

Project Nestly

E6156 - Topics in SW Engineering Management: Cloud Rental Property Management System

Focal Point: Zhang, Youcheng(yz4589)

Team Members:

Yao, Xirui, xy2571

Chen, Shiying, sc5299

Lin, Runze, rl3376

Wang, Isa, yw3886

Demo video link

Demo video one piece:

<https://drive.google.com/drive/folders/1NHVRm7xkr4ZcbKqtPqhoU3HXuEmL0UWI>

Demo video in parts (intro + 10 required parts + business logic)

<https://drive.google.com/drive/folders/1fl35cs0cxPva7Q74Hgsx2wl9x5aow5Ey>

Entire code zip link

https://drive.google.com/drive/folders/1N-gnaxZV_EN-9F4rD7GrQ-IAM9Me81n3

Introduction

Nestly: Cloud Rental Property Management, the complexities are numerous:

- Facilitating seamless property listings and bookings for diverse users.
- Ensuring smooth interactions between hosts, guests (both registered and unregistered), and administrators.
- Integrating multiple cloud services and maintaining architectural consistency across them.

Nestly, a cloud-native platform tailored to revolutionize the rental property management landscape. It's not just a tool but an embodiment of efficiency, designed to cater to the varied needs of property listing, management, and booking.

Personas and Roles

There are 3 personas:

1. Guest
 - a. Unregistered guest
 - b. Registered guest
2. Host/Landlord
3. Customer Service (Admin)

Key Features

An Unregistered Guest can:

1. Search for properties based on address, house type, house size, availability, host_id, and price.

A Registered Guest can:

1. Register and log in using Google SSO
2. Search for properties based on address, house type, house size, availability, host_id, and price.
3. Create and cancel bookings for available properties
4. View their own bookings

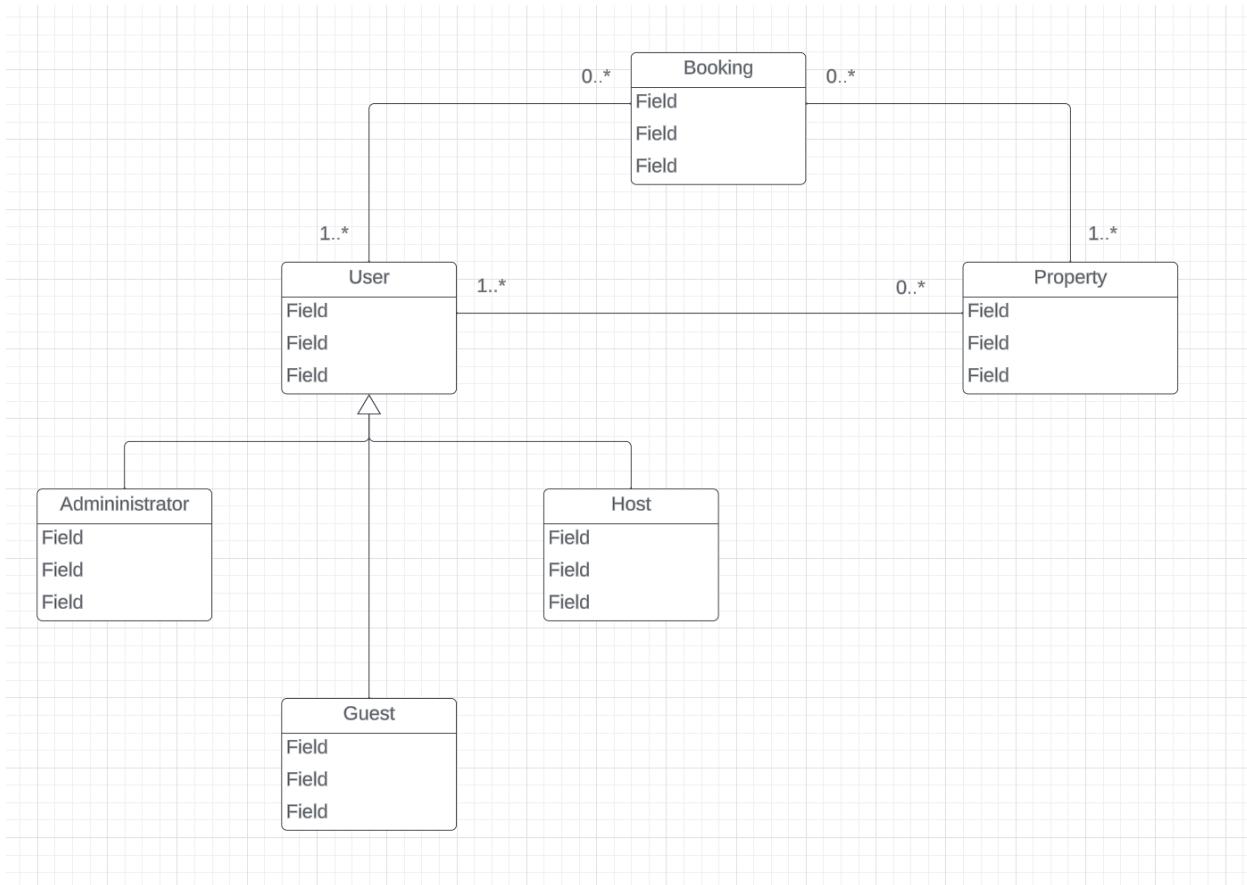
A Host can:

1. Register and log in using Google SSO
2. List their properties, availability, and pricing.
3. Receive notifications for bookings made by guests for their property.
4. Update the availability and pricing of their property as needed.
5. View their own properties
6. View their own bookings

An Administrator can:

1. Register and log in using Google SSO
2. Manage all property listings. Get, create, update, and remove property listings.
3. Manage all user information. Get, create, update, and delete users.
4. Manage all bookings. Get, create, update, and remove bookings

Domain Model



The domain model outlines the main entities and their relationships within the system. At its core is the "User" entity, which can be an admin, a host, an unregistered guest, or a registered guest. Users can make and own multiple bookings and properties. "Booking" represents the reservation of a property, while "Property" signifies accommodations available for booking. Admins have overarching control, hosts offer properties, and guests book them. The relationships between these entities highlight the interactions within the system like users making bookings or owning properties.

Resource Paths

/api/docs: OpenAPI documentation.

Users:

- Guests (Registered):
 - /api/users/{user_id}
- Host:
 - /api/users/{user_id}
- Admin:
 - /api/users (all users) (filter & pagination enabled)
 - /api/users/{user_id}

Property:

- /api/properties (filter & pagination enabled)
- /api/properties/{property_id}
- /api/properties/search (for guests)
- /api/properties/host_property_management/{host_id} (for host)

Booking:

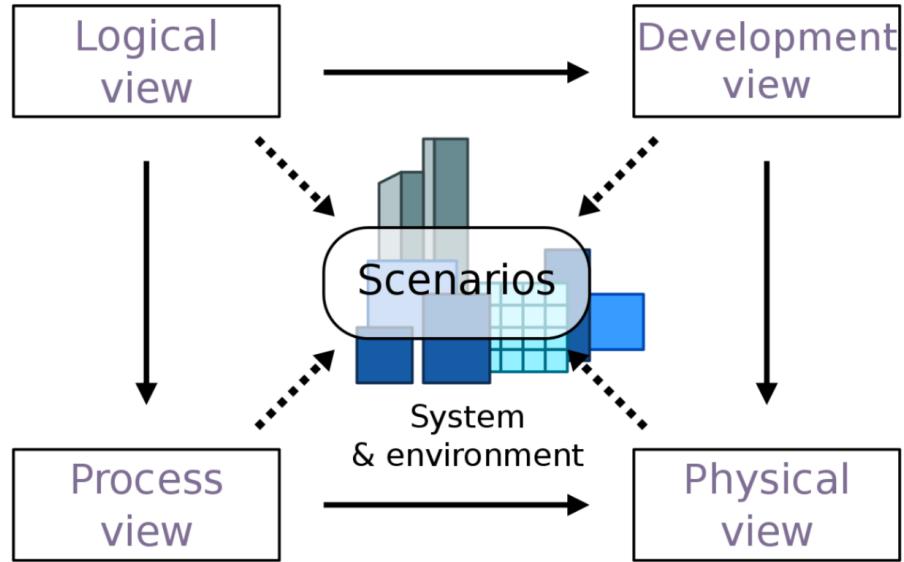
- /api/bookings (filter & pagination enabled)
- /api/bookings/{booking-id}
- /api/bookings/booking_management_host/{host_id}
- /api/bookings/user_booking_management/{host_id}

