Importing Data to a Firestore Database

Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

1. Click **Activate Cloud Shell** at the top of the Google Cloud console.

2. Click through the following windows:

    - Continue through the Cloud Shell information window.
    - Authorize Cloud Shell to use your credentials to make Google Cloud API calls.

When you are connected, you are already authenticated, and the project is set to your **Project_ID**, `qwiklabs-gcp-04-76475ca51f43`. The output contains a line that declares the **Project_ID** for this session:

```
Your Cloud Platform project in this session is set to qwiklabs-gcp-04-
76475ca51f43
```

`gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

3. (Optional) You can list the active account name with this command:

```
gcloud auth list
```

4. Click **Authorize**.

Output:

```
ACTIVE: *
ACCOUNT: student-01-a02127997a10@qwiklabs.net

To set the active account, run:
    $ gcloud config set account `ACCOUNT`
```

5. (Optional) You can list the project ID with this command:

```
gcloud config list project
```

Output:

```
[core]
project = qwiklabs-gcp-04-76475ca51f43
```

Note: For full documentation of `gcloud`, in Google Cloud, refer to the gcloud CLI overview guide.

Task 1. Set up Firestore in Google Cloud

1. On the Cloud Console Navigation menu (≡), click **View All Products** and under **Databases** select **Firestore**.

2. Click **Create a Firestore database**.

3. Select **Standard Edition**.

4. Under Configuration options, select **Firestore Native**.

5. For Security rules, choose **Open**.

6. In Location type, click **Region**, and then select the lab region `us-east1` from the list.

   Note: If the list of regions is not populated, refresh the browser or try the wizard again from the Cloud console menu.

Note: Both modes are high performing with strong consistency, but they look different and are optimized for different use cases.

7. Leave the other settings as their defaults, and click **Create Database**.

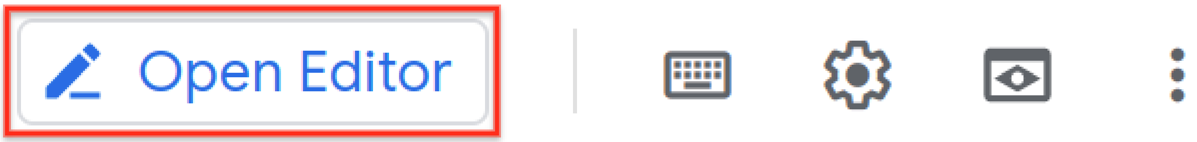
Task 2. Write database import code

As Patrick said, the customer data will be available in a CSV file. Help Patrick create an app that reads customer records from a CSV file and writes them to Firestore. Since Patrick is familiar with Javascript, build this application with the Node.js JavaScript runtime.

1. In Cloud Shell, run the following command to clone the Pet Theory repository:

```
git clone https://github.com/rosera/pet-theory
```

2. Use the Cloud Shell Code Editor (or your preferred editor) to edit your files. From the top ribbon of your Cloud Shell session, click **Open Editor**, it will open a new tab. If prompted, click **Open in a new window** to launch the code editor:



3. Then change your current working directory to `lab01`:

```
cd pet-theory/lab01
```

In the directory you can see Patrick's `package.json`. This file lists the packages that your Node.js project depends on and makes your build reproducible, and therefore easier to share with others.

An example `package.json` is shown below:

```
{
        "name": "lab01",
        "version": "1.0.0",
        "description": "This is lab01 of the Pet Theory labs",
        "main": "index.js",
        "scripts": {
                "test": "echo \"Error: no test specified\" && exit 1"
        },
        "keywords": [],
        "author": "Patrick - IT",
        "license": "MIT",
        "dependencies": {
                "csv-parse": "^5.5.3"
        }
}
```

To allow Patrick's code to write to the Firestore database, you need to install some additional peer dependencies.

