CS5617 Software Engineering : Specification Document

> Narvik Nandan, 111901035, Content Module Team Member

Overview

The content module is an essential part of the application which involves messaging and file sharing among clients. This module is divided into three main components : UX, Client-side content and Server-side content components.

- The UX component deals with the design of the user interface, implementation of required user actions and displaying of messages / files.
- The Client-side content component handles the business logic and processing of various events sent to / received from UX component and communicates with the server-side content component via networking module for storing data.
- Finally, the Server-side content component is responsible for storing and fetching chats and files from database, processing the received requests and sending them to the client side content component via networking module.

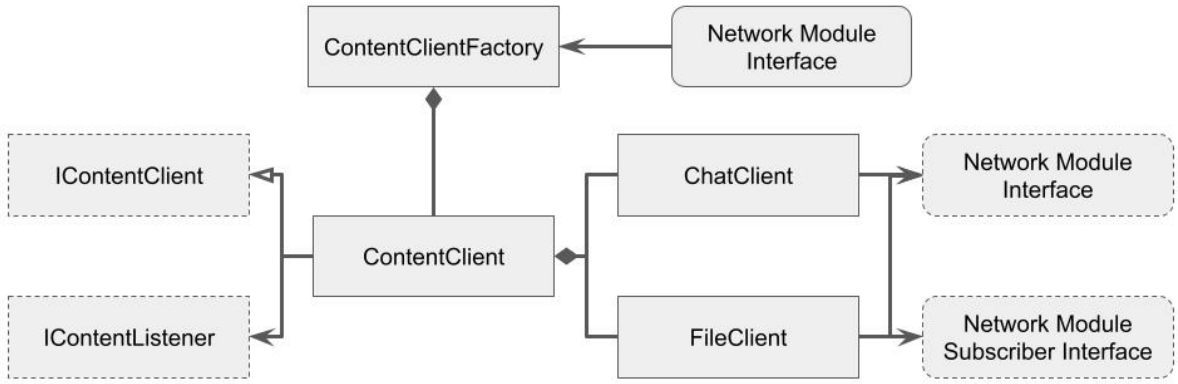
I am a team member in the content module team and will be working on the client-side content component.

Objectives

The main objectives of client side of content management are :

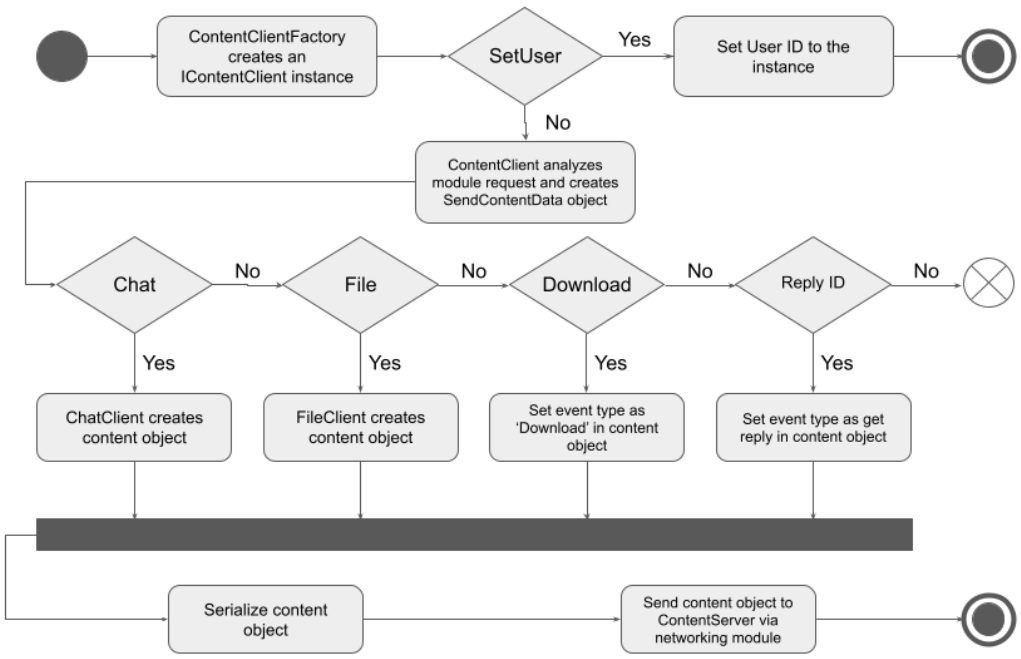
- Implementation of required interfaces and classes to communicate with other modules. Also includes publisher-subscriber design pattern between client-side content component (publisher) and clients on the UX component (subscribers)
- Receiving messages from UX component, processing them and sending them to network module.
- Handling various user actions such as editing and deleting messages, replying to specific chats, uploading and downloading files, etc.
- Subscribing to network module , processing the received messages and informing the subscribers (clients subscribed to client-side content component) about the received messages.

