Student name:

Student ID:

SIT225: Data Capture Technologies

# Activity 5.1: Firebase Realtime database

The Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data between your users in real-time. Data is stored as JSON and synchronized in real-time to every connected client. In this activity, you will set up and perform operations such as queries and updates on the database using Python programming language.

## Hardware Required

* No hardware is required.

## Software Required

Firebase Realtime database

Python 3

## Steps

|  |  |
| --- | --- |
| **Step** | **Action** |
| 1 | **Create an Account**:  First, you will need to create an account in the Firebase console, follow instructions in the official Firebase document (<https://firebase.google.com/docs/database/rest/start> ). |
| 2 | **Create a Database**:  Follow the above Firebase document to create a database. When you click on Create Database, you have to specify the location of the database and the security rules. Two rules are available – locked mode and test mode; since we will be using the database for reading, writing, and editing, we choose test mode. |
| 3 | **Setup Python library for Firebase access**:  We will be using Admin Database API, which is available in *firebase\_admin* library. Use the below command in the command line to install. You can follow a Firebase tutorial here (https://www.freecodecamp.org/news/how-to-get-started-with-firebase-using-python ).  $ pip install firebase\_admin  Firebase will allow access to Firebase server APIs from Google Service Accounts. To authenticate the Service Account, we require a private key in JSON format. To generate the key, go to project settings, click Generate new private key, download the file, and place it in your current folder where you will create your Python script. |
| 4 | **Connect to Firebase using Python version of Admin Database API**:  A credential object needs to be created to initialise the Python library which can be done using the Python code below. Python notebook can be downloaded here (<https://github.com/deakin-deep-dreamer/sit225/blob/main/week_5/firebase_explore.ipynb> ).    The databaseURL is a web address to reach your Firebase database that you have created in step 2. This URL can be found in the Data tab of Realtime Database.    If you compile the code snippet above, it should do with no error. |
| 5 | **Write to database Using the set() Function**:  We set the reference to the root of the database (or we could also set it to a key value or child key value). Data needs to be in JSON format as below.    A reference point always needed to be set where the data read/write will take place. In the code above, the reference point is set at the root of the NoSQL Document, where consider the database is a JSON tree and / is the root node of the tree). The set() function writes (overwrites) data at the set reference point.  You can visualise the data in the Firebase console as below - |
| 6 | **Read data using get() function**:  Data can be read using get() function on the reference set beforehand, as shown below.    Consider the reference set in line 1 and the output compared to the reference set at line 14 and the bottom output line to understand the use of db.reference() and ref.get(). |
| 7 | **Write to database Using the push() Function**:  The push() function saves data under a *unique system generated key*. This is different than set() where you set the keys such as Book1, Book2, Book3 and Book4 under which the content (author, genre, price and title) appears. Let’s try to push the same data in the root reference. Note that since we already has data under root / symbol, setting (or pushing) in the same reference point will eventually rewrite the original data.    The output will reset the previous data set in / node. The current data is shown below.    As you can see, under /Books/Best\_Sellers there are 4 nodes where the node head (or node ID) is a randomly generated key which is due to the use of push() function. When data key does not matter, the use of push() function desirable. |
| 8 | **Update data**:  Let’s say the price of the books by J. R. R. Tolkien is reduced to 80 units to offer a discount. The first 3 books are written by this author, and we want to apply for a discount on all of them.    As you can see, the author name is compared and the new price is set in the best\_sellers dictionary and finally, an update() function is called on the ref, however, the current ref is a ‘/Books/Best\_Sellers/’, so we need to locate the child under the ref node, so ref.child(key) is used in line 13. The output is shown below with a discounted price. |
| 9 | **Delete data**:  Let's delete all bestseller books with J.R.R. Tolkien as the author. You can locate the node using db.reference() (line 4) and then locate specific record (for loop in line 6) and calling set() with empty data {} as a parameter, such as set({}). The particular child under the ref needs to be located first by using ref.child(key), otherwise, the ref node will be removed – BE CAREFUL.    This keeps only the other author data, as shown below.    If ref.child() not used, as shown the code below, all data will be removed.    Now in Firebase console you will see no data exists. |
| 10 | Question: Run all the cells in the Notebook you have downloaded in Step 4, fill in the student information at the top cell of the Notebook. Convert the Notebook to PDF and merge with this activity sheet PDF.  Answer: Convert the Notebook to PDF and merge with this activity sheet PDF. |
| 10 | Question: Create a sensor data structure for DHT22 sensor which contains attributes such as sensor\_name, timestamp, temperature and humidity. Remember there will be other sensors with different sensor variables such as DHT22 has 2 variables, accelerometer sensor has 3. For each such sensor, you will need to gather data over time. Discuss how you are going to handle multiple data values in JSON format? Justify your design.  Answer: **I will organize the data so that every sensor is saved under sensors/{sensor\_id}, including a metadata section (sensor\_name, type, location, units) and a readings section that contains all recorded values.**  **Each reading will feature a shared timestamp (Unix ms) along with only the variables pertinent to that sensor — for instance, temperature and humidity for DHT22, or x, y, and z values for an accelerometer.**  **To facilitate continuous logging, new entries will be appended under readings using Firebase push IDs, which prevents key collisions and naturally arranges the data chronologically.**  **This structure reduces redundant metadata, accommodates various sensor types, simplifies time-based queries (latest values or by specific range), and guarantees efficient storage and retrieval in Firebase.** |
| 11 | Question: Generate some random data for DHT22 sensor, insert data to database, query all data and screenshot the output here.  Answer: |
| 12 | Question: Generate some random data for the SR04 Ultrasonic sensor, insert data to database, query all data and screenshot the output here.  Answer: |
| 13 | Question: Firebase Realtime database generates events on data operations. You can refer to section ‘Handling Realtime Database events’ in the document (<https://firebase.google.com/docs/functions/database-events?gen=2nd> ). Discuss in the active learning session and summarise the idea of database events and how it is handled using Python SDK.  Note that these events are useful when your sensors (from Arduino script) store data directly to Firebase Realtime database and you would like to track data update actions from a central Python application such as a monitoring dashboard.  Answer: The Firebase Realtime Database activates events whenever data is created, updated, deleted, or written. By utilizing Cloud Functions (Gen 2), you can link triggers like onValueCreated, onValueUpdated, onValueDeleted, and onValueWritten to a specific path (for instance, /sensors/{sensorId}/readings/{pushId}), and the function will run with each modification. The handler will obtain a snapshot for create/delete events or before/after snapshots for write/update events, enabling you to validate, compute aggregates, or send alerts without altering client code. You can use path wildcards (such as {sensorId}) to handle multiple sensors, and it's wise to deploy the function in the same region as the database to reduce latency.  In Python, the Admin SDK allows secure read/write operations but does not support realtime listeners. A monitoring application can either (1) use REST streaming (SSE) to follow a path in near-realtime, or (2) periodically check for recent readings. A suggested method is to let Cloud Functions normalize updates (for example, writing to /latest/{sensorId} or /events) and have the Python dashboard keep an eye on that single path through SSE or light polling. |

