CMPUT 404 Web Applications and Architecture

Project: Distributed Social Networking

Description

The web is fundamentally interconnected and peer to peer. There's no really great reason why we should all use facebook.com or google+ or myspace or something like that. If these social networks came up with an API you could probably link between them and use the social network you wanted. Furthermore you might gain some autonomy.

Thus in the spirit of diaspora https://diasporafoundation.org/ we want to build something like diaspora but far far simpler.

This blogging/social network platform will allow the importing of other sources of information (github) as well allow the distribution and sharing of posts and content.

An author sitting on one server can aggregate the posts of authors they follow on other servers.

We are going to go with an inbox model where by you share posts to your followers by sending them your posts. This is similar to activity pub: https://www.w3.org/TR/activitypub/ Activity Pub is great, but too complex for a class project.

We also won't be adding much in the way of encryption or security to this platform. We're keeping it simple and restful.

Choose at least 3 other groups to work with!

Scenario

I log into SocialDistribution. I see my stream which is filled with posts that have arrived in my inbox combined with public posts that my server knows about. I browse them and I click like on anything by my friend Steph who is on another node.

When I click like, my node sends a like object to Steph's inbox that references her post.

I then comment on Steph's post. This sends a comment object to Steph's inbox that references her post.

Steph's node will process events at her inbox and record the comments and likes appropriately.

Then I write a public post, a public service announcement (PSA) about how public service announcements are pretentious performative social media and you shouldn't make them. The irony is lost on me. I make an unlisted image post that contains an image for the PSA and reference it from my PSA post. Nonetheless my node records my post and makes a URL for both posts and then proceeds to send my public posts to the inboxes of everyone who follows me. My node knows who follows me and thus can just send the public post to each of those inboxes. Perhaps there will be a scaling problem in the future.

Later I write a friends-only post about how much I hate the movie The Room (2003). Tommy Wiseau can't see it because I'm not friends with Tommy Wiseau, but this post is sent to Steph's inbox and she can see it because she's my friend.

When Steph logs into her node she'll see her stream and my public post and my friends-only post will be on her stream. She should also see that I've liked and commented her post.

Scenario Summary

All actions from authors are routed through the inbox of the receiving authors by the node itself.

Nodes will store copies of posts because they receive them in the inbox.

Likes and comments on posts are all sent to the inbox of the author.

Public posts are sent to the inboxes of all followers of the author.

Friends-only posts are sent to the inboxes of all friends of the author.

Posts, likes, comments, posts, are all sent to the inboxes of the authors that should be able to see them.

Project Parts

- Part 0 - sign up a repo
- Part 1 - 1/2 way implementation
- Part 2 - Complete implementation
- Part 3 - Connect with Groups
- Part 4 - Finish it off!

Collaboration

- You may consult with others but the submission should be your own team's source code.
- You may work with 5 other students (groups of 5)
- Work will be submitted together
- Collaboration must be documented in the README.md file
- Any external source code must be referenced and documented in the README.md file.
- You make collaborate/consult with other groups in order to get your webservices to integrate with each other

User Stories

- Identity
    - As an author, I want a consistent identity per server, so that URLs to me/my posts are predictable and don't stop working.
        - Note: This includces API links and web frontend links.
        - Note: This doesn't include authors/posts that are deleted.
    - As a server admin, I want to host multiple authors on my server, so I can have a friendly online community.
    - As an author, I want a public page with my profile information, so that I can link people to it.
    - As an author, I want to my (new, public) github activity to be automatically turned into public posts, so everyone can see my github activity too.
    - As an author, I want my profile page to show my public posts (most recent first), so they can decide if they want to follow me.
    - As an author, I want to be able to use my web-browser to manage my profile, so I don't have to use a clunky API.
- Posting
    - As an author, I want to make public posts, so that any other author can see them.
    - As an author, I want to edit my posts locally, so that I'm not stuck with a typo on a popular post.
    - As an author, I want my edits to take effect remotely, so that people don't keep seeing the old version.
    - As an author, posts I make can be in CommonMark, so I can give my posts some basic formatting.
    - As an author, posts I make can be in simple plain text, because I don't always want all the formatting features of CommonMark.
    - As an author, posts I create can be images, so that I can share pictures and drawings.
    - As an author, posts I create that are in CommonMark can link to images, so that I can illustrate my posts.
    - As a server admin, images can be hosted on my server, so that my users can use them in their CommonMark posts.
    - As an author, I want to delete my own posts locally, so I can remove posts that are out of date or made by mistake.
    - DELETED ~~As an author, I want my deletions to take effect remotely, so I know remote users don't keep seeing my deleted posts forever.~~
    - As an author, I want to be able to use my web-browser to manage/author my posts, so I don't have to use a clunky API.
    - As an author, I want to be able to make posts that are unlisted, that are publicly shareable by URI alone (or for embedding images).
- Reading
    - As an author, I want a "stream" page which shows me all the public posts my server knows about and all the posts by people I follow.
    - As an author, I want my "stream" page to be sorted with the most recent posts first.
- Visibility

