# Notepad

Brian Smith, Daniel Crowell, Ertan Dogrultan, Nathan Bergen, Sky Magnuson, Trip Volpe

**System Design Specification and Planning Document**

Draft 1

4/28/2008

CSE 403 - CSRocks Inc.

Revisions

| Version  | Primary Author(s)  | Description of Version  | Date Completed  |
| --- | --- | --- | --- |
| 0.5 | All | Initial revision - skeleton and basic info | 4/27/08 |
| 1.0 | All | First full draft  | 4/28/08  |

System Architecture

.

# Introduction

The Notepad system consists of two major program modules: the server application and the client application. The server application resides on the remote machine and is connected to a database of music projects and user account data. The client is a Microsoft Silverlight / Windows Presentation Foundation application that will be hosted on a Notepad web site and accessed through the user's web browser. Clients can connect to the server to view lists of available music projects and open them for editing. During editing, both the server and all clients editing the same file store a representation of that file locally, with the server arbitrating state synchronization between the clients.

The purpose of this document is to further describe and specify the details of this architecture.

# System Architecture Overall View



# Design view



# Process view

**Sequence Diagram 1. Sample Editing Sequence, Single User**

**Sequence Diagram 2. Music Playback**



This sequence diagram shows the music playback process - when the user issues the "Play" command, a MusicPlayer object is initialized and instructed to play, which initiates an asynchronous operation in which each staff is played simultaneously. Information about which notes are being played can be sent back to the interface, which might highlight them concurrently in a future release.

# Database Schema

The schema for our server’s backend database is fairly simple, consisting entirely of score records, user entry records (with authentication data to be added in the future), and mappings between them.

# http://docs.google.com/File?id=dgs9r8dm_4dpkmkbgk_bThis schema reflects our intention to have multiple users and songs that have a many to many relationship with each other.  For now, we plan on serializing our song object into the database, which is why the song has an image data field.

# Schema diagram

# Design Alternatives and/or Assumptions

Originally, we considered designing a solution with a standalone client/server application that could be used to construct ad hoc server sessions, as well as connect to those sessions. While this would be easier to implement than a WPF/Silverlight web client and dedicated single-point server system, we eventually chose the latter option because of usability concerns.

The primary difficulty with the ad hoc server approach is that it requires users to be responsible for setting up their own servers, distributing IP addresses, and preserving the state of a file among themselves while they are not working on it. We felt that most users interested in a collaborative music editor would not be comfortable managing IP addresses and running servers, and that the fewer steps required for our users to get up and running, the better. With that in mind we decided that a web-based client would provide the most ease of use, since not even any downloading or installing would be needed (though users would have to have an up-to-date version of the .NET framework). Designing the web client to connect to a single-point server also enabled us to make that server responsible for storing and managing users' projects, which is necessary to ensure that all users can connect to the most up-to-date version of their project at any time.

Development Plan

# Team Structure

ROLES:

Documentation and Test Lead: Sky
Midi Guru: Trip
Interface Team: Trip, Daniel, Ertan
Database/Server Logic Team: Brian, Nathan
Flex: Sky

MEETING TIMES:

Tuesday at 10:30 AM

Thursday at 2:30 PM

Sunday at 11:00 AM (sometimes)

ONLINE COLLABORATION:

Wiki: <http://cubist.cs.washington.edu/cse403-08sp/index.php/Notepad>

SVN Site: <http://www.assembla.com/wiki/show/CSE403Notepad>

TEAM STATUS:

We estimate that in the beginning, setting up the DB to save and load songs, along with UI graphics and tools, is going to take up most of our time, just to get minimal functionality. But for some of the later features, like handling fast, concurrent user activities, along with playing music, will involve more logic between the UI and the DB. We hope to store songs in the DB in a way that we can modify the format of how we store a song, without changing any DB code, so hopefully Brian and Nathan will stop touching the DB after the Beta release. Sky will be helping any wherever help is needed, especially if deadlines look like they might not be met. We already have group mailing lists, regular meetings in 022, a SVN repository, and Google Documents working very well within our team, so we do not intend to change any of that. The status of each member will obvious during our meetings, and any changes to our team structure will be made as problems arise.

