# Esercizio 1:
# Galleria Immagini

Andrea Lampis

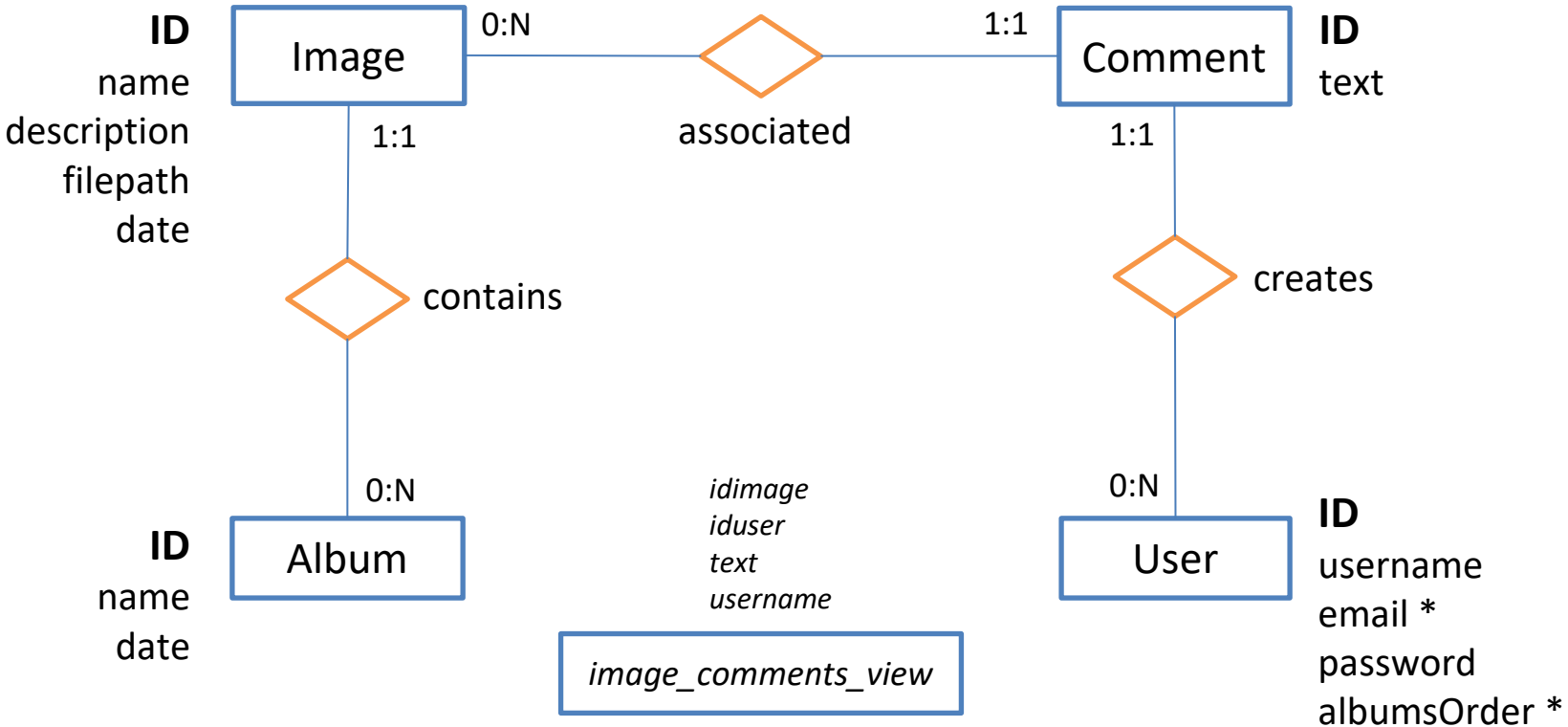
# Analisi dei dati

Un’applicazione web consente la gestione di una galleria d’immagini. Ogni immagine è memorizzata nella base di dati mediante un titolo, una data, un testo descrittivo e il percorso del file dell’immagine nel file system del server su cui l’applicazione è rilasciata. Le immagini sono raggruppate in album e sono associate a uno o più commenti. Un commento ha un testo e il nome dell’utente che lo ha creato. Un album ha un titolo e una data di creazione.

Completamento specifica: Gli utenti sono salvati nel database mediante un username e una password

**Entities, attributes, relationships**

# Database design




* only for RIA version

# Local database schema

```
CREATE TABLE `user` (
`id` int NOT NULL AUTO_INCREMENT,
`username` varchar(50) NOT NULL,
`email` varchar(255) NOT NULL,
`password` varchar(45) NOT NULL,
`albumsOrder` varchar(45) DEFAULT NULL,
PRIMARY KEY (`id`)
UNIQUE KEY `username_UNIQUE` (`username`))
```

```
CREATE TABLE `album` (
`id` int NOT NULL AUTO_INCREMENT,
`name` varchar(200) NOT NULL,
`date` date NOT NULL,
PRIMARY KEY (`id`)
UNIQUE KEY `title_UNIQUE` (`name`))
```

# Local database schema

```
CREATE TABLE `image` (
`id` int NOT NULL AUTO_INCREMENT,
`name` varchar(400) NOT NULL,
`description` text NOT NULL,
`date` date NOT NULL,
`album_id` int NOT NULL,
PRIMARY KEY (`id`),
KEY `album_idx` (`album_id`),
CONSTRAINT `album_id` FOREIGN KEY (`album_id`)
REFERENCES `album` (`id`) ON UPDATE CASCADE))
```

# Local database schema

```
CREATE TABLE `comment` (
`id` int NOT NULL AUTO_INCREMENT,
`text` text NOT NULL,
`image_id` int NOT NULL,
`user_id` int NOT NULL,
`album_id` int NOT NULL,
PRIMARY KEY (`id`),
KEY `idimage_idx` (`image_id`),
KEY `iduser_idx` (`user_id`),
CONSTRAINT `idimage` FOREIGN KEY (`image_id`)
REFERENCES `image` (`id`) ON DELETE CASCADE ON UPDATE CASCADE),
CONSTRAINT `iduser` FOREIGN KEY (`user_id`)
REFERENCES `user` (`id`) ON DELETE CASCADE ON UPDATE CASCADE))
```

# Local database schema

```
CREATE VIEW `image_comments_view` AS
SELECT `i`.`id` AS `idimage`,`c`.`id` AS `idcomment`,`c`.`text` AS
`text`,`u`.`username` AS `username`
FROM ((`image` `i` join `comment` `c`) join `user` `u`)
WHERE ((`i`.`id` = `c`.`image_id`) and (`u`.`id` = `c`.`user_id`))
```

Esempio:

| idimage | idcomment | text | username |
|---|---|---|---|
| 1 | 3 | Bellissima foto! | andrea |
| 1 | 4 | Wow stupenda | andrea |
| 2 | 5 | Bello Scatto! | andrea |
| 9 | 6 | Mi piace | andrea |
| 2 | 8 | Ottima composizione | andrea |

# Versione HTML Pure

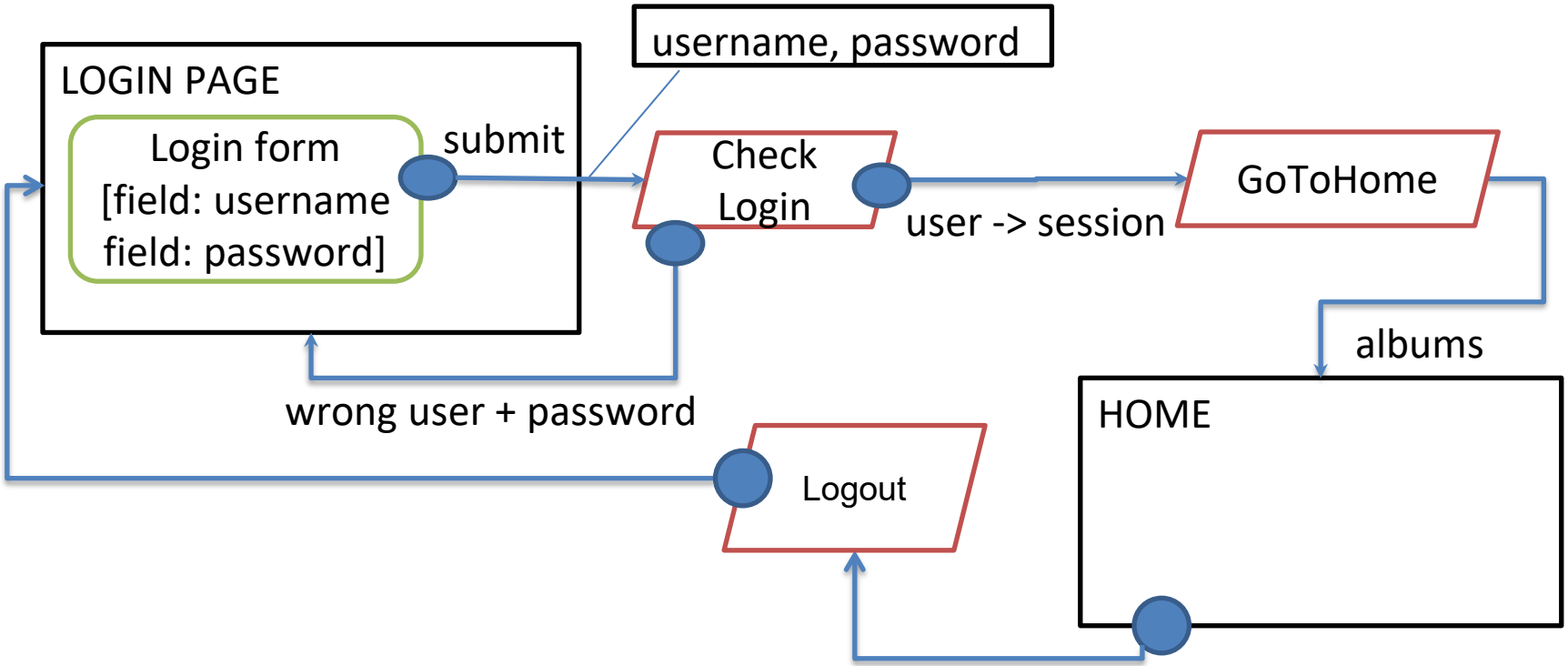
# Application requirements analysis

Quando l'utente accede all'HOME PAGE, questa presenta l'elenco degli album, ordinato per data di creazione decrescente. Quando l'utente clicca su un album nell'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. 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, la pagina ALBUM 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. L'invio del commento con un bottone INVIA ripresenta la pagina ALBUM PAGE, con tutti i dati aggiornati della stessa immagine. La pagina ALBUM PAGE contiene anche un collegamento per tornare all'HOME PAGE.

