



UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Documentazione Sistema Di Gestione Del Ciclo Di Vita Di Una Pagina Wiki

Giuseppe Pollio, Mario Lombardo

Anno accademico 2023-2024

Indice

1	Dominio del problema	2
1.1	Analisi dei requisiti	2
1.2	Diagramma del Dominio del Problema	4
1.3	Dizionari	5
1.3.1	Dizionario delle Entità	5
1.3.2	Dizionario delle Associazioni	5
2	Impementazione e Dominio della Soluzione	6
2.1	Architettura BCED	6
2.1.1	Boundary	6
2.1.2	Control	6
2.1.3	Entity	6
2.1.4	Database	6
2.2	Controller	6
2.3	Model	7
2.4	Database e DAO	7
2.5	Interfaccia Utente	8
2.6	Diagramma della Soluzione	9
3	Overview dell'Applicazione	10
3.0.1	LoginPage	10
3.0.2	RegisterPage	11
3.0.3	PageView	13
3.0.4	PageCreate	14
3.0.5	PageEdit	15
3.0.6	PageHistory	16
3.0.7	UserNotifications	17
3.0.8	ReviewUpdates	18
4	Sequence Diagram	19
4.1	Login (senza notifiche)	19
4.2	Modifica Pagina	20

1 Dominio del problema

1.1 Analisi dei requisiti

In questa sezione si analizza la traccia in maniera specifica con lo scopo di definire le funzionalità che la base di dati deve soddisfare. Individueremo le entità e le associazioni del mini-word, producendo la prima versione del modello concettuale che sarà poi rielaborato nelle fasi successive della progettazione.

"Si sviluppi un sistema informativo, composto da una base di dati relazionale e da un applicativo Java dotato di GUI (Swing o JavaFX), per la gestione del ciclo di vita di una pagina di una wiki"

"Una pagina di una wiki ha un titolo e un testo. Ogni pagina è creata da un determinato autore. Il testo è composto di una sequenza di frasi. Il sistema mantiene traccia anche del giorno e ora nel quale la pagina è stata creata. Una frase può contenere anche un collegamento. Ogni collegamento è caratterizzato da una frase da cui scaturisce il collegamento e da un'altra pagina destinazione del collegamento."

"Il testo può essere modificato da un altro utente del sistema, che seleziona una o più delle frasi, scrive la sua versione alternativa (modifica) e prova a proporla"

Dall'introduzione individuiamo il mini-world da rappresentare, ovvero un *"Sistema informativo per la gestione del ciclo di vita di una pagina di una wiki"*, e iniziamo a riconoscere la prima entità: **Page** avente come attributi **title** e **creation_date**. Inoltre una pagina della Wiki deve essere composta da un testo, a sua volta composto da una sequenza di frasi, e questo testo deve poter essere modificabile da un utente, individuiamo così un'entità associata alla pagina: **PageText** contenente come attributi, la riga di testo **text** e un indice per l'ordinamento delle righe di testo **order_num**. Invece di utilizzare un singolo attributo per salvare tutto il contenuto di una pagina l'utilizzo dell'entità **PageText** ottimizza molto le operazioni di modifica del testo poiché lavora su singole righe di testo invece di lavorare su l'intero contenuto. Una frase del testo di una pagina può contenere un collegamento ad un'altra pagina, otteniamo questo comportamento aggiungendo all'entità **PageText** l'attributo **link** (attributo parziale) tenendo traccia del **page_id** della Pagina alla quale ci riferiamo. Andiamo ad utilizzare il formato {**page_id:riga_di_testo**} quando una riga di testo è un collegamento, sarà l'applicativo a mostrare solo la **riga_di_testo** e gestire il collegamento alla pagina.

"La modifica proposta verrà notificata all'autore del testo originale la prossima volta che utilizzerà il sistema. L'autore potrà vedere la sua versione originale e la modifica proposta. Egli potrà accettare la modifica (in quel caso la pagina originale diventerà ora quella con la modifica apportata), rifiutare la modifica (la pagina originale rimarrà invariata). La modifica proposta dall'autore verrà memorizzata nel sistema e diventerà subito parte della versione corrente del testo. Il sistema mantiene memoria delle modifiche proposte e anche delle decisioni dell'autore (accettazione o rifiuto)."

"Nel caso in cui si fossero accumulate più modifiche da rivedere, l'autore dovrà accettarle o rifiutarle tutte nell'ordine in ordine dalla più antica alla più recente"

