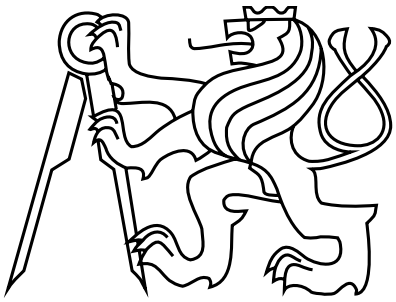


ČESKÉ
VYSOKÉ
UČENÍ
TECHNICKÉ
V PRAZE

Fakulta Elektrotechnická
Katedra radioelektroniky

IOT Časovač

Implementace časovače a budíku na minipočítači

Bakalářská práce

Autor:
Jakub Kadlec

Vedoucí práce:
doc. Ing., Ph.D.
Stanislav Vítek

Studijní program: Elektronika a komunikace

Praha 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kadlec** Jméno: **Jakub** Osobní číslo: **484282**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

IoT časovač

Název bakalářské práce anglicky:

IoT Timer

Pokyny pro vypracování:

Navrhněte a implementujte IoT časovač (budík). Zařízení obsahuje vlastní databázi událostí, kterou je možné plnit lokálně ze zařízení, nebo pomocí vhodné služby s veřejným API (např. Google Calendar). Zařízení může plnit funkci klasického budíku (např. přehraje zvukovou stopu v zadaný čas), komunikovat s externími službami, nebo ovládat další zařízení, např. prostřednictvím sítě ZigBee. V rámci práce diskutujte možnosti zabezpečení komunikace s externímu službami.

Seznam doporučené literatury:

[1] GAY, Warren. Raspberry Pi hardware reference. Apress, 2014.
[2] HU, Fei. Security and privacy in Internet of things (IoTs): Models, Algorithms, and Implementations. CRC Press, 2016.
[3] SERPANOS, Dimitrios; WOLF, Marilyn. Internet-of-things (IoT) systems: architectures, algorithms, methodologies. Springer, 2017.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Stanislav Vítek, Ph.D. katedra radioelektroniky FEL

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

Datum zadání bakalářské práce: **02.02.2022** Termín odevzdání bakalářské práce: **15.08.2022**

Platnost zadání bakalářské práce: **30.09.2023**

doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) práce	doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. Datum převzetí zadání

Podpis studenta

Prohlašuji, že jsem předloženou bakalářskou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s „Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.“

V Praze dne 15. 8. 2022

Abstrakt

Práce se zabývá návrhem a implementací IoT časovače a budíku propojeného s webovou službou Google Kalendář. Jako platforma byl zvolen jednodeskový počítač Raspberri Pi v kombinaci s dotykovým displejem a příslušenstvím potřebným pro jeho provoz. Obsahem práce je výběr vhodné platformy, vývojových nástrojů, design grafického uživatelského prostředí a popis vnitřního fungování některých částí kódu, který funkce zprostředkovává.

Abstract

This work is focused on the implementation of IoT timer and alarm clock with the ability to communicate with Google Calendar. The platform used in this project is Raspberri Pi in combination with a touchscreen and other necessary accessories that are needed to ensure it's operation. In this text I chose a suitable platform and development enviroment. Apart from that I designed graphical user and described the development process with explanation of some of the code.

Obsah

Kapitola 1

Úvod

Cílem tohoto projektu je vytvořit IoT časovač a budík. Zatímco na splnění vytyčeného cíle je možné použít mobilní telefon, někteří uživatelé by pro uvedenou aplikaci mohli preferovat použití dedikovaného přístroje, který bude vždy na jednom místě. Dalším přínosem je, že aplikace umožňuje měřit časové intervaly a v kombinaci s upozorněními na nadcházející události by se dala využít pro zadávání práce ve firemním prostředí, přičemž změřená data by pak mohla sloužit k vyhodnocení různých fází výroby.

Snahou je vytvořit samostatné zařízení, které se bude skládat z jednodeskového počítače, reproduktoru a dotykového displeje. To zvládne bez dalších periferií plnit funkci budíku a časovače s offline databází událostí. Po připojení k internetu bude umožňovat synchronizaci s Google kalendářem.

Tato práce se zabývá popisem softwarových nástrojů, které umožní komunikaci s Google kalendářem a diskutuje různé hardwarové platformy potenciálně použitelné pro projekt. Kromě toho se bude věnuje návrhu uživatelského prostředí samotné aplikace a jeho ovládání pomocí skriptovacího jazyka Python

Kapitola 2

Teoretický rozbor

2.1 Internet věcí (IoT)

Zdroje [1] a [2] se shodují na tom, že pojem internetu věcí není přesně definovaný. Lze ho podle nich chápat jako nástroj pro připojení všeho kolem nás ke globální síti – internetu. Přitom se očekává „chytré“ chování takových zařízení a nějaká přidaná hodnota, zabezpečení a autonomita zmiňovaného spojení. Do takto definované kategorie můžeme zařadit průmyslové stroje, automobily, mobilní telefony, ale i některé mnohem větší a komplexnější konstrukce, jako jsou domy, města a další.

Článek [3] považuje internet věcí jako jakousi druhou generaci internetu. Ta první podle něj byla určena pro pohyb dat vytvořených lidmi. Proti tomu za internet věcí pokládá generování a dopravu dat mezi zařízeními připojenými ke globální síti. Jako nejpřesnější definicí internetu věcí pak uvádí: *„Otevřená a všeobecná síť chytrých objektů, které mají schopnost se samostatně organizovat, sdílet informace, data, zdroje, reagovat a jednat tváří v tvář situacím a změnám v prostředí.“* Značné množství pozornosti, které se v poslední době IoT dostává, je podle něj způsobené skutečností, že internet věcí umožňuje komunikaci člověk-člověk, člověk-zařízení i zařízení-zařízení, což otevírá veliké možnosti.

Naproti tomu autoři [2] definici internetu věcí dělí na základě typu zařízení:

- Fyzická zařízení připojená k internetu, přičemž nemusí přímo využívat IP protokolu.
- Senzorové sítě měřící v reálném čase.

- Dynamické sítě zařízení složené z embedded[1] systémů.

Citovaná kniha považuje za praktické využívat pojem „systém internetu věcí“. V tomto případě se jedná o systém určený k řešení zadaných úloh, spíše než síť všech globálně propojených zařízení. Přitom ale systém uvažuje dynamiku fyzických systémů – výsledek, který od takového systému očekáváme jsou data v časové posloupnosti.

Zásadním parametrem takových systémů může být cena – pro sběr dat může stačit jednoduchý mikrokontroler a senzor v kombinaci s komunikačním modulem, kdy cena nebude vysoká nebo se může jednat o soustavu kvalitních čidel s vysokým nárokem na výpočetní výkon a infrastrukturu, kdy se cena může vyšplhat poměrně vysoko. Pro některé aplikace může být zásadní minimalizace spotřeby energie – prvky internetu věcí napájené bateriemi musí dobře hospodařit s energií. Kromě toho je důležité, aby se naměřená data spolehlivě a bezpečně dostala ke správci systému.

2.1.1 Historie

O předchůdci IoT se zmiňoval roku 1991 Mark Weiser v [4] a koncept představil pod pojmem *„Ubiquitous Computing“*. Autor se zabýval vytvořením prostředí spotřebičů a zařízení – vypínačů, termostatů, zesilovačů, trub a dalších, které by byly propojeny do všudypřítomné sítě.

Mark Weiser kritizoval osobní počítače a tvrdil, že byly příliš veliké a nešikovné na to aby byly jedinými výpočetními nástroji, které lidstvo používá. Ihned ale přichází s řešením. Jeho vize byla aby každá místnost obsahovala desítky až stovky mobilních výpočetních přístrojů různých velikostí a uživatelé by si podle potřeby vybrali, které použijí. Mark Weiser a jeho kolegové vytvořili takový pokoj, který sloužil k demonstraci tohoto konceptu.

Vstup demonstrační místnosti byl stejně jako příchozí vybaven infračerveným přístupovým zařízením. Toto bylo zhruba velikosti jmenovky a sloužilo k ověření totožnosti, udělení přístupu a k zobrazení menšího množství informací na malém integrovaném displeji. Pokud byla osoba oprávněna vstupu do místnosti, odemkly

[1] Výpočetní systémy dedikované pro nějakou konkrétní úlohu.

se jí dveře a místnost ji jmenovitě přivítala a automaticky jí byly do místnosti přesměrovány hovory. Tato přístupová zařízení v představě Marka Weisera hrála roli zástupce fyzicky menších osobních počítačů.

Větším prvkem bylo zařízení, které neslo označení *„tab“*, které se podobalo dnešnímu grafickému tabletu. Bylo to relativně tenké mobilní zařízení ovládané rukou nebo speciálním perem. Fyzicky největším zástupcem konceptu *„Ubiquitous Computing“* byla tabule, která mohla být ovládana klávesnicí nebo dotykem.

Samotný termín *„Internet of Things“* (internet věcí) je poprvé v tomto kontextu použit v roce 1999 Kevinem Ashtonem [5].

2.1.2 Využití

Systémy internetu věcí jsou využívané pro plnění širokého spektra aplikací. Jsou přítomné v těch nejjednodušších zařízeních, ale i ve velmi komplexních systémech.

Nalezneme je v průmyslových systémech, kde mohou monitorovat výrobní proces, samotný výrobek nebo stav nástrojů, které jsou používány. To může pomoci s evidencí výrobních procesů a jejich zkvalitněním. Moudré využití takové technologie vede ke zlevnění a zrychlení výrobního procesu. Zároveň umožňuje předvídat selhání nástroje na základě dat, která jsou o něm nasbíraná.

Za demonstraci úspěšné implementace IoT v průmyslu se dá považovat program „Smart Fields“ (inteligentní pole) společnosti Shell. Ten se rozběhl roku 2002, kdy se na 50 ropných polích začaly v reálném čase monitorovat těžební jámy, stroje používané k těžbě ropy a potrubní systémy. Ve světle nasbíraných dat bylo možné optimalizovat výrobní procesy a zrychlovat tak těžbu. V letech 2008-2009 byla vypracovaná studie, která určila, že za uplynulých 7 let program nepřímo přinesl hodnotu 5 miliard amerických dolarů [6].

Systémy internetu věcí se používají také v chytrých budovách a domácnostech, kde se mohou podílet na lokalizaci osob, mohou monitorovat stav budovy (teplotu, vlhkost, koncentraci oxidu uhličitého ve vzduchu místnosti,...), dokáží autonomně ovládat klimatizaci, topení a osvětlení, čímž mohou napomoci snižování spotřeby energií a zvyšování kvality prostředí v budově. Kromě toho se mohou starat o odemykání a zamykání, chytré rozsvěcení, zalévání květin a dalších věcí. Navíc některé

možnosti chytrých domácností mohou využít chytrá města – monitorováním chodců a dopravní situace má potenciál pomoci městu zvýšit plynulosti dopravy.

Zdravotnické systémy mohou tuto technologii použít ke sledování pacienta na domácím lůžku nebo v nemocnici. To pomáhá zlevnit provozní náklady – práce mikrokontroleru je výrazně méně nákladná než práce personálu nemocnice. Kromě toho dochází ke zkvalitnění péče tím, že lékaři mají neustále k dispozici čerstvá data o životních funkcích pacienta.

2.2 Požadavky na projekt

Tato práce do jisté míry rozšiřuje již zpracovanou [7]. Více se ale zabývám popisem rozhraní mezi aplikací a Google kalendářem.

Cílem projektu je vytvořit IoT časovač s následujícími vlastnostmi:

- Zařízení umí pracovat i bez připojení k internetu.
- Zařízení má vlastní databázi událostí.
- Zařízení umí synchronizovat události se službou s veřejným API (Google Kalendář).
- Zařízení může plnit funkci budíku.
- Zařízení může komunikovat s externími službami nebo ovládat další zařízení.

