

 indicates that a section was completed but did not request any screenshots or written answers.

TABLE OF CONTENTS:

<b>I. Lab 8.1a: API gateway.....</b>	<b>4</b>
1. MOTD Function.....	4
2. API Integration.....	4
3. Lambda Code.....	4
4. Test Code.....	5
5. Clean Up.....	5
6. gettime API.....	6
7. Implement Code.....	6
8. Test Code.....	6
9. Clean Up.....	6
<b>II. Lab 8.2a: Lambda, API Gateway guestbook.....</b>	<b>7</b>
1. Overview.....	7
2. Obtain AWS Account ID.....	7
3. REST API Code.....	7
4. Deploy the Lambda for viewing entries.....	7
5. Create API in API Gateway.....	8
6. Enable API to invoke lambda function.....	8
7. API Endpoint for viewing entries (1).....	8
8. API endpoint for viewing entries (2).....	8
9. CORS setup for viewing entries.....	9
10. Deploy API to production and view entries.....	10
11. API endpoint for signing (1).....	11
12. API endpoint for signing (2).....	11
13. CORS setup for signing.....	12
14. Deploy API to production and sign.....	12
15. frontend code.....	12
16. Configure and deploy the frontend.....	13
17. Clean Up.....	13
<b>III. Lab 8.2g: Cloud functions API gateway guestbook.....</b>	<b>14</b>
1. Overview.....	14
2. Cloud function backend (GET).....	14
3. Cloud function backend (POST).....	14
4. Deploy the cloud function.....	15
5. Test the cloud functions deployment.....	15
6. API Gateway.....	15
7. Create OpenAPI specification.....	16
8. -.....	16
9. Service account setup.....	16

10. Create the API gateway.....	16
11. test the api via python requests (GET).....	17
12. test the api via python requests (POST).....	17
13. Client-side guestbook application.....	18
14. guestbook.js.....	18
15. version #1: local file system.....	18
16. version #2: google cloud storage bucket.....	19
17. clean up.....	19
<b>IV. Lab 8.3g: OAuth2 Guestbook.....</b>	<b>20</b>
1. oauth2 guestbook.....	20
2. oauth2 authorization code flow.....	20
3. checkout code.....	20
4. oauth initiation code.....	20
5. identity provider interaction.....	20
6. callback code.....	21
7. -.....	21
8. signing page.....	21
9. model code.....	21
10. build and deploy the code.....	21
11. set up identity provider.....	22
12. -.....	22
13. update deployment.....	22
14. visit the application.....	23
15. secrets manager deployment.....	26
16. removing access.....	27
17. clean up.....	27
<b>V. Lab 8.4g: Firebase.....</b>	<b>28</b>
1. firebase web application.....	28
2. project setup.....	28
3. application setup.....	28
4. authentication setup.....	29
5. database setup.....	29
6. storage setup.....	29
7. cli setup.....	30
8. bundling with webpack.....	30
9. configure firebase within application.....	31
10. initialize firebase within application.....	31
11. test application.....	31
12. add authentication.....	32
13. update UI.....	34
14. test application with authentication.....	35

15. add text messaging.....	36
16. test application with text messaging.....	37
17. manual message insertion.....	38
18. add image messaging.....	39
19. Test application with image messaging.....	40
20. deploy application.....	41
21. Clean Up.....	41

## I. Lab 8.1a: API gateway

### 1. MOTD Function

- Bring up lambda in the web console and create a function from scratch named lambda-amminer, using the latest python runtime. Assign it the LabRole.



### 2. API Integration

- Add a trigger. API Gateway accommodates our web request trigger. Select “Create and API” and specify HTTP and open security (publicly accessible).



### 3. Lambda Code

- Replace the default `lambda_function.py` code with something functionally equivalent to the lab example.
  - The function (`lambda_handler`) first generates a random URL to an image site with a message of the day.
  - It then retrieves the contents of the URL and returns a dictionary object whose key-value pairs represent
    - the HTTP response status,
    - the `Content-Type`: response header,
    - and the payload returned.

The format of this dictionary is required by the API gateway.

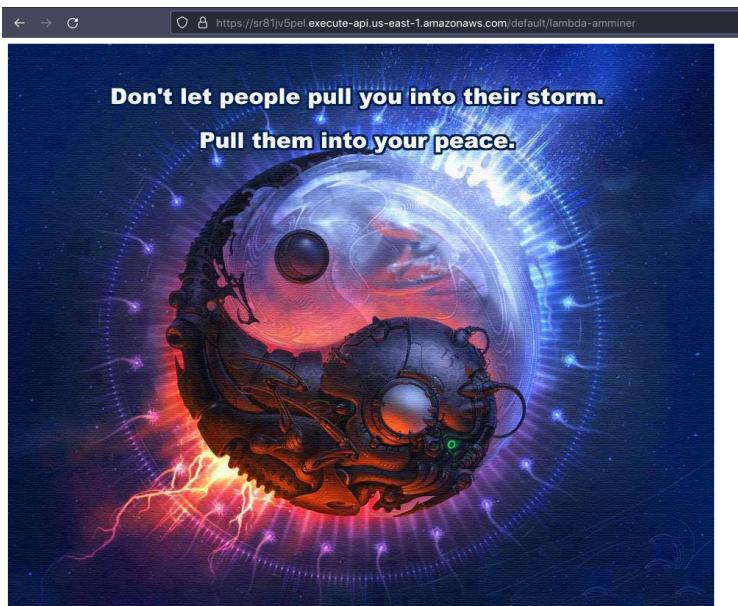
Click deploy, then test.



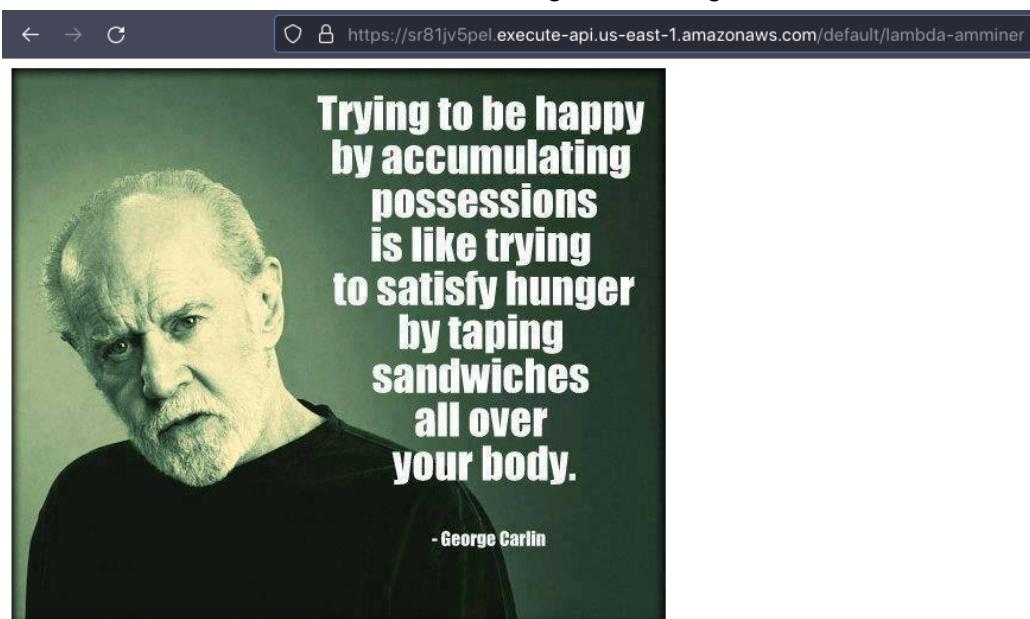
## 4. Test Code

- Create a test event. the body does not matter, our lambda function will ignore it. Test the function. Go to API Gateway in the console and find the endpoint for the function. Copy the url of the default stage into your browser's URL bar, and append your lambda function's name to the end of the base url. Load the image.

**AWS python 3.11 runtime has urllib3 1.x at the time of writing.**



- Reload the browser and show that the image has changed.



## 5. Clean Up

- Delete the lambda and the API.

## 6. gettime API

- Just like the last exercise, create a lambda function named gettime-amminer with the LabRole role and an API Gateway/REST API trigger.



## 7. Implement Code

- Implement the example code from the lab.

The code

- retrieves the current time via `datetime.now()`,
- then creates a Python dictionary that contains a single key 'currentTime' and sets its value to the string representation of the current time.
- Then, it creates a response object for API Gateway that sets the Content-Type: header to be `application/json`
- and returns the JSON encoding of the dictionary in the body via `json.dumps()`.



## 8. Test Code

- Test and run the code. Use curl on your Linux machine to access the endpoint externally and show the result.

```
meelzBox:~ > curl https://nym3b0zwq6.execute-api.us-east-1.amazonaws.com/default/gettime-amminer && echo
{"currentTime": "2023-11-15 06:25:28.587261"}
meelzBox:~ >
```

## 9. Clean Up

- Delete the function and api.



## II. Lab 8.2a: Lambda, API Gateway guestbook

### 1. Overview

- We're splitting the guestbook into a serverless REST API backend using Gateway and Lambda + a static front end hosted on an S3 bucket.



### 2. Obtain AWS Account ID

- Get your credentials:

```
aws sts get-caller-identity
aws_account_id=$(aws sts get-caller-identity | jq '.Account | tonumber')
echo $aws_account_id
```



**Lower case shell variable??**

### 3. REST API Code

- Clone the class repo and cd into cs430-src/06\_aws\_restapi\_lambda/restapi/lambda.  
Examine sign.py.



### 4. Deploy the Lambda for viewing entries

- set a variable in the shell called odin\_id to your OdinID and create a zip file containing the code for the entry viewer lambda.

```
aws lambda create-function --function-name ${odin_id}-gb-lambda \
--zip-file fileb://./function.zip \
--handler get_entries.handler \
--runtime python3.7 \
--environment Variables='{TABLE='guestbook'}' \
--role arn:aws:iam::217663952189:role/LabRole
```

Wait until the function comes up, then test the function by invoking it and sending its output to a file.

```
aws lambda invoke --function-name ${odin_id}-gb-lambda
gb-lambda.out
cat gb-lambda.out
```



## 5. Create API in API Gateway

- Create a rest API and take note of its id. Set a shell variable `api_id`.

```
aws apigateway create-rest-api --name ${odin_id}-gb-restapi
```



## 6. Enable API to invoke lambda function

- Security.

```
aws lambda add-permission --function-name ${odin_id}-gb-lambda \
    --statement-id apigateway-test-2 \
    --action lambda:InvokeFunction \
    --principal apigateway.amazonaws.com \
    --source-arn "arn:aws:execute-api:us-east-1:${aws_account_id}:$api_id/*"
```



## 7. API Endpoint for viewing entries (1)

- Get the resource ID of the API's root. Store it in a `root_id` shell var.

```
aws apigateway get-resources --rest-api-id $api_id
```

Instantiate a resource under the root endpoint for viewing the entries, called `/entries`.

```
aws apigateway create-resource --rest-api-id $api_id \
    --path-part entries \
    --parent-id $root_id
```

Record the resource's ID - call it `entries_id`.

