

***Academia de Studii Economice***

***Facultatea de Cibernetică, Statistică și Informatică Economică***

Translator automat

**Besoiu Theodora Gabriela**

**Cimpeanu Ionut Alexandru**

**Dragomir Denis Alexandru**

**Strachinariu Corina Andreea**

# Problema

Se dorește implementarea unei aplicații Web pentru traducerea de texte și gestionarea textelor traduse de către utilizatori. Aplicația trebuie să asigure posibilitatea de autentificare, de traducere a unui text dintr- o limbă în alta și reținerea textelor traduse în funcție de limbă într- un istoric personal al utilizatorului curent.

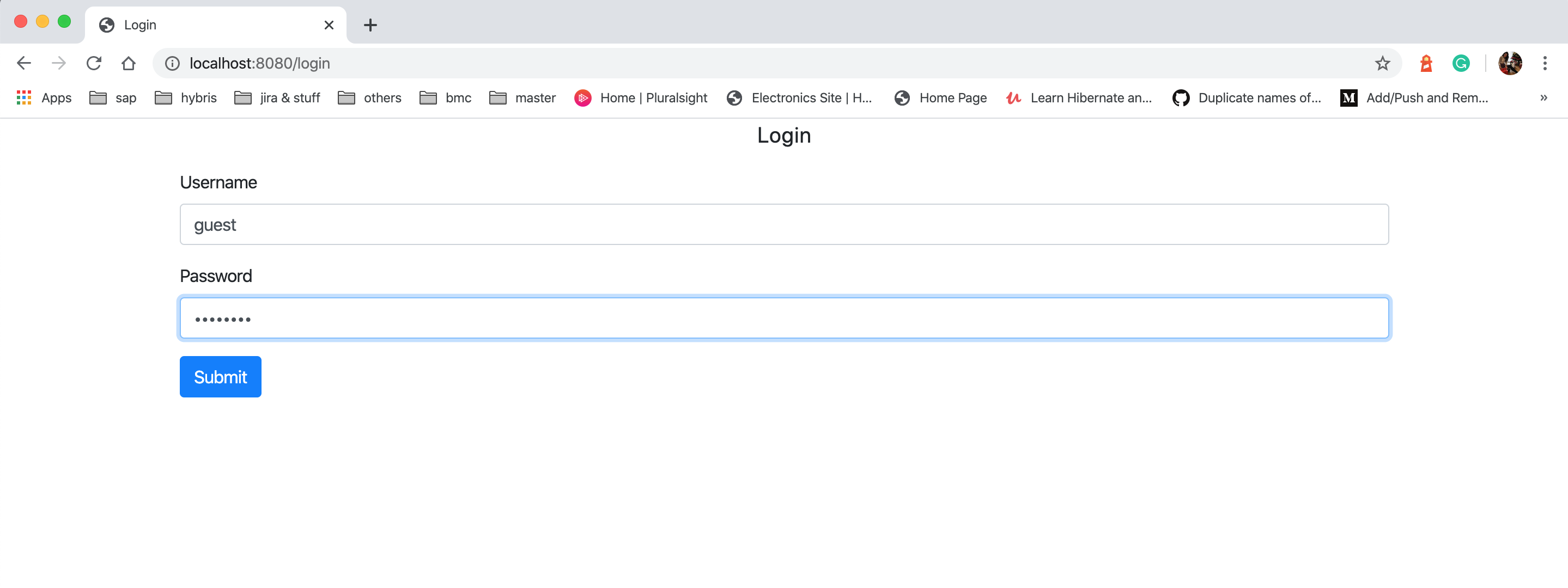
Problema presupune utilizarea de tehnologii moderne pentru crearea sistemului. Se vor utiliza Spring Boot și limbajul de programare Java pentru backend și Spring MVC și Thymeleaf pentru frontend. Pentru autentificare se utilizează Spring Security.

# Solutia

Funcționalitățile sunt prezentate în cadrul unei interfețe grafice intuitive. Pentru accesarea intefeței se utilizează url- ul <http://localhost:8080>.

