Politecnico di Milano

Design and Implementation of Mobile Applications

iSport

Design Document

di
Gianluigi Oliva

February 10, 2019

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe the design and prototyping phases used for the realization of the "iSport" mobile application. In detail, the main components, features and user experience will be discussed.

The main aim of iSport application is visualizing information and data related to the sport field. In particular, we will focus on the most relevant daily news, also showing a section with all football final scores. Moreover, to encourage interaction throughout a community discussion, the application will provide a system of live chat to exchange opinions.

This project is the result of the implementation of the knowledge acquired during the course "Design and Implementation of Mobile Applications" provided by the Milan Polytechnic.

1.2 Intended Audience

This document is produced for those who develop, evaluate and use iSport mobile application:

- The engineers who had the idea and developed the application.
- The testers that must verify the effective implementation of all the described components and functions.
- The user who will use the application and take advantage of its functionalities.
- The future contributors who wish to develop new features.

1.3 Definitions, acronyms, abbreviations

Definitions

- **Platform**: The application as a whole.
- **Guest**: A person who can view public contents
- **User**: A guest who already performed the login operation successfully.

- **Match**: A match between two teams that has already occurred or is in progress
- **Framework**: Reusable set of libraries or classes for a software system.
- **News**: A news related to the world of sport present in some journalism
- **Forecast**: A prediction on the football scores among a class of possible results
- **Odds**: The remuneration value of a prediction relative to a given match
- **REST**: is a way of providing interoperability between computer systems on the Internet.

Acronyms

- **MVC**: Model - View - Controller
- **HTTPS**: HyperText Transfer Protocol Secure
- **IDE**: Integrated Development Environment
- **API**: Application Programming Interface
- **JSON**: JavaScript Object Notation
- **UML**: Unified Modelling Language.
- **UX**: User Experience
- **URL**: Uniform Resource Locator

Abbreviations

- **App**: Mobile Application

1.4 Mobile Application Scope

iSport has been developed for those who love sports, with the aim to unify under one application all the services on the market. In this way we want to give an ongoing service to the end user, without having to browse multiple applications to achieve the same result.
In particular, the application will be divided into three screens:

- **News**
- **Live**
- **Bet**

- **Chat**

In the "News" section there will be the daily sport news displayed with a preview image and a small description. Moreover, by pressing on the single news you can read the complete article.
In the "Live" section there will be all current matches with the final scores if already completed or the current one if still in progress. Pressing on the single game, the user will consult all the related information such as the markers and goal time, cards, training and statistics.
In the "Bet" section there will be the shares related to the daily football matches. By pressing on the single one the user will bet on the winning game composing a ticket; once the process is completed the application will calculate the potential winnings based the bet amount.
In the "Chat" section you will connect to a global room where you can talk to other users who are using the application and exchange comments and opinions.

1.5 Framework

The development of iSport was achieved through the use of native iOS SDKs, in particular by using the Swift programming language. This choice allowed greater control of system resources and access to system services, otherwise not possible if using cross-platform frameworks such as PhoneGap or React Native. The purpose is to implement different functionalities and integration with other sites.

1.6 Functional Requirements

The product provides to users a simple and user-friendly interface to:

1. View news previews
2. Read the complete article
3. View football match results real time
4. Display goal-scores
5. Display booking (caution)
6. Display team playing the match
7. Display statistics
8. Display game share
9. Compose your ticket

10. Display the potential winnings of the ticket
11. Share news on Facebook
12. Save news into favorites section
13. Chat with other users
14. Log in

1.7 Non Functional Requirements

The application must be able to:

- Run both on phone and tablet (only if they have an iOS).
- Work without requiring user sensitive data and services that may require a user cost (such as calls or SMS).
- Occupy the entire screen available.
- Keep preferences and status at every start.

1.8 Assumptions, Dependencies and Constraints

Costraints

- **Hardware limitations**: our application runs on every mobile device like smartphones and tablets. Therefore, as the App consumes a low amount of RAM, the only hardware constraint for the users is to have a mid-range device. (for instance iPhone 5 or better).
- **Parallel operations**: the application must be able to handle multiple parallel requests with high reactivity.

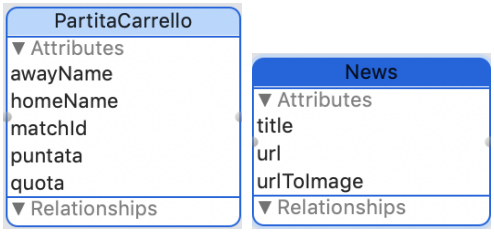
Assumptions and Dependecies

- **Internet Connection**: the device used by the users dispose of an internet connection and a sufficient bandwidth to use the application.
- **No privileged users**: there are no priviled users or administrators with particular functions.
- **No user connections**: every user is independent from the others.
- **API availability**: the API provided by third part's services are always available.
- **OS Permission Granted**: the user will always grant to his OS's device the permission to access to all the needed services.

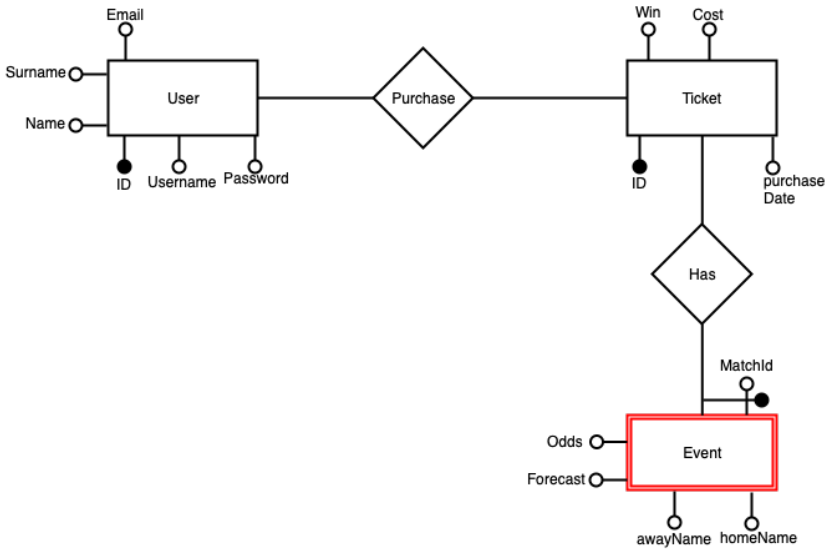
2 Architecture

2.1 Database

Since the application receives all the necessary data from external services through the API, the only data that need to be saved are the games that make up a ticket. Moreover, since there is no interaction between the different users, the data is saved locally using the Core Data present in the iOS SDK.



In order to purchase the ticket, we used an online database through the Firebase platform. The E / R model used is the follow:



To guarantee privacy, the access to the Database is regulated by the following rules:

```
{
  "rules": {
    "Schedina": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

2.2 Client

For the implementation of the application we have chosen a mobile back-end, that is a client architecture. This choice was made mainly because the application does not interface with other users and because for various services it uses third-party APIs. Communication with third-party services is based on HTTPS REST requests, in particular through GET requests.
The client uses the traditional MVC pattern:

- Model: this package contains all the classes representing data to be shown to the single user, taken by the Controller and published by the View.
- View: this package contains all the components that display data to the user and interact with him.
- Controller: this package contains all the objects in charge to interact between one or more view objects of the application and one or more model objects.

3 Use case functional requirements analysis

This section describes how actors can interact with iSport in order to use all the features implemented in the app. The focus of this part is on the front-end and we show the operations that can be performed by the actors without taking care of the system architecture behind the app.

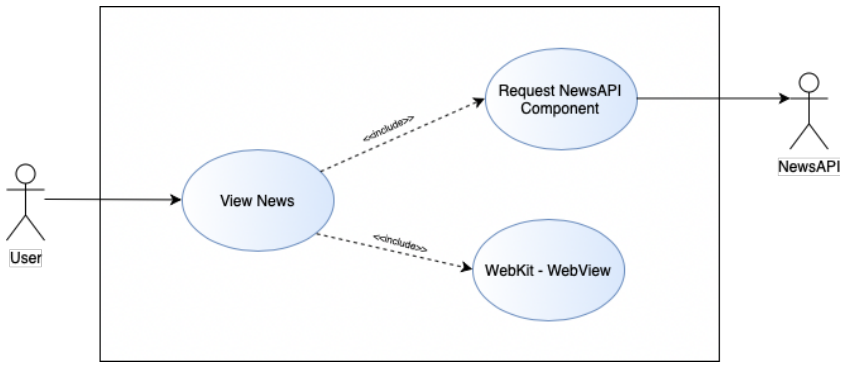


Figure 3.1: Use case related to visualization and handling news

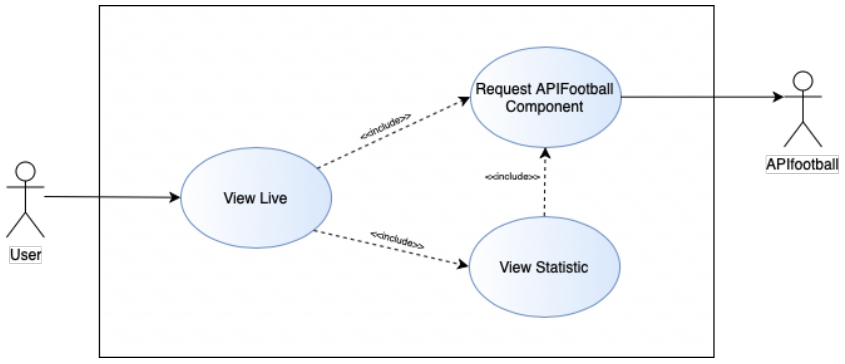


Figure 3.2: Use case related to visualization and handling results

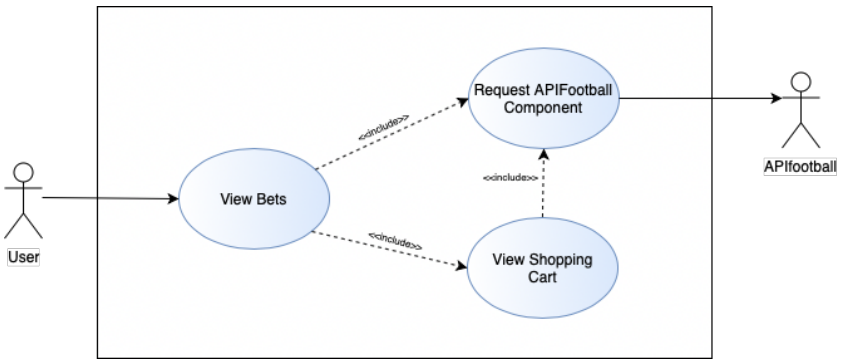


Figure 3.3: Use case related to visualization and handling odds

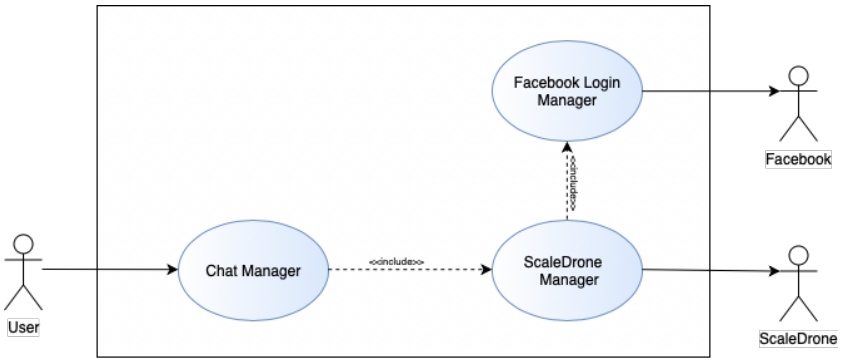


Figure 3.4: Use case related to chat service

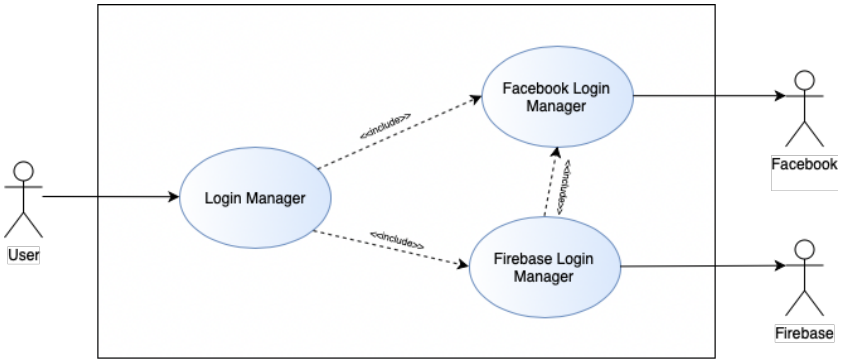


Figure 3.5: Use case related to log in activity

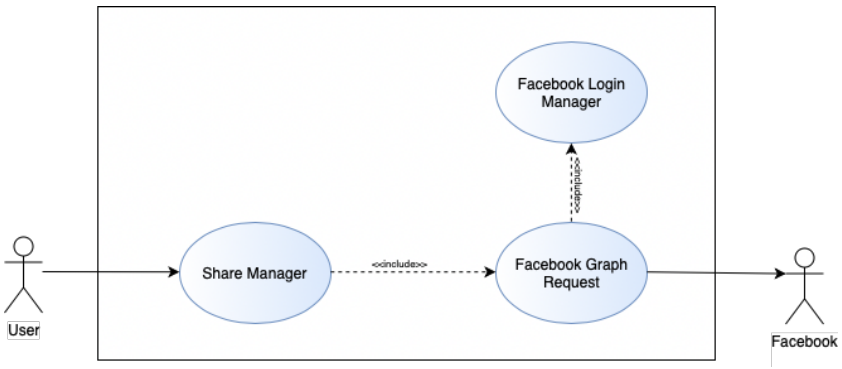


Figure 3.6: Use case related to news sharing

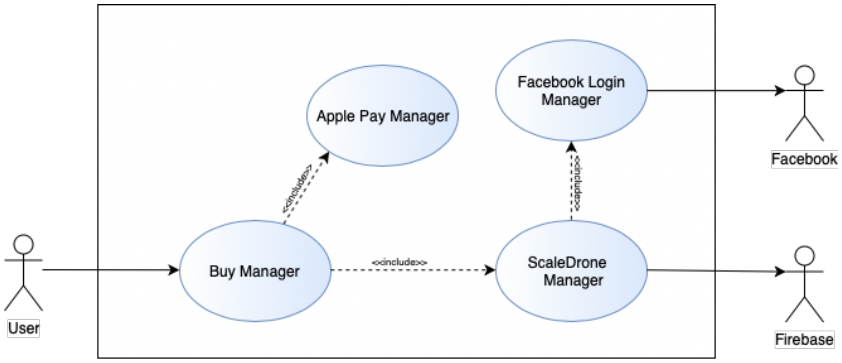


Figure 3.7: Use case related to ticket purchase

Login

Name	Login
Actor	Guest
Entry Condition	Guest with login credentials
Goal	14
Event Flow	• The user opens the application • The user presses "Login" located in the "Side Menu" • The user is redirected on Facebook to enter credentials • The app logs in through Facebook • The app login into Firebase using Facebook account
Exit condition	Actor becomes User
Exceptions	The user is not connected to the network or he hasn't a Facebook account.

Logout

Name	Logout
Actor	User
Entry Condition	Actor is logged in
Goal	14
Event Flow	• The user presses "Logout" located in the "Side Menu" • The app logout from Facebook • The app logout from Firebase
Exit condition	Actor is logged out and becomes Guest
Exceptions	The user is not connected to the network.

View News

Name	View News
Actor	Guest
Entry Condition	The user downloaded the mobile app
Goal	1, 2
Event Flow	• The user opens the app • The user presses on the "News" tab located to the "Side Menu" • The app shows a selection of the most important news • The user presses the news to read the whole article • The user presses "Done" to finish reading
Exit condition	The user read the news of interest.
Exceptions	The user is not connected to the network so he cannot send a request to read the article. The news has been released from the origin website, but not from the database.

Share News

Name	Share News
Actor	User
Entry Condition	The user logged in correctly
Goal	11
Event Flow	• The user opens the app • The user presses the "News" tab located in the "Side Menu" • The app shows a selection of the most important news • The user presses on the "Share" function • The user writes the message he wants to attach to the shared post • The user presses "Publish"
Exit condition	The user shares on Facebook the news of interest.
Exceptions	The user is not connected to the network so he cannot send a request to share the article. The news has been released from the origin website, but not from the database or the connection is weak.

Save News

Name	Save News
Actor	User
Entry Condition	The user downloaded the mobile app
Goal	12
