Requisiti funzionali del sistema Sailing club

Il sistema software Sailing Club deve permettere la gestione di un circolo velico.
Il sistema dovrà gestire due tipi di account uno per i soci e uno per lo staff.
Un socio deve essere in grado di registrarsi inserendo le proprie informazioni personali e scegliendo un username e una password per identificarsi ed accedere al sistema.
Gli account dello staff saranno memorizzati già all'interno del sistema.
Un socio dopo aver effettuato il login deve poter:

- Aggiungere e rimuovere delle barche;
- Visualizzare le barche che ha memorizzato nel sistema;
- Pagare la tassa di associazione e di rimessaggio delle barche da lui possedute (entrambe con scadenza annuale);
- Visualizzare le gare organizzate dal circolo e potersi iscrivere ad esse pagando la tassa d'iscrizione;
- Visualizzare i pagamenti da lui effettuati.

Lo staff del circolo potrà accedere al sistema con il proprio account e gestire:

- Le informazioni dei membri già registrati, aggiungendo o rimuovendo barche di un singolo membro;
- L'aggiunta e la rimozione dell'account di un membro;
- L'aggiunta e la rimozione di una gara.

Inoltre, deve essere in grado di visualizzare oltre alle iscrizioni alle gare anche uno storico di tutti i pagamenti effettuati dai membri riguardanti le tasse d'iscrizione alle gare, di rimessaggio e di associazione, con l'eventuale possibilità di poter mandare delle notifiche ai membri per il pagamento della quota di rimessaggio o della quota di associazione.

