

USE CASE DIAGRAM

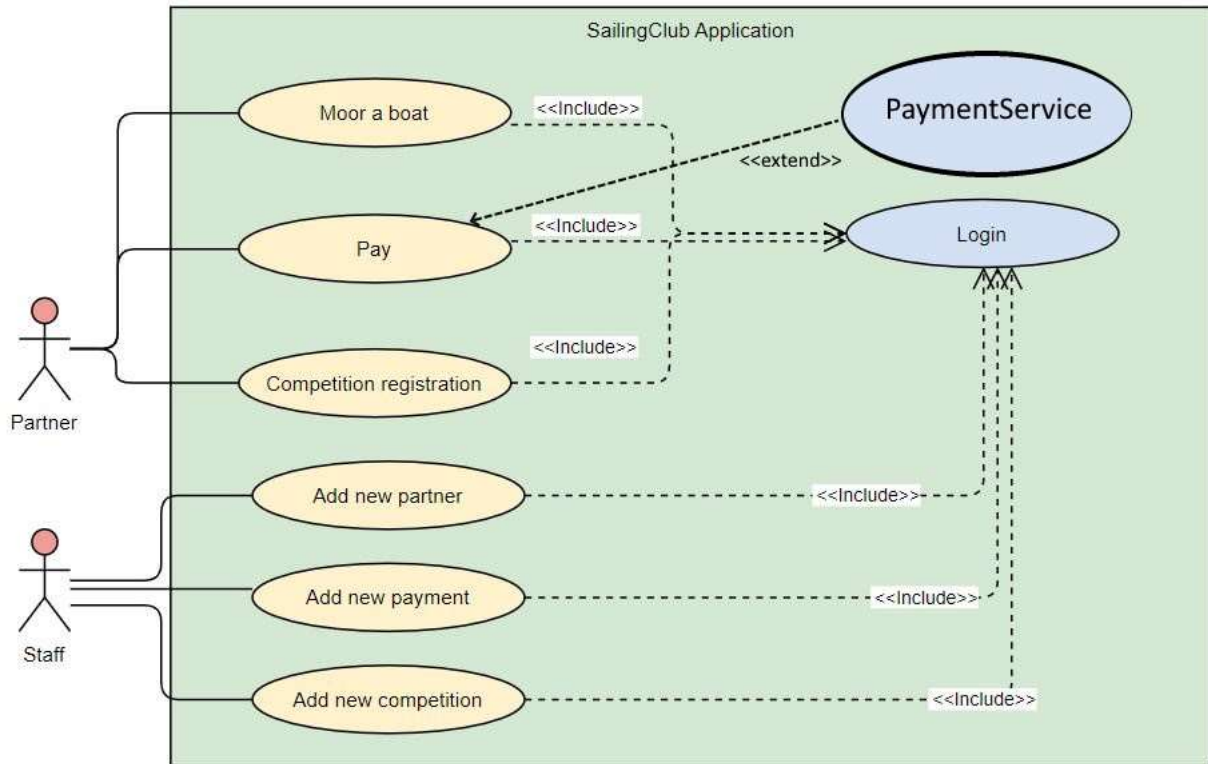