Composite:

- /api/clientType
- /api/user (stands for guest here)
- /api/host

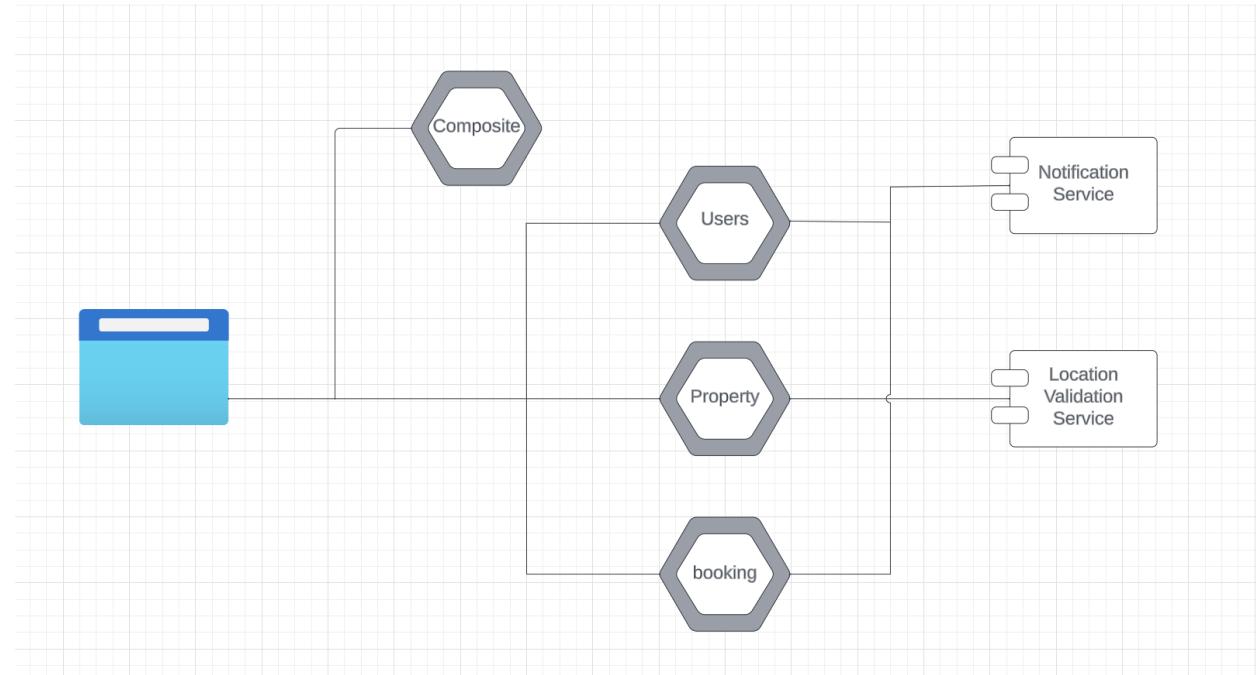
Architecture

Our team uses the subset of the approach: logical view and physical view. And will be more complete over time.



Logical View

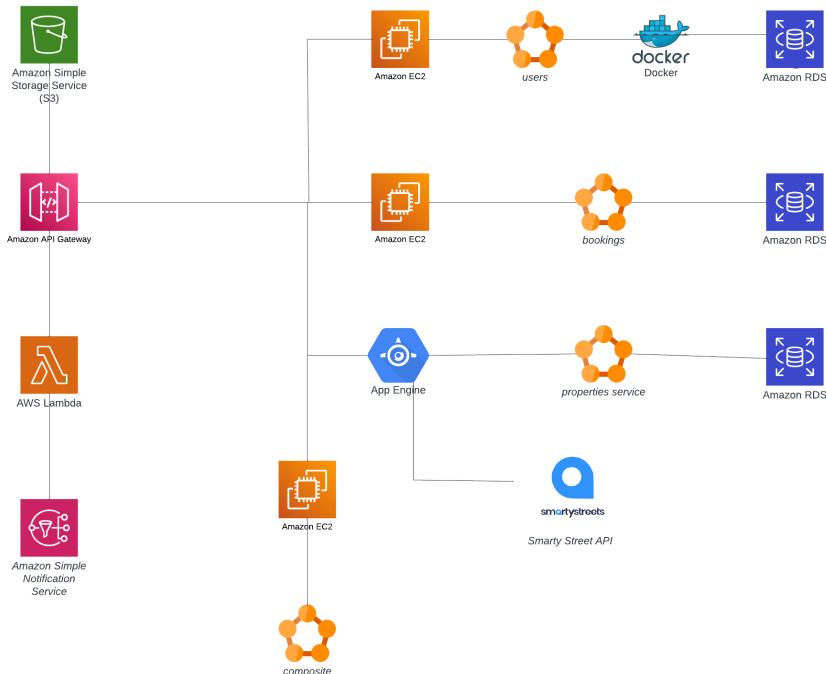
A basic concept in our logical view is following Hexagonal Architecture.



1. Composite: Acts as an orchestrator or aggregator, consolidating data and operations from other microservices to provide comprehensive functionality. Mostly in charge of redirecting to their related functions for different roles (host/guest).
2. User Management Service:
 - a. Manage registration and login functionalities for all user types other than Unregistered Guests: (Registered) Guest, Host, and Administrator
 - b. Manage role-based access control
 - c. Allow update & delete of users
3. Property Management Service:
 - a. Allow hosts to list, update, and manage their properties.
 - b. Handle searches for properties based on various criteria such as location, availability, and price
 - c. Manage the availability and pricing updates for properties.
4. Booking Service:
 - a. Handle the entire booking process, including creating & canceling(deleting) process
 - b. Send email notifications to hosts when their properties get booked.
 - c. Allow Admin to mange bookings (E.g. update the guest_id and property_id linked with a bookings)

Physical View

This is the draft, in-progress physical view of the first few milestones.



Three microservices:

1. User Management Service on EC2 with Docker
 - a. Database interaction with Amazon RDS
2. Property Management Service on GCP:
 - a. Database interaction with Amazon RDS
3. Booking & Review Service on EC2 without Docker
 - a. Database interaction with Amazon RDS

Codebase

<https://github.com/Runze-Lin/Micoservice-User>

<https://github.com/Runze-Lin/Micoservice-Bookings-and-Review>

<https://github.com/Runze-Lin/Micoservice-Property>

<https://github.com/Runze-Lin/Composite>

<https://github.com/users/syuchen010223/projects/1>

Links to our deployment

User Management (EC2 with Docker):

<http://ec2-3-144-182-9.us-east-2.compute.amazonaws.com:8012>

User Management Interface

Filter Users

Apply Filters

Get All Users

[Get All Users](#)

Users Results

Create User

Create User

Update User

Note: Only User ID is required for updating! You may update one or multiple other fields.

