**“Software Engineering”**

**Course**

**a.a. 2019-2020**

**Lecturer: Prof. Henry Muccini (henry.muccini@univaq.it)**

**Progetto 1-Team Fresh**

|  |  |
| --- | --- |
| **Date** | <05/12/2019> |
| **Deliverable** | D1 |
| **Team (Name)** | Team Fresh |

|  |  |  |
| --- | --- | --- |
| **Team Members** | | |
| **Name & Surname** | **Matriculation Number** | **E-mail address** |
| **Piero Ferri** | *#245484* | *piero.ferri@student.univaq.it* |
| **Davide Fasciano** | *#245433* | *davide.fasciano@student.univaq.it* |
|  |  |  |
|  |  |  |
|  |  |  |

1. **Progetto 1. Gestionale offerte di lavoro (#12 #13)**

Il software consiste in un gestionale che permette agli offerenti (ditte, enti, docenti) di descrivere gli impieghi offerti e ai richiedenti (ad esempio, studenti, ma non solo) di cercare tra le offerte e candidarsi.

Il software dovrà essere in grado, tra le altre cose, di:

* elaborare le offerte per effettuare una categorizzazione delle stesse;
* elaborare i dati utente con le rispettive preferenze di lavoro;
* generare un punteggio in base ad ogni candidatura;
* presentare in primo piano all’utente le offerte di lavoro che fanno match, al meglio, con il proprio profilo.
* presentare in primo piano agli offerenti i candidati che fanno match, al meglio, con le proprie offerte;
* registrazione ditte e registrazione richiedenti

Il software dovrà inoltre:

* garantire una valutazione equa delle candidature secondo dei criteri stabiliti;
* essere in grado di notificare gli utenti sulle nuove offerte e sullo stato delle candidature in maniera opportuna e senza essere invasivo.

Ove possibile, il software dovrà minimizzare il data-entry, integrandosi con applicativi esistenti. L’applicativo deve essere indipendente da una specifica Università.

L’applicativo dovrà rispettare le norme relative alla sicurezza e privacy nella conservazione dei dati utenti, dovrà essere in grado di garantire che l’utente abbia ricevuto l’invito al colloquio, dovrà permettere l’integrazione con altri sistemi.

## List of Challenging/Risky Requirements or Task

|  |  |  |  |
| --- | --- | --- | --- |
| **Challenging Task** | **Date the task is identified** | **Date the challenge is resolved** | **Explanation on how the challenge has been managed** |
| Matching tra offerte e  candidati | 28/11 | 01/12 | La scelta della struttura del matching è stata sempre al centro dei primi brainstorming riguardanti la strutturazione dell’intero sistema.  Abbiamo superato questa difficoltà utilizzando matching factors per stabilire le correlazioni tra le offerte (“candidatura ottima”) degli offerenti e dei candidati |
| Categorizzazione | 22/11 | 27/11 | Trovare la giusta categorizzazione è cruciale per avere un sistema più snello, intuitivo, ma comunque espressivo.  Abbiamo costruito una categorizzazione partendo dai matching factors ed estrapolando delle macro-categorie che vanno ad identificare dei settori lavorativi. |
| Candidatura ottima | 28/11 | 30/11 | Affinché si possa avere una valutazione equa dei candidati abbiamo riflettuto su cosa fosse un candidato ideale.  Abbiamo risolto il problema identificando una candidatura ottima come una candidatura con punteggio relativo dei matching factors più alto. |
| Web Services | 12/12 | 20/12 | L’utilizzo di web services è stato un punto molto importante che abbiamo affrontato. Abbiamo deciso di definirli e orientarli tenendo conto dell’elevato grado di scalabilità senza appesantire il sistema. |
| Notifiche nuove offerte | 11/12 | 13/12 | il sistema necessita di notificare al candidato le nuove offerte all’accesso. Abbiamo deciso di mettere questa area notifiche nella dashboard e renderla il primo contatto con le offerte |

A. Stato dell’Arte

Durante i primi giorni dalla consegna delle specifiche del progetto (dal 22/11 al 25/11) abbiamo ragionato sullo sviluppo dello stesso, cercando idee ed intuizioni riguardanti il suo funzionamento e su come effettivamente concretizzarle.

Essendo la prima esperienza nello sviluppo di una piattaforma di “cerco/trovo lavoro” abbiamo affiancato al processo creativo un’attenta ricerca in rete per avere un’idea più completa e delineata di ciò che saremo andati a sviluppare.

Abbiamo scelto, tra le varie opzioni online, Linkedin come esempio principe, in quanto permette di creare profili dettagliati attraverso l’elenco delle varie competenze che possono essere aggiunte.

Questo aspetto è stato molto utile poiché ci ha permesso di capire meglio come ideare i matching factors che in seguito definiscono la candidatura ottima di un utente; è stato un buono spunto per la categorizzazione delle offerte in base a competenze e macro-settori lavorativi.

Infine, ci ha aiutato nel comprendere la user experience di una piattaforma di questo genere: essa, senza dubbio, deve essere intuitiva, snella ma comunque espressiva.

Per comprendere appieno le funzionalità di Linkedin abbiamo creato un account, compilato il profilo e svolto alcuni test nel sistema, cercando offerte tramite categorie e in base a diversi ambiti lavorativi.

Un’altra piattaforma che abbiamo analizzato è stato il sito di annunci online “Subito.it”, nello specifico la sezione “Lavoro”.

La principale differenza tra LinkedIn e Subito.it è stata in generale il come avviene l’incontro tra richiedenti lavoro e offerenti e come viene gestito/mediato dal sistema a parità di funzione.

Alla possibilità di specificità e completezza del profilo e dei fattori di confronto di LinkedIn per mediare la connessione tra i candidati e gli offerenti, Subito.it risponde con una generale semplicità e una presenza di molto inferiore nella ricerca e messa in contatto tra offerta e candidatura.

Abbiamo deciso prendere spunto da entrambi gli esempi per ottenere un sistema che tenga conto delle sensazioni di “non-invasività” e “immediatezza” che utenti potrebbero provare ma anche dell’efficienza di un sistema che “funzioni” e funzioni bene.

Fattore chiave su cui abbiamo speso molto tempo è l’approccio dei due sistemi sopracitati al matching tra richiedenti e offerenti.

Abbiamo deciso di incorporare alcuni aspetti comuni ad entrambi i sistemi, come il matching-per-tag di titolo di studi, settore lavorativo e Regione di cui si vuole filtrare i risultati, ed affiancarli con soluzioni che crediamo possano valorizzarli.

Abbiamo intervistato, inoltre, alcuni nostri colleghi provenienti da vari corsi di laurea per capire cosa si aspetta un utente che accede ad una piattaforma di questo genere per la prima volta; abbiamo cosi ottenuto un quadro più oggettivo e di ampio respiro che si scosti dalla nostra propensione tecnica.

Per ottenere delle risposte più facilmente analizzabili abbiamo ideato una serie di domande da proporre ai nostri colleghi in cui richiediamo di definire 3 aspetti fondamentali must-have e funzionalità che a loro parere potrebbero migliorare l’utilizzo del sistema.

I dati ottenuti dalle interviews hanno evidenziato una “naturale” propensione dei nostri colleghi a una semplicità di utilizzo molto simile alla piattaforma Subito.it, da noi già analizzata.

Un altro aspetto emerso dalle risposte è la necessità di alcuni nostri colleghi di avere i contatti ufficiali (e-mail, telefono...) delle aziende offerenti, i requisiti e i dettagli dell’offerta posti in maniera sintetica ed in primo piano.

B. Raffinamento dei Requisiti

*A partire dai servizi minimali richiesti, raffinate la descrizione dei servizi offerti dal vostro applicativo. Descrivete anche I requisiti non funzionali.*

Analisi delle specifiche

**Categorizzazione :**

La categorizzazione è fondamentale per il corretto sviluppo della nostra piattaforma essendo la funzione che ci permette di raggruppare i vari settori lavorativi e indicizzarli ai Candidati con eguale preferenza ( specificata nella compilazione del proprio profilo utente di tipo Candidato).

Inoltre ci permette di “snellire” il sistema e renderlo più intuitivo per gli utenti che non sono costretti a cercare il proprio settore lavorativo tra i vari presenti.

**Elaborazione dati utente :**

I dati dell’utente all’atto dell’iscrizione al sistema saranno raccolti tramite compilazione di campi qualificanti del candidato ( competenze, titolo di studio, settore lavorativo ecc.).

Questi dati saranno utilizzati dal sistema per confrontarli con i dati forniti dagli utenti offerenti ( coloro che pubblicano offerte di lavoro). Questa comparazione darà modo all’utente candidato di poter visionare offerte di lavoro il più possibile simili a quelle che sta cercando.

**Generazione punteggio di matching :**

Il punteggio di matching verrà generato in base ai dati forniti dall’Utente Candidato e dai dati forniti nelle offerte pubblicate(specificati dagli Utenti Offerenti).

