STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Ucmekod.eu
Interaktivní výuka programování

Luboš Zápotočný

Pardubických kraj Pardubice 2019

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Ucmekod.eu
Interaktivní výuka programování

Ucmekod.eu
Interactive teaching of programming

Autoři: Luboš Zápotočný
Škola: DELTA - Střední škola informatiky a ekonomie, s.r.o.
Ke Kamenci 151, 530 03 Pardubice
Kraj: Pardubický
Konzultant: RNDr. Jan Koupil, Ph.D.

Pardubice 2019

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Pardubicích dne 15.3.2019 ……

Luboš Zápotočný

Poděkování

Tímto bych rád poděkoval RNDr. Janu Koupilovi, Ph.D. za odborné vedení při tvorbě maturitního projektu a práce SOČ.

Anotace

Práce SOČ shrnuje vývoj webového systému, který umožňuje žákům cvičit své programátorské dovednosti pomocí programovacích cvičení s automatizovanou zpětnou vazbou. Žáci mohou tato cvičení řešit ve svém webovém prohlížeči bez nutnosti instalace dodatečného softwaru. Systém je připraven na efektivní distribuci mezi ostatní školy – instituce.

Klíčová slova

výuka programování; webová aplikace; Docker; multitenance

Annotation

The thesis summarizes the development of a web-based system that allows pupils to train their programming skills through programming exercises with automated feedback. Pupils can solve these exercises in their web browser without installing additional software. The system is ready for effective distribution among other schools - institutions.

Keywords

teaching of programming; web application; Docker; multitenancy

OBSAH

Úvod

V dnešní době jsou počítačové technologie na vzestupu, mnoho odvětví se digitalizuje a počítače, či obecně programově řízené stroje, již dokáží převzít lidskou práci, přičemž pracují rychleji a efektivněji. Moderní technologie jsou nyní daleko dostupnější než před několika lety.

I samotné programování se zpřístupnilo pro širší veřejnost. Vznikly programovací jazyky, které odbouraly nutnost psaní kódu „pomocí nul a jedniček“. Moderní editory určené pro psaní kódu dokáží za programátora odhalit chybu a zvýraznit ji, napovídat další „text“ či navrhovat určité úpravy.

Spolu s rozšiřováním digitálních technologií přichází přirozeně i potřeba vzdělávání v tom, jak takové systémy nejen používat, ale i navrhovat. Základní školy mají možnost do výuky zařadit programování již dnes, bohužel tuto možnost řada škol zatím nevyužila a místo toho se žáci učí maximálně používat Excel, což je absolutní minimum. [1]

Přenesení výuky programování do škol, obzvlášť v případě, že v určitém věku, resp. úrovni schopností, jsou opouštěna grafická prostředí a nahrazena psaním počítačového kódu, přináší na školu specifické požadavky. Je vhodné, či spíše nutné, nabídnout žákům dostatek učebního materiálu, který jim umožní programátorsky růst postupně, krok za krokem. Podobně jako v matematice či fyzice mají žáci k dispozici kromě učitelského výkladu také sbírku řešených úloh a především – pro vlastní přípravu – běžnou sbírku úloh. Podobné řešení by bylo vhodné přinést i do výuky programování. Sbírku řešených úloh mohou dobře zastoupit ukázky kódu na webových zdrojích typu itnetwork.cz, nebo komplexně pojatá učebnice jakou je například (C# – Pavel Bory), chybí ale bohatší sbírka úloh, tím spíše taková, kde by se žák po vyřešení mohl podívat na výsledek (*„vyšlo mi to“*), ale přitom se nemohl podívat na celý způsob řešení.

1 Požadavky na systém

Cílem tohoto projektu bylo vytvořit systém, který by poskytoval možnosti popsané v poslední sekci předchozí kapitoly. Jaký by tedy měl být a co by měl dokázat:

- Systém by měl nabízet uživatelům – žákům sady programovacích cvičení, které budou obsahovat zadání, prostředí pro psaní a ladění kódu a také sadu automatických testů, které uživateli dokáží poskytnout zpětnou vazbu, zdali jeho řešení vyhovuje zadání.
- Jednotlivá cvičení by měla být řazena do logických celků, jakými jsou kurzy a sekce.
- Uživatelům – učitelům by měl systém poskytnout možnost tvořit vlastní kurzy, tvořit a vkládat do nich programovací cvičení a přehledně zobrazovat odeslaná žákovská

řešení. Také by mělo být možné označit některá cvičení jako skrytá, povinná (test) nebo vložit obecné přílohy (např. prezentaci s přednáškou).

- V systému by mělo být snadné spravovat uživatelské účty a kurzy a přiřazovat je k sobě navzájem.
- Celý systém by měl být snadno dostupný a použitelný bez vysokých hardwarových nároků.
- Distribuci aplikace by měla zajišťovat softwarová architektura zvaná *multitenance*.
- Systém by měl podporovat několik různých programovacích jazyků a mělo by být možné přidávat do něj další podle potřeb aktuálního zákazníka – školy nebo jiné instituce. Stejně tak by měl být efektivně rozšiřitelný o další typy obsahu apod.
- Bylo by vhodné, aby systém obsahoval nebo byl v nějaké fázi vývoje schopen snadno obsáhnout nějaké prvky tzv. gamifikace, vedoucí k vyšší vnitřní motivaci žáků.

2 POUŽÍVANÉ POJMY

API

API (Application Programming Interface) označuje v informatice rozhraní pro programování aplikací. Jde o sbírku procedur, funkcí, tříd či protokolů nějaké knihovny, které může programátor využívat. API určuje, jakým způsobem jsou funkce knihovny volány ze zdrojového kódu programu. [2]

REST API

REST (Representational State Transfer) je architektura rozhraní pomocí které lze jednoduše vytvořit, číst, editovat nebo smazat informace ze serveru pomocí HTTP požadavků. [3]

Standardní proudy

Standardní proudy jsou vstupní a výstupní komunikační kanály mezi počítačovým programem a prostředím, ve kterém je spuštěn. Existují tři typy těchto kanálů:

- stdin (standard input)
- stdout (standard output)
- stderr (standard error)

Standardní proud *stdin* je vstupní proud textových dat. Proud *stdout* je výstupní proud dat z programu, který tato data vypisuje. Proud *stderr* je dalším typem výstupního proudu, který typicky obsahuje chybová hlášení. [4]

SDK

Software development kit (SDK) je typická sada vývojových nástrojů umožňující vytváření aplikací pro určité softwarové balíčky, frameworky, počítačové systémy, herní konzole, operační systémy nebo podobné platformy. [5]

Tenant

Tenant je skupina uživatelů, kteří sdílejí společný přístup a specifická oprávnění k instanci dané aplikace. [6]

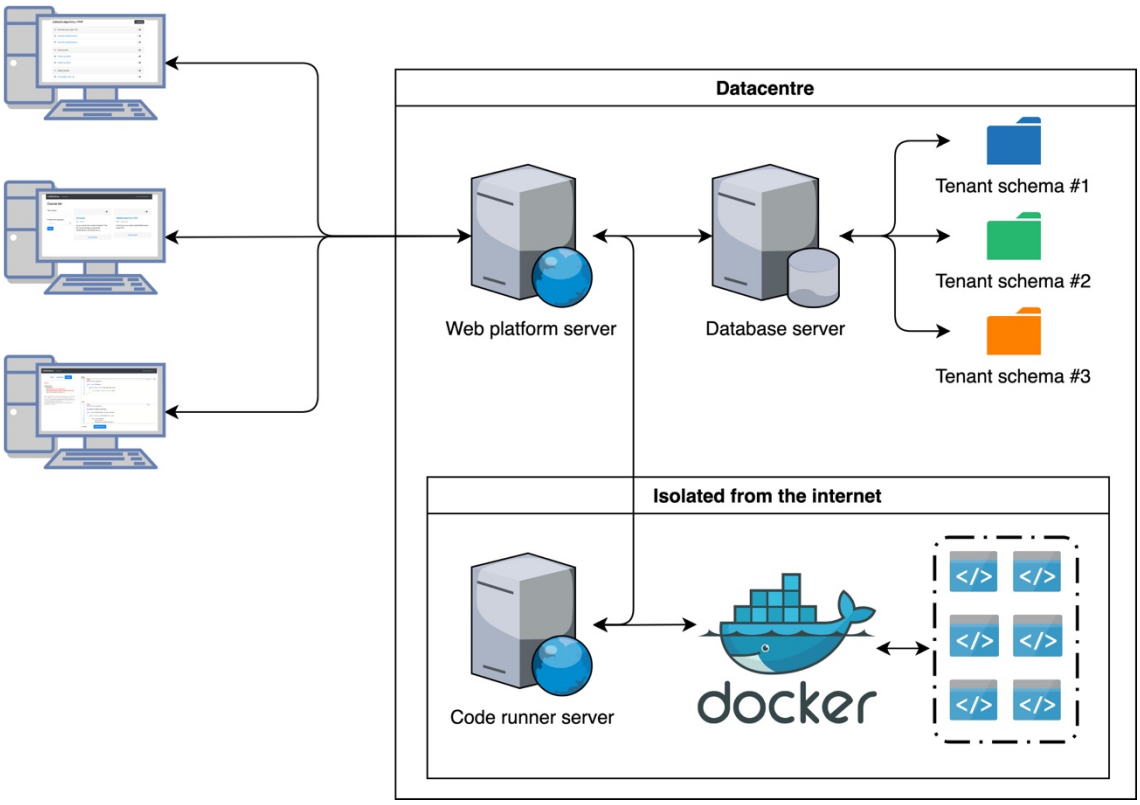
3 Architektura

Z požadavků na systém vyplývá, že by aplikace měla být rozdělena do dvou oddělených projektů, které by spolu komunikovaly pomocí REST API. Tyto části byly pojmenovány „Code runner“ a „webová platforma“.

