

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DOMENIUL: Calculatoare și tehnologia informației

SPECIALIZAREA: Tehnologia informației

# Auction House Sniper Tool V1

Proiect la disciplinele PSBD & PW

Proiect realizat de:  
Cană Andrei 1408B

# Cuprins

<b>Cuprins</b>	<b>1</b>
<b>Capitolul 1. Scurtă Introducere &amp; Tehnologii Folosite</b>	<b>1</b>
Flask	2
Oracle Database 11.2c	3
cx_Oracle	3
Werkzeug	4
Jinja2	4
Hashlib	5
MySQL Connector	5
OAuth și Requests	6
<b>Capitolul 2. Prezentarea funcțională a aplicației</b>	<b>6</b>
<b>Capitolul 3. Detaliile de implementare ale aplicației</b>	<b>9</b>
Pachetele PL/SQL	10
table_dml_pkg	10
table_management_pkg	11
Scriptul de verificare și inițializare a bazei de date	11
Obținerea datelor despre licitații prin API-ul Blizzard	11
Trimiterea email-urilor de notificare	12
Requesturile dinamice AJAX pentru managementul datelor	12
Exemplu din browse, addToWishList:	12
Endpoint-ul http /addToWishList	13

## Capitolul 1. Scurtă Introducere & Tehnologii Folosite

Acest proiect își propune crearea unei aplicații web cu public țintă jucătorii de World of Warcraft. Funcționalitățile oferite sunt monitorizarea casei de licitații și crearea unor wishlist-uri pentru a putea primi notificări prin email în cazul postării unei licitații sub prețul pieții.

Pentru realizarea acestei aplicații, am folosit limbajul de programare Python 3 și framework-ul web Flask, cu implementare Python. Mai jos voi enumera și explica multele tehnologii și librării folosite.

# Flask

Flask este un web framework Python ce permite construirea de aplicații web. A fost dezvoltat de Armin Ronacher (un programator austriac și un speaker cunoscut la conferințele în domeniul software).

Acest framework este mult mai ușor de înțeles decât framework-ul Django și mult mai ușor de învățat deoarece comunitatea suport este mai mare, iar API-ul este mai intuitiv. Flask, spre deosebire de Django, merge pe un model modular și presupune adăugarea funcționalităților pe măsură ce sunt utilizate.