## 8. API endpoint for viewing entries (2)

- Configure a method for the `entries` resource in API Gateway - this configures the outward-facing interface.

```
aws apigateway put-method --rest-api-id $api_id \
    --resource-id $entries_id \
    --http-method GET \
    --authorization-type NONE \
    --no-api-key-required
```

Integrations, as opposed to methods, configure the internal interface:

```
aws apigateway put-integration --rest-api-id $api_id \
    --resource-id $entries_id \
    --http-method GET \
    --type AWS_PROXY \
    --integration-http-method POST \
    --uri
arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us
-east-1:$aws_account_id:function:${odin_id}-gb-lambda/invocations
```

Since the application forwards JSON directly between the front and backend, we can just pass it straight through like this, no need to specify any response actions for the method or integration, but note that it has this option. Visit the API Gateway page in the console and test the API.

### What a URI!



## 9. CORS setup for viewing entries

- specify the method for the OPTIONS request:

```
aws apigateway put-method --rest-api-id $api_id \
    --resource-id $entries_id \
    --http-method OPTIONS \
    --authorization-type NONE \
    --no-api-key-required
```

specify the method response:

```
aws apigateway put-method-response --rest-api-id $api_id \
    --resource-id $entries_id \
    --http-method OPTIONS \
    --status-code 204 \
    --response-parameters
'{"method.response.header.Access-Control-Allow-Origin": true,
"method.response.header.Access-Control-Allow-Methods": true,
"method.response.header.Access-Control-Allow-Headers": true}'
```

specify the integration and its response as well. Since we don't actually need an integration for this functionality, we use the MOCK type.

```
aws apigateway put-integration --rest-api-id $api_id \
    --resource-id $entries_id \
    --type MOCK \
    --http-method OPTIONS \
    --request-templates '{ "application/json" : "{\"statusCode\": 200}" }'
aws apigateway put-integration-response --rest-api-id $api_id \
    --resource-id $entries_id \
    --http-method OPTIONS \
    --status-code 204 \
    --selection-pattern "-" \
    --response-templates "{\"application/json\": \"Empty\"}" \
    --response-parameters '{"method.response.header.Access-Control-Allow-Headers": \
    :
    """Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token,X-Am
    z-User-Agent""", "method.response.header.Access-Control-Allow-Methods": \
    """GET,POST""", "method.response.header.Access-Control-Allow-Origin": \
    """*"""}'
```

Test the endpoint as before.

## 10. Deploy API to production and view entries

- Deploy the API in the prod stage (stages are generally used to stage changes before deployment to production):

```
aws apigateway create-deployment --rest-api-id $api_id \  
--stage-name prod
```

View the deployment in the web console and copy its invoke URL for the front end code.



- Checkout the course repo locally, cd

cs430-src/06\_aws\_restapi\_lambda/frontend/src and edit the baseApiUrl constant in static/guestbook.js. Take note of how this URL is used to build the URLs for the other endpoints and be sure to paste in the correct path (to the prod deployment). View 06\_aws\_restapi\_lambda/frontend/src/index.html in a browser - finally, a screenshot! Show that the page loads and pulls down your entry/entries.

file:///home/meelz/cs430-src/06\_aws\_restapi\_lambda/frontend/src/index.html



### Guestbook

Name:

Email:

Message:

Sign

### Entries

Amelia <amminer@pdx.edu>  
signed on 2023-11-05 06:57:20.315270  
Hello ECS!

---

## 11. API endpoint for signing (1)

- Since we've gone over the process for the entry retrieval API, I'll be a little less verbose here. Note how big of a pain in the butt this is and how nice it would be to automate it. In the cloud shell, zip sign.py. Create the lambda, add permissions, and create the /entry endpoint resource. Configure the method and integration requests for the endpoint.



## 12. API endpoint for signing (2)

- Test the API endpoint as before. Show the passing test.

**/entry - POST method test results**

Request	Latency	Status
/entry	2454	200

Response body

```
[{"message": "Hello ECS!", "date": "2023-11-05 06:57:20.315270", "email": "amminer@pdx.edu", "name": "Amelia"}, {"message": "Hello API Gateway", "date": "2023-11-16 05:26:15.739919", "email": "amminer@pdx.edu", "name": "Amelia"}]
```

Response headers

```
{  
    "Access-Control-Allow-Origin": "*",  
    "X-Amzn-Trace-Id": "Root=1-6555a7f5-443bbf6faccf797863d0d1b1;Sampled=0;  
lineage=d7c05e72:0"  
}
```

Log

```
Execution log for request 057352c8-1cd-4299-b018-182f5195a745  
Thu Nov 16 05:26:13 UTC 2023 : Starting execution for request: 057352c8-1cd-4299-  
b018-182f5195a745  
Thu Nov 16 05:26:13 UTC 2023 : HTTP Method: POST Resource Path: /entry
```

### 13. CORS setup for signing

- Steps are the same as for the `entries` route, but with the `entry` route.  


### 14. Deploy API to production and sign

- From your local clone of the class repo, bring up the static front end again and sign the guestbook. See that it worked... no screenshot required?



### 15. frontend code

- Examine the frontend code more closely from the cloud shell.  


## 16. Configure and deploy the frontend

- Edit your API's prod deployment's base URL into guestbook.js in the cloud shell clone.

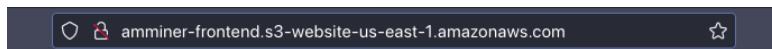
Create and configure an S3 bucket to be publicly readable by anyone.

Sync the frontend code to the bucket and set the index document:

```
aws s3 sync . s3://${odin_id}-frontend/
```

```
aws s3 website s3://${odin_id}-frontend/ --index-document index.html
```

Sign it and show that it worked.



### Entries

Amelia <amminer@pdx.edu>  
signed on 2023-11-05 06:57:20.315270  
Hello ECS!

---

Amelia <amminer@pdx.edu>  
signed on 2023-11-16 05:26:15.739919  
Hello API Gateway

---

Amelia <amminer@pdx.edu>  
signed on 2023-11-16 05:33:54.954505  
Hello API Gateway from local HTML

---

Amelia <amminer@pdx.edu>  
signed on 2023-11-16 05:58:08.294100  
Hello S3, API Gateway, Lambda!

---

## 17. Clean Up

- Remove the lambda, API, and bucket... “Keep your cloud9 environment”

I accidentally closed the tab I ran all of the commands in at an earlier step, so I can't save anything at this point. For the record I'm not perfectly clear on what this step is asking for... printenv & save the environment variables from the process?

✓, I guess...

### III. Lab 8.2g: Cloud functions API gateway guestbook

#### 1. Overview

- We will now do the equivalent of 8.2a but on Google's platform.



#### 2. Cloud function backend (GET)

- In cloud shell, cd cs430-src/06\_gcp\_restapi\_cloudfunctions and examine the entries function in main.py, which interfaces with the entries API endpoint. Instead of generating the server rendering pages from Jinja2 templates, it sends back JSON and rendering is handled by the static client-side frontend. Note that the CORS OPTIONS endpoint is also handled here. Note that you must edit your project idea into the model code.



#### 3. Cloud function backend (POST)

- Examine the entries function, which interfaces with the entries API endpoint. Note that the CORS endpoint for this route is also handled here.



## 4. Deploy the cloud function

- In the directory containing main.py, deploy the cloud functions:

```
gcloud functions deploy entries \
    --runtime python37 \
    --trigger-http \
    --service-account
guestbook@$GOOGLE_CLOUD_PROJECT.iam.gserviceaccount.com
```

```
gcloud functions deploy entry \
    --runtime python37 \
    --trigger-http \
    --service-account
guestbook@$GOOGLE_CLOUD_PROJECT.iam.gserviceaccount.com
disallow unauthenticated access
Examine the functions' URLs etc. using gcloud functions describe <funcname>.
https://us-central1-cloud-miner-amminer.cloudfunctions.net/entries
https://us-central1-cloud-miner-amminer.cloudfunctions.net/entry
✓
```

## 5. Test the cloud functions deployment

- Use curl to hit the entries function. You should get an error since you're not auth'd up.  
We can auth up by attaching an identity token to our request header:

```
curl https://...cloudfunctions.net/entries -H "Authorization:
Bearer $(gcloud auth print-identity-token)"
```



## 6. API Gateway

- enable the required services:

```
gcloud services enable apigateway.googleapis.com
gcloud services enable servicemanagement.googleapis.com
gcloud services enable servicecontrol.googleapis.com
```

and create the API:

```
gcloud api-gateway apis create gbapi --project=$GOOGLE_CLOUD_PROJECT
✓
```

## 7. Create OpenAPI specification

- Edit your entries function's URL into openapi.yaml.



## 8. -

- Do the same for the entry function.



