Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Automatizované testování domácích úkolů z programování

Filip Havíř

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Autor práce:	Filip Havíř
Studijní program:	Informatika
Vedoucí práce:	Ing. Tomáš Vokoun
Garantující pracoviště:	Katedra informačních technologií
Jazyk práce:	Čeština
Název práce:	**Automatizované testování domácích úkolů z programování**
Název anglicky:	**Automated testing of programming assignments**
Cíle práce:	Hlavní cíl: Hlavním cílem práce je navrhnut architekturu a následně sestavit komplexní webovou aplikaci díky které bude moci učitel jednoduše kontrolovat odevzdané úkoly z programování psané v různých programovacích jazycích. Dílčí cíle: - Zanalyzovat, které technologie jsou nejlepší na použití a které ne (Teoretická část) - Zkonstruovat podrobnou architekturu aplikace (od deploymentu na server až po webový frontend) za pomocí různých diagramů a schémat (Teoretická část) - Sestrojit samotnou webovou aplikace (Praktická část)
Metodika:	Metodika řešení teoretické části bakalářské práce bude založena na studiu a analýze odborných informačních zdrojů a následné architektuře řešení. Na základě znalostí získaných v teoretické části práce bude v praktické části zkonstruována celá webová aplikace. Na základě syntézy teoretických poznatků a výsledků praktické časti budou formulovány závěry práce. Na metodiku vývoje SW bude použit Prototypový přístup, kde bude testováno, zda vytvořená část softwaru splňuje dané požadavky a pokud ano, tak se vývoj posune k další části. A před samotným programováním bude provedena analýza pomocí vývojového diagramu.
Doporučený rozsah práce:	40-50 stran
Klíčová slova:	testování, automatizace, domácí úkoly

Doporučené zdroje informací:

1. KASPRZAK, J. *Red Hat Linux 5.1 [elektronický zdroj]*. Praha: Computer Press, 1998. ISBN 80-7226-106-1.
2. SUMMERFIELD, M. *Python 3: výukový kurz*. Brno: Computer Press, 2010. ISBN 978-80-251-2737-7.

Předběžný termín obhajoby: 2022/23 LS - PEF

Elektronicky schváleno: 14. 7. 2022
doc. Ing. Jiří Vaněk, Ph.D.
Vedoucí katedry

Elektronicky schváleno: 27. 10. 2022
doc. Ing. Tomáš Šubrt, Ph.D.
Děkan

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci " Automatizované testování domácích úkolů z programování" jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2023 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Tomáši Vokounovi za odborné vedení práce a také své ženě, Marii Havířové za její trpělivost během psaní této práce.

Automatizované testování domácích úkolů z programování

Abstrakt

Tato práce se zabývá návrhem a implementací systému pro automatické testování domácích úkolů z programování. Systém bude fungovat jako webová aplikace určená pro učitele a bude běžet na linuxovém serveru.

Klíčová slova: C, C++, Java, Automatické testování domácích úkolů, Webová aplikace, Flask, Python, Linux

Automated testing of programming assignments

Abstract

This thesis focuses on the design and implementation of an automatic homework testing from programming. System will work like web application designed for teachers and it will run on some Linux distribution.

Keywords: C, C++, Java, Automatic homework testing, Web application, Flask, Python, Linux

Obsah

1 Úvod

Programování je činnost, která se dá z větší části naučit pouze neustálým cvičením, učením se z vlastních chyb a čtením již napsaného kódu. Je tedy potřeba, aby člověk, který se tuto dovednost chce naučit, vytvořil velké množství malých i velkých projektů. Tato metodika se uplatňuje často při výuce programování na vysoké škole, ale i na střední škole. Studenti v průběhu předmětu odevzdávají mnoho projektů z programování a náročnost se postupně posouvá.

V případě, že by učitel musel všechny tyto domácí úkoly ručně zkontrolovat a oznámkovat, zabralo by mu to velké množství času, který by mohl věnovat spíše kvalitnější výuce a konzultaci se studenty. Vzniká tedy potřeba mít nějaký automatizovaný nástroj, který tuto práci odvede za něj a učitel pouze dostane o každém studentovi informaci, jak si vedl a kde udělal chybu.

Aby se dálo lépe pochopit, proč je takový automatický systém pro kontrolu úkolů potřeba, zde je pro představu jednoduchá úloha na zamyšlení:

Mějme učitele IT na vysoké škole, který vyučuje programování ve 3 třídách po 20 žácích. Učitel stráví u jednoho domácího úkolu odhadem 3 minuty, takže 3 třídy by kontroloval po dobu 180 minut. A kdyby žáci měli na každý týden jednu úlohu, za měsíc je to 720 minut, tedy 12 hodin. Nyní se udělají podobné výpočty, ale s tím, že existuje systém pro automatické testování, dále jen jako „aplikace“:

Učitel stráví u jednoho úkolu odhadem pouze 20 sekund, takže kontrola 3 tříd mu bude trvat 20 minut. Při týdenním odevzdávání je to za měsíc 80 minut, tedy 1,33 hodiny, což je **89% úspora času než bez použití aplikace**. Ušetřený čas tedy činí 10,5 hodiny, a to už je značný rozdíl.

Tento výpočet byl ale pouze velmi hrubým odhadem, který není podložený reálným měřením toho, kolik času zabere učiteli kontrolování domácích úkolů. Cílem této úlohy je pouze poukázat na možnost úspory času v kontrolování domácích úkolů z programování v případě existence nějakého automatizačního nástroje.

Vývoj automatizační aplikace pro kontrolování domácích úkolů z programování bude cílem této bakalářské práce.

1.1 Cíl práce

Cílem této práce je navrhnout a dále zkonstruovat komplexní webovou aplikaci, která usnadní práci učitele při výuce programování na vysokých a středních školách. Učitel tak nebude muset trávit mnoho času procházením úkolů z programování, ale bude se moct spíše zaměřit na samotnou výuku.

Hlavní cíl je rozdělen do několika dílčích cílů:

- Zanalyzovat, které programovací a skriptovací jazyky by měla tato aplikace podporovat
- Zanalyzovat, které webové technologie a operační systémy jsou nejlepší na použití a které ne
- Zkonstruovat podrobnou architekturu za pomocí různých diagramů a schémat
- Sestrojit samotnou webovou aplikaci

1.2 Metodika

Metodika řešení teoretické části bakalářské práce bude založena na studiu a analýze odborných informačních zdrojů a následné architektuře řešení. Dále budou porovnány různé operační systémy a webové technologie a budou hledány ty nejvhodnější na použití. Na základě znalostí získaných v teoretické části práce budou v praktické části provedeny analýzy výběru IT technologií a na jejich základě bude zkonstruována webová aplikace. Konstrukce aplikace bude předvedena na vzorovém příkladě. Na základě syntézy teoretických poznatků a výsledků praktické časti budou formulovány závěry práce.

Na metodiku vývoje SW bude použit Prototypový přístup, kde bude testováno, zda vytvořená část softwaru splňuje dané požadavky. Pokud ano, vývoj se posune k další části. Před samotným programováním bude provedena analýza pomocí vývojového diagramu.

2 Teoretická východiska

2.1 Windows

2.1.1 Základní informace

Windows je operační systém, který spravuje hardwarové a softwarové prostředky počítače. Poskytuje platformu pro spouštění aplikací, ovládání vstupních a výstupních zařízení a správu souborů a složek. První verze systému Windows, známá jako Windows 1.0, byla vydána v roce 1983. Od té doby společnost Microsoft vydala řadu verzí, z nichž nejnovější je Windows 11 (1).

Systém Windows podporuje širokou škálu aplikací, včetně softwaru pro produktivitu, herního softwaru a multimediálního softwaru. Je kompatibilní s celou řadou hardwarových zařízení, včetně tiskáren, skenerů a fotoaparátů. Systém Windows také poskytuje uživatelsky přívětivé grafické uživatelské rozhraní (GUI), které umožňuje uživatelům komunikovat se systémem prostřednictvím ikon, nabídek a oken (2).

2.1.2 Použití

Systém Windows je nejrozšířenějším operačním systémem na světě. Podle údajů společnosti StatCounter z ledna 2023 má systém Windows celosvětový podíl na trhu 74,4 %, druhým nejoblíbenějším operačním systémem je OS X nebo také jako MacOS s podílem 15,33 %. Systém Windows se používá v celé řadě odvětví, včetně podnikání, vzdělávání, zdravotnictví a státní správy (3).

2.2 Linux

2.2.1 Základní informace

Linux je svobodný operační systém s otevřeným zdrojovým kódem, který vychází z operačního systému Unix. Jádro poskytuje nízko-úrovňové rozhraní mezi hardwarem a softwarem systému. Jádro spravuje systémové prostředky, včetně paměti, vstupních/výstupních zařízení a sítě (4).

Operační systém Linux se skládá z různých součástí, jako jsou systémové knihovny, nástroj příkazového řádku a grafického uživatelského rozhraní. Tyto součásti spolupracují a poskytují výkonný a flexibilní operační systém, který lze přizpůsobit konkrétním potřebám uživatele (4).

2.2.2 Použití

Linux se používá v široké škále zařízení, včetně serverů, stolních počítačů a vestavěných systémů. Na trhu serverů je Linux oblíbenou volbou pro webové servery, databázové servery a souborové servery. Je známý svou stabilitou a bezpečností, díky čemuž je oblíbenou volbou pro kritické aplikace. Na trhu stolních počítačů používá Linux stále více uživatelů, kteří oceňují jeho stabilitu, bezpečnost a flexibilitu. Na trhu vestavěných systémů se Linux používá v široké škále zařízení, včetně směrovačů, chytrých televizorů a automobilových systémů. Flexibilita

systému Linux z něj činí oblíbenou volbu pro tato zařízení, protože umožňuje vývojářům přizpůsobit operační systém specifickým potřebám zařízení (4).

2.2.3 Výhody systému

Linux má oproti jiným operačním systémům řadu výhod. Jednou z hlavních výhod je jeho flexibilita. Linux lze přizpůsobit konkrétním potřebám uživatele, ať už jde o stolní počítač, server nebo vestavěný systém. Další výhodou systému Linux je jeho bezpečnost. Linux je známý svými silnými bezpečnostními prvky, včetně systému oprávnění založeného na uživatelích a schopnosti běžet bez grafického uživatelského rozhraní. Linux má také menší plochu pro útoky než jiné operační systémy, a proto je méně zranitelný vůči malwaru a dalším bezpečnostním hrozbám. Linux je také známý svou stabilitou. Díky způsobu, jakým Linux spravuje systémové prostředky, je méně náchylný k pádům a jiným typům selhání systému. Díky tomu je oblíbenou volbou pro kritické aplikace, jako jsou webové servery a databázové servery. A konečně, Linux je zdarma a má otevřený zdrojový kód. To znamená, že jeho kód může používat, upravovat a šířit kdokoli, což vedlo ke vzniku velké a aktivní komunity vývojářů. Tato komunita vytvořila pro Linux širokou škálu softwaru a nástrojů, což z něj činí výkonný a flexibilní operační systém (4) (**Error! Reference source not found.**).

2.2.4 Nevýhody systému

Přes mnohé výhody má Linux i některé nevýhody. Jednou z hlavních nevýhod je jeho kompatibilita pouze s určitým softwarem a hardwarem. Některý software a hardware není s Linuxem kompatibilní, což může uživatelům ztížit přechod z jiných operačních systémů. Další nevýhodou systému Linux je nedostatek uživatelsky přívětivých rozhraní. Distribuce Linuxu, jako jsou Ubuntu a Fedora, sice v této oblasti dosáhly značných zlepšení, ale k jejich efektivnímu používání jsou stále zapotřebí určité technické znalosti. Proto může být Linux méně přístupný začátečníkům, kteří neznají rozhraní příkazového řádku. V neposlední řadě může být Linux oproti jiným operačním systémům náročnější na nastavení a údržbu. Přestože distribuce Linuxu i v této oblasti významně pokročily, jejich instalace a konfigurace stále vyžaduje určité technické znalosti (5).

2.3 Kontejnerizace

Kontejnerizace je metoda balení softwarových aplikací a jejich závislostí do lehkých, přenosných jednotek zvaných kontejnery. Kontejnery poskytují určitou úroveň izolace mezi aplikací a základní infrastrukturou, takže jsou ideální pro nasazení v různých prostředích. Docker a Podman jsou dva populární kontejnerizační nástroje, které se objevily v posledních letech. Kontejnerizace zahrnuje zabalení aplikace a jejích závislostí do jedné spustitelné jednotky. Tato jednotka je pak spuštěna v kontejneru, který poskytuje lehké a přenosné běhové prostředí. Kontejnery jsou izolovány od sebe navzájem i od hostitelského operačního systému, což umožňuje, aby na jednom počítači běželo více kontejnerů bez konfliktů. Kontejnerizace se v posledních letech stává stále populárnější díky mnoha výhodám, mezi které patří např.:

- Přenositelnost: Kontejnery lze snadno přesouvat mezi různými prostředími, například vývojovým, testovacím a produkčním.
- Efektivita: Kontejnery jsou lehké a využívají méně prostředků než tradiční virtuální stroje.
- Izolace: Kontejnery poskytují určitou úroveň izolace mezi aplikací a základní infrastrukturou, což je činí bezpečnějšími a snadněji spravovatelnými (**Error! Reference source not found.**).

2.3.1 Docker

Docker je platforma pro kontejnerizaci, která byla poprvé uvedena na trh v roce 2013. Díky snadnému použití a rozsáhlé sadě funkcí se rychle stala průmyslovým standardem pro kontejnerizaci. Docker se skládá ze dvou hlavních součástí: Docker Engine a Docker Hub.
Docker Engine je běhové prostředí pro kontejnery Docker. Poskytuje rozhraní pro vytváření, spouštění a správu kontejnerů a také sadu rozhraní API pro interakci s démonem Docker. Docker Hub je cloudový registr obrazů Docker, který lze použít k ukládání a distribuci obrazů kontejnerů (7).

Kontejnery Docker jsou založeny na obrazech Docker, což jsou v podstatě snímky kontejneru v určitém časovém okamžiku. Obrazy se vytvářejí pomocí Dockerfiles, což jsou textové soubory, které specifikují součásti a konfiguraci kontejneru. Jakmile je obraz sestaven, lze jej použít ke spuštění kontejnerů na jakémkoli systému kompatibilním s Dockerem (7).

2.3.2 Podman

Podman je nástroj pro kontejnerizaci, který byl poprvé vydán v roce 2018. V mnoha ohledech se podobá nástroji Docker, ale zároveň má řadu zásadních rozdílů. Na rozdíl od nástroje Dockeru nevyžaduje nástroj Podman ke spouštění kontejnerů démona. Místo toho jsou kontejnery spouštěny jako jednotlivé procesy v hostitelském operačním systému. Díky tomu je Podman bezpečnější a snadněji se spravuje, protože pro kontejnery neexistuje centrální řídicí bod. Dalším klíčovým rozdílem mezi Podmanem a Dockerem je, že Podman je plně kompatibilní se standardy Open Container Initiative (OCI). To znamená, že Podman může spustit jakýkoli obraz kontejneru kompatibilní s OCI bez ohledu na to, zda byl vytvořen pomocí nástroje Docker nebo jiného nástroje pro kontejnerizaci. Díky tomu je Podman flexibilnější a interoperabilnější nástroj než Docker (8) (9).

2.4 Kompilátor

Kompilace je proces převodu zdrojového kódu napsaného v programovacím jazyce do spustitelného strojového kódu. Je to zásadní krok při vývoji softwaru, protože umožňuje programátorům vytvářet programy, které lze spustit na různých platformách a operačních systémech. Pro kompilaci je k dispozici několik nástrojů, například GCC, G++. Pro jednodušší kompilaci rozsáhlých zdrojových kódů existují automatizační nástroje sestavování jako například Make a Apache Maven (10).

2.4.1 GCC

GCC (GNU Compiler Collection) je populární open-source kompilátor, který je široce používán pro kompilaci jazyka C. GCC je vyvíjen projektem GNU a je k dispozici pod obecnou veřejnou licencí GNU. Je podporován na různých platformách, včetně systémů Linux, Windows a MacOS (11).

Jednou z hlavních výhod GCC je jeho robustnost a stabilita. Vyvíjí se již více než 30 let a používají jej miliony vývojářů po celém světě. GCC je také vysoce přizpůsobitelný a vývojáři mohou používat různé úrovně optimalizace pro zlepšení výkonu kódu. Kromě toho má GCC vynikající podporu pro ladění a hlášení chyb, což z něj činí ideální nástroj pro vývoj softwaru (12).

2.4.2 G++

G++ je kompilátor, který je součástí GCC a je speciálně navržen pro kompilaci jazyka C++. Poskytuje mnoho stejných funkcí a výhod jako GCC, například stabilitu, robustnost a přizpůsobitelnost. Je však optimalizován pro kompilaci kódu v C++ a obsahuje další funkce a optimalizace, které jsou specifické pro tento jazyk (11) (13).

Jednou z hlavních výhod G++ je jeho kompatibilita s GCC. Vývojáři, kteří znají GCC, mohou snadno přejít na používání G++, protože oba kompilátory mají mnoho stejných funkcí a syntaxi. Kromě toho obsahuje G++ několik rozšíření specifických pro jazyk C++, jako jsou šablony, jmenné prostory a výjimky, což z něj činí vynikající volbu pro vývoj v jazyce C++ (13).

2.4.3 Make

Make je nástroj pro automatizaci sestavování, který se používá ke správě procesu kompilace softwarových projektů. Používá makefile, textový soubor, který určuje závislosti mezi zdrojovými soubory a příkazy potřebnými k jejich kompilaci. Make automatizuje proces kompilace zdrojového kódu a zajišťuje, aby se při změnách překompilovaly pouze potřebné soubory, čímž se zkracuje doba kompilace a zvyšuje se efektivita (14).

Jednou z hlavních výhod programu Make je jeho flexibilita a možnost přizpůsobení. Vývojáři mohou vytvářet složité soubory make, které určují pořadí kompilace a možnosti kompilace jednotlivých souborů. Kromě toho lze Make použít k automatizaci dalších úloh, jako je testování, generování dokumentace a nasazení. Další výhodou Make je jeho přenositelnost. Make je k dispozici na různých platformách, včetně Linuxu, Windows a MacOS (14).

2.4.4 Apache Maven

Apache Maven je nástroj pro automatizaci sestavování, který se používá především pro projekty v jazyce Java. Zjednodušuje proces sestavování tím, že poskytuje sadu předdefinovaných šablon projektů. Maven používá deklarativní přístup ke konfiguraci sestavení, který vývojářům umožňuje specifikovat závislosti projektu a požadavky na sestavení v jediném konfiguračním souboru (15).

Výhodou nástroje Maven je jeho integrace s dalšími nástroji a platformami. Maven lze používat s oblíbenými vývojovými prostředími, jako jsou Eclipse a IntelliJ IDEA, a lze jej také integrovat s nástroji pro kontinuální integraci, jako jsou Jenkins a Travis CI. To z něj činí ideální volbu pro vytváření a správu projektů Java v prostředí společného vývoje (15).

2.5 Vykreslování webu

V oblasti vývoje webových stránek existují různé způsoby vykreslování obsahu na webové stránce. Tři běžné metody jsou vykreslování na straně klienta (anglicky client-side rendering), vykreslování na straně serveru (anglicky server-side rendering) a předběžné vykreslování (anglicky pre-rendering). Každá metoda má své výhody a nevýhody. Níže budou popsány vlastnosti a rozdíly jednotlivých metod (16).

2.5.1 Vykreslování na straně klienta

Vykreslování na straně klienta je metoda, která zahrnuje vykreslování webových stránek na straně klienta pomocí frameworků nebo knihoven jazyka JavaScript, jako je React nebo Angular. Klient odešle požadavek na server a server odpoví potřebnými daty ve formátu JSON. Klient poté vykreslí stránku pomocí dat přijatých ze serveru. Tato metoda je rychlá, protože vyžaduje, aby klient stahoval pouze kód JavaScriptu, a nikoli celý dokument HTML. Umožňuje také vytvářet dynamický obsah a interaktivitu. Může však způsobit pomalé počáteční načítání a nižší optimalizaci pro vyhledávače (SEO), protože vyhledávače nevidí obsah (16).

2.5.2 Vykreslování na straně serveru

Vykreslování na straně serveru zahrnuje vykreslení webové stránky na serveru a odeslání plně vykresleného dokumentu HTML klientovi. Tato metoda vyžaduje, aby klient odeslal požadavek na server, a server odpoví plně vykresleným dokumentem HTML. Tato metoda je výhodná pro SEO, protože vyhledávače mohou obsah přečíst. Poskytuje také rychlejší počáteční načítání, protože klient nemusí před vykreslením stránky stahovat kód JavaScriptu. Nemusí však být tak citlivá a interaktivní jako vykreslování na straně klienta a může vyžadovat další prostředky serveru (16) (17).

