|  |
| --- |
| University of Pretoria |
| Architectural Specification – Stream2Me |
| COS 301 Team Zeon |

|  |
| --- |
| Zenadia Groenewald, Bernhard Muller, Lecton Ramasila  8-22-2014 |

Table of Contents

**Introduction**3

Purpose3

Project Scope3

**System Description3**

**Overall Architecture**3

Architectural Patterns4

Architectural Strategies6

**Details of the System**8

System Layout8

Task Handling9

**System Overview**9

Change log

|  |  |  |
| --- | --- | --- |
| Date | Version | Description |
| 01/08/2014 | v0.1 | Introduction added |
| 01/08/2014 | v0.1 | System description added |
| 02/08/2014 | v0.2 | Overall architecture added |
| 02/08/2014 | v0.2 | Details of system added |
| 02/08/2014 | v0.3 | System layout diagram |
| 02/08/2014 | v0.3 | Task handling diagram |
| 20/08/2014 | v0.4 | Performance quantified |
| 20/08/2014 | v0.4 | Performance graph added |

1 Introduction

* 1. Purpose

The purpose of this document is to give a high-lever abstraction of the possible architectural strategies and the foundations to be implemented to ensure that the requirements are met as specified in the requirements specification. This document further serves as a preliminary and intermediate specification between high-level requirements and the system’s design.

* 1. Project Scope

The scope of the Stream2Me project is defined as a software solution to enable a user to send, receive and stream media data to/from one or more users who are making use of the same application. In its completion, the project will further serve to eliminate the need for actions such as turning a monitor to show someone a video clip, or having to share earphones to listen to audio. This system can be further expanded to business and/or academic environments for collaborative work through the use of user groups in order to co-ordinate activities on projects and similar group activities.

2 System Description

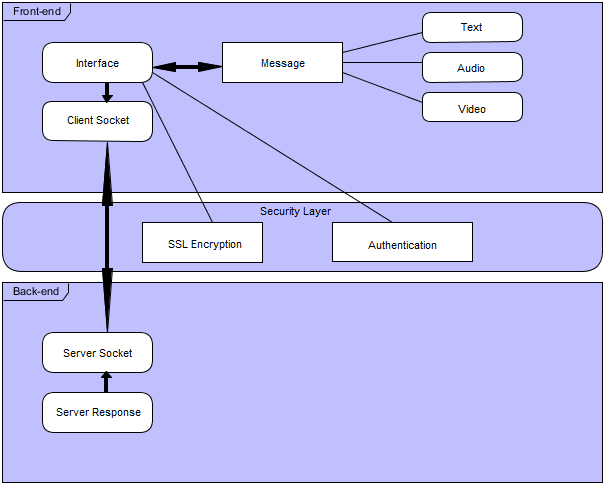
The goal of the Stream2Me project is to provide a data/media streaming service to both desktop and mobile devices to allow for ease of media access and communication within an academic environment. Its primary goal is to eliminate the necessity for the sharing of hardware between persons when attempting to exchange information, as well as to provide a convenient and efficient method of exchanging media.

3 Overall Architecture

The chosen architectural frameworks include JSE and Android, as these form part of the desktop and mobile platforms on which the system is to be developed. This system is to be deployed on Android mobile platforms as well as Windows 7 desktop platforms.

The architecture chosen is the MVC (Model-View-Controller) architecture. This specific architecture is chosen because it easily encapsulates the various actions and interactions of the user with the interface; these actions are interpreted and their interaction with the rest of the system is then determined from them and the appropriate response is given. This is all accomplished through the transfer of various different types of messages.

3.1 Architectural patterns

The MVC with a layering foundation is the primary architectural foundation that is to be used in the implementation of the Stream2Me project; but in contrast to the typical multi-layer MVC architecture, the project works on only two primary components and three predominant layers, namely the back-end (the server to be hosted locally, or preferably on the Internet, which is responsible for the distribution and sending of media and messages to and from various clients; as well as the authentication and security of messages being sent), the security layer (responsible for the encryption of transmitted data as well as ensuring the secure transmission of the data), and the front-end (the client-side interface and the primary interactions with the end-user; where all the actions available to the client are defined). This structure is applicable to both the mobile and desktop applications. The diagram below provides a simple illustration of the aforementioned architectural structure:

(Figure 1: Architectural Overview)

Front-end

This layer is essentially the adapter to the back-end and is what allows the user to interact with the system on a higher level. This contains the user work flow, ensures the appropriate actions and requests are propagated to the correct parts of the back-end, and renders the results of various actions to the user(s) and/or user groups (where applicable).

