TIW 2019-2020
Esercizio 2: gestione preventivi

Federico Mainetti Gambera
cod. persona: 10589654
matr: 872629

INDICE

PURE HTML

RICH INTERNET APPLICATION - RIA

PURE HTML

1 Data analysis

1.1 Testo dell'esercizio

Un'applicazione web consente la gestione di richieste di preventivi per prodotti personalizzati. 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. Mediante la form 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 fa 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.

1.2 Analisi dei dati

1.2.1 Entità

- Preventivo (quotation);
- prodotto (product);
- opzione (option);
- cliente (client);
- impiegato (employee).

1.2.2 Attributi

- Codice prodotto (ID);
- immagine prodotto (image);
- nome prodotto (name);
- codice opzione (ID);
- tipo opzione (type);
- nome opzione (name);
- prezzo (price);
- nome utente (username);
- email cliente (aggiunta per coerenza con la versione RIA);
- password.

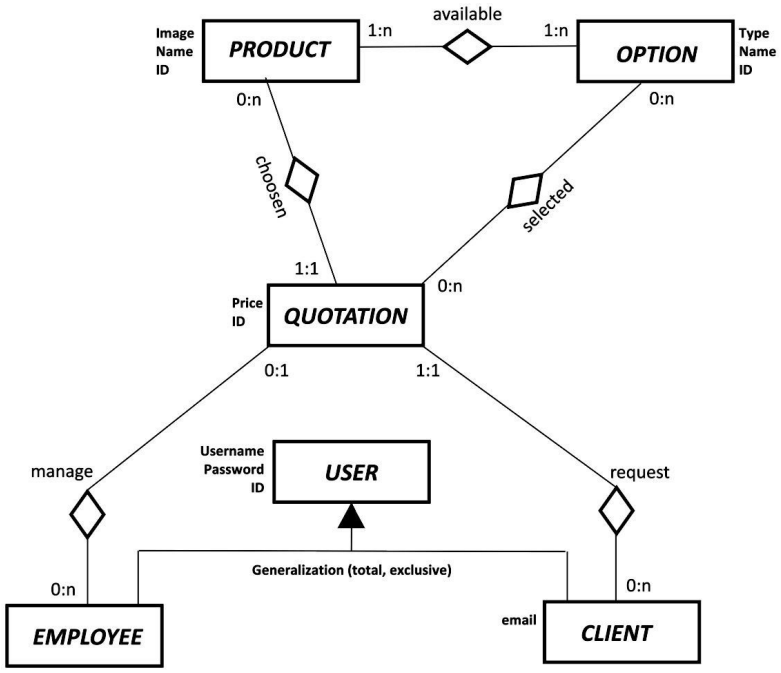
1.2.3 Relazioni

- Un preventivo è associato a un prodotto;
- un cliente richiede un preventivo;
- un impiegato gestisce un preventivo;
- un preventivo ha una o più opzioni;
- un prodotto ha delle opzioni disponibili.

2 Database

2.1 Database design

2.1.1 Entity-relationship schema



2.1.2 Schema relazionale

PRODUCT

ID	IMAGE	NAME

OPTION

ID	TYPE	NAME

QUOTATION

ID	PRICE	PRODUCT_ID	CLIENT_ID

EMPLOYEE

ID	USERNAME	PASSWORD

CLIENT

ID	USERNAME	EMAIL	PASSWORD

SELECTED

OPTION_ID	QUOTATION_ID

AVAILABLE

PRODUCT_ID	OPTION_ID

MANAGE

EMPLOYEE_ID	QUOTATION_ID

Gli attributi sottolineati rappresentano le chiavi primarie. Le chiavi esterne sono state omesse in quanto evidenti: sono tutti gli attributi della forma “NOMETABELLA_ID” e fanno riferimento all’attributo “ID” della tabella “NOMETABELLA”.

Le relazioni “CHOOSEN” (fra “PRODUCT” e “QUOTATION”) e “REQUEST” (fra “CLIENT” e “QUOTATION”) sono associazioni uno a molti con partecipazione obbligatoria, dunque sono state collassate nella tabella “QUOTATION” nei rispettivi attributi “PRODUCT_ID” e “CLIENT_ID”.

La generalizzazione totale ed esclusiva fra “USER” e “EMPLOYEE” e “CLIENT” è stata risolta accorpando gli attributi del genitore nelle entità figlie. Questa scelta agevola le ricerche nel database e non comporta nessuno svantaggio in termini di memoria occupata.

Per controllare che un preventivo sia stato già prezzato o meno, non è necessario controllare se esiste una tupla nella tabella “MANAGE”, ma è sufficiente controllare se l’attributo price è NULL o se ha un valore.

L’unico controllo che viene fatto sull’integrità del database è la corrispondenza delle chiavi esterne. L’integrità è comunque garantita in quanto opportuni controlli vengono sempre eseguiti a lato server.

2.2 Local Database schema

```
CREATE TABLE `db_quotation_management`.`product` (
    `ID` INT NOT NULL AUTO_INCREMENT,
    `image` LONGBLOB NOT NULL,
    `name` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`ID`));

CREATE TABLE `db_quotation_management`.`option` (
    `ID` INT NOT NULL AUTO_INCREMENT,
    `type` VARCHAR(45) NOT NULL,
    `name` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`ID`));

CREATE TABLE `db_quotation_management`.`quotation` (
    `ID` INT NOT NULL AUTO_INCREMENT,
    `price` DECIMAL(10,2) DEFAULT NULL,
    `productID` INT NOT NULL,
    `clientID` INT NOT NULL,
    PRIMARY KEY (`ID`),
    CONSTRAINT `productSelected`
        FOREIGN KEY (`productID`)
        REFERENCES `db_quotation_management`.`product` (`ID`)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT `requestingClient`
        FOREIGN KEY (`clientID`)
        REFERENCES `db_quotation_management`.`client` (`ID`)
        ON DELETE CASCADE
        ON UPDATE CASCADE);

CREATE TABLE `db_quotation_management`.`employee` (
      `ID` INT NOT NULL AUTO_INCREMENT,
      `username` VARCHAR(45) NOT NULL,
      `password` VARCHAR(45) NOT NULL,
      PRIMARY KEY (`ID`));

CREATE TABLE `db_quotation_management`.`client` (
      `ID` INT NOT NULL AUTO_INCREMENT,
      `username` VARCHAR(45) NOT NULL,
      `email` VARCHAR(120) NOT NULL,
      `password` VARCHAR(45) NOT NULL,
      PRIMARY KEY (`ID`));

CREATE TABLE `db_quotation_management`.`selectedoption` (
    `optionID` INT NOT NULL,
    `quotationID` INT NOT NULL,
```

```
    PRIMARY KEY (`optionID`, `quotationID`),
    CONSTRAINT `optionSelected`
        FOREIGN KEY (`optionID`)
        REFERENCES `db_quotation_management`.`option` (`ID`)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT `quotationLink`
        FOREIGN KEY (`quotationID`)
        REFERENCES `db_quotation_management`.`quotation` (`ID`)
        ON DELETE CASCADE
        ON UPDATE CASCADE);

CREATE TABLE `db_quotation_management`.`availableoption` (
    `productID` INT NOT NULL,
    `optionID` INT NOT NULL,
    PRIMARY KEY (`productID`, `optionID`),
    CONSTRAINT `productReference`
        FOREIGN KEY (`productID`)
        REFERENCES `db_quotation_management`.`product` (`ID`)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT `optionID`
        FOREIGN KEY (`optionID`)
        REFERENCES `db_quotation_management`.`option` (`ID`)
        ON DELETE CASCADE
        ON UPDATE CASCADE);

CREATE TABLE `db_quotation_management`.`management` (
    `employeeID` INT NOT NULL,
    `quotationID` INT NOT NULL,
    PRIMARY KEY (`employeeID`, `quotationID`),
    CONSTRAINT `employeeReference`
        FOREIGN KEY (`employeeID`)
        REFERENCES `db_quotation_management`.`employee` (`ID`)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT `quotationManaged`
        FOREIGN KEY (`quotationID`)
        REFERENCES `db_quotation_management`.`quotation` (`ID`)
        ON DELETE CASCADE
        ON UPDATE CASCADE);
```

