Activity 2.2.3: Build a web-site visit tracker and host in in Google App Engine

Table of Contents

Overview

1. **Create your Cloud project**

   Learn how to create a Cloud project and then set up the App Engine resources for your web service.

2. **Write your web service**

   Learn how to write and locally test a basic web service, and then define the configuration files that you need for deploying that web service to App Engine.

3. **Test your web service**

   Learn how to test the web-service you have developed.

4. **Configure your web service in App Engine**

   Learn how to configure your web service in App Engine.

5. **Deploy your web service to App Engine**

   Learn how to deploy your Python 3 code and then view your web service running on App Engine.

6. **Handle Data with Datastore**

   Learn how to use Datastore to store and retrieve data about site requests.

7. **Add Firebase to your web service**

   Learn how to add Firebase to your Cloud project and web service.

8. **Authenticate users with Firebase**

   Learn how to use Firebase Authentication to verify user credentials, serve user information, and allow data access.

9. **Personalize data for authenticated users**

   Learn how to use authentication to personalize data storage and retrieval for authenticated users.

10. **Clean up resources**

Learn how to clean up your project and avoid the possibility of incurring charges for resources you aren't using.

1. Create your cloud project

To create a Cloud project and your App Engine resources using the Google Cloud SDK,

1. **Create a new project** by running.

```
gcloud projects create YOUR-PROJECT-ID.
```

2. Set the above created project as the default project by running the command below.

```
gcloud config set project YOUR-PROJECT-ID
```

3. Run the following `gcloud` command to enable App Engine and create the associated application resources. Note that the location you select cannot be changed later.

```
gcloud app create
```

4. Make sure that billing is enabled for your project.

Learn how to enable billing

Note: Running only the sample app in this topic does not exceed the free quotas. You will be charged only if you exceed those quotas, for example, by running other samples and adding other services to the same GCP project.

2. Write your web service

The initial iteration of your web service uses Flask to serve a Jinja-based HTML template.

To set up your web service:

1. Create your `templates/index.html` file:

appengine/standard_python3/building-an-app/building-an-app-1/templates/index.html

View on GitHub

```
<!doctype html>
<html>
<head>
  <title>Datastore and Firebase Auth Example</title>
  <script src="{{ url_for('static', filename='script.js')
}}"></script>
  <link type="text/css" rel="stylesheet" href="{{
url_for('static', filename='style.css') }}">
</head>
<body>

  <h1>Datastore and Firebase Auth Example</h1>

  <h2>Last 10 visits</h2>
  {% for time in times %}
    <p>{{ time }}</p>
  {% endfor %}

</body>
</html>
```

2. Add behaviors and styles with `static/script.js` and `static/style.css` files:

appengine/standard_python3/building-an-app/building-an-app-1/static/script.js

View on GitHub

```
'use strict';

window.addEventListener('load', function () {
  console.log("Hello World!");
});
```

appengine/standard_python3/building-an-app/building-an-app-1/static/style.css

View on GitHub

```
body {
  font-family: "helvetica", sans-serif;
  text-align: center;
}
```

3. In your `main.py` file, use Flask to render your HTML template with the placeholder data:

appengine/standard_python3/building-an-app/building-an-app-1/main.py

View on GitHub

```
import datetime

from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def root():
    # For the sake of example, use static information to inflate
the template.
    # This will be replaced with real information in later steps.
    dummy_times = [
        datetime.datetime(2018, 1, 1, 10, 0, 0),
        datetime.datetime(2018, 1, 2, 10, 30, 0),
        datetime.datetime(2018, 1, 3, 11, 0, 0),
    ]

    return render_template("index.html", times=dummy_times)


if __name__ == "__main__":
    app.run(host="127.0.0.1", port=8080, debug=True)
```

4. Configure all dependencies you will need for your web service in your `requirements.txt` file:

appengine/standard_python3/building-an-app/building-an-app-1/requirements.txt

View on GitHub

```
Flask==2.1.0
```

3. Test your web service

Test your web service by running it locally in a virtual environment.

Windows:

1. Create an isolated Python environment in a directory external to your project andactivate it.

```
cd ..
python -m venv env
env\Scripts\activate
```

2. Navigate to your project directory and install dependencies:

```
cd building-an-app-1
python -m pip install -r requirements.txt
```

3. Run the application:

```
python main.py
```

4. In your web browser, enter the following address:

```
http://localhost:8080
```

In your terminal window, press **Ctrl+C** to exit the web server.