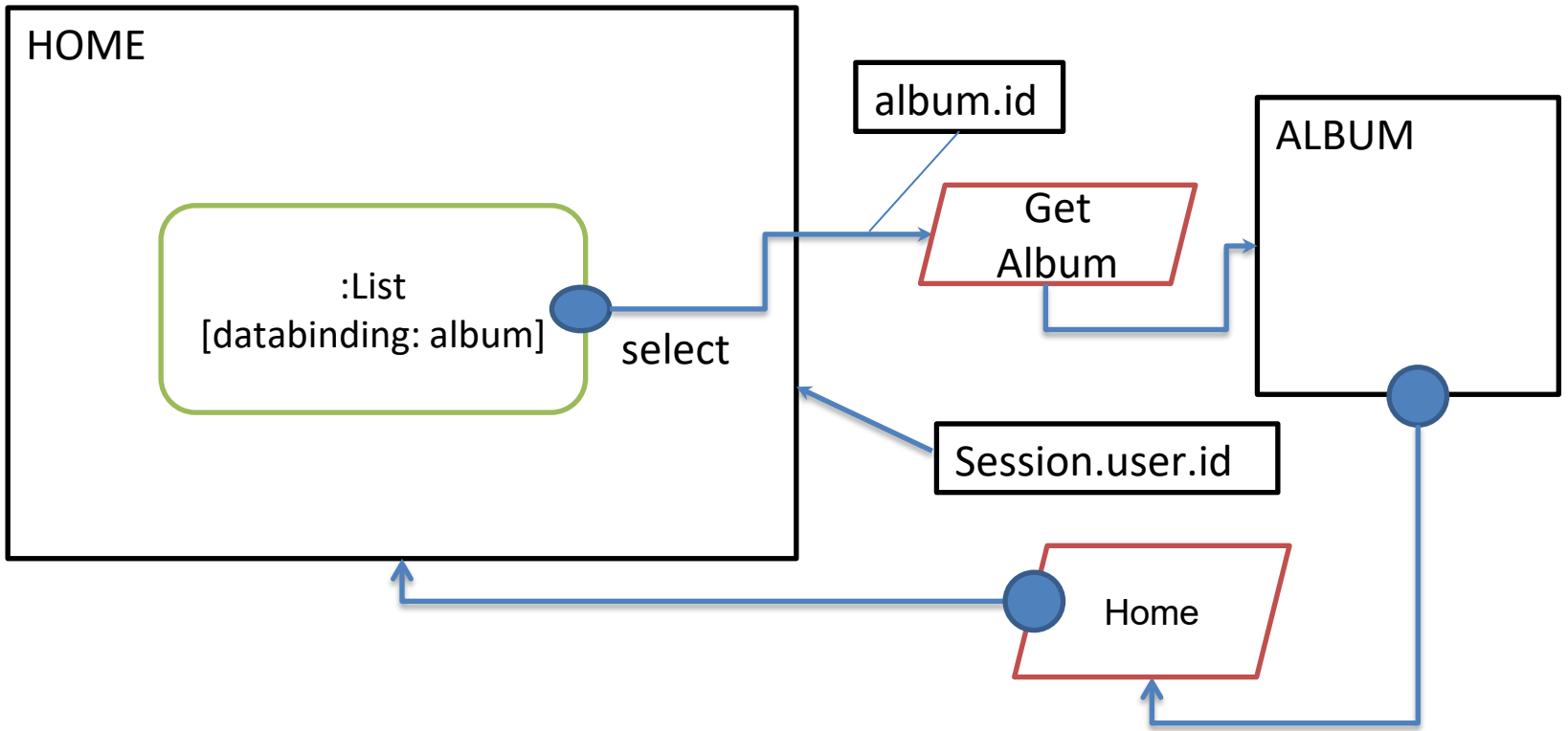
Completamento specifica: La pagina di default contiene la form di login

**Pages (views), view components, events, actions**

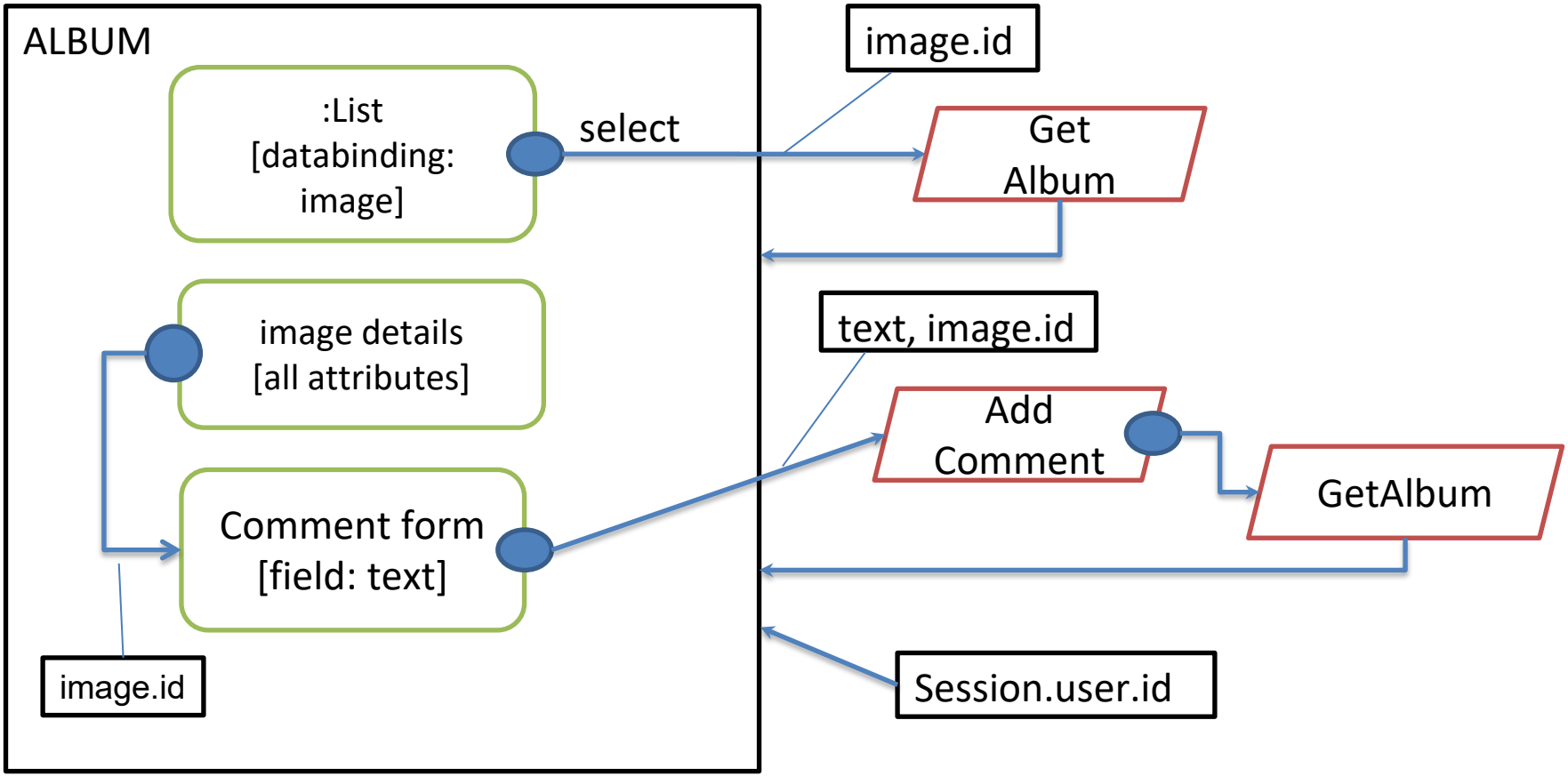
# Application design



# Application design



# Application design



# Components

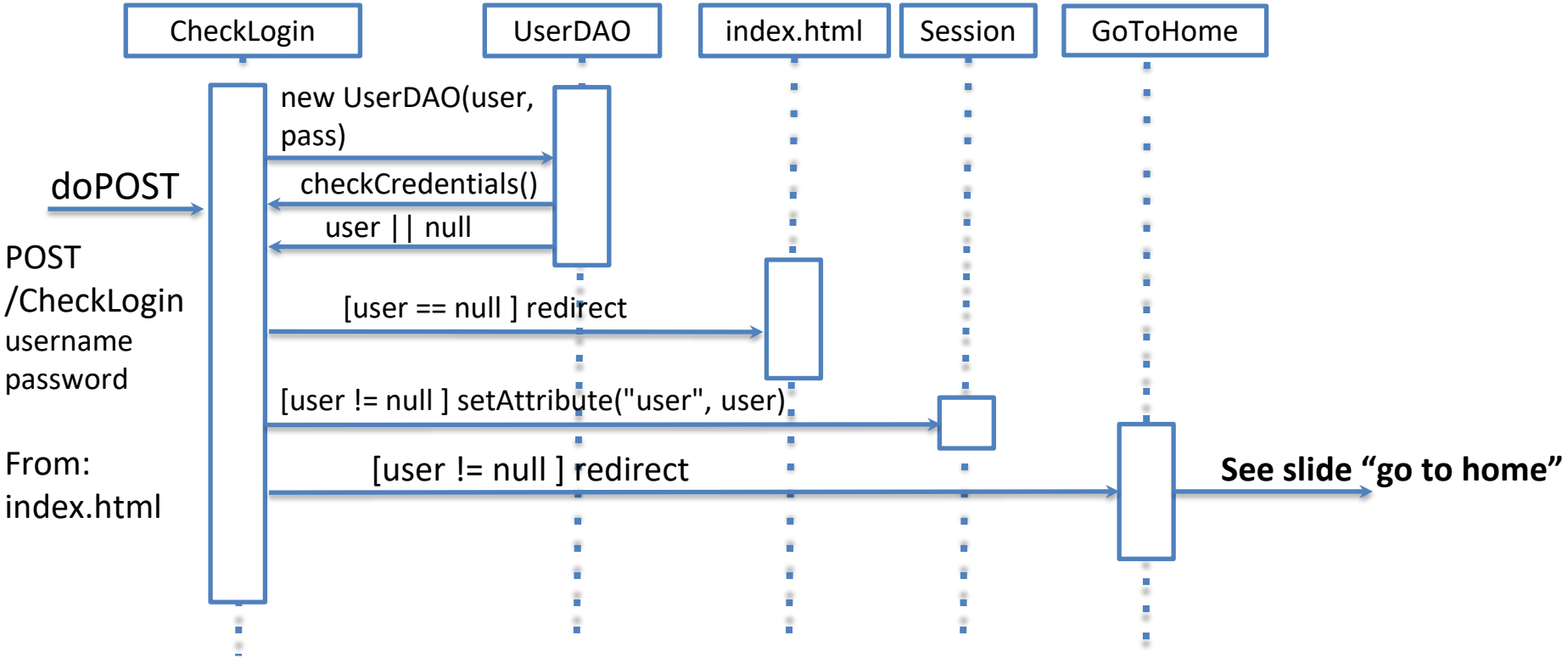
- Model objects (Beans)
  - User
  - Image
  - Album
  - Comment