    - As an author, posts I create can be friends-only, so that I don't have to worry about people I don't know seeing them.
    - As an author, I want my friends-only posts and images to only be visible to my friends, so I can feel safe about posting.
        - Note: public posts (and image posts) are public.
    - As an author, other authors cannot modify my posts, so that I don't get impersonated.
    - As an author, posts I create should always be visible to me, so I can find them to edit them or review them or get the link or whatever I want to do with them.
- Sharing
    - As an author, I can share other author's public posts, so I can make things go viral!
    - As an author, posts that I share will show up on the timeline of anyone who is following me.
    - As a server admin, I want to share public images with users on other servers, so that they are visible by users of other servers.
    - As an author, I want my friends-only/unlisted images and posts sto *not* be shareable, so I know that if someone wants to re-post it they'll at least have to take a screenshot.
        - Note: public posts (and image posts) are re-shareable.
    - As an author, I should be able to browse the public posts of everyone, so that I can see what's going on beyond authors I follow.
- Following/Friends
    - As an author, I want to follow local authors, so that I can see their public posts.
    - As an author, I want to follow remote authors, so that I can see their public posts.
    - As an author, I want to be able to approve or deny other authors following me, so that I don't get followed by people I don't like.
    - As an author, I want to know if I have follow requests, so I can approve them.
    - As an author, I want to unfollow authors I am following, so that I don't have to see their posts anymore.
    - As an author, if I am following a local author and they are following me, I want us to be considered friends, so that they can see my friends-only posts.
    - As an author, I want to unfriend remote authors by unfollowing them, so that they can no longer see my friends-only posts.
    - As an author, my server will know about my followers, who I am following, and my friends, so that I don't have to keep track of it myself.
- Comments/Likes
    - As an author, I want to comment on posts that I can access, so I can make a witty reply.
    - As an author, I want to like posts that I can access, so I can show my appreciation.
    - As an author, when someone sends me a public post I want to see the likes, so I can tell if its good or not.
    - As an author, comments on my friends-only posts are visible only to my friends and the comment's author.
- Server Management
    - As a server admin, I want to be able add, modify, and delete authors, to fix problems or remove unwanted users.
    - As a server admin, I want to OPTIONALLY be able allow users to sign up but require my OK to finally be on my server, so that I can prevent unwanted users or spam bots.
    - As a server admin, I want to be able to connect to remote servers by entering only the URL of the remote server, a username, and a password, so that I don't have to edit code.
    - As a server admin, I want a RESTful interface for most operations, so that I can connect to other servers and allow my users to use alternate clients other than the web frontend.
    - As a server admin, I want to be able to add nodes to share with.
    - As a server admin, I want to be able to remove nodes and stop sharing with them.
    - As a server admin, I can prevent nodes from connecting to my node if they don't have a valid username and password.
    - As a server admin, node to node connections can be authenticated with HTTP Basic Auth, so that I don't have to deal with tokens.
    - As a server admin, I can disable the node to node interfaces for connections that I no longer want, in case another node goes bad.

    - As a server admin, I want everything to be stored in a well-indexed relational database, so that my website is snappy and I can write SQL to fix things if I need to, make backups, etc...
        - We strongly recommend Postgres on Heroku and sqlite for testing on your local machine.
        - We strongly recommend against other DBaaS (e.g. Firebase), since this always seems to cause problems for teams who chose them.
    - As a server admin, I don't want arrays to be stored in database fields, so that my server won't get slower over time.
    - As a server admin, I don't want to have seperate frontend and backend web servers, so I don't have to manage two webservers/services.

Main Concepts

- Author
    - makes posts
    - following other authors
    - can have followers
    - makes friends
    - likes posts
    - comments on posts
    - a generally nice person
- Server Admin
    - manages a node
    - allows people to sign up
    - responsible for private data :(
- Follower
    - Someone who follows you
    - Your server will send posts to their inbox
- Following
    - Someone you follow
    - Their server will send posts to your inbox
- Friend
    - You follow them and they follow you
- Server
    - a host that hosts authors and vouches for them
- Restful service
    - The model of the service and its API
- UI
    - The HTML/CSS/JS coated version user interface
- Public Post
    - this is a post that will show up publicly.
    - it has a public URL
    - anyone can see it
    - Public posts can be liked
    - public posts can have comments
- Friends-Only Post
    - this is a post that is shared to friends (followers)
    - since it is sent, it is a message and not changeable
    - Friends-Only posts can be liked
    - Friends-Only posts can have comments sent back to the author via the author's inbox
- Inbox
    - This is what a READER or USER of the social network has. They follow authors, and the authors they follow send objects to their inbox.
    - This is something that only exists in the API. There is no special inbox in the User Interface, posts sent to a user's inbox are integrated into a their stream.
    - This forms the backbone of the timeline of the social media user.
    - This receives likes and comments.
- Stream
    - The user interface showing date-sorted (most recent first) local, public posts combined with posts from everyone that user follows.
- Remote

    - A node to node connection. Requests from another node. HTTP Basic Auth authenticated.
- Local
    - A local user accessing the REST API. Likely will use their cookie-auth, basic auth, or token auth. Local usually implies you check whether or not the user should have access. For instance local API access to the inbox should be limited to only that authenticated authors---don't snoop!
- Profile Page
    - A page that shows information about me as well as my public posts.

Authentication

- Remote node authentication is global between nodes. It requires an admin to allow node to node access.
- Remote node authentication is done solely with HTTP basic auth
- Local auth uses your local auth mechanisms (cookies, tokens, basic auth).
- Local refers the RESTful API for authors on that server.
    - This is useful for frameworks like react, vue, or angular to get access to data, and enable a more client heavy UI.
    - Even if your frontend does not use them, endpoints marked [local] should be usable by a future Android/iOS client to achieve the same functionality as frontend.

Pagination

If something is paginated it has query options:

- page - how many pages of objects have been delivered
- size - how big is a page
- Page 4 of objects http://service/author/{author_id}/posts/{post_id}/comments?page=4
- Page 4 of objects but 40 per page http://service/author/{author_id}/posts/{post_id}/comments?page=4&size=40
- 1 based indexing. First page is 1.

Communication

- HTTP Methods (PUT POST GET DELETE) not explicitly listed are not allowed methods
    - Most HTTP methods are local only, and provided for local node use.
    - Local node use means for use by the frontend to communicate with the backend it came from.
- You are free to add your own endpoints for local node use, but you must document and test them.

Who talks to Who

- Required: The backends talk to each other, mostly trough POST requests to the inbox URLs for remote authors.
- Optional: The frontend for a single node talks to the backend for a single node.
- Forbidden: The frontend talks to a backend of another node.

Overview

Almost all server-to-server communication proceeds by the server where something (post/like/comment/follow request) was created POSTing that thing that was created to the relevant authors inbox on a remote node.

Server-to-server (marked as "[remote]") request other than "POST to inbox" are rarely needed, but you should support them in case the remote server needs to check something.

Example (Server-to-Server API View)

1. I sign up for SocialDistribution on http://node1
2. node1 administrator approves my account.
3. Now an author object exists that represents me at http://node1/api/authors/555555555
4. I use the interface to make a follow request to Steph, who is node2/api/authors/777777777

5. My server POSTs the follow request to Steph's inbox at http://node2/api/authors/777777777/inbox
6. When Steph logs in, she sees my follow request and approves it.
7. Now Steph's server (node2) knows that I am following her.
8. Steph makes a public post.
9. Steph's server (node2) sends Steph's new post to my inbox with POST http://node1/api/authors/555555555/inbox.
10. I eventually see Steph's new post, and click like on it.
11. My server sends the like to Steph's inbox with POST http://node2/api/authors/777777777/inbox

The Frontend-to-Backend (also known as [local]) communication for this scenario is up to your team, this only describes the Backend-to-Backend communication between two different nodes.