<<System>>
Sailing Club

Unregistered user

Sign up

Condition: user has entered all the data correctly

Display success

Extends

Extends

Display error

Condition: user has entered an invalid username or tax code

Condition: receives a non-payment notification if a staff has reported

Display notification

Extends

Sign in

0.. 1

Condition: member is registered and correct credential

Extends

Display success

<<include>>

View boats

<<include>>

Condition: wrog credential or the member is not registered

Extends

Display error

<<include>>

Add boat

<<include>>

<<include>>

Remove boat

<<include>>

Pay storage fee boat

Subscribe race

Pay membership fee

Extends

Extends

Extends

People

Member

Staff

Condition: Staff is registered and correct credential

Display success

Extends

Login

Display error

Extends

Condition: wrog credential

Add boat

<<include>>

Remove Boat

<<include>>

Choice member

Add member

Delete member

Add race

Remove race

Choice race

View race participant

<<include>>

Condition: the staff checks the payments and can send a notification

Extends

View storage fee payments

Send notice no payment

Extends

View membership fee payments

<<include>>

<<include>>

View payment of the participation fee in the race

<<include>>

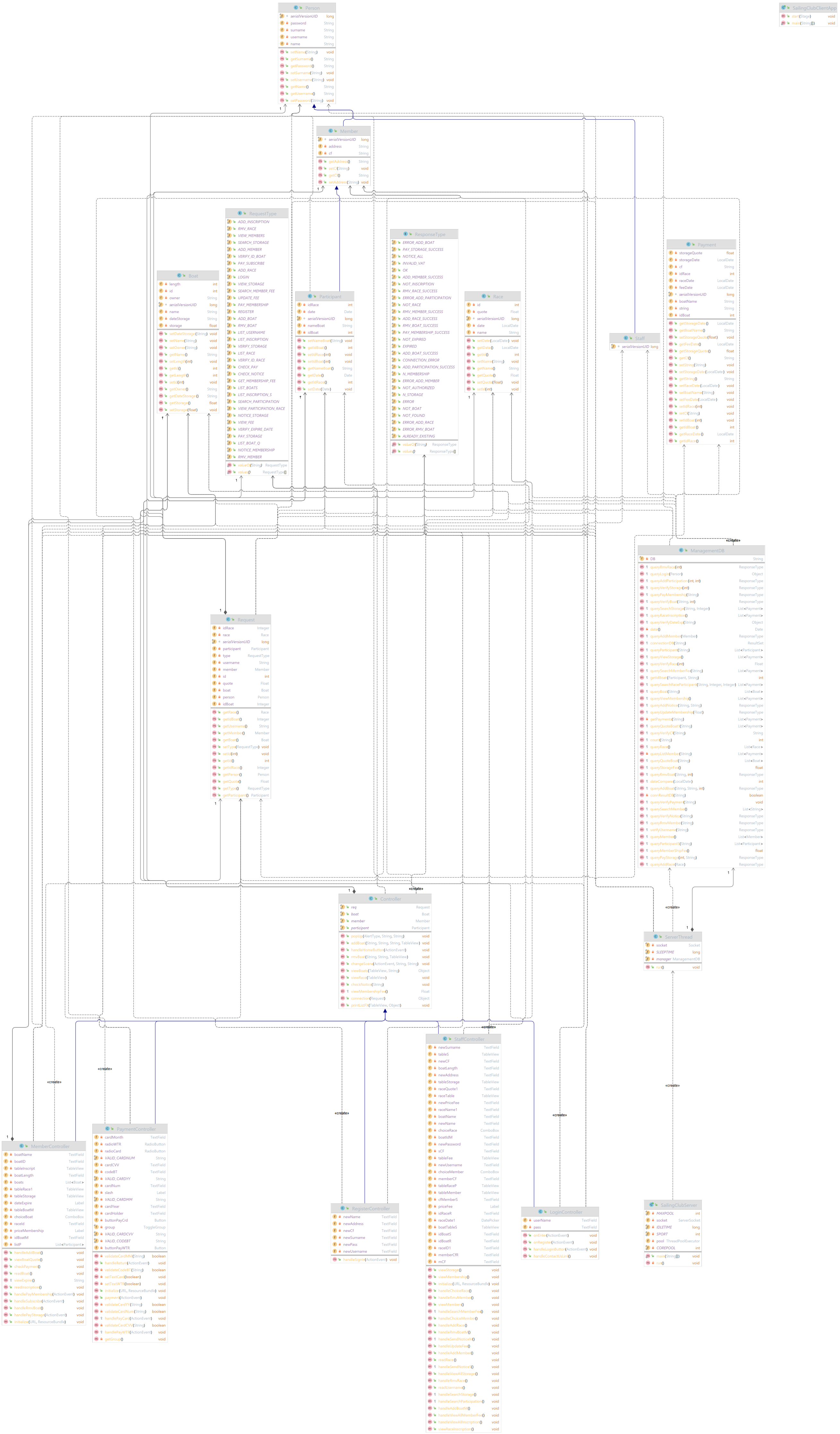
View payments

<<include>>

View to search for tax code or boat id or race id

Update membership fee

Extends


Person
Member
RequestType
ResponseType
Boat
Participant
Race
Payment
Staff
SailingClubClientApp
ManagementDB
Request
Controller
ServerThread
StaffController
MemberController
PaymentController
RegisterController
LoginController
SailingClubServer
«create»

Descrizione testuale sequence diagram

- Add Boat

1) L'attore cioè il Member attiva il metodo handleAddBoat presente nella classe MemberController tramite il pulsante "add boat" presente nella GUI.

2) Il metodo handleAddBoat prende l'input dell'attore e chiama il metodo addBoat presente nella classe Controller.

2.1) Il metodo addBoat ha tre condizioni da verificare:
IF: il campo "name" o il campo "length" sono vuoti il sistema mostra all'attore un popUp (cioè un Alert) descrivendo l'errore.

2.2) IF: il campo "length" non è un numero di tipo intero maggiore di zero il sistema mostra all'attore un popUp (cioè un Alert) descrivendo l'errore.

2.3) IF: il campo "length" non è un numero di tipo intero il sistema mostra all'attore un popUp (cioè un Alert) descrivendo l'errore.

3) Il metodo addBoat crea un nuovo oggetto di tipo Boat passando i campi che ha inserito l'attore.

4) Il metodo addBoat crea un nuovo oggetto di tipo Request utilizzando come parametri l'oggetto Boat stesso e un tipo di RequestType.

5) Il metodo addBoat chiama il metodo connection passando come argomento l'oggetto Request creato pocanzi.

6) Il metodo connection stabilisce una connessione con la classe SailingClubServer (che contiene una porta per creare una server socket e un metodo run() per far gestire le varie richieste dei client dalla classe ServerThread che contiene un metodo run()) e a sua volta trasferisce l'oggetto di tipo Request tramite un ObjectOutputStream.

