Gestione di immagini

JavaScript

Gestione di immagini

Si realizzi un'applicazione client-server web che modifica le specifiche precedenti come segue:

• 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.

Gestione di immagini

• 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

Local database schema

```
CREATE TABLE `Album` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Title` varchar(45) NOT NULL,
  `Creator` varchar(45) NOT NULL,
  `CreationDate` date NOT NULL
DEFAULT (curdate()),
  PRIMARY KEY (`ID`),
  KEY `creatore_idx` (`Creator`),
  CONSTRAINT `creator` FOREIGN KEY
(`Creator`) REFERENCES `User`
(`Email`) ON DELETE RESTRICT ON
UPDATE CASCADE
)
```

```
CREATE TABLE `Comment` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Text` varchar(255) NOT NULL,
  `User` varchar(45) NOT NULL,
  `Image` int NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `utente_idx` (`User`),
  KEY `image_idx` (`Image`),
  CONSTRAINT `image` FOREIGN KEY
(`Image`) REFERENCES `Image` (`ID`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `user` FOREIGN KEY
(`User`) REFERENCES `User` (`Email`)
ON DELETE CASCADE ON UPDATE CASCADE
)
```

Local database schema

```
CREATE TABLE `Image` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Title` varchar(45) NOT NULL,
  `CreationDate` date NOT NULL
DEFAULT (curdate()),
  `Description` varchar(255) DEFAULT
NULL,
  `Path` varchar(255) NOT NULL,
  `User` varchar(45) NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `utente_idx` (`User`),
  CONSTRAINT `utente` FOREIGN KEY
(`User`) REFERENCES `User` (`Email`)
ON UPDATE CASCADE
)
```

```
CREATE TABLE `User` (
  `Email` varchar(45) NOT NULL,
  `Password` varchar(45) NOT NULL,
  `Name` varchar(45) NOT NULL,
  `Surname` varchar(45) NOT NULL,
  PRIMARY KEY (`Email`)
)
```

Local database schema

```
CREATE TABLE `ImageOfAlbum` (
  `Album` int NOT NULL,
  `Image` int NOT NULL,
  `Position` int NOT NULL,
  `User` varchar(45) NOT NULL,
  PRIMARY KEY (`Album`,`Image`,`User`),
  KEY `fk_ImageOfAlbum_1_idx` (`Image`),
  KEY `fk_ImageOfAlbum_2_idx` (`Album`),
  KEY `fk_ImageOfAlbum_3_idx` (`User`),
  CONSTRAINT `fk_ImageOfAlbum_1` FOREIGN KEY (`Image`)
REFERENCES `Image` (`ID`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `fk_ImageOfAlbum_2` FOREIGN KEY (`Album`)
REFERENCES `Album` (`ID`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `fk_ImageOfAlbum_3` FOREIGN KEY (`User`)
REFERENCES `User` (`Email`) ON DELETE CASCADE ON UPDATE
CASCADE
)
```

Application requirements analysis

• 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.

Pages (views), view components, events, actions

Application requirements analysis

• 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.

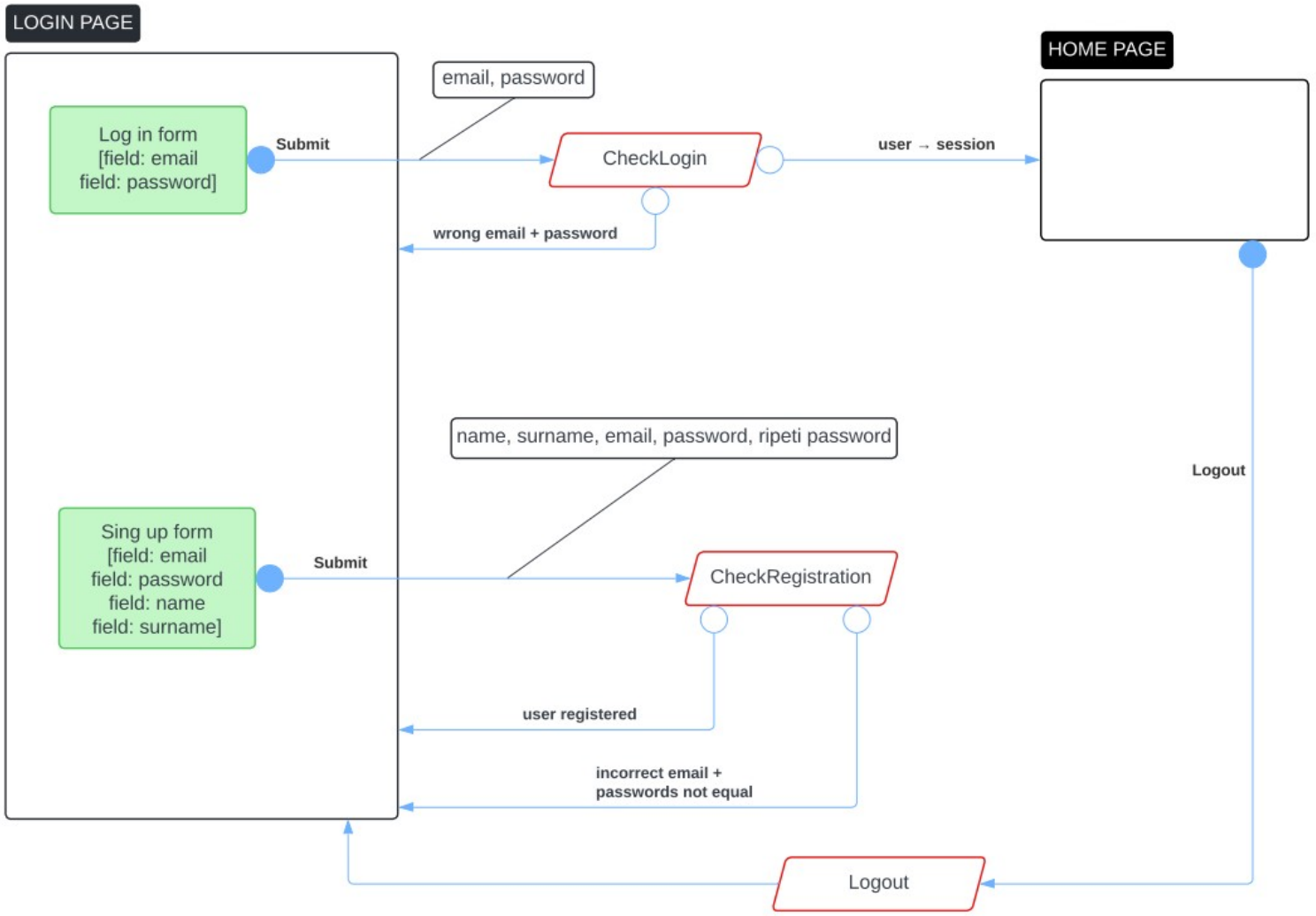
• L'applicazione controlla anche a lato client che non si invii un commento vuoto.

• Errori a lato server devono essere segnalati mediante un messaggio di allerta all'interno della pagina.

