de Donato Simone 10677578, Donno Valerio 10709082

Esercizio 4: Trasferimento di denaro

Versione pure HTML

Analisi dei dati

Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. 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. Un utente ha un nome, un cognome, uno username e uno o più conti correnti. Un conto ha un codice, un saldo, e i trasferimenti fatti (in uscita) e ricevuti (in ingresso) dal conto. Un trasferimento ha una data, un importo, un conto di origine e un conto di destinazione. Quando l'utente accede all'applicazione appare una pagina LOGIN per la verifica delle credenziali. In seguito all'autenticazione dell'utente appare l'HOME page che mostra l'elenco dei suoi conti. Quando l'utente seleziona un conto, appare una pagina STATO DEL CONTO che mostra i dettagli del conto e la lista dei movimenti in entrata e in uscita, ordinati per data discendente. La pagina contiene anche una form per ordinare un trasferimento. La form contiene i campi: codice utente destinatario, codice conto destinatario, causale e importo. All'invio della form con il bottone INVIA, l'applicazione controlla che il conto di destinazione appartenga all'utente specificato e che il conto origine abbia un saldo superiore o uguale all'importo del trasferimento. In caso di mancanza di anche solo una condizione, l'applicazione mostra una pagina con un avviso di fallimento che spiega il motivo del mancato trasferimento.
Nel caso in cui entrambe le condizioni siano soddisfatte, l'applicazione deduce l'importo dal conto di origine, aggiunge l'importo al conto di destinazione e mostra una pagina CONFERMA TRASFERIMENTO che presenta i dati dell'importo trasferito e i dati del conto di origine e di destinazione con i rispettivi saldi precedenti al trasferimento e aggiornati dopo il trasferimento. L'applicazione deve garantire l'atomicità del trasferimento: ogni volta che il conto di destinazione viene addebitato, il conto di origine deve essere accreditato. Ogni pagina contiene un collegamento per tornare alla pagina precedente. L'applicazione consente il logout dell'utente.

[Entities, attributes, relationships]

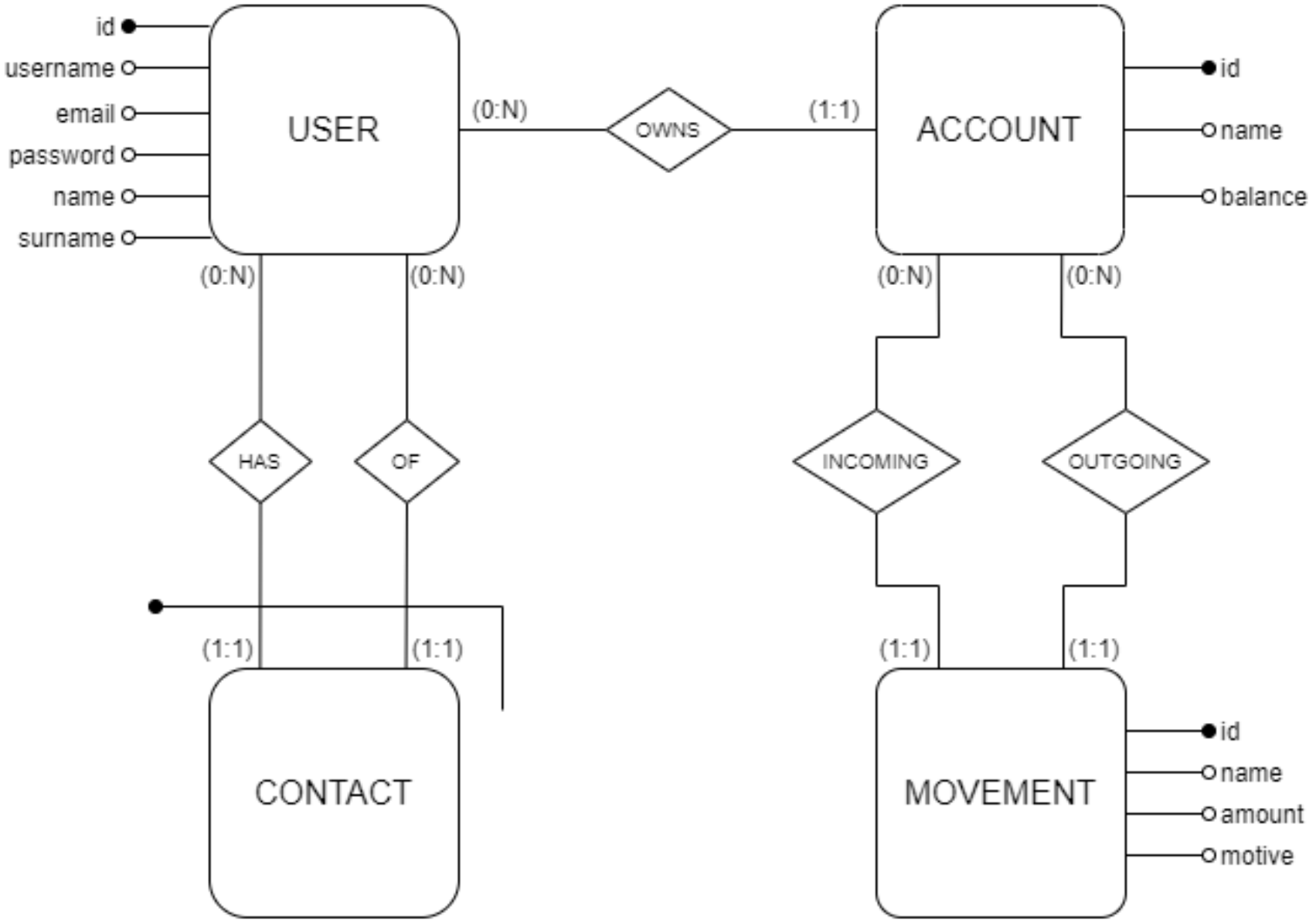
Al fine di progettare una singola base di dati per entrambe le applicazioni richieste, di seguito si analizzano anche i dati richiesti dalla versione con JavaScript.

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

- La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza tra i campi "password" e "ripeti password", anche a lato client.
- Dopo il login, 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.
- I controlli di validità dei dati di input (ad esempio importo non nullo e maggiore di zero) devono essere realizzati anche a lato client.
- L'avviso di fallimento è realizzato mediante un messaggio nella pagina che ospita l'applicazione.
- L'applicazione chiede all'utente se vuole inserire nella propria rubrica i dati del destinatario di un trasferimento andato a buon fine non ancora presente. Se l'utente conferma, i dati sono memorizzati nella base di dati e usati per semplificare l'inserimento. Quando l'utente crea un trasferimento, l'applicazione propone mediante una funzione di auto-completamento i destinatari in rubrica il cui codice corrisponde alle lettere inserite nel campo codice utente destinatario.

[Entities, attributes, relationships]

Schema ER database



L'entità "CONTACT" realizza la rubrica delle specifiche. La rubrica è pensata come una relazione N:M tra uno user e i suoi user in rubrica. Questa relazione N:M è tradotta tramite l'entità contact.