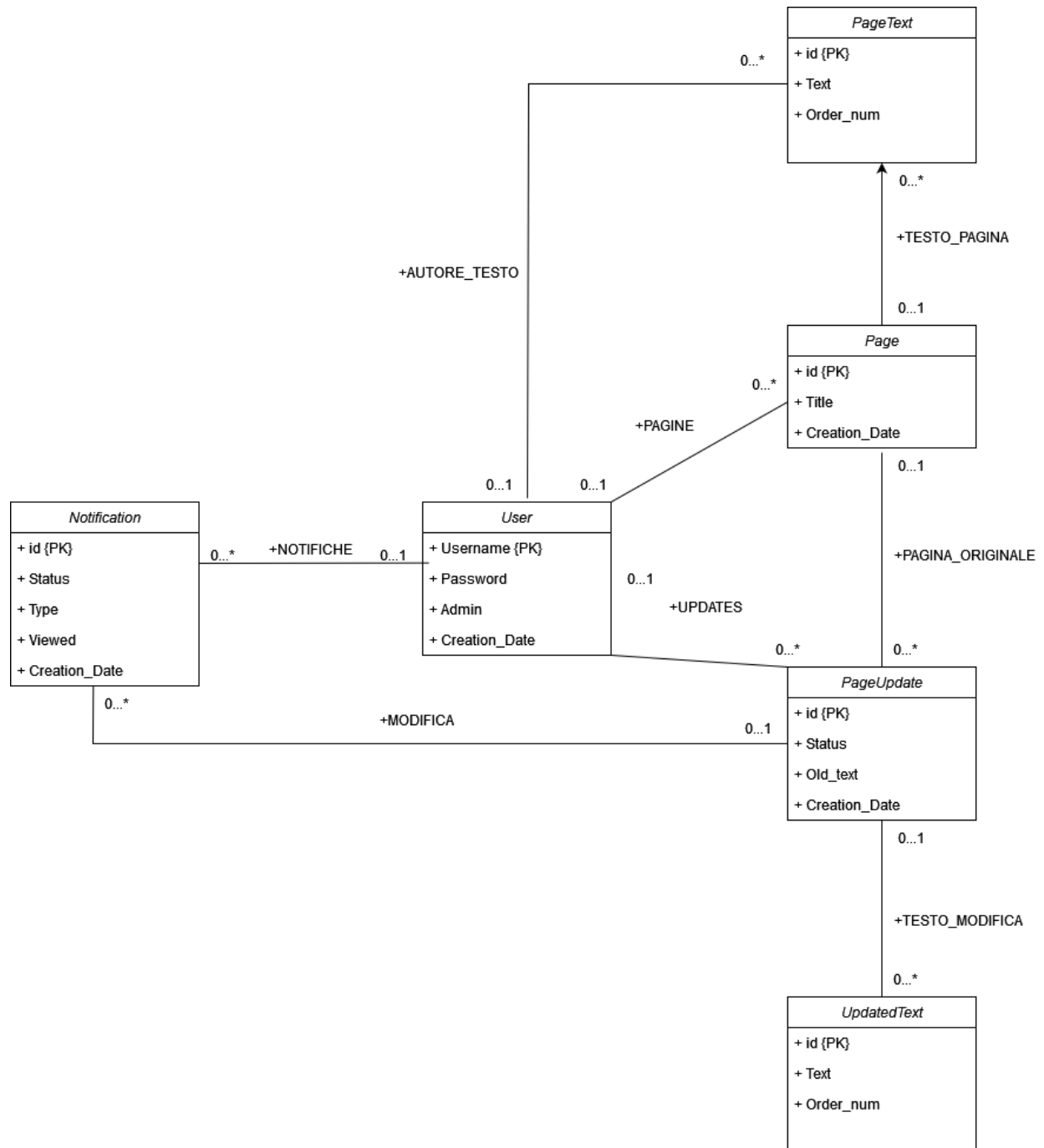
"Gli utenti generici del sistema potranno cercare una pagina e il sistema mostrerà la versione corrente del testo e i collegamenti. Gli autori dovranno prima autenticarsi fornendo la propria login e password. Tutti gli autori potranno vedere tutta la storia di tutti i testi dei quali sono autori e di tutti quelli nei quali hanno proposto una modifica"

Sarà necessario tenere traccia degli utenti della Wiki per poter permettere loro pubblicare nuove pagine e gestire le pagine esistenti (richiedendo modifiche agli autori), da questo andiamo a creare l'entità **User**, alla quale sarà possibile accedere al sistema tramite una **password** e un **username**. Inoltre il sistema tiene conto del fatto che un utente possa essere un Gestore della Wiki tramite l'attributo **admin** (attributo parziale).

All'entità **Page** individuiamo la necessità di possedere un autore inserendo l'attributo **author**. Le modifiche del testo di una pagina (**Page**) saranno salvate tramite le l'entità **PageUpdate** avente attributi: **status**, che indica lo stato della modifica, e un collegamento all'utente **User** autore della modifica (**author**). Le righe di testo da modificare sono individuate tramite l'entità **UpdatedText** che contiene l'attributo **text** che indica la "nuova riga di testo", un collegamento **link** (attributo parziale) e un collegamento all'entità **PageUpdate**. Quando una modifica viene inoltrata all'autore esso invece deve essere avvisato tramite una notifica, questa verrà gestita dall'entità **Notification** avente come attributi **status** (stato della notifica, OPEN, CLOSED), il tipo di notifica **type** (REQUEST_UPDATE, UPDATE_ACCEPTED, UPDATE_REJECTED), **viewed** per indicare se la notifica è stata letta, e ovviamente, contiene le relazioni all'utente destinatario **User** e alla modifica del testo **PageUpdate**. In fine *Tutti gli autori potranno vedere tutta la storia di tutti i testi dei quali sono autori e di tutti quelli nei quali hanno proposto una modifica* sarà possibile per via delle relazioni **User** → **Page** usando l'attributo **author** contenuto in **Page**, e **PageUpdate** → **Page** usando l'attributo **page_id** in **PageUpdate**.

1.2 Diagramma del Dominio del Problema

In seguito alle considerazioni espresse nella sezione precedente, si è prodotto il seguente class diagram del dominio del problema:



1.3 Dizionari

1.3.1 Dizionario delle Entità

Entità	Descrizione	Attributi
User	Account Utente della wiki.	Username (Stringa): nome identificativo dell'account utente. Password (Stringa): password necessaria per accedere all'account utente.
Page	Pagina presente sulla wiki.	Title (Stringa): Titolo della pagina. CreationDate (Data/Timestamp): Data e ora di creazione della pagina.
PageText	Frase di una Pagina (Page).	Text (Stringa): Contenuto della frase. Order_num (Intero): Ordinamento della frase all'interno del testo complessivo.
PageUpdate	Modifica proposta ad pagina da parte di un altro utente.	Status (Intero): Stato della richiesta di modifica.
UpdatedText	Nuovo testo proposto durante la modifica (PageUpdate).	Text (Stringa): Contenuto della riga di testo. Order_num (Intero): Ordinamento della frase all'interno del testo complessivo.
Notification	Notifiche di un Utente (User) .	Status (Intero): Stato della notifica. (OPEN, CLOSED) Viewed (Booleano): Stato di lettura di una notifica. Type (Intero): Tipologia di notifica. (REQUEST_UPDATE, UPDATE_ACCEPTED, UPDATE_REJECTED)
Admin	Specializzazione parziale di un utente, un amministratore può modificare ed eliminare le pagine di altri utenti senza dover prima inviare una notifica di accettazione delle modifiche.	
Link	Specializzazione parziale di una riga di testo che rappresenta se una riga di testo è un collegamento o meno.	

