto Tauri udalosť zachytí obsluha klávesových skratiek a pomocou mapy klávesových kombinácií zistí, ktorým oknám modulov má byť poslaná udalosť značiaca aktiváciu skratky. Po získaní týchto okien, je pre každé z nich vytvorená Tauri udalosť s identifikátorom skratky na aktivovanie a tieto udalosti sú odoslané cieľovým oknám modulov.

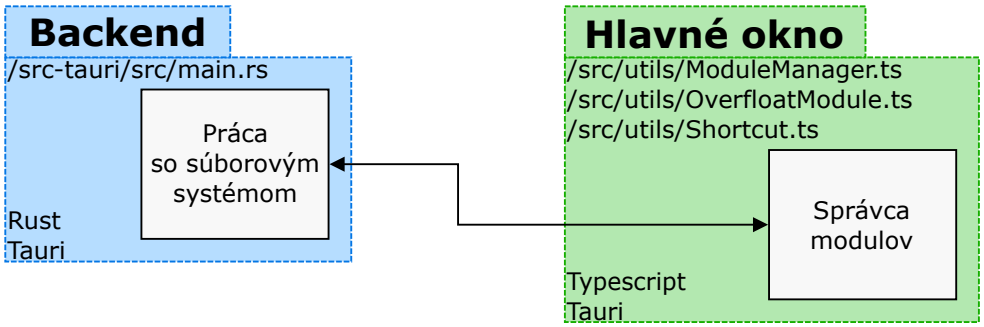
Zachytávanie udalostí je riešené v rámci časti API pre prácu s klávesovými skratkami. Táto časť API je implementovaná v súbore `src/api/ShortcutManager.ts`. API pre klávesové skratky implementuje triedu správcu skratiek okna, ktorá používa návrhový vzor jedináčik. Inštancia tejto triedy je sprístupnená pod názvom `ShortcutManager`, pričom je opäť využitá interakcia osobitnej inštancie webového prehliadača pre každé okno. Toto spôsobí, že každé okno má práve jednu vlastnú inštanciu tejto triedy. Táto interakcia je bližšie popísaná v časti sekcie 6.2 popisujúcej zjednodušenie práce s monitorovaním súborového systému.

Okrem zachytávania Tauri udalostí reprezentujúcich aktiváciu skratky, táto trieda ponúka metódy pre pridávanie a správu skratiek. Tieto metódy bolo potrebné pridať pretože inštancia správcu klávesových skratiek sa nachádza v inej inštancii webového prehliadača a teda nie je možné jej metódy volať priamo. Na komunikáciu zo správcu skratiek okna do správcu klávesových skratiek v hlavnom okne aplikácie sú využité Tauri udalosti. Postup vytvárania týchto Tauri udalostí je podobný tomu, ktorý je popísaný v sekcii 6.4. Zachytávanie udalostí v hlavnom okne rieši správca klávesových skratiek, a na vytváranie a odosielanie týchto udalostí slúžia metódy inštancie správcu skratiek okna.

Časť API pre klávesové skratky poskytuje nasledujúce dátové typy a metódy inštancie správcu skratiek okna:

- `Key` – enumerovaný dátový typ, ktorý obsahuje hodnoty reprezentujúce klávesy, ktoré nie sú modifikačné.
- `ModifierKey` – enumerovaný dátový typ, ktorý obsahuje hodnoty reprezentujúce modifikačné klávesy.
- `KeyCombination` – dátový typ reprezentujúci klávesovú kombináciu. Tento dátový typ má nasledujúce polia:
  - `key: Key` – hodnota reprezentujúca hlavný kláves kombinácie.
  - `modifiers: ModifierKey[]` – pole reprezentujúce modifikačné klávesy, ktoré musia byť aktívne aby sa stlačenie hlavnej klávesy počítalo ako aktivácia tejto klávesovej kombinácie.
- `addShortcut(id, name, description, callback, defaultKeybinds): boolean` – metóda inštancie `ShortcutManager`, ktorá vytvorí novú skratku pre okno, z ktorého bola zavolaná. Táto metóda vracia informáciu o tom, či bola skratka vytvorená úspešne alebo nie. Parametre tejto metódy sú nasledovné:
  - `id: string` – identifikátor skratky. Táto hodnota musí byť unikátna pre každú skratku okna.
  - `name: string` – názov skratky.
  - `description: string` – popis skratky.
  - `callback: () => void` – funkcia, ktorá má byť spustená po aktivácií skratky.
  - `defaultKeybinds: KeyCombination[]` – pole kombinácií kláves, ktorými má byť skratka po pridaní spustiteľná. Tento parameter nie je povinný.
- `removeShortcut(id)` – metóda inštancie `ShortcutManager`, ktorá odstráni zvolenú skratku z okna, z ktorého bola zavolaná. Táto metóda má nasledujúci parameter:
  - `id: string` – identifikátor skratky, ktorá má byť odstránená.

- `removeAllShorcuts()` – metóda inštancie `ShortcutManager`, ktorá odstráni všetky skraty z okna, z ktorého bola zavolaná.



Obr. 6.7: Časť architektúry, v ktorej je implementovaná funkcionalita správy profilov. Ukladanie profilu prebieha postupným serializovaním aktívnych modulov. Po dokončení celkovej serializácie sú profily zapísané na disk pomocou časti backend určenej na prácu so súborovým systémom. Pri načítaní profilov je naopak pomocou tejto časti backend získaný obsah súboru s profilmi z disku a aktívne moduly sú postupne spúšťané.

6.6 Práca s profilmi

Profily predstavujú celkové rozšírené grafické rozhrania vytvorené kombináciou modulov. Jednou z požiadavok na výslednú aplikáciu bola možnosť uloženia viacerých profilov. Okrem aktívnych modulov a rozloženia okien modulov je súčasťou profilu aj nastavenie klávesových kombinácií, ktoré aktivujú rôzne skratky okien modulov. Časti architektúry, v ktorých je práca s profilmi implementovaná sú ilustrované na obrázku 6.7.

Profily sú ukladané do súboru vo formáte JSON. Lokácia tohto súboru je odlišná pri použití nainštalovanej aplikácie a verzie aplikácie pre vývoj modulov. Pri použití nainštalovanej aplikácie sú profily uložené do súboru `profiles.json` v podpriečinku `config` priečinka, v ktorom je aplikácia nainštalovaná. V prípade použitia verzie aplikácie pre vývoj modulov sú profily uložené do súboru `src/config/profiles.json`. Tento súbor obsahuje JavaScript objekt, v ktorom sú jednotlivé profily uložené pod kľúčmi, ktorými sú ich názvy. Každý profil je reprezentovaný polom *serializovaných modulov*. Elementy tohto polia značia, ktoré moduly sú v profile aktívne a poskytujú bližšie informácie o nastavení týchto modulov. Každý serializovaný modul obsahuje atribút `moduleName` obsahujúci názov modulu, atribút `mainWindow` obsahujúci objekt s nastaveniami hlavného okna modulu a atribút `subwindows` obsahujúci pole objektov s nastaveniami otvorených podokien daného modulu.