## 9. Service account setup

- set up the permissions to allow API Gateway to authenticate its requests to the Cloud Function backend:

```
gcloud iam service-accounts create gbapisa
gcloud projects add-iam-policy-binding ${GOOGLE_CLOUD_PROJECT} \
    --member
serviceAccount:gbapisa@${GOOGLE_CLOUD_PROJECT}.iam.gserviceaccoun
t.com \
    --role roles/cloudfunctions.invoker
```



## 10. Create the API gateway

- Create an api gateway configuration:

```
gcloud api-gateway api-configs create gbapiconfig \
    --api=gbapi --openapi-spec=openapi.yaml \
    --project=${GOOGLE_CLOUD_PROJECT}
--backend-auth-service-account=gbapisa@${GOOGLE_CLOUD_PROJECT}.ia
m.gserviceaccount.com
```

and instantiate it:

```
gcloud api-gateway gateways create gbapigw \
    --api=gbapi --api-config=gbapiconfig \
    --location=us-central1 --project=${GOOGLE_CLOUD_PROJECT}
```

Show the output containing your API's hostname.

```
amminer@cloudshell:~ (cloud-miner-amminer)$ gcloud api-gateway gateways describe gbapigw --location=us-central1
apiConfig: projects/376082578160/locations/global/apis/gbapi/configs/gbapiconfig
createTime: '2023-11-16T18:48:59.895758963Z'
defaultHostname: gbapigw-4srpuui8.uc.gateway.dev
displayName: gbapigw
name: projects/cloud-miner-amminer/locations/us-central1/gateways/gbapigw
state: ACTIVE
updateTime: '2023-11-16T18:54:33.185409676Z'
amminer@cloudshell:~ (cloud-miner-amminer)$
```

**gbapigw-4srpuui8.uc.gateway.dev**

## 11. test the api via python requests (GET)

- In a python interpreter interactive session, use the requests package to get the entries. Examine the status\_code, headers, and text attributes and the return value of the json function. Assign the response JSON to a variable and print the name, email, date, and message of the first guestbook entry returned in a loop.

```
>>> while True:  
...     print(rj[0]['name'])  
...     print(rj[0]['email'])  
...     print(rj[0]['date'])  
...     print(rj[0]['message'])  
...     sleep(1)  
...  
Amelia  
amminer@pdx.edu  
2023-11-13 01:26:36.732318+00:00  
Hello Kubernetes!  
Amelia  
amminer@pdx.edu  
2023-11-13 01:26:36.732318+00:00  
Hello Kubernetes!  
Amelia  
amminer@pdx.edu  
2023-11-13 01:26:36.732318+00:00  
Hello Kubernetes!  
^CTraceback (most recent call last):  
  File "<stdin>", line 6, in <module>  
KeyboardInterrupt  
>>> []
```

## 12. test the api via python requests (POST)

- Within the same interpreter session, import the JSON package and create a dictionary containing a Guestbook entry with your name, email, and message of "Hello Cloud Functions from Python Requests!". POST it. Show the response status, headers, and text.

```
>>> resp = requests.post('https://gbapigw-4srpuu18.uc.gateway.dev/entry', json=mydict)  
>>> print(resp.status_code, resp.headers, resp.text, sep='\n')  
200  
{'content-type': 'application/json', 'access-control-allow-origin': '*', 'function-execution-id': 'sir278cy2a3c', 'x-cloud-trace-context': '1483d14e0c5850c945f6a0b8963de4ea;0=1', 'alt-svc': 'h3=":443"; ma=2592000,h3-29=":443"; ma=2592000', 'Date': 'Thu, 16 Nov 2023 19:32:48 MT', 'Server': 'Google Frontend', 'Content-Length': '1151'}  
[{"name": "Amelia", "email": "amminer@pdx.edu", "date": "2023-11-13 01:26:36.732318+00:00", "message": "Hello Kubernetes!"}, {"name": "Amelia", "email": "amminer@pdx.edu", "date": "2023-10-30 01:05:00.939198+00:00", "message": "Hello Cloud Shell!"}, {"name": "Amelia", "email": "amminer@pdx.edu", "date": "2023-10-29 17:39:21.049839+00:00", "message": "Hello Datastore"}, {"name": "Amelia", "email": "amminer@pdx.edu", "date": "2023-11-13 02:32:23.019014+00:00", "message": "Hello Cloud Build!"}, {"name": "Amelia", "email": "amminer@pdx.edu", "date": "2023-11-03 00:43:25.874230+00:00", "message": "Hello App Engine!"}, {"name": "Amelia", "email": "amminer@pdx.edu", "date": "2023-10-30 03:51:15.596005+00:00", "message": "Hello Compute Engine!"}, {"name": "Amelia", "email": "amminer@pdx.edu", "date": "2023-10-30 08:12+00:00", "message": "Hello Docker Datastore!"}, {"name": "Amelia", "email": "amminer@pdx.edu", "date": "2023-11-04 06:16:59.114919+00:00", "message": "Hello Cloud Run!"}, {"name": "Amelia", "email": "amminer@pdx.edu", "date": "2023-11-16 19:32:47.745000+00:00", "message": "Hello Cloud Functions from Python Requests!"}]  
>>> []
```

### 13. Client-side guestbook application

- Examine the single-page application source code in frontend-src/index.html  
**This is the view in the MVP architecture, right?**



### 14. guestbook.js

- Examine frontend-src/static/guestbook.js.  
**And this is the presenter?**

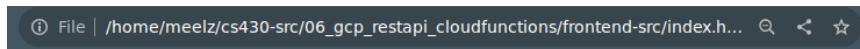


### 15. version #1: local file system

- Ensure that you've edited your API's base URL into guestbook.js. Load up index.html and view the developer console's network tab. Show the preflight request to the API that allows access, as well as the subsequent fetch request.

Name	Status	Type	Initiator	Size	Time	Waterfall
r@adaamminer@ada:~ammin	200	document	Other	844 B	16 ms	
er@ada:~amminer@ada:~am	200	stylesheet	index.html:5	566 B	75 ms	
miner@ada:~amminer@ada:	200	script	index.html:6	2.3 kB	76 ms	
~amamminer@ada:~ ammine	200	fetch	guestbook.js:68	359 B	2.12 s	
r@ada:~ amminer@ada:~ a	200	preflight	Preflight ↗	0 B	2.10 s	
mmamminer@ada:~ >ammine						
r@ada:~ >amminer@ada:~						

- Sign the guestbook with "Hello API Gateway from local SPA!".



**Guestbook**

Name:

Email:

Message:  
Hello API Gateway from local SPA! ↗

**Entries**

Amelia <amminer@pdx.edu>  
signed on 2023-11-13 01:26:36.732318+00:00  
Hello Kubernetes!

---

Amelia <amminer@pdx.edu>  
signed on 2023-10-30 01:05:00.939198+00:00  
Hello Cloud Shell!

---

Amelia <amminer@pdx.edu>  
signed on 2023-10-23 17:39:21.049839+00:00  
Hello Datastore

---

Amelia <amminer@pdx.edu>  
signed on 2023-11-13 02:32:23.019014+00:00  
Hello Cloud Build!

---

Amelia <amminer@pdx.edu>  
signed on 2023-11-03 00:43:25.874230+00:00  
Hello App Engine!

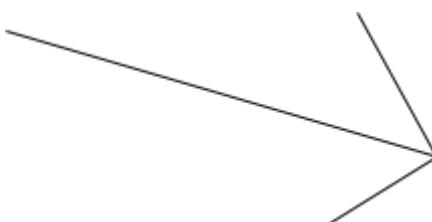
---

Amelia <amminer@pdx.edu>  
signed on 2023-10-30 01:35:11.596005+00:00  
Hello Compute Engine!

---

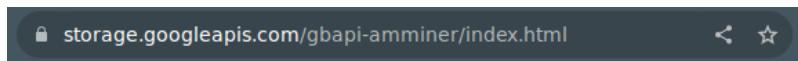
Amelia <amminer@pdx.edu>  
signed on 2023-11-16 22:58:16.340936+00:00  
Hello API Gateway from local SPA! ↗

---



## 16. version #2: google cloud storage bucket

- In cloud shell, cd into the frontend-src dir. Ensure that the correct base URL is in guestbook.js. Create a bucket (`gs://gbapi-amminer`) and assign a policy to it that makes it viewable by anyone (`gsutil iam ch allUsers:objectViewer gs://gbapi-amminer`). Copy the contents of the working directory in (`gsutil cp -r . gs://gbapi-amminer`). Open the public view of the frontend ([at `https://storage.googleapis.com/gbapi-amminer/index.html`](https://storage.googleapis.com/gbapi-amminer/index.html)) and sign it “Hello API Gateway from SPA in GCS!”.



## Guestbook

Name:

Email:

Message:

Hello API Gateway from SPA in GCS! 🚀 ☁️ 🚧

G

## Entries

Amelia <[amminer@pdx.edu](mailto:amminer@pdx.edu)>  
signed on 2023-11-13 01:26:36.732318+00:00  
Hello Kubernetes!

---

Amelia <[amminer@pdx.edu](mailto:amminer@pdx.edu)>  
signed on 2023-10-30 01:05:00.939198+00:00  
Hello Cloud Shell!

---

Amelia <[amminer@pdx.edu](mailto:amminer@pdx.edu)>  
signed on 2023-11-16 23:28:26.570650+00:00  
Hello API Gateway from SPA in GCS! 🚀 ☁️ 🚧

---

## 17. clean up

- delete the storage bucket, the API Gateway, and the Cloud Functions.



## IV. Lab 8.3g: OAuth2 Guestbook

### 1. oauth2 guestbook

- We'll use oauth2 + third party authentication to obtain user info for guestbook entries in this lab.



### 2. oauth2 authorization code flow

- One common flow is as follows:
  - i. App client makes a request to app server
  - ii. app server requests auth capability from ID provider which returns auth URL
  - iii. app server redirects app client to auth URL
  - iv. app client auths with ID Provider directly
  - v. ID Provider redirects app client to redirect URL w/ one-time code to send to app server
  - vi. app client accesses redirect URL/sends one-time code to app server
  - vii. app server exchanges one-time code with ID Provider to get app client's info



### 3. checkout code

- In your linux VM, cd into cs430-src/07\_oauth. Examine the source code; differences:
  - callback.py, logout.py, and oauth\_config.py facilitate communication,
  - index.py, index.html, sign.py, sign.html, \_\_init\_\_.py and model\_datastore.py have all been modified to render the new oauth-based guestbook entries.



### 4. oauth initiation code

- Examine oauth\_config.py. Notice that it reads in a few environment variables (CLIENT\_ID, CLIENT\_SECRET, REDIRECT\_CALLBACK) and defines a base URL for authorization and token retrieval from Google.
- Examine sign.py. Map the code to the steps described earlier, taking notice of where the implementation contains details not described in that high level overview..



### 5. identity provider interaction

- Note that granting consent to an application allows it to access the consented information until it is explicitly removed.



## 6. callback code

- Note that app.py implements a callback route for step 6 which uses the code in callback.py.



## 7. -

- Note that the oauth token that this flow produces is saved into a session cookie (the same one we originally checked to see if we needed to perform the flow).



## 8. signing page

- Examine the modified sign code and note the hidden inputs which handle the info we're now getting from our google using the auth token.



## 9. model code

- Examine the updated model code.

**The discrepancy between the name of the image URL field in the backend (profile) and the name we give the field when we render the page (picture) gives me pause.**

**Why do this?**



## 10. build and deploy the code

- Edit your email into the maintainer field of the Dockerfile. Use cloud build to build the container and publish it in gcr, naming it gcp\_oauth\_gb.

