SOFTWARE ENGINEERING

SPRINT 2 - KICKOFF PROJECT

Functional & Non-Functional Requirements

In this section we briefly describe our functional and non-functional requirements.

Functional Requirements

As stated in the project proposal, we were committed to deliver **functional requirements** and we were committed in sprint one to deliver the following:

- Application for player institutes under the name **Player**.
- Ability to **Signup** and **login** as a player.
- Ability to view the **Player's profile**.
- Ability to **search for a court owner** with the court owner name.
- Search is Sorted by **distance**.
- Ability to **view court owners** selected from search.
- Ability to **view courts** associated with the owning institute.
- Ability to **book a reservation** to a court.
- Ability for the player to **Pay** for the reservation.
- Ability for the court owner to **confirm the payment**.

Non-Functional Requirements

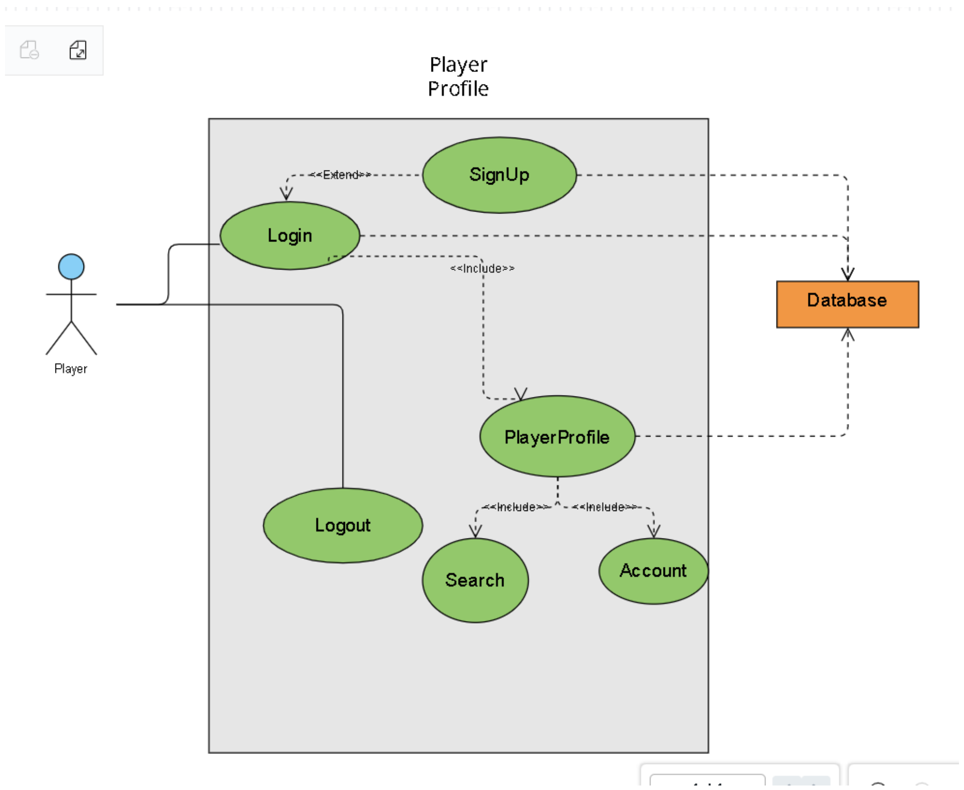
Non-functional requirements elicited were:

- Announcements viewing on any side shall have different views for each side.
- Players can only view announcements in courts. They can not add or edit any announcement as well as deleting announcements. Court Owners can update announcements.
- Players can book reservations at any time even when there is a pending reservation, unless the reservation is booked.

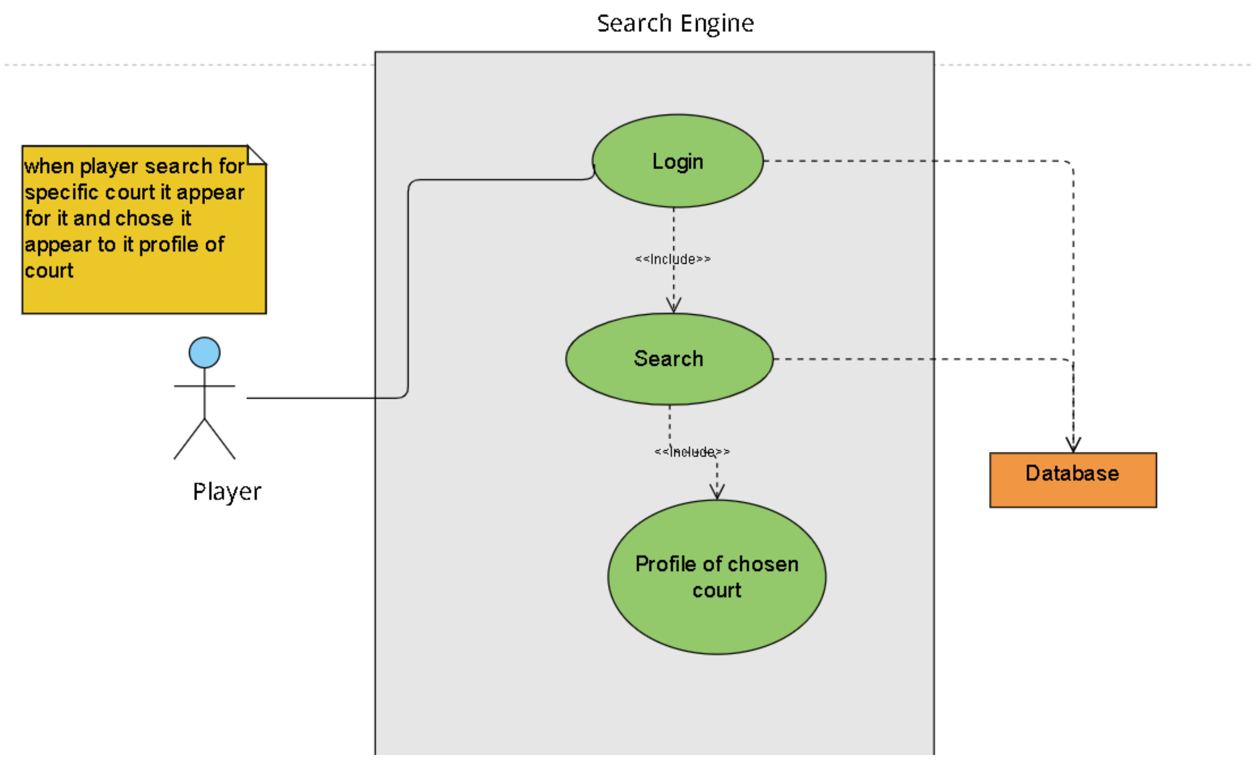
- Players shall view reservations with filtering.
- Search Engine provides courts within the perimeter of the user first then those far away later.
- Players & court owners can do their operational functionalities in efficient response time.
- Players & court owners interact with a user friendly graphical user interface.
- The player and the court owner graphical user interface is separated in design to distinguish their experience in using the application.
- The application offers a high degree of availability.
- The low percentage of data corrupted is ensured in case of failure of application.

