Traccia 4 – pure HTML

Gruppo 66

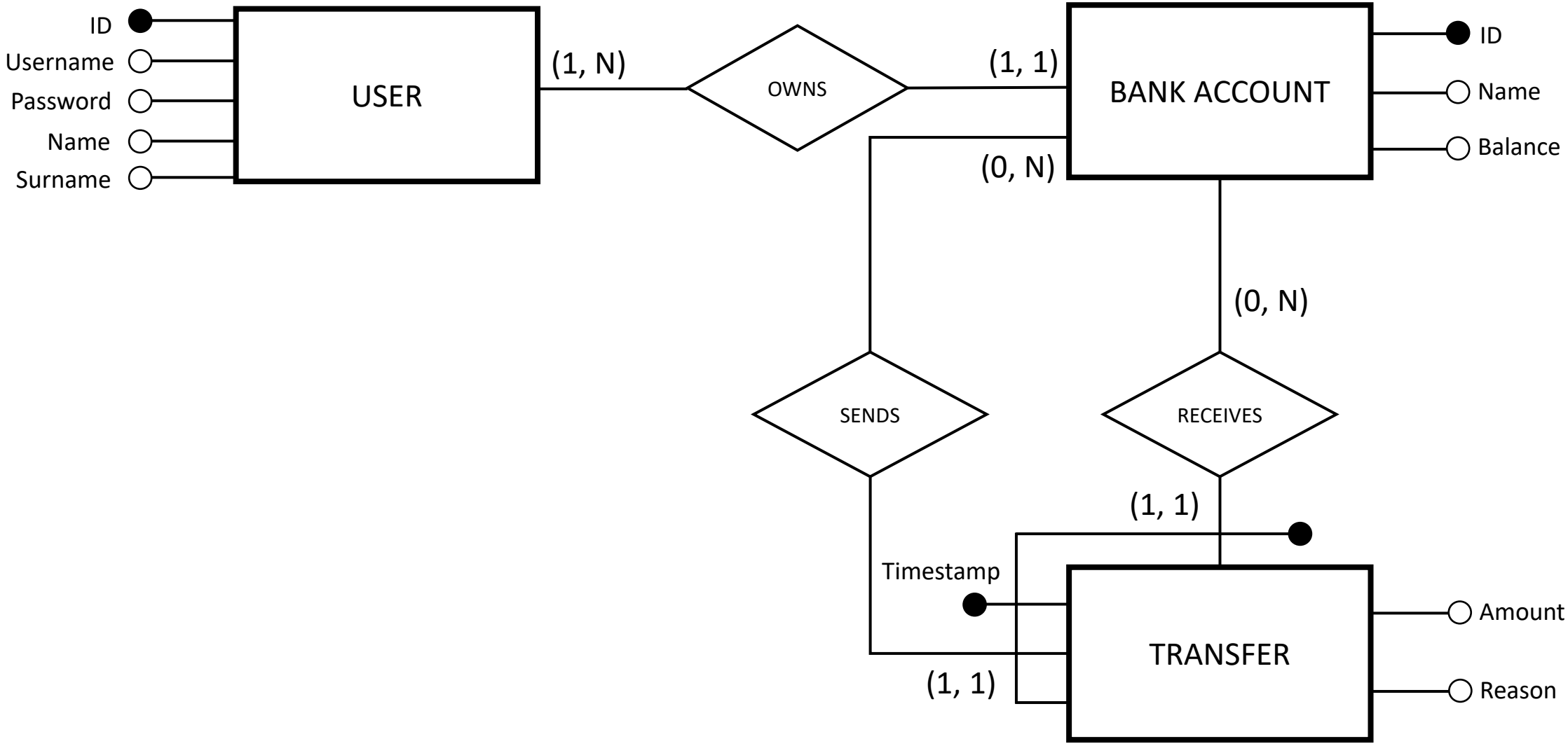
Federico Toschi – Sara Resta

Indice

Analisi ER

- 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 desti- nazione 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, Relations

Diagramma ER



Database schema (1/3)

```
CREATE TABLE `user` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `username` varchar(50) NOT NULL,
  `password` varchar(50) NOT NULL,
  `name` varchar(20) NOT NULL,
  `surname` varchar(30) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE `username` (`username`)
) ENGINE=InnoDB;
```

Database schema (2/3)

```
CREATE TABLE `bank_account` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `userID` int NOT NULL,
  `name` varchar(40) NOT NULL,
  `balance` decimal(10,2) NOT NULL,
  PRIMARY KEY (`ID`),
  CONSTRAINT `UID` FOREIGN KEY (`userID`)
  REFERENCES `user` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Database schema (3/3)

```
CREATE TABLE `transfer` (
  `senderID` int NOT NULL,
  `recipientID` int NOT NULL,
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `reason` varchar(255) NOT NULL,
  `amount` decimal(10,2) NOT NULL,
  PRIMARY KEY (`timestamp`,`senderID`,`recipientID`),
  CONSTRAINT `DAccount` FOREIGN KEY (`recipientID`)
  REFERENCES `bank_account` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `OAccount` FOREIGN KEY (`senderID`)
  REFERENCES `bank_account` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `PositiveAMount` CHECK ((`amount` >= 0))
) ENGINE=InnoDB;
```

Analisi dei requisiti

- 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.