2.3 Python

Python existuje už více než tři desetiletí a je značně populárním skriptovacím jazykem. Dlouhodobě se na žebříčcích popularity pohybuje v první desítce. Používají ho takoví techničtí giganti jako Google, Netflix, Dropbox a další. Projekty psané v Pythonu se pohybují v rozsahu délky od několika řádků kódu až ke komplexním programům, které obsahují více než několik milionů řádků. Současnou verzí Pythonu je Python 3.x a proto se v následujících sekcích budu zabývat popisem této verze [8].

2.3.1 Obecný přehled

Python je dynamický jazyk (skriptovací jazyk). To znamená, že v porovnání se statickými programovacími jazyky (například C++ nebo s Java) dovoluje změnu datového typu proměnných. Skriptovací jazyky se dále liší tím, že jsou interpretovány za běhu programu, narozdíl od statických, které musí být před během zkompilovány. Tato flexibilita ale stojí nějaký výpočetní výkon procesoru, který negativně ovlivní použitelný výpočetní výkon stroje.

2.3.2 Základní proměnné typy

V Pythonu jsou veškeré datové typy implementovány jako objekty. Najdeme zde základní proměnné typy jako jsou `boolean`, `int`, `float` a `string`, ale i mnohem sofistikovanější objekty, které obstarávají komplexní operace.

2.3.3 Práce s časy a daty

Pro práci s daty a s časovými údaji nabízí Python modul `datetime`. V něm jsou zadefinované čtyři základní datové typy.

- Typ `date`, který dokáže uložit informace o roku, měsíci a dni.
- Typ `time`, jenž ukládá informace o hodinách, minutách a sekundách.
- Univerzálnější typ `datetime` kombinuje funkcionality předchozích dvou.
- Nakonec typ `timedelta`, který je určený pro ukládání informací o časových a datových intervalech.

Objekt `datetime` nabízí mimo jiné i metodu pro zjišťování aktuálního data a času. Mimo to implementuje metody pro formátování uloženého času a data do textového řetězce nebo dokáže extrahovat datum a čas z textového řetězce v předdefinovaném formátu. Názvy zmiňovaných metod jsou `strftime` respektive `strptime`.

2.4 Grafické uživatelské prostředí

Pro to, aby byla aplikace přívětivá k uživatelům, je vhodné vytvořit grafické uživatelské prostředí. To by mělo uživateli umožnit intuitivně interagovat s programem bez hlubší znalosti vnitřního fungování. Tvorbu uživatelského prostředí mohou usnadnit knihovny s definovanými grafickými ovládacími prvky (widgety). Ty lze používat a modifikovat pro potřeby aplikace. Na následujících řádcích se zaměřím na několik takových knihoven, které je možné ovládat ze skriptu psaného v Pythonu. Zároveň jsem se omezil na knihovny a designové nástroje, které nejsou zpoplatněné a byly k dispozici pro implementaci mého zadání.

2.4.1 Tk

Pozoruhodnou knihovnou, která nabízí grafické ovládací prvky je Tk. Ta byla už od roku 1988 vyvíjena pro vysokoúrovňové, dynamické programovací jazyky, konkrétně pro Tcl [9]. Funguje na různých platforách jako jsou Windows a na platformách na bázi Unixu jako Mac OS, Ubuntu, Fedora a na dalších. To zjednodušuje vývoj softwaru pro různé platformy současně. Tk je poměrně vysokoúrovňová, což má za důsledek, že spousta detailů, které by programátor musel ošetřit kódem ve své aplikaci, je vyřešena už v rámci knihovny.

Použití Tk z pythonového skriptu umožňuje modul Tkinter. V kombinaci s ním se knihovna stala velmi oblíbenou pro aplikace psané v Pythonu. Bohužel v důsledku stáří knihovny došlo k tomu, že existuje veliké množství zastaralých materiálů, které znesnadňují hledání aktuálních informací.

2.4.2 Kivy

Dalším představitelem takové knihovny je Kivy. Ta je komunitně vyvíjená a snaží se uspokojit potřeby moderního programátora. Důraz je kladen na vícedotykové ovládání [10], rychlý chod uživatelského prostředí aplikace a na flexibilitu. Mimo to podporuje i méně standardní metody ovládání, jako jsou dotyková pera.

Mezi podporované platformy patří nejen tradiční operační systémy pro osobní

počítače, ale mobilní operačních systémy Android a iOS.

2.4.3 wxWidgets

Jako ostatní tak i wxWidgets je multiplatformní knihovna napsaná v jazyce C++, která umožňuje vytvořit uživatelské prostředí, které bude bez modifikací fungovat na Windows, Mac OS, Ubuntu a dalších [11]. Její snahou je navrhnout takové grafické uživatelské prostředí, které se bude co nejlépe hodit do prostředí použitého operačního systému.

Pro využití ze skriptu psaném v Pythonu je potřeba nainstalovat a použít modul wxPython, který nabízí nástroje potřebné k obsluze grafického uživatelského prostředí, jež používá prvky z knihovny wxWidgets.

2.4.4 Qt

Qt je dobře zavedená knihovna napsaná v C++. Pro potřeby tohoto projektu je zásadní, že nabízí moduly pro Python, které zprostředkovávají její ovládání. Dobře známé a hojně užívané jsou PyQt a PySide. Na cestě za multiplatformním softwarem zachází Qt daleko. Definuje vlastní multimediální, databázové, síťové a jádrové funkce, aby se mohla zaručit za správný chod na různých platformách. Mezi podporované platformy patří mimo klasické desktopové (Windows, Mac OS, operační systémy s linuxovým jádrem – Ubuntu, Red Hat,...) a mobilní operační systémy (iOS, Android) například WatchOS, tvOS a podporu naleznou také vývojáři grafických aplikací pro mikrokontrolery ST. Vyčerpávající seznam všech podporovaných platforem je k nalezení na [12].

2.5 Knihovna Qt

2.5.1 Designová strategie

Knihovna Qt s modulem PyQt má mnoho předdefinovaných grafických prvků, které dovedou významným způsobem mohou ulehčit implementaci požadovaných funkcí.

V této sekci bych se rád zabýval krátkým popisem specifik knihovny, ale i konkrétními grafickymi prvky.

Důležitým mechanizmem při interakci s knihovnou Qt jsou signály a sloty [13]. Základní myšlenkou signálů a slotů je vzájemná komunikace mezi objekty. Pomocí slotů a signálů je možné na základě nějaké akce upozornit související objekty na probíhající změny. Signál je vyslán vždy, když dojde k definovanému internímu ději. Sloty jsou funkce, které se spustí ve chvíli, kdy se aktivuje signál, který je do nich napojen. Grafické prvky knihovny mají předdefinované signály a sloty pro vybrané události. Pokud je ale potřeba, lze je zadefinovat podle potřeby i jinak.

2.5.2 Grafické prvky

2.5.3 QStackedWidget

Pokud je potřeba na jednom místě grafického uživatelského prostředí zobrazit v různých časech různé prvky, pak je příhodný `QStackedWidget`. Ten dovolí narovnat různé prvky na jedno místo do vrstev a potom pomocí funkce `setCurrentWidget`, do které se jako argument vloží vrstva, na kterou se přepíná [14]. Widget zobrazuje vždy právě jednu vrstvu, na kterou přepnul.

2.5.4 QLineEdit

Editaci řádku textu uživatelem umožňuje prvek QLineEdit [15]. V něm může uživatel psát a mazat text. Zároveň prvek umožňuje validaci vstupu a dovoluje omezit vysílání některých signálů jenom na případ, kdy vstupní text splňuje stanovená kritéria. Kromě toho umožňuje zobrazit text před tím, než uživatel začne psát. To lze využít kupříkladu k drobné navigaci po aplikaci bez použití přídavných informačních prvků. QLineEdit dokáže signalizovat při změně textu nebo při dokončení editace a při dalších událostech.

2.5.5 QListWidget

Někdy je potřeba zobrazit jednoduchý seznam položek. Tuto funkci v Qt zprostředkovává QListWidget [16]. Ten dokáže ve sloupečku zobrazit řádky textu v kombinaci s ikonkou. Kromě toho umožňuje interakci s jednotlivými položkami, změnu jejich pořadí, přidávání a odebírání položek a další.

2.5.6 QCalendarWidget

Výběr data je poměrně komplexní úkol – uživatel musí vybrat rok, měsíc a datum, přičemž je vhodné indikovat o jaký den v týdnu se jedná, neboť to může ulehčit orientaci. S daným úkolem se utkává QCalendarWidget [17], který v přehledné formě zobrazí kalendář aktuálního měsíce nebo dovolí přepnout na jiný měsíc, případně rok. Tato data je pak možné přečíst a použít v praktické formě. Widget posílá signály když se změní výběr dat, když je aktivován, po kliknutí nebo po změně stránky.

2.5.7 QPushButton

Řadu akcí je výhodné ovládat pomocí pomocí tlačítek. Ty jsou v Qt implementované třídou QPushButton [18]. Kromě klasické funkce lze přepnout tlačítka do módu zaškrtávacího políčka. To v praxi znamená, že po kliknutí tlačítko zůstane zmáčknuté až do doby, kdy je na něj znovu kliknuto. Tlačítka mohou v různých stavech vysílají různé signály. Mezi události, kdy tlačítko vyšle signál patří kliknutí – zmáčknutí myší a následné puštění, zmáčknutí – první moment, kdy uživatel stiskne tlačítko s puštění – chvíle kdy po držení ho uživatel pustí. Podobně jako QListWidget dokáží tlačítka zobrazit malou ikonku.

2.5.8 QLabel

Poměrně jednoduchým prvkem, který neumožňuje žádnou interaktivitu s uživatelem QLabel [19]. Tento Widget dokáže zobrazovat obrázky nebo text, nabízí také text ze skriptu měnit, ale už nedovoluje změny provádět přímo na popud uživatelem.

Kapitola 3

Zařízení

Zadání by bylo pravděpodobně možné implementovat na poměrně jednoduchá zařízení, ale důsledekem takového přístupu by byla ztráta flexibility – neměl bych k dispozici veliký výpočetní výkon a proto bych musel opatrněji přístupovat ke zdrojům. Jednodeskový počítač vypadal jako vhodný kompromis mezi výkonem a kompaktností. Stačilo ho doplnit o obrazovku. Uživatelské rozhraní nebude výrazně složité a proto na jeho zobrazení postačí menší displej. Pojmem jednodeskový počítač myslím zařízení, skládající se z jedné desky s plošného spoje, na které jsou všechny komponenty potřebné k samostatnému provozu desky jako počítače a jeho připojení k periferiím.

Zařízením, které jsem měl pro vývoj aplikace k dispozici bylo Raspberry Pi 4B se sedmipalcovým barevným dotykovým displejem o rozlišení 800x480 pixelů, což je pro danou aplikaci vhodná kombinace. Pochopitelně se nejedná o jediný použitelný jednodeskový počítač pro takovou aplikaci. Existují mnohé alternativy a jejich porovnání je uvedené v 3.1.

Pro zvukový výstup jsem použil reproduktor s interním napájením a zesilovačem, zapojeným do audio výstupu Raspberry Pi.

3.1 Raspberry Pi

Základním stavebním kamenem projektu je jednodeskový počítač. Velmi dostupné a dobře dokumentované jsou produkty výrobce Raspberry Pi. Různé typy se liší velikostí operační paměti (RAM), počtem jader procesoru, periferními porty, cenou,

fyzickou velikostí desky a dalšími parametry. V tabulce 3.1 jsem se pokusil udělat výběr z [20] a ukázat alespoň několik parametrů různých počítačů a porovnat je mezi sebou.