La generazione di tale punteggio avverà comparando i dati dell’offerta di lavoro con quelli forniti dall’Utente Candidato: la quantità di parametri rispettati (match) dall’Utente Candidato rispetto all’offerta genererà così un punteggio percentuale di affinità.

**Offerte in primo piano :**

Una volta iscritto al sistema l’Utente Candidato compila i campi del suo Profilo Candidato e specifica le proprie competenze e preferenze lavorative ( inserendo il settore lavorativo).

In base al settore scelto il sistema esporrà all’Utente Candidato le offerte di lavoro, inerenti al proprio settore, visualizzando in cima alla lista quelle con punteggio più alto.

**Candidati in primo piano :**

Una volta pubblicata un’offerta nel sistema, gli Utenti Offerenti potranno visualizzare sulla propria dashboard una sezione ( Gestione Offerte) che comprende l’elenco di tutti gli Utenti Candidati che hanno deciso di iscriversi all’offerta.

Aperta questa sezione l’Utente Offerente può così visualizzare il punteggio di ogni Utente Candidato, in ordine decrescente, e scegliere in base alle proprie esigenze.

**Registrazione Candidato e Offerente :**

Un’utente si iscrive alla piattaforma inserendo le proprie credenziali (username e password) e scegliendo il tipo di profilo che intende creare in base alle proprie esigenze : essi sono Candidato e Offerente ovvero chi cerca offerte di lavoro e chi le inserisce. Una volta effettuato l’accesso, accederanno a differenti dashboard, specifiche per entrambi.

Un utente di tipo Candidato inserisce i propri dati quali competenze e preferenze mentre l’utente Offerente inserirà i dati rispettivi alla figura professionale che cerca.

**Notifiche :**

L’utente Candidato riceverà periodicamente offerte di lavoro che corrispondono il più possibile con le proprie caratteristiche e verrano visualizzate nella propria area Notifiche a cui accede dalla dashboard.

Qui potrà a sua volta visualizzare se le proprie candidature sono state accettate.

A.1 Servizi (con prioritizzazione)

|  |  |  |
| --- | --- | --- |
| FR1 | Ricerca Offerte | Alta |
| FR2 | Candidatura a Offerte | Alta |
| FR3 | Pubblicazione Offerte | Alta |
| FR4 | Gestione Profilo | Alta |
| FR5 | Analisi Candidature | Alta/Media |
| FR6 | Controllo Status Candidature | Alta/Media |
| FR7 | Gestione Offerte Pubblicate | Alta/Media |
| FR8 | Notifica Offerte | Media |
| FR9 | Sign-in | Alta/Media |
| FR10 | Richiesta di Colloquio | Alta |
| FR11 | Chiusura Offerte | Bassa |
| FR12 | Elimina Candidatura | Bassa |
| FR13 | Log-in | Bassa |

* **Log-in**: L’utente accede al sistema immettendo le proprie credenziali (username e password) precedentemente immessi durante la registrazione.
* **Sign-in**: L’utente non registrato al sistema crea un profilo della tipologia a lui congeniale.
* **Gestione Profilo**: L’utente loggato, attraverso la propria dashboard, entra nel suo profilo e nella sezione Gestione Profilo gestisce i propri dati immettendo o modificando le proprie competenze ed esperienze.
* **Ricerca** **Offerte**: La Ricerca Offerte è una funzione del sistema che permette agli utenti di ricercare lavoro in base ai macro-ambiti lavorativi o in base a settori già definiti.
* **Candidatura a Offerte**: La candidatura a Offerte permette ad un utente di candidarsi ad un’offerta presente nel sistema.
* **Nuove offerte**: Le nuove offerte è una funzione che permette all’utente di visualizzare nella propria dashboard le offerte che si avvicinano il più possibile alle competenze dello stesso utente. Vengono notificate all’utente le ultime offerte pubblicate nel settore che ha scelto
* **Controllo Status Candidature**: Il Controllo Status Candidature permette all’Utente Candidato di monitorare l’avanzamento della propria candidatura rispetto ad una determinata offerta.
* **Elimina candidature**: Tale funzione serve ad eliminare una candidatura in caso di errori o impossibilità o nel caso l’offerta relativa viene eliminata.
* **Pubblicazione offerta**: La Pubblicazione Offerta permette alle aziende presenti nel sistema attraverso un account Offerente di pubblicare offerte di lavoro comprensive delle competenze richieste.
* **Gestione offerte pubblicate**: Questa funzione permette ai possessori di un account Offerente di gestire le proprie offerte di lavoro pubblicate e modificare in caso fosse necessario.
* **Analisi candidature**: Questa funzione permette all’Utente Offerente di monitorare le eventuali candidature per una data offerta di lavoro pubblicata precedentemente.
* **Richiesta di colloquio**: Una volta analizzate le possibili candidature, l’offerente può scegliere le candidature più consone per ricoprire il ruolo che cerca e richiedere il colloquio ai candidati scelti.
* **Chiusura offerte**: Scelte le candidature ed inviate le notifiche di colloquio ai candidati scelti, l’offerente può chiudere le offerte da lui pubblicate eliminandole dal sistema.

A.2 Requisiti non Funzionali

* **NFR1 Dependability** **- fault tolerance:** Il sistema deve continuare ad offrire i servizi anche se si è persa la funzionalità di una componente.
* **NFR2 Dependability - security:** Il sistema deve essere in grado di proteggersi da attacchi esterni, accidentali o intenzionali. La security diventa un requisito essenziale, visto che il nostro sistema dovrà essere connesso alla rete per permettere l’accesso ai dati di offerte e candidature nonché dei dati personali di offerenti e candidati
* **NFR3 Scalabilty**: Il sistema garantisce un’architettura scalabile per supportare future espansioni (es. web, mobile).
* **NFR4 Usability**: Il sistema deve garantire un’esperienza user-friendly e risultare il più intuitiva possibile
* **NFR5 Performance**: il sistema deve risultare efficiente e deve lavorare con tempi di esecuzione accettabili
* **NFR6 Privacy**: Il trattamento dei dati personali deve rispettare le norme vigenti in materia nei paesi in cui opererà.
* **NFR7 Equa valutazione**: La valutazione del candidato da parte del sistema deve risultare equa e improntata su nient’altro che le abilità che il candidato pone nel suo profilo

A.3 Scenari d’uso dettagliati

**Scenario 1: Iscrizione e gestione profilo Candidato**

1. L’utente si registra inserendo nome utente, password e i dati del suo profilo.
2. Una volta registrato l’utente accede al sistema tramite le credenziali e premendo il pulsante “log-in”.
3. Dopo il log-in, viene caricata la dashboard utente del tipo selezionato all’atto dell’iscrizione (sono possibili 2 tipi di account: Candidato e Offerente).
4. Dalla Dashboard utente di tipo Candidato si possono effettuare diverse operazioni:
   * la gestione del profilo
   * la ricerca delle offerte di lavoro
   * la gestione delle candidature
   * controllare le notifiche.

**Scenario 2: Iscrizione e gestione profilo Offerente**

1. L’utente di tipo Offerente segue una procedura simile d’iscrizione dell’utente Candidato, ma orientata all’offerta di lavoro che pubblicherà nel sistema.
2. Dalla Dashboard di tipo Offerente si potranno svolgere diverse funzioni tra le quali:
   * la pubblicazione delle offerte
   * la gestione delle offerte pubblicate
   * l’analisi delle candidature con eventuale richiesta di lavoro.

**Scenario 3: Pubblicazione di un’offerta**

1. L’utente di tipo Offerente, dalla propria Dashboard, clicca sul pulsante “Pubblica Offerta”
2. Una volta cliccato si apre la sezione della pubblicazione delle offerte, in cui l’Offerente compila un modulo dove specifica una breve descrizione dell’offerta, compila i campi delle competenze richieste e specifica l’ambito lavorativo (Macro-Categoria)
3. Una volta pubblicata l’offerta attraverso il pulsante “Pubblica”, essa sarà visibile nella Dashboard “Le Mie Offerte”.

**Scenario 4: Ricerca dell’Offerta**

1. L’utente Candidato ha due modi per accedere alle Offerte ovvero:

* Dalla propria Dashboard nell’area notifiche (queste saranno le offerte presenti nel sistema che fanno match con il “Punteggio” massimo, calcolato sui matching factors)
* nell’area Ricerca Offerte.

1. Nell’Area Ricerca Offerte il Candidato potrà cercare tra le Offerte di lavoro tramite l’ambito lavorativo a lui congeniale.