Event Flow	• The user opens the app • The user presses the "News" tab located in the "Side Menu" • The app shows a selection of the most important news • The user presses on "Bookmark" to save the news • The app saves the information in the local database
Exit condition	The user saves the news for a following reading.
Exceptions	The user is not connected to the network, so he cannot send a request to save the article. The news has been released from the origin website, but not from the database.

See Saved News

Name	See Saved News
Actor	User
Entry Condition	The user saved the news
Goal	1, 2, 12
Event Flow	• The user opens the app • The user presses the "News" tab located in the "Side Menu" • The user presses on "Bookmark" in the Navigation Bar to see saved news • The app shows a list of the saved news
Exit condition	The user read the news of interest.
Exceptions	The user is not connected to the network so he cannot send a request to read the article. The news has been released from the origin website, but not from the database.

See Matches

Name	See Matches
Actor	Guest
Entry Condition	The user downloaded the app
Goal	3
Event Flow	• The user opens the app • The user presses the "Live" tab located in the "Side Menu" • The app shows a list of matches
Exit condition	The user saw scores.
Exceptions	The user is not connected to the network, so he cannot send a request to obtain the final scores.

View Lineup

Name	View Lineup
Actor	Guest
Entry Condition	The user downloaded the app
Goal	3, 6
Event Flow	• The user opens the app • The user presses the "Live" tab located in the "Side Menu" • The app shows a list of daily matches • The user chooses the match to obtain the requested information • The user presses the button showing the game field
Exit condition	The user saw lineups for the requested match
Exceptions	The user is not connected to the network, so he cannot send a request to obtain results or the connection could be weak.

View Goal Scorer

Name	View Goal Scorer
Actor	Guest
Entry Condition	The user downloaded the app
Goal	3, 4
Event Flow	• The user opens the app • The user presses the "Live" tab located in the "Side Menu" • The app shows a list of daily matches • The user chooses the match to obtain the requested information • The user presses the button showing a ball
Exit condition	The user saw the goal scorers of the requested match.
Exceptions	The user is not connected to the network, so he cannot send a request to obtain results or the connection could be weak.

View Match Statistics

Name	View Match Statistics
Actor	Guest
Entry Condition	The user downloaded the app
Goal	3, 7
Event Flow	• The user opens the app • The user presses the "Live" tab located in the "Side Menu" • The app shows a list of daily matches • The user chooses the match to obtain the requested information • The user presses the button showing a chart
Exit condition	The user saw the match statistics.
Exceptions	The user is not connected to the network, so he cannot send a request to obtain results or the connection could be weak.

View Match Booking (Cautions)

Name	View Match Booking (Cautions)
Actor	Guest
Entry Condition	The user downloaded the app
Goal	3, 5
Event Flow	• The user opens the app • The user presses the "Live" tab located in the "Side Menu" • The app shows a list of daily matches • The user chooses the match to obtain the requested information • The user presses the button showing a card
Exit condition	The user saw booked and expelled players.
Exceptions	The user is not connected to the network, so he cannot send a request to obtain results or the connection could be weak.

View Odds

Name	View Odds
Actor	Guest
Entry Condition	The user downloaded the app
Goal	8
Event Flow	• The user opens the app • The user presses the “Bet” tab located in the ”Side Menu” • The app shows a list of principle odds of the daily matches
Exit condition	The user saw the odds of the daily matches.
Exceptions	The user is not connected to the network, so he cannot send a request to obtain results or the connection could be weak.

View Ticket

Name	View Ticket
Actor	Guest
Entry Condition	The user downloaded the app
Goal	9
