CHRIST

(DEEMED TO BE UNIVERSITY)

BANGALORE · INDIA

INBOUND

by

Arjun Joshua (1741009)

Thomas Abraham (1741059)

Under the guidance of

Dr Arokia Paul Rajan
&
Prof Rupali Singh Wagh

A Project Lab record submitted in partial fulfilment of
the requirements for the award of degree of Bachelor of
Computer Applications of CHRIST (Deemed to be University)

March – 2020

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

CERTIFICATE

This is to certify that the report titled ***Inbound*** *is a bon a fide record of work done by* ***Arjun Joshua (1741009) and Thomas Abraham (1741059)*** *of CHRIST (Deemed to be University), Bangalore, in partial fulfilment of the requirements of VI Semester BCA during the year 2020 .*

Head of the Department **Faculty in Charge**

Valued-by:	Name	: Arjun Joshua Thomas Abraham
1.	Register Number(s)	: 1741009 1741059
2.	Examination Centre	: CHRIST (Deemed to be University)
	Date of Exam	:

ACKNOWLEDGEMENT

While working on this project we were accompanied and supported by our peers and the teaching faculty. This is a very pleasant aspect for us as now, we have the opportunity to express our gratitude towards all of them for their constant guidance and encouragement. Our effort would not have been successful if it was not for them. First and foremost, we express our deep sense of gratitude to **Dr Fr Abraham V M**, Vice Chancellor of CHRIST (Deemed to be University) for providing the necessary facilities that helped us in many ways to accomplish this project. We feel a deep sense of gratitude to **Dr Fr Joseph CC**, Pro-Vice Chancellor of CHRIST (Deemed to be University) for his moral support.

We would also like to thank our head of the department **Prof Joy Paulose**, Department of Computer Science, CHRIST(Deemed to be University), for his invaluable support during the course of this project. We acknowledge our sincere thanks to **Dr Deepthi Das**, Coordinator of BCA, Department of Computer Science, for her valuable guidance throughout the course of the project. We are most thankful to our project guides **Dr Arokia Paul R.** and **Rupali Singh Wagh,** Department of Computer Science, CHRIST (Deemed to be University), for guiding us with his tremendous knowledge in the field with true spirit throughout the project. Last but not least, we are thankful to have help received from all our friends and classmates. We would also like to thank each and everyone who contributed directly or indirectly towards the successful completion of our project.

ABSTRACT

Local basketball matches in Bangalore along with their upcoming players are only followed by those who are aware of the timings as well as the locations of these matches. This results in basketball enthusiasts and fans of the game missing out on opportunities to watch local matches. The purpose of the system is to create a comprehensive application that will help network all members of the basketball community in Bangalore, by providing full virtual support to organizers of tournaments.

Hundreds of aspiring basketball players in Bangalore and other cities play tournaments in hope of recognition. Hundreds of people are interested but do not have the time to attend these games and watch these players. Inbound aims to correct this issue by providing a platform for hosts, players as well as viewers to have live access to scoreboards, statistic sheets, as well as team information and player data.

The primary intention and objective of the inbound application is to create a platform for local basketball enthusiasts to come together and watch local basketball matches as well as follow local players as their game develops. One of the main advancements that will occur through the implementation of inbound is the overall reduction in the amount of paper used to record the various statistics of a basketball match which should benefit the environment.

The inbound application has a future in today's local basketball structure where there are no digital applications being used. In local basketball games in Bangalore where paper is still used to maintain the score as well as individual player statistics of an ongoing match an application such as inbound will be a welcome change as it will reduce the amount of work and paper required in a game consisting of four quarters. Systems such as this have already been implemented in countries where basketball is an extremely popular sport and they have shown great success.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 PROJECT DESCRIPTION

Local basketball in Bangalore receives zero to no media attention. This results in basketball enthusiasts and fans of the game missing out on opportunities to watch local matches. The purpose of the system is to create a comprehensive application that will help network all members of the basketball community in Bangalore, by providing full virtual support to organizers of tournaments. Project will include features such as full customizable rosters, live scoreboards, tournament registration and details.

1.2 EXISTING SYSTEM

The most similar current system available is the application Game On by Sportelos which has limited functionality and a poor interface along with incomplete databases. It poorly designed user interface can be quite difficult to use for the average user who has no prior knowledge of local basketball teams present in Bangalore

1.3 PROPOSED SYSTEM

The proposed system will include a comprehensive environment for organizers hosting a tournament as well as interested participants for registration, statistics and live scoreboards. The system will include full rosters for teams with options for pictures along with live statistics for players. Additionally, the application will contain a section dedicated to individual improvement in basketball. The activities of the user can be monitored through this section by linking the users Google fit profile.

1.4 BENEFITS OF PROPOSED SYSTEM

The mobile application will allow basketball enthusiasts and fans of the game to be up to date on the standings of local basketball leagues happening around them. The project will also incorporate user tracking through the Google fit. This will enable users to keep track of their basketball workouts irrespective of what fitness tracker they use.

2. SYSTEM ANALYSIS AND REQUIREMENTS

2.1 PROBLEM DEFINITION

Hundreds of aspiring basketball players in Bangalore and other cities play tournaments in hope of recognition. Hundreds of people are interested but do not have the time to attend these games and watch these players. Inbound aims to correct this issue by providing a platform for hosts, players as well as viewers to have live access to scoreboards, statistic sheets, as well as team information and player data.

2.2 REQUIREMENT SPECIFICATION

2.2.1 FUNCTIONAL REQUIREMENTS

Requirement ID	Requirement Name	Description
C_F_R1	Scoreboard	The primary feature of the application, i.e. live scoreboards and game tracking should be available 24/7, with no downtime of the network.
C_F_R2	Access	Host, Player and Viewer should be able to access the system with dedicated privilege sets. Each user of the system should have

		his/her own set of functions and dedicated interface.
C_F_R3	Fitness Tracking	The fitness tracker should be able to sync with Google Fit and any smart devices connected to it to pull data regarding fitness information.
C_F_R4	Registration	Registration of teams and entry of team related data should take place without any glitches or errors.

Table 2.1 - Functional Requirements

2.2.2 NON-FUNCTIONAL REQUIREMENTS

Requirement ID	**Requirement Name**	**Description**
C_NF_R1	Performance	As it is a mobile application, the network, hardware and other related infrastructure plays a vital role in determining the application performance. • Number of page navigations are reduced to the minimum.

		• Basic and simple color and graphics settings are implemented to help the performance. • Disabling browsing the server.
C_NF_R2	Security	Performing frequent backup can reduce the data loss due to sudden server or system crash. The user will be able to access individual team and player data only after their security credentials are verified.
C_NF_R2	Quality Requirements • Usability	The interface for each type of user kept very simple and complete for ease of its user. Manuals, demos or the documents made available, simple, clear and definite set of interfaces makes the context easy to understand and use.
	• Reliability	This system uses atomicity features where each task flow is performed to complete a

		process successful, failure of single task halts the transaction and the process fails. This avoids the occurrence of incomplete order processing. Data backup ensures the availability of data to the system users all the time.
	• Availability	This web application handles multiple service requests and the server is up all the time, which is made sure by the robust algorithms and server design architecture.

Table 2.2 - Non-Functional Requirements

2.3 BLOCK DIAGRAM

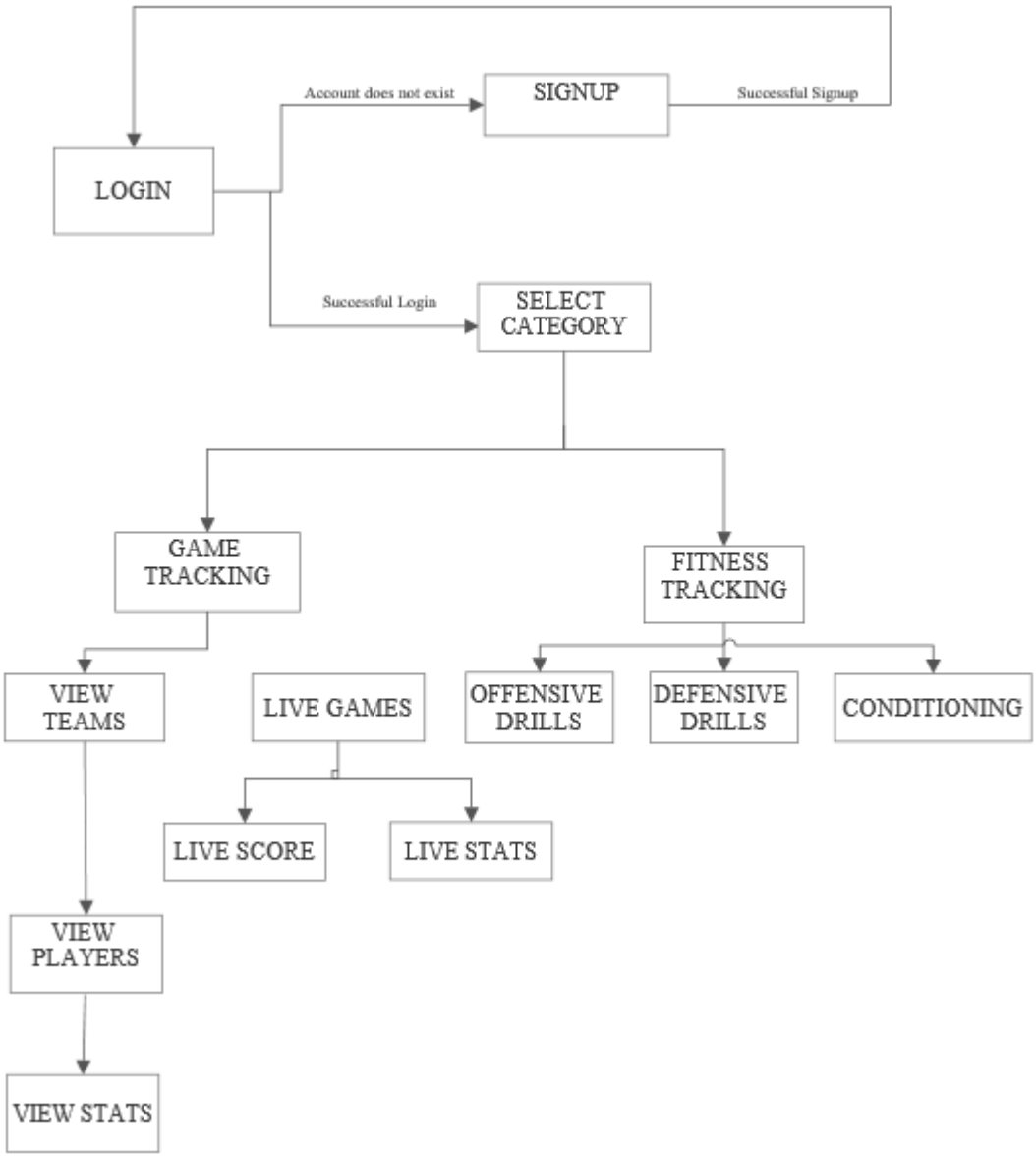

Fig 2.1 - Block Diagram

2.4 SYSTEM REQUIREMENTS

2.4.1 USER STORIES

The high-level client requirements in terms of user stories are given as below:

• The system should provide access to Host, Player and Viewer with defined

privilege set for each.

• The system should provide a safe connection for live scores.

• Host should be able to view, and edit the data from all teams as well as scoreboards.

• Players should be able to register and view information of their own team.

• Viewers should be able to view scorecards and scoreboards as well as statistics sheets and basic player information.

2.4.2 USER CHARACTERS

2.4.2.1 USERS

• Host: Who holds the responsibility of creating, managing and updating data for a tournament as well as updating live scores.

• Host admin: Who holds the high privilege of Inbound and can manage all data and respond to queries and reports of inappropriate data.

• Player: Is an individual player or representative of a team who enters data for each player and is responsible for registration and so on.

• Viewer: Is a user who uses the application to view statistic sheets and live scoreboards.

2.4.3 USE CASE DIAGRAM

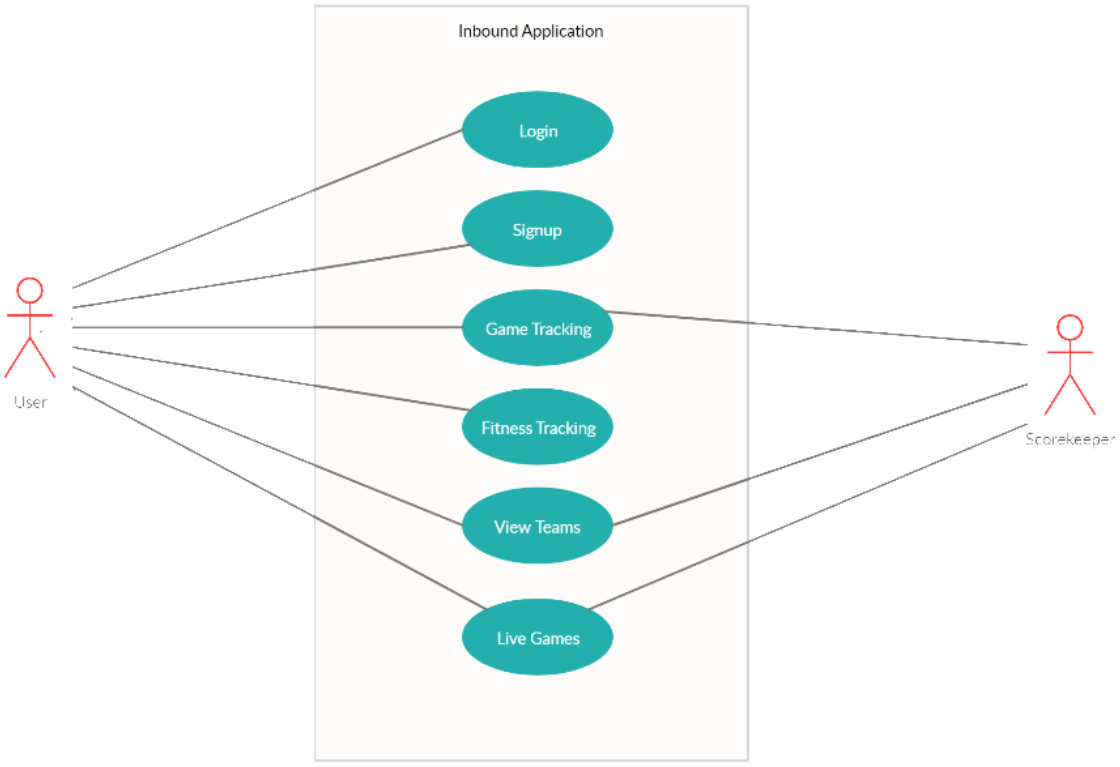

Fig 2.2 - Use Case Diagram

2.4.4 SOFTWARE AND HARDWARE REQUIREMENTS

2.4.4.1 HARDWARE REQUIREMENTS

All the hardware requirements that are necessary in developing and deploying the proposed system are listed below.