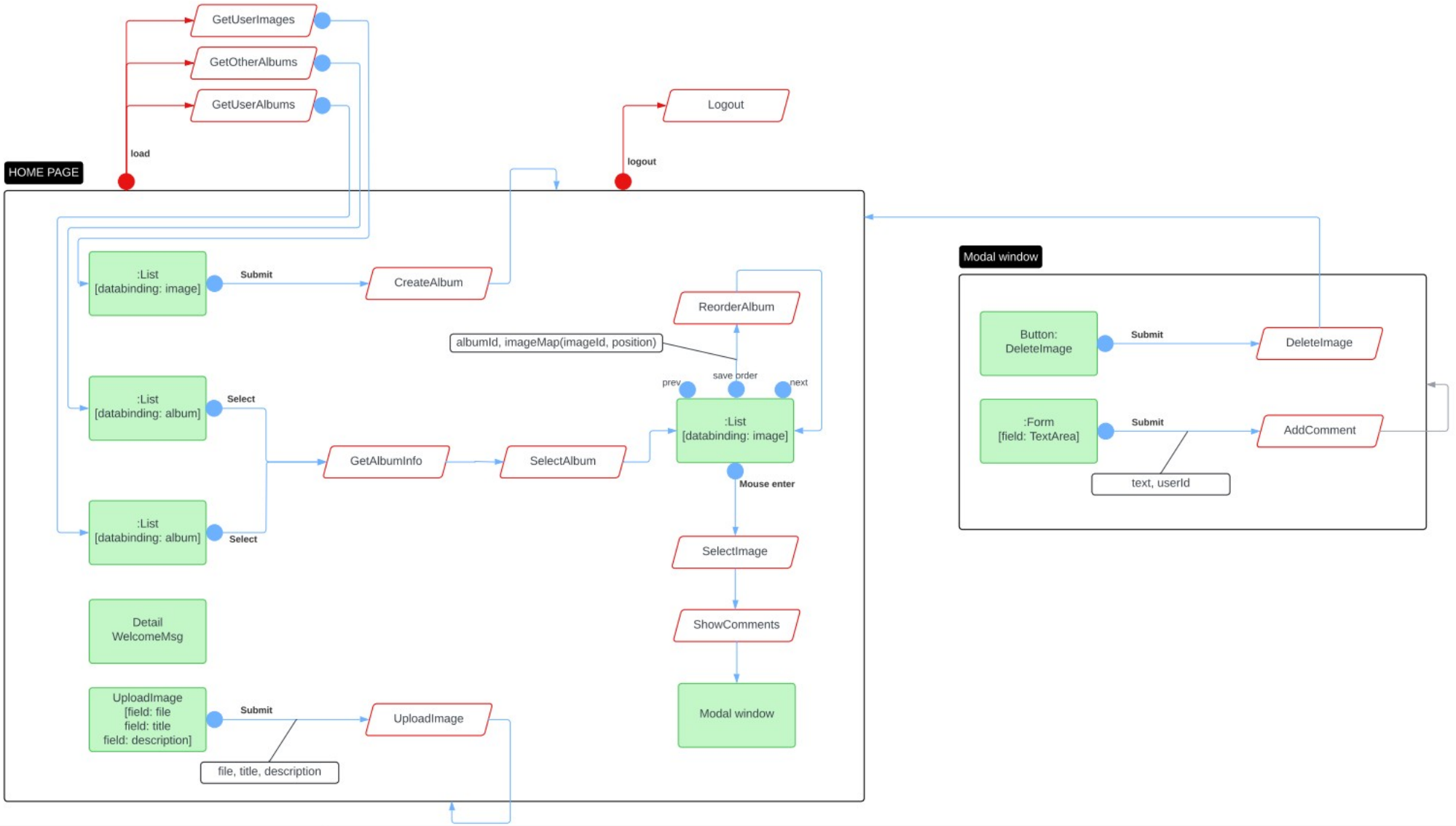
• Si deve consentire all'utente di riordinare l'elenco delle immagini all'interno di un album con un criterio personalizzato diverso da quello di default (data decrescente). L'utente accede a un elenco dei titoli delle immagini ordinato secondo il criterio correntemente in uso: default o personalizzato. 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

Pages (views), view components, events, actions

Application design



Application design



Eventi & azioni

CLIENTSIDE			SERVERSIDE		
Evento	Azione		Evento	Azione	
Index -> login form -> submit	Controllo dati		POST Username password	Controllo credenziali	
Index -> login form -> submit	Controllo dati		POST Username password	Controllo credenziali di registrazione	
Home page -> load	Aggiorna view: [immagini associate all'utente per creare un album] [Lista album altri utenti] [Lista album user]		Get (nessun parametro)	Estrazione immagini dell'utente, album utente ed album altri utenti	
Home page -> elenco album -> seleziona album	Aggiorna view con dati dell'album		GET id album	Estrazione titolo, creatore, immagini	
Home page -> upload Image -> submit	Controllo form, dati e file		POST (dati immagine)	Inserimento immagine	
Home page -> create album -> submit	Controllo form		POST (titolo album, immagini album)	Creazione nuovo album	
Home page -> next/prev	Scorrimento tabella album				
Modal -> delete image	Controllo appartenenza immagine		GET imageId	Eliminazione immagine	
Modal -> create comment -> submit	Controllo validità commento		POST (image id, text)	Aggiunta nuovo commento per l'immagine	
Logout			GET	Terminazione sessione	

Controller / event handler

EVENTO	CONTROLLORE		EVENTO	CONTROLLORE
Index -> login form -> submit	Function makeCall		POST Username password	CheckLogin (servlet)
Index -> login form -> submit	Function makeCall		POST Username password	CheckRegistration (servlet)
Home page -> load			Get (nessun parametro)	GetUserImages, GetOtherAlbums, GetUserAlbums (servlet)
Home page -> elenco album -> seleziona album	Function albumDetails.show		GET id album	GetAlbumInfo, SelectAlbum (servlet)
Home page -> upload Image -> submit	Function makeCall		POST (dati immagine)	UploadImage (servlet)
Home page -> create album -> submit	Function makeCall		POST (titolo album, immagini album)	CreateAlbum (servlet)
Home page -> next/prev	Function albumDetails.update			
Modal -> delete image	Function makeCall		GET imageId	DeleteImage (servlet)
Modal -> create comment -> submit	Function makeCall		POST (image id, text)	AddComment (servlet)
Logout			GET	Logout (servlet)

Server side: DAO & model objects

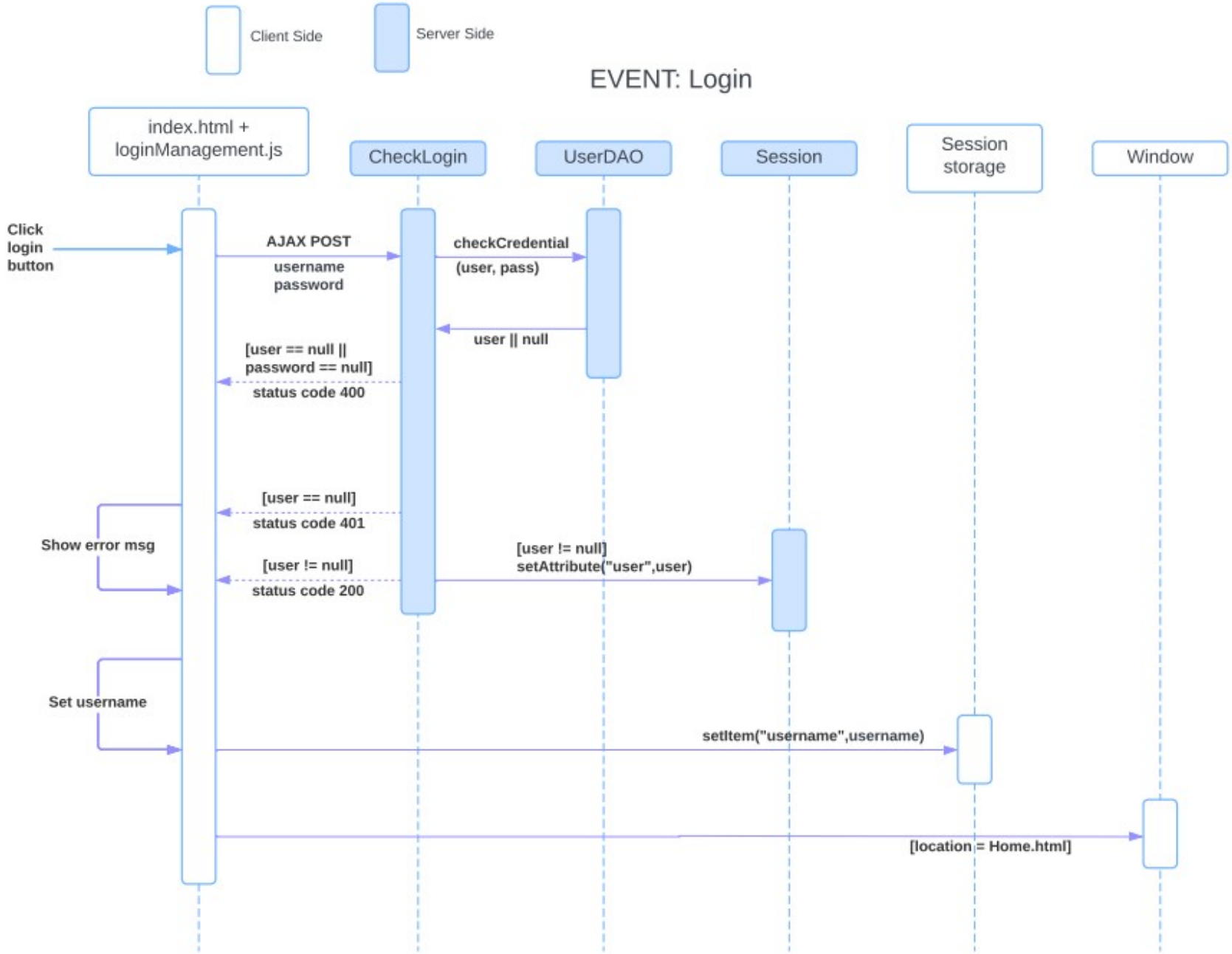
- Model objects (Beans)
  - User
  - Comment
  - Image
  - Album

