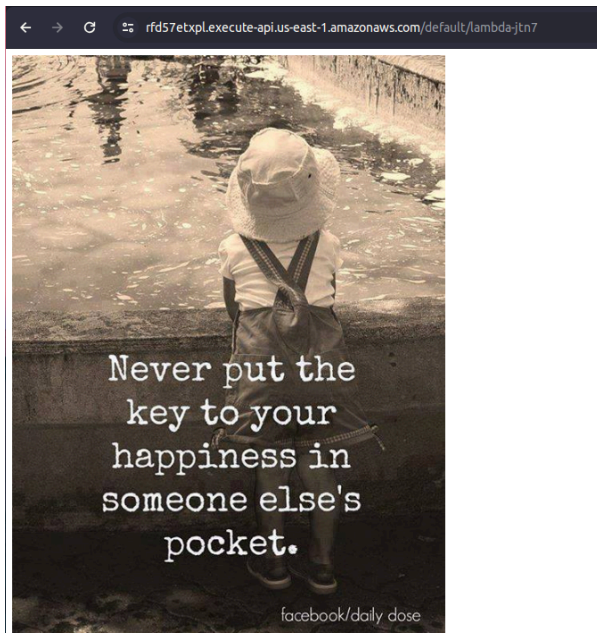


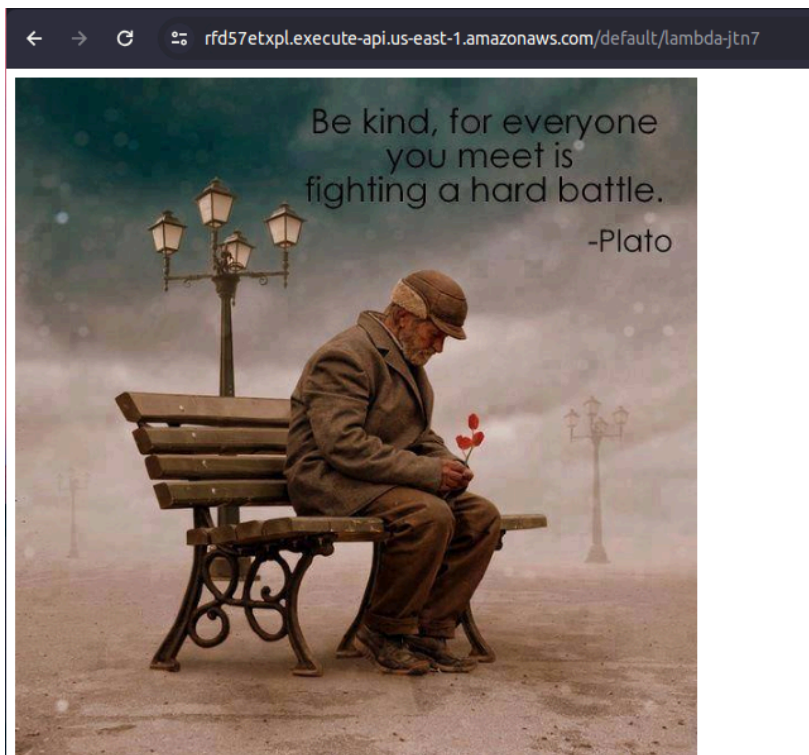
8.1a

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.

Take a screenshot of the resulting page including the URL bar.



Click "Reload" in the browser and take another screenshot showing the image has changed:



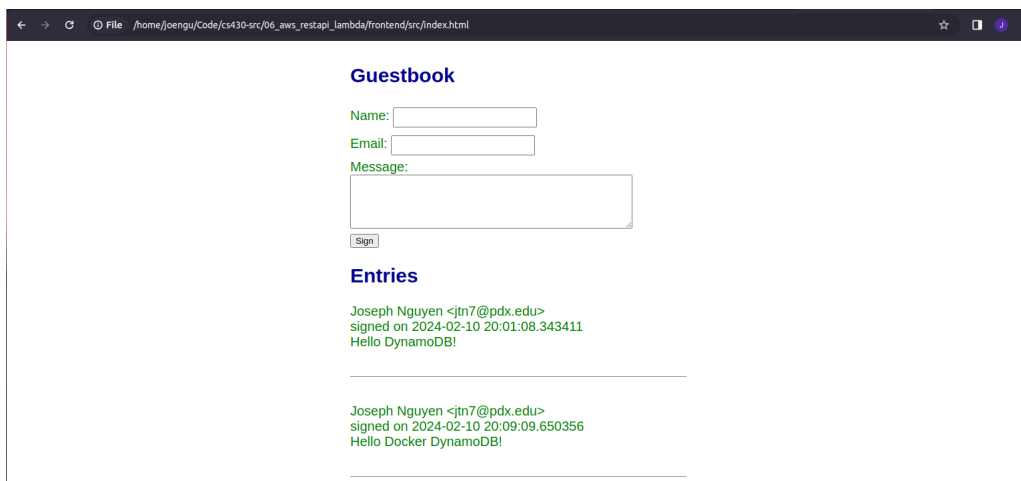
Use curl on your Linux VM to access the API endpoint and show the results. Take a screenshot for your lab notebook.



