

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]

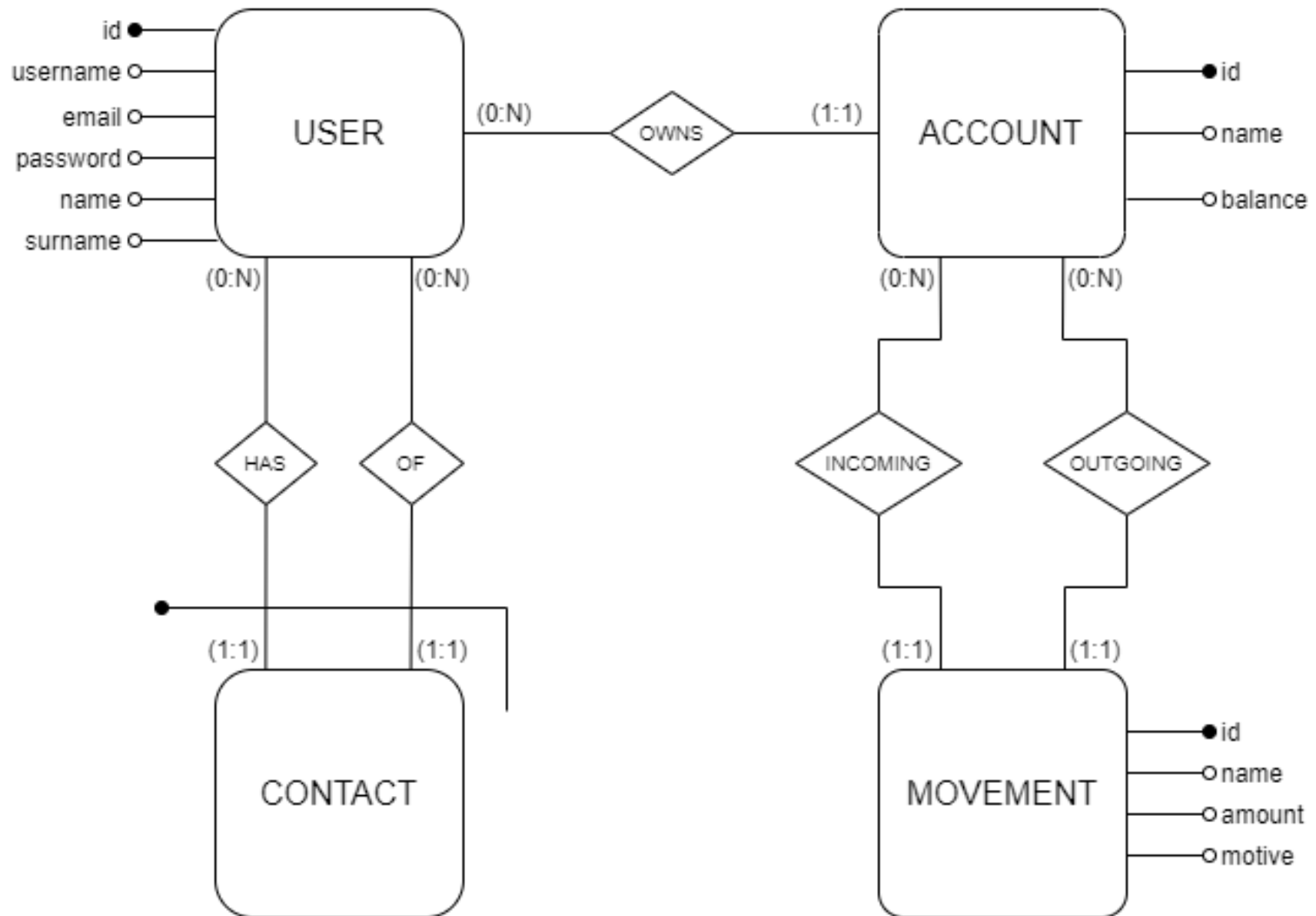
Al fine di progettare una singola base di dati per entrambe le applicazioni richieste, di seguito si analizzano anche i dati richiesti dalla versione con JavaScript.

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**.