3 Application requirements analysis

3.1 Testo dell'esercizio

Un'applicazione web consente la gestione di richieste di preventivi per prodotti personalizzati. 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. Mediante la form 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 fa 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.*

3.2 Analisi dei requisiti dell'applicazione

3.2.1 Pagine

Per quanto riguarda la pagina del cliente, siccome l'esercizio deve essere svolto senza programmazione a lato client, ho deciso di dividere la form per creare un preventivo in due pagine: SELEZIONE PRODOTTO e SELEZIONE OPZIONE.

3.2.2 Componenti delle viste

Nel testo dell'esercizio viene descritta una form* "dinamica" per la pagina del cliente che non è implementabile senza codice a lato client. Questa form è stata suddivisa in un bottone nella home, che, se cliccato, rimanda alla pagina SELEZIONE PRODOTTO, che contiene dei prodotti che quando selezionati rimandano alla pagina SELEZIONE OPZIONE, che contiene una form che permette di selezionare una o più opzioni e confermare la creazione del preventivo.

3.2.3 Eventi

UTENTE NON AUTENTICATO:

- Accedere all'applicazione;
- Log In;

CLIENTE:

- Accedere alla HOME PAGE del cliente (GoToClientHomePage).
- Cliccare sul bottone per la creazione di un nuovo preventivo (GoToProductSelectionPage);
- Selezionare un prodotto (GoToOptionSelectionPage);
- Selezionare un'opzione e confermare la creazione del prodotto (CreateQuotation);

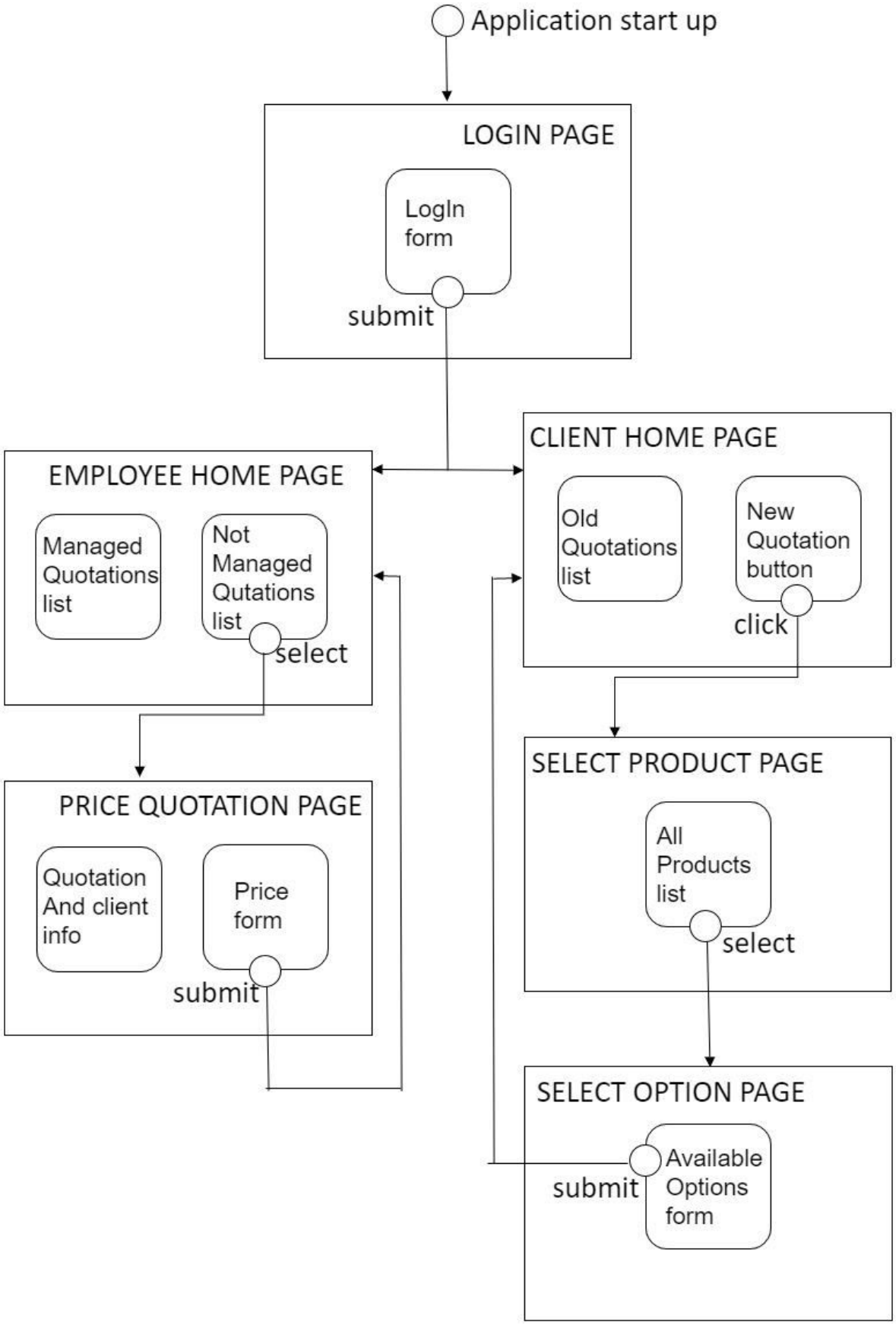
IMPIEGATO:

- Accedere alla HOME PAGE dell'impiegato (GoToEmployeeHomePage).
- Selezionare un preventivo non ancora prezzato (GoToPriceQuotationPage);
- Inserire il prezzo e inviare (PriceQuotation).