Detailed Use Case Diagrams

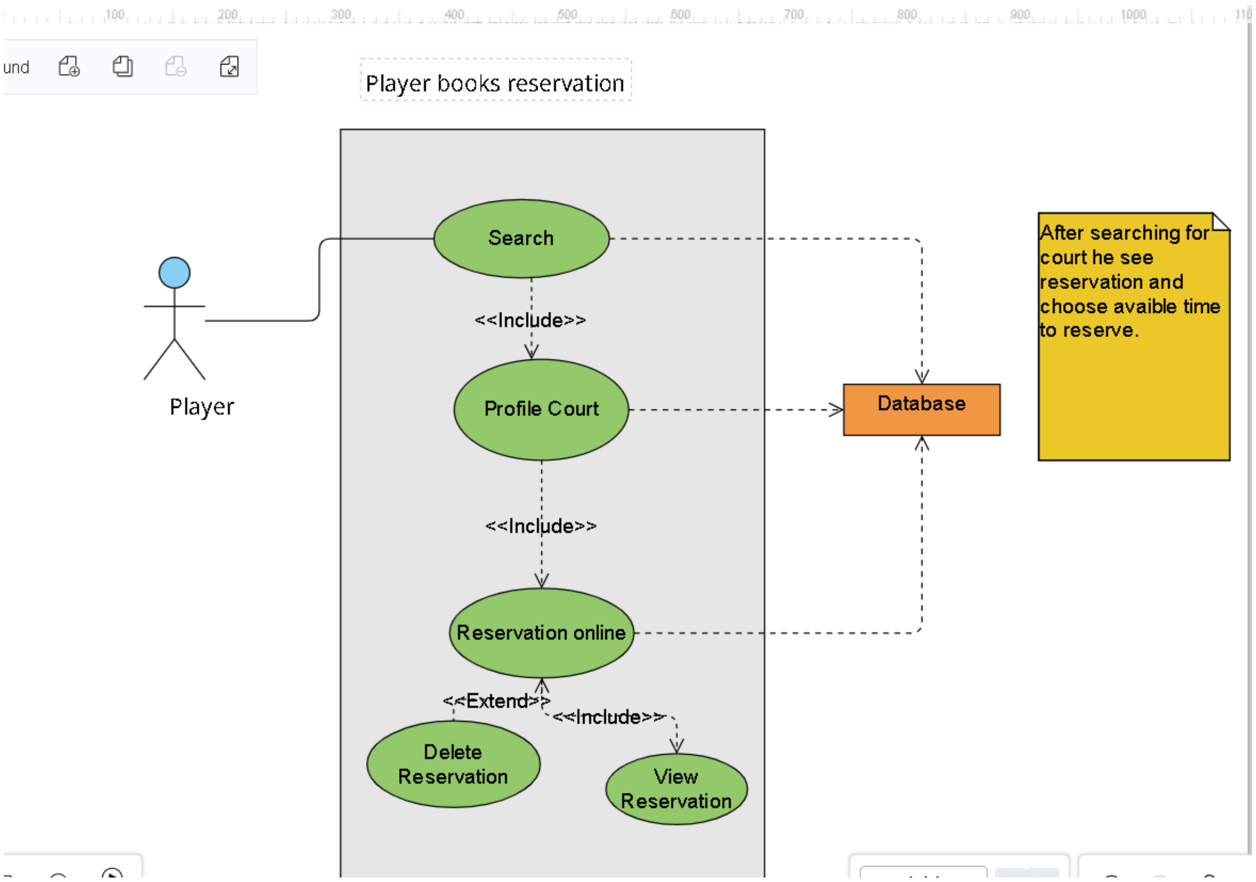
1. Use Case Diagram: Login/Signup Component