Schema logico database

```
CREATE TABLE `user` (
  `id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `name` varchar(45) NOT NULL,
  `surname` varchar(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `email_UNIQUE` (`email`),
  UNIQUE KEY `username_UNIQUE` (`username`)
)
```

```
CREATE TABLE `account` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `balance` decimal(12,2) NOT NULL,
  `userid` int NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name_UNIQUE` (`name`),
  KEY `userid_idx` (`userid`),
  CONSTRAINT `userid` FOREIGN KEY (`userid`) REFERENCES `user` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE
)
```

```
CREATE TABLE `movement` (
  `id` int NOT NULL AUTO_INCREMENT,
  `date` timestamp NOT NULL,
  `amount` decimal(12,2) NOT NULL,
  `motive` varchar(200) NOT NULL,
  `inAccountID` int NOT NULL,
  `outAccountID` int NOT NULL,
  PRIMARY KEY (`id`),
  KEY `inAccountid_idx` (`inAccountID`),
  KEY `outAccountid_idx` (`outAccountID`),
  CONSTRAINT `inAccountid` FOREIGN KEY (`inAccountID`) REFERENCES `account` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `outAccountid` FOREIGN KEY (`outAccountID`) REFERENCES `account` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE
)
CREATE TABLE `contact` (
  `ownerUserid` int NOT NULL,
  `contactUserid` int NOT NULL,
  PRIMARY KEY (`ownerUserid`,`contactUserid`),
  KEY `contactUserid_idx` (`contactUserid`),
  CONSTRAINT `contactUserid` FOREIGN KEY (`contactUserid`) REFERENCES `user` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `ownerUserid` FOREIGN KEY (`ownerUserid`) REFERENCES `user` (`id`)
    ON DELETE CASCADE ON UPDATE CASCADE
)
```

Analisi requisiti dell'applicazione

Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. 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. Un utente ha un nome, un cognome, uno username e uno o più conti correnti. Un conto ha un codice, un saldo, e i trasferimenti fatti (in uscita) e ricevuti (in ingresso) dal conto. Un trasferimento ha una data, un importo, un conto di origine e un conto di destinazione. Quando l'utente accede all'applicazione appare una pagina LOGIN per la verifica delle credenziali *(i.e. invio della form)*. In seguito all'autenticazione dell'utente appare l'HOME page che mostra l'elenco dei suoi conti. Quando l'utente seleziona un conto, appare una pagina STATO DEL CONTO che mostra i dettagli del conto e la lista dei movimenti in entrata e in uscita, ordinati per data discendente. La pagina contiene anche una form per ordinare un trasferimento. La form contiene i campi: codice utente destinatario, codice conto destinatario, causale e importo. All'invio della form con il bottone INVIA, l'applicazione controlla che il conto di destinazione appartenga all'utente specificato e che il conto origine abbia un saldo superiore o uguale all'importo del trasferimento. In caso di mancanza di anche solo una condizione, l'applicazione mostra una pagina con un avviso di fallimento che spiega il motivo del mancato trasferimento.

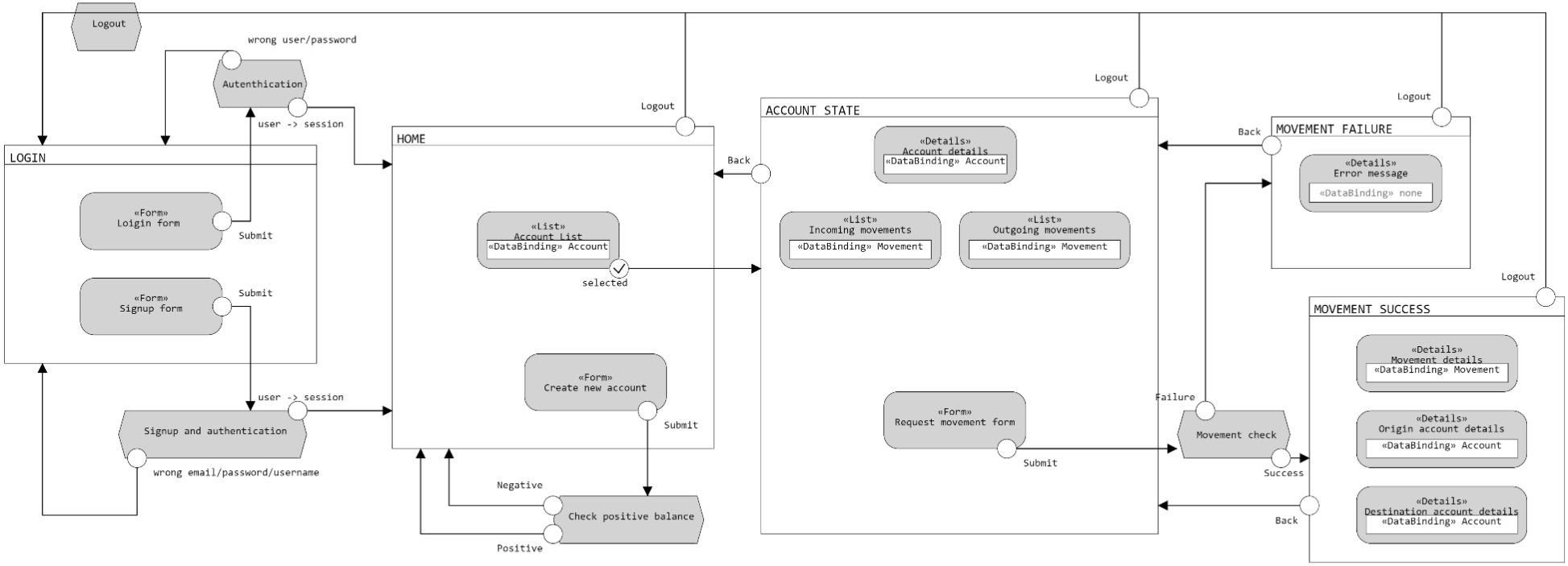
Nel caso in cui entrambe le condizioni siano soddisfatte, l'applicazione deduce l'importo dal conto di origine, aggiunge l'importo al conto di destinazione e mostra una pagina CONFERMA TRASFERIMENTO che presenta i dati dell'importo trasferito e i dati del conto di origine e di destinazione con i rispettivi saldi precedenti al trasferimento e aggiornati dopo il trasferimento. L'applicazione deve garantire l'atomicità del trasferimento: ogni volta che il conto di destinazione viene addebitato, il conto di origine deve essere accreditato. Ogni pagina contiene un collegamento per tornare alla pagina precedente. L'applicazione consente il logout dell'utente.

[Pages (views), view components, events, actions]

Completamento delle specifiche

- La pagina di login contiene form sia per la registrazione che per il login
- L'invio del form di registrazione causa, oltra alla creazione di un nuovo utente, anche l'autenticazione, e l'utente è portato alla pagina HOME
- L'azione di logout è concessa da ogni pagina (eccetto il LOGIN) e, in seguito al logout, riporta l'utente alla pagina di LOGIN
- L'azione di ritorno alla pagina precedente e logout coincidono nella pagina HOME
- La pagina HOME contiene un form per creare un nuovo conto all'invio della form con un saldo iniziale arbitrario positivo
- Qualora si tentasse l'accesso a una risorsa a cui non si ha diritto, si verrà reindirizzati alla pagina di LOGIN (se l'utente non è loggato) o alla HOME (se invece è loggato) con un messaggio di errore che spiega il motivo del reindirizzamento

IFML Design


Logout
wrong user/password
Autenthication
user -> session
LOGIN
«Form»
Loigin form
Submit
«Form»
Signup form
Submit
user -> session
Signup and authentication
wrong email/password/username
Logout
HOME
Back
«List»
Account list
«DataBinding» Account
selected
«Form»
Create new account
Submit
Negative
Check positive balance
Positive
Logout
ACCOUNT STATE
«Details»
Account details
«DataBinding» Account
«List»
Incoming movements
«DataBinding» Movement
«List»
Outgoing movements
«DataBinding» Movement
«Form»
Request movement form
Submit
Logout
Back
MOVEMENT FAILURE
«Details»
Error message
«DataBinding» none
Logout
MOVEMENT SUCCESS
«Details»
Movement details
«DataBinding» Movement
«Details»
Origin account details
«DataBinding» Account
«Details»
Destination account details
«DataBinding» Account
Failure
Movement check
Success
Back

Components

Model object (Beans)

- User
- Account
- Movement

(N.B.: Il contact non è un Model Object in quanto è semplicemente una lista di User posseduta da uno User. La funzionalità per ottenere questa lista non serve nell'applicazione pure HTML e quindi è omessa dalle funzioni dei DAO)

