UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

RELAZIONE

Incapsulare la complessità per offrire uno strumento semplice: rendere interattivo l'insegnamento della lingua inglese nella scuola primaria utilizzando NLTK

Candidato: *Chiara Zamberti*

Relatore: *Prof. Augusto Ciuffoletti*

Correlatore: *Prof. Alessandro Lenci*

Anno Accademico 2015-2016

Indice

Introduzione

Il compito dell'elaborato è quello di illustrare uno strumento destinato alle scuole elementari per l'apprendimento dell'inglese come seconda lingua (*L2*). Nei tempi recenti si sente sempre più la necessità di rendere interattive le lezioni sin dalla scuola primaria, sia per una questione di semplificazione dell'insegnamento, che per la verifica integrale delle competenze acquisite. Per i nativi digitali, infatti, è più intuitivo testare ciò che si impara con strumenti all'avanguardia e, meglio ancora, se con l'ausilio di attività ludiche. Il bambino viene così messo di fronte a delle *challenge* dove può provare ciò che ha appreso tramite delle semplici interazioni, senza la necessità di carta, penna e altri strumenti tradizionali, facilitando così anche le correzioni da parte del docente. Al giorno d'oggi quasi tutte le scuole sono dotate di connessione a Internet e di laboratori multimediali, ma il passaggio definitivo alla totale digitalizzazione delle lezioni non è ancora avvenuto e la strada per farlo è abbastanza tortuosa. Il principale problema è spesso dovuto alla mancanza di fondi per l'acquisto e la gestione delle attrezzature multimediali e alla scarsità di personale per la loro manutenzione.[1]

Come si legge in un articolo di *OrizzonteScuola.it*[2], nelle scuole elementari italiane vi è a disposizione appena "un pc ogni 15 studenti" e non di rado esistono problemi relativi alla connessione di rete. In riferimento a quest'ultima, è emerso che quando essa è disponibile, non è sempre a banda larga o comunque non riesce a coprire tutti i dispositivi presenti negli istituti. Inoltre, non raramente vi sono aule computer organizzate parzialmente con macchine datate e prive di accesso a Internet e solo alcune di esse sono aggiornate e collegate alla rete.

Tale situazione è stata riscontrata in una scuola del *Sud Pontino*, la "*Scuola Primaria De Amicis IC Pollione*" di Formia (LT)[3] che, pur essendo stata di recente portata all'avanguardia, continua a presentare lacune simili.

1 *Attrezzature informatiche si usurano e rompono, mancano figure specifiche nelle scuole.* 17 novembre 2016. Redazione di OrizzonteScuola.it. Link all'articolo: http://www.orizzontescuola.it/attrezzature-informatiche-si-usurano-e-rompono-mancano-figure-specifiche-nelle-scuole/.

2 *I numeri della scuola digitale. Ci vogliono più risorse, appena un PC ogni 15 studenti elementari.* 20 gennaio 2014. Redazione di OrizzonteScuola.it. Link all'articolo: http://www.orizzontescuola.it/numeri-della-scuola-digitale-ci-vogliono-pi-risorse-appena-pc-ogni-15-studenti-elementari/.

Ciò si è appreso durante un colloquio informale con la docente ed esperta in inglese Cinzia di Marco, maestra elementare presso la scuola, che ha espresso opinioni sulla carenza negli istituti italiani di supporti digitali e materiale didattico da esservi destinato e ha marcato la necessità di "informatizzare" le lezioni per renderle più accattivanti.

Si è poi passati a discutere della fornitura di apparati multimediali in relazione all'Istituto; la scuola, proprio in seguito al piano di interventi del *PON*[4] ha ricevuto grazie ai *Fondi Strutturali Europei* il materiale necessario per l'informatizzazione: rete *Wireless*, *LIM* per numerose classi, aule multimediali con computer all'avanguardia e tablet per l'utilizzo da parte di docenti e alunni. Nonostante le operazioni di modernizzazione degli ambienti per il miglioramento delle attività di laboratorio è però emerso, come già accennato, che non tutti i supporti hanno a disposizione la connessione a Internet, necessaria per lo svolgimento di gran parte degli esercizi interattivi.

Come segnalato dalla docente, sono disponibili numerosi *software* fruibili offline, ma solo compatibili con determinati sistemi operativi (*S.O.*), spesso *Microsoft Windows* o *Mac OS*. Questo non è di solito un grosso problema, poiché la maggior parte dei computer o supporti sono equipaggiati con una versione di Windows ma, di recente, con la diffusione di sistemi operativi dedicati ai bambini, si stanno sensibilizzando le scuole a introdurre gli alunni all'utilizzo di S.O. liberi, come *Ubuntu* o altre distribuzioni *Linux*[5]. Per tali sistemi dunque, resterebbero a disposizione i tanti *tools* online, che però non sono accessibili se non connessi.

Un altro problema rilevante è emerso essere quello della carenza di tecnici di laboratorio con conseguente scarsità di manutenzione dei supporti multimediali: non è raro infatti restare con un apparecchio non allestito appropriatamente, non funzionante o che presenta problemi anche per molto tempo.

---

3 *Scuola Primaria "De Amicis IC Pollione"*, Formia (LT). Sito web della scuola: http://www.icpollione.it/.

4 *Programma Operativo Nazionale*: *Per la Scuola, competenze e ambienti per l'apprendimento*. Link al sito: http://www.istruzione.it/pon/.

5 *Computer per bambini: migliori sistemi divertenti, educativi e sicuri.* 8 febbraio 2017. Redazione di Navigaweb.it. Link al sito: http://www.navigaweb.net/2012/07/computer-per-bambini-migliori-sistemi.html.

Inoltre –secondo la maestra– spesso sono proprio i bambini che, messi di fronte a un esercizio interattivo, tendono a distrarsi e a "giocare" con il dispositivo invece che svolgere l'attività, compromettendone talvolta –innocentemente– il funzionamento.
Proprio grazie a questa preziosa testimonianza, è nata l'idea di realizzare uno strumento per introdurre l'interattività in classe, utilizzando i mezzi digitali messi a disposizione dalla scuola, senza particolari richieste di macchine con *hardware* recente o connessione a Internet assicurata.
Inizialmente, si era pensato di creare un software da usare durante le ore di italiano per aiutare i bambini ad approcciarsi all'apprendimento delle parole e all'analisi grammaticale; successivamente però si è preso in considerazione di sviluppare uno strumento da utilizzare durante l'insegnamento dell'inglese. Tale scelta è stata effettuata poiché, sebbene per l'italiano ci siano a disposizione dei tool di analisi automatica in sviluppo, essi non sono ancora molto precisi e attendibili. Si è dunque scelto di integrare nello strumento didattico un analizzatore automatico più affidabile e progettato in particolare per l'inglese, *NLTK*, del quale si tratterà a seguire.
Pertanto, è subito venuto a galla un ulteriore problema: la difficoltà di far funzionare tale supporto didattico in un ambiente scolastico con poco personale tecnico esperto.
Si è dunque giunti a una soluzione: quella di incapsulare in una macchina virtuale tutto ciò che l'utente finale "non deve né vedere, né toccare" e di rendere usufruibile lo strumento dal browser tramite pochissime interazioni.
Essendo l'inglese, oltretutto, materia di competenza della docente e potendo avere a disposizione del materiale da visionare grazie alla stessa, si è pensato che l'idea di creare un qualcosa da utilizzare durante le ore di inglese fosse una buona soluzione.
Per avere un'infarinatura su cosa si svolgesse in classe durante le lezioni, sono state fornite dalla maestra le informazioni riguardanti il livello della lingua insegnata nelle classi (dalla prima alla quinta) e i titoli dei testi da consultare per capire gli esercizi tipo svolti durante le lezioni.

Lo scopo dell'applicazione

Lo scopo dell'applicazione è la verifica delle competenze di grammatica inglese di livello *A1* tramite un semplice gioco di abbinamento di parole alle parti del discorso corrispondenti; lo strumento è anche dotato di un'attività più semplice sulla pronuncia e sulla comprensione di termini, la quale richiede la partecipazione attiva del docente.

La parte di analisi è gestita da NLTK (*Natural Language ToolKit*), un insieme di strumenti per il trattamento del linguaggio naturale che utilizza *Python* per la creazione dei programmi.

Per permettere allo strumento didattico di essere indipendente dal sistema operativo sul quale si utilizza e non intaccabile in alcun modo, è stato strutturato in due parti: una costituita da un web server virtualizzato e l'altra da un'interfaccia accessibile da un comune browser in *localhost*[6].

Nei capitoli più avanti verranno spiegate nel dettaglio le varie parti e funzionalità dell'applicazione.

6 Il browser, puntando all'indirizzo *localhost*, chiederà al sistema di risolvere l'hostname "localhost", che di solito corrisponde a quello dell'interfaccia di loopback http://127.0.0.1/. Per "loopback" si intende ciò che viene utilizzato in una rete TCP/IP per identificare una macchina in essa. La porta sulla quale vengono effettuate tali operazioni è la 5000, ovvero quella predefinita per la trasmissione sotto il protocollo di rete TCP.

1. L'interattività nella didattica

L'impiego di strumenti interattivi nelle lezioni è ormai altamente consigliato, in particolare nell'istruzione primaria, dove i bambini provengono da una realtà prevalentemente ludica quale è la scuola materna.

Per poter avere una prova di quanto accennato in precedenza a proposito della fornitura in termini di apparati multimediali nelle scuole e del relativo impiego, in particolare quelle elementari, è stata consultata la maestra di inglese Cinzia di Marco della "Scuola Primaria De Amicis IC Pollione". La docente, durante uno dei colloqui tenuti, ha affermato che l'Istituto ha solo di recente introdotto l'utilizzo della tecnologia nell'apprendimento e spera che nei prossimi anni questa possa agevolare lo svolgimento delle lezioni. "La lezione frontale tradizionale è da preferire" – afferma – "ma bisogna adattarsi a nuove tendenze ed esigenze".

Inoltre, sostiene che l'impiego di giochi, quiz o canzoni come prova di ciò che si è imparato è utile anche in caso vi siano alunni portatori di disabilità: in questo modo, nella maggior parte dei casi, non sono costretti a seguire attività richiedenti obiettivi minimi. Il gioco, infatti, unisce.

Discutendo con la docente sulla disponibilità di attrezzature informatiche nella scuola e sull'assenza di un software indipendente da Internet per la verifica delle competenze, compatibile anche con Linux (dato che alcuni PC della scuola sono equipaggiati con Ubuntu) e non necessitante grande manutenzione o requisiti di installazione, è nata l'idea del progetto oggetto di tale relazione.

Quest'ultimo si propone come reale strumento da consultare nelle ore di laboratorio di inglese nella suddetta e in altre scuole, sebbene la prova *in situ* verrà organizzata solo in futuro.

1.1 Il materiale consultato

Il primo passo verso l'ideazione e la successiva realizzazione del progetto è stata la presa visione dei testi adottati per le classi seguite dalla docente, per capire il livello di competenza raggiunto dagli alunni.

Dopo un rapido esame, è emerso che:

- Nelle classi prime e seconde gli esercizi del testo[7] vertono soprattutto sul vocabolario di base, ristretto all'ambito dei saluti, degli oggetti di uso comune, dei giocattoli e dei numeri.

  Sono inoltre presenti molti esercizi di abbinamento (ad esempio, del colore alla relativa rappresentazione), le didascalie sono in inglese ma molto semplici e si limitano quasi esclusivamente a espressioni di base di uso quotidiano;
- Dalle classi terze in poi[8], vengono proposte invece attività di completamento di frasi elementari e il vocabolario viene esteso con ulteriori espressioni e termini, riguardanti, a esempio, le parti del corpo, il cibo, la casa e il tempo.

  A differenza dei libri per le prime classi, in questo caso iniziano a esserci dei riferimenti alla grammatica inglese, con tipologie di esercizi come quelli di completamento (*gapped text*) e di riordinamento.

Gli obiettivi stabiliti dalla collana di testi *Let's be friends* sono quelli di sviluppare ed esercitare la conoscenza della lingua inglese e favorire, grazie a ciò, l'avvicinamento dei bambini a culture e realtà diverse e l'educazione al rispetto reciproco[9] (tali finalità sono sommariamente le stesse di *Top Secret*, secondo la docente).

Dopo un'attenta analisi dei testi (in particolare, dei tre volumi di *Top Secret*) e un confronto ulteriore con la maestra, è emerso che gli esercizi più di successo tra i bambini sono quelli che hanno uno sfondo ludico. Nel libro, e altrettanto nel CD-ROM allegato, sono presenti rivisitazioni delle attività che riprendono quelle in formato cartaceo; sono risultati essere molto interessanti degli esercizi interattivi di inserimento di parole nella relativa frase.

È stato notato che le attività proposte sono incentrate principalmente sul vocabolario e sui verbi ma carenti sull'aspetto dell'analisi grammaticale, fondamentale per i bambini dalla terza elementare in su per una buona conoscenza di base della lingua.

7 Foster, Frances e Brunel Brown. 2016. *Let's be Friends*, con espansione online. Per la Scuola Elementare, volumi 1 e 2. Londra, Pearson.

8 Foster, Frances e Brunel Brown. 2014. *Top secret*, con fascicolo, e-book, ed espansione online. Per la Scuola Elementare, con CD-ROM, volumi 3, 4 e 5. Londra, Pearson.

9 Carter, Johanna, Giulia Achilli, Brunel Brown. 2016. *Guida per i genitori a supporto di Let's Be friends,* Londra, Pearson.

Proprio tale mancanza è servita da ulteriore spinta per la creazione della parte principale dello strumento oggetto dell'elaborato, che verte sull'abbinamento di parole alla relativa parte del discorso di una frase a scelta dell'alunno o del docente.

Come è venuto alla luce da un ulteriore confronto con l'insegnante e come si evince da quanto proposto nei libri di testo, tale attività trova il suo migliore impiego a partire dalle classi terze in poi, poiché prima i bambini non hanno ancora le competenze grammaticali richieste ma solo un piccolo bagaglio di vocaboli ed espressioni.
A colmare dunque la lacuna venutasi a creare, ovvero la mancanza di attività nello strumento interattivo per le classi prime e seconde, è stata aggiunta una sezione per l'ascolto e per la comprensione di parole casuali appartenenti a sette categorie (casa, cibo, numeri e colori, vestiti, parti del corpo, scuola e città).
Sebbene non tutti i vocaboli siano insegnati nelle classi prime e seconde, l'attività ha anche lo scopo di scoprirne di nuovi prima di essere insegnati successivamente.
Tali vocaboli sono stati estratti manualmente dal software *Inglese, Lessico di base. Impariamo la parola* di Rossana Cannavacciuolo[10], un eseguibile per Microsoft Windows contenente circa 250 parole di base del lessico inglese, suddivise nelle sette categorie dette, conosciuto e utilizzato in passato dalla docente.

Mentre l'attività di abbinamento della parola alla parte del discorso corrispondente è effettuabile da soli o anche a casa per una verifica, la seconda prevede invece un continuo *feedback* da parte dell'insegnante, che deve invitare l'alunno o il gruppo classe a pronunciare la parola dopo l'applicazione e a giudicarne l'esito.

1.2 Approccio al problema

Il principale obiettivo dello strumento è quello di andare a colmare la lacuna creata da molti software solo disponibili online o in formati univoci per quanto riguarda il sistema operativo (es.: eseguibili *.exe* per Microsoft Windows o immagini *.dmg* per

10 *Impariamo la parola*: *Inglese, Lessico di base* di Rossana Cannavacciuolo. Versione 1.1. Link al sito: http://rossanaweb.altervista.org/blog/il-mio-software/inglese-e-francese-lessico-di-base/.

Mac OS), dunque non compatibili con ogni tipo di macchina. Inoltre, si pone lo scopo di essere un'applicazione installabile in pochi passi, non invasiva, distribuibile liberamente e, soprattutto, non necessitante di grandi manutenzioni da parte di personale addetto.

Infatti, non in tutte le scuole –da quanto è emerso– è presente un personale prettamente idoneo alla gestione dei supporti multimediali e questo causa discontinuità nel funzionamento di apparati che richiedono manutenzione.

Parte dei problemi alle macchine non è dovuta esclusivamente all'avaria causata dal frequente utilizzo delle postazioni o all'invecchiamento dell'hardware, ma anche da manomissioni, volontarie o meno, dei piccoli utilizzatori.

Inoltre, non è semplice rendere ogni macchina pronta ad accogliere qualsiasi nuova installazione: c'è bisogno infatti di librerie di sistema, *frameworks* o altri tipi di *dipendenze*.

Si supponga di dover dotare del software una serie di macchine, in un'aula computer, ad esempio. Per permetterne il funzionamento, dovrebbero essere prima di tutto dotate dei requisiti necessari e ciò implicherebbe molto tempo, si creerebbero incompatibilità con software preesistenti e, nel peggiore dei casi, il sistema operativo potrebbe essere compromesso.

Una buona soluzione che permette di evitare troppi interventi tecnici e rende la macchina pronta a far funzionare il software in pochi minuti è quella della virtualizzazione.

Virtualizzare, in gergo tecnico, significa astrarre le caratteristiche hardware della macchina, in modo tale da poterle utilizzare come risorse virtuali per il software[11]. Ciò offre la possibilità di installare un "sistema operativo sul sistema operativo", in modo da lavorare all'interno di questo come se si fosse dentro una *sandbox*, ovvero un ambiente a sé stante, dove qualsiasi operazione effettuata è totalmente indipendente dal sistema operativo ospitante (chiamato *host*, per l'appunto). In caso di problemi all'interno del sistema ospite (*guest*), basterà importare di nuovo la macchina virtuale, ma l'host resterà in ogni caso illeso.

11 Kusnetzky. 2011. *Virtualization: A Manager's Guide.* In: Chapter 1, A Model of Virtualization, What Is Virtualization?, Massachusetts, O'REILLY Media, p. 1.

Per rendere possibile quanto detto, le postazioni (preferibilmente computer, con caratteristiche hardware non troppo datate e la virtualizzazione abilitata) dovranno essere dotate solo di un programma per l'esecuzione della macchina virtuale e non di tutte le dipendenze sopra elencate. Successivamente, il tecnico di laboratorio o semplicemente il docente dovrà importare il sistema da virtualizzare in tale programma e avviarlo; la risorsa sarà poi disponibile localmente in un browser web a scelta.

Il sistema ospite da importare sarà in formato applicazione virtuale, vale a dire una sorta di "pacchetto" con tutto il necessario creato in precedenza.

Nel caso in analisi, come software per la virtualizzazione è stato scelto *Oracle VM VirtualBox*[12] poiché con due grossi punti a favore: l' essere *multipiattaforma* e *open-source,* oltre che gratuito. Per la fruizione sarà solo necessario, dunque, installare VirtualBox su ogni macchina e, successivamente, importare l'applicazione virtuale e avviarla; in questo modo si avrà a disposizione lo strumento pronto all'utilizzo da parte degli alunni, nel browser. Di tale argomento, si approfondirà nel prossimo capitolo.

1.3 Sommario dell'applicazione

Come già accennato in precedenza, lo strumento ha l'obiettivo di essere poco invasivo per la macchina e adoperabile da chiunque; per rendere ciò possibile è stato necessario inserire il tutto in un ambiente "chiuso" a parte, soltanto consultabile dal browser.

La struttura del software è costituita da un web server creato con *Flask* in Python, ospitato su una macchina virtuale guest in VirtualBox con sistema operativo *Ubuntu Linux Minimal*. Nel web server è presente NLTK, la libreria di Python che gestisce l'analisi del linguaggio e permette così di trattare il testo inserito in input. Tramite VirtualBox è effettuata un'operazione di inoltro della porta 5000 *TCP/HTTP* per permettere all'utente di accedere allo strumento tramite un browser sulla macchina fisica, all'indirizzo http://localhost:5000/ (vedere figura 1).

12 *VirtualBox*, sito ufficiale del progetto. Link: https://www.virtualbox.org/.

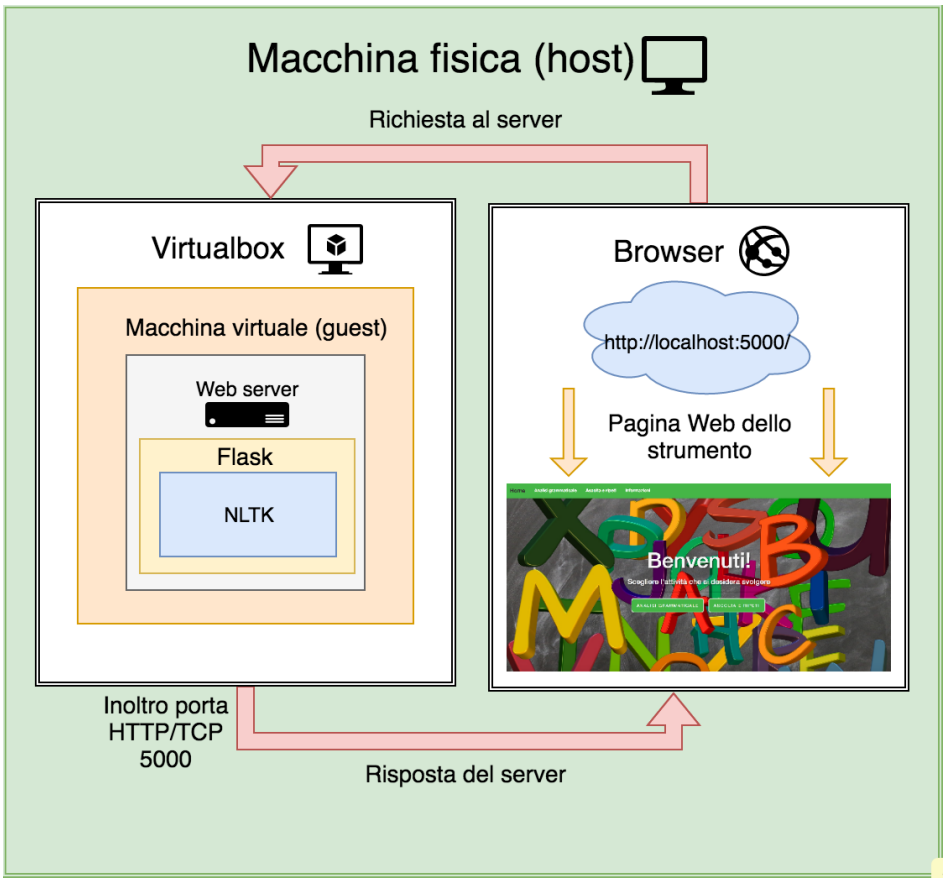

Figura 1. Schema del funzionamento dell'applicazione

Stabilendo in questo modo la comunicazione tra le due macchine (host e guest), è possibile usufruire delle funzioni offerte dall'applicazione semplicemente avviando la macchina e attendendo il suo caricamento.

A tal punto, ci si troverà di fronte a una pagina principale con due bottoni per ciascuna delle attività proposte. Cliccando sul primo (*Analisi grammaticale*), verrà caricata un'interfaccia dove l'alunno sarà invitato a scrivere una frase e a completare con successo un gioco di abbinamento di parole alle relative parti del discorso. Selezionando la seconda voce (*Genera, ascolta e ripeti!*), invece, si sarà condotti a uno strumento utile soprattutto per le prime e le secondi classi, che prevede la generazione di una parola casuale da una determinata categoria di vocaboli e l'ascolto della sua pronuncia in inglese.

Della parte tecnica relativa all'applicazione, verranno stilati i dettagli nei prossimi capitoli.

2. VirtualBox come supporto di virtualizzazione

VirtualBox (formalmente, Oracle VM VirtualBox) è un software per l'esecuzione di macchine virtuali (*VM*, *Virtual Machine*) atte a ospitare sistemi operativi (*OS*, *Operating System*) avviabili su altrettanti dischi virtuali. Questo programma (alla versione 5.1.18) offre la possibilità di ospitare qualsiasi OS supportato: basta seguire il *wizard*. Esso guida nella definizione di una nuova VM, operazione che include la creazione di un disco fisso virtuale (con estensione *.vdi*, a grandezza fissa o dinamica), che va a costituire l'allocazione per il sistema e conduce all'importazione dell'immagine disco del sistema operativo che si vuole installare o avviare in modalità live. Permette inoltre di impostare il quantitativo di memoria *RAM* da destinare alla VM; tuttavia per un controllo più preciso delle risorse hardware si dovrà accedere alle impostazioni tramite l'icona dedicata dopo il termine della procedura guidata. Da qui sarà possibile determinare, oltre alla quantità di RAM (nel caso in questione, impostata a 512MB essendo la macchina senza interfaccia grafica), altri settaggi relativi a ulteriori risorse, come CPU, memoria video, accelerazione hardware e abilitazione a supporti di *input* aggiuntivi.