Sl No.	Requirement Name	Description
1	Processor	1.6 GHz or higher
2	Hard Disk Space	30 MB or higher
3	RAM	512 MB or higher

Table 2.3 - Hardware Requirements

2.4.4.1 SOFTWARE REQUIREMENTS

Sl No.	Requirement Name	Description
1	Operating System	Android 5.0 (Lollipop)
2	Software Tools Required	Android Studio
3	Database	SQLite

Table 2.4 - Software Requirements

3. SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

3.1.1 SYSTEM PERSPECTIVE

After gathering and understanding the requirements, the system design is performed based on the analysis and feasibility study from previous development phases. The following designs are accepted as a whole. These diagrams give a whole picture of the entire system that is being built.

3.1.2 ARCHITECTURE

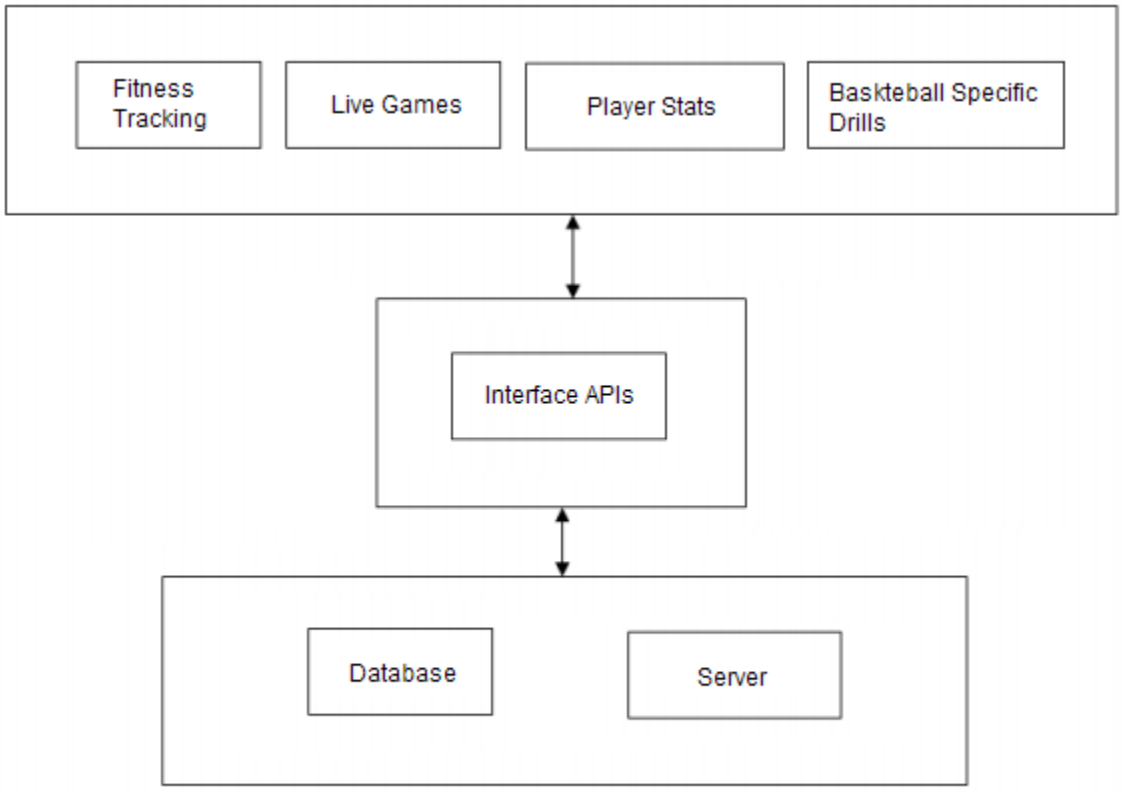

Fig 3.1 - Architecture

The above figure depicts a high-level architecture of the system. The first level holds four different modules that provide a set of functionalities to its user at the front. The middle layer which is the business layer makes an interface connection between the top and bottom layer in serving and processing the user request. The bottom most layer holds the backend database and web server that provides the required information to all the above layers.

3.2 MODULE DESIGN

3.2.1 LIVE SCOREBOARDS

This particular module allows the user to view the live score of an ongoing match. The main complexity of this module lies in data transfer over the interface of multiple teams, database interconnection and coordinating its operations during every play of a live game.

3.2.2 PLAYER STATISTICS

This module will allow the user to view the real time statistics of every active player on a team's roster. The displayed statistics will include field goals attempted, field goals made, assists, rebounds and a wide array of both conventional and advanced basketball statistics. The complexity of this module lies in real time updating of both the team's statistics as well as each individual player statistics

3.2.3 FITNESS TRACKING

This particular module will allow the user to link the application with their Google Fit application, and therefore any smart devices such as watches and fitness trackers. This data will be then used to provide users with advice and information specific to basketball players, regarding how amount of activity is required and reached for a particular day.

3.2.4 BASKETBALL DRILLS

This module will provide users with skill specific basketball training drills, which will progress in difficulty over time. Drills can be chosen very specifically to suit particular skill sets, as well as general basketball training drills.

3.3 DATA FLOW DIAGRAM

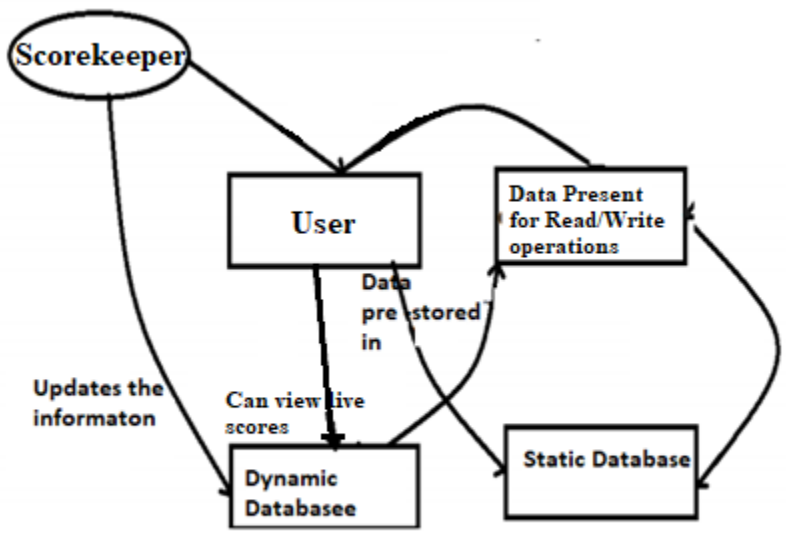

Fig 3.2 - Data Flow Diagram

The above diagram depicts the overall inbound system, composed of all different modules and associated user interface screens and forms associated with corresponding functionality. Players, coaches and casual users can basically perform the functions that clearly mapped with his roles and responsibilities hence a dedicated access level and User Interface is given.

3.4 ER DIAGRAM

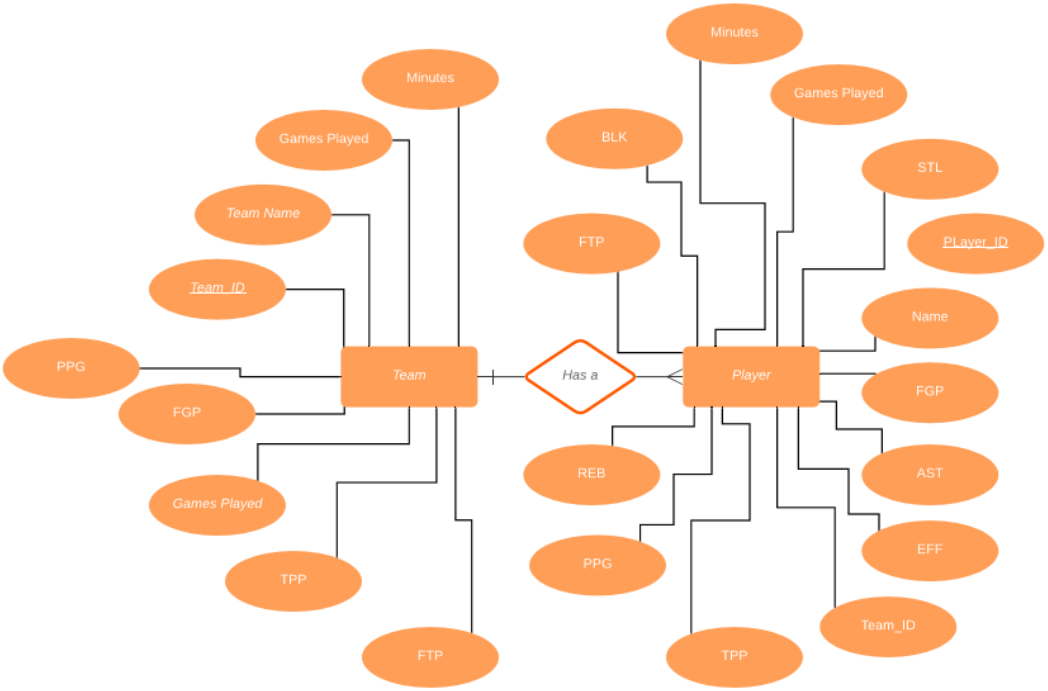

Fig 3.3 - ER Diagram

3.5 DATABASE DESIGN

3.5.1 TABLE DESIGN

Field	Datatype	Constraint
user_id	Varchar	Primary Key
username	Varchar	Not Null
fullname	String	Not Null

Table 3.1 - User Design

Field	Datatype	Constraint
team_id	Varchar	Primary Key
teamname	Varchar	Not Null
number_player	Number	Not Null
coach_name	Varchar	

Table 3.2 - Team Table

Field	Datatype	Constraint
player_id	Varchar	Primary Key
team_id	Varchar	Foreign Key
player_name	Varchar	Not Null
player_number	Number	Not Null
instagram_id	Varchar	

Table 3.3 - Player Table

3.5.2 CONFIGURATION

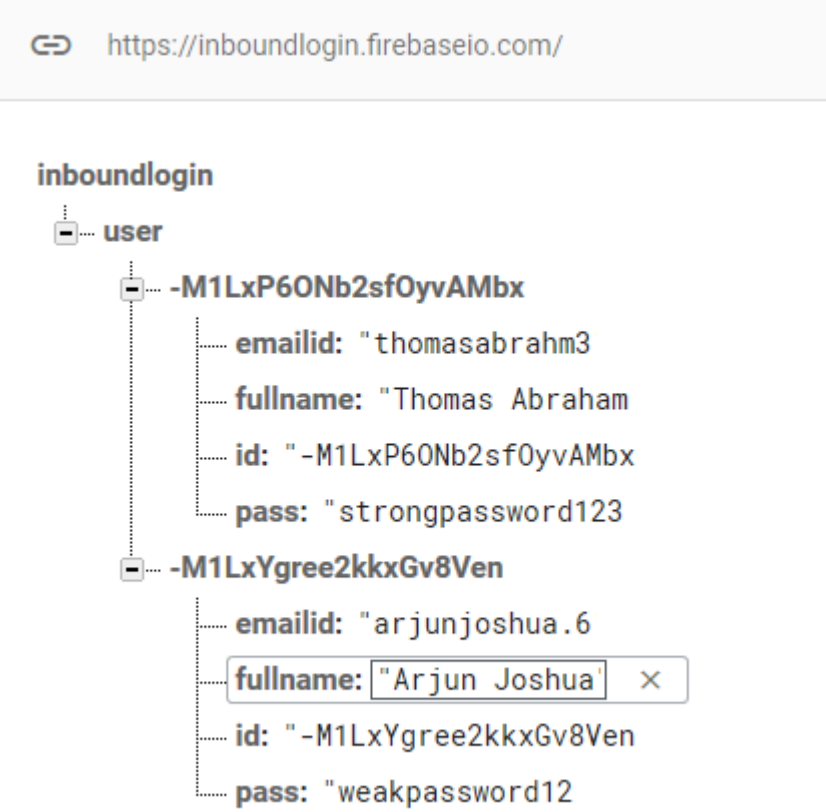

Fig 3.4 User Table Configuration

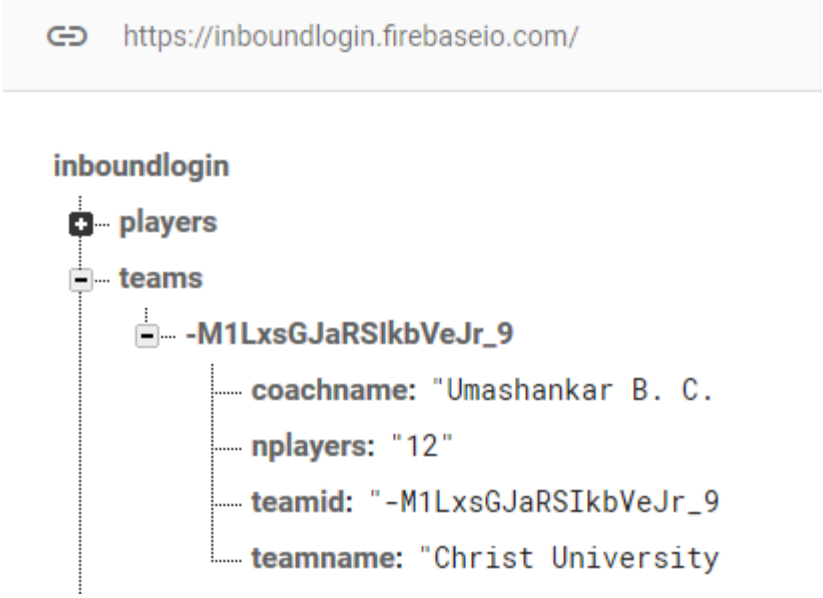

Fig 3.5 Team Table Configuration

Fig 3.6 Player Table Configuration

3.6 INTERFACE AND PROCEDURAL DESIGN

3.6.1 USER INTERFACE DESIGN

The Inbound project is being designed for the following users, which provides user specific interface features to perform role-based tasks.

a. **Users (Players, Coaches and Casual Users):** He/She is the primary user of this application, who accesses this application to perform the tasks like adding teams, removing teams, and viewing live scores and statistics in the realtime database. Four modules were developed as mentioned above to support these needs.

b. **Administrator (Scorekeeper) :** Being the scorekeeper of the application, this individual will be given access to the data of all teams registered to a particular match, league or tournament. He or she will then use this data to alter live scores which will appear on the database as well as to the users of the app.

