Delivery 1: KPIs

Marta Llurba, Albert Chica

Code Structure

Cs Files

DataTransmission.cs

What DataTransmission.cs does is handle, as the name indicates, the data transmission between the game, in this case the simulator in unity, to the server, in our case the PhpmyAdmin.

This will allow us to keep track of the player activities in the game, and store all the data on the server.

```
using System;
using System.Collections;
using UnityEngine;
using UnityEngine.Networking;

public class DataTransmission : MonoBehaviour
{
    uint currentUserId;
    uint currentSessionId;
    uint currentPurchaseId;

    private enum ActionType { NewPlayer, StartSession, EndSession, BuyItem }

    private void OnEnable()
    {
        Simulator.OnNewPlayer += (name, country, date) =>
        {
            WWWForm form = CreateForm(ActionType.NewPlayer, name, country, date.ToString("yyyy-MM-dd HH:mm:ss"));
            StartCoroutine(UploadData(ActionType.NewPlayer, form, "https://citmalumnes.upc.es/~albertcf5/Player_Data.php"));
        };

        Simulator.OnNewSession += (date) =>
        {
            WWWForm form = CreateForm(ActionType.StartSession, date: date.ToString("yyyy-MM-dd HH:mm:ss"));
            StartCoroutine(UploadData(ActionType.StartSession, form, "https://citmalumnes.upc.es/~albertcf5/Session_Data.php"));
        };

        Simulator.OnEndSession += (date) =>
        {
            WWWForm form = CreateForm(ActionType.EndSession, date: date.ToString("yyyy-MM-dd HH:mm:ss"));
            StartCoroutine(UploadData(ActionType.EndSession, form, "https://citmalumnes.upc.es/~albertcf5/Close_Session_Data.php"));
        };

        Simulator.OnBuyItem += (item, date) =>
        {
            WWWForm form = CreateForm(ActionType.BuyItem, date: date.ToString("yyyy-MM-dd HH:mm:ss"), item: item);
            StartCoroutine(UploadData(ActionType.BuyItem, form, "https://citmalumnes.upc.es/~albertcf5/Purchase_Data.php"));
        };
    }
```

OnEnable listens for events like creating new players, starting and ending sessions and items purchased by the Simulator.

```
private WWWForm CreateForm(ActionType actionType, string name = null, string country = null, string date = null, int item = 0)
{
    WWWForm form = new WWWForm();
    switch (actionType)
    {
        case ActionType.NewPlayer:
            form.AddField("Name", name);
            form.AddField("Country", country);
            form.AddField("Date", date);
            break;

        case ActionType.StartSession:
            form.AddField("User_ID", currentUserId.ToString());
            form.AddField("Start_Session", date);
            break;

        case ActionType.EndSession:
            form.AddField("User_ID", currentUserId.ToString());
            form.AddField("End_Session", date);
            form.AddField("Session_ID", currentSessionId.ToString());
            break;

        case ActionType.BuyItem:
            form.AddField("Item", item.ToString());
            form.AddField("User_ID", currentUserId.ToString());
            form.AddField("Session_ID", currentSessionId.ToString());
            form.AddField("Buy_Date", date);
            break;
    }
    return form;
}
```

The CreateForm makes a form with specific fields of each action.

```
IEnumerator UploadData(ActionType actionType, WWWForm form, string url)
{
    using (UnityWebRequest www = UnityWebRequest.Post(url, form))
    {
        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
        {
            UnityEngine.Debug.LogError($"{actionType} data upload failed: " + www.error);
        }
        else
        {
            string answer = www.downloadHandler.text.Trim(new char[] { '\uFEFF', '\u200B', ' ', '\t', '\r', '\n' });

            if (actionType == ActionType.NewPlayer && uint.TryParse(answer, out uint parsedUserId) && parsedUserId > 0)
            {
                currentUserId = parsedUserId;
                CallbackEvents.OnAddPlayerCallback.Invoke(currentUserId);
            }
            else if (actionType == ActionType.StartSession && uint.TryParse(answer, out uint parsedSessionId) && parsedSessionId > 0)
            {
                currentSessionId = parsedSessionId;
                CallbackEvents.OnNewSessionCallback.Invoke(currentSessionId);
            }
            else if (actionType == ActionType.EndSession)
            {
                CallbackEvents.OnEndSessionCallback.Invoke(currentSessionId);
            }
            else if (actionType == ActionType.BuyItem && uint.TryParse(answer, out uint parsedPurchaseId) && parsedPurchaseId > 0)
            {
                currentPurchaseId = parsedPurchaseId;
                CallbackEvents.OnItemBuyCallback.Invoke();
            }
            else
            {
                UnityEngine.Debug.LogError($"Invalid response for {actionType}: " + answer);
            }
        }
    }
}
```

The UpdateData sends the form data into the server.

Php files

db_connect.php

Using MySQLi we establish a connection to the database.

