**Vyšší odborná škola a Střední průmyslová škola elektrotechnická Olomouc,**

**Božetěchova3**

**PRAKTICKÁ ZKOUŠKA Z ODBORNÝCH PŘEDMĚTŮ**

Tvorba systém kontroly výdeje obědů

2019 Ondřej WÜNSCH

Prohlašuji, že jsem praktickou zkoušku z odborných předmětů vypracoval samostatně a všechny prameny jsem uvedl v seznamu použité literatury.

……………………………  
 jméno a příjmení žáka

Chtěl bych vyslovit poděkování panu Ing. Marku Nožkovi za odborné konzultace a poskytnuté informace.

……………………………  
 jméno a příjmení žáka

Prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé práce nebo její části se souhlasem školy.

……………………………  
 jméno a příjmení žáka

# Abstrakt

1. V tomto projektu jsem se zaměřil na vylepšení školního sytému pro vydávání obědů. Snažil jsem se přidat funkce které by ulehčili práci obsluze kuchyně a dozoru v jídelně.

2. Cílem práce bylo vytvořit nový systém, který bude číst id žáků z jejich ISIC karet a kontrolovat zda oběd mají objednán a jestli přišly ve správný čas daný rozvrhem.

3. Systém je realizován pomocí webového serveru. Tuto metodu jsem zvolil, kvůli vysoké kompatibilitě a možnosti úprav. Server je možné spustit na PC s operačním systémem Windows nebo Linux a jako monitor pro zobrazení výdeje obědů lze použít jakékoli jiné zařízení v síti (např. tablet) nebo monitor připojený k hlavnímu zařízení. Také je možné živě měnit rozvrh podle potřeby.

4. Přínosem mé práce je zefektivnění výdeje obědů ve školní jídelně a odstranění tříd nedodržujících rozvrh. Také mi přinesla mnoho zkušeností v práci na větších projektech a hledání informací ohledně problémů, které při práci nastanou.

# Obsah

[Úvod 6](#_Toc4705715)

[1. Komunikace 7](#_Toc4705716)

[1.1 Knihovna Flask 7](#_Toc4705717)

[1.1.1 Základní struktura Flask aplikace 7](#_Toc4705718)

[1.2 Nástavba flask-socketio 9](#_Toc4705719)

[1.2.1 Základ socketio komunikace 9](#_Toc4705720)

[2. Server 12](#_Toc4705721)

[2.1 Obecná funkčnost 12](#_Toc4705722)

[2.2 Čtení karet 12](#_Toc4705723)

[2.3 Dělení na vlákna 13](#_Toc4705724)

[2.4 Komunikace s databází 13](#_Toc4705725)

[2.5 Rozvrhy 15](#_Toc4705726)

[2.5.1 Struktura rozvrhu 15](#_Toc4705727)

[2.5.2 Kontrola rozvrhu 16](#_Toc4705728)

[2.5.3 Kontrola času 17](#_Toc4705729)

[3. Klient 18](#_Toc4705730)

[3.1 Obecná funkčnost 18](#_Toc4705731)

[3.2 Hlavní stránka 18](#_Toc4705732)

[3.3 Stránka pro změny rozvrhů 19](#_Toc4705733)

[Závěr 21](#_Toc4705734)

[Seznam použité literatury 22](#_Toc4705735)

[Seznam obrázků 23](#_Toc4705736)

[Přílohy 24](#_Toc4705737)

# Úvod

Téma této práce jsem si vybral kvůli problému, který mě provázel celými čtyřmi roky studia. Tímto problémem byla přeplněná školní jídelna a nedostatek organizace. Pokud jsme na oběd nevyšli už před zvoněním, bylo jasné že strávíme většinu přestávky v řadě. Většina tříd totiž nedodržuje rozvrh přestávek určených na obědy.

Můj vylepšený systém toto řeší tím, že kontroluje nejen zda má žák oběd objednán, ale i rozvrh přestávek. Tudíž strávníkům nedodržujícím rozvrh jednoduše oběd nevydá. Samozřejmě jsem implementoval i výjimky pro žáky, kteří mají pro porušení rozvrhu dobrý důvod a obsluha kuchyně má tedy možnost oběd vydat i přes porušení podmínky.