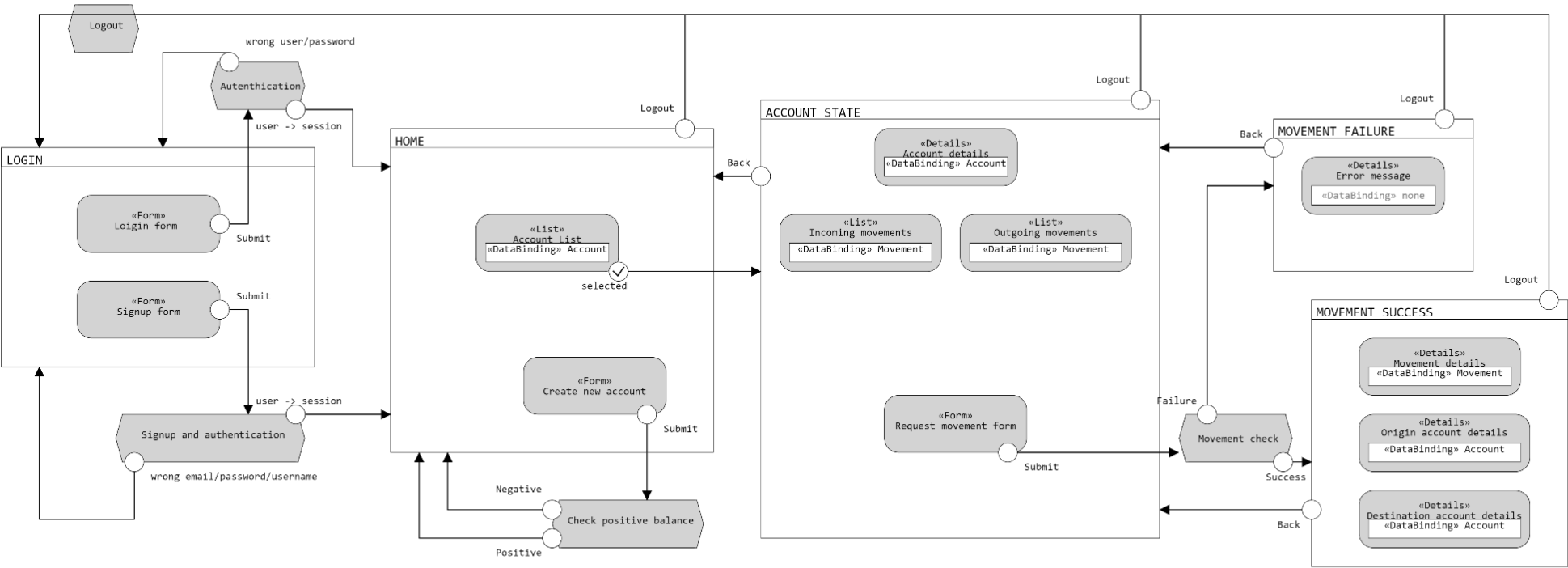
Nel caso in cui entrambe le condizioni siano soddisfatte, l'applicazione **deduce l'importo dal conto di origine, aggiunge l'importo al conto di destinazione** e mostra una **pagina CONFERMA TRASFERIMENTO** che presenta i **dati dell'importo trasferito** e i **dati del conto di origine e di destinazione 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, oltre 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



Components

Model object (Beans)

- User
- Account
- Movement

(N.B.: La funzionalità per ottenere i contatti non serve nell'applicazione pure HTML e quindi è omessa dai beans e dalle funzioni dei DAO)

Data Access Objects (Classes)

- UserDao
 - checkCredentials(username, password)
 - 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)

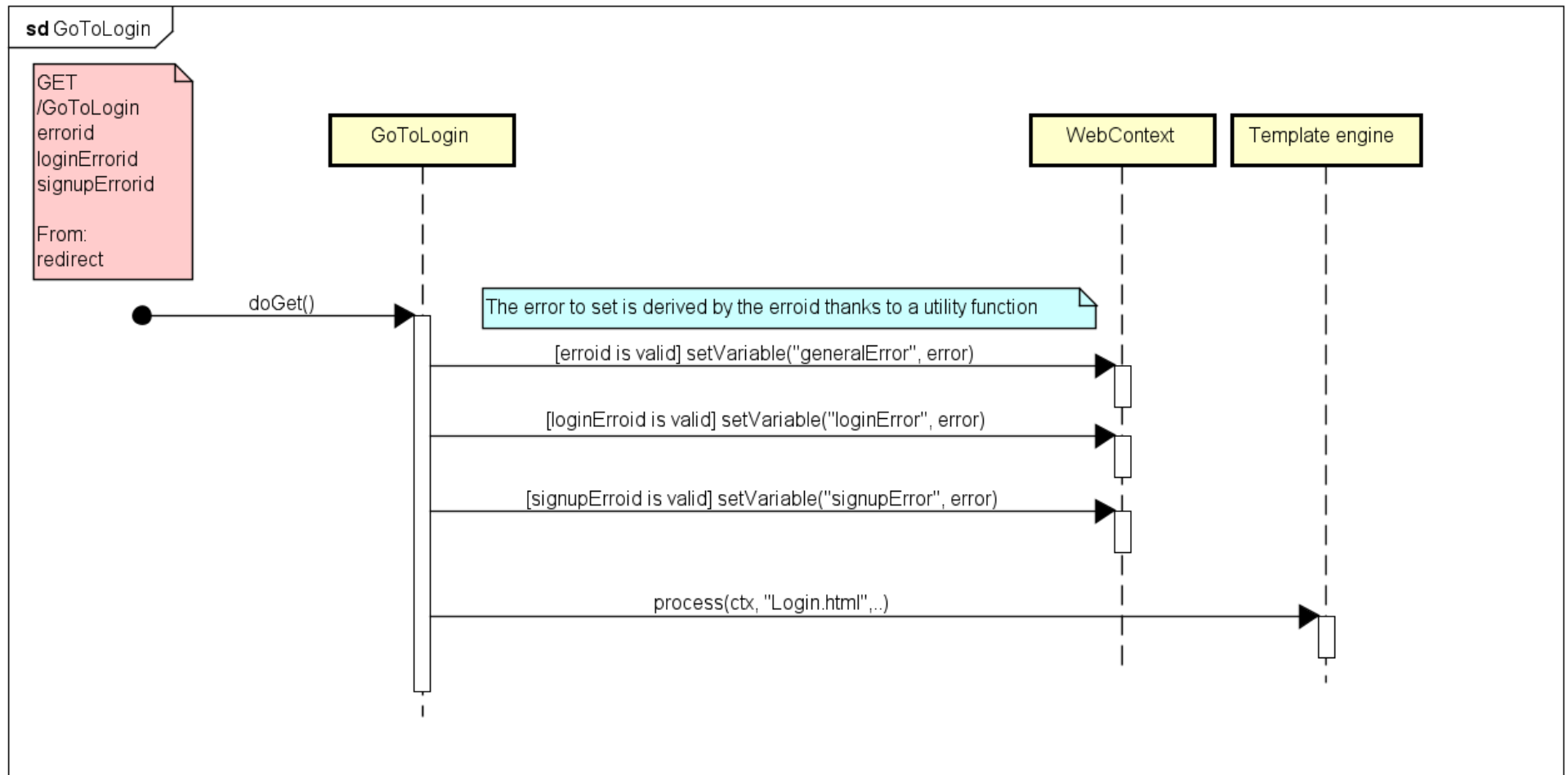
Controllers (Servlets)

