

Gestione di immagini

Sommario

0	Informazioni generali.....	5
1	Documentazione versione “pure HTML”	6
1.1	Progettazione base di dati	6
1.1.1	Da specifica a schema concettuale.....	6
1.1.2	Da schema concettuale a schema logico.....	8
1.2	Bean e DAO.....	10
1.2.1	Bean.....	10
1.2.1.1	Ulteriori vincoli sui dati	10
1.2.1.2	Bean class diagram.....	11
1.2.2	DAO.....	12
1.2.2.1	DAO class diagram	13
1.2.2.2	Gestione delle transazioni.....	13
1.2.2.3	Prevenzione da attacchi “SQL injection”.....	14
1.3	Application requirements analysis	15
1.3.1	Analisi della consegna	15
1.3.2	Definizione dei controller e dei template.....	17
1.3.3	La gestione delle immagini.....	18
1.3.3.1	Caricamento di un’immagine nel file system.....	18
1.3.3.2	Reperimento di un’immagine dal file system	19
1.4	Application design (IFML diagrams)	20
1.4.1	Parte “pubblica”	20
1.4.2	Parte “privata”	21
1.5	Sequence diagrams.....	22
1.5.1	Eventi parte “pubblica”	22
1.5.1.1	Accesso alle pagine di registrazione e login (non indicato nella lista di eventi estrapolati dalla specifica)	22
1.5.1.2	Registration form submit	23
1.5.1.3	Login form submit	24
1.5.2	Eventi parte “privata”	25
1.5.2.1	Click su miniatura immagine	25

1.5.2.2	Click sul nome di un album	25
1.5.2.3	Click sui bottoni di navigazione dell'ALBUM PAGE	26
1.5.2.4	Click su "cancella album"	26
1.5.2.5	Click su "invia commento"	26
1.5.2.6	Click su "cancella immagine"	27
1.5.2.7	Click su "carica immagine"	27
1.5.2.8	Click su "crea album"	28
1.5.2.9	Click su "log-out"	29
1.5.2.10	Click su "torna alla homepage"	29
1.5.2.11	Click su "torna all'album"	29
2	Documentazione versione "con JavaScript"	30
2.1	Ampliamento base di dati da "pure HTML"	30
2.2	Modifica Bean e DAO da "pure HTML"	32
2.3	Application requirements analysis	33
2.3.1	Analisi della consegna	33
2.3.2	Definizione dei controller e degli oggetti JS	34
2.4	Application design (IFML diagrams)	36
2.4.1	Parte "pubblica"	36
2.4.2	Parte "privata"	36
2.5	Sequence diagrams.....	37
2.5.1	Eventi parte "pubblica"	37
2.5.1.1	Registration form submit	37
2.5.1.2	Login form submit	37
2.5.1.3	Click per chiusura del paragrafo di errore	37
2.5.2	Eventi parte "privata"	38
2.5.2.1	Click su miniatura immagine	38
2.5.2.2	Click sul nome di un album	38
2.5.2.3	Click sui bottoni di navigazione dell'ALBUM PAGE	39
2.5.2.4	Click su "cancella album"	39
2.5.2.5	Click su "invia commento" (non dalla finestra modale)	40
2.5.2.6	Click su "cancella immagine"	40
2.5.2.7	Click su "carica immagine"	40
2.5.2.8	Click su "crea album"	41
2.5.2.9	Click su "log-out"	41
2.5.2.10	Click su "torna alla homepage"	42

2.5.2.11	Click su "torna all'album"	42
2.5.2.12	"mouseover" su miniatura.....	42
2.5.2.13	Click sul bottone per inviare un commento nella finestra modale.....	43
2.5.2.14	"dragover" su immagini nel form per l'ordinamento personalizzato.....	43
2.5.2.15	Click su "salva ordinamento"	44

0 Informazioni generali

Realizzato da:	Andrea Bellani (codice persona: 10733192, matricola: 956505)
Data ultima modifica	05/07/2024
Appello d'esame scelto	luglio 2024
Data colloquio assegnata	10/7/2024

Software utilizzati

Creazione basi di dati	<ul style="list-style-type: none"> MySQL Workbench 8.0 CE
Tool implementazione applicazioni web	<ul style="list-style-type: none"> Sublime Text 3 Eclipse IDE for Enterprise Java and Web Developers – 2024-03 JDK 19 Apache Tomcat 9.0.86 Google Chrome 126.0.6478.127
Principali librerie esterne	<ul style="list-style-type: none"> gson 2.8.6 thymeleaf 3.0.11 apache.commons.validator
Design schemi ER e logici	<ul style="list-style-type: none"> miro.com
Design diagrammi UML	<ul style="list-style-type: none"> Astah UML
Design diagrammi IFML	<ul style="list-style-type: none"> miro.com ifmledit.org

1 Documentazione versione “pure HTML”

1.1 Progettazione base di dati

1.1.1 Da specifica a schema concettuale

Ritagliamo dalla specifica le informazioni utili alla progettazione della base di dati per la versione “pure HTML”:

La registrazione controlla la validità sintattica dell’indirizzo di email e l’uguaglianza tra i campi “password” e “ripeti password”. La registrazione controlla l’unicità dello username. Ogni immagine è memorizzata come file nel file system del server su cui l’applicazione è rilasciata. Inoltre nella base di dati sono memorizzati i seguenti attributi: un titolo, una data di creazione, un testo descrittivo e il percorso del file dell’immagine nel file system del server. Le immagini sono associate all’utente che le carica. L’utente può creare album dalla HOME PAGE e associare a questi le proprie immagini. Un album ha un titolo, il creatore e la data di creazione. La stessa immagine può appartenere a più di un album. Le immagini sono associate a uno o più commenti inseriti dagli utenti (dal proprietario o da altri utenti). Un commento ha un testo e il nome dell’utente che lo ha creato.

Suddividiamo le parti utili alla progettazione della specifica in:

- entità e relativi attributi;
- relazioni;
- altre informazioni utili alla progettazione.

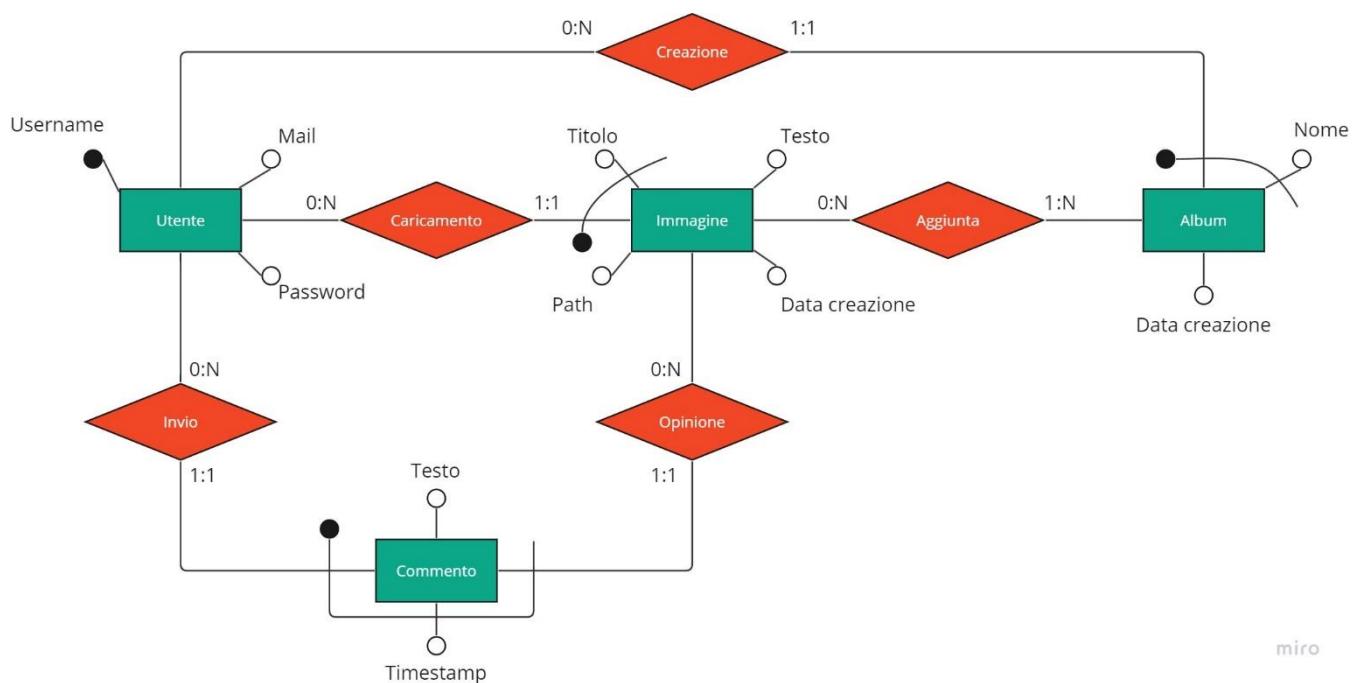
Identifichiamo così (in **grassetto** gli attributi scelti come identificativi):

- entità:
 - “utente” : **username**, mail e password;
 - “immagine” : **titolo**, data creazione, descrizione, path [**univoco tra le immagini di un utente**]
 - “album” : **titolo**, data creazione [**univoco tra gli album di un utente**]
 - “commento” : **timestamp**, testo [**univoco tra tutti i commenti di utente per un’immagine**]
- relazioni:
 - a un utente corrispondono da “0” a “N” immagini e un’immagine corrisponde ad uno e un solo utente;
 - a un utente corrispondono da “0” a “N” album e un album corrisponde uno e un solo utente;
 - un’immagine può appartenere a “0” o a “N” album e un album contiene da “1” ad “N” immagini;

- un utente può creare da “0” a “N” commenti e un commento è creato da uno e un solo utente;
- un commento corrisponde a una e una sola immagine e un’immagine può avere da “0” a “N” commenti.

Si noti che, benché non esplicitamente richiesto nella specifica, abbiamo aggiunto l’attributo “timestamp” all’entità “commento”, il quale fornisce un pratico identificativo relativo a un commento di utente su un’immagine. In questo modo possiamo distinguere commenti di uno stesso utente per una stessa immagine. In alternativa, avremmo potuto utilizzare un identificato (numerico) univoco tra tutti i commenti di uno stesso utente su una stessa immagine; l’utilizzo di un timestamp ci sembrava un’informazione ugualmente efficace all’identificazione ma più interessante per gli utenti (gli utenti potrebbero essere interessati a conoscere il timestamp in cui un commento è stato creato).

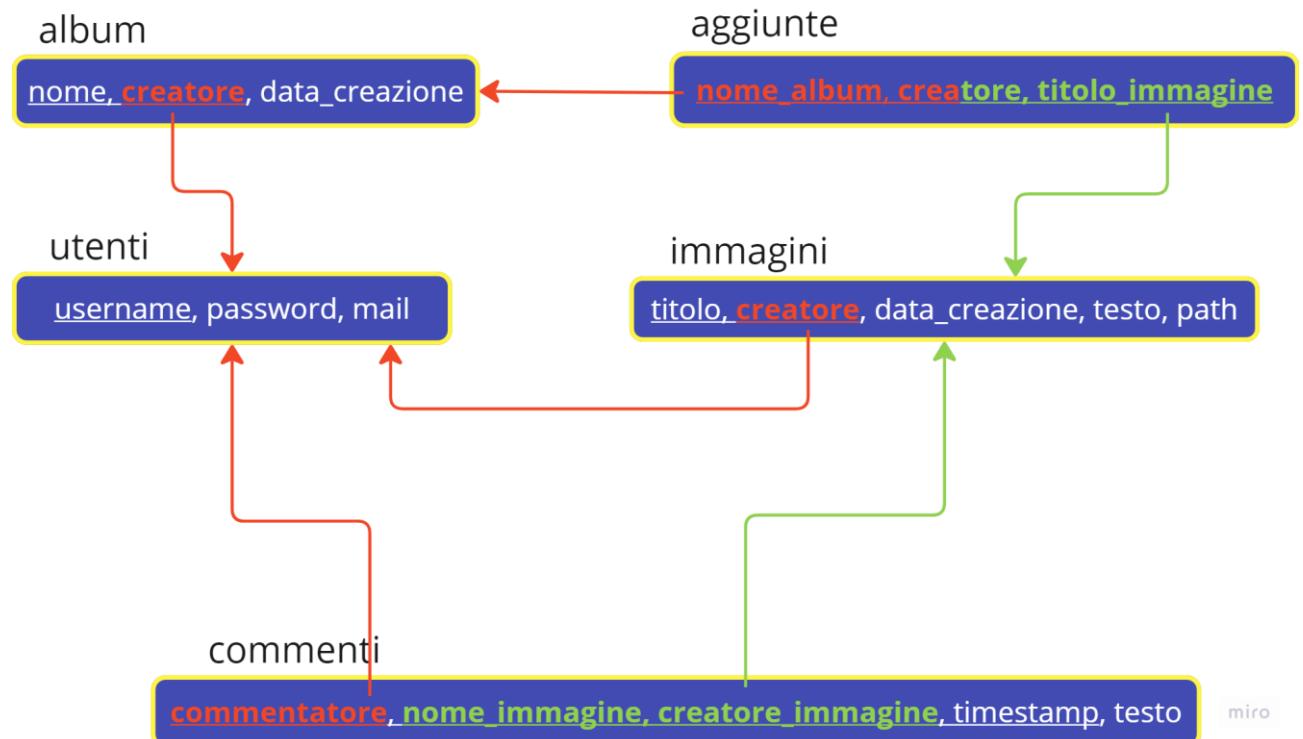
Abbiamo ora tutte le informazioni che realizzare lo schema ER della base di dati:



In questo schema ER non viene rappresentato il vincolo “un utente non può aggiungere a un album immagini caricate da altri utenti”. Per garantire questo utilizzeremo i vincoli di integrità referenziale del modello logico relazionale.

1.1.2 Da schema concettuale a schema logico

Nello schema ER progettato non sono presenti costrutti non rappresentabili nel modello relazionale possiamo così procedere alla traduzione in schema relazionale:



Nel dettaglio, i domini degli attributi e i relativi vincoli:

	Attributo	Dominio	Vincoli
utenti	username	VARCHAR(50)	NOT NULL, PRIMARY KEY
	password	VARCHAR(50)	NOT NULL
	mail	VARCHAR(319)	NOT NULL
immagini	titolo	VARCHAR(50)	NOT NULL
	creatore	VARCHAR(50)	NOT NULL
	data_creazione	DATE	NOT NULL
	testo	VARCHAR(500)	NOT NULL
	path	VARCHAR(500)	NOT NULL
album	nome	VARCHAR(50)	NOT NULL
	creatore	VARCHAR(50)	NOT NULL
	data_creazione	DATE	NOT NULL

commenti	commentatore	VARCHAR(50)	NOT NULL
	nome_immagine	VARCHAR(50)	NOT NULL
	creatore_immagine	VARCHAR(50)	NOT NULL
	timestamp	TIMESTAMP	NOT NULL
	testo	VARCHAR(500)	NOT NULL
aggiunte	nome_album	VARCHAR(50)	NOT NULL
	creatore	VARCHAR(50)	NOT NULL
	titolo_immagine	VARCHAR(50)	NOT NULL

Vincoli di tupla:

immagini	PRIMARY KEY(titolo, creatore)
	FOREIGN KEY(creatore) REFERENCES utenti(username)
	ON DELETE CASCADE ON UPDATE CASCADE
album	PRIMARY KEY(nome, creatore)
	FOREIGN KEY(creatore) REFERENCES utenti(username)
	ON DELETE CASCADE ON UPDATE CASCADE
commenti	PRIMARY KEY(commentatore, nome_immagine, creatore_immagine, timestamp)
	FOREIGN KEY(commentatore) REFERENCES utenti(username)
	ON DELETE CASCADE ON UPDATE CASCADE
	FOREIGN KEY(nome_immagine, creatore_immagine) REFERENCES immagini(titolo, creatore)
aggiunte	ON DELETE CASCADE ON UPDATE CASCADE
	PRIMARY KEY(nome_album, creatore, titolo_immagine)
	FOREIGN KEY(nome_album, creatore) REFERENCES album(nome, creatore)
	ON DELETE CASCADE ON UPDATE CASCADE
	FOREIGN KEY(creatore, titolo_immagine) REFERENCES immagini(creatore, titolo)
	ON DELETE CASCADE ON UPDATE CASCADE

Il vincolo che non siamo stati in grado di esprimere nel modello ER “un utente non può aggiungere a un album immagini caricate da altri utenti” è stato implementato utilizzando un unico attributo “creatore” nella tabella “aggiunte”, sul quale sono applicati ambedue i vincoli di integrità referenziale con le tabelle “Album” e “immagini”. Detto in altri termini, un’immagine può essere aggiunta a un album solo se il creatore dell’album ha lo stesso username (il quale è identificativo di un utente) dell’utente che ha caricato l’immagine.

