Esercizio 4 – pure HTML

Matteo Bettini – Andrea Aspesi

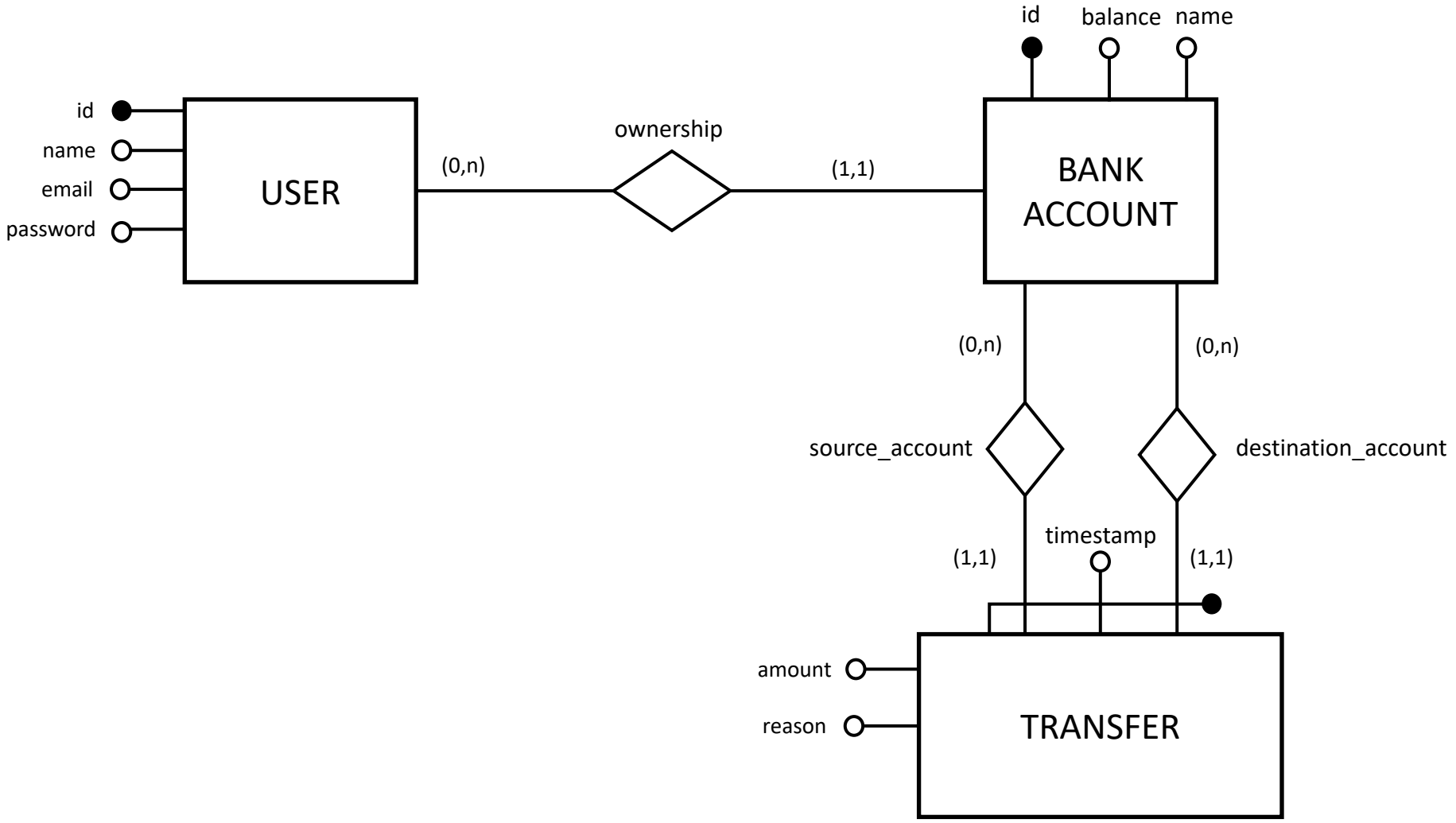
Esercizio 4: trasferimento denaro

- Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. Un utente ha un nome, un codice 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. In caso di verifica di entrambe le condizioni, l'applicazione deduce l'importo dal conto origine, aggiunge l'importo al conto destinazione e mostra una pagina CONFERMA TRASFERIMENTO che presenta i dati del conto di origine e destinazione, con i rispettivi saldi aggiornati. L'applicazione deve garantire l'atomicità del trasferimento: ogni volta che il conto di destinazione viene addebitato il conto di origine deve essere accreditato e viceversa.

Analisi dati per database

- Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. Un utente ha un nome, un codice 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. In caso di verifica di entrambe le condizioni, l'applicazione deduce l'importo dal conto origine, aggiunge l'importo al conto destinazione e mostra una pagina CONFERMA TRASFERIMENTO che presenta i dati del conto di origine e destinazione, con i rispettivi saldi aggiornati. L'applicazione deve garantire l'atomicità del trasferimento: ogni volta che il conto di destinazione viene addebitato il conto di origine deve essere accreditato e viceversa.
- Entities, attributes, relationships

Database Design



Database Schema (1/2)

```
create table `user` (
  `id` int not null auto_increment,
  `name` varchar(40) not null,
  `email` varchar(40) not null,
  `password` varchar(40) not null,
  primary key (`id`),
  unique(`email`)
) engine=InnoDB;
```

```
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 default '0.00',
  primary key (`id`),
  constraint `accountOfUser` foreign key (`userid`)
  references `user`(`id`) on delete cascade
                          on update cascade,
  constraint `positiveOrZeroBalance` check (`balance` >= 0)
) engine=InnoDB;
```

Database Schema (2/2)

```
create table `transfer` (
  `source_account` int not null,
  `destination_account` int not null,
  `timestamp` timestamp not null default current_timestamp,
  `reason` varchar(255) not null,
  `amount` decimal(10,2) not null,
  primary key (`source_account`,`destination_account`,`timestamp`),
  constraint `destinationAccount` foreign key (`destination_account`)
  references `bank_account` (`id`) on delete cascade
                                    on update cascade,
  constraint `sourceAccount` foreign key (`source_account`)
  references `bank_account` (`id`) on delete cascade
                                    on update cascade,
  constraint `positiveAmount` check (`amount` > 0)
) engine=InnoDB;
```

Analisi Requisiti Applicazione

- Un'applicazione web consente la gestione di trasferimenti di denaro online da un conto a un altro. Un utente ha un nome, un codice 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. In caso di verifica di entrambe le condizioni, l'applicazione deduce l'importo dal conto origine, aggiunge l'importo al conto destinazione e mostra una pagina CONFERMA TRASFERIMENTO che presenta i dati del conto di origine e destinazione, con i rispettivi saldi aggiornati. L'applicazione deve garantire l'atomicità del trasferimento: ogni volta che il conto di destinazione viene addebitato il conto di origine deve essere accreditato e viceversa.
- pages (views), view components, events, actions