1.3.2 Dizionario delle Associazioni

Associazione	Tipologia	Descrizione
Autore Pagina	uno-a-molti	Associa ad ogni pagina (Page) un utente (User) che ne rappresenta l'autore
Autore Modifica	uno-a-molti	Associa ad ogni modifica (PageUpdate) un utente (User) che ne rappresenta l'autore
Testo Pagina	uno-a-molti	Associa ad ogni pagina (Page) tutto il testo (PageText) appartenente a quella determinata pagina.
Testo Modifica	uno-a-molti	Associa ad ogni modifica (PageUpdate) tutto il testo (UpdatedText) appartenente a quella determinata modifica.
Proprietario Notifica	uno-a-molti	Associa ad ogni notifica (Notification) un utente (User) che ne rappresenta l'autore.
PageUpdate Notifica	uno-a-molti	Associa ad ogni notifica (Notification) una modifica (PageUpdate) che ne aiuta a rappresentare il contenuto.
Page PageUpdate	uno-a-molti	Associa ad ogni Modifica proposta (PageUpdate), La pagina (Page) per cui è stata proposta.

2 Impementazione e Dominio della Soluzione

In questa sezione esploriamo l'architettura dell'applicazione e i vari pattern utilizzati, fornendo occasionali annotazioni implementative. Abbiamo scelto di escludere la trattazione del codice effettivo, il quale è disponibile nella documentazione JavaDoc del codice sorgente e, naturalmente, nel sorgente stesso.

2.1 Architettura BCED

L'applicazione è stata progettata seguendo i principi dell'architettura *Boundary-Control-Entity con estensione Database*, nota come BCED. Il pattern BCED fornisce una struttura ben organizzata per separare le diverse responsabilità all'interno di un software, basandosi su quattro componenti principali:

2.1.1 Boundary

Il componente boundary corrisponde all'interfaccia dell'applicazione che si occupa di far comunicare applicazione e utente attraverso il controller. All'interno del codice della wiki questo componente è implementato attraverso il package **GUI**.

2.1.2 Control

Il componente *control* gestisce il ruolo chiave di coordinare le comunicazioni tra i componenti *Boundary* e *Entity*. Il control è il cuore di un software che basato sull'architettura BCED, avendo un ruolo di controllo sull'intera applicazione. Nella nostra soluzione abbiamo scelto di avere un solo Controller **WikiController** per facilità implementative e di gestione del codice. Questo componente è implementato attraverso il package **Controllers**.

2.1.3 Entity

Il componente Entity costituisce l'insieme delle informazioni da memorizzare durante l'esecuzione che sono di interesse nel dominio. Nel contesto di un'applicazione, essa è rappresentata dal modello di dati interno.

Questo componente è stato definito attraverso il package **Model**.

2.1.4 Database

Nel contesto della nostra applicazione usiamo come database Postgres. Il database funge da posto in cui conservare i dati nel lungo termine. La connessione tramite database è stata implementata tramite il package: **Database**. Le chiamate al database invece vengono gestite tramite l'uso di un altro pattern Architetturale chiamato DAO, cioè **Data access objects**, che viene implementato tramite i package: **DAO** e **DAOImplementations**. Il pattern DAO verrà approfondito nella sezione dedicata di questo documento.

2.2 Controller

Il controller come già detto precedentemente è si occupa di gestire tutta la logica dell'applicazione, nell'applicazione questa componente è stata integrata tramite il package **Controllers** a sua volta contenente una singola classe: **WikiController.java**. Nello specifico nella nostra applicazione le funzioni principali del WikiController sono:

- **Gestione e Controllo del model:** Nel Controller sono contenute tutte le istanze di classi Model necessarie durante l'esecuzione.
- **Comunicazione r/w col Database:** Il Controller si occupa di comunicare col database dando la possibilità di inserire nuovi record, ottenere i Models già salvati e aggiornando i record esistenti.

- **Comunicazione della GUI:** La GUI invia informazioni al Controller in modo da poter poi eseguire azioni sul database e/o model. E viceversa, il Controller invierà le informazioni necessarie per mostrare una determinata GUI.

2.3 Model

Tutte le entità che sono state identificate nel dominio nella sezione **Dominio del problema** sono state tradotte in classi all'interno del package **Models** tuttavia non sono state tradotte esplicitamente le relazioni fra di esse. Nella nostra soluzione abbiamo deciso di optare per una tecnica di ottimizzazione del caricamento dei dati dal database, piuttosto che caricare tutti i Models in una volta sola si è deciso di utilizzare la tecnica del lazy loading, ossia piuttosto che prendere tutti i dati dal database assieme, si andranno a prendere solo i dati che saranno necessari in quel momento, questa tecnica molto utilizzata nei programmi più moderni permette di limitare l'utilizzo di memoria e i tempi di caricamento.

Dato l'utilizzo del lazy loading l'unica istanza di Model presente all'interno del Controller, e dunque persistente per tutta l'esecuzione del programma è:

- **User loggedUser:** Istanza di **User** che rappresenta l'utente attualmente loggato. All'interno dell'istanza sono presenti anche le notifiche.

Il resto dei dati verrà caricato in modo dinamico, ad esempio se apro una determinata pagina della wiki saranno presi dal database in tempo reale solo i dati che servono al funzionamento di quella pagina, tuttavia una volta cambiata pagina i dati precedenti verranno cancellati dalla memoria.

2.4 Database e DAO

L'implementazione della comunicazione al database è dato dal pattern DAO (*Data Access Object*). Questo pattern propone la creazione di un'interfaccia astratta di comunicazione tra l'applicativo e il database contenente l'insieme di metodi per le interazioni necessarie al database. Il software è quindi progettato per interagire tramite la base di dati esclusivamente attraverso questa interfaccia, in questo modo saranno le classi di implementazione dei DAO a gestire completamente le chiamate al database, evitando qualsiasi dipendenza da un'applicazione specifica. Ciò consente l'applicativo di essere indipendente rispetto la base di dati, poichè non ha una conoscenza specifica sul database con cui sta interagendo.

Nella soluzione implementiamo il pattern DAO tramite due package

1. **DAO:** dove sono inserite esclusivamente le interfacce dei DAO utilizzati.
2. **DAOImplementations:** dove sono inserite le implementazioni delle interfacce contenute nel package precedente. Qui sono scritte le vere e proprie interazioni al database.