2.5.3 Předběžné vykreslování

Předběžné vykreslování je metoda, která zahrnuje vykreslení webové stránky na serveru a uložení výsledku do mezipaměti. Když klient požádá o stránku, server odpoví dokumentem HTML uloženým v mezipaměti. Tato metoda je výhodná pro SEO, protože vyhledávače mohou obsah přečíst, a také zajišťuje rychlé počáteční načítání, protože klient nemusí čekat, až server stránku vykreslí. Je také užitečná pro jednostránkové aplikace, které nevyžadují časté změny obsahu. Nemusí však být tak dynamická a interaktivní jako vykreslování na straně klienta a může vyžadovat další prostředky serveru pro ukládání do mezipaměti (18).

Dalším možným vývojem je využití edge computingu k zajištění vykreslování na straně serveru blíže ke klientovi. Tento přístup může snížit latenci mezi serverem a klientem, což může být výhodné zejména pro uživatele, kteří se nacházejí daleko od serveru. Edge computing může také pomoci snížit zatížení serveru a zajistit rychlejší odezvu (18).

Závěrem lze říci, že volba metody vykreslování by měla záviset na konkrétních požadavcích webové aplikace. Vykreslování na straně klienta může být rychlé a citlivé, ale může způsobit pomalé počáteční načítání a nižší SEO. Vykreslování na straně serveru je výhodné pro SEO a poskytuje rychlejší počáteční načítání, ale může vyžadovat další prostředky serveru. Předběžné vykreslování je užitečné pro SEO a poskytuje rychlé počáteční načítání, ale nemusí být tak dynamické a interaktivní jako vykreslování na straně klienta a může vyžadovat další serverové zdroje pro ukládání do mezipaměti. Hybridní přístupy a využití edge computingu mohou přinést další zlepšení vykreslování webových aplikací.

2.6 Webové technologie

2.6.1 Javascript

JavaScript je vysokoúrovňový programovací jazyk, který se hojně používá k vytváření dynamických a interaktivních webových stránek. Je to univerzální jazyk, který vývojářům umožňuje vytvářet interaktivní uživatelská rozhraní, manipulovat s daty a vytvářet výkonné webové aplikace (19).

Vlastnosti jazyka JavaScript

JavaScript je dynamicky typovaný jazyk, což znamená, že proměnné nemají specifikovaný datový typ. Je to také objektově orientovaný jazyk, což znamená, že k ukládání dat a metod používá objekty. JavaScript má také možnosti funkcionálního programování, které vývojářům umožňují vytvářet opakovaně použitelné funkce a funkce vyššího řádu. Podporuje asynchronní programování, což vývojářům umožňuje vytvářet neblokující kód, který může zpracovávat více požadavků současně (19) (20).

Využití jazyka JavaScript

JavaScript se používá především pro programování na straně klienta, které zahrnuje manipulaci s uživatelským rozhraním a obsluhu uživatelských událostí. Používá se také pro programování na straně serveru, které zahrnuje manipulaci s daty a logikou na straně serveru. JavaScript se běžně používá k vytváření webových aplikací, jako jsou platformy sociálních médií, weby elektronického obchodování a online hry. Lze jej použít také k vytváření mobilních aplikací pomocí frameworků, jako jsou React Native a Ionic (20).

2.6.2 HTML

HTML neboli Hypertext Markup Language je standardní značkovací jazyk používaný pro vytváření a strukturování webového obsahu. Jazyk HTML se používá v kombinaci s dalšími webovými technologiemi, jako jsou CSS a JavaScript, k vytváření vizuálně atraktivních a interaktivních webových stránek (21).

Syntaxe jazyka HTML

Jazyk HTML se skládá z řady značek, které se používají k definování struktury webové stránky. Značky jsou uzavřeny ve špičatých závorkách (< >) a slouží k identifikaci různých prvků na webové stránce. Například značka <html> se používá k definování začátku dokumentu HTML, zatímco značka <body> slouží k definování hlavního obsahu webové stránky. Značky mohou obsahovat také atributy, které poskytují další informace o daném prvku. Například značka <img> se používá k zobrazení obrázků a obsahuje atribut pro zdrojový soubor obrázku (22) (23).

Úloha jazyka HTML při vývoji webových stránek

Jazyk HTML je základem celého vývoje webových stránek. Poskytuje základní strukturu a obsah webové stránky, který je následně stylizován pomocí CSS a JavaScriptu. Jazyk HTML je nezbytný pro vytváření přístupných webových stránek, protože poskytuje jasnou strukturu,

kterou mohou interpretovat čtečky obrazovky a další asistenční technologie. Bez jazyka HTML by nebylo možné vytvářet složité a interaktivní webové aplikace, které dnes používáme (21).

2.6.3 CSS

Kaskádové styly neboli CSS je jazyk stylů používaný k popisu vizuálního vzhledu webových stránek napsaných v jazycích HTML a XHTML. CSS poskytuje způsob, jak oddělit prezentaci dokumentu od jeho obsahu, což umožňuje větší flexibilitu a kontrolu nad designem webových stránek (24).

Klíčové vlastnosti
CSS poskytuje výkonnou sadu nástrojů pro ovládání vzhledu webových stránek. Mezi jeho klíčové funkce patří:

- Selektory: CSS používá selektory k zaměření konkrétních prvků na webové stránce a použití stylů na ně. Selektory mohou být založeny na typu prvku, třídě, ID, atributu a dalších.
- Kaskádování: CSS je navržen jako kaskádový jazyk stylů, což znamená, že styly lze aplikovat na různých úrovních (např. globálně, lokálně, inline) a podle potřeby se kaskádově přenášejí na podřízené prvky.
- Dědičnost: CSS také podporuje dědičnost, což znamená, že styly použité na rodičovský prvek budou automaticky zděděny jeho podřízenými prvky.
- Box model: CSS poskytuje box model pro rozvržení, který vývojářům umožňuje řídit velikost a umístění prvků na webové stránce.
- Dotazy na velikost: CSS podporuje dotazy, které vývojářům umožňují vytvářet responzivní návrhy, které se přizpůsobují různým velikostem obrazovky a zařízení (25).

2.6.4 JSON

JavaScript Object Notation (JSON) je lehký formát pro výměnu dat, který je stále populárnější při vývoji webových stránek. JSON poskytuje jednoduchý a flexibilní způsob reprezentace a výměny dat mezi různými aplikacemi a platformami (26).

Klíčové vlastnosti

JSON poskytuje řadu klíčových vlastností, díky nimž se dobře hodí pro výměnu dat mezi různými aplikacemi a platformami. Mezi jeho klíčové vlastnosti patří:

- Nenáročnost: JSON je jednoduchý formát, který lze snadno analyzovat a přenášet po síti. Díky tomu je vhodný pro použití ve webových aplikacích, kde je klíčový výkon a rychlost.
- Čitelný pro člověka: JSON je navržen tak, aby byl čitelný pro člověka a snadno srozumitelný. To usnadňuje vývojářům práci s ním a jeho ladění. Také jej to činí přístupnějším pro netechnické zainteresované strany.
- Flexibilní: JSON poskytuje flexibilní datový model, který může reprezentovat širokou škálu datových struktur, včetně polí, objektů a vnořených dat. Díky tomu se dobře hodí pro použití v různých aplikacích a platformách.

- Nezávislost na jazyce: JSON je formát nezávislý na jazyce, který lze použít v široké škále programovacích jazyků a platforem. To usnadňuje integraci s různými aplikacemi a službami (26) (27).

Vliv JSON na vývoj webových aplikací

Formát JSON má významný vliv na vývoj webových aplikací, protože umožňuje vývojářům vyměňovat data mezi různými aplikacemi a platformami jednoduchým a flexibilním způsobem. Díky tomu, že JSON poskytuje lehký a lidsky čitelný formát pro výměnu dat, pomohl zjednodušit a zefektivnit mnoho aspektů vývoje webových aplikací.

Jednou z klíčových oblastí, kde měl JSON vliv, je vývoj rozhraní API. Rozhraní API poskytují různým aplikacím a službám možnost vzájemné komunikace a JSON se stal oblíbeným formátem pro výměnu dat mezi rozhraními API a aplikacemi JavaScript na straně klienta (28).

2.6.5 XML

XML (Extensible Markup Language) je značkovací jazyk, který se široce používá pro reprezentaci a výměnu dat na webu (29).

Syntaxe

Syntaxe jazyka XML je založena na souboru pravidel pro vytváření značkovacích jazyků. Dokumenty XML se skládají z elementů, které jsou ohraničeny špičatými závorkami. Každý prvek může mít atributy sloužící k poskytnutí dalších informací o prvku. Prvky mohou také obsahovat další prvky, čímž vytvářejí hierarchickou strukturu dat. Dokumenty XML musí mít kořenový element, který obsahuje všechny ostatní elementy v dokumentu (29).

Datové typy

XML podporuje několik datových typů, včetně:

- String: posloupnost znaků.
- Number: číselná hodnota, která může být buď celé číslo, nebo číslo s pohyblivou desetinnou čárkou.
- Boolean: pravdivá nebo nepravdivá hodnota.
- Null: speciální hodnota, která představuje neexistenci hodnoty.
- Date a time: hodnota data a času reprezentovaná ve specifickém formátu.
- Binární data: binární data reprezentovaná jako posloupnost bajtů.
- Vlastní datové typy: XML umožňuje vytvářet vlastní datové typy, které reprezentují specifické datové struktury (30).

Použití XML jako formátu pro výměnu dat má několik výhod:

- Flexibilita: XML je flexibilní formát, který lze použít k reprezentaci široké škály datových typů.
- Samopopisný: Dokumenty XML jsou samopopisné, což znamená, že obsahují všechny informace potřebné k interpretaci jejich obsahu.
- Nezávislost na platformě: XML je nezávislý na platformě, což znamená, že jej lze použít s jakýmkoli programovacím jazykem nebo platformou.

- Průmyslový standard: XML je průmyslový standardní formát, který se široce používá pro výměnu dat v různých odvětvích.
- Široce podporovaný: XML je široce podporován moderními webovými prohlížeči a je používán mnoha webovými rozhraními API (30).

XML má mnoho způsobů použití, například:

- Webová rozhraní API: XML se používá k přenosu dat mezi webovými službami a webovými aplikacemi, takže je ideálním formátem pro webová rozhraní API.
- Ukládání dat: XML se používá k ukládání dat v databázích a dalších systémech pro ukládání dat.
- Konfigurační soubory: XML se používá k ukládání konfiguračních dat pro webové aplikace a další software.
- Správa dokumentů: XML se používá k ukládání a výměně dokumentů mezi různými systémy.
- Publikování: XML se používá při publikování k oddělení obsahu od prezentace, což umožňuje efektivnější produkci a distribuci obsahu (30).

2.6.6 Django

Django je populární a výkonný webový framework vyvinutý v jazyce Python. Jedná se o open-source framework, který poskytuje sadu nástrojů a funkcí pro vytváření webových aplikací. Byl navržen tak, aby poskytoval snadno použitelný a efektivní webový framework, který by vývojářům pomohl rychle vytvářet složité webové aplikace. Od svého vydání si Django získalo popularitu a stal se jedním z nejpoužívanějších webových frameworků na světě (31).

Klíčové vlastnosti

Django poskytuje širokou škálu vlastností a funkcí, které z něj činí výkonný a oblíbený webový framework. Mezi jeho klíčové funkce patří:

- Objektově-relační mapování (ORM): Django poskytuje výkonné ORM, které umožňuje vývojářům komunikovat s databázemi pomocí kódu Pythonu. To usnadňuje práci s databázemi a odstraňuje potřebu ručních dotazů SQL.
- Rozhraní správce: Django je vybaveno rozhraním pro správu: Django poskytuje vestavěné administrátorské rozhraní, které usnadňuje správu dat a obsahu webových aplikací. Rozhraní správce lze přizpůsobit potřebám aplikace.
- Směrování adres URL: Django disponuje systémem směrování URL, který vývojářům umožňuje mapovat adresy URL na konkrétní funkce v aplikaci.
- Systém šablon: Django poskytuje robustní systém šablon, který vývojářům umožňuje vytvářet dynamické, opakovaně použitelné šablony HTML pro jejich webové aplikace.
- Zabezpečení: Django poskytuje řadu bezpečnostních funkcí, včetně ochrany proti běžným webovým zranitelnostem, jako je cross-site scripting (XSS) a cross-site request forgery (CSRF) (32).

Vliv Djanga na vývoj webových aplikací

Django má od svého vydání významný vliv na vývoj webových aplikací. Vývojářům usnadnilo a urychlilo vytváření složitých webových aplikací. Obliba Djanga v průběhu let výrazně vzrostla a nyní je jedním z nejpoužívanějších webových frameworků na světě. Jednou z klíčových oblastí, kde má Django vliv, je vývoj systémů pro správu obsahu (CMS). Django poskytuje výkonnou sadu nástrojů pro správu obsahu, včetně vestavěného administrátorského rozhraní a robustního ORM. Mnoho populárních platforem CMS, jako jsou Wagtail a Mezzanine, je postaveno na platformě Django. Django mělo také vliv na vývoj platforem pro elektronické obchodování. Poskytuje širokou škálu funkcí a vlastností, které usnadňují vytváření komplexních aplikací pro elektronické obchodování. Mezi oblíbené platformy pro elektronické obchodování postavené na Djangu patří Saleor a Oscar (33) (34).

2.6.7 Flask

Flask je populární webový framework v jazyce Python, který se používá k vytváření webových aplikací. Je lehký a flexibilní, takže je snadné s ním začít pracovat a je ideální pro vytváření malých až středně velkých aplikací (35).

Klíčové vlastnosti

Flask má několik zásadních funkcí, díky kterým je oblíbenou volbou pro vytváření webových aplikací. Jednou z jeho klíčových vlastností je jeho jednoduchost. Flask poskytuje jednoduché a snadno pochopitelné rozhraní pro vytváření webových aplikací, což vývojářům usnadňuje začátky používání a rychlé vytváření aplikací (36).

Další klíčovou vlastností Flasku je jeho flexibilita. Flask umožňuje vývojářům přidávat další funkce podle potřeby, což usnadňuje přizpůsobení a rozšíření aplikací. Flask také podporuje řadu rozšíření třetích stran, která lze použít k přidání dalších funkcí do aplikací, jako je například ověřování a integrace s databází (36).

Flask je také známý svou rozsáhlou a komplexní dokumentací, která vývojářům usnadňuje učení a používání frameworku. Dokumentace frameworku Flask poskytuje jasné a podrobné vysvětlení jeho funkcí a způsobů jejich použití (36).

Vliv na vývoj webových aplikací

Popularita Flasku vedla také k rozvoji živého ekosystému rozšíření a zásuvných modulů, které lze použít k rozšíření jeho funkcí. Mezi oblíbená rozšíření Flask patří Flask-SQLAlchemy, Flask-Login a Flask-WTF (37).

Vliv Flasku na vývoj webových aplikací je patrný také z jeho využití v průmyslu. Mnoho společností, včetně Uberu, Netflixu a Lyftu, používá Flask k vytváření svých webových aplikací. Díky své jednoduchosti a flexibilitě je Flask ideální volbou pro společnosti, které potřebují rychle vytvářet webové aplikace (37).

2.6.8 NodeJS

NodeJS je open-source, multiplatformní, back-endové běhové prostředí JavaScriptu. Je určeno ke spouštění kódu JavaScriptu mimo webový prohlížeč a umožňuje vývojářům vytvářet

aplikace v JavaScriptu na straně serveru. NodeJS je postaven nad JavaScriptovým frameworkem V8 (38).

Klíčové vlastnosti

NodeJS má několik klíčových vlastností, díky nimž je oblíbenou volbou pro vytváření aplikací na straně serveru. Jednou z jeho klíčových vlastností je neblokující model I/O, který umožňuje NodeJS zpracovávat velké množství souběžných připojení, aniž by blokoval provádění jiného kódu. Díky tomu je ideální pro vytváření aplikací pracujících v reálném čase, jako jsou chatovací aplikace a online hry (39).

Další klíčovou vlastností NodeJS je jeho modulární architektura. NodeJS používá modulární přístup, kdy každý modul představuje samostatnou jednotku kódu, kterou lze opakovaně použít v různých aplikacích. Vývojáři tak mohou snadno vytvářet složité aplikace kombinováním různých modulů. NodeJS má také živou komunitu, která poskytuje řadu knihoven a frameworků rozšiřujících jeho možnosti. Mezi oblíbené frameworky NodeJS patří Express, Koa a NestJS (40).

Vliv na vývoj webových aplikací

NodeJS má významný vliv na vývoj webových aplikací. Umožnil vytvářet aplikace na straně serveru pomocí jazyka JavaScript, který zná mnoho webových vývojářů. To vedlo k výraznému nárůstu počtu vývojářů v jazyce JavaScript a popularitě jazyka JavaScript jako jazyka na straně serveru (38).

2.6.9 PHP

PHP je populární open-source skriptovací jazyk na straně serveru používaný pro vývoj webových stránek. Je navržen tak, aby byl jednoduchý a snadno použitelný, což je ideální pro vytváření dynamických a interaktivních webových stránek (41).

Klíčové vlastnosti jazyka

Jazyk PHP má několik klíčových vlastností, díky kterým je oblíbenou volbou pro vývoj webových stránek. Jednou z jeho klíčových vlastností je snadné používání.

Další klíčovou vlastností jazyka PHP je jeho flexibilita. Pomocí jazyka PHP lze vytvářet širokou škálu webových aplikací, od jednoduchých blogů až po složité webové stránky elektronických obchodů. PHP také podporuje řadu databází, včetně MySQL, Oracle a PostgreSQL, což usnadňuje integraci se stávajícími systémy. Jazyk PHP je také známý svou velkou a aktivní komunitou vývojářů. Komunita PHP vytvořila rozsáhlý ekosystém nástrojů, knihoven a frameworků, které lze použít k rozšíření funkcí jazyka PHP a k rychlejšímu a efektivnějšímu vývoji webových stránek (42).

Vliv na vývoj webových stránek

Jazyk PHP je také široce podporován webhostingovými společnostmi. Vliv jazyka PHP na vývoj webových stránek je patrný také ve vývoji populárních systémů pro správu obsahu (CMS), jako jsou WordPress, Drupal a Joomla. Tyto systémy CMS jsou postaveny nad

systémem PHP a poskytují uživatelsky přívětivé rozhraní pro správu a publikování webového obsahu (43).

2.6.10 ASP.NET

ASP.NET je výkonná technologie Microsoftu určená pro vytváření webových aplikací běžících na straně serveru (44).

Klíčové funkce

ASP.NET má několik klíčových vlastností, které z něj činí oblíbenou volbu pro vývoj webových stránek. Jednou z jeho klíčových vlastností je schopnost podporovat více programovacích jazyků, včetně jazyků C#, Visual Basic a F#. Vývojáři si tak mohou snadno zvolit jazyk, který jim nejvíce vyhovuje, a přesto mohou framework používat. Pomocí ASP.NET lze vytvářet vše od jednoduchých webových stránek až po složité webové aplikace s pokročilými funkcemi, jako je ověřování uživatelů, přístup k datům a podobně (44).

ASP.NET je také známý svým výkonem a škálovatelností. Tento framework je optimalizován pro výkon a disponuje funkcemi, jako je kompilace just-in-time (JIT) a ukládání do mezipaměti, které pomáhají zvýšit rychlost a odezvu webových aplikací. ASP.NET také podporuje clustering a vyvažování zátěže, což usnadňuje škálování webových aplikací pro zvládnutí velkého objemu provozu (44) (45).

Vliv na vývoj webových aplikací

Vliv ASP.NET na vývoj webových aplikací je patrný také z jeho širokého rozšíření a použití v mnoha populárních webových aplikacích. Samotná společnost Microsoft používá ASP.NET pro mnoho svých vlastních webových aplikací. Dále ASP.NET používá také společnost Slack, Alibaba, MasterCard nebo SpaceX (46).

2.7 Výuka programování

Programování je základní dovednost, která v moderní digitální éře získala značný význam. Používá se v různých oblastech, včetně vývoje softwaru, robotiky, analýzy dat, umělé inteligence a tvorby webových stránek. V důsledku toho se v různých odvětvích zvyšuje poptávka po odbornících s dovednostmi v oblasti programování. Aby tuto mezeru překlenuly, začaly mnohé země zavádět programování jako předmět do škol (47).

