Corso di Intelligenza Artificiale

29 marzo 2021

Suggerimento di contenuti tramite Prolog in un contesto di E-Learning

Studente
Marco Mirizio

Docente
Prof. Stefano Ferilli

Indice

1 Obiettivo

Presentare nuovi contenuti sotto forma di suggerimento agli utenti di una piattaforma online di e-learning, basati sulle loro preferenze passate.

2 Realizzazione

- Aggiunta di una componente ad un sistema di e-learning open-source che permetta l'estrazione di dati presenti sulla piattaforma e il dialogo con un'interprete Prolog;
- Calcolo di uno score in Prolog tra Learning Object ed interessi dello studente sulla base di modelli di relazioni tra tag definiti sui Learning Object;
- Fornire come consigli all'interno della piattaforma stessa i Learning Object che più si avvicinano ai gusti dello studente.

3 Dominio: E-Learning

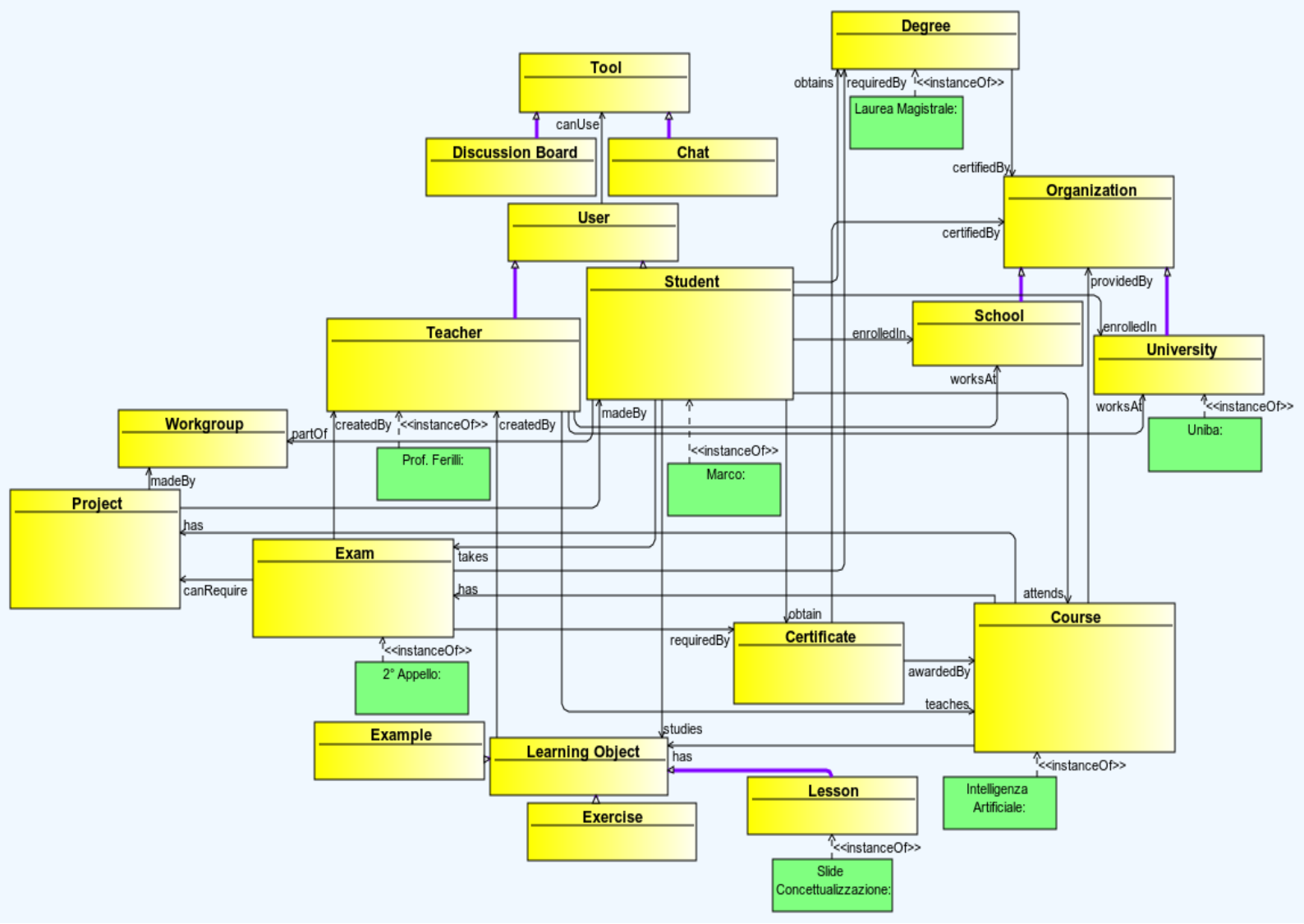
Con il termine E-Learning si identifica l'unione tra il mondo della tecnologia e di Internet con quello dell'apprendimento, allo scopo di migliorarne la qualità e la fruibilità.
Questo modo di fornire educazione online ha da subito un chiaro vantaggio sulla didattica normale per quanto riguarda l'accesso ai contenuti. Infatti i requisiti minimi richiesti per seguire corsi erogati in modalità di E-Learning sono una connessione ad Internet ed un device (personal computer, smartphone) in grado di collegarsi alla piattaforma desiderata. Successivamente sarà possibile usufruire del materiale didattico in qualsiasi momento ed in qualsiasi luogo, senza vincoli di orario o di presenza fisica.
I corsi erogati in modalità E-Learning possono valorizzare i contenuti proposti su una più ampia dimensione multimediale, integrando in maniera più diretta tecnologie e strumenti che servono ad aumentare l'interattività del materiale didattico per favorire una migliore comprensione dell'argomento trattato. Questi corsi vengono organizzati in Learning Object (moduli didattici), pensati per essere svolti in un breve lasso di tempo e che rispondano ad un concreto obiettivo formativo, per una personalizzazione completa delle modalità di approccio alla formazione da parte dello studente. Chiaramente è importante non dimenticarsi della parte di interazione umana docente/studente e studente/studente che è necessaria per la creazione di un sano ambiente di apprendimento.

La gestione della distribuzione dei corsi E-Learning è affidata a delle piattaforme software chiamate LMS (Learning Management System). Queste piattaforme generalmente sono in grado di tenere traccia delle statistiche degli utenti, come contenuti utilizzati e tempo di utilizzo e mettono a disposizione strumenti ulteriori di supporto alla didattica sincrona ed asincrona. In modalità sincrona vi sono strumenti quali l'aula virtuale, la lavagna condivisa o la possibilità di effettuare lezioni in videoconferenza e chat in tempo reale con gli studenti. In modalità asincrona, invece, vengono messi a disposizione forum di discussione, mail e metodi di condivisione file per appunti o qualsiasi altro tipo di materiale.
Inoltre i più moderni software LMS facilitano la gestione dei Learning Object fornendo direttamente sulla piattaforma strumenti adatti alla loro creazione o importazione da altri software.

4 Concettualizzazione Generico Sistema E-Learning

Analizzando le componenti che fanno generalmente parte di un contesto di erogazione di corsi in modalità E-Learning tramite un software LMS troviamo in primis tutti gli strumenti che la piattaforma mette a disposizione, come forum di discussione, chat oppure strumenti per la creazione di Learning Object etc..
Poi ci sono le organizzazioni, tipo scuole o Università che gestiscono la piattaforma in sé e quindi decidono quali corsi rendere disponibili in modalità online e che quindi garantiscono a livello legale le competenze o le certificazioni acquisite dagli studenti in caso di completamento del percorso di formazione online.
Infine come utenti della piattaforma troviamo da un lato gli studenti che si iscrivono appunto per usufruire dei corsi online, dall'altro i docenti che invece sono responsabili della creazione dei Learning Object del corso, quindi lezioni, esercizio, test e tutto il materiale necessario agli studenti per la materia.

Un esempio di concettualizzazione delle entità e delle loro relazioni appena descritte all'interno di un sistema di E-Learning.


Degree
Tool
obtains
requiredBy
<<instanceOf>>
Laurea Magistrale:
canUse
Discussion Board
Chat
certifiedBy
Organization
User
certifiedBy
Student
providedBy
School
enrolledIn
enrolledIn
Teacher
University
worksAt
worksAt
<<instanceOf>>
Uniba:
Workgroup
partOf
createdBy
<<instanceOf>>
createdBy
madeBy
<<instanceOf>>
Prof. Ferilli:
Marco:
madeBy
Project
has
Exam
takes
canRequire
has
attends
obtain
Course
requiredBy
Certificate
<<instanceOf>>
awardedBy
2° Appello:
teaches
Example
studies
Learning Object
has
<<instanceOf>>
Lesson
Intelligenza Artificiale:
Exercise
<<instanceOf>>
Slide Concettualizzazione:

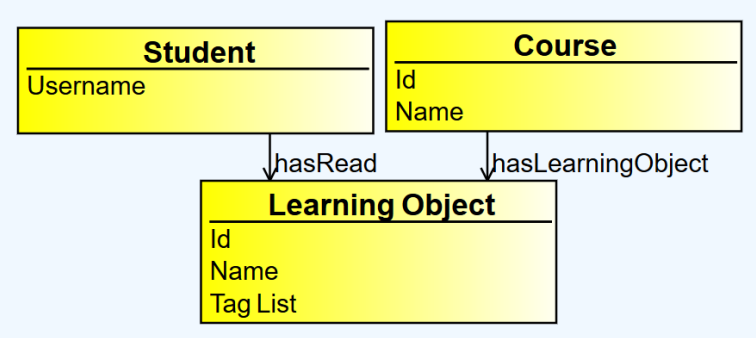
5 Applicazione

5.1 Scelta del Software

A seguito della ricerca effettuata per l'elaborazione della relazione sui Learning Managment Systems, allegata insieme a questo documento, è stato scelto come software LMS OpenOLAT.

5.2 Modello ristretto al contesto reale

Adattando il modello di concettualizzazione generico descritto in precedenza al software scelto ed al ristretto scope di classi necessarie al nostro obiettivo possiamo soffermarci nello specifico ad analizzare le relazioni che ci sono tra queste tre classi del nostro sistema di E-learning.



- Uno Student caratterizzato da un Username può aver letto un Learning Object
- Un Course caratterizzato da un Id univoco e da un Name è formato da diversi Learning Object
- Un Learning Object è caratterizzato da un Id univoco, un Name ed una Tag List

Estrapolando le relazioni tra queste tre classi dal database della piattaforma OpenOLAT, generiamo i fatti per la nostra Knowledge Base Prolog utilizzata come base dai predicati per arrivare all'individuazione dei Learning Object da consigliare.
Quindi in generale l'obiettivo è consigliare ad uno Student dei Learning Object con cui non è in relazione hasRead sulla base di quelli con cui lo è.

5.3 Relazioni tra Learning Object

I tag contenuti nelle liste che caratterizzano i Learning Object vengono associati all'oggetto dal suo autore, teoricamente il docente che lo crea, e indicano l'argomento dei contenuti trattati (informatica, matematica, etc..) e/o il modo in cui vengono presentati (lezione, video, etc..).
Presi due Learning Object possiamo quindi trovare dei gradi di relazione tra di essi analizzando le loro liste di tag.

