FabbricaSemantica - Progetto di Metodologie di Programmazione 2018/2019 - prof. Roberto Navigli, dott.ssa Bianca Scarlini e dott. Paolo Spadoni

Questo progetto ha l'obiettivo di creare una piattaforma collaborativa di annotazione e validazione di dati linguistici e visuali. La piattaforma dispone di una componente front-end e di una back-end. La parte front-end si occuperà di creare tutte le pagine di interazione con l'utente, mentre la parte back-end sarà responsabile di elaborare e salvare i dati forniti dall'utente.
L'applicazione idealmente girerà su una macchina server, ovvero su una macchina raggiungibile tramite internet e interrogabile mediante il protocollo HTTP. Ai fini della consegna, tuttavia, sarà sempre possibile utilizzare e far girare tutto sulla propria macchina di sviluppo, utilizzando l'indirizzo localhost (127.0.0.1).
Il progetto è un'applicazione client-server, per cui sarà necessario installare Tomcat (un web application server che ospita i servizi web sviluppati in Java) sul proprio computer o server remoto.
Nel caso in cui l'applicazione venga eseguita in locale la home del sito web sarà: *http://127.0.0.1:8080/fabbricasemantica/index.html*

Componente front-end (20 punti)

Lato front-end, ovvero per quanto riguarda la componente che si occupa di presentare le pagine all'utente, la libreria che utilizzerete è JSweet, una libreria per la creazione di pagine HTML a partire da codice Java e che permette di integrare al suo interno anche JavaScript e JQuery. **La libreria supporta fino a Java 8, quindi dovrete compilare con quella versione** (tasto destro sul progetto, properties, java compiler, enable project specific setting, 1.8)**.**
Per chi vuole sviluppare solo la parte front-end oppure testare la propria componente, sarà fornita una versione "giocattolo" del back-end scaricabile da questo link.

Componente back-end (15 punti)

La componente back-end dovrà fornire all'utente i dati per i task di annotazione e validazione e gestire i dati ricevuti dall'utente e salvarli. Questa componente dovrà anche gestire le sessioni degli utenti assicurandosi che un utente, una volta effettuato l'accesso alla piattaforma, resti autenticato fino al logout. L'accesso alle pagine di annotazione e validazione (vedi sotto) sarà quindi consentito ai soli utenti autenticati.

Struttura del progetto

Le richieste del progetto, che verranno approfondite nelle prossime pagine, si possono così riassumere:

1. Front-end (**20 punti**)
    a. Creare, utilizzando JSweet, le pagine web per la gestione degli utenti e delle annotazioni/validazione (o come pagine javascript o come videogioco, secondo il tipo di progetto -- si veda sotto).
    b. creare il proprio task di annotazione/validazione (solo progetto base).
2. Back-end (**10 punti**)
    a. Creare le classi per il salvataggio dei dati usando una delle tecnologie proposte.
    b. Implementare la logica richiesta utilizzando le Servlet.
    c. Implementare le classi per ottenere dati di annotazione/validazione in maniera più “intelligente” (es. utilizzando WordNet implementato nell’ homework 3!). (**5 punti extra**)

Nelle pagine seguenti viene fornita una descrizione dettagliata delle funzionalità **minime** che il sito web deve avere. Tuttavia, sono apprezzati e incoraggiati eventuali miglioramenti e/o abbellimenti della piattaforma (ad esempio rendendola più interattiva, user-friendly o con piccole funzionalità in più), sebbene non esplicitamente richiesti. La valutazione finale sarà altamente influenzata da diversi fattori tra cui: **riuso del codice**, incapsulamento, ereditarietà, polimorfismo ed eventuale uso di design pattern appropriati, **documentazione**.

Progetto base

(può essere scelto da tutti gli studenti)

L'obiettivo del progetto base è creare un'applicazione web costituita da un serie di pagine web che gestiscono il profilo dell'utente e i task di annotazione/validazione che dovrà effettuare. L'interazione tra l'applicazione e l'utente, quindi, avverrà attraverso delle normali pagine web che risiederanno sul lato server e saranno recuperate dal client tramite il browser.