Mac OS/Linux:

1. Create an isolated Python environment in a directory external to your project andactivate it:

```
python3 -m venv env
source env/bin/activate
```

2. Follow the same Steps 2-4 documented previously for the Windows platform.

4. Configure your web service for App Engine

To deploy your web service to App Engine, you need an `app.yaml` file. This configuration file defines your web service's settings for App Engine.

To configure your web service for deployment to App Engine, create your `app.yaml` file in the root directory of your project, for example `building-an-app`:

appengine/standard_python3/building-an-app/building-an-app-1/app.yaml

View on GitHub

```
runtime: python39

handlers:
  # This configures Google App Engine to serve the files in the
app's static
  # directory.
- url: /static
  static_dir: static

  # This handler routes all requests not caught above to your
main app. It is
  # required when static routes are defined, but can be omitted
(along with
  # the entire handlers section) when there are no static files
defined.
- url: /.*
  script: auto
```

Notice that for this simple web service, your `app.yaml` file needs to define only the runtime setting and handlers for static files.

For more complicated web services, you can configure additional settings in your `app.yaml`, like scaling, additional handlers, and other application elements like environment variables and service names. For more information and a list of all the supported elements, see the `app.yaml` reference.

5. Deploy your web service

5.1 Deploying your service

To deploy your web service, run the `gcloud app deploy` command from the root directory of your project where your `app.yaml` file is located:

```
gcloud app deploy
```

Each time you deploy your web service, a new version of that app is created in Google App Engine. During deployment, a container image is created using the Cloud Build service, and then a copy is uploaded to Google Cloud Storage before it is run in App Engine.

5.2 Viewing your service

To quickly launch your browser and access your web service at `https://YOUR-PROJECT-ID.REGION_ID.r.appspot.com`, enter the following command:

```
gcloud app browse
```

[Tip:] If you would like to change the URL of your web service to something other than the default `https://YOUR-PROJECT-ID.REGION_ID.r.appspot.com` URL, you can add a custom domain.

Managing services and versions

You've just deployed a version of the web service to App Engine. Each time that you deploy a version of your code, that version is created in a service. The initial deployment to App Engine must be created in the `default` service, but for subsequent deployments, you can specify the name of your service in your `app.yaml` file.

You can update a service at any time by running the `gcloud app deploy` command and deploying new versions to that service. Each time that you update a service, traffic is automatically routed to the version last deployed. However, you can include `gcloud` flags to change the deploy command behavior.

Use the Cloud Console to manage and view the services and versions that you deploy to App Engine.

- Use the Cloud Console to view your App Engine services: Go to the services page
- Use the Cloud Console to view your versions: Go to the versions page

For more information about the multi-service design pattern, see An Overview of App Engine. To learn how to send requests to specific services and versions, see Splitting Traffic.

6. Handle data with Datastore

6.1 Storing and retrieving Datastore entities

Store and retrieve site request times as Datastore entities by completing the following:

1. Add the following code to your `main.py` file:

appengine/standard_python3/building-an-app/building-an-app-2/main.py

View on GitHub

```
from google.cloud import datastore

datastore_client = datastore.Client()

def store_time(dt):
    entity = datastore.Entity(key=datastore_client.key("visit"))
    entity.update({"timestamp": dt})

    datastore_client.put(entity)


def fetch_times(limit):
    query = datastore_client.query(kind="visit")
    query.order = ["-timestamp"]

    times = query.fetch(limit=limit)

    return times
```

The `store_time` method above uses the Datastore client libraries to create a new entity in Datastore. Datastore entities are data objects that consist of *keys* and *properties*. In this case, the entity's key is its custom *kind*, `visit`. The entity also has one property, `timestamp`, containing time of a page request.

The `fetch_times` method uses the key `visit` to query the database for the ten most recent `visit` entities and then stores those entities in a list in descending order.

2. Update your `root` method to call your new methods:

appengine/standard_python3/building-an-app/building-an-app-2/main.py

View on GitHub

```
@app.route("/")
def root():
    # Store the current access time in Datastore.
    store_time(datetime.datetime.now(tz=datetime.timezone.utc))

    # Fetch the most recent 10 access times from Datastore.
    times = fetch_times(10)

    return render_template("index.html", times=times)
```

3. Update your `templates/index.html` file to print the `timestamp` of each entity:

appengine/standard_python3/building-an-app/building-an-app-2/templates/index.html

View on GitHub

```
<h2>Last 10 visits</h2>
{% for time in times %}
  <p>{{ time['timestamp'] }}</p>
{% endfor %}
```

4. Ensure that your `requirements.txt` file includes all necessary dependencies:

appengine/standard_python3/building-an-app/building-an-app-2/requirements.txt

View on GitHub

```
Flask==2.1.0
google-cloud-datastore==2.15.1
```

For more information about Datastore entities, properties, and keys, see Entities, Properties, and Keys. For more information on using Datastore client libraries, see Datastore Client Libraries.

