Progetto TIW – RIA
Gestione di immagini

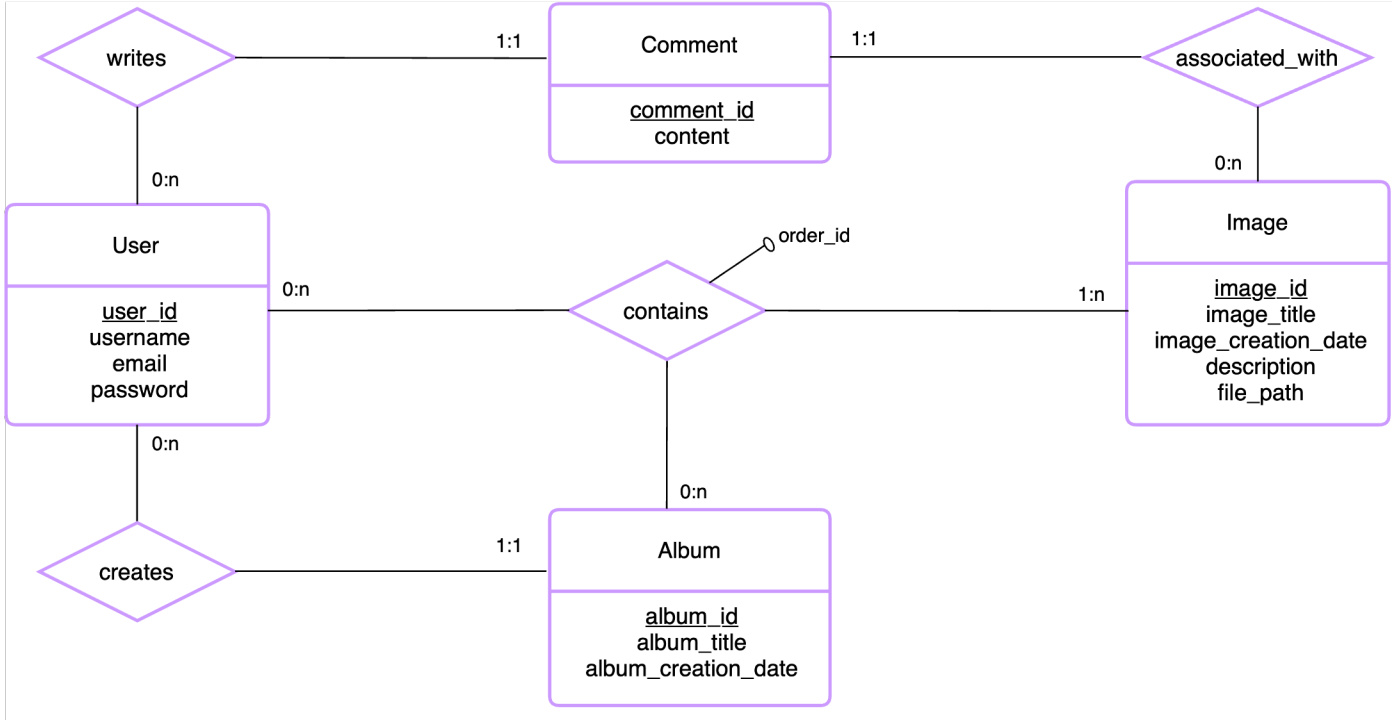
Kevin Ziroldi – Matteo Volpari

Analisi dei dati

Entities, **attributes**, **relationships**

Un'applicazione web consente la gestione di una galleria d'immagini. L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form. 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.
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.

Schema ER


writes
1:1
Comment
comment_id
content
1:1
associated_with
0:n
Image
image_id
image_title
image_creation_date
description
file_path
0:n
User
user_id
username
email
password
0:n
order_id
contains
1:n
0:n
creates
1:1
Album
album_id
album_title
album_creation_date

Schema logico

User(user_id, username, email, password)
Comment(comment_id, content, user_id, image_id)
Comment.user_id -> **User**.user_id
Comment.image_id -> **Image**.image_id
Image(image_id, image_title, image_creation_date, description, file_path)
Album(album_id, album_title, album_creation_date, user_id)
Album.user_id -> **User** -> user_id
Contains(user_id, album_id, image_id, order_id)
Contains.image_id -> **Image**.image_id
Contains.album_id -> **Album**.album_id
Contains.user_id -> **User**.user_id

Album

```
CREATE TABLE `Album` (
  `album_id` int NOT NULL AUTO_INCREMENT,
  `album_title` varchar(200) NOT NULL,
  `album_creation_date` date NOT NULL,
  `user_id` int NOT NULL,
  PRIMARY KEY (`album_id`),
  KEY `user_id_fk_idx` (`user_id`),
  CONSTRAINT `user_fk` FOREIGN KEY (`user_id`) REFERENCES `User`
(`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

Comment

```
CREATE TABLE `Comment` (
  `comment_id` int NOT NULL AUTO_INCREMENT,
  `content` varchar(500) NOT NULL,
  `user_id` int NOT NULL,
  `image_id` int NOT NULL,
  PRIMARY KEY (`comment_id`),
  KEY `user_id_fk_idx` (`user_id`),
  KEY `image_fk_idx` (`image_id`),
  CONSTRAINT `image_fk` FOREIGN KEY (`image_id`) REFERENCES `Image`
(`image_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `user_id_fk` FOREIGN KEY (`user_id`) REFERENCES `User`
(`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

Contains

```
CREATE TABLE `Contains` (
  `image_id` int NOT NULL,
  `album_id` int NOT NULL,
  `order_id` int NOT NULL,
  `user_id` int NOT NULL,
  PRIMARY KEY (`image_id`,`album_id`,`user_id`),
  KEY `album_id_fk_idx` (`album_id`),
  KEY `user_id_fk_idx` (`user_id`),
  CONSTRAINT `album_id_fk` FOREIGN KEY (`album_id`) REFERENCES `Album`
(`album_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `image_id_fk` FOREIGN KEY (`image_id`) REFERENCES `Image`
(`image_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `user_id_fk_contains` FOREIGN KEY (`user_id`) REFERENCES
`User` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

Image

```
CREATE TABLE `Image` (
  `image_id` int NOT NULL AUTO_INCREMENT,
  `image_title` varchar(45) NOT NULL,
  `image_creation_date` date NOT NULL,
  `description` varchar(500) NOT NULL,
  `file_path` varchar(260) NOT NULL,
  PRIMARY KEY (`image_id`)
)
```

User

```
CREATE TABLE `User` (
  `user_id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `email` varchar(100) NOT NULL,
  `password` varchar(30) NOT NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE KEY `username` (`username`),
  UNIQUE KEY `email` (`email`)
)
```

Analisi requisiti

Pages (views), view components, events, actions

Un'applicazione web consente la gestione di una galleria d'immagini. L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form. 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.

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.
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 il numero di immagini non è un multiplo di 5 la tabella deve avere sempre 5 celle, lasciando quelle più a destra vuote. Le miniature sono ordinate da sinistra a destra per data decrescente. Se l'album contiene più di cinque immagini, sono disponibili comandi per vedere il precedente e successivo insieme di cinque immagini. Se la pagina ALBUM PAGE mostra il primo blocco d'immagini e ne esistono altre successive nell'ordinamento, compare a destra della riga il bottone SUCCESSIVE, che permette di vedere le successive cinque immagini. Se la pagina ALBUM PAGE mostra l'ultimo blocco d'immagini e ne esistono altre precedenti nell'ordinamento, compare a sinistra della riga il bottone PRECEDENTI, che permette di vedere le cinque immagini precedenti. Se la pagina ALBUM PAGE mostra un blocco d'immagini e ne esistono altre precedenti e successive nell'ordinamento, compare a destra della riga il bottone SUCCESSIVE, che permette di vedere le successive cinque immagini, e a sinistra il bottone PRECEDENTI, che permette di vedere le cinque immagini precedenti.

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. Il bottone di cancellazione appare solo se l’immagine appartiene all’utente. L’invio del commento con un bottone INVIA ripresenta la pagina IMAGE PAGE, con tutti i dati aggiornati della stessa immagine.

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.

