Gestione preventivi: versione RIA

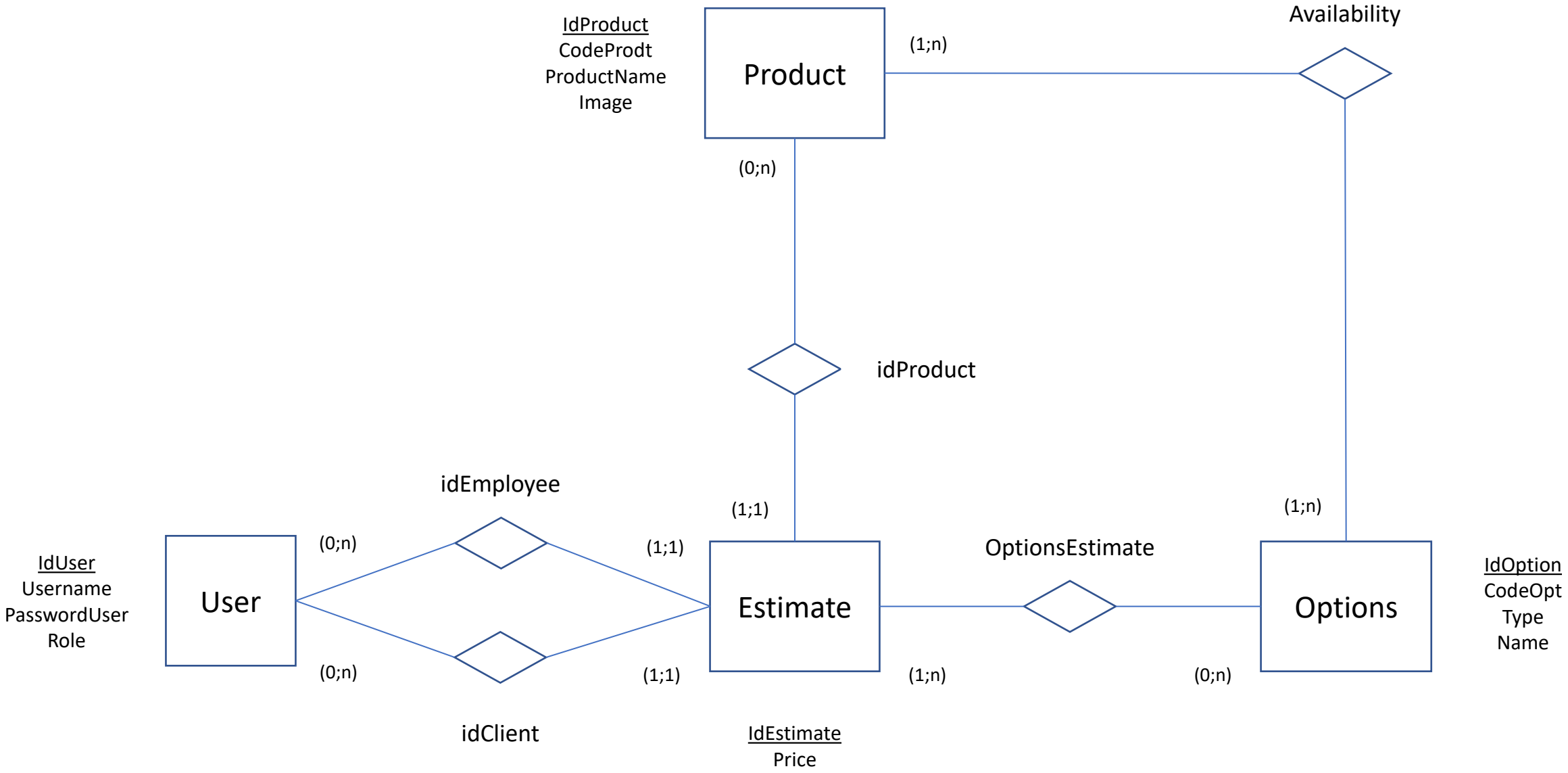
Andrea Lazzarini, Luca Muroni – Gruppo 33

Analisi dei dati

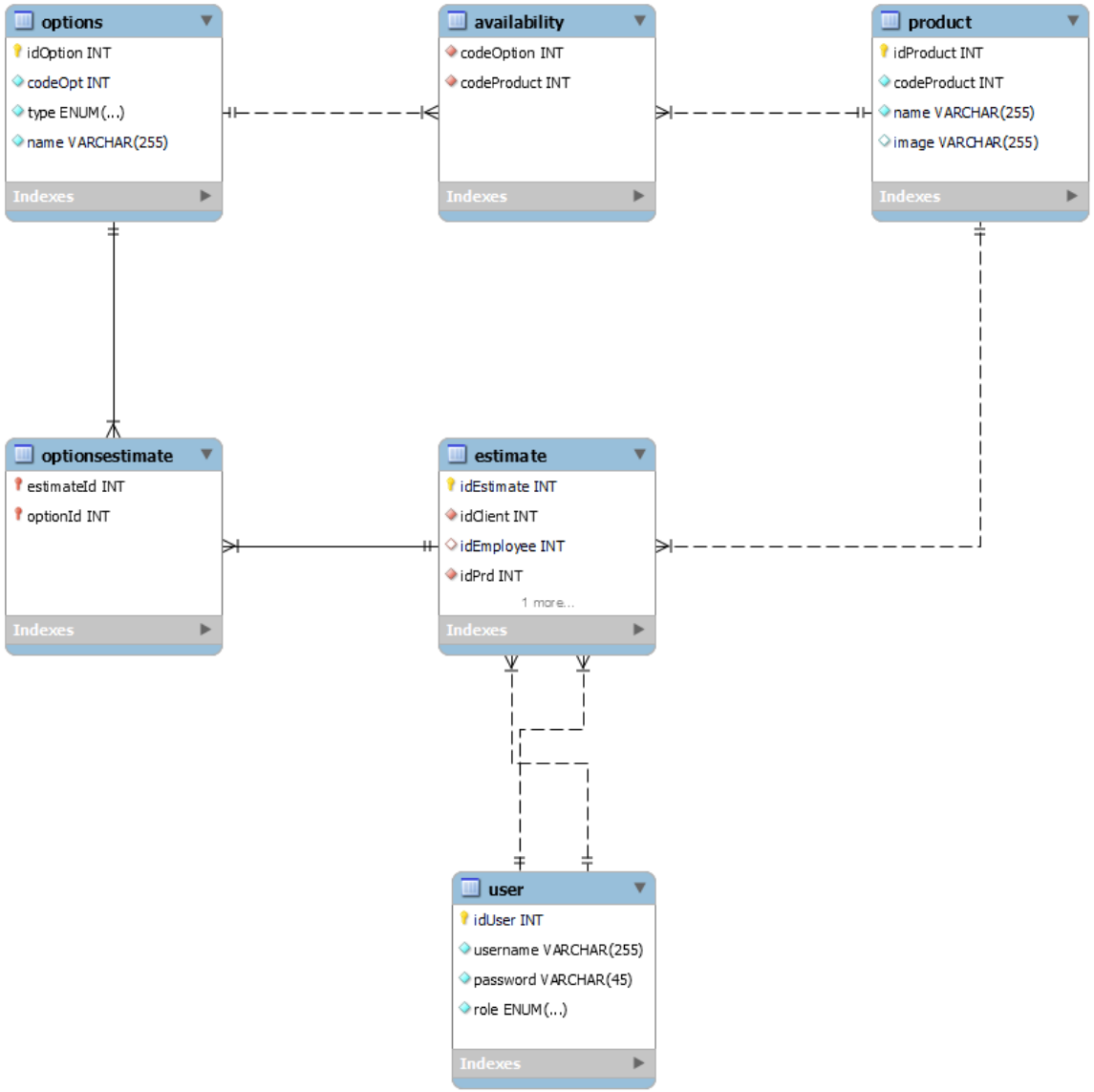
Un'applicazione web consente la gestione di richieste di preventivi per prodotti personalizzati. L'applicazione supporta registrazione e login di clienti e impiegati mediante una pagina pubblica con opportune form. La registrazione controlla l'unicità dello username. Un preventivo è associato a un prodotto, al cliente che l'ha richiesto e all'impiegato che l'ha gestito. Il preventivo comprende una o più opzioni per il prodotto a cui è associato, che devono essere tra quelle disponibili per il prodotto. Un prodotto ha un codice, un'immagine e un nome. Un'opzione ha un codice, un tipo ("normale", "in offerta") e un nome. Un preventivo ha un prezzo, definito dall'impiegato. Quando l'utente (cliente o impiegato) accede all'applicazione, appare una LOGIN PAGE, mediante la quale l'utente si autentica con username e password. Quando un cliente fa login, accede a una pagina HOME PAGE CLIENTE che contiene una form per creare un preventivo e l'elenco dei preventivi creati dal cliente. Selezionando uno dei preventivi il cliente ne visualizza i dettagli. Mediante la form di creazione di un preventivo l'utente per prima cosa sceglie il prodotto; scelto il prodotto, la form mostra le opzioni di quel prodotto. L'utente sceglie le opzioni (almeno una) e conferma l'invio del preventivo mediante il bottone INVIA PREVENTIVO. Quando un impiegato effettua il login, accede a una pagina HOME PAGE IMPIEGATO che contiene l'elenco dei preventivi gestiti da lui in precedenza e quello dei preventivi non ancora associati a nessun impiegato. Quando l'impiegato seleziona un elemento dall'elenco dei preventivi non ancora associati a nessuno, compare una pagina PREZZA PREVENTIVO che mostra i dati del cliente (username) e del preventivo e una form per inserire il prezzo del preventivo. Quando l'impiegato inserisce il prezzo e invia i dati con il bottone INVIA PREZZO, compare di nuovo la pagina HOME PAGE IMPIEGATO con gli elenchi dei preventivi aggiornati. Il prezzo definito dall'impiegato risulta visibile al cliente quando questi accede all'elenco dei propri preventivi e visualizza i dettagli del preventivo. La pagina PREZZA PREVENTIVO contiene anche un collegamento per tornare alla HOME PAGE IMPIEGATO. L'applicazione consente il logout dell'utente.

