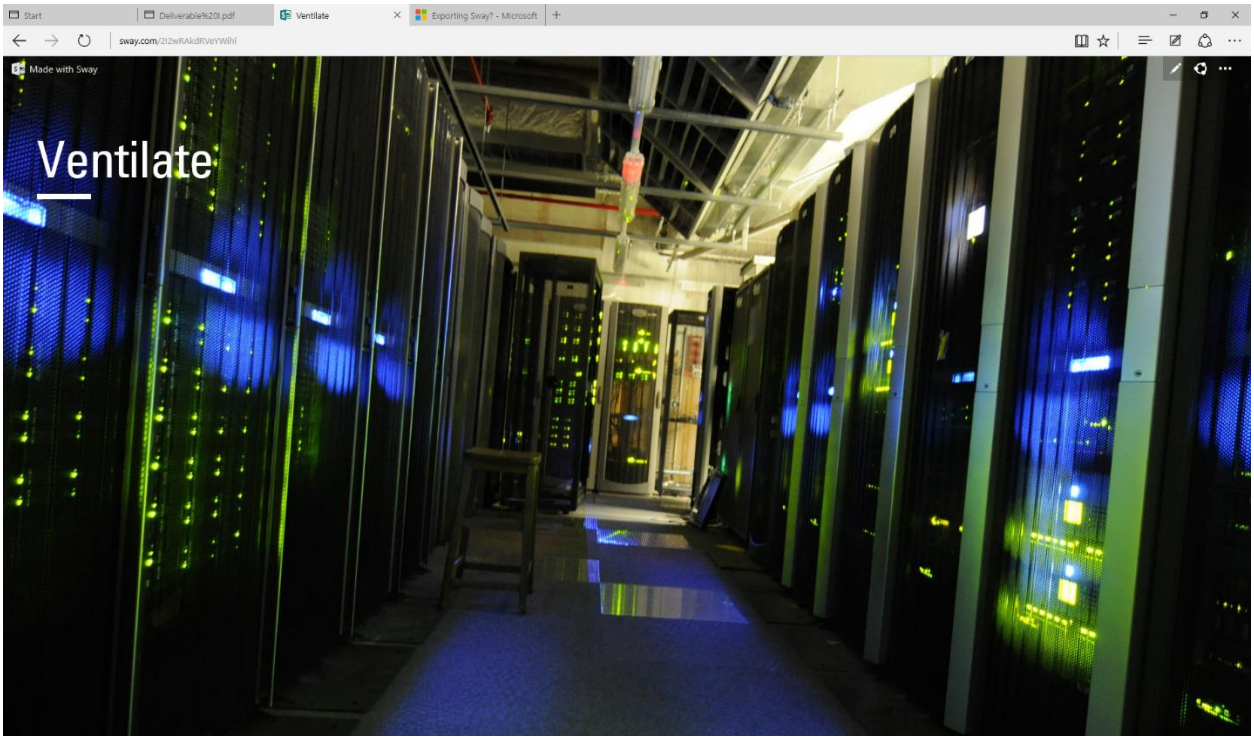
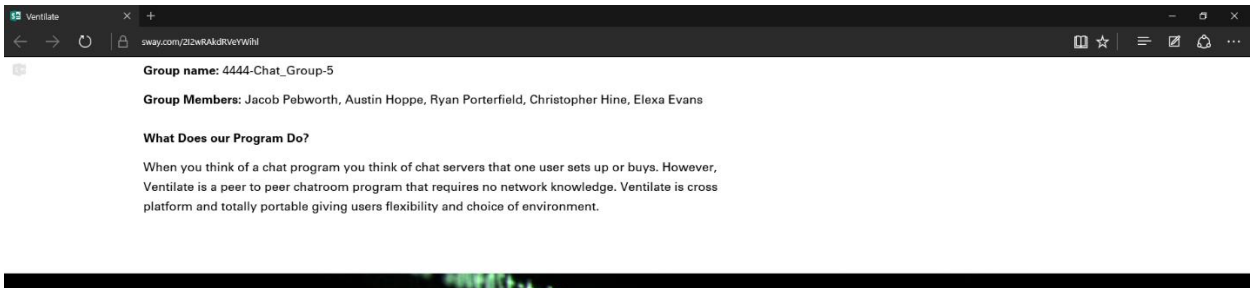
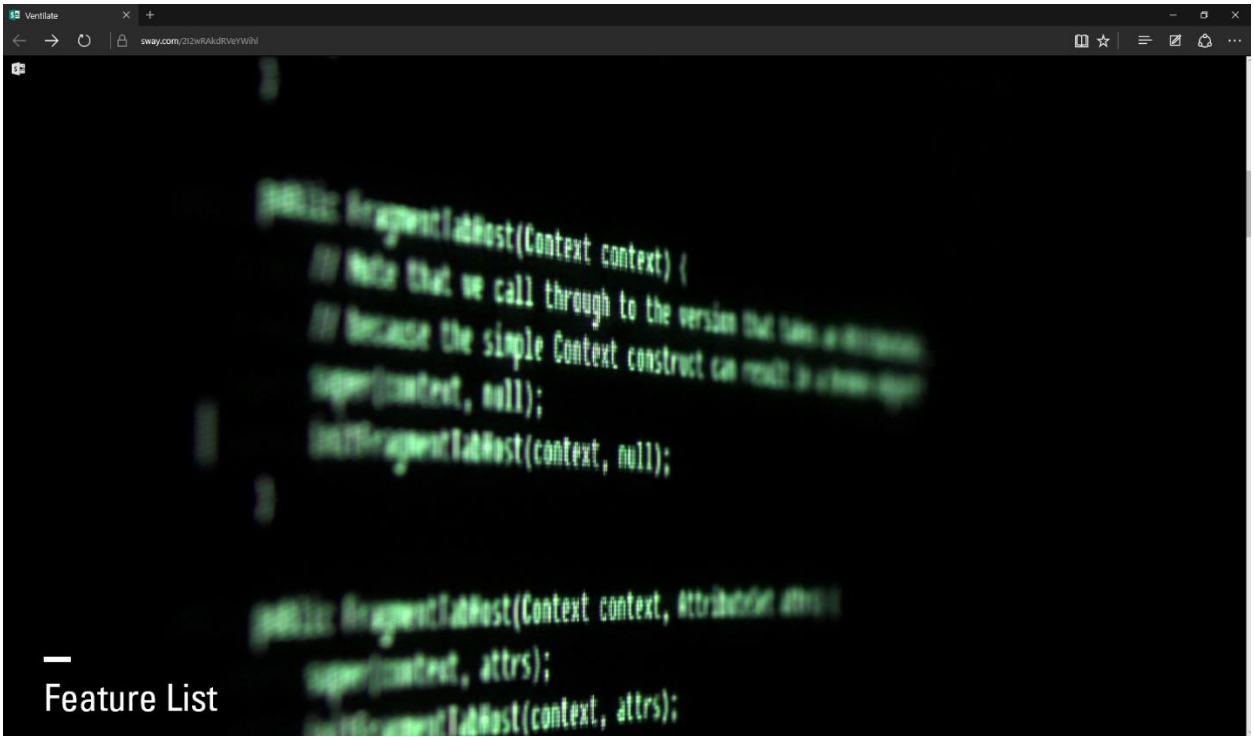