Sebbene creare e configurare una macchina virtuale sia un'operazione relativamente semplice, toglie senz'altro tempo e può non essere alla portata di tutti, specie se devono essere installati requisiti specifici.
Per una questione di versatilità è dunque stato pensato di offrire il tutto già pronto, importabile con un semplice click.
Al fine di rendere ciò possibile, essa viene fornita come applicazione virtuale con estensione *OVA* (*Open Virtual Appliance*), un archivio in formato *tar* creato tramite VirtualBox, con all'interno il disco virtuale con tutto il necessario già installato[13].

13 *Document Number DSP0243*. 2010. In: Section 5, Open Virtualization Format Specification. Link al documento: http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.1.0.pdf.

2.1 Fruizione dello strumento

L'unico *step* che dovrà essere compiuto per rendere la postazione operativa con lo strumento sarà aprire la OVA in VirtualBox; dopo tale mansione tutto sarà pronto per la fruizione.

La OVA potrà essere importata tramite il menù *File* > *Importa applicazione virtuale...* (o con doppio click sull'icona del file stesso); sarà a tal punto disponibile in VirtualBox una nuova voce con il nome della VM, chiamata *Interactive_English*.

Tale macchina con tutto il suo contenuto costituisce il cosiddetto *server*, ovvero il sistema che fornisce i dati richiesti da altri elaboratori –nel caso in esame, il computer sul quale risiede e il rispettivo browser– che vengono usati per accedere alla risorsa, ovvero l'interfaccia con la quale devono interagire gli studenti.

Come accennato, il software ha il vantaggio di poter essere utilizzato anche in assenza di rete e, per tale ragione, qualsiasi connessione tra il client (la macchina host) e il server (la macchina virtuale con il *web server* del programma avviato, di cui si parlerà più avanti) viene effettuata in locale.

Per accedere al contenuto è consigliabile avviare la macchina in modalità *headless*, cioè in modalità silenziosa per evitare l'ingombro della finestra di VirtualBox che può essere erroneamente chiusa. Inoltre, non sarà necessario effettuare il login, in quanto lo script *Interactive_English* per l'avvio del web server (del quale si tratterà successivamente) è inserito nella lista di processi di avvio durante la fase di *boot* (in *init.d*).

Dopo l'avvio, bisognerà indirizzarsi tramite un browser all'URL **http://localhost:5000**/, dove la dicitura "localhost" indica *l'IP* 127.0.0.1 dell'interfaccia di *loopback*[14] e 5000 la porta impostata per la connessione HTTP/TCP[15]. Tali impostazioni di *port forwarding* sono applicate via VirtualBox alla creazione della VM prima dell'esportazione in formato OVA, tramite la voce "Inoltro delle porte", nidificata in "Impostazioni", sotto la scheda "Rete" (figura 2).

14 *What is a localhost?* Link: http://whatismyipaddress.com/localhost.

15 *Lista porte standard di TCP/UDP.* Link al sito: http://www.tcp-udp-ports.com/ports5000-5100.html.

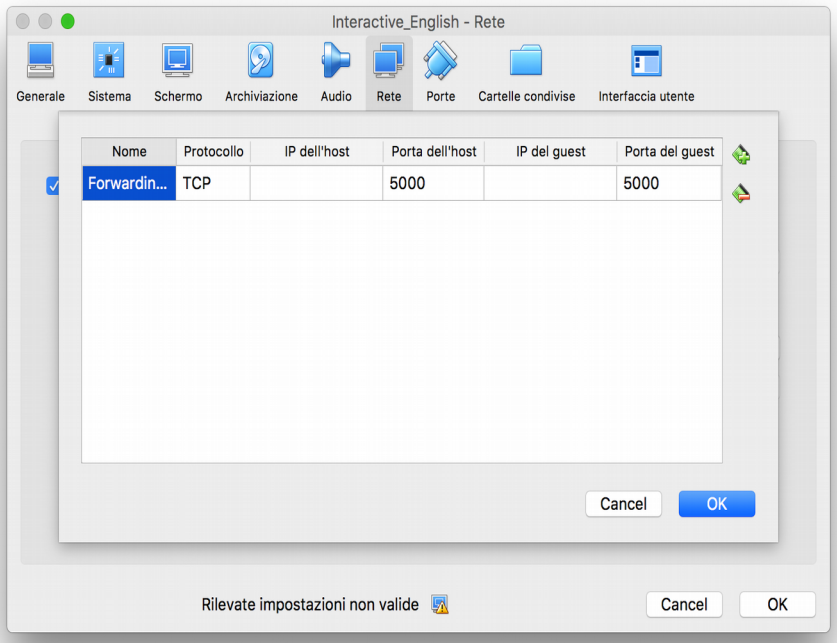

Figura 2. Inoltro delle porte in VirtualBox

2.2 Preparazione della macchina virtuale

Come accennato precedentemente, lo strumento didattico è distribuito in formato OVA per una questione di semplicità di fruizione. Verrà ora mostrato come è stata preparata la macchina a partire dall'installazione del sistema operativo.

Dopo aver selezionato, da wizard o manualmente, l'immagine di Ubuntu come supporto di avvio per la macchina virtuale appena creata, è possibile fare il primo *boot* di essa. Al primo avvio viene presentata la schermata con le varie voci di boot e si potrà procedere all'installazione del sistema tramite una procedura guidata.

Una volta impostati i parametri relativi alla lingua, alla tastiera, all'*hostname*, e al profilo utente (nel caso in questione si è scelto come nome *int_en* e come password "**quattro volte spazio**"), è richiesto su quale supporto installare l'immagine. Bisogna a questo punto selezionare l'unico disco a disposizione, ovvero quello virtuale creato in precedenza, formattarlo e proseguire. Una volta caricate tutte le componenti necessarie, inizia l'installazione effettiva. È inoltre chiesto se devono

essere inclusi dei pacchetti agguntivi a quelli di base: in questo momento non è necessario selezionare alcuna voce. La procedura guidata conduce al termine dell'installazione, la quale si concluderà con un riavvio. [16]

Successivamente, per accedere al sistema saranno richieste le credenziali precedentemente definite; una volta effettuato il login dovranno essere installate le dipendenze necessarie, ovvero la serie di pacchetti indispensabili per il funzionamento del server in Python.

2.3 Il sistema operativo della VM: Ubuntu

Come sistema operativo per la macchina virtuale è stato scelto Ubuntu[17], una distribuzione Linux nota per la semplicità di utilizzo e per il grande team di supporto in rete. Si è preferito basarsi sulla versione 16.04 *Xenial Xerus* poiché, sebbene non fosse l'ultima disponibile, è una *LTS* (*Long Term Support*), ovvero con aggiornamenti garantiti fino a cinque anni dalla data di rilascio (in questo caso, fino ad aprile 2021).

Dato che il sistema operativo della macchina guest ha il semplice compito di fornire un ambiente di incapsulamento per il web server in Python, è stato scelto di non installare la versione completa di Ubuntu, ma la *CD Minimal,* ovvero un'immagine con solo il sistema di base con i pacchetti essenziali.

Quest'ultima, molto piccola –di 48 MB, contro i circa 1.5 GB della versione completa– e priva di interfaccia grafica e di modalità live, è l'ideale per essere utilizzata in macchina virtuale poiché poco esosa di risorse in termini di memoria fisica (disco vdi) e virtuale (RAM)[18]. Per una ragione di compatibilità con PC non dotati di sistema di virtualizzazione hardware o comunque abbastanza datati è stata inoltre scelta la versione a *32 bit* anziché quella a *64 bit*.

16 Per la guida completa, consultare: Istituto Majorana, *Guida all'Installazione Minimale di Ubuntu*, versione 1.0, pp. 5-18. La guida è basata su versioni precedenti di Ubuntu Minimal ma è valida anche per le successive.

17 *Ubuntu*, link al sito ufficiale: http://www.ubuntu-it.org/.

18 "Il CD minimale è avviabile anche su macchine con RAM inferiore ai 700 MB. Risulta inoltre essere un'alternativa al CD Alternate, disponibile solo per Lubuntu dalla versione 12.10." Link: http://wiki.ubuntu-it.org/Installazione/CdMinimale.

2.4 Installazione dei requisiti per il web server

Dopo aver installato il sistema operativo nella VM, è stato necessario configurarlo con il necessario per il funzionamento dello strumento realizzato.

Le operazioni di installazione sono state effettuate tramite il comando `sudo apt-get install nome_pacchetto`, utilizzando quindi lo strumento *APT* (*Advanced Packaging Too*l) di Ubuntu.

I pacchetti installati via `apt-get` sono:

- *git* (*Git*, il noto software di controllo versione)[19];
- *python-pip* (*pip*, un semplice sistema di gestione dei pacchetti in Python)[20];
- *python-numpy* (*NumPy*, una serie di librerie per la gestione avanzata di operazioni numeriche in Python)[21];
- *python-nltk* (Natural Language ToolKit, una collezione di strumenti per l'elaborazione automatica del testo)[22].

Dopo tale operazione, è stato scaricato il necessario per l'elaborazione testuale con NLTK, tramite l'*installer* interno di tale libreria, come segue.

NLTK, che più avanti verrà trattato nel dettaglio, costituisce la serie di tool necessari per il trattamento automatico del testo e delle parti del discorso nei due esercizi interattivi proposti nell'applicazione. Una volta installato tramite apt, è stato necessario scaricare i moduli di NLTK richiesti dal web server per il corretto funzionamento.

In questo caso, è stato sufficiente effettuare il download di:

- *punkt* (il modulo per la gestione della tokenizzazione)[23];
- *averaged_perceptron_tagger* (un *POS tagger* per l'annotazione morfo-sintattica delle varie parti del discorso)[24].

---

19 *Git*, sistema di controllo versione. Link al sito ufficiale: https://git-scm.com/.

20 Per la documentazione completa, consultare: *Pip documentation*, Release 10.0.0.dev0, March 24, 2017. Link al documento: https://media.readthedocs.org/pdf/pip/latest/pip.pdf.

21 *NumPy Manual*. 2017. User Guide, Setting Up, Installing Numpy. Link al sito: https://docs.scipy.org/doc/numpy/index.html.

22 Bird, Steven, Ewan Klein ed Edward Loper. 2009. *Natural Language Processing with Python.* In: Preface, Massachusetts, O'REILLY Media.

23 Bird, Steven, Ewan Klein ed Edward Loper. *Op. Cit.*. In: Chapter 3, Processing Raw Text. Section 3.8, Segmentation, p. 112.

24 Bird, Steven, Ewan Klein ed Edward Loper. *Op. Cit.*. In: Chapter 5, Categorizing and Tagging Words. Section 5.1, Using a Tagger, p. 179.

Per l'aggiunta di questi ultimi, è bastato aprire una shell Python (tramite il comando "`python`"), digitare "`import nltk`" per importare NLTK e, successivamente, accedere al pannello che permette di scaricare i pacchetti necessari per l'analisi computazionale con "`nltk.download()`". Dopo tale operazione, sono state mostrate a video una serie di scelte disponibili: è stata selezionata quella contrassegnata dalla lettera "`d`", ovvero "download" e inserito il nome del pacchetto da scaricare. Tale operazione è stata ripetuta sia per il modulo punkt che per average_perceptron_tagger, per soddisfare con successo i requisiti del web server che andrà poi inserito nella macchina e mandato in esecuzione.
In seguito, è stata configurata e installata tramite pip la serie di pacchetti per il framework Flask, con il quale è strutturato il web server (del quale si parlerà in seguito).
Per preparare Ubuntu al funzionamento dell'applicazione costruita con Flask è stato necessario installare, tramite l'ausilio di pip in una *shell* Python, con il comando `sudo pip install nome_pacchetto`, quanto segue:

- *flask* (i files del framework in Python)[25];
- *flask_bootstrap* (l'estensione per gestire gli elementi del framework front-end *Bootstrap,* del quale si tratterà)[26];
- *flask_nav* (una libreria per la *creazione* in modo semplice di menù di navigazione nei templates HTML)[27].

Soddisfatti tali requisiti, il server web è perfettamente integrato nella macchina. A tal punto, è stato sufficiente clonare la cartella con l'occorrente nella directory *home* del sistema tramite il comando:

```
git clone https://bitbucket.org/c_zamberti/Interactive_English/
```

Al termine di questa serie di procedure, per provare ad avviare il server è stato necessario entrare nella cartella appena scaricata –con il comando di *UNIX*

25 Grinberg, Miguel, 2014. *Flask Web Development.* In: Chapter 1, Installation. Installing Python Packages with pip, Massachussets, O'REILLY Media, p. 6.
26 Grinberg, Miguel. *Op. Cit.*. In: Chapter 3, Templates, Twitter Bootstrap Integration with Flask-Bootstrap, pp. 26-28.
27 *Flask Nav*, link al repository: https://github.com/mbr/flask-nav.

`cd/Interactive_English`– e digitare nel terminale `python interactiveEnglish.py`.

2.5 Il web server nella macchina

All'interno della VM, dopo il termine delle operazioni sopra elencate, è disponibile l'applicazione *interactiveEnglish.py*, un web server in Python costruito con l'impiego del framework Flask –del quale si tratterà più avanti– che costituisce il vero motore del tutto. È essa, infatti, a "lavorare" all'indirizzo localhost:5000 e proprio grazie alle impostazioni di port forwarding di cui si è parlato, è accessibile da host tramite browser.

Per una questione di semplicità (per evitare dunque che l'utente finale debba avviare il server in Python nella VM), si è provveduto ad automatizzare la procedura di avvio.

Per rendere ciò possibile, è stato inserito nel sistema guest uno script (codice in appendice) per la *shell UNIX*[28], *Interactive_English*, in grado di mandare in esecuzione a ogni avvio dello stesso *interactiveEnglish.py*[29].

Lo script contiene una serie di condizioni standard per l'avvio e per il controllo dell'esecuzione di un determinato comando; è stato però adattato per puntare al server in Python e per specificare per quale utente lanciarlo (in questo caso, per l'unico utente del sistema, *int_en*). Le parti in questione sono quelle con etichetta *dir, cmd e user*. Al fine di renderlo eseguibile a ogni avvio della VM, tale script è stato inserito nella cartella */etc/init.d*, insieme agli altri a esecuzione automatica. Inoltre, è stato necessario modificare i permessi per avviare lo script correttamente in fase di boot, con il comando da terminale `chmod 755 Interactive_English`. In questo caso, "755" è un numero composto da tre *cifre ottali* che contiene le regole per consentire:

- al proprietario (*root*) di leggere, modificare ed eseguire lo script;

28 Baldan, Paolo. 2004-2005. *Introduzione a Unix.* Lucidi per il corso di Laboratorio di Sistemi Operativi, Università Ca' Foscari di Venezia.

29 *Init-script*-template. Link al repository: https://github.com/fhd/init-script-template/blob/master/template.

- al gruppo del quale fa parte l'utente (*int_en*) e ad altri di leggere ed eseguire lo script. [30].

30 Cannon, Jason. 2014. *Linux Succinctly*. Morrisville, Syncfusion Technology Resource Portal, pp. 35-46.

3. NLTK per l'analisi del linguaggio naturale

Il fulcro dello strumento didattico è NLTK, una libreria open source per il linguaggio Python creata da Steven Bird, Edward Loper ed Ewan Klein, utile nello sviluppo di software e nell'educazione. È corredata di una guida introduttiva al suo impiego nella linguistica computazionale con Python, che la rende utilizzabile anche dai non esperti nel settore. La libreria include più di 50 *corpora* (collezioni di testi scritti o orali traslitterati) e risorse lessicali come *Penn Treebank Corpus*, *Open Multilingual Wordnet*, *Problem Report Corpus* e *Lin's Dependency Thesaurus*.[31]

Tecnicamente parlando, NLTK offre un metodo valido per il trattamento del linguaggio naturale (*Natural Language Processing,* abbreviato in *NLP*). Per "linguaggio naturale" si intende ciò che è usato dagli umani per la comunicazione; ne fanno parte per esempio l'inglese, l'hindi e il portoghese. Al contrario dei linguaggi artificiali come quelli di programmazione, i linguaggi naturali si sono evoluti di generazione in generazione e per tale motivo sono difficili da individuare secondo regole specifiche. Le tecnologie basate sul NLP si stanno diffondendo su larga scala: trovano impiego ad esempio nella predizione di parole durante la scrittura e nelle traduzioni automatiche. Esse stanno trovando particolare impiego in campo scientifico, economico e culturale.[32]

NLTK è utile a linguisti e programmatori per raccogliere dati statistici su testi singoli o in collezione e per creare strumenti che, tramite l'analisi e il riconoscimento automatico di dati linguistici, possono essere utilizzati in vari campi, come quello dell'educazione. Le opportunità che questa offre sono innumerevoli, ma in questo caso l'attenzione verterà sull'annotazione morfo-sintattica, sul cosiddetto *Part of Speech Tagging* (*POS Tagging*), utilizzato nel gioco in esame nell'attività di abbinamenti della parola alla parte del discorso.[33]

---

31 *Techopedia explains Natural Language Toolkit (NLTK)*.
Link alla pagina : https://www.techopedia.com/definition/30343/natural-language-toolkit-nltk.

32 Bird, Steven, Ewan Klein ed Edward Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*, O'REILLY Media, pp. IX-X.

33 Dell'attività sull'ascolto dei vocaboli appartenenti alle sei categorie dette si tratterà più avanti, in quanto essa non prevede l'impiego di particolari modelli di NLTK ma solo del tokenizzatore e di un analizzatore di testo piano.

Tabella 1. Esempio di tagset (usato nel Penn Treebank Project)

Numero	Tag	Descrizione
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

3.1 L'utilizzo del tagger

Un POS tagger elabora una sequenza di parole e vi associa l'etichetta in base alla parte del discorso che coprono (*POS*, *part of speech*). Vi sono annotatori per diverse lingue, incluso il cinese, l'hindi e il catalano, ma in questo caso ci si basa su quello dell'inglese standard.

Le POS sono contenute in un *tagset*, ovvero una raccolta di parti del discorso identificate da una sigla (vedere tabella 1)[34]. Al momento dell'elaborazione, lo strumento individua la parola (o espressione) nel corpo del testo e ne associa l'etichetta in base al contesto. Tale procedura, grazie alle svariate ricerche condotte, è diventata molto precisa, ma purtroppo ha un margine di errore.

Da un punto di vista più specifico, il POS tagger è utilizzabile in Python importandolo dalla libreria di NLTK e facendolo operare su una determinata stringa. Sarà necessario dividerla però prima nelle singole entità, tramite il processo di *tokenizzazione,* ovvero spezzarla in unità minime di analisi, dette *token*. Un token può essere una parola delimitata da due spazi, ma non è sempre così: vi sono infatti molti casi particolari ed espressioni più lunghe che possono costituirne uno solo.[35]

Dopo il processo di divisione in token, potrà essere avviata l'annotazione.

Di seguito è illustrato un frammento di codice nel quale viene effettuata la serie di operazioni appena dette:

```
import nltk
text = nltk.word_tokenize("And now for something completely
different")
nltk.pos_tag(text)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'),
('completely', 'RB'), ('different', 'JJ')] [36]
```

Come si può notare, la frase nella variabile `text` è divisa con il tokenizzatore `word_tokenize` e successivamente vi è applicato l'annotatore sintattico `pos_tag`.

34 *Alphabetical list of part-of-speech tags used in the Penn Treebank Project*.
Link alla fonte: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

35 Per maggiori informazioni, consultare il testo completo.

36 Bird, Steven, Ewan Klein ed Edward Loper. *Op. Cit.*. In: Chapter 5, Categorizing and tagging words, Using a tagger, Section 5.1, pp. 179-181.

In output viene restituita una struttura dati contenente le tuple (coppie) parola-etichetta della parte del discorso: "`and`" è etichettato come "`CC`" (*coordinating conjunction*, congiunzione coordinante), "`now`" e "`completely`" sono "`RB`" (*adverb*, avverbio), "`for`" è "`IN`" (*preposition*, preposizione), "`something`" è "`NN`" (*noun*, nome) e "`different`" è "`JJ`" (*adjective*, aggettivo).

È proprio da tale struttura dati che, come si vedrà in seguito, verranno estratte le parti del discorso nell'esercizio interattivo sull'analisi grammaticale.

Nelle operazioni di etichettatura è usato *Averaged Perceptron*, un tagger con un algoritmo ad alta affidabilità. Non essendo effettuata alcuna operazione preventiva di *training* (di "allenamento" su determinate collezioni di testi, con espressioni non standard), il tagger funzionerà con le impostazioni di riconoscimento di default.

3.2 Margine di errore ed esempio degli omonimi

Sebbene l'annotazione morfo-sintattica effettuata tramite l'impiego di tool automatici sia piuttosto precisa, non lo è del tutto. Ogni parte della frase è classificata, come detto, a seconda della posizione e del contesto di essa, ma ciò non basta per avere una totale accuratezza. Può succedere ad esempio che un termine venga erroneamente identificato come sostantivo anziché come verbo o come nome plurale invece che singolare, specie se si analizzano testi prettamente settoriali.

Nonostante ciò, lo strumento è incredibilmente efficace se si prendono in input piccole porzioni di testo, come frasi non molto complesse. In tal caso, la gestione di parole che cambiano significato a seconda del contesto è a dir poco ottima. Un esempio è quello degli omonimi (vedere figura 4). Essi, grazie al contesto, vengono riconosciuti ed etichettati correttamente.

Di seguito è riportata una porzione di codice che illustra il trattamento di due parole identiche tra loro ma con semantica differente[37]:

```
import nltk
text = nltk.word_tokenize("They refuse to permit us to obtain the
refuse permit"
nltk.pos_tag(text)
```

37 Bird, Steven, Ewan Klein ed Edward Loper. *Op. Cit.*. In: Chapter 5, Categorizing and tagging words, Using a tagger, Section 5.1, p. 180.

```
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'),
('us', 'PRP'), ('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'),
('refuse', 'NN'), ('permit', 'NN')]
```

Come è possibile notare, in un caso NLTK identifica "`refuse`" come verbo "rifiutare" e nell'altro come "rifiuto". La correttezza dell'etichettatura è favorita dall'analisi della forza associativa tra le parole; ciò si ottiene analizzando le collocazioni, ovvero una sequenza di parole che di solito occorrono insieme. [38]

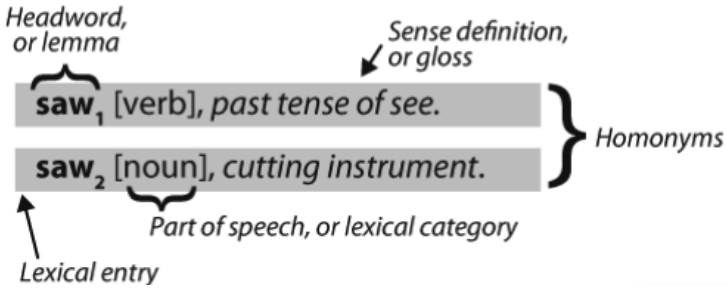

Figura 3. Omonimi con funzioni grammaticali diverse

38 Bird, Steven, Klein, Loper. *Op. Cit.*, In: Chapter 1, *Language Processing and Python*, Computing with Language: Simple Statistics, section 1.3, p. 19.

4. I linguaggi di programmazione utilizzati

Per la creazione dello strumento sono stati utilizzati diversi linguaggi, ma il ruolo centrale l'ha svolto *Python*.

Oltre a quest'ultimo, sono stati impiegati anche linguaggi di *templating* (*Jinja2*), di *markup* e formattazione (*HTML* e *CSS*), di *scripting* (*JavaScript*) ed è stato utilizzato il framework in Python Flask, che consta di alcune regole proprie. In seguito si spiegheranno nel dettaglio questi ultimi e verrà illustrata la loro utilità.

