Assignment T1: Preliminary Project Proposal

**Part 1:**

* Team: Pyoneers
  + Andrew Tang, at3456
  + Benjamin Snyder, bs3148
  + Raymond Fang, rjf2157
  + Jordan Dreibelbis, jed2205
  + Bryce Natter, bdn2113
* Nothing has changed.

**Part 2:**

1. **What will our service do? What kind of functionality or features will it provide?**

Real-Time-editing-as-a-Service (see page 2 for an outline of functionality)

“GitHub Live”

Service that allows clients to edit code/text collaboratively in real time (like google docs). It makes revisions available during a period without requests.

Create documents, update the authorized users, update the document content, delete documents

Users can request a version history of the document, and revert to a previous version.

1. **Who or what will be its users? What might they use the functionality for?**

The intended users are the developers of various IDEs/text editors (like VSCode) or business tools like Microsoft Office or web-apps that need real-time multi-user editing. These users will use the functionality for developing real-time collaboration capabilities.

1. **What kind of data will your service create or accumulate? What will the data be used for?**

Our service will store UTF-8 (text). The service will create user data (i.e. username, password, id), statistics about the file, and offer the full document back to the users

# Provides a diff from local to remote revision

GET document/{documentID}

Auth ID: user auth info

current rev{string or null}: hash of revision you have locally

@returns

“diff patch” to apply to the local document to update it

# Deletes document

DELETE document/{documentID}

# Receives diffs from users to a document

POST document/{documentID}/updateDocument

Auth ID: user auth info

diff{JSON}: content of diff patch to put on the server

timestamp: a timestamp for the diff so we know the order to

apply them

# returns user details

GET user/{userID}

@returns the user’s details

# creates a user

POST user/{userID}/create/

one of:

email and authID

user and authID

anonymousID and documentID

# returns confirmation of authentication

GET auth/document/{docID}/user/{userID}

qs above

@returns auth string

# returns a document ID for the newly created document

POST document/create

name: name of document

users/auth ids allowed?

@returns documentID

# updates the authenticated users for a doc

PATCH document/{documentID}/updateAuth

authID list: adds auth ids to the doc

# replaces the current remote revision

PUT document/{docID}/revision/{revision ID}

# Squash revisions for past X time

POST document/{docID}/revision/{revisionID}/create

revisionID & name\*

**Part 3:**

For no client testing we will use a lot of unit testing, since this is entirely testing business logic. Below we have expanded on ideas for our full testing suite, including integration testing and manual testing that would require mock clients.

1. **How will you test that your service does what it is supposed to do and provides the intended functionality?** 
   1. unit testing (use valid inputs)
      1. We will write a minimum of three unit tests for each parameter of every method, covering typical, atypical, and invalid user inputs.
   2. integration testing
      1. ensuring that a script that uses the services together could run from end to end (creating a document and users and entering input and then checking the result against expected document results that were manually generated)
   3. manual testing -- literally “testing by hand”
      1. manually checking that the correct data is returned when we submit correct inputs, if hooked up to a mock gui, or tested with postman.
2. **How will you check that your service does not behave badly if its clients use it in unintended ways or provide invalid inputs?** 
   1. unit testing (use invalid inputs)
      1. null, empty inputs -- eg null diff
      2. invalid inputs -- (weird characters, numbers instead of strings, etc)
      3. invalid auth -- unauthorized user attempt to edit
      4. invalid resource path -- user attempts to edit a document that does not exist
   2. integration testing -- ensures that it all works when put together
      1. see what happens when a user
   3. manual testing
      1. Postman
      2. Ask people who have never seen the code to blackbox test the service and try to break the service with as many invalid inputs as possible.
3. **How will you test that your service handles its data the way it's supposed to?** 
   1. unit testing
      1. For instance, we compare whether the document stored in the database is correct after the document is edited locally.
   2. We will also make sure we sanitize our database inputs. Any user inputs will not be executed (no functions such as eval(), for instance), preventing malicious code injections.