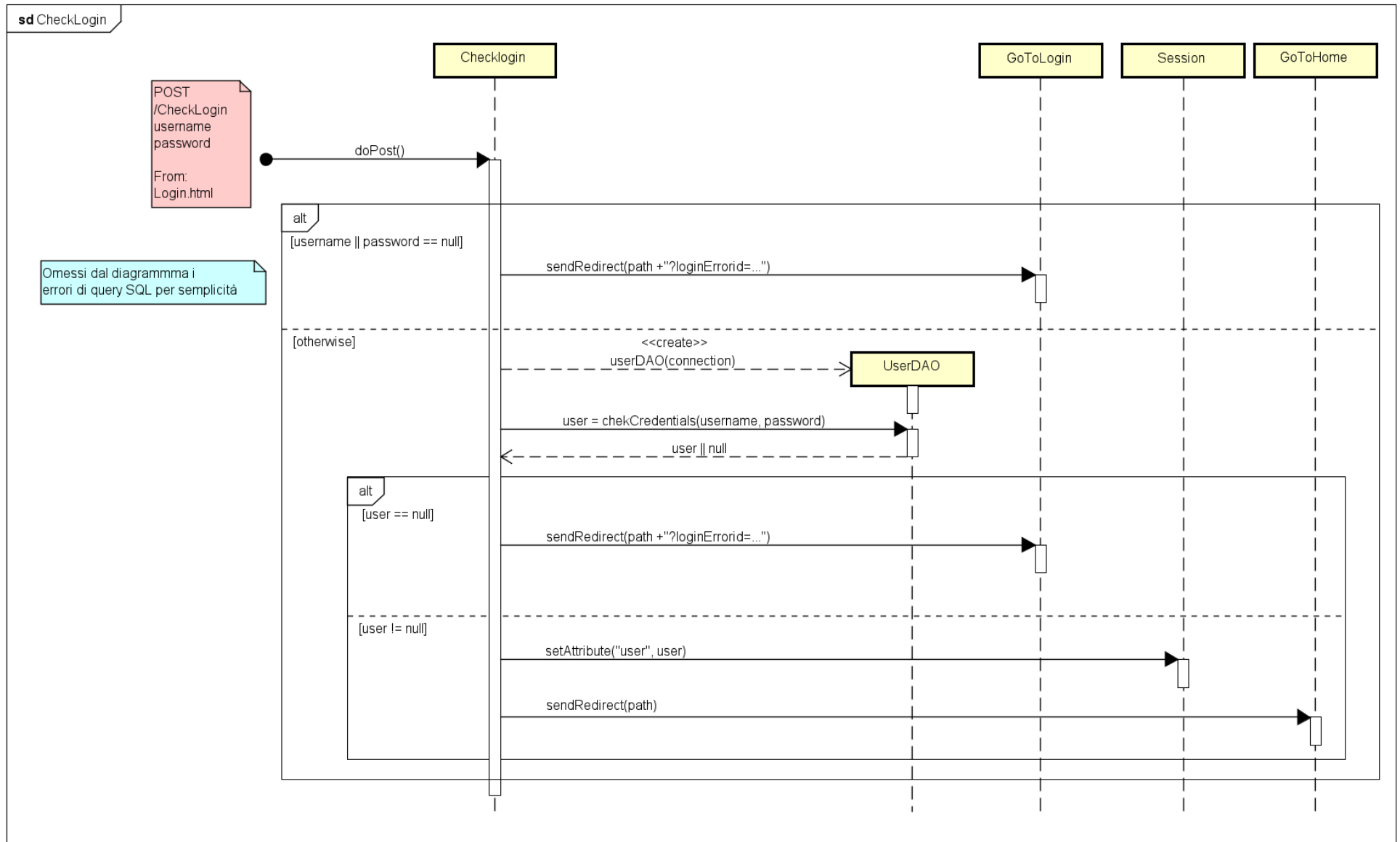
- GoToLogin
- CheckLogin
- CheckSignup
- GoToHome
- CreateAccount
- GoToAccountState
- RequestMovement
- GoToMovementFailure
- GoToMovementSuccess
- Logout

