META-ARCHITECTURE DOCUMENT

Version 1.1

Project Title: **CHAT APPLICATION**

Team Members: Christopher Cargile
Jonathon Ng
Gayathri Parthasarathy
Nicklaus Rhodes

VER. 2014

VERSION HISTORY

Version#	Revision Date
1.0	02/19/2014
1.1	04/19/2014
1.2	04/25/2014

Table of Contents Page No

META-ARCHITECTURE DOCUMENT

TEAM CUPRIC SECKEL

ARCHITECTURAL VISION

This document describes the architectural layout for a Peer-to-Peer chat system. We envision our system to be a highly distributed browser-based chat client with a centralized server providing registration services. The chat itself will be Peer-to-Peer, allowing for direct connections between users in the system.

The system architectural design will emphasize the idea of providing our clients with direct connections for their communication requirements. The emphasis is on ease of use for those users. To meet the goal of high usability in the end product, we emphasize well thought-out and developed product architecture. The interfaces developed should be intuitive for both the end user as well as the product developer. The underlying structure of the product should be able to easily extend to add new features, refine existing ones or eliminate undesirable ones. The system will provide log in authentication to aid in confidentiality and integrity because the users who log in will have the right to view their own chat and to send messages. Availability will be upheld because the user's machine will be its own host and client for messages between other clients.

PRINCIPLES

Principle Name	**K.I.S.S. - Keep It Simple, Silly-head**
Description	Architectural decisions should make implementation as simple and direct as possible. Simplicity is defined in this case as "less convoluted", less dependencies between components and generally favoring a decoupled design.
Rationale/Benefits	We have limited experience and compressed timeframe. By sticking to a more easily understood design, we expect to each gain a fuller comprehension of the entire system; understanding the system fully will allow us each as developers to make beneficial design decisions during implementation.
Implications	Designs that are simpler would be favored because it will allow team members to understand the code in the project as well as allow users to interact with the code with little or no training. Care must be taken to make sure the simpler design is still viable.

Counterargument	Could do something cool and unique with custom code, cooler toolkit, or some other new-fangled widget.

Principle Name	**"Scale" The Mountain**
Description	Even though our project prototype may only need to support 2 – 4 clients for a demo, realistically, we need to account for much larger participation in the architectural design wherever possible. Even though the implementation would not be able to support large numbers (due to our lack or resources) of users and dynamic server loading, the high level design should be scalable.
Rationale/Benefits	If you are going to do something, you should try to do it right. Who knows? We might accidentally develop a great app and become wildly rich.
Implications	This consideration runs counter to our **K.I.S.S.** principle previously stated. Each design aspect will need to be designed to allow for change which may make the system more complex with function capabilities that will not be used until a later point of time that may or may not happen at all.
Counterargument	Would be a lot easier to not do this.

Principle Name	**A.G.I.L.E. – Another Good Idea Loses Effect**
Description	The goal is to allow for improvements to existing functionality and new functionality to be added to the system easily. Agile development puts an emphasis on delivering functionality incrementally and often.
Rationale/Benefits	Will allow new changes to be implemented quickly. Will allow the team to be very responsive regarding workload. Sounds very ninja-like.
Implications	Engineering design process would need to identify fundamental requirements for a minimum-functionality prototype, then rank additional features for inclusion.
Counterargument	The development cycle is very short and there will not be time for multiple cycles that usually define an "agile" approach. A project plan that exemplifies the waterfall model is more appropriate since we have very clear periods of time of Design, implementation and testing defined in our class syllabus. We also do not wish to go backwards in time to add components due to our time constraints.

STYLES

The Chat application will be a distributed system with three modules – client, server and database. The Architectural styles for implementing Chat Application are selected based on the initial implementation functionalities intended for the application.

Application will be implemented with the following functionalities –User Profile update (Registration Information), Notification, User Search, Chat room.