4. Run the following command to do so:

```
npm install @google-cloud/firestore
```

5. To enable the app to write logs to Cloud Logging, install an additional module:

```
npm install @google-cloud/logging
```

After successful completion of the command, the `package.json` will be automatically updated to include the new peer dependencies, and will look like this.

```
...

"dependencies": {
  "@google-cloud/firestore": "^7.3.0",
  "@google-cloud/logging": "^11.0.0",
  "csv-parse": "^5.5.3"
}
```

Now it's time to take a look at the script that reads the CSV file of customers and writes one record in Firestore for each line in the CSV file. Patrick's original application is shown below:

```
const csv = require('csv-parse');
const fs  = require('fs');

function writeToDatabase(records) {
  records.forEach((record, i) => {
    console.log(`ID: ${record.id} Email: ${record.email} Name:
${record.name} Phone: ${record.phone}`);
  });
  return ;
}

async function importCsv(csvFilename) {
  const parser = csv.parse({ columns: true, delimiter: ',' }, async
function (err, records) {
    if (e) {
      console.error('Error parsing CSV:', e);
      return;
    }
    try {
      console.log(`Call write to Firestore`);
      await writeToDatabase(records);
      console.log(`Wrote ${records.length} records`);
    } catch (e) {
      console.error(e);
      process.exit(1);
    }
  });

  await fs.createReadStream(csvFilename).pipe(parser);
}

if (process.argv.length < 3) {
  console.error('Please include a path to a csv file');
  process.exit(1);
}

importCsv(process.argv[2]).catch(e => console.error(e));
```

It takes the output from the input CSV file and imports it into the legacy database. Next, update this code to write to Firestore.

6. Open the file `pet-theory/lab01/importTestData.js`.

To reference the Firestore API via the application, you need to add the peer dependency to the existing codebase.

7. Add the following Firestore dependency on line 3 of the file:

```
const { Firestore } = require("@google-cloud/firestore");
```

Ensure that the top of the file looks like this:

```
const csv = require('csv-parse');
const fs  = require('fs');
const { Firestore } = require("@google-cloud/firestore"); // Add this
```

Integrating with the Firestore database can be achieved with a couple of lines of code. Ruby has shared some template code with you and Patrick for exactly that purpose.

8. Add the following code underneath line 34, or after the `if (process.argv.length < 3)` conditional:

```
async function writeToFirestore(records) {
  const db = new Firestore({
    // projectId: projectId
  });
  const batch = db.batch()

  records.forEach((record)=>{
    console.log(`Write: ${record}`)
    const docRef = db.collection("customers").doc(record.email);
    batch.set(docRef, record, { merge: true })
  })

  batch.commit()
    .then(() => {
       console.log('Batch executed')
    })
    .catch(err => {
       console.log(`Batch error: ${err}`)
    })
  return
}
```

The above code snippet declares a new database object, which references the database created earlier in the lab. The function uses a batch process in which each record is processed in turn and given a document reference based on the identifier added. At the end of the function, the batch content is committed (written) to the database.

9. Update the `importCsv` function to add the function call to **writeToFirestore** and remove the call to **writeToDatabase**. It should look like this:

```
async function importCsv(csvFilename) {
  const parser = csv.parse({ columns: true, delimiter: ',' }, async
function (err, records) {
    if (err) {
      console.error('Error parsing CSV:', err);
      return;
    }
    try {
      console.log(`Call write to Firestore`);
      await writeToFirestore(records);
      // await writeToDatabase(records);
      console.log(`Wrote ${records.length} records`);
    } catch (e) {
      console.error(e);
      process.exit(1);
    }
  });

  await fs.createReadStream(csvFilename).pipe(parser);
}
```

10. Add logging for the application. To reference the Logging API via the application, add the peer dependency to the existing codebase. Add the following line just below the other require statements at the top of the file:

```
const { Logging } = require('@google-cloud/logging');
```

Ensure that the top of the file looks like this:

```
const csv = require('csv-parse');
const fs  = require('fs');
const { Firestore } = require("@google-cloud/firestore");
const { Logging } = require('@google-cloud/logging');
```

11. Add a few constant variables and initialize the Logging client. Add those just below the above lines in the file (~line 5), like this:

```
const logName = "pet-theory-logs-importTestData";

// Creates a Logging client
const logging = new Logging();
const log = logging.log(logName);

const resource = {
  type: "global",
};
```

12. Add code to write the logs in `importCsv` function just below the line "console.log(`Wrote ${records.length} records`);" which should look like this:

```
// A text log entry
success_message = `Success: importTestData - Wrote ${records.length}
records`;
const entry = log.entry(
       { resource: resource },
       { message: `${success_message}` }
);
```

```
log.write([entry]);
```