Update User

Delete User

Delete User

Booking & Review Service (EC2): <http://ec2-3-144-93-114.us-east-2.compute.amazonaws.com:8012>

Booking Management System

Filter Bookings

Apply Filters

Get All Bookings

[Get All Properties](#)

Bookings Results

Add Booking

Update Booking

User ID: Host ID: Property ID:

 Total Price:

Delete Booking

Property Management Service on GCP:
<https://e6156-i-am-bezos-402423.ue.r.appspot.com>

Property Management Interface

Filter Properties

Get All Properties

Properties Results

Create Property

Update Property

Note: Only Property ID is required for updating! You may update one or more other fields.

Delete Property

Note: frontend implementation is not restricted to the above screenshots, please refer to our demo for more connected frontend implementations

API gateway:

<https://eqfosdyv30.execute-api.us-east-1.amazonaws.com/v1/>

Resources

The screenshot shows the AWS API Gateway 'Resources' section. At the top is a 'Create resource' button. Below it is a tree view of API endpoints under a root node represented by a folder icon and a '/' character. The tree includes:

- /bookings
 - DELETE
 - GET
 - OPTIONS
 - POST
 - PUT
 - {booking_id}
 - DELETE
 - PUT
- /properties
 - DELETE
 - GET
 - OPTIONS
 - POST
 - PUT
 - {property_id}
 - DELETE
 - PUT
- /users
 - DELETE
 - GET
 - OPTIONS
 - POST
 - PUT
 - {user_id}
 - DELETE
 - PUT

(with resources):

<https://eqfosdyv30.execute-api.us-east-1.amazonaws.com/v1/users> (unauthorized if directly click, due to authorizer & jwt token implementation)

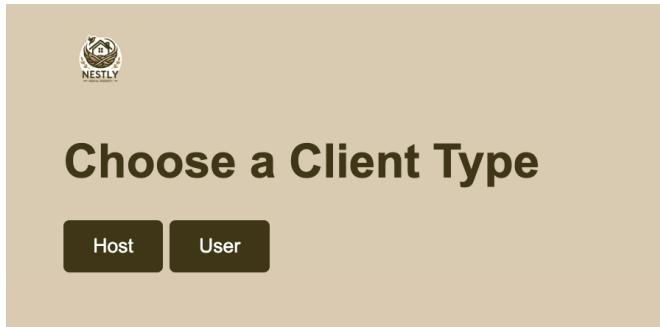
```
{"message": "Missing Authentication Token"}
```

<https://eqfosdyv30.execute-api.us-east-1.amazonaws.com/v1/properties>

<https://eqfosdyv30.execute-api.us-east-1.amazonaws.com/v1/bookings>

Composite:

<http://ec2-52-14-240-60.us-east-2.compute.amazonaws.com:8012/>



S3:

<https://nestly6156.s3.us-east-2.amazonaws.com>

Amazon S3 > Buckets > nestly6156

nestly6156 Info Publicly accessible

Objects Properties Permissions Metrics Management Access Points

Objects (4) Info

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Actions

Create folder **Upload**

Find objects by prefix

| Name | Type | Last modified | Size | Storage class |
|------------------|--------|---------------|------|---------------|
| bookings_static/ | Folder | - | - | - |
| composite/ | Folder | - | - | - |
| property_static/ | Folder | - | - | - |
| users_static/ | Folder | - | - | - |

Implementation Screenshots

Enhance Microservices Implementation:

CRUD (finished on all 3 microservices, using Users as example):

GET: (getting all here)

⚠ 不安全 | ec2-3-144-182-9.us-east-2.compute.amazonaws.com:8012

Get All Users

[Get All Users](#)

Users Results

```
ID: 00000001, Username: Tarantula, Name: Morty Stokey, Email: mstokey0@slashdot.org, Credit: 294, OpenID: 9385195611, Role: host
ID: 00000002, Username: Bontebok, Name: Marguerite Farington, Email: mfarington1@adobe.com, Credit: 1885, OpenID: 0967253799, Role: guest
ID: 00000003, Username: Pheasant, common, Name: Amelia Bexon, Email: abexon2@jiathis.com, Credit: 2402, OpenID: 6571275465, Role: guest
ID: 00000004, Username: Azara's zorro, Name: Timmie Beine, Email: tbeine3@msu.edu, Credit: 4815, OpenID: 9514213866, Role: host
ID: 00000005, Username: Dusky rattlesnake, Name: Eileen Geary, Email: egeary4@t-online.de, Credit: 2599, OpenID: 3706445166, Role: host
ID: 00000006, Username: Weaver, chestnut, Name: Jemima Braghini, Email: jbraghini5@google.co.jp, Credit: 3503, OpenID: 3453605004, Role: host
ID: 00000007, Username: Gazer, sun, Name: Liv Toumlin, Email: ltoumlin@1und1.de, Credit: 4341, OpenID: 6593544108, Role: host
ID: 00000008, Username: Shrike, southern white-crowned, Name: Darwin Adshead, Email: dadshead7@npr.org, Credit: 232, OpenID: 1350189928, Role: guest
ID: 00000009, Username: Arboral spiny rat, Name: Reynold Lambden, Email: rlambden8@java.com, Credit: 1715, OpenID: 2332988181, Role: guest
ID: 00000010, Username: Lemming, arctic, Name: Talia Reuven, Email: treuven9@wp.com, Credit: 4542, OpenID: 2251888098, Role: guest
ID: 00000011, Username: Cattle egret, Name: Verne Gendrich, Email: vgendricha@cbslocal.com, Credit: 1471, OpenID: 1500125113, Role: host
ID: 00000012, Username: Grey fox, Name: Dedie Squelch, Email: dsquelchb@pagesperso-orange.fr, Credit: 1533, OpenID: 3610011556, Role: guest
ID: 00000013, Username: Green-winged macaw, Name: Adorna McIlvenny, Email: amcilvernyc@europa.eu, Credit: 4283, OpenID: 5106976995, Role: host
ID: 00000014, Username: Emu, Name: Ines Ayton, Email: iayton@unesco.org, Credit: 4160, OpenID: 7225331302, Role: guest
ID: 00000015, Username: Gull, pacific, Name: Janka Scotchmore, Email: jscotchmoree@flavors.me, Credit: 1881, OpenID: 6556520225, Role: guest
ID: 00000016, Username: Hawk, ferruginous, Name: Colin Main, Email: cmainf@who.int, Credit: 363, OpenID: 5467203681, Role: host
ID: 00000017, Username: Nilgai, Name: Celestine Lamps, Email: clamps9@imgur.com, Credit: 3242, OpenID: 4443379878, Role: host
ID: 00000018, Username: Pigeon, wood, Name: Miranda Crowch, Email: mcrowch@typepad.com, Credit: 1898, OpenID: 9093904108, Role: guest
ID: 00000019, Username: Painted stork, Name: Hermine Blazic, Email: hblazici@dmoz.org, Credit: 706, OpenID: 7513278881, Role: host
ID: 00000020, Username: Arctic hare, Name: Charity Nelissen, Email: cnelissenj@fc2.com, Credit: 843, OpenID: 6810216715, Role: host
ID: 00000021, Username: Black-eyed bulbul, Name: Janith Riccetti, Email: jriccettik@deliciousdays.com, Credit: 522, OpenID: 6725080715, Role: guest
```

PUT: (added a test user 'testtest' and the GET on username=testtest)

⚠ 不安全 | ec2-3-144-182-9.us-east-2.compute.amazonaws.com:8012

Get All Users

[Get All Users](#)

Users Results

...4-182-9.us-east-2.compute.amazonaws.com:8012 显示
User created

[确定](#)

Create User

testtest firstname lastname test@gmail.com 300

Filter Users

| | | | | |
|---------|--|------------------|-----------|-------|
| User ID | testtest | First Name | Last Name | Email |
| Credit | Credit Greater Than | Credit Less Than | Role | 10 |
| 0 | <input type="button" value="Apply Filters"/> | | | |

Get All Users

[Get All Users](#)

Users Results

ID: 00000301, Username: testtest, Name: firstname lastname, Email: test@gmail.com, Credit: 300, OpenID: null, Role: guest

POST: (updated the added user using user id=00000301, username to 'changed!', and the GET on user id=00000301, showing username changed to 'changed!')