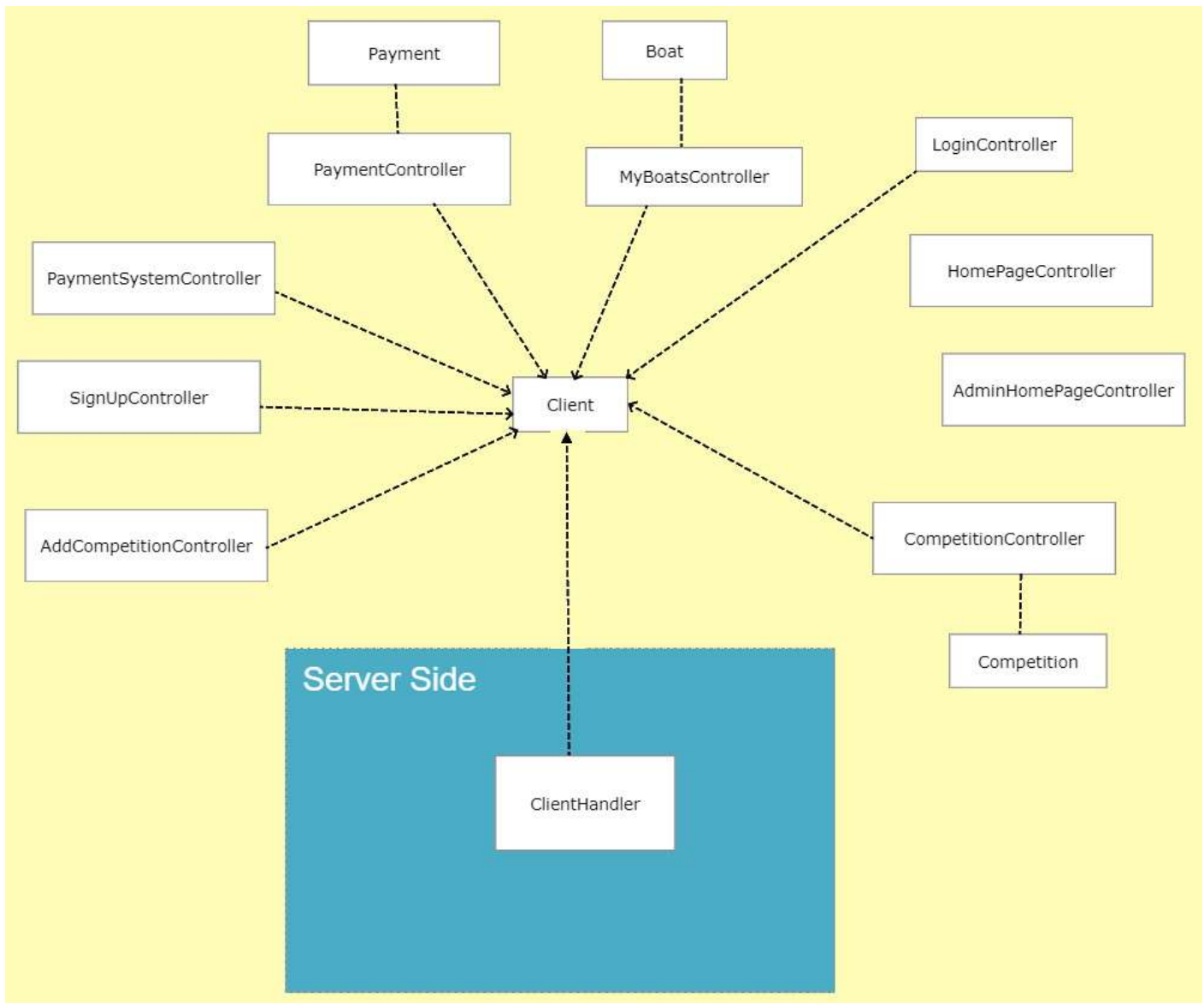