Requisiti versione RIA

- L'applicazione supporta registrazione e login mediante una pagina pubblica con opportune form. La registrazione richiede l'inserimento di username, indirizzo di email e password e controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i campi "password" e "ripeti password", anche a lato client. La registrazione controlla l'unicità dello username.
- Dopo il login dell'utente, l'intera applicazione è realizzata con un'unica pagina.
- Ogni interazione dell'utente è gestita senza ricaricare completamente la pagina, ma produce l'invocazione asincrona del server e l'eventuale modifica del contenuto da aggiornare a seguito dell'evento.
- 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.
- Quando l'utente passa con il mouse su una miniatura, l'applicazione mostra una finestra modale con tutte le informazioni dell'immagine, tra cui la stessa a grandezza naturale, i commenti eventualmente presenti e la form per inserire un commento.

- L’applicazione controlla anche a lato client che non si invii un commento vuoto.
- Errori a lato server devono essere segnalati mediante un messaggio di allerta all’interno della pagina.
- 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. Ai successivi accessi, l’ordinamento personalizzato, per quell’utente, è usato al posto di quello di default.

Completamento delle specifiche

- Tutti i campi delle form sono obbligatori.
- Se un utente sta compilando una form e preme il tasto invio, l'applicazione simula un click per l'invio della form in maniera asincrona.
- La pagina di default è quella che contiene la form di login e di signup.
- Dopo la registrazione, l'utente accede direttamente alla home.
- Un utente può loggarsi usando email e password oppure usando username e password.
- Se l'utente non è loggato e tenta di accedere alla home page, viene reindirizzato alla pagina di login.
- Un utente non può creare più album con lo stesso nome.
- Utenti diversi possono avere album con stesso nome.