Prvním funkčním požadavkem bylo vytvoření programu Code runner. Jeho účelem je bezpečné spouštění automatických testů, které vyhodnocují správnost předaného zdrojového kódu.

Druhou fází bylo vytvoření webové platformy, která by uživatelům poskytovala možnost procvičovat programátorské dovednosti pomocí programovacích cvičení. Tato platforma by měla být snadno dostupná přes webový prohlížeč. Distribuci aplikace by měla zajišťovat softwarová architektura zvaná *multitenance*. Podrobnější popis této metody je popsán v sekci „multitenance“ (5.1.4). Uživatel s dostatečným oprávněním by měl mít možnost zobrazit každý uživatelský pokus o vyřešení cvičení a poté případně k tomuto řešení poskytnout zpětnou vazbu. Aplikace by měla být schopna předcházet většině známých bezpečnostních chyb.

Propojení těchto dvou programů vytváří interaktivní prostředí, ve kterém lze programovat bez nutnosti instalace dodatečného softwaru. Komunikaci mezi jednotlivými částmi systému a klientem zobrazuje obrázek 1.



Obrázek 1 - Diagram architektury celého systému

4 CODE RUNNER

Code runner je nejdůležitejší součástí celé aplikace. Hlavním úkolem tohoto programu je bezpečně spouštět sady automatických testů, které indikují správnost zdrojového kódu. Tento program je z bezpečnostního hlediska spuštěn na odděleném serveru.

4.1 Použité technologie

4.1.1 Docker

Docker je počítačový program, který umožňuje spouštět tzv. **kontejnery**. Tyto *kontejnery* jsou odlehčenou verzí *virtuálních strojů*. Rozdíl je v tom, že kontejnery jsou spouštěny na společném *kernelu* operačního systému. Toto je činí velmi výkonově nenáročnými a lze jich spustit velké množství i na relativně málo výkonném serveru.

Kontejner je vytvořen na základě tzv. **image**. Jedná se o předpřipravený obraz operačního systému, který definuje obsah daného kontejneru. Tyto *image* vznikají modifikací nebo kombinací existujících *image*.

Dockerfile je soubor s instrukcemi pro Docker, aby věděl, jak má výslednou *image* vytvořit. Lze jej přirovnat k instalačním skriptu. Definuje sadu kroků, které určují výsledný obsah *image*. [7]

Autor využil „Docker SDK for Python“ aby umožnil efektivní komunikaci s *Docker API* pomocí příkazů napsaných v programovacím jazyce Python. [8]

4.1.2 Django

Django je open source webový aplikační framework napsaný v programovacím jazyce Python. Volně implementuje softwarovou architekturu MVC. Původně bylo navrženo pro správu několika zpravodajských webových aplikací. Pod veřejnou licencí BSD bylo vydáno v červnu 2005. [9]

4.2 Funkcionalita

Code runner je webová služba, která pomocí rozhraní REST API komunikuje s ostatními aplikacemi. Není vyvíjena specificky pro potřeby tohoto systému, což umožňuje znovupoužitelnost v jiných aplikacích.

Jeden ze zaregistrovaných *endpointů* komunikačního API je „/executor/execute-tests“. HTTP požadavek na tento *endpoint* spustí proces vyhodnocení sady automatických testů. Tento *endpoint* obsahuje tři povinné parametry:

- název programovacího jazyka
- zdrojový kód k otestování
- sada automatických testů

Parametr *název programovacího jazyka* určuje, která *Dockerová image* se využije při vytváření izolovaného prostředí – kontejneru. Tato *image* je předpřipravený operační systém s předinstalovanými nástroji, které jsou zkonfigurované právě pro účely Code runneru.

Obrázky 2 a 3 ukazují obsah souborů *Dockerfile* pro aktuálně podporované programovací jazyky – C# a PHP. Tyto *Dockerfile* soubory jsou instrukce pro *Docker*, aby mohl správně sestavit *Docker image.*

```
FROM alpine:3.8
MAINTAINER Lubos Zapotocny <zapotocnylubos@gmail.com>

# Issues:
# * Added musl package because incompatibility in apline:3.8
#     https://github.com/frol/docker-alpine-mono/issues/9

ENV DEPENDENCIES_PATH=/executor/dependencies \
    WORSKPACE_PATH=/executor/workspace

ENV MUSL_VERSION=\>1.1.20 \
    MONO_VERSION=~5.10 \
    NUNIT_VERSION=3.11.0 \
    CONSOLE_RUNNER_VERSION=3.9.0

RUN apk add --no-cache musl=$MUSL_VERSION --repository http://dl-cdn.alpinelinux.org/alpine/edge/main && \
    apk add --no-cache mono=$MONO_VERSION --repository http://dl-cdn.alpinelinux.org/alpine/edge/testing && \
    apk add --no-cache --virtual=.build-dependencies ca-certificates && \
    cert-sync /etc/ssl/certs/ca-certificates.crt && \
    apk del .build-dependencies

WORKDIR $DEPENDENCIES_PATH

RUN wget https://dist.nuget.org/win-x86-commandline/latest/nuget.exe && \
    mono nuget.exe install NUnit -Version $NUNIT_VERSION -ExcludeVersion && \
    mono nuget.exe install NUnit.ConsoleRunner -Version $CONSOLE_RUNNER_VERSION -ExcludeVersion

ENV MONO_PATH=$DEPENDENCIES_PATH/NUnit.ConsoleRunner/tools:$DEPENDENCIES_PATH/NUnit/lib/net45

WORKDIR $WORSKPACE_PATH
```

Obrázek 2 - Dockerfile pro programovací jazyk C#

Pro programovací jazyk C# je v kontejneru předinstalované prostředí Mono, které umožňuje v operačním systému Linux spouštět programy napsané v programovacím jazyce C#. [10]

Předinstalovaný je také NUnit – open source testovací framework pro technologie Microsoft .NET. [11]

```
FROM alpine:3.8
MAINTAINER Lubos Zapotocny <zapotocnylubos@gmail.com>

ENV DEPENDENCIES_PATH=/executor/dependencies
ENV WORSKPACE_PATH=/executor/workspace

RUN apk add --no-cache php7 && \
    apk add --no-cache php7-phar && \
    apk add --no-cache php7-dom

WORKDIR $DEPENDENCIES_PATH

RUN wget https://phar.phpunit.de/phpunit-7.phar -O phpunit && \
    chmod +x phpunit && \
    mv phpunit /usr/local/bin/phpunit

WORKDIR $WORSKPACE_PATH
```

Obrázek 3 - Dockerfile pro programovací jazyk PHP

Programovací jazyk PHP je standardně podporován operačním systémem Alpine Linux, na kterém jsou založeny používané *Docker image*, proto jej lze nainstalovat přímo z oficiálních repozitářů. Testovací framework PHPUnit umožňuje spuštění automatických testů pro tento programovací jazyk.

Parametry *zdrojový kód k otestování* a *sada automatických testů* jsou obsahy souborů, které budou při spuštění kontejneru poskytnuty kontejneru ve sdílené složce.

Před spuštěním samotného kontejneru je potřeba vytvořit dočasnou složku pro předání daných parametrů (*zdrojový kód* a *sada automatických testů*) ve formě spustitelných souborů. Dané parametry jsou použity jako obsahy těchto souborů.

Tato složka je připojena ke kontejneru ve formě sdílené složky, což znamená, že kontejner může číst nebo modifikovat obsah této složky. Z ní je také po ukončení kontejneru načten soubor s výsledky automatických testů.

Takto vytvořený kontejner se stává velice dobře izolovaným prostředím od serverového operačního systému, což umožnuje bezpečné spuštění potenciálně nebezpečného uživatelského kódu.

Konfigurace kontejneru také osahuje vstupní příkaz. Tento příkaz je vykonán ihned po spuštění kontejneru a dokončením příkazu je kontejner automaticky ukončen. Tento příkaz nastavuje testovací nástroje a spouští automatické testy (Obrázek 4).

```
compile_command = " ".join([
    "mcs",
    "code.cs",
    "tests.cs",
    "-target:library",
    "-out:onlytests.dll",
    "-r:/executor/dependencies/NUnit/lib/net45/nunit.framework.dll",
])

run_tests_command = " ".join([
    "mono",
    "/executor/dependencies/NUnit.ConsoleRunner/tools/nunit3-console.exe onlytests.dll",
])

command = f"sh -c \"{compile_command} && {run_tests_command}\""
```

Obrázek 4 - Vstupní příkaz pro programovací jazyk C#

Program napsaný v programovacím jazyce C# je nutné před otestováním přeložit do strojového kódu, protože se jedná o kompilovaný jazyk. [12] K tomu slouží *kompilátor* MCS, který se sám poprvé zkompiloval 28. prosince 2001. Aktuálně je to standardní *kompilátor* pro jazyk C# na platformě Mono. [13]

```
command = " ".join([
    "phpunit",
    "--bootstrap autoload.php",
    ".",
    "--log-junit",
    "TestResult.xml"
])
```

Obrázek 5 - Vstupní příkaz pro programovací jazyk PHP

PHP je interpretovaný jazyk, proto není třeba zdrojový kód *kompilovat*. O spuštění zdrojového kódu se postará nainstalovaný *interpret* při spuštění kódu (Obrázek 5). [14]

Před ukončením automatických testů je ve sdílené složce vygenerován soubor s výsledky automatických testů. Obsah tohoto souboru je po ukončení kontejneru načten do paměti Code runneru.