Front-end

La parte front-end si divide in due moduli: pagine per la gestione utente e pagine per la gestione annotazioni/validazioni. Le pagine per la gestione dell'utente devono permettere le funzionalità base di autenticazione e registrazione. Le pagine per la gestione annotazioni/validazioni devono permettere all'utente di eseguire i vari task di annotazione e validazione (per maggiori dettagli sui task di annotazione vedere la sottosezione Gestione Validazioni/Annotazioni). Per ciascuna pagina html sarà richiesto di creare una classe Java (tenuto conto del riuso del codice necessario) che sarà "transpilata" tramite JSweet in codice Javascript/JQuery che dovrà essere incluso nella corrispondente pagina html, come illustrato in appendice.

Gestione Utente

Le tipologie di pagine da codificare per la **gestione dell'utente** sono le seguenti:

- *login.html*: permette all'utente di autenticarsi al sistema. La pagina di login deve prevedere:
    - Una form in cui l'utente deve inserire:
        - email con cui ha effettuato la registrazione
        - password
    - Un pulsante per fare il submit della form chiamato *login*. Una volta cliccato questo pulsante i dati inseriti vengono mandati al server all'indirizzo *login.jsp.*
    - Un link per andare alla pagina di registrazione utente (vedi sotto) chiamato *signup.*
- *signup.html*: permette all'utente di registrarsi. La pagina di registrazione prevede:
    - Una form in cui l'utente deve inserire:
        - email (che verrà utilizzato come identificativo per l'utente)
        - password, questo campo deve essere oscurato*.

        - ripetizione della password inserita, questo campo deve essere oscurato*.
        - lingue parlate come madrelingua, questo campo prevede una *checkbox (selezione multipla)* con le lingue presenti nel sistema. L'utente deve selezionare una o più lingue.
        - altre lingue parlate e il livello con cui sono parlate (A1, A2, B1, B2, C1, C2). Questo campo è opzionale (può essere lasciato vuoto dall'utente).
    - Un pulsante per fare il *submit* della form chiamato *signup*. Una volta cliccato questo pulsante i dati inseriti vengono mandati al server all'indirizzo *signup.jsp.*
    - Un link per andare alla pagina di login chiamato *login.*

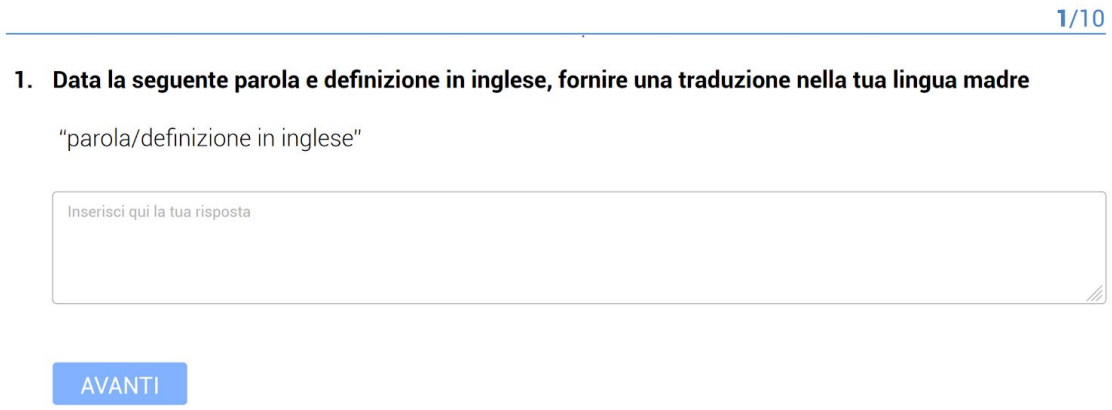
***** se un campo è oscurato significa che nel caso in cui l'utente digita "123" nella form viene visualizzato "***".

N.B. Al fine del progetto è possibile limitare il numero di lingue a EN (English) e IT (Italian).