Objekt nastavení okna obsahuje atribúty reprezentujúce titulok okna, informáciu o tom, či je okno zobrazené alebo minimalizované, súradnice polohy okna na obrazovke, rozmery okna a pole nastavení klávesových skratiek. Ak sa jedná o objekt nastavenia podokna, je doplnená informácia o názvu typu podokna modulu, ktoré daný objekt reprezentuje, informácia o tom, či je podokno transparentné a objekt obsahujúci parametre predávané podoknu pri otvorení pomocou dvojíc typu kľúč-hodnota. Pole nastavení klávesových skratiek obsahuje objekty s identifikátorom skratky a poľom reťazcov reprezentujúcich kombinácie kláves, ktoré majú danú skratku aktivovať.

Obsah tohto súboru je generovaný správcom modulov pri vyžiadaní vytvorenia nového profilu, aktualizácie existujúceho profilu alebo zmazania profilu. Správca modulov postupne

generuje obsah tohto objektu volaním serializačnej funkcie aktívnych modulov. Okrem súboru s profilmi, je používaný aj konfiguračný súbor `config.json`, ktorý sa nachádza v rovnakom priečinku ako súbor s profilmi. Tento súbor obsahuje JavaScript objekt v ktorom je pod kľúčom `activeProfile` uložený názov aktuálne používaného modulu. Táto informácia bola od súboru s profilmi oddelená pre zjednodušenie rozšírenia aplikácie v budúcnosti.

Keďže v aplikačnom rámci Tauri je každé okno spustené v osobitnom procese, každá práca s oknom trvá určitú dobu. Preto sú v tomto aplikačnom rámci všetky funkcie pre prácu s oknami asynchrónne. Keďže pri zmene modulu je potrebné načítať súčasne viacero okien a potom nastaviť ich vlastnosti, je potrebné počkať, kým sú špecifické okná inicializované. Toto spôsobovalo problém, pretože samotná informácia o tom, že okno dokončilo svoju inicializáciu je poslaná v podobe Tauri udalosti len do tohto novo vytvoreného okna. Podobný problém nastáva s informáciou o zavretí okna.

Tieto problémy boli vyriešené v triede `OverfloatModule` reprezentujúcej samotné moduly. Keďže všetky okná modulov sú otvárané pomocou metód tejto triedy, bolo možné vytvoriť dva prísľuby (*Promise*) pre každé okno, ktoré je otvorené. Tieto prísľuby sú vytvorené tak, aby sa automaticky nikdy nevyriešili, ale mohli byť vyriešené zavolaním špecifických funkcií. Jedna z týchto funkcií je potom pri vytváraní nového okna priradená obsluhe udalosti reprezentujúcej dokončenie inicializácie okna a druhá obsluhe udalosti reprezentujúcej zatvorenie okna.

Keďže inštancie tried reprezentujúcich aktívne moduly sú udržiavané v správcovi modulov, toto umožnilo synchronizovať časti načítavania profilu. Pri načítaní profilu najskôr prebehne rozoslanie požiadaviek na zatvorenie okna každému otvorenému oknu. Je potrebné počkať na správne ukončenie všetkých okien, na čo sú použité vyššie spomenuté prísľuby. Nastáva čakanie na vyriešenie všetkých prísľubov zatvorenia okna nasledované prechodom do fázy načítavania profilu.

Pri načítavaní profilu sú postupne aktivované potrebné moduly. Pri aktivácií modulu je vytvorené hlavné okno a všetky uložené podokná. Všetky tieto okná sú nastavené ako minimalizované, aby bolo obmedzené dočasné zobrazovanie okien, ktoré nemajú byť v profile zobrazené. Pri vytváraní okien sa zbierajú všetky prísľuby inicializovaní okien. Po tom, ako sú všetky moduly aktivované sa čaká na vyriešenie všetkých týchto prísľubov a teda na dokončenie inicializácie všetkých okien. Po tom, ako sú všetky okná inicializované, sú okná, ktoré majú byť viditeľné zobrazené pomocou metód správcu modulov.

6.7 Grafické rozhranie hlavného okna aplikácie

Hlavné okno aplikácie pozostáva z troch častí. Prvou z týchto častí je ovládacia lišta. Na tejto lište je umiestnené tlačidlo na zobrazenie nastavení profilov a modulov, tlačidlo na zobrazenie nastavení klávesových skratiek, a tlačidlá na ovládanie hlavných okien aktívnych modulov. Tieto tlačidlá majú odlišnú farbu podľa toho, či je dané okno aktuálne zobrazené alebo minimalizované. V prípade, že má konkrétny modul otvorené podokná, je pod tlačidlom na ovládanie hlavného okna zobrazené tlačidlo, ktoré umožňuje zobraziť tlačidlá na ovládanie podokien tohto modulu. Stlačenie tlačidla na ovládanie okna alebo podokna modulu dané okno zobrazí, v prípade, že bolo minimalizované, alebo minimalizuje, v prípade, že bolo zobrazené. Táto časť grafického rozhrania ovláda okná aktívnych modulov pomocou správcu modulov.

Druhou časťou tohto grafického rozhrania je zobrazenie nastavení profilov a modulov. Nastavenia profilov sa skladajú z ovládacích tlačidiel a elementu, pomocou ktorého je možné zvoliť jeden z existujúcich profilov. Ovládacie tlačidlá umožňujú zvolený profil načítať, ak-