1.2 Bean e DAO

1.2.1 Bean

1.2.1.1 Ulteriori vincoli sui dati

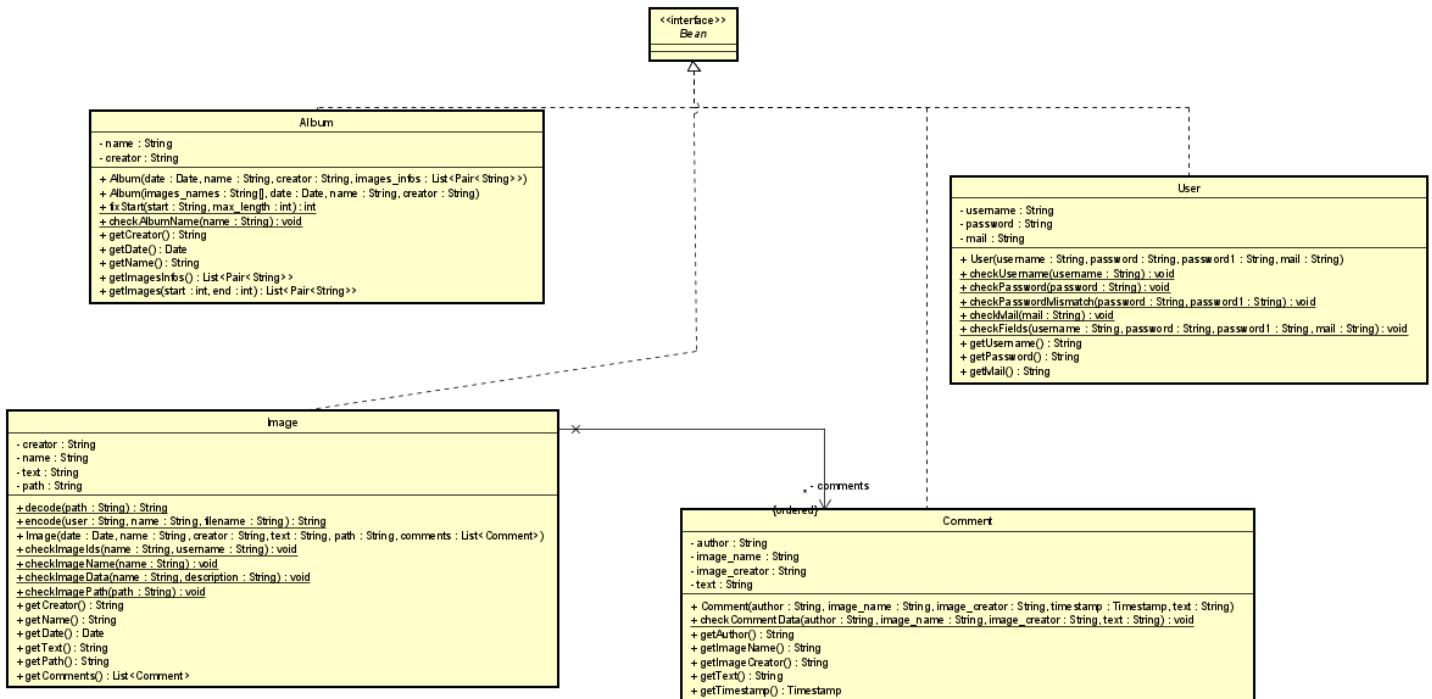
L'applicazione effettua i seguenti controlli sui dati non espressi nell'SQL:

- vincoli sull'utente:
 - un indirizzo mail valido deve soddisfare una specifica sintassi (deve contenere uno e uno solo “@”, non può contenere “.” consecutivi, deve contenere almeno un “.” dopo alla “@” ma non immediatamente dopo ecc.). Utilizziamo la funzione di libreria “isValid()” da “org.apache.commons.validator.routines.EmailValidator”;
- vincoli sulle immagini:
 - i file caricabili devono essere immagini (un file “.pdf”, ad esempio, non è considerabile come un’immagine);
- vincoli sugli album:
 - non è ammesso creare album vuoti (la validità di questo vincolo deve essere rispettata anche a seguito di modificazioni dei dati da parte dell’utente: quando l’utente elimina un’immagine vengono eliminati anche tutti gli album che contenevano solamente quell’immagine);

I seguenti vincoli (come anche i vincoli sulla dimensione e l’obbligatorietà dei campi) vengono controllati lato-server da metodi statici (in questo modo possono essere invocati anche senza istanziare oggetti bean) che vengono invocati sia dai DAO che dai costruttori dei bean, in questo modo si ha la garanzia che un qualunque oggetto bean istanziato con successo ha valori assunti dai propri attributi conformi ai vincoli definiti.

1.2.1.2 Bean class diagram

Il design dei bean rifrasa, con opportune semplificazioni, le relazioni tra tabelle dello schema logico (si omettono i metodi “get” degli attributi privati):



Gli unici metodi presenti che non compiono azioni legate alla rappresentazione dei dati sono il metodo “fixStart” della classe “Album”, il quale è utilizzato per “correggere” il parametro “start” nella “ALBUM PAGE” (verrà spiegato meglio nella sezione dedicata ai sequence diagram) e i metodi “encode” e “decode” della classe “Image”, i quale viene utilizzato per codificare/decodificare i path in cui memorizzare le immagini.

1.2.2 DAO

Evidenziamo nella specifica le parti che identificano le operazioni sulla base di dati (escludendo le informazioni utili alla definizione dello schema, già analizzate nei paragrafi precedenti, e le informazioni relative alla singola estrazione di informazioni relative “singoli elementi”):

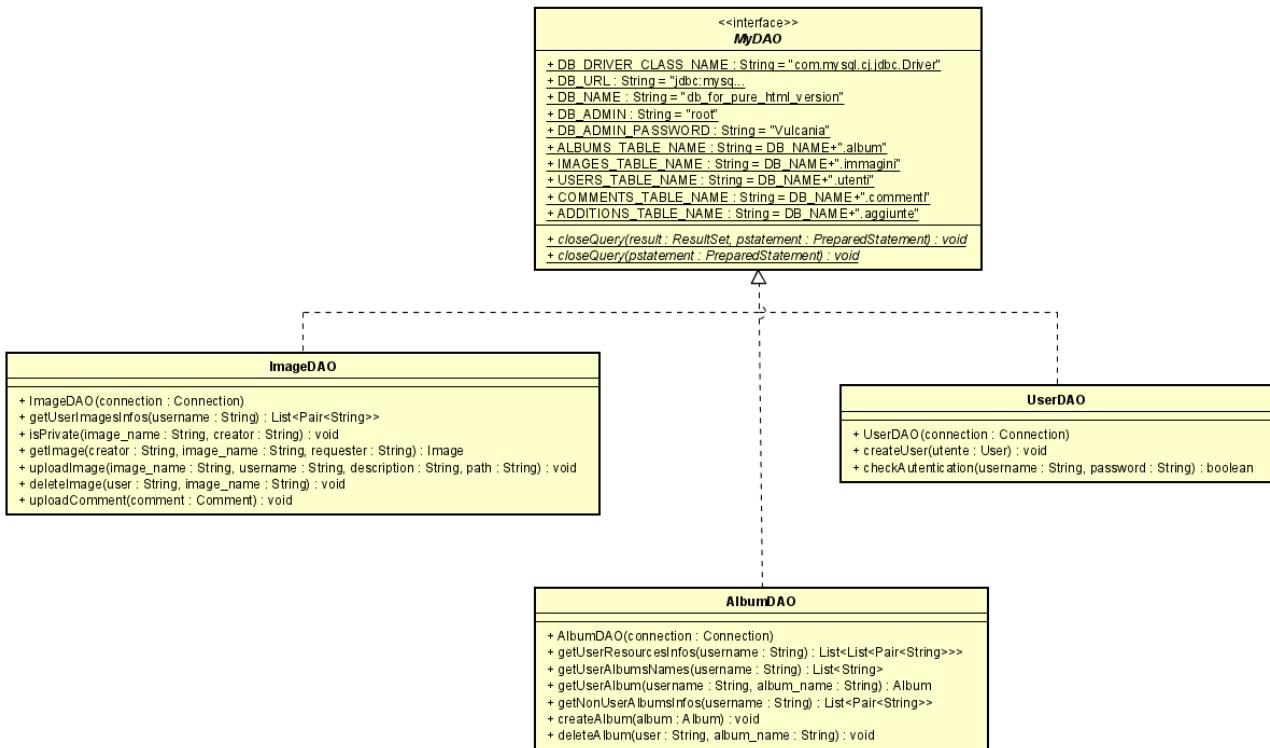
L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form. Quando l'utente accede all'HOME PAGE, questa presenta l'elenco degli album che ha creato e l'elenco degli album creati da altri utenti. Entrambi gli elenchi sono ordinati per data di creazione decrescente. La pagina mostra anche una form per aggiungere un commento e un bottone per cancellare l'immagine e tutti i commenti ad essa associati.

- operazioni relative agli utenti;
- operazioni relative alle immagini;
- operazioni relative agli album.

Distinguiamo così tre DAO (in **rosso** le operazioni non esplicitamente richieste dalla specifica ma aggiunte per propria comodità/usabilità):

- “userDAO” :
 - verifica validità login;
- “imageDAO” :
 - cancellazione immagine (e relativi commenti);
 - **estrazione dati immagini caricate dall'utente;**
- “albumDAO” :
 - estrazione dati album creati dall'utente;
 - estrazione dati album creati da altri utenti;
 - **cancellazione album.**

1.2.2.1 DAO class diagram



Si noti che, ove possibile (ovvero, ove la cosa non porterebbe il codice delle classi utilizzatrici dei DAO a complicarsi senza motivo), i metodi dei DAO non ricevono input/restituiscono in output parametri “non raggruppati” in bean. Tutti i metodi dei DAO verificano comunque la conformità dei parametri di input grazie ai metodi statici delle classi bean; ricevere in input oggetti bean già istanziati assicura che la validità dei loro attributi siano stati già verificati.

1.2.2.2 Gestione delle transazioni

Nella nostra applicazione identifichiamo tre modalità per garantire l’atomicità di sequenze critiche di operazioni con la base di dati:

- metodo di un DAO che esegue un’unico statement e che nessuna servlet utilizza in sequenza con altri metodi dei DAO:** modalità auto-commit attivata di default;
- metodo di un DAO che esegue più statement ma che nessuna servlet utilizza in sequenza con altri metodi dei DAO:** sostituzione di auto-commit con “commit()” e “rollback()” “manuali”;
- metodi di DAO utilizzati in sequenza da una stessa servlet:** raggruppamento della sequenza di chiamate dei metodi che farebbe la servlet in un unico metodo su cui vengono effettuati “commit()” e “rollback()” “manuali”.

L’unico in caso in cui si rende necessario l’utilizzo della terza modalità (la quale è l’unica che ha un impatto sul design delle classi DAO) è il metodo “getUserResourcesInfos()” di “AlbumDAO”, il quale non fa altro che chiamare i tre metodi “getUserImagesInfos()”, “getUserAlbumsNames()” e

“getNonUserAlbumsInfos()” garantendo però l’atomicità dell’esecuzione dei tre metodi. Questo, come mostrano i paragrafi seguenti, è utilizzando nella homepage, la quale presenta all’utente le proprie immagini, i propri album e gli album degli altri utenti. Se non si fosse creato un metodo che atomicizzasse questa sequenza di chiamate, un utente avrebbe potuto vedere nella propria homepage contenuti non coerenti allo stato attuale della base di dati (ad esempio, se tra la chiamata di “getUserImagesInfos()” e la chiamata di “getUserAlbumsNames()” si fosse cercato, da un’altra sessione, di cancellare un’immagine, l’utente avrebbe potuto visionare ancora quella immagine nella homepage pur non vedendo gli album che la contenevano come unica immagine).

1.2.2.3 Prevenzione da attacchi “SQL injection”

Tutte le query alla base di dati vengono eseguite seguendo il seguente pattern:

```
public void createUser (User utente) throws DbErrorException
{
    String update = "INSERT INTO " +MyDAO.USERS_TABLE_NAME+ " VALUES (?,?,?);";
    PreparedStatement pstatement = null;

    try
    {
        pstatement = connection.prepareStatement(update);

        pstatement.setString(1, utente.getUsername());
        pstatement.setString(2, utente.getPassword());
        pstatement.setString(3, utente.getMail());

        pstatement.executeUpdate();
    }
    catch (SQLException e)
    {
        System.out.println("Error in insert user");
    }
}
```

ovvero, i valori provenienti dal client (password, nomi immagini/album, testi commenti ecc.) sono inseriti in template al posto dei placeholder mediante un “prepared statement”. In questo modo si scongiurano attacchi di tipo injection poiché i valori inseriti nei template vengono interpretati secondo la “forma” del template e non come se fossero, a priori, “parti di istruzioni” SQL.

1.3 Application requirements analysis

1.3.1 Analisi della consegna

Analizziamo ora la consegna al fine di identificare:

- le pagine web da realizzare;
- gli elementi grafici delle pagine web;
- gli eventi scatenabili;
- le azioni da effettuare al seguito degli eventi;

L'applicazione supporta **registrazione** e **login** mediante una **pagina pubblica** con **opportune form**. Le immagini sono associate all'utente che le **carica**. L'utente può **creare album** dalla **HOME PAGE** e associare a questi le proprie immagini. Le immagini sono associate a uno o più commenti inseriti dagli utenti (dal proprietario o da altri utenti). Un commento ha un testo e il nome dell'utente che lo ha **creato**. Quando l'utente **accede** all'**HOME PAGE**, questa presenta l'**elenco degli album** che ha **creato** e l'**elenco degli album** creati da altri utenti. Quando l'utente **clicca** su un album che appare negli elenchi della **HOME PAGE**, appare la pagina **ALBUM PAGE** che contiene inizialmente **una tabella** di una riga e cinque colonne. Ogni cella contiene **una miniatura (thumbnail)** e il **titolo** dell'immagine. Se l'album contiene più di cinque immagini, sono disponibili comandi per vedere il **precedente** e **successivo** insieme di cinque immagini. Quando l'utente **seleziona** una miniatura, una pagina **IMAGE PAGE** mostra tutti i dati dell'immagine scelta, tra cui la stessa immagine a grandezza naturale e i commenti eventualmente presenti. La pagina mostra anche una **form** per aggiungere un **commento** e un **bottone** per cancellare l'immagine e tutti i commenti ad essa associati. La pagina **IMAGE PAGE** contiene collegamenti per tornare all'**HOME PAGE** e alla **pagina ALBUM PAGE**. La pagina **ALBUM PAGE** contiene un collegamento per tornare all'**HOME PAGE**. L'applicazione consente il **logout** dell'utente.

Dunque, il sito web sarà costituito da diverse pagine “pubbliche” e “private” (in **rosso** quelle non esplicitamente delineate dalla consegna) contenenti diversi oggetti grafici (in **rosso** quelli non esplicitamente delineati della consegna):

- pagine “pubbliche”:
 - pagina per la registrazione:
 - registration form;
 - e una per la login:
 - login form.
- pagine “private”:
 - HOME PAGE:
 - **elenco delle proprie immagini;**
 - elenco dei propri album;
 - elenco degli album altrui;
 - ALBUM PAGE:
 - tabella per i dati dell’album;
 - tabella per le “5” immagini;
 - bottone “PRECEDENTI”;
 - bottone “SUCCESSIVE”;
 - **bottone per cancellare l’album;**
 - IMAGE PAGE:
 - tabella per i dati dell’immagine e relativi commenti;
 - form per l’aggiunta di un commento;
 - bottone per cancellare l’immagine;
 - **UPLOAD IMAGE PAGE:**
 - **upload image form;**
 - **CREATE ALBUM PAGE:**
 - **create album form.**

Per comodità, in tutte le pagine “private” è presente una menubar che presenta per quella pagina i pulsanti di utilità, quali:

- bottone per il “logout”;
- bottone per tornare alla homepage;
- bottone per tornare all’ALBUM PAGE (nel caso di “IMAGE PAGE” raggiunte da “ALBUM PAGE”).

Infine, associamo i possibili eventi generati dall'utente con le azioni da intraprendere:

- registration form submit ⇒ creazione nuova utenza ⇒ accesso HOME PAGE;
- login form submit ⇒ autenticazione ⇒ accesso HOME PAGE;
- click su una miniatura ⇒ accesso IMAGE PAGE;
- click sul nome di un album ⇒ accesso ALBUM PAGE con le prime “5” immagini;
- click su “PRECEDENTI”/“SUCCESSIVE” ⇒ presentazione delle precedenti/seguenti “5” immagini;
- click su “cancella album” ⇒ cancellazione dell’album;
- click su “invia commento” ⇒ creazione nuovo commento;
- click su “cancella immagine” ⇒ cancellazione dell’immagine;
- click su “carica immagine” ⇒ caricamento immagine e inserimento dei relativi dati;
- click su “crea album” ⇒ creazione album e associazione delle immagini;
- click su “log-out” ⇒ disconnessione dalla sessione;
- click su “torna alla homepage” ⇒ accesso HOME PAGE;
- click su “torna all’album” ⇒ accesso ALBUM PAGE.

1.3.2 Definizione dei controller e dei template

Definiamo due tipi di servlet:

- “pages creators” : le servlet atte a popolare i template Thymeleaf, ne abbiamo una per ogni pagina (anche le pagine “pubbliche”, scriveranno nei template gli eventuali errori lato-server) e li diamo il nome delle pagine di cui popolano il template;
- “action performers” : le servler atte ad eseguire le azioni (esclusi i rendirizzamenti). Le distinguiamo per avere nome che comincia per “Do”:
 - “DoRegister” : creazione nuova utenza;
 - “DoLogin” : autenticazione;
 - “DoDeleteAlbum” : cancellazione album;
 - “DoUploadComment” : creazione commento;
 - “DoDeleteImage” : cancellazione immagine;
 - “DoUploadImage” : caricamento immagine;
 - “DoCreateAlbum” : creazione album;
 - “DoLogout” : disconessione.