4.1 Python

Python –impiegato nella realizzazione della parte server dello strumento– è un linguaggio *multiparadigma* scelto per leggibilità del codice e la relativa semplicità di impiego per la produzione di software di qualità.[39] La versione sulla quale ci si è basati è la 2.7, per avere una buona compatibilità con la libreria NLTK; tuttavia il codice è stato testato con successo anche eseguendolo in shell di Python 3.x.

Si è scelto di utilizzare questo linguaggio poiché incluso nel sistema operativo della VM nel quale è incapsulato il server e poiché indispensabile per utilizzare la libreria NLTK e il framework Flask, scritti in Python.

In generale, questo linguaggio è da preferire ad altri non solo per la leggibilità, ma anche perché offre compattezza nella scrittura del codice: un programma in Python è solitamente da un terzo a un quinto più piccolo di uno equivalente in *C++* o *Java*. Ciò sta a significare scrivere meno e, di conseguenza, fare meno errori. Inoltre, non serve compilare i programmi prodotti: possono essere eseguiti immediatamente, velocizzando così il lavoro dello sviluppatore.

Esso è multipiattaforma, ovvero utilizzabile indipendentemente dal sistema operativo sul quale risiede: è utilizzabile dunque per creare software eseguibile su un grandissimo numero di OS. Con l'installazione di base, Python consta già di innumerevoli funzionalità, offerte dalla cosiddetta *libreria standard,* la quale supporta una serie di operazioni di programmazione che vanno dal trattamento di porzioni di testo alla creazione di script di rete. Inoltre, può essere esteso con

39 Lutz, Mark. 2013. *Learning Python*, 5th Edition. In: Preface, Massachusetts, O'REILLY Media.

applicazioni di terze parti, come strumenti per lo sviluppo web, per operazioni numeriche, per l'accesso a porte seriali, per lo sviluppo di videogiochi e tanto altro. È anche ampiamente integrabile con altri linguaggi di programmazione, come *C* e *C++*. [40]

Grazie, infine, alla relativa semplicità di utilizzo e alla grandissima rigidità della struttura (è obbligatorio rispettare la sintassi e l'indentazione per il corretto funzionamento del programma), è consigliato anche a studenti o sviluppatori alle prime armi. [41]

4.1.1 Flask

Flask è un piccolo framework, così minuto da poter essere chiamato *micro-framework* e da essere compreso, dopo un po' di pratica, anche da chi non ha dimestichezza con strumenti simili. Sebbene sia molto leggero (occupa meno di 10 MB di spazio su disco), ciò non toglie che sia molto potente, alla pari di altri framework più "grossi" (ad esempio, *django*[42]). Flask, infatti, è stato progettato per essere estensibile. Al momento dell'installazione (effettuabile con pip, come per la maggior parte delle estensioni) il framework offre un *core* con i servizi di base: i *plugin* possono essere scaricati poi all'occorrenza come singoli pacchetti.

Di seguito sono riportate le funzioni di Flask essenziali per la comprensione del funzionamento dell'applicazione presa oggetto della relazione.

Esso ha due dipendenze principali: il sottosistema *Web Server Gateway Interface* (*WSGI*) che è offerto da *Werkzeug* e Jinja2, per la gestione dei template.[43]

Flask va a gestire sul server l'interazione con il lato client, di solito costituito da un browser web. In poche parole, il client invia delle richieste al web server, che in risposta le spedisce all'applicazione Flask, dove vengono elaborate. Per una questione di ordine su cosa eseguire a una determinata richiesta, il framework si serve di *routes*, associazioni tra URL e la rispettiva funzione da eseguire.

---

40 Lutz, *Op. cit.*. In: Chapter 1, A Python Q&A Session: Why do people use Python?, p. 3.
41 Per maggiori informazioni sul linguaggio e sulla relativa sintassi, consultare il libro citato.
42 *Django*, link al progetto: https://www.djangoproject.com/
43 Grinberg, Miguel. 2014. *Flask Web Development*. In: Chapter 1, Installation, Installing Python Packages with pip, Massachussets, O'REILLY Media, p. 4.

Una *route* è definita dal *decorator* (una *feature* standard in grado di modificare il contenuto di una funzione) `app.route`, come nell'esempio che segue:

```
@app.route('/') def index():
      return '<h1>Hello World!</h1>'
```

In questo caso, la funzione `index` è associata alla radice dell'URL: se l'applicazione è ad esempio lanciata su un server associato al dominio www.example.com, viene eseguita la funzione `index()`. Il valore restituito dal client (browser), detto *response*, è ciò che viene mostrato all'utente. Le funzioni come quella nell'esempio sono dette *view* e possono contenere da una semplice stringa HTML a strutture molto complesse. È possibile inserire in un decorator anche un URL dinamico, mettendo la parte variabile tra parentesi angolari, come segue:

```
@app.route('/user/<name>')
def user(name):
      return '<h1>Hello, %s!</h1>' % name
```

Quando la funzione viene invocata, Flask invia il componente variabile come argomento dell'URL. Nelle route dinamiche è possibile inserire anche valori come `int`, `float` e `path` (ad esempio: `/user/<int:id>`, per fare il matching solo con URL che hanno un numero intero nell'identificatore). Pertanto, non si tratterà di route dinamiche in questo capitolo, poiché non sono utilizzate nel prodotto finale.
Per mandare in esecuzione il framework con il *debugging* attivo, è necessario avere al termine del programma:

```
if __name__ == '__main__':
      app.run(debug=True)
```

In qualsiasi app Flask deve essere creata un'*istanza* dell'applicazione. Questa sarà necessaria al web server, tramite il protocollo WSGI, per inviare tutte le richieste che riceve dal client a essa per la gestione.

L'istanza dell'applicazione è un oggetto della classe Flask e è dichiarato come segue:

```
from flask import Flask
app = Flask(__name__)
```

Il solo argomento richiesto al *costruttore* della classe `Flask` è il nome del modulo o pacchetto dell'applicazione. Normalmente, la variabile `__name__` costituisce il valore corretto; Flask usa l'argomento `name` per determinare il percorso della root dell'applicazione, in modo da trovare i files relativi a essa.

Per ottenere una piccola ma completa applicazione, è sufficiente scrivere quanto segue in un qualsiasi editor di testo per l'elaborazione di codice:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
      def index():
      return '<h1>Hello World!</h1>'
if __name__ == '__main__':
app.run(debug=True)
```

e in seguito salvare come *hello.py*.

Una volta creata, l'applicazione potrà essere mandata in esecuzione da terminale [44] con il comando `python hello.py` (vedere figura 4). Si aprirà un processo di Python e saranno mostrate le righe relative allo stato e al debugging.[45]

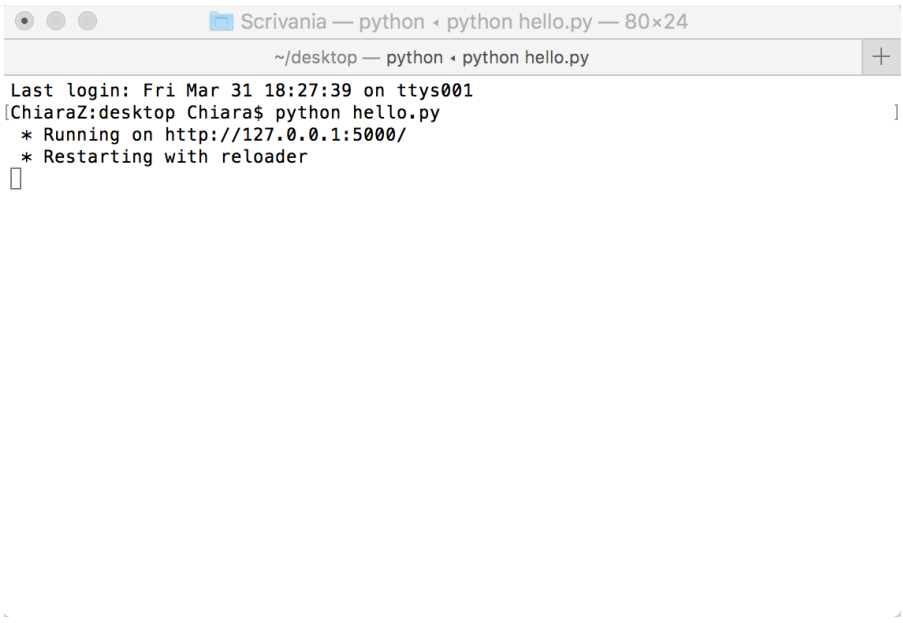

Figura 4. Avvio di un'applicazione Flask

44 Sulla macchina dove si vogliono provare gli esempi devono essere installati i medesimi requisiti per il funzionamento di Flask nella VM oggetto del capitolo 2.

45 Grinberg, Miguel, *Op.Cit.*. In: Chapter 2, Basic Application Structure, pp. 8-10.

L'applicazione sarà così usufruibile da browser, all'indirizzo http://127.0.0.1:5000/ (localhost). Il risultato sarà quello mostrato in figura 5.

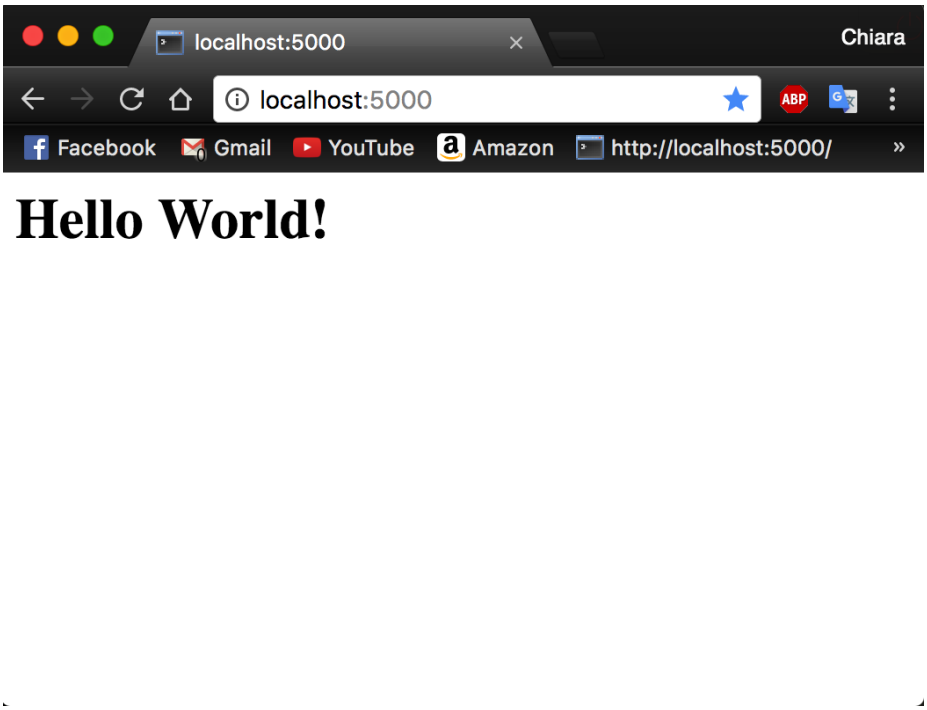

Figura 5. Esempio di applicazione Flask in esecuzione

Con quanto mostrato, si avrà un'infarinatura base per creare le fondamenta di un'applicazione simile allo strumento didattico proposto. Si parlerà in seguito delle ulteriori classi ed estensioni di Flask utilizzate, per capire cosa gestiscono all'interno di esso.

4.1.1.1 Request, Response e architettura REST

L'applicazione in esame basa il suo principale funzionamento su un ciclo di richiesta-risposta (*request-response cycle*) tra web server e client. Quando Flask riceve una richiesta dal client, c'è la necessità di creare degli oggetti da far gestire alla funzione view; un buon esempio è *request*, che incapsula nel protocollo HTTP quanto inviato dal client. Per evitare di ingombrare le views con troppi argomenti che possono essere inutili in quel momento, Flask usa i contesti (*contexts*) per rendere gli oggetti

accessibili globalmente solo temporaneamente. Grazie a questi ultimi, possono essere scritte funzioni view come quella seguente:

```
from flask import request
@app.route('/')
def index():
      user_agent = request.headers.get('User-Agent') return '<p>Your
      browser is %s</p>' % user_agent
```

In questo caso, la funzione request è trattata come una variabile globale, ma il context permette di non farla interferire con altro; tale *feature* si rivela utile soprattutto per l'impiego su server *multithreaded*, dei quali però non si tratterà.

In Flask sono presenti due contesti: quello dell'applicazione (*application context*) e quello della richiesta (*request context*), con quattro variabili di contesto (vedere tabella 2). Flask attiva questi due contesti prima di spedire una richiesta e poi li rimuove quando essa è stata gestita. Quando l'application context è inviato, le variabili `current_app` e `g` diventano disponibili. Allo stesso modo, `request` e `session` lo diventano quando il request context è inviato. Se qualcuna di queste variabili è fuori dal relativo contesto, viene generato un errore.

Nel web server realizzato, tuttavia, è soltanto utilizzata request nel request context, dunque non si tratterà delle altre tre.

Tabella 2. Variabili globali di contesto in Flask

Nome della variabile	Contesto	Descrizione
`current_app`	Application context	L'istanza dell'applicazione per l'applicazione attiva.
`g`	Application context	Un oggetto che l'applicazione può usare per immagazzinare temporaneamente dei dati durante la gestione di una richiesta. La variabile è resettata a ogni richiesta.
`request`	Request context	L'oggetto della richiesta, che incapsula i contenuti di una richiesta HTTP inviata dal client.
`session`	Request context	La sessione dell'utente, un dizionario che l'applicazione può usare per conservare valori da "ricordare" durante le richieste.

Quando Flask invoca una funzione view, aspetta una *response* alla request; nella maggior parte dei casi essa è una stringa inviata al client come pagina HTML; tuttavia, il protocollo HTTP ha bisogno di più di una semplice stringa come risposta: necessita infatti di uno *status code* (codice di stato), che Flask imposta a `200` di default per le richieste andate a buon fine. Se ciò non dovesse verificarsi, si otterrà in risposta sul client un altro codice, come ad esempio `400` (*Bad Request*). [46]

Questo meccanismo di request-response è incluso nell'architettura *REST* (*Representational State Transfer*), che è emersa essere la preferita per le applicazioni web poiché costruita sul modello familiare del *World Wide Web*.

Flask è un framework ideale per costruire applicazioni *RESTful*, grazie alla sua versatilità.

REST è contraddistinto da cinque caratteristiche:

1. *Client-Server*

   Deve esserci una netta separazione tra client e server;

2. *Stateless*

   Una richiesta dal client deve contenere tutto il necessario da mandare in esecuzione; il server non deve conservare nessuno stato del client tra una richiesta e l'altra;

3. *Cache*

   Le risposte dal server possono essere o meno inserite nella cache del client per questioni di ottimizzazione;

4. *Layered System*

   I server proxy, le cache o i gateway possono inframmezzarsi tra il client e il server per migliorare le prestazioni, l'affidabilità e la scalabilità;

5. *Code-on-Demand*

   I client possono scaricare solo se necessario del codice dal server ed eseguirlo nel loro contesto.

46 Grinberg, Miguel, *Op.Cit.*. In: Chapter 2, Basic Application Structure, The Request-Response Cycle, pp.12-13.

Un'applicazione basata sul paradigma REST si serve di una lista di metodi per le richieste HTTP (vedere tabella 3), che determinano il modo in cui le risorse devono essere spedite avanti e indietro tra client e server.

Tabella 3. Metodi di richesta HTTP in un'applicazione RESTful

Metodo	Target	Descrizione	Status code
`GET`	Risorsa individuale URL	Ottenere la risorsa	`200`
`GET`	Collezione di risorse URL	Ottenere la collezione di risorse (o una pagina da essa se il server ha la paginatura implementata)	`200`
`POST`	Collezione di risorse URL	Crea una nuova risorsa e la aggiunge alla collezione. Il server sceglie l'URL della nuova risorsa e la restituisce nella response in un header di allocazione	`201`
`PUT`	Risorsa individuale URL	Modifica una risorsa esistente. Questo metodo può essere utilizzato anche per creare una nuova risorsa quando il client può scegliere l'URL della risorsa.	`200`
`DELETE`	Risorsa individuale URL	Cancella una risorsa.	`200`
`DELETE`	Collezione di risorse URL	Cancella tutte le risorse nella collezione.	`200`

Nello strumento didattico creato sono stati utilizzati unicamente `GET` e `POST`: GET per il caricamento delle risorse (le pagine web) e POST per l'elaborazione dei dati spediti al server, per esempio nei *form*[47]. I metodi da utilizzare sono stati indicati accanto alla route, in modo analogo al seguente:

```
@app.route('/', methods=['GET', 'POST'])
```

e sono stati impiegati quando necessario nelle views.

Solitamente, ciò che si deve inviare a una determinata richiesta è inserito in formati adatti per lo scambio di dati tra client e server; molto usato è il *JSON* (*JavaScript Object Notation*), versatile poiché con una struttura ben trattabile dal lato client via JavaScript. [48]

47 Per ulteriori informazioni sui form, consultare: Grinberg, Miguel, *Op.Cit.*. Chapter 4, Web Forms, Form Handling in View Functions, pp.41-46.

48 Grinberg, Miguel, *Op.Cit.*. In: Chapter 14, Application Programming Interfaces, pp.175-178.

Durante la creazione dello strumento didattico non è stato incapsulato alcun dato in formato JSON: quanto richiesto al server è stato scritto in output senza l'ausilio di tale strategia grazie al linguaggio Jinja2, del quale si parlerà più avanti.

4.1.1.2 Render_template e Jinja2

La funzione `render_template` integra Jinja2, il *template engine* di Flask, nell'applicazione. Essa prende come primo dato in ingresso il nome del file del template (in HTML) e come ulteriori argomenti le variabili (come coppie chiave-valore) da andare a "rappresentare" nella pagina dove viene richiesta.

Un esempio di route che utilizza `render_template` è:

```
@app.route('/user/<name>')
def user(name):
      return render_template('user.html', name=name)
```

In questo caso, nella pagina *user.html* viene chiamato il valore immesso nella variabile `name` nel momento in cui in essa verrà inserito:

```
{{name}}
```

Tale sintassi è tipica di Jinja2, il linguaggio di *templating* usato da Flask per richiamare nelle pagine web le variabili definite in Python lato server. Jinja2 riconosce qualsiasi tipo di variabile, da singoli valori o stringhe a strutture dati complesse come dizionari, liste e oggetti. Esse possono essere modificate tramite l'impiego di *filtri*, aggiungibili dopo il nome della variabile usando il carattere *pipeline* (|) come separatore.

Un esempio di variabile con un filtro applicato è:

```
Hello, {{ name|capitalize }}
```

In questo caso, il suo contenuto verrà mostrato in lettere maiuscole grazie al filtro `capitalize`. Nella tabella 4 sono riportati alcuni dei filtri più usati in Jinja2 e la relativa funzione.

Tabella 4. Lista dei filtri più comuni di Jinja2

Nome del filtro	Descrizione
`safe`	Renderizza il valore in modo sicuro (senza caratteri che possono interferire con il rendering)
`capitalize`	Converte in maiuscolo il primo carattere nel valore e lascia invariati i restanti
`lower`	Converte i caratteri nel valore in minuscolo
`upper`	Converte i caratteri nel valore in maiuscolo
`title`	Converte in maiuscolo ogni parola nel valore
`trim`	Rimuove gli spazi bianchi dall'inizio e dalla fine del valore
`striptags`	Rimuove ogni tag HTML dal valore prima del rendering

Jinja2 oltre a riportare semplicemente le variabili definite lato server offre diversi metodi per il controllo di esse, come comandi condizionali e iterazioni.

Un esempio di dichiarazione di comando condizionale in Jinja2 è:

```
{% if user %}
        Hello, {{ user }}!
    {% else %}
        Hello, Stranger!
{% endif %}
```

Per le iterazioni, invece, si può procedere come segue:

Per le liste semplici:

```
{% for comment in comments %}
      {{ comment }}
{% endfor %}
```

Per i dizionari[49]:

```
{% for key, value in my_dict.iteritems() %}
    {{ key|e }}
    {{ value|e }}
{% endfor %}
```

La sintassi è molto semplice e pulita e, per tale motivo, facilmente interpretabile. Come si evince dai precedenti esempi, ciascuna struttura di controllo è racchiusa da

49 *Template Designer Documentation*: *List of Control Structures.* Link alla guida online di Jinja2: http://jinja.pocoo.org/docs/2.9/templates/.

un delimitatore di apertura e uno di chiusura, entrambi con la sintassi `{%controllo %}`.

Allo stesso modo di un controllo o di un ciclo, c'è la necessità di delimitare alcuni elementi che possono essere modificati da un template derivato; ciò può essere effettuato con il tag `block`, che identifica un blocco. Nel seguente esempio, ci sono dei blocchi chiamati `title`, `head` e `body`:

```
 {% extends "base.html" %}
    {% block title %}Index{% endblock %}
    {% block head %}
{{ super() }}
<style>
</style>
{% endblock %}
{% block body %}
<h1>Hello, World!</h1>
{% endblock %}
```

Il tag `extends` sta a significare che il template include elementi derivati da *base.html*, dove sono definiti. Tale direttiva è seguita dalle nuove definizioni dei tre blocchi, inseriti nel posto appropriato del template in uso. Nel blocco `head` è inserito il tag `{{super()}}` per mantenere il contenuto originale e non rimpiazzarlo con quello contenuto in *base.html*, dove la head non è vuota.

4.1.1.3 Flask_bootstrap

Bootstrap è un framework open source derivato da *Twitter* che offre una serie di componenti per la creazione di pagine web *responsive*, ideato per essere compatibile con tutti i nuovi browser web. Essendo *client-side*, non interferisce con il server direttamente; tutto ciò che quest'ultimo deve fornirgli è la risposta HTML connessa al CSS e al JavaScript che utilizza per il caricamento dei componenti desiderati.

Normalmente, gli sviluppatori *front-end* utilizzano Bootstrap importandolo direttamente all'interno delle pagine HTML via collegamento nella head, ma con Flask è possibile procedere in modo più pulito.

Per semplificare l'integrazione con il template, infatti, Flask dispone di un'estensione chiamata `flask_bootstrap` (installabile come componente aggiuntivo tramite `pip`), importabile come segue nell'applicazione:

```
from flask_bootstrap import Bootstrap
```

ed inizializzabile così:

```
Bootstrap(app)
```

In seguito, per permettere di implementarla all'interno dei template, viene in aiuto Jinja2 con la direttiva `extends`, che consente di importare il layout al percorso `bootstrap/base.html`, una sorta di scheletro minimale di pagina web[50].

Un esempio di template che utilizza `flask_bootstrap` è il seguente:

```
{% extends "bootstrap/base.html" %}
{% block title %}Flasky{% endblock %}
{% block navbar %}
<div class="navbar navbar-inverse" role="navigation"> <div
class="container">
      <div class="navbar-header">
            <button type="button" class="navbar-toggle"
            data-toggle="collapse" data-target=".navbar-collapse">
                  <span class="sr-only">Toggle navigation</span>
                  <span class="icon-bar"></span>
                  <span class="icon-bar"></span>
                  <span class="icon-bar"></span>Chapter 3: Templates
            </button>
            <a class="navbar-brand" href="/">Flasky</a>
      </div>
      <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
            <li><a href="/">Home</a></li>
            </ul>
      </div>
      </div>
</div>
{% endblock %}
{% block content %}
<div class="container">
      <div class="page-header">
      <h1>Hello, {{ name }}!</h1>
      </div>
```

50 Per approfondire, consultare Grinberg, Miguel, *Op.Cit.*. In: Chapter 3, Templates, pp. 21-29.

