ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA
STROJNÍ

DIPLOMOVÁ PRÁCE

Multiplatformní aplikace pro vizualizaci informací o dostupnosti vzdálených online služeb

VYPRACOVAL: Bc. Martin Cé

2023

VEDOUCÍ: Ing. Matouš Cejnek Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Cé** Jméno: **Martin** Osobní číslo: **475417**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídící techniky**

Studijní program: **Automatizační a přístrojová technika**

Specializace: **Automatizace a průmyslová informatika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Multiplatformní aplikace pro vizualizaci informací o dostupnosti vzdálených online služeb

Název diplomové práce anglicky:

Multiplatform application to visualize information about the availability of remote online services

Pokyny pro vypracování:

• Nastudujte možné způsoby tvorby desktopové aplikace, která komunikuje se vzdáleným serverem
• Vytvořte aplikaci, která dokáže ze vzdáleného API získávat a zpracovávat informace, které zobrazí uživateli pomocí vhodné reprezentace
• Připravte aplikaci včetně stručného uživatelského návodu tak, aby fungovala na všech důležitých platformách a byla jednoduše instalovatelná

Seznam doporučené literatury:

[1] I. Ristić, BULLETPROOF SSL AND TLS, Londýn: Feisty Duck, 2015
[2] W. O. Galitz, The Essential Guide to User Interface Design An Introduction to GUI Design Principles and Techniques, Indianapolis: Wiley Publishing, Inc., 2007

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Matouš Cejnek, Ph.D. U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **28.04.2023** Termín odevzdání diplomové práce: **08.06.2023**

Platnost zadání diplomové práce: _____

Ing. Matouš Cejnek, Ph.D.	prof. Ing. Tomáš Vyhlídal, Ph.D.	doc. Ing. Miroslav Španiel, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Prohlášení:

Prohlašuji, že jsem diplomovou práci na téma “Multiplatformní aplikace pro vizualizaci informací o dostupnosti vzdálených online služeb“ vypracoval samostatně pod vedením Ing. Matouše Cejnka, Ph.D. a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

V Praze dne: 4. června 2023 Podpis:

Poděkování:

Mé poděkování patří Ing. Matouši Cejnkovi Ph.D. za vedení a hodnotné odborné konzultace k mé práci. Děkuji také své rodině za trpělivost a všem kteří mi při mé prací byli nápomocni.

V Praze dne: 4. června 2023

Abstrakt

Cílem této práce je vytvoření aplikace, která umožní snadný a zabezpečený přístup k datům uloženým na serveru. Na serveru probíhá proces sledování dostupnosti testovaných vzdálených serverů v uživatelem daných intervalech. Umožňuje provádět různé úkoly spojené s administrací a správou těchto dat. Aplikace je vyvíjená s využitím multiplatformních technologií, které jí umožní její nasazení na různé operační systémy pro počítač. V aplikaci jsou funkce pro úpravu a zobrazení dat v grafech, jejich vyhledávání, filtrování a případné mazání pomocí přehledné tabulky i možnost zobrazení uživatelem zadaných souřadnic v mapě. Důraz je kladen na kvalitu a spolehlivost aplikace, také na snadné a intuitivní ovládání pro uživatele. Výsledkem je instalovatelná multiplatformní aplikace, která umožňuje uživatelům snadný přístup k datům na serveru a jejich úpravy.

Klíčová slova:
Multiplatformní aplikace; grafy; tabulka; mapa; přístup na server; Electron; frameworky; JavaScript; zabezpečené spojení

Abstract

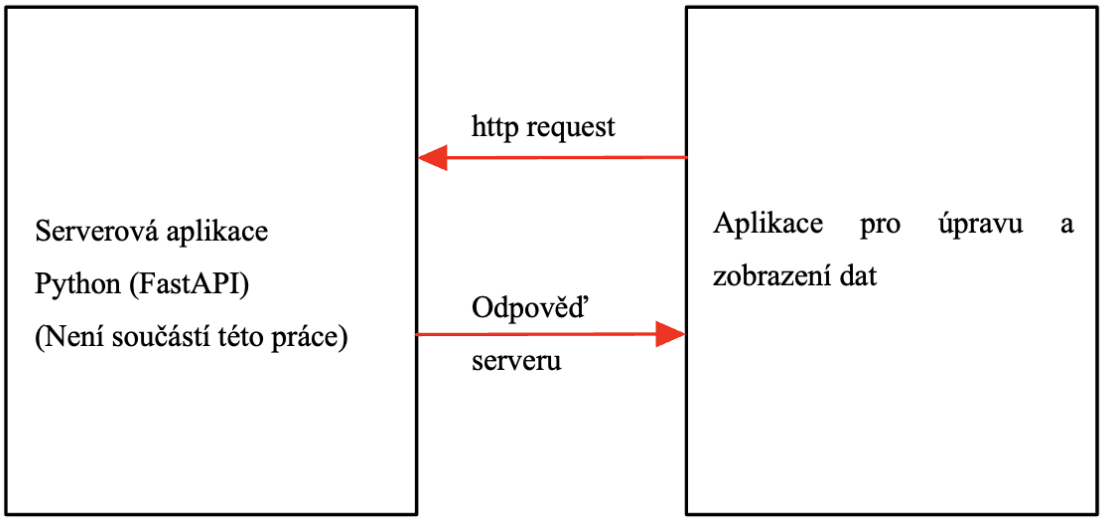
The aim of this work is to create an application that allows easy and secure access to data stored on a server. On a server there is a process of monitoring the availability of tested remote servers at user-given intervals. It allows to perform various tasks related to the administration and management of this data. The application is developed using multiplatform technologies that allow it to be deployed on various operating systems for computers. In the application there are functions for editing and displaying data in graphs, searching them, filtering them and possibly deleting them using a clear table as well as the possibility of displaying user-specified coordinates in a map. Emphasis is placed on the quality and reliability of the application, as well as on easy and intuitive operation for the user. The result is an installable multiplatform application that allows users easy access to data on the server and their editing.

Key words:
Multiplatform application; graphs; table; map; server access; Electron; frameworks; JavaScript; secured connection

Obsah

1. Úvod

Cílem této diplomové práce je vytvoření aplikace pro zobrazení dat uložených na serveru, která bude spustitelná na operačních systémech Windows, Linux a macOS s architekturou x64. Testovací serverová aplikace která není součástí této práce je tvořená paralelně. Je psána v programovacím jazyku Python a komunikuje pomocí FastAPI tzv. HyperText Transfer Protocol (HTTP) požadavků. Základní aplikace má určité úkoly. V tuto chvíli se jedná pouze o ping na uživatelem zvolené adresy v zadaných periodách. Mým cílem je najít vhodnou metodu pro ovládání a vykreslování dat, která testovací serverová aplikace generuje. Pomocí různých typů grafů vykresluje výsledné hodnoty naměřených dat a vykresluje polohy testovaných serverů do mapy pomocí uživatelem zadaných souřadnic. Dalším cílem je zajištění zabezpečené výměny dat mezi serverem a mou aplikací. Aplikace by měla poskytnout uživateli možnost jednoduchým způsobem vytvářet, upravovat, zastavovat nebo spouštět dané úkoly. Cílem práce je tudíž ošetření uživatelských vstupů pro korektní zadávání, zpracování dat do uživatelsky přívětivé podoby a zajištění aktualizace aplikace při vydání nové verze bez nutnosti přeinstalace celé aplikace. Závěrem bude aplikace připravena pro instalaci na Windows, Linux a macOS. Schéma serverové testovací aplikace a aplikace pro zobrazení a úpravu dat je na (obr. 1)

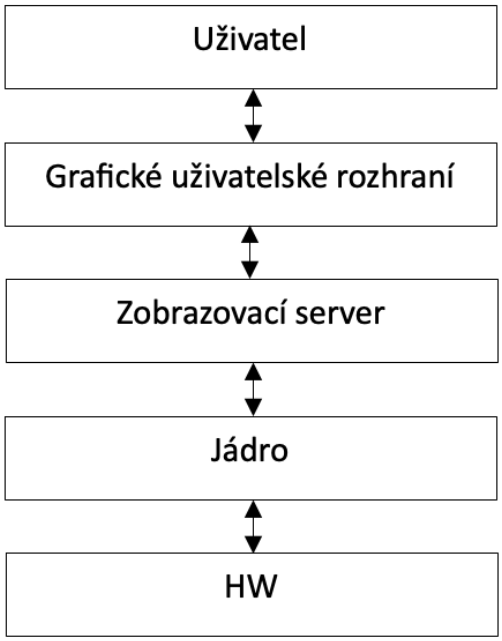


Obr. 1 Schéma struktury serverového pozadí a aplikace pro zobrazení a úpravu dat.

2. Teoretická část

2.1 Grafické rozhraní tvorba a základy

Podle [1] a [2] lze uvést následující. Grafické rozhraní (GUI) je prostředí, skrze které uživatel ovládá a dostává zpětnou vazbu z aplikace, která je spuštěná v počítači nebo na vzdáleném serveru. Schéma fungování aplikací od hardware (HW) po GUI na (obr. 2). Základem tohoto prostředí jsou tlačítka, vstupní textová pole, rozbalovací seznamy, vyskakovací okénka atp. Tyto prvky zajistí užívání a čitelnost odezev z aplikace pro uživatele. Důležitou částí je zabezpečení přenosu dat a výkon GUI. Zabezpečením se rozumí udělení přístupu uživateli k funkcím a k datům, která mu jsou určena. Validace vstupů zamezuje uživateli nesprávné ovládání nebo vkládání nevhodných dat a upozorňuje uživatele na případné chyby v zadaných parametrech, dále zajišťuje šifrování přihlašovacích údajů a citlivých dat, které uživatel zadává. Výkon GUI značí, jaké nároky klade samostatné uživatelské rozhraní na systém a HW. Tyto klíčové vlastnosti rozhodují, jak komfortním bude aplikace nástrojem pro koncového uživatele. Tvorba probíhá skrze nástroje GUI frameworky, ty se rozlišují funkcemi, designem a programovacím jazykem, ve kterém jsou psány.

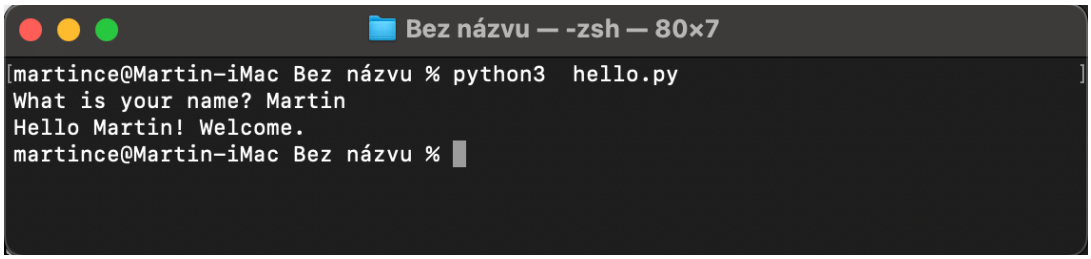


Obr. 2 Struktura programů od HW po grafické rozhraní promítané uživateli.

2.2 Prvky grafického rozhraní

Existují různé druhy GUI:

Textový terminál – komunikace čistě pomocí příkazového řádku, základní druh GUI. Komunikace mezi uživatelem a počítačem je pomocí jednoduchého okna, ve kterém je pouze textový vstup a výstup. Jak je uvedeno na příkladu (obr. 3), kde je jednoduchá aplikace, která vypíše dotaz pro zadání jména a po potvrzení dopíše „s pozdravem“ na následující řádek.

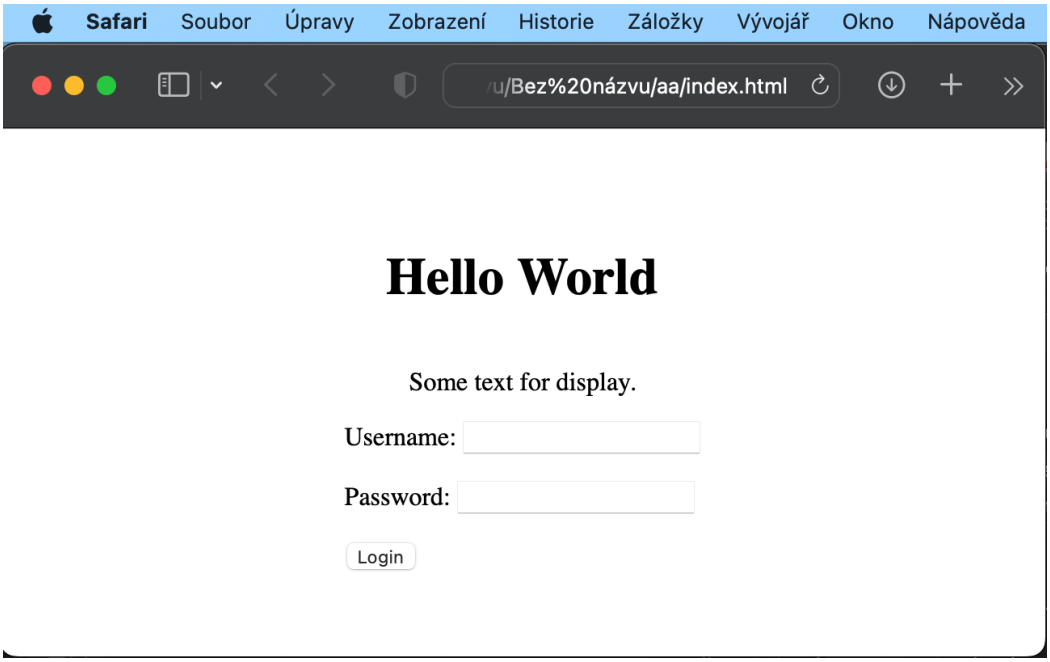


Obr. 3 Příklad aplikace v terminálu.

- Okenní aplikace – samostatná instalovaná aplikace přímo v počítači. Jedná se o interakci s uživatelem pomocí interaktivních prvků. Není třeba dalších aplikací pro její fungování. Příkladem je (obr. 4), kde je patrný horní panel nástrojů samotné aplikace nikoli prohlížeče. Není zde použito pro prohlížeče charakteristické vyhledávací pole pro Uniform Resource Locator (URL) adresy.

Obr. 4 Příklad okenní aplikace.

- Webová aplikace – pro zobrazení ovládacích prvků a obsahu využívá webového prohlížeče (Chrome, Safari, Edge atp.) a využívá technologie jako je HyperText Markup Language (HTML), Cascading Style Sheet (CSS) a Javascript. Příkladem je přihlašovací formulář v Safari (obr. 5). Porovnáme-li tuto aplikaci s výše zmíněným typem okenní aplikace dojdeme k závěru, že ve webovém prohlížeči není možnost nastavení ovládacího panelu a nachází se zde vyhledávací pole. Aplikace může být spuštěna lokálně na uživatelově počítači nebo na vzdáleném serveru.



Obr. 5 Příklad aplikace v internetovém prohlížeči.

2.2.1 Základní prvky grafického rozhraní okenní aplikace a webové aplikace.

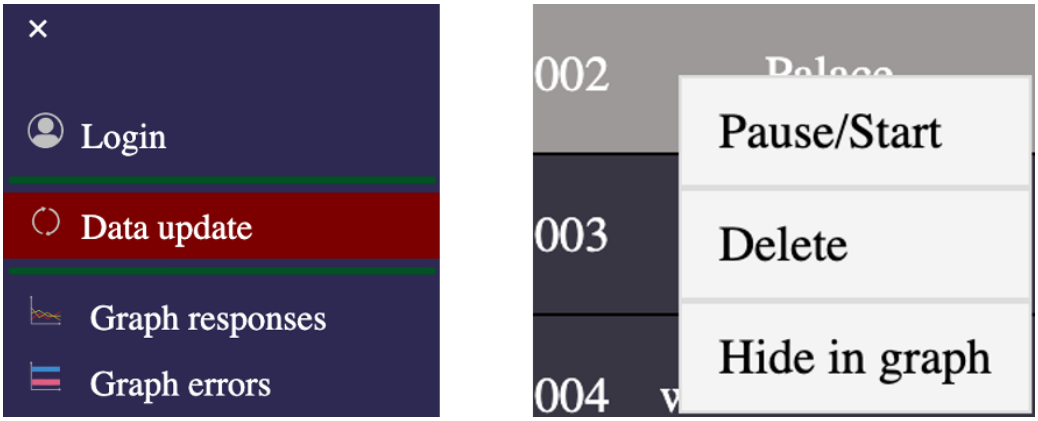
Prvky jsou následující podle [3].

- Okno

V základním oknu se nachází veškeré prvky, které budou popsány dále. To je základním kamenem grafického rozhraní a je s ním možné pohybovat po obrazovce případně jej minimalizovat na lištu počítače. Příkladem může být (obr. 5).

- Menu

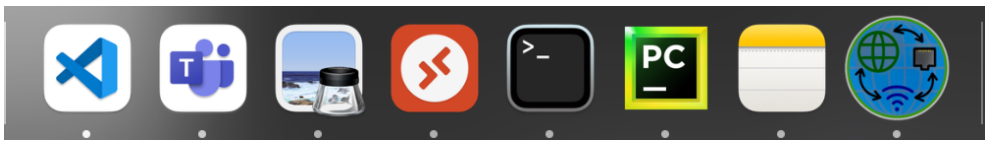
Navigační menu obsahuje seznam voleb nebo oken mezi kterými je možné vybírat. Dalším druhem menu je kontextové menu, které se zobrazí například po kliknutí pravým tlačítkem, kdy se rozbalí nabídka umístěná pod kurzorem a poté je možné vybrat jednotlivé položky. Jako je na (obr. 6).



Obr. 6 Navigační menu (vlevo) kontextové menu (vpravo).

- Ikony

Jednotlivé prvky aplikace jsou zastoupeny obrázky - ikonami, které spustí určitou akci po kliknutí. Ikony jsou například v (obr. 6) vlevo a představují je miniaturní piktogramy operací, které se na jednotlivých stránkách dále nacházejí. Nejznámějším prvkem je lišta s ikonami v operačním systému. Na (obr. 7) je znázorněna lišta s ikonami zastupujícími otevřené aplikace.



Obr. 7 Ikony zastupující aplikace macOS.

- Ovládací prvky

Jednotlivé prvky přímo ovlivňující chod aplikace. Jsou zde tlačítka, zaškrtávací políčka, posuvníky, rozbalovací menu atp. Tyto prvky jsou pro uživatele klíčové. Skrze ně může komunikovat s počítačem. Např rozbalovací menu zajistí korektní uživatelské vstupy a nabídne předem danou paletu všeho, co je možné pro vstup do aplikace zadat. Příklady ovládacích prvků na (obr. 8).

Obr. 8 Ovládací prvky.

2.3 Frameworky ke tvorbě multiplatformních aplikací

Podle práce [4] lze říci, že pojem multiplatformní aplikace znamená nezávislost na operačním systému (OS) nebo HW (telefon, tablet, PC a další), na kterém aplikace bude spouštěná. V této práci se budu zabývat hlavními OS pro PC, kterými jsou Windows, MacOS nebo Linux.