API Endpoints

Authors

- URL: ://service/authors/
  - GET [local, remote]: retrieve all profiles on the server (paginated)
    - page: how many pages
    - size: how big is a page
- Example query: GET ://service/authors?page=10&size=5
  - Gets the 5 authors, authors 45 to 49.
- Example: GET ://service/authors/

```
{
    "type": "authors",
    "items":[
        {
            "type":"author",
            "id":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
            "url":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
            "host":"http://127.0.0.1:5454/",
            "displayName":"Greg Johnson",
            "github": "http://github.com/gjohnson",
            "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
        },
        {
            "type":"author",

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
            "host":"http://127.0.0.1:5454/",
            "displayName":"Lara Croft",

"url":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
            "github": "http://github.com/laracroft",
            "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
        }
    ]
}
```

Single Author

- URL: ://service/authors/{AUTHOR_ID}/
  - GET [local, remote]: retrieve AUTHOR_ID's profile
  - PUT [local]: update AUTHOR_ID's profile
- Example Format:

```
{
    "type":"author",
    // ID of the Author
    "id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
    // the home host of the author
    "host":"http://127.0.0.1:5454/",
    // the display name of the author
    "displayName":"Lara Croft",
    // url to the authors profile
    "url":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
    // HATEOS url for Github API
    "github": "http://github.com/laracroft",
    // Image from a public domain
    "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
}
```

Followers

- URL: ://service/authors/{AUTHOR_ID}/followers
    - GET [local, remote]: get a list of authors who are AUTHOR_ID's followers
- URL: ://service/authors/{AUTHOR_ID}/followers/{FOREIGN_AUTHOR_ID}
    - Note: foreign author ID should be a percent encoded URL of the foreign author. An example URL would be:
        - `http://example-node-1/authors/178aba49-ca39-4741-b227-f40d072b1222/followers/http%3A%2F%2Fexample-node-2%2Fauthors%2F5f57808f-0bc9-4b3d-bdd1-bb07c976d12d`
    - DELETE [local]: remove FOREIGN_AUTHOR_ID as a follower of AUTHOR_ID
    - PUT [local]: Add FOREIGN_AUTHOR_ID as a follower of AUTHOR_ID (must be authenticated)
    - GET [local, remote] check if FOREIGN_AUTHOR_ID is a follower of AUTHOR_ID
        - Should return 404 if they're not
        - Should return similar format to Follow Request below if they are.
- Example: GET ://service/authors/{AUTHOR_ID}/followers

```
{
    "type": "followers",
    "items":[
        {
            "type":"author",
            "id":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
            "url":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
            "host":"http://127.0.0.1:5454/",
            "displayName":"Greg Johnson",
            "github": "http://github.com/gjohnson",
            "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
        },
        {
            "type":"author",

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
            "host":"http://127.0.0.1:5454/",
            "displayName":"Lara Croft",

"url":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
            "github": "http://github.com/laracroft",
            "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
        }
    ]
}
```

Friend/Follow Request

- When author 1 tries to follow author 2, author 1's server send the follow request to author 2's server.
- If the author 2 accepts the Follow Request then author 1 is following author 2.
    - If author 2 is also already following author 1, then they are now friends.
- Sent to inbox of "object"
- Example format:

```
{
    "type": "Follow",
    "summary":"Greg wants to follow Lara",
    "actor":{
        "type":"author",
        "id":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
        "url":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
        "host":"http://127.0.0.1:5454/",
        "displayName":"Greg Johnson",
        "github": "http://github.com/gjohnson",
        "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
    },
    "object":{
        "type":"author",
        // ID of the Author
        "id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
        // the home host of the author
        "host":"http://127.0.0.1:5454/",
        // the display name of the author
        "displayName":"Lara Croft",
        // url to the authors profile
        "url":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
        // HATEOS url for Github API
        "github": "http://github.com/laracroft",
        // Image from a public domain
        "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
    }
}
```

