Oide
Tacú leis an bhFoghlaim Ghairmiúil i measc Ceannairí Scoile agus Múinteoirí
Supporting the Professional Learning of School Leaders and Teachers

LCCS JavaScript Workshop

Day 3

A tutorial on how to use JavaScript to interact with Firebase

Contents

Overview and Conventions Used

The objective of this tutorial is to demonstrate how to create a Firebase database and link it to a web application developed using HTML and JavaScript. The web application will include:
a Contact Us page which will contain an entry form that includes an email address and a <submit> button. When the user clicks on <submit> the value entered by the user will be read and sent to a Firebase database. In the second part of the tutorial the data is retrieved from the same database and displayed in a text area (and as an ordered list) on the same page.

The intention is that the tutorial would be presented as a code along activity. The teacher walks through each step of the tutorial providing a full commentary of each step and, stopping at frequent and regular intervals as the students code along. Enjoy!

The octocat (shown here to the left) is the GitHub integration symbol. Throughout this tutorial you will notice this symbol appears along with code used. When you click on the octocat you will be directed to the source code on GitHub in your web browser.

It is recommended that you copy the code from GitHub to your preferred Integrated Development Environment (IDE).

```
All HTML code appears in a salmon/pink colour like this sentence.
```

```
JavaScript code appears in green.
The Courier New font is used for all HTML and JavaScript code.
```

```
If the code appears inside a solid border like this one, it means that the
code belongs to a new file. You can choose to download the file into your
working directory/folder from GitHub or create the file manually yourself
and then copy+paste the code in.
```

```
If the code appears inside a dashed border like this one, it means it is
part of an existing file. You should copy+paste code like this into the
file it belongs to.
```

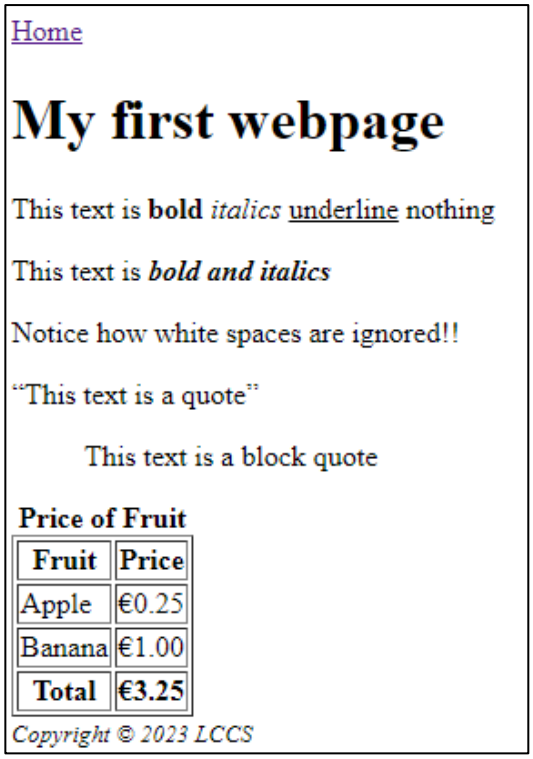
TASK 1 – Getting Started - setting up the Home Page.

Firstly, you need to decide what Development Environment you are using e.g. Notepad, Glitch, repl.it).

Create the file `index.html`. Click on the Octocat (see code listing below) and download/copy+paste the HTML code below into the platform of your choice. (If you are using Glitch, you could simply browse to the Glitch project at https://glitch.com/edit/#!/start-nw5-demo and REMIX to make your own.) This is the code for the Home Page.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Example 1</title>
    <meta charset="utf-8">
    <!-- import the webpage's stylesheet -->
    <!--link rel="stylesheet" href="style.css"-->
  </head>

  <body>

    <a href="index.html">Home</a>  
    <!-- TASK 2A: Insert a link to a contact Us page here -->


    <h1>My first webpage</h1>

    <p>This text is <b>bold</b> <i>italics </i><u>underline</u> nothing</p>
    <p>This text is <i><b>bold and italics</b></i> </p>
    <p>Notice
      how
        white
          spaces
              are

    ignored!!</p>

    <q>This text is a quote</q>

    <blockquote>This text is a block quote</blockquote>

    <table border="1">
      <caption><strong>Price of Fruit</strong></caption>
      <tr>              <!-- <tr> inserts a table row -->
        <th>Fruit</th> <!-- insert a heading cell -->
        <th>Price</th>
      </tr>
      <tr>
        <td>Apple</td> <!-- insert a data cell -->
        <td>€0.25</td>
      </tr>
      <tr>
        <td>Banana</td>
        <td>€1.00</td>
      </tr>

      <!-- TASK 1 (Modify): Add another row to the table for Pineapples (€2) -->


      <tr>
        <th>Total</th>
        <th>€3.25</th>
      </tr>
    </table>

    <small><i>Copyright &copy; 2023 LCCS</i></small>
  </body>
</html>
```

Let's walkthrough a quick PRIMM activity.



Predict.
What do you think the webpage resulting from the above code would look like?

Run.
Load the page. You should see a window that looks something like the one shown here.

Investigate.
Play with the code (`index.html`)
What can you say about white spaces?
Identify the block level elements (these are the ones that start a new line). List the *inline* elements.

Modify.
Insert a few different level headings (e.g. h3)
Insert another paragraph.
Add some lists (ordered, unordered and description).
Try using `&euro;` instead of €.
Insert a new row for pineapples into the table (see below).

Try these,

```
<tr>
  <td>Pineapples</td>
  <td>€2.00</td>
</tr>
```

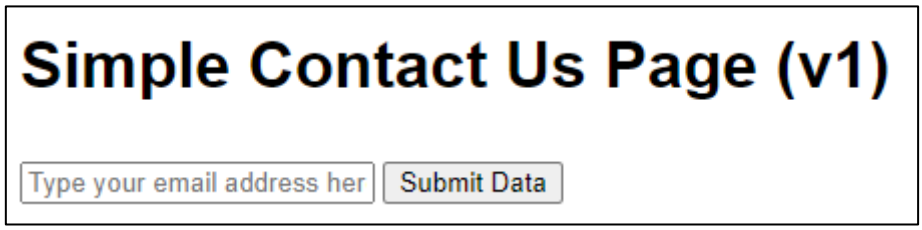
and