8.2a

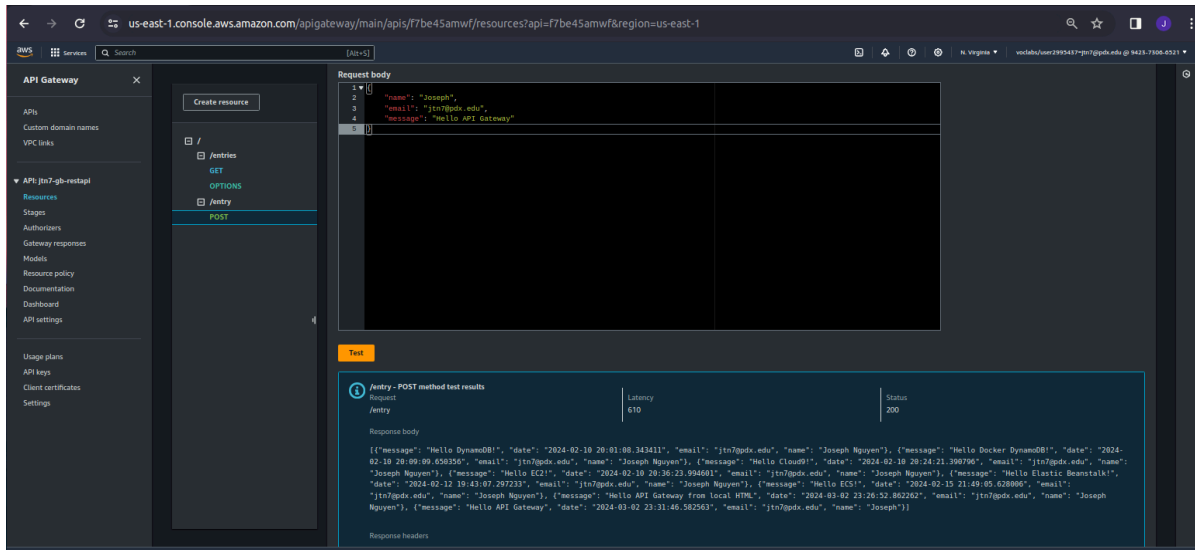
Bring up a browser and view the main HTML for the guestbook at `06_aws_restapi_lambda/frontend/src/index.html`

Take a screenshot that shows that you can view the entries in the backend database.



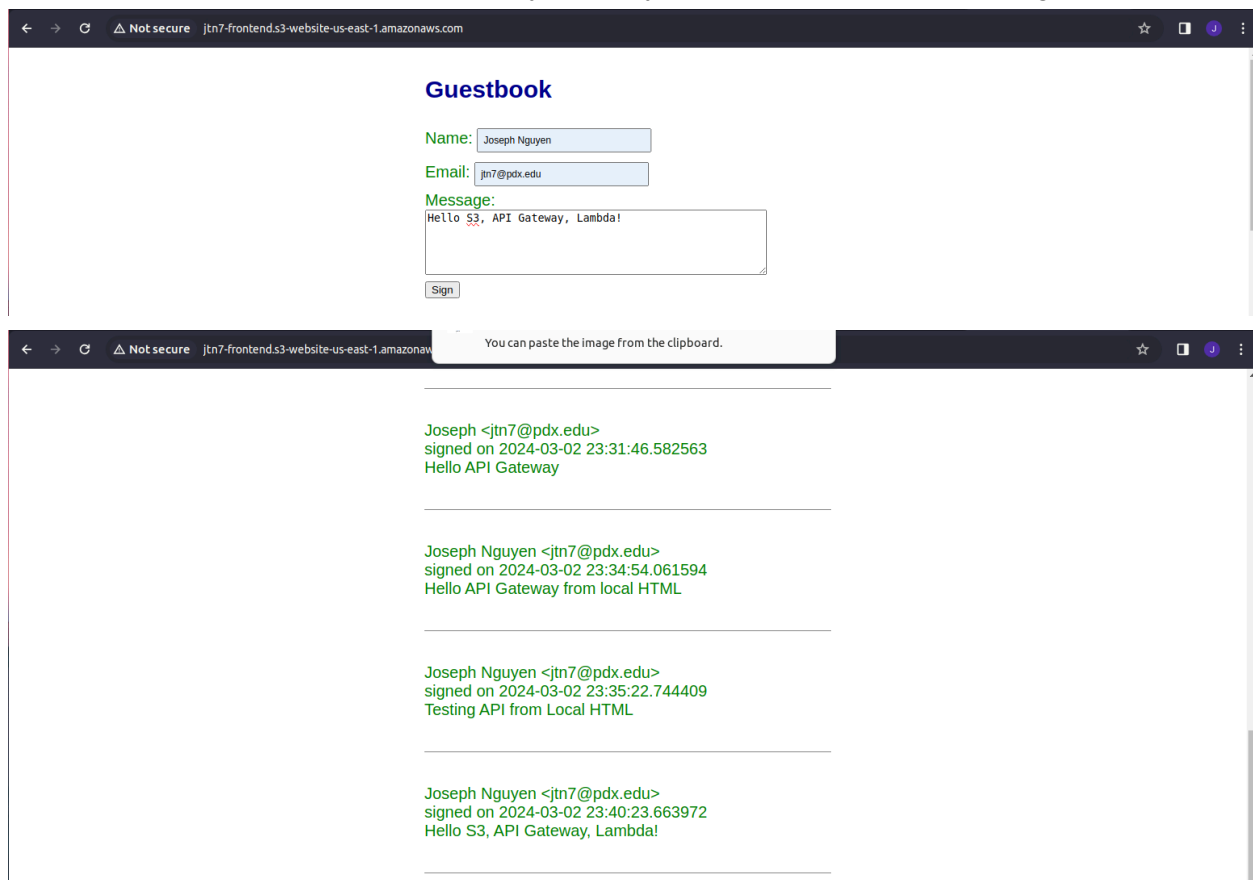
Click on TEST, then view the response. The API returns the entire contents of the guestbook including the entry you have added.

Take a screenshot showing that the submission worked.



Copy the URL and open the website in Chrome. Add an entry with the message "Hello S3, API Gateway and Lambda!".