Class Diagram

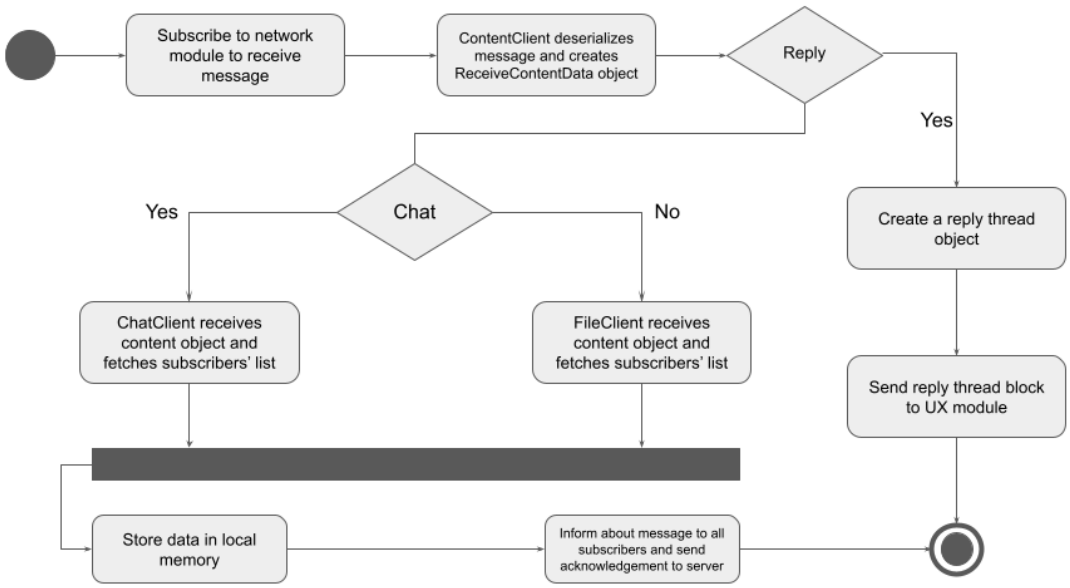


Client-side content module class diagram

Activity Diagram



Activity diagram for Client-side content component while sending messages



Activity diagram for Client-side content component while receiving messages

Design

Each user will have a client-side content manager (`ContentClient` class) which will provide interfaces to other modules and also deals with the various actions such as editing messages, reacting to messages, etc. done by the users. Since each user is

assigned with a unique ID and will have only one instance of the client-side content manager, we used a **singleton design pattern** to ensure that the class has only one instance and provide a global point of access to it.

As both chatting and file sharing are provided by the content module, the `ContentClient` class can't anticipate the class of objects (either a chat object or a file object) it must create. So, we use the **factory design pattern** to defer instantiation to subclasses and make the design more flexible. The `ContentClientFactory` class composes the `ContentClient` class which implements the `IContentClient` interface. The `ContentClient` class has two subclasses : `ChatClient` and `FileClient` that deal with the *chat message objects* and *file message objects* respectively.

The **publisher-subscriber design pattern** is used for communication between the Client-side Content component and the UX component. This pattern is also used for the Client-side content component to communicate with the Network module. The publishers provide a `subscribe()` function to all the subscribers and the listeners will implement a **listener interface** to listen to messages. For example, `IContentListener` interface is provided for the clients to listen to `ContentClient` class.

