UNIVERSITÀ
DI PARMA

Corso di Ingegneria del Software

DOCUMENTAZIONE PROGETTO
Sailing Club

Martina Gualtieri
Matr. 308783

Cristian Cervellera
Matr. 305459

ANNO ACCADEMICO 2021/2022

Contents

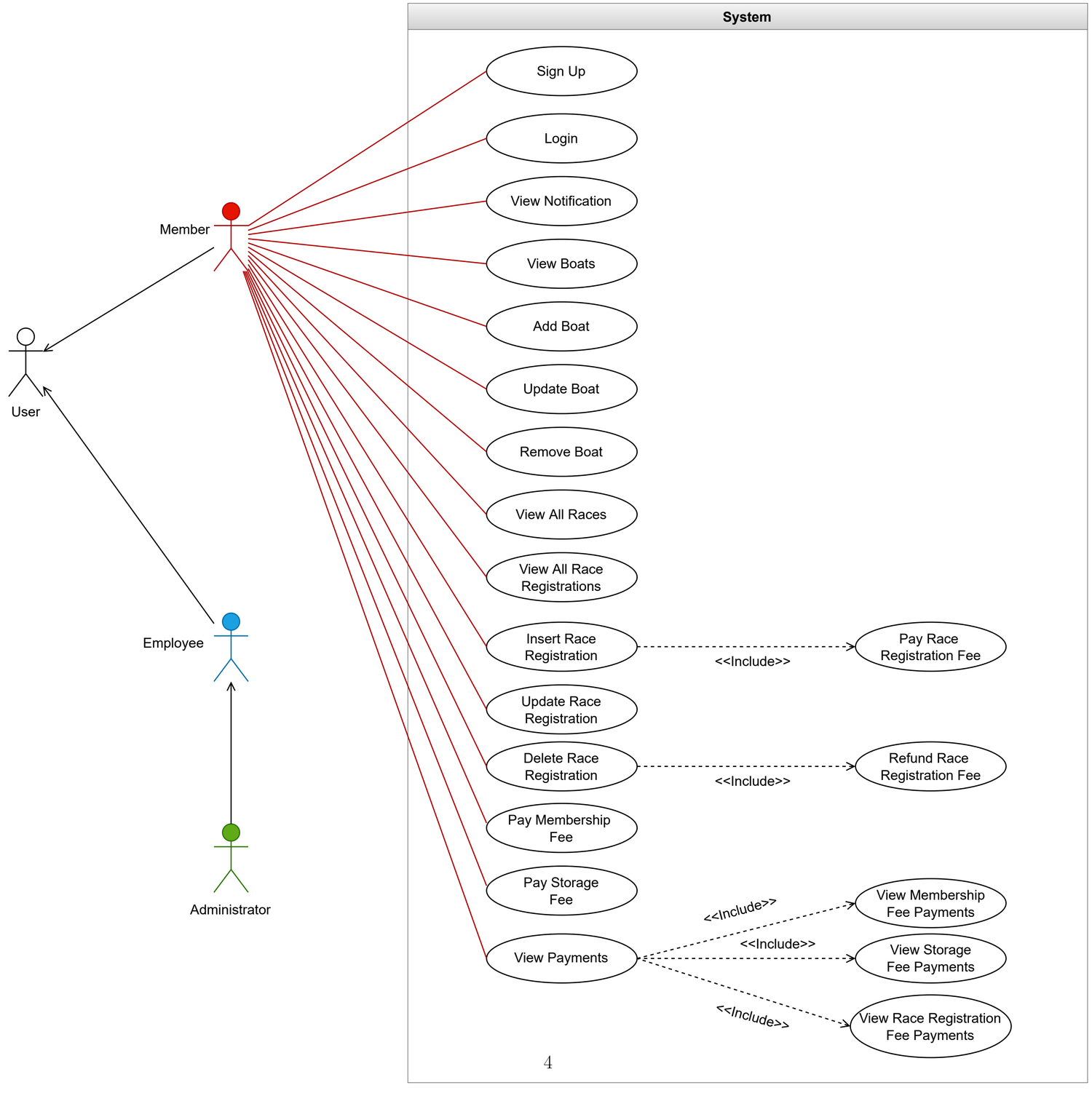
1 Requisiti funzionali

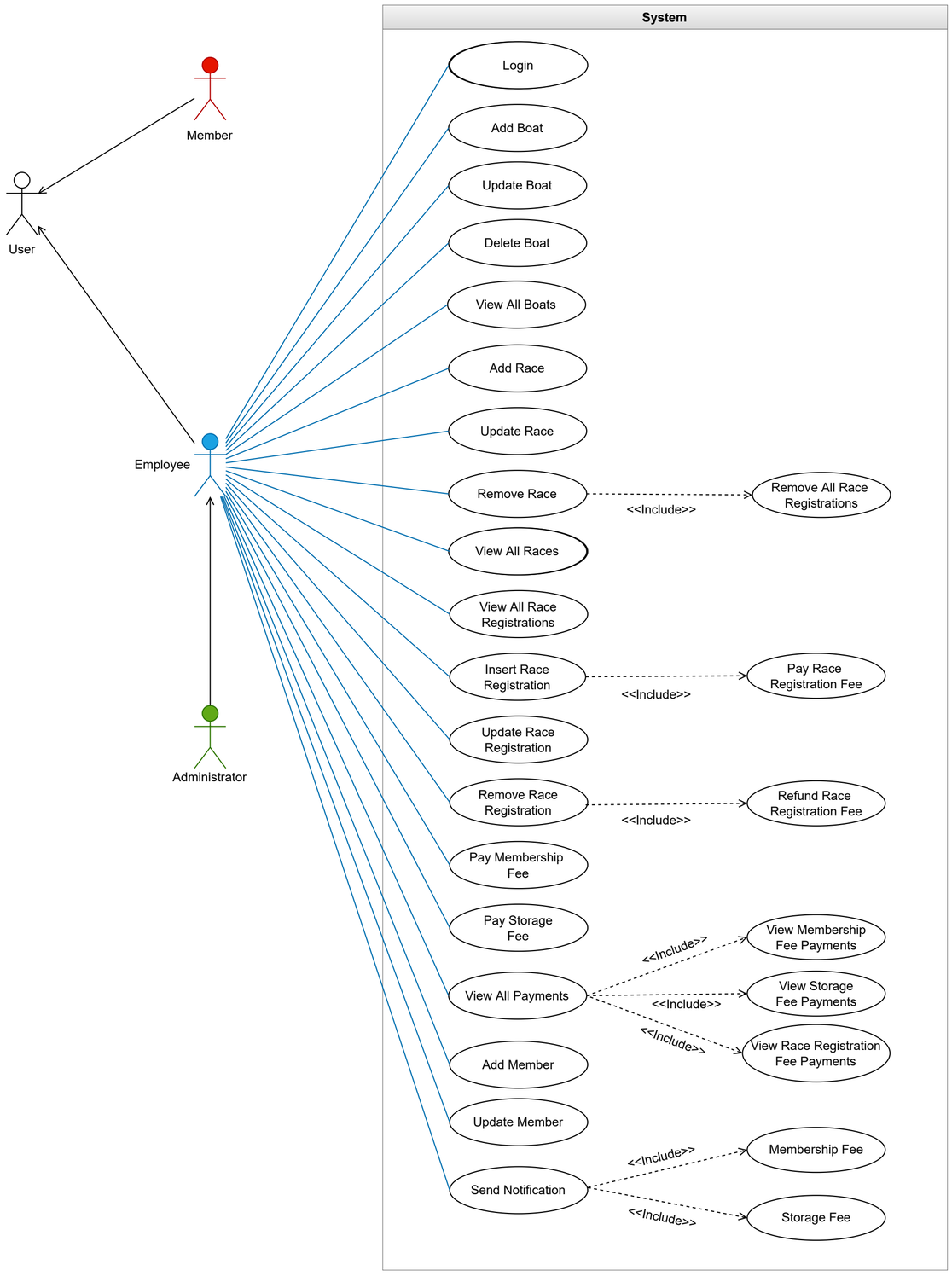
I requisiti funzionali sono definizioni di funzionalità e servizi che il sistema "Sailing Club" deve fornire. In particolare, il sistema deve garantire:

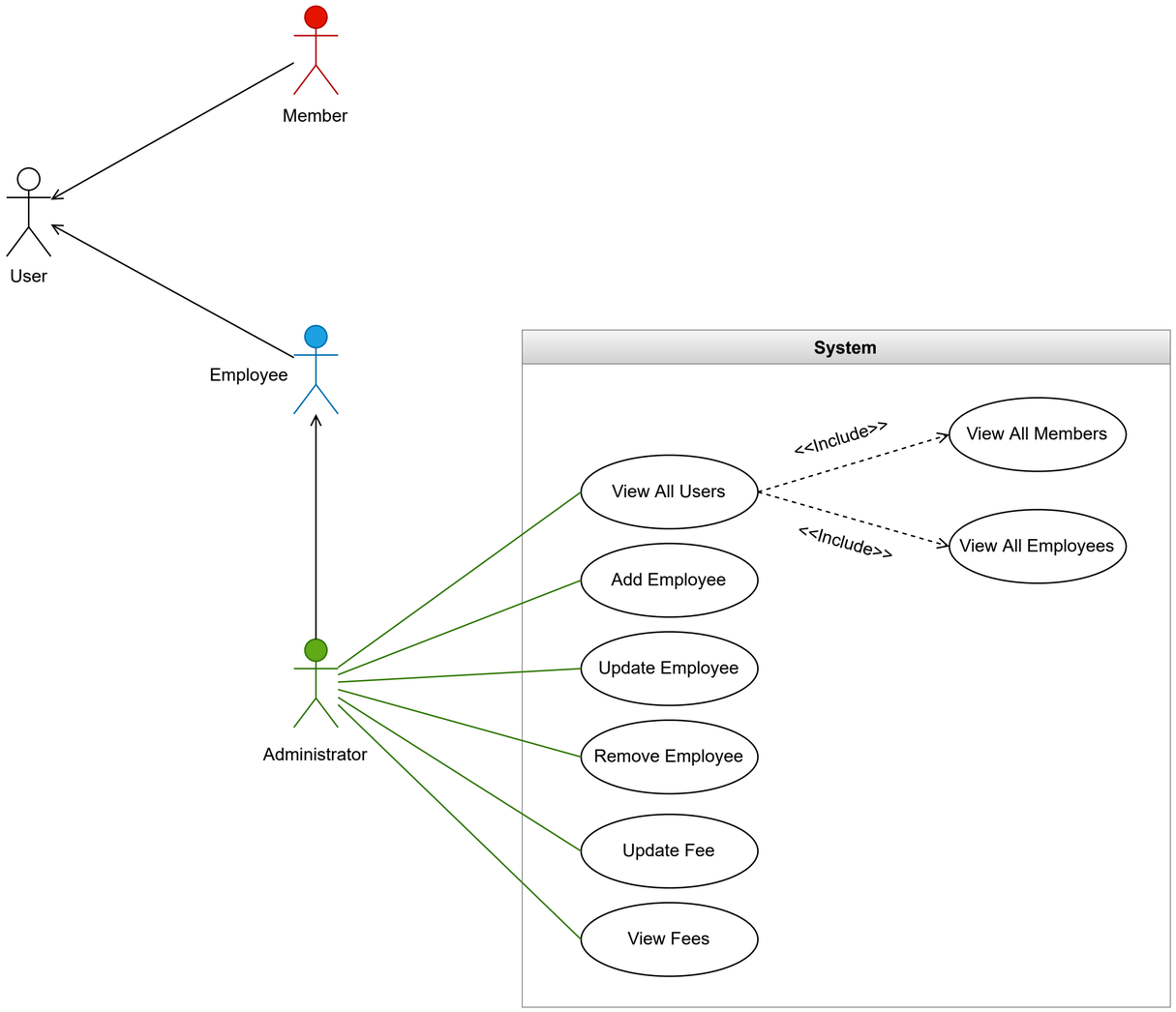
1. A soci ed impiegati di autenticarsi tramite email e password
2. Ai soci di registrarsi inserendo codice fiscale, nome, cognome, indirizzo, email e password
3. A soci ed impiegati di aggiungere una nuova imbarcazione
   3.1 Nel caso in cui è l'impiegato ad aggiungere una nuova imbarcazione, esso deve specificare anche il socio a cui l'imbarcazione appartiene
4. A soci ed impiegati di modificare un'imbarcazione
5. A soci ed impiegati di rimuovere un'imbarcazione
6. Ai soci di visualizzare tutte le imbarcazioni di loro proprietà e agli impiegati di visualizzare tutte le imbarcazioni
7. Agli impiegati di aggiungere una nuova gara
8. Agli impiegati di modificare una gara
9. Agli impiegati di rimuovere una gara
10. A soci ed impiegati di visualizzare tutte le gare
11. A soci ed impiegati di iscrivere un'imbarcazione ad una gara e di pagare la quota di iscrizione
    11.1 Nel caso in cui è l'impiegato ad iscrivere un'imbarcazione, esso deve selezionare il socio, un'imbarcazione appartenente a tale socio ed il metodo di pagamento con il quale il socio vuole pagare la quota di iscrizione alla gara
12. A soci ed impiegati di modificare l'imbarcazione iscritta ad una gara
13. A soci ed impiegati di rimuovere l'iscrizione di un'imbarcazione ad una gara
14. A soci ed impiegati di visualizzare tutte le iscrizioni ad una gara
15. A soci ed impiegati di pagare la quota di associazione e di rimessaggio per un'imbarcazione
    15.1 Nel caso in cui è l'impiegato ad effettuare il pagamento, esso deve specificare il socio e, per il pagamento della quota di rimessaggio, l'imbarcazione
16. Ai soci di visualizzare i propri pagamenti delle quote di associazione, rimessaggio e di iscrizione alle gare
17. Agli impiegati di visualizzare tutti i pagamenti effettuati da tutti i soci
18. Agli impiegati di notificare il pagamento della quota di associazione e di rimessaggio ad un socio
19. Ai soci di visualizzare le notifiche di pagamento
20. Agli impiegati di aggiungere un nuovo socio
21. Agli impiegati di modificare i dati di un socio
22. Agli impiegati di visualizzare tutti i soci

23. Agli amministratori di aggiungere un nuovo impiegato
24. Agli amministratori di modificare i dati di un impiegato
25. Agli amministratori di eliminare un impiegato
26. Agli amministratori di visualizzare tutti gli impiegati
27. Agli amministratori di modificare la quota di associazione e di rimessaggio
28. Agli amministratori di visualizzare le informazioni sulle quote