**Did I miss something? It looks like we need to do this part in the cloud shell...**

**Why did I perform the previous steps on my VM?**



- Deploy the container on cloud run:

```
gcloud run deploy gcp-oauth-gb \
    --image gcr.io/${GOOGLE_CLOUD_PROJECT}/gcp_oauth_gb \
    --service-account
guestbook@${GOOGLE_CLOUD_PROJECT}.iam.gserviceaccount.com \
    --region=us-west1 \
    --allow-unauthenticated
```

When the service has been deployed, make a note of its URL. We will need this URL to specify the Callback URL for Google's OAuth provider to redirect users to after authentication and authorization has been performed (remember to append /callback).

<https://gcp-oauth-gb-id4ecfzw6q-uw.a.run.app/callback>



## 11. set up identity provider

- Register the Guestbook app in gcloud under APIs & Services -> OAuth Consent Screen. Enable the openid scope which includes the non-sensitive scopes userinfo.email and userinfo.profile.



## 12. -

- Register the application (create an OAuth Client ID) to obtain a client\_id and client\_secret under APIs & Services -> Credentials. Specify the callback URL from earlier as the Authorized redirect URI. Keep your client ID and client secret around - we will use them to set the REDIRECT\_CALLBACK, CLIENT\_ID, and CLIENT\_SECRET env vars in the next step.

ID:

376082578160-g6nmdslhj44jvl45v55um20br9lr21mg.apps.googleusercontent.com

Secret:

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



## 13. update deployment

- Update the deployed guestbook app as follows:

```
gcloud run services update gcp-oauth-gb \
    --update-env-vars
    REDIRECT_CALLBACK=https://gcp-oauth-gb-id4ecfzw6q-uw.a.run.app/ca
    llback \
    --update-env-vars
    CLIENT_ID=376082578160-g6nmdslhj44jvl45v55um20br9lr21mg.apps.goog
    leusercontent.com \
    --update-env-vars
    CLIENT_SECRET=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 \
    --region=us-west1
```



## 14. visit the application

- In an Incognito Window, visit the site's URL. Bring up Developer Tools and navigate to the "Network" tab. Then, click on the link to sign the Guestbook. Examine the first two requests in the Network tab by clicking on their headers. Show these requests including the URL and returned HTTP status code.

The image contains two vertically stacked screenshots of the Google Chrome Developer Tools Network tab. Both screenshots show a list of network requests for the URL `https://accounts.google.com/v3/signin/identifier?oppara...`. The requests are listed in chronological order from top to bottom.

**Screenshot 1 (Top):** This screenshot shows the initial steps of the sign-in process. The first request is a `302` redirect to `https://gcp-oauth-gb-id4ecfzw6q-uw.a.run.app/sign`. Subsequent requests include JavaScript files (`m=_b_tp`, `m=_n73qwf,SCuOPb,I`), fonts (`KFOlCnqEu92Fr1Mm`, `4UaGrENHsxJIGDuG`), and other resources. The status bar on the right indicates the location is `FAB 82-01` and the user is `amminer@ada:~ >`.

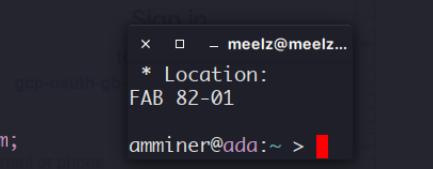
Request	Response
302	GET https://gcp-oauth-gb-id4ecfzw6q-uw.a.run.app/sign
200	GET https://gcp-oauth-gb-id4ecfzw6q-uw.a.run.app/sign
200	GET https://gcp-oauth-gb-id4ecfzw6q-uw.a.run.app/auth?response_type=code&client_id=376082578160-g6nmdsljh44jvl45v55um20br9lr21mg.apps.googleusercontent.com&auth-gb-id4ecfzw6q-uw.a.run.app/callback&scope=https://mail.googleapis.com/auth/userinfo.profile&state=OgyW&prompt=login
200	GET https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=376082578160-g6nmdsljh44jvl45v55um20br9lr21mg.apps.googleusercontent.com&auth-gb-id4ecfzw6q-uw.a.run.app/callback&scope=https://mail.googleapis.com/auth/userinfo.profile&state=OgyW&prompt=login

**Screenshot 2 (Bottom):** This screenshot shows the continuation of the sign-in process. It includes the same initial requests as Screenshot 1, followed by additional requests such as `m=_lDFwf,Rusgnf,Cts` and `m=RqjULd`. The status bar on the right indicates the location is `FAB 82-01` and the user is `amminer@ada:~ >`.

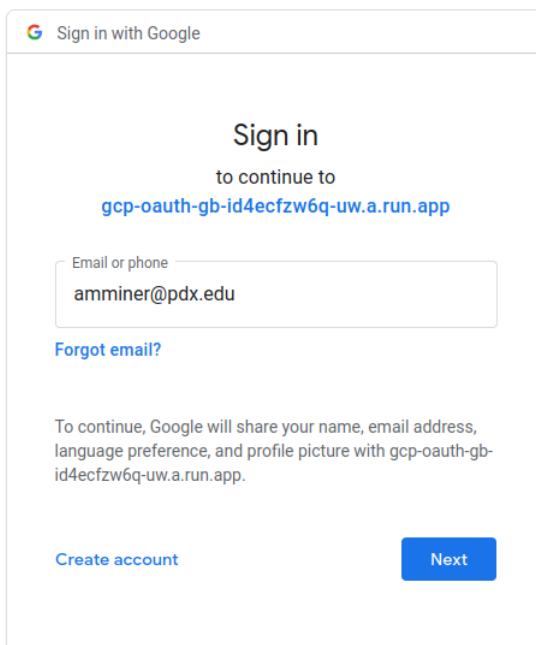
Request	Response
200	GET https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=376082578160-g6nmdsljh44jvl45v55um20br9lr21mg.apps.googleusercontent.com&auth-gb-id4ecfzw6q-uw.a.run.app/callback&scope=https://mail.googleapis.com/auth/userinfo.profile&state=OgyW&prompt=login
200	GET https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=376082578160-g6nmdsljh44jvl45v55um20br9lr21mg.apps.googleusercontent.com&auth-gb-id4ecfzw6q-uw.a.run.app/callback&scope=https://mail.googleapis.com/auth/userinfo.profile&state=OgyW&prompt=login
200	GET https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=376082578160-g6nmdsljh44jvl45v55um20br9lr21mg.apps.googleusercontent.com&auth-gb-id4ecfzw6q-uw.a.run.app/callback&scope=https://mail.googleapis.com/auth/userinfo.profile&state=OgyW&prompt=login

- Based on the descriptions of the source code, what lines of code in our app are responsible for the second request?  
**At the end of step 3 in the auth code flow, the app server redirects the app client to the authorization URL. The second request we're seeing is the client reaching out to that auth URL. In the source code, this redirect happens on line 28 of sign.py.**

```
7 class Sign(MethodView):
8     def get(self):
9         # If client has an OAuth2 token, use it to get their information and render
10        #   the signing page with it
11        if 'oauth_token' in session:
12            google = OAuth2Session(client_id, token=session['oauth_token'])
13            userinfo = google.get('https://www.googleapis.com/oauth2/v3/userinfo').json()
14            return render_template('sign.html', name=userinfo['name'], email=userinfo['email'],
15                                   userinfo['picture'])
16        else:
17            # Redirect to the identity provider and ask the identity provider to return the cl
18            #   back to /callback route with the code
19            google = OAuth2Session(client_id,
20                                   redirect_uri=redirect_callback,
21                                   scope='https://www.googleapis.com/auth/userinfo.email ' +
22                                         'https://www.googleapis.com/auth/userinfo.profile')
23            authorization_url, state = google.authorization_url(authorization_base_url, pr
24
25            # Identity provider returns URL and random "state" that must be echoed later
26            #   to prevent CSRF.
27            session['oauth_state'] = state
28            return redirect(authorization_url)
29
30    def post(self):
31        """
32            Accepts POST requests, and processes the form;
33            Redirect to index when completed.
34        """
35
```



- Take a screenshot of the permissions you (as a user) are granting the Guestbook access to.



- Grant the consent and auth with your PSU address. Click the callback request in the developer tools window/network tab. Show the headers including the entire callback URL and its return code. What is the URI for the location that the user is sent to by the callback?

The user is redirected to the /sign resource.

The screenshot shows the Network tab in the Developer Tools interface for the URL <https://gcp-oauth-gb-id4ecfzw6q-uw.a.run.app/sign>. A single network request is listed:

Status	Method	URL	Initiator	Type	Size
302	GET	<a href="#">callback?state=1tJz67cKMO7</a>	SetSID...	document	2.48 kB

The response details show a redirect to `/sign`:

```

Status: 302
Version: HTTP/3
Transferred: 2.48 kB (802 B size)
Referrer Policy: strict-origin-when-cross-origin
Request Priority: Highest
DNS Resolution: System
  
```

Response Headers (1.679 kB)

```

alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
content-length: 197
content-type: text/html; charset=utf-8
date: Sat, 18 Nov 2023 23:16:52 GMT
location: /sign
server: Google Frontend
set-cookie: session=eJy9VMmyqkgQ_RfWTy-DTHeHMiqFF0WB6uggmClnBUR48f694UZPy
171srly62Sek3V-Yo0_9JnX9X4fY58YR9nhg1P4MyKwsmiHlys3PchxeF4OtnYjz_T-6ala-zzJ-a
HYdx1f52xySf5rY8Lyd4LpkMb6HTVxHDG4_BwRdt86yBglekQLxHuji7hpdk6Bz7oxi71XAuGH
  
```

- Find the request within Developer Tools that fetches the embedded image and take a screenshot of its URL.

The screenshot shows the Network tab in the Developer Tools interface for the URL <https://gcp-oauth-gb-id4ecfzw6q-uw.a.run.app/sign>. Several network requests are listed, including one for an image:

Status	Method	URL	Initiator	Type	Size
302	GET	<a href="#">callback?state=1tJz67cKMO7</a>	SetSID...	document	2.48 kB
200	GET	<a href="#">favicon.ico</a>	Favicon...	image	5.4 kB
200	GET	<a href="#">sign</a>	document	text/html	1.07 kB
304	GET	<a href="#">style.css</a>	stylesheet	text/css	566 B
200	GET	<a href="#">ACg8ocJ42EDwmeFzo2PdiYZCQEhEcaV9UW--F5RLiZsuf2H=s96-c</a>	ACg8ocJ42EDwmeFzo...	image	1.33 kB
404	GET	<a href="#">favicon.ico</a>	Favicon...	image	207 B

The response details for the image request show the URL and headers:

```

Status: 200
Version: HTTP/2
Transferred: 1.33 kB (1.33 kB size)
Referrer Policy: no-referrer
Request Priority: Low
DNS Resolution: System
  
```

Response Headers (505 B)

```

access-control-allow-origin: *
  
```

- Sign the guestbook and log out.