3.2.4 Azioni

Esposte dettagliatamente nei diagrammi di flusso degli eventi (vedi più avanti, paragrafo 6).

4 Design dell'applicazione



5 Componenti

5.1 Oggetti di modello - Beans

```
public class Option {
          private int ID;
          private String type;
          private String name;
          //... getter and setters
}
public class Product {
          private int ID;
          private String name;
          private byte[] image;
          private ArrayList<Option> options;
        //… getter and setters
}
public class Quotation {
          private int ID;
          private Money price;
          private String employeeUsername;
          private String clientUsername;
          private Product product;
          //... getter and setters
}
public class User {
          private String username;
          private int ID;
          private boolean isClient;
        //... getter and setters
}
public class Money {
          private int wholePart;
          private int decimalPart;
          //... getter and setters
}
```

5.2 Data Access Objects - DAO

```
public class ClientDAO {
          public User findByID(int clientID){}
          public User checkCredentials(String username, String password)
}

public class EmployeeDAO {
          public User checkCredentials(String username, String password)
}

public class ProductDAO {
          public ArrayList<Product> findAllWithOptions()
          public ArrayList<Product> findAllWithoutOptions()
          public ArrayList<Option> findAvailableOptionsForProduct(int productID)
          public byte[] findProductImage(int productID)
}

public class QuotationDAO {
          public ArrayList<Quotation> findAllByClientID(int clientID)
          public ArrayList<Quotation> findAllByEmplyeeID(int emplyeeID)
          public void createQuotation(int productID, int clientID,
                                      ArrayList<Integer> optionsID )
          public void priceQuotation(int quotationID, Money price, int employeeID)
          public ArrayList<Quotation> findAllNotManaged()
          public Quotation findByID(int quotationID)
          public ArrayList<Option> getOptionsSelected(int quotationID)
}
```

5.3 Servlets - Controllers

```
public class CheckLogin {...}

public class CreateQuotation {...}

public class GoToClientHomePage {...}

public class GoToEmployeeHomePage {...}

public class GoToOptionSelectionPage {...}

public class GoToPriceQuotationPage {...}

public class GoToProductSelectionPage {...}

public class ImageGetter {...} // es. <img src=”./ImageGetter?ID=1”></img>

public class PriceQuotation {...}
```

5.4 Views - Templates

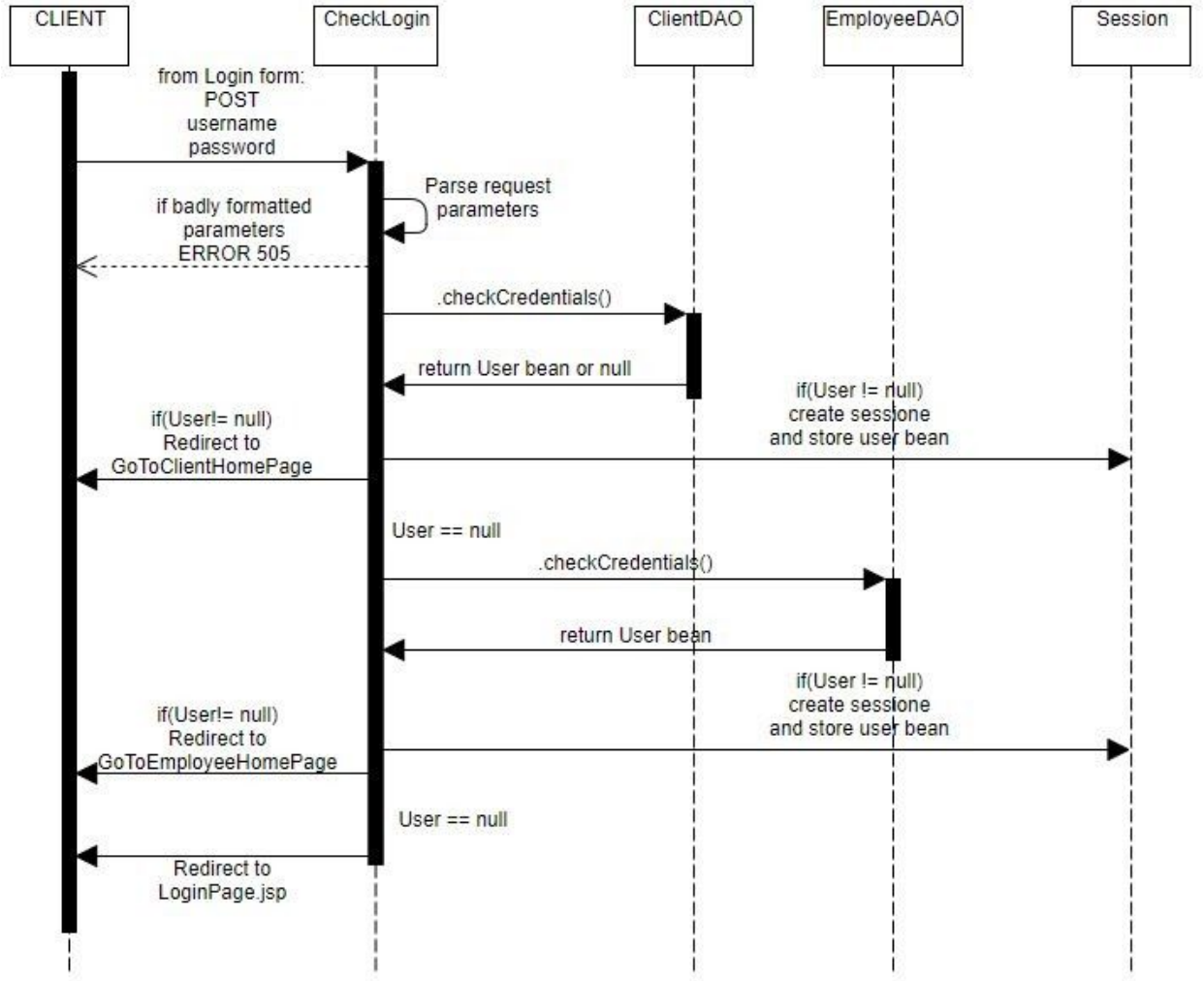
```
Loginpage.jsp

ClientHome.jsp

EmployeeHome.jsp

ProductSelection.jsp

OptionSelection.jsp

PriceQuotation.jsp
```