2.7.1 Programování ve školách

Programování se ve školách vyučuje různými metodami zahrnujícími učebnice, výuková videa, online kurzy a přednášky. Studenti se učí programovat pomocí programovacích jazyků, jako jsou Python, Java, C/C++ a JavaScript. Seznamují se také s pojmy z oblasti programování, jako jsou proměnné, funkce, cykly, podmíněné příkazy a datové struktury (47).

Podle Národních osnov pro Anglii z roku 2013 je programování zahrnuto do předmětu Computing, který je povinným předmětem pro všechny žáky od pěti do šestnácti let. Učební plán uvádí, že žáci by se měli naučit navrhovat, psát a ladit programy, které dosahují konkrétních cílů. Učební osnovy také zdůrazňují význam informatického myšlení, což je

dovednost řešení problémů, která spočívá v rozdělení složitých problémů na menší, lépe zvládnutelné části (48).

Ve Spojených státech se programování zavádí i do škol. Asociace učitelů informatiky (Computer Science Teachers Association, CSTA) vypracovala soubor standardů pro výuku informatiky a programování. Tyto standardy zahrnují témata, jako jsou algoritmy, programovací jazyky a řešení problémů. CSTA rovněž poskytuje učitelům zdroje pro tvorbu jejich učebních osnov a hodnocení (49).

2.7.2 Zadání pro programování

Programovací úlohy jsou nezbytnou součástí výuky programování ve školách. Pomáhají studentům aplikovat naučené koncepty programování při řešení reálných problémů. Programovací úlohy také pomáhají studentům rozvíjet kritické myšlení, zlepšují jejich schopnosti řešit problémy a zvyšují jejich kreativitu (47).

Jedním z typů programovacích úkolů je kódovací úkol, kdy studenti dostanou zadání problému a mají napsat kód, který tento problém vyřeší. Studenti mohou být například požádáni, aby napsali program, který vypočítá součet všech sudých čísel mezi dvěma zadanými čísly. Kódovací úlohy se často používají v soutěžích v programování, jako je například International Collegiate Programming Contest (ICPC), kde se testují programátorské dovednosti studentů (50).

Dalším typem programovacích úloh jsou projektové úlohy, kdy studenti pracují na větším projektu po delší dobu. Studenti mohou být například požádáni, aby vytvořili hru nebo webovou aplikaci. Projektové úkoly umožňují studentům uplatnit své programátorské dovednosti na větším projektu a naučit se dovednostem řízení projektu, týmové práci a komunikačním dovednostem (51).

Programátorské úkoly lze také použít k hodnocení toho, jak studenti rozumí programátorským konceptům. Studenti mohou být například požádáni, aby napsali program, který prokáže jejich porozumění určitému programovacímu konceptu, jako jsou smyčky nebo podmíněné příkazy (51).

2.8 Vývoj software

Proces vývoje softwaru zahrnuje řadu činností, včetně shromažďování požadavků, návrhu, kódování, testování a údržby.

2.8.1 Shromažďování požadavků

Shromažďování požadavků je první fází procesu vývoje softwaru. Tato fáze je klíčová, protože vytváří základ pro celý proces vývoje. Fáze shromažďování požadavků zahrnuje spolupráci se zúčastněnými stranami s cílem identifikovat potřeby systému. To může zahrnovat pochopení obchodních požadavků, potřeb uživatelů a technických požadavků. Cílem této fáze je vytvořit jasný a komplexní soubor požadavků, jimiž se bude řídit proces vývoje (52).

2.8.2 Návrh

Ve fázi návrhu vytváří tým pro vývoj softwaru podrobný plán architektury systému, návrhu databáze a uživatelského rozhraní. Fáze návrhu zahrnuje tvorbu diagramů, vývojových diagramů a další dokumentace, která nastiňuje strukturu softwarové aplikace. Cílem této etapy je vytvořit podrobný plán, kterým se lze řídit ve fázi implementace (52).

2.8.3 Implementace

Ve fázi implementace tým vývojářů softwaru píše kód, který tvoří softwarovou aplikaci. Tato fáze je obvykle nejdelší a nejsložitější a zahrnuje řadu programovacích jazyků, knihoven a frameworků. Fáze implementace může zahrnovat spolupráci s více vývojáři, testery a projektovými manažery. Cílem této fáze je vytvořit kvalitní kód, který splňuje požadavky nastíněné ve fázích sběru požadavků a návrhu (52).

2.8.4 Testování

Ve fázi testování tým vývojářů testuje software, aby se ujistil, že splňuje požadavky a funguje tak, jak má. Fáze testování může zahrnovat ruční testování, automatizované testování nebo kombinaci obou. Cílem této fáze je identifikovat a opravit případné chyby nebo problémy v softwaru před jeho nasazením do výroby (52).

2.8.5 Nasazení

Ve fázi nasazení je software nasazen do výroby, kde jej používají koncoví uživatelé. Fáze nasazení může zahrnovat spolupráci s týmy infrastrukturního oddělení při nastavování serverů, konfiguraci databází a zajištění toho, aby byl software přístupný koncovým uživatelům. Cílem této fáze je zajistit, aby byl software stabilní, bezpečný a splňoval potřeby koncových uživatelů (52).

2.8.6 Klíčové pojmy při vývoji softwaru

Existuje několik klíčových pojmů, které jsou pro vývoj softwaru zásadní. Patří mezi ně programovací jazyky, knihovny a rámce, řízení verzí a agilní vývoj.

2.8.7 Programovací jazyky

Vývojáři softwaru používají programovací jazyky k psaní kódu, který je prováděn počítačem. K dispozici je široká škála programovacích jazyků, z nichž každý má své silné a slabé stránky. Mezi oblíbené programovací jazyky patří např.:

- Java: Vysokoúrovňový programovací jazyk, který se široce používá pro podnikové aplikace, webové aplikace a mobilní aplikace.
- Python: Vysokoúrovňový programovací jazyk, který se hojně používá pro analýzu dat, vědecké výpočty a strojové učení.
- C: nízkoúrovňový programovací jazyk, který se široce používá pro programování věstavěných systémů.
- C++: rozšířená verze jazyka C, používá se na programování věstavěných systémů a vývoj her. Podporuje práci s objekty.
- JavaScript: Vysokoúrovňový programovací jazyk, který se hojně používá pro vývoj webových stránek a front-end (54).

2.8.8 Knihovny a frameworky

Vývojáři softwaru používají knihovny a frameworky, aby urychlili proces vývoje a snížili množství kódu, který je třeba napsat. Knihovny a rámce poskytují předpřipravené funkce a moduly, které lze použít ke zpracování běžných úloh, jako je přístup k databázi, ověřování uživatelů a návrh uživatelského rozhraní. Mezi oblíbené knihovny a frameworky patří např:

- React: Knihovna JavaScriptu pro tvorbu uživatelských rozhraní.
- Angular: framework založený na TypeScriptu pro vytváření webových aplikací.
- Django: framework pro tvorbu webových aplikací založený na Pythonu.
- Spring: framework založený na Javě pro tvorbu podnikových aplikací (54).

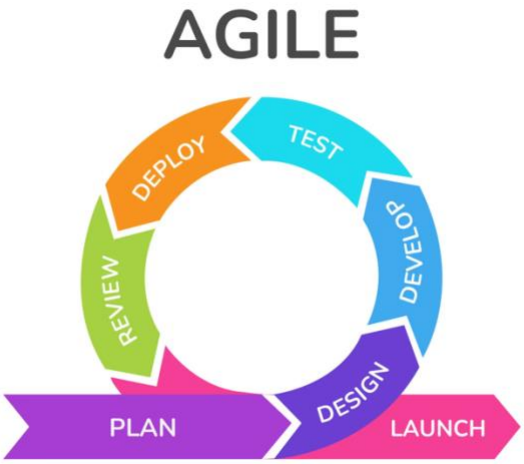
2.8.9 Řízení verzí

Řízení verzí je proces sledování změn zdrojového kódu v průběhu času. Systémy pro správu verzí umožňují vývojářům spolupracovat na kódu, sledovat změny a v případě potřeby se vrátit k předchozím verzím. Mezi oblíbené systémy pro správu verzí patří např.:

- Git: distribuovaný systém pro správu verzí, který se široce používá pro vývoj open source a podnikový vývoj softwaru.
- SVN: Centralizovaný systém pro správu verzí, který je široce používán pro vývoj podnikového softwaru (55).

2.8.10 Agilní vývoj

Agilní vývoj je iterativní přístup k vývoji softwaru, který klade důraz na spolupráci, flexibilitu a neustálé zlepšování.

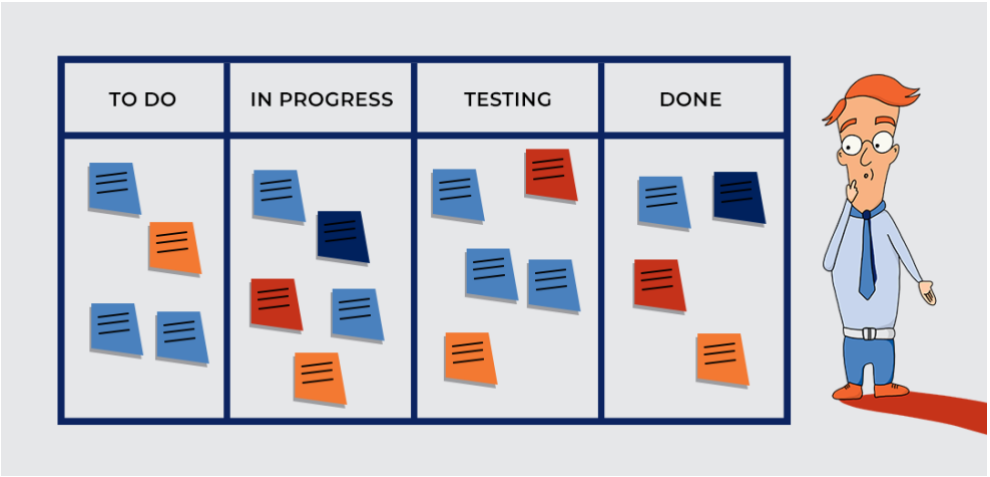


Obrázek 1 Schéma agilního vývoje (53)

Agilní vývoj zahrnuje rozdělení procesu vývoje na malé, zvládnutelné části a průběžné testování a zdokonalování softwaru v průběhu jeho vývoje (53). Mezi oblíbené metodiky agilního vývoje patří např:

- Scrum: agilní rámec, který klade důraz na týmovou práci, odpovědnost a iterativní vývoj.

- Kanban: agilní rámec, který klade důraz na vizualizaci práce, omezování rozpracované práce a průběžné dodávání, viz obrázek č. 2 (56).



Obrázek 2 Tabule Kanban (56)

2.9 Testování Softwaru

Testování softwaru je proces ověřování a validace softwarových aplikací nebo systémů s cílem zajistit, aby neobsahovaly chyby, splňovaly stanovené požadavky a fungovaly tak, jak bylo zamýšleno. Účelem testování softwaru je identifikovat a zmírnit potenciální problémy, které by mohly ovlivnit výkon, funkčnost a uživatelský komfort softwaru.

2.9.1 Význam testování softwaru

Testování softwaru hraje klíčovou roli při zajišťování kvality a spolehlivosti softwarových aplikací. Testování pomáhá identifikovat problémy a vady v rané fázi vývojového cyklu softwaru, což může snížit náklady na pozdější opravu chyb. Je také rozhodujícím krokem k zajištění toho, aby softwarové aplikace splňovaly stanovené požadavky a fungovaly tak, jak bylo zamýšleno (57).

Testování softwaru je nezbytné z řady důvodů, mezi něž patří např.:

- Zlepšení kvality: Testování softwaru pomáhá identifikovat a odstraňovat vady, což vede k vyšší kvalitě softwaru. Testování také pomáhá zajistit, aby softwarové aplikace splňovaly stanovené požadavky a fungovaly tak, jak bylo zamýšleno.
- Snížení nákladů: Identifikace a odstranění chyb v rané fázi vývojového cyklu softwaru je méně nákladné než jejich pozdější identifikace a odstranění. Testování pomáhá snížit náklady na opravu problémů tím, že je identifikuje v rané fázi vývojového cyklu.
- Lepší uživatelská zkušenost: Testování pomáhá identifikovat a opravit problémy, které by mohly ovlivnit uživatelský komfort softwarových aplikací. To pomáhá zajistit, aby uživatelé měli při používání softwarových aplikací pozitivní zkušenosti.
- Dodržování předpisů: V některých odvětvích, například ve zdravotnictví a finančnictví, je testování softwaru zákonným požadavkem pro zajištění souladu s oborovými normami a předpisy (58).

2.9.2 Typy testování softwaru

Existuje několik typů testování softwaru, které lze použít k identifikaci a odstranění vad softwarových aplikací. Mezi nejběžnější typy testování softwaru patří:

- Jednotkové testování: Testování jednotek je typ testování, který se zaměřuje na jednotlivé jednotky nebo součásti softwarových aplikací. Testy jednotek jsou obvykle automatizované a jsou určeny k identifikaci problémů s konkrétními částmi kódu.
- Integrační testování: Integrační testování je typ testování, který se zaměřuje na to, jak různé jednotky nebo součásti softwarových aplikací spolupracují. Integrační testování pomáhá identifikovat problémy, které vznikají při integraci různých součástí softwarových aplikací.
- Systémové testování: Systémové testování je typ testování, který hodnotí celou softwarovou aplikaci. Systémové testování pomáhá zajistit, aby softwarová aplikace splňovala stanovené požadavky a fungovala tak, jak bylo zamýšleno.
- Akceptační testování: Akceptační testování je typ testování, které provádějí koncoví uživatelé, aby se ujistili, že softwarová aplikace splňuje jejich potřeby a požadavky.
- Regresní testování: Regresní testování je typ testování, které se provádí s cílem zajistit, aby změny provedené v softwarových aplikacích nezpůsobily nové vady nebo problémy (59).

2.9.3 Osvědčené postupy pro efektivní testování softwaru

Efektivní testování softwaru vyžaduje strukturovaný přístup a dodržování osvědčených postupů. Mezi osvědčené postupy pro efektivní testování softwaru patří:

- Plánování testů: Správné plánování testů je pro efektivní testování softwaru nezbytné. Plánování testování zahrnuje určení cílů testování, definování rozsahu testování a vytvoření plánu testování, který popisuje přístup k testování, testovací případy a očekávané výsledky.
- Automatizace testování: Automatizace testování může pomoci zlepšit účinnost a efektivitu testování softwaru. Automatizace testování zahrnuje použití nástrojů a frameworků k automatizaci opakujících se a časově náročných úloh.
- Správa testovacích dat: Správa testovacích dat zahrnuje vytváření a správu testovacích dat, která jsou reprezentativní pro reálné scénáře. Správa testovacích dat pomáhá zajistit, aby byly softwarové aplikace testovány v reálných podmínkách, což vede k přesnějším a spolehlivějším výsledkům testů.
- Průběžné testování: Kontinuální testování zahrnuje integraci testování do procesu vývoje softwaru. Průběžné testování pomáhá zajistit, aby byly vady identifikovány a odstraněny v rané fázi vývojového cyklu, což vede k vyšší kvalitě softwarových aplikací.
- Testování výkonnosti: Testování výkonu je typ testování, který hodnotí výkonnost a škálovatelnost softwarových aplikací. Testování výkonnosti pomáhá identifikovat problémy s výkonností a škálovatelností. softwarových aplikací a zajistit, aby zvládaly očekávanou pracovní zátěž.
- Testování bezpečnosti: Bezpečnostní testování je typ testování, který hodnotí bezpečnost softwarových aplikací. Bezpečnostní testování pomáhá identifikovat

zranitelnosti a slabiny softwarových aplikací a zajišťuje jejich bezpečnost a ochranu před potenciálními hrozbami.
- Testování použitelnosti: Je typ testování, který hodnotí uživatelský komfort softwarových aplikací. Testování použitelnosti pomáhá zajistit, aby se softwarové aplikace snadno používaly a navigovaly, což vede k pozitivním uživatelským zkušenostem (59).

2.9.4 Závěr

Testování softwaru je kritický proces při vývoji softwaru, který zajišťuje, že softwarové aplikace neobsahují chyby, splňují stanovené požadavky a fungují tak, jak bylo zamýšleno. Efektivní testování softwaru vyžaduje strukturovaný přístup a dodržování osvědčených postupů, včetně správného plánování testů, automatizace testů, správy testovacích dat, průběžného testování, testování výkonu, testování zabezpečení a testování použitelnosti. Dodržováním těchto osvědčených postupů mohou týmy zabývající se vývojem softwaru zajistit, aby softwarové aplikace byly kvalitní, spolehlivé a splňovaly potřeby a požadavky koncových uživatelů.

Testování softwaru je navíc kontinuální proces, který začíná v raných fázích vývoje softwaru a pokračuje po celou dobu životního cyklu vývoje softwaru. Tím je zajištěno, že případné vady nebo problémy jsou odhaleny a vyřešeny co nejdříve, což vede k výrazným úsporám nákladů a zlepšení kvality.

Testování softwaru je složitý proces, který vyžaduje specializované dovednosti a znalosti. Proto je nezbytné mít tým zkušených a kvalifikovaných softwarových testerů, kteří dokáží efektivně plánovat, provádět a řídit činnosti testování softwaru.

Na závěr je třeba poznamenat, že význam testování softwaru nelze podceňovat. Softwarové aplikace jsou kritickými součástmi mnoha podniků a organizací a náklady na selhání softwaru mohou být značné, pokud jde o ušlé příjmy, poškození pověsti a právní závazky.

2.10 Moodle

Moodle je bezplatný systém pro správu výuky (LMS) s otevřeným zdrojovým kódem, který je navržen tak, aby pomáhal pedagogům vytvářet a spravovat online kurzy. Jeho autorem je Martin Dougiamas, který chtěl vytvořit platformu zaměřenou na sociálně konstruktivní pedagogiku a kolaborativní učení. Od té doby se Moodle stal jedním z nejpoužívanějších LMS, který má po celém světě více než 200 milionů uživatelů (**Error! Reference source not found.**).

Moodle poskytuje řadu funkcí a nástrojů na podporu online výuky, včetně možnosti vytvářet a spravovat kurzy, sledovat pokrok studentů, provádět hodnocení a komunikovat se studenty prostřednictvím fór a zpráv. Má také řadu zásuvných modulů třetích stran, které lze použít k rozšíření jeho funkcí, například zásuvné moduly pro videokonference a odhalování plagiátorství (**Error! Reference source not found.**).

Jednou z klíčových vlastností systému Moodle je jeho flexibilita a možnost přizpůsobení. Pedagogové mohou platformu přizpůsobit svým specifickým potřebám a preferencím, například výběrem různých formátů kurzů, systémů hodnocení a metod

hodnocení. Moodle také poskytuje podporu pro různé typy obsahu, včetně textu, multimédií a interaktivních prvků, a lze jej používat na celé řadě zařízení, od stolních počítačů až po chytré telefony a tablety (**Error! Reference source not found.**).

Závěrem lze říci, že Moodle je výkonný a všestranný systém LMS, který se stal základním nástrojem pro pedagogy a organizace, které chtějí poskytovat online vzdělávání. Jeho flexibilní a přizpůsobitelný design, široká škála funkcí a početná komunita uživatelů přispěly k tomu, že se stal jedním z nejpoužívanějších a nejuznávanějších systémů LMS na světě.

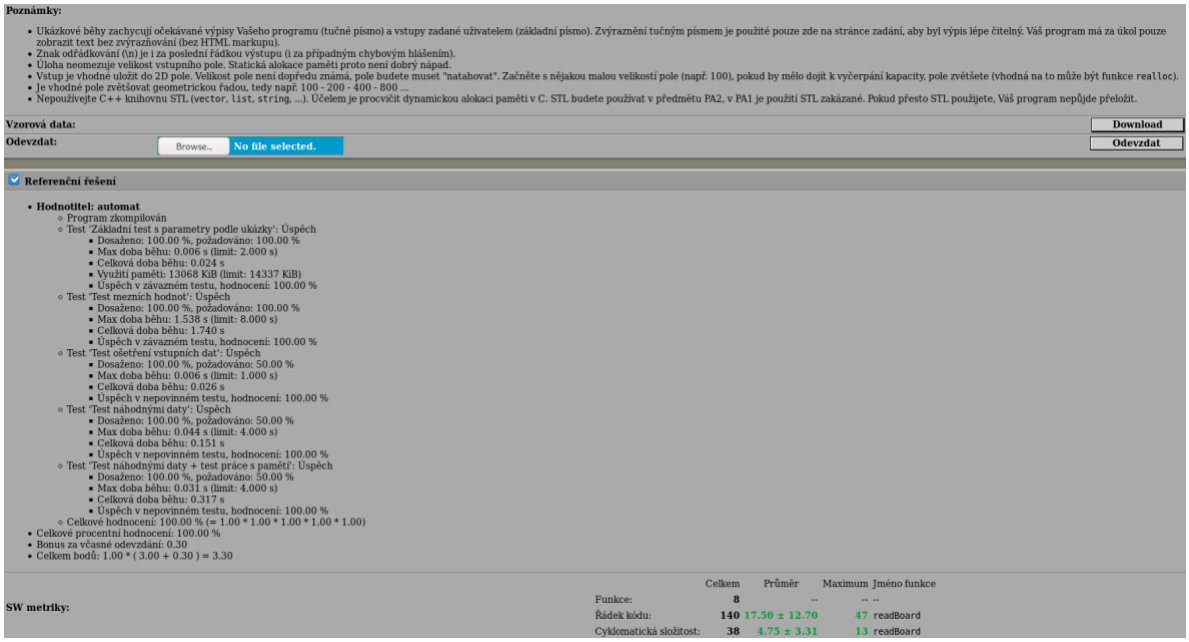
2.11 Rešerše existujících řešení

Systémy s podobným fungováním nejsou příliš rozšířeny. V rámci rešerše bylo nalezeno jedno českého řešení a tím je ProgTest. Webovou aplikaci ProgTest navrhl Ing. Ladislav Vagner, Ph.D. pro potřeby výuky programování na Českém vysokém učení technickém na fakultě informačních technologií (61).