5.3.1 Definizione dei tag in OpenOLAT

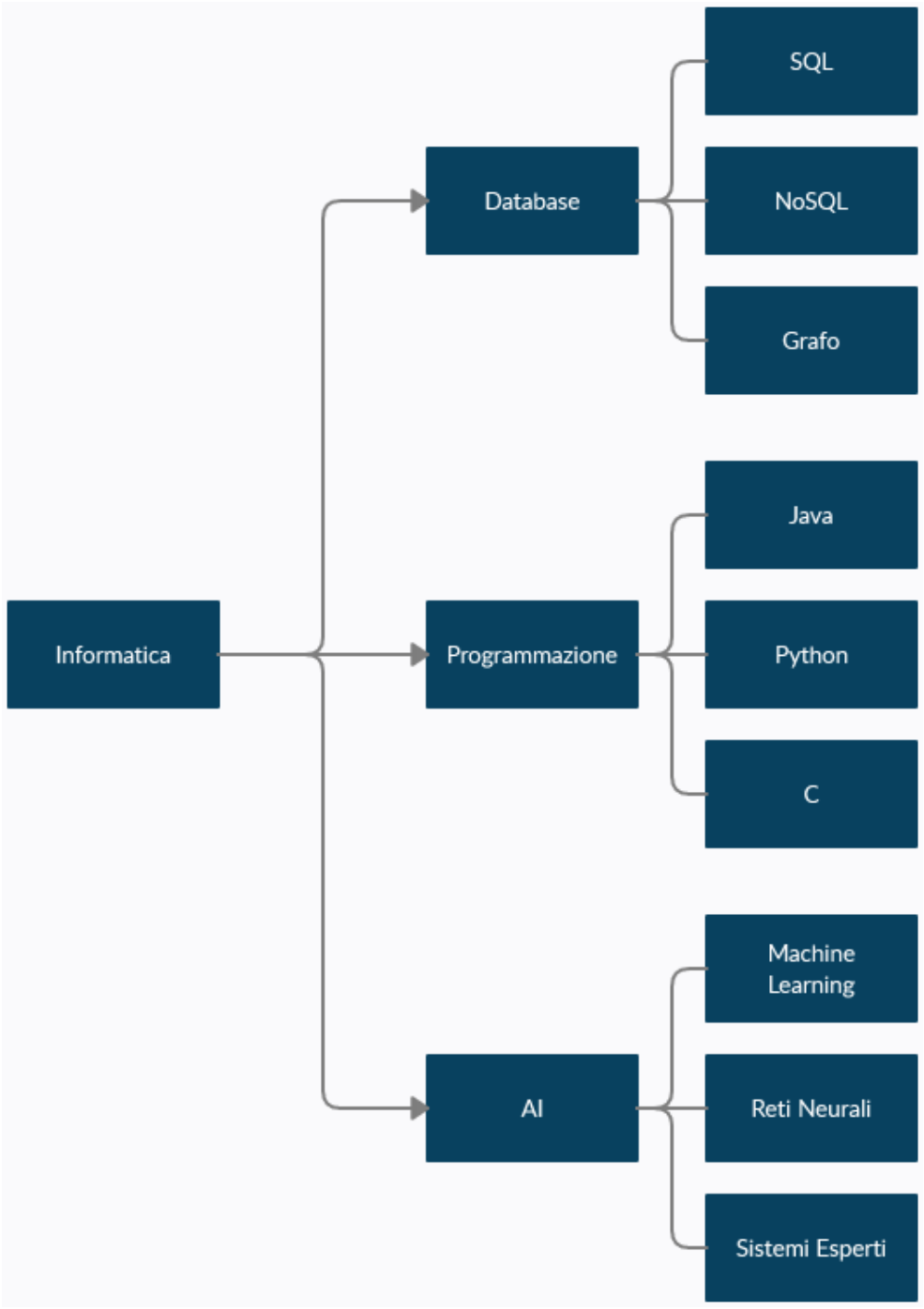
La nostra idea di caratterizzare i Learning Object con dei tag, non è prevista delle funzionalità messe a disposizione da OpenOLAT. Per cui abbiamo pensato di inserire i tag nella sezione "Description" dei Learning Object, indicandoli con un "#". Inoltre i tag non possono contenere spazi.

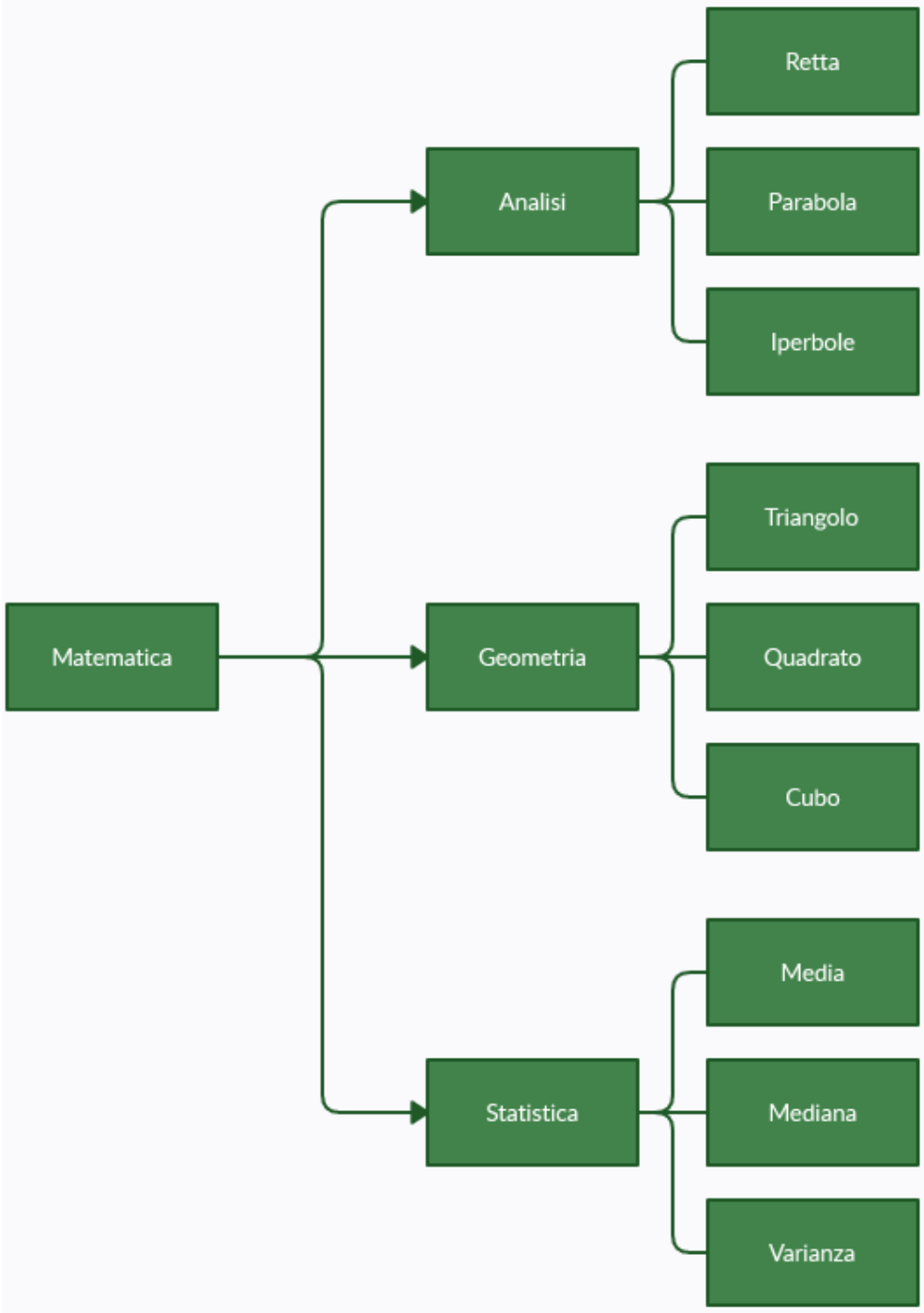
Per il corretto funzionamento del codice di questo progetto è necessario definire a priori le relazioni tra i tag che possono essere assegnati ai Learning Object. Per questo sono stati creati dei piccoli esempi di relazioni sul dominio dell'informatica, della matematica e della medicina, sotto forma di modelli ad albero che vanno da argomenti più generici ad argomenti più specifici.

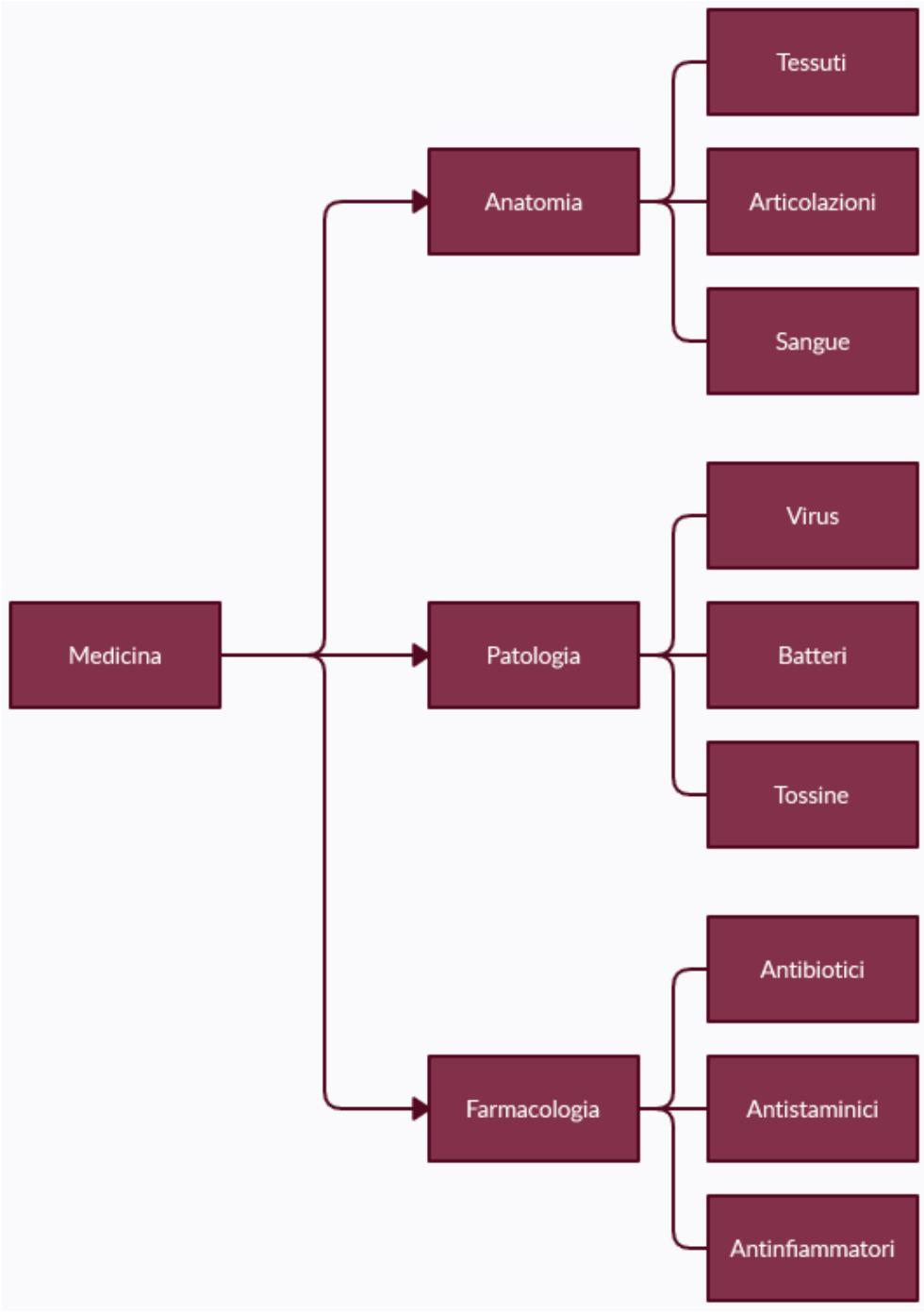
Nonostante ciò potrà sicuramente rivelarsi interessante in futuro provare ad esplorare il comportamento dei suggerimenti prodotti da questo progetto utilizzando come modello di relazione tra tag qualche ontologia ben definita su questo tipo di dominio.

5.3.2 Relazioni tra tag definite nel progetto

Mostrate nei seguenti schemi vi sono le semplici relazioni tra tag definite in questo progetto.


Informatica
Database
SQL
NoSQL
Grafo
Programmazione
Java
Python
C
AI
Machine Learning
Reti Neurali
Sistemi Esperti


Matematica
Analisi
Retta
Parabola
Iperbole
Geometria
Triangolo
Quadrato
Cubo
Statistica
Media
Mediana
Varianza


Medicina
Anatomia
Patologia
Farmacologia
Tessuti
Articolazioni
Sangue
Virus
Batteri
Tossine
Antibiotici
Antistaminici
Antinfiammatori

5.4 Suggerimento dei contenuti

I Learning Object vengono proposti agli utenti in due pagine della piattaforma, la pagina principale dove vi è la lista di tutti i corsi e nelle pagine specifiche di ogni lezione.
Nella pagina principale vengono consigliati dei Learning Object in base all'attività passata dell'utente, nella pagine delle lezioni singole invece vengono consigliati dei Learning Object vicini al oggetto della pagine corrente senza tenere conte delle preferenze dell'utente, un po' come la differenza tra i video consigliati nella nostra Homepage di Youtube ed i video consigliati al lato di un video specifico nella sua pagina.
Questi suggerimenti sono il risultato del calcolo in Prolog di uno score. Questo score viene calcolato tra due liste di tag con il predicato scoreTaglistTaglist(X,Y,S). Se si vuole calcolare lo score tra due Learning Object basta semplicemente calcolare lo score tra le loro due liste di tag, mentre nel caso si voglia calcolare lo score tra un utente ed un Learning Object bisogna "ridurre" l'utente ad una lista di tag contenente l'unione di tutti i tag dei Learning Object visti in precedenza.
Riusciamo quindi in entrambi i casi ad utilizzare lo stesso predicato scoreTaglistTaglist semplicemente effettuando differenti operazioni preliminari, fornite dai predicati scoreObjectObject e scoreUserObject.