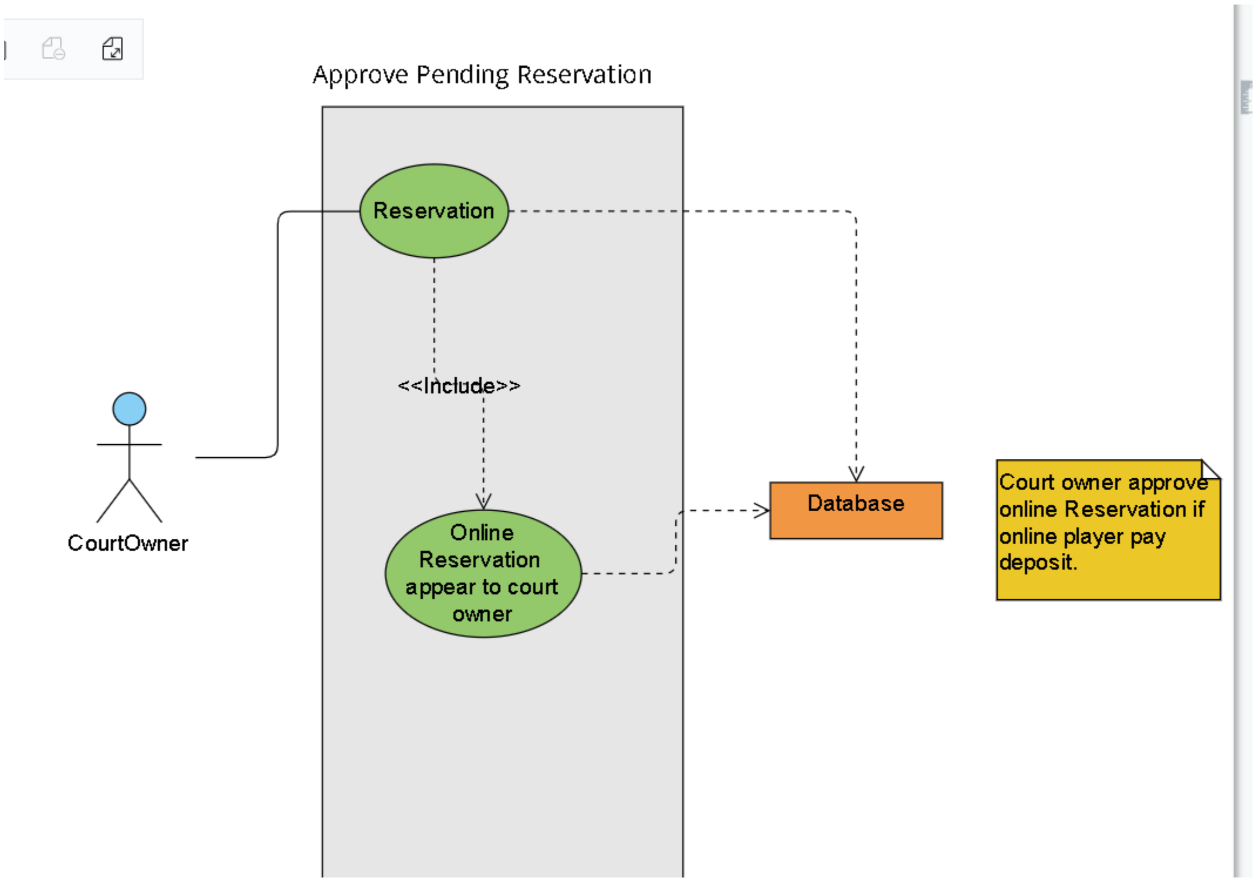
2. Use Case Diagram: Search Engine (Player Search for Court)


Search Engine
when player search for specific court it appear for it and chose it appear to it profile of court
Player
Login
<<Include>>
Search
<<Include>>
Profile of chosen court
Database