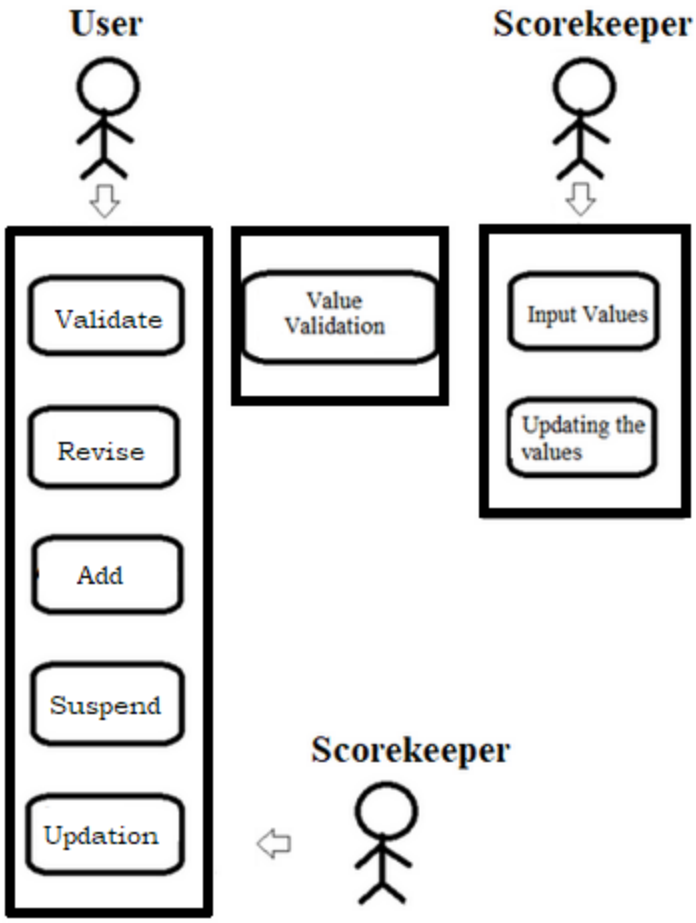

Fig 3.7 - User Interface Design

3.6.2 USER INTERFACE SCREENSHOTS

The following images are screenshots of the mobile applications user interface

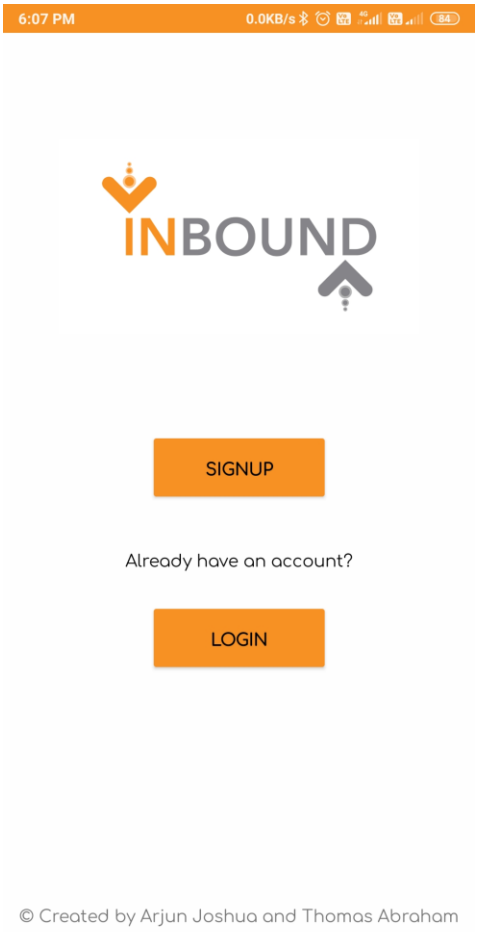

Fig 3.8 Main Activity Page

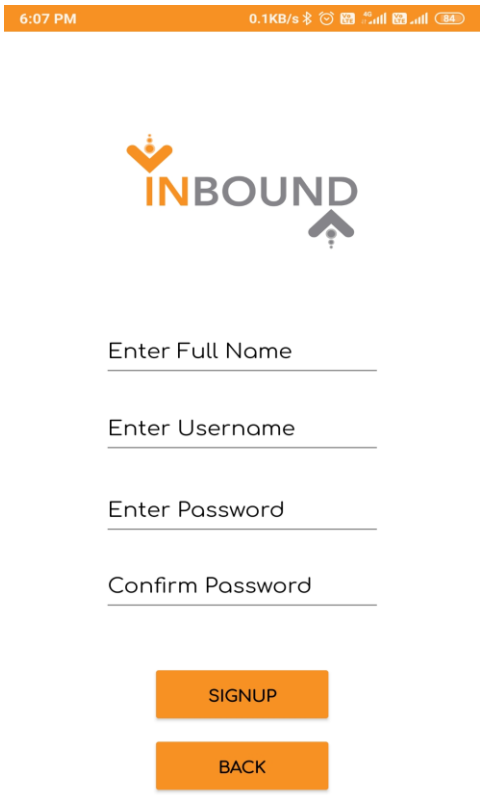

Fig 3.9 User Signup Page

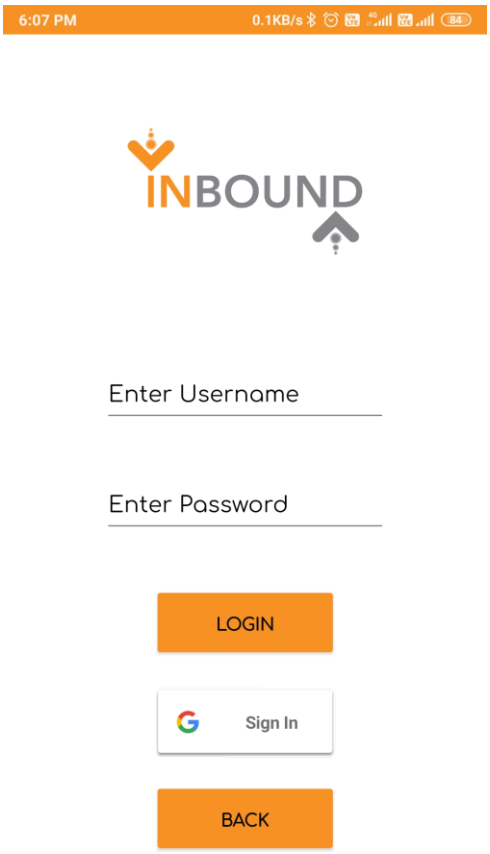

Fig 3.10 User Login Page

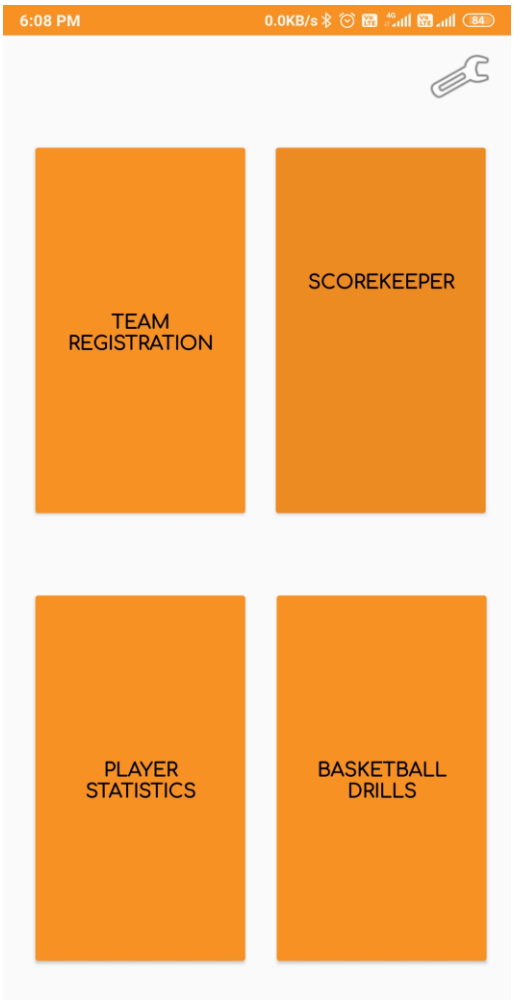

Fig 3.11 Home Page of Application

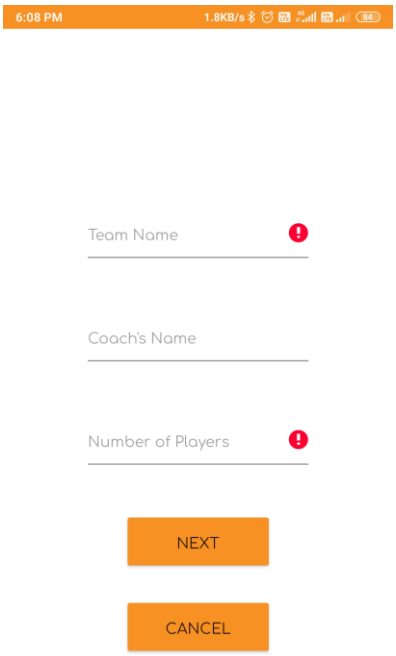

Fig 3.11 Team Registration Page

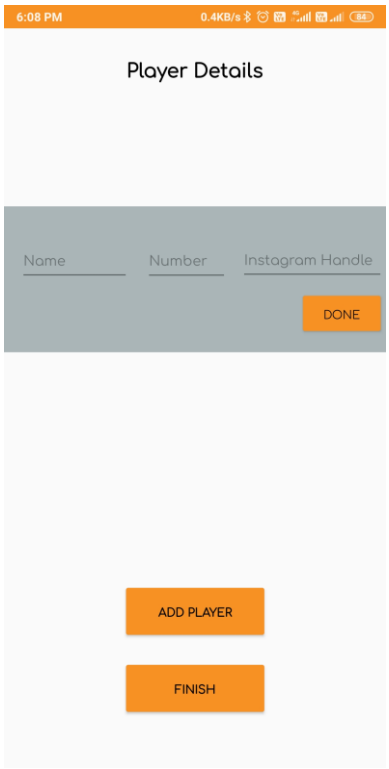

Fig 3.12 Player Registration Page

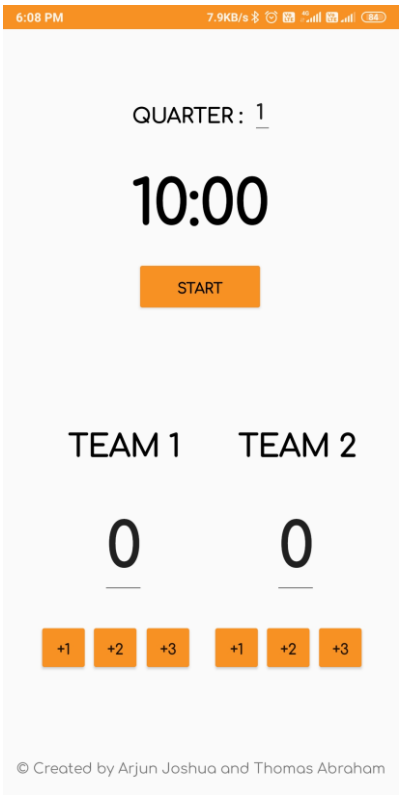

Fig 3.13 Scorekeeper Page

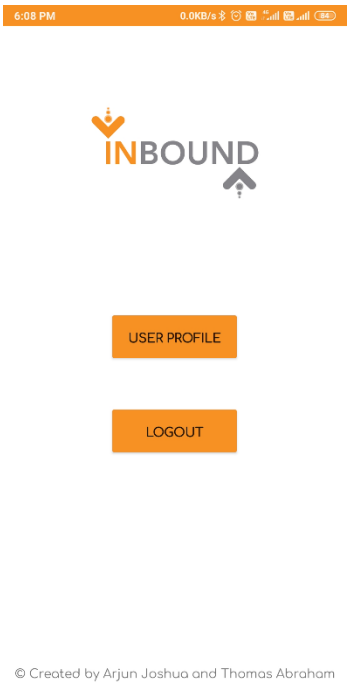

Fig 3.14 User Settings Page

4. IMPLEMENTATION

4.1 IMPLEMENTATION APPROACH

During implementation of this application faced a pile of challenges and hurdles during the initial stages of system analysis and development. Bringing various technologies and concepts like realtime database and Google API's helped the team to better understand the application requirements over the time and deliver and the product on installments in multiple timeframes. Testing is extensively done at various stages of development process and testing life cycle, making sure quality application is delivered in order to ensure customer satisfaction.

The below brief information tells the stages of implementing this solution.

• Gathered requirement were being refined and development started with the UI architecture. Modules were derived from the requested UI forms and functionalities.

• Logic was framed for each functional requirement.

• Then database integration was done to connect all the modules data flow.

• All the above components were glued together into a single system.

System was being tested at various testing phases.

4.2 CODING STANDARD

a. Java:

Java makes up the entire business logic that serves the database interface and main functional workflows requirements. Since java is a secure and platform independent in nature it is pulled in to address the needs of secure logical functions and secure database interactions.

b. XML:

All user interface additions and modifications in Android Studio are done through XML files. The manipulation of code in the XML file can result in various UI changes to every object or layout present on that particular page. Every XML file is linked to a page consisting of Java code which is used to perform actions on the page.

c. Android Studio:

Android Studio IDE is being used in developing java layer implementations, as it gives all the java development features and environment to develop and maintain java code. Android Studio and java are freely available technologies. Android Studio provides extended feature of an editor like syntax checker and dependency suggestions makes it friendlier in developing java code.

d. Firebase:

Firebase is a Backend-as-a-Service — BaaS — that started as a YC11 startup and grew up into a next-generation app-development platform on Google Cloud Platform. It allows user to pull various API's as well as use its realtime database or cloud store database

4.3 CODING DETAILS

4.3.1 Validation

4.3.1.1

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_inbound_signup);

        username = findViewById(R.id.username);
        password = findViewById(R.id.password);
        confirm = findViewById(R.id.confpass);
        fname = findViewById(R.id.fullname);

        databaseUsers= FirebaseDatabase.getInstance().getReference("user");

        Button next = (Button) findViewById(R.id.button5);
        next.setOnClickListener((view) -> {
        if(confirm.getText().toString().equals(password.getText().toString())) {
                writeNewUser();
```

```
            Intent i = new Intent(this, Inbound_login.class);
            startActivity(i);
        }
        else
        {
            confirm.setError("Passwords Do Not Match");
        }
        });

        Button cancel = (Button) findViewById(R.id.button6);
        cancel.setOnClickListener((view) -> {
        Intent j = new Intent(this, MainActivity.class);
        startActivity(j);

        Toast.makeText(this, "Signup Cancelled", Toast.LENGTH_SHORT).show();
        });
        }
```

4.3.1.2

```
@Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_team_registration);
        next=(Button) findViewById(R.id.btnnext);
        cancel=(Button) findViewById(R.id.btncancel);
        team=(EditText) findViewById(R.id.TeamName);
        coach=(EditText) findViewById(R.id.CoachName);
        playerno=(EditText) findViewById(R.id.PlayerNo);

        databaseTeams = FirebaseDatabase.getInstance().getReference("teams");
        next.setOnClickListener(v -> {
```