DIAGRAM CLASS



Nelle pagine successive sono descritte le principali classi e il loro utilizzo nell'applicazione.


CLIENT CLASS

Client è l'interfaccia che usano i controller per inviare le richieste al Server. Ho voluto usare un'unica classe per inviare le richieste per aggiungere uno strato di astrazione in modo che se ci fossero modifiche lato server dovrei modificare solamente Client e non tutti i controller.

Quando nel resto del documento verrà citato il server darò per scontato il passaggio attraverso la classe client.

Client
<pre>~ objectInputStream : ObjectInputStream ~ dataOutputStream : DataOutputStream ~ bufferedReader : BufferedReader ~ inputStreamReader : InputStreamReader - socket : Socket - host : String - port : int</pre>
<pre>~ Client(address : String, port : int) ~ Client() + ClientInit() : void + clientSendString(socketMsg : String) : int + clientLogin(user : String, psu : String) : int + clientUserRegistration(name : String, surname : String, psu : String, cf : String, address : String) : int + getBoatsList(CF : String) : ObservableList<Boat> + getPaymentList(CF : String) : ObservableList<Payment> + getCompetitionList(CF : String) : ObservableList<Competition> + pay(idPayment : int) : int + addBoat(CF : String, name : String, length : int) : int + raceRegistration(CF : String, idCompetition : int) : int + addPayment(CF : String, amount : Float, deadLine : String, description : String) : int + addCompetition(name : String, amount : Float, date : String) : int + finish() : void + getBufferedReader() : BufferedReader + getDataOutputStream() : DataOutput</pre>

LOGIN PAGE



Codice Fiscale

Password

Login

La pagina di login viene gestita dalla classe LoginController che si occupa di inviare la richiesta di autenticazione al Server. E' possibile autenticarsi come socio o come admin. A seconda del tipo di autenticazione cambiano le azioni permesse.

LoginController

```
- pswTextField : PasswordField
- usrTextField : TextField
- errorMessage : Text
- clientInitialize : Boolean
- client : Client
- root : Parent
- scene : Scene
- stage : Stage
```

```
+ initialize(url : URL, resourceBundle : ResourceBundle) : void
# LoginButtonPressed(event : ActionEvent) : void
```

HOME PAGE



Se il client viene autenticato come socio (utente normale) si arriverà alla home page che è gestita dalla classe `HomeController`. Gestisce semplicemente i 3 “bottoni-scritta” presenti nella schermata (anche il log out).

```
HomeController
```

```
- userCF : String  
- root : Parent  
- scene : Scene  
- stage : Stage
```

```
# LogoutButtonPressed(event : MouseEvent) : void  
# PaymentTextPressed(event : MouseEvent) : void  
# MyBoatTextPressed(event : MouseEvent) : void  
# CompetitionTextPressed(event : MouseEvent) : void  
+ getController() : HomeController  
+ setUserCF(userCF : String) : void
```

[illegible]

```
MyBoatsController

- moorButton : Button
- lengthTextField : TextField
- boatNameTextField : TextField
- length : TableColumn<Boat, Float>
- boatName : TableColumn<Boat, String>
- id : TableColumn<Boat, Integer>
- boatTable : TableView<Boat>
~ list : ObservableList<Boat>
- client : Client
- userCF : String
- root : Parent
- scene : Scene
- stage : Stage

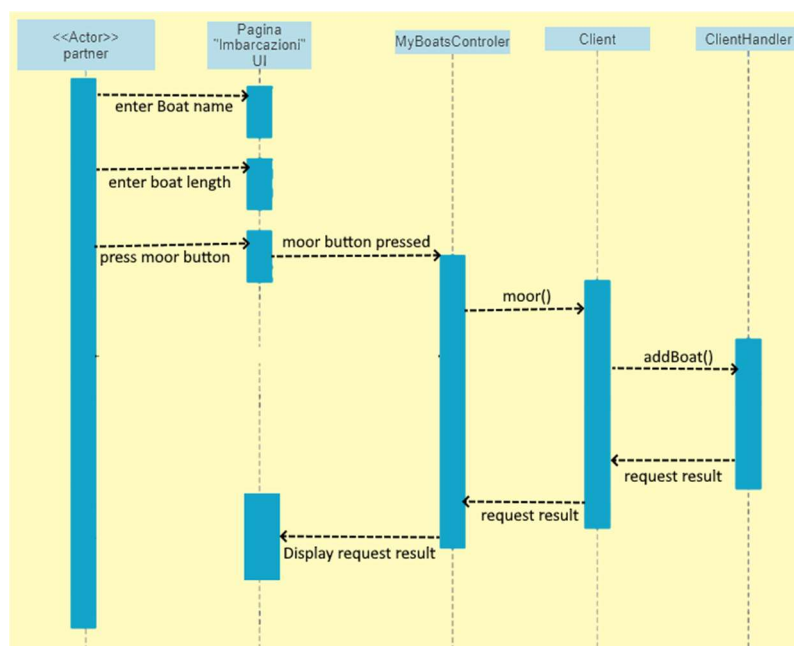
+ setUserCF(userCF : String) : void
+ getUserCF() : String
+ getController() : MyBoatsController
# moor() : void
+ fillList() : void
+ initialize(url : URL, rb : ResourceBundle) : void
# returnToHomePage(event : MouseEvent) : void
```