Entities, attributes, relationships

Database Design



Diagram


options
idOption INT
codeOpt INT
type ENUM(...)
name VARCHAR(255)
Indexes
availability
codeOption INT
codeProduct INT
Indexes
product
idProduct INT
codeProduct INT
name VARCHAR(255)
image VARCHAR(255)
Indexes
optionsestimate
estimateId INT
optionId INT
Indexes
estimate
idEstimate INT
idClient INT
idEmployee INT
idPrd INT
1 more...
Indexes
user
idUser INT
username VARCHAR(255)
password VARCHAR(45)
role ENUM(...)
Indexes

Local Database Schema

```
CREATE TABLE `options` (

`idOption` int NOT NULL AUTO_INCREMENT,
`codeOpt` int NOT NULL UNIQUE,
`type` enum("normal", "on_sale") NOT NULL,
`name` varchar(255) NOT NULL,
PRIMARY KEY (`idOption`)

)
```

```
CREATE TABLE `product` (

`idProduct` int NOT NULL AUTO_INCREMENT,
`codeProduct` int NOT NULL UNIQUE,
`name` varchar(255) NOT NULL,
`image` varchar(255),
PRIMARY KEY (`idProduct`)

)
```

```
CREATE TABLE `user` (

`idUser` int NOT NULL AUTO_INCREMENT,
`username` varchar(255) NOT NULL UNIQUE,
`password` varchar(45) NOT NULL,
`role` enum('employee', 'client') NOT NULL,
PRIMARY KEY (`idUser`)

)
```

```
CREATE TABLE `estimate` (

`idEstimate` int NOT NULL AUTO_INCREMENT,
`idClient` int NOT NULL,
`idEmployee` int,
`idPrd` int NOT NULL,
`price` float DEFAULT -1,
PRIMARY KEY (`idEstimate`),
KEY `client` (`idClient`),  CONSTRAINT `client` FOREIGN KEY (`idClient`)
REFERENCES `user` (`idUser`) ON DELETE NO ACTION ON UPDATE
CASCADE,
KEY `employee` (`idEmployee`),  CONSTRAINT `employee` FOREIGN KEY
(`idEmployee`) REFERENCES `user` (`idUser`) ON DELETE NO ACTION ON
UPDATE CASCADE,
KEY `prd` (`idPrd`),  CONSTRAINT `prd` FOREIGN KEY (`idPrd`) REFERENCES
`product` (`idProduct`) ON DELETE NO ACTION ON UPDATE CASCADE

)
```

Local Database Schema

```
CREATE TABLE `availability` (

`codeOption` int NOT NULL,
`codeProduct` int NOT NULL,
PRIMARY KEY (`codeOption`,`codeProduct`),
KEY `opt` (`codeOption`),  CONSTRAINT `opt` FOREIGN KEY (`codeOption`)
REFERENCES `options` (`idOption`) ON DELETE NO ACTION ON UPDATE CASCADE,
KEY `prod` (`codeProduct`),  CONSTRAINT `prod` FOREIGN KEY (`codeProduct`)
REFERENCES `product` (`idProduct`) ON DELETE NO ACTION ON UPDATE CASCADE

)
```

```
CREATE TABLE `product` (

`idProduct` int NOT NULL AUTO_INCREMENT,
`codeProduct` int NOT NULL UNIQUE,
`name` varchar(255) NOT NULL,
`image` varchar(255),
PRIMARY KEY (`idProduct`)

)
```

Application requirements analysis

- La registrazione controlla la validità sintattica dell'indirizzo di email e l'uguaglianza trai campi "password" e "ripeti password", anche a lato client.
- Dopo il login, l'intera applicazione è realizzata con un'unica pagina per ciascuno dei ruoli: una pagina singola per il ruolo di cliente e una pagina singola per il ruolo di impiegato.
- 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.
- Nella pagina del cliente, la scelta del prodotto comporta la successiva visualizzazione delle opzioni senza produrre un'ulteriore chiamata al server.
- L'invio del preventivo da parte del cliente deve produrre la verifica dei dati anche a lato client (almeno un'opzione scelta).
- Nella pagina dell'impiegato, il controllo del prezzo (non nullo e maggiore di zero) deve essere fatto anche a lato client.
- Eventuali errori a lato server devono essere segnalati mediante un messaggio di allerta all'interno della pagina del cliente o dell'impiegato

Pages (views), view components, events, actions

Aggiunta alle specifiche

- L’applicazione permette di fare logout
- La creazione di un nuovo preventivo avviene tramite l’utilizzo di un wizard. Il wizard contiene due moduli: uno per la scelta del prodotto, e il successivo per la scelta delle opzioni
- I bottoni "precedente" e "successivo" consentono di avanzare al prossimo e retrocedere al precedente modulo del wizard
- Il bottone "invia" serve per inviare i dati e creare un nuovo preventivo
- Aggiunta modalità per incrementare/decrementare prezzo di un preventivo da schermo senza bisogno di input da tastiera

Pages (views), view components, events, actions

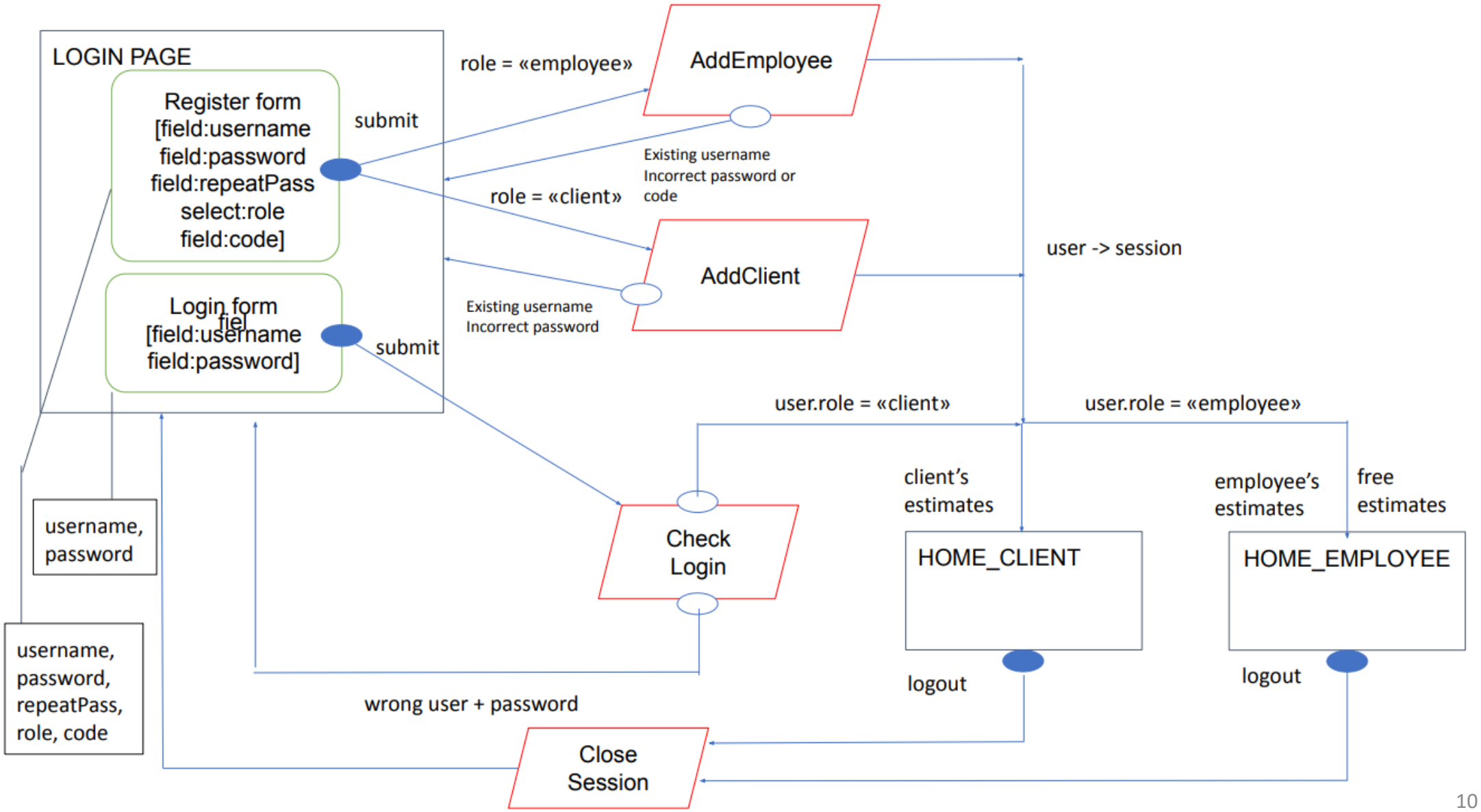
Sommario

- Viste e componenti
  - Pagina HomeClient
    - Lista preventivi del client
    - Wizard
      - Modulo scelta prodotto
      - Modulo scelta opzioni
    - Dettaglio preventivi
    - Messaggio benvenuto

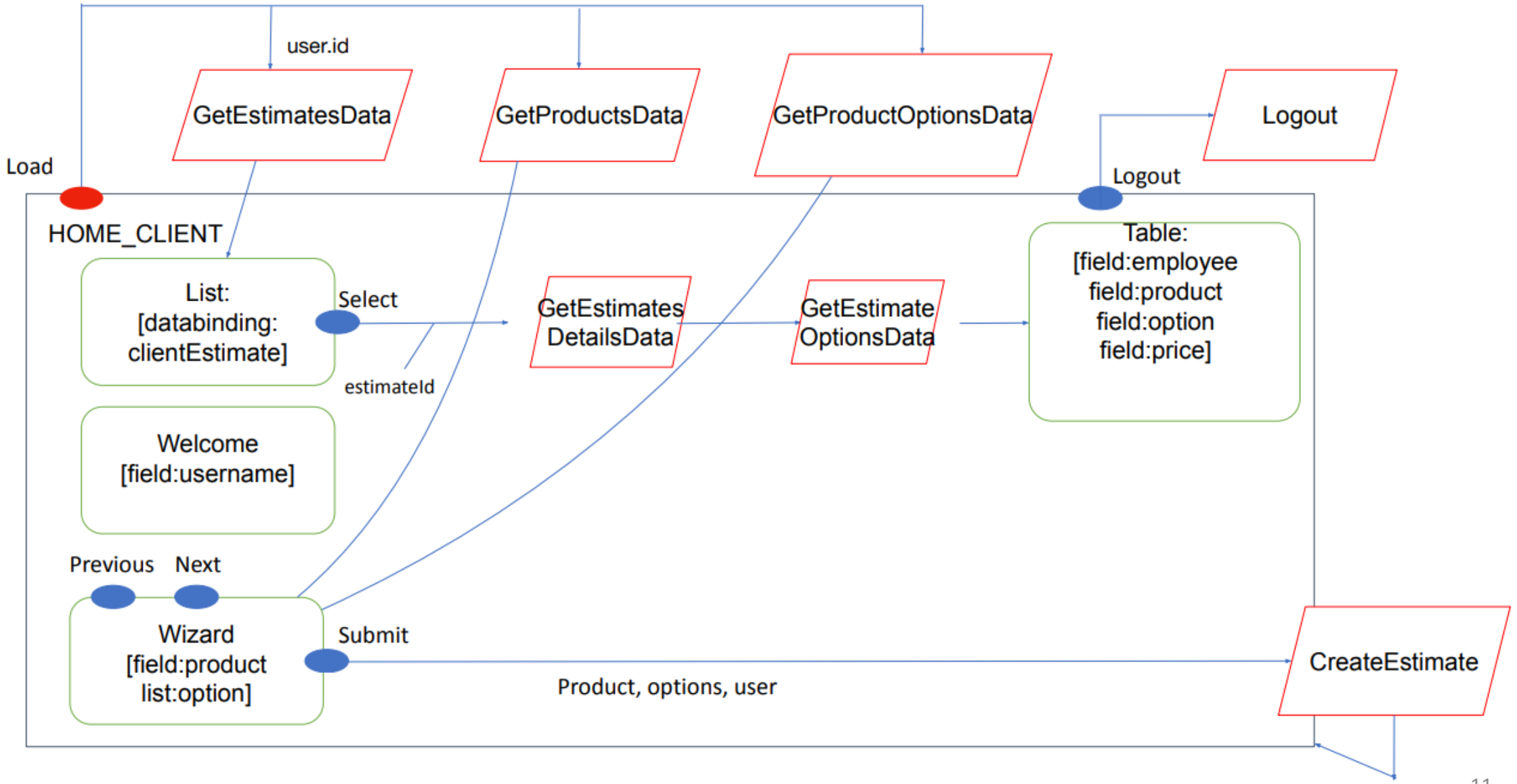
- Viste e componenti
  - Pagina HomeEmployee
    - Lista preventivi dell'employee
    - Lista preventivi non gestiti
    - Dettaglio preventivi (sia gestiti che non)
    - Messaggio benvenuto
    - Campo per l'aggiunta del prezzo nel dettaglio dei preventivi non gestiti

