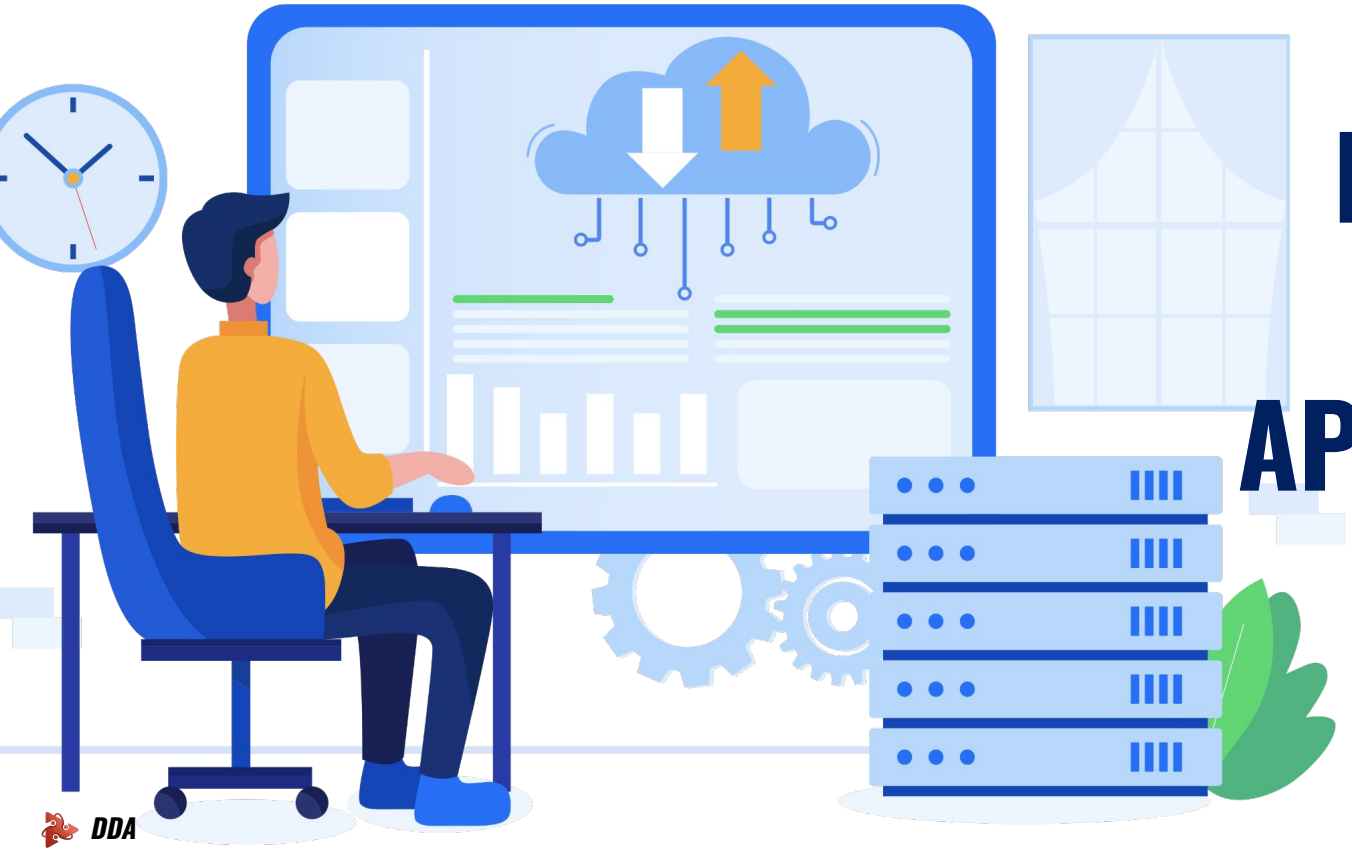


(DDA) 2.2

DEVELOPING DYNAMIC APPLICATIONS

2021

DDA

RECAP

Week 1

Learning Objectives

1. Set up Firebase
2. Working with JSON
3. Working with Firebase

C# + Unity + JSON

JsonUtility.ToJson

Declaration

public static string **ToJson**(object **obj**);

Declaration

public static string **ToJson**(object **obj**, bool **prettyPrint**);

Parameters

obj	The object to convert to JSON form.
prettyPrint	If true, format the output for readability. If false, format the output for minimum size. Default is false.

Generate a JSON representation of the **public** fields of an object.

Reading Reference
https://docs.unity3d.com/ScriptReference/JsonUtility.ToJson.html

DDA

COVERAGE

Week 2

Learning Objectives

1. JSON Tree & Data Manipulation
2. Querying Data in Firebase
3. CRUD
4. Authentication w/ Firebase
5. CA: Data Challenge - Creating a CRUD
6. CA: Design your Logins & Leaderboards

FIREBASE
#JSONTREE

C# Classes w/ JSON - Things to Note

```
/* Example of a User class */
public class User
{
    //public - access modifiers (seen outside the class)
    //required when we are using JsonUtility
    public string username;
    public string email;

    //private is default for all classes in c#
    //private only seen inside the class
    private int age = 10;

    //empty constructor
    public User()
    {

    }

    //constructor with 2 values
    public User(string username, string email)
    {
        //this refers to the properties in the class
        this.username = username;
        this.email = email;
    }
}
```

Got to set class properties to **public** access modifier
Required when we want to use JsonUtility.ToJson()

This property age won't be ported over when using JsonUtility.ToJSON()

https://docs.unity3d.com/ScriptReference/JsonUtility.ToJson.html

```
User squid = new User();
squid.username = "Oh Ii-Nam";
squid.email = "iinam@squid.com";
Debug.Log(JsonUtility.ToJson(squid));
//Output Json
//age variable value is missing here
//{"username": "Oh Ii-Nam", "email":
"iinam@squid.com"}
```

Revisiting JSON - What you should **NOT** do…

JSON has the following syntax.

- **Objects** are enclosed in braces (`{}`), their name-value pairs are separated by a comma (`,`), and the name and value in a pair are separated by a colon (`:`). Names in an object are strings, whereas values may be of any of the seven value types, including another object or an array.
- **Arrays** are enclosed in brackets (`[]`), and their values are separated by a comma (`,`). Each value in an array may be of a different type, including another array or an object.

  The order is important

When objects and arrays contain other objects or arrays, the data has a tree-like structure.

We can use them together.

