Traccia "Galleria di Immagini"

Emanuele Paci e Andrea Riboni

aprile 2022

Contents

Passo 1

Analisi della traccia

1.1 Versione pure-HTML

Si vuole creare un'applicazione web per gestire una galleria di immagini

1.1.1 Struttura del sito

Homepage

l'utente che accede alla homepage vede un elenco degli album creati da se stesso e un elenco di quelli creati dagli altri utenti. Entrambi gli elenchi sono ordinati per data di creazione decrescente.
L'utente può cliccare un album e vedere la album page descritta di seguito.

Album page

La album page contiene una tabella di una riga e cinque colonne

- Ogni cella contiene una thumbnail e il titolo dell'immagine
- Le thumbnail sono ordinate da sinistra a destra per data decrescente
- Se l'album contiene più di 5 immagini sono disponibili comandi per vedere il precedente e il successivo insieme di 5 immagini
  - Se siamo al primo blocco di immagini e ne esistono di successive, a destra della riga si troverà il bottone SUCCESSIVE
  - Se ci troviamo in un blocco diverso dal primo e ne esistono di precedenti, a sinistra della riga si troverà il bottone PRECEDENTI
  - Se esistono sia blocchi precedenti che successivi compariranno entrambi i bottoni
- Le thumbnail sono cliccabili

La album page mantiene inoltre un collegamento per tornare alla homepage

Selezione di una thumbnail Quando si seleziona una miniatura, la album page mostra tutti i dati dell'immagine scelta, tra cui l'immagine in risoluzione originale e gli eventuali commenti presenti.
Viene mostrato anche il form per l'aggiunta di un commento con annesso bottone invia.

Invio di un commento Dopo aver inserito un commento e inviato il form attraverso il bottone invia la album page viene refreshata con tutti i dati aggiornati della stessa immagine

1.1.2 Funzionalità

Login, Logout e Registrazione

Registrazione e login tramite opportune form pubbliche

Gestione degli album

- caricamento di una foto
- creazione di un album
- associazione delle foto caricate ad un album

Aggiunta di un commento

Gli utenti possono aggiungere commenti alle foto, proprie e non

Registrazione occorre tenere conto di

- validità sintattica della mail
- uguaglianza tra *password* e *ripeti password*
- registrazione attraverso username univoco

1.1.3 Database

Si salvino le singole immagini come file nel file system del server

Dati presenti nel db

- Foto
  - titolo
  - data
  - testo descrittivo
  - path
  - utente che l'ha caricata
- Album
  - titolo
  - creatore
  - data di creazione
- Commenti
  - foto di riferimento
  - testo
  - utente che l'ha scritto

1.2 Versione RIA

Considerare la versione precedente e introdurre le modifiche descritte nelle righe seguenti

1.2.1 Login, Logout e Registrazione

- Viene controllata la validità sintattica della mail e l'uguaglianza tra i due campi password anche lato client
- Dopo il login, l'intera applicazione viene realizzata con un'unica pagina

1.2.2 Richieste asincrone

Ogni interazione dell'utente viene gestita senza ricaricare completamente la pagina ma invocando richieste asincrone al server che andranno eventualmente a modificare il contenuto da aggiornare. In particolare abbiamo che

Visualizzazione di un album gli eventi associati ai bottoni PRECEDENTI e SUCCESSIVE per visualizzare un differente set di 5 foto vengono gestiti lato client senza generare richieste al server

1.2.3 Javascript sui form

Anteprima di una thumbnail Quando l'utente passa con il mouse su una thumbnail, l'applicazione mostra una finestra modale con tutte le informazioni dell'immagine relativa (tra cui l'immagine a grandezza naturale, i commenti e il form per inserire un commento)

Invio di commenti Viene controllato lato client che non si invii un commento vuoto

1.2.4 Gestione degli errori

Gli errori che avvengono lato server devono essere segnalati mediante un messaggio di allerta all'interno della pagina

1.2.5 Gestione degli album

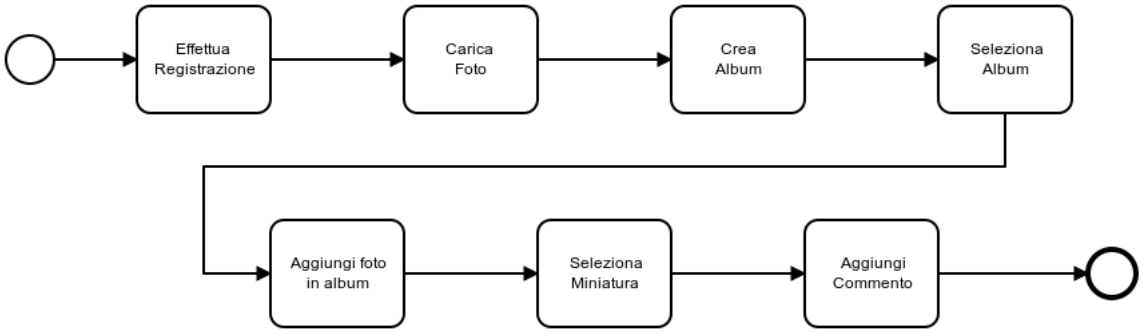
L'utente può riordinare i propri album con criteri diversi da quello di default (per data decrescente)

- L'utente trascina il titolo di un album nell'elenco e lo colloca in una posizione diversa per realizzare l'ordinamento, senza che venga notificato il server
- Una volta che l'utente ha raggiunto l'ordinamento desiderato, usa un bottone *salva ordinamento* per memorizzare la sequenza sul server
- Gli accessi successivi mostreranno l'ordinamento personalizzato

1.3 Completamento delle specifiche

Non viene descritto il funzionamento del caricamento di nuove foto e dell'aggiunta di foto esistenti agli album: optiamo allora per il seguente flusso:

- L'utente all'interno della homepage ha a disposizione sia un form per effettuare l'upload di una nuova foto, sia per creare un nuovo album
  - La creazione di un album richiede all'utente di specificarne solo il titolo e, una volta effettuata la richiesta, verrà rappresentato l'album appena creato come album vuoto
  - Il form di upload di una foto richiede invece il titolo della foto, il file da caricare effettivamente e, eventualmente, la descrizione (testo alternativo). L'utente effettuerà l'upload di una foto e questa verrà memorizzata nel file-system del server ma, di fatto, mai visualizzata esplicitamente all'interno di ImageGallery fino a che non verrà aggiunta ad un album (chiameremo *sfuse* le foto prive di album)
- All'interno della albumpage sarà invece presente un form per aggiungere la foto all'album visualizzato: il form presenterà la possibilità di aggiungere all'album una foto qualsiasi tra quelle caricate dall'utente creatore dell'album (nonché unico a poter effettuare questa azione) e che non siano già presenti al suo interno



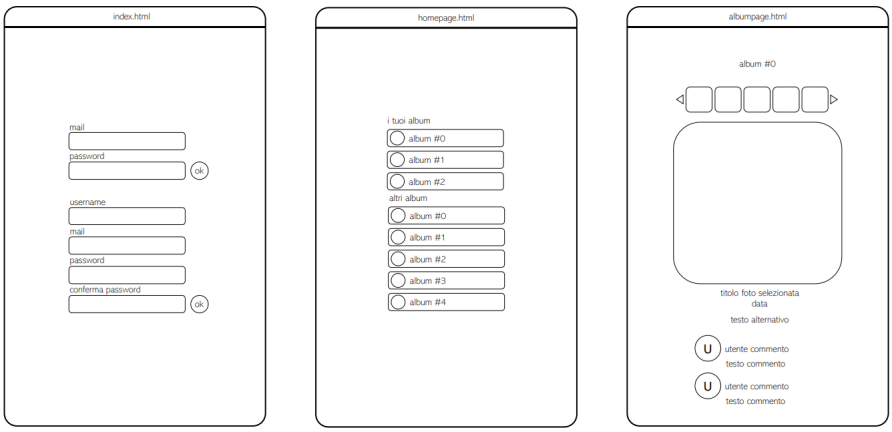
1.4 Scelte progettuali

- Gli album manterranno nel db un attributo *sort_id* che determinerà, per ogni utente, l'ordine di visualizzazione dei propri album
- L'utente può creare album con lo stesso nome
- L'utente può caricare foto con lo stesso nome
- Un album può essere vuoto
- Una foto può appartenere a più album: i suoi commenti verranno replicati
- La foto può non appartenere a nessun album (appena caricata, in attesa di essere aggiunta a un album)
- Non è possibile aggiungere più di una volta una foto nello stesso album
- Email e Username degli utenti devono essere univoci
- Nelle varie servlet l'username viene letto dalla session
- Viene utilizzato bootstrap (in particolare, la versione "MDBootstrap") e si cerca di non utilizzare esplicitamente JQuery
- Le password degli utenti, per garantire un minimo livello di sicurezza, vengono memorizzate nel db come il proprio digest (SHA-256)

1.4.1 Mantenimento delle foto

Inizialmente le foto venivano memorizzate in una cartella *photos* interna alla workspace del progetto. Ignorando momentaneamente il fatto che questa non sia una buona pratica, questo creava problemi relativi al fatto che il database usato per le due versioni è unico ma le workspaces sono differenti: se un utente caricava una foto in pureHTML, questa poi risultava caricata anche in RIA ma non era visualizzabile in quanto il path relativo faceva riferimento ad un'altra cartella madre.
Per questo motivo le foto vengono ora salvate in un path esterno alle due workspaces e la servlet *DownloadPhoto* effettua il mapping da URL a file sul filesystem.