Post

- URL: ://service/authors/{AUTHOR_ID}/posts/{POST_ID}
  - GET [local, remote] get the public post whose id is POST_ID
    - friends-only posts: must be authenticated
  - DELETE [local] remove the post whose id is POST_ID
    - local posts: must be authenticated locally as the author
  - PUT [local] update a post where its id is POST_ID
    - local posts: must be authenticated locally as the author
- Creation URL ://service/authors/{AUTHOR_ID}/posts/
  - GET [local, remote] get the recent posts from author AUTHOR_ID (paginated)
    - Not authenticated: only public posts.
    - Authenticated locally as author: all posts.
    - Authenticated locally as friend of author: public + friends-only posts.
    - Authenticated as remote server: This probably should not happen. Remember, the way remote server becomes aware of local posts is by local server pushing those posts to inbox, not by remote server pulling.
  - POST [local] create a new post but generate a new id
    - Authenticated locally as author
- Be aware that Posts can be images that need base64 decoding.
  - posts can also hyperlink to images that are public
- Example Format:

```
{
    "type":"post",
    // title of a post
    "title":"A post title about a post about web dev",
    // id of the post

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/764efa883dda1e11db4767
    // where did you get this post from?
    "source":"http://lastplaceigotthisfrom.com/posts/yyyyy",
    // where is it actually from
    "origin":"http://whereitcamefrom.com/posts/zzzzz",
    // a brief description of the post
    "description":"This post discusses stuff -- brief",
    // The content type of the post
    // assume either
    // text/markdown -- common mark
    // text/plain -- UTF-8
    // application/base64
    // image/png;base64 # this is an embedded png -- images are POSTS. So you might have
a user make 2 posts if a post includes an image!
    // image/jpeg;base64 # this is an embedded jpeg
    // for HTML you will want to strip tags before displaying
    "contentType":"text/plain",
    "content":"Þā wæs on burgum Bēowulf Scyldinga, lēof lēod-cyning, longe þrāge folcum
gefrǣge (fæder ellor hwearf, aldor of earde), oð þæt him eft onwōc hēah Healfdene; hēold
þenden lifde, gamol and gūð-rēow, glæde Scyldingas. Þǣm fēower bearn forð-gerīmed in
worold wōcun, weoroda rǣswan, Heorogār and Hrōðgār and Hālga til; hȳrde ic, þat Elan cwēn
Ongenþēowes wæs Heaðoscilfinges heals-gebedde. Þā wæs Hrōðgāre here-spēd gyfen, wīges
weorð-mynd, þæt him his wine-māgas georne hȳrdon, oð þæt sēo geogoð gewēox, mago-driht
micel. Him on mōd bearn, þæt heal-reced hātan wolde, medo-ærn micel men gewyrcean, þone
yldo bearn ǣfre gefrūnon, and þǣr on innan eall gedǣlan geongum and ealdum, swylc him god
sealde, būton folc-scare and feorum gumena. Þā ic wīde gefrægn weorc gebannan manigre
mǣgðe geond þisne middan-geard, folc-stede frætwan. Him on fyrste gelomp ǣdre mid yldum,
þæt hit wearð eal gearo, heal-ærna mǣst; scōp him Heort naman, sē þe his wordes geweald
wīde hæfde. Hē bēot ne ālēh, bēagas dǣlde, sinc æt symle. Sele hlīfade hēah and horn-
```

```
gēap: heaðo-wylma bād, lāðan līges; ne wæs hit lenge þā gēn þæt se ecg-hete āðum-swerian
85 æfter wæl-nīðe wæcnan scolde. Þā se ellen-gǣst earfoðlīce þrāge geþolode, sē þe in
þȳstrum bād, þæt hē dōgora gehwām drēam gehȳrde hlūdne in healle; þǣr wæs hearpan swēg,
swutol sang scopes. Sægde sē þe cūðe frum-sceaft fīra feorran reccan",
    // the author has an ID where by authors can be disambiguated
    "author":{
        "type":"author",
        // ID of the Author
        "id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
        // the home host of the author
        "host":"http://127.0.0.1:5454/",
        // the display name of the author
        "displayName":"Lara Croft",
        // url to the authors profile
        "url":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
        // HATEOS url for Github API
        "github": "http://github.com/laracroft",
        // Image from a public domain (optional, can be missing)
        "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
    },
    // comments about the post
    // return a maximum number of comments
    // total number of comments for this post
    "count": 1023,
    // the first page of comments

"comments":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/de305d54-
75b4-431b-adb2-eb6b9e546013/comments"
    // commentsSrc is OPTIONAL and can be missing
    // You should return ~ 5 comments per post.
    // should be sorted newest(first) to oldest(last)
    // this is to reduce API call counts
    "commentsSrc":{
        "type":"comments",
        "page":1,
        "size":5,

"post":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/de305d54-
75b4-431b-adb2-eb6b9e546013"

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/de305d54-
75b4-431b-adb2-eb6b9e546013/comments"
        "comments":[
            {
                "type":"comment",
                "author":{
                    "type":"author",
                    // ID of the Author (UUID)

"id":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
                    // url to the authors information

"url":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
                    "host":"http://127.0.0.1:5454/",
                    "displayName":"Greg Johnson",
                    // HATEOS url for Github API
                    "github": "http://github.com/gjohnson",
                    // Image from a public domain
                    "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
                },
                "comment":"Sick Olde English",
                "contentType":"text/markdown",
                // ISO 8601 TIMESTAMP
                "published":"2015-03-09T13:07:04+00:00",
                // ID of the Comment (UUID)

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/de305d54-
75b4-431b-adb2-eb6b9e546013/comments/f6255bb01c648fe967714d52a89e8e9c",
            }
        ]
    }
    // ISO 8601 TIMESTAMP
    "published":"2015-03-09T13:07:04+00:00",
    // visibility ["PUBLIC","FRIENDS","UNLISTED"]
    "visibility":"PUBLIC"
    // for visibility PUBLIC means it is open to the wild web
    // FRIENDS means if we're friends I can see the post
    // FRIENDS should've already been sent the post so they don't need this
}
```

Image Posts

Image Posts are just posts that are images. But they are encoded as base64 data. You can inline an image post using a data url or you can use this shortcut to get the image if authenticated to see it.

- URL: ://service/authors/{AUTHOR_ID}/posts/{POST_ID}/image
  - GET [local, remote] get the public post converted to binary as an iamge
  - return 404 if not an image
- This end point decodes image posts as images. This allows the use of image tags in markdown.
- You can use this to proxy or cache images.

Comments

- URL: ://service/authors/{AUTHOR_ID}/posts/{POST_ID}/comments
  - GET [local, remote] get the list of comments of the post whose id is POST_ID (paginated)
  - POST [local] if you post an object of "type":"comment", it will add your comment to the post whose id is POST_ID
- example comment from ://service/authors/{AUTHOR_ID}/posts/{POST_ID}/comments

```
{
    "type":"comment",
    "author":{
        "type":"author",
        // ID of the Author (UUID)
        "id":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
        // url to the authors information
        "url":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
        "host":"http://127.0.0.1:5454/",
        "displayName":"Greg Johnson",
        // HATEOS url for Github API
        "github": "http://github.com/gjohnson",
        // Image from a public domain
        "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
    }
    "comment":"Sick Olde English",
    "contentType":"text/markdown",
    // ISO 8601 TIMESTAMP
    "published":"2015-03-09T13:07:04+00:00",
    // ID of the Comment (UUID)

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/de305d54-
75b4-431b-adb2-eb6b9e546013/comments/f6255bb01c648fe967714d52a89e8e9c",
}
```

- example comments from a post:

```
{
    "type":"comments",
    "page":1,
    "size":5,

"post":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/de305d54-
75b4-431b-adb2-eb6b9e546013"

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/de305d54-
75b4-431b-adb2-eb6b9e546013/comments"
    "comments":[
        {
            "type":"comment",
            "author":{
                "type":"author",
                // ID of the Author (UUID)
                "id":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
                // url to the authors information
                "url":"http://127.0.0.1:5454/authors/1d698d25ff008f7538453c120f581471",
                "host":"http://127.0.0.1:5454/",
                "displayName":"Greg Johnson",
                // HATEOS url for Github API
                "github": "http://github.com/gjohnson",
                // Image from a public domain
                "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
            },
            "comment":"Sick Olde English",
            "contentType":"text/markdown",
            // ISO 8601 TIMESTAMP
            "published":"2015-03-09T13:07:04+00:00",
            // ID of the Comment (UUID)

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/de305d54-
75b4-431b-adb2-eb6b9e546013/comments/f6255bb01c648fe967714d52a89e8e9c",
        }
    ]
}
```