## Comanda instalare Flask

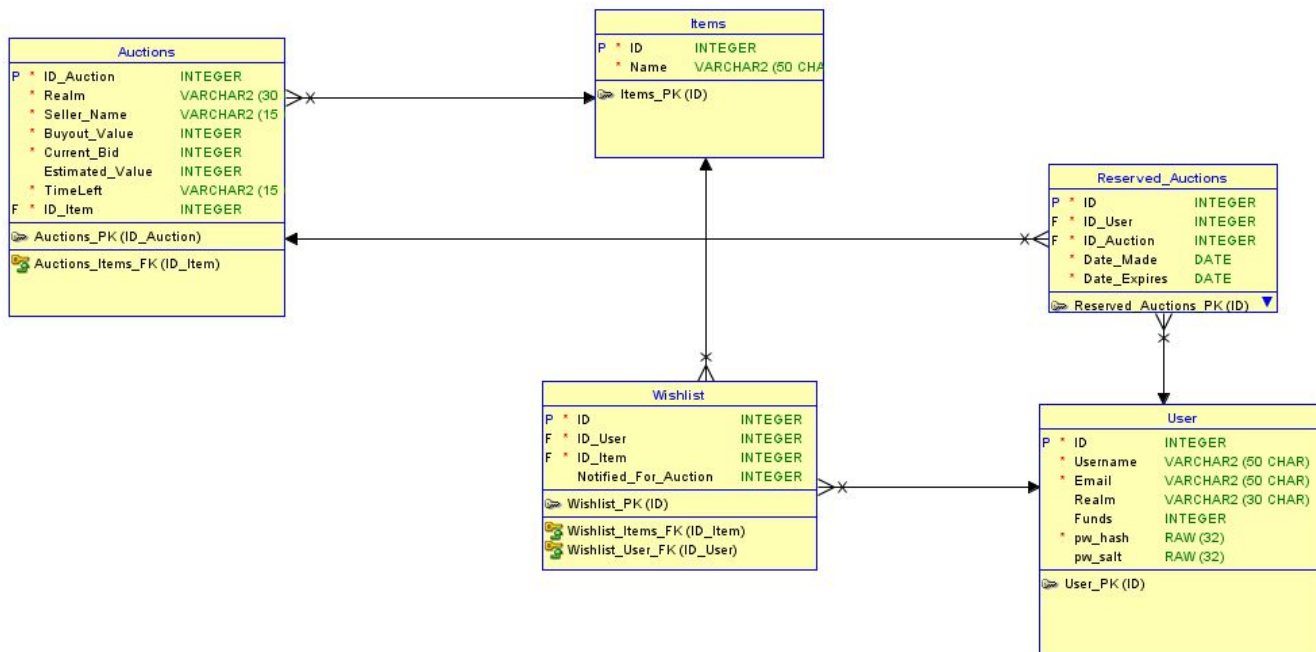
```
pip install Flask
```

## Un exemplu de aplicație Hello World folosind Flask:

```
from flask import Flask
app = Flask(__name__) # Flask constructor
# A decorator used to tells the application
# which URL is associated function
@app.route('/')
def hello():
    return 'HELLO'
if __name__ == '__main__':
    app.run()
```

# Oracle Database 11.2c

Pentru stocarea datelor legate de funcționalitatea aplicației web AHST - V1, am folosit o bază de date relațională Oracle. Tabelele folosite și schema lor sunt următoarele:



## cx\_Oracle

cx\_Oracle este un modul de extensie Python care permite accesul la Oracle Database.

Conformă cu specificației API 2.0 a bazei de date Oracle, cu un număr considerabil de adăugiri și câteva excluderi.

Pentru a utiliza cx\_Oracle 7 cu Python și Oracle Database aveți nevoie de:

- Python 2.7 sau 3.5 și mai mult. Versiunile mai vechi ale cx\_Oracle pot funcționa cu versiuni mai vechi ale Python.
- Bibliotecile client Oracle. Acestea pot fi de la clientul Oracle Instant Client gratuit sau de la cele incluse în Oracle Database dacă Python se află pe aceeași mașină ca și baza de date. Bibliotecile clienților Oracle 19, 18, 12 și 11.2 sunt acceptate pe Linux, Windows și MacOS. Utilizatorii au raportat, de asemenea, succesul cu alte platforme.
- O bază de date Oracle. Standardul de interoperabilitate Oracle standard pentru client-server permite cx\_Oracle să se conecteze la bazele de date mai vechi și mai noi.

Instalare

```
python - m pip install cx_Oracle – upgrade
```

Exemplu utilizare:

## Werkzeug

Werkzeug este o librărie python pentru utilități WSGI (Web Service Gateway Interface) prin intermediul căreia Flask realizează maparea între requesturile http și funcțiile ce se ocupă de soluționarea cererilor. Werkzeug este parte din Flask Core și nu necesită instalare adițională.

## Jinja2

Jinja este o librărie python folosită pentru a genera dinamică a paginilor web, folosind o sintaxă specifică, în baza unor template-uri html.

Folosind funcția `render_template` se poate specifica ce template să fie folosit atunci când se generează răspunsul la cererea web, dar și transmite toți parametrii necesari generării.

Jinja2 dispune de mai multe construcții și funcții pentru a putea realiza asamblarea paginii web.