Una barca è rappresentata dalla classe Boat

Boat
<pre>- length : int - boatName : String - id : Integer</pre>
<pre>+ setLength(length : int) : void + getLength() : int + setBoatName(boatName : String) : void + getBoatName() : String + setId(id : Integer) : void + getId() : Integer ~ Boat(id : int, name : String, length : int)</pre>

L'elenco delle barche viene ottenuto dopo una richiesta al server che a sua volta interroga il database e fornisce il risultato della richiesta inviandolo attraverso socket.

Di seguito il sequence diagram che rappresenta l'operazione di aggiunta di una nuova barca.



Sequence Diagram 1: aggiunta di una nuova barca

[illegible]

```
PaymentController

~ list : ObservableList<Payment>
- amount : TableColumn<Payment, Float>
- state : TableColumn<Payment, CheckBox>
- paymentDate : TableColumn<Payment, String>
- deadLine : TableColumn<Payment, String>
- description : TableColumn<Payment, String>
- paymentTable : TableView<Payment>
- client : Client {readOnly}
- idPayment : int
- userCF : String
- root : Parent
- scene : Scene
- stage : Stage

# returnToHomePage(event : MouseEvent) : void
+ initialize(url : URL, rb : ResourceBundle) : void
# pay(event : ActionEvent) : void
+ setUserCF(userCF : String) : void
+ getController() : PaymentController
+ fillList() : void
```


Il singolo pagamento è rappresentato dalla classe payment

Payment
<pre>- state : String - amount : Float - paymentDate : String - deadLine : String - description : String - idPayment : int</pre>
<pre>+ setState(state : String) : void + getState() : String + setAmount(amount : Float) : void + getAmount() : Float + setPaymentDate(paymentDate : String) : void + getPaymentDate() : String + setDeadLine(deadLine : String) : void + getDeadLine() : String + getDescription() : String + setDescription(description : String) : void + getIdPayment() : int + Payment(idPayment : int, description : String, deadLine : String, paymentDate : String, amount : Float, state : Boolean)</pre>

Una volta selezionato un pagamento e premuto il tasto paga si verrà reindirizzati alla pagina del pagamento dove ci sarà la possibilità di pagare con carta o caricando un PDF della ricevuta di bonifico.