To separate the business logic of the content module from the user interface, we are implementing this module using the **Model-View-ViewModel** (MVVM) design pattern.

Interfaces

`IContentClient` interface :

```
public interface IContentClient {
  /// <summary>
  /// Sends chat or file data to clients
  /// </summary>
  /// <param name="contentData">Data to send, can be chat or file content data.</param
>
  void ClientSendData(SendContentData contentData);

  /// <summary>
  /// Edits a chat message
  /// <summary>
  /// <param name="messageID">Chat message ID to be updated</param>
  /// <param name="messageContent">Chat message content to be updated</param>
  void ClientEditChat(int messageID, string messageContent);

  /// <summary>
  /// Deletes the chat message
  /// </summary>
  /// <param name="messageID">Chat message ID to be deleted</param>
```

```
  void ClientDeleteChat(int messageID);

  /// <summary>
  /// Stars a message (will be included in the Dashboard summary)
  /// </summary>
  /// <param name="messageID">Chat message ID to be starred</param>
  void ClientStarChat(int messageID);

  /// <summary>
  /// React to a message with emojis
  /// </summary>
  /// Have to decide on the parameters and implementation
  void ClientReact();

  /// <summary>
  /// Downloads file on the client machine in the specified path
  /// </summary>
  /// <param name="messageId">File message ID to be downloaded</param>
  /// <param name="path">Path in which file will be downloaded</param>
  void ClientDownload(int messageID, string path);

  /// <summary>
  /// Subscribes to content module for listening to received messages
  /// </summary>
  /// <param name="subscriber">An instance of IContentListener interface</param>
  void ClientSubscribe(IContentListener subscriber);

  /// <summary>
  /// Gets the reply message ID
  /// </summary>
  /// <param name="replyID">Reply message ID</param>
  void ClientGetReplyID(int replyID);
}
```

`IContentListener` interface :

```
public interface IContentListener{
  /// <summary>
  /// Handles messages received by content module
  /// </summary>
  /// <param name="contentData">Received content data</param>
  void OnMessage(ReceiveContentData contentData)

  /// <summary>
  /// Handles all messages sent to client
  /// </summary>
  /// have to decide on parameters and functionality
  void OnAllMessages()
}
```

Classes defined based on the Class diagram and Interfaces :

```
/// ContentClientFactory class
public class ContentClientFactory{
  /// <summary>
  /// Sets the User ID to the instance
  /// </summary>
  /// <param name="userID">User ID of client</param>
  public static void SetUser(int userID){}

  /// <summary>
  /// Returns instance of implementation of IContentClient interface
  /// </summary>
  public static IContentClient GetInstace(){}
}

/// ContentClient class
class ContentClient : IContentClient{
  /// implements functions mentioned in IContentClient and
  /// other functions such as notify and receive message data
}

/// ChatClient class
class ChatClient{
  /// implements functions for sending, reacting, editing and starring messages
  /// handles other stuff related to chat messages
}

/// FileClient class
class FileClient{
  /// implements funtions for sending and downloading files
  /// handles other stuff related to file messages
}
```

Analysis

The singleton factory method helps in maintaining a single instance of the class throughout the use of the module. This reduces the redundancy, makes the code more robust and easy to extend. Chat and file messages are handled by different subclasses to reduce dependency. Note that the chats are stored on both client and server side. This is done so as to increase the performance of fetching the data on the client side. Since the data is not stored in a specific database, we can choose certain data structures that help in improving the efficiency of the data retrieving code. For example, hash tables and other related data structures can be used to store the message IDs. Other improvements can be made in handling emojis and downloading files.

Summary and Conclusions

At the start of the session, the client side content manager is created which will create the sever side content component. If a new user is created, a client-side content manager is created based on the User ID. All the existing messages before the user has entered the session are sent.

When the user sends a message, the content client manager will redirect the requests to chat client or file client based on the type of request. The subclasses deal with the processing of the messages, serializing and sending it to the server side of the content module.

On receiving a message from the network module, the chat client or the file client process the received message accordingly and inform the subscribers about the received message so that they can fetch it.