

Deliverable III
CoNetPad
TheACMGroup (Cesar Ramirez, Justin Anderson, Josh Tan)

I. User Manual

1. Requirements

CoNetPad has been programmed using java. To run this application you need a Java Runtime Environment, you can download it from:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Please follow the instructions to your respective platform.

2. Extract the installer

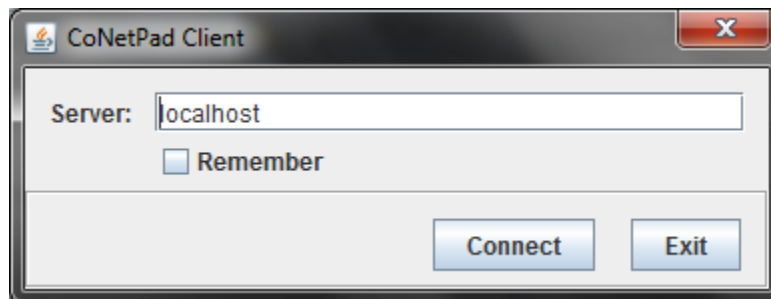
CoNetPad comes into a .zip folder named using the convention conetpad-{dateofpublication}.zip. This file contains both the server, client and all the libraries required to run the applications. Uncompress the folder in any location where you have access to read and write, a good location is usually you home directory.

3. Start the server

Go to the installation folder and then to the 'server' folder. In this folder you are going to find 'conetpad-server.jar', 'data' and 'lib'. To start the server, double click in conetpad-server.jar or run 'java -jar conetpad-server.jar'.

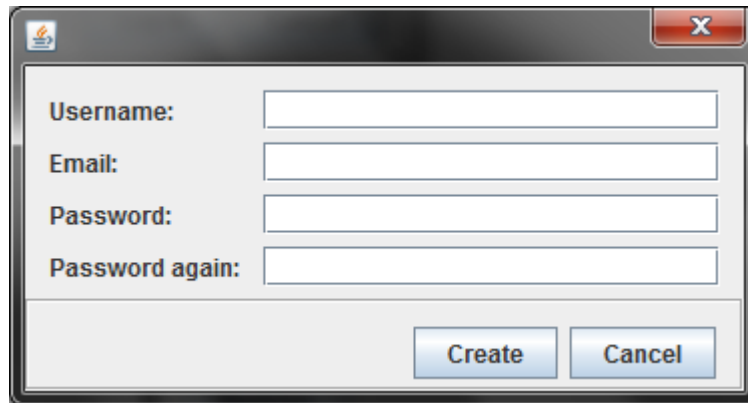
4. Connect to the server

To connect to a server, make sure you have the IP address of the server and your firewall is set to allow connections in port 4444. Once you have this information go to the installation directory and then to 'client'. In this folder you should double click conetpad-client.jar or run 'java -jar conetpad-client.jar'. A window will appear, input the IP address of the server into the text field and click connect. If the connection is successful you will be directed to the next window, if the connection failed you will be notified.



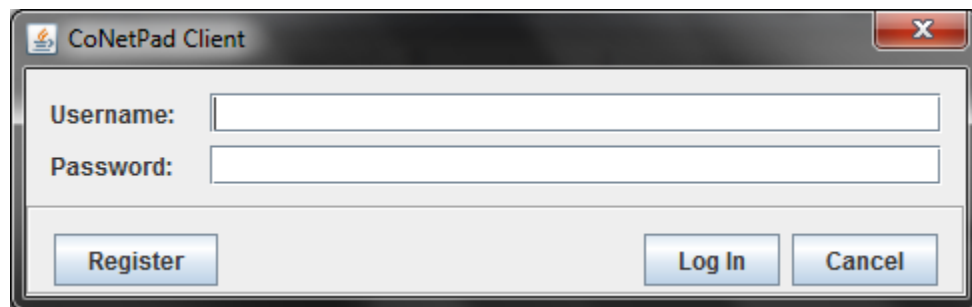
5. Create an account

In the next window you can create an account by clicking on 'Register'. In this window you should fill the information required and click 'Create'. You will receive a pop-up window confirming your account was created.



