Electronics and Computer Science

Faculty of Engineering and Physical Sciences

University of Southampton Malaysia

COMP3207

CLOUD APPLICATION DEVELOPMENT

Task 4: Application Security

- CloudTalk

Name: Hin Jian Heng

Student ID: 33399948

Email: jhh1e22@soton.ac.uk

Lecturers	Dr. Syed Hamid Hussain Madni
	Dr. Muhamad Najib Zamri

Submission Date: 28/4/2025

1. Table of Contents

2. Introduction

As the fourth part of the continuous assessment of COMP3207, this report focuses on the security design and implementation of the CloudTalk application. Based on the requirements of Task 1, the system design of Task 2, and the development practice of Task 3, this task details the various security measures implemented to protect the CloudTalk application and its user data. These measures cover user authentication, access authorisation, encryption of data during transmission and storage, and API communication security, aiming to build a secure and reliable cloud communication platform. In addition, CloudTalk also uses the Kubernetes infrastructure layer based on Google Kubernetes Engine to ensure security, demonstrating how to combine application layer and infrastructure layer security measures to build a deep defence system. This report will explain the implementation details and code examples.

3. Authentication and Authorization

For chat application, security is the most important factor in whether users trust this cloud application. Among them, identity authentication (who is the user) and authorization (what the user can do) are the cornerstones of CloudTalk security. For chat software, no one wants their chat content to be easily revealed and browsed by others, so this section will mainly focus on this part.

3.1 User Authentication

3.1.1 Firebase Authentication (Email/Password, Google Sign-in)

- CloudTalk uses the Firebase Authentication service to handle the core user authentication process.
- Support email/password registration and login: Firebase securely handles password hashing and storage, and I do not need to directly access plain text passwords.
- During signup, email verification will also be performed to ensure that the email account is authentic and belongs to the user.
- Integrate Google Sign-in (OAuth 2.0): Provides a convenient and secure third-party login option to reduce the burden of remembering passwords for users.
- Implementation Description: The front-end implements registration and login through the Firebase SDK, uses createUserWithEmailAndPassword and signInWithEmailAndPassword for authentication, and obtains the Firebase ID Token through getIdToken after login. The back end uses the verifyIdToken method of the Firebase Admin SDK to verify the ID Token sent by the front-end to ensure the request is from authenticated user. The front-end passes the Token as a Bearer Token, and the back end continues to process the user request after successful verification.

3.1.2 Two-Factor Authentication (2FA) - Google Authenticator

- To enhance account security, CloudTalk uses the Google Authenticator app to integrate 2FA based on time-based one-time password (TOTP).
- Users can choose to enable 2FA in the security settings of the user profile page. The activation process includes generating a key, displaying a QR code for users to scan, and verifying the TOTP entered by the user to complete the binding.
- When logging in, users who have enabled 2FA need to enter a dynamic verification code generated by Google Authenticator after entering their password.

Implementation Description:

In this application, the implementation of Google Authenticator relies on the Time-based One-Time Password (TOTP) algorithm, which is a widely used open standard (RFC 6238).

Its technical core lies in:

Shared Secret Key:

- When a user enables 2FA, the system (backend) generates a unique, usually Base32-encoded secret key.

- This secret is securely provided to the user's Google Authenticator application through a QR code (otpauth:// URI generated by the qrcode-generator library).
- The server also stores this secret in a database record associated with the user.

Time Synchronization Factor:

- The core of TOTP is to divide time into fixed Time Steps (30 seconds).
- In each time interval, both the server and the user's Authenticator application use the same shared secret key and the current time interval value as input to apply a hash message authentication code algorithm (HMAC-SHA1).

Dynamic Code Generation:

- The result of the HMAC calculation is a long hash value. This hash value is truncated and processed in a specific way to generate a 6-digit numeric code.
- Because time passes, the time interval changes, so the 6-digit code displayed on the Authenticator app is also updated every 30 seconds.

Verification process:

- When the user needs to verify, they enter the 6-digit code currently displayed by the Authenticator app.
- After receiving the code, the backend server performs the same calculation as the Authenticator app: Get the stored shared secret for the user, get the current time interval, and use HMAC-SHA1 to calculate the expected 6-digit code.
- Compare the code entered by the user and the valid code calculated by the server. If they match, verification is successful.

3.2 Access Control

3.2.1 Role-Based Access Control (RBAC)

- In CloudTalk application, different user roles are defined in each chatroom: Regular User, Group Admin, and Moderator.
- Different roles have different operation permissions, such as group admin can disband the group and appoint moderators; moderators can add/remove members, pin messages, etc.; regular users can only send messages and view messages.
- **Implement Description**: Role information is stored in the Chatroom_User. Backend logic, front end logic, and database rule will check the user's role and permissions before performing sensitive operations.

3.2.2 Firebase Realtime Database Security Rules

- Use Firebase Realtime Database Security Rules to enforce data access permissions at the database level.
- Rules allow or deny read and write operations based on the user's authentication status (auth != null), user ID (auth.uid), and data in the database (such as user roles, group memberships).
- Example: Only group members are allowed to read the group's messages; Only group administrators/moderators are allowed to modify group settings or member lists.

Implementation Description:

Firebase Realtime Database's security rules are a key defence to protect database data from unauthorized access. By setting appropriate rules, we can ensure that only authenticated users can access specific data and have permission control for different data operations. The following is an example that demonstrates how to set security rules for a chatroom application to ensure that only joined users can read and write chat data.

Parts of the Rules in CloudTalk

- In this example, we define a chatrooms node and set an index (indexOn) and access control rules for it. The rules include:
- chatrooms node: set an index based on chatType and createdAt to improve query performance.
- $chatroomId child node: only authenticated users who exist in the chatroom_users child node of the chatroom can access the chatroom data (read and write permissions).

```
{
  "chatrooms": {
    ".indexOn": ["chatType", "createdAt"],
    ".read": true,
    "$chatroomId": {
      ".read": "auth != null &&
root.child('chatroom_users').child($chatroomId).child(auth.uid).exists()",
      ".write": "auth != null &&
root.child('chatroom_users').child($chatroomId).child(auth.uid).exists()",
     …….
      "pending": {
        ".read": "auth != null",
        ".write": "auth != null"
        ……
      },
    }
  }
}
```

3.2.3 Firebase Cloud Storage Security Rules

- Like Realtime Database, Cloud Storage also uses Security Rules to control access to files in a bucket.
- Read and write access control: Rules can decide who can upload (allow write) and download/view (allow read) files based on the user's authentication status (request.auth != null), user ID (request.auth.uid), file metadata (such as the uploader ID or group ID that may be stored in resource.metadata), and the requested file path.

Implementation Description:

- Only allow logged-in users to upload files to their personal directory or a specified group directory.
- Only allow members of a specific group to read files under the group directory.
- File size limit: Security rules allow setting file size limits, for example, request.resource.size < 5 * 1024 * 1024 means that only files smaller than 5MB are allowed to be uploaded. This helps prevent storage abuse and control costs.
- Implementation description: Storage Rules are an important barrier to protect the security of files in cloud storage, preventing unauthorized file access and malicious uploads. Provide the storage.rules file (or its key parts) as an example.

```
service firebase.storage {
  match /b/{bucket}/o {
    match /group-photos/{photoId} {
      allow read: if true;
      allow write: if request.auth != null || request.resource.size < 5 *
1024 * 1024;
    }
    match /{allPaths=**} {
      allow read, write: if true;
    }
  }
```

3.2.4 Backend API Permission Verification (Multi-Layer Verification Mechanism)

- As described in Task 3, for operations that require modifying or obtaining database content, multiple layers of verification are implemented:
- Front-end preliminary check: Determine whether the user has the operation permission based on the local state (basic UI control).
- API Token verification: When the backend Cloud Function receives the request, it must verify the validity of the Firebase ID Token carried in the request header to confirm that the user is logged in.
- Backend logic verification: The backend code checks again based on the business logic whether the user has the permission to perform the specific operation (for example, check whether the user is an administrator of the target group).
- Firebase rule verification: The final database operation must also pass the verification of the Firebase Realtime Database security rules.
- This layered verification provides defence in depth, ensuring that even if one layer is bypassed, other layers can still provide protection.
- For example, when the admin deletes the entire messages and activity log, the deletion operation is verified by the front-end admin, mandatory authorisation by Firebase security rules, an operation confirmation pop-up window, and an audit log record to ensure security.

4. Data Security

4.1 Encryption in Transit (HTTPS/TLS)

- All communications between clients and CloudTalk backend services (Firebase Hosting, Cloud Functions, Realtime Database) are mandatory to use HTTPS/TLS encryption.
- This ensures that data cannot be eavesdropped or tampered with while in transit over the Internet.
- This is a standard feature provided by the Firebase platform and no additional configuration is required.

4.2 Encryption at Rest

4.2.1 Firebase Platform Default Encryption

- Firebase Realtime Database and Cloud Storage automatically encrypt stored data on the server side.
- This is a basic platform-level security guarantee.

4.2.2 Application-Level Encryption (Messages)

Firebase is a **PaaS** platform, so in addition to relying on the transport layer encryption (HTTPS) and static storage encryption provided by the Firebase platform, CloudTalk itself needs to **manage the application layer and data,** so further implements application layer encryption to enhance the confidentiality and integrity of the core communication content is required. For learning purposes, I implemented application layer encryption by designing a custom symmetric key-based encryption protocol instead of directly using the encryption package from library.