```
if( team.getText().toString().length() == 0 || playerno.getText().toString().length()==0)
{
        team.setError("Please Enter Team Name");
        playerno.setError("Please Enter Number of Players");
}
else {
        writeNewTeam();
        Intent i = new Intent(this, PlayerInfo.class);
        i.putExtra("teamname",team.getText().toString());
        startActivity(i);
}
});
cancel.setOnClickListener(v-> {
Intent i = new Intent(this, HomePage.class);
startActivity(i);

Toast.makeText(this, "Team Registration Cancelled", Toast.LENGTH_SHORT).show();
});

}

private void writeNewTeam(){
String teamn = team.getText().toString();
String coachn = coach.getText().toString();
String nplay = playerno.getText().toString();
String id = databaseTeams.push().getKey();

Teams team = new Teams(id,teamn,coachn,nplay);

databaseTeams.child(id).setValue(team);
}
```

```
}
```

4.3.1.3

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_profile);

        String id = getIntent().getStringExtra("useid");

        userprofile = findViewById(R.id.userprofile);
        nam=findViewById(R.id.name);
        usernam=findViewById(R.id.username);
        pass=findViewById(R.id.oldpassword);
        newpass=findViewById(R.id.newpassword);
        confpass=findViewById(R.id.confirmpassword);
        changepass=findViewById(R.id.cbpassword);
        btnupdate=findViewById(R.id.update);

        databaseUsers= FirebaseDatabase.getInstance().getReference("user").child(id);

        changepass.setOnClickListener(v -> {
        TransitionManager.beginDelayedTransition(userprofile);
        pass.setVisibility(View.VISIBLE);
        btnupdate.setVisibility(View.VISIBLE);
        newpass.setVisibility(View.VISIBLE);
        confpass.setVisibility(View.VISIBLE);
        changepass.setVisibility(View.GONE);
        });
        getData(id);
```

```
//Button update = (Button) findViewById(R.id.update);
btnupdate.setOnClickListener((view) -> {
if(confpass.getText().toString().equals(newpass.getText().toString())) {
        checkData(id);
        Toast.makeText(this, "User Details Updated", Toast.LENGTH_SHORT).show();
        Intent i = new Intent(this, Inbound_login.class);
        startActivity(i);
}
else
{
        confpass.setError("Passwords Do Not Match");
}
});

}

private void getData(final String useid){
databaseUsers.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

        Users user = dataSnapshot.getValue(Users.class);

        if(user.getId().matches(useid))
        {
        usernam.setText(user.getEmailid());
        //pass.setText(user.getPass());
        nam.setText(user.getFullname());
        }

}
```

```
@Override
public void onCancelled(@NonNull DatabaseError databaseError) {

}
});
}

private void checkData(final  String useid){
databaseUsers.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        Users user = dataSnapshot.getValue(Users.class);

        if (pass.getText().toString().matches(user.getPass()))
        {
        updateData(useid);
        }
        else
        {
        pass.setError("Original Password Incorrect");
        }
}

@Override
public void onCancelled(@NonNull DatabaseError databaseError) {

}
});
}
private void updateData(final String useid){
```

```
        String username = usernam.getText().toString();
        String password = newpass.getText().toString();
        String name = nam.getText().toString();

        Users user = new Users(username,password,name,useid);

        databaseUsers.setValue(user);
        }
}
```

4.3.2 Team Addition

```
@Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_team_registration);
        next=(Button) findViewById(R.id.btnnext);
        cancel=(Button) findViewById(R.id.btncancel);
        team=(EditText) findViewById(R.id.TeamName);
        coach=(EditText) findViewById(R.id.CoachName);
        playerno=(EditText) findViewById(R.id.PlayerNo);

        databaseTeams = FirebaseDatabase.getInstance().getReference("teams");
        next.setOnClickListener(v -> {
        if( team.getText().toString().length() == 0 || playerno.getText().toString().length()==0)
        {
                team.setError("Please Enter Team Name");
                playerno.setError("Please Enter Number of Players");
        }
        else {
```

```
            writeNewTeam();
            Intent i = new Intent(this, PlayerInfo.class);
            i.putExtra("teamname",team.getText().toString());
            //i.putExtra("Player number", playerno.getText());
            startActivity(i);
        }
        });
        cancel.setOnClickListener(v-> {
        Intent i = new Intent(this, HomePage.class);
        startActivity(i);

        Toast.makeText(this, "Team Registration Cancelled", Toast.LENGTH_SHORT).show();
        });

        }

        private void writeNewTeam(){
        String teamn = team.getText().toString();
        String coachn = coach.getText().toString();
        String nplay = playerno.getText().toString();
        String id = databaseTeams.push().getKey();

        Teams team = new Teams(id,teamn,coachn,nplay);

        databaseTeams.child(id).setValue(team);
        }

}
```

4.3.3 View Player Statistics

```
public class PlayerInfoAdd extends Fragment {

        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                 Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_player_info_add, container, false);
        }
        @Override
        public void onViewCreated(View view, Bundle savedInstanceState) {
        EditText name=(EditText)getView().findViewById(R.id.Player_Name);
        EditText number=(EditText)getView().findViewById(R.id.Player_Number);
        EditText IGid=(EditText)getView().findViewById(R.id.Player_id);
        Button done=(Button)getView().findViewById(R.id.btndone);

        done.setOnClickListener(v ->{
        PlayerInfo.Tname[PlayerInfo.count]=name.getText().toString();
        PlayerInfo.Tid[PlayerInfo.count]=IGid.getText().toString();
        PlayerInfo.Tnumber[PlayerInfo.count]=number.getText().toString();
        getActivity().onBackPressed();
        });
        }
        }


@Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_player_info);

        databasePlayers = FirebaseDatabase.getInstance().getReference("players");
```

```
Button addPlayer = (Button) findViewById(R.id.btnadd);
final FrameLayout f = (FrameLayout) findViewById(R.id.frameLayout);
EditText name1 = (EditText) findViewById(R.id.Name1);
EditText number1 = (EditText) findViewById(R.id.number1);
EditText id1 = (EditText) findViewById(R.id.id1);
EditText name2 = (EditText) findViewById(R.id.Name2);
EditText number2 = (EditText) findViewById(R.id.number2);
EditText id2 = (EditText) findViewById(R.id.id2);
EditText name3 = (EditText) findViewById(R.id.Name3);
EditText number3 = (EditText) findViewById(R.id.number3);
EditText id3 = (EditText) findViewById(R.id.id3);
EditText name4 = (EditText) findViewById(R.id.Name4);
EditText number4 = (EditText) findViewById(R.id.number4);
EditText id4 = (EditText) findViewById(R.id.id4);
EditText name11 = (EditText) findViewById(R.id.Name11);
EditText number11 = (EditText) findViewById(R.id.number11);
EditText id11 = (EditText) findViewById(R.id.id11);
EditText name12 = (EditText) findViewById(R.id.Name12);
EditText number12 = (EditText) findViewById(R.id.number12);
EditText id12 = (EditText) findViewById(R.id.id12);
addPlayer.setOnClickListener((view) -> {
if (count == 1) {
        name1.setText(Tname[count]);
        number1.setText(Tnumber[count]);
        id1.setText(Tid[count]);
        name1.setVisibility(View.VISIBLE);
        number1.setVisibility(View.VISIBLE);
        id1.setVisibility(View.VISIBLE);
} else if (count == 2) {
        name2.setText(Tname[count]);
```

```
                number2.setText(Tnumber[count]);
                id2.setText(Tid[count]);
                name2.setVisibility(View.VISIBLE);
                number2.setVisibility(View.VISIBLE);
                id2.setVisibility(View.VISIBLE);
        } else if (count == 3) {
                name3.setText(Tname[count]);
                number3.setText(Tnumber[count]);
                id3.setText(Tid[count]);
                name3.setVisibility(View.VISIBLE);
                number3.setVisibility(View.VISIBLE);
                id3.setVisibility(View.VISIBLE);
        } else if (count == 12) {
                name12.setText(Tname[count]);
                number12.setText(Tnumber[count]);
                id12.setText(Tid[count]);
                name12.setVisibility(View.VISIBLE);
                number12.setVisibility(View.VISIBLE);
                id12.setVisibility(View.VISIBLE);
        }
        count++;
        FragmentTransaction fragmentTransaction =
getSupportFragmentManager().beginTransaction();
        fragmentTransaction.add(R.id.frameLayout, new PlayerInfoAdd());
        fragmentTransaction.addToBackStack(null);
        fragmentTransaction.commit();
        });

        Button finish = (Button) findViewById(R.id.btnfinish);
        finish.setOnClickListener((view) -> {
        Intent j = new Intent(this, HomePage.class);
```

```
        startActivity(j);

                for(int i=0; i<count; i++)
                {
                writeNewPlayer(i);
                }

                Toast.makeText(this, "Team Registration Complete",
Toast.LENGTH_SHORT).show();

                Tname=null;
                Tnumber=null;
                Tid=null;
        });

        }

        private void writeNewPlayer(int i) {
        String pname=Tname[i];
        String pnumber=Tnumber[i];
        String pinstagram=Tid[i];
        String id=databasePlayers.push().getKey();

        Players obj = new Players(pname,pnumber,pinstagram,id);

        databasePlayers.child(id).setValue(obj);
        }
}
```

4.3.3 Database Structure

4.3.3.1 User Database

```
package com.example.inboundlogin;

public class Users {

    String emailid;
    String pass;
    String fullname;
    String id;


    public Users(){

    }

    public Users(String emailid, String pass, String fullname, String id) {
    this.emailid = emailid;
    this.pass = pass;
    this.fullname = fullname;
    this.id = id;
    }

    public String getEmailid() {
    return emailid;
    }

    public String getPass() {
```

```
        return pass;
        }

        public String getFullname() { return fullname; }

        public String getId() {
        return id;
        }
}
```

4.3.3.2 Team Database

```
package com.example.inboundlogin;

public class Teams {
        String teamid;
        String teamname;
        String coachname;
        String nplayers;

        public Teams(){

        }

        public Teams(String teamid, String teamname, String coachname, String nplayers) {
        this.teamid = teamid;
        this.teamname = teamname;
        this.coachname = coachname;
        this.nplayers = nplayers;
        }
```

```
	public String getTeamid() {
	return teamid;
	}

	public String getTeamname() {
	return teamname;
	}

	public String getCoachname() {
	return coachname;
	}

	public String getNplayers() {
	return nplayers;
	}
}
```

4.3.3.3 Player Database

```
package com.example.inboundlogin;

public class Players {
	String name;
	String number;
	String instagram;
	String playerid;

	public Players(){

	}
```

```
        public Players(String name, String number, String instagram, String playerid) {
        this.name = name;
        this.number = number;
        this.instagram = instagram;
        this.playerid = playerid;
        }

        public String getName() {
        return name;
        }

        public String getNumber() {
        return number;
        }

        public String getInstagram() {
        return instagram;
        }

        public String getPlayerid() {
        return playerid;
        }
}
```

4.4 SCREENSHOTS (CODING)

The following screenshots consist of various Java and XML files used to create the application in Android Studio. Functions used to communicate with Firebase Database are also added.