3. Use Case Diagram: Player book online Reservation.

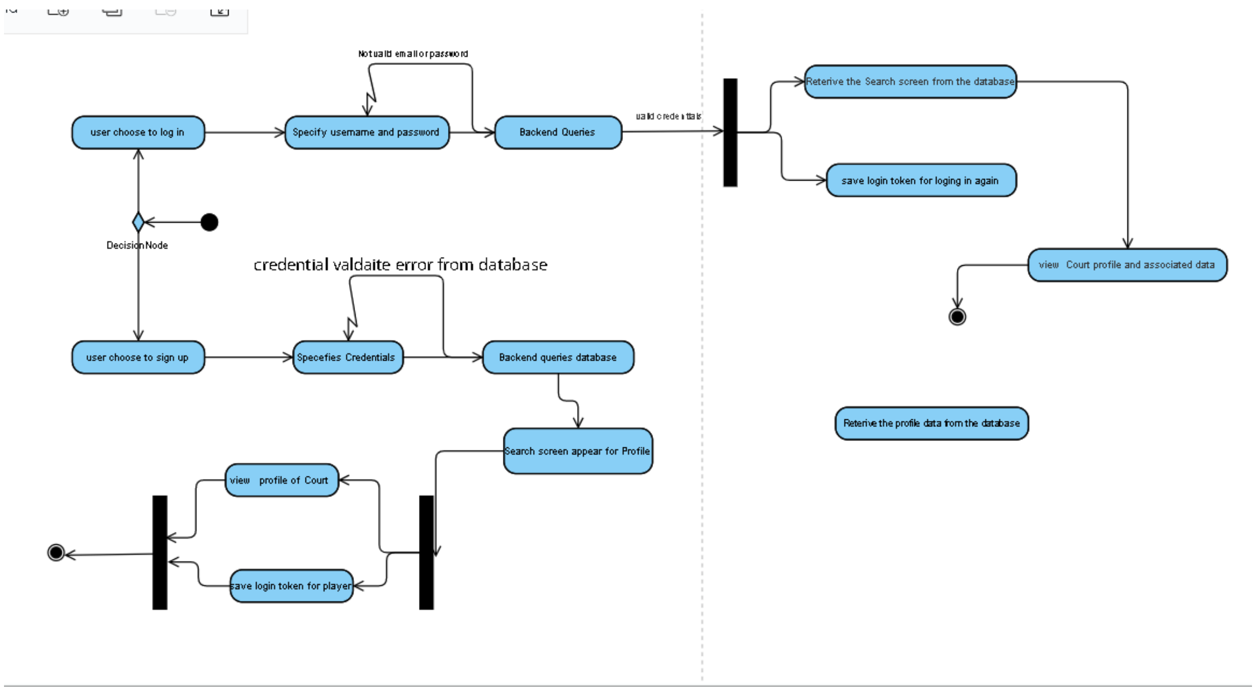


4. Use Case Diagram: CourtOwner Accept Online Reservation if it player pay deposit .

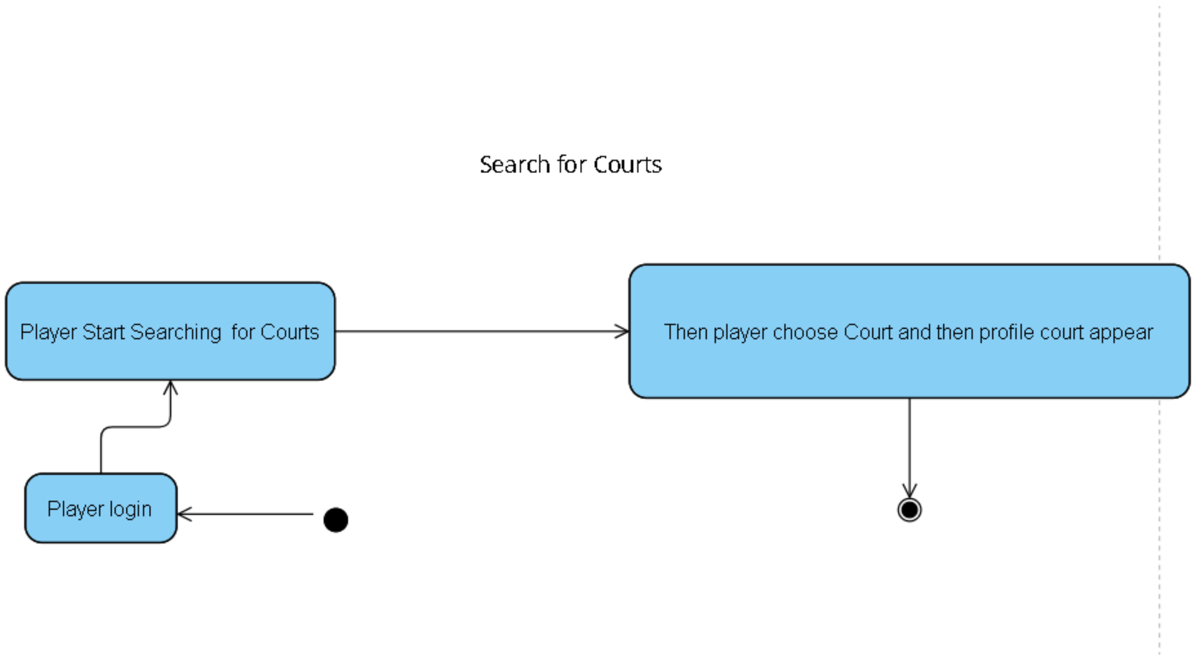

Approve Pending Reservation
Reservation
<<Include>>
Online Reservation appear to court owner
CourtOwner
Database
Court owner approve online Reservation if online player pay deposit.

Activity Flow Diagrams

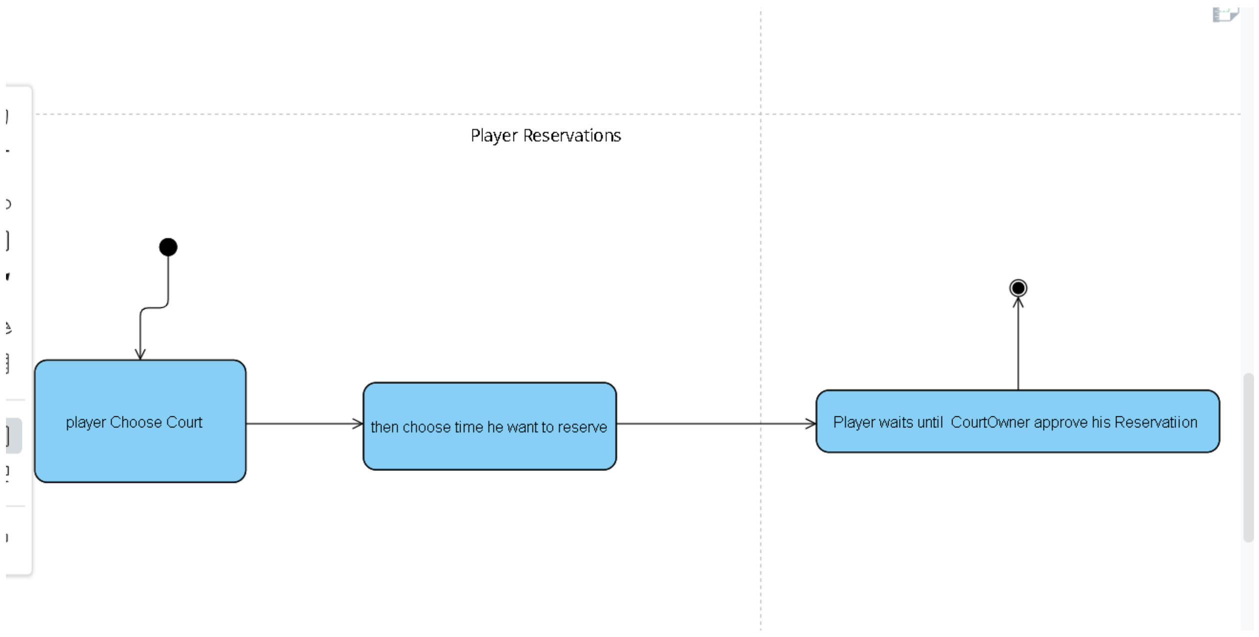
1. Activity Diagram: Login/Signup


Not valid email or password
user choose to log in
Specify username and password
Backend Queries
valid credentials
Reterive the Search screen from the database
save login token for loging in again
view Court profile and associated data
Decision Node
credential valdaite error from database
user choose to sign up
Specefies Credentials
Backend queries database
Search screen appear for Profile
Reterive the profile data from the database
view profile of Court
save login token for player