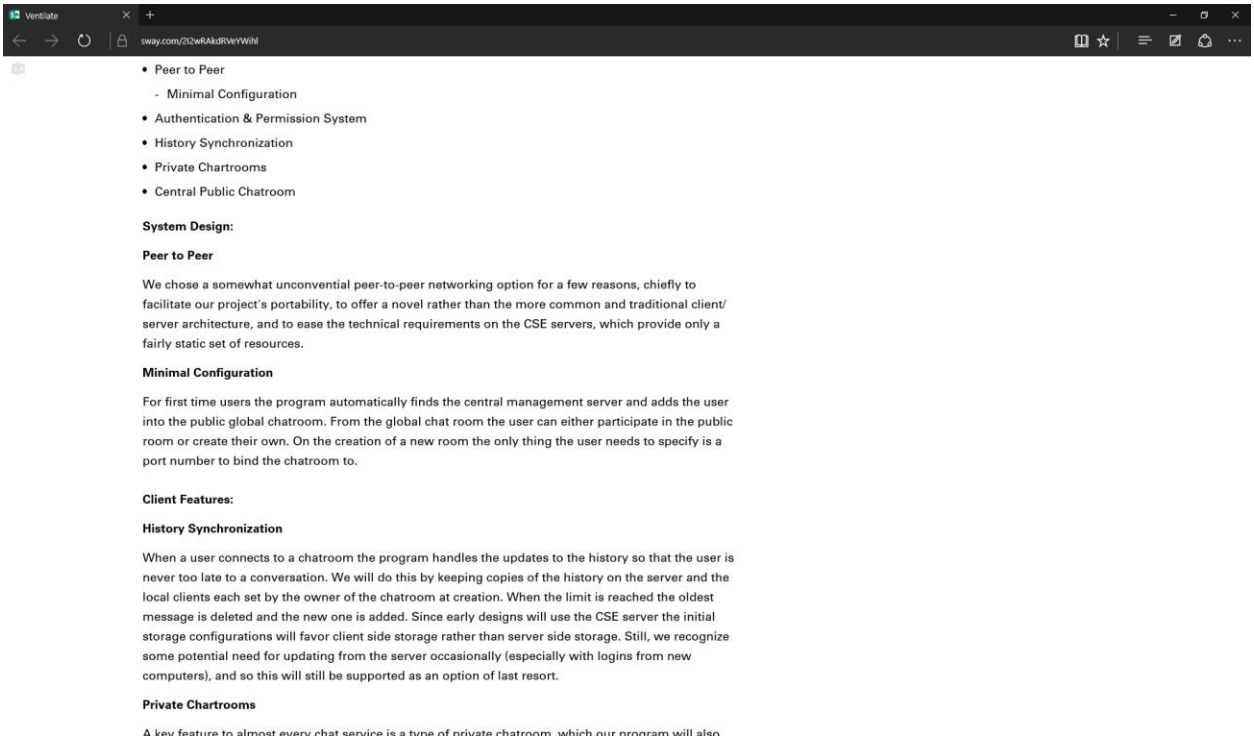
https://sway.com/2I2wRAkdRVeYWihl