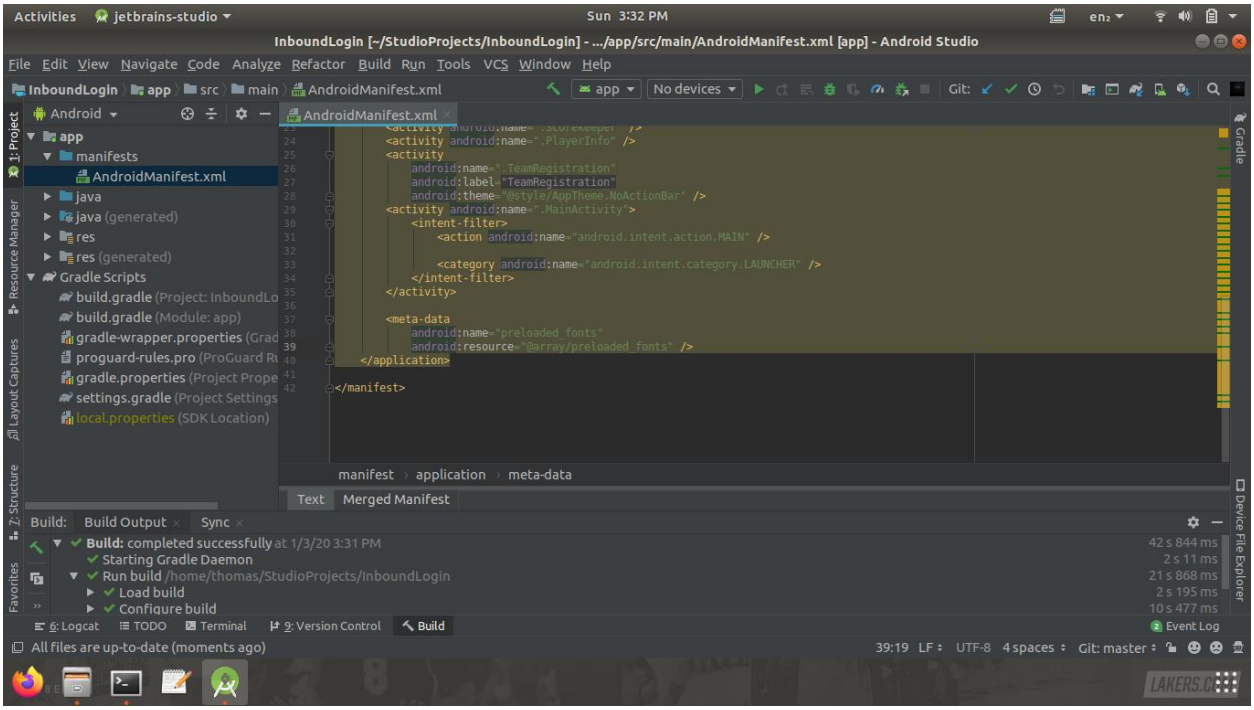

Fig 4.1 Android Manifest File

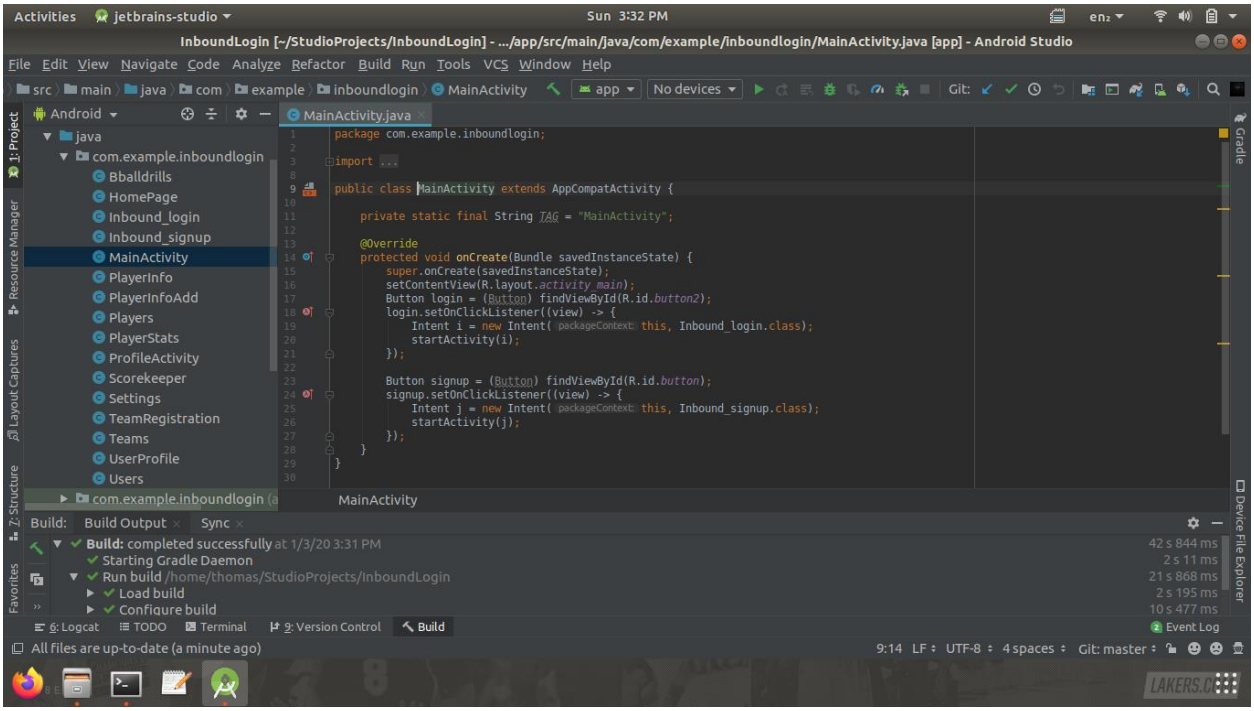

Fig 4.2 Main Activity File

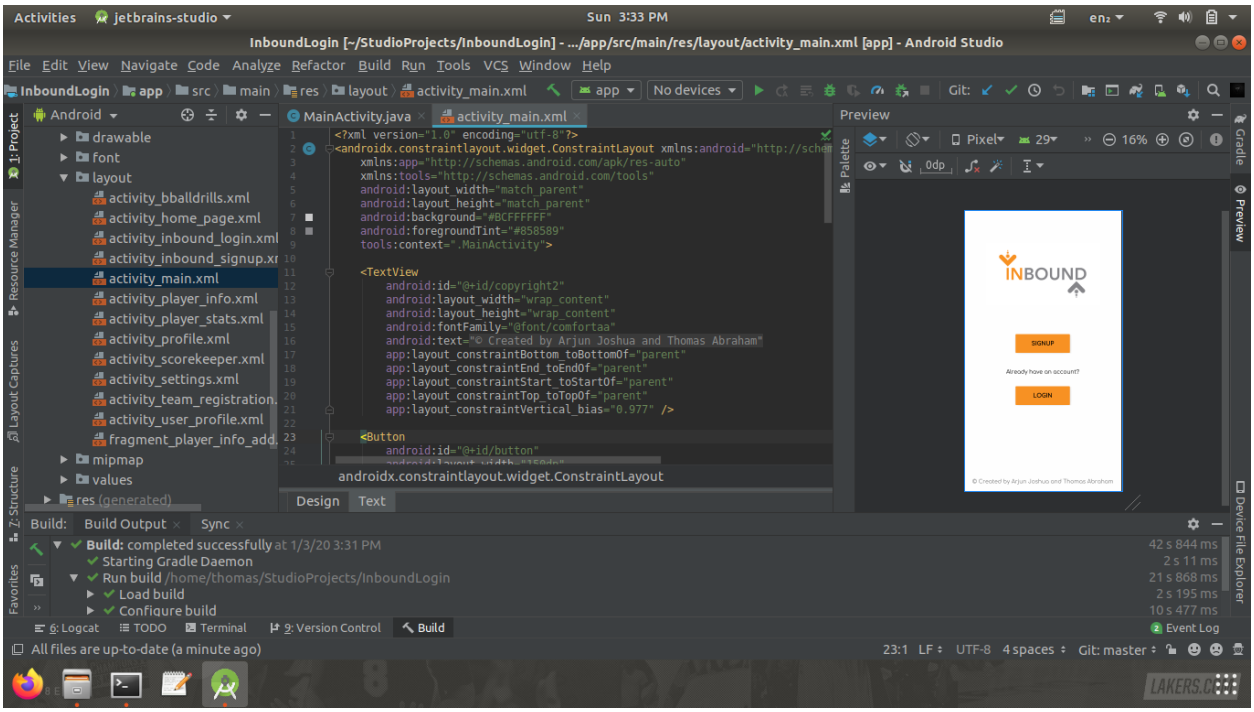

Fig 4.3 Main Activity XML File

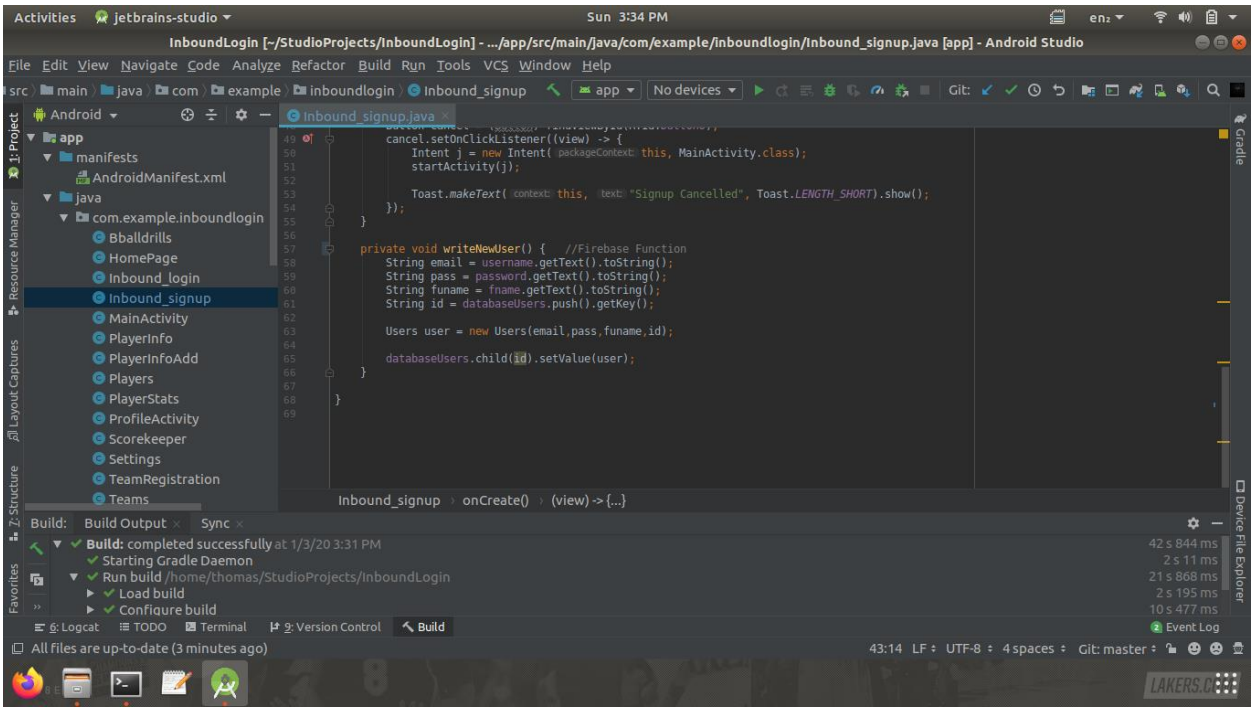

Fig 4.4 Signup Page

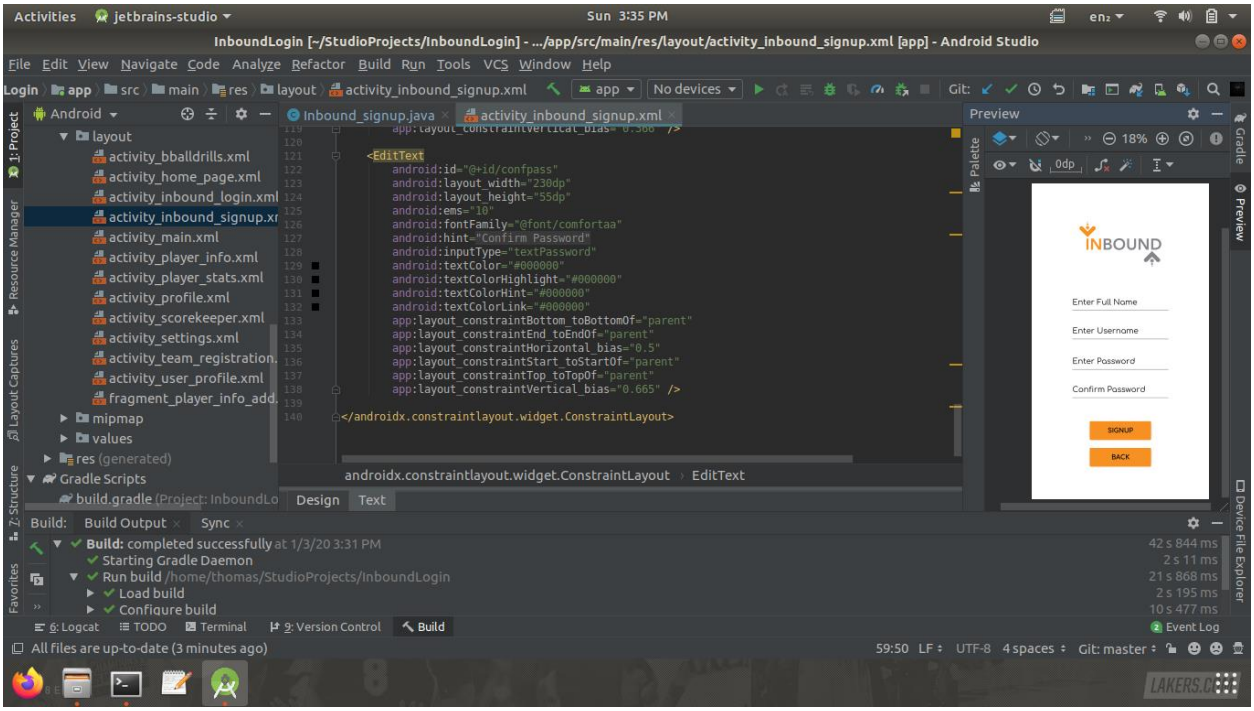

Fig 4.5 Signup XML Page

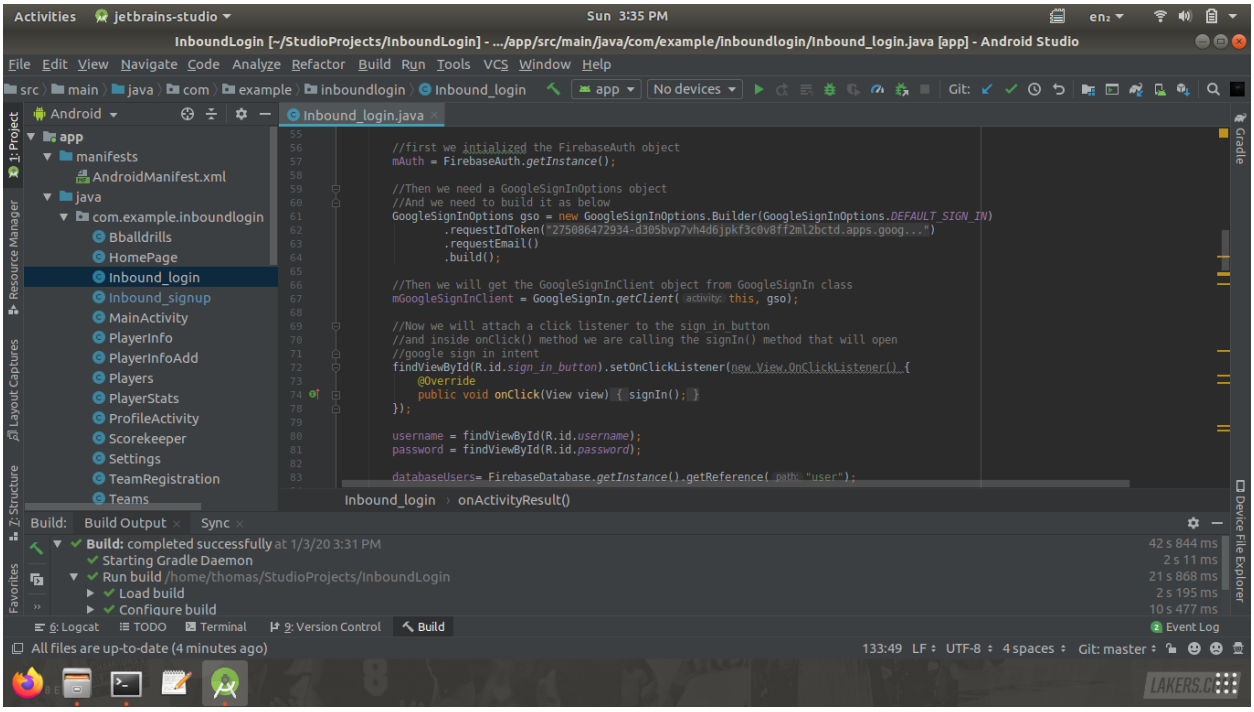

Fig 4.6 Login Page

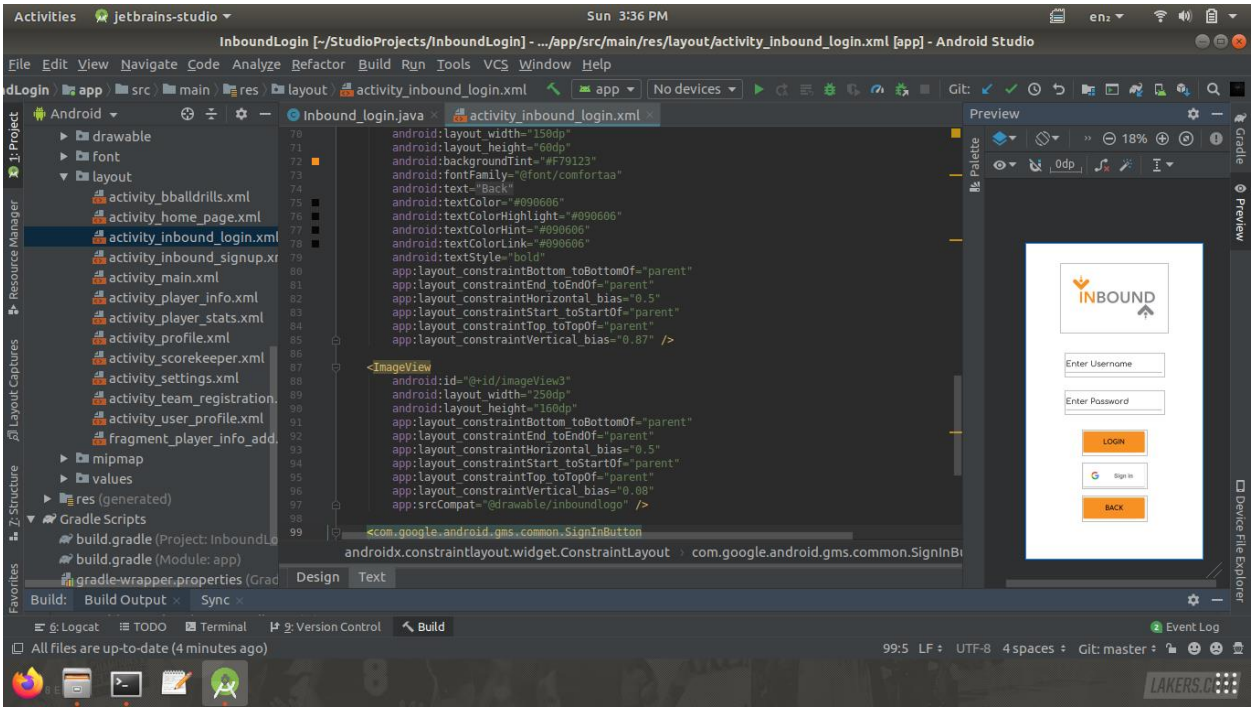

Fig 4.7 Login XML Page

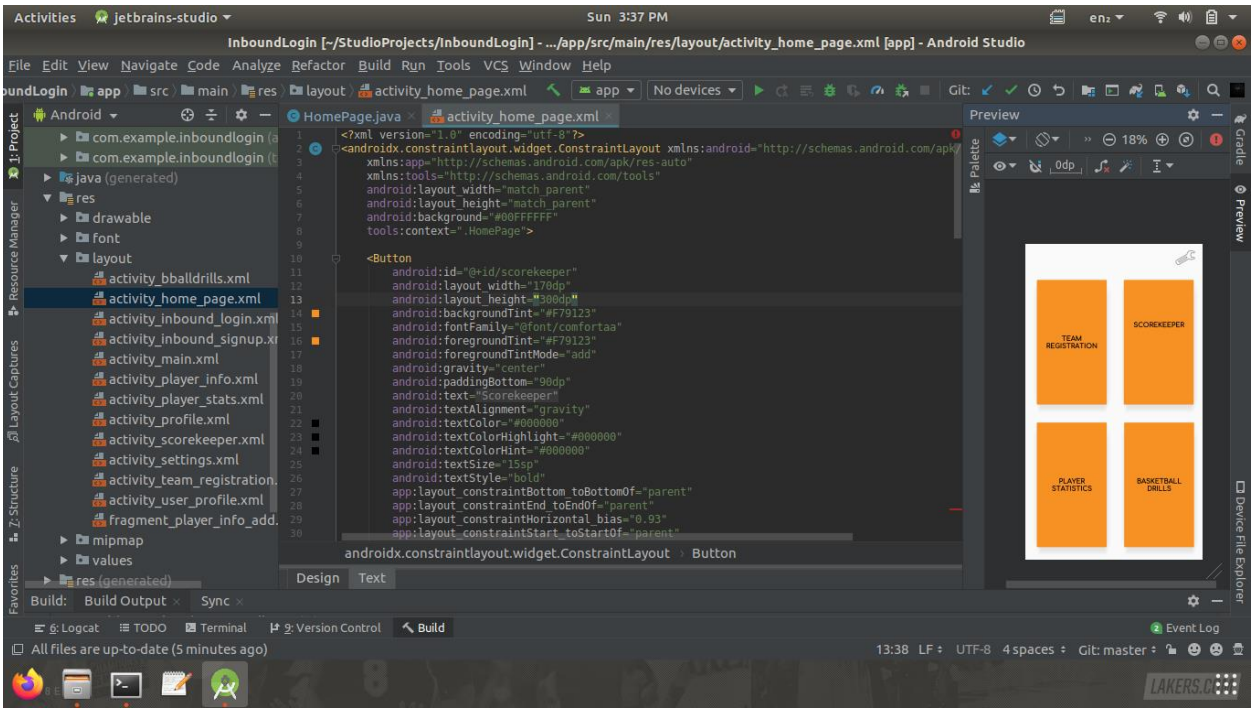

Fig 4.8 Home Page XML File

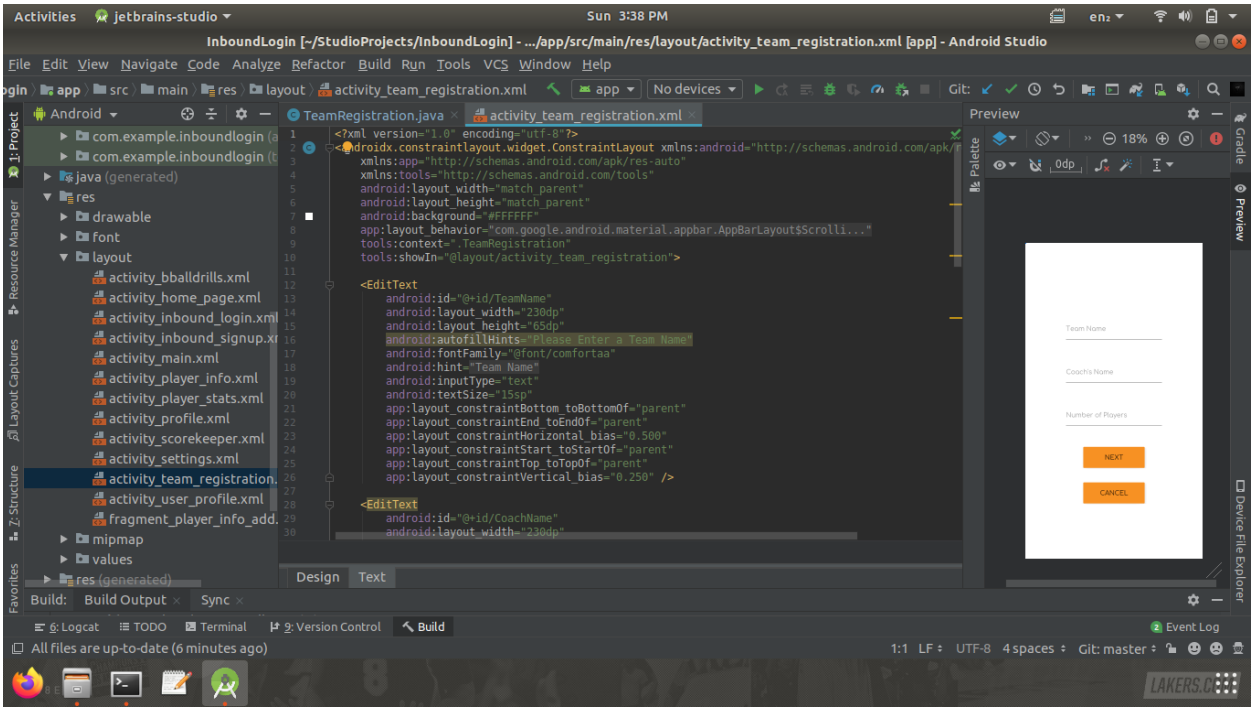

Fig 4.9 Team Registration XML Page

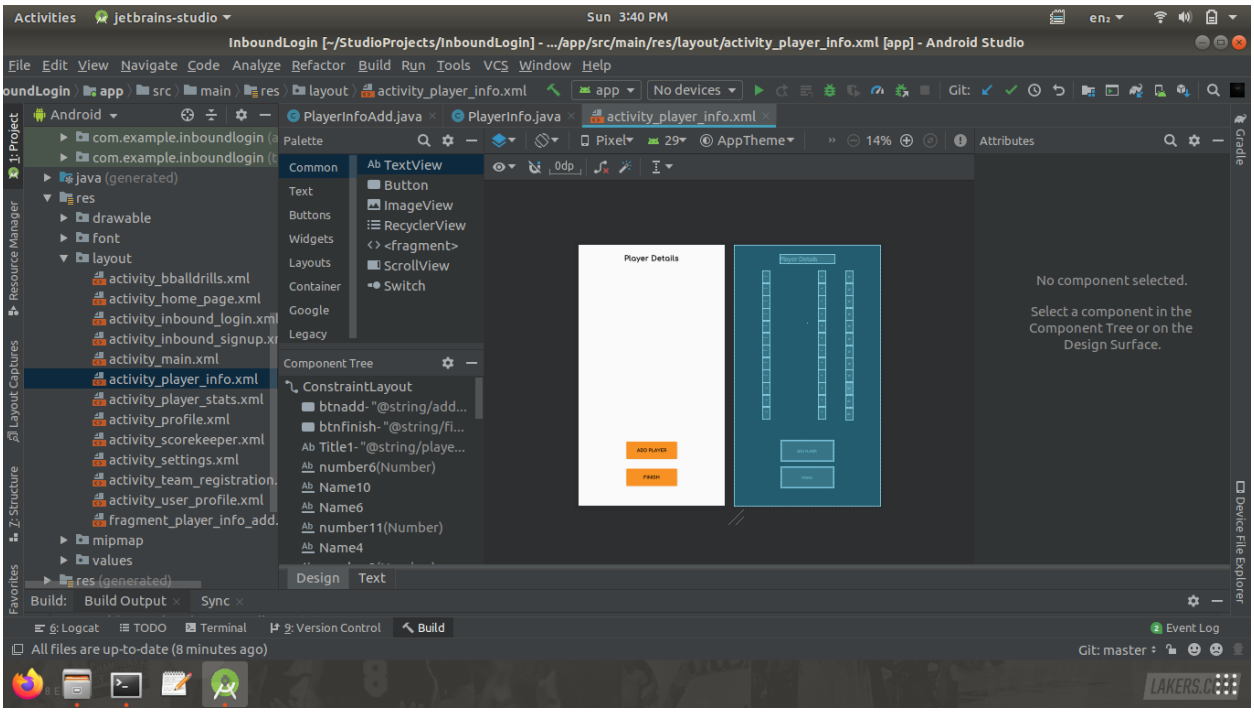

Fig 4.10 Player Registration XML File

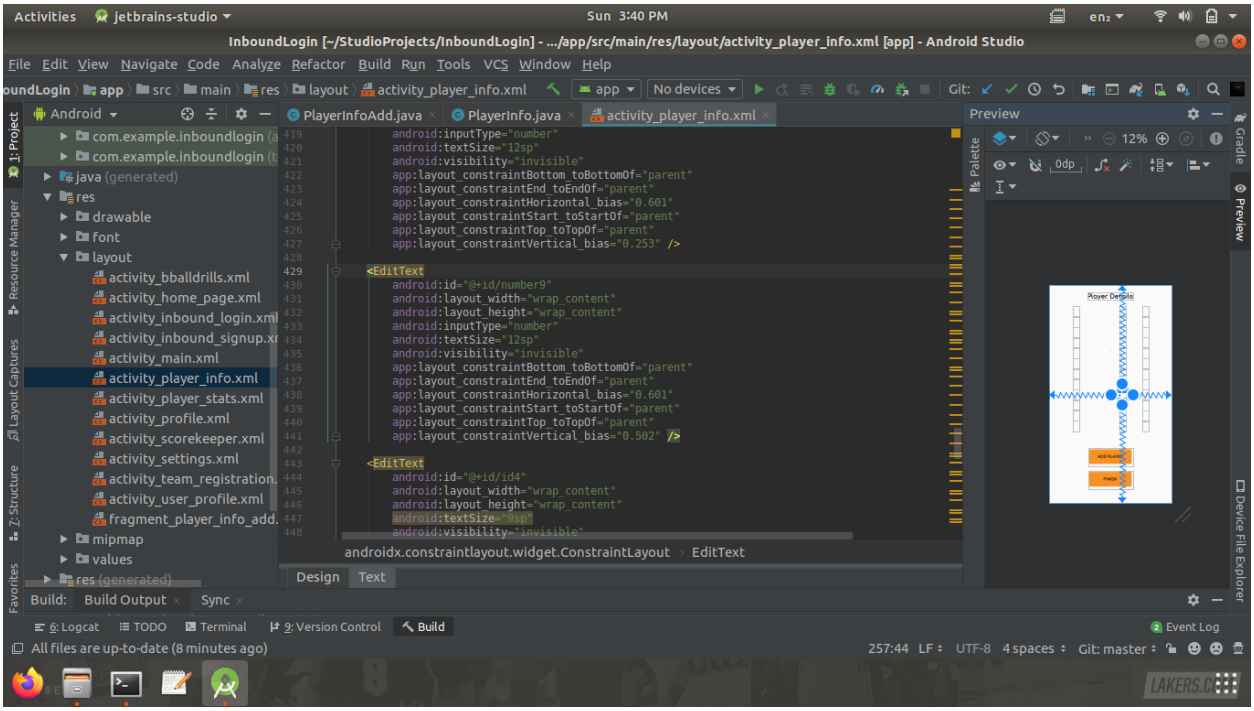

Fig 4.11 Player Registration XML File

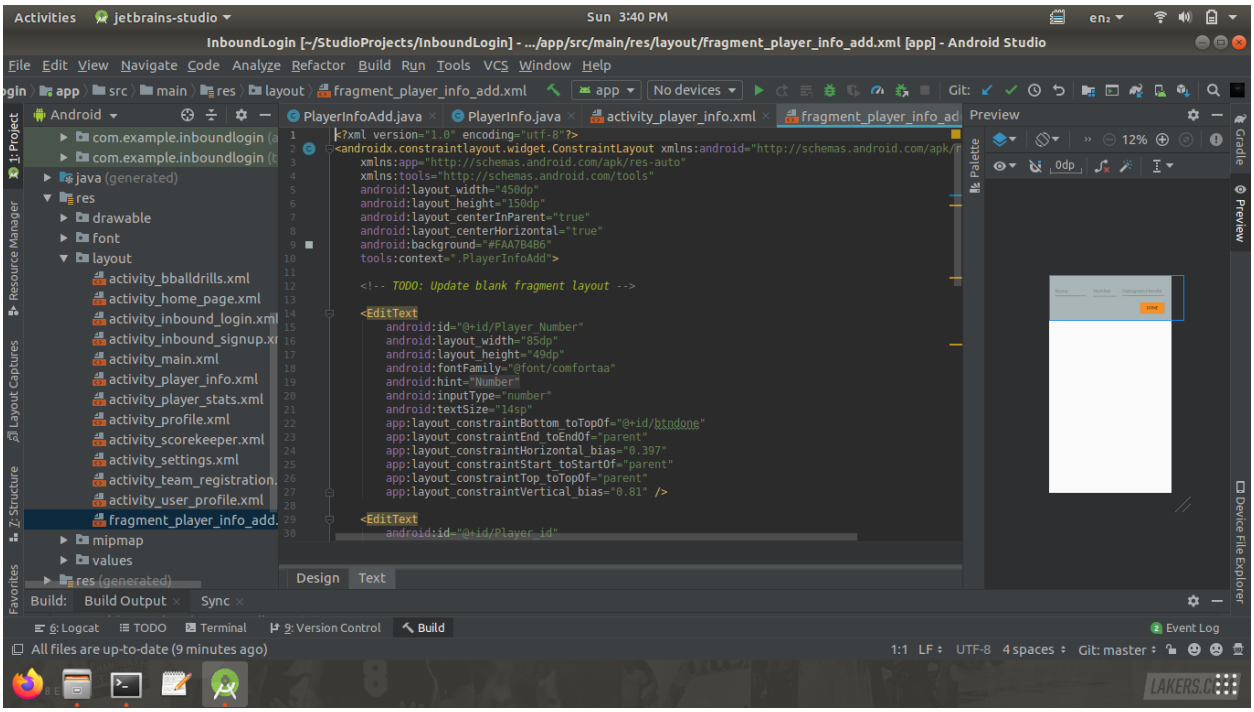

Fig 4.12 Player Registration Fragment

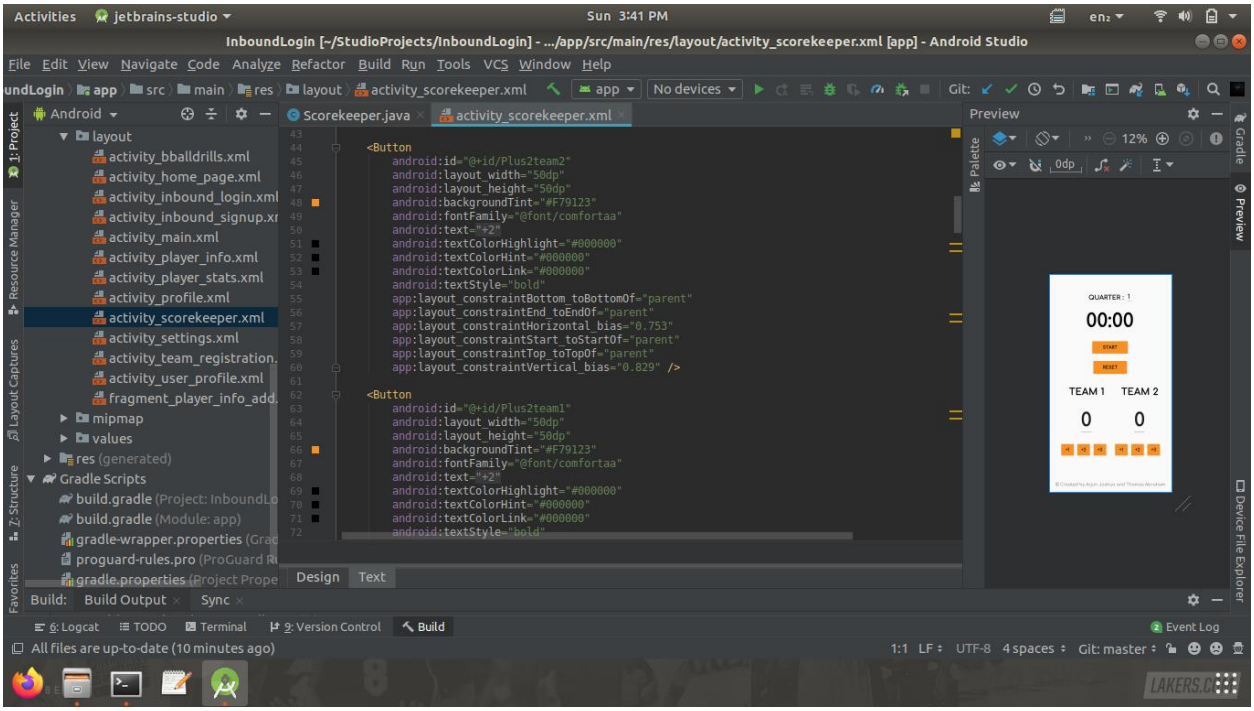

Fig 4.13 Scorekeeper XML File

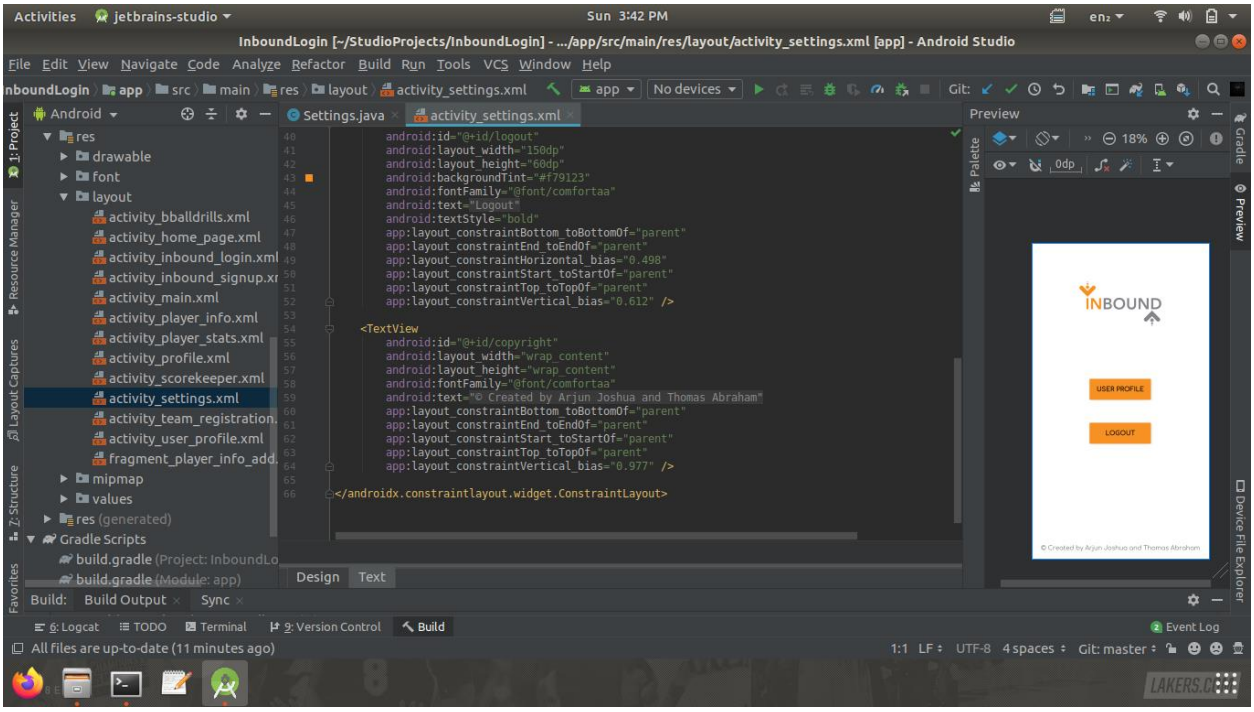

Fig 4.14 Settings XML File

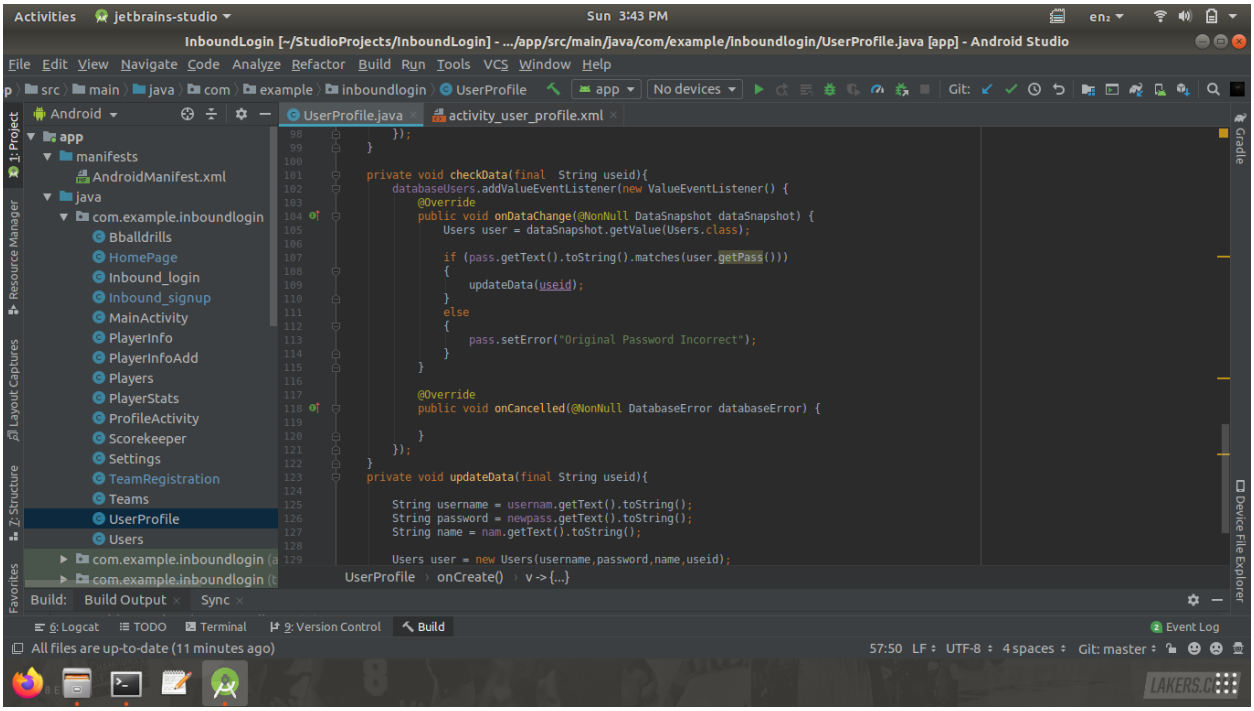

Fig 4.15 User Profile Java Page

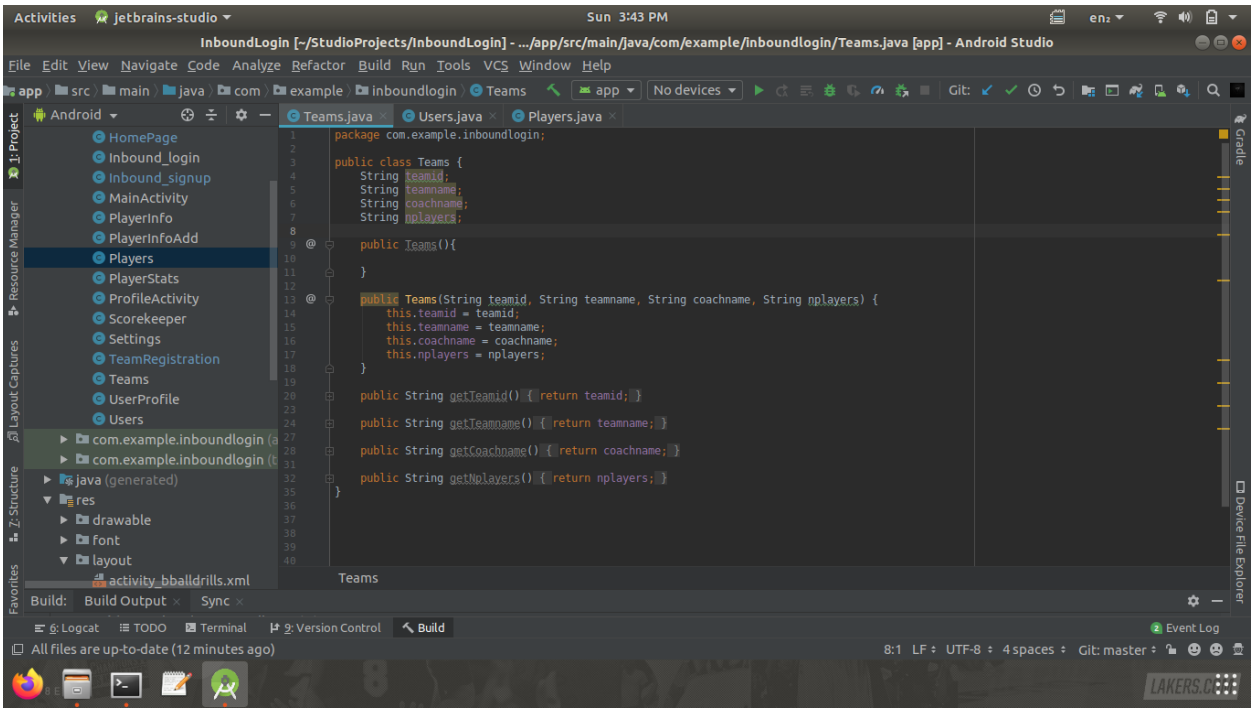

Fig 4.16 Team Database Page

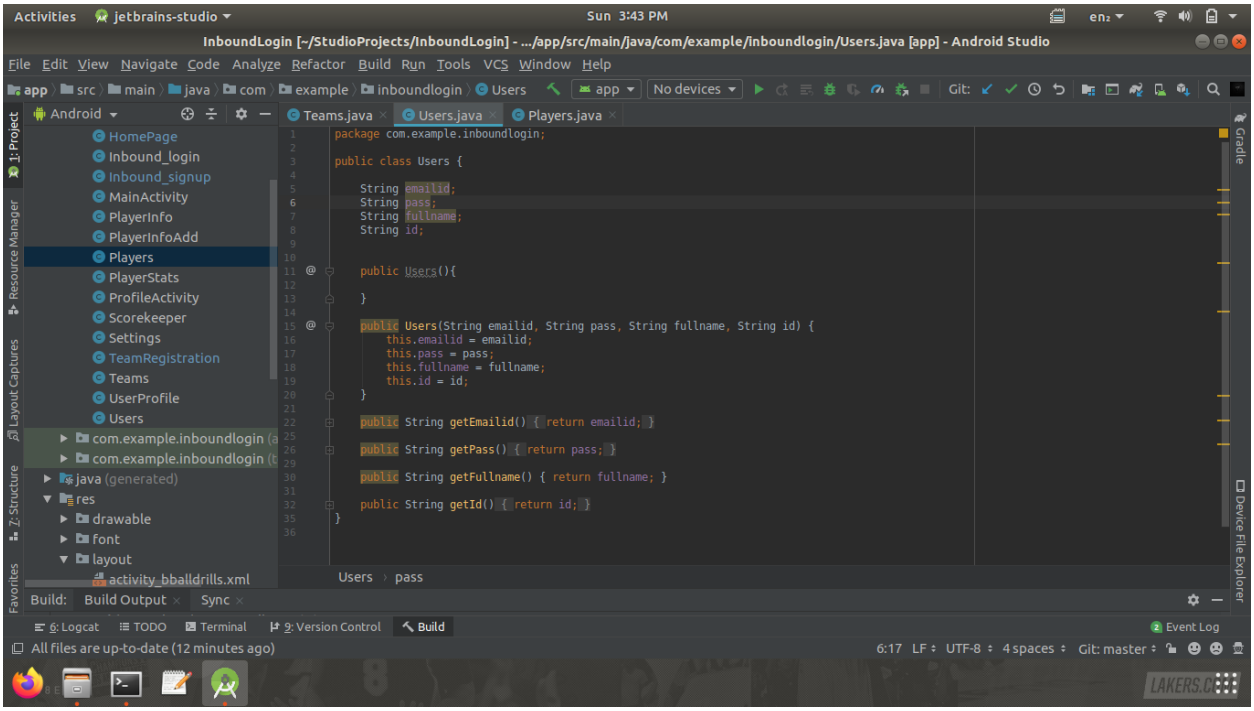

Fig 4.17 User Database Page

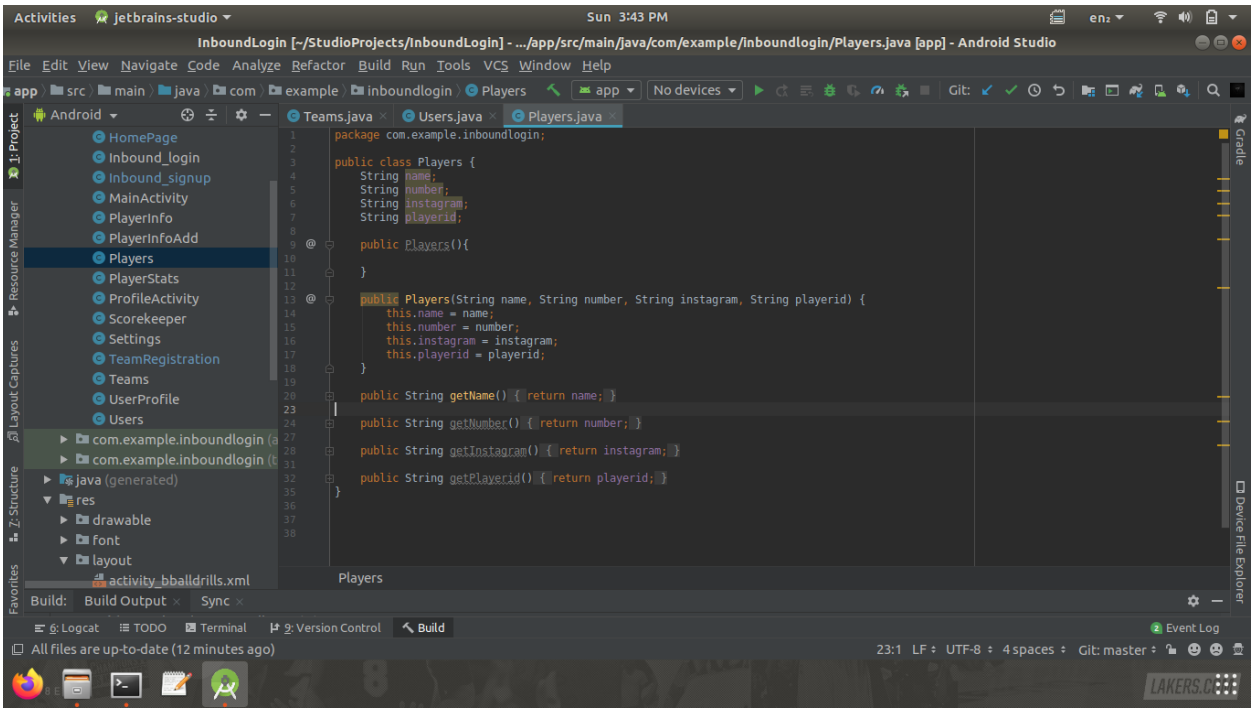

Fig 4.17 Player Database Page

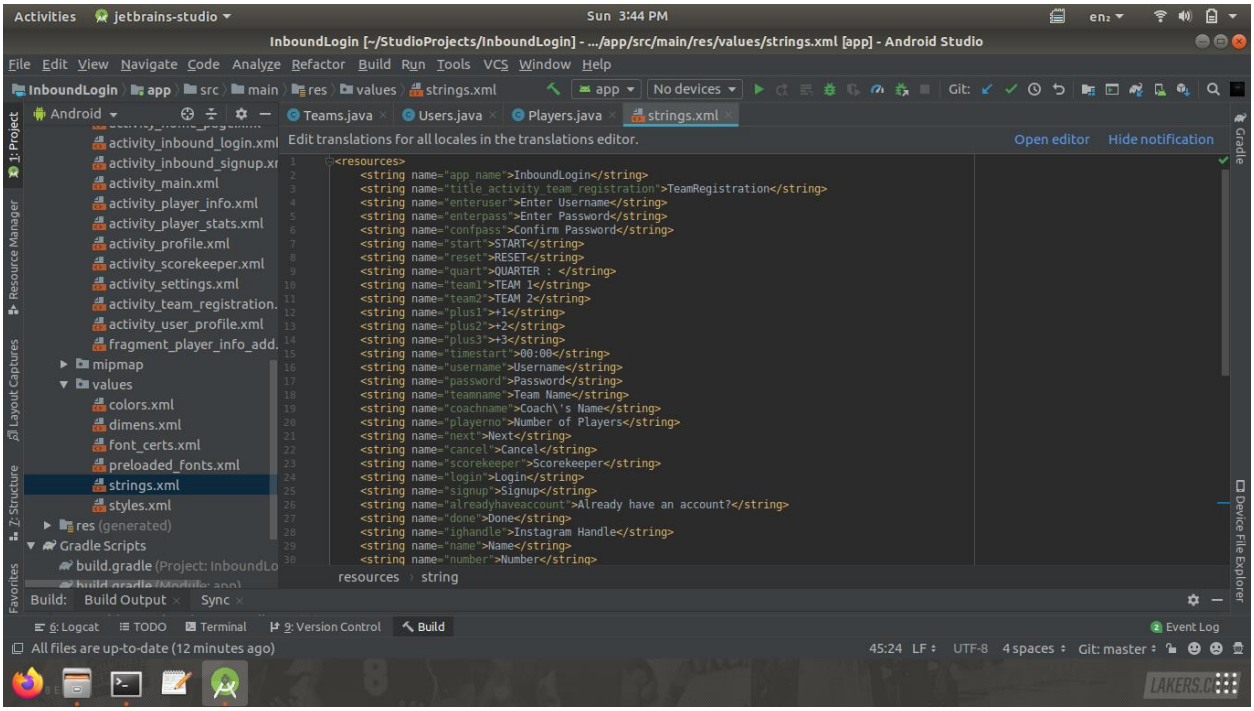

Fig 4.18 String XML File

5. TESTING

5.1 TESTING APPROACHES

5.1.1 Unit Testing

Unit testing was basically performed by the developers during and after the feature or UI element implementation and appropriate action was taken. This process or phase involved identifying all the units that were being developed and making sure they are displayed and behaves as expected under different user interaction inputs. Since this application contains four main modules and multiple sub modules, so many elements, forms and other objects being used to build the interface and function which in turn resulted in a big unit testing curve.
Example: Login button – Login button was tested with all the possible positive and negative testing scenarios to check if it works well in all possible scenarios.

5.1.2 Integration Testing

This is the testing phase where all the developed individual components of the application are meshed together to see if it functionally absolute. This system is made up of modules like Team Registration and Player Registration that need to receive data from each other in order to communicate with the database and maintain maximum functionality and this basically involves regression areas that usually occurs while integrating two different things which may arise due to change of code or functional behavior and this was tested as regression testing. This testing was done by the tester.

Example: Accessing team registration details is done by the Player Registration module from the Team Registration module, so these two interface interactions were tested to check that there were no errors.