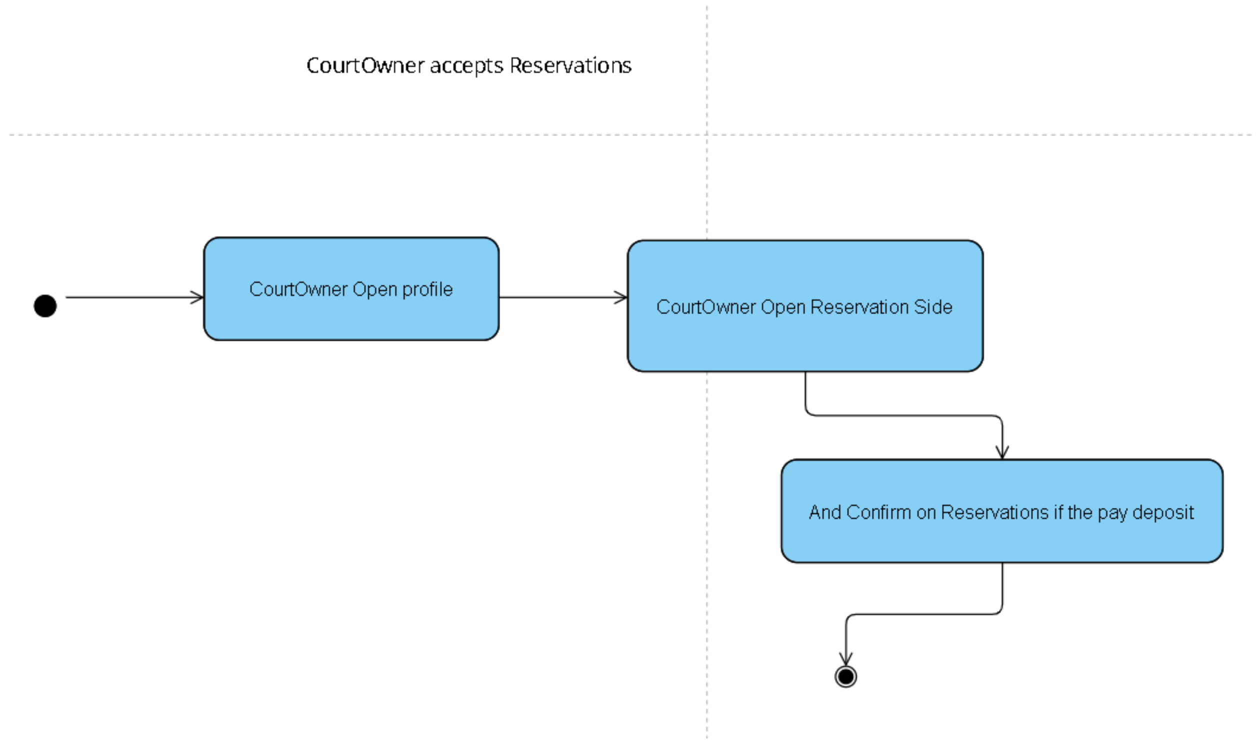
2. Activity Diagram: Search for Courts


Search for Courts
Player Start Searching for Courts
Then player choose Court and then profile court appear
Player login

3. Activity Diagram: player Reservation.


Player Reservations
player Choose Court
then choose time he want to reserve
Player waits until CourtOwner approve his Reservatiion

4. Activity Diagram: CourtOwner Accept Reservation .


CourtOwner accepts Reservations
CourtOwner Open profile
CourtOwner Open Reservation Side
And Confirm on Reservations if the pay deposit

Changes/Refactoring

No significant changes were issued after the initial project proposal.

Scrum Master Report

Sprint Planning Report

The sprint plan was represented in the following steps:

1. Requirement specification and creating the stories.

2. Creating tasks and assigning each task. Making sure that there are 2 or 3 people working in each story.
3. Developing and implementing each task will merging the related task iteratively
4. Testing and refactoring each task and component then the whole system
5. Fixing any bugs and reviewing the codes.

Please refer to *GIT/GITHUB REPORT* as it reports the pull requests, code reviews, etc.,.

Jira Report

Current Jira Roadmap

In the Jira roadmap, we have finished epics in this sprint associated with:

- Announcements
- Player
- Player Reservations
- Sprint One Bug Fixes
- Sprint One Edits and Enhancements
- Search Engine

Milestone sprint listing the included requirements Jira Stories

In this milestone we were deliberated to deliver an application designed for the players so they can book a reservation online and authenticate the payments associated with each reservation. Having delivered the court owner application, we were obliged to fix minor bugs in the application and so we have done. Both players and court owners now can have their stand-alone application delivered and working independently. One of our proposed services that is provided in this milestone is the search engine. Last milestone we are planning to introduce the other services as communication interfaces between players and court owners as well as minor fixes and code refactoring.

Stories, tasks, and subtasks estimates

Stories

- As a player, I need to search for courts in order to book a reservation in it. Pressing the search button should show the player the nearest courts within a constant perimeter. The player can opt out of that and just search the court by its name. Upon selecting a court from the search, the court profile shall pop up to the user.
- As a player, I need to upload a photo to identify my identity, specify my full name, edit my info, view my profile.
- The court owner needs to update his institute and keep subscribers updated with their court information. The court owner might post updates related to setting a certain court out of order, re-opening of a court, closing a court, etc.,. Once a post is posted into announcements associated with court owners, the players subscribed to the court shall be notified about that update. This feature is related to subscribers.
- As a player, I need to keep updated with the courts I have played in or might play in information. A player can subscribe to one court owner or more. Each court owner shall have a subscribe button. Upon pressing subscribe, the user shall be subscribed through a subscriber channel (Observer pattern is useful here).
- This feature is related to announcements posted by court owners. A player shall reserve a reservation after using the search engine to find a court in a court owner. The player then can specify the time for the reservation in case it does not violate the already reserved times constraint.
- A player shall view the reservations associated with his history. They can view past reservations, pending, awaiting confirmation, and booked reservations with all the associated data.
- A player can cancel a pending reservation. Booked reservations shall only be canceled offline.
- An online registered player can search for a court owner then select a dedicated owner. Once selecting a court associated with the court owner, the player shall see the already reserved times

- Court owners can approve pending reservations once they have received a payment receipt from the player that booked the reservation. In case of violation, the violating party is severely penalized (in the penalty agent sprint).