Tabulka 3.1: Porovnání různých typů Raspberry Pi

Platforma	**4B**	**3B+**	**Zero**	**B**
SOC	4x1.8 GHz	4x1.8 GHz	1x1 GHz	1x700 MHz
Kapacita RAM [MB]	4096	1024	512	512
Přibližná cena [$]	55	35	5	35
Počet USB 3.0	2	0	0	0
Počet USB 2.0	2	4	3 (micro)	2
Počet HDMI	2 (micro)	1	1 (mini)	1
Ethernet Nx[Mbps]	1x1000	0	0	1x100

Kromě Raspberry Pi existují i alternativní jednodeskové počítače. Mezi ně by se daly zařadit například Orange Pi, Asus Tinker Board, Odroid, Banana Pi a další.

3.1.1 Operační systém

Přestože jsem v předchozí sekci psal o Raspberry Pi jako o počítači, [21] upozorňuje na to, že na Raspberri Pi nebude možné používat jakýkoliv operační systém. Zmiňované desky používají procesory s architekturou ARM, která není pro stolní počítače obvyklá. Proto jí musí být přizpůsoben i operační systém. Následuje výčet několika operačních systémů, z nichž jsem pro svůj projekt vybíral.

Raspberry Pi OS

Oficiální operační systém Raspberry Pi OS (dříve Raspbian) byl poprvé dostupný nedlouho po vypuštění prvních počítačů Raspberry Pi na trh, v roce 2012. Je poskytovaný a spravovaný přímo společností Raspberry Pi. Jedná se o operační systém založený na distribuci linuxového Debianu a dostupný ve dvouch diametrálně odlišných verzích:

- Raspberry Pi OS Lite – minimální verze bez grafického uživatelského rozhraní, která je šetrnější k využívání prostředků počítače,
- Raspberry Pi OS PIXEL – verze pro stolní počítač, která disponuje i grafickým uživatelským prostředím a nástroji pro programování

Kromě předinstalovaných programů může uživatel pochopitelně instalovat i další. Databáze dostupných programů je kvůli jiné architektuře procesoru poněkud menší než bychom mohli očekávat u verze pro stolní počítače, ale mnoho aplikací na které bychom mohli být zvyklí z počítačů s arhitekturou x86 máme k dispozici i zde.

Podle [22] je tento operační systém považován za spolehlivý a kompatibilní s hardwarem počítačů Raspberry Pi. Zároveň je dostatečně flexibilní pro různé projekty.

Windows IoT Core

Tento operační systém je produktem společnosti Microsoft. Jak se můžeme dočíst na [23], jedná se o menší verzi Windows 10, která postrádá některé drivery. Ty je v případě nutnosti možné zkompilovat ze zdrojového kódu. To ale přidává na komplexitě přípravy projektu. Navíc ze sortimentu Raspberry je dostupný pouze pro Pi 2 a 3.

Retro Pie

Retro Pie je další zástupce operačního systému běžícím na linuxovém jádře. Podobně jako Raspberry Pi OS se jedná o nadstavbu na Debian. Jeho hlavním zaměřením je udělat z Raspberri Pi počítač, na kterém poběží retro hry. Toho se snaží dosáhnout výběrem předinstalovaných programů, aby měl uživatel co nejméně práce s jejich manuální instalací.

MicroPython

MicroPython je (ne úplně kompletní) implementace programovacího jazyka Python. Nejnovější verze MicroPythonu je 3.5. Ten je optimalizovaný pro běh přímo na hardwaru. Narozdíl od předešlých možností se tedy nejedná ani o operační systém

v pravém slova smyslu. Využití nachází především na méně výkonných deskách, jako například Raspberry Pi Pico.

Kapitola 4

Serverová část

4.1 API

Zkratkou API[1] se označuje rozhraní, které aplikaci zprostředkuje komunikaci s jinou aplikací nebo službou [24]. To umožňuje spolupráci bez hlubších znalostí vnitřního fungování druhé strany. V dokumentaci je potom uvedený soubor pravidel, který definuje jakými mechanizmy je možné k prostředkům aplikace přistupovat.

Existují různé druhy API přizpůsobené pro různé aplikace. Může se i stát, že k jedné službě bude možné přistupovat pomocí různých API. Kterou API vývojář vybere už závisí jen na něm, případně na tom jaké funkce jednotlivá API nabízí.

4.1.1 REST API

Přívlastkem REST[2] je označená architektura API navržená v roce 2000 Royem Fieldingem v [25].

Jedná se o popis spojení mezi dvěma entitami připojenými k internetu – společné síti. Jednotlivé zdroje, kterými disponuje server, ať už se jedná o webovou aplikaci, databázi, objekt nebo například algoritmus, jsou identifikovatelné textovým řetězcem [26]. Klient použije dotazovací metody poskytované internetovým HTTP[3] k tomu, aby přistupoval, modifikoval, mazal a vytvářel zdroje na serveru. Axiomy REST API jsou [27]:

[1] **A**pplication **P**rogramming **I**nterface
[2] **RE**presentational **S**tate **T**ransfer
[3] **H**ypertext **T**ransfer **P**rotocol

- Všechny požadavky na nějaký konkrétní zdroj by měly vypadat stejně.
- Server a klient by na sobě měli být úplně nezávislí – nekomunikují spolu jinak než právě přes REST API.
- Každá odpověď serveru musí být samonosná – nepředpokládá předchozí komunikaci s klientem. Znamená to, že požadavek na server musí nést všechny informace nutné pro vykonání požadavku.
- Platnost dat by měla být sdělena klientovi serverem. Pokud jsou data konstantního charakteru, měl by klient data uložit a pokud je znova bude potřebovat, obejde se bez požadavku na server.
- Každá aplikace která používá REST API musí počítat s tím, že nekomunikuje přímo se zdrojovou aplikací. Mohou mezi nimi existovat prostředníci.
- Pokud server posílá nějakým způsobem zakódovaná data, měl by také poskytnout kód, který vede k dekódování a jejich reprezentaci.

4.1.2 RPC API

Další architekturou api je RPC[4] API [28]. Je to nejstarší architektura. Ta umožňuje klientovi spustit na serveru proceduru s příslušnými parametry. Výsledek volání je zpět odeslán klientovi. Tento princip má výhodu v tom, že rozsah dat, který je potřeba poslat je omezený na potřebná data – přesně specifikovaný v požadavku, což šetří množství dat putující mezi serverem a klientem.

4.1.3 SOAP API

Poslední architekturou, kterou zmíním je takzvaná SOAP API[5]. Funkcí se podobá REST API, ale je mezi nimi zásadní rozdíl – SOAP API je přímo striktně definovaný komunikační protokol.

[4] **R**emote **P**rocedure **C**all
[5] **S**imple **O**bject **A**ccess **P**rotokol

4.2 Google kalendář

Kalendář Google je služba, kterou zdarma poskytuje Google všem vlastníkům účtu Google. Slouží k záznamu plánovaných akcí, ale nabízí i možnosti sociální interakce a sdílení. Kalendář umožňuje přistupovat z webového rozhraní, mobilní aplikace, nebo pomocí webového API.

4.2.1 Struktura kalendáře

K profilu Google je přidružená služba kalendáře. Na tuto službu není vhodné pohlížet jako na klasický kalendář. Její možnosti jsou flexibilnější. Uživatel má možnost vytvořit a provozovat několik kalendářů, které jsou na sobě nezávislé. Při tvorbě je možné kalendář pojmenovat, přidat jeho popis a zvolit v jakém časovém pásmu se budou události odehrávat.

V kalendáři nalézáme 2 typy záznamů. Jedná se o události a úkoly. Události jsou komplexnější struktury – nabízí různé konfigurace. Uživatel si volí název, datum konání (v případě, že je událost vícedenní tak datový inteval) mimo to lze nastavit i sofistikované možnosti opakování události do budoucna. Není-li událost celodenní tak se nabízí možnost zvolit čas konání. Kromě těchto základních možností nabízí Google i možnost přizvat k události jiné uživatele, naplánovat videohovor přes službu Google Meet, uvést místo konání, popis události nebo přiložit k události nějaký soubor. Zároveň implementuje i možnosti nastavení upozornění před začátkem události.

Druhým typem záznamu jsou úkoly, které dovolují menší míru konfigurace. Lze pro ně nastavit čas a datum, opakování a popis.

4.2.2 API kalendáře

Použitá API je typu REST. Co to pro design mojí aplikace znamená jsem krátce popsal v kapitole 4.1. V této sekci se budu zabývat popisem konkrétní API – Google Calendar API, která umožňuje přístup k datům z Google kalendáře.

Poskytovaná API umožňuje poměrně flexibilní přístup ke kalendáři [29], který můžeme rozdělit do několika logických kategorií. Popisem těch nejdůležitějších pro

moji práci se budu dále zabývat.

Kontrola přístupu – z hlediska oprávnění

Všechny požadavky na tuto oblast API se provádějí na konkrétní kalendáře identifikované unikátním id. Tímto způsobem je možné ovládat přístupnost k danému kalendáři [30]. Pravidla mohou být nastavena pro konkrétní doménu, skupinu uživatelů, konkrétního uživatele nebo globálně. Pro vybraný z předchozích subjektů pak lze nastavovat z následující hierarchie pravomocí žádné oprávnění, oprávnění k zobrazení zda je vlastník kalendáře v daný čas k dispozici, oprávnění ke čtení, k modifikaci nebo k vlastnictví.

Seznam kalendářů

Seznamem kalendářů se myslí kolekce kalendářů, které si uživatel zvolil do vlastního seznamu kalendářů – toho, který se ve webovém rozhraní zobrazí na levé liště. V rámci tohoto přístupu můžeme přes API přistupovat k vlastnostem kalendářů [31], jako jsou přístupové oprávnění, barva, výchozí připomenutí, popis, id, výchozí časová zóna a další. Skrze tento přístup můžeme přidávat a odstraňovat existující kalendáře ze zobrazeného výběru pro konkrétního uživatele, aniž bychom modifikovali výchozí nastavení pro všechny kdo k danému kalendáři mají přístup. Tento přístup ale neumožňuje modifikovat výchozí parametry, jako jsou například popis, id, časová zóna a další.

Kalendáře

Kalendáře je kolekce všech kalendářů ke kterým má uživatelem přístup. Narozdíl od seznamu kalendářů se zabývá výchozími parametry kalendářů [32] spíše než nastaveními pro konkrétní uživatele. Díky tomu můžeme přistupovat a v mezích našeho oprávnění modifikovat parametry jako popis kalendáře, výchozí časovou zónu, titulek a další.

Události

Události jsou jedním ze základních prvků kalendáře. Jejich struktura a parametry jsou popsané v 4.2.1. Klíčovými prvky této struktury jsou začátek a konec. Google Kalendář je reprezentuje různými způsoby v závislosti na tom zda se jedná o celodenní událost nebo událost ohraničenou přesným časovým intervalem [33].

- Celodenní událost začne v nějaký den a jiný den končí. Blíže nespecifikuje čas začátku respektive konce. V tomto případě jsou časy reprezentované hodnotami `start.date` respektive `end.date`. Tyto jsou textové hodnoty ve formátu *„yyyy-mm-dd“* (například: *„2020-03-27“*).
- Pokud je ale událost přesně časově ohraničená, využívá se hodnot `start.dateTime` respektive `end.dateTime`. Tyto hodnoty jsou textové, kombinované z data a času události podle RFC3339. Za předpokladu, že událost nemá definovanou časovou zónu, musí tato hodnota nést také informaci o časové zóně.

Oba zmiňované typy událostí nelze kombinovat, tedy událost nemůže začínat jako celodenní, a potom končit v přesný čas. Analogicky není možné aby byl začátek přesně časově vymezen a konec byl celodenní.