Data Access Objects (Classes)

- UserDAO
  - checkCredentials(username, passwrod)
  - registerUser(name, surname, username, email, password)
  - getUserByID(userID)
  - getUserByUsername(username)
  - getUserByEmail(email)
- AccountDAO
  - getAccountsByUser(userID)
  - getAccountByID(accountID)
  - getAccountByName(name)
  - createAccount(userID, balance)
- MovementDAO
  - getIncomingMovementsByAccount(accountID)
  - getOutgoingMovementsByAccount(accountID)
  - getMovementByID(movementID)
  - requestMovement(amount, date, reason, inAccount, outAccount)

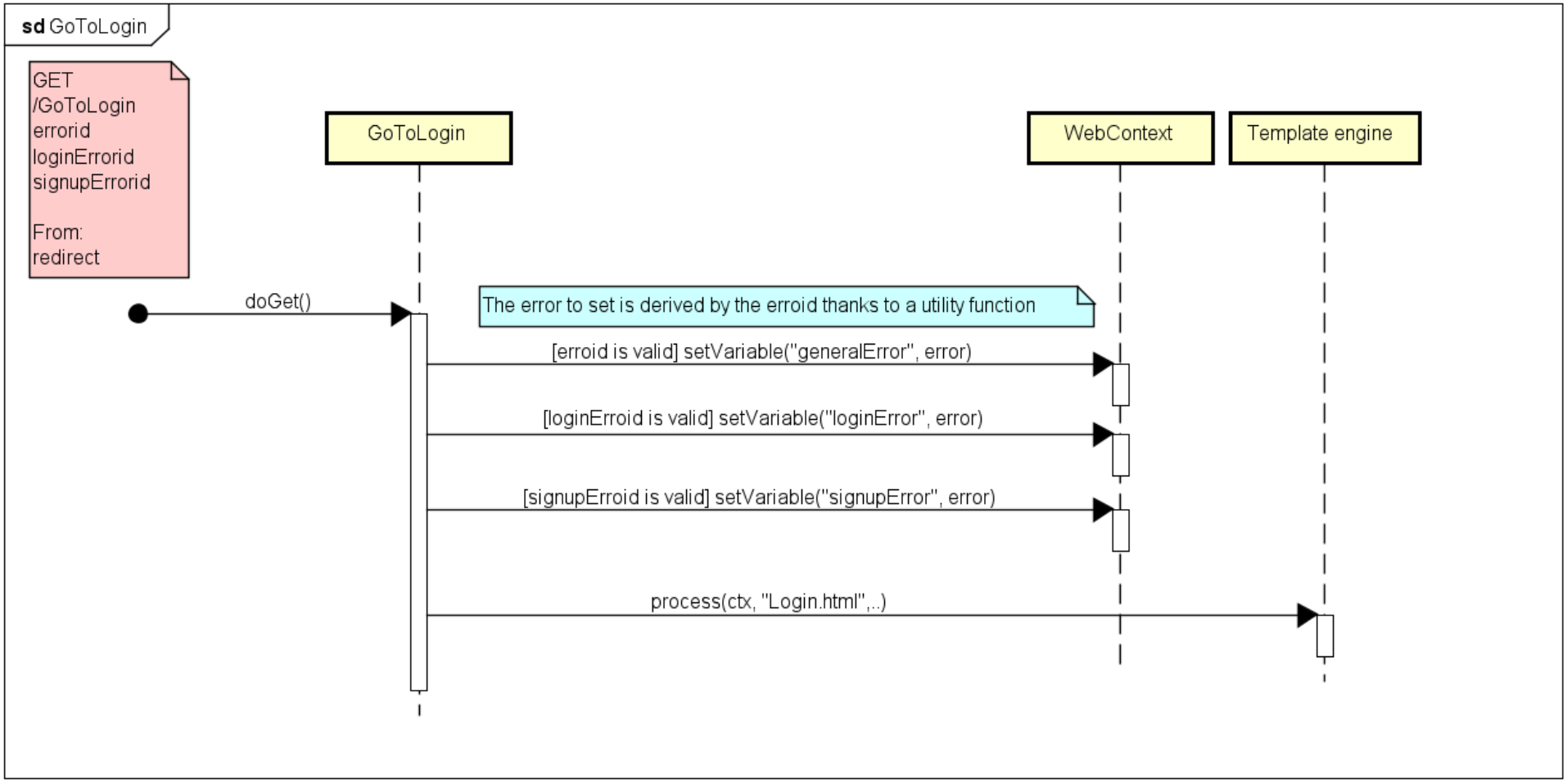
Controllers (Servlets)

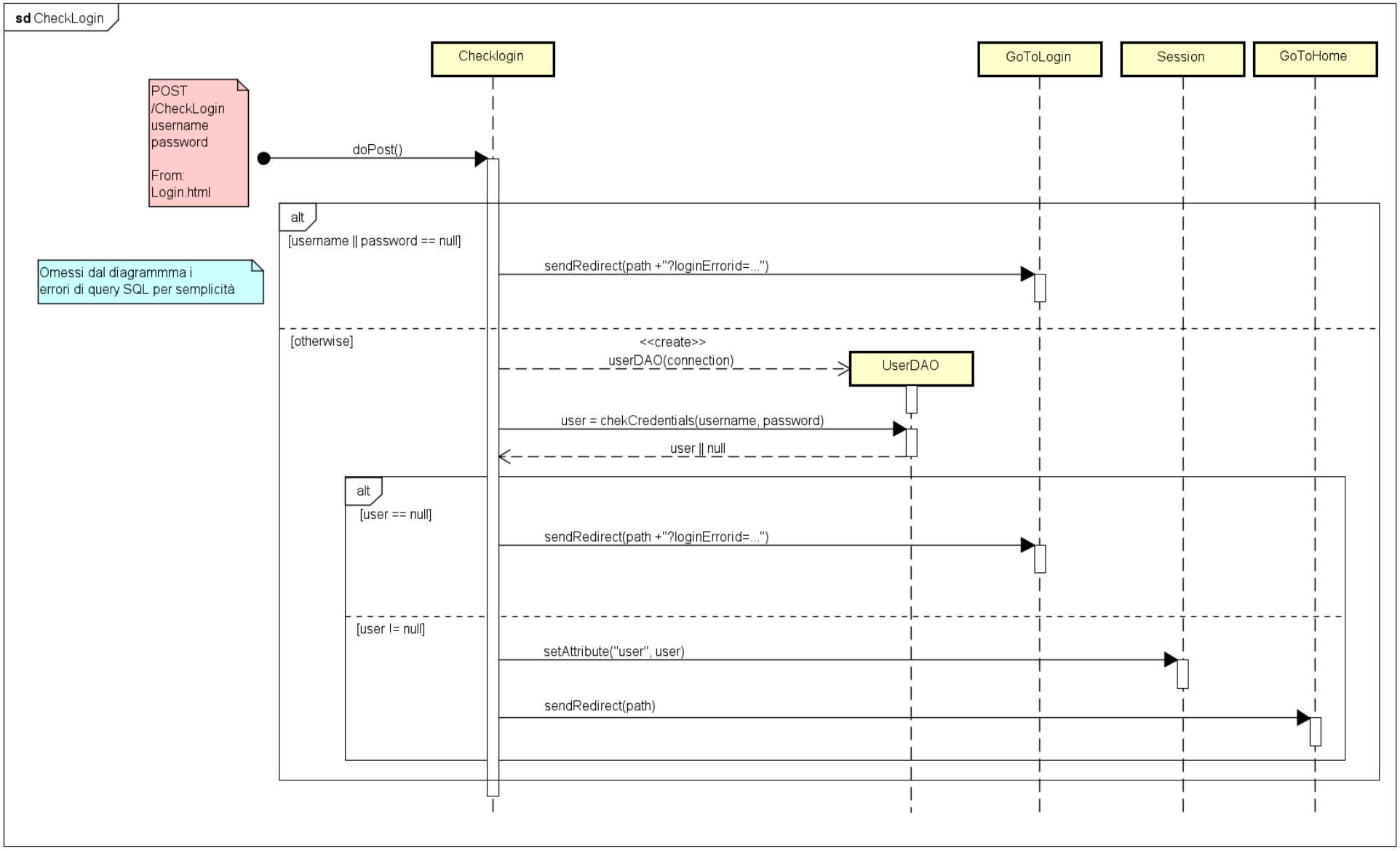
- GoToLogin
- CheckLogin
- CheckSignup
- GoToHome
- CreateAccount
- GoToAccountState
- RequestMovement
- GoToMovementFailure
- GoToMovementSuccess
- Logout