Revisiting JSON - What you should **NOT** do…

```
{
   "levels": [
    {
        "name": "The Beginning",
   "enemies": ["archers", "skeletons", "lava monsters"]
  }

    {
        "name": "Dawn",
   "enemies":["slimes", "wild boars",]
  }

    {
        "name": "Dusk",
  "enemies":["vampires", "zombies",]
  }
  ]

  "characters":[
  {
        "name": "Maria",
    "abilities": "healing",
    "description": "A local priest that prays religiously",}


    {
        "name": "Max",
    "abilities": "sentry gun",
    "description": "A maniac from the other worlds",}


    {
        "name": "Pipo",
    "abilities": "transformation",
    "description": "Humanoid from moon",}
  ]
```

Invalid JSON
Missing commas
Additional commas

```
"leaderboard":[
  {
      "name": "Player 1",
  "score": "2142",
  }

  {
      "name": "Player 2",
  "score": "2542",
  }

  {
      "name": "Player 3",
  "score": "4142",
  }
]
}
```

JSON File submitted as…
JSON.cs ✘
`yourfilename.json`

Try and fix

https://jsoneditoronline.org/#left=cloud.da2b3fd10f0f4ec48b79147ff8f64df0

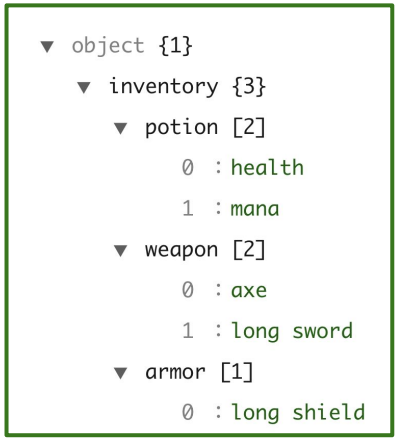
Revisiting JSON - What you should **NOT** do…

```
{
  "playerID": "",
  "inventory": [
    "potion",
    "weapon",
    "armour/armor"
    ],
  "stats": [
    "gold",
    "cdr",
    "pDmg",
    "mDmg",
    "mHP",
    "brainDmg",
    "atkSpd",
    "mana",
    "shield"
    ],
  "weightLimit": [
    "lightWeight",
    "mediumWeight",
    "heavyWeight"
    ]
}
```

Usage of slash

** Although this gives valid json, it does not make much sense to store it with a slash*
Forward slashes are special characters. If it was to be used in as a value, then it has to be properly escaped using backslash
"\/armor"

Missing data



Wrong Usage of arrays []

We would use arrays to contain values, however in this example it is containing keys or properties of the inventory
Potion? What potion?
Weapon? What weapon?

How can we store the values?

```
"inventory": {
  "potion": ["health", "mana"],
  "weapon": ["axe", "long sword"],
  "armor": ["long shield"]
}
```

List of simple values stored using arrays

Revisiting JSON - What you should **NOT** do…

```
{
  'Player Leaderboard" : {
    "Rank 1" : {
      "key" : "-MmWj7m7iMwQPCIgl5Cf",
      "name" : "Dan",
      "score" : 2000
    },
    "Rank 2" : {
      "key" : "-MmWj7m7iMwQPCIgl5Cg",
      "name" : "Kirdesh",
      "score" : 1900
    },
    "Rank 3" : {
      "key" : "-MmWj7m7iMwQPCIgl5Ch",
      "name" : "Jonathan",
      "score" : 1870
    },
    "Rank 4" : {
      "key" : "-MmWj7m7iMwQPCIgl5Ci",
      "name" : "Lin",
      "score" : 1790
    },
    "Rank 5" : {
      "key" : "-MmWj7m7iMwQPCIgl5Cj",
      "name" : "Zavier",
      "score" : 1660
    }
  }
```

Avoid having spaces in your JSON properties

It's ok to have spaces in values but not so in the properties

Try and fix

https://jsoneditoronline.org/#left=cloud.cc366249773542b295f9b3fe6bbf3547

Revisiting JSON - What you can do

```
{
  "player-stats": {
    "name": "yes",
    "level": "1000",
    "hp": "100",
    "mana": "500",
    "damage": "10",
    "armor": "5"
  },
  "player-state": {
    "alive": true,
    "walking": false,
    "sprinting": true
  },
  "status-effect": {
    "poisoned": true,
    "burning": false,
    "drowsy": false,
    "bleeding": false
  }
}
```

Usage of strings for numeric values

Consider using numeric values instead of using quotes.

JSON will still be able to read both ways, just that numeric values have a better structure

Usage of true/false values

Using true or false values allows us to quickly determine the current state of events.

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

Revisiting JSON - Best Practices

Instead of Hyphens use underscores (_). But using lower case or camelCase is the best. See samples below.

```
{"first-name":"Rachel","last-name":"Green"}  is not right. ✘
{"first name":"Rachel","last name":"Green"} is okay ✓
{"firstname":"Rachel","lastname":"Green"} is okay ✓
{"firstName":"Rachel","lastName":"Green"} is the best. ✓
```


Reading Reference : Best practices for JSON
https://json-ld.org/spec/latest/json-ld-api-best-practices/#bp-summary

Revisiting JSON - Best Practices

When possible, property values *SHOULD* use native JSON datatypes such as numbers (*integer*, *decimal* and *floating point*) and booleans (`true` and `false`).

```
{
  "canPigsFly": null,      // null
  "areWeThereYet": false, // boolean
  "answerToLife": 42,      // number
  "name": "Bart",          // string
  "moreData": {},          // object
  "things": []             // array
}
```

If a property requires quotes, double quotes must be used. All property names must be surrounded by double quotes. Property values of type string must be surrounded by double quotes. Other value types (like boolean or number) should not be surrounded by double quotes.

Reading Reference : Best practices for JSON
https://json-ld.org/spec/latest/json-ld-api-best-practices/#bp-summary

Revisiting JSON - Best Practices

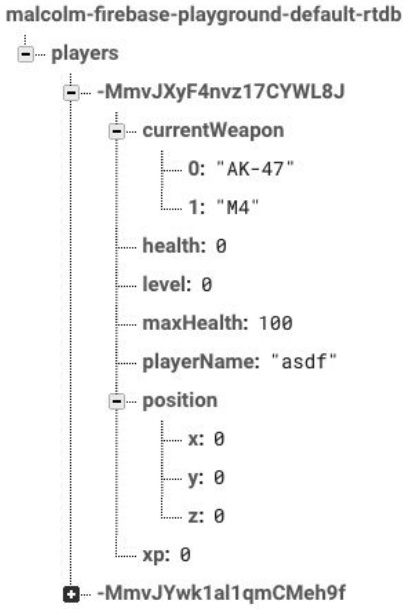
- Property names should be meaningful names with defined semantics.
- Property names must be camel-cased, ascii strings.
- The first character must be a letter, an underscore (_) or a dollar sign ($).
- Subsequent characters can be a letter, a digit, an underscore, or a dollar sign.

```
{
    "thisPropertyIsAnIdentifier": "identifier value"
}
```

If a property requires quotes, double quotes must be used. All property names must be surrounded by double quotes. Property values of type string must be surrounded by double quotes. Other value types (like boolean or number) should not be surrounded by double quotes.