6.2 Testing your web service

Test your web service by running it locally in a virtual environment.

1. Run the following commands in your project's main directory to install new dependencies and run your web service. If you have not set up a virtual environment for local testing, see testing your web service.

```
pip install -r requirements.txt
python main.py
```

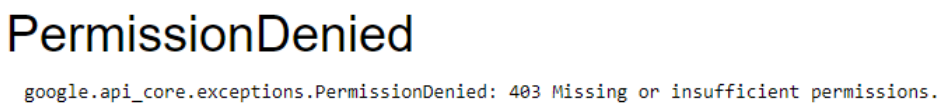
[Note:] See if you run into a errors similar to the ones show below on the command line or browser.

google.auth.exceptions.DefaultCredentialsError: Could not automatically determine credentials. Please set GOOGLE_APPLICATION_CREDENTIALS or explicitly create credentials and re-run the application. For more information, please see https://cloud.google.com/docs/authentication/getting-started

OR

```
Traceback (most recent call last):
  File "C:\Users\[your-user-account]\Documents\RMIT\OUA\2023\SP1\Week2\python-
    docs-samples\appengine\standard_python3\venv\lib\site-
    packages\google\api_core\grpc_helpers.py", line 72, in error_remapped_callable
    return callable_(*args, **kwargs)
  File "C:\Users\[your-user-account]\Documents\RMIT\OUA\2023\SP1\Week2\python-
    docs-samples\appengine\standard_python3\venv\lib\site-
    packages\grpc\_channel.py", line 946, in __call__
    return _end_unary_response_blocking(state, call, False, None)
  File "C:\Users\[your-user-account]\Documents\RMIT\OUA\2023\SP1\Week2\python-
    docs-samples\appengine\standard_python3\venv\lib\site-
    packages\grpc\_channel.py", line 849, in _end_unary_response_blocking
    raise _InactiveRpcError(state)
grpc._channel._InactiveRpcError: <_InactiveRpcError of RPC that terminated with:
        status = StatusCode.PERMISSION_DENIED
        details = "Missing or insufficient permissions."
        debug_error_string = "UNKNOWN:Error received from peer
    ipv6:%5B2404:6800:4009:810::200a%5D:443 {grpc_message:"Missing or insufficient
    permissions.", grpc_status:7, created_time:"2023-03-
    07T04:38:09.931801195+00:00"}"
    >
```

OR (on the browser**)**



If you do, the reason for this error to occur could be, Google Datastore being a secure service, it requires your application to authenticate itself before you can perform operations against it. Setting your credentials via environment or bash variable GOOGLE_APPLICATION_CREDENTIALS is one such form of configuring your credentials to be able to authenticate your app with Google Datastore.

To do this, follow the steps documented in
https://cloud.google.com/docs/authentication/provide-credentials-adc#local-key

2. Enter the following address in your web browser to view your web service:

```
http://localhost:8080
```

[Tip:] Refresh the page to create additional page requests and store more entities in Datastore.

You can view the entities that are created by your web service in the Cloud Console:

Go to the Datastore entities page

6.3 Deploying your web service

Now that you have Datastore working locally, you can re-deploy your web service to App Engine.

Run the following command from the root directory of your project, where your `app.yaml` file is located:

```
gcloud app deploy
```

All traffic is automatically routed to the new version you deployed.

For more information on managing versions, see Managing Services and Versions.

6.4 Viewing your service

To quickly launch your browser and access your web service at `https://YOUR-PROJECT-ID.REGION_ID.r.appspot.com`, run the following command:

```
gcloud app browse
```

7. Add Firebase to your Web Service

7.1 Adding Firebase to your Cloud project

To use Firebase authentication with your web service, add Firebase to your Cloud project and configure your authentication settings.

