HiveText

Design Document

Team 25

1/30/2018

George Albrecht
Arun Arjunakani
Bryan Battles
Alex Geier
Paramesh Pradeep
Guangqi Sun

Index

Purpose

In certain collaborative situations, software developers may wish to work concurrently on various types of simple text files. These situations involve multiple users making numerous edits to the same file, that are visible to everyone in real time. We will develop a desktop application, HiveText, that allows users to work simultaneously on any simple text file. It will be similar to existing text editors such as Sublime Text or Notepad++, and similar to other collaborative tools like git and Google Docs, but will improve upon these existing tools by allowing users to work concurrently on files while providing visual tracking of changes. A consensus will be required to permanently save user changes. In addition, our application will provide users with a chat box with several channels for communication between collaborators working on the same file.

Design Outline

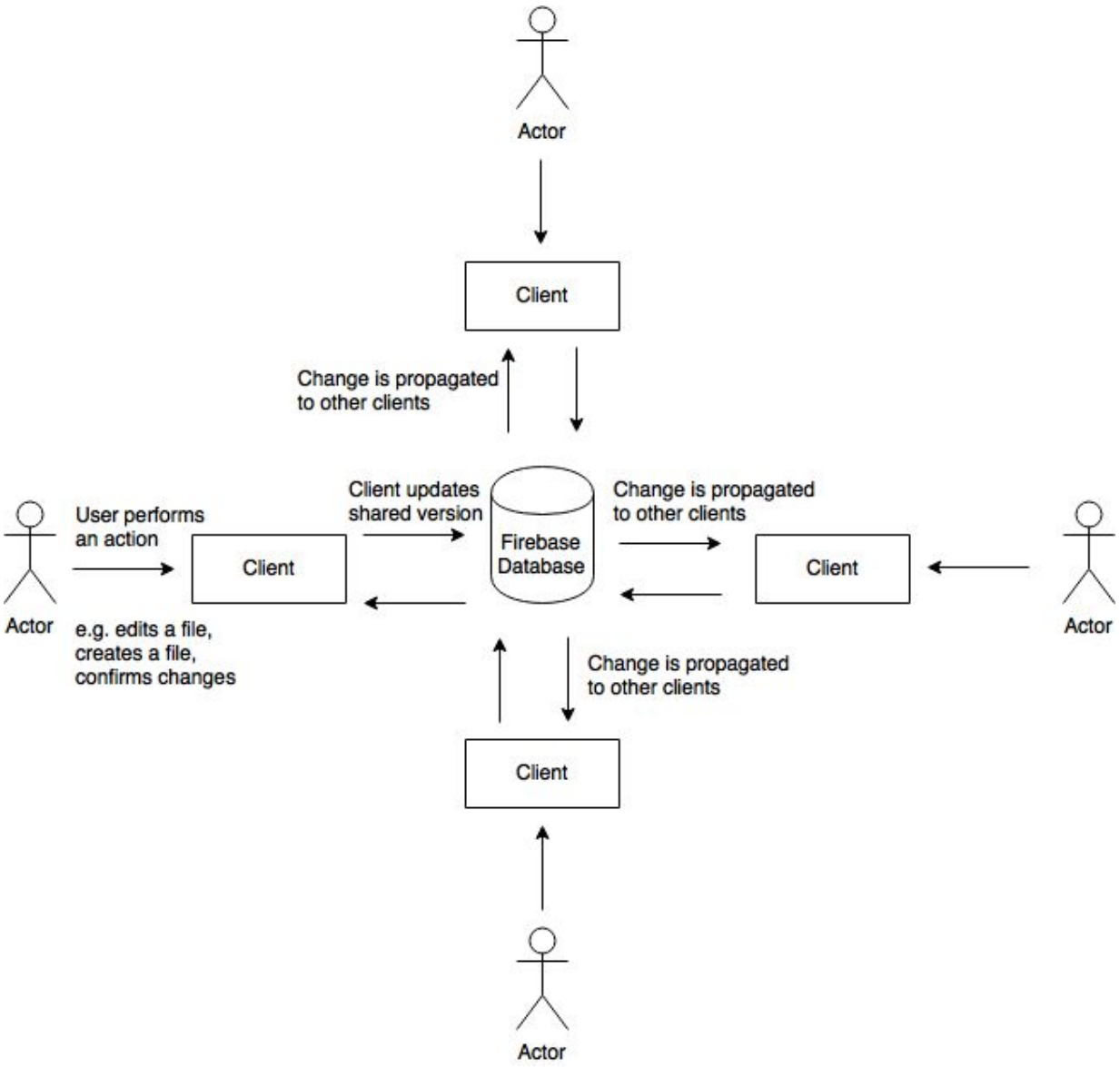
1. **Electron Client**
   a. **The Electron client will act as the interface of our system.**
   b. **The client will retrieve and send data to Google Firebase via the Google Firebase API.**
   c. **Electron will receive data from the server, and then parse, format, and present the data to the user through display information such as the updated version of the current file, current online users, and messages sent in the chat box.**
   d. **The Electron client will act as a thin client, with most of the computation performed by the server.**

2. **Google Firebase Server**
   a. **The Firebase server will provide an abstraction layer on top of our database to make it easy for clients to request and send information through javascript functions.**
   b. **The Firebase server will query the database to retrieve information for the client.**
   c. **The Firebase server will post information to the database in JSON form.**