Il package DAO contiene l'interfaccia al database rappresentata da una serie di interfacce Java. Ciascuna interfaccia DAO definisce la firma delle operazioni eseguibili per l'entità corrispondente.

La connessione al database viene effettuata tramite il package **database**, il database scelto è PostgreSQL.

2.5 Interfaccia Utente

L'interfaccia utente è stata progettata in parte attraverso Swing Designer, contenuto nell'IDE IntelliJ IDEA, questo programma si occupa di creare in automatico un file form con estensione .xml nella quale verrà contenuto il design della pagina, successivamente in fase di runtime verranno settati tutti i binding all'interno di una classe con lo stesso nome del form. Un'altra parte di interfaccia grafica è stata scritta a mano attraverso e racchiusa in classi specifiche che vengono istanziate dinamicamente a seconda del solo caso specifico, in particolare queste classi sono:

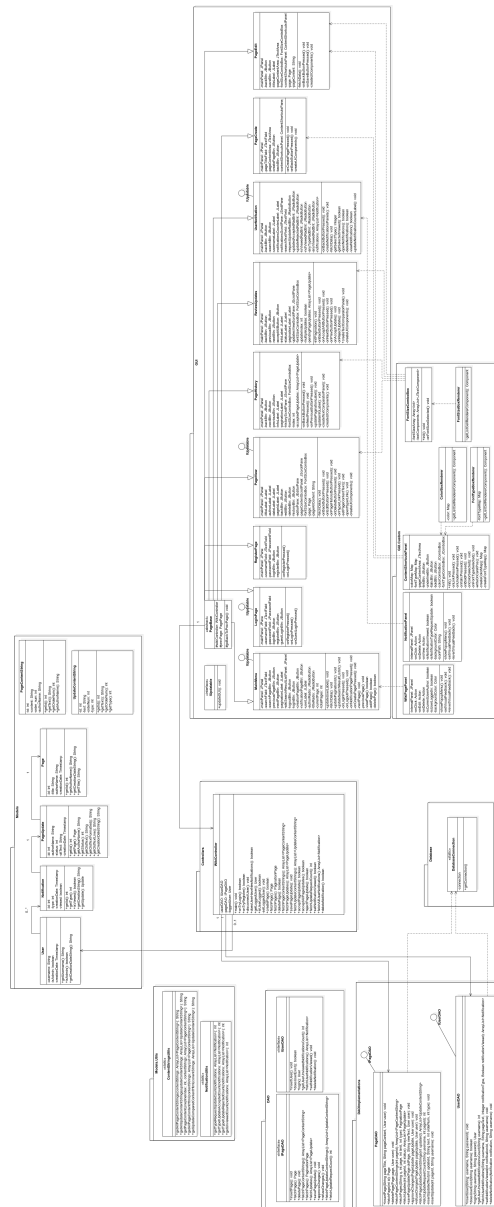
- **FontSizeComboBox**: Una combobox per cambiare a runtime le dimensioni del testo visualizzato.
- **ContentShortcutsPanel**: Un pannello con diverse impostazioni di inserimento del testo. (ad es. bold, italic).
- **NotificationPanel**: Un pannello per visualizzare una notifica e per poterci interagire.
- **WikiPagePanel**: Un pannello per visualizzare una pagina e per poterci interagire.
- **UpdateTextComparatorPanel**: Un pannello per la visualizzazione di due testi, viene impiegato per la comparazione delle modifiche del testo di una pagina.

Queste classi sono chiamate dinamicamente e sono tutte racchiuse sotto il package **GUI.Custom**, ciò è molto utile nella nostra soluzione poiché queste classi hanno il bisogno di essere istanziate più volte nella stessa GUI o in GUI diverse. Inoltre abbiamo utilizzato dei Custom Renderers per rendere più piacevole all'utente finale l'utilizzo di alcune comboBox:

- **ColorBoxRenderer**: Mostra delle preview sui colori selezionabili nella comboBox.
- **FontSizeBoxRenderer**: Mostra le grandezze selezionabili nella comboBox nel formato $\text{b}_i\text{Numeropx}/\text{b}_i$.
- **FontTypeBoxRenderer**: Mostra in grandezze diverse i tipi di testo utilizzabili. Nel nostro caso sono Titolo, sottotitolo e paragrafo.

Per la GUI è stata utilizzata la libreria Java Swing con l'implementazione della libreria **Flatlaf** che ne migliora l'aspetto.