5.4.1 Distanza tra due tag

La distanza tra due tag è data dal numero N di salti che bisogna fare nell'albero delle relazioni per raggiungere partendo da un tag l'altro.
Ovviamente la distanza è 0 nel caso in cui i due tag siamo uguali, mentre non esiste se tra i due tag non c'è nessuna relazione.
Nelle liste di distanza calcolate dai predicati Prolog la non esistenza della distanza viene indicata simbolicamente con il valore di -1.

Ad esempio basandoci sulle relazioni delle tabelle di prima la distanza tra i tag database e sql sarà uguale ad 1, oppure quella tra tessuti e batteri sarà 4, infine la distanza tra geometria e farmacologia non esiste e sarà assegnato il valore -1.

5.4.2 Distanza tra due liste di tag

Per dare un valore concreto alla distanza tra due liste di tag abbiamo scelto di basarci sull'indice di Jaccard modificandone il comportamento adattandolo ai nostri bisogni.

L'indice di Jaccard [1] [2] è uno strumento semplice per dare un valore alla distanza tra due insiemi, calcolando il rapporto tra la cardinalità della loro intersezione e la cardinalità della loro unione.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

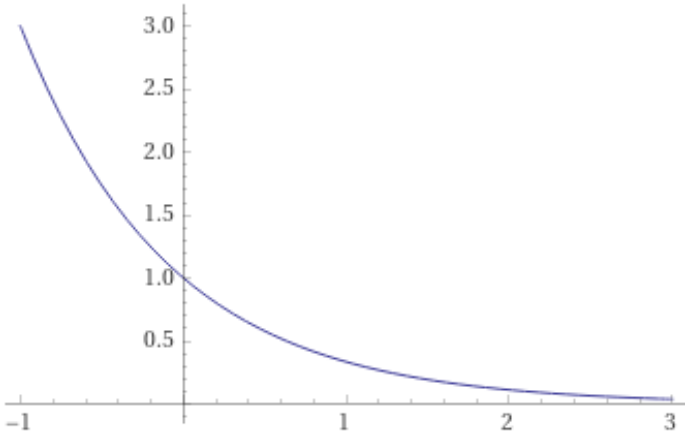
Per esempio due liste tipo [a,b,c,d] e [b,c,e] avranno come indice di Jaccard $2/5 = 0{,}4$. Ovviamente se le liste risultano uguali l'indice sarà uguale a 1 mentre se non vi è nemmeno un elemento in comune l'indice sarà 0.

Nel nostro caso però non è sufficiente identificare solamente i tag uguali, abbiamo anche bisogno di dare importanza ai tag che sono in relazione tra di loro. Per questo abbiamo bisogno di rilassare il vincolo dell'intersezione. La distanza da noi proposta in questo progetto si ottiene calcolando il rapporto tra la somma delle distanze tra i tag in relazione ed il loro numero.

5.4.3 La funzione score

La funzione score è definita con la formula $Y=1/a^X$ con $a>0$. Le proprietà fondamentali che sfruttiamo in questa funzione sono:

- Se X=0 allora Y=1
- La Y è asintotica a 0 per X che tende a $+\infty$



[1] A. Gardner, J. Kanno, C. A. Duncan, S. Rastko - Measuring Distance Between Unordered Sets of Different Sizes

[2] K.J. Horadam, M.A. Nyblom - Distances between sets based on set commonality

Queste proprietà non sono influenzate dal parametro a, che regola solamente la forma della curva. In questo progetto è stato utilizzato il valora a = 3.
Il valore X che viene dato in input a questa funzione è la distanza tra due liste di tag descritta precedentemente. Questa distanza non può mai essere negativa quindi lo score non potrà mai essere maggiore di 1. Lo score sarà 1 nel caso in cui le due liste siano identiche quindi aventi distanza 0. Più aumenta il valore della distanza più basso sarà lo score calcolato da questa funzione.

5.4.4 Il parametro di penalizzazione

Il risultato della funzione score verrà moltiplicato per un valore di penalizzazione P = 1-R^b, dove R è il rapporto tra il numero di tag non relazione nelle due liste e il numero totale di tag, mentre b è un parametro che influenza il peso della penalizzazione. In questo progetto è stato scelto il valore b = 0.5, ovvero $\sqrt{R}$.
Nel caso in cui tutti i tag siano in relazione tra loro, R sarà 0 quindi non verrà applicata nessuna penalizzazione allo score, sarà infatti moltiplicato per 1.

5.4.5 Calcolare lo score

Il primo passo per calcolare lo score è quello di sfoltire le due liste di tag, lasciando solo i tag "foglia" ovvero i tag più specifici.
Successivamente dalle due taglist ridotte si vanno a togliere i tag in comune, sostanzialmente si calcola l'intersezione tra queste due liste e poi si sottrae ad entrambe. In seguito si calcola la lista delle distanze tra i tag tra ognuna delle taglist ridotte e l'altra lista meno l'intersezione. Queste due liste di distanze vengono concatenate e vengono aggiunti tanti elementi di valore 0 quanti sono gli elementi dell'intersezione calcolata in precedenza. Da questa lista di distanze vengono sommati gli elementi >=0 ed il risultato diviso per il loro numero. Il risultato di questa operazione viene dato come input alla funzione score Y=$1/3^X$. Questo score viene moltiplicato per un fattore di penalizzazione 1-$\sqrt{R}$, dove R è il rapporto tra il numero di tag non in relazione ed i tag totali.

Esempio pratico del calcolo dello score

1. Prendiamo due taglist:
    - A = [database,sql,matematica,varianza]
    - B = [sql,farmacologia,statistica]

2. Le taglist ridotte saranno:
    - AR = [sql,varianza]
    - BR = [sql,farmacologia,statistica]
3. Calcoliamo l'intersezione:
    - I = [sql]
4. Sottraiamo l'intersezione dalle taglist ridotte:
    - AS = [varianza]
    - BS = [farmacologia,statistica]
5. Calcoliamo le distanze tra le taglist AR-BS e BR-AS:
    - AD = [-1,-1,-1,1]
    - BD = [-1,-1,1]
6. Concateniamo le liste delle distanze AD e BD e aggiungiamo tanti 0 quanti sono gli elementi dell'intersezione:
    - D = [-1,-1,-1,1,-1,-1,1,0]
7. Calcoliamo la distanza da noi proposta:
    - DJ = 1+1+0/3 = 0,66666
8. Calcoliamo la funzione di score:
    - Y = $1/3^{0,666}$ = 0,4807
9. Calcoliamo la penalizzazione da applicare:
    - R = 5/8 = 0,625
    - P = 1 - $\sqrt{R}$ = 1 - 0,791 = 0,209.
10. Applichiamo la penalizzazione calcolata:
    - S = 0,4807 * 0,209 = 0,100
11. Lo score tra le taglist [database,sql,matematica,varianza] e [sql,farmacologia,statistica] è 0,100.

6 Documentazione Tecnica

6.1 Java

Le classi Java implementate si trovano nel package ai.core (src/main/java/ai/core)

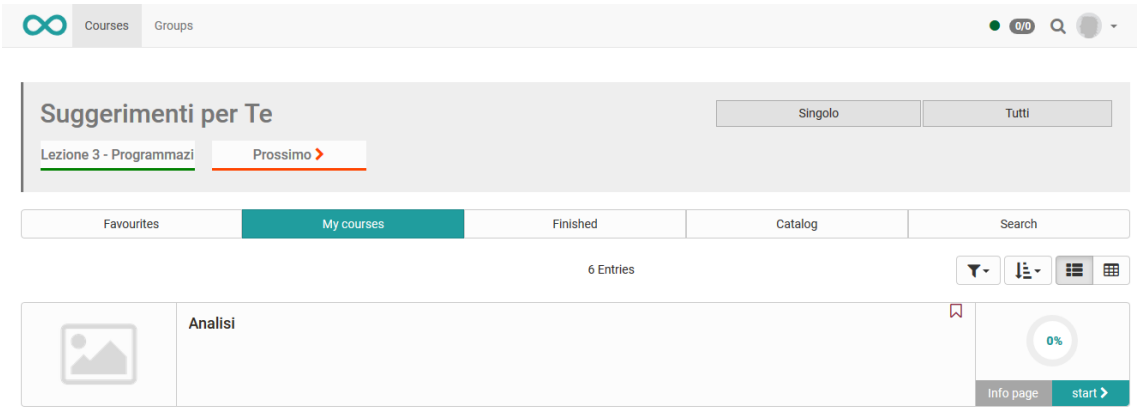
- **DBScraper** - Contiene le funzioni per leggere dal database e dai file xml i dati necessari alla costruzione della Knowledge Base Prolog
- **PrologEngine** - Contiene le funzioni necessarie per inizializzare e interagire con il Prolog
- **PrologQuery** - Contiene le query che si possono fare alla base di conoscenza
- **RefreshPrologJob** - CronJob che si occupa di rigenerare la Knowledge Base constantemente (tempo default impostato ad 1 minuto, modificabile nel file src/main/java/org/olat/core/commons/services/scheduler/_spring/schedulerContext.xml)
- **Suggerimento** - Classe base Java con getter e setter per convertire in oggetti Java i suggerimenti ottenuti in output dal Prolog

Le funzioni definite in questi file sono poi state inserite nel codice originale del software per andarne a modificare il comportamento.

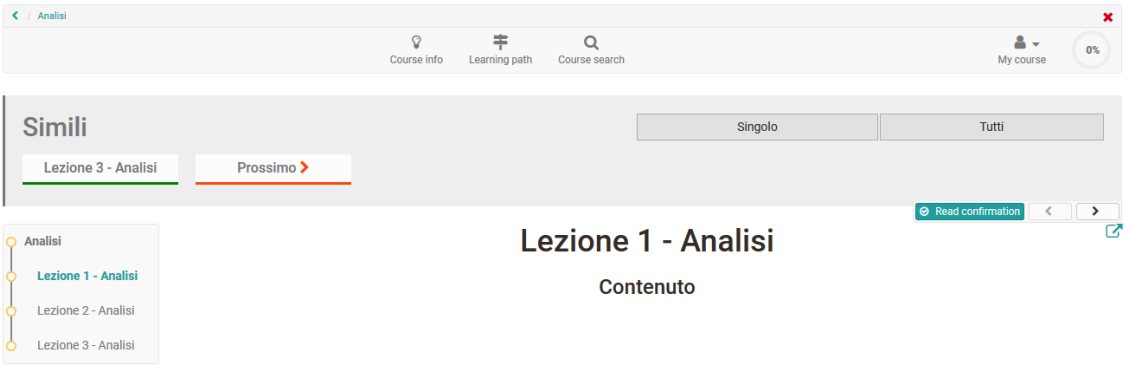
In particolare, ma non solo, i file :

- src/main/java/org/olat/repository/ui/list/OverviewRepositoryListController.java
- src/main/java/org/olat/repository/ui/list/_content/overview.html
- src/main/java/org/olat/course/run/RunMainController.java
- src/main/java/org/olat/course/run/_content/index.html

- **OverviewRepositoryListController** - Controller della pagina con l'elenco dei corsi a cui è stata aggiunta la visualizzazione dei corsi suggeriti all'utente in base alle sue preferenze passate



- **RunMainController** - Il controller della pagina delle lezioni nei corsi modificata con l'aggiunta della visualizzazione dei suggerimenti relativi ai tag della stessa lezione



6.2 Prolog

Il codice Prolog è suddiviso nei seguenti file

- fatti.pl - Contiene la Knowledge Base estrapolata dal database
- predicati.pl - Contiene tutte le funzioni prolog
- relazioni.pl - Contiene le relazioni tra tag da noi definite

6.2.1 Fatti

La Knowledge Base su cui agiscono i predicati, che viene estrapolata dal database tramite le funzioni presenti nel file DBScraper.java

- student(X) - X è l'username degli utenti iscritti alla piattaforma
- course(X) - X è il nome dei corsi presenti nella piattaforma
- courseId(X,Y) - Identifica con Y l'id dei corsi X
- learningObject(X) - X è il nome dei Learning Object presenti
- learningObjectTaglist(X,Y) - Associa al Learning Object X la lista Y dei tag che gli sono assegnati
- learningObjectId(X,Y) - Associa al Learning Object X il suo id Y
- hasLearningObject(X,Y) - Specifica che il corso X contiene il Learning Object Y
- hasRead(X,Y) - Specifica che l'utente X ha confermato di aver visualizzato il Learning Object Y

6.2.2 Predicati

Documentazione sull'utilizzo e sul significato dei predicati presenti nel file predicati.pl

- rootPath(X,Y) = restituisce in Y il genitore di X, fino ad arrivare alla radice.

```
5 ?- rootPath(sql,X).
X = sql ;
X = database ;
X = informatica ;
```

- fullRootPath(X,L) = restituisce in L una lista con tutti i nodi genitore.

```
6 ?- fullRootPath(sql,X).
X = [sql, database, informatica].
```