7) Il metodo run presente nella classe ServerThread legge l'oggetto input ricevuto dal client, verifica il tipo di Request ricevuto e a sua volta utilizza il metodo queryAddBoat passando come parametro l'oggetto Boat creato precedentemente.

8) Il metodo queryAddBoat presente nella classe ManagamentDB crea una connessione con il database del sistema ed effettua l'inserimento della nuova barca, restituendo come risultato un ResponseType che descrive l'effettivo successo dell'inserimento oppure un errore.

9) Il metodo connection restituisce la risposta del server e il metodo addBoat verifica la riposta e mostra un popUp (Alert) informativo all'attore.

10) Di conseguenza il metodo addBoat chiama il metodo viewBoats.
11)Il metodo viewBoats crea un nuovo oggetto di tipo Request e lo utilizza come parametro per il metodo connection, cosicché invia la nuova richiesta al ServerThread.

12) Il metodo run presente nella classe ServerThread legge l'oggetto input ricevuto dal client, verifica il tipo di Request ricevuto e a sua volta utilizza il metodo queryBoat passando come parametro lo username dell'attore (Member).

13) Il metodo queryBoat presente nella classe ManagamentDB a sua volta chiama il metodo queryQuoteBoat1 che seleziona dal database la lista delle barche dell'attore che ha già effettuato il pagamento della tassa di rimessaggio.

14) Il metodo queryBoat successivamente effettua la chiamata al metodo queryStorageFee che tramite il metodo connectionDB si collega al database e ritorna la quota fissa della tassa di rimessaggio.

15) A questo punto il metodo queryBoat effettua la connessione al database tramite sempre il metodo connectionDB e ritorna con la lista di tutte le barche possedute dall'attore.

16) Il metodo queryBoat crea un nuovo oggetto di tipo Payment per confrontare i vari codici identificativi delle barche tramite un for.

17) il metodo queryBoat verifica se l'id della barca è presente nella lista dei pagamenti e a questo punto chiama il metodo dateCompare passandogli la data di pagamento della tassa di quella determinata barca in modo tale da verificare se è ancora valida o no.

18) il metodo dateCompare ritorna un risultato.

18.1) IF: se la data di pagamento è ancora valida il metodo queryBoat crea un nuovo oggetto di tipo Boat con l'attributo date settato con la data di scadenza.