⚠ 不安全 | ec2-3-144-182-9.us-east-2.compute.amazonaws.com:8012

...4-182-9.us-east-2.compute.amazonaws.com:8012 显示

User updated

ID: 00000301, Username: testtest, Name: firstname lastname, Email: test@gmail.com, Credit: 300, OpenID: null, Role: guest

确定 Role: guest

Users Results

Create User

testtest firstname lastname test@gmail.com 300

Guest Create User

Update User

Note: Only User ID is required for updating! You may update one or multiple other fields.

00000301 changed!

First Name Last Name Email Credit Select Role

Update User

Filter Users

00000301 Username First Name Last Name Email

Credit Credit Greater Than Credit Less Than Role 10

0 Apply Filters

Get All Users

Get All Users

Users Results

ID: 00000301, Username: changed!, Name: firstname lastname, Email: test@gmail.com, Credit: 300, OpenID: null, Role: guest

DELETE (DELETE user id=00000301, and GET user id=00000301 returning “No (such) users were found”):

▲ 不安全 | ec2-3-144-182-9.us-east-2.compute.amazonaws.com:8012

...4-182-9.us-east-2.compute.amazonaws.com:8012 显示

User deleted

ID: 00000301, Username: changed!, Name: firstn...
Email: test@gmail.com, Role: guest

确定

Create User

testest firstname lastname test@gmail.com 300
Guest Create User

Update User

Note: Only User ID is required for updating! You may update one or more other fields. 00000301 changed!
First Name Last Name Email Credit Select Role
Update User

Delete User

00000301 Delete User

Filter Users

00000301 Username First Name Last Name Email
Credit Credit Greater Than Credit Less Than Role 10
0 Apply Filters

Get All Users

Get All Users

Users Results

No (such) users were found

Querying (realized on Properties)
(E.g. price_gt=350, return properties with price greater than 350)

◀ → C ▲ e6156-i-am-bezos-402423.ue.r.appspot.com/properties?price_gt=350

```
[{"property_id": "0000000002", "property_address": "202 Birch Boulevard, Central City, USA", "house_type": "studio", "house_size": "100", "price": 392, "availability": 0, "host_id": "00000004"}, {"property_id": "0000000007", "property_address": "707 Sycamore Place, Fabletown, USA", "house_type": "1b1b", "house_size": "179", "price": 378, "availability": 0, "host_id": "00000013"}, {"property_id": "0000000008", "property_address": "789 Pine Road, Metropolis, USA", "house_type": "2b2b", "house_size": "68", "price": 394, "availability": 0, "host_id": "00000016"}, {"property_id": "0000000011", "property_address": "202 Birch Boulevard, Central City, USA", "house_type": "2b1b", "house_size": "188", "price": 467, "availability": 1, "host_id": "00000020"}, {"property_id": "0000000013", "property_address": "123 Maple Street, Springfield, USA", "house_type": "3b2b", "house_size": "300", "price": 500, "availability": 1, "host_id": "00000024"}, {"property_id": "0000000014", "property_address": "404 Fir Terrace, Atlantis, USA", "house_type": "2b2b", "house_size": "101", "price": 461, "availability": 0, "host_id": "00000025"}, {"property_id": "0000000017", "property_address": "406 Elm Lane, Star City, USA", "house_type": "1b2b", "house_size": "105", "price": 417, "availability": 1, "host_id": "00000027"}, {"property_id": "0000000018", "property_address": "101 Elm Lane, Star City, USA", "house_type": "1b2b", "house_size": "105", "price": 443, "availability": 1, "host_id": "00000029"}, {"property_id": "0000000019", "property_address": "505 Redwood Circle, Olympia, USA", "house_type": "2b1b", "house_size": "175", "price": 494, "availability": 1, "host_id": "00000032"}, {"property_id": "0000000020", "property_address": "123 Elm Lane, Star City, USA", "house_type": "2b1b", "house_size": "175", "price": 494, "availability": 1, "host_id": "00000037"}, {"property_id": "0000000023", "property_address": "404 Fir Terrace, Atlantis, USA", "house_type": "2b2b", "house_size": "115", "price": 491, "availability": 1, "host_id": "00000045"}, {"property_id": "0000000025", "property_address": "123 Maple Street, Springfield, USA", "house_type": "2b2b", "house_size": "183", "price": 378, "availability": 1, "host_id": "00000052"}, {"property_id": "0000000026", "property_address": "123 Maple Street, Springfield, USA", "house_type": "studio", "house_size": "84", "price": 399, "availability": 1, "host_id": "00000055"}, {"property_id": "0000000028", "property_address": "456 Oak Avenue, Gotham, USA", "house_type": "2b1b", "house_size": "95", "price": 414, "availability": 1, "host_id": "00000059"}, {"property_id": "0000000030", "property_address": "101 Elm Lane, Star City, USA", "house_type": "2b1b", "house_size": "95", "price": 414, "availability": 1, "host_id": "00000061"}, {"property_id": "0000000034", "property_address": "456 Oak Avenue, Gotham, USA", "house_type": "1b1b", "house_size": "158", "price": 453, "availability": 0, "host_id": "00000074"}, {"property_id": "0000000037", "property_address": "707 Sycamore Place, Fabletown, USA", "house_type": "2b2b", "house_size": "166", "price": 458, "availability": 1, "host_id": "00000077"}, {"property_id": "0000000040", "property_address": "707 Sycamore Place, Fabletown, USA", "house_type": "2b2b", "house_size": "68", "price": 427, "availability": 0, "host_id": "00000082"}, {"property_id": "0000000044", "property_address": "789 Pine Road, Metropolis, USA", "house_type": "3b2b", "house_size": "64", "price": 374, "availability": 1, "host_id": "00000083"}, {"property_id": "0000000046", "property_address": "789 Pine Road, Metropolis, USA", "house_type": "studio", "house_size": "35", "price": 488, "availability": 1, "host_id": "00000089"}, {"property_id": "0000000046", "property_address": "101 Elm Lane, Star City, USA", "house_type": "1b1b", "house_size": "199", "price": 417, "availability": 1, "host_id": "00000091"}]
```

(E.g. with front end, house_type = 1b1b)

Filter Properties

| | | | | |
|----------------------|------------------|--------------------|-----------------|---------------------------|
| Property ID | Property Address | 1b1b | House size | House size Greater Than |
| House size Less Than | Price | Price Greater Than | Price Less Than | Availability (true/false) |
| Host ID | Limit | Offset | Apply Filters | |

Get All Properties

[Get All Properties](#)