Core mechanism:

- The protocol maintains an encryption state (RatchetState) for each chat (group chat or private chat of a specific user), and its core is a rootKey.
- When sending or receiving messages, the protocol uses an increasing message counter (messageCounter) combined with the rootKey and context information to deterministically derive a unique messageKey for each message through SHA256.
- The actual encryption operation uses the combinedKey derived from the groupKey (decrypted from the rootKey) and the messageKey.

Message encryption:

- For text messages, their content is encrypted using AES-CBC mode and the above combinedKey. A randomly generated IV is stored together.
- To ensure the integrity and source authenticity of the message, the original message content (JSON string before encryption) is signed using HMAC-SHA256, and the signing key is also combinedKey.
- The messageContent finally stored in the database is a JSON package containing metadata such as ciphertext, IV, HMAC signature, version number, message sequence

number, etc. The receiver needs to use the same logic to derive messageKey and combinedKey for decryption and verification.

4.2.3 Application-Level Encryption (Files)

File content encryption: Before or during uploading to Firebase Cloud Storage (the specific logic may be in the upload code or backend implementation not shown), the file is encrypted using the groupKey of the corresponding group chat in AES-CBC mode and uploaded to Cloud Storage.

URL encryption: After the file is successfully uploaded to Cloud Storage and its access URL is obtained, the URL itself is not stored in plain text in the message record of Realtime Database. Instead, the URL is encrypted using the same encryption mechanism as the text message.

Purpose: The main purpose of encrypting the URL is to prevent link leakage. Even if an attacker can access the Realtime Database message data, they cannot directly obtain a valid Storage URL pointing to the (possibly encrypted) file without first going through the process of decrypting the message content.

Access process: When a user clicks on a file message, the frontend first decrypts the message content (decryptFileUrl) to get the original Storage URL. The backend then uses this URL to get the file content from Cloud Storage. If the file content itself is also encrypted, after the client obtains the encrypted Blob data, it also needs to call the decryptFile function and use the groupKey to decrypt it before it can finally display or download the file.

4.2.4 Key Management and Rotation

The security of this application layer cryptographic protocol relies on the effective management of the keys involved:

Key types:

- rootKey: The base key for each chat session. The rootKey of a group chat is generated by the system, stored in the database encrypted with the systemKey, and distributed to members (encrypted and stored with each member's private key). The rootKey of a private chat is derived via PBKDF2 based on the "private key" and ID of both users.
- messageKey: A symmetric key for each message, deterministically derived from the rootKey, message counter, and context information via SHA256. It changes for each message.
- combinedKey: The key used for the actual AES encryption and HMAC calculation, generated by combining the groupKey and the messageKey.
- userPrivateKey: This is the key credential for the user's identity and is used to decrypt the groupKey distributed to them.
- systemKey: A global key used to encrypt the original group_key stored in the database. .

Key rotation mechanism:

- messageKey rotation: Since messageKey is derived based on an incrementing message counter, the actual symmetric encryption key (combinedKey) used will change for each message sent or received.
- userPrivateKey rotation: It is triggered when the message_counter of a single user reaches a preset threshold. The function generates a new user private key, encrypts it with a key derived from the user ID, replaces the old key in the database, and resets the counter.

4.3 Sensitive Data Handling (Password Handling)

- CloudTalk does not store user passwords directly. Password verification is handled entirely by Firebase Authentication, which uses industry-standard secure hashing algorithms (such as bcrypt) to store password credentials.

4.4 Kubernetes Infrastructure Security Layer

- Deployment Environment Overview

CloudTalk is deployed in the Google Kubernetes Engine (GKE) environment. This GKE environment enables the implementation of robust infrastructure-level security controls. Key features configured for CloudTalk include network ingress security (HTTPS enforcement, TLS policies), web application firewall integration (Cloud Armor), rate limiting, and secure data persistence strategies, complementing the application-layer security. It should be noted that due to cost considerations (it cost me RM250 in three days), I closed the Google Kubernetes Engine. Currently, all applications are placed on Firebase Hosting and Firebase Cloud Function. However, I didn't remove the relevant deployment code, so only the relevant terminal script needs to be re-written to re-deploy Google Kubernetes Engine.

- Network Ingress Security
    - GKE Ingress Controller
        - As the traffic ingress, it manages external access in a unified way.
    - HTTPS Enforcement and TLS Transmission Encryption
        - Implement HTTP to HTTPS mandatory redirection (redirectToHttps) and prohibit HTTP access (allow-http: false) through Ingress and FrontendConfig.
        - Implement TLS termination through Ingress's tls configuration and cloudtalk-tls Secret to ensure the security of data transmission from the client to the ingress.
    - TLS Security Policy:
        - Use FrontendConfig's sslPolicy to enforce GKE's recommended TLS 1.2+ security configuration and disable weak encryption algorithms and protocols.
    - Rate Limiting:
        - Use FrontendConfig's rateLimit function to limit the frequency of requests from a single source to mitigate DoS attacks and brute-force attempts.
    - Web Application Firewall (WAF - Cloud Armor):
        - Through BackendConfig's securityPolicy, associate Google Cloud Armor policy to provide firewall protection. Cloud Armor provides strong protection against common web attacks such as OWASP Top 10, SQL injection, and XSS.
- Data Persistence Security:

  - Data at Rest Encryption: The Persistent Volumes (PVs) utilized by the Redis StatefulSet rely on the underlying cloud provider's storage service (e.g., Google Cloud Persistent Disk) for encryption at rest by default. This infrastructure-level encryption protects the confidentiality of persisted data (like Redis cache state) should the physical storage media be compromised.
- Availability & Resilience Design - Indirect Security
  - High Availability for Stateful Services (Redis): The design included deploying Redis as a `StatefulSet` with multiple `replicas`. This ensures higher availability and fault tolerance for the caching layer, meaning the failure of a single Redis instance would not bring down the entire caching service.
  - Scalability and Controlled Updates for GKE Workloads: Kubernetes features like the Horizontal Pod Autoscaler (HPA) were designed to automatically scale stateless `Deployments` (like the planned `cloudtalk-app` service, if handling significant load within GKE) based on resource utilization or custom metrics. For services exposed via GKE Ingress, `BackendConfig` options like `connectionDraining` were included in the design to ensure graceful shutdown of pods during updates, maintaining stability for those specific GKE-hosted services. Stateful workloads like the designed Redis cluster utilize their own update strategies (e.g., `RollingUpdate` for StatefulSets) for controlled rollouts.

5. API Security

- The main purpose of API Security is to protect backend API endpoints from unauthorized access and abuse

5.1 HTTPS Communication

- All API requests are provided through Firebase Cloud Functions and are forced to use HTTPS.

5.2 API Authentication and Authorization (Token Validation)

- When calling the backend API, the client (Nuxt.js frontend) must include a valid Firebase ID Token (JWT) in the request header.
- Each Cloud Function will first verify the signature and validity period of this token and extract the user ID (uid) from it before executing the business logic.
- Then perform subsequent permission checks based on uid and request parameters
- Implementation example (upload.post.ts)
  - Authorization header verification: This function checks for the presence of an Authorization header with a Bearer token.
  - ID token verification: Extract and verify the token using the verifyIdToken method of the Firebase Admin SDK. If the token is invalid or missing, an error is thrown. This ensures that only authenticated users can proceed with the request.

```
export default defineEventHandler(async (event) => {
  // 1. Authorization
  const authHeader = getHeader(event, "Authorization");
  if (!authHeader?.startsWith("Bearer ")) {
    throw new Error("Unauthorized: No ID Token provided");
  }

  const idToken = authHeader.split("Bearer ")[1];
  let userId: string;
  try {
    userId = (await adminAuth.verifyIdToken(idToken)).uid;
  } catch (error) {
    throw new Error("Unauthorized: Invalid ID Token");
  }

  // ...
  // Additional logic related to form data, encryption, file upload, etc.
  // ...
});
```

6. Code Demonstration

- Codebase Link: https://github.com/JianHengHin0831/Cloutalk
- Key security implementation files/directories:
  - Firebase Realtime Database security rules: database.rules.json
  - Firebase Cloud Storage security rules: storage.rules
  - Backend API (Cloud Functions): Related files in the server/api/ directory (e.g. api/groups/[groupId]/disband.put.ts, which is the function for disbanding a group)
  - Front-end authentication logic:
  - The part of the Nuxt.js page/component that calls the Firebase Auth SDK
  - 2FA related components/logic (front-end settings page, back-end authHelper.js part)
  - Encryption/decryption tool functions/modules (in chatroom-service.js, service/api/featchFile.post.ts and other similar files)
- Code snippet examples:
  - Firebase Database Rule example

```
{
  "chatrooms": {
    ".indexOn": ["chatType", "createdAt"],
    ".read": true,
    "$chatroomId": {
      ".read": "auth != null &&
root.child('chatroom_users').child($chatroomId).child(auth.uid).exists()",
      ".write": "auth != null &&
root.child('chatroom_users').child($chatroomId).child(auth.uid).exists()",
     …….
      "pending": {
        ".read": "auth != null",
        ".write": "auth != null"
        ……
      },
    }
  }
}
```

