VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

PODPORA PŘÍPRAVKU FITKIT V PROSTŘEDÍ VISUAL STUDIO CODE

FITKIT PLUGIN FOR VISUAL STUDIO CODE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN CHALOUPKA**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. ZDENĚK VAŠÍČEK, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Chaloupka Jan**
Program: Informační technologie
Název: **Podpora přípravku FITkit v prostředí Visual Studio Code**
FITkit Plugin for Visual Studio Code
Kategorie: Web

Zadání:

1. Seznamte se platformou FITkit. Zaměřte se především na způsob správy projektů, jejich překlad a programování přípravku FITkit. Seznamte se s tvorbou zásuvných modulů rozšiřujících funkcionalitu editoru Visual Studio Code (VSCode).
2. Navrhněte rozšíření pro VSCode, které bude umožňovat autentizovaným uživatelům pohodlně pracovat s přípravkem FITkit bez nutnosti manuální instalace softwarového vybavení potřebného pro překlad a programování. Modul by měl umožňovat zejména správu projektů, editaci zdrojových kódů projektu, překlad projektu s využitím dedikovaného překladového stroje, programování a komunikaci s přípravkem.
3. Navržený systém implementujte a diskutujte jeho vlastnosti a možnosti dalšího rozšíření.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

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

Vedoucí práce: **Vašíček Zdeněk, doc. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 25. října 2019

Abstrakt

Cílem této bakalářské práce je návrh a implementace rozšíření do editoru Visual Studio Code, které umožní správu, editaci, překlad a simulaci FITkit projektů a jejich následné naprogramování do přípravku FITkit bez nutnosti instalace dalšího software. Překlad a simulace projektů je zajištěna dedikovaným serverem obsahující překladový software.

Abstract

This thesis aims to design and implement an extension for Visual Studio Code editor. The extension allows students to edit, build, simulate and manage FITkit projects without the need for any additional software. Dedicated server is used to remotely build and simulate FITkit projects.

Klíčová slova

FITkit, sériová komunikace, vzdálený překlad, Visual Studio Code, rozšíření, TypeScript, WebSocket, Go, Golang, noVNC, XPRA, JWT, programování mikrokontroleru, multiplatformní software

Keywords

FITkit, serial communication, remote building, Visual Studio Code, extension, TypeScript, WebSocket, Go, Golang, noVNC, JWT, microcontroller flashing, cross-platform software

Citace

CHALOUPKA, Jan. *Podpora přípravku FITkit v prostředí Visual Studio Code*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Zdeněk Vašíček, Ph.D.

Podpora přípravku FITkit v prostředí Visual Studio Code

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Ing. Zdeňka Vašíčka, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jan Chaloupka
28. května 2020

Poděkování

Tímto bych rád poděkoval vedoucímu mé práce, panu Doc. Ing. Zdeňku Vašíčkovi, Ph.D za odborné vedení, pomoc a užitečné rady při řešení této práce.

Obsah

Kapitola 1

Úvod

Platforma FITkit se využívá při výuce některých předmětů na Fakultě informačních technologií. Aby si studenti nemuseli do svých počítačů instalovat komplexní vývojářské nástroje, utility a knihovny, jsou v současné době k dispozici obrazy virtuálních strojů. Tyto obrazy jsou dostupné ke stažení všem studentům a obsahují veškerý potřebný software k práci na projektech platformy FITkit. Toto řešení má bohužel několik nevýhod. Především dostupné obrazy virtuálních strojů jsou objemné, jelikož obsahují kromě samotných nástrojů i celý operační systém. Tento problém lze částečně řešit komprimací nebo použitím starších, méně rozsáhlých operačních systémů, ale i poté je jejich velikost v řádech gigabajtů. Pro studenty s pomalým internetovým připojením tak může být problém vůbec tyto obrazy stáhnout. Další z problémů je poměrně velká náročnost na výpočetní výkon a paměť – obzvláště syntéza VHDL kódu může na pomalejších strojích trvat dlouhou dobu.

Tato bakalářská práce popisuje návrh a implementaci alternativního řešení – kombinace lokální správy FITkit projektů v podobě rozšíření pro editor Visual Studio Code a serveru pro vzdálený překlad a simulaci. Celý návrh je vytvořen s ohledem na nezávislost na jedné platformě, aby student nebyl vázán na konkrétní operační systém a pokud možno, aby mohl pracovat na projektu všude, kde lze spustit editor Visual Studio Code. Práce obsahuje také návrh autentizace uživatelů, aby překladový server mohli využívat pouze oprávnění lidé (například studenti fakulty).

Hlavním cílem práce je tedy umožnit studentům práci na FITkit projektu v editoru Visual Studio Code s aktivovaným rozšířením bez nutnosti stahovat nebo instalovat další specializované programy nebo utility.

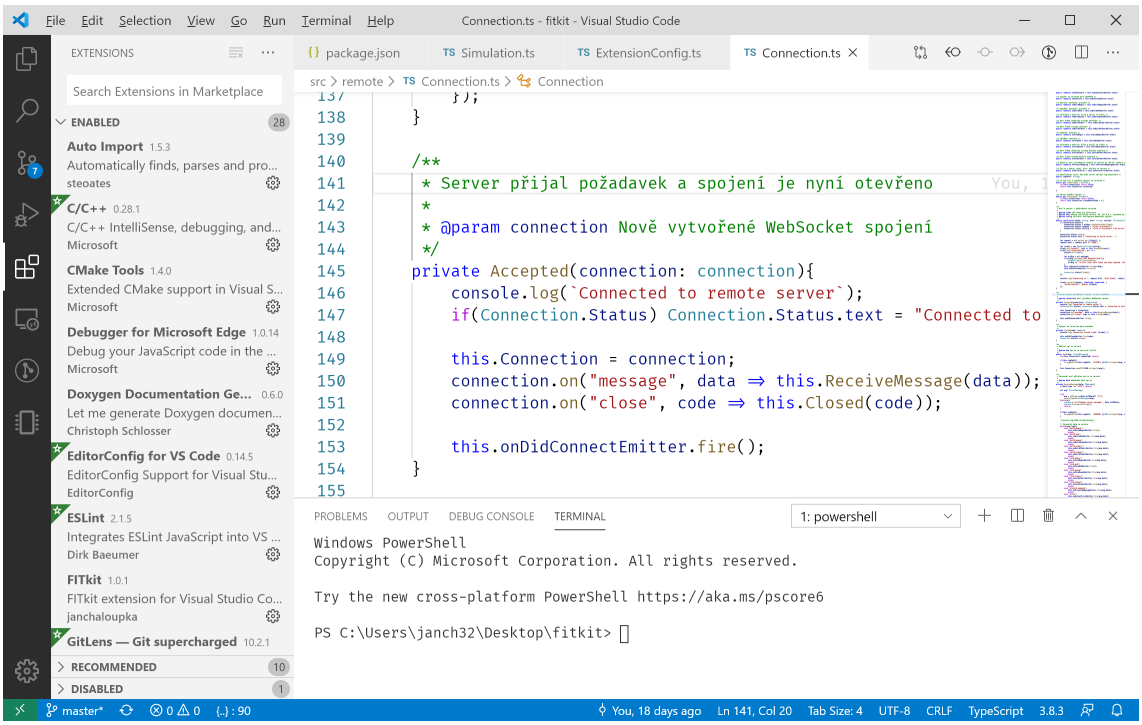
Text práce je členěn do 4 kapitol. První kapitola tvoří teoretickou část práce a věnuje se technologiím použitým ve výsledném řešení. Na tuto kapitolu navazuje návrh řešení. V následující kapitole je popsán způsob implementace navrhovaného řešení. Poslední kapitola popisuje funkčnost v praxi a ověření kompatibility se současným systémem.

Kapitola 2

Teoretická část

V této kapitole jsou postupně popsány tři oblasti, se kterými je důležité se seznámit, aby bylo možné vytvořit a implementovat výsledné řešení. Nejprve je stručně představen editor Visual Studio Code, jeho základní vlastnosti a architektura. V této sekci je rovněž detailněji rozveden princip tvorby rozšíření, pomocí kterých je možné do editoru přidat další funkce. Druhá část je věnována platformě FITkit, zejména existujícímu překladovému systému, správy projektů a způsobu komunikace. Poslední část se zabývá možnostmi realizace vzdáleného přenosu aplikací, především přenosu grafického rozhraní.

2.1 Editor Visual Studio Code



Obrázek 2.1: Editor Visual Studio Code s otevřeným prohlížečem rozšíření (v levém panelu)

Editor Visual Studio Code (dále označovaný jako VS Code nebo VSC) je, stejně jako programovací jazyk TypeScript, open-source (MIT licence) a vyvinutý firmou Microsoft. Jedná se primárně o multiplatformní editor zdrojových kódů. Byť by k tomu mohl název navádět, tento editor nemá žádnou souvislost s vývojovým prostředím Visual Studio od stejné firmy. Visual Studio Code staví od základu na systému Node.js a frameworku Electron. Tento framework využívá vykreslovací jádro z open-source projektu Chromium a umožňuje snadno vytvářet multiplatformní aplikace pouze se znalostí HTML, CSS a jazyku JavaScript. Samotný editor je psaný převážně v jazyce TypeScript, který je detailněji popsán v následující sekci 2.1.2.

Mezi základní funkce editoru patří editace zdrojových kódů s podporou inteligentního doplňování, podpora ladění, sestavení a verzování díky napojení na mnoho populárních verzovacích nástrojů. Hlavní síla editoru ale spočívá v jeho rozšiřitelnosti. Je tak možné instalací dalších modulů (anglicky nazývané „extensions“) přidat například podporu dalších programovacích jazyků, včetně ladění a zobrazování chyb přímo v otevřeném souboru. Moduly jsou snadno dostupné přímo v editoru z integrovaného prohlížeče, kde je lze také vyhledávat a provést na jedno kliknutí instalaci [4]. Při otevření některých souborů může editor přímo doporučit vhodná rozšíření, pokud ještě nejsou nainstalovaná.

V každoročním průzkumu webu StackOverflow uvedlo v roce 2019 přes 50 % dotazovaných vývojářů jako svůj nejoblíbenější editor právě Visual Studio Code [7]. Toto číslo navíc každým rokem stoupá, a právě z tohoto důvodu se v současné době editor nabízí jako ideální kandidát na tvorbu rozšíření k podpoře přípravku FITkit.

2.1.1 Tvorba rozšíření (extensions) pro Visual Studio Code

Editor je již od počátku navržený pro rozšiřitelnost a díky tomu je dostupné rozsáhlé a dobře dokumentované programové rozhraní (dále jako API). Tato dokumentace je dostupná na stránkách editoru, spolu s mnoha návody a užitečnými informacemi, jak rozhraní správně používat[1]. Hlavním jazykem pro tvorbu rozšíření je TypeScript, ale je možné použít i běžný JavaScript.

Rozšíření běží v samostatném procesu a nemá přímo přístup k procesu editoru – veškerá komunikace tak probíhá pouze přes API. Nelze tak například příliš zasahovat do vzhledu a rozložení okna editoru nebo vytvářet vlastní metody rozhraní.

Díky tomu, že editor využívá k běhu systém Node.js, je možné v rozšíření použít všechna rozhraní Node.js. Tím je umožněno například pracovat se souborovým systémem, komunikovat pomocí socketů nebo využít nespočetné množství balíčků, které nabízí ekosystém Node.js (například pomocí správce balíčku NPM).

Příklady možných využití rozšíření [5]:

- Úprava vzhledu editoru změnou barevného schéma a ikon.
- Přidání vlastních prvků do rozložení editoru.
- Zobrazení vlastní webové stránky pomocí `WebView` komponenty.
- Přidání podpory nového programovacího jazyka, včetně našeptávání a syntaktické kontroly.
- Rozšíření ladicích (debug) schopností editoru (napojení na existující ladicí program nebo vytvoření vlastní logiky pro ladění).

[1] Dokumentace API Visual Studio Code: https://code.visualstudio.com/api

Přímo v editoru lze vygenerovat nový projekt rozšíření a dále projekt ladit a testovat, kdy je spuštěna testovací instance editoru s aktivovaným vyvíjeným rozšířením.

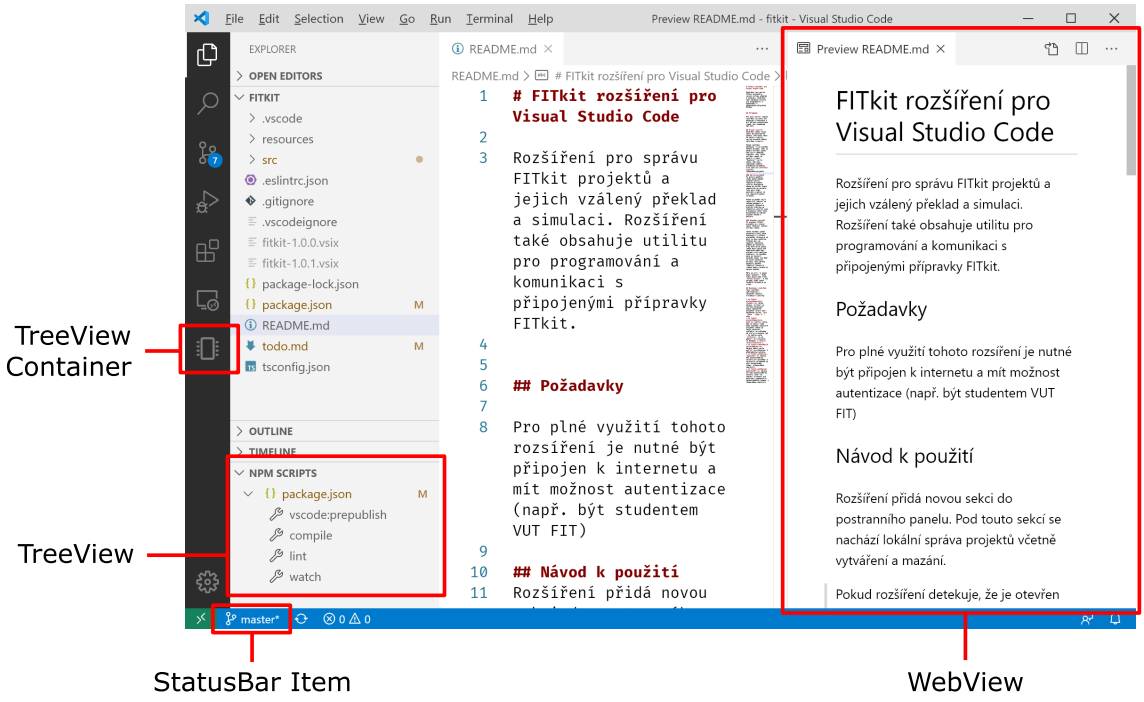
Editor také dokáže dynamicky aktivovat a deaktivovat rozšíření. Každé rozšíření pak definuje, za jakých podmínek se má spustit (například rozšíření pro podporu jazyka C je spuštěno pouze pokud se v projektu nachází zdrojový soubor jazyka C). Toto řešení má výhodu v rychlosti a odezvě editoru, kdy jsou aktivována pouze nutná rozšíření.

Základní schopnosti rozhraní editoru

Rozhraní editoru umožňuje vytvoření nových příkazů, které jsou dostupné ve výchozím nastavení pod klávesou `F1`. Dále je dostupné rozhraní pro zobrazení dialogových oken zobrazující stavové zprávy, vstupní (input) pole pro zadávání textu nebo výběr z několika položek. V neposlední řade je možné vytvořit vlastní okno terminálu, tzv. `Pseudoterminal`[2], kam rozšíření posílá vlastní text, který je zobrazován v editoru, stejně jako výstup ostatních „pravých“ terminálů (včetně podpory escape sekvencí[3]).

Vlastní ovládací prvky

Rozhraní editoru je značně omezené, co se vytváření vlastních ovládacích prvků týče. Neumožňuje přímou editaci a vkládání jednotlivých grafických prvků. Ve svém rozhraní ale nabízí několik způsobům, jak okno editoru rozšířit pomocí několika předdefinovaných komponent. Na obrázku 2.2 jsou vyznačeny jednotlivé typy komponent.



Obrázek 2.2: Popis dostupných míst, kde mohou být přidány vlastní komponenty

[2]Dokumentace rozhraní `Pseudoterminal`: https://code.visualstudio.com/api/references/vscode-api#Pseudoterminal

[3]Escape sekvence je sekvence řídících znaků, které umožňují změnit chování terminálu

Komponenta `TreeView` slouží k zobrazení a interakci s položkami ve stromové struktuře. Aby byl `TreeView` zobrazen, je potřeba jej umístit do kontejneru (`TreeViewContainer`). K tomu lze vybrat existující kontejnery (například kontejner zobrazující soubory aktuálně otevřeného projektu) nebo definovat vlastní kontejner.