- Data Access Objects (Classes)
  - UserDAO
    - checkCredentials(usrname, pwd)
  - AlbumDAO
    - getAllAlbums
  - ImageDAO
    - findImageById(imageid)
    - getImagesByAlbumId(albumid)
    - getImagesByAlbumAndGroupId(albumid, groupid)
    - getImagesByAlbumAndImageId(albumid, imageid)
  - CommentDAO
    - getCommentsByImageId(imageid)
    - addComment(comment, image, user)

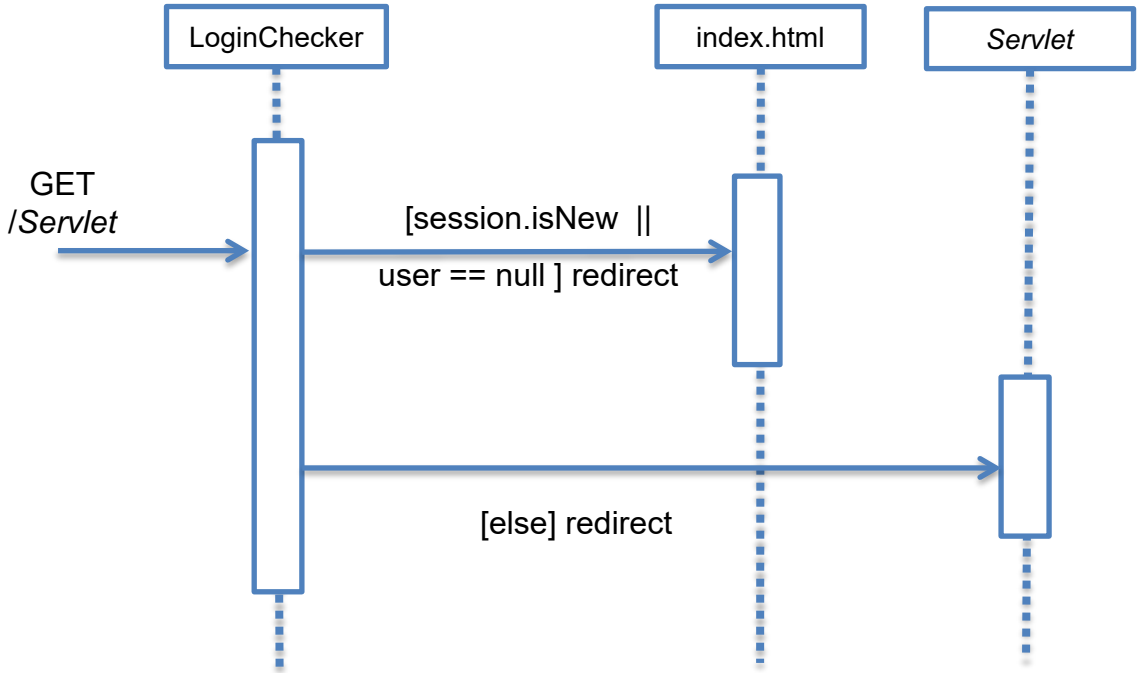
- Controllers (servlets)
  - CheckLogin
  - GoToHomePage
  - GetAlbum
  - AddComment
  - Logout

- Views (Templates)
  - Login (index)
  - Home
  - Album

# Event: login

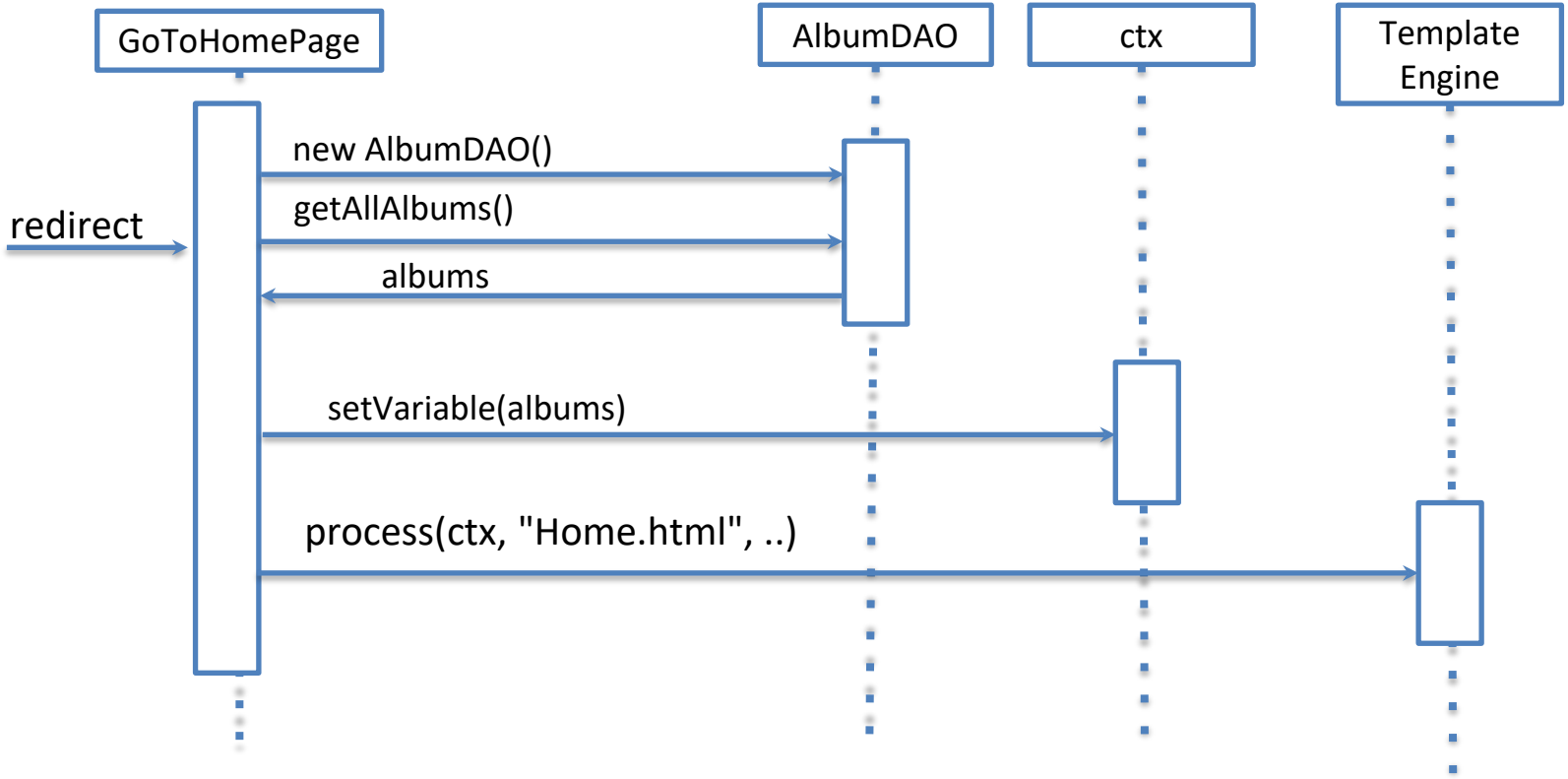

CheckLogin
UserDAO
index.html
Session
GoToHome
new UserDAO(user, pass)
doPOST
checkCredentials()
user || null
POST /CheckLogin
username
password
[user == null ] redirect
[user != null ] setAttribute("user", user)
From: index.html
[user != null ] redirect
See slide “go to home”