Tato aplikace je však přístupná pouze pro studenty této školy, a tak tento automatizační nástroj nelze prozkoumat oficiální cestou. Systém je přístupný na adrese https://progtest.fit.cvut.cz. Viz. obrázek č. 3.

Obrázek 3 Přihlášování do systému ProgTest, foto pořízeno ze stránek progtest.fit.cvut.cz

ProgTest funguje na tomto principu: Učitel vytvoří slovní zadání domácí úlohy z programování, nastaví, jaká data budou testována, jaké bude maximum využité paměti, a nakonec zadá datum, dokdy má být úloha odevzdána. Student následně začne vytvářet zdrojový kód a když si je jistý funkčností, tak odevzdá kód na web ProgTestu. ProgTest si studentův program spustí, otestuje na velké škále testovacích dat a vrátí výsledné hodnocení (61). Viz. obrázek č. 4.



Obrázek 4 Vyhodnocení úspěšnosti (61)

3 Vlastní práce

3.1 Výchozí požadavky na funkčnost

Cílem této práce je, aby aplikace dokázala plnohodnotně nahradit práci učitele při kontrolování, a aby si dokázala poradit s testováním nejrůznějších typů domácích úkolů. Nejjednodušším typem je program, který pouze vypisuje čísla na výstup console (stdout) na základě zadaných vstupů z klávesnice (stdin). Při některých úlohách však mohou vycházet trochu jiná čísla podle toho, jak je program napsaný, a tak je nutné, aby systém uměl pracovat s tolerancí. Dalším typem je oblíbená úloha "stromečky", kde si žák procvičuje práci s cykly. Výstup takové úlohy vypadá například takto:

```
*
**
***
****
*****
```

nebo:

```
    *
   ***
  *****
*******
```

Aby aplikace uměla číst tyto grafické výstupy, musí umět správně číst konce řádků, které jsou ohraničené znakem \n. Neměly by být opomenuty úlohy, ve kterých se procvičuje práce se čtením a zapisováním do textových souborů. Aplikace tedy musí mít možnost nastavit si, z jakého rozhraní budou data kontrolována a do jakého zapisována, zda ze Stdout, nebo z nějakého textového souboru. Součástí výuky programování je i práce s argumenty při spouštění, a i to by mělo být možné otestovat. Jelikož jsou testovaná data vytvořená tak, aby bylo zjištěno, zda program neskončí s chybovou hláškou, může se stát, že testovaný program zůstane v nekonečné smyčce. I s tím by si měla aplikace umět poradit, jinak by došlo k tomu, že by stále čekala na ukončení, které by nikdy nenastalo. Dalším požadavkem bylo, aby si učitel mohl nahrát k testování domácí úkoly od všech studentů z jedné třídy najednou a nemusel je manuálně testovat úkol po úkolu.

Samozřejmostí by měla být možnost projít si detaily vyhodnocování, včetně procházení běhu programu. Uživatel (učitel) si tak bude moci zobrazit výstup testovaného programu a zároveň uvidí, jaký měl vypadat správný výsledek.

Aplikace by měla umět testovat domácí úkoly v různých programovacích jazycích. Jelikož z jazyka C vychází většina dnešních programovacích jazyků jako je C#, Java, Javascript, Objective C a mnoho dalších, měl by tedy mít jazyk C primární podporu. Student se při používání tohoto jazyka naučí dobře pracovat s pamětí a pochopí, jak fungují například pointery. Dále by aplikace měla podporovat jazyk C++, který je „pouhou" nadstavbou jazyka C, avšak podporuje objektové programování. Dále široce používaným jazykem je Java. Má

velkou komunitu, používá se více než C++, je multiplatformní a existuje pro něj mnoho vývojářských nástrojů (IDE) (52).

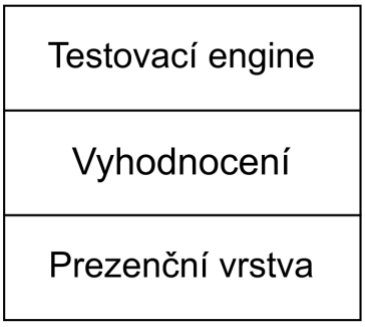
Další důležitou součástí je uživatelské rozhraní (dále jen jako UI), skrze které bude uživatel (většinou učitel) ovládat celou aplikaci. UI by mělo být jednoduché a příjemné na ovládání. Uživatel by měl být schopný spustit aplikaci jak na počítači, tak v telefonu či tabletu.

V momentě, kdy se na serveru spouští cizí počítačový program od nějakého studenta, musí být jistota, že tento program nepoškodí ostatní služby a aplikace, které již na serveru jsou. Dále také musí být ošetřeno to, že testovaný program bude mít přístup jen tam, kam mu to bude explicitně dovoleno. Musí být zamezeno tomu, aby se dal tento systém na automatické testování domácích úkolů dobře využít pro injekci škodlivého kódu, a aby tak útočník nepřevzal kontrolu nad celým serverem. Z těchto důvodů je nutné, aby testovaný program běžel ve speciálním uzavřeném prostředí, ze kterého nebude mít přístup nikam jinam a toto prostředí by také mělo být odříznuté od internetového připojení.

Aplikace by měla počítat s budoucím rozvojem a přidáváním dalších funkcionalit, jako je například integrace s Moodle portálem.

3.2 Architektura aplikace

Aplikaci bude nutné rozdělit do několika vrstev (viz. obrázek 5). První a nejspodnější bude vrstva testovacího enginu neboli jádra, která bude srdcem celého systému. Bude tam probíhat kompilace testovaných programů na základě zvoleného programovacího jazyka a testování funkčnosti programu za pomocí opakovaného spouštění nad různými vstupními daty.
Druhá vrstva se bude starat o vyhodnocení správnosti výstupních dat z testovaného programu a poslední třetí vrstva bude prezenční. V té se bude řešit zobrazování celkového výsledku z testování a budou se skrze ni i nahrávat vstupní data o testování a samotný zdrojový kód testovaného programu.

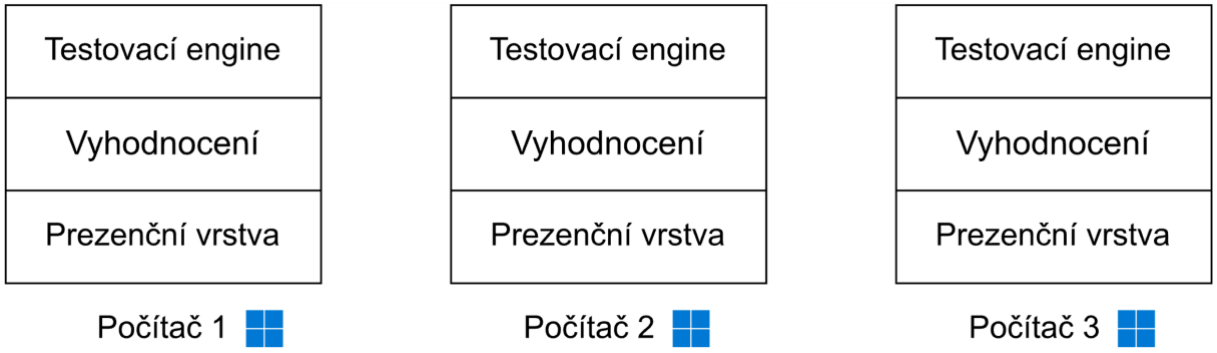


Obrázek 5 Jednoduchá struktura aplikace

3.2.1 Tlustý klient nebo tenký klient

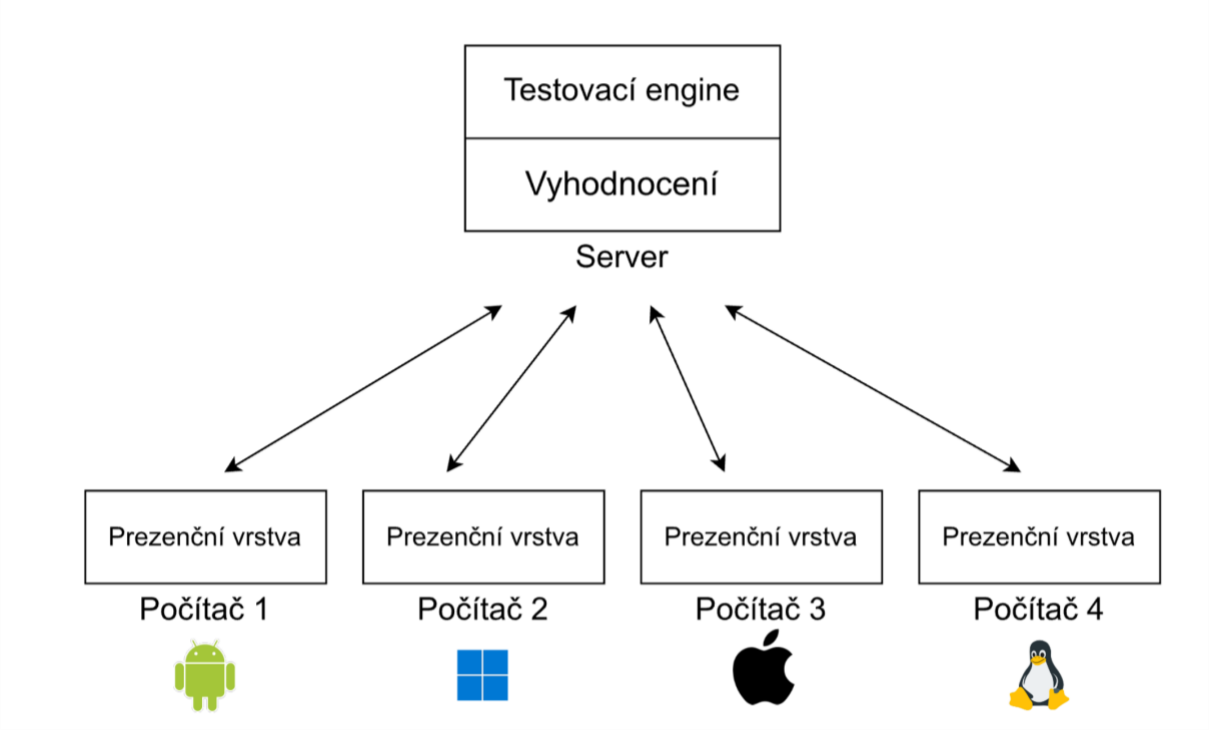
To jinými slovy znamená, jestli bude veškerá aplikační logika u klienta nebo jestli bude oddělena aplikační logika od UI. To je důležitá otázka, protože na ní záleží, jak bude aplikace navržena. Výhoda tlustého klienta je, že k němu nepotřebujeme žádné síťové připojení a odezva je vždy okamžitá. Tlustý klient je ale hardwarově náročnější než tenký a vývojář musí být více

pozorný při programování, protože se může vyskytnout více chybových stavů. Další nevýhodou je, že vývojář se musí starat o to, aby jeho aplikace byla kompatibilní s mnoha verzemi jednoho operačního systému. Ještě k tomu musí vyvíjet verzi pro jiné operační systémy. Aplikace psaná jako tlustý klient by musela nést všechny tři vrstvy systému, což by bylo zbytečně redundantní (viz níže obrázek č. 6). Nabízí se použít multiplatformní nástroj typu Qt. Ten usnadňuje vývoj aplikace pro více operačních systémů tím, že programátor píše jeden univerzální kód a tento nástroj ho přeloží pro daný OS. Nicméně pro tento případ je tento způsob příliš komplikovaný a existuje jednodušší cesta (62).



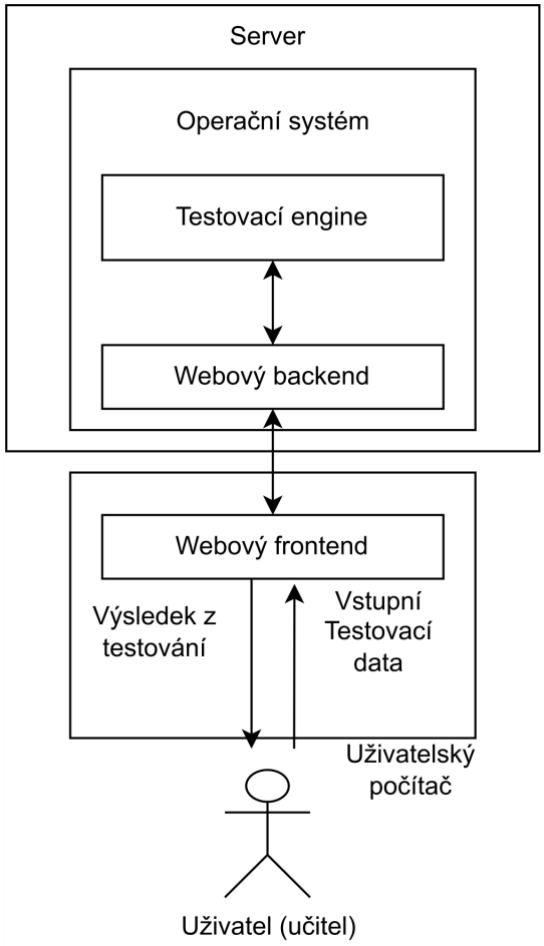
Obrázek 6 Ukázka rozvržení struktury aplikace za použití thick client neboli tlustého klienta

Tenký klient je v těchto ohledech daleko lepší. Systém vyhodnocuje všechny domácí úkoly na jednom místě, na jednom serveru a uživatel pouze vzdáleně nahrává zadání a domácí úkoly a zpět dostane výsledky o vyhodnocení. Uživatel takto může přistupovat k systému téměř odkudkoliv, kde má připojení k síti. Jelikož se nyní od klienta nevyžaduje, aby dělal všechno, včetně kompilace, testování a vyhodnocování, ale pouze nahrávání, stahování a zobrazování, můžeme použít webovou technologii, která je opravdu multiplatformní. To znamená, že se vyvíjí jeden kód bez ohledu na operační systém, na kterém by aplikace běžela. Tato technologie nám umožní přistupovat k aplikaci z operačního systému Android, iOS, MacOS, Windows a Linux (viz. obrázek 7).



Obrázek 7 Ukázka rozvržení struktury aplikace za použití thin client, neboli tenkého klienta

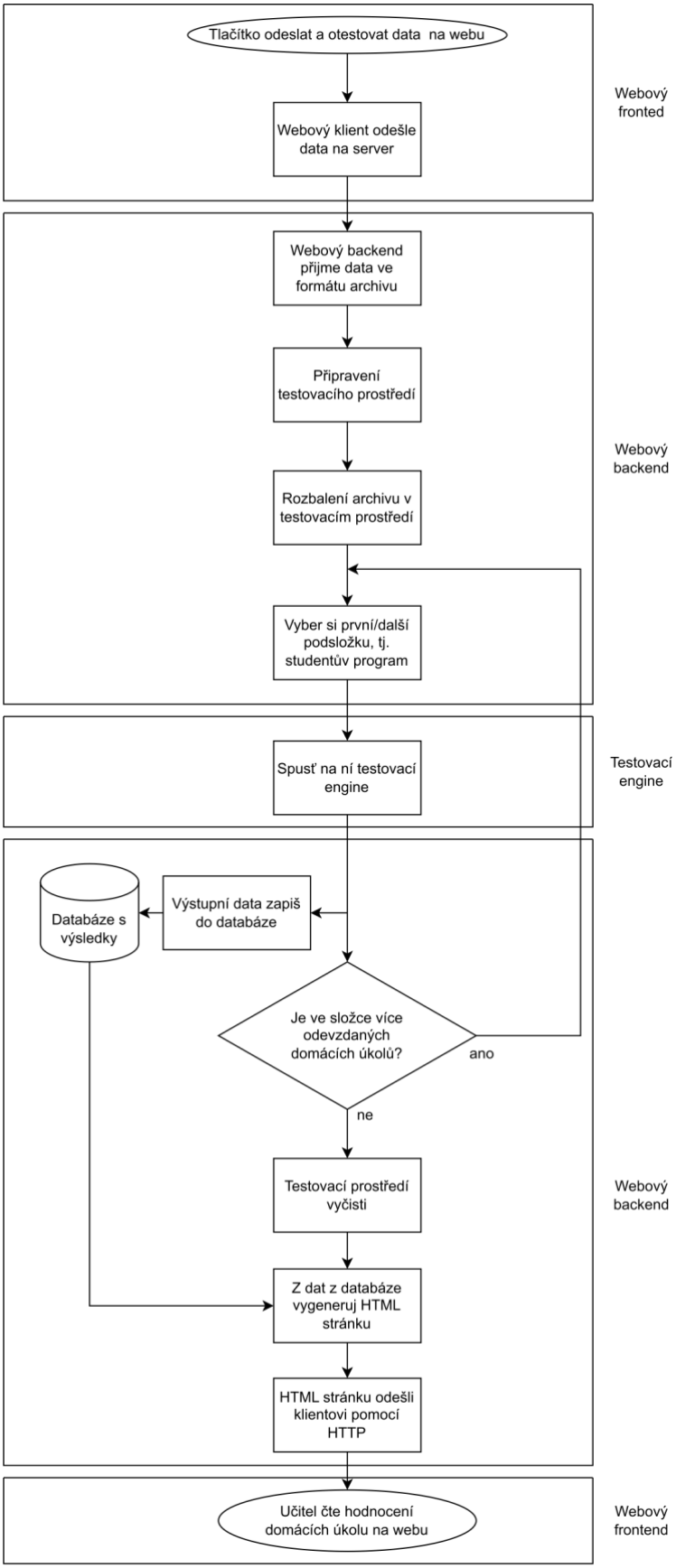
Touto problematikou se již zabýval Ing. Roman Gerhát ve své diplomové práci na téma „Vývoj internetových aplikací typu klient-server s využitím moderních přístupů“ (62). Dospěl sice k závěru, že tlustý klient je lepší než tenký, avšak pro tvorbu síťových počítačových her. Z jeho práce vyplývá, že pro vývoj webové aplikace, která nebude pracovat s 3D grafikou je lepší použít tenkého klienta, tedy webový prohlížeč. Dospěl tedy ke stejnému závěru.



Obrázek 8 První návrh aplikace

Na obrázku č. 8 je první návrh aplikace. Aplikace bude typu tenký klient a komunikace s uživatelem bude probíhat pomocí webového rozhraní. Toto rozhraní tak bude sloužit jako vrstva prezenční. Zbylé dvě vrstvy na serveru jsou vyhodnocování a testovací engine. Vyhodnocování bude probíhat na webovém backendu neboli na webovém pozadí, a to bude komunikovat s testovacím enginem.

3.2.2 Procesy uvnitř aplikace



Obrázek 9 Vývojový diagram fungování celé aplikace

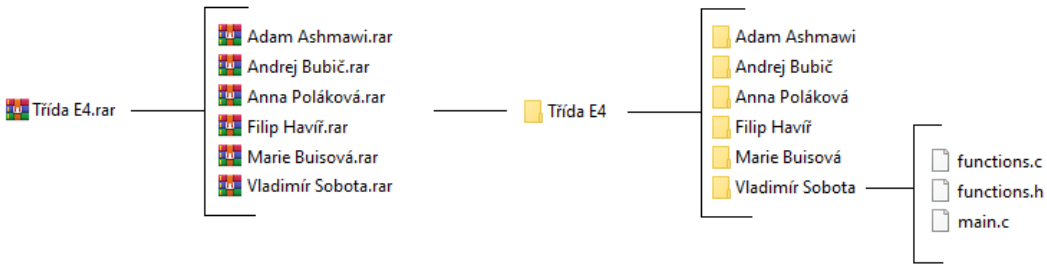
Na obrázku č. 9 je zobrazeno vnitřní fungování celé aplikace. Vpravo je popis, které ze tří vrstev se týká daná část diagramu. Lze tam také vidět, jak spolu pracují jednotlivé vrstvy a ve který moment si předávají data. Diagram je sestavený tak, aby ukázal fungování aplikace od počátečního kliknutí „Zahájit testování“ na webu až po zobrazení výsledných dat opět na webu.