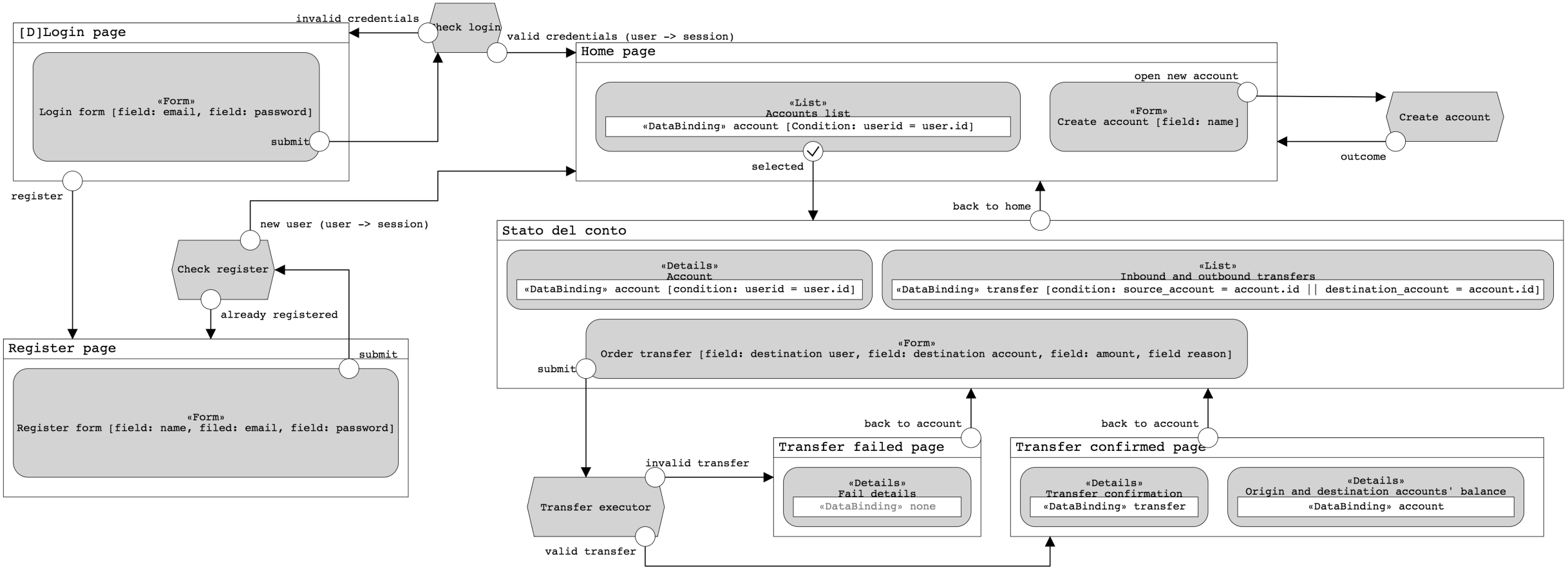
Completamento delle Specifiche

- Ogni utente viene identificato dal suo codice utente o dalla sua email.
- Per credenziali di login si intendono email e password.
- In qualsiasi pagina, se l'utente è loggato, è possibile fare logout e tornare quindi alla pagina di login
- Se l'utente è loggato e tenta di accedere alla pagine di LOGIN o REGISTER, verrà reindirizzato automaticamente alla HOME. Viceversa se l'utente non è loggato e tenta l'accesso a zone protette, verrà reindirizzato alla pagina di LOGIN.
- Se in un qualsiasi momento si verificano azioni impreviste (attributi della richiesta invalidi o tentato accesso a pagine per cui non ha l'autorizzazione), l'utente viene indirizzato a una pagina di ERRORE contenente il motivo dell'errore e dalla quale può tornare al login.
- Dalla pagina di login, è possibile spostarsi nella pagina di REGISTRAZIONE, dove inserendo nome, email e password in una form, è possibile creare un utente; se la mail fornita è già in uso l'utente viene riportato alla pagina di REGISTRAZIONE in cui gli si mostra un messaggio di errore, altrimenti viene indirizzato alla pagina di LOGIN.
- Non è possibile fare trasferimenti con conto origine uguale al conto destinazione.
- L'importo del trasferimento deve essere sempre > 0.
- Per migliorare la presentazione ogni conto ha un nome che sarà univoco per ogni utente.
- Dalla pagina home, è anche possibile creare un nuovo conto specificando il nome; se esiste già un conto di quell'utente con lo stesso nome, ciò verrà notificato nella HOME, altrimenti il conto verrà creato e l'utente reindirizzato alla HOME.

Components

- **Model objects (Beans)**
  - User
  - BankAccount
  - Transfer
- **Filters**
  - checkLoggedUser
  - checkNotLoggedUser
- **Controllers (servlets) [access rights]**
  - Login [not logged user]
  - Logout [all]
  - Register [not logged user]
  - GoToLoginPage [not logged user]
  - GoToRegisterPage [not logged user]
  - GoToHome [logged user]
  - GoToTransferConfirmedPage [logged user]
  - CreateAccount [logged user]
  - GoToAccountStatus [logged user]
  - MakeTransfer [logged user]
- **Data access objects (DAO)**
  - UserDAO
    - findUser(email, password) : User
    - getUserByID(id) : User
    - getUserByEmail(email) : User
    - registerUser(name, email, password)
  - BankAccountDAO
    - getAccountByID(id) : BankAccount
    - getAccountByUserIDAndName(id) : BankAccount
    - getAccountsByUserID(userId) : List<BankAccount>
    - createAccount(userId, name)
  - TransferDAO
    - getTransfersByAccountId(accountId): List<Transfer>
    - makeTransfer(sourceAccountId, destAccountId, reason, amount)
- **Views (Templates)**
  - login.html
  - register.html
  - header.html (included in pages when logged)
  - error.html
  - home.html
  - accountStatus.html
  - transferFailed.html
  - transferConfirmed.html

Design Applicativo (IFML)



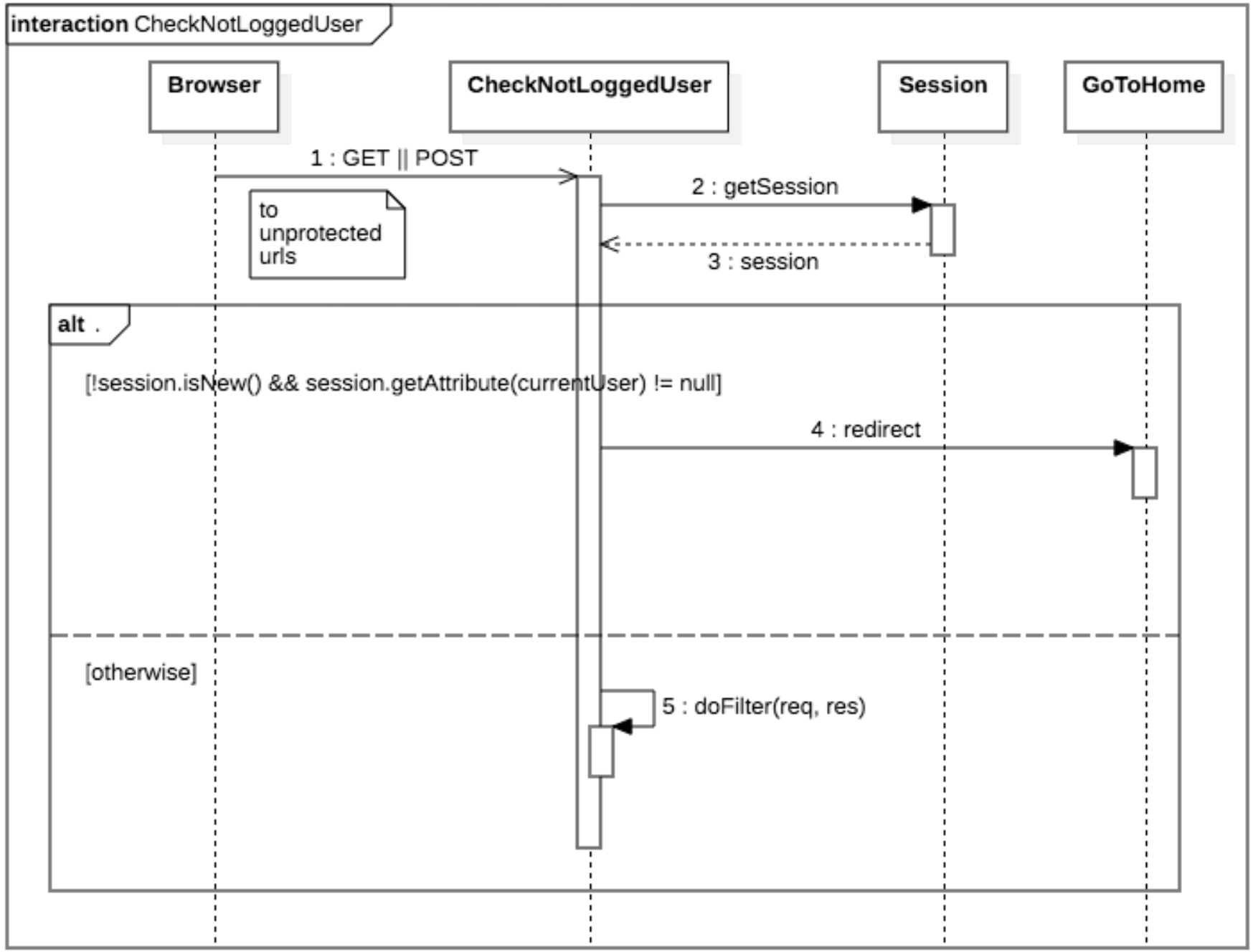
Refer to sequence diagrams for detailed interactions.