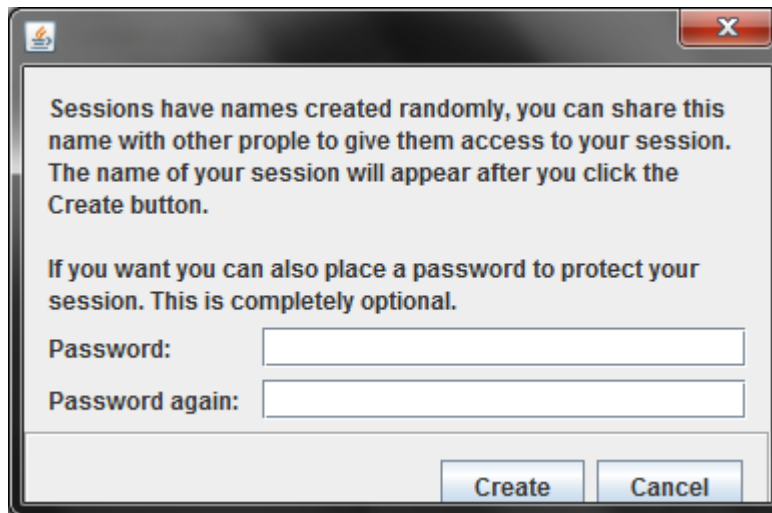
6. Log in to the server

Once you have an account you can enter your credentials in the fields and click 'Log in'. If the login process was successful you will be presented with a dialog to join or create a session.



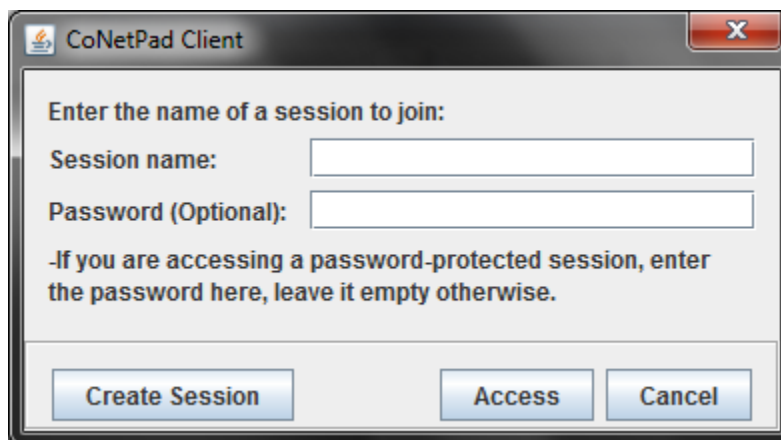
7. Create a session

If you do not have a session or you want to create another session, click I the 'Create Session' button. Remember that you can not choose the name of your session, put you can use a password to protect it if you decide to. If you do not want to use a password, just leave this fields empty. Once you are ready to create a session, click on 'Create'. You will receive a pop-up with a confirmation that the session was directed and you will be shown the session log-in screen. On this screen you can see the name of your session, share this name with anyone you want to give access to your session.