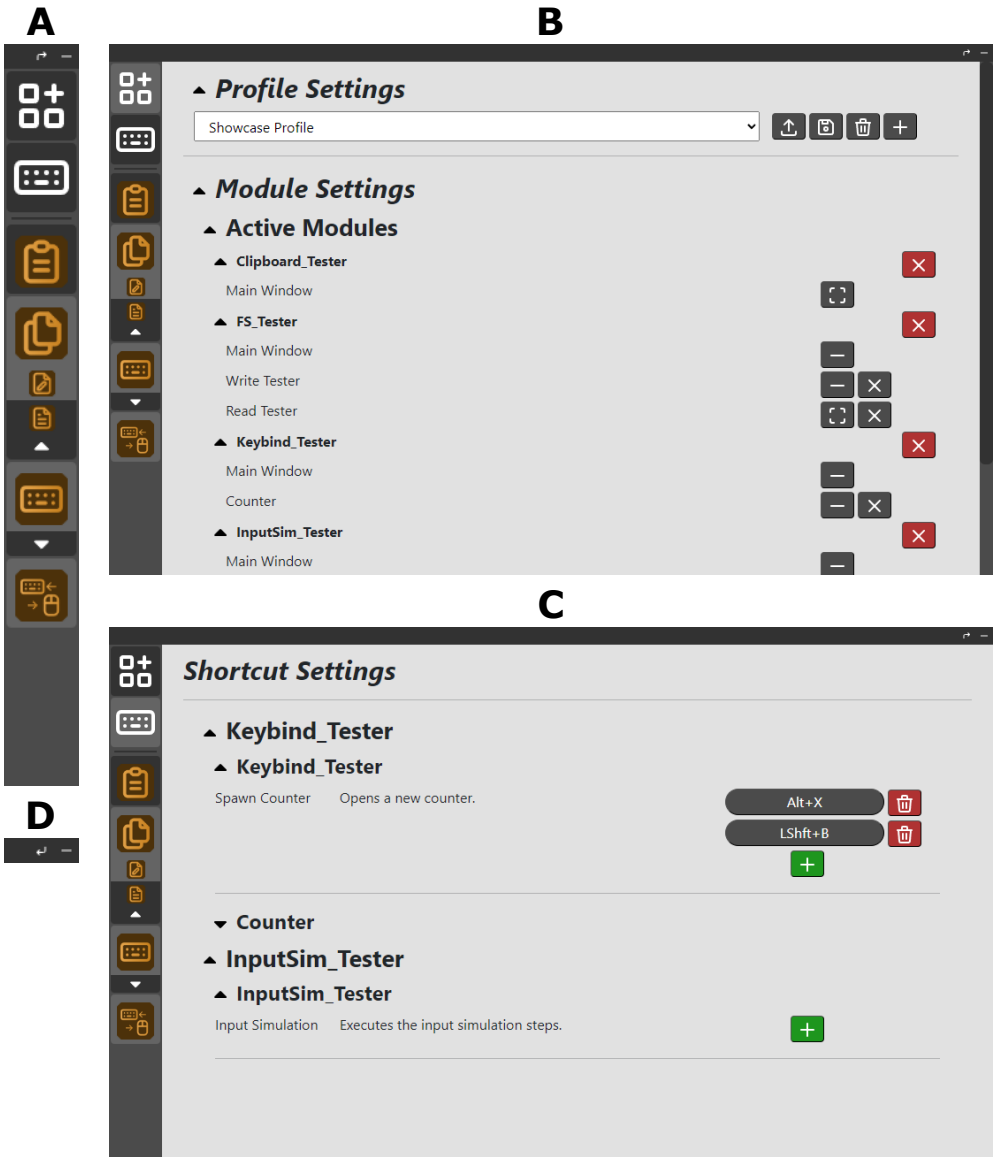
tualizovať, zmazať, alebo vytvoriť nový profil s aktuálny rozložením okien a nastavením klávesových skratiek. Nastavenia modulov sa skladajú z časti zobrazujúcej aktívne moduly a časti zobrazujúcej neaktívne moduly. Pri aktívnych moduloch sú zobrazené názvy modulov, tlačidlá na ich vypnutie a aktívne okná daných modulov. Pri každom aktívnom okne je zobrazený jeho titulok, tlačidlo na zobrazenie alebo minimalizovanie okna a tlačidlo na zatvorenie okna. Pri neaktívnych moduloch sú zobrazené názvy modulov a tlačidlá na spustenie týchto modulov. Podobne ako pri ovládacej lište táto časť grafického rozhrania pracuje so správcom modulov.

Poslednou časťou sú nastavenia klávesových skratiek. V tejto časti sú zobrazené aktívne moduly, ktoré ponúkajú možnosť nastavenia klávesových skratiek na vykonanie určitých akcií. Takto zobrazené moduly potom zobrazujú konkrétne okná, ktoré tieto skratky používajú. Pri každej skratke je uvedený jej názov, popis a klávesové kombinácie, ktorými ju je možné momentálne aktivovať. Každú z týchto klávesových kombinácií je možné zmeniť kliknutím na miesto, kde je zobrazená alebo zmazať kliknutím na tlačidlo na pravej strane. Pod týmto zobrazením aktuálne používaných klávesových kombinácií sa nachádza tlačidlo pre pridanie novej kombinácie kláves, ktorá má danú skratku aktivovať. Táto časť grafického rozhrania pracuje so správcom klávesových skratiek.

Pri názvoch častí, ktoré zobrazujú viac informácií sa nachádzajú šípky, ktorými je možné dané časti zabaliť alebo rozbaliť. Samotné okno aplikácie môže byť zobrazené buď ako samostatná ovládacia lišta, ovládacia lišta so zobrazením nastavení alebo len ako titulná lišta. Pre prepínanie medzi normálnym režimom zobrazenia a zobrazenia len hornej lišty je možné použiť jedno z tlačidiel v pravom hornom rohu okna. Hlavné okno aplikácie môže byť minimalizované použitím druhého z týchto tlačidiel. Po minimalizácií je možné toto okno opäť zobraziť pomocou ponuky v systémovej lište. Pomocou tejto ponuky je taktiež možné aplikáciu ukončiť. Rôzne spôsoby zobrazenia hlavného okna aplikácie sú ilustrované na obrázku 6.8.

Grafické rozhranie hlavného okna aplikácie je implementované v súboroch priečinka `src/components`. Pri vývoji boli použité jazyky TypeScript, CSS, HTML, knižnica React, rámec pre grafické rozhrania Bootstrap[6] a zbierka ikon Tabular Icons[7].

[6] https://getbootstrap.com/
[7] https://tabler.io/icons



Obr. 6.8: Grafické rozhranie hlavného okna aplikácie. Na obrázku sú viditeľné 4 spôsoby zobrazenia hlavného okna. A znázorňuje zobrazenie iba ovládacej lišty. B znázorňuje zobrazenie nastavení profilov a modulov. C znázorňuje zobrazenie nastavení klávesových skratiek. D znázorňuje zobrazenie len titulnej lišty.

Kapitola 7

Testovanie

Ako spôsob testovania bolo zvolené testovanie vytvorením niekoľkých modulov, ktoré sú zamerané na špecifické časti funkcionality aplikácie. Tento spôsob testovania umožní súčasne testovať implementáciu funkcionality a použiteľnosť API pri vytváraní modulov. Testovanie funkcionality prebehlo použitím vytvorených modulov na zariadení s operačným systémom Windows 11, verzia 10.0.22631 (*zariadenie A*) a zariadení s operačným systémom Linux, distribúcia Kubuntu Jammy Jelly 22.04 LTS (*zariadenie B*).

7.1 Modul na testovanie práce so súborovým systémom

Tento modul sa skladá z hlavného okna a dvoch typov podokien. Hlavné okno obsahuje dve tlačidlá, ktoré reprezentujú dva typy podokien tohto modulu a po stlačení otvárajú nové podokná týchto typov.

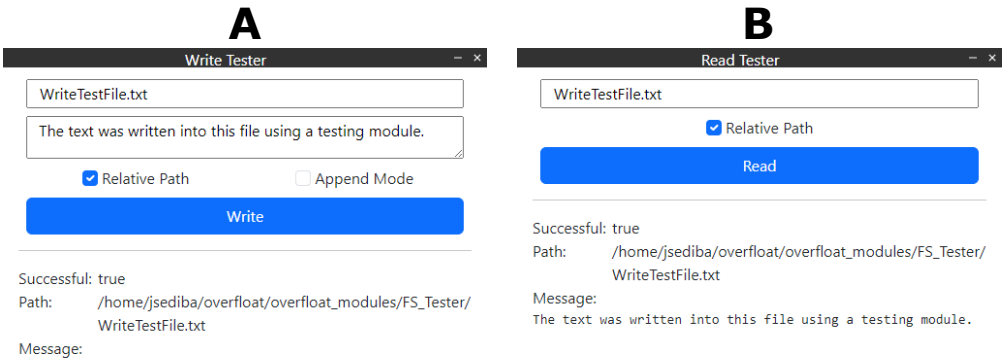
Prvý typ podokna je okno pre zápis. Toto okno obsahuje textové pole pre zadanie cesty k súboru, textové pole pre zadanie obsahu, ktorý má byť do súboru zapísaný a 2 zaškrtávacie políčka, ktoré reprezentujú posledné 2 argumenty API funkcie pre zápis. Jedná sa o zaškrtávacie políčko pre použitie relatívnej cesty a zaškrtávacie políčko pre ponechanie pôvodného obsahu súboru pri zápise. Pod týmito prvkami pre vstup sa nachádza tlačidlo, ktoré zavolá API funkciu pre zápis so zvolenými parametrami. Pod ovládacou časťou tohto okna sa nachádza priestor, do ktorého je vypísaný výsledok zápisu. Výsledok zápisu sa skladá z 3 častí: informácie o tom, či zápis prebehol úspešne, cesty k súboru, ku ktorému sa pristupovalo a správy, ktorá v prípade neúspešného zápisu obsahuje chybové hlásenie, ktoré vzniklo pri zápise.