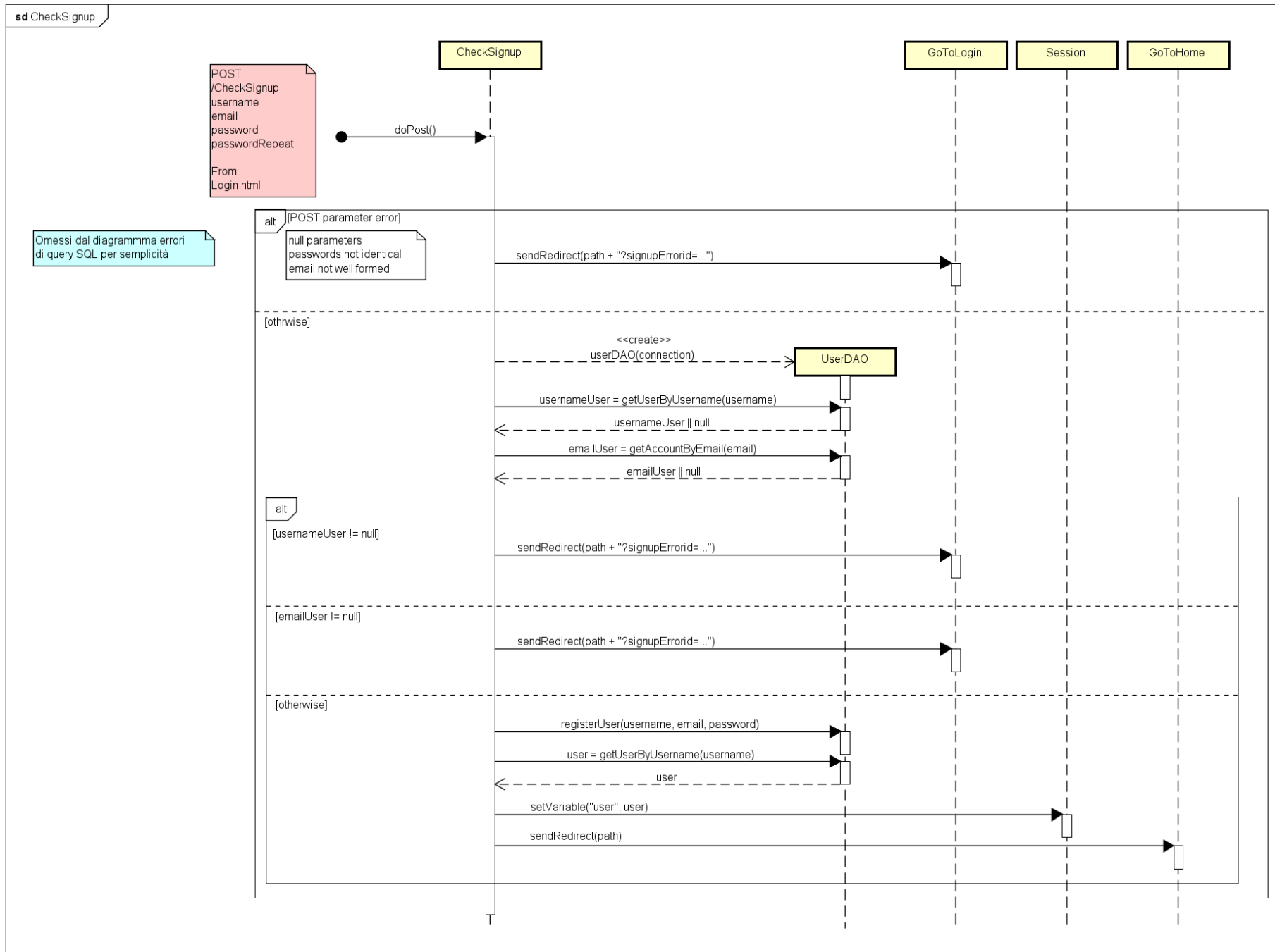
Views (Templates)

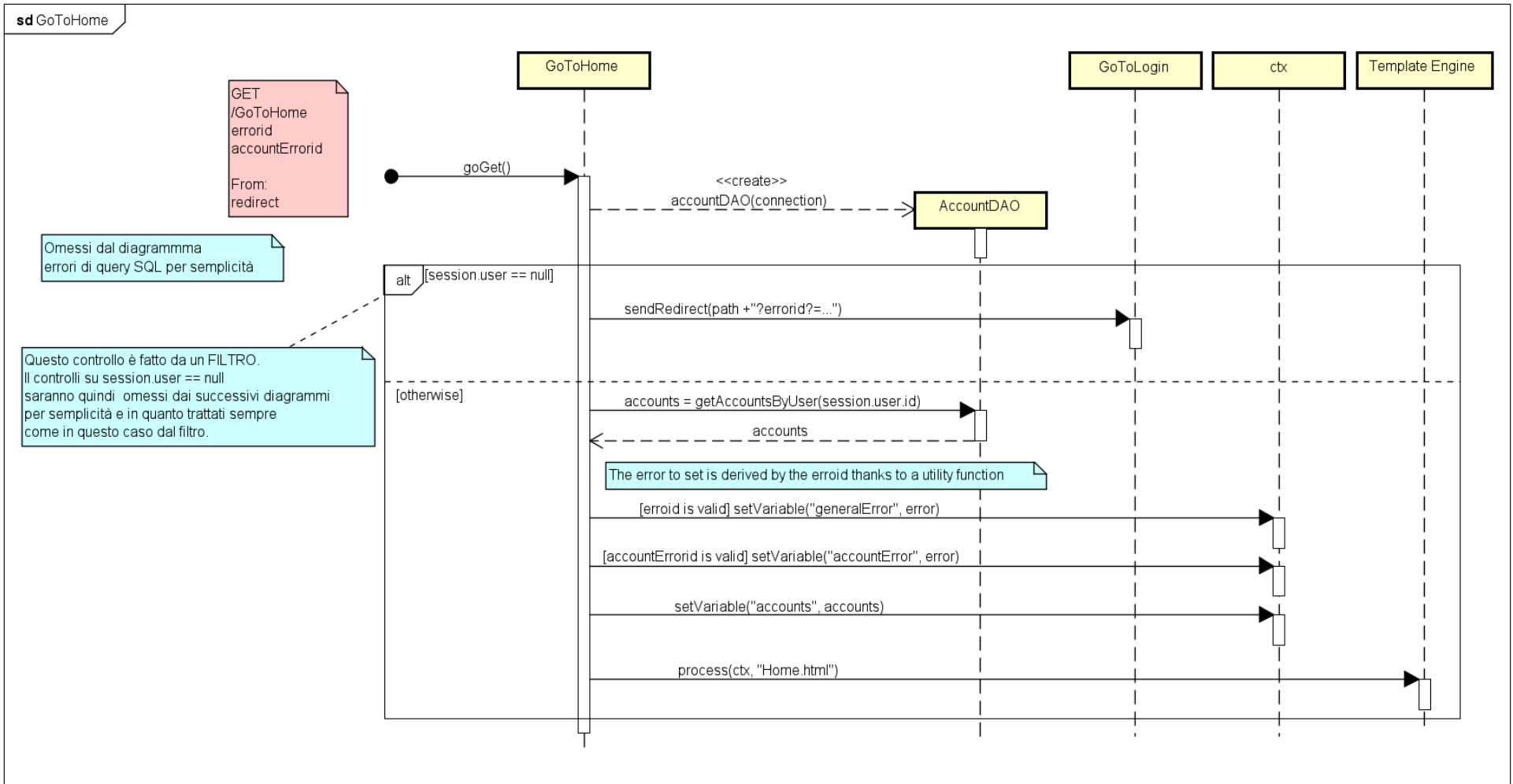
- Login
- Home
- AccountState
- MovementFailure
- MovementSuccess
- TopBar (aggiunta dinamicamente alle altre pagine)