The code defines the database connection parameters like the server, username, password and database name. Using these parameters it creates a new connection to MySQL, and if it fails, displays an error message and terminates the script.

```php
<?php
$servername = "localhost:3306";
$username = "albertcf5";
$password = "48103884m";
$database = "albertcf5";

// Create connection
$conn = new mysqli($servername, $username, $password, $database);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
```

Player_Data.php

The player data does:

- Retrieves the player data from the POST request
    - Name
    - Country
    - Date
- Adds the player's information into the database

```php
<?php
include 'db_connect.php';

$name = $_POST["Name"];
$country = $_POST["Country"];
$date = $_POST["Date"];

error_log("Received player data: Name={$name}, Country={$country}, Date={$date}");

$stmt = $conn->prepare("INSERT INTO `Players`(`Name`, `Country`, `Date`) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $name, $country, $date);

if ($stmt->execute()) {
    echo $conn->insert_id;
} else {
    error_log("Error in Player_Data.php: " . $stmt->error);
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conn->close();
?>
```

Purchase_Data.php

The Purchase_Data is responsible for handling the purchasing transaction and storing it in the database.

- Collects the information sent by the user
    - User Id
    - Session ID
    - Item
    - Purchase Date
- Adds the information into the database table
- Saves the purchase details in the database

```php
<?php
include 'db_connect.php';

$userId = $_POST["User_ID"];
$sessionId = $_POST["Session_ID"];
$itemId = $_POST["Item"];
$buyDate = $_POST["Buy_Date"];

error_log("Received purchase data: User_ID={$userId}, Session_ID={$sessionId}, Item={$itemId}, Buy_Date={$buyDate}");

$stmt = $conn->prepare("INSERT INTO `Purchases`(`userId`, `sessionId`, `itemId`, `buyDate`) VALUES (?, ?, ?, ?)");
$stmt->bind_param("iiis", $userId, $sessionId, $itemId, $buyDate);

if ($stmt->execute()) {
    echo $conn->insert_id;
} else {
    error_log("Error in Purchase_Data.php: " . $stmt->error);
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conn->close();
?>
```

Session_Data.php

The Session_Data handles the insertion of session data into the database.

- Gets the user ID and session start time from POST data.
- Saves the user ID and session start time in the “Sessions” table.

```php
<?php
include 'db_connect.php';

$userId = $_POST["User_ID"];
$startSession = $_POST["Start_Session"];

error_log("Received session start data: User_ID={$userId}, Start_Session={$startSession}");

$stmt = $conn->prepare("INSERT INTO `Sessions`(`userId`, `startSession`) VALUES (?, ?)");
$stmt->bind_param("is", $userId, $startSession);

if ($stmt->execute()) {
    echo $conn->insert_id;
} else {
    error_log("Error in Session_Data.php: " . $stmt->error);
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conn->close();
?>
```

Close_Session_Data.php

The Close_Session_Data handles the closing session in the database

- Gets 2 parameters from POST

    - Session ID
    - End Session
- Updates the “Sessions” table. It will change the “endSession” value for a specific session ID
- On successful update
    - Echoes “End_Session” value
- If no rows are affected
    - Echoes “No session updated”

```php
<?php
include 'db_connect.php';

$sessionId = $_POST["Session_ID"];
$endSession = $_POST["End_Session"];

error_log("Received end session data: Session_ID={$sessionId}, End_Session={$endSession}");

$stmt = $conn->prepare("UPDATE `Sessions` SET `endSession` = ? WHERE `sessionId` = ?");
$stmt->bind_param("si", $endSession, $sessionId);

if ($stmt->execute()) {
    if ($stmt->affected_rows > 0) {
        //echo "Session closed successfully";
        echo $endSession;
    } else {
        error_log("No session updated in Close_Session_Data.php");
        echo "No session updated";
    }
} else {
    error_log("Error in Close_Session_Data.php: " . $stmt->error);
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conn->close();
?>
```

KPIs

- DAU/MAU/Stickiness:

We used this as the main code to calculate these 3 KPIs. Later we changed dates and looked at the data. Even though there is some things marked as errors the code worked normally.

```sql
SELECT DATE(s.startSession) AS sessionDate,
COUNT(DISTINCT s.userId) AS dailyActiveUsers,
SUM(COUNT(DISTINCT s.userId)) OVER() AS MonthlyActiveUsers,
COUNT(DISTINCT s.userId) * 100 / SUM(COUNT(DISTINCT s.userId)) OVER() AS stickiness
FROM Sessions s WHERE s.startSession BETWEEN '2022-09-01' AND '2022-09-31'
GROUP BY DATE(s.startSession)
ORDER BY dailyActiveUsers DESC
```

