**CS673 Software Engineering** 

**Team 1 - Project Name**

**Software Design Document**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Wasupol | Team Leader | *WT.* | 09/23/2020 |
| Michael |  | *MR* | 10/22/2020 |
| Jian | Backup leader | *JM* | 10/10/2020 |
| Peter | QA leader | *PL* | 10/22/2020 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
| **1.0** | **Team 1** | **10/22/2020** |  |
|  |  |  |  |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_buttcq9i221r)

[Design Patterns](#_x18fj36s1121)

[Key Algorithms](#_mtfbusfb0eq3)

[Classes and Methods](#_7ucksmkf6rzx)

[References](#_15tmymhipvdv)

[Glossary](#_8n34lvocupub)

# Introduction

The Software Design Document(SDD) is a document to provide documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Software Design Document are narrative and graphical documentation of the software design for the project including software architecture, database design, design patterns, key algorithms, classes and methods, and other supporting requirement information.

The purpose of the Software Design Document is to provide a description of the design of a system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to be built. The Software Design Document provides information necessary to provide description of the details for the software and system to be built.

This Software Design Document is for a base level system which will work as a proof of concept for the use of building a system the provides a base level of functionality to show feasibility for large scale production use. This Software Design is focused on the base level system and critical parts of the system. For this particular Software Design Document, the focus is placed on generation of the documents and modification of the documents. The system will be used in conjunction with other pre-existing systems and will consist largely of a document interaction facade that abstracts document interactions and handling of the document objects.

# Software Architecture

In this section, you will describe the decomposition of your software system, which include each component (which may be in terms of package or folder) and the relationship between components. You shall have a diagram to show the whole architecture, and class diagram for each component. The interface of each component and dependency between components should also be described. If any framework is used, it shall be defined here too. Database design should also be described if used.

# 

# Database Design (if applied)

In this section, you shall describe any database if used in your software system.

# 

# Design Patterns

In this section, you shall describe any design patterns used in your software system.

* Facade Design Patterns
  + To reduce the complexity of the code by using Facade Design Patterns, this will encapsulate the hard part and delegate responsibility to other components, for example, the client does not have to know how the chat room is created. By sending a REST API call to the backend, it will generate and assign chatrooms for the client.
* Adapter Patterns
  + By default, all listview in android uses an adapter pattern for displaying content in the screen, It will adapt between the actual view to ViewHolder class.
* Observer Pattern
  + Firebase Firestore used observer pattern to achieve the real time database by the client observe some interested node (eg. chat message). When another client sent a message to the same node. Then, Firebase Firestore will trigger a change to the client so the new message is added.

# Key Algorithms

### End-to-end encryption algorithm

We care about the safety of users’ data, so we adopt the end-to-end encryption technology to protect users’ privacy.

End-to-end encrypted messaging means that the users within that specific chat can only read messages sent between two people. To enable this, the messages that are sent are encrypted before leaving a user’s device, and can only be decrypted by the intended recipient (end-user).

We use Virgil’s encryption/decryption platform to prevent anyone except the intended parties from reading messages. No one in your company, nor any cloud provider you use, can read these messages. In essence, even if a malicious person gained access to the database containing the messages, that person would only see encrypted text, called ciphertext.

To build this app, we’ll mostly rely on two libraries, Stream Chat Android and Virgil Security for Kotlin. Our outcome will encrypt text on the device before sending a message. Decryption and verification will both happen in the receiver’s device. Stream’s [Messaging API](https://getstream.io/chat/#key-features-for-chat) will only see ciphertext, ensuring our user’s data is never seen by anyone else, including us.

To accomplish this, the app performs the following steps:

1. A user authenticates with your backend.
2. The user’s app requests a Stream auth token and API key from the backend. The Android app creates a [Stream Chat Client](https://getstream.io/chat/docs/#init_and_users) for that user.
3. The user’s app requests a Virgil auth token from the backend and registers with Virgil. This generates their private and public key. The private key is stored locally, and the public key is stored in Virgil.
4. Once the user decides who they want to chat with the app creates and joins a [Stream Chat Channel](https://getstream.io/chat/docs/#initialize_channel).
5. The app asks Virgil for the receiver’s public key.
6. The user types a message and sends it to Stream. Before sending, the app passes the receiver’s public key to Virgil to encrypt the message. The message is relayed through Stream Chat to the receiver. Stream receives ciphertext, meaning they can never see the original message.
7. The receiving user decrypts the sent message using Virgil. When the message is received, the app decrypts the message using the Virgil and this is passed along to Stream’s UI components. Virgil verifies the message is authentic by using the sender’s public key.