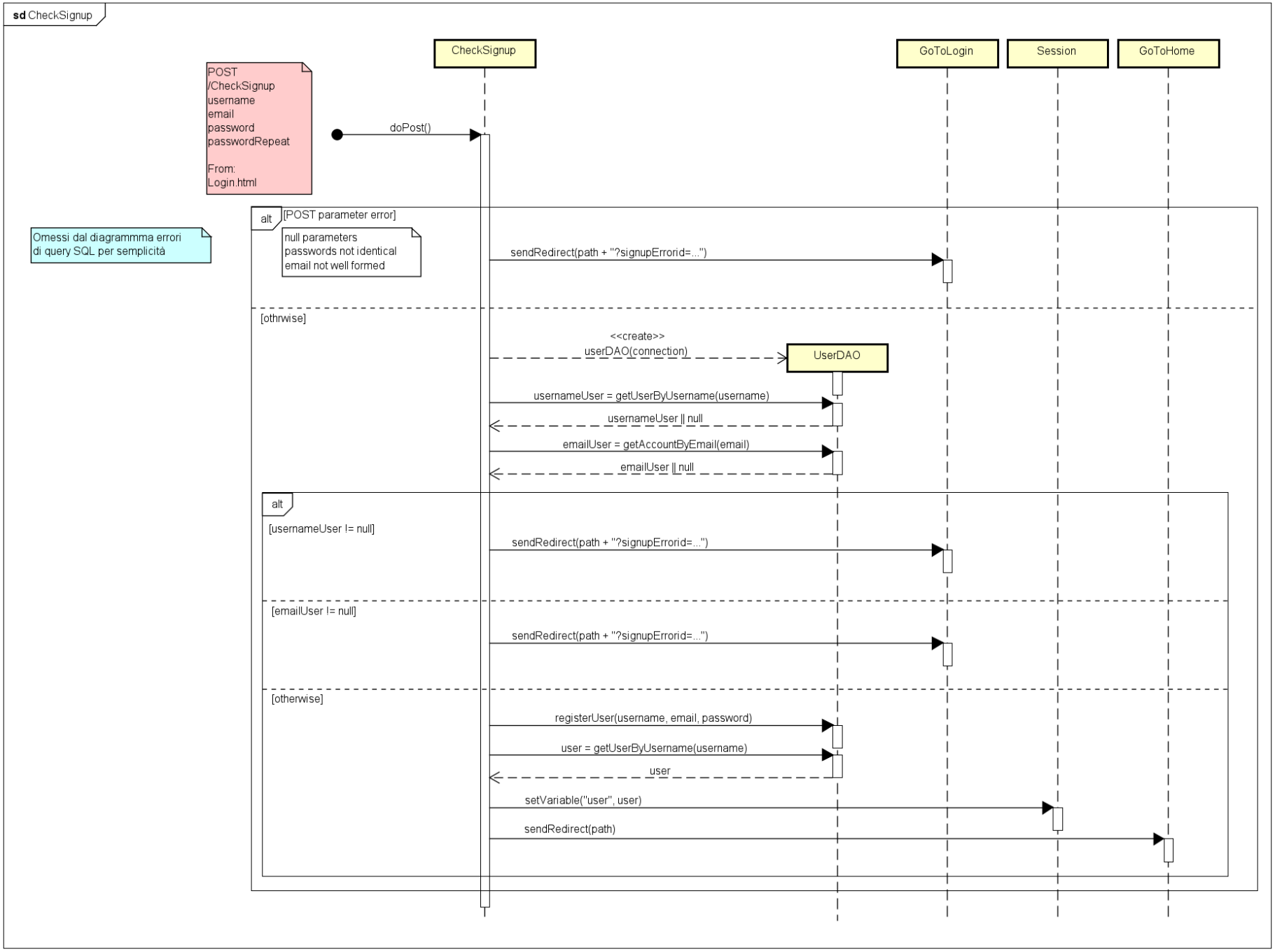
Views (Templates)

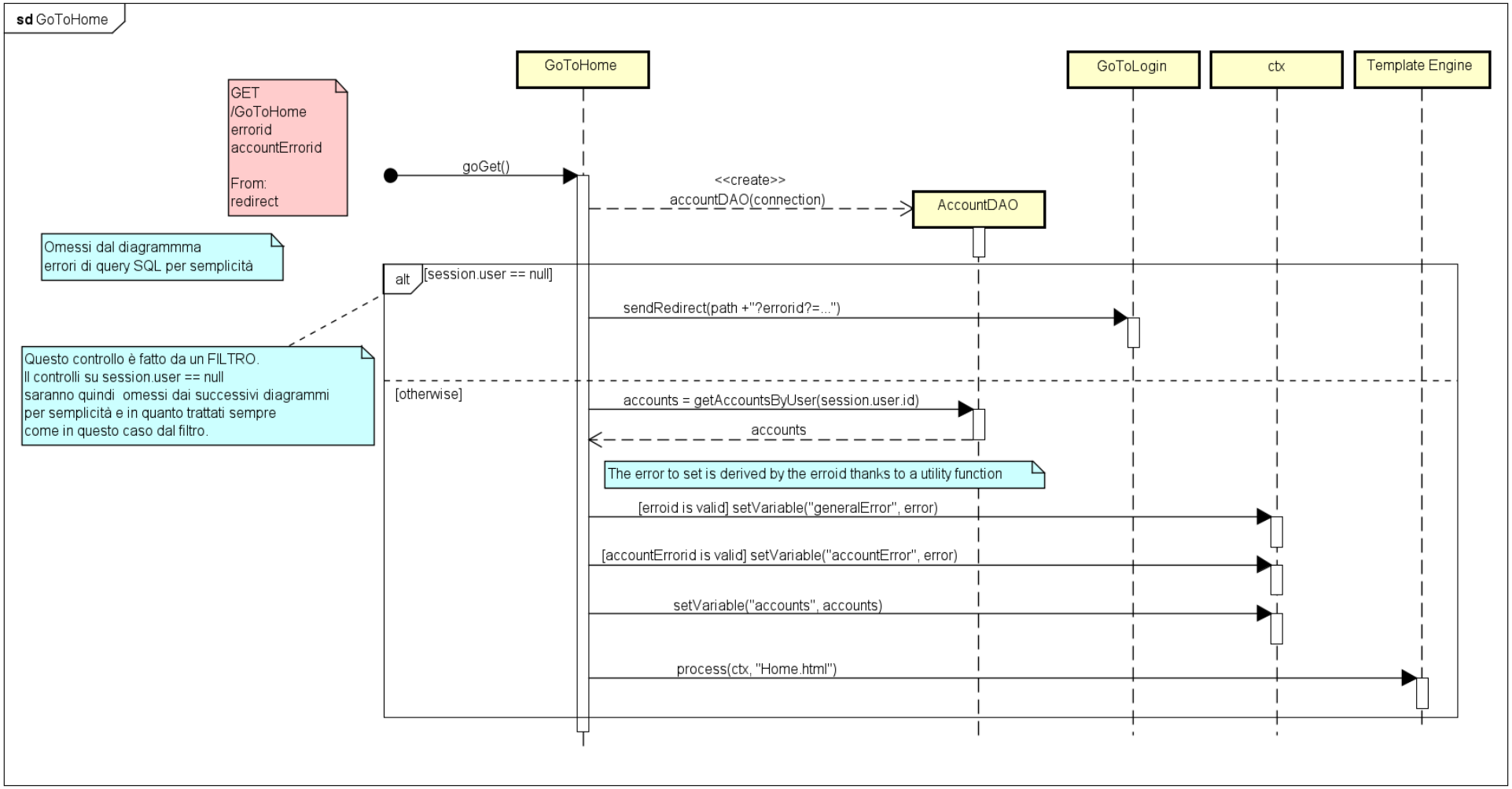
- Login
- Home
- AccountState
- MovementFailure
- MovementSuccess
- TopBar (aggiunta dinamicamente alle altre pagine)