- Pages (Views), View Components, Events, Actions

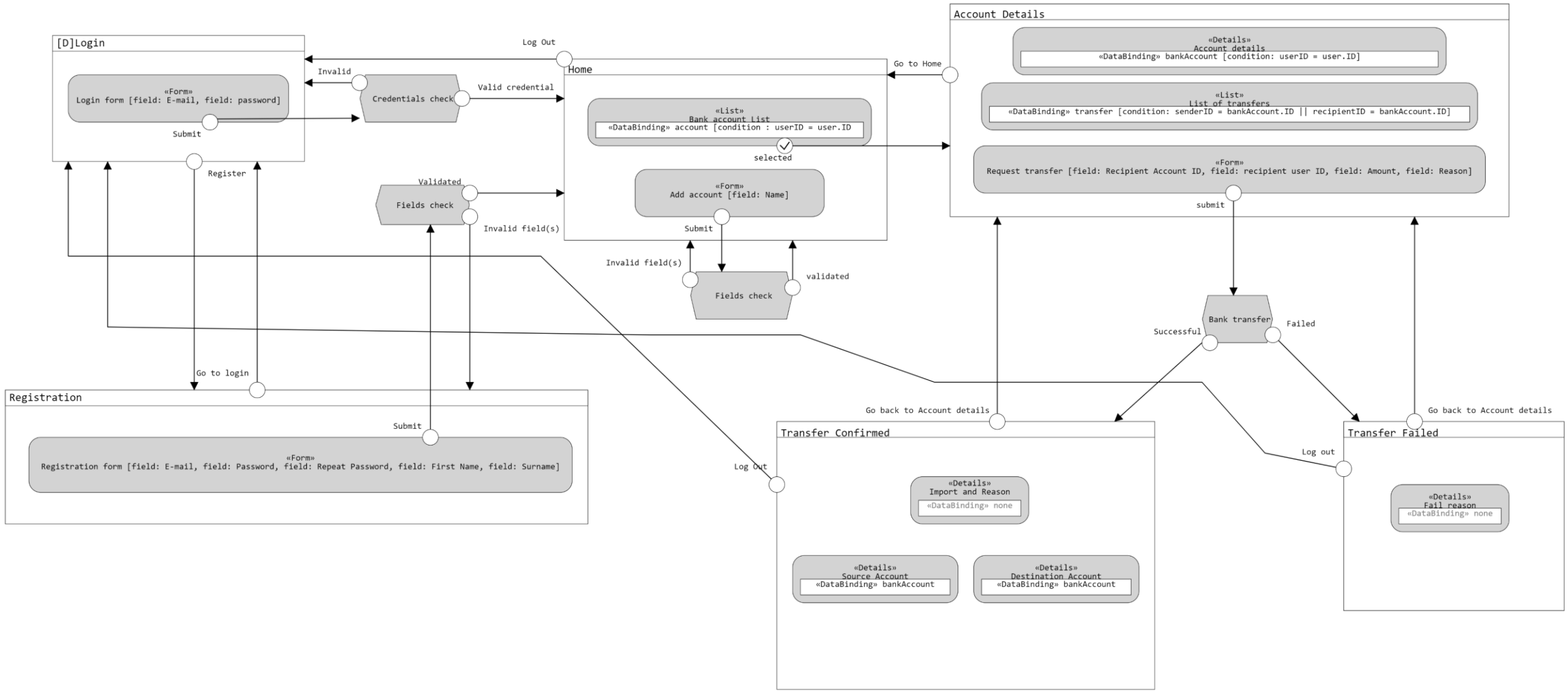
Completamento delle specifiche

- L’utente è identificato tramite ID o username (che coincide con l’indirizzo e-mail)
- Per credenziali di login si intendono email e password.
- Ogni utente deve possedere almeno un conto corrente, al momento della registrazione viene creato un conto corrente di default avente saldo pari a zero
- Nella pagina Home è presente la somma del saldo di tutti i conti dell’utente
- Dalla pagina Home è possibile creare un nuovo conto specificandone il nome. Il nuovo conto creato avrà saldo nullo. Non sono ammessi nomi duplicati.
- Se l’utente ha effettuato il login e tenta di accedere alle pagine di LOGIN o REGISTER verrà reindirizzato automaticamente alla HOME. Viceversa se non ha effettuato il login e tenta di accedere a pagine protette verrà reindirizzato alla pagina di LOGIN
- Se si verificano azioni impreviste, l’utente viene reindirizzato ad una pagina di ERRORE contenente il motivo dell’errore ed un pulsante per tornare alla pagina precedente (Rispetto alla servlet a cui è stata effettuata la richiesta)