Properties Results

```
ID: 0000000001, Address: 202 Birch Boulevard, Central City, USA, Type: 1b1b, Size: 102, Price: 135, Availability: 0, Host ID: 00000001
ID: 0000000007, Address: 707 Sycamore Place, Fabletown, USA, Type: 1b1b, Size: 179, Price: 370, Availability: 0, Host ID: 00000013
ID: 0000000014, Address: 606 Sequoia Drive, Emerald City, USA, Type: 1b1b, Size: 105, Price: 417, Availability: 1, Host ID: 00000027
ID: 0000000016, Address: 123 Maple Street, Springfield, USA, Type: 1b1b, Size: 81, Price: 176, Availability: 0, Host ID: 00000035
ID: 0000000021, Address: 505 Redwood Circle, Olympia, USA, Type: 1b1b, Size: 61, Price: 390, Availability: 0, Host ID: 00000042
ID: 0000000022, Address: 202 Birch Boulevard, Central City, USA, Type: 1b1b, Size: 137, Price: 316, Availability: 1, Host ID: 00000044
ID: 0000000034, Address: 456 Oak Avenue, Gotham, USA, Type: 1b1b, Size: 158, Price: 453, Availability: 0, Host ID: 00000074
ID: 0000000038, Address: 404 Fir Terrace, Atlantis, USA, Type: 1b1b, Size: 127, Price: 242, Availability: 1, Host ID: 00000078
ID: 0000000046, Address: 101 Elm Lane, Star City, USA, Type: 1b1b, Size: 199, Price: 417, Availability: 1, Host ID: 00000091
ID: 0000000051, Address: 358 Broome Street New York, NY, Type: 1b1b, Size: 1, Price: 1, Availability: 1, Host ID: 00000002
```

Pagination & offset (realized on Properties)

(E.g. Limit = 5 & Offset = 5 -> we got IDs from 6 to 10, instead of from 1 to 50)

<http://e156-i-am-bezos-402423.ue.r.appspot.com/properties?limit=5&offset=5>

```
[{"property_id": "0000000006", "property_address": "456 Oak Avenue, Gotham, USA", "house_type": "2b2b", "house_size": 116, "price": 168, "availability": 0, "host_id": "00000011"}, {"property_id": "0000000007", "property_address": "707 Sycamore Place, Fabletown, USA", "house_type": "1b1b", "house_size": 179, "price": 370, "availability": 0, "host_id": "00000013"}, {"property_id": "0000000008", "property_address": "789 Pine Road, Metropolis, USA", "house_type": "2b2b", "house_size": 60, "price": 394, "availability": 0, "host_id": "00000016"}, {"property_id": "0000000009", "property_address": "606 Sequoia Drive, Emerald City, USA", "house_type": "3b2b", "house_size": 69, "price": 145, "availability": 1, "host_id": "00000017"}, {"property_id": "0000000010", "property_address": "404 Fir Terrace, Atlantis, USA", "house_type": "2b1b", "house_size": 118, "price": 159, "availability": 1, "host_id": "00000019"}]
```

Filter Properties

| | | | | |
|----------------------|------------------|--------------------|-----------------|---------------------------|
| Property ID | Property Address | House Type | House size | House size Greater Than |
| House size Less Than | Price | Price Greater Than | Price Less Than | Availability (true/false) |
| Host ID | 10 | 10 | Apply Filters | |

Get All Properties

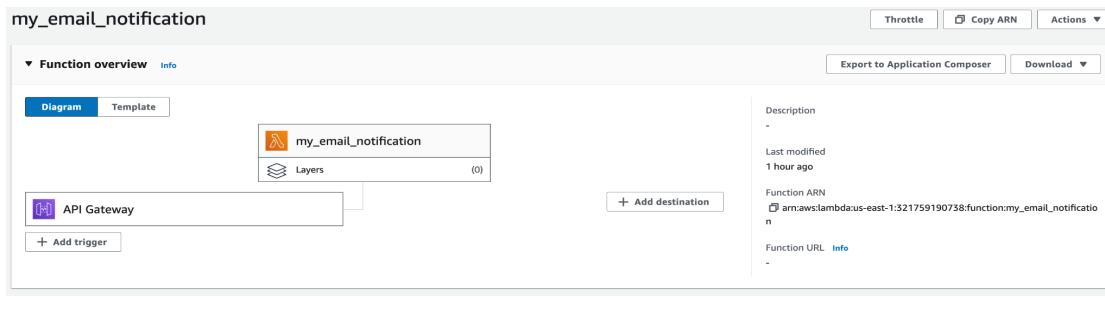
[Get All Properties](#)

Properties Results

```
ID: 0000000011, Address: 202 Birch Boulevard, Central City, USA, Type: 2b1b, Size: 180, Price: 467, Availability: 1, Host ID: 00000020
ID: 0000000012, Address: 123 Maple Street, Springfield, USA, Type: 3b2b, Size: 187, Price: 400, Availability: 0, Host ID: 00000024
ID: 0000000013, Address: 404 Fir Terrace, Atlantis, USA, Type: 2b2b, Size: 101, Price: 461, Availability: 0, Host ID: 00000025
ID: 0000000014, Address: 606 Sequoia Drive, Emerald City, USA, Type: 1b1b, Size: 105, Price: 417, Availability: 1, Host ID: 00000027
ID: 0000000015, Address: 101 Elm Lane, Star City, USA, Type: 3b2b, Size: 190, Price: 433, Availability: 1, Host ID: 00000029
ID: 0000000016, Address: 123 Maple Street, Springfield, USA, Type: 1b1b, Size: 81, Price: 176, Availability: 0, Host ID: 00000035
ID: 0000000017, Address: 101 Elm Lane, Star City, USA, Type: 2b2b, Size: 67, Price: 249, Availability: 1, Host ID: 00000038
ID: 0000000018, Address: 505 Redwood Circle, Olympia, USA, Type: 2b1b, Size: 75, Price: 494, Availability: 1, Host ID: 00000039
ID: 0000000019, Address: 202 Birch Boulevard, Central City, USA, Type: 2b2b, Size: 55, Price: 278, Availability: 1, Host ID: 00000040
ID: 0000000020, Address: 123 Maple Street, Springfield, USA, Type: 2b2b, Size: 141, Price: 212, Availability: 0, Host ID: 00000041
```

Events, Notifications, Pub/Sub:

We implemented a serverless architecture to handle event-driven notifications for new bookings. The system utilizes AWS Lambda, triggered by the API Gateway, to process booking events. Upon a new booking creation, the Lambda function parses the booking details and sends a POST request to an EC2 microservice endpoint to handle the booking logic. After receiving a successful response from the EC2 service, the Lambda function publishes a message to an Amazon Simple Notification Service (SNS) topic, which is configured to send an email notification to the host. This seamless integration of services ensures that the host is promptly notified of new bookings, enhancing the efficiency of the booking system.



```
Code Test Monitor Configuration Aliases Versions

Code source Info

File Edit Find View Go Tools Window Test Deploy

Go to Anything (⌘ P) lambda_function Environment Var

my_email_notification
└── lambda_function.py

1 import boto3
2 import json
3 import json
4
5 def lambda_handler(event, context):
6     # Parse the event
7     booking_data = json.loads(event['body']) # assuming the event body is JSON
8
9     # Prepare the request to EC2 endpoint
10    ec2_endpoint = "ec2-3-144-93-114.us-east-2.compute.amazonaws.com"
11    port = 80
12    url = "/bookings"
13    headers = {"Content-type": "application/json"}
14
15    # Send an HTTP request to the EC2 endpoint
16    connection = http.client.HTTPConnection(ec2_endpoint, port)
17    connection.request("POST", url, json.dumps(booking_data), headers)
18
19    response = connection.getresponse()
20
21    if response.status == 200:
22        # Read and parse the response from EC2
23        response_data = response.read().decode()
24        try:
25            ec2_response_data = json.loads(response_data)
26        except json.JSONDecodeError:
27            # Handle the case where the response is not JSON
28            connection.close()
29            return {
30                "statusCode": 500,
31                "body": json.dumps('Invalid JSON from EC2')
32            }
33
```

```

Error  /tmp/nestly_lambda_function.py
36     # Send an email (using SNS or SES)
37     sns_client = boto3.client('sns')
38     snsArn = "arn:aws:sns:us-east-1:321759190738:nestlyNotification"
39     message = f"Dear host, Great news! We are thrilled to inform you that your property has been successfully booked through Nesty. Details: {ec2_response_data}"
40
41     sns_response = sns_client.publish(
42         TopicArn=snsArn,
43         Message=message,
44         Subject='New Booking Alert: Your Property has been Booked on Nesty!'
45     )
46
47     connection.close()
48     return {
49         'statusCode': 200,
50         'body': json.dumps('Email sent successfully')
51     }
52 else:
53     # Handle errors from EC2 request
54     connection.close()
55     return {
56         'statusCode': response.status,
57         'body': json.dumps('Error communicating with EC2')
58     }
59