V prvním kroku uživatel odešle ze svého webového klienta na server svůj soubor s konfigurací testování a archiv s odevzdaným domácím úkolem od studenta nebo od studentů. Webový backend tento archiv rozbalí do složky, která je umístěna v testovacím prostředí na serveru. Toto prostředí je bezpečně oddělené od zbytku serveru. Více informací o bezpečnosti se nachází v kapitole Analýza výběru technologie, část Testovací engine.

Ve chvíli, kdy je vše připravené pro testování se aktivuje Testovací engine. Ten začne tím, že si do paměti načte konfiguraci od učitele. V konfiguraci bude možné zapsat více testovacích iterací, což znamená, že testovaný program bude vždy spuštěn tolikrát sekvenčně za sebou, kolik bude iterací. Výsledkem bude, že studentův program bude kvalitně otestovaný na základě několika různých vstupně-výstupních dat. Engine si z konfigurace vyčte, který programovací/skriptovací jazyk se bude testovat, na tomto základě vybere správný kompilátor a následně program zkompiluje. Studentův program bude tímto krokem připravený na automatické testování.

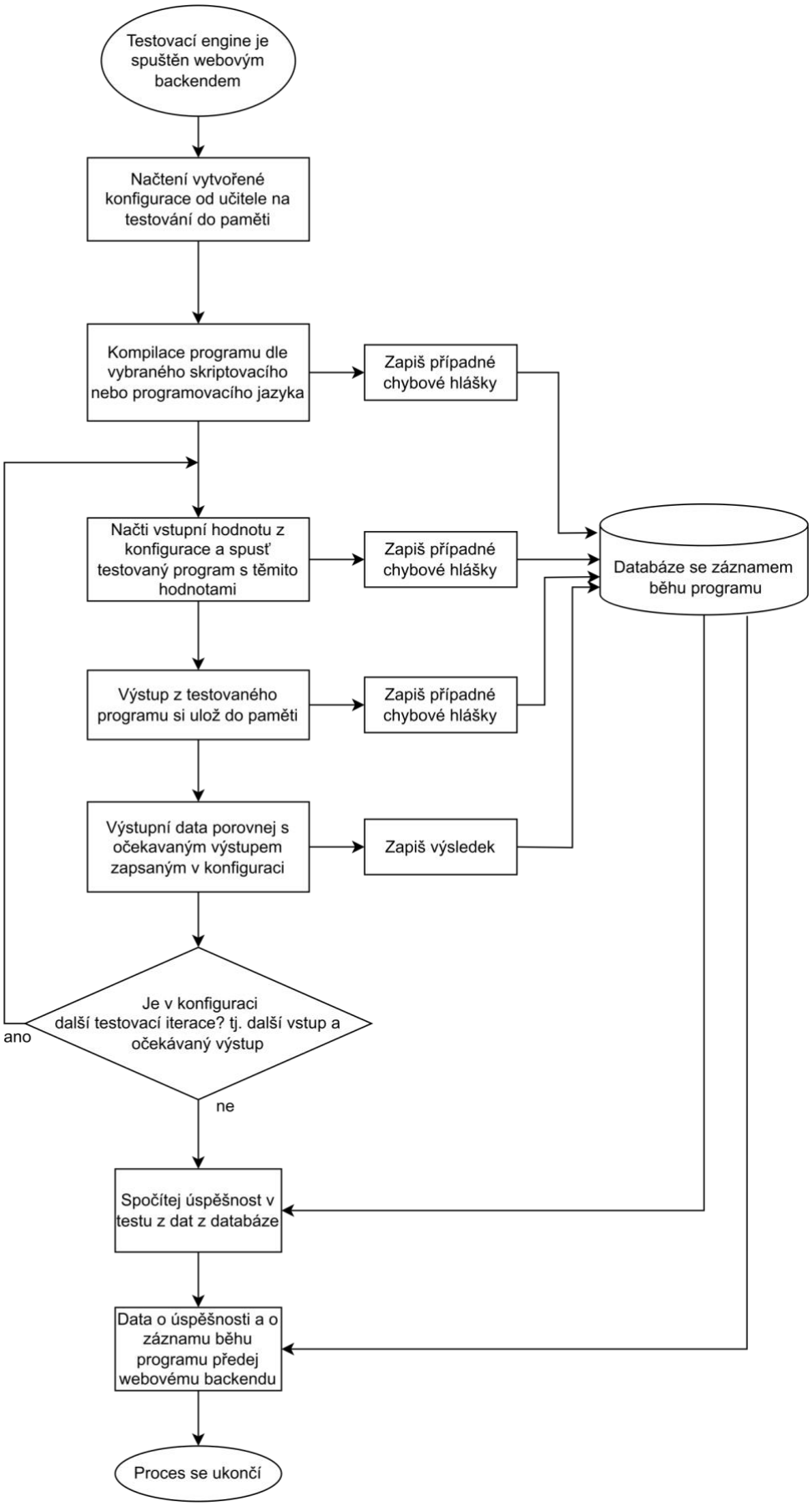
Engine bude nyní zadávat vstupní data z konfigurace na datový tok stdin, nebo je vloží jako argument ke studentovu programu nebo vytvoří textový soubor, který přiloží k testovanému programu. Záležet bude pouze na tom, jak uživatel chce, aby Engine předal tato vstupní data. Po skončení běhu testovaného programu si Engine, zaznamená data z datového toku stdout a stderr. Tato data porovná s očekávanými výstupními daty, která jsou opět zapsaná v konfiguraci.

Výsledek testu, záznam z běhu testovaného programu a chybové hlášky budou zaznamenány do databáze. Celý proces po kompilaci se bude iterovat podle počtu testovacích iterací. V případě že v rozbalené složce bude více než jeden odevzdaný domací úkol, bude Testovací Engine interován ještě navíc na úrovni kompilace. Příklad lze vidět na obrázku č. 10. Uživatel nahrál jeden archiv a v něm je 6 odevzdaných studentských prací. Testovací engine bude tedy spuštěn 6krát a pokud by v konfiguraci bylo například 5 testovacích iterací, tak při jednom běhu Testovacího enginu bude 5krát puštěn testovaný program s různými daty.



Obrázek 10 Struktura odevzdaného archivu skrze webového klienta

Samotnou vnitřní logiku Testovacího enginu je možno graficky vidět na vývojovém diagramu na obrázku č. 11. Jakmile Testovací engine dodělá veškerou svou práci, předá svou databázi webovému backendu. Ten buď vykreslí data z databáze do HTML kódu a hotový HTML kód se odešle webovému klientovi, nebo v případě Vykreslování na straně klienta se klientovi pošlou pouze data o celkovém výsledku testování a HTML kód bude vykreslen až na webovém frontendu, nebo na webovém popředí. Téma vykreslování webu bude rozebráno později v této práci.



Obrázek 11 Vývojový diagram vnitřních procesů uvnitř vrstvy Testovacícho enginu

3.3 Analýza výběru technologie

Sestrojit takto rozsáhlý systém je komplexní záležitost. Vyžaduje to rozhodnutí o tom, které technologie použijeme, a jakým způsobem je mezi sebou propojíme. Nestačí tedy napsat jednoduchý skript v Bashi nebo v Powershellu, který pouze zkompiluje kód, spustí ho se vstupními daty, která si vyčte ze souboru, a nakonec se výstup programu porovná se správným výsledkem. Později by totiž nebylo možné splnit všechny Výchozí požadavky na funkčnost, které jsou nadefinované výše. V těchto skriptovacích jazycích je horší práce s textovými řetězci a jejich úpravou, takže není možné provádět nějakou pokročilou logiku nad výstupními textovými řetězci. Co se týče UI (česky uživatelské rozhraní) tak tyto skriptovací jazyky nenabízí nic jiného než pouhé ovládání z konsole, což opět nesplňuje výchozí požadavky. Je tedy potřeba navrhnout rozsáhlejší systém, který bude složen z různých drobných skriptů, jejich úkolem bude například kompilovat testovaný program a dále z hlavního programu, který bude nad tím vykonávat vyšší logiku, tedy analýzu výsledných dat a jejich vyhodnocení, a nakonec v systému budou použity i webové technologie, díky nímž bude možné k aplikaci přistupovat vzdáleně skrze UI (viz. obrázek č. 4). Níže budou porovnány výhody a nevýhody různých počítačových technologií a na tomto základě bude vybrána výsledná technologie pro každou vrstvu.

3.3.1 Výběr značkovacího jazyku pro zadávání testovacích dat

Aplikace musí mít uložená data o tom, které hodnoty budou testovány, jak budou testovány, který programovací jazyk se bude kompilovat, jaká data mají být očekávaná na výstupu, zkrátka celá konfigurace pro daný domácí úkol. Nabízí se tři značkovací jazyky, které lze vybrat a to JSON, XML nebo CSV.

CSV je pro tento případ příliš jednoduchý, protože se do něj nezapisují jména parametrů, nelze do něj uložit pole hodnot a nelze v něm udělat složitější strukturu dat jako v JSON a XML. Jediné, co CSV umí, je ukládat hodnoty od sebe oddělené středníkem, a to je nedostačující. Proto tato možnost byla vyloučena hned na začátku (63).

Výběrem mezi JSONem a XML pro webové aplikace se zabývali již Zia Ul Haq1, Gul Faraz Khan2, Tazar Hussain ve své společné práci (64). Na straně 104 a 105 jsou přehledné tabulky výhod a nevýhod a některé z nich jsou zde uvedeny:

Výhody JSONu oproti XML:

JSON +	XML -
Kompletně naprogramovaná technika pro de-serializaci a serializaci objektů JavaScriptu s velmi malým kódováním.	Vývojář musí napsat kód, aby serializoval a de-serializoval a vytvořil XML
Většina prohlížečů má dostatečnou podporu JSON.	Všechny nové prohlížeče mají vestavěný analyzátor XML, ale může to být trochu složitější, pokud jde o analýzu XML napříč prohlížeči.
Formát je velmi stručný díky přístupu založenému na páru název/hodnota.	Kvůli značkám a jmenným prostorům je formát velmi zdlouhavý.
de-serializace je v JavaScriptu rychlá velmi	Deserializace je v JavaScriptu pomalejší

Většina knihoven JavaScriptu a sad nástrojů AJAX má dobrou podporu JSON	Sady nástrojů AJAX pro to nemají silnou podporu.

Tabulka 1 Výhody značkovacího jazyka JSON nad XML

Výhody XML oproti JSONu:

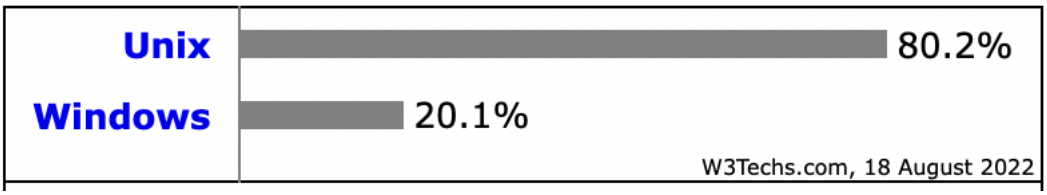
XML +	JSON -
Zatímco XML má schéma XML a definici typu dokumentu, které lze použít k definování gramatických pravidel	Neexistuje žádná podpora gramatiky
XML má více možností použití	JSON byl dělaný jen na výměnu dat web-server
Pro XML existuje XSD – jazyk pomocí kterého se kontroluje integrita výchozího XML souboru	JSON nic takového nemá, zkontrolovat integratu dat je možné až v kódu, kde si voláme JSON soubor
XML je na trhu delší dobu	JSON je mladší
Lepší pro vytváření GUI	Na toto JSON není určený

Tabulka 2 Výhody značkovacího jazyka XML nad JSON

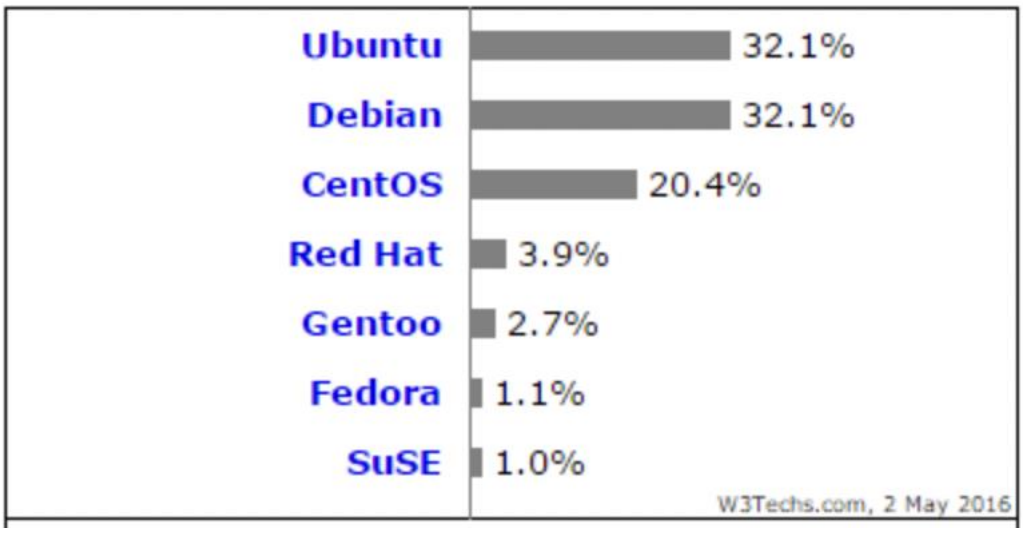
Z tabulky jasně vyplývá, že JSON byl už z definice navržen pro výměnu dat mezi webovým frontendem a serverem. Také je jednodušší na zápis dat a je méně redundantní. A tak je tedy v tomto případě vhodnější pro použití.

3.3.2 Operační systém

Ve světě na poli serverových operačních systémů jsou jen dva systémy, které dominují a těmi jsou Windows server a Linuxové distribuce. Oba tyto systémy mají svá pro a proti. Dle webu W3Techs, Unix/Linux má 80% podíl na trhu, kdežto Windows pouhých 20,1 % (65).



Graf 1 Podíl operačních systému Windows a Unix mezi webovými servery



Graf 2 Podíl konkrétních linuxových distribuci GNU/Linux ve světě webových technologií

Grafy 1 a 2 napovídají, který systém si vybralo nejvíce administrátorů, ale to nemůže být jediným důvodem, proč vybrat určitý systém. Nejprve je důležité určit si požadavky na operační systém, tedy které parametry musí splňovat, a poté se rozhodnout, který bude lepší pro systém automatického testování. Zde jsou požadavky na OS:

1. Vybraný OS musí mít podporu kompilace programovacích jazyků C, C++, Java a další, pokud se bude systém v budoucnu rozšiřovat.
2. Dále by OS měl mít podporu nějaké kontejnerizace, protože jinak by se pouštěl testovaný program přímo na daném OS, bez jakékoli fyzické zábrany, a to by mohlo být nebezpečné. V dnešní době se kontejnerizace provádí nejčastěji pomocí nástroje Docker, Kubernetes a Podman.
3. Je také potřeba aby OS umožňoval běh webového serveru.
4. Stabilní verze s pravidelnými aktualizacemi je také jedna z podmínek výběru.
5. Vybraný OS musí podporu skriptovacího jazyka, který nebude umět jen pár základních věcí, ale bude mít širokou použitelnost.
6. V neposlední řadě by měl být samotný OS bezpečný, aby odolal různým útokům a virům.
7. Jeho licence by neměla být drahá, nejlépe by měla být zadarmo.
8. Podpora v případě vyskytlých problému.

Na porovnání systému Windows a Linux již pracoval Miloslav Kameník ve své práci na straně 49-51 (66). Tabulka 3, viz níže, je syntézou mých vlastních poznatků z používání těchto systémů a poznatků z práce pana Kameníka.

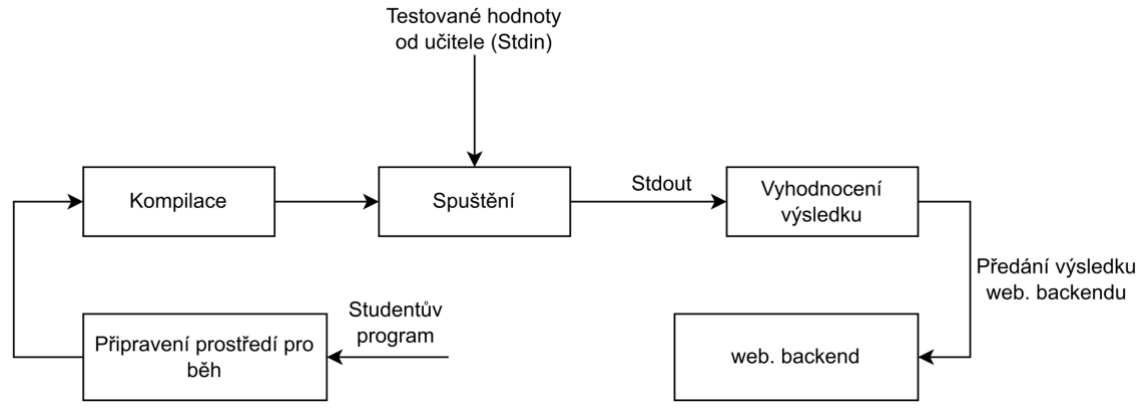
	Windows	Linux
1.	Ano	Ano
2.	Ne nativně	Ano
3.	Ano	Ano
4.	Windows je v tomto ohledu více nestálý a náchylný na chyby	Ano, Ubuntu server LTS verze mají dokonce podporu 10 let
5.	Windows disponuje PowerShellem, ovšem ten nedisponuje tolika možnostmi jako Bash	Ano
6.	Windows Servery jsou často útokem hackerů	Díky tomu že je celý systém open-source, je skvěle zabezpečený. Red Hat Enterprise Linux dokonce nabízí vyšší level ochrany pomocí technologie SELinux, která je velmi napřed před Windows
7.	Licence na Windows Servery se platí podle počtu jader a velikosti RAM, takže se cena může vyšplhat do řádů statisíců.	Linux je zadarmo, až na Red Hat distribuci
8.	Ano	Ano

Tabulka 3 porovnání windows a linux OS

Z tabulky 3, grafů 1 a 2 je nyní již jasně vidět, proč si většina vývojářů a administrátorů pro své účely zvolila některou z linuxových distribucí. Je to zkrátka proto, že Linux se více hodí pro účely webové služby. Z těchto důvodů bude vybrán Linux, konkrétně verze Ubuntu server LTS xx.04, jako OS pro webový server.

3.3.3 Testovací engine

Testovací engine bude jádrem celého systému. Jeho úkolem bude ze vstupních testovacích dat a přibaleného studentova programu získat výsledek o tom, jak si student vedl, zkrátka zda jeho program vyhověl v porovnání s výstupními daty v testovacím souboru. Je tedy potřeba vybrat vhodný programovací nebo skriptovací jazyk pro kompilaci a spuštění programu a pro vyhodnocení výsledku. Schéma, jak budou vypadat všechny části testovacího enginu je vidět na obrázku č. 11 a 12.



Obrázek 12 Předávání vstupních a výstupních dat v Testovacícm enginu

Prostředí pro běh

Prvně musí být vytvořeno prostředí, ve kterém se bude testovaný program pouštět. Tato část by měla být vytvořena bezpečnou cestou, abychom měli jistotu, že studentův program se nedostane nikam dál. Jak již bylo nastíněno v předešlé kapitole, je potřeba vybrat jakýsi "izolátor", který fyzicky oddělí toto prostředí od zbylého OS. Toho lze docílit pomocí kontejnerizace a nejznámějšími nástroji jsou Docker a Podman. Jinou možností by bylo vytvořit virtuální počítač pouze pro testování programů. A nakonec by ještě mělo být uváženo použití programu isolate.