Se pot declara blocuri într-o pagină de bază folosind sintaxa următoare:

```
{% block NUME_BLOC %} {% endblock %}
```

Aceste blocuri, plasate într-un fișier html de bază, pot conține sau nu elemente html. Prin directiva `extends`, se pot crea pagini care conțin doar informațiile strict relevante unui anumit view.

```
{% extends "layout.html" %}
```

În acest exemplu, documentul html care extinde "layout" va necesita doar folosirea acelorași blocuri declarate în fișierul de bază "layout.html" și adăugarea de conținut.

Pentru a accesa și printa variabilele transmise prin `render_template` se folosește sintaxa

```
{{ nume_variabilă }}
```

Aceasta are ca efect substituirea în html cu valoarea variabilei transmise. Astfel se pot transmite parametri de stil css, parametri către apeluri de funcții javascript și așa mai departe.

Jinja2 suportă și construcții iterative, foarte utile pentru construirea tabelor sau a formularelor.

```
{% for item in items %} ... {% endfor %}
```

Este important de precizat că pentru ca Jinja2 să funcționeze, fișierul html transmis prin apelul `render_template` să fie plasat în directorul "templates", plasat în același folder cu modulul python din care este apelată funcția. În caz contrar, trebuie transmisă calea absolută, generată prin funcția `url_for`.

Astfel, în interior, prin iteratorul `item`, se pot accesa în parte fiecare obiect al colecției `items` - transmisă prin `render_template`. Exemplu de apel `render_template` - pentru generarea unei pagini browse din cadrul aplicației:

```
return render_template(  
    'browse.html',  
    title='Browse',  
    message='Browse to your heart''s content...',  
    year=datetime.now().year,  
    items = results  
)
```

Aici sunt transmise variabilele `year`, `message`, `title` și colecția `items` către Jinja2 - care va genera o pagină web și o va returna ca răspuns http.

## Hashlib

Pentru stocarea parolei utilizatorilor, este considerat un must de securitate ca aceasta să nu fie în format plain-text. Astfel, am ales să folosesc funcția `pbkdf2_hmac` de hash criptografic împreună cu algoritmul SHA-256 pentru a hash-ui parola utilizatorului împreună cu un salt generat aleator și unic pentru fiecare utilizator (generat la crearea contului). În baza de date sunt stocate deci hash-urile parolelor împreună cu salt-urile și salt-urile în sine.

Funcția `doLogin` din `views.py` conține toate elementele ce țin de autentificarea utilizatorului.

## MySQL Connector

Pentru preluarea datelor din baze de date terțe ( [Underminejournal.com](http://Underminejournal.com)) am folosit această librărie python pentru a putea executa query-uri MySQL prin care am obținut date în legătură cu istoricul prețurilor pentru obiectele de la casa de licitație. Vezi capitolul 2 pentru clarificații.

## OAuth și Requests

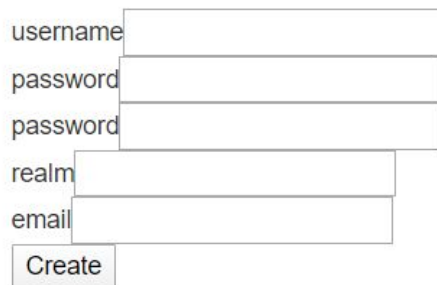
Pentru a putea obține informații de la Blizzard, am folosit Web API-ul lor. (Vezi mai multe detalii la <https://develop.battle.net/> ) folosind OAuth cu flow-ul client credentials pentru a putea obține un token de access cu care am putut apela API-ul REST prin care erau expuse datele din joc în legătură cu licitațiile valabile. Am folosit și Requests pentru a face foarte simplu requesturi HTTP.

## Capitolul 2. Prezentarea funcțională a aplicației

AHST - Auction House Sniper Tool este o aplicație web ce urmărește să ofere jucătorilor de World of Warcraft o unealtă prin care să urmărească online casa de licitații din joc și să primească notificări pentru obiectele de interes plasate sub prețul pieții.

Pentru a realiza aceasta, utilizatorul își crează un cont. Apoi, evident, se loghează.