Reading Reference : Best practices for JSON
https://json-ld.org/spec/latest/json-ld-api-best-practices/#bp-summary

Working JSON + C# + Classes



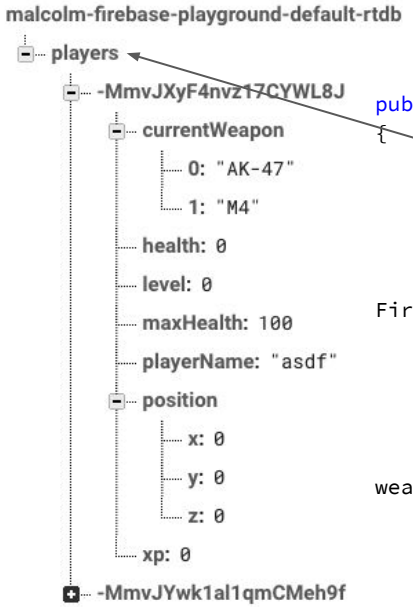

```
public class GamePlayer
{
    public string playerName;
    public int level;
    public int xp;
    public int health;
    public int maxHealth;
    public Vector3 position;
    public List<string> currentWeapon;

    public GamePlayer(){}

    public GamePlayer(string playerName, int level, int xp,
        int health, int maxHealth,
       )
    {
        this.playerName = playerName;
        this.level = level;
        this.xp = xp;
        this.health = health;
        this.maxHealth = maxHealth;
        this.position = position;
        this.currentWeapon = currentWeapon;
    }

}
```

Set to public for converting to JSON private access modifiers won't be converted

This is our **GamePlayer class** that defines a blueprint for us to create GamePlayer objects

Notice we can skip using MonoBehaviour as we are not using any Unity features at this point

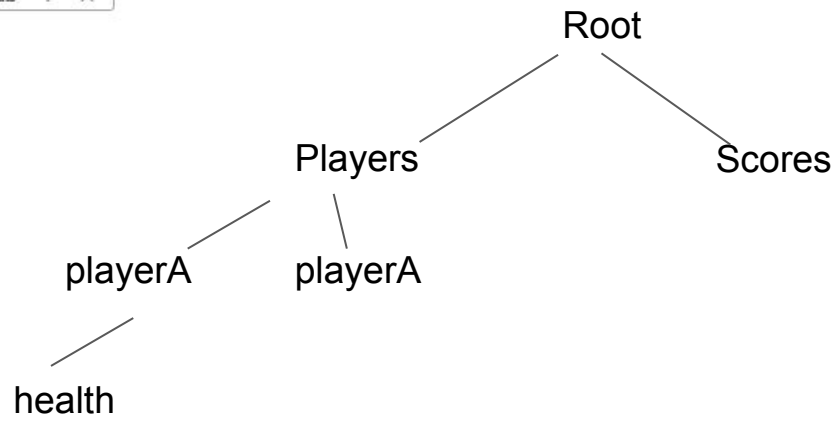
Working JSON + C# + Classes




```
public class PlayerWeaponManager : MonoBehaviour
{
    public DatabaseReference dbReference;

    private void Awake()
    {
        dbReference =
FirebaseDatabase.DefaultInstance.RootReference;
        List<string> weapons = new List<string>();
        weapons.Add("AK-47");
        weapons.Add("M4");

        CreateNewPlayer("Royden", new Vector(0,0,0),
weapons );

    }
```

```
public void CreateNewPlayer(DatabaseReference dbRef, string
playerName, Vector3 position, List<string> currentWeapon,
int level = 0, int xp = 0, int health = 0, int maxHealth =
100)
{
    GamePlayer player = new GamePlayer(playerName, level,
            xp, health, maxHealth, position,currentWeapon);
    //Overwrites the root of the tree
    //dbRef.SetRawJsonValueAsync(player.SaveToJson());

    //Overwrites the node location "players"
//dbRef.Child("players").SetRawJsonValueAsync(player.SaveTo
Json());

    //Writes to node location "players" /players/$key
    //creates a key
    var playerReference = dbRef.Child("players").Push();
playerReference.SetRawJsonValueAsync(player.SaveToJson());
    }
}//end class
```

Mapping to a JSON Tree



Structuring your Data

```
{
  "players": {
    "harrypotter": {
      "name": "Harry Potter",
      "weapons": { "ak46": true },
    },
    "nuggetwarrior": { ... },
    "samuraiburger": { ... }
  }
}
```

For example, consider a MMO game application that allows players to store a basic profile and weapon inventory.

A typical player profile is located at a path, such as /players/$pid. The player **harrypotter** might have a database entry that looks something like this:

If you create your own keys, they must be UTF-8 encoded, can be a maximum of 768 bytes, and cannot contain ., $, #, [,], /, or ASCII control characters 0-31 or 127.
You cannot use ASCII control characters in the values themselves, either.

Structuring your Data - The Bad

```
{
  // This is a poorly nested data architecture, because iterating the children
  // of the "chats" node to get a list of conversation titles requires
  // potentially downloading hundreds of megabytes of messages
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "messages": {
        "m1": { "sender": "ghopper", "message": "Relay malfunction found. Cause:
moth." },
        "m2": { ... },
        // a very long list of messages
      }
    },
    "two": { ... }
  }
}
```

With this nested design, iterating through the data becomes problematic.

For example, listing the titles of chat conversations requires the **entire chats tree**, including all members and messages, to be downloaded to the client.

Path to take: /chats/$key/title

Retrieves ALL messages and details

Structuring your Data - The Good

```
{
  // Chats contains only meta info
about each conversation
  // stored under the chats's unique ID
  "chats": {
    "one": {
      "title": "Historical Tech
Pioneers",
      "lastMessage": "ghopper: Relay
malfunction found. Cause: moth.",
      "timestamp": 1459361875666
    },
    "two": { ... },
    "three": { ... }
  },
```

```
// Conversation members are
easily accessible
// and stored by chat
conversation ID
  "members": {
    // we'll talk about indices
like this below
    "one": {
      "ghopper": true,
      "alovelace": true,
      "eclarke": true
    },
    "two": { ... },
    "three": { ... }
  },
```

```
// Messages are separate from data
we may want to iterate quickly
  // but still easily paginated and
queried, and organized by chat
  // conversation ID
  "messages": {
    "one": {
      "m1": {
        "name": "eclarke",
        "message": "The relay seems
to be malfunctioning.",
        "timestamp": 1459361875337
      },
      "m2": { ... },
      "m3": { ... }
    },
    "two": { ... },
    "three": { ... }
  }
}
```

Go as Flat and Lead

To take Title

Path to take: /chats/$key/title

Details of members and messages are not retrieved

Structuring your Data - Scalable Data

```
// An index to track Ada's memberships
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      // Index Ada's groups in her profile
      "groups": {
        // the value here doesn't matter, just
that the key exists
        "techpioneers": true,
        "womentechmakers": true
      }
    },
    ...
  },
```

Quickly and efficiently fetch Ada's memberships,

/users/$uid/groups/$group_id
Check whether it is null

```
"groups": {
    "techpioneers": {
      "name": "Historical Tech Pioneers",
      "members": {
        "alovelace": true,
        "ghopper": true,
        "eclarke": true
      }
    },
    ...
  }
}
```

What's needed is an elegant way to list the groups a user belongs to and fetch only data for those groups. An *index* of groups can help a great deal here:

We can create 2-way relationships.
This is using Indexing Methods

This duplicates some data by storing the relationship under both Ada's record and under the group.