Take a screenshot as before that shows your entry and the static website hosting URL.



8.2g

Include the hostname for the API gateway in your lab notebook.

gbapigw-d5uz5ovh.uc.gateway.dev

Take a screenshot of the loop and its output

```
>>> response = resp.json()
>>> properties = ['name', 'email', 'date', 'message']

>>> for property in properties:
...     print(response[0][property])
...
Joseph Nguyen
jtn7@pdx.edu
2024-02-23 21:40:59.669138+00:00
Hello Cloud Build!
```

Print the response status, the response headers, and the response text that indicates a successful insertion within the Python interpreter as before.

Take a screenshot of the output for your lab notebook

```
>>> resp = requests.post('https://gbapigw-d5uz5ovh.uc.gateway.dev/entry', json=mydict)
>>> print(resp.status_code, resp.headers, resp.text)
200 {'content-type': 'application/json', 'access-control-allow-origin': '*', 'function-executio
n-id': '4nvxuo2h6nqj', 'x-cloud-trace-context': '4e7daaf789089e549ec01702aac9185;o=1', 'alt-sv
c': 'h3=:443'; ma=2592000,h3-29=:443'; ma=2592000', 'Date': 'Thu, 29 Feb 2024 18:36:04 GMT',
'Server': 'Google Frontend', 'Content-Length': '1059'} [{"name": "Joseph Nguyen", "email": "jtn
7@pdx.edu", "date": "2024-02-23 21:40:59.669138+00:00", "message": "Hello Cloud Build!"}, {"nam
e": "Joseph Nguyen", "email": "jtn7@pdx.edu", "date": "2024-02-29 18:36:04.027490+00:00", "mess
age": "Hello Cloud Functions from Python Requests!"}, {"name": "Joseph Nguyen", "email": "jtn7@
pdx.edu", "date": "2024-02-11 06:11:11.537376+00:00", "message": "Hello Datastore!"}, {"name":
"Joseph Nguyen", "email": "jtn7@pdx.edu", "date": "2024-02-12 19:23:04.517089+00:00", "message"
: "Hello App Engine!"}, {"name": "Joseph Nguyen", "email": "jtn7@pdx.edu", "date": "2024-02-23
21:09:51.486680+00:00", "message": "Hello Kubernetes!"}, {"name": "Joseph Nguyen", "email": "jt
n7@pdx.edu", "date": "2024-02-11 06:51:50.427049+00:00", "message": "Hello Compute Engine!"}, {
"name": "Joseph Nguyen", "email": "jtn7@pdx.edu", "date": "2024-02-11 06:26:07.393490+00:00",
"message": "Hello Docker Datastore!"}, {"name": "Joseph Nguyen", "email": "jtn7@pdx.edu", "date"
: "2024-02-18 22:24:10.996036+00:00", "message": "Hello Cloud Run!"}]
>>>
```

Take a screenshot showing the preflight request to the API that allows API access, as well as the subsequent fetch request have been successful.

The screenshot shows a web browser at the URL `/home/jtn7/Desktop/cs430-src/06_gcp_restapi_cloudfunctions/frontend-src/index.html`. The page displays a "Guestbook" form with fields for "Name:", "Email:", and "Message:". Below the form is a "Sign" button. The "Entries" section shows a single entry: "Joseph Nguyen <jtn7@pdx.edu> signed on 2024-02-23 21:40:59.669138+00:00 Hello Cloud Build!". The Network tab is open, showing a list of requests. The "entries" request is highlighted, showing a status of 200, a type of "fetch", and a size of 446 B. The "Preflight" request is also shown, with a status of 200 and a size of 0 B.

The screenshot shows the same web browser, but the "Name:" field is now filled with "Joseph Nguyen" and the "Email:" field is filled with "jtn7@pdx.edu". The "Message:" field contains the text "Hello API Gateway from local SPA!". The "Sign" button is visible. The "Entries" section shows the same entry as before. The Network tab is open, showing a list of requests. The "entries" request is highlighted, showing a status of 200, a type of "fetch", and a size of 446 B. The "Preflight" request is also shown, with a status of 200 and a size of 0 B.

The screenshot shows the same web browser, but the "Name:" field is now filled with "Joseph Nguyen" and the "Email:" field is filled with "jtn7@pdx.edu". The "Message:" field contains the text "Hello API Gateway from local SPA!". The "Sign" button is visible. The "Entries" section shows the same entry as before. The Network tab is open, showing a list of requests. The "entries" request is highlighted, showing a status of 200, a type of "fetch", and a size of 446 B. The "Preflight" request is also shown, with a status of 200 and a size of 0 B.

Enter a message using your name, PSU e-mail address, and the message "Hello API Gateway from SPA in GCS!". Scroll down to find the message posted. Take a screenshot of the Guestbook including the URL.

