STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor: 18. Informatika

Vývoj uživatelsky přívětivého správce pracovního prostředí pro macOS

Martin Mikšík

Brno 2019

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

VÝVOJ UŽIVATELSKY PŘÍVĚTIVÉHO SPRÁVCE PRACOVNÍHO PROSTŘEDÍ PRO MACOS

DEVELOPING A USER-FRIENDLY MACOS WORKSPACE MANAGER

AUTOR	Martin Mikšík
ŠKOLA	Střední průmyslová škola a Vyšší odborná škola Brno, Sokolská, příspěvková organizace
KRAJ	Jihomoravský
ŠKOLITEL	Ing. Hodál Jaroslav, Ph.D.
OBOR	18. Informatika

Brno 2019

Prohlášení

Prohlašuji, že svou práci na téma *Vývoj uživatelsky přívětivého správce pracovního prostředí pro macOS* jsem vypracoval samostatně pod vedením Ing. Hodála Jaroslava, Ph.D. a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Dále prohlašuji, že tištěná i elektronická verze práce SOČ jsou shodné a nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a změně některých zákonů (autorský zákon) v platném znění.

V Brně dne: _____ _____

Martin Mikšík

Poděkování

Děkuji svému vedoucímu Ing. Jaroslavu Hodálovi, Ph.D. za podnětné připomínky a trpělivost, kterou mi během práce poskytoval.

Anotace

Cílem této práce je vytvořit nástroj pro zjednodušení přecházení mezi pracovními prostředími na macOS, který se bude od konkurence lišit svojí jednoduchostí používání a možností ukládat interní stavy aplikací.

Klíčová slova

správce oken; správce pracovní plochy; macOS; automatizace práce

Abstract

The main purpose of this work is to create a tool for switching workspaces on macOS, which will differ from the competitors by its easy to use and the ability to store internal application states.

Keywords

window manager; workspace manager; macOS; work automation

Obsah

Kapitola 1

Úvod

1.1 Motivace

Mojí motivací vytvořit tuto aplikaci byla ve skutečnosti lenost. I přesto, že se macOS pyšní intuitivním uživatelským rozhraním, které je vždy o krok napřed před uživatelem, jednu věc stále ve výchozím stavu neřeší. Tou je fakt, že různé úkony na počítači vyžadují rozličný set použitých aplikací či nastavení.

Tento problém řeší například společnost Adobe v programu Photoshop [1]. V pravém horním rohu si můžete ze seznamu vybrat například profil "essentials" – to změní rozložení pracovní plochy, zobrazí určité nástroje a skryje jiné. Máte tak připravené pracovní prostředí na jedno kliknutí (viz. obrázek 8.2 v příloze na straně 38).

I přesto, že se jedná o funkci, která šetří čas a zpříjemňuje uživateli používání aplikací v macOS, neexistuje její systémové řešení. Uživatel je tak nucen při změně pracovního prostředí (např. přechod z grafické práce na vývoj softwaru) jednotlivé aplikace zavřít popřípadě postupně otevřít. K tomu musí ještě uživatel přepnout nastavení pro posun modrého světla nebo třeba zapnout tmavý režim. To vše vyžaduje otevření systémového nastavení a proklikávání se složitými nabídkami.

Mým cílem je tedy vyvinout aplikaci, která implementuje[1] funkcionalitu podobnou té v programech od firmy Adobe, ale na celosystémové úrovni.

1.2 Konkurence

Během vývoje jsem narazil na spoustu aplikací, které jsou nepřímou konkurencí. Jsou to různé utility[2] pro správu oken. Například jako HyperDock, který do macOS přináší náhledy oken otevřených aplikací v Docu a aktivní okraje obrazovky [2]. Jedinou přímou konkurenci, kterou jsem byl schopen najít, je aplikace se jménem Workspaces. Nabízí podobnou funkci, kdy dávkově otevře aplikace. Nevýhodou je, že neobnovuje jejich stav. Workspaces nebyla už přes rok aktualizována a kvůli uzavřenému zdrojovému kódu je tak neznámo, jestli vývoj bude pokračovat [3]. Tabulka 1.1 přehledně uvádí klady a zápory jednotlivých aplikací.

	HyperDock	Workspaces	Moje aplikace
Dávkové otevírání aplikací	✗	✓	✓
Uložení interního stavu aplikací	✗	✗	✓
Správa otevřených aplikací	✓	✗	✓
Změna nastavení systému	✗	✗	✓
Umístění na AppStore	✓	✓	✗
Otevřený zdrojový kód	✗	✗	✓

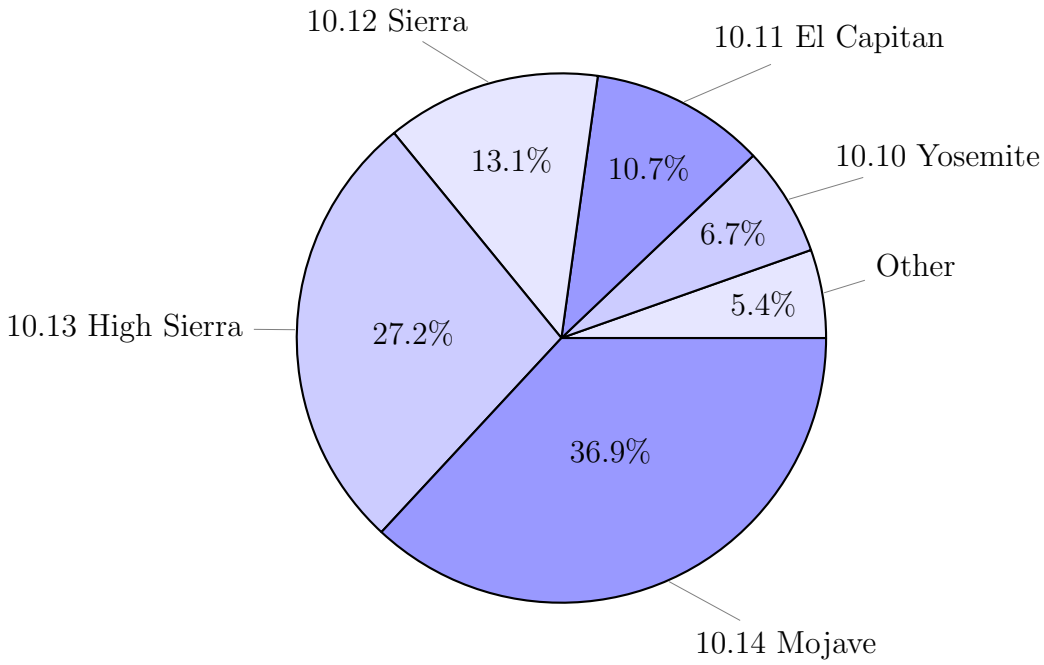
Tabulka 1.1: Porovnání vytvořené aplikace s konkurencí [2] [4]

[1] Implementace neboli realizace znamená naprogramování dané funkce nebo sady funkcí a jejich zavedení do programu.

[2] Utilita je program, který svou funkcí šetří čas uživateli.

1.3 Požadavky na počítač

Hardwarové požadavky na systém jsou minimální. Program po spuštění zabírá v rozmezí 20 - 30 MB paměti. To je zcela běžné pro aplikaci na macOS. Z hlediska software je program zkompilován pro operační systémy macOS 10.13 a vyšší. Tuto verzi jsem zvolil, abych mohl používat nejnovější aplikační rozhraní a jak lze vidět v tabulce 1.1 10.13 a 10.14 tvoří v současnoti 63 % všech aktivních instalací macOS. Verzi 10.13 je možno nainstalovat na všechny počítače společnosti Apple vyrovené po roce 2009 a tedy lze prohlásit, že moje aplikace je kompatibilní se všemi počítači se systémem macOS vyrobenými po tomto roce.