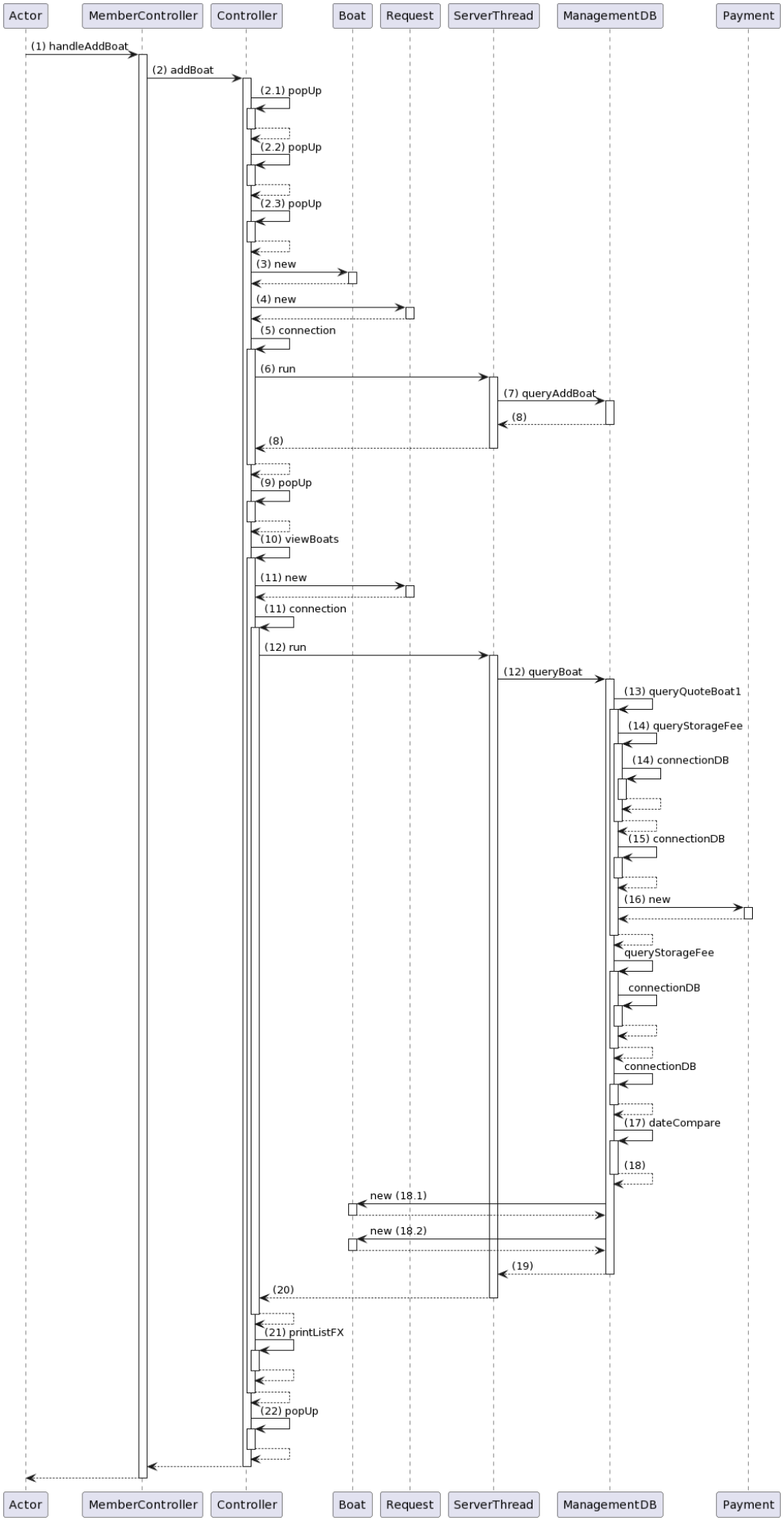
18.2) IF: se la data non è valida o non esiste proprio il metodo queryBoat crea un nuovo oggetto di tipo Boat con l'attributo date settato ad "expired".

19) Il metodo queryBoat ritorna come risposta al client cioè il metodo viewBoats un oggetto di tipo List contenente tutte le barche.

20) il metodo viewBoats appena riceve l'oggetto di risposta lo passa come parametro al metodo printListFx.

21) Il metodo printListFx stampa l'oggetto di tipo List in una determinata tabella della GUI.

22) Infine, il metodo popUp (Alert) mostra un messaggio dell'effettivo inserimento della nuova Barca.


Actor
MemberController
Controller
Boat
Request
ServerThread
ManagementDB
Payment
(1) handleAddBoat
(2) addBoat
(2.1) popUp
(2.2) popUp
(2.3) popUp
(3) new
(4) new
(5) connection
(6) run
(7) queryAddBoat
(8)
(8)
(9) popUp
(10) viewBoats
(11) new
(11) connection
(12) run
(12) queryBoat
(13) queryQuoteBoat1
(14) queryStorageFee
(14) connectionDB
(15) connectionDB
(16) new
queryStorageFee
connectionDB
connectionDB
(17) dateCompare
(18)
new (18.1)
new (18.2)
(19)
(20)
(21) printListFX
(22) popUp
Actor
MemberController
Controller
Boat
Request
ServerThread
ManagementDB
Payment

- Pay membership fee

1) L'attore cioè il Member attiva il metodo handlePayMembership presente nella classe MemberController tramite il pulsante "Pay Fee" presente nella GUI.

1.1) IF: la data del pagamento è diversa dalla Stringa "expired" vuol dire che l'attore ha già pagato la tassa e il sistema tramite il metodo popUp gli mostra l'avviso.

2) Il metodo handlePayMembership crea un nuovo oggetto di tipo Alert per mostrare all'attore un avviso che verrà reindirizzato nella dashboard dei pagamenti.

3) Il metodo handlePayMembership chiama il metodo changeScene presente nella classe controller per passare dalla dashboard dell'attore alla dashboard dei pagamenti; così facendo la gestione passa dal MemberController al PaymentController.