## 15. secrets manager deployment

- Remove the CLIENT\_SECRET env var:

```
gcloud run services update gcp-oauth-gb \
--remove-env-vars CLIENT_SECRET \
--region=us-west1
```

Create the secret in Secrets Manager with the name client-secret:

```
echo -n "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" | gcloud secrets
create client-secret --data-file=-
```

Allow the Guestbook service account to access secrets stored in Secrets Manager:

```
gcloud projects add-iam-policy-binding ${GOOGLE_CLOUD_PROJECT} \
--member
serviceAccount:guestbook@${GOOGLE_CLOUD_PROJECT}.iam.gserviceac
ount.com \
--role roles/secretmanager.secretAccessor
```

Finally, update the deployment to point the CLIENT\_SECRET environment variable to client-secret stored in Secrets Manager.

```
gcloud run services update gcp-oauth-gb \
--update-secrets=CLIENT_SECRET=client-secret:latest \
--region=us-west1
```

Sign in again using a different account and add another entry to the Guestbook.

Take a screenshot showing multiple authenticated accounts have been able to sign the Guestbook.

The screenshot shows a web browser window with the URL <https://gcp-oauth-gb-id4ecfzw6q-uw.a.run.app>. The page title is "Guestbook". It features a "Sign here | Logout" button and a "Entries" section. Two entries are listed:

**Amelia Miner <amminer@pdx.edu>**  
on 2023-11-18 23:21:35.766954+00:00  
Hello from OAuth2!

---

**Amelia Miner <ameliamminer@gmail.com>**  
on 2023-11-19 00:50:51.475402+00:00  
Hello from my personal Google account!

---

## 16. removing access

- Visit <https://myaccount.google.com/permissions>. Go into the details for Guestbook's permissions and show them, then remove them from your account.

The screenshot shows the Google My Account permissions page for the 'Guestbook' service. The URL in the address bar is https://myaccount.google.com/u/1/connections/details/AbuSVb0iwtVLRzBhkzLbyVcGAO-uRMO\_w2pbUaG2gf4XfvUHrrnmwHi75gV5bwIzn2Yg5cY9B77nNjScS/. The main content area displays a cartoon illustration of a person standing next to a tree and a flower, with a speech bubble above it. Below the illustration, there is a section titled 'How Google helps you sign in to Guestbook' with two options: 'Sign in with Google' and 'See your profile info'. A 'See details' button is located at the bottom right of this section. To the right of the main content, a sidebar shows the user's email (amminer@pdx.edu) and account status (Managed by pdx.edu). It also displays a profile picture with a letter 'A' and a welcome message 'Hi, Amelia!'. Below this, there is a 'Recommended actions' section and a list of accounts under 'Hide more accounts'. The list includes two entries for 'Amelia Miner' (one associated with a Gmail account and one with an Apple ID), and a link to 'Add another account'.

## 17. clean up

- Remove the cloud run service and delete the container image. Remove the OAuth client credentials.



## V. Lab 8.4g: Firebase

### 1. firebase web application

- Firebase is a backend as a service platform with real-time database support. Modern applications shift functionality to the client with the backend providing only the model portion of the web app triad. We'll demonstrate this pattern in this lab.



### 2. project setup

- Create a new gcloud project named firebase-amminer and associate it with the course's billing account. Then create a new firebase project in the separate management console at <https://console.firebaseio.google.com/> and associate it with the new gcloud project of the same name. Disable analytics.

**The name got a little funky since I accidentally created this project on my personal google account first and deleting a project takes a long time.**



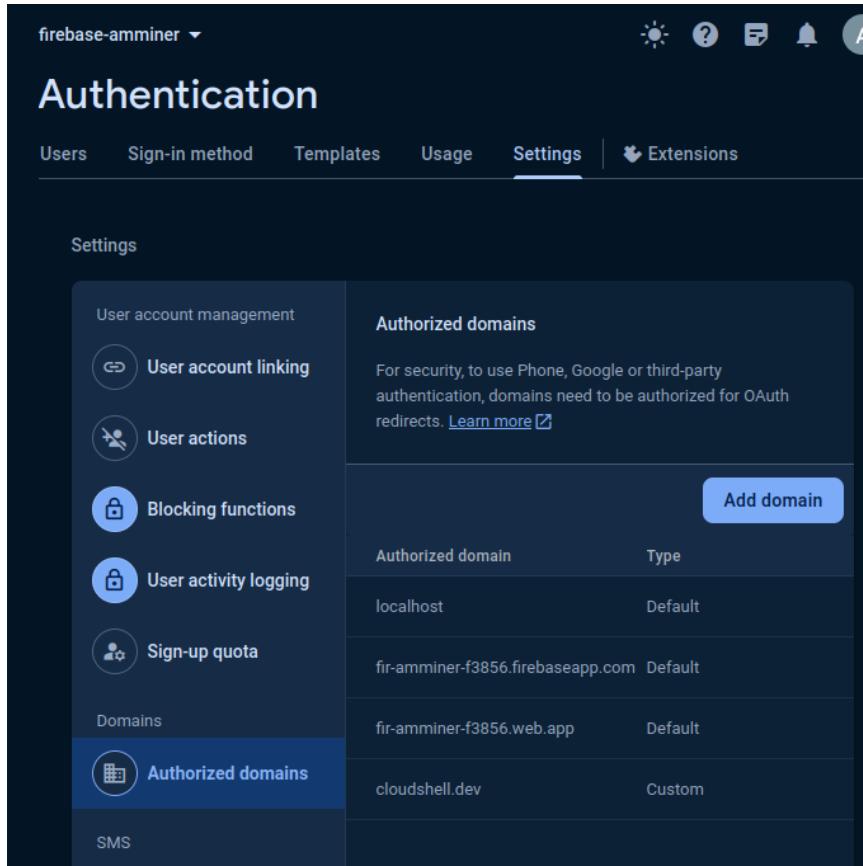
### 3. application setup

- Register a web app with the project. Name it Friendly Chat. Check the Firebase Hosting box. Skip the remaining steps.



## 4. authentication setup

- Enable google logins from the firebase console -> Build -> Authentication.  
Add cloudshell.dev to the list of authorized domains for the project so we can access the application from cloud shell. What other domains have access to the project by default?  
**localhost, fir-amminer-f3856.firebaseio.com, and fir-amminer-f3856.web.app**



The screenshot shows the 'Authentication' section of the Firebase console. The 'Settings' tab is selected. On the left, a sidebar lists various settings: User account management, User account linking, User actions, Blocking functions, User activity logging, Sign-up quota, Domains, Authorized domains (which is highlighted in blue), and SMS. The main area is titled 'Authorized domains'. It contains a note about OAuth redirects and a 'Learn more' link. A large 'Add domain' button is visible. Below it is a table listing the current authorized domains:

Authorized domain	Type
localhost	Default
fir-amminer-f3856.firebaseio.com	Default
fir-amminer-f3856.web.app	Default
cloudshell.dev	Custom

## 5. database setup

- Build -> Firestore Database -> Create Database. Enable “start in test mode”.  


## 6. storage setup

- Build -> Storage -> Get Started. Use test mode.  


## 7. cli setup

- Back in the gcloud console, clone the lab repo and install the firebase CLI with NPM:

```
git clone https://github.com/firebase/codelab-friendlychat-web
npm -g install firebase-tools
```

Then log out and back in to force a reauthorization and obtain an OAuth token:

```
firebase logout
```

```
firebase login --no-localhost
```

Follow the instructions. When you have an OAuth code, paste it into the cloud shell to complete the login. cd into codelab-friendlychat/web-start and connect the Firebase CLI to the Firebase Project from earlier with `firebase use --add`. Give the project the alias "default".



## 8. bundling with webpack

- Bring up public/index.html in the cloud shell editor. Examine it. The html is a basic skeleton that will be filled out by webpack via public/scripts/main.js, generated based on the code in the src dir.

Bring up a new cloud shell terminal, change into web-start, and run `npm install` & `npm run start`. Leave this terminal open for the remainder of the lab. Show the first 10 lines in the file that webpack produces.

The screenshot shows the Google Cloud Platform Cloud Shell interface. The title bar says "Google Cloud" and "firebase-amminer". The main area is titled "CLOUD SHELL Editor" and shows a file named "main.js". The file content is as follows:

```
JS main.js
=====
web-start > public > scripts > JS main.js > ...
1  /*
2   * ATTENTION: An "eval-source-map" devtool has been used.
3   * This devtool is neither made for production nor for readable output files.
4   * It uses "eval()" calls to create a separate source file with attached SourceMaps
5   * in the browser devtools.
6   * If you are trying to read the output file, select a different devtool (https://
7   * webpack.js.org/configuration/devtool/)
8   * or disable the default devtool with "devtool: false".
9   * If you are looking for production-ready output files, see mode: "production"
10  (https://webpack.js.org/configuration/mode/).
11 */
12 /**
13  ./node_modules/@firebase/auth/dist/esm2017/index-e3d5d3f4.js":
14 /*!***** ./.node_modules/@firebase/auth/dist/esm2017/index-e3d5d3f4.js ***!
15 | !*** ./.node_modules/@firebase/auth/dist/esm2017/index-e3d5d3f4.js ***
16 | \***** */
17 /**
18  (_unused_webpack_module, _webpack_exports_, _webpack_require_) => {
19    eval("/* unused harmony exports $, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q,
R, S, T, U, V, W, X, Y, Z, _, a, a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, aA, aB, aC,
```

## 9. configure firebase within application

- In the firebase console, go to project settings and scroll down to view the web app from earlier. Select “config” in the SDK setup section and copy the value of the const, then paste it into web-start/src/firebase-config.js. webpack will automatically recompile the application.