Obrázek 1.1: Aktivní instalace macOS v roce 2019 [5]

Kapitola 2

Základní pojmy

2.1 Spaces

Virtuální plocha je termín používaný pro označení funkce uživatelského rozhraní, která vytváří dojem, že je k počítači připojeno více fyzických monitorů. Uživatel tak může na jednotlivé plochy roztřídit svoje aplikace. Mezi jednotlivými plochami se přepíná například pomocí klávesových zkratek. Ukázka virtuálních ploch z macOS je na obrázku 2.1 na straně 12. Ten implementuje virtuální plochy od verze 10.5 pod názvem Spaces [6].

2.2 Dock a systémová lišta

V macOS je systémová lišta podobná té z Windows rozdělena na dvě části. První část dock zobrazuje aktivní aplikace a nachází se ve výchozím stavu na spodní části obrazovky [8]. Druhá systémová lišta se vždy nachází v horní části obrazovky a zobrazuje čas, stav signálu wifi apod. Lze ji přirovnat k System Tray z Windows.

Během vývoje mé aplikace jsem zjistil, že aplikace Dock dělá více než se zdá. Například přepínání virtuálních ploch nebo jejich vytváření i samotný posun oken mezi plochami je spravován právě aplikací Dock. Bohužel samotný Dock ve výchozím stavu nenabízí žádné veřejné aplikační rozhraní.

Obrázek 2.1: Ukázka přehledu virtuálních ploch v macOS [7]

Na jiných než macOS platformách by toto byl jen těžko řešitelný problém. Výhoda macOS tkví v jazyce Objective-C, ve kterém pomocí jeho specifických vlastností lze tento problém vyřešit. Běhové prostředí jazyka Objective-C totiž umožňuje již spuštěným aplikacím změnit implementaci tříd. Můžeme je tak upravit nebo přidat další metody. Tomuto se říká "Method Swizzling". Obecně se jedná o nedoporučovaný postup, ovšem není-li lepší cesty, tak je to jedno z možných řešení.

Nevýhodou je, že pro použití této metody musí uživatel vypnout SIP – System Integrity Protection, což vyžaduje restart systému a spuštění v nouzovém režimu. Pro běžného uživatele nesplnitelný krok. [9]

2.3 Quartz – X Server v macOS

MacOS má své kořeny v operačním systému NextStep, který byl vyvíjen společností Next pod vedením Steva Jobse poté, co byl vyhozen v roce 1985 z práce. Na počítačích NextStep byl spuštěn například první webový server.

Samotný NextStep OS[1] vychází z Unixu[2]. Unixové systémy implementují svoje grafické rozhraní pomocí softwarové vrstvy zvané x window server (dále X11), která se nejen stará o interakci s grafickým hardware, ale také o zpracování vstupů a výstupů z klávesnice. MacOS implementuje vlastní obdobu X11 s názvem Quartz. Některé části mé aplikace tedy přímo interagují subsystémem Quartz – Quartz2D také známý pod jménem Core Graphics [10].

Samotná knihovna je velmi obsáhlá a nemá smysl zde zabíhat do podrobností. Co je ale důležité mít na mysli je to, že některé specifické funkce, které moje aplikace využívá, nejsou a nejspíš ani nebudou zdokumentované a běžně dostupné. Většinou se jedná o funkce, které jsou na rozdíl od těch v Docku nízkoúrovňové. Jako příklad si uveďme třeba funkci, která navrací seznam oken pro každou pracovní plochu.

[1] OS je skratka pro operační systém.

[2] Unix je rodina operačních systému. Architekturou jsou si tedy podobné.

Kapitola 3

Nástroje

3.1 Xcode

Xcode je unifikované vývojové prostředí pro zařízení od společnosti Apple. V základě tedy obsahuje pokročilý editor zdrojového kódu, který umí například našeptávání nebo zvýraznění kódu. Obsahuje také systémové knihovny nebo kompilátor – nástroj převádějící kód do strojového jazyka z programovacích jazyků C, C++, Objective-C a Swift, tak i debugger, nástroj který umožňuje například prozkoumat hodnoty proměnných za chodu aplikace [11]. Soubor nesoucí informace o projektu obdobný těm jako ".solution" nebo "MakeFile" má příponu ".xcodeproj" popřípadně ".xcworkspace".

3.2 Objective C

Objective-C byl po dlouhou dobu primárním programovacím jazykem pro macOS, iOS. Jeho existence se datuje do roku 1981 a byl popularizován zejména díky NextStepu a následně společnosti Apple. Syntaxí se značně liší od většiny dnes používaných jazyků, které se v převážné většině podobají jazyku C++. Objective-C je "dynamický" jazyk. Tedy nastavení datových typů nebo výběru implementace metody třídy nastává za běhu

aplikace. Je tedy zcela validní zkompilovat program, pro který například daná metoda ve třídě není implementována. Rozřešení těchto závislostí za běhu programu zajišťuje knihovna Objective-C Runtime[1], která je přítomná ve všech Objective-C programech. Tyto specifické vlastnosti samozřejmě nesou svoje pro i proti. Zásadní výhodou takové dynamičnosti jsou například univerzální metody, které mohou přijímat číslo v celočíselné i desetinné podobě. Tou zásadní nevýhodou je o něco pomalejší běh programu oproti jazykům statickým.

3.3 Swift

Swift byl představen roku 2014 jako náhrada jazyka Objective-C. Svojí syntaxí je více podobný jazyku C++ nebo Java nežli Objective-C. Také neobsahuje dynamické vlastnosti Objective-C. Tedy datové typy i výběr metod, které jsou volány, jsou určeny během kompilace. Samotný jazyk svým návrhem odpovídá 21. století a implementuje nejnovější trendy v počítačové vědě [12]. Kompletní zdrojový kód implementace jazyka je zveřejněn na GitHubu[2].

3.4 AppleScript

AppleScript je jednoduchý skriptovací jazyk pro automatickou interakci se systémem macOS a jeho aplikacemi. Jeho syntaxe je velice podobná přirozené anglické řeči.

[1] Běhové prostředí poskytuje potřebné knihovny a dílčí programy, které umožňuje chod aplikace.

[2] https://github.com/apple/swift

3.5 Automator

Je automační nástroj pro macOS. Svými funkcemi se podobá jazyku AppleScript, ale nabízí grafické rozhraní s prvkem drag'n'drop[3].

[3] Doslovný překlad zní "táhni a pusť". Na počítači nejčastěji využijeme tohoto principu při práci se soubory. Myší jeden označíme a při držení pravého tlačítka přesuneme do požadované složky. Ovšem nemusí se jednat pouze o soubory.

3.6 CocoPods

CocoPods je balíčkovací manažer pro jazyk Objective-C a Swift. Pro jeho použití stačí vytvořit soubor "Podfile", ve kterém se nadefinují požadované knihovny, a v dané složce se v terminálu zavolá příkaz "pod install".