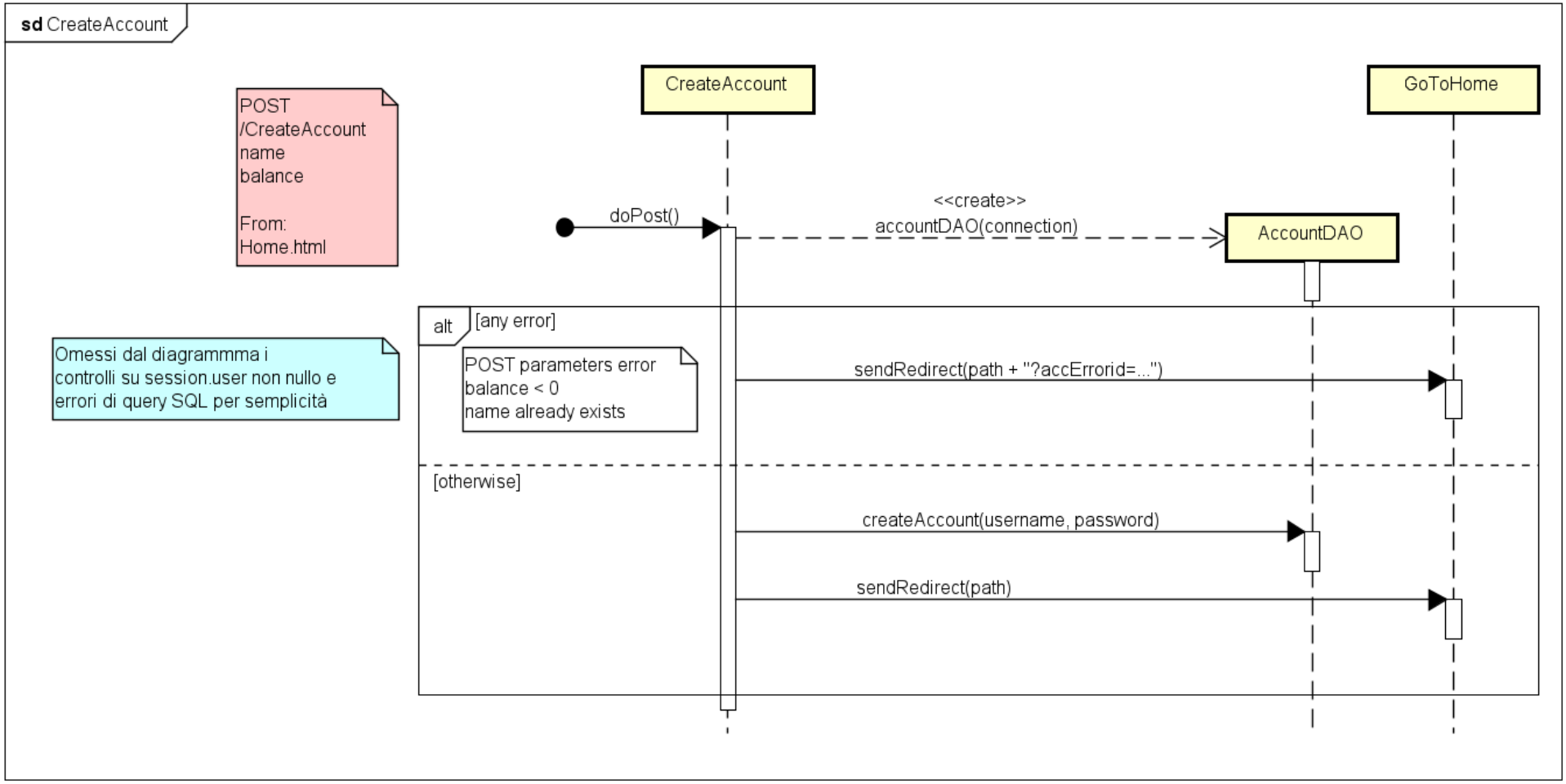
Sequence diagrams

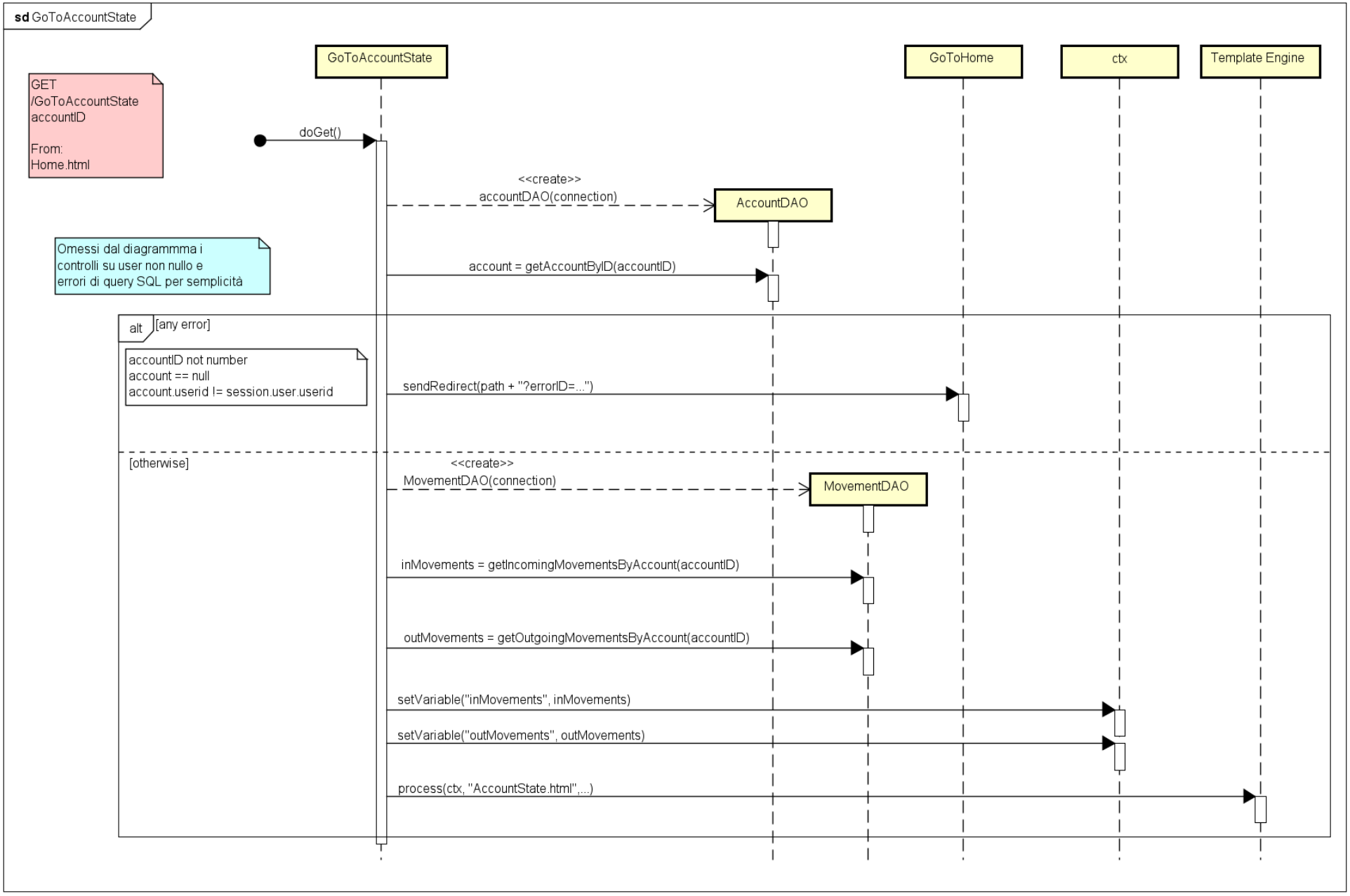


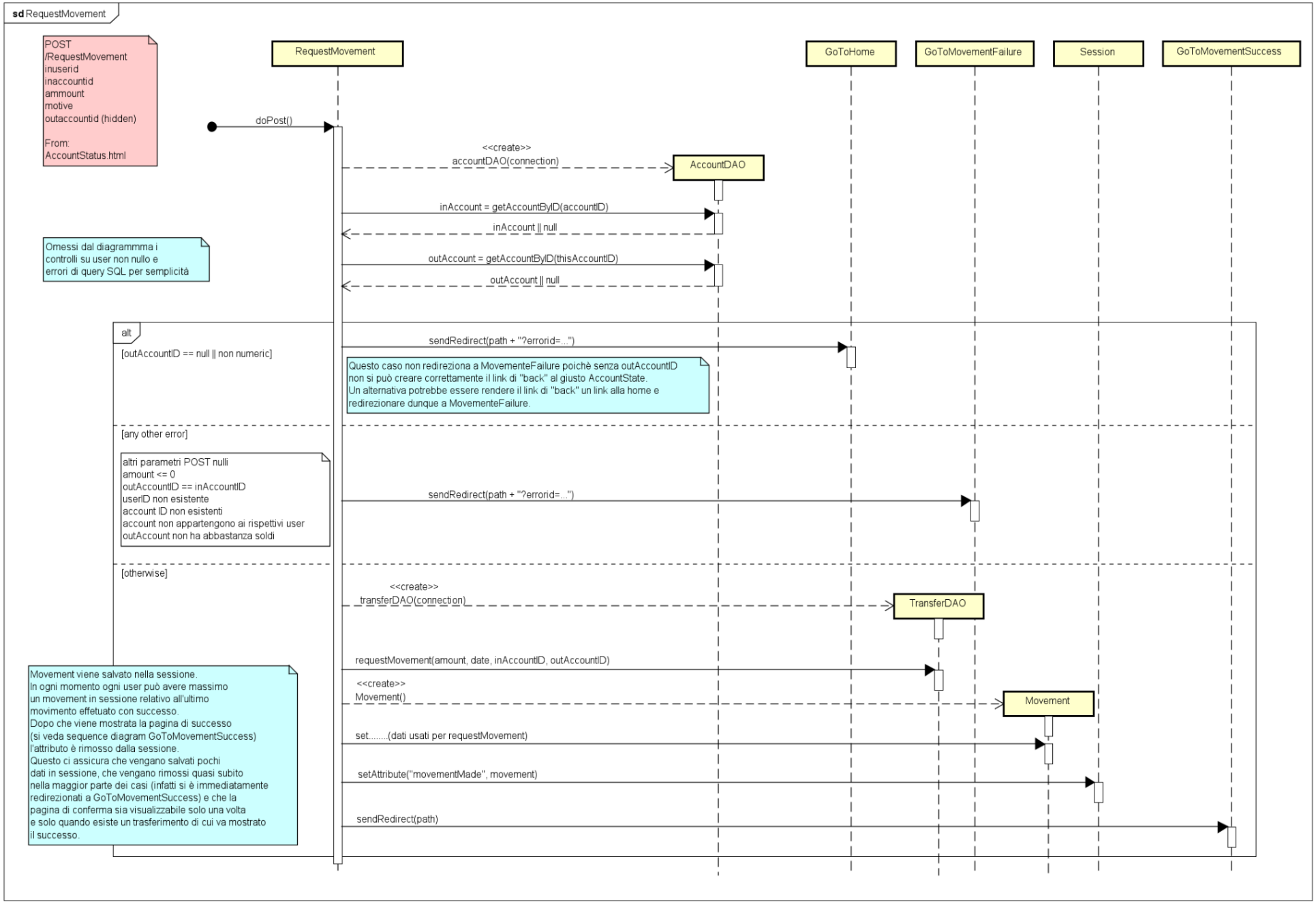

sd CheckLogin
Checklogin
GoToLogin
Session
GoToHome
POST
/CheckLogin
username
password
From:
Login.html
doPost()
alt
[username || password == null]
sendRedirect(path +"?loginErrorid=...")
Omessi dal diagrammma i
errori di query SQL per semplicità
[otherwise]
<<create>>
userDAO(connection)
UserDAO
user = chekCredentials(username, password)
user || null
alt
[user == null]
sendRedirect(path +"?loginErrorid=...")
[user != null]
setAttribute("user", user)
sendRedirect(path)