Event Flow	• The user opens the app • The user presses the “Bet” tab located in the ”Side Menu” • The user presses the button showing a cart
Exit condition	The user sees the ticket.
Exceptions	The user is not connected to the network, so he cannot send a request to obtain results or the connection could be weak.

Add Match To The Ticket

Name	Add Match To The Ticket
Actor	Guest
Entry Condition	The user downloaded the app
Goal	9
Event Flow	• The user opens the app • The user presses the “Bet” tab located in the ”Side Menu” • The app shows a list of principle odds of the daily matches • The user presses on the odd and add the forecast to the ticket
Exit condition	The user added a forecast to the ticket.
Exceptions	The user is not connected to the network, so he cannot send a request to obtain results or the connection could be weak.

Delete a Match

Name	Delete a Match
Actor	Guest
Entry Condition	The user downloaded the app
Goal	9
Event Flow	• The user opens the app • The user presses the “Bet” tab located in the ”Side Menu” • The user presses the button showing a cart • The user swipes to the left to delete the match
Exit condition	The user deleted a forecast from the ticket
Exceptions	None

Calculate The Potential Winning

Name	Calculate The Potential Winning
Actor	Guest
Entry Condition	The user downloaded the app
Goal	10
Event Flow	• The user opens the app • The user presses the “Bet” tab located in the ”Side Menu” • The user presses the button showing a cart • The user selects the amount of money he wants to bet • The user presses on “Done”
Exit condition	The user sees the potential winning
Exceptions	None

Purchase ticket

Name	Purchase ticket
Actor	User
Entry Condition	The user logged in correctly
Goal	9, 10
Event Flow	• The user opens the app • The user presses the “Bet” tab located in the ”Side Menu” • The user presses the button showing a cart • The user selects the amount of money he wants to bet • The user presses on ”Buy”
Exit condition	The user completes the purchase.
Exceptions	Absent connection to the network, the user cannot complete the purchase. The user didn’t log correctly or he didn’t add matches to the ticket.

View Purchased Tickets

Name	View Purchased Tickets
Actor	User
Entry Condition	The user logged in correctly
Goal	9, 10
Event Flow	• The user opens the app • The user presses the “Bet” tab located in the ”Side Menu” • The user presses on “chronology” in the NavBar • The app downloads the information about all purchased tickets • The user presses on the corresponding cell of the ticket he wants to see • The app loads the ticket and the user sees all the details
Exit condition	The user sees a purchased ticket previously.
Exceptions	The user is not connected to the network, so he cannot send a request to obtain results or the connection could be weak. The user didn’t log correctly or he didn’t buy a ticket.

Chat

Name	Chat
Actor	User
Entry Condition	The user logged in correctly
Goal	13
Event Flow	• The user opens the app • The user presses the “Chat” tab located in the ”Side Menu” • The app logs into Scaledrone’s room using Facebook credentials • The user writes messages in the text field • The user presses send • The app loads all the messages of the room
Exit condition	The user interacts with other people.
Exceptions	The user is not connected or he has not a Facebook account.

4 Sequence Diagram

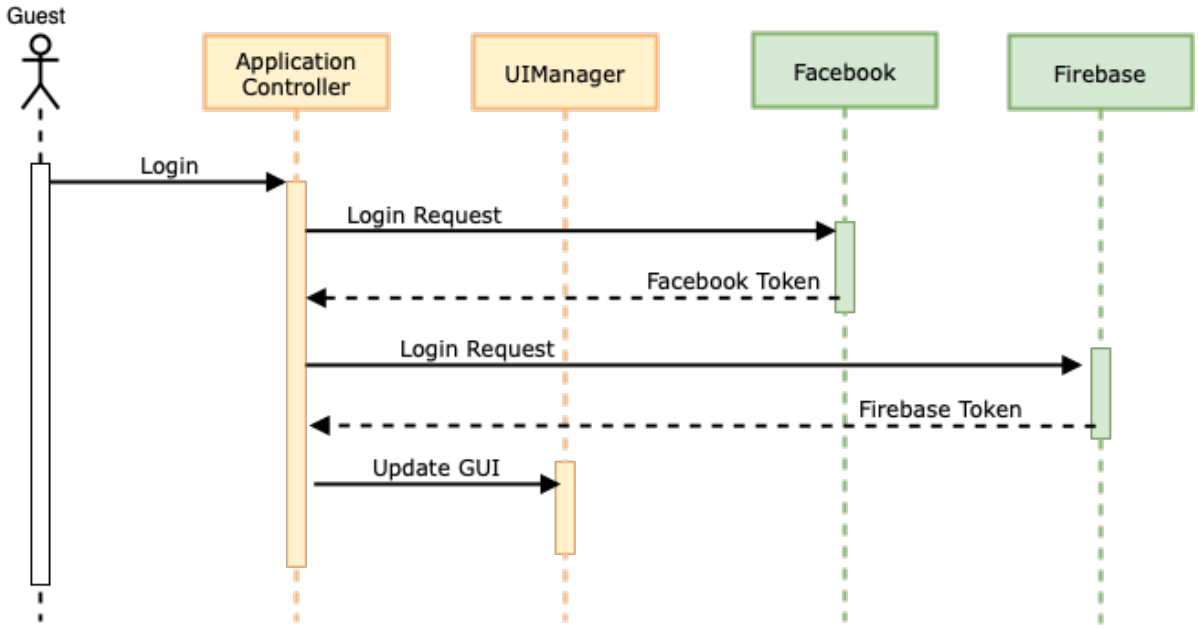
In order to explain our work and facilitate further implementations we have decided to show the logical flows aimed to create some features. The sequence diagrams will describe the interactions between the different parts of the system and the user.

Login

The "Login" procedure starts when the user presses the button on the Side Menu. Once pressed, the user is redirected to the Facebook site to complete the registration. Immediately after logging on Facebook the application will log into the Firebase Database.

Once this procedure is completed, the system will update the GUI by enabling the sections available only to subscribers.

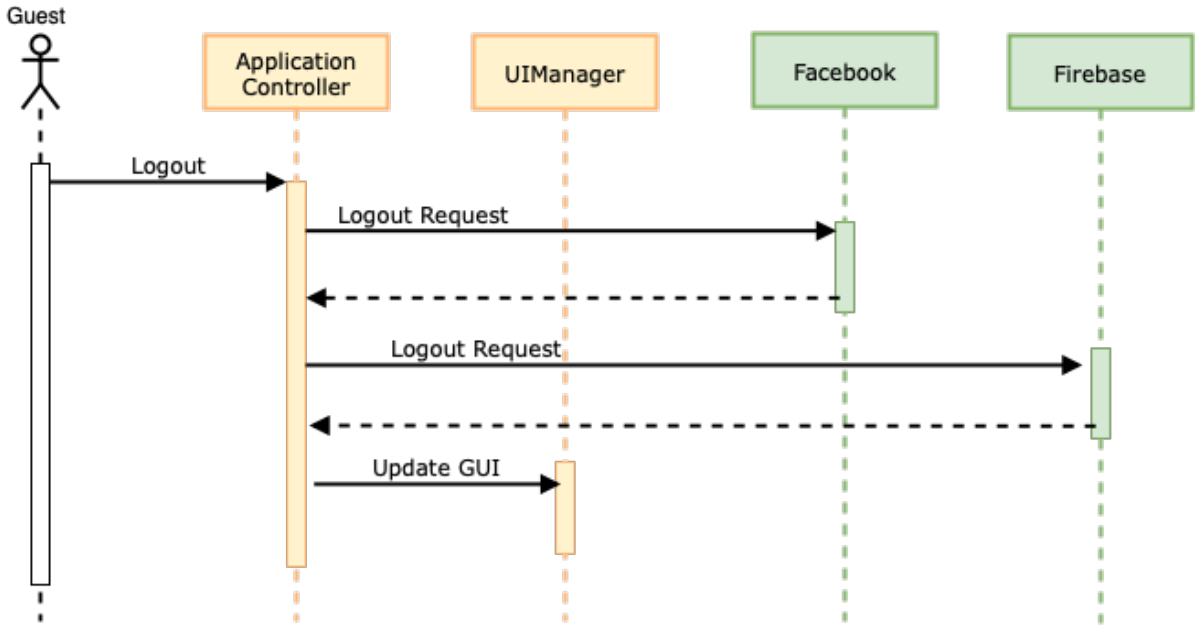
In the event of an error, the system will show an error message.



Logout