Now alovelace is indexed under a group, and techpioneers is listed in Ada's profile. So to delete Ada from the group, it has to be updated in two places.

So what's #CRUD

Reading your Data

```
using System.Threading.Tasks;
using Firebase.Extensions;// for ContinueWithOnMainThread
```

Include **Firebase Extensions**.
This is to handle Threading

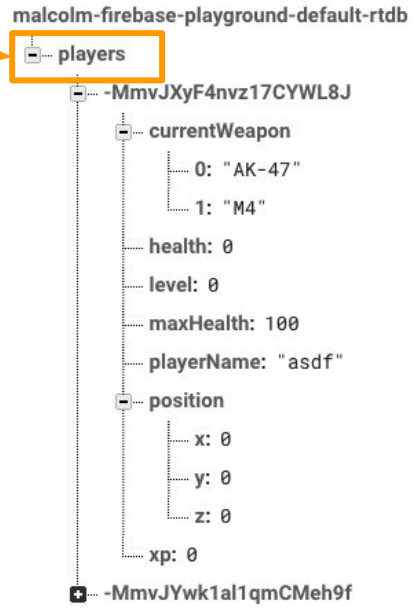
```
FirebaseDatabase.DefaultInstance
    .GetReference("players")
    .GetValueAsync().ContinueWithOnMainThread(task =>
{
        if (task.IsFaulted) {
          // Handle the error...
        }
        else if (task.IsCompleted) {
          DataSnapshot snapshot = task.Result;
          // Do something with snapshot...
        }
      });
```

Depending on how you structure, get the reference node of your JSON tree. This is the json path (“path/to/query…. .”) eg. ("players/…..”)

We use **ContinueWithOnMainThread** to use the main thread that Unity is using.

Other alternatives is to use Coroutines

** You might see examples using ContinueWith(..), this uses a new thread on your computer*



The task result will contain a snapshot containing all data (entries) at that location, including child data. If there is no data, the snapshot returned is null.

https://firebase.google.com/docs/reference/unity/clas…

https://firebase.google.com/docs/database/unity/retrieve-data

READING
#CRUD
#DEMO

Working with Handlers

Listen for events

You can add event listeners to subscribe on changes to data:

Event	Typical usage
`ValueChanged`	Read and listen for changes to the entire contents of a path.
`ChildAdded`	Retrieve lists of items or listen for additions to a list of items. Suggested use with `ChildChanged` and `ChildRemoved` to monitor changes to lists.
`ChildChanged`	Listen for changes to the items in a list. Use with `ChildAdded` and `ChildRemoved` to monitor changes to lists.
`ChildRemoved`	Listen for items being removed from a list. Use with `ChildAdded` and `ChildChanged` to monitor changes to lists.
`ChildMoved`	Listen for changes to the order of items in an ordered list. `ChildMoved` events always follow the `ChildChanged` event that caused the item's order to change (based on your current order-by method).

Reading Reference

https://firebase.google.com/docs/reference/unity/class/firebase/extensions/task-extension

https://firebase.google.com/docs/database/unity/retrieve-data

Working with Handlers

```
FirebaseDatabase.DefaultInstance
        .GetReference("Leaders")
        .ValueChanged += HandleValueChanged;
    }

    void HandleValueChanged(object sender,
ValueChangedEventArgs args) {
      if (args.DatabaseError != null) {

Debug.LogError(args.DatabaseError.Message);
        return;
      }
      // Do something with the data in
args.Snapshot
    }
```

ValueChangedEventArgs contains a **DataSnapshot** that contains the data at the specified location in the database at the time of the event. Calling **Value** on a **snapshot returns a Dictionary<string, object>** representing the data. If no data exists at the location, calling Value returns null.

In this example, **args.DatabaseError** is also examined to see if the read is canceled. For example, a read can be canceled if the client doesn't have permission to read from a Firebase database location. The DatabaseError will indicate why the failure occurred.

Reading Reference

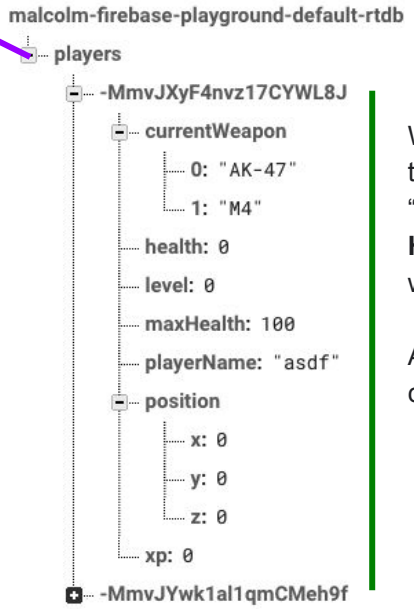
https://firebase.google.com/docs/reference/unity/class/firebase/extensions/task-extension

https://firebase.google.com/docs/database/unity/retrieve-data

Working with Handlers

```
DatabaseReference playerRef =
FirebaseDatabase.DefaultInstance.GetReference("players");

playerRef.ValueChanged += HandlePlayerValueChanged;//parent node
has changes
void HandlePlayerValueChanged(object send, ValueChangedEventArgs
args)
   {
       if (args.DatabaseError != null)
       {
           Debug.LogError(args.DatabaseError.Message);
           return;
       }

       numPlayers = (int)args.Snapshot.ChildrenCount;
       txtPlayerCount.text = "Total Players in Game: " +
numPlayers;

       Debug.Log("HandlePlayerChange numChild:" + numPlayers);
   }
```

In this example, we are having the listener function to update the total number of players in the game



Whenever there's movement in the nodes under parent node "players", the listener **HandlePlayerValueChanged()** will be triggered.

A snapshot of the node and its children will be send back.