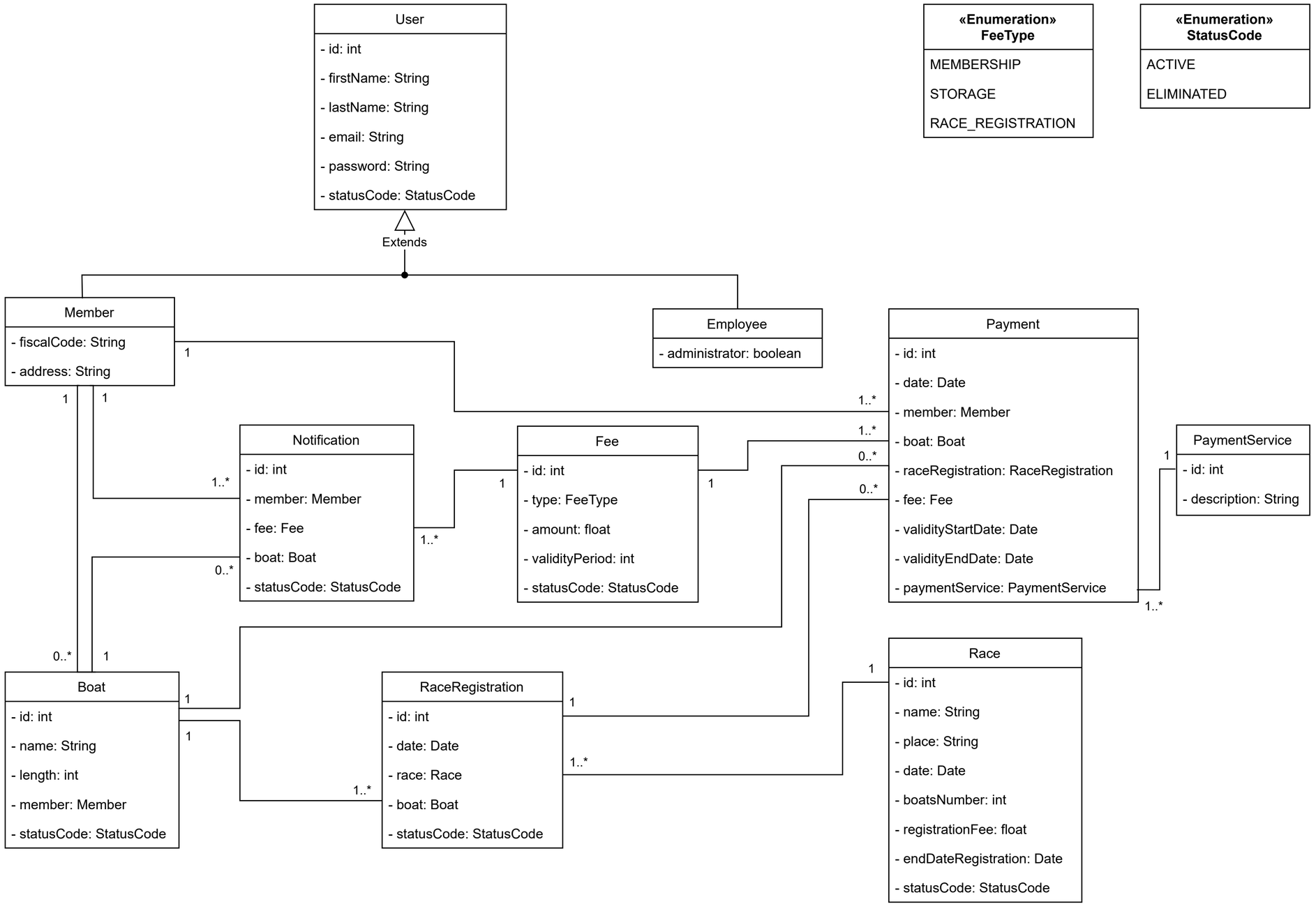
2 Diagramma UML dei casi d'uso




System
Member
User
Employee
Administrator
Login
Add Boat
Update Boat
Delete Boat
View All Boats
Add Race
Update Race
Remove Race
<<Include>>
Remove All Race Registrations
View All Races
View All Race Registrations
Insert Race Registration
<<Include>>
Pay Race Registration Fee
Update Race Registration
Remove Race Registration
<<Include>>
Refund Race Registration Fee
Pay Membership Fee
Pay Storage Fee
View All Payments
<<Include>>
View Membership Fee Payments
<<Include>>
View Storage Fee Payments
<<Include>>
View Race Registration Fee Payments
Add Member
Update Member
Send Notification
<<Include>>
Membership Fee
<<Include>>
Storage Fee


Member
User
Employee
Administrator
System
View All Users
<<Include>>
View All Members
<<Include>>
View All Employees
Add Employee
Update Employee
Remove Employee
Update Fee
View Fees

3 Diagramma UML delle classi



4 Diagrammi UML di sequenza

4.1 Aggiunta di una nuova imbarcazione

4.1.1 Descrizione testuale

1. L'attore (socio oppure impiegato), dopo essersi autenticato, deve cliccare il bottone "**add boat**", nella pagina principale delle barche (gestita dalla classe **BoatsController**), per aggiungere una nuova imbarcazione
2. L'azione del click sul bottone, genera un evento che richiama il metodo **initUpsertBoat** della classe **App**. A tale metodo viene passato il parametro *null* in quanto deve essere aggiunta una nuova imbarcazione
3. Il metodo **initUpsertBoat** carica la pagina contenente il form che permette all'utente di inserire i dati relativi alla nuova imbarcazione. Tale pagina è gestita dalla classe **UpsertBoatController**
4. Quando l'attore clicca sul bottone "**save**" del form, genera un nuovo evento che richiama il metodo **insertBoat**
5. A questo punto, il metodo **insertBoat** della classe **UpsertBoatController** esegue diverse operazioni
6. Il metodo **insertBoat** verifica se l'utente ha inserito correttamente il nome e la lunghezza dell'imbarcazione. Nel caso in cui almeno un campo è vuoto, tale metodo chiama la funzione **showAlert** della classe **App** che mostra un messaggio di errore all'utente
7. Il metodo **insertBoat** richiama la funzione **convertToInteger** della classe **App** per convertire la lunghezza inserita dall'utente in un numero intero

   7.1 Se la lunghezza non è un numero corretto, il metodo **convertToInteger** richiama **showAlert** della classe **App**, che mostra l'errore all'utente
8. Il metodo **insertBoat** crea un nuovo oggetto **Boat** con i dati inseriti dall'attore nel form
9. Il metodo **insertBoat** chiama **getResponseType** della classe **ClientHelper**, passando come parametro un nuovo oggetto **Request**. In questo caso i parametri dell'oggetto **Request** sono il tipo di richiesta e una lista contenente: l'oggetto **Boat** creato precedentemente e l'email del socio proprietario dell'imbarcazione (ovvero il socio loggato oppure il socio selezionato dall'impiegato nel form di inserimento)

   9.1 Il metodo **getResponseType** chiama **connection** della classe **ClientHelper**, che permette al client di stabilire una connessione con il server e di effettuare richieste. In particolare, tale metodo salva l'oggetto di tipo **Request** in **ObjectOutputStream**

   9.2 Per gestire la richiesta del client, viene chiamato il metodo **run** della classe **ServerThread**

   9.3 Il metodo **run** legge la richiesta ricevuta dal client (*INSERT_BOAT*) e chiama la funzione **processRequest** per eseguire la richiesta e restituire la risposta corretta. Per semplicità, tale richiesta viene modellata in un altro diagramma di sequenza collegato al principale

   9.3.1 Il metodo **processRequest** controlla il tipo di richiesta e chiama il metodo **insertBoat** della classe **ServerHelper**. Tale metodo recupera dall'oggetto richiesta i dati dell'imbarcazione (oggetto **Boat**) e l'email del socio

   9.3.2 Il metodo **insertBoat** chiama la funzione **getUserByEmail** della classe **UserDAO** passando come parametro l'email del socio per verificare se esiste un utente nel database associato a quella email. Tale metodo, prepara la query per ottenere un utente data l'email, la esegue e restituisce un oggetto **Member**. Se non viene trovato alcun utente, il metodo restituisce *null*

   9.3.3 Nel caso in cui, il metodo **getUserByEmail** di **UserDAO** restituisce *null* allora **insertBoat** di **ServerHelper** imposta tramite la funzione **setResponseType** un errore il quale indica che non esiste nessun utente con l'email inserita

   9.3.4 Il metodo **insertBoat** chiama la funzione **getBoatByName** della classe **BoatDAO** per verificare se esiste già un'imbarcazione con il nome inserito dall'attore ed appartenente all'utente trovato precedentemente. In particolare, tale metodo prepara la query per ottenere una barca dal database dato il nome, la esegue e restituisce un oggetto **Boat**. Se non viene trovata alcuna imbarcazione, il metodo restituisce *null*

   9.3.5 Nel caso in cui il metodo **getBoatByName** di **BoatDAO** restituisce un oggetto **Boat** diverso da *null* allora **insertBoat** di **ServerHelper** imposta tramite la funzione **setResponseType** un errore il quale indica che esiste già un imbarcazione con il nome inserito

   9.3.6 Il metodo **insertBoat** chiama la funzione **checkPaymentMembershipFee** della classe **PaymentDAO** per verificare se il proprietario dell'imbarcazione è in regola con il pagamento della quota di associazione del circolo. In particolare, tale metodo prepara la query, la esegue e restituisce *true* se il pagamento della quota di associazione è stato effettuato altrimenti *false*

9.3.7 Se il metodo **checkPaymentMembershipFee** di **PaymentDAO** restituisce *false* allora **insertBoat** di **ServerHelper** imposta tramite la funzione **setResponseType** un errore relativo al pagamento della quota di associazione

9.3.8 Il metodo **insertBoat** di **ServerHelper** chiama la funzione **insertBoat** della classe **BoatDAO** per inserire l'imbarcazione. Esso prepara la query con i dati inseriti dall'attore e con l'oggetto **Member** che identifica il proprietario dell'imbarcazione. Infine esegue la query