- Firebase Storage Rule example:

```
service firebase.storage {
  match /b/{bucket}/o {
    match /group-photos/{photoId} {
      allow read: if true;
      allow write: if request.auth != null || request.resource.size < 5
* 1024 * 1024;
    }
    match /{allPaths=**} {
      allow read, write: if true;
    }
```

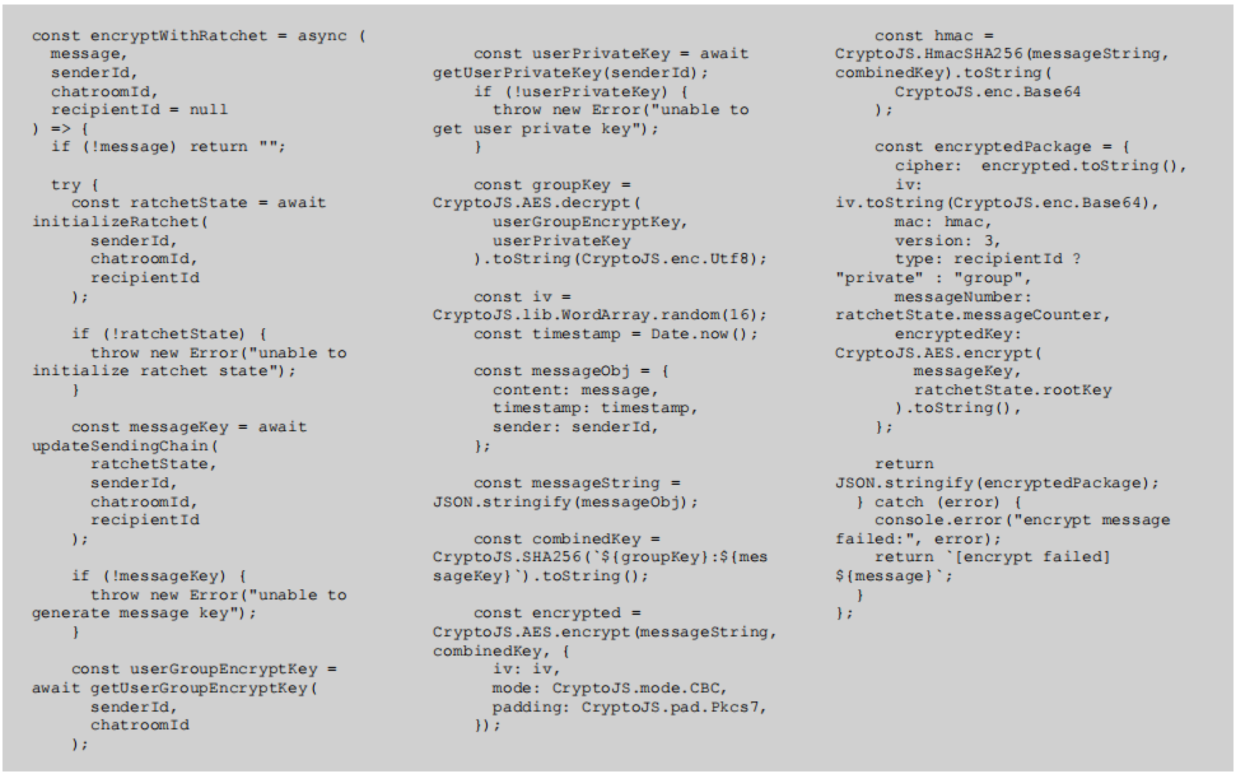
- Cloud Function Token verification example:

```
export default defineEventHandler(async (event) => {
  // 1. Authorization
  const authHeader = getHeader(event, "Authorization");
  if (!authHeader?.startsWith("Bearer ")) {
    throw new Error("Unauthorized: No ID Token provided");
  }

  const idToken = authHeader.split("Bearer ")[1];
  let userId: string;
  try {
    userId = (await adminAuth.verifyIdToken(idToken)).uid;
  } catch (error) {
    throw new Error("Unauthorized: Invalid ID Token");
  }

  // ...
  // Additional logic related to form data, encryption, file upload,
etc.
  // ...
});
```

- Application layer encryption call example:



```
const encryptWithRatchet = async (
  message,
  senderId,
  chatroomId,
  recipientId = null
) => {
  if (!message) return "";

  try {
    const ratchetState = await
initializeRatchet(
      senderId,
      chatroomId,
      recipientId
    );

    if (!ratchetState) {
      throw new Error("unable to
initialize ratchet state");
    }

    const messageKey = await
updateSendingChain(
      ratchetState,
      senderId,
      chatroomId,
      recipientId
    );

    if (!messageKey) {
      throw new Error("unable to
generate message key");
    }

    const userGroupEncryptKey =
await getUserGroupEncryptKey(
      senderId,
      chatroomId
    );

    const userPrivateKey = await
getUserPrivateKey(senderId);
    if (!userPrivateKey) {
      throw new Error("unable to
get user private key");
    }

    const groupKey =
CryptoJS.AES.decrypt(
      userGroupEncryptKey,
      userPrivateKey
    ).toString(CryptoJS.enc.Utf8);

    const iv =
CryptoJS.lib.WordArray.random(16);
    const timestamp = Date.now();

    const messageObj = {
      content: message,
      timestamp: timestamp,
      sender: senderId,
    };

    const messageString =
JSON.stringify(messageObj);

    const combinedKey =
CryptoJS.SHA256(`${groupKey}:${mes
sageKey}`).toString();

    const encrypted =
CryptoJS.AES.encrypt(messageString,
combinedKey, {
      iv: iv,
      mode: CryptoJS.mode.CBC,
      padding: CryptoJS.pad.Pkcs7,
    });

    const hmac =
CryptoJS.HmacSHA256(messageString,
combinedKey).toString(
      CryptoJS.enc.Base64
    );

    const encryptedPackage = {
      cipher:  encrypted.toString(),
      iv:
iv.toString(CryptoJS.enc.Base64),
      mac: hmac,
      version: 3,
      type: recipientId ?
"private" : "group",
      messageNumber:
ratchetState.messageCounter,
      encryptedKey:
CryptoJS.AES.encrypt(
        messageKey,
        ratchetState.rootKey
      ).toString(),
    };

    return
JSON.stringify(encryptedPackage);
  } catch (error) {
    console.error("encrypt message
failed:", error);
    return `[encrypt failed]
${message}`;
  }
};
```

- 2FA verification example:

```
function verifyTOTPCode(secret, code) {
  const timeStep = 30;
  const window = 1;
  const now = Math.floor(Date.now() / 1000);

  const currentCode = generateTOTPCode(secret, now);
  const prevCode = generateTOTPCode(secret, now - window);
  const nextCode = generateTOTPCode(secret, now + window);

  return code === currentCode || code === prevCode || code === nextCode;
}
```

7. Overview of CloudTalk Security

Table 7 describes all the security deployments of CloudTalk, but currently all Kubernetes-related security deployments are disabled due to cost considerations. Although the GKE part is currently inactive, this report still retains the description of its design and configuration to fully demonstrate the originally planned multi-layered defense-in-depth security architecture. The relevant Kubernetes deployment code and configuration files are still retained, and this infrastructure layer can be activated by redeploying in the future when the cost is controllable.

Security Measure / Implementation	**Layer**	**Purpose / Protection Provided**
User Identity & Authentication		
Firebase Authentication (Email/Password, Google Sign-in)	Application	Securely authenticate users, manage user sessions, and handle password hashes.
Two-Factor Authentication (2FA - Google Authenticator/TOTP)	Application	Add an extra layer of security to user logins, mitigating the possibility of stolen credentials.
Email Verification	Application	Make sure the email address used is real and belongs to user
Access Control & Authorization		
Role-Based Access Control (RBAC - Custom Logic)	Application	Grant permissions based on the user's role in the chat room (admin, moderator, user).
Firebase Realtime Database Security Rules	Application	Enforce fine-grained read/write permissions at the database level based on authentication status/data.
Firebase Cloud Storage Security Rules	Application	Enforce fine-grained read/write/upload permissions for files based on authentication status/metadata/path.
Backend API Multi-Layer Verification	Application	Defense in Depth for API operations: Front-end Check -> API Token Authentication -> Back-end Logic -> Database Rules. Reduce the possibility of users bypassing verification and obtaining data directly.
Data Protection - In Transit		
HTTPS/TLS (Firebase Hosting/Functions Default)	Application	Application Layer Encrypts communications between clients and Firebase backend services.
HTTPS Forced (GKE Ingress redirectToHttps)	Infrastructure	Ensures that all external traffic accessing the application uses encrypted HTTPS connections.