Sequence diagrams







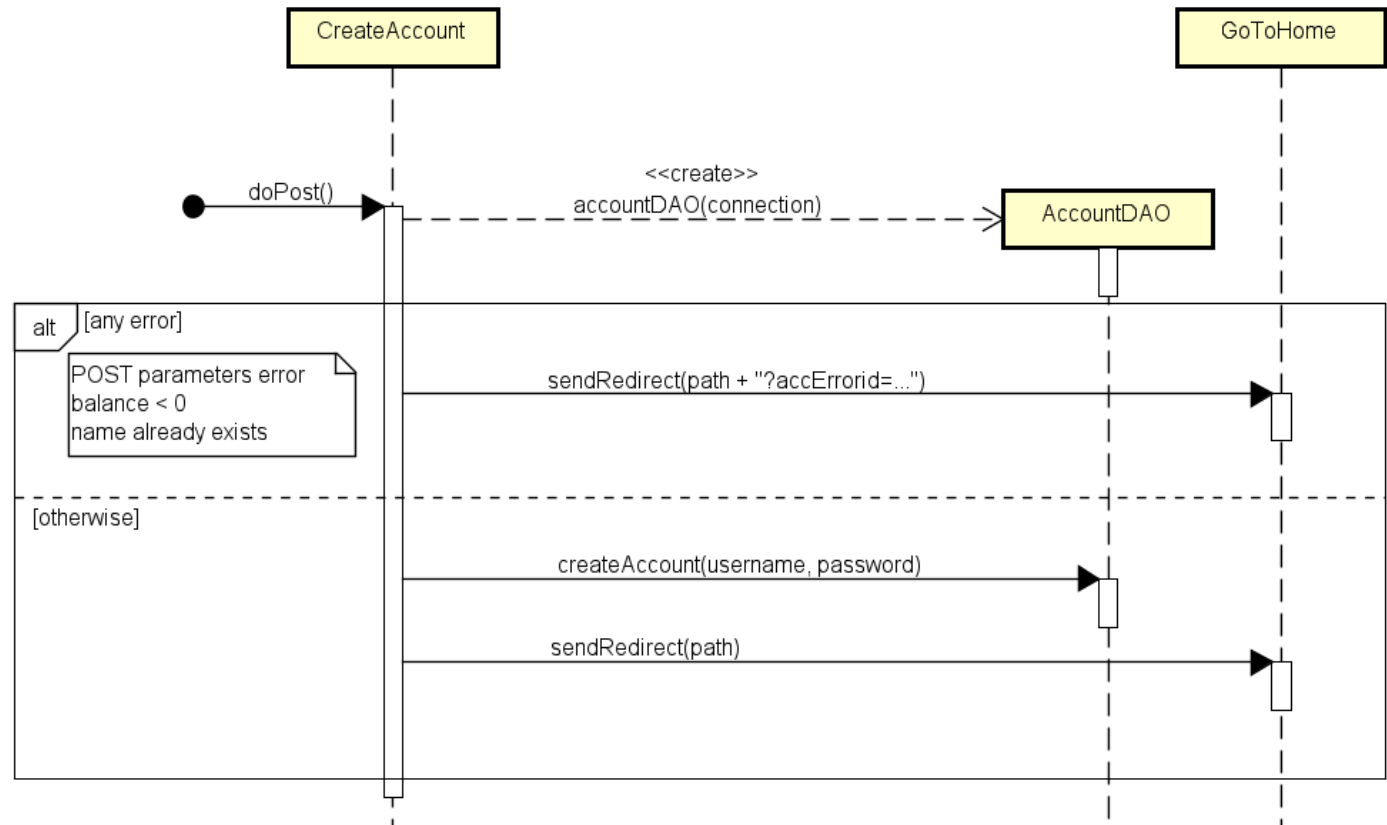


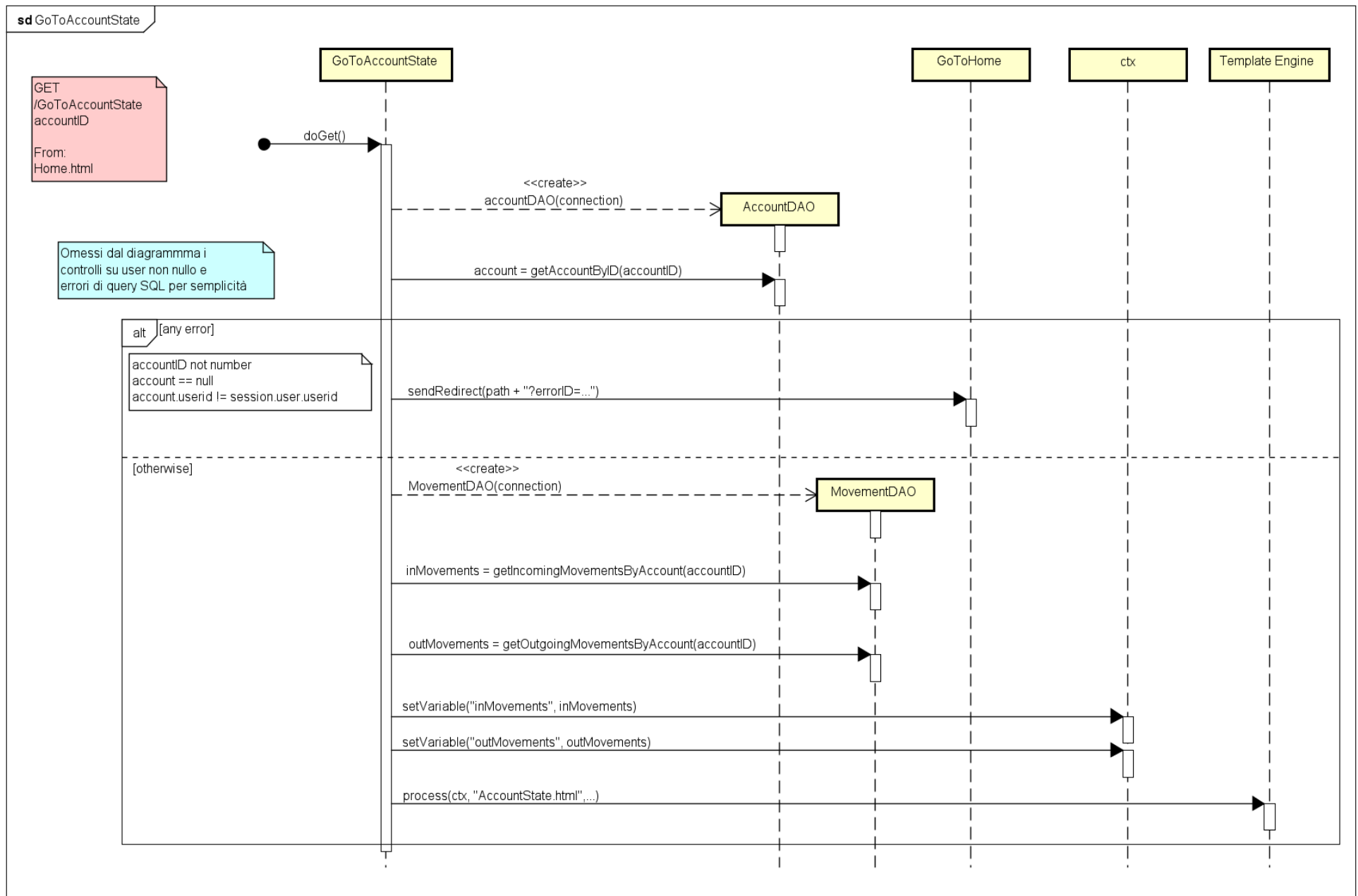
sd CreateAccount

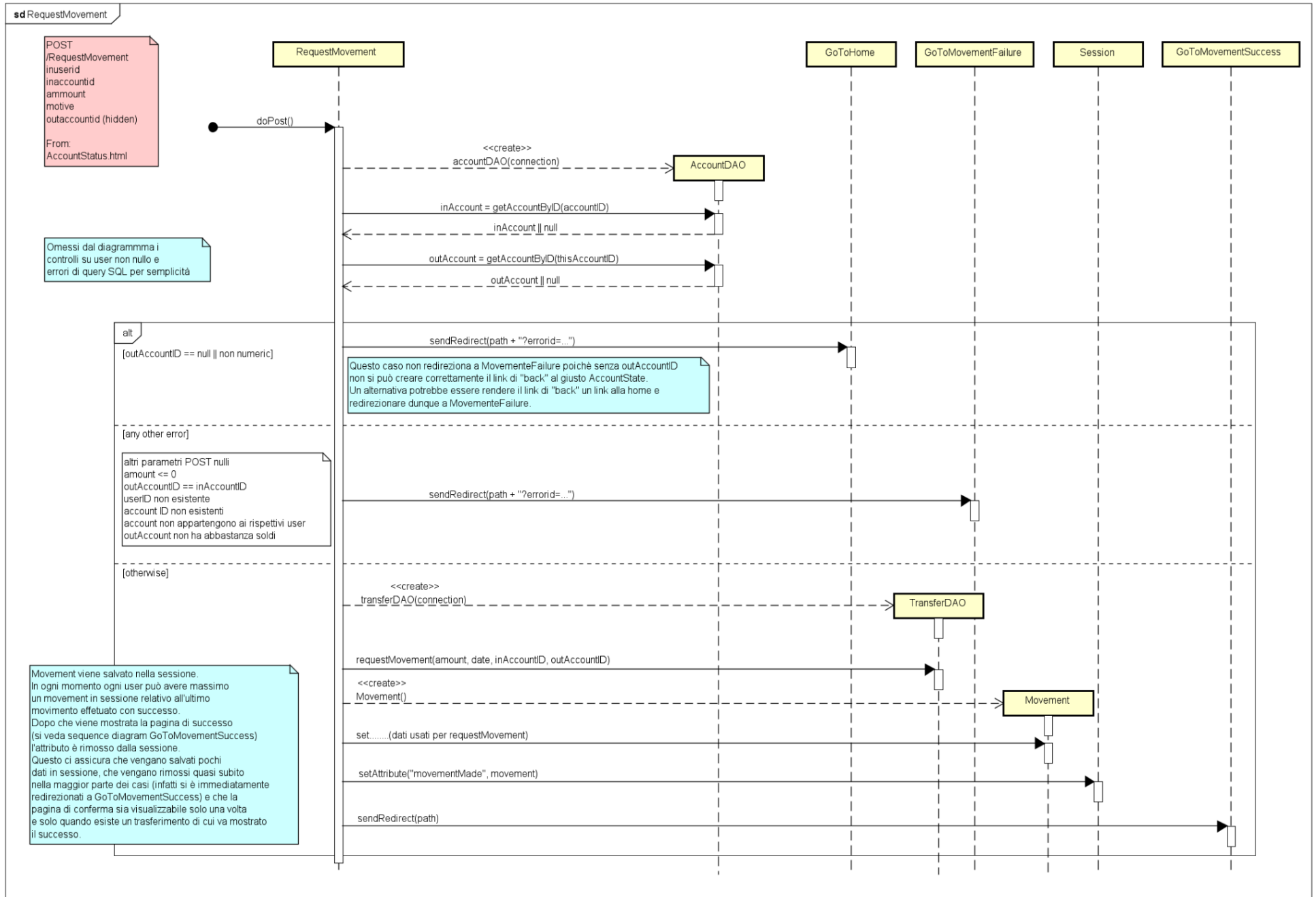
