**Software Design**

**Document**

**1. Introduction**

The end product of this project is a desktop application, which enables users to join into online work groups and communicate. This application provides a simple and fast way of sharing or exchange information, news, or ideas. The usage of this product is simple and interfafce is familiar for most potential users. Each work group channel provides privacy for the participants, each room have to be created with a password.

**1.1 Purpose**

The purpose of this document is to present in depth description of the designs of the WorkGroupApp. It will also include overview of the design process involved and technologies used, development proces and guidelines for the end user. In conclusion, this document could be used as a starting point for future upgrades or improvements and a usage information source for a user.

**2. System requirements**

WorkGroup application is a software for personal computers and laptops and can be run on any machine provided it has Java virtual machine installed. There is no minimum requirements for the hardware as it is a light-weight application.

**Requirements:**

OS with JVM

Internet Connection

**3.Use cases**

**3.1 GUI client-side application**

**1. Service**

WorkGroup application client software is designed for end user, and establishes connection to server. The product will provide real time communication service among it’s clients. Each user can create or join a group, which is hosted on server, and interact with others using a text based chat function. Groups are password protected and it ensures only authorized connections are accepted.

**2. User interaction**

Clients run the software on their system and are presented with GUI for interaction. User interface is kept simple and adheres to the familiar standart of other applications, which should provide better experience with minimal adaptation.

**2.1 Authorization**

WorkGroup application requires clients to be registered and authenticated before gaining access to main features. Main menu part - “User” - displays log in and new user creation options, both of these present an apropriate pop up window with a required fields to fill. After either option is completed succesfully client is presnted with a main view of an application.

**2.2 Main view**

Main view consists of a list of up to date groups hosted at the server. Group list view provides name of a group and a “Join” button. If joining a group, application displays a dialog box requiring group room password, provided entry is correct user is taken to main work group chat window.

**2.3 Work group view**

Work group view contains text area where messages are displayed, up to date user list and message entry field for sending messages.

**3. Design considerations**

**3.1 Constraints**

The software is only usable when internet connection is available. The main server has to be running for establishing connections and creating work group channels from client applications.

**3.2 Design methodology**

The system is designed with flexibility and maintainablity as its main focus. Software code is divided into manageable modules each responsible for concrete functionality.

WorkGroup application consists of two main parts: client-side UI application and a program to act as an end point for connections of client-side.

**3.3 Functionality design**

**1.System service**

The client-side software will be single-user application providing connectivity to server, which will serve as a host for multiple client connections. Server handles all the users, and processes them. Each client is monitored and event-based bidirectional exchange is established, according to information recieved server provides services for clients. Server will be responsible for client requests, like: authorization, group creation and allocation, grouping client connections.

**2. Data persistence**

Server side provides some level of data persistence by the use of remote online database. Client application users are required to be registered and authorized by the server to gain access to core features. User credentials are recorded on registration and converted to objects at server code. These objects are processed again and translated to JSON format and stored at DB. On log in request from client, server must validate request data against DB records, and, depending on result, handle the connection.

**4. Technology used**

**4.1 Client application**

**1. Java**

WorkGroup client-side implementation is written in Java programming language. The main reasons of using this programming language is its popularity and portability over wide range of systems. Java language has a massivfe amount of libraries and tools available. On top of that JVM enables computer to run Java programs independent of hardware as it is compiled to Java bytecode.

**2. JavaFX**

This poroject is using JavaFX library for GUI development. Being written in native Java code it is also portable to systems running JVM. That is a main rationale of using this library for GUI development. It is fairly new approach to user interface creating and contains lots of components for this purpose.

**3. Socket.IO Client API**

For bidirectional communication with server, system is using Socket.IO client library for Java. It enables real-time event-based means of information exchanging between the client and server. Connections can be established even in the presence of firewall, due to the way it is creating connection. It contains lots of readily available features, which are vital for WorkGroup application functionality. Socket.IO allows to create individual namespaces, which act as a spearate communication pools but using the same connection. Possibility of broadcasting to all connected users or to specific groups. Message sending and recieving is based on events, which in turn can be used to implement further features at each end of connection.

**4.2 Server side**

**1. Javascript**

Javascript is a programming language used to develop server for this software. It is an interpreted language and does’nt require a lot of resources to run. For a server with a possibility of heavy loads it is a big advantage.