Example #1: Setup the Backend



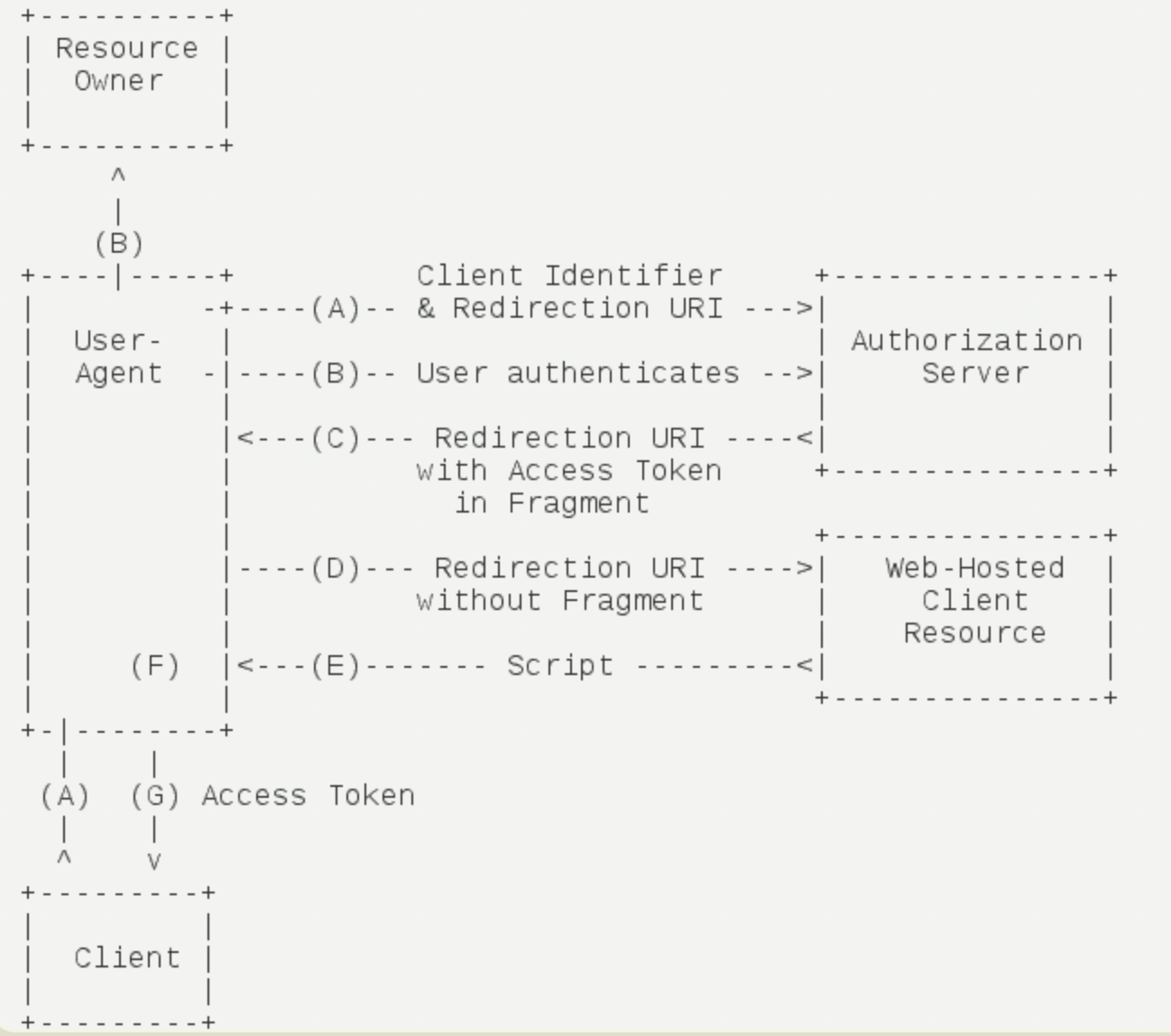
Example #2: Create a Private 1:1 Channel and Sending an Encrypted Message



### **Using OAuth 2.0 to Access Google APIs**

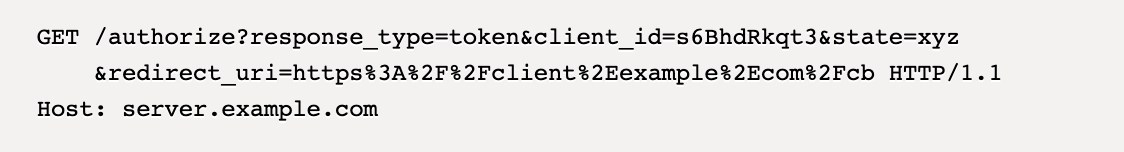
Users login our application through Google authentication and authorization, we use OAuth 2.0 to Access Google APIs.

Basically, OAuth sets up an authorization layer between the "client" and the "service provider". The "client" cannot directly log in to the "service provider", but can only log in to the authorization layer to distinguish the user from the client. The token used by the "client" to log in to the authorization layer is different from the user's password. The user can specify the scope and validity period of the authorization layer token when logging in. After the "client" logs in to the authorization layer, the "service provider" will open the user's stored information to the "client" according to the scope and validity of the token.