K zobrazení informaci o stavu nebo průběhu akce slouží `StatusBarItem`, který je po inicializace umístěn do spodního stavového řádku editoru. Text může doprovázet ikona a lze nadefinovat akci, která se provede po kliknutí na zobrazený text.

WebView

Speciální komponentou je takzvaný `WebView`. V jádru se jedná o rámec sloužící k zobrazení běžné HTML stránky. Jak je v sekci 2.1 popsáno, Visual Studio Code využívá k zobrazení vykreslovací jádro Chromium. Díky tomu tato komponenta podporuje většinu moderních funkcí, které se běžně nachází v plnohodnotných prohlížečích — například WebSocket komunikaci (popsáno v sekci 2.3.1) nebo vykreslování na plátno `canvas`.

I přes to má `WebView` několik omezení. Není zavedena podpora jakéhokoliv typu lokálního úložiště – cookies ani `Web Storage API`[4] není dostupné. Dále není možné využít žádný typ dialogových oken, od jednoduchých vyskakovacích informačních dialogů po přihlašovací formuláře.

Souborový systém

Přístup rozšíření k souborovému systému počítače lze řešit dvěma způsoby – využitím API systému Node.js nebo API editoru. Každý způsob má své výhody i nevýhody. Node.js `fs API`[5] je robustní modul obsahující mnoho metod pro práci se soubory a složkami. Rozhraní editoru staví na tomto API a přidává další užitečné metody, například sledování změny souborů otevřených složek. Bohužel je v jiných ohledech rozhraní editoru limitované a neumožňuje například přidání textu do již existujícího souboru bez přepsání existujícího souboru.

Komunikace s externími procesy

Programové rozhraní editoru neobsahuje žádné metody správy procesů. Pokud rozšíření potřebuje spouštět nebo komunikovat s externími procesy, je možné využít nativní rozhraní systému Node.js, jehož součástí je například modul `child_process`[6], který je k tomu určen.

2.1.2 Jazyk TypeScript

Jazyk TypeScript je open-source programovací jazyk vyvinutý v roce 2012 firmou Microsoft. Jedná se o nadstavbu jazyka JavaScript, do kterého je také překládán. Hlavní vlastnost jazyka je zavedení statické typové kontroly do standardního kódu psaném v jazyce JavaScript. Oproti běžnému JavaScriptu nabízí dále možnost definice výčtů, typů, tříd, rozhraní a generik. Díky statické kontrole typů lze snadno předejít těžko vyhledatelným chybám v přetypování, ke kterém v běžném kódu jazyka JavaScript může dojít [3].

Syntakticky jazyk staví na jazyku JavaScript, inspiruje se ale také jazykem C# – například podporou výše zmíněných generik. JavaScript kód je validní zdrojový kód jazyka

[4] Web Storage API je moderní rozhraní sloužící k dlouhodobému uložení dat ve formátu klíč/hodnota v prohlížeči. https://html.spec.whatwg.org/multipage/webstorage.html

[5] Dokumentace Node.js modulu `fs` - https://nodejs.org/api/fs.html

[6] Dokumentace Node.js modulu `child_process`: https://nodejs.org/api/child_process.html

TypeScript, je tak možné jednoduše začít tento jazyk používat v existujícím projektu bez velkých změn. Zdrojové soubory jsou značeny příponou `.ts` a definiční (hlavičkové) soubory příponou `.d.ts`. Hlavičkové soubory je možné použít v případě, že je volán kód z jazyka JavaScript a je třeba definovat metody a typy tohoto kódu. Hlavičkový soubor se používá především u přeložených knihoven [6].

```
public static async GetActiveConnection(): Promise<Connection>{
    if(this.ActiveConnection?.Connected)
        return this.ActiveConnection;

    return new Connection(
        await Authentication.GetToken(),
        ExtensionConfig.RemoteServerIp
    );
}
```

Výpis 2.1: Ukázka kódu v jazyce TypeScript

2.1.3 Přístup k periferním zařízením

V době psaní textu bohužel neexistuje žádný způsob, jak nativně v editoru komunikovat přes sériové nebo USB rozhraní. Existují komunitní knihovny, které tuto funkcionalitu přidávají. Všechny knihovny ale využívají předkompilované binární soubory, které s aktualizací na novější verzi editoru mohou přestat fungovat. Takové řešení používá například rozšíření pro Arduino[7]. Toto řešení má ale jednu obrovskou nevýhodu, jak je patrné z některých hlášených problému Arduino rozšíření. Při vydání nové verze editoru se může stát, že použité binární knihovny přestanou být funkční a je třeba knihovnu aktualizovat (příklad takového problému je „issue“ číslo 918[8] na GitHub stránce Arduino rozšíření). Toto řešení je dosti nepraktické, co se udržitelnosti týče. Z tohoto důvodu je nutné komunikaci realizovat multiplatformní nativní aplikací, která funguje jako prostředník mezi editorem a zařízením.

Aby aplikaci šlo distribuovat spolu s rozšířením, je nutné zvolit vhodný programovací jazyk, ve kterém lze psát multiplatformní aplikace, komunikovat se sériovým nebo USB rozhraním, a vytvářet spustitelné soubory. Potenciální kandidáti jsou jazyk C# (ve spojení s frameworkem .NET Core) a jazyk Go. Důvodem zúžení výběru právě na tyto jazyky je jejich podpora tzv. cross-compiling, neboli možnosti generovat spustitelné soubory pro více platforem z jednoho systému. Vytvořené binární soubory jazyka C# jsou bohužel příliš velké na to, aby se dali zahrnout v rozšíření. Důvodem je především zahrnutím interpretu v každém spustitelném souboru a tak, i přes veškeré komprimační techniky, i ta nejjednodušší aplikace dosahuje velikosti v řádek desítkách megabajtů. Proto je nejvhodnější vytvořit komunikační utilitu právě v jazyku Go, který je blíže popsán v následující podsekci.

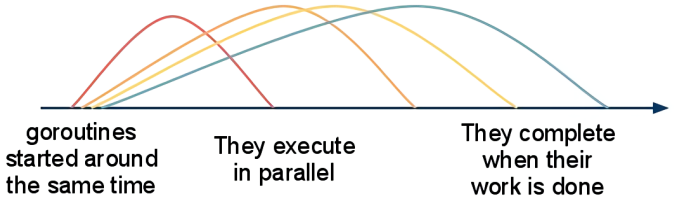
Popis jazyka Go (Golang)

Vyvinut v roce 2009 firmou Google, Go je programovací jazyk překládaný do nativního strojového kódu, který syntakticky vychází především z jazyka C. Jedná se o staticky typovaný jazyk a obsahuje garbage collector zajišťující automatickou správu paměti. Jazyk byl

[7] Repozitář zdrojového kódu rozšíření Arduino: https://github.com/microsoft/vscode-arduino

[8] Jeden z příkladu problému předkompilovaných binárních balíčku při aktualizaci editoru: https://github.com/microsoft/vscode-arduino/issues/918

vyvinut s cílem zjednodušit a zpřehlednit zdrojový kód (oproti jazykům C a C++) a také zefektivnit vícevláknové operace. K tomu se používají tzv. gorutiny (goroutines – vychází z anglického slova „coroutines“). Na gorutinu můžeme nahlížet jako na odlehčenou verzi procesu. Spuštěná funkce v gorutině tak běží nezávisle od hlavního programu, ale režie není zdaleka tak velká, jako v případě běhu funkce v novém procesu. Jazyk dále umožňuje komunikaci mezi gorutinami a ochranu pomocí mutexů při přístupu ke sdíleným prostředkům [8]. V současné době se jazyk používá především u síťových aplikací, kde je vysoká míra paralelizace.



Obrázek 2.3: Vizualizace průběhu gorutin vzhledem k hlavnímu vláknu programu (převzato z http://golangtutorials.blogspot.com/2011/06/goroutines.html)

Další vlastností jazyka a jeho překladače je výše zmíněný cross-compiling. Je tak možné z jednoho stroje, na kterém běží například operační systém Linux, vytvořit spustitelné nativní binární soubory pro MacOS, Windows nebo FreeBSD. Překlad na odlišnou platformu má ale velké omezení — nelze přeložit projekty, které využívají `cgo` – překladač Go projektů obsahující část kódu, která je psaná v jazyce C. Jazyk C se používá především u knihoven, které zpřístupňují nativní funkce systému, které nejsou obsažený v základních knihovnách jazyka Go (například komunikaci s USB zařízeními).

2.2 Platforma FITkit

Platforma FITkit se používá k výuce hardwarových předmětů na fakultě informačních technologii. Jde o zařízení, které obsahuje programovatelný mikrokontroler a hradlové pole (FPGA). Mimo těchto dvou čipů se na zařízení nachází další komponenty a konektory – například displej, klávesnice nebo obecně vstupně výstupní porty (GPIO). V současné době se platforma využívá především pro výuku jazyka VHDL, kterým lze popsat chování integrovaného obvodu.

2.2.1 Správa projektů na platformě FITkit

Projekty platformy FITkit jsou tvořeny ze dvou částí – kódu mikrokontroleru a kódu pro FPGA čip. Zdrojový kód mikrokontroleru je psán v programovacím jazyce C a překládá se pomocí překladače `msp430-gcc` – jde o variantu překladače `gcc` pro rodinu mikrokontrolerů Texas Instruments MSP430. Kód pro FPGA je psaný v jazyce VHDL. Ke správě, syntéze a simulaci VHDL kódu je použito softwarové řešení ISE WebPACK firmy Xilinx.

Všechny projekty, knihovny a další soubory nutné k překladu jsou umístěné v jednom repozitáři. Kopii repozitáře je možné stáhnout ze stránek projektu FITkit. Verzování repozitáře zajišťuje systém Subversion[9]. Základní struktura repozitáře a popis složek je na obrázku 2.4.

[9]Stránky projektu Subversion: https://subversion.apache.org/

/

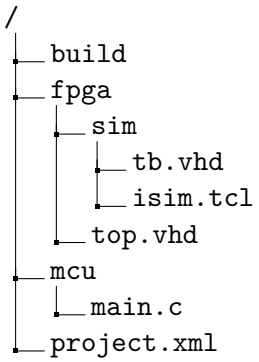
- `apps` ..... Obsahuje všechny projekty rozdělené podle kategorií.
- `fpga` ..... Knihovny pro práci s periferii připojenými k FPGA a definice dostupných architektur.
- `mcu` ..... Knihovny usnadňující práci s komponenty na zařízení. Obsahuje také knihovnu `libfitkit`, která poskytuje základní funkce pro interakci mikrokontroleru s terminálem [9].
- `base` ..... Soubory definující způsob překladu a simulace projektů. Obsahuje soubor `Makefile.inc`, na který odkazuje `Makefile` projektu.

Obrázek 2.4: Základní struktura repozitáře FITkit projektů s popisem jednotlivých složek

Složka `/apps` obsahuje další podsložky, které dělí jednotlivé projekty podle kategorie. V těchto složkách se již nachází přímo složky konkrétních projektů. Příklad cesty k projektu: `/apps/demo/keyboard`, kde `demo` je kategorie a `keyboard` je konkrétní projekt.

Struktura projektu

V kořenové složce projektu se nachází konfigurační XML soubor, standardně pojmenovaný `project.xml`. Tento soubor popisuje projekt, jeho soubory a závislosti na dalších projektech. Soubor je využit při kompilaci, jejíž výstup se ukládá do složky `build` v projektu. Standardně se také používá rozdělení zdrojových kódu do dvou složek podle určení – složka `fpga` obsahuje VHDL soubory FPGA čipu a složka `mcu` soubory jazyka C pro mikrokontroler.



Obrázek 2.5: Příklad struktury FITkit projektu

Konfigurace projektu (`project.xml`)

Soubor `project.xml` se dělí na tři základní části – obecný popis projektu, část s informacemi pro generování kódu MCU (mikrokontroleru) a část s informacemi pro generování kódu FPGA čipu. Obecný popis projektu obsahuje základní údaje, jako jsou název projektu, textový popis, jméno a e-mail autora a číslo revize (datum modifikace ve formátu `YYYYMMDD`). MCU sekce slouží ke specifikaci souborů, které jsou zapotřebí pro překlad kódu mikrokontroleru. Lze také specifikovat závislost na jiném projektu, analyzátor poté rekurzivně projde

tyto odkazované projekty a zahrne do překladu i jejich zdrojové soubory. Cesta může být zadána relativně i absolutně. Při zadání relativní cesty se nejdříve prohledá kořenový adresář projektu, poté adresář `mcu` uvnitř projektu, a nakonec kořenový adresář repozitáře všech projektů. Na podobném principu funguje FPGA sekce, kde se navíc specifikují parametry nastavení FPGA čipu, jako jsou typ architektury, frekvence hodin nebo název hlavní entity [10].

```
<project>
    <name>IVH - Fifteen</name>
    <mcu>
        <file>main.c</file>
    </mcu>
    <fpga dcmfrequency="40MHz" architecture="pc">
        <include>fpga/ctrls/keyboard/package.xml</include>
        <file>top.vhd</file>
    </fpga>
</project>
```

Výpis 2.2: Ukázková konfigurace projektu

Překlad projektu

Při překladu se pracuje s `Makefile`, který je vygenerován z XML souboru popisující projekt `project.xml`[10]. Vygenerovaný `Makefile` soubor obsahuje pouze informace týkající se překladu konkrétního projektu a dále se odkazuje na sadu překladových pravidel umístěných v adresáři `base` repozitáře [10]. Program `make` poté umožňuje použít následující příkazy[11]:

- `make` – Dojde k překladu projektu, pokud se změnily jeho soubory. Výstupem jsou tři vygenerované soubory v adresáři `build` projektu – soubory končící na `_f1xx.hex` a `_f2xx.hex` pro MCU a soubor s koncovkou `.bin` pro FPGA.
- `make load` – Naprogramuje MCA a FPGA čipy připojeného zařízení pomocí programu `fkflash`.
- `make term` – Otevře terminál a začne komunikaci s přípravkem FITkit (tím také dojde ke spuštění kódu v zařízení).
- `make isim` – Spustí nástroj ISIM, který je určený k simulaci VHDL kódů.
- `make clean` – Odstraní soubory vytvořené překladem.

2.2.2 Komunikace a programování přípravku FITkit

Zařízení FITkit obsahuje dva hlavní čipy – mikrokontroler řady MSP430 a hradlové pole FPGA řady Spartan 3. FITkit využívá ke komunikaci s počítačem USB převodník FT2232, který zpřístupňuje hostiteli dvě sériové linky. Na přípravku FITkit je k první sériové lince (kanál A) připojen FPGA čip, přes který lze navázat spojení přímo mezi počítačem a FPGA

[10] K vygenerování `Makefile` se využívá utilita `fcmake`

[11] Uvedeny jsou pouze základní příkazy. Celý výpis je dostupný na adrese https://merlin.fit.vutbr.cz/FITkit/docs/navody/kompilacev2.html

čipem. Druhá sériová linka (kanál B) je připojena k programovacím (`RESET`, `TST`) a datovým (`RxD`, `TxD`) pinům mikrokontroleru. Pomocí tohoto kanálu je možné přeprogramovat FITkit nebo komunikovat s běžícím programem v mikrokontroleru [11]. S programem běžícím v mikrokontroleru probíhá komunikace rychlostí 460800 baudů. Během programování je použito více přenosových rychlostí, přičemž maximální rychlost při použití utility `fkflash` je rovněž 460800 baudů.

Programování

Nahrávání programu do mikrokontroleru se provádí pomocí tzv. boot loaderu, který je uložen v nepřepisovatelné paměti mikrokontroleru a se kterým lze pomocí sériové linky komunikovat a manipulovat tak s obsahem FLASH pamětí nebo spouštět vlastní kód. Nevýhodou vestavěného boot loaderu je jeho rychlost (resp. pomalost), která je omezena na 28400 baud.

K programování se používá utilita `fkflash`, která byla navržena s cílem maximálně urychlit a zpříjemnit programování. Utilita je psaná v jazyce Python a ke komunikaci využívá knihovnu `libkitclient`. Po spuštění utilita vyvolá boot loader mikrokontroleru a pomocí něj nahraje do RAM svůj vlastní optimalizovaný boot loader, který následně spustí. V tuto chvíli se nastaví maximální komunikační rychlost na 460800 baudů a programování je dále zajištěno optimalizovaným boot loaderem [12].