The "Logout" procedure starts when the user presses the button on the Side Menu. Once pressed, a confirmation message will be displayed. In case of approval, the logout from Facebook and Firebase will be effective.

Now that this procedure is completed, the system will update the GUI by enabling sections not available for guests.



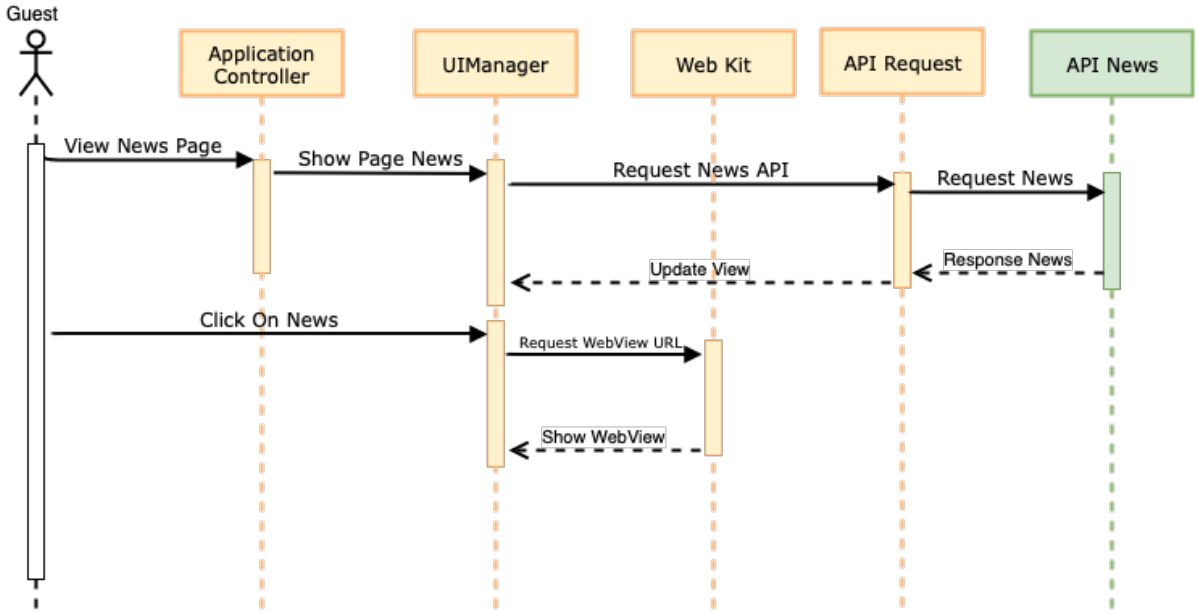
View News

The "View News" procedure starts when the user opens the application or presses the "News" section in the Tab Navigator. The sequence diagram shows the normal procedure.

After the user activates the News section, the application will send a request to the "NewsAPI" service in order to obtain all the information related to the most important daily news.

Once this information is obtained, the Controller will create a UITableViewCell for each news and insert them into the TableView.

Furthermore and asynchronously, the application will download the preview images to allow the user to read the news headlines. When a cell is pressed the Controller will open a WebView addressed to the URL of the news in order to allow the user to read the complete article.

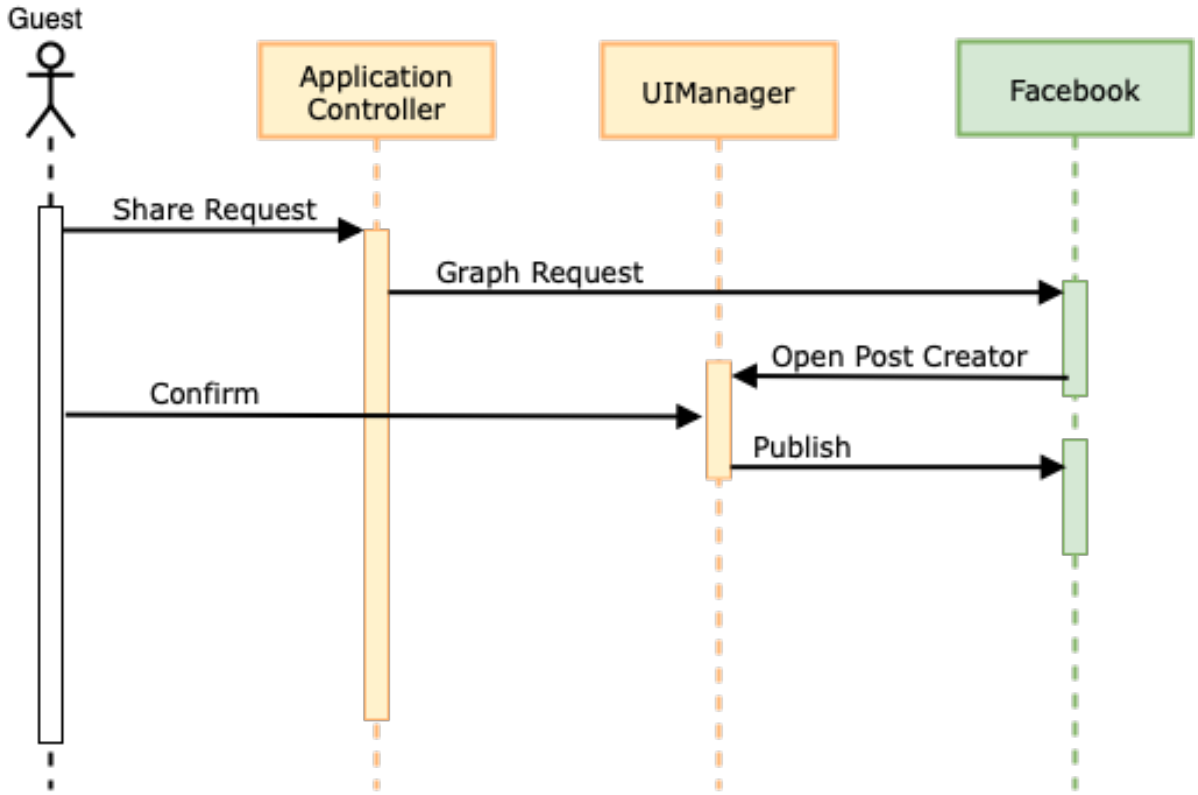


Share News

The "Share News" procedure starts when the user opens the application or presses "News" on the Side Menu. The sequence diagram shows the normal procedure.

After the normal "View News" procedure the user should press the share button. At that point the application will make a request to the "Graph" API of Facebook to create the post with the message entered by the user.

To make this operation possible is necessary that the user is already authenticated otherwise the system will show an error notice.



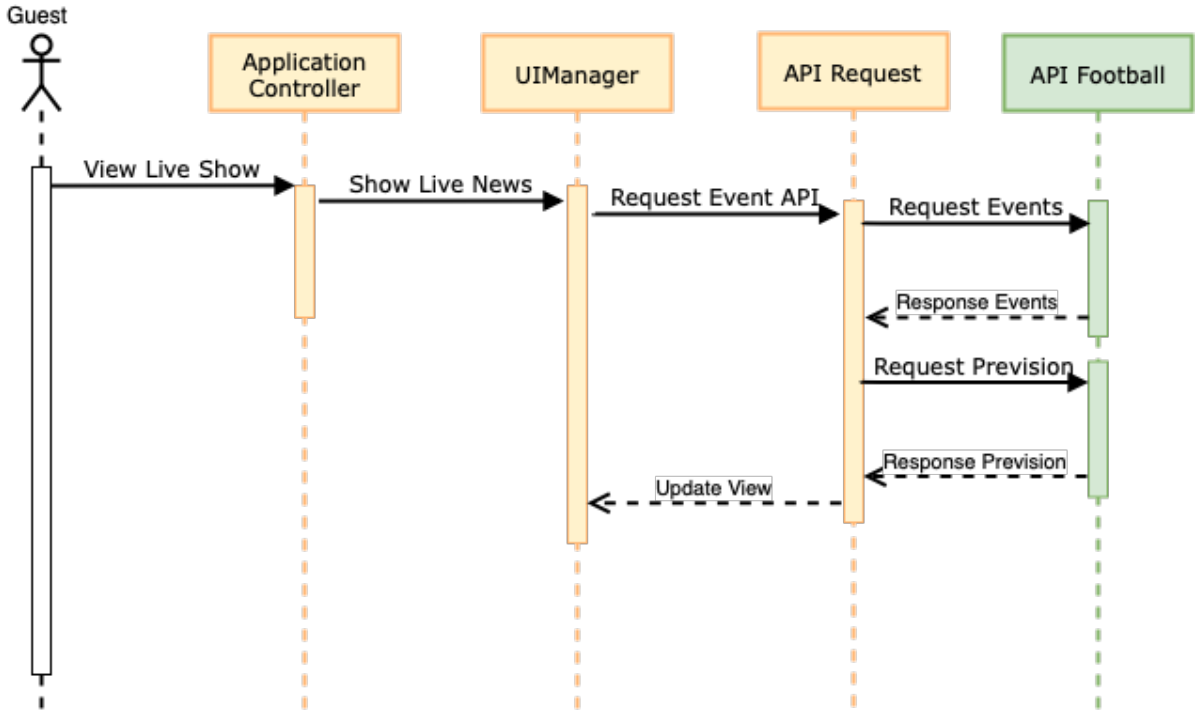
View Match

The "View Match" procedure starts when the user presses the "Live" section of the Tab Bar Navigator.

After activating the "Live" section, the application will send a request to the "API-Football" service to obtain information about the daily matches and their forecasts.

When this information has been reached, the Controller will create a UITableViewCell for all the matches and insert them into the TableView.

At regular intervals the Controller will repeat these requests to keep the matches updated.

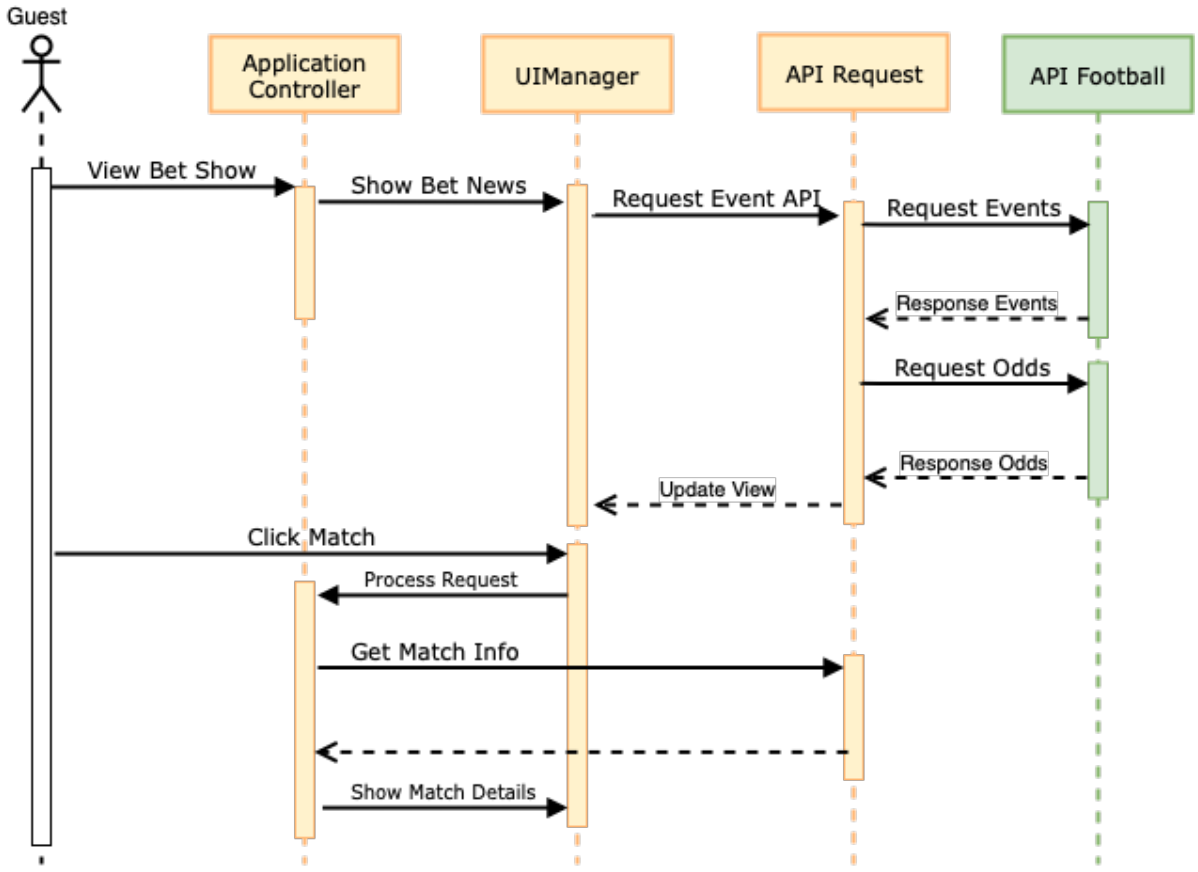


View Match Details

The "View Match" procedure starts when the user is in the "Live" section and presses on a single match to know the details.

Now the Controller has access to the data for the specific match, previously downloaded from the "APIFootball" service and creates Views for every information.

After opening the detail page, the user can change the desired information through the relative buttons.

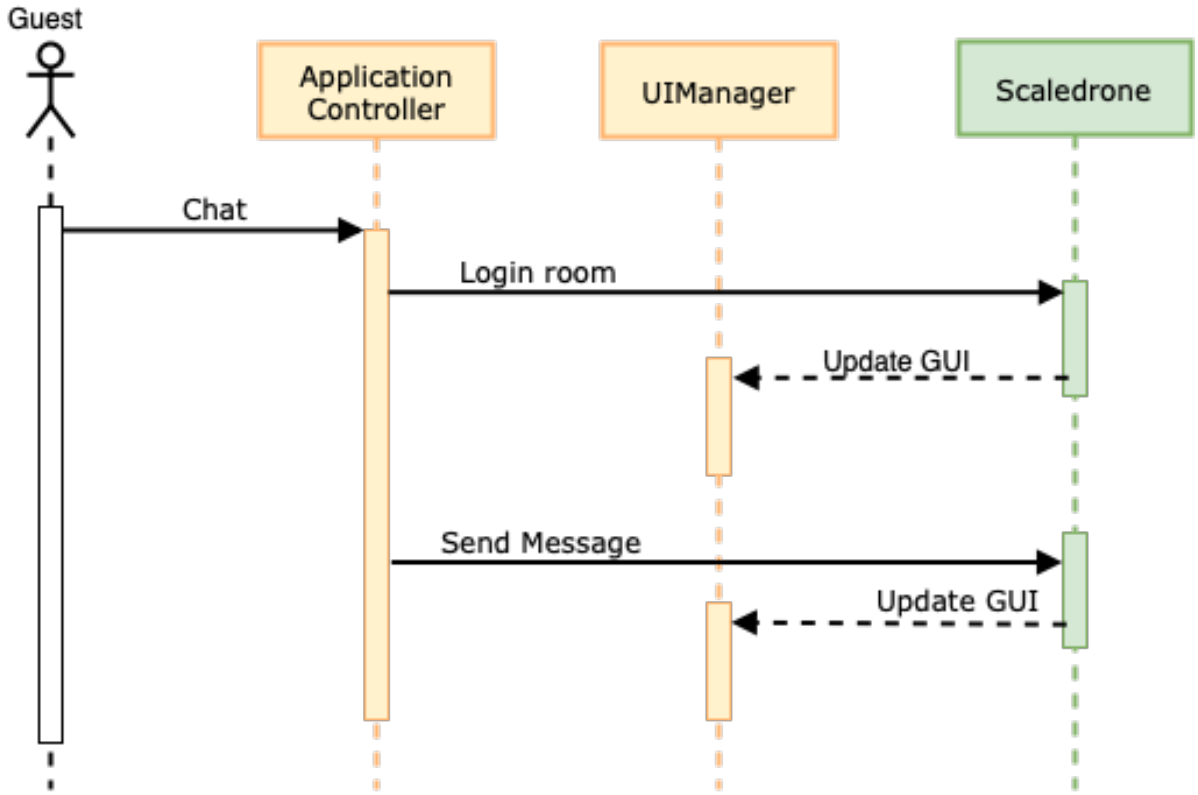


Chat

The "Chat" procedure starts when the user presses the button on the Side Menu. Once the screen is loaded, the application will show all messages sent by other users in real time. The user can then write and send his message.

The application will also show Facebook profile images of other users next to the message sent.

To make this operation possible is necessary that the user is already authenticated otherwise the system will show an error notice.



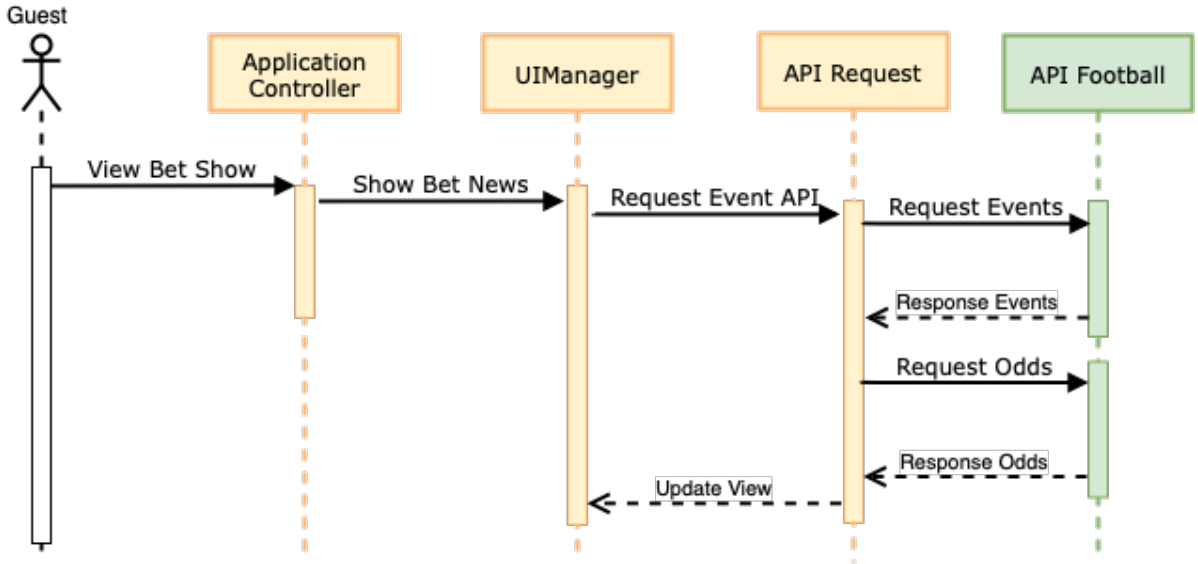
View Odds

The "View Odds" procedure starts when the user presses the "Bet" section on the Tab Bar Navigator.

Activated the "Bet" section, the application will send a request to the "APIFootball" service to obtain information about the daily matches and the relative odds.

At this point, the Controller will create a UITableViewCell for each news and insert them into the TableView. In particular, we inserted three buttons (one for each prediction) with the relative odd indicated.

At regular intervals the Controller will repeat these requests to keep the matches updated.



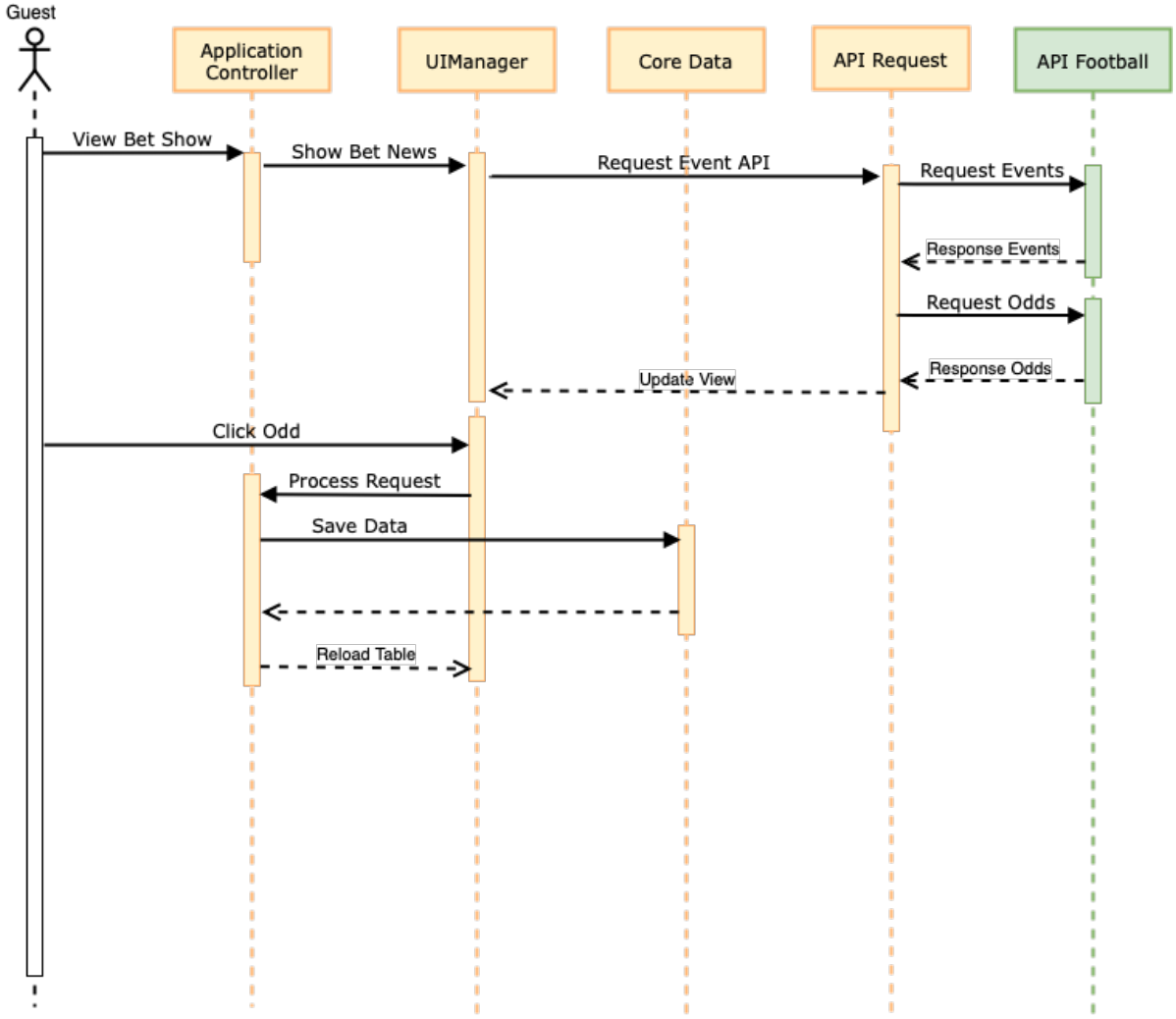
Add Odd

The "Add Odd" procedure starts when the user is in the "Bet" section and presses on the prediction of the match he wants to bet on.

At that point the Controller has access to the data for the specific match, previously downloaded from the "APIFootball" service and create the data structure that will be saved in the Database.