Likes

- You can like posts and comments
- Send them to the inbox of the author of the post or comment
- URL: ://service/authors/{AUTHOR_ID}/inbox
  - POST [local, remote]: send a like object to AUTHOR_ID
- URL: ://service/authors/{AUTHOR_ID}/posts/{POST_ID}/likes
  - GET [local, remote] a list of likes from other authors on AUTHOR_ID's post POST_ID
- URL: ://service/authors/{AUTHOR_ID}/posts/{POST_ID}/comments/{COMMENT_ID}/likes
  - GET [local, remote] a list of likes from other authors on AUTHOR_ID's post POST_ID comment COMMENT_ID
- Example like object:

```
{
    "summary": "Lara Croft Likes your post",
    "type": "Like",
    "author":{
        "type":"author",
        "id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
        "host":"http://127.0.0.1:5454/",
        "displayName":"Lara Croft",
        "url":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
        "github":"http://github.com/laracroft",
        "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
    },

"object":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/764efa883dda1e11db
}
```

Liked

- URL: ://service/authors/{AUTHOR_ID}/liked
  - GET [local, remote] list what public things AUTHOR_ID liked.
  - It's a list of of likes originating from this author
  - Note: be careful here private information could be disclosed.
- Example liked object:

```
{
    "type":"liked",
    "items":[
        {
            "summary": "Lara Croft Likes your post",
            "type": "Like",
            "author":{
                "type":"author",

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
                "host":"http://127.0.0.1:5454/",
                "displayName":"Lara Croft",

"url":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
                "github":"http://github.com/laracroft",
                "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
            },
"object":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/764efa883dda1e11db
        }
    ]
}
```

Inbox

- The inbox is all the new posts from who you follow, as well as follow requests, likes, and comments you should be aware of
- The inbox is the API equivalent of the stream in the UI
- URL: ://service/authors/{AUTHOR_ID}/inbox
    - GET [local]: if authenticated get a list of posts sent to AUTHOR_ID (paginated)
        - Should be sorted most recent first
    - POST [local, remote]: send a post to the author
    - if the type is "post" then add that post to AUTHOR_ID's inbox
    - if the type is "follow" then add that follow is added to AUTHOR_ID's inbox to approve later
    - if the type is "Like" then add that like to AUTHOR_ID's inbox
    - if the type is "comment" then add that comment to AUTHOR_ID's inbox
    - DELETE [local]: clear the inbox
- Example, retrieving an inbox:

```
{
    "type":"inbox",
    "author":"http://127.0.0.1:5454/authors/c1e3db8ccea4541a0f3d7e5c75feb3fb",
    "items":[
        {
            "type":"post",
            "title":"A Friendly post title about a post about web dev",

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/764efa883dda1e11db4767
            "source":"http://lastplaceigotthisfrom.com/authors/xxxxxx/posts/yyyyy",
            "origin":"http://whereitcamefrom.com/authors/yyyyyy/posts/zzzzz",
            "description":"This post discusses stuff -- brief",
            "contentType":"text/plain",
            "content":"Þā wæs on burgum Bēowulf Scyldinga, lēof lēod-cyning, longe þrāge
folcum gefrǣge (fæder ellor hwearf, aldor of earde), oð þæt him eft onwōc hēah Healfdene;
hēold þenden lifde, gamol and gūð-rēow, glæde Scyldingas. Þǣm fēower bearn forð-gerīmed
in worold wōcun, weoroda rǣswan, Heorogār and Hrōðgār and Hālga til; hȳrde ic, þat Elan
cwēn Ongenþēowes wæs Heaðoscilfinges heals-gebedde. Þā wæs Hrōðgāre here-spēd gyfen,
wīges weorð-mynd, þæt him his wine-māgas georne hȳrdon, oð þæt sēo geogoð gewēox, mago-
driht micel. Him on mōd bearn, þæt heal-reced hātan wolde, medo-ærn micel men gewyrcean,
þone yldo bearn ǣfre gefrūnon, and þǣr on innan eall gedǣlan geongum and ealdum, swylc
him god sealde, būton folc-scare and feorum gumena. Þā ic wīde gefrægn weorc gebannan
manigre mǣgðe geond þisne middan-geard, folc-stede frætwan. Him on fyrste gelomp ǣdre mid
yldum, þæt hit wearð eal gearo, heal-ærna mǣst; scōp him Heort naman, sē þe his wordes
geweald wīde hæfde. Hē bēot ne ālēh, bēagas dǣlde, sinc æt symle. Sele hlīfade hēah and
horn-gēap: heaðo-wylma bād, lāðan līges; ne wæs hit lenge þā gēn þæt se ecg-hete āðum-
swerian 85 æfter wæl-nīðe wæcnan scolde. Þā se ellen-gǣst earfoðlīce þrāge geþolode, sē
þe in þȳstrum bād, þæt hē dōgora gehwām drēam gehȳrde hlūdne in healle; þǣr wæs hearpan
swēg, swutol sang scopes. Sægde sē þe cūðe frum-sceaft fīra feorran reccan",
            "author":{
                "type":"author",

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
                "host":"http://127.0.0.1:5454/",
                "displayName":"Lara Croft",

"url":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
                "github": "http://github.com/laracroft",
                "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
            },

"comments":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/de305d54-
75b4-431b-adb2-eb6b9e546013/comments"
            "published":"2015-03-09T13:07:04+00:00",
            "visibility":"FRIENDS"
        },
        {
            "type":"post",
            "title":"DID YOU READ MY POST YET?",

"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/999999983dda1e11db4767
            "source":"http://lastplaceigotthisfrom.com/authors/xxxxxx/posts/yyyyy",
            "origin":"http://whereitcamefrom.com/authors/wwwwww/posts/zzzzz",
            "description":"Whatever",
            "contentType":"text/plain",
            "content":"Are you even reading my posts Arjun?",
            "author":{
                "type":"author",
"id":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
                "host":"http://127.0.0.1:5454/",
                "displayName":"Lara Croft",
"url":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e",
                "github": "http://github.com/laracroft",
                "profileImage": "https://i.imgur.com/k7XVwpB.jpeg"
            },
"comments":"http://127.0.0.1:5454/authors/9de17f29c12e8f97bcbbd34cc908f1baba40658e/posts/de305d54-
75b4-431b-adb2-eb6b9e546013/comments"
            "published":"2015-03-09T13:07:04+00:00",
            "visibility":"FRIENDS"
        }
    ]
}
```