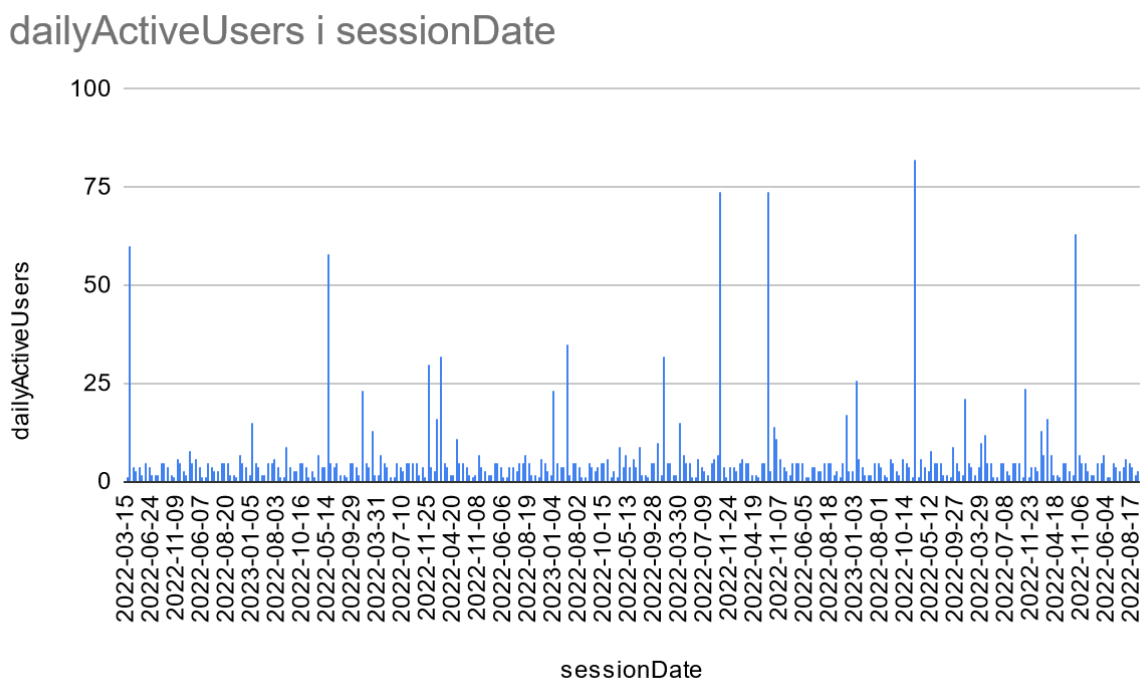
This is an example of the 9th month of 2022, MonthlyActiveUsers repeats everyday, but you can see the stickiness and the daily connections of every day.

sessionDate	dailyActiveUsers 1	MonthlyActiveUsers	stickiness
2022-09-24	17	144	11.8056
2022-09-25	5	144	3.4722
2022-09-29	5	144	3.4722
2022-09-26	5	144	3.4722
2022-09-06	5	144	3.4722
2022-09-03	5	144	3.4722
2022-09-30	5	144	3.4722
2022-09-10	5	144	3.4722
2022-09-27	5	144	3.4722
2022-09-07	5	144	3.4722
2022-09-01	5	144	3.4722
2022-09-28	5	144	3.4722
2022-09-18	4	144	2.7778
2022-09-08	4	144	2.7778
2022-09-15	4	144	2.7778
2022-09-05	4	144	2.7778
2022-09-22	4	144	2.7778
2022-09-12	4	144	2.7778
2022-09-02	4	144	2.7778
2022-09-19	4	144	2.7778

After looking at some data we extracted the monthly users KPI to another table to look at the users connected in every month. Things we can mark as well is the declining in the 12 months, between peaks of connections, we can deduce this might be christmas affecting normal connections.

```sql
SELECT YEAR(startSession) as year,
MONTH(startSession) as month,
COUNT(DISTINCT userId) as monthlyUsers
FROM Sessions
GROUP BY YEAR(startSession), MONTH(startSession)
ORDER BY YEAR(startSession), MONTH(startSession);
```

year	month	monthlyUsers
2022	2	77
2022	3	35
2022	4	149
2022	5	106
2022	6	87
2022	7	89
2022	8	31
2022	9	22
2022	10	23
2022	11	115
2022	12	22
2023	1	194

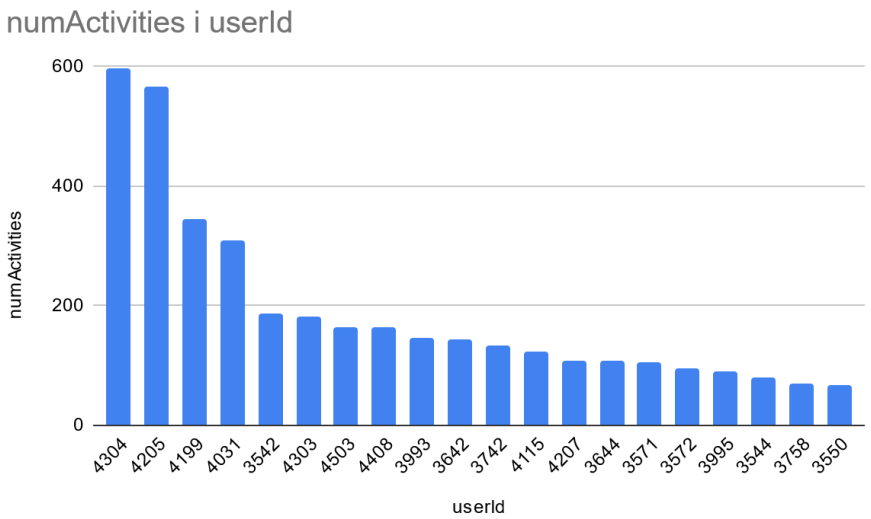


- User Activity
    - Number of Sessions per User

We used this data to check on the users that have more connections and how many those had. Later you can see a chart depicting those statistics and the difference between users.

```
SELECT userId, COUNT(sessionId) AS numActivities
FROM Sessions
GROUP BY userId
ORDER BY numActivities DESC
```

userId	numActivities 1
4304	597
4205	566
4199	345
4031	310
3542	186
4303	183
4503	165
4408	163
3993	147
3642	143
3742	133
4115	124
4207	108
3644	108
3571	105
3572	96
3995	90
3544	81
3758	71
3550	68