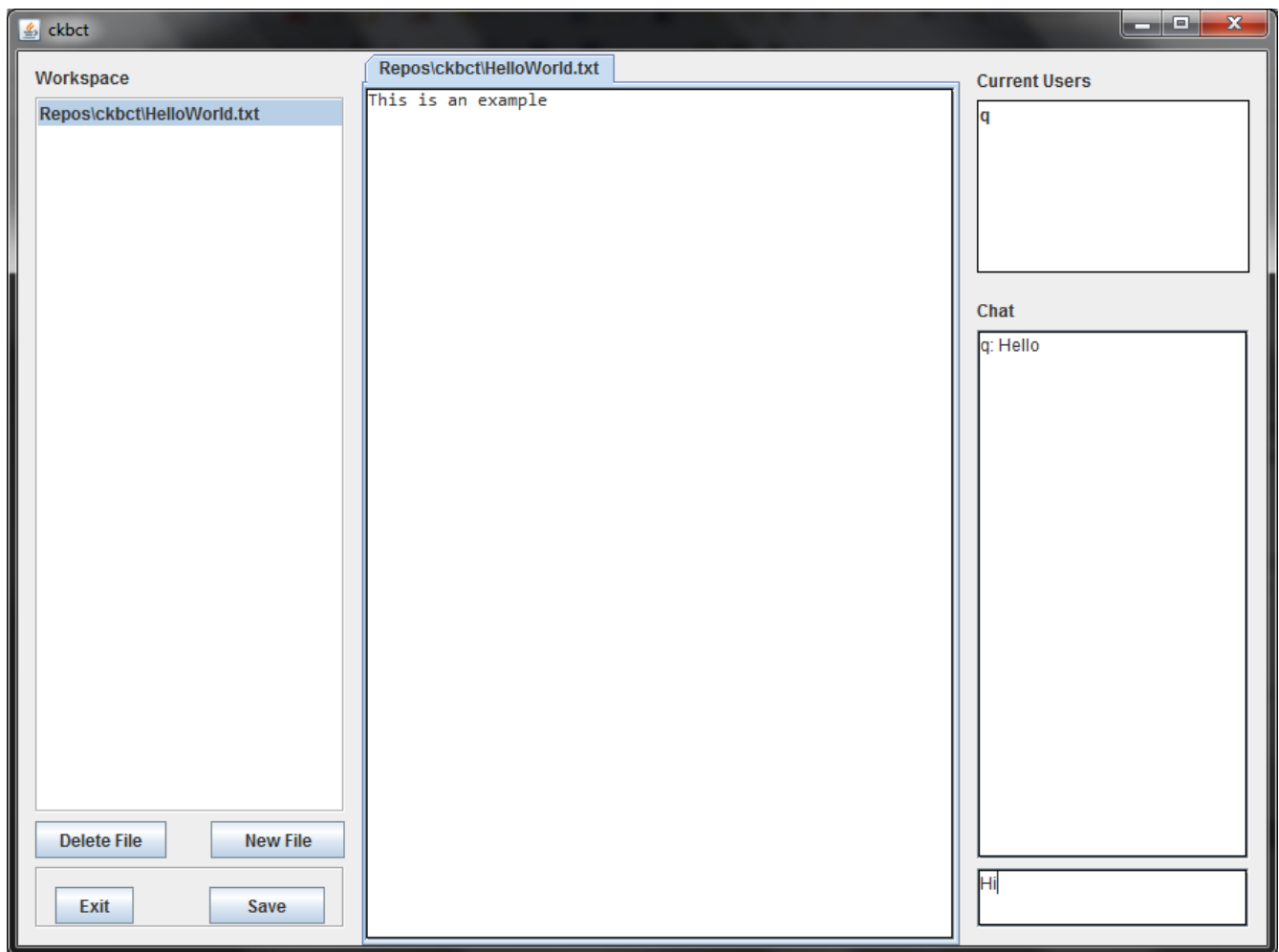
8. Join a session

If you have access to a session, on the session log-in screen input the session name and the password and click 'Access'. If you have access to the session you will be redirected towards the editor, if not, a pop-up will appear indicating the error.



9. Understanding the interface

The CoNetPad editor has a simple and clean user interface divided in three areas. On the right panel you can see the users that are currently logged in into the session and you can also use the text box at the bottom to communicate with all of them. On the right side is the list of files as well as the 'New file', 'Delete file', 'Save' and 'Exit' buttons. The central panel is used to edit the files that you have opened.