Sailing Club

Proprietario

Data di scadenza

Codice di sicurezza

Paga

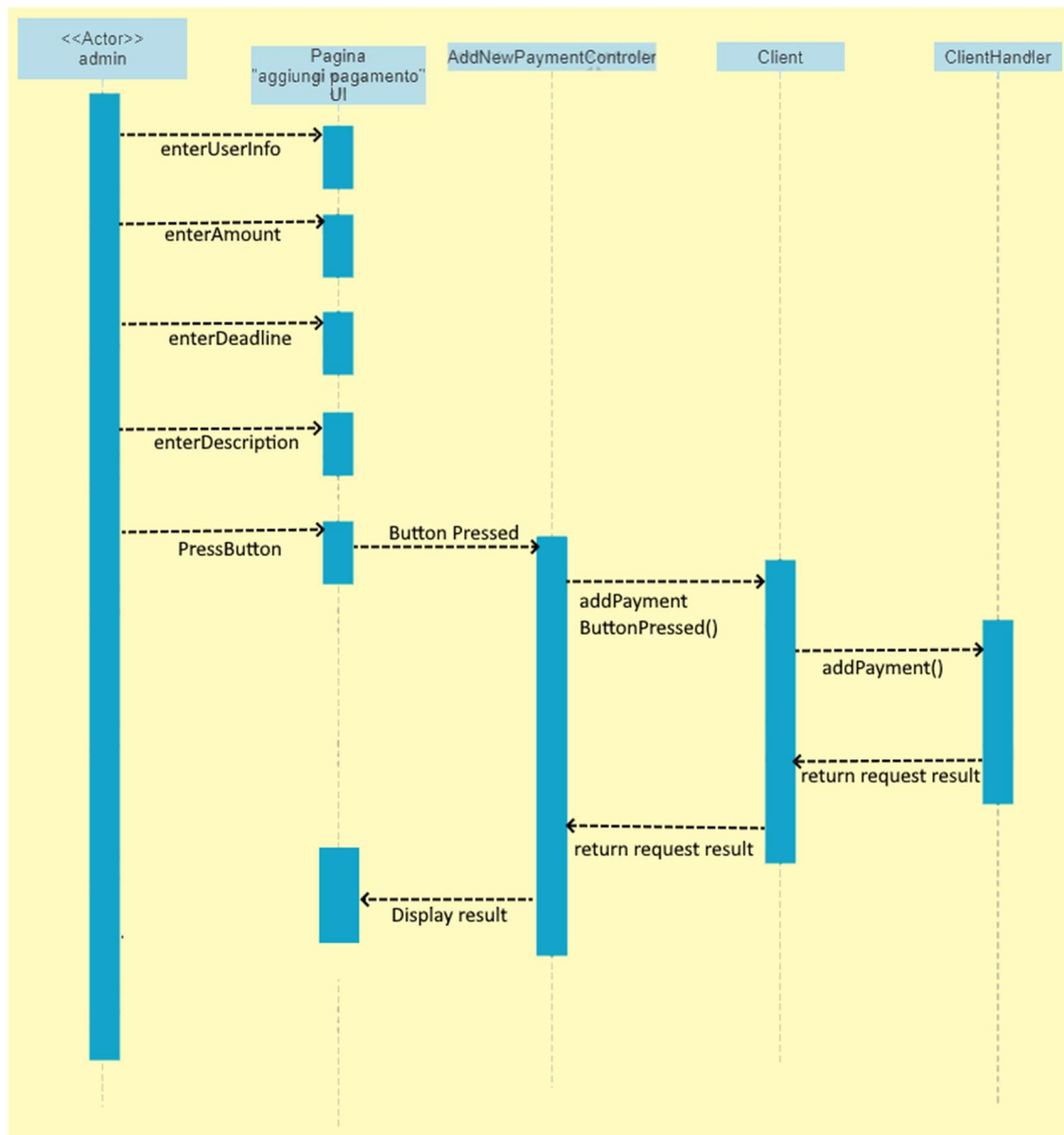
Ricevuta Bonifico

[Torna indietro](#)