1. Add Firebase to your existing Cloud project using the "**Add project**" option in the Firebase console.

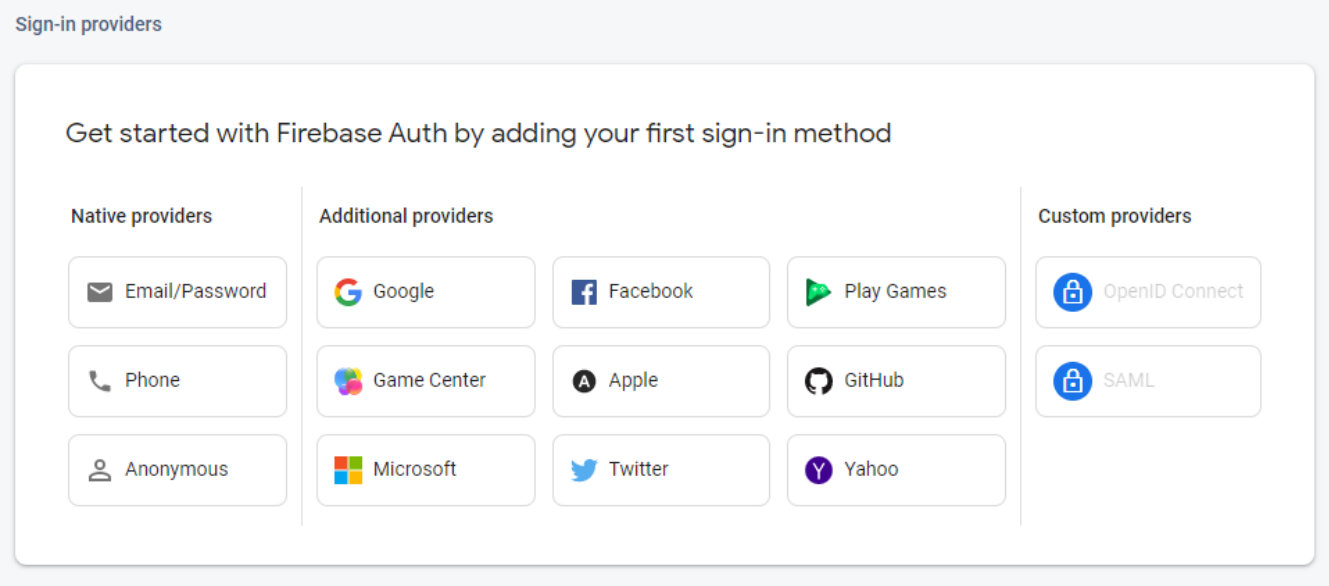
You can also choose to use a Firebase account with a different name, not associated with your existing Cloud project.

2. Enable the authentication sign-on providers in the Firebase console. For this webservice, you will enable **Email/Password** and **Google** sign-in providers:

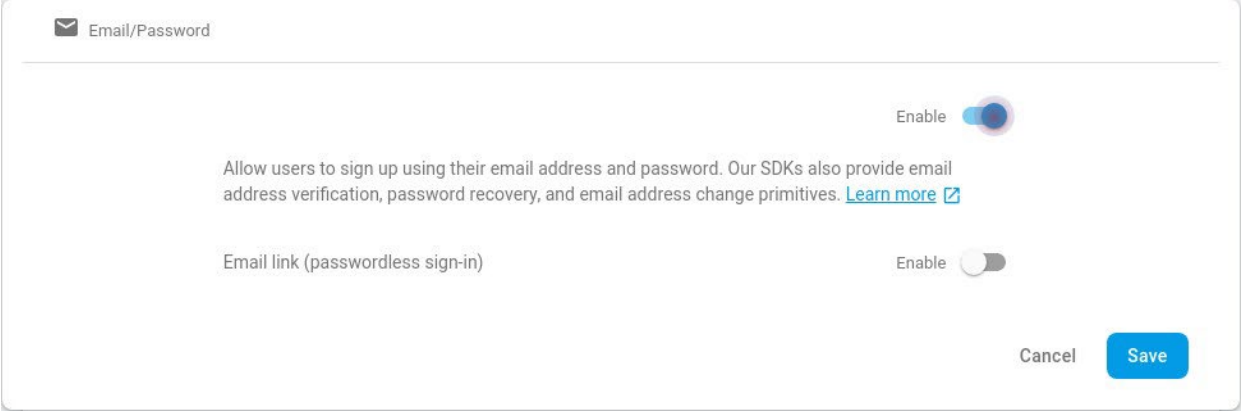
    a. Select "Build" -> "Authentication" from the left-hand side menu.

    b. Then, select **"Sign-in method"** from the tabbed view available under the "Authentication" section.

    c. Under **"Sign-in providers"**, select **"Email/Password"** provider.



d. Toggle the **Enable** button to use **Email/Password** authentication.



e. After enabling the provider, click **Save**.

f. Do the same with the sign-in provider **"Google"** via **"Add a new provider"** option.