- first([H|_],H) = restituisce H il primo elemento di una lista.

```
7 ?- first([sql,database,informatica],X).
X = sql.
```

- scorriLista([H|T],A) = restituisce ogni elemento di una lista

```
8 ?- scorriLista([sql,database,informatica],X).
X = sql ;
X = database ;
X = informatica ;
```

- path(X,Y,P) = restituisce P il percorso tra i nodi X e Y, se esiste.

```
14 ?- path(sql,nosql,P).
P = [sql, database, nosql].
```

- distanza(X,Y,D) = restisuisce D la distanza tra i nodi X e Y, se esiste.

```
15 ?- distanza(sql,nosql,D).
D = 2.
```

- esisteDistanza(X,Y,D) = restituisce true se esiste una distanza tra X e Y, altrimenti restituisce -1.

```
16 ?- esisteDistanza(sql,nosql,D).
D = 2.

17 ?- esisteDistanza(sql,medicina,D).
D = -1.
```

- distanzaTaglistTag([H|T],E,Q) = ciclo sulla lista in input e trovo (Q) la distanza dall'elemento in testa con E.

```
18 ?- distanzaTaglistTag([informatica,medicina],sql,D).
D = 2 ;
D = -1 ;
```

- distanzaTaglistTagAsList(L,T,D) = restituisce in forma lista tutte le distanze dagli elementi di L con il tag T.

```
19 ?- distanzaTaglistTagAsList([informatica,medicina],sql,D).
D = [2, -1].
```

- distanzaTaglistTaglist(L,[H|T],Q) = calcola tutte le distanze tra gli elementi delle due liste.

```
22 ?- distanzaTaglistTaglist([informatica,medicina],[sql,nosql],D).
D = 2 ;
D = -1 ;
D = 2 ;
D = -1 ;
```

- distanzaTaglistTaglistAsList(L,T,D) = restituisce in forma di lista tutte le distanze tra gli elementi delle due liste.

```
23 ?- distanzaTaglistTaglistAsList([informatica,medicina],[sql,nosql],D).
D = [2, -1, 2, -1].
```

- countExNex([H|T],E,N) = restituisce in E il numero di collegamenti esistenti e in N il numero di quelli non esistenti.

```
26 ?- countExNex([2,-1,2,2,-1],E,N).
E = 3,
N = 2.
```

- ratioEsistenti(L,R) = ritorna il rapporto tra il numero di collegamenti esistenti e quelli totali.

```
27 ?- ratioEsistenti([2,-1,2,2,-1],R).
R = 0.6.
```

- ratioNonEsistenti(L,R) = ritorna il rapporto tra il numero di collegamenti non esistenti e quelli totali.

```
28 ?- ratioNonEsistenti([2,-1,2,2,-1],R).
R = 0.4.
```

- sumDistanze([H|T],S,C) = restituisce la somma della distanze e il conto delle distanze esistenti (>=0).

```
30 ?- sumDistanze([2,-1,2,2,-1],S,C).
S = 6,
C = 3.
```

- countDistanzaNEX([H|T],C) = restituisce il conto dei collegamenti non esistenti (distanza <0).

```
31 ?- countDistanzaNEX([2,-1,2,2,-1],S).
S = 2.
```

- scoreX(L,S) = rapporto tra somma delle distanze e numero di collegamenti esistenti.

```
32 ?- scoreX([2,-1,2,2,-1],S).
S = 2.
```

- scoreFunction(X,Y) = calcola la funzione score Y = 1/(3**X).

```
34 ?- scoreFunction(2,Y).
Y = 0.1111111111111111.
```

- score(L,S) = calcola lo score sulla lista delle distanze tra due liste di tag.

```
32 ?- scoreX([2,-1,2,2,-1],S).
S = 2.
```

- backchain([H|T],P1) = restituisce il fullRootPath di ogni tag presente nella lista senza il tag in questione.

```
39 ?- backchain([sql,nosql,database,medicina],X).
X = [database, informatica] ;
X = [database, informatica] ;
X = [informatica] ;
X = [] ;
```

- leafs(L,S) = data una lista di tag restituisce un'altra lista togliendo tag genitori di altri tag presenti nella lista.

```
40 ?- leafs([sql,nosql,database,medicina],X).
X = [sql, nosql, medicina].
```

- distPadding([_|T],L) = restituisce un array di zero di dimensione scelta.

```
5 ?- distPadding([a,b,c],X).
X = [0, 0, 0].
```

- scoreTaglistTaglist(X,Y,D) = calcola lo score tra due liste di tag.

```
6 ?- scoreTaglistTaglist([sql,medicina,database],[nosql,matematica,java],X).
X = 0.006796422928602739.
```

- scoreObjectObject(X,Y,D) = calcola lo score tra due oggetti.

```
9 ?- scoreObjectObject('Lezione 1 - Database','Lezione 2 - Database',S).
S = 0.1111111111111111.
```

- scoreObjectObjectNoZero(X,Y,D) = calcola lo score tra due oggetti ma ritorna false se è 0.

- orderedScoreObjectObject(X,O,S) = restituisce ordinati per score tutti gli oggetti simili all'oggetto dato input.

```
11 ?- orderedScoreObjectObject('Lezione 1 - Database',O,S).
O = 'Lezione 3 - Database',
S = 0.1111111111111111 ;
O = 'Lezione 2 - Database',
S = 0.1111111111111111 ;
O = 'Lezione 3 - Programmazion',
S = 0.012345679012345678 ;
O = 'Lezione 3 - IA',
S = 0.012345679012345678 ;
O = 'Lezione 2 - Programmazion',
S = 0.012345679012345678 ;
O = 'Lezione 2 - IA',
S = 0.012345679012345678 ;
O = 'Lezione 1 - Programmazion',
S = 0.012345679012345678 ;
O = 'Lezione 1 - IA',
S = 0.012345679012345678 ;
```

- userHasReadTag(U,T) = restituisce i tag di un oggetto visto dall'utente.

```
15 ?- userHasReadTag('test2',X).
X = [informatica, database, nosql] ;
X = [informatica, programmazione, java] ;
X = [informatica, programmazione, python] ;
X = [medicina, anatomia, tessuti] ;
X = [medicina, anatomia, articolazioni] ;
X = [medicina, anatomia, sangue].
```

- userTaglist(U,S) = costruisce una lista senza duplicati di tutti i tag degli oggetti visti dall'utente.

```
21 ?- userTaglist('test',X).
X = [informatica, database, sql, medicina, anatomia, tessuti, matematica, analisi, retta].
```

- scoreUserObject(U,Y,D) = calcola lo score tra la lista dei tag degli oggetti visti dall'utente e tutti gli oggetti non visti.

```
29 ?- scoreUserObject('test','Lezione 1 - Patologia',S).
S = 0.0022654743095342463.
```

- scoreUserObjectNoZero(U,Y,D) = filtra gli score uguali a 0.

- orderedScoreUserObject(X,O,S) = restituisce ordinati per score tutti gli oggetti da consigliare all'utente.

```
32 ?- orderedScoreUserObject('test2',X,S).
X = 'Lezione 3 - Programmazion',
S = 0.015645374777024282 ;
X = 'Lezione 4 - Patologia',
S = 0.01084789699309083 ;
X = 'Lezione 3 - Database',
S = 0.007521511683153875 ;
X = 'Lezione 1 - Database',
S = 0.007521511683153875 ;
X = 'Lezione 3 - Patologia',
S = 0.00361596566436361 ;
X = 'Lezione 3 - IA',
S = 0.00361596566436361 ;
X = 'Lezione 3 - Farmacologia',
S = 0.00361596566436361 ;
X = 'Lezione 2 - Patologia',
S = 0.00361596566436361 ;
X = 'Lezione 2 - IA',
S = 0.00361596566436361 ;
X = 'Lezione 2 - Farmacologia',
S = 0.00361596566436361 ;
X = 'Lezione 1 - Patologia',
S = 0.00361596566436361 ;
X = 'Lezione 1 - IA',
S = 0.00361596566436361 ;
X = 'Lezione 1 - Farmacologia',
S = 0.00361596566436361 ;
```

- scoreUserObjectRepeat(U,Y,D) = calcola lo score tra la lista dei tag degli oggetti visti dall'utente e tutti gli oggetti già visti.

```
34 ?- scoreUserObjectRepeat('test2',X,S).
X = 'Lezione 2 - Database',
S = 0.015645374777024282 ;
X = 'Lezione 1 - Programmazion',
S = 0.03254369097927249 ;
X = 'Lezione 2 - Programmazion',
S = 0.03254369097927249 ;
X = 'Lezione 1 - Anatomia',
S = 0.0676936051483849 ;
X = 'Lezione 2 - Anatomia',
S = 0.0676936051483849 ;
X = 'Lezione 3 - Anatomia',
S = 0.0676936051483849 ;
```

- scoreUserObjectRepeatNoZero(U,Y,D) = filtra gli score uguali a 0.
- orderedScoreUserObjectRepeat(X,O,S) = restituisce ordinati per score tutti gli oggetti da riconsigliare all'utente.

```
35 ?- orderedScoreUserObjectRepeat('test2',X,S).
X = 'Lezione 3 - Anatomia',
S = 0.0676936051483849 ;
X = 'Lezione 2 - Anatomia',
S = 0.0676936051483849 ;
X = 'Lezione 1 - Anatomia',
S = 0.0676936051483849 ;
X = 'Lezione 2 - Programmazion',
S = 0.03254369097927249 ;
X = 'Lezione 1 - Programmazion',
S = 0.03254369097927249 ;
X = 'Lezione 2 - Database',
S = 0.015645374777024282 ;
```

6.2.3 Relazioni

Le relazioni hasPart(X,Y) definite in questo file seguono gli schemi mostrati in 5.3.2.

```
hasPart(informatica,database).
hasPart(database,sql).
hasPart(database,nosql).
hasPart(database,grafo).
hasPart(informatica,programmazione).
hasPart(programmazione,java).
hasPart(programmazione,python).
hasPart(programmazione,c).
hasPart(informatica,ai).
hasPart(ai,machine_learning).
hasPart(ai,reti_neurali).
hasPart(ai,sistemi_esperti).
```

7 Installazione Guidata

7.1 Codice del progetto

È possibile trovare il codice del progetto all'indirizzo:

https://github.com/DamnMiri/OpenOLAT

Il codice originale del software senza le modifica apportate in questo progetto si può trovare qua:

https://github.com/OpenOLAT/OpenOLAT

7.1.1 Versione originale OpenOLAT

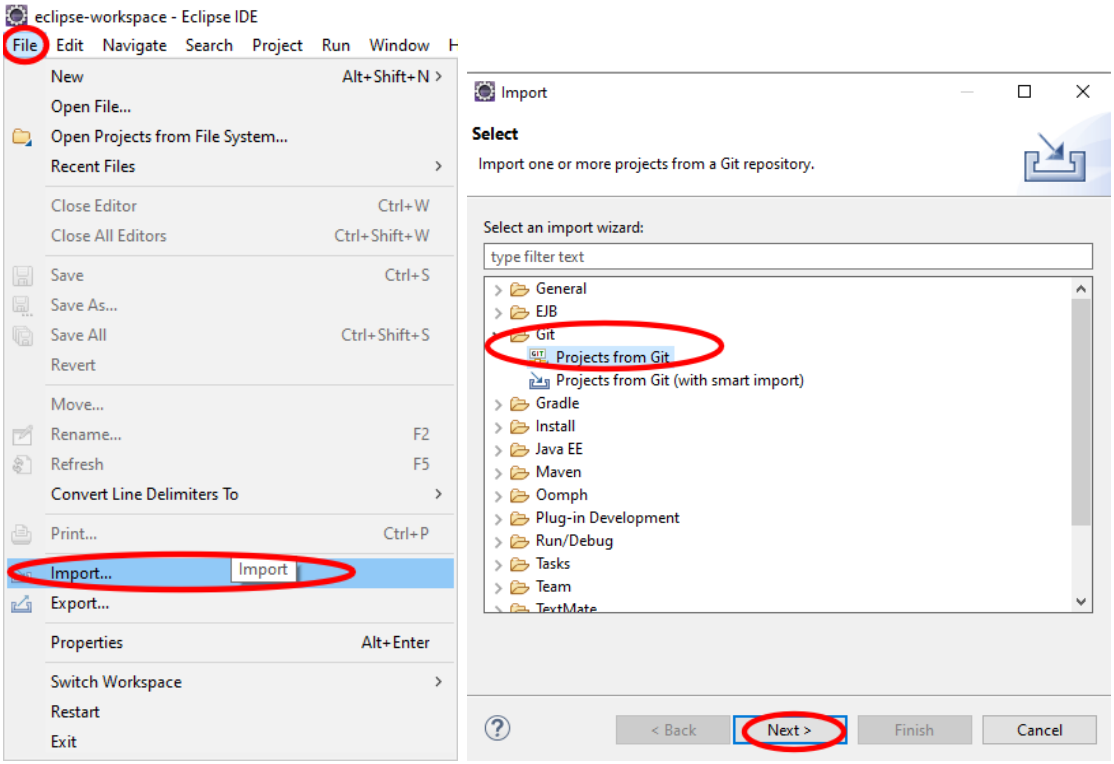
La versione 15.4 di OpenOLAT è quella utilizzata come base di questo progetto.