6 Diagrammi di flusso degli eventi

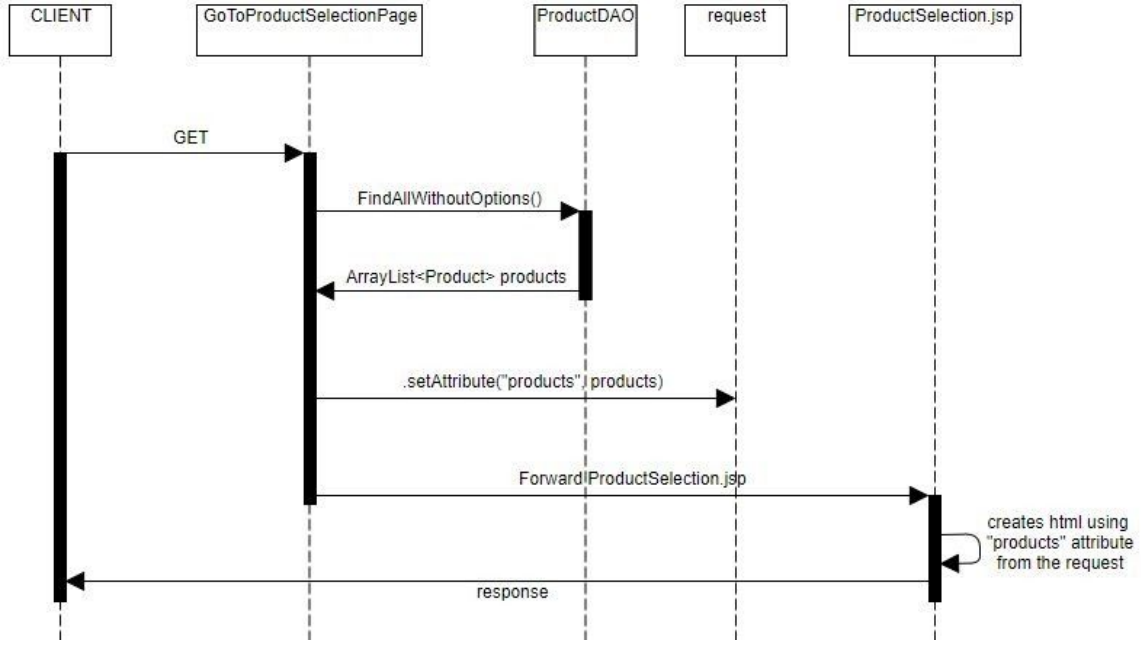
6.1 Application start up

[omesso perché banale: caricamento della pagina ***statica*** "LogInPage"]

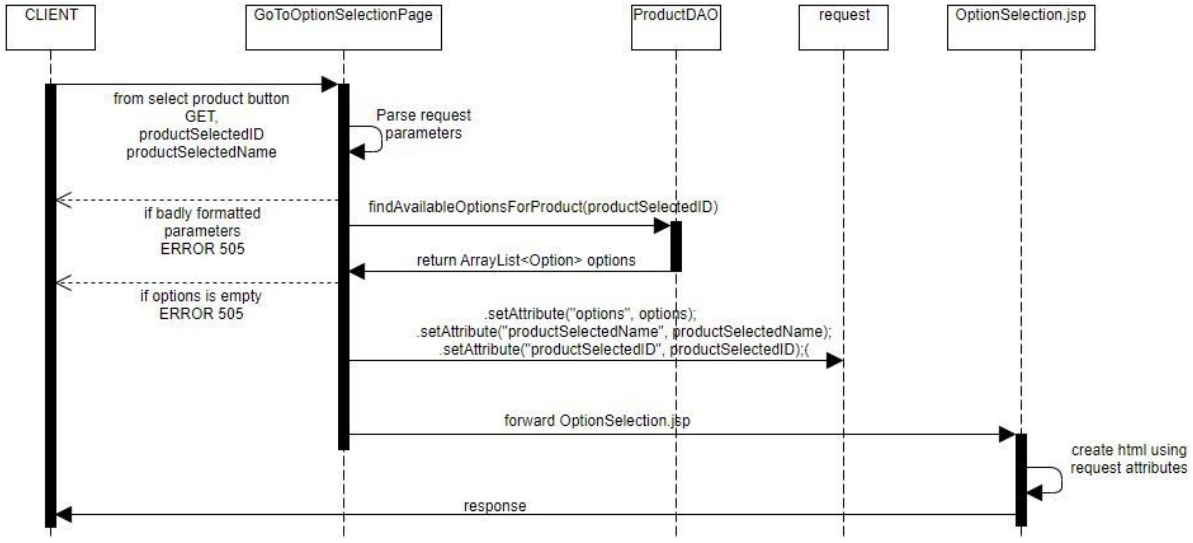
6.2 LogIn



6.3 GoToProductSelectionPage



6.4 GoToOptionSelectionPage



6.5 GoToClientHomePage

[omesso perché estremamente simile a “GoToProductSelectionpage” o “GoToOptionSelectionPage”]