```
<dl>
  <dt>Coffee</dt>
  <dd>Black hot drink</dd>
  <dt>Milk</dt>
  <dd>White cold drink</dd>
</dl>
```

Make
Make a new page – see next. Remember a page is just a HTML file.

TASK 2 – Create a Contact Us page

We want to add a new page that looks like this.



Task 2A

Before we create the Contact Us page, we will add the line to link the Home Page to the Contact Us page. Open the file `index.html` and insert the line below at the correct location (indicated by the comment). You can copy and paste the code from GitHub or type it in as it appears here.

```
    <!-- TASK 2A: Insert a link to a contact Us page here -->
    <a href="contact1.html">Contact Us</a>  
```

Task 2B

Create or download the file `contact1.html`. – see below.

```
<!-- TASK 2B: Create contact1.html -->

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Contact 1</title>
    <meta charset="utf-8">
  </head>

  <body>
    <a href="index.html">Home</a>  
    <header>
      <h1>Simple Contact Us Page (v1)</h1>
    </header>
    <main>
      <form>
        <input id="email" type="text" maxlength="100" placeholder="Type
your email address here..." >
        <button type="submit" id="submit-data">Submit Data</button>
      </form>

      <!—TASK 5A: Paste the section code in here -->

    </main>
  </body>
</html>

<!-- TASK 4A: Include firebase.js here -->


<!-- TASK 2C: Include contact1.js here -->
```

Note that the value of the `id` attribute for the `input` control is *email*.
Q. What is the value of the `id` attribute for the `button` control?
(A. *submit-data*)

Now test to make sure these changes have worked. Load the Home Page.
Does the link to the Contact Us page work?
Does the link from the Contact Us page to the Home Page work?

Task 2C
In this task we will make the Contact Us page interactive. To do this we need to use JavaScript.

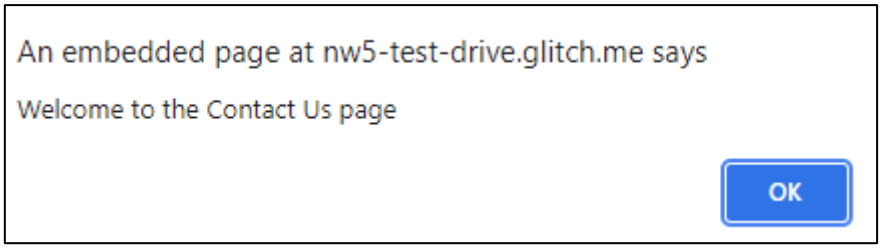
Create or download the file `contact1.js`. – see below.

```
alert("Welcome to the Contact Us page");

// TASK 4B – Copy your web app's Firebase configuration here ...

// TASK 4C

// Task 2D - Tell JavaScript to call saveContacts when SUBMIT button is clicked

// Task 5B - Code to retrieve and display the data goes here ...

// Task 5C - Code to retrieve and display the data in a list goes here ...
```

Now insert (copy+paste from GitHub or type it in yourself) the following line at the end (bottom) of `contact1.html`. This tells the browser to load and run the JavaScript code in the file `contact1.js`.

```
<!-- TASK 2C: Include contact1.js here -->
<script src="contact1.js"></script>
```

Test this change - load the Home Page again.
The JavaScript code in `contact1.js` should be executed.
You should see a message box like this displayed.



Open the `contact1.js` file again. What do you think the statement does?
`alert("Welcome to the Contact Us page");`

Comment out the `alert` statement using double slash (//).
`//alert("Welcome to the Contact Us page");`

Task 2D

We will now add in the code to display the alert box in response to the user clicking the button, <Submit Data>. Copy+paste in the following JavaScript code into the appropriate location in the file `contact1.js`:

```
// Task 2D - tell JavaScript to call saveContacts when SUBMIT button is clicked
const btn = document.getElementById("submit-data");
btn.addEventListener("click", saveContacts);

// Submit clicked so post the data to the server
function saveContacts() {
  alert("SUBMIT clicked!!!");

  // Task 2E - read the data from the email field

}  // ← GOTCHA!
```

When you run the code and click on the <Submit Data> button you should see the alert box that looks something like the one displayed below.

The above behaviour is created by the previous code snippet.
Notice that among other things we have defined a function called `saveContacts`. The function has no parameters and its body is enclosed by opening and closing (curly) braces { and }.

Unlike Python, indentation is not part of the syntax of JavaScript.

While it is not necessary to understand every aspect of how this code works it is nonetheless important to understand its overall function. This is because you will need to use this pattern each time you want to attach some functionality to a button.

- Every time you click on a webpage you trigger an *event* – called a click event.
- Events can be intercepted in a program using special functions called *handlers*.
- Handlers are special types of functions which you must write but don't have to call.
- Instead of writing a line of code to call a handler you must associate an event with a handler.

- In this example (see code snippet) we are associating the click event for the submit button with a handler called `saveContacts`. (Recall from the `html` file that 'submit-data' is the value of the button's `id` attribute – you can see this in `contact1.html`).
- Once you associate an event with a handler, the JavaScript sub-system looks after the rest. It does this by listening for events and calling the appropriate handler each time an event is triggered.
- The net result is that in this example the function `saveContacts` is called whenever the user clicks on the <Submit Data> button. The code in the function body is executed and the alert box is displayed.

Task 2E

In this task we will add the code to read the email address entered by the user into a JavaScript variable.

First, make sure you have comment out the `alert` statement in `contact1.js` using double slash (//)

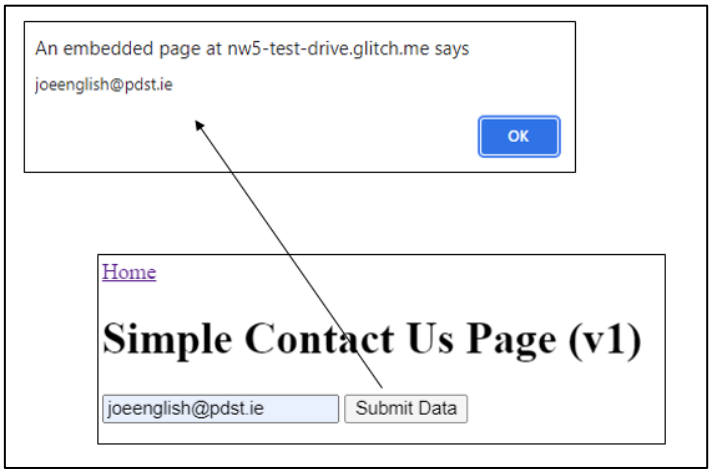
```
//alert("SUBMIT clicked!!!");
```

Next insert the following lines of code at the end of the function (anywhere before the closing brace, }).