Oltre a queste, avremo anche una servlet “ImagesHandler” per il reperimento delle immagini dal file system.

Mentre i template rispecchiamo le pagine previste dalla specifica:

- index.html (pagina statica);
- register.html (template, ma solo per la stampa dell'errore);
- login.html (template, ma solo per stampa dell'errore);
- homepage.html;
- upload_image.html (template, ma solo per la stampa dell'errore);
- image_page.html;
- create_album.html (template, ma solo per la stampa dell'errore);
- album_page.html.

1.3.3 La gestione delle immagini

Ad eccezione del vincolo sul “dove” memorizzare le immagini (nel file system anziché nella base di dati), la consegna non dà vincoli sulla maniera in cui le immagini vanno memorizzate e reperite dal file system. Presentiamo allora il metodo che abbiamo scelto e le motivazioni a supporto.

1.3.3.1 Caricamento di un’immagine nel file system

Abbiamo pensato che l’utente medio gradisce che un’immagine scaricata dal server abbia lo stesso nome dell’immagine in esso caricata. Se, ad esempio, l’utente desidera caricare un’immagine “paesaggio.jpg” desidererebbe poterla riscaricare con lo stesso nome. La cosa più comoda per noi è utilizzare un’unica grande cartella per memorizzare tutte le immagini degli utenti tuttavia, il semplice nome “paesaggio.jpg” può non essere univoco tra i nomi delle immagini caricate dagli altri utenti o persino da quello stesso utente. Risolviamo allora questo problema creando una sottocartella per ciascun utente chiamata con lo username dell’utente, al cui interno abbiamo ciascuna sottocartella per ciascuna immagine (chiamata col nome scelto per l’immagine, non con il nome del file dell’immagine) e in quella cartella mettiamo il file specifico (il “paesaggio.jpg”) che corrisponde a quell’immagine.

Risolto il problema dell’univocità del path abbiamo un ulteriore problema nel caricamento: vogliamo dare all’utente la possibilità di utilizzare dei caratteri unicode nel proprio username e nel nome delle proprie immagini e l’unicode non è, in generale, permesso nei nomi delle cartelle nei file system. Risolviamo questo problema (questo è ciò che fa il metodo “Image.encode()”) chiamando tutte le cartelle con la codifica in “UTF-8” dei nomi scelti, in questo modo l’utente può utilizzare qualunque carattere unicode nel proprio username e nelle proprie immagini.

Ad esempio, l’utente “admin” vuole caricare l’immagine “paesaggio.jpg” chiamandola “bel paesaggio 😊”, questa viene caricata nel path:

- admin\bel+paesaggio%C3%B0%C2%9F%C2%98%C2%8A\paesaggio.jpg

Questo esatto path viene memorizzato nella base di dati.

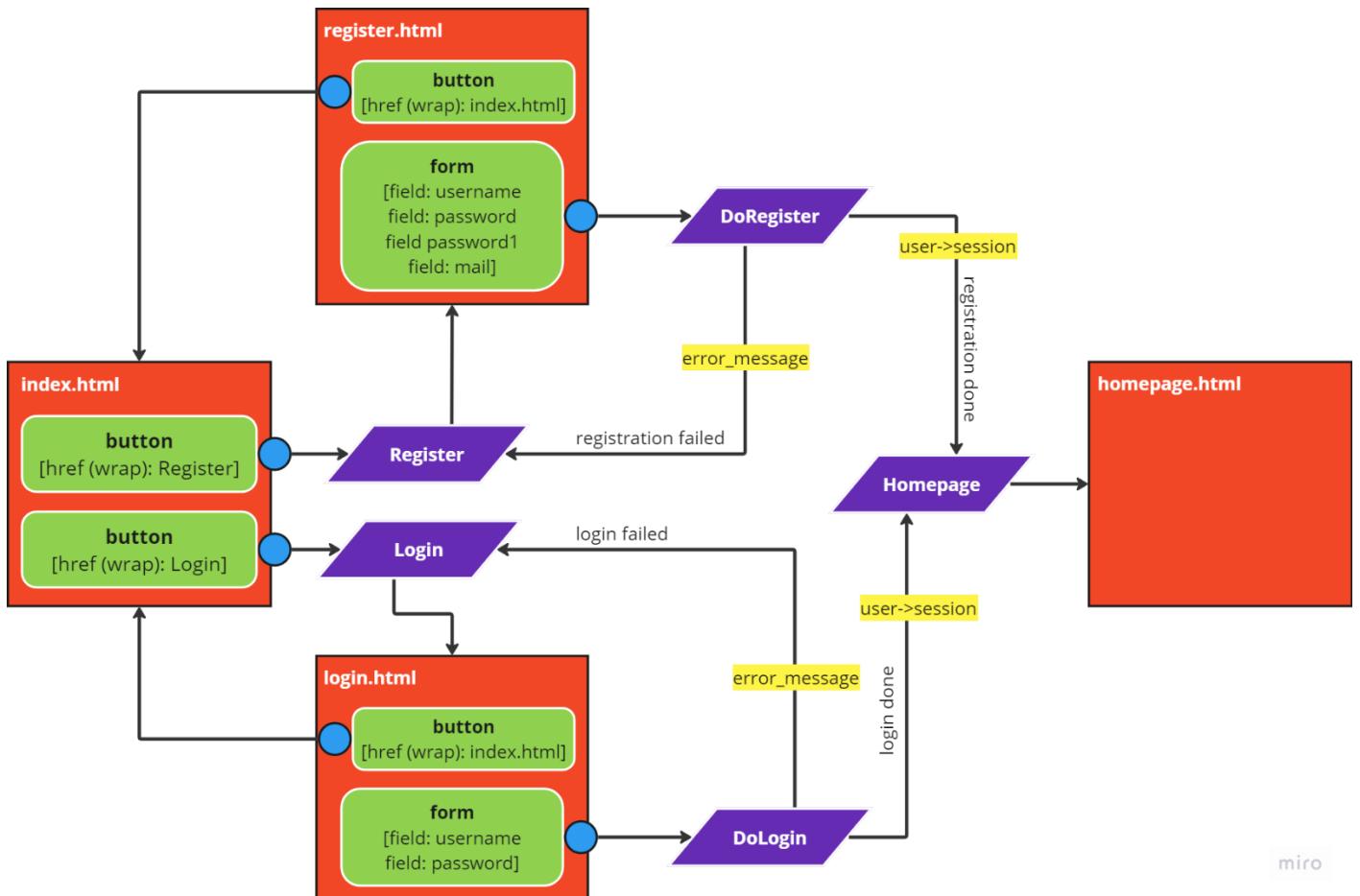
1.3.3.2 Reperimento di un'immagine dal file system

Il path che va fornito a “ImagesHandler” non è esattamente quello scritto nella base di dati. Per come è progettata “ImagesHandler” (ovvero risponde a richieste “GET” che contengono il path dell’immagine da reperire dal file system), l’utilizzo della codifica UTF-8 per tutte le parti del path è piuttosto rischioso. Infatti, il server cercherà di decodificare quel path prima di “darlo” alla servlet e non è quello che vogliamo, l’UTF-8 del path non va in alcun modo interpretato. Allora quello che facciamo è codificare in UTF-8 ancora una volta il path in modo tale che la decodifica che il server fa “in automatico” dell’url passato a “ImagesHandler” porti a decodificare l’ultima decodifica fatta e così “ImagesHandler” avrà il path così com’è stato scritto nella base di dati. In buona sostanza, abbiamo usato “UTF-8” per “incapsulare” un URI.

Infine, “ImagesHandler” controlla con un metodo di “ImageDAO” che l’utenza che richiede l’immagine o sia l’utenza che l’ha caricata oppure che sia una qualunque utenza registrata ma solo se quell’immagine è presente in almeno un album. Questo evita che immagini caricate ma non presenti in alcun album vengano viste da utenti diversi da chi le ha caricate. La specifica infatti non prevede la visione di immagini “singole”, ma solo immagini a partire dagli album che le contengono. Questo controllo è chiaramente fatto anche da “ImagePage” e da “DoPostComment” (in modo tale che se l’immagine diventasse “privata”, ovvero non presente in alcun album, in un istante successivo a quello del caricamento dell’ImagePage, un utente diverso dal proprietario di quell’immagine non potrebbe caricarvi commenti).

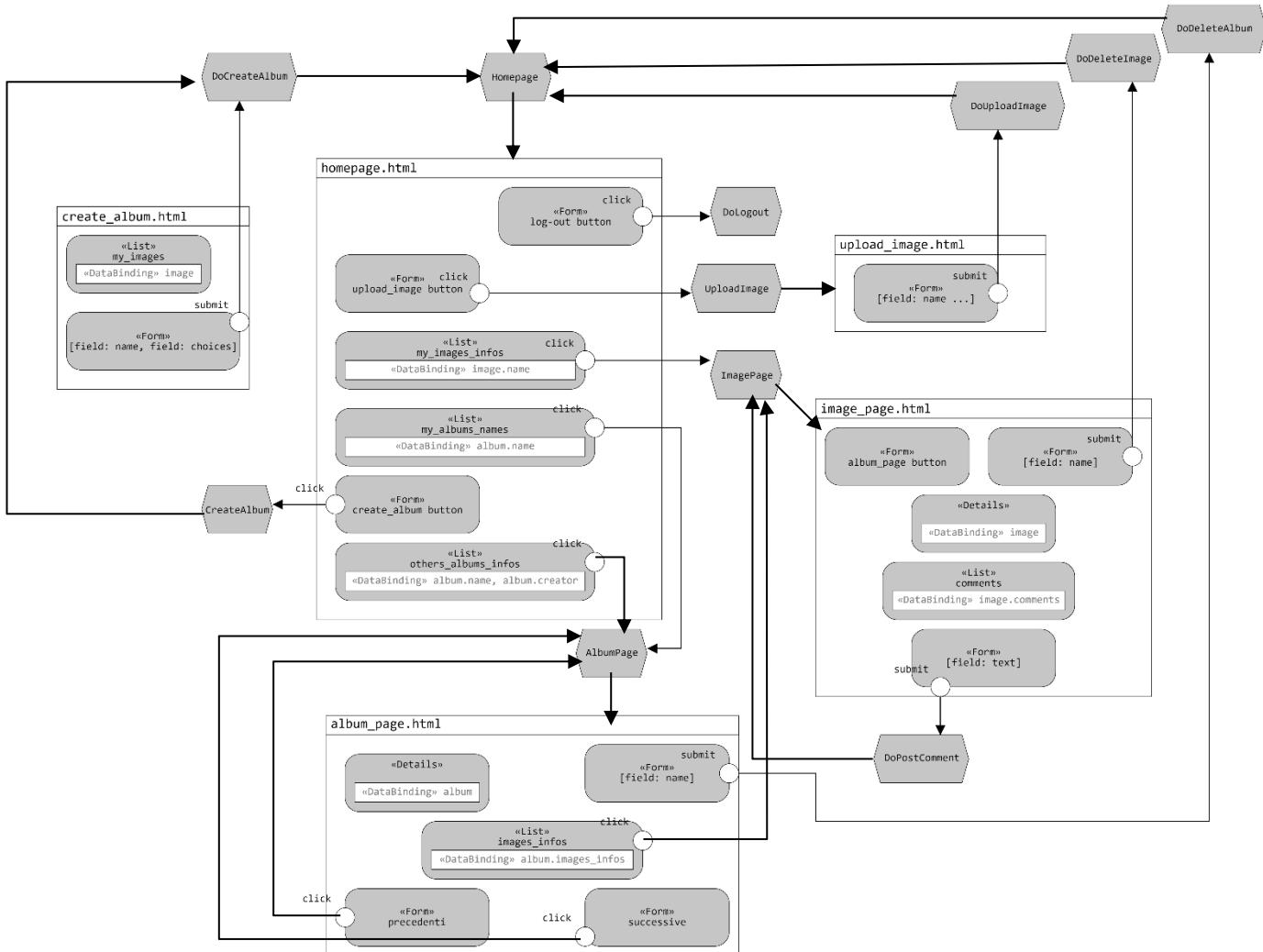
1.4 Application design (IFML diagrams)

1.4.1 Parte “pubblica”



miro

1.4.2 Parte “privata”



Abbiamo trascurato per semplicità i bottoni “torna all’homepage” e “log-out” (quest’ultimo è considerato solo nella HOME PAGE) benché presenti in ciascuna delle pagine della sezione “privata” del sito.

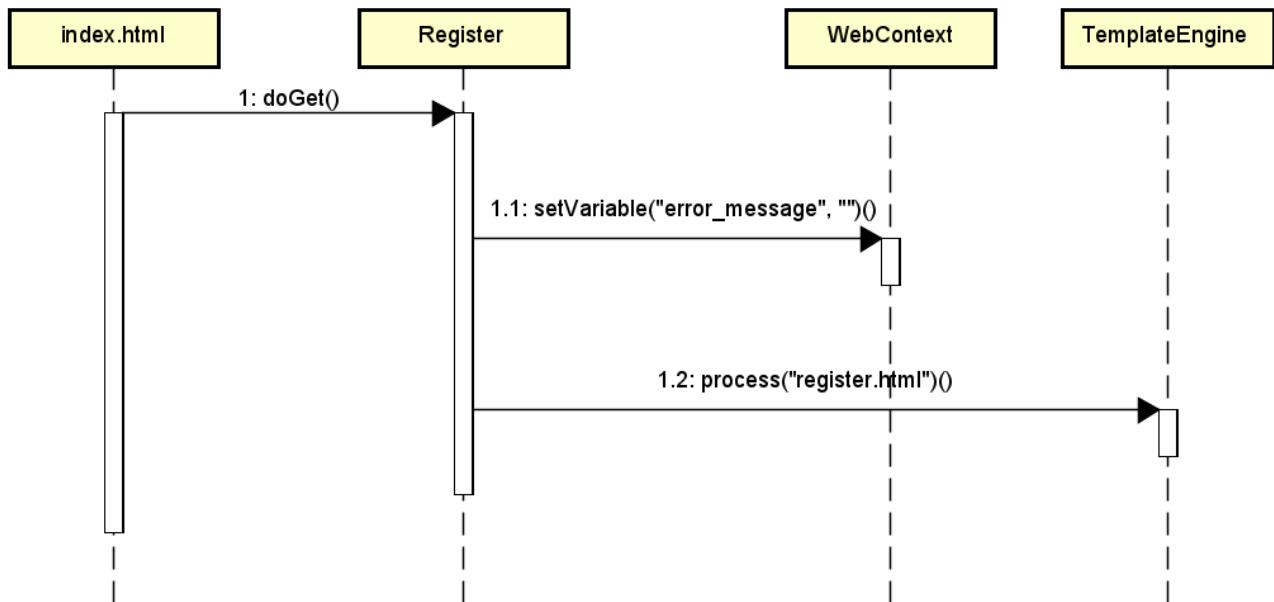
1.5 Sequence diagrams

1.5.1 Eventi parte “pubblica”

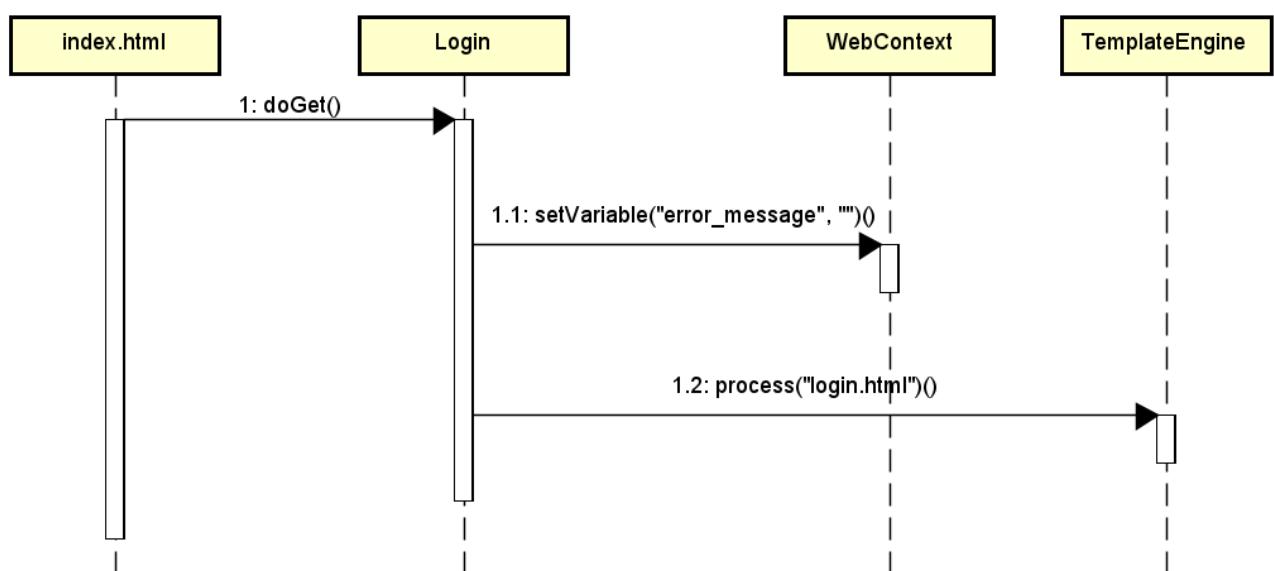
1.5.1.1 Accesso alle pagine di registrazione e login (non indicato nella lista di eventi estrapolati dalla specifica)