[Tip:] For more information about enabling other providers, see the "Before you begin" sections of the Facebook, Twitter, and GitHub guides on Firebase.

3. For Firebase to authenticate properly, your domain needs to be authorized for OAuth redirects.

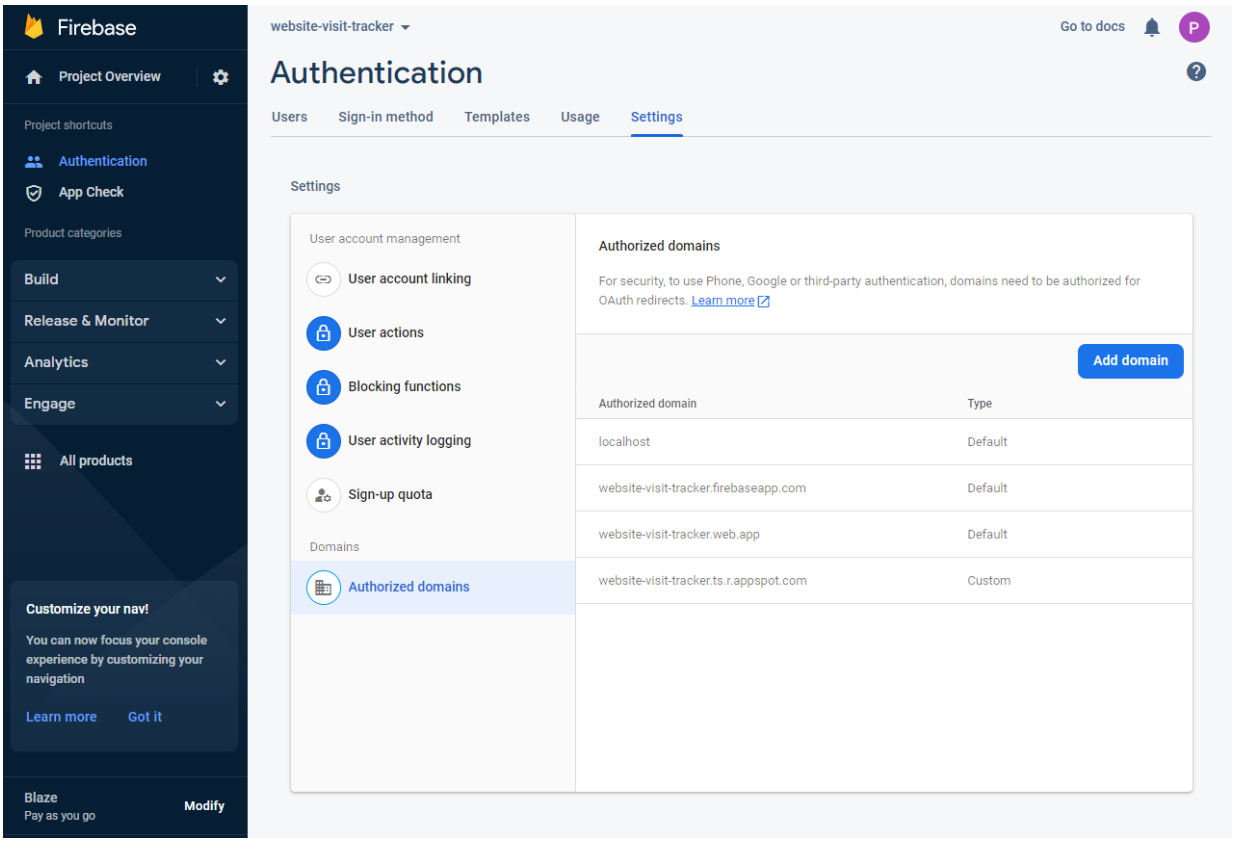
To authorize your domain,

a. select "Settings" tab on the tabbed view within the "Authentication page.

b. scroll down to **"Authorized Domains"** section, click **"Add Domain"**, and then enter the domain of your app on App Engine, excluding the `http://` prefix:

`YOUR_PROJECT_ID.REGION_ID.r.appspot.com` where `YOUR_PROJECT_ID` is the ID of your Cloudproject.

If successful, you would see a "**custom**" domain gets added under the "**Authorized domains**" section, as below (e.g. website-visit-tracker.ts.r.appspot.com).



7.2 Adding Firebase to your web service

To add Firebase to your web service, copy your Firebase project's custom code snippet, JavaScript and CSS files into your web service:

1. Go to the Firebase console and select your project.

2. From the project overview page, click **Add app**, select **Web**. If you already have an app added to the project, you may not see this text.