```
// Task 2E - read the data from the email field
const emailField = document.getElementById("email");
const emailFieldValue = emailField.value;
alert(emailFieldValue);
```

At this stage it is not necessary to understand the minute detail of every line.
Do you notice that the value of the `id` attribute for the `input` field is called `email`? You should confirm this by looking back at the `contact1.html` file.

The behaviour of the form is illustrated below. When the user types in an email address e.g. joe.english@oide.ie and then clicks on <Submit Data> the alert box showing the data entered is displayed as illustrated below.



Well done! We now know how to read a field and display its contents.

The next step is to add the code snippet shown below. This code clears the contents of the input field and then give it the focus (i.e. place the cursor in the field). This makes the system more user-friendly.

```
// reset form
emailField.value = ''; // clear the field
emailField.focus(); // set the focus

// TASK 4D
```

Now that we have completed our front-end (the client-side piece) we are ready to create the back-end component which will be used to save the data to a database.

The database we will use is called the Firebase real time database.

TASK 3 – Setup Firebase

Follow the steps in Appendix 1 to create a Firebase project, database and register a web application.

Once this is done you will need to know how locate and copy the Firebase configuration details for your web application.

TASK 4 – Save Data to Firebase

In this task we will save the email address entered by the user on the Contact Us page to the Firebase database.

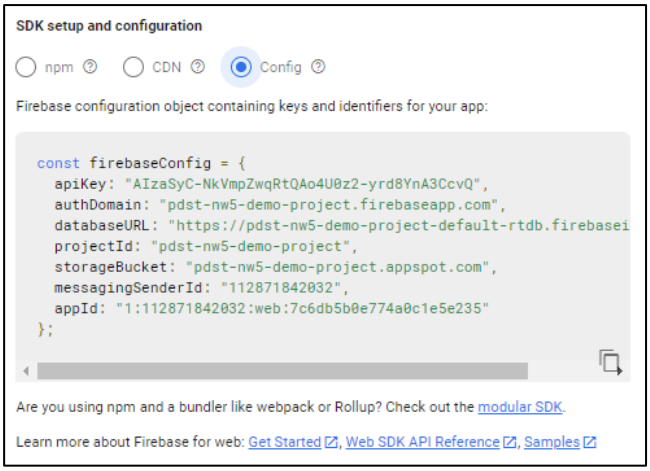
Task 4A - changes to `contact1.html`

Almost all the code we need in order to save the email address to Firebase will be added to `contact1.js`. But first we need to add one line – highlighted below in bold – at the end (bottom) of `contact1.html`. This line enables us to use the Firebase SDK in `contact1.js`. It tells the browser to load the Firebase library (`firebase.js`) before our own script (`contact1.js`) which will use them.

```
<!-- TASK 4A: Include firebase.js here -->
<script src="https://www.gstatic.com/firebasejs/8.4.0/firebase.js"></script>
```

TASK 4B – changes to `contact1.js`

Copy your web application's Firebase configuration details into `contact1.js`. You will find your configuration details under the Project settings on the Firebase Console.



The configuration details for my Firebase project are shown here. Yours will look like this but the values will be slightly different. *This will be the only difference between your code and all the other code you will use in this tutorial*.

Copy YOUR configuration details from Firebase and paste them into to the start of `contact1.js`

```
// TASK 4B – Copy your web app's Firebase configuration here ...
const firebaseConfig = {
  apiKey: "AIzaSyC-NkVmpZwqRtQAo4U0z2-yrd8YnA3CcvQ",
  authDomain: "pdst-nw5-demo-project.firebaseapp.com",
  databaseURL: "https://pdst-nw5-demo-project-default-rtdb.firebaseio.com",
  projectId: "pdst-nw5-demo-project",
  storageBucket: "pdst-nw5-demo-project.appspot.com",
  messagingSenderId: "112871842032",
  appId: "1:112871842032:web:7c6db5b0e774a0c1e5e235"
};

// TASK 4C
```

TASK 4C

Next you will add the following two lines of code exactly as they appear here. These lines should be added immediately after your configuration details.

```
// TASK 4C
// Initialize Firebase
firebase.initializeApp(firebaseConfig);

// Retrieve the database handle
const myDBCxn = firebase.database().ref('/contacts');
```

Notice that `contacts` is the name of the root node in the project data tree.

TASK 4D

Finally, you will need to modify the function `saveContacts` by adding in the lines that are highlighted in red and bold below (directly beneath where it says TASK 4D).