2.3 Sdílený přístup k aplikacím běžícím na serveru

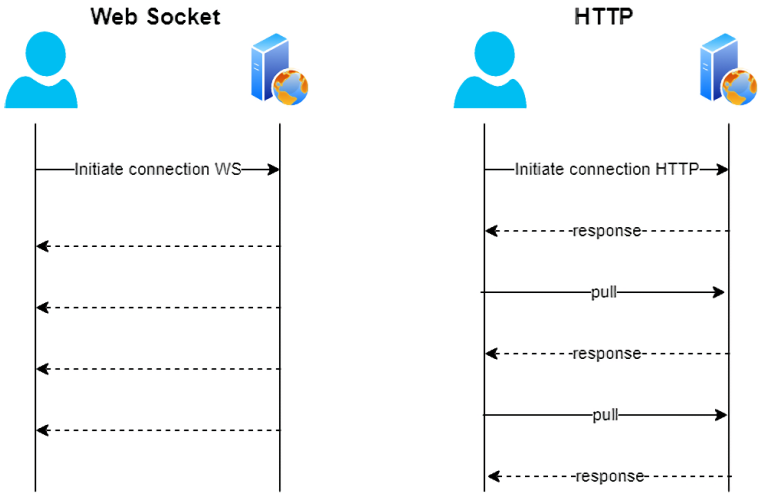
Tato sekce pojednává o různých způsobech přenosu vzdálených aplikací do počítače klienta. Sekce se věnuje přenosu textových i grafických uživatelských rozhraní. Vzhledem k tomu, že se aplikace budou přenášet do rozšíření editoru, je nutné zvolit komunikační mechanismus, který je pro webové technologie (JavaScript, HTML) běžně dostupný. Samotné Node.js (na kterém editor běží) umožňuje komunikaci čistými TCP/UDP sockety. Tento přístup je ale pro naše použití příliš nízko úrovňový a bylo by nutné mnoho věcí vlastnoručně implementovat (autentizaci, kontrolu stavu spojení). K tomu se nabízí použití populárního protokolu WebSocket, který je popsán v navazující sekci a který umožňuje obousměrnou komunikaci klienta se serverem.

2.3.1 Komunikace v reálném čase (WebSocket)

WebSocket je komunikační protokol postavený nad protokolem TCP. Standard RFC6455[12] definuje současnou verzi tohoto protokolu. Úvodní navázání spojení (handshake) je provedeno za pomocí protokolu HTTP, další komunikace poté probíhá ve formátu, na kterém se dohodne klient se serverem (například text, binární nebo JSON formát). Na rozdíl od protokolu HTTP je umožněna obousměrná komunikace klienta se serverem (viz obrázek 2.6 porovnávající tyto protokoly). Vzhledem k použití HTTP v úvodním handshake je tak možné využít vlastnosti HTTP protokolu – například autentizaci uživatele nebo předání původní adresy při použití HTTP proxy.

Při návrhu protokolu byl kladen důraz na co nejnižší režii, a proto se protokol WebSocket hodí pro rychlou výměnu dat. V současné době se jedná o populární protokol, především díky podpoře ve většině současných internetových prohlížečích.

[12] `Standard RFC6455 popisující protokol WebSocket:` https://tools.ietf.org/html/rfc6455



Obrázek 2.6: Porovnání běžné HTTP komunikace s komunikací protokolem WebSocket (převzato z https://ambassadorpatryk.com/2020/03/publish-web-socket-in-the-experience-layer)

2.3.2 Možnosti přenosu obrazu

Díky rozsáhlé podpoře protokolu WebSocket existuje několik projektů, které dokáží zobrazit a ovládat aplikace na vzdáleném počítači v běžném webovém prohlížeči, bez nutnosti instalace specializovaných doplňků. Vzhledem k tomu, že navržené řešení by mělo stavět na open-source technologiích, připadají v úvahu pouze dvě existující varianty open-source řešení přenosu obrazu do webového prohlížeče. O těchto řešeních a porovnání jejich funkcionality pojednává tato sekce.

XPRA

Komunitní multiplatformní projekt XPRA[13] umožňuje spustit aplikace na vzdáleném stroji a přesměrovat grafický výstup spuštěných aplikací zpět do lokálního stroje. Výhodou jsou velké možnosti typů připojení – například SSH, TCP, šifrované SSL spojení, atd.

Navíc obsahuje HTML5 klientskou aplikaci, kterou lze spustit ve většině moderních prohlížečích. Díky tomu lze jednoduše spustit jakoukoliv aplikaci a ovládat ji odkudkoliv z prohlížeče bez nutnosti instalace dalších specializovaných programů. Komunikace může probíhat i šifrovaně a samozřejmostí je integrovaná autentizace uživatelů. Díky autentizaci tak na jednom TCP portu může běžet více sezení, ke kterým jsou uživatelé poté správně směrování podle zadaných přihlašovacích údajů.

noVNC

K přenosu obrazu a ovládání aplikací na vzdáleném počítači slouží také projekt noVNC[14]. Tento projekt je úzce zaměřen na zobrazení a ovládání VNC sezení ve webovém prohlížeči. K přenosu dat využívá protokol WebSocket, který je blíže popsán v kapitole 2.3.1. Samotný noVNC server slouží pouze jako přenosová mezivrstva mezi VNC serverem a prohlížečem, ve kterém je otevřena stránka s noVNC klientem. K použití je tedy vyžadováno,

[13]Stránky projektu XPRA – https://xpra.org/
[14]Stránky projektu noVNC – https://novnc.com/

aby na vzdáleném počítači běžela služba zprostředkovávající VNC připojení (VNC server). Pokud VNC server neumí komunikovat protokolem WebSocket, obsahuje projekt noVNC utilitu **websockify**[15], která slouží jako prostředník a překládá TCP komunikaci do noVNC kompatibilní WebSocket komunikace.

Hlavní využití noVNC je především v ovládání a konfiguraci virtuálních počítačů, kdy je možné zobrazit obrazovku virtualizovaného stroje v prohlížeči bez nutnosti instalace specializovaných VNC klientů.

Rozdíly mezi projekty XPRA a noVNC

Hlavní rozdíl mezi projekty XPRA a noVNC při použití k zobrazení aplikací v internetovém prohlížeči je ve způsobu přenosu obrazu. V případě projektu XPRA se na straně klienta vytvoří virtuální správce oken, do kterého jsou následně vykreslována okna vzdálených aplikací. noVNC využívá k přenosu VNC protokol, který slouží k přenosu celé obrazovky. Proto v případě noVNC běží správce oken na straně vzdáleného počítače a do prohlížeče je přenášen obraz celého správce oken.

Bohužel bez experimentálního ověření není jednoduché předem určit, který přístup je vhodnější. Ačkoliv by se projekt XPRA mohl na první pohled zdát jako jednoznačná volba k přenosu aplikací, pro účely této práce se jako vhodnější ukázalo použití noVNC (detailní důvod výběru tohoto projektu viz sekce 3.2.3), především z důvodu nižší latence a nižších nároků na kapacitu přenosové linky.

2.3.3 Izolace aplikací (sandboxing)

Na serveru poběží překladové a simulační aplikace, se kterými bude moci uživatel interagovat. Aby uživatelé nemohli skrze spuštěnou aplikaci provádět změny na serveru nebo procházet cizí soubory, je nutné spuštěné aplikace „sandboxovat“, neboli omezit jejich přístup pouze na nejnutnější soubory a funkce systému. Pro operační systém Linux existuje mnoho aplikací, které slouží k sandboxování aplikací. Pro naše použití je nutné vybrat pouze open-source aplikace, které nevyžadují ke spuštění a běhu administrátorská práva (root přístup). Tato kritéria zúžili výběr na dva projekty, a to aplikaci Mbox[16] a Firejail[17].

Mbox je jednoduchá aplikace, která umožňuje sandboxing aplikací, vytvoření dočasného souborového systému nebo omezení síťové komunikace. Bohužel v současné době již není aktivně vyvíjena, a tak ji pro nový projekt není vhodné použít, obzvláště v případě existence podobné aplikace, která je populárnější a stále vyvíjená.

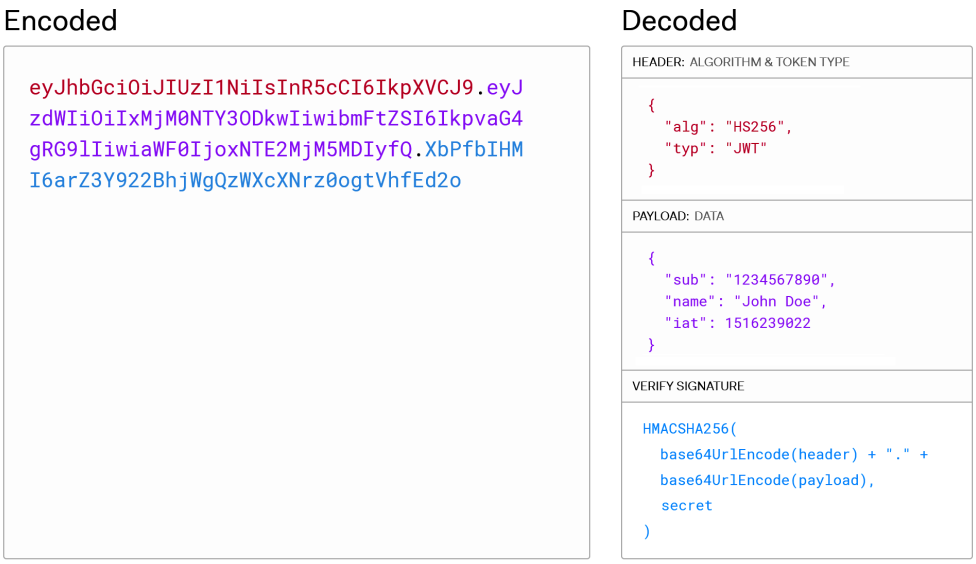
Komunitní aplikace **Firejail**, podobně jako v případě Mbox, dokáže omezit spuštěným programům přístup k souborovému systému, procesům nebo síťovým zařízením. Utilita je psaná v jazyce C a je kompatibilní s operačním systémem Linux postaveném na jádru 3.x nebo novějším. Díky využití bezpečnostních funkcí dostupných přímo v jádru operačního systému (především funkce Linux Namespaces[18]) má sandboxování minimální dopad na výkon. Utilita je vůči spuštěnému programu transparentní, je tak možné spustit jakýkoliv typ aplikace, včetně grafických [2].

[15] Repozitář projektu Websockify (noVNC) – https://github.com/novnc/websockify
[16] Stránky projektu Mbox – http://pdos.csail.mit.edu/mbox/
[17] Stránky projektu Firejail – https://firejail.wordpress.com/
[18] Linux Namespaces – https://lwn.net/Articles/531114/

2.3.4 Možnosti autentizace (JWT)

JSON Web Token (zkráceně JWT) je otevřený standard (RFC 7519[19]), který definuje způsob bezpečného přenosu informací mezi subjekty ve formátu JSON. Hlavní metodou zabezpečení je digitální podepsání přenášených informací, kterým příjemce ověří, že zpráva nebyla modifikovaná. Příjemci k ověření stačí ověřovací klíč a nepotřebuje nijak komunikovat s odesílatelem.

Tento standard je nejčastěji používán u autentizace uživatele. Po přihlášení je uživateli vygenerován autentizační JWT token, který je digitálně podepsán a který obsahuje informace o přihlášeném uživateli. Uživatel poté při každém požadavku předá svůj token serveru, který tento token ověří a následně vyhoví požadavku uživatele.



Obrázek 2.7: Ukázka JWT a vizualizace uložených dat v online nástroji `https://jwt.io/`

Struktura

Data ve formátu JWT se skládají ze tří částí (hlavička a data jsou JSON objekty):

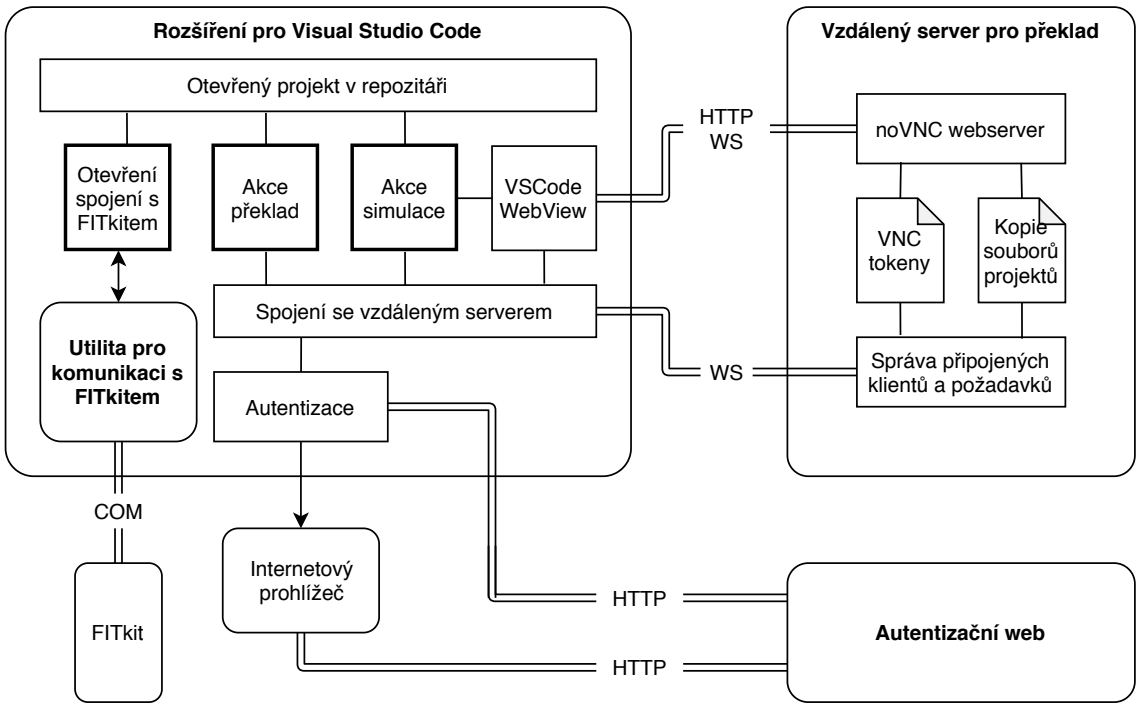
- Hlavička (Header)
- Samotná data (Payload)
- Podpis (Signature)

Hlavička obsahuje základní informace o formátu a použitém algoritmu podpisu. Po hlavičce následuje sekce **data**, kde se vyskytují libovolná data, která jsou digitálně podepsána. Standard definuje několik základních hodnot – například název vydavatele tokenu (`"iss"` – issuer), čas expirace tokenu (`"exp"` — expiration time) nebo název subjektu, pro který byl token vygenerován (`"sub"` — subject). Poslední částí je **podpis** dat, která je použita k ověření integrity dat. Podpis dat může být v různých formátech, závisející na zvoleném typu algoritmu [1]. Při přenosu jsou jednotlivé části převedeny do formátu Base64 a následně spojeny tečkou do jednoho řetězce viz obrázek 2.7.

[19]Standard RFC 7529 popisující JSON Web Token – `https://tools.ietf.org/html/rfc7519`

Kapitola 3

Návrh řešení



Obrázek 3.1: Diagram zobrazující propojení a funkčnost jednotlivých částí návrhu. Hlavní části jsou označeny tučným textem.

V této kapitole je popsán návrh všech částí práce. Výsledný návrh je vypracován s ohledem na tři základní požadavky. Řešení by mělo být jednoduše použitelné studenty bez ohledu na jejich operační systém – editor Visual Studio Code je multiplatformní a rozšíření by mělo být možné spustit a používat všude, kde lze použít tento editor. Nemělo by být nutné instalovat další programy nebo knihovny – pouhé stažení a aktivace rozšíření by mělo být vše, co je potřeba k práci a testování FITkit projektů (a připojení k internetu, pro překlad a simulaci). Dále by měla být zachována co největší kompatibilita se současnou správou FITkit projektů, aby šlo používat současné (qDevKit) a navrhované alternativní řešení zároveň bez velkých komplikací. Přístup k funkcím překladového serveru by měl být dostupný pouze oprávněným osobám (v tomto případě studentům), proto je jako součást

práce navrhnut i způsob autentizace uživatelů. Návrh je rozdělen na čtyři funkční celky, které jsou v této kapitole dále specifikovány:

- Samotné rozšíření editoru Visual Studio Code, které bude určeno ke správě a editaci lokálních projektů. Kromě toho komunikuje s ostatními částmi projektu při vzdálené simulaci a překladu.
- Utilita zprostředkující komunikaci mezi zařízením FITkit a rozšířením
- Vzdálený server s programem obsluhující požadavky na vzdálený překlad a simulaci FITkit projektů.
- Webová stránka pro autentizaci uživatelů a generování přístupových tokenů určených k připojení na překladový server.

Na obrázku 3.1 je přehledně zobrazeno propojení a funkčnost všech částí.