If you don't have an account, create one!



A registration form with the following fields and a button:

- username
- password
- password
- realm
- email
- 

---

© - AHST - V1

O dată logat, userul este redirecționat spre pagina home, unde găsește explicații și o bară de navigare.

# Auction House Sniper Tool

AHST is a web app that allows you to find amazing deals on WoW items posted at the Auction House.

## Account

Here you can manage your wishlist and reserved auctions as well as updating your account settings.

## Browse

Here you can browse the auctions database and add items to your wishlist or reserve auctions.

## About

Here you'll find information about the developer and service.

© 2020 - AHST - V1

Pentru a putea primi notificări pe email-ul folosit la crearea contului, userul are nevoie de obiecte adăugate în wishlist. El poate adăuga obiecte în wishlist folosindu-se de pagina Browse.

## Browse to your hearts content...

GO TO PAGE (1-1406)

Item Name	Buyout Value	Current Bid	Estimated Value	Discount	Realm	Time Left	Auction ID	Item ID	ACTIONS
Tideguard Maul	865.0g 99.0s 99c	822.0g 69.0s 99c	90.0g 30.0s 64c	-858.96	Silvermoon	SHORT	218467460	159551	<div>Add item to wishlist</div> <div>Add auction to reserved</div>
Dark Animator's Gloves	325.0g 0.0s 0c	292.0g 50.0s 0c	149.0g 75.0s 0c	-117.03	Silvermoon	SHORT	218467469	166827	<div>Add item to wishlist</div> <div>Add auction to reserved</div>



Aici sunt prezentate toate licitațiile valabile pe realm-ul userului - conform ultimului dump generat de Blizzard. Pentru fiecare licitație este afișat obiectul pus la vânzare, prețul cumpărării imediate, prețul la care s-a ajuns cu licitatul și discountul calculat față de prețul mediu de vânzare al obiectului. Datele licitației sunt obținute prin API-ul Community Game Data oferit de Blizzard. Datele legate de prețurile medii ale fiecărui obiect sunt obținute de pe [underminejournal.com](http://underminejournal.com) - un site economic pentru joc, dezvoltat și susținut de jucători.

Folosind butoanele Add Item to wishlist și Add auction to reserved userul își poate exprima interesul spre un anumit obiect sau spre o anumită licitație. Pentru a putea rezerva o licitație, userul trebuie să aibă fondurile necesare în cont. Licitările rezervate nu vor mai fi afișate și altor utilizatori și nu se vor trimite notificări pentru ele.

Panoul Account permite utilizatorului să:

- Vadă și modifice informațiile aferente contului:

# Account Information

Email:

Realm:

Account Funds:

Password Update

Old password

New password

Confirm new password

- Vadă și modifice wishlistul

Item wishlist		
Item Name	Item ID	ACTIONS
Saronite Bar	36913	<button>Remove item from wishlist</button>
Razorwind Axe	55281	<button>Remove item from wishlist</button>
Royal Quartz Loop	153683	<button>Remove item from wishlist</button>
Bands of Fading Light	37590	<button>Remove item from wishlist</button>

- Vadă și modifice rezervările

Reserved Auctions					
Reservation ID	Item Sold	Discount	Auction Timeleft	Reservation Expiry	ACTIONS
218467478	Golem Girdle	57.82	SHORT	2020-01-04 00:00:00	<button>Cancel Reservation</button>
218467559	Wendigo Girdle	21.17	SHORT	2020-01-04 00:00:00	<button>Cancel Reservation</button>
218467612	Shrieking Bow	64.51	SHORT	2020-01-04 00:00:00	<button>Cancel Reservation</button>
218486923	Royal Quartz Loop	-4.93	SHORT	2020-01-04 00:00:00	<button>Cancel Reservation</button>

- Să dea logout

Pagina About oferă date de contact ale dezvoltatorului aplicației (eu, Andrei Cană) și detalii generice despre aplicație.