```
</div>
{% endblock %}
```

4.1.1.4 Flask_Nav

Flask_Nav[51] è un'estensione di Flask per creare facilmente gli elementi di navigazione (menù e voci di menù) nei template senza doverli inserire manualmente ogni volta. Il suo utilizzo è molto semplice e si avvale di due parti: una all'interno dell'applicazione e una nel template.

Nell'app Flask l'estensione è importabile con:

```
from flask_nav import Nav
```

Per la gestione della *navbar* con all'interno le voci di menù da associare alle funzioni *view* è invece necessario inserire quanto segue:

```
from flask_nav.elements import Navbar, View
```

È necessario poi richiamare l'oggetto `Nav()` e assegnarlo alla variabile `nav`:

```
nav = Nav()
```

Per la creazione di una barra di navigazione con le voci `Home` e `Hello`, si dovrà procedere nel seguente modo:

```
@nav.navigation()
def mynavbar():
  return Navbar(
      View('Home', 'home'),
      View('Hello', 'hello'),
  )
```

e inserire all'interno delle pagine HTML nelle quali si vuole mostrare il menù

`{{nav.mynavbar.render()}}`, preferibilmente inserito in un block, come segue:

```
{% block navbar %}
{{nav.mynavbar.render()}}
{% endblock %}
```

51 *Flask-nav*. Link al repository: https://github.com/mbr/flask-nav.

Come si può notare, a ogni voce di menù dovrà corrispondere una funzione view. Al click su ognuna di esse, si verrà ricondotti al relativo template con la view.

Il menù si integra bene con gli elementi di Bootstrap e ha un aspetto pulito ed essenziale (vedere figura 6).

Figura 6. Menù creato con Flask_Nav

Quanto scritto nelle sezioni precedenti illustra il funzionamento essenziale di Flask e delle relative componenti utilizzate per la creazione del progetto, del quale si tratterà nel capitolo seguente. Nello schema sottostante è riportato un riassunto del funzionamento generale di Flask (figura 7).

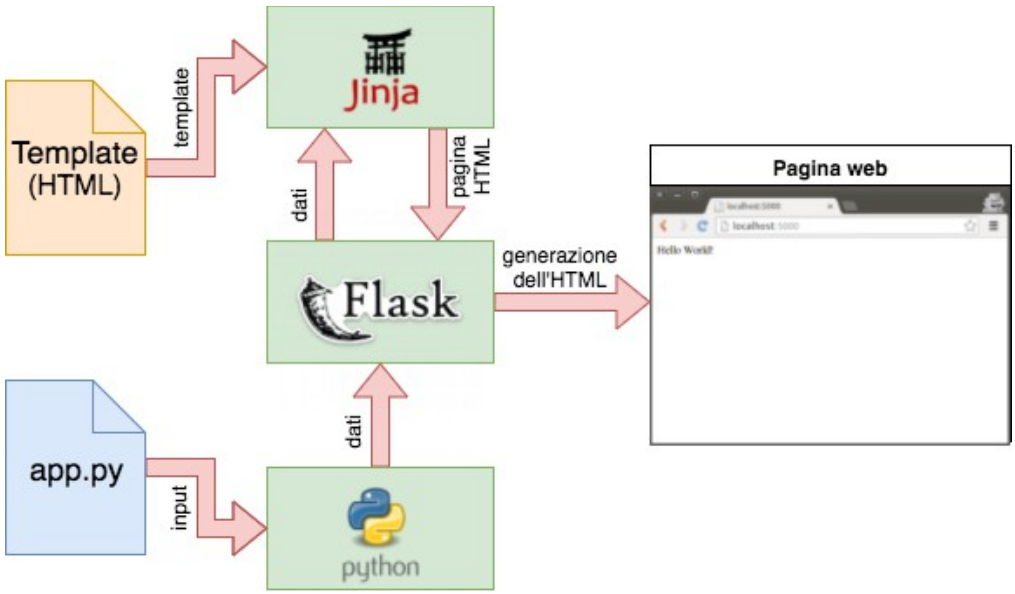

Figura 7. Funzionamento generale di Flask

4.1.2 Moduli standard di Python

Oltre alle librerie NLTK (del quale si è trattato nel capitolo precedente) e Flask, sono stati inclusi nel programma alcuni moduli contenuti nell'installazione standard di Python per eseguire determinate operazioni, delle quali si parlerà in seguito.

4.1.2.1 Regular Expressions: re

Re è il modulo di Python che permette di effettuare il *matching* di caratteri all'interno di stringhe tramite l'utilizzo di espressioni regolari. Esso risulta utile in associazione con NLTK per la pulizia del testo da caratteri speciali o ad esempio per prendere in considerazione una porzione di testo con determinate caratteristiche invece che un'altra. Per utilizzare tale modulo, è necessario importarlo con:

```
import re
```

Un esempio di espressione regolare è il seguente, per eliminare tutto ciò che non rientra tra caratteri (minuscoli e maiuscoli), numeri e carattere "_" (*underscore*):

```
.*[A-Za-z0-9].*
```

dove:

"`*`" indica "zero o più occorrenze";

"`.`" identifica "ogni carattere tranne quello di invio a capo";

`[A-Za-z0-9]` sono i caratteri da prendere in considerazione (tutti i caratteri ASCII sia maiuscoli che minuscoli, i numeri da zero a nove e l'underscore).

Per comodità, il *pattern* (la sequenza di caratteri speciali per compiere determinate operazioni) di un'espressione regolare è di solito incorporato in un oggetto con `re.compile`:

```
noPunt=re.compile('.*[A-Za-z0-9].*')
```

così da essere pronto per effettuare il matching successivamente, come nel seguente ciclo all'interno di una variabile :

```
tes_tok_nopunt = [w for w in testo_tokenizzato if noPunt.match(w)]
```

Il risultato ottenuto è la variabile contenente il testo ripulito dalla punteggiatura e da caratteri speciali.

I pattern utilizzabili sono molti e sono reperibili nel manuale online di Python[52]; pertanto nello strumento realizzato è stata solo utilizzata la precedente espressione regolare con il rispettivo pattern e un'altra per la gestione degli underscore.

4.1.2.2 Collections e OrderedDict

Collections è un modulo che introduce delle strutture dati particolari all'interno del codice, utili per risolvere alcuni problemi relativi a quelle incluse di default in Python. Una di queste, è *OrderedDict*.

OrderedDict[53] è una sottoclasse di collections, utile per la creazione di dizionari ordinati. Infatti, in Python (come in altri linguaggi di programmazione), la struttura dati dei dizionari viene ottimizzata per la ricerca e per tale motivo gli elementi non vengono inseriti al loro interno seguendo l'ordine di immissione. In soccorso a tale situazione, viene OrderedDict che, rispetto ai dizionari standard *dict*, ricorda l'ordine in cui è inserito il contenuto.

Per l'utilizzo, innanzitutto devono essere importati il modulo `collections` e la classe `OrderedDict` :

```
import collections
from collections import OrderedDict
```

Successivamente, si dovrà dichiarare quanto inserire nel dizionario, come nell'esempio seguente:

```
d = collections.OrderedDict()
d['a'] = 'A'
d['b'] = 'B'
d['c'] = 'C'
```

Il dizionario così ottenuto presenterà le coppie chiave-valore ordinate tra loro:

```
OrderedDict([('a', A), ('b', B), ('c', C)])
```

52 *Regular Expression HOWTO,* Kuchling.
Link al sito: https://www.tutorialspoint.com/python/python_reg_expressions.htm.

53 *OrderedDict: Remember the Order Keys are Added to a Dictionary*.
Link al sito: https://pymotw.com/3/collections/ordereddict.html.

In un dizionario comune, la struttura ottenuta sarebbe stata simile alla seguente:

```
{'a': A, 'c': C', 'b': B'}
```

Nello strumento didattico è stato necessario usare tale tipo di dizionario poiché vi era la necessità di avere in output le parole così come venivano inserite in input. Nel capitolo successivo verrà spiegato nel dettaglio come è stato utilizzato OrderedDict, in riferimento al progetto.

4.2 Markup e static files

Nelle sezioni precedenti si è parlato di Python, usato per la realizzazione dell'applicazione lato server. In questo sottocapitolo si tratterà invece di ciò che costituisce la parte client, ovvero di cosa è impiegato per costruire le pagine web dopo una risposta dal server. Come già detto in precedenza, i template sono scritti in codice HTML e hanno al loro interno parti in Jinja2 (oltre che sezioni in JavaScript) per comunicare con l'applicazione contenente le direttive di Flask.

Tuttavia, una pagina web per essere completa ha bisogno di altri elementi che devono essere collegati a essa, ovvero di fogli di stile, script in JavaScript e immagini.

Questi ultimi sono definiti *static files* e sono inseriti nella cartella *static*, nidificata all'interno della directory principale dell'applicazione in Python. Tali files sono trattati in modo speciale: a ognuno di essi viene infatti assegnato un percorso standard, che inizia con `/static/` e vengono inclusi nei templates come segue:

```
{% block head %}
{{ super() }}
<link rel="icon" href="{{ url_for('static', filename =
'favicon.ico') }}" type="image/x-icon">
{% endblock %}
```

Nell'esempio, all'interno dell'`href` non viene inserito il classico collegamento come quando si crea una comune pagina web, ma un costrutto di Jinja2 che indica al template la cartella dove risiede il file (`static`) e il relativo nome ed estensione

(`favicon.ico`). La sintassi `{{url_for('static', filename='')}}` è utilizzata per importare nel template qualsiasi tipo di risorsa richieda la pagina per il funzionamento.

4.2.1 Bootstrap e CSS

Come già accennato, la struttura e la grafica delle pagine web sono state costruite utilizzando il framework Bootstrap con i relativi script e fogli di stile. Per una questione di comodità, esso è stato integrato in Flask e importato tramite Jinja2 con il costrutto `extends` e dunque non linkato direttamente nei templates (tranne che nel template *layout.html*). Tuttavia, è stato necessario ritoccare alcuni parametri (colori e dimensioni) dello stile tramite un altro foglio CSS con delle regole in grado di sovrascrivere quelle di Bootstrap.[54]

4.2.2 JavaScript, jQuery e jQuery UI

JavaScript (abbreviato in *JS*), un linguaggio di scripting orientato a oggetti e a eventi e è il motore dei contenuti dinamici lato client nelle pagine web. Nell'applicazione per la didattica realizzata, JS svolge un ruolo molto importante poiché consente di caricare su richiesta alcuni contenuti e permette all'utente di interagirvi. Jinja2 si integra perfettamente con esso se introdotto nei templates[55] e grazie a questo è possibile inserire dinamicamente quanto definito lato server all'interno di elementi come, per esempio, i *div*. Per una questione di semplicità nella gestione di eventi e contenuti è stata introdotta *jQuery*, una libreria di JavaScript piena di funzionalità, indispensabile per il funzionamento delle attività. Come ogni file nell'applicazione, la libreria è stata scaricata dal sito ufficiale[56] per poterla utilizzare offline e integrata come ogni altro file all'interno della head del template, via Jinja2. Inoltre, è stata impiegata *jQuery User Interface* (*jQuery UI*) per la gestione degli *slot* contenenti le

54 *CSS*, *Bootstrap*, *JavaScript*. Per maggiori informazioni, consultare il sito W3Schools: https://www.w3schools.com/.

55 Pertanto, JavaScript all'esterno dei template non supporta la sintassi di Jinja2. Per rendere visibili le variabili a script non nel corpo delle pagine web, è necessario utilizzare JSON, ma ciò non è trattato poiché piuttosto laborioso.

56 JQuery, link al sito: https://jquery.com/.

parole da associare alla relativa parte del discorso nell'attività di analisi grammaticale.

Essa è una raccolta di strumenti, effetti, temi e *widgets* per l'interazione dell'utente con l'interfaccia e, come si evince dal nome, è basata su jQuery. Come quest'ultima, anche i files di jQuery UI sono stati scaricati e inseriti nella cartella static: è stato necessario collegare al template il file con estensione *.js* per il codice e quello *.css* per gli stili. Anch'essa, infatti, come Bootstrap, ha un proprio foglio di stile con delle regole predefinite; è inoltre possibile scaricare dal sito ufficiale vari temi grafici per l'interfaccia utente a seconda delle esigenze. In tal caso è stato tuttavia utilizzato il tema di default. Essenzialmente, di jQuery UI sono state usati due plugin: *draggable* e *droppable*, per permettere all'utente di trascinare alcuni contenuti (gli slot con le parole) e rilasciarli in una determinata posizione (gli slot con le parti del discorso).

Draggable, il plugin per permettere il trascinamento, è richiamabile mediante il metodo `draggable()`. Esso è applicabile a ogni elemento con un determinato *id* o *classe* e supporta diverse opzioni che possono essere specificate con la seguente sintassi:

```
$("#trascinami").draggable({opzione:valore});
```

Alcune delle opzioni disponibili per il plugin sono riportate nella tabella 5.

Tabella 5. Opzioni del metodo draggable()

Opzione	Tipo	Default	Descrizione
`axis`	stringa	`false`	Limita il drag solo orizzontalmente ('x') o verticalmente ('y').
`containment`	selettore, elemento, stringa, array	`false`	Delimita l'area entro il quale l'elemento può essere draggato. Possibili stringhe sono 'document', 'parent', 'window'.
`cursor`	stringa	`'auto'`	Serve per determinare l'aspetto del cursore del mouse durante l'effetto; si utilizzano le specifiche dei CSS per l'attributo *cursor* (ad es. *pointer*).
`delay`	intero	`0`	Tempo in millisecondi tra l'azione dell'utente (click e trascinamento) e lo scatenarsi dell'effetto. Serve per evitare drag accidentali.
`disabled`	booleano	`false`	Disabilita il plugin se impostato a true.
`distance`	intero	`1`	Indica la distanza minima (in pixel) del trascinamento prima che si scateni l'effetto. Anche questo serve a prevenire drag non voluti.
`handle`	elemento, selettore	`false`	Definisce un elemento che fungerà da "maniglia" per il trascinamento. L'effetto partirà quando l'utente cliccherà ed effettuerà il trascinamento di questo elemento.
`helper`	stringa, funzione	`original`	Elemento da mostrare durante il drag. Strighe possibili: 'original' o 'clone'. Se impostato su 'clone', per esempio, verrà creato un clone dell'oggetto da trascinare e sarà spostato quest'ultimo invece dell'elemento originale fino al suo rilascio.
`opacity`	float	`false`	Imposta l'opacità dell'helper mentre è in movimento. Può essere utile, infatti, creare un effetto trasparenza sull'elemento in movimento.
`revert`	booleano, stringa	`false`	Se impostato a true, l'elemento ritorna nella posizione iniziale quando il drag è terminato. Possibili stringhe: 'valid' o 'invalid'

È possibile associare a quanto *draggato* alcune funzioni di *callback* in associazione agli eventi generati dall'elemento trascinato. Queste sono `start`, `drag` e `stop` e servono rispettivamente per segnalare l'inizio, il movimento e il termine dello spostamento. Nell'esempio seguente, è mostrato come possono essere utilizzate.

```
$("#trascinami").draggable({
  start: function() {
    alert("Inizio...");
  },
  drag: function() {
    alert("...mi stai trascinando...");
```

```
  },
  stop: function() {
    alert("...sono fermo");
  }
});
```

Per modificare il comportamento del plugin esistono inoltre dei metodi:

- `.draggable ("destroy")`, che consente all'elemento di tornare allo stato iniziale, rimuovendone le funzionalità associate del plugin;
- `.draggable("disable")`, che consente di bloccare l'operazione di spostamento dell'elemento;
- `.draggable("enable")`, che consente di abilitare l'operazione di spostamento dell'elemento;
- `.draggable("option",nome_opzione,[valore])`, che consente di impostare un'opzione aggiuntiva;
- `.draggable("widget")`, che consente di restituire un elemento con associata la classe `ui-draggable` (necessaria a identificare ciò che deve essere spostato).

Essi sono utilizzabili come segue:

```
$("#trascinami").draggable("nome_metodo"[,parametri]);
```

L'altro plugin citato invece, droppable, serve a stabilire una destinazione (*target*) per quanto etichettato come draggable; detto in altri termini, va a definire quale elemento si attiverà o compirà una determinata azione quando uno draggable vi sarà posizionato sopra.

Per comprenderne meglio il funzionamento, viene fornito un esempio completo di codice HTML con l'elemento da trascinare ("`trascinami`") e la posizione in cui rilasciarlo ("`destinazione`"):

```
<script type="text/javascript">
$(document).ready(function() {
```

```
  $("#trascinami").draggable();
  $("#destinazione").droppable({
    drop: function(event, ui) {
      $(this).find('p').html('Eccomi!');
    }
  });
});
</script>
<div id="trascinami">
  <p>Spostami all'interno del box qui accanto...</p>
</div>
<div id="destinazione">
  <p>Portami qui dentro!</p>
</div>
```

Come si può intuire dall'esempio, nel momento in cui l'elemento trascinabile viene posizionato nella posizione specificata, il testo contenuto all'interno del `div` "`destinazione`" viene cambiato in "`Eccomi!`". Analogamente a draggable, anche droppable presenta delle personalizzazioni effettuabili tramite parametri, metodi e callback a un determinato evento (vedere tabella 6).

Inoltre, i metodi previsti dal plugin sono gli stessi di Draggable.

Tabella 6. Opzioni del metodo droppable()

Opzione	Tipo	Default	Descrizione
`accept`	selettore, funzione	`'*'`	Permette di definire quale elemento draggable sarà accettato. Di default tutti, ma si possono anche effettuare diversificazioni (indicando, per esempio, il selettore del solo elemento accettato).
`activeClass`	stringa	`false`	Classe da aggiungere all'elemento droppable mentre avviene il drop.
`disabled`	booleano	`false`	Disabilita il plugin se impostato a true.
`tolerance`	stringa	`'intersect'`	Definisce la modalità con la quale considerare avvenuto il drop. Le possibili stringhe sono: 'fit', 'intersect', 'pointer' e 'touc'h.

In connessione a questo plugin, vi sono degli eventi che determinano il comportamento degli elementi draggable e droppable all'inizio, alla fine o durante lo spostamento:

- `activate`, che stabilisce quando far iniziare il movimento dell'elemento draggable;
- `deactive`, che termina il movimento dell'elemento draggable accettato;
- `over`, che stabilisce che l'elemento draggable è sopra quello droppable una volta accettato;
- `out`, che stabilisce che l'elemento draggable sta uscendo da quello droppable, una volta accettato;
- `drop`, che stabilisce che l'elemento droppable è stato rilasciato e che il trascinamento è terminato.[57]

4.2.3 ResponsiveVoice

ResponsiveVoice[58] è una libreria di sintesi vocale basata su *HTML 5* che permette di leggere ad alta voce il testo su siti web e applicazioni per smartphone, tablet e PC. Essa è molto completa, poiché supporta ben 51 lingue e 168 voci e è priva di dipendenze. Inoltre, pesa solo 14kb e dunque è leggerissimo. Per lo strumento didattico è stata impiegata all'interno della sezione "Genera, ascolta e ripeti!", destinata alle classi prime e seconde. Il funzionamento di ResponsiveVoice si basa sul sistema *text-to-speech* di terza generazione (vedere figura 8), che non prevede più la creazione sul server di file audio in formato *MP3* con il parlato corrispondente alle parole ma si basa unicamente sul motore di sintesi vocale presente nella macchina. Il suono corrispondente al testo (che viene, innanzitutto, tokenizzato) viene generato e riprodotto direttamente nel browser; tale metodo permette di risparmiare tempo nel *rendering* della pagina web, ha latenza ridotta e non prevede necessariamente una connessione di rete.

57 *Guida jQuery UI di Mr. Webmaster*. 2009. *Draggable e Droppable*. Link di riferimento: https://www.mrwebmaster.it/jquery/guide/guida-jquery-ui/.

58 *ResponsiveVoice*, link al sito del progetto e alla relativa documentazione: https://responsivevoice.org/.

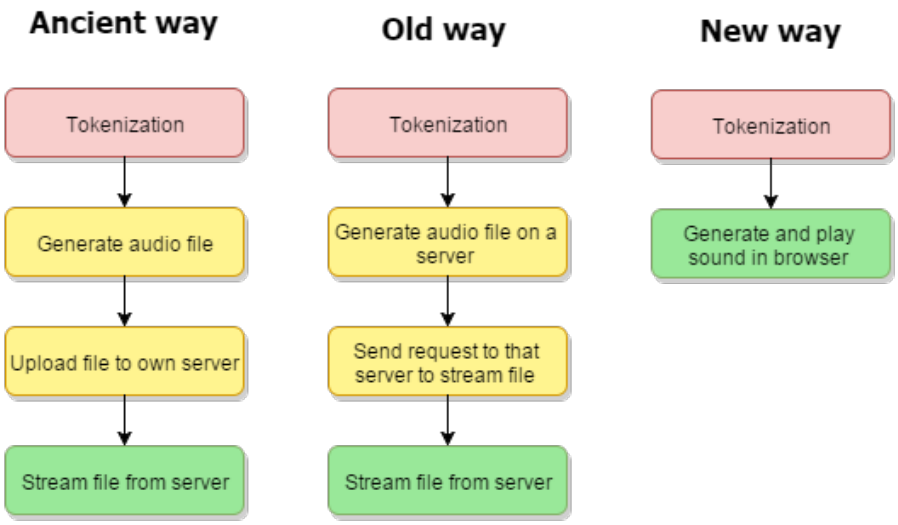

Figura 8. Differenti generazioni di text-to-speech

Di default, la libreria ha come voce di sintesi vocale "*UK English Female*", ovvero di donna parlante l'inglese britannico. È stata lasciata invariata l'opzione nell'attività didattica preparata, poiché sembrava essere quella più adatta. Come accennato precedentemente, ResponsiveVoice funziona nel browser anche senza un collegamento a Internet: ciò è vero, ma purtroppo con alcune limitazioni. Infatti, pur essendo scaricabile in locale, essa ha il necessario per la gestione delle voci su server. Di conseguenza, senza connessione il motore di sintesi funzionerà comunque ma utilizzando la voce nella lingua di sistema, meno accurata, soprattutto in caso di macchina con lingua di base diversa dall'inglese. Se si desidera utilizzare lo strumento offline, dunque, è buona norma impostare come lingua di sistema l'inglese per l'occasione.

L'utilizzo di ResponsiveVoice è molto semplice: è necessario importare la libreria (*responsivevoice.js*) e utilizzarla in questo modo, con la voce di default:

```
responsiveVoice.speak('parola da leggere')
```

Naturalmente, è possibile impiegarlo nella creazione di testo dinamico generato via JavaScript o con variabili ottenute dal server con Jinja2, ad esempio.

Quanto descritto verrà riportato in relazione allo strumento didattico realizzato, nel capitolo seguente.

5. La realizzazione del progetto

In questo capitolo verrà illustrato come è stato realizzato lo strumento didattico da essere utilizzato durante le ore di insegnamento della lingua inglese per verificare le competenze acquisite dagli alunni. Grazie a quanto scritto in precedenza, sarà possibile ora comprendere i vari passi che sono stati effettuati per creare il prodotto finito.

Poiché il codice completo verrà fornito in *Appendice*, verranno illustrati nel dettaglio solo i punti salienti del progetto.

A seguire, sono inseriti degli *screenshots* dell'applicazione, prima di entrare nel pieno della descrizione.

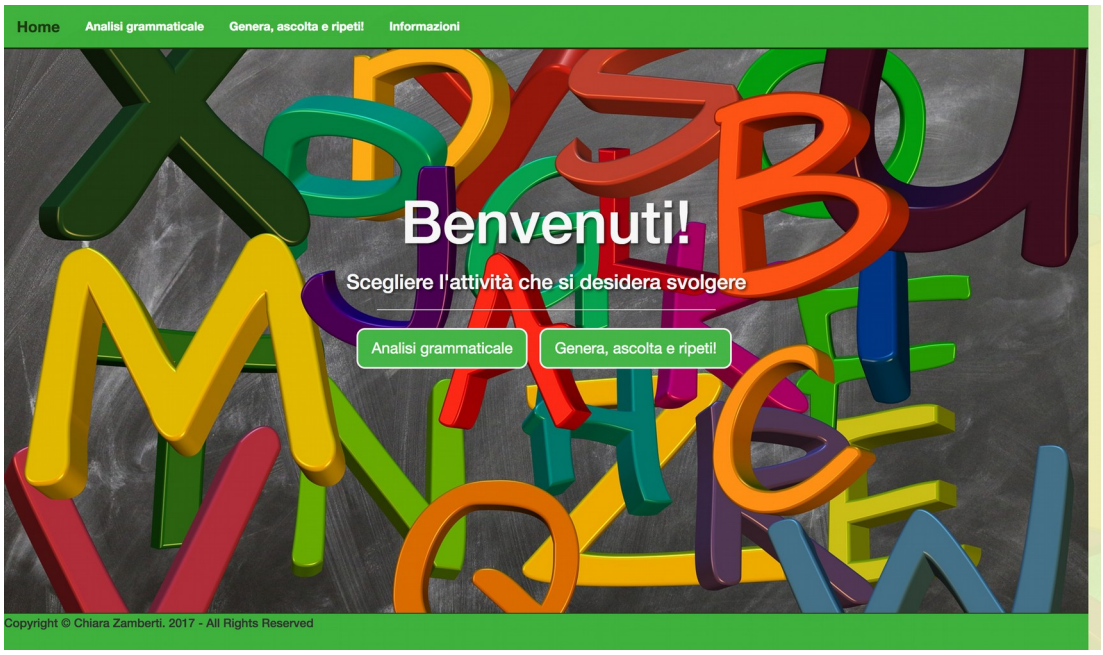

Figura 9. Homepage

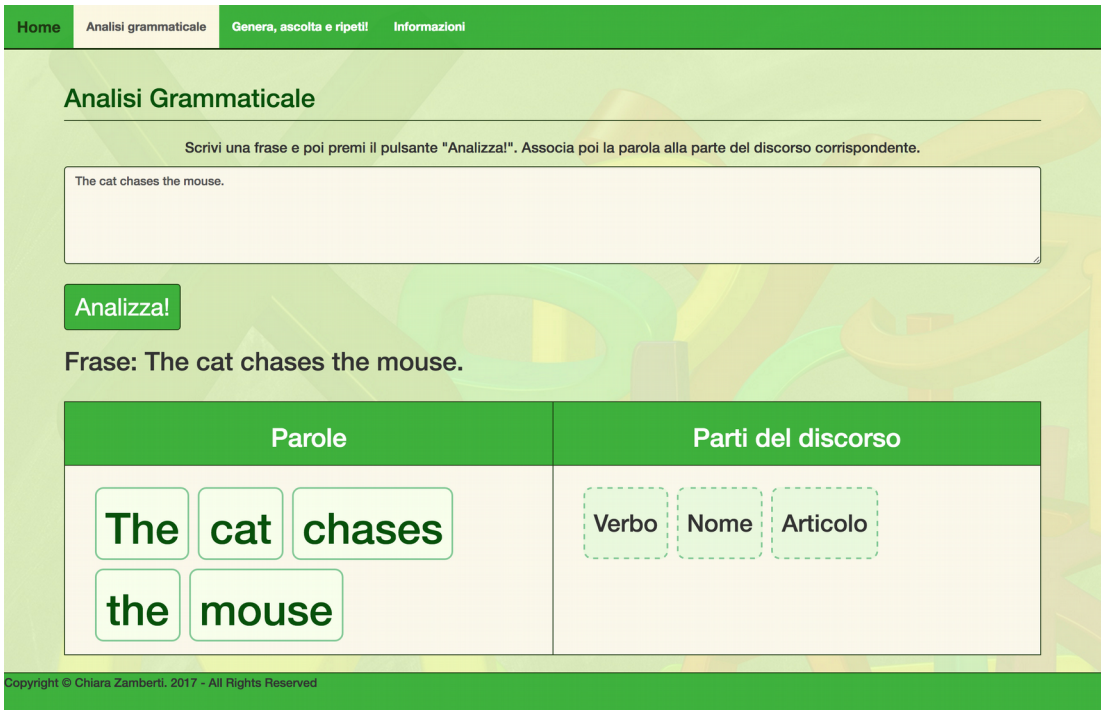

Figura 10. Analisi grammaticale

Figura 11. Genera, ascolta e ripeti!

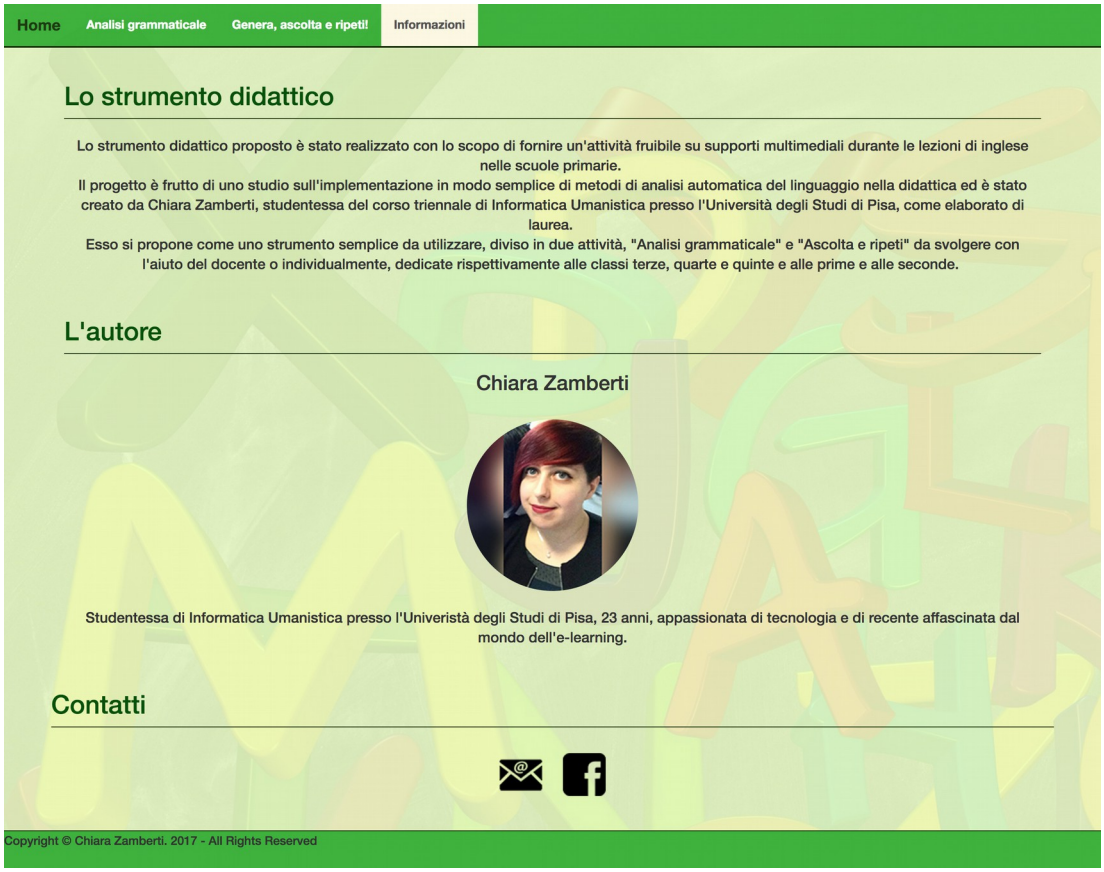

Figura 12. Informazioni

5.1 InteractiveEnglish.py

Il primo passo verso la creazione del prodotto in oggetto è stato quello di realizzare l'applicazione Python *interactiveEnglish.py*, partendo dalla struttura dell'esempio *hello.py* citato nel capitolo 4, alla quale sono state poi aggiunte passo passo le componenti necessarie.

Essa si compone di quattro parti parti principali:

- una dedicata allo strumento per l'analisi grammaticale interattiva;
- una per la sezione con l'attività di generazione e ascolto di parole casuali da sette categorie di vocaboli;
- una per la gestione della homepage;
- una per la pagina contenente le informazioni sullo strumento e sull'autore.

Innanzitutto, sono stati importate dalla libreria `flask` le classi `Flask`, `render_template` e `request`, necessarie come si è detto in precedenza per la gestione dei template. Successivamente, è servito includere la classe `Bootstrap` dalla relativa libreria `flask_bootstrap` per la struttura e la grafica di questi ultimi. Di plugin di Flask è stato anche introdotto `flask_nav` con la classe `Nav` e le componenti `Navbar` e `View`. Quanto elencato è tutto ciò che serve per il funzionamento del framework nel caso in esame. Dopo tali operazioni, è stato necessario importare la libreria `nltk` e la classe `PlainTextCorpusReader`, per la lettura di corpora (in formato *txt*) caricati esternamente. Sono state inoltre incluse la libreria `re` per la gestione delle espressioni regolari e la classe `OrderedDict` da `collections` per la creazione di dizionari ordinati.

Fatto ciò, si è proceduto a creare un'istanza dell'applicazione, è stato richiamato l'oggetto `Nav()` per la creazione dei menù di navigazione ed è stata inizializzata la classe `Bootstrap`.
A questo punto si è rivelato utile creare una `view` con il contenuto dei menù di navigazione, uguale per tutti i template, con nav, come segue:

```
@nav.navigation()
def mynavbar():
  return Navbar(
      View('Home', 'index'),
      View('Analisi grammaticale', 'analisi_grammaticale'),
      View('Genera, ascolta e ripeti!', 'genera_parola'),
      View('Informazioni', 'about')
  )
```

5.1.1 Analisi_grammaticale

Si è passati a tal punto alla definizione delle routes e delle relative views per i quattro template detti in precedenza. La prima route (con la relativa funzione view) creata è stata `analisi_grammaticale`, quella dell'attività che prevede il gioco di abbinamenti di parole alle relative parti del discorso.
In associazione al percorso, sono stati anche inseriti i metodi da utilizzare per le richieste HTTP, gestite da `request`: `GET`, per il caricamento degli elementi del

template e `POST` per le operazioni di invio e ricezione di dati tra client e server. Sono stati poi inizializzati le variabili, le liste e i dizionari da essere utilizzati successivamente (`raw`, `lungfrase`, `postag`, `dictionary`, `parole`, `partedisc`). A questo punto, è stato inserito un `if` per determinare il metodo da utilizzare per le operazioni successive: `POST`. Prima del comando condizionale, infatti, è adottato `GET` come metodo predefinito. È in seguito definita la sorgente dalla quale prendere il testo in input: dal *form* contenuto nel template `analisi_grammaticale` (vedere sez. sui template), costruito parallelamente alla view:

```
raw = request.form.get('analisi_grammaticale')
```

Una volta effettuate tali operazioni preliminari, si è passati alla realizzazione del vero e proprio motore della view in questione (e di un po' tutto il programma), che permette di effettuare l'elaborazione e l'analisi delle frasi scritte dall'utente in input: NLTK. È stato quindi, innanzitutto, definito il percorso della cartella dove sono presenti tutti i moduli di quest'ultima. Solitamente, essi si trovano all'interno della cartella `nltk_data`, pertanto la sintassi del percorso da inserire nel programma è:

```
nltk.data.path.append('./nltk_data/')
```

Dopo aver puntato alla cartella, si è proceduto a definire la variabile `testo_tokenizzato`, ovvero quella nella quale viene racchiuso il testo diviso in token. Infatti, dopo l'immissione di testo nel form e il suo successivo invio (*submit*) al server, avviene l'incapsulamento nella variabile `raw` ma, per essere elaborato, esso deve essere diviso in unità singole, dette *token*.
Il testo tokenizzato verrà immesso nella variabile, nel modo seguente:

```
testo_tokenizzato = nltk.word_tokenize(raw)
```

dove vi è una chiamata al tokenizzatore `word_tokenize`, che va a processare il testo nella variabile `raw`.
A tal punto il testo, diviso in tante entità, deve essere pulito da caratteri speciali e punteggiatura, poiché è necessario lavorare su dati più puliti possibile per evitare problemi nella procedura di annotazione sintattica. Per fare ciò, viene definita un'espressione regolare per rimuovere tutto ciò che non rientra tra caratteri della

tabella *ASCII* minuscoli, maiuscoli e numeri e tale pattern è inserito in un oggetto, come segue:

```
nonPunct = re.compile('.*[A-Za-z0-9].*')
```

Successivamente, questa è applicata al testo, utilizzando `re.match` in un ciclo `for` *inline* che lo scorre parola per parola:

```
tes_tok_nopunt = [w for w in testo_tokenizzato if nonPunct.match(w)]
```

Viene inoltre popolata la variabile `lungfrase`, precedentemente inizializzata, con la lunghezza della stringa contenuta in `tes_tok_punt`, utilizzando la funzione `len`. La variabile servirà successivamente come condizione di terminazione per il gioco di associazione della parola alla parte del discorso. In questo momento, il testo è pronto per l'operazione di *part of speech tagging.* Per fare ciò, viene creata la lista `postag`, che presenterà una struttura dati particolare, come visto in precedenza, nella quale sono inserite le tuple parola-parte del discorso. `Postag` contiene al suo interno `tes_tok_nopunt`, alla quale è applicato il modulo `nltk.postag`, come segue:

```
postag=nltk.pos_tag(tes_tok_nopunt)
```

Dato che `postag` presenta una struttura alquanto complessa da elaborare (in particolare nel template, con Jinja2), è divisa in due liste: `parole` e `partedisc`, precedentemente inizializzate. Viene dunque eseguito un ciclo `for` su di essa e le parole vengono inserite in `parole`, mentre le parti del discorso (POS) in `partedisc`, con il metodo `.append()`:

```
for n in postag:
      parole.append(n[0])
      partedisc.append(n[1])
```

Da questo momento, si avranno (se propriamente renderizzate e inserite nel template) in output le due liste con le parole e le abbreviazioni standard del POS, come in tabella 1.

Essendo l'applicazione destinata ai bambini, che conoscono poco l'inglese e non sono abituati di certo a nomenclature tecniche o abbreviazioni, si è proceduto a rinominare le etichette con le corrispondenti parti del discorso in italiano e a

semplificarle. Il tutto è stato effettuando iterando su `partedisc`, precedentemente usata come argomento della funzione `enumerate` (che serve ad associare un numero di indice a ciascuna parte del discorso). È possibile vedere quanto detto nella porzione di codice seguente:

```
for n,i in enumerate(partedisc):
            if i=="CC":
                  partedisc[n]="Congiunzione"
            if i=="CD":
                  partedisc[n]="Numero"
            if i=="DT":
                  partedisc[n]="Articolo"
            if i=="EX":
                  partedisc[n]="'There' esistenziale"
            if i=="FW":
                  partedisc[n]="Parola straniera"
            if i=="IN":
                  partedisc[n]="Preposizione"
            if i=="JJ":
                  partedisc[n]="Aggettivo"
            if i=="JJR":
                  partedisc[n]="Aggettivo"
            if i=="JJS":
                  partedisc[n]="Aggettivo"
```

Come si può constatare dalla parte di codice riportata, ogni occorrenza di ciascuna etichetta viene sostituita da quella corrispondente tradotta in italiano. Inoltre, è stata effettuata un'operazione di unificazione di parti del discorso appartenenti alla stessa categoria: per esempio gli aggettivi possessivi, i superlativi di maggioranza, minoranza ecc. sono stati tutti raggruppati sotto il tag "Aggettivo". Tale scelta è stata fatta su suggerimento della docente Cinzia di Marco poiché i bambini fino alla quinta elementare non hanno ancora le competenze necessarie per distinguere i vari tipi di aggettivo, avverbio, nome e altro della lingua inglese.

Successivamente, le liste create sono state organizzate in una struttura ottimale per essere poi processate all'interno del template. Dunque, sono state prima compresse in un'unica grande lista con la funzione `zip`, che a sua volta è stata passata come argomento a `OrderedDict`, per la creazione di un dizionario.

Dato che nei dizionari vengono rimosse le occorrenze doppie e tale cosa sarebbe andata a svantaggio dei dati in ingresso –può capitare che, infatti, che in una frase vi siano più parole identiche tra loro e con funzioni grammaticali diverse– è stata applicata la funzione `enumerate` a `parole`, in modo da associare un indice a ciascuna parola e non perdere alcuna ripetizione. Di conseguenza, indicizzando le parole, anche le parti del discorso non vengono perse se ripetute.
Ciò è stato scritto per questione di compattezza in una sola riga:

```
dictionary = OrderedDict(zip(enumerate(parole), partedisc))
```

Il dizionario così creato avrà, in output, una struttura simile alla seguente:

```
OrderedDict([((0, u'The'), 'Articolo'), ((1, u'cat'), 'Nome'), ((2,
u'is'), 'Verbo'), ((3, u'sleeping'), 'Verbo')])
```

Una volta costruito il dizionario ordinato, esso è stato *renderizzato* nel template di destinazione (*analisi_grammaticale.html*) insieme al titolo e alle altre variabili da utilizzare, come segue:

```
return render_template('analisi_grammaticale.html',title="Analizza
la frase!",raw=raw, lungfrase=lungfrase, dictionary=dictionary)
```

Quanto detto è tutto ciò che è stato necessario alla `view` per funzionare con il relativo template, del quale si parlerà più avanti nella sezione dedicata.

5.1.2 Genera_parola

La seconda route creata è `genera_parola`, insieme alla relativa funzione view. Questa parte del web server va a gestire il template *genera_parola.html* che contiene al suo interno l'attività che prevede la generazione di una parola casuale da sette categorie e ne permette la lettura e l'ascolto della pronuncia. I vocaboli sono stati estratti dal software per l'insegnamento dell'inglese *Impariamo la parola* di Rossana Cannavacciuolo, citato precedentemente.
Dopo la definizione della route, si è passati alla view `genera_parola`. In questo caso, non è stato necessario specificare quali metodi di HTTP usare, poiché questa parte dell'applicazione richiede unicamente `GET`, non essendoci `form` o altro nel template associato. In questa sezione viene usata `PlaintextCorpusReader`,

l'estensione per la lettura dei files di testo piano di cui si è parlato precedentemente. Sono state dunque create sette variabili con in associazione il relativo percorso al file di testo da aprire, analogamente al caso seguente, per ogni piccola collezione:

```
home=open('vocabolari/voc_home.txt','rU')
```

Successivamente, sono stati letti i dati presi in input con il comando `read`, come nel caso seguente:

```
rawh=home.read()
```

Il testo letto è stato poi diviso in token con il tokenizzatore di NLTK, precedentemente importato:

```
tes_tok_sh = nltk.word_tokenize(rawh)
```

ed è stato ripulito dagli underscore (sostituiti con gli spazi) inseriti tra i nomi composti nei files di testo per evitare che venissero segmentati durante la tokenizzazione, tramite l'espressione regolare con il seguente pattern:

```
(r'(\w*)_(\w*)',r'\1 \2')
```

inserita come argomento al metodo `.sub()` e applicata a ciascuna parola incontrata con tale occorrenza, in questo modo:

```
parole_home = [re.sub(r'(\w*)_(\w*)',r'\1 \2', w) for w in
tes_tok_sh]
```

Sono state poi definite le variabili contenenti la lunghezza di ogni lista creata, usando la funzione `len`, allo stesso modo della seguente:

```
lunhome=len(parole_home)
```

Dopo aver effettuato tutte le sostituzioni, le liste create con le parole lette in input sono state passate al template con il rendering, insieme alle lunghezze e al titolo:

```
return render_template('genera_parola.html',title="Genera, ascolta e
ripeti!",parole_home=parole_home,parole_food=parole_food,parole_numc
ol=parole_numcol,parole_cl=parole_cl,parole_pb=parole_pb,parole_sc=p
arole_sc,parole_cy=parole_cy,lunhome=lunhome,lunfood=lunfood,lunnumc
ol=lunnumcol,luncl=luncl,lunpb=lunpb,lunsc=lunsc,luncy=luncy)
```

5.1.3 Index e about

Dopo aver curato le parti precedenti, sono state create le route e le view per i template della homepage (*index.html*) e della pagina delle informazioni (*about.html*). Tali sezioni sono state trattate alla fine poiché, essenzialmente, necessarie solo al rendering dei template, in quanto (quasi) prive di contenuti da gestire sul server. Qui sotto sono riportate le due parti dette, nella loro –minimale– completezza:

Homepage:

```
@app.route('/')
def index():
      return render_template('index.html',title="Home")
```

Pagina delle informazioni:

```
@app.route('/about')
def about():
      return render_template('about.html', title="Informazioni")
```

5.2 I templates

Parallelamente alla realizzazione del web server *interactiveEnglish.py*, sono stati creati i template relativi a ciascuna route e view, più uno a parte per semplificare il caricamento degli stili, del menù e di Bootstrap in tutte le pagine (*layout.html*).
Per la descrizione di questi si seguirà l'ordine nel quale sono stati creati come fatto per la parte in Python.

5.2.1 Layout.html

Il primo template creato è stato *layout.html*, ovvero quello con al suo interno il necessario per la struttura e gli stili generali degli altri template.
Innanzitutto, è stato importato il file *base.html* di `flask_bootstrap`, una pagina HTML con all'interno la struttura minimale per una pagina web con Bootstrap, con la seguente riga in Jinja2:

```
{% extends "bootstrap/base.html" %}
```

È stato poi il momento di definire il blocco contenente il foglio di stile CSS (*styles.css*, riportato in appendice), contenente le regole da applicare a tutti i layout. Sebbene infatti Bootstrap da solo sia sufficiente a donare un buon aspetto alle pagine web, nello strumento realizzato vi sono sezioni che hanno bisogno di stili definiti a parte (come la parte dedicata all'analisi grammaticale). Per una ragione di sicurezza, nel blocco è stato collegato anche il foglio di stile di Bootstrap: tale operazione è stata effettuata in particolare per evitare problemi usufruendo dello strumento in modalità offline. Inoltre, è stato inserito un collegamento alla libreria jQuery (alla versione 3.1.1) per evitare di importarla in ogni pagina.
Il blocco è stato così strutturato:

```
{% block styles %}
{{super()}}
<link rel="stylesheet" href="{{url_for('.static',
filename='bootstrap.min.css')}}">
<link rel="stylesheet" href="{{url_for('.static',
filename='styles.css')}}">
<script src="{{ url_for('static', filename='jquery-
3.1.1.js') }}"></script>
{% endblock %}
```

Dopo la chiusura di tale sezione, si è passati alla creazione del `block title`, ovvero la sezione contenente il titolo della pagina:

```
{% block title %}
{{title}}
{% endblock %}
```

e a quello necessario al rendering della barra di navigazione:

```
{% block navbar %}
{{nav.mynavbar.render()}}
{% endblock %}
```

La creazione di questo template può sembrare superflua, poiché tali semplici dichiarazioni possono essere fatte in cima a tutti i template, ma è comoda per importare con il comando `extend` tali proprietà comuni a tutti con una semplice riga, come si potrà vedere, senza la necessità di andare a ridefinire il tutto.

5.2.2 Analisi_grammaticale.html

Questo template è quello con la struttura più complessa: contiene lo script che gestisce il gioco di abbinamenti della parola alla parte del discorso corrispondente. Come accennato nel capitolo 4, è necessario inserire gli script nei quali utilizzare contenuti ottenuti dal server direttamente nell'HTML e non in un file JavaScript separato, poiché questo linguaggio non riesce a interpretare la sintassi di Jinja2 al di fuori del template.

La prima operazione effettuata per costruire lo scheletro del template è stata quella di applicare quanto definito in `layout.html`, come segue:

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

Successivamente, è stato inserito un blocco per gli script con all'interno `{{super()}}` per evitare a quanto segue di essere sostituito con il contenuto di `layout.html`. All'interno del `block scripts` si è poi proceduto a importare jQuery UI con il relativo foglio di stile, la libreria di JavaScript indispensabile per le operazioni di spostamento e posizionamento previste dal gioco:

```
<script src="{{ url_for('static', filename='jquery-
ui.js') }}"></script>
<link rel="stylesheet" href="{{ url_for('static', filename='jquery-
ui.css') }}"></script>
```

Si è poi definito un `container`, ovvero la sezione nella quale inserire tutti i contenuti e, al suo interno, è stato inserito un form. Quest'ultimo si compone di una `textarea`, dove deve essere scritta la frase che si vuole analizzare e di un bottone di tipo `submit` per l'invio al server dei dati inseriti. L'interazione del form con il server è gestita tramite Jinja2, che con la sintassi `{{url_for()}}` indica quale view utilizzare per l'elaborazione dei dati:

```
<form role="form" action="{{ url_for('analisi_grammaticale') }}"
method="POST" >
      <div class="form-group">
      <textarea id="txt" class="form-control" rows="5"
      name="analisi_grammaticale" id="comment"></textarea>
      </div>
      <input type="submit" id="invia" class="btn btn-primary"
      value="Analizza!">
```

```
</form>
```

Il metodo `HTTP` utilizzato per l'invio della frase in input al server, come si può vedere in quanto riportato, è impostato come `POST`, così come nella corrispondente parte della view in Python.

Successivamente, è creata la struttura restante del template, racchiusa nel `div content`, e è richiamata la variabile `{{raw}}`, che contiene il testo della frase scritta in input una volta inviata al server per l'elaborazione.

È poi inserita una tabella (con Bootstrap) con due colonne e due *header* per le relative intestazioni: una per le parole e una per le rispettive parti del discorso.