- Viste e componenti
  - Pagina index
    - Form per login
    - Form per registrazione
    - Campo password segreta per registrazione di employee

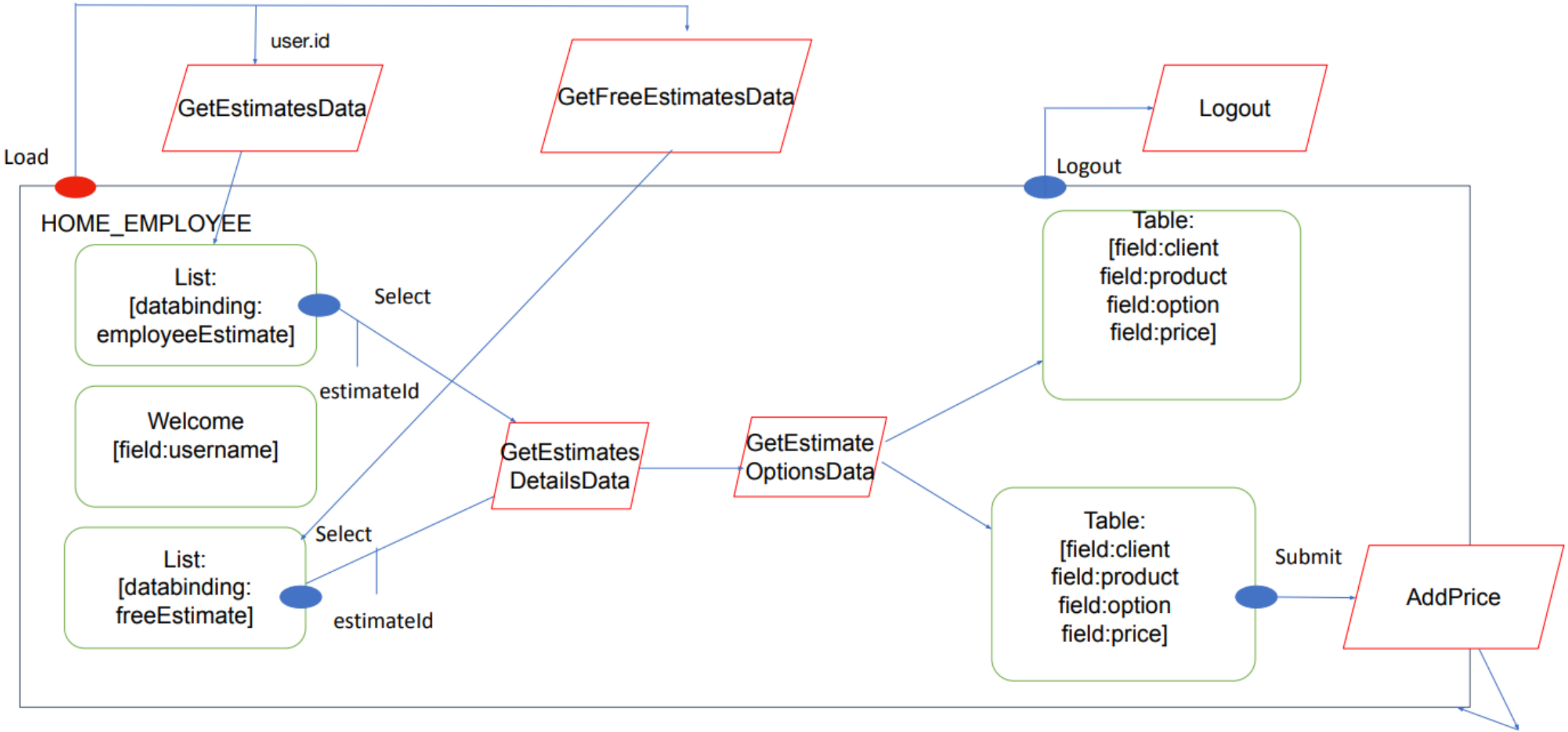
Application design



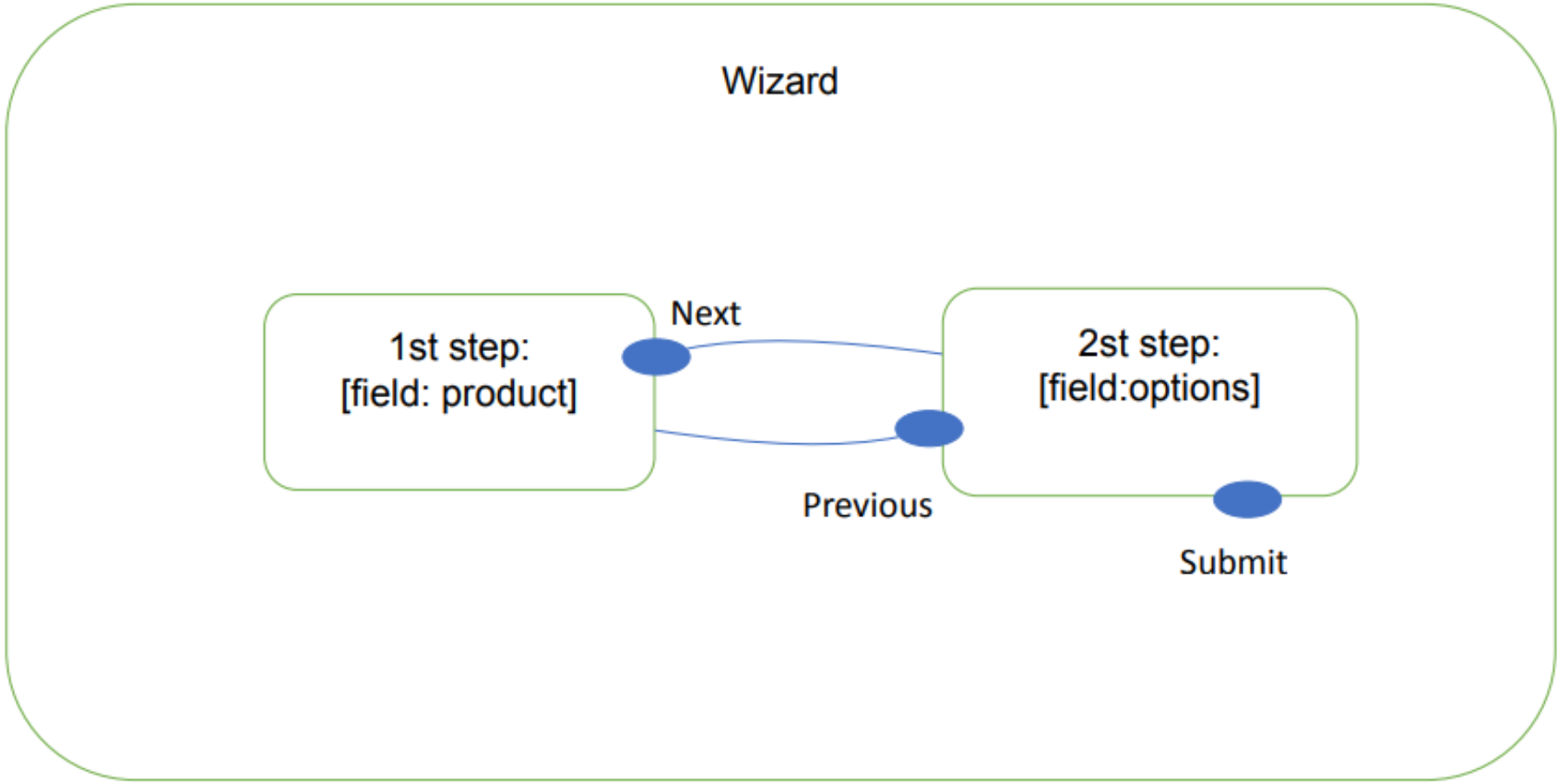
Application design



Application design



Application design



Eventi & Azioni

NB: i controlli di validità dei dati (client e server side) e di autorizzazione (server side) all'accesso sono previsti per tutti gli eventi che li richiedono e non sono riportati nella tabella per brevità

Client Side		Server Side	
Evento	**Azione**	**Evento**	**Azione**
Index → login form → submit	Controllo dati	Post(username, password)	Controllo credenziali
Index → registration form → submit	Controllo dati	Post(username, password, repeatPassword,code)	Controllo credenziali
HomeClient → load	Aggiorna view con dati elenco	Get(nessun parametro)	Estrazione estimate del client
HomeEmployee → load	Aggiorna view con dati elenco	Get(nessun parametro)	Estrazione estimate del employee e free
HomeClient → elenco estimate → seleziona estimate	Aggiorna la view con i dati dell'estimate	Get(estimateId)	Estrazione dati Estimate
HomeEmployee → elenco estimate → seleziona estimate	Aggiorna la view con i dati dell'estimate	Get(estimateId)	Estrazione dati Estimate

Eventi & Azioni

NB: i controlli di validità dei dati (client e server side) e di autorizzazione (server side) all'accesso sono previsti per tutti gli eventi che li richiedono e non sono riportati nella tabella per brevità

Client Side		Server Side	
Evento	**Azione**	**Evento**	**Azione**
HomeEmployee → elenco free estimate → seleziona estimate	Aggiorna la view con i dati dell'estimate e appare il bottone addPrice	Get(estimateId)	Estrazione dati Estimate
Wizard → next,previous	Controllo dati, cambio modulo del wizard		
Wizard → submit	Controllo dati(almeno una opzione)	Post(dati estimate)	Inserimento dell'estimate
AddPrice button → submit	Controllo dati(price maggiore di zero)	Post(price, estimateId, employee)	Inserimento nell'estimate selezionato
Logout		Get	Terminazione della sessione