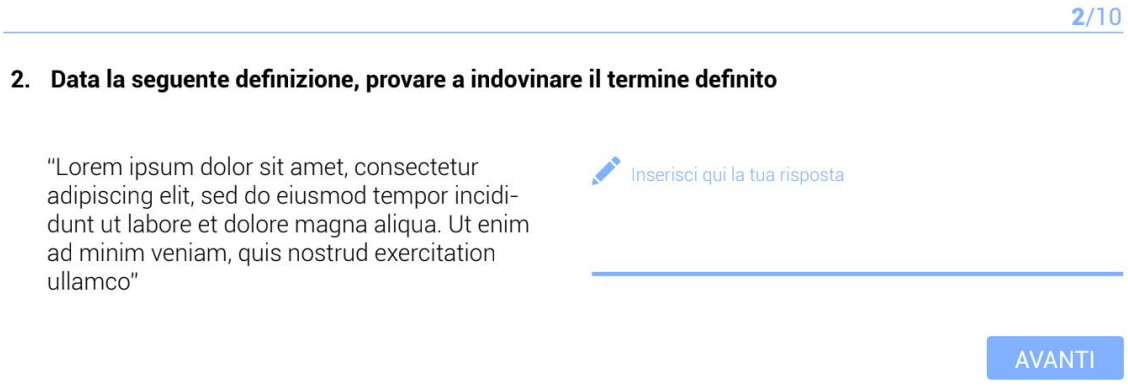
Gestione Annotazioni e Validazioni

Le tipologie di pagine da codificare per la **gestione delle annotazioni/validazioni** sono le seguenti (le immagini sono d'esempio e non si richiede in nessun caso di utilizzare quella disposizione né il testo ivi riportato):

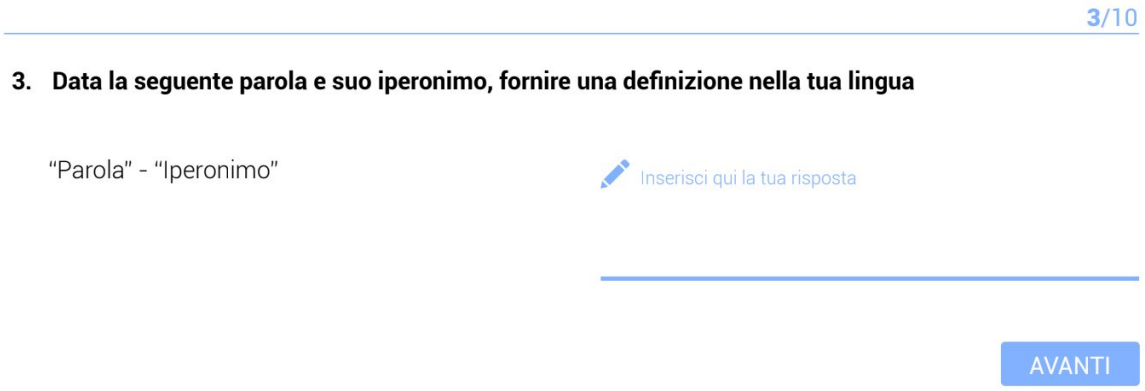
- *home.html*: permette all'utente di iniziare ad eseguire i task di annotazione/validazione. La home prevede:
    - Un pulsante di nome *start* (potrebbe essere un'immagine, ad es. l'icona play). Una volta premuto questo pulsante l'utente viene indirizzato ad una pagina di annotazione/validazione scelta a caso tra quelle associate ai task implementati (vedi sotto).
    - Un link attraverso cui l'utente effettua il logout e viene indirizzato alla pagina *logout.jsp.*
- *translationAnnotation.html*: data una parola e una sua definizione in inglese, l'utente deve fornire una traduzione nella sua lingua madre. La pagina prevede:
    - Parola e definizione visualizzati all'interno della pagina.
    - Un campo di testo in cui l'utente deve scrivere la sua traduzione.