## Capitolul 3. Detaliile de implementare ale aplicației

## Pachetele PL/SQL

Aplicația folosește Oracle DB Express 11.2g și prin urmare suportă instalarea pachetelor PL/SQL pentru a facilita un control mai fin la nivelul operațiilor DML. Așadar am creat două pachete: `table_dml_pkg` și `table_management_pkg`.

### `table_dml_pkg`

Conține proceduri PL/SQL pentru inserare, updatare și ștergere din cele 5 tabele (`auctions`, `items`, `reserved_auctions`, `useraccounts` și `wishlists`).

Din acest pachet, un exemplu de procedură mai complexă este `UPDATE_USER_PARAM`, care primește ca parametru atât parametrul de updatat cât și noua valoare, face verificări dacă parametrul de updatat este în lista celor admiși și apoi construiește o instrucțiune DML pe care o execută.

```
PROCEDURE UPDATE_USER_PARAM(userID IN NUMBER, paramToUpdate IN  
VARCHAR2, newValue IN VARCHAR2, opRez OUT NUMBER) IS  
    command VARCHAR2(200);  
    BEGIN  
        IF LOWER(paramToUpdate) IN ('username', 'email', 'realm', 'funds') THEN  
            command := 'UPDATE USERACCOUNTS SET ' || LOWER(paramToUpdate) || '=' ||  
LOWER(newValue) || ' WHERE id=' || userID;  
            dbms_output.put_line('Comanda ce se va executa:' || command);  
            EXECUTE IMMEDIATE command;  
            opRez := 0;  
            COMMIT;  
        ELSE  
            opRez := 99;  
        END IF;  
        EXCEPTION WHEN OTHERS THEN  
            opRez := 99;  
            RAISE;  
    END UPDATE_USER_PARAM;
```

Toate aceste proceduri sunt apelate din Python, folosind un cursor obținut prin `cx_Oracle.Connection()`.

Exemplu de apel din `ajax_handlers.py`

```
orclCursor.callproc('table_dml.update_user_param',[userID,param,newVal,opRez])
```

Rezultatul operației este returnat prin intermediul unui parametru OUT și apoi este returnat un răspuns în funcție de el.

## table\_management\_pkg

Conține 3 proceduri fără parametri: delete\_tables, create\_tables și reset\_tables. Aceste proceduri sunt folosite în inițializarea bazei de date printr-un script ce verifică la pornirea serverului web dacă tabelele au fost inițializate și conțin date.

**Pachetele în forma integrală pot fi găsite în folderul AHST/static/sql**

## Scriptul de verificare și inițializare a bazei de date

La pornirea serverului, se pornește un thread separat pentru funcția initOracleDB() din db\_config.py; Această funcție crează două obiecte globale de tip singleton pentru conexiunile către Oracle (PL/SQL) și Underminejournal (MySQL ).

Apoi funcția încearcă un select pe tabelele Oracle relevante pentru a vedea dacă conțin date. Dacă apar excepții, este apelat table\_management.reset\_tables și funcția se re apelează.

## Obținerea datelor despre licitații prin API-ul Blizzard

Procedura de obținerea datelor despre joc este realizată printr-un API securizat cu OAuth. Asta înseamnă că la fiecare apel RESTful al API-ului web pentru AH Data (endpoint: <https://eu.api.blizzard.com/wow/auction/data/>) este necesar un token de access. Acesta se obține prin alt apel http către <https://eu.battle.net/oauth/token> unde sunt date prin header http date despre clientul înregistrat pe ( <https://develop.battle.net/> ), obținându-se un json ce conține token-ul de access - valid 24 de ore.

Folosind tokenul pentru un apel la endpoint-ul /wow/auction/data se obține un json ce conține locația ultimului dump cu licitațiile de pe un anume realm. Acest dump, de regula 20-25MB, este apoi preluat printr-un alt request web și parsat pentru a fi introduse datele în tabelele Oracle folosind table\_dml.insert\_auction.