Controller / Event Handler

NB: makeCall indica una funzione che fa una chiamata asincrona al server

Client Side		Server Side	
Evento	**Controllore**	**Evento**	**Controllore**
Index → login form → submit	Function makeCall	Post(username, password)	CheckLogin (servlet)
Index → registration form → submit	Function makeCall	Post(username, password, repeatPassword,code)	AddClient (servlet) AddEmployee (servlet)
HomeClient → load	Function PageOrchestrator…	Get(nessun parametro)	GetEstimatesData (Servelet) GetProductsData (Servelet) GetProductOptionsData (Servelet)
HomeEmployee → load	Function PageOrchestrator…	Get(nessun parametro)	GetEstimatesData (Servelet) GetFreeEstimatesData (Servelet)
HomeClient → elenco estimate → seleziona estimate	Function EstimateDetails.show	Get(estimateId)	GetEstimateDetailsData (Servelet) GetEstimateOptionsData (Servelet)
HomeEmployee → elenco estimate → seleziona estimate	Function EmployeeEstimateDetails.show	Get(estimateId)	GetEstimateDetailsData (Servelet) GetEstimateOptionsData (Servelet)

Controller / Event Handler

NB: makeCall indica una funzione che fa una chiamata asincrona al server

Client Side		Server Side	
Evento	**Controllore**	**Evento**	**Controllore**
HomeEmployee → elenco free estimate → seleziona estimate	Function EstimateDetails.show	Get(estimateId)	GetEstimateDetailsData (Servelet) GetEstimateOptionsData (Servelet)
Wizard → next,previous	Function Wizard		
Wizard → submit	Function Wizard	Post(dati estimate)	CreateEstimate (Servelet)
AddPrice button → submit	Function EstimateDetails	Post(price, estimateId, employee)	AddPrice (Servelet)
Logout		Get	Logout (Servelet)

Server side DAO & model objects

Controllers

- AddClient
- AddEmployee
- AddPrice
- CheckLogin
- CreateEstimate
- GetEstimateOptionsData
- GetEstimatesData
- GetEstimateDetailsData
- GetFreeEstimatesData
- GetOptionsData
- GetProductOptionsData
- GetProductsData
- Logout

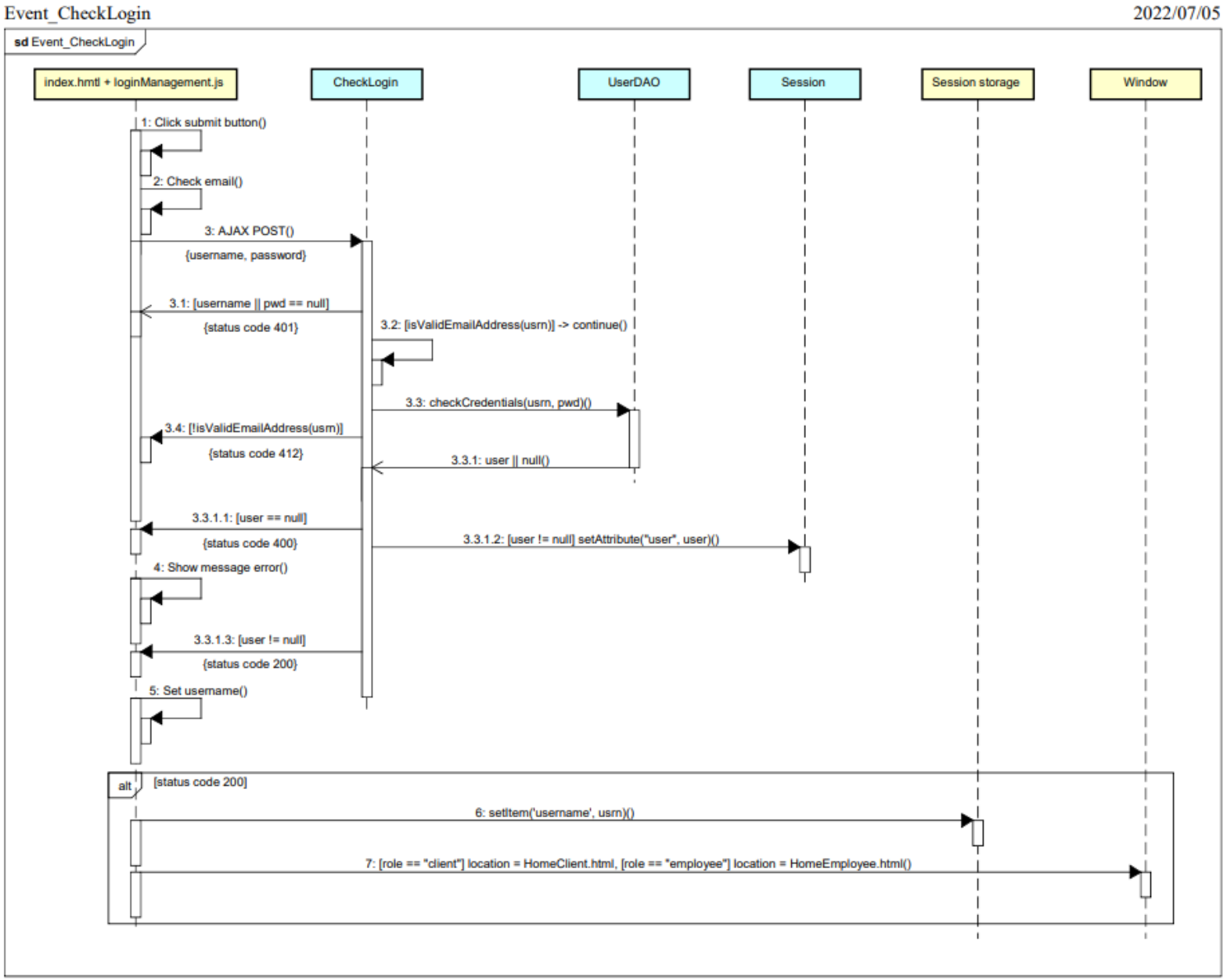
- Model Objects (Beans):
  - Estimate
  - Option
  - Product
  - User
- Data Access Objects (Classes):
  - AvailabilityDAO
    - findProductOptions(idProd)
  - EstimateDAO
    - findEstimateByClientId(clientId)
    - findEstimateById(estimateId)
    - findProductNameByEstimate(estimateId)
    - addPriceEmployee(employeeId,price,estimateId)
    - findFreeEstimates()
    - findEstimatesByEmployeeId(employeeId)
    - createEstimate(productId,clientId)
    - searchLastEstimate(clientId)
  - OptionsEstimateDAO
    - addOptionsToEstimate(idEst,optionId[])
    - findEstimateOptions(idEst)
  - ProductDAO
    - getAllProducts()
    - findProductById(idProd)
  - UserDAO
    - checkCredentials(usrn,pwd)
    - checkUsername(usrn)
    - AddCredentials(usrn,pwd,role)

Client side: view & view component

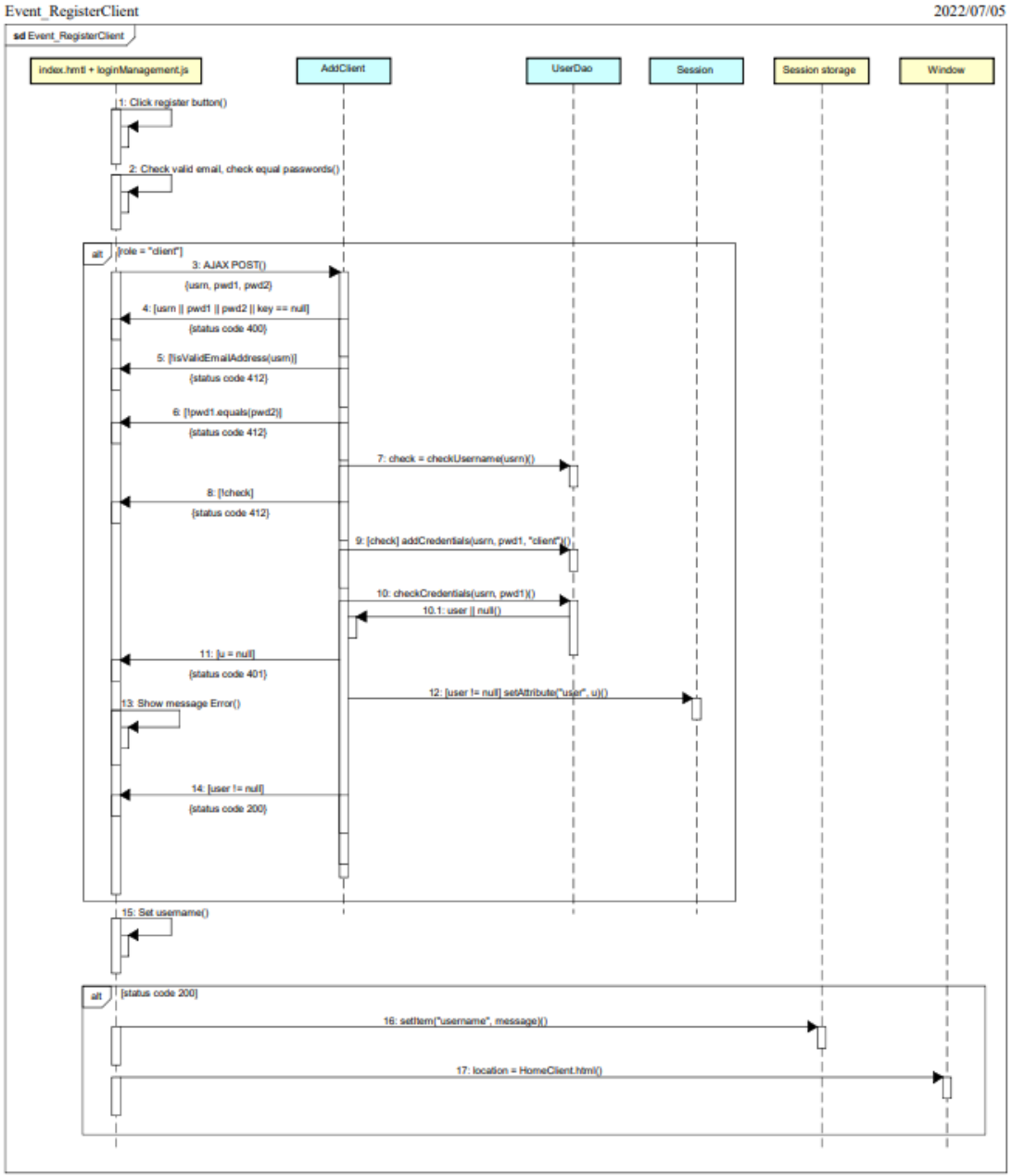
- Index
  - Login form
    - Gestione del submit per il login e degli errori
  - Registration form
    - Gestione del submit per la registrazione e degli errori
- HomeClient
  - EstimatesList
    - show(): richiede i dati al server relativi all'elenco degli estimate del client
    - update(): riceve dati dal server e aggiorna la lista
    - autoclick(): seleziona in automatico un elemento della lista per mostrarne i dettagli
  - EstimateDetails
    - show():richiede al server i dettagli dell'estimate
    - update():riceve dati dal server e aggiorna dettagli
    - updateOptions():riceve dati dal server e aggiorna dettagli della lista options
    - reset():imposta le condizioni iniziali di visibilità dei vari sottocomponenti
  - Wizard
    - registerEvents():associa al componente le funzioni per gestire gli eventi
    - show():richiede i dati al server relativi alla lista dei products
    - getOptions():richiede i dati al server relativi alla lista delle options per un prodotto
    - update():riceve i dati dal server e aggiorna immagine
    - updateOptions():riceve dati dal server e aggiorna lista delle options
    - reset():imposta le condizioni iniziali di visibilità dei moduli
    - updateOptionsVisibility():cambia visibilità delle options a seconda del product
    - changeStep():cambia il modulo visualizzato