4.2.3 Klientská knihovna pro Google kalendář

Pro zjednodušení práce s API Google Kalendáře jsou k dispozici knihovny, které zajišťují velikou část technických záležitostí a programátorovi ušetří zkoumání vnitřního fungování API. V době psaní práce Google poskytoval klientské knihovny ve vyspělejším stádiu vývoje pro 5 programovacích nebo skriptovacích jazyků [34]. Patřily mezi ně Java, JavaScript, .NET, ObjC a Python. Pro následujících 5 byl vývoj ve stádiu alpha nebo beta: Dart, Go, Node.js, Ruby. Kromě poskytnutí klientské knihovny Google také zpracoval dokumentaci a ukázkové skripty pro zmiňované knihovny. Kromě toho jsou všechny ze zmiňovaných knihoven licensované jako open source – Apache-2.0 nebo BSD-3.

4.2.4 Zmocnění uživatele do Google Kalendáře

V kalendáři mohou uživatelé ukládat i citlivé informace. Informace o tom, kde se daná osoba v konkrétní datum a čas náchází může být poměrně snadno zneužita. Aby bylo takovému scénáři zabráněno používá Google postupy pro ověření identity uživatele. Teprve po tom co se uživatel prokáže, je mu umožněno přistupovat k neveřejným informacím, ke kterým mu byl přístup udělen. Pro ověření identity uživatele do API Google Kalendáře lze využít výhradně OAuth 2.0 [35]. Více informací o tomto způsobu ověření lze najít v sekci 4.3.1. Součástí této metody je evaluace požadovaného rozsahu přístupu. Pro služby Google probíhá tím způsobem, že se otevře okno ve webovém prohlížeči, ve kterém aplikace žádá konkrétní účet o udělení přístupu k nějakým službám nebo zdrojům. Rozsah požadovaného přístupu je tam jednoznačně popsán. Uživatel se musí přihlásit ke svému účtu a tento požadavek odsouhlasit. Teprve po schválení může aplikace získat token.

4.2.5 Rozsah přístupu aplikace k API kalendáře

V minulé sekci bylo popsáno, že aby aplikace získala ověřovací token, musí jí být nejprve schválen přístup disponentem účtu. Tyto rozsahy se pro různé služby Google liší [36]. V API Google Kalendáře se setkáme s rozsahem – celý kalendář. Ten dovolí úpravu i čtení všech zdrojů v kalendáři daného účtu. Rozsah je možné omezit pouze na čtení. Omezenou alternativou je udělit přístup jenom k událostem. Tuto možnost je též možné limitovat na čtení. Posledním rozsahem přístup k nastavení, ale jenom v režimu čtení.

4.3 Zabezpečení

4.3.1 Autorizace uživatele

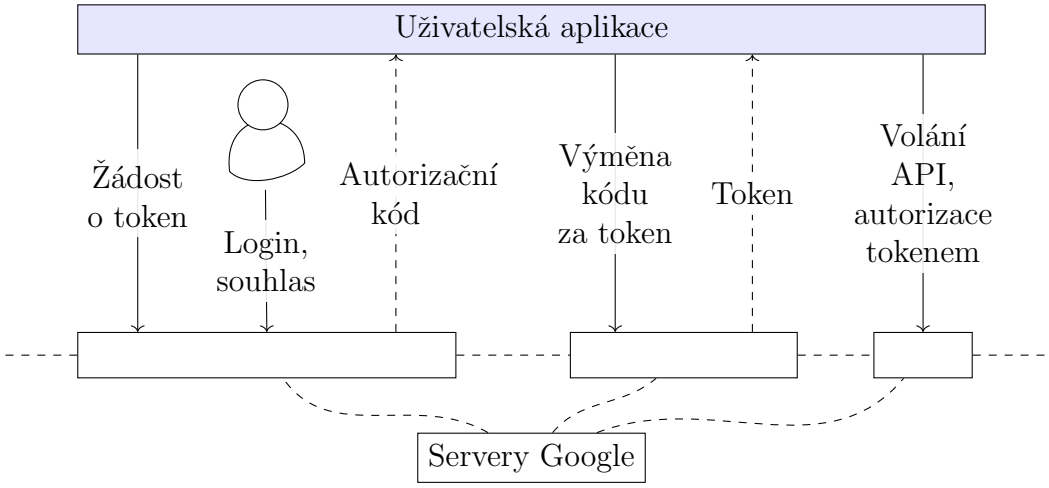
OAuth 2.0

Protokol OAuth 2.0 [37] je metodou, která slouží k ověření identity uživatele a udělení příslušných oprávnění uživateli (aplikaci) ke zdrojům jako jsou například

aplikace nebo data. Nejedná se o konkrétní implementaci tohoto ověření, ale popisuje obecný postup jak ho dosáhnout.

Postup udělení přístupu

Aplikace chce získat přístup ke zdroji, který náleží nějakému vlastníkovi. Autorizačnímu serveru pošle požadavek s přístupovými údaji, které předtím musela získat od vlastníka, a požadovaným rozsahem přístupu. Server na základě přístupových údajů vyhodnotí zda udělí v žádaném rozsahu povolení. Toto povolení zastupuje přístupové údaje aplikace v žádosti o token [6]. Pokud autorizační server žádost schválí, pošle aplikaci nazpátek token. Ten už může aplikace použít k tomu aby se u serveru prokázala a ten jí v daném rozsahu obsluhoval. Je pro aplikaci ověřovacím dokumentem. S požadavkem ho aplikace posílá serveru a ten vyhodnocuje zda má aplikace k danému požadavku skutečně oprávnění. Pokud ano, je aplikaci poskytnut přístup k informaci nebo zdroji, nebo je proveden požadavek. Token vžy ale platí poze v nějakém rozsahu – takovám, pro nějž byl token schválen. Na obrázku 4.1 je ilostrován mechanizmus OAuth2.



Obrázek 4.1: Způsob ověření uživatele pomocí OAuth2

Konkrétní podobu tokenu protokol OAuth 2.0 nestanovuje a způsob implementace zůstává na vývojáři.

Výhodnou tohoto systému je, že klient nepotřebuje přesně vědět jakým mechanismem samotné ověření funguje, protože probíhá přímo na serveru.

[6] data, ze kterých bude server schopen spolehlivě udělit oprávnění aplikaci

Některé zranitelnosti systému

Model OAuth 2.0 zajišťuje spolehlivou autorizace uživatele. To znamená, že zajišťuje aby k danému zdroji získal přístup jen takový uživatel nebo aplikace, kteří k němu přístup skutečně mají mít. Tento požadavek je splněn tehdy, když k tajným údajům uživatele (přístupovým údajům, bezpečnostním tokenům) má pouze oprávněný uživatel nebo aplikace. K jejich úniku ale za určitých okolností může dojít. Výběr několika napadnutelných nebo slabších míst tohoto systému je uveden dále.

Přímý útok na uživatele

Jedním ze způsobů odcizení přístupových údajů uživatele je takzvaný phishingový útok. Spočívá v tom, že se útočník pokusí přimět uživatele k přímému sdílení přístupových údajů. Velmi jednuduchá verze útoku může vypadat jako email, ve kterém se útočník vydává za někoho, u koho by se mohlo zdát přijatelné, že žádá o tak citlivá data Sofistikovanější forma může vypadat jako normální webový formulář pro přihlášení do aplikace, který je co nejvěrnější kopií originálu. Přitom ale přístupové údaje dostává útočník. Prevalence neustálé potřeby se někde autorizovat přispěla k tomu, že si uživatelé zvykli na různá přesměrování, na kterých jsou uživatelé vyzíváni k vyplňování přihlašovacích údajů. Při nepozornosti nebo neopatrnosti se může uživateli stát, že své přihlašovací údaje pošle někomu jinému než autorizačnímu serveru.

DNS spoofing

Zmiňovaný útok spočívá v tom, že útočník má administrátorský přístup k síti, na které je klient připojen. Poté může vydávat falešný DNS záznam pro danou doménu a tím přesměrovat data jinam než na požadovaný server. Samotný protokol OAuth 2.0 žádným způsobem neověřuje autentičnost autorizačního serveru a proto není tento druh útoku snadno detekovatelný. Náchylnosti k tomuto typu útoku lze eliminovat používáním TLS [7], který je blíže popsán v 4.3.2. Pomocí něho je možné spolehlivě ověřit identitu autorizačního serveru a tomuto typu útoku se vyhnout.

Pokud není klientský přístup k tokenu implementován dobře, může dojít k odcizení tokenu a v důsledku toho k odcizení dat, ke kterým uděluje token přístup. Systém chrání před úplným odcizením zdrojů, protože v tomto případě zůstávají

[7] Transport Layer Security protokol

přístupové údaje v bezpečí.

4.3.2 Zabezpečení komunikace se serverem

Protokol SSL

Protokol SSL pracuje na rozhraní transportní a aplikační vrstvy OSI modelu komunikace mezi klientem a serverem. Může být použit pro zabezpečení různých druhů komunikace. Jeho význam je popsán v [38] a klade si tři hlavní cíle:

- Ověřit autentičnost obou stran komunikace.
- Zajistit, že obsah komunikace bude přístupný pouze subjektům komunikace a ne dalším třetím stranám.
- Ověřit správnost přenesených dat – zjistit, jestli při přenosu nedošlo k poškození dat.

Těchto cílů je dosaženo pomocí několika kroků. Nejprve klient naváže se serverem nezabezpečenou komunikaci a vyžádá si zabezpečenou komunikaci. V požadavku pošle také informaci o tom jaké kryptografické algoritmy klient používá. Server vybere takový, který je sám schopen využít. Server také poskytne certifikát, díky kterému klient ověří autentičnost serveru. Následně klient i server bezpečně vygenerují bezpečnostní klíče, které budou v zabezpečené komunikaci používat k šifrování dat.

protokol TLS

Tento protokol je přímým nástupcem SSL. Vznikl aby nahradil jeho bezpečnostní nedostatky. Struktura protokolu přitom zůstala stejná a hlavní rozdíly jsou v použitých kryptografických algoritmech.

Kapitola 5

Implementace

5.1 Náplň práce

Požadavky na vlastnosti práce byly popsány v kapitole 2.2. Z nich plynou jednotlivé kroky, které v rámci práce budu muset podniknout abych se dostal ke konečnému produktu. Těmi kroky jsou:

- Vybrat vhodnou hardwarovou platformu – to v praxi znamená najít vhodný jednodeskový počítač s displejem, který bude dostatečně výkonný k tomu aby na něm mohla být spuštěna grafická aplikace. Musí umožňovat připojení k internetu a mít dostatečné množství paměti pro uložení lokální databáze událostí (poslední požadavek je téměř samozřejmostí).

- Vybrat vhodnou softwarovou platformu. To bude obnášet selekci dobrého, dostupného a spolehlivého operačního systému, na jehož „bedrech“ poběží aplikace. Najít použitelné vývojové prostředí pro vývoj aplikace, vybrat takový programovací jazyk, který nabízí prostředky pro implementaci požadovaných funkcí – musí mít klientské knihovny pro API Google Kalendáře a rozhraní pro grafické uživatelské prostředí.

- Navrhnout vhodné grafické uživatelské prostředí. Tento úkol bude vyžadovat stanovení požadovaných funkcí, k jejichž ovládání budou přiřazeny grafické prvky tak, aby bylo ovládání intuitivní a rychlé.

- Vytvořit aplikaci a implementovat potřebné funkce bude nejkomplexnější úkol.

Pro jeho dosažení podniknu následující:

- Implementuji programové ovládání uživatelského prostředí.
- Vytvořím databázi, do které ve vhodném formátu uložím data týkající se uložených událostí.
- Zjistím jak komunikovat s API Google Kalendáře a za použití klientské knihovny implementuji a zajistím synchronizaci.
- Zajistím správný chod aplikace i bez připojení k internetu – nesmí být závislá na Google Kalendáři.