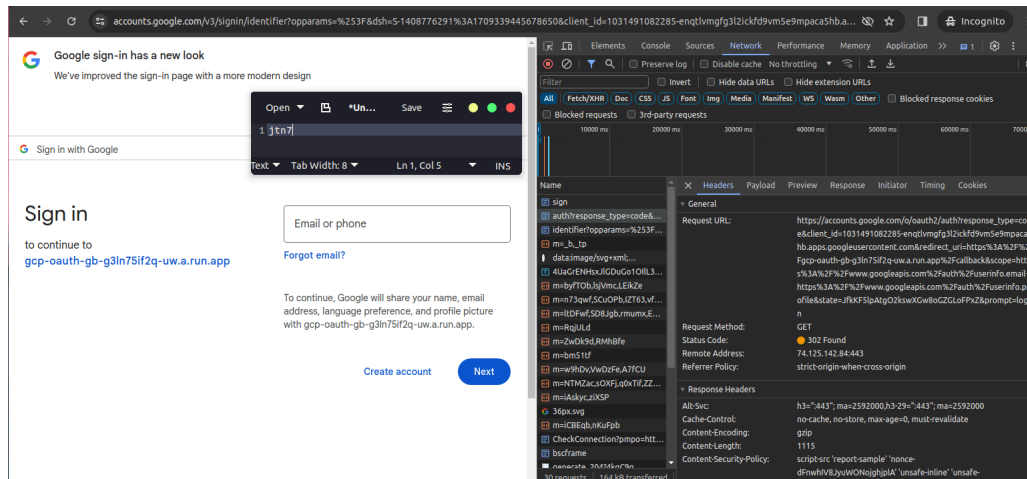
The screenshot shows a web browser at `storage.googleapis.com/gbapi-jtn/index.html`. The page has a title "Guestbook". Below it are three input fields: "Name:" with the value "Joseph Nguyen", "Email:" with the value "jtn7@pdx.edu", and "Message:" with the value "Hello API Gateway from SPA in GCS!". Below the message field is a "Sign" button. Below the form is a section titled "Entries" which contains the following text: "Joseph Nguyen <jtn7@pdx.edu> signed on 2024-02-23 21:40:59.669138+00:00 Hello Cloud Build!"

The screenshot shows the "Entries" section of the Guestbook. It contains three entries, each separated by a horizontal line. The first entry is: "Joseph Nguyen <jtn7@pdx.edu> signed on 2024-02-18 22:24:10.996036+00:00 Hello Cloud Run!". The second entry is: "Joseph Nguyen <jtn7@pdx.edu> signed on 2024-03-01 06:44:02.374622+00:00 Hello API Gateway from local SPA!". The third entry is: "Joseph Nguyen <jtn7@pdx.edu> signed on 2024-03-01 07:00:36.047975+00:00 Hello API Gateway from SPA in GCS!".

8.3

Take a screenshot of the Headers that includes the URL and the returned HTTP status code for the first two requests for your lab notebook.

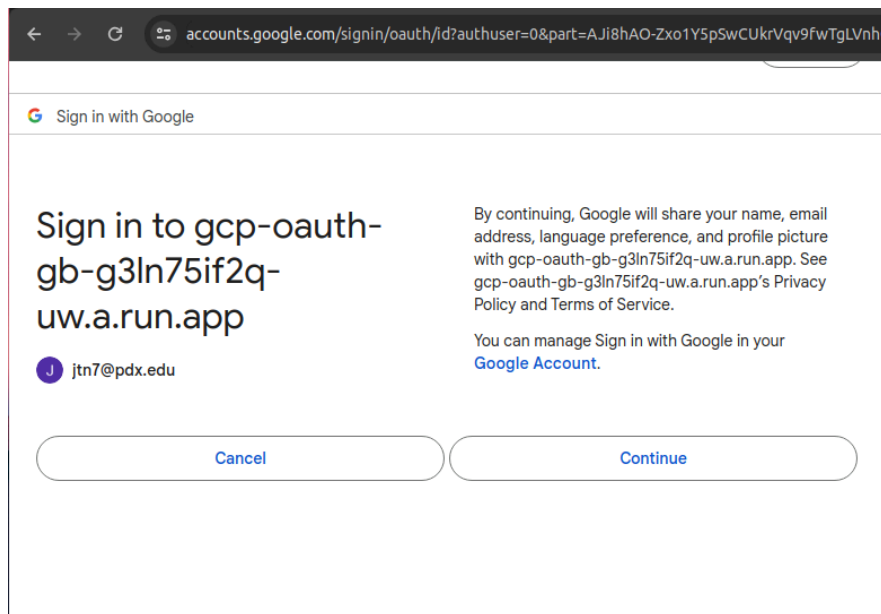
The screenshot shows a Google sign-in page in a browser. The page title is "Google sign-in has a new look". Below the title is a "Sign in with Google" button. Below that is a "Sign in" section with a "Email or phone" input field and a "Forgot email?" link. Below the input field is a "Create account" button and a "Next" button. To the right of the sign-in page is a network inspector showing the headers for the first two requests. The first request is a GET request to `https://gcp-oauth-gb-g3ln75if2q-uw.a.run.app/sign` with a status code of 302 Found. The second request is a GET request to `https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=1031491082285-enqtlvmfg3l2ickfd9vm5e9mpacashb.a...` with a status code of 200 OK.



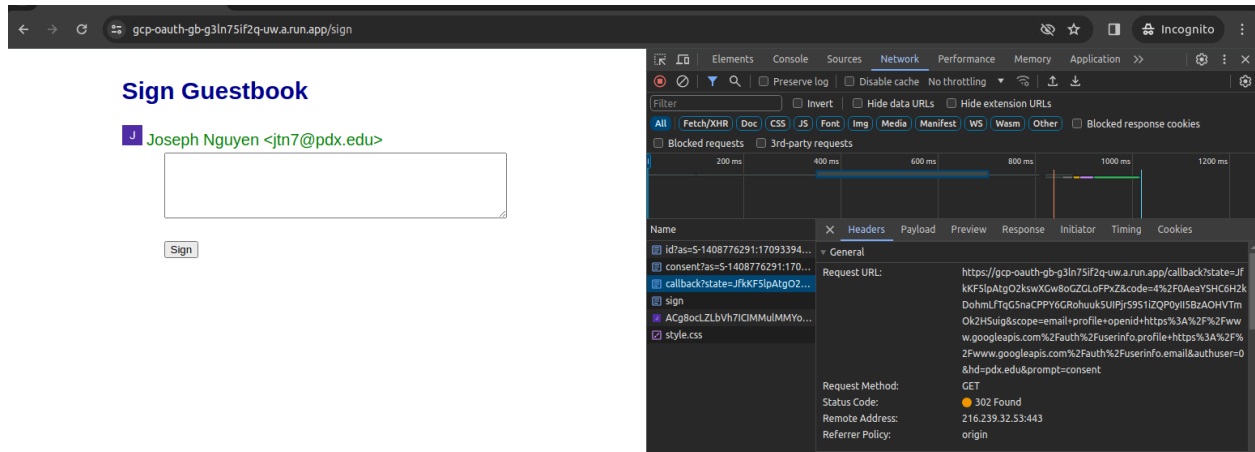
Based on the description of the source code, what lines of code in our application are responsible for the second request shown?