OAuth 2.0 defines four authorization methods: Authorization code mode (authorization code), Simplified mode (implicit), Password mode (resource owner password credentials), Client credentials. For users’ convenience, we use the second mode (implicit grant type), which does not go through the server of a third-party application, and directly applies for a token from the authentication server in the browser, skipping the "authorization code" step. All steps are completed in the browser, the token is visible to visitors, and the client does not need to be authenticated.

The process is shown in the figure below, which is taken from RFC 6749.

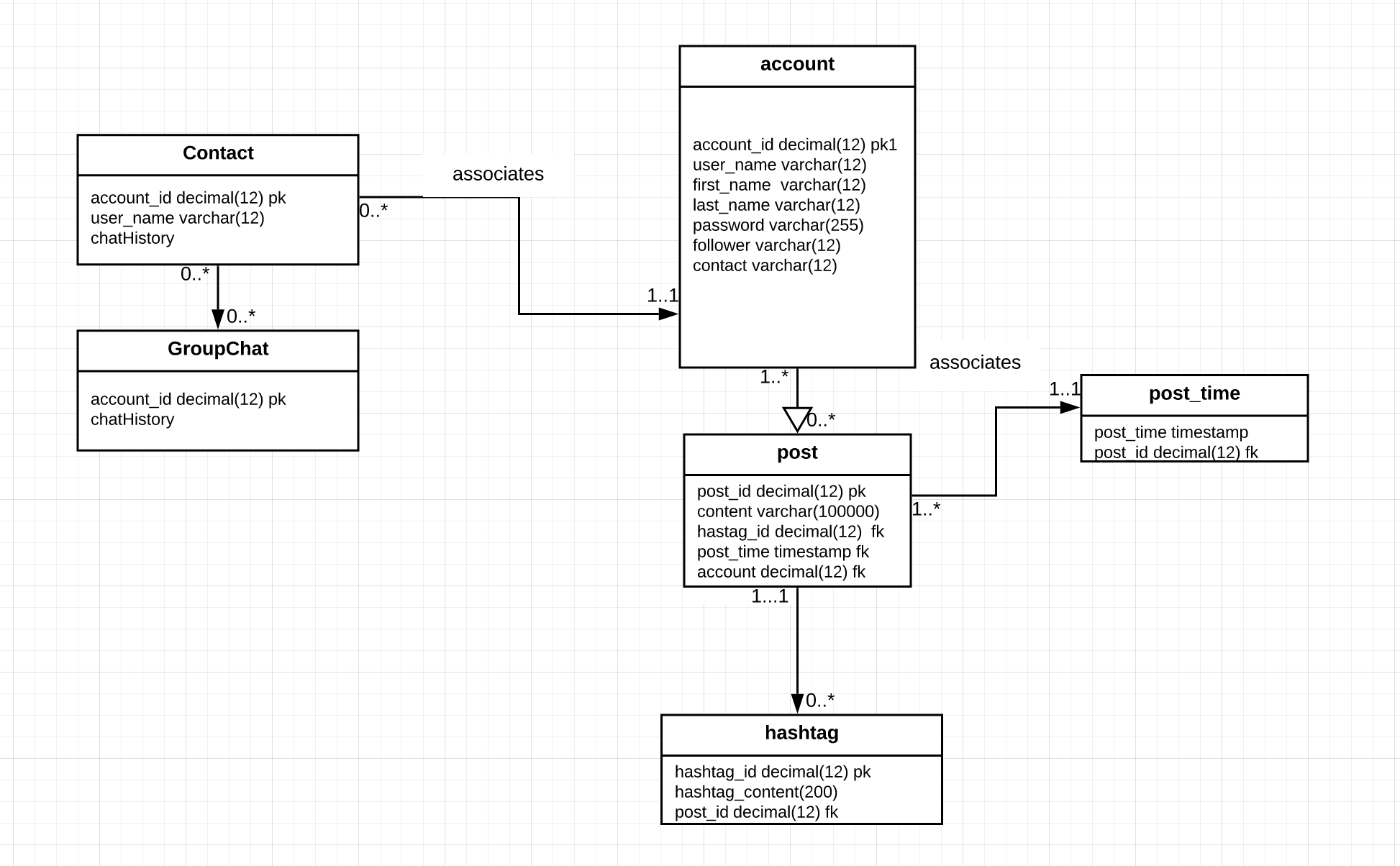
Example:



# Classes and Methods

This part can be a reference to an automatic generated document for all classes and methods.

This is a diagram depicting the object relationships:



The link below is the code structure and methods:

[Inner Circle Classes and Methods Documentation](https://inner-circle-fdd5f.web.app)

# References

* + [Dokka document generation for Kotlin](https://kotlin.github.io/dokka)

# Glossary