Sequence diagrams

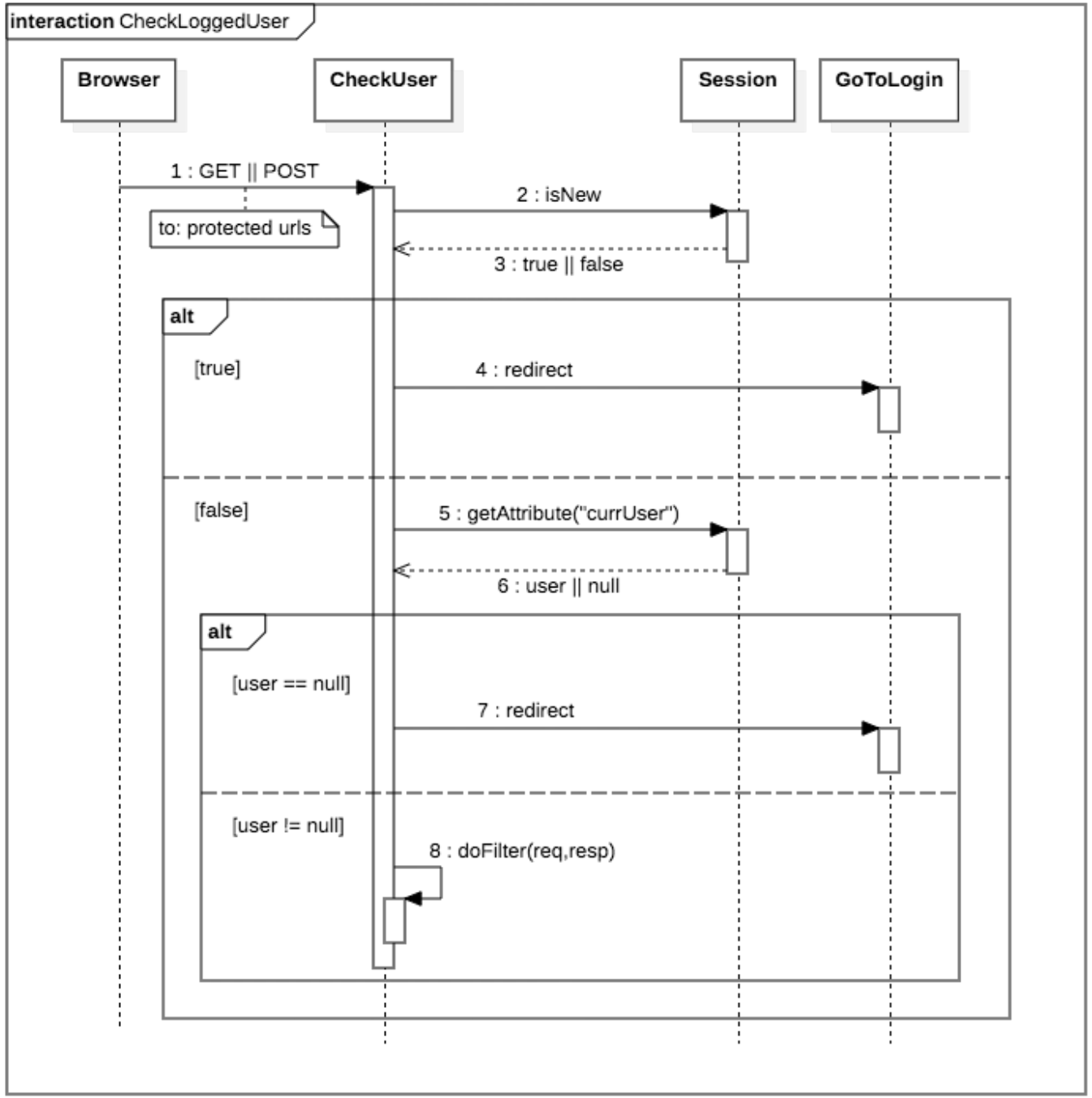
The following sequence diagrams aim to depict the interaction flow underlying the core events of the web application.
Although the authors tried their best to pursue and achieve full clarity, some minor details have been left out due to their repetitiveness (e.g. internal server errors, database access errors, null parameters).
Specifically database access errors always lead to the error page.

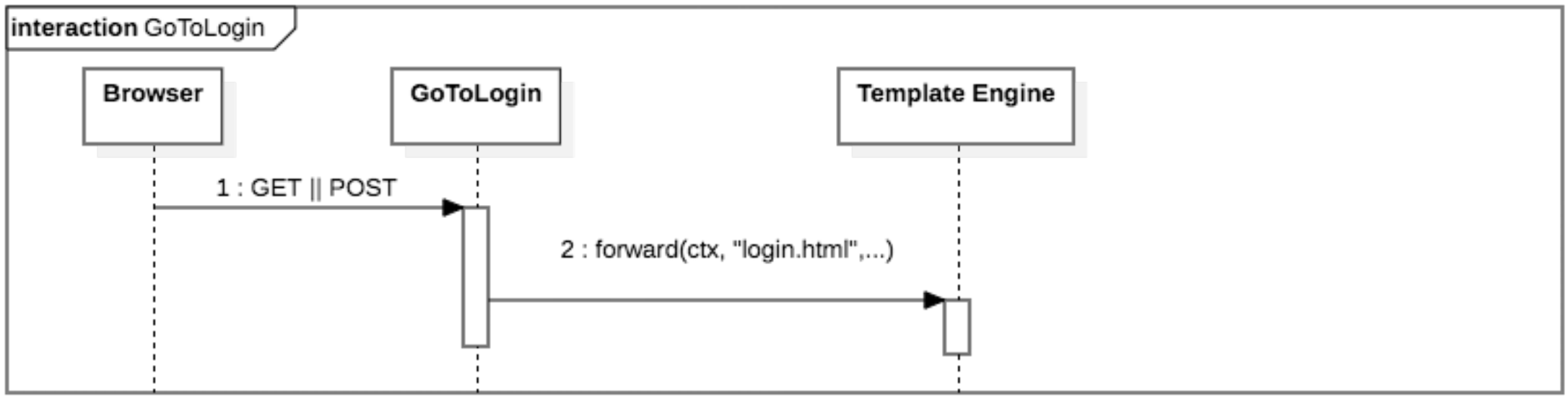
Furthermore, filter checks are represented in the first diagrams and are omitted in the subsequent ones, which does not entail that they are not called when supposed to.