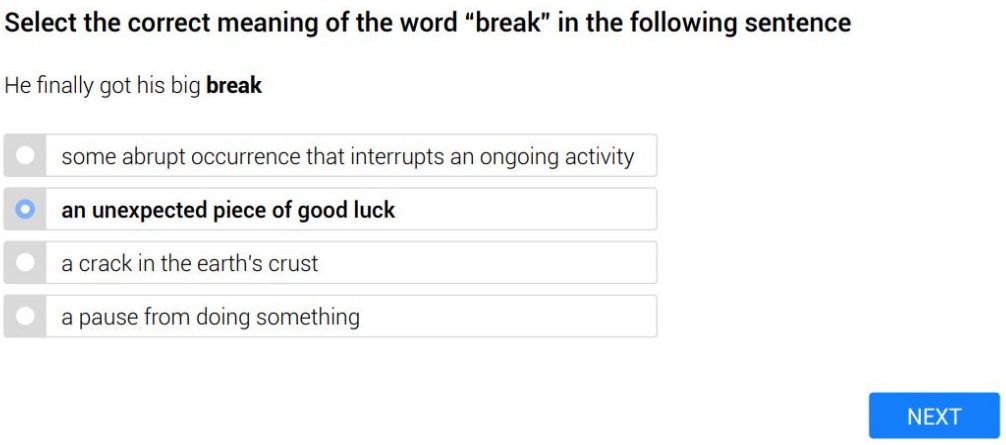
- *wordAnnotation.html*: data una definizione in inglese, l'utente deve provare a indovinare la parola definita. La pagina prevede:
    - Definizione visualizzata all'interno della pagina.
    - Un campo di testo in cui l'utente deve scrivere la parola definita.



- *definitionAnnotation.html*: data una parola e un suo iperonimo (ovvero, una sua generalizzazione, ad es. veicolo come iperonimo di macchina), l'utente deve fornire una definizione nella sua lingua. La pagina prevede:
    - Parola e iperonimo visualizzati all'interno della pagina.
    - Un campo di testo in cui l'utente deve scrivere la sua traduzione.



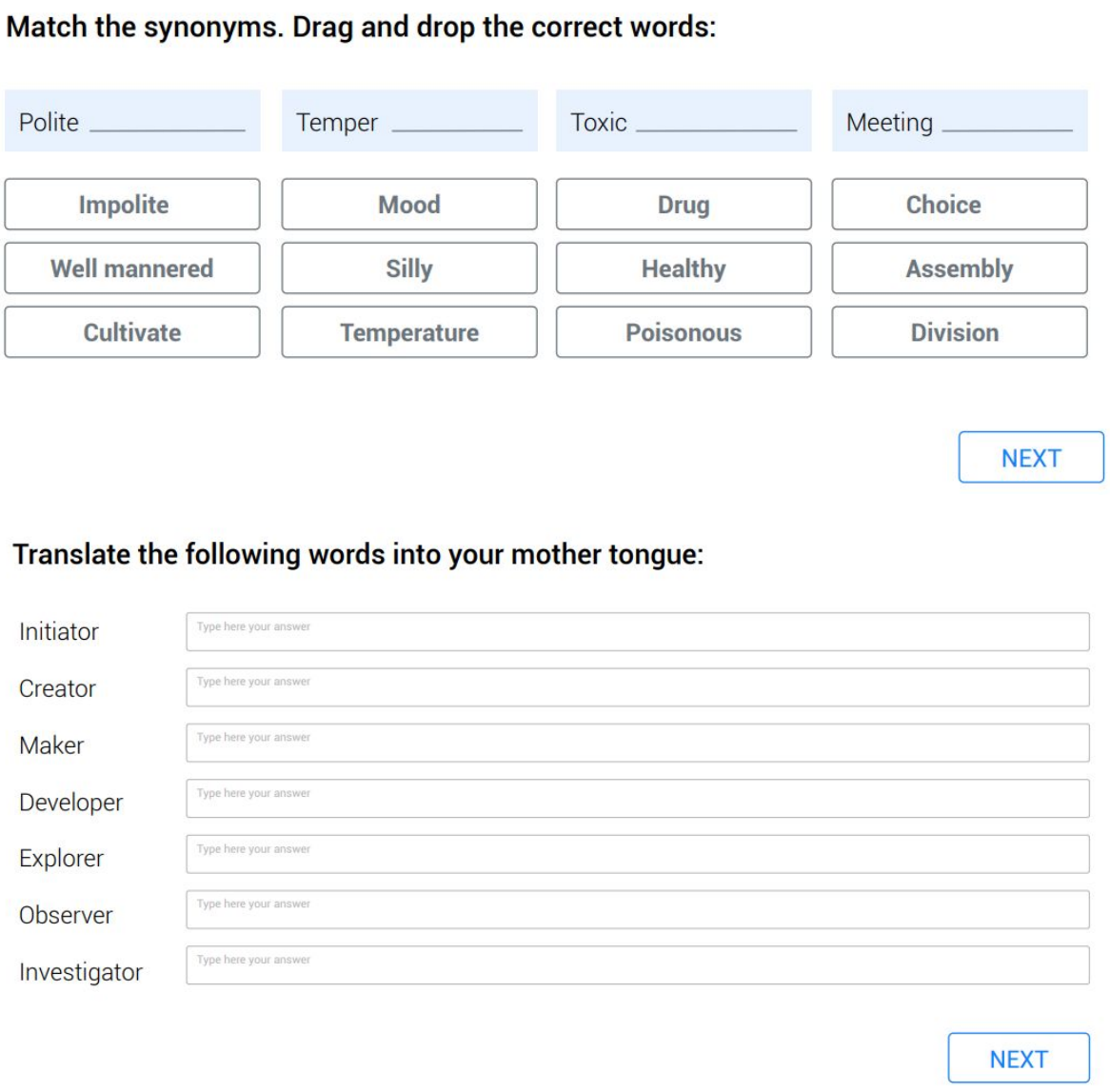
- *senseAnnotation.html*: data una parola e una frase in cui la parola appare, l'utente deve selezionare il senso appropriato. La pagina prevede:
  - Parola e frase visualizzate all'interno della pagina.
  - Una *checkbox (selezione multipla)*. Tra le opzioni devono essere presenti i sensi forniti dal sistema.