4) L'attore attiva il metodo handlePayCard presente nella classe PaymentController tramite il pulsante "Pay Now" presente nella GUI.

4.1) IF: i campi sono vuoti il sistema mostra all'attore un Alert tramite il metodo popUp avvisando l'attore che ha dimenticato di compilare alcuni campi obbligatori.

4.2) Il metodo handlePayCard chiama il metodo validateCardNum per verificare se il numero della carta di credito inserito dall'attore è valido.

4.3) Il metodo handlePayCard chiama il metodo validateCardMM per verificare se il mese inserito dall'attore è valido.

4.4) Il metodo handlePayCard chiama il metodo validateCardYY per verificare se l'anno inserito dall'attore è valido.

4.5) Il metodo handlePayCard chiama il metodo validateCardCVV per verificare se il codice CVV inserito dall'attore è valido.

5) Successivamente, il metodo handlePayCard chiama il metodo payment presente su PaymentController.

6) il metodo payment deve controllare il tipo di pagamento che sta effettuando l'attore, il controllo è possibile tramite un if che verifica il tipo di pagamento tramite un oggetto di tipo RequestType che può assumere i valori di PAY_STORAGE, PAY_SUBSCRIBE e PAY_MEMBERSHIP.

6.1) IF: la RequestType è di tipo PAY_MEMBERSHIP, il metodo payment crea un nuovo oggetto di tipo Request passando come parametri lo username dell'attore e il tipo di pagamento.

6.2) A sua volta chiama il metodo connection passando come parametro l'oggetto Request creato pocanzi.

6.3) Il metodo connection attiva una connessione con il ServerThread e traferisce l'oggetto Request.

6.4) La classe ServerThread tramite il metodo run legge l'oggetto di tipo Request ricevuto, verifica il tipo di richiesta e chiama il metodo queryPayMembership.

6.5) il metodo queryPayMembership presente nella classe ManagementDB, chiama il metodo date.

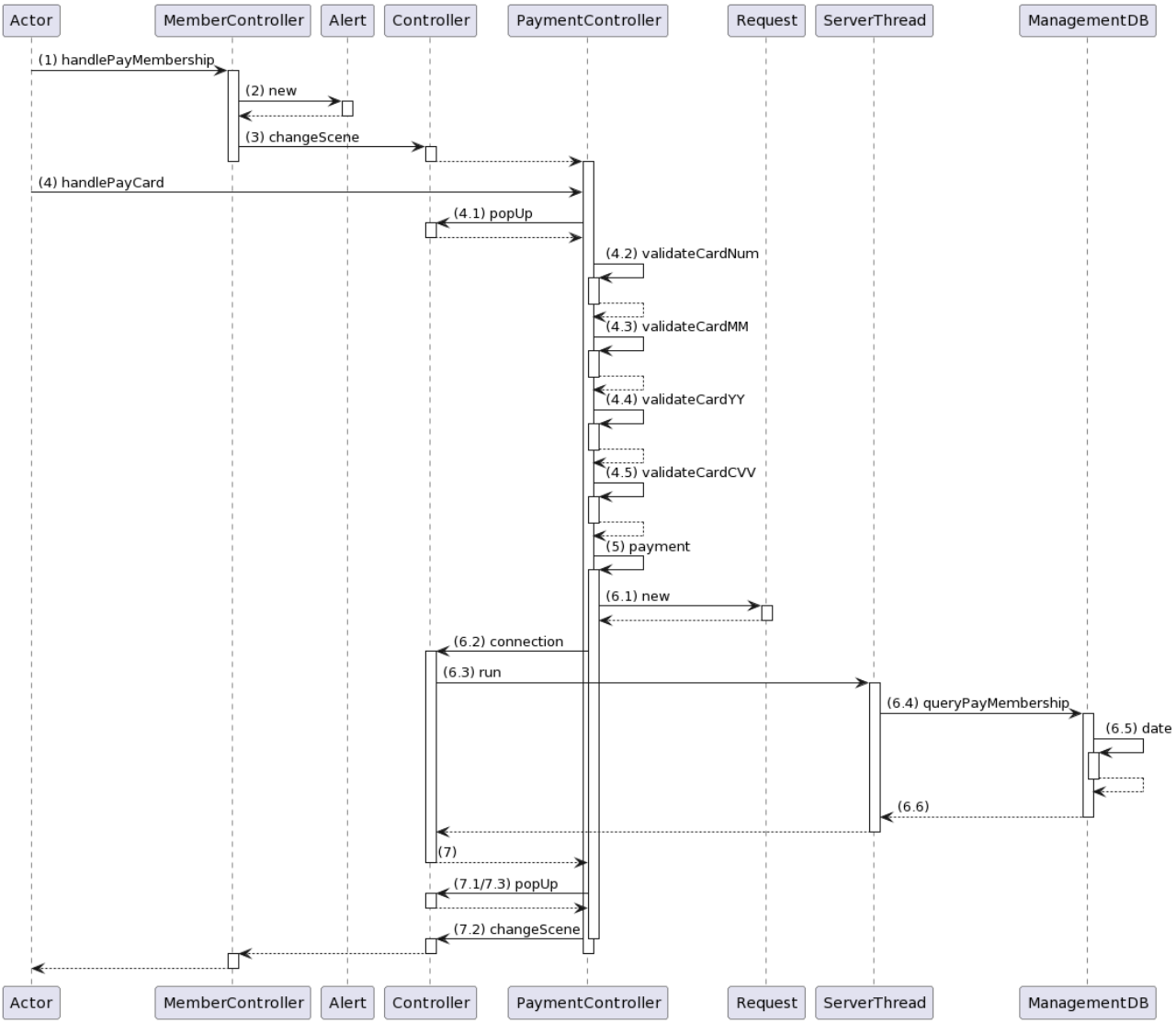
6.6) il metodo date ritorna al metodo queryPayMembership la data corrente e in seguito il metodo queryPayMembership si connette al database e memorizza i dettagli di pagamento.