    - Average Session Duration

We have calculated the average time of a session of every user in minutes to see the relation with the average time of a session in general. WE joined tables because we were having some problems while making the sql code, but we are aware is not needed.

```sql
SELECT p.userId,
COUNT(DISTINCT s.sessionId) AS numberOfSessions,
AVG(TIMESTAMPDIFF(MINUTE, s.startSession, s.endSession)) as timeSession
FROM Players p
LEFT JOIN Sessions s ON p.userId = s.userId
WHERE s.startSession IS NOT null AND
s.endSession IS NOT null
GROUP BY p.userId
ORDER BY `timeSession`  DESC
```

userId	numberOfSessions	timeSession 1
4377	1	51129.0000
4302	1	11546.0000
4198	1	10624.0000
4385	1	9328.0000
4115	104	9318.3462
4303	167	8880.1138
4390	1	8654.0000
4387	1	8597.0000
4403	1	8429.0000
4114	5	8384.0000
4199	318	8374.8742
4116	39	8340.0256
4193	1	7278.0000
4202	1	7276.0000
4500	1	7202.0000
4404	1	7070.0000
4389	1	7063.0000
4392	1	7049.0000
4406	1	7043.0000
4194	1	6828.0000
4488	1	6524.0000
4502	1	6473.0000
4499	1	6456.0000

```sql
SELECT AVG(TIMESTAMPDIFF(MINUTE, startSession, endSession)) AS averageTime
FROM Sessions
WHERE startSession IS NOT NULL
AND endSession IS NOT NULL
```

averageTime
2969.3862

- Most Active Users
  - Based on Session Count

The number of sessions every player had.

```sql
SELECT userId,
COUNT(DISTINCT sessionId) AS numSessions
FROM Sessions
GROUP BY userId
ORDER BY numSessions DESC
```

userId	numSessions 1
4304	597
4205	566
4199	345
4031	310
3542	186
4303	183
4503	165
4408	163
3993	147
3642	143
3742	133
4115	124
3644	108
4207	108
3571	105
3572	96
3995	90
3544	81
3758	71
3550	68

  - Based on Total Session Duration

The total of seconds every player uses our application, we added every session time total of every user.

```sql
SELECT userId,
SUM(TIMESTAMPDIFF(HOUR, startSession, endSession)) AS totalTimeHours
from Sessions
WHERE startSession IS NOT null
AND endSession IS NOT null
GROUP BY userId
ORDER BY totalTimeHours DESC
```

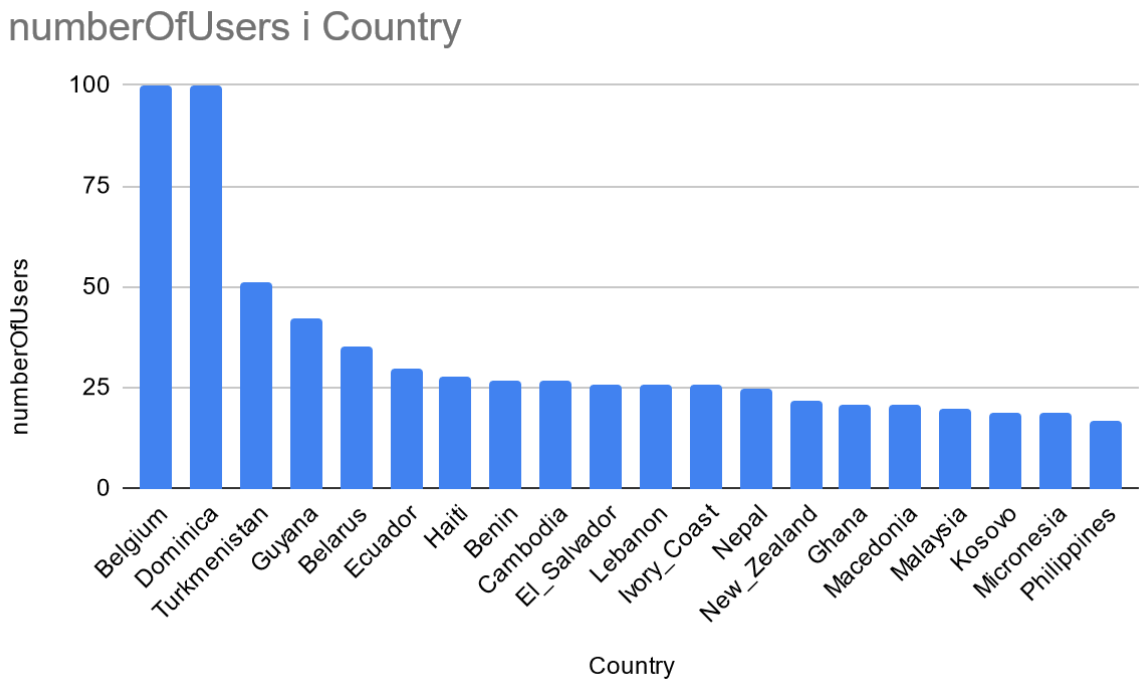
←T→				userId	totalTimeHours 1
☐	Edit	Copy	Delete	4304	52831
☐	Edit	Copy	Delete	4205	46353
☐	Edit	Copy	Delete	4199	44234
☐	Edit	Copy	Delete	4303	24634
☐	Edit	Copy	Delete	4115	16104
☐	Edit	Copy	Delete	4503	13239
☐	Edit	Copy	Delete	4408	9723
☐	Edit	Copy	Delete	4031	7218
☐	Edit	Copy	Delete	4207	6523
☐	Edit	Copy	Delete	4116	5402
☐	Edit	Copy	Delete	3992	3398
☐	Edit	Copy	Delete	3993	2351
☐	Edit	Copy	Delete	3542	2062
☐	Edit	Copy	Delete	3642	1340
☐	Edit	Copy	Delete	3571	1326
☐	Edit	Copy	Delete	3742	1262
☐	Edit	Copy	Delete	3882	992
☐	Edit	Copy	Delete	3572	945
☐	Edit	Copy	Delete	4430	916
☐	Edit	Copy	Delete	3916	891
☐	Edit	Copy	Delete	3995	887
☐	Edit	Copy	Delete	4377	852
☐	Edit	Copy	Delete	3758	777
☐	Edit	Copy	Delete	4112	713