Druhý typ podokna je okno pre čítanie. Toto okno obsahuje textové pole pre zadanie cesty k súboru, zaškrtávacie políčko pre použitie relatívnej cesty a tlačidlo, ktoré spustí API funkciu čítania súboru so zvolenými parametrami. Pod týmito ovládacími prvkami sa nachádza priestor pre výpis výsledku čítania. Podobne ako výsledok zápisu, sa tento výsledok skladá z informácie o úspechu čítania, cesty k súboru, ku ktorému sa pristupovalo a správy, ktorá obsahuje text súboru v prípade úspešného čítania alebo chybové hlásenie v prípade neúspechu. Snímky týchto podokien sú zobrazené na obrázku 7.1.

7.1.1 Priebeh testovania

Testovanie pomocou tohto modulu pokrýva zápis a čítanie súborov. Testované boli nasledovné prípady vrátane všetkých kombinácií relatívnej alebo absolútnej cesty a nastavenia ponechania pôvodného obsahu súboru pri zápise:

- zápis do neexistujúceho súboru,
- zápis do už existujúceho súboru, pre ktorý má používateľ zapisovacie práva,
- zápis do už existujúceho súboru, pre ktorý používateľ nemá zapisovacie práva,
- čítanie z neexistujúceho súboru,
- čítanie z už existujúceho súboru, pre ktorý má používateľ práva na čítanie,
- čítanie z už existujúceho súboru, pre ktorý používateľ nemá práva na čítanie.



Obr. 7.1: Snímky podokien modulu na testovanie práce so súborovým systémom. Podokno A slúži na testovanie zápisu do súboru a podokno B na čítanie zo súboru. V hornej časti týchto okien sú umiestnené ovládacie prvky na nastavenie parametrov funkcií pre prácu so súborovým systémom. V dolnej časti je zobrazený výsledok operácie.

7.1.2 Výsledky testovania

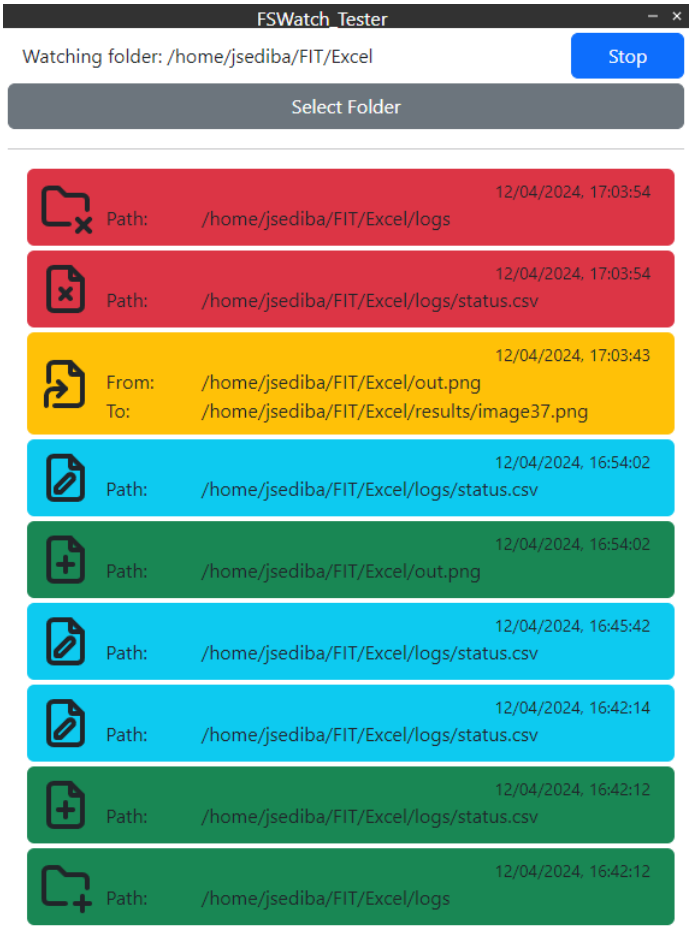
Všetky testy prebehli úspešne s očakávaným výsledkom. Pri snahe pristúpiť k súboru, ku ktorému používateľ nemá prístup je vo výsledku operácie nastavená informácia o úspechu na hodnotu `false` a v správe je korektný text chybového hlásenia.

7.2 Modul na testovanie monitorovania súborového systému

Tento modul sa skladá len z hlavného okna a slúži na testovanie funkcionality monitorovania súborového systému. V tomto okne je použitá funkcia `open`, poskytovaná aplikačným rámcom Tauri, ktorá zobrazí okno pre výber priečinka na monitorovanie. Okno tohto modulu obsahuje tlačidlo pre výber priečinka, ktoré používa spomenutú funkciu `open`, textové pole, v ktorom je vypísaná cesta k aktuálne monitorovanému priečinku a tlačidlo na zastavenie monitorovania. Pod týmito prvkami sa nachádza priestor do ktorého sú zobrazované udalosti, ktoré v danom priečinku počas monitorovania nastali.

Tieto udalosti sú zobrazované pomocou kariet, ktorých farba je určená typom udalosti, ktorá nastala. Červená farba značí zmazanie, zelená farba značí vytvorenie, modrá farba značí zmenu obsahu a žltá farba značí presun. Okrem farby, tieto karty obsahujú aj ikony z balíčka ikon Tabler Icons, ktoré bližšie identifikujú typ operácie a informáciu o tom, či sa jedná o priečinok alebo súbor. Dodatočne tieto karty zobrazujú čas, kedy udalosť nastala a

cestu k priečinku alebo súboru, ktorého sa udalosť týka. V prípade presunu sú zobrazené dve cesty: pôvodná cesta a nová cesta. Snímka okna tohto modulu je zobrazená na obrázku 7.2.



Obr. 7.2: Snímka okna modulu na testovanie monitorovania súborového systému. V hornej časti tohto okna sa nachádzajú tlačidlá na nastavenie a zmenu sledovaného priečinka. V dolnej časti sú zobrazené udalosti, ktoré počas monitorovania v sledovanom priečinku nastali. Tieto udalosti sú zobrazené v kartách, ktorých farby a ikony indikujú typ udalosti. Pri udalostiach je uvedená cesta, na ktorej udalosť nastala a čas, kedy udalosť nastala.