7) A questo punto il metodo payment riceve un oggetto di risposta.
IF: se l'oggetto di risposta è di tipo ResponseType.Success.

7.1) Il metodo payment chiama il metodo popUp che mostra all'attore un avviso dell'effettivo riscontro positivo del pagamento.

7.2) Infine, il metodo payment chiama il metodo changeScene per passare dalla dashboard del pagamento alla dashboard privata dell'attore.

7.3) IF: se l'oggetto di risposta è diverso dal tipo ResponseType.Success.
Il metodo payment chiama il metodo popUp che mostra all'attore un avviso, il quale spiega che il pagamento è fallito.


Actor
MemberController
Alert
Controller
PaymentController
Request
ServerThread
ManagementDB
(1) handlePayMembership
(2) new
(3) changeScene
(4) handlePayCard
(4.1) popUp
(4.2) validateCardNum
(4.3) validateCardMM
(4.4) validateCardYY
(4.5) validateCardCVV
(5) payment
(6.1) new
(6.2) connection
(6.3) run
(6.4) queryPayMembership
(6.5) date
(6.6)
(7)
(7.1/7.3) popUp
(7.2) changeScene
Actor
MemberController
Alert
Controller
PaymentController
Request
ServerThread
ManagementDB

- Registration for a race

1) L'attore cioè il Member attiva il metodo handleSubscribe presente nella classe MemberController tramite il pulsante "Subscribe" presente nella GUI.

2) Il metodo handleSubscribe effettua dei controlli tramite dei if.

2.1) IF: il campo "ID race" è vuoto o non è stata scelta una barca il sistema tramite il metodo popUp mostra all'attore un avviso descrivendo l'errore.

2.2) IF: il campo "ID race" è diverso da un numero intero il sistema tramite il metodo popUp mostra all'attore un avviso descrivendo l'errore.

2.3) IF: la barca inserita è gia iscritta alla gara scelta, il sistema tramite il metodo popUp mostra all'attore un avviso descrivendo l'errore.

3) Il metodo handleSubscribe crea un nuovo oggetto di tipo Request, per verificare se l'ID della gara inserito dall'attore esista.

4) Il metodo handleSubscribe chiama il metodo connection passando come parametro l'oggetto Request.

5) Il metodo connection stabilisce una connessione con la classe SailingClubServer (che contiene una porta per creare una server socket e un metodo run() per far gestire le varie richieste dei client dalla classe ServerThread che contiene un metodo run()) e a sua volta trasferisce l'oggetto di tipo Request tramite un ObjectOutputStream.