10. Send a chat message

If you want to communicate with the people logged in, you can do so by typing a message in the text box located in the lower-left corner and then typing enter. The message will be displayed on the text area right above it, all users connected will receive the message

11. Edit a file

The center of the application is used for file editing. If you double click on a file from the list, a tab will open with the content of the file. The content on this panel is synchronized across all users, any changes you do will be displayed on realtime to all connected users.

12. Other Actions

On the right side you will find the list of files that currently exist in the session as well as some actions that you can take. You can click on 'New file' to create a new file with a name of your choice. You can also select a single or multiple files and click on 'Delete file' to delete them. Both 'New file' and 'Delete file' require you to have special permissions in order to use them. The 'Exit' button will close your connection and the 'Save' button will cause the server to write the current changes to the disk.

II. Directions for Compiling and Running CoNetPad

1. Compiling

CoNetPad can be compiled using the Apache Ant build tool. To compile the CoNetPad source code and package it into executable .jar files, execute “**ant dist**” from the root directory of the repository. This command will place the compiled .class files into the “build” directory and the executable .jar files into the “dist” directory.

The executable .jar for the client can be found at “**dist/client/conetpad-client.jar**”. Running this jar will require the “dist/client/lib” directory to be in the same directory as the .jar file.

The executable .jar for the server can be found at “**dist/server/conetpad-server.jar**”

A .zip file containing both the client and server executables will also be placed in the “dist” directory.

2. Running

To run the CoNetPad client, navigate to the “**dist/client**” directory and execute “**java -jar conetpad-client.jar**”.

To run the CoNetPad server, navigate to the “**dist/server**” directory and execute “**java -jar conetpad-server.jar**”.

Note: In order for this application to function correctly, you may need to modify your firewall settings (if applicable).

III. Automated Test Case Information

We have chosen to use JUnit test cases to test our application. In creating our JUnit test cases, we have focused on non-trivial parts of the code (e.g. ignoring setter and getter methods). In cases where the unit testing approach was not possible without creating extensive stub classes, we have performed manually functional testing. This was the case for those parts of the project that involved to sending and receiving tasks through a network using threads.

To automate the running of JUnit test cases, navigate to the root directory of the repository and execute “**ant test**”. This will execute all JUnit test cases for our project and display their results on the console. In addition, more detailed reports for the JUnit test cases can be found in “**test_reports/TEST-<ClassName>Test.txt**”.

IV. Discussion of Features

1. Implemented

We have successfully implemented communication between the client and server over a network. Clients can connect to the server, create session and files, and perform concurrent

edits on these files. In addition, clients can chat with one another, with chat messages being processed in order. It was decided to drop the feature whereby clients could connect an external chat client to the session, in favor of a centralized chat architecture.

Clients can create (and delete) multiple files and these files are stored in a session Git repository on the server. Both user accounts and sessions can be made private by requiring password authentication.

2. Not Implemented

Given the amount of features that were planned, many have not yet been implemented. The next phase of our project will focus on implementing these features, in order of the priorities specified in the requirements document. These features include those related to security, such as using SSL sockets for client-server communication. The hierarchy of user control will need to be implemented, by limiting file creation and deletion to the session leader. If CoNetPad is to be deployed in an educational setting, controls for restricting chat among session users will need to be created for the session leader.

Features related to files and Git repositories will also be implemented in the next phase. These include remote compilation of source files, the ability to download source and binary files, and the ability to clone session Git repositories.

Lastly, we still need to implement features related to source file editing. This will include distinct highlighting of source code, depending on the user that made the edit. Two other features are syntax highlighting and auto-complete functionality. Although these features are not essential to the goal of CoNetPad, they will greatly improve the user experience.

V. Meeting Minutes

Meeting Minutes for August 30, 2012

- 1) Determined which features are required, and the features that were additional.
- 2) We finished deciding on the project details.
- 3) We came up with a project title.
- 4) We broke the project into parts and came up with a tentative plan for the project.
- 5) We came up with a list of possible risks.
- 6) We came up with a tentative idea/plan for the presentation.