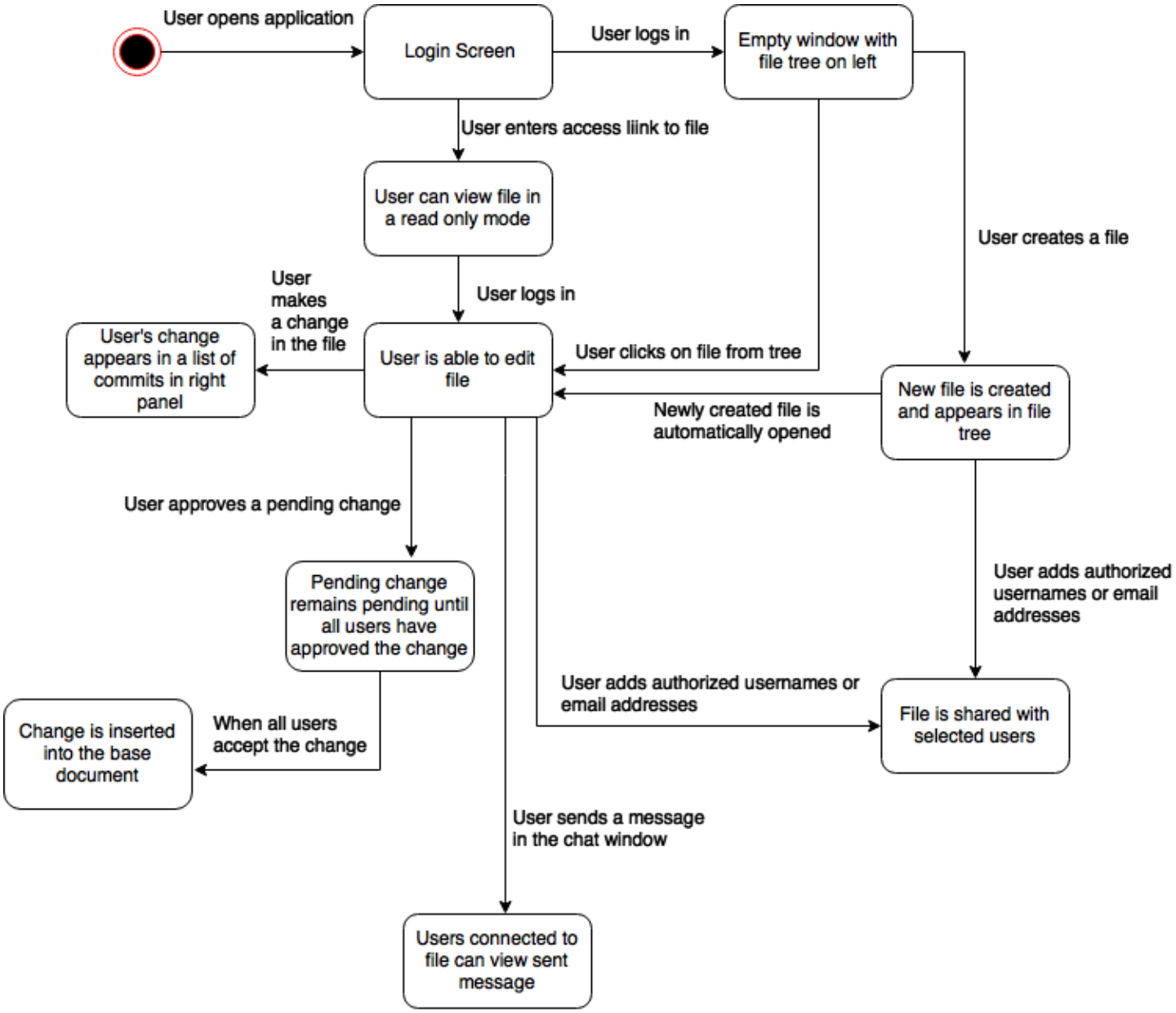
3. **Google Firebase Realtime Database**
   a. **Data will be stored in Google Firebase as a JSON tree of JSON objects.**
   b. **Typical data stored in the database will be the list of users, current version of a file, edits made to files, commits made on a file, users connected to a file, and chat logs.**
   c. **Changes to the database are distributed to all other clients using the database.**

4. **Ace Text Editor**
   a. **Ace is an embeddable text editor written in JS which will be the primary means through which the user will interact with the client.**
   b. **Provides syntax highlighting support for 45+ languages**
   c. **Auto-indenting and line numbering**

High-level Structure



Activity / State Diagram



Design Issues

Functional Issues

1. In what situation will the commit will be accepted or denied?
   Option 1: The commit will be accepted in any case.
   Option 2: The commit will be executed by an administrator.
   Option 3: The commit will be accepted if all of the online users agreed.

   Discussion: We chose to design the program to accept the commit if all of the online users agreed. The main feature of the program is concurrent editing for a group, so an agreement should be reached before changing the file permanently. If time allows, we would like to add functionality for an optional admin permission that allows the admin to commit all current changes without a consensus.

2. Does the user have to be online to use the program?
   Option 1: The user cannot use the program when offline.
   Option 2: The user can use the program locally with their current version

   Discussion: We plan to make the program available when the user is offline, but they will not be able to commit until they back online. For example, the user could continue his work on a flight or in a place with unstable internet connection, and the program will create a timestamp for his commits and stage their changes once the user connects to the server again.

3. What database is most appropriate for our project?
   Option 1: Google FireBase
   Option 2: mySQL
   Option 3: mongoDB

   Discussion: We decided on Google Firebase because we wanted a fast, non-relational database that allowed concurrent access. FireBase provides atomicity and propagation of changes to all users, while being non-relational in data structure.

4. Does the user have to have an account to use the program?
   Option 1: The user does not need an account to use the program.

Option 2: The user has to have an account to use the program.
Option 3: The user could use the program without an account but only has reading permission.

Discussion: We decided that the user could use the program without an account if they have an access link to a file, but only with reading permission. A user may want to share a file with others that are not contributors and any unauthorized change should be prevented.

5. How will the user permission be managed?
   Option 1: Every user has reading and writing permissions.
   Option 2: Every user has reading and writing permissions, but a administrator can change other user's permission.