```

AWS Notifications <no-reply@sns.amazonaws.com> 00:44 (14小时前) ★ (

发送至 我 ▾

Dear host, Great news! We are thrilled to inform you that your property has been successfully booked through Nesty. Details: Booking created successfully

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:321759190738:nestlyNotification_e32f4b80-237a-4b4a-8f08-1ca9fc4efcf&Endpoint=shiyi010223@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

Composition/Aggregators

Sync: (designated order always)

```

INFO:     Started server process [14318]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8012 (Press CTRL+C to quit)
INFO:     127.0.0.1:52368 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:52368 - "GET /openapi.json HTTP/1.1" 200 OK
users service has returned a response (in sync)
properties service has returned a response (in sync)
bookings service has returned a response (in sync)
INFO:     127.0.0.1:52369 - "GET /sync-aggregator HTTP/1.1" 200 OK
users service has returned a response (in sync)
properties service has returned a response (in sync)
bookings service has returned a response (in sync)

```

Async: different orders

```

INFO:     127.0.0.1:52408 - "GET /async-aggregator HTTP/1.1" 200 OK
users service has returned a response (in async)
properties service has returned a response (in async)
bookings service has returned a response (in async)
INFO:     127.0.0.1:52415 - "GET /async-aggregator HTTP/1.1" 200 OK
bookings service has returned a response (in async)
properties service has returned a response (in async)
users service has returned a response (in async)
INFO:     127.0.0.1:52420 - "GET /async-aggregator HTTP/1.1" 200 OK
bookings service has returned a response (in async)
properties service has returned a response (in async)
users service has returned a response (in async)
INFO:     127.0.0.1:52424 - "GET /async-aggregator HTTP/1.1" 200 OK
users service has returned a response (in async)
bookings service has returned a response (in async)
properties service has returned a response (in async)
INFO:     127.0.0.1:52428 - "GET /async-aggregator HTTP/1.1" 200 OK
bookings service has returned a response (in async)
users service has returned a response (in async)
properties service has returned a response (in async)
INFO:     127.0.0.1:52436 - "GET /async-aggregator HTTP/1.1" 200 OK
bookings service has returned a response (in async)
users service has returned a response (in async)
properties service has returned a response (in async)
INFO:     127.0.0.1:52440 - "GET /async-aggregator HTTP/1.1" 200 OK
users service has returned a response (in async)
bookings service has returned a response (in async)
properties service has returned a response (in async)
INFO:     127.0.0.1:52444 - "GET /async-aggregator HTTP/1.1" 200 OK
bookings service has returned a response (in async)
users service has returned a response (in async)
properties service has returned a response (in async)
INFO:     127.0.0.1:52449 - "GET /async-aggregator HTTP/1.1" 200 OK
properties service has returned a response (in async)
bookings service has returned a response (in async)
users service has returned a response (in async)
INFO:     127.0.0.1:52453 - "GET /async-aggregator HTTP/1.1" 200 OK

```

Authorizer:

We implemented a custom Lambda function authorizer that integrates with AWS API Gateway to manage user authentication and authorization through JWT tokens. Upon receiving an API request, the authorizer extracts and decodes the JWT token to verify the user's identity and role. Depending on the role—whether 'admin', 'guest', or 'host'—the function constructs a policy document that outlines the permitted actions and resources for that user. Admin users are granted full access, while guest and host roles have restricted permissions, limited to operations on their respective user information and viewing properties. The authorizer is also equipped to respond to exceptions by issuing messages for expired or invalid tokens, thereby reinforcing the API's security by enforcing role-based access control.

The screenshot shows two views of the AWS Lambda service. The top view is the 'Function overview' for a function named 'auth'. It displays a diagram showing the function connected to an 'API Gateway' trigger. The bottom view is the 'Code source' editor for the same function. The code editor shows a Python file named 'lambda_function.py' with the following content:

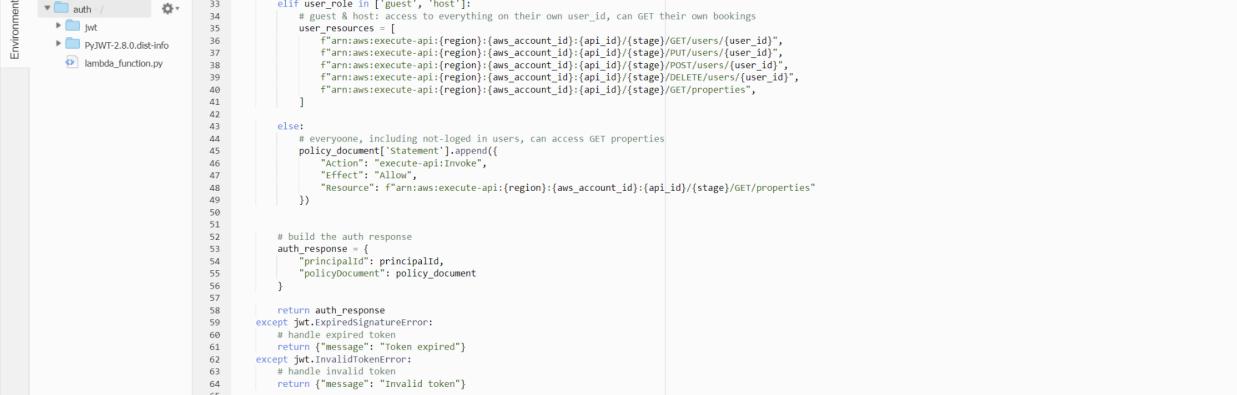
```
import jwt
def lambda_handler(event, context):
    token = event['authorizationToken'].split(' ')[1] # extract the token
    secret_key = "Doritos"
    principalId = 'default_user'

    try:
        decoded = jwt.decode(token, secret_key, algorithms=["HS256"])
        user_role = decoded.get('role')
        user_id = decoded.get('userId')
    except:
        policy_document = {
            "Version": "2012-10-17",
            "Statement": []
        }

    arn = event['methodArn']
    region = arn.split(':')[3]
    aws_account_id = arn.split(':')[4]
    api_id = arn.split(':')[5].split('/')[0]
    stage = arn.split(':')[5].split('/')[1]

    if user_role == 'admin':
        # admin: allow access to everything
        policy_document['Statement'].append({
            "Action": "execute-api:Invoke",
            "Effect": "Allow",
            "Resource": "*"
        })
    else:
        # guest/host: restrict access
        policy_document['Statement'].append({
            "Action": "execute-api:Invoke",
            "Effect": "Deny",
            "Resource": "*"
        })

    return {
        "principalId": principalId,
        "policyDocument": policy_document
    }
```



```

33     elif user_role in ['guest', 'host']:
34         # guest & host: access to everything on their own user_id, can GET their own bookings
35         user_id = event['user_id']
36         policy_document['Statement'].append({
37             "Action": "arn:aws:execute-api:(region):(aws_account_id):(api_id)/(stage)/GET/users/(user_id)",
38             "Effect": "Allow",
39             "Resource": "arn:aws:execute-api:(region):(aws_account_id):(api_id)/(stage)/PUT/users/(user_id)",
40             "Condition": {
41                 "StringLike": {
42                     "aws:sourceIp": event['source_ip'],
43                     "aws:header": {
44                         "Authorization": "Bearer " + event['token']
45                     }
46                 }
47             }
48         })
49     else:
50         # everyone, including not-logged in users, can access GET properties
51         policy_document['Statement'].append({
52             "Action": "execute-api:Invoke",
53             "Effect": "Allow",
54             "Resource": "arn:aws:execute-api:(region):(aws_account_id):(api_id)/(stage)/GET/properties"
55         })
56
57     # build the auth response
58     auth_response = {
59         "principalId": principal_id,
60         "policyDocument": policy_document
61     }
62
63     return auth_response
64 except jwt.ExpiredSignatureError:
65     # handle expired token
66     return {"message": "Token expired"}
67 except jwt.InvalidTokenError:
68     # handle invalid token
69     return {"message": "Invalid token"}
70