```
// Submit clicked so save the data on Firebase
function saveContacts() {
  //alert("SUBMIT clicked!!!");

  // read the data from the email field
  const emailField = document.getElementById("email");
  const emailFieldValue = emailField.value;
  alert(emailFieldValue)

  // reset form
  emailField.value = ''; // clear the field
  emailField.focus(); // set the focus

  // TASK 4D
  // code to save the data to Firebase GOES HERE!
  const data = myDBCxn.push();
  data.set( {email: emailFieldValue
            });

} // end saveContacts
```

Data is added to the database as a set of name-value pairs under the contacts node. The name-value pair for each new record is *email* and *emailFieldValue*. `email` corresponds to the field name.

TEST!

We are now ready to test our web application.

We will know the application is working if, after we enter a new email address it is saved on Firebase. We can use the Firebase console to check the database.

The screen shot confirms the database is empty.

We now use our web application to add a new contact e.g. joeenglish@pdst.ie

After clicking <Submit Data> we check the contents of the database again. The screenshot confirms that the entry has been added.

Notice that `contacts` is the name of the root node in the project data tree. The comes from the code in task 4D. `contacts` can be thought of as a database table i.e. the contacts table.

Data is added to the database as children of this node. Each child node is called a document and corresponds to a record. Every time the user clicks submit, a new child node will be created with the email address saved as part of a name-value pair.

Exercises

1. Try adding some more contacts yourself.
2. Share the URL of your application with others and monitor your Firebase as they start adding contacts.
3. Ask others to change the name of the root node in their code from `contacts` to something else e.g. `testData`. Now observe what happens when they start adding more contacts.

TASK 5 – Read Data from Firebase

In this task we will retrieve data from Firebase and display it on our own page.
In this way we will not have to use the Firebase console every time we want to see new data being added.

TASK 5A – changes to `contact1.html`

Add the following section immediately after the closing form element in `contact1.html`.

```
<!—TASK 5A: Paste the section code in here -->
<section>
  <!-- An area on the screen to display the contact details -->
  <textarea rows="20" cols="50" id="contacts"></textarea>
  <!--ol id="rows"></ol-->
</section>
```

This creates a placeholder on the screen where we will display the retrieved data. There are two place holders. The first is a *text area* and the second is an *ordered list*. In task 5B we will display the data in the text area and in task 5C we will display the data as an ordered list.

TASK 5B – changes to `contact1.js`

Open the file `contact1.js`.
Copy and paste the code show below after the function `saveContacts`.

This sets up a handler called `displayData` which will be called each time a new entry is added to the database. (A handler is a special kind of function called automatically by JavaScript in response to something happening.)

```
// Task 5B - Code to retrieve and display the data goes here ...
myDBCxn.on("child_added", displayData);

function displayData(data, prevChildKey) {
    const datapoint = data.val();
    document.getElementById("contacts").value += datapoint.email + "\n";
}
```

When you add a new contact, it should now be displayed in the text area on the screen that looks something like this.

TASK 5C

With the few simple changes outlined below we can display the contacts as list items in an ordered list. The desired result is illustrated in the screenshot shown here.

In `contact1.html` comment out the text area and uncomment the `ol` element as highlighted below.

```
      <section>
        <!-- An area on the screen to display the contact details -->
        <!--textarea rows="20" cols="50" id="contacts"></textarea-->
        <ol id="rows"></ol>
      </section>
```

In `contact1.js` insert the following block of code after the code for task 5B.

```
// Task 5C - Code to retrieve and display the data in a list goes here ...
myDBCxn.on("child_added", displayDataAsList);

// A handler to display the Firebase data as a list
function displayDataAsList(data, prevChildKey) {

    const datapoint = data.val();

    // create a new list item element and set its value to the email address
    const newListItem = document.createElement('li');
    newListItem.innerHTML = datapoint.email;

    // append the above list item to the ordered list identified by rows
    const rowList = document.getElementById('rows');
    rowList.appendChild(newListItem);