- Al momento della creazione, un album può essere vuoto. È possibile aggiungere immagini dal proprio file system all'album dalla home page. È possibile aggiungere immagini già caricate in altri album ad nuovo album dalla image page.
- Un utente può inserire, tra le immagini già presenti sul server, solamente le sue immagini.
- Se l'immagine appartiene a più album, al momento della cancellazione, viene cancellata solo da un album.

Server-side components

- **Model objects (Beans)**
  - Album
  - Comment
  - Image
  - User

- **Data Access Objects (DAO)**
  - AlbumDAO
    - findAlbumsByUserSorted(int userId)
    - findAlbumsOfOthersSorted(int userId)
    - findAlbumById(int albumId)
    - createAlbum(String albumTitle, int userId)
    - getUserIdByAlbumId(int albumId)
    - getAlbumIds()
  - CommentDAO
    - findCommentsByImageId(int imageId)
    - addComment(String content, int userId, int imageId)
    - getUsernameByCommentId(int commentId)
  - ContainsDAO
    - changeImagesOrder(int userId, int albumId, List<Integer> imageIds)
  - ImageDAO
    - addNewImageToAlbum(String title, String description, String imagePath, int albumId)
    - findImageById(int imageId)
    - findImagesForAlbumDefaultSort(int albumId)
    - findImagesForAlbumActualSort(int albumId, int userId)
    - deleteImageFromAlbum(int albumId, int imageId)
    - getUserIdByImageId(int imageId)
    - addExistingImageToAlbum(int albumId, int imageId)
    - albumContainsImage(int albumId, int imageId)
  - UserDAO
    - checkCredentials(String usr, String pwd)
    - getUserByUsername(String username)
    - getUserByEmail(String email)
    - registerUser(String username, String email, String password)

Server-side components

- **Controllers (servlets)**
  - AddCommentToImage
  - AddExistingImageToAlbum
  - AddNewImageToAlbum
  - ChangeImagesOrder
  - CheckLogin
  - CheckSignup
  - CreateAlbum
  - DeleteImageFromAlbum
  - GetAlbumImagesData
  - GetAlbumsData
  - GetImageData
  - Logout

- **Filters**
  - CheckUserLogged
  - NoCacher

- **Utils**
  - ConnectionHandler
    - getConnection
    - closeConnection
  - AlbumOrder

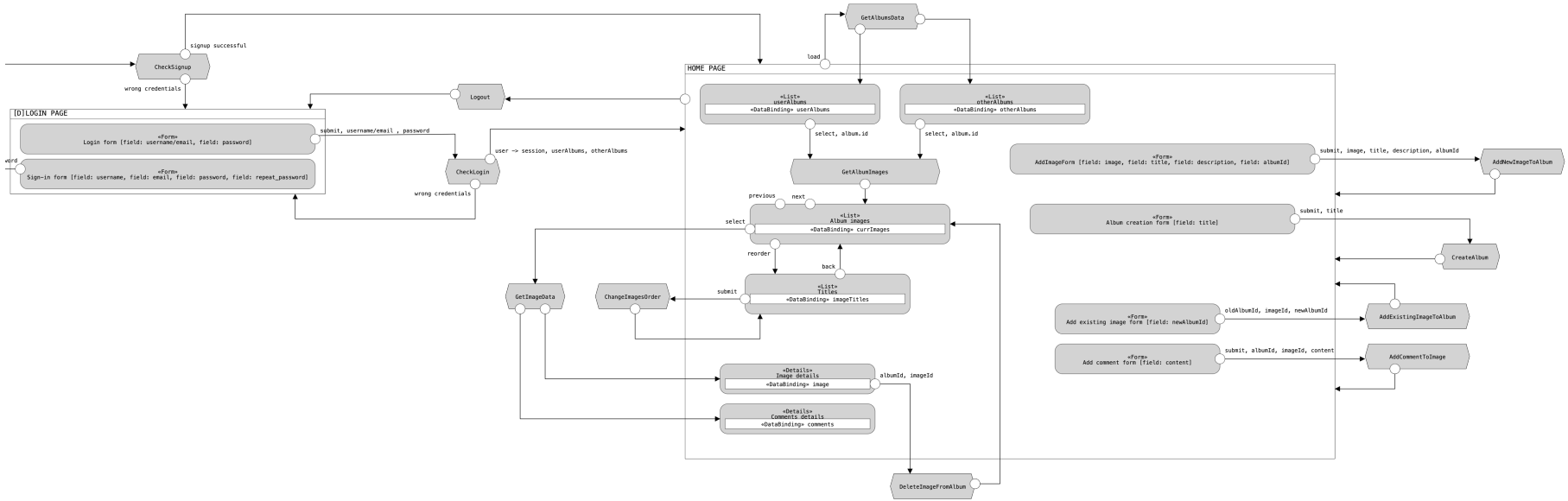