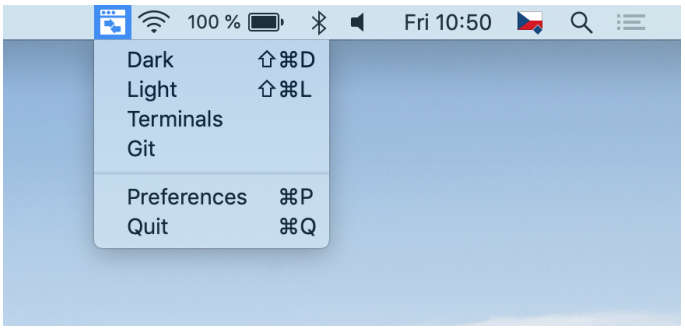
3.7 Seznam použitých frameworků třetích stran

- Realm: Populární multiplatformní databázový systém.
- ShellOut: Knihovna pro práci s příkazovým řádkem.
- KeyHolder: Knihovna obsahující kontrolní prvek pro zaznamenávání klávesové zkratky.

Kapitola 4

Seznámení s aplikací

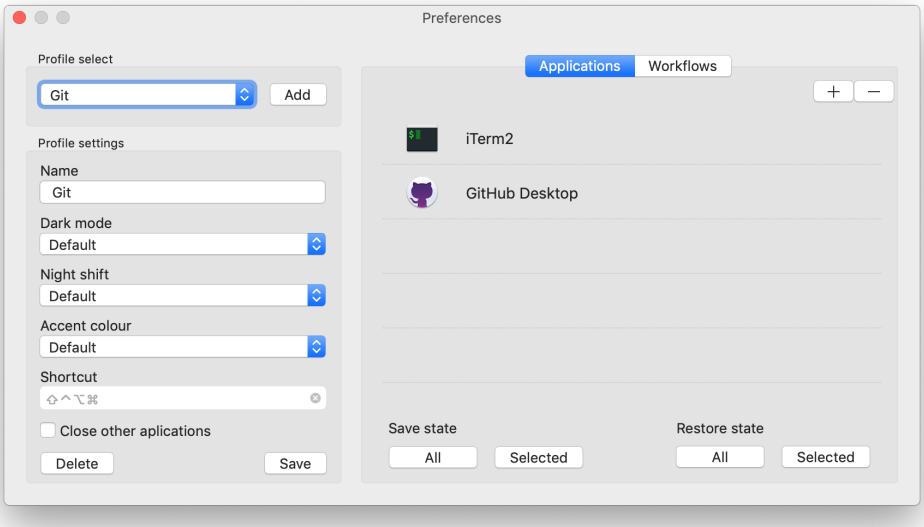
Jádrem aplikace jsou profily vytvořené samotným uživatelem. Profily si lze představit jako záložky v knížce, které vám umožní pokračovat v používání plochy ve stavu, v jakém jste si ji uložili. Zástupce aplikace se nachází v systémové liště a po jeho rozkliknutí se zobrazí seznam již definovaných profilů, jejich klávesové zkratky a tlačítko okna předvolby (viz. obrázek 4.1). V tomto okně lze profily zakládat, mazat a upravovat. Okno je rozděleno na tři boxy, které jsou graficky odděleny (viz. obrázek 4.2 na straně 19).



Obrázek 4.1: Rozkliknutý zástupce aplikace v systémové liště

V prvním boxu se pomocí rozbalovací nabídky vybírá profil k editaci. Nachází se zde i tlačítko pro přidání profilu nového. V druhém boxu se nastavují základní charakteristiky profilu – jméno profilu, vzhledové

atributy systému po načtení profilu nebo klávesová zkratka pro jeho načtení. Vzhled celého systému lze ovlivnit aktivací světlého nebo tmavého režimu, akcentní barvou nebo zapnutím redukce modrého světla. Pro každý z těchto atributů má uživatel na výběr z hodnot zapnout, vypnout a výchozí, kde hodnota výchozí zanechá současné nastavení systému. V neposlední řadě se ve druhém boxu nachází ještě zaškrtávací políčko "zavřít ostatní aplikace", jehož důsledkem je uzavření všech ostatních aplikací při načtení profilu, a tlačítko pro smazání profilu. V případě jeho zmáčknutí musí uživatel v nově otevřeném modálním okně akci potvrdit (viz. obrázek 8.1 v příloze na straně 37).



Obrázek 4.2: Rozdělení okna předvolby

Třetí box obsahuje dvě záložky. Na záložce "aplikace" jsou vypsány všechny aplikace přiřazené k profilu. Přiřazené aplikace jsou ty, které se po obnovení profilu spustí z předem uloženého stavu. Pro snazší orientaci je seznam seřazený abecedně. Na jednotlivých řádcích se vždy prvně nachází ikona aplikace a pak vedle ní jméno aplikace.

Do seznamu se aplikace přidají kliknutím na tlačítko "+" nad

seznamem. Otevře se dialogové okno průzkumníka souborů, kde lze vybrat pouze soubor s příponou ".app" (viz. obrázek 8.3 v příloze A na straně 39). Vedlejším tlačítkem "-" se odebírají označené aplikace ze seznamu. V seznamu lze označit jednu nebo více aplikací kliknutím na řádek seznamu a tažením kurzoru myši. Pokud je pro aplikaci už uložen nějaký stav, je před odebráním uživatel dotázán na potvrzení operace. Pod seznamem se nachází dvě dvojice tlačítek.

Levá dvojice s titulkem "uložit stav" se stará o ukládání stavů přiřazených aplikací. Pravá dvojice s titulkem "obnovit" se stará analogicky o jejich obnovení z uloženého stavu. Vždy první tlačítko z dvojice vyvolá danou akci pro všechny aplikace ze seznamu. Druhé vyvolá danou akci pouze pro aplikace označené.

Záložka Workflows je skoro identická k první záložce, až na pár rozdílů. V dialogovém okně lze vybrat pouze soubory s příponou ".workflow", což jsou skripty generovány z automatizačního prostředí Automator. Tyto skripty například dokáží vysypat koš, seřadit soubory podle určitých atributů nebo změnit pozadí plochy. Značně se tak rozšiřuje možnost každého profilu, ale i celé aplikace. Spodní dvojice tlačítek nemají na workflow vliv.

V příloze od strany 41 jsou zobrazeny aktivace několika vzorových profilů. Lze tak vidět, jak aplikace dokáže změnit systémové nastavení nebo obnovit stav či rozložení oken aplikací. Videa zobrazující používání aplikace lze nalézt na adrese http://bit.ly/desktop-profiles-demo nebo po naskenování qr kódu na obrázku 4.3.

Obrázek 4.3: QR kód obsahující ukázková videa aplikace

4.1 Mezinárodní lokalizace

Aby moje aplikace mohla oslovit co největší množství lidí, rozhodl jsem se implementovat anglickou i českou lokalizaci. Nastavení jazyka se při spuštění provede automaticky podle jazyka systémového, a pokud není dostupný překlad, zvolí se výchozí Angličtina. V současnosti je k dispozici pouze český překlad. Vytvořit další lokalizace ale není komplikované [13].

4.2 Uživatelská oprávnění

Při prvním spuštění je uživatel vybídnut, aby povolil aplikaci spouštění programů ve scriptovacím jazyce AppleScript a přístup k částem aplikačního rozhraní, které například umožňuje pohybovat s okny po ploše [14].

Kapitola 5

Implementace

5.1 Výrazy v programovacím jazyce Swift

- Protokol: objekt definující metody třídy bez jejich samotné implementace. Každá třída může implementovat neomezený počet protokolů [15].
- Extension (rozšíření): přidává novou funkcionalitu do již existující třídy nebo protokolu [16].
- Entita: Představuje řádek v databázové tabulce, se kterým se pracuje jako s běžným objektem. Všechny entity v realmu(databázový systém, který používám) musí podědit objekt "Object" z modulu Swift Realm [17].
- Enum: Je datový typ vytvořen výčtem pojmenovaných identifikátorů s přiřazenými hodnotami. Většinou jsou přiřazené hodnoty celočíselné, ale může jít například i o textový řetězec [18].