}
```

Final Challenges

1. Can you add a second field? What about a drop down list, checkbox or a group of radio buttons?

2. **Remix** https://glitch.com/edit/#!/nw5-fb-demo and use it to learn more about how to create Firebase web applications using JavaScript.

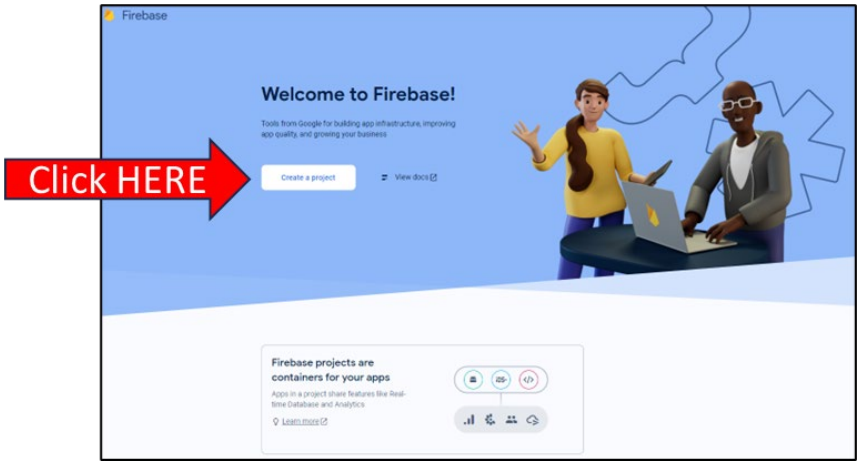
Appendix 1 – Creating a real-time Firebase and link it to a Web Application

Firebase is a platform developed by Google for creating mobile and web applications. It was originally an independent company founded in 2011. In 2014, Google acquired the platform and it is now their flagship offering for app development.

STEP 1 Create a firebase project

Browse to https://console.firebase.google.com/u/0/

You will see a screen that looks like this..



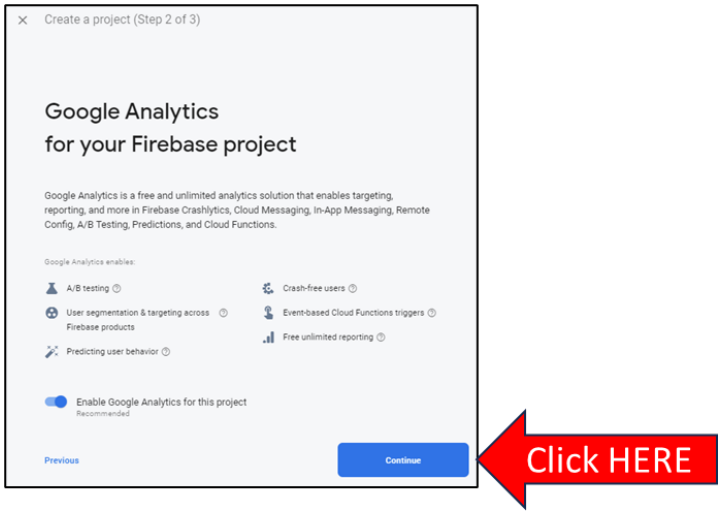
Click on **Create Project** and the screen shown below appears ....

Enter a project name e.g. *My-Demo-Project* and then click on the **Continue** button as shown.

The project name is an internal only name i.e. not exposed in any publicly visible Firebase or Google Cloud product, service, or resource; it simply serves to help you more easily distinguish among multiple projects.

The project name can only contain letters, numbers, spaces, and these characters: - ! ' "