Styles selected for implementing the distributed chat application are

1. Client-Server
2. Object Oriented
3. Peer to Peer

Client – Server Style:

The client server architectural style will be used to implement the distributed chat client and the centralized server. The web based client access the server through protocol and initiates a socket connection, through which it communicates with the server. The server keeps track of all chat client requests and provides response and services requested.

Rationale: Client –server style helps implement distributed chat application in layers, which enables to have multiple clients. Clients can independently access the server. Centralized server handles all the requests from the clients and provides centralized data storage.

There may be single point failure, but in future enhancements it will be taken care of by considering measures like server replication during peak load.

Object Oriented Style:

The service requests by the chat clients to the server are implemented using object oriented architectural style. The client requests service through remote procedure calls. The functionalities that are to be implemented using this style are

1. Registration – client registers with valid credentials
2. Server Notification – Whenever a chat client signs in, they will be notified of all chat clients online.
3. User search – Chat client can search for a user to initiate chat.

Rationale: To provide abstraction and data integrity in the system. Provides reusability through abstraction and suits for the dynamic behavior of the chat application. It makes the application more understandable.

Peer to Peer Style

We will be implementing the communication between the clients in peer to peer architectural style. The peers communicate using protocol. The textual chat conversations are delivered between the peer through the sockets established between the peers. Once if they want to leave the chat, the connection will be disabled.

Rationale: To provide mutual independence between chat clients and to facilitate the distributed system.

Alternate styles

We would like to consider following alternate styles for the chat application.

1. Black Board Style

System will have a centralized database which holds all the information and there won't be server mediating the connection. Instead the client just accesses the database, which holds the logic to connect and user details. Then the connection between clients will be established in peer to peer style.

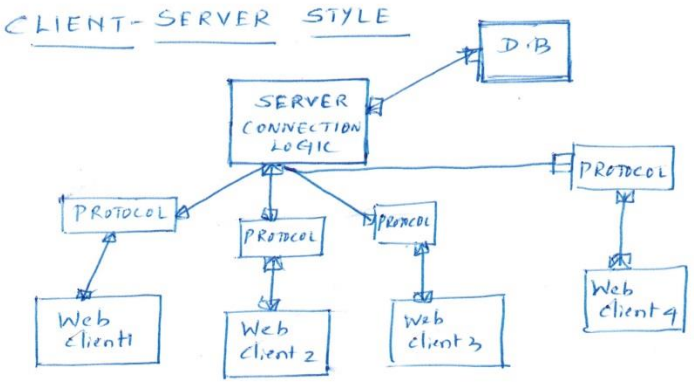
Rationale for not considering in the system: Still the system will have single point failure and we did not want to burden the database with all the information to handle the system. So we prefer have a server that mediates the chat application.

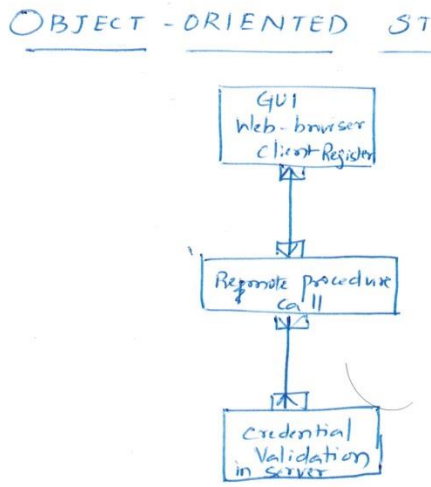
2. Pipe and Filter

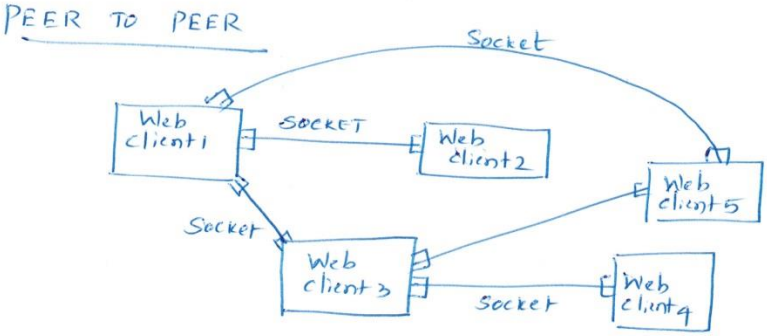
Chat User component and connection component are used instead of centralized server. Connection component establishes the connection between the chat users. Once the connection is established, further message exchange between the users takes place in peer to peer style.