- Player Demographics
    - Distribution of Players by Country

We found we have a lot of users in Belgium and Dominica. This alone does not help us to understand our users but we can add this information on top of other KPIs.

```sql
SELECT Country,
COUNT(userId) as numberOfUsers
FROM Players
GROUP BY Country
ORDER BY numberOfUsers DESC
```

Country	numberOfUsers 1
Belgium	100
Dominica	100
Turkmenistan	51
Guyana	42
Belarus	35
Ecuador	30
Haiti	28
Benin	27
Cambodia	27
El_Salvador	26
Lebanon	26
Ivory_Coast	26
Nepal	25
New_Zealand	22
Ghana	21
Macedonia	21
Malaysia	20
Kosovo	19
Micronesia	19
Philippines	17

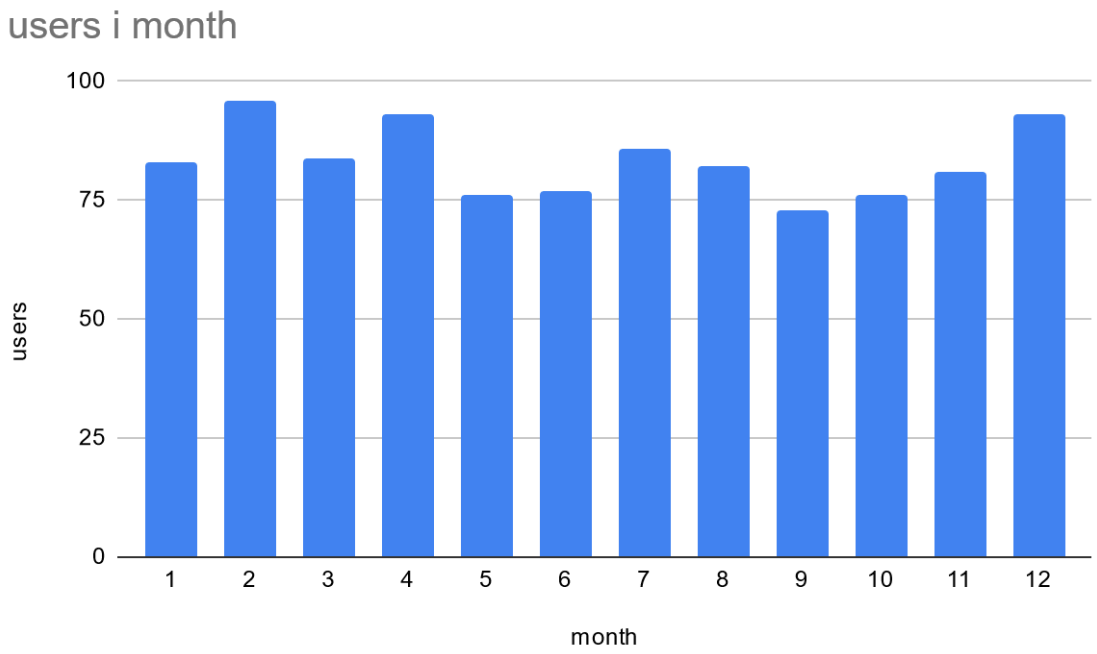


- New Player Acquisition Rate

First we tried by day but data was too low to understand anything, we ended up using month as time measure. Something that was interesting is again the 12th month, it has a lot of new users that do not connect at all as we have seen in the user connections per month.

```sql
SELECT YEAR(Date) AS year,
MONTH(Date) AS month,
COUNT(userId) AS users
FROM Players
GROUP BY month
ORDER BY users DESC
```

year	month	users 1
2022	2	96
2022	12	93
2022	4	93
2022	7	86
2022	3	84
2022	1	83
2022	8	82
2022	11	81
2022	6	77
2022	10	76
2022	5	76
2022	9	73



