CS 5617 Software Engineering
IIT Palakkad

Specification Document
Content Team

111901010, Anurag Jha

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 the 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 the database, processing the received requests and sending them to the client side content component via the networking module.

I am part of the content module team and I will be working on the server-side content component.

Objectives

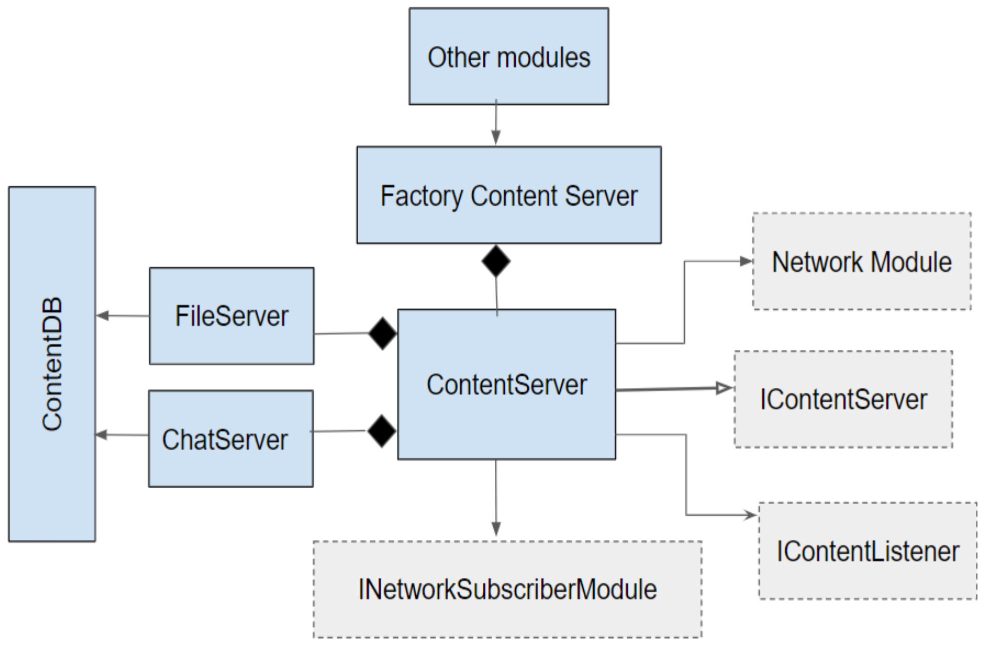
The main objectives of the server side of content management are:

- Subscribing to the network module to listen to the message objects.
- Deserializing the message objects received and separating them into 2 sets.
  - In case of a message, add it to the database.

    - In case of a file, store it in a database, and allow clients to download whenever necessary.
- Create a message object and convert into XML string, and then either broadcast it or send it privately based on the type of request.
- Handle sending data to dashboard module upon request from the dashboard.