5.2 Popis mého řešení

Protože toto bylo poprvé, co jsem programoval pro macOS, nebyl jsem tak nijak obeznámen s jejich vývojovým prostředím ani aplikačním rozhraním.

Ještě před začátkem implementace jsem začal extenzivní rešerši splnitelnosti definovaných cílů a jejich možné implementace. Získané teoretické předpoklady jsem ověřoval na jednoduchých programech, které se postupně více a více podobaly finálnímu provedení. Během této fáze vývoje ale přišlo i několik zlomových bodů, které značně zkomplikovaly a zdržely celý vývoj. Předně to bylo zjištění, že pro macOS neexistuje udržovaná knihovna pro práci s virtuálními plochami. To je nejspíše z důvodu postupného uzavírání aplikačního rozhraní ze strany Applu. Tento problém se mi nakonec podařilo překonat.

Při návrhu finální aplikace jsem začal diagramem (zobrazen na obrázku 8.4 v příloze na straně 40), do kterého jsem postupně zanášel požadované funkce. Z výsledného diagramu bylo relativně jednoduché syntetizovat jednotlivé protokoly pro třídy a entity pro databázi. Jako databázový systém jsem zvolil populární nástroj Realm, především kvůli jeho jednoduchosti. Při jeho používání jsem se ale setkal s několika nepříjemnostmi. Realm není "thread-safe", což znamená, že k databázi nejde přistupovat z více vláken najednou. Toto se ukázalo jako překážka zejména proto, že navrácené entity z databáze jsou s databází stále interně spojeny. I když data z databáze získáte v hlavním vlákně aplikace, nemůžete k nim přistupovat ve vlákně jiném. Toto mě donutilo implementovat si vlastní "deep-copy", tedy metodu která projde danou entitu a každý její prvek zkopíruje. Tady se výhodou stala velká komunita okolo Realm, jelikož nespočet lidí řeší stejný problém jako já. Další problém nastal, když jsem si skrze svoji aplikaci vytvářel instance Realmu. Zjistil jsem, že nedochází k jejich správnému dealokování. To vše mě vedlo k vytvoření singleton třídy "Facade", třídy která má pouze jednu statickou instanci, a z ní k Realmu přistupovat.

Pokračoval jsem implementací entity Profile, kterou lze vidět ve výpise 5.1 na straně 24. Ta si udržuje záznam o jménu a dalších atributech o profilu formou textu nebo celého čísla. Přiřazené Aplikace a Workflows si udržuje jako pole entit. Takto definovaná entita je zcela validní pro Realm.

```
1  class Profile: BaseEntity {
2      @objc dynamic var name = ""
3      @objc dynamic var darkMode : Options = .keep
4      @objc dynamic var nightShift : Options = .keep
5      @objc dynamic var accentColour : Color = .keep
6      @objc dynamic var closeOtherApps : Bool = false
7      @objc dynamic var shortcutKeyCode : Int = -1
8      @objc dynamic var shortcutModifierCode : Int = -1
9      let apps = List<App>()
10     let workflows = List<Workflow>()
11 }
```

Výpis kódu 5.1: Implementace entity Profile

Z diagramu mi ale vyšlo, že výhodné by bylo, abych byl schopen nad profilem zavolat funkci pro načtení profilu nebo uložení stavu pro přiřazené aplikace.

Napsal jsem si proto vlastní rozšíření třídy – extension, ve které si potřebné metody dodefinovávám. Tento přístup jasně odděluje definici entity od přídavných funkcí, které jsou nezávislé na použitelném databázovém systému. Kód je tak jasně čitelný a do budoucna umožňuje případně jednoduchý přechod na jiný databázový systém jako je třeba CoreData[1] přímo od společnosti Apple. Zvláště zajímavá je například implementace metody copy, kterou lze vidět ve výpise 5.2 na straně 25. Ta přijímá jakýkoliv datový typ, který je kolekcí implementující protokol Sequence a zároveň datový typ elementu této kolekce je entita App.

Dále je zajímavá metoda restoreSettings, která má na starosti úpravu vzhledu systému. Na základě definovaných hodnot se postupně spustí dílčí skripty napsané v jazyce AppleScript, který je ideálním jazykem pro jednoduchou interakci se systémem. Protože samotný běh skriptu vzhledem k interakci s uživatelským rozhraním může být pomalejší, jejich spuštění probíhá ve speciální frontě a tedy samostatném vlákně. Zdrojový kód

[1] https://developer.apple.com/documentation/coredata

```
1  extension Profile {
2      func copy<T: Sequence>(apps: T) where T.Element == App
       ↪  {
3          for app in self.apps {
4              try! app.data.copy()
5          }
6      }
7  }
```

Výpis kódu 5.2: Implementace metody copy

skriptu, který mění akcentní barvu je ve výpise 5.3 na straně 26.

V přiloženém skriptu se nejprve převede argument z příkazovém řádku do proměnné setTo. Dále se otevře aplikace Předvolba systému (System Preferences) a přepne se do záložky obecné. Následuje malá prodleva, která má zajistit, že aplikace bude na záložce obecné i na pomalejších počítačích. Následně se využije rozhraní System Events, které umožňuje simulovat kliknutí myši a psaní klávesnice. Tomuto rozhraní se předá informace o tom, aby v aplikaci předvoleb simulovalo kliknutí na zaškrtávací políčko n, kde n je rovno hodnotě ze setTo.

Entita aplikace si uchovává informace přiřazené aplikaci a je opět rozšířena o další metody. Jsou to metody pro zavření a otevření aplikace. Pro ukázku: zavírání aplikace probíhá tak, že za pomoci příkazu pgrep načtu "process identificator"(PID) [19] aplikace, což je unikátní číslo pro každý proces/sy a následně se zavolá příkaz kill s tímto číslem jako parametrem. K tomuto řešení jsem přistoupil poté, co nastávalo chybné uzavření aplikací původní metodou. Zjistil jsem totiž, že některé aplikace vytváří více procesů, které se dříve nezavíraly. Pro otevření aplikací se využívá příkaz "open -a". Parametr "-a" značí, že chci otevřít aplikaci. Platí zde ale jedna vyjímka. Tento příkaz nelze zavolat nad aplikací Finder. Došlo by totiž k poškození systému a byl by nutný restart. Pakliže zavřu aplikaci Finder, systém sám okamžitě vynutí její opětovné spuštění.

```
1  on run arg
2      set setTo to arg's first item
3      tell application "System Preferences"
4          activate
5          reveal pane "com.apple.preference.general"
6          delay 1
7      end tell
8      tell application "System Events" to tell process "
       ↪ System Preferences"
9          click checkbox setTo of window "General"
10     end tell
11 end run
```

Výpis kódu 5.3: Skript pro změnu akcentní barvy

Kdybych potom znovu zavolal "open", Finder se otevře dvakrát. Což by způsobilo nedefinované chování a druhý proces by nešlo zavřít, protože systém ho opakovaně otevírá.

Rozšíření také definuje dynamicky vyhodnocenou proměnou stateData, která odpovídá protokolu definující funkce copy, restore a clean. Tyto metody se používají pro ukládání a obnovu stavu a rozmístění oken aplikace. Během rešerše jsem přišel na to, jak obnovit stav prakticky jakékoliv aplikace a rozmístění jejich oken na dané ploše.