7.2.1 Priebeh testovania

Testovanie monitorovania súborového systému prebiehalo spustením modulu pre monitorovanie súborového systému, zvolením priečinka na monitorovanie a postupným vykonávaním zmien v tomto priečinku. V sledovanom priečinku boli vykonané nasledovné typy zmien:

- vytvorenie nového súboru,
- presun súboru,
- premenovanie súboru,
- úprava obsahu súboru,

- zmazanie súboru,
- vytvorenie priečinka,
- presun priečinka,
- premenovanie priečinka,
- zmazanie prázdneho priečinka,
- zmazanie priečinka obsahujúceho súbory.

7.2.2 Výsledky testovania

Premenovanie súboru a priečinka boli hlásené ako udalosť typu presun. Toto je však očakávaný výsledok a teda testy prebehli úspešne. Mazanie prázdneho priečinka prebehlo na zariadení B úspešne, no na zariadení A bola udalosť detekovaná ako zmazanie súboru. Toto je spôsobené tým, že implementácia sledovateľa súborového systému pre operačný systém Windows z crate Notify nešpecifikuje presnejší typ udalosti zmazania.

Pri mazaní priečinka obsahujúceho súbory, boli zaznamenané rozdielne sekvencie udalostí na zariadení A a zariadení B. Na zariadení A bola pre každý súbor v priečinku evidovaná udalosť zmeny obsahu priečinka nasledovaná udalosťou zmazania súboru. Po týchto udalostiach bola evidovaná finálna udalosť zmeny obsahu nasledovaná udalosťou zmazania. Tieto 2 finálne udalosti boli nesprávne klasifikované ako udalosti súboru. Toto je opäť spôsobené vyššie spomenutou chýbajúcou špecifikáciou typu udalosti. Na zariadení B boli zachytené udalosti vymazania súboru pre každý súbor v priečinku nasledované korektne evidovanou udalosťou vymazania priečinka. Neboli evidované žiadne udalosti zmeny obsahu priečinka.

Pri hlbšom testovaní sa ukázalo, že na zariadení B neboli generované udalosti zmeny obsahu priečinka. Udalosti presunu, vytvorenia a zmazania priečinka však boli generované správne. Udalosti súborov boli na oboch zariadeniach zachytené správne.

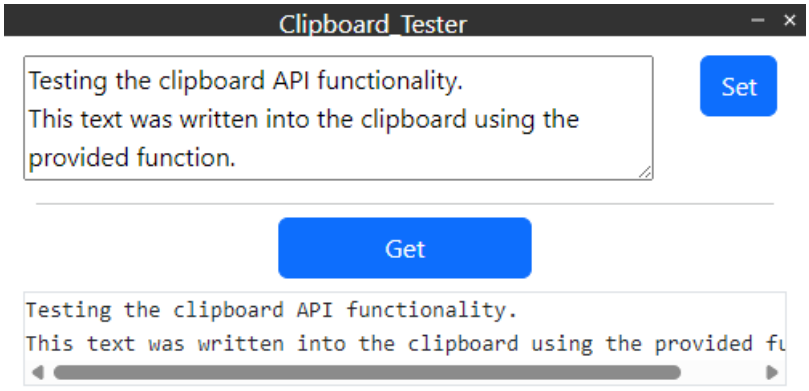
7.3 Modul na testovanie manipulácie schránky

Tento modul pozostáva len z hlavného okna, ktoré je rozdelené na dve časti. Prvá časť obsahuje textové pole pre zadanie obsahu, ktorý má byť do schránky vložený a tlačidla na jeho vloženie do schránky. Druhá časť obsahuje tlačidlo na získanie aktuálneho obsahu schránky a priestor pre výpis tohto obsahu. Snímka okna tohto modulu je zobrazená na obrázku 7.3.

7.3.1 Priebeh testovania

Testovanie práce so schránkou pozostávalo z nasledovných úkonov:

- získanie obsahu z prázdnej schránky,
- získanie obsahu zo schránky, ktorá obsahovala obsah z iného zdroja,
- zápis obsahu do schránky,
- použitie obsahu zapísaného do schránky v inom programe.



Obr. 7.3: Snímka okna modulu na testovanie práce so schránkou. Horná časť okna obsahuje priestor na zadanie textu, ktorý má byť do schránky vložený. Po zadaní textu je možné pomocou tlačidla vedľa tohto priestoru zadaný text vložiť do schránky. Dolná časť tohto okna obsahuje tlačidlo na získanie obsahu zo schránky. Po stlačení tohto tlačidla je obsah schránky vložený pod toto tlačidlo.

7.3.2 Výsledky testovania

Všetky testy prebehli úspešne s očakávaným výsledkom. V prípade snahy o získanie obsahu z prázdnej schránky bol vrátený prázdny reťazec.

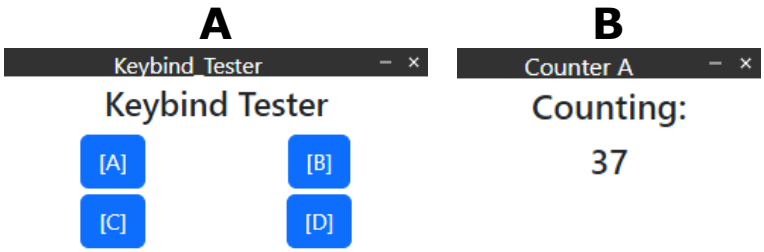
7.4 Modul na testovanie klávesových skratiek

Tento modul je určený na testovanie funkcionality globálnych klávesových skratiek a možnosti nastavenia hodnôt podokna pri jeho otváraní. Tento modul obsahuje hlavné okno a jeden typ podokna. Podokno tohto modulu je jednoduché počítadlo stlačení tlačidla, ktoré implementuje klávesovú skratku pre zvýšenie hodnoty počítadla. Klávesovú kombináciu, ktorá má túto hodnotu zvýšiť je možné nastaviť pri otváraní podokna. Hlavné okno obsahuje 4 tlačidlá na otvorenie okien počítadiel s prednastavenými klávesmi na zvýšenie hodnoty počítadla. Jedná sa o klávesy A, B, C a D. Okrem týchto tlačidiel hlavné okno implementuje aj klávesovú skratku na otvorenie nového počítadla bez prednastaveného klávesu pre zvýšenie hodnoty počítadla. Snímky okien tohto modulu sú zobrazené na obrázku 7.4.

7.4.1 Priebeh testovania

V tomto module boli testované globálne klávesové skratky. Testy pozostávali z nasledujúcich častí:

- otvorenie počítadla s prednastaveným klávesom na zvýšenie hodnoty pomocou tlačidla v hlavnom okne,
- stlačenie klávesovej kombinácie pre zvýšenie hodnoty počítadla keď je okno počítadla zobrazené a zamerané,
- stlačenie klávesovej kombinácie pre zvýšenie hodnoty počítadla keď je okno počítadla zobrazené ale nie je zamerané,



Obr. 7.4: Snímky okien modulu na testovanie funkcionality globálnych klávesových skratiek. Na snímke A je zobrazené hlavné okno, ktoré poskytuje 4 tlačidlá. Tieto tlačidlá otvárajú počítadlá stlačenia klávesu s prednastaveným klávesom. Na snímke B je zobrazené samotné podokno počítadla. V tomto okne je zobrazené číslo, ktoré sa zvyšuje pri stlačení nastavenej klávesovej kombinácie.

