

# API Docs

👤 Created by	👤 Upal Kundu
🕒 Created time	@February 10, 2025 5:52 PM
🏷️ Tags	

## Install Services:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\19866>
PS C:\Users\19866> docker pull upal113/user-service:latest
latest: Pulling from upal113/user-service
Digest: sha256:0952609e028a7441a0ab5a894ala50a652ae86492e48349310dcc3085c1a59a4
Status: Image is up to date for upal113/user-service:latest
docker.io/upal113/user-service:latest
PS C:\Users\19866> docker run -p 8090:8090 upal113/user-service:latest

:: Spring Boot ::                (v3.4.1)

2025-02-11T03:13:41.878Z INFO 1 --- [remote-pc-control] [main] c.c.c.remotepccontrol.Application : Starting Application v0.0.1-SNAPSHOT using Java 23.0.2 with PID 1 (/app/app.jar started by root in /app)
2025-02-11T03:13:41.880Z INFO 1 --- [remote-pc-control] [main] c.c.c.remotepccontrol.Application : No active profile set, falling back to 1 default profile: "default"
2025-02-11T03:13:43.076Z INFO 1 --- [remote-pc-control] [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JDBC repositories in DEFAULT mode.
2025-02-11T03:13:43.099Z INFO 1 --- [remote-pc-control] [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 15 ms. Found 0 JDBC repository interfaces.
2025-02-11T03:13:43.660Z INFO 1 --- [remote-pc-control] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8090 (http)
2025-02-11T03:13:43.682Z INFO 1 --- [remote-pc-control] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-02-11T03:13:43.683Z INFO 1 --- [remote-pc-control] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.34]
```

Download Docker and run the following commands. It's for running the user service. It runs on port 8090.



```
[
  {
    "userID": "user1",
    "organization": "org1",
    "serialNumber": "SN12345",
    "uniqueID": "abcd1234efgh5678ijkl9101112mno1314151617",
    "emailAddress": "user1@example.com"
  },
  {
    "userID": "user2",
    "organization": "org2",
    "serialNumber": "SN67890",
    "uniqueID": "ijkl1234abcd5678mnop9101112qrst1314151617",
    "emailAddress": "user2@example.com"
  }
]
```

- **Status Code:** 204 No Content
  - If no users are found in the database.
  - **Response Body:** Empty.

---

## 2. POST /users

### Description:

Adds a new user to the database after validating that the provided serial number and organization exist in the VerifiedUsers table.

### Request:

- **Content-Type:** application/json
- **Request Body:**

```
{
  "userID": "user3",
  "organization": "org3",
```

```
"serialNumber": "SN112233",
"uniqueID": "abcd1234efgh5678ijkl9101112mno1314151617",
"emailAddress": "user3@example.com"
}
```

```
('Company A', 'support@companya.com', 'Device123', 'uniqueID123', 'support@c
('Company B', 'support@companyb.com', 'Device456', 'uniqueID456', 'support@
```

For testing, use any of these 2 organizations. Change the uniqueID to a hash, use different users, and keep the same company, email, and device ID.

### Response:

- **Status Code:** 200 OK
  - **Response Body:**

```
{
  "message": "User added successfully!"
}
```

- **Status Code:** 400 Bad Request
  - If the input data fails validation.
  - **Response Body:**

```
{
  "userID": "UserID Can't be blank",
  "organization": "Organization Can't be blank"
}
```

- **Status Code:** 404 Not Found
  - If the Serial Number and Organization are not found in the VerifiedUsers table.
  - **Response Body:**

```
{
  "error": "Serial Number and Organization must exist in the VerifiedUsers table"
}
```

- **Status Code:** 409 Conflict
  - If the `userID` already exists in the database.
  - **Response Body:**

```
{
  "error": "User ID 'user3' already exists."
}
```

- **Status Code:** 500 Internal Server Error
  - If there is an unexpected server error while adding the user.
  - **Response Body:**

```
{
  "error": "Failed to add user"
}
```

### 3. GET /users/checkUser (Needs Update - None needs to use)

#### Description:

Checks if a user exists in the database based on the provided fields: `userID`, `organization`, `serialNumber`, `uniqueID`, and `emailAddress`.

#### Request:

- **Query Parameters:**
  - `userID`: The user ID (e.g., "user3")
  - `organization`: The organization name (e.g., "org3")
  - `serialNumber`: The serial number (e.g., "SN112233")

- **uniqueID** : The unique ID (e.g., "abcd1234efgh5678ijkl9101112mno1314151617")
- **emailAddress** : The user's email address (e.g., "user3@example.com")

### Example Request URL:

```
GET /users/checkUser?userID=user3&organization=org3&serialNumber=SN112233&uniqueID=abcd1234efgh5678ijkl9101112mno1314151617&emailAddress=user3@example.com
```

### Response:

- **Status Code:** 200 OK

- **Response Body:**

```
{
  "message": "User exists in the database."
}
```

- **Status Code:** 404 Not Found

- If the user does not exist in the database.

- **Response Body:**

```
{
  "error": "User not found in the database."
}
```

### Error Handling:

- All errors are handled and returned in a consistent format:

```
{
  "error": "Error message"
}
```

- The following error statuses are used:
    - `400 Bad Request` : When validation or request formatting issues are found.
    - `404 Not Found` : When a user cannot be found based on the provided criteria.
    - `409 Conflict` : When a conflict occurs, such as a duplicate userID.
    - `500 Internal Server Error` : For unexpected server issues.
- 

## Additional Information:

- **Validations:**
    - `userID` , `organization` , `serialNumber` , `uniqueID` , and `emailAddress` must not be blank.
    - `uniqueID` must match the format of a valid hash ( `^[a-fA-F0-9]{64}$` ).
    - The `emailAddress` field should follow a valid email format.
  - **Logging:**
    - All operations are logged, including any errors, actions performed, and status changes.
- 

# Certificate Management API Documentation

## Overview

The `CertificateManagement` service handles the management, storage, and retrieval of certificates using AWS Secrets Manager. It ensures certificates are securely stored and allows authorized retrieval. It also verifies user existence before allowing certificate operations.

---

## Endpoints

### 1. Push Certificate

**Endpoint:** `POST http://localhost:8080/certificates/push`

## Description:

Uploads and updates certificates in AWS Secrets Manager after verifying the user exists.

## Request Body: (JSON)

```
{
  "userID": "string",
  "organization": "string",
  "serialNumber": "string",
  "uniqueID": "string",
  "emailAddress": "string",
  "certificates": {
    "certificate.pem": "base64-encoded-certificate-data"
  }
}
```

## Response:

- **200 OK** : Certificate successfully stored or updated.
- **400 Bad Request** : User does not exist in the system.
- **500 Internal Server Error** : AWS Secrets Manager operation failed.

## 2. Get Certificate

**Endpoint:** **GET** <http://localhost:8080/certificates/get>

## Description:

Retrieves the certificate data from AWS Secrets Manager based on organization, serial number, and unique ID.

## Query Parameters:

Parameter	Type	Required	Description
<b>organization</b>	String	Yes	The user's organization name.



<code>serialNumber</code>	String	Yes	The device or certificate serial number.
<code>uniqueID</code>	String	Yes	A unique identifier for the certificate.

## Response:

- **200 OK** : Returns certificate data.

```
{
  "certificate.pem": "base64-encoded-certificate-data"
}
```

- **404 Not Found** : Certificate not found.
- **500 Internal Server Error** : Error retrieving certificate.

## 3. Download Certificate Files

### Method: Internal Method

### Description:

Writes certificate data to local storage for further processing.

### Usage:

```
certificateManagement.downloadCertificateFiles(certificateData, "/path/to/store");
```

## Error Handling

- **Invalid User**: If a user does not exist, the service logs a warning and does not push certificates.
- **AWS Secret Not Found**: If a secret does not exist, it is created.
- **AWS API Failure**: Any AWS API failures will result in a **500 Internal Server Error**.

## Security Considerations

- Uses **AWS Secrets Manager** to store sensitive certificate data securely.
  - Avoids hardcoding AWS credentials; IAM roles should be used instead.
  - Verifies users before storing certificates to prevent unauthorized storage.
- 

## Dependencies

- **Spring Boot** (REST API)
  - **AWS SDK for Java** (Secrets Manager Integration)
  - **Dotenv** (Environment Variable Management)
  - **Apache HTTP Client** (HTTP Communication)
  - **SLF4J** (Logging)
- 

# DASH Control API Documentation

## Overview

The **DASH Control** API allows remote management of AMD DASH-enabled PCs using command-line tools. It supports checking system status and starting KVM redirection.

---

## Endpoints

### 1. Check System Status

**Endpoint:** **GET** `http://localhost:8080/api/check_status`

#### **Description:**

Checks the status of a remote AMD DASH-enabled PC.

#### **Query Parameters:**

Parameter	Type	Required	Description
<code>host</code>	String	Yes	The target PC's IP or hostname.
<code>user</code>	String	Yes	The username for authentication.
<code>password</code>	String	Yes	The password for authentication.

## Response:

- `200 OK` : Returns the system status output.
- `500 Internal Server Error` : Error executing the DASH CLI command.

## 2. Start KVM Redirection

**Endpoint:** `POST http://localhost:8080/api/start_kvm`

## Description:

Starts a KVM redirection session on a remote AMD DASH-enabled PC.

## Query Parameters:

Parameter	Type	Required	Description
<code>host</code>	String	Yes	The target PC's IP or hostname.
<code>user</code>	String	Yes	The username for authentication.
<code>password</code>	String	Yes	The password for authentication.

## Response:

- `200 OK` : Returns the KVM redirection initiation output.
- `500 Internal Server Error` : Error executing the DASH CLI command.

## Implementation Details

- Uses **ProcessBuilder** to execute DASH CLI commands.
- Commands are executed within `C:\Program Files\DASH CLI 7.0\bin`.
- The command output is captured and returned as a response.

## Security Considerations

- Requires valid authentication credentials for remote execution.
  - Uses HTTPS for secure command execution.
  - Avoids storing sensitive credentials in logs or application memory.
- 

## Dependencies

- **Spring Boot** (REST API framework)
  - **Java ProcessBuilder** (Command execution)
  - **DASH CLI 7.0** (AMD DASH command-line tool)
- 

# Task Queue Management with AWS API Gateway

## Overview

This API allows you to manage task queues using AWS API Gateway and SQS. Tasks can be added to a queue associated with a specific `unique-id` and later retrieved using the same `unique-id`. The API supports error handling for missing parameters and invalid queues.

---

## Base URL