```

[API Gateway](#) > [APIs](#) > [nestly \(eqfosdyv30\)](#) > [Authorizers](#) > auth

| auth | |
|---|--|
| Authorizer details | |
| Authorizer ID | Token source |
| 50uawo | Authorization |
| Lambda function | Token validation - optional |
| auth (us-east-1) | None |
| Lambda invoke role - <i>optional</i> | Authorization caching |
| - | 300 seconds |
| Lambda event payload | |
| Token | |
| Test authorizer | |
| Invoke your authorizer with test values and verify the response | |
| Token source | Token value |
| Authorization | Bearer eyJhbGciOiJIUzIwMjNClsInR5cCl6IkpxVCJ9eyJzdWlOIiwMDAwMD |
| <pre> ① Authorizer test: auth 200 Policy { "Version": "2012-10-17", "Statement": [{ "Action": "execute-api:Invoke", "Effect": "Allow", "Resource": "*/*" }] } </pre> | |

API Gateway:

/bookings endpoint allows clients to create a new booking (POST), retrieve a list of all bookings (GET), or delete all bookings (DELETE). It also supports updating booking information (PUT).

{booking_id} is a parameterized endpoint where clients can target a specific booking by its unique identifier to update (PUT) or delete (DELETE) it.

/properties endpoint functions similarly to /bookings, facilitating the creation (POST), retrieval (GET), deletion (DELETE), and updating (PUT) of property records.

`{property_id}` enables specific operations on an individual property identified by its unique ID, allowing for updates (PUT) and deletion (DELETE).

/users endpoint is designed to manage user records, allowing for the creation (POST), retrieval (GET), and deletion (DELETE) of user information, as well as the ability to update user details (PUT) and handle preflight requests for cross-origin resource sharing (OPTIONS).

`{user_id}` allows for targeted operations on a user's record by their unique identifier, enabling updates (PUT) and deletion (DELETE).

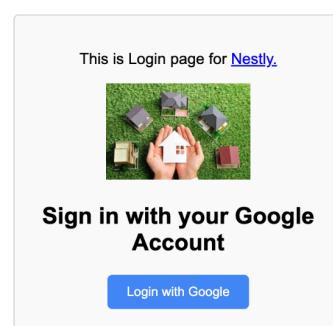
Resources

The screenshot shows a user interface for managing resources. At the top, there is a button labeled "Create resource". Below it, a navigation bar has a single item: a folder icon followed by a slash (/). Underneath this, a tree view of API endpoints is displayed:

- Root level:
 - `/bookings`
 - DELETE**
 - GET**
 - OPTIONS**
 - POST**
 - PUT**
 - `/{booking_id}`
 - DELETE**
 - PUT**
 - `/properties`
 - DELETE**
 - GET**
 - OPTIONS**
 - POST**
 - PUT**
 - `/{property_id}`
 - DELETE**
 - PUT**

```
☒ /users
  DELETE
  GET
  OPTIONS
  POST
  PUT
☒ /{user_id}
  DELETE
  PUT
```

Single Sign-On (SSO):



Authorized request to Google:

← → C ▲ 不安全 | <https://ec2-3-144-182-9.us-east-2.compute.amazonaws.com/auth/callback?code=4%2F0AfJohXkgfq3bHvICSGSd32VXMXFkPQDEd26jbIAog-xHJEs...> ⌂ ⌂

User Information



ID: 100214969192775684788

Email: yw3886@columbia.edu

First Name: Isa

Last Name: Wang

Display Name: Isa Wang

```
INFO: 209.2.224.170:0 - "GET /login HTTP/1.1" 200 OK
INFO: 209.2.224.170:0 - "GET /rent.jpeg HTTP/1.1" 200 OK
INFO: 209.2.224.170:0 - "GET /auth/login HTTP/1.1" 303 See Other
Request = <starlette.requests.Request object at 0x7f861454cf10>
URL = http://localhost:8012/auth/callback?code=4%2F0AfJohXm0_faSL0ANyP5QDe8xzn28smd8g2hFlb960PPrKhdkuaZ3uDA2uWeaePWQ_wROAA&scope=email+profile+openid+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.profile&authuser=0&hd=columbia.edu&prompt=consent
INFO: 209.2.224.170:0 - "GET /auth/callback?code=4%2F0AfJohXm0_faSL0ANyP5QDe8xzn28smd8g2hFlb960PPrKhdkuaZ3uDA2uWeaePWQ_wROAA&scope=email+profile+openid+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.profile&authuser=0&hd=columbia.edu&prompt=consent HTTP/1.1" 200 OK
```

External API:

API: Smarty Street API for Address Validation on Properties microservice.

Functionality: for PUT and POST on Properties, can be successful if address is validated, but returns “Address Invalid” and fails the PUT and POST, if address is not valid.

Success case: (used Valid Address of Mudd Building)

The screenshot shows a web application interface for managing properties. At the top, there's a navigation bar with a lock icon and the URL 'e6156-i-am-bezos-402423.ue.r.appspot.com'. Below the navigation, there are three main sections: 'Get All Properties' (with a 'Get All Properties' button), 'Properties Results' (with a 'Properties Results' heading), and 'Create Property' (with fields for address, unit, square footage, price, and availability). A modal window is open over the 'Properties Results' section, displaying the message 'e6156-i-am-bezos-402423.ue.r.appspot.com 显示' (Showing) and 'Property updated successfully' with a blue '确定' (Confirm) button. The 'Create Property' section contains input fields for address ('500 W 120th St, New York, NY'), unit ('2b3b'), square footage ('400'), price ('5000'), and availability ('Not Available'). There's also a dropdown menu and a 'Create Property' button.

Filter Properties

| | | | | |
|----------------------|------------------------------|--------------------|-----------------|---------------------------|
| Property ID | 500 W 120th St, New York, NY | House Type | House size | House size Greater Than |
| House size Less Than | Price | Price Greater Than | Price Less Than | Availability (true/false) |
| Host ID | 10 | 0 | Apply Filters | |

Get All Properties

Properties Results

ID: 000000052, Address: 500 W 120th St, New York, NY, Type: 2b3b, Size: 400, Price: 5000, Availability: 0, Host ID: 00000001

Fail case (use obviously invalid address of “invalid address”):

The screenshot shows a web application interface. At the top, there's a header bar with a lock icon and the URL 'e6156-i-am-bezos-402423.ue.r.appspot.com'. Below the header, there are three main sections: 'Get All Properties', 'Properties Results', and 'Create Property'. In the 'Create Property' section, there is a text input field with the placeholder 'I know this will fail' which is highlighted with a red rectangle. An arrow points from this field to a modal dialog box that appears over the 'Properties Results' section. The modal box contains the text 'e6156-i-am-bezos-402423.ue.r.appspot.com 显示' (e6156-i-am-bezos-402423.ue.r.appspot.com displays) and 'Invalid address' below it. There is a blue '确定' (Confirm) button at the bottom right of the modal.