# Filter: Checking user logged in

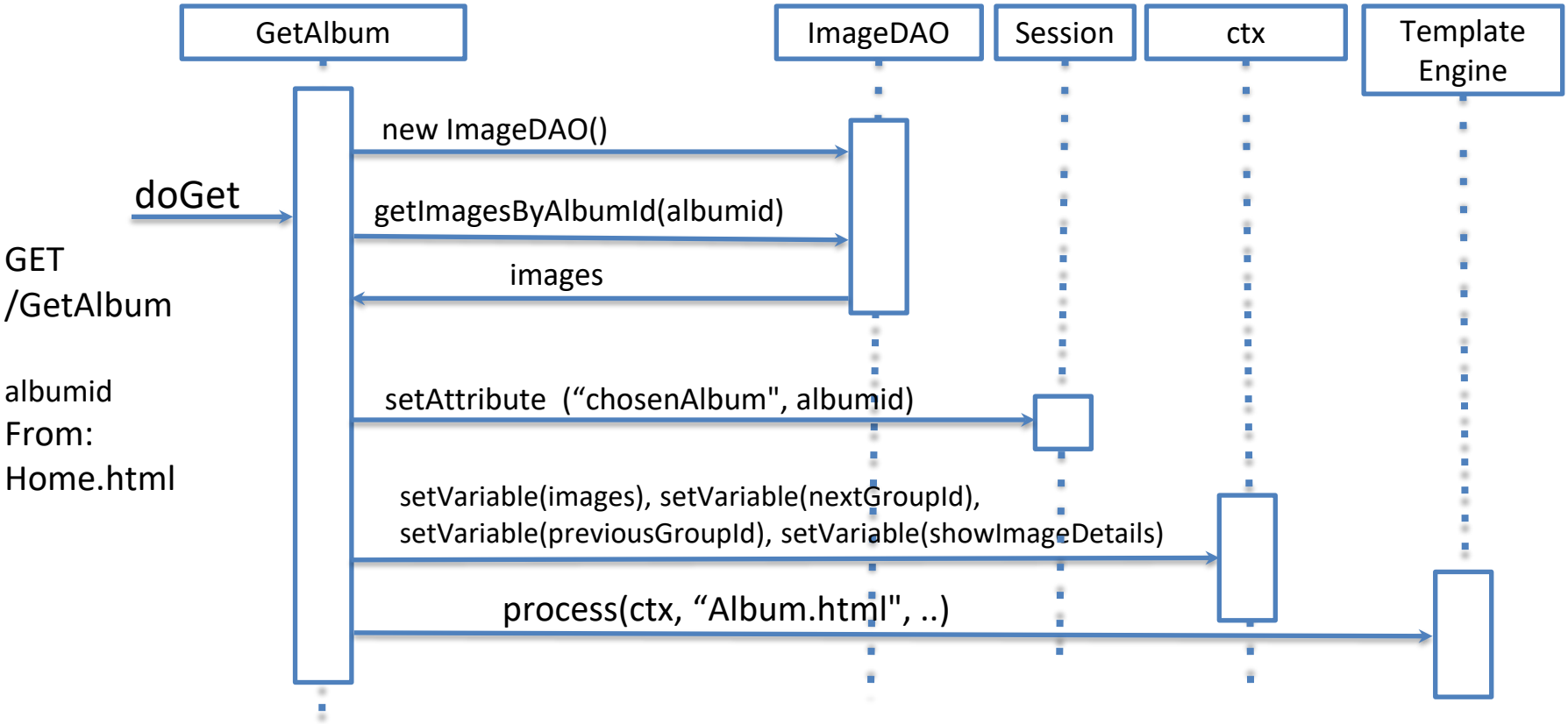



The same check is done in all servlets that manage events

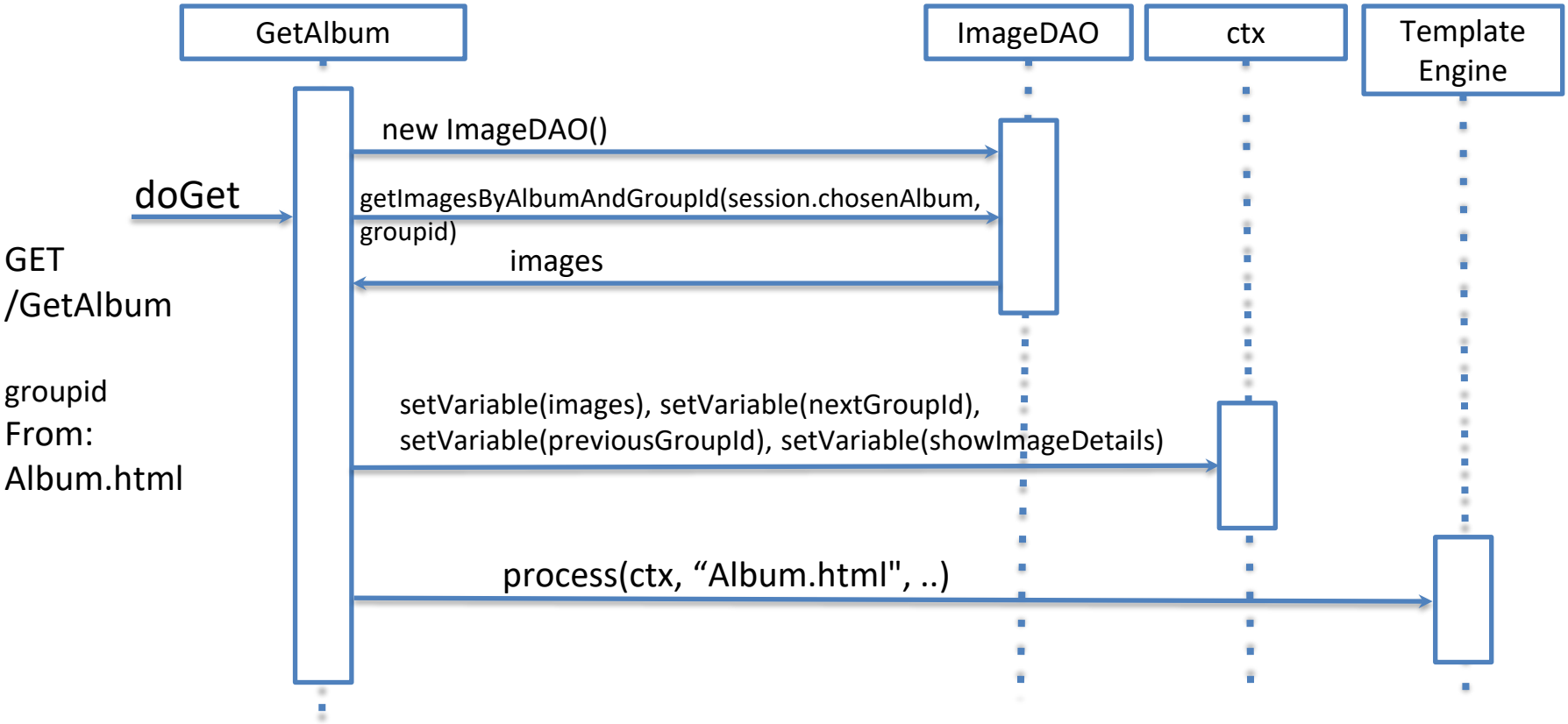
# Event: go to home



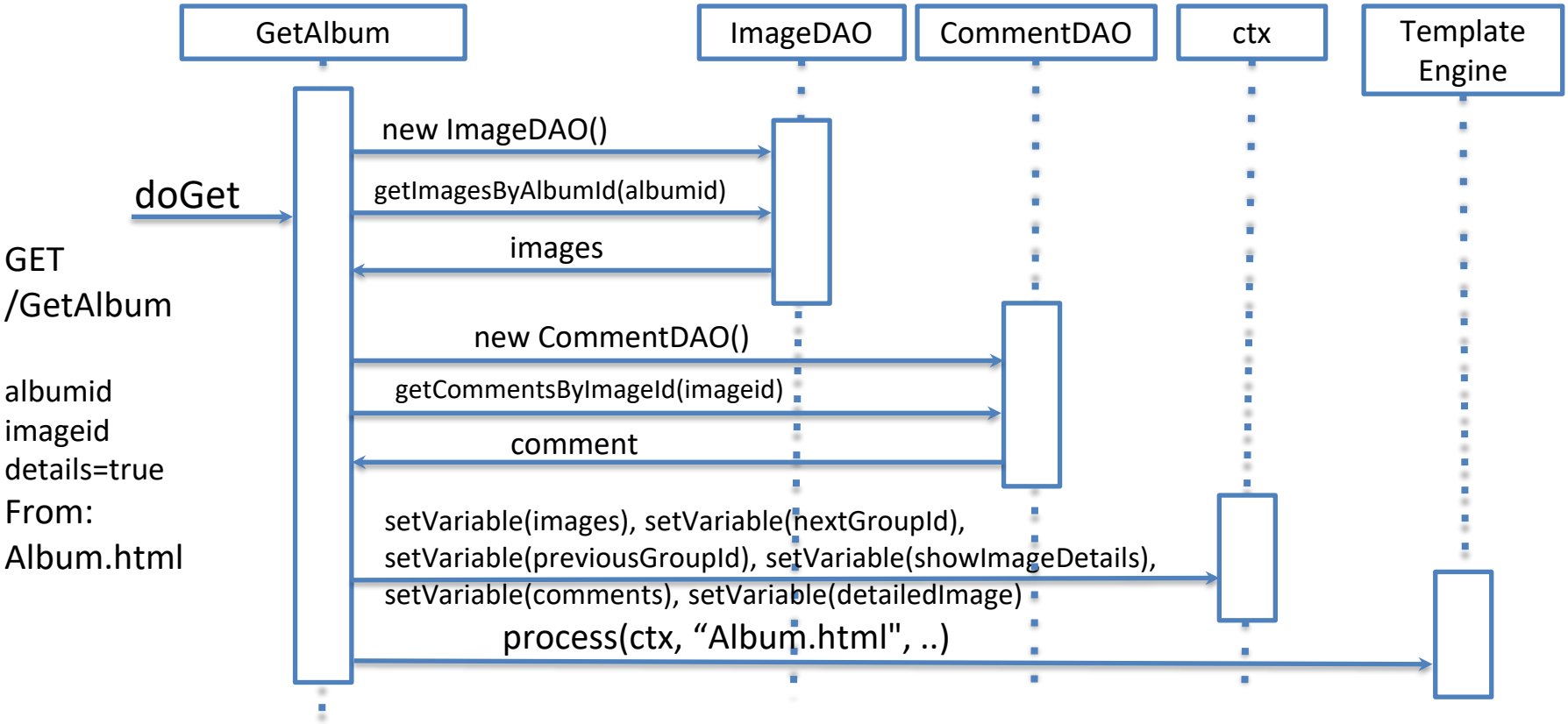
# Event: get album



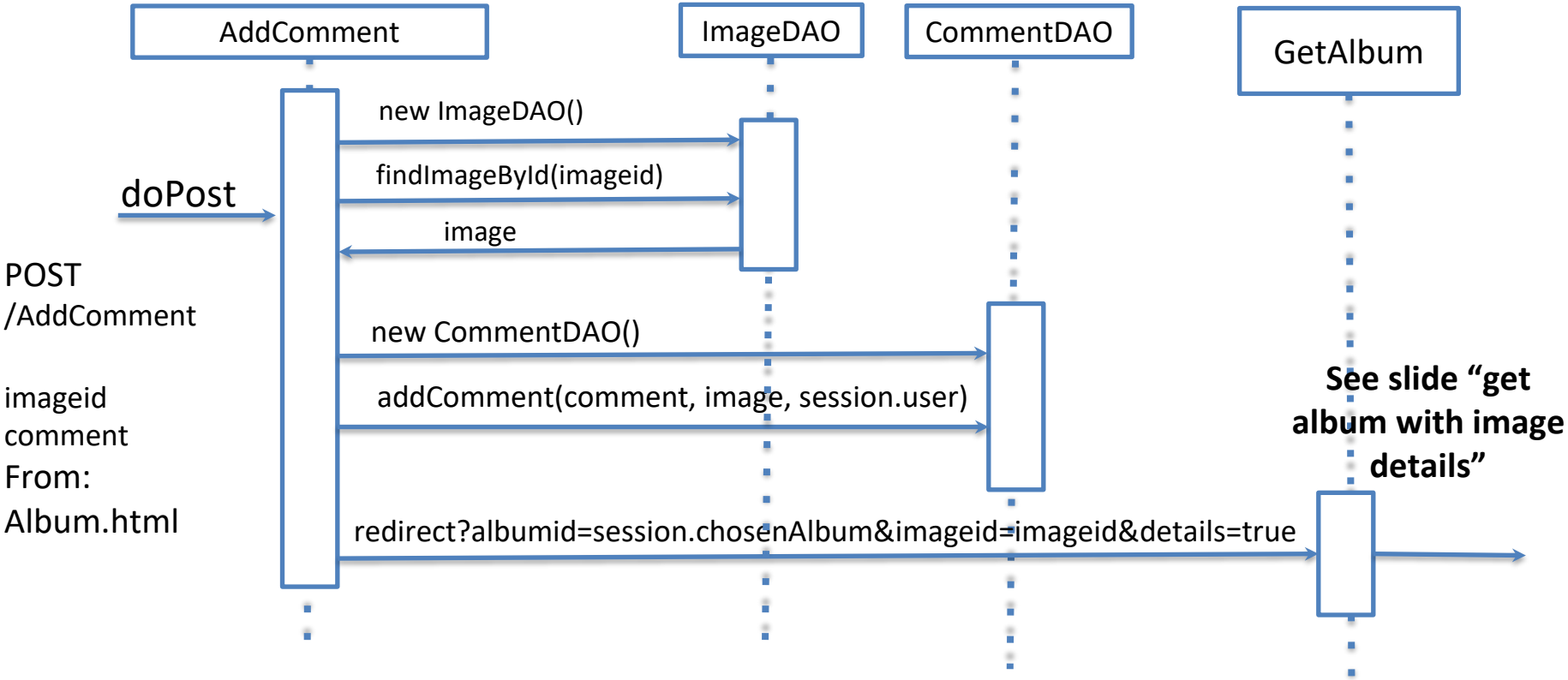
# Event: get album – next/previous



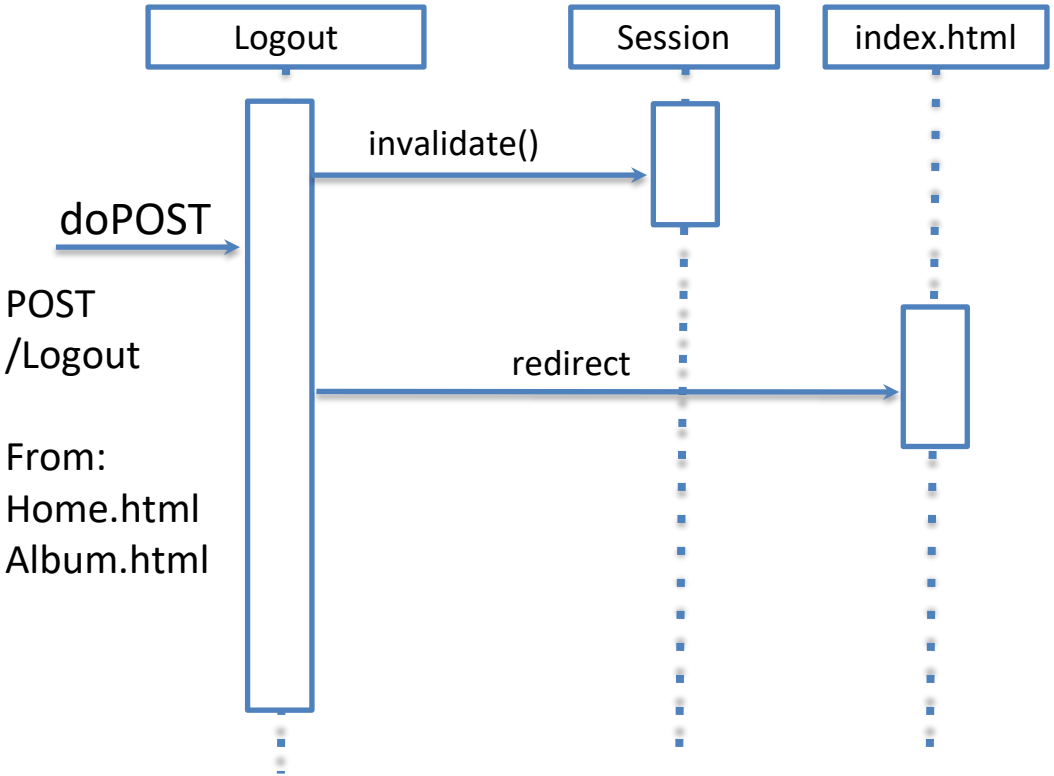
# Event: get album with image details



# Event: add comment

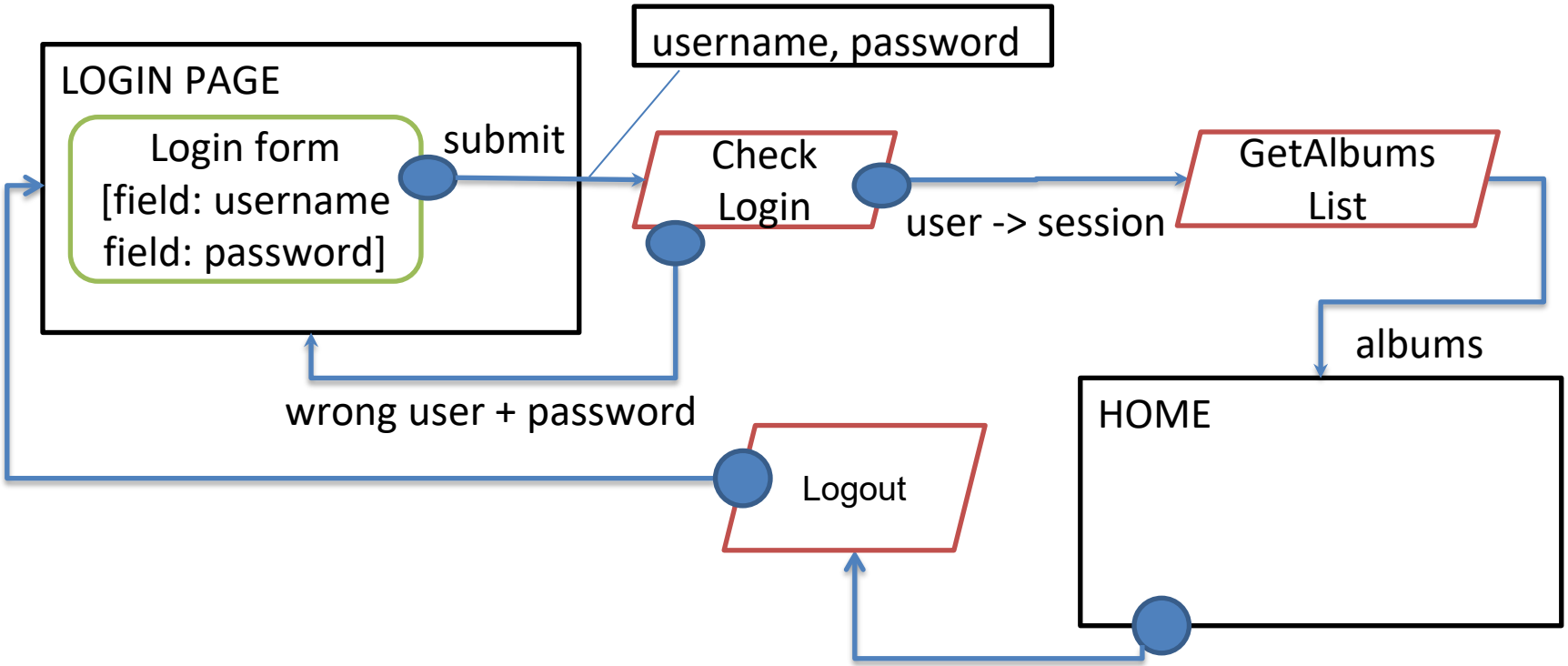