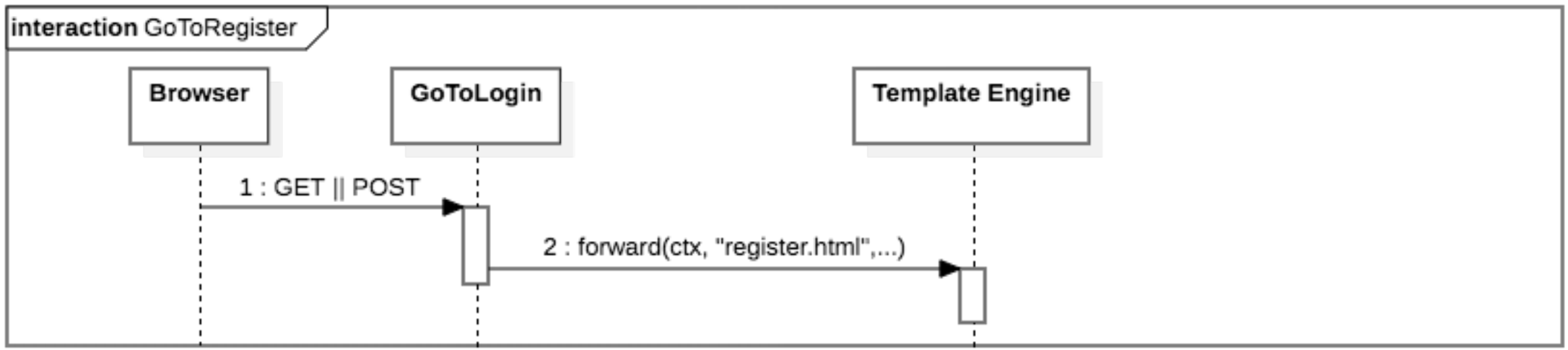
The name of the event/servlet is in the top-left corner.

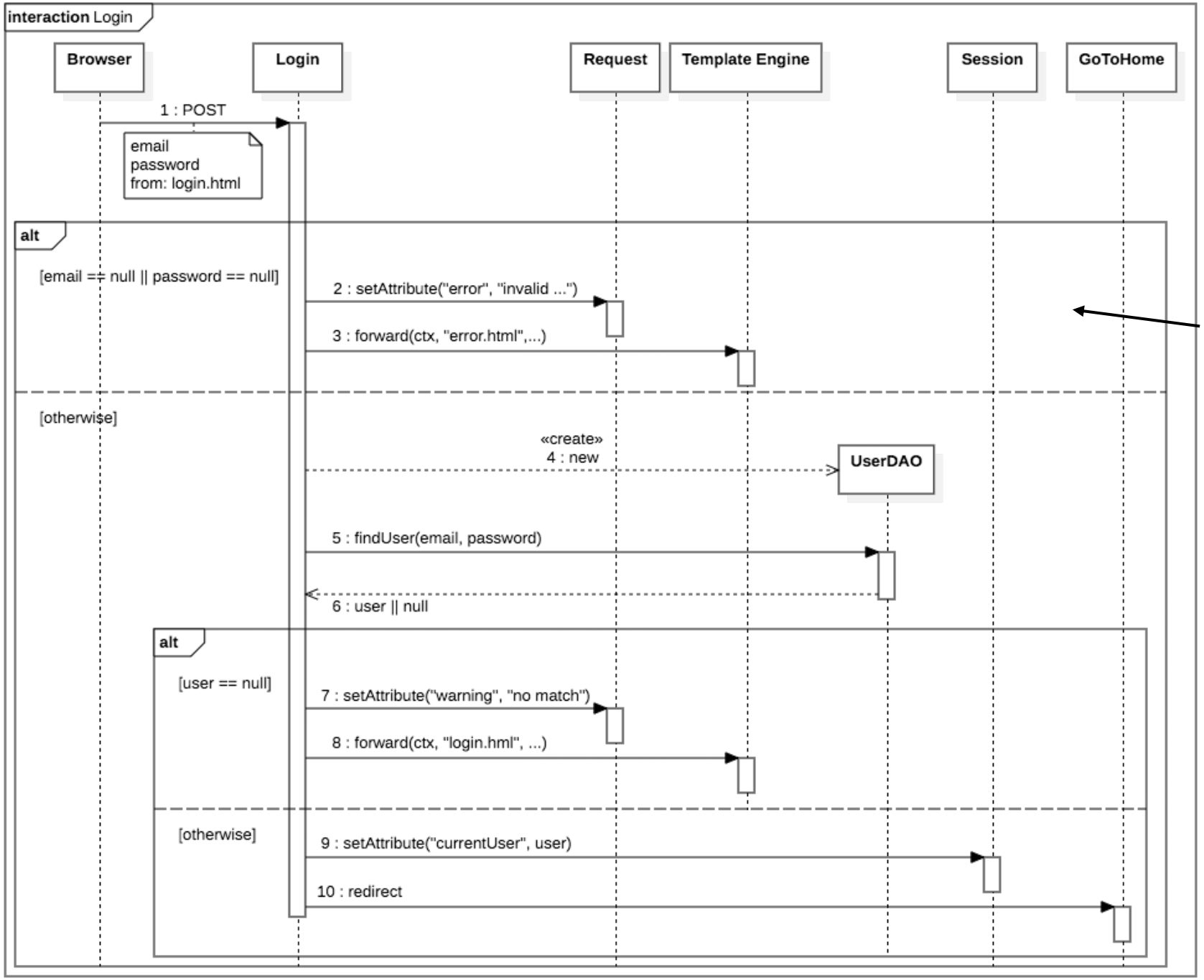

interaction CheckNotLoggedUser
Browser
CheckNotLoggedUser
Session
GoToHome
1 : GET || POST
to unprotected urls
2 : getSession
3 : session
alt .
[!session.isNew() && session.getAttribute(currentUser) != null]
4 : redirect
[otherwise]
5 : doFilter(req, res)

All the interactions reported in the following illustrate the flow considering the **filter checks already done**