- Purchases
  - Average Purchase Value

The average value of every purchase is varelly costlier than our 3rd product out of 5.

```
SELECT COUNT(DISTINCT p.purchaseId) as purchases,
SUM(Price) as totalIncome,
SUM(Price)/COUNT(DISTINCT p.purchaseId) as averagePurchaseValue
FROM Purchases p
join Items i on p.itemId = i.Id
```

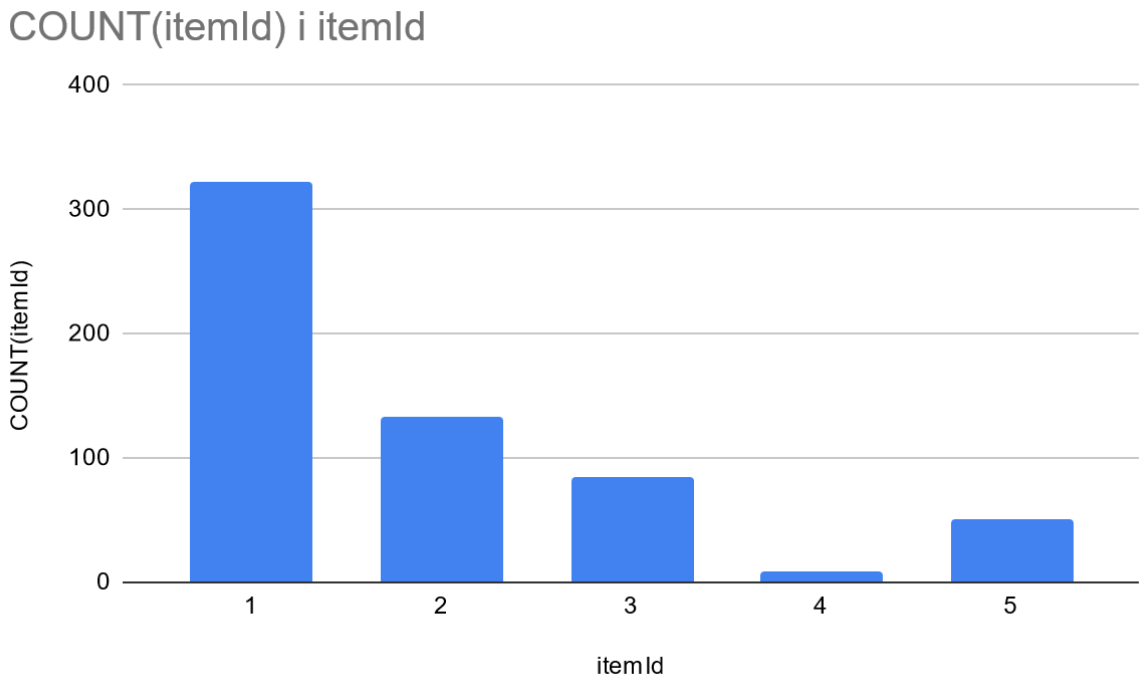
purchases	totalIncome	averagePurchaseValue
598	6922.019889593124	11.575284096309572

  - Popular Items

Additionally we calculated how many products of each one of them have we sold.

```sql
SELECT itemId,
COUNT(itemId)
FROM Purchases
GROUP BY itemId
```

itemId	COUNT(itemId)
1	322
2	133
3	84
4	8
5	51



- User Engagement
  - Conversion Rate from Sessions to Purchases

This KPI, somewhat connected to ARPU and ARPPU showed us a 10% of conversion rate between sessions and purchases.

```sql
SELECT COUNT(DISTINCT s.sessionId) as totalSessions,
COUNT(DISTINCT p.sessionId) as purchaseSessions,
COUNT(DISTINCT p.sessionId) * 100/COUNT(DISTINCT s.sessionId) as convertionRate
FROM Sessions s
LEFT JOIN Purchases p ON s.sessionId = p.sessionId
```

totalSessions	purchaseSessions	convertionRate
6056	598	9.8745

- Retention Rate
  - Retention D1

We calculate the retention rate of day 1, 3 and 7. It seems to be unimportant due to the lack of data we have, it would have been better to check which have connected after that day so we make sure they keep playing instead of daily playing users.

```
SELECT COUNT(DISTINCT p.userId) as d1Users
FROM Sessions s
LEFT JOIN Players p ON s.userId = p.userId
WHERE DAY(p.Date + INTERVAL 1 DAY) = DAY(s.startSession)
AND MONTH(p.Date) = MONTH(s.startSession)
AND YEAR(p.Date) = YEAR(s.startSession)
ORDER BY p.Date ASC
```