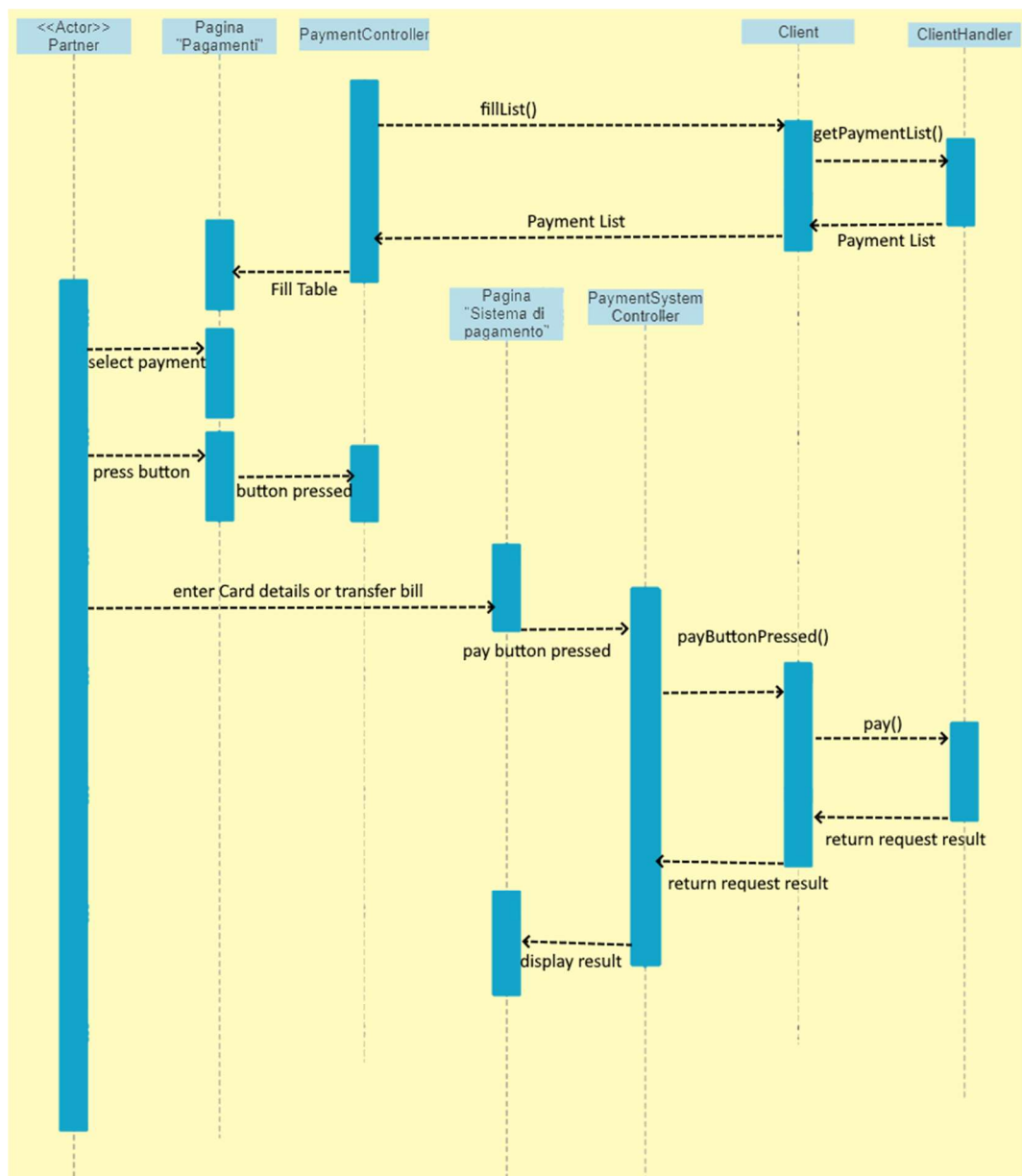
Il sistema di pagamento è gestito dalla classe `SystemPaymentsController`.

<code>PaymentSystemController</code>
<pre>~ logMessage : Text ~ cvc : TextField ~ deadLineCard : TextField ~ cardOwner : TextField ~ cardNumber : TextField - idPayment : int - client : Client - userCF : String - root : Parent - scene : Scene - stage : Stage</pre>
<pre>+ setUserCF(userCF : String) : void + setIdPayment(idPayment : int) : void + getController() : PaymentSystemController # vatButtonPressed(e : ActionEvent) : void - displayLogMessage(result : int) : void # payButtonPressed(e : ActionEvent) : void # backButtonPressed(event : MouseEvent) : void + initialize(url : URL, resourceBundle : ResourceBundle) : void</pre>

La notifica di pagamento della quota annuale viene inserita dall'admin che aggiunge il pagamento attraverso l'apposita pagina mostrata più avanti. Di seguito i Sequence diagram che mostrano rispettivamente l'aggiunta del pagamento da parte dell'admin e il pagamento da parte del socio.