   Discussion: We decided to have a administrator in our system base on security concerns, unauthorized changes could be prevented by doing so. The administrator could be the manager or the head of the project team.

Non-functional Issues:

1. Should the user be able to save the file locally?
   Option 1: User can save the file locally.
   Option 2: User can save the file locally if they are offline.

   Discussion: We decided to let the user save the file locally if they are offline since the user cannot commit along with other users. The client will save a local copy of the file as the base to record the offline user's edition.

2. How will editions be visualized in the file?
   Option 1: No visualizations.
   Option 2: Add a tag after user's edits
   Option 3: Assign a unique color to each user, highlight their editions with their color. Use strikethrough to visualize deletion.

   Discussion: We decided that color highlighting should be used to visualize other user's contributions clearly. Deletions should be kept in order to track edits across multiple users.

3. How will users communicate with each other?
   Option 1: Short messages like email.
   Option 2: Real-time chat like Slack.
   Option 3: In-line comments

   Discussion: We decided that the program should have the functionality of real-time chatting if time allows. The purpose of the program is to provide a platform for concurrent editing, it will be helpful if the group members can exchange their opinions within the program in real time. In-line comments should be reserved for code descriptions and could lessen the clarity of the code written.

4. Will plugins be allowed?
   Option 1: No plugins, all functionality is native.
   Option 2: Allow plugins.

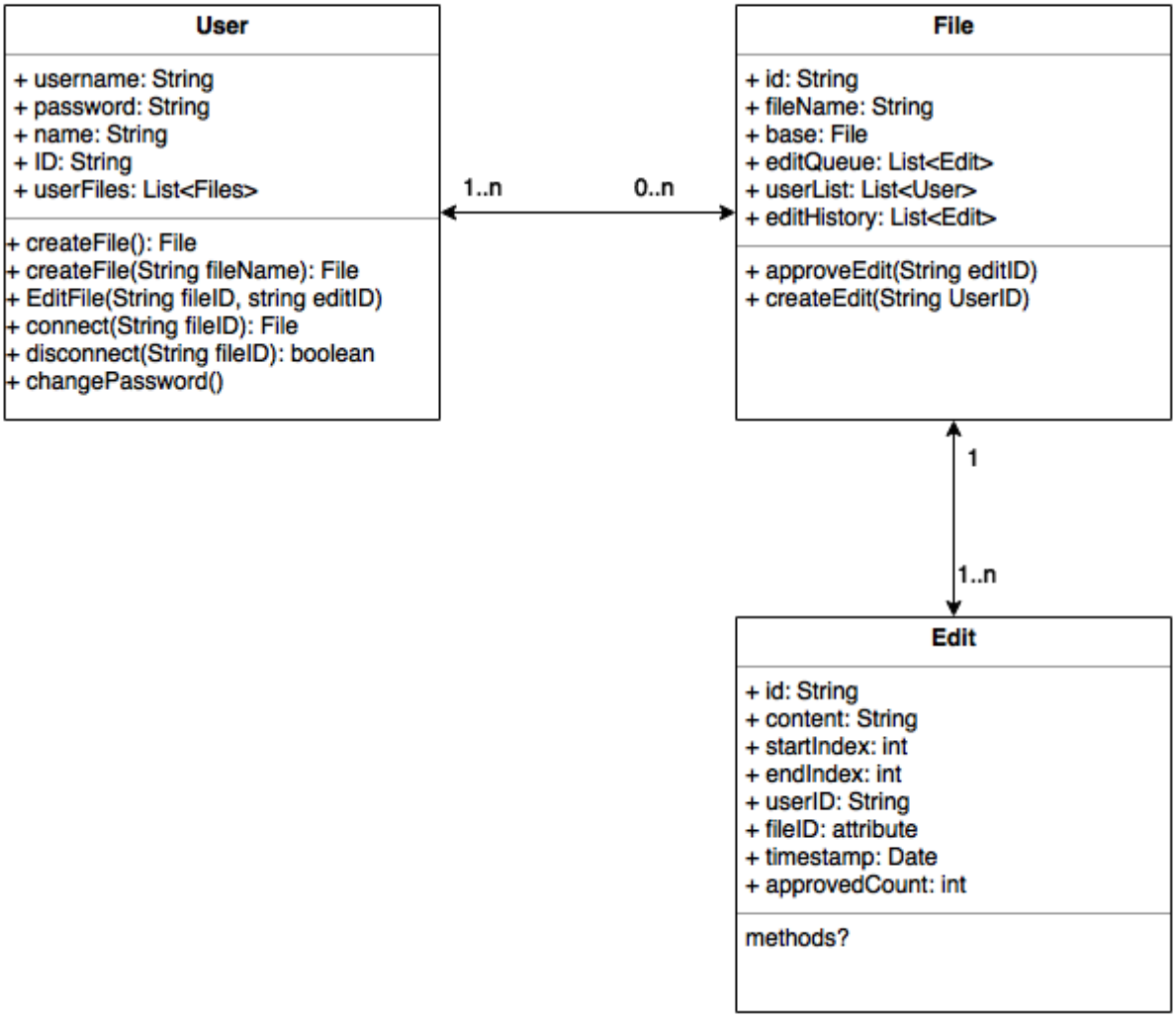
   Discussion: We decided that plugins should be allowed if time allows. Plugins could increase the functionality of the program and it fit into different working scenarios, which increase the usability of our program. Compatibility of plugins is a very interesting feature for the program, yet we think it is hard to achieve within the given time.

5. How will the file be shared with people not in the group?
   Option 1: Through a link via email.
   Option 2: Send a copy of the file.

   Discussion: We decided to share the file using a link via email due to security concerns. By clicking the link, a person could have access to the file with read and write permission on their account.