3.1 Rozšíření pro Visual Studio Code

Rozšíření editoru je hlavní část práce. Zpřístupňuje uživateli správu projektů, editaci otevřeného projektu, překlad a simulaci projektu s využitím překladového serveru a komunikaci s připojeným zařízením FITkit.

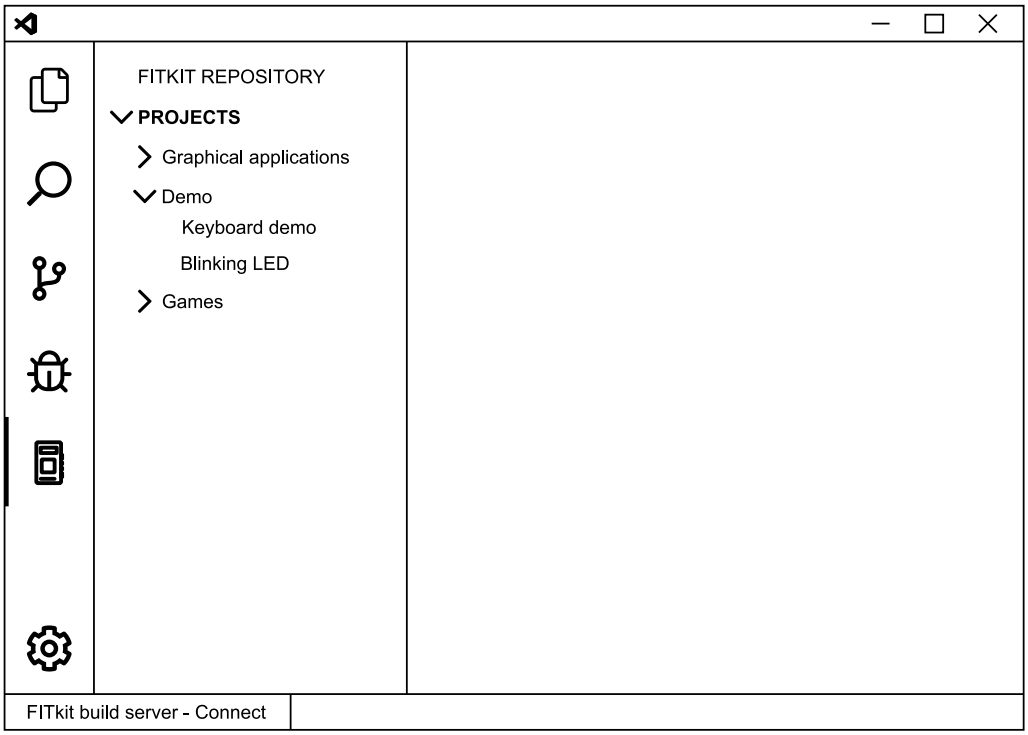
3.1.1 Správa projektů

Jedním z cílů je zachování kompatibility se současným způsobem práce s projekty v programu qDevKit. Proto je zachován současný způsob a struktura uložení projektu, kdy se projekty nachází v repozitáři `FITkitSVN` spolu s dalšími knihovnami FPGA a MCU části. Více informací o způsobu uložení projektů v repozitáři se nachází v sekci 2.2.1. Pro správu projektů přidá rozšíření po inicializaci do editoru nový panel, kde jsou vypsány všechny projekty, které se nachází v lokální kopii repozitáře. Projekty jsou ve stromové struktuře uspořádány do kategorii, podobným způsobem, jako je tomu v programu qDevKit. Obrázek 3.2 zobrazuje návrh tohoto panelu a způsob organizace projektů.

Vytvoření (stažení) kopie repozitáře

Po instalaci je nutné vytvořit lokální kopii repozitáře. Ve výchozím stavu se bude kopie repozitáře nacházet ve složce `fitkitsvn` v adresáři aktuálního uživatele (například ve Windows `C:/Users/{user}/fitkitsvn` nebo v OS Linux `/home/{user}/fitkitsvn`). Pokud tato složka nebude existovat nebo nebude mít správnou strukturu (jak vypadá struktura repozitáře je popsáno v sekci 2.2.1), je uživateli nabídnuto stažení této složky. Potvrzením se automaticky na pozadí stáhne komprimovaná kopie ze vzdáleného serveru (například ze stránek projektu FITkit) a tento archív je poté rozbalen do cílové složky. Uživatel je průběžně informován o stavu této operace.

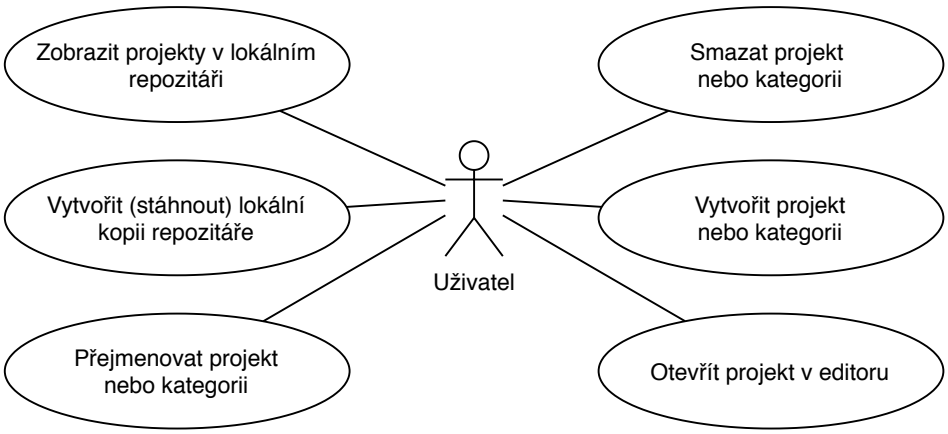
V nastavení editoru lze změnit jak umístění lokální kopie repozitáře, tak internetovou adresu, ze které je komprimovaná kopie repozitáře stažena.



Obrázek 3.2: Návrh rozložení panelu správy FITkit projektů ve Visual Studio Code.

Operace s projekty

Se zobrazenými kategoriemi a projekty je možné provádět různé operace. Všechny operace, které uživatel může s projekty v repozitáři provádět jsou přehledně zobrazeny na obrázku 3.3.



Obrázek 3.3: Diagram případů užití zobrazující práci uživatele s projekty.

Mezi základní operace patří **přejmenování** projektu nebo kategorie, **otevření v systémovém prohlížeči souborů** (průzkumníku) nebo **smazání**. V případě smazání je nutné se uživatele před provedením akce dotázat, zda tuto akci chce opravdu provést a zda chápe důsledek této akce. To znamená jasně vysvětlit uživateli (hlavně v případě mazání kategorie), že jde o destruktivní akci, která odstraní veškeré soubory uvnitř tohoto projektu

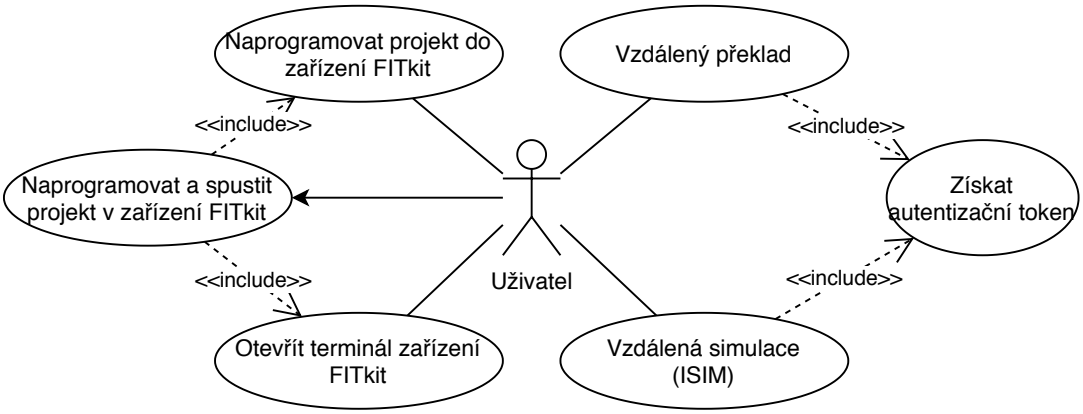
(nebo kategorie). Tímto se předejde nechtěnému smazání důležitých souborů, které by se poté velice špatně obnovovali.

Každý zobrazený projekt je možné dvojklikem (alternativně volbou v kontextové nabídce) **otevřít v editoru**. Touto akci se složka projektu vybere jako aktuální otevřený projekt editoru a je následně možné editovat zdrojové soubory.

K pokročilejším operacím patří **vytváření nových kategorií a projektů**. Při vytváření nové položky (projektu nebo kategorie) je nutné zadat název položky, který se bude zobrazovat a také název složky, která bude tuto položku v repozitáři reprezentovat. Pro usnadnění je název složky vygenerován ze zadaného „Pěkného“ názvu položky. Příklad: uživatel zadá název nové kategorie „Grafické aplikace“, jako název složky je vygenerován text `graficke_aplikace`. Uživatel tak pouze tento název potvrdí nebo přepíše svým vlastním. Při vytváření projektu je kromě názvu nutné zadat autora a volitelně i popis projektu. Jméno autora se rozšíření pokusí předvyplnit jménem aktuálně přihlášeného uživatele v operačním systému. Jde pouze o předvyplnění vstupního pole, je možné zadat jakékoliv jiné jméno. Po zadání všech dat se vygeneruje ve specifikované složce konfigurační soubor, který obsahuje všechny vyplněné údaje – v případě kategorie se jedná o soubor s názvem `description.xml`, u projektu jde o soubor `project.xml`.

3.1.2 Práce s otevřeným projektem

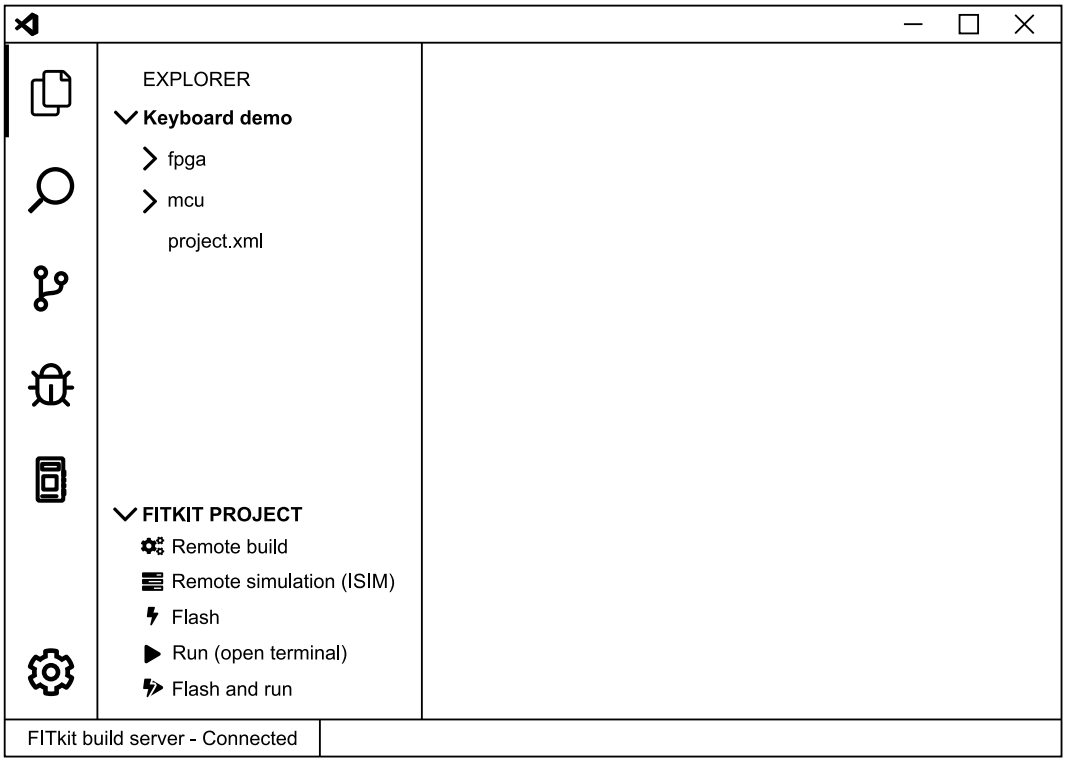
Pokud bude v editoru otevřená složka FITkit projektu, rozšíření aktivuje několik funkcí. Přítomnost FITkit projektu je detekována podle struktury otevřené složky. Především, zda se otevřená složka nachází uvnitř adresáře repozitáře a zda je v kořenu složky umístěn konfigurační soubor projektu `project.xml`. V případě, že projekt nebude detekován, zůstane zobrazena pouze ikona repozitáře v levém sloupci, která zobrazuje seznam lokálních projektů (jak je popsáno v sekci 3.1.1). Detekce je zavedena proto, aby uživatel nebyl zahlcen zbytečnými funkcemi, rozhodne-li se pracovat v editoru na jiném typu projektu nebo editovat jiné soubory.



Obrázek 3.4: Diagram případů užití zobrazující práci uživatele s otevřeným projektem.

Při otevřeném projektu je na panel „explore“ přidána sekce s tlačítky zpřístupňující všechny proveditelné akce. Panel „explore“ je hlavní panel, ve kterém jsou vypsány soubory otevřené složky. Návrh dostupných akcí je přehledněji uveden na obrázku 3.4. Funkčnost jednotlivých akcí je popsána v následujících podsekcích. Všechny akce jsou dostupné jako příkaz, pokud uživatel preferuje ovládání editoru klávesnici. Panel příkazů se ve standardním nastavení editoru otevírá klávesou `F1`. Všechny příkazy obsahují prefix „FITkit“. Díky

tomu lze jednoduše příkazy vyhledat a rozlišit od příkazů editoru a ostatních rozšíření. Na stavový řádek je přidána informace o stavu připojení na překladový server. Kliknutím na tento stav se spojení ukončí. Spojení běžně zůstává otevřené po dokončení překladu nebo simulace, aby nebylo nutné při každém požadavku na server stále provádět autentizaci. Celkový návrh rozložení prvků uživatelského rozhraní při práci s projektem je vyobrazen na obrázku 3.5.



Obrázek 3.5: Návrh panelu akcí otevřeného FITkit projektu ve Visual Studio Code.

Autentizace

Aby se rozšíření mohlo připojit k překladovému serveru, je nutné při navázání spojení odeslat autentizační token. Způsob autentizace je podrobněji popsán v sekci 3.4. Aplikační rozhraní editoru poskytuje přístup k lokálnímu úložišti, kam rozšíření tento token ukládá. Pokud token není dostupný, rozšíření vyzve uživatele k autentizaci. Potvrzením této výzvy začne proces autentizace. Prvně je zaslán na rozhraní autentizačního webu požadavek na nový token. Autentizační web odpoví ID nového požadavku. Po přijetí tohoto identifikátoru je ve výchozím webovém prohlížeči otevřena stránka, kde uživatel po přihlášení potvrdí vytvoření tokenu. Hlavní důvod otevření externího prohlížeče je kvůli transparentnosti celého procesu. Uživatel nemusí mít obavy, že zadává své přihlašovací údaje do neznámé aplikace a může si v prohlížeči ověřit, na kterou webovou stránku se skutečné přihlašuje a zda je toto spojení zabezpečené. Mezitím se rozšíření periodicky dotazuje aplikačního rozhraní autentizační stránky, zda již byl token vygenerován. Pokud token existuje, rozšíření ho získá a uloží do lokálního úložiště pro další použití. Adresy URL pro komunikaci s autentizačním rozhraním je možné změnit v nastavení rozšíření v případě náhlé změny adres těchto stránek.

Vzdálený překlad

Aby nebylo nutné instalovat celý softwarový balík Xilinx WebPack, lze otevřený projekt přeložit na vzdáleném serveru. K překladu tak stačí přístup na internet a validní autentizační token (jak rozšíření získá token je popsáno v podsekci 3.1.2 Autentizace). Po spuštění akce vzdáleného překladu je analyzován soubor `project.xml` v kořenovém adresáři projektu. Tím je získána konfigurace projektu, seznam zdrojových souborů a závislostí na ostatních projektech.

Struktura konfiguračního souboru je popsána v sekci 2.2.1. Každý konfigurační soubor může tagem `<include>` uvést závislost na jiných knihovnách a projektech. Pro načtení všech zdrojových souborů je nutné rekurzivně analyzovat i odkazované konfigurační soubory projektů. U syntézy VHDL kódu záleží na pořadí souborů, a proto je nutné zachovat pořadí souborů a závislostí tak, jak jsou uvedené v konfiguraci. Pokud je na některý zdrojový soubor odkazováno více než jednou, bere se v potaz pouze první výskyt a ostatní jsou přeskočeny.