5.2 TEST CASES

5.2.1 Test Scenario 1: Inbound Login Function

Objective: The main objective of this test case is to verify the application login functionality that involves both positive and negative scenarios.

Step	Test Steps	Expected Result
1	Launch Inbound System	Application launched successfully
2	Enter a username	Username is entered successfully
3	Enter a password	Password entered successfully
4	Press the Login button	Login successful
5	Repeat steps 2 and 3 with incorrect credentials	Login unsuccessful
6	Close the application	Application closed successfully

Table 5.1 - Test Scenario 1

5.2.2 Test Scenario 2: Team and Player Registration Workflow

Objective: Perform all the team and player registration functions and verify if all the player search criteria works as expected.

Step	Test Steps	Expected Result
1	Login to the Inbound application	User successfully logged in
2	Navigate to the team registration page	Team registration page opened successfully
3	Enter Team Name, Coach's Name and number of players	Data successfully sent to firebase database
4	Navigate to the add players button	Fragment opens
5	Enter Player Name, Jersey Number and Instagram ID	Data successfully sent to firebase database
6	Player details displayed on the page	Player data successfully pulled from the database
7	Select any particular detail and edit it	Details can be edited and the changes reflect in the database
8	Press Finish button	Successfully navigated back to home page
9	Logout of the application	Application successfully closed

Table 5.2 - Test Scenario 2

5.2.3 Test Scenario 2: Scorekeeper Page

Objective: Perform and verify the functionality of scorekeeper validation.

Step	Test Steps	Expected Result
1	Login to the Inbound application	User successfully logged in
2	Navigate to the scorekeeper page	Scorekeeper page opened successfully
3	Select Team Name from the drop-down menus	Team names pulled from database
4	Navigate to the start timer button	Timer begins at 10:00 and goes backward
5	Press required point buttons	Total team point details updated in database
6	Press pause or reset button	Timer should pause or reset
8	Press Back button	Successfully navigated back to home page
9	Logout of the application	Application successfully closed

Table 5.3 - Test Scenario 3

5.3 TEST REPORTS

This application is being tested with different testing approaches such as standard workflow test cases, automation scripts and exploratory testing which produces its own versions of test reports. The first two approaches produce results in the form of expected and actual outcomes for each step with the test step result either pass or fail, whereas the exploratory was being done without following any structured tests hence results are not produced.