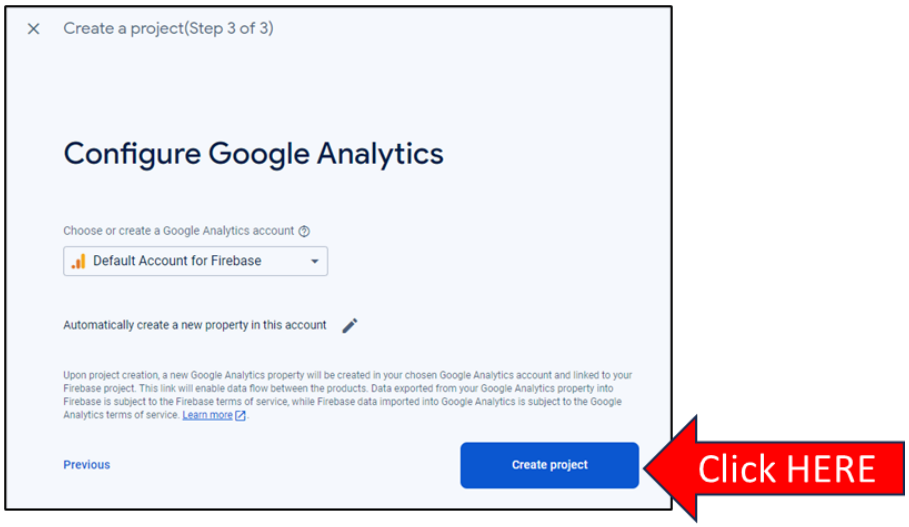
The next screen looks like this ...



You may get either of the of the following UIs. Please follow the steps for the UI you see.

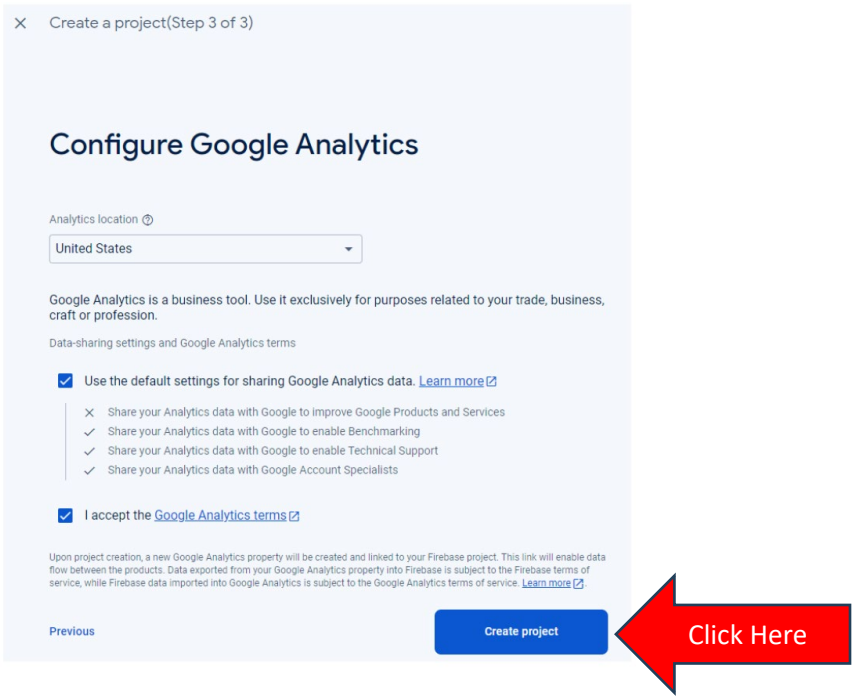
UI 1

Click **Continue** and you will see a screen that looks like this ...

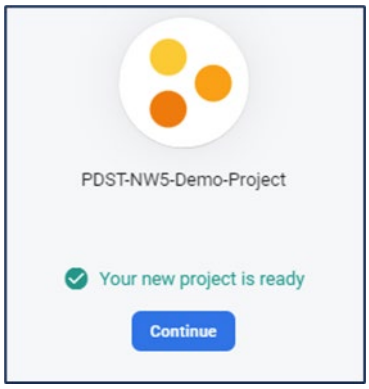


OR

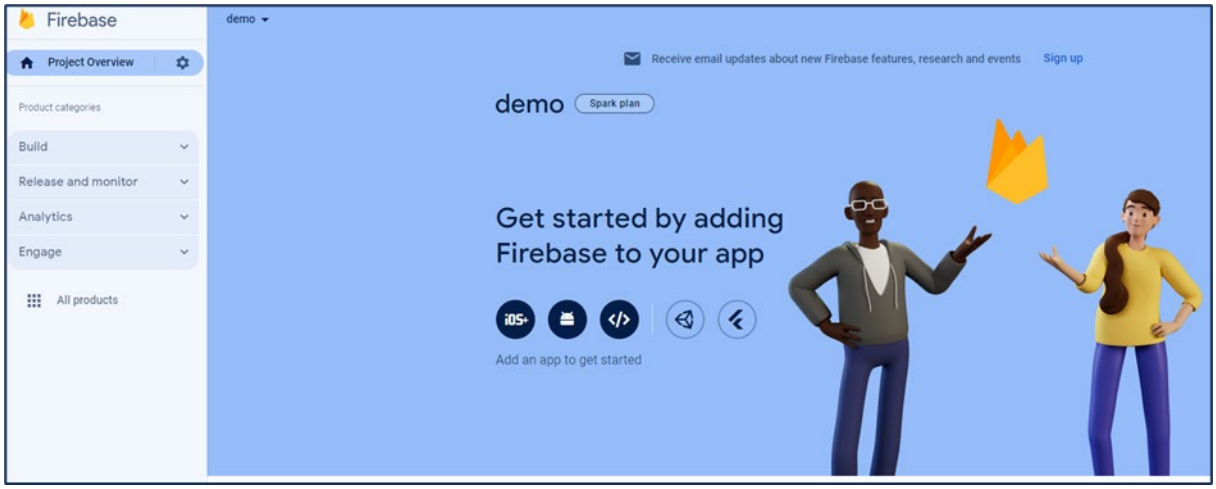
UI 2



Click **Create project** as shown and after a short wait you will see a screen that looks like this.



Click **Continue** and you will see this ....



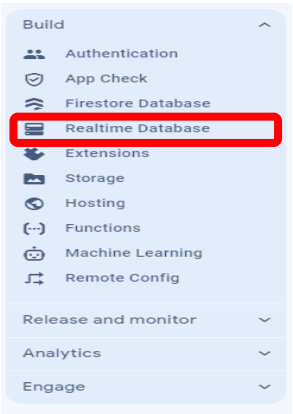
Congratulations!

You have just created a Firebase project.

STEP 2 Create the database

The next step is to create a Realtime Database.

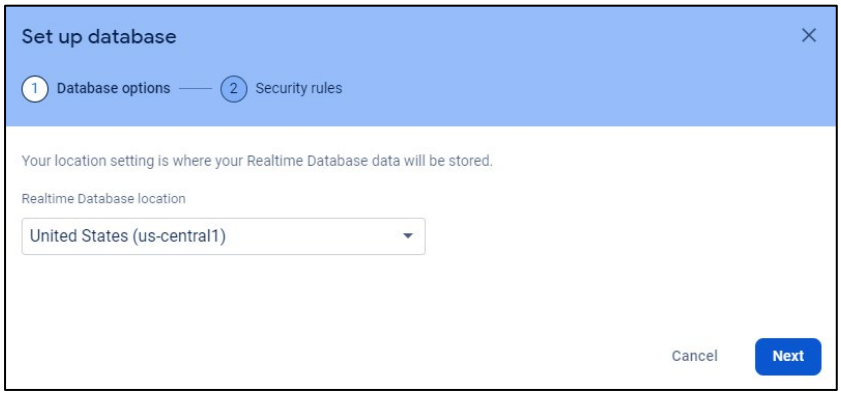
Click on the **Realtime Database** option from the **Build** menu on the left hand side of the console.



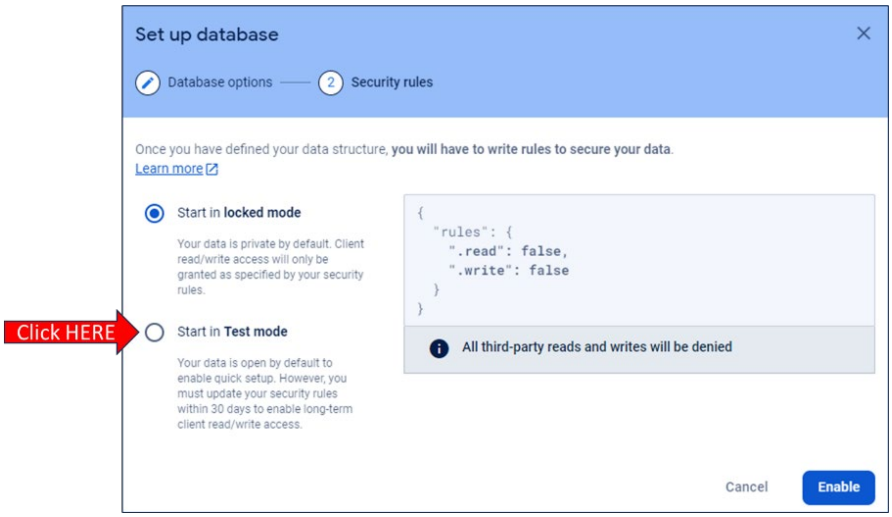
You should now have the option to create the database as shown below ....

Click on the **Create Database** button.

You should now see a screen that looks like this ....

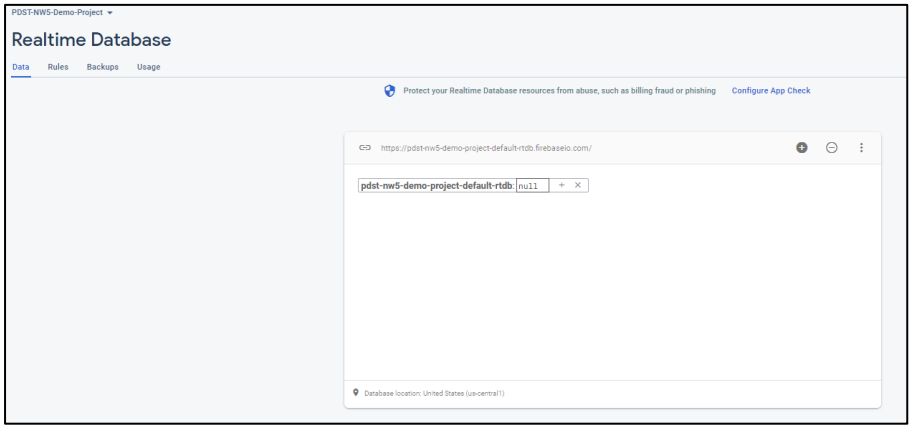


Click **Next**



Click HERE