Framework je základním kamenem pro tvorbu aplikací, od desktopových po mobilní. Jedná se o soubor knihoven, pravidel a dalších nástrojů, díky čemuž může programátor rychleji a efektivněji vyvíjet aplikace. Jsou různé druhy frameworků, například pro tvorbu webových stránek, počítačových aplikací nebo her. Každý framework má své specifické vlastnosti, syntaxi a pravidla, které je nutné dodržovat pro optimální využití funkcí, jež jsou jeho součástí. Použití frameworku výrazně usnadňuje práci na aplikacích, přičemž programátor nemusí základní strukturu vytvářet od začátku a využije již hotové části kódu. Na druhou stranu se zde může vyskytnout problém při použití frameworků. Například menší kontrola programátora nad fungováním aplikace a jejích prvků.

2.3.1 Tři možné cesty pro vývoj multiplatformní aplikace.

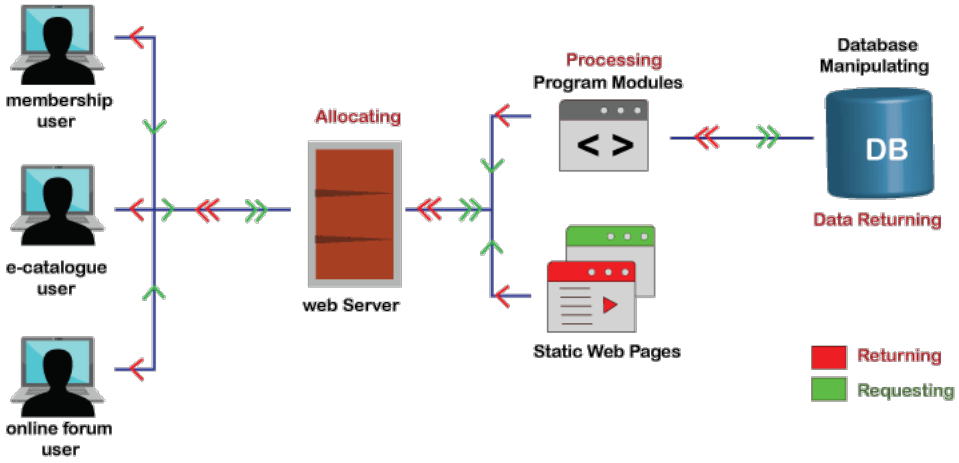
- **Nativní aplikace**

Podle [5] a [6] lze říci, že aplikace je vyvíjená přímo na daný operační systém s programovacím jazykem, který je pro tento systém specifický takže není nutné využít prostředníka v podobě internetového prohlížeče. Například pro platformu Windows jsou C++, C#, Visual Basic a další. U platformy Mac OS to jsou Swift, C a Objective C. U systému Linux to jsou C a Objective C, což znamená, že pro každou z výše zmíněných platforem je nutné napsat a udržovat program ve verzi pro daný operační systém. Díky plné kompatibilitě a optimalizaci aplikace pro jednotlivé OS může být výhodou naopak její rychlost a odezva. Pro vlastní běh nativní aplikace není nutností internetové připojení.

- **Webová aplikace**

Tento druh aplikace je závislý na internetovém připojení a její obsah je zobrazován v internetovém prohlížeči počítače, jak je patrné z [6]. Aplikace samotná bývá spuštěná na vzdáleném webovém serveru, díky čemuž je možné aplikaci spouštět bez instalace. Případné aktualizace se pak řeší pouze na samostatném serveru. Nevýhodou tohoto řešení

mohou být problémy v podpoře aplikace jednotlivými prohlížeči, jak je k na (obr. 9). To znamená, že v některých prohlížečích může nastat problém s vykreslením prostředí aplikace a s omezením jejích možností. Další z nevýhod je omezený přístup k HW a OS. Programování této aplikace je v programovacích jazycích pro webové servery (PHP, Java, Python a další) pro popředí na klientově počítači je využité HTML, JavaScript a CSS.



Obr. 9 Schéma Webové aplikace.

- **Cross-platform framework**

Z [6] vyplývá, že pro tento typ frameworku je programovacím jazykem HTML, CSS a JavaScript pro všechny platformy. Z toho dále plyne možnost využití webových technologií a knihoven, které jsou vytvořeny pro JavaScript. Oproti Webovým aplikacím je zde výhoda možnosti instalovat tento typ přímo na uživatelský počítač, kdy není nutné trvalé připojení k internetu při provozu. To poskytuje možnost vytvářet i off-line aplikace. Další výhodou je možnost využívat API operačních systému jako jsou dialogová okna a další.

2.4 Porovnání vybraných Cross-platform Frameworků pro tvorbu aplikace

Při vývoji své aplikace se budu dále věnovat Cross-platform frameworkům, které umožňují její vývoj pro všechny platformy.
Dále jsem vybral čtyři frameworky, z nichž budu volit framework, který pak využiji při tvorbě výsledné aplikace.

2.4.1 Kivy

Obr. 10 Logo kivy [7].

Z publikace [7] plyne, že se jedná se o open-source framework. Pro psaní aplikace lze využít Pycharmu nebo VS Code. Podmínkou je zde možnost psát kód v programovacím jazyku Python. Kivy je založen na knihovně SDL. Výhodou je možnost psaní kódu aplikace a grafického rozhraní v Pythonu. Kivy podporuje Windows, macOS a Linux, takže jím tvořené aplikace lze distribuovat i na Android a iOS. Kivy se využívá zejména pro vývoj mobilních her a aplikací, ale také poskytuje možnost tvorby desktopových aplikací.

Struktura kódů, nastavení projektu, architektura a programovací jazyk

Pro začátek je nutné stáhnout a instalovat kompilovanou verzi Kivy, která je dostupná na stránkách společnosti. Aplikaci je nutné instalovat s možností spuštění v terminálu. Pro zobrazení jednoduchého okna s nápisem „Hello world“ je nutné nadefinovat třídu, ve které jsou jednotlivá nastavení prvků, které se mají vykreslovat. Příklad (Kód 1) představuje pouze label s textem „Hello World“. Příslušná třída se následně vyvolá v hlavním programu, který je pod podmínkou IF.

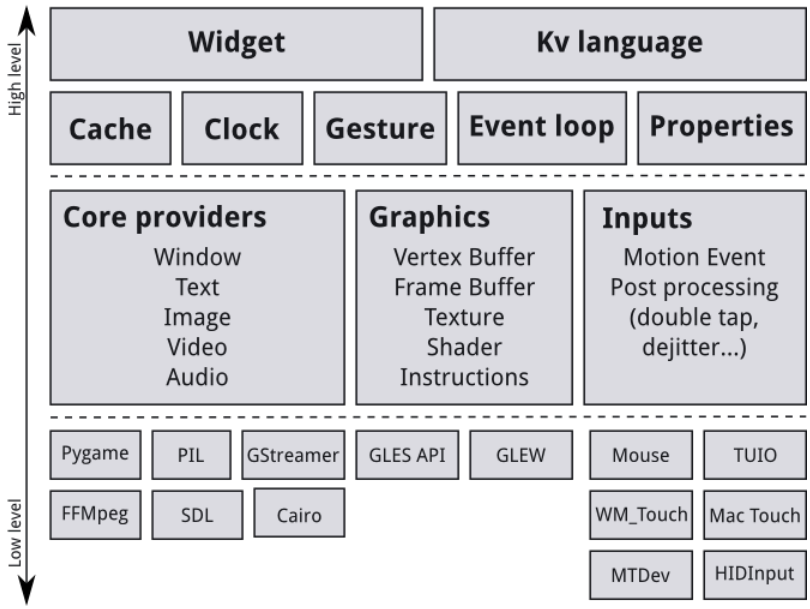
```
from kivy.app import App
from kivy.uix.label import Label

class HelloWorldApp(App):
    def build(self):
        label = Label(text='Hello world!')
        return label

if __name__ == '__main__':
    HelloWorldApp().run()
```

Kód 1 Příklad okenní aplikace v Kivy [7]

Architektura Kivi dle dokumentace [8] je rozdělená na tři vrstvy:

- Model: data používaná v aplikaci
- Controller: řídí interakci mezi Modelem a View
- View: uživatelské rozhraní aplikace

Hlavním prvkem je okno, představující view vrstvu aplikace a zajišťuje zobrazení dat. Ovládáním vstupů a výstupů se uživatel skrze vrstvu controller dostává do samotného jádra aplikace, do vrstvy Model. Na (obr. 11) je architektura Kivy, kdy je rozdělená do úrovní (levelů). V nejvyšší úrovni je zmíněné View, ve střední je Controller a v nejnižší najdeme Model.



Obr. 11 Architektura Kivy [8]

2.4.2 JavaFX

Obr. 12 JavaFX logo [10].

Dle článku [10] se jedná o open-source framework vyjma pár funkcí, které jsou pod komerční licencí. Programovacím jazykem je Java. Původně byl z dílny Sun Microsystems, která je v tuto chvíli pod společností Oracle. První uvedení frameworku proběhlo v roce 2008. Základem jsou technologie Java SE a Java ME. Vývojářům nabízí širokou paletu nástrojů pro tvorbu interaktivních aplikací s grafikou, animacemi a efekty. Knihovna obsahuje předpřipravené prvky pro tvorbu GUI, kterými jsou tlačítka, textová pole, seznamy, tabulky, grafy a další. Tyto prvky umožňují tvorbu jakýchkoli aplikaci včetně mobilních, desktopových a webových aplikací. Dalším prvkem je možnost využití jazyka FXML, který umožňuje oddělení uživatelského rozhraní od kódu samotné aplikace, čímž je zjednodušen vývoj a údržba aplikace. JavaFX je podporována na všech OS jako Windows, macOS a Linux.

Struktura kódů, nastavení projektu, architektura a programovací jazyk

Programování pomocí javaFX je popsané v [11]. Pro programování je vhodné nainstalovat vývojové prostředí Eclipse, NetBrains nebo IntelliJ IDEA. Poté je nutná instalace JavaFX. Od verze Javy číslo 11 je JavaFX již součástí balíčku. Poté je možné začít s programováním. Na (Kód 2) je napsaná jednoduchá okenní aplikace s výpisem „Hello“.

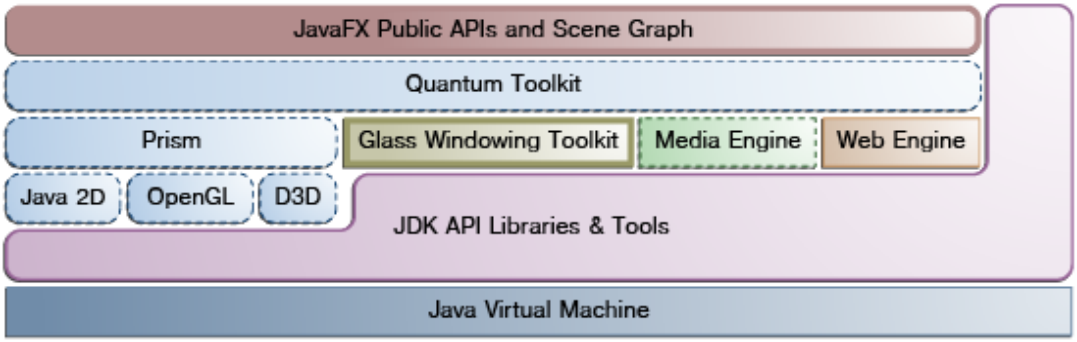
```
<? xml version="1.0" encoding="UTF-8"?>
<? import javafx.scene.control.Label?>
<? import javafx.scene.image.Image?>
<? import javafx.scene.image.ImageView?>
<? import javafx.scene.layout.AnchorPane?>
<? import javafx.scene.layout.Pane?>
<    AnchorPane    accessibleText="″    xmlns:fx="http://javafx.com/fxml/1"
xmlns="http://javafx.com/javafx/17">
   < children>
      < prefHeight="400.0" prefWidth="400.0">
         < children>
            < Label text="Hello"/>
         </children>.
      </Pane>.
   </children>.
</AnchorPane>.
```

Kód 2 Příklad okenní aplikace v JavaFX [12]

Architektura dle [10] je podobná výše zmíněnému Kivy a návrhový vzor je:

- Model: aplikace a využívaná data
- Controller: interakce mezi Modelem a View
- View: grafické rozhranní

Model reprezentuje datovou část. Obsahuje třídy, popisující objekty a jejich logiku při práci s nimi. Tyto třídy jsou nezávislé na View a Controller. View komponenta obsahuje prvky grafického rozhraní a promítá data z Modelu. Controller zajišťuje spojení mezi View a Model. Obsahuje též obecné prvky pro aplikaci, příkladem je inicializace nebo práce s komponentami systému. Architektura je k vidění na (obr. 13).



Obr. 13 Architektura JavaFX [10]

2.4.3 Qt

Obr. 14 Logo Qt [13].

V knize [14] a dokumentaci [13] je uvedeno, že se jedná o multiplatformní framework pro vývoj grafických prostředí psaný v jazyce C++ podporuje také QML a Python. Framework byl vyvinut společností Qt Company, která dříve spadala pod firmu Nokia. Poskytuje paletu nástrojů pro tvorbu aplikací s dynamickým grafickým prostředím. Jeho obsahem jsou různé widgety pro vytvoření ovládacích prvků, správy sítě a multimédií. Jeho vlastností je též možnost tvořit desktopové aplikace pro různé platformy jako jsou macOS, Windows a Linux. Taktéž lze tvořit mobilní aplikace pro iOS a Android. Je k dispozici ve dvou verzích open-source a komerční verze. Komerční verze přidává další možnosti a rozšíření funkcí.

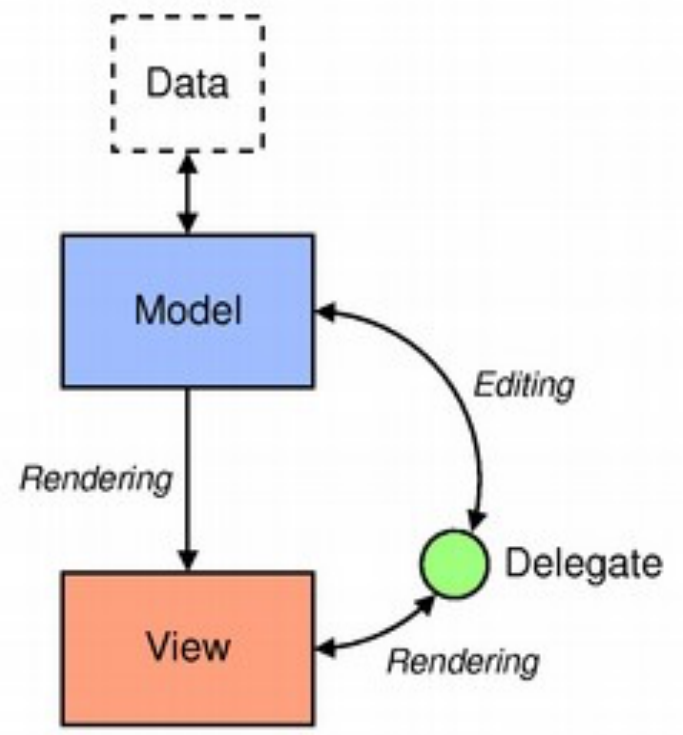
Struktura kódů, nastavení projektu, architektura a programovací jazyk

Pro tvorbu aplikací v Qt je v prvním kroku podle [13] potřeba nainstalovat poslední verzi tohoto frameworku. Tato aplikace obsahuje vlastní editor pro tvoření a úpravu kódů. V (Kód 3) představuje jednoduchou okenní aplikaci s výpisem slova „Hello“.

```
importing QtQuick 2.12
import QtQuick.Window 2.0
Window {
  id: root
  width: 600
  height: 600
  title: "Priklad"
  Text {
    text: "Hello world!"
  }
}
```

Kód 3 Příklad okenní aplikace ve Qt [12]

Z [13] vyplývá, že architektura Qt je model-view-programing architekturou, která je pro tvorbu aplikací běžně využívána. Data jsou zde oddělena podle toho, jak jsou zobrazována uživateli. Jednotlivá data jsou ukládána v modelech. Tyto modely zajišťují jejich správu a aktualizace. Pohledy (view) naopak zobrazují data uživateli a zpracovávají interakci s uživatelem. Tyto vstupy jsou dále zpracovány pomocí složky Delegate. Ta zajišťuje vykreslení jednotlivých položek. Rendering je proces díky, kterému jsou data z modelů převáděná do grafické podoby. Editing zajišťuje úpravu dat, kdy je poslán povel k editaci z Delegate do modelu, který data upraví.



Obr. 15 Architektura Qt [13].

2.4.4 Electron

Obr. 16 Logo Electron [15]

Z dokumentace [15] a [12] vyplívá, že framework Electron je open-source, který je využíván k vývoji desktopových aplikací. Vývojářům umožňuje psát kód běžnými způsoby pro webové technologie jako jsou HTML, CSS a JavaScript. Tento framework byl vyvinut společností GitHub v roce 2013. Jeho původním účelem bylo použití pro aplikace Atom, což je textový editor a pro Slack chatovací platformu. Výhodou tohoto frameworku je možnost využití rozsáhlé knihovny doplňků. Často je využíván pro tvorbu multiplatformních aplikací ve větších korporacích.

Příklady aplikací vytvořených v tomto frameworku:
Dropbox – aplikace sloužící pro správu dat uložených v cloudovém úložišti.
VS Code – aplikace pro vývoj a úpravu kódů.
Discord – chatovací platforma.
GitHub Desktop – aplikace pro správu projektů uložených na platformě GitHub.