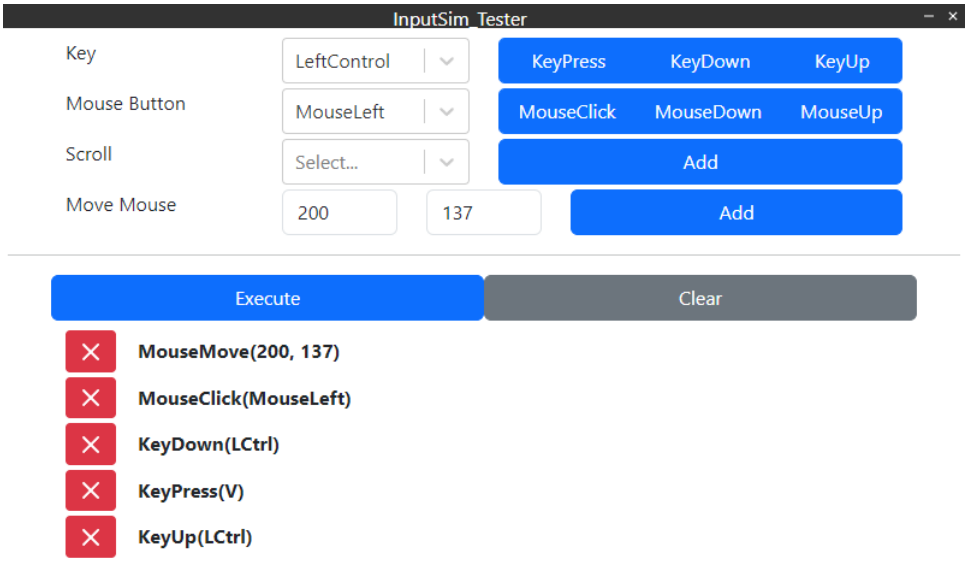
- stlačenie klávesovej kombinácie pre zvýšenie hodnoty počítadla keď je okno počítadla minimalizované,
- otvorenie nového počítadla bez nastaveného klávesu pomocou klávesovej skratky,
- zmena klávesovej kombinácie pre zvýšenie počítadla nasledovaná aktiváciou pôvodnej a novej klávesovej kombinácie,
- pridanie druhej klávesovej kombinácie pre zvýšenie hodnoty počítadla,
- vytvorenie nového profilu po zmene klávesových kombinácií,
- načítanie vytvoreného profilu a aktivácia pôvodných a upravených klávesových kombinácií.

7.4.2 Výsledky testovania

Pri testovaní sa ukázalo, že na zariadení A nie je možné zachytiť klávesové kombinácie ktoré boli stlačené so zameraným oknom iného programu, v prípade, že daný program bol spustený so správcovskými právami. Pri hlbšom testovaní bolo zistené, že takéto klávesové kombinácie je možné zachytiť v prípade, že aplikácia ja taktiež spustená so správcovskými právami. Ostatné testy prebehli na zariadení A úspešne. Na zariadení B prebehli všetky testy úspešne.

7.5 Modul na testovanie simulácie vstupov

Tento modul pozostáva len z hlavného okna. Toto okno obsahuje riadky na pridanie simulačných krokov pre klávesy, tlačidlá myši, rolovanie kolieska myši a presun ukazovateľa myši. Každý z týchto riadkov obsahuje prvky pre zvolenie parametrov pre tieto simulačné kroky. Pri simulovaní klávesov a tlačidiel myši riadky obsahujú 3 tlačidlá pre pridanie rôznych typov stlačení alebo uvolnenia klávesu alebo tlačidla myši do simulačnej sekvencie. Pri rolovaní kolieska myši a presune ukazovateľa myši riadky obsahujú len 1 tlačidlo pre pridanie týchto krokov do simulačnej sekvencie. Pod týmito riadkami sa nachádzajú tlačidlá na spustenie alebo vynulovanie zadanej simulačnej sekvencie. Pod týmito tlačidlami je zobrazená zadaná simulačná sekvencie a tlačidlá pre odstránenie jednotlivých prvkov zo zadanej simulačnej sekvencie. Snímka okna tohto modulu je zobrazená na obrázku 7.5.



Obr. 7.5: Snímka okna modulu na testovanie simulácie vstupných zariadení. Horná časť tohto okna slúži na zostavenie simulačnej sekvencie pridávaním krokov so zvolenými parametrami. Pod touto časťou sa nachádzajú tlačidlá na spustenie simulačnej sekvencie a vynulovanie zostavenej simulačnej sekvencie. Pod týmito tlačidlami sú zobrazené jednotlivé kroky simulačnej skevencie. Pri každom kroku je umiestnené tlačidlo, ktoré umožňuje jeho odstránenie zo sekvencie.

7.5.1 Priebeh testovania

Testovanie tohto modulu prebiehalo vytvorením niekoľkých simulačných sekvencií a ich následným spúšťaním. Testy pozostávali z nasledujúcich častí:

- simulovanie stlačenia klávesov.
- simulovanie aktivácie klávesových kombinácií pomocou stlačenia modifikačného klávesu bez uvoľnenia,
- simulovanie stlačenia sekvencie kláves pri držaní modifikačného klávesu stlačeného na klávesnici,
- simulovanie presunu ukazovateľa myši,
- simulovanie stlačenia tlačidiel myši,
- simulovanie presúvania pomocou kombinácie stlačenia tlačidla myši bez jeho uvoľnenia a presunu ukazovateľa myši,
- simulovanie rolovania kolieska myši.

7.5.2 Výsledky testovania

Všetky typy simulačných krokov fungovali očakávaným spôsobom. Pri presune ukazovateľa myši na záporné súradnice, alebo súradnice mimo obrazovku, bol tento ukazovateľ presunutý

korektne na hranicu obrazovky. Pri simulácií stlačení klávesov, bol korektne výsledný text závislý na zvolenom rozložení klávesnice. Pri simulovaní stlačenia klávesu v čase, kedy bol na klávesnici stlačený modifikačný kláves `Shift`, boli konkrétne znaky textu modifikované ako pri bežnom stlačení pod efektom klávesu `Shift`. Simulované stlačenia klávesových kombinácií správne spúšťajú klávesové skratky modulov tejto aplikácie. Toto umožňuje automatizáciu jedného modulu pomocou iného modulu. Keďže je podporované takéto spúšťanie klávesových skratiek, spustená simulačná sekvencie môže spôsobiť zacyklenie v prípade, že na spustenie tejto simulačnej sekvencie existuje klávesová skratka, ktorú aktivuje klávesová kombinácia simulovaná touto simulačnou sekvenciou.

Počas testovania sa podobne ako pri testovaní globálnych klávesových skratiek ukázalo, že na zariadení A neboli simulované stlačenia klávesov zachytené v aplikáciach spustených so správcovskými právami. Po hlbšom testovaní sa opäť ukázalo, že ak je aplikácia taktiež spustená so správcovskými právami, sú ňou simulované stlačenia klávesov zachytené aj v iných aplikáciách spustených so správcovskými právami.