Implicitamente, il primo evento scatenato dall'utente è l'accesso alle pagine “register.html” e “login.html” (pagine web contenenti, rispettivamente, i form di registrazione e login):

- accesso a “register.html”:



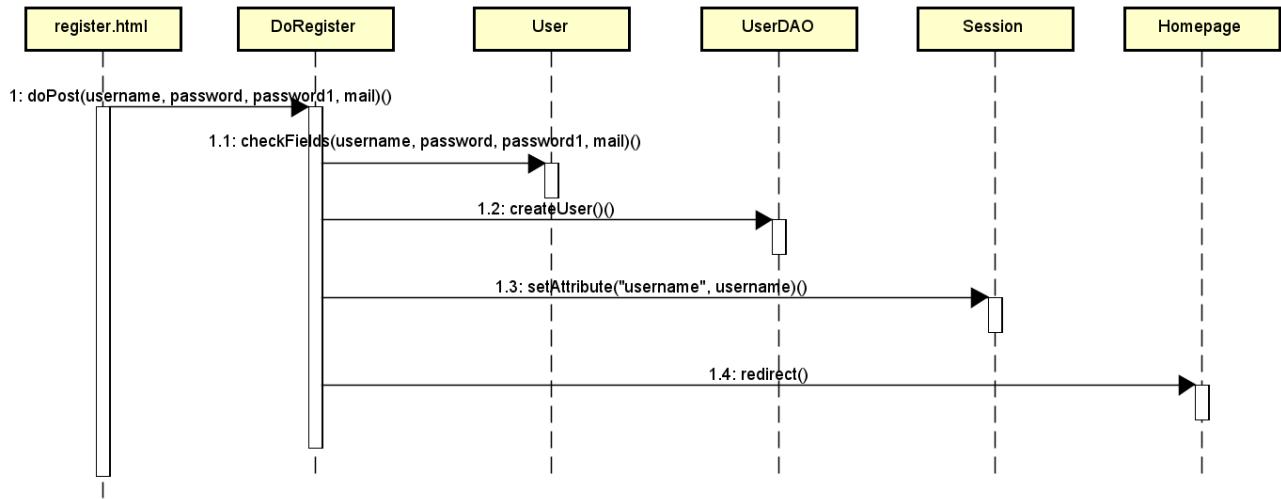
- accesso a “login.html”:



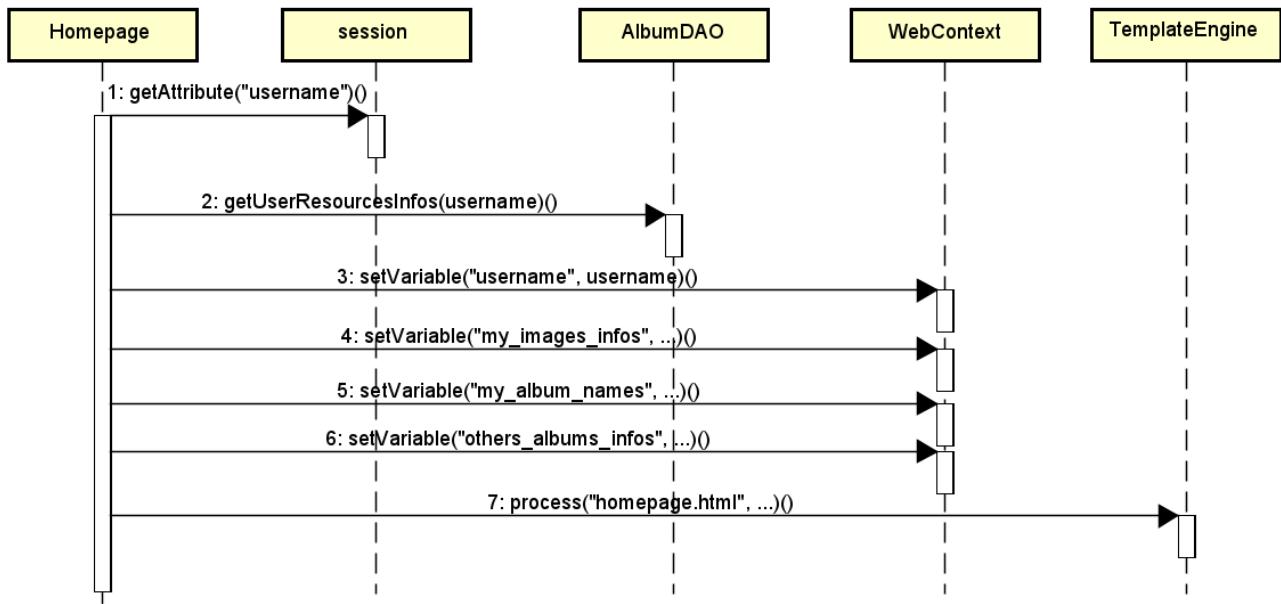
In caso di autenticazione già avvenuta (attributo di sessione “username” non “null”) le servlet “Register” e “Login” fanno redirezione direttamente su “Homepage” (l’utente non può accedere con credenziali diverse se prima non effettua il logout dalla sessione attuale).

1.5.1.2 Registration form submit

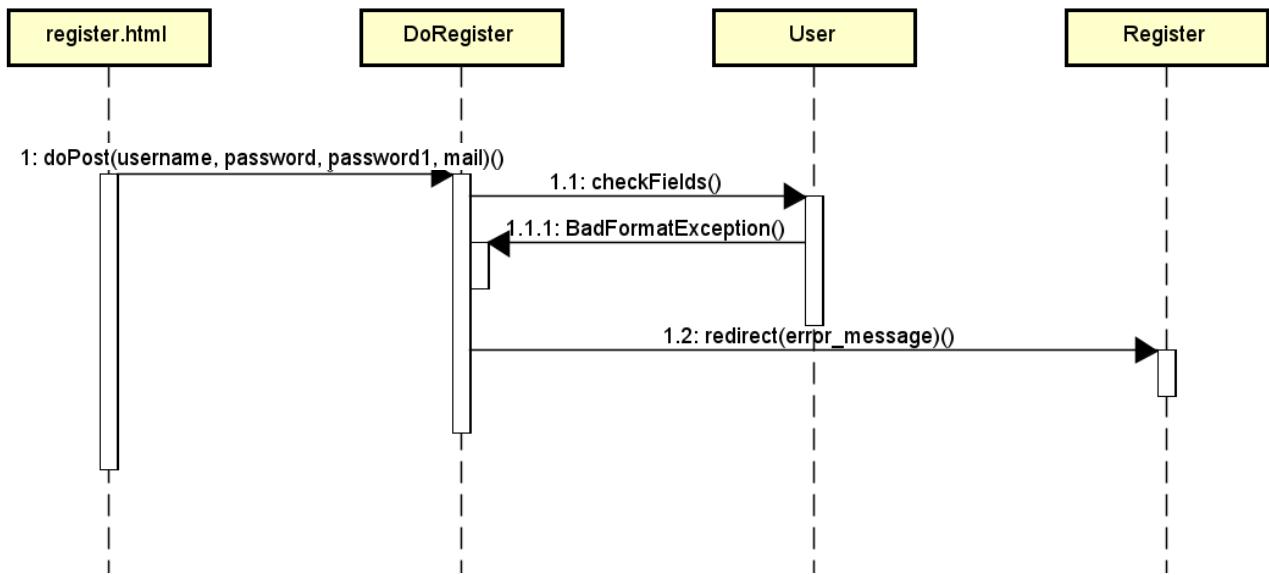
L'utente invia i dati per la registrazione di una nuova utenza senza che si verifichino errori di formato o nell'inserimento nella base di dati:



“Homepage” popola il template con tutte le informazioni da mostrare nella HOME PAGE:



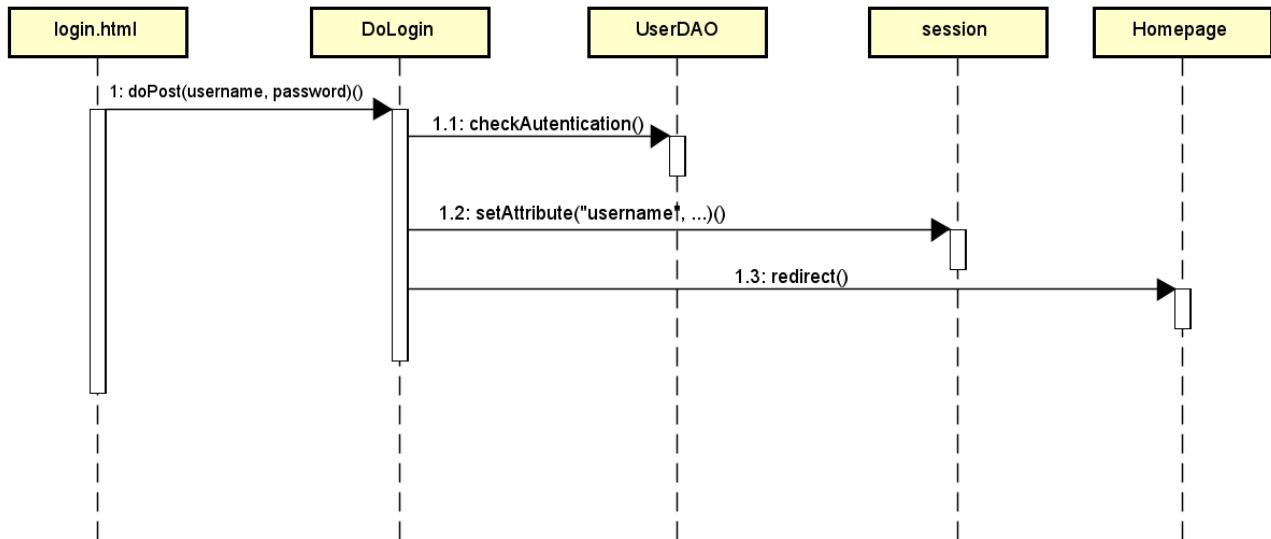
Se si verifica un errore a causa di invio di dati il cui formato non è valido (ad esempio, campo “username” vuoto):



Errori segnalati dal DBMS vengono notificati al client in maniera analoga, cambia solamente l'eccezione lanciata (“`DbErrorException`”).

1.5.1.3 Login form submit

Se l'autenticazione va a buon fine:

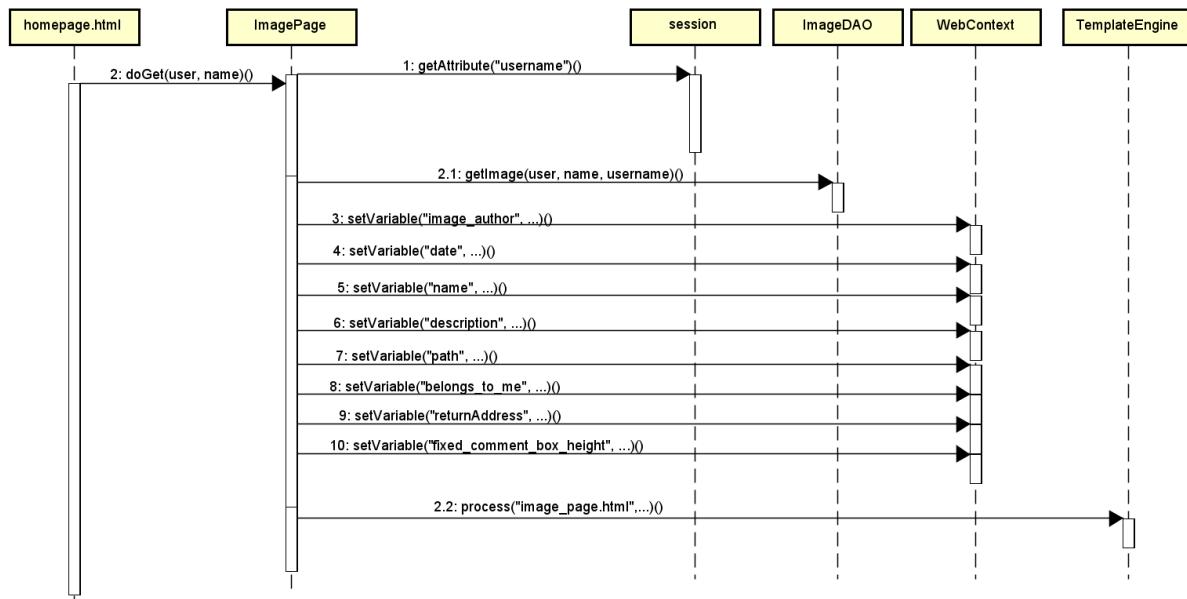


altrimenti, la sequenza è pressoché analoga a quella illustrata per gli errori nella registrazione (le classi “Beam” o, a seconda dell'errore, “DAO”) restituiscono un'eccezione “`BadFormatException`” o “`DbErrorException`”.

1.5.2 Eventi parte “privata”

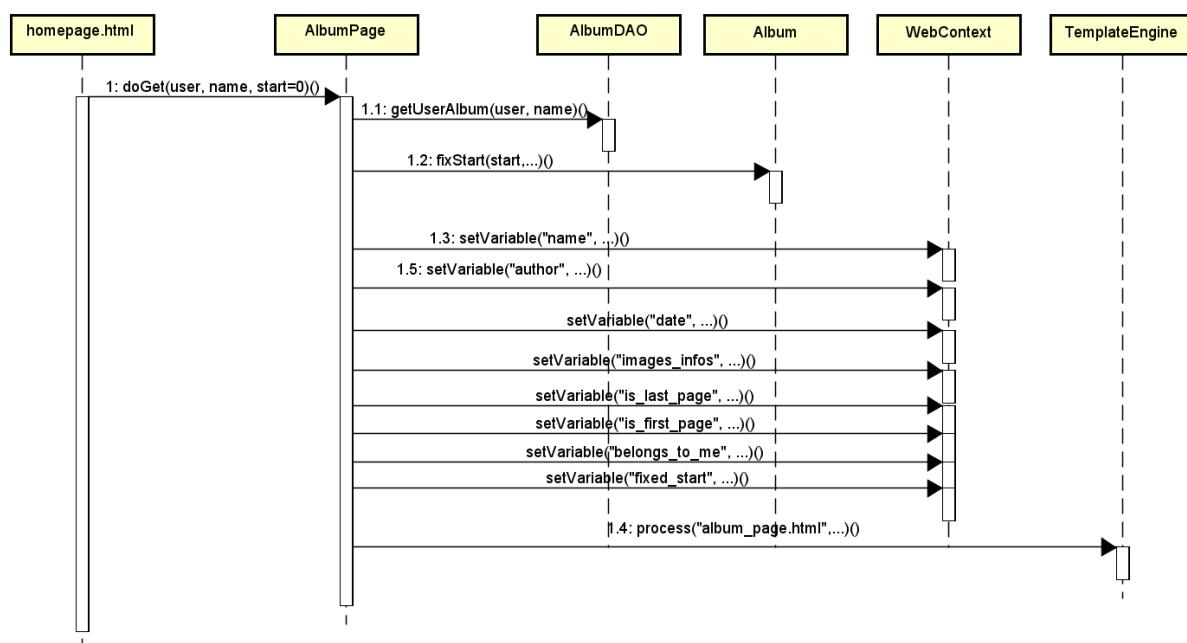
1.5.2.1 Click su miniatura immagine

Come mostrato nell'application design, è possibile visualizzare la miniatura di un'immagine sia dalla “HOME PAGE” che dalla “ALBUM PAGE”, abbiamo così due punti di accesso all'IMAGE PAGE. Tra i due c'è solo una minima differenza, nell'accesso da “HOME PAGE” non viene riempito il campo “returnAddress” (nella richiesta “GET”) mentre nell'accesso da “ALBUM PAGE” viene riempito con il nome dell'album visualizzato nell'ALBUM PAGE:



1.5.2.2 Click sul nome di un album

L'accesso all'ALBUM PAGE, come richiesto dalla specifica, mostra solo le prime cinque immagini dell'album:

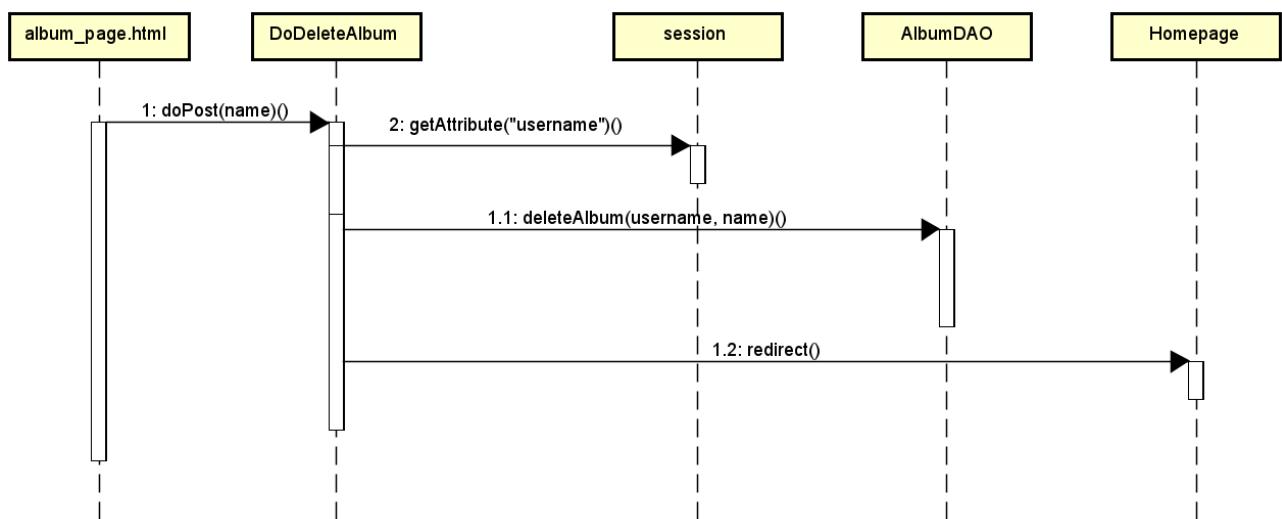


“start” è il parametro utilizzato per specificare l’indice della prima immagine da mostrare delle cinque (“start=0” richiede dunque le prime cinque immagini).