- Non è possibile richiedere trasferimenti con conto origine uguale al conto destinazione
- In seguito ad un trasferimento, se andato a buon fine, la servlet GoToTransferConfirmed mostrerà l’ultimo trasferimento effettuato in uscita dall’account dell’utente.
- Ogni stringa inserita dall’utente viene sanificata tramite escaping.
- Viene effettuato unescaping di ogni stringa prima dell’invio all’utente.

Components

- Model objects: beans
  - User
  - BankAccount
  - Transfer
- Data Access Object (DAO)
  - UserDAO
    - findUser(String email, String password): User
    - findUserByID(int ID): User
    - registerUser (String email, String password, String name, String surname): int
    - isEmailTaken(String email): boolean
  - BankAccountDAO
    - findAccountByID(int ID): BankAccount
    - findAllAccountsByUserID (int userID): List<BankAccount>
    - createAccount(int userid, String name, BigDecimal balance)
    - isNameTaken(int userID, String name): boolean
  - TransferDAO
    - getTransferByAccountID (int accountID): List<Transfer>
    - makeTransfer(BigDecimal amount, String reason, int senderid, int recipentid)
    - getLastTransferByUserID(int userID): Transfer
- Filters
  - LoginFilter
  - NotLoggedFilter
- Controllers (servlets) [access rights]
  - Login [not logged user]
  - Register [not logged user]
  - Logout [*]
  - SelectAccount [logged user]
  - MakeTransfer [logged user]
  - GoToLogin [not logged user]
  - GoToTransferConfirmed [logged user]
  - GoToHome [logged user]
- Views (templates)
  - Login
  - Register
  - Home
  - AccountDetails
  - TransferConfirmed
  - TransferFailed
  - Error

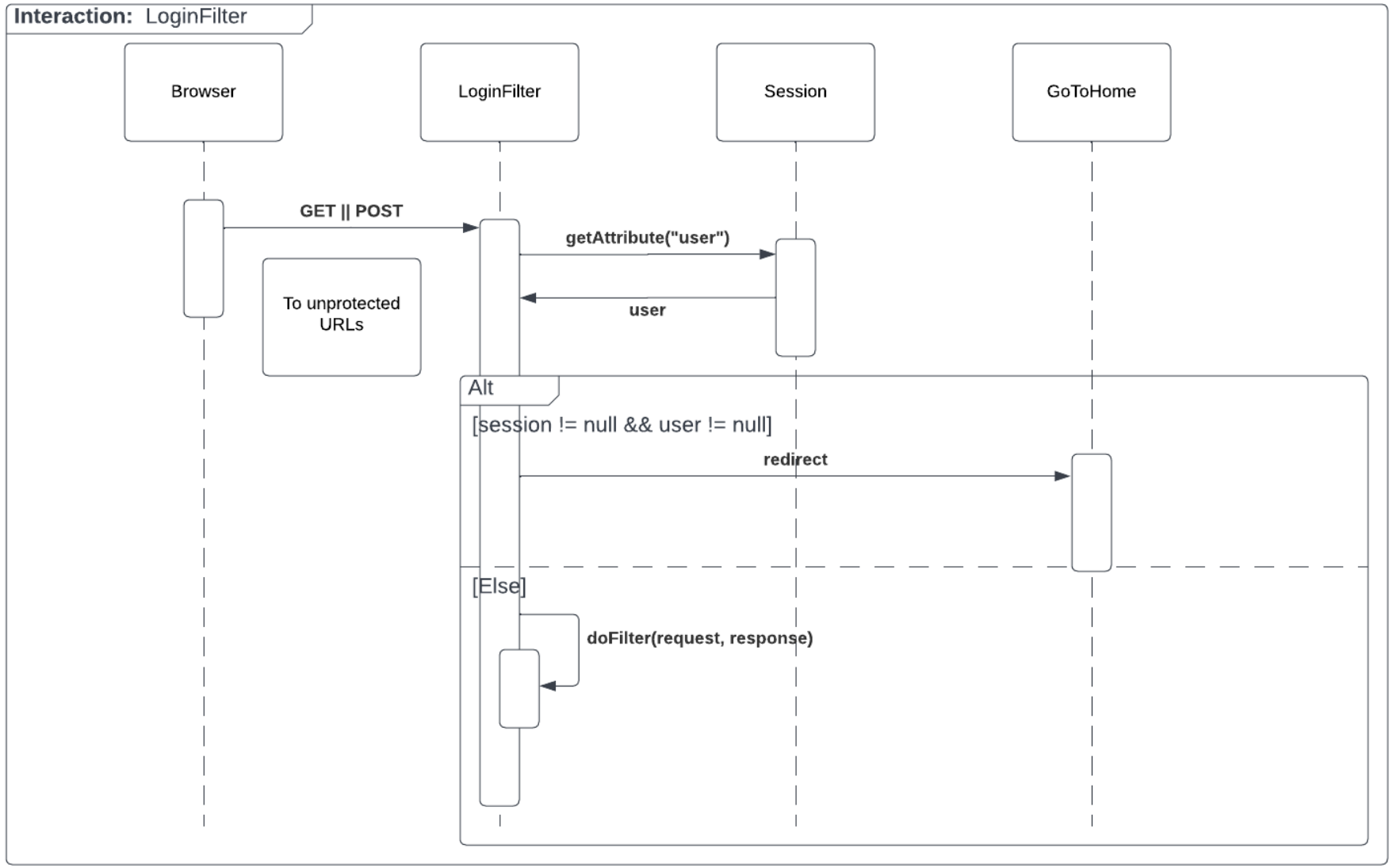
Design Applicativo - IFML

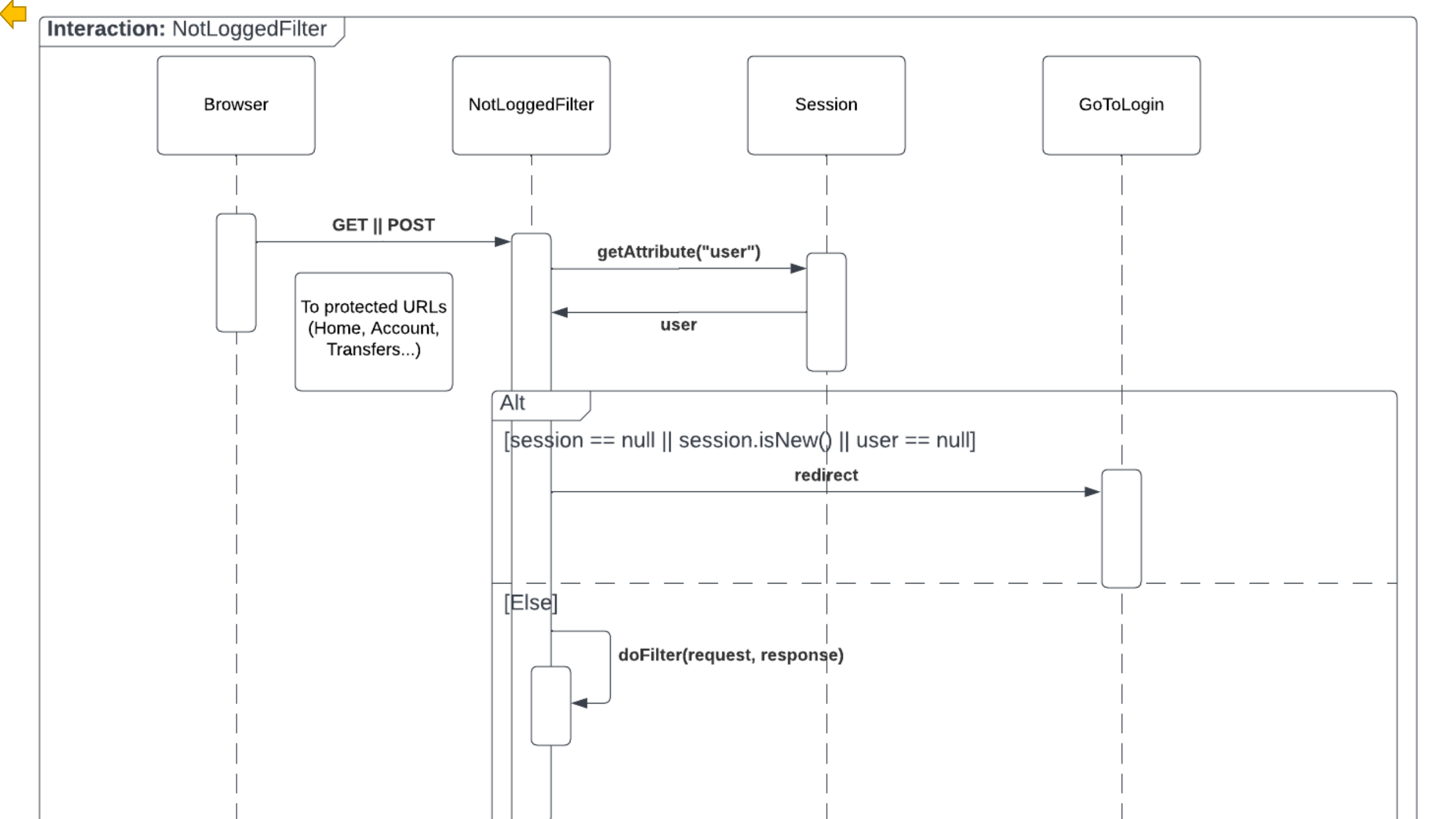


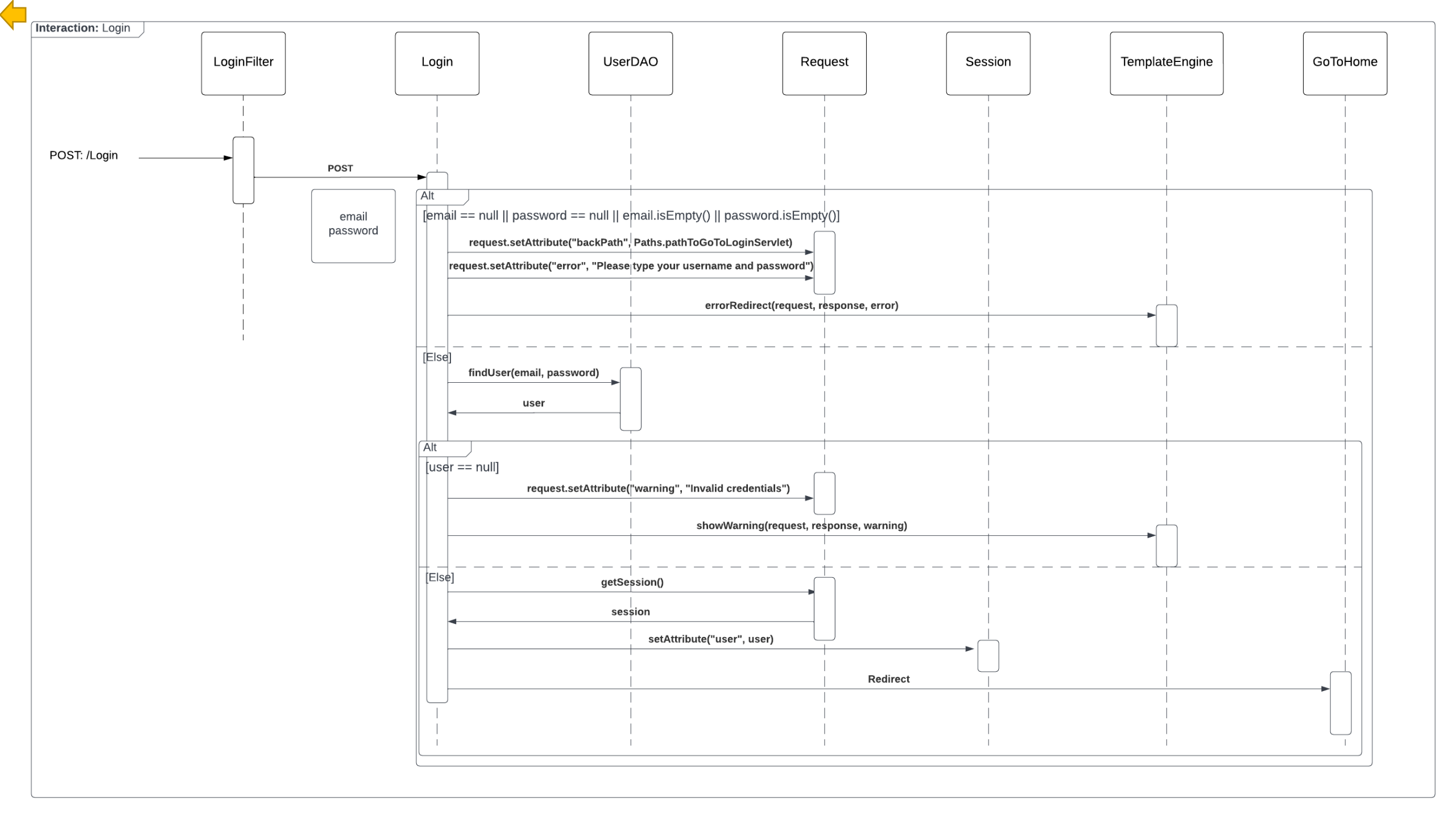
Sequence diagrams

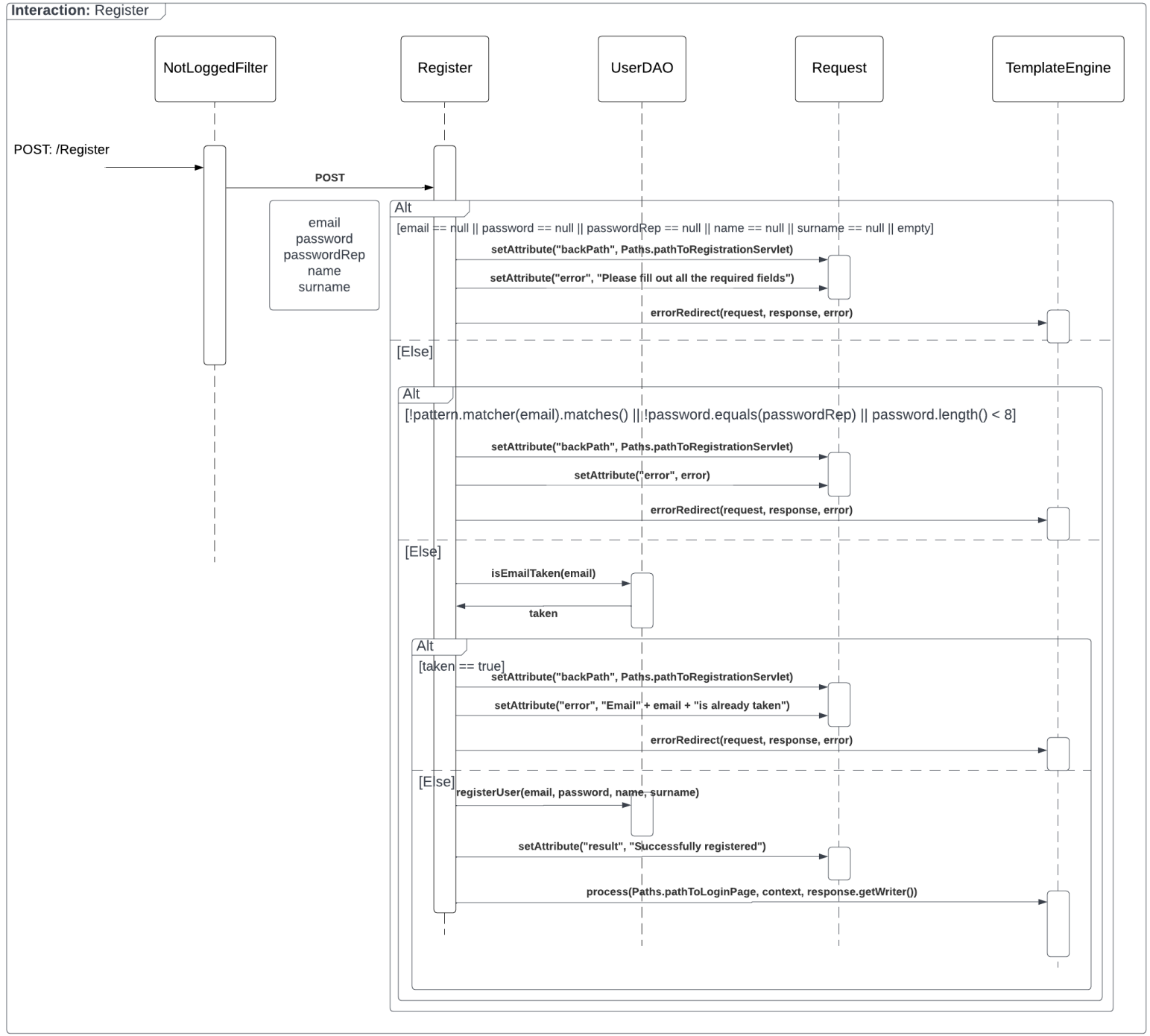
The following sequence diagrams aim to depict the interaction flow underlying the core events of the web application. Although the authors tried their best to pursue and achieve full clarity, some minor details have been left out due to their repetitiveness (e.g. Internal server errors, Database access errors, etc.).