TLS Termination & Strong Policy (Ingress + sslPolicy)	Infrastructure	Securely handle TLS encryption at the edge of the network, using strong protocols (TLS 1.2+) and cipher suites.
Data Protection - At Rest		
Firebase Platform Default Encryption	Application	Encrypt data stored in Firebase Realtime Database and Cloud Storage (server-side)
Cloud Provider Storage Encryption (e.g., for PVs)	Infrastructure	Encrypt persistent data stored on the underlying cloud disk (e.g. Redis data).
Application-Level Message Encryption (AES-CBC + HMAC)	Application	End-to-end encryption of message content; ensure confidentiality and integrity even if the database is accessed. Dynamically change the key. Even if the key of one message is stolen, that message can only be seen, and cannot see other messages before and after it
Application-Level File URL Encryption	Application	Encrypts the file storage URL stored in the database; prevents the valid URL from being directly obtained when the database is accessed.
Application-Level File Content Encryption	Application	Encrypt file content before uploading; ensure confidentiality even if the bucket is accessed.
Secure Password Handling (Delegated to Firebase Auth)	Application	Avoid storing plain text passwords; use secure hashing algorithms (such as bcrypt).
API & Network Security		
API Authentication (Firebase ID Token Verification)	Application	Protect backend Cloud Functions/API endpoints and ensure that calls come from authenticated users.
Web Application Firewall (WAF - Cloud Armor)	Infrastructure	Defend against common web attacks (such as SQL injection, XSS, OWASP Top 10 vulnerabilities) at the edge of the network.
Rate Limiting (GKE FrontendConfig)	Infrastructure	Mitigate Denial of Service (DoS) attacks and abusive request patterns.
Internal Service Isolation (K8s Service Types)	Infrastructure	Limit direct external access to internal components (such as database Redis).
Infrastructure & Configuration Security		
Secret Management (Kubernetes Secrets)	Infrastructure	Securely store and manage sensitive data such as TLS keys, API credentials, and database passwords.

Container Resource Limits	Infrastructure	Prevent a single container from exhausting resources and improve stability and availability.
Container Health Checks (Liveness/Readiness Probes)	Infrastructure	Ensure traffic is routed only to healthy application instances to improve reliability.
High Availability & Auto-Scaling (Replicas + HPA)	Infrastructure	Enhance resilience to failures and load peaks to maintain service availability.

Table 7 Overview of CloudTalk Security

8. Conclusion

In summary, CloudTalk application pays great attention to security in the design and development phase to improve the reliability of the system and the trust of users. First, I use Google Kubernetes Engine (GKE) to deploy the application and implement infrastructure security upon it. However, due to the cost consideration, I close the GKE and just use Firebase as PaaS, the code related to GKE haven't been removed and can re-deploy again if cost is not a consideration. Then, we use the security services provided by Firebase, such as authentication, real-time database and storage rules, and hosting/functional HTTPS to protect CloudTalk, which allows us to be very well protected, especially in terms of network, server, and storage. In addition, we have built a multi-layer security protection system. These key internal security measures include two-factor authentication (2FA), fine-grained role-based access control (RBAC), multi-layer verification mechanism, and customized application layer symmetric key encryption protocol to effectively protect the application layer.

This encryption protocol is designed to improve the confidentiality of message content. It borrows the idea of ratchet protocol by managing state (such as message counter) and deriving a unique encryption key for each message. At the same time, the use of HMAC ensures the integrity and authenticity of the message, thereby enhancing security.

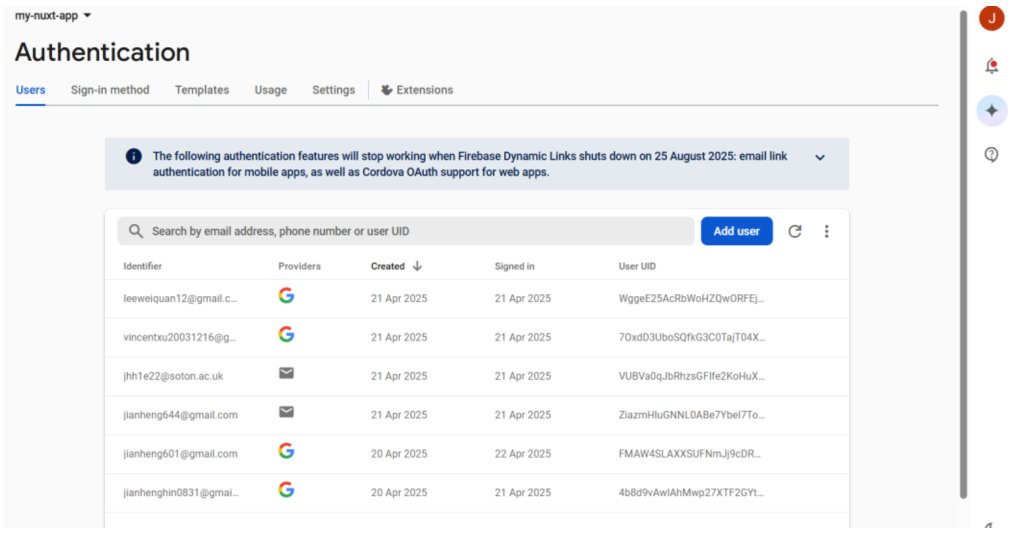
Since this coursework is for learning purposes, I did not import the existing encryption package from library directly but customized my own encryption protocol. Implementing such a complex custom encryption protocol is a huge challenge, while I strive to ensure the robustness of the design, any custom encryption protocol needs to be verified and audited before it can be used.

In summary, CloudTalk takes a variety of measures to ensure user security. While there is no absolute security, the combination of these measures is designed to maximize the confidentiality, integrity, and availability of user accounts, data, and communication content.

9. Appendix

A1. Firebase Authentication

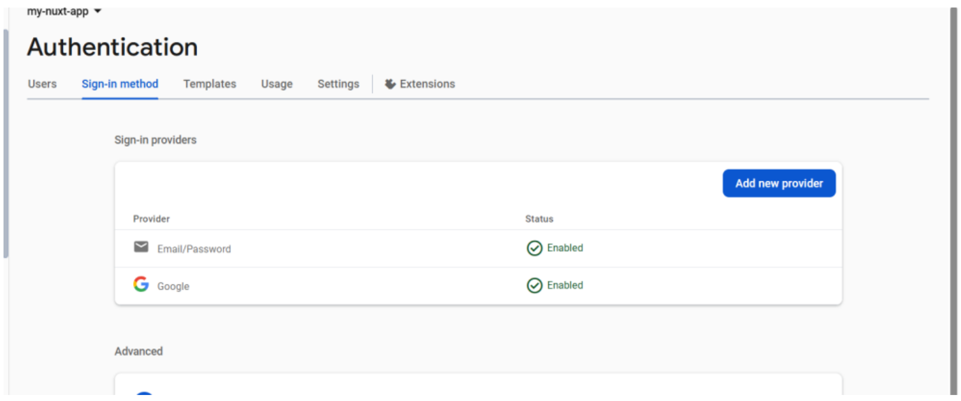
Firebase Authentication Screenshot



The following authentication features will stop working when Firebase Dynamic Links shuts down on 25 August 2025: email link authentication for mobile apps, as well as Cordova OAuth support for web apps.

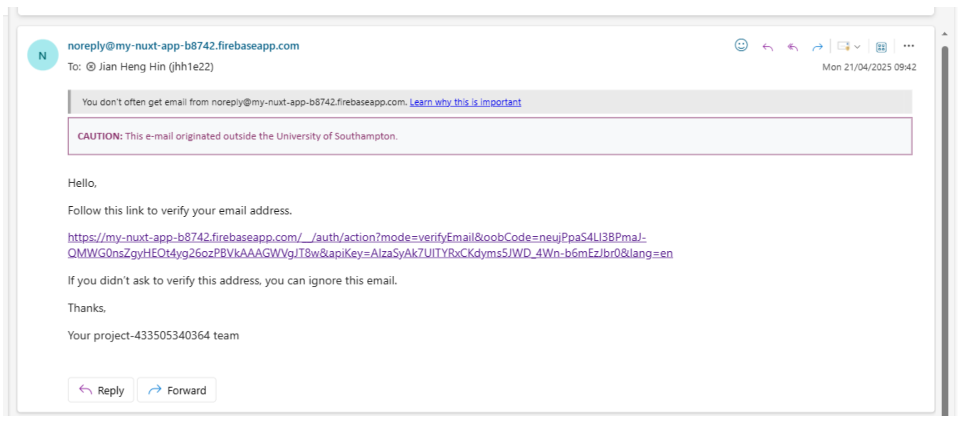
Search by email address, phone number or user UID

Identifier	Providers	Created	Signed in	User UID
leeweiquan12@gmail.c...	G	21 Apr 2025	21 Apr 2025	WggeE25AcRbWoHZQwORFEj...
vincentxu20031216@g...	G	21 Apr 2025	21 Apr 2025	7OxdD3UboSQfkG3C0TajT04X...
jhh1e22@soton.ac.uk	✉	21 Apr 2025	21 Apr 2025	VUBVa0qJbRhzsGFIfe2KoHuX...
jianheng644@gmail.com	✉	21 Apr 2025	21 Apr 2025	ZiazmHluGNNL0ABe7YbeI7To...
jianheng601@gmail.com	G	20 Apr 2025	22 Apr 2025	FMAW4SLAXXSUFNmJj9cDR...
jianhenghin0831@gmai...	G	20 Apr 2025	21 Apr 2025	4b8d9vAwIAhMwp27XTF2GYt...