- HomeEmployee
  - EmployeeEstimatesList
    - show():richiede i dati al server relativi all'elenco degli estimate dell'employee
    - update():riceve dati dal server e aggiorna la lista
    - reset():imposta le condizioni iniziali di visibilità dei vari sottocomponenti
    - autoclick(): seleziona in automatico un elemento della lista per mostrarne i dettagli
  - EmployeeEstimateDetails
    - show():richiede al server i dettagli dell'estimate
    - update():riceve dati dal server e aggiorna dettagli
    - updateOptions():riceve dati dal server e aggiorna dettagli della lista options
    - reset():imposta le condizioni iniziali di visibilità dei vari sottocomponenti
  - EstimatesList
    - show():richiede i dati al server relativi all'elenco degli estimate non ancora prezzati
    - update():riceve dati dal server e aggiorna la lista
    - reset():imposta le condizioni iniziali di visibilità dei vari sottocomponenti
    - autoclick(): seleziona in automatico un elemento della lista per mostrarne i dettagli
  - EstimateDetails
    - registerEvents():associa al componente le funzioni per gestire gli eventi
    - show():richiede al server i dettagli dell'estimate
    - update():riceve dati dal server e aggiorna dettagli
    - updateOptions():riceve dati dal server e aggiorna dettagli della lista options
    - reset():imposta le condizioni iniziali di visibilità dei vari sottocomponenti

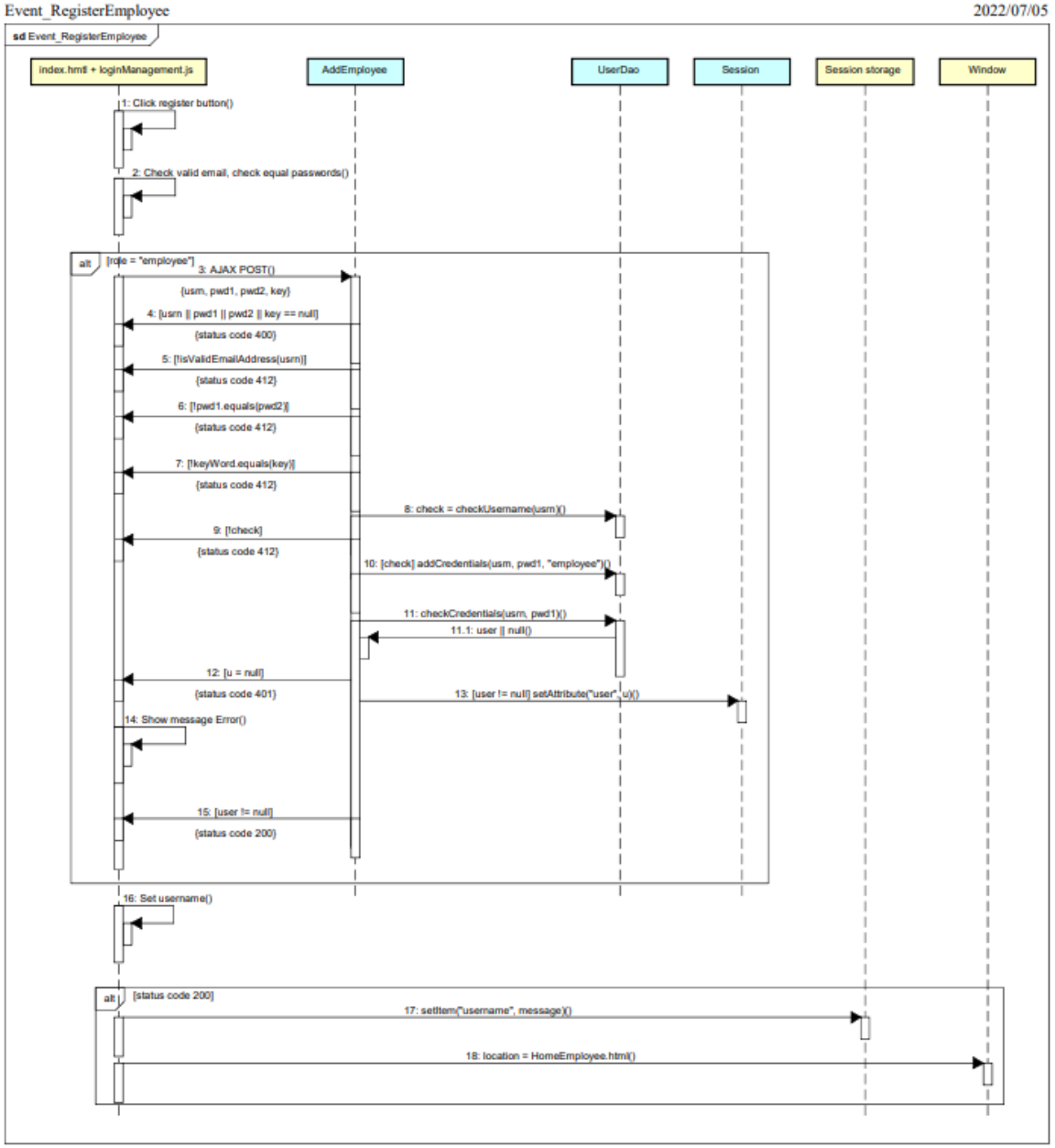
Event: Login


Event_CheckLogin
2022/07/05
sd Event_CheckLogin
index.hmtl + loginManagement.js
CheckLogin
UserDAO
Session
Session storage
Window
1: Click submit button()
2: Check email()
3: AJAX POST()
{username, password}
3.1: [username || pwd == null]
{status code 401}
3.2: [isValidEmailAddress(usrn)] -> continue()
3.3: checkCredentials(usrn, pwd)()
3.4: [!isValidEmailAddress(usrn)]
{status code 412}
3.3.1: user || null()
3.3.1.1: [user == null]
{status code 400}
3.3.1.2: [user != null] setAttribute("user", user)()
4: Show message error()
3.3.1.3: [user != null]
{status code 200}
5: Set username()
alt
[status code 200]
6: setItem('username', usrn)()
7: [role == "client"] location = HomeClient.html, [role == "employee"] location = HomeEmployee.html()