**Scenario 5: Richiesta di Colloquio Offerente**

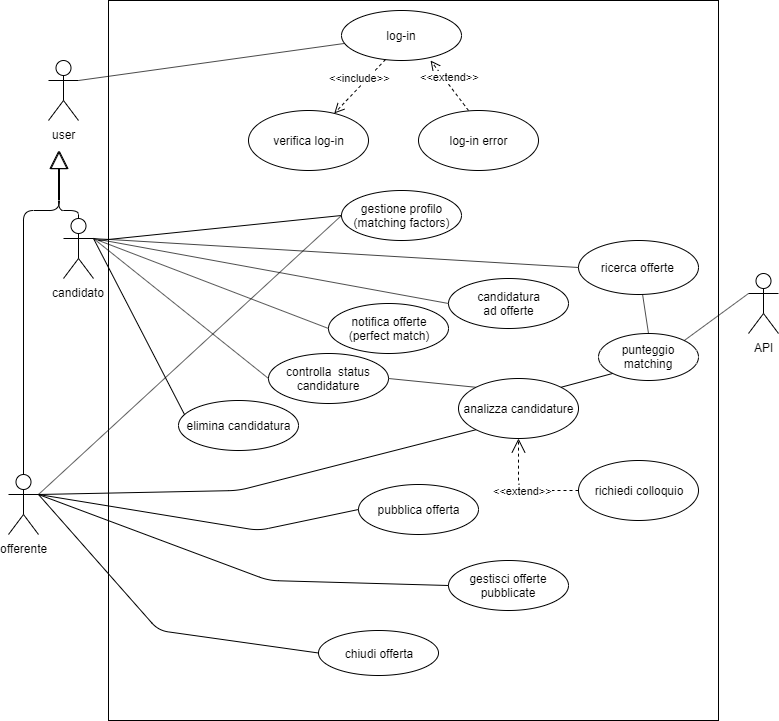
1. L’Offerente, tramite la Dashboard “Le mie Offerte”, analizzerà i profili dei candidati e sceglierà, in base alle proprie esigenze, i candidati da invitare a colloquio attraverso l’apposito pulsante “Richiedi Colloquio”.
2. Tale azione produrrà una Notifica di “Richiesta di Colloquio”, la quale sarà presente nella Dashboard del candidato scelto dall’Offerente.

A.4 Excluded Requirements

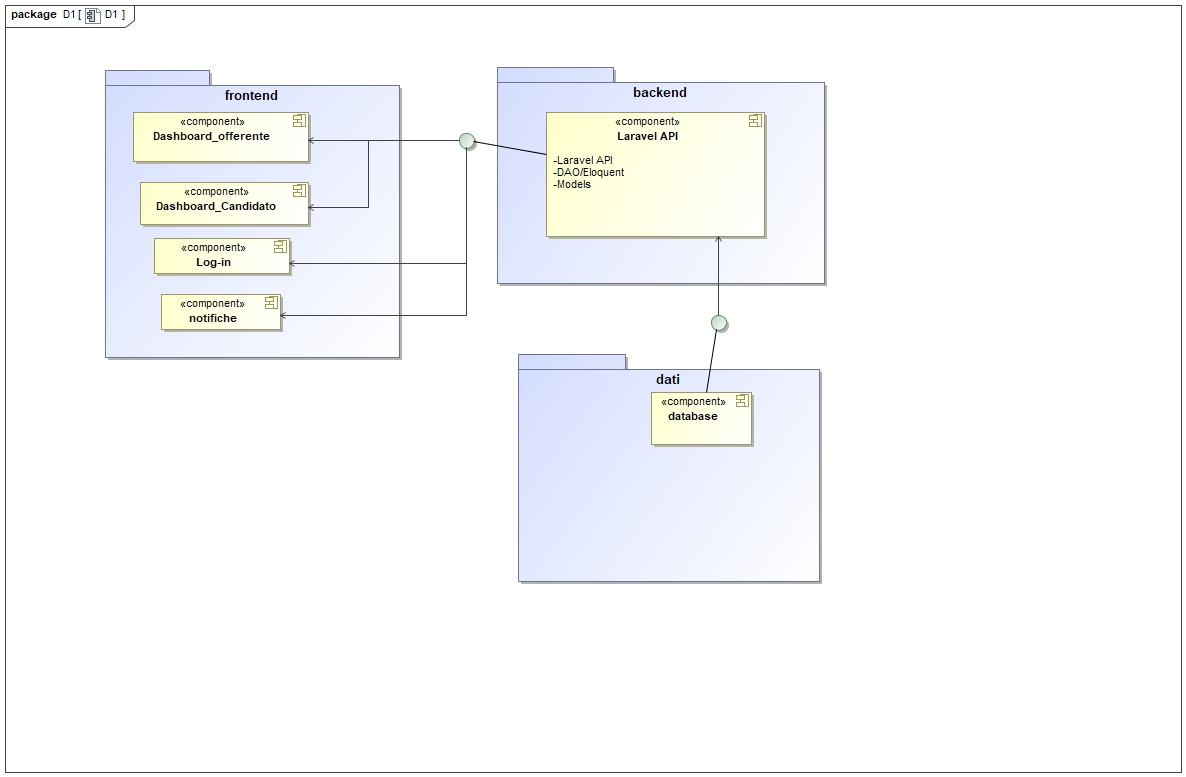
* Abbiamo escluso il servizio di invio del curriculum e il servizio di corrispondenza elettronica in-app in favore di un “Profilo/curriculum” sul nostro sistema e un servizio di notifiche per rendere più efficace il matching, più snello il sistema in generale e meno invasivo. La corrispondenza tra Offerenti e Candidati e l’eventuale invio del curriculum non verrà coperto dal sistema evitando ulteriori potenziali minacce a livello privacy.
* Il database non verrà gestito o manutenuto da noi; le specifiche dell’hardwere verranno indicate dal nostro team, mentre i costi dello stesso e della manutenzione saranno a carico dello stakeholder.

A.5 Assunzioni

* Potrà utilizzare il sistema solo un utente registrato
* Assumiamo che l’utente accetti le condizioni di conservazione dei dati personali a noi necessarie per il funzionamento del sistema
* Assumiamo che le leggi in vigore per il trattamento dei dati personali siano rispettate dal sistema
* I dati degli utenti sono opportunamente immagazinati nel database

A.6 Use Case DiagramsC. Architettura Software

C.1The static view of the system: Component Diagram



Il Component Diagram è un diagramma che ha lo scopo di rappresentare la struttura interna del sistema software, modellando i componenti principali e le relazioni fra di essi. Nel dettaglio i componenti presenti nel sistema:

**Nella parte di frontend**

**Log-in**: che permette all’utente di accedere al sistema. Prende i dati dal componente User. (FR12)

**Dashboard Candidato e Dashboard Offerente**: componente che permette all’user di accedere ai servizi del sistema a seconda del tipo di user (offerente o richiedente). Prende i dati dal relativo profilo per associargli le corrette operazioni a seconda dell’user

**Mostra Candidature sulle Offerte**: Componente che serve a visualizzare le candidature degli utenti Candidati ad una data Offerta (FR5). Usa i dati di Offerta, Candidatura e Offerente.

**Sign in**: Componente che permette all’utente di iscriversi alla piattaforma.

acquisisce i dati dell’Users. All’atto dell’iscrizione verra chiesto se si vuole un account offerente o un account per candidarsi alle offerte e questa informazione verrà salvata nel campo flag (FR9).

**Ricerca/Mostra Offerte**: Componente il cui ruolo è quello di permettere all’Utente Candidato di ricercare e visualizzare le Offerte all’interno del sistema (FR1). Usa i dati di Profilo\_Candidato e Offerta.

**Nella parte di backend**

**Profilo\_Offerente**: model che identifica lo specifico profilo Offerente (NFR6/NFR7).

Prende i alcuni dati dal componente user offerente (un Profilo ha uno e un solo User)

**Profilo\_Candidato**: model che identifica lo specifico profilo Candidato (NFR6/NFR7). Prende i dati dal componente Candidato (un Profilo ha uno e un solo User

**User** : model che identifica l’utente Offerente o Candidato, ovvero colui che pubblicherà offerte nella piattaforma (NFR6/NFR7) o colui che ricercherà offerte e si candiderà ad esse(FR2/FR10).

**Offerta**: model che identifica le offerte pubblicate nel sistema(FR2). I dati di questa componente vengono utilizzati da Mostra Candidature sulle Offerte e Ricerca/Mostra offerte. Acquisisce dati dal componente Offerente.