- *translationValidation.html*: data una parola e una sua definizione in inglese, l'utente deve scegliere la miglior traduzione tra quelle fornite o specificare <nessuna>. La pagina prevede:
  - Parola e definizione visualizzate all'interno della pagina.
  - Una *checkbox (selezione multipla)*. Tra le opzioni devono essere presenti le traduzioni fornite dal sistema e l'opzione <nessuna>. L'opzione <nessuna> deve essere selezionata dall'utente quando nessuna delle traduzioni proposte è considerata corretta.

- *senseValidation.html*: data una parola e una frase in cui appare, l'utente deve verificare se il senso fornito dal sistema è appropriato. La pagina prevede:
  - Parola, frase e senso visualizzati all'interno della pagina.
  - Un *radio button* con le opzioni "Sì" e "No".

- *myAnnotation.html*: che implementa un nuovo task a piacere dello studente (**N.B.** potrebbe essere un mini-gioco o un task più dinamico, grazie alla flessibilità di JSweet, ad esempio con componenti trascinabili). In questo tipo di annotazione è molto apprezzata la fantasia, purché sensata, ad esempio:



Tutte le pagine di annotazione/validazione dovranno prevedere anche le seguenti componenti:

- Un pulsante chiamato *next* attraverso cui l'utente invia la propria annotazione al server. Una volta cliccato questo pulsante i dati inseriti vengono mandati al server all'indirizzo *{nome_pagina}.jsp.* (Ad esempio, se mi trovo nella pagina *translationAnnotation.html* viene chiamata la pagina *translationAnnotation.jsp* che salverà i dati lato back-end).
- Un pulsante chiamato *skip* attraverso cui l'utente salta l'annotazione corrente. Una volta cliccato questo pulsante l'utente viene indirizzato ad una pagina di annotazione/validazione a caso tra quelle implementate.

- Un link alla pagina *home.html.*
- Un link attraverso cui l'utente effettua il logout e viene indirizzato alla pagina *logout.jsp.*

Per la parte di stylesheet (ovvero per formattare lo stile delle pagine web), sarà possibile utilizzare CSS o bootstrap. Viene fornita l'implementazione con JSweet di una pagina di esempio "ExamplePage.java".