## 10. initialize firebase within application

- In src/index.js, go to the bottom of the file and under //TODO 0, add `initializeApp(firebaseAppConfig);`



## 11. test application

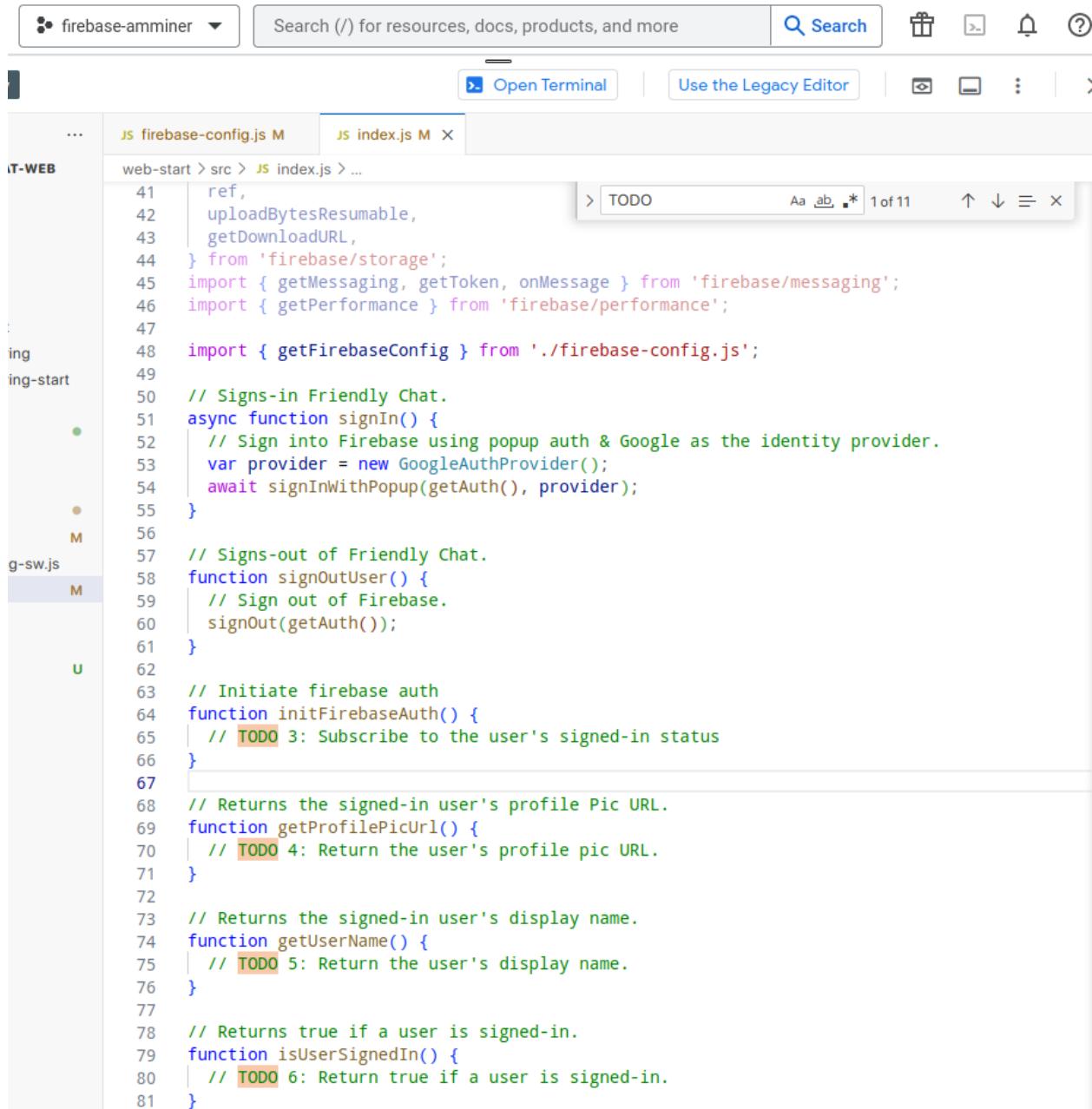
- In the cloud shell terminal, run `firebase serve --only hosting` from the web-start dir. Preview the app. Note that the URL contains the domain we enabled earlier. Go back to the cloud shell and `ctrl+c` to exit.  
**It may be worth mentioning here that we have to kill the `npm run start` command from earlier to proceed.**



## 12. add authentication

- View the application code in src/index.js in the cloud shell editor. What missing functions deal with user auth?

I forgot to take screenshots until after I edited the code from below in, apologies.  
`signIn`, `signOutUser`, `initFirebaseAuth`, `isUserSignedIn`.



The screenshot shows the Google Cloud Platform Cloud Shell interface. At the top, there's a navigation bar with a dropdown for the project ('firebase-amminner'), a search bar ('Search (/) for resources, docs, products, and more'), and various icons for notifications and help. Below the navigation bar is a toolbar with buttons for 'Open Terminal' and 'Use the Legacy Editor', along with other standard file operations like Open, Save, and Delete.

The main area is a code editor displaying the file 'index.js'. The code is written in JavaScript and includes imports for Firebase services like storage, messaging, performance, and auth. It defines several functions: `signIn` (using GoogleAuthProvider), `signOutUser` (signing out of Firebase), `initFirebaseAuth` (which has a TODO comment for subscribing to user signed-in status), `getProfilePicUrl` (which has a TODO comment for returning the user's profile pic URL), and `getUserName` (which has a TODO comment for returning the user's display name). There are also two additional TODO comments in the code.

```
... JS firebase-config.js M JS index.js M X
...
41 | ref,
42 | uploadBytesResumable,
43 | getDownloadURL,
44 | } from 'firebase/storage';
45 | import { getMessaging, getToken, onMessage } from 'firebase/messaging';
46 | import { getPerformance } from 'firebase/performance';
47 |
48 | import { getFirebaseConfig } from './firebase-config.js';
49 |
50 | // Signs-in Friendly Chat.
51 | async function signIn() {
52 |   // Sign into Firebase using popup auth & Google as the identity provider.
53 |   var provider = new GoogleAuthProvider();
54 |   await signInWithPopup(getAuth(), provider);
55 | }
56 |
57 | // Signs-out of Friendly Chat.
58 | function signOutUser() {
59 |   // Sign out of Firebase.
60 |   signOut(getAuth());
61 | }
62 |
63 | // Initiate firebase auth
64 | function initFirebaseAuth() {
65 |   // TODO 3: Subscribe to the user's signed-in status
66 | }
67 |
68 | // Returns the signed-in user's profile Pic URL.
69 | function getProfilePicUrl() {
70 |   // TODO 4: Return the user's profile pic URL.
71 | }
72 |
73 | // Returns the signed-in user's display name.
74 | function getUserName() {
75 |   // TODO 5: Return the user's display name.
76 | }
77 |
78 | // Returns true if a user is signed-in.
79 | function isUserSignedIn() {
80 |   // TODO 6: Return true if a user is signed-in.
81 | }
```

- What missing functions deal with sending and receiving messages?  
**saveMessage, loadMessages, saveImageMessage.**

firebase-amminier ▾

Search (/) for resources, docs, products, and more

Open Terminal

Use the Legacy Editor

... JS firebase-config.js M JS index.js M X

T-WEB

ng ng-start

● M g-sw.js M M u

```
web-start > src > JS index.js > ...
  59  // Sign out of Firebase.
  60  signOut(getAuth());
  61 }
  62
  63 // Initiate firebase auth
  64 function initFirebaseAuth() {
  | // TODO 3: Subscribe to the user's signed-in status
  65 }
  66
  67
  68 // Returns the signed-in user's profile Pic URL.
  69 function getProfilePicUrl() {
  | // TODO 4: Return the user's profile pic URL.
  70 }
  71
  72
  73 // Returns the signed-in user's display name.
  74 function getUserName() {
  | // TODO 5: Return the user's display name.
  75 }
  76
  77
  78 // Returns true if a user is signed-in.
  79 function isUserSignedIn() {
  | // TODO 6: Return true if a user is signed-in.
  80 }
  81
  82
  83 // Saves a new message on the Cloud Firestore.
  84 async function saveMessage(messageText) {
  | // TODO 7: Push a new message to Cloud Firestore.
  85 }
  86
  87
  88 // Loads chat messages history and listens for upcoming ones.
  89 function loadMessages() {
  | // TODO 8: Load and listen for new messages.
  90 }
  91
  92
  93 // Saves a new message containing an image in Firebase.
  94 // This first saves the image in Firebase storage.
  95 async function saveImageMessage(file) {
  | // TODO 9: Posts a new image as a message.
  96 }
  97 }
```

- Use the code below to implement signIn for when the user clicks the "Sign in with Google" button.

```
// Sign into Firebase using popup auth & Google as the identity provider.  
var provider = new GoogleAuthProvider();  
await signInWithPopup(getAuth(), provider);
```

Then use this code for the signOut function:

```
// Sign out of Firebase.  
signOut(getAuth());
```



## 13. update UI

- Add the following to init.FirebaseAuth:

```
// Subscribe to the user's signed-in status  
onAuthStateChanged(getAuth(), authStateObserver);
```

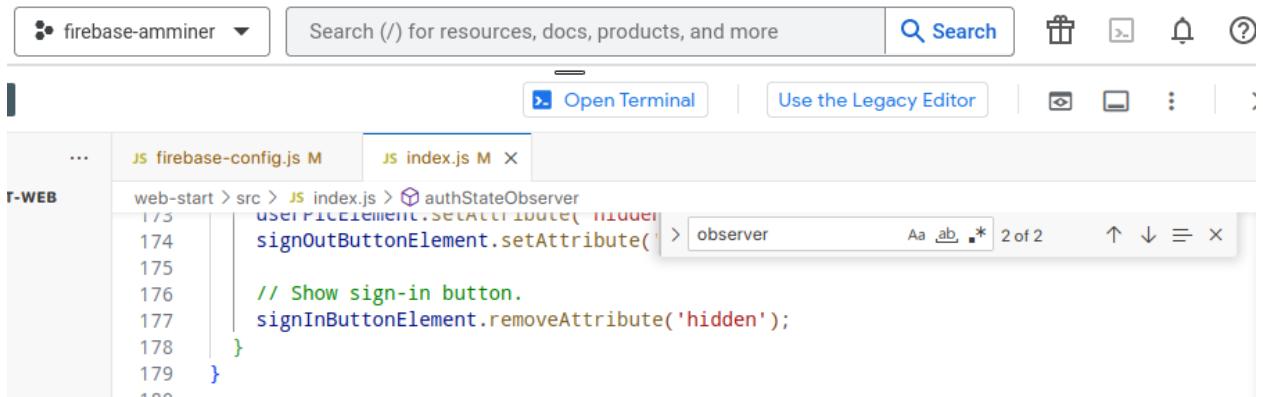
Check out the authStateObserver function which we're registering as the callback for when auth state changes. What are the names of the elements that are hidden when the user is signed out?