Struktura kódů, nastavení projektu, architektura a programovací jazyk

K vytvoření aplikace lze využít editoru VS Code nebo Atom. Po spuštění inicializace si Electron vyžádá potřebná data, z nichž se vytvoří soubor s konfiguračními vlastnostmi. Vygenerovaný soubor může vypadat například jako (Kód 4). Zde jsou nastavené hlavní vlastnosti tvořené aplikace.

```
{
    "name": "myAPP",
    "version": "1.0.0",
    "main": "main.js",
    "author": "Me",
    "license": "MIT"
}
```

Kód 4 Konfigurační soubor pro electron

Další část tvoří *main.js* (Kód 5). Tento kód slouží k načtení okna a prelaod procesu. Main proces má přímý přístup k funkcím OS.

```
const { application, BrowserWindow } = require('electron')
const path = require('path')
function createWindow () {
  const window = new BrowserWindow({
      width: 500,
      height: 500,
      webPreferences: {
    nodeIntegration: false,
    contextIsolation : true,
    preload: path.join(__dirname, 'preload.js')
    }
  })
  window.loadFile('index.html') }
  application.whenReady().then(() => { createWindow()
  application.on('activate', () => {
  if (BrowserWindow.getAllWindows().length == 0) {
        createWindow();
      }
  })
})
```

Kód 5 Příklad kódu pro main.js

Main proces komunikuje skrze Inter Process Communication (IPC) s popředím – rendererem. IPC je zaváděné v programu nazvaném *preload.js*. Zde se nastaví druh a vlastnosti komunikace mezi *main.js* a rendery, které mají přímý přístup do HTML. V (Kód 6) je příklad dokumentu *preload.js* pro komunikaci rendereru s *main.js*. a je zde vždy nastavení *send* pro posílání dat z renderu do mainu, *recieve* pro komunikaci opačným směrem. Pro vyšší zabezpečení komunikace jsou nastavené parametry validních kanálů, přes které aplikace komunikují.

```
const    {    contextBridge,    ipcRenderer    }    =    require("electron");
contextBridge.exposeInMainWorld("api", {
    send: (chan, da) =>
      if ("toMainLoad" == chan) {
      ipcRenderer.send(chan, da);
            console.log("%c  -" + chan +  " send", 'color:green')
       }
       },
    receive: (chan, da) =>
      if ("fromMainLoad" == chan) {
            console.log("%c  -" + chan + " recieve", 'color:green')
            ipcRenderer.once(chan, (event, ...args) => da(...args));
       }
    }
});
```

Kód 6 Příklad kódu pro preload.js

V (Kód 7) je HTML stránka, do které se promítají data ze scriptu, který je (Kód 8). V tomto případě se načítá pouze slovo „Hello“. V tomto kódu je nastavená komunikace, kdy se zavolá IPC pro načtení textu z *main.js* a pak následuje přijetí zprávy a její výpis do konzole.

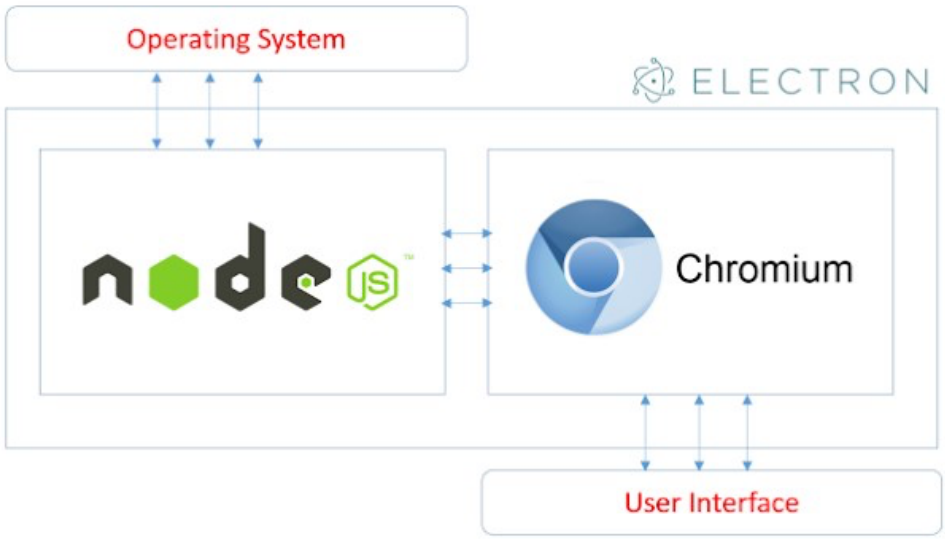
```
<!DOCTYPE html>
<html>
  <head>
    <title>Example</title>
    <link rel="stylesheet" href="style/style.css" />
  </head>
  <body>
      <p id="text"></p>
      <script src="./podpr/renderer.js"></script>
  </body>
</html>
```

Kód 7 Příklad kódu pro index.html pro okno v electronu.

```
document.getElementById("text").value = "Hello"

window.api.send("toMainLoad")
window.api.receive("fromMainLoad", (text) => {    console.log(text)
})
```

Kód 8 Příklad renderer.js volaný z html stránky.

Architektura Electronu dle [9] je následující: Základem je Node.js což je JavaScriptová knihovna. Tato knihovna obsahuje funkce pro tvorbu webových stránek. Díky tomu není nutné spuštění serveru a lze aplikaci tvořit jako webovou stránku. Knihovna zajišťuje komunikaci mezi aplikací a OS počítače. Další vrstvou je Chromium, kdy se jedná o nástroj pro zobrazení výsledné webové stránky. Stránky jsou vykreslovány pomocí WebKitu což je open-source vykreslovací nástroj pro webové prohlížeče a desktopové aplikace. Struktura Electronu je na (obr. 17).



Obr. 17 Architektura electron [9]

2.5 Závěrečné srovnání a výběr frameworku

Tabulka srovnání frameworků při porovnávání podle stupnice od 1 do 5 je vždy 5 nejlepší a 1 nejhorší.

1 Srovnávací tabulka frameworků.

Framework	**Kivy**	**JavaFX**	**Qt**	**Electron**
Jazyk	Python	Java	C++, JavaScript	HTML, CSS a JavaScript
Náročnost pro naučení	4/5	4/5	2/5	4/5
Komunita	3/5	3/5	4/5	5/5
Windows	Ano	Ano	Ano	Ano
Linux	Ano	Ano	Ano	Ano
macOS	Ano	Ano	Ano	Ano
Dokumentace	3/5	4/5	3/5	4/5
Open-source	Ano	Ano	Ano	Ano
Typ licence	MIT	GPL	GPL	MIT
Typ aplikace	Nativní	Nativní	Nativní	Nativní

Vyhodnocení srovnání frameworků

Při další práci budu srovnávací údaje z tabulky (tab. 1) zohledňovat. Beru v potaz náročnost vlastního učení a intuitivnost kódu, to bude určovat, jak náročná pak bude tvorba výsledné aplikace. Podstatná je také velikost komunity přispívajících programátorů, která určuje rozsah knihoven pro framework. Čím větší je komunita, tím více uživatelských knihoven je k dispozici. Při vyhodnocení jsem přihlédl i k podporovaným operačním systémům a rozsahu dokumentace, open-source licenci a jejím podmínkám, druhu licence a typu, a také jak bude výsledná aplikace fungovat.

Z tabulky plyne, že veškeré porovnávané frameworky jsou open-source. Podle [16] jsou vlastnosti licencí porovnávány a nabízejí se dva druhy licence Massachusetts Institute

of Technology (MIT) a General Public License (GPL). Vytvořené aplikace s licencí GPL jsou volné s určitými omezeními a podmínkami. To znamená, že i výsledná aplikace vytvořená pod GPL musí mít shodnou licenci. Naproti tomu licence MIT, dává vývojáři volnou ruku. Autoři mohou psaný kód pod touto licencí dále distribuovat pod vlastní licencí nebo jej i prodávat. Z tohoto důvodu vychází lépe frameworky Kivy a Electron. Dále je vhodné zmínit, že veškeré posuzované frameworky umožňují distribuci aplikací na Linux, Windows nebo macOS i distribuci na mobilní platformy jako jsou iOS nebo Android. Kivy, Qt a JavaFX umožňují tuto distribuci výchoze bez dalších nástrojů. V případě použití Electronu pro Android nebo iOS je nutná instalace dalších nástrojů. Rozsah a srozumitelnost dokumentace je další prvek, který je nutný posuzovat při výběru. Veškeré uvedené frameworky umožňují nativní instalaci vytvořené aplikace, to znamená možnost extrahování do .exe, .dmg nebo .snap. Tyto soubory jsou pak připravené na distribuci a instalaci.

Ze srovnání v tabulce jsem se rozhodl využít frameworku Electron a to nejen díky jeho rozsáhlé základně uživatelů, z níž plyne velké množství knihoven a dobrá dokumentace. Dalším faktorem pro mé rozhodnutí je licence MIT, která dává vyšší volnost oproti licenci GPL. Neposledním rozhodujícím faktorem je možnost psát kód pomocí HTML, JavaScript a CSS.

2.6 SSL/TLS zabezpečená komunikace

V této části se věnuji zabezpečení komunikace mezi mou aplikací a severem a popisu průběhu navázání komunikace z pohledu aplikace.

Secure Socket Layer/Transport Layer Security (SSL/TLS)

Podle [17] byl SSL protokol vyvinut společností Netscape. První veřejně vydaná verze tohoto certifikátu byla verze 2 z roku 1994 a poslední 3. verze SSL je z roku 1995. Tuto verzi nahradil v roce 1996 TLS od společnosti Internet Engineering Task Force. V tuto chvíli nejaktuálnější verzí TLS je 1.3, která byla vydána v roce 2018.

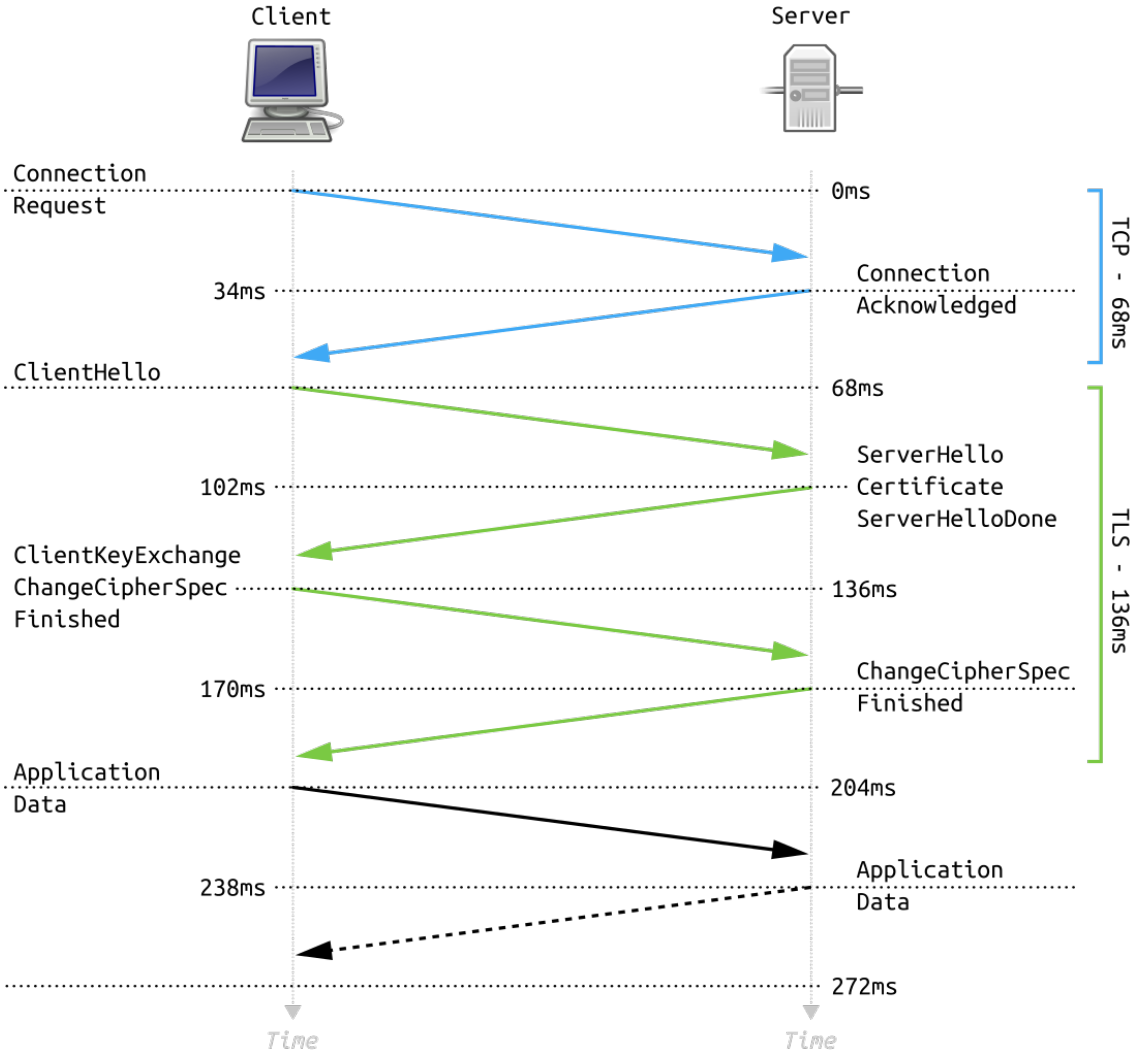
Zabezpečená komunikace mezi serverem a aplikací probíhá pomocí SSL/TLS šifrování s asymetrickou šifrou. SSL/TLS protokol je způsob, jak zabezpečit komunikaci mezi aplikací, která komunikuje se serverem nebo webovým serverem. Bezpečná komunikace mezi aplikací a serverem je zásadní pro ochranu citlivých dat, jako jsou například přihlašovací údaje, bankovní informace nebo osobní údaje. Aplikace, která komunikuje se serverem, může být například webový prohlížeč, mobilní aplikace nebo jiný typ software (SW). Tato aplikace využívá protokolu HTTPS (HTTP + SSL/TLS) pro zajištění bezpečného přenosu dat mezi klientem a serverem. Rozdíl mezi TLS a SSL vyplývá z úvodní části. Jak již bylo uvedeno, TLS je novějším protokolem oproti SSL, tudíž TLS je v této době preferovanějším protokolem pro zabezpečení.

Komunikace z pohledu klienta

Dle [17] komunikace začíná tzv handshakem. Klient posílá požadavek na server, server odpovídá s digitálním certifikátem. Tento certifikát obsahuje informaci o identitě serveru a jeho veřejný klíč. Ten se využívá pro šifrování posílaných dat směrem k serveru. Veřejný klíč klient použije k vytvoření nového klíče pro šifrování a dešifrování zpráv, které se posílají.

Při komunikaci s klientem se na straně serveru vytvoří klíč pro šifrování a dešifrování zpráv, které jsou posílány klientovi. Tento klíč je jedinečný a pro každou komunikaci se vytváří nový a nazývá se session key. Klíč zajišťuje bezpečnost komunikace proti

odposlechu komunikace. Odchycená data by bez použití session key byla zbytečná. Zároveň se při komunikaci mezi klientem a serverem ověřuje platnost digitálního certifikátu, který vydává ověřená autorita. Tento certifikát je vložen v aplikaci nebo webovém prohlížeči a na serveru samotném. Z toho obě strany poznají jsou-li si navzájem důvěryhodné. Příklad handshake navázání komunikace je na (obr. 18).



Obr. 18 Handshake a autentizace komunikace TLS [18].

Popis handshake a autentizace ze strany serveru z (obr. 18).

1. Klient započne komunikaci a pošle své parametry na server.
2. Server nastaví spojové parametry.
3. Server odešle certifikát (pouze tehdy kdy je požadována autentizace)
4. V případě provedení kroku 3 server odešle informace ke generování seshion key pro komunikaci.
5. Server odesílá potvrzení o dokončení jeho části prováděných úloh k nastavování komunikace.
6. Klient posílá informace ke generování seshion key ke komunikaci.
7. Klient přepíná do šifrované komunikace a posílá o tom informaci serveru.
8. Klient odešle MAC z handshaku a komunikuje již šifrovaně.
9. Server přepíná na šifrovanou komunikaci a odešle informaci klientovi.
10. Server odesílá MAC handshaku a komunikuje již šifrovaně.

Podstatnou částí při nastavování SSL/TLS je brát v úvahu výkon a využití zdrojů počítače pro jejich provoz. Vysoká úroveň zabezpečení sebou přináší vyšší nároky na výkon a výpočetní kapacitu klientů.

Celkem lze tedy říci, že pro zajištění bezpečné komunikace mezi serverem a aplikací je využití protokolu SSL/TLS, který zajistí zabezpečený přenos dat procesu a komunikace. Aplikaci je vhodné navrhnout způsobem, aby umožnila použití SSL/TLS protokolu a musí být vybavena mechanismy pro ověření platnosti certifikátů, které přijímá od serveru.

2.7 Sestavení a distribuce aplikace pro základní tři OS

V knize [19] je uvedeno, že pro vytvoření instalačního souboru na jednotlivé platformy je potřeba aplikaci a její soubory zabalit do instalačního balíčku. Formát balíčků se pro každou platformu liší. V příponě souboru například pro Windows je rozšířená koncovka .exe pro macOS .dmg a pro Linux .snap. Pro každou platformu je nutné dodržovat určitá pravidla pro zajištění správné kompatibility. Další podmínkou při distribuci je nastavení architektury cílového systému. Po sestavení aplikace je také nutné zajistit její distribuci koncovým uživatelům, a to například pomocí obchodů pro dané platformy nebo ke stažení z webu. Příkladem obchodů jsou (obr. 19) Microsoft Store (vlevo) pro Windows, App Store (uprostřed) pro macOS nebo Snap Store (vpravo) pro Linux. Další variantou stažení z webu je využití GitHubu ve složce Releases. Na tuto složku je následně možné odkazovat z webové stránky produktu.

K získání instalačních souborů v Electronu je k dispozici několik knihoven, které zajistí vytvoření instalačních balíčků pro každý systém. Z těchto knihoven jsou výše zmíněné instalační soubory.

Obr. 19 Obchody k distribuci aplikací.

Postup pro stavbu aplikace probíhá následujícím způsobem:

- Nastaví se cílový operační systém, pro který se bude aplikace stavět
- Upraví se soubor *package.json* příkladem je (Kód 9)
- Nastaví se soubory, co se budou ve výsledné aplikaci vyskytovat. Těmi se myslí jednotlivá okna, *main.js*, rendery, HTML s CSS a případně další vytvořené soubory s obrázky či jiným obsahem.
- Nastaví se ikona aplikace, která existuje pro každý ze systémů pod různými koncovkami, pro Windows má koncovku .ico pro macOS je to .icns a pro Linux je to .png
- Po tomto kroku je možné již nechat sestavit instalační soubor aplikace. Tento soubor se bude nacházet ve složce dist.

```
"files": [
    "src/**/*"
  ],
"build": {
    "appId": "com.MCNetworkMonitor.app",
    "productName": "Network Monitor app",
    "directories": {
      "output": "dist"
    },
    "win": {
      "icon": "ic/win.ico"
    },
    "mac": {
      "icon": "ic/mac.icns"
    },
    "linux": {
      "icon": "ic/linux.png"
    }
  },
```

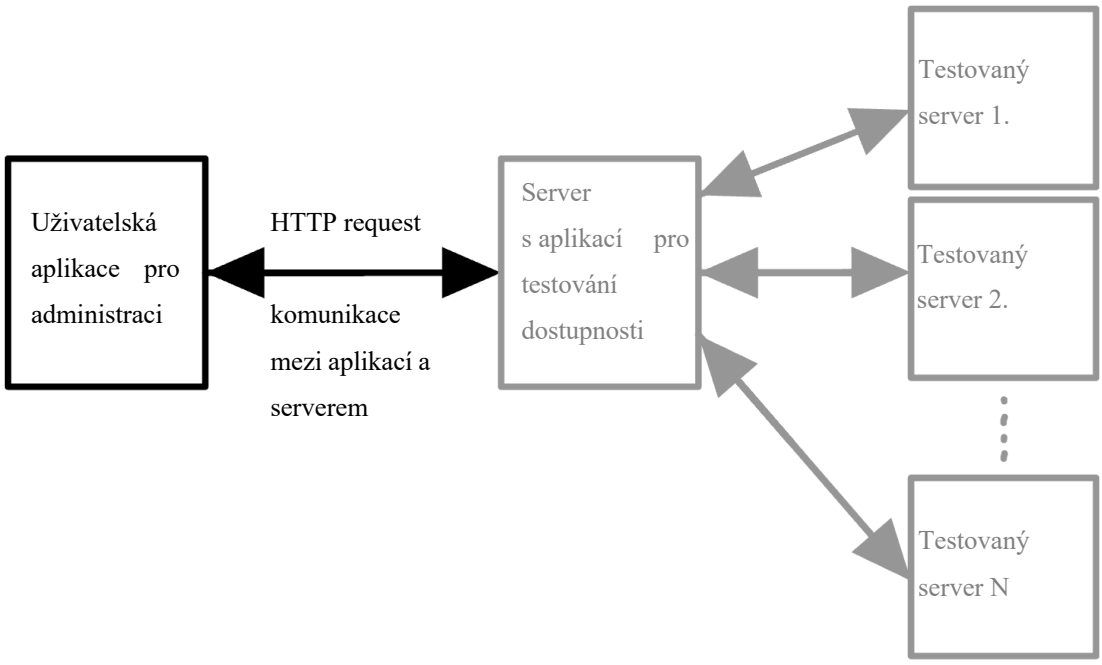
Kód 9 Část z preload.js s nastavením pro distribuci aplikace.