1.4.2 Idea della rappresentazione grafica



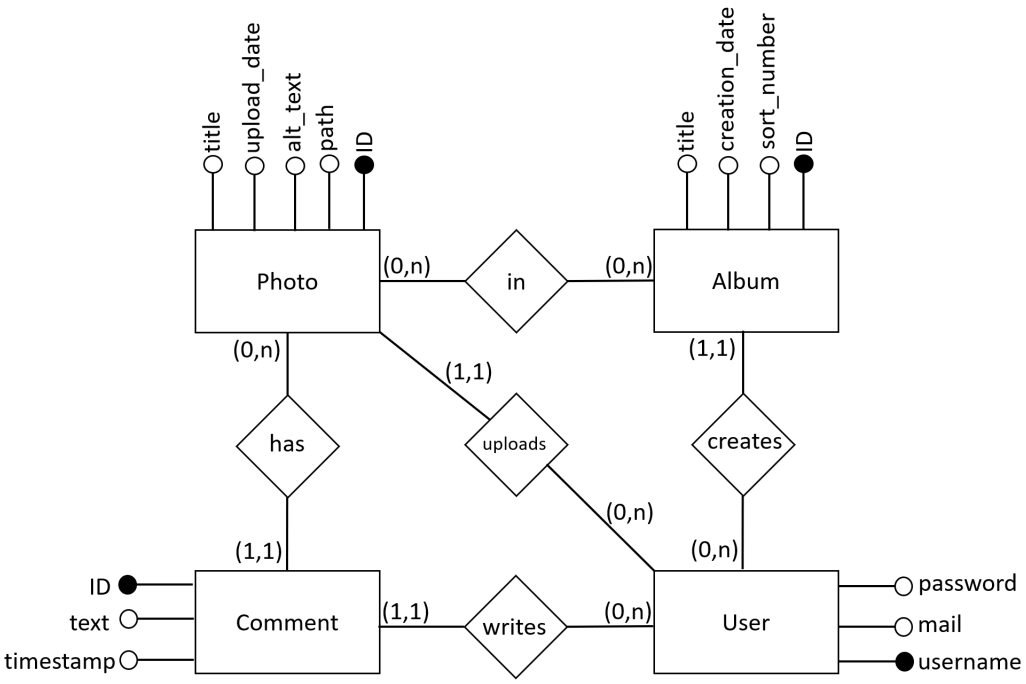
Passo 2

Progettazione pure-HTML

2.1 Progettazione del database

2.1.1 Progettazione concettuale

Di seguito il diagramma ER progettato



2.1.2 Schema logico

Dal diagramma ER abbiamo ricavato il seguente schema logico

- User(<u>username</u>, mail, password)
- Album(<u>ID</u>, title, creation_date, sort_number, IDUser)
  - IDUser references User.username
- Photo(<u>ID</u>, title, upload_date, alt_text, path, IDUser)
  - IDUser references User.username
- Comment(<u>ID</u>, text, timestamp, IDPhoto, IDUser)
  - IDPhoto references Photo.ID
  - IDUser references User.username
- PhotoInAlbum(<u>IDPhoto</u>, <u>IDAlbum</u>)
  - IDPhoto references Photo.ID
  - IDAlbum references Album.ID

2.1.3 Tabelle SQL

```
CREATE TABLE IF NOT EXISTS User (
    username VARCHAR(25) PRIMARY KEY,
    mail VARCHAR(89) UNIQUE NOT NULL,
    password VARCHAR(64) NOT NULL
);

CREATE TABLE IF NOT EXISTS Photo (
    ID INTEGER AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(40) NOT NULL,
    upload_date DATE NOT NULL DEFAULT (CURRENT_DATE),
    alt_text TEXT,
    path VARCHAR(256) NOT NULL,
    IDUser VARCHAR(25) NOT NULL REFERENCES User.username ON UPDATE CASCADE ON DELETE
    CASCADE
);

CREATE TABLE IF NOT EXISTS Album (
    ID INTEGER AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(40) NOT NULL,
    creation_date DATE NOT NULL DEFAULT (CURRENT_DATE),
    sort_id INT NOT NULL DEFAULT 0,
    IDUser VARCHAR(25) NOT NULL REFERENCES User.username ON UPDATE CASCADE ON DELETE
    CASCADE
);

CREATE TABLE IF NOT EXISTS Comment (
    ID INTEGER AUTO_INCREMENT PRIMARY KEY,
    text TEXT NOT NULL,
    tstamp TIMESTAMP NOT NULL DEFAULT (CURRENT_TIMESTAMP),
    IDUser VARCHAR(25) NOT NULL REFERENCES User.username ON UPDATE CASCADE ON DELETE
    CASCADE,
    IDPhoto INT NOT NULL REFERENCES Photo.ID ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS PhotoInAlbum (
    IDPhoto INT NOT NULL REFERENCES Photo.ID ON UPDATE CASCADE ON DELETE CASCADE,
    IDAlbum INT NOT NULL REFERENCES Album.ID ON UPDATE CASCADE ON DELETE CASCADE,
    PRIMARY KEY(IDPhoto, IDAlbum)
);
```

2.2 Componenti

2.2.1 Beans

- Album
- Comment
- Photo
- User

2.2.2 DAOs

- AlbumDAO
  - getAlbumByUser(username, avoid) : List of Album
  - getAlbumByID(id_album) : Album
  - createAlbum(title, username) : int
  - getPhotoIfContained(album_id, photo_id) : Photo
  - checkCreator(username, album_id) : boolean
  - addPhotoIntoAlbum(album_id, photo_id) : int
- CommentDAO

  - getCommentsByPhoto(photo_id) : List of Comment
  - createComment(text, username, photo_id) : int
- PhotoDAO
  - getPhotosByAlbum(album_id, start) : List of Photo
  - getPhotoByUserNotInAlbum(user_id, album_id) : List of Photo
  - createPhoto(title, altText, path, user) : int
- UserDAO
  - createUser(username, mail, password) : int
  - getUser(username) : User
  - checkCredentials(username, password) : User
  - checkByUsername(username) : boolean
  - checkByEmail(email) : boolean

2.2.3 Filters

- HomeChecker
- LoginChecker

2.2.4 Controllers

Sintassi: Nome servlet [diritti di accesso]

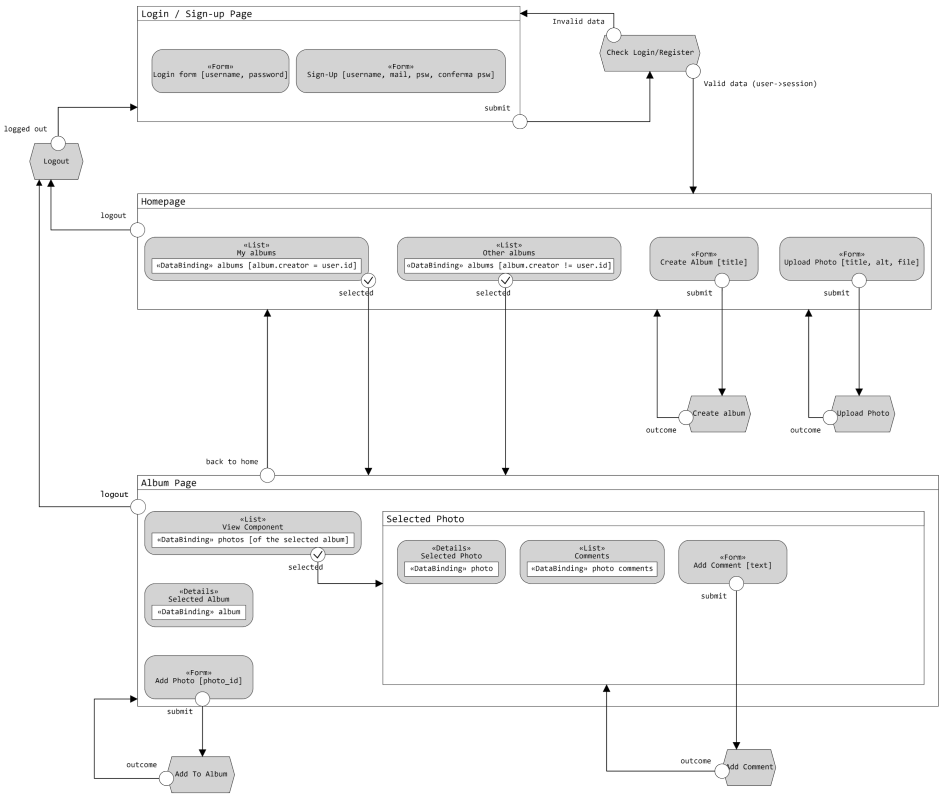
- AddComment [logged users]
- AddToAlbum [logged users]
- CreateAlbum [logged users]
- DownloadPhoto [logged users]
- LoginRegister [non-logged users]
- Logout [all]
- ShowAlbum [logged users]
- ShowHomepage [logged users]
- UploadPhoto [logged users]

2.2.5 Views

- index.html (login)
- homepage.html
- albumpage.html
- errorpage.html

2.3 IFML

Di seguito viene riportato il diagramma IFML delle principali interazioni. Maggiori dettagli verranno riportati attraverso i sequence diagram



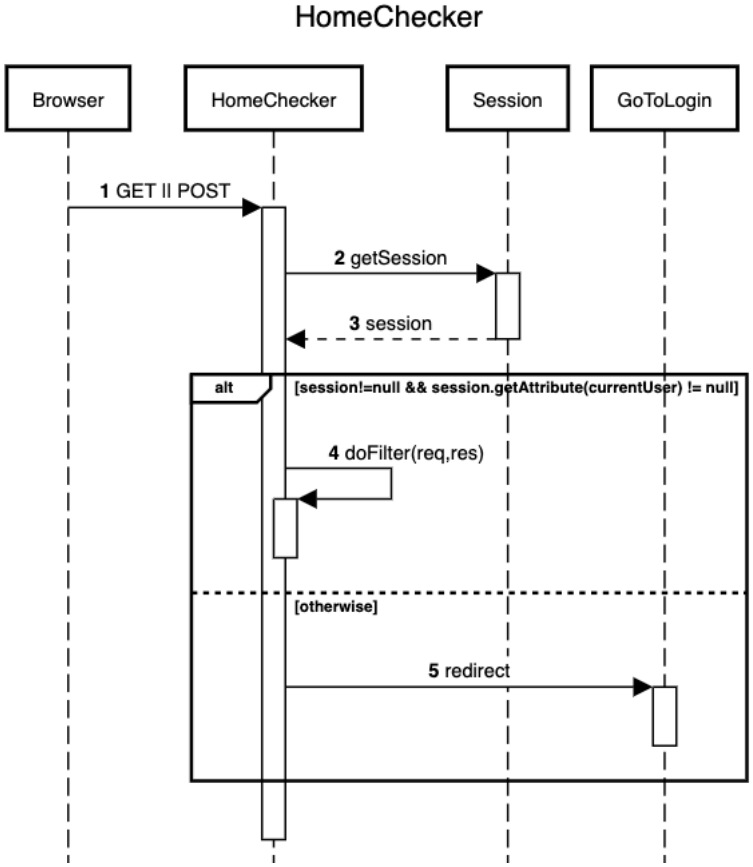
2.4 Diagrammi di sequenza

Per quanto riguarda i diagrammi di sequenza, bisogna tenere conto che, nella realtà dei fatti