Client-side components

- **Login**
  - Login form
    - Gestione login ed errori
  - Signup form
    - Gestione registrazione ed errori
- **Home**
  - AlertModalWindow
    - registerEvents: binding evento di chiusura della finestra modale
    - reset: imposta la finestra modale come non visibile
    - show: mostra il messaggio di errore, rende la finestra modale visibile
  - AlbumList
    - registerEvents: binding evento di click sui bottoni delle form
    - reset: imposta le liste e i form come non visibili
    - show: richiede al server i dati degli album
    - update: mostra le liste di album e le form
    - buildAlbumRow: crea e aggiunge una riga per un album alla lista
  - AlbumImages
    - registerEvents: binding eventi di immagini precedent e successive, torna alla home e riordina immagini
    - reset: imposta le immagini e i bottoni come non visibili
    - show: richiede al server i dati delle immagini di un album
    - update: mostra la tabella con le prime 5 immagini

Client-side components

- ImageModalWindow
  - registerEvents: binding eventi di cancellazione immagine, aggiunta immagine ad altro album e aggiunta commento
  - reset: imposta l'immagine, i dettagli e le form come non visibili
  - show: richiede al server i dati dell'immagine
  - update: mostra l'immagine, le sue informazioni, i commenti, bottoni e form
- SortImages
  - registerEvents: binding evento di salvataggio del nuovo ordinamento sul server
  - reset: imposta la lista di titoli delle immagini come non visibile
  - update: mostra una tabella contente la lista di titoli delle immagini
  - buildImageRow: costruisce una riga per l'immagine
  - titleDragStart: quando il trascinamento inizia, salva l'elemento trascinato
  - titleDragOver: al passaggio su un altro titolo, modifica la sua classe CSS
  - titleDragLeave: quando termina il passaggio su un altro titolo, modifica la sua classe CSS
  - titleDragDrop: quando l'elemento trascinato viene rilasciato, modifica l'ordinamento
  - unselectRows: ripristina la classe CSS di tutte le righe

Application design