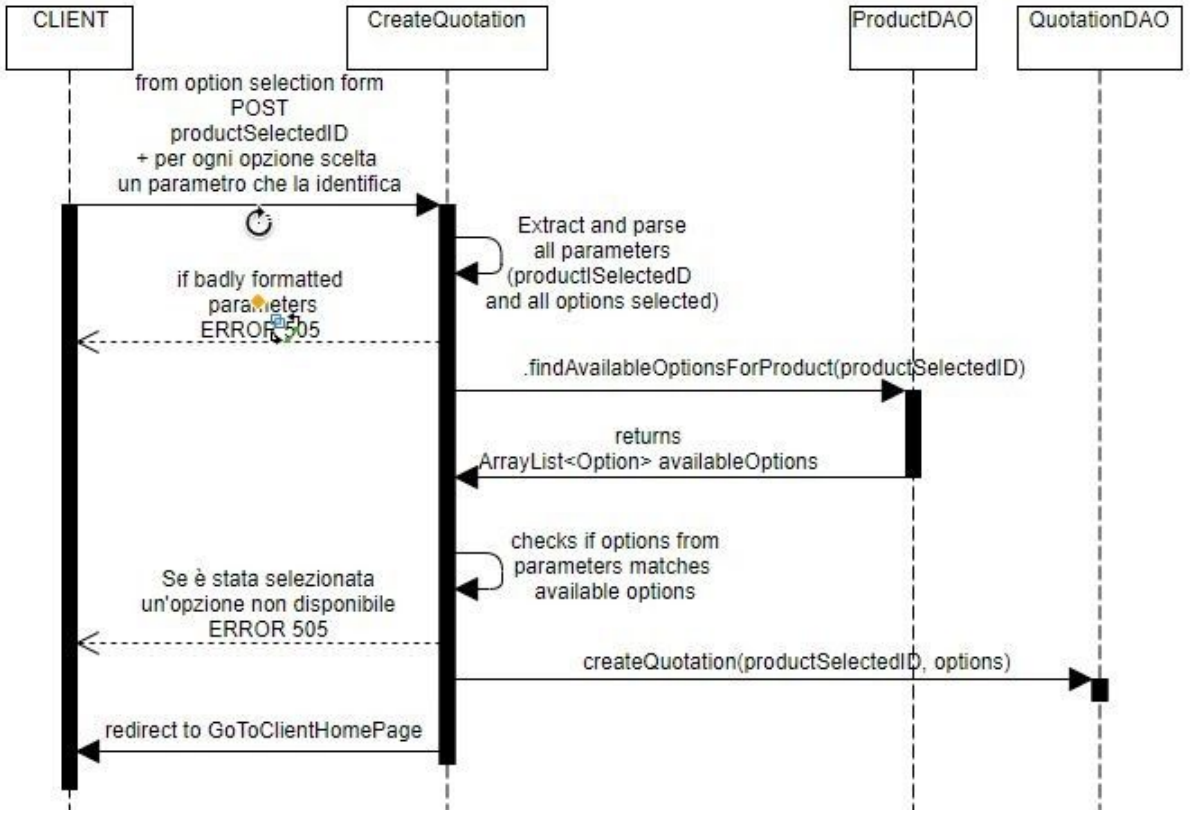
6.6 GoToEmployeeHomePage

[omesso perché estremamente simile a “GoToProductSelectionpage” o “GoToOptionSelectionPage”]

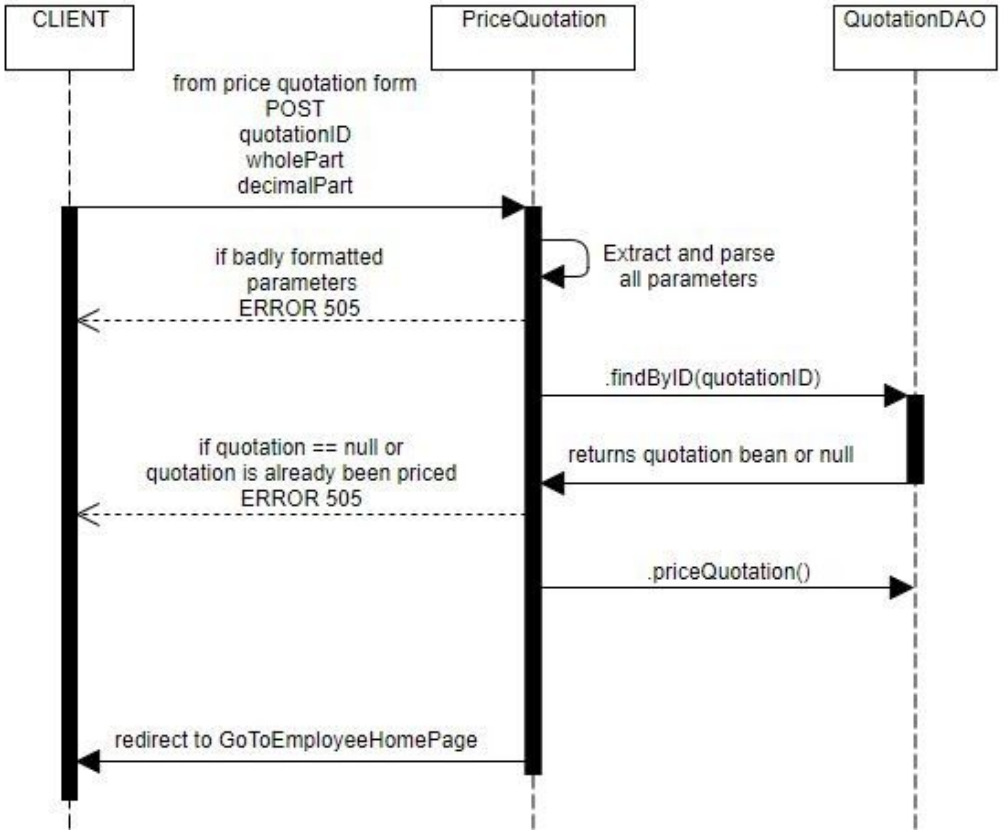
6.7 GoToPriceQuotationPage

[omesso perché estremamente simile a “GoToProductSelectionpage” o “GoToOptionSelectionPage”]

6.8 CreateQuotation



6.9 PriceQuotation



RICH INTERNET APPLICATION - RIA

1 Data analysis

1.1 Testo dell'esercizio

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

- *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", anche a lato client. La registrazione controlla l'unicità dello username.*
- *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.*

1.2 Analisi dei dati

[sezione omessa: corrisponde a quello della versione PURE HTML]

1.2.1 Entità

1.2.2 Attributi

- email cliente;

1.2.3 Relazioni

2 Database

[sezione omessa: corrisponde a quello della versione PURE HTML]

2.1 Database design

2.1.1 Entity-relationship schema

2.1.2 Schema relazionale

2.2 Local Database schema

3 Application requirements analysis

3.1 Testo dell'esercizio

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

- *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", anche a lato client. La registrazione controlla l'unicità dello username.*
- *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.*

3.2 Analisi dei requisiti dell'applicazione

Sono state evidenziate solo le differenze rispetto alla versione PURE HTML.

3.2.1 Pagine

L'applicazione presenta solo tre pagine:

- LogIn.html
- ClientHome.html
- Employee.html

3.2.2 Componenti delle viste

Rispetto alla versione PURE HTML, le uniche differenze sono:

- form per registrarsi;
- In RIA è ora possibile creare una form dinamica per la creazione di nuovi preventivi;
- messaggio di allerta per eventuali errori a lato server;
- bottone per fare il Log Out (non richiesto);

3.2.3 Eventi

Rispetto alla versione PURE HTML, vengono aggiunti i seguenti eventi:
UTENTE NON AUTENTICATO:

- Registrazione nuovo account;

CLIENTE:

- Eventi di selezione prodotto, selezione opzioni e invio del preventivo.
- Log Out;

IMPIEGATO:

- Log Out

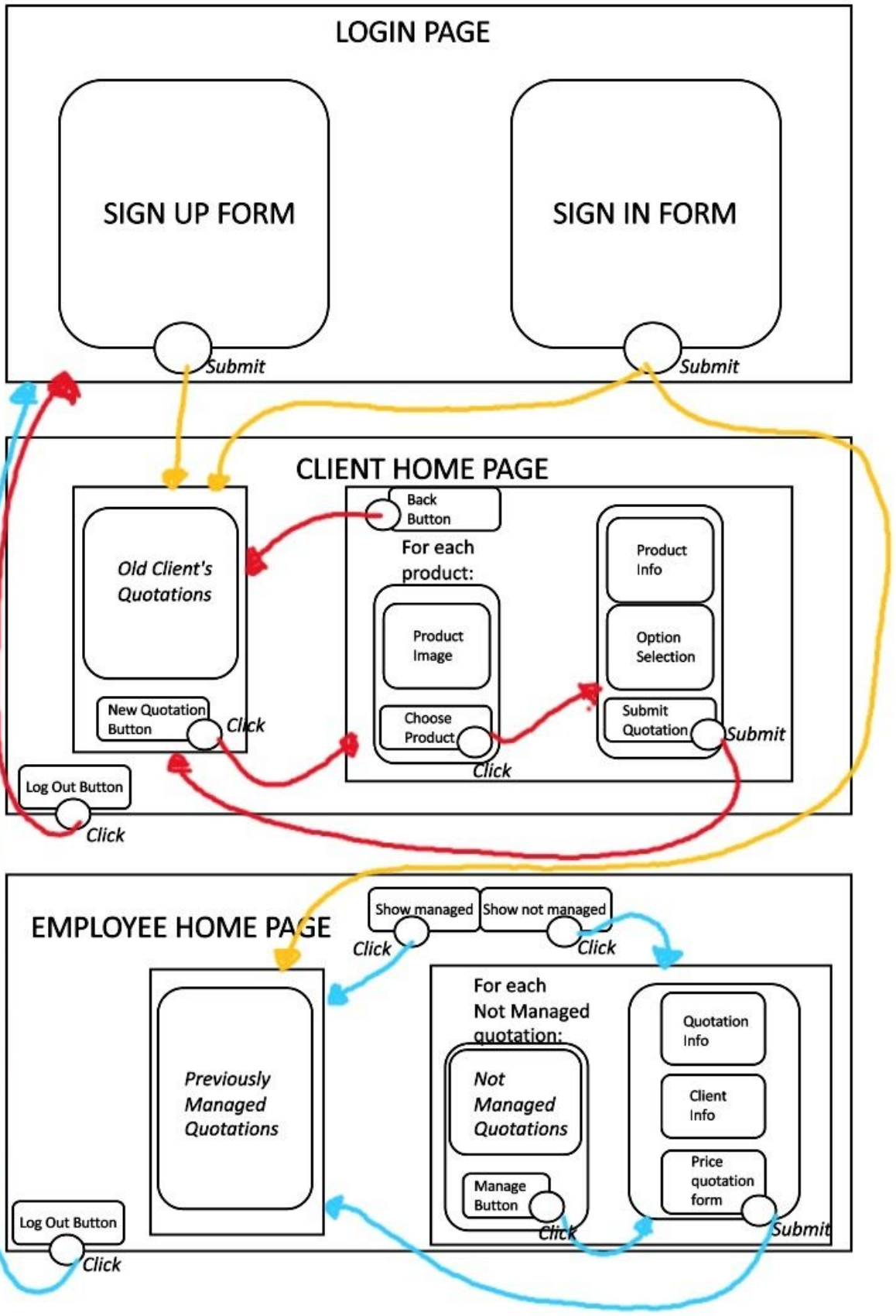
ERRORE:

- Segnalazione di un errore a lato server;

3.2.4 Azioni

Esposte dettagliatamente nei diagrammi di flusso degli eventi (vedi più avanti, paragrafo 6).

4 Design dell’applicazione



5 Componenti

5.1 Server Side

5.1.1 Oggetti di modello - Beans

```
public class Option {
         //... Same as PURE HTML
}
public class Product {
         //... Same as PURE HTML
}
public class Quotation {
         //... Same as PURE HTML
}
public class User {
         //... Same as PURE HTML
}
public class Money {
         //... Same as PURE HTML
}
```

5.1.2 Data Access Objects - DAO

```
public class ClientDAO {
         //... Same as PURE HTML but with:
         public User findByUsername(String username)
         public User createClient(String username, String email, String password)
}

public class EmployeeDAO {
         //... Same as PURE HTML
}

public class ProductDAO {
         //... Same as PURE HTML
}

public class QuotationDAO {
         //... Same as PURE HTML
}
```

5.1.3 Servlets - Controllers

```
public class CheckLogin {...}

public class CreateQuotation {...}

public class getAllProducts {...}

public class getClientInfo {...}

public class getEmployeeInfo {...}

public class getManagedQuotations {...}

public class getNotManagedQuotations {...}

public class getQuotations {...}

public class ImageGetter {...}

public class LogOut {...}

public class PriceQuotations{...}

public class SignUp {...}
```

5.1.4 Views - Templates

```
LogIn.html

ClientHome.html

EmployeeHome.html
```

5.2 Client Side (Javascript)

5.2.1 LogInPage

Il file /WebContent/JS/Login/main.js contiene tutte le funzionalità necessarie al corretto funzionamento della form per il sign in e della form per il sign up:

- **CheckSignUpValidity()**: funzione utilizzata per controllare la corretta compilazione dei campi della form per il sign up;
- **CheckSignInValidity()**: funzione utilizzata per controllare la corretta compilazione dei campi della form per il sign in.
- **Gestione submit della form di sign up**: chiama CheckSignUpValidity(), se la form non è ben compilata segnala l'errore, altrimenti invia i dati al server. Una volta che il server ha elaborato i dati ritorna al client una risposta contenente o un messaggio di errore o la conferma dell'avvenuta creazione del nuovo account. In caso di presenza di un errore, esso viene mostrato al cliente, in caso di successo nella creazione del nuovo account, viene caricata la pagina ClientHome.html.
- **Gestione submit della form di sign in**: chiama CheckSignInValidity(), se la form non è ben compilata segnala l'errore, altrimenti invia i dati al server. Una volta che il server ha elaborato i dati ritorna al client una risposta contenente o un messaggio di errore o la conferma del corretto log in. In caso di presenza di un errore, esso viene mostrato al cliente, in caso di corretto log in viene caricata la pagina ClientHome.html o EmployeeHome.html in base ai dati ricevuti dal server.

5.2.2 ClientHomePage

I file che contengono le funzionalità necessarie al corretto funzionamento della pagina del cliente sono contenuti all'interno della directory /WebContent/JS/ClientHome/ e sono:

- **main.js;**
- **model.js**
- **orchestrator.js**
- **serverGate.js**
- **view.js**

L'oggetto **Model** rappresenta un contenitore di dati con metodi getter e setter, inoltre presenta funzioni che permettono di fare il parsing di un generico oggetto bean ricevuto dal server (JSON) e immagazzinarlo al suo interno.