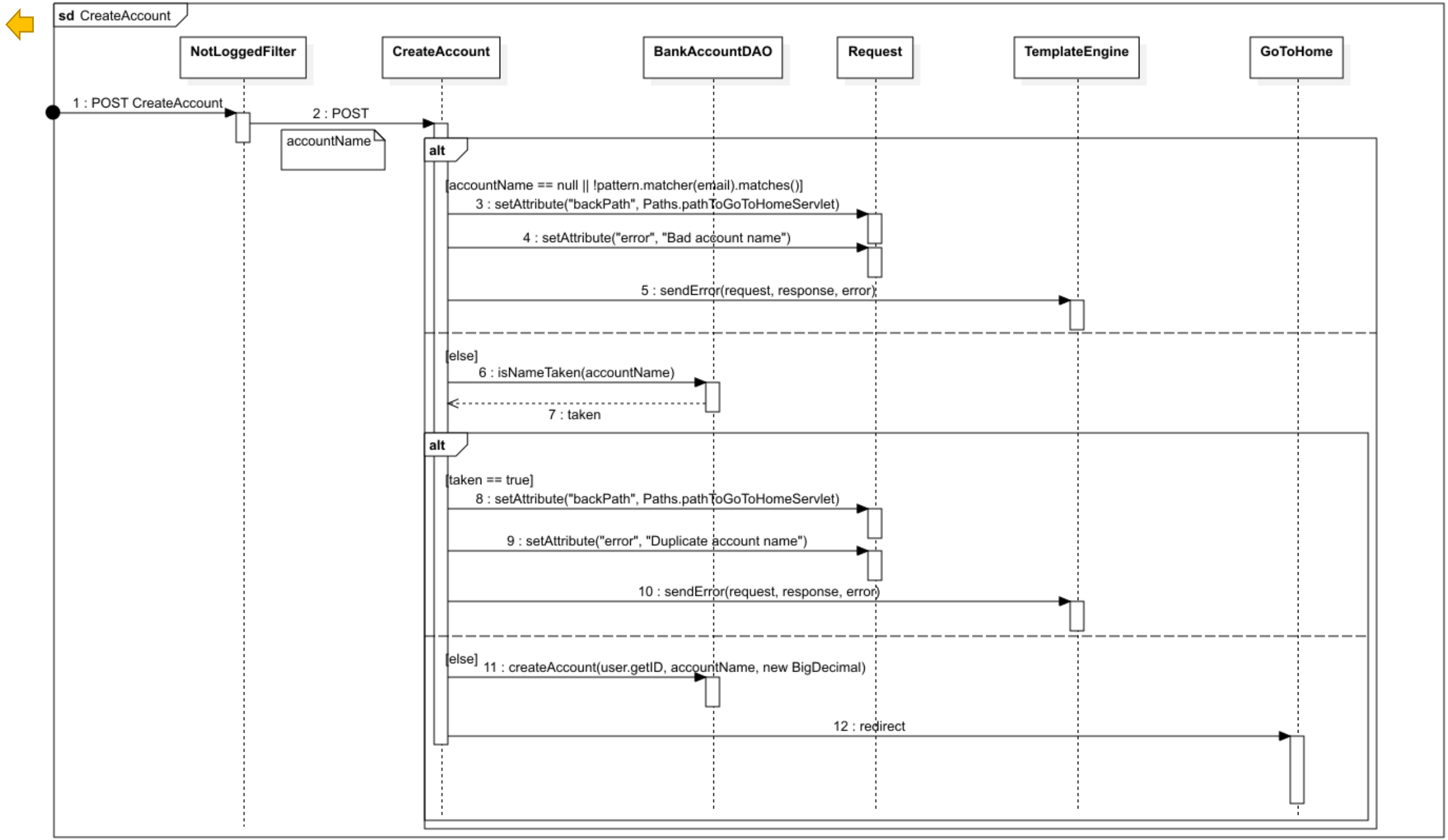
Filter checks will be represented in the first diagrams and omitted in the subsequent ones.