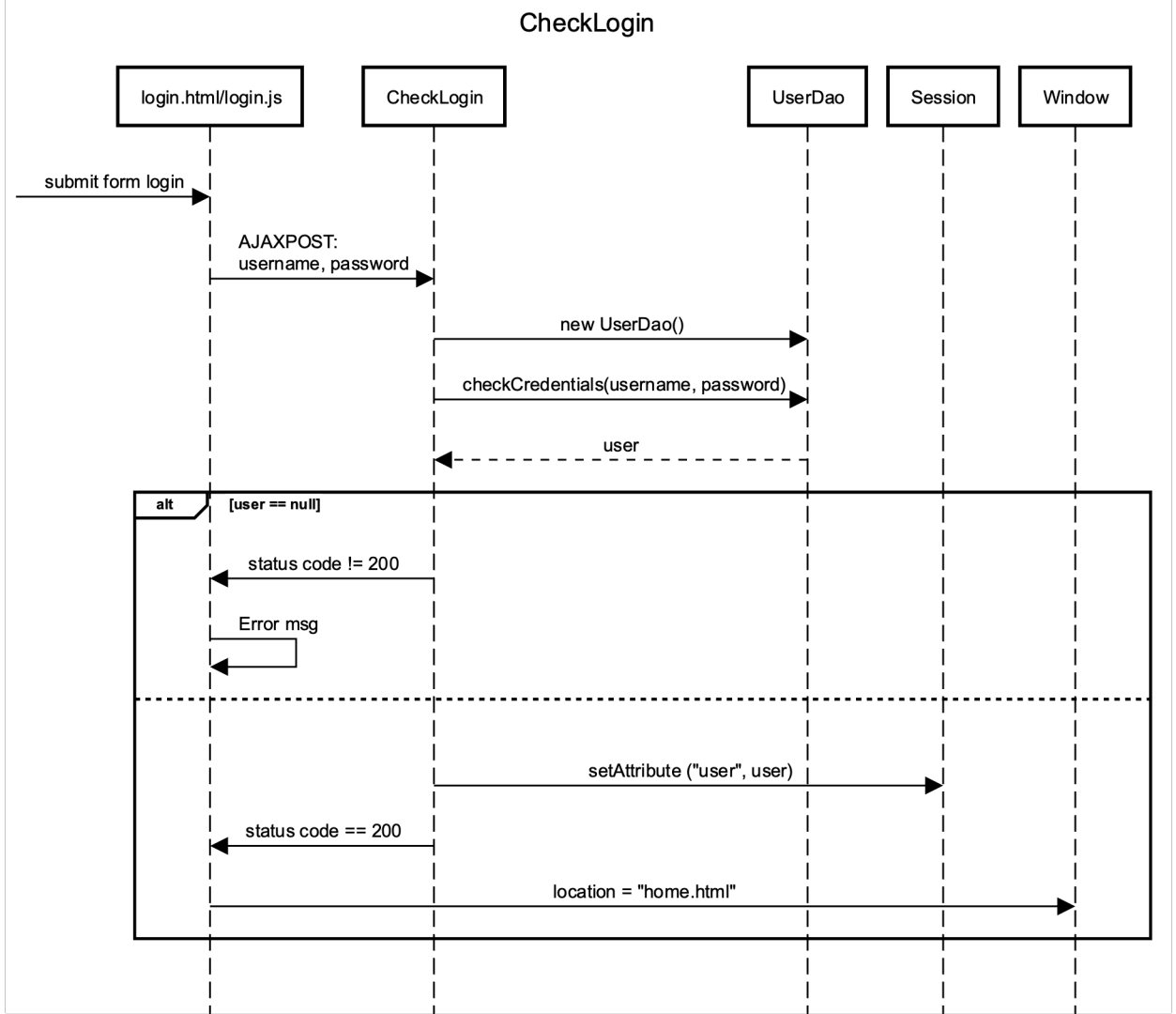


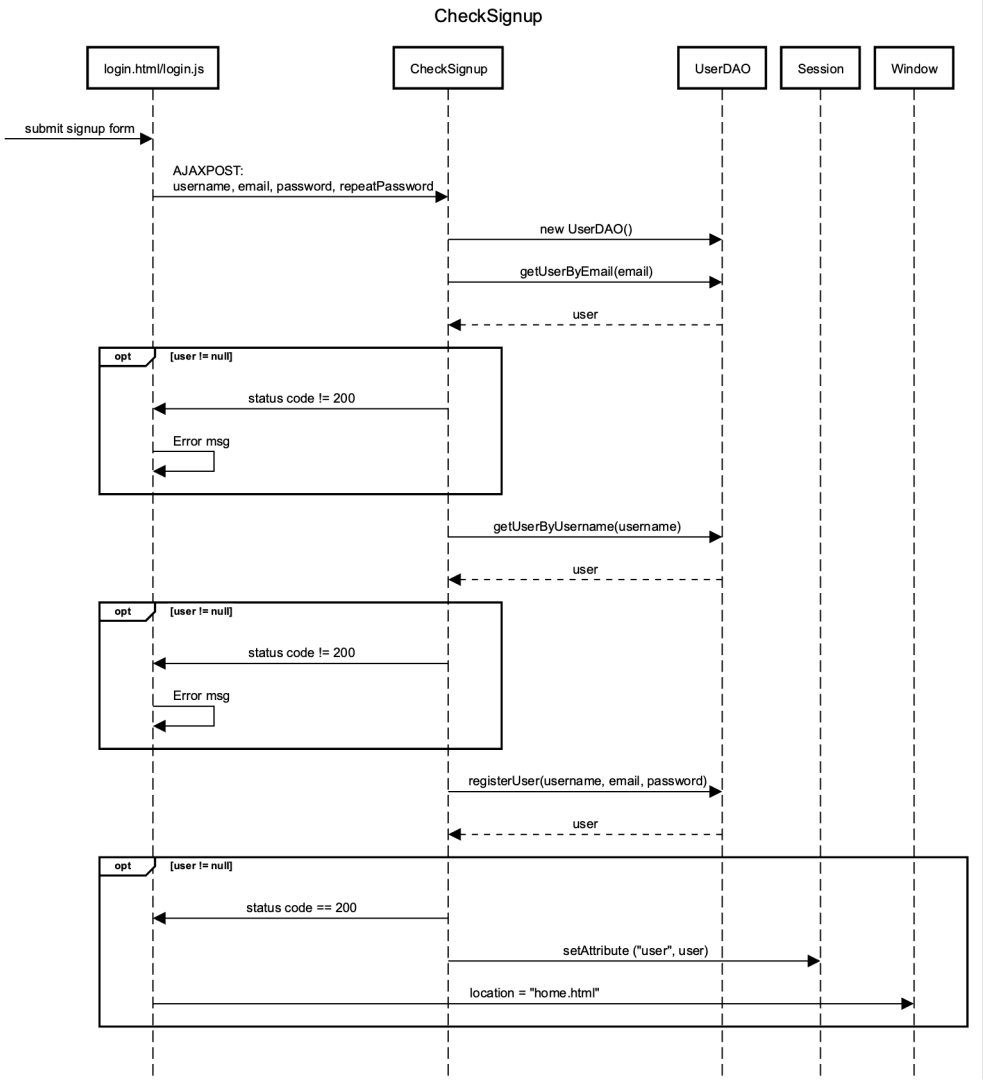
Event-Action table

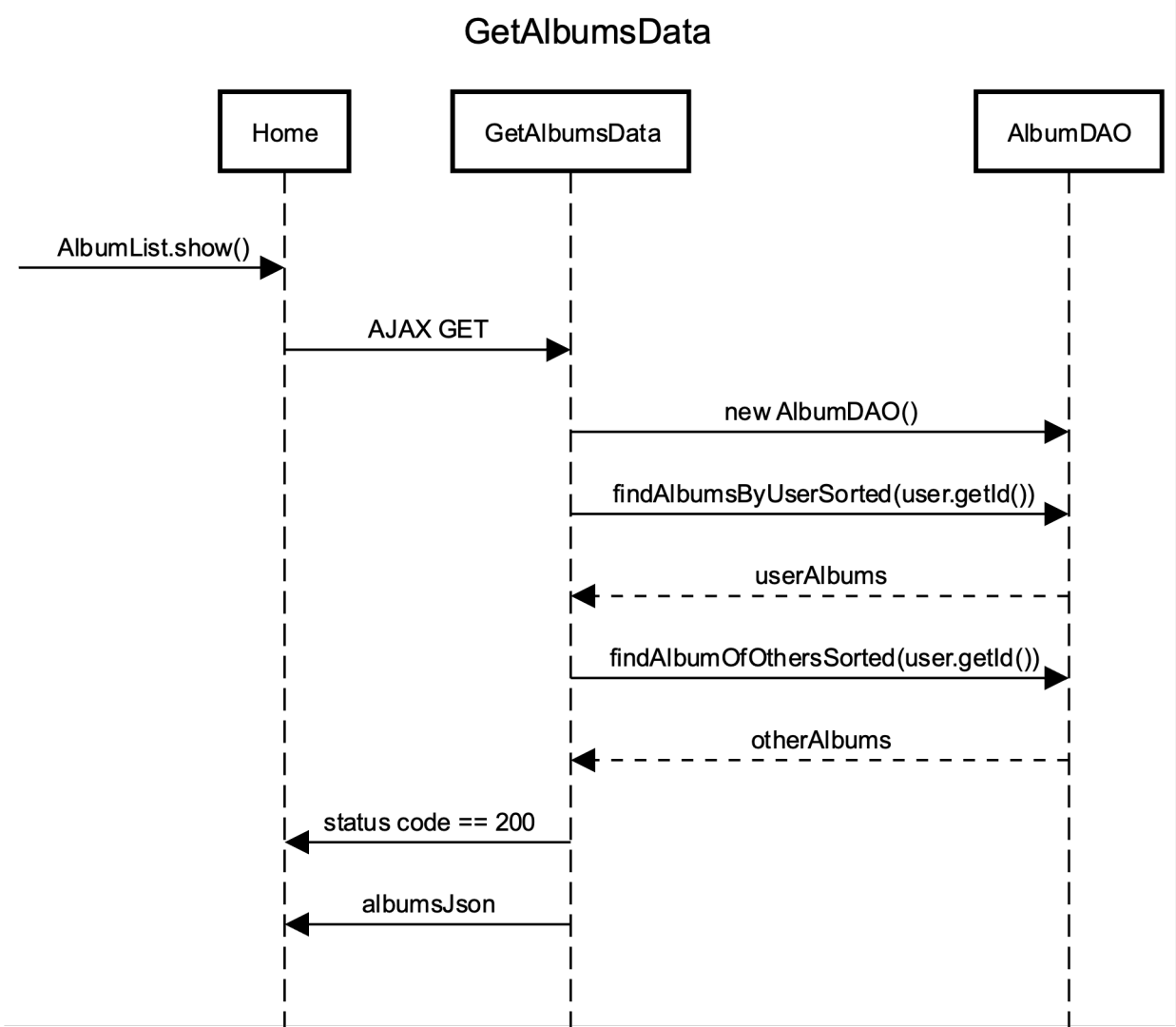
	Client-side		Server-side	
N°	Evento	Azione	Evento	Azione
1	Login-> Login form-> submit	Login	POST (username, password)	Controllo credenziali
2	Login->Sign-in form-> submit	SignUp	POST (username, email, password, repeatPassword)	Controllo unicità credenziali e uguaglianza campi password e repeatPassword
3	Home->load	Visualizza la pagina home	POST (user, userAlbums, otherAlbums)	Estrae gli album dell'utente e degli altri utenti e carica la pagina home
4	Home -> Album creation form -> submit	Aggiorna lista dei suoi album, EVENTO 3	POST (title)	Aggiunge l'album e ricarica la pagina home
5	Home -> userAlbums/otherAlbums -> select	Mostra le prime cinque immagini dell'album selezionato	GET (albumId)	Estrae tutte le immagini dall'album selezionato
6	Home -> AddImageForm -> submit	-	POST (albumId, title, description, image)	Aggiunge l'immagine all'album selezionato
7	Home -> AlbumImages -> pulsante next	Visualizza le cinque immagini successive	-	-
8	Home -> AlbumImages -> pulsante previous	Visulaizza le cinque immagini precedenti	-	-
9	Home -> pulsante reorder	Apre una finestra modale per la possibilità di riordinare le immagini	-	-
10	Home -> Titles -> submit	Cambia l'ordine delle immagini e mostra le AlbumImages	POST (albumId, imageIds)	Cambia l'ordine salvato precedentemente con quello impostato dall'utente
11	Home -> AlbumImages -> select	Mostra la finestra modale contenente l'immagine e i dati dell'immagine	GET (imageId, albumId)	Estrae i dati e i commenti dell'immagine selezionata
12	Home -> ImageDetails -> pulsante delete	Elimina l'immagine dall'album e mostra altre 5 immagini di quell'album	GET (albumId, imageId)	Elimina l'immagine dall'album e, se non è presente in nessun album, anche dal db
13	Home -> addExistingImageForm -> submit	Controllo dati	POST (oldAlbumId, newAlbumId, imageId)	Aggiunge l'immagine al nuovo
14	Home -> AddCommentForm -> submit	Ricarica imageDetails con il nuovo commento	POST (albumId, imageId, content)	Aggiunge il commento all'immagine