Využívám k tomu systémovou funkci, která při uzavření aplikace uloží obraz paměti využívané aplikací do systémové složky [20]. Pro většinu aplikací stačilo tedy vytvořit funkci, která serializovaná data zkopírovala pro zvolený profil. Zde ale nastala komplikace, kdy například populární aplikace Google Chrome ukládá svůj stav jinak [21]. Vytvořil jsem si proto programové rozhraní definující zmíněné metody copy, restore, clean. Pro drtivou většinu aplikací se potom využívá výchozí implementace a pro ostatní aplikace lze jednoduše doimplementovat požadované chování (viz. výpis kódu 5.4). Jaké třídě se ve finále vytvoří instance je vyřešeno pomocí jednoduchého switche. Místo používání textových konstant jsem si vytvořil

```
1 extension App {
2     var data: CustomApplicationStateData {
3         switch bundleIdentifier {
4         case AppSpecificBehaviour.Chrome.rawValue:
5             return ChromeStateData(self)
6         default:
7             return MacOSDefaultSaveStateData(self)
8         }
9     }
```

Výpis kódu 5.4: Implementace proměnné udržující stav aplikace

enum. Kód je tak přehlednější, a pokud by mělo někdy dojít ke změně identifikátoru aplikace, stačí upravit jeden řádek a změna se zpropaguje do celé aplikace.

To vše probíhá na pozadí bez povšimnutí uživatelem. Čeho si ale uživatel všimne je grafické rozhraní. Jeho vzhled jsem popisoval v předchozí kapitole. Uživatelské rozhraní je vytvořeno ve view-controller stylu. Tedy pro každou komponentu existují dva soubory. Jeden se samotným designem rozhraní – view a druhý, který se stará o uživatelskou interakci, například zpracuje kliknutí na tlačítko – controller. Design rozhraní se vytváří v programu s názvem Interface Builder. Tam se požadované komponenty pomocí drag'n'drop vloží do formulářového okna a rozmístí se dle potřeby.

Kapitola 6

Umístění na githubu

Aplikaci jsem veřejně publikoval jako repozitář na server GitHub[1]. Věřím, že uživatelé sami vědí, jaké další funkce by se jim v aplikaci hodily, a protože cílím na pokročilé uživatele macOSu, velká část z nich budou i programátoři. Ti tak mohou pomoci s dalším rozvojem aplikace a její údržbou do budoucna. Jelikož jsou dostupné všechny zdrojové kódy, může moje aplikace posloužit i dalším programátorům, kteří se zabývají obdobnými problémy na macOS, a pomoci jim s vývojem jejich vlastních programů. V repozitáři se také nachází jednoduchý tutoriál, který provede uživatele používáním aplikace a několik videí, které představí nejdůležitější vlastnosti aplikace ještě před tím, než si ji nainstaluje.

Obrázek 6.1: QR kód obsahující url gitového repozitáře aplikace

[1] https://github.com/mamiksik/Desktop-Profiles

Kapitola 7

Uvedení

I přesto, že aplikace byla, je a bude ve fázi vývoje, sdílel jsem své prototypy s dalšími uživateli macOS. Jejich reakce byly již na začátku překvapivě pozitivní.

“Desktop Profiles uhodila hřebíček přímo na hlavičku. Je to ten typ aplikace, který možná ani nevíte že chcete, ale jakmile si na ni zvyknete, je pro vás obtížné používat počítač, který ji nemá.” - Patrik Procházka, organizátor přednášek o podnikání.

Kapitola 8

Závěr

Už samotná příležitost vyzkoušet si vyvíjení pro operační systém macOS a proniknout hlouběji do jeho fungování pro mě byla přínosná. Nelze ale zapomenout, že se povedlo i naplnit cíle, jež jsem si na začátku nastavil zdárně implementovat. Vytvořil jsem tak aplikaci, která zjednodušuje používání macOS a nahrazuje rutinní úkoly, se kterými se setkává každý uživatel. Dosáhl jsem reálného dopadu na produktivitu uživatelů.

Oproti uzavřeným řešením ale nabízím možnosti rozšíření, protože i méně zkušený programátor je schopen si naprogramovat vlastní požadované funkce a obyčejný uživatel je schopen o ni programátory požádat skrze github. V budoucnu se proto chci zejména zaměřit na získání aktivní komunity v oblasti uplatnění aplikace a věnovat se jejímu dalšímu rozšíření.

Lze si také představit doplňkovou aplikaci například na chytré telefony, která by umožňovala přepínat profily pro počítače v okolí. Toto by bylo zejména užitečné například v počítačových učebnách, kde by učitel po příchodu do třídy mohl všem studentům zapnout potřebné nástroje pro danou výuku. Vzhledem k popularitě macOS především v Severní Americe, lze předpokládat, že by zájem o doplňkovou aplikaci byl právě zde.

Literatura

1. ADOBE. *Adobe Photoshop workspace basics* [online] [cit. 2019-01-05]. Dostupné z: https://helpx.adobe.com/photoshop/using/workspace-basics.html.
2. BAHOOM. *HyperDock* [online] [cit. 2019-02-09]. Dostupné z: https://bahoom.com/hyperdock.
3. APPTORIUM. *Workspaces* [online] [cit. 2019-01-05]. Dostupné z: https://itunes.apple.com/us/app/workspaces/id1219826448?mt=12.
4. APPTORIUM. *Workspaces* [online] [cit. 2019-01-05]. Dostupné z: https://www.apptorium.com/workspaces.
5. STATCOUNTER. *Desktop macOS Version Market Share Worldwide* [online] [cit. 2019-02-05]. Dostupné z: http://gs.statcounter.com/macos-version-market-share/desktop/worldwide/#yearly-2017-201.
6. APPLE. *Working with Spaces* [online] [cit. 2019-01-05]. Dostupné z: https://support.apple.com/cs-cz/guide/mac-help/work-in-multiple-spaces-mh14112/mac.
7. APPLE. *Use Mission Control on your Mac* [online] [cit. 2019-02-07]. Dostupné z: https://support.apple.com/en-us/HT204100.
8. APPLE. *Dock - System Capabilities - macOS - Human Interface Guidelines - Apple Developer* [online] [cit. 2019-01-05]. Dostupné z: https://developer.apple.com/design/human-interface-guidelines/macos/system-capabilities/dock/.

9. APPLE. *Configuring System Integrity Protection* [online] [cit. 2019-01-07]. Dostupné z: https://developer.apple.com/library/archive/documentation/Security/Conceptual/System_Integrity_Protection_Guide/ConfiguringSystemIntegrityProtection/ConfiguringSystemIntegrityProtection.html.

10. APPLE. *Getting Started with Graphics Animation* [online] [cit. 2019-01-15]. Dostupné z: https://developer.apple.com/library/archive/referencelibrary/GettingStarted/GS_GraphicsImaging/_index.html.

11. APPLE. *Xcode Help* [online] [cit. 2019-02-03]. Dostupné z: https://help.apple.com/xcode/mac/current/#/devc8c2a6be1.

12. APPLE. *Swift* [online] [cit. 2019-01-05]. Dostupné z: https://swift.org.

13. APPLE. *Localizing Your App!* [online] [cit. 2019-01-05]. Dostupné z: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPInternational/LocalizingYourApp/LocalizingYourApp.html.

14. COMPANY, The Eclectic Light. *Managing Mojave's privacy protection: Privacy controls* [online] [cit. 2019-01-05]. Dostupné z: https://eclecticlight.co/2018/09/17/managing-mojaves-privacy-protection-privacy-controls/.