**Logare**

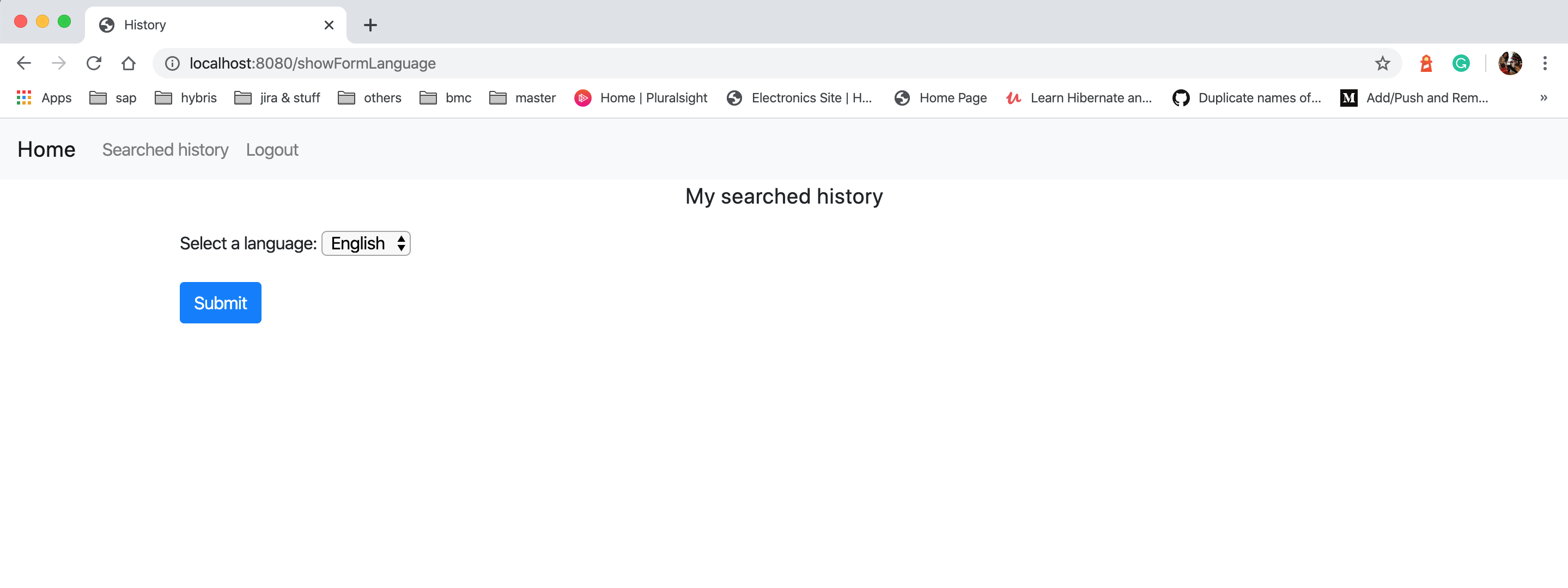
Pentru a putea traduce un text, utilizatorul trebuie să se autentifice. Acesta poate utiliza credențiale userilor deja existenți. Astfel, poate alege să se logheze cu username- ul guest (parola: guest123), admin (parola: admin123) etc.