3. Praktická část

V teoretické části jsem zvolil pro stavbu aplikace framework Electron. V této části se budu užívat tohoto frameworku a knihoven, které s ním souvisí. Jak bylo výše zmíněno, Electron je postaven na webových technologiích. Z toho tedy plyne psaní samotného kódu pomocí HTML, ve kterém se nastavuje pozice a velikost prvků. JavaScript zajišťuje dynamické funkce těchto prvků, např. zajišťuje vkládání a mazání textu. Poslední je CSS, které souvisí s HTML a jsou v něm zanesené styly jednotlivých prvků, které jsou zobrazovány.

3.1 Funkční požadavky a návrh aplikace.

Níže je uvedeno schéma aplikace a hostitelského serveru (obr. 20). Zde se nachází (černou barvou) uživatelská aplikace pro administraci a HTTP request pro komunikaci se serverem, na kterém je spuštěna testovací aplikace. Šedou barvou je označena testovací serverová část, která již není předmětem mojí práce. Uvádím ji však pro lepší názornost.



Obr. 20 Schéma aplikace včetně serverové části.

Uživatel aplikace si může jejím prostřednictvím ověřovat dostupnost serveru, na kterém provozuje vlastní i veřejně dostupné služby, například webové stránky, internetové bankovnictví aj. Dalším prvkem aplikace je pozorování odezvy serveru a jeho odpovědi

v čase pro posouzení dostupnosti uvedených služeb. Aplikace dále umožňuje testování více serverů najednou, ale i zařízení připojených na lokální síti. V průběhu vytváření aplikace jsem testoval několik vybraných velmi rozdílných a globálně využívaných serverů pro znázornění funkcí aplikace.

Náplní této aplikace je stahování, zobrazování i úprava dat a prováděných úloh na serveru s testovací aplikací. Serverová aplikace s mojí aplikací bude komunikovat pomocí HTTP requestů s TLS šifrováním. V prvním kroku se uživatel do aplikace přihlásí předem nastaveným uživatelským jménem a heslem. Tím se zpřístupní provádění úkonů na serveru. Při přihlášení se teprve zpřístupní možnost proklikávat mezi jednotlivými okny aplikace a provádět úpravy nebo sledovat stavy jednotlivých úkolů. Jednotlivá podokna, která aplikace bude obsahovat, jsou v (tab. 2).

2 Tabulka úkonů, které aplikace bude obsahovat.

Č.	**Okno**	**Popis**
1	Přihlášení	Okno přihlášení bude složené z přihlašovacího formuláře, ve kterém bude uživatelské jméno, heslo a volitelné pole pro adresu serveru v případě, že není spuštěn na uživatelově počítači.
2	Administrace	Tabulka, v níž bude výpis všech serverů, pro které je testována dostupnost. Objeví se zde formulář, který bude sloužit pro úpravu nebo vytvoření nového testování serveru.
3	Graf odezvy	Spojnicový graf zobrazující aktuální hodnotu odezvy s možností načtení předchozích stavů.
4	Graf úspěšnosti	Horizontální sloupcový graf znázorňující počty úspěšné/neúspěšné odezvy.
5	Graf celkového počtu měření	Koláčový graf znázorňující poměr počtu všech měření pro jednotlivé servery.
6	Graf průměrné odezvy	Prstencový graf znázorňující průměrnou hodnotu odezvy všech serverů.
7	Mapa polohy testovaných serverů	Mapa s body znázorňujícími polohu testovaných serverů.
8	Manuál	Manuál k aplikaci.

3.2 Nastavení projektu Electron, Node.js a struktura souborů

Pro programování využiji aplikace VS Code. Nainstaluji aplikaci Node.js pro získání možnosti využití příkazu „npm“. Následně nainstaluji Electron a inicializuji projekt pomocí příkazů (Kód 10) v kořenovém adresáři. Verzování kódů budu provádět pomocí GitHubu.

instalace electronu

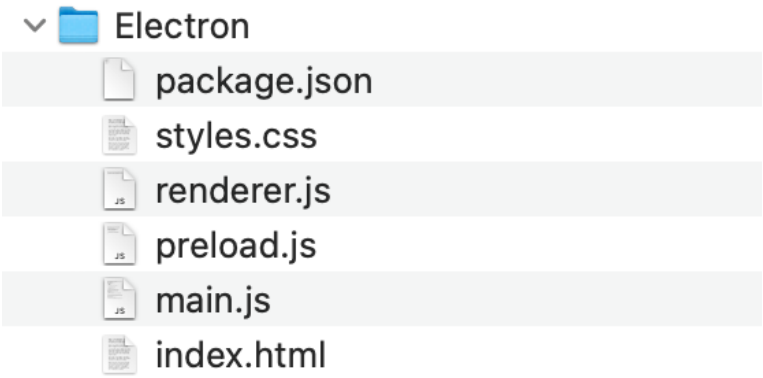
```
sudo npm install -g electron --unsafe-perm=true --allow-root
```

inicializace projektu

```
npm init
```

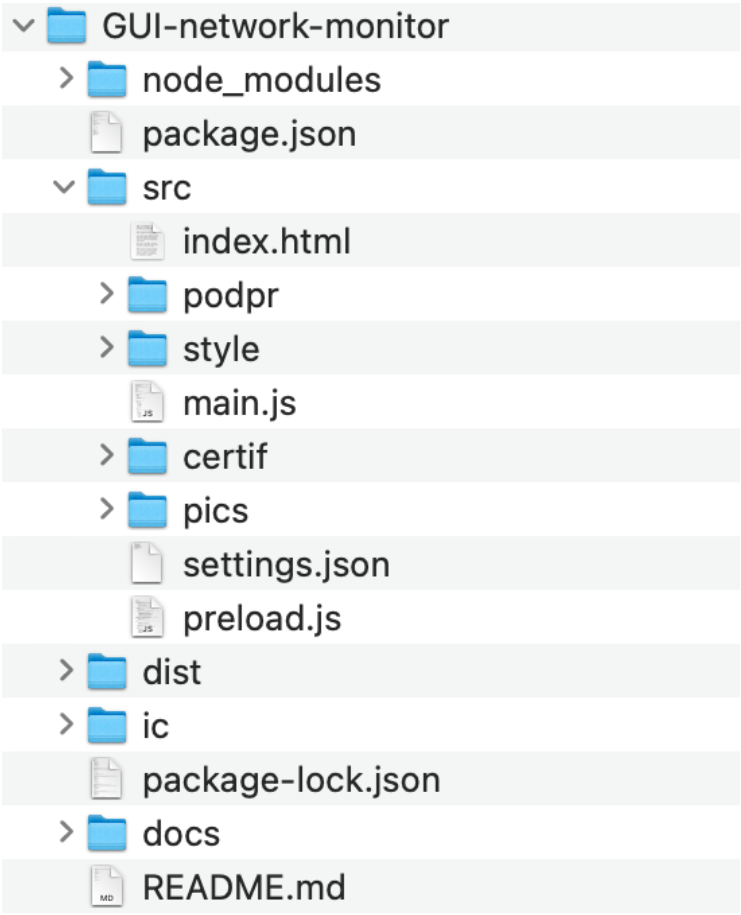
Kód 10 Příkazy k instalaci Electronu a nastavení projektu.

Po provedení instalace a inicializace projektu se mi vytvoří soubor s názvem *package.json*, v němž se nachází hlavní inicializační vlastnosti, využité knihovny a popis aplikace. Dále vytvářím soubor *main.js*, který má přístup k systémovým souborům a akcím. Tento soubor je jádrem celé aplikace a veškerá komunikace s počítačem a serverem probíhá skrze něj. Následuje soubor *preload.js* – tento soubor zajišťuje komunikaci popředí aplikace takzvaného *rendereru.js* s hlavním souborem *main.js*. Dalším souborem je *index.html* a *style.css* sloužící k vykreslení okna. Posledním souborem je *renderer.js*, který slouží k dynamickým změnám HTML stránky a s pozadím komunikuje přes zmíněný *preload.js*. Struktura souborů po inicializaci a vytvoření hlavních souborů je na (obr. 21).



Obr. 21 Soubory základní aplikace.

Tyto soubory jsou pro další tvorbu klíčovými prvky, ale roztřídím je pro lepší orientaci, jak je znázorněno na (obr. 22). Soubory a adresáře, které budou generované automaticky jsou například zaváděcí soubory, adresáře knihoven a aplikace k distribuci zůstanou v kořenové složce. Soubory, které budu vytvářet či jinak upravovat budou ve složce *src*. Na (obr. 22) je struktura souborů, které jsou zálohované na GitHubu k verzování kódu. Složka *node_modules* obsahuje instalované knihovny, složka *dist* obsahuje aplikaci k distribuci, složka *ic* obsahuje ikony aplikace, která slouží pouze k distribuci, složka *docs* a soubor *README.ms* slouží k prezentaci na GitHubu. Pro samotnou aplikaci a její tvorbu nemají význam.



Obr. 22 Struktura souborů výsledné aplikace.

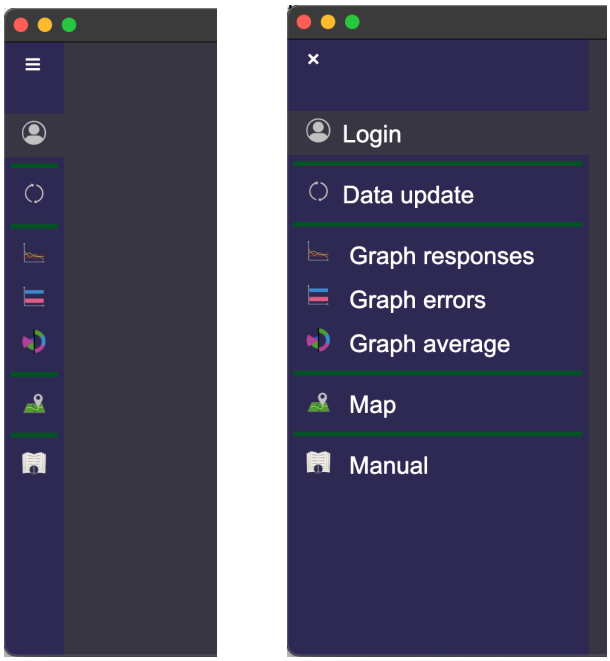
V (Kód 11) je nastavení hlavního okna a vzhledu navigačního panelu v *main.js*. V této části jsem nastavoval výšku okna při otevření aplikace a minimální velikost, kterou může okno aplikace dosahovat. Dále se při změně velikosti okna udržuje poměr stran 16:9. V další části jsem nastavil položky kontextového menu pro načítání a ukládání uživatelova seznamu serverů pro případ přenosu na jiný server.

```
// nastavení vlastností okna a načtení preload.js
mainWindow = new BrowserWindow({
    width: 1280,
    height: 720,
    resizable: true,
    minWidth: 960,
    minHeight: 540,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')}
  })

  // nastavení názvů a fungování položek v menu
  const menu = [
      {
        label: 'File',
        submenu: [
             { label: 'Load file', click: () => handleFileOpen() },
             { label: 'Save', click: () =>
                   mainWindow.webContents.send("fromMainhowSave") },
        ]
      },
  ]
))
//nastavení položek do menu
Menu.setApplicationMenu(Menu.buildFromTemplate(menu))

app.whenReady().then(() => {
//nastavení výšky a šířky okna na poměr stran 16:9 při změně velikosti okna
  mainWindow.on('resize', () => {
    const [width, height] = mainWindow.getSize();
    if (height !== Math.round(width * 9 / 16)) {
      mainWindow.setSize(width, Math.round(width * 9 / 16));
    }
  });
}
```

Kód 11 Nastavení okna aplikace v main.js.

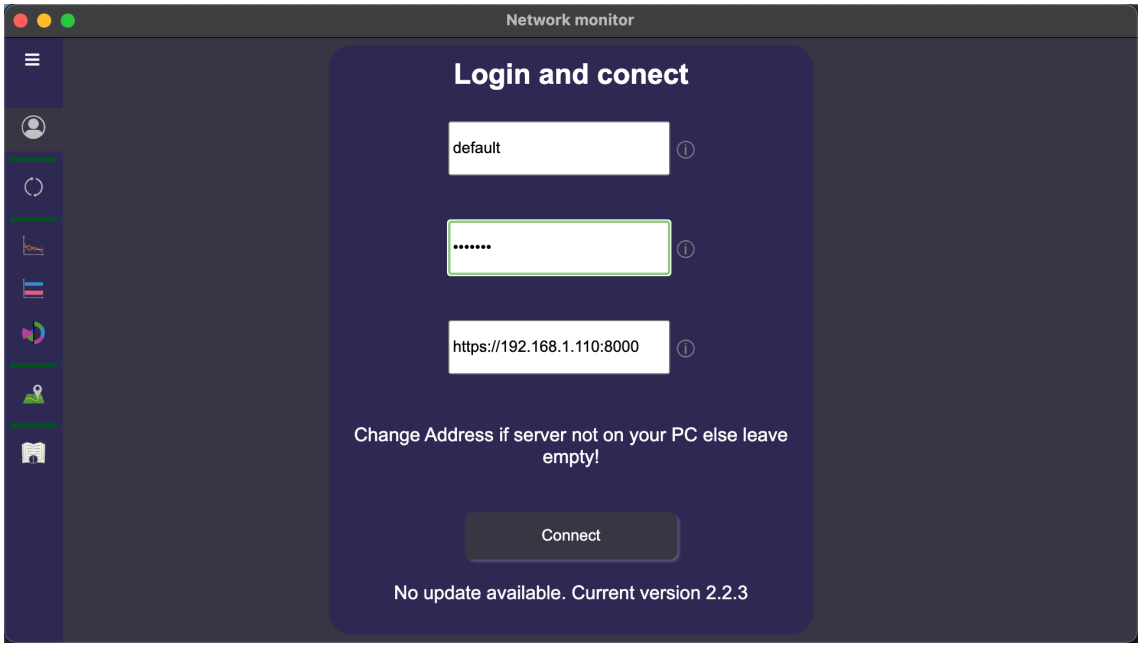
3.3 Návrh vzhledu přihlašovacího menu a rozložení prvků

Pro rozložení prvků v hlavním okně jsem se rozhodl následovně podle [20]. Po levé straně aplikace bude navigační panel, na kterém bude možné přepínat mezi jednotlivými okny, ve výchozím stavu bude obsahovat pouze ikony zastupující jednotlivá okna a po kliknutí na rozbalovací tlačítko bude uživateli zobrazen i popis jednotlivých podoken pro lepší orientaci v aplikaci. Zavření menu probíhá pomocí tlačítka křížek nebo po kliknutí na některou z voleb. Ikony jsou vytvořené pomocí aplikace Sketches. Na (obr. 23) je navigační panel složený (vlevo) rozložený (vpravo).

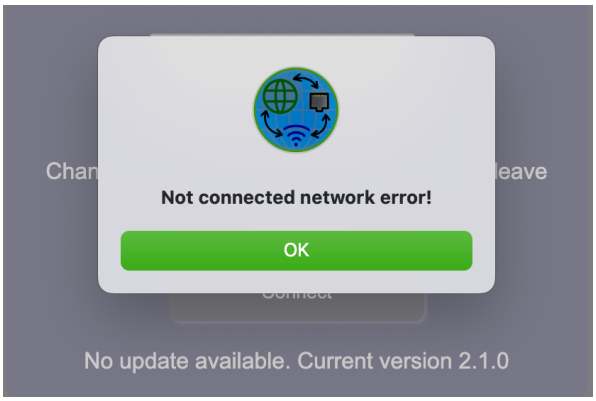


Obr. 23 Navigační panel skrytý(vlevo) rozbalený (vpravo).

Přihlašovací formulář je umístěn ve středu okna s třemi vstupními poli pro Jméno, Heslo a Adresu serveru, na kterém běží aplikace. Vedle polí se nachází ⓘ s informací, ve kterém tvaru má být pole vyplněno. Pak následuje tlačítko pro připojení a přihlášení. Pod tímto tlačítkem se nachází dynamický text, který zobrazuje, jakou verzi aplikace uživatel využívá jako na (obr. 24). V případě, kdy bude aplikace neaktuální, budou zadány špatné přihlašovací údaje nebo nebude aktivní připojení k internetu zobrazí se vyskakovací okno s upozorněním na problém (obr. 25). Níže je uveden příklad kdy aplikace nemá připojení k serveru.



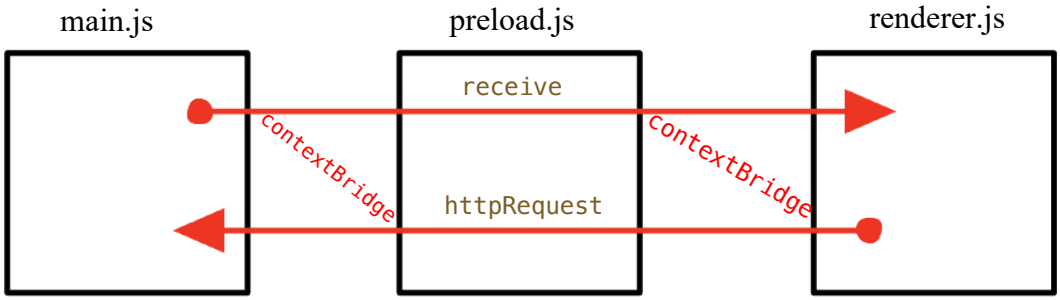
Obr. 24 Okno aplikace s přihlašovacím formulářem.