Provider	Status
Email/Password	✓ Enabled
Google	✓ Enabled

Email Verification



noreply@my-nuxt-app-b8742.firebaseapp.com

To: Jian Heng Hin (jhh1e22)

Mon 21/04/2025 09:42

You don't often get email from noreply@my-nuxt-app-b8742.firebaseapp.com. Learn why this is important

CAUTION: This e-mail originated outside the University of Southampton.

Hello,

Follow this link to verify your email address.

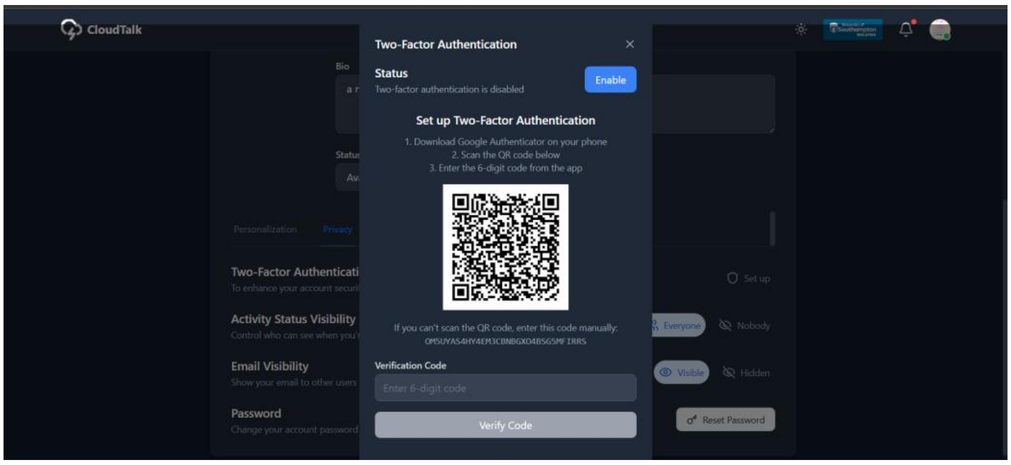
https://my-nuxt-app-b8742.firebaseapp.com/__/auth/action?mode=verifyEmail&oobCode=neujPpaS4LI3BPmaJ-QMWG0nsZgyHEOt4yg26ozPBVkAAAGWVgJT8w&apiKey=AIzaSyAk7UITYRxCKdyms5JWD_4Wn-b6mEzJbr0&lang=en

If you didn't ask to verify this address, you can ignore this email.

Thanks,

Your project-433505340364 team

A2 Google Authenticator


CloudTalk

Two-Factor Authentication

Status
Two-factor authentication is disabled

Set up Two-Factor Authentication

1. Download Google Authenticator on your phone
2. Scan the QR code below
3. Enter the 6-digit code from the app

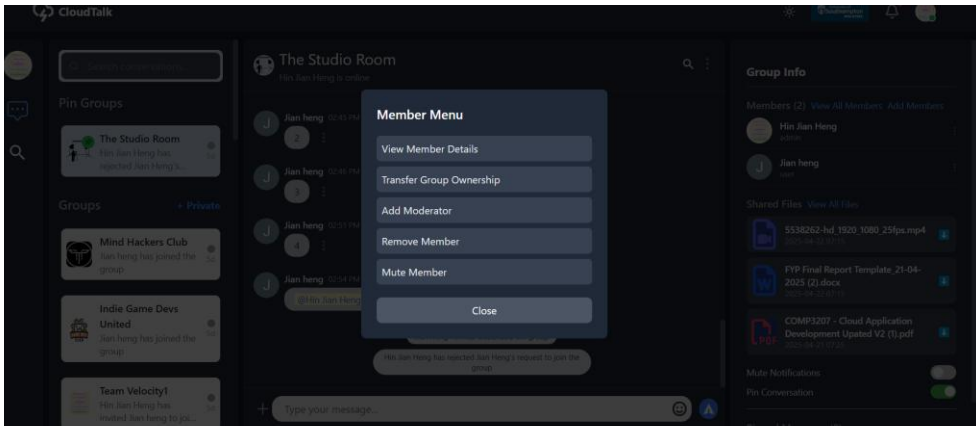
If you can't scan the QR code, enter this code manually:
OM5UYAS4HY4EM3CBNBGXO4BSG5MFIRRS

Verification Code

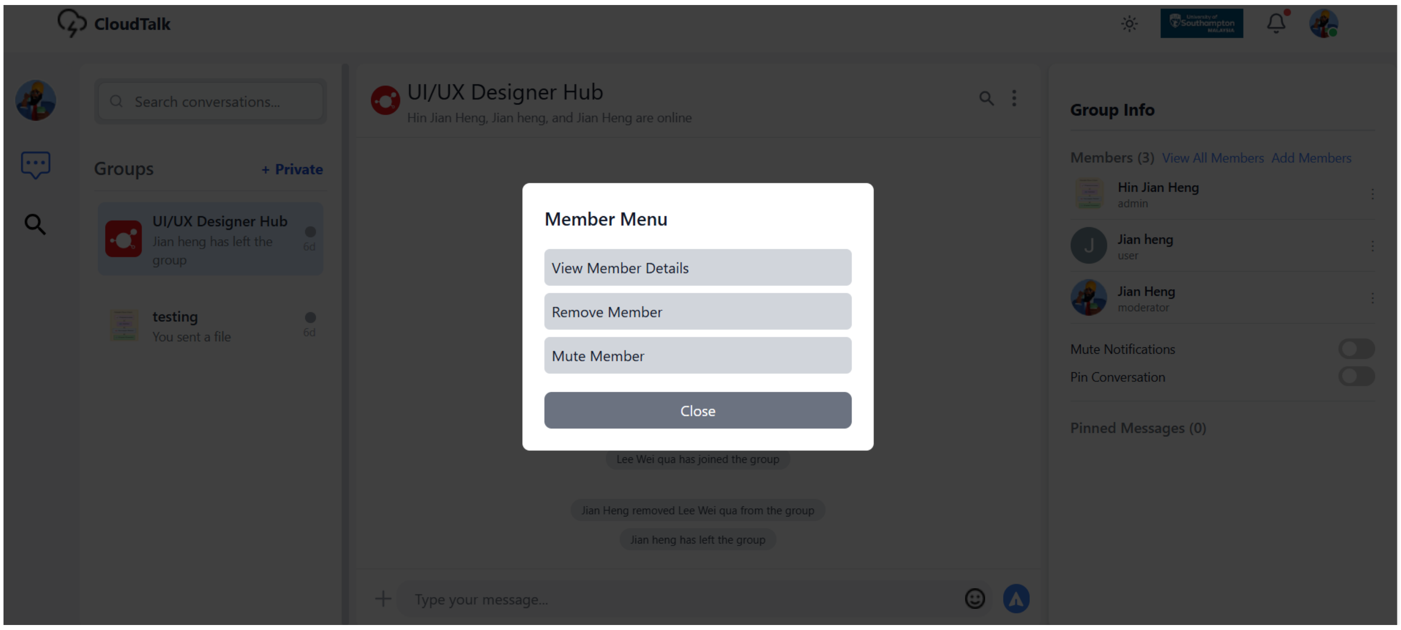
Two-Factor Authentication — To enhance your account security
Activity Status Visibility — Control who can see when you're
Email Visibility — Show your email to other users
Password — Change your account password

A3. Role Based Access Control

Example of Admin Permissions

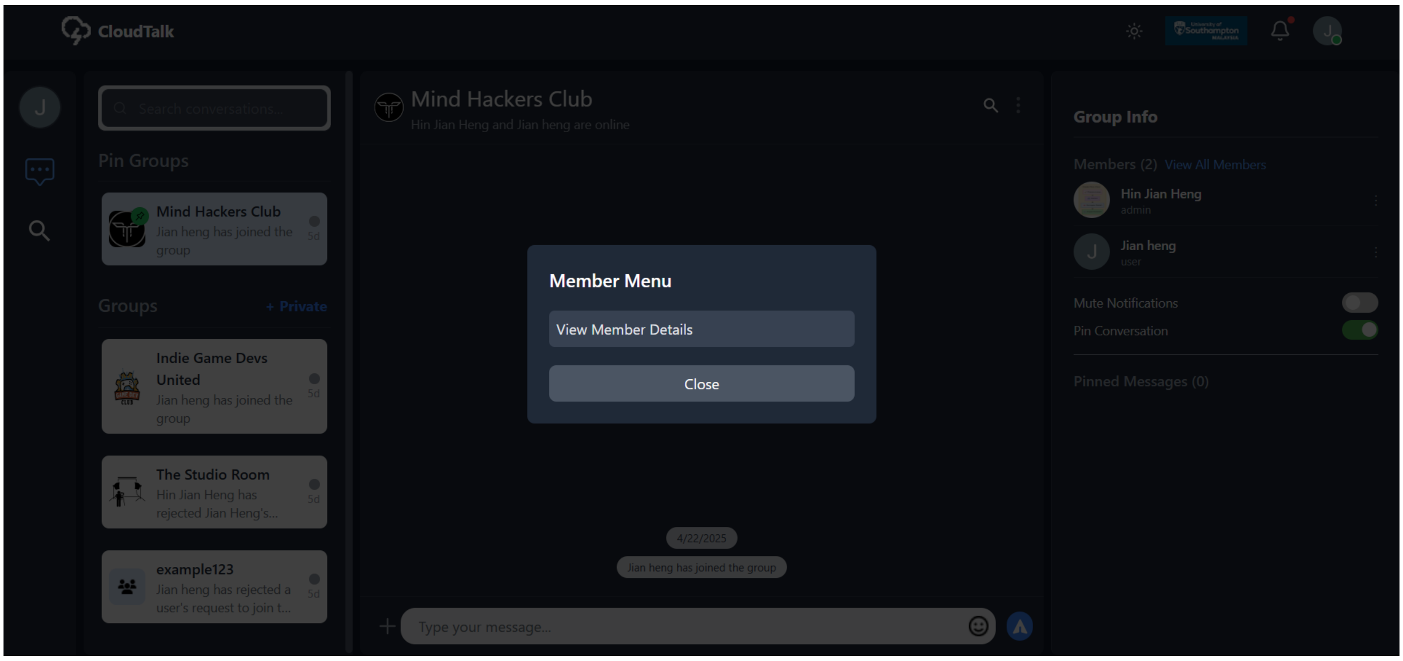


Example of Moderator Permissions


UI/UX Designer Hub
Hin Jian Heng, Jian heng, and Jian Heng are online

Lee Wei qua has joined the group
Jian Heng removed Lee Wei qua from the group
Jian heng has left the group

