This is the set of design documents for Scribble.
This document details the design for the Scribble server.

1.1 Authentication and Authorization

Authentication is managed by the frontend via Passport.
2.1 Process

New revisions are stored in the database as follows:
1. When the user creates a revision, it is bzipped and stored in GridFS.
2. A corresponding entry is added to the Blobs collection, making note of its filename and offsets.
3. The revision is added to the Project tree.

Revisions are usually updated as follows:
1. The existing revision is retrieved from GridFS.
2. The new revision is concatenated onto the end of the retrieved data.
3. A corresponding entry is added to the Blobs collection, making note of its filename and offsets.
4. The revision is added to the Project tree.

If the concatenated uncompressed data exceeds a certain size threshold, a new GridFS file is created.

2.2 Rationale

bzip2 offers good compression on repetitive data (which is what revisions essentially are), and has good performance. It has better compression than diff chains as bzip2 is better at recognizing similarities between revisions and doesn’t enforce forward/backward diff chains. Theoretically, this scheme should offer better performance and compression than storing a full revision every $n$ changes.
This section documents the API calls and models for resources on the server.

A convention for fields is that if they are nullable or optional, their type will be suffixed with ?., and all array types are surrounded with 

### 3.1 Account

This resource is associated with user information and aspects such as authentication and authorization.

#### 3.1.1 User

A user.

**Model**

- `name :: String` A friendly display name.
- `emails :: [String]` Emails associated with the user for logging in.
- `pwhash :: String` Password hash.

**Endpoints**

**GET /account/user**

Look up an account.

**Possible query string parameters:**

- `name :: String?` Friendly name.

**Returns:**

An array of:

- `_id :: String` User ID.
- `name :: String` Friendly name.
GET /account/user/:id

Get account details by ID.

**Returns:**

* _id :: String* User ID.
* name :: String* Friendly name.

### 3.2 Workspace

This resource corresponds to collections of documents and textual data. Mongoose is too inconvenient to use with this.

#### 3.2.1 Project

A project, which manages documents.

**Model**

* name :: String* Name of the project.
* tree :: Tree* The representation of the project tree.

**authorizations :: { ObjectID : [String] }** Authorizations associated with the project. An authorization consists of a user and a list of permissions.

#### 3.2.2 Tree

A project tree.

**Example**

```json
{
  'foo' : [
    {
      'hash' : 'ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb',
      'timestamp' : 0
    },
    {
      'hash' : 'ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb',
      'timestamp' : 2
    }]
  },
  'baz' : {
    'bar' : [
      {
        'hash' : 'd6cbb053abf2933889a0ccbf6ac244623a63a2e3397e991dde09266bdaa932d1',
        'timestamp' : 0
      },
      ...
    ]
  }
}```
3.2.3 Revision

A revision in the tree.

Model

This model is a subdocument of Project.

**hash** :: **String** Blob hash. Set to **null** if file was deleted.

**timestamp** :: **Timestamp** Timestamp of the revision.

3.2.4 Blob

A single blob in the tree. Blobs are stored bzipped in GridFS. The offsets correspond to those in the uncompressed data.

Model

**hash** :: **String** SHA256 hash of data.

**filename** :: **String** GridFS file name.

**type** :: **String** Type of the blob.

**start** :: **Integer** Start offset of GridFS file.

**end** :: **Integer** End offset of GridFS file.
CHAPTER
FOUR

INDICES AND TABLES

• genindex
• search