POST
/CreateAccount
name
balance

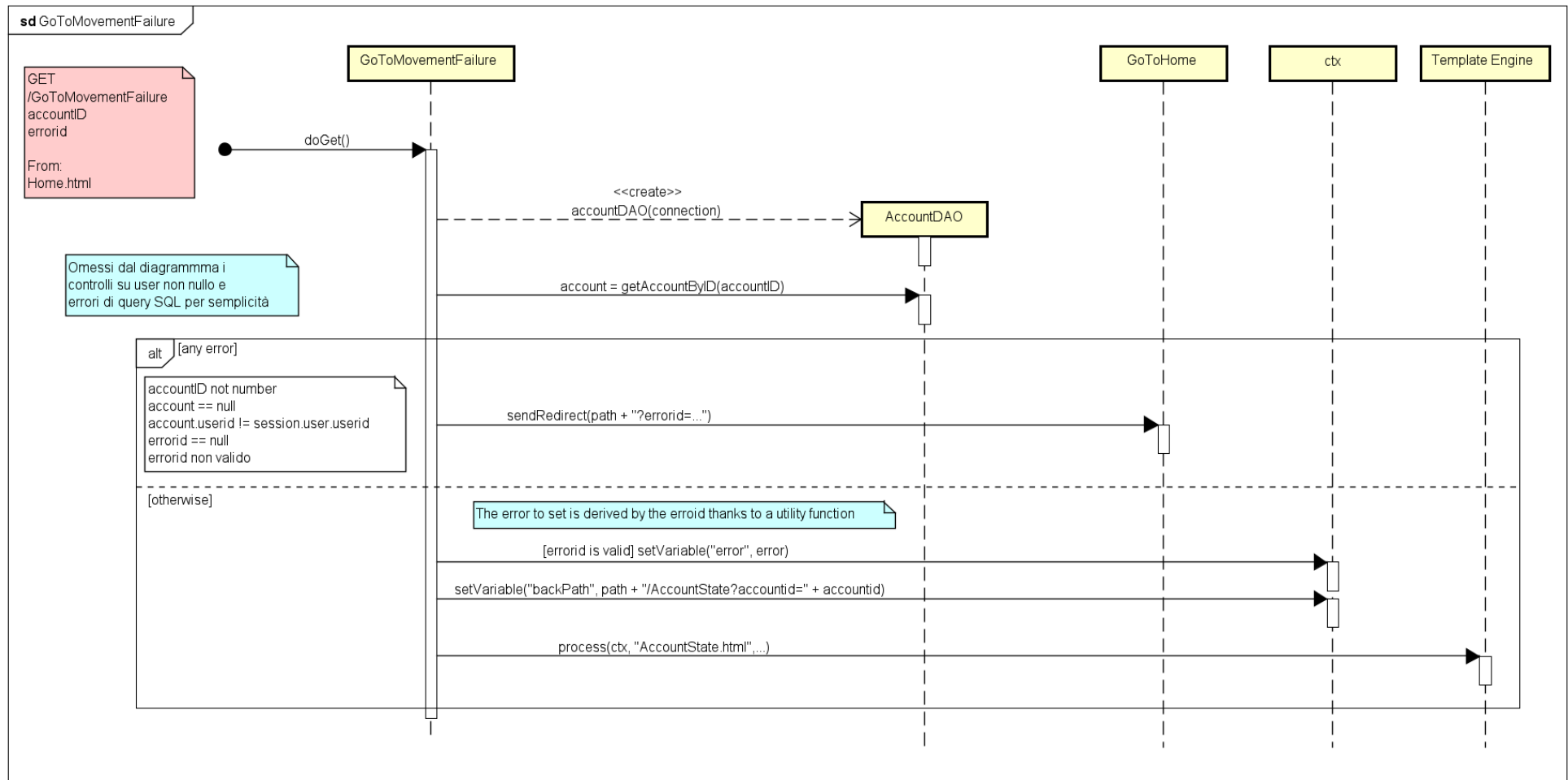
From:
Home.html

Omessi dal diagramma i
controlli su session.user non nullo e