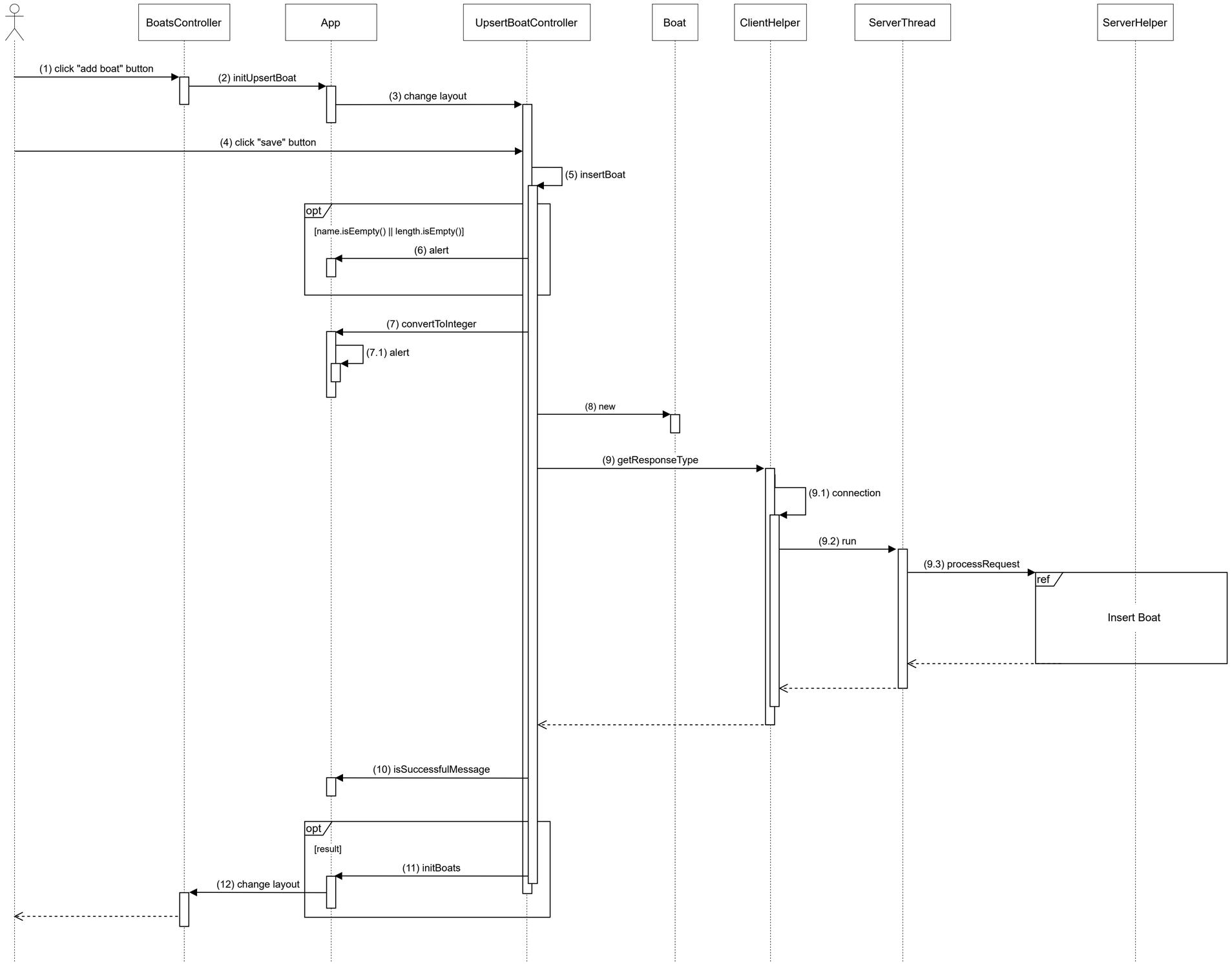
9.3.9 Il metodo **insertBoat** della classe **ServerHelper**, imposta tramite la funzione **setResponseType** un messaggio di successo relativo all'inserimento dell'imbarcazione

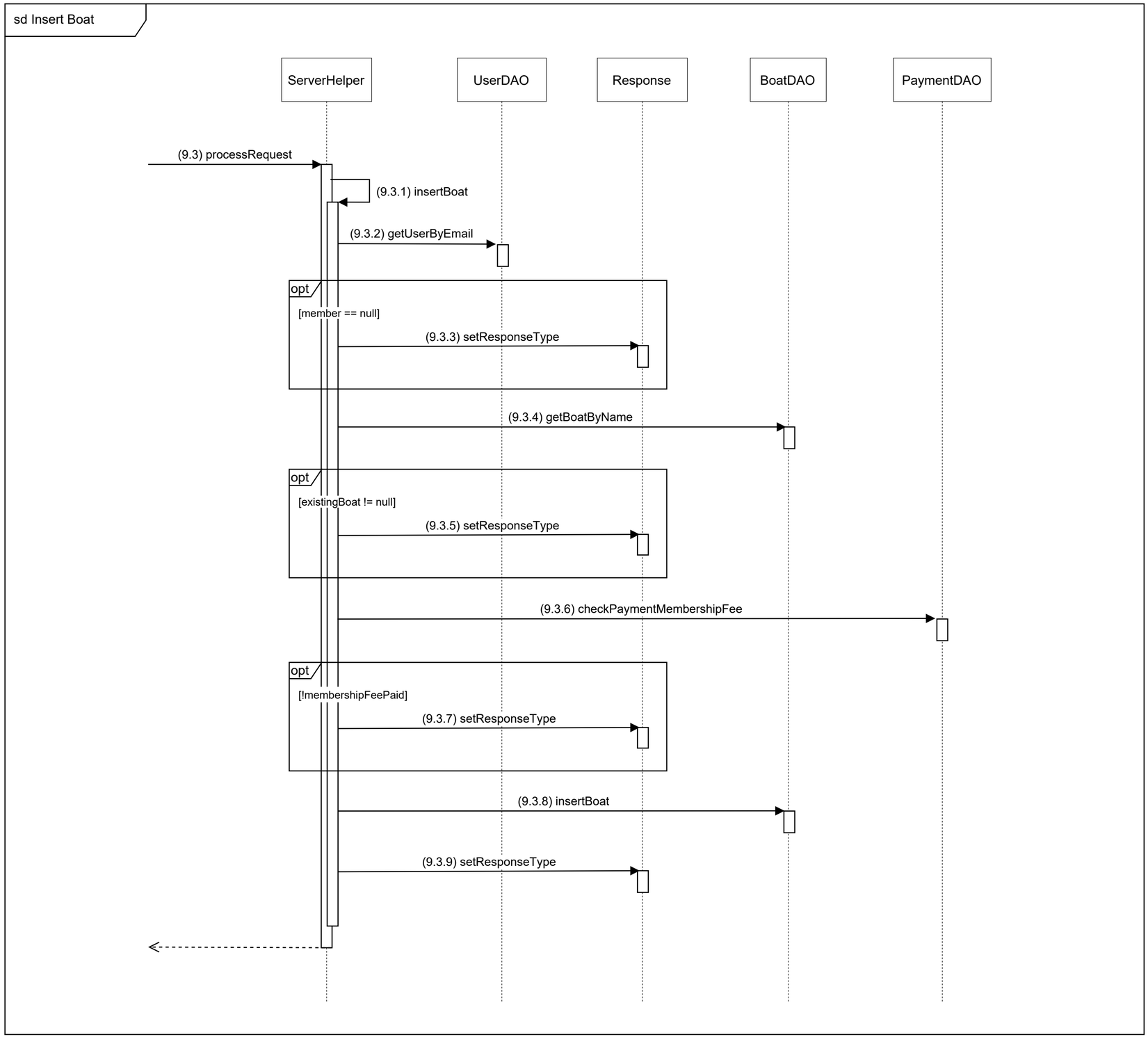
9.4 Il metodo **connection** della classe **ClientHelper** ottiene la risposta dal server tramite l'oggetto **ObjectInputStream** e restituisce a **getResponseType** la risposta. A questo punto, la funzione **getResponseType** verifica la risposta e restituisce un oggetto di tipo **ResponseType** al metodo **insertBoat** della classe **UpsertBoatController**

10. Il metodo **insertBoat** chiama la funzione **isSuccessfulMessage** della classe **App**. Tale metodo controlla se il tipo di risposta ricevuta dal server è positiva oppure è un errore. In entrambi i casi chiama **showAlert**, sempre di **App**, per mostrare il risultato dell'operazione all'attore e successivamente restituisce *true* nel caso in cui l'operazione di inserimento ha avuto successo

11. Se il risultato è *true* allora viene chiamato il metodo **initBoats** della classe **App**

12. Il metodo **initBoats** carica la pagina principale nella quale sono presenti tutte le imbarcazioni, compresa quella nuova appena inserita.