**userNameElement, userPicElement, signOutButtonElement.**

The screenshot shows a code editor interface with the following details:

- Project:** firebase-amminer
- File:** index.js
- Line 146:** `// Triggers when the auth state change for instance when the user signs-in or signs-out.`
- Line 147:** `function authStateObserver(user) {`
- Line 148:** `if (user) {`
- Line 149:**  `// User is signed in!`
- Line 150:**  `// Get the signed-in user's profile pic and name.`
- Line 151:**  `var profilePicUrl = getProfilePicUrl();`
- Line 152:**  `var userName =.getUserName();`
- Line 153:**  `// Set the user's profile pic and name.`
- Line 154:**  `userPicElement.style.backgroundImage =`
- Line 155:**  `| 'url(' + addSizeToGoogleProfilePic(profilePicUrl) + ')';`
- Line 156:**  `userNameElement.textContent = userName;`
- Line 157:**  `// Show user's profile and sign-out button.`
- Line 158:**  `userNameElement.removeAttribute('hidden');`
- Line 159:**  `userPicElement.removeAttribute('hidden');`
- Line 160:**  `signOutButtonElement.removeAttribute('hidden');`
- Line 161:**  `// Hide sign-in button.`
- Line 162:**  `signInButtonElement.setAttribute('hidden', 'true');`
- Line 163:**  `// We save the Firebase Messaging Device token and enable notifications.`
- Line 164:**  `saveMessagingDeviceToken();`
- Line 165:** `} else {`
- Line 166:**  `// User is signed out!`
- Line 167:**  `// Hide user's profile and sign-out button.`
- Line 168:**  `userNameElement.setAttribute('hidden', 'true');`
- Line 169:**  `userPicElement.setAttribute('hidden', 'true');`
- Line 170:**  `signOutButtonElement.setAttribute('hidden', 'true');`
- Line 171:** `}`
- Line 172:** `}`
- Line 173:** `}`
- Line 174:**

- What is the name of the element that is not hidden when the user is signed out?  
**signInButtonElement**



```
... JS firebase-config.js M JS index.js M X
F-WEB web-start > src > JS index.js > authStateObserver
173   userInputElement.setAttribute('hidden');
174   signOutButtonElement.setAttribute('hidden');
175
176   // Show sign-in button.
177   signInButtonElement.removeAttribute('hidden');
178 }
179 }
```

- Fill in the following code:

- `getProfilePicUrl`:  

```
return getAuth().currentUser.photoURL ||
  '/images/profile_placeholder.png';
```
- `getUserName`:  

```
return getAuth().currentUser.displayName;
```
- `isUserSignedIn`:  

```
return !!getAuth().currentUser;
```



## 14. test application with authentication

- Launch the local hosting emulator again and bring up the site. You should see a message that says you must sign in first. Sign in with your PSU account. Send a message again; no error should appear, but nothing should happen since we haven't implemented messages yet.



## 15. add text messaging

- Use the following code for saveMessage:

```
// Add a new message entry to the database.  
try {  
    await addDoc(collection(getFirestore(), 'messages'), {  
        name: getUserName(),  
        text: messageText,  
        profilePicUrl: getProfilePicUrl(),  
        timestamp: serverTimestamp()  
    });  
}  
catch(error) {  
    console.error('Error writing new message to Firebase Database', error);  
}
```

And then for loadMessages:

```
// Create the query to load the last 12 messages and listen for new ones.  
const recentMessagesQuery = query(collection(getFirestore(), 'messages'),  
orderBy('timestamp', 'desc'), limit(12));  
// Start listening to the query.  
onSnapshot(recentMessagesQuery, function(snapshot) {  
    snapshot.docChanges().forEach(function(change) {  
        if (change.type === 'removed') {  
            deleteMessage(change.doc.id);  
        } else {  
            var message = change.doc.data();  
            displayMessage(change.doc.id, message.timestamp, message.name,  
                           message.text, message.profilePicUrl, message.imageUrl);  
        }  
    });  
});  
✓
```

## 16. test application with text messaging

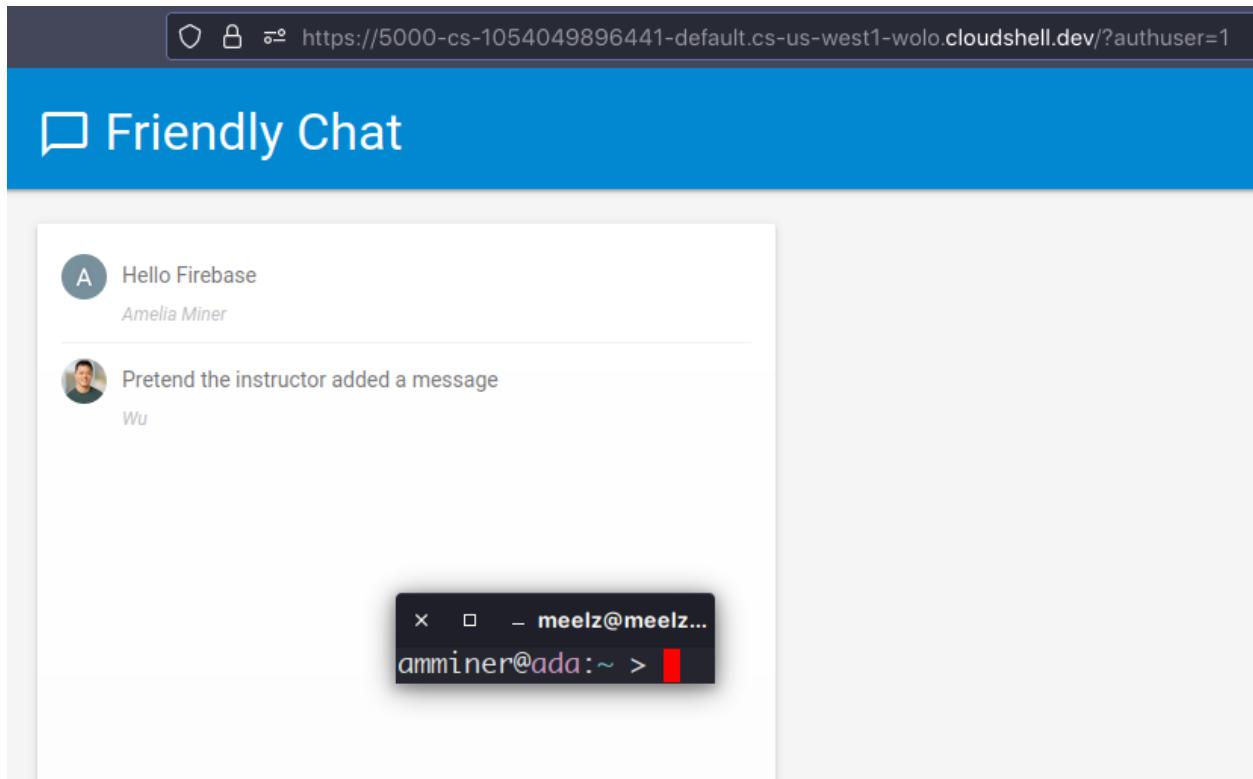
- Launch the hosting emulator again. Visit the site, sign in, and send a message. In the firebase console, bring up the database, expand the messages collection, and find the document containing the message you just sent. Show the message and its fields.

The screenshot shows the Firebase Cloud Firestore interface. At the top, there are two cards: one about budget management and another about app check. Below the cards, there's a dropdown for 'Default' and buttons for 'Panel view' and 'Query builder'. The main area shows a navigation path: Home > messages > gG9eB0VJ0vH7uXs7xGTv.. The left sidebar has a 'messages' collection and a document named 'gG9eB0VJ0vH7uXs7xGTv'. The right panel displays the fields of the selected document:

name	value
name	"Amelia Miner"
profilePicUrl	"https://lh3.googleusercontent.com/a/ACgE...
text	"Hello Firebase"
timestamp	November 19, 2023 at 9:16:15 AM UTC-8

## 17. manual message insertion

- Under the messages collection click “add document” and click “auto-id”. Set the name field to Wu, profilePicUrl to <https://www.pdx.edu/computer-science/sites/computerscience.web.wdt.pdx.edu/files/2020-06/Wu-chang.jpg>, text to “Pretend the instructor added a message”, and timestamp to the current date and time. Save the document. Show the message in the Friendly Chat app UI.



## 18. add image messaging

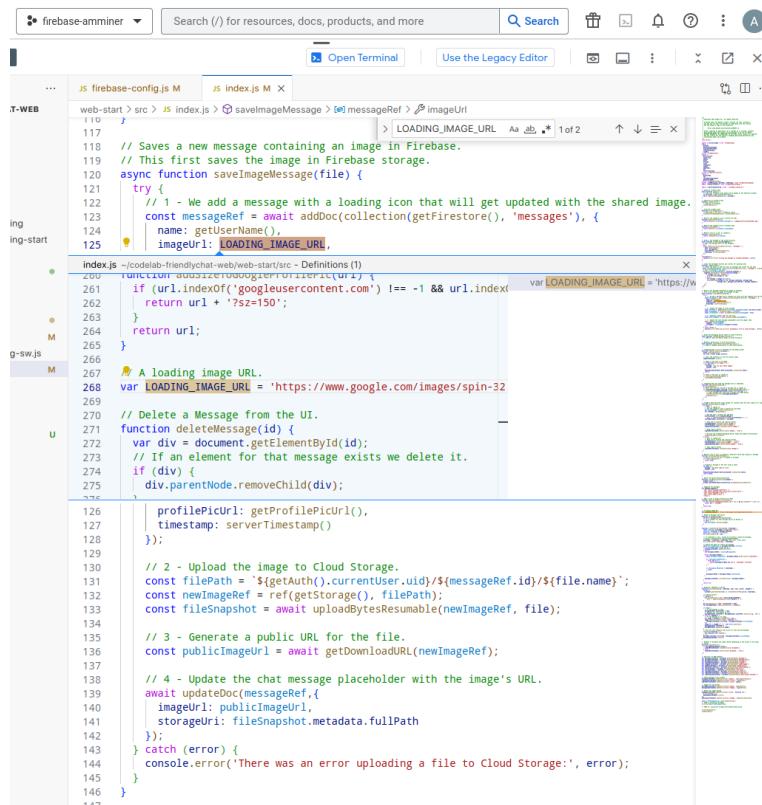
- Add the following to the saveImageMessage function:

```
try {
    // 1 - We add a message with a loading icon that will get updated with the shared
    image.

    const messageRef = await addDoc(collection(getFirestore(), 'messages'), {
        name: getUserName(),
        imageUrl: LOADING_IMAGE_URL,
        profilePicUrl: getProfilePicUrl(),
        timestamp: serverTimestamp()
    });

    // 2 - Upload the image to Cloud Storage.
    const filePath = `${getAuth().currentUser.uid}/${messageRef.id}/${file.name}`;
    const newImageRef = ref(getStorage(), filePath);
    const fileSnapshot = await uploadBytesResumable(newImageRef, file);
    // 3 - Generate a public URL for the file.
    const publicImageUrl = await getDownloadURL(newImageRef);
    // 4 - Update the chat message placeholder with the image's URL.
    await updateDoc(messageRef, {
        imageUrl: publicImageUrl,
        storageUri: fileSnapshot.metadata.fullPath
    });
} catch (error) {
    console.error('There was an error uploading a file to Cloud Storage:', error);
}
```