**2. Node.js**

Open-source and cross-platform JavaScript run-time environment which executes JS code outside of a browser. It is especially well designed to run server-side and provides many features for that purpose. It has event-based architecture and capable of asynchronous I/O, this optimizes scalability of server application. Node is designed for network applications, capable of handling many connections concurrently, furthermore there is no concers of dead-locking since it has no locks. These features are essential for WorkGroup application functionality and are well suited for purpose.

**3. Socket.IO Server API**

As in client-side, this project uses Socket.IO server-side API. As both, client and server API’s are based on the same concept, handling connections becomes efficient and less complicated. Node.js integrates Socket.IO and real-time event-based communications between server and client are implemented on a well developed solution of sockets.

**4. Mongoose**

An Object Data Modeling library for MongoDB and Node.js. Used to translate objects in code to documents in MongoDB. MongoDB is a NoSQL document database and it speeds up the process of designing DB and reduces the complexity. WorkGroup application uses Mongoose library to store user credentials for validation purposes. Converted objectes are stored as documents of JSON format, which is used widely and easy to work with.

**5. System architecture**

**5.1 General overview**

WorkGroup software system at a high level utilizes a *client-server* architecture. A multiple remote client machines, using computer network, request and recieve data/information/events from a centralized host system.

This particular system server establishes event-based communication with client machines. Client and server software is designed to be communicating using a pre-difiend contract of message structure.

Implementation uses Observer Design pattern to inform about state changes of Subject class.

**5.2 Server architecture**

**1. Overview**

Server is written in Javascript and uses Node.js framework, Socket.IO Server API and Mongoose library. These enables to provide required service for client computes. Server-side is composed of several scritps:

* ServerMain.js – main entry point module for the program
* ConnHandler.js – module handling communication on sockets, core functions
* Client.js – module representing remote client
* WorkGroup.js – module representing work group
* UserDbModel.js – a module, containing Mongoose schema which maps to MongoDB object

**2. Modules**

**2.1 ServerMain**

This module is main entry point for server program. It loads required modules and initializes server to listen for incoming connections on port 31337. Socket.IO server object is set to listen for ‘*connection’* event. When event is recieved from client socket, it signals connection esatblishment. Client socket is setnt to callback function of server object, then server object reference and socket are passed to ConnHandler.

**2.2 ConnHandler module**

Module containing all the bussines logic of a communication handling between client and server.

It takes two parameters: server reference and client socket. Here we set event listeners on client socket. Event listeners are monitoring client sent events and executes appropriate functions.

Client sent events are handled and response is sent back, this way communication between two sockets continues until disconnect.

**2.3 UserDbModel**

A module holding a connection to user credentials database at MongoDB and schema which maps object to document.

**5.3 Client architecture**

**1. Overview**

Client application implements a connection handler over Socket.IO API to communicate with server. At start client must establishes connection to server and binds socket to it. After communication can begin between client and server, various events are sent to server using GUI interaction. Response events from server are handled by client and dealt with.

**2. Classes**

**2.1 ConnectionSocket.java**

Class responsible for initializing connection to server and dealing with server responses.

Requests and events are sent from client app using ConnectionOutput class methods, server catches the event and sends response or performs some other actions.

Server responses are dealt with using this class instance. Using private instance methods and Socket class methods, each event response is registered with a Listener which listens for server responses and deals with them. ConnectionSocket class implements Subject interface, which Observer objects can observe and be notified about any state changes.

**2.2 ConnectionOutpu.java**

Class responsible for output to socket bound to server. Various user events are sent to server. Custom named event is sent to server with some JSON data if required. These events are then captured at server side and in turn server sends response or performs some actions at server side.

**2.3 GUI package**

Contains all GUI classes.

**UI.java -** Main JavaFX launcher class. Loads main .fxml file and holds a primary stage of application UI on which all other components are built.

**MainViewController.java -** Main UI controller of an application. Using this controller instance applications is providing UI functionality

**EventManager.java -** Class responsible for dealing with events from server and updating UI according to event.

**3. Interfaces**

**3.1 Observer –** functional observer interface with single *update()* method. Classes implementing this interface get update() method called when server response event is registered.

**3.2 Subject –** interface declaring observable subject methods. Class implementing this interface has a list of observers and if its state changes, observers method update() is called.