BoatsController
App
UpsertBoatController
Boat
ClientHelper
ServerThread
ServerHelper
(1) click "add boat" button
(2) initUpsertBoat
(3) change layout
(4) click "save" button
(5) insertBoat
opt
[name.isEempty() || length.isEmpty()]
(6) alert
(7) convertToInteger
(7.1) alert
(8) new
(9) getResponseType
(9.1) connection
(9.2) run
(9.3) processRequest
ref
Insert Boat
(10) isSuccessfulMessage
opt
[result]
(11) initBoats
(12) change layout


sd Insert Boat
ServerHelper
UserDAO
Response
BoatDAO
PaymentDAO
(9.3) processRequest
(9.3.1) insertBoat
(9.3.2) getUserByEmail
opt
[member == null]
(9.3.3) setResponseType
(9.3.4) getBoatByName
opt
[existingBoat != null]
(9.3.5) setResponseType
(9.3.6) checkPaymentMembershipFee
opt
[!membershipFeePaid]
(9.3.7) setResponseType
(9.3.8) insertBoat
(9.3.9) setResponseType

4.2 Iscrizione di un'imbarcazione ad una gara

4.2.1 Descrizione testuale

1. L'attore (socio oppure impiegato), dopo essersi autenticato, deve cliccare il link "**races**" nel menù in alto (gestito dalla classe **MainController**) per visualizzare la pagina relativa a tutte le gare
2. L'azione del click sul link, genera un evento che richiama il metodo **initRaces** della classe **App**
3. Il metodo **initRaces** carica la pagina contenente l'elenco di tutte le gare. Tale pagina è gestita dalla classe **RacesController**
4. A questo punto, l'attore deve premere la barra spaziatrice su una determinata riga della tabella relativa alle gare per iscrivere un'imbarcazione alla gara selezionata
5. Quando l'attore preme la barra spaziatrice, genera un nuovo evento che richiama il metodo **initUpsertRaceRegistration**, della classe **App**, a cui viene passato l'oggetto **Race** relativo alla gara selezionata
6. Il metodo **initUpsertRaceRegistration** carica la pagina contenente il form che permette all'utente di selezionare l'imbarcazione da iscrivere alla gara e il metodo di pagamento da utilizzare per pagare la quota di iscrizione alla gara. Nel caso in cui l'attore è un impiegato, esso dovrà scegliere prima il socio e successivamente una sua imbarcazione. In particolare, tale pagina è gestita dalla classe **UpsertRaceRegistrationController**
7. Quando l'attore clicca sul bottone "**save**" del form, genera un nuovo evento che richiama il metodo **insertRaceRegistration**
8. Il metodo **insertRaceRegistration** della classe **UpsertRaceRegistrationController** esegue diverse operazioni
9. Il metodo **insertRaceRegistration** chiama la funzione **getResponseType** della classe **ClientHelper**, passando come parametro un nuovo oggetto **Request**. In questo caso i parametri dell'oggetto **Request** sono il tipo di richiesta e una lista contenente i dati del form (email del socio, nome dell'imbarcazione, descrizione del metodo di pagamento e la gara selezionata).

   9.1 Il metodo **getResponseType** chiama la funzione **connection** della classe **ClientHelper**, che permette al client di stabilire una connessione con il server e di effettuare richieste. In particolare, tale metodo salva l'oggetto di tipo **Request** in **ObjectOutputStream**

   9.2 Per gestire la richiesta del client, viene chiamato il metodo **run** della classe **ServerThread**

   9.3 Il metodo **run** legge la richiesta ricevuta dal client (*INSERT_RACE_REGISTRATION*) e chiama la funzione **processRequest** per eseguire la richiesta e restituire la risposta corretta. Per semplicità, la richiesta viene modellata in un altro diagramma di sequenza collegato al principale

   9.3.1 Il metodo **processRequest** controlla il tipo di richiesta e chiama **insertRaceRegistration** della classe **ServerHelper**. Tale metodo recupera dall'oggetto richiesta i dati inviati dal client

   9.3.2 Il metodo **insertRaceRegistration** chiama la funzione **getUserByEmail** della classe **UserDAO** passando come parametro l'email del socio per verificare se esiste un utente nel database associato a quella email. Tale metodo, prepara la query per ottenere un utente data l'email, la esegue e restituisce un oggetto **Member**. Se non viene trovato alcun utente, il metodo restituisce *null*

   9.3.3 Nel caso in cui, il metodo **getUserByEmail** di **UserDAO** restituisce *null* allora **insertRaceRegistration** di **ServerHelper** imposta tramite la funzione **setResponseType** un errore il quale indica che non esiste nessun utente con l'email data

   9.3.4 Il metodo **insertRaceRegistration** chiama la funzione **getBoatByName** della classe **BoatDAO** per verificare se esiste un'imbarcazione, dell'utente trovato precedentemente, con il nome inserito dall'attore. In particolare, tale metodo prepara la query per ottenere una barca nel database dato il nome, la esegue e restituisce un oggetto **Boat**. Se non viene trovata alcuna imbarcazione, il metodo restituisce *null*

   9.3.5 Nel caso in cui, il metodo **getBoatByName** di **BoatDAO** restituisce *null* allora **insertRaceRegistration** di **ServerHelper** imposta tramite la funzione **setResponseType** un errore il quale indica che non esiste alcuna imbarcazione con il nome dato e di proprietà del socio trovato precedentemente

   9.3.6 Il metodo **insertRaceRegistration** chiama la funzione privata **checkOpenRegistration** della classe **ServerHelper**, che verifica se sono aperte le iscrizioni alla gara. Vale a dire che l'ultimo giorno per iscrivere un'imbarcazione ad una gara deve essere maggiore o uguale rispetto alla data in cui l'attore vuole iscrivere un'imbarcazione. In questo caso la funzione restituisce *true*, altrimenti *false*

   9.3.7 Nel caso in cui la funzione **checkOpenRegistration** restituisce *false* allora **insertRaceRegistration** imposta tramite la funzione **setResponseType** un errore il quale indica che non è più possibile iscrivere un'imbarcazione alla gara

   9.3.8 Il metodo **insertRaceRegistration** chiama la funzione **getNumberBoatsInRace** della classe **RaceRegistrationDAO** per recuperare il numero di imbarcazioni iscritte alla gara. In particolare, tale metodo prepara la query per ottenere il numero totale di imbarcazioni iscritte, la esegue ed aggiunge uno (ovvero l'imbarcazione che si sta iscrivendo in questo momento)

9.3.9 Se il numero totale di imbarcazioni iscritte alla gara (compresa quella che l'attore vuole iscrivere) è maggiore del numero di imbarcazioni totali che possono partecipare alla gara allora il metodo **insertRaceRegistration** imposta tramite la funzione **setResponseType** un errore il quale indica che è stato raggiunto il numero massimo di imbarcazioni

9.3.10 Il metodo **insertRaceRegistration** chiama la funzione **checkBoatInRaceByMember** della classe **RaceRegistrationDAO** per verificare se il socio (loggato oppure selezionato dall'impiegato) ha già registrato un'imbarcazione alla gara. Tale metodo esegue la query e restituisce *true* se il socio non ha iscritto alcuna imbarcazione alla gara

9.3.11 Nel caso in cui il metodo **checkBoatInRaceByMember** di **RaceRegistrationDAO** restituisce *false* allora **insertRaceRegistration** imposta tramite la funzione **setResponseType** un errore il quale indica che il socio ha già iscritto un'imbarcazione per la gara scelta

9.3.12 Il metodo **insertRaceRegistration** chiama la funzione privata **checkPaymentsByRaceRegistration** della classe **ServerHelper**, per verificare se il socio ha pagato regolarmente la quota di associazione del circolo e la quota di rimessaggio per l'imbarcazione che vuole iscrivere alla gara. Se almeno una delle due quote non è stata pagata allora ritorna *false*

9.3.13 Nel caso in cui la funzione **checkPaymentsByRaceRegistration** restituisce *false*, il metodo **insertRaceRegistration** imposta tramite **setResponseType** un errore nel pagamento delle quote

9.3.14 Il metodo **insertRaceRegistration** chiama **getPaymentServiceByDescription** della classe **PaymentDAO**, passando come parametro la descrizione del servizio di pagamento scelto dall'attore per pagare la quota di iscrizione alla gara. Tale metodo verifica se esiste un servizio di pagamento nel database associato a quella descrizione; esso prepara la query, la esegue e restituisce un oggetto **PaymentService**. Se non viene trovato alcun servizio di pagamento, il metodo restituisce *null*

9.3.15 Nel caso in cui, il metodo **getPaymentServiceByDescription** di **PaymentDAO** restituisce *null* allora **insertRaceRegistration** di **ServerHelper** imposta tramite la funzione **setResponseType** un errore il quale indica che non esiste alcun servizio di pagamento associato a quella descrizione

9.3.16 Il metodo **insertRaceRegistration** chiama **registerBoatAtRace** della classe **RaceRegistrationDAO** per iscrivere l'imbarcazione alla gara. Esso prepara la query di inserimento con il codice della gara e dell'imbarcazione ed infine esegue l'interrogazione

9.3.17 A questo punto, Il metodo **insertRaceRegistration** recupera l'iscrizione appena effettuata e l'oggetto **Fee** relativo alla quota di iscrizione alla gara e successivamente chiama la funzione **payFee** della classe **PaymentDAO** per pagare l'importo relativo alla quota