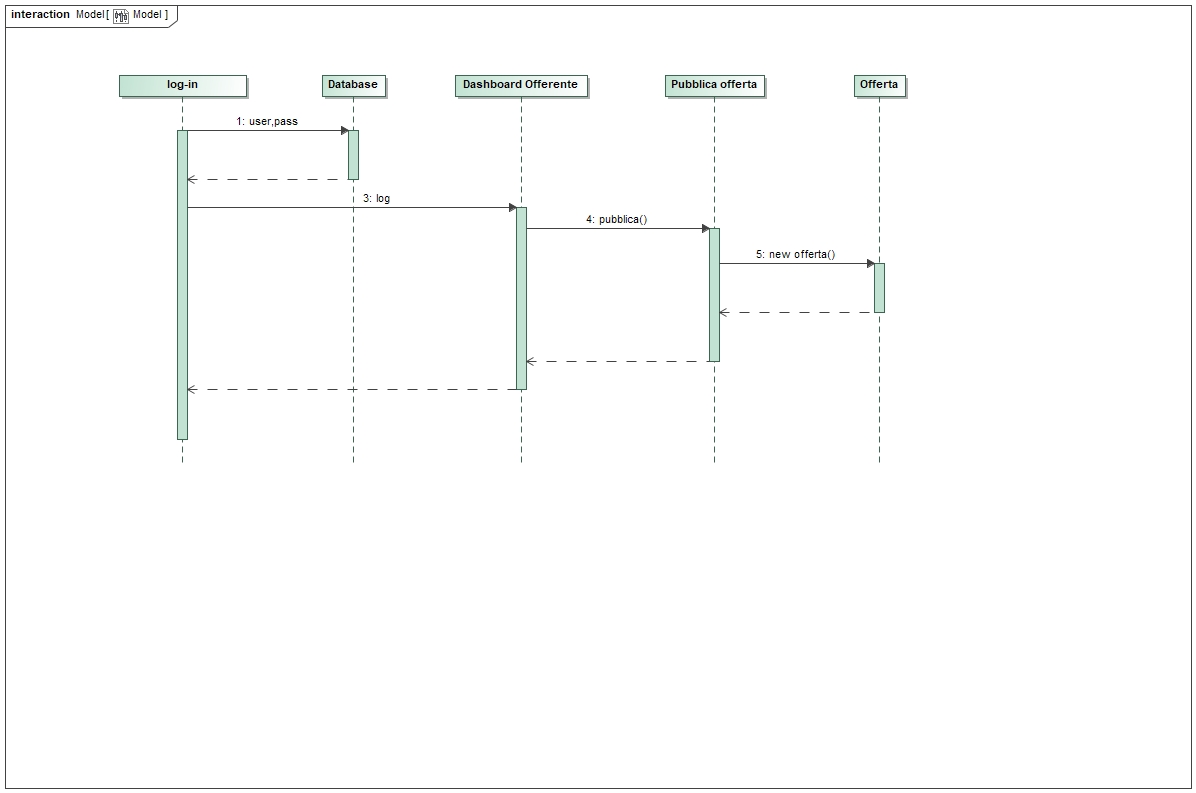
**Candidatura**: model che identifica le candidature degli utenti Candidato alle Offerte presenti nella piattaforma (FR2/FR5). Acquisisce dati da Candidato, Profilo\_Candidato e Offerta. Fornisce dati a Mostra Candidature sulle Offerte e Notifiche.

**Notifiche**: model che serve ad avvisare gli Utenti sullo stato delle Candidature e delle Offerte (FR8). Acquisisce dati da Offerta, prof. Offerente, prof. Candidato e Candidatura.

I dati che i model appena mensionati utilizzano vengono gestiti da dei relativi controller come di uso ne pattern MVC. In questi controlle risiedono i metodi DAO e altri metodi per la gestione dei dei model.

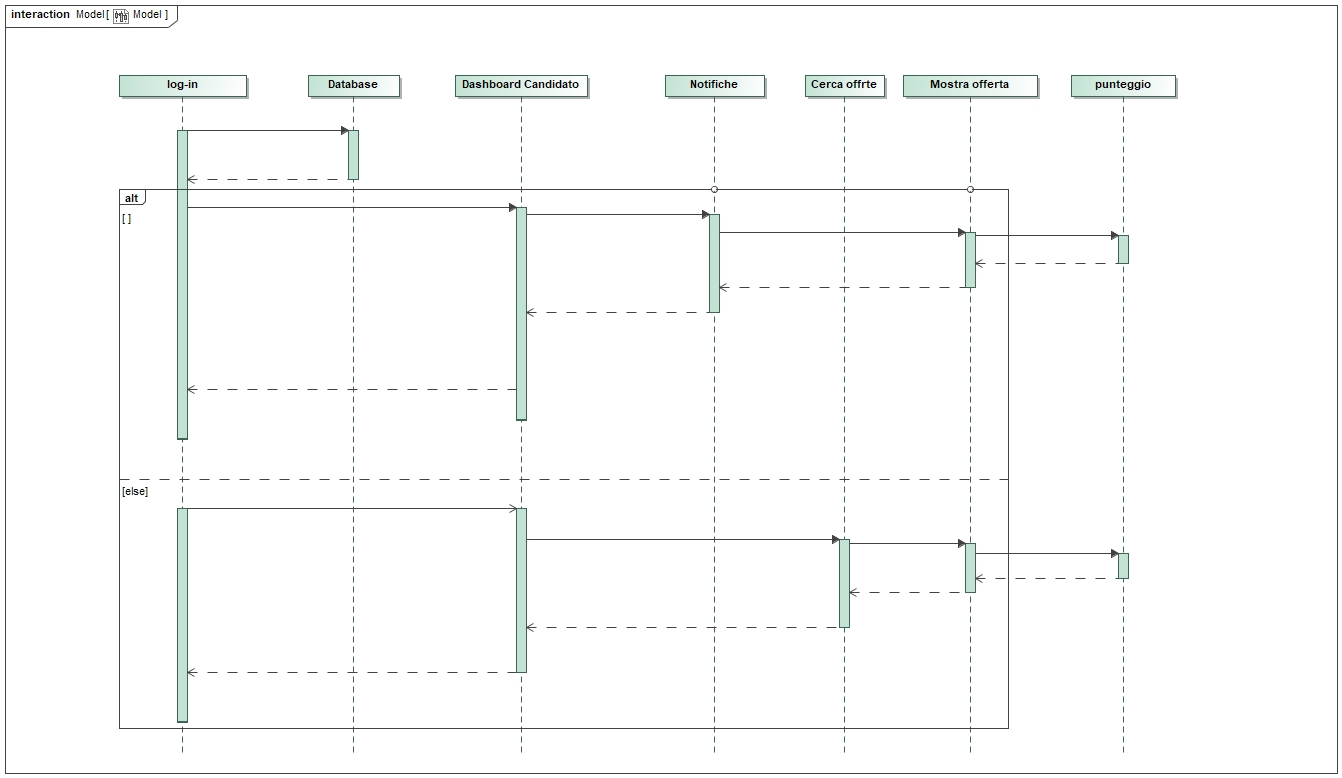
Nel database invece ci sono tutti i dati che vengono salvati e recuperati dai vari metodi del backend (come user per il log-in passando per il DAO dell’API ad esempio)

***C.2 The dynamic view of the software architecture: Sequence Diagram***



Scenario 3:

1. L’Utente Offerente effettua il Log-in: esso inserisce username e passaword del proprio account interpella il database che verifica le credenziali e fa acceder l’utente al sistema.
2. Effettuato l’accesso con il Log-in, l’utente accede alla propria dashboard.
3. Attraverso la dashboard Offerente seleziona la sezione Pubblica Offerta.
4. Una volta aperta la sezione Pubblica Offerta, crea una nuova offerta.



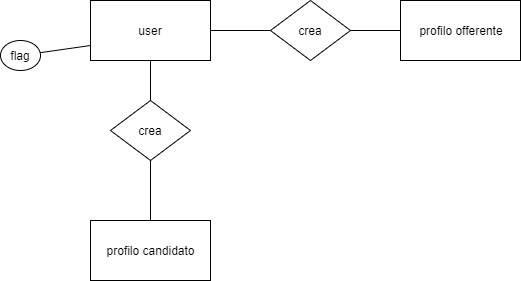
Scenario 4:

1. Una volta effettuato il log in ( Descrizione Scenario 3), l’Utente Candidato accede alla dashboard Candidato
2. Nella dashboard Candidato l’Utente potrà vedere lo stato delle sue candidature nella sezione Notifiche e visualizzare l’offerta, attraverso Mostra offerte.
3. Altrimenti, l’Utente Candidato ricerca l’offerta e in seguito visiona l’offerta tramite Mostra Offerta.

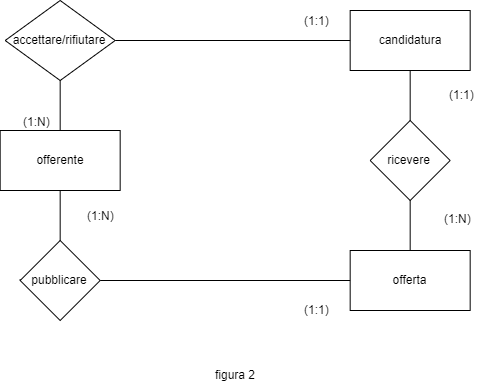
D. Dati e loro modellazione

*Definite le sorgenti di dati a voi necessarie per realizzare I servizi di cui sopra. Modellat tali dati tramite un ER o similari. Specificate se e quali di tali dati sono gia’ forniti da applicativi esistenti.*

Come accennato nel component diagram e, come spiegeremo nella sezione di design decision, la nostra architettura presenta due fondamentali generalizzazioni che sono interconnesse in maniera univoca allo scopo di rappresentare i due aspetti dell’utilizzarore della piattaforma: il suo lato pubblico (profilo) e i suoi dati privati (utente).