```
<table class=" table table-bordered table-striped table-hover table-
condensed table-responsive">
  <thead>
    <tr>
      <th>
        <h2>Parole:</h2>
      </th>
      <th>
        <h2>Parte del discorso:</h2>
      </th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>
        <div id="origineParola"></div>
      </td>
      <td>
        <div id="destinazParteD"> </div>
      </td>
    </tr>
  </tbody>
</table>
```

In seguito, è chiuso il div `content` ed è creato un ulteriore div, `successo`, –nascosto di default– con all'interno un form, da essere mostrato una volta completato il gioco con successo, utile per resettare i campi e ricominciare l'attività (`init()` è la funzione di inizializzazione del gioco, come si vedrò in seguito):

```
<div id="successo">
  <h2>Ottimo lavoro!</h2>
```

```
    <form>
      <input class="MyButton" type="button" onclick="init();"
      value="Vuoi riprovare?"/>
      </form>
</div>
```

Al termine, è collocato il *footer*, uguale a quello presente in tutti i template.
La parte interattiva del template così strutturato è inserita in cima a esso, per evitare latenze nel caricamento. Essa si compone di uno script in JavaScript[59], inserito tra due tag `script`. Il suo funzionamento non è semplicissimo, ma è piuttosto intuitivo se si entra nel ragionamento; esso si basa su jQuery UI, un set di interazioni e plugin di jQuery per l'interfaccia utente.
La prima operazione effettuata nella creazione dello script è stata quella di inizializzare a zero una variabile per la verifica del numero degli abbinamenti parola-parte del discorso corretti:

```
var abbinamentiCorretti = 0;
```

È stata poi creata la funzione `init()`, che gestisce l'inizializzazione del gioco.
Per una questione di ordine sono, innanzitutto, definiti i parametri che gestiscono il comportamento del div `successo`, ovvero la finestra da essere mostrata al termine del gioco:

```
function init() {
  $('#successo').hide();
  $('#successo').css( {
    left: '580px',
    top: '250px',
    width: 0,
    height: 0
  } );
```

Come emerge da quanto riportato, è associato l'attributo `hide` al div al caricamento della pagina, poiché c'è bisogno di tenerlo nascosto fino a quando esso non deve essere utilizzato. Vi sono poi associate alcune regole in CSS, utilizzando jQuery.

59 *JQuery Essential Guide*, esempio drag e drop su slot numerati. Link al sito: http://www.elated.com/articles/drag-and-drop-with-jquery-your-essential-guide/.

Dopo le operazioni sul div `successo`, si è passati alla definizione di quanto necessario per il *reset* del gioco:

```
abbinamentiCorretti = 0;
$('#origineParola').html( '' );
$('#destinazParteD').html( '' );
```

`origineParola` e `destinazioneParteD` sono i div da essere generati dinamicamente, in questo punto inizializzati come vuoti tramite il metodo di jQuery `.html()` poiché si è nella sezione dedicata al reset. Tali div verranno inseriti nella tabella citata sopra, `origineParola` nella colonna delle parole e `destinazioneParteD` in quella delle parti del discorso. Come suggeriscono i nomi, il primo è il div popolato con la parola della frase sottomessa al server e è quanto deve essere poi trascinato sul secondo, il div con la parte del discorso, che funge da destinazione per i precedenti.

A questo punto, è trattata la parte sulla creazione e sul riempimento degli *slot* delle parole immesse e delle relative parti del discorso; in tale sezione sono usati gli strumenti forniti da jQuery UI per determinarne il comportamento.

Come visto in precedenza, in *interactiveEnglish.py* è creato il dizionario ordinato `dictionary` con le parole indicizzate per evitare la perdita di occorrenze; tale struttura è risultata essere comoda proprio poiché dotata di coppie chiave-valore e utile per la realizzazione di quanto segue. Innanzitutto, è stato creato un ciclo `for` su `dictionary` in Jinja2, con `.iteritems()` (per la scansione degli elementi chiave e valore), seguendo questa struttura:

```
{% for key, value in my_dict.iteritems() %}
    {{key}}
    {{value}}
{% endfor %}
```

Nel caso di `dictionary`, la chiave corrisponde alla parola (`parola`) e il valore alla parte del discorso (`pd`) a essa connessa:

```
{%for parola, pd in dictionary.iteritems()%}
{{parola}}
{{pd}}
{%endfor%}
```

A questo punto, è stato necessario utilizzare le chiavi e i valori ottenuti per creare i relativi div in due parti distinte: una per le parole e una per le parti del discorso. Per la generazione degli slot delle parole (trascinabili) al momento dell'invio dei dati al server, si è dovuto procedere come segue:

```
$("<div>"+"<tr>"+"{{parola[1]}}"+"</tr>"+"</div>").data( 'valore',
"{{pd}}").attr( 'id',
'card'+"{{pd}}" ).appendTo( '#origineParola' ).draggable( {
      containment: '#content',
      stack: '#origineParola div',
      cursor: 'move',
      revert: true,
      helper:"clone",
    } );
```

Come si può vedere, nella prima riga è definita, dopo l'alias `$` di jQuery, la parte da inserire dinamicamente negli slot delle parole, comprensiva di codice HTML. Da notare che non viene presa l'intera chiave `{{parola}}` ma `{{parola[1]}}` poiché, avendo applicato la funzione enumerate, essa contiene sia la parola che l'indice e dunque è necessario prendere solo ciò che vi è in posizione `[1]`. Successivamente, è utilizzato il metodo `.data()` per associarvi una coppia chiave-valore: `"valore"` e `{{pd}}`, per permettere poi gli abbinamenti. Infatti, fondamentalmente gli abbinamenti nel gioco sono valutati come corretti se e solo se la chiave contenuta in data corrisponde al relativo valore. Il metodo `.appendTo()`, invece, serve a indicare al programma dove inserire i dati processati: in questo caso andranno aggiunti in coda al div con id `origineParola`. `.draggable()`, invece, è una funzione di jQuery UI e serve a trasformare il div creato con il contenuto in un elemento trascinabile, al quale vengono applicate le proprietà inserite tra parentesi, alcune delle quali sono state trattate nel capitolo precedente. In `containment`, è inserito dove generare i div, in questo caso nel div `content`; `stack` indica il modo in cui impilare i dati generati (in questo caso nei div figli di `origineParola`); `cursor` serve a determinare lo stile del cursore nel momento del trascinamento (in questo caso settato come `move`, ovvero il simbolo della croce direzionale tipica di quando si sposta una finestra); `revert` offre all'elemento la possibilità di tornare al suo posto in caso di spostamento e ha come parametro un *booleano*; `helper`, in base al parametro, determina cosa

mostrare durante l'operazione di drag (in questo caso, `clone`, ovvero un clone del div trascinato).

Successivamente, sempre nel ciclo, è inserito quanto segue, per la creazione degli slot di destinazione con le parti del discorso:

```
$('<div class="pd">' + "<tr>"+ "{{pd}}" +'</tr>').data( 'valore',
      "{{pd}}").appendTo( '#destinazParteD' ).droppable( {
      accept: '#origineParola div',
      hoverClass: 'hovered',
      drop: handleCardDrop
    } );
```

Il funzionamento di tale porzione di codice è pressoché identica a quanto visto precedentemente per gli elementi draggable, con la sola differenza che in questo caso l'elemento viene creato come droppable, ovvero come allocazione di destinazione dei div draggable. In termini semplici, in questo modo vengono creati i div con le parti del discorso dove verranno allocate le parole. Il controllo della correttezza dell'abbinamento è gestito anche in questo caso dal metodo `.data()`. In `.droppable()`, invece, sono presenti le opzioni `accept` (per determinare cosa accettare, nel caso in esame il div draggable `origineParola` e div figli), `hoverClass` (per aggiungere la classe `hovered`, che cambia il colore della parte del discorso sulla quale si sta posizionando la parola) e `drop`, nella quale è contenuta l'invocazione alla funzione `handleCardDrop`, che gestisce cosa fare quando si verifica l'evento di drop.

A questo punto, per una questione di pulizia dell'interfaccia, vengono eliminate le parti del discorso duplicate, in quanto superflue poiché, nel momento del drop, le parole devono scomparire al loro interno e non sovrapporsi a esse, come si vedrà più avanti. Per fare ciò si è utilizzato jQuery che, agendo sulla classe `.pd` (dei div delle parti del discorso) con il selettore `gt()` e il metodo `.remove()`, rimuove tutte le occorrenze che presentano lo stesso testo:

```
$('.pd').each(function () {
     $('.pd:contains("' + $(this).text() + '"):gt(0)').remove();
});
```

Dato che, così strutturato, il gioco mancherebbe di senso (sarebbe normale associare la prima parola alla prima parte del discorso e così via), le parti del discorso sono state mescolate. Lo strumento, infatti, si pone l'obiettivo di creare una piccola challenge per testare le competenze acquisite dell'alunno in fase di apprendimento. Dunque, per il mescolamento dei div generati con le parti del discorso, si è proceduto come segue, utilizzando il metodo `.splice()` (per la rimozione e l'inserimento di elementi nella lista secondo regole definite) e le funzoni `Math.floor()` e `Math.random()` rispettivamente usate per ottenere un numero intero approssimato per difetto (come risultato del prodotto di `Math.random()` con `divs.length`) e per restituire gli elementi in ordine casuale:

```
$(function () {
    var parent = $("#destinazParteD");
    var divs = parent.children();
    while (divs.length) {
        parent.append(divs.splice(Math.floor(Math.random() *
        divs.length), 1)[0]);
    }
});
```

Dopo l'eliminazione delle parti del discorso doppie e il loro mescolamento, la funzione `init()` è stata chiusa, e ne è stata creata un'altra, `handleCardDrop`, chiamata nell'opzione `drop` associata all'elemento etichettato come droppable.

`HandleCardDrop` si serve di due parametri formali: `event` e `ui`; lo scopo della funzione è quello di definire cosa effettuare nel momento in cui un div draggable viene posizionato su uno droppable. Come detto in precedenza, le associazioni sono gestite in base alla parte del discorso inserita come valore nel metodo `.data()` dell'elemento draggable e di quello droppable: se esse sono confrontate e sono identiche tra loro, il matching può avvenire. La funzione per la gestione degli eventi da avviare al drop di un elemento draggable con la parola su uno slot droppable con la parte del discorso contiene quanto segue:

- le variabili nelle quali sono inseriti i valori inseriti in argomento al metodo `.data()` di entrambi i tipi di elemento:

```
var valoreParola = $(this).data( 'valore' );
var valoreParteD = ui.draggable.data( 'valore' );
```

- una serie di regole e stili da applicare agli slot delle parole in caso di successo nell'abbinamento (come per esempio l'aggiunta di classi, il cambio di dimensioni del font e il blocco del div draggable all'interno di quello droppable):

```
if ( valoreParola == valoreParteD) {
      ui.draggable.addClass( 'correct' );
      ui.draggable.toggle("fade",500, "nascondi" );
      ui.draggable.css("border", "none");
      ui.draggable.css("background", "none");
      ui.draggable.css("color", "green");
      ui.draggable.css("font-size", "25px")
      ui.draggable.draggable( 'disable' )
      ui.draggable.position( { of: $(this), my: 'center', at:
      'center' } );
      ui.draggable.draggable( 'option', 'revert', false );
```

- l'incremento della variabile `abbinamentiCorretti++`, che contiene il numero delle parole abbinate correttamente alle relative parti del discorso:

```
abbinamentiCorretti++;
```

- e, infine, un comando if per controllare se sono stati abbinate tutte le parole della frase ai relativi slot delle parti del discorso, dove `lungfrase` è una variabile definita nell'applicazione in Python che contiene la lunghezza del testo inserito in input:

```
if ( abbinamentiCorretti == {{lungfrase}}) {
       $('#successo').show();
       $('#successo').animate( {
            left: '35vw',
            top: '30vh',
            width: '300px',
            height: '250px',
            margin: 'auto',
            opacity: 1
      } );
$('#destinazParteD').css("visibility", "hidden");
$('#origineParola').css("visibility", "hidden");
$('#frase_raw').css("visibility", "hidden");
}
```

Da quanto riportato si può notare che, al verificarsi della condizione detta, viene mostrato il *popup* del messaggio di successo (preceduto da una breve animazione) con il bottone che invita a riprovare e vengono nascosti tutti i div draggable e droppable creati e il testo della frase inserita in input. Questa operazione deve essere effettuata poiché, sebbene nella funzione `init()` sia presente l'inizializzazione dei div vuoti (che, al momento del riavvio del gioco, fungono da reset), succede spesso che i dati inviati e ricevuti dal server restano in cache e non vengono rimossi dopo una semplice operazione di *refresh*.

Tali funzioni, inserite appropriatamente nel template, costituiscono il cuore della parte interattiva del gioco. In seguito, si parlerà dell'altra attività proposta e del relativo template.

5.2.3 Genera_parola.html

Genera_parola.html, il template collegato all'omonima funzione view, contiene l'attività secondaria contenuta nello strumento didattico, ovvero quella che prevede la generazione di una parola casuale da sette categorie di vocaboli e ne permette l'ascolto. Essa si propone come un esercizio di laboratorio da essere svolto insieme all'insegnante, che deve invitare gli alunni a ripetere la parola dopo la pronuncia da parte del sintetizzatore vocale e deve spronarli a migliorare in caso non riuscissero a imitarla correttamente. Si vedrà ora come è creato il template.

All'inizio di esso, è posto come di consueto il comando `extends` per inserire nella pagina le regole per il layout definite all'interno di layout.html. Sono poi create la `head` e il blocco degli script con `super()` allo stesso modo degli altri template. All'interno di `{%block script%}` è importata responsiveVoice, la libreria che offre la sintesi vocale della quale si è trattato nel capitolo precedente:

```
<script src="{{ url_for('static',
filename='responsivevoice.js') }}"></script>
```

È stato poi necessario dare una struttura alla pagina; sono così stati creati:

- un menù a tendina nel quale inserire come scelte le sette categorie dette più una vuota come scelta predefinita;

- un bottone per generare e ascoltare la parola casuale, collegato alla funzione `generaAscolta();`, in seguito definita;
- il paragrafo nel quale inserire in output la parola generata;
- le restanti componenti della struttura, pressoché identiche a quelle di *analisi_grammaticale.html* (container, footer, ecc).

come segue:

```
<div class="dropdown">
<h2>Categoria di vocaboli:</h2>
      <div class="categoria1wrap">
            <select id="selezionaCat">
            <option disabled selected value="SelCat"> -- Seleziona
                  una categoria -- </option>
                  <option value="Casa">Casa</option>
                  <option value="Cibo">Cibo</option>
                  <option value="Num_Col">Numeri e colori</option>
                  <option value="Vestiti">Vestiti</option>
                  <option value="Pa_Co">Parti del corpo</option>
                  <option value="Scuola">Scuola</option>
                  <option value="Citta">Città</option>
            </select>
      </div>
</br>
<!--definizione del bottone per la generazione e per l'ascolto della
parola-->
<button class="btn btn-default" id="checkbtn"
onclick="generaAscolta();" type="button">Genera e ascolta! </button>
<!--definizione del paragrafo di destinazione della parola
generata-->
<p id="parGen"></p>
```

A questo punto, si è tornati nella sezione degli script e si è lavorato sulla funzione `generaAscolta()`. Al suo interno è stata, prima di tutto, stabilita tramite *DOM* la destinazione dell'HTML alla quale associare quanto definito nella funzione: il menù `selezionaCat`. Sono state poi create otto variabili, ciascuna con all'interno la rispettiva lista richiamata con Jinja2 e una vuota per la voce predefinita del menù. Su queste ultime sono applicate le funzioni `Math.floor` e `Math.random` per permettere la generazione di una parola casuale dalle stesse:

```
var casa = {{parole_home | safe}}
[Math.floor(Math.random()*{{lunhome}})];
```

Per l'operazione di mescolamento, come evidente nel codice riportato, viene usata la variabile `{{lunghome}}` definita in Python, con la lunghezza della lista.
In ciascuna variabile è inoltre inserito il *filtro* di Jinja2 `safe`, che serve ad evitare problemi con i caratteri ASCII nel momento della ricezione dei dati sul client.
In seguito, è stata creata una cascata di condizioni `if` per determinare cosa fare quando viene selezionata una determinata categoria dal menù precedentemente creato, come illustrato nel caso seguente:

```
if(categoria.value=="Casa"){
      document.getElementById("parGen").innerHTML = "<h2>"+"La
      parola è: <p class='par'>" +casa +"</p></h2>";
      responsiveVoice.speak(casa);
}
```

Nella porzione di codice riportata è utilizzata la proprietà DOM `innerHTML` per definire come e con cosa popolare il paragrafo di output alla selezione di una determinata categoria dal menù. All'interno di ciascun controllo è utilizzata responsiveVoice che, tramite il metodo `.speak()`, va a riprodurre quanto inserito tra parentesi. Nel caso visto, al click del pulsante trigger della funzione ("Genera e ascolta!"), viene letta ad alta voce dal sintetizzatore vocale una parola del vocabolario sulla casa generata al momento (ad esempio, "*kitchen*").
Essenzialmente, è questo il funzionamento della parte interattiva della seconda attività. Per una maggiore accuratezza, è consigliabile essere collegati alla rete per usufruire di tutta la potenza delle voci personalizzate di responsiveVoice.

5.2.4 Index.html

Index.html è il template della homepage dello strumento, collegato alla relativa view. La sua struttura è molto semplice e unicamente creata con Bootstrap. Gli unici elementi presenti sono, oltre al menù generale, un'intestazione e due pulsanti che portano rispettivamente all'attività "*Analisi* grammaticale" e a *"Genera, ascolta e ripeti!"*. I pulsanti hanno nell'`href` il collegamento alle relative view:

```
<ul class="list-inline intro-social-buttons">
 <li>
      <a href="{{ url_for('analisi_grammaticale') }}" class="btn
      btn-default btn-lg bottoni-home">Analisi grammaticale</a>
```

```
</li>
<li>
     <a href="{{ url_for('genera_parola') }}" class="btn btn-
     default btn-lg bottoni-home">Genera, ascolta e ripeti!</a>
</li>
</ul>
```

Al termine, vi è il footer, come di consueto. In tale pagina non sono importati static file come script o librerie di JavaScript.

5.2.5 About.html

Infine, l'ultimo template realizzato è *about.html*, ovvero quello che va a costituire la pagina contenente le informazioni in sintesi sul progetto e sull'autore. È in assoluto il template con la struttura più elementare, in quanto non contiene script.

È impiegato Jinja2 soltanto per richiamare alcune immagini, come nel caso seguente:

```
<img class="img-circle img-responsive img-center"
src="{{ url_for('static', filename='img/imginfo.jpg') }}"
alt="immagine info">
```

Tutti i div e le sezioni sono definite usando Bootstrap.

Non ci si sofferma molto su tale template poiché dalla struttura molto semplice; sarà pertanto disponibile in appendice per la consultazione insieme a tutto il codice.

5.3 Incapsulamento e testing

Dopo la creazione parallela del web server e dei templates, è stato affinato l'aspetto grafico, andando a ritoccare alcune regole in *styles.css* e aggiungendo eventuali immagini.

Al termine del progetto, il tutto è stato caricato su *Bitbucket* e scaricato sulla macchina virtuale di VirtualBox *Interactive_English* tramite il comando `git clone`, come spiegato nel capitolo 2. La macchina è stata poi spenta, esportata in formato OVA, importata in VirtualBox dopo aver distrutto quella preesistente e ne è stato testato l'effettivo funzionamento.

Si è così ottenuto un prodotto finito semplice da utilizzare grazie alla strategia dell'incapsulamento in macchina virtuale, pronto per essere distribuito.

Conclusioni e sviluppi futuri

Lo strumento è stato realizzato, come si è visto, per offrire un'attività interattiva da essere svolta durante le ore di lingua inglese nelle classi di scuola elementare. I principali obiettivi sono stati: fornire un software multipiattaforma utilizzabile senza connessione di rete, indipendente dal sistema *host*, facile da installare e da consultare e, allo stesso tempo, accattivante.

È così nata l'idea di creare qualcosa che andasse a riempire tutte le lacune causate dalle carenze riportate precedentemente e un metodo risolutivo si è trovato nella realizzazione di un software fruibile su ogni tipo di macchina compatibile con il programma open source Oracle VM VirtualBox, per la virtualizzazione. Per semplificare l'installazione del tutto sul sistema, è stato dunque scelto di incapsulare lo strumento all'interno di una macchina virtuale e di distribuirlo in formato applicazione virtuale (OVA), importabile in un solo passaggio all'interno del software di virtualizzazione VirtualBox. Così facendo, si va non solo a creare uno strumento non intaccabile (poiché l'utente medio non ha la possibilità di rovinare la parte server, che contiene il necessario al funzionamento), ma si semplifica anche il lavoro dell'addetto alla preparazione del computer che deve accogliere il software. Il prodotto è infatti mirato a essere, in primis, semplice da far funzionare: se il tutto non fosse incapsulato all'interno dell'OVA bisognerebbe installare tutti i requisiti incorporati nella macchina virtuale. Tale operazione non solo toglierebbe molto tempo, ma rischierebbe comunque di non essere alla portata del personale addetto: non rientra solitamente nelle competenze di un tecnico di laboratorio saper installare Python, Flask, NLTK e le altre dipendenze richieste.

Inoltre, lo strumento didattico è stato ideato come una sorta di "trampolino di lancio" verso il mondo dell'utilizzo del *Natural Language Processing* nelle scuole italiane, come mezzo di analisi automatica del linguaggio. Pur non essendo NLTK una *suite* che offre un'elaborazione del testo precisa come quella di un essere umano, è comunque uno strumento molto potente e che può sicuramente trovare buon impiego nel campo dell'educazione se correttamente utilizzato, magari sotto la continua

supervisione di un insegnante per scongiurare eventuali errori dell'interprete automatico.
Quanto è stato realizzato verrà testato entro la fine dell'anno scolastico con un gruppo di alunni della "Scuola Primaria De Amicis" sotto la guida dalla docente di inglese Cinzia di Marco ed è stato inoltre caricato online, per semplificarne la distribuzione[60]. Lo strumento potrà essere, infine, integrato con altri giochi o esercizi interattivi che prevedono l'impiego di NLTK per avere una gamma più ampia di attività da poter svolgere durante le ore di lezione.

Come visto, il software è stato creato per essere fruibile su postazioni non dotate di connessione a Internet e per essere utilizzato necessita di essere scaricato (da un computer con accesso alla rete), importato in VirtualBox e avviato. Un possibile sviluppo potrebbe, però, essere quello di rendere accessibile il tutto direttamente online, come un'applicazione web. Ciò sarebbe comodo in caso si volesse consultare lo strumento ovunque, senza dover installare o importare nulla nel sistema. Una soluzione efficace sarebbe quella della sua implementazione in un servizio *PaaS* (*Platform as a Service*), ovvero in una piattaforma che offre uno spazio online per la realizzazione e la distribuzione di applicazioni. Quella più completa e diffusa è *Heroku*, una piattaforma di *cloud* nata nel 2007 che consta di centinaia di migliaia di utenti in tutto il mondo, supporta ben sei linguaggi di programmazione (*Java*, *Node.js*, *Scala*, *Clojure*, *Python* e *PHP*) e si serve dei sistemi operativi *Debian* e Ubuntu. Essa permette agli sviluppatori di creare, gestire e distribuire applicazioni direttamente online ed è ben integrata con Git: ciò consente agli utenti di sviluppare un'app su una piattaforma esterna e poi importare il progetto in Heroku, che servirà a renderla funzionante. L'obiettivo del PaaS è quello di fornire agli utenti quanto necessario alla distribuzione e all'esecuzione nel cloud delle applicazioni; tali strumenti agevolano gli sviluppatori, che non hanno bisogno di avere

60 *Interactive English Ova*, link al repository: https://bitbucket.org/c_zamberti/interactive_english_ova.
Il prodotto è disponibile anche su Google Drive per permetterne lo scaricamento in maniera semplice. Link all'applicazione: https://drive.google.com/open?id=0B6ELqs6wBndQLXRzSEp2YTdkaWc

un'infrastruttura informatica adeguata a supporto di quanto creato, poiché a coprire queste esigenze ci sarà la piattaforma. Heroku include un sistema centrale per l'allocazione delle risorse con un pannello di monitoraggio (figura 13) e si avvale di container di funzioni detti *dyno,* per l'esecuzione di un singolo comando (processo *web*, *worker* o altro).

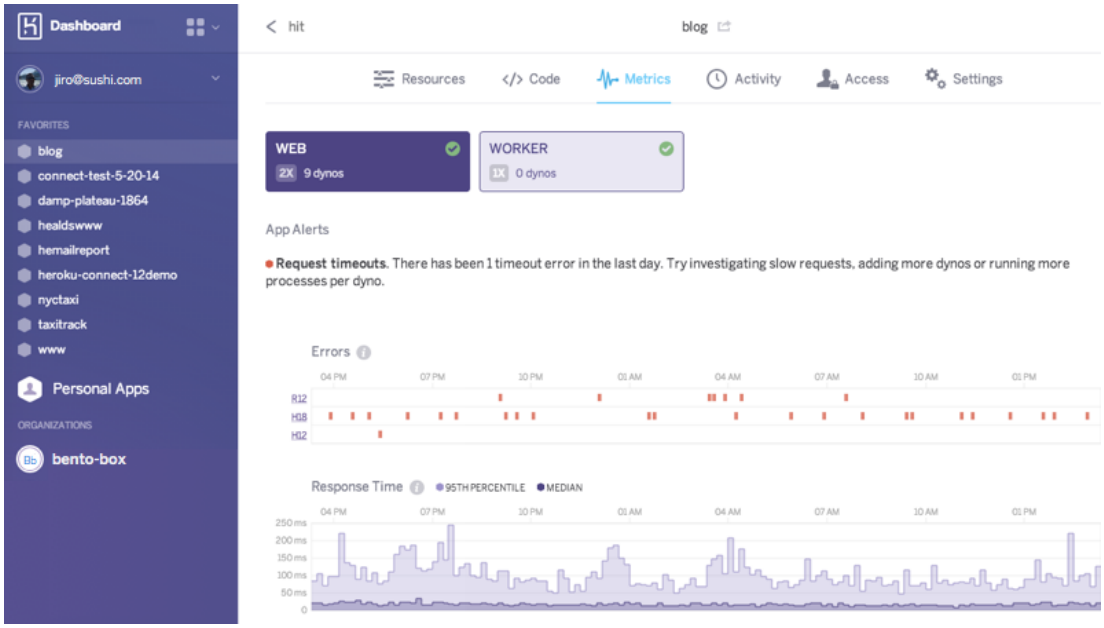

Figura 13. Pannello di monitoraggio del sistema allocazione risorse di Heroku

A tale piattaforma si è solo accennato poiché non rientra negli attuali progetti di sviluppo dello strumento didattico[61]; tuttavia essa potrà essere utile in futuro se si vorrà renderlo totalmente esente da vincoli di prerequisiti su macchina (es: VirtualBox) e facilmente accessibile anche da smartphone o tablet (con *Android*, *iOS* e altri OS per dispostivi mobili). Per rendere ciò possibile, però, dovrà essere soddisfatto il requisito della rete: tutti gli apparati multimediali nelle scuole dovranno essere dotati di connessione –ovviamente– per poter accedere a un'applicazione web; tale realtà è purtroppo ancora poco vicina ma si spera nei prossimi anni di

61 Per maggiori informazioni su Heroku, consultare: Middleton, Schneeman. 2013. *Heroku Up & Running: Effortless application deployment and scaling*, Massachusetts, O'REILLY Media e il link: http://www.fastweb.it/web-e-digital/cos-e-e-come-funziona-heroku-piattaforma-cloud-per-app/.

raggiungere tale obiettivo con le operazioni di modernizzazione del sistema di istruzione previste dal Piano Operativo Nazionale del Miur.

Appendice

Di seguito è inserito il codice relativo allo strumento didattico creato. Ogni sezione sarà identificata dal titolo del file contenente il codice riportato.

Verranno escluse dall'appendice le librerie di JavaScript e di Python poiché scaricabili dai relativi siti web e tramite `pip`. Sono inoltre riportati i piccoli vocabolari utilizzati nell'attività "Genera, ascolta e ripeti!".

Web Server

interactiveEnglish.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#importazione delle librerie e delle classi necessarie
from flask import Flask, render_template, request
from flask_bootstrap import Bootstrap
from flask_nav import Nav
from flask_nav.elements import Navbar, View
import re
import nltk
from nltk.corpus import PlaintextCorpusReader
import collections
from collections import OrderedDict
#creazione di un'istanza dell'applicazione
app = Flask(__name__)
#inizializzazione di Bootstrap
Bootstrap(app)
#creazione dell'oggetto per il menù di navigazione
nav = Nav()
#definizione della route e della funzione view per il menù di
navigazione
@nav.navigation()
def mynavbar():
  return Navbar(
      View('Home', 'index'),
      View('Analisi grammaticale', 'analisi_grammaticale'),
      View('Genera, ascolta e ripeti!', 'genera_parola'),
      View('Informazioni', 'about')
  )
nav.init_app(app)

#definizione della route e della funzione view per la homepage
@app.route('/')
def index():
      return render_template('index.html',title="Home")
```

```
#definizione della route e della funzione view per il gioco
dell'analisi grammaticale
@app.route('/analisi_grammaticale', methods=['GET', 'POST'])
#definizione dei metodi
def analisi_grammaticale():
      #definizione di variabili e liste vuote
      raw, lungfrase="", 0
      postag, dictionary={}, {}
      parole, partedisc=[], []
      if request.method == 'POST':
            raw = request.form.get('analisi_grammaticale')
            nltk.data.path.append('./nltk_data/')
            #definizione del percorso della cartella nltk_data
            testo_tokenizzato = nltk.word_tokenize(raw)
            nonPunct = re.compile('.*[A-Za-z0-9].*')
            #definizione dell'espressione regolare per escludere la
            punteggiatura
            tes_tok_nopunt = [w for w in testo_tokenizzato if
            nonPunct.match(w)]#parole senza punteggiatura
            lungfrase = len(tes_tok_nopunt)
            #creazione della variabile con il testo annotato
            postag=nltk.pos_tag(tes_tok_nopunt)
            #creazione delle due liste nelle quali inserire la
            parola e il pos corrispondente
            for n in postag:
                  parole.append(n[0])
                  partedisc.append(n[1])

            #cambio le etichette delle parti del discorso in
            stringhe più "leggibili"
            for n,i in enumerate(partedisc):
                  if i=="CC":
                        partedisc[n]="Congiunzione"
                  if i=="CD":
                        partedisc[n]="Numero"
                  if i=="DT":
                        partedisc[n]="Articolo"
                  if i=="EX":
                        partedisc[n]="'There' esistenziale"
                  if i=="FW":
                        partedisc[n]="Parola straniera"
                  if i=="IN":
                        partedisc[n]="Preposizione"
                  if i=="JJ":
                        partedisc[n]="Aggettivo"
                  if i=="JJR":
                        partedisc[n]="Aggettivo"
```

```
                    if i=="JJS":
                          partedisc[n]="Aggettivo"
                    if i=="LS":
                          partedisc[n]="Simbolo"
                    if i=="MD":
                          partedisc[n]="Modale"
                    if i=="NN":
                          partedisc[n]="Nome"
                    if i=="NNS":
                          partedisc[n]="Nome"
                    if i=="NNP":
                          partedisc[n]="Nome"
                    if i=="NNPS":
                          partedisc[n]="Nome"
                    #aggettivo indefinito, ma talvolta pr. possessivo
                    if i=="PDT":
                          partedisc[n]="Aggettivo"
                    if i=="POS":
                          partedisc[n]="Possessivo"
                    if i=="PRP":
                          partedisc[n]="Pronome"
                    if i=="PRP$":
                          partedisc[n]="Pronome"
                    if i=="RB":
                          partedisc[n]="Avverbio"
                    if i=="RBR":
                          partedisc[n]="Avverbio"
                    if i=="RBS":
                          partedisc[n]="Avverbio"
                    if i=="RP":
                          partedisc[n]="Particella"
                    if i=="SYM":
                          partedisc[n]="Simbolo"
                    if i=="TO":
                          partedisc[n]="'To' per infinito"
                    if i=="UH":
                          partedisc[n]="Esclamazione"
                    if i=="VB":
                          partedisc[n]="Verbo"
                    if i=="VBD":
                          partedisc[n]="Verbo"
                    if i=="VBG":
                          partedisc[n]="Verbo"
                    if i=="VBN":
                          partedisc[n]="Verbo"
                    if i=="VBP":
                          partedisc[n]="Verbo"
                    if i=="VBZ":
```

```
                        partedisc[n]="Verbo"
                  if i=="WDT":
                        partedisc[n]="Avverbio"
                  if i=="WP":
                        partedisc[n]="Pronome"
                  if i=="WP$":
                        partedisc[n]="Pronome"
                  if i=="WRB":
                        partedisc[n]="Avverbio"

      #compressione delle due liste con i relativi indici in un
      dizionario ordinato
      #Sfruttando enumerate, si ottengono tutte le occorrenze anche
      se doppie
      dictionary = OrderedDict(zip(enumerate(parole), partedisc))

      return
      render_template('analisi_grammaticale.html',title="Analizza la
      frase!",raw=raw, lungfrase=lungfrase, dictionary=dictionary)

#definizione della route e della funzione view per l'attività di
sorting della parola casuale
@app.route('/genera_parola')
def genera_parola():
      # apertura del vocabolario sulla casa
      home=open('vocabolari/voc_home.txt','rU')
      # apertura del vocabolario sul cibo
      food=open('vocabolari/voc_food.txt','rU')
      # apertura del vocabolario sui numeri e colori
      numcol=open('vocabolari/voc_num_col.txt','rU')
      # apertura del vocabolario sui vestiti
      clothes=open('vocabolari/voc_clothing.txt','rU')
      # apertura del vocabolario sulle parti del corpo
      part_body=open('vocabolari/voc_pobody.txt','rU')
      # apertura del vocabolario sulla scuola
      school=open('vocabolari/voc_school.txt','rU')
      # apertura del vocabolario sulla città
      city=open('vocabolari/voc_city.txt','rU')

      #lettura i dati in input
      rawh=home.read()
      rawf=food.read()
      rawnc=numcol.read()
      rawcl=clothes.read()
      rawpb=part_body.read()
      rawsc=school.read()
      rawcy=city.read()
```

```
        #divisione del testo in token
        tes_tok_sh = nltk.word_tokenize(rawh)
        tes_tok_sf = nltk.word_tokenize(rawf)
        tes_tok_snc = nltk.word_tokenize(rawnc)
        tes_tok_scl = nltk.word_tokenize(rawcl)
        tes_tok_spb = nltk.word_tokenize(rawpb)
        tes_tok_sco = nltk.word_tokenize(rawsc)
        tes_tok_cy = nltk.word_tokenize(rawcy)

        #sostituzione dello spazio all'underscore nelle parole
        composte e creazione delle liste con le parole per ogni
        categoria
        parole_home = [re.sub(r'(\w*)_(\w*)',r'\1 \2', w) for w in
        tes_tok_sh]
        parole_food = [re.sub(r'(\w*)_(\w*)',r'\1 \2', w) for w in
        tes_tok_sf]
        parole_numcol = [re.sub(r'(\w*)_(\w*)',r'\1 \2', w) for w in
        tes_tok_snc]
        parole_cl = [re.sub(r'(\w*)_(\w*)',r'\1 \2', w) for w in
        tes_tok_scl]
        parole_pb = [re.sub(r'(\w*)_(\w*)',r'\1 \2', w) for w in
        tes_tok_spb]
        parole_sc = [re.sub(r'(\w*)_(\w*)',r'\1 \2', w) for w in
        tes_tok_sco]
        parole_cy = [re.sub(r'(\w*)_(\w*)',r'\1 \2', w) for w in
        tes_tok_cy]

        #definizione della lunghezza delle liste con le parole
        lunhome=len(parole_home)
        lunfood=len(parole_food)
        lunnumcol=len(parole_numcol)
        luncl=len(parole_cl)
        lunpb=len(parole_pb)
        lunsc=len(parole_sc)
        luncy=len(parole_cy)

        return render_template('genera_parola.html',title="Genera,
        ascolta e ripeti!",
        parole_home=parole_home,parole_food=parole_food,
        parole_numcol=parole_numcol,parole_cl=parole_cl,
        parole_pb=parole_pb,parole_sc=parole_sc,parole_cy=parole_cy,
        lunhome=lunhome,lunfood=lunfood,lunnumcol=lunnumcol,
        luncl=luncl,lunpb=lunpb,lunsc=lunsc,luncy=luncy)

#definizione della route e della view della pagina delle
informazioni
@app.route('/about')
def about():
```

```
      return render_template('about.html', title="Informazioni")

if __name__ == "__main__":
      app.run(debug=True, host="0.0.0.0")
```

Templates

about.html

```
{% extends "layout.html" %}
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Informazioni</title>
</head>
<body>
{%block content%}
<!--definizione dell'intestazione-->
      <!-- Page Content -->
      <div class="container">
            <!-- creazione del testo introduttivo -->
            <div class="row">
                  <div class="col-lg-12">
                        <h2 class="page-header titolo">Lo strumento
                        didattico</h2>
                        <p class="info">Lo strumento didattico
                        proposto è stato realizzato con lo scopo di
                        fornire un'attività fruibile su supporti
                        multimediali durante le lezioni di inglese
                        nelle scuole primarie.<br/>
                        Il progetto è frutto di uno studio
                        sull'implementazione in modo semplice di
                        metodi di analisi automatica del linguaggio
                        nella didattica ed è stato creato da Chiara
                        Zamberti, studentessa del corso triennale di
                        Informatica Umanistica presso l'Università
                        degli Studi di Pisa, come elaborato di
                        laurea.
                        <br/>
                        Esso si propone come uno strumento semplice
                        da utilizzare, diviso in due attività,
                        "Analisi grammaticale" e "Ascolta e ripeti"
                        da svolgere con l'aiuto del docente o
                        individualmente, dedicate rispettivamente
                        alle classi terze, quarte e quinte e alle
                        prime e alle seconde.
                        <br/></p>
                  </div>
```

```
            </div>
            <!-- definizione del div con le info sull'autore -->
            <div class="row">
                  <div class="col-lg-12">
                        <h2 class="page-header titolo">L'autore</h2>
                        <h3 class="centrato">Chiara Zamberti</h3>
                        <br/>
                  </div>
                        <img class="img-circle img-responsive img-
                        center" src="{{ url_for('static',
                        filename='img/imginfo.jpg') }}"
                        alt="immagine info">
                        <br/>
                  </div>
            <div class="row">
                  <div class="col-lg-12">
                        <p class="info">Studentessa di Informatica
            Umanistica presso l'Univeristà degli Studi di Pisa, 23
            anni, appassionata di tecnologia e di recente
            affascinata dal mondo dell'e-learning. </p>
                  <div class="row">
                        <h2 class="page-header titolo">Contatti</h2>
                        <p class="centrato">
                        <a href="https://www.facebook.com/
                        chiara.zamberti.1296520668">
                        <img src="{{ url_for('static',
                        filename='img/email.png') }}"></a>
                        <a href=
                        "mailto:chiara.zamberti@gmail.com"><img
                        src="{{ url_for('static',
                        filename='img/facebook.png') }}"></a>
                        </p>
                  </div>
            </div>
            </div>
            </div>
            <!--definizione del footer-->
            <br/>
            <footer>
                  <div class="row">
                        <div class="col-lg-12">
                              <p>Copyright &copy; Chiara Zamberti.
                              2017 - All Rights Reserved</p>
                        </div>
                  </div>
            </footer>
</body>
{% endblock %}
```

```
</html>
```

analisi_grammaticale.html

```
{% extends "layout.html" %}
<!DOCTYPE html>
<html>
<head>
{% block scripts %}
{{ super() }}
<script type="text/javascript">
//inizializzo la variabile che conta gli abbinamenti della parola
alla parte del discorso corretti
var abbinamentiCorretti = 0;
$( init );
function init() { //funzione per l'inizializzazione dell'attività
      $('#successo').hide(); //nasconde il messaggio di successo
      (che deve essere mostrato alla fine del gioco)
      $('#successo').css( { //stili per il messaggio di successo
            left: '580px',
            top: '250px',
            width: 0,
            height: 0
      } );
      // Reinizializzazione della variabile per gli abbinamenti e
      svuotamento html per il reset
      abbinamentiCorretti = 0;
      $('#origineParola').html( '' );
      $('#destinazParteD').html( '' );
//ciclo Jinja2 per iterare sulla struttura (dizionario ordinato) con
all'interno le coppie chiave-valore parola-parte del discorso

{%for parola, pd in dictionary.iteritems()%}
//riempimento dei div delle parole, impostazione dell'attributo
necessario all'associazione e definizione delle regole per il
dragging con jQueryUI
$("<div>"+"<tr>"+"{{parola[1]}}"+"</tr>"+"</div>").data( 'valore',
"{{pd}}").attr( 'id','card'+"{{pd}}" ).appendTo( '#origineParola' ).
draggable( {      containment: '#content',
                  stack: '#origineParola div',
                  cursor: 'move',
                  revert: true,
                  helper:"clone",
            } );

//riempimento dei div di destinazione delle parti del discorso,
impostazione dell'attributo necessario all'associazione e
definizione delle regole per il dragging con jQueryUI
```

```
$('<div class="pd">' + "<tr>"+ "{{pd}}" +'</tr>').data( 'valore',
"{{pd}}").appendTo( '#destinazParteD' ).droppable( {
                  accept: '#origineParola div',
                  hoverClass: 'hovered',
                  drop: handleCardDrop
            } );

{%endfor%}

//rimuove i duplicati delle parti del discorso
$('.pd').each(function () {
                  $('.pd:contains("' + $(this).text() +
'"):gt(0)').remove();
});

//funzione per il mescolamento del secondo gruppo di div (relativi
alle parti del discorso)
$(function () {
            var parent = $("#destinazParteD");
            var divs = parent.children();
            while (divs.length) {

      parent.append(divs.splice(Math.floor(Math.random() *
      divs.length), 1)[0]);
            }
});
}
//funzione per la gestione del drag e del drop dei div
function handleCardDrop( event, ui ) {
      var valoreParola = $(this).data( 'valore' );
      var valoreParteD = ui.draggable.data( 'valore' );
      //controllo per il comportamento in caso di abbinamento
      effettuato con successo
      if ( valoreParola == valoreParteD) {
            ui.draggable.addClass( 'correct' );
            ui.draggable.toggle("fade",500, "nascondi" );
            ui.draggable.css("border", "none");
            ui.draggable.css("background", "none");
            ui.draggable.css("color", "green");
            ui.draggable.css("font-size", "25px")
            ui.draggable.draggable( 'disable' )
            ui.draggable.position( { of: $(this), my: 'center', at:
            'center' } );
            ui.draggable.draggable( 'option', 'revert', false );
            abbinamentiCorretti++;
      }
      //controllo per mostrare la finestra di successo in caso di
      abbinamento di tutte le parole alle parti del discorso
```

```
      if ( abbinamentiCorretti == {{lungfrase}}) {
            $('#successo').show();
            $('#successo').animate( {
            //animazione alla comparsa della finestra
                  left: '35vw',
                  top: '30vh',
                  width: '300px',
                  height: '250px',
                  margin: 'auto',
                  opacity: 1
            } );
            //svuotamento dei div alla pressione del pulsante
            "Riprovare?" nella finestra di successo
            $('#destinazParteD').css("visibility", "hidden");
            $('#origineParola').css("visibility", "hidden");
            $('#frase_raw').css("visibility", "hidden");
      }

}
</script>
<!--importazione degli script-->
<script src="{{ url_for('static', filename='jquery-
ui.js') }}"></script>
<link rel="stylesheet" href="{{ url_for('static', filename='jquery-
ui.css') }}"></script>
</head>
<body>
<div class="container">
<!--definizione dell'intestazione e del form-->
      <h2 class="page-header titolo">Analisi Grammaticale</h2>
      <p class="info">Scrivi una frase e poi premi il pulsante
      "Analizza!". Associa poi la parola alla parte del discorso
      corrispondente.</p>
      <!--definizione della view di destinazione e del metodo di
      invio dei dati al server-->
      <form role="form"
      action="{{ url_for('analisi_grammaticale') }}" method="POST" >
            <div class="form-group">
                  <textarea id="txt" class="form-control" rows="5"
                  name="analisi_grammaticale"
                  id="comment"></textarea>
            </div>
            <input type="submit" id="invia" class="btn btn-primary"
            value="Analizza!">
      </form>
<!--definizione del contenuto-->
<div id="content">
<!--output della frase inserita-->
```

```
<h2>
Frase:
<span id="frase_raw">
{{raw}}
</span>
</h2>
</br>
<!--tabella con i div delle parole e delle parti del discorso-->
<table class=" table table-bordered table-striped table-hover table-
condensed table-responsive">
      <thead>
            <tr>
                  <th>
                        <h2>Parole</h2>
                  </th>
                  <th>
                        <h2>Parti del discorso</h2>
                  </th>
            </tr>
      </thead>
      <tbody>
            <tr>
                  <td>
                  <!--definizione del div per le parole -->
                        <div id="origineParola"></div>
                  </td>
                  <td>
                  <!--definizione del div per le parti del discorso
                  -->
                        <div id="destinazParteD"> </div>
                  </td>
            </tr>
      </tbody>
</table>
</div>
      <!--div del messaggio di successo -->
      <div id="successo">
            <h2>Ottimo lavoro!</h2>
            <form>
            <input class="MyButton" type="button" onclick="init();"
            value="Vuoi riprovare?"/>
            </form>
      </div>
</div>
<!--definizione del footer-->
      <footer>
                  <div class="row">
                              <div class="col-lg-12">
```

```
                                    <p>Copyright &copy; Chiara
                                    Zamberti. 2017 - All Rights
                                    Reserved</p>
                                </div>
                    </div>
      </footer>
</body>
{% endblock %}
</html>
```