Example of User Permissions



Mind Hackers Club
Hin Jian Heng and Jian heng are online

Mind Hackers Club — Jian heng has joined the group — 5d
Indie Game Devs United — Jian heng has joined the group — 5d
The Studio Room — Hin Jian Heng has rejected Jian Heng's... — 5d
example123 — Jian heng has rejected a user's request to join t... — 5d

Member Menu

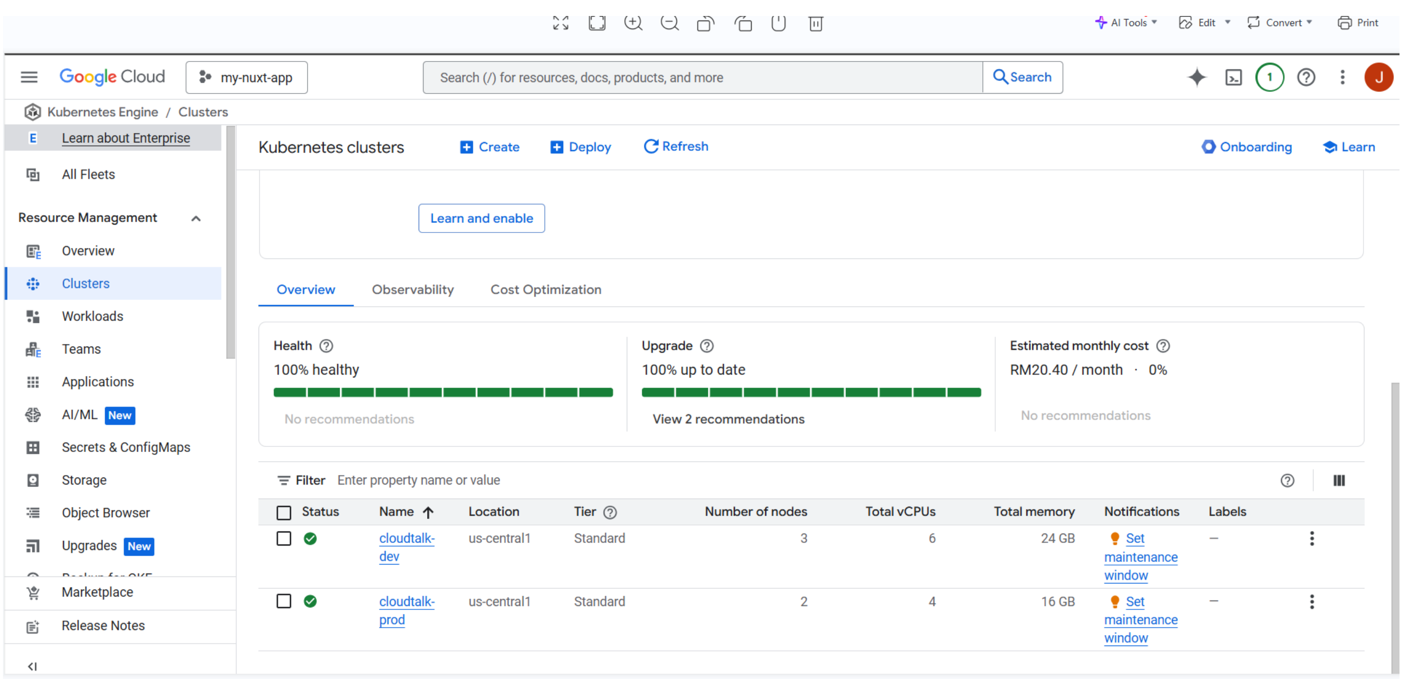
View Member Details

Close

4/22/2025
Jian heng has joined the group

A4 Google Kubernetes Engine and Virtual Machine

Google Kubernetes Engine



Kubernetes clusters

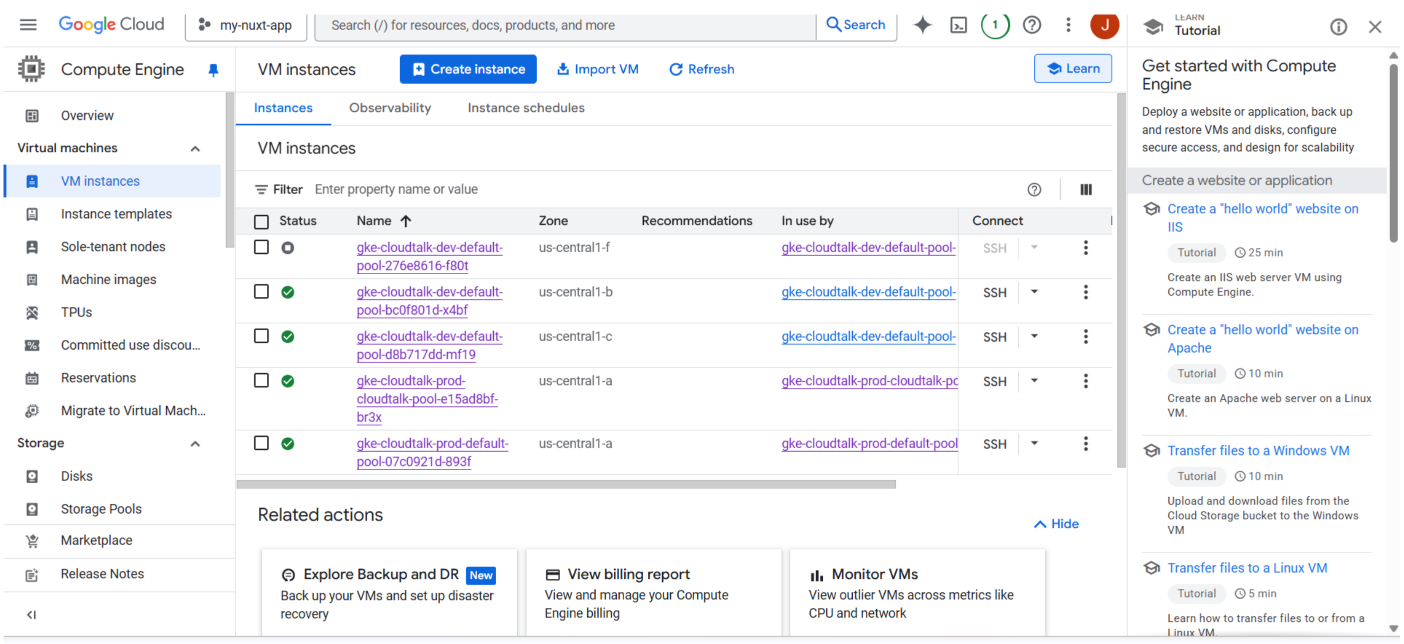
Health: 100% healthy — No recommendations

Upgrade: 100% up to date — View 2 recommendations

Estimated monthly cost: RM20.40 / month · 0% — No recommendations

Status	Name	Location	Tier	Number of nodes	Total vCPUs	Total memory	Notifications	Labels
✓	cloudtalk-dev	us-central1	Standard	3	6	24 GB	Set maintenance window	—
✓	cloudtalk-prod	us-central1	Standard	2	4	16 GB	Set maintenance window	—

Virtual Machine Instance



VM instances

Status	Name	Zone	Recommendations	In use by	Connect
○	gke-cloudtalk-dev-default-pool-276e8616-f80t	us-central1-f		gke-cloudtalk-dev-default-pool-	SSH
✓	gke-cloudtalk-dev-default-pool-bc0f801d-x4bf	us-central1-b		gke-cloudtalk-dev-default-pool-	SSH
✓	gke-cloudtalk-dev-default-pool-d8b717dd-mf19	us-central1-c		gke-cloudtalk-dev-default-pool-	SSH
✓	gke-cloudtalk-prod-cloudtalk-pool-e15ad8bf-br3x	us-central1-a		gke-cloudtalk-prod-cloudtalk-pc	SSH
✓	gke-cloudtalk-prod-default-pool-07c0921d-893f	us-central1-a		gke-cloudtalk-prod-default-pool	SSH

Related actions

Hide

Explore Backup and DR New
Back up your VMs and set up disaster recovery

View billing report
View and manage your Compute Engine billing

Monitor VMs
View outlier VMs across metrics like CPU and network

Get started with Compute Engine
Deploy a website or application, back up and restore VMs and disks, configure secure access, and design for scalability

Create a website or application

Create a "hello world" website on IIS
Tutorial 25 min
Create an IIS web server VM using Compute Engine.