Sequence Diagram 2: Aggiunta del nuovo pagamento da parte dell'admin



Sequence Diagram 3: Operazione del pagamento del client

COMPETIZIONI

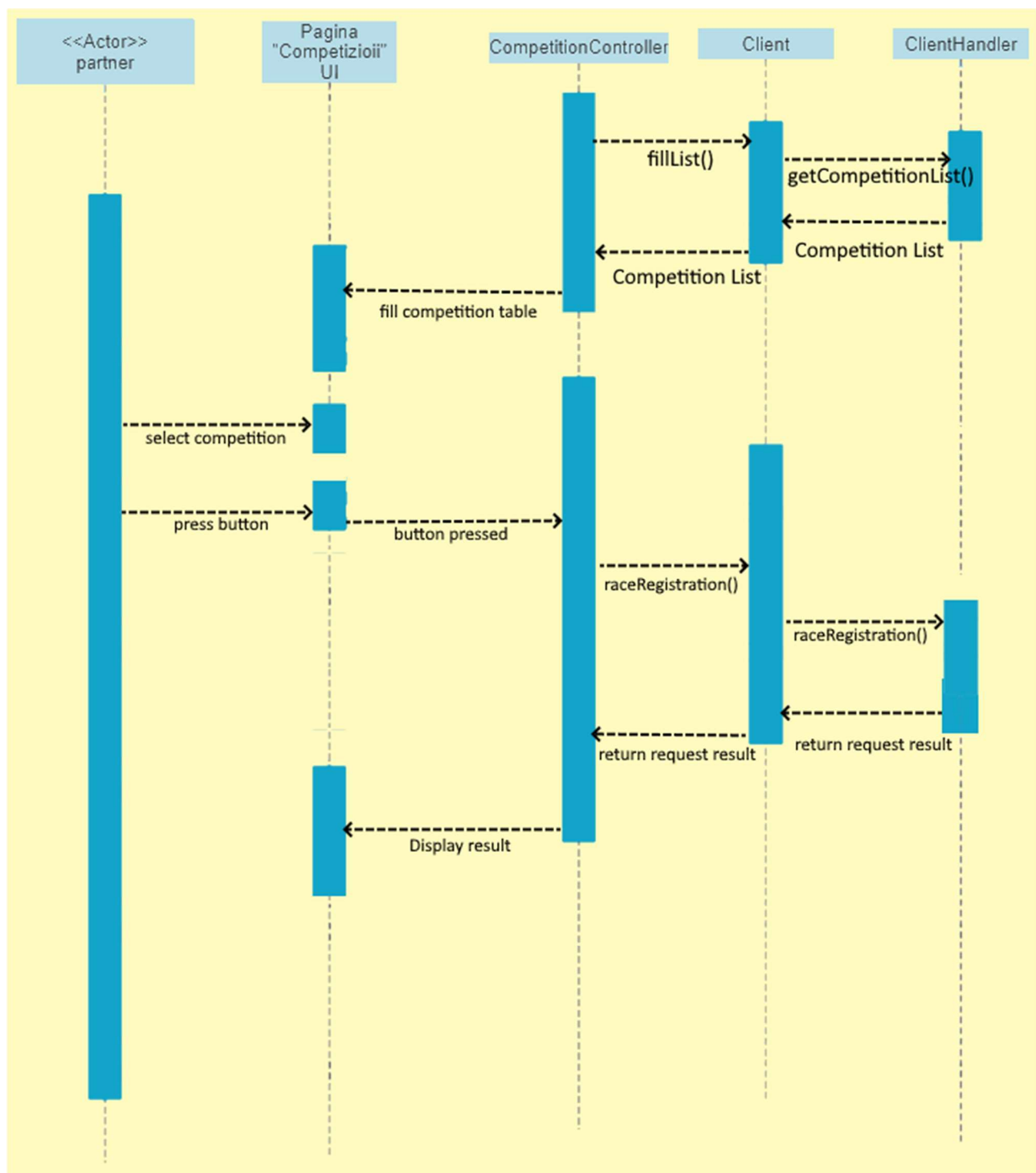
ISCRIVITI

CompetitionController

```
- client : Client {readOnly}
~ list : ObservableList<Competition>
- state : TableColumn<Competition, String>
- price : TableColumn<Competition, Float>
- name : TableColumn<Competition, String>
- idCompetition : TableColumn<Competition, Integer>
- competitionTable : TableView<Competition>
- userCF : String

+ getController() : CompetitionController
+ setUserCF(userCF : String) : void
# returnToHomePage(event : MouseEvent) : void
# raceRegistration() : void
+ fillList() : void
+ initialize(url : URL, resourceBundle : ResourceBundle) : void
```

Di seguito viene illustrato il procedimento di iscrizione a nuova gara attraverso un sequence diagram.



Sequence Diagram 4: Iscrizione ad una gara

ADMIN HOME PAGE

Se l'utente viene autenticato come admin si presenterà la seguente schermata.



Il controller gestisce le interazioni con il menu.

```
AdminHomeController
```

```
- root : Parent  
- scene : Scene  
- stage : Stage
```

```
+ getController() : AdminHomeController  
# logOutButtonPressed(event : MouseEvent) : void  
# addCompetitionTextPressed(event : MouseEvent) : void  
# addPaymentTextPressed(event : MouseEvent) : void  
# addUserTextPressed(event : MouseEvent) : void
```

AGGIUNGI SOCIO



Registration form for Sailing Club:

-
-
-
-
-


[Registrati](#)

[Torna indietro](#)

In questa pagina si manda al server la richiesta di aggiungere un nuovo socio. Il controller della pagina è il seguente:

SignUpController
<pre>- errorMessage : Text - address : TextField - fiscalCode : TextField - password : TextField - surname : TextField - name : TextField - client : Client - stage : Stage - scene : Scene - root : Parent</pre>