Change from 'Start in locked mode' to '**Start in test mode'** and then click the **Enable** button. You should see a screen that looks like this.



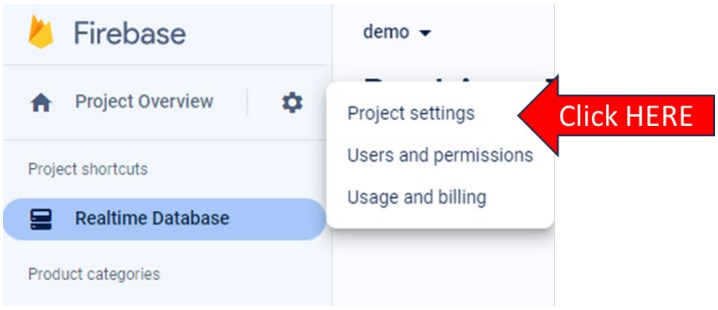
Congratulations!

You have now created a Realtime database hosted on the Firebase platform.

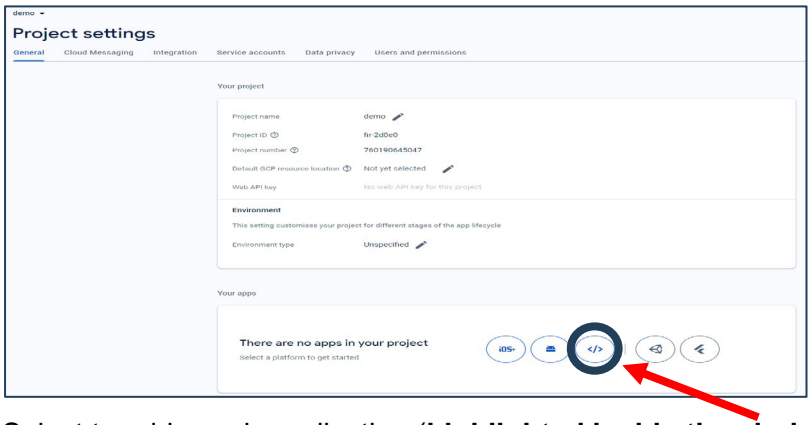
The next step is to register our web application so that it can 'talk' to the database

STEP 3 Register the web application

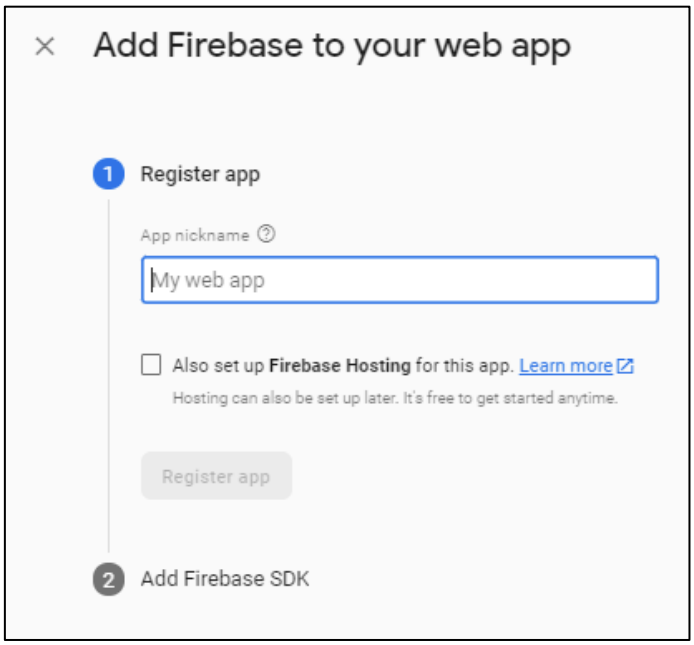
Click on the gear icon (beside Project Overview) and from the pop out menu click on **Project settings**



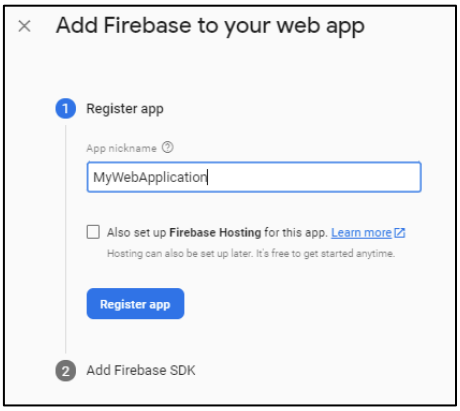
You will now see a screen that looks like this ...



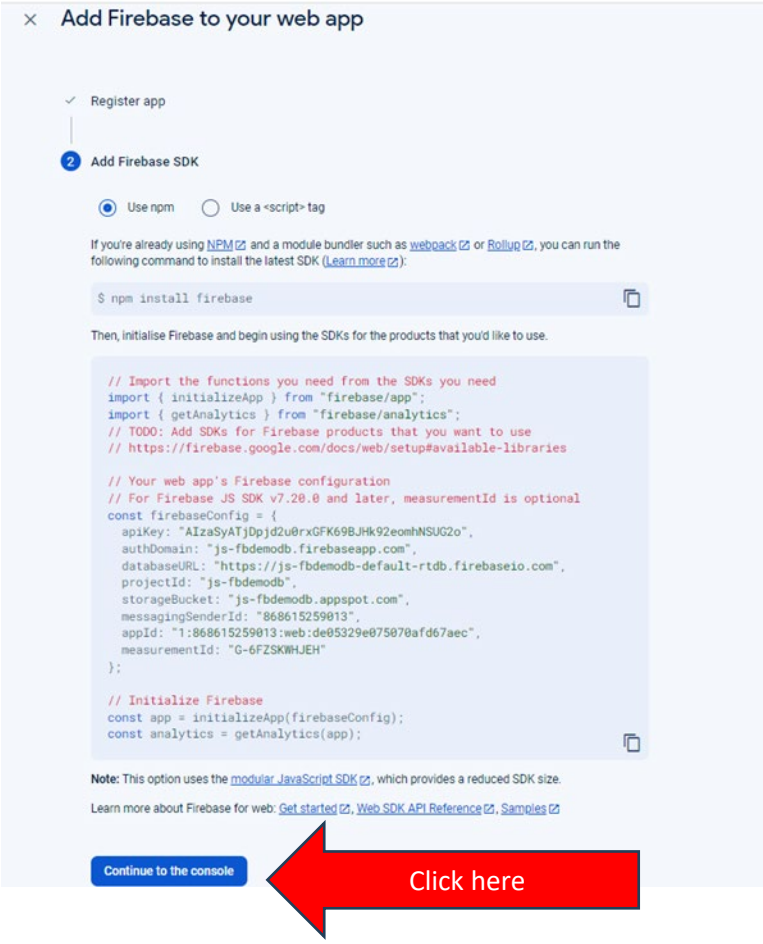
Select to add a web application (**highlighted inside the circle**)....



Type in a nickname for your web application e.g. *MyWebApplication* as shown ....

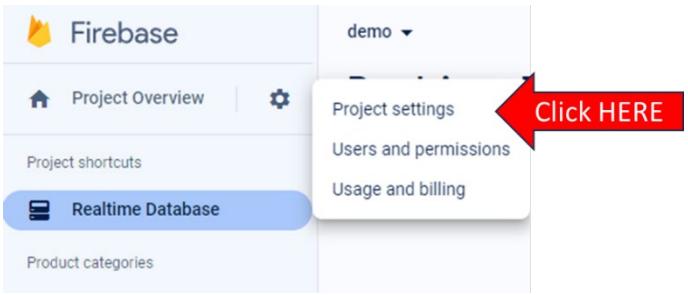


Click on the **Register app** button and then click **Continue to the console.**



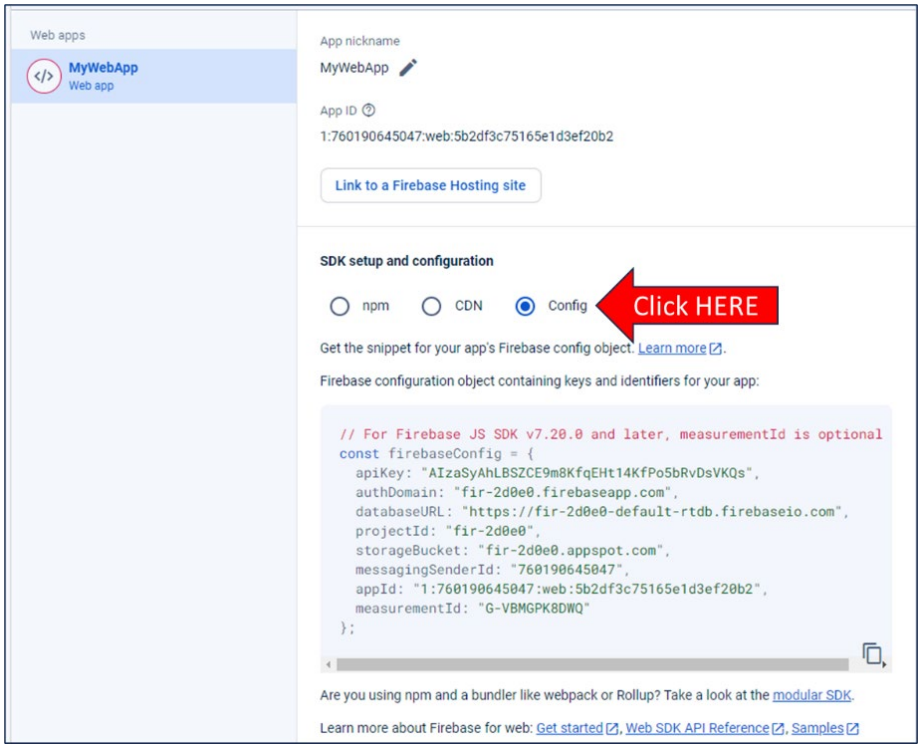
STEP 4 Find your app's config details

Click on the gear icon (beside Project Overview) and from the pop out menu click on **Project settings**



Scroll down and select the **Config** radio button.