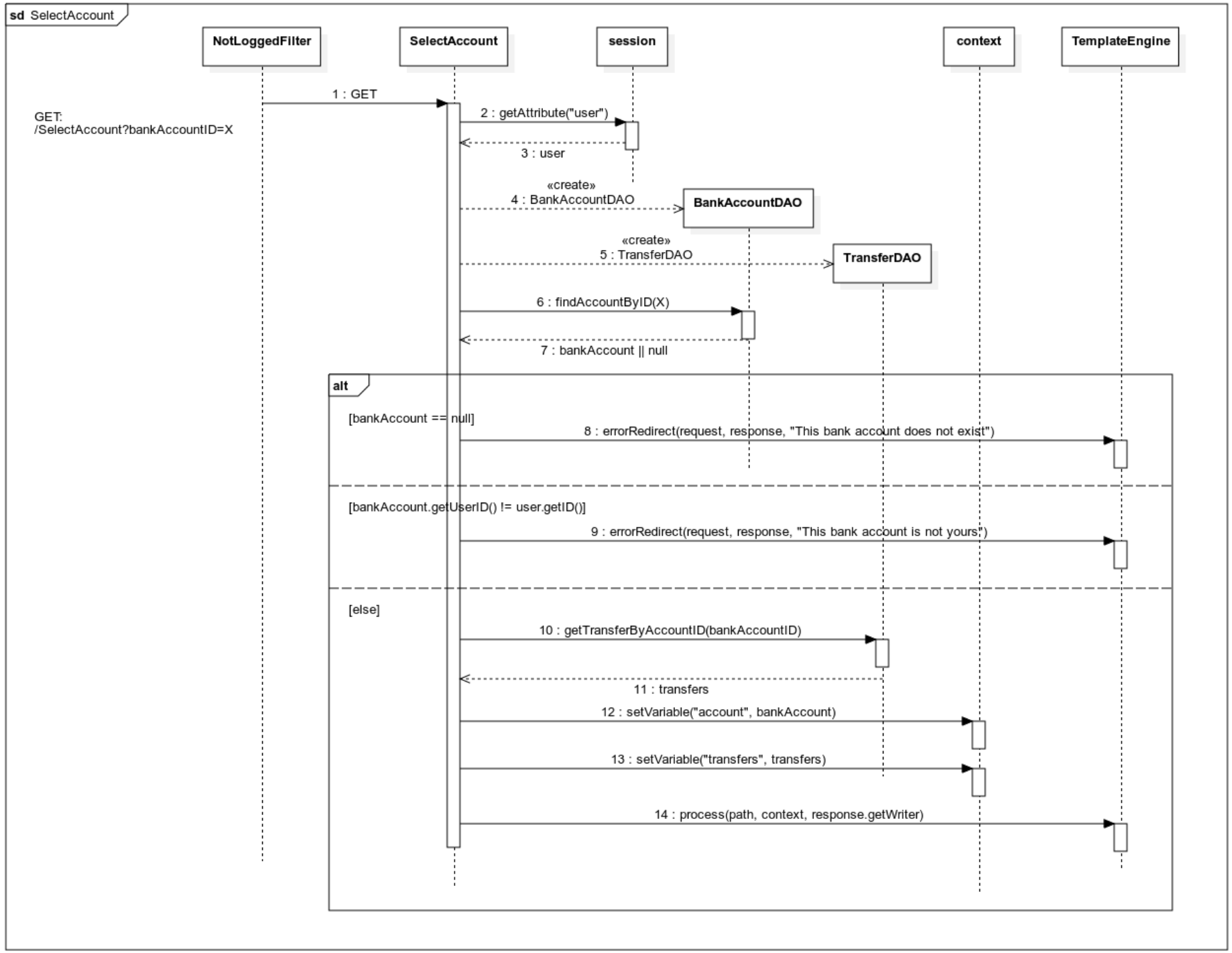

Interaction: LoginFilter
Browser
LoginFilter
Session
GoToHome
GET || POST
getAttribute("user")
To unprotected URLs
user
Alt
[session != null && user != null]
redirect
[Else]
doFilter(request, response)

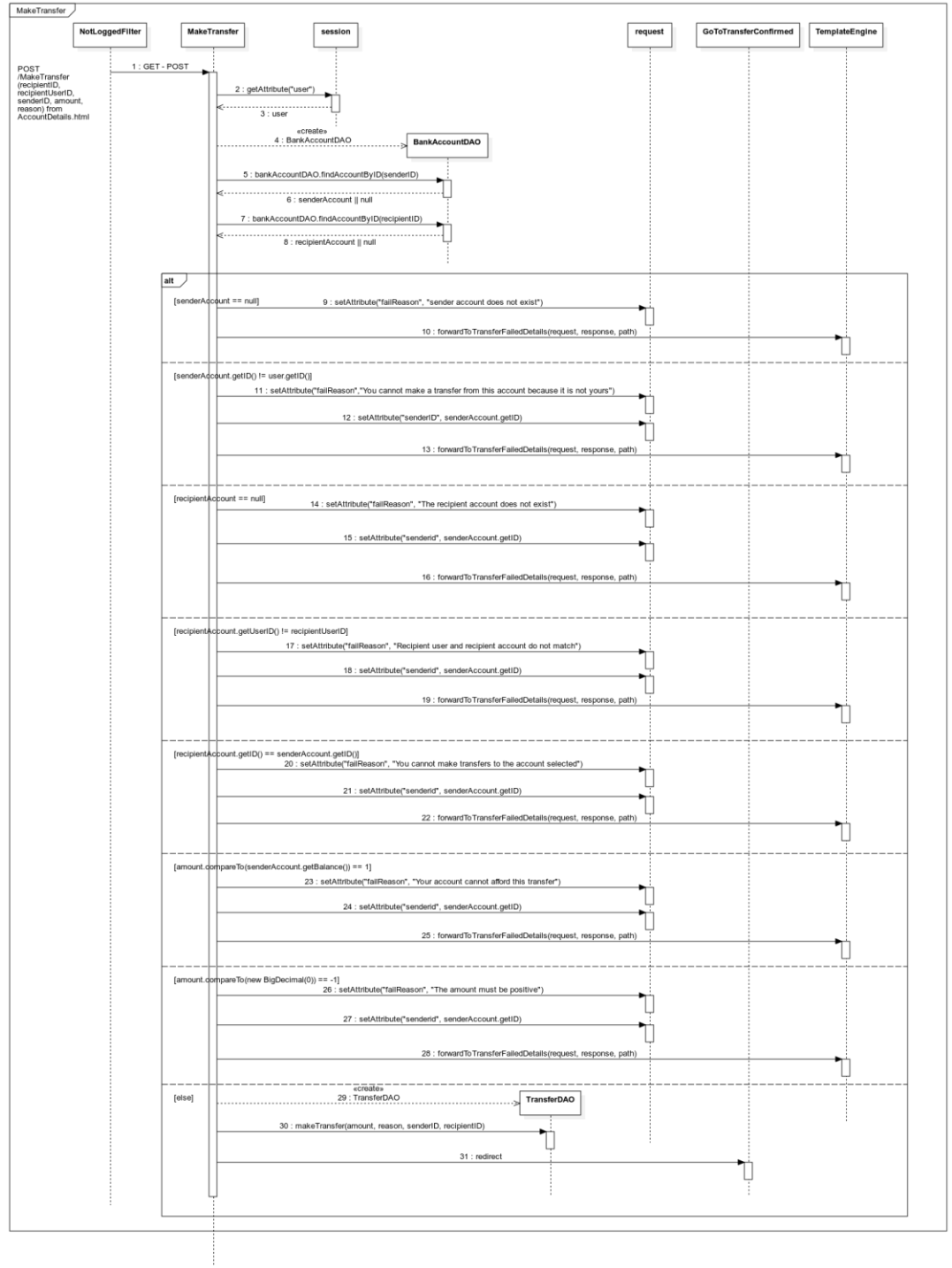

Interaction: NotLoggedFilter
Browser
NotLoggedFilter
Session
GoToLogin
GET || POST
getAttribute("user")
To protected URLs (Home, Account, Transfers...)