* Android client – the mobile application which runs on the Android architecture and provides the interface for the mobile application.
* Windows application – the desktop application and interface through which the user may interact with the system and clients may issue requests to the server.

Security Layer

This layer is responsible for ensuring the security and encryption of transmitted data through encryption on data packets and by enforcing authentication of user information before data may be transferred. The security of the system is imposed both implicitly by the MVC architectural pattern, and explicitly by the added security layer. The security layer implements various measures to ensure the secure access, transfer and receiving of data streams; this includes the use of the following technologies and restrictions:

* SSL encryption.
* SSL connection with TLS\_ECDHE\_RSA\_WITH\_RC4\_128\_SHA cypher suit.
* Strong user password and login authentication using Base64 SHA1 Message Digest with variable Salt.
* Session IDs and timeouts.
* Compression.
* Strong authorisation matching on any user action.
* Fixed user IDs to disallow malicious or compromising changes and adaptations.
* Disallows inter-group communication.

Back-end

This layer is responsible for the functionality and actions performed on the server side and is responsible for processing, executing and delivering the results of client requests back to the user at the front-end. These activities are all performed using the TCP protocol as a means of transferring data between the client(s) and the server when an action is performed on the front-end.

* TCP
* Client-server communications.

3.2 Architectural Strategies

Methodologies

* An extension of the Agile methodology, the Scrum methodology, is utilised to allow for on-the-fly development and adjustments to be made throughout the project’s progression. This allows for changes to be made easily and for additional features to be added, as many of the individual project units are not interdependent and are largely isolated.
* Allows a project’s direction to be adjusted or reoriented based on the completed work, and not speculation or predictions, made beforehand.

Auditability

* User details are retained with each instance of access in the system.
* Message histories are retained per user, per group and include every communication action performed by users, when they are performed and the recipient’s details.
* The use of session IDs and tracking of user activity allows for easier auditing and allows the hosts to monitor activity and enforce security within the application.

Constraints

* Open source and available on two different architectures/platforms.
* Audio and video streaming, due to the size of the streamed objects, might cause bottlenecks.
* Currently creates a message arrival delay of up to 40 seconds.
* Must be able to accommodate 1000 (at minimum) users at any given time, on the same server and users must be able to communicate and stream data with minimal delay.

Integrability

* All services are to be implemented first separately and integrated into the system once unit tests are complete.

Maintainability

* Services and system modules will be designed to work both independently and when they are integrated into the system – they will be decoupled in their development.
* Coding standards are to be used in order to make the code flexible and understandable.
* User manuals, help functions and troubleshooting manuals will be included.

Performance

* Threading and priority scheduling to assist in performance enhancement in the events of network traffic or delays.
* Messages, without media streaming, have an average transfer time of about 1 – 2 seconds regardless of the message type.
* Messages, with streaming active, have an average transfer time of between 20 and 70 seconds depending on the size of the streamed object (Video resolution)
* Messages sizes for video averages at about 235.688 KB; audio averages at about 8.04251 KB; and others average between 0.24683 KB and 8.08496 KB.

(Figure 2: Evaluation of the average, maximal and minimal delays expected of the various message types.)

Reliability

* System must allow for multiple stable client connections simultaneously.
* Reliability to be tested using unit testing.
* Connections are persistent and remain connected while the user is active.

Scalability

* The framework on both the client and server ends provide resource management (e.g. threading).
* The system must allow for multiple concurrent users (about 400 users maximum).