You will need to copy these details into your JavaScript program.



You are now ready to return to task 4 of the main tutorial (page 9).

Appendix 2 – References

JavaScript Manual for LCCS teachers:

https://drive.google.com/file/d/1KAogeP681Swo2EyVXoCx84gak00Yyd8u/view?usp=sharing

PDST GitHub repository for JavaScript:

https://github.com/pdst-lccs/lccs-javascript

Glitch (Development Environment)
https://glitch.com/

Replit (Development Environment)
https://replit.com/

Mozilla (HTML)	https://developer.mozilla.org/en-US/docs/Web/HTML/Element
Mozilla (CSS)	https://developer.mozilla.org/en-US/docs/Web/CSS
Mozilla (JavaScript)	https://developer.mozilla.org/en-US/docs/Web/JavaScript https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
W3Schools	https://www.w3schools.com/tags/default.asp
W3Schools	https://www.w3schools.com/js/default.asp
Tutorials Point	https://www.tutorialspoint.com/html/html_tags_reference.htm
TutorialsPoint	https://www.tutorialspoint.com/javascript/
JavaScript.info	https://javascript.info/js
Oracle	https://developer.oracle.com/javascript
geeksforgeeks	https://www.youtube.com/playlist?list=PLo3w8EB99pqLEopnunz-dOOBJ8t-Wgt2g
Web Demystified	https://www.youtube.com/playlist?list=PLo3w8EB99pqLEopnunz-dOOBJ8t-Wgt2g