Once the data structure is created, the Controller will check if the match is already present in the database; If so, it will update the element with the new forecast, otherwise it will create a new record.

After updating the DB the Controller will update the current View to show the change in a visual feedback.

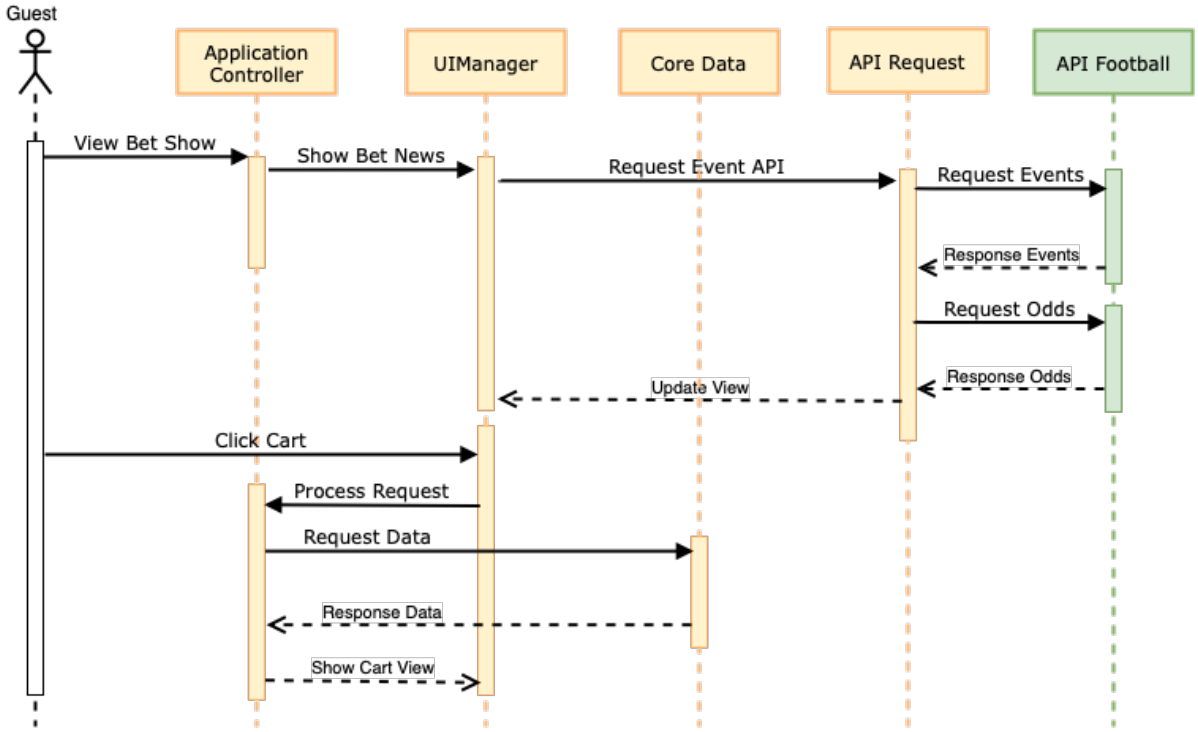


View Cart

The "View Cart" procedure starts when the user is in the "Bet" section and presses the button showing a cart in the Navigation bar.

At that point, the Controller has access to the data in the Database and for each of them it will create a UITableViewCell to be inserted in the TableView.

The user can change the amount of money gambled by using the corresponding input. Once the amount has been updated, the Controller will calculate the potential winnings by multiplying all the odds with the amount played.

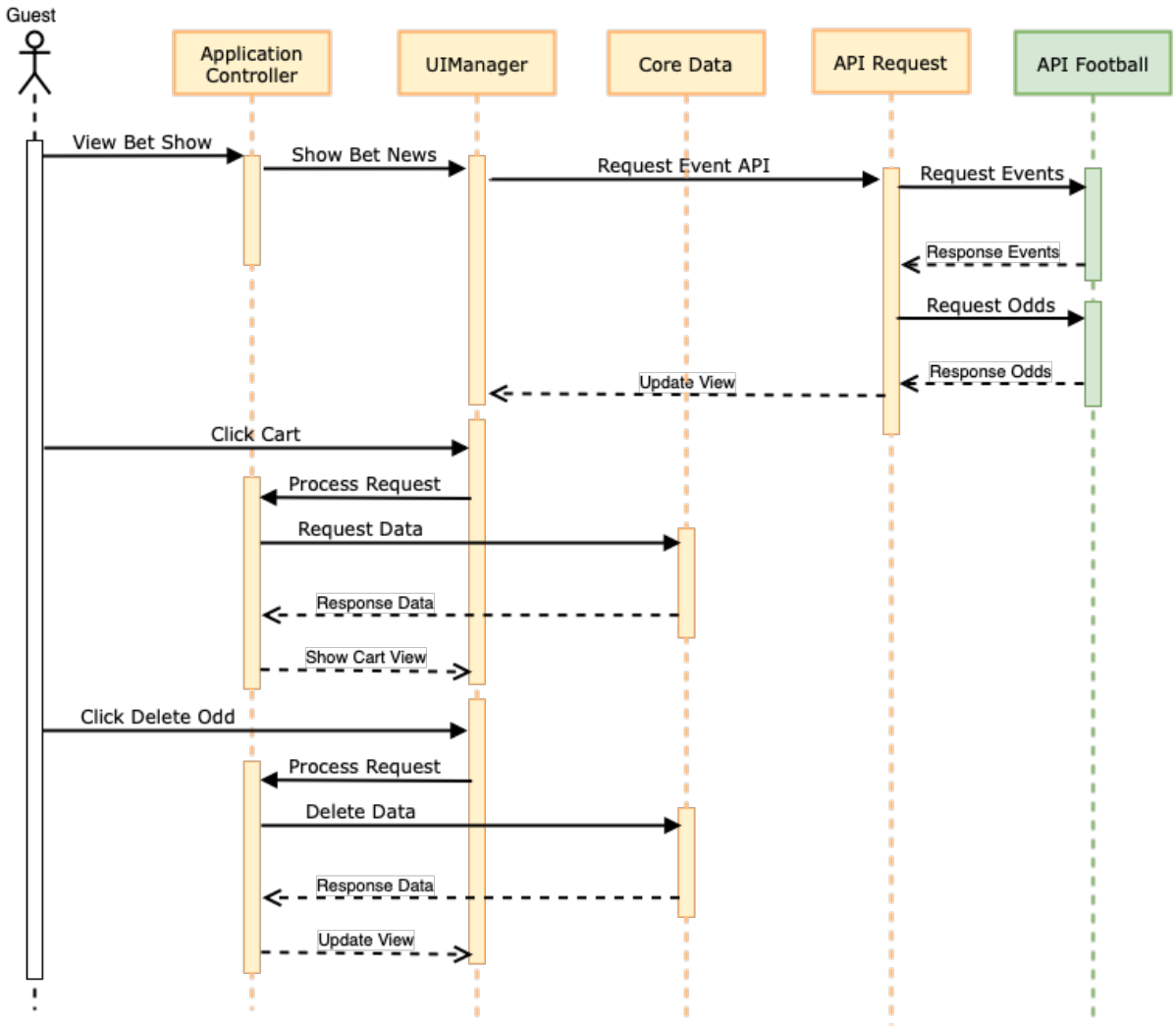


Remove Odd

The procedure of "Remove Odd" begins when the user is in the "Bet" section and he is checking the cart.

In this section the user can delete each bet by swiping to the left with his finger.

At that point the Controller will delete the corresponding record from the Database and update the contents of the shopping cart also recalculating the new potential winnings.

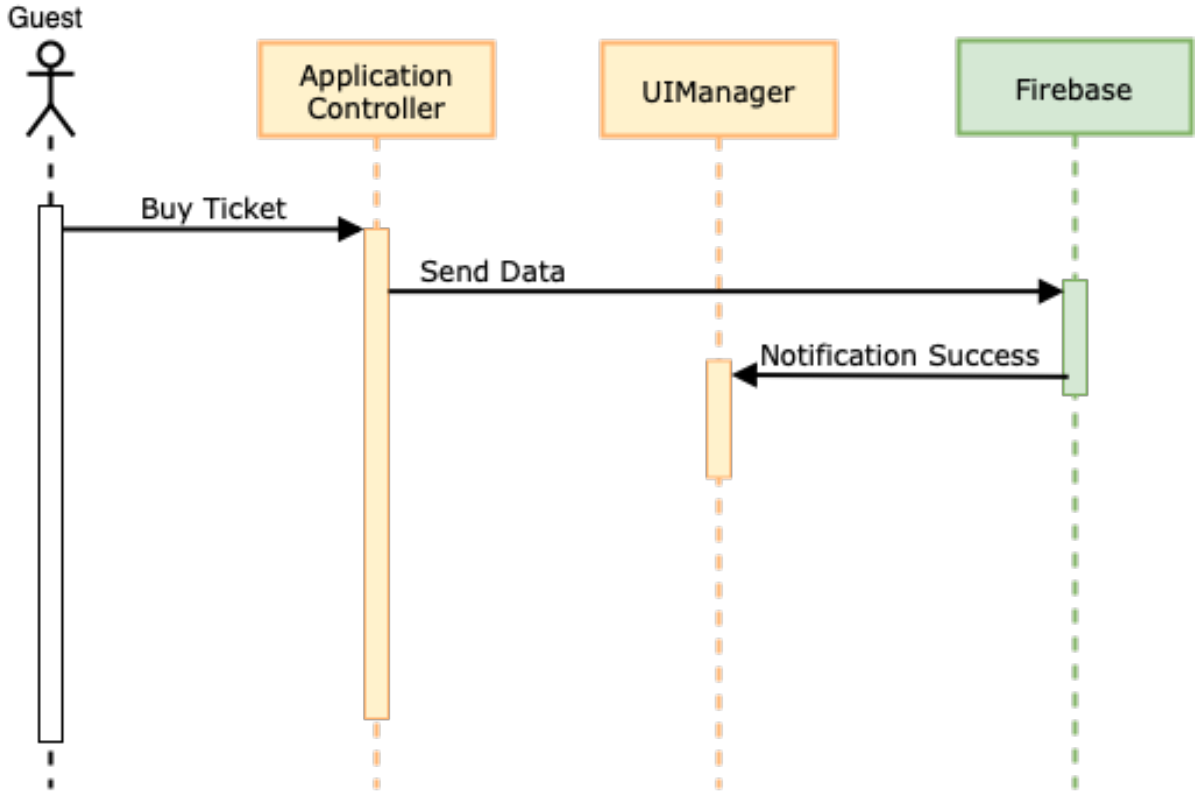


Buy Ticket

When the user completes the composition of his ticket, the "Buy Ticket" procedure may begin.

By pressing the buy button, the application will check the correctness of the ticket and proceed with the purchase.

To make this operation possible is necessary that the user is already authenticated otherwise the system will show an error notice.

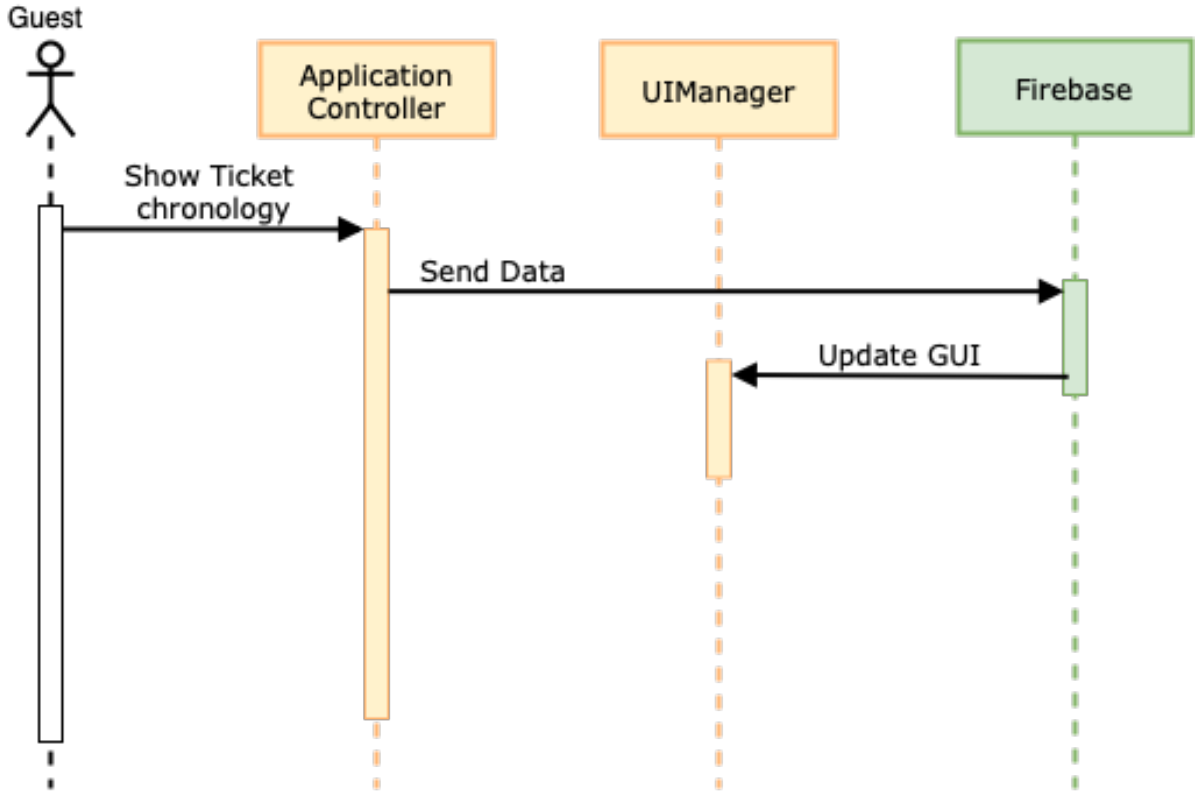


Ticket Chronology

The "Ticket Chronology" procedure starts when the user is in the "Bet" section by pressing the button in the Side Menu.

Moreover, by pressing "History" on the NavBar the application access Firebase database showing all the tickets purchased previously by the user.

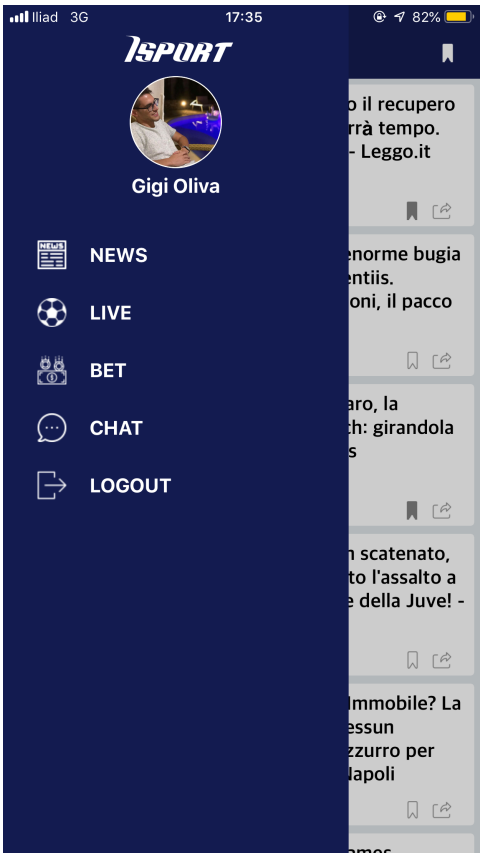
To make this procedure possible is necessary that the user is already authenticated otherwise the system will show an error notice.



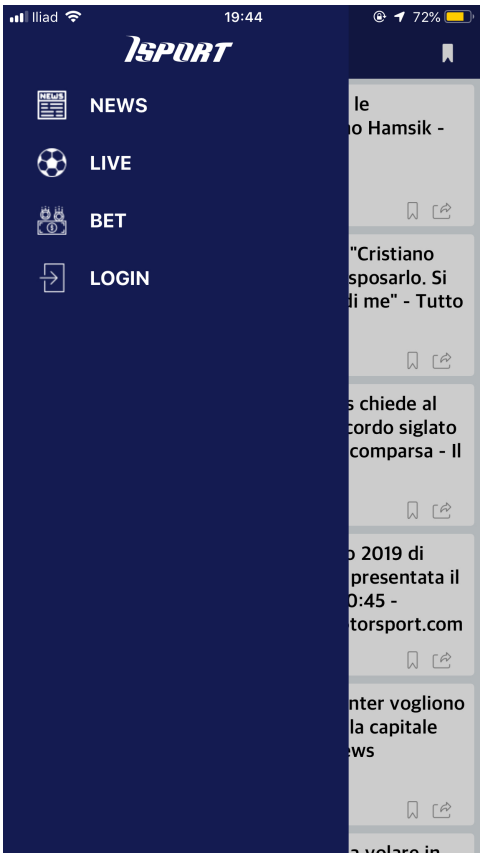
5 User Interfaces

In this chapter will be discussed and displayed the user interface of each page highlighting the functionalities of the iSport application.

Side Menu

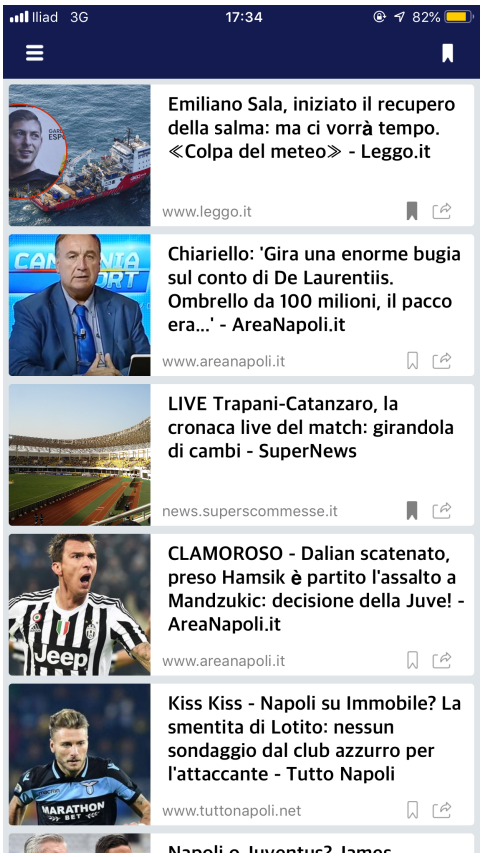


(a) In questa schermata viene mostrato il Side Menu di un utente autenticato



(b) In questa schermata viene mostrato il Side Menu di un utente non autenticato

News



(a) In questa schermata vengono mostrate tutte le notizie con la possibilità di salvarle o pubblicarle

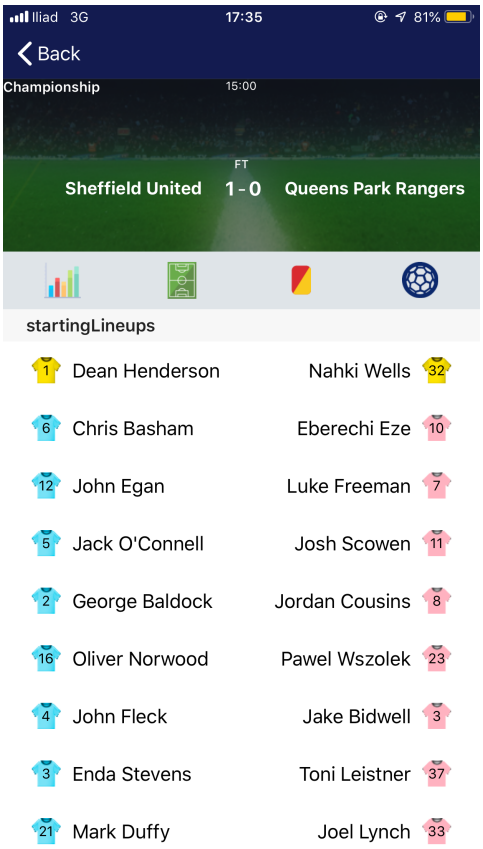
(b) n questa schermata vengono mostrate tutte le notizie salvate

(a) In questa schermata viene mostrata la condivisione di News

(b) WebView per la visione della notizia completa

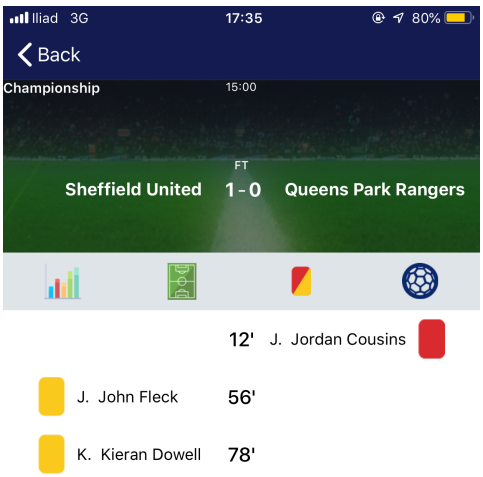
Live

(a) In questa schermata vengono mostrate tutte le partite live



(b) n questa schermata vengono mostrate le formazioni delle due squadre

(a) In questa schermata viene mostrate i goal della partita

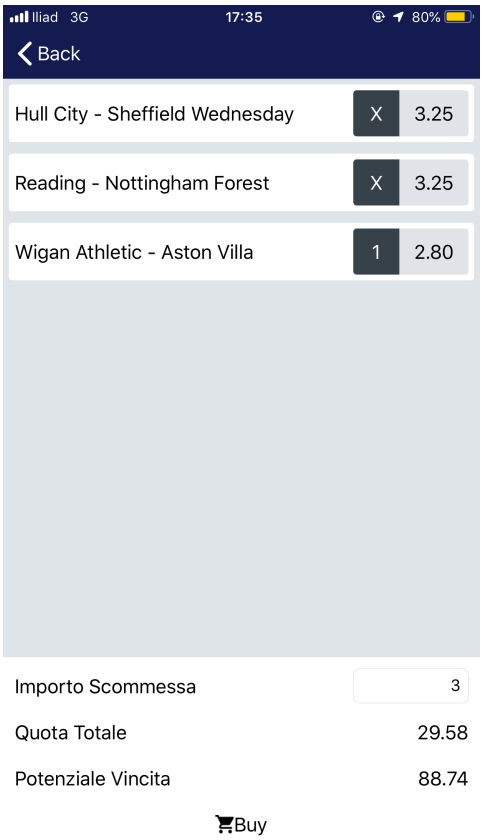


(b) In questa schermata viene mostrate i cartellini della partita

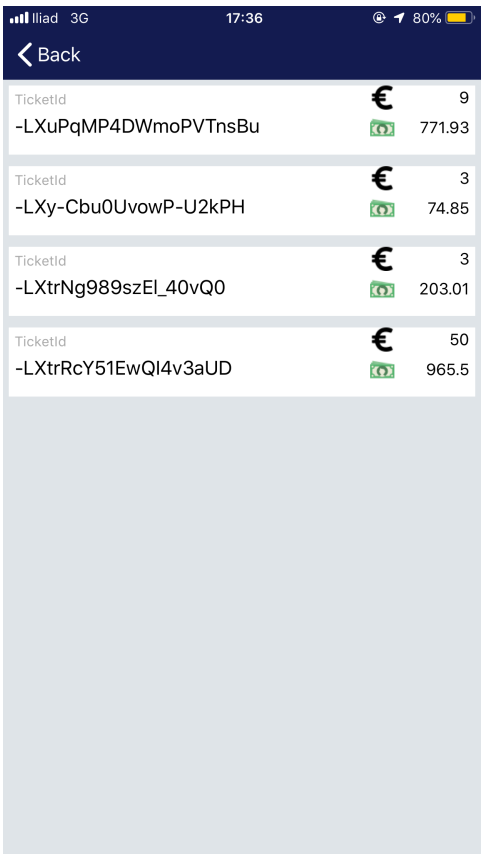
(a) In questa schermata vengono mostrate le varie statistiche

Odds

(a) In questa schermata viene mostrata la composizione di una schedina

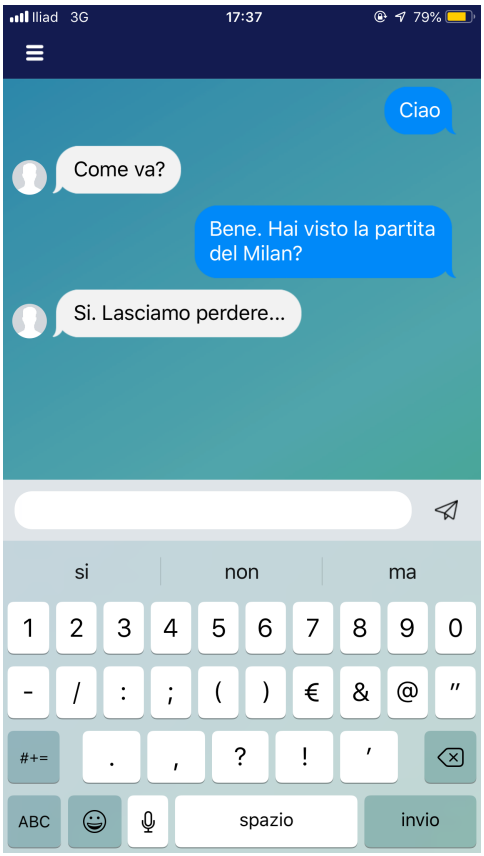


(b) n questa schermata viene mostrato la lista delle partite inserite in una schedina



(a) In questa schermata viene mostrato lo storico di tutte le schedine giocate

Chat



(a) In questa schermata viene mostrata la chat

6 External Services and Libraries

iSport application uses several third parties API in order to provide all the services to the user.
Per il parsing dei dati JSON è stato usato il componente JSONDecoder nativo dell UIKit.

Facebook SDK

Per la gestione del login sono state utilizzate le Facebook SDK. Quando un utente vuole autenticarsi l'applicazione esegue la chiamata al servizio di Facebook mediante:

```
@objc func loginButtonClicked() {
  let loginManager = LoginManager()
  loginManager.logIn([ .publicProfile ], viewController: self) { loginResult in
    switch loginResult {
    case .Failed(let error):
      print(error)
    case .Cancelled:
      print("User cancelled login.")
    case .Success(let grantedPermissions, let declinedPermissions, let accessToken):
      print("Logged in!")
  }
}
```

Tale metodo rimanda al sito o all'applicazione di Facebook e una volta autenticato si viene reindirizzati ad "iSport" che andrà a memorizzare il token fornitogli.

Per la pubblicazione delle notizie invece viene utilizzato il servizio di "Graph API":

```
let content = LinkShareContent(url: NSURL("https://developers.facebook.com"))
try ShareDialog.show(from: myViewController, content: content)
```

Per utilizzare il servizio di Facebook sono stati inclusi i seguenti pod:

```
1 pod 'FacebookCore'
2 pod 'FacebookLogin'
3 pod 'FacebookShare'
```

Firebase SDK

Per l'accesso al database vengono utilizzate le SDK di Firebase. In particolare viene utilizzato il login tramite le credenziali di Facebook in modo da avere un login unico.

```
Auth.auth().signInAndRetrieveData(with: credential) { (authResult, error) in
    if let error = error {
        print("Errore Log In Firebase \(error)")
        return
    }
    print("Accesso eseguito Firebase")
}
```

Per l'accesso al database è stato usato il DatabaseManager incluso nelle SDK:

```
if let user = Auth.auth().currentUser {
    let database = Database.database().reference()
    database.child("Schedina").child(user.uid).observeSingleEvent(of: .value) { (snapshot) in

    }

}
```

Per utilizzare il servizio di Firebase sono stati inclusi i seguenti pod:

```
pod 'Firebase/Core'
pod 'Firebase/Auth'
pod 'Firebase/Database'
```

Scaledrone

Per il servizio di chat in tempo reale sono state utilizzate le API di Scaledrone. Per la connessione alla room viene utilizzata la seguente estensione:

```
extension ChatService: ScaledroneDelegate {
    func scaledroneDidConnect(scaledrone: Scaledrone, error: NSError?) {
        print("Connected to Scaledrone")
        room = scaledrone.subscribe(roomName: "observable-room")
        room?.delegate = self
    }

    func scaledroneDidReceiveError(scaledrone: Scaledrone, error: NSError?) {
        print("Scaledrone error", error ?? "")
    }

    func scaledroneDidDisconnect(scaledrone: Scaledrone, error: NSError?) {
        print("Scaledrone disconnected", error ?? "")
    }
}
```

Mentre per la ricezione dei messaggi:

```swift
extension ChatService: ScaledroneRoomDelegate {
    func scaledroneRoomDidConnect(room: ScaledroneRoom, error: NSError?) {
        print("Connected to room!")
    }

    func scaledroneRoomDidReceiveMessage(
        room: ScaledroneRoom,
        message: Any,
        member: ScaledroneMember?) {

        guard
            let text = message as? String,
            let memberData = member?.clientData,
            let member = Member(fromJSON: memberData)
            else {
                print("Could not parse data.")
                return
        }

        let message = Message(
            member: member,
            text: text,
            messageId: UUID().uuidString)
        messageCallback(message)
    }
}
```

Per utilizzare il servizio di Scaledrone sono stati inclusi i seguenti pod:

```
1 pod 'Scaledrone', '~> 0.3.0'
```

NewsAPI

Quando un utente vuole visualizzare le principali notizie del giorno l'applicazione esegue le seguenti operazioni:

- Esegue una richiesta GET al l'endpoint fornito dal servizio
- Se la risposta è valida effettua il parsing del contenuto JSON
- Crea le celle della tabella contenente le informazioni fornite dal servizio e parsate
- Scarica in maniera asincrona le immagini di anteprima

L'endpoint utilizzato è "https://newsapi.org/v2/top-headlines?country=it&category=sports &apiKey=API_KEY". La risposta fornita dal servizio è:

```
{
   status: "ok",
   totalResults: 70,
 - articles: [
    - {
       - source: {
            id: null,
            name: "Milannews.it"
         },
         author: "Salvatore Trovato",
         title: "Milan, La Repubblica: \"Vincoli e crescita, Gazidis indica la via di
         mezzo\" - Milan News",
         description: "Un “sentiero ragionevole” lungo il quale possano camminare
         insieme sostenibilità finanziaria e libertà di investimento. È questa -
         riporta il quotidiano La Repubblica",
         url: https://www.milannews.it/news/milan-la-repubblica-vincoli-e-crescita-
         gazidis-indica-la-via-di-mezzo-321868,
         urlToImage: https://net-
         storage.tccstatic.com/storage/milannews.it/img_notizie/thumb3/34/34bd9b6fe7d0e17
         48194-639c61846db6ba1fb8fe64bca2ce28f5.jpeg,
         publishedAt: "2019-01-25T09:37:26Z",
         content: "Un “sentiero ragionevole” lungo il quale possano camminare insieme
         sostenibilità finanziaria e libertà di investimento. È questa - riporta il
         quotidiano La Repubblica - la via di uscita dal braccio di ferro tra Milan e
         Uefa suggerita da Ivan Gazidis. \"Lo scop… [+300 chars]"
      },
    - {
       - source: {
            id: null,
            name: "Tuttojuve.com"
         },
```


Per il parsing quindi è stata usata la seguente struttura dati:

```
struct Article: Decodable {
    let author: String?
    let title: String?
    let description: String?
    let url: String?
    let urlToImage: String?
    let publishedAt: String?
}

struct Articoli: Decodable {
    let status: String
    let articles: [Article]
}
```

APIFootball

Quando un utente vuole visualizzare i risultati delle partite del giorno l'applicazione esegue le seguenti operazioni:

- Esegue una richiesta GET al l'endpoint fornito dal servizio
- Se la risposta è valida effettua il parsing del contenuto JSON
- Crea le celle della tabella contenente le informazioni fornite dal servizio e parsate
- Aggiorna le informazioni ogni 30 secondi per mantenere i dati in tempo reale

L'endpoint utilizzato per ottenere i risultati è "https://apifootball.com/api/?action=get_events". La risposta fornita dal servizio è:

```
[ ⊟
  { ⊟
    "match_id":"277488",
    "country_id":"170",
    "country_name":"Italy",
    "league_id":"79",
    "league_name":"Serie A",
    "match_date":"2018-04-22",
    "match_status":"FT",
    "match_time":"15:00",
    "match_hometeam_name":"Juventus",
    "match_hometeam_score":"0",
    "match_awayteam_name":"SSC Napoli",
    "match_awayteam_score":"1",
    "match_hometeam_halftime_score":"0",
    "match_awayteam_halftime_score":"0",
    "match_hometeam_extra_score":"",
    "match_awayteam_extra_score":"",
    "match_hometeam_penalty_score":"",
    "match_awayteam_penalty_score":"",
    "match_hometeam_system":"4-3-2-1",
    "match_awayteam_system":"4-3-3",
    "match_live":"1",
    "goalscorer":[ ⊞ ],
    "cards":[ ⊞ ],
    "lineup":{ ⊞ },
    "statistics":[ ⊞ ]
  }
]
```


Per il parsing quindi è stata usata la seguente struttura dati:

```
struct Partita: Decodable {
    let leagueName: String?
    let matchId: String?
    let matchStatus: String?
    let matchTime: String?
    let matchDate: String?
    let matchHometeamName: String?
    let matchHometeamScore: String?
    let matchAwayteamName: String?
    let matchAwayteamScore: String?
    let matchHometeamHalftimeScore: String?
    let matchAwayteamHalftimeScore: String?
    let goalscorer: [GoalList]
    let cards: [CardList]
    let statistics: [Statistic]
    let lineup: Formazione
}

struct GoalList: Decodable {
    let time: String?
    let homeScorer: String?
    let score: String?
    let awayScorer: String?
}
```

```
struct CardList: Decodable {
    let time: String?
    let homeFault: String?
    let card: String?
    let awayFault: String?
}

struct Statistic: Decodable {
    let type: String?
    let home: String?
    let away: String?
}

struct Formazione: Decodable {
    let home: Campo?
    let away: Campo?
}

struct Campo: Decodable {
    let startingLineups: [Lineup]?
    let substitutions: [Lineup]?
}

struct Lineup: Decodable{
    let lineupPlayer: String?
    let lineupNumber: String?
    let lineupPosition: String?
    let lineupTime: String?
}
```

Per ottenere le previsioni della partita l'endpoint utilizzato è "https://apifootball.com/api/?action=get_predictions". La risposta fornita dal servizio è:

```
[
  {
    "match_id":"369090",
    "country_id":"253",
    "country_name":"Australia",
    "league_id":"684",
    "league_name":"A-League",
    "match_date":"2019-01-08",
    "match_status":"FT",
    "match_time":"08:50",
    "match_hometeam_name":"Western Sydney Wanderers FC",
    "match_hometeam_score":"2",
    "match_awayteam_name":"Wellington Phoenix FC",
    "match_awayteam_score":"3",
    "match_hometeam_halftime_score":"1",
    "match_awayteam_halftime_score":"1",
    "match_hometeam_extra_score":"",
    "match_awayteam_extra_score":"",
    "match_hometeam_penalty_score":"",
    "match_awayteam_penalty_score":"",
    "match_hometeam_system":"4-3-3",
    "match_awayteam_system":"3-4-3",
    "match_live":"0",
    "prob_HW":"20.00",
    "prob_D":"25.00",
    "prob_AW":"55.00"
  }
]
```


Per il parsing quindi è stata usata la seguente struttura dati:

```
struct Prediction: Decodable{
    let matchId: String?
    let probHw: String?
    let probD: String?
    let probAw: String?
}
```

Per ottenere le quote delle partite l'endpoint utilizzato è "https://apifootball.com/api/?action=get_odds". La risposta fornita dal servizio è:


```
[
  {
    "match_id":"148356",
    "odd_bookmakers":"10Bet",
    "odd_date":"2017-02-07 07:41:36",
    "odd_1":"1.77",
    "odd_x":"3.74",
    "odd_2":"5.30"
  }
]
```


Per il parsing quindi è stata usata la seguente struttura dati:

```
struct Odds: Decodable{
    let matchId: String?
    let odd1: String?
    let oddX: String?
    let odd2: String?
}
```

7 Software System Attribute

7.1 Reliability

As the majority of function requires an internet connection, the software is reliable as long as there are no connectivity problems.

7.2 Availability

The availability parameter also relies on the internet connection signal and on the responses provided by "https://newsapi.org" and "https://apifootball.com". No problems of availability were found both during the developement phase and beta testing with users.

7.3 Security

The security of the iSport application was the main concern during the developement. As all the information provided are retrived from the web, there are different checks to perform in order to keep the user information safe. Furthermore, for the communication with external services, was chosen HTTPS protocol to guarantee greater security

7.4 Maintainability

The entire application is very maintainable as the code is entirely documented, in particular in several critical function. Therefore any developer who wants to improve it or make changes is able to do it without relevant difficulty.

7.5 Usability

The usability was another of our concern for the development. In order to improve it, a beta version of the application was given to 5 users and tested for 1 weeks. The result of this test highlighted usability issues. In particular, in the first version the buttons for changing information on the match detail page were not clearly visible. Another report concerned the lack of visual feedback after adding a quota to their ticket. Both of these problems were then solved in a later version of the application.

8 Test Cases

9 Cost Estimation