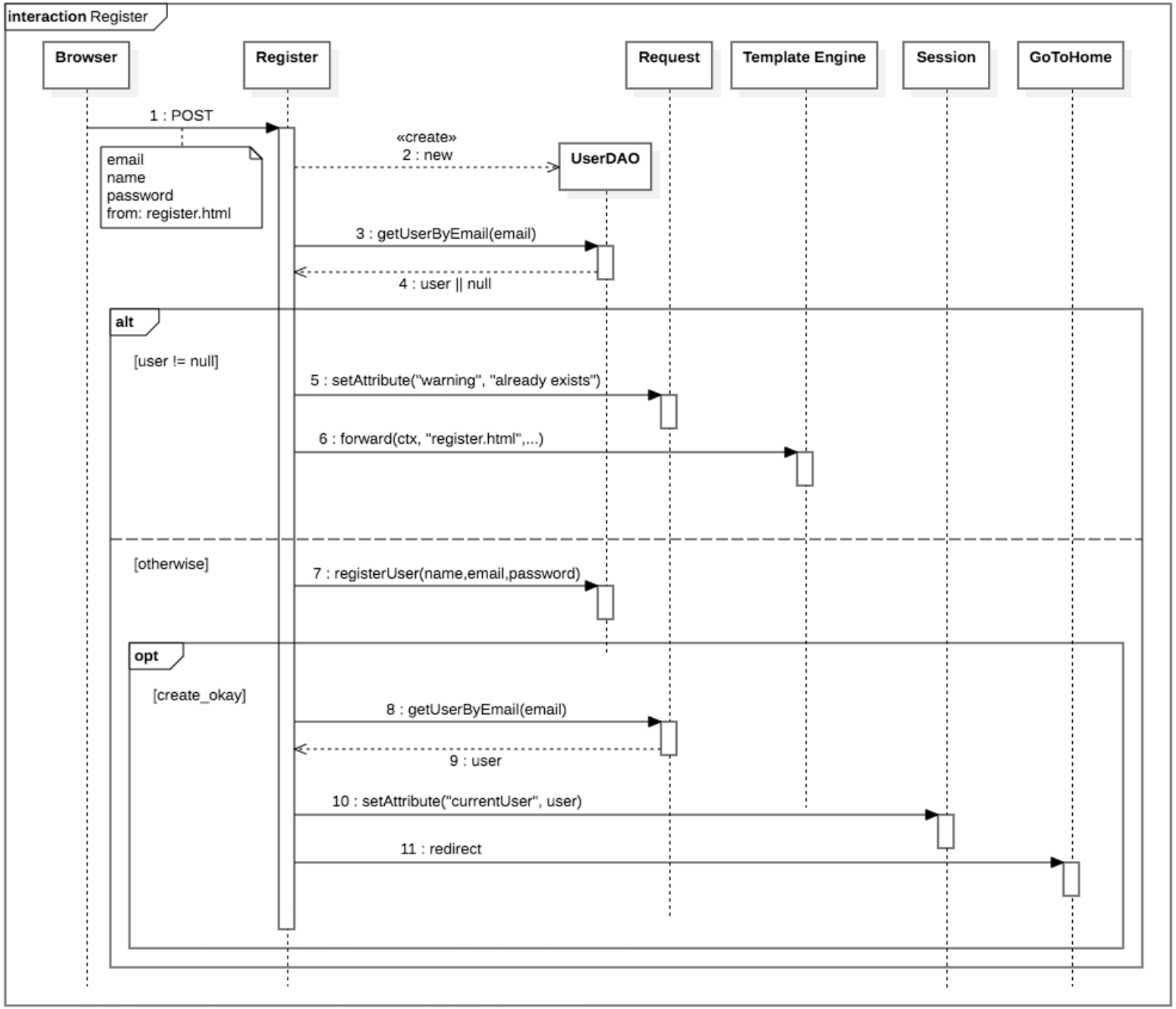


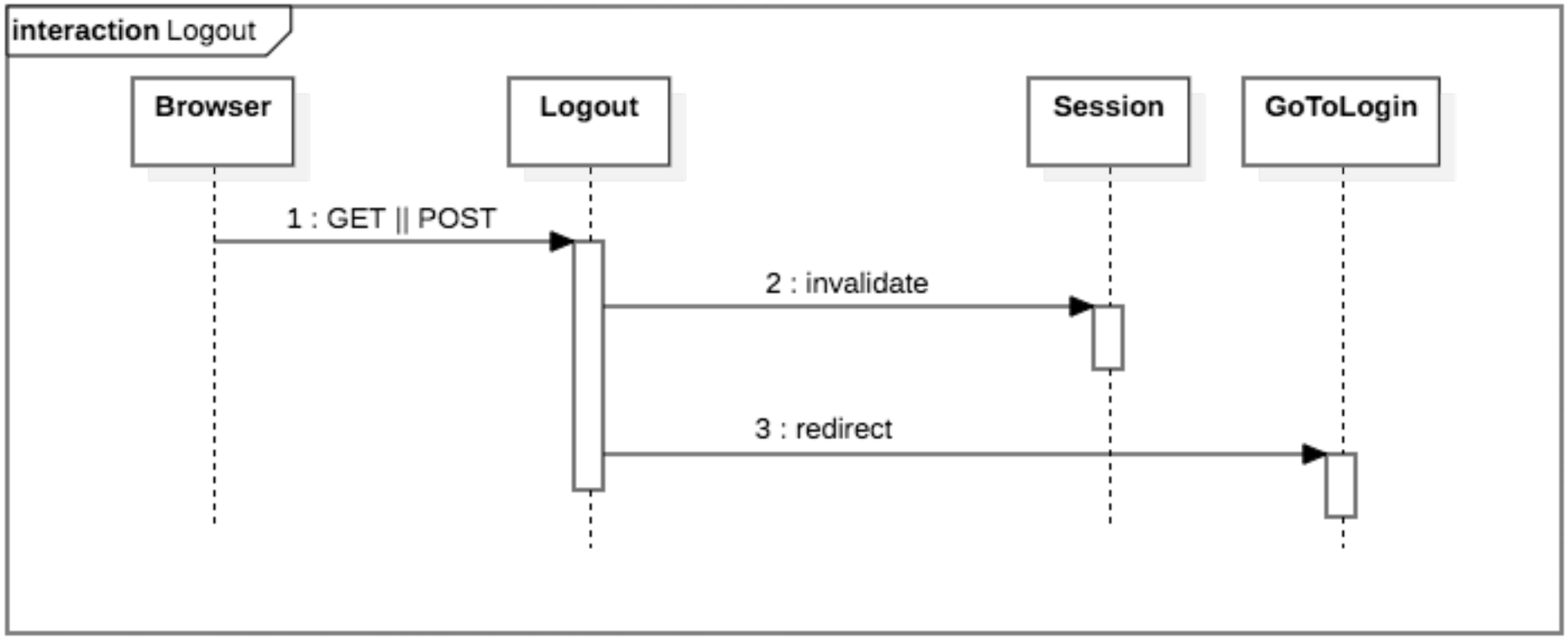

interaction GoToLogin
Browser
GoToLogin
Template Engine
1 : GET || POST
2 : forward(ctx, "login.html",...)