15	Home -> pulsante Logout	-	GET()	Termina la sessione

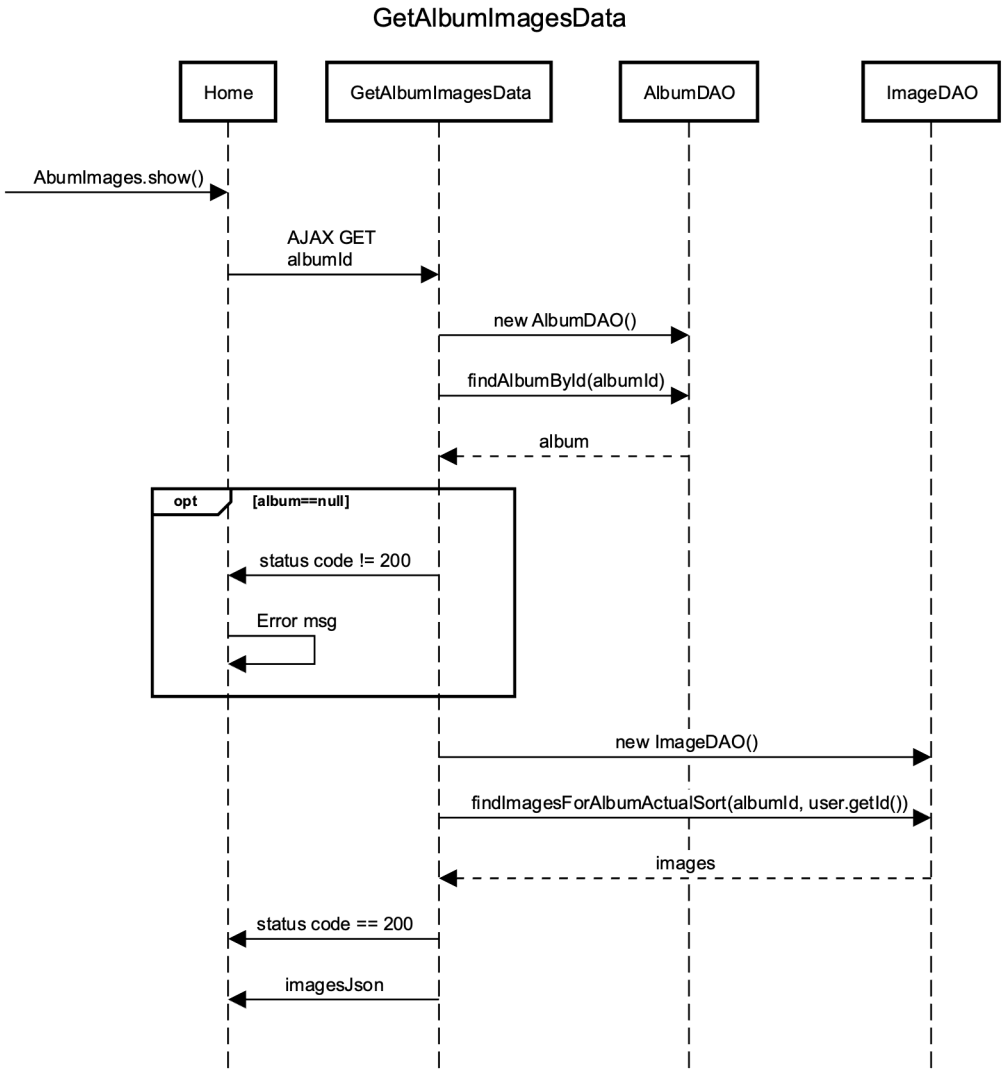
Event-Controllers table

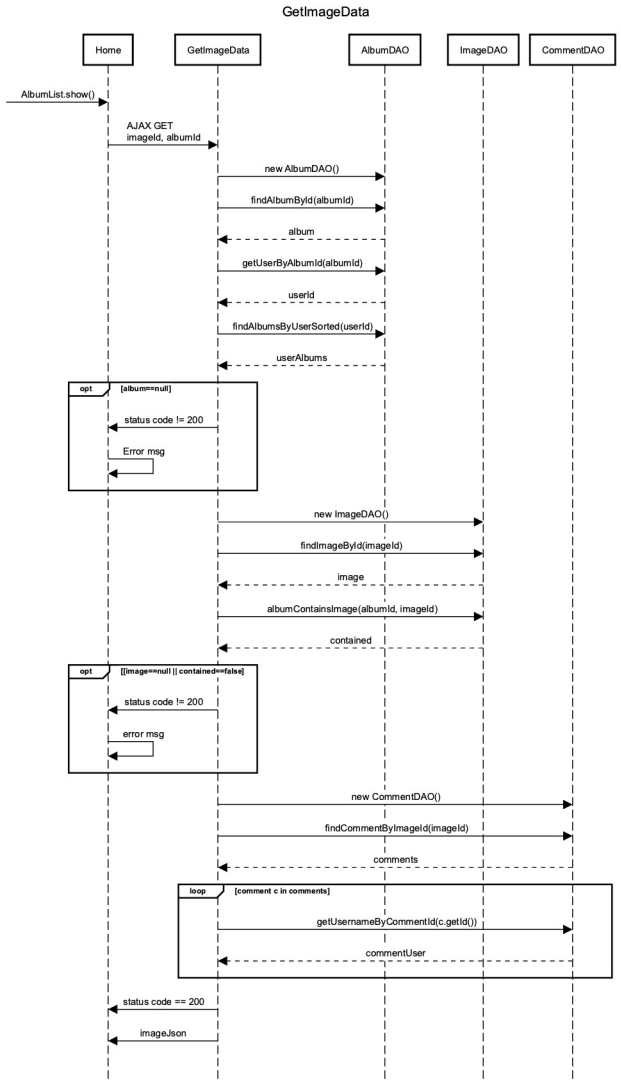
	Client-side		Server-side	
N°	Evento	Controllore	Evento	Controllore
1	Login-> Login form-> submit	SendToServer makeCall	POST (username, password)	CheckLogin
2	Login->Sign-in form-> submit	SendToServer makeCall	POST (username, email, password, repeatPassword)	CheckSignup
3	Home->load	makeCall	POST (user, userAlbums, otherAlbums)	GetAlbumsData
4	Home -> Album creation form -> submit	makeCall	POST (title)	CreateAlbum
5	Home -> userAlbums/otherAlbums -> select	makeCall	GET (albumId)	GetAlbumImagesData
6	Home -> AddImageForm -> submit	makeCall	POST (albumId, title, description, image)	AddNewImageToAlbum
7	Home -> AlbumImages -> pulsante next	AlbumImages.update	-	-
8	Home -> AlbumImages -> pulsante previous	AlbumImages.update	-	-
9	Home -> pulsante reorder	SortImages.update	-	-
10	Home -> Titles -> submit	makeCall	POST (albumId, imageIds)	ChangeImagesOrder
11	Home -> AlbumImages -> select	makeCall	GET (imageId, albumId)	GetImageData
12	Home -> ImageDetails -> pulsante delete	makeCall	GET (albumId, imageId)	DeleteImageFromAlbum
13	Home -> addExistingImageForm -> submit	makeCall	POST (oldAlbumId, newAlbumId, imageId)	AddExistingImageToAlbum
14	Home -> AddCommentForm -> submit	makeCall	POST (albumId, imageId, content)	AddCommentToImage
15	Home -> pulsante Logout	-	GET	Logout

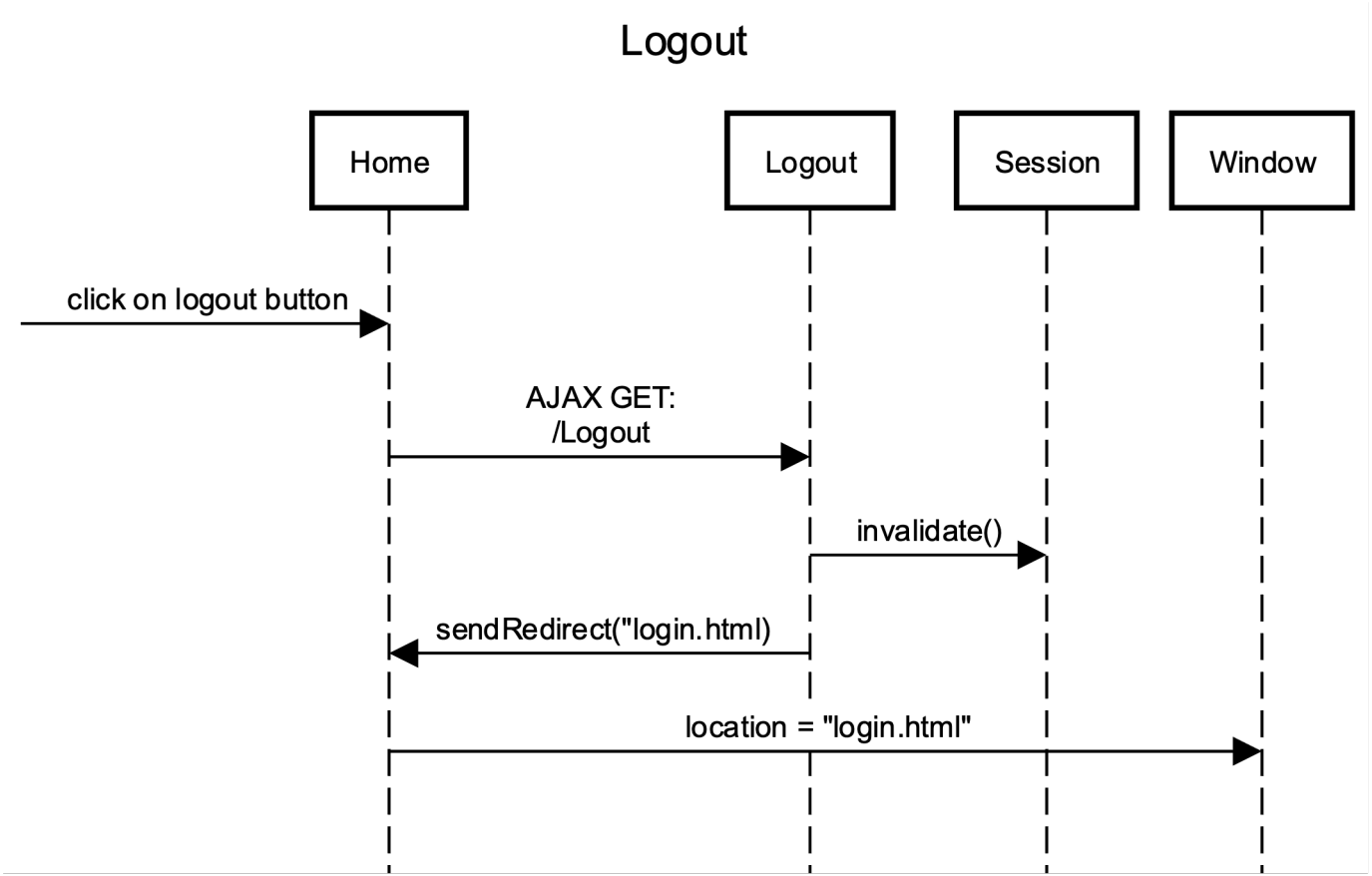

CheckLogin
login.html/login.js
CheckLogin
UserDao
Session
Window
submit form login
AJAXPOST:
username, password
new UserDao()
checkCredentials(username, password)
user
alt
[user == null]
status code != 200
Error msg
setAttribute ("user", user)
status code == 200
location = "home.html"

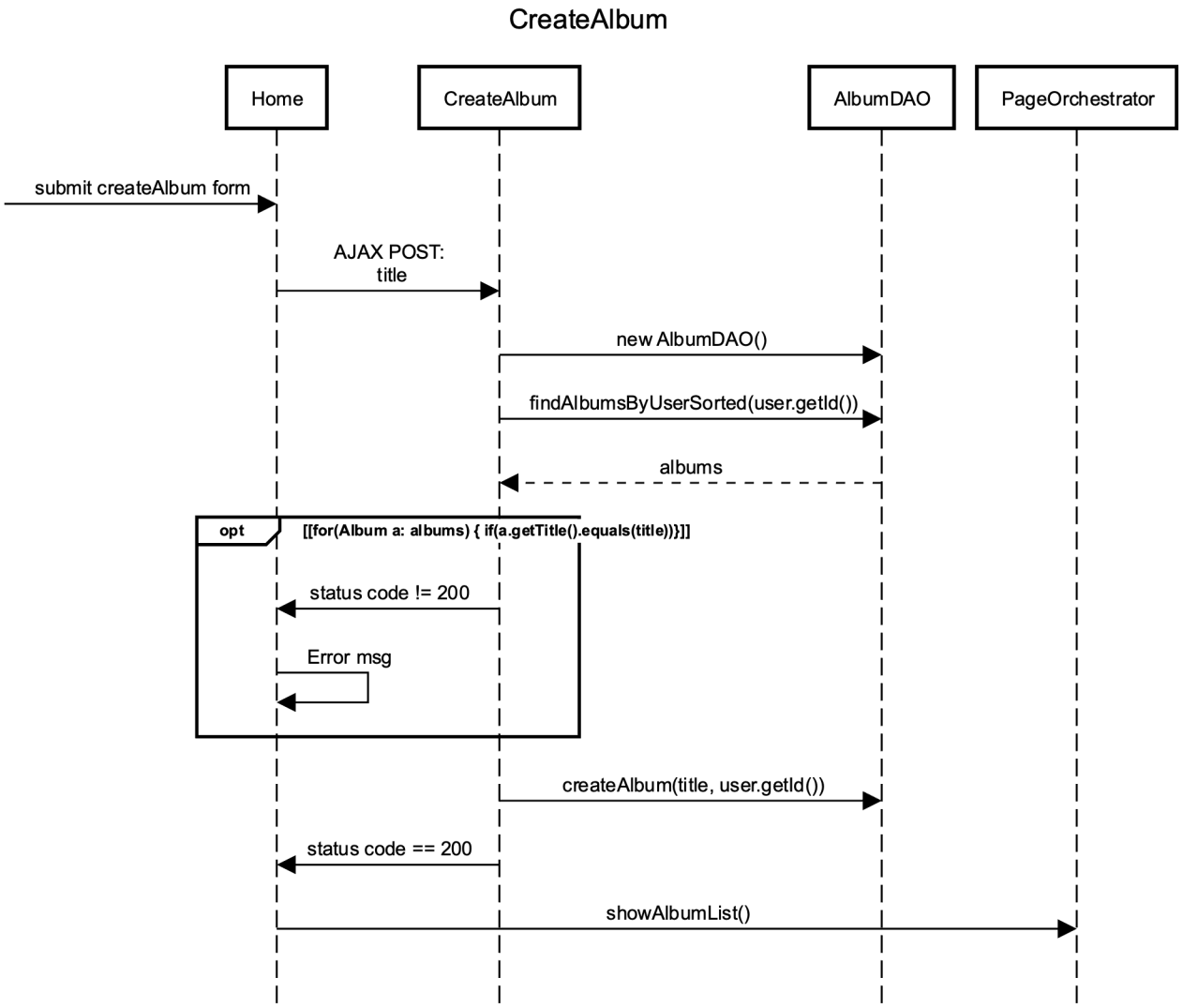

CheckSignup
login.html/login.js
CheckSignup
UserDAO
Session
Window
submit signup form
AJAXPOST:
username, email, password, repeatPassword
new UserDAO()
getUserByEmail(email)
user
opt
[user != null]
status code != 200