d1Users
3

  - Retention D3

d3Users
5

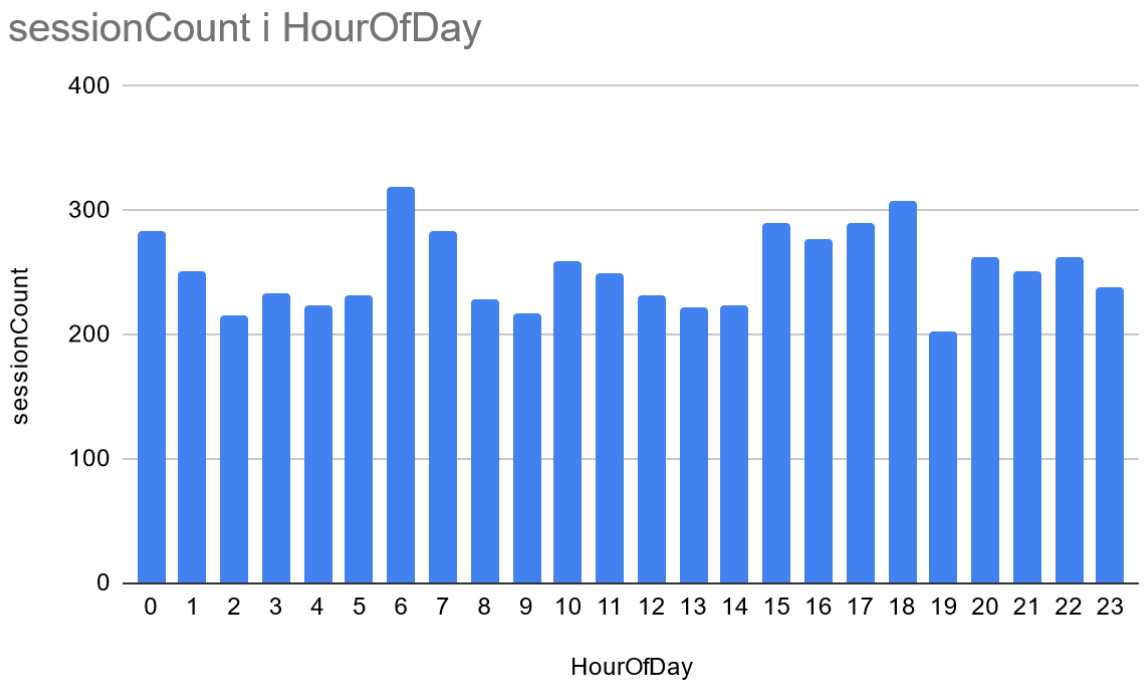
  - Retention D7

d7Users
7

- Session Analysis
  - Peak Usage Hours

We used the sessions table to see how many players connected each hour of the day, we found peaks in the hours 6 and 19.

```
SELECT HOUR(s.startSession) AS HourOfDay,
COUNT(s.userId) AS sessionCount
FROM Sessions s
GROUP BY HOUR(s.startSession)
ORDER BY HourOfDay ASC
```

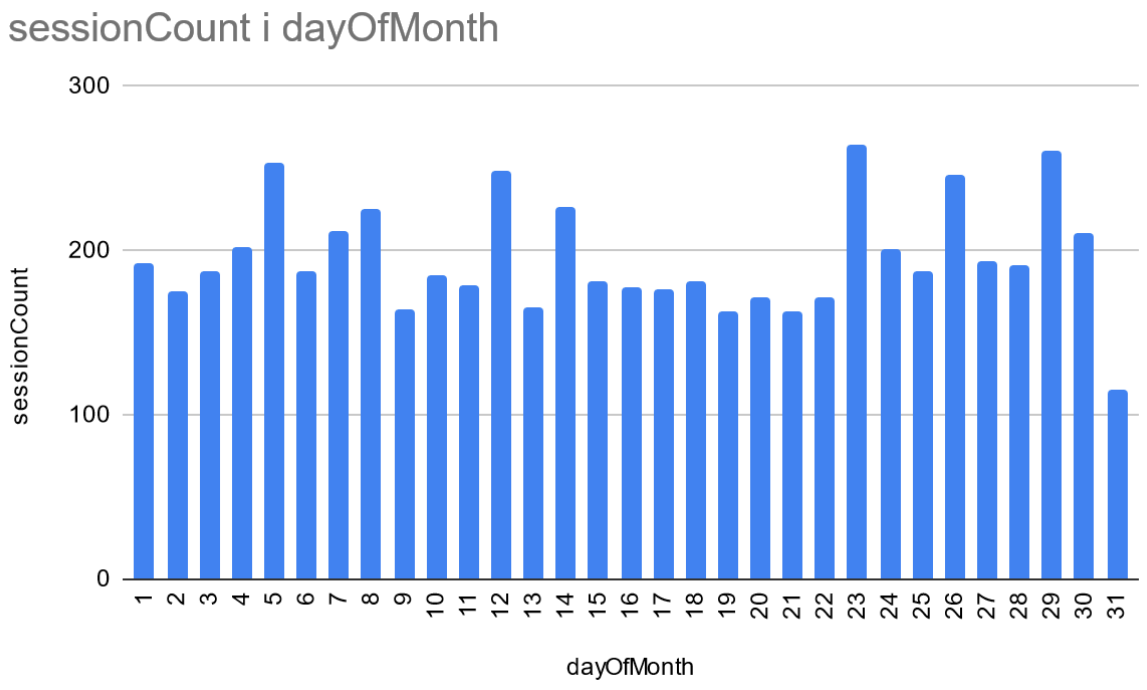


- Peak Usage Days

As we did with the hours of the day, we also calculated the amount of users that connected any day of the month. The low data in day 31 is due to the lack of days numbered 31 among a year.

```
SELECT DAY(s.startSession) AS dayOfMonth,
COUNT(s.userId) AS sessionCount
FROM Sessions s
GROUP BY DAY(s.startSession)
ORDER BY sessionCount DESC LIMIT 10
```