U zdrojových souborů mikrokontroleru je problém s hlavičkovými soubory. Tyto soubory nejsou běžně uvedeny v konfiguraci projektu, ale i tak jsou pro překlad důležité. Ideálním řešením by bylo použít utilitu, která by analyzovala zdrojový `.c` soubor a vrátila seznam odkazovaných hlavičkových souborů (například program `gcc` s vhodně zvolenými argumenty). Kvůli požadavku na přenositelnost kódu a nezávislost na jedné platformě by bylo nutné spolu s rozšířením distribuovat více verzí spustitelných souborů této utility pro různé platformy, což by bylo problematické, co se týče udržitelnosti a velikosti výsledného instalačního balíčku. Na místo toho je použita prostá, avšak přesto funkční alternativa. Při každém nalezeném `.c` souboru jsou ze stejné složky načteny i všechny hlavičkové `.h` soubory. Toto řešení bylo otestováno na prototypu načítání souborů a funguje se všemi ukázkovými projekty v repozitáři.

Po shromáždění všech důležitých souborů jsou (spolu s konfigurací projektu) odeslány v jednom požadavku na překladový server. Formát a způsob odeslání je popsán v sekci 3.2.4 zabývajícím se aplikačním rozhraním překladového serveru. Pokud spojení s překladovým serverem není otevřeno, rozšíření se pokusí spojení navázat. Výchozí adresa překladového serveru je daná v manifestu rozšíření, lze ji však v nastavení změnit (například v případě, kdy si uživatel vytvoří vlastní instanci překladového serveru). Při požadavku na připojení je odeslán i autentizační token. Pokud token není validní, server spojení odmítne. Odesláním požadavku začne proces překladu, v editoru se vytvoří virtuální terminál, kde bude vypisován průběh překladu. Po skončení překladu server odešle vytvořené soubory a rozšíření je uloží do složky `build` v projektu. Server nemusí vždy po skončení překladu odeslat soubory. Žádný soubor není odeslán například pokud dojde během překladu chyba. Ukončením překladu se virtuální terminál deaktivuje a uživatel ho může uzavřít.

Vzdálená simulace

Ladění VHDL kódu probíhá pomocí simulace. V programovém balíku Xilinx WebPack je k tomu určen program ISIM. Podpora simulace není uvedena v zadání práce, ale přesto jsem ji zahrnul v návrhu, neboť se dle mého názoru jedná o nezbytný nástroj při práci na FITkit projektech, ať už pro testování kódu (pokud není k dispozici zařízení FITkit), nebo při hledání chyb. K simulaci FITkit projektu je nutné, aby projekt obsahoval alespoň jeden tzv. „testbench“ soubor a konfigurační soubor ISIM simulace. Konfigurační soubor simulace má pevně danou cestu v projektu, a to `fpga/sim/isim.tcl`. Testbench soubory můžou být uvedené v `project.xml` nebo uloženy ve složce `fpga/sim`. Před spuštěním simulace je nutné

zkontrolovat, zda tato složka existuje. Pokud složka existuje, všechny `.vhd` soubory se označí jako soubory nutné k simulaci projektu.

Po spuštění této akce je průběh z počátku stejný jako v případě vzdáleného překladu, viz sekce 3.1.2 Vzdálený překlad – provede se shromáždění zdrojových souborů, konfigurace projektu, a vše se jako požadavek na vzdálenou simulaci odešle překladovému serveru. Server po zpracování požadavku odpoví odkazem na webovou stránku. Rozšíření vytvoří WebView panel (popis komponenty WebView viz sekce 2.1.1), který se zobrazí v hlavní části editoru. Uvnitř panelu se provede navigace na stránku získanou od serveru, kde se zobrazí uživatelské rozhraní vzdáleného programu ISIM, se kterým je možné běžným způsobem interagovat. Ukončením vzdálené aplikace nebo uzavřením WebView panelu se proces simulace ukončí.

Komunikace se zařízením FITkit

Rozšíření zpřístupňuje přímo v editoru programování a komunikaci s připojeným zařízením FITkit. Ke spojení je použita utilita, která obsahuje veškerou funkcionalitu, co se komunikace a programování týče. Návrh a důvod použití této utility je popsán v sekci 3.3. Předpokládá se, že většina uživatelů bude používat pouze jeden FITkit, a tak je využita funkce utility, kdy se naváže spojení s prvním nalezeným zařízením FITkit.

V editoru jsou k dispozici tři akce spojené s FITkit komunikací. První akce je **otevření terminálu a aktivace nahraného programu** v zařízení FITkit. Rozšíření spustí utilitu jako externí proces. Standardní vstup a výstup je zobrazen v terminálu editoru, kde je možné se zařízením dále komunikovat. Další z akcí je **nahrání programu** do zařízení. K tomu je zapotřebí složka `build` v adresáři projektu, obsahující přeložené soubory pro MCU a FPGA (tyto soubory jsou získány například ze vzdáleného překladu). Rozšíření spustí utilitu v režimu programování, předá cesty k přeloženým souborům a průběh opět zobrazí v terminálu editoru. Poslední akcí je kombinace předchozích dvou akcí – **nahrání a spuštění projektu**. Jde o zjednodušení práce uživateli, aby nebylo nutné projekt pokaždé nahrát, vyčkat, a poté otevřít terminál. Tato akce nahraje nový program a pokud vše proběhlo v pořádku, tak je nahraný program ve stejném terminálu rovnou spuštěn.

3.2 Překladový server

Pojmem překladový server označuje v tomto případě vzdálený server, na kterém jsou nainstalované všechny důležité nástroje na překlad a simulaci projektů, a další programy obsluhující příchozí požadavky. Návrh se zakládá na použití operačního systému Linux, který je běžně používaný v prostředí serverů. V návrhu se pracuje s kombinací existujících programů a programů vytvořených speciálně v této práci.

Na serveru tedy musí být nainstalován balík programů Xilinx WebPack, sloužící k překladu a simulaci VHDL části projektu. Překlad MCU části řeší program MSP430-GCC. Více o překladu FITkit projektů pojednává sekce 2.2.1 Překlad projektu. Oba produkty jsou k dispozici pro operační systém Linux a neměl by tak být problém s instalací. Požadavky klientů spravuje vlastní program, jehož vlastnosti jsou popsány v sekci 3.2.1. Výstup simulace je přenášena kombinací VNC serveru s projektem noVNC (viz sekce 2.3.2). Důvod použití a způsob přenosu obrazu je popsán dále v sekci 3.2.3.

3.2.1 Správa klientů a požadavků

Správu klientů zajišťuje speciálně vytvořený program, který běží stále jako systémová služba. Jde o centrální prvek řídící veškeré dění na překladovém serveru. Program naslouchá na síťovém portu a lze s ním navázat spojení protokolem WebSocket (popis tohoto protokolu viz sekce 2.3.1). Při otevření nového spojení je provedena autentizace. Součástí autentizace je kontrola přítomnosti JWT řetězce (více o JWT viz sekce 2.3.4) v hlavičce, validace jeho digitálního podpisu veřejným klíčem a kontrola platnosti klíče. Program udržuje se všemi připojenými klienty aktivní spojení, dokud jej klient neukončí nebo není přerušeno externími vlivy. Dále program spravuje lokální úložiště dočasných složek FITkit projektů. Jednotlivé složky projektu mají dobu života danou délkou trvání akce (po dokončení překladu nebo simulace je složka smazána). V jednotlivých složkách projektů jsou uloženy soubory přijaté při žádosti klienta o překlad nebo simulaci. Spolu s uložením přijatých souborů se vytvoří i další soubory nutné k provedení akce. Které složky jsou vytvořeny a jak překlad nebo simulace probíhá je podrobněji popsáno v navazující sekci.

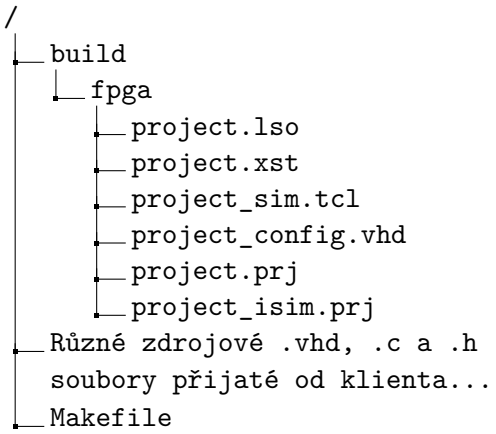
Překlad a simulace, obzvláště VHDL kódu, jsou náročné na výkon serveru. Aby se předešlo přetížení celého serveru, umožňuje program nastavit **limit současně běžících překladů a simulací**. Limit se odvíjí od dostupných prostředcích serveru, proto lze hodnotu upravit individuálně pro každý server. Použití funkce fronty není vyžadováno, administrátor serveru může nastavit neomezený počet požadavků a frontu tak efektivně zrušit. Pokud je limitu dosaženo a přijde další požadavek, je řazen na konec fronty čekajících požadavků. Klient je o tomto faktu informován stavovou zprávou. Je také informován o každé změně, která ve frontě nastane. Uživatel si díky tomu může udělat představu o době, kterou ve frontě stráví. Klient může kdykoliv frontu opustit (zrušit požadavek). Tím také přijde o svoji pozici ve frontě, rozhodne-li se požadavek opakovat (požadavek se opět zařadí na konec fronty). Hned jakmile dojde k dokončení některé z probíhající akcí, je spuštěn první požadavek čekající ve frontě. Obě akce, simulace a překlad, mají oddělenou frontu – uživatele čekající na překlad nemusí čekat na dokončení simulace (a naopak).

3.2.2 Překlad a uložení souborů projektu

Překlad projektů je prováděn podobně jako v případě překladu programem qDevKit. Ve složce s projektem je vygenerován lokální `Makefile` projektu obsahující seznam zdrojových souborů. Lokální `Makefile` projektu uvnitř odkazuje na globální `Makefile.inc` soubor, který definuje způsob překladu zdrojových souborů. Návrh používá s několika úpravami současnou podobu tohoto souboru, který je součástí repozitáře FITkit projektů. Mezi hlavní úpravy patří změna cest k překladovým programům a odstranění nevyužitých částí (například spuštění simulace v programu ModelSIM). Kromě `Makefile` je v adresáři projektu vytvořena složka `build` určená k ukládání všech vygenerovaných a dalších souborů důležitých ke spuštění překladu. Struktura a seznam těchto souborů je zobrazen na obrázku 3.6. Soubory jsou generovány stejným způsobem jako při použití utility `fcmake`. Při návrhu bylo čerpáno ze zdrojového kódu této utility, který je dostupný v SVN repozitáři na serveru FITkit projektu[1].

Soubor s koncovkou `.lso` obsahuje název hlavní knihovny. Soubor s koncovkou `.xst` definuje parametry syntézy, jako jsou úroveň optimalizace, cílový FPGA čip a název hlavní entity. Simulace programem ISIM načítá soubor s koncovkou `.tcl`, jež obsahuje cesty k simulovaným souborům. Speciální případ je soubor s názvem končícím na `_config.vhd`, který

[1] Zdrojové soubory `fcmake` – https://merlin.fit.vutbr.cz/svn/FITkitUtils/trunk/fcmake/



Obrázek 3.6: Příklad struktury složky projektu před překladem

obsahuje standardní VHDL kód. Tento soubor je vygenerován pouze pokud má projekt specifikovanou architekturu a určuje frekvenci, na které obvod v FPGA čipu poběží. Nakonec složka obsahuje dva `.prj` soubory, jeden pro syntézu a jeden pro simulaci. Oba soubory obsahují seznam zdrojových VHDL souborů projektu.

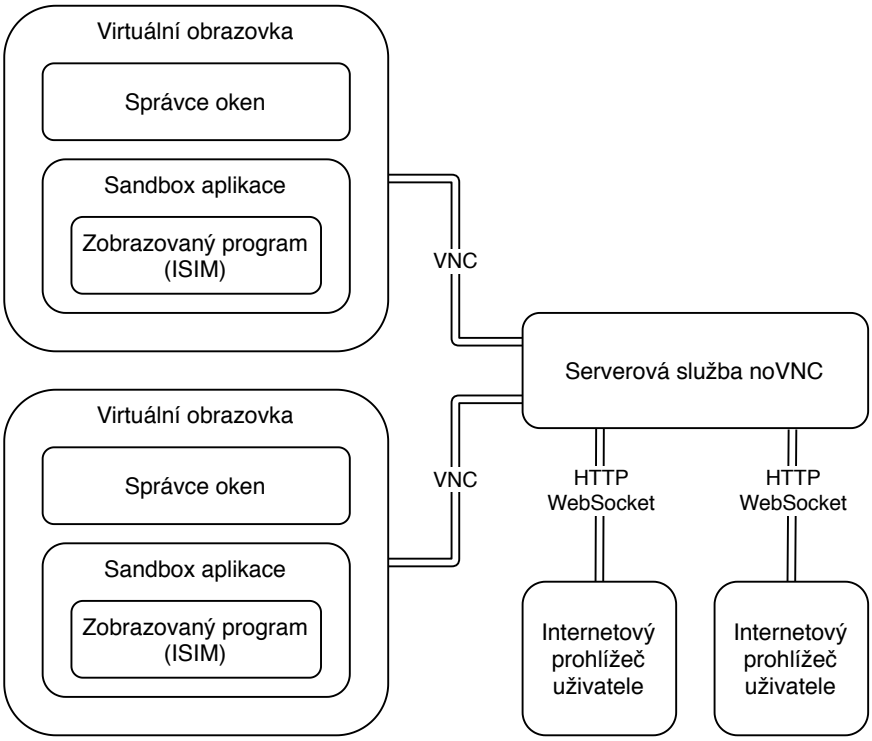
Po vygenerování všech souborů je programem spravující požadavky spuštěn v novém procesu program `make`, který pracuje v dočasném adresáři projektu. Tímto se spustí překlad MCU i FPGA části projektu. Konzolový výstup programu je přeposílán klientovi, který jej může zobrazit uživateli. Pokud proces skončí úspěšně, jsou ze složky `build` načteny dva soubory pro MCU a jeden binární soubor pro FPGA čip. Obsah souborů je odeslán klientovi. Nezávisle na úspěšnosti překladu klient dostane zprávu o dokončení. Zpráva obsahuje stavový kód vrácený programem `make` a výsledné soubory, pokud byly vytvořeny.

3.2.3 Simulace a přenos grafických programů

Jak již bylo zmíněno, k simulaci VHDL kódu je použit program ISIM. Jde o grafický program, který simuluje projekt a zobrazuje stav signálů v čase. Je třeba zajistit, aby uživatel měl přístup k tomuto grafickému rozhraní. Tímto problémem se zabývá sekce 2.3.2. Každý uživatel by měl mít možnost zobrazit simulační okno aplikace bez nutnosti instalace speciálních programů. Proto je cílem zobrazovat rozhraní aplikace v internetovém prohlížeči. Tuto technologii umí dva open source projekty – XPRA a noVNC. Projekt XPRA je k tomuto účelu vhodnější, neboť obsahuje vlastní správce oken a je k zobrazování aplikací přímo určen. Bohužel při testování potenciálního návrhu postaveném na projektu XPRA bylo zjištěno, že způsob přenosu obrazu není zcela kompatibilní se způsobem vykreslování programu ISIM a přenášený obraz vykazoval velkou latenci bez ohledu na nastavení komprese a rychlosti linky. Z toho důvodu je návrh založený na projektu noVNC, který přenáší obraz protokolu VNC do internetového prohlížeče (běžně se využívá například ke vzdálené správě serverů). Při testování řešení s využitím noVNC bylo dosaženo mnohem nižší latence. Samotný projekt noVNC ale pouze přenáší obraz protokolu VNC do prohlížeče. Dále je nutný program, který vytvoří virtuální obrazovku a jehož výstupem je obraz v protokolu VNC. Výstup programu se poté předá běžící službě noVNC serveru. Uvnitř virtuální obrazovky je spuštěn správce oken a samotná aplikace. Toto je nutné provést s každou novou instancí vzdáleného překladu. Pro lepší představu je na obrázku 3.7 zobrazena struktura spuštěných

programu v případě přenosu protokolem noVNC. Návrh lze snadno využít k přenosu obrazu jakékoliv aplikace, pokud tomu v budoucnu bude třeba.

Požadavek klienta na spuštění simulace má na počátku stejný průběh jako požadavek na překlad (viz sekce 3.2.2) – uloží se přijaté zdrojové soubory do dočasné složky projektu, vytvoří se konfigurační soubory simulace a požadavek je zařazen do fronty. Jakmile nastane na požadavek řada, spustí se program (externí proces), který inicializuje novou virtuální obrazovku. Po inicializaci program vypíše číslo nové obrazovky. Jakmile správce požadavků číslo obdrží, spustí v dalších podprocesech správce oken a samotný program ISIM. Po spuštění je vygenerován unikátní klíč. Tento klíč se spolu s číslem virtuální obrazovky uloží do složky noVNC klíčů. Ve složce se nachází klíče všech aktivních spojení. Služba noVNC při novém spojení podle obdrženého klíče vyhledá ve složce odpovídající virtuální obrazovku a s ní klienta propojí. Díky tomu lze na jednom síťovém portu obsluhovat více připojených klientů a je tím také vyřešena autentizace – klient musí k navázání spojení znát unikátní klíč.