After these updates, your `importCsv` function code block should look like the following:

```
async function importCsv(csvFilename) {
  const parser = csv.parse({ columns: true, delimiter: ',' }, async function (err, records) {
    if (err) {
      console.error('Error parsing CSV:', err);
      return;
    }
    try {
      console.log(`Call write to Firestore`);
      await writeToFirestore(records);
      // await writeToDatabase(records);
      console.log(`Wrote ${records.length} records`);
      // A text log entry
      success_message = `Success: importTestData - Wrote ${records.length} records`;
      const entry = log.entry(
             { resource: resource },
             { message: `${success_message}` }
      );
      log.write([entry]);
    } catch (e) {
      console.error(e);
      process.exit(1);
    }
  });

  await fs.createReadStream(csvFilename).pipe(parser);
}
```

Now when the application code is running, the Firestore database will be updated with the contents of the CSV file. The function `importCsv` takes a filename and parses the content on a line by line basis. Each line processed is now sent to the Firestore function `writeToFirestore`, where each new record is written to the "customer" database.

Note: In a production environment, you will write your own version of the import script.

Task 3. Create test data

1. First, install the "faker" library, which will be used by the script that generates the fake customer data. Run the following command to update the dependency in `package.json`:

```
npm install faker@5.5.3
```

2. Now open the file named **createTestData.js** with the code editor and inspect the code. Ensure it looks like the following:

```
const fs = require('fs');
const faker = require('faker');

function getRandomCustomerEmail(firstName, lastName) {
  const provider = faker.internet.domainName();
  const email = faker.internet.email(firstName, lastName, provider);
  return email.toLowerCase();
}

async function createTestData(recordCount) {
  const fileName = `customers_${recordCount}.csv`;
  var f = fs.createWriteStream(fileName);
  f.write('id,name,email,phone\n')
  for (let i=0; i<recordCount; i++) {
    const id = faker.datatype.number();
    const firstName = faker.name.firstName();
    const lastName = faker.name.lastName();
    const name = `${firstName} ${lastName}`;
    const email = getRandomCustomerEmail(firstName, lastName);
    const phone = faker.phone.phoneNumber();
    f.write(`${id},${name},${email},${phone}\n`);
  }
  console.log(`Created file ${fileName} containing ${recordCount}
records.`);
}

recordCount = parseInt(process.argv[2]);
if (process.argv.length != 3 || recordCount < 1 || isNaN(recordCount))
{
  console.error('Include the number of test data records to create.
Example:');
  console.error('    node createTestData.js 100');
  process.exit(1);
}

createTestData(recordCount);
```

3. Add Logging for the codebase. On line 3, add the following reference for the Logging API module from the application code:

```
const { Logging } = require("@google-cloud/logging");
```

The top of the file should now look like this:

```
const fs = require("fs");
const faker = require("faker");
const { Logging } = require("@google-cloud/logging"); //add this
```

4. Now, add a few constant variables and initialize the Logging client. Add those just below the `const` statements:

```
const logName = "pet-theory-logs-createTestData";
```

```
// Creates a Logging client
const logging = new Logging();
const log = logging.log(logName);

const resource = {
       // This example targets the "global" resource for simplicity
       type: "global",
};
```

5. Add code to write the logs in the **createTestData** function just below the line "console.log(`Created file ${fileName} containing ${recordCount} records.`);" which will look like this:

```
// A text log entry
const success_message = `Success: createTestData - Created file
${fileName} containing ${recordCount} records.`;
const entry = log.entry(
       { resource: resource },
       {
              name: `${fileName}`,
              recordCount: `${recordCount}`,
              message: `${success_message}`,
       }
);
log.write([entry]);
```

6. After updating, the `createTestData` function code block should look like this:

```
async function createTestData(recordCount) {
  const fileName = `customers_${recordCount}.csv`;
  var f = fs.createWriteStream(fileName);
  f.write('id,name,email,phone\n')
  for (let i=0; i<recordCount; i++) {
    const id = faker.datatype.number();
    const firstName = faker.name.firstName();
    const lastName = faker.name.lastName();
    const name = `${firstName} ${lastName}`;
    const email = getRandomCustomerEmail(firstName, lastName);
    const phone = faker.phone.phoneNumber();
    f.write(`${id},${name},${email},${phone}\n`);
  }
  console.log(`Created file ${fileName} containing ${recordCount}
records.`);
  // A text log entry
  const success_message = `Success: createTestData - Created file
${fileName} containing ${recordCount} records.`;
  const entry = log.entry(
         { resource: resource },
         {
              name: `${fileName}`,
              recordCount: `${recordCount}`,
              message: `${success_message}`,
         }
  );
  log.write([entry]);
}
```