7.6 Príklad reálneho využitia

Okrem modulov na testovanie špecifickej funkcionality aplikácie, bol vytvorený aj modul, ktorý znázorňuje príklad reálneho využitia aplikácie. Jedná sa o modul vytvorený na zjednodušenie obchodovania v počítačovej hre Path of Exile[1]. Tento modul pozostáva z hlavného okna a jedného podokna. Hlavné okno tohto modulu slúži na zadanie cesty k súboru, do ktorého táto hra generuje záznamy o udalostiach z hry. Pre voľbu cesty k tomuto súboru je podobne ako pri module pre monitorovanie súborového systému použitá funkcia `open`, ktorú poskytuje aplikačný rámec Tauri. Po zvolení cesty k tomuto súboru je pomocou funkcionality pre prácu so súborovým systémom vytvorený súbor, ktorý tento modul používa na uloženie zvolenej cesty na disk. Toto umožní, že po prvotnom nastavení cesty ju nie je potrebné pri nasledujúcich spusteniach modulu nastavovať znova. Hlavné okno tohto modulu taktiež obsahuje tlačidlo na otvorenie podokna tohto modulu.

Podokno tohto modulu slúži na zobrazovanie prichádzajúcich žiadostí o obchodovanie. Pri otváraní tohto podokna je pomocou API pre prácu s oknami nastavené, aby toto podokno malo priehľadné pozadie. Okrem toho je použité poskytovanie informácií pomocou parametrov v URL adrese pri otváraní podokna. Týmto spôsobom je podoknu poskytnutá cesta k súboru obsahujúcemu udalosti z hry. Toto podokno používa funkcionalitu monitorovania súborového systému na sledovanie súboru s týmito udalosťami a pri zmene obsahu reaguje na pridaný záznam. Tento pridaný záznam je skúmaný pomocou regulárnych výrazov a v prípade, že sa jedná o novú žiadosť o obchod, je táto žiadosť pridaná do zoznamu žiadostí.

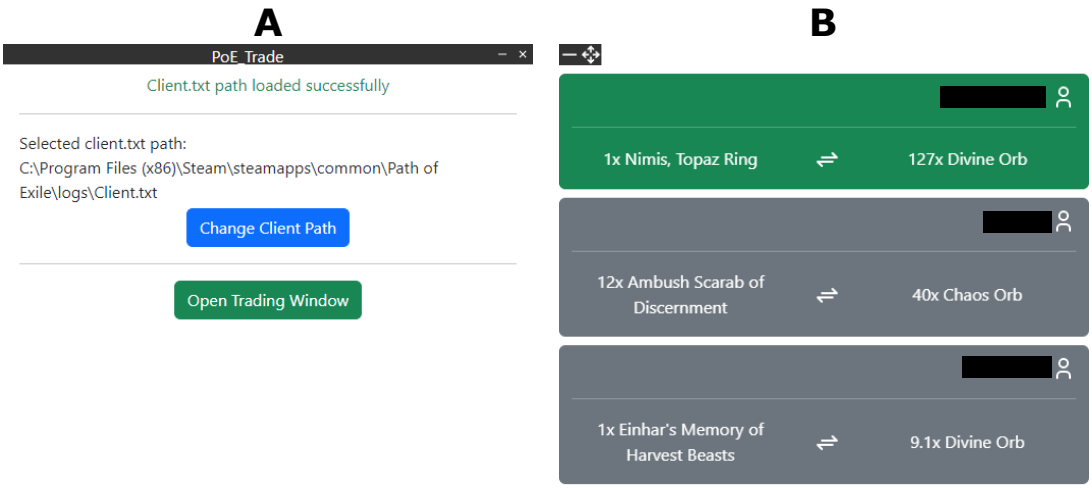
Všetky žiadosti o obchod zo zoznamu žiadostí sú vykreslené v podokne pomocou kariet, v ktorých sú zobrazené informácie o jednotlivých žiadostiach. Tieto žiadosti sú zobrazené v chronologickom poradí od najnovšej po najstaršiu. Pre navigáciu týmito žiadosťami je použitá funkcionalita globálnych klávesových skratiek. Pomocou tejto funkcionality je možné vybrať s ktorou žiadosťou sa aktuálne pracuje. Okrem výberu aktívnej žiadosti sú vytvorené aj klávesové skratky pre automatické pozvanie hráča, ktorý žiadosť o obchodovanie odoslal do skupiny, zahájenie obchodu a odstránenie žiadosti.

Samotná automatizácia týchto akcií je zabezpečená funkcionalitou simulácie vstupných zariadení. Pozvanie hráča do skupiny a zahájenie obchodu je možné v tejto hre vykonať

[1] https://www.pathofexile.com/

zadaním príkazu do okna pre správy. Tento príkaz je v zostavený v podokne modulu a vložený do schránky pomocou API pre prácu so schránkou. Následne je simulované stlačenie klávesu `Enter` na otvorenie okna pre správy nasledované simuláciou stlačenia klávesovej kombinácie `Ctrl + V` pre vloženie obsahu schránky a opätovné stlačenie klávesu `Enter` pre spustenie príkazu.

Tento modul kombinuje všetky časti funkcionality na poskytnutie prehľadnejšieho grafického rozhrania pre obchodovanie. Okrem toho tento modul zjednodušuje samotný priebeh obchodovania poskytnutím klávesových skratiek pre rôzne časti procesu obchodovania. Snímky okien tohto modulu sú zobrazené na obrázku 7.6



Obr. 7.6: Snímky okien modulu pre zjednodušenie obchodovania v počítačovej hre Path of Exile. Na snímke A je zobrazené hlavné okno modulu, pomocou ktorého je nastavená cesta k súboru, do ktorého táto hra zapisuje udalosti. Na snímke B je zobrazené podokno pre prichádzajúce žiadosti o obchodovanie. V tomto okne sú prichádzajúce žiadosti a informácie o nich zobrazené v chronologickom poradí. Aktuálne zvolená žiadosť je od ostatných odlíšená zelenou farbou pozadia. Mená hráčov boli v tomto obrázku cenzurované.

Kapitola 8

Záver

Cieľom tejto bakalárskej práce bolo vyriešiť problém nutnosti častého prepínania aktívnych okien pri práci vyžadujúcej informácie z externých zdrojov. Pri práci na tejto bakalárskej práci bola vytvorená multiplatformová aplikácia umožňujúca vytváranie modulárnych externých používateľských rozhraní zobrazených v plávajúcich oknách. Táto aplikácie umožňuje vývoj modulov pomocou webových technológií a poskytnutého aplikačno-programovacieho rozhrania. V oknách vytvorených modulov je možné zobrazovať informácie z externých zdrojov, použitím funkcií pre čítanie súborov alebo štandardných prístupov pre získavanie obsahu z externých webových stránok.

Okrem tejto funkcionality vytvorená aplikácie poskytuje úroveň automatizácie založenej na podpore globálnych klávesových skratiek pre aktiváciu funkcií, reakcií na zmeny v sledovaných častiach súborového systému a podpory simulácie vstupných zariadení. Vytvorená aplikácia ponúka alternatívu s otvoreným zdrojovým kódom k existujúcim riešeniam. Na rozdiel od existujúcich riešení, aplikácia podporuje okrem operačného systému Windows aj operačný systém Linux.