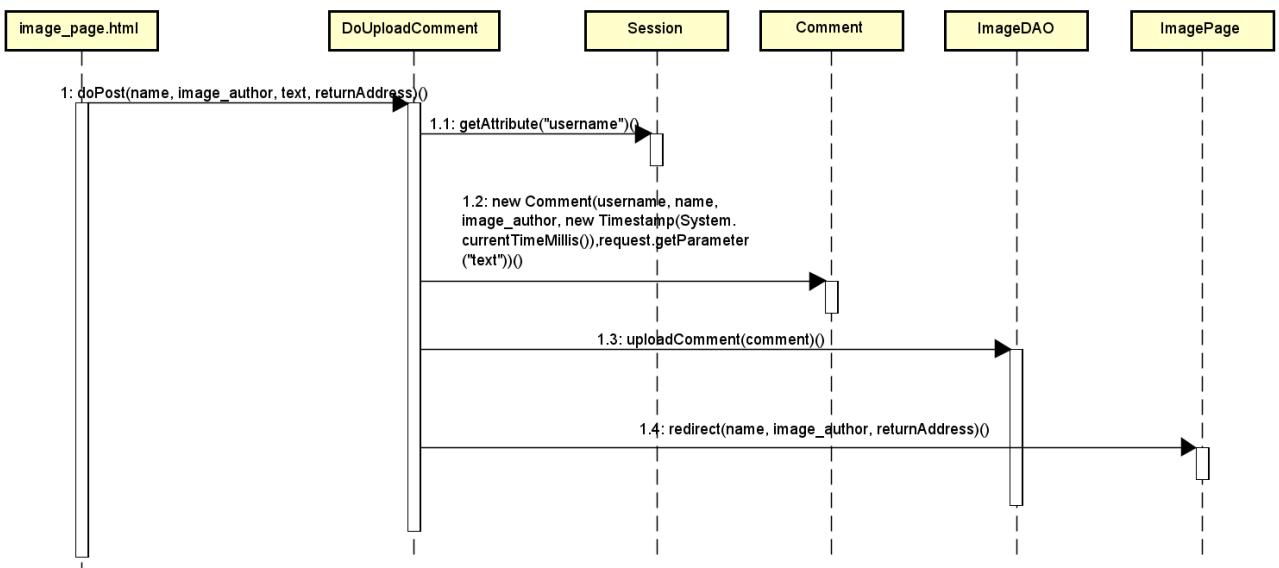
1.5.2.3 Click sui bottoni di navigazione dell’ALBUM PAGE

Il sequence diagram è praticamente identico a quello del paragrafo precedente, semplicemente si cambia il valore di “start” (“fixStart()” si occupa di correggere il valore).

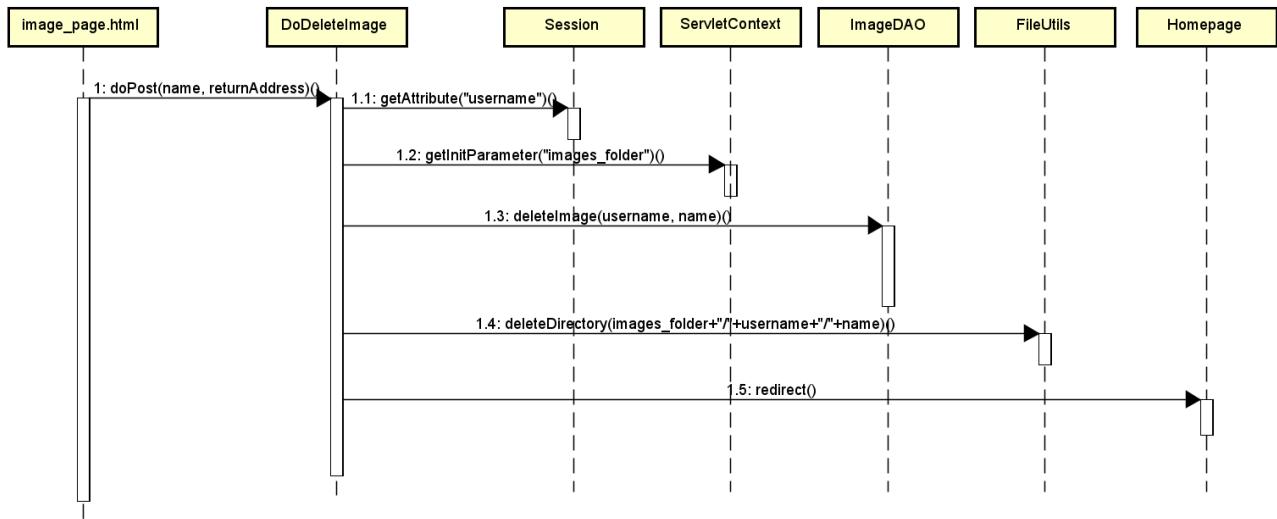
1.5.2.4 Click su “cancella album”



1.5.2.5 Click su “invia commento”

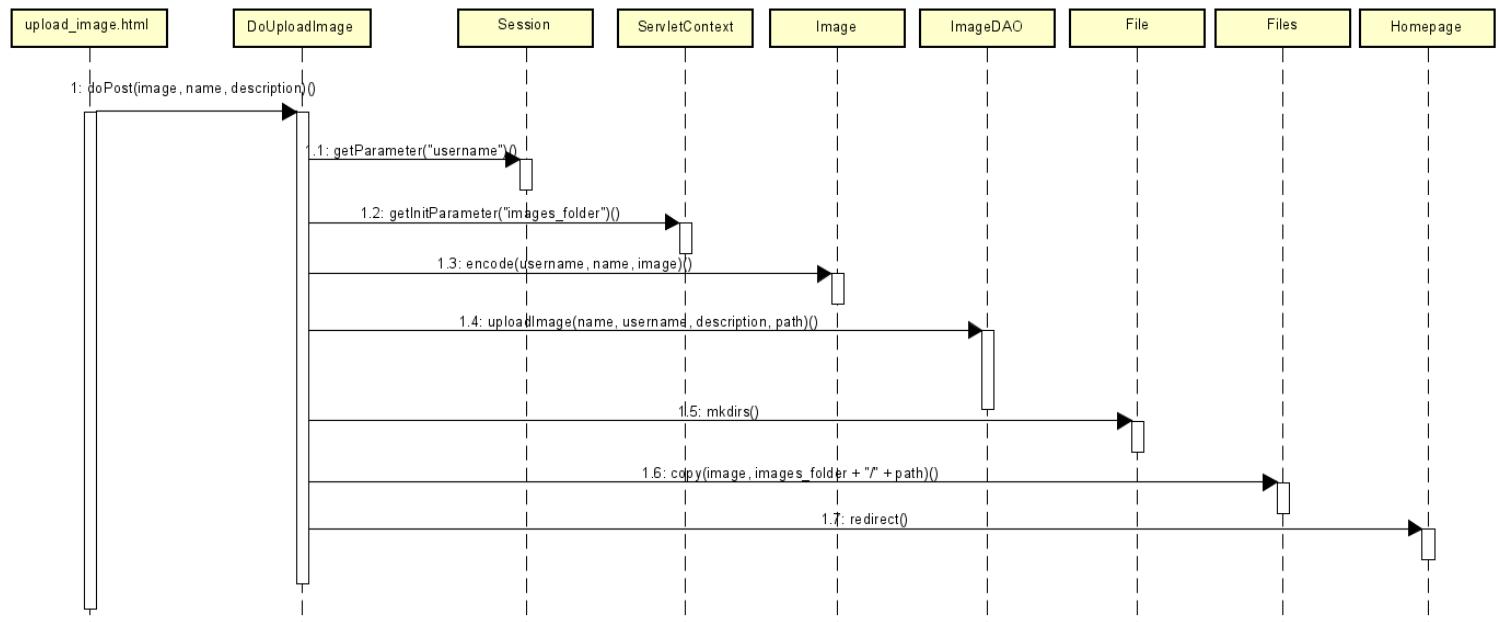


1.5.2.6 Click su “cancella immagine”



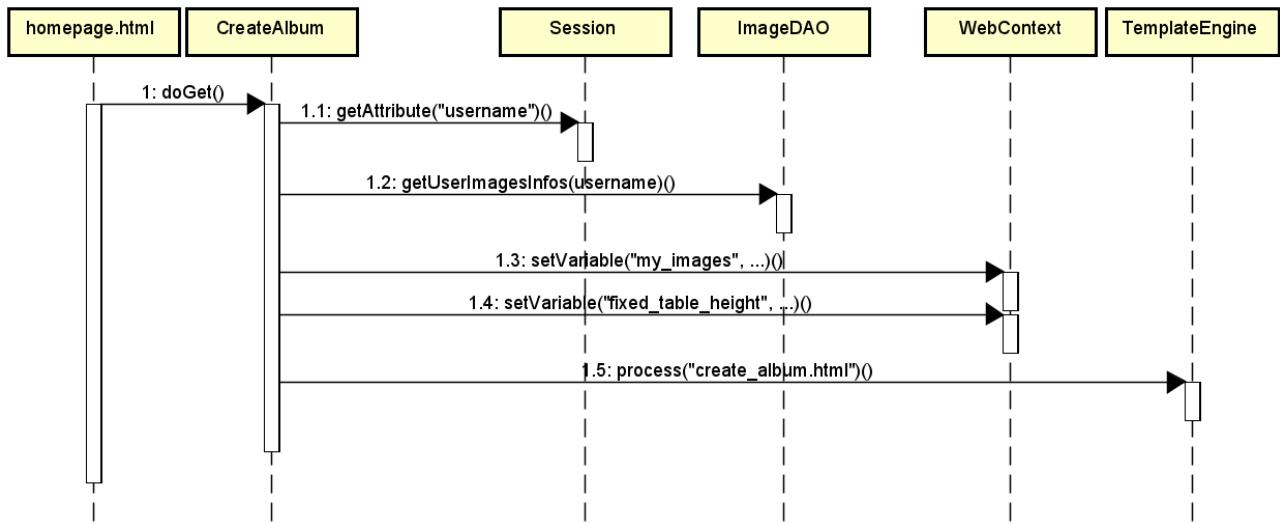
1.5.2.7 Click su “carica immagine”

L'accesso alla pagina per l'upload di un'immagine è, in buona sostanza, l'accesso a una pagina statica, la servlet “UploadImage” serve semplicemente a riempire il template con un eventuale messaggio d'errore. Una volta in “upload_image.html” si carica un'immagine:

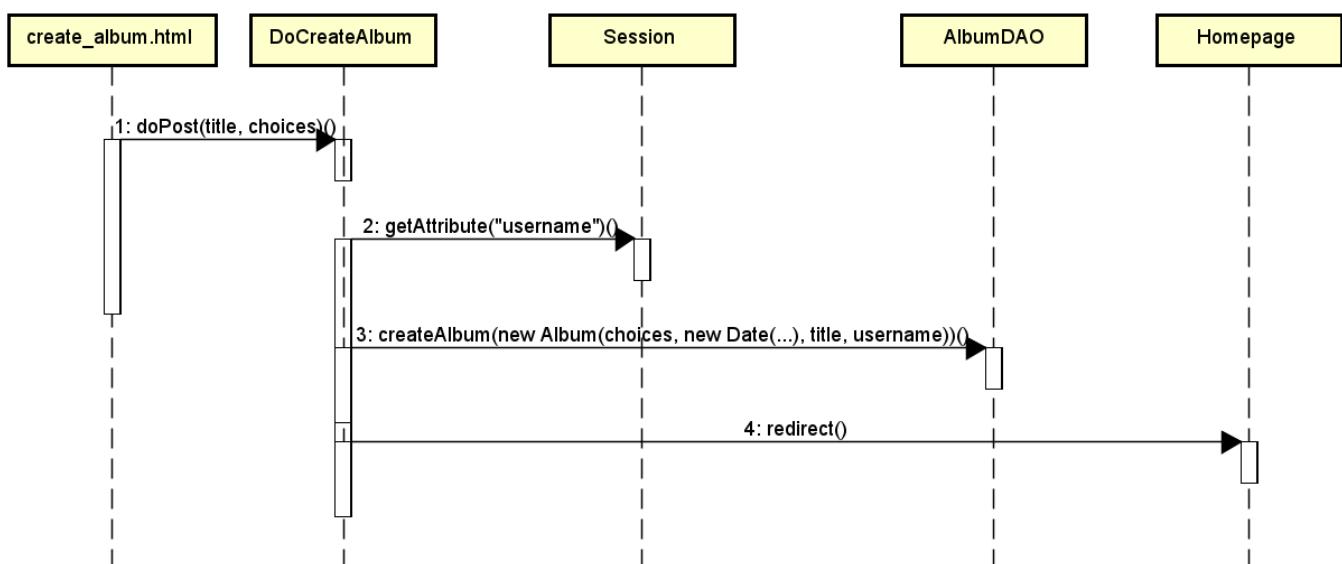


1.5.2.8 Click su “crea album”

Per accedere alla pagina di creazione di un album (“create_album.html”):

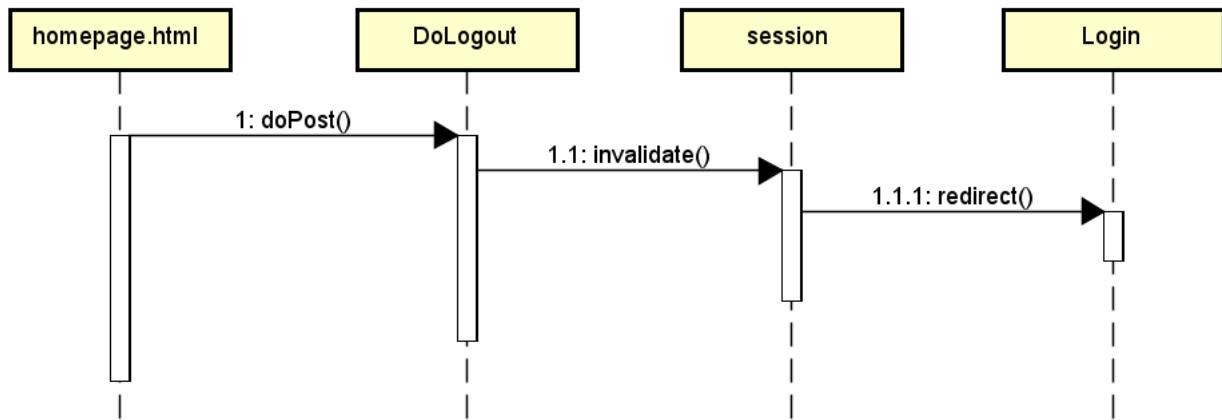


Dal click sul pulsante “crea album”:



1.5.2.9 Click su “log-out”

Il “log-out” è invocabile da qualunque pagina privata, per semplicità lo modelizziamo invocato dalla “HOME PAGE”:



1.5.2.10 Click su “torna alla homepage”

Il ritorno alla homepage si risolve esattamente come l'accesso all'home page dopo il login.

1.5.2.11 Click su “torna all'album”

Questo evento è pressoché identico al click su una miniatura di un'immagine, cambia solo che il parametro “returnAddress” è specificato nella richiesta “GET” inviata a “ImagePage”.