# Activity 5.2: Data wrangling

Data wrangling is the process of converting raw data into a usable form. The process includes collecting, processing, analyzing, and tidying the raw data so that it can be easily read and analyzed. In this activity, you will use the common library in python, "pandas".

## Hardware Required

* No hardware is required.

## Software Required

Python 3

Pandas Python library

## Steps

|  |  |
| --- | --- |
| **Step** | **Action** |
| 1 | Install Pandas using the command below. Most likely you already have Pandas installed if you have installed Python using Anaconda disribution (https://www.anaconda.com/download).  $ pip install pandas  A Python notebook is shared in the GitHub link (<https://github.com/deakin-deep-dreamer/sit225/tree/main/week_5> ). There will be a data\_wrangling.ipynb, shopping\_data.csv and shopping\_data\_missingvalue.csv files among others. Download the week\_5 folder in your computer, open a command prompt in that folder, and write the command below in the command line:  $ jupyter lab  This will open Python Jupyter Notebook where in the left panel you can see the files (labeled as 1 in figure).    Each cell contains Python code (labeled as 2 in figure), you can run a cell by clicking on the cell, so the cursor appears in that cell and then click on the play button at the top of the panel (labeled as 3 in the figure). |
| 2 | Question: Run each cell to produce output. Follow instructions in the notebook to complete codes in some of the cells. Convert the notebook to PDF from menu File > Save and Export Notebook As > PDF. Convert this activity sheet to PDF and merge with the notebook PDF.  Answer: There is no answer to write here. You have to answer in the Jupyter Notebook. |
| 3 | Question: Once you went through the cells in the Notebook, you now have a basic understanding of data wrangling. Pandas are a powerful tool and can be used for reading CSV data. Can you use Pandas in reading sensor CSV data that you generated earlier? Describe if any modification you think necessary?  Answer: Yes, Pandas can easily handle sensor CSV files with read\_csv(). The main tweaks I would make are cleaning and preparing the data: converting timestamps into proper datetime format and using them as the index, standardizing column names and units (e.g., temperature, humidity, distance\_cm), handling missing values with fill or interpolation, dropping duplicates, and filtering unrealistic readings such as humidity over 100% or negative distance values. |
| 4 | Question: What do you understand of the Notebook section called Handling Missing Value? Discuss in group and briefly summarise different missing value imputation methods and their applicability on different data conditions.  Answer: I checked out the "Handling Missing Value" section and took it to mean "first spot the NaNs, then pick a fill that fits the data." If there are just a few missing rows and they seem random, I'd just get rid of them; for numeric columns, I'd use the mean or median (going with the median if the data is skewed), and for categorical data, I'd go for the mode.  When it comes to sensor time-series data, I'd use forward or backward filling or time-based interpolation for short gaps. Plus, if the features are linked, a simple model like KNN could work well. For example, with DHT22 temperature, I'd fill in a short gap with interpolation, for a skewed variable like Income, I'd pick the median, and for Gender, I'd choose the mode—then I'd make sure to note down the changes for clarity. |