Problém je, že nástroje pro automatické testování nemají jednotný formát výsledného souboru. Autor proto navrhl univerzální strukturu výsledných dat ve formátu JSON, která tento problém řeší.

```xml
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<test-run
        id="2" testcasecount="1" result="Passed" total="1" passed="1" failed="0" inconclusive="0" skipped="0"
        asserts="1" engine-version="3.9.0.0" clr-version="4.0.30319.42000" start-time="2019-03-11 07:35:26Z"
        end-time="2019-03-11 07:35:32Z" duration="5.831509">
    <command-line>
        <![CDATA[/executor/dependencies/NUnit.ConsoleRunner/tools/nunit3-console.exe onlytests.dll]]>
    </command-line>
    <test-suite
            type="Assembly" id="0-1002" name="onlytests.dll" fullname="onlytests.dll" runstate="Runnable"
            testcasecount="1" result="Passed" start-time="2019-03-11 07:35:31Z" end-time="2019-03-11 07:35:31Z"
            duration="0.564535" total="1" passed="1" failed="0" warnings="0" inconclusive="0" skipped="0"
            asserts="1">
        <environment
                framework-version="3.11.0.0" clr-version="4.0.30319.42000" os-version="Unix 4.9.125.0"
                platform="Unix"
                cwd="/executor/workspace" machine-name="c756fc02e830" user="root" user-domain="c756fc02e830"
                culture=""
                uiculture="" os-architecture="x64"/>
        <settings>
            <setting name="DisposeRunners" value="True"/>
            <setting name="WorkDirectory" value="/executor/workspace"/>
            <setting name="ImageRuntimeVersion" value="4.0.30319"/>
            <setting name="ImageRequiresX86" value="False"/>
            <setting name="ImageRequiresDefaultAppDomainAssemblyResolver" value="False"/>
            <setting name="NumberOfTestWorkers" value="2"/>
        </settings>
        <properties>
            <property name="_PID" value="12"/>
            <property name="_APPDOMAIN" value="domain-"/>
        </properties>
        <test-suite
                type="TestFixture" id="0-1000" name="HelloWorldTest" fullname="HelloWorldTest"
                classname="HelloWorldTest" runstate="Runnable" testcasecount="1" result="Passed"
                start-time="2019-03-11 07:35:31Z" end-time="2019-03-11 07:35:31Z" duration="0.345766" total="1"
                passed="1" failed="0" warnings="0" inconclusive="0" skipped="0" asserts="1">
            <test-case
                    id="0-1001" name="HelloWorldStringTest" fullname="HelloWorldTest.HelloWorldStringTest"
                    methodname="HelloWorldStringTest" classname="HelloWorldTest" runstate="Runnable"
                    seed="1123665869"
                    result="Passed" start-time="2019-03-11 07:35:31Z" end-time="2019-03-11 07:35:31Z"
                    duration="0.116648" asserts="1"/>
        </test-suite>
    </test-suite>
</test-run>
```

Obrázek 6 – Příklad obsahu výsledného souboru testovacího frameworku NUnit ve formátu XML (C#)

Obsah tohoto výsledného souboru (Obrázek 6) je následně převeden na univerzální výsledek, který má stejnou formu pro oba podporované programovací jazyky, a může být použit i pro libovolné další jazyky přidávané později (Obrázek 7).

```json
{
  "convertedResult": {
    "name": "onlytests.dll",
    "total": 1,
    "failures": 0,
    "duration": 0.564535,
    "testSuites": [
      {
        "name": "HelloWorldTest",
        "total": 1,
        "failures": 0,
        "duration": 0.345766,
        "testSuites": [],
        "testCases": [
          {
            "name": "HelloWorldStringTest",
            "duration": "0.116648",
            "failure": null,
            "output": null
          }
        ]
      }
    ],
    "testCases": []
  }
}
```

Obrázek 7 - Příklad zformátovaného výsledku automatických testů do univerzálního formátu

Kompletní výsledek obsahuje navíc výstupy ze *standardních proudů (stdout a stderr)*, které mohou obsahovat informace o tom, proč se nepovedlo spustit automatické testy nebo další důležité informace.

Tento kompletní výsledek je odeslán jako odpověď na HTTP požadavek.

4.3 Kontrola chyb

Pokud se při vykonávání testů vyskytne chyba nebo nelze automatické testy spustit, informace o této chybě jsou vypisovány do *standardních proudů (stderr a stdout)*. Výstupy z těchto proudů jsou součástí kompletního výsledku o spuštění automatických testů.

4.3.1 Kontrola syntaxe

Kontrola syntaxe upozorňuje programátora na nevalidní zápis zdrojového kódu. Poskytuje mu zpětnou vazbu s hlášením o výskytu chyby. Tato hlášení jsou následně odesílána v kompletním výsledku z Code runneru.

Například pokud ve zdrojovém kódu chybí středník na konci příkazu, ve výstupním okně se programátorovi zobrazí chybové hlášení, které ho upozorňuje na chybu v syntaxi daného programovacího jazyka, jak je tomu na obrázku 8.

C#: code.cs(7,8): error CS1525: Unexpected symbol `}', expecting `;'

PHP: PHP Parse error: syntax error, unexpected '}', expecting ';' in /executor/workspace/code.php on line 9

Obrázek 8 - Příklad chybového hlášení o nevalidní syntaxi

4.3.2 Nekonečný cyklus

Uživatelský kód může obsahovat nekonečný cyklus, který se neustále opakuje (Obrázek 9). Je to následek programátorské chyby, kdy programátor nenapíše ukončující podmínku. [15]

```
while (true) {
}
```

Obrázek 9 - Příklad nekonečného cyklu v programovacím jazyku C#

Tento problém je vyřešen *asynchronním* čekáním na dokončení běhu kontejneru po stanovenou dobu (10 sekund). Po vypršení tohoto časového limitu je kontejner ukončen.

Všechny získané informace jsou zpracovány a odeslány v kompletním výsledku z Code runneru.

4.4 Zabezpečení

Spuštění uživatelského kódu na vlastních serverech je velice nebezpečné, proto autor projektu řešil zabezpečení jako problém s nejvyšší prioritou. Program Code runner operuje na vlastním odděleném serveru bez možnosti jakékoli komunikace s internetem.

Uživatelský kód je spuštěn v *Dockerových kontejnerech*, které automaticky poskytují velice dobrou izolovanost od serverového operačního systému.

Poskytovatel serverového *hostingu* umožnuje privátní komunikaci mezi servery ve stejném *datacentru*. To znamená, že je možné komunikovat mezi vlastními servery pomocí *privátních IP adres*. Tato privátní komunikace zajišťuje, že se serverem, na kterém operuje Code runner mohou komunikovat pouze nakonfigurované servery.

Spolu s nastavením firewallu, které nepovoluje příchozí ani odchozí internetové připojení, toto řešení poskytuje ochranu i v případě, kdyby škodlivý uživatelský kód ovlivnil serverový operační systém.

5 Webová platforma

Webová platforma je program pro uživatele, kteří chtějí trénovat své programátorské dovednosti řešením algoritmických úloh. Všechny části uživatelského prostředí jsou přístupné přes libovolný moderní webový prohlížeč.

Při řešení jednotlivých algoritmických úloh je správnost uživatelského algoritmu vyhodnocena pomocí sady automatických testů.

5.1 Použité technologie

5.1.1 Django

Framework Django byl již popsán v sekci o použitých technologiích v programu Code runner (4.1.2).

Autor práce z tohoto frameworku využil velké množství předpřipravené funkcionality, což umožnilo relativně rychlý vývoj webové platformy.

5.1.2 PostgreSQL

PostgreSQL je objektově-relační databázový systém vydávaný pod licencí typu MIT a tudíž se jedná o open source software. [16]

Tato databáze byla zvolena jako nejvhodnější z důvodu existence možnosti vytvářet v jedné databázi více tabulek se stejným jménem. Podrobnější popis této funkcionality popisuje sekce „multitenance“ (5.1.4).

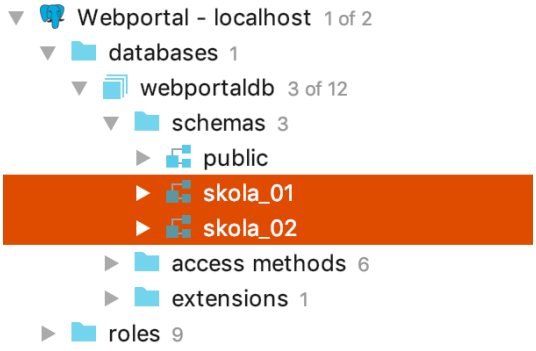
5.1.3 Bootstrap

Bootstrap je jednoduchá a volně stažitelná sada nástrojů pro tvorbu webu a webových aplikací. Výhodou této knihovny je snadné zpracování jakéhokoli uživatelského rozhraní ve webové aplikaci. [17]

5.1.4 Multitenance

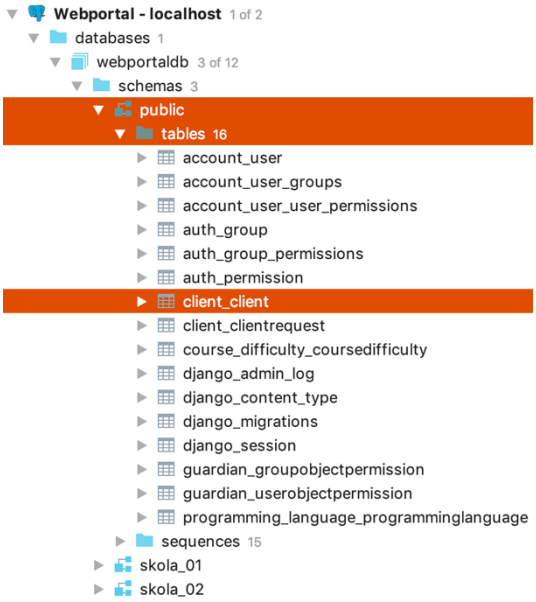
Multitenance je softwarová architektura, která umožnuje jedné instanci aplikace zprostředkovat své služby několika *tenantům*. [18]

K webové platformě je přiřazena jedna PostgreSQL databáze. V této databázi lze vytvářet více oddělených *schémat*, která umožňují vytvořit v databázi více tabulek se stejným názvem. Tato struktura by se dala přirovnat ke složkám v souborovém systému. Soubor se stejným názvem nemůže existovat v jedné složce dvakrát. Řešením je přesunout tyto soubory do svých vlastních složek. Podobný postup je u autorova řešení *multitenance*. Databázové tabulky jednotlivých *tenantů* jsou rozděleny do vlastních databázových *schémat*.