Reading Reference

https://firebase.google.com/docs/reference/unity/class/firebase/extensions/task-extension

https://firebase.google.com/docs/database/unity/retrieve-data

HANDLERS
#CRUD
#DEMO

CRUD - Update/Write

Basic write operations

For basic write operations, you can use SetValueAsync() to save data to a specified reference, replacing any existing data at that path. You can use this method to pass types that correspond to the available JSON types as follows:

- string
- long
- double
- bool
- Dictionary<string, Object>
- List<Object>

If you use a typed C# object, you can use the built in JsonUtility.ToJson() to convert the object to raw Json and call **SetRawJsonValueAsync().** For example, you may have a User class that looked as follows:

Reading Reference
https://firebase.google.com/docs/reference/unity/class/firebase/extensions/task-extension
https://firebase.google.com/docs/database/unity/retrieve-data

CRUD - Update

Map the given path and key, then set the desired value

```
DatabaseReference dbReference = FirebaseDatabase.DefaultInstance.RootReference;
dbReference.Child("/path/to/save").SetValueAsync(playerName);

//dbPlayerReference.Child(playerKey).Child("playerName").SetValueAsync(playerName);

DatabaseReference reference = FirebaseDatabase.Instance.GetReference("path/to/save");
reference.UpdateValueAsync(new Dictionary<string, object>(){
    {"child1","value1"},{"child2","value2"}
}, 10, (res) =>
{
    if (res.success)
    {
        Debug.Log("Write success");
    }
    else
    {
        Debug.Log("Write failed : " + res.message);
    }
});
```

Reading Reference

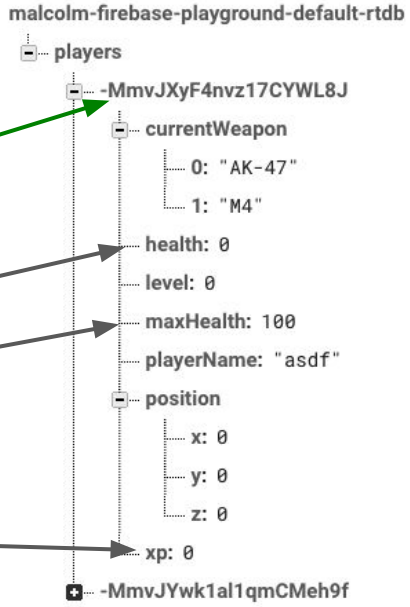
https://firebase.google.com/docs/database/unity/retrieve-data

https://firebase.google.com/docs/reference/unity/class/firebase/database/database-reference

CRUD - Update

Map the given path and key, then set the desired value

```
DatabaseReference dbReference = FirebaseDatabase.DefaultInstance.RootReference;
dbReference.Child("/path/to/save").SetValueAsync(playerName);

//dbPlayerReference.Child(playerKey).Child("playerName").SetValueAsync(playerName);

DatabaseReference reference =
FirebaseDatabase.DefaultInstance.GetReference("players/");
// Handle the error...
string id = "-MmvJXyF4nvz17CYWL8J";
Dictionary<string, object> childUpdates = new Dictionary<string, object>();
childUpdates[id + "/health"] = 456;
childUpdates[id + "/maxHealth"] = 456;
childUpdates[id + "/xp"] = 456;

reference.UpdateChildrenAsync(childUpdates);
```



Reading Reference

https://firebase.google.com/docs/database/unity/retrieve-data

https://firebase.google.com/docs/reference/unity/class/firebase/database/database-reference

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=net-5.0

Updating
#CRUD
#DEMO

CRUD - Delete

The simplest way to delete data is to call **RemoveValueAsync**() on a reference to the location of that data.

You can also delete by specifying null as the value for another write operation such as SetValueAsync() or UpdateChildrenAsync(). You can use this technique with UpdateChildrenAsync() to delete multiple children in a single API call.

```
dbRef.OrderByChild("playerName").EqualTo(playerToSearch).Reference.RemoveValueAsync();
```

What the code means
dbRef - The FirebaseDatabase reference you are using
OrderByChild - this is an ordering / sorting feature of Firebase. It sorts by the desired child property
EqualTo - Does a filter based on the child given, in this case "playerToSearch"
Reference - Retrieves a snapshot of the found nodes
RemoveValuesAsync - Makes a call to **permanently remove** the nodes affected on that location

Reading Reference
https://firebase.google.com/docs/database/unity/retrieve-data

Deleting
#CRUD
#DEMO

Writing #CRUD #DEMO

#CA

Firebase - CRUD Data (SOLO)

In this activity, we work with Realtime Databases to allow us to Create, Read, Update, Delete data

1. Create a very simple CRUD system based on JSON or a simple Player Class

CA GOLD: Create relevant UI and additional relevant classes to be stored (e.g Leaderboard)

Deliverables

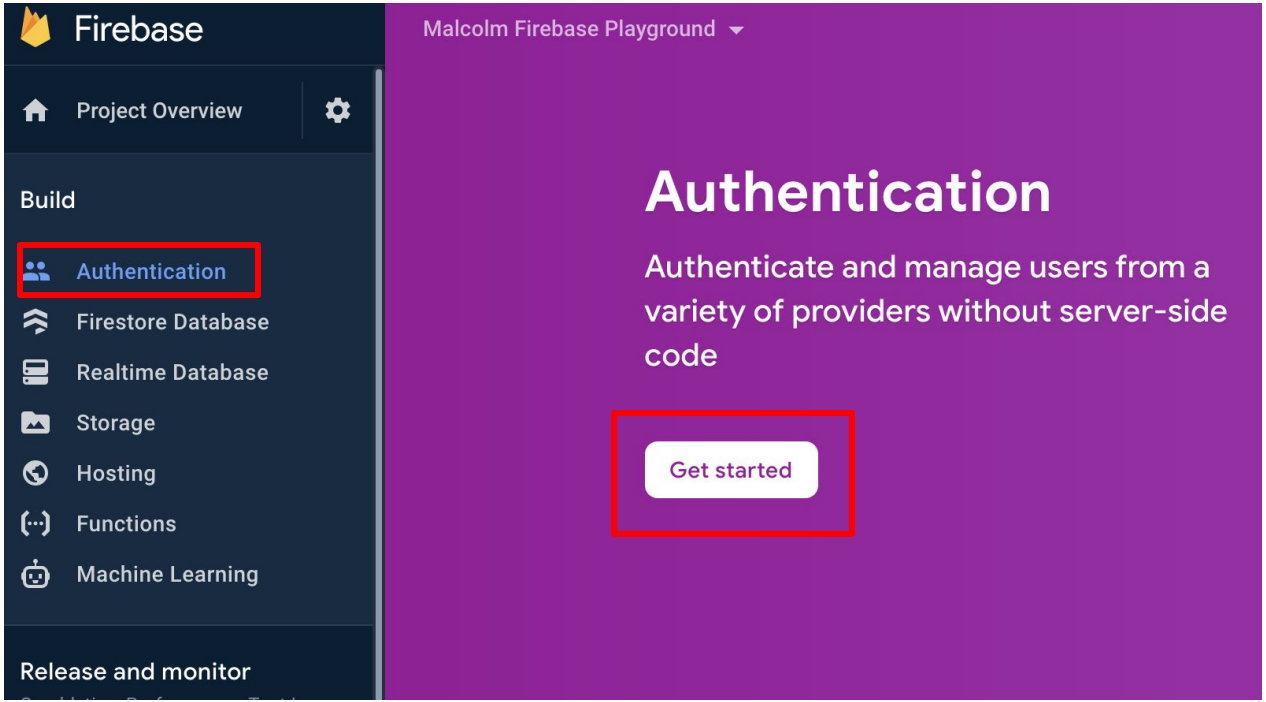
.unitypackage of your scene, scripts, any useful related files (don't need Firebase SDK etc)