genera_parola.html

```
{% extends "layout.html" %}
<!DOCTYPE html>
<html>
<head>
{% block scripts %}
{{ super() }}
<!--importazione dello script per il funzionamento di
ResponsiveVoice-->
<script src="{{ url_for('static',
filename='responsivevoice.js') }}"></script>
<script language="javascript">
//creazione della funzione per gestire il contenuto della select e
avviare la sintesi vocale
function generaAscolta(){
//individuazione del div della select
var categoria=document.getElementById("selezionaCat");
//definizione delle variabili delle categorie di vocaboli dalle
quali trarre un vocabolo a caso
//è usato il filtro di Jinja2 "safe" per evitare problemi con i
caratteri unicode

var cibo={{parole_food | safe}}
[Math.floor(Math.random()*{{lunfood}})];
var num_col={{parole_numcol | safe}}
[Math.floor(Math.random()*{{lunnumcol}})];
var vestiti={{parole_cl | safe}}
[Math.floor(Math.random()*{{luncl}})];
var pa_co={{parole_pb | safe}}[Math.floor(Math.random()*{{lunpb}})];
var scuola={{parole_sc | safe}}
[Math.floor(Math.random()*{{lunsc}})];
var citta={{parole_cy | safe}}[Math.floor(Math.random()*{{luncy}})];
//controlli per la generazione di una parola a seconda della
categoria selezionata e per la relativa pronuncia con
ResponsiveVoice
      if(categoria.value=="SelCat"){ //categoria vuota
```

```
            document.getElementById("parGen").innerHTML =
"<h2>"+"Seleziona prima una categoria</h2>";
 }
      if(categoria.value=="Casa"){
            document.getElementById("parGen").innerHTML = "<h2>"+"La
      parola è: <p class='par'>" +casa +"</p></h2>";
            responsiveVoice.speak(casa);
 }
      if(categoria.value=="Cibo"){
            document.getElementById("parGen").innerHTML = "<h2>"+"La
      parola è: <p class='par'>" + cibo+"</p></h2>";
            responsiveVoice.speak(cibo);
 }
      if(categoria.value=="Num_Col"){
            document.getElementById("parGen").innerHTML = "<h2>"+"La
      parola è: <p class='par'>" + num_col+"</p></h2>";
            responsiveVoice.speak(num_col);
 }
      if(categoria.value=="Vestiti"){
            document.getElementById("parGen").innerHTML = "<h2>"+"La
      parola è: <p class='par'>" + vestiti+"</p></h2>";
            responsiveVoice.speak(vestiti);
 }
      if(categoria.value=="Pa_Co"){
            document.getElementById("parGen").innerHTML = "<h2>"+"La
      parola è: <p class='par'>" +pa_co+"</p></h2>";
            responsiveVoice.speak(pa_co);
 }
      if(categoria.value=="Scuola"){
            document.getElementById("parGen").innerHTML = "<h2>"+"La
      parola è: <p class='par'>" + scuola+"</p></h2>";
            responsiveVoice.speak(scuola);
 }
      if(categoria.value=="Citta"){
            document.getElementById("parGen").innerHTML = "<h2>"+"La
      parola è: <p class='par'>" + citta+"</p></h2>";
            responsiveVoice.speak(citta);
 }

}

</script>
</head>
<body>
<div class="container">
<!--definizione dell'intestazione-->
      <h2 class="page-header titolo">Genera, ascolta e ripeti!</h2>
```

```
      <p class="info">Genera una parola dalla categoria che
preferisci, ascolta come si pronuncia e ripeti.</p>
</br>
<!--definizione del definizione della select-->
<div class="dropdown">
<h2>Categoria di vocaboli:</h2>
      <div class="categoria1wrap">
            <select id="selezionaCat">
            <option disabled selected value="SelCat"> -- Seleziona
                  una categoria -- </option>
                  <option value="Casa">Casa</option>
                  <option value="Cibo">Cibo</option>
                  <option value="Num_Col">Numeri e colori</option>
                  <option value="Vestiti">Vestiti</option>
                  <option value="Pa_Co">Parti del corpo</option>
                  <option value="Scuola">Scuola</option>
                  <option value="Citta">Città</option>
            </select>
      </div>
</br>
<!--definizione del bottone per la generazione e per l'ascolto della
parola-->
<button class="btn btn-default" id="checkbtn"
onclick="generaAscolta();" type="button">Genera e ascolta! </button>
<!--definizione del paragrafo di destinazione della parola
generata-->
<p id="parGen"></p>

<br/>
</div>
</div>
<!--definizione del footer-->
      <footer>
                  <div class="row">
                              <div class="col-lg-12">
                                    <p>Copyright &copy; Chiara
                                    Zamberti. 2017 - All Rights
                                    Reserved</p>
                              </div>
                  </div>
      </footer>
</body>
{% endblock %}
</html>
```

index.html

```
{% extends "layout.html" %}
<!DOCTYPE html>
```

```
<html lang="en">
<head>
<meta charset="utf-8">
<title>Benvenuti</title>
</head>
<body>
{%block content%}
      <!--definizione dell'intestazione-->
      <div class="intro-header">
            <div class="container">
                  <div class="row">
                        <div class="col-lg-12">
                              <div class="intro-message">
                              <h1>Benvenuti!</h1>
                              <h3>Scegliere l'attività che si
                              desidera svolgere</h3>
                              <hr class="intro-divider">
                              <!--definizione dei bottoni che
                              portano alle due attività-->
                              <ul class="list-inline intro-
                              social-buttons">
                              <li>
                              <a href="
                              {{ url_for('analisi_grammaticale') }}"
                              class="btn btn-default btn-lg bottoni-
                              home"> Analisi grammaticale</a>
                              </li>
                              <li>
                              <a href=
                              "{{ url_for('genera_parola') }}"
                              class="btn btn-default btn-lg bottoni-
                              home">Genera, ascolta e ripeti!</a>
                              </li>

                              </ul>
                              </div>
                        </div>
                  </div>
            </div>
      </div>
<!--definizione del footer-->
      <footer>
            <div class="row">
                  <div class="col-lg-12">
                        <p>Copyright &copy; Chiara Zamberti. 2017 -
                        All Rights Reserved</p>
                  </div>
            </div>
```

```
      </footer>
</body>
{%endblock%}
</html>
```

layout.html

```
{% extends "bootstrap/base.html" %}
{% block styles %}
{{super()}}
<!--importazione del foglio di stile di bootstrap e di jQuery-->
<link rel="stylesheet" href="{{url_for('.static',
filename='bootstrap.min.css')}}">
<link rel="stylesheet" href="{{url_for('.static',
filename='styles.css')}}">
<script src="{{ url_for('static', filename='jquery-
3.1.1.js') }}"></script>
{% endblock %}
<!--definizione del blocco del titolo-->
{% block title %}
{{title}}
{% endblock %}
<!--definizione del blocco della barra di navigazione-->
{% block navbar %}
{{nav.mynavbar.render()}}
{% endblock %}
```

Foglio di stile

styles.css

```
/*definizioni degli stili generali*/
body,
html {
      font-family: "Lato","Helvetica Neue",Helvetica,Arial,sans-
      serif;
      background: url(img/sfondo_ag.jpg);
      width: 100%;
      height: 100%;
      font-weight: 500;
}

a {
      cursor:pointer;
      font-weight: bold;
}

th{
      color:white;
```

```
      text-align:center;
      background:#3fb13d
}

td{
      background-color: #fbf8eb;
      width:50%;
      min-width:50%;
}

footer {
      border-top: 1px solid #1c350f;
      padding: 1px 0 12px;
      background-color: #3fb23e;
}

/*definizioni degli stili per la barra di navigazione*/
.navbar-default {
      background-color: rgba(4, 155, 14, 0.73);
      border-bottom: 2px solid #1c350f;
      border-top:none;
      border-left:none;
      border-right:none;
      border-radius: 0px;
      margin-bottom: 0;
      color: #fff;
}

.navbar-default .navbar-brand {
      color: #fff;
      background-repeat: no-repeat;
      background-position-x: 50%;
}

.navbar-default .navbar-nav>li>a {
      color: #fff;
}

.navbar-default .navbar-nav>.active>a, .navbar-default .navbar-
nav>.active>a:focus, .navbar-default .navbar-nav>.active>a:hover {
      color: #555;
      background-color: #fcf8e3;
}

.navbar-default .navbar-toggle .icon-bar {
      background-color: #fff;
}
```

```
@media (min-width: 768px){
.navbar>.container .navbar-brand, .navbar>.container-fluid .navbar-
brand {
    color: #1f3b13;
}
}

#start_btn {
      position: fixed; /* or absolute */
      top: 45%;
      left: 45%;
}

/*definizione di una regola per spingere il footer in fondo
definendo le dimensioni del container*/
.container{
      min-height: 70%;
      min-height: -webkit-calc(100% - 50px);
      min-height: -moz-calc(100% - 50px);
      min-height: calc(100% - 50px);
}
/*definizioni degli stili per il titolo*/
.titolo{
      color: #08520c;
      margin-bottom: 3%;
}

/****stili pagina analisi_grammaticale****/
/*definizione degli stili per il pulsante di invio dati al server*/
input[type=submit] {
      -webkit-appearance: button;
      cursor: pointer;
      color: #ffffff;
      font-size: 2em;
}

.btn-primary {
      color: #fff;
      border: 1px solid;
      background-color: #3fb13d;
      border-color: #1c350f;
}
.btn-primary:hover {
      background-color: #338a32;
}

/*definizioni degli stili per le sezioni del gioco*/
.wideBox {
```

```
	clear: both;
	text-align: center;
	margin: 70px;
	padding: 10px;
	background: #ebedf2;
	border: 1px solid #333;
}

.wideBox h1 {
	font-weight: bold;
	margin: 20px;
	color: #666;
	font-size: 1.5em;
}

/*definizione degli stili del bottone per ricominciare il gioco*/
input.MyButton {
	padding: 20px;cursor: pointer;
	font-weight: bold;
	font-size: 150%;
	background: #93ff9b;
	color: #1c350f;
	border: 2px dashed #44b646;
	border-radius: 10px;
	-moz-box-shadow: 6px 6px 5px #999;
	-webkit-box-shadow: 6px 6px 5px #999;
	box-shadow: 6px 6px 5px #999;
}

input.MyButton:hover {
	color: #ddffdd;
	background: #1c350f;
	border: 2px dashed #45b747;
	-moz-box-shadow:: 5px 5px 4px #adadad;
	-webkit-box-shadow:: 5px 5px 4px #adadad;
	box-shadow:: 5px 5px 4px #adadad;
}

.bottoni-home {
	padding: 10px 16px;
	font-size: 18px;
	line-height: 1.3333333;
	border-radius: 10px;
	color: #f8f8f8;
	background-color: #45b647;
	border: 2px solid #f8f8f8;
}
```

```
.form-control {
      margin-bottom: 2%;
      display: block;
      width: 100%;
      padding: 6px 12px;
      font-size: 14px;
      line-height: 1.42857143;
      color: #555;
      background-color: #fbf8eb;
      border: 1px solid #1c350f;
      border-radius: 4px;
      -webkit-box-shadow: inset 0 1px 1px rgba(0,0,0,.075);
      box-shadow: inset 0 1px 1px rgba(0,0,0,.075);
      -webkit-transition: border-color ease-in-out .15s,-webkit-box-
      shadow ease-in-out .15s;
      -o-transition: border-color ease-in-out .15s,box-shadow ease-
      in-out .15s;
      transition: border-color ease-in-out .15s,box-shadow ease-in-
      out .15s;
}

/*definizione degli stili per la tabella dell'analisi grammaticale*/
.table-bordered>tbody>tr>td, .table-bordered>tbody>tr>th, .table-
bordered>tfoot>tr>td, .table-bordered>tfoot>tr>th, .table-
bordered>thead>tr>td, .table-bordered>thead>tr>th {
      border: 1px solid #1c350f;
}
.table-bordered {
      border: 1px solid #1c350f;
}

/*definizione degli stili per gli slot da trascinare e la relativa
allocazione*/
#draggable { width: 100px; height: 100px; padding: 0.5em;
      float: left; margin: 10px 10px 10px 0; }
#droppable { width: 150px; height: 150px; padding: 0.5em;
      float: left; margin: 10px; }

/*definizioni degli stili per gli slot delle parole e delle parti
del discorso*/
#origineParola {
      margin: 0 auto;
}

#destinazParteD, #origineParola {
      height: 120px;
      padding: 20px;
      -moz-border-radius: 10px;
```

```
	-webkit-border-radius: 10px;
	border-radius: 10px;
	-moz-box-shadow: 0 0 .3em rgba(0, 0, 0, .8);
}

#destinazParteD div, #origineParola div {
	text-align: center;
	float: left;
	width: auto;
	height: auto;
	min-width: 2em;
	padding: 10px;
	padding-top: 10px;
	padding-bottom: 0;
	border: 2px solid #84ca97;
	-moz-border-radius: 10px;
	-webkit-border-radius: 10px;
	border-radius: 10px;
	margin: 0 0 10px 10px;
	background: rgba(147, 255, 155, 0.16);
}

#destinazParteD div.hovered {
	background: #aaa;
}

#destinazParteD div {
	min-height:3em;
	border-style: dashed;
	font-size: 2em;
	padding-top:20px;
}

#origineParola div {
	background: #f8ffec;
	color: #08520c;
	font-size: 50px;
}

#origineParola div.ui-draggable-dragging {
	-moz-box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
	-webkit-box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
	box-shadow: 0 0 .5em rgba(0, 0, 0, .8);
}

/*definizione degli stili per il messaggio di successo*/
#successo{
	position: absolute;
```

```
      text-align: center;
      width: auto;
      height: auto;
      z-index: 100;
      background: #dfd;
      border: 2px solid #84ca97;
      -moz-border-radius: 10px;
      -webkit-border-radius: 10px;
      border-radius: 10px;
      padding: 20px;
      color: #1c350f;
}

/****stili pagina home****/
.intro-header {
      padding-top: 50px;
      padding-bottom: 50px;
      text-align: center;
      color: #f8f8f8;
      background: url(img/sfondo.jpg) no-repeat center center;
      background-size: cover;
}

.intro-message {
      position: relative;
      padding-top: 10%;
      padding-bottom: 20%;
}

.intro-message > h1 {
      margin: 0;
      text-shadow: 2px 2px 3px rgba(0,0,0,0.6);
      font-size: 5em;
}

.intro-divider {
      width: 400px;
      border-top: 1px solid #f8f8f8;
      border-bottom: 1px solid rgba(0,0,0,0.2);
}

.intro-message > h3 {
      text-shadow: 2px 2px 3px rgba(0,0,0,0.6);
}

@media(max-width:767px) {
      .intro-message {
            padding-bottom: 15%;
```

```
      }

      .intro-message > h1 {
            font-size: 3em;
      }

      ul.intro-social-buttons > li {
            display: block;
            margin-bottom: 20px;
            padding: 0;
      }

      ul.intro-social-buttons > li:last-child {
            margin-bottom: 0;
      }

      .intro-divider {
            width: 100%;
      }
}

.network-name {
      text-transform: uppercase;
      font-size: 14px;
      font-weight: 400;
      letter-spacing: 2px;
}

.content-section-a {
      padding: 50px 0;
      background-color: #f8f8f8;
}

.content-section-b {
      padding: 50px 0;
      border-top: 1px solid #e7e7e7;
      border-bottom: 1px solid #e7e7e7;
}

.section-heading {
      margin-bottom: 30px;
}

.section-heading-spacer {
      float: left;
      width: 200px;
      border-top: 3px solid #e7e7e7;
}
```

```
.banner {
      padding: 100px 0;
      color: #f8f8f8;
      background-size: cover;
}

.banner h2 {
      margin: 0;
      text-shadow: 2px 2px 3px rgba(0,0,0,0.6);
      font-size: 3em;
}

.banner ul {
      margin-bottom: 0;
}

.banner-social-buttons {
      float: right;
      margin-top: 0;
}

@media(max-width:1199px) {
      ul.banner-social-buttons {
      float: left;
      margin-top: 15px;
      }
}

@media(max-width:767px) {
      .banner h2 {
            margin: 0;
            text-shadow: 2px 2px 3px rgba(0,0,0,0.6);
            font-size: 3em;
      }

      ul.banner-social-buttons > li {
            display: block;
            margin-bottom: 20px;
            padding: 0;
      }

      ul.banner-social-buttons > li:last-child {
            margin-bottom: 0;
      }
}

p.copyright {
```

```
            margin: 15px 0 0;
}

/****stili pagina genera_parola****/
#checkbtn{
      font-size:1.5em;
}
.dropdown{
      text-align:center;
}
/*definizione del menù a tendina per la selezione delle categorie*/
.dropdown{
      font-size:1.5em;
}

.par{
      color:#08520c;
      font-size:2.5em;
}

.page-header {
      padding-bottom: 9px;
      margin: 40px 0 20px;
      border-bottom: 1px solid #1c350f;
}

/****stili pagina about****/
.info{
      text-align: center;
      font-size:1.2em;
}
.img-circle {
      margin: auto;
      border-radius: 50%;
}
.centrato{
      text-align: center;
}
.centrato img{
      margin: 10px;
}
```

Vocabolari

voc_city.txt

```
airplane airport ambulance bench bicycle billboard bridge building
bus car church garden hospital industry lamppost motorcycle pavement
petrol_station police policeman road shop skyscraper square stadium
station traffic_lights traffic_signal train tree truck villa
zebra_crossing
```

voc_clothing.txt

```
anorak bag bathrobe belt boots bow_tie bracelet button coat dress
earrings glasses gloves hat heavy_jacket jacket necklace pyjamas
raincoat ring scarf shirt shoes shorts skirt slippers socks sweater
swimsuit tie trainers trousers t-shirt umbrella underpants wallet
watch
```

voc_food.txt

```
apple apricot banana bread butter cake candy carrot cheese cherry
chicken chocolate coffee cookie egg fish ham ice_cream lemon
lemonade meat milk mushroom onion orange orangeade pasta peach pear
pepper pizza potato rice salad salami sandwich sausage soup
strawberry tea tomato water wine
```

voc_home.txt

```
armchair bath bed bottle broom brush bucket calendar chair
chandelier clock cup cutlery door fork fridge glass iron kitchen
knife lamp oven pan phone picture plate pot scales sink sofa table
tap television tent toilet toilet_paper vacuum_cleaner
```

voc_num_col.txt

```
zero one two three four five six seven eight nine ten eleven twelve
thirteen fourteen fifteen sixteen seventeen eighteen nineteen twenty
twenty-one twenty-two thirty forty fifty sixty seventy eighty ninety
one_hundred one_thousand ten_thousand hundred_thousand one_million
black blue brown gray green light_blue orange pink purple red white
yellow
```

voc_pobody.txt

```
arm back belly chin ear eye eyebrow eyelash finger foot hair  hand
head leg mouth neck nose shoulder teeth
```

voc_school.txt

```
blackboard book brush bus calculator chair chalk classroom clip
closet colours compass computer desk diary door line litter_bin
notebook pen pencil rubber ruler schoolbag scissors sellotape
set_square sharpener sheet student teacher teacher's_desk window
```

Script di avvio automatico del server web nella VM (in /etc/init.d)

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start daemon at boot time
# Description:       Enable service provided by daemon.
### END INIT INFO

dir="/home/int_en/Interactive_English"
cmd="python interactiveEnglish.py"
user="int_en"

name=`basename $0`
pid_file="/var/run/$name.pid"
stdout_log="/var/log/$name.log"
stderr_log="/var/log/$name.err"

get_pid() {
    cat "$pid_file"
}

is_running() {
    [ -f "$pid_file" ] && ps `get_pid` > /dev/null 2>&1
}

case "$1" in
    start)
    if is_running; then
        echo "Already started"
    else
        echo "Starting $name"
        cd "$dir"
        if [ -z "$user" ]; then
            sudo $cmd >> "$stdout_log" 2>> "$stderr_log" &
```

```
        else
            sudo -u "$user" $cmd >> "$stdout_log" 2>> "$stderr_log"
            &
        fi
        echo $! > "$pid_file"
        if ! is_running; then
            echo "Unable to start, see $stdout_log and $stderr_log"
            exit 1
        fi
    fi
    ;;
    stop)
    if is_running; then
        echo -n "Stopping $name.."
        kill `get_pid`
        for i in {1..10}
        do
            if ! is_running; then
                break
            fi

            echo -n "."
            sleep 1
        done
        echo

        if is_running; then
            echo "Not stopped; may still be shutting down or
                  shutdown may have failed"
            exit 1
        else
            echo "Stopped"
            if [ -f "$pid_file" ]; then
                rm "$pid_file"
            fi
        fi
    else
        echo "Not running"
    fi
    ;;
    restart)
    $0 stop
    if is_running; then
        echo "Unable to stop, will not attempt to start"
        exit 1
    fi
    $0 start
    ;;
```

```
    status)
    if is_running; then
        echo "Running"
    else
        echo "Stopped"
        exit 1
    fi
    ;;
    *)
    echo "Usage: $0 {start|stop|restart|status}"
    exit 1
    ;;
esac

exit 0
```

Bibliografia

- Baldan, Paolo. 2004-2005. *Introduzione a Unix*. Lucidi per il corso di Laboratorio di Sistemi Operativi, Università Ca' Foscari di Venezia.
- Bird, Steven, Ewan Klein ed Edward Loper. 2009. *Natural Language Processing with Python*, Massachusetts, O'REILLY Media.
- Cannon, Jason. 2014. *Linux Succinctly*, Morrisville, Syncfusion Technology Resource Portal.
- Carter, Johanna, Giulia Achilli, Brunel Brown. 2016. *Guida per i genitori a supporto di Let's Be Friends*, Londra, Pearson.
- Foster, Frances e Brunel Brown. 2016. *Let's be Friends*, con espansione online. Per la Scuola Elementare, volumi 1 e 2. Londra, Pearson.
- Foster, Frances e Brunel Brown. 2014. *Top secret*, con fascicolo, e-book, ed espansione online. Per la Scuola Elementare, con CD-ROM, volumi 3, 4 e 5. Londra, Pearson.
- Grinberg, Miguel, 2014. *Flask Web Development*, Massachussets, O'REILLY Media.
- Istituto Majorana, *Guida all'Installazione Minimale di Ubuntu*, versione 1.0.
- Kusnetzky. 2011. *Virtualization: A Manager's Guide*, Massachusetts, O'REILLY Media.
- Lutz, Mark. 2013. *Learning Python*, 5th Edition, Massachusetts, O'REILLY Media.
- Middleton, Schneeman. 2013. *Heroku Up & Running: Effortless application deployment and scaling*, Massachusetts, O'REILLY Media.

Sitografia

- *Alphabetical list of part-of-speech tags used in the Penn Treebank Project*.

  Link alla fonte: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

- *Computer per bambini: migliori sistemi divertenti, educativi e sicuri*. 8 febbraio 2017. Redazione di Navigaweb.it.

  Link al sito: http://www.navigaweb.net/2012/07/computer-per-bambini-migliori-sistemi.html.

- *CSS*, *Bootstrap*, *JavaScript*.

  Per maggiori informazioni, consultare il sito W3Schools: https://www.w3schools.com/.

- *Django*

  Link al progetto: https://www.djangoproject.com/

- *Document Number DSP0243*. 2010. In: Section 5, Open Virtualization Format Specification.

  Link al documento: http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.1.0.pdf.

- *Flask-nav*.

  Link al repository: https://github.com/mbr/flask-nav.

- *Git*, sistema di controllo versione.

  Link al sito ufficiale: https://git-scm.com/.

- *Guida a jQuery UI di Mr. Webmaster*. 2009. Draggable e Droppable.

  Link di riferimento: https://www.mrwebmaster.it/jquery/guide/guida-jquery-ui/.

- *Heroku, cos'è e come funziona la piattaforma cloud per app*.

Link al sito: http://www.fastweb.it/web-e-digital/cos-e-e-come-funziona-heroku-piattaforma-cloud-per-app/.

- *Impariamo la parola: Inglese, Lessico di base* di Rossana Cannavacciuolo. Versione 1.1.

  Link al sito: http://rossanaweb.altervista.org/blog/il-mio-software/inglese-e-francese-lessico-di-base/.

- *Init-script-template*.

  Link al repository:

  https://github.com/fhd/init-script-template/blob/master/template.

- *Interactive English Ova*.

  Link al repository: https://bitbucket.org/c_zamberti/interactive_english_ova.

  Link al file su Google Drive:

  https://drive.google.com/open?id=0B6ELqs6wBndQLXRzSEp2YTdkaWc

- *JQuery*.

  Link al sito: https://jquery.com/.

- *JQuery Essential Guide*, esempio drag e drop su slot numerati.

  Link al sito: http://www.elated.com/articles/drag-and-drop-with-jquery-your-essential-guide/.

- *Lista porte standard di TCP/UDP.*

  Link al sito: http://www.tcp-udp-ports.com/ports5000-5100.html.

- *NumPy Manual*. 2017. User Guide, Setting Up, Installing Numpy.

  Link al sito: https://docs.scipy.org/doc/numpy/index.html.

- *OrderedDict: Remember the Order Keys are Added to a Dictionary*.

Link al sito: https://pymotw.com/3/collections/ordereddict.html.

- *Pip documentation*, Release 10.0.0.dev0, March 24, 2017.

  Link al documento: https://media.readthedocs.org/pdf/pip/latest/pip.pdf.

- *Programma Operativo Nazionale: Per la Scuola, competenze e ambienti per l'apprendimento*.

  Link al sito: http://www.istruzione.it/pon/.

- *Regular Expression HOWTO*, Kuchling.

  Link al sito:
  https://www.tutorialspoint.com/python/python_reg_expressions.htm.

- *ResponsiveVoice.*

  Link al sito del progetto e alla relativa documentazione:
  https://responsivevoice.org/.

- *Scuola Primaria "De Amicis IC Pollione"*, Formia (LT).

  Link al sito web della scuola: http://www.icpollione.it/.

- *Techopedia explains Natural Language Toolkit (NLTK)*.

  Link alla pagina :

  https://www.techopedia.com/definition/30343/natural- language-toolkit-nltk.

- *Template Designer Documentation: List of Control Structures*.

  Link alla guida online di Jinja2: http://jinja.pocoo.org/docs/2.9/templates/.

- *Ubuntu.*

  Link al sito ufficiale: http://www.ubuntu-it.org/.

- *VirtualBox, s*ito ufficiale del progetto.

Link: https://www.virtualbox.org/.

- *What is a localhost?*

  Link: http://whatismyipaddress.com/localhost.