Obr. 25 Vyskakovací okno s upozorněním při problému s připojením.

3.4 Propojení main.js s popředím (renderer.js)

Na tomto příkladu uvádím nastavení komunikace mezi procesem *main.js* a *renderer.js* pro přístup k HTTP requestům nebo načítání dat ze souboru podle publikace [21]. V následujícím (obr. 26) je zobrazena komunikace, při které se nastavuje takzvaný kontextový most, přes něhož probíhá následná výměna dat mezi popředím (*rendererem.js*) a pozadím (*main.js*). Tento most je tvořen pro vyšší zabezpečení uživatele a s jeho pomocí se provádí veškerá komunikace. Rendererům se tímto způsobem zamezuje přímý přístup k funkcím OS. V tomto příkladu *render.js* zažádá o data. Tato žádost prochází skrze *preload.js* do *main.js*, kde se provede stažení dat, a ta se následně posílají zpět přes *preload.js* do *rendereru.js*, který je vypíše do HTML stránky.



Obr. 26 Komunikace hlavního main.js a rendererů.

V (Kód 12) je znázorněn kód, který se nachází v souboru *prelaod.js* v němž nastavuji způsob výměny dat a povolené kanály pro komunikaci. Je zde funkce, která obsahuje „recieve“ poslání dat z *main.js* a „httpRequest“ pro žádost o data ze strany *render.js*.

```
// nastavení funkcí pro komunikaci mezi procesy
contextBridge.exposeInMainWorld("api", {
// recieve slouží k odeslání dat z main do rendereru
    receive: (channel, func) => {
        let validChannels = ["fromMainRequestLoadAll"];
        if (validChannels.includes(channel)) {
            ipcRenderer.once(channel, (event, ...args) => func(...args));
        }
    },
// httpRequest slouží naopak k odelílání dat z rendereru do main
    httpRequest: async(channel, reqValues)=>{
        let validChannels = ["requestLoadAll"];
        if (validChannels.includes(channel)) {
            await ipcRenderer.invoke(channel,reqValues)
        }
    },
});
```

Kód 12 Úryvek nastavení meziprocesové komunikace v preload.js.

Při zmáčknutí tlačítka uživatelem se aktivuje požadavek pro získání dat z *main.js* v (Kód 13). Po odeslání požadavku o data renderer čeká na odpověď s daty, o která bylo zažádáno. Celý proces je zakončen v tomto případě pouze výpisem do konzole. Tato data mohou být například data z HTTP requestu nebo ze souboru atp.

```
window.api.httpRequest("requestLoadAll")
window.api.receive("fromMainRequestLoadAll", (allRespInterv) => {
      console.log(allRespInterv)
})
```

Kód 13 Odeslání požadavku, následný příjem a zpracován dat v renderer.js.

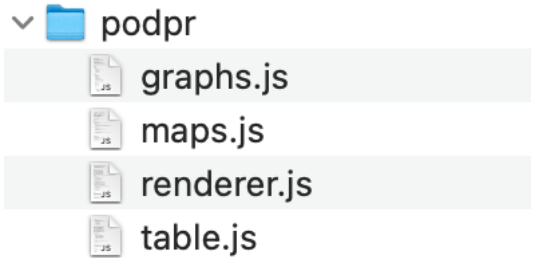
V (Kód 14) je následně napsán kód, který v případě zažádání o data ze strany *rendereru.js* odesílá zpět odpověď.

```
ipcMain.handle("requestLoadAll", async (event)=>{
      mainWindow.webContents.send("fromMainRequestLoadAll", response.data)
  })
```

Kód 14 Příjem požadavku z rendereru a odeslání požadovaných dat v main.js.

Renderery aplikace jsem rozdělil a vložil do *src/podpr/* (obr. 27):

- *graphs.js* – jedná se o podprogram, který spravuje obsluhu a vykreslování grafů
- *maps.js* – tento podprogram se stará o vykreslování a úpravy mapy
- *renderer.js* – zajišťuje hlavní menu, přihlašovací okno a přepínání mezi okny aplikace
- *table.js* – stará se o úpravu tabulky testů a interakce od uživatele s touto tabulkou



Obr. 27 Renderery pro jednotlivé části aplikace.

3.5 HTTP requesty pomocí zabezpečené komunikace

V této části nastavuji zabezpečenou komunikaci se serverem ze zdroje [22] a [23].
K zabezpečené komunikaci využívám knihovny *https*. Tato knihovna umožňuje sestavovat hlavičku requestů pro zabezpečenou komunikaci. Knihovna je již obsažena v Node.js tudíž instalace není nutná. V (Kód 16) je v první části zajištěno načtení požadovaných souborů, kterými jsou certifikát a klíč. Tyto soubory se využívají pro navázání zabezpečené komunikace, jak bylo zmíněno v kapitole 2.6 SSL/TLS zabezpečená komunikace. Klíč a certifikát jsou generovány bez certifikační autority pouze lokálně. Jedná se o takzvaný samo podepsaný certifikát.
Soubory se vloží pomocí knihovny do proměnné agent, která zastřešuje zmíněné zabezpečení. Vstupem je klíč a certifikát. Závěrem je nutné nastavit, aby aplikace nekontrolovala certifikát serveru testovací aplikace u certifikační autority, který je taktéž vygenerován bez certifikační autority. K zabezpečené komunikaci vyhovují samo podepsaný certifikát a klíč a splňují požadavky pro zabezpečení ve fázi vývoje i při provozu. Zde důvěřuji autorům serverové části s testovací aplikaci. V budoucnu bude možné certifikát generovat přímo od vydavatele certifikátů, a tím by poslední část kódu bylo možné vynechat, nicméně tato varianta je zpoplatněna.
K navázání komunikace jsem zvolil využití knihovny *Axios*, která umožňuje posílání a následný příjem pomocí HTTP requestů. V první řadě je nutné knihovnu do projektu instalovat pomocí příkazu (Kód 15), který se zadá do příkazového řádku v kořenovém adresáři. Po instalaci v (Kód 16) navážu na předchozí část vytvořením inicializační části *Axios*, do níž zahrnu IP adresu serveru a agenta popsaného výše. Poté je již umožněná výměna dat mezi mou aplikací a serverovou částí. Zde je uveden příkaz POST, který provádí přihlášení pomocí jména a hesla na server.

instalace axios knihovny

```
npm install axios
```

Kód 15 Příkazy k instalaci https a axios knihovny.

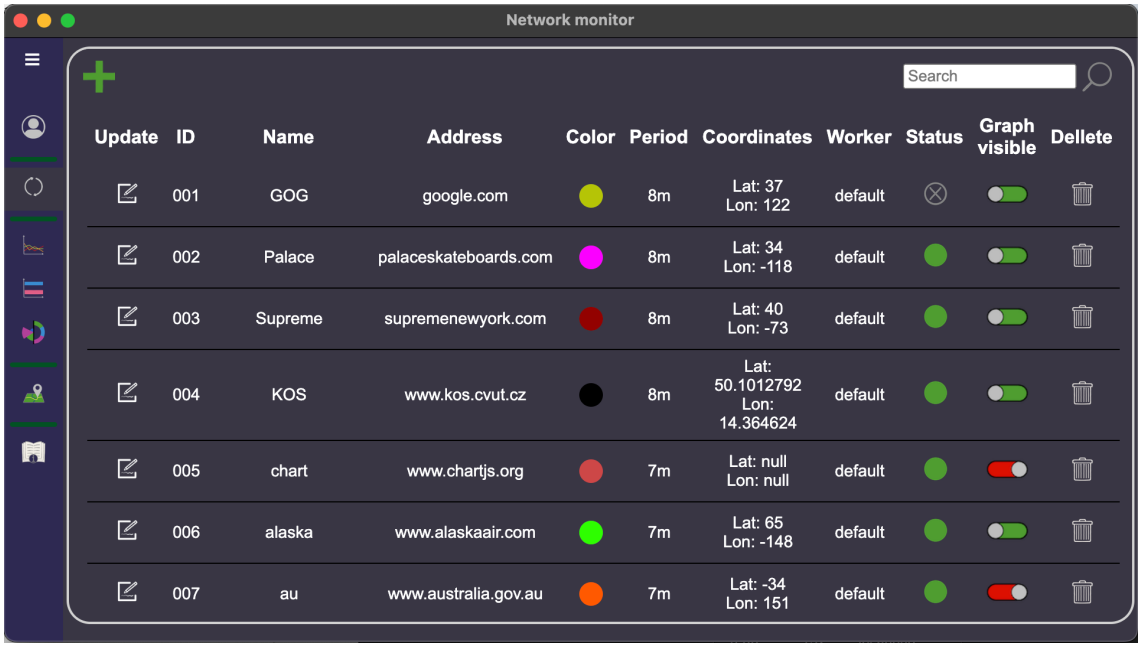
```
mainWindow = new BrowserWindow({
// Načtení certifikátu a klíče pro TLS komunikaci
const cert = fs.readFileSync(path.join(__dirname, 'certif/Localcrt.pem'));
const key = fs.readFileSync(path.join(__dirname, 'certif/LocalKey.pem'));

//Vytvoření agenta, který bude zajišťovat zabezpečenou výměnu dat se serverem
  const agent = new https.Agent({
    cert: cert,
    key: key,
    rejectUnauthorized: false//Pro samo podepsané certifikáty
  });

// nastavení hlavičky http requesty
  axiosInst = axios.create({
    httpsAgent: agent,
    baseURL: "https://127.0.0.1:8000" // Výchozí URL pro komunikaci
  })

// vložení tokenu, který byl přijat od serveru při přihlášení
fh = {headers: {
        'Authorization': `Bearer `+ accessToken,
        'Accept': 'application/json'
        }
      }

axiosInst.post("/init", null, fh)
  .then(function (response) {
      // Provede se v případě úspěšného spojení response jsou vrácená data
  })
  .catch(function (error) {
      // V případě erroru se provede tato část
    }
  })
  .finally(()=>{
      // Provede se v každém případě
  })
```

Kód 16 Nastavení requestů pomocí axios a jeho zabezpečení pomocí TLS v main.js.

3.6 Zobrazení a úprava úloh na serveru

V této části se již budu věnovat samotnému rozložení a úpravám ze strany aplikace k serveru. Tabulka je na (obr. 28). Pro zobrazení právě spuštěných úloh využiji tabulky, která se bude skládat z několika sloupců a obsah řádků se po stažení dat ze serveru vždy aktualizuje a promítne do tabulky. Nad tabulkou se nachází tlačítko + k přidání dalšího testování. Po jeho kliknutí se objeví pole pro přidání nového úkolu (obr. 29). Po pravé straně se nachází vyhledávací pole pro zadání jména serveru, vedle kterého bude . Po zadání jména a stisknutí vyhledávacího tlačítka se otevře okno k úpravám, kde uživatel může rovnou daný test upravit nebo pozastavit. V případě že bude zadáno jméno, které není v seznamu, objeví se vyskakovací okno, kde bude uživatel upozorněn, že se tato položka v tabulce nenachází.

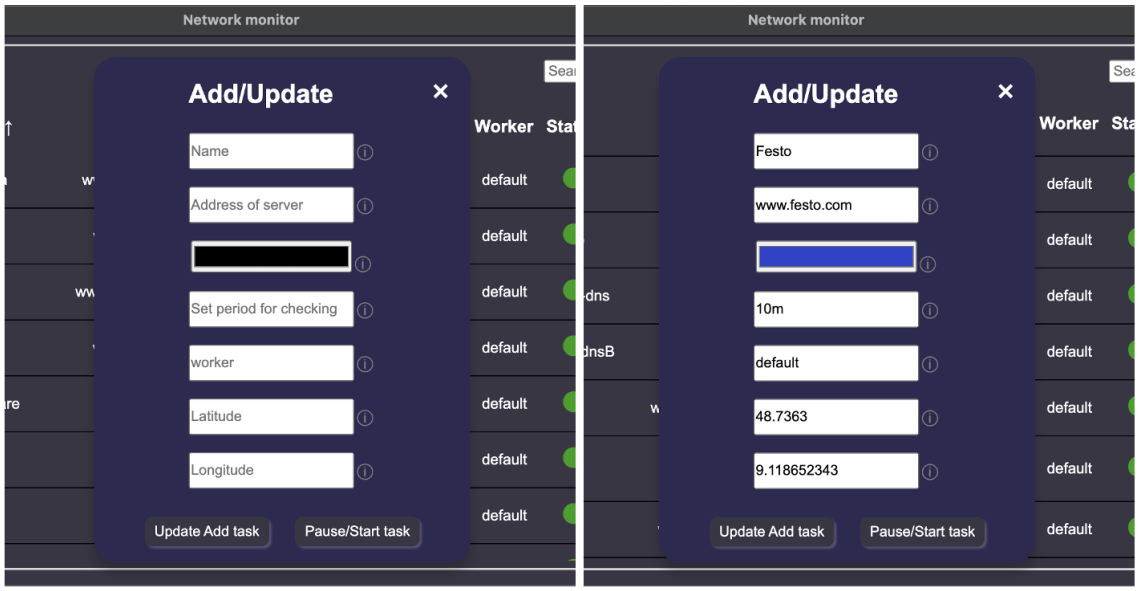
V tabulce je možnost změny řazení podle sloupců. Ve výchozím nastavení se tabulka řadí podle identifier (ID). Uživateli je umožněno řadit tabulku podle slepců Name, Address, Status a Graph visible. Vždy po kliknutí na příslušnou položku se v hlavičce tabulky řádky seřadí podle abecedy, čísel nebo stavu (pravda/nepravda) vzestupně, po dalším kliknutí se naopak bude řadit sestupně. Tabulka je tvořená čistě pomocí JavaScriptu a není využívána knihovna. Nastavení a obsluha tabulek je v renderu *table.js*.



Obr. 28 Okno s tabulkou prováděných testů stavu.

Kolonky v řádcích sloupců Update, Status a Graph visible obsahují tlačítka, pod kterými se nachází následující funkce:

- Update – na příslušném řádku úkolu se objeví okno (obr. 29) s předvyplněnými políčky vlastnostmi upravovaného úkolu.
- Status – převrátí aktuální hodnotu stavu. To znamená, že když úkol běží je zde  Po kliknutí se tento úkol na serveru zastaví a objeví se  na znamení, že se daný úkol neprovádí, ale veškerá data zůstanou zachovaná.
- Graph visible – tento přepínač značí zobrazení daného úkolu v grafech. Když je  znamená že se nebude načítat do grafů. Tento symbol  značí, že v grafech budou tato data načítána.
- Delete – Po kliknutí na tuto ikonu se daný úkol smaže včetně dat, která byla naměřená.



Obr. 29 Vyskakovací okénko pro přidání úkolu (vlevo) a aktualizaci úkolu (vpravo).

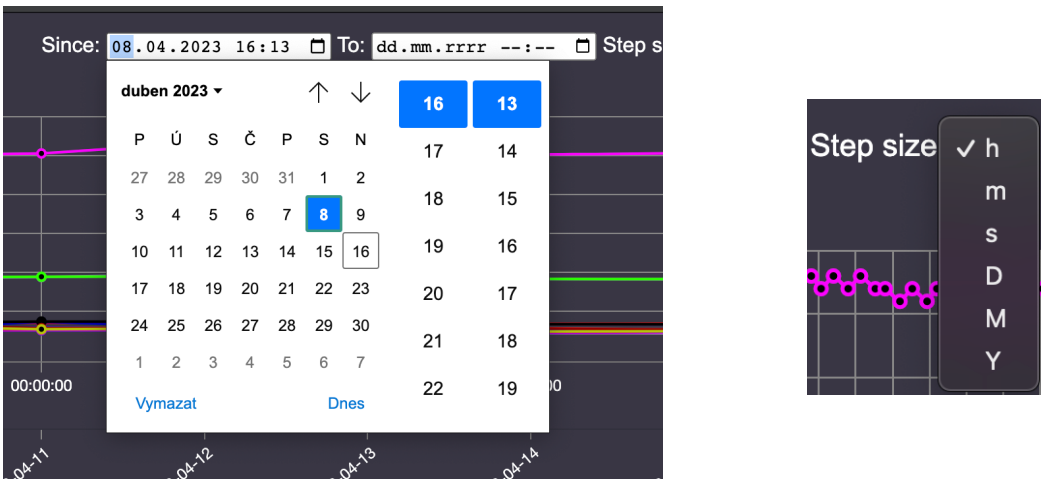
3.7 Grafy, jejich nastavení a obsluha pro zobrazení dat ze serveru

Rozložení okna grafů jsem zvolil tak, že v horní části nad grafem se bude nacházet ovládací panel (obr. 30). S jeho pomocí bude uživateli usnadněná manipulace s daty a nastavením grafů. Toto pole bude společné pro veškeré grafy.

Obr. 30 Menu pro grafy.

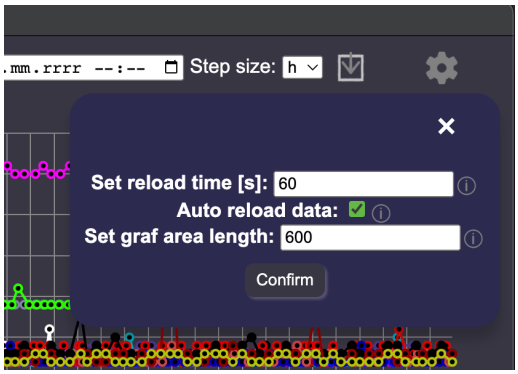
Panel se skládá ze tří sekcí:

- V první sekci je uveden čas do dalšího načtení dat a tlačítko pro načtení nových dat. Data se v tomto případě načítají maximálně jeden den zpět.
- Ve druhé sekci je zobrazeno zadávací pole data a času pro hodnotu, od a do které se data budou načítat (obr. 31). V případě, že uvedená pole zůstanou nevyplněná, tak časový údaj nebude brán v potaz. Následuje rozbalovací menu, ve kterém je možné zvolit krok pro načítaná data (obr. 31). Za ním je tlačítko pro stažení zvolených dat.



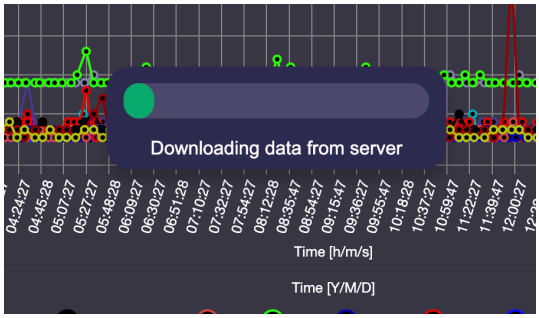
Obr. 31 Výběr času, od kterého se budou načítat hodnoty (vlevo) a rozbalovací menu (vpravo).