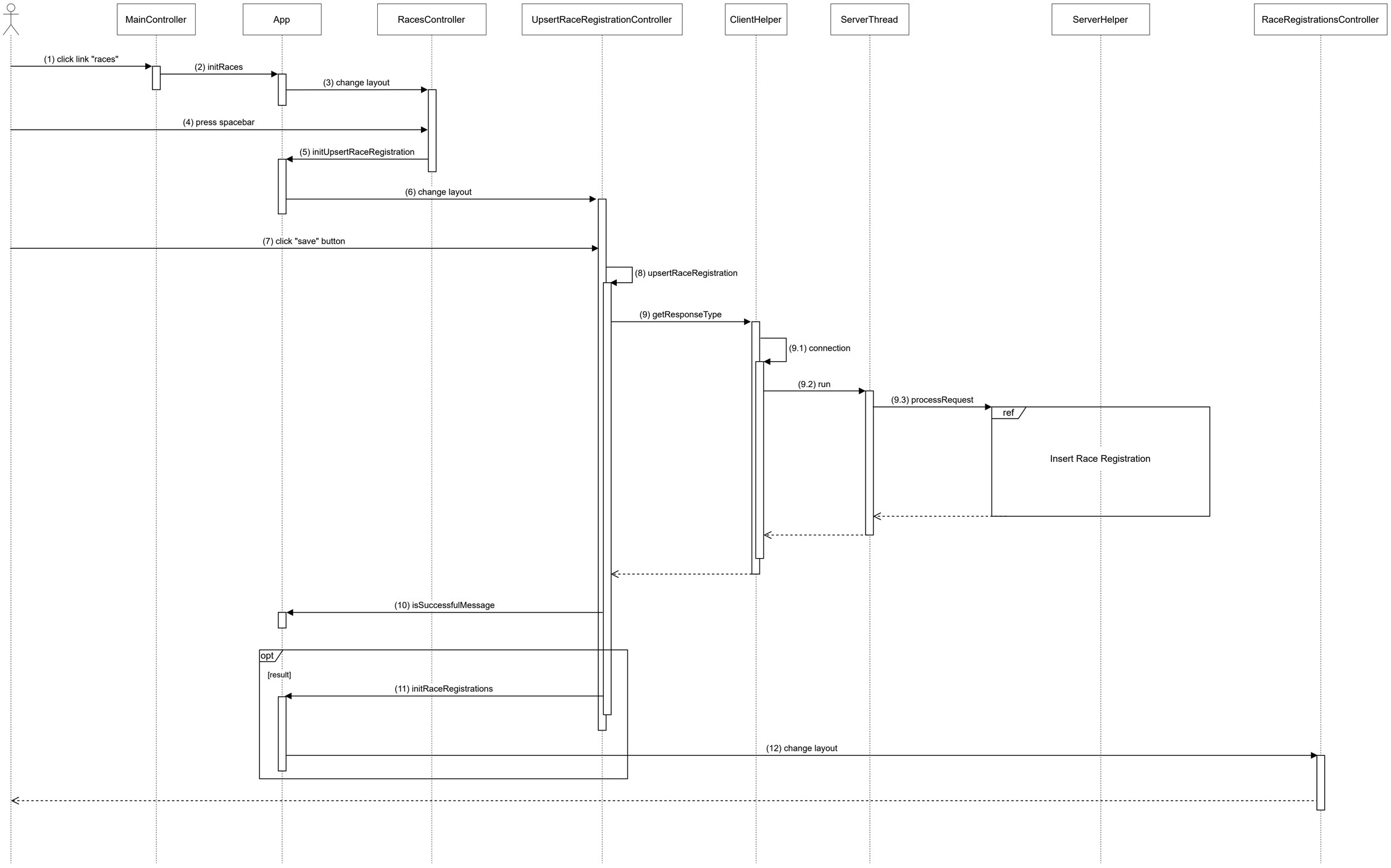
9.3.18 Il metodo **insertRaceRegistration** della classe **ServerHelper**, imposta tramite la funzione **setResponseType** un messaggio di successo relativo all'iscrizione dell'imbarcazione alla gara

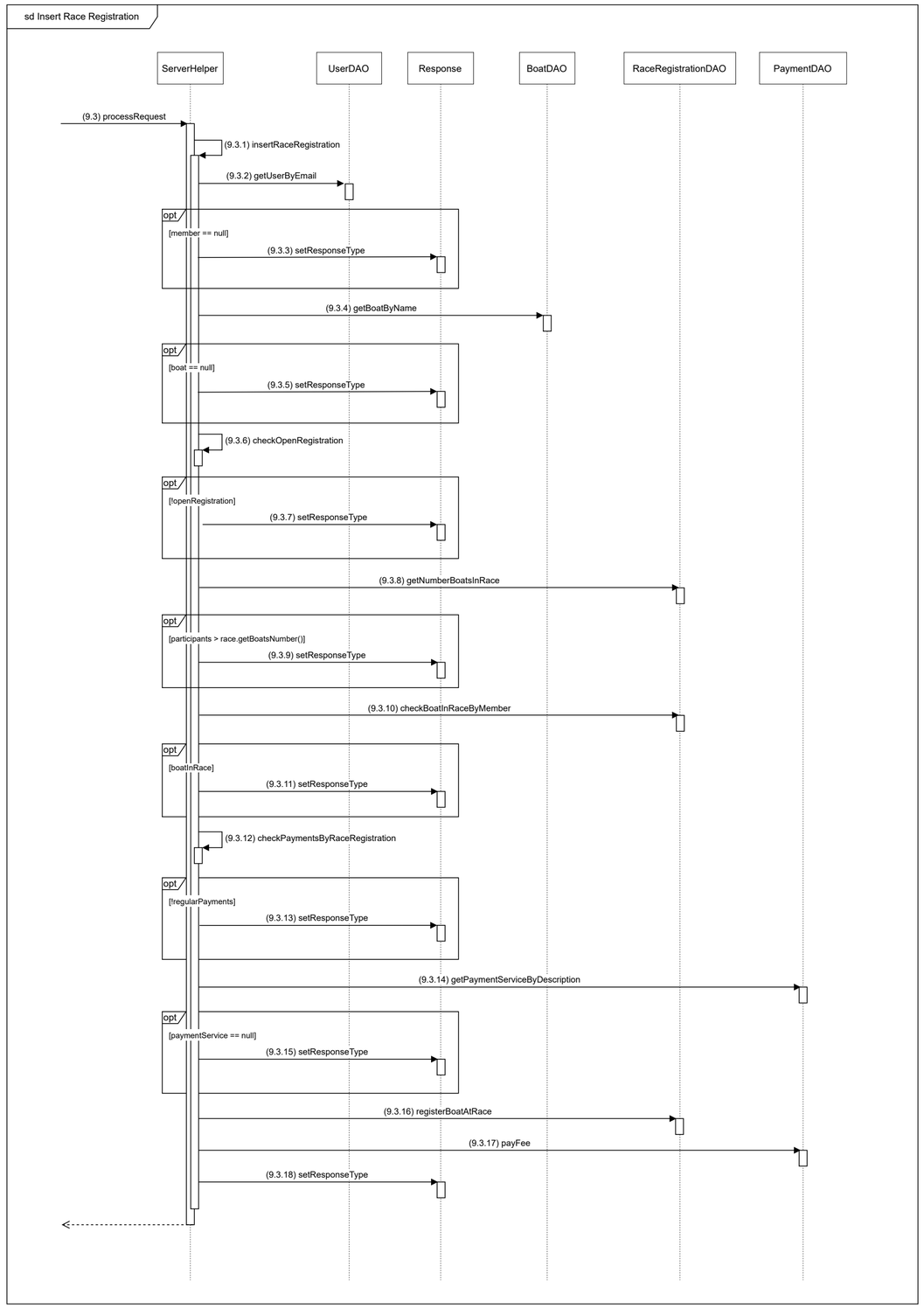
9.4 Il metodo **connection** della classe **ClientHelper** ottiene la risposta dal server tramite l'oggetto **ObjectInputStream** e restituisce a **getResponseType** la risposta. La funzione **getResponseType** verifica la risposta e restituisce un oggetto di tipo **ResponseType** al metodo **insertRaceRegistration** della classe **UpsertRaceRegistrationController**

10. Il metodo **insertRaceRegistration** chiama la funzione **isSuccessfulMessage** della classe **App**. Tale metodo controlla se la il tipo di risposta ricevuta dal server è positiva oppure è un errore. In entrambi i casi chiama **showAlert**, sempre di **App**, per mostrare il risultato dell'operazione all'attore e successivamente restituisce *true* nel caso in cui l'operazione di inserimento ha avuto successo

11. Se il risultato è *true* allora viene chiamato il metodo **initRaceRegistrations** della classe **App**

12. Il metodo **initRaceRegistrations** carica la pagina relativa alle iscrizioni della gara selezionata dall'attore (compresa quella nuova appena inserita) gestita dalla classe **RaceRegistrationsController**