- Peer to Peer
  - Minimal Configuration
- Authentication & Permission System
- History Synchronization
- Private Chartrooms
- Central Public Chatroom

System Design:

Peer to Peer

We chose a somewhat unconventional peer-to-peer networking option for a few reasons, chiefly to facilitate our project's portability, to offer a novel rather than the more common and traditional client/server architecture, and to ease the technical requirements on the CSE servers, which provide only a fairly static set of resources.

Minimal Configuration

For first time users the program automatically finds the central management server and adds the user into the public global chatroom. From the global chat room the user can either participate in the public room or create their own. On the creation of a new room the only thing the user needs to specify is a port number to bind the chatroom to.

Client Features:

History Synchronization

When a user connects to a chatroom the program handles the updates to the history so that the user is never too late to a conversation. We will do this by keeping copies of the history on the server and the local clients each set by the owner of the chatroom at creation. When the limit is reached the oldest message is deleted and the new one is added. Since early designs will use the CSE server the initial storage configurations will favor client side storage rather than server side storage. Still, we recognize some potential need for updating from the server occasionally (especially with logins from new computers), and so this will still be supported as an option of last resort.

Private Chartrooms

A key feature to almost every chat service is a type of private chatroom, which our program will also

- Peer to Peer
  - Minimal Configuration
- Authentication & Permission System
- History Synchronization
- Private Chartrooms
- Central Public Chatroom

System Design:

Peer to Peer

We chose a somewhat unconvential peer-to-peer networking option for a few reasons, chiefly to facilitate our project's portability, to offer a novel rather than the more common and traditional client/server architecture, and to ease the technical requirements on the CSE servers, which provide only a fairly static set of resources.

Minimal Configuration

For first time users the program automatically finds the central management server and adds the user into the public global chatroom. From the global chat room the user can either participate in the public room or create their own. On the creation of a new room the only thing the user needs to specify is a port number to bind the chatroom to.

Client Features:

History Synchronization

When a user connects to a chatroom the program handles the updates to the history so that the user is never too late to a conversation. We will do this by keeping copies of the history on the server and the local clients each set by the owner of the chatroom at creation. When the limit is reached the oldest message is deleted and the new one is added. Since early designs will use the CSE server the initial storage configurations will favor client side storage rather than server side storage. Still, we recognize some potential need for updating from the server occasionally (especially with logins from new computers), and so this will still be supported as an option of last resort.

Private Chartrooms

A key feature to almost every chat service is a type of private chatroom, which our program will also