Event: AddNewClient

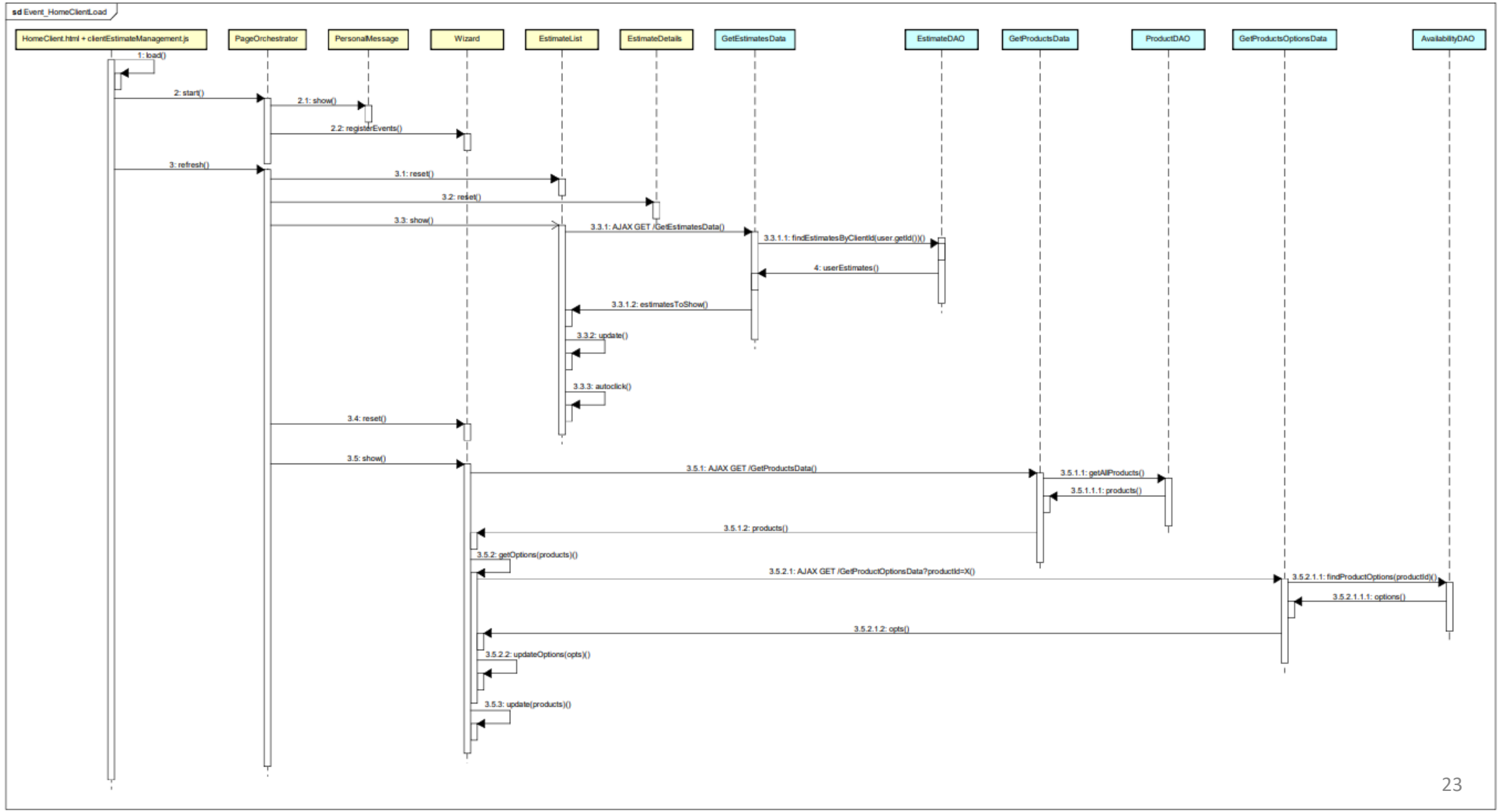


Event: AddNewEmployee



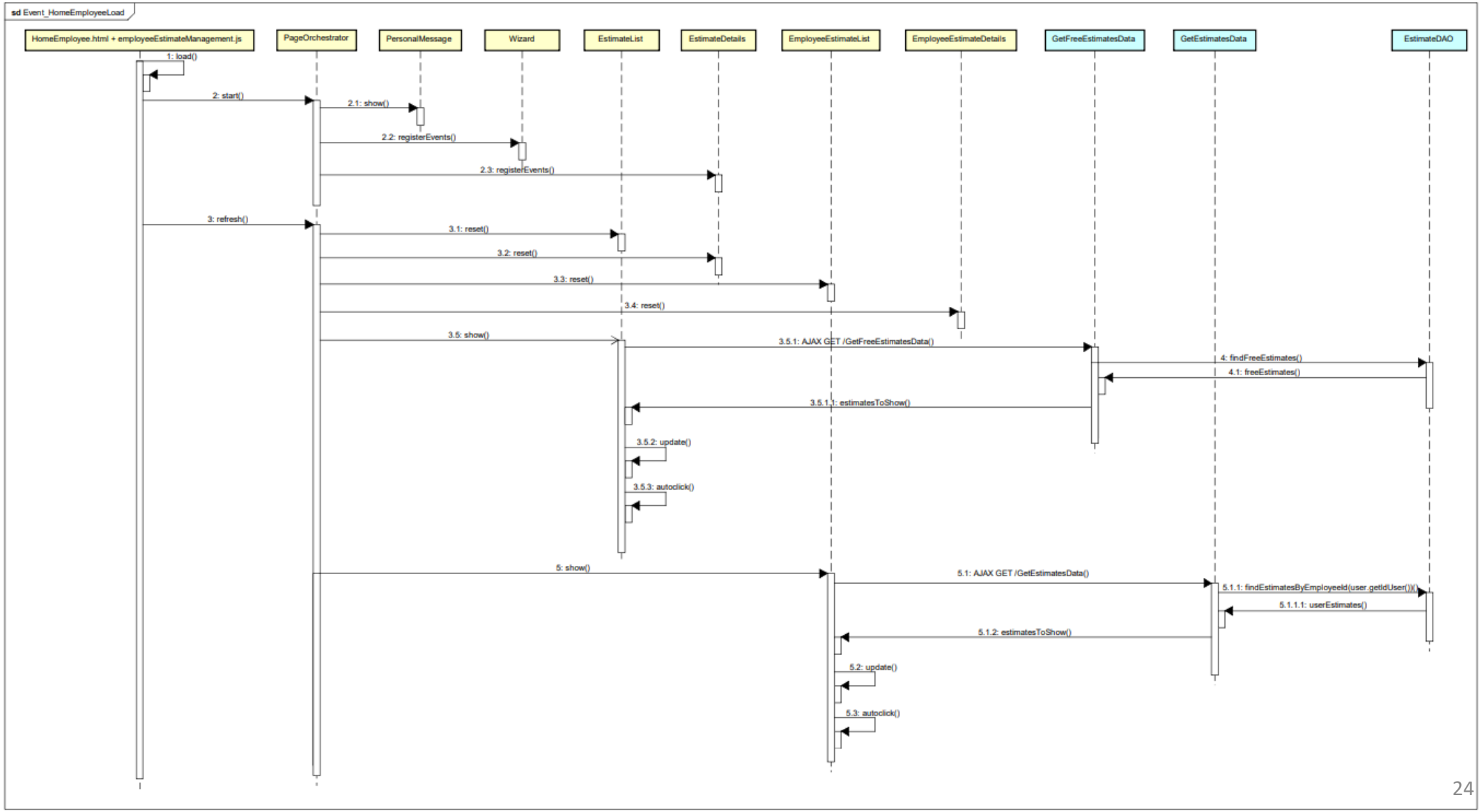
Event: HomeClientLoad

Event_HomeClientLoad 2022/07/05

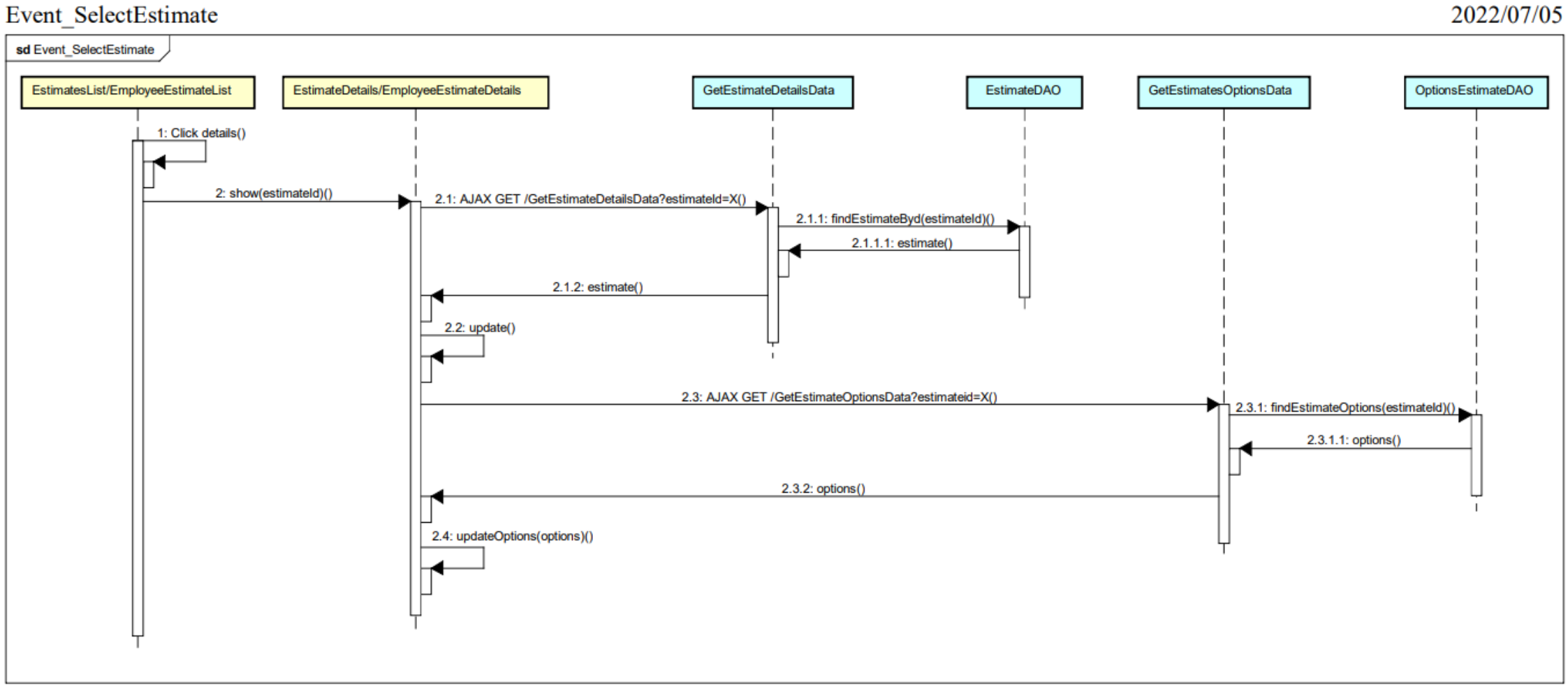


Event: HomeEmployeeLoad

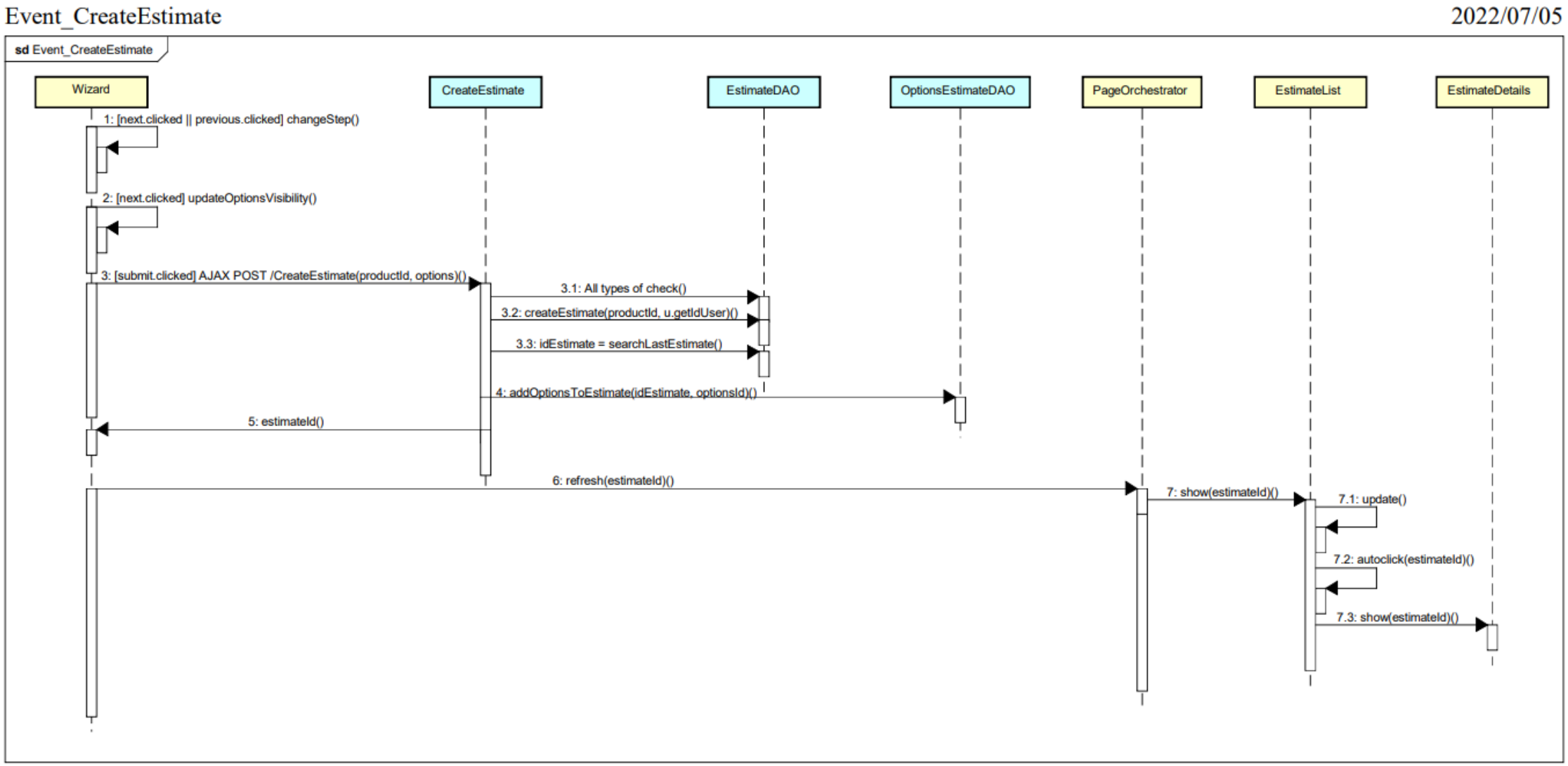
Event_HomeEmployeeLoad 2022/07/05