- V poslední Sekci je tlačítko nastavení. Po kliknutí na toto tlačítko se rozbalí menu s nastavením (obr. 32). Toto okno obsahuje tři nastavení. Prvním je perioda, ve které se data budou načítat a výchozí hodnotou je 60 s, minimální hodnotou je 20 s. Pak následuje zaškrtávací pole, které udává, jestli je automatické načítání dat zapnuté. V případě zaškrtnutého políčka je automatické načítání zapnuté a v opačném případě je automatické načítání vypnuté. Toto menu je možné zavřít kliknutím na křížek nebo kliknutím mimo toto pole.



Obr. 32 Okno s nastavením pro auto obnovování dat v grafu.

Při stahování dat je uživatel s aktuálním stavem seznámen pomocí postupně vyplňující se stavové lišty (obr. 33). Toto je vhodné v případě stahování většího množství dat nebo při pomalejším internetovém připojení.



Obr. 33 Načítací lišta se stavem stahování.

Formát stahovaných dat, jeho úprava pro grafy a mapu

Data, která stahuji ze serveru, jsou nutná upravit do potřebného formátu pro další práci a zobrazení v grafech nebo mapě. V aplikaci při stahování dat z testovacího serveru využívám dva requesty. První pro stažení tabulky s parametry, kterými jsou barva, souřadnice, jméno atp. (Kód 17) zde jsou stažené parametry pro testování serverů – uvedená data jsou pro tabulku, graf i mapu. Toto pole obsahuje všechny parametry testů, které se provádí na serveru. Pro zkrácení zde uvádím pouze jeden testovaný server, který je na adrese KOS.cvut.cz. Z těchto dat mohu tvořit tabulku i mapu, postačí mi pouze vyfiltrovat data, která jsou potřeba. Například v mapě potřebuji jméno serveru a souřadnice. Tento výběr probíhá pomocí cyklu, který vybírá název a souřadnice pro mapu. Pro grafy navíc probíhá kontrola, zda se daná data do grafů mají vykreslit. Tato informace je obsažena v části *hide*. V případě že je hodnota true dané výsledky testů v grafu nebudou vykresleny při hodnotě false se data do grafů vykreslí.

```
[{
  "longitude": 14.364624,
  "worker": "default",
  "runing": true,
  "task": "ping",
  "address": "www.kos.cvut.cz",
  "color": "#000000",
  "name": "KOS",
  "latitude": 50.1012792,
  "hide": false,
  "frequency": "8m",
  "id": 9
},
...
]
```

Kód 17 Data pro test na KOS.cvut.cz.

V (Kód 18) je ukázka třídění dat pro mapu. Tato data se načtou do pole, které se bude dále nahrávat do samotné mapy pomocí značek na daných souřadnicích.

```
data.forEach(element => {
      if (element.latitude!=null && element.longitude!=null){
        locations.push([element.name, element.latitude, element.longitude])
      }
    });
```

Kód 18 Třízení dat pro tabulku, mapu a grafy pro KOS.cvut.cz.

Druhý request slouží pro stažení naměřených dat v daném intervalu nebo všech naměřených dat. V (Kód 19) je ukázka stažených dat od 5.5.2023 16:00:47 – 16:01:52. Jak je zde patrné, data nejsou nijak upravená, a každé měření obsahuje adresu, čas testu (*time*) v milisekundách od času 1.1.1970 0:0:0 a hodnotu (*value*), která byla naměřená. Tato data upravuji do formátu, který je vhodný pro grafy. Převádím čas na uživatelsky přívětivější formát, posléze počítám počet měření pro každé testování a součet všech časů odezvy, z čehož dopočtu průměrnou hodnotu odezvy.

```
[
    {
      "address": "www.kos.cvut.cz",
      "task": "ping",
      "time": 1683302447910,
      "value": 4,
      "worker": "default"
    },
...
    {
      "address": "www.kos.cvut.com",
      "task": "ping",
      "time": 1683302512110,
      "value": 3,
      "worker": "default"
    },
  ]
```

Kód 19 Stahovaná data jednotlivých měření.

Do grafů využívám data v kombinaci z obou requestů. Jméno serveru je nutné spojit s adresou testovaného serveru pro názvy dat v grafech a ověřit, jestli daný test má být v grafu vykreslován. Výstupem je pole dat pro každý z testovaných serverů. V (Kód 20) je uveden testovaný server KOS, v němž se nachází potřebné informace pro vykreslení grafů. Na počátku je uvedena adresa, počet neúspěšných a úspěšných měření pro stejnojmenný sloupcový graf a hodnota průměrné odezvy pro koláčový graf. Následuje label, kterým budou data popsána – jedná se o uživatelem dané jméno. Předposlední část je pro barvu daného serveru. Poslední částí jsou data, která budou v lineárním grafu vykreslena pro osu y. Data pro prstencový graf celkového počtu měření jsou pouze součtem napočítaných úspěšných testů a testů neúspěšných.

```
{
    "address": "www.kos.cvut.cz",
    "failed": 0,
    "passed": 2,
    "average": 3.5,
    "label": "KOS",
    "borderColor": "#000000",
    "backgroundColor": "black",
    "data": [
        4,
        3
    ]
}
```

Kód 20 Data připravená pro distribuci do grafů.

Hodnoty času, které budou sloužit jako popis bodů na ose y, jsou v poli nazvaném data v (Kód 20), kdy každá první hodnota v daném načítaném dni má navíc rok, měsíc a den pro sekundární osu. Tyto hodnoty jsou získávány postupným procházením naměřených hodnot z (Kód 19). Vždy když přijde hodnota, která má čas měření nevyskytující se v grafu, přidá se tato hodnota na další pozici do pole. Výsledné pole pro popisky osy x je v (Kód 21).

```
[
      ["16:00:47", "2023-5-5"],
      ["16:01:52"]
]
```

Kód 21 Časy pro popis osy x.

Tato data budou v následující části využívána k příkladům, kdy je budu vkládat do grafů a mapy.

Graf odezvy serverů

Pro zobrazení dat odezvy testů jsem zvolil lineární graf, ve kterém budou vykreslené hodnoty odezvy podle [24]. Pro grafy jsem vyhledal a vybral knihovnu *Chart.js*. Tato knihovna umožňuje jednoduché nahrání a aktualizaci dat do grafů.

Pro instalaci knihovny je (Kód 22) zadaný v kořenovém adresáři v příkazovém řádku. Po instalaci jsem do souboru HTML vložil následující (Kód 23). Veškeré úpravy a nastavení grafů již probíhá v souboru *graphs.js* (Kód 24). Zde uvedu příklad, jakým způsobem se vytvoří jednoduchý lineární graf. V první řadě načtu prvek z HTML podle ID. V tomto prvku bude následně obsažen vykreslený graf. Proměnná *cofigL* (Kód 24) obsahuje nastavované vlastnosti jako je typ grafu, data obsahují *labels* – hodnoty na ose x a *datasets* – zde je pole vykreslovaných dat do osy y, jméno a barva čáry. Závěrečný příkaz nahraje nastavené hodnoty a aktualizuje pole grafu s novými daty.

```
instalace chart.js
      npm install chart.js
```

Kód 22 Příkaz k instalaci chart.js knihovny.

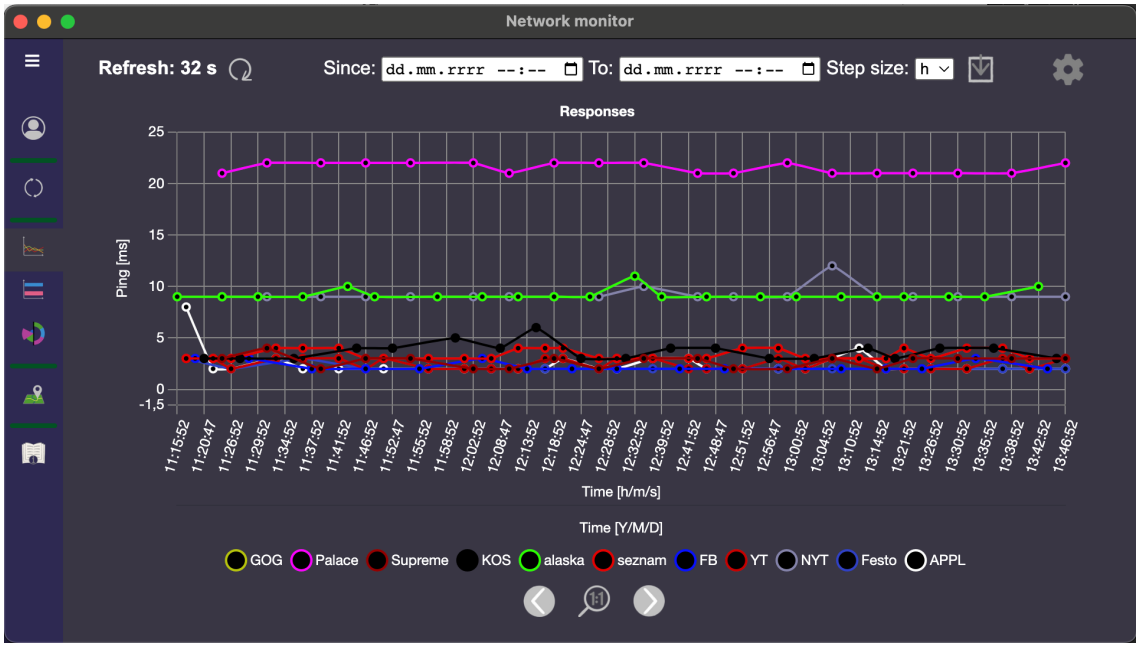
```
<canvas id="myChart"></canvas>
```

Kód 23 HTML kód kde se nastavuje prostor pro lineární graf.

```
const Lchart = document.getElementById('myChart'). //načtení prvku podle ID
//definice vlastností grafů
const configL = {
  type: 'line',
  data: {
    labels: [["16:00:47", "2023-5-5"], ["16:01:52"]],
    datasets: [
      {
          label: 'KOS',
          borderColor: '#ff0000',
          backgroundColor: 'rgba(0, 0, 0, 0)',
          data: [4, 3]
      }]
  },
};
//nahrání nastavených hodnot a jejich aktualizace
var myChart = new Chart(Lchart, configL);
myChart.update();
```

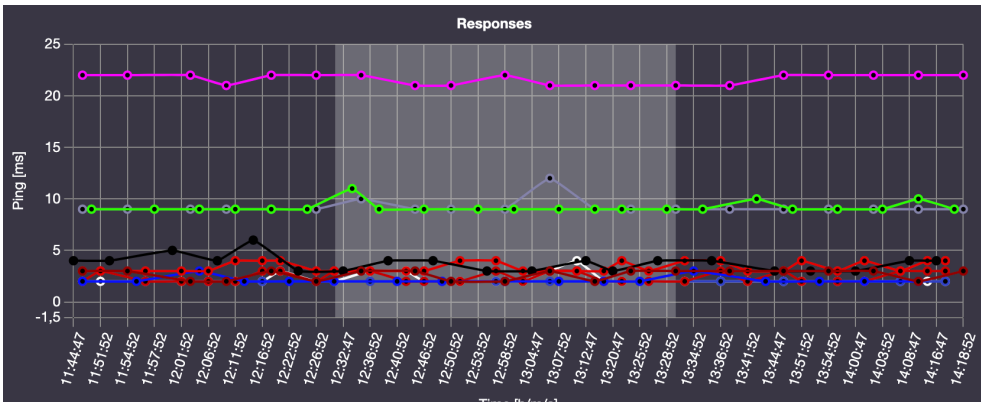
Kód 24 Definice lineárního grafu v graphs.js.

Na (obr. 34) je graf zobrazující hodnoty odezvy serverů v čase. Seznam jmen testovaných serverů a jejich barev je umístěn v legendě pod grafem. Po kliknutí na jednotlivá jména testovaných serverů lze hodnoty v grafu skrýt. Po opětovném kliknutí se hodnoty opět objeví. Pod legendou jsou ovládací tlačítka, která slouží posouvání grafu při přiblížení. Prostřední tlačítko zajišťuje obnovení přiblížení do původního stavu.



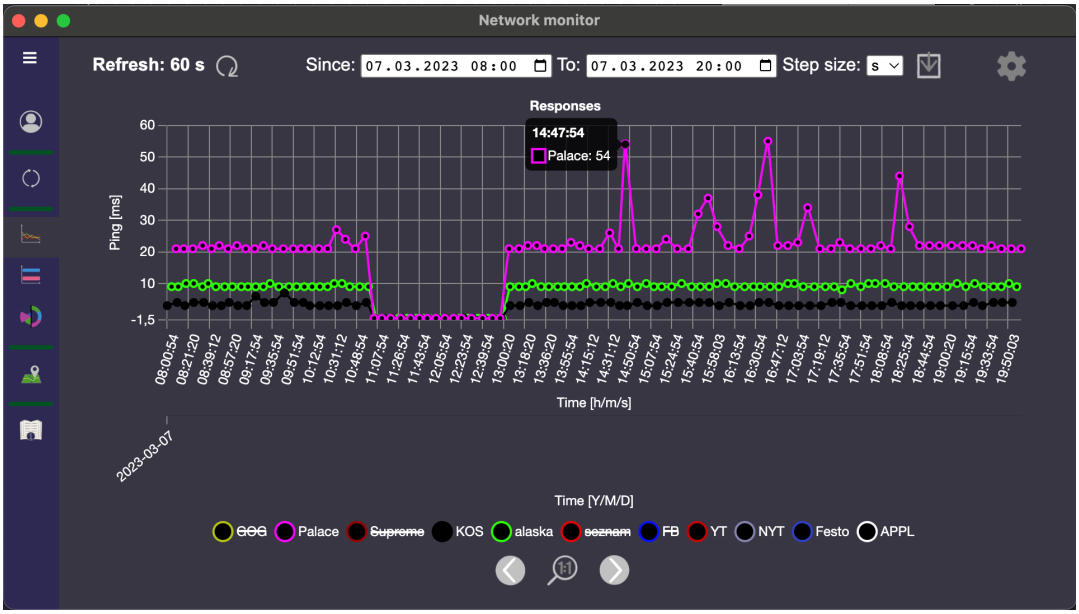
Obr. 34 Okno s lineárním grafem s aktuálním měřením.

Přiblížení se provádí kliknutím a tažením výběrového pole (obr. 35). Pro graf jsem zvolil dvě osy x1 a x2. X1 je čas v hodinách, minutách a sekundách a x2 představuje rok, měsíc a den. Na ose y je hodnota odezvy serverů v ms.



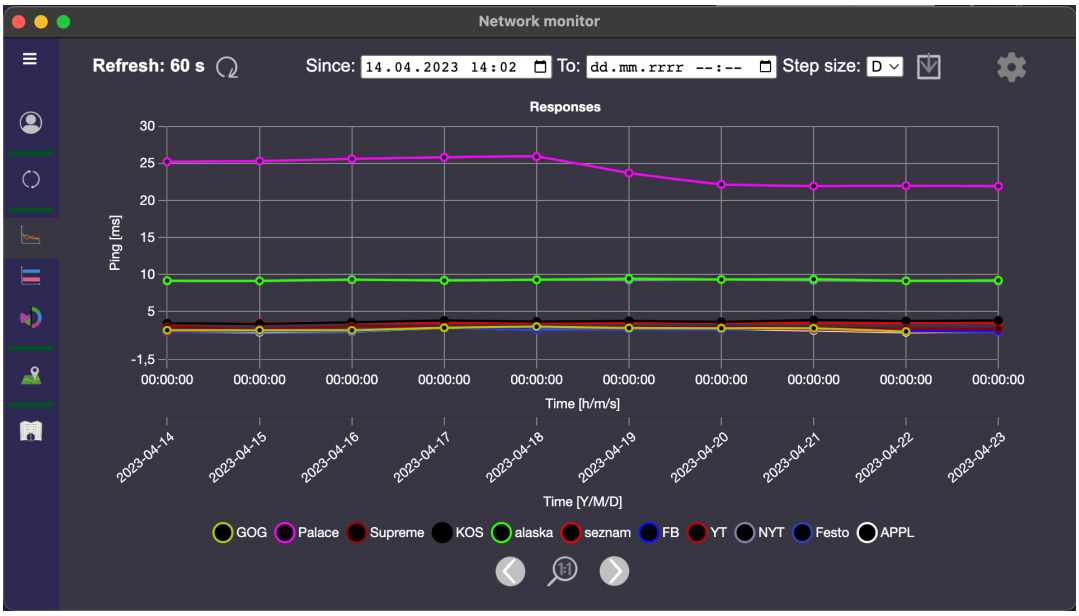
Obr. 35 Přiblížení grafu pomocí kliknutí a tažení.

Na (obr. 36) je graf, kde jsou načtená data v zadaném intervalu. Na testovacím serveru byl simulován výpadek internetu, z čehož plyne nedostupnost služeb. Tento výpadek se projevuje zápornou hodnotou odezvy viditelnou v grafu pro všechny testované servery. Po najetí myší na vybraný bod v grafu se zobrazí vyskakovací popisek s informacemi pro daný bod. V této simulaci jsem vypnul zobrazování serveru seznam. Z toho plyne skrytí hodnot červené barvy a škrtnutí názvu „seznam“ v legendě.



Obr. 36 Graf s výpadkem služeb, skrytými položkami a bublinou se jménem a hodnotou po najetí myší.

Následující graf zobrazuje testování vybraných testovaných serverů v delším časovém úseku. Na (obr. 37) jsou načtená data, která jsou OD zadaného data v poli „Since“ DO aktuálního času (nevyplněné pole) „To“. Hodnoty jsou zaokrouhlené na dny, protože se jednalo o delší časový úsek než v minulém případě.



Obr. 37 Načtené hodnoty od zadaného data zaokrouhlené na dny.