Docker a Podman zabírají podobný prostor v ekosystému kontejnerů, nejsou ale stejné a mají různé filozofie a přístupy k tomu, jak fungují. Například Docker je all-in-one platforma s nástroji pro konkrétní úkoly, zatímco Podman pro určité účely spolupracuje s jinými nástroji. Například spoléhá na Buildah při vytváření kontejnerových images. Existují také architektonické rozdíly: Docker například nemá nativní koncept podů. Pod je architektonická filozofie. Je to set různých kontejnerů, které jsou dohromady zabalené do většího kontejneru a tím je pod. I když má pod jeden kontejner, tak je stejně v podu a teprve pod je přístupný z OS. Dalším důležitým rozdílem je, že Docker se při vytváření images a spouštění kontejnerů spoléhá na nepřetržitě běžící program démona na pozadí, zatímco Podman spouští kontejnery a pody jako samostatné podřízené procesy. Tento aspekt návrhu Dockeru má důležité důsledky pro bezpečnost a to, že útočníkovi stačí, když se dostane do démona a má přístup ke všem kontejnerům, zatímco v Podmanu má každý pod svůj proces, a tak se útočník dostane maximálně na pod a ne ke všem podům/kontejnerům. Podman je tedy lepší, co se týče bezpečnosti. (8) (9).

Použití virtuálního počítače, pro potřeby práce s testovacím programem by bylo zbytečně složité. Například pokud by ve virtuálním počítači byl nainstalován OS Ubuntu Server LTS, velikost tohoto počítače by byla příliš veliká, byly by to řády gigabajtů. Další nevýhodou by bylo složité přenášení dat mezi testovacím enginem a zbytkem webové aplikace.

Poslední vyjmenovaná možnost, program Isolate, je velmi zajímavá volba. Je to program, který vytvořili Mgr. Martin Mareš, Ph.D. a Bernard Blackham, Ph.D. Martin Mareš pracuje na Matematicko-fyzikální fakultě Univerzity Karlovy na katedře aplikované informatiky a tam vyvinul tento unikátní program, který umí spustit téměř jakýkoli linuxový program v uzavřeném sandboxu, takže spouštěný program nemůže komunikovat s vnějším světem. Celý tento Marešův program je veliký zhruba několik desítek kilobajtů a umí přesně to, co je potřeba a nic navíc (67).
Z těchto důvodů bude použit nástroj isolate. Dalším důvodem je to, že Docker a Podman jsou příliš komplexní a složité pro tento případ.

Kompilace

Kompilace kódu psaného v jazyce C se provádí standartně pomocí kompilátoru GCC. Jazyk C++ pomocí kompilátoru G++. Java je poněkud složitější na kompilaci, ale dá se použít například nástroj Apache Maven. Cílem je, aby byl celý proces kompilace prováděn automaticky a uživatel do něj nemusel zasahovat, pokud přímo nechce. Dá se totiž předpokládat, že ne každý uživatel této webové aplikace bude zběhlý v kompilaci kódu na Linuxu.

Pro kompilaci kódů se také často používá nástroj Make. Tento nástroj má výhodu, když se kompiluje rozsáhlý program, protože Make umí rozpoznat, kde se provedla změna v kódu, a

která část kódu zůstala stejná a zkompiluje jenom tu pozměněnou část, čímž razantně sníží celkový čas kompilace. Dají se v něm také nastavovat různé závislosti, takže nedojde k tomu, že se zkompiluje kus kódu, který ale ke své funkci potřebuje jinou část kódu, která však ještě nebyla zkompilována. Toto všechno se nastavuje v souboru zvaném Makefile. Nevýhodou makefile je, že se těžko automatizuje, naopak se do něj musí všechno napsat ručně, včetně přesných jmen souborů s kódem. Také se ho nevyplatí používat v případě, že testovaný program je rozdělen například jen do 2 souborů, a přitom v nejčastějších případech, bude tato webová aplikace řešit kód napsaný pouze v jednom souboru (68). Domácí úkoly zkrátka většinou nejsou veliké programy. Ve výsledku tedy bude jednodušší vytvořit krátký skript v bashi, který bude automatizovaně kompilovat testované programy pomocí kompilátorů, které jsou vypsané na začátku podkapitoly.

Vyhodnocení výsledků

Jak již bylo v této práci zmíněno, pro vyhodnocování výsledků bude potřeba vyvinout algoritmus, který bude na základě řady nastavených parametrů zpracovávat textové výstupy z testovaných programů. Dá se tedy očekávat, že v algoritmu bude pokročilá práce s textovými řetězci. Z toho důvodu nelze napsat tento algoritmus v Bashi, protože na to Bash není uzpůsobený, ale bude vybrán programovací/skriptovací jazyk, který toho bude schopen. Ve vybraném programovacím jazyku bude pak napsán i celý webový backend. Důvodem je, že jednotný programovací jazyk nám umožní lepší přenášení dat mezi vyhodnocováním a webovým backendem. Kdybychom totiž měli tuto část napsanou například v C++ a webový backend by byl napsaný v Javě, bylo by daleko obtížnější zajistit mezi těmito vrstvami výměnu dat. Data by se musela vyměňovat buď přes sdílenou paměť RAM nebo pomocí nějakého značkovacího jazyka, kam by se ukládala data nebo také pomocí síťového spojení. Všechny tyto tři varianty by byly ale zbytečně složité. Výběr programovacího jazyka pro testovací engine se tedy bude řídit výběrem programovacího jazyka pro webový backend, ale s tím ohledem, že by v něm měla být snadná práce s textovými řetězci.

3.3.4 Webový backend

Vybraný jazyk by měl splňovat kritéria níže vypsaná. Zde jsou kritéria na programovací jazyk a jeho framework, což je již připravená knihovna funkcí na použití:

1. Kompatibilita s operačním systémem GNU/Linux

Tento požadavek je základním předpokladem v dalším vývoji. Šlo by snad vyvíjet na Linuxu aplikaci v jazyce, který je kompatibilní pouze s Windows?

2. Pokročilá podpora práce s textovými řetězci

3. Jednoduché čtení a zapisování do souboru JSON

Některé jazyky nativně podporují práci s tímto formátem, zatímco u ostatních to může být komplikovanější.

4. Jednoduché volání, čtení a zapisování externích skriptů z bashového shellu

Vybraný jazyk bude často volat bash skripty, a to kvůli kompilaci a spuštění programů a tyto dvě komponenty si budou předávat velké množství dat. Kromě standartní funkce na volání shellu by vybraný jazyk měl umožnovat pokročilé nastavení jako je nastavení timeout času, vyčtení stderr a podobně.

5. Jednoduchá výměna dat s webovým frontendem

Opět důležité kritérium. Některé jazyky umožňují lehkou integraci s webovým frontendem.

6. Jednoduchost samotného psaní

Jelikož na vývoj této aplikace je omezený čas, je potřeba vybrat takový jazyk s kterým se bude moct jednoduše se naučit. Posuzováno je tedy to, jestli by trvalo dlouho se naučit dobře pracovat s tímto jazykem.

7. Možnost vývoje frontendové části stejným frameworkem jako na backendu v případě využití Server-side Rendering (viz. Teoretická část, podkapitola Webové technologie)

Rozhodování bude probíhat mezi nepoužívanějšími backendovými jazyky. Webová stránka Intellipaat.com uvádí tyto: Python, PHP, Java, C# (69).

Jiná webová stránka, www.medium.com, uvádí tyto jako nejlepší jazyky na backendový vývoj a k nim tyto frameworky: Java - Spring, Ruby - Rails, Python - Django, Python - Flask, Javascript - Express.js, PHP - Laravel, PHP - CakePHP, Scala - Play a Golang – Fiber a C# - ASP.NET (70).

Z důvodu mnoha kritérií a variant, bude potřeba sofistikovanější analýza porovnání, a pro tyto potřeby byla analýza provedena vícekriteriální analýzou variant, kdy byla vybrána kritéria vypsaná výše. Níže jsou uvedena ještě v tabulce s přidanými váhami. Viz tabulka č. 4. Jak je vidět, nejdůležitější byla kompatibilita s operačním systémem Linux. Vysokou váhu má také kritérium volání bashových skriptů. Skripty budou totiž volány často, a tak je důležité, aby se tato činnost dala dobře provádět. Naopak nejméně důležitým kritériem je kritérium č. 7. V případě, že by vybraný framework podporoval tuto funkcionalitu, je to určitě výhoda, ale i kdyby to daný framework nepodporoval, může být přesto vybrán.

Kritéria	Typ	Stupnice	Váha
1. Kompatibilita	MAX	POM	0,5
2. Podpora text. řetězců	MAX	POM	0,35
3. Čtení/zápis do JSON	MAX	POM	0,3
4. volání bash skriptů	MAX	POM	0,4
5. Výměna dat s web. front.	MAX	POM	0,25
6. Jednochost psaní	MAX	POM	0,2
7. Stejný framew. na backendu i na frontendu	MAX	POM	0,1

Tabulka 4 Kritéria a váhy pro analýzu

Varianty	1	2	3	4	5	6	7
PHP - CakePHP	9	5	7	3	8	2	0
PHP - Laravel	9	7	7	3	7	2	0
Scala - Play	8	3	6	2	6	4	0
Golang - Fiber	10	4	8	4	6	5	0
C# - asp.net	3	8	5	4	5	5	0
Java - Spring	6	7	7	6	7	5	0
Ruby - Rails	7	6	6	6	6	6	0
Python - Django	10	9	10	8	9	7	10
Python - Flask	10	9	10	8	9	10	10
JS - Express.js	8	6	10	3	9	6	0

Tabulka 5 Matice programovacích jazyků

Varianty	Skóre	Pořadí
Python - Flask	19,6	1
Python - Django	19	2
JS - Express.js	13,75	3
Golang - Fiber	12,9	4
Java - Spring	12,7	5
Ruby - Rails	12,5	6
PHP - Laravel	12,4	7
PHP - CakePHP	11,95	8
Scala - Play	9,95	9
C# - asp.net	9,65	10

Tabulka 6 Skóre a pořadí programovacích jazyků a jejich frameworků

Zdroje: (71) (72) (73) (74) (75) (76) (77) (78) (79) (80)

Z tabulky č. 5 a 6 vyplývá, že nejvhodnější jazyk pro napsání webového backendu a zároveň testovacího enginu je Python s Flask frameworkem. V závěsu za ním je framework také psaný v Pythonu a to Django. Django získalo pouze o 0,6 nižší skóre. Dalo by se tedy uvažovat o použití i Djanga, avšak z tabulky č. 5 je vidět, že Flask je jednodušší na naučení a následné psaní. Z důvodu omezeného času na vývoj byl vybrán Flask.

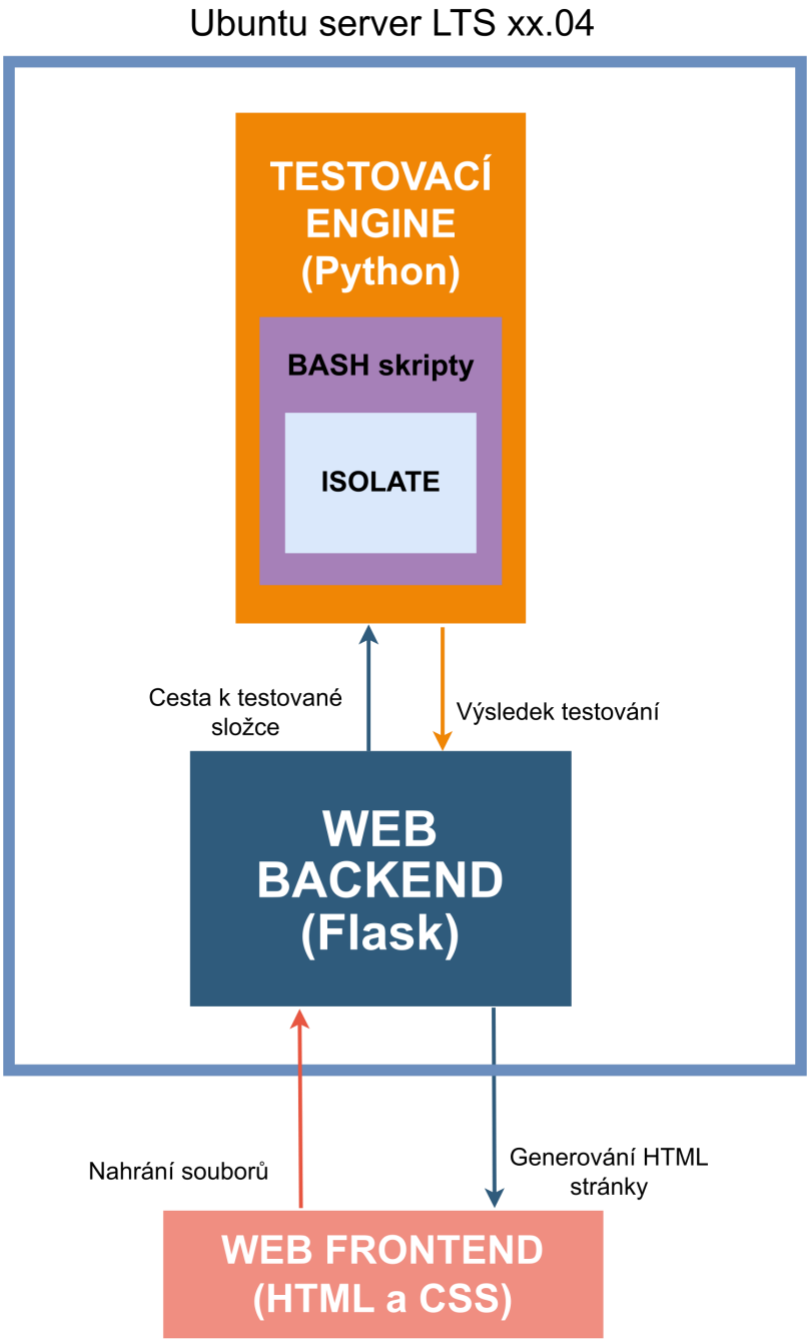
3.3.5 Webový frontend

Cílem této vrstvy je, aby uživatel, v tomto případě učitel, mohl skrze ni nahrát testovaný program nebo soubor mnoha testovaných programů a k tomu také soubor typu JSON, kde bude specifikováno, jak se bude studentův program testovat. Na výstupu by měl uživatel dostat informaci o tom, jak si student nebo studenti vedli v testu. To znamená chybové hlášky, porovnání skutečného výstupu a očekávaného výstupu a konečně také vypočtené výsledné hodnocení studenta.

Na začátku vývoje této vrstvy je důležité rozhodnout se jaký typ webového vykreslování (anglicky website rendering) bude zvolen. Na výběr je vykreslování na straně klienta, vykreslování na straně serveru anebo předběžné vykreslování. Více teoretických informací lze nalézt v teoretické části, v kapitole Webové vykreslování. V tomto případě bude použita metoda vykreslování na straně serveru. Důvodem je to, že pro účely této webové aplikace stačí pouze

jedna nebo dvě stránky, a tak by bylo zbytečné vyvíjet systém dynamického vykreslování. Vybraná metoda vykreslování je méně náročnější na vývoj, protože není potřeba psát části kódu v Javascriptu a jeho frameworcích jako jsou ReactJS, AngularJS nebo VueJS. Stačí zkrátka jeden nebo více HTML souborů a jeden CSS soubor (81).

3.3.6 Výsledek analýzy



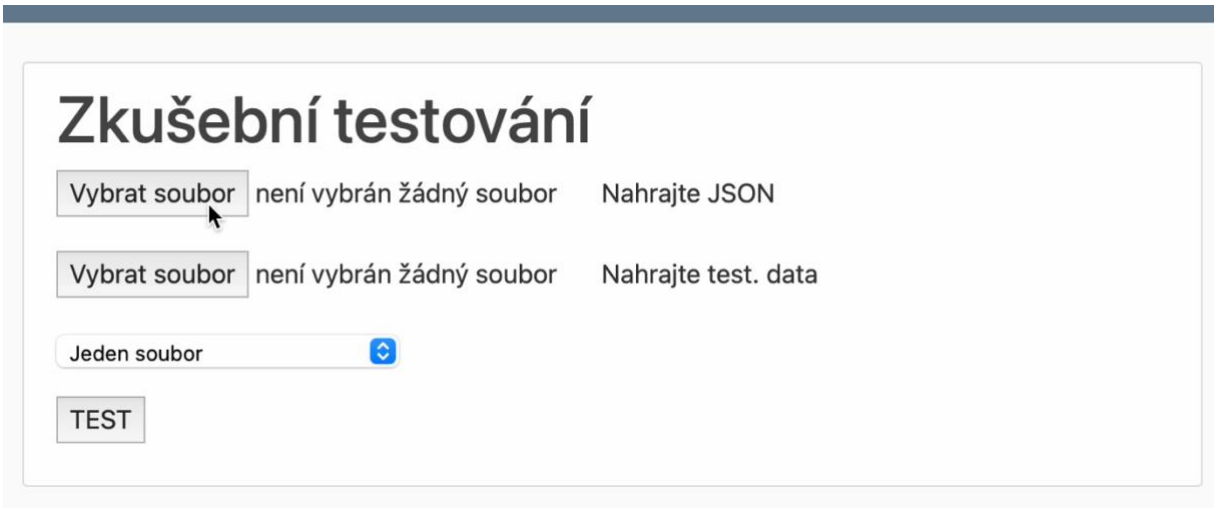
Obrázek 13 Výsledné schéma vybraných technologií

Na obrázku č. 13 je vidět výsledný přehled toho, jaké technologie na každé vrstvě budou použity. Přehled je sestavený od úrovně serverového operačního systému až po konečného uživatele. Pro každou vrstvu bylo analyzováno na základě mnoha parametrů, která technologie

by byla nejlepší. Také bylo zohledněno to, aby mezi sebou vybrané technologie uměly komunikovat, zkrátka aby byly vzájemně kompatibilní.

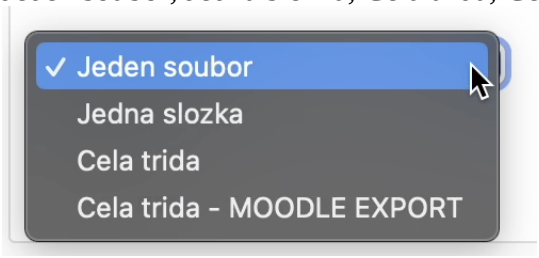
3.4 Ukázka sestrojené aplikace

Po otevření webové stránky se zobrazí tato hlavní nabídka, viz obrázek 14. Je to velmi jednoduchý webový frontend s pouze jednou HTML stránkou.



Obrázek 14 Webový klient ukázka 1

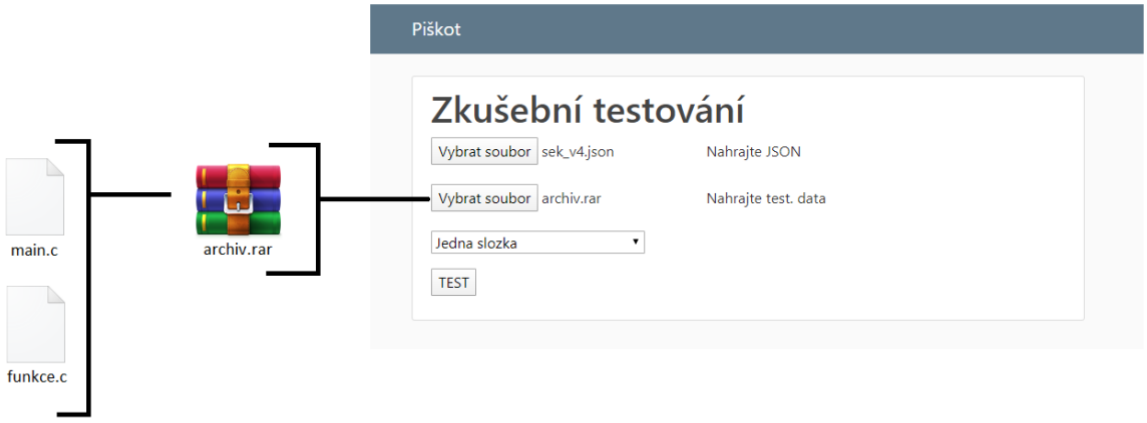
Skrze první horní tlačítko Vybrat soubor nahraje uživatel konfiguraci testování a to ve formátu JSON souboru. Kliknutím na druhé tlačítko Vybrat soubor nahrajeme testovaný zdrojový kód programu. Aplikace umožňuje více variant, jak zdrojový kód nahrát. Nápovědou je výběrové menu, které je umístěno nad tlačítkem TEST, viz. obrázek č. 15. Jsou tam varianty Jeden soubor, Jedna slozka, Cela trida, Cela trida – MOODLE EXPORT.