7. Run the following command in Cloud Shell to create the file `customers_1000.csv`, which will contain 1000 records of test data:

```
node createTestData 1000
```

You should receive a similar output:

```
Created file customers_1000.csv containing 1000 records.
```

8. Open the file `customers_1000.csv` and verify that the test data has been created.

Task 4. Import the test customer data

1. To test the import capability, use both the import script and the test data created earlier:

```
node importTestData customers_1000.csv
```

You should receive a similar output:

```
Writing record 500
Writing record 1000
Wrote 1000 records
```

2. If you get an error that resembles the following:

```
Error: Cannot find module 'csv-parse'
```

Run the following command to add the `csv-parse` package to your environment:

```
npm install csv-parse
```

3. Then run the command again. You should receive the following output:

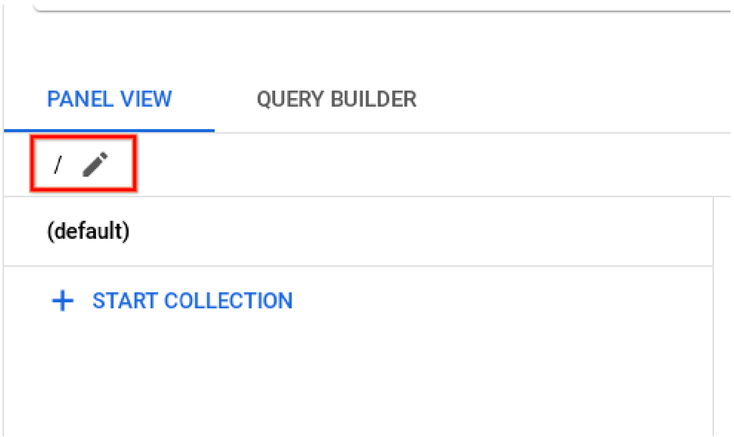
```
Writing record 500
Writing record 1000
Wrote 1000 records
```

Over the past couple of sections you have seen how Patrick and Ruby have created test data and a script to import data into Firestore. Patrick now feels more confident about loading customer data into the Firestore database.

Task 5. Inspect the data in Firestore

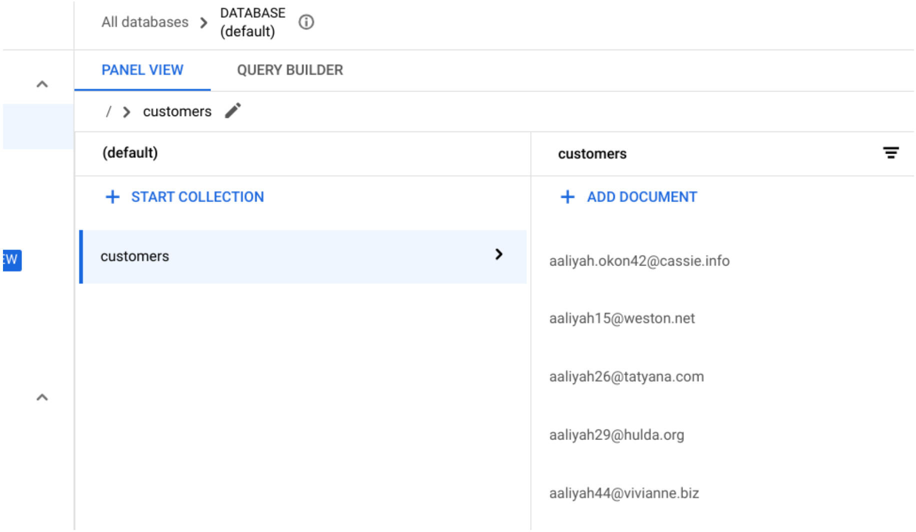
With a little help from you and Ruby, Patrick has now successfully migrated the test data to the Firestore database. Open up Firestore and see the results!

1. Return to your Cloud Console tab. In the **Navigation menu** (≡), click **View All Products** and under **Databases** select **Firestore** then click on **default** database, Once there, click on the pencil icon.



2. Type in `/customers` and press **Enter**.

3. Refresh your browser tab and you should see the following list of customers successfully migrated:



aaliyah.okon42@cassie.info

aaliyah15@weston.net

aaliyah26@tatyana.com

aaliyah29@hulda.org

aaliyah44@vivianne.biz