Back-end

Per salvare le informazioni da fornire all'utente e quelle di annotazione/validazione provenienti dallo stesso utente, sarà necessario scegliere una tra le diverse tecnologie possibili:

1) **File di testo:** Tutte le informazioni possono essere salvate come file di testo opportunamente formattati (ad esempio un file .tsv in cui le informazioni sono separate da un carattere *tab*, fino a formati più elaborati come JSON o XML).
   a) Ad esempio gli utenti possono essere salvati in un file *utenti.tsv* nel seguente modo (`\t` indica il carattere *tab*):

```
username1 \t password1 \t nome1 \t lingua1 \t livello1 ...
username2 \t password2 \t nome2 \t lingua2 \t livello2 ...
...
```

   b) Nel caso di utilizzo di formati più complessi come XML (gestibile con la libreria *DOM* già all'interno di Java), questo formato permette di salvare i dati in maniera strutturata creando degli elementi identificati da un `tag` e da degli attributi che definiscono l'elemento. La struttura può essere ricorsiva, ossia un elemento può contenere al suo interno altri elementi. Esempio:

```
...
<utenti>
	<utente id=”1”>
		<username>mario99@java.it</username>
		<password>pass1234</password>
	</utente>
	<utente id=”2”>
		<username>carla00@cmail.com</username>
		<password>1234pass</password>
	</utente>
	...
</utenti>
...
```

2) MySQL (scaricato come jar) o PostgreSQL, un relational database management system, ovvero un sistema di gestione di basi di dati relazionali (ulteriori informazioni qui).

Nel corso di questo documento, indipendentemente dalla tecnologia scelta per gestire i dati dell'utente e delle annotazioni/validazioni, useremo il termine ***base di dati*.**

La logica del back-end sarà implementata mediante l'utilizzo di *Servlet* Java. Le servlet sono semplici classi Java con un ciclo di vita ben definito:

- Come prima cosa la servlet riceve tutti i dati spediti dal client e li salva nell'oggetto `HttpServletRequest request`.
- A seconda del tipo di richiesta effettuata vengono invocati i metodi `doGet()` o `doPost()`, a cui viene passato l'oggetto `request`.
- Alla fine del metodo `doGet()` o `doPost()`, la servlet può reindirizzare l'utente a una nuova pagina web (`sendRedirect`), oppure creare la propria pagina utilizzando l'oggetto di output `HttpServletResponse response`.

N.B. Per semplificare lo sviluppo delle servlet, al posto di utilizzare i metodi `doGet()` o `doPost()`, sarà necessario implementare un unico metodo `doSomething()`. Le servlet sono già preimpostate per cui occorrerà soltanto implementare la logica a seconda del compito da svolgere; *tuttavia, se lo si ritiene opportuno, è possibile cambiare la loro struttura purché il sito web faccia quanto richiesto*.

Gestione Utente

Le Servlet per la gestione dell'utente devono permettere di elaborare i dati inviati dalle rispettive pagine *.html* create per il front-end. Per assicurarsi che un utente rimanga loggato per tutta la durata della sua navigazione all'interno del sito, occorrerà salvare il suo `username` (solo al momento del login) in un oggetto specifico chiamato `HttpSession`:

```
HttpSession session = request.getSession();
session.setAttribute("username", username);
```

Quindi, per vedere se un utente è autenticato al sistema basterà controllare se l'attributo `username` è settato oppure no all'interno dell'oggetto session nel seguente modo:

```
HttpSession session = request.getSession();
boolean isLoggedIn = session.getAttribute("username") != null;
```

Per il back-end riguardante la **gestione dell'utente** è necessario implementare la logica delle seguenti servlet (sottopackage `user`):

- *LoginServlet.java*: questa servlet viene invocata all'indirizzo login.jsp . Il metodo `doSomething()` dovrà almeno:
    - Prendere l'username, la password dall'oggetto `request`
        - Se nella base di dati è presente un utente con username e password uguali allora si salva l'utente come autenticato e si reindirizza l'utente alla pagine principale *home.html*
        - Altrimenti, si reindirizza l'utente alla pagina di login *login.html*
- SignupServlet.java: questa servlet viene invocata all'indirizzo *signup.jsp . Il metodo* `doSomething()` dovrà almeno:
    - Prendere i campi della form (username, password, conferma password, ...) all'interno dell'oggetto `request`
        - Se lo username esiste già all'interno della base di dati allora si reindirizza l'utente alla pagina di *signup.html*
        - Se la password e la sua conferma non corrispondono si reindirizza l'utente alla pagina di *signup.html*
        - Altrimenti si salva lo username, l'utente viene salvato all'interno della base di dati e si reindirizza l'utente a *home.html*
- LogoutServlet.java: questa servlet viene invocata all'indirizzo *logout.jsp.* Il metodo `doSomething()` dovrà almeno:
    - Rimuovere lo username dalla sessione
    - Redirigere l'utente alla pagina *login.html*