Requirements

- WARNING: Check this over again
- Implement the webservice as described in the user stories
- Provide a webservice interface that is restful
- Provide a web UI interface that is usable
- Prove your project by connecting with at least 1 clone of your project.
- Prove your project by connecting with at least 2 other groups.
- Prove your project by connecting with at least 3 other groups.
- Make a video demo of your blog (desktop-recorder is ok)
    - Your video may not be a part of your presentation.
- Make a presentation about your blog
- Follow the guidelines in the project spec for URLs and services
- Allow users to accept or reject follow requests
- Images get the same protection that posts get as they are POSTS

API Requirements

When building your API, try to adhere to these rules for easy compatibility with other groups:

- REST API calls may be prefixed. ie. `http://service_address/api/authors/{AUTHOR_ID}/posts/`
- Document your service address, port, hostname, prefix(if used), and the username/password for HTTP Basic Auth in your README so that HTTP clients can connect to your API.
- You **must** be compatible with the API specification and examples listed above in this document.
    - You **may** need to add some additional things for compatibility with other groups due to varying interpretations.
    - You **may** add additional endpoints and JSON keys as long as you provide the ones listed above.

Test Requirements

The REST API must be fully tested. * Every API endpoint must be tested. * Every API functionality must be tested. * Every API method must be tested. * Every user story must be tested at the API level.

These would be system/acceptance tests if the project didn't have a frontend.

Testing the user stories at the API level should automatically give you almost all of the tests you need for every API endpoint, every API functionality, and every API method.

Not every user story has an API to test. For example, adding/removing servers to connect with usually does not have an API. For these functions, please test them in whatever way is most convienent for your project, and then verify them using the API. For example, you could write a test that adds a server to connect with by calling Python code or modifying the database directly, then use the API to check that the new server connection is working.

Front-end (Selenium, etc.) tests are not required. Code coverage (line coverage, statement coverage, branch coverage, MC/DC, etc.) is not required. Unit testing is not required.

Take-aways

- 1 Working Website
- 1 Github git repo
- 1 Presentation
  - May not include the video
- 1 OpenAPI ('Swagger') specification for your API
- 1 Video

Restrictions

- You must use a single git repository that contains everything.
- Must run on:
  - Heroku or Cybera VM
    - Other hosting on approval: you must get approval in advance from the instructor.
      - You are responsible for any problems that come up as a result.
      - You are responsible for any surprise fees that come up as a result.
      - Often when groups use some other hosting service that isn't Heroku they get shut off, blocked by firewalls, and/or hit with surprise fees.
  - Use Django or Flask (otherwise get approval from the instructor)
    - You must use a single django or a single flask to serve both frontend (for browser) and backend api, distinguish between the two using routes or django "apps" or whatever.
    - Use Python 3.8+ (otherwise get approval from the instructor)
  - Must use a postgres, sqlite, or mariadb database for storage.
    - For Heroku this must be postgres (sqlite doesn't work correctly on Heroku).
    - Database provider must be the same provider as your website hosting.
  - Splitting frontend/backend hosting is not allowed, this has caused too many problems.
- License your code properly (use an OSI approved license)
  - Put your name (or some representation of you like GeneralHuxFan768) on it!

Things that are allowed

- You may use React if you choose, but generally this is more difficult and takes more work so it is not recommended.
  - Django templates or lighter-weight frameworks like Vue are generally easier and take less work.
  - If you use React, you must still serve everything (including React js and html) through the same Django/Flask server that is serving the API backend.

Groupwork Tips

- These optional. They are just advice. They usually lead to less conflict and a better end result.

The most successful teams:

- Do not use pull requests.
  - Pull requests are great in general, but they tend to cause problems because the project is very small and groups are also small.
- Use very very few branches (e.g. production and staging).
  - Do not use author branches (or branches where a single author is the only one touching them).
- Commit, pull, push the code they're working on, to the same branch that everyone else is working on, at least once an hour.
  - Commit code that doesn't break the overall project.

    - Avoid your version being broken for more than an hour.
    - Do not stay up all night, change everything, and push a hundred changes all over the code base as one giant pull request.
- Divide teamwork by user story, instead of frontend vs backend.

Submission Instructions

- Submission will be by GitHub Classroom. Please follow the link on eClass.

Warning!!!!

This spec is subject to change!

Marking

- Usually all teammates will share the same mark.
- An individual student's marks will be reduced if the instructor (or a TA in consultation with the instructor) finds that:
    - The student is not contributing or contributing significantly less than their teammates.
    - The individual student is significantly preventing their teammates from participating or contributing.
    - The individual student's is significantly misusing git/github/heroku/other software tools.
    - The indivudual student is engaging in significant "intellectual violence."
        - Intellectual violence is when one teammate uses their skill, knowledge, or experience, to intimidate or control the other teammate(s) rather than sharing and helping them learn.
    - The individual student is not communicating or only communicating very rarely with their team.
    - The individual student does not complete the peer feedback form (-1 mark for each part 1-4).
- Any concerns about teamwork must be brought to the instructor's (not the TA) attention *by email* well before the last lab.
    - It takes time to investigate these things.

Project Part 0: Group Formation

- 1 mark
- Submitted on eClass by all members
- 1 Github repo with a README and LICENSE

Project Part 1: Centralized

Submission

Due 4PM on the day of your lab.

eClass has a limitation where it only shows the due date for the last lab section of the week, but for Monday labs it is due Monday. For Wednesday labs it is due Wednesday.

Create a git tag "part1" before 4PM and submit only the link to your tag.

- A tag is just a name for a commit, so do not create the tag until you have your final commit for part 1!

Don't forget to push the tag to GitHub.