7.1.2 Software e Tools Utilizzati

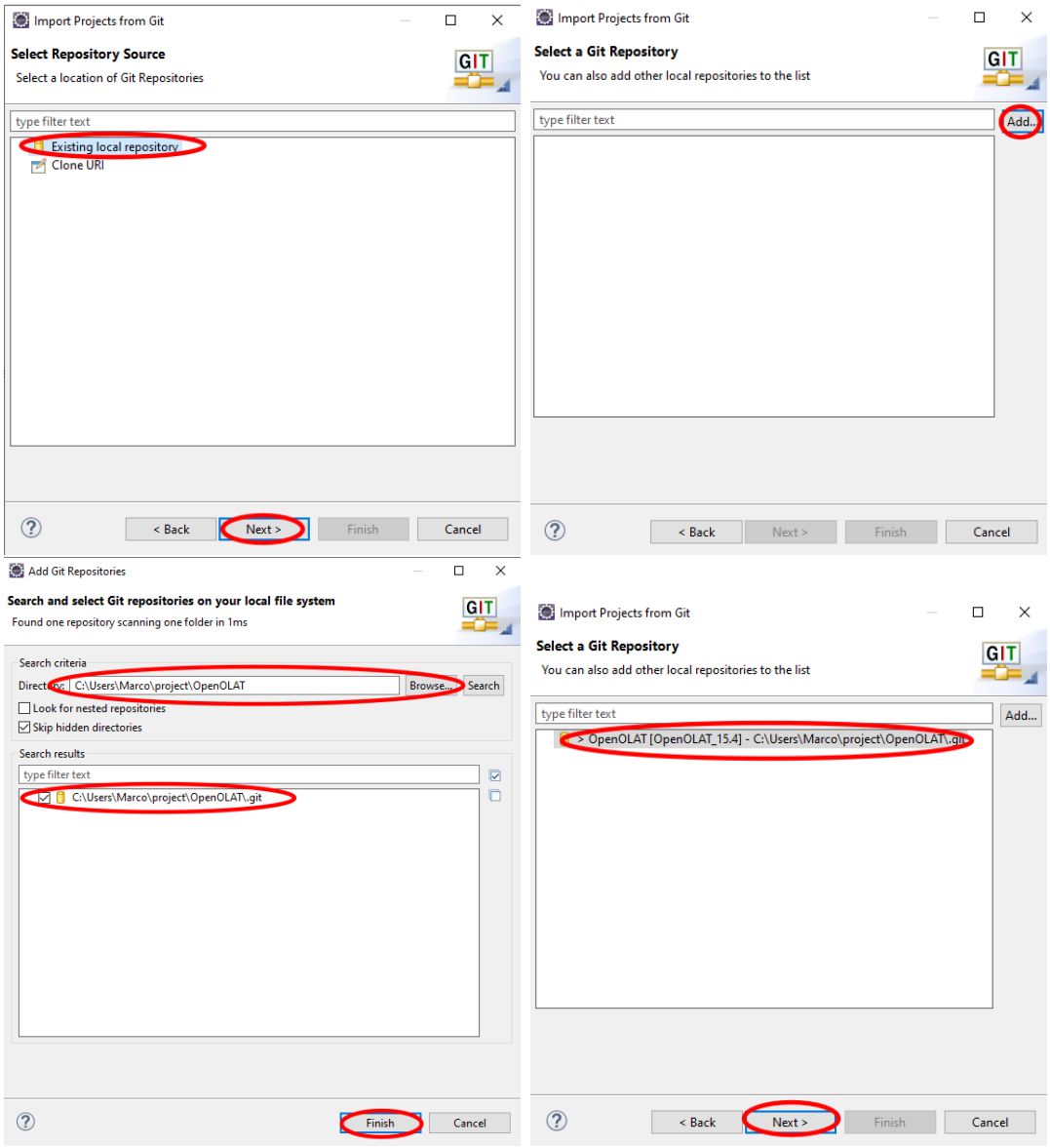
- Maven 3.6.3
- Tomcat 9
- Java 13
- MariaDB 10.4.13
- SWI-Prolog 8.2.3
- Libreria JPL.jar (di default contenuta in SWI-Prolog)
- Eclipse 2020-12 (4.18.0)

Il software è stato anche testato con successo su database Postgresql 13.2

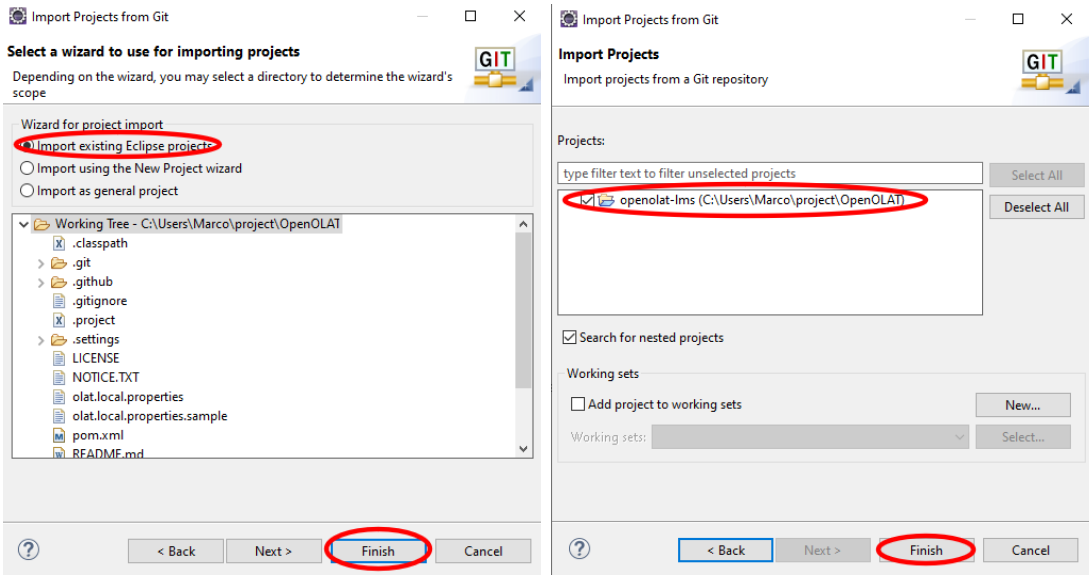
7.1.3 Procedimento



- Importare in Eclipse il progetto contenuto nella cartella OpenOLAT
- Selezionare come metodo di importazione "Projects from Git"



- Selezionare "Existing local repository"
- Cliccare "Add" e indicare la cartella della repository
- Selezionare la repository di OpenOLAT
- Cliccare "Next"



- Selezionare "Importo existing Eclipse project"
- Cliccare "Finish"
- Importare la libreria jpl.jar presente nella cartella prolog nella repository locale di Maven

  ```
  mvn install:install-file -Dfile=<percorso libreria jpl.jar>
  -DgroupId=org.jpl7 -DartifactId=jpl -Dversion=1.0 -Dpackaging=jar
  -DgeneratePom=true
  ```

- Eseguire i seguenti comandi Maven all'interno della cartella del progetto OpenOLAT:

  1. `mvn -Declipse.workspace=<percorso workspace eclipse> eclipse:configure-workspace`
  2. `mvn eclipse:clean eclipse:eclipse`

```
C:\Users\Marco\project\OpenOLAT>mvn -Declipse.workspace="C:\Users\Marco\eclipse-workspace" eclipse:configure-workspace
[INFO] Scanning for projects...
```

```
C:\Users\Marco\project\OpenOLAT>mvn eclipse:clean eclipse:eclipse
[INFO] Scanning for projects...
```

- Creare un user e un database per il software (Database supportati Mysql e Postresql)
  MySQL

  ```
  CREATE DATABASE IF NOT EXISTS openolat;
  GRANT ALL PRIVILEGES ON openolat.* TO 'openolat' IDENTIFIED BY 'openolat';
  UPDATE mysql.user SET HOST='localhost' WHERE USER='openolat' AND
  HOST='%';
  FLUSH PRIVILEGES;
  ```

- Caricare lo schema del DB base oppure solo per MySQL già popolato.
  Se si utilizza lo schema base del DB la cartella openData verrà creata al primo avvio del software, invece nel caso del database già popolato sarà necessario copiare la cartella openData presente in questo progetto perchè contiere i file fisici a cui il database fa riferimento.
  MySQL

  ```
  mysql -u openolat -p openolat < src/main/resources/database/
  mysql/setupDatabase.sql
  ```

  Postgres

  ```
  psql -U openolat -d openolat < src/main/resources/database/
  postgresql/setupDatabase.sql
  ```

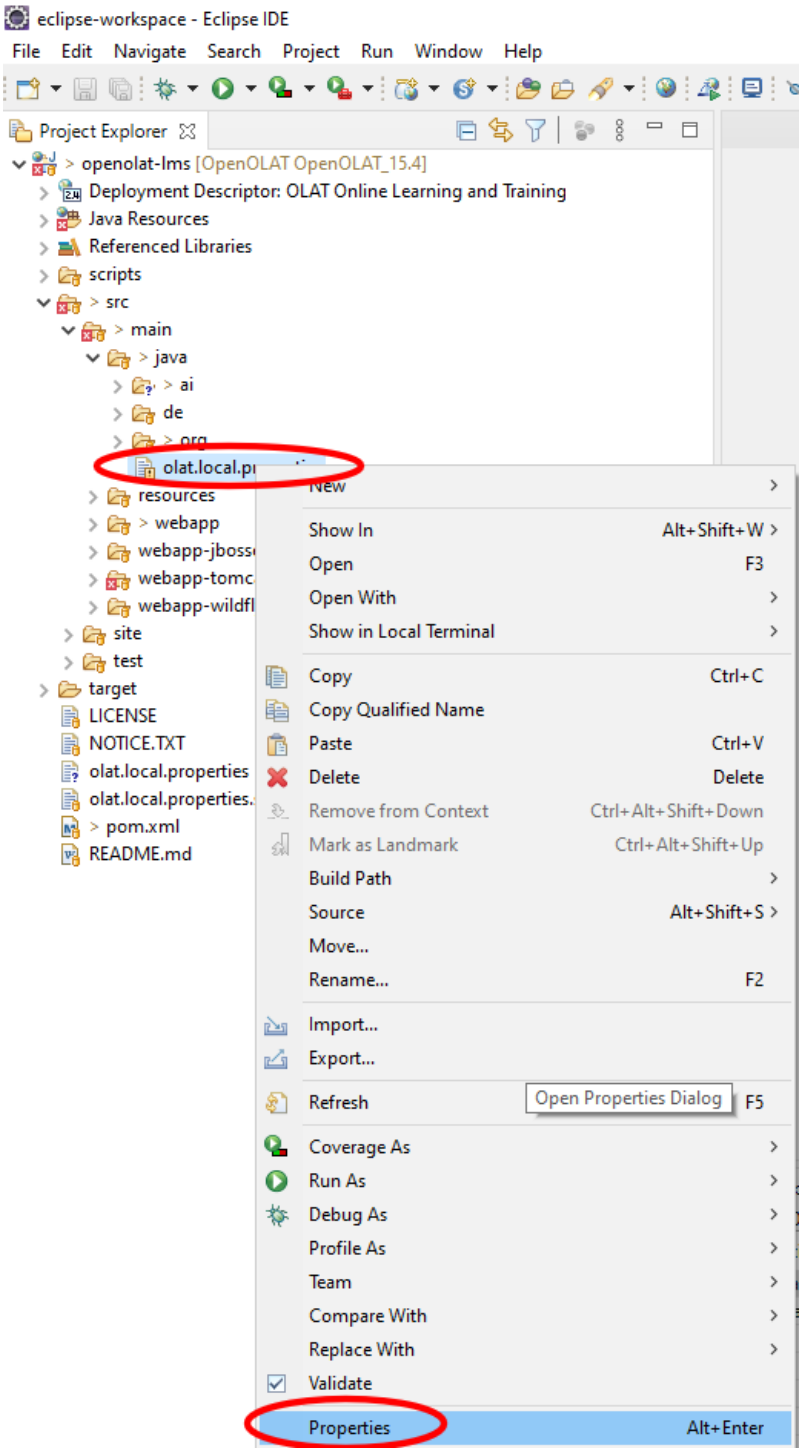
- Lo schema MySQL già popolato si trova nella cartella dbDump (ricordarsi di utilizzare la cartella openData del progetto).
  MySQL

  ```
  mysql -u openolat -p openolat < dbDump/openolatdb.sql
  ```