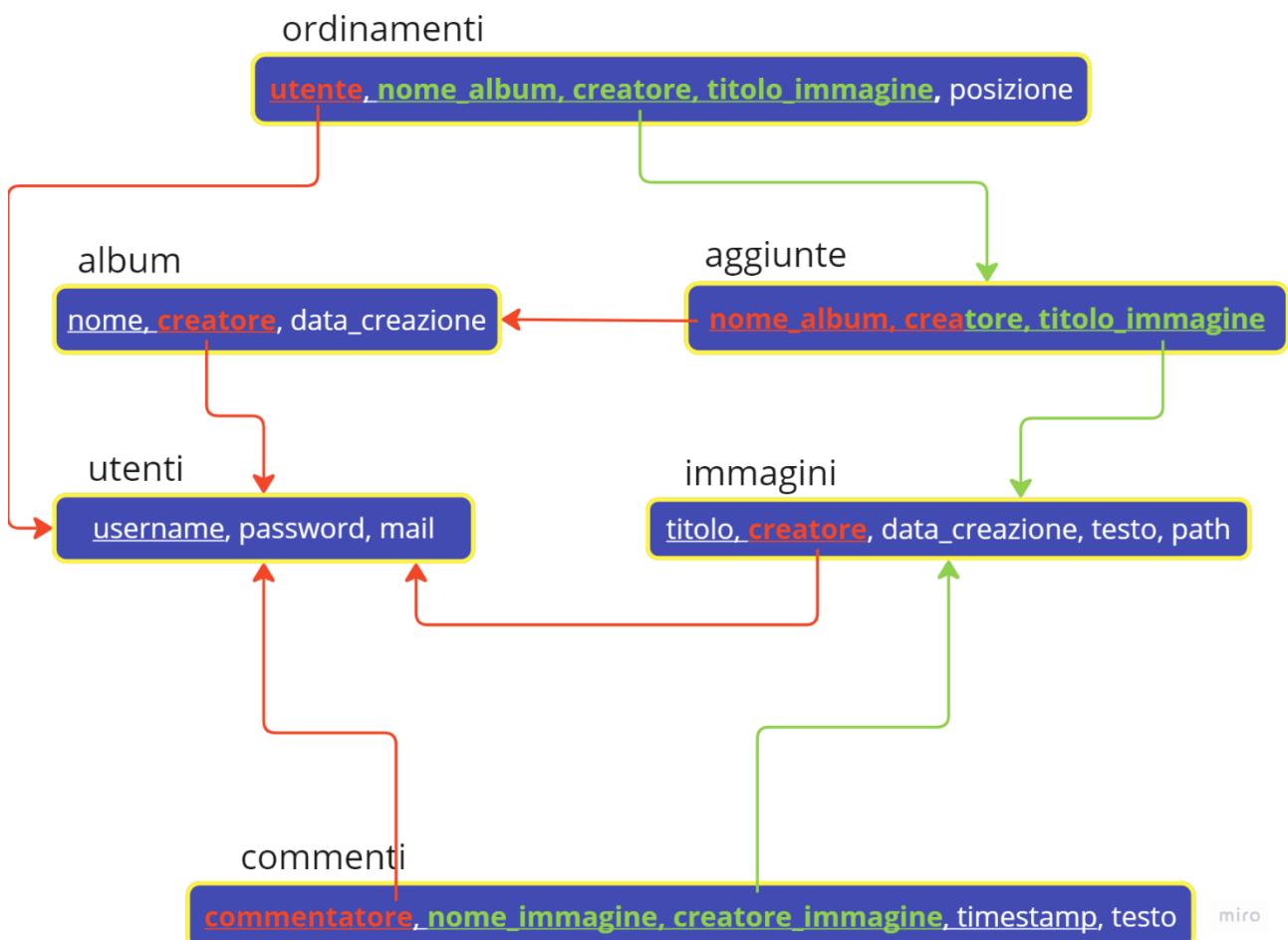
2 Documentazione versione “con JavaScript”

2.1 Ampliamento base di dati da “pure HTML”

Alcune delle richieste per la versione “con JavaScript” potrebbero contenere modifiche da apportare alla base di dati:

Si deve consentire all’utente di riordinare l’elenco delle immagini all’interno di un album con un criterio personalizzato diverso da quello di default (data decrescente). L’utente accede a un elenco dei titoli delle immagini ordinato secondo il criterio correntemente in uso: default o personalizzato. Trascina il titolo di un’immagine nell’elenco e lo colloca in una posizione diversa per realizzare l’ordinamento che desidera, senza invocare il server. Quando l’utente ha raggiunto l’ordinamento desiderato, usa un bottone “salva ordinamento”, per memorizzare la sequenza sul server.

La scelta che ci è sembrata più semplice da applicare alla base di dati è stata l’aggiunta di una tabella “ordinamenti” che tenesse le corrispondenze tra “utente – immagine in un album – posizione scelta per l’ordinamento”. Lo schema logico della base di dati diventa dunque:



Nel dettaglio, i domini degli attributi e i relativi vincoli:

Attributo	Dominio	Vincoli
utente	VARCHAR(50)	NOT NULL
nome_album	VARCHAR(50)	NOT NULL
creatore	VARCHAR(50)	NOT NULL
titolo_immagine	VARCHAR(50)	NOT NULL
posizione	INT	NOT NULL

Vincoli di tupla:

PRIMARY KEY(utente, nome_album, creatore, titolo_immagine)
FOREIGN KEY(utente) REFERENCES utenti(username) ON DELETE CASCADE ON UPDATE CASCADE
FOREIGN KEY(nome_album, creatore, titolo_immagine) REFERENCES aggiunte(nome_album, creatore, titolo_immagine)
ON DELETE CASCADE ON UPDATE CASCADE

2.2 Modifica Bean e DAO da “pure HTML”

L’adattamento delle classi Bean e DAO dalla versione “pure HTML” è pressoché minimo:

- modifiche sui bean:
 - tolti i metodi “fixStart()” e “getImages()” da “Album” : non avevano più utilità dal momento che la gestione del carosello nell’ALBUM PAGE è fatta lato-client;
 - aggiunta classe “FullAlbum” per rappresentare un album con al proprio interno le immagini sottoforma di “List<Image>”. Nella versione pure HTML era sufficiente avere le informazioni utili a reperire le immagini di un Album, mentre la specifica richiede che:

L’**evento di visualizzazione del blocco precedente/successivo d’immagini di un album** è gestito a lato client senza generare una richiesta al server. L’applicazione carica le informazioni necessarie per la visualizzazione di tutte le immagini di un album e dei relativi commenti mediante un’unica chiamata.

ovvero si rende necessario l’invio al client delle informazioni di un album e con esse di tutte le informazioni di tutte le immagini in esso contenute;

- modifiche sui DAO:
 - in “albumDAO”, è stato aggiunto il metodo “sortAlbum()” per l’inserimento nella base di dati di un ordinamento personalizzato e modificato opportunamente “getUserAlbum()” in modo che, in caso di ordinamento personalizzato, restituisca le immagini secondo quest’ultimo (sono stati anche aggiunti metodi che “atomicizzano” l’esecuzione in sequenza di altri metodi delle DAO, in modo da garantire che ciascuna servlet chiama uno e uno solo metodo delle DAO, su cui sono eventualmente applicate le tecniche di gestione delle transazioni);
 - controlli di correttezza sull’ordinamento scelto.

Omettiamo così i class diagram dei Bean e delle DAO, in quanto aventi minime variazioni rispetto a quelli illustrati per la versione pure HTML.

2.3 Application requirements analysis

2.3.1 Analisi della consegna

Rispetto alla specifica pure HTML, abbiamo bisogno di ulteriori “oggetti grafici”, quali:

- form per l’invio di un ordinamento personalizzato inseribile mediante “drag & drop”;
- finestra modale per mostrare tutte le informazioni relative a un’immagine e per inviare commenti a quell’immagine.

Manteniamo in oltre tutte le aggiunte e le scelte grafiche adottate nella versione pure HTML (ad esempio, l’utilizzo di una menubar).

Abbiamo così “4” nuovi eventi e “4” corrispondenti nuove azioni:

- “mouseover” su miniatura ⇒ presentazione dati dell’immagine e commenti (e conseguente “mouseleave” per nascondere la finestra modale);
- “click” sul bottone per inviare un commento nella finestra modale ⇒ invio del commento inserito;
- “dragover” su immagini nel form per l’ordinamento personalizzato ⇒ swap tra immagini (azione “lato-client”);
- click su “salva ordinamento” ⇒ invio dell’ordinamento scelto.

Aggiungiamo a questi anche l’evento di chiusura dell’eventuale messaggio di errore (“click” su un qualunque punto esterno al riquadro contenente il messaggio).

Visivamente l’utente avrà l’impressione di navigare tra differenti pagine web analogamente alla versione “pure HTML” ma anziché utilizzare delle “vere e proprie” pagine web esisterà un’unica pagina web contenente tutto l’HTML “statico” del sito i cui segmenti verranno mostrati/nascosti e popolati col contenuto dinamico con gli script in JavaScript.

2.3.2 Definizione dei controller e degli oggetti JS

Benché non si utilizzino più i template Thymeleaf, ha ancora senso distinguere le servlet in “pages creators” e “actions performers” (la servlet “ImagesHandler” resta invariata):

- “pages creators” : restano gli stessi di prima, si tolgoano semplicemente quelli per le pagine pubbliche (gli errori vengono scritti nella pagina web dagli script JavaScript) e si aggiunge “SortAlbum” (anziché mettere le informazioni nei template le inviano al client come JSON);
- “actions performers” : restano gli stessi di prima, con l’aggiunta di “DoSortAlbum” .

Piccolo appunto: non esiste più il concetto di “redirezione”, tutta la gestione del flusso delle chiamate è stabilita lato client).

Le pagine web non sono dei template ma del “semplice” HTML che viene modificato dagli script:

- index.html (mai modificata dagli script);
- register.html (modificata solo per la visualizzazione di un messaggio d’errore);
- login.html (modificata solo per la visualizzazione di un messaggio d’errore);
- home.html.

“Restano gli stessi di prima” a livello di “ruolo”, la loro funzione viene limitata alle sole operazioni che è necessario effettuare lato-server:

- le “pages creators” si limitano a inviare il contenuto dinamico in formato JSON;
- le “actions performers” si limitano a effettuare le operazioni di aggiornamento della base di dati senza effettuare più le redirezioni alle “pages creators” relative alle pagine web da visualizzare in seguito all’azione.

Per la parte pubblica abbiamo due script JavaScript distinti (uno per registrazione e l’altro per login) che si occupano di gestire i dati forniti dall’utente ed inviarli al server come richieste asincrone, per la parte privata, lo script “homePageRunner.js” è stato organizzato con la seguente modalità (si è cercato di rendere il codice il più modulare e di facile lettura possibile, seguendo un approccio che ricorda l’object-orientation).

Gli obiettivi che si è cercato di raggiungere con le scelte di design fatte sono:

- riutilizzo del codice (ad esempio, esiste un unico oggetto per la finestra modale o per i caroselli di immagini);
- leggibilità;
- definizione organizzata dei listener (i listeners non sono definiti in un unico metodo ma, come vedremo, tutti i listener vengono aggiunti e rimossi all’interno di oggetti particolari detti “listeners directors”).

All'interno della IIFE (non vi è utilizzo di variabili globali o di funzioni definite globalmente ad eccezione della funzione di utilità per l'invio delle richieste asincrone), lo script parte con l'invocazione del metodo “start” dell'oggetto “pageOrchestrator”. Potremmo suddividere gli “oggetti” JavaScript definiti in:

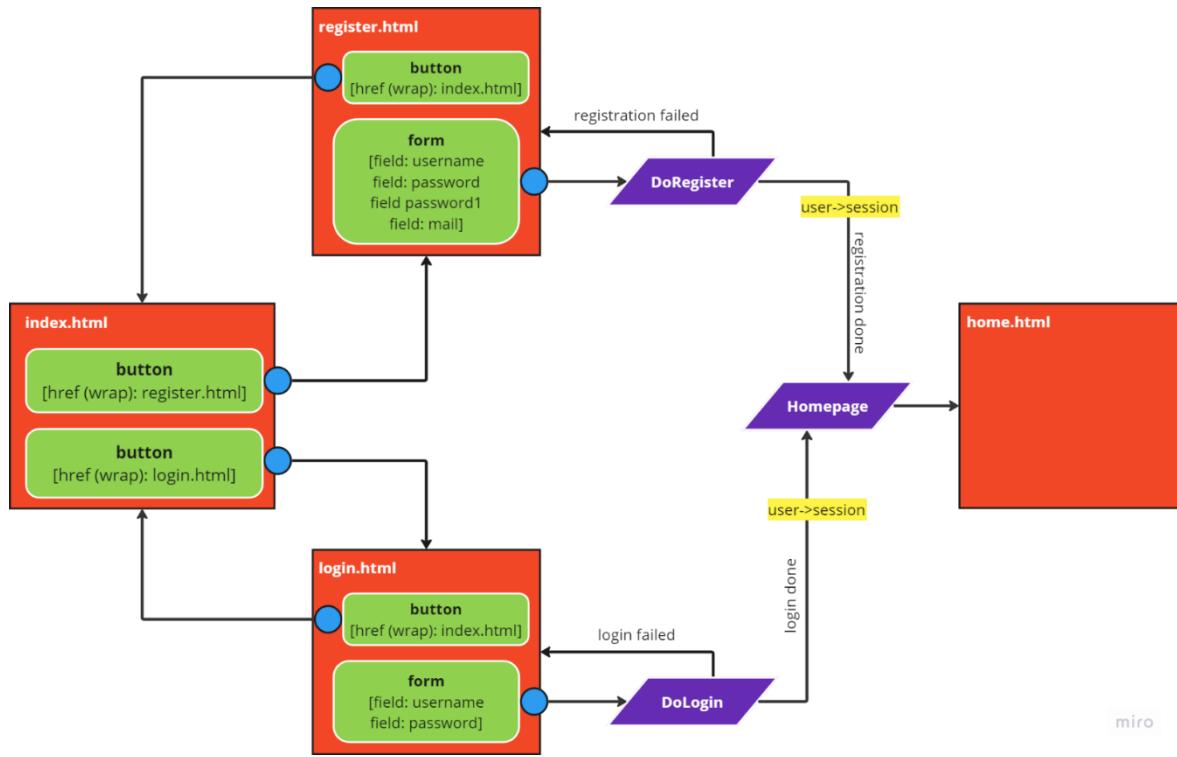
- “pageOrchestrator”: entità centralizzata che racchiude l’istanziazione di tutti gli altri;
- “view objects”: le entità atte a rappresentare e gestire gli oggetti grafici;
- “listeners directors”: entità che definiscono al loro interno tutti i listener per gli oggetti grafici (ogni “view object” ha il proprio).

Spieghiamo qui in breve la struttura di ciascuna tipologia (le meccaniche meno intuitive sono spiegate brevemente nella sezione dedicata ai sequence diagrams):

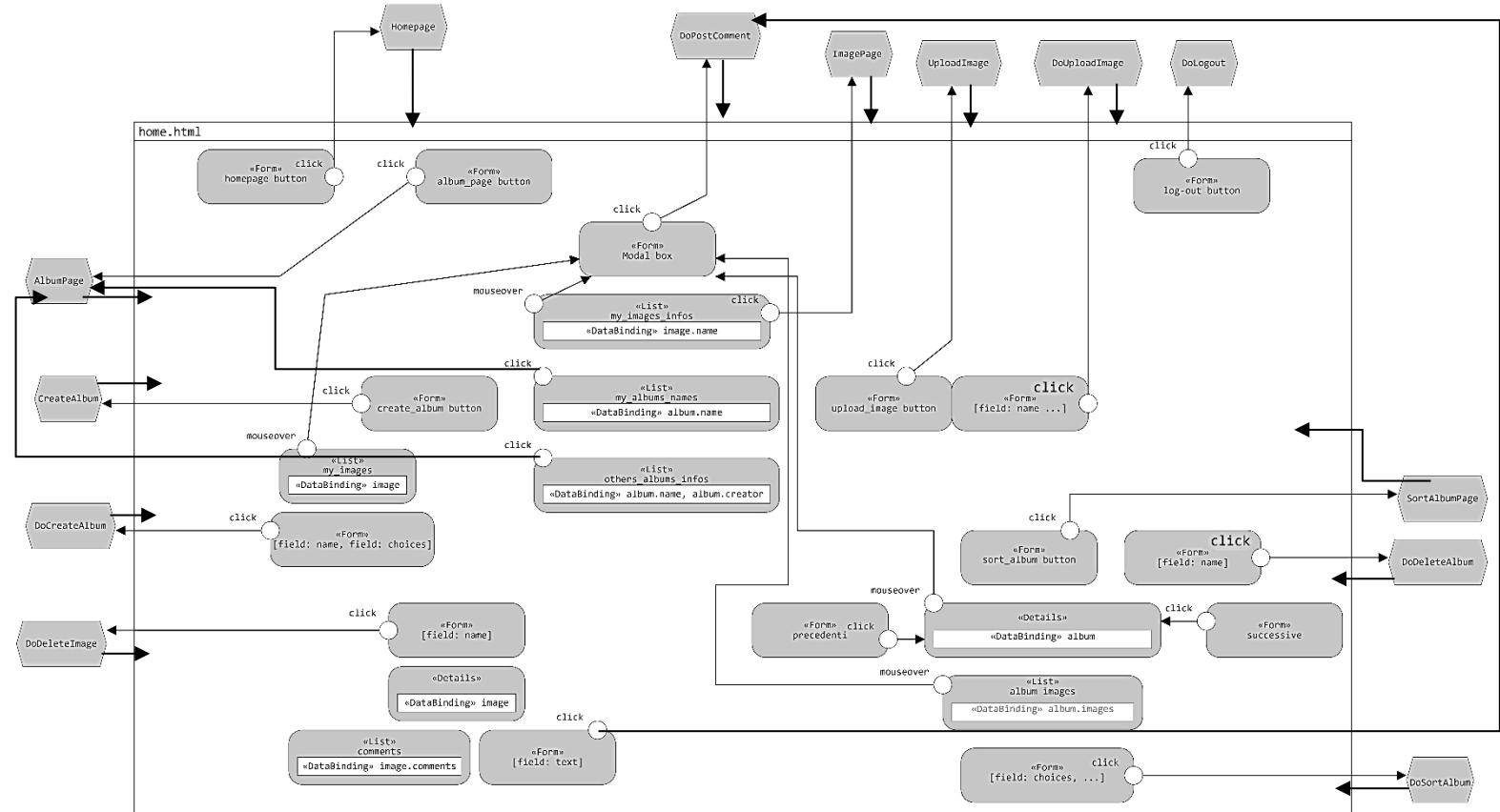
- “PageOrchestrator” : contiene i “listeners directors” (accessibili all'esterno del PageOrchestrator) e i metodi per nascondere/mostrare i “view objects”;
- “view objects” : a parte qualche sporadica eccezione tutti loro possiedono solo tre “metodi”:
 - “show” : richiede al server le informazioni utili al riempimento dell'oggetto grafico;
 - “update” : riempie l'oggetto grafico;
 - “hide” : nasconde l'oggetto grafico;
- “listeners directors” : possiamo vederli come suddivisi i tre parti:
 - “metodi” “pubblici” (nel senso che gli altri “oggetti” possono accedervi) per la rimozione e aggiunta dei listeners;
 - “metodi” “privati” (detti “shape”) che costituiscono i listeners “veri e propri”. L’idea è stata quella di non esporre direttamente i listeners, questi ultimi vengono “scelti” e modificati solamente dai “listeners director”. Non esiste listener non definito e aggiunto da un “listeners director” (l’unico che fa eccezione è “hide” del view object per il paragrafo d’errore, per rimuovere dalla visualizzazione il paragrafo di errore);
 - eventuali altri metodi pubblici con cui modificare gli “attributi” “privati” (intesi come variabili definite localmente). Sono particolarmente utili dato che, eccezione fatta per i directors della finestra modale, vengono tutti “istanziati” una sola volta nel pageOrchestrator.