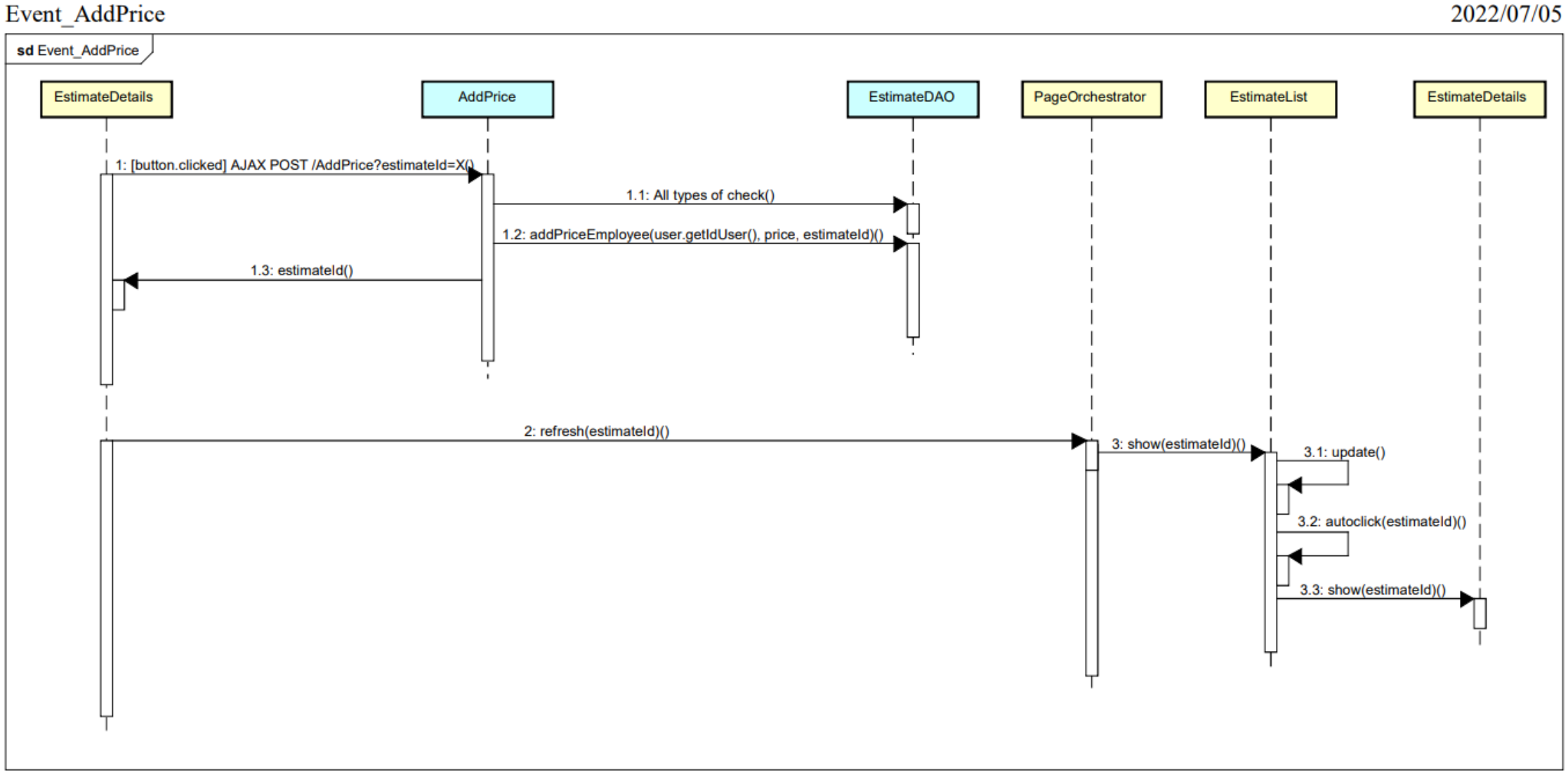
Event: SelectEstimate



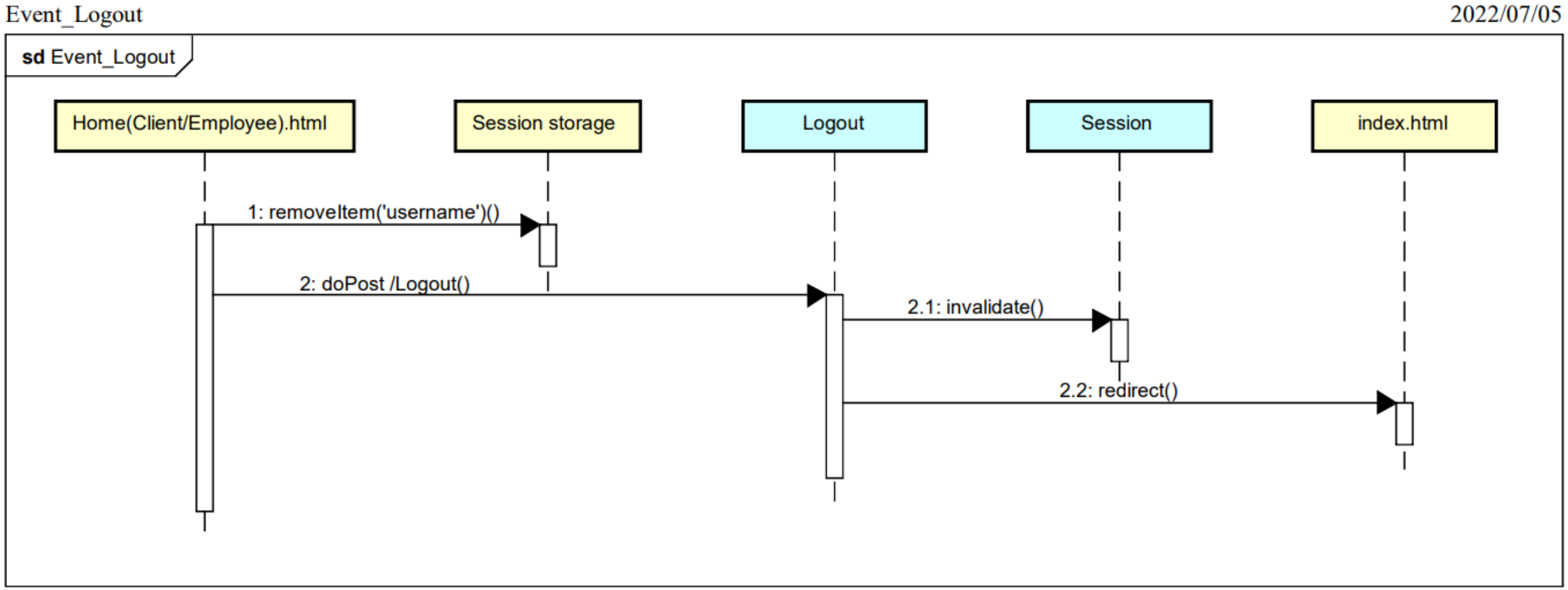
Event: CreateEstimate



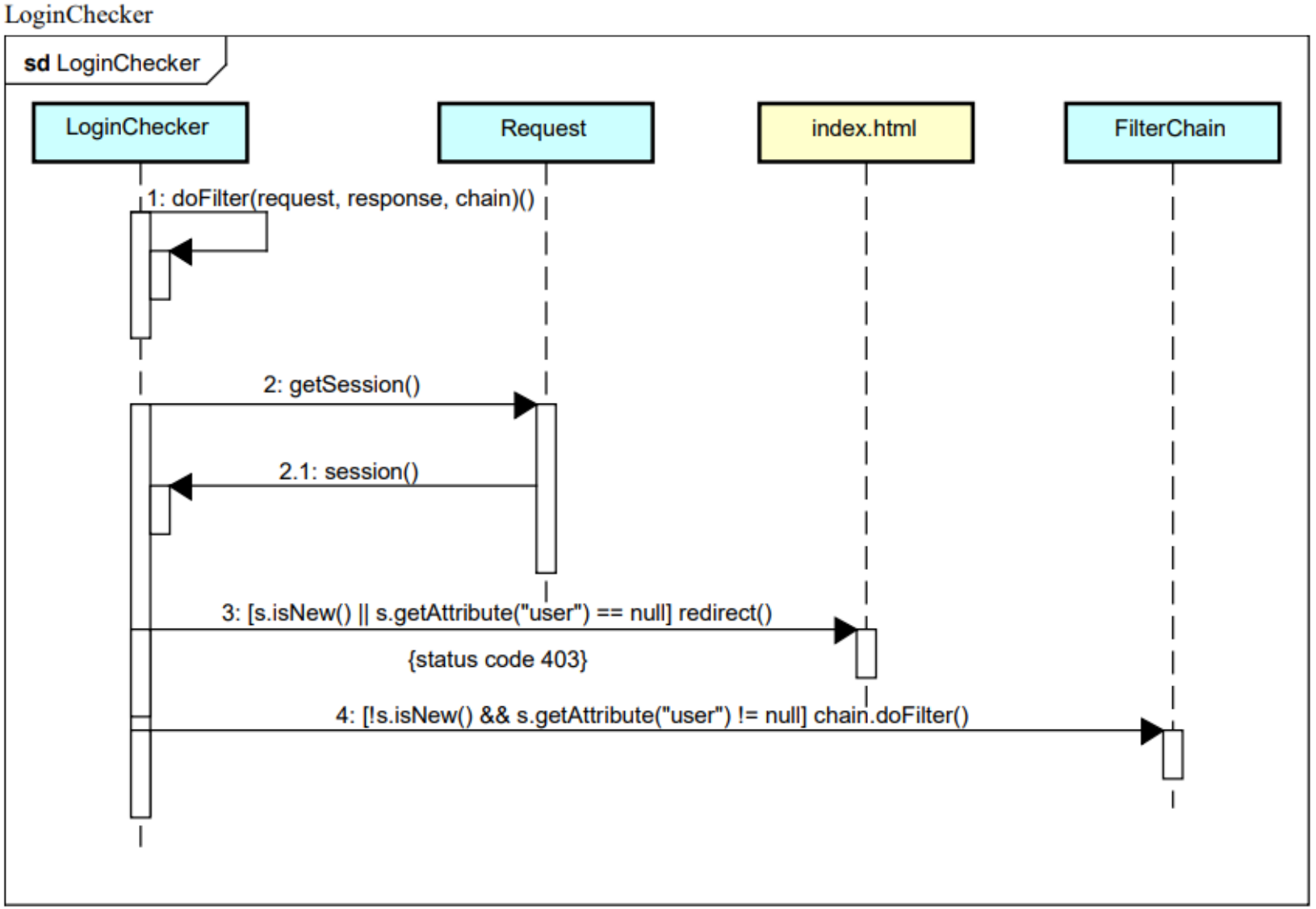
Event: AddPriceEstimate



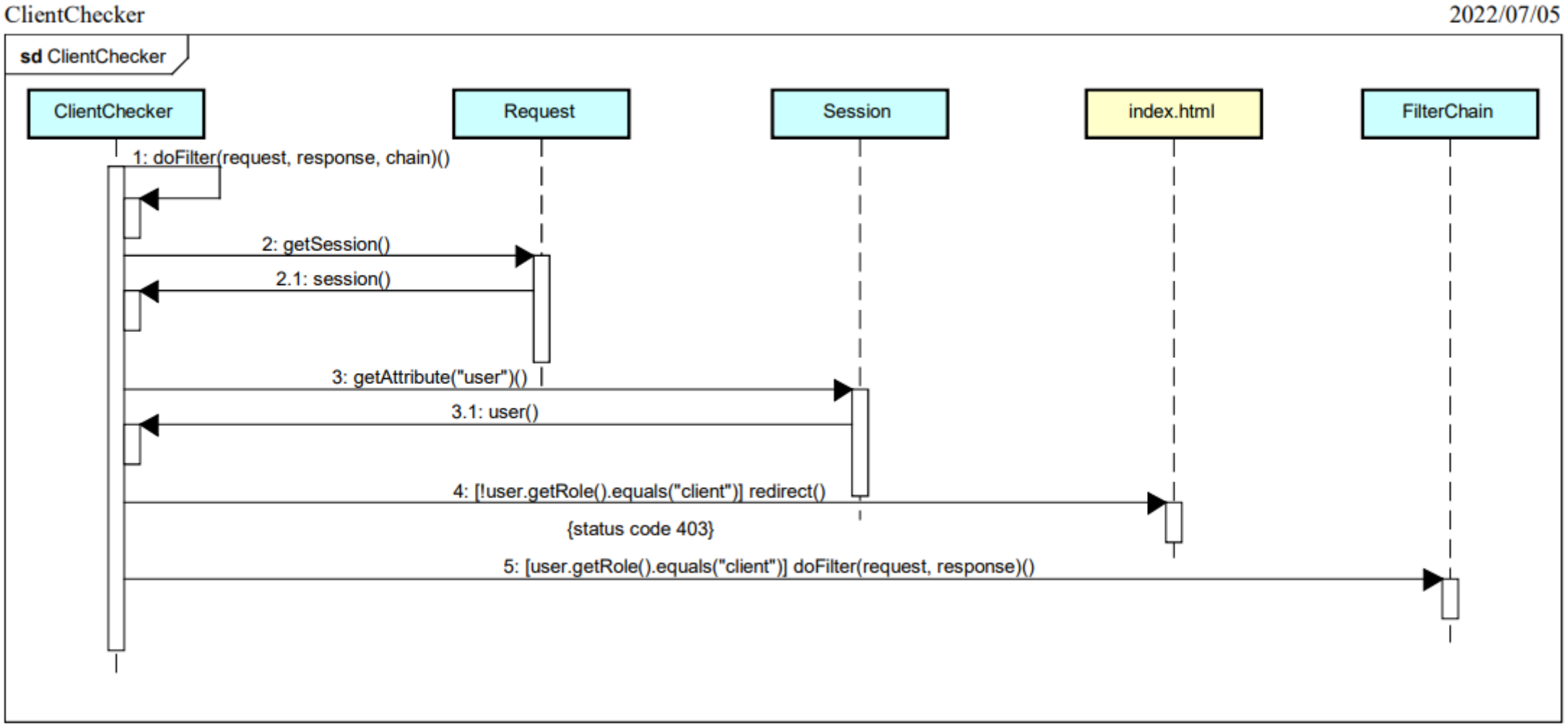
Event: Logout



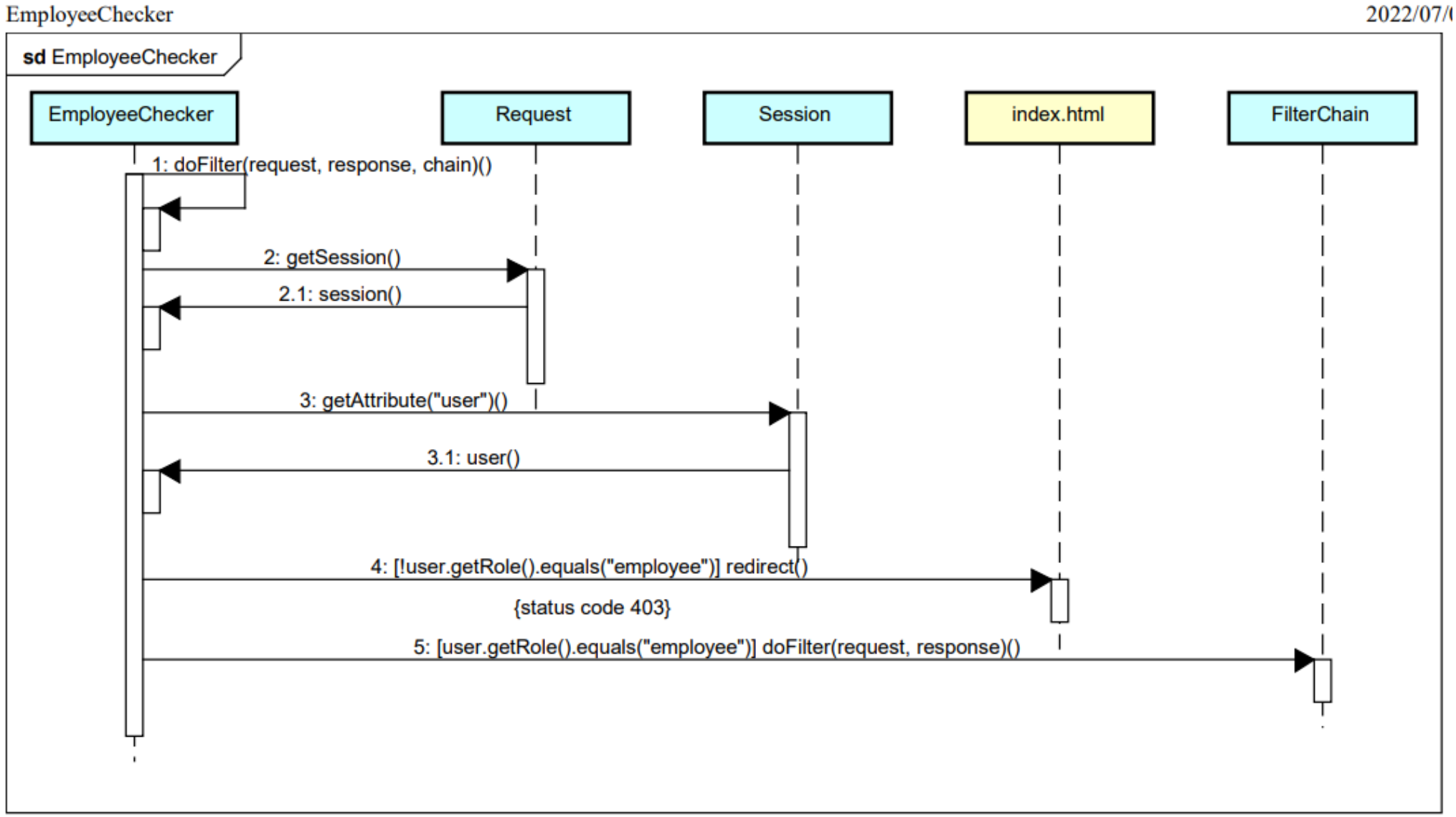
Login Checker