What is the URL of the image that is first shown in the UI as the message is loading?  
<https://www.google.com/images/spin-32.gif?a>



```
... JS firebase-config.js M JS index.js M X
J-WEB web-start > src > JS index.js > saveImageMessage > messageRef > imageUrl
116 // Saves a new message containing an image in Firebase.
117 // This first saves the image in Firebase storage.
118 // 1 - We add a message with a loading icon that will get updated with the shared image.
119 // 2 - Upload the image to Cloud Storage.
120 // 3 - Generate a public URL for the file.
121 // 4 - Update the chat message placeholder with the image's URL.
122
123 const messageRef = await addDoc(collection(getFirestore(), 'messages'), {
124     name: getUserName(),
125     imageUrl: LOADING_IMAGE_URL,
126
127     // A loading image URL.
128     var LOADING_IMAGE_URL = 'https://www.google.com/images/spin-32.
129
130     // Delete a Message from the UI.
131     function deleteMessage(id) {
132         var div = document.getElementById(id);
133         if (div) {
134             div.parentNode.removeChild(div);
135
136             // 1 - We add a message with a loading icon that will get updated with the shared image.
137             const messageRef = await addDoc(collection(getFirestore(), 'messages'), {
138                 name: getUserName(),
139                 imageUrl: LOADING_IMAGE_URL,
140                 profilePicUrl: getProfilePicUrl(),
141                 timestamp: serverTimestamp()
142             });
143             // 2 - Upload the image to Cloud Storage.
144             const filePath = `${getAuth().currentUser.uid}/${messageRef.id}/${file.name}`;
145             const newImageRef = ref(getStorage(), filePath);
146             const fileSnapshot = await uploadBytesResumable(newImageRef, file);
147             // 3 - Generate a public URL for the file.
148             const publicImageUrl = await getDownloadURL(newImageRef);
149             // 4 - Update the chat message placeholder with the image's URL.
150             await updateDoc(messageRef, {
151                 imageUrl: publicImageUrl,
152                 storageUri: fileSnapshot.metadata.fullPath
153             });
154         } catch (error) {
155             console.error('There was an error uploading a file to Cloud Storage:', error);
156         }
157     }
158 }
```

## 19. Test application with image messaging

- Serve the app locally again from cloud shell. Sign in and send an image. In the firebase console, bring up the database. Expand the messages collection and find the document containing the message you just sent. How do the fields in an image document differ from those in a text document?

**An image document is the same as a text document except that the text field is absent and two new fields, imageUrl and storageUri, replace it.**

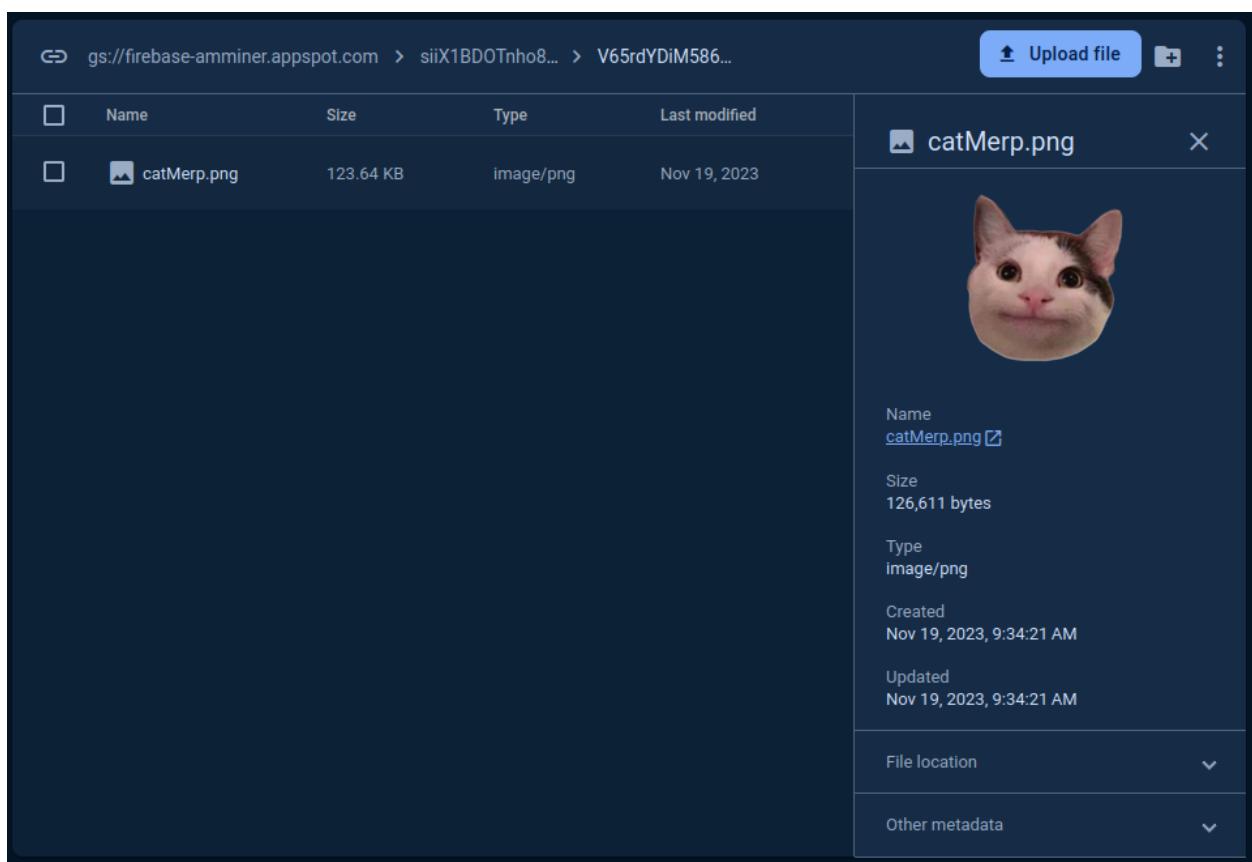
- What URL and storage location can the image be found at?

<https://firebasestorage.googleapis.com/v0/b/firebase-amminer.appspot.com/o/siiX1BDOTnho8FQXdPLo7Z7q8Hu1%2FV65rdYDiM586MD7L6TMu%2FcatMerp.png?alt=media&token=f81705f5-13ea-4c8b-9acb-34ff9dd9a69c>

and

**siiX1BDOTnho8FQXdPLo7Z7q8Hu1/V65rdYDiM586MD7L6TMu/catMerp.png**

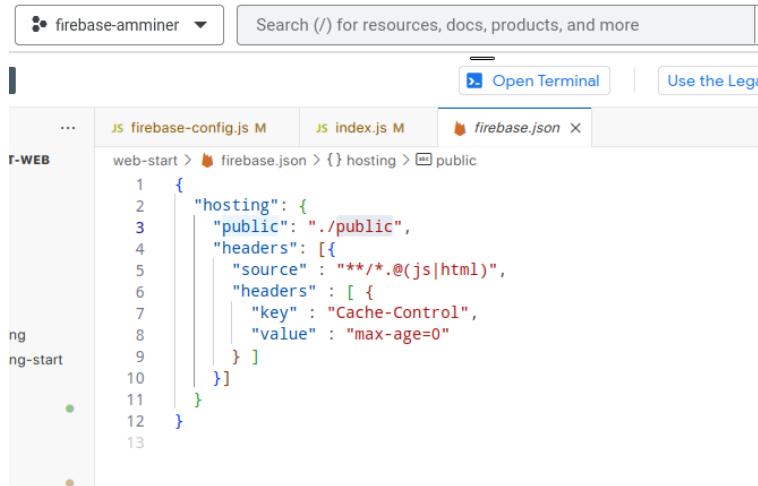
- Visit the storage tab of the console. Take a screenshot of the image in the storage bucket.



## 20. deploy application

- Firebase manages actual deployment using the firebase.json file. What directory is the application going to be served from?

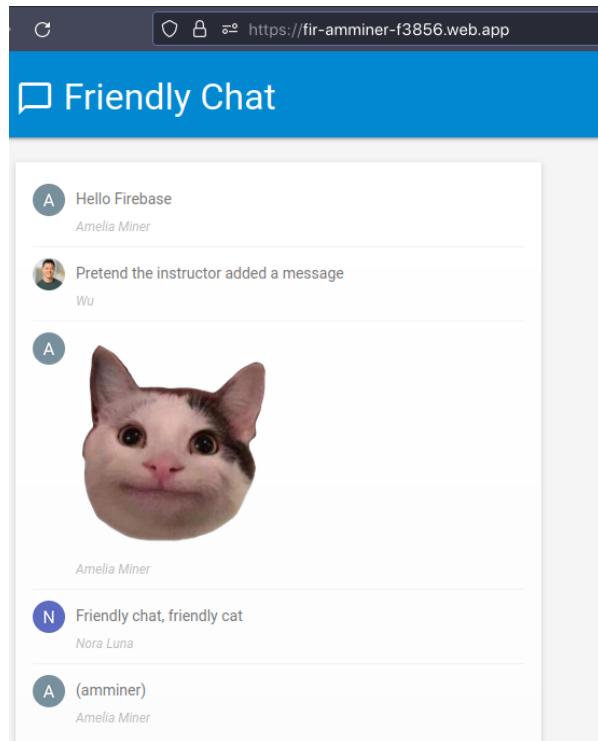
**public**



The screenshot shows the Firebase console interface. At the top, there's a search bar with placeholder text "Search (/) for resources, docs, products, and more". Below the search bar are two buttons: "Open Terminal" and "Use the Logs". The main area displays a file tree under the "T-WEB" tab. The "firebase.json" file is selected, indicated by a blue border. The code in the editor is:

```
1  {
2   "hosting": {
3     "public": "./public",
4     "headers": [
5       {
6         "source" : "**/*.{js|html}",
7         "headers" : [ {
8           "key" : "Cache-Control",
9           "value" : "max-age=0"
10        }]
11      }
12    }
13 }
```

- Deploy with `firebase deploy --except functions`. Visit the app and send the URL to someone you know from the class, the TA, or the instructor. Take a screenshot of the message they send including the URL.



## 21. Clean Up

- Delete the firebase project. 