# Event: logout

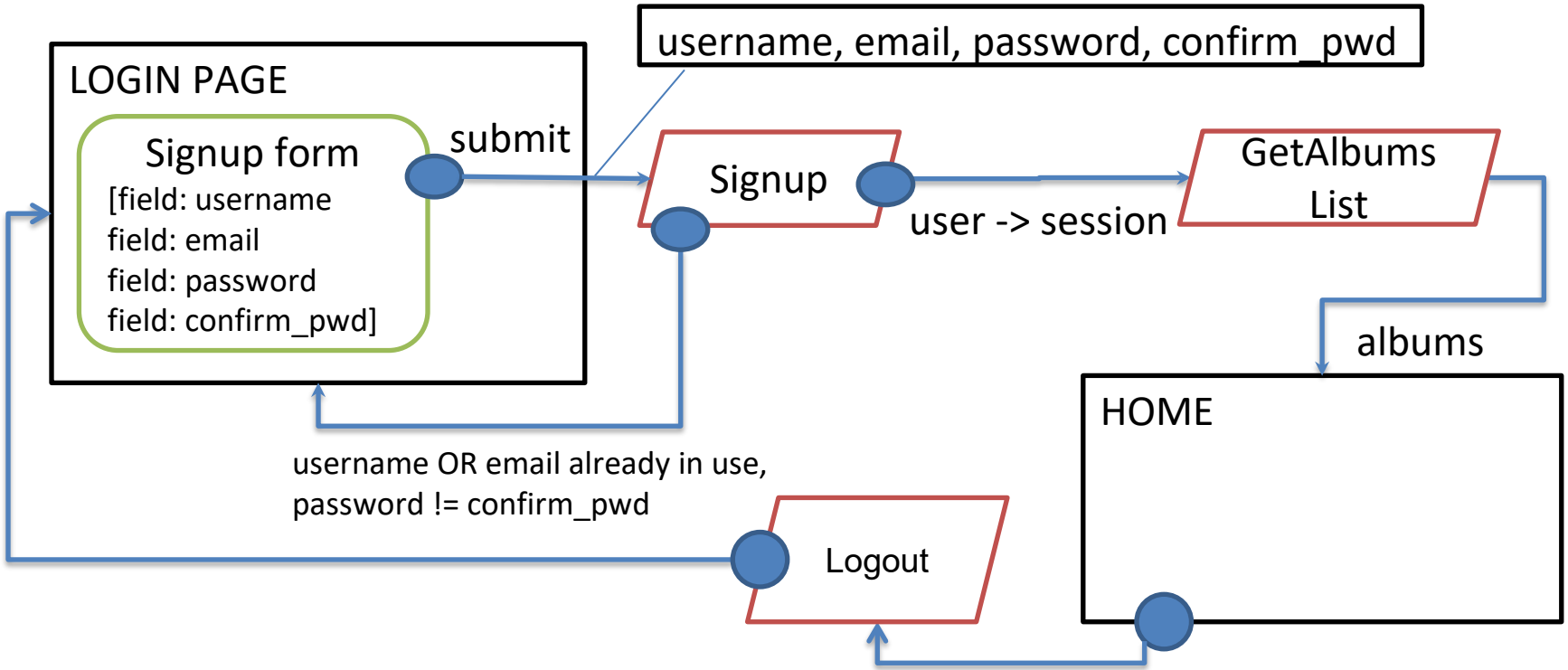


# Versione RIA

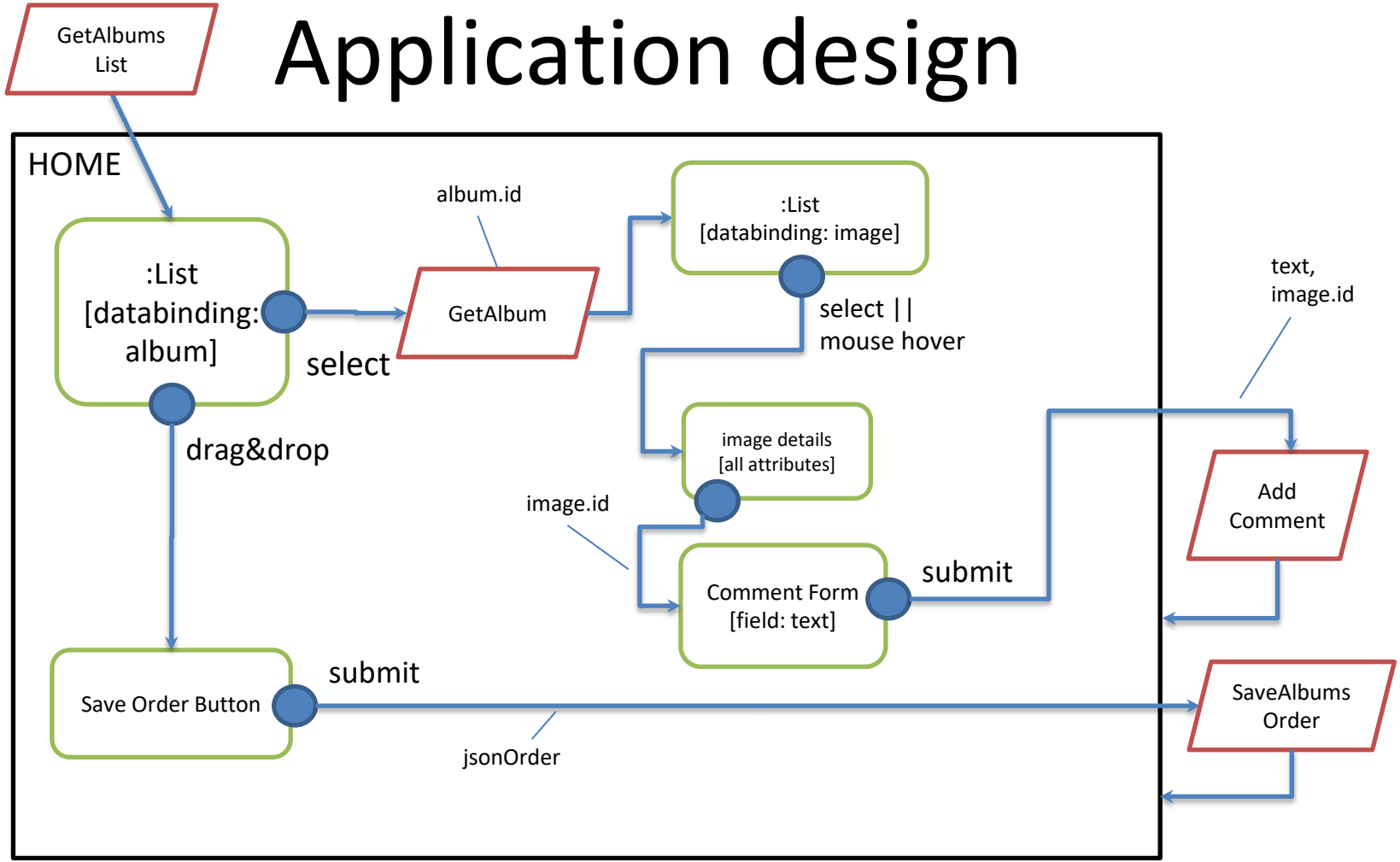
# Application design



# Application design



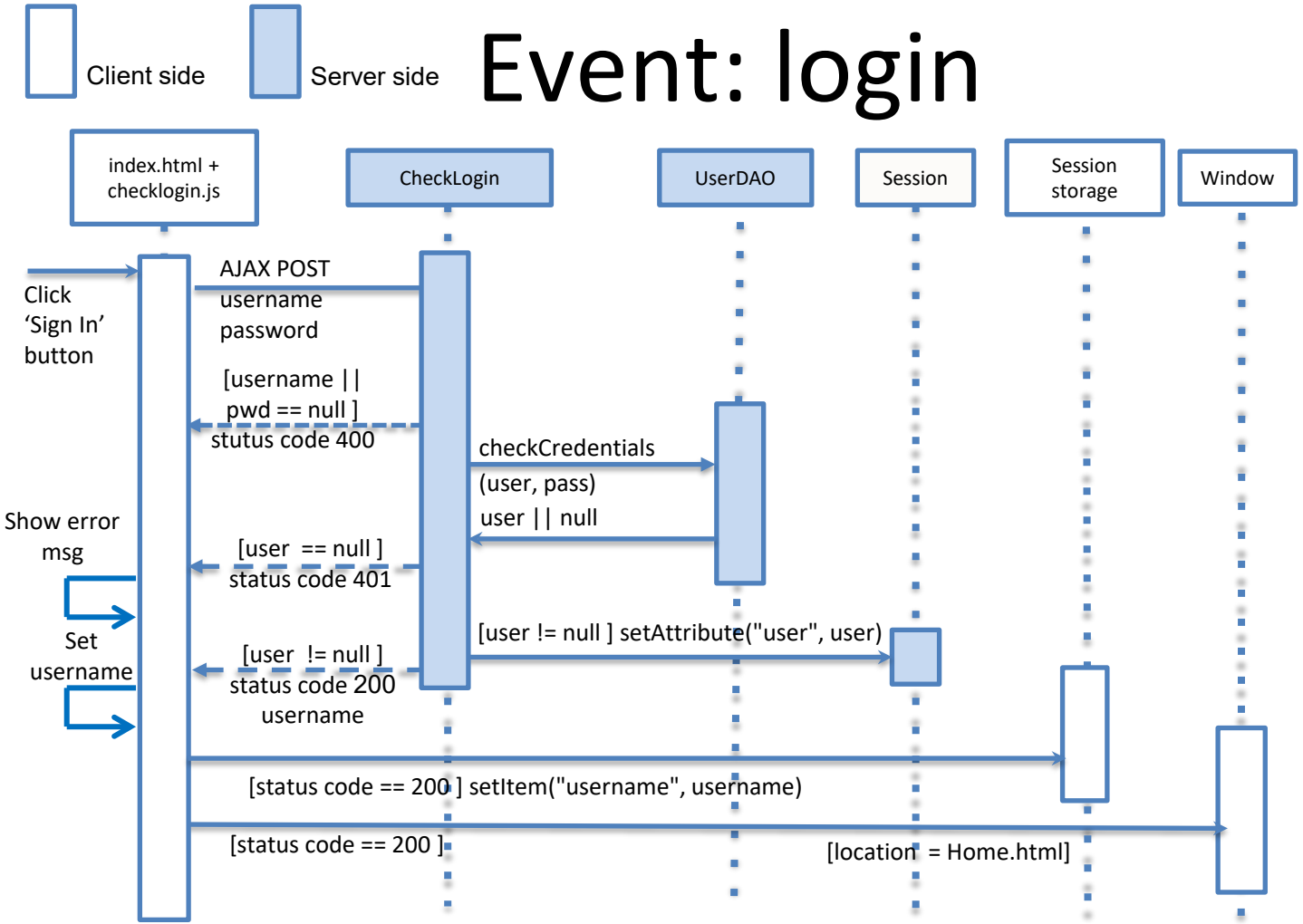
# Application design



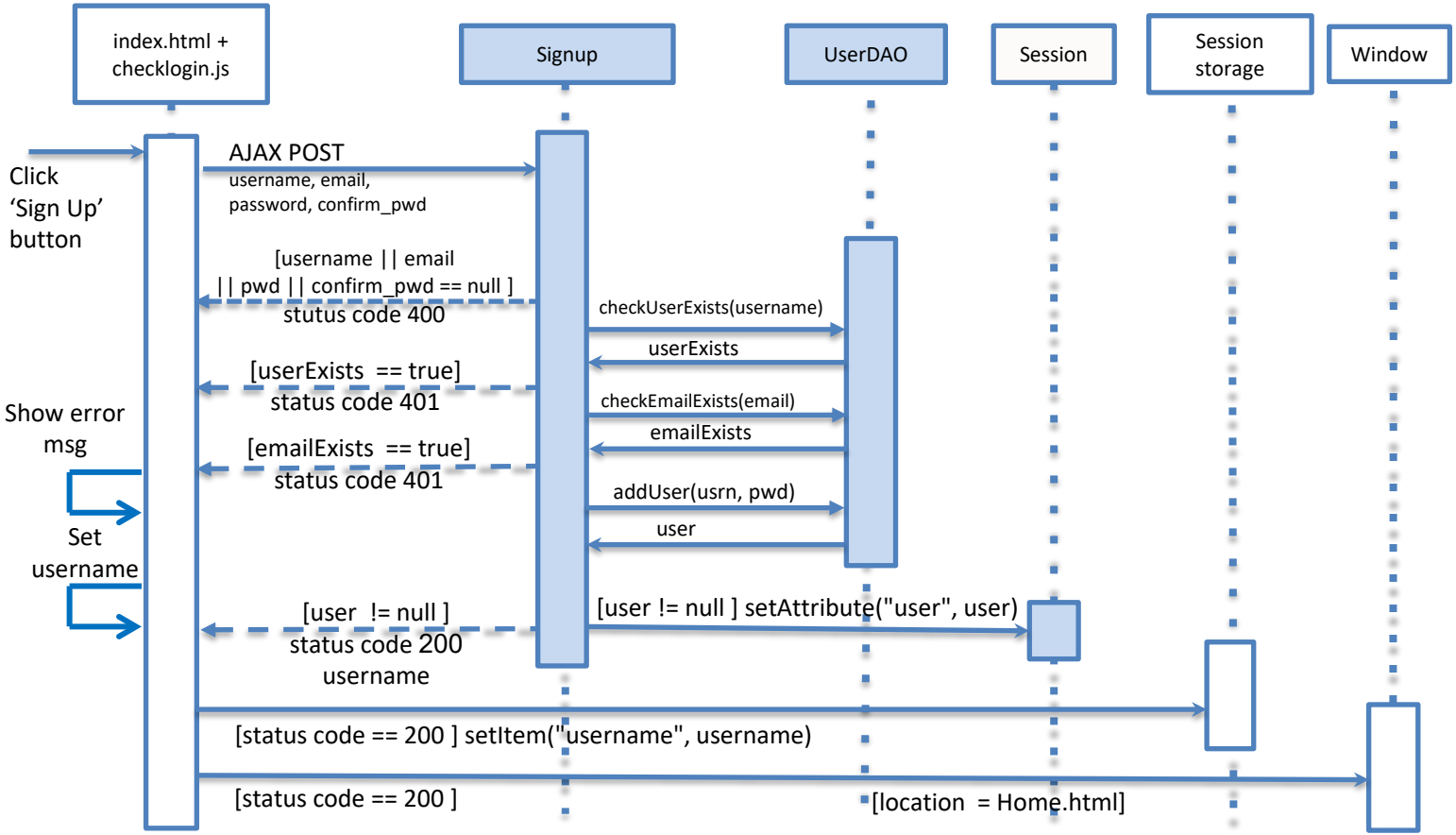
# Components

- Model objects (Beans)
  - User
  - Image
  - Album
  - Comment
- Data Access Objects (Classes)
  - UserDAO
    - checkCredentials(usrname, pwd)
    - addUser(username, email, pwd)
    - checkUserExists(username)