Celý systém funguje na principu webového serveru a klienta, který se na něj připojuje. V první kapitole se tedy zaměřím na komunikaci mezi nimi, protože ta je na obou stranách řešena podobným způsobem.

V kapitole druhé popíšu vnitřní funkce serveru, jako je: čtení karet, komunikace se školní databází a kontrola rozvrhů.

V poslední kapitole vysvětlím funkce dostupné obsluze přes stránky v prohlížeči.

# Komunikace

## Knihovna Flask

Jako základ celého systému požívám knihovnu Flask pro programovací jazyk Python. Je to velice jednoduchý framework pro tvorbu webových aplikací. Jednoduchost použití ale neubírá na použitelnosti knihovny i pro větší projekty. Díky vysoké popularitě a aktivní komunitě bylo vytvořeno mnoho nástaveb, které rozšiřují funkčnost. V tomto projektu ale nepoužívám většinu funkcí, které knihovna nabízí. Proto se zaměřím jen na základní funkce, které v projektu používám. Další informace je možné najít v dokumentaci zmíněné v kapitole použité literatury.

### Základní struktura Flask aplikace

Pro vysvětlení funkce bych rád popsal základní strukturu serveru ve Flask knihovně. Na tomto základním modelu je pak postavena celá aplikace.

**from** flask **import** Flask

app **=** Flask**(**\_\_name\_\_**)**

**@app.route(**"/"**)**

**def** index**():**

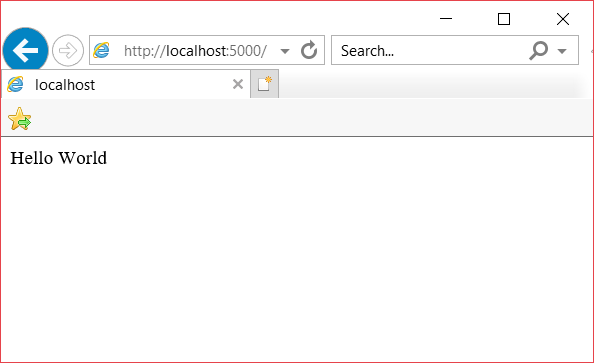
**return** "Hello World"

**if** \_\_name\_\_ **==** "\_\_main\_"**:**

app**.**run**()**

1. Nejprve importujeme třídu Flask() a případně další moduly a funkce z knihovny.
2. Vytvoříme instanci dané třídy většinou označovanou jako „app“. V této části je také dobré dle potřeby definovat globální proměnné pro pozdější použití.
3. Dále definujeme takzvané „routes“ neboli funkce označené dekorátorem route(). Tyto funkce jsou pak aktivovány když server dostane dotaz s adresou dané funkce. V tomto případě na dotaz „/“ server odpoví řetězcem „Hello World“.
4. Nakonec server spustit příkazem app.run().

Tento jednoduchý kód vytvoří lokálně přístupný server, ke kterému se připojíme z jakéhokoli prohlížeče adresou: http://localhost:5000. Po připojení prohlížeč automaticky vyšle dotaz „/“ nebo „/index“ a server provede funkci index() Takže nás v prohlížeči program přivítá frází „Hello World“.

Tato základní funkčnost je však velice přizpůsobitelná. Namísto jednoduchého řetězce mohou funkce odesílat celé HTML dokumenty a spouštět interní funkce serveru. V mém programu tento způsob komunikace používám jen pro načínání prvotní načítání stránek.

Obrázek : Zobrazení základního programu

**@app.route(**"/"**)**

**@app.route(**"/index"**)**

**def** index**():**

**return** render\_template**(**"table.html"**)**

**@app.route(**"/rozvrhy"**)**

**def** interface**():**

**with** open**(**"schedule.json"**,** "r"**)** **as** json\_data**:**

schedule\_table **=** json**.**load**(**json\_data**)**

**return** render\_template**(**"rozvrhy.html"**,** schedule\_table**=**schedule\_table**)**

Funkce render\_template() je součástí šablonovacího jazyka Jinja2, který umožňuje, mimo jiné, vkládat do HTML dokumentu proměnné.

var schedule\_table = {{schedule\_table | safe}};

Tedy v dokumentu rozvrhy.html nahradí {{schedule\_table | safe}} daty přečtenými ze souboru schedule.json. Toto zrychluje načítání stránky, protože se stránka o data nemusí později dotazovat.