Rationale for not considering in the system: System will become more complex by increasing the number of components and will require installation of those components in the user system. Currently we are focusing on implementing a browser based GUI.

Figures depicting architectural styles selected:







PATTERNS & INTERCONNECTION MECHANISMS

PATTERN [THREE TIERED] (SEE FIGURE ON PAGE 10)

SUMMARY DESCRIPTION –

Provides a mechanism for end-users to connect to a server through a set of intermediary components comprising the GUI layer and a connector which pair in order to engage a remote server that affords a means for connectivity, routes and processes chat dialogue messaging requests and directs handshaking requests providing means for security.

CONTEXT OF USE (INTENT)

Consistency is made of clients using a web-browser which interact with a server that processes requests into and from a database

PROBLEM STATEMENT

A three-tier pattern of implementation separates the Chat GUI resources from the connection and processing server and remote database resources to ensure chat requests are directed towards clearly-defined tiers responsible for various functionalities within the system.

SOLUTION DESCRIPTION

Tiers for a chat system require the chat user to interact with a frontal GUI tier. The front tier engages a data connectivity-purposed middle tier which accepts peer registration and/or system login requests. Chat messaging requests are also forwarded through the middle tier which can retain data for processing, filtering and forwarding of it to be stored in a distributed database system. The database system is accessed at the point new users engage in becoming registered or previously registered users submit a login request so a Registry Server component can allow or disallow access dependent on acceptable access credentials.

VARIANTS AND RELATED PATTERNS

Layered systems are used to process requests or invocation separately but do not do so horizontally so much as three-tiered systems

KNOWN USES -

Allows for GUI components to store and retrieve data held in remote locations and to interact with a server possibly located in another separate location.

CONSEQUENCES

The architect needs to specify interaction mechanisms and tiers need adhere to the request-reply approach for responding to state change

Class: ServerConnector	Collaborator: RegistryServer
Responsibility: Accept chat-client login request call; delegate associated details to Server	

PATTERN [MVC]

SUMMARY DESCRIPTION –

Three well-articulated component roles comprise the roles in implementation using the pattern. A model component holds relational data for the system with a clear purpose of staying accessible by the controller and display components to ensure data is shown to all affected users as necessary.

CONTEXT OF USE (INTENT)

Data from the user input is represented in a view that remains representative of data by a control mechanism required

PROBLEM STATEMENT

Users logged in to a chat room should access and display the same dialogue from the data associated with a chat room

SOLUTION DESCRIPTION

There is a loose coupling between actions of the view and controller components, which tie into a model component, so that data for different chat rooms can change independent of each other and users display appropriate dialogue information.

VARIANTS AND RELATED PATTERNS

Presentation-activation-controller (PAC) pattern is inspired by MVC

KNOWN USES –

Data resources correspond to model objects; visual representation of data through a browser or other interface is a view member; controller code responds to user input and causes the interface to change display data.