Discountul pentru fiecare licitație este calculat retroactiv printr-un trigger Oracle ce folosește tabela Items ce conține prețul mediu pentru fiecare item. Datele despre iteme și corelarea ID-Nume este obținută de pe Underminejournal - un website economic pentru comunitatea World of Warcraft.

**Funcția întreagă se află în db\_config.py**

## Trimiterea email-urilor de notificare

Pentru a evita complicarea și mai mare a unui proiect deja complicat - am ales să fac notificarea utilizatorului pentru ofertele găsite pentru obiectele din wishlist la request și nu printr-un cron job.

Pentru trimiterea email-ului am folosit un buton ce face o cerere AJAX către endpointul /sendEmails din ajax\_handlers.py

În cadrul acestei funcții folosesc serverul SMTP Google prin intermediul unui cont care are opțiunea *Allow Less Secure Apps* ON și librăria python *smtplib* pentru a crea un email multipart cu formatare HTML ce conține toate licitațiile cu un discount de peste 50% pentru obiectele din wishlist-ul utilizatorului. Query-ul executat pentru a afla datele relevante din tabelele Oracle este următorul:

```
wIQ = ""SELECT i.name, a.buyout_value, a.discount, a.timeleft FROM WISHLISTS W, ITEMS  
I, AUCTIONS A  
WHERE w.id_item=i.id AND a.id_item=w.id_item AND A.discount>50 AND  
w.notified_for_auction=0 AND w.id_user={}"".format(userID)
```

**Restul funcției se află în ajax\_handlers.py**

## Requesturile dinamice AJAX pentru managementul datelor

Pentru a face mai fluidă experiența utilizatorului, am ales să folosesc requesturi asincrone http folosind obiecte XMLHttpRequest din javascript care apeleau endpoint-uri specifice fiecărei funcționalități.

Exemplu din browse, addToWishList:

```
function addToWishList(itemID) {  
    console.log("S-a apelat functia addToWishList cu id-ul de item: " + itemID);  
    var xmlhttp = new XMLHttpRequest();  
    xmlhttp.onreadystatechange = function () {  
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {  
            console.log("Response:" + xmlhttp.responseText);  
            alert(xmlhttp.responseText);  
        }  
    }  
}  
xmlhttp.open("POST", '/addToWishList', true); // true for asynchronous
```

```

xmlHttp.setRequestHeader("itemID", itemID);
xmlHttp.send(null);
}

```

\*apelul către funcție s-a generat folosind Jinja2 la generarea paginii.

## Endpoint-ul http /addToWishList

```

@app.route('/addToWishList', methods=['POST'])
def addToWishList():
    itemID = request.headers.get('Itemid');
    user = request.cookies.get("loggedUser",None)
    print("User {} wants to add item {} to his wishlist.".format(user,itemID))

    orclCursor = orclConnection.cursor()
    assignedID = orclCursor.var(int)
    try:
        orclCursor.callproc('table_dml.insert_wishlist',[user,itemID,assignedID])
        if assignedID.getvalue() is None:
            print("Itemul {} exista deja in wishlistul lui {}".format(itemID,user))
            return "The item {} already exists in your wishlist.".format(itemID)
        else:
            print("Inserted wishlist entry with id={}".format(assignedID.getvalue()))
            return "Inserted wishlist entry with id={}".format(assignedID.getvalue())
    except:
        e = sys.exc_info()[0]
        print("Database Error: {}".format(e.args))
        return "Exception occured:{}".format(e)
    return "Successfully created wishlist entry with id {}".format(assignedID.getvalue())

```

**Codul integral pentru requesturile JS sunt în fișierele din ASHST/static/myjs/  
Codul integral pentru handling-ul acestor requesturi este în AHST/ajax\_jandlers.py**

**Pentru explorarea code-base-ului, consultați repo-ul github:**

<https://github.com/Aythriel/AH SniperToolV1>