Použitie aplikačného rámca Tauri namiesto aplikačného rámca Electron.js viedlo k výrazne nižšej veľkosti výslednej aplikácie. Výsledná veľkosť spustiteľnej verzie vyvinutej aplikácie pre operačný systém Windows, ktorá obsahuje okrem hlavnej aplikácie aj 6 modulov, bola približne 8MB. V porovnaní s tým, spustiteľná verzia demonštračnej aplikácie používajúcej aplikačný rámec Electron.js, vytvorenej pomocou balíčka create-electron-app[1] je približne 170MB.

Vytvorená aplikácia má nasledovné nedostatky:

- Do aplikácie v súčasnom stave nie je možné pridávať nové moduly po inštalácií, pretože aplikačný rámec použitý pri vývoji túto funkcionalitu nepodporuje. V budúcnosti je plánované túto funkcionalitu pridať vytvorením Tauri balíčku, ktorý by takéto pridávanie modulov umožnil.
- V aktuálnom stave aplikácia pri operačnom systéme Linux podporuje len zobrazovací systém X11. Toto je spôsobené tým, že použité Rust balíčky na globálne zachytávanie udalostí klávesnice a udalostí súborového systému nepodporujú zobrazovací systém Wayland.
- Pri monitorovaní súborového systému v operačnom systéme Windows nie je možné rozlíšiť, či udalosť zmazania značí zmazanie súboru alebo priečinka. Pri operačnom

[1] https://www.npmjs.com/package/create-electron-app

systéme Linux nie sú pri monitorovaní priečinka zachytávané udalosti o zmene obsahu pod-priečinkov pri manipulácií s objektami súborového systému umiestnenými v nich.

Využiteľnosť tejto aplikácie bola znázornená v sekcii 7.6 vytvorením modulu pre zjednodušenie obchodovania v počítačovej hre Path of Exile. Tento modul používa funkcionalitu monitorovania súborového systému na získavanie informácií o prichádzajúcich obchodných ponukách zo súboru obsahujúceho históriu okna pre správy. Takto zachytené ponuky sú potom zobrazené v plávajúcom okne nad samotnou hrou, čo vedie k zvýšenej prehľadnosti. Následne tento modul pomocou kombinácie práce so schránkou a simulácie vstupných zariadení ponúka klávesové skratky pre pozvanie hráča, ktorý túto ponuku poslal, do skupiny a odoslanie žiadosti o zahájenie obchodovania s týmto hráčom.

Literatúra

[1] Arch Linux Community. *Window manager* [online]. 21. marca 2024 [cit. 2024-23-03]. Dostupné z: https://wiki.archlinux.org/title/window_manager.

[2] Arnold, K., Gosling, J. a Holmes, D. *The Java Programming Language*. 4. vyd. Addison-Wesley Professional, august 2005 [cit. 2023-10-14]. ISBN 978-0321349804.

[3] Bishop, J. a Horspool, N. Cross-Platform Development: Software that Lasts. *Computer*. Október 2006, zv. 39, č. 10, s. 26–35, [cit. 2023-12-7]. DOI: 10.1109/MC.2006.337. ISSN 0018-9162. Dostupné z: https://ieeexplore.ieee.org/document/1707631.

[4] Christensson, P. *Backend Definition* [online]. TechTerms.com, 11. apríla 2020 [cit. 2023-11-14]. Dostupné z: https://techterms.com/definition/backend.

[5] Christensson, P. *Frontend Definition* [online]. TechTerms.com, 18. apríla 2020 [cit. 2023-11-14]. Dostupné z: https://techterms.com/definition/frontend.

[6] Høgsberg, K. *Wayland* [online]. Intel Corporation, 2012 [cit. 2024-1-3]. Dostupné z: https://wayland.freedesktop.org/docs/html/.

[7] Microsoft. *Desktop window manager - win32 apps* [online]. Microsoft, Aug 2019 [cit. 2023-12-23]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dwm/dwm-overview.

[8] Microsoft. *Driver design guides for display, graphics, and compute accelerator devices - windows drivers* [online]. Microsoft, 23. februára 2023 [cit. 2023-12-23]. Dostupné z: https://learn.microsoft.com/en-us/windows-hardware/drivers/display/.

[9] Microsoft. *Progressive Web Apps (PWAs) with Microsoft Edge and Chromium* [online]. Microsoft, 24. januára 2023 [cit. 2024-03-04]. Dostupné z: https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/#pwa-benefits.

[10] Pickle, B. *Multiplatform Definition* [online]. TechTerms.com, 7. novembra 2022 [cit. 2023-11-14]. Dostupné z: https://techterms.com/definition/multiplatform.

[11] Russell, A. *Progressive Web Apps: Escaping Tabs Without Losing Our Soul* [online]. infrequently.org, 15. júna 2015 [cit. 2024-2-1]. Dostupné z: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/.

[12] Schechter, G. *How underlying WPF concepts and technology are being used in the DWM* [online]. Microsoft, 9. júna 2006 [cit. 2024-2-7]. Dostupné z: https://learn.microsoft.com/en-us/archive/blogs/greg_schechter/how-underlying-wpf-concepts-and-technology-are-being-used-in-the-dwm.

[13] Scheifler, R. W. a Gettys, J. The X Window System. *ACM Trans. Graph.* New York, NY, USA: Association for Computing Machinery. Apríl 1986, zv. 5, č. 2, s. 79–109, [cit. 2023-11-17]. DOI: 10.1145/22949.24053. ISSN 0730-0301. Dostupné z: https://doi.org/10.1145/22949.24053.

[14] Scoccia, G. L. a Autili, M. Web Frameworks for Desktop Apps: an Exploratory Study. In: ACM / IEEE. *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. New York, NY, USA: Association for Computing Machinery, 2020, s. 1–6 [cit. 2023-12-27]. ESEM '20. DOI: 10.1145/3382494.3422171. ISBN 978-14-50375-80-1. Dostupné z: https://doi.org/10.1145/3382494.3422171.

[15] Stuart, J. A. *A unified approach for cross-platform software development*. Reno, NV, USA, 2006. [cit. 2023-12-8]. Dizertačná práca. University of Nevada, Reno. ISBN 978-1-4503-7580-1. Dostupné z: https://www.proquest.com/dissertations-theses/unified-approach-cross-platform-software/docview/305275825/se-2.

[16] Tauri Contributors. *Process Model* [online]. 11. novembra 2022 [cit. 2023-11-4]. Dostupné z: https://tauri.app/v1/references/architecture/process-model.

[17] Tauri Contributors. *Inter-Process Communication* [online]. 8. marca 2023 [cit. 2023-11-4]. Dostupné z: https://tauri.app/v1/references/architecture/inter-process-communication/.

[18] Tauri Contributors. *Webview Versions* [online]. 22. júna 2023 [cit. 2023-11-4]. Dostupné z: https://tauri.app/v1/references/webview-versions.

[19] Vassallo, K. a Garg, L. Cross-platform development frameworks: overview of contemporary technologies and methods for cross-platform application development. In: ICCCM. *2nd International Conference on Computers and Management*. Kota, Rajasthan, India: ICCCM, 2016, s. 1–6 [cit. 2023-11-27]. Dostupné z: https://www.um.edu.mt/library/oar/handle/123456789/25282.