Error msg
getUserByUsername(username)
user
opt
[user != null]
status code != 200
Error msg
registerUser(username, email, password)
user
opt
[user != null]
status code == 200
setAttribute ("user", user)
location = "home.html"


GetAlbumsData
Home
GetAlbumsData
AlbumDAO
AlbumList.show()
AJAX GET
new AlbumDAO()
findAlbumsByUserSorted(user.getId())
userAlbums
findAlbumOfOthersSorted(user.getId())
otherAlbums
status code == 200
albumsJson

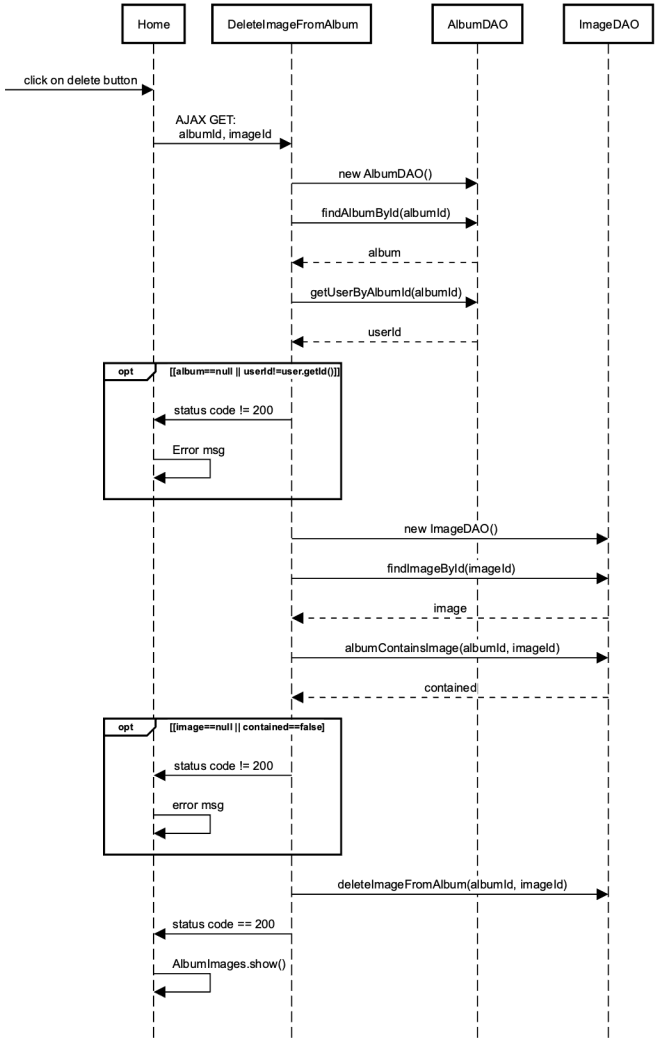

GetAlbumImagesData
Home
GetAlbumImagesData
AlbumDAO
ImageDAO
AbumImages.show()
AJAX GET
albumId
new AlbumDAO()
findAlbumById(albumId)
album
opt
[album==null]
status code != 200
Error msg
new ImageDAO()
findImagesForAlbumActualSort(albumId, user.getId())
images
status code == 200
imagesJson

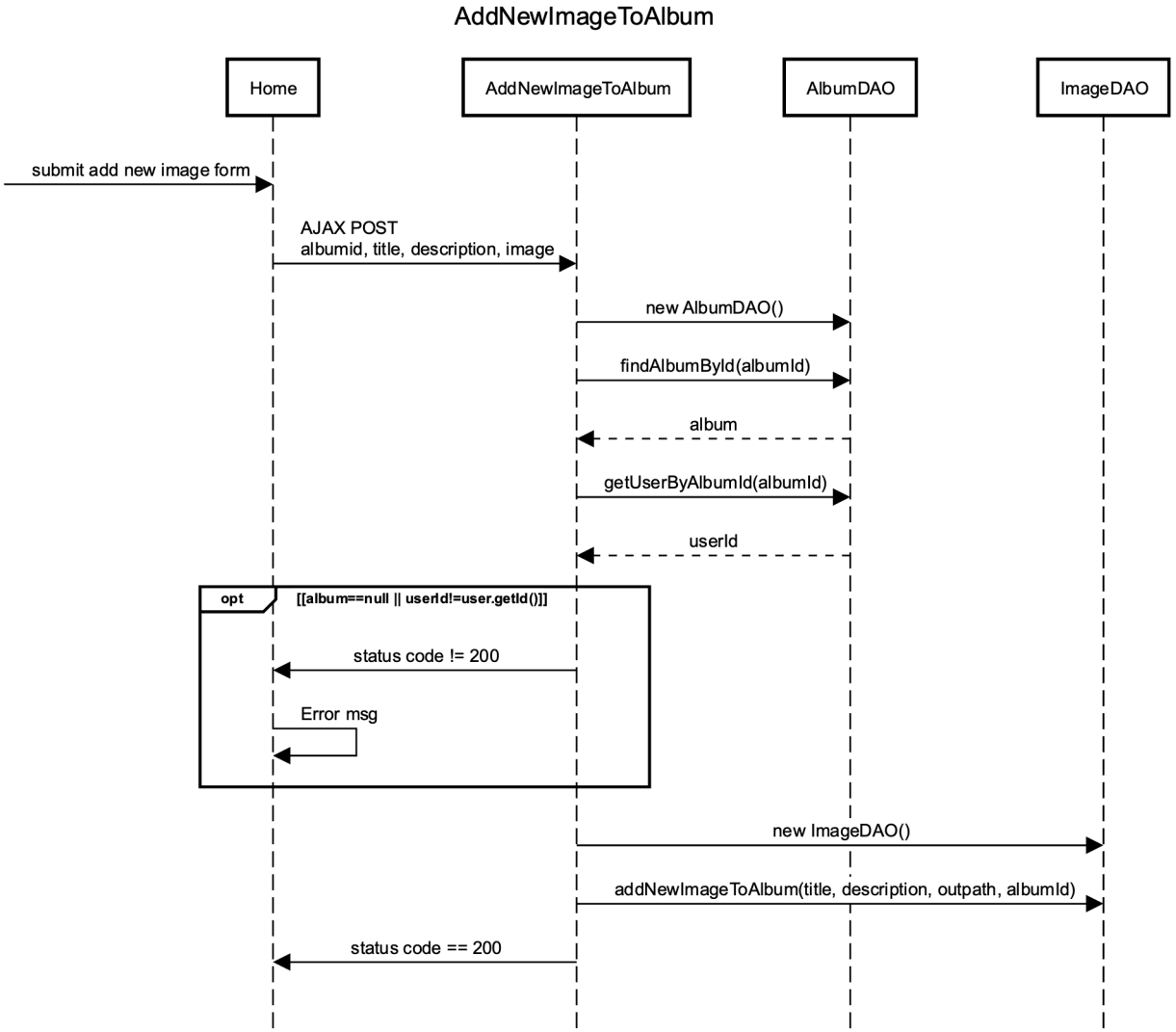

GetImageData
Home
GetImageData
AlbumDAO
ImageDAO
CommentDAO
AlbumList.show()
AJAX GET
imageId, albumId
new AlbumDAO()
findAlbumById(albumId)
album
getUserByAlbumId(albumId)
userId
findAlbumsByUserSorted(userId)
userAlbums
opt
[album==null]
status code != 200
Error msg
new ImageDAO()
findImageById(imageId)
image
albumContainsImage(albumId, imageId)
contained
opt
[[image==null || contained==false]
status code != 200
error msg
new CommentDAO()
findCommentByImageId(imageId)
comments
loop
[comment c in comments]
getUsernameByCommentId(c.getId())
commentUser
status code == 200
imageJson

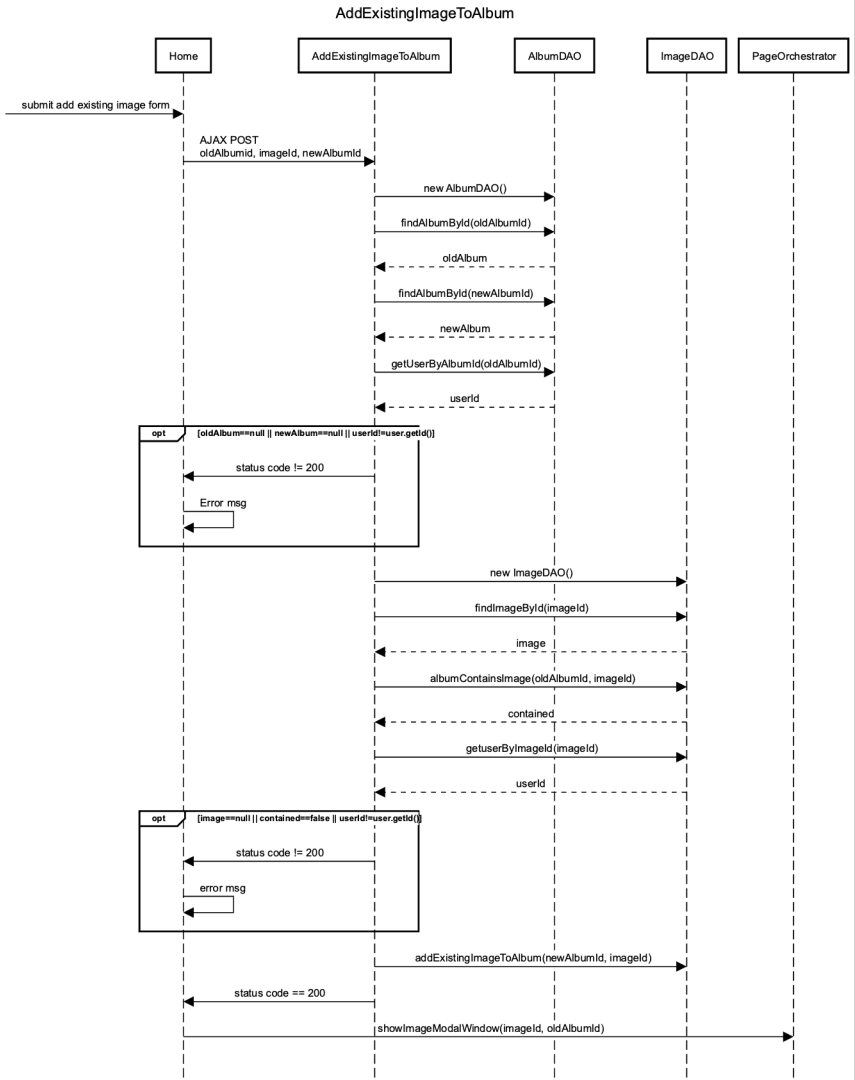

Logout
Home
Logout
Session
Window
click on logout button
AJAX GET:
/Logout
invalidate()
sendRedirect("login.html)
location = "login.html"

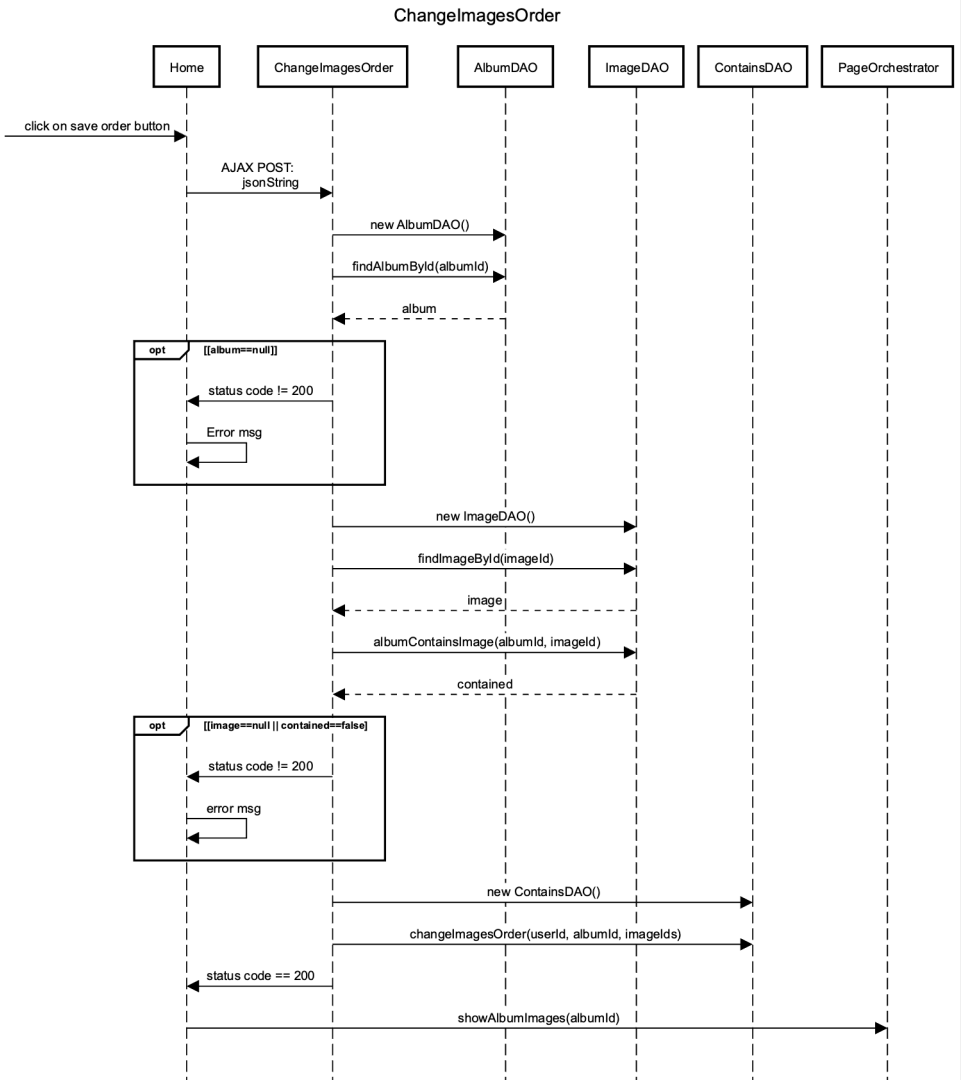

CreateAlbum
Home
CreateAlbum
AlbumDAO
PageOrchestrator
submit createAlbum form
AJAX POST:
title
new AlbumDAO()
findAlbumsByUserSorted(user.getId())
albums
opt
[[for(Album a: albums) { if(a.getTitle().equals(title))}]]
status code != 200
Error msg
createAlbum(title, user.getId())
status code == 200
showAlbumList()

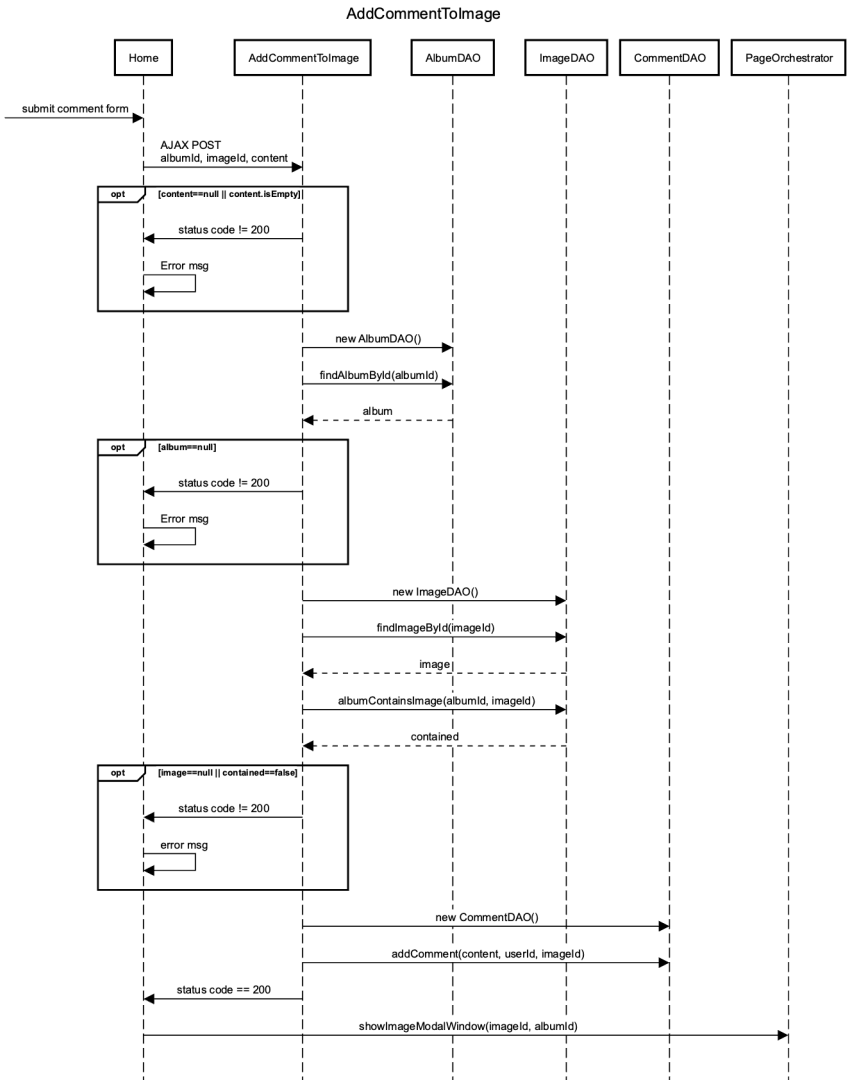
DeleteImageFromAlbum




AddNewImageToAlbum
Home
AddNewImageToAlbum
AlbumDAO
ImageDAO