Tasks & Subtasks Estimates

- Front End
- Back End

Breaking down tasks by assignee.

Zeyad Zidan

Task	Estimated Time	Description
Frontend Reservations - Player Side	1 day	The player can book a reservation at any court he wants and know the court schedule in order to detect the desired time he wants to book at.
Feature Announcements to the court owner side.	3 hours	Design announcements page in the frontend of court owner side application
Feature announcements to the player side	1 hour	Design announcements page in the frontend of court owner side application
Reservations Deletion	6 hours	Court owners can delete reservations of any state. The player also can delete a reservation as long as it is pending or awaiting confirmation
Frontend GUI Enhancements	2 days	Huge improvements and modification to the application functionality and UI design.
Player Reservations Page	1 day	Design a page for players to view their various stated reservations
GUI for the cancellation of	1 hour	Requests for canceling booked, pending, and awaiting

a pending or booked reservation		confirmation tickets.
Ticket Model Edits	15 minutes	New attributes are added to the ticket model. These new attributes introduced new features and enhanced UX.
Court Model Full Stack Intercommunication	1 days	Courts are court owners that need to be linked together. The information associated with these are used later to help with the player-side application
View reservations in both ascending and descending orders	1 hour	Dropped to backlog
Reservations can be overlapping in days.	1 hour	Make the reservation overlapping in days in case they are pending or many courts for one court owner in a day.
Court Addition Fixes	3 hours	Fix bugs found in adding new courts.
Courts viewing bugs	1 day	• The frontend model does not support the full model implemented in the backend. • Enhance the application GUI. • The backend does not send the full model in the http requests.
Reservations' view on date.	1 hour	Bug fixes for this feature
Basic Player Announcement View	1 day	A player can view the court's announcements by simply visiting the court and selecting the courts announcements tab.
Payments Authentication	1 day	Provide a GUI interface for court owners and players to confirm bookings

Abdelrahman Elsayed Ahmed Aly

Task	Estimated Time	Description
Stay signed in improvements	3 days	Making the court owner and the player logged in when quit the application.
Routings	2 days	Solving sign up routing problems as well as routing through the whole application.
Implement image caching system	1 day	Using cache to improve application run time and performance
Fix logout	1 day	Fixing the bugs resulted from logging out of the application.
Reservation bugs.	1 hour	Fix Frontend reservations bugs.

Abdelaziz Mohammed

Task	Estimated Time	Description
Player profile frontend	2 days	Introduce a GUI for player profile.
Player Navigation Bar	1 day	Create a navigation bar for player to solve routings
Login Interface	2 days	Provide login interface.
Signup Interface	2 days	Provide sign up interface.

Abdelrahman Elsayed Gad

Task	Estimated Time	Description

Handle the pending booked reservations on the backend	1 hour	player can book a reservation but it is in pending state till it confirms
Handle the cancellation process on the backend.	1 hours	players can cancel the pending reservation.
add DateTime reservedTime to the request model	30 minutes	add the attributes to the model of the reservation
Map between announcements, court owner, and players	1 hours	Court owner shall introduce announcements and player shall view announcements. A mapping shall be introduced among all of these.
Pending Constraint	4 hours	The reservation must be booked within 30% of the difference between the time it was booked and the time of the actual reservation.
Validate every data field received on every request that it is valid data	3 hours	validate at setPending(startDate & endDate, startHour & endHour with startWorkingHour & endWorkingHour) validate at createCourt(morningCost, nightCost, minBookingHours, startWorkingHours & finishWorkingHours, finishMorning) validate courtOwnerSignup
Sort reservation by time	30 minutes	The Reservation Comparator makes it compare by time.
get All CourtOwner attributes for the player	1 hour	get the rest of the courtowner attributes in request
Booked Reservations Deletion Bugs.	3 hours	Booked reservations are not deleted from the backend view nor the database. That is not the case in the pending reservations deletion. used later to help with the player-side application
Reservations bugs.	3 hour	Backend reservations related bugs. Total Cost Calculation.

Reservations Time Calculations Bugs.	2 hour	handling that work court day can start in one day and end in another and the reservation as will so the time calculated should handle that.
Support announcements services	2 hours	Create and handle the service of the announcement like shown in the design.
Support announcements in the backend.	2 hours	Design and implement the announcements data model for court and player side applications in the backend.

Omar Khairat

Task	Estimated Time	Description
Player model backend	40m	Design the player entity model in the backend in order to be persisted in the database.
Player controller backend	40m	Map the actions requested by the player in the front to back.
Signup Backend Support	1 hour 30m	Add player signup functionality to backend with all required data wanted in order to provide with player with best experience.
Login Backend Support	30m	Adding player login with checking if the email and passwords are correct functionality to backend.
Handle the online booked reservations on the backend	1 day	for the player, the Controller and service of the process that the player send the picture of payment for his reservation and send it to the courtOwner to review and set payment to “need approval“
Handle the confirmation process of a reservation on the backend	1 day	for the CourtOwner, after the player pay and confirm the payment the courtOwner need to review that it is correct payment and approve the reservation and set it booked
GUI for the	3 hours	The player can search for any court owner by their name

cancellation of a pending or booked reservation		& the court owners that match the result will be ordered by their distance to the player's current location.
Search for Court owner	3 hours	New attributes are added to the ticket model. These new attributes introduced new features and enhanced UX.
Player services backend	2 hours	Manage the player's main functionalities like seeing courts & managing reservations.