    - checkEmailExists(email)
    - saveAlbumsOrderForUser(jsonOrder, user)
  - AlbumDAO
    - getAllAlbums
  - ImageDAO
    - findImageById(imageid)
    - getImagesByAlbumId(albumid)
  - CommentDAO
    - getCommentsByImageId(imageid)
    - addComment(comment, image, user)
- Controllers (servlets)
  - CheckLogin
  - Signup
  - GetAlbum
  - GetAlbumsList
  - SaveAlbumsOrder
  - AddComment
  - Logout
- Views (Templates)
  - Login (index)
  - Home

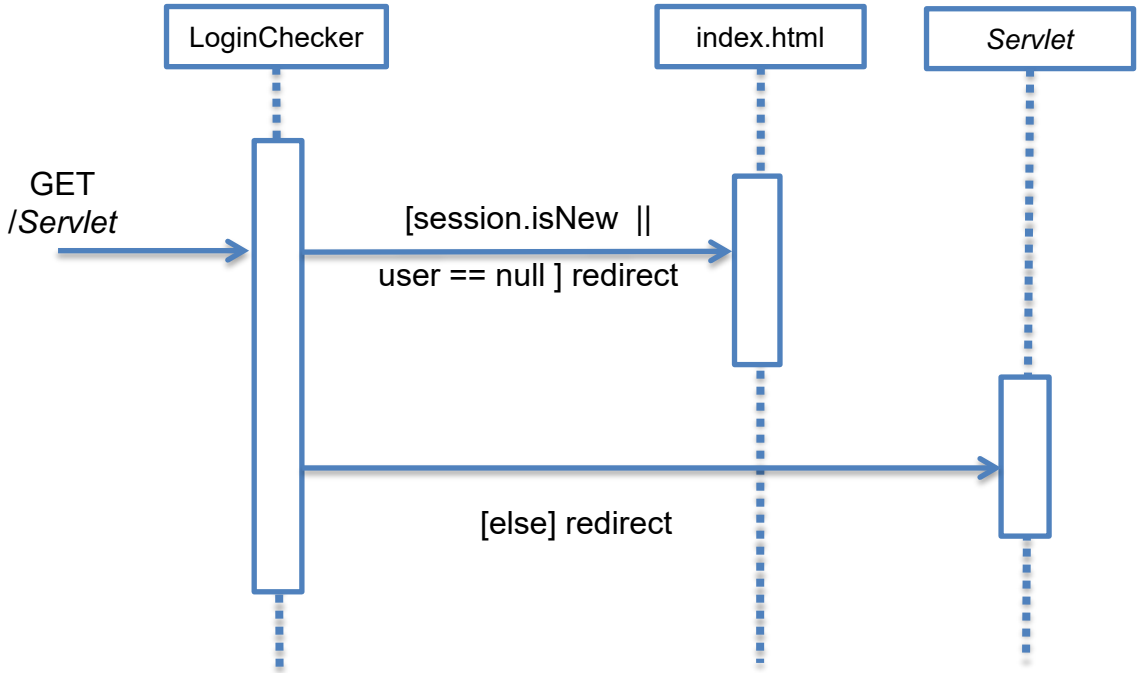
# Event: login


Client side
Server side
index.html + checklogin.js
CheckLogin
UserDAO
Session
Session storage
Window
Click 'Sign In' button
AJAX POST
username
password
[username || pwd == null ]
stutus code 400
checkCredentials
(user, pass)
user || null
Show error msg
[user == null ]
status code 401
Set username
[user != null ] setAttribute("user", user)
[user != null ]
status code 200
username
[status code == 200 ] setItem("username", username)
[status code == 200 ]
[location = Home.html]