# Project Schedule

*Zero feature release, due by 10pm on Wed April 30*

*Beta release, due by 9am Mon May 12*

*Beta 2 release for customer test, due by 9am Mon May 19*

*Final release, due by 9am Thurs June 6*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Task/Milestone***  | ***Estimated effort*** | ***Date due***  | ***Resource(s)***  |
| Set up Assembla  | 1 hour  | April 24  | Nathan  |
| Fill out introduction | 10 mins | April 27  | Trip  |
| Finish System Architecture Diagram  | 30 mins  | April 27  | Sky |
| Finish Class Diagram | 2 hour  | April 27  | Ertan |
| Finish Sequence Diagram 1 | 1 hour | April 27  | Daniel  |
| Finish Sequence Diagram 2  | 1 hour  | April 27  | Ertan  |
| Finish Database Schema | 30 mins  | April 27  | Brian  |
| Finish Team Structure | 10 mins | April 27 | Sky  |
| Outline Project Schedule  | 30 mins  | April 27  | Sky  |
| Fill Risk Assessment | 30 mins  | April 27  | Trip  |
| Create Test Plan | 30 mins | April 27 | Nathan  |
| Create Documentation Plan  | 30 mins  | April 27  | Nathan  |
| Create Presentation Slides  | 1 hour  | April 27  | Daniel, Trip, Sky |
| Turn in the SDS | 30 mins | April 28 | Trip  |
| Present SDS | 10 mins | April 28 or 30 | All |
| Have a persistant DB running on iprojsrv, and entire team given admin privileges  | 1 day | April 29 | Brian |
| Have a web page anyone can view | 1 day | April 29 | Trip |
| Have a skeleton framework in the repository | 2 day | April 29 | Nathan |
| Have an automated build running daily | 1 day | April 29 | Sky |
| Write up ZFR write-up | 1 day | April 30 | Ertan, Daniel |
| Submit ZFR, and any write-up | 30 mins | April 30 | Trip |
| Get Buttons, text, pictures to show up on a browser using our skeleton code | 1 day | May 1 | Ertan |

**Project Schedule – Continued**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Task/Milestone***  | ***Estimated effort*** | ***Date due***  | ***Resource(s)***  |

|  |  |  |  |
| --- | --- | --- | --- |
| Get the DB and a browser (Firefox) to interact in any way | 1  day | May 4 | Nathan, Brian |
| Have a set of musical symbols that we can use | 1 day | May 4 | Trip, Daniel |
| Have a firm set of methods allowing contact with the DB, might not be fully implemented | 1 day | May 6 | Nathan, Brian  |
| Add / delete musical notes / rests to a score with mouse | 1 day | May 6 | 2 UI members |
| The ability for the DB to save a whole song object, and then load it out again | 2 days | May 8 | Nathan, Brian  |
| Account creation, login, logout all work | 1day | May 8 | 1 UI member, Brian  |
| Song creation, deletion, renaming to all work | 1 day | May 8 | 1 UI member, Brian  |
| Grant other users the ability to alter scores of your own | 1 day | May 8 | 1 UI member, Brian  |
| Add accidentals w/mouse | 1 hour | May 8 | 1 UI member |
| Unit tests for many classes | constant discipline | May 8 | All |
| Submit Beta | 30 mins | May 9 | Trip |
| Add multiple lines of music | 1 day | May 12 | 1 UI member |
| Supports key signatures | 1 day | May 12 | 1 UI member  |
| Add support for altering scores with keyboard | 1 day | May 19 | Daniel |
| Allow multiple contributors at the same time | 2 days | May 19  | Nathan, Brian |
| Automatically delineate measures | 2 days | Jun 6  | unknown  |
| Chat with other users | 2 days | Jun 6  | Nathan, Brian, Daniel |
| Change tempo | 1 day | Jun 6  | Ertan |
| Supports Chords | 1 day | Jun 6  | unknown  |
| Allow different clefs | 1 day | Jun 6  | unknown  |
| "Legality" checks on the user's note placement | unknown | Jun 6  | Nathan |

**Project Schedule - Continued**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Task/Milestone***  | ***Estimated effort*** | ***Date due***  | ***Resource(s)***  |

|  |  |  |  |
| --- | --- | --- | --- |
| "Grammar" checks on user note placement (slur note & rest) | unknown  | Jun 6  | unknown  |
| Convert the music to a playable format | unknown  | Jun 6  | Trip |
| Play music w/highlight | unknown  | unknown  | unknown  |
| Highlight locks / other user cursor positions | unknown  | unknown  | unknown  |
| Read in Midi Files | unknown  | unknown  | Trip |
| Export to a visual format | unknown  | unknown  | unknown  |
| Print musical scores | 1 day | unknown | unknown  |
| Allow multiple sheets per score  | 1 day | unknown | unknown  |
| Side by side comparison view (by measure) | 1 day | unknown | unknown  |

# Risk Assessment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk**  | **Chance of occurring** | **Impact** | **Preventative steps** | **Mitigation plan** |
| Unable to get application and database running correctly on Windows server |  Low  |  Medium  | This would mainly be an issue if left to the last minute, so we will be sure to test code on the server as soon as possible. | Use a personal home-based machine as a development server while finding an alternative. |
| Tech: Silverlight proves difficult to use for UI implementation due to differences from familiar APIs |  Medium  |  High | We can begin learning Silverlight's API early in the project to become familiar with its quirks and differences from WinForms. | Switch to a standalone application using WPF only, instead of WPF and Silverlight.  |
| Performance: state synchronization doesn't scale well, and client interface suffers lag   |   Low  |   High  | We are designing state synchronization to reduce the size of update messages, and to only query the database during serialization. This should keep network traffic low, so latency depends mainly on user ping time, which is (mostly) beyond our control. | As a last resort, limit the number of active users; too many users would make the editing experience hectic anyway. |
| Requirements: Clients request that more features be moved into the final release. | Medium | Medium  | We have discussed desired features carefully with the clients to hopefully ensure that our specs will satisfy their needs. | If there is sufficient time, we would attempt to accommodate clients’ wishes; otherwise, meet with clients again to discuss priorities. |
| Scheduling: We begin running out of time to implement final features as deadline nears. | Medium |   High | We are creating and hopefully will be paying attention to our schedule. We may designate one person to maintain team focus on high-priority items. | After putting in lots of overtime, if necessary we might discuss some goal shifting with the clients. |