Graf počtů úspěšných spojení a selhání při spojení

Další graf pro zobrazení dostupnosti jsem zvolil horizontální sloupcový, ve kterém jsou u každého jména testovaného serveru vždy dva sloupce. Jeden znázorňuje počet úspěšných testů a druhý počet těch neúspěšných. Knihovna *Chart.js* je i pro tento graf shodná jako u předchozího grafu, tudíž instalace je již provedena. Vykreslení v HTML je pomocí následujícího řádku (Kód 25).

```
<canvas id="BarChart"></canvas>
```

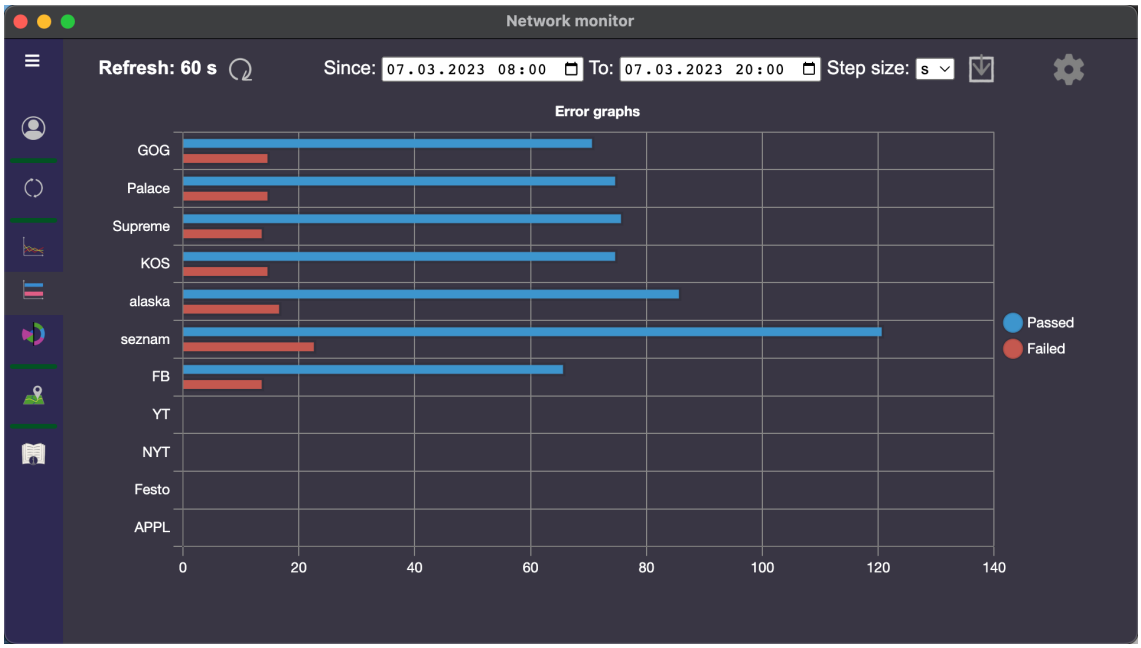
Kód 25 HTML kód kde se nastavuje prostor pro sloupcový graf.

(Kód 26) představuje nastavení vzhledu pro sloupcový graf. Uvádím zde průběh nastavení a nahrání dat pro jejich vykreslení v grafu. V tomto příkladu je uveden jeden testovaný server, u něhož jsou vykresleny dva sloupce, které představují počty testů. Modrý sloupec je pro úspěšné testy a červený představuje testy neúspěšné. V (Kód 26) je podobné nastavení jako u lineárního grafu (Kód 24). Prvky se také načítají podle ID z HTML stránky, ale rozdíl je v části konfigurace a datech. Závěrečný příkaz nahraje nastavené hodnoty a aktualizuje pole grafu s novými daty.

```
const Bchart = document.getElementById('BarChart').//načtení prvku podle ID
const configB = {.  //definice vlastností grafů
  type: 'bar',
  data: {
    labels: ["KOS"],
    datasets: [
      {      // úspěšné testy nastavená barva je červená
        label: "Passed",
        backgroundColor: ["#3e95cd"],
        data: [2]
      },
      {      // neúspěšné testy nastavená barva je červená
        label: "Failed",
        backgroundColor: ["#c45850"],
        data: [0]
      }
  ]}
};
var Bchart = new Chart(Bchart, configB); //nahrání nastavených hodnot
Bchart.update();//aktualizace grafu
```

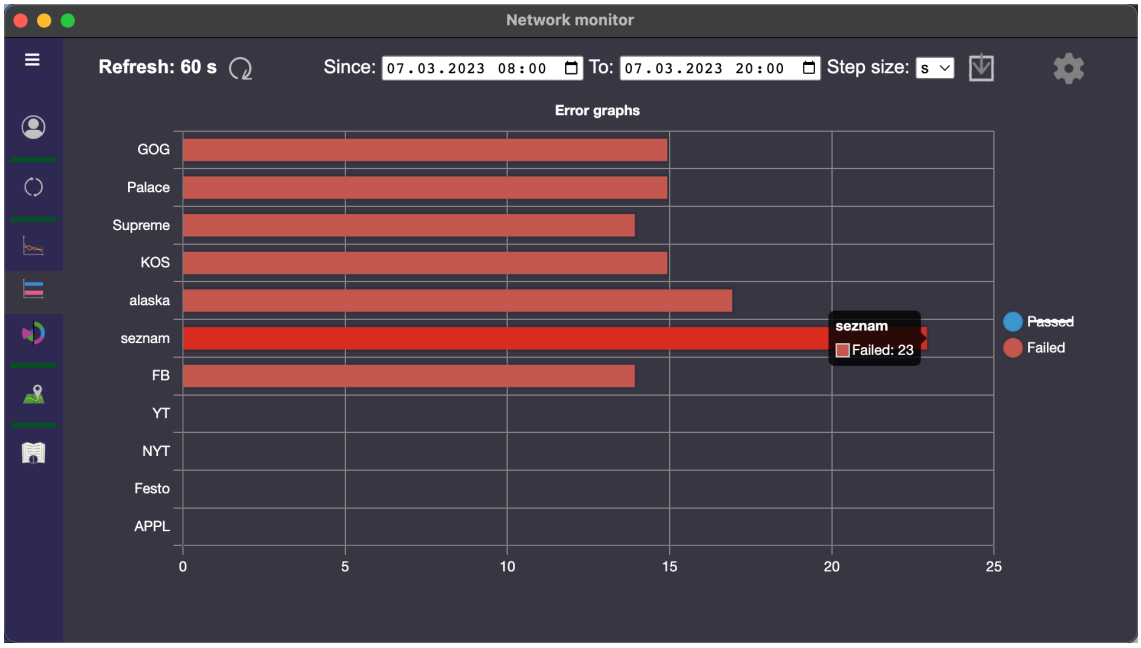
Kód 26 Definice sloupcového grafu v horizontální poloze v graphs.js.

Na (obr. 38) vpravo vedle grafu se nachází legenda pro skrývání a zobrazování dat. Na příkladu je graf, ve kterém jsou načtena data v zadaném intervalu OD zadaného data v poli „Since“ DO data v poli „To“. Je zde patrný poměr úspěšných testů a testů neúspěšných.



Obr. 38 Horizontální graf úspěšných měření a měření kdy byl server off-line.

Na (obr. 39) je graf, ve kterém byl počet úspěšných testů vypnutý a uživatel může porovnat počty testů, které nebyly úspěšné. Po najetí myší na jednotlivé sloupce se uživateli zobrazí vyskakovací popisek s informacemi pro daný sloupec.



Obr. 39 Horizontální graf zobrazující pouze počet neúspěšných spojení.

Graf průměrné odezvy a poměr provedených testování

Pro zobrazení průměrné odezvy a celkového počtu testů jsem zvolil koláčový a prstencový graf. Koláčový graf slouží pro zobrazení průměrné odezvy serverů. Lze z tohoto grafu dobře posoudit, jakou odezvu měl každý server v porovnání s ostatními testovanými servery. Prstencový graf naopak dobře znázorňuje porovnání celkového počtu provedených testů pro každý ze serverů. Tyto grafy jsem se rozhodl dát dva do jednoho podokna. Mají podobu kruhů tudíž by nebylo možné samostatný graf roztáhnout a vyplnit jím celou šířku okna, jak to je možné u předchozích dvou grafů. Legenda vykreslovaných testů je umístěná pod každým z grafů.

Knihovna pro tyto grafy je shodná jako u předchozích grafů a je jí *Chart.js*. Instalace je tedy již provedená. Vykreslení grafů v HTML představují následující řádky v (Kód 27).

```
<div id="left">
  <canvas id="Pchart"></canvas>
</div>
<div id="right">
  <canvas id="Dchart"></canvas>
</div>
```

Kód 27 HTML kód kde se nastavuje prostor pro koláčový a prstencový graf umístěné budou vedle sebe.

Prvky se také načítají podle ID z HTML stránky, ale rozdíl je v části konfigurace a datech. Závěrečný příkaz nahraje nastavené hodnoty a aktualizuje pole grafu s novými daty, jak je v (Kód 28).

```
const Pchart = document.getElementById('BarChart').//načtení prvku podle ID
//definice vlastností grafů
const configP = {
  type: 'polarArea',
  data: {
    labels: ["KOS"],
    datasets: [{
        data: [3.5],
        backgroundColor: ['rgba(0, 0, 0, 1)']
      }]
  }
}
//nahrání nastavených hodnot a jejich aktualizace
var Pchart = new Chart(Pchart, configP);
Pchart.update();
```

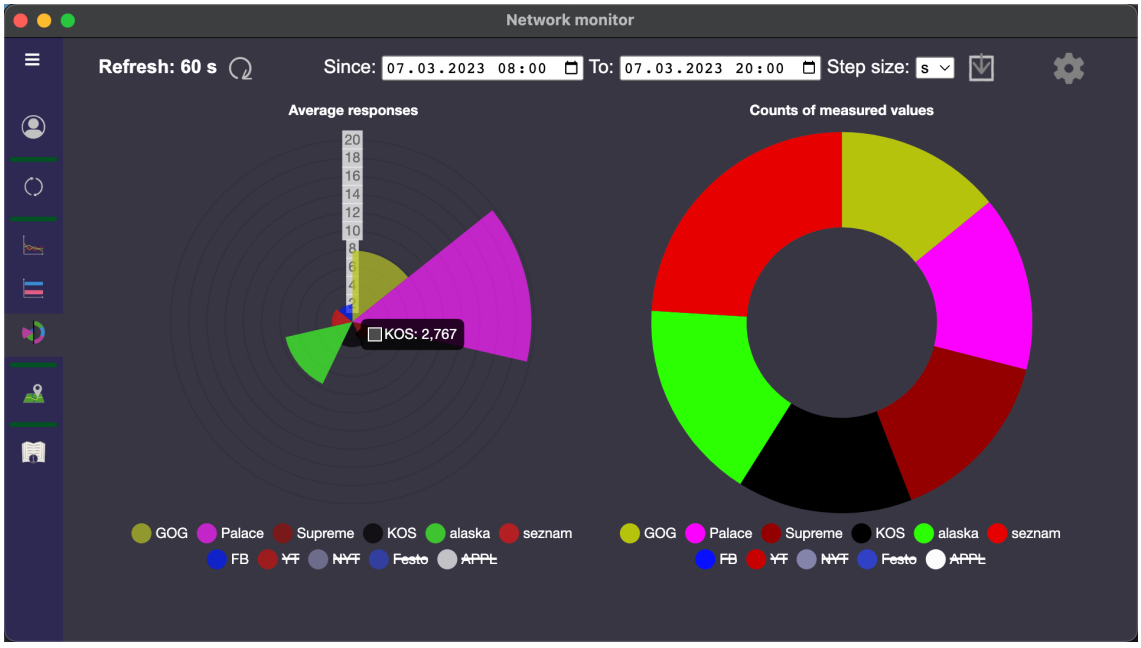
Kód 28 Definice koláčového grafu v graphs.js.

Prvky se i zde načítají podle ID z HTML stránky, ale rozdíl je opět v části konfigurace a datech. Závěrečný příkaz nahraje nastavené hodnoty a aktualizuje pole grafu s novými daty.

```
const Dchart = document.getElementById('Dchart').//načtení prvku podle ID
//definice vlastností grafů
const configD = {
  type: 'doughnut',
  data: {
    labels: ['KOS'],
    datasets: [{
      data: [2],
      backgroundColor: ['rgba(0, 0, 0, 1)']
    }],
  }
};
//nahrání nastavených hodnot a jejich aktualizace
var Dchart = new Chart(Dchart, configD);
Dchart.update();
```

Kód 29 Definice prstencového grafu v graphs.js.

Koláčový graf průměrné odezvy je umístěn vlevo a vpravo je umístěn prstencový graf počtu naměřených hodnot, jak je možné vidět na (obr. 40). V grafech je možné jednotlivé testované servery zneviditelnit pomocí kliknutí na spodní legendu. Škrtnutý název serveru znamená skrytí v grafu. Po najetí myší na jednotlivé výseče se objeví vyskakovací popisek s informacemi o názvu a přesné hodnotě průměrné odezvy nebo počtu testů.



Obr. 40 Grafy celkového počtu testů (vpravo) a průměrné odezvy (vlevo).

3.8 Mapa pro zobrazení pozic serverů

K zobrazení pozic serverů v mapě jsem zvolil knihovnu *Leaflet*, která umožňuje zobrazení mapy světa a dynamické přidávání bodů do této mapy podle [25]. Pro instalaci knihovny je (Kód 30) zadaný v kořenovém adresáři v následujícím příkazovém řádku.

```
instalace leaflet
        npm install leaflet
```

Kód 30 Příkazy k instalaci leaflet knihovny.

Vykreslení mapy v HTML je pomocí následujícího řádku (Kód 31). Zde nastavím oblast, do které se bude mapa vykreslovat a nastavím jí ID.

```
<div id="map"></div>
```

Kód 31 Řádek inicializace mapy v HTML.

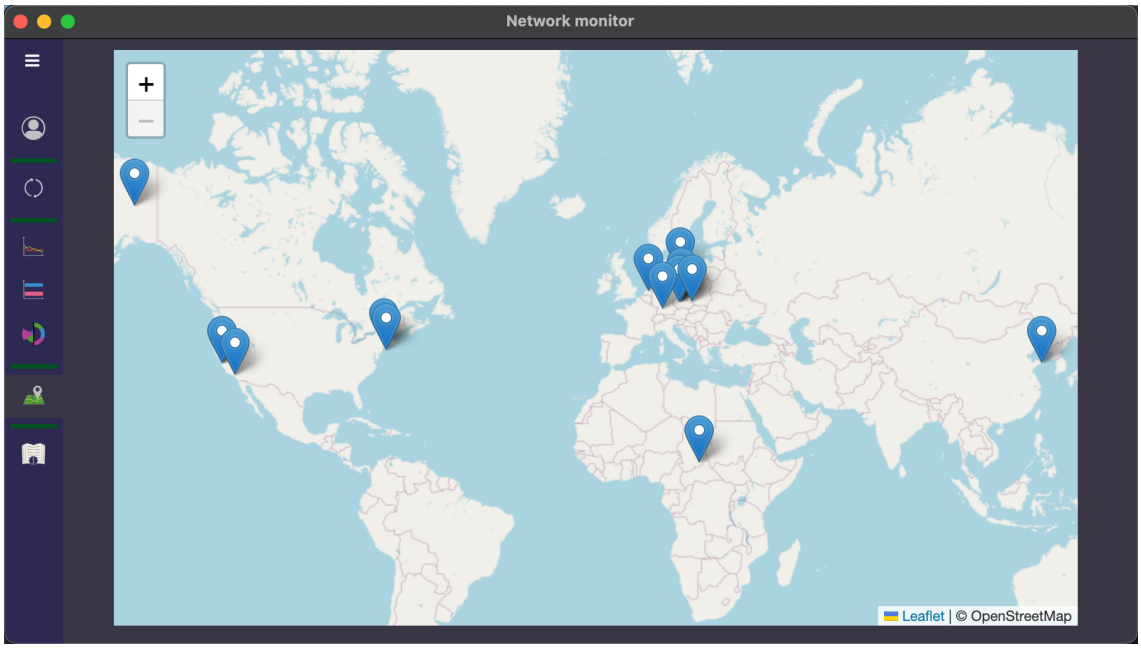
V (Kód 32) je uvedeno nastavení mapy v *maps.js*. V první řadě se inicializuje pole pro vykreslení podle ID a nastaví se výchozí pohled. Dále jsem nastavil adresu odkud se mají stahovat mapové podklady. V poslední části je přidání značky do mapy a nastavení co se má dít po kliknutí na značku. V tomto případě se jedná o vyskakovací popisek s názvem serveru.

```
//inicializace mapy podle ID
var map = L.map('map').setView([50.087465, 14.421254],3)
//nastavení odkud se data do mapy mají stahovat
L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 8,
    minZoom:2,
    attribution: '© OpenStreetMap',
}).addTo(map);

//přidání značky a bubliny po kliknutí
L.marker([14.364624, 50.1012792], {riseOnHover: true})
          .bindPopup('KOS')
          .addTo(map);
```

Kód 32 Příkazy k inicializaci mapy a nastavení bodu s bublinou po kliknutí v maps.js.

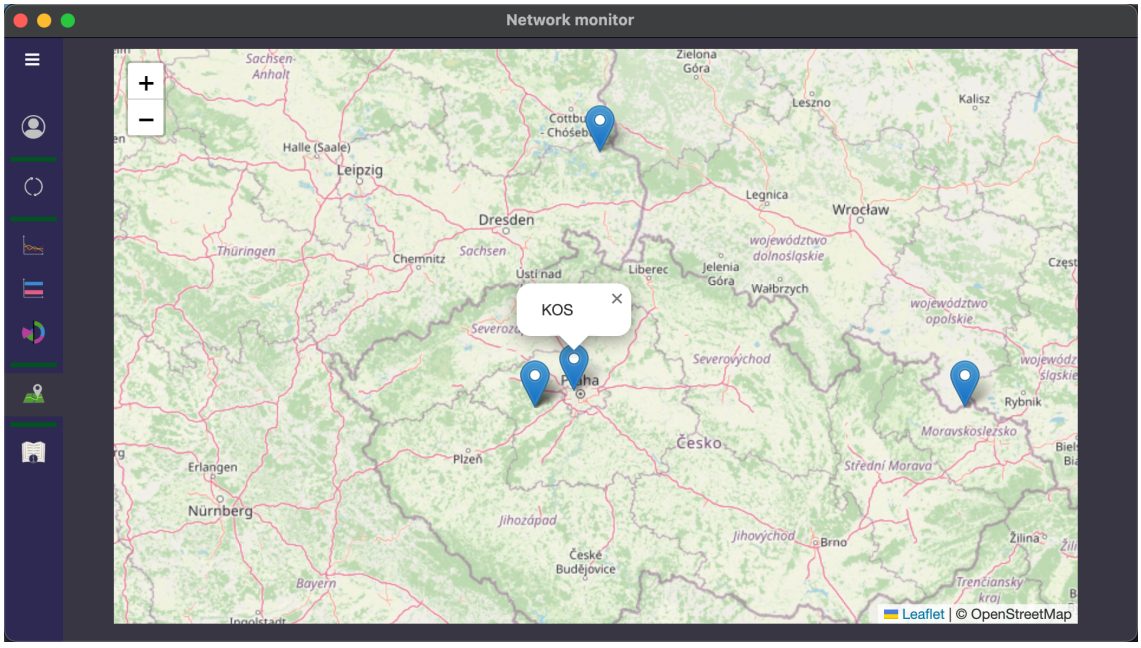
Mapa je umístěna na oblast jednoho z podoken (obr. 41). Mapa se dynamicky mění se změnou velikosti okna aplikace. Po přihlášení a stažení dat si podprogram *maps.js* vyžádá data obsahující jméno a souřadnice testovaných serverů. V případě, že tyto

hodnoty uživatel nevyplnil, nebude se tento bod zobrazovat v mapě. V levém horním rohu jsou dvě tlačítka +/- pro přiblížení nebo oddálení mapy.



Obr. 41 Okno s mapou znázorňující polohu zadaných serverů.