Before Next Class

Reference

https://firebase.google.com/docs/database/unity/save-data

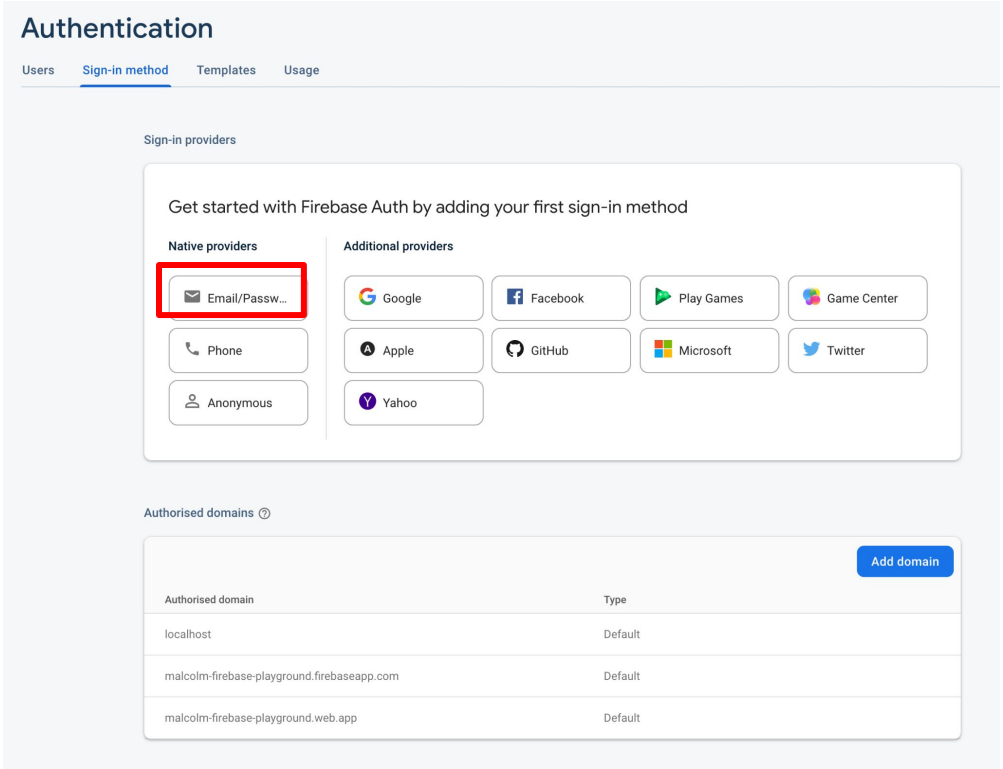
Creating Authentication - Getting Started (1)



We will take on Firebase Authentication Provides a powerful authentication system by email, OAuth, Auth Providers (sign-in) options

Start by Going to your Firebase projects
> Click on Authentication
> Click on **Get Started**

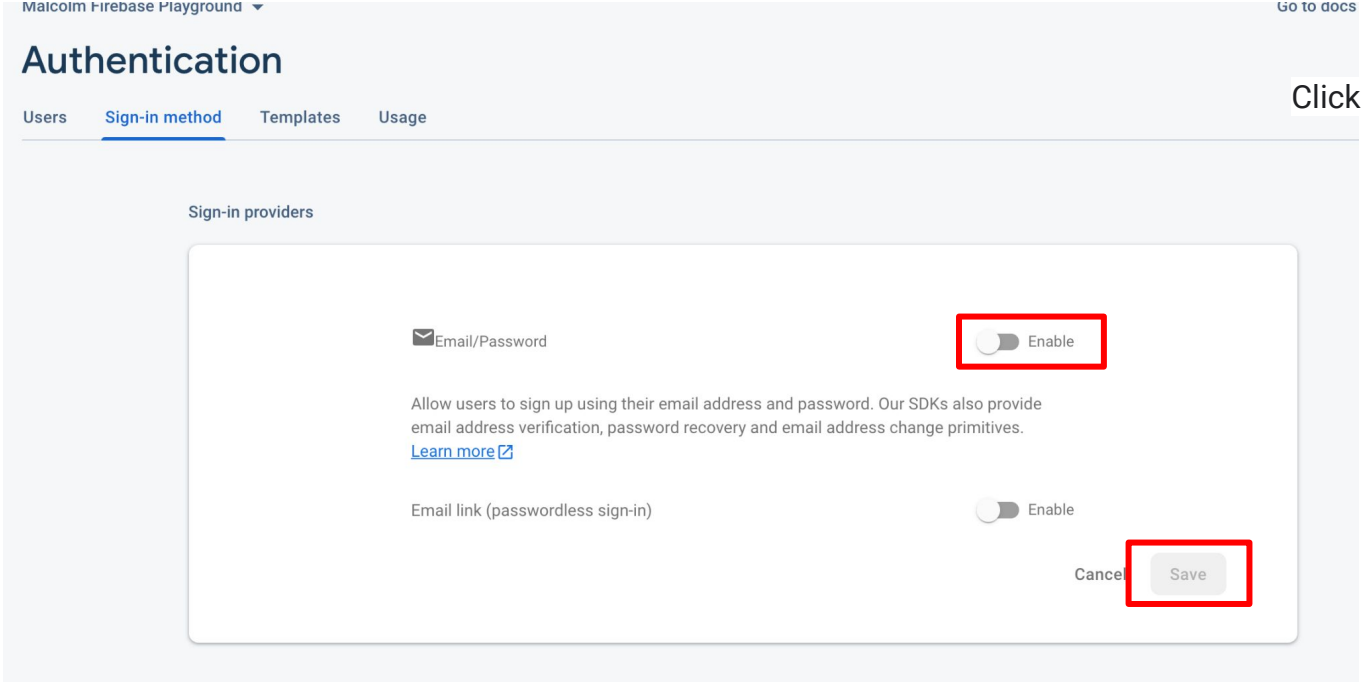
Creating Authentication - Getting Started (2)



Firebase Auth provides a ready stream of authentication methods.