Obrázek 10 - Rozdělení databáze do schémat

Na obrázku 10 jsou vidět dvě *schémata* pojmenovaná „skola_01“ a „skola_02“. K těmto schématům existuje ve speciální tabulce „client_clients“ v „public“ *schématu* (Obrázek 11) přiřazená URL adresa (Tabulka 1).



Obrázek 11 - Lokace tabulky „client_clients“

id	domain_url	schema_name
6	skola01.domena.cz	skola_01
7	skola02.domena.cz	skola_02

Tabulka 1 – Obsah tabulky „client_clients“

Pomocí této tabulky a URL adresy požadavku lze určit ke kterému *schématu* chce uživatel přistupovat. Aplikace využije tento záznam a hodnotu ve sloupci „schema_name“ použije pro budoucí dotazy do databáze.

Schéma „public“ obsahuje speciální tabulky s daty, která jsou společná pro všechny *tenanty*, například seznam podporovaných programovacích jazyků.

5.2 Programovací cvičení

Cvičení je jedna z učebních metod, kde opakování dané aktivity může zlepšit její pochopení nebo zvládnutí. Například sportovní tým neustále trénuje před závodem. Hudebník několik let trénuje hru na hudební nástroj, aby tuto aktivitu plně zvládl.

Efektivita cvičení se odvíjí od několika faktorů:

- zručnost jedince
- četnost cvičení
- zpětná vazba

Pokud není poskytnuta vhodná zpětná vazba, například od instruktora nebo mentora, může mít cvičení minimální až odpudivý účinek. Pokud jedinec necvičí v dostatečně pravidelných intervalech, stává se cvičení také neefektivním. Z tohoto důvodu je cvičení dané aktivity

často plánováno v intervalech. Potřebná četnost se odvíjí od typu aktivity a zručnosti jedince. Někdo potřebuje cvičit více a častěji než ostatní. [19]

Programování

Programování je v informatice pojem pro řešení problému pomocí výpočetní techniky. Zahrnuje činnosti jako pochopení zadaného problému, nalezení optimálního algoritmu a zapsání daného algoritmu do zdrojového kódu. [20]

Programátor je ten, kdo toto programování provádí, analyzuje požadavky na nové programy a určuje návrh nového softwarového řešení. Tento návrh je vhodný pro všechny typy projektů a poskytuje programátorům jasně definované požadavky na systém. [21]

Programovací cvičení

Programovací cvičení jsou úlohy pro programátory, které jim připraví fiktivní problémy, na kterých si mohou procvičit jak své programátorské dovednosti, tak i své logické myšlení.

Obvykle takové cvičení obsahuje slovní popis, který programátorovi vysvětlí daný problém, například na popisu reálné situace. Příklad takového zadání ukazuje obrázek 12:

Napište funkci compare(a, b), která pro dvě zadaná čísla a, b vypíše informaci o tom, zda jsou stejná, nebo je jedno z nich větší (a které to je)

Obrázek 12 - Ukázkové zadání programovacího cvičení
dostupné z https://www.umimeprogramovat.cz/interaktivni-python-if/54

5.2.1.1 Již existující cizí řešení

Některé aplikace umožňují takto zadané cvičení řešit přímo ve svém prostředí – nejčastěji webové stránce. Například na webu https://www.codewars.com mohou uživatelé trénovat své programátorské dovednosti pomocí podobných cvičení, která jsou rozdělena do sedmi úrovní obtížnosti. Tato aplikace je vhodná pro pokročilé programátory. Nenapovídá totiž jaké postupy se k vyřešení problému mají použít a také zde neexistuje možnost trénovat specifickou oblast znalostí.

Webová aplikace dostupná na adrese https://www.freecodecamp.org je užitečná pro začínající programátory. Obsahuje sedm sekcí zaměřených na výuku programování webových aplikací. Problém je, že neobsahuje sady cvičení, které by si mohl programátor vyzkoušet za účelem lepšího zapamatování a procvičení probíraných sekcí.

Asi nejpodobnější této práci SOČ je řešení, které nabízí nedávno spuštěný placený Teacher mód serveru https://repl.it . Nicméně, tento má jen velmi slabou správu a třídění úloh – lze pouze vytvářet třídy, ne však už v nich např. úlohy řadit, považovat některé za povinné apod.

5.2.1.2 Vlastní řešení

V čem autorovo řešení vyniká mezi zmíněnými aplikacemi, je existence funkcionality sledování postupu řešení uživatelů. Instruktor vidí všechna finálně odeslaná řešení daného cvičení a může tento seznam filtrovat, aby získal například jenom správná řešení od určité skupiny uživatelů.

Tato možnost umožnuje instruktorům poskytnout velmi kvalitní zpětnou vazbu, protože si mohou prohlédnout autorovy změny v době řešení daného programovacího cvičení.

5.3 Funkcionalita

Webová platforma je webová stránka poskytující všechny potřebné služby při výuce programování. Implementuje záměr autora a zpracovává všechna uživatelská data. Poskytuje uživateli možnost procvičovat své programátorské dovednosti z libovolného moderního webového prohlížeče.

Sada programovacích cvičení

Uživatelé s dostatečným oprávněním mohou v aplikaci vytvářet programovací cvičení. Tato cvičení jsou zaměřena na trénink specifické oblasti výuky programování. Cvičení obsahuje tyto povinné atributy:

- název cvičení
- slovní popis
- kostra zdrojového kódu
- veřejná sada automatických testů
- neveřejná sada automatických testů

Atribut *název cvičení* určuje, pod jakým názvem se bude uživateli cvičení zobrazovat na stránce se seznamem cvičení. Například „První cvičení v PHP“. Pro uživatele to je jedna z nejdůležitějších informací před tím, než se rozhodne vyzkoušet si toto cvičení vyřešit.

Slovní popis cvičení je blok textu, který uživateli vysvětluje zadaný problém a případně objasňuje chování algoritmu (Obrázek 13). Pomocí tohoto popisu by měl uživatel pochopit, jaký je zadaný problém a jak ho má řešit. Například:

Upravte zdrojový kód tak, aby funkce "HelloWorldString" vracela řetězec "Hello World"

Obrázek 13 – Ukázka slovního popisu programovacího cvičení

Kostra zdrojového kódu (Obrázek 14) je předpřipravená část zdrojového kódu, která uživateli umožnuje se plně soustředit na psaní daného algoritmu.

```php
<?php
declare(strict_types=1);

final class HelloWorld
{
    public static function HelloWorldString()
    {
        // své řešení napište na tento řádek
    }
}
```

Obrázek 14 - Ukázka kostry zdrojového kódu programovacího cvičení

Veřejná sada automatických testů je malá předpřipravená sada automatických testů, které kontrolují správnost uživatelského řešení. Odesláním uživatelského řešení a sady automatických testů do programu Code runner je vyhodnocena správnost uživatelského algoritmu. Při řešení zadaného cvičení uživatel tuto sadu testů vidí (Obrázek 15) a může ji libovolně modifikovat.

```php
<?php
declare(strict_types=1);

use PHPUnit\Framework\TestCase;

final class HelloWorldTest extends TestCase
{
    public function testHelloWorld(): void
    {
        $this->assertEquals(
            'Hello World',
            HelloWorld::HelloWorldString()
        );
    }
}
```

Obrázek 15 - Ukázka veřejné sady automatických testů

Neveřejná sada automatických testů je uživateli skrytá a obvykle důkladnější než sada veřejná. Pokud uživatel označí své řešení jako finální, je toto řešení odesláno pro ověření funkčnosti s neveřejnými testy do programu Code runner. Pokud i tyto testy potvrdí funkčnost, je toto cvičení uživateli označeno jako úspěšně vyřešené.

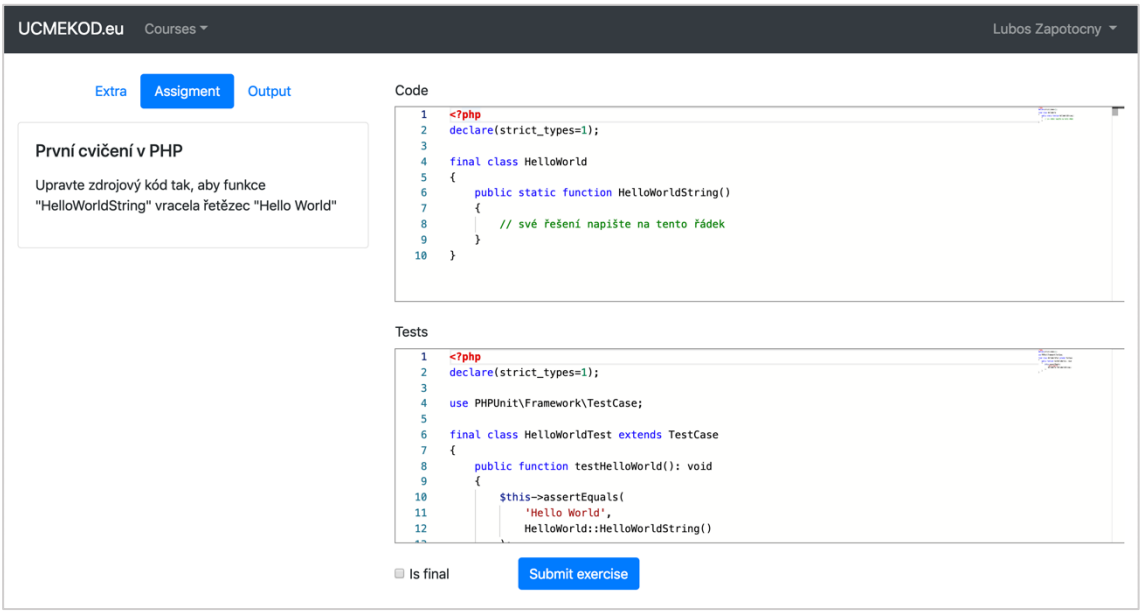
5.3.1 Uživatelské rozhraní

Uživatelské rozhraní pro řešení programovacích úloh je rozděleno do dvou částí: „informační panel“ a „pracovní plocha“.