Ad un profilo (qualunque tipologia esso sia) corrisponde uno ed un solo utente (qualunque tipologia esso sia dettato dal flag:[0=candidato:1=offerente]) e viceversa. I dati per queste due entità verranno presi durante la registrazione del profilo dopo il sign-in e verranno suddivisi in base alla sensibilità dei dati nelle 2 categorie (personali in tipo-utente, pubbliche in tipo-profilo)



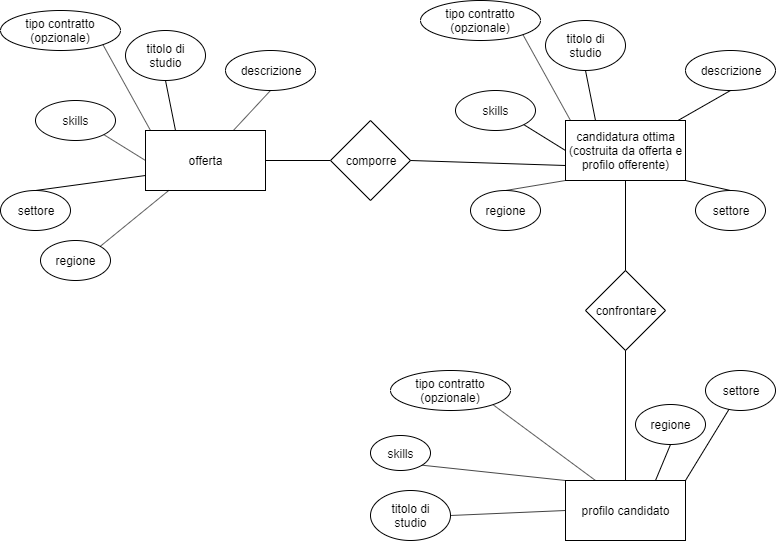
Lo schema ER della figura 2 descrive il comportamento dell’Offerente all’atto di pubblicazione di un’offerta.

L’Offerente può pubblicare un’offerta ( da una a N offerte) e tale offerta potrà ricevere candidature da parte dagli utenti Candidati ( da 1 a N candidature). Infine l’Offerente può accettare o rifiutare una candidatura ad un’offerta pubblicata.

Immagine che contiene testo

Descrizione generata automaticamente

Lo schema ER in figura 3 descrive il comportamento dell’utente Candidato nell’inviare una Candidatura per una data Offerta. Il Candidato sceglie un’offerta presente nel sistema (da 1 a N) e invia la sua univoca Candidatura per tale Offerta.



Il matching avviene tra due entità con stessi attributi: l’entità candidatura ottima (profilo-Candidato fittizzio creato a partire dall’offerta e il relativo offerente) e il profilo del candidato. Ovviamente ad una coppia di offerta e offerente appartiene una sola candidatura ottima che verrà confrontata con le candidature per identificare il punteggio di matching. Integrativamente ai campi di skills e requisiti che vengono chiesti al candidato l’offertente può scrivere una breve descrizione del lavoro.

E. Design Decisions   
<Document here the **5** most important design decisions you had to take. You can use both a textual or a diagrammatic specification.>

**Functional requirements:**

* Decomposizione funzionale: il sistema viene scomposto in funzionalità, ordinato per operazione e decomposto in altri piccoli moduli;
* Decomposizione object-oriented: il sistema viene scomposto in classi, ogni classe è un’entità importante nel dominio dell’applicazione, le classi possono essere scomposte in piccole classi.

Scelta:

Il team ha deciso di optare per la decomposizione funzionale perché ritenuto più consono per svolgere tale progetto.

**Non-functional requirements:** il team ha deciso di dar più importanza ai seguenti requisiti non funzionali essendo essenziali per il corretto funzionamento del sistema:

* Dependability;
* Scalability;
* Usability;
* Performance.

**Trattamento dei dati personali e equità della valutazione:** Per garantire una valutazione equa e al contempo una migliore gestione dei dati personali abbiamo deciso di dividere l’entità generalizate di Utente (offerente e candidato) e l’entità generalizzate di profilo (Profilo-offerente e Profilo-candidato); le entità Utente-like gestiranno i dati sensibili e personali (email e password) degli utenti mentre le entità Profilo-like gestirà tutto ciò che verrà utilizzato per fare maching, dati comuni e non molto indicativi della persona ma specifici per una posizione lavorativa (skills, tipologia di lavoro, tipo di contratto...).

I dati sensibili non saranno parte del processo di matching: alcuni dati sensibili come e-mail e numero di telefono saranno disponibili all’offerente ma non saranno campi obbligatori o nel caso dell’email si potrà inserire un indirizzo per il log-in ed uno di “lavoro” che sarà visibile.

**Matching:** Come accennato nella sezione D e nel paragrafo di questa sezione dedicato al trattamento dei dati personali e dell’equità della valutazione il confronto tra richiesta di lavoro e offerta dello stesso viene gestita tra i profili di offerenti, i profili candidato e le offerte stesse per i dati relativi alle capacità che il candidato deve avere per il ruolo richiesto.

Abbiamo deciso per un matching di questo genere perché ci è sembrato il metodo che si incastonava meglio nel contesto di un’architettura che fa da mediatore super partes tra richiedenti e offerenti lavoro.

Suggerendo non solo offerte relative al settore scelto dal candidato (con relativo punteggio) ma anche valutando offerte di altri settori su richiesta del user candidato, il sistema non pone limiti sulle candidature (come un limite di una certa percetuale di matching per inviare una candidatura): più parametri e skills posti dall’offerente fanno matching, più alto è il punteggio ma detto punteggio è un aiuto visivo più che una soglia di sbarramento.

**Programming languages Decision:** il team ha deciso per lo sviluppo del software di utilizzare Laravel per motivi che affronteremo nella parte del prototipo come metodi e scaffolding che ci hanno aiutato nella creazione del prototipo.

**Mvc pattern decision:** il team ha deciso di impiegare il “pattern mvc” per rappresentare l’architettura del software e il class diagram ad esso collegato. Tale decisione e’ stata concordata per organizzare il sistema in modo ottimale e logico, dividendolo quindi in tre parti. Siamo cosi’ in grado di facilitare la scalabilità e la manutenzione dell’applicazione. L’MVC Pattern (Model-View-Controller) é un Pattern Architetturale molto diffuso nello sviluppo software, in particolare nell’ambito della programmazione orientata ad oggetti.

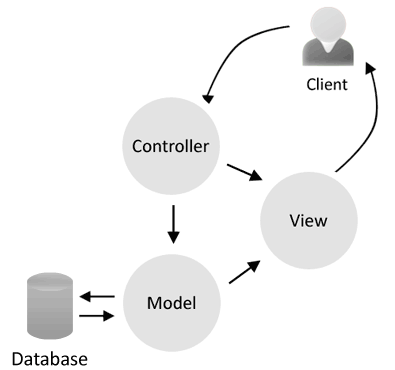
Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

Model: fornisce i metodi per accedere ai dati utili all'applicazione

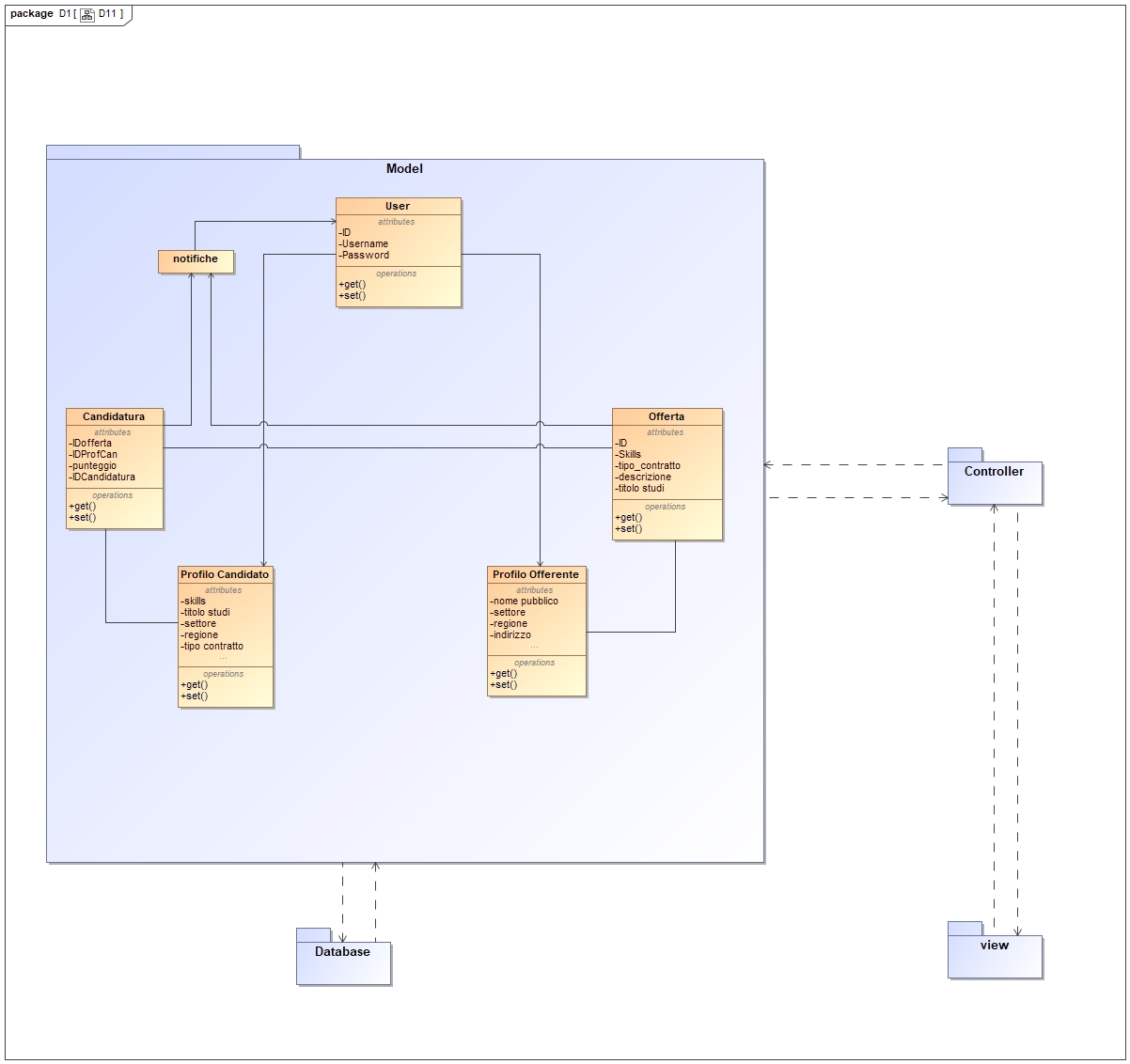
View: visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti

Controller: riceve i comandi dell’utente attraverso il View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato del View.

La struttura architetturale a tre livelli dell’MVC permette un ottima integrazione con servizi web a mediare l’accesso dei dati al database e facilità di espasioni future del sistema



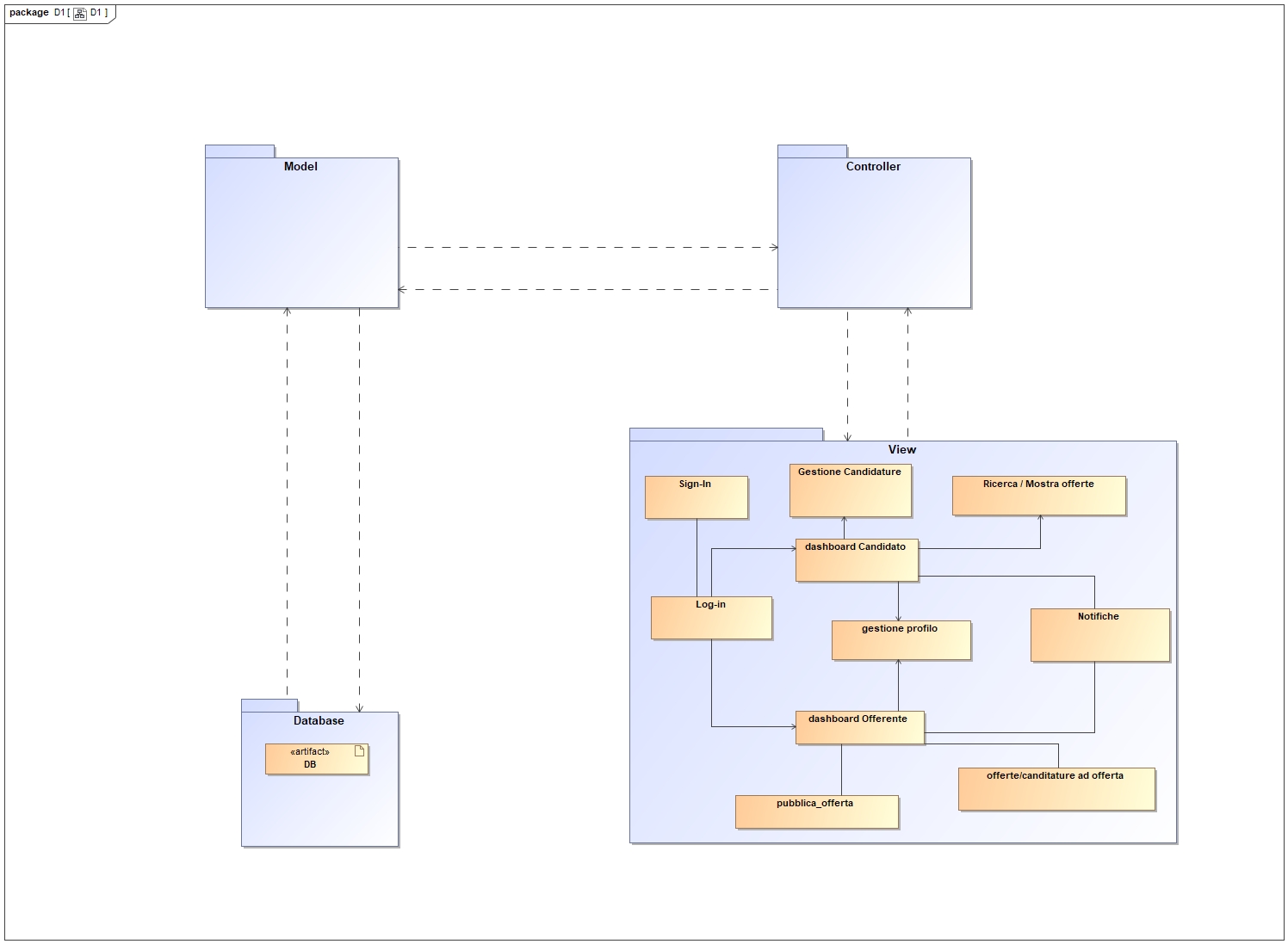
F. Design di Basso Livello



Abbiamo rappresentato semplicemente le classi principali che caratterizzano il nostro sistema. Ognuna di esse è caratterizzata dal proprio nome, una lista di attributi e metodi. Abbiamo rappresentato 4 componenti interconnesse tra loro: Model, DataBase, Controller e View.

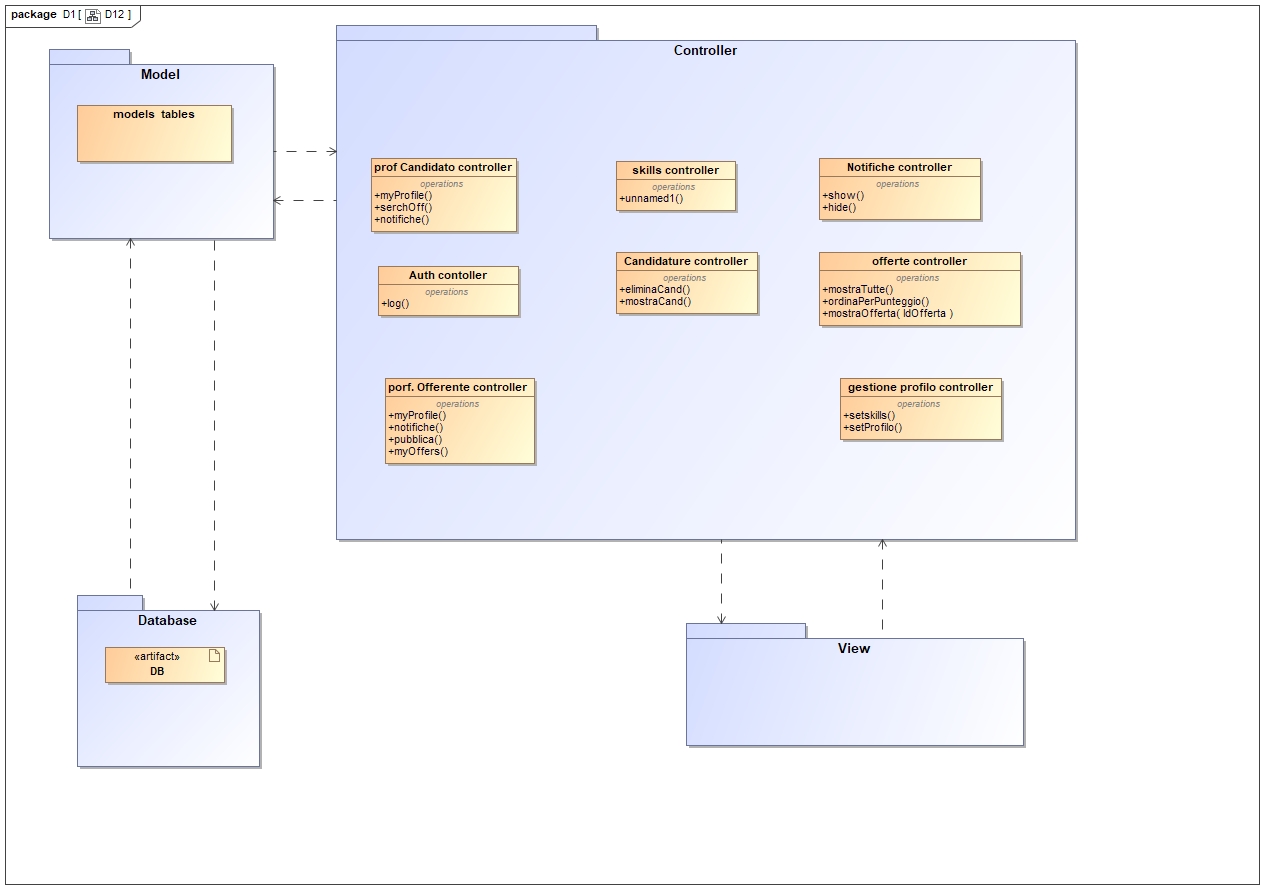
All’interno di Model troveremo le classi che caratterizzano il nostro Sistema: esse sono collegate attraverso associazioni.