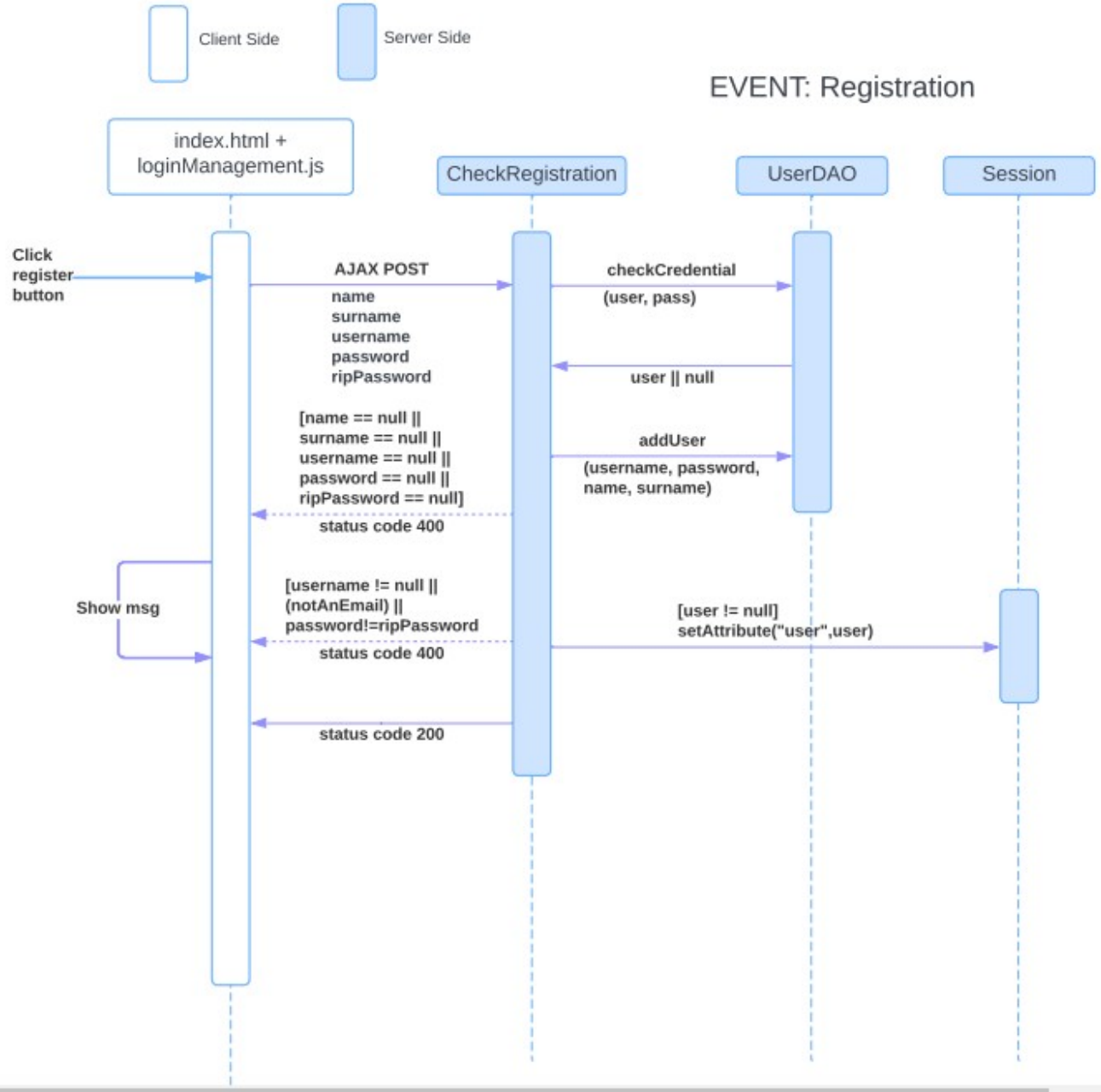
- Data Access Objects (Classes)
  - AlbumDAO
    - createAlbum(title, email, images)
    - findOther(email)
    - findByUser(email)
    - findById(albumId)
    - reorderAlbum(albumId, imagesOrder, email)
  - ImageDAO
    - addImage(path, title, description, email)
    - findByUser(email)
    - findByAlbum(albumId, email)
    - findById(imageID)
    - deleteImage(imageId)
  - UserDAO
    - checkCredentials(email, password)
    - addUser(email, password, name, surname)
  - CommentDAO
    - addComment(text, email)
    - findByImage(imageId)

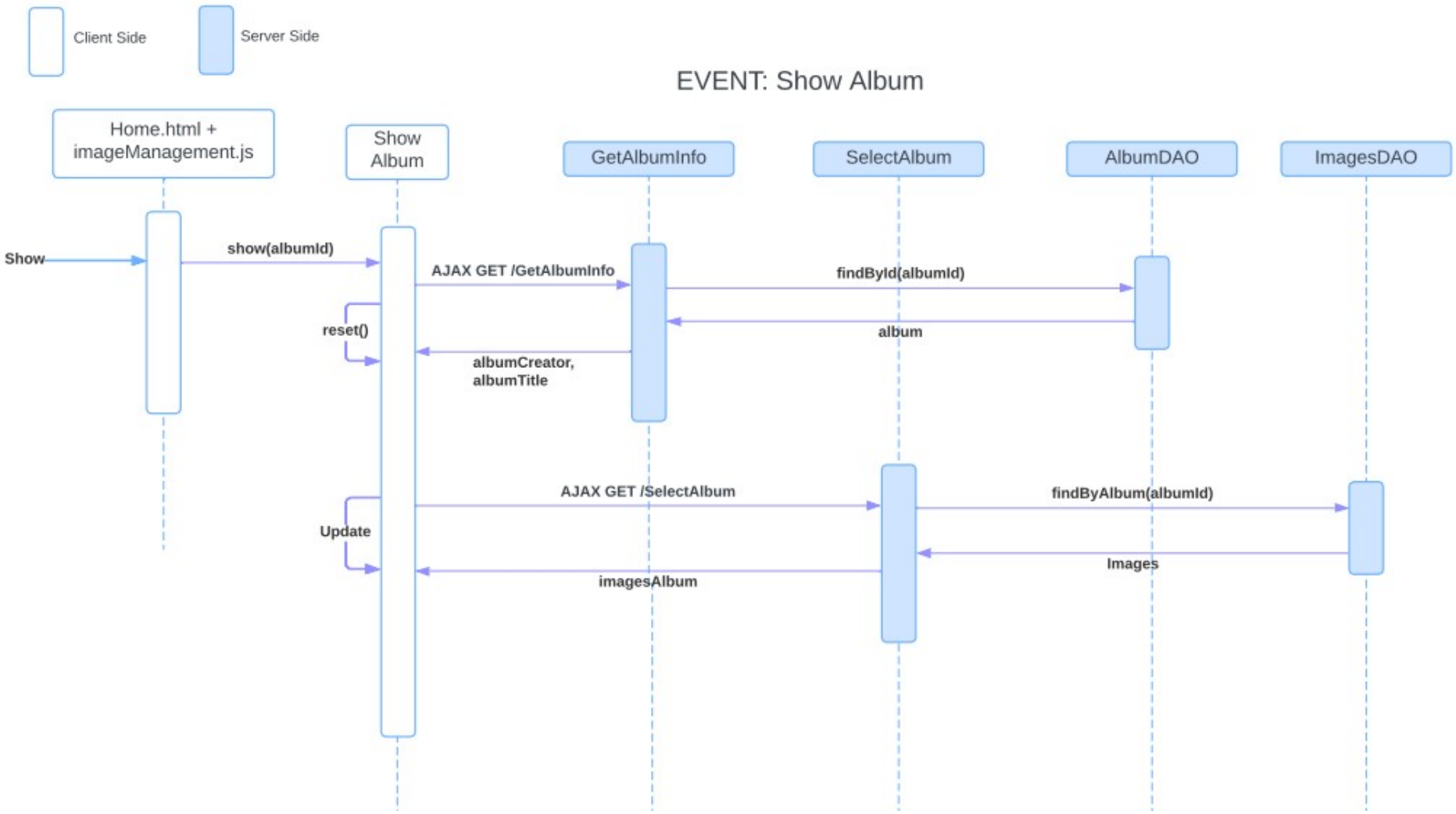
- Controllers (Servlets)
  - AddComment
  - CheckLogin
  - CheckRegistration
  - CreateAlbum
  - DeleteImage
  - GetAlbumInfo
  - GetImage
  - GetOtherAlbums
  - GetUserAlbums
  - GetUserImages
  - Logout
  - ReorderAlbum
  - SelectAlbum
  - SelectImage
  - ShowComments
  - UploadImage