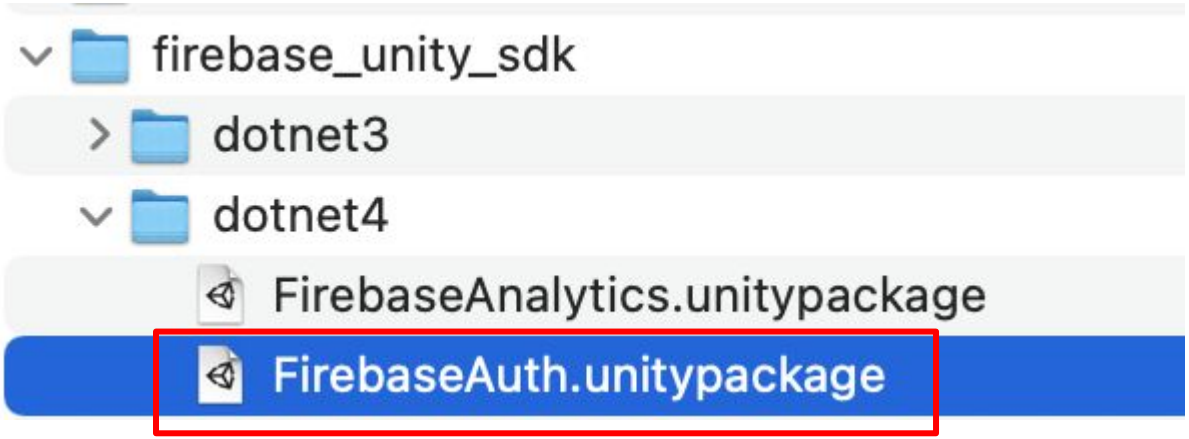
We will use Email/Password method and build up along the way

Creating Authentication - Getting Started (3)



Click on **Enable** and then **Save**

Creating Authentication - Getting Started (4)



Go to your Unity Project
Import Package > Custom Package
>> Choose from “Firebase SDK folder”
>> donet4
>> **FirebaseAuth.unitypackage**

Pre-requisites
Unity 2020.3.20f

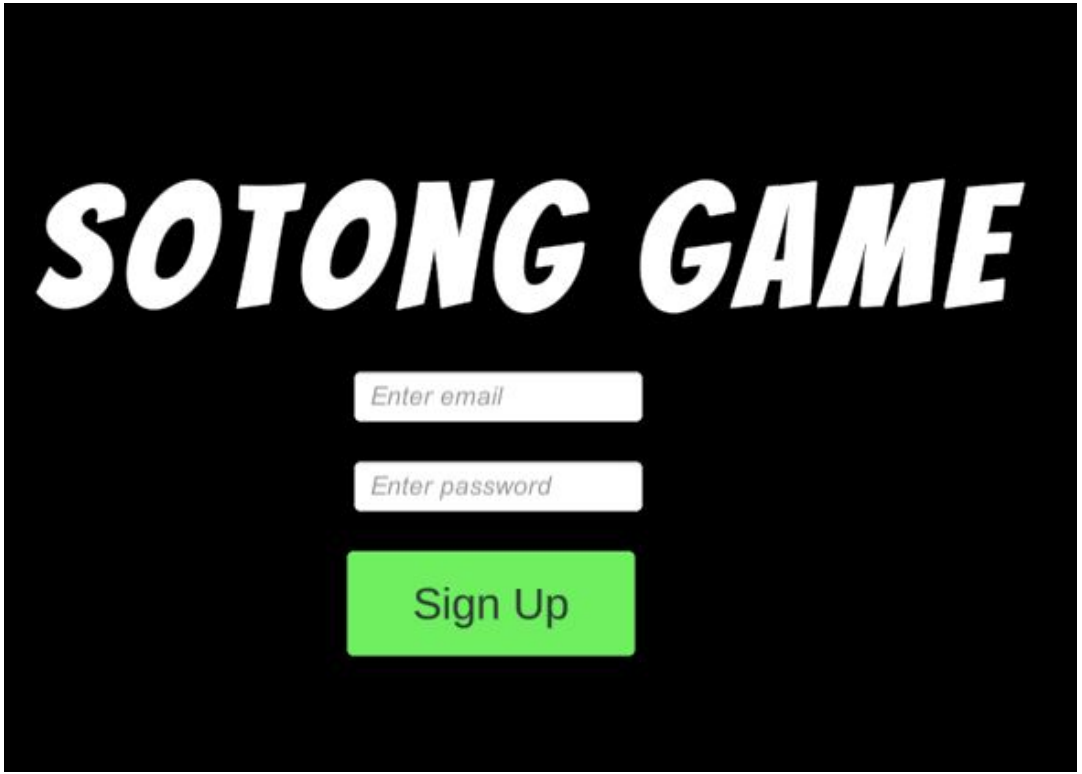
Working Firebase project with the following properly done up

- Firebase Authentication
- Firebase Realtime Database
- FirebaseDatabase.unitypackage
- google-services.json

Creating Authentication - Create simple form

Before you can use Firebase Authentication, you need to:

- Register your Unity project and configure it to use Firebase.
  - If your Unity project already uses Firebase, then it's already registered and configured for Firebase.
  - If you don't have a Unity project, you can download a sample app.
- Add the Firebase Unity SDK (specifically, FirebaseAuth.unitypackage) to your Unity project.



Creating Authentication - Code Basics

```
//place other needed directives
using Firebase.Auth;

public class AuthManager : MonoBehaviour
{
   Firebase.Auth.FirebaseAuth auth;

   //initialize our auth instance
   private void Awake()
   {
       auth = FirebaseAuth.DefaultInstance;

   }
//methods to handle authentication
```