We are getting redirected to the callback in order to get authenticated. This functionality is implemented in the `sign.py` from lines approximately from lines 18-23.

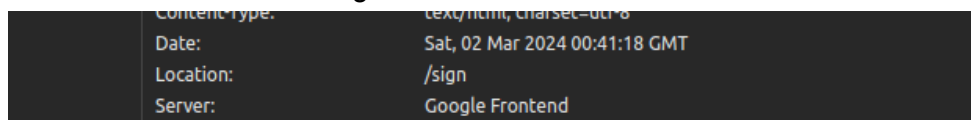
Take a screenshot of the permissions you (as a user) are granting the Guestbook access to.



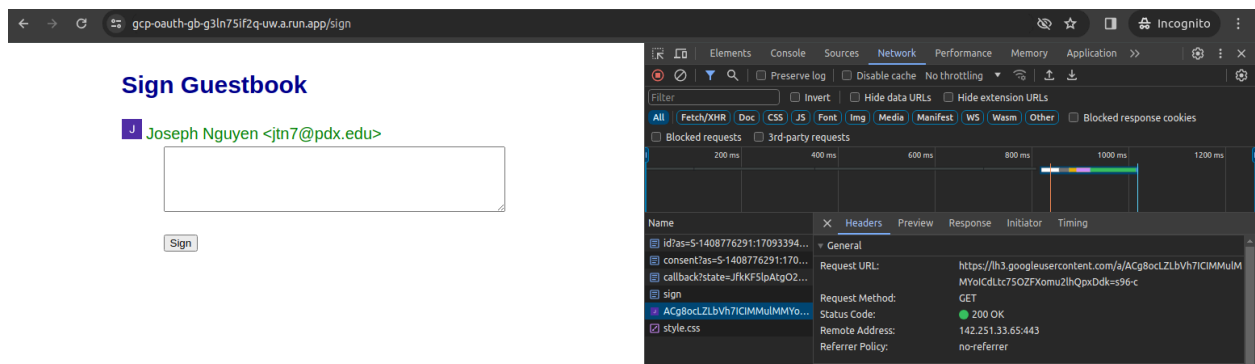
Take a screenshot of the Headers that includes the entire Callback URL and its returned HTTP status code. What is the URI for the Location that the User sent to by the Callback?



The user is sent back to /sign

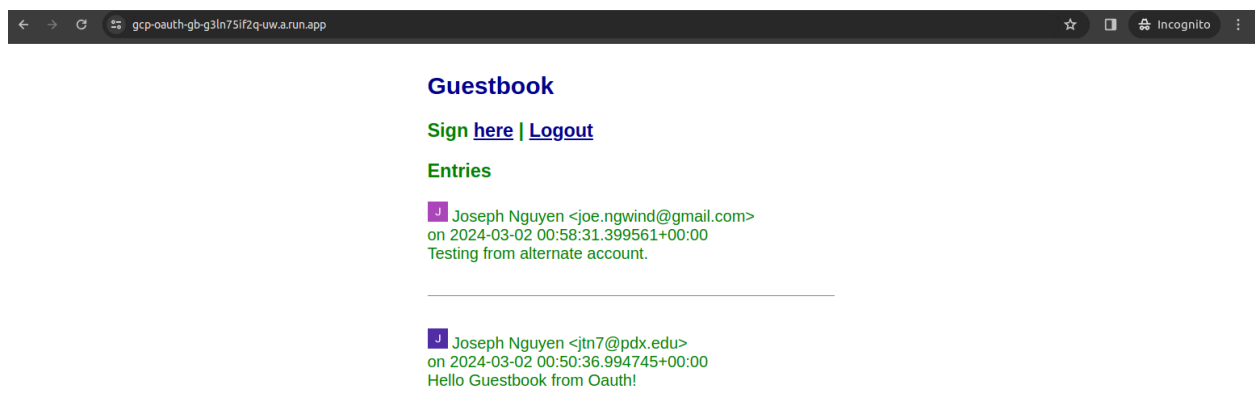


Find the request within Developer Tools that fetches the embedded image and take a screenshot of its URL.