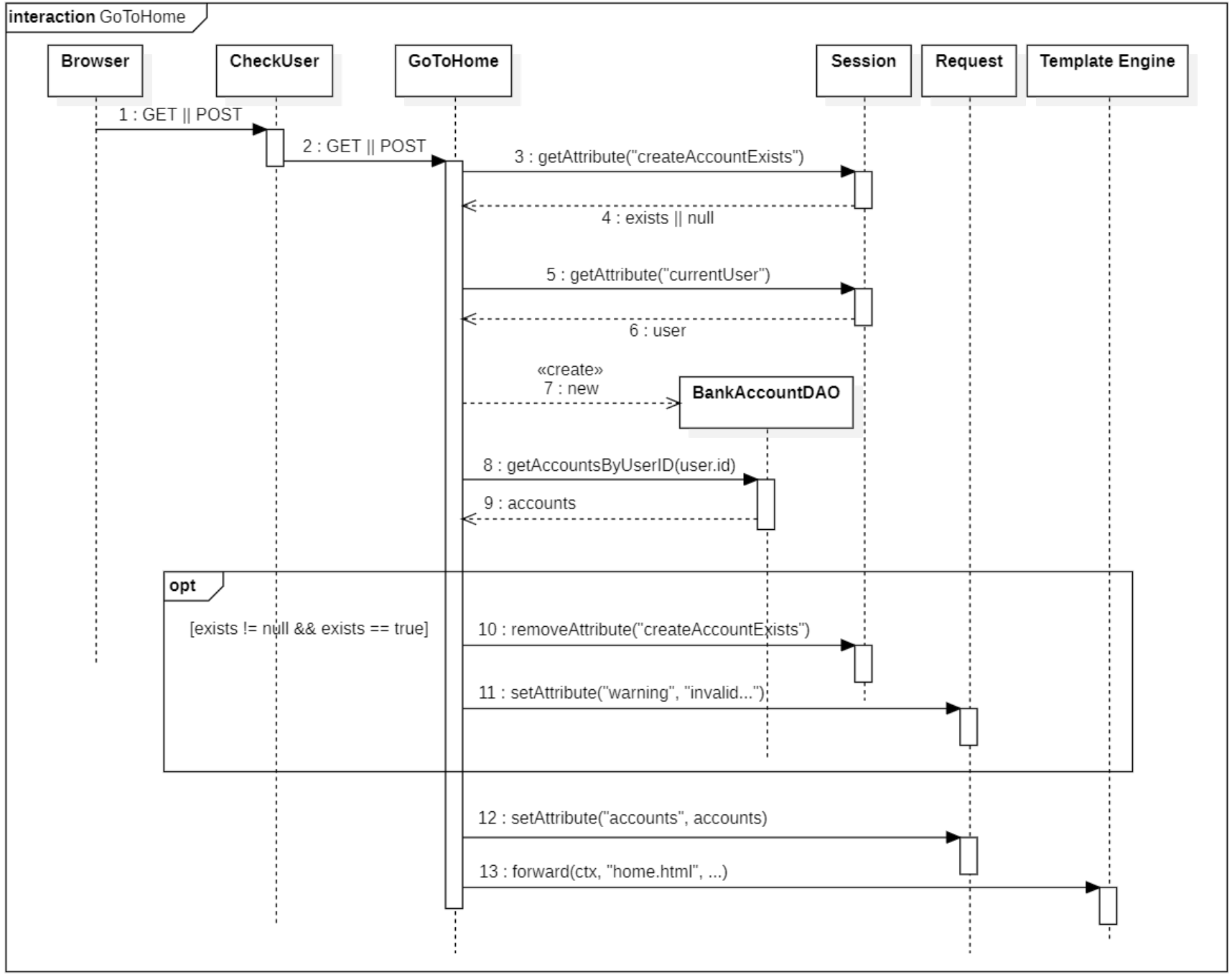

interaction GoToRegister
Browser
GoToLogin
Template Engine
1 : GET || POST
2 : forward(ctx, "register.html",...)