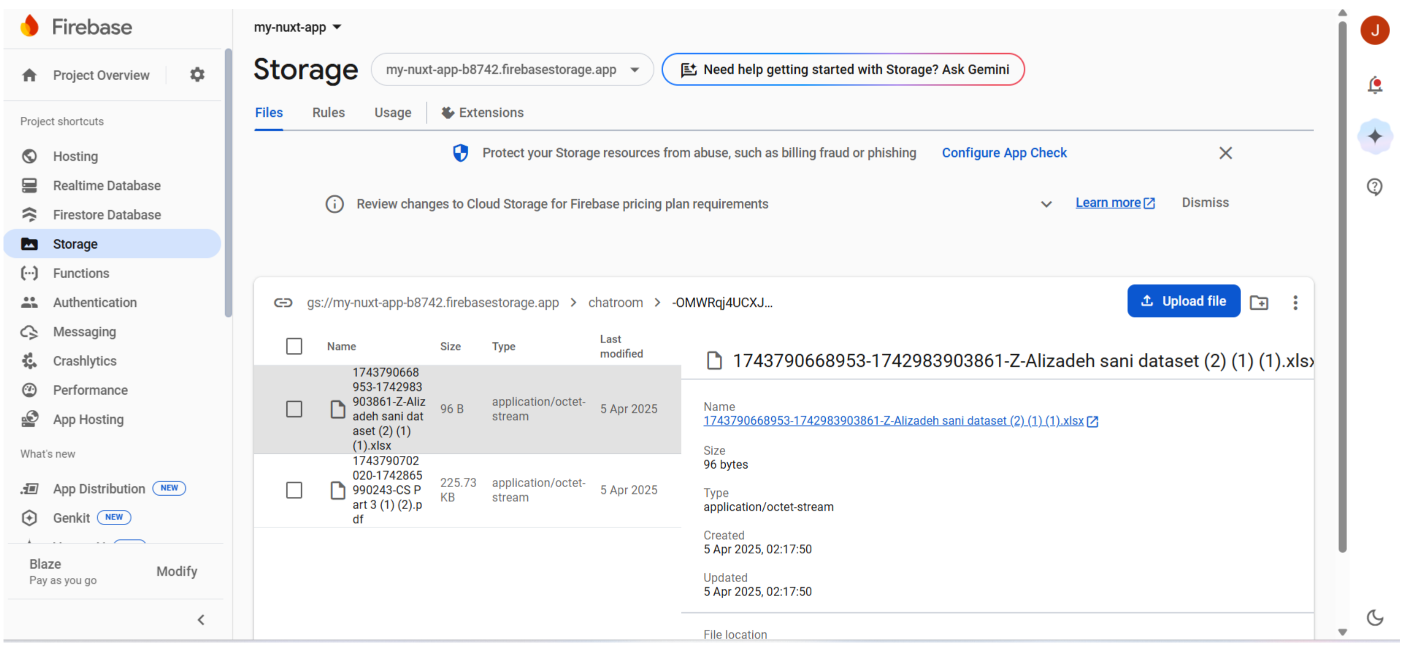
Create a "hello world" website on Apache
Tutorial 10 min
Create an Apache web server on a Linux VM.

Transfer files to a Windows VM
Tutorial 10 min
Upload and download files from the Cloud Storage bucket to the Windows VM

Transfer files to a Linux VM
Tutorial 5 min
Learn how to transfer files to or from a Linux VM

A5 End-to-End Encryption

File Encryption



Protect your Storage resources from abuse, such as billing fraud or phishing Configure App Check

Review changes to Cloud Storage for Firebase pricing plan requirements Learn more Dismiss

gs://my-nuxt-app-b8742.firebasestorage.app > chatroom > -OMWRqj4UCXJ...

Name	Size	Type	Last modified
1743790668953-1742983903861-Z-Alizadeh sani dataset (2) (1) (1).xlsx	96 B	application/octet-stream	5 Apr 2025
1743790702020-1742865990243-CS Part 3 (1) (2).pdf	225.73 KB	application/octet-stream	5 Apr 2025

1743790668953-1742983903861-Z-Alizadeh sani dataset (2) (1) (1).xlsx

Name
1743790668953-1742983903861-Z-Alizadeh sani dataset (2) (1) (1).xlsx

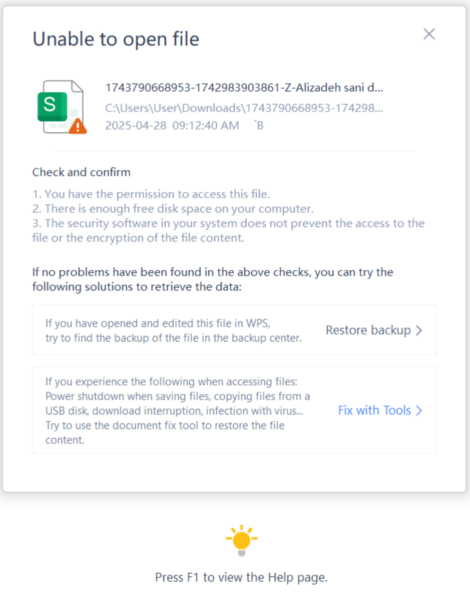
Size
96 bytes

Type
application/octet-stream

Created
5 Apr 2025, 02:17:50

Updated
5 Apr 2025, 02:17:50

File location



Unable to open file

1743790668953-1742983903861-Z-Alizadeh sani d...
C:\Users\User\Downloads\1743790668953-174298...
2025-04-28 09:12:40 AM `B

Check and confirm

1. You have the permission to access this file.
2. There is enough free disk space on your computer.
3. The security software in your system does not prevent the access to the file or the encryption of the file content.

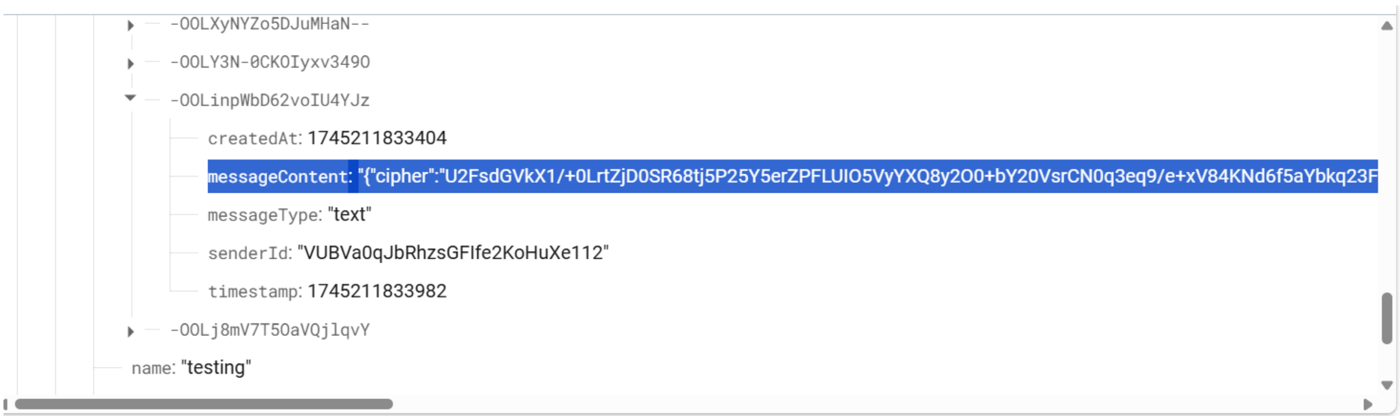
If no problems have been found in the above checks, you can try the following solutions to retrieve the data:

If you have opened and edited this file in WPS, try to find the backup of the file in the backup center. Restore backup >

If you experience the following when accessing files: Power shutdown when saving files, copying files from a USB disk, download interruption, infection with virus... Try to use the document fix tool to restore the file content. Fix with Tools >

Press F1 to view the Help page.