user
Alt
[session == null || session.isNew() || user == null]
redirect
[Else]
doFilter(request, response)

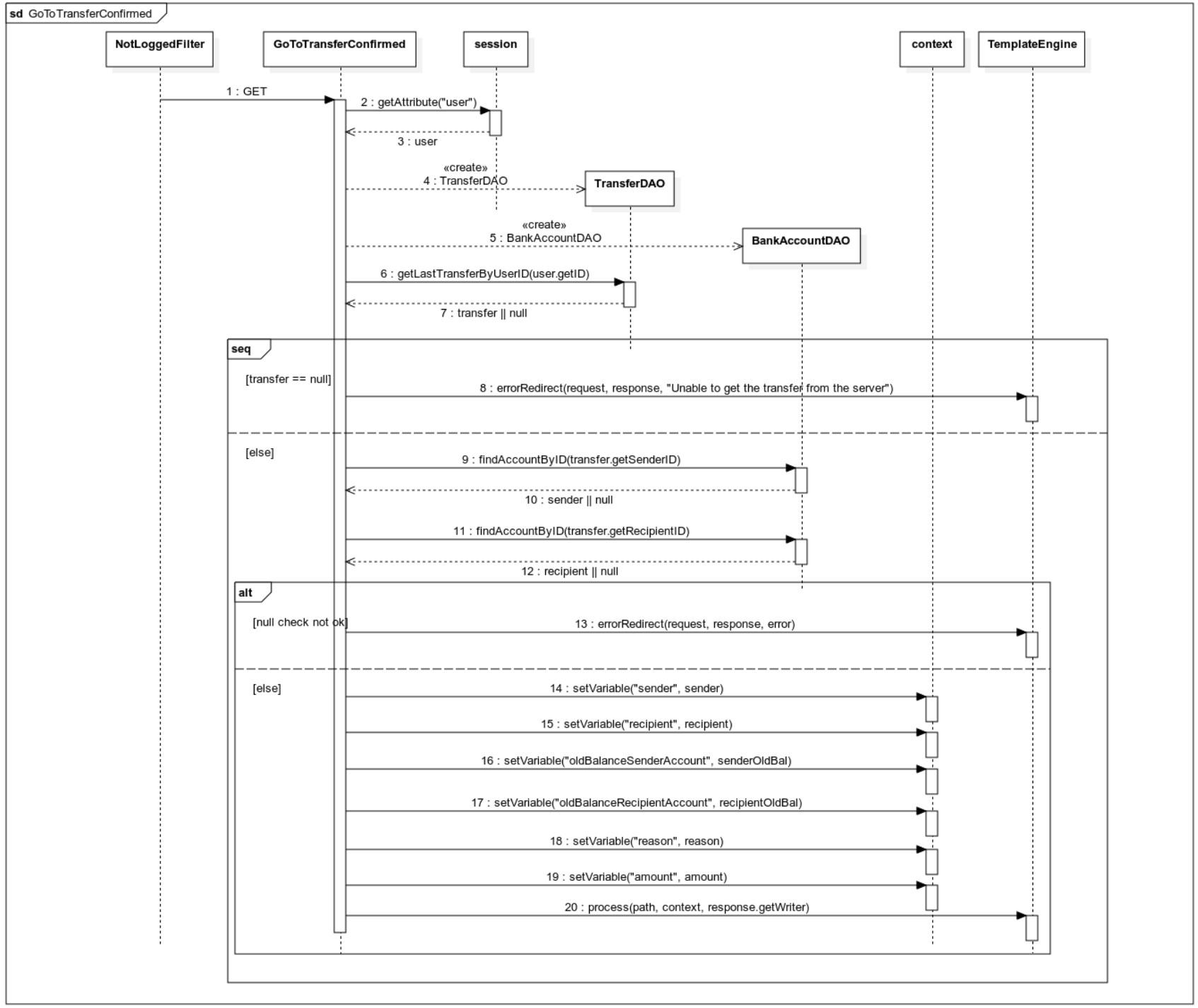

Interaction: Login
LoginFilter
Login
UserDAO
Request
Session
TemplateEngine
GoToHome
POST: /Login
POST
email
password
Alt
[email == null || password == null || email.isEmpty() || password.isEmpty()]
request.setAttribute("backPath", Paths.pathToGoToLoginServlet)
request.setAttribute("error", "Please type your username and password")
errorRedirect(request, response, error)
[Else]
findUser(email, password)
user
Alt
[user == null]
request.setAttribute("warning", "Invalid credentials")
showWarning(request, response, warning)
[Else]
getSession()
session
setAttribute("user", user)
Redirect

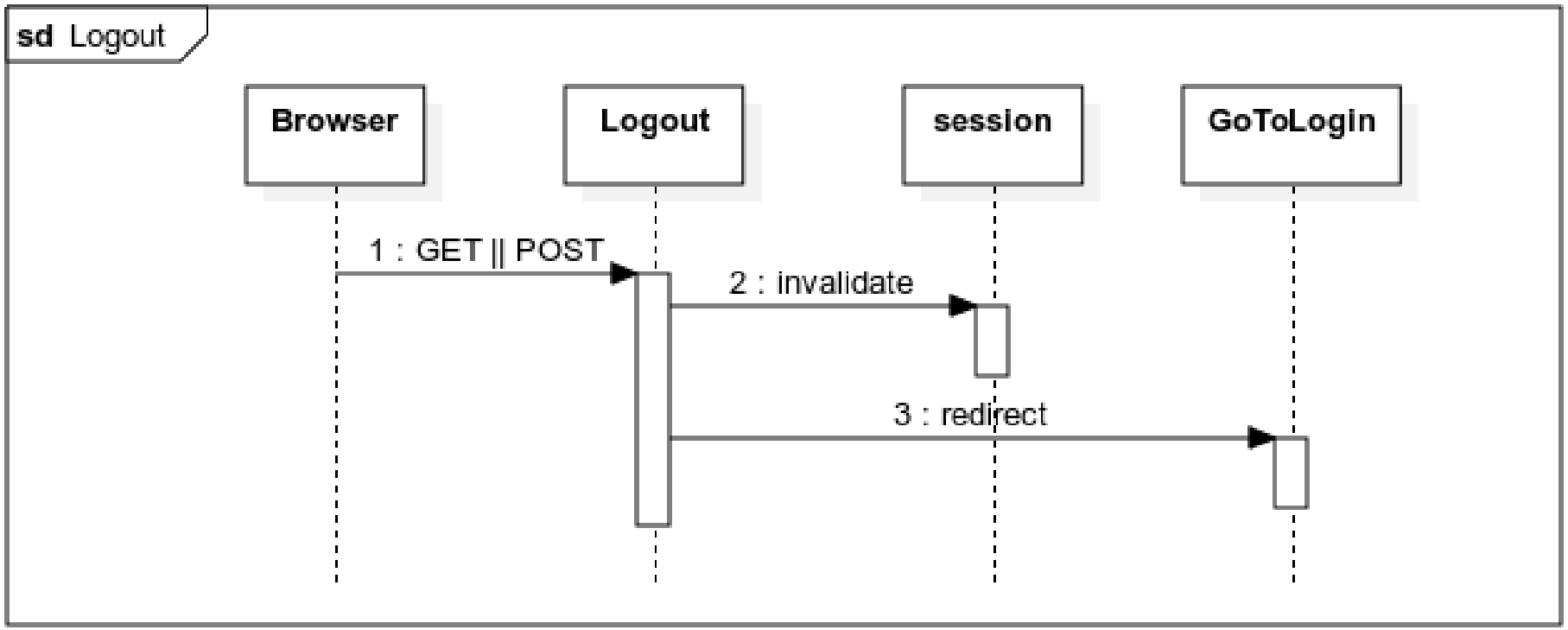

Interaction: Register
NotLoggedFilter
Register
UserDAO
Request
TemplateEngine
POST: /Register
POST
email
password
passwordRep
name
surname
Alt
[email == null || password == null || passwordRep == null || name == null || surname == null || empty]
setAttribute("backPath", Paths.pathToRegistrationServlet)
setAttribute("error", "Please fill out all the required fields")
errorRedirect(request, response, error)
[Else]
Alt
[!pattern.matcher(email).matches() ||!password.equals(passwordRep) || password.length() < 8]
setAttribute("backPath", Paths.pathToRegistrationServlet)
setAttribute("error", error)
errorRedirect(request, response, error)
[Else]
isEmailTaken(email)
taken
Alt
[taken == true]
setAttribute("backPath", Paths.pathToRegistrationServlet)
setAttribute("error", "Email" + email + "is already taken")
errorRedirect(request, response, error)
[Else]
registerUser(email, password, name, surname)
setAttribute("result", "Successfully registered")
process(Paths.pathToLoginPage, context, response.getWriter())


sd CreateAccount
NotLoggedFilter
CreateAccount
BankAccountDAO
Request
TemplateEngine
GoToHome
1 : POST CreateAccount
2 : POST
accountName
alt
[accountName == null || !pattern.matcher(email).matches()]
3 : setAttribute("backPath", Paths.pathToGoToHomeServlet)
4 : setAttribute("error", "Bad account name")
5 : sendError(request, response, error)
[else]
6 : isNameTaken(accountName)
7 : taken
alt
[taken == true]
8 : setAttribute("backPath", Paths.pathToGoToHomeServlet)
9 : setAttribute("error", "Duplicate account name")
10 : sendError(request, response, error)
[else]
11 : createAccount(user.getID, accountName, new BigDecimal)
12 : redirect


sd SelectAccount
NotLoggedFilter
SelectAccount
session
context
TemplateEngine
GET:
/SelectAccount?bankAccountID=X
1 : GET
2 : getAttribute("user")
3 : user
«create»
4 : BankAccountDAO
BankAccountDAO
«create»
5 : TransferDAO
TransferDAO
6 : findAccountByID(X)
7 : bankAccount || null
alt
[bankAccount == null]
8 : errorRedirect(request, response, "This bank account does not exist")
[bankAccount.getUserID() != user.getID()]
9 : errorRedirect(request, response, "This bank account is not yours")
[else]
10 : getTransferByAccountID(bankAccountID)
11 : transfers
12 : setVariable("account", bankAccount)
13 : setVariable("transfers", transfers)
14 : process(path, context, response.getWriter)