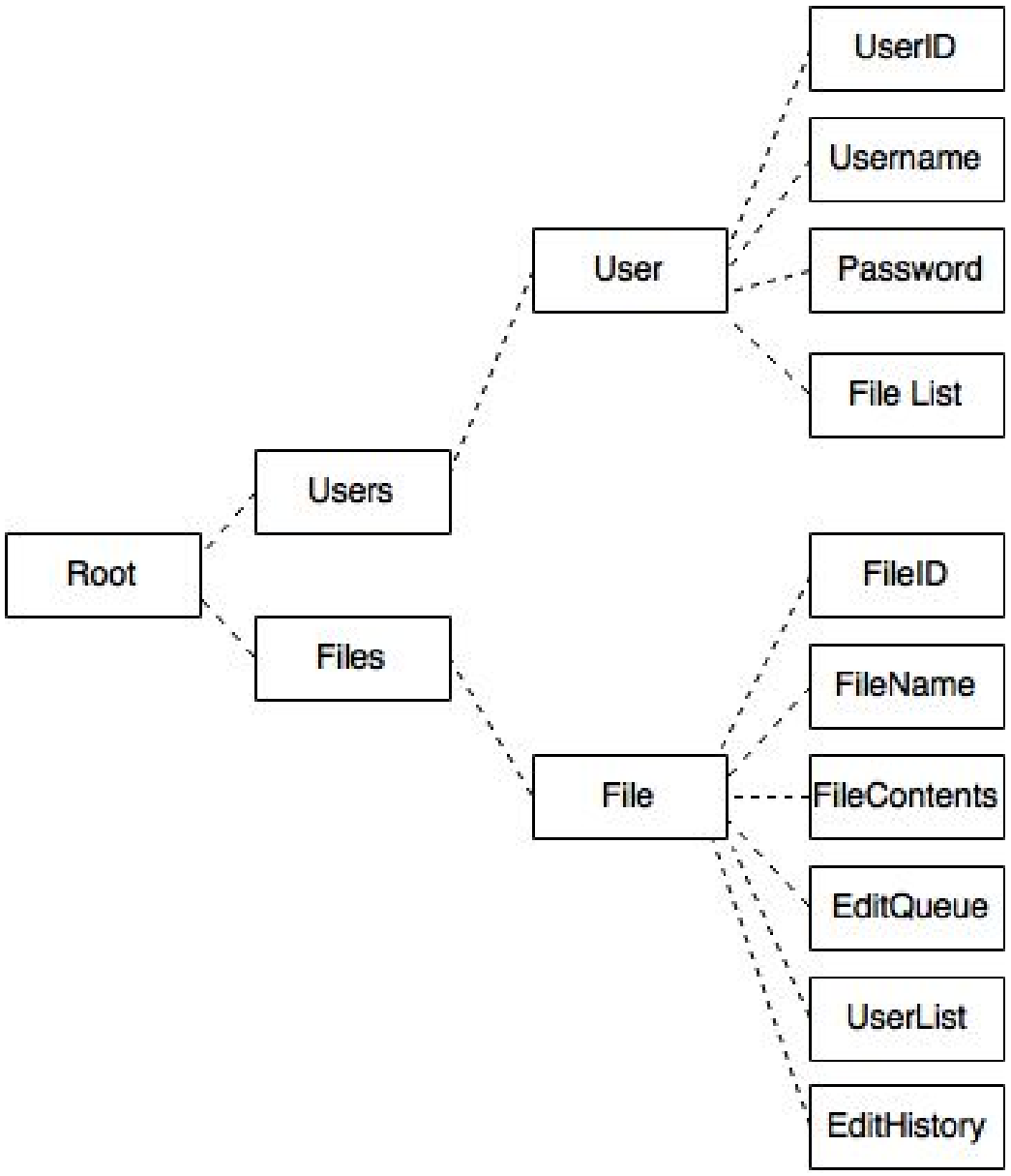
Design Details

Data Class Level-Design



Database Design

Google's Firebase Realtime Database is structured using a JSON tree. Data added to the database is added as a JSON node identified with a unique key.



Classes and Interactions Between Classes

User:

- Represents a user with editing capabilities
- Contains information about the user like name, login details such as username and password, a unique user-id and a list of files which the user has access to.
- Provides capabilities for the user to create or edit a file.

File:

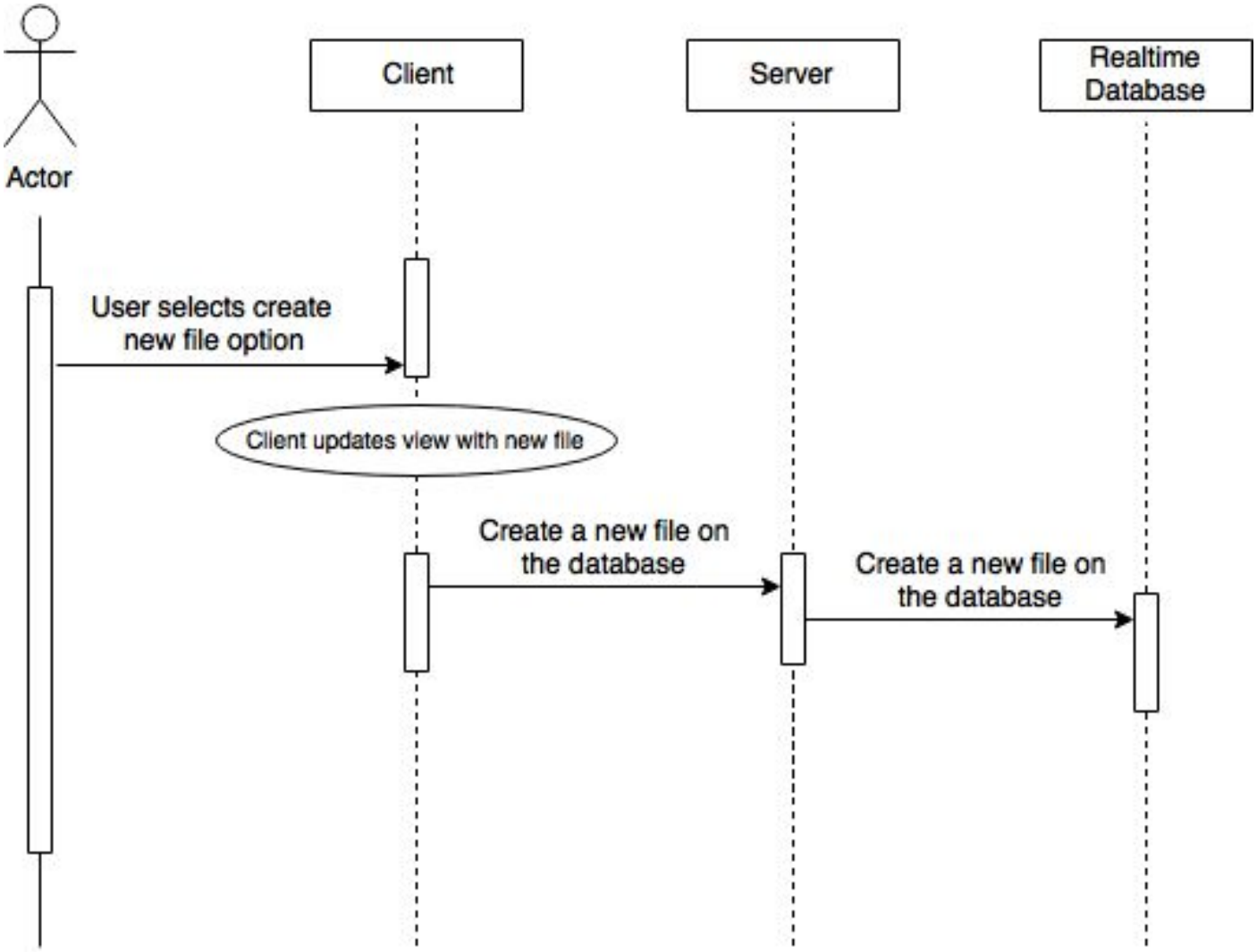
- Represents a version of a given document.
- Has a singular base file with only saved/committed changes.
- Stores all current tracked changes as a queue until they are committed.
- Contains a list of user-ids that have permission to access the file
- Backlogs all previous staged file versions to create a viewable revision history

Edit:

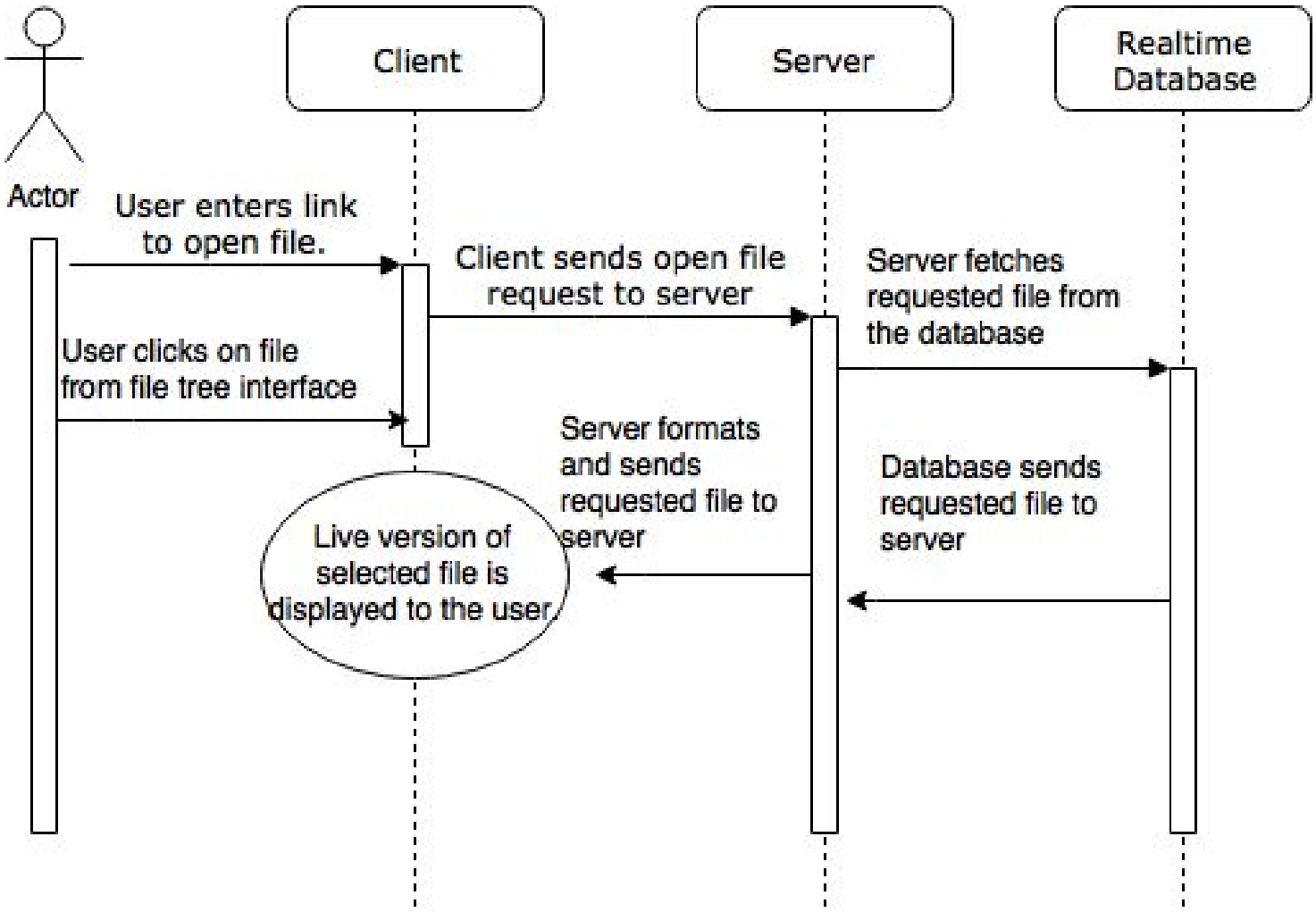
- Represents a change made to an file.
- Contains the contents of the edit, who made the edit, and when the edit was made.
- Has a variable that represents if this edit has been approved or not.