Annotazioni e Validazioni

Le Servlet da implementare per la **gestione delle annotazioni e validazioni** sono molto simili tra di loro in quanto svolgono le medesime funzioni. L'unico aspetto per cui differiscono sono il tipo di dati che devono gestire.
Di seguito per *{nome_pagina}* si intende uno dei task richiesti*: translationAnnotation, wordAnnotation, definitionAnnotation, senseAnnotation, translationValidation, senseValidation.* Le rispettive servlet si trovano all'interno del sottopackage `task`.

- *Task{nome_pagina}Servlet.java*: questa servlet viene invocata all'indirizzo *{nome_pagina}.jsp. Il metodo* `doSomething()` dovrà almeno:
    - Se è stato premuto il pulsante *skip,* reindirizza l'utente a una pagina di annotazione/validazione a caso tra quelle implementate.
    - Se è stato premuto il pulsante *next,* l'annotazione (o validazione) viene salvata nella base di dati. L'utente viene infine reindirizzato a una pagina di annotazione/validazione a caso tra quelle implementate.

API REST

Alcune funzionalità base vengono fornite attraverso chiamate a indirizzi predefiniti che implementano funzionalità utili.
N.B. Non sono servlet da implementare, sono già implementate per voi.

Di seguito vengono riportate le chiamate agli indirizzi con l'input richiesto (ove necessario) e il tipo di dati restituiti.

- */nextExample.jsp*: Una chiamata a questo indirizzo prende un parametro "`task`" che indica il task per il quale si vuole il prossimo esempio. Un esempio di chiamata al servizio lo si trova all'interno di "ExamplePage.java". Sarà sufficiente chiamare la pagina come: "`nextExample.jsp?task=<taskID>`" per avere il prossimo esempio per il task `<taskID>`.
    - restituisce una stringa JSON contenente il prossimo elemento da annotare per il `<taskID>` specificato.
- */isLoggedIn.jsp*: Una chiamata a questo indirizzo restituisce true o false a seconda che l'utente sia loggato o meno al sistema.

Punti Extra

La versione del back-end fornita di base ritorna sempre lo stesso esempio, uno per ogni task, utilizzando il `DataProvider` standard (vedi progetto). Per ottenere un punteggio > 30 sarà necessario implementare la propria classe `DataProvider` affinché fornisca dati in maniera più "intelligente".

Progetto gioco

(in alternativa al progetto base, può essere scelto solo da quegli studenti che abbiano ottenuto >= 27 a entrambi i primi due esoneri o 27 al primo esonero della teledidattica)

Lato front-end, il progetto richiede lo sviluppo di un gioco semplice (si suggerisce snake o pacman) utilizzando il framework di JSweet (che genera il corrispondente html/javascript) che ingloba lo scopo di annotare o validare uno dei compiti su cui lavora il progetto base. L'integrazione delle annotazioni nel gioco sono lasciate alla fantasia dello studente. Un esempio di gioco implementato con JSweet si trova al seguente link.

Gestione Utente

La gestione dell'utente rimane esattamente uguale a quella del progetto base.

Annotazioni/Validazioni

In questa parte c'è libera fantasia nell'implementare un gioco che abbia come scopo annotare o validare uno dei task proposti nel progetto base. Di conseguenza il gioco dovrà essere "transpilato" in javascript e inglobato in una pagina html, come per i task standard del progetto base. È possibile implementare ed utilizzare le servlet del progetto base, tuttavia in questo caso le annotazioni dovranno essere salvate senza interrompere lo svolgimento del gioco (ad esempio possono essere salvate a fine partita, o incrementalmente utilizzando chiamate in background).

API REST

Tutte le servlet che implementano API REST rimangono liberamente utilizzabili per l'implementazione del gioco.

Punti Extra

Si vedano le indicazioni del progetto base.

Modalità di consegna

La consegna deve essere effettuata mediante il progetto `metodologie2019_progetto` all'interno del proprio account github. E' necessario compilare il seguente modulo ai fini della consegna. La scadenza per la consegna è prevista per la **mezzanotte in qualsiasi parte del pianeta** 5 giorni prima del giorno dello scritto di metodologie fino a febbraio 2020. Ad esempio, se lo scritto si tiene il 25 dicembre <:'), la consegna è fissata entro la mezzanotte del 20 dicembre.