Obrázek 3.7: Struktura spuštěných programů pro přenos okna simulace a ukázka více spuštěných instancí simulace.

3.2.4 Aplikační rozhraní

Překladový server zpřístupňuje dva síťové porty. Jeden z nich slouží ke komunikaci s programem přijímající požadavky na překlad či simulaci. Komunikace probíhá protokolem WebSocket, konkrétně jsou jednotlivé zprávy ve formátu JSON. Spojení zůstává otevřené i po dokončení akce. Při požadavku o navázání spojení se musí klient identifikovat platným autentizačním tokenem ve formátu JWT. Jak klient získá tento token je popsáno v sekci 3.4. Token je předán v hlavičce požadavku tak, jak je tomu v HTTP komunikaci zvykem. Pokud

proběhne autentizace v pořádku, je spojení akceptováno, jinak je ukončeno a klient obdrží patřičný stavový kód s textovým popisem důvodu zamítnutí požadavku (např. stavový kód: 401, důvod: „konec platnosti tokenu“). Po vytvoření spojení server čeká na příchozí požadavky od klienta. Klient může poslat požadavek na spuštění překladu nebo simulace. Součást tohoto požadavku musí být informace o konfiguraci projektu a všechny zdrojové soubory nutné k přeložení projektu (včetně zdrojových souborů použitých knihoven). Každá položka popisující soubor obsahuje jeho původní název a obsah souboru kódovaný do formátu Base64. V tomto formátu jsou kódovány také soubory poslané překladovým serverem (například soubory vytvořené během překladu).

Na nový požadavek může server zareagovat více způsoby. Pokud je požadavek zařazen do fronty, server odešle zprávu informující o této skutečnosti a dále odesílá zprávy o stavu fronty. Po spuštění a ukončení akce server pošle klientovi stavovou zprávu. V případě překladu zpráva o ukončení akce obsahuje návratový kód programu a vygenerované soubory. U simulace zpráva o ukončení neobsahuje žádná data, ale naopak zpráva o začátku simulace obsahuje URL adresu, na které je klientovi dostupné sezení s otevřeném programem ISIM, který je možné vzdáleně ovládat.

Druhý ze síťových portů komunikuje se jednoduchým web serverem služby noVNC a je určen k přenosu obrazu a ovládání okna vzdálené simulace. V tomto případě je použit mix protokolů HTTP a WebSocket. Protokolem HTTP se načítá rozložení, styly a skripty stránky, která v prohlížeči (nebo v našem případě editoru) komunikuje protokolem WebSocket s noVNC službou a zpřístupní tak vzdálené ovládání aplikace. V požadavku na vytvoření spojení pro vzdálené ovládání je nutné specifikovat token konkrétní instance. Nejedná se o autentizační JWT token uživatele, jde o náhodně vygenerovaný token, který zašle překladový server při odbavení požadavku vzdálené simulace.

3.3 Utilita na komunikaci se zařízením FITkit

S přípravkem FITkit lze komunikovat pomocí sériové linky. Jak popisuje sekce 2.1.3, v době psaní textu bohužel neexistuje žádný způsob, jak nativně v editoru[2] komunikovat přes sériové nebo USB rozhraní. Proto je zvolen jiný způsob komunikace – vytvoření samostatné utility ke komunikaci se zařízením FITkit, která funguje jako prostředník mezi editorem a zařízením.

Jedná se o utilitu s rozhraním v příkazovém řádku, která obstarává vše ohledně komunikace se zařízeními FITkit – **vyhledávání připojených zařízení**, **otevření terminálu** a **nahrání nového programu**. Při implementaci je důležitá přenositelnost a výsledná velikost spustitelného souboru, aby bylo možné utilitu zahrnout v rozšíření. Rovněž návrh počítá se samostatným použitím mimo rozšíření, pokud se uživatel rozhodne stáhnout pouze tuto utilitu.

3.3.1 Vyhledání připojených zařízení

Základní funkcionalitou je vyhledání všech připojených zařízení FITkit. Výstupem je seznam nalezených zařízení ve formátu JSON (pole objektů), z důvodu jednoduchého napojení na jiné aplikace. Každý záznam o nalezeném zařízení obsahuje název sériového portu, na kterém naslouchá mikroprocesor; verzi FITkit zařízení a revizi. Pokud nejsou nalezeny žádné zařízení, je výstupem prázdné pole. Důležité je zajistit, aby standardní výstup byl vždy validní JSON objekt.

[2]Editor Visual Studio Code je postaven na technologii Node.js

3.3.2 Otevření terminálu pro běžnou komunikaci

Spuštěním utility v módu terminálu je se zařízením FITkit navázáno spojení v normálním režimu, kdy je spuštěn nahraný program. Pokud není specifikován konkrétní sériový port, utilita provede před spojením autodetekci připojených zařízení a připojí se na první detekovaný FITkit. Po navázání spojení je výstup a vstup sériové linky přesměrován na standardní vstup a výstup utility a lze tak s běžícím programem interaktivně komunikovat. V tomto režimu se utilita chová jako prostředník komunikace přes sériovou linku, je tedy teoreticky utilitu v tomto módu použít ke komunikaci s jakýmkoliv zařízením na sériové lince (za předpokladu, že toto zařízení používá ke komunikaci stejné parametry jako FITkit).

3.3.3 Programování zařízení

Způsob programování je totožný s utilitou fkflash (viz sekce 2.2.2), která se na programování FITkitu používá v současné době. Stejně jako v případě módu terminálu, lze zadat konkrétní sériový port nebo přenechat volbu sériového portu na této utilitě, která se spojí s prvním nalezeným zařízením. Při spuštění je nutné uvést cestu k binárnímu souborů obsahující program, který se nahraje do paměti FPGA čipu a cestu ke dvěma souborům ve formátu Intel HEX, jejímž obsahem je program pro MCU. Dva soubory jsou použity z důvodu dvou různých verzí mikrokontroleru v zařízení FITkit verze 1.x a 2.x. Stejně jako v případě utility fkflash je i v tomto případě použit speciální boot loader, který vytvořil Doc. Ing. Zdeněk Vašíček, PhD. Tento boot loader výrazně zrychluje programování.

3.4 Autentizace uživatelů

V této sekci je popsán návrh webového rozhraní, které slouží ke generování tokenů určených k autentizaci na překladovém serveru. Generovaný token je ve formátu JWT (viz 2.3.4), který je digitálně podepsán. K podpisu je využit asymetrickým klíč, kdy se dají data ověřit veřejným klíčem, ale již nelze s tímto klíčem vytvořit nový JWT token. Celý proces je navrhnut s ohledem na soukromí uživatele. Rozhraní se skládá ze dvou částí – rozhraní pro aplikace, které žádají o nový token a uživatelského rozhraní. Způsob získání autentizačního tokenu je následující:

1. Externí aplikace zažádá o vytvoření nového požadavku. Rozhraní odpoví novým ID požadavku.
2. Aplikace požádá operační systém, aby zobrazil stránku pro potvrzení generace nového tokenu. Tato stránka je otevřena ve výchozím internetovém prohlížeči a je předáno ID požadavku.
3. Mezitím se aplikace v pravidelných intervalech dotazuje na stav vyřízení požadavku.
4. Uživatel po přihlášení na stránku zvolí, zda požadavek na získání tokenu povolit či zamítnout.
5. Po zvolení jedné z možností se zobrazí potvrzení, uživatel může stránku zavřít a vrátit se zpět do aplikace.
6. Aplikace při dalším dotazu na stav požadavku dostane informaci o tom, zda byl požadavek povolen a pokud byl, poté má k dispozici i vygenerovaný token.

3.4.1 Aplikační rozhraní

Jde o část rozhraní, která umožňuje externí aplikaci (například FITkit VSCode rozšíření) vytvořit nový požadavek na autentizační token a následně tento token získat. Každý požadavek je identifikován unikátním ID. Toto ID je vytvořeno při první komunikaci. U dotazu na stav vyřízení požadavku je nutné, aby aplikace předala ID požadavku. Pokud uživatel zamítl požadavek, je aplikace informována o zamítnutí požadavku, v opačném případě aplikace získá vygenerovaný token, který může použít při navázání spojení s překladovým serverem. Jak je uvedeno výše, token je digitálně podepsán, a tak aplikace ani uživatel nemůžou změnit data tokenu. Token obsahuje informace o vydavateli, jméno uživatele a datum platnosti, kdy po uplynutí data bude token překladovým serverem odmítnut a je třeba požádat o opětovné vygenerování tokenu. Je doporučeno použít rozumnou délku platnosti, aby se tato akce nemusela provádět často. Jako rozumná doba se nabízí délka jednoho semestru (4 měsíce).

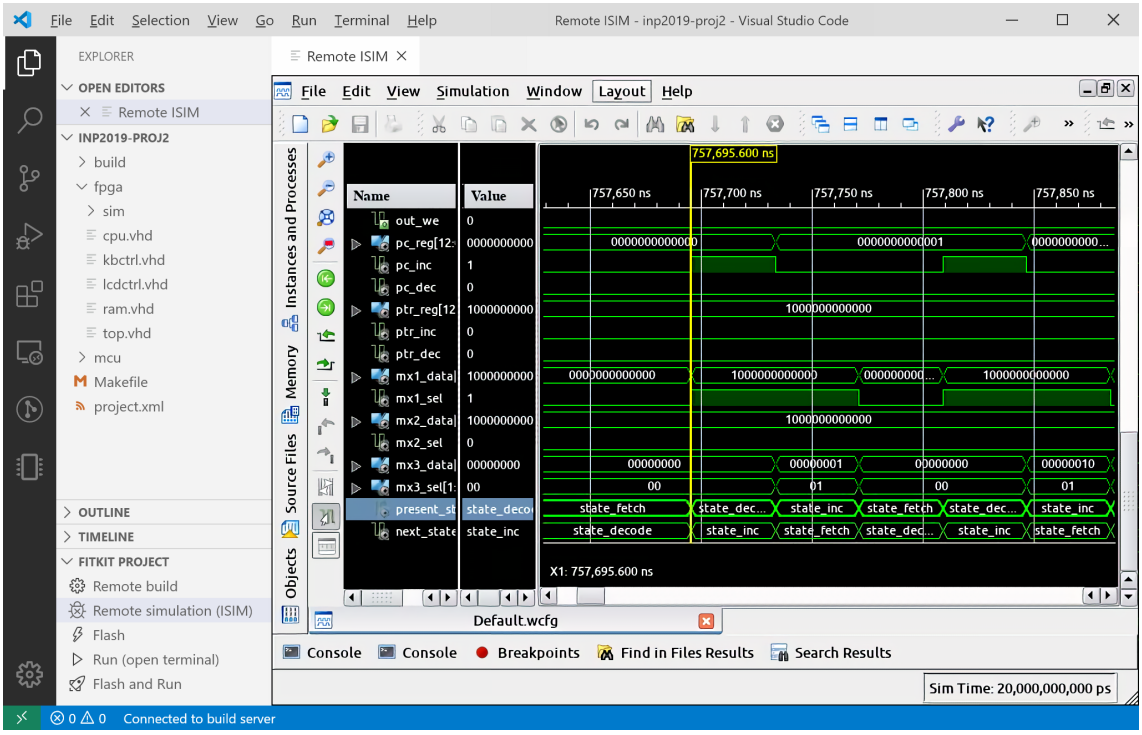
3.4.2 Uživatelské rozhraní

Uživatelské rozhraní se skládá z webové stránky, kterou aplikace zobrazí uživateli při novém požadavku. Uživatel provede přihlášení a vybere, zda požadavek aplikace povolit, či zamítnout. Uživatel tak má absolutní kontrolu nad generací přístupových tokenu. Tato stránka neprovádí autentizaci uživatele, a je závislá na existující autentizaci samotného web serveru. Návrh počítá s umístěním této stránky do sekce webu, kam mají přístup pouze přihlášení uživatele – například do privátní sekce FITkit webu (stejná sekce, kde jsou například umístěny obrazy virtuálních strojů).

Kapitola 4

Implementace

Tato kapitola popisuje implementaci všech částí návrhu. Implementace se úzce drží návrhu z předchozí kapitoly, a tak jsou zde rozvedeny do většího detailu především použité technologie a další důležité části implementace.



Obrázek 4.1: Zobrazení okna spuštěné simulace ze vzdáleného serveru přímo v editoru.

4.1 Rozšíření editoru

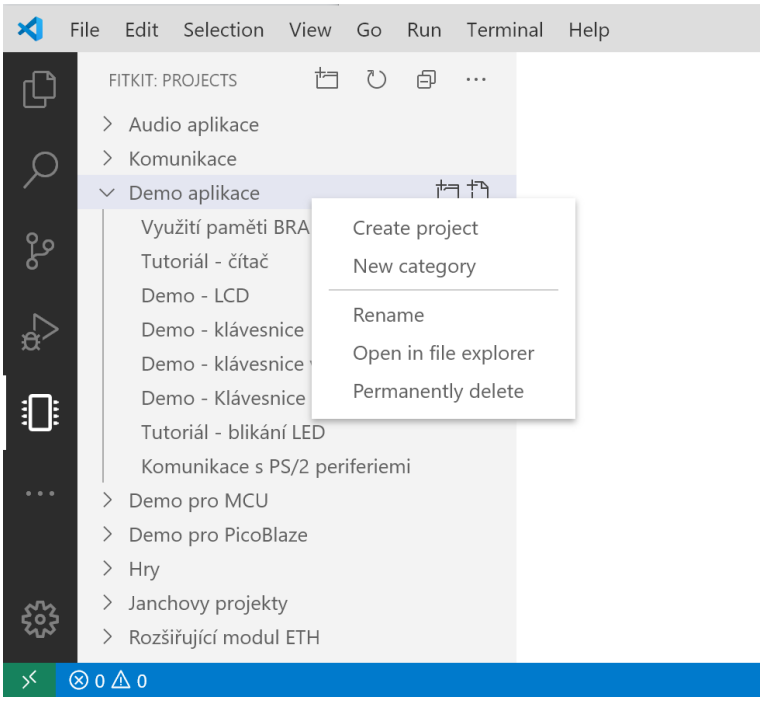
Hlavní část práce, rozšíření editoru je psána v jazyce TypeScript. Jazyk TypeScript neprovádí typovou kontrolu za běhu. Podporu typové kontroly za běhu přidává modul s názvem `typescript-is`[1]. Kontrola typů za běhu se hodí především při získávání externích dat,

[1]Detail modulu `typescript-is` – https://www.npmjs.com/package/typescript-is

u kterých si nemůžeme být jisti jejich správností. Konkrétně v tomto rozšíření tak typovou kontrolou prochází objekty načtené z konfiguračních XML souborů a příchozí zprávy ze serveru. Modul tak výrazně usnadňuje práci a přehlednost zdrojového kódu, který nemusí obsahovat kontrolu typu každé položky objektu. Aby šlo využít plného potenciálu modulu `typescript-is`, je nutné k překladu projektu použít komunitní odnož TypeScript překladače, která se nazývá TTypeScript[2]. Je plně kompatibilní s originálním překladačem, ale navíc podporuje pluginy, které mohou upravit chování překladače při generaci výstupních souborů. Takovým pluginem je právě `typescript-is`, který při překladu vygeneruje kód běhové typové kontroly.

4.1.1 Správa projektů

Způsob a dostupné akce správy repozitáře a projektů se drží návrhu ze sekce 3.1.1. Výsledné zobrazení projektů je možné vidět na obrázku 4.2. Zobrazením panelu se aktivuje kód pro načtení dostupných projektů. Pokud složka repozitáře neexistuje, zobrazí se uživateli varovná zpráva v pravém spodním rohu editoru s tlačítkem ke stažení repozitáře. V kontextové nabídce hlavního panelu se skrývá možnost kdykoliv stáhnout poslední verzi repozitáře. Pokud existuje lokální kopie, jsou existující soubory přepsány novou verzí staženou ze serveru.