Git/Github Report

Branches, Pull Requests, Code Review Comments

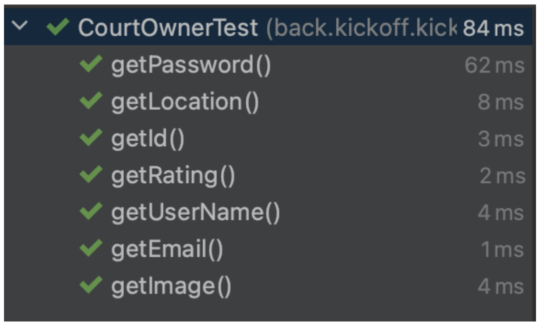
Unit Testing Report

Unit tests are created to test all our components in the following backend packages:

- **Model package**

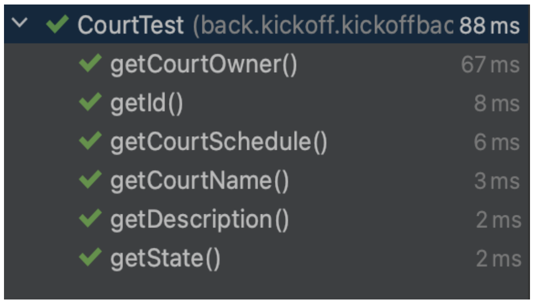
  Unit tests are created in that package to ensure the attributes of each model class are persisted and stored correctly as they are essential in our business logic. The classes that are attributes are tested are the following:

  - *CourtOwner*



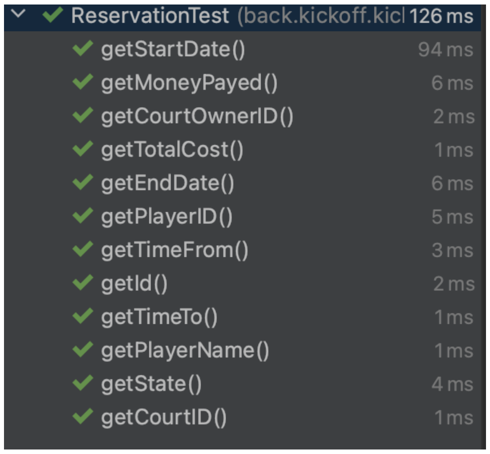
```
✓ CourtOwnerTest (back.kickoff.kick 84 ms
    ✓ getPassword()            62 ms
    ✓ getLocation()             8 ms
    ✓ getId()                   3 ms
    ✓ getRating()               2 ms
    ✓ getUserName()             4 ms
    ✓ getEmail()                1 ms
    ✓ getImage()                4 ms
```

  - *Court*



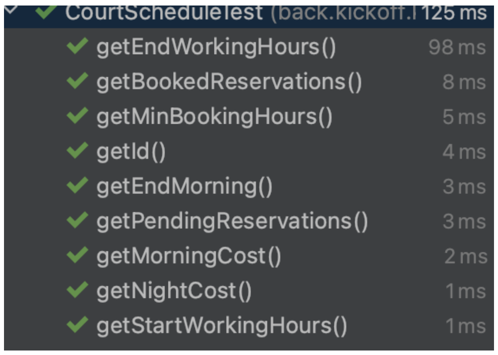
```
✓ CourtTest (back.kickoff.kickoffbac 88 ms
    ✓ getCourtOwner()              67 ms
    ✓ getId()                       8 ms
    ✓ getCourtSchedule()            6 ms
    ✓ getCourtName()                3 ms
    ✓ getDescription()              2 ms
    ✓ getState()                    2 ms
```

- *Reservation*



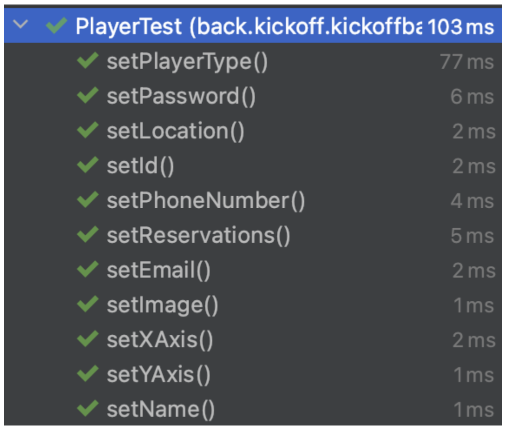
```
✓ ReservationTest (back.kickoff.kicl 126 ms
    ✓ getStartDate()               94 ms
    ✓ getMoneyPayed()               6 ms
    ✓ getCourtOwnerID()             2 ms
    ✓ getTotalCost()                1 ms
    ✓ getEndDate()                  6 ms
    ✓ getPlayerID()                 5 ms
    ✓ getTimeFrom()                 3 ms
    ✓ getId()                       2 ms
    ✓ getTimeTo()                   1 ms
    ✓ getPlayerName()               1 ms
    ✓ getState()                    4 ms
    ✓ getCourtID()                  1 ms
```

- *CourtSchedule*



```
✓ CourtScheduleTest (back.kickoff.l 125 ms
    ✓ getEndWorkingHours()         98 ms
    ✓ getBookedReservations()       8 ms
    ✓ getMinBookingHours()          5 ms
    ✓ getId()                       4 ms
    ✓ getEndMorning()               3 ms
    ✓ getPendingReservations()      3 ms
    ✓ getMorningCost()              2 ms
    ✓ getNightCost()                1 ms
    ✓ getStartWorkingHours()        1 ms
```