I progetti che ottengono un voto >= 30 saranno premiati con una maglietta di BabelNet!

Plagio

Indicazioni significative di plagio porteranno gli studenti coinvolti all'annullamento di tutti gli esoneri svolti (anche in precedenza) e all'annullamento di eventuali esami sostenuti in precedenza, **senza distinguere tra chi ha fatto copiare e chi ha copiato**. Si consiglia **caldamente** **di non condividere il codice con altri studenti (neanche per 5 minuti)**.

Appendice

Come creare un progetto Web

Come prima cosa è necessario installare sul proprio computer le seguenti componenti:

- Il web server **Apache Tomcat** (versione 8.5 o successive). Basterà scaricare ed estrarre l'archivio in un'apposita cartella sul proprio computer.
- Eclipse EE (Enterprise Edition), l'IDE di Eclipse che include strumenti per lo sviluppo di software per il web.

Una volta installate queste componenti è necessario configurare Tomcat all'interno del proprio Eclipse workspace seguendo i passi specificati in questo link.

A questo punto basterà importare il progetto del back-end in EclipseEE. Per poter navigare all'interno del proprio sito web è necessario prima avviare il server Tomcat sul proprio computer. È possibile avviare il server sia da linea di comando che da EclipseEE, selezionando la cartella principale del progetto e cliccando su *Run as* → *Run on server* (avviarlo da Eclipse è consigliato onde evitare problemi).

Come strutturare il proprio progetto

È disponibile il progetto *FabbricaSemantica* (progetto in EclipseEE per il lato back-end) da usare come scheletro per implementare il proprio Web server.
Il progetto è strutturato nella seguente maniera:

- All'interno della sottocartella `src` di `JavaResources` è contenuta l'implementazione delle componenti base del server giocattolo.
- La cartella `WebContent` rappresenta la cartella che contiene il sito web. Al suo interno devono essere presenti:
    - Tutte le pagine .html che fanno parte del proprio sito web.
    - Una sottocartella *js* con all'interno le pagine .js create attraverso JSweet.

Per evitare conflitti tra JSweet e Tomcat si richiede di creare un nuovo progetto *FabbricaSemanticaJSweet* per la sola gestione della parte di front-end. Quindi, è possibile creare i file javascript all'interno del progetto *FabbricaSemanticaJSweet* e poi spostarli all'interno dell'apposita cartella del progetto *FabbricaSemantica*. Un modo semplice per creare il progetto di JSweet è partire da un progetto maven già

configurato reperibile al seguente link. Una volta clonata la repository, dovrete cambiare il nome ed implementare il vostro codice (modulare) al suo interno.
I progetti si troveranno quindi all'URL https://github.com/<NOME_UTENTE>/metodologie_<COGNOME>.<MATRICOLA>/tree/master/`metodologie2019_progetto`.

Come inserire javascript all'interno di pagine html

La creazione di una pagina web è composta da alcuni semplici passaggi ma che possono essere confusi facilmente:

1. Per prima cosa bisogna creare una classe `<nome>.java` nel progetto *FabbricaSemanticaJSweet* contenente un metodo *main* con all'interno il codice che crea il Javascript della pagina web.
2. Selezionare la cartella principale del progetto e selezionare *Run as → Maven generate-sources*. Questo comando crea le pagine Javascript (`<nome>.js`) nella cartella `target` del progetto FabbricaSemanticaJSweet.
3. Prendiamo i file .js e li spostiamo nella cartella `WebContent/js` del progetto FabbricaSemantica.
4. Creiamo una pagina `<pagina>.html` all'interno di `WebContent` e dentro il tag `body` andiamo ad aggiungere la seguente riga:
   `<script type="text/javascript" src="js/<nome>.js"></script>`

Un esempio di pagina html è "`example.html`" all'interno del progetto back-end, la quale incorpora anche tutti i link necessari per lavorare con bootstrap.

Link utili

Per saperne di più sui tag html/css e i loro attributi, un buon link di riferimento si trova qui.