6) Il metodo run presente nella classe ServerThread legge l'oggetto input ricevuto dal client, verifica il tipo di Request ricevuto e a sua volta utilizza il metodo queryVerifyRace passando come parametro l'ID della gara.

7) Il metodo queryVerifyRace effettua la connessione al database tramite il metodo connectionDB e verifica se esiste l'id della gara.

8) Il metodo queryVerifyRace ritorna una risposta al Server che a sua volta trasferisce al metodo handleSubscribe.

9) il metodo handleSubscribe verifica la risposta ricevuta.

9.1) IF: se la risposta è di tipo ResponseType.not_found, il metodo handleSubscribe chiama il metodo popUp che mostra all'attore un avviso descrivendo l'errore.

9.2) IF: se la risposta non è negativa, il metodo handleSubscribe chiama il metodo popUp che mostra all'attore un avviso descrivendo l'informazione che si sta procedendo al pagamento della tassa di iscrizione.

10) Il metodo handleSubscribe chiama il metodo changeScene per passare alla dashboard dei pagamenti.

11) L'attore attiva il metodo handlePayCard presente nella classe PaymentController tramite il pulsante "Pay Now" presente nella GUI.

11.1) IF: i campi sono vuoti il sistema mostra all'attore un Alert tramite il metodo popUp avvisando l'attore che ha dimenticato di compilare alcuni campi obbligatori.

11.2) Il metodo handlePayCard chiama il metodo validateCardNum per verificare se il numero della carta di credito inserito dall'attore è valido.

11.3) Il metodo handlePayCard chiama il metodo validateCardMM per verificare se il mese inserito dall'attore è valido.

11.4) Il metodo handlePayCard chiama il metodo validateCardYY per verificare se l'anno inserito dall'attore è valido.

11.5) Il metodo handlePayCard chiama il metodo validateCardCVV per verificare se il codice CVV inserito dall'attore è valido.

12) Successivamente, il metodo handlePayCard chiama il metodo payment presente su PaymentController.

13) il metodo payment deve controllare il tipo di pagamento che sta effettuando l'attore, il controllo è possibile tramite un if che verifica il tipo di pagamento tramite un oggetto di tipo RequestType che può assumere i valori di PAY_STORAGE, PAY_SUBSCRIBE e PAY_MEMBERSHIP.

13.1) IF: la RequestType è di tipo PAY_SUBSCRIBE, il metodo payment crea un nuovo oggetto di tipo Request passando come parametri lo username dell'attore, l'id della barca, l'id della gara e il tipo di pagamento.

13.2) A sua volta chiama il metodo connection passando come parametro l'oggetto Request creato pocanzi.

13.3) Il metodo connection attiva una connessione con il ServerThread e traferisce l'oggetto Request.

13.4) La classe ServerThread tramite il metodo run legge l'oggetto di tipo Request ricevuto, verifica il tipo di richiesta e chiama il metodo queryAddParticipation.

13.5) il metodo queryPayMembership presente nella classe ManagementDB, chiama il metodo date.

13.6) il metodo date ritorna al metodo queryPayMembership la data corrente e in seguito il metodo queryPayMembership si connette al database e memorizza i dettagli di pagamento.