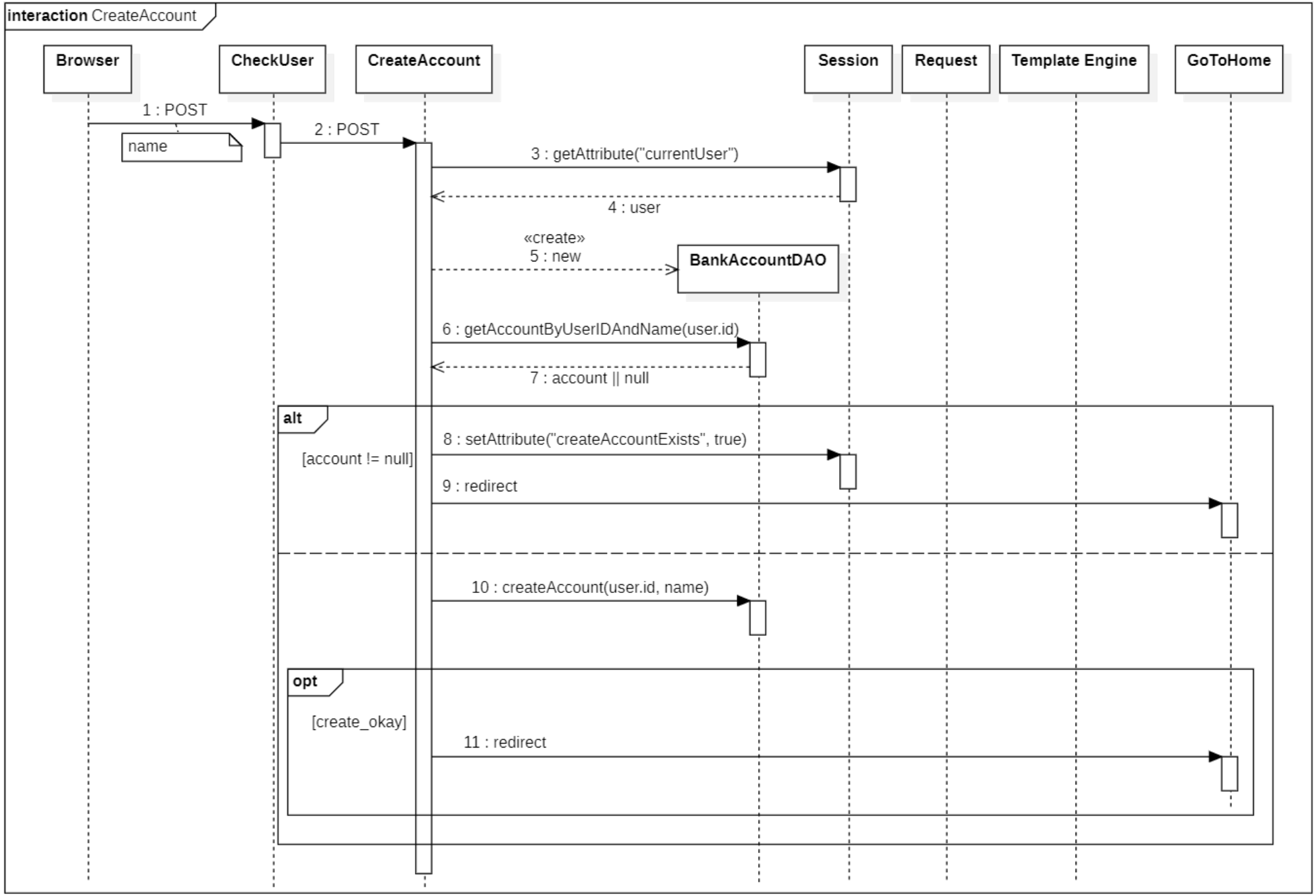


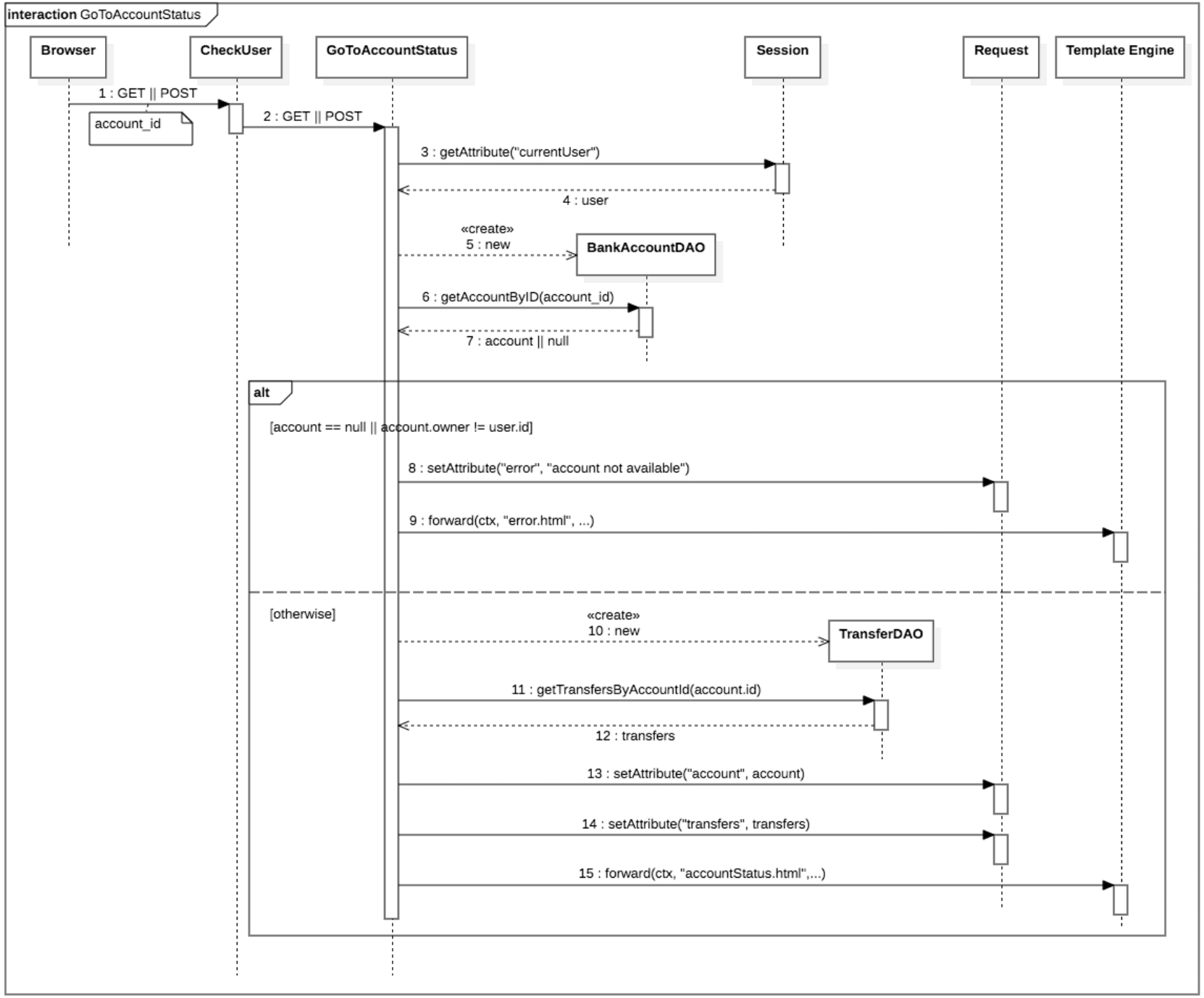
Invalid or null parameters checks will be omitted in the following sequence diagrams, as they are always treated like in this case

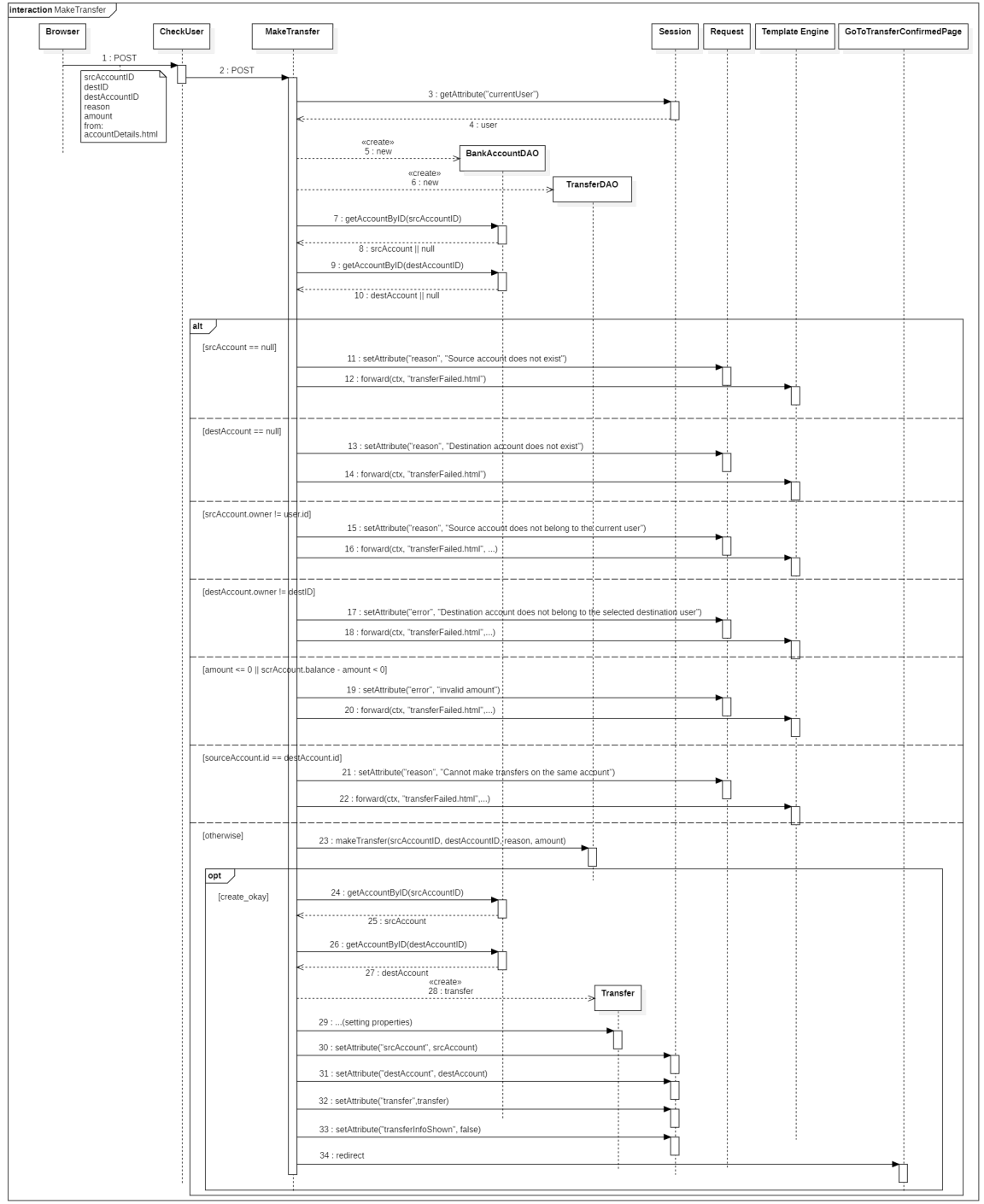

interaction Register
Browser
Register
Request
Template Engine
Session
GoToHome
1 : POST
email
name
password
from: register.html
«create»
2 : new
UserDAO
3 : getUserByEmail(email)
4 : user || null
alt
[user != null]
5 : setAttribute("warning", "already exists")
6 : forward(ctx, "register.html",...)
[otherwise]
7 : registerUser(name,email,password)
opt
[create_okay]
8 : getUserByEmail(email)
9 : user
10 : setAttribute("currentUser", user)
11 : redirect