5.2 Operační systém

Pro projekt jsem vybral operační systém Raspberry Pi OS kvůli jeho dobré podpoře a také pro širokou paletu dostupných programů. Pro účely projektu tuto výhodu ani nevyužiji, ale mohla by se hodit pro budoucí vývoj.

5.3 Nástroje pro vývoj aplikace

5.3.1 Programovcí jazyk

Rozhodoval jsem se mezi dvěma kandidáty – Pythonem a JavaScriptem. Tyto dva jsem volil proto, že s oběma mám nějakou zkušenost a chtěl jsem se vyhnout programování poměrně komplexních úkonů v nižších programovacích jazycích jako jsou například C nebo C++, s nimiž jsem měl v minulosti také zkušenost. Zásadním výběrovým kritériem byla existence klientské knihovny pro API Google Kalendáře. Pro oba zmiňované kandidáty existuje oficiální API a proto oba vyhovovaly požadavkům aplikace. Za důležitou jsem považoval i existenci kvalitních knihoven pro vývoj grafického uživatelského prostředí. Těch je ale k dispozici pro obě možnosti dostatek.

Abych se dobře rozhodl, pokusil jsem se vytvořit jednoduchý testovací skript, který měl za úkol připojit se ke Kalendáři Google a získat nějaká data. Při implementaci této funkce v JavaScriptu jsem ale narazil na problémy, se kterými jsem si ani s pomocí dokumentace nebyl schopný poradit. Oproti tomu při použití knihovny

pro Python jsem dosáhl kýženého výsledku a API se v té krátké zkušenosti chovalo podle očekávání. Tato zkušenost zváhla misky pomyslných vah ve prospěch Pythonu.

5.3.2 Grafické uživatelské rozhraní

Výběr vhodné knihovny

Různé knihovny pro vývoj grafického uživatelského prostředí v Pythonu jsou popsané v sekci 2.4. Ze zmiňovaných jsem vyzkoušel Qt a Tk. Qt jsem zvolil kvůli tomu, že jsem s ní měl předchozí zkušenosti. Tk se zdála jako dobrá alternativa – knihovna slibovala vysokoúrovňové rozhraní s grafickými prvky, navrženými od základu pro vyšší dynamické programovací jazyky. Oproti tomu Qt je napsaná v C++. Tyto předpoklady se promítly i do mých zkušeností – Tk v kombinaci s Tkinter nabízely poměrně elegantní rozhraní pro Python, které mi bylo sympatičtější. Když ale došlo na implementaci specifičtějších funkcí, tak mi připadalo, že rozhraní Tkinter postrádalo flexibilitu, kterou jsem si představoval. Kromě toho jsem shledal, že designově uživatelské prostředí trochu zaostávalo. Naproti tomu Qt v kombinaci s rozhraním PyQt5 nabízela dostatečnou flexibilitu společně s uspokojivým designem. Mimo to Qt obsahuje všechny grafické prvky, které jsem potřeboval, detailní dokumentaci a další podpůrný software, jenž mi značným způsobem usnadnil návrh uživatelského prostředí. Analogické nástroje se mi pro Tk najít nepodařilo.

Návrhové nástroje

K návrhu grafického uživatelského prostředí v Qt jsou k dispozici četné oficiální nástroje. Já jsem využil Qt Designer. Ten umožňuje návrh uživatelského prostředí nezávisle na zdrojovém kódu aplikace. V návrhu se pužívají předdefinované grafické prvky, kterým se v dokumentaci přezdívá widgety. Mimo to umožňuje zvolit rozložení okna aplikace, což se postará o dobré využití prostoru okna aplikace při jeho škálování.

V programu Qt Designer jsem vytvořil návrh uživatelského prostředí. Tento návrh je uložen do souboru s připonou `ui`. Soubor reprezentuje rozložení uživatel-

ského prostředí stromovou strukturou XML[1] v čitelném formátu. Ten samotný ve skriptu přímo použít nelze. Ke konverzi do formátu, který je použitelný pro skript Pythonu jsem používal nástroj `pyuic5` z kolekce [39]. Ten při volání z příkazového řádku s vlajkami `-x` (po té následuje název vstupního ui souboru) a `-o` (za ní následuje název generovaného výstupního souboru), vytvoří nový soubor ve formě skriptu Pythonu. Obsah takto vygenerovaného souboru stačí ve skriptu aplikace importovat. Ve své aplikaci to provádím příkazem `from ui.ui import *`. Tady stojí za zmínku, že import se týká souboru s názvem „ui.py“, který se nachází ve složce s názvem „ui“, přestože by se mohlo zdát, že to je jinak. Pojmenovávací konvence, kterou jsem zvolil není v tomto případě praktická.

5.4 Příprava Raspberry Pi

5.4.1 Instalace operačního systému

Předtím než jsem mohl začít aplikaci testovat přímo na Raspberry Pi, bylo potřeba nainstalovat operační systém. Pro tuto úlohu jsem použil nástroj Balena Etcher. Ten dokáže přečíst soubor s obrazem operačního systému, který jsem stáhl z oficiálních stránek [40] a zapsat ho na médium, ze kterého vytvoří bootovací disk. Minimální požadované množství místa na médiu je 8 GB. Já jsem obraz operačního systému nechal zapsat na 64GB micro SD kartu, již jsem později vložil do příslušného slotu na desce Raspberry Pi. Po připojení jednodeskového počítače k napájení se spustí už nainstalovaný operační systém a zbývá pouze nastavit několik věcí jako jazyk a rozložení klávesnice.

5.4.2 Příprava před spuštěním programu

Na čerstvé instalaci Raspberri Pi OS nejsou nainstalované všechny potřebné balíčky, které program pro správné fungování potřebuje. Proto jsem použil nástroj pro instalaci balíčků a nainstaloval balíčky uvedené v příloze A.1. Ty umožňují ovládat knihovnu Qt z Pythonu a přidávají podporu pro virtuální klávesnici v aplikaci.

[1] e**X**tensible **M**arkup **L**anguage – značkovací jazyk, standard pro přenos dat

Ostatní potřebné balíčky byly už nainstalované.

Kromě systémových balíčků bylo potřebovat doinstalovat i některé moduly pro Python. jejich seznam je k nalezení v příloze A.2. Ty se z veliké části týkaly instalace klientských knihoven pro komunikaci s API Google Kalendáře. Pro instalaci jsem využil správce balíčků pro Python – pip. S jeho pomocí jsem balíčky instaloval z příkazového řádku následujícím způsobem:

```
pip install <nazev-balicku1> <nazev-balicku2> <nazev-balicku3>
```

5.5 Autorizace do API Google Kalendáře

K tomu aby aplikace mohla používat API Google kalendáře je potřebné aby se autorizovala. Proces autorizace je poněkud nešikovný – mnoho několik kroků a proto je zde blíže popsán a ilustrován snímky obrazovky. Přotože jich je mnoho tak budou k nalezení v příloze B

5.5.1 Příprava na straně Google Cloudu

Ve webovém rozhraní Google Cloudu [41] je potřeba nejprve vytvořit nový projekt. Teprve k projektu lze přiřadit službu ověření uživatele (OAuth2) a Google Calendar API. Jediný parametr, který se při vytváření nového projektu musí zadat je libovolně zvolený název.

V rámci nově vytvořeného projektu je potřeba nastavit obrazovku pro ověření uživatelů a udělení oprávnění aplikaci ke kalendáři uživatele. Během tohoto procesu jsou nastaveny informace o projektu a jeho nastavení. Nakonec se přidají uživatelé, kteří budou mít k projektu přístup. Postup je zdokumentován v příloze B.2.

Posledním krokem, který jsem podnikl uvnitř webového prostředí Google Cloud je generace klientské identifikace. V našem případě se ale přesněji řečeno jedná o identifikaci aplikace. Pro každého klienta je potřeba zvolit 2 parametry – typ aplikace a identifikační název. V případě mého programu se z hlediska typu jednalo o desktopovou aplikaci a název jsem zvolil libovolně, ale tak aby byl deskriptivní. Po vygenerování příslušných identifikačních dat je Pro potřeby aplikace se musí vygenerované identifikační údaje stáhnout ve formátu `json`, stažený soubor přejmenovat

na `client_secret.json` a vložit ho do složky secret. Postup je opět dokumentován v B.3.

5.6 Návrh programu a grafického uživatelského prostředí

Grafické uživatelské prostředí jsem navrhoval v programu Qt Designer. Navrhovaná aplikace se všemi implementovanými funkcemi nevyžaduje více než jedno okno pro provoz. To je plátnem pro všechny grafické prvky. Návrh se sestával z vybírání vhodných grafických prvků z hlediska funkcionality, ale také z hlediska zakomponování do uživatelského prostředí. Hlavní okno se automaticky při spuštění přepne do režimu na celou obrazovku, aby byla využita co největší část displeje. Aby nebylo potřeba připojovat k Raspberry Pi klávesnici, je aplikace nastavena tak, že při kliknutí na prvky, ve kterých je potřeba zadávat text, zobrazí na displeji klávesnici.

Základní funkce, které aplikace zprostředkovává jsou:

- Grafické uživatelské prostředí
- Správa událostí v lokální databázi
- Komunikace s Google Kalendářem prostřednictvím API
- Budík
- Stopky

Ty stěžejní budou detailněji popsány v dalších sekcích.

5.6.1 Grafické uživatelské prostředí

Pro zjednodušení ovládání dotykem umožňuje aplikace přístup ke všem funkcím z jediného okna. Také jsem při návrhu bral v potaz fakt, že aplikace bude zobrazena na displeji s rozlišením 800x480 pixelů. Tomu jsem se snažil návrh přizpůsobit – uživatelské rozhraní jsem směřoval k využití celé šířky obrazovky a k ovládání jsem

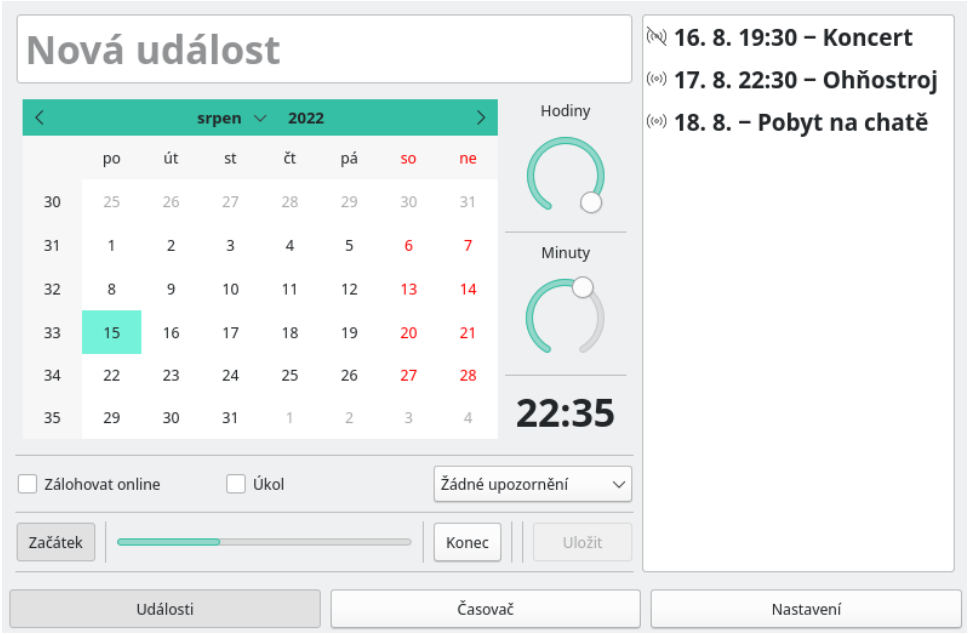
upřednostňoval tlačítka. V popisu se vyskytují názvy grafických prvků z knihovny bez vysvětlení. Popis jednotlivých prvků se nachází v sekci 2.5.2.