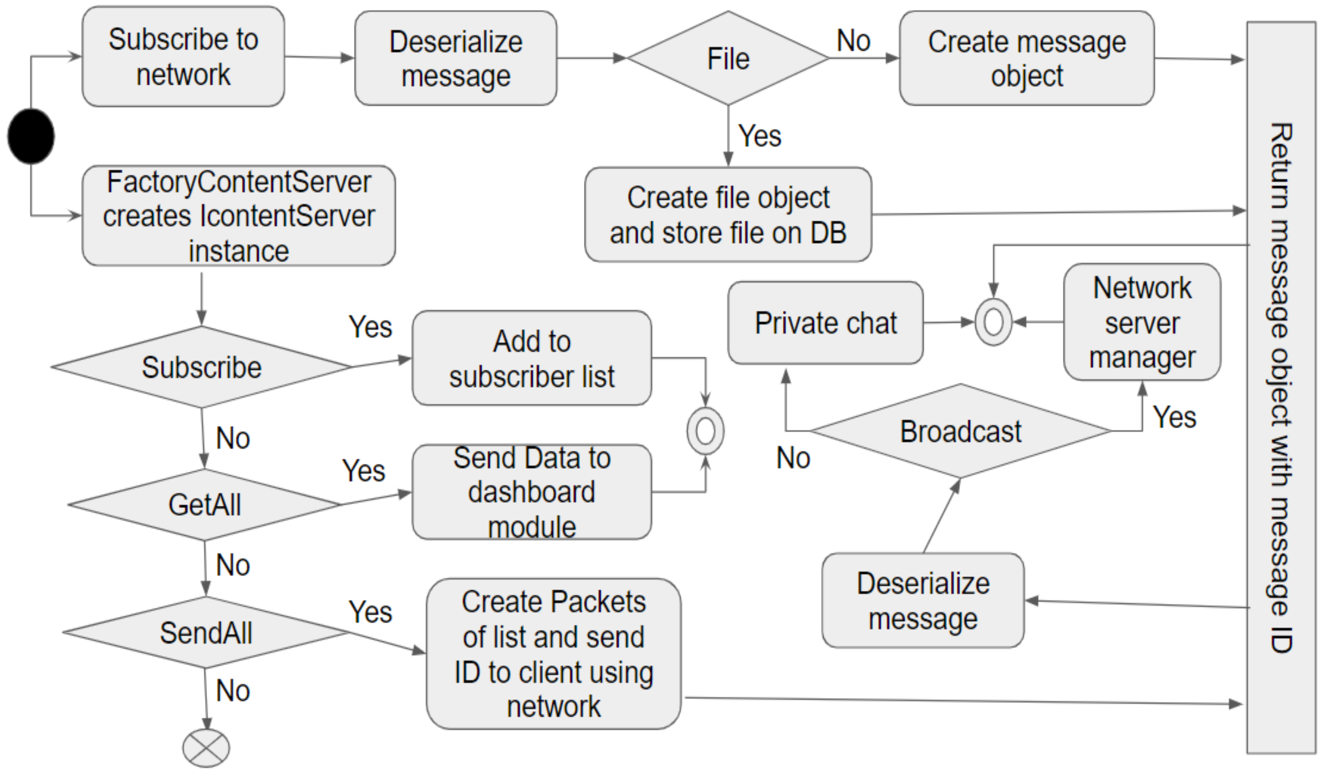
Class Diagram

Class diagram represents the static view of an application. The class diagram for content server side would look like:



Activity diagram

Activity diagram is a flowchart to represent the flow of control among the activities in a system. It shows the dynamic view of the system. The activity diagram for content server side would look like:


Subscribe to network
Deserialize message
File
No
Create message object
Yes
Create file object and store file on DB
FactoryContentServer creates IcontentServer instance
Return message object with message ID
Network server manager
Private chat
Subscribe
Yes
Add to subscriber list
No
Broadcast
Yes
No
GetAll
Yes
Send Data to dashboard module
No
Deserialize message
SendAll
Yes
Create Packets of list and send ID to client using network
No

Design

We have a FactoryContentServer that serves as a singleton factory for contentServer and returns an IContentServer implementation on the server side of the content. When creating objects using the factory design, we hide the creation logic from the client and utilize a common interface to refer to recently created objects. You can ensure that a class has only one instance while still granting global access to all users by using the singleton design pattern. The server side instance will also be initialized simultaneously with the client side interface. It offers a single point of access and simple maintenance.

The publisher-subscriber design pattern is also utilized on the server side. This offers a structure for communication between publishers and subscribers. This design involves a message broker who passes messages from the publisher to the subscribers. Subscribers can join a channel by signing up to receive communications that have been published by the host (publisher). Clients that send messages publish their contents to servers via the network, while servers subscribe to network modules to receive messages. Objects that implement the IContentListener interface will be provided to modules wanting to listen to messages received on the server.

Interfaces

An interface contains definitions for a group of related functionalities that a non-abstract class or a struct must implement. Static methods may be defined by an interface.

Server side contains the IContentServer interface. The methods called by this interface are:

```
public static void ServerSubscribe(IContentListener listener) {
/*
This function uses the publisher-subscriber design pattern. It is
mainly intended for usage by the dashboard module. When the dashboard
team requests for the starred messages, they will need to provide an
object that implements the IContentListener interface.
*/
}


public static List<ChatContent> ServerGetMessages(){
/*
This function can be used to fetch all the messages that have been
sent. It basically returns a list. The list can further be filtered
based on the time of creation.
*/
}


public static void ServerSendMessages(userID){
/*
This function sends all the messages that have been sent till now to
the new user that has just connected to the server.
*/
}
```

ContentServer class implements the IContentServer interface. It is composed of **FileServer** and **ChatServer** classes. The methods supported by this class are:

```
public static void ServerSubscribe(IContentListener listener)
public static List<ChatContent> ServerGetMessages()
public static void ServerSendMessages(userID)

void Receive(data)
/// Receive data from the notification handler and deserialize the
data.

void Send(messageData)
/// Send message to the clients

void FileSend(messageData)
/// Send the file to the client who has requested for the download.

void Notify(messageData)
/// Notify all the modules that have subscribed.
```

FileServer class will have methods that will handle:

- Storing the files on the server
- Create a message object and send it to other clients.
- Handle download requests.

ChatServer class will have methods that will handle:

- Fetching and storing the chat data to the database(**ContentDB**)
- Updating the messages.
- Starring the messages
- Reacting to the message
- Replying to message

ContentDB class has methods to handle storage, retrieval and searching of chat and files on the database.

FactoryContentServer class returns an object implementing IContentServer interface. It implements the singleton design pattern.

IContentListener interface has methods that can handle messages received from the content module and it also handles the scenario in which multiple clients send and receive messages at the same time.

Analysis

When any clients wish to retrieve data from the database, the process will be quick and accurate since the system will be consistent and the message will be stored appropriately. We can use a variety of strategies, such as sorting algorithms, to make data retrieval quicker. When a client requests a file download, the file must be stored in a way that makes it easily accessible. Since we use a client-server architecture, the entire content management system needs to be safe so that no single client has complete control over the functionality.

The application has been improved with the use of design patterns. Component reuse is an option offered by the Singleton factory. Finding resources is made simpler by the publisher-subscriber design pattern. All this will make the system more reliable.

Summary & Conclusion

The content server will subscribe to the network and will deserialize the message. For files, it will create a file object and store it in the database and make it readily available on download request. If it is a chat it, store it in the database appropriately and make a chat object. While returning the message object, it will either broadcast it or send it to a client privately depending on the request. Server also creates an instance of an interface which will add users to subscriber list, send list of starred messages upon request from dashboard module and it will also create packets of list of chat and file tables and send it to the ID that has newly joined the session on client side using network. The content module will then be integrated to the plexshare application to create an efficient lab monitor tool.