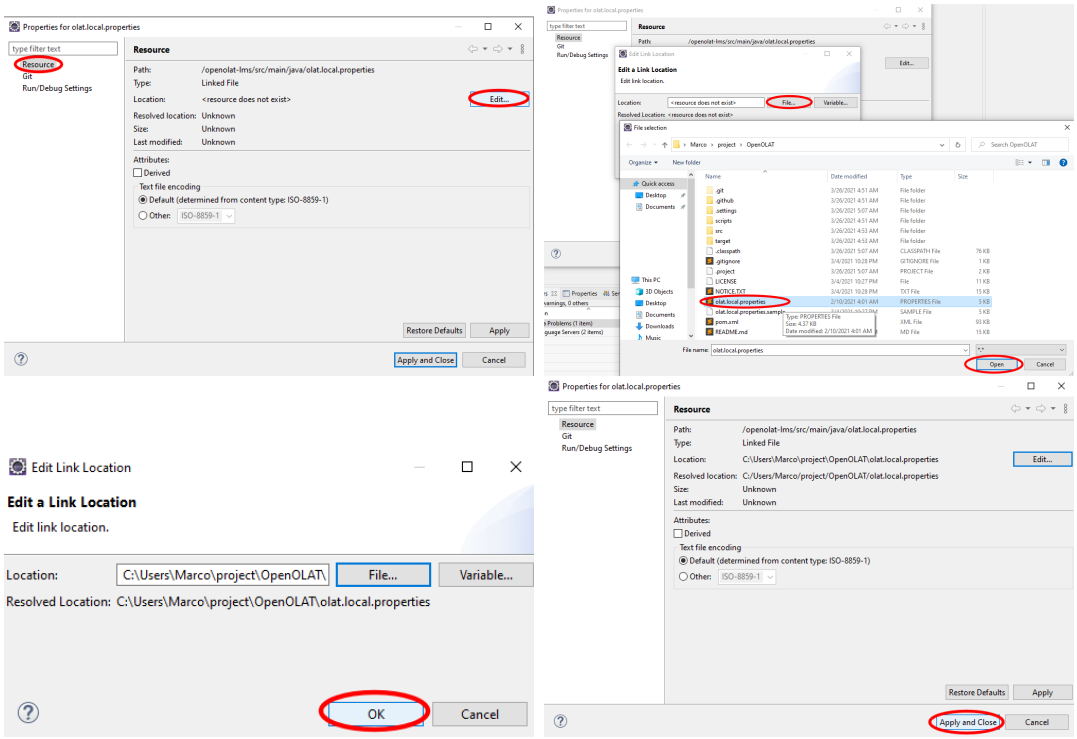
```
# runtime application data directory. Tomcat user
userdata.dir=C:\\Users\\Marco\\Desktop\\openData
# Database settings
db.vendor=mysql
db.host.port=3306

# default ports: mysql 3306 - postgresql 5432
# the name of the application database
db.name=openolat
# the name of the OLAT database user
db.user=openolat
```

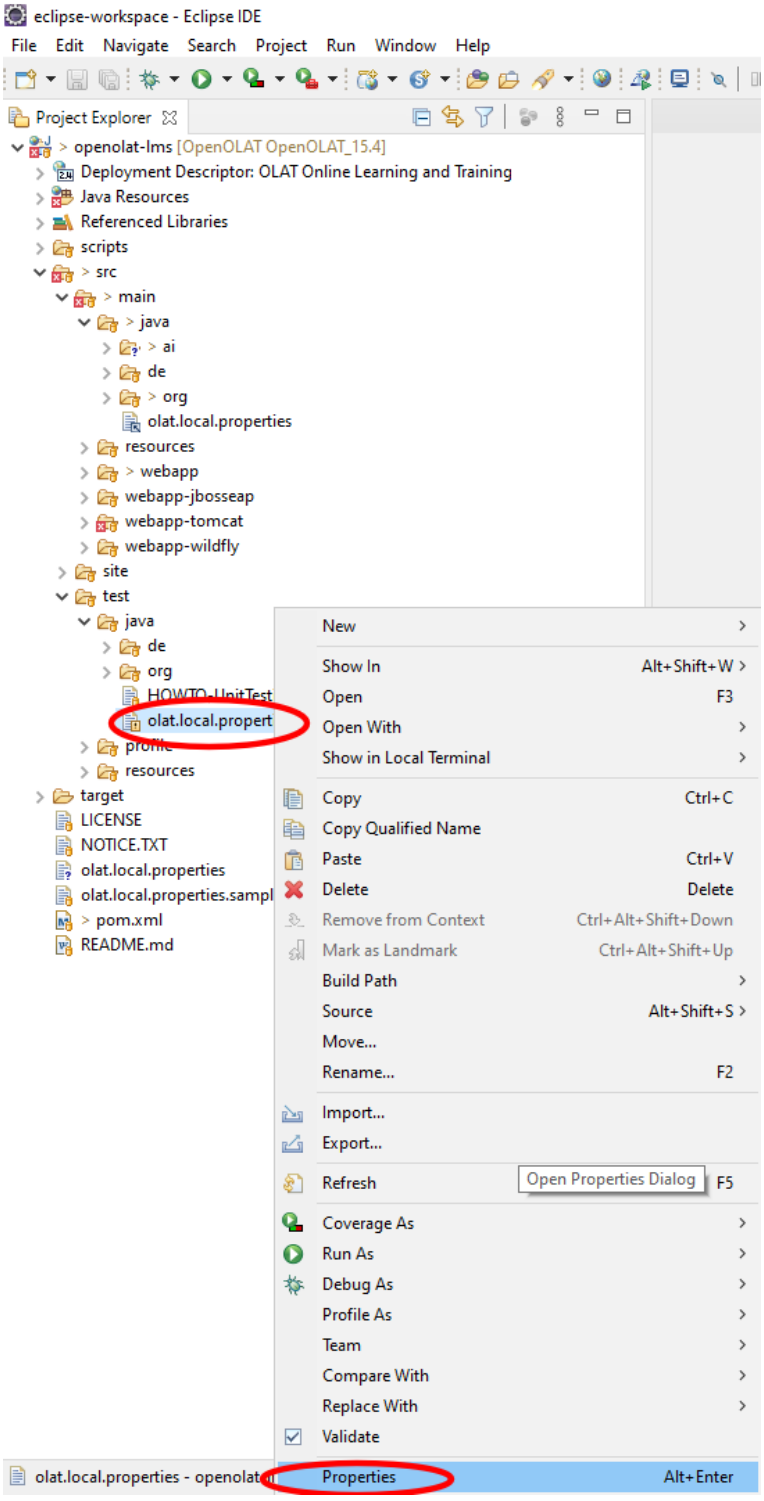
- Personalizzare il file olat.local.properties con i valori relativi alla propria installazione (i valori configurabili sono elencati nel file olat.list.properties)



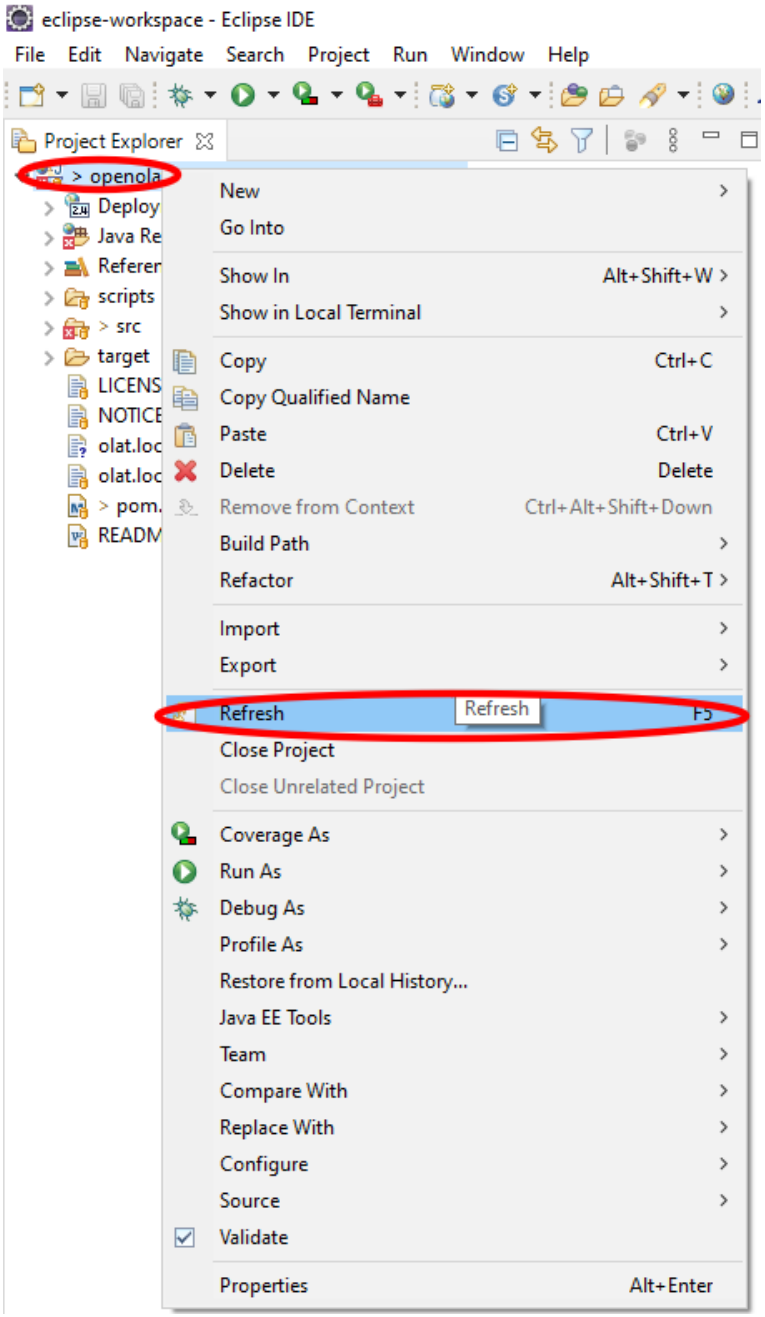
- In "Properties" collegare il file *src/main/java/olat.local.properites* al file omonimo personalizzato in precedenza



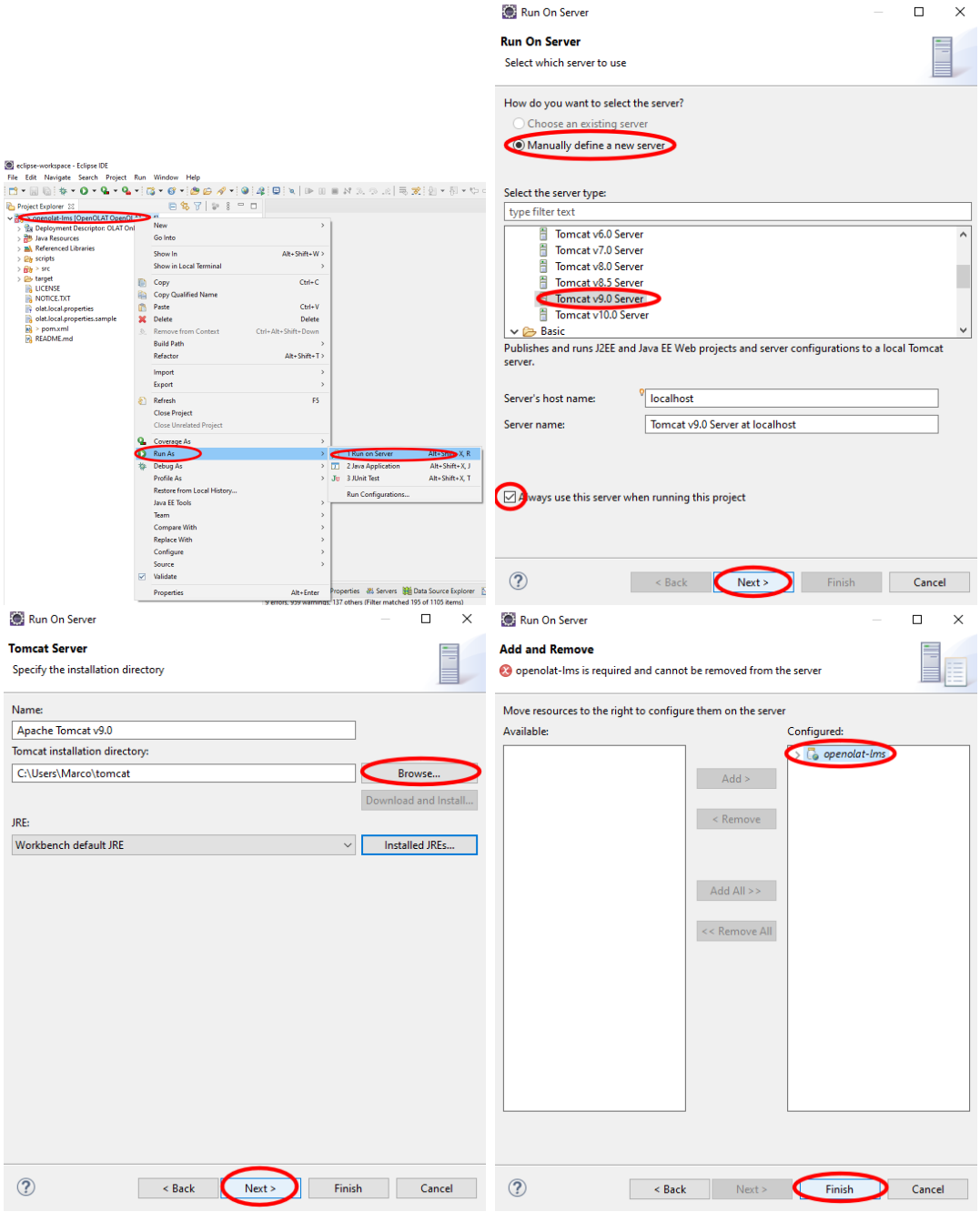
- Selezionare "Edit"
- Cliccare "File" e indicare il file properties modificato in precedenza
- Continuare a applicare i cambiamenti