MakeTransfer
NotLoggedFilter
MakeTransfer
session
request
GoToTransferConfirmed
TemplateEngine
POST /MakeTransfer (recipientID, recipientUserID, senderID, amount, reason) from AccountDetails.html
1 : GET - POST
2 : getAttribute("user")
3 : user
«create»
4 : BankAccountDAO
BankAccountDAO
5 : bankAccountDAO.findAccountByID(senderID)
6 : senderAccount || null
7 : bankAccountDAO.findAccountByID(recipientID)
8 : recipientAccount || null
alt
[senderAccount == null]
9 : setAttribute("failReason", "sender account does not exist")
10 : forwardToTransferFailedDetails(request, response, path)
[senderAccount.getID() != user.getID()]
11 : setAttribute("failReason","You cannot make a transfer from this account because it is not yours")
12 : setAttribute("senderID", senderAccount.getID)
13 : forwardToTransferFailedDetails(request, response, path)
[recipientAccount == null]
14 : setAttribute("failReason", "The recipient account does not exist")
15 : setAttribute("senderid", senderAccount.getID)
16 : forwardToTransferFailedDetails(request, response, path)
[recipientAccount.getUserID() != recipientUserID]
17 : setAttribute("failReason", "Recipient user and recipient account do not match")
18 : setAttribute("senderid", senderAccount.getID)
19 : forwardToTransferFailedDetails(request, response, path)
[recipientAccount.getID() == senderAccount.getID()]
20 : setAttribute("failReason", "You cannot make transfers to the account selected")
21 : setAttribute("senderid", senderAccount.getID)
22 : forwardToTransferFailedDetails(request, response, path)
[amount.compareTo(senderAccount.getBalance()) == 1]
23 : setAttribute("failReason", "Your account cannot afford this transfer")
24 : setAttribute("senderid", senderAccount.getID)
25 : forwardToTransferFailedDetails(request, response, path)
[amount.compareTo(new BigDecimal(0)) == -1]
26 : setAttribute("failReason", "The amount must be positive")
27 : setAttribute("senderid", senderAccount.getID)
28 : forwardToTransferFailedDetails(request, response, path)
[else]
«create»
29 : TransferDAO
TransferDAO
30 : makeTransfer(amount, reason, senderID, recipientID)
31 : redirect


sd GoToTransferConfirmed
NotLoggedFilter
GoToTransferConfirmed
session
context
TemplateEngine
1 : GET
2 : getAttribute("user")
3 : user
«create»
4 : TransferDAO
TransferDAO
«create»
5 : BankAccountDAO
BankAccountDAO
6 : getLastTransferByUserID(user.getID)
7 : transfer || null
seq
[transfer == null]
8 : errorRedirect(request, response, "Unable to get the transfer from the server")
[else]
9 : findAccountByID(transfer.getSenderID)
10 : sender || null
11 : findAccountByID(transfer.getRecipientID)
12 : recipient || null
alt
[null check not ok]
13 : errorRedirect(request, response, error)
[else]
14 : setVariable("sender", sender)
15 : setVariable("recipient", recipient)
16 : setVariable("oldBalanceSenderAccount", senderOldBal)
17 : setVariable("oldBalanceRecipientAccount", recipientOldBal)
18 : setVariable("reason", reason)
19 : setVariable("amount", amount)
20 : process(path, context, response.getWriter)


sd Logout
Browser
Logout
session
GoToLogin
1 : GET || POST
2 : invalidate
3 : redirect