- Quando viene eseguito *forwardToErrorPage* l'esecuzione termina e viene distribuita una pagina d'errore opportunamente dettagliata
- Quando il browser richiede il rendering di un'immagine caricata precedentemente, farà una richiesta alla servlet *DownloadPhoto* specificando come parametro l'ID della foto che necessita ottenere
- Quando una servlet necessita di conoscere lo username dell'utente che ha effettuato l'accesso, lo legge attraverso la sessione attiva corrente

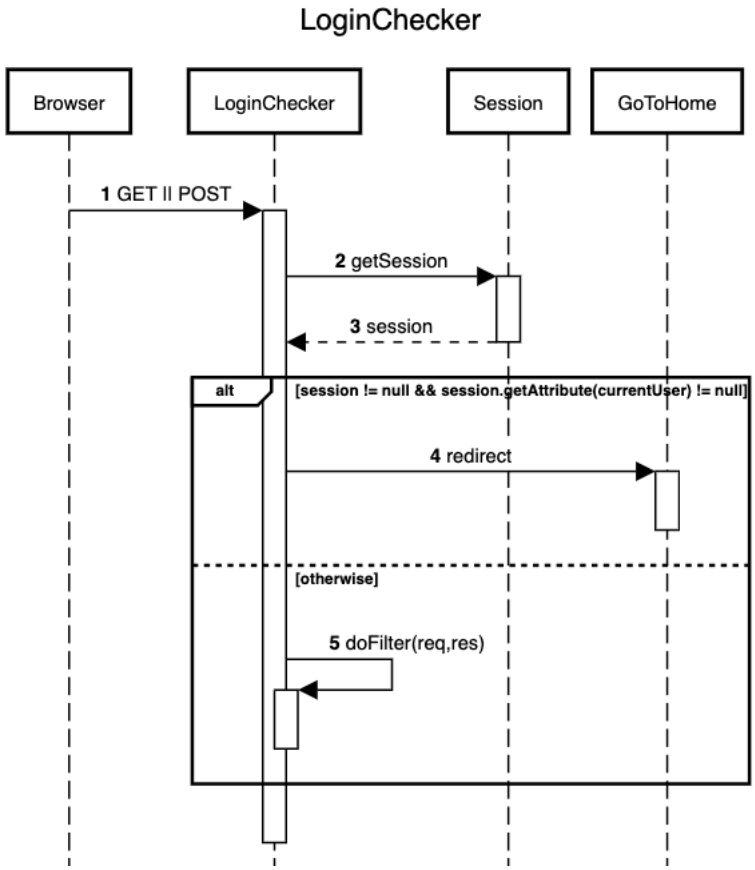
2.4.1 Home Checker

Home Checker è una filter che controlla che l'utente non loggato non possa accedere alle servlet per cui non dispone sufficienti permessi (riferimento a 2.2.4) Il suo funzionamento non è stato approfondito negli altri sequence diagram ma è sempre in esecuzione.

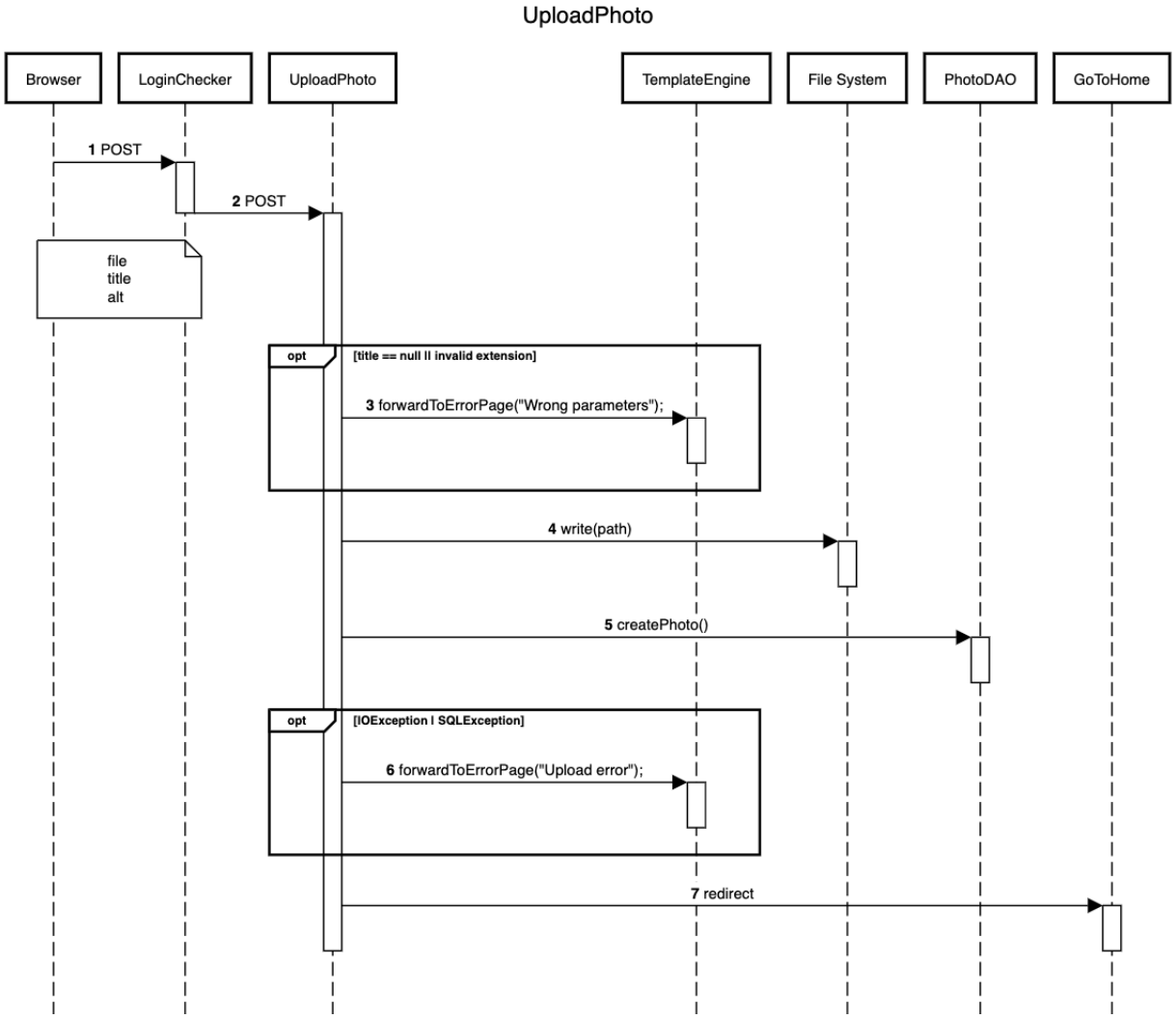


2.4.2 Login Checker

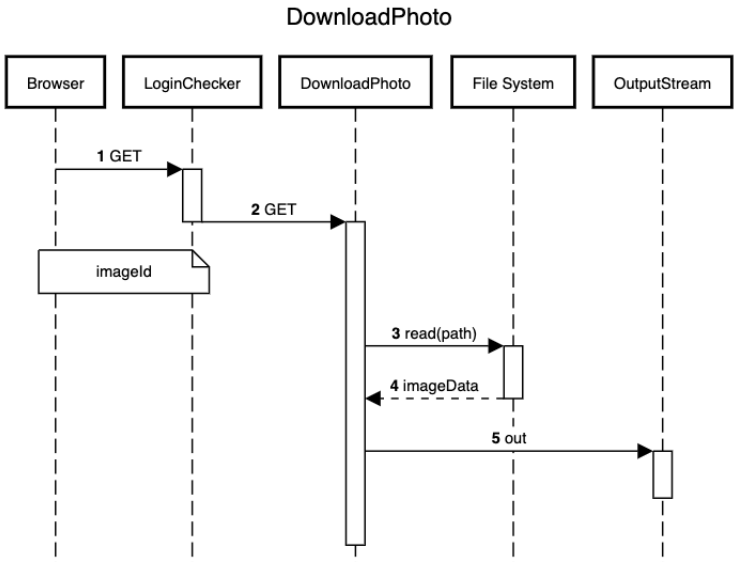
Login Checker è la filter duale a HomeChecker: riferimento a 2.2.4. Il suo funzionamento non è stato approfondito negli altri sequence diagram ma è sempre in esecuzione.