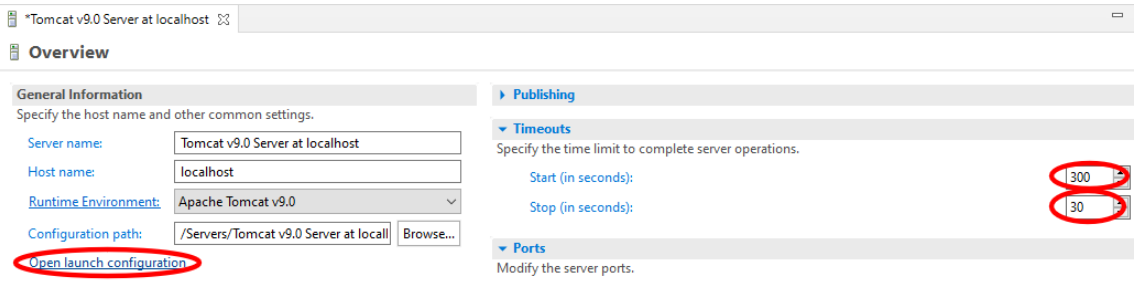
- Ripetere lo stesso procedimento per il collegamento in *src/test/java/olat.local.properites*

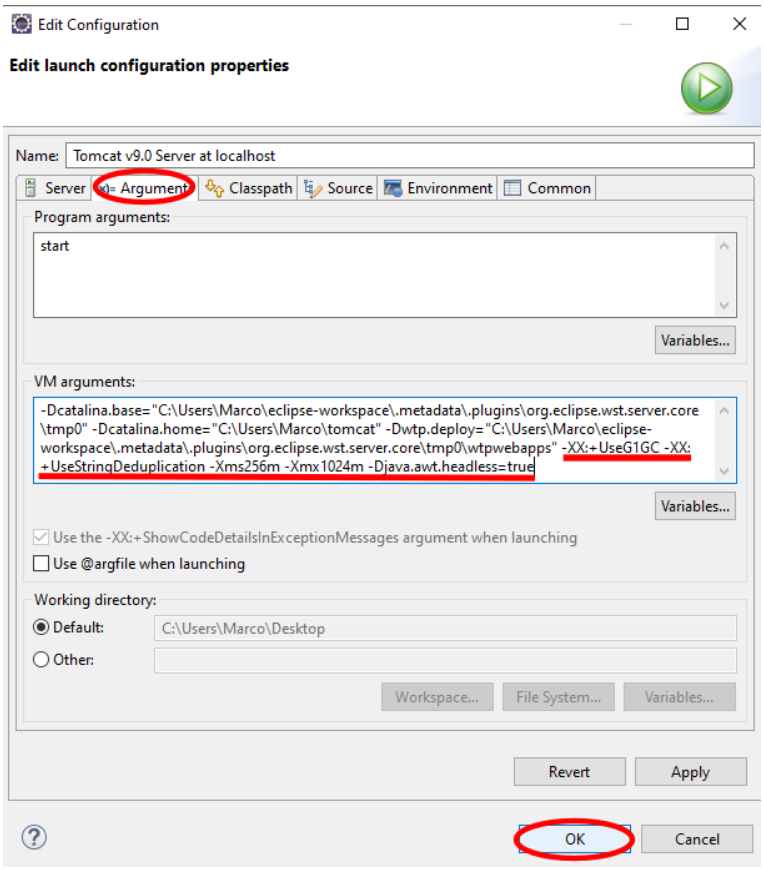


- Fare il refresh del progetto e aspettare che venga completata la fase di Build del progetto



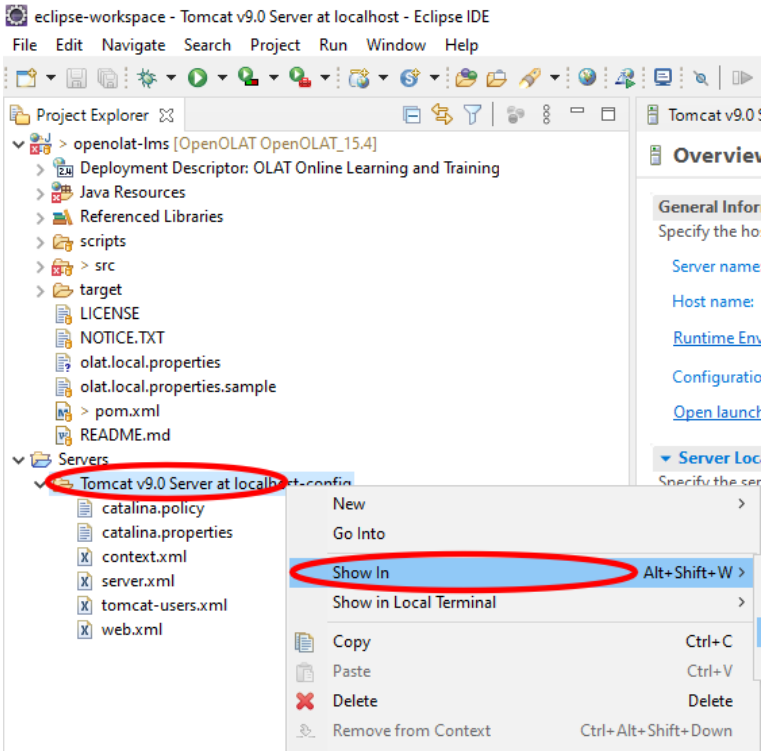
- Fare il "Run" del progetto scegliendo "Run on Server"
- Scegliere Tomcat 9.0 Server e indicare la sua cartella di installazione
- Includere la risorsa "openolat-lms"



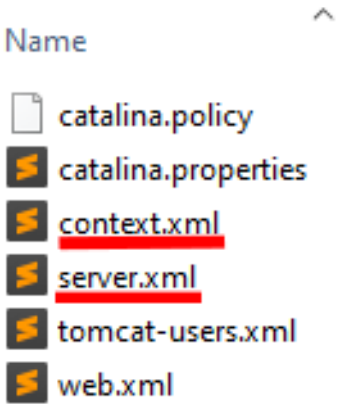


- Aprire i dettagli del server appena creato
- Modificare il tempo di Timeout: 300s / 30 s
- Aprire le configurazioni del server e andando nella scheda "Arguments" aggiungere nella sezione "VM Arguments" i seguenti comandi:

  `-XX:+UseG1GC -XX:+UseStringDeduplication -Xms256m -Xmx1024m -Djava.awt.headless=true`



- Trovare la cartella dove è stato creato il server con la funzione "Show in" -> "System Explorer"



```
--><!-- The contents of this file will be loaded for each web application --><Context>

    <!-- Default set of monitored resources. If one of these changes, the    -->
    <!-- web application will be reloaded.                                   -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <WatchedResource>WEB-INF/tomcat-web.xml</WatchedResource>
    <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat restarts -->
    <!--
    <Manager pathname="" />
    -->
  <Resources cacheMaxSize="51200" />
</Context>
```

- Modificare il file "context.xml" inserendo l'elemento

```
<Resources cacheMaxSize="51200" />
```

```
      <Host appBase="webapps" autoDeploy="true" name="localhost" unpackWARs="true">

        <!-- SingleSignOn valve, share authentication between web applications
             Documentation at: /docs/config/valve.html -->
        <!--
        <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
        -->

        <!-- Access log processes all example.
             Documentation at: /docs/config/valve.html
             Note: The pattern used is equivalent to using pattern="common" -->
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs" pattern="%h %l %u %t &quot;%r&quot; %s %b"
        prefix="localhost_access_log" suffix=".txt"/>

      <Context docBase="openolat-lms" path="/olat" reloadable="false" source="org.eclipse.jst.j2ee.server:openolat-lms"/></Host>
    </Engine>
  </Service>
</Server>
```

```
    <Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" URIEncoding="UTF-8" redirectPort="8443"/>
```

```
<GlobalNamingResources>
  <!-- Editable user database that can also be used by
       UserDatabaseRealm to authenticate users
  -->
  <Resource auth="Container" description="User database that can be updated and saved" factory="
  org.apache.catalina.users.MemoryUserDatabaseFactory" name="UserDatabase" pathname="conf/tomcat-users.xml" type="
  org.apache.catalina.UserDatabase"/>
  <Resource auth="Container" driverClassName="com.mysql.cj.jdbc.Driver" type="javax.sql.DataSource"
        maxIdle="4" maxTotal="16" maxWaitMillis="10000"
        name="jdbc/OpenOLATDS"
        password="openolat" username="openolat"
        url="jdbc:mysql://localhost:3306/openolat?useUnicode=true&amp;characterEncoding=UTF-8&amp;cachePrepStmts=true&amp;
        cacheCallableStmts=true&amp;autoReconnectForPools=true"
        testOnBorrow="true" testOnReturn="false"
        validationQuery="SELECT 1" validationQueryTimeout="-1"/>
</GlobalNamingResources>
```

- Modificare il file "server.xml":
  - settando nell'elemento Context il parametro

    `reloadable="false"`

  - settando nell'elemento Connector il parametro

    `URIEncoding="UTF-8"`

  - aggiungendo nell'elemento GlobalNamingResources uno dei seguenti elementi Resource, in base al database in utilizzo, modificando i valori username password e nome db scelti con quelli scelti in precedenza.

    MySQL

    ```
    <Resource auth="Container" driverClassName="com.mysql.cj.jdbc.Driver"
    type="javax.sql.DataSource" maxIdle="4" maxTotal="16"
    maxWaitMillis="10000" name="jdbc/OpenOLATDS" password="openolat"
    username="openolat" url="jdbc:mysql://localhost:3306/openolat?
    useUnicode=true&amp;characterEncoding=UTF-8&amp;cachePrepStmts=true&amp;
    cacheCallableStmts=true&amp;autoReconnectForPools=true"
    testOnBorrow="true" testOnReturn="false"
    validationQuery="SELECT 1" validationQueryTimeout="-1"/>
    ```

    Postgres

    ```
    <Resource auth="Container" driverClassName="org.postgresql.Driver"
    type="javax.sql.DataSource" maxIdle="4" maxTotal="16"
    maxWaitMillis="-1" name="jdbc/OpenOLATPostgresDS"
    username="postgres" password="postgres"
    url="jdbc:postgresql://localhost:5432/olat" testOnBorrow="true"
    testOnReturn="false" validationQuery="SELECT 1"
    validationQueryTimeout="-1"/>
    ```

Ora il server dovrebbe partire sull'indirizzo:

http://localhost:8080/olat

PROBLEMI COMUNI

- Il database potrebbe dare problemi di Timezone, basta settarne una qualsiasi:

  ```
  SET GLOBAL time_zone = 'Europe/Zurich';
  ```

- OutOfMemoryException: Controllare di aver inserito i valori nel campo Arguments del server Run > Run Configurations > Arguments > VM Arguments:

  ```
  -XX:+UseG1GC -XX:+UseStringDeduplication -Xms256m -Xmx1024m
  -Djava.awt.headless=true
  ```

- Timeout Exception: Aumentare il tempo di Timeout del server Tomcat.

- Se Tomcat si avvia ma non trova OpenOlat il server partirà molto velocemente senza dare errori, ma ovviamente l'applicazione non sarà partita: Tasto destro sul server e cliccare "Publish".