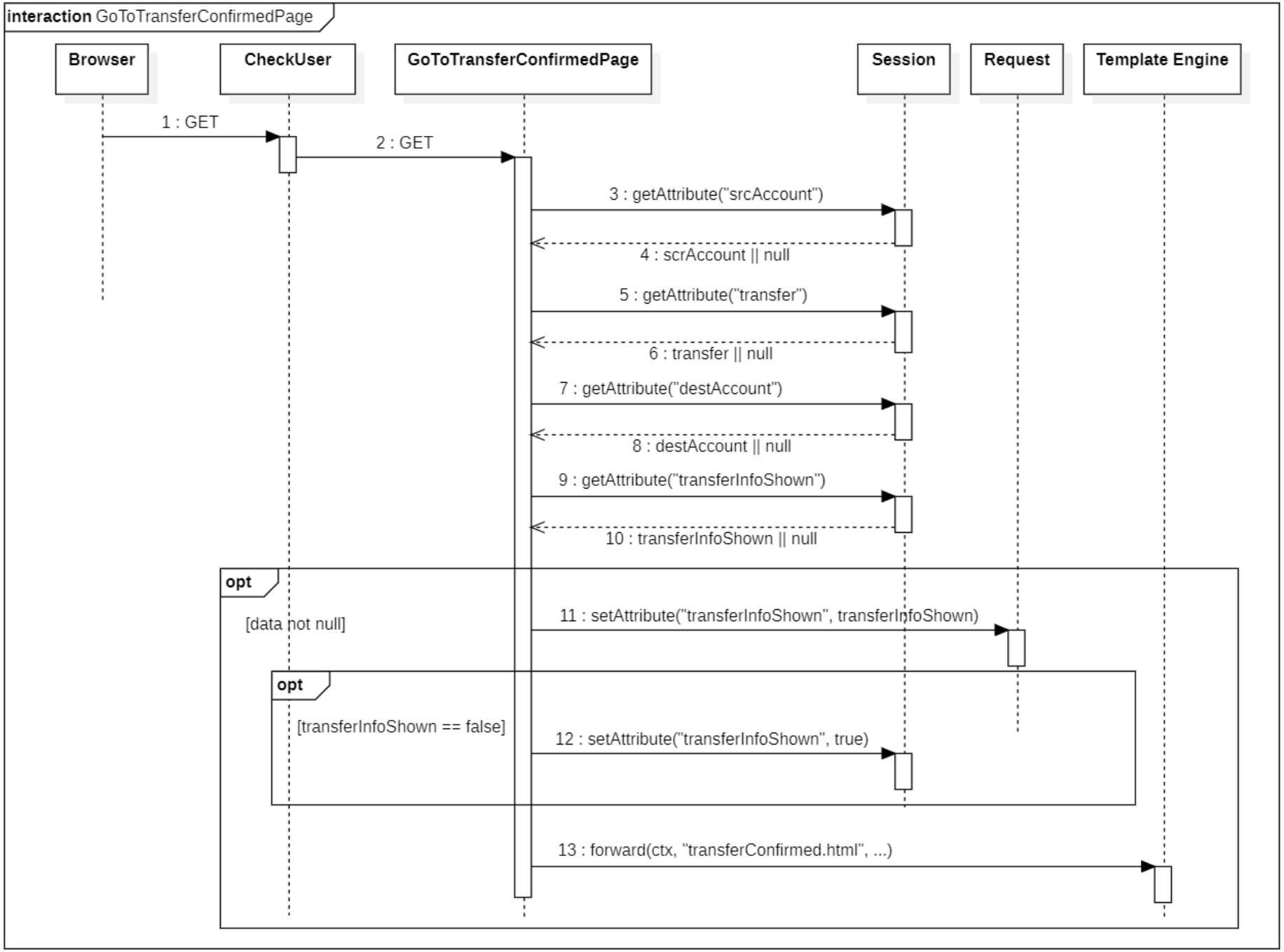

interaction Logout
Browser
Logout
Session
GoToLogin
1 : GET || POST
2 : invalidate
3 : redirect


interaction GoToHome
Browser
CheckUser
GoToHome
Session
Request
Template Engine
1 : GET || POST
2 : GET || POST
3 : getAttribute("createAccountExists")
4 : exists || null
5 : getAttribute("currentUser")
6 : user
«create»
7 : new
BankAccountDAO
8 : getAccountsByUserID(user.id)
9 : accounts
opt
[exists != null && exists == true]
10 : removeAttribute("createAccountExists")
11 : setAttribute("warning", "invalid...")
12 : setAttribute("accounts", accounts)
13 : forward(ctx, "home.html", ...)


interaction CreateAccount
Browser
CheckUser
CreateAccount
Session
Request
Template Engine
GoToHome
1 : POST
name
2 : POST
3 : getAttribute("currentUser")
4 : user
«create»
5 : new
BankAccountDAO
6 : getAccountByUserIDAndName(user.id)
7 : account || null
alt
[account != null]
8 : setAttribute("createAccountExists", true)
9 : redirect
10 : createAccount(user.id, name)
opt
[create_okay]
11 : redirect


interaction GoToAccountStatus
Browser
CheckUser
GoToAccountStatus
Session
Request
Template Engine
1 : GET || POST
account_id
2 : GET || POST
3 : getAttribute("currentUser")
4 : user
«create»
5 : new
BankAccountDAO
6 : getAccountByID(account_id)
7 : account || null
alt
[account == null || account.owner != user.id]
8 : setAttribute("error", "account not available")
9 : forward(ctx, "error.html", ...)
[otherwise]
«create»
10 : new
TransferDAO
11 : getTransfersByAccountId(account.id)
12 : transfers
13 : setAttribute("account", account)
14 : setAttribute("transfers", transfers)
15 : forward(ctx, "accountStatus.html",...)


Interaction MakeTransfer
Browser
CheckUser
MakeTransfer
Session
Request
Template Engine
GoToTransferConfirmedPage
1 : POST
srcAccountID
destID
destAccountID
reason
amount
from:
accountDetails.html
2 : POST
3 : getAttribute("currentUser")
4 : user
«create»
5 : new
BankAccountDAO
«create»
6 : new
TransferDAO
7 : getAccountByID(srcAccountID)
8 : srcAccount || null
9 : getAccountByID(destAccountID)
10 : destAccount || null
alt
[srcAccount == null]
11 : setAttribute("reason", "Source account does not exist")
12 : forward(ctx, "transferFailed.html")
[destAccount == null]
13 : setAttribute("reason", "Destination account does not exist")
14 : forward(ctx, "transferFailed.html")
[srcAccount.owner != user.id]
15 : setAttribute("reason", "Source account does not belong to the current user")
16 : forward(ctx, "transferFailed.html", ...)
[destAccount.owner != destID]
17 : setAttribute("error", "Destination account does not belong to the selected destination user")
18 : forward(ctx, "transferFailed.html",...)
[amount <= 0 || scrAccount.balance - amount < 0]
19 : setAttribute("error", "invalid amount")
20 : forward(ctx, "transferFailed.html",...)
[sourceAccount.id == destAccount.id]
21 : setAttribute("reason", "Cannot make transfers on the same account")
22 : forward(ctx, "transferFailed.html",...)
[otherwise]
23 : makeTransfer(srcAccountID, destAccountID, reason, amount)
opt
[create_okay]
24 : getAccountByID(srcAccountID)
25 : srcAccount
26 : getAccountByID(destAccountID)
27 : destAccount
«create»
28 : transfer
Transfer
29 : ...(setting properties)
30 : setAttribute("srcAccount", srcAccount)
31 : setAttribute("destAccount", destAccount)
32 : setAttribute("transfer",transfer)
33 : setAttribute("transferInfoShown", false)
34 : redirect


interaction GoToTransferConfirmedPage
Browser
CheckUser
GoToTransferConfirmedPage
Session
Request
Template Engine
1 : GET
2 : GET
3 : getAttribute("srcAccount")
4 : scrAccount || null
5 : getAttribute("transfer")
6 : transfer || null
7 : getAttribute("destAccount")
8 : destAccount || null
9 : getAttribute("transferInfoShown")
10 : transferInfoShown || null
opt
[data not null]
11 : setAttribute("transferInfoShown", transferInfoShown)
opt
[transferInfoShown == false]
12 : setAttribute("transferInfoShown", true)
13 : forward(ctx, "transferConfirmed.html", ...)