Zobrazení informačního panelu je rozděleno do tří módů. Základní mód je „zadání“. Tento mód uživateli zobrazuje název a slovní popis daného programovacího cvičení. Druhý mód je „výstup“, který graficky zobrazuje výsledek spuštěných automatických testů. Také zobrazuje výstupy z obou *standardních proudů (stdout a stderr)*. Poslední mód nazvaný „extra“ obsahuje několik doplňkových funkcí.

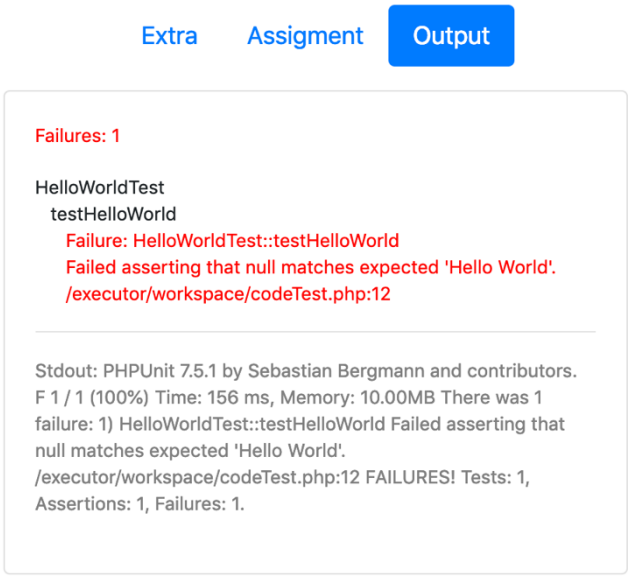
Pracovní plocha obsahuje dva editory kódu (Obrázek 16). Knihovna, která zprostředkovává tyto editory kódu se jmenuje Monaco Editor. Tuto knihovnu používá pro své editory kódu například velmi populární vývojové prostředí Visual Studio Code. [22]

Pod těmito editory je zaškrtávací políčko „Je finální“. Toto políčko indikuje, zdali uživatel chce otestovat své řešení na neveřejné sadě automatických testů.



Obrázek 16 - Ukázka stránky s detailem programovacího cvičení

Pokud uživatel výše zobrazený algoritmus odešle, ve výstupním panelu se mu zobrazí hlášení o chybě v automatickém testu, které říká, že hodnota *null* není očekávaný řetězec „Hello World“ (Obrázek 17). Tato chyba napovídá, že algoritmus nevrací správnou hodnotu.



Obrázek 17 .- Ukázka grafického výstupu automatických testů – nesprávné řešení

Uživatel poté může napsat kód podobný tomu na obrázku 18. Na první pohled toto řešení vypadá správně, ale správné řešení to není. Uživatel porušil syntaxi jazyka PHP a zapomněl na konci příkazu ukončující symbol – středník.

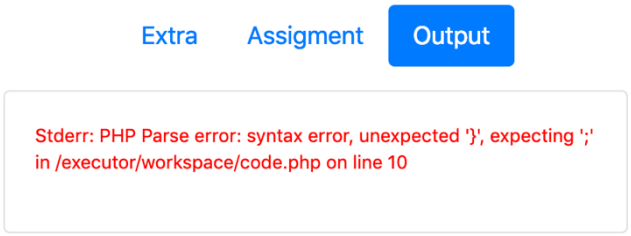
Code

```php
<?php
declare(strict_types=1);

final class HelloWorld
{
    public static function HelloWorldString()
    {
        // své řešení napište na tento řádek
        return "Hello World"
    }
}
```

Obrázek 18 - Ukázka programátorské chyby – zapomenutý středník

Důsledek této chyby je, že Code runner nemohl spustit automatické testy. Uživateli je zobrazen výstup ze *standardního proudu stderr* (Obrázek 19), který ho informuje o chybné syntaxi ve zdrojovém kódu.



Obrázek 19 - Ukázka grafického výstupu chybového hlášení

Po opravení této chyby výsledný uživatelský kód vypadá podobně jako na obrázku 20:

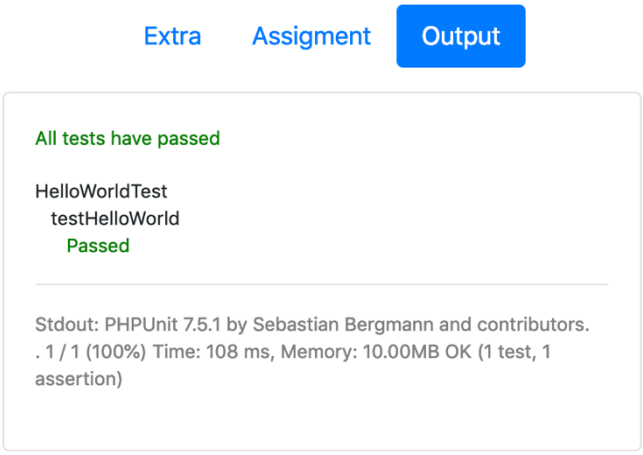
Code

```php
<?php
declare(strict_types=1);

final class HelloWorld
{
    public static function HelloWorldString()
    {
        // své řešení napište na tento řádek
        return "Hello World";
    }
}
```

Obrázek 20 - Ukázka správného kódu programovacího cvičení

Výsledek automatických testů v informačním panelu „výstup“ zeleným písmem oznamuje, že všechny automatické testy potvrdily správnou funkčnost algoritmu (Obrázek 21).



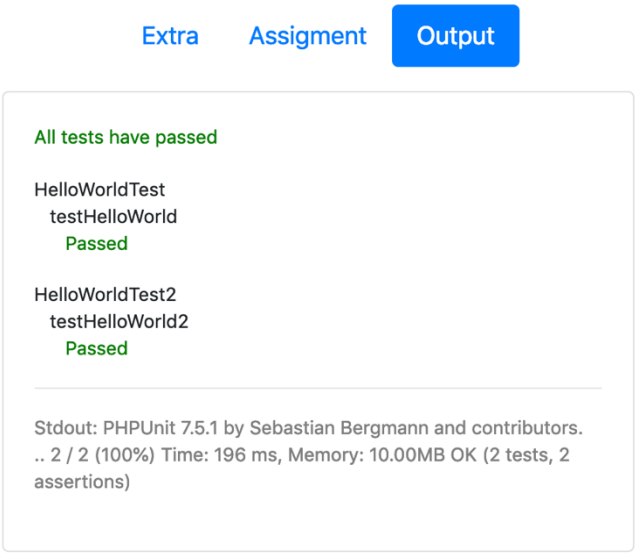
Obrázek 21 - Ukázka grafického výstupu automatických testů – správné řešení

Uživatelské řešení je nyní potřeba otestovat na neveřejné sadě automatických testů. Tato sada může obsahovat složitější či dynamické automatické testy. Tuto sadu automatických testů nemůže žák modifikovat, proto se lze na výsledek spolehnout.

Uživatel zaškrtne formulářové políčko „je finální“ a znovu odešle formulář (Obrázek 22).

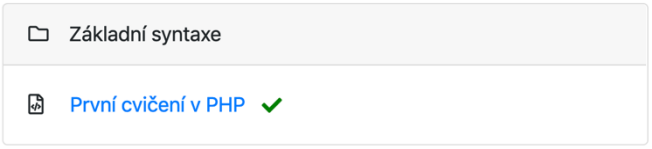
Obrázek 22 - Formulářové políčko "je finální"

Po dokončení automatického testování uživatel vidí jiný výsledek ve výstupním panelu (Obrázek 23). Tentokrát se spustily dva automatické testy, které jsou definovány v neveřejné sadě automatických testů stejným způsobem jako automatické testy ve veřejné sadě automatických testů.



Obrázek 23 - Ukázka grafického výstupu neveřejných automatických testů – správné řešení

Cvičení je nyní označeno jako úspěšně vyřešené (Obrázek 24).