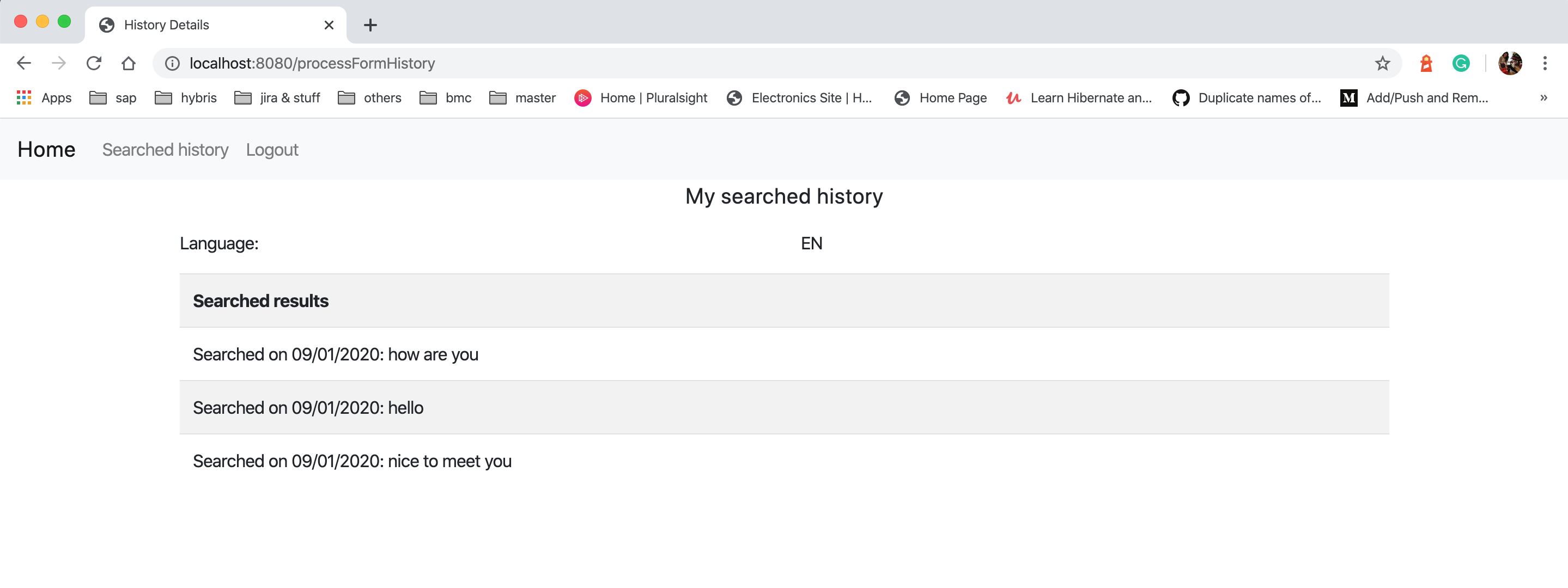
Odată logat, acesta poate selecta între mai multe opțiuni.

**Vizualizare istoric**

Pentru a vizualiza textele ce au fost introduse de către el, poate accesa opțiunea Searched History. Acesta va selecta o limbă și va apasă butonul Submit pentru a vizualiza rezultatele.

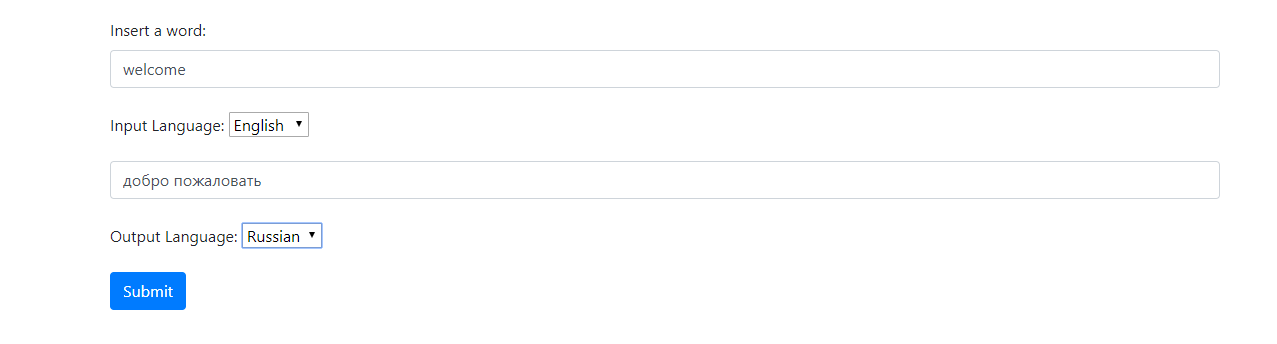


Odată selectată o limbă, se vor afișa textele ce au fost introduse de acesta pentru limba engleză pentru a fi traduse.



**Traducerea**

Pentru a putea traduce, un utilizator trebuie sa introduca un cuvant/ cuvinte, sa selecteze limba din care doreste sa traduca si limba in care doreste traducerea.



# Modele de proiectare utilizare

**Singleton** - În cadrul proiectului s- a utilizat design pattern-ul Singleton pentru gestionarea userilor. Întrucât se dorește accesarea userilor de- alungul aplicației, se utilizează acest model pentru crearea unui singure instanțe ce conține atât lista cu useri existenți, cât și userul curent. Clasa ce implementează acest pattern se numește *UserSingleton* și se găsește în pachetul *ro.ase.translatorApp.userManagement*. În cadrul clasei este instanțiat un obiect static de tipul UserSingleton, ce poate fi accesat prin intermediul metodei getInstance(). Pentru a se împiedica instanțierei unui alt obiect din afara clasei, se definește un constructor de tip private. Obiectul de tip UserSingleton este instanțiat lazy în cadrul metodei getInstance(). Se folosește keyword- ul synchronized pentru împiedicarea creării a mai multor obiecte de către mai multe threaduri diferite. Această clasă este folosită în clasele SecurityConfig și TranslateController, fiind apelată metoda getInstance() pentru a accesarea obiectului.