2.4 Application design (IFML diagrams)

2.4.1 Parte “pubblica”



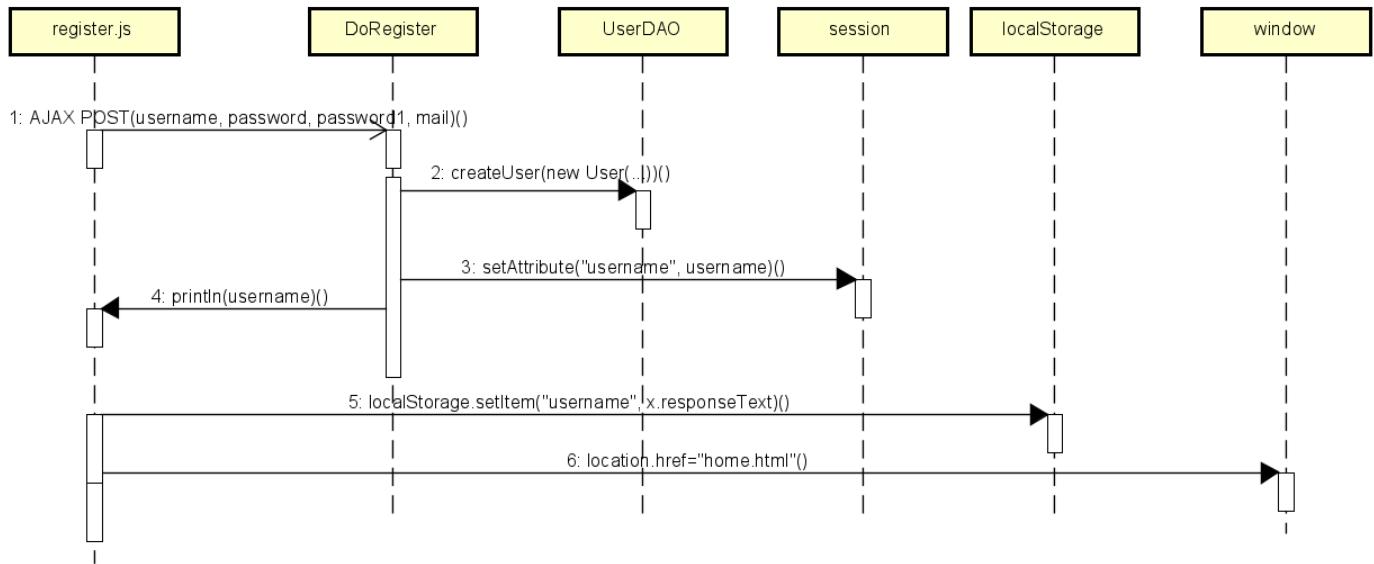
2.4.2 Parte “privata”



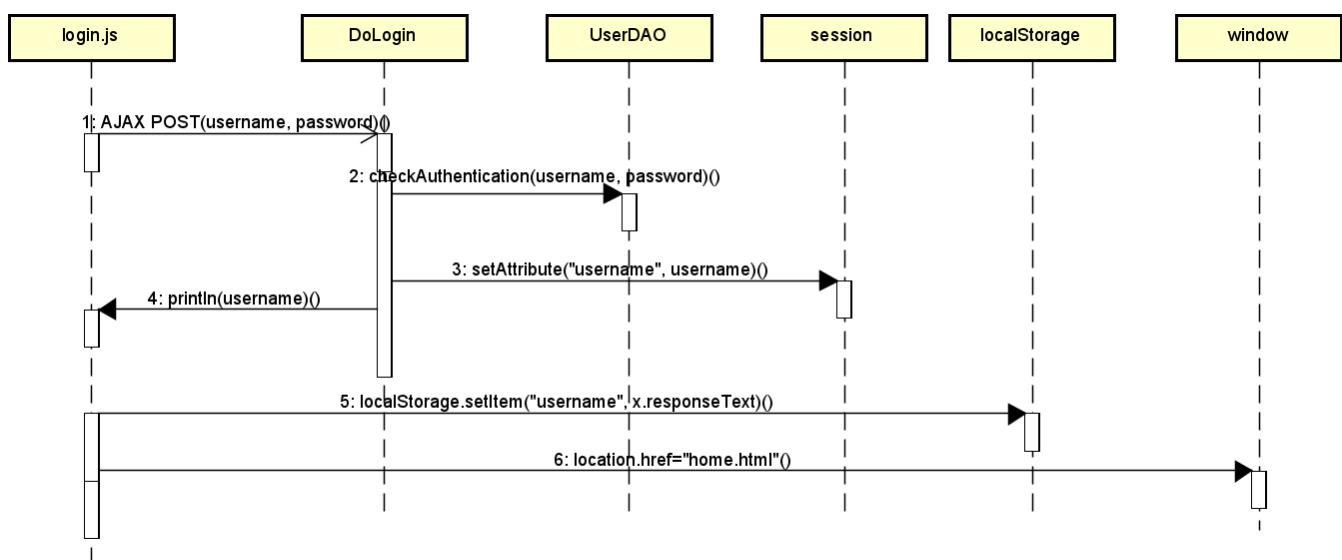
2.5 Sequence diagrams

2.5.1 Eventi parte “pubblica”

2.5.1.1 Registration form submit



2.5.1.2 Login form submit

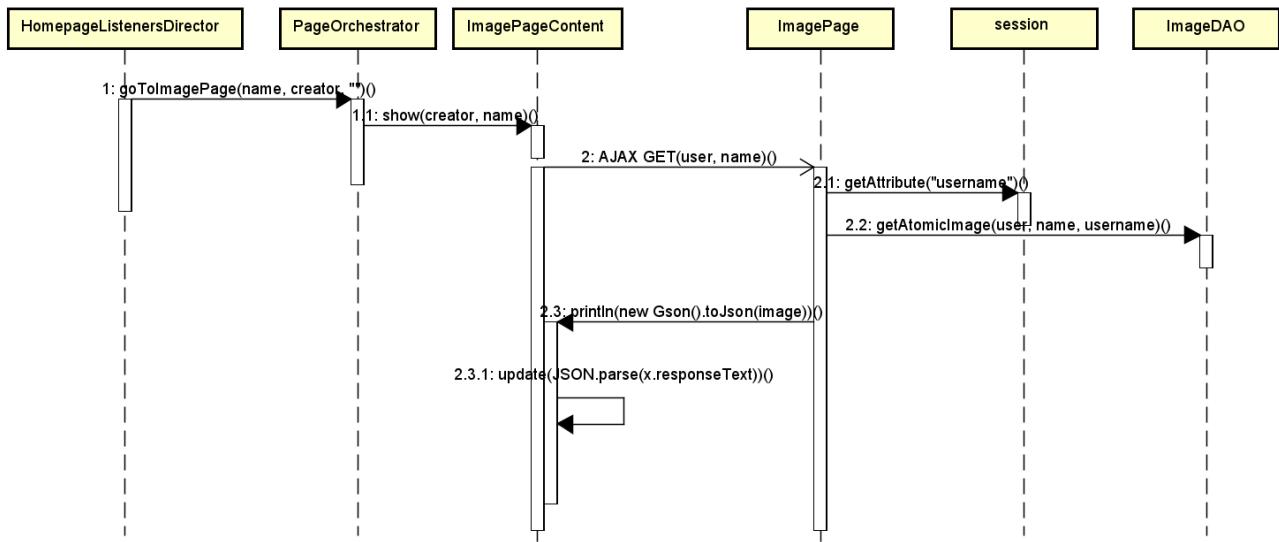


2.5.1.3 Click per chiusura del paragrafo di errore

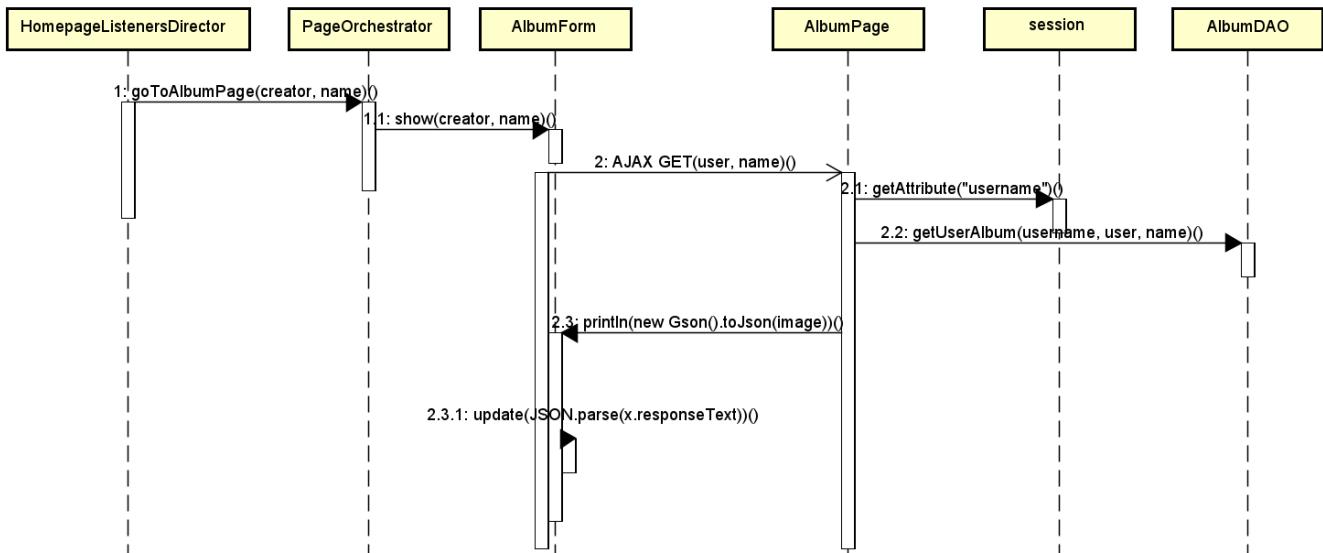
Qualunque errore (sia lato-client che lato-server) viene notificato al client con la visualizzazione di un paragrafo di errore (si invoca il suo metodo “update”). Lo si può chiudere semplicemente cliccando in un qualsiasi punto dello schermo su cui non è presente il paragrafo di errore (si invoca il suo “metodo” “hide”). Ne il riempimento ne lo “svuotamento” del paragrafo di errore coinvolge il server, viene completamente gestito lato-client (il server invia solamente JSON).

2.5.2 Eventi parte “privata”

2.5.2.1 Click su miniatura immagine

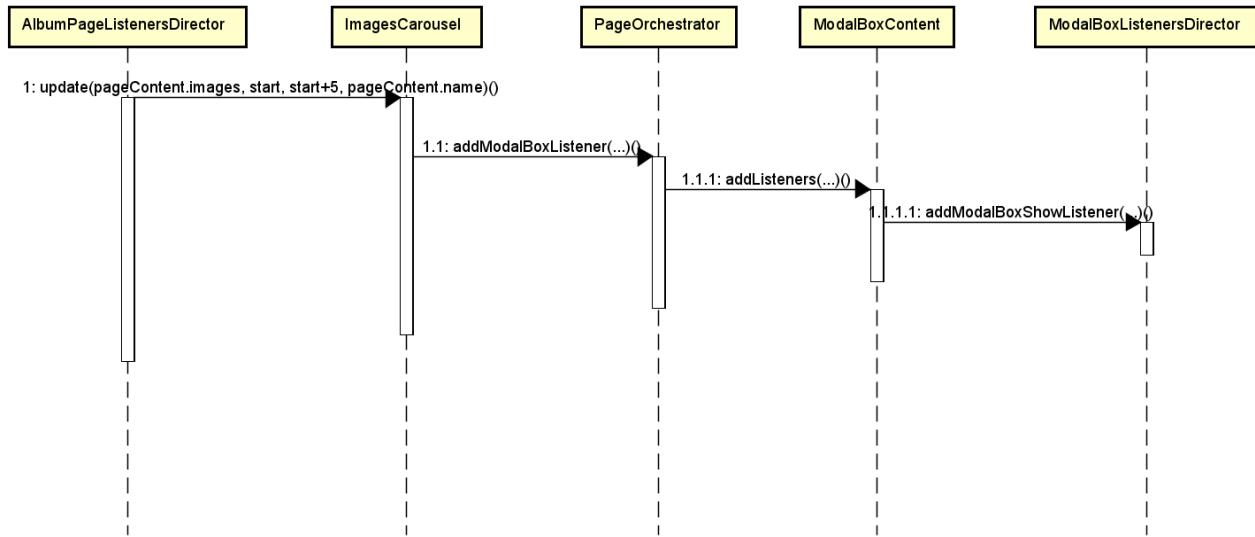


2.5.2.2 Click sul nome di un album



2.5.2.3 Click sui bottoni di navigazione dell'ALBUM PAGE

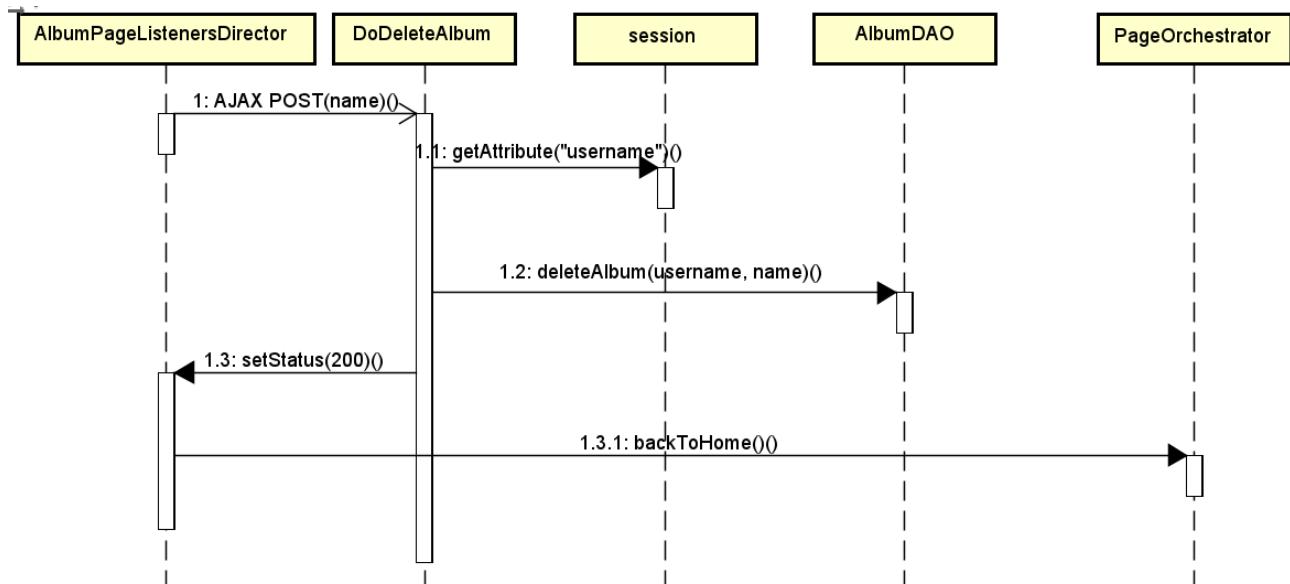
L'evento di aggiornamento del carosello delle immagine è triviale, semplicemente si invoca “update” (non “show”, poiché si tratta di informazioni già in possesso dal client). Tuttavia, è necessario aggiornare i listener relativi alle modal box (comunque anche questo senza fare richieste al server, poiché abbiamo già tutte le informazioni):