15. ABHIMURALIDHARAN. *All about protocols in swift* [online] [cit. 2019-01-05]. Dostupné z: https://medium.com/@abhimuralidharan/all-about-protocols-in-swift-11a72d6ea354.

16. APPLE. *Extensions — The Swift Programming Language* [online] [cit. 2019-01-05]. Dostupné z: https://docs.swift.org/swift-book/LanguageGuide/Extensions.html.

17. REALM. *Realm: Create reactive mobile apps in a fraction of the time* [online] [cit. 2019-01-06]. Dostupné z: https://realm.io/docs/swift/latest/.

18. ABHIMURALIDHARAN. *Enums in swift* [online] [cit. 2019-01-09]. Dostupné z: https://medium.com/@abhimuralidharan/enums-in-swift-9d792b728835.

19. STORTI, Brian. *An introduction to UNIX processes* [online] [cit. 2019-01-05]. Dostupné z: https://www.brianstorti.com/an_introduction_to_unix_processes/.

20. APPLE. *The Core App Design* [online] [cit. 2019-02-11]. Dostupné z: https://developer.apple.com/library/archive/documentation/General/Conceptual/MOSXAppProgrammingGuide/CoreAppDesign/CoreAppDesign.html#//apple_ref/doc/uid/TP40010543-CH3-SW26.

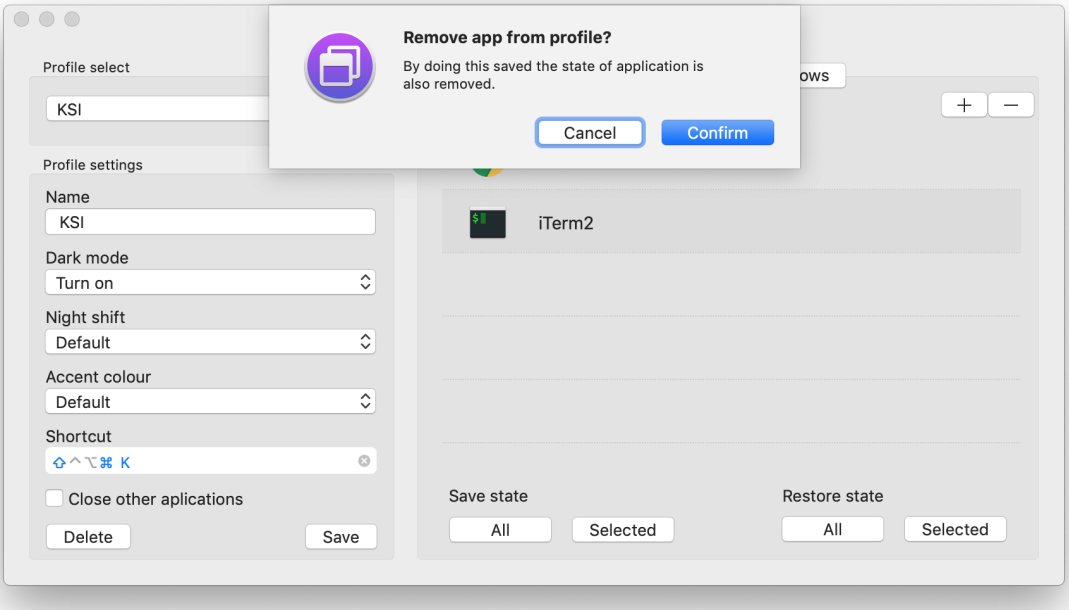
21. MONOMEETH. *Where are Google Chrome bookmarks stored in macOS for multiple profiles?* [online] [cit. 2019-01-05]. Dostupné z: https://apple.stackexchange.com/questions/322935/where-are-google-chrome-bookmarks-stored-in-macos-for-multiple-profile.