Okno se skládá ze dvou částí, které se starají o rozdílné funkce. Zatímco navigační panel (dole) zajišťuje přepínání mezi funkcionalitami, funkční okno (vrchní část) je určeno pro interakci se samotnými funkcemi aplikace. Pro upřesnění představy je k popisu přiložen obrázek 5.2.

Funkční okno

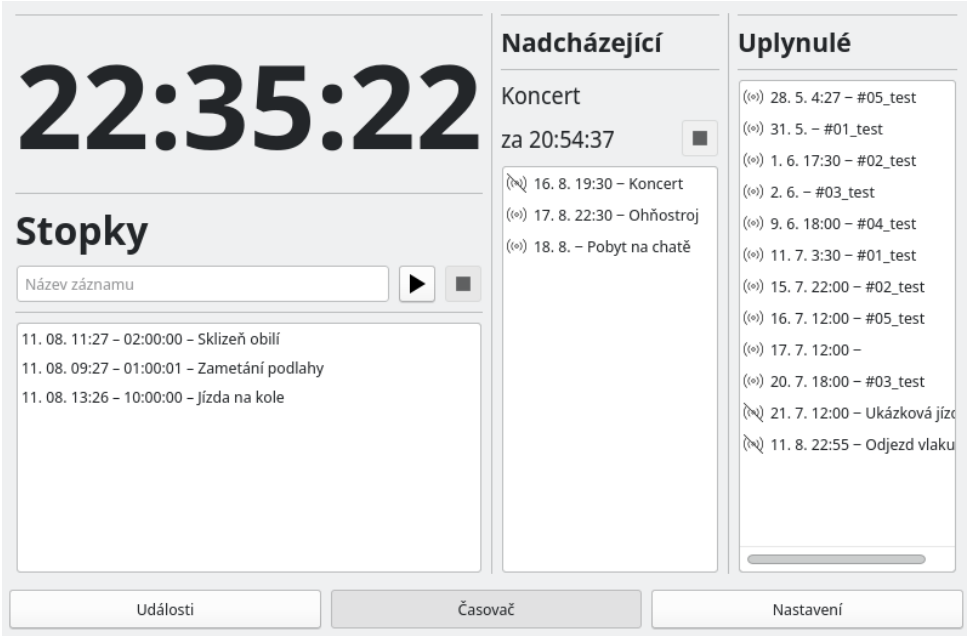
Funkčnímu oknu dominuje veliký `QStackedWidget`. Ten je základnou všech grafických prvků určených pro interakci s jednotlivými funkcionalitami.

Na obrazovce pro tvorbu událostí nalezneme `QCalendarWidget`, na kterém uživatel vybírá datum. Vedle se nachází dva ciferníky, na nichž se nastaví čas – hodiny respektive minuty. Tímto procesem je vybrán začátek události. Kliknutím na tlačítko „Konec“ se zobrazí duplikát ovládacích prvků, který ale slouží k výběru data a času konce události. Na pravém boku nalezneme seznam nadcházejících událostí, zobrazených pomocí QListWidget. Název vytvářené události se píše do QLineEdit ve vrchní části okna. Nastavený název je podmínkou pro uložení události. Zároveň zde lze nastavit, zda má být nová událost synchronizovaná s Google kalendářem, nebo jestli má existovat pouze v offline databázi.



Obrázek 5.1: Rozložení uživatelského prostředí

V režimu časovače jsou důležitým prvkem digitální hodiny ukazující aktuální čas. Ty jsou realizovány pomocí prvku QLabel, jehož text je ze skriptu každou sekundu měněn. V levé části se nachází funkcionalita stopování (časovače). Ta umožňuje měřit uběhnutý čas a záznam pojmenovat a uložit. Zastavení časovače se děje na základě zmáčknutí tlačítka s ikonou černého čtverce. Pod stopkami je pomocí `QListWidget` zobrazen výčet uložených měření. Stejný prvek je použit v prostředním i pravém sloupci. V těch případech se ale jedná o seznam nadcházejících respektive uplynulých událostí. Pod titulkem nadcházejících událostí je také zobrazen odpočet, který ukazuje čas do začátku nadcházející události.



Obrázek 5.2: Rozložení uživatelského prostředí

Poslední obrazovkou jsou nastavení. Na ní lze pomocí různých prvků nastavit některé z parametrů, které ovlivní chod aplikace.

Navigační panel

Hlavním určením navigačního panelu je přepínání jednotlivých funkcionalit aplikace. Panel se skládá ze třech tlačítek, přičemž každé z nich přepíná na jednu z funkcionalit systému. Aplikace využívá signálu `clicked`. Ten je vyslán ve chvíli zmáčknutí tlačítka. V ten moment se vyhodnotí, které tlačítko bylo zmáčknuto a náležitým způsobem se přepne obsah funkčního okna. Přepíná se mezi módem tvorby kalendáře, budíkem a nastavením. Tlačítka jsou nastavená v módu `checkable`. Zmáčknuté tlačítko zůstane zmáčknuté až do chvíle, kdy uživatel zmáčkne jiné a v tu chvíli se přepne na další. To znamená, že v jeden moment může být zmáčknuté maximálně jediné tlačítko.

5.6.2 Správa událostí v lokální databázi

Aplikace kromě komunikace s Google Kalendářem umožňuje ukládání událostí i do lokální databáze. Abych nepracoval s dvěmi různými typy dat, navrhl jsem lokální da-

tabázi tak, aby pracovala s daty ve stejném formátu jako Google Kalendář. To znamená, že obdrží data ve formátu JSON, který je pro použití v Pythonu konvertován na slovník. Uložení probíhá formou exportu zpět do formátu JSON a pak uložení do souboru. Kromě toho při importování uložených událostí dojde k extrakci data a času začátku a konce události do objektu `datetime`. To potom umožňuje jednoduché porovnávání dat a tím dovoluje i jednoduchou filtraci událostí na základě data a času konání. Toho využívá aplikace ve chvíli, kdy rozděluje události na nadcházející a uplynulé.

5.6.3 Komunikace s Google Kalendářem

O komunikaci se servery Google se stará objekt `CalManager`, který byl navržen právě pro stahování respektive nahrávání do Google Kalendáře.

Ještě před prvním zpuštěním aplikace je potřeba nastavit účet Google pro to aby umožňoval přístup k datům přes webové API. Toho lze dosáhnout opakováním kroků popsaných v 5.5.1. Kromě toho je potřebné do konstruktoru objektu `CalManager` specifikovat v argumentu `main_cal` název kalendáře, se kterým bude aplikace interagovat.

Po úspěšném nastavení objekt zprostředkovává autorizaci, tvorbu nových událostí a stahování dat o existujících událostech.

Kapitola 6

Závěr

V bakalářské práci jsem se zabýval návrhem IoT časovače/budíku. Základním kritériem bylo aby vzniklá aplikace mohla komunikovat s externí službou Google Kalendář za použití API. Kromě toho měla umožňovat i ukládání do lokální databáze událostí.

Projekt jsem vyvinul na platformě jednodeskového počítače Raspberry Pi 4B s operačním systémem Raspberry Pi OS. Zadané funkce jsem implementoval ve skriptovacím jazyce Python za použití externích knihoven i navžené databáze. Pro návrh grafického uživatelského prostředí jsem využil nástrojů knihovny Qt. Ovládat zařízení lze pomocí dotykové obrazovky, která je pevně přidělaná na zařízení. K audio výstupu Raspberry Pi je připojen reproduktor se zesilovačem, který zprostředkovává akustické upozornění uživatele.

Do veliké míry se implementace podařila, ale výsledná aplikace má i tak určité nedostatky. Asi největším z nich je nutnost zdlouhavého nastavování ve službě Google Cloud. K automatizaci tohoto procesu by mohl v směřovat další vývoj aplikace. Kromě toho implementované možnosti nastavení jsou omezené a aplikace není tím pádem příliš flexibilní.

Seznam literatury

[1] Internet of Things (IoT): Definitions, Challenges and Recent Research Directions. *International Journal of Computer Applications*. 2015, **2015**(128), 37-47.

[2] SERPANOS, Dimitrios a Marilyn WOLF. *Internet-of-Things (IoT) Systems: Architectures, Algorithms, Methodologies*. Springer Cham, 2017. ISBN 978-3-319-69715-4. Dostupné z: doi:https://doi.org/10.1007/978-3-319-69715-4.

[3] SOMAYYA, Madakam, Ramaswamy R. a Tripathi SIDDHARTH. Internet of Things (IoT): A Literature Review. *Journal of Computer and Communications*. 2015, **3**(5), 164-173. Dostupné z: doi:10.4236/jcc.2015.35021.

[4] WEISER, Mark. The Computer for the 21st Century. *Scientific American*. 1991, **1991**(9), 94-104.

[5] FOOTE, Keith. A Brief History of the Internet of Things. *Dataversity* [online]. 2022 [cit. 2022-05-03]. Dostupné z: www.dataversity.net/brief-history-internet-things/.

[6] BERG, Frans, Robert PERRONS, Ian MOORE a Gert SCHUT. *Business Value From Intelligent Fields*. 2010. Dostupné z: doi:10.2523/128245-MS.

[7] TASHPULATOV, Jakhongir. *IoT časovač* [online]. 2021 [cit. 2022-08-15]. Dostupné z: http://hdl.handle.net/10467/96705. Bakalářská práce. České vysoké učení technické v Praze.

[8] LUBANOVIC, Bill. *Introducing Python*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O-Reilly Media, 2014. ISBN 978-1-449-35936-2.

[9] *Tk Backgrounder* [online]. [cit. 2022-08-07]. Dostupné z: http://tkdocs.com/resources/backgrounder.html.

[10] *Kivy Project, Philosophy* [online]. [cit. 2022-08-07]. Dostupné z: https://kivy.org/doc/stable/philosophy.html.

[11] *Overview of wxPython* [online]. [cit. 2022-08-07]. Dostupné z: https://www.wxpython.org/pages/overview/.

[12] *Supported Platforms* [online]. [cit. 2022-08-07]. Dostupné z: https://www.qt.io/product/supported-platforms-languages.

[13] *Signals & Slots* [online]. [cit. 2022-08-13]. Dostupné z: https://doc.qt.io/qt-5/signalsandslots.html.

[14] *QStackedWidget Class* [online]. [cit. 2022-08-13]. Dostupné z: https://doc.qt.io/qt-5/qstackedwidget.html.

[15] *QLineEdit Class* [online]. [cit. 2022-08-13]. Dostupné z: https://doc.qt.io/qt-5/qlineedit.html.

[16] *QListWidget Class* [online]. [cit. 2022-08-13]. Dostupné z: https://doc.qt.io/qt-5/qlistwidget.html.

[17] *QCalendarWidget Class* [online]. [cit. 2022-08-14]. Dostupné z: https://doc.qt.io/qt-5/qcalendarwidget.html.

[18] *QPushButton Class* [online]. [cit. 2022-08-13]. Dostupné z: https://doc.qt.io/qt-5/qpushbutton.html.

[19] *QLabel Class* [online]. [cit. 2022-08-13]. Dostupné z: https://doc.qt.io/qt-5/qlabel.html.

[20] MOHAN, Vishnu. Raspberry Pi Models Comparison: Which Pi is Right for My Application? *Raspberry Expert* [online]. 2021 [cit. 2021-12-01]. Dostupné z: https://raspberryexpert.com/raspberry-pi-models-comparison/.

[21] GAY, Warren W. Mastering the Raspberry Pi: A complete reference guide and project idea generator for the Raspberry Pi. Apress: Berkeley, CA, USA, 2014. ISBN 978-1-4842-0181-7.