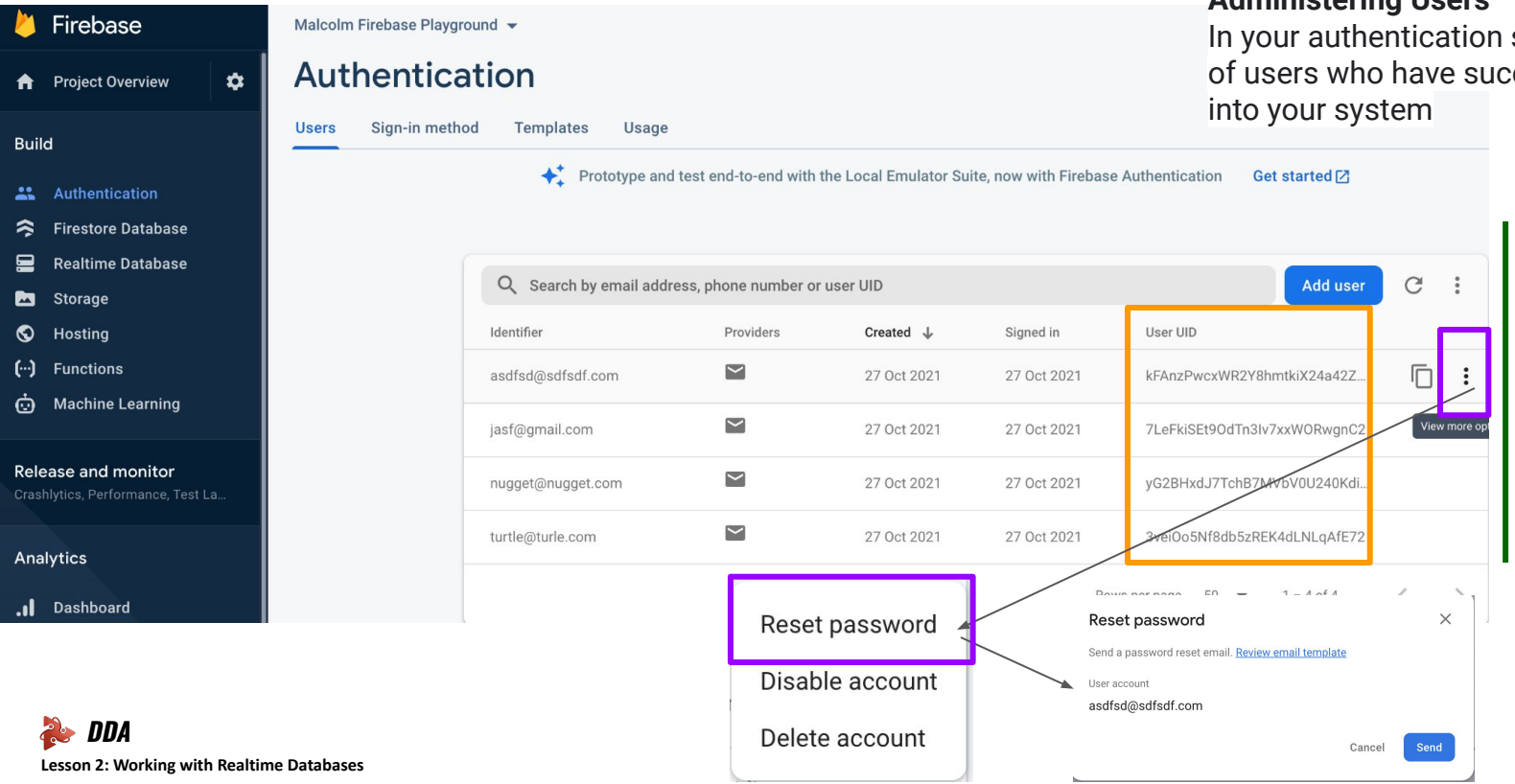
```
//automatically pass user info to the firebase project
//attempt to create new user or check with there's already one
auth.CreateUserWithEmailAndPasswordAsync(email, password).ContinueWith(task =>
{
    //perform task handling
    if(task.IsFaulted || task.IsCanceled)
    {
        Debug.LogError("Sorry, there was an error creating your new account,
ERROR: " + task.Exception);
        return;//exit from the attempt
     }else if (task.IsCompleted)
     {
         Firebase.Auth.FirebaseUser newPlayer = task.Result;
         Debug.LogFormat("Welcome to Sotong Games {0}", newPlayer.Email);
         //do anything you want after player creation eg. create new player

     }
});
```

Creating Users

Perform validation and checks on your form, then pass the details to the `CreateUserWithEmailAndPasswordAsync` This will create the user IF the email is a proper email and the password meets Auth requirements
Note that the function executes separately

Reading Reference
https://firebase.google.com/docs/auth/unity/start

Creating Authentication - Checking the works



Administering Users
In your authentication service, you get a list of users who have successfully registered into your system

List of Users
User UID - is a unique key generated by Firebase. We can adopt this into our Players data structure

Options
We can send a password reset email to the users

Making #AUTH #DEMO

#CA

Firebase - Simple Auth (SOLO)

In this activity, we will create a simple auth system with user sign up by email and password

1. Create a simple authentication with email and password

CA GOLD: Make pretty UI for your login panels

Deliverables

.unitypackage of your scene, UI, scripts

Before Next Class

Reference

https://firebase.google.com/docs/auth/unity/start

Creating Authentication - What's Next?

Start Making Your Players
We can adopt the generated unique User UID when a new user is created. That can be our $key for new players into the system.

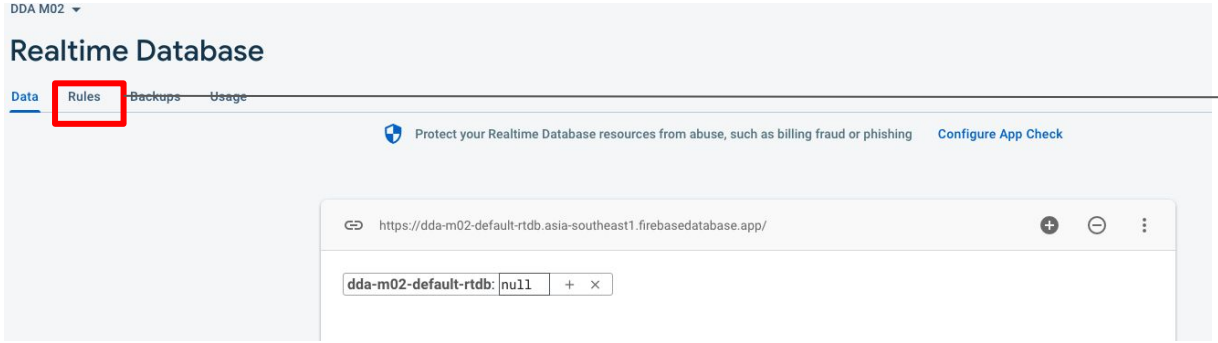
Users can further update their profile details using a data structure in your Realtime Database

Start Modifying Your Reset Emails

Start Creating User Related Features
Some features to think about are:

- Logout
- Login errors
- Modifying email verification email
- handling Authentication issues
- storing the User UID in the game,
- Anything else that is user related :)
- Checking for unique usernames
- Using Multi-Sign in providers options
- Handling Lost password

Common Error: Everything is ok but database not updated!



FIX:
Change the rules in your Database

This is because it was not set to test mode earlier and permissions were locked

Setting up public access

The Realtime Database provides a declarative rules language that allows you to define how your data should be structured, how it should be indexed, and when your data can be read from and written to.

Note: By default, read and write access to your database is restricted so only authenticated users can read or write data. To get started without setting up Authentication, you can configure your rules for public access. This does make your database open to anyone, even people not using your app, so be sure to restrict your database again when you set up authentication.

WEEK 3

NEXT WK HOME BASED!

Lesson Mode
SYNCHRONOUS
MS TEAMS

Let's Make #GAME #Auth2

READING

Best practices for JSON https://json-ld.org/spec/latest/json-ld-api-best-practices/#bp-summary
JSON Style Guide https://google.github.io/styleguide/jsoncstyleguide.xml#Comments
Firebase Errors https://firebase.google.com/docs/reference/unity/class/firebase/database/database-error
Learning Classes in C# https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class
JsonUtility to JSON https://docs.unity3d.com/ScriptReference/JsonUtility.ToJson.html
Introduction to TextMesh Pro https://www.raywenderlich.com/22175776-introduction-to-textmesh-pro-in-unity
Firebase Authentication https://firebase.google.com/docs/auth/unity/start