Seznam obrázků

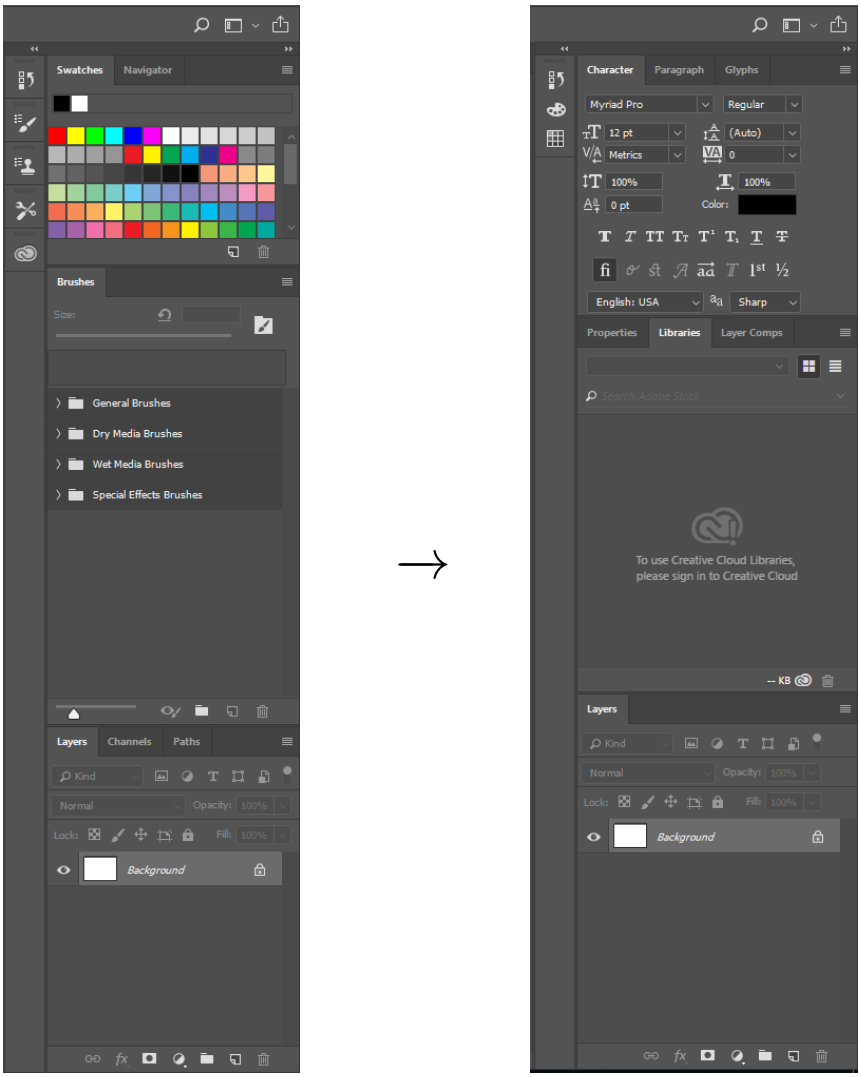
Seznam tabulek

Seznam zdrojových kódů

Přílohy

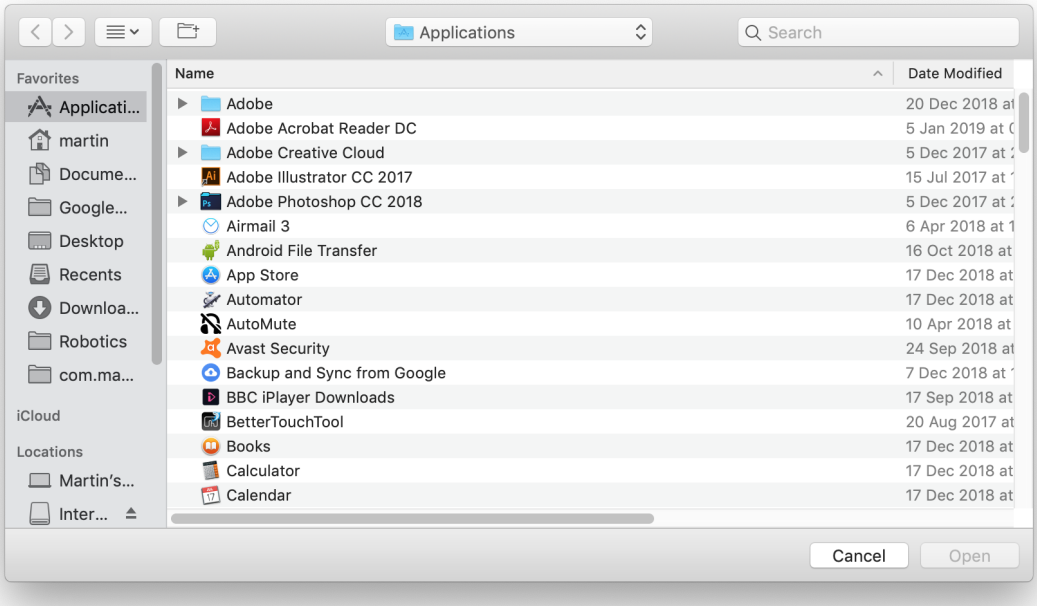


Obrázek 8.1: Potvrzení odebrání vybraných aplikací

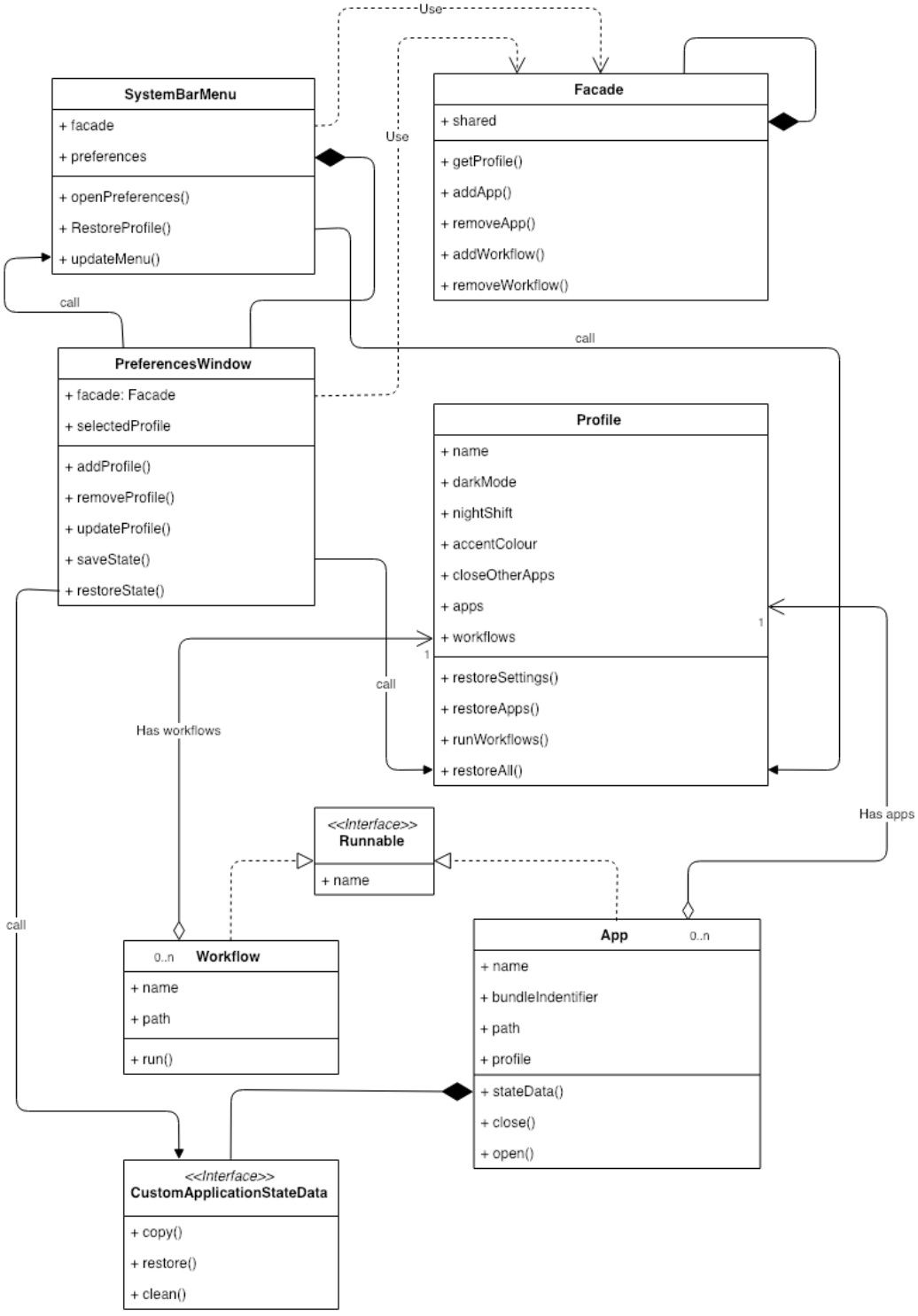


(a) Výchozí rozložení, A (b) Rozložení po přepnutí profilu, A

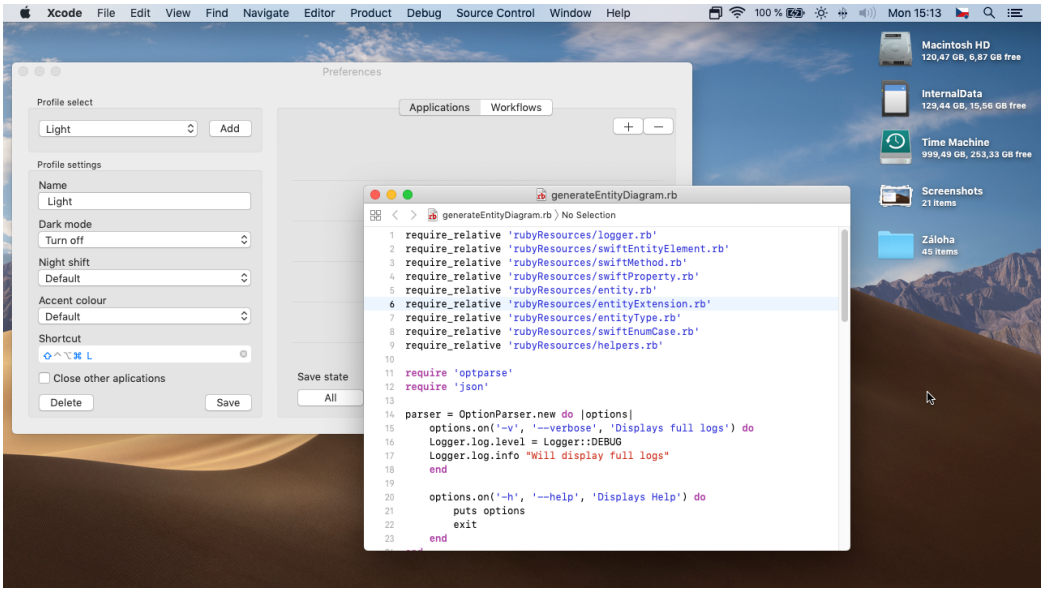
Obrázek 8.2: Implementace profilů pracovních ploch v Photoshopu