[22] Raspberry Pi OS. *Raspberry Pi* [online]. 2021 [cit. 2021-12-02]. Dostupné z: https://www.raspberrypi.com/documentation/computers/os.html.

[23] An overview of Windows 10 IoT Core. *Microsoft Docs* [online]. 2021 [cit. 2021-12-02]. Dostupné z: https://docs.microsoft.com/en-us/windows/iot-core/windows-iot-core.

[24] [online]. [cit. 2022-08-01]. Dostupné z: https://www.ibm.com/cloud/learn/api.

[25] FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Irvine, California, 2000. Dissertation. UNIVERSITY OF CALIFORNIA.

[26] XIANJUN, Chen, Ji ZHOUPENG, Yu FAN a Yogsong ZHAN. Restful API Architecture Based on Laravel Framework. *Journal of Physics: Conference Series*. 2017, **910**, 012016. Dostupné z: doi: 10.1088/1742-6596/910/1/012016.

[27] *REST APIs* [online]. IBM Cloud Education, 2021 [cit. 2022-07-29]. Dostupné z: https://www.ibm.com/cloud/learn/rest-apis.

[28] *Different API Types* [online]. New York: Techolution, 2019 [cit. 2022-07-28]. Dostupné z: https://techolution.com/types-of-apis/.

[29] *Calendar API Reference*. 2021. Dostupné také z: https://developers.google.com/calendar/api/v3/reference.

[30] [online]. [cit. 2022-07-30]. Dostupné z: https://developers.google.com/calendar/api/v3/reference/acl.

[31] [online]. [cit. 2022-07-30]. Dostupné z: https://developers.google.com/calendar/api/v3/reference/calendarList.

[32] [online]. [cit. 2022-07-30]. Dostupné z: https://developers.google.com/calendar/api/v3/reference/calendars.

[33] [online]. [cit. 2022-07-30]. Dostupné z: https://developers.google.com/calendar/api/v3/reference/events.

[34] [online]. [cit. 2022-07-30]. Dostupné z: https://developers.google.com/calendar/api/downloads.

[35] *Authorizing Requests to the Google Calendar API* [online]. [cit. 2022-08-08]. Dostupné z: https://developers.google.com/calendar/api/guides/auth.

[36] *OAuth 2.0 Scopes for Google APIs* [online]. [cit. 2022-08-11]. Dostupné z: https://developers.google.com/identity/protocols/oauth2/scopes.

[37] BUCHER, Patrick P. a Christopher J. CHRISTENSEN. *OAuth 2*. 2019, 4-7. oauth_good. Dostupné také z: https://raw.githubusercontent.com/patrickbucher/inf-stud-hslu/master/infkol/thesis/paper.pdf.

[38] OPPLINGER, Rolf. *SSL and TLS: Theory and Practice*. 685 Canton Street Norwood, MA 02062: ARTECH HOUSE, 2009. ISBN 978-1-59693-447-4.

[39] *Pyqt-tools* [online]. [cit. 2022-08-08]. Dostupné z: https://pypi.org/project/pyqt5-tools/.

[40] *Operating system images* [online]. [cit. 2022-08-10]. Dostupné z: https://www.raspberrypi.com/software/operating-systems/.

[41] *Google Cloud Dashboard* [online]. [cit. 2022-08-10]. Dostupné z: https://console.cloud.google.com/home/dashboard.

Příloha A

A.1 Dodatečné instalace

- `python3-pyqt5`
- `qtvirtualkeyboardplugin`
- `qml-module-qt-labs-folderlistmodel`

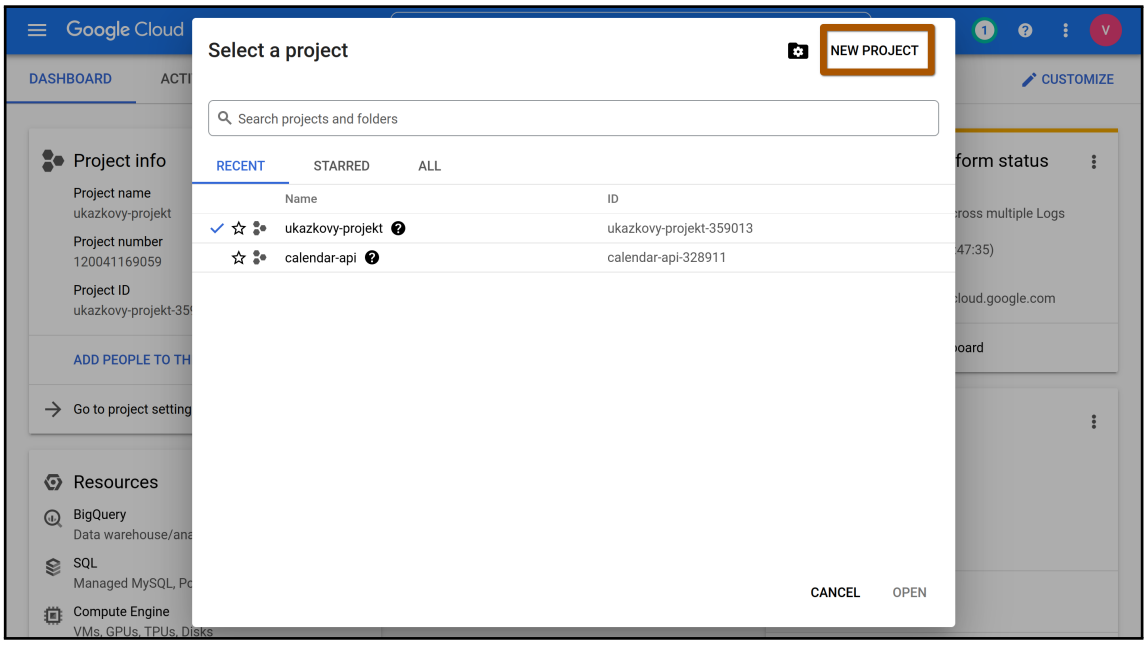
A.2 Instalované moduly Pythonu

- `pandas`
- `pygame`
- `aqtinstall`
- `google-auth-httplib2`
- `google-auth-oauthlib`
- `google-api-python-client`