- *Player*

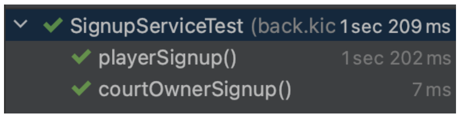


```
✓ PlayerTest (back.kickoff.kickoffba 103 ms
    ✓ setPlayerType()              77 ms
    ✓ setPassword()                 6 ms
    ✓ setLocation()                 2 ms
    ✓ setId()                       2 ms
    ✓ setPhoneNumber()              4 ms
    ✓ setReservations()             5 ms
    ✓ setEmail()                    2 ms
    ✓ setImage()                    1 ms
    ✓ setXAxis()                    2 ms
    ✓ setYAxis()                    1 ms
    ✓ setName()                     1 ms
```

- **Service package**

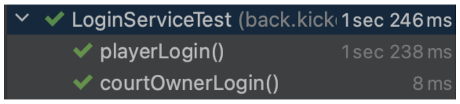
Unit tests are created in that package to ensure that each component in the service package is doing its required functionality efficiently and provide its necessary services and interacting with other packages correctly in the following test cases . The components that are attributes are tested are the following:

- *SignupService*



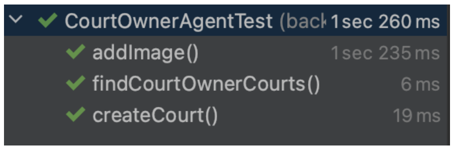
```
✓ SignupServiceTest (back.kic 1 sec 209 ms
    ✓ playerSignup()          1 sec 202 ms
    ✓ courtOwnerSignup()              7 ms
```

- *LoginService*



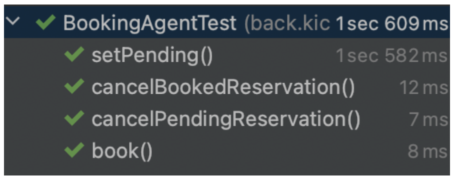
```
✓ LoginServiceTest (back.kick 1 sec 246 ms
    ✓ playerLogin()           1 sec 238 ms
    ✓ courtOwnerLogin()               8 ms
```

- *CourtOwnerAgent*

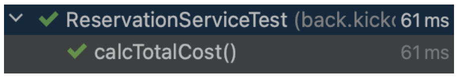


```
✓ CourtOwnerAgentTest (back 1 sec 260 ms
    ✓ addImage()            1 sec 235 ms
    ✓ findCourtOwnerCourts()        6 ms
    ✓ createCourt()                19 ms
```

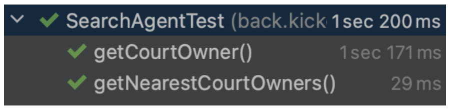
- *BookingAgent*


```
BookingAgentTest (back.kic 1 sec 609 ms
    setPending()                 1 sec 582 ms
    cancelBookedReservation()          12 ms
    cancelPendingReservation()          7 ms
    book()                              8 ms
```

- *ReservationService*


```
ReservationServiceTest (back.kick 61 ms
    calcTotalCost()                  61 ms
```

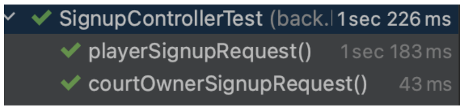
- *SearchAgent*


```
SearchAgentTest (back.kick 1 sec 200 ms
    getCourtOwner()          1 sec 171 ms
    getNearestCourtOwners()         29 ms
```

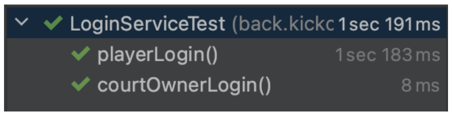
- **Controller package**

  Unit tests are created in that package to ensure that each controller interacts with desired services to provide correct responses towards the user requests coming from the view layer with data provided. The components that are attributes are tested are the following:

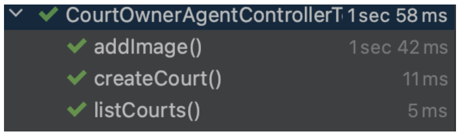
  - *SignupController*


```
SignupControllerTest (back.l 1 sec 226 ms
    playerSignupRequest()       1 sec 183 ms
    courtOwnerSignupRequest()          43 ms
```

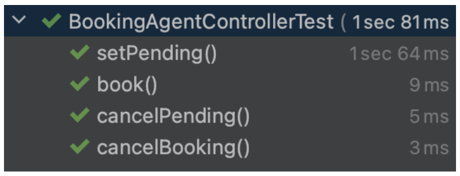
  - *LoginController*


```
LoginServiceTest (back.kickc 1 sec 191 ms
    playerLogin()              1 sec 183 ms
    courtOwnerLogin()                  8 ms
```

  - *CourtOwnerAgentController*

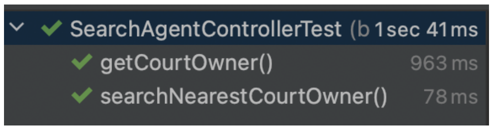

```
CourtOwnerAgentControllerT 1 sec 58 ms
    addImage()                 1 sec 42 ms
    createCourt()                   11 ms
    listCourts()                     5 ms
```

- *BookingAgentController*



```
✓ BookingAgentControllerTest ( 1 sec 81 ms
    ✓ setPending()               1 sec 64 ms
    ✓ book()                            9 ms
    ✓ cancelPending()                   5 ms
    ✓ cancelBooking()                   3 ms
```

- *SearchAgentController*



```
✓ SearchAgentControllerTest (b 1 sec 41 ms
    ✓ getCourtOwner()                 963 ms
    ✓ searchNearestCourtOwner()        78 ms
```

Collaborators & Links

Scrum master for sprint one was ***Zeyad Zidan***.

Collaborators

• **Zeyad Zidan**	**19015709**	**Github**
• **Abdelrahman Aly**	**19015893**	**Github**
• **Abdelrahman Gad**	**19015894**	**Github**
• **Abdel-Aziz Mohammed**	**19015941**	**Github**
• **Omar Khairat**	**19016063**	**Github**

Project

- **Github Repository**
- **JIRA Repository**
- **Firebase Repository**