Test report structure fields includes the following information:

• Test evidence: Workflow screenshots are taken for each verification point as evidence and reported for later reference.

• Run details: Test run details like Environment, Browser, Machine, OS Build, Tester and Software Build Version are captured as part of test result reports.

• Actual and Expected: Actual result and expected behavior of the system are captured stepwise with pass and fail status.

The below report sample shows the approaches followed and the result extracted after testing the application.

5.3.1 Test Report 1: Inbound Login Function

Description: The main objective of this test case is to validate the application expected login functionality outputs involving both positive and negative scenarios.

Step	Test Steps	Input 1	Result/Output	Input 2	Result/Output
1	Launch Inbound Application				
2	Enter Username	Correct	Passed	Incorrect	Failed
3	Enter Password	Correct	Passed	Incorrect	Failed
4	Press the Login button		Passed		Failed
5	Close the application				

Table 5.4 - Test Report 1

5.3.2 Test Report 2: Scorekeeper Page

Description: The main objective of this test case is to validate the application expected scorekeeper functionality outputs involving all possible scenarios.

Step	Test Steps	Input 1	Result/Output	Input 2	Result/Output
1	Launch Inbound Application				
2	Start Timer	Correct	Passed	Incorrect	Failed
3	Pause Timer	Correct	Passed	Incorrect	Failed
4	Reset Timer		Passed		Failed
5	Close Page				

Table 5.5 - Test Report 2

6. CONCLUSION

The primary purpose of the Inbound application is to connect basketball enthusiasts all over Bangalore so that they can view local basketball matches that are played between professional clubs or even amateur teams as well as colleges and universities.
Online scoreboards and individual player tracking systems have already been deployed and implemented in multiple countries where basketball is an extremely popular sport. The implementation of such systems has had a great benefit on the sport as it allows fans and enthusiasts to keep up with their favorite teams and players in real time.

6.1 DESIGN AND IMPLEMENTATION ISSUES

6.1.1 DESIGN ISSUES

The design and overall structure of the application was easily understandable but took a fair amount of time to design as the user interface had to be made for both the administrator or scorekeeper as well as the average user. While Android Studio using Java was the clear choice when it came to the development of the application, it was a rather long process to choose a viable database option. Eventually Firebase was chosen as it seemed to be the most efficient fit for our application.

6.1.2 IMPLEMENTATION ISSUES