Obrázek 24 - Indikátor úspěšně vyřešeného cvičení

5.4 Kurzy

Kurz je ucelený soubor přednášek a cvičení zaměřených na specifické téma. Kurzy existují v několika formách:

- prezenční
- dálkové
- e-learningové

E-learningové kurzy obsahují různé studijní materiály a úkoly, které lze řešit z libovolného místa. K těmto kurzům se nejčastěji přistupuje pomocí webového prohlížeče. [23]

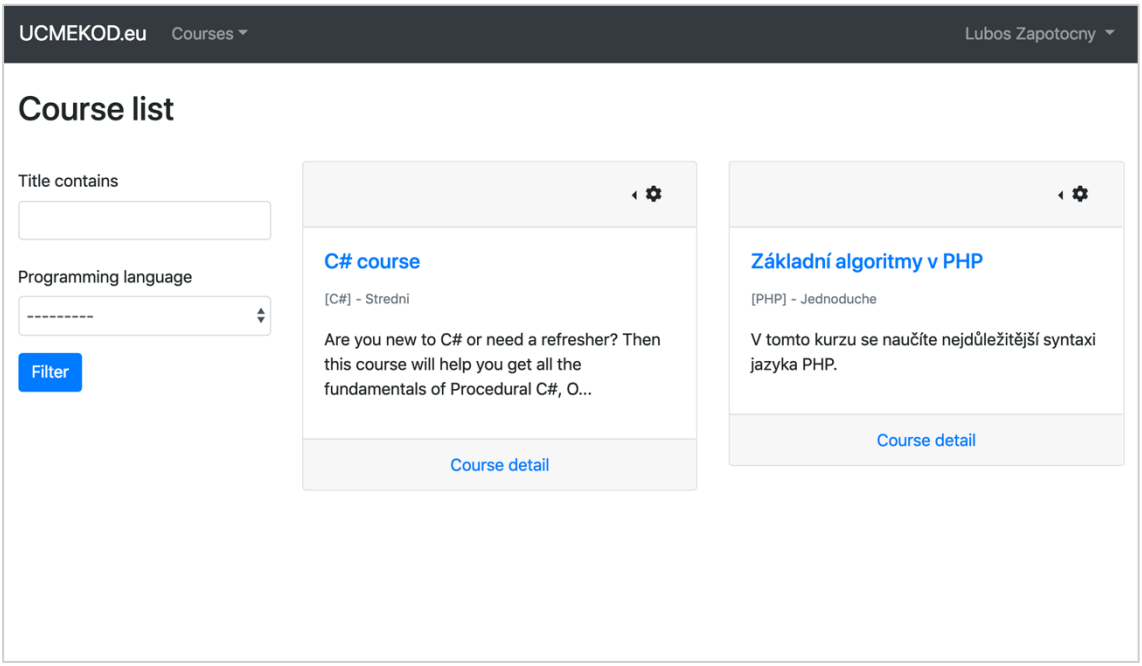
Webová platforma v rámci tohoto projektu definuje kurz jako logickou jednotkou obsahující různé sekce, které sdružují programovací cvičení dle obsahových podobností. Nastavení kurzu určuje chování všech podružných cvičení.

Nastavení kurzu obsahuje tyto atributy:

- název
- popis
- programovací jazyk
- obtížnost

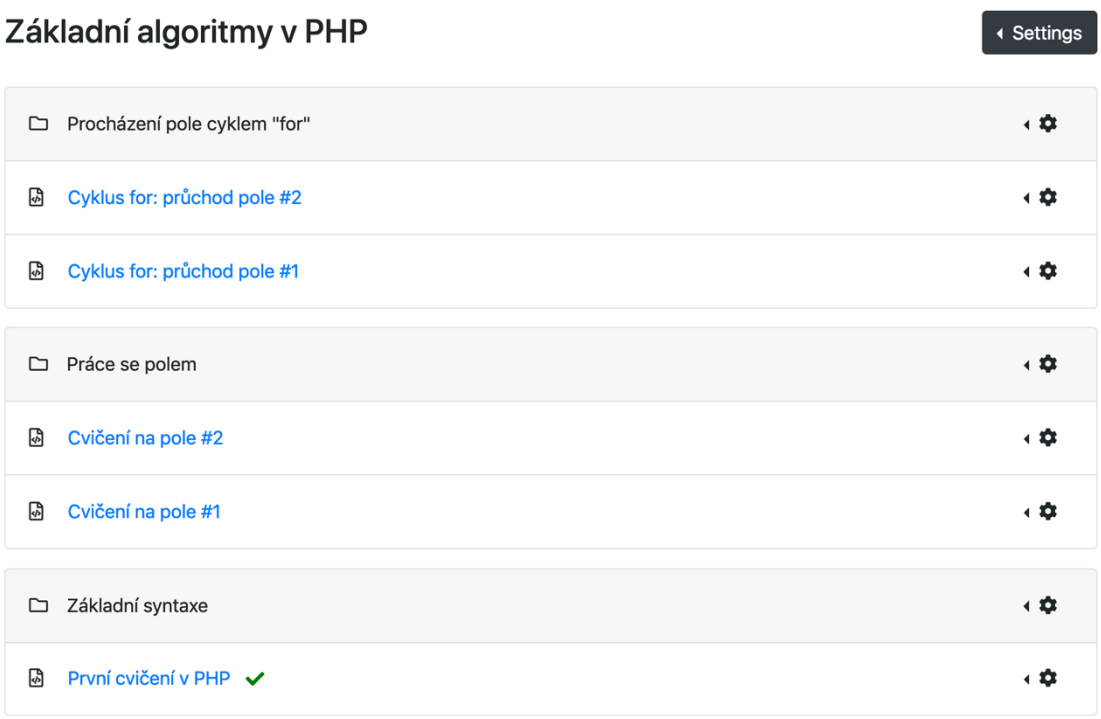
Atribut *programovací jazyk* nastavuje chování pro všechna cvičení v daném kurzu. Tento atribut je použit pro konfiguraci editorů kódu v daném kurzu. Ostatní atributy mají pouze informativní účel.

Stránka se seznamem přístupných kurzů obsahuje formulář pro filtrování a daný seznam kurzů (Obrázek 25).



Obrázek 25 - Stránka se seznamem kurzů

Stránka s detailem kurzu vypisuje uživateli seznam všech sekcí a cvičení v nich (Obrázek 26).



Obrázek 26 - Detailní stránka kurzu

5.5 Uživatelská oprávnění

Webová platforma rozlišuje několik základních uživatelských oprávnění. Některá z těchto oprávnění mají platnost v celé aplikaci a některá se přiřazují k specifickému objektu.

Všechna níže popsaná uživatelská oprávnění lze přiřadit jednotlivým uživatelům nebo skupině uživatelů. Lze tak docílit flexibility při přidělování práv uživatelům v aplikaci. Oprávnění přiřazená skupině jsou při kontrole práv vyhodnocena jako uživatelova vlastní.

Oprávnění „create_course“ umožňuje uživateli vytvářet nové kurzy a celý jejich obsah. Po vytvoření kurzu jeho autor automaticky získává oprávnění „manage_course“ k danému kurzu.

```
@method_decorator(permission_required('course.create_course'), name='dispatch')
class CourseCreateView(CreateView):
    model = Course
    form_class = forms.CourseCreateForm
    template_name = 'course/create.html'

    def form_valid(self, form):
        response = super().form_valid(form)
        assign_perm('course.manage_course', self.request.user, self.object)
        return response
```

Obrázek 27 - Definice stránky pro vytváření kurzu

V kódu na obrázku 27 lze nalézt definici ochranné funkce, která zamezí přístup na danou stránku všem uživatelům, kteří nemají přiřazené oprávnění „create_course“.

Manažeři kurzu (uživatelé s oprávněním „manage_course“) mohou manipulovat jenom s těmi kurzy, ke kterým mají toto oprávnění přiřazeno. Poté mohou modifikovat všechny atributy kurzu, sekcí a programovacích cvičení daného kurzu. Také mají přístup ke všem finálně odeslaným uživatelským řešením všech programovacích cvičení v daném kurzu. Oprávnění „manage_course“ uživateli automaticky povoluje všechny funkce, které umožňuje oprávnění „view_course“.

```
class CourseUpdateView(PermissionRequiredMixin, UpdateView):
    permission_required = 'course.manage_course'

    model = Course
    form_class = forms.CourseUpdateForm
    template_name = 'course/update.html'

    def get_success_url(self):
        return self.request.GET.get('next', self.object.get_absolute_url())
```

Obrázek 28 - Příklad definice stránky ochráněné oprávněním k specifickému objektu

Oprávnění view_course je také přidělováno ke konkrétnímu kurzu (Obrázek 28). Uživateli se po přidělení tohoto oprávnění daný kurz zobrazuje v seznamu „moje kurzy“. Společně s oprávněním view_course získává uživatel přístup ke všem programovacím cvičením obsaženým v daném kurzu.

```python
class CourseListView(AnyPermissionListMixin, ListView):
    permission_required = ['course.manage_course', 'course.view_course']

    model = Course
    context_object_name = 'courses'
    template_name = 'course/list.html'

    def get_queryset(self, *args, **kwargs):
        qs = super().get_queryset()
        self.filter = filters.CourseFilter(self.request.GET, queryset=qs)
        return self.filter.qs

    def get_context_data(self, *, object_list=None, **kwargs):
        context = super().get_context_data()
        context.update({
            'filter': self.filter
        })
        return context
```

Obrázek 29 - Příklad definice stránky „moje kurzy“

Kód na obrázku 29 obsahuje upravené chování, které automaticky filtruje kurzy, ke kterým má uživatel oprávnění „manage_course“ nebo „view_course“. Zároveň obsahuje formulář na filtrování kurzů.

5.6 Zabezpečení

Použitý webový framework Django obsahuje velmi dobré zabezpečení. Níže jsou popsány nejčastější typy kybernetických útoků. Autor projektu si je rizik těchto útoků vědom a vytvořený systém je i díky použití frameworku Django vůči podobným zranitelnostem imunní.