submit add new image form
AJAX POST
albumid, title, description, image
new AlbumDAO()
findAlbumById(albumId)
album
getUserByAlbumId(albumId)
userId
opt
[[album==null || userId!=user.getId()]]
status code != 200
Error msg
new ImageDAO()
addNewImageToAlbum(title, description, outpath, albumId)
status code == 200


AddExistingImageToAlbum
Home
AddExistingImageToAlbum
AlbumDAO
ImageDAO
PageOrchestrator
submit add existing image form
AJAX POST
oldAlbumid, imageId, newAlbumId
new AlbumDAO()
findAlbumById(oldAlbumId)
oldAlbum
findAlbumById(newAlbumId)
newAlbum
getUserByAlbumId(oldAlbumId)
userId
opt
[oldAlbum==null || newAlbum==null || userId!=user.getId()]
status code != 200
Error msg
new ImageDAO()
findImageById(imageId)
image
albumContainsImage(oldAlbumId, imageId)
contained
getuserByImageId(imageId)
userId
opt
[image==null || contained==false || userId!=user.getId()]
status code != 200
error msg
addExistingImageToAlbum(newAlbumId, imageId)
status code == 200
showImageModalWindow(imageId, oldAlbumId)


ChangeImagesOrder
Home
ChangeImagesOrder
AlbumDAO
ImageDAO
ContainsDAO
PageOrchestrator
click on save order button
AJAX POST:
jsonString
new AlbumDAO()
findAlbumById(albumId)
album
opt
[[album==null]]
status code != 200
Error msg
new ImageDAO()
findImageById(imageId)
image
albumContainsImage(albumId, imageId)
contained
opt
[[image==null || contained==false]
status code != 200
error msg
new ContainsDAO()
changeImagesOrder(userId, albumId, imageIds)
status code == 200
showAlbumImages(albumId)


AddCommentToImage
Home
AddCommentToImage
AlbumDAO
ImageDAO
CommentDAO
PageOrchestrator
submit comment form
AJAX POST
albumId, imageId, content
opt
[content==null || content.isEmpty]
status code != 200
Error msg
new AlbumDAO()
findAlbumById(albumId)
album
opt
[album==null]
status code != 200
Error msg
new ImageDAO()
findImageById(imageId)
image
albumContainsImage(albumId, imageId)
contained
opt
[image==null || contained==false]
status code != 200
error msg
new CommentDAO()
addComment(content, userId, imageId)
status code == 200
showImageModalWindow(imageId, albumId)