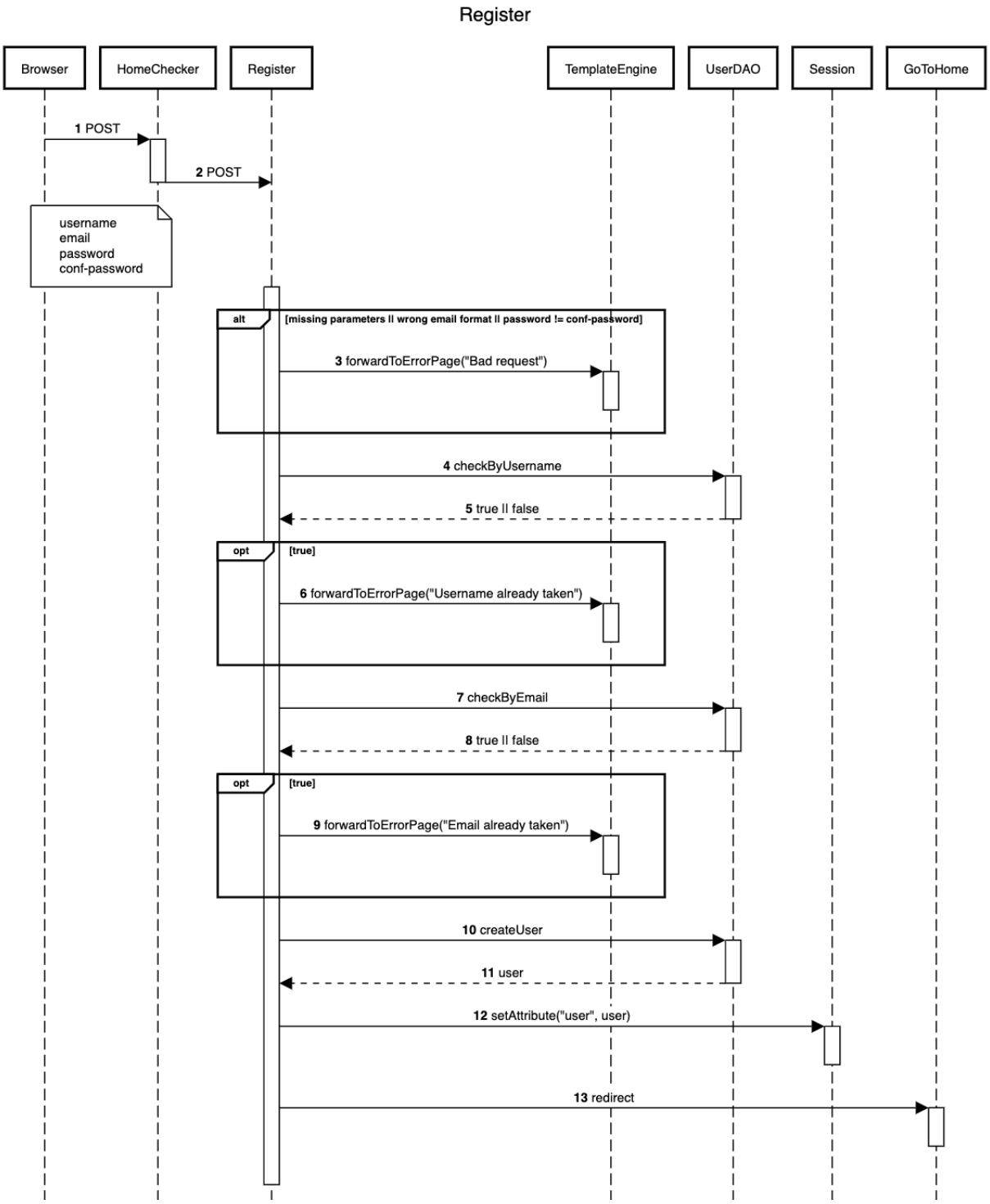
2.4.3 Upload Photo



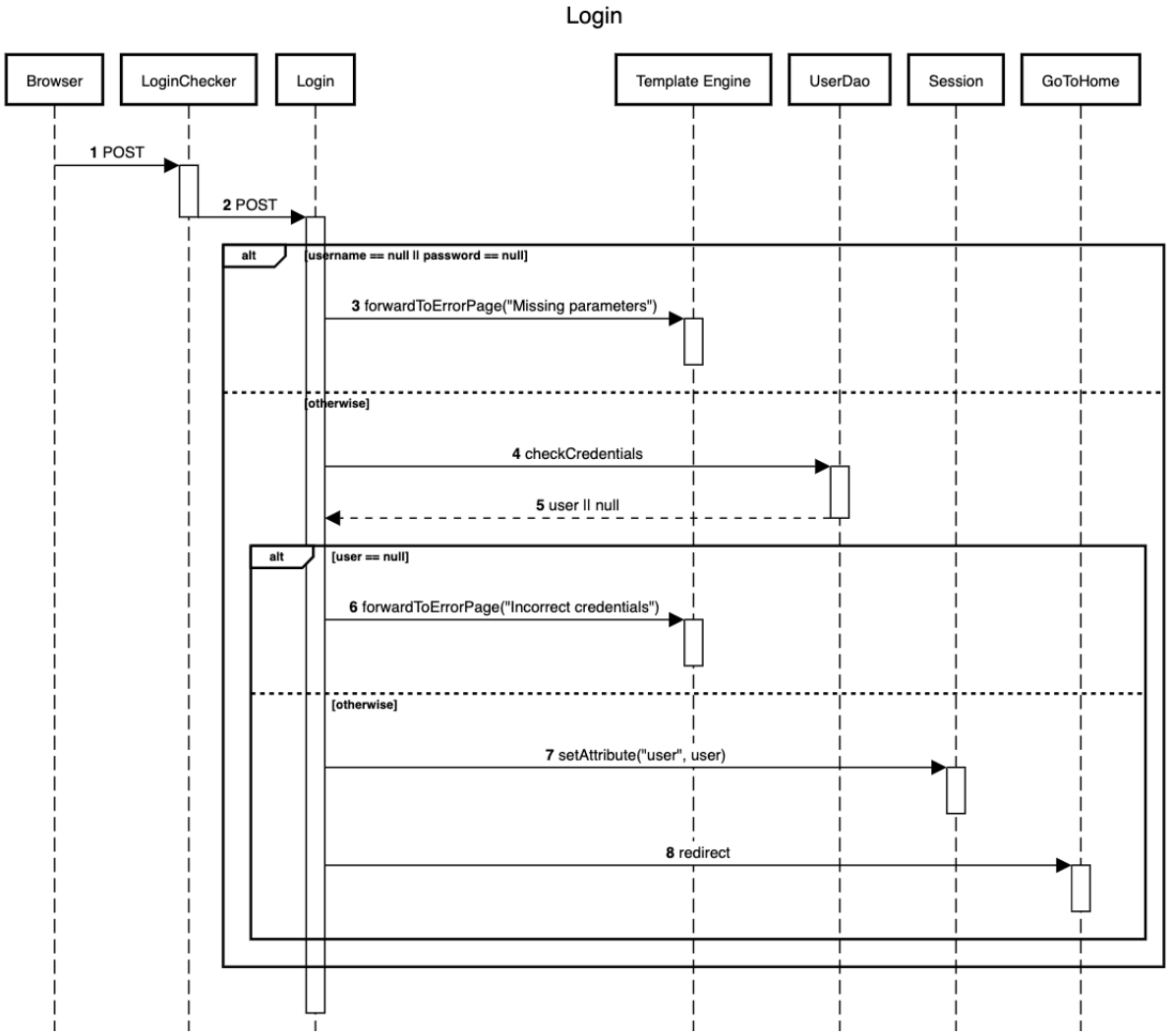
2.4.4 Download Photo



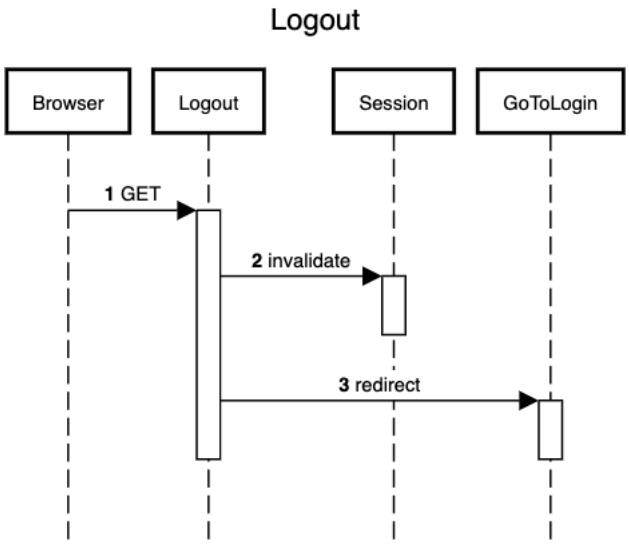
2.4.5 Register



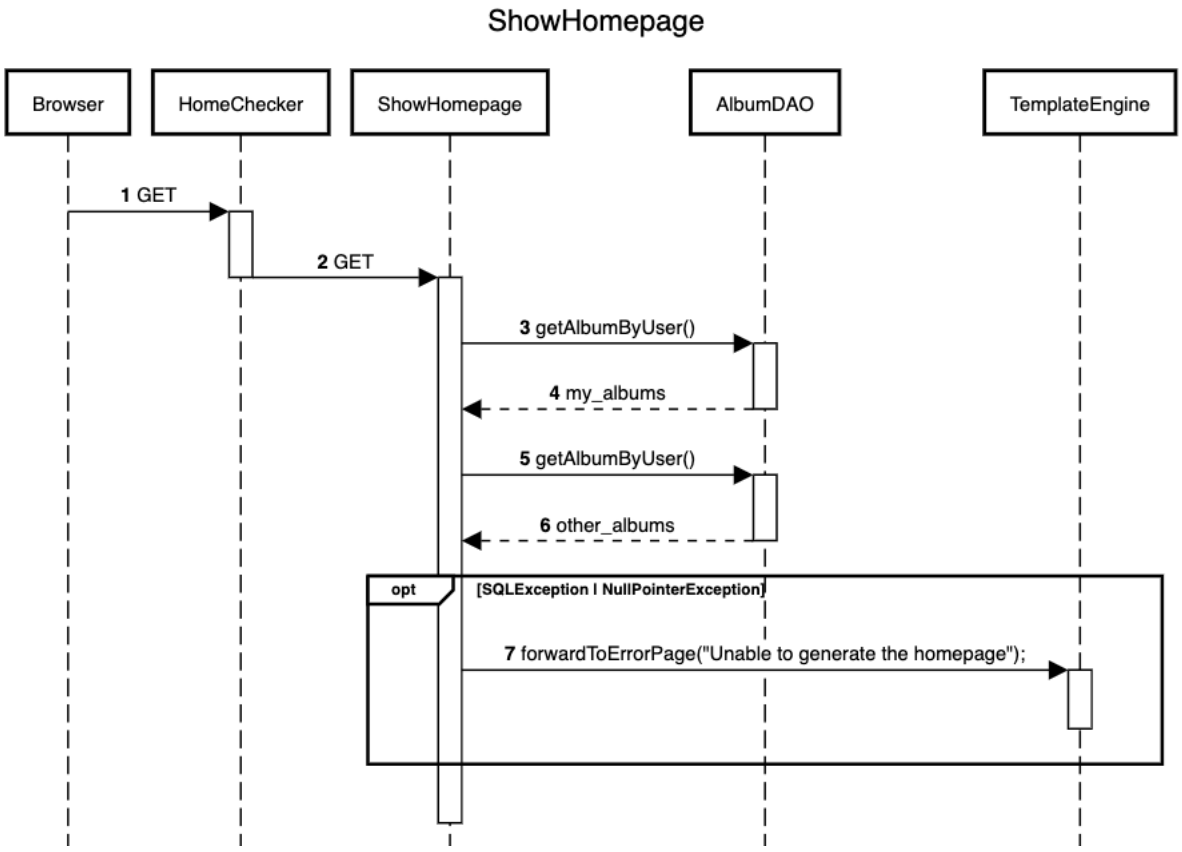
2.4.6 Login



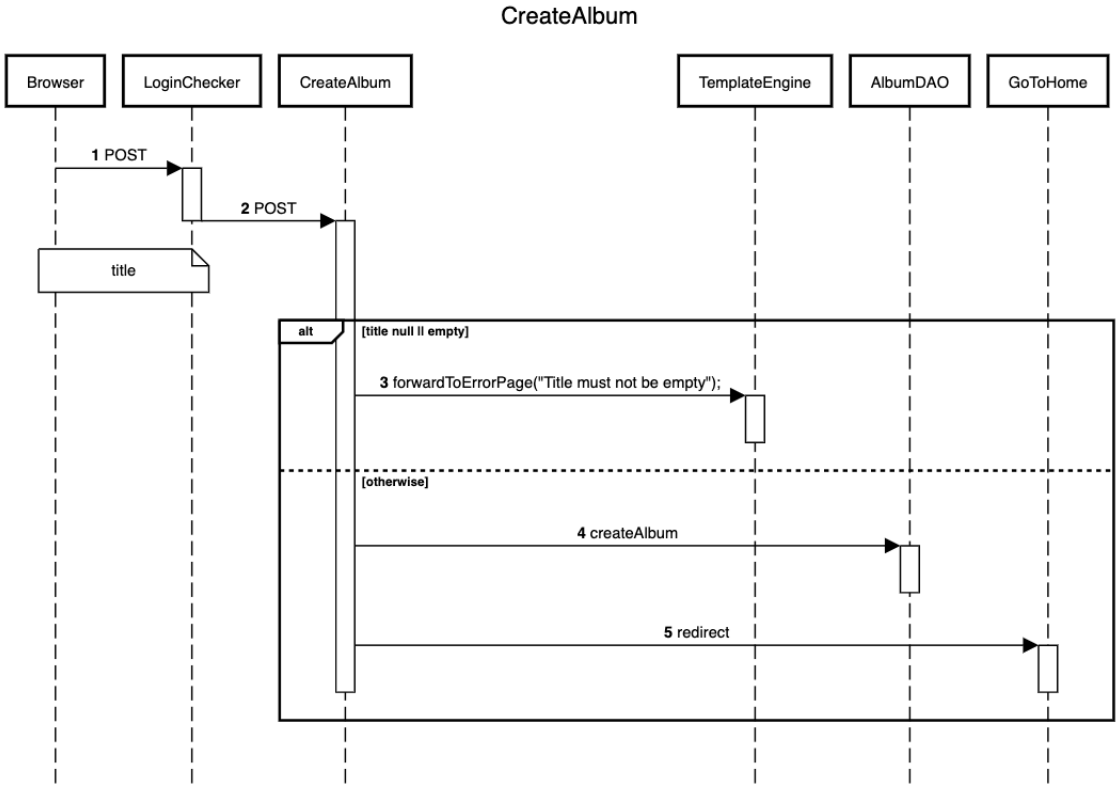
2.4.7 Logout



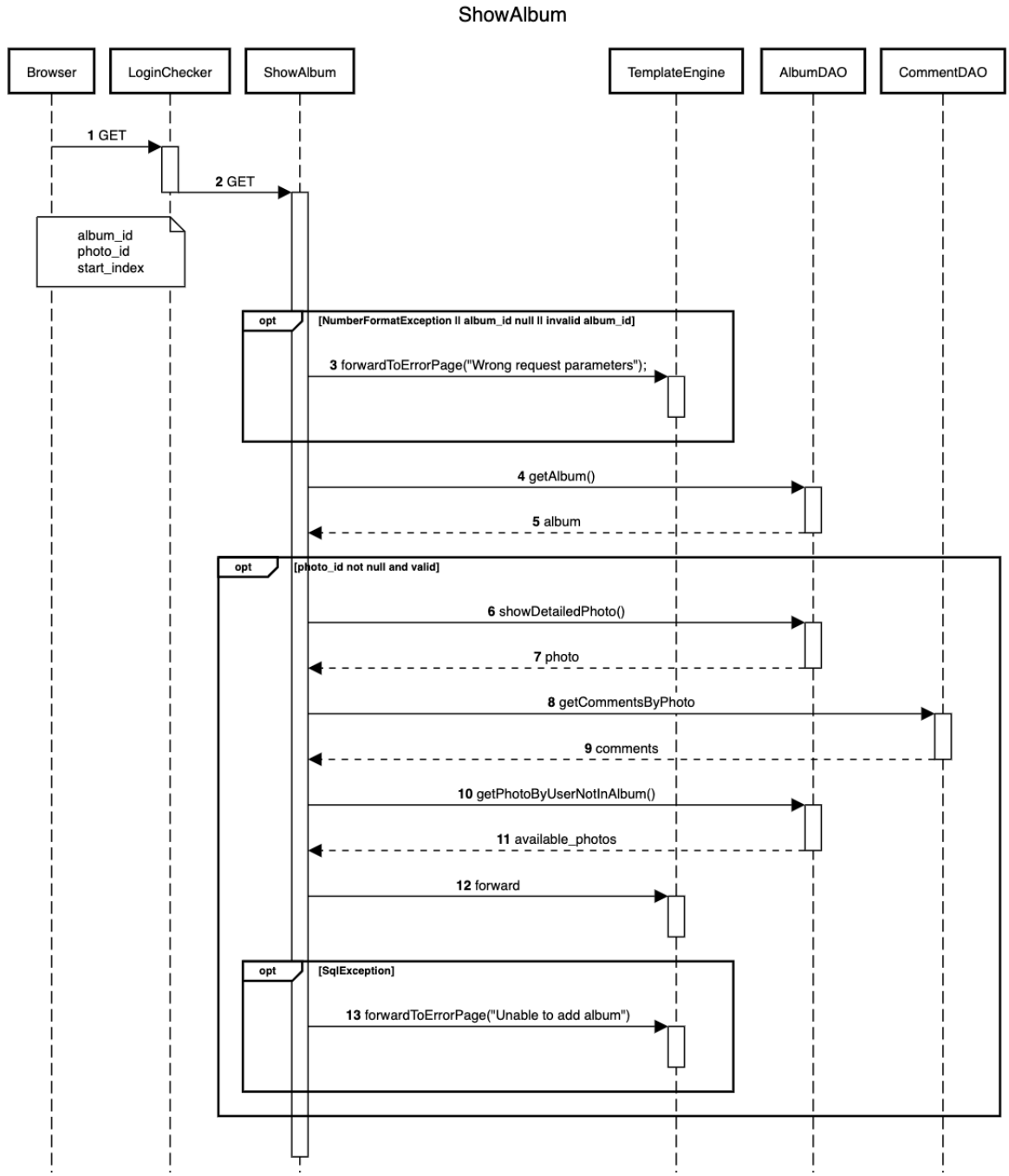
2.4.8 Show Homepage



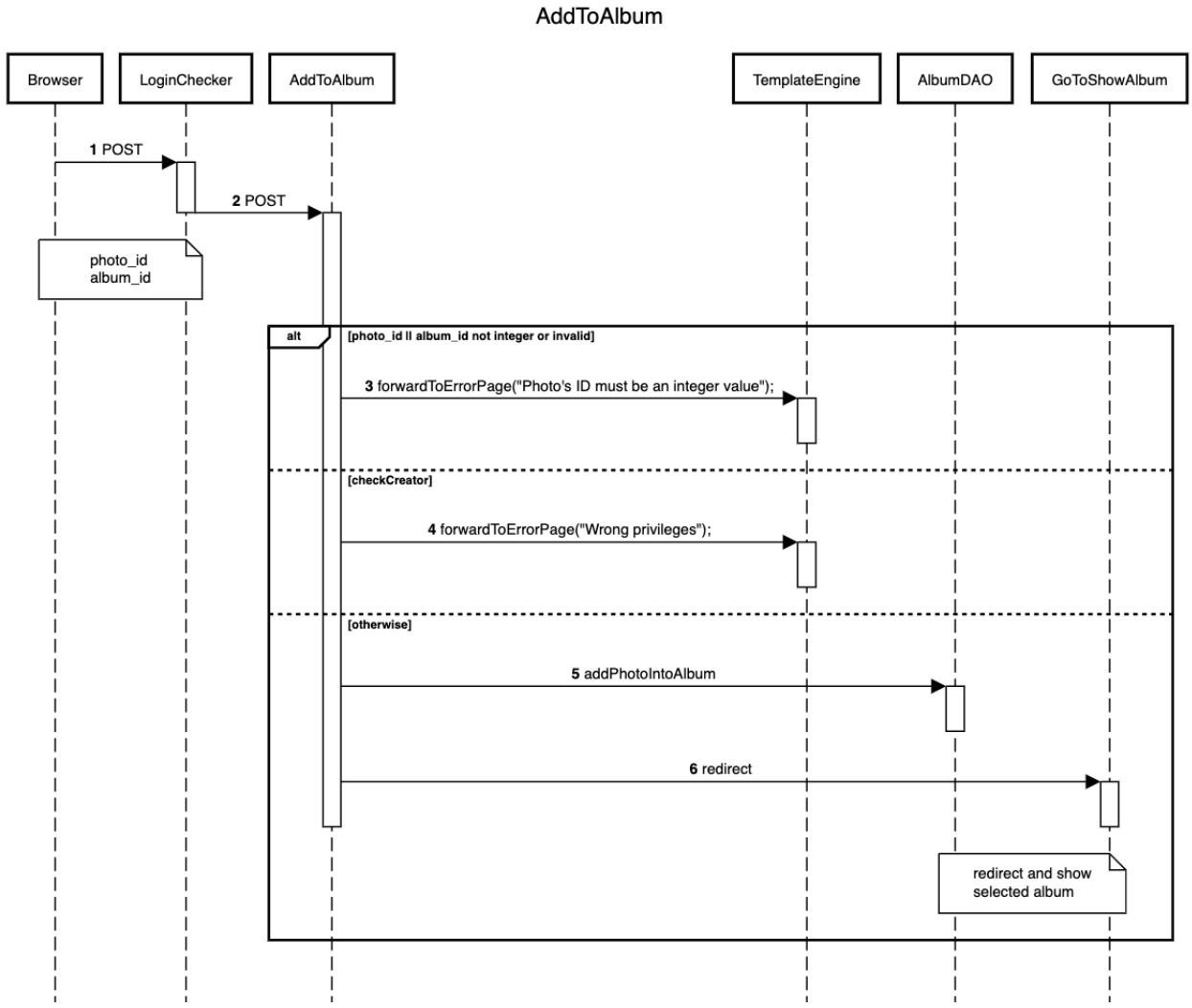
2.4.9 Create Album



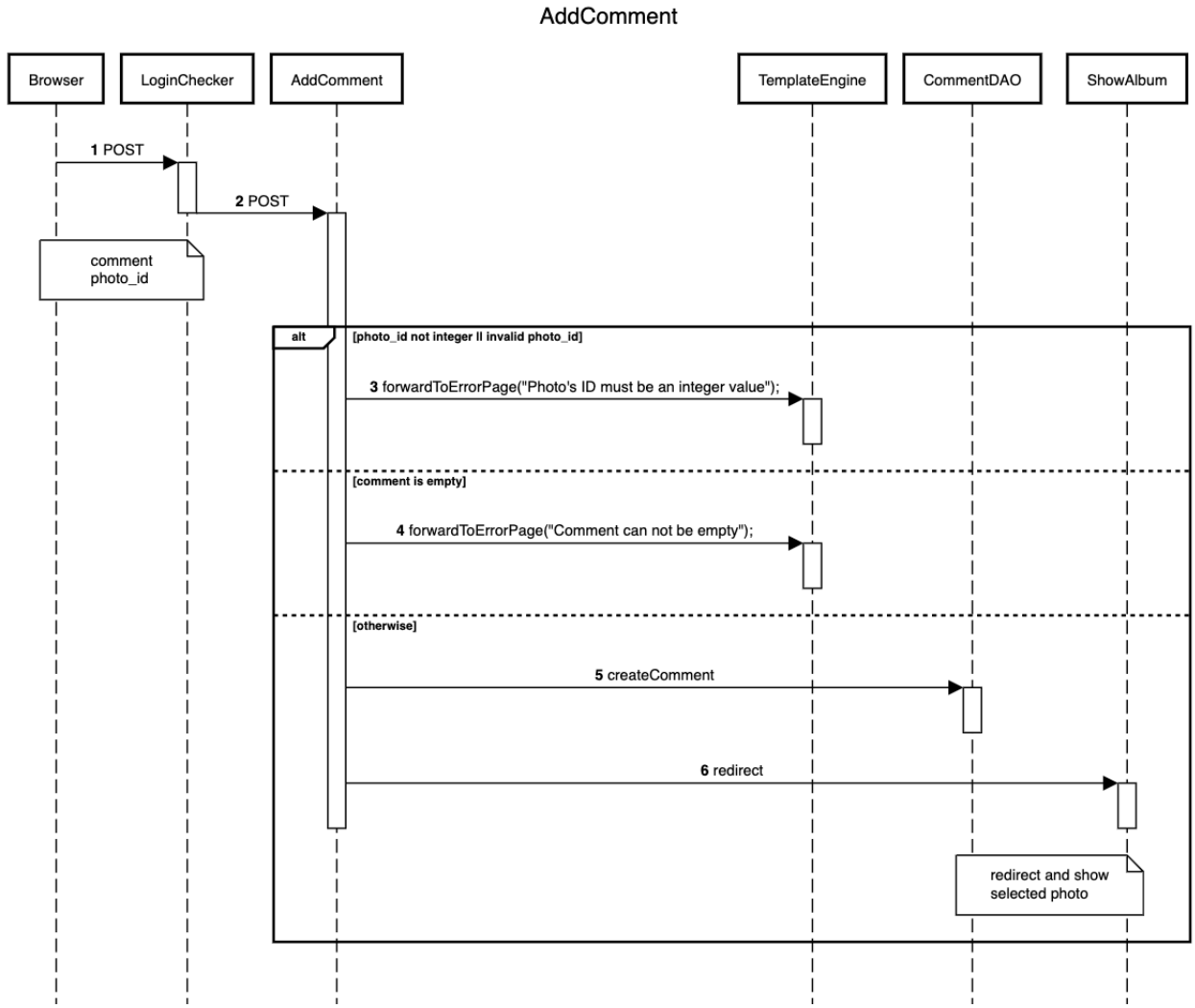
2.4.10 Show Album



2.4.11 Add To Album



2.4.12 Add Comment



Passo 3

Progettazione RIA

3.1 Aggiunta di funzionalità user-friendly

Sono state introdotte le seguenti funzionalità

- Possibilità di inviare i form con il pulsante INVIO
- Scroll automatico per mettere in risalto punti della pagina che potrebbero cambiare in maniera asincrona (per esempio, il banner che mostra eventuali errori)
- Auto-dismiss del banner di errore dopo 10 secondi dalla sua creazione
- Navbar con collegamenti rapidi a homepage e logout

3.2 Progettazione del database