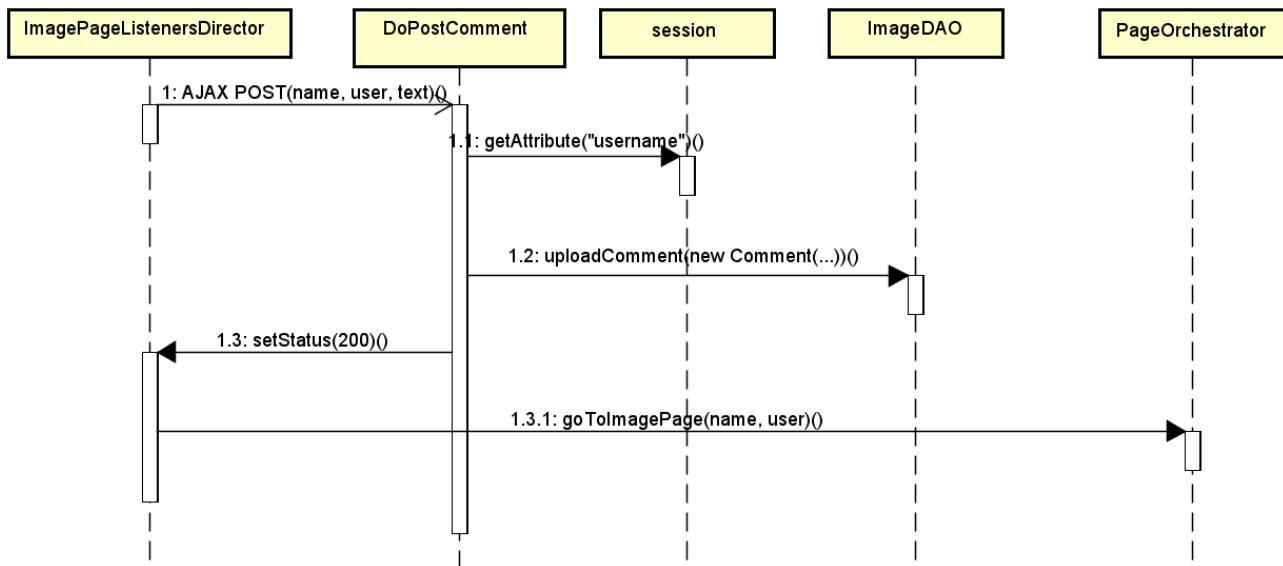
Tra il click su “SUCCESSIVE” e quello su “PRECEDENTI”, cambia solo il sommare/sottrarre “5” a “start” prima di chiamare “update”.

In maniera analoga, questo aggiornamento dei listener relativi alle modal box viene fatto in tutte le “pagine” che mostrano miniiture (lo omettiamo negli altri sequence diagram), il metodo “update” del view object che contiene le miniiture invoca “addModalBoxListener” per aggiornare i listener relativi alle modal box coi nuovi contenuti da presentare all'evento “mouseover”.

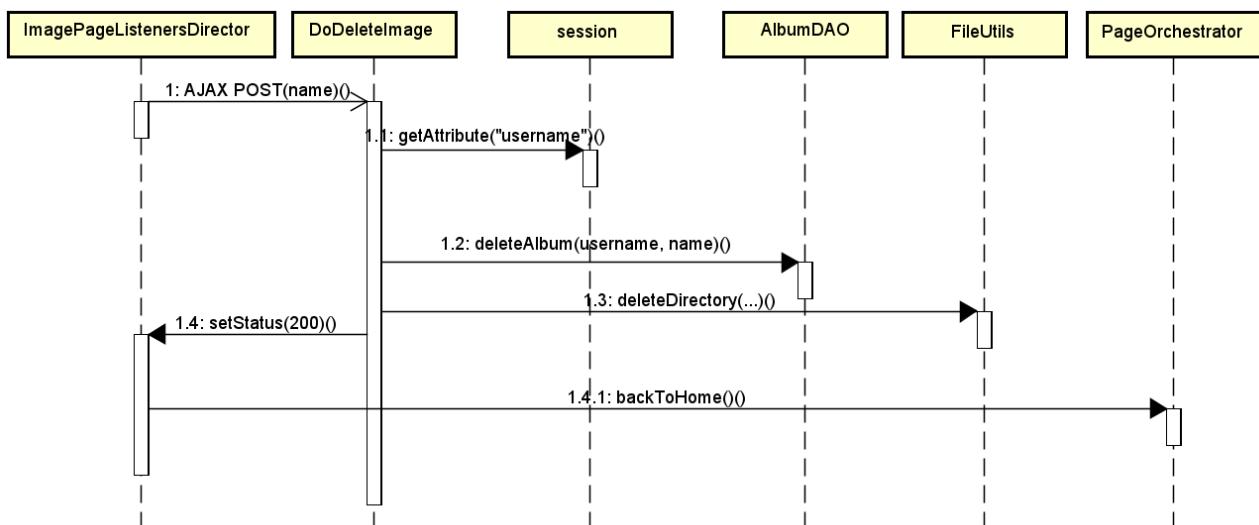
2.5.2.4 Click su “cancella album”



2.5.2.5 Click su “invia commento” (non dalla finestra modale)



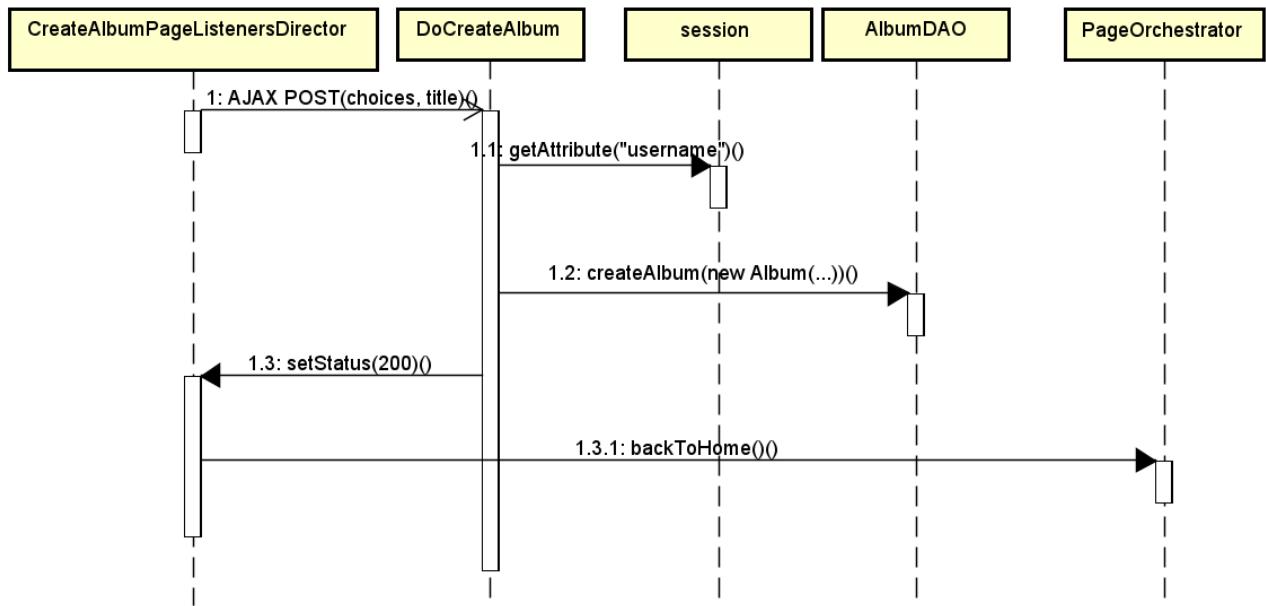
2.5.2.6 Click su “cancella immagine”



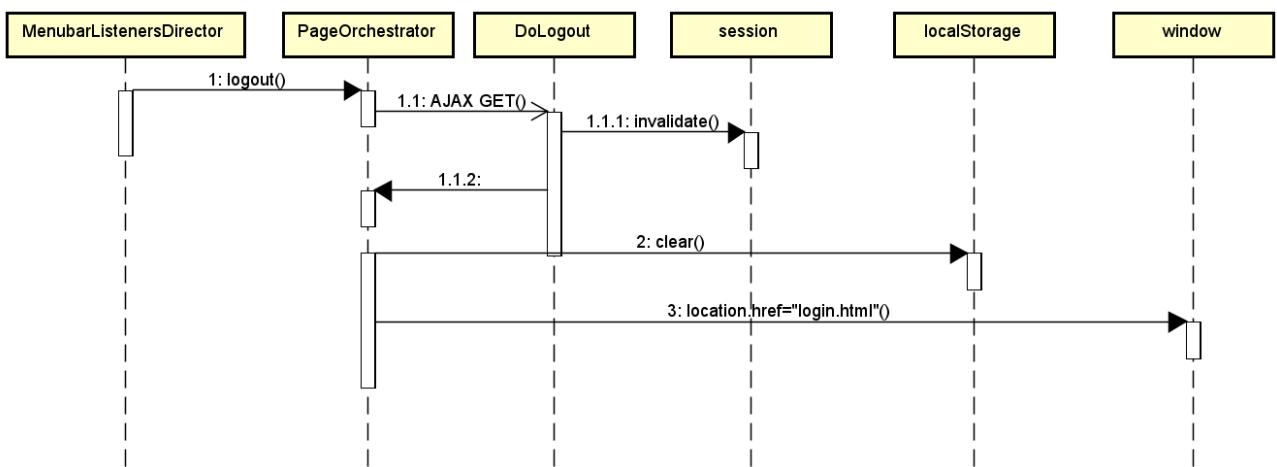
2.5.2.7 Click su “carica immagine”

Non cambia praticamente nulla dal caso “pure HTML”, semplicemente si invia una richiesta AJAX (“POST”) e la funzione di call-back chiama “backToHome()” di “pageOrchestrator.”

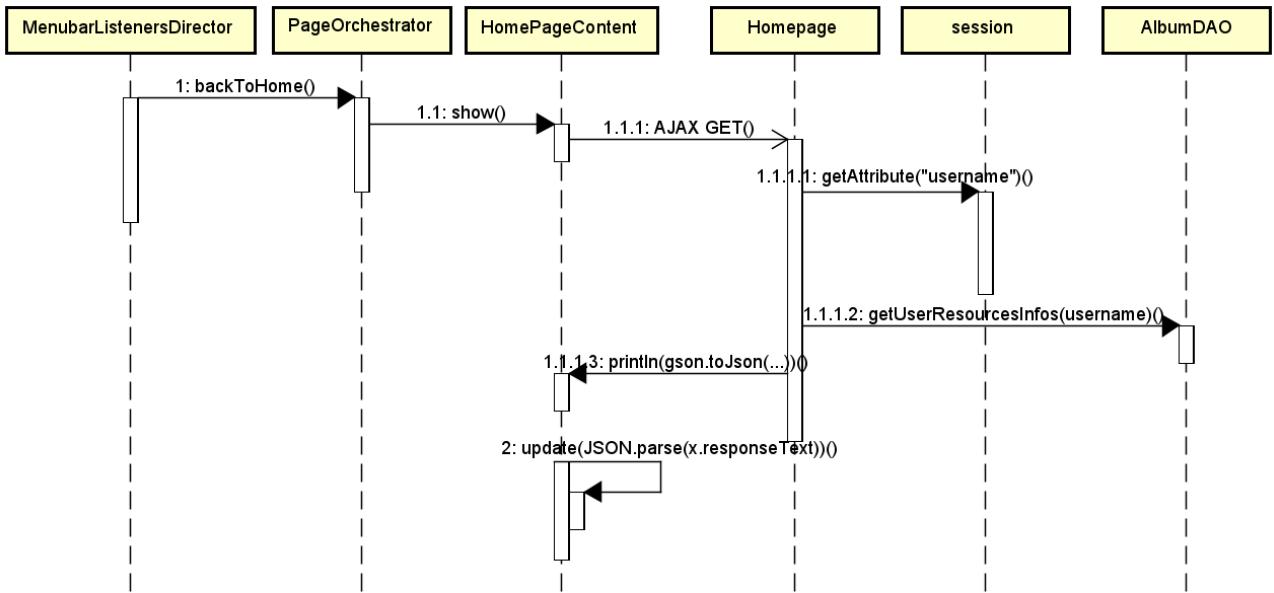
2.5.2.8 Click su “crea album”



2.5.2.9 Click su “log-out”



2.5.2.10 Click su “torna alla homepage”



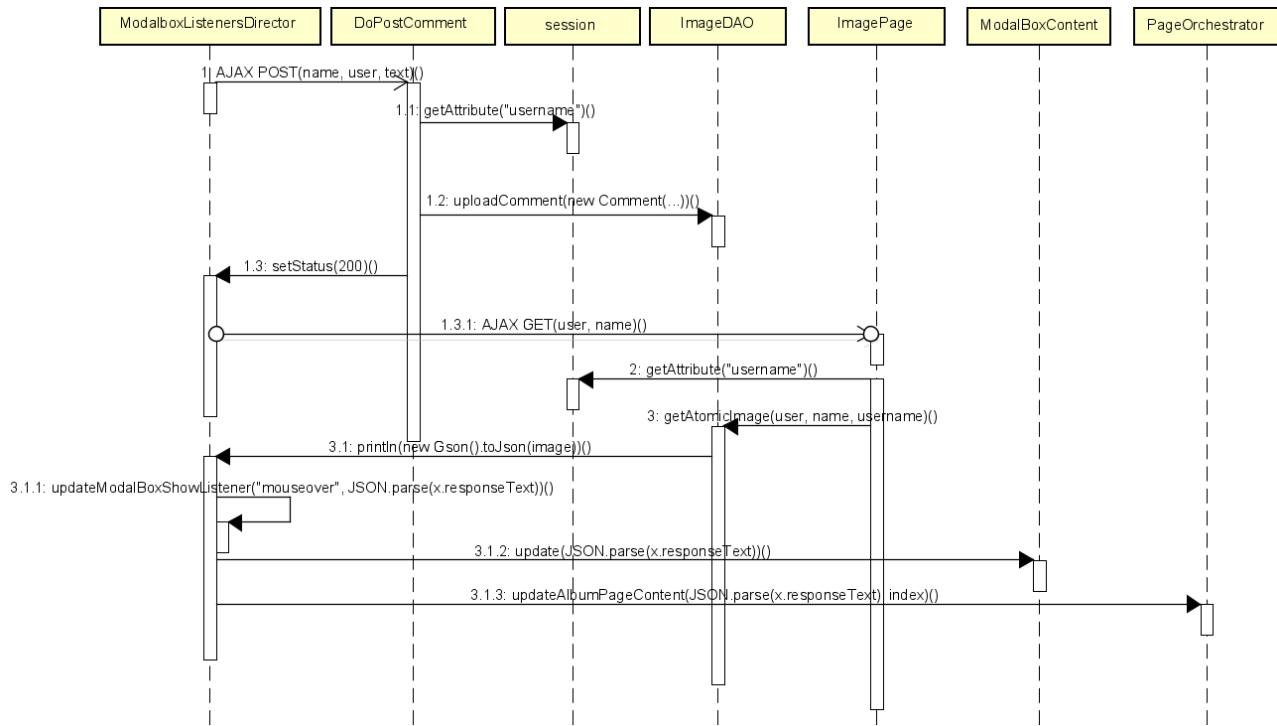
2.5.2.11 Click su “torna all’album”

Il sequence diagram che ne risulta è analogo a quello per il click del nome di un album (semplicemente si “parte” da “MenubarListenersDirector” poiché il bottone di ritorno all’album è parte della menubar).

2.5.2.12 “mouseover” su miniatura

Molto semplicemente, si invoca “update” di “ModalBoxContent” passando le informazioni precedentemente aggiunte al “ModalBoxListenersDirector” della finestra modale da mostrare (si veda il paragrafo “2.5.2.3”).

2.5.2.13 Click sul bottone per inviare un commento nella finestra modale



In buona sostanza viene prima caricato il commento e poi viene (solo se l'invio è andato a buon fine) richiesto il nuovo contenuto per la finestra modale al server. Ottenuto il contenuto aggiornato lo si “aggiunge” (coi metodi “update”) alla finestra modale. “updateAlbumPageContent” è utile ad aggiornare il contenuto dell’album page in maniera conforme ai dati appena aggiornati (questo evita che la navigazione tra le immagini di un album “ripristini” i contenuti per la finestra modale al loro stato prima dell’aggiornamento).

2.5.2.14 “dragover” su immagini nel form per l’ordinamento personalizzato

L’ordinamento personalizzato è gestito con una logica leggermente diversa dal “drag & drop” “classico”:

- l’utente inizia a trascinare una delle righe della tabella (evento “dragstart”);
- l’utente passa sopra un’altra riga (evento “dragover” sulla riga su cui si è passati sopra);
- si scambiano i contenuti delle due righe.

Avendo previsto lo scambio sul “dragover” anziché su eventi quali “drop” o “dragend” la scelta dell’ordinamento non è un vero e proprio “drag & drop”, lo potremmo chiamare “drag & swap” poiché l’utente non “trascina e rilascia” ma “trascina” e trascinando una riga sull’altra si effettua lo scambio delle righe.

2.5.2.15 Click su “salva ordinamento”