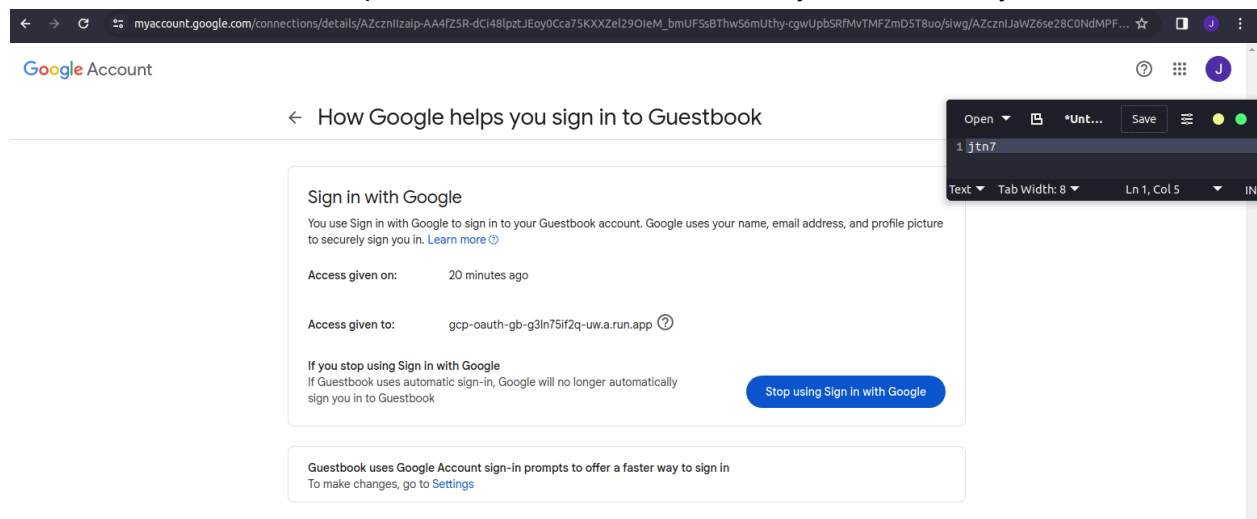


Sign in again using a different account and add another entry to the Guestbook.

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



Take a screenshot of the expanded information that includes your OdinId for your lab notebook.

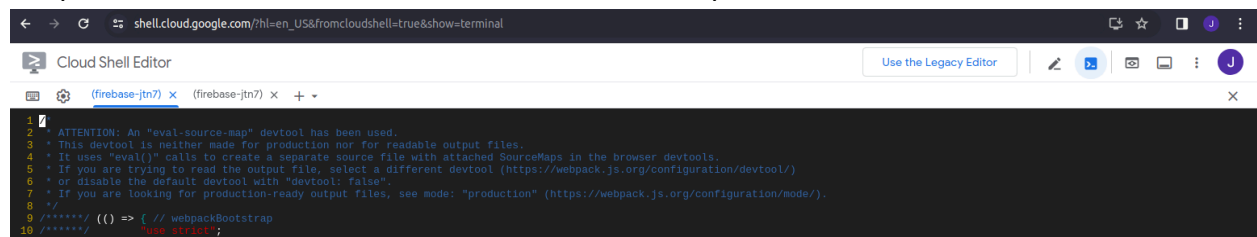


8.4

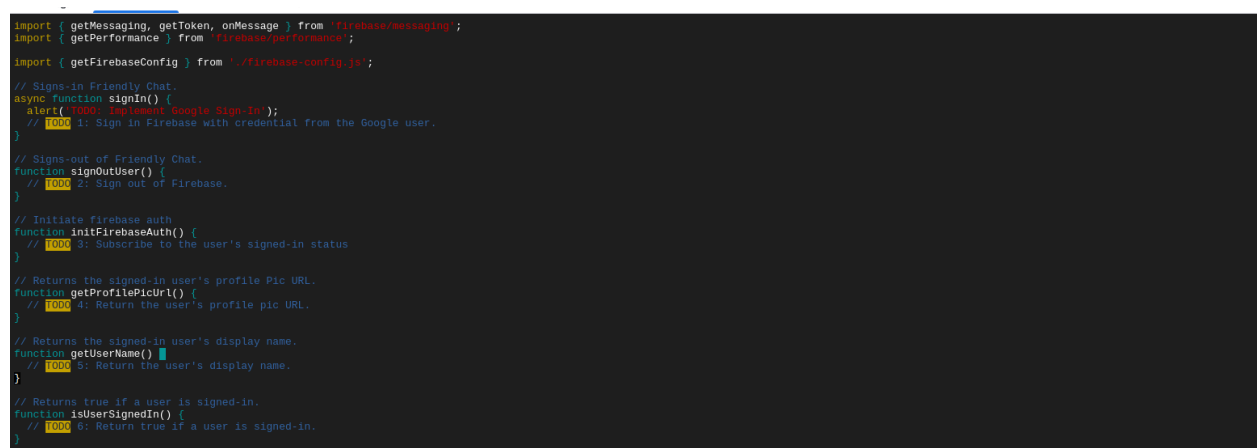
What other domains are given access to this Firebase project by default?

localhost, fir-jtn7.firebaseio.com, fir-jtn7.web.app

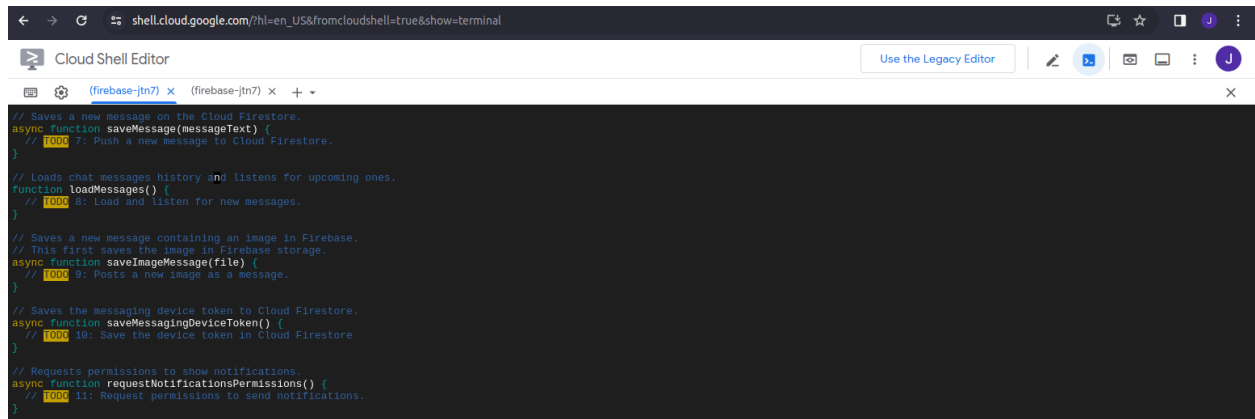
Go back to the other Cloud Shell terminal and bring up the file that has been produced by webpack. Take a screenshot of the first 10 lines of the produced file.