Il database è in comune tra la versione RIA e quella pureHTML, dunque le specifiche delineate in 2.1 non subiscono cambiamenti.

3.3 Componenti lato server

3.3.1 Beans

Anche i beans sono rimasti inalterati e si può quindi continuare a fare riferimento alla sezione 2.2.1.

3.3.2 DAOs

I DAO subiscono leggere modifiche: verranno riportate in grassetto le aggiunte e rigate le eliminazioni

- AlbumDAO
  - getAlbumByUser(username, avoid) : List of Album
  - getAlbumByID(id_album) : Album
  - createAlbum(title, username) : int
  - ~~getPhotoIfContained(album_id, photo_id) : Photo~~
  - checkCreator(username, album_id) : boolean
  - addPhotoIntoAlbum(album_id, photo_id) : int
  - **updateOrder(username, albums_id) : void**
- CommentDAO
  - getCommentsByPhoto(photo_id) : List of Comment
  - createComment(text, username, photo_id) : int
- PhotoDAO
  - getPhotosByAlbum(album_id, start) : List of Photo

  - getPhotoByUserNotInAlbum(user_id, album_id) : List of Photo
  - createPhoto(title, altText, path, user) : int
  - **findPhotoById(photoId) : Photo**
- UserDAO
  - createUser(username, mail, password) : int
  - getUser(username) : User
  - checkCredentials(username, password) : User
  - checkByUsername(username) : boolean
  - checkByEmail(email) : boolean

3.3.3 Packets

- AlbumPacket
  - Album
  - List of album's photos
  - List of available-to-add photos
- PhotoPacket
  - Photo
  - List of photo's comments

3.3.4 Controllers

Sintassi: Nome servlet [diritti di accesso]. Le aggiunte vengono evidenziate con il grassetto e le rimozioni vengono barrate

- AddComment [logged users]
- AddToAlbum [logged users]
- CreateAlbum [logged users]
- DownloadPhoto [logged users]
- **GetAlbumDetails [logged users]**
- **GetAlbums [logged users]**
- **GetPhoto [logged users]**
- ~~LoginRegister [non-logged users]~~
- **Login [non logged users]**
- Logout [all]
- **Register [non logged users]**
- ~~ShowAlbum [logged users]~~
- ~~ShowHomepage [logged users]~~
- **UpdateOrder [logged users]**
- UploadPhoto [logged users]