Obrázek 15 Webový klient ukázka 2

3.4.1 Nahrání jednoho domácího úkolu

Pokud uživatel chce nahrát jeden domácí úkol neboli jeden vytvořený počítačový program, může ho buď celý zabalit do archivu ve formátu zip nebo rar a následně ho nahrát jako jeden soubor (viz obrázek č. 16), nebo pokud je celý zdrojový kód v jednom samostatném souboru, může ho nahrát tak jak je, tedy nemusí ho archivovat, ale může odevzdat zdrojový kód v původním formátu. Druhá varianta, která je také na obrázku č. 17, se může týkat jednodušších programů psaných v C/C++, kde v jednom souboru je celý program.

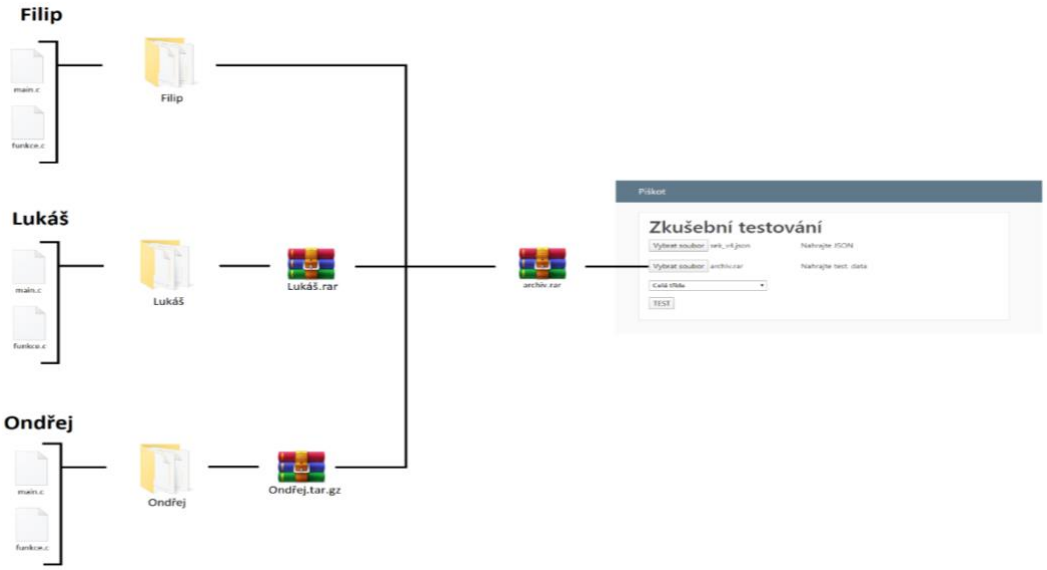


Obrázek 17 Nahrání více souborů přes archiv, vybrána metoda 'Jedna slozka'

Obrázek 16 Nahrání jednoho souboru, vybrána metoda 'Jeden soubor'

3.4.2 Nahrání úkolů od celé třídy

Pokud by uživatel chtěl na web odevzdat archiv, kde budou domácí úkoly od více studentů nebo od celé třídy, musí být uvnitř tohoto archivu dodržena adresářová struktura, viz obrázek 18.



Obrázek 18 Nahrání domácích úkolů od celé třídy, vybrána metoda 'Jeden soubor'

Student svůj úkol odevzdá buď ve formě složky (např. při odevzdávání úkolů na společný síťový disk ve školní síti, kdy lze nahrát i složku) nebo jako archiv (např. při odevzdávání přes email, kdy nelze nahrát složku, ale jen archiv). Výsledné jméno složky/archivu by mělo být studentovo jméno bez mezer s podtržítky, například Jan_Novák, protože toto jméno pak bude dále používáno v průběhu testování. Učitel pak všechny tyto odevzdané úkoly zabalí do jednoho velkého archivu a nahraje na web.

Ve výběrovém menu, ještě zbývá možnost Cela trida – MOODLE EXPORT. Tento výběr simuluje situaci, kdy studenti odevzdávají domácí úkoly skrze odevzdávací portál Moodle. Učitel si pak tyto odevzdané úkoly jedním kliknutím v Moodle portálu exportuje k sobě do počítače, ve formátu jednoho velikého archivu. Tento archiv následně nahraje na tuto webovou aplikaci.

3.4.3 Příklady použití aplikace

Zde bude předvedeno, jak lze aplikaci využít na konkrétní úloze z programování.

Název úlohy: Hloubka studny

Zadání od učitele: Napište program v jazyce C, který změří, jaká je odhadovaná hloubka studny. Jako vstup dostane program údaj o tom, jak dlouho trval pád kuličky na dno studny. Předpokládáme, že ve studni nebude žádná voda. Vypište hloubku studny na výstupu programu. Vstup bude v sekundách a výstup bude v metrech.

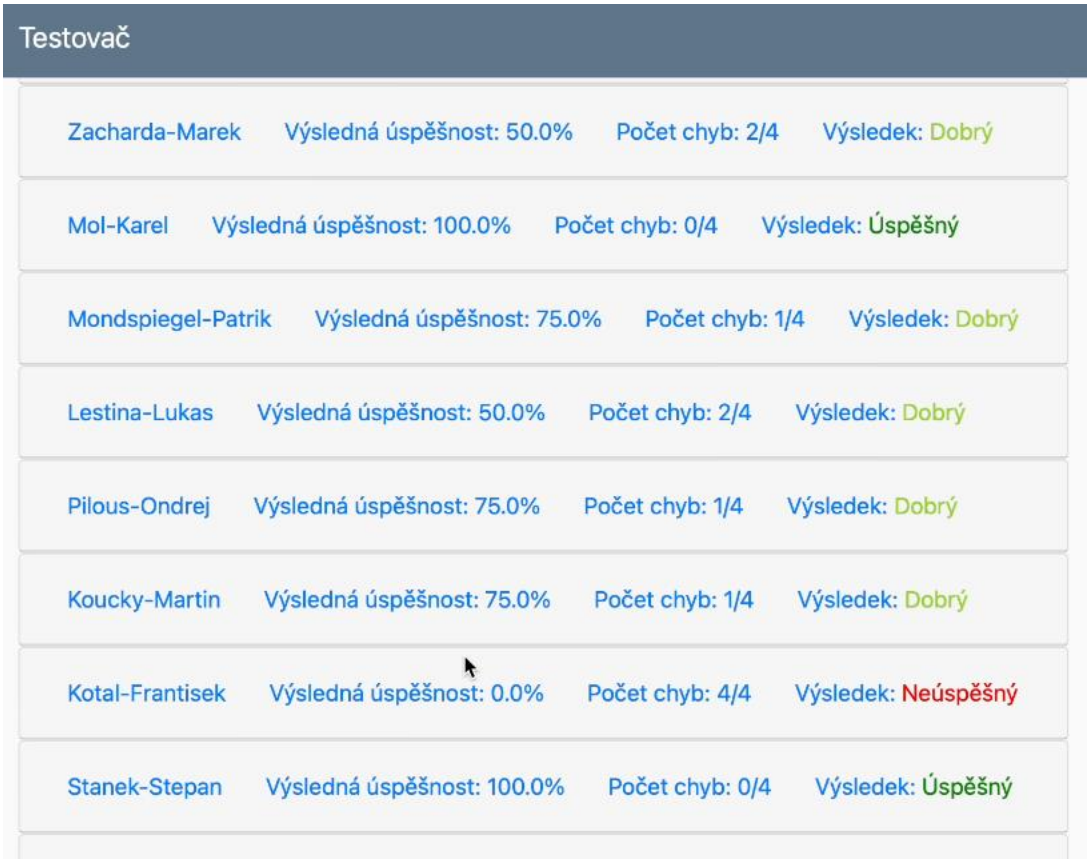
Příklady:
Vstup: 5 Výstup: 122.625
Vstup: 10 Výstup: 490.

Konfigurační soubor JSON:

```
{
      "jazyk" : "c",
      "toler" : 15,
      "data" : [
      { "input_output" : [{ "in" : [5], "out" : [122.625]}] },
      { "input_output" : [{ "in" : [10], "out" : [490.5]}] },
      { "input_output" : [{ "in" : [15], "out" : [1103.625]}] },
      { "input_output" : [{ "in" : [2.5], "out" : [30.656]}] }
      ]
}
```

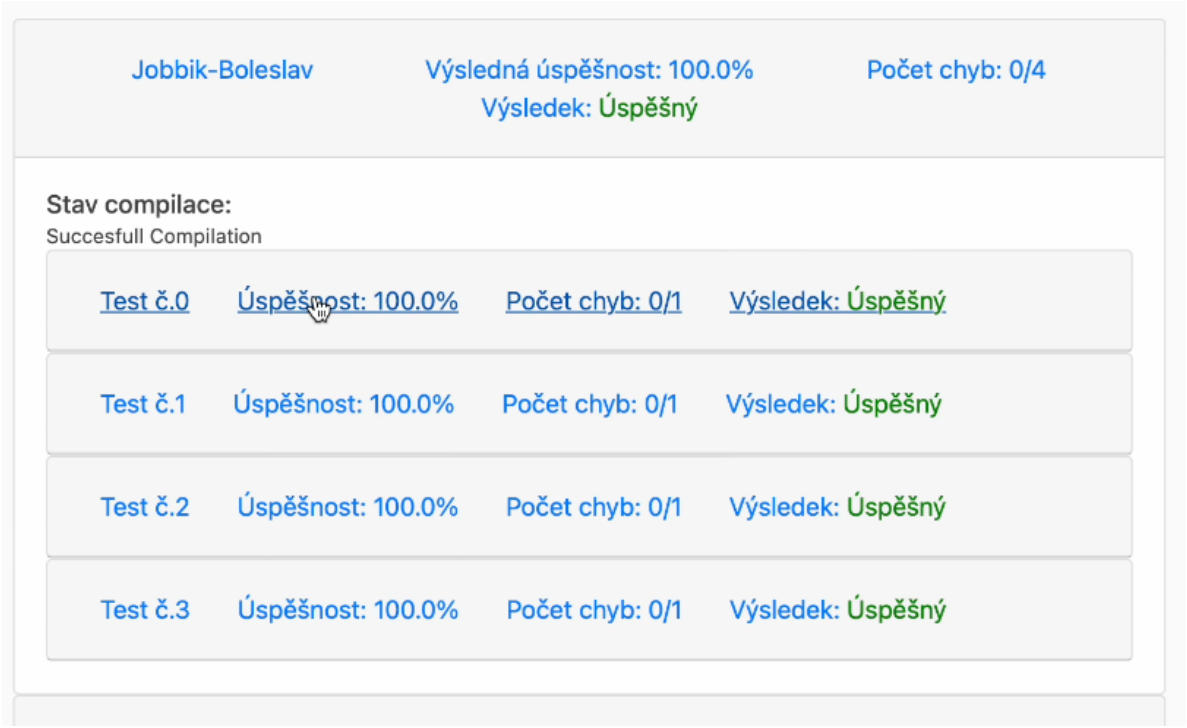
Z konfigurace je vidět, že uživatel vytvořil 4 testovací iterace (proměnná input_output v konf.) a v každé nastavil vstup a výstup (proměnné in a out v konf.). Nahoře ještě nastavil parametr jazyk, což znamená informaci o tom, jakého programovacího jazyka se tato úloha týká. Na základě tohoto parametru si totiž aplikace připraví příslušný kompilátor. Stojí za povšimnutí také parametr toler. Tím se nastavuje povolená tolerance výstupního čísla od očekávaného čísla, v tomto případě je tolerance nastavená na plus minus 15 metrů.

Učitel nahrál na aplikaci tento konfigurační soubor JSON, přiložil také archiv s odevzdanými úkoly z celé třídy a kliknutím spustil svůj vytvořený test. Po chvíli se mu na webovém klientu ukáže konečné hodnocení každého žáka a jeho úspěšnost v testu, jak je vidět



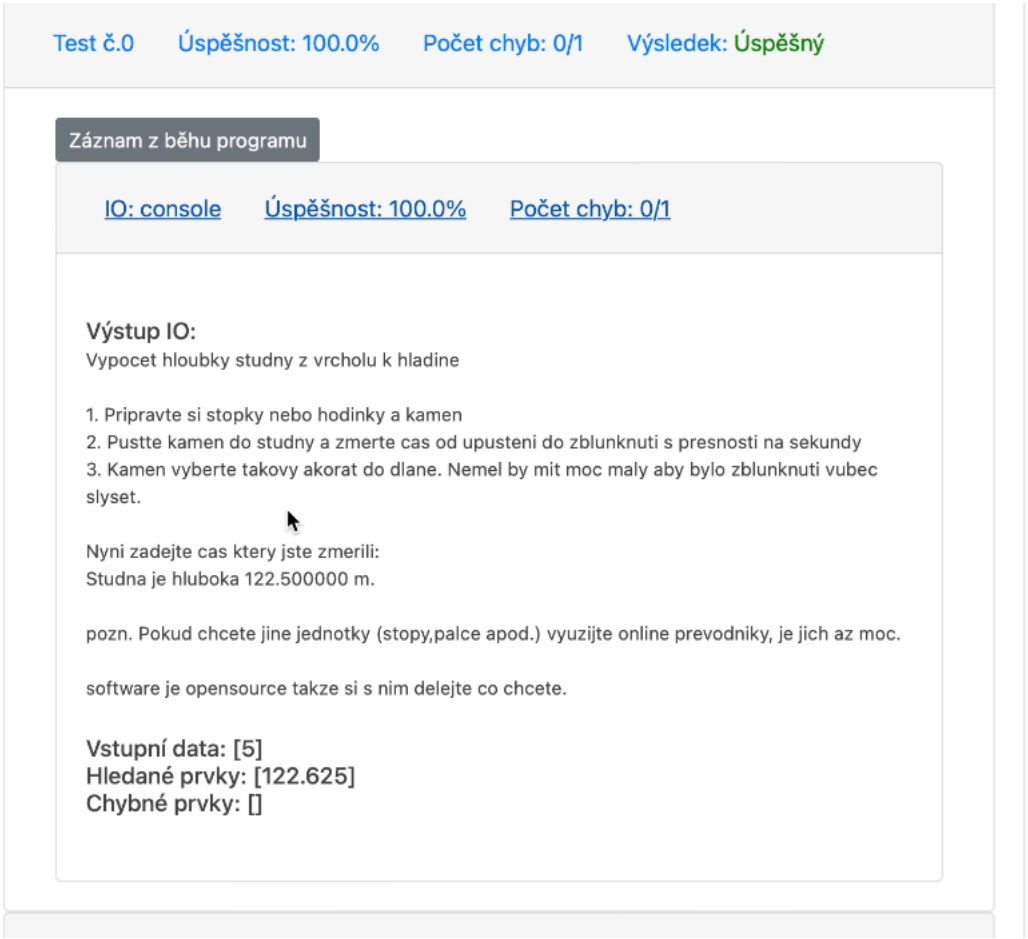
Obrázek 19 Webový klient ukázka 3

na obrázku č. 19. Učitel má okamžitý přehled o tom, jakou měl každý žák procentuální úspěšnost a který žák prospěl úspěšně, dobře nebo neúspěšně. U každého žáka se dá kliknutím na text, rozevřít tabulka, kde je ukázáno, jak si žák vedl v každé testovací iteraci a zda jí prošel. Viz. obrázek č. 20.



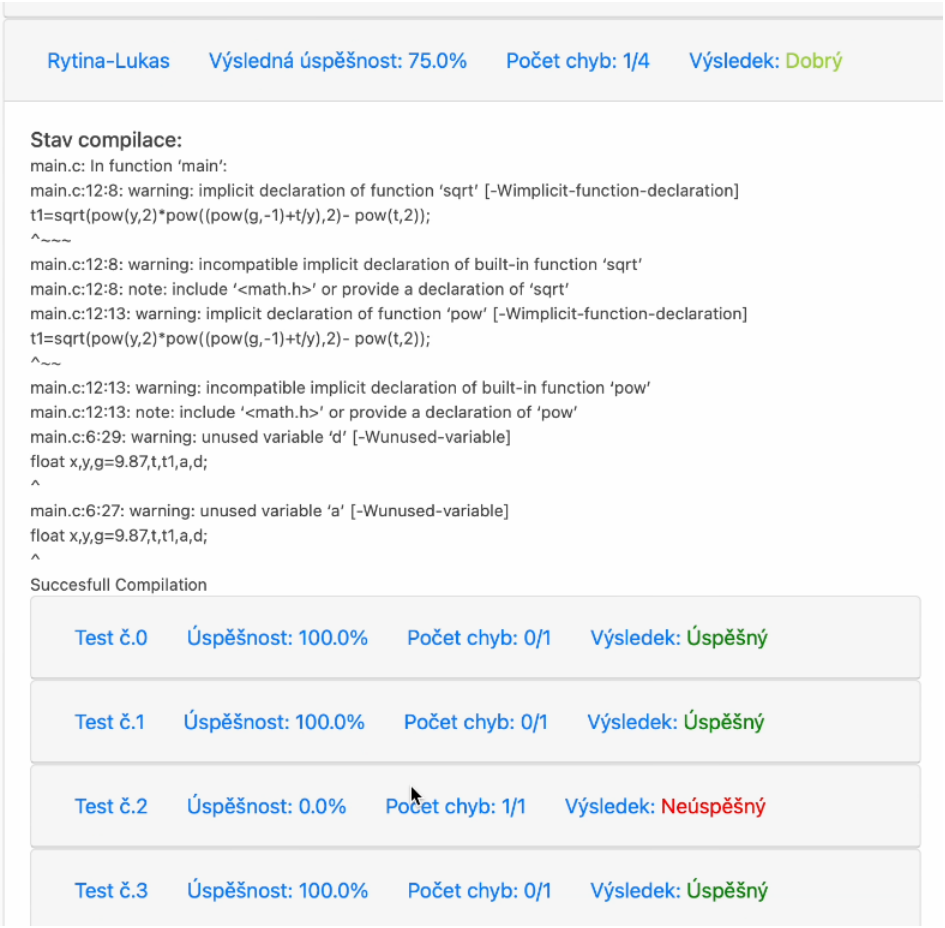
Obrázek 20 Webový klient ukázka 4

Po kliknutí na text testovací iterace lze zobrazit informace z průběhu testování. Je možné tak zjistit, jaký výstup byl vytvořen testovacím programem, myšleno stdout, a jaký výstup byl očekáván dle učitelovy konfigurace. Viz. obrázek č. 21.

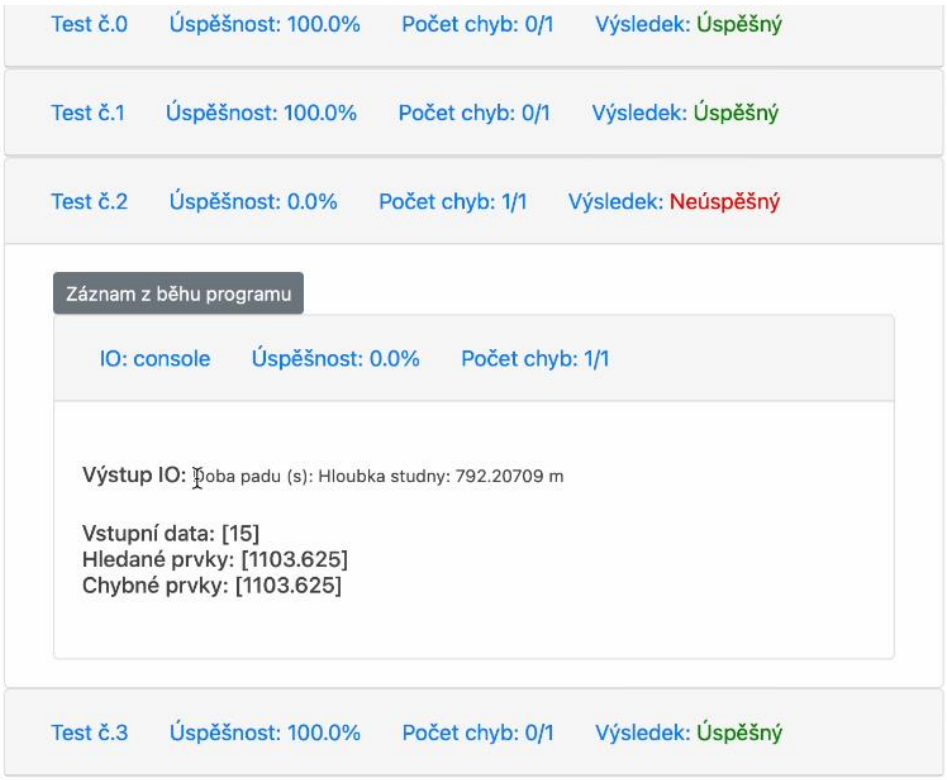


Obrázek 21 Webový klient ukázka 5

U jiného žáka však byly nalezeny chyby, viz. obrázek č. 22 a obrázek č. 23. Aplikace našla první chybu hned během pokusu o kompilaci. Žák pravděpodobně zapomněl importovat knihovny, které jsou důležité pro běh programu, a tak se ukázaly chybové hlášky, nicméně jsou to pouze chyby upozorňující, takže studentův kód byl nakonec zkompilován. Další nalezená chyba je v iteraci č. 2. Studentovi vyšlo úplně jiné číslo, než které je správně a nepomohla mu ani povolená tolerance, a tak byla tato iterace chybně vyhodnocena. Jelikož má ostatní iterace správně, dostal 75% úspěšnost.