MainController
App
RacesController
UpsertRaceRegistrationController
ClientHelper
ServerThread
ServerHelper
RaceRegistrationsController
(1) click link "races"
(2) initRaces
(3) change layout
(4) press spacebar
(5) initUpsertRaceRegistration
(6) change layout
(7) click "save" button
(8) upsertRaceRegistration
(9) getResponseType
(9.1) connection
(9.2) run
(9.3) processRequest
ref
Insert Race Registration
(10) isSuccessfulMessage
opt
[result]
(11) initRaceRegistrations
(12) change layout


sd Insert Race Registration
ServerHelper
UserDAO
Response
BoatDAO
RaceRegistrationDAO
PaymentDAO
(9.3) processRequest
(9.3.1) insertRaceRegistration
(9.3.2) getUserByEmail
opt
[member == null]
(9.3.3) setResponseType
(9.3.4) getBoatByName
opt
[boat == null]
(9.3.5) setResponseType
(9.3.6) checkOpenRegistration
opt
[!openRegistration]
(9.3.7) setResponseType
(9.3.8) getNumberBoatsInRace
opt
[participants > race.getBoatsNumber()]
(9.3.9) setResponseType
(9.3.10) checkBoatInRaceByMember
opt
[boatInRace]
(9.3.11) setResponseType
(9.3.12) checkPaymentsByRaceRegistration
opt
[!regularPayments]
(9.3.13) setResponseType
(9.3.14) getPaymentServiceByDescription
opt
[paymentService == null]
(9.3.15) setResponseType
(9.3.16) registerBoatAtRace
(9.3.17) payFee
(9.3.18) setResponseType

4.3 Rinnovo della quota di associazione

4.3.1 Descrizione testuale

1. L'attore (socio oppure impiegato), dopo essersi autenticato, deve cliccare il link "**payments**" nel menù in alto (gestito dalla classe **MainController**) per visualizzare la pagina relativa a tutti i pagamenti
2. L'azione del click sul link, genera un evento che richiama il metodo **initPayments** della classe **App**
3. Il metodo **initPayments** carica la pagina contenente l'elenco di tutti i pagamenti. Tale pagina è gestita dalla classe **PaymentsController**
4. A questo punto, l'attore deve premere il bottone "**pay membership fee**" nella sezione principale riguardante i pagamenti delle quote di associazione al circolo
5. L'azione del click sul bottone, genera un evento che richiama il metodo **initPayFee** della classe **App**. A tale metodo viene passato il tipo di quota che l'utente deve pagare, in questo caso la quota di associazione
6. Il metodo **initPayFee** carica la pagina contenente il form che permette all'utente di scegliere il servizio di pagamento con il quale pagare la quota. Nel caso in cui l'attore è un impiegato, esso dovrà selezionare il socio per il quale gestirà il pagamento. In particolare, tale pagina è gestita dalla classe **PayFeeController**
7. Quando l'attore clicca sul bottone "**pay**" del form, genera un nuovo evento che richiama il metodo **payFee**
8. Il metodo **payFee** della classe **PayFeeController** esegue diverse operazioni
9. Il metodo **payFee** richiama **getResponseType** della classe **ClientHelper**, passando come parametro un nuovo oggetto **Request**. In questo caso i parametri dell'oggetto **Request** sono il tipo di richiesta e una lista contenente i dati del form (email del socio e descrizione del servizio di pagamento selezionato)

   9.1 Il metodo **getResponseType** richiama **connection** della classe **ClientHelper**, che permette al client di stabilire una connessione con il server e di effettuare richieste. In particolare, tale metodo salva l'oggetto di tipo **Request** in **ObjectOutputStream**

   9.2 Per gestire la richiesta del client, viene chiamato il metodo **run** della classe **ServerThread**

   9.3 Il metodo **run** legge la richiesta ricevuta dal client (*PAY_FEE*) e chiama la funzione **processRequest** per eseguire la richiesta e restituire la risposta corretta. Per semplicità, la richiesta viene modellata in un altro diagramma di sequenza collegato al principale

   9.3.1 Il metodo **processRequest** controlla il tipo di richiesta e richiama **payFee** della classe **ServerHelper**. Tale metodo recupera dall'oggetto richiesta i dati inviati dal client

   9.3.2 Il metodo **payFee** chiama la funzione **getUserByEmail** della classe **UserDAO** passando come parametro l'email del socio per verificare se esiste un utente nel database associato a quella email. Tale metodo, prepara la query per ottenere un utente data l'email, la esegue e restituisce un oggetto **Member**. Se non viene trovato alcun utente, il metodo restituisce *null*

   9.3.3 Nel caso in cui, il metodo **getUserByEmail** di **UserDAO** restituisce *null* allora **payFee** di **ServerHelper** imposta tramite la funzione **setResponseType** un errore il quale indica che non esiste nessun utente con l'email data

   9.3.4 Il metodo **payFee** chiama la funzione **getFeeByType** della classe **FeeDAO** per ottenere l'oggetto **Fee** dato il tipo, ovvero **Membership**. Tale funzione, prepara la query di selezione della quota di associazione dal database, la esegue e restituisce l'oggetto **Fee** corrispondente

   9.3.5 Il metodo **payFee** chiama **getPaymentServiceByDescription** della classe **PaymentDAO** passando come parametro la descrizione del servizio di pagamento scelto dall'attore per pagare la quota di associazione al circolo. Tale metodo verifica se esiste un servizio di pagamento nel database associato a quella descrizione; prepara la query, la esegue e restituisce un oggetto **PaymentService**. Se non viene trovato alcun servizio di pagamento, il metodo restituisce *null*

   9.3.6 Nel caso in cui, il metodo **getPaymentServiceByDescription** di **PaymentDAO** restituisce *null* allora **payFee** di **ServerHelper** imposta tramite la funzione **setResponseType** un errore il quale indica che non esiste alcun servizio di pagamento associato a quella descrizione

   9.3.7 Il metodo **payFee** chiama la funzione **getLastPaymentFee** della classe **PaymentDAO** per recuperare la data dell'ultimo pagamento relativo alla quota di associazione

   9.3.8 A questo punto, la data dell'ultimo pagamento e una variabile di tipo *boolean* che indica se il socio (loggato oppure selezionato dall'impiegato) ha depositato imbarcazioni nel circolo, consentono di determinare il modo in cui deve essere pagata la quota di associazione. Infatti, se la da data dell'ultimo pagamento è *null*, o è antecedente alla data attuale, oppure se il socio non ha imbarcazioni nel circolo, il metodo **payFee** di **ServerHelper** chiama **payFee** di **PaymentDAO** per completare il pagamento della quota di associazione a partire dalla data attuale. In particolare, quest'ultimo metodo (**payFee** di **PaymentDAO**) prepara una query di inserimento con tutti i dati del pagamento e la esegue, in modo da aggiungere il nuovo pagamento al database

9.3.9 Al contrario, se il socio ha almeno un'imbarcazione depositata nel circolo, il metodo **payFee** chiama la funzione **payFee** di **PaymentDAO** ciclicamente per completare i pagamenti della quota di associazione a partire dall'ultimo pagamento effettuato

9.3.10 Dopo aver pagato la quota di associazione, il metodo **payFee** di **ServerHelper** richiama la funzione **updateStatusCodeNotification** della classe **NotificationDAO** per aggiornare lo stato relativo alla notifica di pagamento (se precedentemente inviata da un impiegato)