Submit only the link to the tag in the following format:

https://github.com/uofa-cmput404/w24-project-example-team/tree/part1

Marking

- 7 Marks
- Total Project

    - Excellent 7: Excellent effort. Relatively consistent. At least ½ of the project implemented. Clean code. 1/2 includes both UI and webservice.
    - Good 6: Good quality. Some inconsistency. About ½ of the project implemented
    - Satisfactory 5: Codebase in places. Passes some tests. Some parts run
    - Unsatisfactory 3: Effort exists, it's missing lots of components but something is there.
    - Failure 0: Missing. No attempted. Not complete enough to evaluate.
- Code Base
    - Excellent : Excellent effort. Relatively consistent. At least ½ of the project implemented. Clean code
    - Good : Good quality. Some inconsistency. About ½ of the project implemented
    - Satisfactory : Codebase in places. Passes some tests. Some parts run
    - Unsatisfactory : Does not meet Satisfactory level
- Test Cases
    - Excellent: System is well tested
    - Good: System has some blind spots for testing
    - Satisfactory: Effort was placed on testing but it is inconsistent.
    - Unsatisfactory: test cases are inappropriate but exist.
    - Failure: Missing test cases
- UI 2
    - Excellent: UI Exists and is coherent. Shows evidence of planning.
    - Good: UI Exists. Some issues
    - Satisfactory: UI Exists, it's not good. It has issues.
    - Unsatisfactory: A UI was attempted, a UI exists.
    - Failure: No UI, or what was attempted is not substantial.
- Tool Use
    - Excellent: Use of at least Git is Evidence and Obvious
    - Good: Frequent but inconsistent use of Git, etc.
    - Satisfactory: Uses Git, etc.
    - Unsatisfactory: Limited of tool use
    - Failure: Used filesharing and email attachments instead of git
- TA Demo
    - Excellent: Coherent demo, shows off features. Limited snags.
    - Good: Coherent demo, shows off features. Some snags.
    - Satisfactory: Lots of snags. Can demo it.
    - Unsatifactory: Unfinished, hard to demo.
    - Failure: no demo or unable to demo.
- Web Service API & Documentation
    - Excellent: Documented, adheres to the specification & examples listed above where it exists. Open API specification exists, has clear descriptions, and has example requests and responses from your API.
    - Good: Documented, exists, tries to adhere to requirements. Open API specification exists, and has some descriptions and a few example requests and responses.
    - Satisfactory: Some of the webservice exists. Open API specification exists, but no descriptions or example requests and responses.
    - Unsatisfactory: Well you tried right? Open API specification does not exist.
    - Failure: Ok you didn't try.
- Design
    - Excellent: Adheres to standards, well designed
    - Good: Adheres to standards somewhat, some awkward parts
    - Satisfactory: Some good parts, some nasty parts
    - Unsatisfactory: Little effort went into documenting and designing the project
    - Failure: failure to learn from the class and apply concepts even remedially.

Project Part 2: Distribution

Requirements

- Everyone on the team must be able to deploy their own *instance* (copy) of the project to their own Heroku app with their own Heroku Postgres database. This doesn't need to be demonstrated, but it needs to be easy to do using what is in the project repo.
- At least two different team members must be able to demonstrate a fully working implementation, running on Heroku.

    - Both instances must be running the same code but have different servers & databases.
- At least two different team members must be able to demonstrate their instances coordinating. All user stories involving multiple authors must work in both situations:
    - All authors on the same instance
    - Authors on different instances
- For example if the user story is "As an author, I want to be able to approve or deny other authors following me, so that I don't get followed by people I don't like."
    - This needs to work when the author following is on a different instance from the author they are following.
    - It also needs to work if they're on the same instance.

Submission

Due 4PM on the day of your lab.

eClass has a limitation where it only shows the due date for the last lab section of the week, but for Monday labs it is due Monday. For Wednesday labs it is due Wednesday.

Create a git tag "part2" before 4PM and submit only the link to your tag.

Don't forget to push the tag to GitHub.

Submit only the link to the tag in the following format:

https://github.com/uofa-cmput404/w24-project-example-team/tree/part2

Marking

- 7 Marks
- Total Project
    - Excellent 7: Excellent effort. Relatively consistent. All of the project implemented. Clean code. Coordinates and connects fine with another instance of itself. API follows spec & examples listed in this document.
    - Good 6: Good quality. Some inconsistency. All of the project implemented except several user stories. Fixable and functional connection with another instance of itself.
    - Satisfactory 5: Codebase in places. Passes some tests. Most parts run. Some things coordinate and connect with another instance of itself.
    - Unsatisfactory 3: Effort exists, it's missing lots of components but at least ½ is there.
    - Failure 0: Missing. No attempted. Not complete enough to evaluate.
- Web Service Coordination
    - Excellent: Web service coordinates with another instance of itself successfully. Most interoperation requirements met.
    - Good: Web service coordinates with another instance of itself successfully. Most interoperation requirements met. Some snags.
    - Satisfactory: The basics of coordination with another instance of itself are covered. Probably many snags.
    - Unsatisfactory 0: Coordination with another instance of itself barely works.
    - Failure: Failure to coordinate and connect with another instance of itself.
- Code Base
    - Excellent : Excellent effort. Relatively consistent. All of the project implemented. Clean code.
    - Good : Good quality. Some inconsistency. All of the project implemented except several user stories.
    - Satisfactory : At least half of codebase in place. Passes some tests. Some parts run.
    - Unsatisfactory : Does not meet Satisfactory level.
- Test Cases
    - Excellent: System is well tested.
    - Good: System has some blind spots for testing.
    - Satisfactory: Effort was placed on testing but it is inconsistent.
    - Unsatisfactory: test cases are inappropriate but exist.
    - Failure: Missing test cases.
- UI

    - Excellent: UI Exists and is coherent. Shows evidence of planning.
    - Good: UI Exists. Some issues.
    - Satisfactory: UI Exists, it's not good. It has issues.
    - Unsatisfactory: A UI was attempted, a UI exists.
    - Failure: No UI, or what was attempted is not substantial.
- Tool Use
    - Excellent: Use of at least Git is Evidence and Obvious
    - Good: Frequent but inconsistent use of Git, etc.
    - Satisfactory: Uses Git, etc.
    - Unsatisfactory: Limited of tool use
    - Failure: Used filesharing and email attachments instead of git
- TA Demo
    - Excellent: Coherent demo, shows off features. Limited snags.
    - Good: Coherent demo, shows off features. Some snags.
    - Satisfactory: Lots of snags. Can demo it.
    - Unsatifactory: Unfinished, hard to demo.
    - Failure: no demo or unable to demo.
- Web Service API & Documentation
    - Excellent: Documented, adheres to the specification & examples listed above. Open API specification exists, has clear descriptions, and has example requests and responses from your API.
    - Good: Documented, exists, tries to adhere to requirements. Open API specification exists, and has some descriptions and a few example requests and responses.
    - Satisfactory: Some of the webservice exists. Open API specification exists, but no descriptions or example requests and responses.
    - Unsatisfactory: Well you tried right? Open API specification does not exist.
    - Failure: Ok you didn't try.
- Design
    - Excellent: Adheres to standards, well designed.
    - Good: Adheres to standards somewhat, some awkward parts.
    - Satisfactory: Some good parts, some nasty parts.
    - Unsatisfactory: Little effort went into documenting and designing the project.
    - Failure: Failure to learn from the class and apply concepts even remedially.