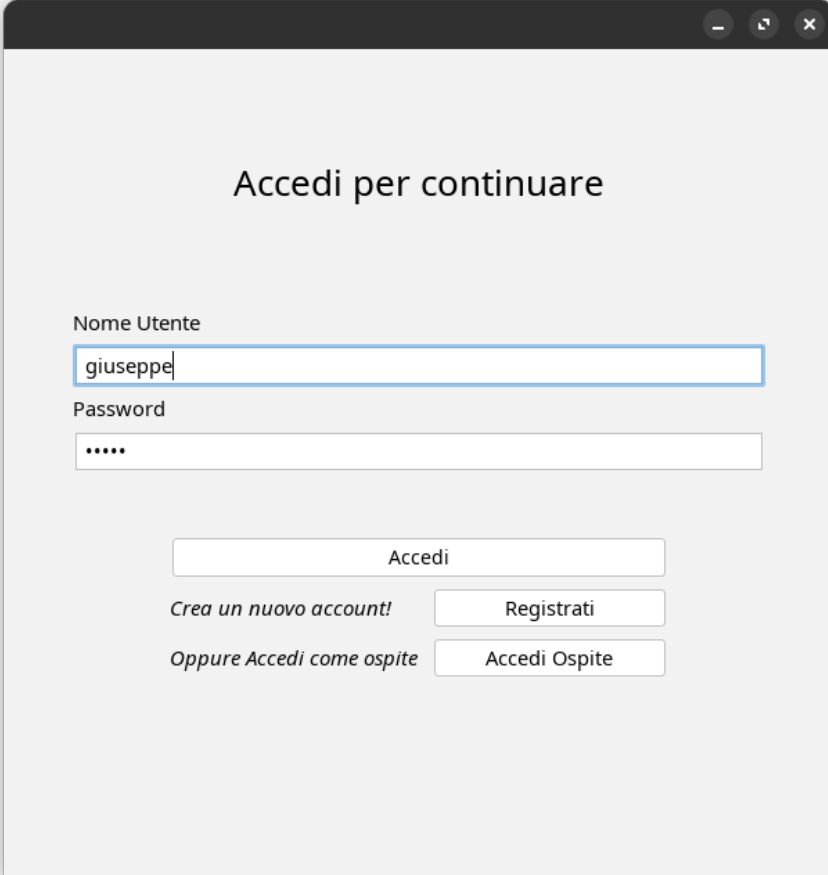
2.6 Diagramma della Soluzione



3 Overview dell'Applicazione

Avendo parlato del funzionamento esterno dell'applicazione ora verrà presentato un "tour" guidato all'interno di essa, pagina per pagina. Specificandone tutte le funzionalità.

3.0.1 LoginPage



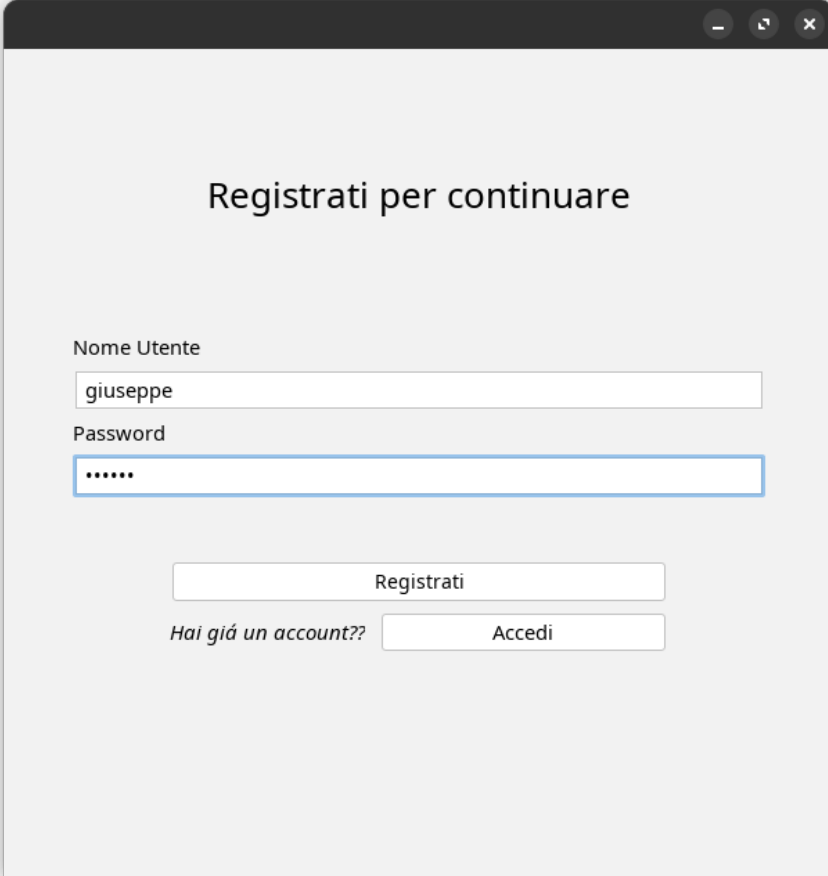
The image shows a web browser window titled "Accedi per continuare". Inside the window, there is a login form. The form has two input fields: "Nome Utente" (Username) and "Password". The "Nome Utente" field contains the text "giuseppe". Below the "Password" field, there are three buttons: "Accedi", "Registrati", and "Accedi Ospite". The "Accedi" button is highlighted with a blue border. The "Registrati" button is disabled. The "Accedi Ospite" button is also disabled. The text "Crea un nuovo account!" is displayed next to the "Registrati" button. The text "Oppure Accedi come ospite" is displayed next to the "Accedi Ospite" button.

Figure 1: L'utente in questo caso ha già inserito i propri dati.

L'applicazione ha come sua view iniziale la **LoginPage**. Qui l'utente può decidere se continuare col proprio account, inserendo i propri dati, registrarsi aprendo la view **RegisterPage** o accedere alla wiki come ospite.

Il primo utente ad iscriversi riceve un account *Admin*. In grado di modificare ed eliminare pagine senza richiedere il permesso dell'autore.

3.0.2 RegisterPage



The image shows a web browser window with a registration form. The title of the page is "Registrati per continuare". The form has two input fields: "Nome Utente" with the value "giuseppe" and "Password" with masked characters "*****". Below the fields are two buttons: "Registrati" and "Accedi". A link "Hai già un account??" is positioned to the left of the "Accedi" button. The browser window has a dark header bar with standard window controls.

Registrati per continuare

Nome Utente
giuseppe

Password

Registrati

Hai già un account?? Accedi

Figure 2: RegisterPage

In questa pagine è possibile registrarsi alla wiki inserendo un proprio nome utente e password.

Nota: Sia la password che il nome utente hanno dei caratteri minimi, inoltre il nome utente è unico per ogni utente della wiki e non è modificabile.