Fundamentals

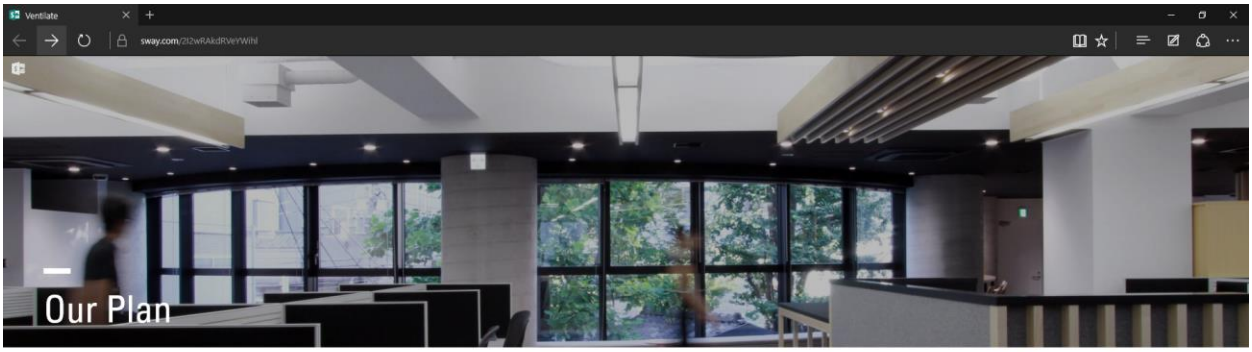
Our program has a few key design aspects which inform our initial choices, among these security and usability.

Security

We will be encrypting sensitive data stored on the server, specifically private chat, passwords and chat room history. This allows for users to have the security afforded by storing their data locally (as suggested by a peer-to-peer approach to networking), while at the same time providing the convenience of being able to access the computer from new machines.

Usability

This ability to log in to the client from any given computer was very important to our design. It reinforces our cross-platform development, and couples well with the design of our server. The GUI for our client-side application will feature easy access to the various abilities available to peers (still in design).



Development

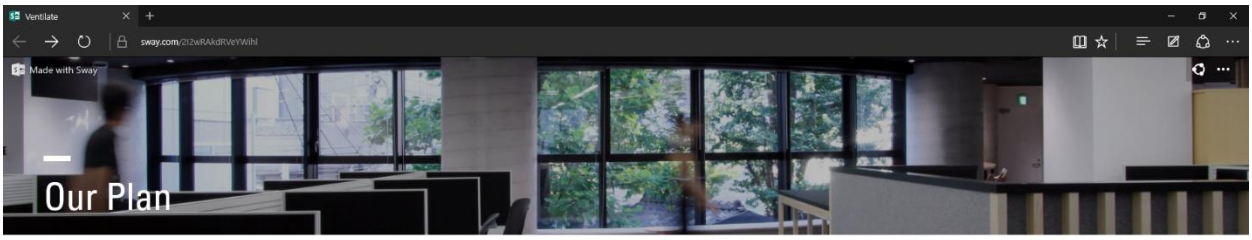
- Language: C/C++, potentially scripting/markup languages as needed
- GUI Library: Qt
- Environments: Windows, Mac, and Linux (client), Linux (server)

Why C++?

C++ was chosen because of the deep cross compatibility of the language, and the rich development environments available for a variety of different platforms. We needed a program that could be run on the Computer Science and Engineering (CSE) server provided by the University of North Texas (UNT) for some administrative tasks, like managing the central chat "room" and providing IP address translation across clients.

Why Qt?

Qt is a cross-platform graphics toolkit, targeting C++ development. We will use this library to develop the GUI for our program, in addition to some data structures and functions which we may be able to make use of.



Development

- Language: C/C++, potentially scripting/markup languages as needed
- GUI Library: Qt
- Environments: Windows, Mac, and Linux (client), Linux (server)

Why C++?

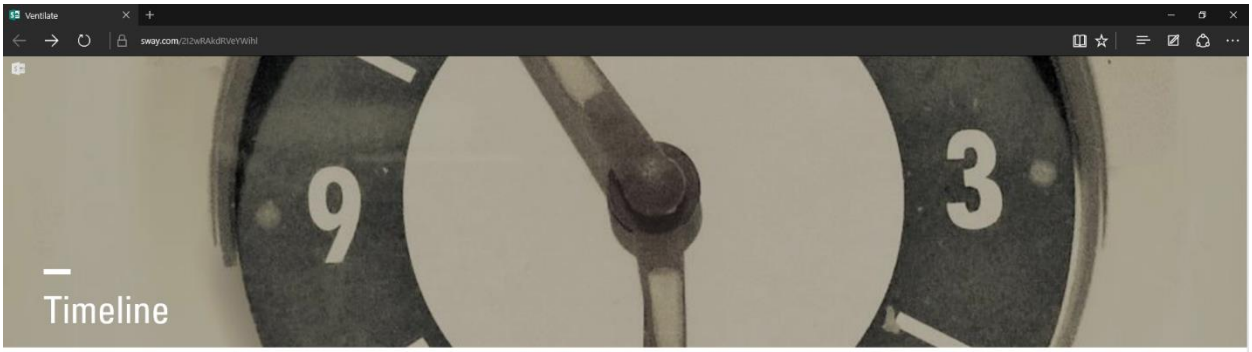
C++ was chosen because of the deep cross compatibility of the language, and the rich development environments available for a variety of different platforms. We needed a program that could be run on the Computer Science and Engineering (CSE) server provided by the University of North Texas (UNT) for some administrative tasks, like managing the central chat "room" and providing IP address translation across clients.