What missing functions deal with user authentication?



What missing functions deal with sending and receiving messages?



```
// Saves a new message on the Cloud Firestore.
async function saveMessage(messageText) {
  // TODO 7: Push a new message to Cloud Firestore.
}

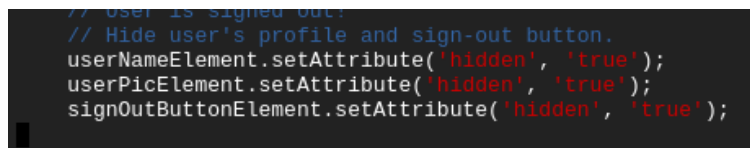
// Loads chat messages history and listens for upcoming ones.
function loadMessages() {
  // TODO 8: Load and listen for new messages.
}

// Saves a new message containing an image in Firebase.
// This first saves the image in Firebase storage.
async function saveImageMessage(file) {
  // TODO 9: Posts a new image as a message.
}

// Saves the messaging device token to Cloud Firestore.
async function saveMessagingDeviceToken() {
  // TODO 10: Save the device token in Cloud Firestore
}

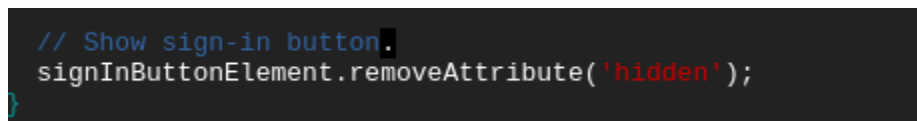
// Requests permissions to show notifications.
async function requestNotificationsPermissions() {
  // TODO 11: Request permissions to send notifications.
}
```

What are the names of the elements that are hidden when the user is signed out?



```
// User is signed out:
// Hide user's profile and sign-out button.
userNameElement.setAttribute('hidden', 'true');
userPicElement.setAttribute('hidden', 'true');
signOutButtonElement.setAttribute('hidden', 'true');
```

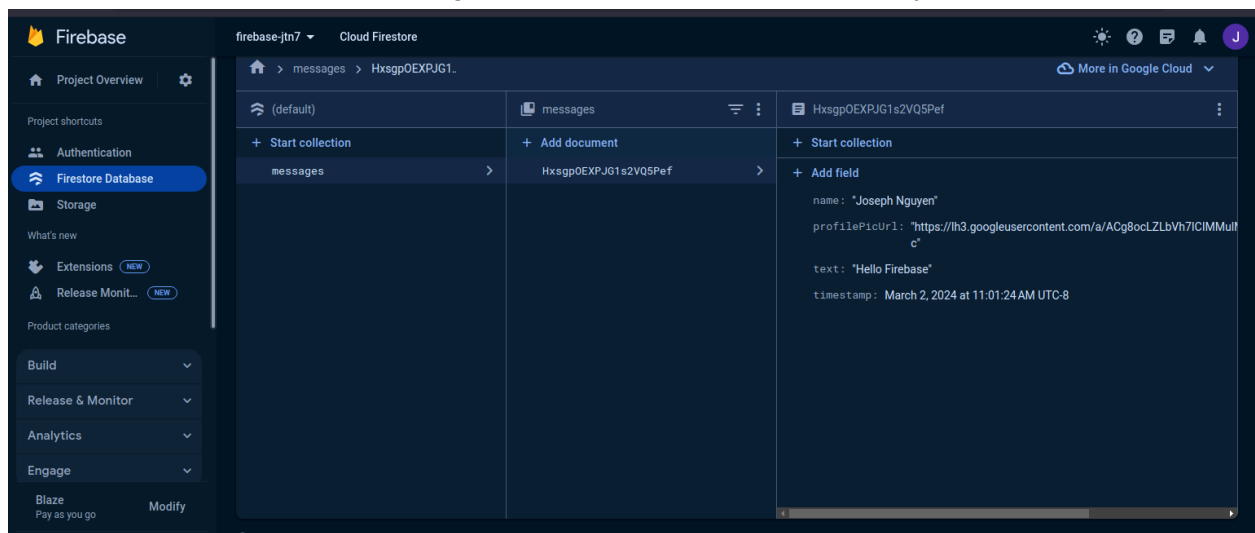
What is the name of the element that is not hidden when the user is signed out?



```
// Show sign-in button.
signInButtonElement.removeAttribute('hidden');
```