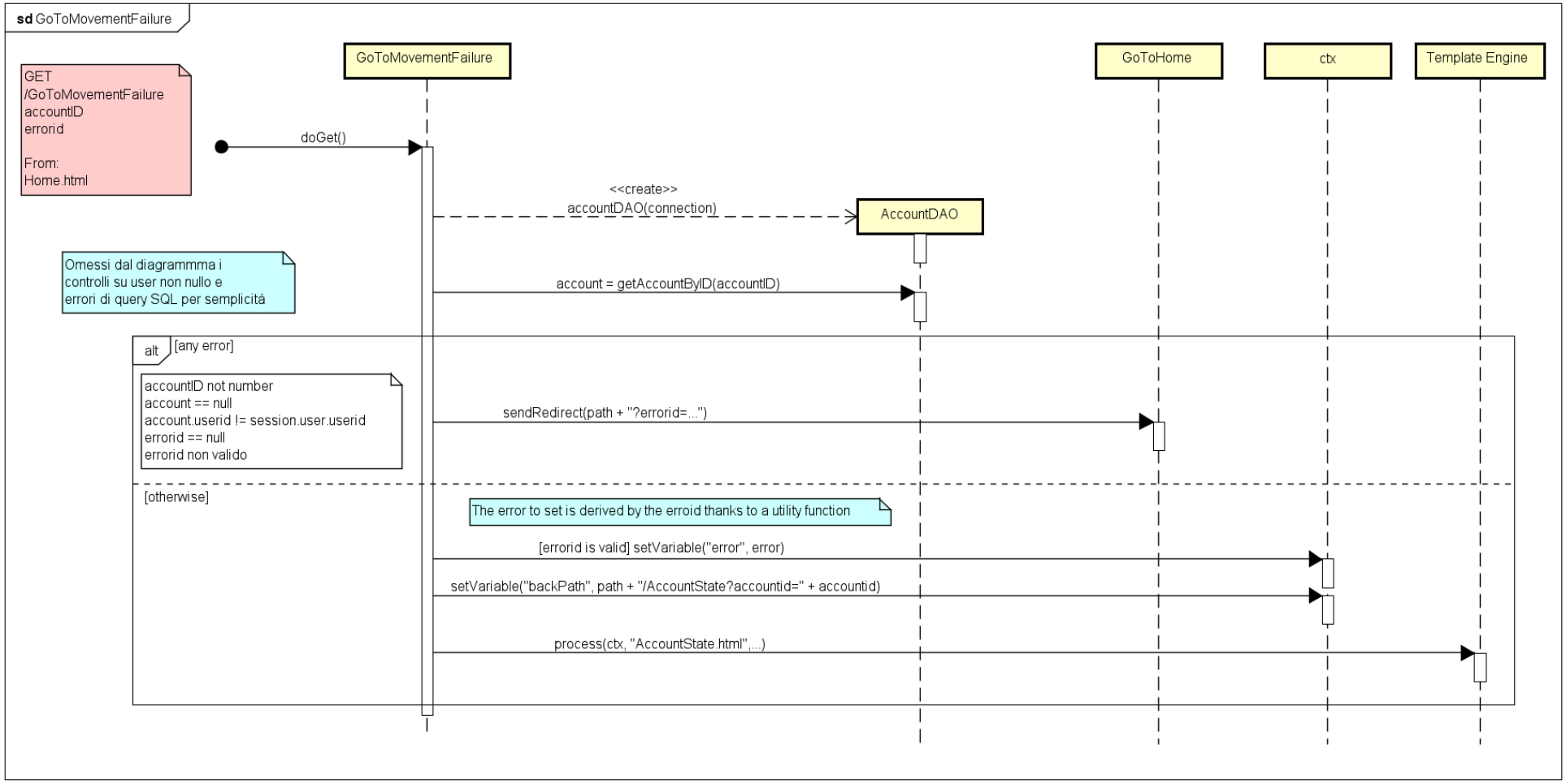

sd CheckSignup
CheckSignup
GoToLogin
Session
GoToHome
POST
/CheckSignup
username
email
password
passwordRepeat
From:
Login.html
doPost()
Omessi dal diagrammma errori
di query SQL per semplicità
alt
[POST parameter error]
null parameters
passwords not identical
email not well formed
sendRedirect(path + "?signupErrorid=...")
[othrwise]
<<create>>
userDAO(connection)
UserDAO
usernameUser = getUserByUsername(username)
usernameUser || null
emailUser = getAccountByEmail(email)
emailUser || null
alt
[usernameUser != null]
sendRedirect(path + "?signupErrorid=...")
[emailUser != null]
sendRedirect(path + "?signupErrorid=...")
[otherwise]
registerUser(username, email, password)
user = getUserByUsername(username)
user
setVariable("user", user)
sendRedirect(path)

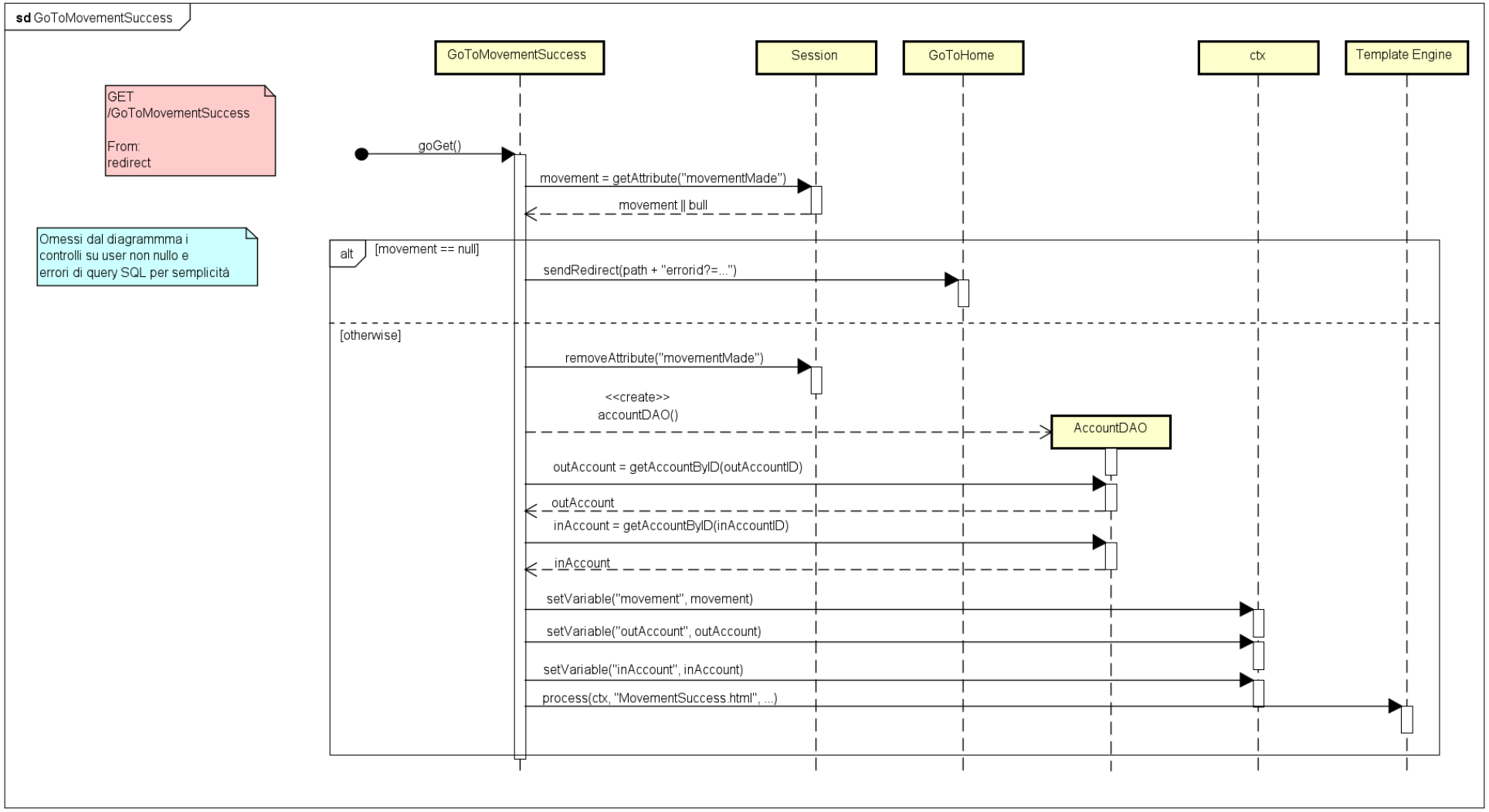

sd GoToHome
GoToHome
GoToLogin
ctx
Template Engine
GET
/GoToHome
errorid
accountErrorid
From:
redirect
goGet()
<<create>>
accountDAO(connection)
AccountDAO
Omessi dal diagrammma
errori di query SQL per semplicità
alt
[session.user == null]
sendRedirect(path +"?errorid?=...")
Questo controllo è fatto da un FILTRO.
Il controlli su session.user == null
saranno quindi omessi dai successivi diagrammi
per semplicità e in quanto trattati sempre
come in questo caso dal filtro.
[otherwise]
accounts = getAccountsByUser(session.user.id)
accounts
The error to set is derived by the erroid thanks to a utility function
[erroid is valid] setVariable("generalError", error)
[accountErrorid is valid] setVariable("accountError", error)
setVariable("accounts", accounts)
process(ctx, "Home.html")