Why Qt?

Qt is a cross-platform graphics toolkit, targeting C++ development. We will use this library to develop the GUI for our program, in addition to some data structures and functions which we may be able to make use of.

Environments

The development group will be actively coding on Windows, Mac OS X and Linux platforms, so a main focus for the team with be compatibility with all three environments.The server portion of our program will be running under a Linux-based virtual machine, on a CSE department server, to which access is provided by UNT.



Client Milestones

Connect to clients - 2 days

Create user accounts - 2 days

Update user logs - 1 day

Authenticate/log in users - 1 day

Reset user passwords - 1 day

Server Milestones

Connect to server - 1 day

Send account creation request to server - 1 day

Send login request to server - 1 day

Send password reset request to server - 1 day

Connect to peers - 3 days

Connect to clients - 2 days

Create user accounts - 2 days

Update user logs - 1 day

Authenticate/log in users - 1 day

Reset user passwords - 1 day

Server Milestones

Connect to server - 1 day

Send account creation request to server - 1 day

Send login request to server - 1 day

Send password reset request to server - 1 day

Connect to peers - 3 days

Distribute chat history to peers - 3 days

Check authenticity of chat history against multiple peers - 1 day

Distribute list of chat rooms among peers - 1 day

Check viability of chat room list (timestamp, data integrity) - 1 day

Update new messages across peers - 1 day

Confirm updates - 1 day

Qt GUI - 7 days

Optional Milestones

More efficient data structure for maintaining connection density between peers - 5 days

Public/private chat room switching - 1 day

Generation of cryptographic keys for user information (passwords, private chats) - 5 days

Generation of cryptographic keys for chat rooms (as necessary) - 2 days

Ventilate
sway.com/2t2wRAkdRVeYWihI

Potential Problems

High-Level Project Scale

Our five-person group presented a particular challenge in designing a suitably complex system. Diverse expertise (across different fields) and evolving design goals have led to a somewhat ambitious set of goals. To this end it is very important that our group works well together, and that we do not lose any members. Our solution to managing this risk is to develop our core features first, completing aspects like the underlying server architecture and client-side networking before additional features.

Server Policies

A very real issue which we will potentially face is incompatibility with the server policies on the CSE machines provided by UNT, particularly with respect to storage space and opening connections with clients. Our choice of peer-to-peer networking was part of our approach to mitigating the impact of this problem. The server is required to store only a minimum of information in order to ensure consistent performance of our program across different client machines, with realtime updates occurring only across peers' connections to each other. Further, apart from maintaining a set of connected users, clients handle all interactive steps themselves (including opening chat rooms and requesting chat logs).

Scaling Problems

High-Level Project Scale

Our five-person group presented a particular challenge in designing a suitably complex system. Diverse expertise (across different fields) and evolving design goals have led to a somewhat ambitious set of goals. To this end it is very important that our group works well together, and that we do not lose any members. Our solution to managing this risk is to develop our core features first, completing aspects like the underlying server architecture and client-side networking before additional features.

Server Policies

A very real issue which we will potentially face is incompatibility with the server policies on the CSE machines provided by UNT, particularly with respect to storage space and opening connections with clients. Our choice of peer-to-peer networking was part of our approach to mitigating the impact of this problem. The server is required to store only a minimum of information in order to ensure consistent performance of our program across different client machines, with realtime updates occurring only across peers' connections to each other. Further, apart from maintaining a set of connected users, clients handle all interactive steps themselves (including opening chat rooms and requesting chat logs).

Scaling Problems

As designed, our program uses a complete graph to model the total number of bidirectional edges connecting peers attached to the system, which has a Big-O performance of O(n^2) (where n is the number of clients connected). We help to mitigate the impact of this by handling most network operations at the peer level (rather than the server handling administration of the clients' network connections). In this way, each client processes their queries in parallel, and only has to manage O(n) many connections to other clients. Again, this is related to the mitigiating aspect of our choice of networking protocols, specifically peer-to-peer.

Personnel Issues

One potential issue with a group as large as ours is staying organized and working efficiently, especially when communication can be somewhat difficult. Maintaining effective policies for who updates particular aspects of the program, and when, will be very important. Specifically, this will involve preventing merge conflicts by appropriately delegating different tasks and milestones to different group members, as timing expertise and opportunity dictate.