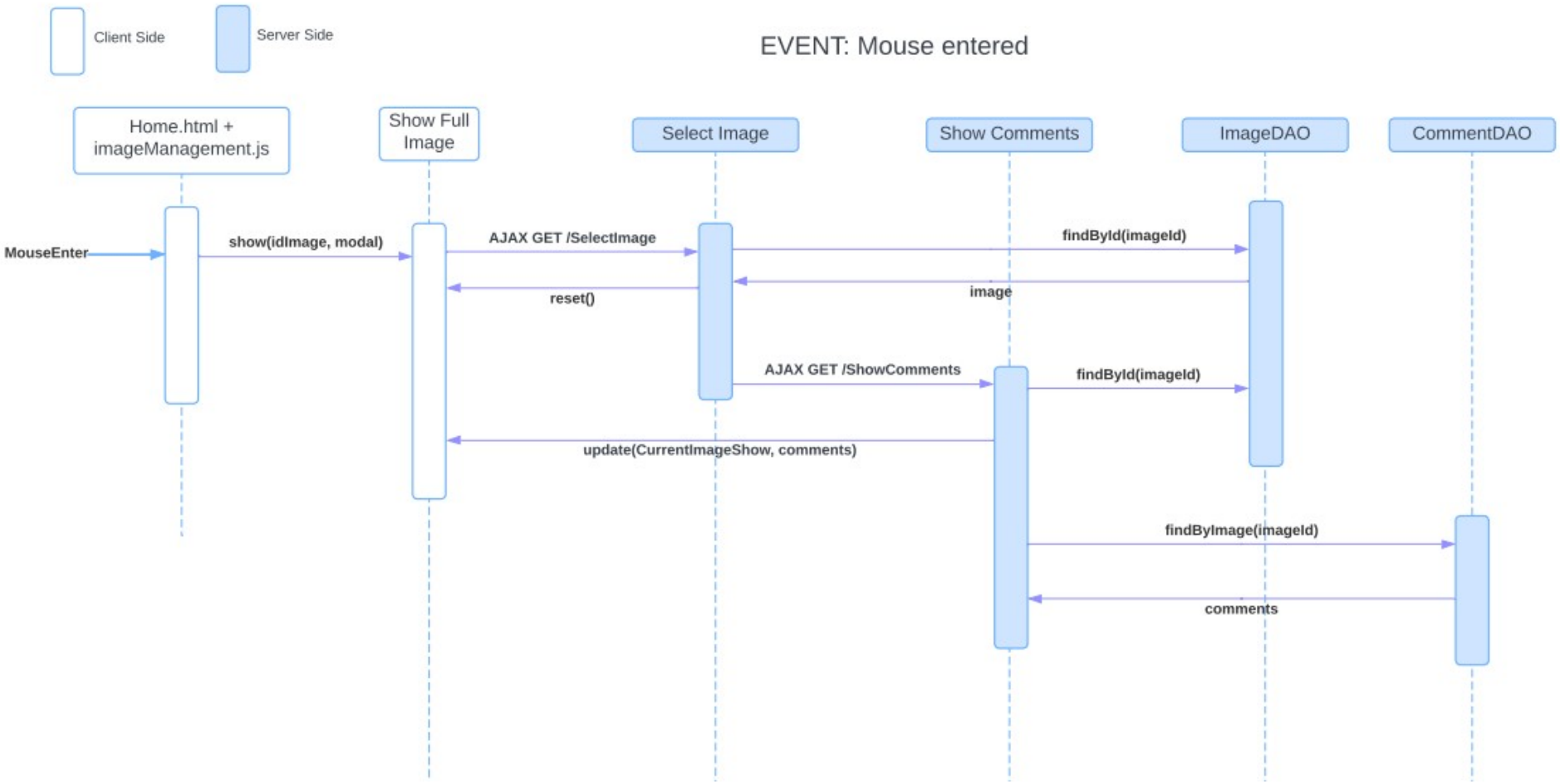
Client side: view & view component

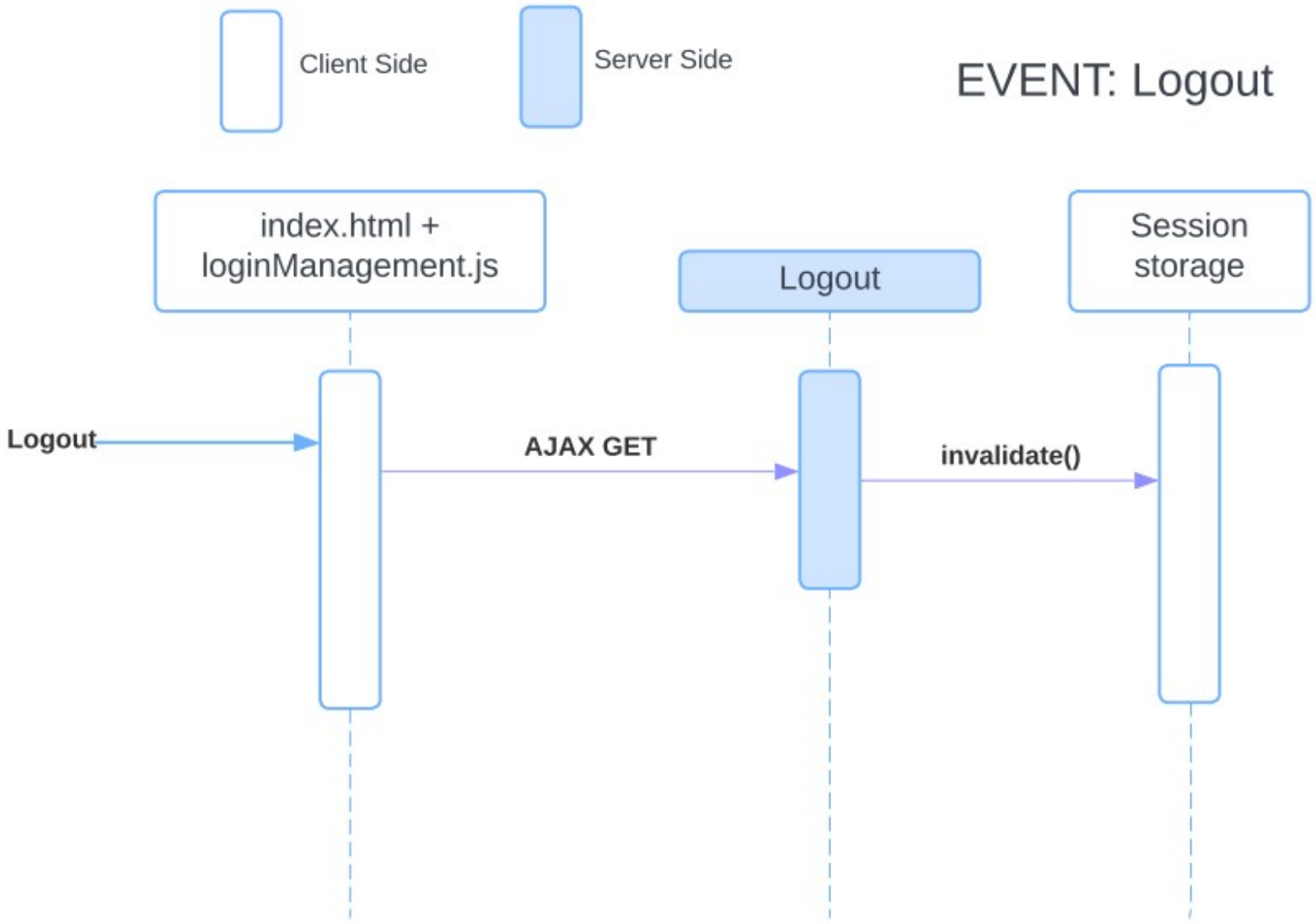
- **Index**
  - **Login form**: form per l'autenticazione dell'utente
  - **Registration form:** form per la registrazione di un nuovo utente
- **Home**
  - **UserAlbums:** tabella contenente tutti gli album creati dall'utente, con colonne titolo, creatore, data di creazione e link per accedere all'album
  - **OtherAlbums:** tabella contenente tutti gli album non creati dall'utente, con colonne titolo, creatore, data di creazione e link per accedere all'album
  - **CreateAlbum:** form per la creazione di un nuovo album. Per la creazione è necessario spuntare le caselle relative ai titoli delle immagini da includere
  - **ShowAlbum:** funzione per mostrare le immagini presenti nell'album secondo il formato di una tabella ad una riga e cinque colonne. Sono inoltre presenti i pulsanti prev, next e save order, i quali servono relativamente per spostarsi nell'album nel caso in cui abbia più di 5 immagini e salvare l'ordine delle immagini spostatili con drag and drop
  - **ShowFullImage:** funzione che permette l'apertura di una finestra modale in cui vi è presente l'immagine a grandezza naturale con relativo titolo e descrizione, la possibilità di eliminare l'immagine (solo nel caso in cui l'immagine appartenga all'utente), una tabella con tutti i commenti relativi all'immagine ed un form per inserire un nuovo commento
  - **UploadImage:** form che permette il caricamento di immagini
  - **PageOrchestrator:**
    - start(): crea e inizializza i componenti dell'interfaccia registrando i gestori degli eventi
    - Refresh(): orchestra il reperimento dei contenuti e la visualizzazione dei componenti