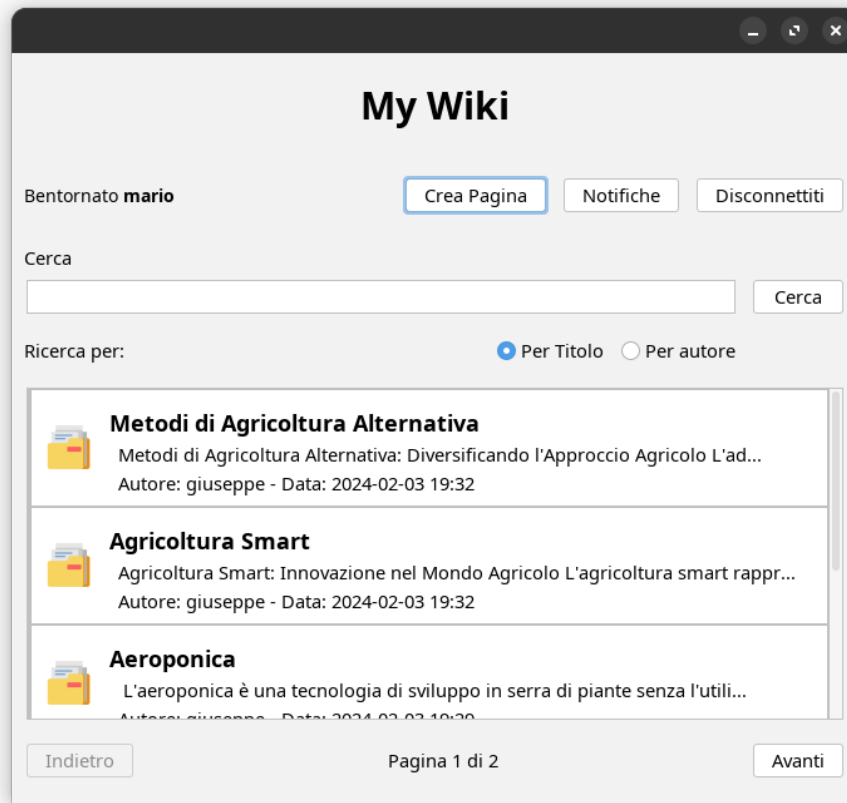


Figure 3: MainMenu

Questa è la pagina centra dalla wiki e anche un pò il suo centro di controllo da qui sono possibili fare le seguenti azioni:

- Visualizzare tutte le pagine presenti sulla wiki.
- Creare nuove pagine **se loggato** attraverso la pagina **PageCreate**.
- Leggere le proprie notifiche **se loggato** attraverso la pagina **UserNotifications**.

3.0.3 PageView

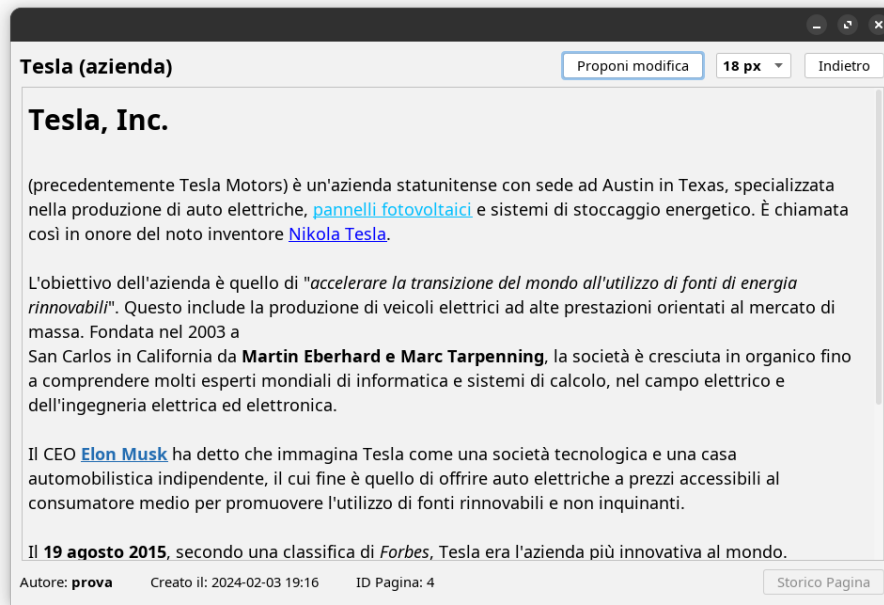


Figure 4: In figura è mostrata una pagina con la maggior parte delle funzioni presenti nell'editor della wiki

In questa finestra è possibile visualizzare una pagina presente sulla wiki. L'applicazione gestisce il testo utilizzando il linguaggio di markup **HTML** (**H**yper**T**ext **M**arkup **L**anguage).

L'utente ha accesso alle seguenti azioni:

- Proporre una modifica alla pagina attraverso la pagina **PageEdit**.
- Modificare la grandezza del font.
- Visualizzare lo storico delle modifiche della pagina andando alla pagina **PageHistory**.