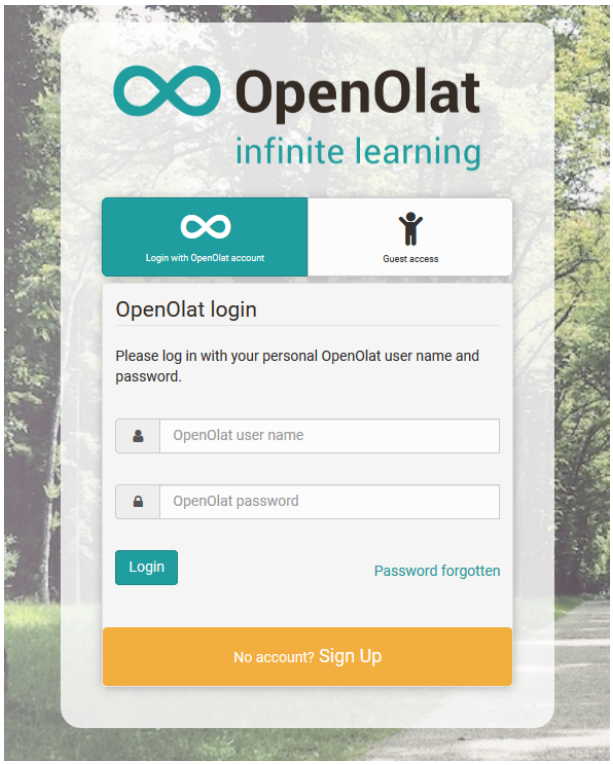
- Se compaiono problemi di tipo "ClassNotFoundException": Fare il "Clean" sul progetto e sul server e rieffettuare la fase di build di tutti il progetto, riavviare tutto il progetto.

- Warning: The predicates below are not defined: Il file prolog fatti.pl non è stato generato correttamente oppure il database è ancora vuoto (non ci sono corsi e interazioni degli utenti con i corsi).

- PrologException: existence_error: Copiare i file prolog .pl dalla cartella prolog alla cartella openData creata sul server dopo il primo avvio oppure il database è ancora vuoto (non ci sono corsi e interazioni degli utenti con i corsi).

8 Manuale Utente

8.1 Account Utente



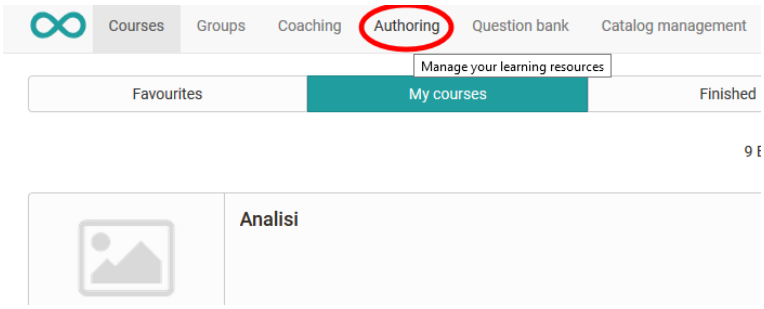
Il software è fornito di alcuni account utente predefiniti (è ovviamente possibile crearne altri) :

- **administrator** - account admin per modificare i parametri di configurazione del software
- **author** - account utente per autore, permette la creazione di nuovi corsi
- **test** - account utente per studenti, permette di iscriversi ai corsi
- **test2** - account utente per studenti, permette di iscriversi ai corsi
- **test3** - account utente per studenti, permette di iscriversi ai corsi

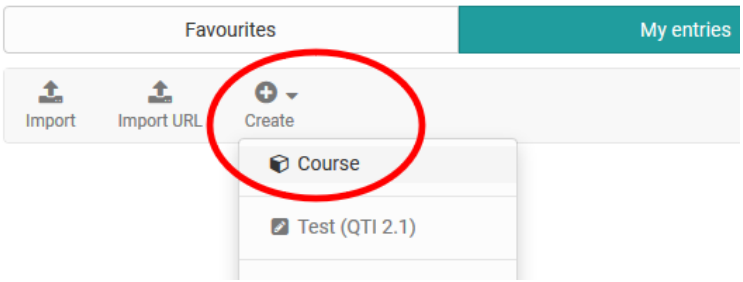
La **password** per l'account administrator è **openolat**, mentre per tutti gli altri account predefiniti è **test**.

8.2 Creazione di un corso

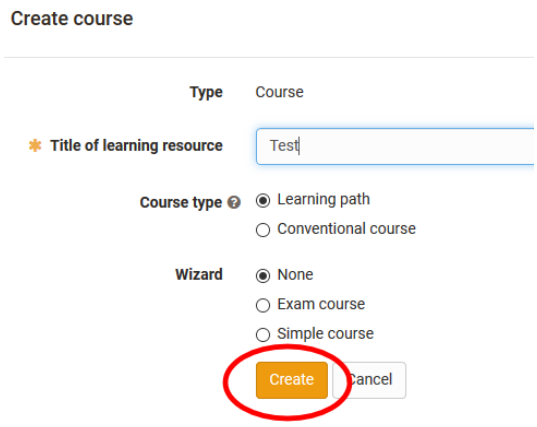
Per creare un nuovo corso è necessario entrare con un account per autori, scegliere la voce **Authoring**



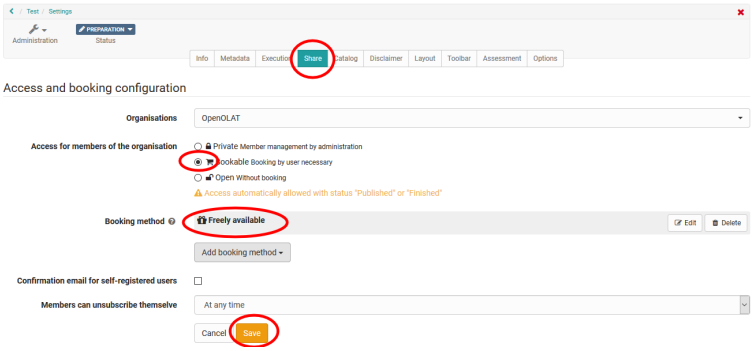
Cliccare prima **Create** e poi **Course**



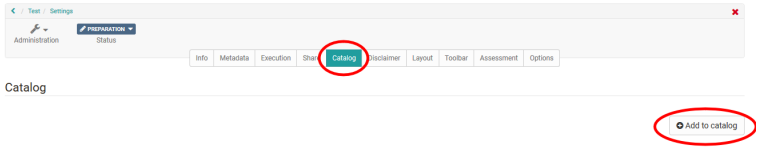
Dare un nome al corso e cliccare **Create**



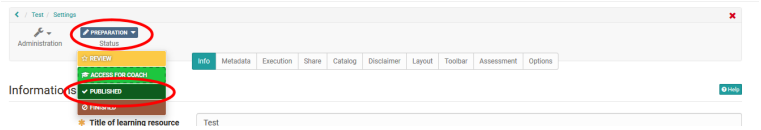
Configurare la sezione **Share** del corso scegliendo **Bookable** e come Booking method selezionare **Freely available** e salvare



Configurare la sezione **Catalog** aggiungendo il corso ad una sezione del catalogo

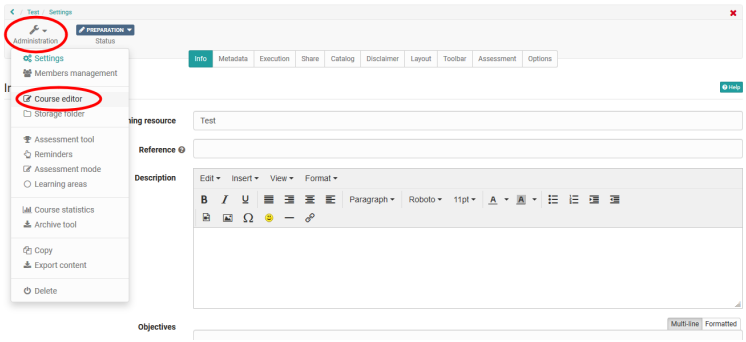


Pubblicare il corso aprendo il menù a tendina **Status** e scegliere l'opzione in verde scuro **PUBLISHED**

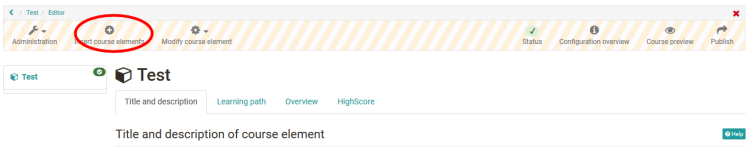


8.3 Creazione di una lezione

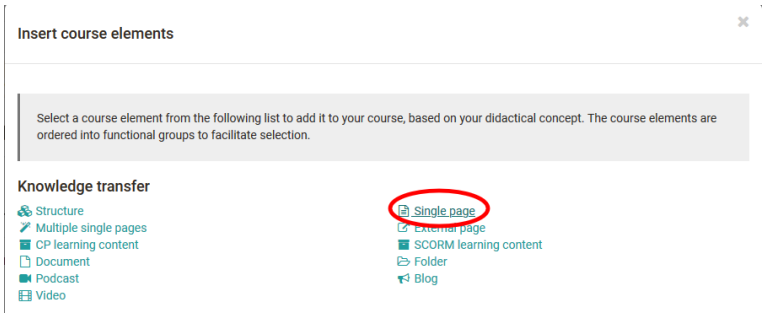
Entrare nella pagina di un corso con l'account dell'autore che l'ha creato, aprire il menù a tendinda **Administration** e scegliere **Course editor**



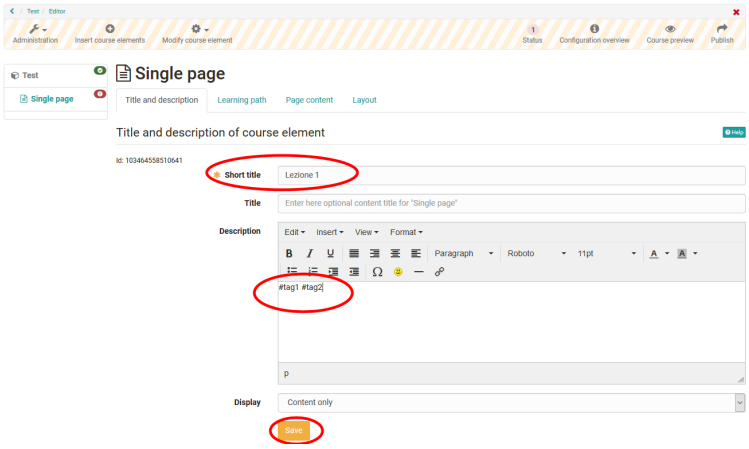
Aggiungere una nuova lezione cliccando **Insert course elements**



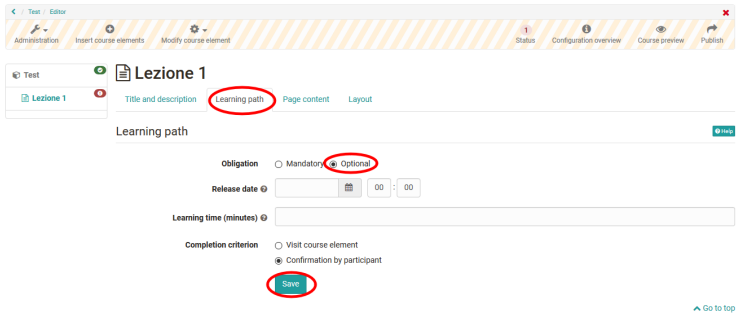
Selezionare il tipo di lezione che si vuole aggiungere, generalmente **Single Page**



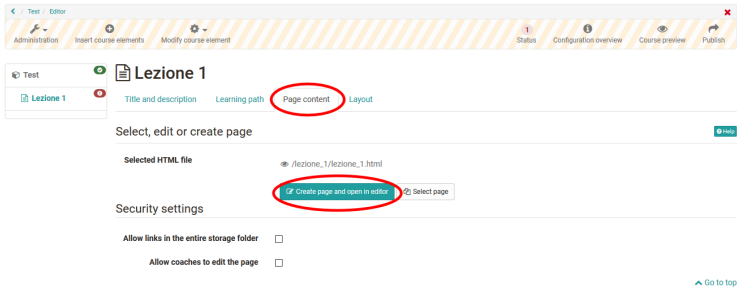
Dare un titolo alla lezione, inserire nella descrizione i tag che la caratterizzano e salvare



Configurare la sezione **Learning path** scegliendo **Optional** e salvare



Configurare la sezione **Page content** creando la pagina della lezione cliccando su **Create page and open in editor** e salvare successivamente



Infine pubblicare la lezione creata con il tasto **Publish** e cliccare **Next** fino a quando non esce il tasto **Finish**

8.4 Iscriversi ad un corso

Per iscriversi ad un corso è necessario entrare con un account per studenti andare nel catalogo dei corsi e cliccare sul tasto verde **book** > relativo al corso che ci interessa