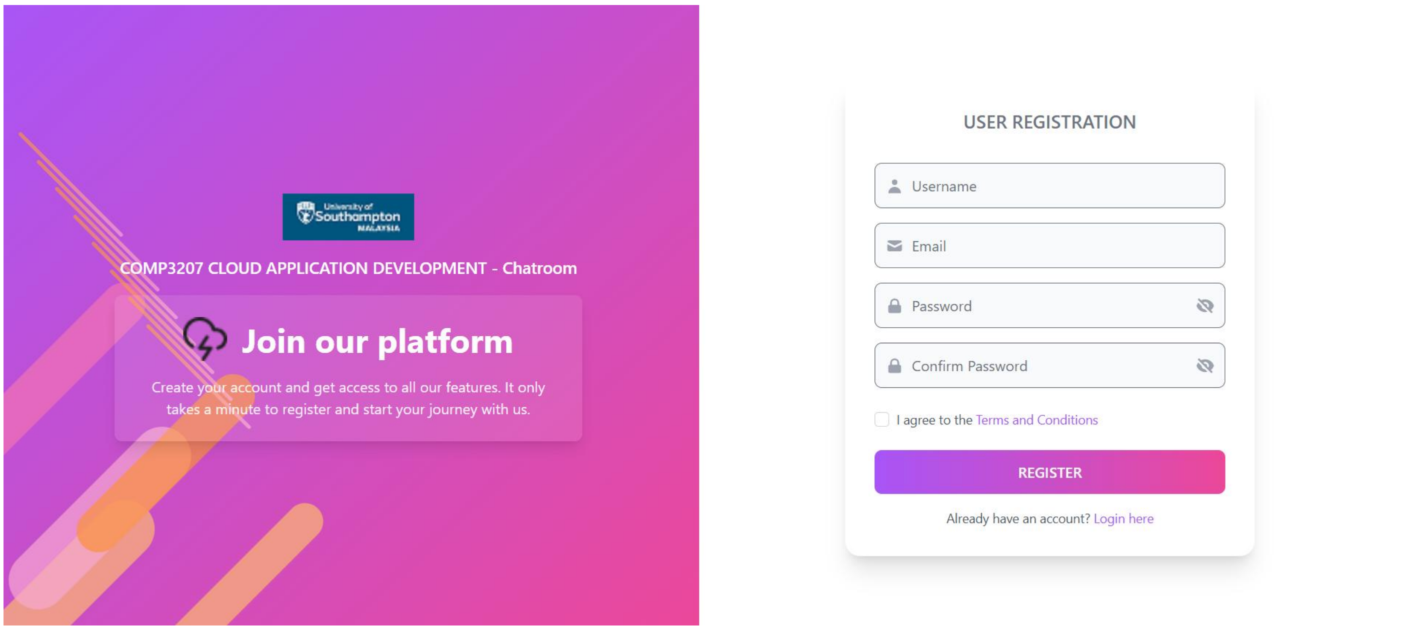
Message Encryption



```
-OOLXyNYZo5DJuMHaN--
-OOLY3N-0CKOIyxv3490
-OOLinpWbD62voIU4YJz
    createdAt: 1745211833404
    messageContent: "{"cipher":"U2FsdGVkX1/+0LrtZjD0SR68tj5P25Y5erZPFLUIO5VyYXQ8y2O0+bY20VsrCN0q3eq9/e+xV84KNd6f5aYbkq23F
    messageType: "text"
    senderId: "VUBVa0qJbRhzsGFIfe2KoHuXe112"
    timestamp: 1745211833982
-OOLj8mV7T50aVQjlqvY
name: "testing"
```

A6 Credentials

Register Page


COMP3207 CLOUD APPLICATION DEVELOPMENT - Chatroom

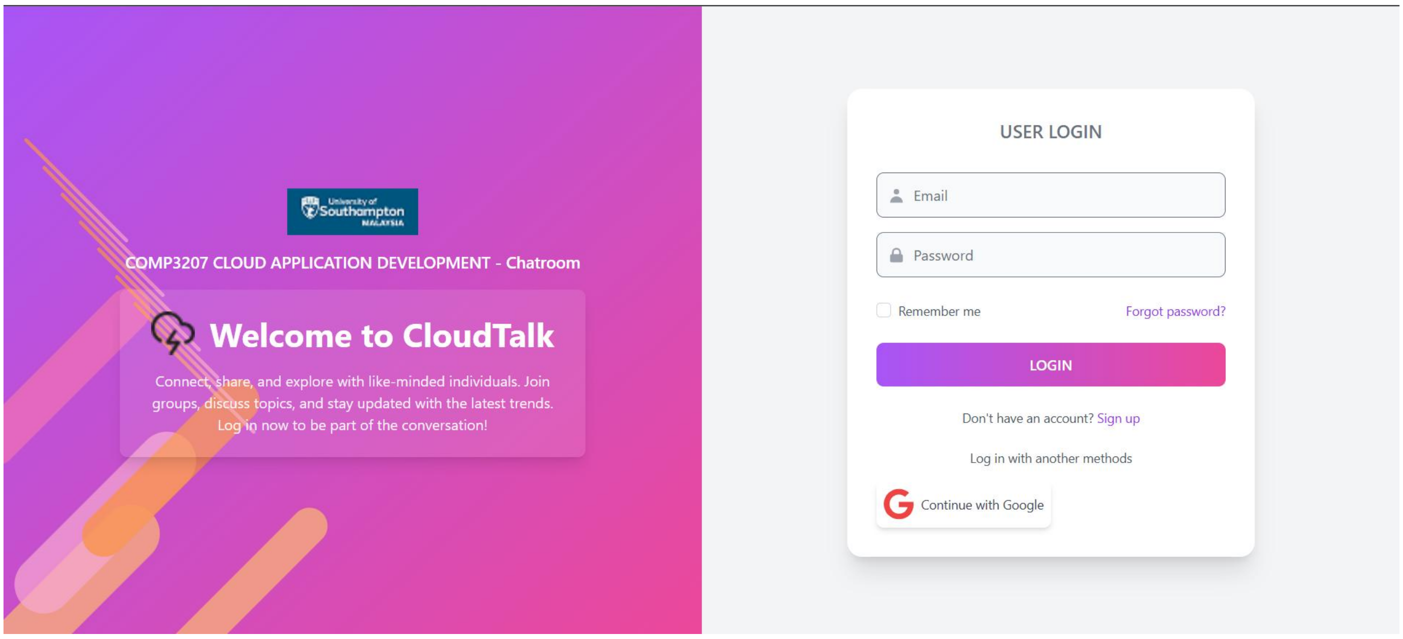
Join our platform

Create your account and get access to all our features. It only takes a minute to register and start your journey with us.

USER REGISTRATION

Username
Email
Password
Confirm Password
I agree to the Terms and Conditions
REGISTER
Already have an account? Login here

Login Page

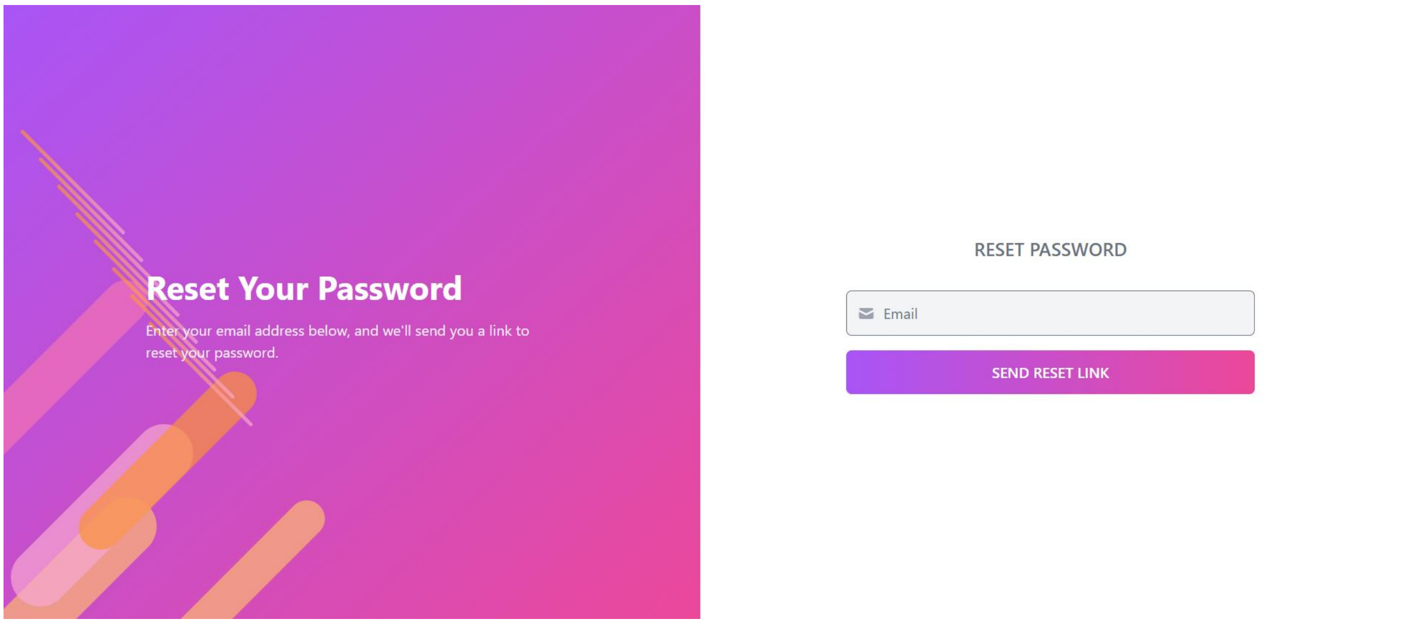

COMP3207 CLOUD APPLICATION DEVELOPMENT - Chatroom

Welcome to CloudTalk

Connect, share, and explore with like-minded individuals. Join groups, discuss topics, and stay updated with the latest trends. Log in now to be part of the conversation!

USER LOGIN

Forgot Password Page


Reset Your Password

Enter your email address below, and we'll send you a link to reset your password.

RESET PASSWORD