CONSEQUENCES

When a user changes their data, his view and other registered views are updated

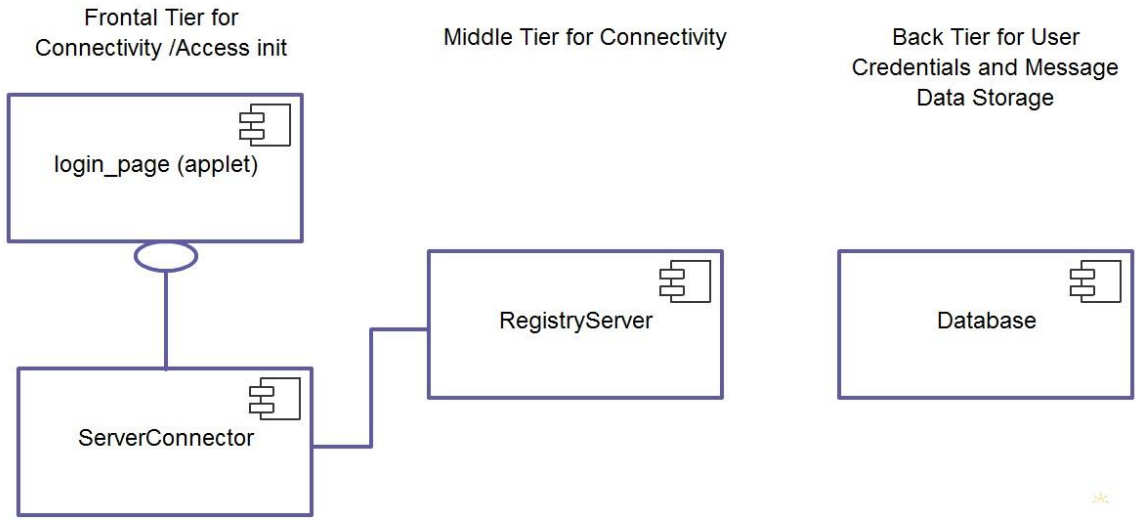
RATIONALE

Separating components into three categories ensures a loose coupling to help in visualization of a system that achieves a larger goal by way of functions separated according to process roles.

Class: P2PConnector	Collaborator: Data model, Display component(s), RegistryServer

Responsibility: acknowledge outgoing registry/messaging requests and allows control to ensure client views are updated per related data model	

FIGURE SHOWING 3-TIERED MODEL OF CHAT PROGRAM



INTERCONNECTION MECHANISM [REST]

The interconnection of chat client users is enabled by using the web as a collection of resources. A web server that responds to client requests to share dialogue data with the remote system is achieved with usage of the transfer mechanism, which is HTTP that connects endpoints using sockets to bind inbound requests to an application that processes HTTP Get/Post requests and in turn forwards necessary dialogues as data in response.

CONNECTORS

LINKAGE CONNECTOR-TYPE

The transport of data and passing of control between the server component and the data source component is achieved through an intermediary known as the data access connector. The thread of control passes from the server to the connector, as the program counter is incremented, and the connector is invoked by passing in the appropriate connector library inside the server component of the program. In this way the data-access connector provides linkage to the database component which the server can inspect and call upon to effect modifications to the underlying messages data that is presented.

DATA ACCESS CONNECTOR-TYPE

The data access connector is a principle between the server module and a database that is configured to hold messages from the messaging peers by submitting data transported by HTTP. The peer-to-peer approach allows multiple users to interact with each other simultaneously, however there is a centralized source that must determine priority for reads and writes to the database so synchronization of messages displayed is maintained. The type of connector used should provide an API so its usage is broadcast consistently to all components throughout the system, whether a peer or another facilitator within the overall system. The data access connector is the more appropriate type of connector for the design considerations relevant in our peer-to-peer chat application.

PHILOSOPHIES & PREFERENCES

1. Each member of the team will be assigned responsibilities associated with the designing, implementation and testing of the system.
2. Every day members of the team should communicate over email to rest of the team about the updates on the project.
3. Team should meet every Saturday to discuss about the entire week updates and schedule for next coming week.
4. A checklist should be created about the scope of work intended for each week schedule.
5. Minutes of Saturday meeting should be sent to everyone in the Team to keep track of the agenda.

GUIDELINES & POLICIES

1. Implementation of the application will be done in stages.
2. Open source software tools and database should be used for the project.
3. Any modification or change in the elements of architecture encountered during the entire phase of the project should be documented in the corresponding documents.
4. Version control should be followed whenever a document is updated.

ADDITIONAL INFORMATION

The read only working copy of the project will be available to everyone in google codes and can be accessed using the following link.

```
http://csci-656-spring14-team-project.googlecode.com/svn/trunk/csci-656-spring14-team-project-read-only
```