3.0.4 PageCreate

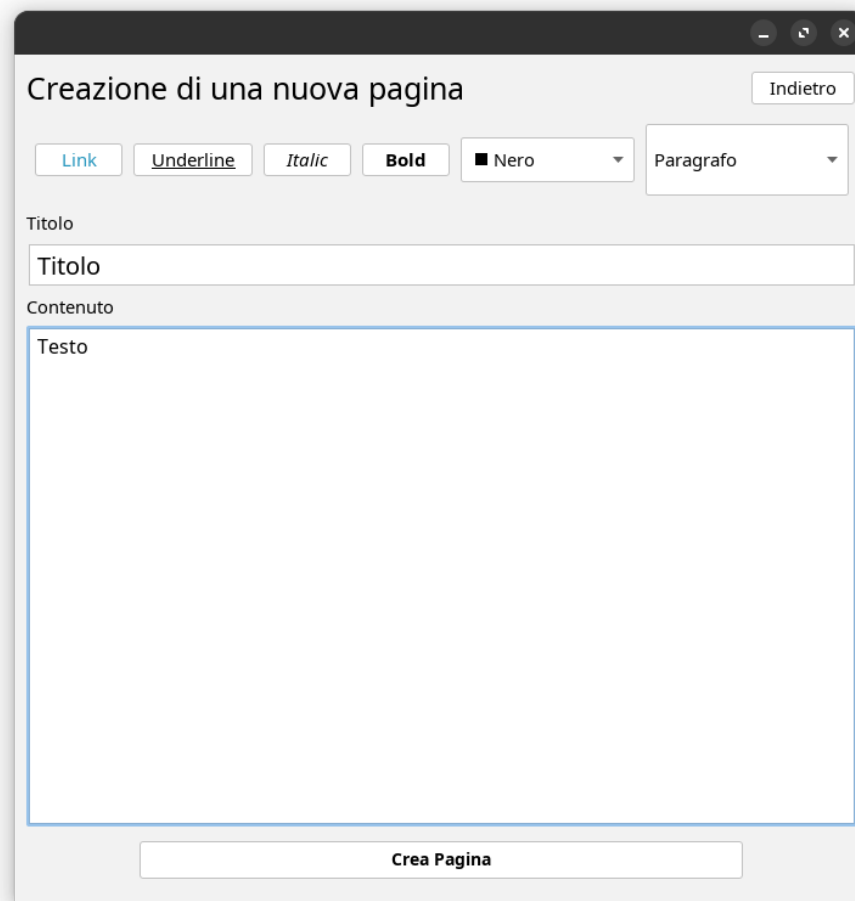


Figure 5: Finestra in cui è possibile creare una nuova pagina

3.0.5 PageEdit

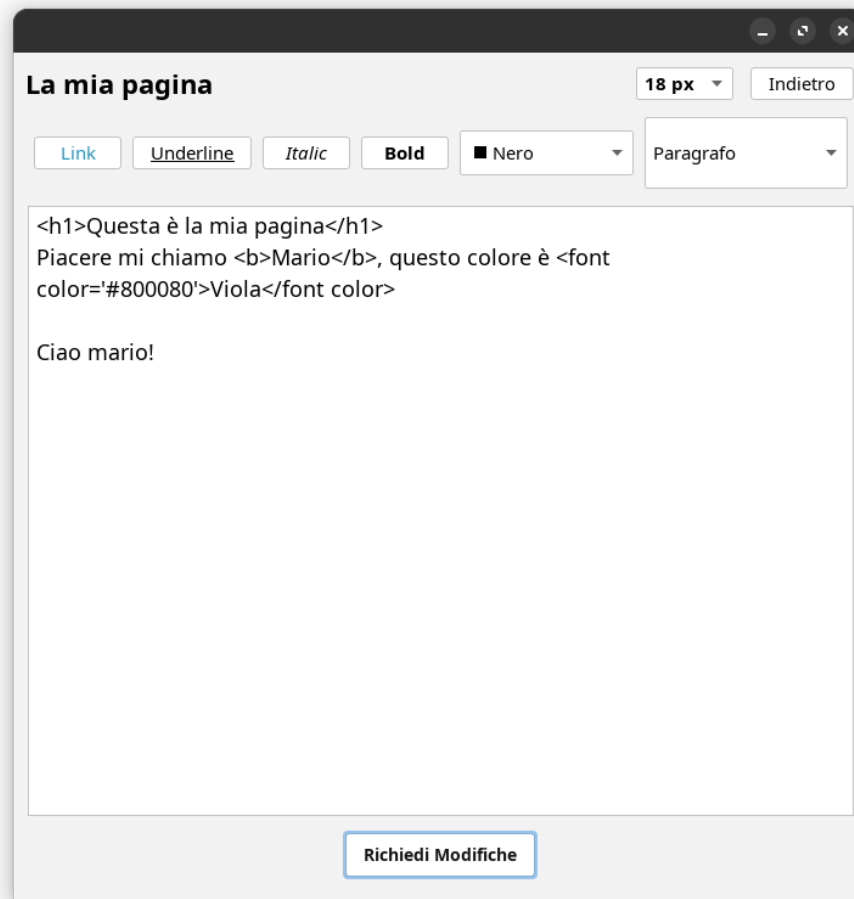


Figure 6: PageEdit

In questa pagina é possibile modificare una pagina a proprio piacimento.

Nota: Questa azione non modificherà mai la pagina direttamente ma creerà sempre prima una richiesta all'utente proprietario della pagina. Tuttavia se l'utente che richiede una modifica è un *Admin* o il proprietario della pagina, la modifica verrà accettata automaticamente.