dayOfMonth	sessionCount 1
23	264
29	261
5	253
12	248
26	246
14	226
8	225
7	212
30	211
4	202



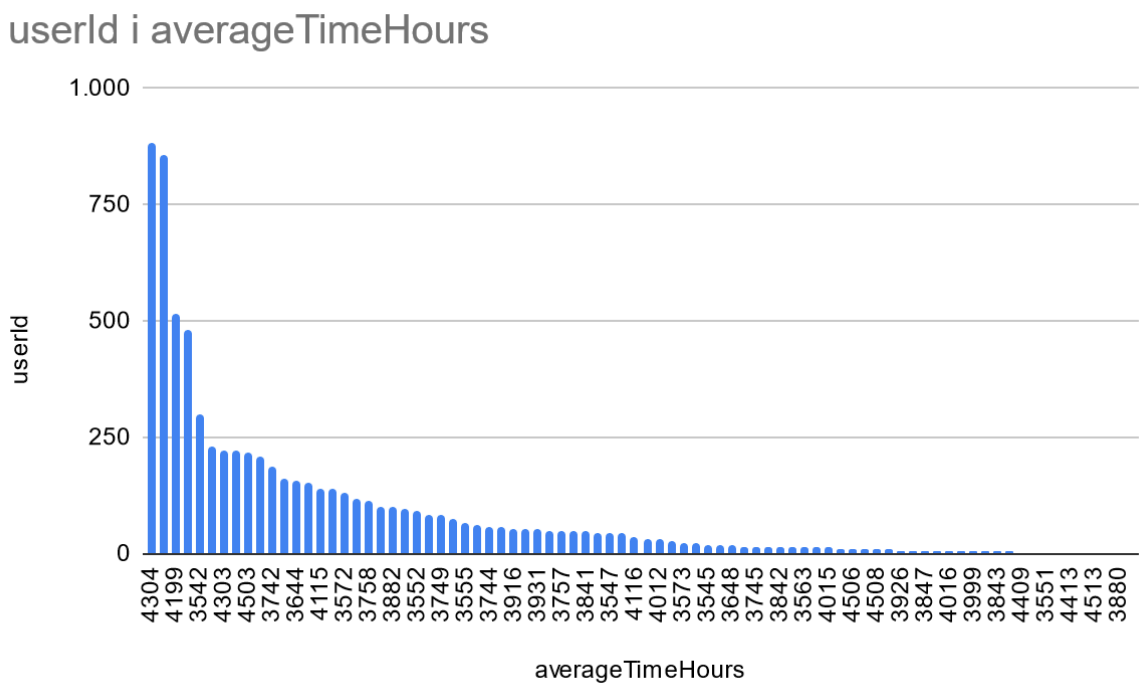
- Average Time Between Sessions

The average time between sessions in hours, we first calculated it in different users, but later with the same code we did an average of all users that stands at 639 h, roughly 26 days.

```sql
SELECT s1.userId,
AVG(TIMESTAMPDIFF(HOUR, s1.endSession, s2.startSession)) AS averageTimeHours
FROM Sessions s1
JOIN Sessions s2 ON s1.userId = s2.userId
AND s1.startSession < s2.startSession
WHERE s1. endSession IS NOT NULL AND
s2.startSession IS NOT NULL AND
s1.endSession < s2.startSession
GROUP BY s1.userId
ORDER BY averageTimeHours DESC
```

userId	averageTimeHours 1
4304	820.5707
4205	797.3257
4031	462.9093
4199	419.5220
3542	293.4203
3993	222.1917
3642	202.2296
3742	181.5119
4408	177.5778
4503	165.0427

averageTimeTotal
639.6162



- Monetization metrics
  - ARPU & ARPPU:

The ARPU and ARPPU calculated together, one interesting fact is the average of 5 items bought for every user that bought.

```sql
SELECT
COUNT(DISTINCT p.userId) AS playersThatBought,
COUNT(p.userId) as totalPurchases,
COUNT(DISTINCT pl.userId) as totalPlayers,
SUM(Price) / COUNT(DISTINCT pl.userId) as ARPU,
SUM(Price) / COUNT(DISTINCT p.userId) as ARPPU
FROM Purchases p
JOIN Items i ON p.itemId = i.Id
RIGHT JOIN Players pl on pl.userId = p.userId
ORDER BY p.purchaseId DESC
```

playersThatBought	totalPurchases	totalPlayers	ARPU	ARPPU
109	598	1000	6.922019889593124	63.50476962929472