# Event: signup

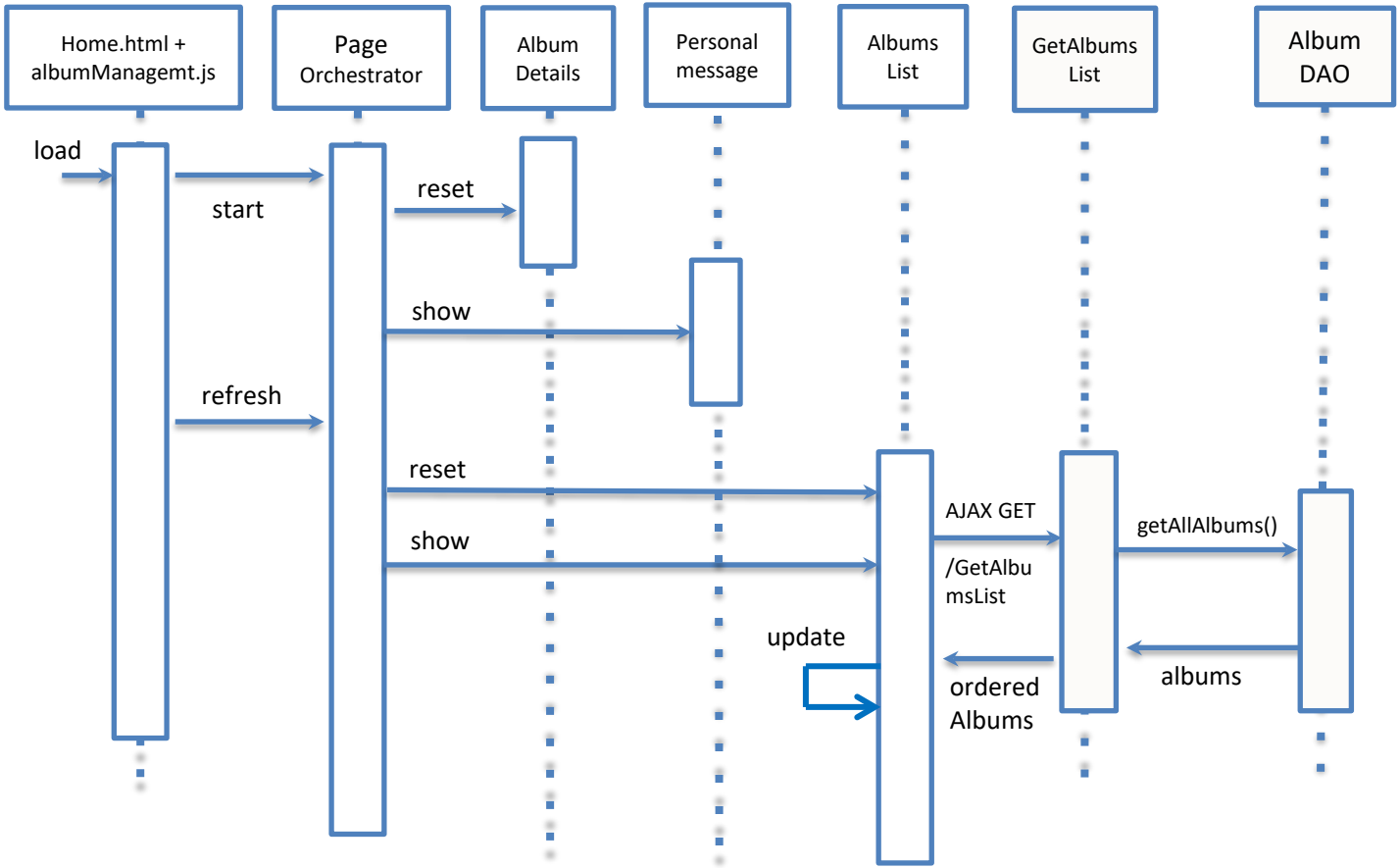


# Filter: Checking user logged in

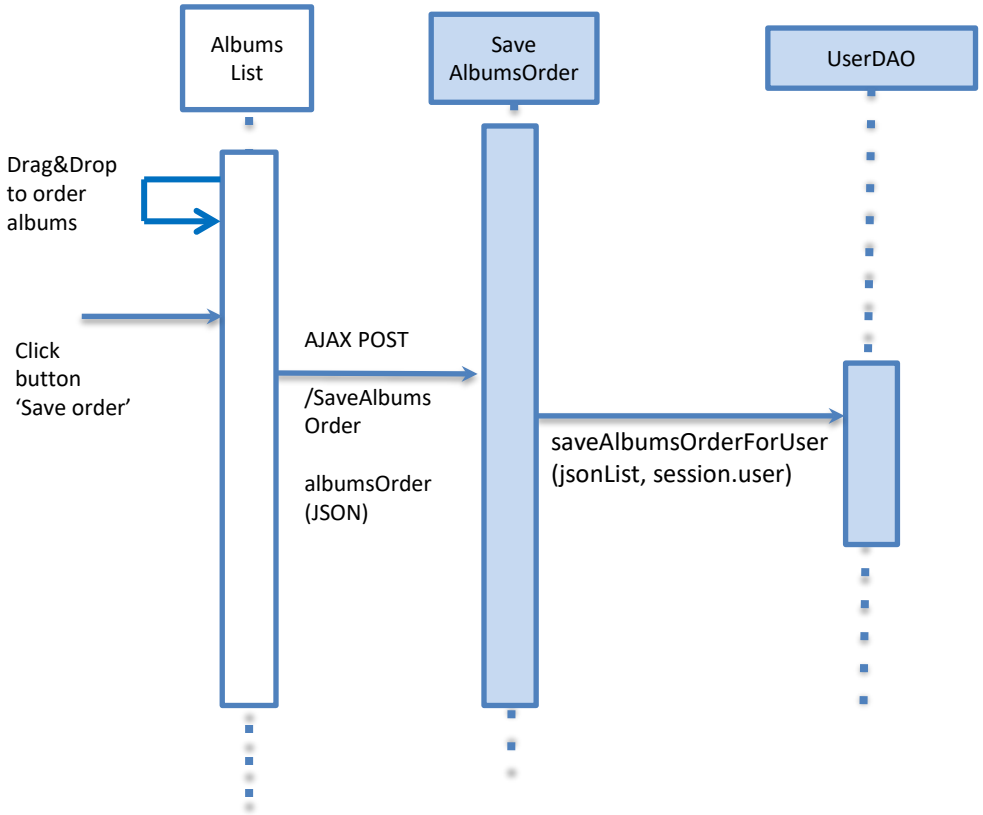



The same check is done in all servlets that manage events

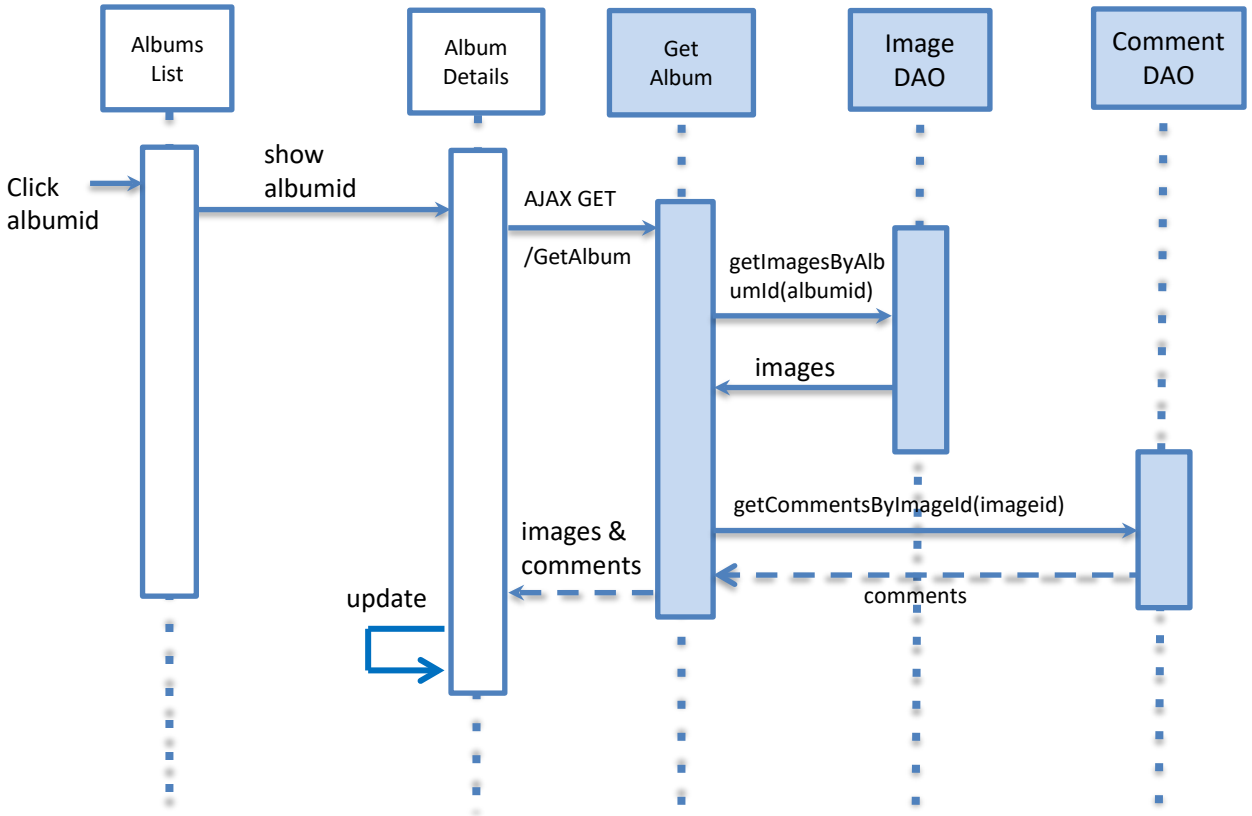
# Event: HomePage loading



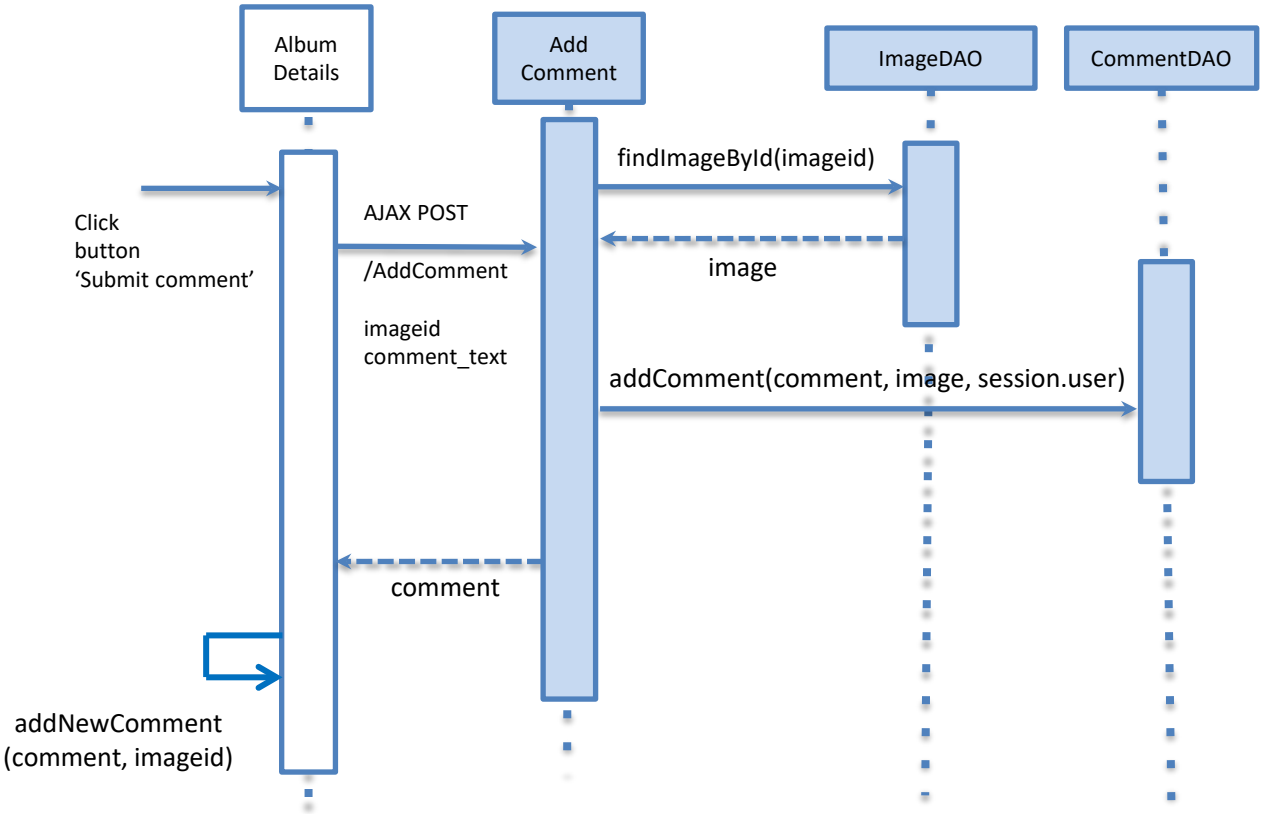
# Event: save albums order



# Evento: get album



# Event: add comment



# Evento: logout