Obrázek 23 Webový klient ukázka 6



Obrázek 22 Webový klient ukázka 7

4 Výsledky a diskuse

Jedním z výsledků této práce byla analýza výběru technologií, která byla provedena v praktické části. Pro každou vrstvu systému byly vybrány nejvhodnější technologie, s ohledem na cílenou funkci aplikace. Vrstvami je Testovací engine, Webový backend a Webový frontend. Výsledkem této analýzy je vybraný operační systém Linux, konkrétně distribuce Ubuntu Server LTS, který se ukázal jako nejvhodnější pro webové servery. Verze LTS má navíc mnohaletou podporu. Dále byl vybrán programovací jazyk Python pro vrstvu Testovacího enginu, díky snadné práci s textovými řetězci a díky jednoduchosti samotného jazyka. Pro vrstvu Webového backendu byl opět zvolen programovací jazyk Python s Flask frameworkem. Díky tomu, že obě vrstvy mají stejný jazyk, lze mezi nimi dobře přenášet data a není tak potřeba se učit více programovacích jazyků. Flask framework byl vybrán na základě vícekriteriální analýze variant, protože se v této analýze ukázal jako vyhovující. Z důvodu bezpečnosti musí být studentův odevzdaný program izolován od zbytku operačního systému na serveru. Řešením této problematiky je nástroj Isolate, vyvinutý na Matematicko-fyzikální fakultě Univerzity Karlovy panem Mgr. Martinem Marešem, Ph.D. Pro nejvrchnější vrstvu aplikace, kterou je webový klient byly zvoleny běžné technologie HTML s CSS. Jak již napovídají informace o Webovém backendu, byl zvolen typ tenký klient – webová aplikace.

Na základě syntézy této podrobné analýzy, teoretických východisek a navržené architektuře, která zahrnuje i vývojové diagramy, byla v poslední sekci praktické části ukázána sestrojená aplikace. Ukázka byla provedena na vzorové domácí úloze z programování. Aplikace je plně funkční, umožňuje testování různých druhů zadání domácích úkolů, které byly psány v různých programovacích jazycích jako jsou C, C++ a Java. Aplikace zjednodušuje práci s kontrolováním domácích úkolů tím, že není potřeba kontrolovat každý domácí úkol zvlášť, ale naopak celá třída může být najednou zkontrolována během chvíle. Do budoucna by se dalo uvažovat o rozšíření podpory programovacích jazyků například o jazyk Python, C# nebo Javascript. Dále by se dalo přemýšlet nad integrací s portálem Moodle, aby celý proces kontrolování byl ještě jednodušší.

5 Závěr

V teoretické části byla popsána východiska, na jejichž základě se dál stavělo v praktické části. Teoretická část byla sestavena tak, aby čtenáře provedla nejvíce používanými IT technologiemi v každé vybrané oblasti. Oblastmi byly operační systémy, kompilace, programovací jazyky, kontejnerizace, archivace, metody vykreslování webu a druhy webových technologií ať už na straně backendu či frontendu. Dále bylo popsáno, jak obecně probíhá vývoj aplikace či softwaru a také jak se výsledný produkt testuje, a to i v průběhu jeho vývoje. V závěru teoretické části byla také provedena rešerše existujících řešení. V této práci, z důvodu rychle se rozvíjejícího odvětví, převažují zejména zahraniční a internetové zdroje.

Praktická část se soustředí na aplikaci samotnou. Tato část začíná výchozím stanovením požadavků na funkčnost aplikace. Na základě těchto požadavků byla navržena architektura celého systému. Součástí architektury je rozdělení systému na tři vrstvy a těmi jsou Testovací engine, Webový backend a Webový frontend. Následně bylo rozhodnuto, zda aplikace bude typu tenký klient nebo tlustý klient a zda půjde vývoj cestou desktopové aplikace, nebo cestou webové aplikace. Součástí je také několik vývojových diagramů, které popisují vnitřní logiku procesů uvnitř aplikace, a to od prvního kliknutí na webu až po doručení výsledných dat zpět na web ze serveru. Dále byla udělána analýza, která IT technologie bude nejlépe vyhovovat potřebám výchozích požadavků na funkčnost. Na základě syntézy teoretických východisek, provedené analýzy a navržené architektury, byla vybrána pro každou z tří vrstev nejlépe vyhovující IT technologie. Při výběru vhodné technologie pro Webový backend byla použita vícekriteriální analýza variant, kdy z důvodu mnoha kritérií nestačilo pouhé porovnání výhod a nevýhod, ale byla potřeba sofistikovanější analýza. V závěru praktické části byla ukázaná sestrojená webová aplikace.

6 Seznam použitých zdrojů

1. Computer Hope. Microsoft Windows History. [online]. 2023 [cit. 16.02.2023]. Dostupné z: https://www.computerhope.com/history/windows.htm

2. What is Windows - javatpoint. Tutorials List - Javatpoint [online]. 2011 [cit. 20.9.2022]. Dostupné z: https://www.javatpoint.com/windows

3. Desktop Operating System Market Share Worldwide | Statcounter Global Stats. *Statcounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share* [online]. StatCounter 2023 [cit. 20.9.2022]. Dostupné z: https://gs.statcounter.com/os-market-share/desktop/worldwide

4. Introduction to Linux Operating System - GeeksforGeeks. *GeeksforGeeks | A computer science portal for geeks* [online]. 2021 [cit. 16.02.2023]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-linux-operating-system/

5. Pros and Cons of Linux Operating System. *Honest Pros Cons - Advantages & Disadvantages Of Things* [online]. 2019 [cit. 20.9.2022]. Dostupné z: https://honestproscons.com/pros-and-cons-of-linux-operating-system/

6. RedHat. What is containerization? [online]. 2021 [cit. 16.10.2023]. Dostupné z: https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization

7. Docker. *Docker Docs: How to build, share, and run applications* [online]. 2013 [cit. 20.9.2022]. Dostupné z: https://docs.docker.com/get-started/overview/

8. RedHat. What is Podman? — Podman documentation. [online]. 2019 [cit. 20.9.2022]. Dostupné z: https://docs.podman.io/en/latest/

9. Imaginary Cloud. Podman vs Docker: What are the differences? [online]. 2010 [cit. 20.9.2022]. Dostupné z: https://www.imaginarycloud.com/blog/podman-vs-docker/

10. Tech Target. What is a compiler?. *Purchase Intent Data for Enterprise Tech Sales and Marketing* [online]. 2022 [cit. 16.02.2023]. Dostupné z: https://www.techtarget.com/whatis/definition/compiler

11. Sallys.org. Překladače a vývojová prostředí [online]. 2015 [cit. 20.9.2022]. Dostupné z: https://www.sallyx.org/sally/c/linux/prekladace

12. Nanyang technological university Singapore. Compiling, Linking and Building C/C++ Applications [online]. 2018 [cit. 27.02.2023] Dostupné z: https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html

13. Die.net. GNU project C/C++ compiler - Linux man page. *Linux Documentation* [online]. 2023 [cit. 20.9.2022]. Dostupné z: https://linux.die.net/man/1/g++

14. GNU Free Software Foundation, Inc. GNU Make [online]. 2019 [cit. 19.08.2022]. Dostupné z: https://www.gnu.org/software/make/

15. Apache.org. *Maven – Welcome to Apache Maven* [online]. 2023 [cit. 19.08.2022]. Dostupné z: https://maven.apache.org

16. Web.dev. Rendering on the Web. [online]. 2022 [cit. 19.08.2022]. Dostupné z: https://web.dev/rendering-on-the-web/

17. HEAVY.AI. What is Server-Side Rendering? [online]. 2022 [cit. 20.9.2022]. Dostupné z: https://www.heavy.ai/technical-glossary/server-side-rendering

18. Next.js. What is Pre-rendering? [online]. 2021 [cit. 20.9.2022]. Dostupné z: https://nextjs.org/docs/basic-features/pages#pre-rendering

19. Flanagan, D. (2011). JavaScript: The Definitive Guide. O'Reilly Media. ISBN: 9781491952023

20. Marijn Haverbeke. Eloquent JavaScript [online]. 2018 [cit. 20.9.2022]. Dostupné z: https://eloquentjavascript.net/

21. Kosek Jiří. Historie a vývoj HTML [online]. 2014 [cit. 27.02.2023]. Dostupné z: http://htmlguru.cz/uvod-historie.html

22. Mozilla Developer Network. HTML. 2014 [cit. 27.02.2023]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

23. W3Schools. HTML Tutorial. 2023 [cit. 27.02.2023]. Dostupné z: https://www.w3schools.com/html/

24. Tutorialspoint. What is CSS? [online]. 2023 [cit. 27.02.2023]. Dostupné z: https://www.tutorialspoint.com/css/what_is_css.htm

25. Jahoda Bohumil. Základy CSS [online]. 2016 [cit. 27.02.2023]. Dostupné z: https://jecas.cz/css-zaklady

26. W3schools. JSON Introduction [online]. 2009 [cit. 20.9.2022]. Dostupné z: https://www.w3schools.in/json/intro

27. JSON.org. JSON: The JavaScript Object Notation [online]. 2023 [cit. 20.9.2022]. Dostupné z: https://www.json.org/

28. Mozilla. Working with JSON - Learn web development [online]. 2023 [cit. 20.9.2022]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

29. Cloud Computing Services - Amazon Web Services. What is XML? - Extensible Markup Language (XML) [online]. 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved. [cit. 2.10.2022]. Dostupné z: https://aws.amazon.com/what-is/xml/

30. W3C.ORG. Extensible markup language (XML) 1.0 (5th ed.) [online]. 2008 [cit. 2.10.2022]. Dostupné z: https://www.w3.org/TR/xml/

31. Mozilla. Django introduction - Learn web development [online]. 2023 [cit. 2.10.2022]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction

32. Django. Django documentation [online]. 2023 [cit. 2.10.2022]. Dostupné z: https://docs.djangoproject.com/en/4.1/

33. Tutorialspoint. What is django ORM [online]. 2022. All Rights Reserved. [cit. 2.10.2022]. Dostupné z: https://www.tutorialspoint.com/what-is-django-orm

34. StackShare - Tech Stack Intelligence. Mezzanine CMS vs Wagtail | What are the differences? [online]. 2023 [cit. 2.10.2022]. Dostupné z: https://stackshare.io/stackups/mezzanine-cms-vs-wagtail

35. Great Learning. Everything you need to know about Flask for beginners [online]. 2022 [cit. 2.10.2022]. Dostupné z: https://www.mygreatlearning.com/blog/everything-you-need-to-know-about-flask-for-beginners/

36. CareerFoundry. The Flask Web Framework: A Beginner's Guide [online]. 2022 [cit. 2.10.2022]. Dostupné z: https://careerfoundry.com/en/blog/web-development/what-is-flask/

37. Great Learning. Flask Vs Django: Which Python Framework to Choose? 2022 [cit. 2.10.2022]. Dostupné z: https://www.mygreatlearning.com/blog/flask-vs-django/

38. Tutorialspoint. Node.js - Introduction [online]. 2022 All Rights Reserved. [cit. 2.10.2022]. Dostupné z: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

39. Node.js. About Node.js. [online]. 2023 [cit. 2.10.2022]. Dostupné z: https://nodejs.org/en/about/

40. Forest Admin Blog. NestJS vs ExpressJS: Which is better for your project [online]. 2022 [cit. 2.10.2022]. Dostupné z: https://blog.forestadmin.com/nestjs-vs-expressjs-which-is-better-for-your-project/

41. PHP. What is PHP? - Manual [online]. 2001 [cit. 2.10.2022]. Dostupné z: https://www.php.net/manual/en/intro-whatis.php

42. InterviewBit. Top Features of PHP You Must Know [online]. 2022 [cit. 2.10.2022]. Dostupné z: https://www.interviewbit.com/blog/features-of-php/

43. Rascasone. Srovnání redakčních systémů: WordPress, Joomla, nebo Drupal? [online]. 2021 [cit. 2.10.2022]. Dostupné z: https://www.rascasone.com/cs/blog/srovnani-cms-wordpress-joomla-drupal

44. Cloud Computing Services - Amazon Web Services, Inc. What is .Net? - Dotnet Explained [online]. 2023 All rights reserved [cit. 2.10.2022]. Dostupné z: https://aws.amazon.com/what-is/net/

45. Tutorialspoint. ASP.NET – Introduction [online]. 2023 [cit. 2.10.2022]. Dostupné z: https://www.tutorialspoint.com/asp.net/asp.net_introduction.htm

46. StackShare - Tech Stack Intelligence. ASP.NET - Reviews, Pros & Cons | Companies using ASP.NET [online]. 2022 [cit. 2.10.2022]. Dostupné z: https://stackshare.io/asp-net

47. Australian Council for Educational Research. A guide to programming languages for coding in class [online]. 2017 [cit. 2.10.2022]. Dostupné z: https://www.teachermagazine.com/au_en/articles/a-guide-to-programming-languages-for-coding-in-class

48. National Curriculum for England.. Computing Programs of Study [online]. 2013 [cit. 2.10.2022]. Dostupné z: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/239033/PRIMARY_national_curriculum_-_Computing.pdf

49. Computer Science Teachers Association. Standards [online]. 2020 [cit. 2.11.2022]. Dostupné z: https://csteachers.org/page/standards

50. The ICPC International Collegiate Programming Contest. *About ICPC* [online]. 2019 [cit. 2.11.2022]. Dostupné z: https://icpc.global/regionals/abouticpc

51. Guzdial, M. Education: Paving the way for computational thinking. Communications of the ACM, 51(8), 25-27 [online]. 2008 [cit. 2.11.2022]. Dostupné z: https://www.researchgate.net/publication/234812396_Education_Paving_the_way_for_computational_thinking

52. Martin, R. C. Clean Code: A Handbook of Agile Software Craftsmanship. 2008 [cit. 2.11.2022]. Pearson Education. ISBN: 0132350882

53. Krasamo, Inc. The Agile Development Process for Mobile Apps [online]. 2022 [cit. 2.11.2022]. Dostupné z: https://www.krasamo.com/agile-development-process/

54. McConnell, S. Code Complete. 2004. Microsoft Press. ISBN: 0735619670

55. Github Inc. About Git - GitHub Docs [online]. 2023 [cit. 2.11.2022]. Dostupné z: https://docs.github.com/en/get-started/using-git/about-git

56. Krusche & Company GmbH. The Kanban system for agile software development explained [online]. 2022 [cit. 2.11.2022]. Dostupné z: https://kruschecompany.com/kanban-method-agile-software-development/

57. Radek Kitner. Typy testování software [online]. 2021 [cit. 2.11.2022]. Dostupné z: https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/

58. The ICE way. Why is software testing so important? [online]. 2021 [cit. 15.11.2022]. Dostupné z: https://www.theiceway.com/blog/why-is-software-testing-so-important

59. Software Testing Help. Types Of Software Testing: Different Testing Types With Details [online]. 2023 [cit. 15.11.2022]. Dostupné z: https://www.softwaretestinghelp.com/types-of-software-testing/

60. Hubken Group. What is Moodle? The ultimate guide to Moodle LMS [online]. 2023 [cit. 15.11.2022]. Dostupné z: https://www.hubkengroup.com/resources/what-is-moodle-lms-guide

61. Ladislav Vagner. První verze ProgTestu vznikla za týden – Buď FIT. Buď FIT – Časopis Fakulty informačních technologií ČVUT [online]. 2021 [cit. 15.11.2022]. Dostupné z: https://casopis.fit.cvut.cz/osobnost/ladislav-vagner-prvni-verze-progtestu-vznikla-za-tyden/

62. Gerhát Roman. Vývoj internetových aplikací typu klient-server s využitím moderních přístupů [online]. 2013 [cit. 15.11.2022]. Dostupné z: https://dspace5.zcu.cz/handle/11025/7419

63. LifeSavvy Media. What Is a CSV File, and How Do I Open It? [online]. 2022 [cit. 10.11.2022]. Dostupné z: https://www.howtogeek.com/348960/what-is-a-csv-file-and-how-do-i-open-it/

64. Haq, Zia Ul, Gul Faraz Khan, and Tazar Hussain. A comprehensive analysis of XML and JSON web technologies. New Developments in Circuits, Systems, Signal Processing, Communications and Computers: 102-109 [online]. 2013 [cit. 10.11.2022]. Dostupné z http://inase.org/library/2015/vienna/bypaper/CSSCC/CSSCC-14.pdf

65. W3Techs. Usage Statistics and Market Share of Operating Systems for Websites, August 2022. - extensive and reliable web technology surveys [online]. 2023 [cit. 18.08.2022]. Dostupné z: https://w3techs.com/technologies/overview/operating_system

66. Kameník Miloslav. Porovnání základních principů a využitelnosti operačních systémů Windows a Linux [online]. 2021 [cit. 18.08.2022]. Dostupné z: https://theses.cz/id/l5lnm1/STAG95633.pdf

67. Mareš Martin a Bernard Blackham, United Computer Wizards. ISOLATE [online]. 2023 [cit. 18.08.2022]. Dostupné z: https://www.ucw.cz/moe/isolate.1.html#_license

68. Opensource.com. What is a Makefile and how does it work? [online]. 2018 [cit. 10.11.2022]. Dostupné z: https://opensource.com/article/18/8/what-how-makefile

69. IntelliPaat. 8 Best Web Development Languages to learn [online]. 2023 [cit. 10.11.2022]. Dostupné z: https://intellipaat.com/blog/best-web-development-languages/

70. Medium. 10 Best Backend Frameworks for Web Development in 2023 [online]. 2023 [cit. 10.11.2022]. Dostupné z: https://medium.com/javarevisited/10-best-backend-frameworks-for-web-development-8d19e337f774

71. Cake Software Foundation Inc. CakePHP Docs [online]. 2023 [cit. 10.11.2022]. Dostupné z: https://book.cakephp.org/4/en/index.html

72. Laravel LLC. Laravel Documentation [online]. 2023 [cit. 10.11.2022]. Dostupné z: https://laravel.com/docs/10.x

73. Lightbend. Play Framework - Build Modern & Scalable Web Apps with Java and Scala [online]. 2021 [cit. 10.11.2022]. Dostupné z: https://www.playframework.com/documentation/2.8.x/Home

74. Fiber. Fiber documentation [online]. 2023 [cit. 10.11.2022]. Dostupné z: https://docs.gofiber.io

75. Microsoft Inc. ASP.NET documentation [online]. 2023 [cit. 10.11.2022]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0

76. Rod Johnson , Juergen Hoeller , Keith Donald. Spring Framework Reference Documentation. Developer's Documentation Collections [online]. 2015 [cit. 10.11.2022]. Dostupné z: http://www.devdoc.net/javaweb/spring/Spring-4.2.3/spring-framework-reference/htmlsingle/#mvc-ann-restcontroller

77. Rails Guides. Getting Started with Rails [online]. 2023 [cit. 10.11.2022]. Dostupné z: https://guides.rubyonrails.org/getting_started.html

78. Django Software Foundation. Django documentation [online]. 2023 [cit. 10.11.2022]. Dostupné z: https://docs.djangoproject.com/en/4.1/

79. The Pallets Projects. Flask Documentation (2.2.x) [online]. 2023 [cit. 10.11.2022]. Dostupné z: https://flask.palletsprojects.com/en/2.2.x/

80. OpenJS Foundation. Express Documentation [online]. 2017 [cit. 10.11.2022]. Dostupné z: https://expressjs.com

81. TutorialsPoint. What is client-side rendering in React [online]. 2023 [cit. 10.11.2022]. Dostupné z: https://www.tutorialspoint.com/what-is-client-side-rendering-in-react

7 Seznam obrázků, tabulek, grafů a zkratek

7.1 Seznam obrázků

7.2 Seznam tabulek

7.3 Seznam grafů

7.4 Seznam použitých zkratek

IT	Informační technologie
HTTP	Hypertext Transfer Protocol
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
LTS	Long Time Support
STDOUT	Standard Output
STDIN	Standard Input
STDERR	Standard Error
JSON	JavaScript Object Notation
GNU	GNU's Not Unix!
UI	User Interface
GUI	Graphical User Interface
XML	Extensible Markup Language
API	Application Programming Interface
CMS	Content management system
ORM	Object–relational mapping
CSRF	Cross-site request forgery
SEO	Search Engine Optimization

Přílohy

Příloha A CD se zdrojovým kódem