Na (obr. 42) je znázorněn vyskakovací popisek s názvem serveru, který se na daném místě nachází. Po kliknutí na křížek v pravém rohu bubliny dojde k jejímu zavření. Při kliknutí na jiný bod v mapě se předchozí vyskakovací popisek zavře.



Obr. 42 Bublina se jménem serveru po kliknutí.

3.9 Sestavení, distribuce a aktualizace výsledné aplikace

K distribuci aplikace vytvořené pomocí frameworku Electron, jsem využil knihovny *electron-builder* podle [26]. Pro instalaci knihovny je (Kód 33) zadaný v kořenovém adresáři v příkazovém řádku. Tato knihovna bude instalována pouze pro vývojářský režim, to znamená v distribuované verzi aplikace nebude tato knihovna dostupná.

instalace electron-builder

```
npm install electron-builder --dev
```

Kód 33 Příkazy k instalaci electron-builder knihovny.

Nastavení distribuce probíhá v *package.json* – část k distribuci je v (Kód 34). První položkou je název aplikace, pak verze distribuované aplikace. Dále nastavuji složku, ve které se nachází hlavní spouštěcí soubor *main.js* a následují složky se soubory pro běh aplikace (.js, .html, .css) v poli *files*. V *repository* uvádím adresu repositáře, ve kterém bude aplikace distribuována. Pod *publish* jsou informace k distribuci. V *build* se nachází ID aplikace, název aplikace a složka, do které bude vytvořený instalační soubor aplikace uložen. Nakonec je zde uvedeno umístění, ve kterém se nachází ikony pro jednotlivé OS.

```
"name": "Network-monitor",
"version": "2.2.2",
"main": "src/main.js",
"files": [ "src/**/*" ],
"repository": "https://github.com/Guestas/GUI-network-monitor",
"publish": {
  "provider": "github",
  "releaseType": "release"
},
"build": {
  "appId": "com.MCNetworkMonitor.app",
  "productName": "Network Monitor app",
  "directories": { "output": "dist" },
  "win": { "icon": "ic/win.ico" },
  "mac": { "icon": "ic/mac.icns" },
  "linux": { "icon": "ic/linux.png" }
},
"scripts": {
    "distW": "electron-builder --win",
    "distA": "electron-builder --mac",
    "distL": "electron-builder --linux snap"
}
```

Kód 34 Natavení distribuce aplikace v package.json.

V (Kód 34) je poslední položkou *scripts* – v níž jsou skripty pro spuštění distribuce pro každý ze tří hlavních OS. Spuštění skriptu k distribuci aplikace probíhá pomocí (Kód 35), který zadávám v příkazovém řádku v kořenovém adresáři aplikace.

```
Spuštění distribuce na windows
      npm run distW
Spuštění distribuce na Mac
      npm run distA
Spuštění distribuce na Linux
      npm run distL
```

Kód 35 Příkazy k vytvoření instalačních souborů pro Windows, Mac a Linux.

Soubory, které po spuštění skriptu knihovna vytvoří, je dále nutné přejmenovat a změnit mezery v názvech na „-“. Tyto soubory dále nahrávám na GitHub do *Releases*, kam vložím do políček *tag* a *version* – verzi aplikace jaká je uvedená v (Kód 36), tudíž verze 2.2.2.

Pro aktualizace aplikace jsem zvolil knihovnu *Electron-updater*. Instalace probíhá pomocí (Kód 36) zadaného v příkazové řádce v kořenovém adresáři.

```
instalace electron-updater
      npm install electron-updater
```

Kód 36 Příkazy k instalaci electron-updater knihovny.

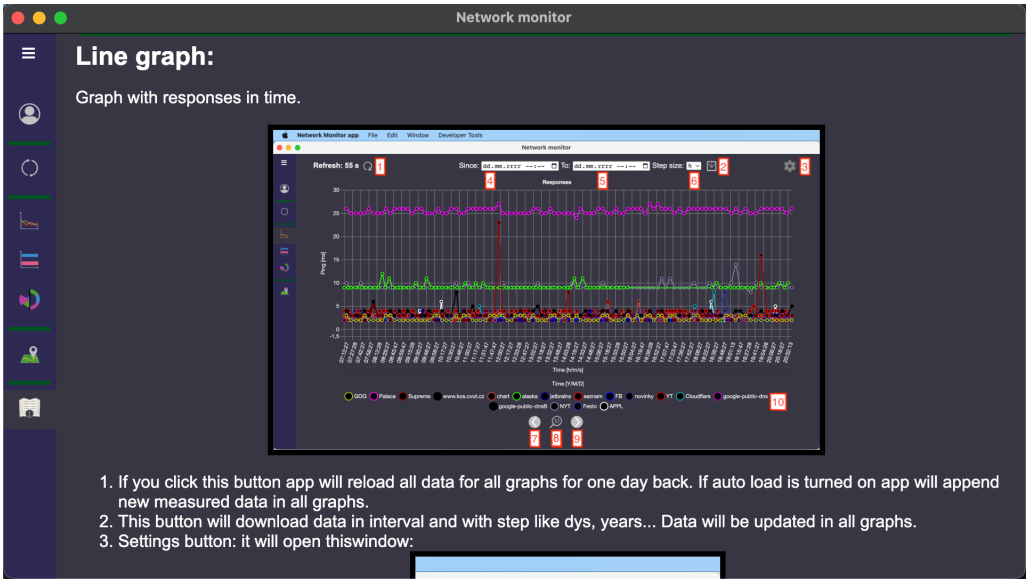
V (Kód 37) představuje postup, který kontroluje aktualizace na GitHub. V případě, že uživatel využívá starší verzi aplikace, vyskočí oznámení s upozorněním na novou verzi. Když uživatel aktualizaci potvrdí, zahájí se stahování souborů. Po dokončení stahování se zobrazí další upozornění s oznámením o dokončení stahování. Po potvrzení tohoto upozornění se zahájí vypnutí aplikace, aktualizace a následný restart aplikace. Tímto je zajištěn bezchybný chod a komunikace GUI se serverovou aplikací. V případě změny nebo aktualizace serverové aplikace se může změnit struktura komunikace a odpovědí ze serveru. To by omezilo nebo znemožnilo funkcionalitu uživatelského komfortu.

```
// funkce pro hledání aktualizace při nové verzi se provede
autoUpdater.on('update-available', (info) => {
  dialog.showMessageBox({ // otevře okno s dotazem ke stažení aktualizace
    type: 'info',
    title: 'Update available',
    message: `A new version of ${app.getName()} is available. Do you want
to download and install it now?`,
    buttons: ['Yes', 'No']
  }).then(box=>{  // když uživatel klikne na yes provede se stažení
    if (box.response === 0) {
       autoUpdater.downloadUpdate();
    }
  });
});

// Informace o stažení aktualizace a potvrzení aktualizace
autoUpdater.on('update-downloaded', (info) => {
  dialog.showMessageBox({ // otevře okno s dotazem k instalaci aktualizace
    type: 'info',
    title: 'Update ready',
    message: 'A new version of the app is ready. Quit and install now?',
    buttons: ['Yes', 'No']
  }).then(box=>{. // když uživatel klikne na yes provede se aktualizace
    if (box.response === 0) {
      setImmediate(() => {
        app.removeAllListeners("window-all-closed")
        if (mainWindow != null) {
          mainWindow.close()
        }
        autoUpdater.quitAndInstall(false)
      })
    }
  });
});
```

Kód 37 Nastavení automatického stahování aktualizací.

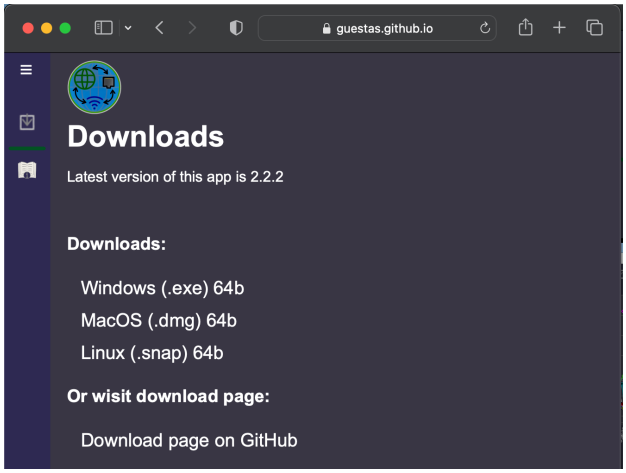
3.10 Manuál k aplikaci a web pro stažení aplikace

Manuál k aplikaci jsem umístil do samotné aplikace jako poslední položku v bočním menu (obr. 43). Stejný manuál se nachází i na webové stránce vytvořené pro aplikaci, kde slouží i jako prezentace aplikace bez nutnosti instalace podle [27]. Manuál jsem tvořil pomocí snímků oken v aplikaci a ke každé aktivní části jsem vložil číslo v rámečku. Pod obrázkem se nachází text s popisem prvků podle těchto čísel.



Obr. 43 Manuál v aplikaci.

Stránka aplikace je tvořená ve stejném stylu jako samotná aplikace. Je zde rozbalovací menu, kde jsou dvě položky *downloads* a *manual*. Na (obr. 44) je znázorněna stránka pro stahování aplikace pro všechny hlavní OS v prohlížeči. Stránka s manuálem je totožná s manuálem v aplikaci.



Obr. 44 Stránka pro stažení aplikace.

4. Závěr

Cílem této práce bylo vytvořit desktopovou aplikaci, která umožní uživatelům snadný a zabezpečený přístup k datům uloženým na serveru, kde je spuštěná testovací aplikace tvořená odděleně mimo mou práci. Mnou vytvořená aplikace je multiplatformní, tedy použitelná na OS Windows, macOS, Linux a umožnuje skrze testovací aplikaci na serveru sledování dostupnosti vzdálených serverů (webových serverů, datových serverů atp.). Aplikace umožňuje provádění administrativních úkonů spojených s daty v testovací aplikaci na serveru, jako je úprava, filtrování, mazání dat pomocí přehledné tabulky, dále zobrazování v grafech, a zobrazení souřadnic v mapě. Důraz jsem kladl na kvalitu, spolehlivost aplikace a na snadné intuitivní ovládání pro uživatele. Před vlastní tvorbou aplikace jsem provedl studie a analýzu tvorby aplikací pro různé operační systémy. Zvolil jsem framework Electron umožňující tvorbu multiplatformních desktopových aplikací, který dokáže využívat knihoven pro JavaScript. Navázal jsem zabezpečenou komunikaci se serverem, na kterém je spuštěna testovací aplikace pomocí TLS zabezpečení a HTTP requestů. Data z testování jsem se rozhodl reprezentovat v různých typech grafů. Aplikace zároveň umožňuje zobrazení uživatelem zadaných souřadnic v mapě. Grafy jsem vytvořil celkem čtyři – lineární graf znázorňující odezvy testovaných serverů, sloupcový horizontální graf shrnující počty úspěšných a neúspěšných spojení, koláčový graf průměrné odezvy testovaných serverů a prstencový graf pro srovnání počtu provedených testů serverů. Po dokončení tvorby aplikace jsem sestavil instalační soubory (.exe, .dmg a .snap) a distribuoval je prostřednictvím GitHubu. Webová stránka byla vytvořena pro reprezentaci aplikace, poskytnutí odkazů na jednotlivé instalační soubory a distribuci manuálu, který jsem k aplikaci vytvořil. Aplikace se serverovou testovací částí umožňuje testování dostupnosti zařízení v domácí síti, jako například domácího souborového serveru a jeho odezvy v čase, což může vést k vyhodnocení funkčnosti a případnému vylepšení tohoto serveru. V globálnějším měřítku mohou správci webových stránek pomocí aplikace snadno kontrolovat dostupnost svých serverů a sledovat jejich výkonnost v čase. Tímto způsobem je možné testovat více serverů najednou. Budoucí vylepšení aplikace je závislé na úpravách serverové testovací aplikace, ale jeví se zde vhodná tzv. triangulace, kdy by si uživatel serverovou testovací aplikaci spustil na třech různých počítačích nebo serverech a testoval by dostupnost svého webu ze tří různých míst, případně i z jiných kontinentů,

což by umožnilo zpřesnit měření a testování webu z více lokalit. Tyto tři testovací servery by bylo možné spravovat pomocí jednoho uživatelského účtu v této desktopové aplikaci. Veškeré vytyčené cíle této práce byly splněny.

Stránka k aplikaci:

Seznam zkratek a vysvětlení pojmů

CSS – Cascading Style Sheet – Kaskádový list stylů

GUI – Graphic User Interface – grafické uživatelské rozhraní

GPL – General Public License – generální veřejná licence

HTTP – HyperText Transfer Protocol – hypertextový přenosový protokol

HTML – Hyper Text Markup Language – značkovací jazyk pro webové stránky

HW – Hardware – označení pro fyzické komponenty počítače

ID – Identifier – Identifikátor

IPC – Inter Process Communication – meziprocesová komunikace

MIT – Massachusetts Institute of Technology – Massachusettský technologický institut

Open-source – otevřený zdroj – Svoboda ve využití, úpravě a distribuci

OS – Operating systém – operační systém počítače

Ping – Čas do odezvy serveru

Platforma – Základna v podobě operačního systému (Windows, Linux a macOS)

SSL – Secure Sockets Layer je protokol zajišťující šifrovanou komunikaci

SW – Software – označení pro programy nefyzické prvky počítače

TLS – Transport Layer Security – Transportní vrstva zabezpečení

URL – Uniform Resource Locator – Jednotný adresní ukazatel zdrojů

Seznam obrázků

Seznam tabulek

Seznam kódů a příkazů v příkazovém řádku

Bibliografie

[1] C. Nederkoorn, „Top 10 Python GUI Frameworks Compared,“ 24 4 2023. [Online]. Dostupné z: https://www.activestate.com/blog/top-10-python-gui-frameworks-compared/.

[2] W. O. Galitz, The Essential Guide to User Interface Design An Introduction to GUI Design Principles and Techniques, Indianapolis: Wiley Publishing, Inc., 2007, ISBN: 978-0-470-05342-3.

[3] D. Meador, „Graphical User Interface (GUI),“ 22 6 2020. [Online]. Dostupné z: https://www.tutorialspoint.com/graphical-user-interface-gui.

[4] E. Vojtěch, Vývoj aplikace s využitím cross-platform frameworku, Pardubice, 2018.

[5] M. Latiyan, „What languages have been used to write Windows, Mac OS and Linux OS?,“ 12 7 2022. [Online]. Dostupné z: https://www.tutorialspoint.com/what-languages-have-been-used-to-write-windows-mac-os-and-linux-os.

[6] B. Koďousková, „CO JE WEBOVÁ A DESKTOPOVÁ APLIKACE A JAKÝ JE MEZI NIMI ROZDÍL?,“ 17 11 2021. [Online]. Dostupné z: https://www.rascasone.com/cs/blog/desktop-web-aplikace.

[7] D. Phillips, Creating Apps in Kivy, Sebastopol: O’Reilly Media, 2014, ISBN: 978-1-491-94667-1.

[8] Kivy, „Architectural Overview,“ 2023. [Online]. Dostupné z: https://kivy.org/doc/stable/guide/architecture.html.

[9] S. Downes, „LEARNING ELECTRON - PART 1,“ 1 13 2019. [Online]. Dostupné z: https://halfanhour.blogspot.com/2019/01/learning-electron-part-1.html.

[10] C. Castillo, „JavaFX Architecture,“ 4 2013. [Online]. Dostupné z: https://docs.oracle.com/javafx/2/architecture/jfxpub-architecture.htm.

[11] N. Hilderbrandt, J. Gordon, C. Castillo a J. Potts, „JavaFX,“ 8 2014. [Online]. Dostupné z: https://docs.oracle.com/javase/8/javafx/JFXST.pdf.

[12] M. Fibor, „4 Best frameworks for cross-platform desktop app development,“ 26 1 2022. [Online]. Dostupné z: https://scythe-studio.com/en/blog/4-best-frameworks-for-cross-platform-desktop-app-development.

[13] Qt Company, „Model/View Programming,“ 2023. [Online]. Dostupné z: https://doc.qt.io/qt-6/model-view-programming.html.

[14] J. Bocklage.-Ryannel, C. Lqorquet a J. Thelin, „Qt 6 Book, “ Qt Company, 2022, [Online]. Dostupné z: https://www.qt.io/product/qt6/qml-book/ch01-meetqt-meet-qt.

[15] OpenJS Foundation, „What is Electron?, “ 2021. [Online]. Dostupné z: https://www.electronjs.org/docs/latest/.

[16] G. Pinto, „Why the MIT licensed is much more used than GPL-3?,“ 21 4 2020. [Online]. Dostupné z: https://gustavopinto.medium.com/why-the-mit-license-is-much-more-used-then-gpl-3-2b3fa4271d6b.

[17] I. Ristić, BULLETPROOF SSL AND TLS, Londýn: Feisty Duck, 2015, ISBN: 978-1-907117-04-6.

[18] U. Hiwarale, „A brief overview of the TCP/IP model, SSL/TLS/HTTPS protocols and SSL certificates,“ 1 2 2020. [Online]. Dostupné z: https://medium.com/jspoint/a-brief-overview-of-the-tcp-ip-model-ssl-tls-https-protocols-and-ssl-certificates-d5a6269fe29e.

[19] P. B. Jensen, Cross-Platform Desktop Applications Using Electron and NW.js, NY: Manning Publications Co., 2017, ISBN: 978-1-61729-284-2.

[20] J. N. Robbins, LEARNING WEB DESIGN, Sebastopol: O’Reilly Media, Inc., 2018, ISBN: 978-1-491-96020-2.

[21] S. KINNEY, Electron in Action, New York: Manning Publications Co., 2019, ISBN: 978-1-61729-414-3.

[22] W. Mwaura, „Making HTTP requests with Axios,“ 19 5 2022. [Online]. Dostupné z: https://circleci.com/blog/making-http-requests-with-axios/#c-consent-modal.

[23] G. Mariani, „Adding trusted CA to node client with axios,“ 16 5 2019. [Online]. Dostupné z: https://giacomo-mariani.medium.com/adding-trusted-ca-to-node-client-with-axios-2792024bca4.

[24] J. Olawanle, „Guide to Creating Charts in JavaScript With Chart.js,“ 6 4 2023. [Online]. Dostupné z: https://stackabuse.com/guide-to-creating-charts-in-javascript-with-chartjs/.

[25] V. Agafonkin, „Leaflet API reference,“ 2010-2023. [Online]. Dostupné z: https://leafletjs.com/reference.html.

[26] Akash, „A complete guide to packaging your Electron app,“ 30 12 2016. [Online]. Dostupné z: https://medium.com/how-to-electron/a-complete-guide-to-packaging-your-electron-app-1bdc717d739f.

[27] G. Myrianthous, „GitHub Pages: An Introductory Tutorial,“ 15 11 2022. [Online]. Dostupné z: https://builtin.com/software-engineering-perspectives/github-pages.

Přílohy

[1]. CD disk