Model è collegata al Database ( che contiene i dati relativi agli Utenti iscritti al sistema, offerte, candidature...) dal quale prende i dati che necessita e li modella nella forma necessaria al sistema. Model è anche collegato a Controller che regola e indirizza lo scambio dei dati e delle operazioni tra model e il presentation layer view.



Il package View descrive l’interfaccia del nostro sistema.

Infatti, qui troveremo le dashboard degli utenti Offerenti e Candidati che sono costituite da sezioni quali: log-in, sign-in gestione del Profilo, pubblica Offerta, gestione Candidature e Notifiche. Le classi di Controller gestiranno ogni click che l’utente effettuerà in ogni interfaccia presente nel View ed effettuerà le chiamate alle classi del Model che contengono le classi e i metodi necessari per il recupero e la gestione dei dati del database



In controller, come già mensionato, potremo trovare le classi e i metodi che prenderanno dalle view i comandi dell’utente e li comunicheranno al model (modifiche, recupero, cancellazione dei dati).

G. Explain how the FRs and the NFRs are satisfied by design

*<Report in this section how the architectural and low level design you produced satisfies the FRs and the NFRs>*

Il team ha costruito l’architettura del sistema continuando a modellarla e migliorarla giorno dopo giorno alternando fasi di analisi e studio a fasi di lavoro hands-on. I **requisiti funzionali** trovati e raffinati durante le fasi di learning sono stati soddisfatti dalle scelte di design, architetturali e nel design di basso livello e descritti nelle sezioni precedenti. In particolare nelle sezioni precedenti abbiamo esplorato matching, registrazione, log-in, candidatura alle offerte e approvazione di quest’ultime.

**Gli NFR1, NFR2 e NFR3** vengono soddisfatti dal modello di architettura e dalle scelte che il team ha tenuto in considerazione utilizzate per la realizzazione del sistema; l’utilizzo del pattern MVC e dei web services garantiscono un approccio più semplice al sistema in caso di malfunzionamento, una struttura facilmente comprensibile in caso di espansione del sistema e un più sicuro accesso ai dati

**L’NFR4** viene garantito dalla View precedentemente descritte che fornisce un’interfaccia semplice ed userfriendly.

**L’NFR5** è garantito dall’architerrura del sistema che abbiamo deciso di rendere snella e performante (come nel caso della strutturazione del matching come visto nelle sezioni precedenti).

**L’NFR6** è garantito dalla scelta di design di far accettare agli utenti iscritti il documento sul trattamento dei dati personali e grazie al fatto che i propri dati sensibili non vengono visualizzati (es. possibilità di loggarsi con una e-mail ma renderne pubblica, sul proprio profilo, un’altra e-mail ”di lavoro” o non essere obbligati a mettere il numero di telefono o campi troppo personali tipo indirizzo etc.) dagli altri utenti presenti all’interno

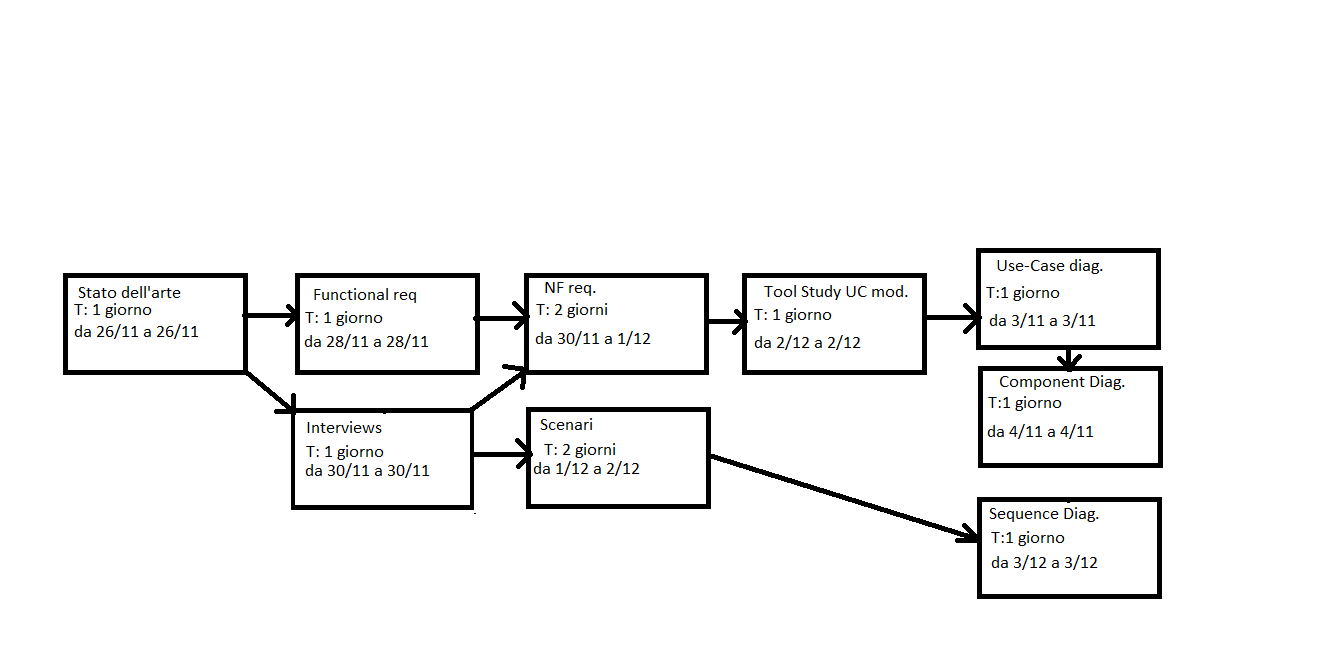
del sistema come approfondito nelle sezioni E e G.

**L’NFR7** è rispettato dalle scelte di design adottate dal team sulla valutazione della Candidatura rispetto ad una offerta (verrà valutata in base ad un punteggio di Matching Factors valutato dal sistema in base alla corrispondenza tra competenze ricercate dall’Offerente e competenze specifiche del Candidato) come spiegato nella sezione E dedicata al trattamento dei dati personali e all’equità della valutazione.

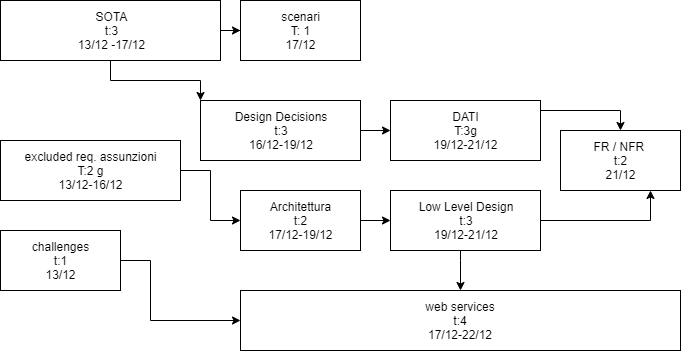
G. Effort Recording

***PERT****Make a PERT documenting the tasks and timing you expect to spend on the deliverable. Try to be as precise as possible. Check, after the deliverable deadline, if and how you satisfied (or not) the deadlines.*

*Milestone-pert*



*D1-pert*

**

***Logging*** *As you are working on the assignment, record what you are doing and how long you spent. As a rule of thumb, you should add a log entry every time you switch tasks. For example, if you do something for two hours straight, that can be one log entry. However, if you do two or three things in half an hour, you must have a log entry for each of them. You do not need to include time for logging, but should include the time spent answering the other parts of this question.*

