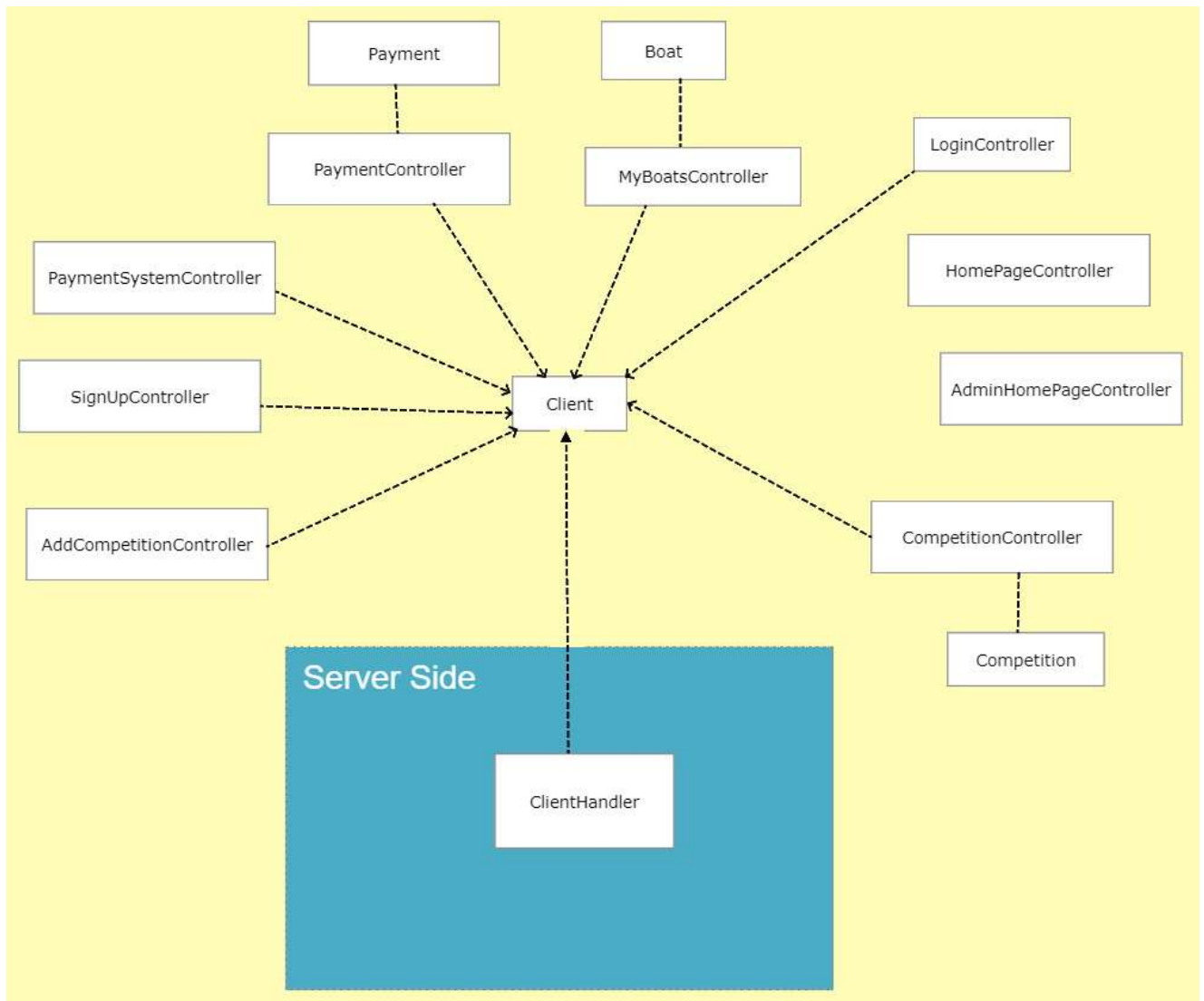


DIAGRAM CLASS



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

Sopra la pagina di login che viene gestita dalla classe LoginController che si occupa di inviare la richiesta di autenticazione al Server (sempre passando per la classe Client):

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 4 “bottoni-scritta” presenti nella schermata.

<code>HomeController</code>
<code>- userCF : String</code> <code>- root : Parent</code> <code>- scene : Scene</code> <code>- stage : Stage</code>
<code># LogoutButtonPressed(event : MouseEvent) : void</code> <code># PaymentTextPressed(event : MouseEvent) : void</code> <code># MyBoatTextPressed(event : MouseEvent) : void</code> <code># CompetitionTextPressed(event : MouseEvent) : void</code> <code>+ getController() : HomeController</code> <code>+ setUserCF(userCF : String) : void</code>

[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
```

```
- 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
<ul style="list-style-type: none">- length : int- boatName : String- id : Integer
<ul style="list-style-type: none">+ 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)

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.

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

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.



Proprietario

Data di scadenza

Codice di sicurezza

Paga

Ricevuta Bonifico

Torna indietro

Il sistema di pagamento è gestito dalla classe
SystemPaymentsController.

PaymentSystemController
<pre>~ logMessage : Text ~ cvc : TextField ~ deadLineCard : TextField ~ cardOwner : TextField ~ cardNumber : TextField - idPayment : int - client : Client - userCF : String - root : Parent - scene : Scene - stage : Stage + 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>

COMPETIZIONI

[illegible]

ISCRIVITI

In questa pagina viene mostrato l'elenco di gare in programma e tutte le relative informazioni, e la possibilità di iscriversi a una di queste. Il tutto gestito dalla classe `CompetitionController`.

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
```

ADMIN HOME PAGE

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




Il controller della pagina è il seguente:

```
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:

-
-
-
-
-

[Torna indietro](#)


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

```
SignUpController
```

```
- errorMessage : Text  
- address : TextField  
- fiscalCode : TextField  
- password : TextField  
- surname : TextField  
- name : TextField  
- client : Client  
- stage : Stage  
- scene : Scene  
- root : Parent
```

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

AGGIUNGI PAGAMENTO



Form for adding a payment:

- Empty text field
- Quantita da pagare
- Scadenza pagamento
- Descrizione pagamento
- Aggiungi button
- Torna indietro link


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



Form for creating a competition:

- Text input field for competition name
- Text input field for "Prezzo iscrizione" (Registration fee)
- Date picker for "Data gara" (Race date)
- Blue "Crea" (Create) button
- Link "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
```