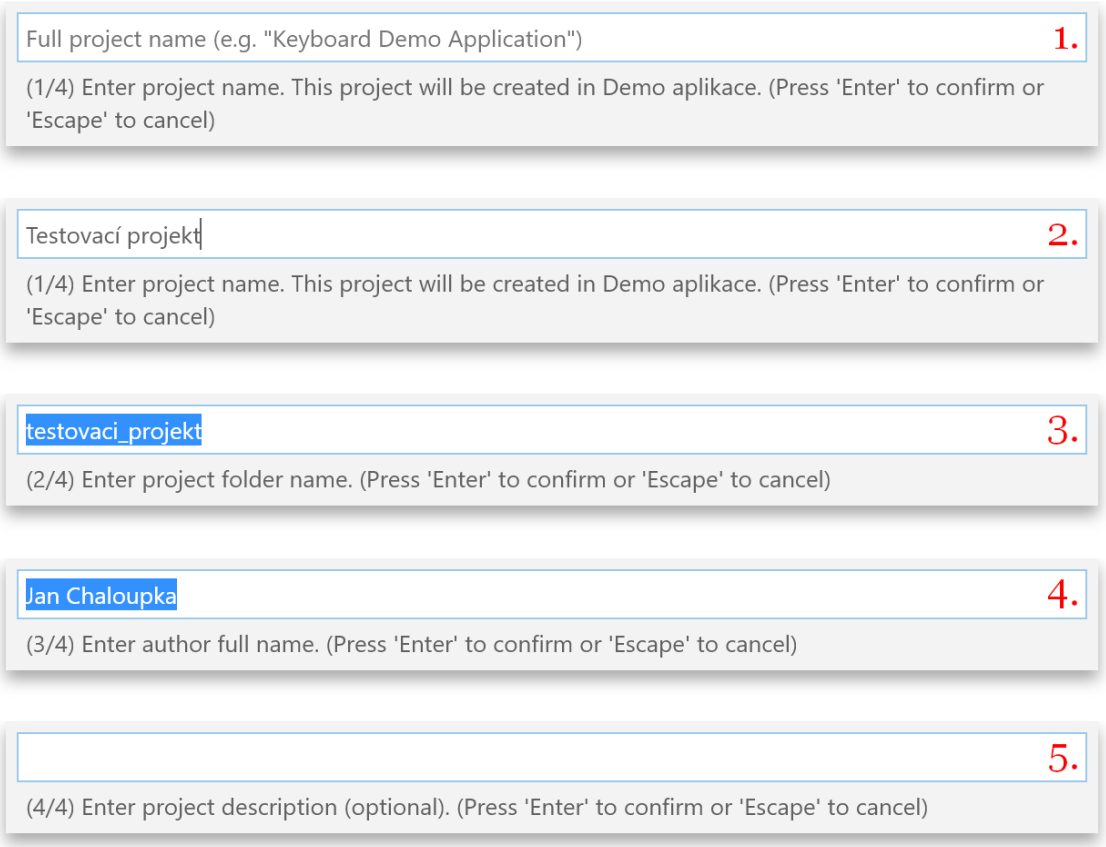


Obrázek 4.2: Seznam lokálních projektů včetně kontextové nabídky zobrazující dostupné akce.

Přejmenování kategorií a projektů bylo původně zamýšleno zobrazením vstupního pole přímo na místě upravované položky (jak je tomu například u přejmenování souborů otevřeného projektu). Tuto funkcionalitu bohužel v současné době aplikační rozhraní editoru nepodporuje. Proto je přejmenování řešeno zobrazením vstupního pole na horní straně edi-

[2]Detail projektu TTypeScript – https://www.npmjs.com/package/ttypescript

toru. Přes vstupní pole se také zadávají údaje při vytváření nové kategorie nebo projektu. Průběh vytváření nového projektu je viditelný na obrázku 4.3. Vytváření projektu je pojato ve stylu průvodce, kdy uživatel ve čtyřech krocích zadá nejnutnější údaje o projektu a poté je projekt vytvořen. Uživatel může kdykoliv průvodce ukončit stisknutím klávesy `ESC`. V ideálním případě stačí uživateli zadat pouze „pěkný“ název projektu. Název složky projektu je předvyplněn, uživatelské jméno je získáno z operačního systému a popis projektu je volitelný. Vygenerovaný projekt obsahuje základní kostru projektu včetně kódu pro MCU a FPGA. Byla použita kostra projektu ze staršího FITkit rozšíření pro editor Sublime Text.



Obrázek 4.3: Průběh vytvoření nového projektu.

4.1.2 Práce s otevřeným projektem

Rozšíření ve svém manifest souboru uvádí další doporučené rozšíření. Jde o C/C++ rozšíření[3], které přidává podporu jazyka C používaného ve zdrojových souborech MCU a rozšíření přidávající základní podporu jazyka VHDL[4].

Vzdálený překlad, simulace a komunikace se zařízením FITkit probíhá přesně podle návrhu v sekci 3.1.2. Data konfiguračních XML souborů projektů jsou pomocí modulu

[3]Stránky C/C++ rozšíření – https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools

[4]Stránky VHDL rozšíření - https://marketplace.visualstudio.com/items?itemName=puorc.awesome-vhdl

`fast-xml-parser` převedeny do objektu jazyka JavaScript. Následně se provede kontrola, zda objekt odpovídá očekávané šabloně. Tyto šablony (definované jako rozhraní) jsou umístěny ve složce `models` uvnitř složky zdrojových souborů rozšíření. Získávání souborů nutných k překladu a simulaci je komplikován faktem, že lze závislost v konfiguračním souboru definovat několika možnými způsoby. Spuštěním každé akce se zobrazí virtuální terminál, zobrazující stav požadavku a příchozí zprávy serveru. V terminálu se zobrazuje i aktuální stav fronty, pokud je server přetížený a požadavek byl umístěn do fronty. Příklad zobrazení stavu fronty je možný vidět na obrázku 4.4. U vzdálené simulace je kromě výstupu terminálu přenášen obraz spuštěné aplikace přímo do editoru. Panel `WebView`, ve kterém se programu vykresluje bohužel neumožňuje přímé načtení stránky z adresy URL. Toto omezení je obejito zobrazením jednoduché HTML stránky, která obsahuje pouze rámec (`<iframe>`) a v tomto rámci je načtena stránka s přenosem vzdálené aplikace. Jak simulace projektu v editoru vypadá je zobrazeno na první stránce kapitoly, na obrázku 4.1.

```
[LOCAL] Establishing connection to build server...
[LOCAL] Connection established. Sending simulation request...

[LOCAL] Unfortunately, the server is at maximum capacity. Your task has been placed in queue.
[LOCAL] You can cancel this task by killing this terminal (note that by doing this you will loose
your position in the queue)

[LOCAL] Your current position in the queue: 2 out of 2 waiting task(s)
[LOCAL] Your current position in the queue: 1 out of 1 waiting task(s)
[LOCAL] You are first in the queue. Expect your task to start soon...
```

Obrázek 4.4: Výstup konzole editoru informující uživatele o velkém vytížení serveru a zařazení požadavku do fronty.

Poslední, ale přesto důležitá část je komunikace se zařízením FITkit. Jak je uvedeno v návrhu (viz sekce 3.1.2), k sériové komunikaci a programování je využita samostatná utilita. Rozšíření cílí na tři hlavní platformy (Windows, Linux, macOS) a tak obsahuje tři spustitelné soubory utility, jeden pro každou platformu. Po aktivaci rozšíření zjistí aktuální architekturu a vybere odpovídající binární soubor. Soubory utility samy o sobě zabírají něco málo pod 10 MB. Díky tomu, že generovaný instalační balíček rozšíření je komprimovaný, je tato velikost dále snížena. Binární soubory lze poměrně dobře komprimovat, neboť jsou z části identické – všechny binární soubory cílí na stejnou architekturu x64 a hlavní logika se mezi operačními systémy nemění.

4.2 Konfigurace překladového serveru

Jako základ pro konfiguraci překladového serveru je použita distribuce Ubuntu, konkrétně ve variantě Ubuntu Server. Jedná se o populární distribuci, která byla vybrána především díky mým existujícím zkušenostem s touto distribucí. Řešení by ale mělo fungovat ve většině populárních Linuxových distribucích. Odzkoušená je pouze distribuce Ubuntu Server, ve verzích 16.04LTS a 20.04LTS. Překladový server obsahuje následující základní programy a utility:

- Programový balík **Xilinx WebPack**, umístěný ve složce `/opt/Xilinx` spolu s licencí. Aby měl k licenci přístup každý uživatel v systému, je tato licence uložena přímo ve složce instalace (konkrétně `/opt/Xilinx/13.1/ISE_DS/ISE/data`).
- **MSP-GCC** překladač zdrojového kódu mikrokontroleru řady MSP430.

- Programy **xorg** a **TigerVNC**[5] k vytvoření virtuální obrazovky přenášenou protokolem VNC. Program TigerVNC byl vybrán díky jeho schopnosti měnit rozlišení obrazovky za běhu. Díky tomu se obraz může přizpůsobit oknu prohlížeče (a není uzamčen na některé pevné rozlišení, například 1024×678).
- Správce oken **ratpoison**[6]. Jde o minimalistického správce oken, který zobrazuje aktivní aplikaci v režimu celé obrazovky bez dalších dekoračních prvků. Používá se také například u různých informačních kiosků. Je tak ideální pro použití k zobrazení okna programu ISIM na virtuální ploše.
- Sandboxovací utilita **firejail** využita k omezení přístupu spuštěných aplikací. Zabraňuje uživateli zobrazení ostatních souborů projektů na serveru. Více o této utilitě se nachází v sekci 2.3.3.
- Služba **Websockify** projektu noVNC, která se stará o přenos obrazu do noVNC klienta běžící ve webovém prohlížeči. Obsahuje také základní autentizaci a přesměrování pomocí tokenů. Na jednom síťovém portu tak může běžet více přenosů.
- Vlastní program správy připojení a požadavků (více o implementaci v sekci 4.3).

Na odevzdaném médiu se nachází instalační skript, který nainstaluje a rozbalí na čistou instanci serveru všechny potřebné programy a soubory a promění tak server na překladový server. Soubory balíku Xilinx WebPack nejsou z licenčních důvodů přiloženy, předpokládá se tak manuální instalace balíku před spuštěním instalačního skriptu. Rovněž je po instalaci nutné nakonfigurovat program správy požadavků, ukázkový konfigurační soubor se po instalaci nachází ve složce `/opt/build-server`.

4.3 Správa připojení a požadavků

Chování a účel programu správy připojení a požadavků je popsán v návrhu v sekci 3.2.1. Program je postaven na prostřední Node.js a je psaný v jazyce TypeScript. Tento jazyk byl zvolen k zachování konzistentnosti s jazykem a strukturou zdrojového kódu rozšíření. Stejně jako v případě rozšíření je zde využito běhové typové kontroly, které se aplikuje na příchozí požadavky klienta. Program běží na pozadí na serveru jako služba. Díky tomu lze jednoduše zjistit jeho stav a záznam (log) je ukládán na stejné místo, jako záznamy ostatních služeb. Pokud by nastala chyba v programu a předčasně došlo k jeho ukončení, systém dokáže automaticky službu restartovat. Při spuštění je načten konfigurační soubor, kde je možné nastavit základní věci, kterými jsou:

- Limit počtu aktivních úloh (simulace a překladu).
- Síťový port, na kterém server naslouchá.
- Nastavení ověřování JWT autentizačních tokenů (cesta k veřejnému klíči a povolené algoritmy).
- Umístění dočasných složek projektů a tokenů VNC serveru.
- Základní adresa, na které naslouchá služba Websockify projektu noVNC, kam bude uživatel po vytvoření prostředí simulace odkázán.

[5] Stránky projektu TigerVNC - https://tigervnc.org/
[6] Stránky projektu ratpoison – https://www.nongnu.org/ratpoison/

4.4 Utilita pro komunikaci se zařízením FITkit

Jak je uvedeno v návrhu (sekce 3.3), utilita je vytvořena především ke zprostředkování komunikace mezi rozšířením a fyzickým FITkit zařízením. Aby utilitu šlo distribuovat spolu s rozšířením, je nutné zvolit vhodný programovací jazyk, ve kterém lze psát multiplatformí aplikace a překládat je do binárního spustitelného souboru. Výsledná utilita je psaná v jazyce Go. Důvod použití právě tohoto jazyka a jeho základní vlastnosti jsou blíže popsány v sekci 2.1.3.

Ke komunikaci přes sériovou linku je využito knihovny `albenik/go-serial`[7]. Tato knihovna staví na základech populární knihovny `bugst/go-serial` a navíc tuto knihovnu rozšiřuje o další nezbytné funkce, jako je například časový limit čtení. Knihovnu využívá mnoho dalších projektů podobných této utilitě – například knihovny komunikující s vývojovými deskami Arduino.

Při vyhledávání dostupných FITkit zařízení je prvně načten seznam dostupných sériových portů. Knihovna pro sériovou komunikaci spolu s dostupnými porty vrátí i ID odpovídajícího USB zařízení (pokud lze zjistit). U každého portu se prvně zkontroluje, zda patří k zařízení s ID `0403:6010` (ID převodníku na zařízení FITkit). Pokud ID souhlasí, naváže se spojení s daným portem a kontroluje se, zda zařízení odpoví očekávaným řetězcem. Kontrola je kvůli rozlišení, zda jde o port kanálu B (FPGA) nebo A (MCU) FTDI převodníku. Pokud řetězec souhlasí, přečte se ze získaných dat verze a revize zařízení FITkit.

Zdrojový kód programování FITkit zařízení vychází z kódu utility fkflash. Ta ke komunikaci s boot loaderem mikrokontroleru používá upravenou knihovnu `mspbsl` pro jazyk Python. V době psaní textu neexistuje žádná alternativa této knihovny v jazyce Go, a proto bylo nutné knihovnu přepsat z jazyka Python do jazyka Go. Knihovna je umístěna v samostatném modulu projektu, lze ji tak použít v dalších projektech. Více o způsobu programování zařízení FITkit je k dispozici v sekci 3.3.3.

4.5 Autentizační stránka

V návrhu autentizační stránky se předpokládá nasazení na současné webové stránky projektu FITkit. Podle hlaviček HTTP odpovědí bylo zjištěno, že FITkit web je postavený na jazyku PHP (údaj o použití jazyka obsahuje položka hlavičky `X-powered-by`). Konkrétně se jedná o verzi 5.6.40. Nutno podotknout, že tato verze již není více než rok podporovaná a bylo by vhodné, aby web přešel na některou z novějších, stále podporovaných verzí. V době psaní práce je poslední stabilní vydání PHP ve verzi 7.4.

Autentizační stránka je z tohoto důvodu implementována v jazyce PHP. Kód je psaný tak, aby byl plně funkční jak v poslední stabilní verzi (7.4), tak i na verzi 5.6 kvůli zachování kompatibility s webem FITkit. Autentizační stránka se skládá ze dvou hlavních skriptů. Těmi jsou `request-token.php` zpřístupňující aplikační rozhraní pro externí aplikace a `generate-token.php` k zobrazení potvrzující stránky přihlášeným uživatelům. Podmínkou správné funkčnosti je umístění obou skriptů na stejný web server, aby měli přístup ke stejným PHP session datům.

Na vytvoření a správu požadavků je využito funkce PHP session[8], která umožňuje pod unikátním klíčem ukládat na web serveru dočasná data sezení. Tato data jsou dostupná všem stránkám běžícím na web serveru. Většinou se PHP session ve webových aplikacích

[7] Repozitář knihovny `albenik/go-serial` – https://github.com/albenik/go-serial/
[8] Dokumentace modulu PHP session – https://www.php.net/manual/en/book.session.php

používá ve spojení s cookie, kdy jsou data sezení uživatele (například zda je uživatel přihlášen) uložena na serveru. Klič k těmto datům má klient ve své lokální paměti (cookie), a při každém požadavku klient klíč odešle web serveru. V našem případě se ale klíč sezení neukládá do cookie, ale předává se jako součást adresy URL jako jeden z parametrů. Toto řešení má několik výhod – odpadá nutnost správy databáze a nevyřízené požadavky jsou po čase automaticky vymazány web serverem (sezení má omezenou délku platnosti, standardně půl hodiny).

4.5.1 Aplikační rozhraní (`request-token.php`)

Skript umožňující externí aplikaci (VSCode rozšíření) vytvořit nový požadavek na token a po potvrzení uživatelem tento token získat. Jde o samostatný soubor bez dalších závislostí, který by měl být dostupný ve veřejné části webu (bez nutnosti přihlášení). Chováním odpovídá návrhu aplikačního rozhraní popsaného v sekci 3.4.1. Při dotazu na vytvoření nového požadavku je odeslán externí aplikaci nový klíč požadavku (který je ve skutečnosti klíčem sezení). Pokud se bude chtít aplikace dotázat na stav požadavku, odešle tento klíč web serveru jako argument v adrese URL. Stejně tak aby klient mohl požadavek potvrdit, aplikace zobrazí stránku pro potvrzení, které také předá tento klíč požadavku. Díky tomu tak obě stránky sdílejí klíč sezení a pokud stránka `generate-token.php` uloží JWT token do paměti sezení, pak bude dostupný i externí aplikaci, která si o něj původně žádala.

