META-ARCHITECTURE DOCUMENT

Project Title: **CHAT APPLICATION**

Team Members: Christopher Cargile
Gayathri Parthasarathy
Nicklaus Rhodes

VER. 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 the robust functionality associated this type of system.

The system architectural design will emphasize the idea of providing the best service for our clients. The emphasis is on ease of use and a pleasant experience 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.

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 over better, but more complicated, designs. 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	**Be Able to "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 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 that impact scalability will need to be heavily scrutinized and weighted to satisfy both principles.
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.

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.

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

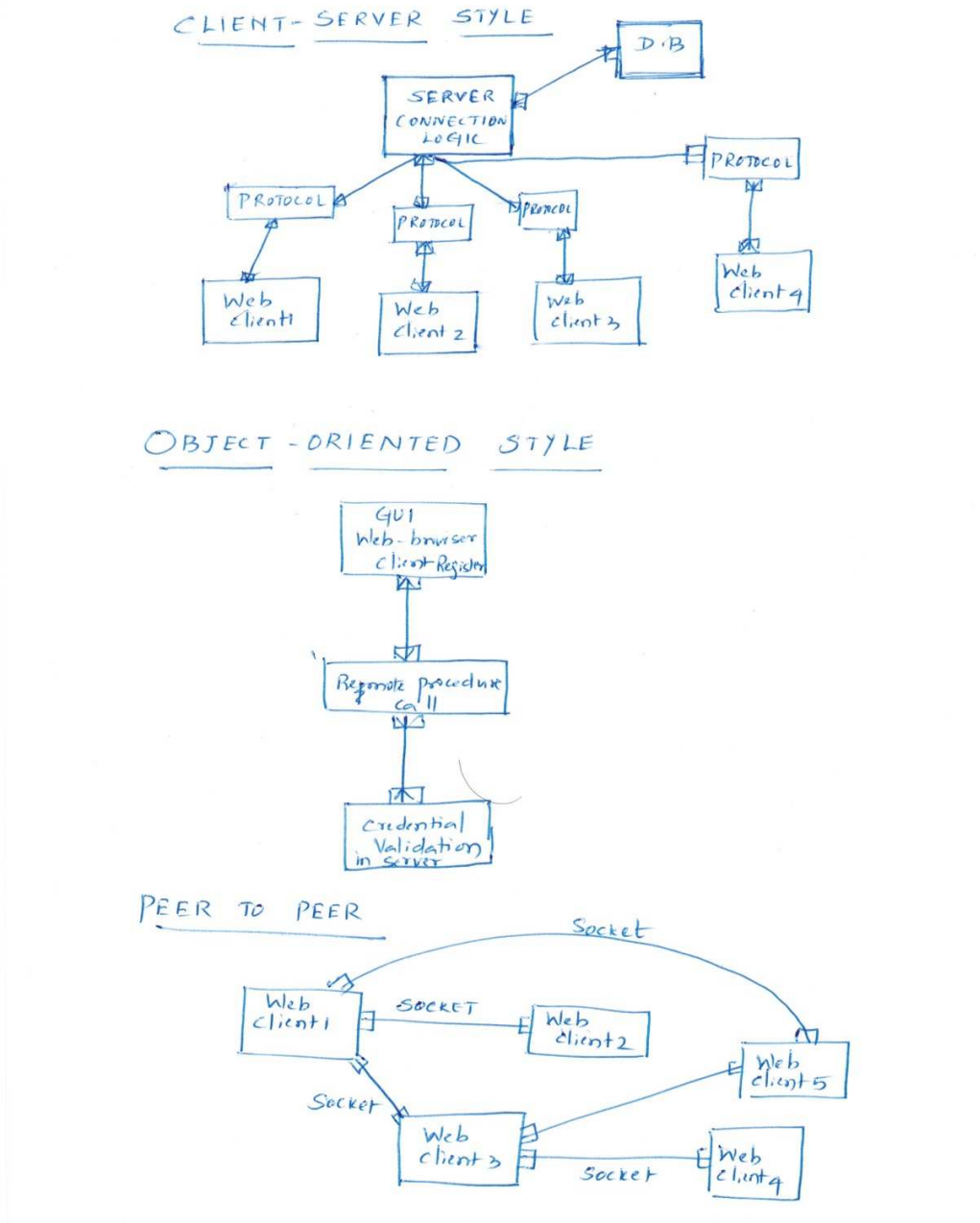
1. Registration – client registers with valid credentials
2. Server Notification – Whenever a chat client signs in, they will be notified of all chat clients online.
3. User search – Chat client can search for a user to initiate chat.

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.



PATTERNS & INTERCONNECTION MECHANISMS

PATTERN [THREE TIERED]

SUMMARY DESCRIPTION –

Provides a mechanism to connect end-user to the server and process chat dialogues

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 allows multiple clients to make chat requests at tiers responsible for the various parts of functionality in the system.

SOLUTION DESCRIPTION

Tiers for a chat system require the chat user to interact with a front tier. The front tier in the system transfers data to the middle tier which processes requests. The chat request is then forwarded for processing to a back tier which can retain the data for processing.

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 -

Distributed users process, store, and retrieve data for science

CONSEQUENCES

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

RATIONALE:

Class: ServletRequest	Collaborator: HTTP post, data connector tier
Responsibility: Listen to client request call; delegate information handling	

PATTERN [MVC]

SUMMARY DESCRIPTION –

Three members comprise the pattern implementation. A model holds data for the system as a member which is accessed by the controller and display to ensure related users display data for their chat session consistently.

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 –

Web resources correspond to model objects; HTML displayed in a browser is a view member; controller code responds to user input and causes the browser 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: HttpServlet	Collaborator: Data model, end-user displays
Responsibility: acknowledge incoming requests and control responses to update client views per the related data model	

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
```