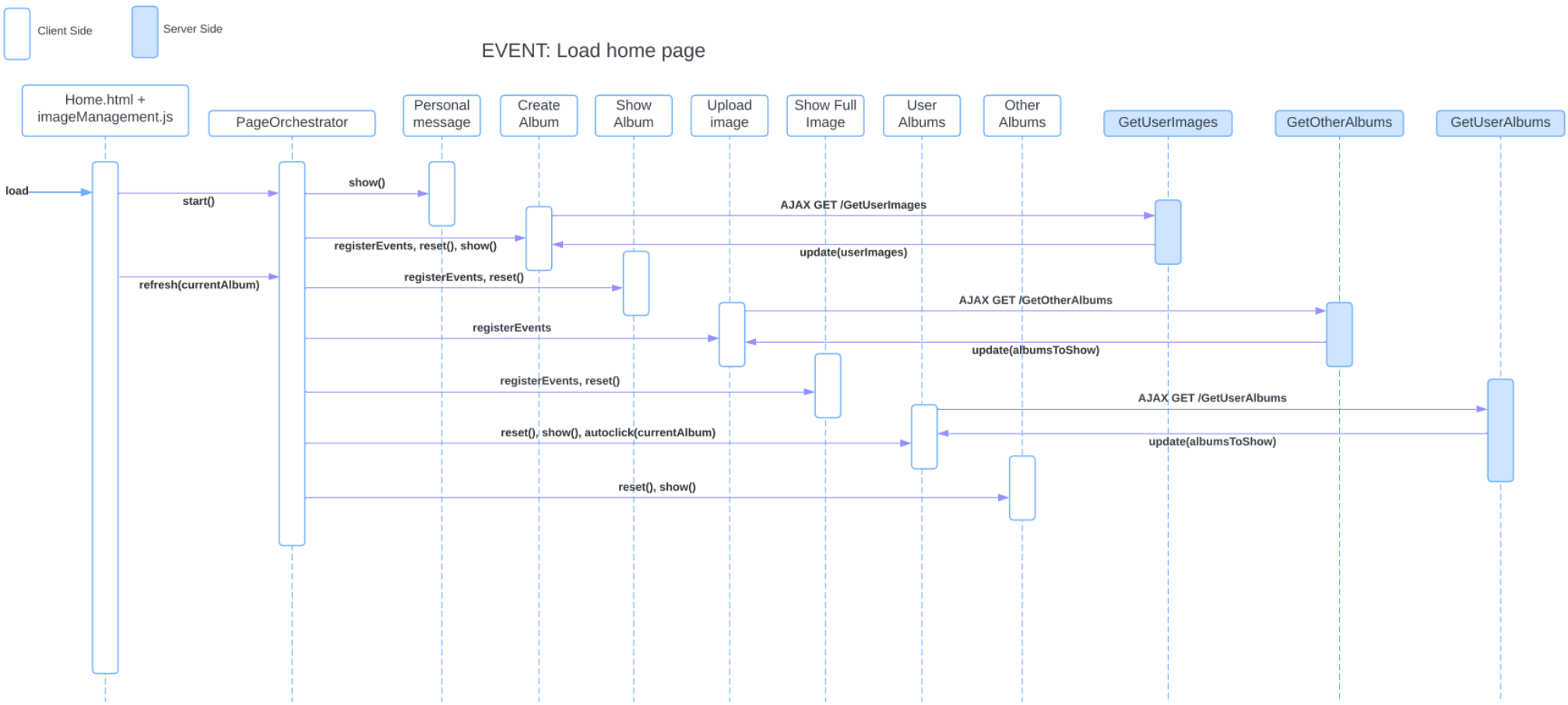

Client Side
Server Side
EVENT: Login
index.html + loginManagement.js
CheckLogin
UserDAO
Session
Session storage
Window
Click login button
AJAX POST
username
password
checkCredential
(user, pass)
user || null
[user == null || password == null]
status code 400
[user == null]
status code 401
Show error msg
[user != null]
status code 200
[user != null]
setAttribute("user",user)
Set username
setItem("username",username)
[location = Home.html]