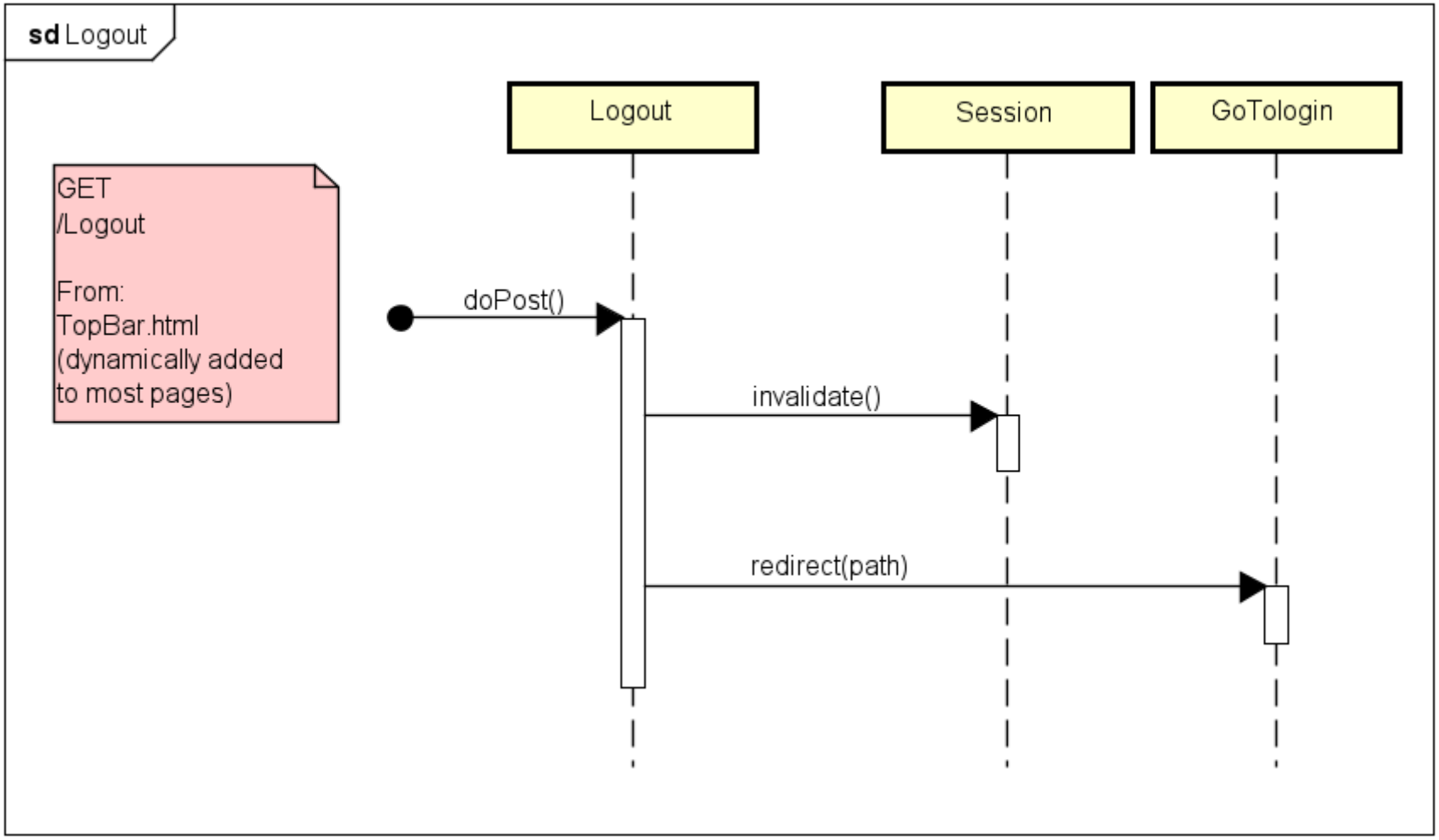

sd CreateAccount
POST
/CreateAccount
name
balance
From:
Home.html
CreateAccount
GoToHome
doPost()
<<create>>
accountDAO(connection)
AccountDAO
Omessi dal diagrammma i
controlli su session.user non nullo e
errori di query SQL per semplicità
alt
[any error]
POST parameters error
balance < 0
name already exists
sendRedirect(path + "?accErrorid=...")
[otherwise]
createAccount(username, password)
sendRedirect(path)


sd GoToAccountState
GoToAccountState
GoToHome
ctx
Template Engine
GET
/GoToAccountState
accountID
From:
Home.html
doGet()
<<create>>
accountDAO(connection)
AccountDAO
Omessi dal diagrammma i
controlli su user non nullo e
errori di query SQL per semplicità
account = getAccountByID(accountID)
alt
[any error]
accountID not number
account == null
account.userid != session.user.userid
sendRedirect(path + "?errorID=...")
[otherwise]
<<create>>
MovementDAO(connection)
MovementDAO
inMovements = getIncomingMovementsByAccount(accountID)
outMovements = getOutgoingMovementsByAccount(accountID)
setVariable("inMovements", inMovements)
setVariable("outMovements", outMovements)
process(ctx, "AccountState.html",...)


sd RequestMovement
POST
/RequestMovement
inuserid
inaccountid
ammount
motive
outaccountid (hidden)
From:
AccountStatus.html
RequestMovement
GoToHome
GoToMovementFailure
Session
GoToMovementSuccess
doPost()
<<create>>
accountDAO(connection)
AccountDAO
inAccount = getAccountByID(accountID)
inAccount || null
Omessi dal diagrammma i
controlli su user non nullo e
errori di query SQL per semplicità
outAccount = getAccountByID(thisAccountID)
outAccount || null
alt
[outAccountID == null || non numeric]
sendRedirect(path + "?errorid=...")
Questo caso non redireziona a MovementeFailure poichè senza outAccountID
non si può creare correttamente il link di "back" al giusto AccountState.
Un alternativa potrebbe essere rendere il link di "back" un link alla home e
redirezionare dunque a MovementeFailure.
[any other error]
altri parametri POST nulli
amount <= 0
outAccountID == inAccountID
userID non esistente
account ID non esistenti
account non appartengono ai rispettivi user
outAccount non ha abbastanza soldi
sendRedirect(path + "?errorid=...")
[otherwise]
<<create>>
transferDAO(connection)
TransferDAO
requestMovement(amount, date, inAccountID, outAccountID)
<<create>>
Movement()
Movement
Movement viene salvato nella sessione.
In ogni momento ogni user può avere massimo
un movement in sessione relativo all'ultimo
movimento effetuato con successo.
Dopo che viene mostrata la pagina di successo
(si veda sequence diagram GoToMovementSuccess)
l'attributo è rimosso dalla sessione.
Questo ci assicura che vengano salvati pochi
dati in sessione, che vengano rimossi quasi subito
nella maggior parte dei casi (infatti si è immediatamente
redirezionati a GoToMovementSuccess) e che la
pagina di conferma sia visualizzabile solo una volta
e solo quando esiste un trasferimento di cui va mostrato
il successo.
set.....(dati usati per requestMovement)
setAttribute("movementMade", movement)
sendRedirect(path)


sd GoToMovementFailure
GoToMovementFailure
GoToHome
ctx
Template Engine
GET
/GoToMovementFailure
accountID
errorid
From:
Home.html
doGet()
<<create>>
accountDAO(connection)
AccountDAO
Omessi dal diagrammma i
controlli su user non nullo e
errori di query SQL per semplicità
account = getAccountByID(accountID)
alt
[any error]
accountID not number
account == null
account.userid != session.user.userid
errorid == null
errorid non valido
sendRedirect(path + "?errorid=...")
[otherwise]
The error to set is derived by the erroid thanks to a utility function
[errorid is valid] setVariable("error", error)
setVariable("backPath", path + "/AccountState?accountid=" + accountid)
process(ctx, "AccountState.html",...)


sd GoToMovementSuccess
GoToMovementSuccess
Session
GoToHome
ctx
Template Engine
GET
/GoToMovementSuccess
From:
redirect
goGet()
movement = getAttribute("movementMade")
movement || bull
Omessi dal diagrammma i
controlli su user non nullo e
errori di query SQL per semplicità
alt
[movement == null]
sendRedirect(path + "errorid?=...")
[otherwise]
removeAttribute("movementMade")
<<create>>
accountDAO()
AccountDAO
outAccount = getAccountByID(outAccountID)
outAccount
inAccount = getAccountByID(inAccountID)
inAccount
setVariable("movement", movement)
setVariable("outAccount", outAccount)
setVariable("inAccount", inAccount)
process(ctx, "MovementSuccess.html", ...)


sd Logout
Logout
Session
GoTologin
GET
/Logout
From:
TopBar.html
(dynamically added
to most pages)
doPost()
invalidate()
redirect(path)