Test and Documentation Plan

# Test Plan

**Testing Strategies**

1. **Unit Test Strategy**
	* **Coverage/Purpose**
	Our unit tests will be designed to test the (correct) functionality of individual components.  Ideally, our unit tests will provide 100% coverage from the bottom up.  Each class will have unit tests written for it to correctly verify all of the intended functionality for that class.
	* **Who is responsible for tests?**
	Our unit tests will be written by the same team member that writes the corresponding class.  If multiple users work on the same class, either collaboratively, or on varying features, it remains their responsibility as authors of the class to define, code, and test the class for completeness.
	* **Testing frequency**
	Unit tests will be run before a commit is made (as a white-box test).
2. **System Test Strategy**
	* **Coverage/Purpose**Our system tests will be used to test the interactions of various parts of our product.  Used both as Black-Box and White-Box, we will use system tests to verify proper integration between old features and new, as well as between simultaneously developed new features.
	* **Who is responsible for tests?**The tests will be developed in tandem with new features, by individual team members as they develop new features.  Developers will be primarily responsible for making sure that new code works with the old, but there will be some smaller emphasis on modifying older code and test harnesses in response to published bugs as well.
	* **Testing frequency**The tests will be run often during the development of new features, and also on the production code once it is integrated at the conclusion of the development with the new features.
3. **Usability Test Strategy**
	* **Coverage/Purpose**Our usability testing is intended to measure the often difficult to measure components of the project.  Intangible assets such as 'how easy is our product to use', and 'how useful is our product' and 'how functional is our product'
	* **Who is responsible for tests?**The tests will be largely black-box tests, run from a user's perspective.  They will be performed manually by us as developers, as well as anyone we can informally or formally subject to our product (just kidding, they'll love it).  As a result, the tests will be run on the latest production build of our project, and so will not require any formal development process outside the realm of our standard deployment practice.
	* **Testing frequency**Starting before our beta product, our usability tests will be run frequently throughout the development process, near-constantly by ourselves as developers, and somewhat less frequently by non-team members that volunteer or agree to take a gander at our product in the name of functionality and usability testing.

**Adequacy of Test Strategy**

Between our three test strategies, our system of testing will be robust, but also dependent on the willingness of developers to manufacturer test cases pertaining to their assigned components.

**Bug Tracking and Plan of Use**

Bug tracking will use the version of Trac that is integrated into Assembla (assembla.com) that we are using to control our SVN repository.  This single-interface for much of our product should greatly enhance our productivity and the ease of filing / resolving / dealing with bugs.  Bugs will be utilized in a fairly standard manner.  If a developer runs into a bug at any point in the development process, he should file a bug, assigning it to the relevant user.  Our team is small enough that it should be relatively easy enough for developers to keep track of which project we are working on.  Developers are responsible for maintaining / keeping their buglist in check.

# Documentation Plan

By the nature of our project, every client is a user.  Admins only exist to the point of the creation of the song, at which point the user is the 'admin' of the song.  Additional users that are added to the song's ACL then become admins within the context of the song as well.  As a result, the vast majority of the documentation associated with Notepad will be in the form of a user-guide.  We hope to make our application as intuitive as possible, but this is of course not a replacement for proper documentation.  Our user manual will be written as each feature is completed, from a user's perspective.  Feature documentation will include robust descriptions of the functionality and intended use of a particular feature, as well as how to make use of it as a user.

In addition, we recognize that a typical user is often resistant to spending the quantity of time necessary to make use of the manual fully, and so we intend to release a 'quick use guide' which will cover the most basic features as fully as possible in a minimum amount of time.  Also, the UI will contain helpful hints in some form to explain what various aspects of the UI do.

In terms of true admin documentation, our product is designed that it be deployed in a single location as a web application, and as we do not intend for it to be universally distributable (rendering ineffective our concept of a single server design, and infringing our IP), the documentation regarding the (un)installation of our product will be geared towards us as the developers, those familiar with the project, and intended to be used in conjunction with a current developer for those not familiar with the project (new team members).  As it is internal documentation, formal admin documentation will be somewhat of a lower priority than our formal customer documentation, and likely take the form of well-rounded and commented code, making the intent and function of internal devices clear to the developer.