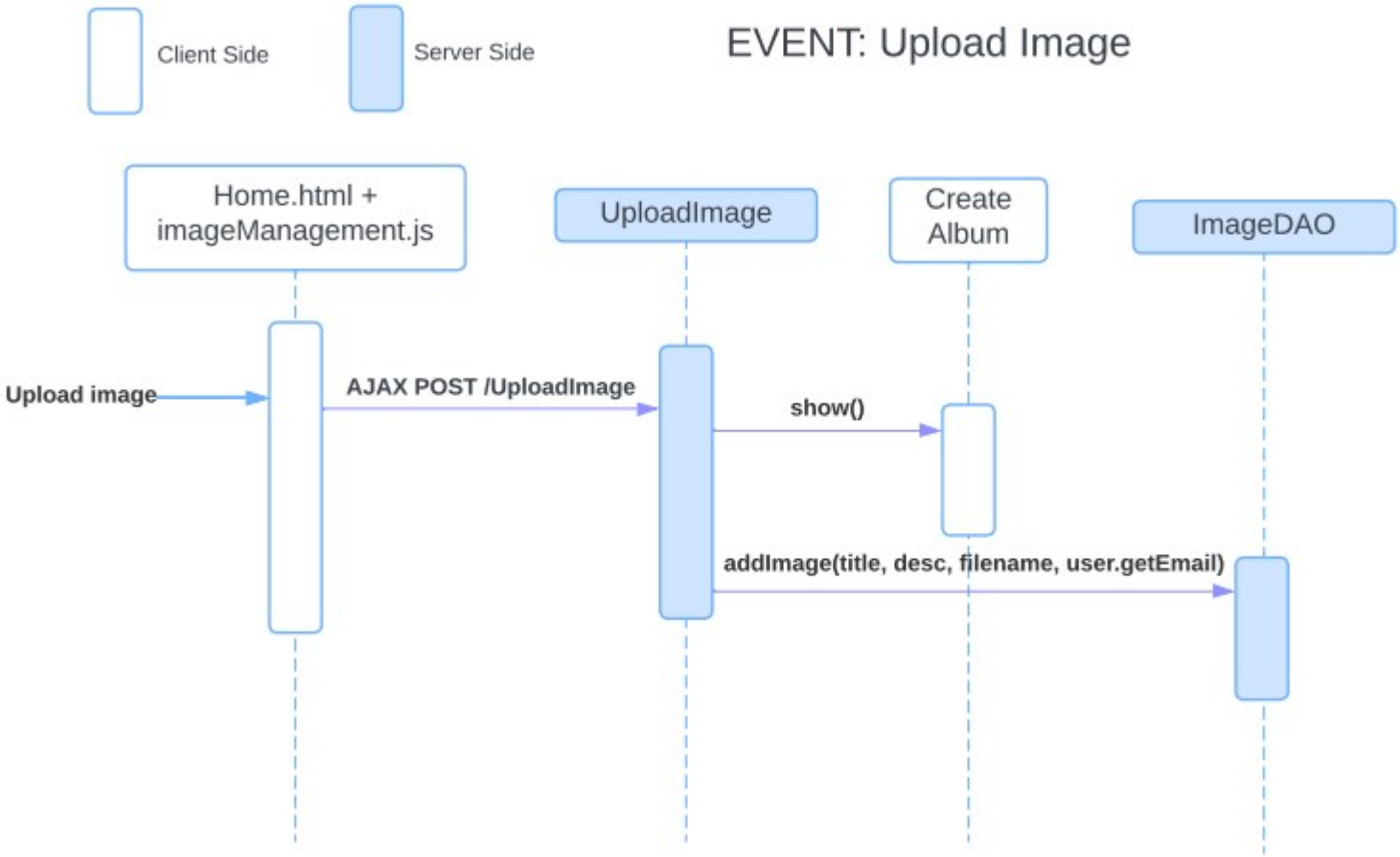

Client Side
Server Side
EVENT: Registration
index.html + loginManagement.js
CheckRegistration
UserDAO
Session
Click register button
AJAX POST
name
surname
username
password
ripPassword
checkCredential
(user, pass)
user || null
[name == null ||
surname == null ||
username == null ||
password == null ||
ripPassword == null]
status code 400
addUser
(username, password, name, surname)
Show msg
[username != null ||
(notAnEmail) ||
password!=ripPassword
status code 400
[user != null]
setAttribute("user",user)
status code 200

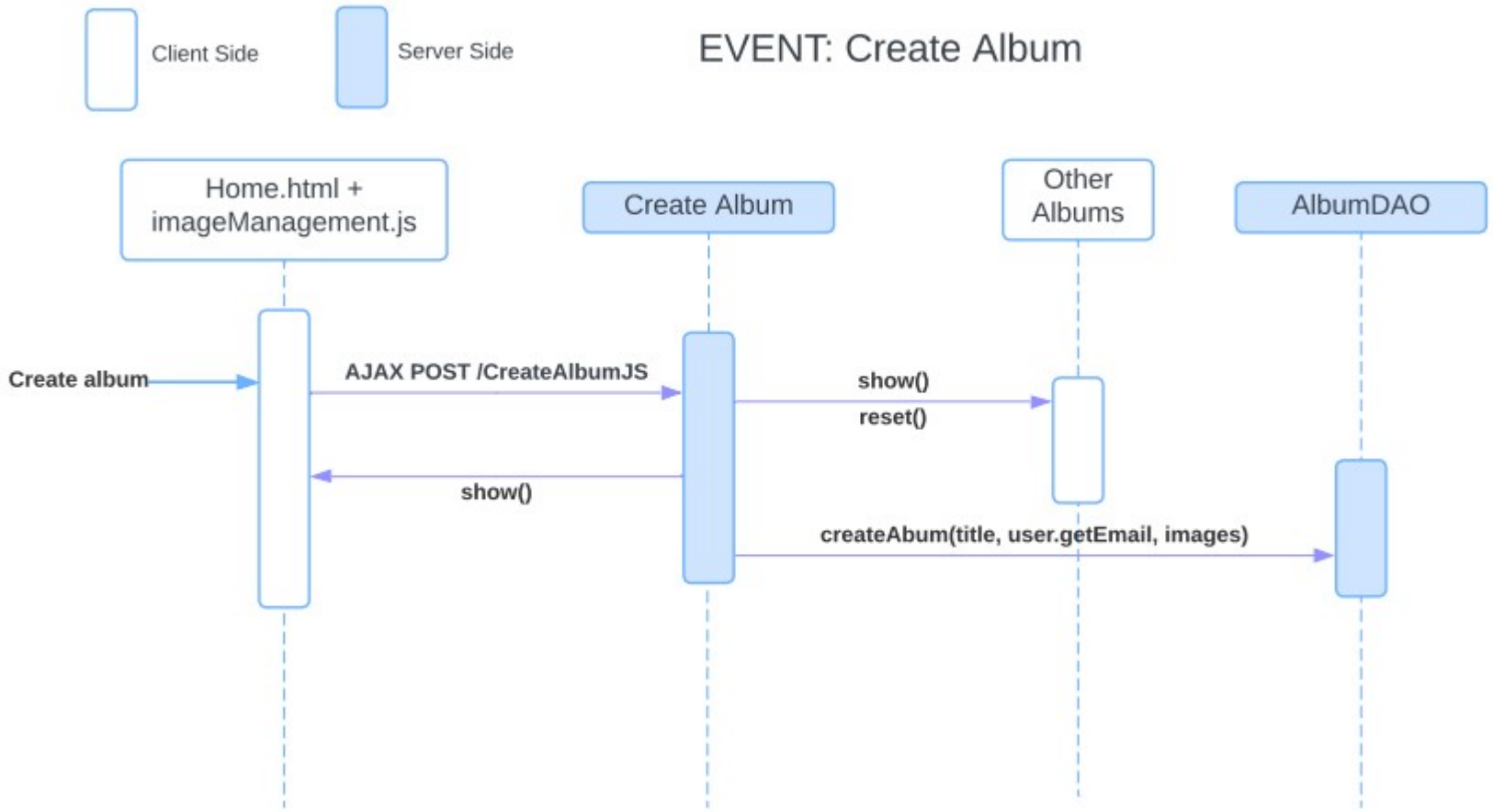

Client Side
Server Side
EVENT: Show Album
Home.html + imageManagement.js
Show Album
GetAlbumInfo
SelectAlbum
AlbumDAO
ImagesDAO
Show
show(albumId)
AJAX GET /GetAlbumInfo
findById(albumId)
album
reset()
albumCreator, albumTitle
AJAX GET /SelectAlbum
findByAlbum(albumId)
Images
Update
imagesAlbum