3.0.6 PageHistory

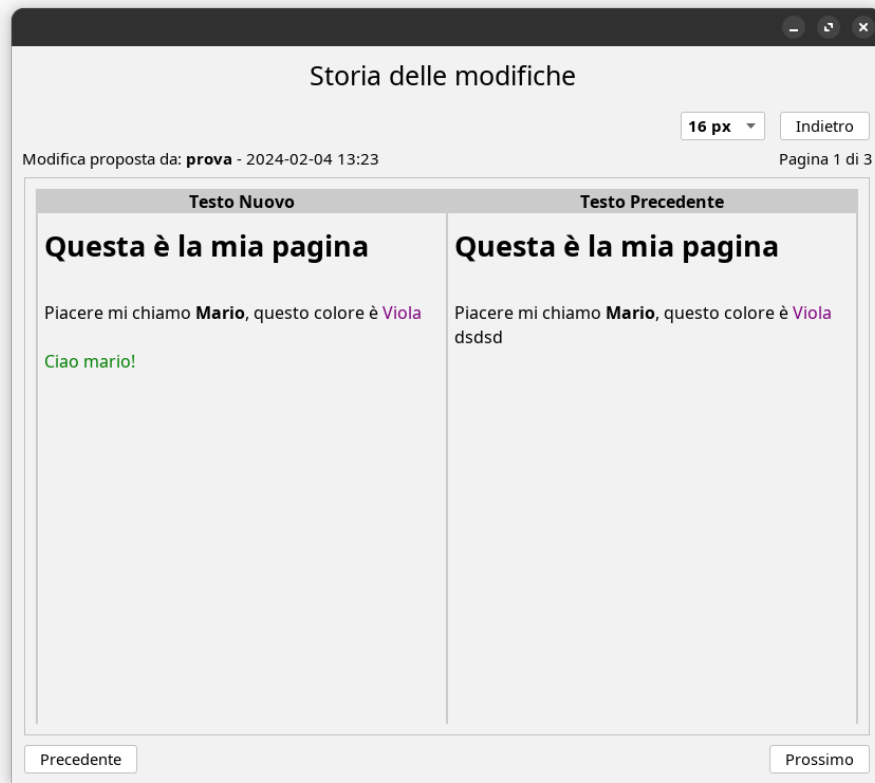


Figure 7: Figura che mostra la cronologia delle modifiche di una pagina

3.0.7 UserNotifications

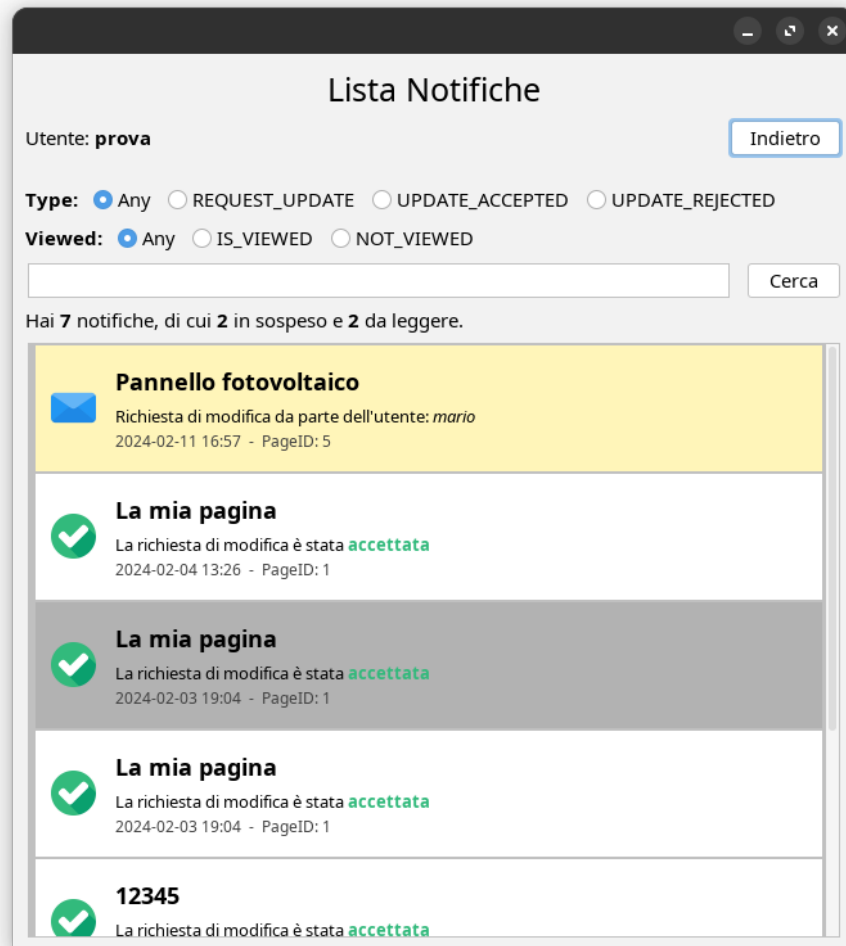


Figure 8: In giallo, modifiche non lette.

In questa pagina sono mostrate tutte le notifiche di un utente, tra cui anche le richieste di modifica pagina che quando cliccate portano alla pagina **ReviewUpdates**.

3.0.8 ReviewUpdates

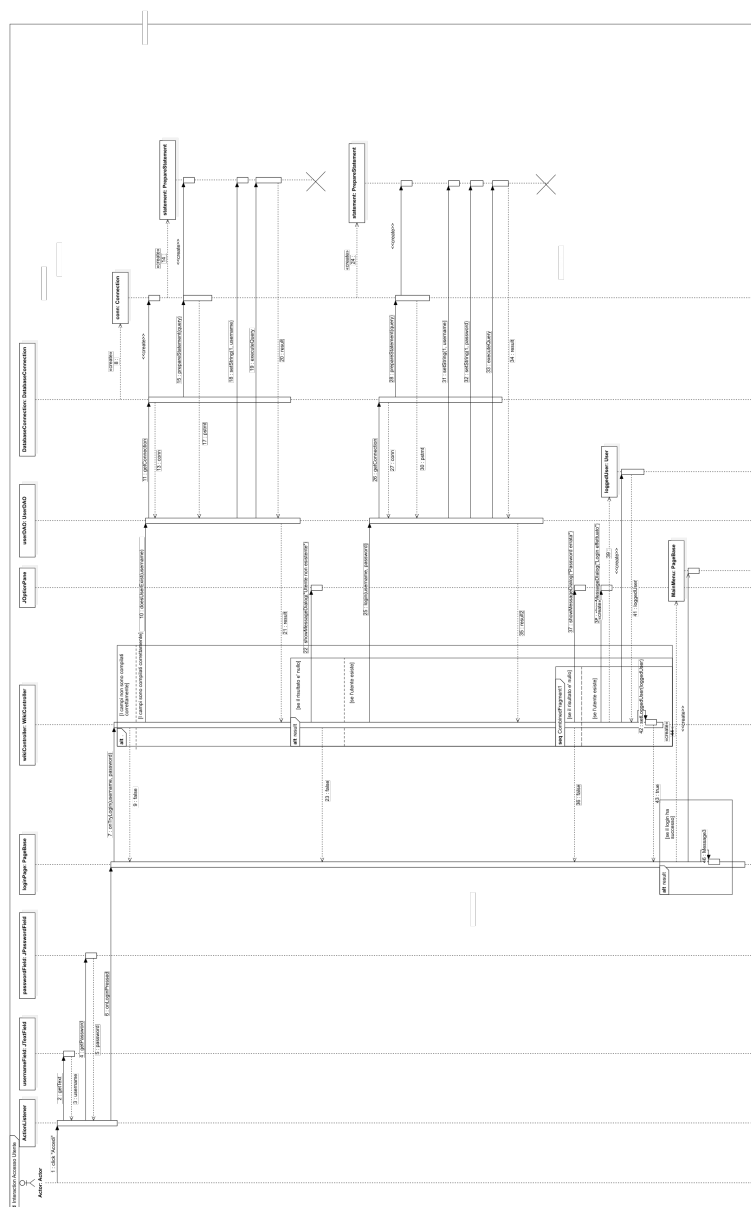


Figure 9: ReviewUpdates

Pagina in cui accettare o modificare le modifiche proposte ad una tua pagina. Le modifiche possono anche essere accettate in bulk (e più di una).

4 Sequence Diagram

4.1 Login (senza notifiche)



4.2 Modifica Pagina