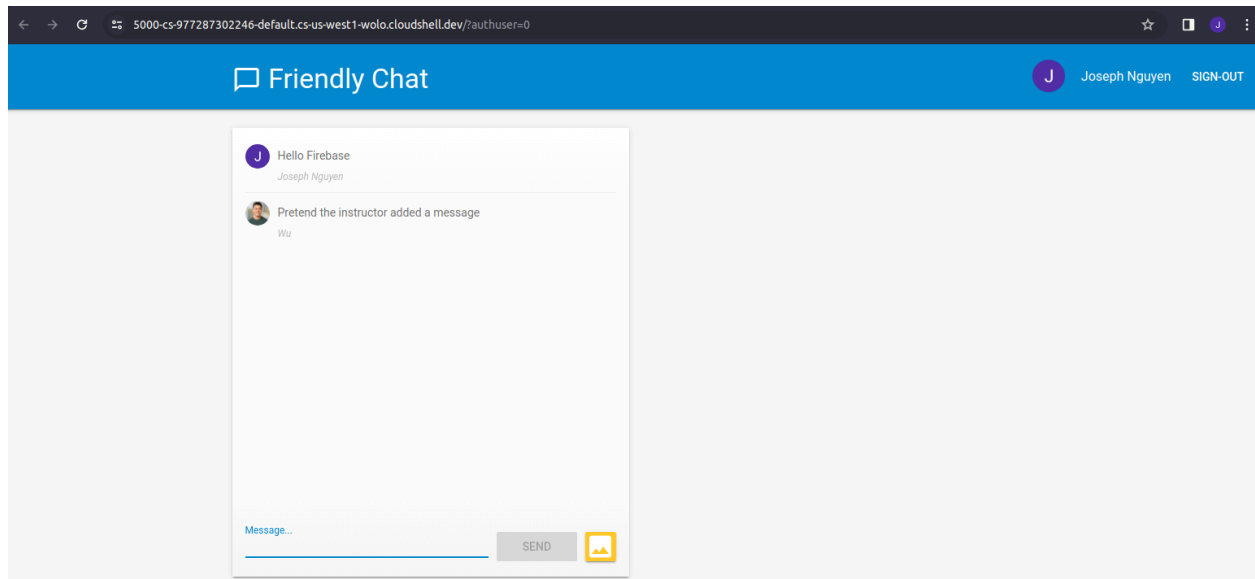
Go back to the Firebase console and bring up the application's Firestore database. Expand the messages collection and find the document containing the message within it.

Include a screenshot of the message and its fields in the database for your lab notebook



Save the document and re-visit the Friendly Chat application.

Include a screenshot of the application with its two messages for your lab notebook



What is the URL of the image that is first shown in the UI as the message is loading?

```
// A loading image URL.  
var LOADING_IMAGE_URL = 'https://www.google.com/images/spin-32.gif?a';
```

Go back to the Firebase console and bring up the application's Firestore database. Expand the messages collection and find the document containing the message within it.

Answer the following questions:

How do the fields in an image document differ from that of the text document?

Instead of having a text field, they have an imageUrl field and a storageUri field.

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

ImageUrl:

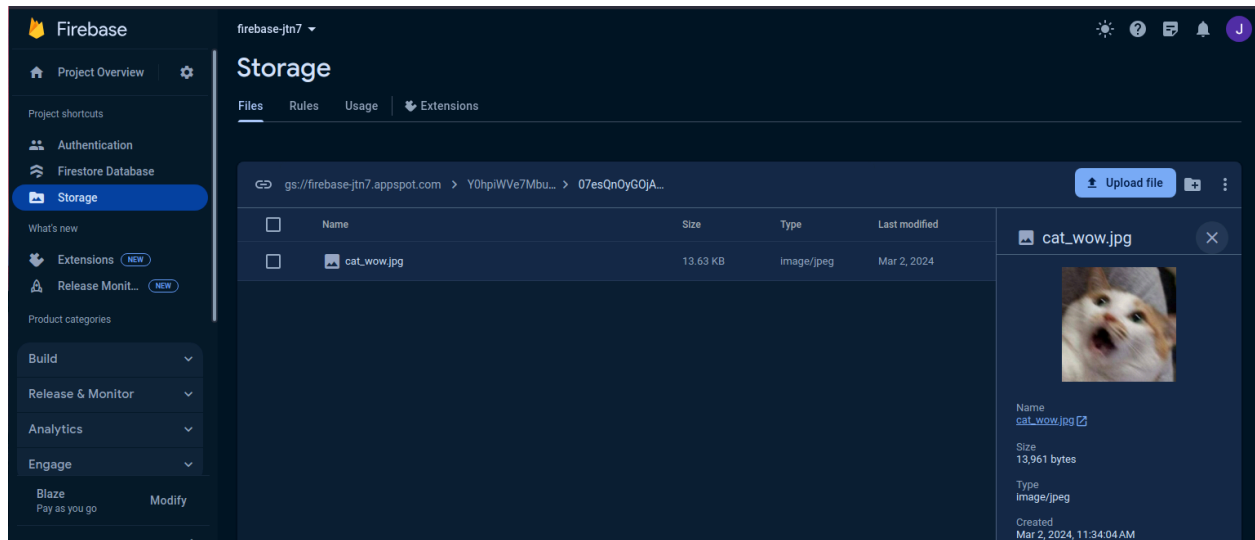
https://firebasestorage.googleapis.com/v0/b/firebase-jtn7.appspot.com/o/Y0hpiWVe7Mbu1ejLZDDCEi79ADj1%2F07esQnOyGOjAA1VtcgPm%2Fcat_wow.jpg?alt=media&token=4b9b91d5-a520-4e32-891b-b31962b62f22

StorageUri:

Y0hpiWVe7Mbu1ejLZDDCEi79ADj1/07esQnOyGOjAA1VtcgPm/cat_wow.jpg"

Visit the "Storage" section in the Firebase console

Take a screenshot of the image in the storage bucket for your lab notebook.



What directory is the application going to be served from?

It will come from ./public

```
{
  "hosting": {
    "public": "./public",
    "headers": [
      {
        "source": "**/*.@(js|html)",
        "headers": [
          {
            "key": "Cache-Control",
            "value": "max-age=0"
          }
        ]
      }
    ]
  }
}
```

Screenshot of friend using friendlychat application