During the initial phase of implementation, the team faced a few challenges such as the inability to synchronize the gradle on Android Studio due to the lack of particular administrator permissions on the network being used. The team also had to choose the appropriate build model to ensure that maximum number of people would be able to use the app in the required resolution. While connection to Firebase was fairly straightforward and pushing data to the database was a simple process, the team faced an issue when the data which was already present in the database had to be updated through the mobile application interface. A similar issue such as the previous one was faced when we had to pull the data from the database to a new page.

6.2 ADVANTAGES AND LIMITATIONS

The primary intention and objective of the inbound application is to create a platform for local basketball enthusiasts to come together and watch local basketball matches as well as follow local players as their game develops. One of the main advancements that will occur through the implementation of inbound is the overall reduction in the amount of paper used to record the various statistics of a basketball match which should benefit the environment.

A few limitations with the current system is that in order for the realtime database to continuously receive and update data, it requires a consistent internet connection which may not always be available.

6.3 FUTURE SCOPE OF THE PROJECT

The inbound application has a future in today's local basketball structure where there are no digital applications being used. In local basketball games in Bangalore where paper is still used to maintain the score as well as individual player statistics of an ongoing match an application such as inbound will be a welcome change as it will reduce the amount of work and paper required in a game consisting of four quarters. Systems such as this have already been implemented in countries where basketball is an extremely popular sport and they have shown great success.

REFERENCES

1. David Etheridge, “An Introduction to Java Programming”, BookBoon Publisher, 1st Edition, 2009.
2. “Developers Guide to Android Studio.”Android Developers. 27 December 2019. 2 January 2020 <https://developer.android.com/guide>.
3. “Google Login Android using Firebase Tutorial.”Simplified Coding. 11 October 2019. 7 January 2020 < https://www.simplifiedcoding.net/google-login-android/.>
4. “Android UI Debugger.”Android Developers. 27 December 2019. 11 January 2020 <https://developer.android.com/studio/debug/layout-inspector.>
5. “Firebase Connection Guide.”Google Firebase. 25 February 2020. 16 January 2020 <https://firebase.google.com/docs/android/setup.>
6. “Firebase Realtime Database Guide.”Google Firebase. 28 January 2020. 21 January 2020 <https://firebase.google.com/docs/database. >
7. “Beginners Guide to Firebase.”Fireship. 18 December 2019. 15 January 2020 <https://fireship.io/lessons/the-ultimate-beginners-guide-to-firebase/.>