Middleware

```

aws Services Search [Option+S]
From github.com:Runze-Lin/Micoservice-Bookings-and-Review
82f7800..917fac4 main -> origin/main
Updating 82f7800..917fac4
Fast-forward
 app.py | 2 ++
 static/user_booking_management.js | 5 +++--
 2 files changed, 4 insertions(+), 3 deletions(-)
[ec2-user@ip-172-31-10-119 Micoservice-Bookings-and-Review]$ python3 app.py
Database connected successfully!
INFO:     Started server process [905790]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8012 (Press CTRL+C to quit)
Request: GET http://ec2-3-144-93-114.us-east-2.compute.amazonaws.com:8012/
Response: Status 307
INFO:     209.2.224.170:51657 - "GET / HTTP/1.1" 307 Temporary Redirect
Request: GET http://ec2-3-144-93-114.us-east-2.compute.amazonaws.com:8012/clientType
Response: Status 307
INFO:     209.2.224.170:51745 - "GET /clientType HTTP/1.1" 307 Temporary Redirect
Request: GET http://ec2-3-144-93-114.us-east-2.compute.amazonaws.com:8012/host
Response: Status 307
INFO:     209.2.224.170:51745 - "GET /host HTTP/1.1" 307 Temporary Redirect
Request: GET http://ec2-3-144-93-114.us-east-2.compute.amazonaws.com:8012/booking_management_host/00000001
Response: Status 307
INFO:     209.2.224.170:51745 - "GET /booking_management_host/00000001 HTTP/1.1" 307 Temporary Redirect
Request: GET http://ec2-3-144-93-114.us-east-2.compute.amazonaws.com:8012/bookings?host_id=00000001
Response: Status 200
INFO:     209.2.224.170:51767 - "GET /bookings?host_id=00000001 HTTP/1.1" 200 OK

```

CI/CD

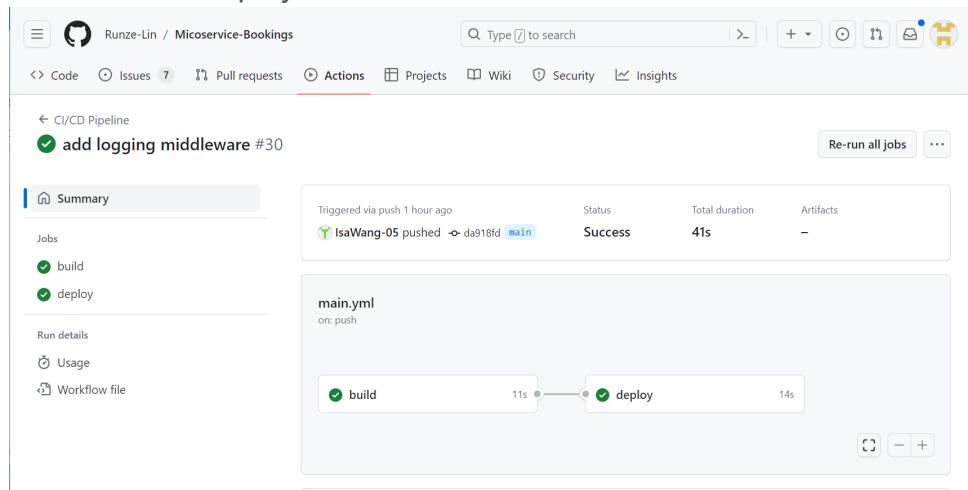
Jobs:

1. Build Job:

- **Environment:** It runs on the latest Ubuntu runner provided by GitHub Actions.
- **Steps:**
 - Checkout: actions/checkout@v2.
 - Python Setup: Sets up Python 3.8
 - Install Dependencies: Upgrades pip and installs pytest along with the packages specified in requirements.txt

2. Deploy Job:

- **Dependency:** It needs the build job to complete first.
- **Condition:** It only runs when a push event happens on the main branch.
- **Environment:** It also runs on the latest Ubuntu runner.
- **Steps:**
 - Checkout: Checks out the code.
 - Install Dependencies
 - SSH Key Setup
 - SSH Keyscan
 - Deploy



The screenshot shows the GitHub Actions interface for the repository "Runze-Lin / Micoservice-Bookings". The left sidebar has sections for Actions, New workflow, All workflows, CI/CD Pipeline, Management, Caches, and Runners. The main area is titled "All workflows" and shows "Showing runs from all workflows". A table lists 16 workflow runs, each with a green checkmark icon, the name of the workflow, the commit hash, the branch (main), the time of the run, and a "..." button. The runs are ordered by event, with the most recent at the top.

| Workflow | Commit | Branch | Time | ... |
|---|------------------------------------|--------|--------------|-----|
| add logging middleware | CI/CD Pipeline #30: Commit da918fd | main | 1 hour ago | ... |
| update host_id insert | CI/CD Pipeline #29: Commit 2cd8a47 | main | 2 hours ago | ... |
| minor bug fix | CI/CD Pipeline #28: Commit 9014698 | main | 8 hours ago | ... |
| removed edit for user | CI/CD Pipeline #27: Commit ddb2303 | main | 8 hours ago | ... |
| update booking_management page and clien... | CI/CD Pipeline #26: Commit b9389df | main | 16 hours ago | ... |
| update host page and host_order_manageme... | CI/CD Pipeline #25: Commit 5293f6e | main | 17 hours ago | ... |
| Update main.yml | CI/CD Pipeline #24: Commit 41d1606 | main | yesterday | ... |

Infrastructure As Code

Use Terraform to create and initialize an EC2 instance in the terminal (`./terraform apply`), then destroy it in the terminal (`./terraform destroy`).

```

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Still creating... [20s elapsed]
aws_instance.example: Still creating... [30s elapsed]
aws_instance.example: Creation complete after 33s [id=i-05d80bde8aa335c24]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
(base)
LilyC@LAPTOP-PV8BHP7E MINGW64 /d/CU/COMS6156/FinalProj/Micoservice-Bookings (base)

```

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

- File Explorer:** Shows a project structure for "MICOSERVICE-BOOKINGS" containing files like .github, .terraform, static, _init_.py, .terraform.lock.hcl, app.py, bookings.py, LICENSE, main.tf, README.md, requirements.txt, terraform.exe, terraform.tfstate, terraform.tfstate.backup, and test.txt.
- Code Editor:** The main.tf file is open, showing Terraform code to create an AWS instance named "example".
- Terminal:** The terminal window shows the execution of the Terraform destroy command, indicating the destruction of one resource over approximately 1m42s.
- Status Bar:** Shows the current file is main*, and the terminal output is from L1yC@LAPTOP-PV8BHP7E MINGW64 /d/CU/COMS6156/FinalProj/Micoservice-Bookings (main).

GraphQL

GraphQL to view the users table Query the id, username, email.

The screenshot shows a GraphQL playground interface with the following details:

- Query:** A GraphQL query to fetch all users, returning their id, username, and email.
- Results:** The results show a list of 10 user objects, each with an id, username, and email. The users are named after animals: Tarantula, Bontebok, Pheasant, common, Azara's zorro, Dusky rattlesnake, Weaver, chestnut, Gazer, sun, and Shrike, southern white-crowned.
- Variables:** A section at the bottom labeled "QUERY VARIABLES" is present but empty.