3.3.5 Views

- index.html (login)
- homepage.html
- ~~albumpage.html~~
- ~~errorpage.html~~

3.4 Componenti lato client

3.4.1 LoginManagement

- Gestione del form di login
- Gestione del form di registrazione

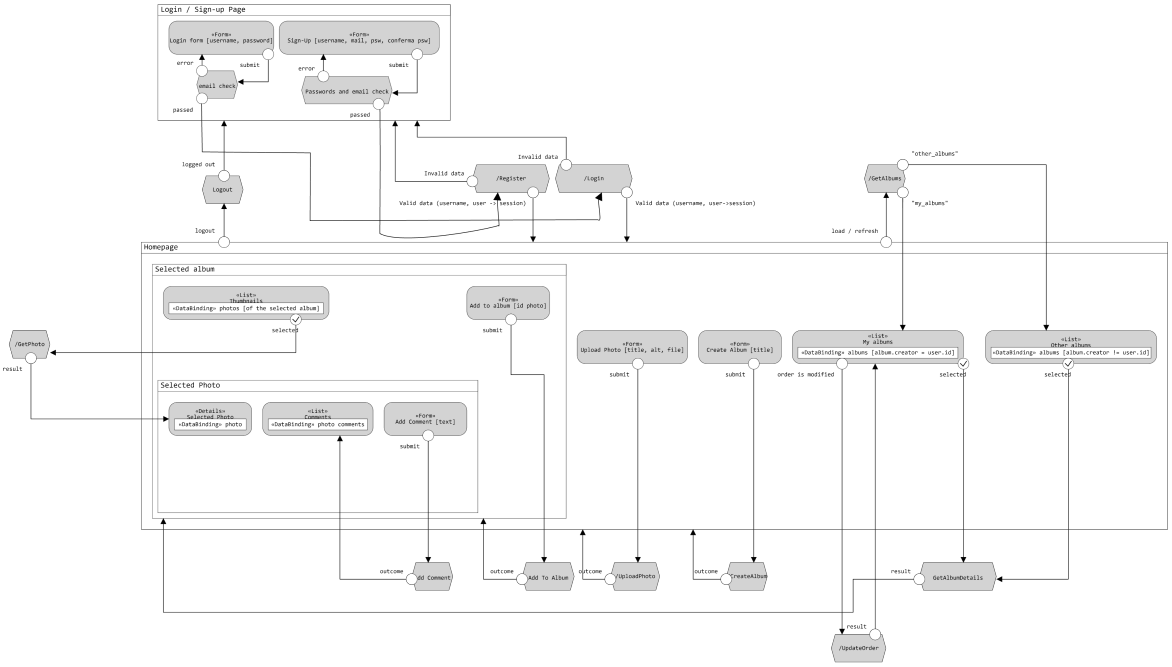
Entrambe le funzionalità supportano l'invio dei dati del form relativo alla corretta servlet e una visualizzazione degli errori in un banner

3.4.2 ImageGalleryManagement

- **AlbumList**
  - **reset** per re-inizializzare la visualizzazione di una lista di album
  - **update** per aggiornare la visualizzazione parziale di una lista di album
- **AlbumDetails**
  - **reset** per re-inizializzare la visualizzazione dettagliata dell'album selezionato
  - **update** per aggiornare la visualizzazione parziale dell'album selezionato
  - **requestPhoto** per effettuare la richiesta di ottenimento dei dati relativi alla foto selezionata
  - **showPhoto** per aggiornare la visualizzazione parziale della foto selezionata
  - **showComments** per visualizzare (e aggiornare) la lista di commenti relativi alla foto selezionata
  - **addAvailablePhoto** per aggiungere una foto che è possibile includere nell'album selezionato nel relativo form
  - **refreshComments** per aggiornare i commenti
  - **refreshAvailablePhotos** per aggiornare le foto che è possibile includere nell'album selezionato
- **FormManager**
  - **start** per aggiungere i listener ai form presenti
  - **sendForm** per l'invio dei dati presenti in un form alla relativa servlet
- **PageOrchestrator**
  - **start** per inizializzare gli oggetti che gestiscono la view della pagina
  - **refresh** per aggiornare la lista dei propri album e la lista di quelli altrui
  - **refreshMyAlbums** per aggiornare la sola lista dei propri album
  - **refreshComments** per aggiornare la lista di commenti relativi alla foto selezionata
  - **refreshAvailablePhotos** per aggiornare la lista di foto che è possibile includere nell'album selezionato

3.5 IFML

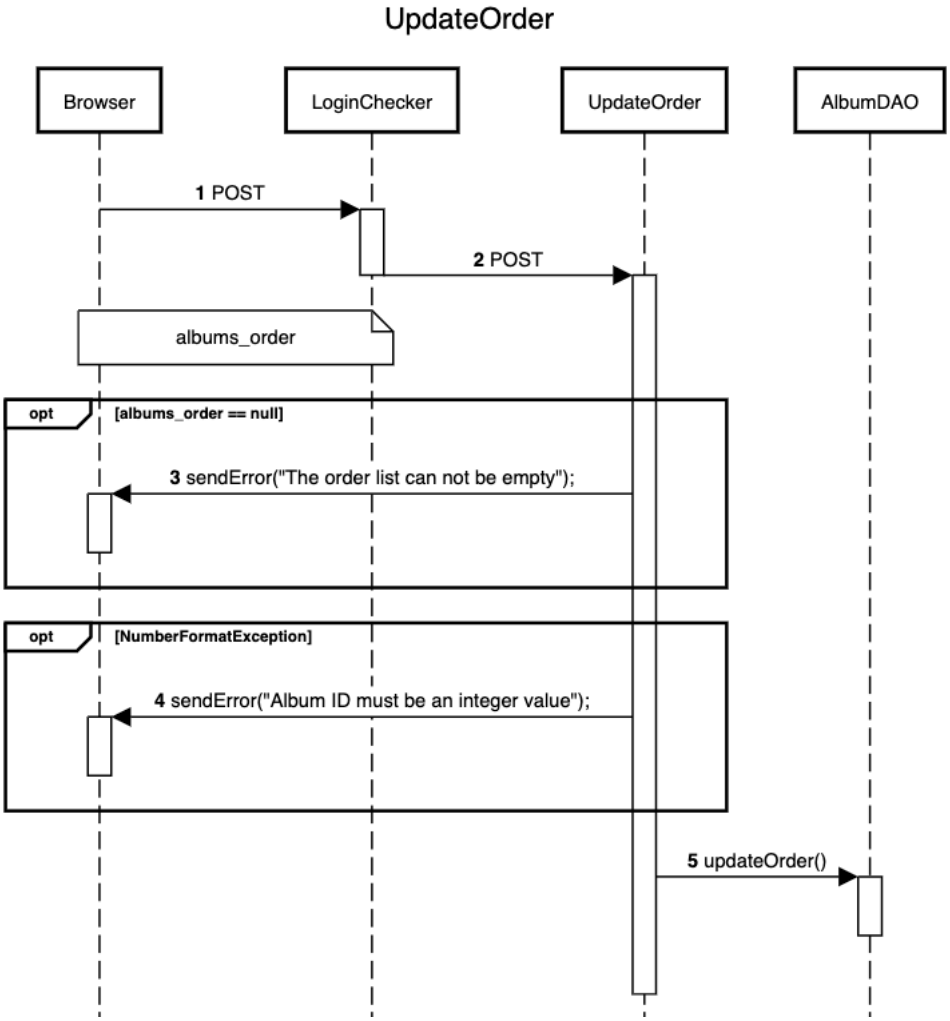
Di seguito viene riportato il diagramma IFML delle principali interazioni. Maggiori dettagli verranno riportati attraverso i sequence diagram



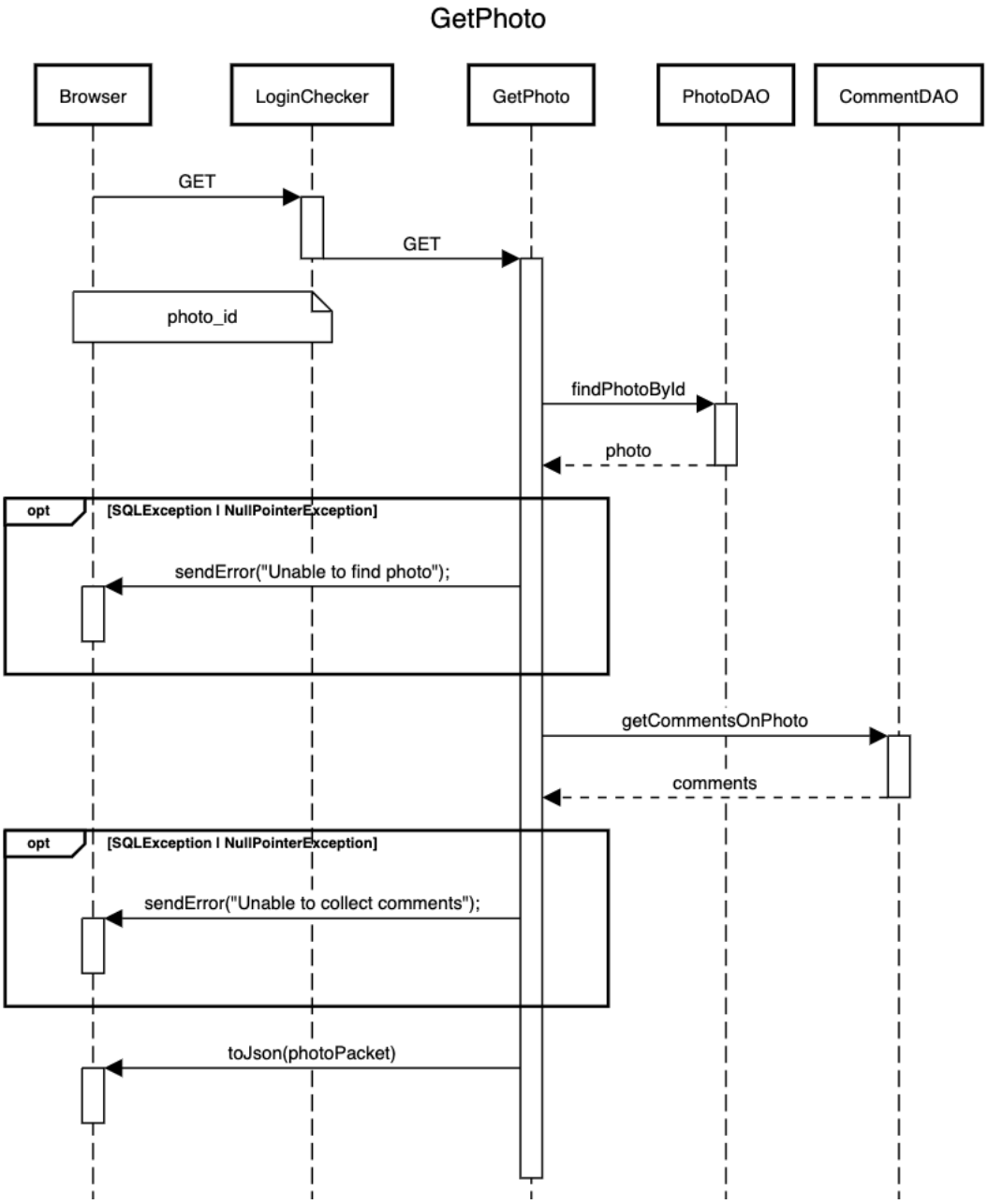
3.6 Diagrammi di sequenza

In aggiunta ai diagrammi di sequenza della versione pure-HTML, per questo sito abbiamo bisogno di servlet aggiuntive, rappresentate qui sotto.
In tutti i sequence diagram in cui dovesse essere eseguito sendError, successivamente l'esecuzione termina. Le servlet che richiedono il rendering di foto utilizzano DownloadPhoto, che fornisce come response la foto richiesta (parametro IDPhoto). I diagrammi delle servlet già esistenti rimangono pressoché inalterati (eccezion fatta per la response JSON) e non vengono riportati