**Abstract Factory** - În cadrul proiectului este folosit și șablonul *Abstract* *Factory* pentru gestionarea istoricului căutărilor pentru userul logat. Este creată clasa abstractă *SearchedText* ce conțin tipul userului (admin, user), limba pentru care s- a făcut căutarea (En, Fr, De etc) și fraza căutată. Clasa este extinsă de către clasele specifice DeText, EnText, EsText etc. Se dorește crearea de fabrici diferite pentru tipurile de utilizator. Astfel, sunt create două clase *AdminSearchedTextFactory* și *UserSearchedTextFactory*. În cadrul acestor clase se implementează metoda statică *createdSearchedText* ce primește ca input limba și fraza. În funcție de limba căutată, se creează un obiect de tipul limbii respective (exemplu: dacă se caută o frază în limba engleză, se va crea un obiect de tip EnText). Cele două factory- uri sunt accesate de către un factory general. Acesta conține metodă createText, ce conține tipul userului, limba și fraza căutată. În funcție de rolul userului, se apelează metoda createSearchedText din cadrul factory ului specific (dacă tipul userului este admin, se va apela metoda din cadrul AdminSearchedTextFactory). Această fabrică este utilizată în cadrul clasei TranslateController. Se apelează metodă din cadrul clasei *SearchedTextFactory* pentru crearea de obiecte diferite, ce sunt salvate pentru userul curent.

**Cache** – In cadrul proiectului am implementat si patternul de cache pentru a minimiza timpul de intoarcere al cuvantului/cuvintelor traduse. Este creat in clasa Cache si se aplica penru toate limbile utilizate in proiect. Se doreste ca cele mai cautate cuvinte sa fie retinute separat pentru a nu mai se face apel catre API-ul de translate, astfel vom obtine o imbunatatire a raspunsului serverului catre client. CacheTranslate este un obiect ce contine cuvintele cautate precum si limbile din care/ in care se vrea traducerea. Aceasta clasa este folosita de TranslatorManager. In cazul in care cuvintele dorite a fi traduse nu se gasesc in cache, se va apela API-ul, se va face traducerea, se va salva in cache si apoi se vor returna in interfata cuvintele traduse.

**Builder** – In cadrul proiectului am implementat si builder-ul pentru a permite utilizatorului sa trimita request-uri personalizate, in functie de preferintele acestuia. In cadrul clasei denumita Request a fost creata clasa RequestBuilder. Atunci cand se trimite un request, parametrii necesari sunt reprezentati de *requestId*, *requestKey*, *requestInput*, *requestFirstLanguage* si *requestSecodLanguage*. Parametrii optionali definesc formatul textului tradus (*requestInputFormat*) si specifica daca se doreste detectarea automata a limbii de intrare (*requestOptions*). Valorile posibile pentru primul parametru optional sunt: plain (text in format implicit) si html (text in format html), iar pentru al doilea sunt true (detectarea automata a limbii) si false.

**Null Object Design Pattern** - În cadrul proiectului este folosit și șablonul *Null Object* pentru gestionarea istoricului căutărilor pentru userul logat in cazul in care se doreste traducerea cu o limba care nu este prezenta in enum-ul Limbilor aplicatiei, dar se doreste salvarea in istoric. Acest lucru s-a realizat prin crearea unei clase abstacte: *List*  care are 2 metode (limba- returneaza limba selectata sau null in cazul in care nu exista aceasta si isNull- returneaza true daca nu exista valoarea si false daca exista). Clasa *NonNullList*  reprezinta obiectul care are ca atribute limba selectata din enum si valoare false a functiei isNull. Aceasta extinde clasa abstracta *List.* Clasa *NullList* reprezinta situatia in care elementul nu face parte din enumeratia din aplicatie, caz in care este „EN”, iar valoarea isNull este true. Aceste clasa sunt utilizate in clasa *Language* pentru a returna obiectul cu o valoare default sau limba selectata.