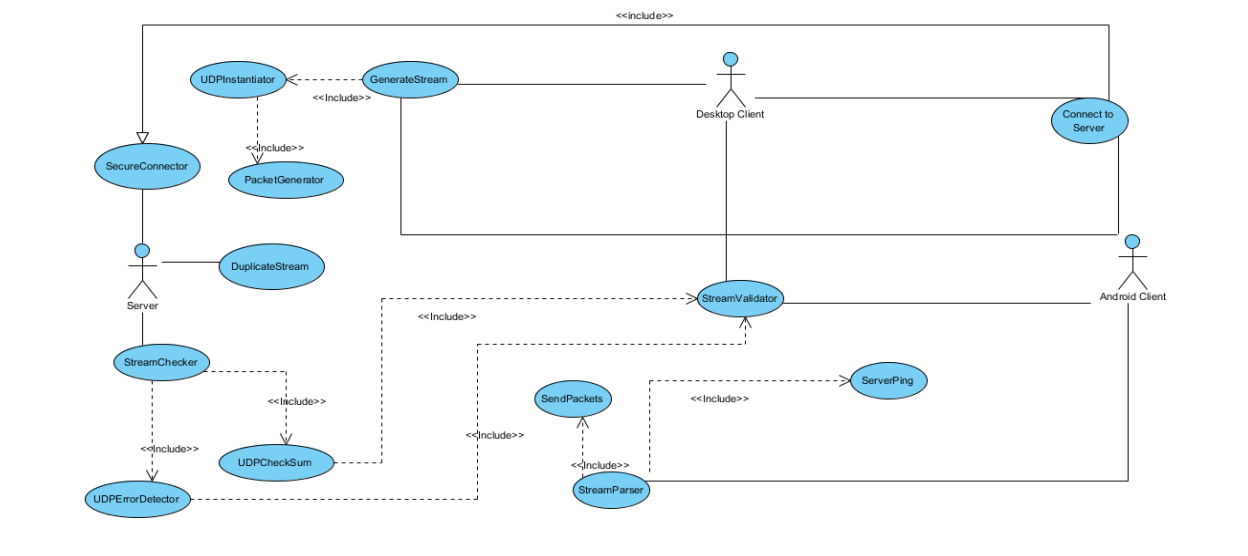
Security

* Secure and maintainable data transfer protocol (TCP).
* SSL encryption.
* System ensures that data is not interceptable during transmission.
* Local authentication of users (possibly provided by a database).

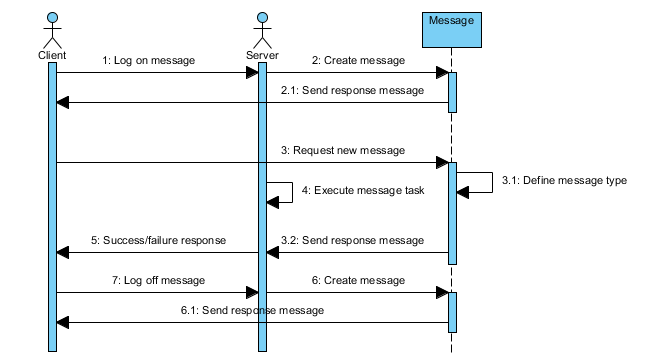
Usability

* The interfaces of both the mobile and desktop applications should be intuitive to use.
* Must be easy to use through a simplified and minimalistic interface.

4 Details of system

4.1 System layout

(Figure 3: An activity diagram illustrating the processes and layout of the system.)

4.2 Task handling

(Figure 4: A sequence diagram illustrating the process of task handling in the system.)

5 System Overview

5.1 Client-side (Netty)

The client works on an asynchronous, event-driven networking framework; this means that when a message is sent and comes in, it is first decompressed and decrypted, after which it is delegated to one of the many message-handling threads. From there, the extrapolated data is sent to the Controller (as illustrated in the MVC pattern), which then updates the Model.

5.2 Server-side (Netty)

When a message comes in, it is handled and its data is sent directly to the database for storage. Depending on the type of message, it may be modified and sent further on – this modification can be anything from a change in message type to group data addition.

5.3 Server-side Security

Database passwords are encrypted using a variable salt key. Data streams are compressed with a v3 certificate as well as a TLS\_ECDHE\_RSA\_WITH\_RC4\_128\_SHA cipher suit using an SSL connection layer. In addition, the certificates are verified and contained in a bouncy castle key store imbedded into the application. Furthermore, all groups are physically split to disallow cross-communication. All database updates are strictly validated and verified for authorised access.

5.4 Client-side Security

All data are checked against their specific user ID; streaming actions are server-allow requests. Server authentication is confirmed upon connection, after which login details are encrypted and encoded to base64 without sending.