```
https://1dz4oqtvri.execute-api.us-east-2.amazonaws.com/prod
```

---

## Endpoints

### 1. POST `/`

Add a task to the queue.

## Request Body

```
{
  "task": "string",          // The task description
  "priority": "string",      // The priority level (e.g., "high", "low")
  "s3-bucket-url": "string", // The S3 URL for the task, if applicable
  "device-id": "string",    // Unique device identifier
  "unique-id": "string"     // Unique identifier for the queue
}
```

## Response

- **Status Code:** 200 OK
- **Body:**

```
{
  "message": "Task added to queue: <unique-id>"
}
```

- **Possible Errors:**
  - **Status Code:** 400 Bad Request
  - **Body:**

```
{
  "error": "Missing required field: task"
}
```

---

## 2. POST /get-tasks

Retrieve tasks from the queue based on a `unique-id`.

## Request Body

```
{
  "unique-id": "string"    // Unique identifier for the queue
}
```

## Response

- **Status Code:** 200 OK
- **Body:**

```
{
  "message": "Tasks retrieved from queue: <unique-id>",
  "tasks": [
    {
      "MessageId": "string",    // SQS message ID
      "Body": "string",        // Task content
      "Attributes": {          // Additional metadata for the task
        "AttributeName": "value"
      }
    },
    ...
  ]
}
```

- **Possible Errors:**
  - **Status Code:** 400 Bad Request
  - **Body:**

```
{
  "error": "unique-id is required"
}
```

- **Status Code:** 404 Not Found
- **Body:**

```
{
  "error": "Queue with unique-id '<unique-id>' does not exist."
}
```

## Example Usage

### Step 1: Add a Task to the Queue

#### Request:

```
Invoke-RestMethod -Uri "https://1dz4oqtvri.execute-api.us-east-2.amazonaws.com/prod/" -Method Post -ContentType "application/json" -Body '{"task": "Configure", "priority": "high", "s3-bucket-url": "aws:s3/upal/bucket", "device-id": "Device12346789", "unique-id": "e99a18c428cb38d5f260853678922e02"}'
```

#### Response:

```
{
  "message": "Task added to queue: e99a18c428cb38d5f260853678922e02"
}
```

### Step 2: Retrieve Tasks from the Queue

#### Request:

```
Invoke-RestMethod -Uri "https://1dz4oqtvri.execute-api.us-east-2.amazonaws.com/prod/get-tasks" -Method Post -ContentType "application/json" -Body '{"unique-id": "e99a18c428cb38d5f260853678922e02"}'
```

#### Response:

```
{
  "message": "Tasks retrieved from queue: e99a18c428cb38d5f26085367892
2e02",
  "tasks": [
    {
      "MessageId": "91b7...",
      "Body": "Task details here...",
      "Attributes": {
        "AttributeName": "value"
      }
    }
  ]
}
```

## Error Handling

### 1. Missing Required Field ( **unique-id** ):

- **Status Code:** 400 Bad Request
- **Response Body:**

```
{
  "error": "unique-id is required"
}
```

### 2. Queue Not Found ( **unique-id** ):

- **Status Code:** 404 Not Found
- **Response Body:**

```
{
  "error": "Queue with unique-id '<unique-id>' does not exist."
}
```



### 3. Invalid Task Data (e.g., missing `task` field):

- **Status Code:** `400 Bad Request`
- **Response Body:**

```
{  
  "error": "Missing required field: task"  
}
```