## Nástavba flask-socketio

Kvůli použití serveru v tomto projektu spíše jako aktivní části a klienta jen jako displeje, který zobrazuje výdaje obědů, není základní komunikační model vhodný. Knihovna Flask totiž umožňuje jen „podmíněnou“ komunikaci. Tedy klient vysílá dotazy a server odpovídá. Flask-socketio zprostředkuje oboustrannou komunikaci, takže je možné odesílat data klientovi bez dotazu z jeho strany. K dekorátorům route() se přidá socketio.on(). Ten funguje stejně, ale reaguje na dotazy socketio.emit(), které je možné nepodmíněně vysílat z obou stran.

### Základ socketio komunikace

V základí flask plikaci je třeba provést jen pár změn, aby se přizpůsobila socketio modelu.

**from** flask **import** Flask

**from** flask\_socketio **import** SocketIO

app **=** Flask**(**\_\_name\_\_**)**

socketio **=** SocketIO**(**app**)**

**@app.route(**"/"**)**

**def** index**():**

**return** render\_template**(**"index.html"**)**

**@socketio.on(**"connected"**)**

**def** connector**(**data**):**

**print(**data**[**"data"**])**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

socketio**.**run**(**app**)**

1. Z knihovny flask-socketio importujeme třídu SocketIO a vytvoříme její instanci.
2. Změníme funkci index(), aby odesílala HTML dokument, ve kterém je uložen komunikační skript.
3. Vytvoříme funkci pro obsluhu socketio dotazů. V tomto případě je to jediná funkce connector().
4. Aplikaci nyní spouštíme pomocí socketio.run(app). Což donutí program přijímat socketio dotazy. Pokud by jsme tuto změnu neprovedli, aplikace by nezačala na dotazy reagovat.

Komunikační skript v index.html:

**var** socket **=** io**.**connect**(**"http://localhost:5000"**);**

socket**.**on**(**"connect"**,** **function(){**

console**.**log**(**"connect"**);**

socket**.**emit**(**"connected"**,** **{**data**:** "connected"**});**

**});**

1. V dokumentu index.html je nejprve nutné importovat knihovnu pro komunikaci pomocí socketio protokolu. Toto většinou provádíme v hlavičce dokumentu.
2. Na začátku komunikace otevřeme socketio kanál na dané adrese a portu. Definováním více kanálů (pod jinýmy jmény) je možné se připojit a odpovídat na více adres.
3. Funkce pro ošetřování dotazů definujeme podobně jako na straně severu.

Po připojení se na server v prohlížeči, vyšle server stránku index.html. Po načtení stránky se aktivuje skript, ten otevře komunikační kanál, po kterém je pak možné posílat dotazy. Když je kanál otevřen, funkce socket.on(“connect“) vyšle serveru zprávu socket.emit(“conected“, {data: “conected“ }). Sever tuto správu přijme a vypíše do konsole „connected“.

Na tomto jednoduchém modelu je pak postavená celá komunikace a přenos dat v projektu. Díky nepodmíněnému vysílání je možné posílat správy a data kdykoli a ze kterékoli části kódu.

Například aktivované stiskem tlačítka v prohlížeči:

**<button** id="send\_button" onclick="send\_data()"**>**Odeslat rozvrh**</button>**

**function** send\_data(){

console.log("data sent");

socket.emit("schedule\_change",   
{data: schedule\_table});

}

Tato funkce odešle změny v rozvrhu serveru po stisku tlačítka.