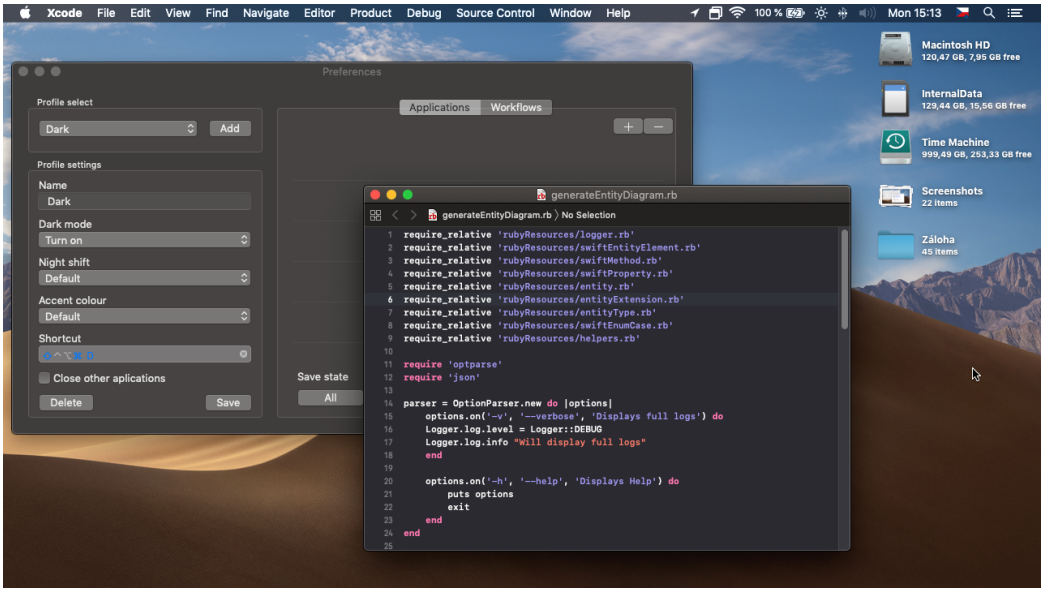
Obrázek 8.3: Dialogové okno pro výběr aplikace



Obrázek 8.4: Diagram zobrazující propojení komponent



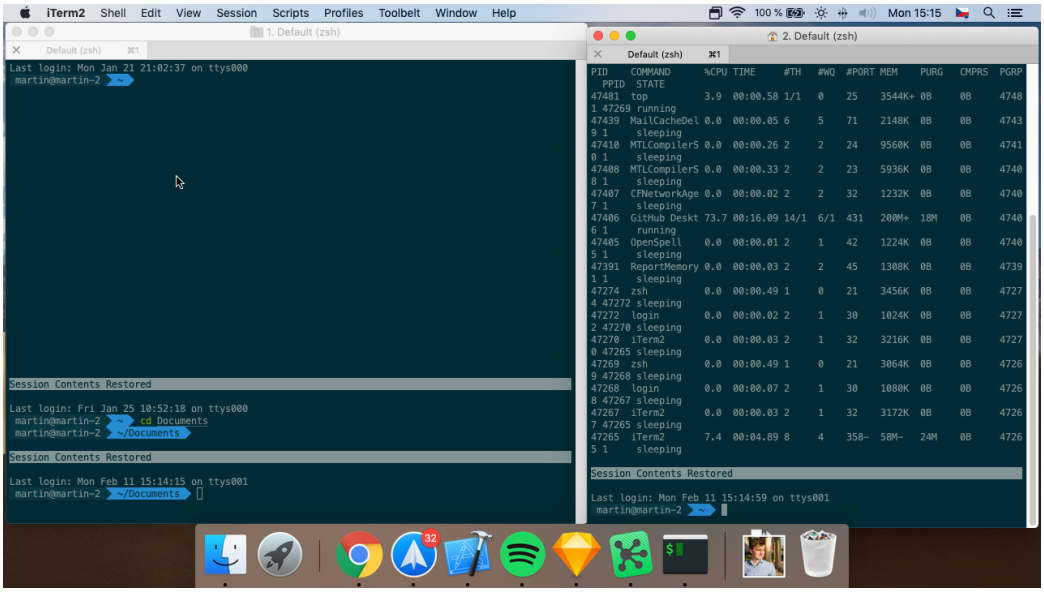
(a)



(b)

Obrázek 8.5: (a) Ve výchozím stavu má systém světlý vzhled. Na ploše se nachází otevřené okno editoru kódu a okno předvoleb profilů. (b) Byl obnoven profil Dark, což mělo za následek změnu pouze systémového nastavení vzhledu. Systém se tedy přepnul do tmavého režimu.

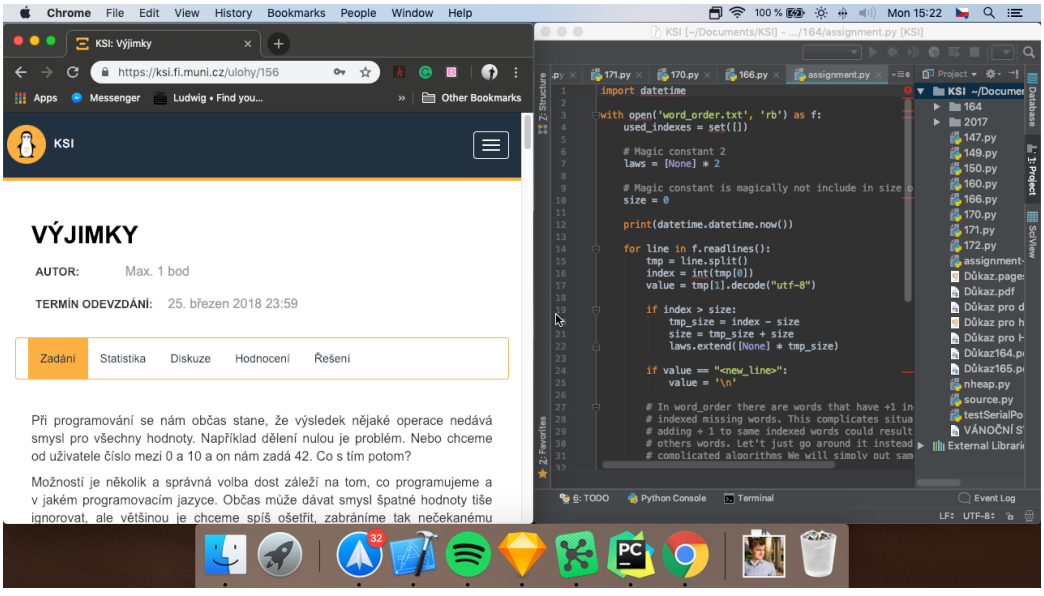
(a)



(b)

Obrázek 8.6: (a) Abych simuloval práci s terminálem, zavřel jsem jedno jeho okno, druhé jsem přemístil, změnil jeho velikost a za pomoci příkazu cd jsem přesunul shell do jiné složky. (b) Obnovení profilu Terminals vrátilo obě okna do původního stavu. Tedy dalo je vedle sebe a přesunulo je do správných složek.

(a)



(b)

Obrázek 8.7: (a) Výchozí stav simuluje například příchod domů. Nyní chci začít pracovat na korespondenčním semináři z informatiky (KSI). K tomu potřebuji otevřít vývojové prostředí a stránku se zadáním úlohy. (b) Profil KSI otevřel prohlížeč na správné stránce a jeho okno umístil vedle okna vývojového prostředí, ve kterém korektně otevřel projekt KSI.