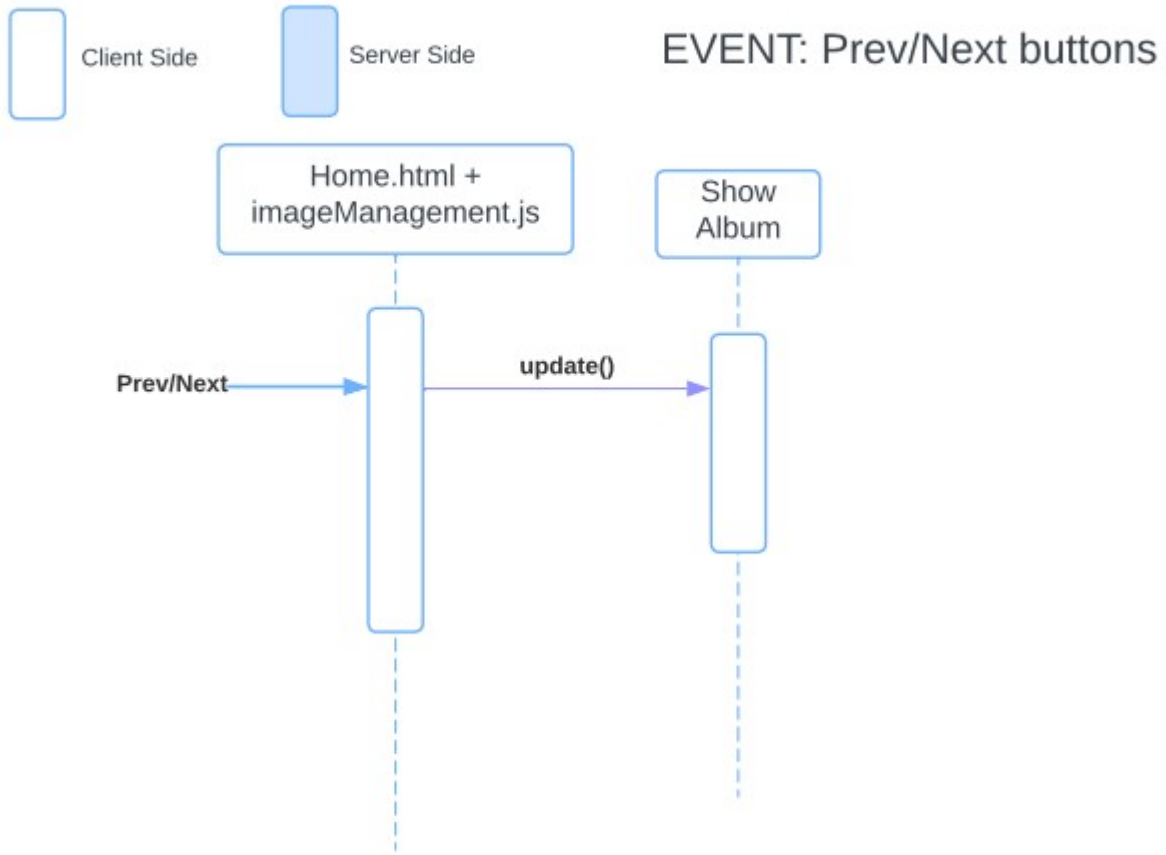

Client Side
Server Side
EVENT: Mouse entered
Home.html + imageManagement.js
Show Full Image
Select Image
Show Comments
ImageDAO
CommentDAO
MouseEnter
show(idImage, modal)
AJAX GET /SelectImage
findById(imageId)
image
reset()
AJAX GET /ShowComments
findById(imageId)
update(CurrentImageShow, comments)
findByImage(imageId)
comments

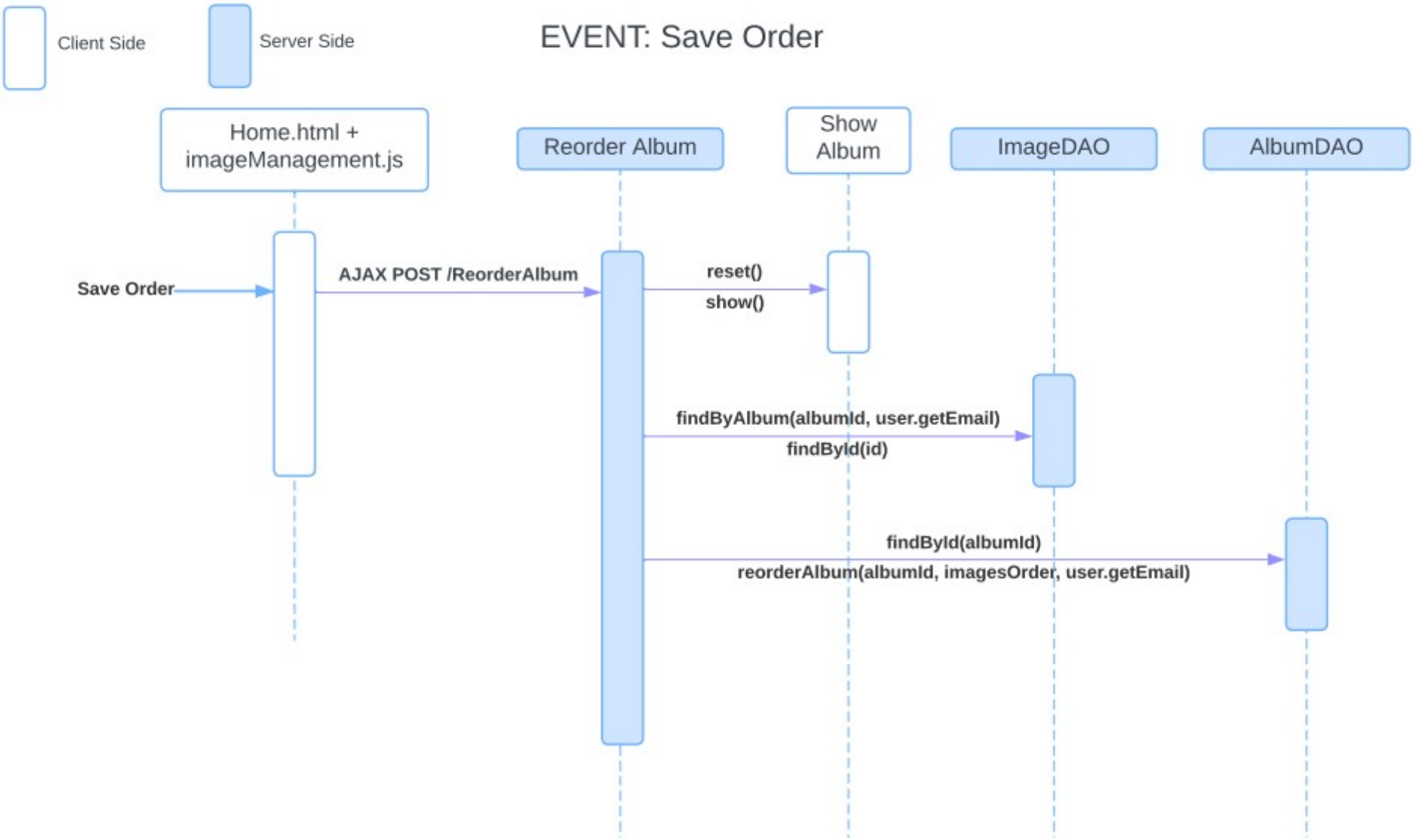

Client Side
Server Side
EVENT: Logout
index.html +
loginManagement.js
Logout
Session
storage
Logout
AJAX GET
invalidate()