5.6.1 Cross-site scripting

Cross-site scripting je metoda narušení webových stránek pomocí které útočník dokáže do stránek podstrčit svůj vlastní *JavaScriptový kód*, což může využít buď pouze k poškození vzhledu stránky, jejímu znefunkčnění, získávání citlivých údajů návštěvníků stránek nebo obcházení bezpečnostních prvků aplikace. Často je též využíván při *phishingu* tak, že je skrze tuto zranitelnost uživateli zobrazen jiný obsah na jinak důvěryhodné stránce. [24]

Django všechna data, která dynamicky vypisuje do šablon *escapuje*. Touto metodou lze zamezit útokům, kdy si uživatel například místo jména nastaví speciální HTML značky. Při vypisování tohoto jména bez metody *escapování* je toto jméno s HTML značkami začleněno jako součást dané webové stránky, přičemž vložené HTML značky mohou obsahovat i škodlivé JavaScriptové kódy upravující například akce jednotlivých uživatelů. [25]

5.6.2 Cross-site request forgery

Cross-site Request Forgery je jedna z metod útoku do internetových aplikací pracující na bázi nezamýšleného požadavku pro vykonání určité akce v této aplikaci, který ovšem pochází z nelegitimního zdroje. Většinou se nejedná o útok směřující k získání přístupu do aplikace. [24]

Formuláře vytvořené pomocí frameworku Django automaticky obsahují skryté políčko s tajným *tokenem*. Tento token je pro každý formulář specifický, jednorázový a má omezenou platnost. Tímto je zamezeno možnosti jednoduše zopakovat odeslání formuláře. [26]

5.6.3 SQL injection

SQL injection je technika napadení databázové vrstvy programu vsunutím kódu přes neošetřený uživatelský vstup a vykonání vlastního poškozujícího SQL příkazu. Toto nezamýšlené a neošetřené chování vzniká při propojení aplikační vrstvy s databázovou vrstvou a zabraňuje se mu pomocí jednoduchého *escapování* potenciálně nebezpečných znaků. [24]

Django při vytváření dotazů používá metodu parametrizace SQL dotazů. Tato metoda nejdříve vytvoří požadovaný dotaz do databáze pouze se zástupnými symboly, do kterých se poté předají parametry. Rozdělení těchto dvou kroků se provádí, protože dané parametry mohou být definované uživatelem a obsahovat nebezpečný kód. [27]

6 ZÁVĚR

Byl vyvinut systém, který umožňuje uživatelům vytvářet programovací cvičení a testy a řadit je do sekcí a kurzů. Pomocí webového prostředí lze řešit tyto úlohy bez nutnosti instalace dodatečného softwaru. Učitelé s dostatečným oprávněním si mohou zobrazit seznam odeslaných žákovských řešení, pomocí kterých mohou žákovi poskytnout velmi kvalitní zpětnou vazbu.

Systém lze distribuovat pomocí metody multitenance. Odesláním jednoho formuláře může *superadministrátor* vytvořit nového *tenanta*.

Uživatelská oprávnění lze přiřazovat specifickému uživateli nebo skupinám uživatelů.

Databázový model kurzu je natolik abstraktní, že lze do jeho sekcí přidávat jakékoli typy obsahu, které budou v aplikaci vytvořeny (např. cvičení, testy, přílohy).

Implementace nového programovacího jazyka do Code runneru je relativně jednoduchá a univerzální.

Budoucí možnosti systému

Do systému bude implementována pohodlná administrace uživatelských účtů. Pro učitele bude zpracován přehledný přístup k údajům o průchodu žáků jednotlivými kurzy. Vnitřní motivace žáků by mohla být podpořena začleněním prvků tzv. gamifikace do systému (odznaky, žebříčky apod.).