Nebo z interní funkce serveru:

**def** dispence\_lunch**(**consumer**,** card\_id**,** lunch\_amount**):**

**print(**"Lunch dispenced"**)**

**if** consumer**[2]** **==** **1:**

lunch\_amount**[**"ones"**]** **-=** **1**

playsound**(**"audio/one.wav"**)**

**elif** consumer**[2]** **==** **2:**

lunch\_amount**[**"twos"**]** **-=** **1**

playsound**(**"audio/two.wav"**)**

**else:pass**

socketio**.**emit**(**"update\_amounts"**,** lunch\_amount**)**

Funkce pro vydání oběda odečte žákův oběd od celkového počtu a odešle na stránku k zobrazení.

# Server

## Obecná funkčnost

Server je hlavní zařízení systému. Obsluhuje čtečku karet, komunikuje s databází a kontroluje rozvrhy. Jedná se o program v jazyce Python jehož základem je knihovna Flask. Základní knihovnu už jsem popsal v předešlé kapitole, zaměřím se tedy na interní funkce serveru, které klient nevidí.

## Čtení karet

Čtečka karet komunikuje s PC přes sériový port. Přečte z karty ID a odešle ho se znakem konce řádku. Ke čtení dat z portu používám knihovnu pyserial.

**import** serial

ser **=** serial**.**Serial**()**

ser**.**baudrate **=** **9600**

ser**.**port **=** 'COM4'

Takto vypadá prvotní nastavení. Je potřeba nastavit přenosovou rychlost odesílání dat ze čtečky. Také musíme nadefinovat port ke kterému je čtečka připojena. Ten se liší na různých zařízeních, proto je nutné zkontrolovat jeho správnost.

**def** reader\_loop**(**lunch\_amount**):**

ser**.**open**()**

**while** **1:**

card\_id **=** ser**.**readline**().**strip**()**

card\_id **=** str**(**card\_id**,** 'utf-8'**)**

**print(**card\_id**)**

**if** card\_id**:**

consumer **=** card\_swipe**(**card\_id**)**

**print(**consumer**)**

**if** consumer**:**

check**(**consumer**,** card\_id**,** lunch\_amount**)**

**else:**

**print(**"invalid id"**)**

Z dat přečtených funkcí ser.readline() je nutné ořezat „bílé znaky“ (znak konce řádku, mezery, atd…) pomocí .strip() a převést je na řetězec o stejném kódování, jako jsou uložená v databázi, tedy utf-8.

## Dělení na vlákna

Jelikož funkce ser.readline() vždy na data čeká, čímž by blokovala zbytek programu, je potřeba mít tuto část kódu oddělenou pomocí vláken. Procesová vlákna umožňují provádění několika akcí paralelně. Toho je většinou využito při vykonávání složitých výpočtů, nebo v tomto případě k obsluze externích zařízení.

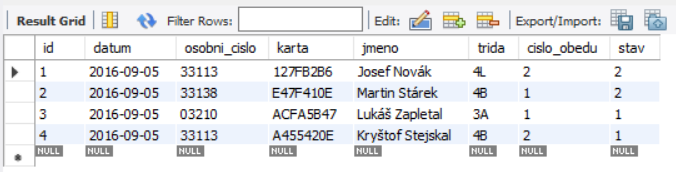
threading**.**\_start\_new\_thread**(**reader\_loop,()**)**

Tímto příkazem se vytvoří vlákno, které bude provádět funkci reader\_loop(). Tato funkce obsahuje nekonečnou smyčku, tudíž vlákno přetrvá až do ukončení programu.

## Komunikace s databází

Školní databáze ke které se připojuji je typu MySQL, proto ke komunikaci s ní používám knihovnu mysql-connector. Tato knihovna umožňuje rozsáhlou komunikaci a práci s databází. V tomto projektu ale využívám jen základní dotazy, jelikož pracuji jen s jednou tabulkou, ze které získám informace o žákově obědu na základě ID jeho karty. Do stejné tabulky také zapisuji jestli si žák oběd vyzvedl.

Pro vývojové účely jsem si vytvořil miniaturní databázi o stejné struktuře pomocí prostředí MySQL Workbench.



Obrázek : Struktura testovací databáze

Databáze obsahuje osobní číslo žáka ve školním systému, ID jeho karty, jméno, třídu, číslo objednaného oběda a stav objednávky (1 – připraveno k odebrání, 2 – odebráno/neobjednáno).