L'oggetto **ServerGate** è utile per astrarre il più possibile l'interazione col server. Esso contiene metodi di utilità per comunicare e ricevere informazioni dal server:

```
function ServerGate(view){
    this.requestClient = function(cback){...}
    this.requestQuotations = function(cback){...}
    this.requestAllProducts = function(cback){...}
    this.submitQuotation = function(){...}
    this.logOut = function(){...}
    this.makeCall = function(method, url, cback) {...}
}
```

L'oggetto **View** contiene tutti le funzioni per generare contenuto html dinamico e registrare eventi a partire dalle informazioni contenute nel Model.

```
function View(model){
    this.updateQuotations = function(){...}
    this.updateNewQuotationTab = function(){...}
    this.createProductsPage = function(){...}
    this.selectOption = function (spanElement, optionID){...}
    this.showError = function(){...}
}
```

Infine, l'oggetto **Orchestrator** contiene tutte le funzioni che coordinano gli oggetti appena descritti. Tipicamente queste funzioni vengono chiamate a seguito di un evento generato dall'utente (per esempio il click su un bottone o la compilazione di una form).

```
function Orchestrator(model, serverGate, view){
    this.init = function(){...}
    this.goToNewQuotation = function(){...}
    this.goToHome = function(){...}
}
```

Nel file **main.js** vengono inizializzati gli oggetti degli altri file ed eseguita la funzione orchestrator.init() che, in poche parole, avvia l'applicazione:

```
    var model = new Model();
    var view = new View(model);
    var serverGate = new ServerGate(view);
    var orchestrator = new Orchestrator(model, serverGate, view);

    orchestrator.init();
```

5.2.3 EmployeeHomePage

La struttura della EmployeeHomePage è del tutto simile a quello della ClientHomePage, cambiano solo i metodi:

```
function ServerGate(view){
    this.requestEmployee = function(cback){
    this.requestManagedQuotations = function(cback){
    this.requestNotManagedQuotations = function(cback){
    this.priceQuotation = function(cback){
    this.logOut = function(){...}
    this.makeCall = function(method, url, cback) {...}
}
function View(model){
    this.updateManagedQuotations = function(){...}
    this.updateNotManagedQuotations = function(){...}
    this.registerBannerListeners = function(){...}
    this.updateError = function(response){...}
    this.showError = function(){...}
```

```
    this.closeTab = function(){...}
    this.openTab = function(){...}
}
function Orchestrator(model, serverGate, view){
    this.init = function(){...}
    this.goToManagedQuotations = function () {...}
    this.goToNotManagedQuotations = function(){...}
}
```

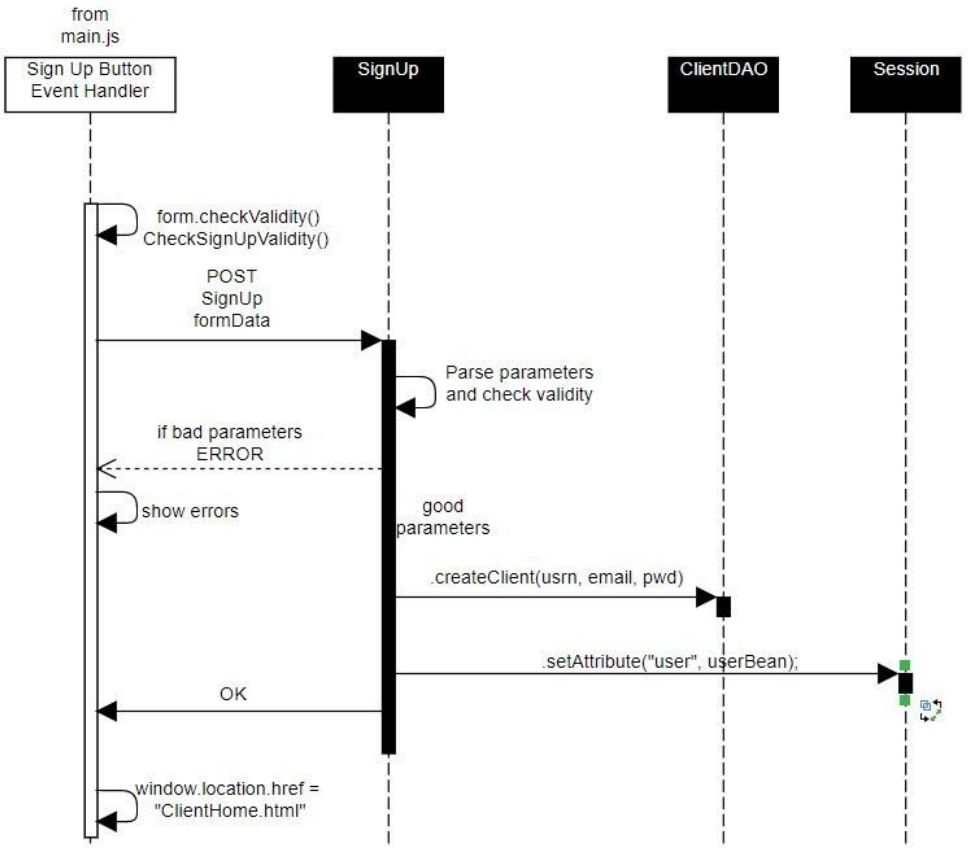
6 Diagrammi di flusso degli eventi

Tutti gli eventi sono:

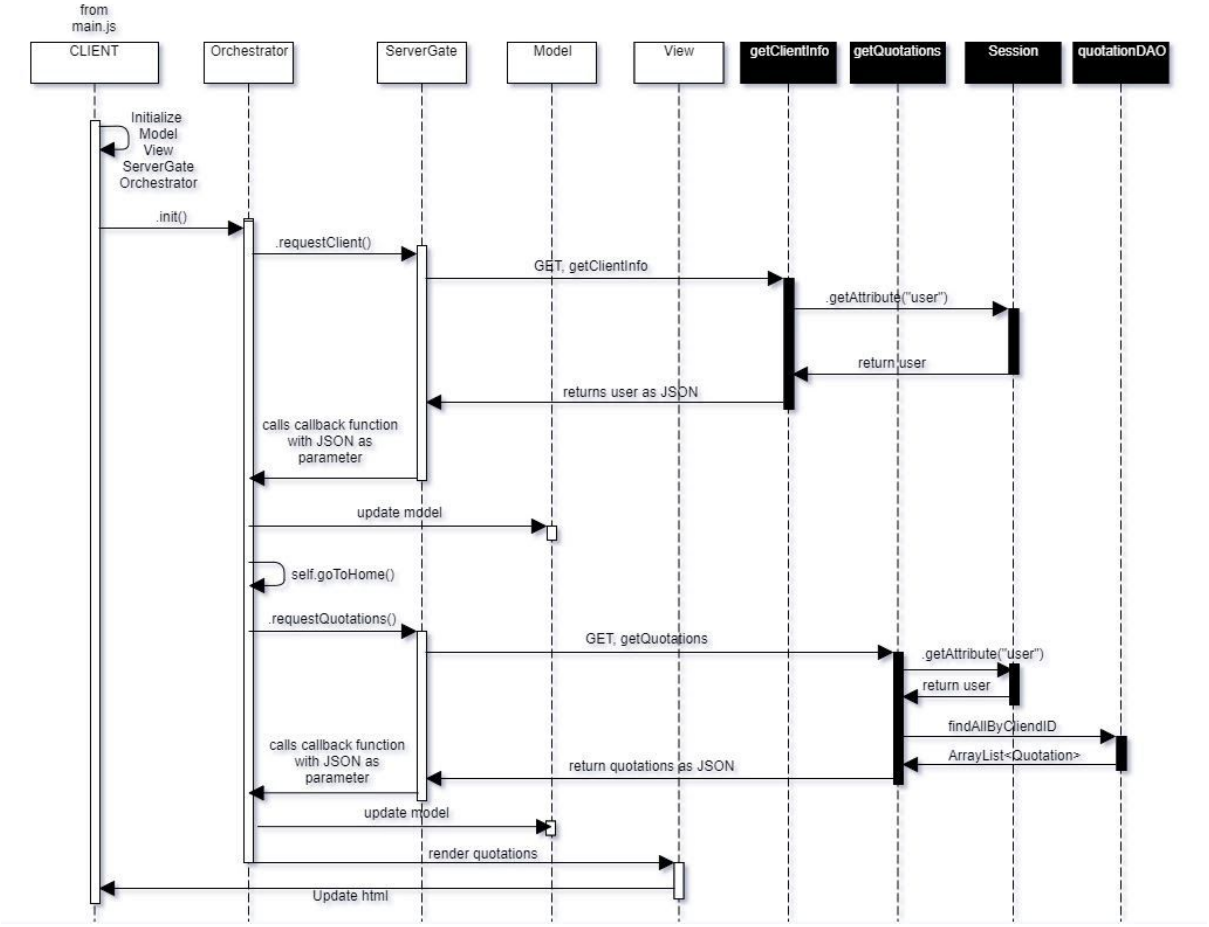
- **LogIn.html first load** (banale: vengono solo registrati gli eventi sui bottoni di submit della form);
- **Submit sign up form** (vedi sotto il diagramma di flusso);
- **Submit sign in form** (banale, simile a PURE HTML);
- **ClientHome.html first load** (vedi sotto il diagramma di flusso);
- **Client log out** (banale: a client side si carica LogIn.html, a server side si cancella la sessione);
- **Client go to home (incluso in "ClientHome.html first load");**
- **Client go to create new quotation** (struttura simile a "Client go to home");
- **Client choose product** (banale: solo client side, viene salvato nel model il prodotto selezionato e aggioranta la view per mostrare le opzioni disponibili);
- **Client choose option(s)** (banale: solo client side, viene salvato nel model l'opzione selezionata);
- **Client submit new quotation** (vedi sotto il diagramma di flusso);
- **EmployeeHome.html first load** (simile a ClientHome.html first load");
- **Employee log out** (banale: a client side si carica LogIn.html, a server side si cancella la sessione);
- **Employee go to previously managed quotations** (struttura simile a "Client go to home");
- **Employee go to all not managed quotations** (struttura simile a "Client go to home");
- **Employee select a not managed quotation** (struttura simile a "Client go to home");
- **Employee submit price** (vedi sotto diagramma di flusso);

Di seguito vengono riportati i diagrammi di flusso degli eventi più importanti per l'applicazione. Gli eventi esclusi o sono eventi banali (come per esempio quelli esclusivamente grafici o che non interagiscono col server..) o estremamente simili a uno di quelli esposti.

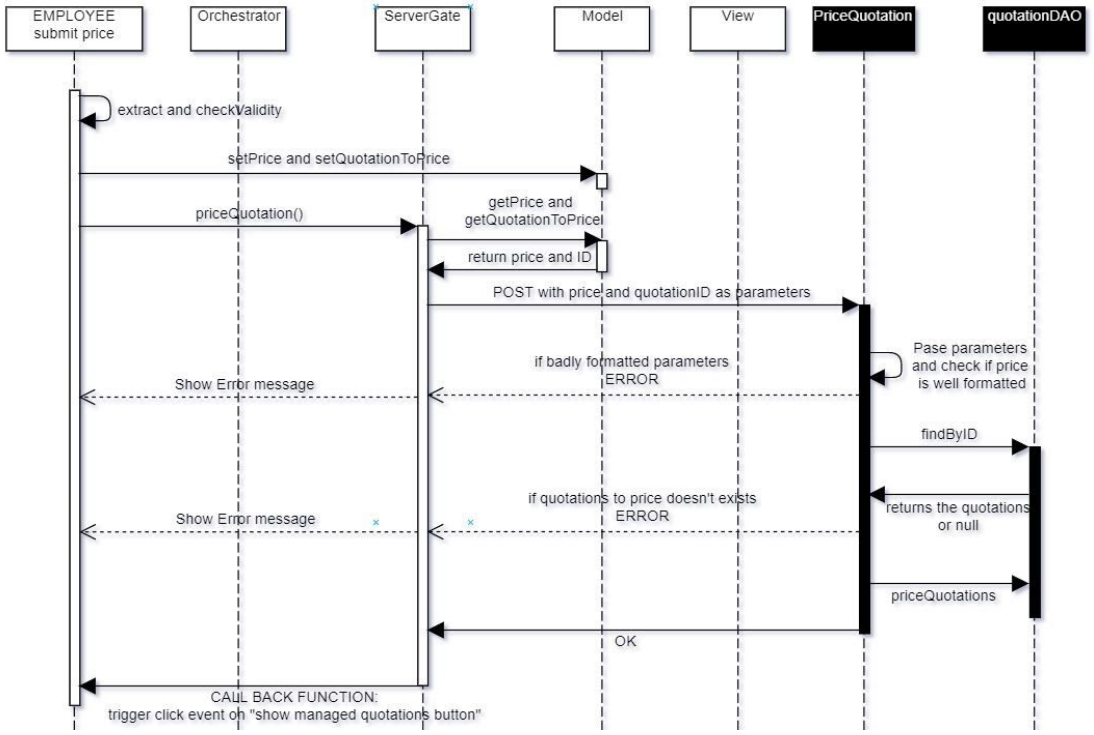
6.1 Submit sign up form



6.2 ClientHome.html first load



6.3 Employee submit price



6.4 Client submit new quotation