Meeting Minutes for September 6, 2012

- 1) Began working on formally defining requirements
- 2) Looked over SRS template
- 3) Discussed different possible approaches to system design

Meeting Minutes for September 13, 2012

- 1) Discussed each section of the SRS template

2) Decided on way in which Git will be used with CoNetPad

Meeting Minutes for September 20, 2012

- 1) Talked and made decisions on our database
- 2) Talked and made decisions on design documents
- 3) Discussed and worked on deliverable 2

Meeting Minutes for September 27, 2012

- 1) Discussed the remaining tasks that need to be done for Deliverable II
 - a) explanation of the architecture of the system
 - b) UML design documents
 - i) class diagram
 - ii) sequence diagram
 - iii) use case diagram(s)
 - c) ER diagram / relational schema
 - d) test plan
 - e) updated risk management
 - f) updated project plan
- 2) use case diagram specifics
 - * actors: user (self), users (other than self), IRC server (optional; only if feature allowing connecting to external IRC server is implemented)
 - * use cases:
 - ** download source/binary
 - ** login/create account
 - ** create/delete/modify files
 - ** edit user settings
 - ** chat
 - ** clone Git repo
 - ** compile source
- 3) specific classes that will likely be used
 - * for the server:
 - ** Repository
 - ** Account
 - ** Session
 - ** Compiler
 - ** IRCServer
 - ** Server
 - ** CommandQueue
 - * for the client:
 - ** UI
 - ** Connection
 - *** ServerConnection

*** IRCCConnection

** User // may or may not have an associated Account

** IRCCCommand

* shared classes:

** SSLSocket

** File

*** ServerFile

*** ClientFile

// Note: these are tentative - will likely change as we progress in the design process

4) specific information that the SQLite database will likely need to handle

* user information

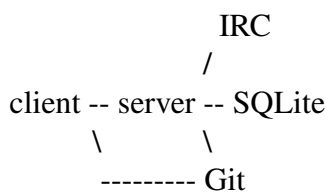
** userid

** username

** password // hashed

** user-specific salt

5) specifics on architecture:



* 3 tier system

* client-server relationship between all directly connected components

6) Plan/Assignments:

* we plan to meet at 9am on Tuesday, for whoever can make it (Cesar and Josh confirmed)

* individual goals for Tuesday:

** Josh - class diagram, system architecture, and updated project plan (also will try to finalize the SRS)

** Cesar - sequence diagram, use case diagrams, updated risk management, and test plan

** Justin - ER diagram, relational schema, user interface layout/design

* the overall goal is to get the majority of Deliverable II done by Thursday

Meeting Minutes for October 2, 2012

1) Discussed what each member was able to accomplish.

2) Listed the components that still needed to be finished and divided the work among members

3) Agreed upon the overall, high-level design for the classes.

Meeting Minutes for October 9, 2012

1) Discussed the ER Diagram for the database. Decided on the various tables that the database

would hold.

- 2) Discussed the system architecture and checked with Dr. Do to ensure it was appropriate.
- 3) Discussed changes that needed to be made to the use case diagram - needed to add a few use cases.
- 4) Divided the remainder of the project tasks into subtasks - to be used in the updated project plan.
- 5) Assigned work to be done that day for the tasks still unfinished:
 - Cesar: use case diagram/text fixes, sequence diagram fixes
 - Justin: ER Diagram, System Architecture
 - Josh: UML Class Diagram, Project Plan, meeting minutes

Meeting Minutes for October 11, 2012

1) Changed the design of the server. Instead of having a TaskQueue located in CNPServer, each ServerSourceFile will have its own TaskQueue (that only serializes Tasks per file). In the new design, the task processing is as follows:

- CNPClient sends a CNPTask to the CNPServer through the SSLSocket using serialization.
- The CNPServer receives the task data on its SSLServerSocket, unserializes it back to a CNPTask, and forwards it to the TaskDistributor.
- Using knowledge of the CNPSession and modified file (stored in CNPTask), the multi-threaded TaskDistributor forwards the CNPTask to a specific ServerSourceFile's single-thread TaskQueue.
- CNPTask changes are made to the ServerSourceFile.
- The ServerSourceFile sends another CNPTask to the TaskDistributor (using a different type of CNPTask).
- The TaskDistributor forwards a serialized version of the CNPTask (in parallel) to all CNPClients in the CNPSession that have the file open (probably implemented by registering interested CNPClients as Listeners of some sort).
- CNPClient receives the CNPTask on their SSLSocket, unserializes it, and forwards it to its own multi-threaded ExecutorService.
- This ExecutorService will execute each CNPTask and update the CNPClient's view to reflect the changes.

2) Defined interfaces for CNPServer/CNPClient, Network/Logic, and Logic/Storage.

3) Divided project into logical components and assigned components to group members (see ProjectDivision.png). Current assignments are:

- Cesar = Network (both Server and Client) - work with classes like CNPConnection, SSLSocket/SSLServerSocket, IRCServer, CNPSession
- Justin = Logic (Client) - work with classes like SourceFile, CNPClient (especially its ExecutorService), CNPTask

- Josh = Logic (Server) - work with classes like CNPSession, CNPTask, TaskDistributor, TaskQueue

Meeting Minutes for October 15, 2012

1) Made major changes in design:

a) The server will listen on a single port and will manually spawn a thread for each client that connects to it.

b) Client/Server no longer communicate tasks using an executable (runnable/callable) task object.

i) Send only plain-text parameters as a string (using delimiters) that specify a particular task

ii) Client/Server should construct the task object on their own side, based on these parameters.

iii) Client/Server will then forward the task object to the appropriate ExecutorService.

* Client will forward it to clientExecutor

* Server will forward it to the appropriate ServerSourceFile's taskSerializer (a single-thread ExecutorService)

iv) CNPTask no longer needs to contain a CNPConnection (these will be stored on each side after a client connects)

c) the permissionType field of a CNPTask should be static

2) We should also work on converting documentation to a plain-text format, whenever possible. This way, changes can be made more easily and version differences are simpler to view.

3) TODO: Interface specification needs to be updated.

Meeting Minutes for October 16, 2012

1) Will no longer use IRC component. CNPServer will handle chat functionality on its own.

- Note: This means clients can no longer connect to chat server with external IRC client.

2) Messages should be passed between client/server using JSON.

- Note: For now, can use basic string parsing.

Eventually, should reuse Java JSON library.

Meeting Minutes for October 23, 2012

1) No longer need a Java interface for the client/server. The JSON API will serve as the interface.

(Still need to explicitly define the API.)

2) Will use GSON as Java JSON library. Will use both JUnit (for unit testing) and some form of functional testing.

- 3) Will introduce Network class to abstract network aspect of system. Probably will have an abstract Network class
which is extended by a ClientNetwork and ServerNetwork classes.

Meeting Minutes for October 25, 2012

- 1) Agreed on source code directory structure.
- 2) Discussed Java package naming conventions.
- 3) Decided to prioritize storage part (Git, SQLite) over JSON API specification.

Meeting Minutes for November 13, 2012

- 1) Discussed what each person is working on.
- 2) Decided to merge each person's own branch into master frequently, as long as there are not build errors.
- 3) Put the creation of constructors/fields for Tasks on the high priority list.

Meeting Minutes for November 16, 2012

- 1) Discussed how Task and TaskResponse work with messages and the message factory.
- 2) Figured out stuff for EditorTask.
- 3) Modified how database treats CNPSession and CNPPriateSession.

Toward the end of the project, many meeting were conducting online using email and chat messaging.