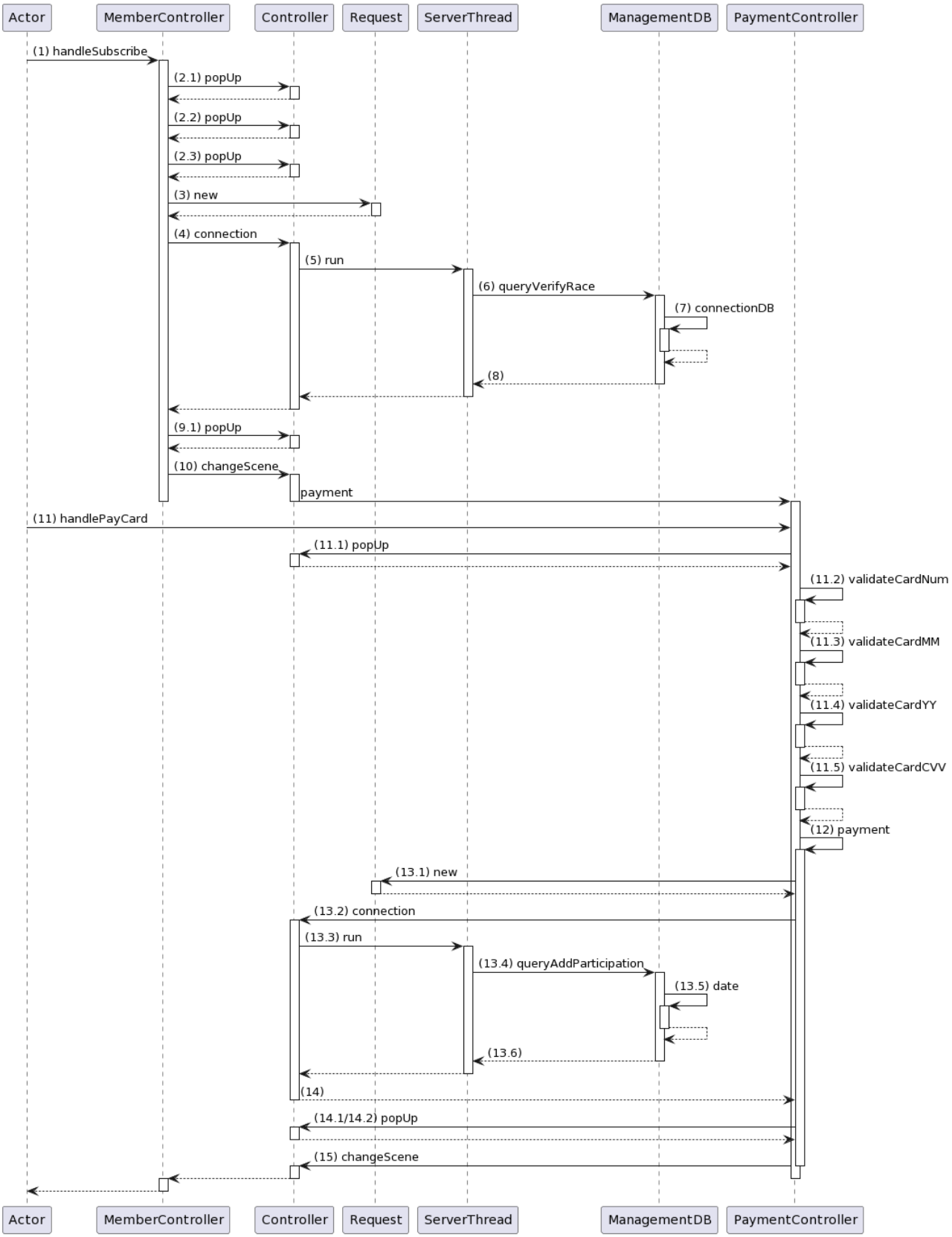
14) A questo punto il metodo payment riceve un oggetto di risposta.

IF: se l'oggetto di risposta è di tipo ResponseType.Success.

14.1) Il metodo payment chiama il metodo popUp che mostra all'attore un avviso dell'effettivo riscontro positivo del pagamento.

14.2) IF: se l'oggetto di risposta è diverso dal tipo ResponseType.Success. Il metodo payment chiama il metodo popUp che mostra all'attore un avviso, il quale spiega che il pagamento è fallito.

15) Infine, il metodo payment chiama il metodo changeScene per passare dalla dashboard del pagamento alla dashboard privata dell'attore.


Actor
MemberController
Controller
Request
ServerThread
ManagementDB
PaymentController
(1) handleSubscribe
(2.1) popUp
(2.2) popUp
(2.3) popUp
(3) new
(4) connection
(5) run
(6) queryVerifyRace
(7) connectionDB
(8)
(9.1) popUp
(10) changeScene
payment
(11) handlePayCard
(11.1) popUp
(11.2) validateCardNum
(11.3) validateCardMM
(11.4) validateCardYY
(11.5) validateCardCVV
(12) payment
(13.1) new
(13.2) connection
(13.3) run
(13.4) queryAddParticipation
(13.5) date
(13.6)
(14)
(14.1/14.2) popUp
(15) changeScene
Actor
MemberController
Controller
Request
ServerThread
ManagementDB
PaymentController