**import** mysql.connector

mydb **=** mysql**.**connector**.**connect**(**host**=**"localhost"**,**

user**=**"root"**,**

passwd**=**"heslo"**,**

db**=**"obedy\_test"**)**

cursor **=** mydb**.**cursor**()**

Po připojení k databázi je nutné vytvořit ukazatel, který pak vykonává dotazy. Ty jsou psány v jazyce SQL. Jeho syntax se většinou píše velkými písmeny. Funkce cursor.execute(dotaz), dotaz odešle a cursor.fetchall() data příjme.

**def** card\_swipe**(**card\_id**):**

cursor**.**execute**(**"SELECT \* FROM OBEDY WHERE karta=%s"**,** **(**card\_id**,))**

result **=** cursor**.**fetchall**()**

**print(**result**)**

**if** result**:**

**return** **(**result**[0][4],** result**[0][5],** result**[0][6],** result**[0][7])**

**else:** **return** False

Funkce card\_swipe je volána po přiložení karty ke čtečce. Jako argument bere už upravené data ze sériového portu. Zde používám dotaz SELECT \* , který z tabulky OBEDY vybere řádek, kde je hodnota ve sloupci karta hodnota rovna argumentu card\_id. Kvůli zabezpečení používám metodu %s, která před dosazením proměnné do dotazu zkontroluje zda se nejedná o SQL injection útok.

Podmínka if result: kontroluje, jestli databáze vrátila použitelnou hodnotu, nebo prázdný seznamn, jehož logická hodnota je nepravda. Pokud je vše v pořádku, funkce card\_swipe() vrátí data z databáze v upravené formě.  
Pokud ale ne, celá funkce vrátí hodnotu nepravda a program nepokračuje v kontrole.

consumer **=** card\_swipe**(**card\_id**)**

**print(**consumer**)**

**if** consumer**:**

check**(**consumer**,** card\_id**,** lunch\_amount**)**

**else:**

**print(**"invalid id"**)**

Podmínka pro kontrolu správnosti dat.

**def** get\_lunch\_amount**():**

cursor**.**execute**(**"SELECT COUNT(\*) FROM OBEDY WHERE cislo\_obedu=1 AND stav=1"**)**

ones **=** cursor**.**fetchall**()**

cursor**.**execute**(**"SELECT COUNT(\*) FROM OBEDY WHERE cislo\_obedu=2 AND stav=1"**)**

twos **=** cursor**.**fetchall**()**

amount **=** **{**"ones"**:**ones**[0][0],** "twos"**:**twos**[0][0]}**

**return** amount

COUNT je další použitý typ dotazu. Tato funkce pouze spočítá celkový počet objednaných obědů. Počet je rozdělen do počtu obědů 1 a obědů 2. Celý počet je pak zabalen do slovníkové struktury vhodné pro odeslání.

## Rozvrhy

### Struktura rozvrhu

Celý rozvrh přestávek je uložen na straně serveru ve slovníkové struktuře. Rozvrh je uspořádán následujícím způsobem:

},

"pa":**{**

**"1":[**"1L"**,**"3A"**,**"3B"**],**

**"2":[**"2A"**,**"2B"**,**"2L"**,**"4A"**],**

**"3":[**"4B"**],**

**"4":[**"1A"**,**"1B"**,**"3L"**,**"4L"**]**

**}**,

"breaks":**{**

**"1":[[11,30],[11,50]],**

**"2":[[12,20],[12,40]],**

**"3":[[13,10],[13,30]],**

**"4":[[13,30],[14,0]]**

**}**

}

Den >> přestávka >> seznam tříd

V posledním úseku je den nahrazen klíčem breaks (přestávky) a vněm jsou uloženy časy přestávek. Časy jsou uloženy jako dvourozměrné pole ve formátu [[hod, min],[hod, min]]. První dvojce označuje začátek a druhá konec přestávky.

Čtení:

**with** open**(**"schedule.json"**,** "r"**)** **as** json\_data**:**

schedule\_table **=** json**.**load**(**json\_data**)**

Zápis:

**with** open**(**"schedule.json"**,** "w"**)** **as** json\_data**:**

json**.**dump**(**data**[**"data"**],** json\_data**)**

K permanentnímu uložení jsem použil soubor typu JSON. Tento způsob je vhodný, protože dokáže ukládat a načítat celé datové struktury ve sejném formátování, jako používá python. K zápisu a čtení souborů JSON je potřeba stejnojmenná knihovna.