errori di query SQL per semplicità





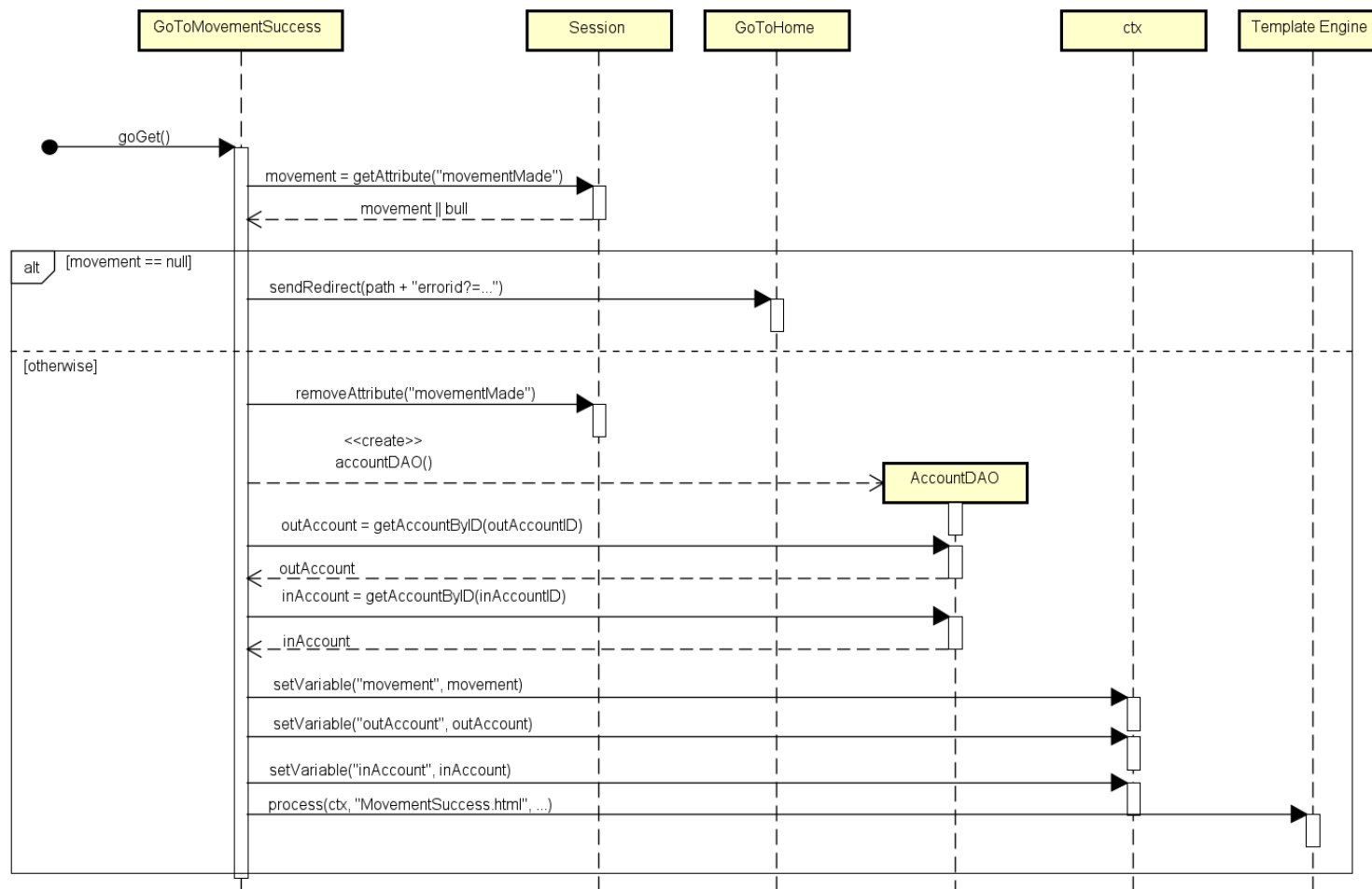




sd GoToMovementSuccess

GET
/GoToMovementSuccess
From:
redirect

Omessi dal diagramma i
controlli su user non nullo e
errori di query SQL per semplicità



sd Logout

GET
/Logout

From:
TopBar.html
(dynamically added
to most pages)