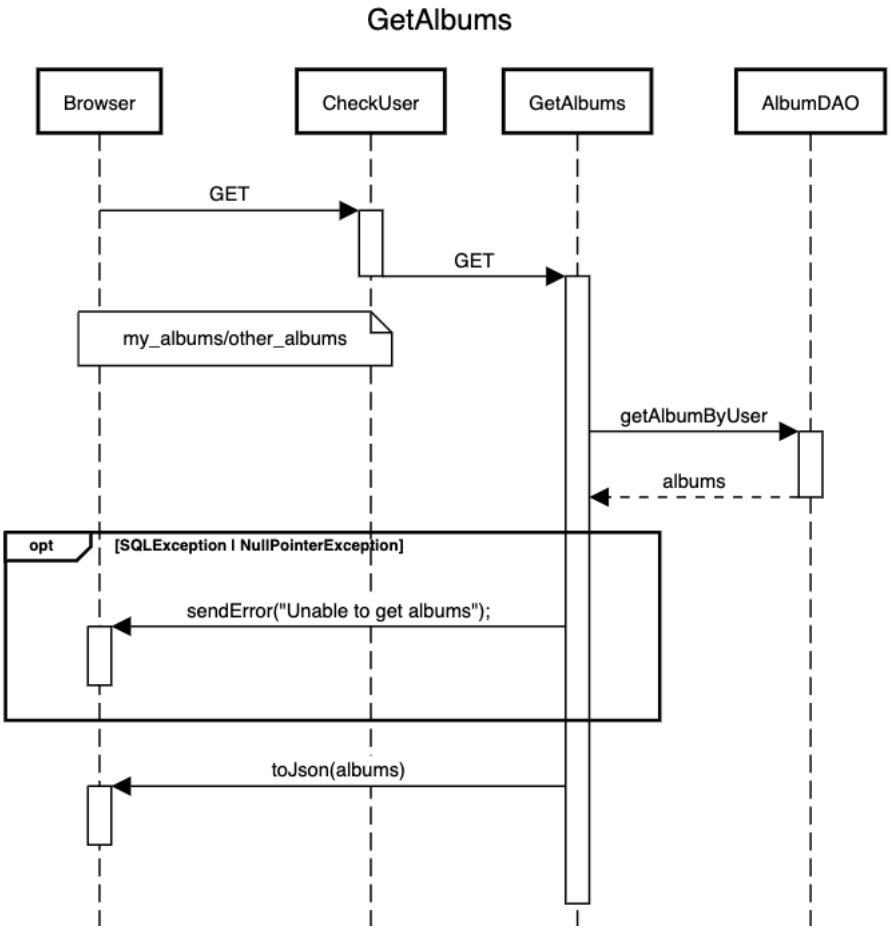
3.6.1 Update Order



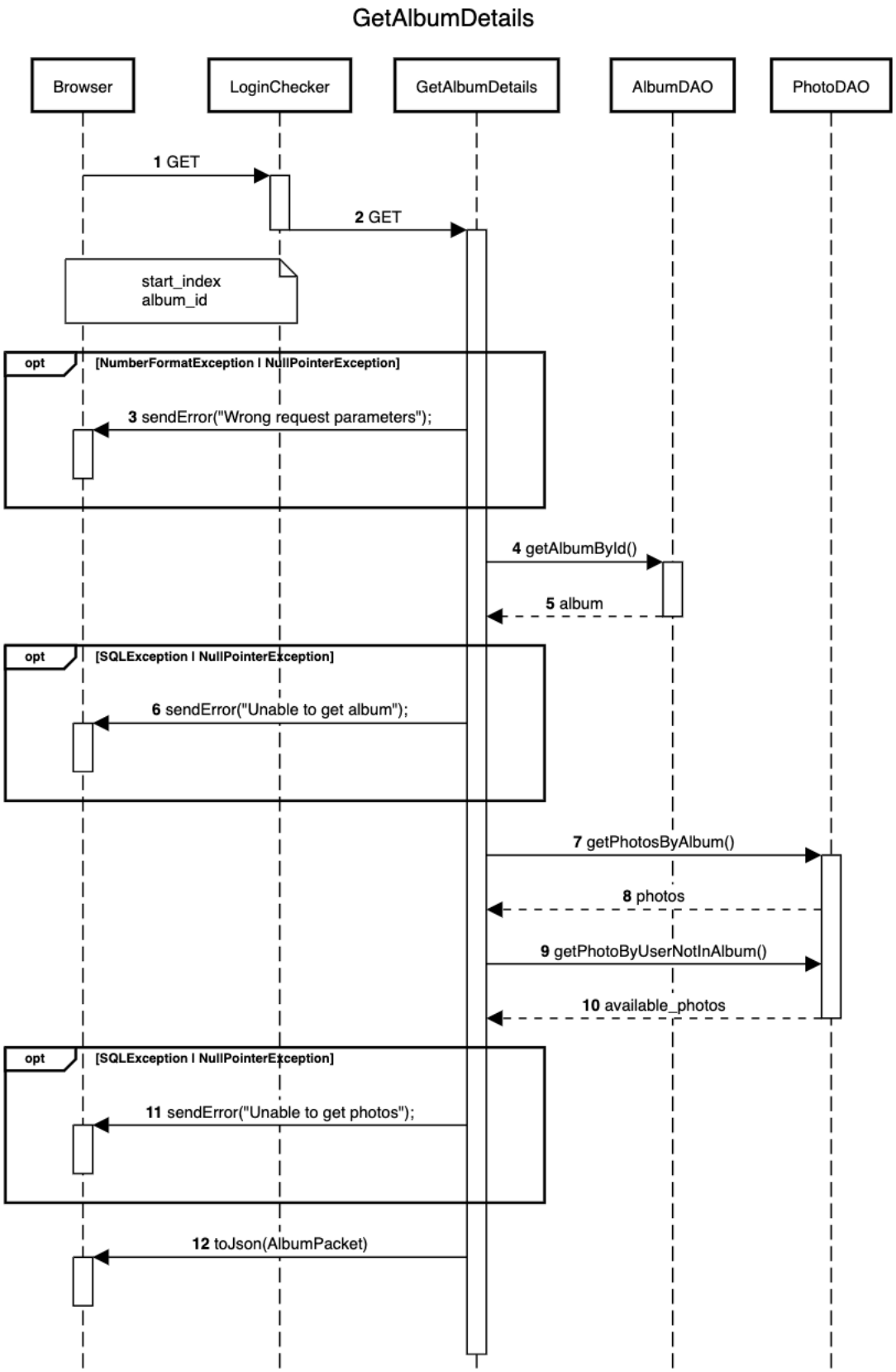
3.6.2 Get Photo



3.6.3 Get Albums



3.6.4 Get Album Details



3.7 Azioni e eventi

Client		Server	
Evento	**Azione**	**Evento**	**Azione**
Submit form login	Check campi vuoti	POST (user, psw)	Check dati
Submit form signup	Check mail, psw e campi vuoti	POST (user, psw, mail)	Check dati
Load di homepage	Update view (albums lists)	GET	Ricezione e parsing JSON
Selezione di un album da homepage (my albums)	Update della view con i dettagli dell'album, le thumbnails e il form "add to this album"	GET (album id)	Ricezione e parsing JSON
Selezione di un album da homepage (other albums)	Update della view con i dettagli dell'album e le thumbnails	GET (album id)	Ricezione e parsing JSON
Digita 'INVIO' in un input	Il bottone di submit viene cliccato	-	-
Modifica dell'ordine delle righe di "My albums"	Mostra il bottone "update order"	-	-
Submit form create album	Check campi vuoti e update della view di my albums	POST (title)	Check dati
Submit form upload photo	Check campi vuoti e update della view di "available photos"	POST (title, alt, file)	Check dati
Submit form add to this album	Check campi vuoti e update view dell'album	POST (idalbum, idphoto)	Check dati
Click "update order"	Submit del nuovo ordine in JSON	POST (ordine JSON)	Check dati
Cursore su una thumbnail	Mostra finestra modale con i dettagli e commenti	-	-
Submit form add comment	Check campi vuoti e update view dei commenti	POST (testo)	Check dati
Click freccetta NEXT	Mostra le thumbnails successive	-	-
Click freccetta PREVIOUS	Mostra le thumbnails precedenti	-	-

3.8 Controllers e Event Handlers

Client		Server	
Evento	**Controllore**	**Evento**	**Controllore**
Submit form login	Funzione makeCall	POST (user, psw)	Login
Submit form signup	Funzione makeCall	POST (user, mail, psw, conferma psw)	Register
Load di homepage	PageOrchestrator.start [my albums list, other albums list]	GET	GetAlbums
Selezione di un album da homepage	AlbumList.reset	GET (album id)	GetAlbumDetails
Submit sul bottone "Update order"	Event listener anonimo	POST (ordine JSON)	UpdateOrder
Submit form create album	FormManager	POST (title)	CreateAlbum
Submit form upload photo	FormManager	POST (title, alt, file)	UploadPhoto
Submit form add to this album	FormManager	POST (idalbum, idphoto)	AddToAlbum
Cursore su una thumbnail	AlbumDetails	GET (idphoto)	GetPhoto
Submit form add comment	FormManager	POST (testo)	AddComment