### Kontrola rozvrhu

Kontrola probíhá následujícím způsobem:

**print(**"checking schedule"**)**

lunchbreak **=** break\_time**(**current\_time**,** schedule\_table**)**

**if** lunchbreak**:**

**if** consumer**[1]** **in** schedule\_table**[**day**][**lunchbreak**]:**

**print(**"On time"**)**

dispence\_lunch**(**consumer**,** card\_id**,** lunch\_amount**)**

**else:**

**print(**"GET OUT"**)**

playsound**(**"audio/miss.wav"**)**

miss\_schedule **=** True

**else:**

**print(**"Here you go"**)**

dispence\_lunch**(**consumer**,** card\_id**,** lunch\_amount**)**

1. Zkontroluje, zda je v daný čas přestávka.
   1. Pokud ne, vydá oběd bez kontroly času.
2. Kontroluje, jestli je žákova třída obsažena v seznamu tříd pro danou přestávku.
   1. Pokud ano, vydá oběd a odešle data na zobrazovací stránku.
   2. Pokud přišel žák mimo rozvrh sever odešle dotaz pro potvrzení. Jak tento dotaz vypadá popíšu v kapitole Klient.

### Kontrola času

K získání informací o čase používám modul time. základem tohoto modulu je funkce time.time(), která vrací počet sekund od 12:00 1. ledna 1970.

Funkce localtime() pak podle časové zóny přepočítá počet sekund na roky, měsíce, dny, atd. Tuto hodnotu vrací v podobě seznamu následujícího formátu.

(tm\_year=2013, tm\_mon=7, tm\_mday=17,

tm\_hour=21, tm\_min=26, tm\_sec=3,

tm\_wday=2, tm\_yday=198, tm\_isdst=0)

V programu používám pouze den v týdnu (tm\_wday), hodiny (tm\_hour) a minuty (tm\_min).

current\_time **=** localtime**(**time**())**

day **=** what\_day**(**current\_time**.**tm\_wday**)**

current\_time **=**

**(**current\_time**.**tm\_hour**,**current\_time**.**tm\_min**)**

**print(**current\_time**,** day**)**

Takto pak vypadá odečítání času prováděné při každém přiložení karty.