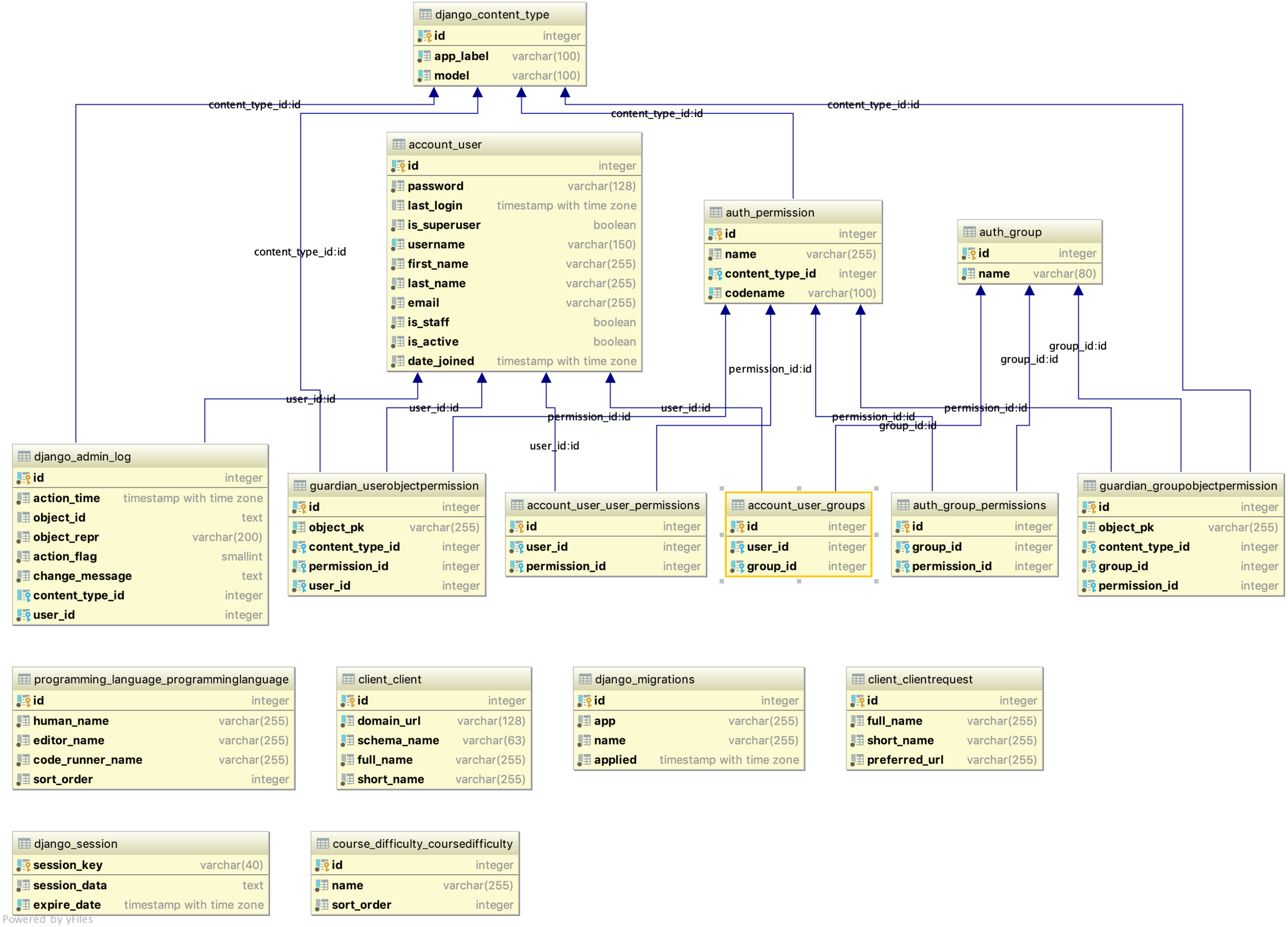
7 BIBLIOGRAFIE

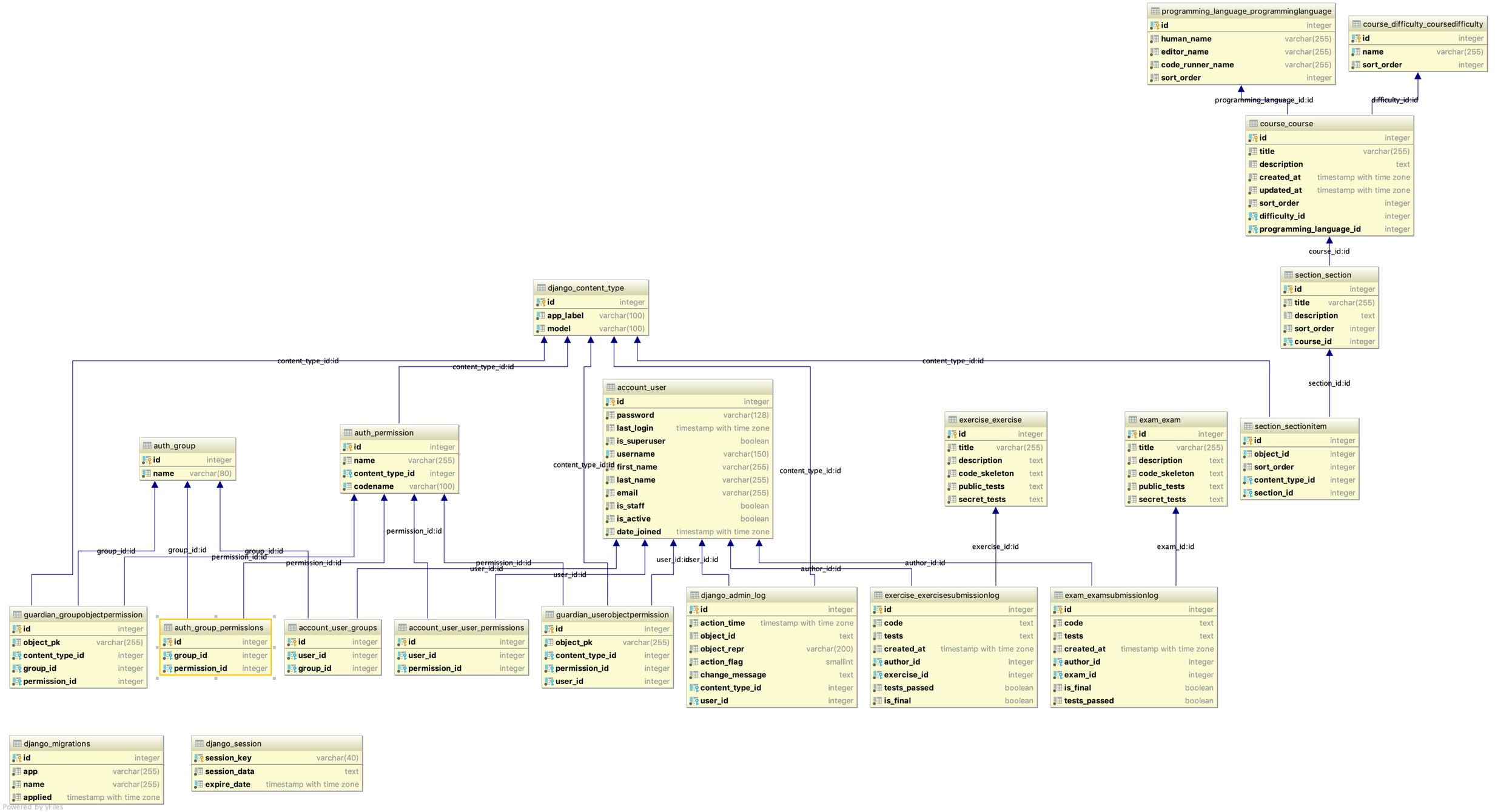
1. THUONG LY, Nguyen. Místo výuky programování přinejlepším Excel. Česku odjíždí informatický vlak Více na https://www.e15.cz/clanek/domaci/misto-vyuky-programovani-prinejlepsim-excel-cesku-odjizdi-informaticky-vlak-1339846. *E15.cz - Byznys, politika, ekonomika, finance, události* [online]. 2017, 15. listopadu 2017 18:54, , 1 [cit. 2019-03-14]. Dostupné z: https://www.e15.cz/clanek/domaci/misto-vyuky-programovani-prinejlepsim-excel-cesku-odjizdi-informaticky-vlak-1339846
2. API. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-13]. Dostupné z: https://cs.wikipedia.org/wiki/API
3. Representational State Transfer. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-10]. Dostupné z: https://cs.wikipedia.org/wiki/Representational_State_Transfer
4. Standard streams. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-03]. Dostupné z: https://en.wikipedia.org/wiki/Standard_streams
5. Software development kit. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-13]. Dostupné z: https://cs.wikipedia.org/wiki/Software_development_kit
6. Multitenancy. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2019 [cit. 2019-02-12]. Dostupné z: https://en.wikipedia.org/wiki/Multitenancy
7. Docker (software). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-02-21]. Dostupné z: https://en.wikipedia.org/wiki/Docker_(software)
8. Docker SDK for Python. *Docker SDK for Python — Docker SDK for Python 3.7.0 documentation* [online]. Docker, c2019 [cit. 2019-03-11]. Dostupné z: https://docker-py.readthedocs.io/en/stable/
9. Django. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-09]. Dostupné z: https://cs.wikipedia.org/wiki/Django
10. Mono (platforma). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-10]. Dostupné z: https://cs.wikipedia.org/wiki/Mono_%28platforma%29
11. NUnit. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-10]. Dostupné z: https://en.wikipedia.org/wiki/NUnit
12. Kompilovaný jazyk. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-11]. Dostupné z: https://cs.wikipedia.org/wiki/Kompilovan%C3%BD_jazyk
13. C# Compiler. *Home | Mono* [online]. c2019 [cit. 2019-03-11]. Dostupné z: https://www.mono-project.com/docs/about-mono/languages/csharp/

14. Interpretovaný jazyk. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-11]. Dostupné z: https://cs.wikipedia.org/wiki/Interpretovan%C3%BD_jazyk
15. Nekonečný cyklus. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-04]. Dostupné z: https://cs.wikipedia.org/wiki/Nekone%C4%8Dn%C3%BD_cyklus
16. PostgreSQL. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-09]. Dostupné z: https://cs.wikipedia.org/wiki/PostgreSQL
17. Bootstrap. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-13]. Dostupné z: https://cs.wikipedia.org/wiki/Bootstrap
18. Multitenancy. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2019 [cit. 2019-02-12]. Dostupné z: https://en.wikipedia.org/wiki/Multitenancy
19. Practice (learning method). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-12]. Dostupné z: https://en.wikipedia.org/wiki/Practice_(learning_method)
20. Programování. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-12]. Dostupné z: https://cs.wikipedia.org/wiki/Programov%C3%A1n%C3%AD
21. Programátor. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-12]. Dostupné z: https://cs.wikipedia.org/wiki/Program%C3%A1tor
22. Monaco Editor. *Monaco Editor* [online]. Microsoft, c2019 [cit. 2019-03-09]. Dostupné z: https://microsoft.github.io/monaco-editor/index.html
23. Kurz (pedagogika). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-11]. Dostupné z: https://cs.wikipedia.org/wiki/Kurz_(pedagogika)
24. Security in Django. *The Web framework for perfectionists with deadlines | Django* [online]. Django Software Foundation, c2005-2019 [cit. 2019-03-13]. Dostupné z: https://docs.djangoproject.com/en/2.1/topics/security/
25. Cross-site scripting. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-13]. Dostupné z: https://cs.wikipedia.org/wiki/Cross-site_scripting
26. Cross-site request forgery. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-13]. Dostupné z: https://cs.wikipedia.org/wiki/Cross-site_request_forgery
27. SQL injection. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-03-13]. Dostupné z: https://cs.wikipedia.org/wiki/SQL_injection

8 SEZNAM OBRÁZKŮ A TABULEK

PŘÍLOHA 1: DATOVÝ MODEL


django_content_type
id integer
app_label varchar(100)
model varchar(100)
account_user
id integer
password varchar(128)
last_login timestamp with time zone
is_superuser boolean
username varchar(150)
first_name varchar(255)
last_name varchar(255)
email varchar(255)
is_staff boolean
is_active boolean
date_joined timestamp with time zone
auth_permission
id integer
name varchar(255)
content_type_id integer
codename varchar(100)
auth_group
id integer
name varchar(80)
django_admin_log
id integer
action_time timestamp with time zone
object_id text
object_repr varchar(200)
action_flag smallint
change_message text
content_type_id integer
user_id integer
guardian_userobjectpermission
id integer
object_pk varchar(255)
content_type_id integer
permission_id integer
user_id integer
account_user_user_permissions
id integer
user_id integer
permission_id integer
account_user_groups
id integer
user_id integer
group_id integer
auth_group_permissions
id integer
group_id integer
permission_id integer
guardian_groupobjectpermission
id integer
object_pk varchar(255)
content_type_id integer
group_id integer
permission_id integer
programming_language_programminglanguage
id integer
human_name varchar(255)
editor_name varchar(255)
code_runner_name varchar(255)
sort_order integer
client_client
id integer
domain_url varchar(128)
schema_name varchar(63)
full_name varchar(255)
short_name varchar(255)
django_migrations
id integer
app varchar(255)
name varchar(255)
applied timestamp with time zone
client_clientrequest
id integer
full_name varchar(255)
short_name varchar(255)
preferred_url varchar(255)
django_session
session_key varchar(40)
session_data text
expire_date timestamp with time zone
course_difficulty_coursedifficulty
id integer
name varchar(255)
sort_order integer
content_type_id:id
user_id:id
permission_id:id
group_id:id
Powered by yFiles


programming_language_programminglanguage
id integer
human_name varchar(255)
editor_name varchar(255)
code_runner_name varchar(255)
sort_order integer
course_difficulty_coursedifficulty
id integer
name varchar(255)
sort_order integer
programming_language_id:id
difficulty_id:id
course_course
id integer
title varchar(255)
description text
created_at timestamp with time zone
updated_at timestamp with time zone
sort_order integer
difficulty_id integer
programming_language_id integer
course_id:id
section_section
id integer
title varchar(255)
description text
sort_order integer
course_id integer
section_id:id
django_content_type
id integer
app_label varchar(100)
model varchar(100)
content_type_id:id
account_user
id integer
password varchar(128)
last_login timestamp with time zone
is_superuser boolean
username varchar(150)
first_name varchar(255)
last_name varchar(255)
email varchar(255)
is_staff boolean
is_active boolean
date_joined timestamp with time zone
exercise_exercise
id integer
title varchar(255)
description text
code_skeleton text
public_tests text
secret_tests text
exam_exam
id integer
title varchar(255)
description text
code_skeleton text
public_tests text
secret_tests text
section_sectionitem
id integer
object_id integer
sort_order integer
content_type_id integer
section_id integer
auth_group
id integer
name varchar(80)
auth_permission
id integer
name varchar(255)
content_type_id integer
codename varchar(100)
group_id:id
permission_id:id
user_id:id
author_id:id
exercise_id:id
exam_id:id
guardian_groupobjectpermission
id integer
object_pk varchar(255)
content_type_id integer
group_id integer
permission_id integer
auth_group_permissions
id integer
group_id integer
permission_id integer
account_user_groups
id integer
user_id integer
group_id integer
account_user_user_permissions
id integer
user_id integer
permission_id integer
guardian_userobjectpermission
id integer
object_pk varchar(255)
content_type_id integer
permission_id integer
user_id integer
django_admin_log
id integer
action_time timestamp with time zone
object_id text
object_repr varchar(200)
action_flag smallint
change_message text
content_type_id integer
user_id integer
exercise_exercisesubmissionlog
id integer
code text
tests text
created_at timestamp with time zone
author_id integer
exercise_id integer
tests_passed boolean
is_final boolean
exam_examsubmissionlog
id integer
code text
tests text
created_at timestamp with time zone
author_id integer
exam_id integer
is_final boolean
tests_passed boolean
django_migrations
id integer
app varchar(255)
name varchar(255)
applied timestamp with time zone
django_session
session_key varchar(40)
session_data text
expire_date timestamp with time zone
Powered by yFiles