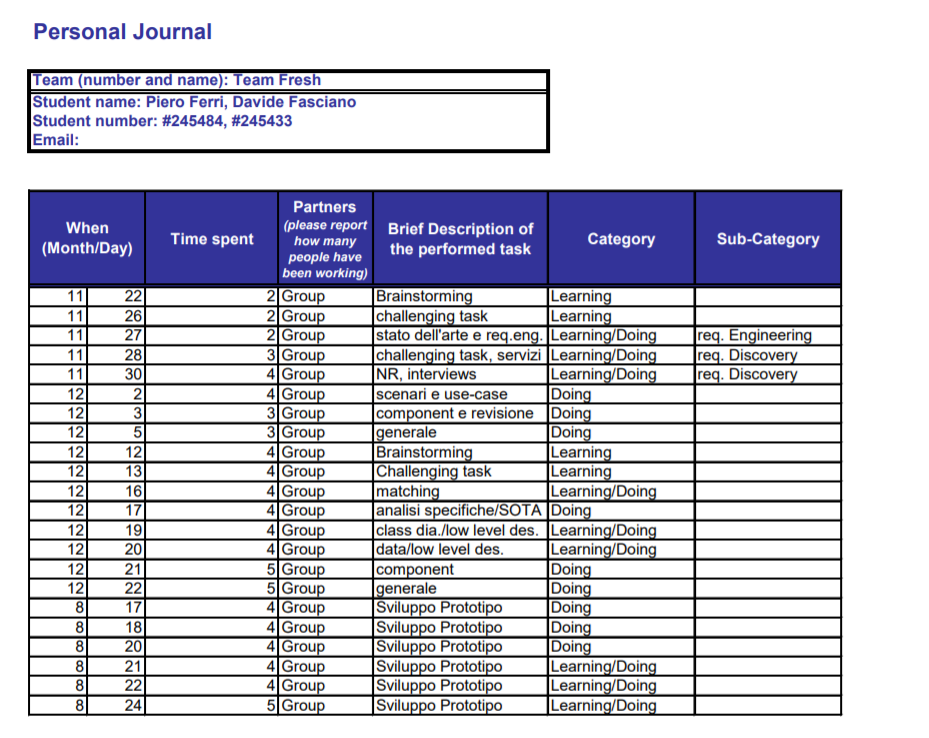
*For this purpose, please use the* ***LogTemplate.xls*** *file.*

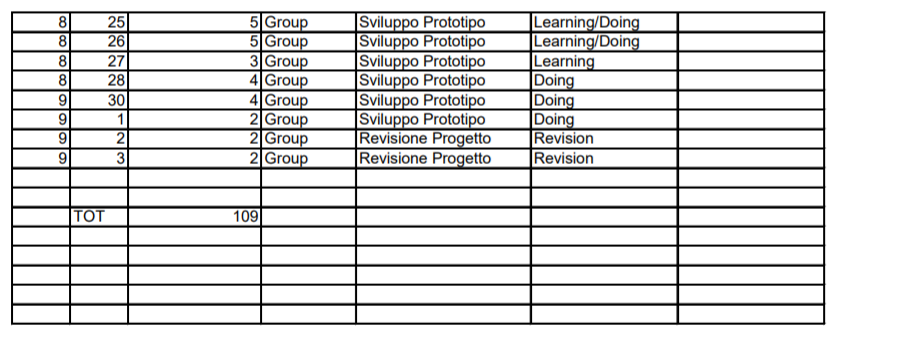
***Categorization*** *When logging the time spent on the project, please create different sub- categories.**Specifically, it is important to clearly distinguish between two main categories: the time spent for “****learning****” (the modeling languages, the tools, etc.) from the time needed for “****doing****” (creating the models, taking the decisions, …). Learning tasks are in fact costs to be paid only once, while doing costs are those that will be repeated through the project.*

*For each category, please define sub-categories. Examples follow. You may add other sub-categories you find useful.*

|  |  |
| --- | --- |
| ***Learning***   * ***Requirements Engineering*** * ***Non functional Requirements*** * ***Use Case Diagrams*** * ***Tool study*** | ***Doing:***   * ***Requirements discovery*** * ***Requirements Modeling (UC diagrams)*** |

***Summary Statistics****Based on the attributes defined above, calculate the summary statistics of the time spent for “learning”, the time spent for “doing”, and the total time.*

******



***TOTALE\_ORE: 109***

Appendix. Prototype   
*<Provide a brief report on your prototype, and especially: information on what you have implemented, how the implementation covers the FR and NFR, how the prototypes demonstrates your project correctness with respect to the FR and NFR. You may add some screenshots to describe what required above. Be ready to show your prototype during the oral examination>*

# Definizione web service

I dati che faremo gestire al nostro servizio web saranno ovviamene quelli inseriti negli ER Diagram. La suddivisione in sotto-domini per entità principali sembra essere la scelta più logica e adatta poiché il framework Laravel rende la gestione delle tables del database molto semplice out-of-the-box

Assumendo che avremo il nostro {{url}} parametrizzato avremo:

{{url}}/users

{{url}}/profili\_off

{{url}}/profili\_can

{{url}}/offerte

{{url}}/candidature

{{url}}/skills

Le request saranno effettuate su questi sotto-domini per accedere ai dati del database, fare il matching sugli stessi e qualche utility: la documentazione della collection di request da noi creata può essere trovata qui

<https://documenter.getpostman.com/view/9807106/SWLYAqoJ?version=latest>

con variabili di environment ‘url’ che sostituisce localhost/api.

Laravel inoltre fornisce scaffolding per tutto quello che riguarda il log-in, le autorizzazioni, recupero password, registrazione etc. che abbiamo modificato per essere utilizzate con la nostra architettura.

Postman offre oltre al servizio di pubblicazione della documentazione anche uno strumento per il testing che abbiamo utilizzato per un check molto generico i cui risultati sono nel file json Fresh.postman\_test\_run.json più che altro concentrandoci sui tempi di risposta delle route con più dati da processare come quelle in cui avviene il matching.

Per una lista delle route

Route::get('/users', 'User@index');

Route::get('/users/{id}', 'User@show');

Route::post('/users', 'User@store');

Route::delete('/users/{id}', 'User@destroy');

Route::put('/users/{id}', 'User@edit');

Route::patch('/users/{id}', 'User@update');

Route::post('/login', 'User@login');

Route::get('/offerte', 'OffertaCtrl@index');

Route::get('/offerte/{id}', 'OffertaCtrl@show');

Route::post('/offerte', 'OffertaCtrl@store');

Route::delete('/offerte/{id}', 'OffertaCtrl@destroy');

Route::put('/offerte/{id}', 'OffertaCtrl@edit');

Route::patch('/offerte/{id}', 'OffertaCtrl@update');

Route::get('/offerte/{id}/skills', 'OffertaCtrl@skilloff');

Route::get('/offerte/{id}/candidature', 'OffertaCtrl@candidature\_a\_offerta');

Route::get('/offerte/{id}/candidature/candidati', 'OffertaCtrl@candidati\_a\_offerta');

Route::get('/candidature', 'CandidaturaCtrl@index');

Route::get('/candidature/{id}', 'CandidaturaCtrl@show');

Route::post('/candidature', 'CandidaturaCtrl@store');

Route::delete('/candidature/{id}', 'CandidaturaCtrl@destroy');

Route::get('/notifiche', 'UserNotifyCtrl@index');

Route::get('/notifiche/{id}', 'UserNotifyCtrl@show');

Route::post('/notifiche', 'UserNotifyCtrl@store');

Route::delete('/notifiche/{id}', 'UserNotifyCtrl@destroy');

Route::get('/skills', 'skillCtrl@index');

Route::get('/skills/categorie/all', 'skillCtrl@showcategorie');

Route::get('/skills/categorie/{cat}', 'skillCtrl@showskillsbycat');

Route::get('/skills/list', 'skillCtrl@indexlist');

Route::get('/skills/{id}', 'skillCtrl@show');

Route::post('/skills', 'skillCtrl@store');

Route::delete('/skills/{id}', 'skillCtrl@destroy');

Route::put('/skills/{id}', 'skillCtrl@edit');

Route::patch('/skills/{id}', 'skillCtrl@update');

Route::get('/profili\_off', 'profilo\_off\_Ctrl@index');

Route::get('/profili\_off/{id}', 'profilo\_off\_Ctrl@show');

Route::post('/profili\_off', 'profilo\_off\_Ctrl@store');

Route::delete('/profili\_off/{id}', 'profilo\_off\_Ctrl@destroy');

Route::put('/profili\_off/{id}', 'profilo\_off\_Ctrl@edit');

Route::patch('/profili\_off/{id}', 'profilo\_off\_Ctrl@update');

Route::get('/profili\_off/{id}/candidature', 'profilo\_off\_Ctrl@showcanbyid');

Route::get('/profili\_off/{id}/candidature/candidati', 'profilo\_off\_Ctrl@showcandidati');

Route::get('/profili\_can', 'profilo\_can\_Ctrl@index');

Route::get('/profili\_can/{id}', 'profilo\_can\_Ctrl@show');

Route::post('/profili\_can', 'profilo\_can\_Ctrl@store');

Route::delete('/profili\_can/{id}', 'profilo\_can\_Ctrl@destroy');

Route::put('/profili\_can/{id}', 'profilo\_can\_Ctrl@edit');

Route::patch('/profili\_can/{id}', 'profilo\_can\_Ctrl@update');

Route::get('/profili\_can/{id}/matchingofferte', 'profilo\_can\_Ctrl@match\_offerte');