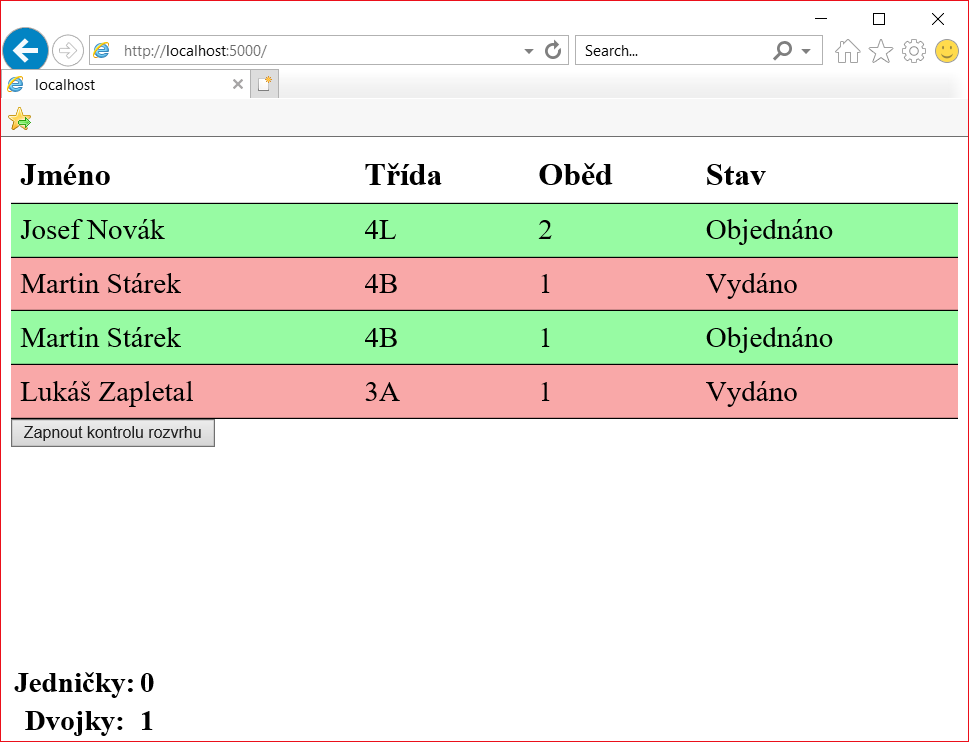
# Klient

## Obecná funkčnost

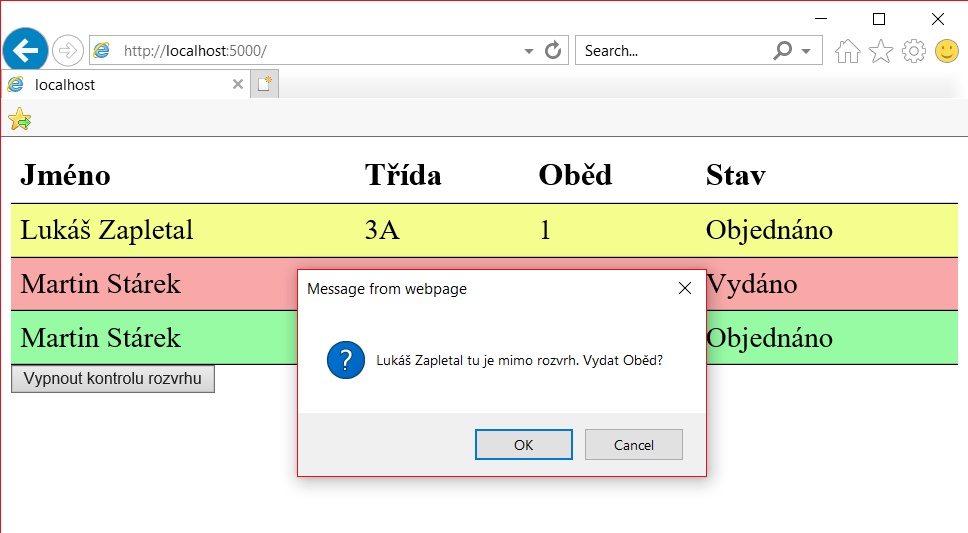
Jak jsem uved v abstraktu, klient je v tomto projektu myšlen jako displej pro zobrazování výdeje obědů. K serveru je možné se připojit z jakéhokoli prohlížeče a zobrazení je přizpůsobeno většině obrazovek. Pro zobrazení je tedy možné využít i vnější zařízení ve stejné síti. Na adrese serveru jsou dostupné 2 stránky. Hlavní zobrazovací stránka a stránka pro změny v rozvrhu.

## Hlavní stránka

Hlavní zobrazovací stránka ukládá poslední 4 strávníky. Zobrazuje jméno, třídu, číslo oběda, a stav objednávky. Na stránce je také zobrazen celkový počet obědů.



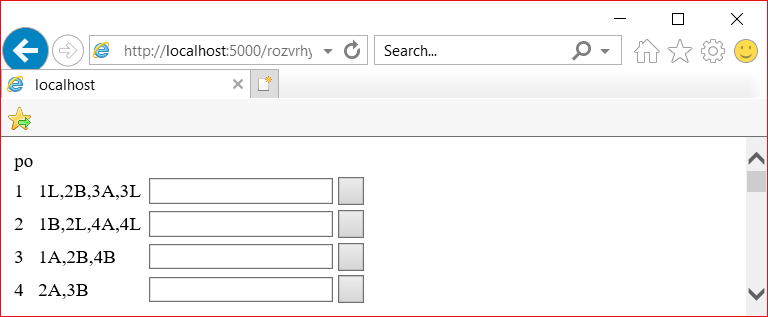
Obrázek : Hlavní stránka bez kontroly rozvrhů

Kvůli nečekaným změnám a příhodnosti použití, je možné kontrolu rozvrhu vypnout/zapnout podle potřeby.