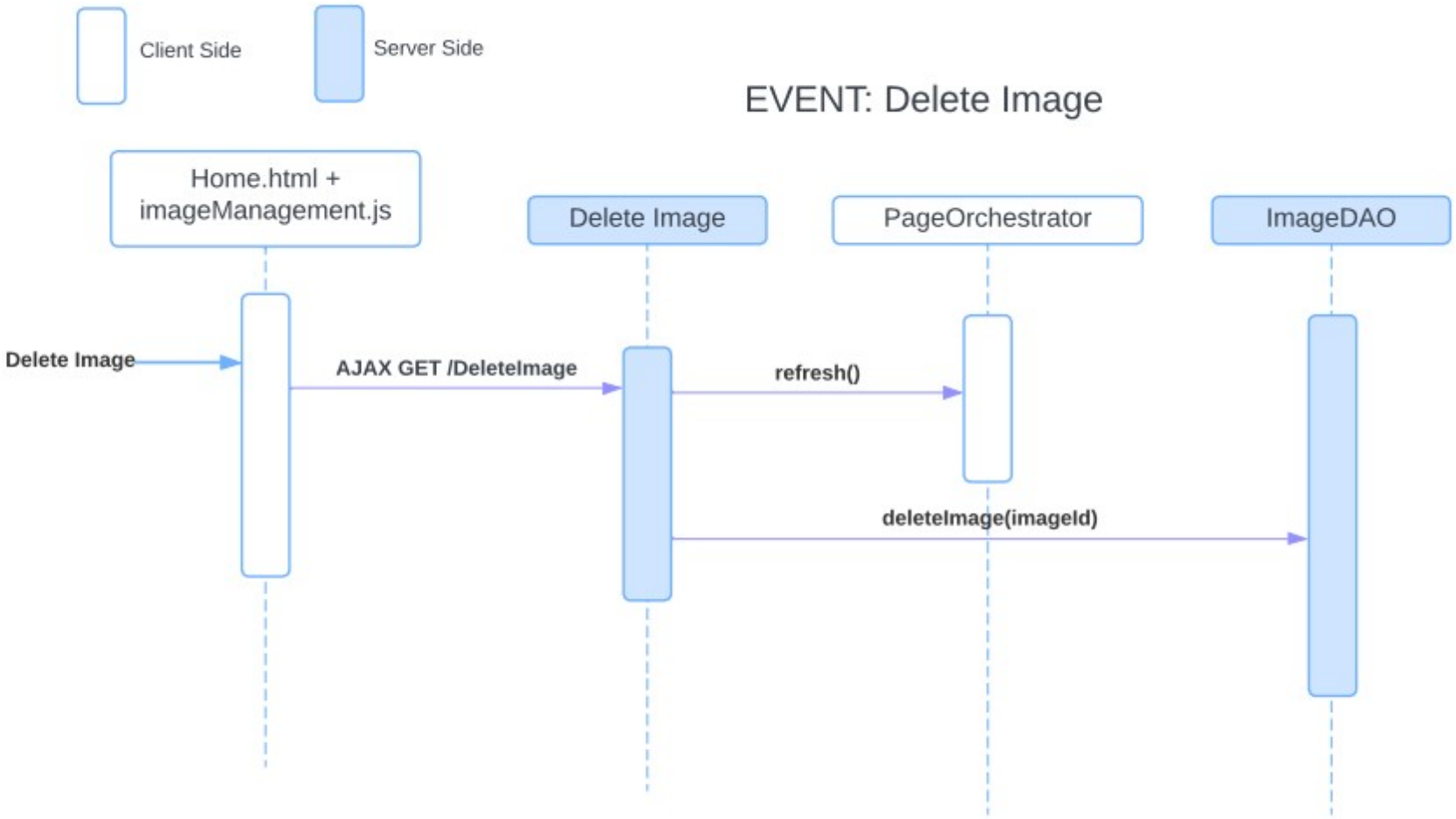

Client Side
Server Side
EVENT: Load home page
Home.html + imageManagement.js
PageOrchestrator
Personal message
Create Album
Show Album
Upload image
Show Full Image
User Albums
Other Albums
GetUserImages
GetOtherAlbums
GetUserAlbums
load
start()
show()
AJAX GET /GetUserImages
registerEvents, reset(), show()
update(userImages)
refresh(currentAlbum)
registerEvents, reset()
AJAX GET /GetOtherAlbums
registerEvents
update(albumsToShow)
registerEvents, reset()
AJAX GET /GetUserAlbums
reset(), show(), autoclick(currentAlbum)
update(albumsToShow)
reset(), show()

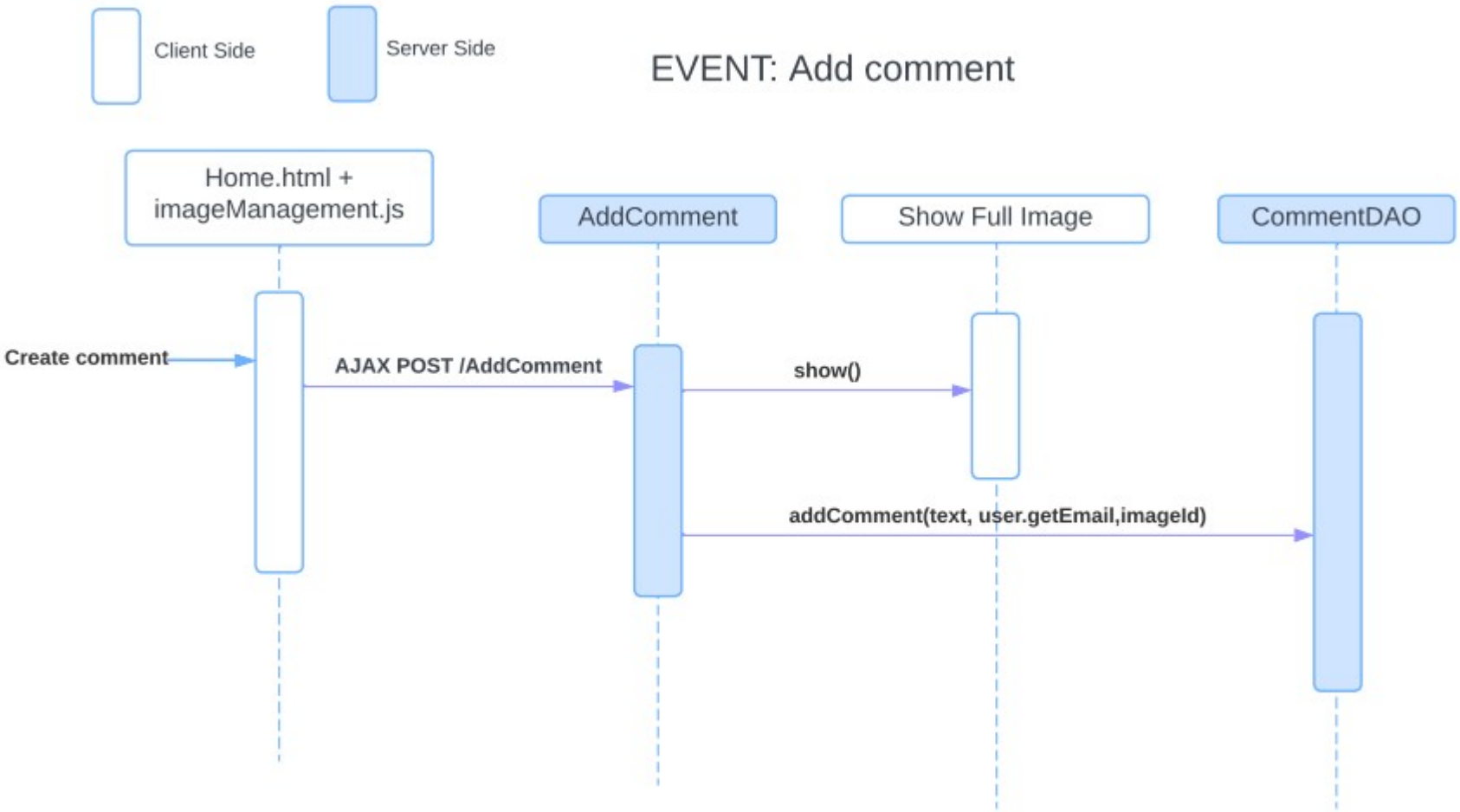

Client Side
Server Side
EVENT: Upload Image
Home.html + imageManagement.js
UploadImage
Create Album
ImageDAO
Upload image
AJAX POST /UploadImage
show()
addImage(title, desc, filename, user.getEmail)


Client Side
Server Side
EVENT: Create Album
Home.html + imageManagement.js
Create Album
Other Albums
AlbumDAO
Create album
AJAX POST /CreateAlbumJS
show()
reset()
show()
createAbum(title, user.getEmail, images)


Client Side
Server Side
EVENT: Prev/Next buttons
Home.html +
imageManagement.js
Show
Album
Prev/Next
update()


Client Side
Server Side
EVENT: Save Order
Home.html + imageManagement.js
Reorder Album
Show Album
ImageDAO
AlbumDAO
Save Order
AJAX POST /ReorderAlbum
reset()
show()
findByAlbum(albumId, user.getEmail)
findById(id)
findById(albumId)
reorderAlbum(albumId, imagesOrder, user.getEmail)


Client Side
Server Side
EVENT: Delete Image
Home.html + imageManagement.js
Delete Image
PageOrchestrator
ImageDAO
Delete Image
AJAX GET /DeleteImage
refresh()
deleteImage(imageId)


Client Side
Server Side
EVENT: Add comment
Home.html + imageManagement.js
AddComment
Show Full Image
CommentDAO
Create comment
AJAX POST /AddComment
show()
addComment(text, user.getEmail,imageId)