<pre># SignUpButtonPressed(event : ActionEvent) : void # BackTextPressed(event : MouseEvent) : void</pre>

AGGIUNGI PAGAMENTO



Form for adding a payment:

- Empty text field for client name
- Text field labeled "Quantita da pagare"
- Date picker labeled "Scadenza pagamento"
- Text area labeled "Descrizione pagamento"
- Blue button labeled "Aggiungi"
- Link labeled "Torna indietro"


L'admin può aggiungere un pagamento a carico di un partner(principalmente iscrizione annuale).

```
AddPaymentController
```

```
~ errorMessage : Text
~ description : TextArea
~ deadLine : DatePicker
~ amount : TextField
~ fiscalCode : TextField
- client : Client
- stage : Stage
- scene : Scene
- root : Parent
```

```
# addPaymentButtonPressed(event : ActionEvent) : void
# BackTextPressed(event : MouseEvent) : void
```

CREA COMPETIZIONE



A screenshot of a web form titled "CREA COMPETIZIONE" (Create Competition) for the "Sailing Club". The form is set against a yellow background. It includes three input fields: a text field for the competition name, a text field for "Prezzo iscrizione" (Registration fee), and a date picker for "Data gara" (Race date). Below these fields is a large blue button labeled "Crea" (Create). At the bottom, there is a link labeled "Torna indietro" (Go back).

L'admin può creare una competizione attraverso questa pagina. Il controller è AddCompetitionController.

```
AddCompetitionController
```

```
~ errorMessage : Text
~ competitionDate : DatePicker
~ competitionPrice : TextField
~ competitionName : TextField
- client : Client
- stage : Stage
- scene : Scene
- root : Parent
```

```
# BackTextPressed(event : MouseEvent) : void
# createButtonPressed(event : ActionEvent) : void
```