Obrázek : Kontrola rozvrhů v akci

Když je kontrola rozvrhů zapnuta, obsluha může rozhodnout, zda oběd vydat, nebo ne. Toto zajistí možnost porušení rozvrhu pro žáky, kteří například potřebují odejít ze školy dříve.

## Stránka pro změny rozvrhů

Tato stránka zobrazuje aktuální rozvrh a umožňuje přidávání/odebírání tříd ze seznamu.

Obrázek : Změny rozvrhu

Do textového pole vedle vypsaného seznamu tříd je možné napsat jméno kterékoli třídy. Pokud seznam tuto třídu obsahuje, po stisku tlačítka bude třída odebrána. Stejným způsobem je pak třída přidána, pokud v seznamu není. Po provedení změn je nutné rozvrh odeslat na server pro uložení.

# Závěr

První dva body zadání mé práce se týkají softwarové části projektu, s jejichž realizací jsem spokojen. Při tvorbě serverové části jsem se naučil pracovat s většími projekty o více modulech. V části HTML stránek jsem byl nakonec nucen použít jazyk Jacascript, čemuž jsem se chtěl nejprve vyhnout. Nakonec se ale ukázalo, že mé obavy byly zbytečné a k programování webových skriptů se rád vrátím v dalších projektech.

Jelikož jedním z mých cílů bylo vyřešit reálný problém a systém uvést do provozu, nejsem spokojen s nynější formou projektu. Hlavně kvůli nedostatku času, jsem nebyl schopen projektu dát konečnou formu. Momentálně se jedná jen o teoretický model. Pro uvedení do provozu bude ještě potřeba vyřešit nastavení sítě, ve které sever poběží a definitivně určit na jakém hardwaru bude fungovat. To by díky jeho kompatibilitě neměl být problém, ale stále je tato otázka potřeba vyřešit.

Před ukončením mého studia bych rád tento projekt dokončil a uvedl do provozu. Věřím že po odstranění nynějších nedostatků bude přínosem pro fungování školy.

# Seznam použité literatury

[1] Dokumentace ke knihovně flask (28. 3. 2019)

<https://media.readthedocs.org/pdf/flask/latest/flask.pdf>

[2] Dokumentace k flask-socketio (28. 3. 2019 )

<https://media.readthedocs.org/pdf/flask-socketio/latest/flask-socketio.pdf>

[3] Online tutoriál ke knihovně time (28. 3. 2019)

<https://www.tutorialspoint.com/python/python_date_time.htm>

[4] Online tutoriál k jazyku Jvascript (28. 3. 2019)

<https://www.w3schools.com/Js/default.asp>

[5] Online tutoriál ke knihovně mysql-connector (28. 3. 2019)

<https://www.w3schools.com/python/python_mysql_getstarted.asp>

# Seznam obrázků

[Obrázek 1: Zobrazení základního programu 8](file:///C:\Users\Ondra\Documents\GitHub\Project_Lunchmoney\documnentation\DMP_Dokumentace.docx#_Toc4703726)

[Obrázek 2: Struktura testovací databáze 13](file:///C:\Users\Ondra\Documents\GitHub\Project_Lunchmoney\documnentation\DMP_Dokumentace.docx#_Toc4703727)

[Obrázek 3: Hlavní stránka bez kontroly rozvrhů 18](file:///C:\Users\Ondra\Documents\GitHub\Project_Lunchmoney\documnentation\DMP_Dokumentace.docx#_Toc4703728)

[Obrázek 4: Kontrola rozvrhů v akci 19](file:///C:\Users\Ondra\Documents\GitHub\Project_Lunchmoney\documnentation\DMP_Dokumentace.docx#_Toc4703729)

[Obrázek 5: Změny rozvrhu 19](file:///C:\Users\Ondra\Documents\GitHub\Project_Lunchmoney\documnentation\DMP_Dokumentace.docx#_Toc4703730)

Všechny obrázky jsou mého vlastního zdroje.

# Přílohy

Příloha č. 1: Poster k maturitní práci.