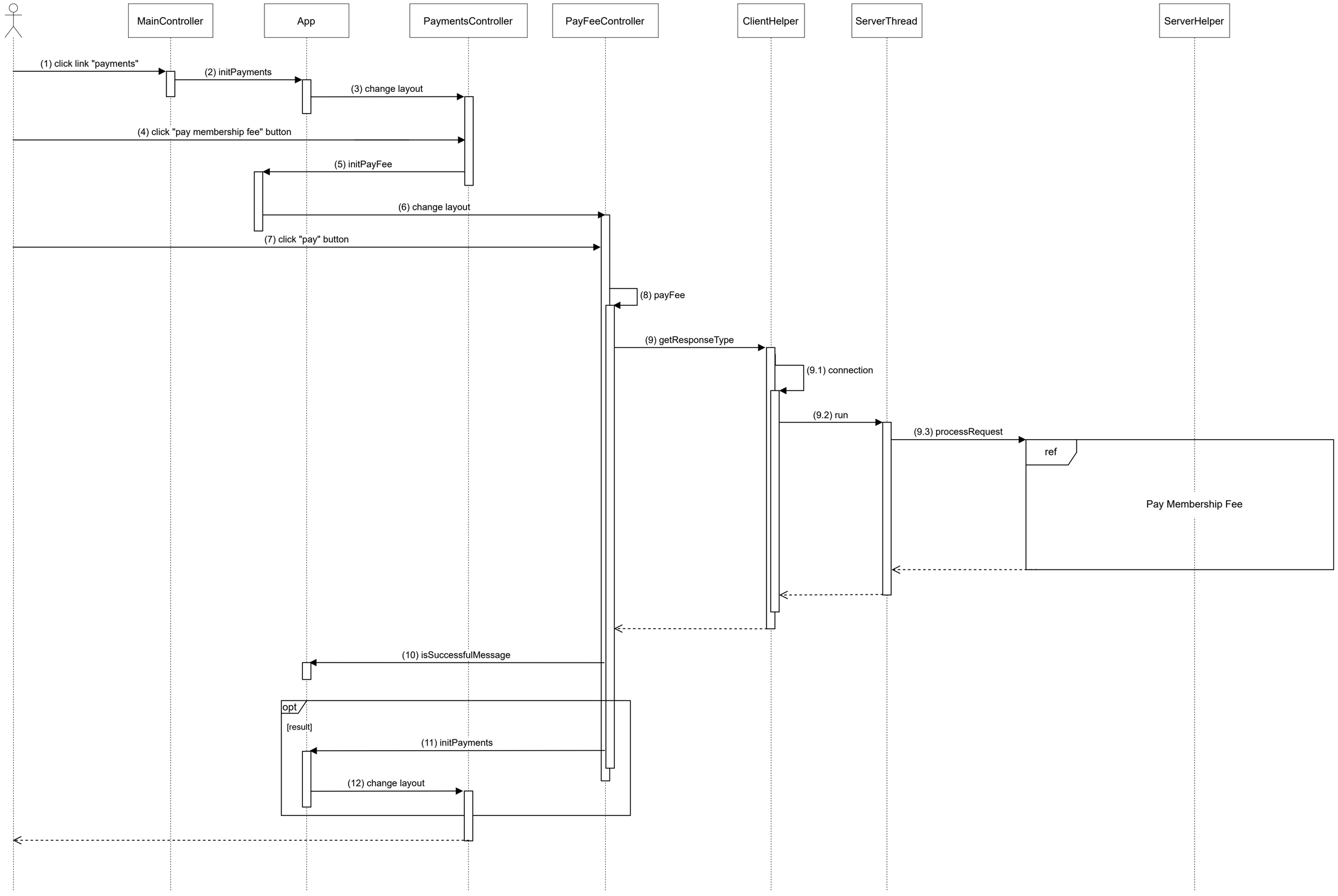
9.3.11 Il metodo **payFee** della classe **ServerHelper**, imposta tramite la funzione **setResponseType** un messaggio di successo relativo al pagamento della quota di associazione

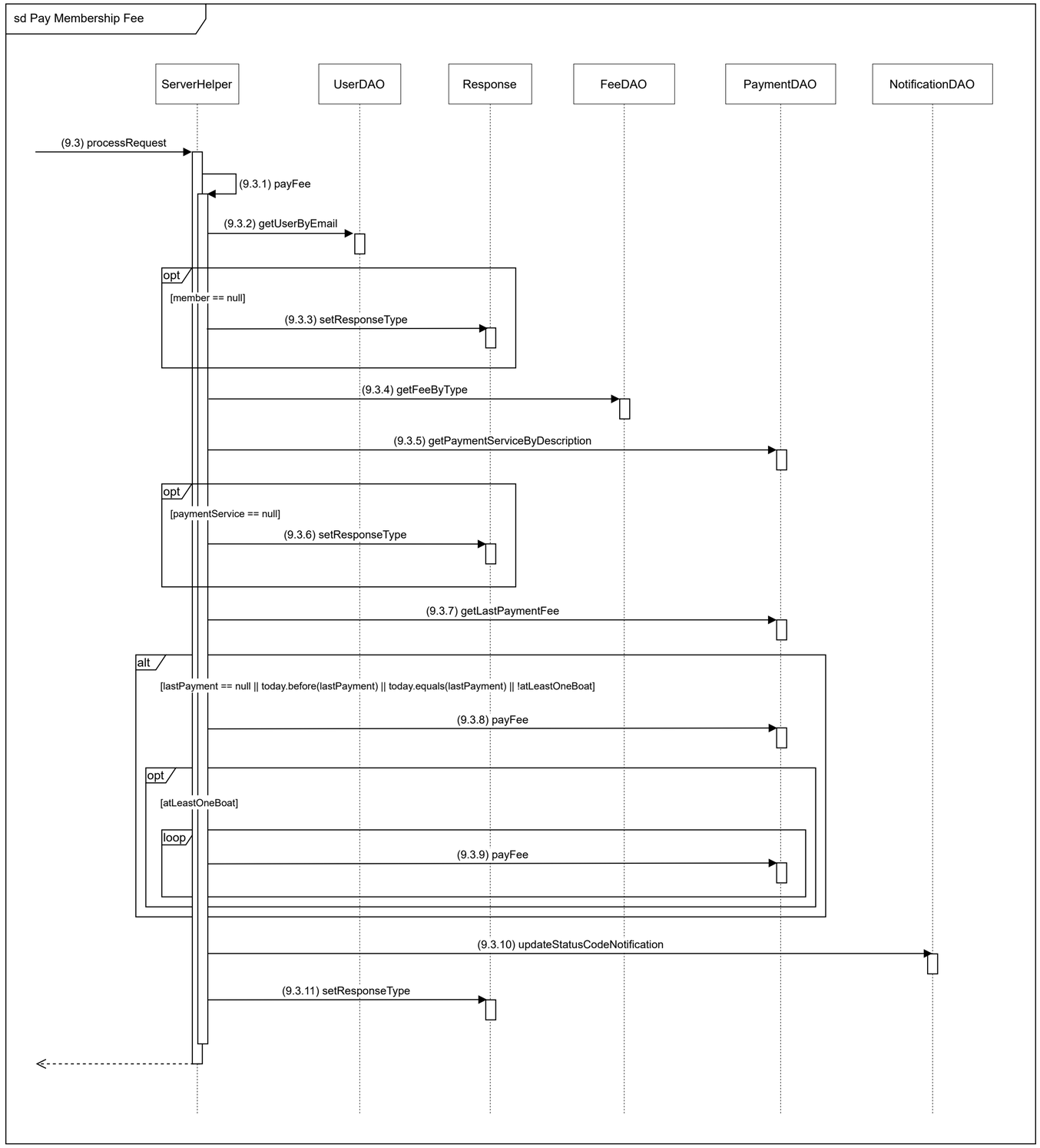
9.4 Il metodo **connection** della classe **ClientHelper** ottiene la risposta dal server tramite l'oggetto **ObjectInputStream** e restituisce a **getResponseType** la risposta. La funzione **getResponseType** verifica la risposta e restituisce un oggetto di tipo **ResponseType** al metodo **payFee** della classe **PayFeeController**

10. Il metodo **payFee** chiama la funzione **isSuccessfulMessage** della classe **App**. Tale metodo controlla se il tipo di risposta ricevuta dal server è positiva oppure è un errore. In entrambi i casi chiama la funzione **showAlert**, sempre di **App**, per mostrare il risultato dell'operazione all'attore e successivamente restituisce *true* nel caso in cui l'operazione di inserimento ha avuto successo

11. Se il risultato è *true* allora viene chiamato il metodo **initPayments** della classe **App**

12. Il metodo **initPayments** carica la pagina relativa ai pagamenti, compreso quello appena effettuato


MainController
App
PaymentsController
PayFeeController
ClientHelper
ServerThread
ServerHelper
(1) click link "payments"
(2) initPayments
(3) change layout
(4) click "pay membership fee" button
(5) initPayFee
(6) change layout
(7) click "pay" button
(8) payFee
(9) getResponseType
(9.1) connection
(9.2) run
(9.3) processRequest
ref
Pay Membership Fee
(10) isSuccessfulMessage
opt
[result]
(11) initPayments
(12) change layout


sd Pay Membership Fee
ServerHelper
UserDAO
Response
FeeDAO
PaymentDAO
NotificationDAO
(9.3) processRequest
(9.3.1) payFee
(9.3.2) getUserByEmail
opt
[member == null]
(9.3.3) setResponseType
(9.3.4) getFeeByType
(9.3.5) getPaymentServiceByDescription
opt
[paymentService == null]
(9.3.6) setResponseType
(9.3.7) getLastPaymentFee
alt
[lastPayment == null || today.before(lastPayment) || today.equals(lastPayment) || !atLeastOneBoat]
(9.3.8) payFee
opt
[atLeastOneBoat]
loop
(9.3.9) payFee
(9.3.10) updateStatusCodeNotification
(9.3.11) setResponseType