Aces Sprint #0 Documentation

September 18, 2016

Contents

1 The Team

1.1 Daniel Dyla

Daniel is a strong coder with an extensive programming background using multiple programming languages and an experienced system administrator. His core strength is in learning new technologies quickly and adapting them for use.

Daniel will be the sprint master for sprint #1, as well as acting as software engineer for the web and server components. He will also be helping to design the protocol for use by the application.

1.2 Gjergji Heqimi

Gjergi Heqimi is a proficient programmer in Java, C++, and C programming languages, as well as decently skilled with Python.

Gjergji will be acting as a software engineer for the desktop client for sprint #1.

1.3 Anthony Hewins

Anthony Hewins is proficient in Java, as well as reasonably familiar with some other programming languages. He has a strong math background and is very good at research.

Anthony will be acting as a software engineer for the server component, in addition to helping Daniel design the chat protocol that the application will use.

1.4 Joshua Lindsay

Joshua Lindsay understands various programming languages including HTML, CSS, and C. He is very organized with files and data and is a proficient researcher.

Joshua will be acting as the user interface designer for sprint #1

1.5 Ryan Richardson

Ryan Richardson is competent in Java and C++.

For sprint #1, Ryan will be acting as an interface developer for the desktop user interface.

1.6 Nathalie Tate

Nathalie Tate is an intermediate programmer with experience with a few different programming languages. She is also a competent system administrator.

For sprint #1, Nathalie will be working on the user interface deisgn, as well as acting as the interface developer for the web component.

2 Rejected Projects

2.1 Symmetric Key Encryption Module

Rejected because it requires too much specialized knowledge to implement properly, as well as being too narrowly focused.

2.2 Beer Rating and Recommendation App

Rejected because a portion of the team is not yet 21 years of age, limiting its usefulness to some members of the team.

2.3 Parking Monitor App

Rejected because it would have either required special hardware to be installed in the parking lots, or a critical mass of users feeding it data for it to be of any practical use.

2.4 Call of Halo: FPS

Rejected because it is far beyond the scope of this class, as well as difficulties securing rights to use the Halo and/or Call of Duty trademarks and intellectual property.

2.5 Class Schedule Helper

Rejected because any reasonably useful implentation would have been far too difficult to implement over the course of a single semester. Existing solutions to this problem are also already very good.

2.6 Calculator

Rejected for being far too simple.

3 Project Introduction

AceChat

3.1 Description

AceChat will be a multi-user chat application with a client/server architecture, allowing it to run on multiple platforms with minimal duplication of work. It will have public chat rooms, as well as private one-on-one chat between users.

3.2 Target Devices

3.2.1 Web

The web component will run using HTML, CSS, and JavaScript. It will be a single page application, using javascript and AJAX or websockets to communicate with the server.

3.2.2 Desktop

The desktop client will be built using Java. Even though Java should work on other platforms, we will be targeting Windows as our supported platform.

3.3 Required Resources and Materials

- Server for development and testing
- Client code
- Server code
- Windows computers to test desktop application
- Phones to test the web component on mobile
- Internet connectivity
- Database for data persistence

4 Project Timeline

4.1 Sprint #1

- Define a basic protocol for use by the client and server
- Set up a server for development and testing
- Design a user interface for the client

4.1.1 Client (Desktop and Web)

- Decide on a UI framework
- Implement basic user interface with no functionality
  - Depends on User interface being designed
- Add basic chat functionality to the user interface
  - Depends on well defined protocol
  - Depends on user interface being implemented

4.1.2 Server

- Implement basic chat protocol for use by clients
  - Depends on well defined protocol

4.1.3 Deliverable by End of Sprint

- A working server implementation
- A web client, possibly with basic chat functionality
- A desktop client, possibly with basic chat functionality

4.2 Sprint #2

- Testing and bug squashing
  - Depends on a partially (at least) working server and client implementation
- Implement fully working public and private chat functionality in desktop and client
- Begin adding extra features such as log-in, multi-line, public channel moderation, etc...
  - Depends on basic chat functionality working well

4.3 Sprint #3 and #4

- Testing and bug squashing
- Implement any extra features

4.4 Sprint #5

This sprint will be solely about testing and bug squashing, as well as getting ready for the final presentation.

5 Production for Sprint #1

5.1 Roles

5.1.1 Sprint Master

Daniel Dyla

5.1.2 Protocol Designer

- Daniel Dyla
- Anthony Hewins

5.1.3 User Interface Designer

- Nathalie Tate
- Joshua Lindsay

5.1.4 Software Engineer

- Server
  - Daniel Dyla
  - Anthony Hewins
- Desktop
  - Gjergji Heqimi
- Web
  - Daniel Dyla

5.1.5 User Interface Developer

- Desktop
  - Ryan Richardson
- Web
  - Nathalie Tate

6 Tools and Technologies

6.1 Development Platform

6.1.1 Server

The server will run on a linux server and be implemented using either NodeJS and Javascript or Python and Flask. It will be a REST interface, allowing the clients to work with it seamlessly independent of their implementations.

The server implementation will be developed using vim, emacs, and sublime text as development environments in a python virtual environment or in a NodeJS project.

The server implentation will use the Flask API framework if Python is used. If NodeJS is used, an API framework has not yet been decided on. It will also require libraries for communication using either sockets or websockets as well as database libraries for data persistence.

6.1.2 Desktop Client

The desktop client will be run on a Windows computer, communicating with the server using a REST API and/or a messaging interface over a network socket or a websocket.

The desktop client will be implemented using the Netbeans Java IDE and target Java 7 on the Windows desktop environment.

Our team has yet to decide on a UI framework to use for the desktop client, but it will require libraries for communication with the server over websockets, REST, and/or a network socket.

6.1.3 Web Client

The web client will be developed in HTML, CSS, and JavaScript It will target the Chrome web browser, but will hopefully work well with other browsers.

The web client will use vim, emacs, and sublime text as development environments.

The web client will require libraries for user interface, as well as for REST API and websocket communication with the server.

6.2 Additional Tools

- The GIMP Graphics Editor
- Microsoft Paint

- Linux
- Microsoft Windows
- Chrome Web Browser
- Chrome Developer Tools
- htop performance monitoring
- Windows Task Manager

7 Risk Analysis

We believe that we may experience problems getting the clients to work well with the server and with each other if they are developed entirely in isolation. In order to address this problem, we are going to be utilizing a test driven development methodology in order to verify that every module adheres strictly to the protocol specification.

8 Reference Materials

So far we have just used online web searches in order to find reference materials. We believe that we will have to use documentation for any programming languages that we use, as well as documentation for frameworks, libraries, and toolkits that we use. Specifically: `docs.python.org`, `devdocs.io/javascript`, and `flask.pocoo.org/docs/0.11/`, though we are sure that there will be others.