3. Once the app is registered, a customized code snippet will be displayed. Copy the contents of the snippet. To see this code snippet again later, navigate to the **settings - General** page for your Firebase app.

4. Update your `templates/index.html` file by completing the following:

a. Add the following lines to the `<head>` tag:

```
<script>
    // For Firebase JS SDK v7.20.0 and later, measurementId is
optional
    const firebaseConfig = {
       apiKey: "<YOUR_API_KEY>",
       authDomain: "<YOUR_PROJECT_ID>.firebaseapp.com",
       projectId: "<YOUR_PROJECT_ID>",
       storageBucket: "<YOUR_PROJECT_ID>.appspot.com",
       messagingSenderId: "<YOUR_MESSAGING_SENDER_ID>",
       appId: "<YOUR_APP_ID>",
       measurementId: "<YOUR_MEASUREMENT_ID>"
   };


  firebase.initializeApp(firebaseConfig);

  if (typeof firebase === 'undefined') {
    const msg = "Please provide your Firebase configuration
details. See instructions in templates/index.html.";
    console.log(msg);
    alert(msg);
  }
</script>
```

b. Add your customized code snippet to the `<body>` tag.

For this tutorial, you can add the code to the top of the body, since the only content in `templates/index.html` is an example of Firebase services. In your production environment, we recommend that you add the code snippet to the bottom of the body, but before you use any Firebase services.

Your custom code will look similar to this mock snippet.

c. Replace the rest of the body with the following code, which you will use later in this guide to display authenticated user data.

appengine/standard_python3/building-an-app/building-an-app-3/templates/index.html

View on GitHub

```
<h1>Datastore and Firebase Auth Example</h1>

<div id="firebaseui-auth-container"></div>

<button id="sign-out" hidden=true>Sign Out</button>

<div id="login-info" hidden=true>
  <h2>Login info:</h2>
  {% if user_data %}
    <dl>
      <dt>Name</dt><dd>{{ user_data['name'] }}</dd>
      <dt>Email</dt><dd>{{ user_data['email'] }}</dd>
      <dt>Last 10 visits</dt><dd>
   {% for time in times %}
          <p>{{ time['timestamp'] }}</p>
        {% endfor %} </dd>
    </dl>
  {% elif error_message %}
    <p>Error: {{ error_message }}</p>
  {% endif %}
</div>
```

8. Authenticating users

8.1 Adding Firebase authentication methods

Firebase provides JavaScript methods and variables that you can use to configure sign-in behavior for your web service. For this web service, add a sign out function, a variable that configures the sign in UI, and a function controlling what changes when a user signs in or out.

To add the behaviors required for an authentication flow, replace your `static/script.js` file's current event listener method with the following code.

appengine/standard_python3/building-an-app/building-an-app-3/static/script.js

View on GitHub

```
window.addEventListener('load', function () {
  document.getElementById('sign-out').onclick = function () {
    firebase.auth().signOut();
  };

  // FirebaseUI config.
  var uiConfig = {
    signInSuccessUrl: '/',
    signInOptions: [
      firebase.auth.GoogleAuthProvider.PROVIDER_ID,
      firebase.auth.EmailAuthProvider.PROVIDER_ID,
    ],
    // Terms of service url.
    tosUrl: '<your-tos-url>'
  };

  firebase.auth().onAuthStateChanged(function (user) {
    if (user) {
      // User is signed in.
      document.getElementById('sign-out').hidden = false;
      document.getElementById('login-info').hidden = false;
      console.log(`Signed in as ${user.displayName}
(${user.email})`);
      user.getIdToken().then(function (token) {
        document.cookie = "token=" + token;
      });
    } else {
      // User is signed out.
      // Initialize the FirebaseUI Widget using Firebase.
      var ui = new firebaseui.auth.AuthUI(firebase.auth());
      // Show the Firebase login button.
      ui.start('#firebaseui-auth-container', uiConfig);
      // Update the login state indicators.
      document.getElementById('sign-out').hidden = true;
      document.getElementById('login-info').hidden = true;
      // Clear the token cookie.
      document.cookie = "token=";
    }
  }, function (error) {
    console.log(error);
    alert('Unable to log in: ' + error)
  });
});
```

Notice that the `onAuthStateChanged()` method, which controls what changes when a user signs in or out, stores the user's ID token as a cookie. This ID token is a unique token that Firebase generates automatically when a user successfully signs in, and is used by the server to authenticate the user.

[Tip:] including a terms of service URL is optional.