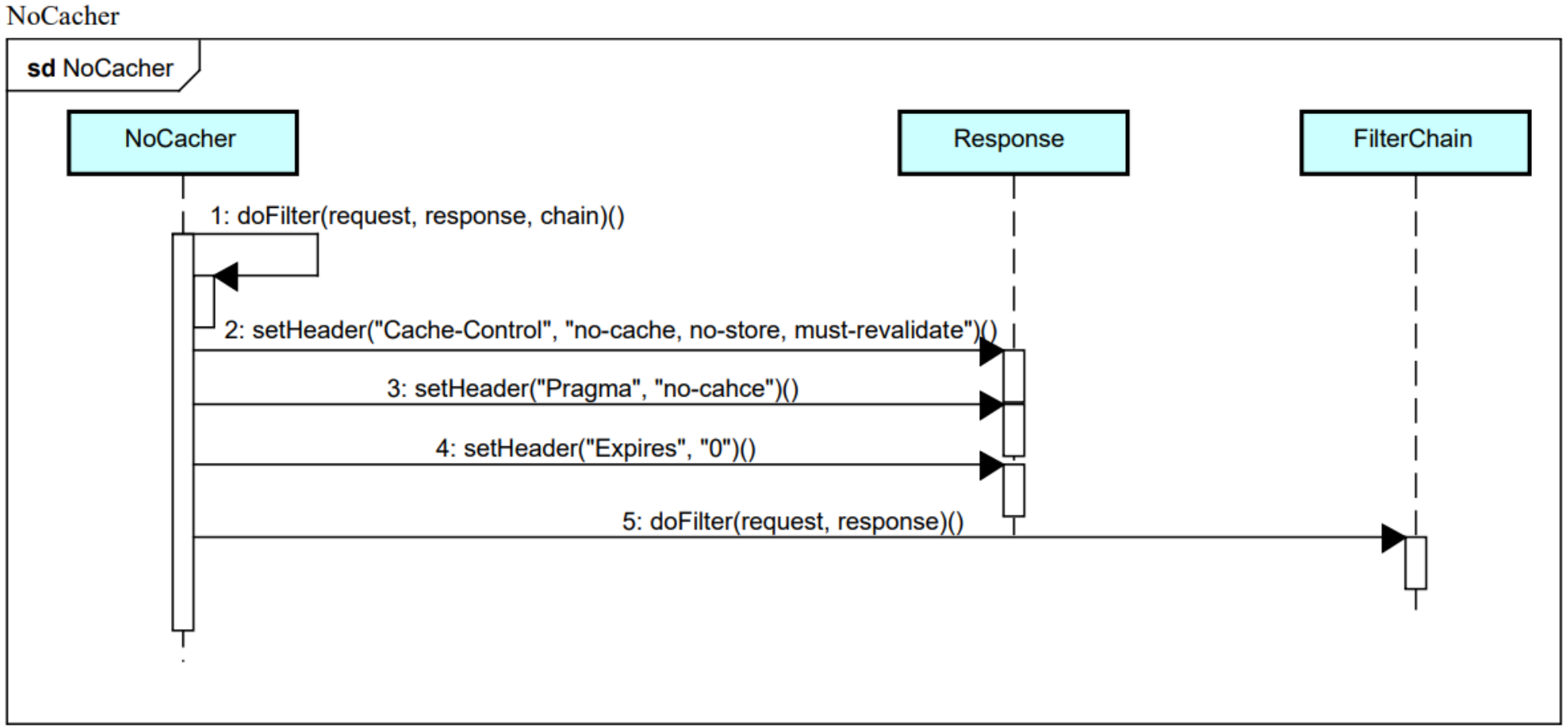
Client Checker



Employee Checker



No Cacher


NoCacher
sd NoCacher
NoCacher
Response
FilterChain
1: doFilter(request, response, chain)()
2: setHeader("Cache-Control", "no-cache, no-store, must-revalidate")()
3: setHeader("Pragma", "no-cahce")()
4: setHeader("Expires", "0")()
5: doFilter(request, response)()