Sequence Diagrams

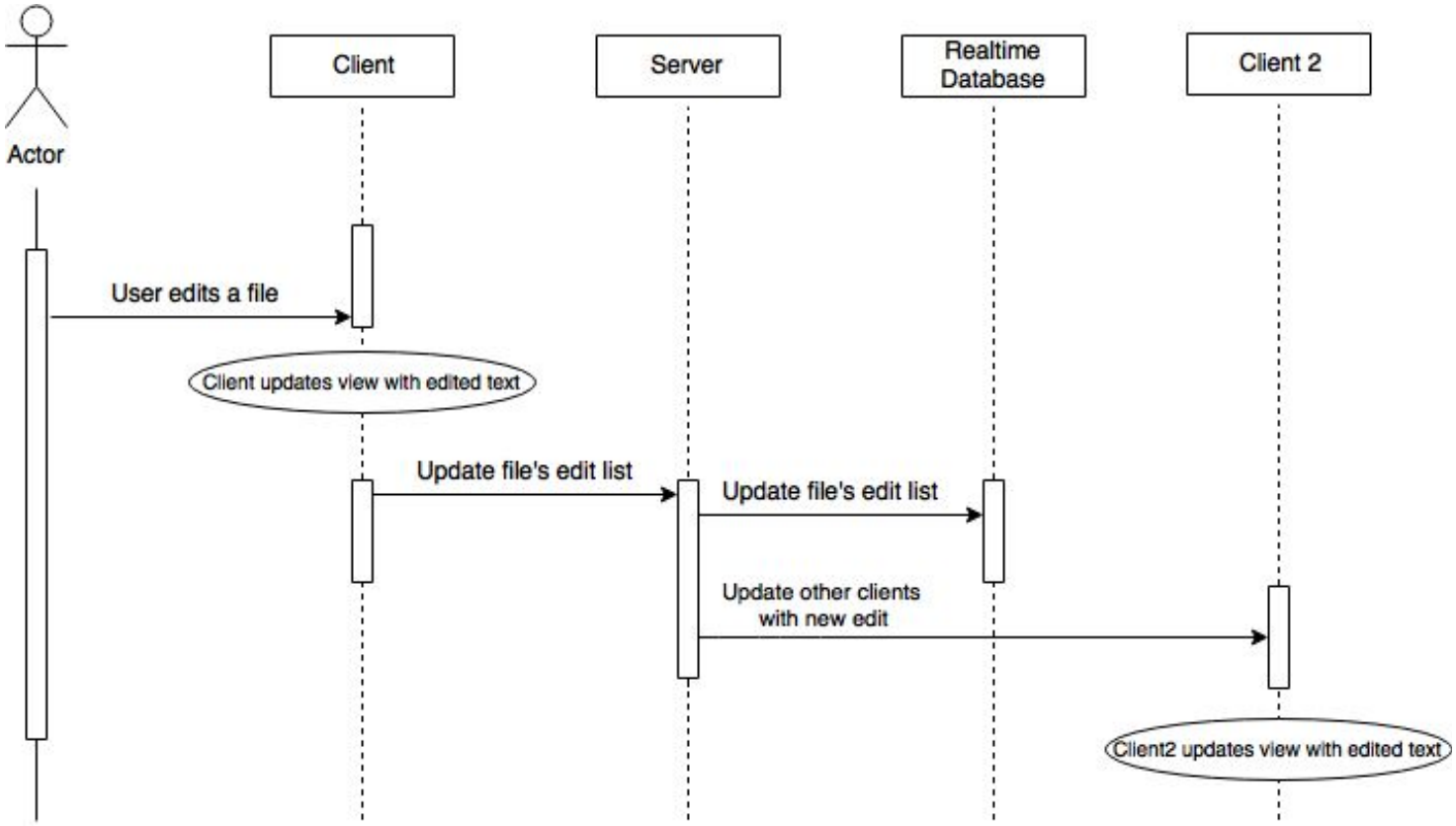
User creates a file



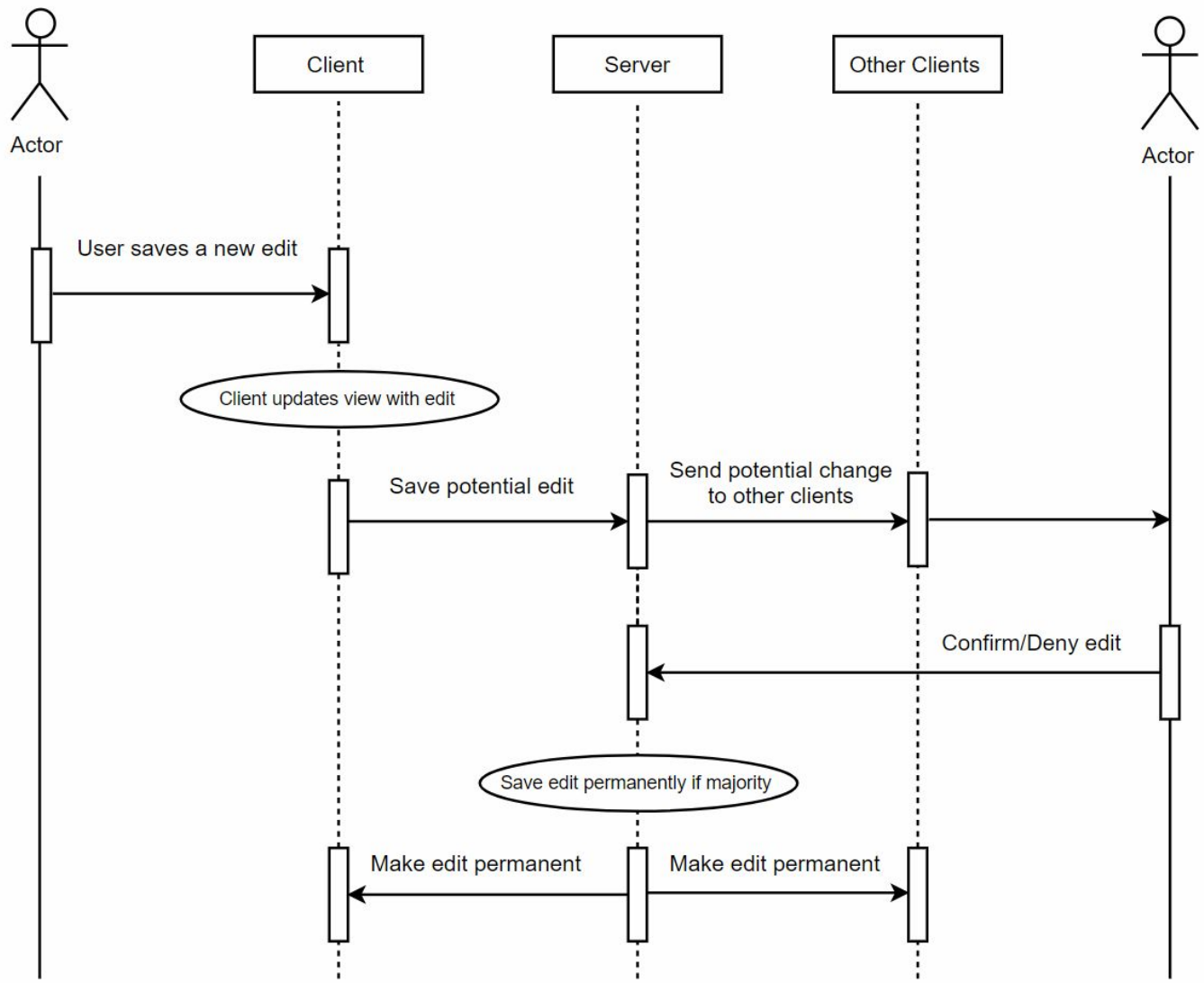
User opens a file



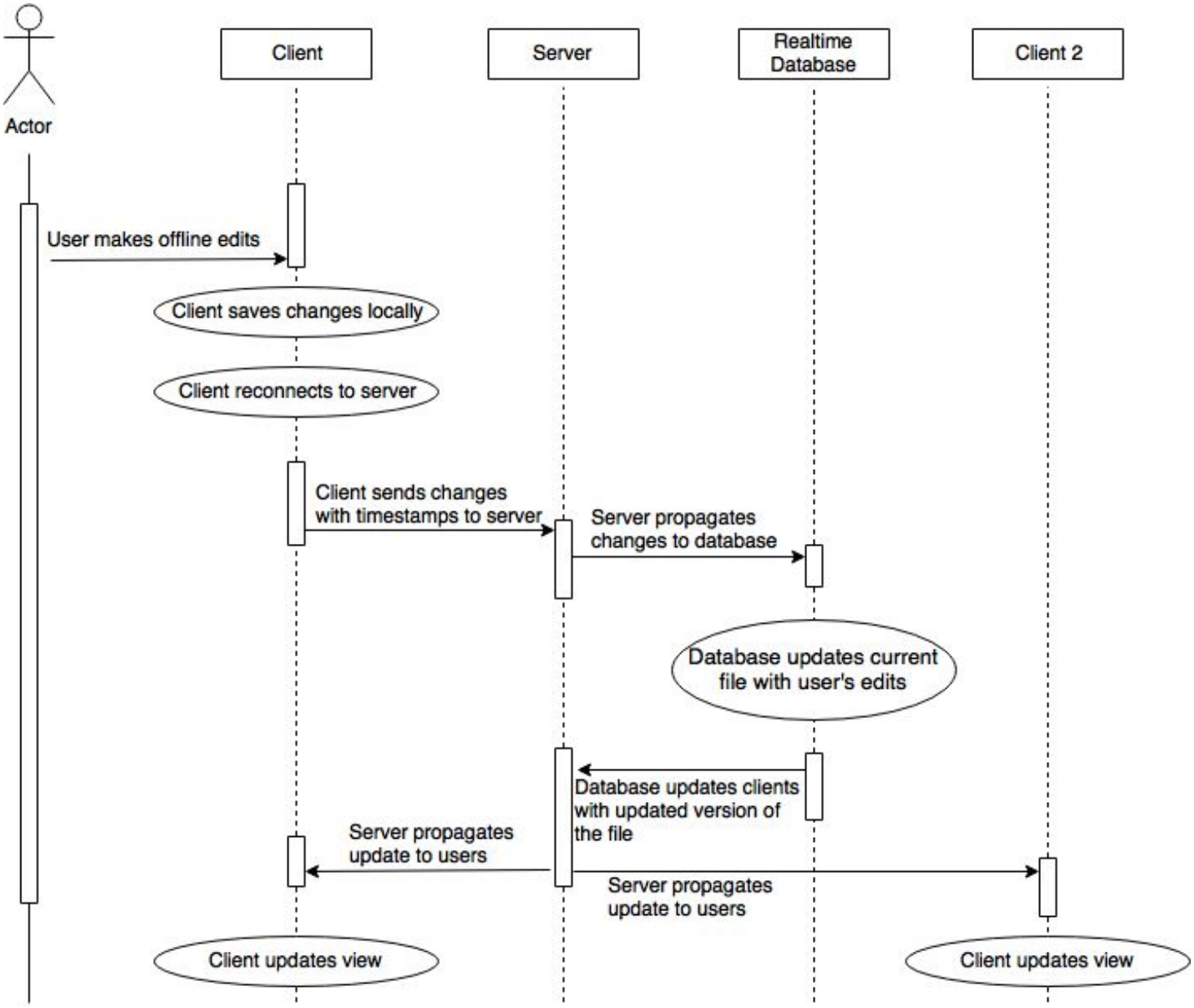
User edits a file



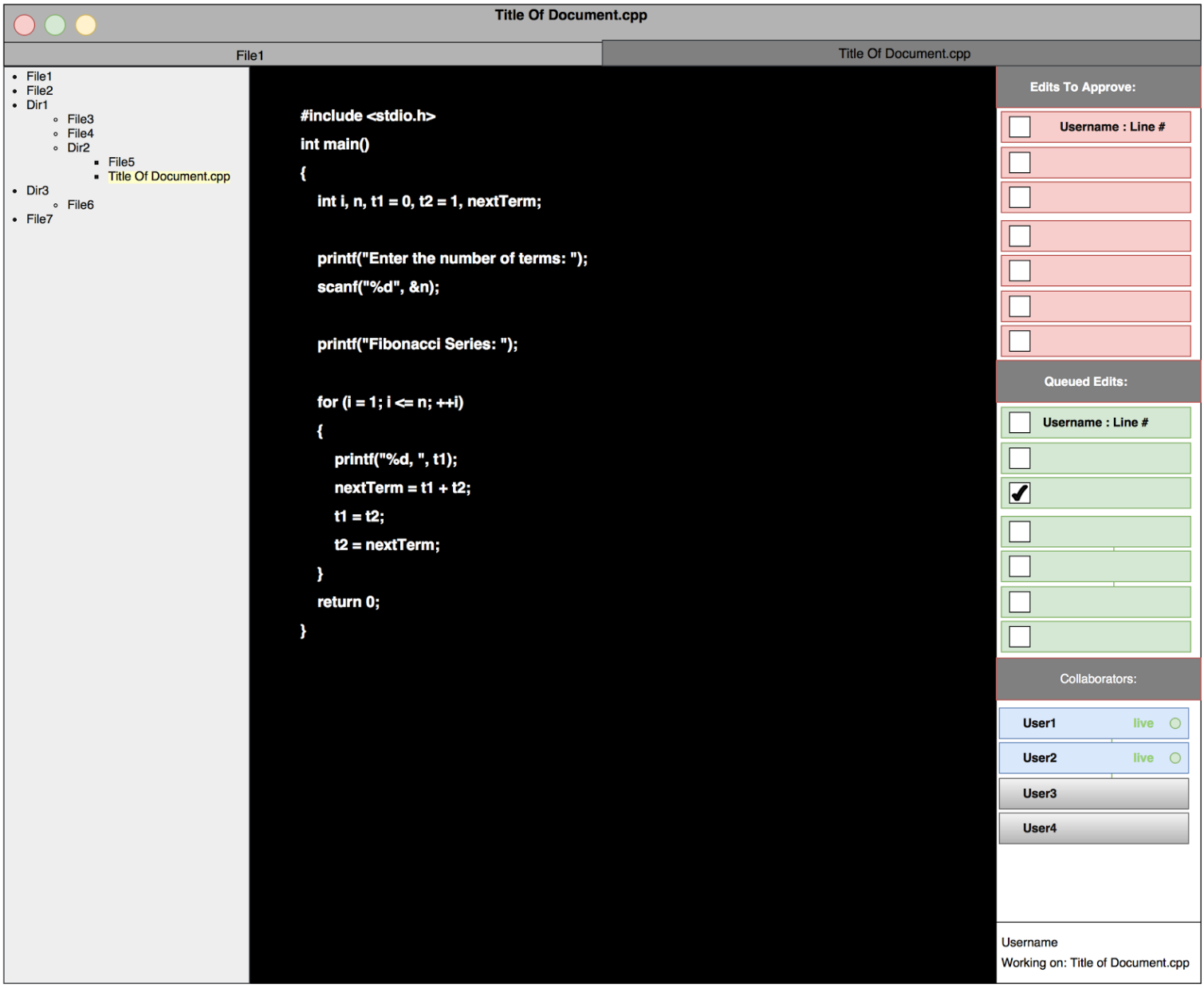
Group commit



Making offline edits and reconnecting to the server



UI Mockup


Title Of Document.cpp
File1
Title Of Document.cpp
File1
File2
Dir1
File3
File4
Dir2
File5
Title Of Document.cpp
Dir3
File6
File7
#include <stdio.h>
int main()
{
int i, n, t1 = 0, t2 = 1, nextTerm;
printf("Enter the number of terms: ");
scanf("%d", &n);
printf("Fibonacci Series: ");
for (i = 1; i <= n; ++i)
{
printf("%d, ", t1);
nextTerm = t1 + t2;
t1 = t2;
t2 = nextTerm;
}
return 0;
}
Edits To Approve:
Username : Line #
Queued Edits:
Username : Line #
Collaborators:
User1
live
User2
live
User3
User4
Username
Working on: Title of Document.cpp