Příloha B

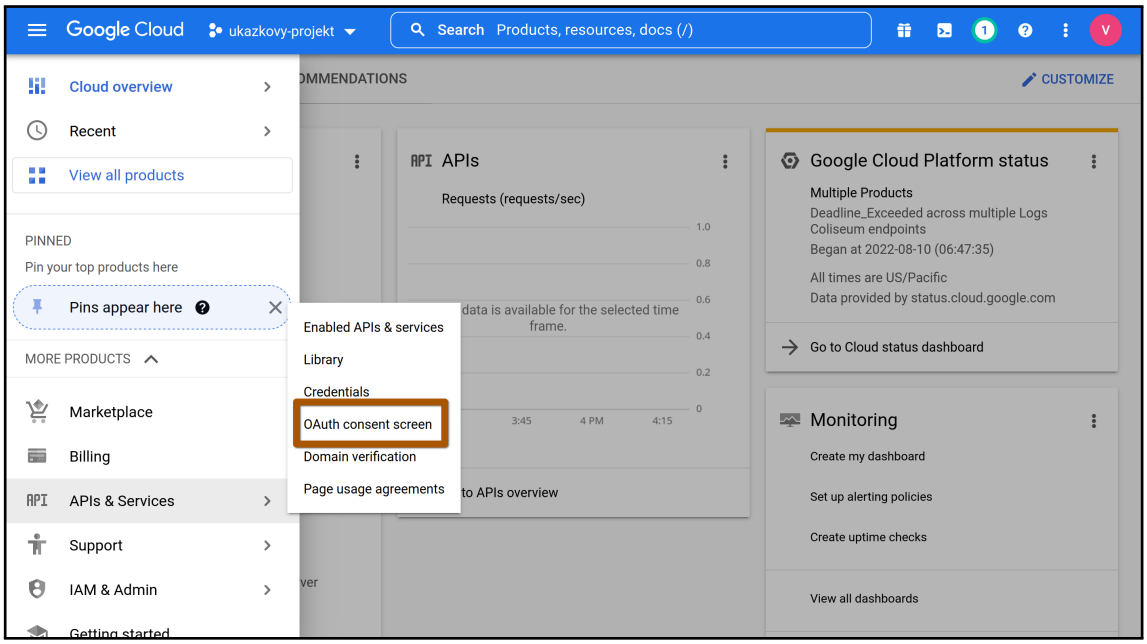
Příprava prostředí Google Cloud

B.1 Vytvoření aplikace

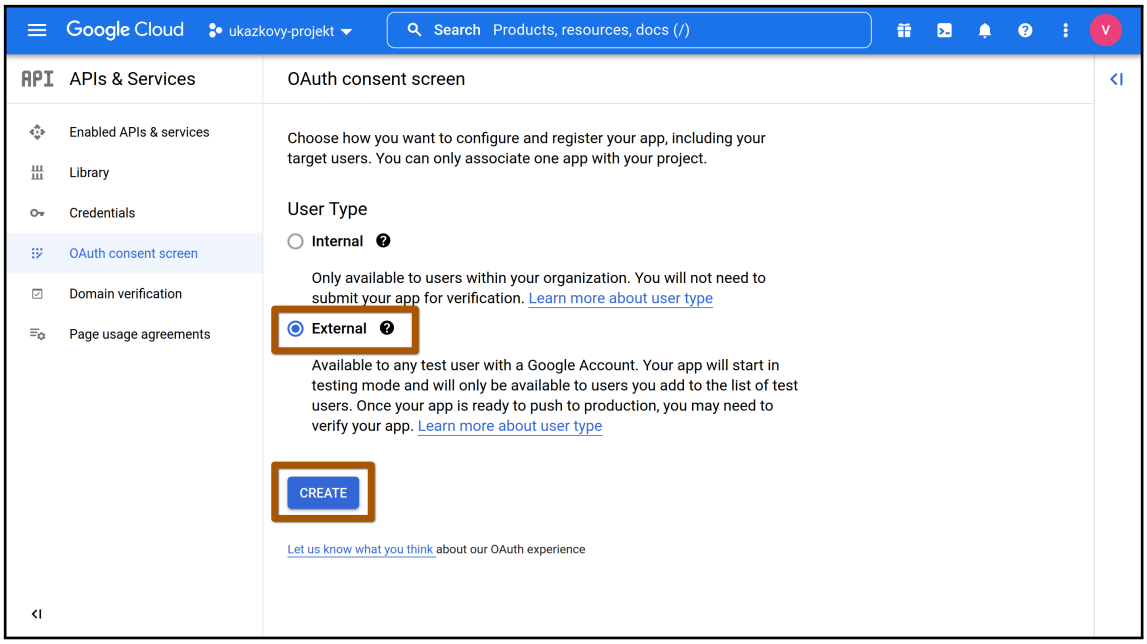


Obrázek B.1: Vytvoření nové aplikace v Google Cloudu

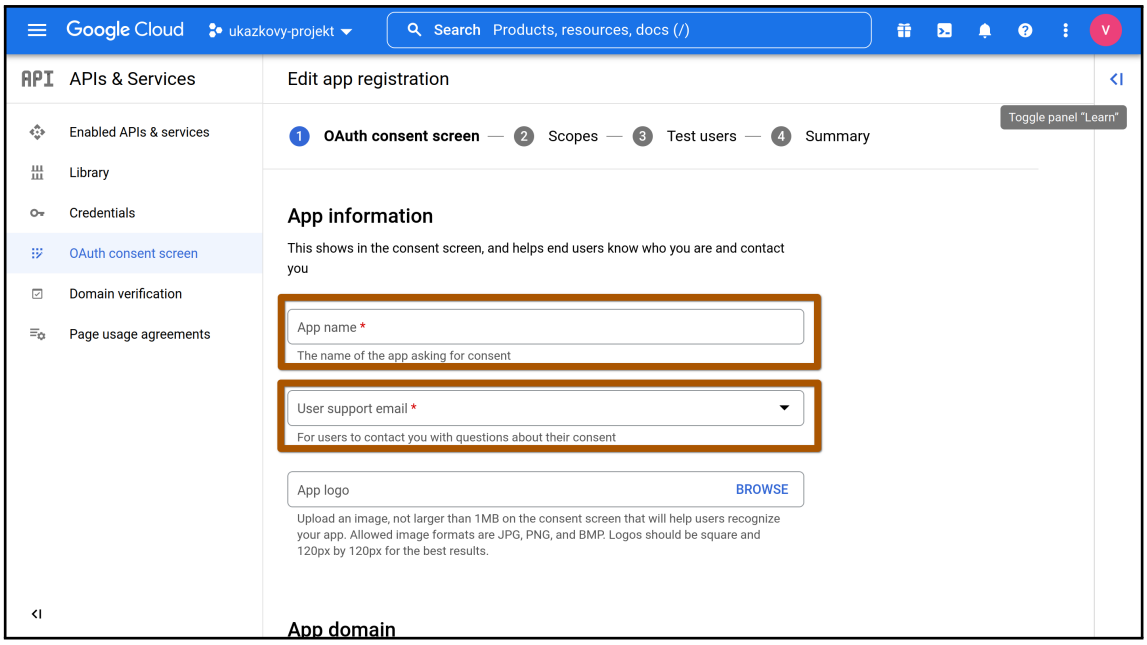
B.2 Nastavení ověřování uživatele



Obrázek B.2: Přepnutí do zobrazení nastavení autorizační obrazovky

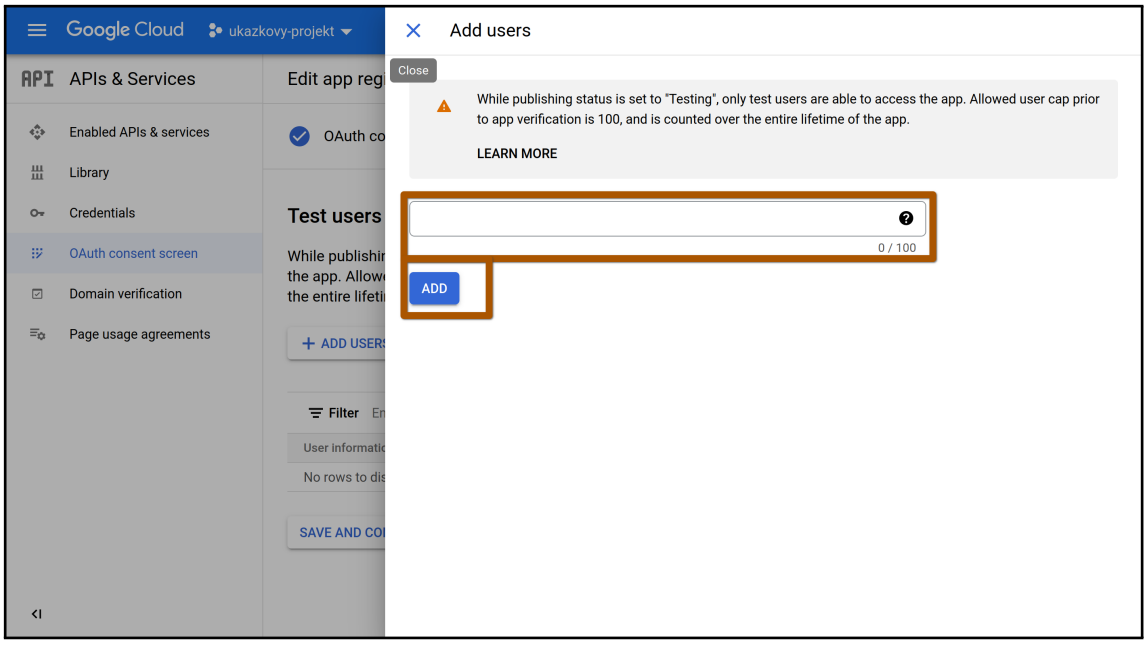


Obrázek B.3: Nastavení typu uživatele interní / externí

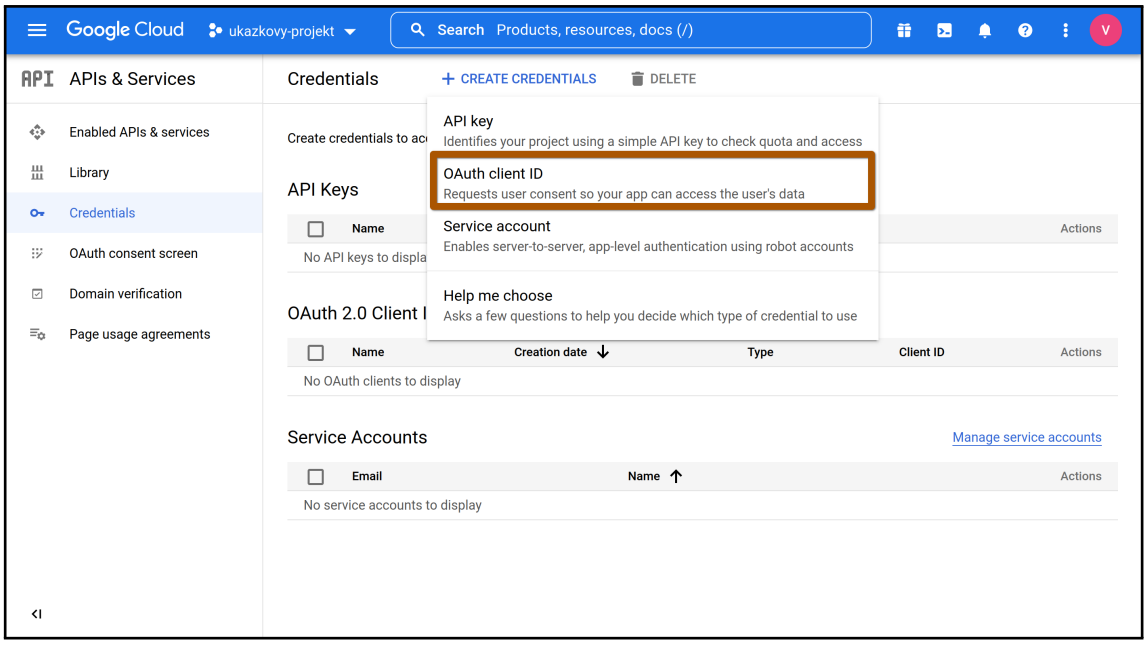


Obrázek B.4: Nastavení informací o aplikaci, které se zobrazí na autorizační obrazovce

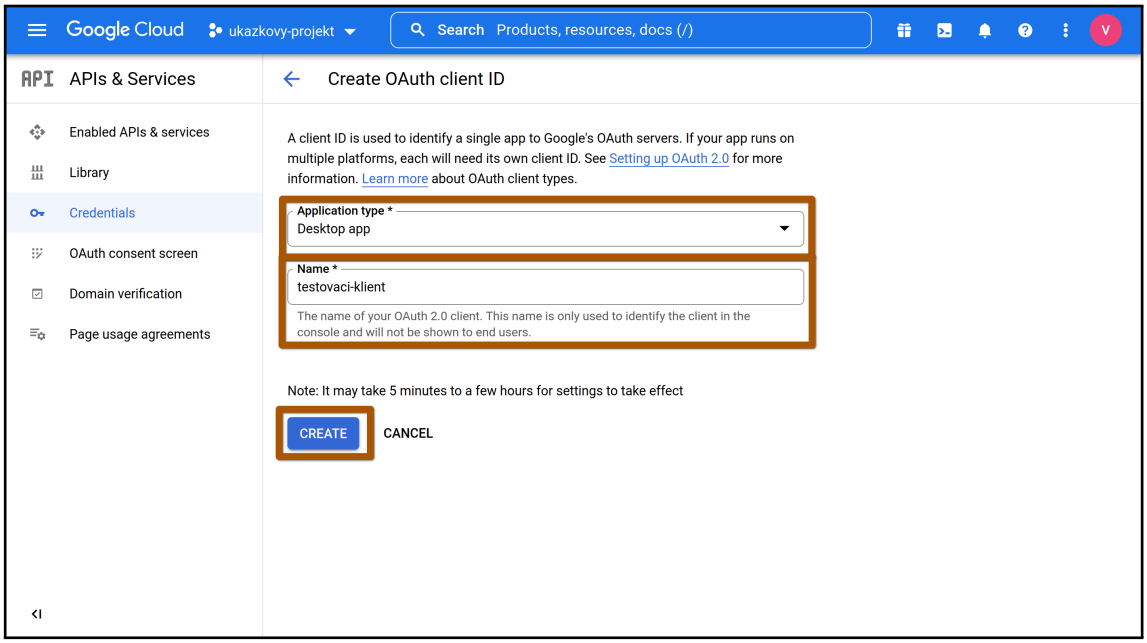
B.3 Přidávání uživatelů



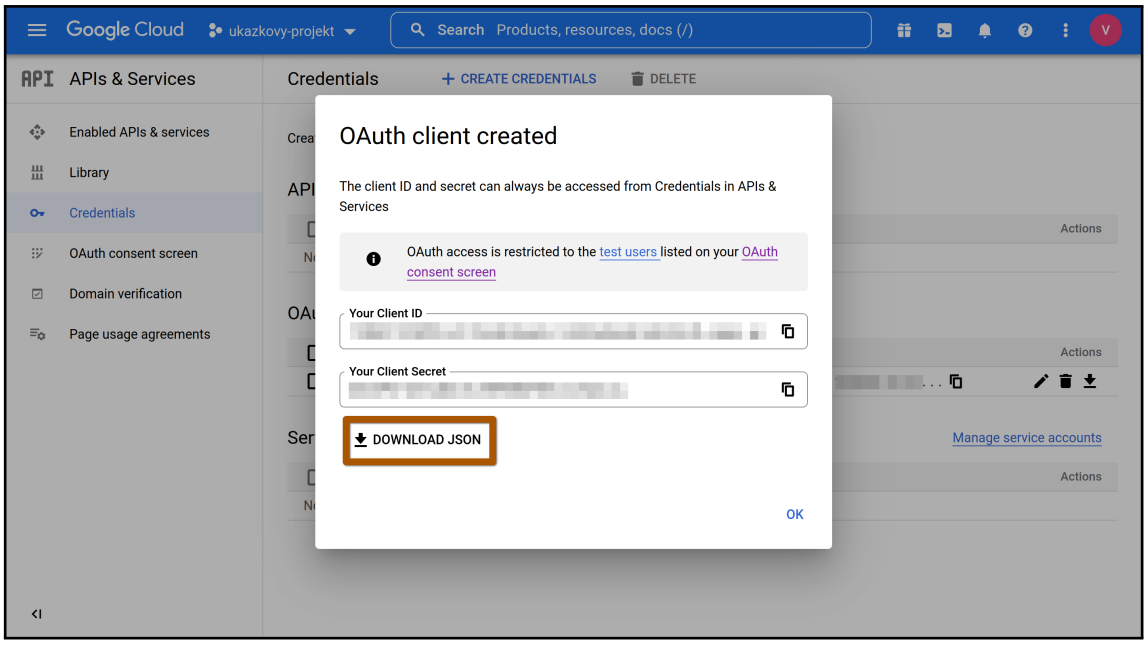
Obrázek B.5: Přidání uživatelů, kteří budou mít přístup k aplikaci



Obrázek B.6: Přepnutí do zobrazení generátoru klientské identifikace



Obrázek B.7: Vytvoření klientských identifikačních údajů na základě typu aplikace



Obrázek B.8: Stažení klientských identifikačních údajů