4.5.2 Uživatelská sekce (`generate-token.php`)

Skript generující autentizační JWT token po odsouhlasení uživatelem. Jde o implementaci podle návrhu v sekci 3.4.2. Ke generování tokenů je využito knihovny PHP-JWT[9]. Tento skript neprovádí sám o sobě autentizaci uživatele, ale je závislý na existující autentizaci webového serveru. Proto je vhodné tento skript umístit tam, kde jej může načíst pouze přihlášený uživatel (na webu FITkit například sekce `private`). Ve stejné složce musí být umístěn konfigurační skript `config.php` a složka `php-jwt` s PHP JWT knihovnou.

Je zavedena ochrana proti okamžitému potvrzení požadavku bez interakce uživatele. K tomu se využívá způsobu kryptografické nonce – v principu jde o náhodně vygenerovaný klíč, který je platný jen na jeden dotaz. Při každém načtení potvrzovací stránky se vygeneruje nový nonce klíč. Aby uživatel požadavek potvrdil, musí znovu stránce tento klíč předat, například kliknutím na tlačítko. Tak je zajištěno, že se potvrzovací stránka před potvrzením alespoň jednou zobrazí. Na této stránce se po první navigaci zobrazí uživateli, o co se jedná, která aplikace žádá o klíč a co to pro uživatele znamená. Je vysvětleno, jaké údaje aplikace potvrzením získá a jak s těmito údaji bude moci naložit. Uživatel se na základě těchto informací rozhodne, zda požadavek zamítnout nebo povolit. Zamítnutím se uživateli zobrazí potvrzení o zamítnutí. Povolením požadavku se vygeneruje JWT token s loginem uživatele a časem platnosti, který je podepsaný privátním klíčem. Všechny parametry generace lze nastavit v konfiguračním souboru. Token se uloží do pamětí sezení a tím je celá akce ukončena. Zobrazení stránky v různých stavech je možno vidět v příloze A.

[9]Repozitář použité knihovny PHP-JWT – https://github.com/firebase/php-jwt

Kapitola 5

Nasazení v praxi

Aby byl navržený systém použitelný, byla během vývoje (jako celek i jeho jednotlivé části) kontrolována jeho uživatelská použitelnosti, i jeho kompatibilita se současným systémem správy projektů (qDevKit).

Funkčnost utility pro komunikaci a programování byla porovnána s utilitou `fkflash`. Oběma utilitami byly nahrány ukázkové projekty z repozitáře FITkit projektů a následně porovnány výsledky – zda se oba nahrané programy na zařízení chovají identicky. V tomto testování byly použity obě verze zařízení FITkit (v1.2 a v2.0). Rovněž byla ověřena funkčnost utility na různých operačních systémech, a to jak samostatně, tak v rozšíření editoru. K tomu byly použity operační systémy Windows a Linux, konkrétně Windows 10 1909 a Ubuntu ve verzi 16.04 a 20.04. Z důvodu nedostupnosti Apple zařízení nebyla ověřena kompatibilita se systémem macOS.

Vzdálený překlad byl, stejně jako v případě utility, ověřován na projektech v repozitáři. Kromě veřejně dostupných projektů z repozitáře byly použity i soubory dřívějších projektů z předmětů IVH, INC a INP. Výstup byl porovnán s překladem v programu qDevKit na virtuálním stroji. Ve virtuálním stroji byl použit obraz `fitkit-vbox-201401.7z` stáhnutý ze stránek projektu FITkit. Tento obraz se běžně používá při práci na FITkit projektech v předmětech bakalářského studijního programu. Samotný překladový server běžel virtualizovaně na lokální síti a měl přiděleno jedno jádro procesoru a 2 GB paměti.

Během návrhu a implementace vzdáleného překladu bylo nutné ověřit použitelnost vzdáleného přenosu simulace i při pomalém internetovém připojení. K tomu bylo využito několik dobrovolníků z řad studentů, kteří mají pomalé internetové připojení. Nejpomalejší připojení dosahovalo rychlosti stahování kolem 6 Mbit/s (měření proběhlo orientačně pomocí webu `speedtest.net`). Při této rychlosti byla vzdálená simulace podle slov studenta stále použitelná. Aby bylo nalezeno minimum, kdy se již začíná projevovat zvýšená odezva, bylo nutné použít umělé omezení rychlosti přímo na přípojce routeru. Postupným snižováním rychlosti se došlo na rychlost 3 Mbit/s. Toto je podle experimentování hraniční rychlost, kdy se dá simulace bez větších problémů stále používat. Zvolený protokol VNC je velice efektivní k zobrazení uživatelského rozhraní, neboť posílá pouze změněné části obrazu. Většinu času tak není potřeba posílat žádná nová data. Největší zátěž na linku nastává v okamžiku, kdy se překreslí velká část plochy (u simulace to je především vyvoláno posunutím časové osy se signály). Na přiloženém médiu se nachází videozáznam testování odezvy právě při umělé omezení rychlosti na 3 Mbit/s.

Kapitola 6

Závěr

Hlavním cílem práce bylo vytvořit rozšíření editoru Visual Studio Code umožňující práci s projekty platformy FITkit a přípravkem FITkit bez nutnosti lokální instalace vývojových nástrojů. Kromě vzdáleného překladu projektů, který je zabezpečen pomocí překladového serveru, byla nad rámec zadání implementována i možnost vzdálené simulace. Instalací kompaktního rozšíření tak uživatel získává plnohodnotné vývojové prostředí pro práci s platformou FITkit.

Během návrhu jsem se setkal s mnoha problémy, které bylo nutné překonat. Hlavní překážkou byla nutnost návrhu klientské části tak, aby nebyla závislá na jedné konkrétní platformě a rozšíření tak mohl používat každý bez ohledu na operační systém. V práci jsem použil mnoho různých technologií. S některými jsem se již dříve setkal – jazyk TypeScript, komunikace protokolem WebSocket nebo vzdálený přenos aplikací. Se spoustou dalších jsem se ale seznámil právě u této práce. Naučil jsem se především používat jazyk Go, způsob programování mikrokontroleru a tvorbu rozšíření editoru Visual Studio Code. Dle mého názoru jsem všech stanovených cílů dosáhl a věřím, že navržený funkční celek bude pro mnoho studentů užitečný.

Kromě rozšíření editoru vznikly další části, které je možné využít samostatně nebo v jiných projektech. Jde především o utilitu pro komunikaci a programování FITkit zařízení, kterou lze stáhnout a používat samostatně v příkazovém řádku. Další samostatně použitelná část je knihovna `mspbsl` v jazyce Go zpřístupňující funkce ke komunikaci s boot loaderem mikroprocesorů řady MSP430.

Dalším krokem je nasazení překladového serveru na fakultní server, přidání autentizační stránky na web projektu FITkit a vydání rozšíření přímo do repozitáře editoru. Do budoucna je v plánu pokračování na vývoji tohoto systému a přidání dalších funkcí. Jedna z plánovaných funkcionalit je především rozsáhlejší možnost simulace projektu (výběr konkrétní simulované komponenty) a zobrazení projektu ve vývojovém prostředí Xilinx ISE na vzdáleném serveru. Kromě toho je pro rozšíření editoru plánovaná podpora více připojených FITkit zařízení a zvýrazňování chyb překladu přímo ve zdrojových souborech.

Literatura

[1] Auth0. JSON Web Token Introduction. *Jwt.io* [online]. [cit. 2020-05-01]. Dostupné z: https://jwt.io/introduction/.

[2] Firejail community. *Firejail security sandbox* [online]. [cit. 2020-05-02]. Dostupné z: https://firejail.wordpress.com/.

[3] Microsoft. *TypeScript - JavaScript that scales* [online]. [cit. 2020-01-09]. Dostupné z: https://www.typescriptlang.org.

[4] Microsoft. Why did we build Visual Studio Code? *Visual Studio Code* [online], 12. prosince 2019 [cit. 2020-01-08]. Dostupné z: https://code.visualstudio.com/docs/editor/whyvscode.

[5] Microsoft. Extension API. *Visual Studio Code* [online], 12. prosince 2019 [cit. 2020-01-08]. Dostupné z: https://code.visualstudio.com/api.

[6] Rozentals, N. *Mastering TypeScript - Second Edition.* 2. vyd. Birmingham: Packt Publishing Ltd., duben 2017. ISBN 978-1-78646-871-0.

[7] Stack Overflow. Developer Survey Results 2019. *Stack Overflow* [online]. 2019 [cit. 2020-01-08]. Dostupné z: https://insights.stackoverflow.com/survey/2019.

[8] Tišnovský, P. Go: minimalistický a překvapivě výkonný programovací jazyk. *Root.cz* [online], 20. listopadu 2018 [cit. 2020-05-01]. Dostupné z: https://www.root.cz/clanky/go-minimalisticky-a-prekvapive-vykonny-programovaci-jazyk/.

[9] Vašíček, Z. Firmware. *FITkit* [online]. 2006 [cit. 2020-05-01]. Dostupné z: https://merlin.fit.vutbr.cz/FITkit/firmware.html.

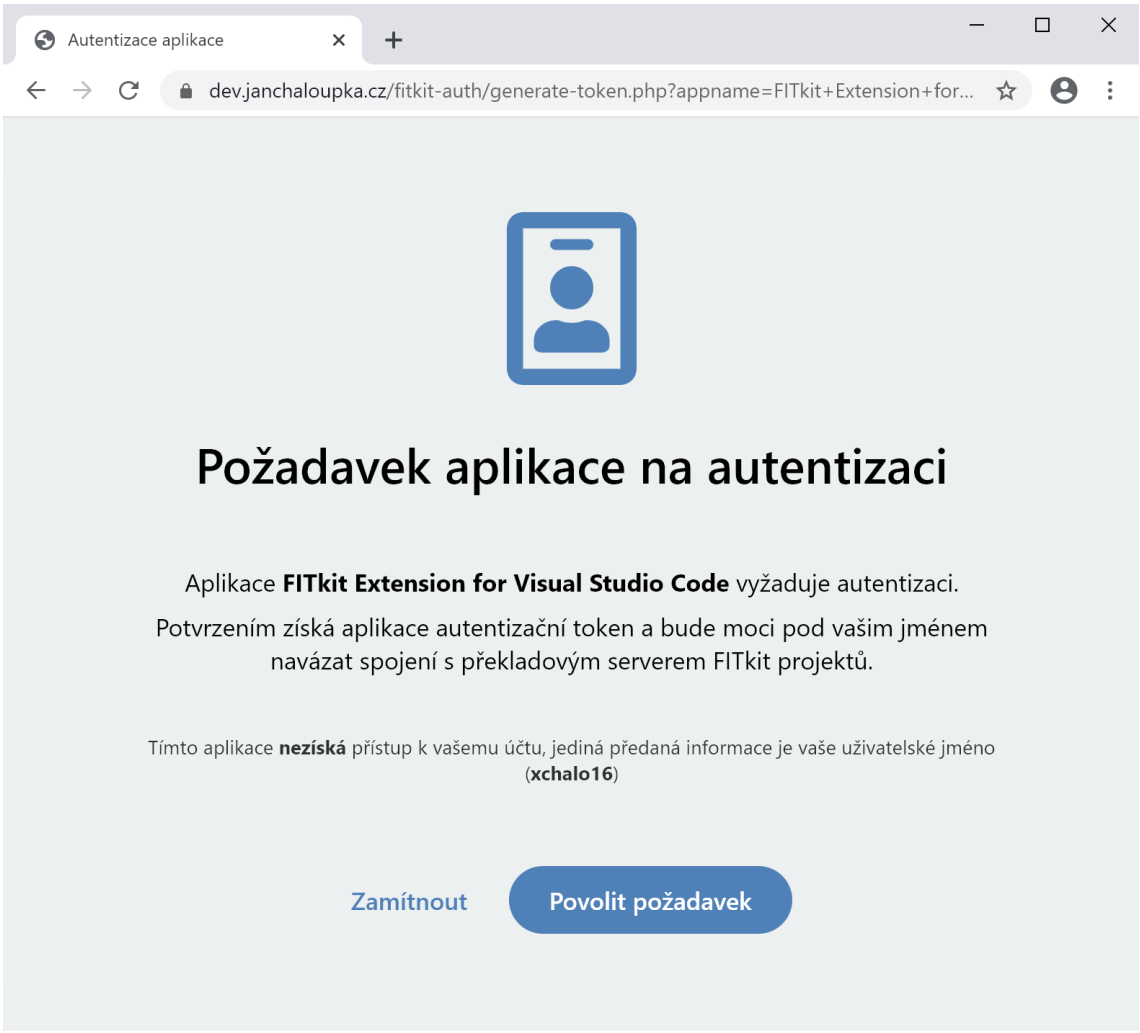
[10] Vašíček, Z. Překladový systém. *FITkit* [online]. 2006. Aktualizováno 3. 5. 2009 [cit. 2020-01-19]. Dostupné z: https://merlin.fit.vutbr.cz/FITkit/docs/navody/kompilacev2.html.

[11] Vašíček, Z. USB převodník FT2232. *FITkit* [online]. 2006. Aktualizováno 3. 4. 2009 [cit. 2020-01-19]. Dostupné z: https://merlin.fit.vutbr.cz/FITkit/docs/hardware/hw_ftdi.html.

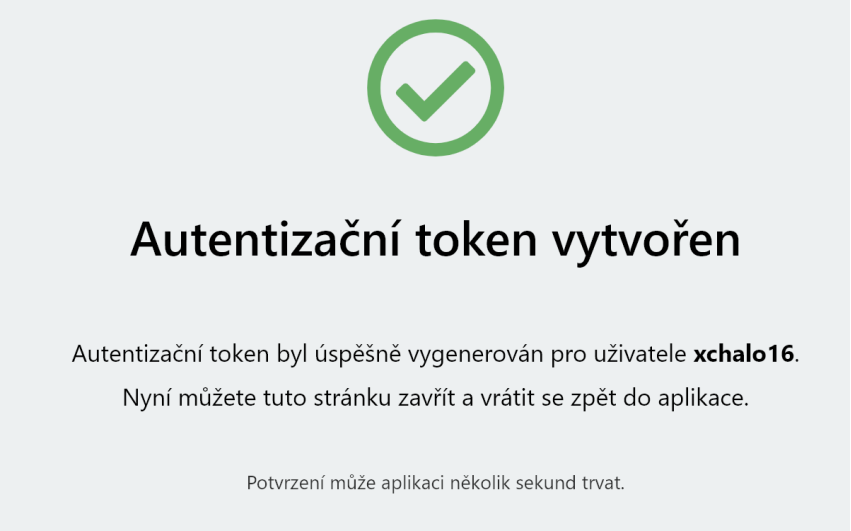
[12] Vašíček, Z. Programování a bootování FITkitu. *FITkit* [online]. 2006. Aktualizováno 3. 5. 2009 [cit. 2020-01-19]. Dostupné z: https://merlin.fit.vutbr.cz/FITkit/docs/hardware/hw_boot.html.

Příloha A

Vzhled uživatelského rozhraní autentizační stránky



Obrázek A.1: Stránka autentizačního webu, kde může uživatel povolit nebo zakázat požadavek externí aplikace na vygenerování autentizačního tokenu.



Obrázek A.2: Text zobrazený po odsouhlasení požadavku informující uživatele o dokončení akce.

Obrázek A.3: Text zobrazený po zamítnutí požadavku informující uživatele o dokončení akce.

Příloha B

Obsah přiloženého média

- `/`
  - `text` — Zdrojové soubory textu práce a samotný text ve formátu PDF.
  - `vscode` — Zdrojové soubory rozšíření pro Visual Studio Code.
  - `serial` — Zdrojové soubory utility pro komunikaci se zařízením FITkit na sériové lince včetně návodu k použití.
  - `build-server` — Zdrojové soubory programu na obsluhu připojených klientů a jejich požadavků.
  - `auth-server` — Zdrojové soubory webu pro generování autentizačních tokenů včetně návodu pro nasazení.
  - `server-setup` — Instalační skript a všechny potřebné soubory pro vytvoření překladového serveru (mimo software Xilinx WebPack). Testováno na linuxové distribuci Ubuntu Server 16.04LTS a 20.04LTS.
  - `resources`
    - `fitkit.vsix` — Vygenerovaný balíček pro přímou instalaci rozšíření do editoru.
    - `fitkit-serial-*` — (3x) Spustitelné binární soubory utility pro sériovou komunikaci (pro x64 platformy Windows, Linux a macOS).
    - `test-isim-3mbit.mkv` — Videozáznam z testování plynulosti přenosu obrazu prostředí simulace při uměle omezené rychlosti internetového připojení (3 Mb/s).