8.2 Updating your web service to use tokens

Next, verify users on the server using their unique Firebase ID token, then decrypt their token so that you can print their data back to them.

To use the Firebase ID token,

1. Retrieve, verify, and decrypt the token in the `root` method of your `main.py` file.

appengine/standard_python3/building-an-app/building-an-app-3/main.py

View on GitHub

```
import datetime

from flask import Flask, render_template, request
from google.auth.transport import requests
from google.cloud import datastore
import google.oauth2.id_token

firebase_request_adapter = requests.Request()
datastore_client = datastore.Client()

app = Flask(__name__)

def store_time(email, dt):
    entity = datastore.Entity(key=datastore_client.key("User",
email, "visit"))
    entity.update({"timestamp": dt})

    datastore_client.put(entity)

def fetch_times(email, limit):
    ancestor = datastore_client.key("User", email)
    query = datastore_client.query(kind="visit",
ancestor=ancestor)
    query.order = ["-timestamp"]

    times = query.fetch(limit=limit)

    return times
```

```
@app.route("/")
def root():
    # Verify Firebase auth.
    id_token = request.cookies.get("token")
    error_message = None
    claims = None
    times = None

    if id_token:
        try:
            claims =
google.oauth2.id_token.verify_firebase_token(id_token,
firebase_request_adapter
            )

            store_time(claims["email"],
datetime.datetime.now(tz=datetime.timezone.utc))
            times = fetch_times(claims["email"], 10)

        except ValueError as exc:
            error_message = str(exc)

    return render_template("index.html", user_data=claims,
error_message=error_message, times=times
    )
```

[Tip:] Make sure to import `request` from Flask so that you can fetch the cookiecontaining the user's ID token.

2. Ensure that your `requirements.txt` file includes all necessary dependencies:

appengine/standard_python3/building-an-app/building-an-app-3/requirements.txt

View on GitHub

```
Flask==2.1.0
google-cloud-datastore==2.15.1
google-auth==2.17.3
requests==2.28.2
```

8.3 Testing your web service

Test your web service by running it locally in a virtual environment:

1. Run the following commands in your project's main directory to install new dependencies and run your web service. If you have not set up a virtual environment for local testing, see testing your web service.

```
pip install -r requirements.txt
python main.py
```

2. Enter the following address in your web browser to view your web service:

```
http://localhost:8080
```

9. Personalize data for authenticated users

9.1 Storing and retrieving user-specific data

You can indicate that data is connected to a certain user by using Datastore ancestors, which allow you to organize your Datastore data hierarchically.

To do this, complete the following steps:

1. Update your `store_time` and `fetch_time` methods to use Datastore ancestors forstoring and retrieving `visit` entities:

appengine/standard_python3/building-an-app/building-an-app-3/requirements.txt

View on GitHub

```
datastore_client = datastore.Client()

def store_time(email, dt):
    entity = datastore.Entity(key=datastore_client.key("User",
email, "visit"))
    entity.update({"timestamp": dt})

    datastore_client.put(entity)


def fetch_times(email, limit):
    ancestor = datastore_client.key("User", email)
    query = datastore_client.query(kind="visit",
ancestor=ancestor)
    query.order = ["-timestamp"]

    times = query.fetch(limit=limit)

    return times
```

Each `visit` entity now has an ancestor that it is connected to. These ancestors are Datastore entities that represent individual authenticated users. Each ancestor's *key* includes the `User` *kind* and a custom ID, which is the authenticated user'semail address. You use the ancestor key to query the database for only the times that are associated with a specific user.

2. Update the `store_times` method call in your `root` method and move it inside the `id_token` conditional so that it only runs if the server has authenticated a user:

appengine/standard_python3/building-an-app/building-an-app-4/main.py

View on GitHub

```
@app.route("/")
def root():
    # Verify Firebase auth.
    id_token = request.cookies.get("token")
    error_message = None
    claims = None
    times = None

    if id_token:
        try:
            claims =
google.oauth2.id_token.verify_firebase_token(
                id_token, firebase_request_adapter
            )

            store_time(claims["email"],
datetime.datetime.now(tz=datetime.timezone.utc))
            times = fetch_times(claims["email"], 10)

        except ValueError as exc:
            # This will be raised if the token is expired or any
other
            # verification checks fail.
            error_message = str(exc)

    return render_template(
        "index.html", user_data=claims,
error_message=error_message, times=times
    )
```

9.2 Configuring indexes

Datastore makes queries based on indexes. For simple entities, Datastore automatically generates these indexes. However, it cannot automatically generate indexes for more

complicated entities, including those with ancestors. Because of this, you need to manually create an index for `visit` entities so that Datastore can perform queries involving `visit` entities.

To create an index for `visit` entities, complete the following steps:

1. Create an `index.yaml` file in the root directory of your project, for example `building-an-app`, and add the following index:

appengine/standard_python3/building-an-app/building-an-app-4/index.yaml

View on GitHub

```
indexes:

- kind: visit
  ancestor: yes
  properties:
  - name: timestamp
    direction: desc
```

2. Deploy your `index.yaml` indexes in Datastore by running the following command and following the prompts:

```
gcloud datastore indexes create index.yaml
```

It can take a while for Datastore to create indexes. Creating indexes before deploying your web service to App Engine both allows you to test locally using those indexes and prevents exceptions that might occur for queries that require an index that is still in the process of being built.

Tip: You can view the index you created in the Datastore console:

View your indexes

For more information on making Datastore indexes, see Configuring Datastore Indexes.

9.3 Testing your web service

Test your web service by running it locally in a virtual environment:

1. Run the following command in your project's main directory to run your web service. If you have not set up a virtual environment for local testing, see testing your web service.

```
python main.py
```

2. Enter the following address in your web browser to view your web service:

```
http://localhost:8080
```

9.4 Deploying your web service

Now that you have Datastore working locally, you can re-deploy your web service to App Engine.

Run the following command from the root directory of your project, where your `app.yaml` file is located:

```
gcloud app deploy
```

All traffic is automatically routed to the new version you deployed.

For more information on managing versions, see Managing Services and Versions.

9.5 Viewing your service

To quickly launch your browser and access your web service at `https://PROJECT_ID.REGION_ID.r.appspot.com`, run the following command:

```
gcloud app browse
```

10. Clean up resources

Clean up your Google Cloud project to avoid incurring charges to your account.

While leaving your project and web service running will not necessarily incur charges, a large amount of traffic to your service might cause you to exceed the free quotas. To avoid incurring charges, you can clean up your project by deleting or disabling your application and other associated resources.

10.1 Deciding how to clean up

You have several options for cleaning up your project that allow you to retain different amounts of project information and application data:

- Disable your application

Retain your project name, application data, and other project resources while preventing charges from traffic to your web service.

- Disable billing

Retain your project name and application data while preventing charges for any resources associated with your project.

- Delete your project

Delete your project name, application data, and all other project resources.

10.2 Disabling your application

Disabling your application stops your application from running instances and serving requests while retaining application data and settings. Note that billing charges can still occur for the other services in your Cloud project, like stored data.

To disable an App Engine application, go to the Application settings page, click **Disable application**, and then follow the prompts:

Go to the Application settings page

If your app is actively processing a request, it will continue to complete that task and can take up to an hour before your app is completely disabled.

For more information about disabling your application, see Disabling an application

10.3 Disabling billing

Disabling billing stops automatic payments for all services in your project. Note that even if you disable billing, you're still be responsible for all outstanding charges on the account, which will be charged to your listed form of payment.

To disable billing for a project:

1. Go to the Google Cloud Console.
2. Open the left side menu and select **Billing**.
3. If you have more than one billing account, select **Go to linked billing account** to manage the current project's billing. To locate a different billing account, select **Manage billing accounts**.
4. Under **Projects linked to this billing account**, locate the name of the project that you want to disable billing for, and then from the menu next to it, select **Disable billing**. You are prompted toconfirm that you want to disable billing for this project.
5. Click **Disable billing**.

Caution: If your billing account remains disabled for a protracted period, some resources might be removed from the projects associated with that account. For example, if you use Google Cloud Platform, your Google Compute Engine resources might be removed. Removed resources are not recoverable.

10.4 Deleting your project

You can release all the Google Cloud resources in your Cloud project by deleting your project.

[Caution:] Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when youdelete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, suchas an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

2. In the Cloud Console, go to the **Manage resources** page.

Go to the Manage resources page

3. In the project list, select the project that you want to delete and then click **Delete** to delete.
4. In the dialog, type the project ID and then click **Shut down** to delete the project.

[Note:] Make sure you delete your project to avoid any unexpected charges.

Additional resources

Datastore mode Client Libraries for Python, PHP and other programming languages can be referenced from

QuickStart: https://cloud.google.com/datastore/docs/reference/libraries

Detail: https://cloud.google.com/datastore/docs/datastore-api-tutorial

Sample code: https://github.com/GoogleCloudPlatform/php-docs-samples

Firebase Admin SDK for PHP can be referenced from https://firebase-php.readthedocs.io/en/stable/