Project Part 3: Federation

Submission

Due 4PM on the day of your lab.

eClass has a limitation where it only shows the due date for the last lab section of the week, but for Monday labs it is due Monday. For Wednesday labs it is due Wednesday.

Create a git tag "part3" before 4PM and submit only the link to your tag.

Don't forget to push the tag to GitHub.

Submit only the link to the tag in the following format:

https://github.com/uofa-cmput404/w24-project-example-team/tree/part3

Marking

- 5 Marks
- Total Project
    - Excellent 5: Excellent effort. Coordinates and connects fine with 2 or more groups and another instance of itself. API follows spec & examples listed in this document.
    - Good 4: Some issues, not quite excellent but definitely fixable and functional with 1 or more groups and another instance itself. API follows spec & examples listed in this document.
    - Satisfactory 3: There are issues, it does run, it does coordinate with 1 or more groups.
    - Unsatisfactory 2: Not connected to other groups, still connects to another instance of itself.
    - Failure 0: Missing. No attempted. Not complete enough to evaluate.

- Web Service API & Documentation
    - Excellent: Documented, adheres to requirements to augments them with compatibility. Open API specification exists, has clear descriptions, and has example requests and responses from your API.
    - Good: Documented, exists, tries to adhere to requirements. Open API specification exists, and has some descriptions and a few example requests and responses.
    - Satisfactory: Some of the webservice exists. Open API specification exists, but no descriptions or example requests and responses.
    - Unsatisfactory: Webservice exists, barely. Open API specification does not exist.
    - Failure: it is not usable.
- Web Service Coordination
    - Excellent: Web service coordinates with 2+ other group projects and itself successfully. Most interoperation requirements met.
    - Good: Web service coordinates with 1+ other group projects and itself successfully. Most interoperation requirements met. Some snags.
    - Satisfactory: The basics of coordination are covered. Probably many snags.
    - Unsatisfactory 0: Coordination barely works.
    - Failure: failure to coordinate
- Design
    - Excellent: Adheres to standards, well designed.
    - Good: Adheres to standards somewhat, some awkward parts.
    - Satisfactory: Some good parts, some nasty parts.
    - Unsatisfactory: Little effort went into documenting and designing the project.
    - Failure: failure to apply what was learned in class.

Project Part 4: Polish

Submission

Due 4PM on the day of your lab.

eClass has a limitation where it only shows the due date for the last lab section of the week, but for Monday labs it is due Monday. For Wednesday labs it is due Wednesday.

Create a git tag "part4" before 4PM and submit only the link to your tag.

Don't forget to push the tag to GitHub.

Submit only the link to the tag in the following format:

https://github.com/uofa-cmput404/w24-project-example-team/tree/part4

Marking

- 10 Marks
- Total Project
    - Excellent 10: Excellent effort. Coordinates and connects fine. Good demo. Clear application of what was learned in class. 3 or more groups connected. Posts with embedded images are visible. Image posts are visible. API follows spec & examples listed in this document.
    - Good 8: Some issues, not quite excellent but definitely operational and functional. 2 or more groups connected. Posts with embedded images are visible. Image posts are visible. API follows spec & examples listed in this document.
    - Satisfactory 6: There are issues, it does run, it does coordinate. Meets satisfactory aspects of rubric. 2 or more group connected. Image posts are visible. API follows spec & examples listed in this document.
    - Unsatisfactory 4: Well you tried, but it's hardly working. Meets unsatisfactory aspects of rubric. 1 or more group connected.
    - Failure 0: Missing. No attempted. Not complete enough to evaluate. Often hits failure aspects of rubric.
- Note: these are ordered by importance, but you need to meet all these parts and we care about the final quality.
- Code Base
    - Excellent: Excellent effort. Relatively consistent. At least 90% of requirements implemented. Clean code

    - Good: Good quality. Some inconsistency. About 90% of requirements implemented.
    - Satisfactory: Codebase in places. Passes some tests. Some parts run.
    - Unsatisfactory: Does not meet Satisfactory level
- UI 3
    - Excellent: UI Exists and works well. Shows evidence of planning. Looks great.
    - Good: UI Exists. Looks good
    - Satisfactory: UI exists. Looks poor.
    - Unsatisfactory: UI exists. Doesn't work well. Worse than poor.
    - Failure: Missing or unusable.
- Web Service Coordination
    - Excellent: Web service coordinates with 2+ other group projects successfully. Most interoperation requirements met.
    - Good: Web service coordinates with 2+ other group projects successfully. Most interoperation requirements met. Some snags.
    - Satisfactory: The basics of coordination are covered. Probably many snags.
    - Unsatisfactory: Coordination doesn't work or barely works.
- Web Service API & Documentation
    - Excellent: Documented, adheres to requirements to augments them with compatibility. Open API specification exists, has clear descriptions, and has example requests and responses from your API.
    - Good: Documented, exists, tries to adhere to requirements. Open API specification exists, and has some descriptions and a few example requests and responses.
    - Satisfactory: Some of the webservice exists. Open API specification exists, and has a few example requests and responses.
    - Unsatisfactory: Effort taken but incomplete. Open API specification exists, but no descriptions or example requests and responses.
    - Failure: API or Documentation Missing. Open API specification does not exist.
- Test Cases
    - Excellent: System is well tested
    - Good: System has some tests
    - Unsatisfactory: test cases are inappropriate
    - Failure: Missing test cases
- Tool Use
    - Excellent: Use of at least Git is Evidence and Obvious
    - Good: Frequent but inconsistent use of Git, etc.
    - Satisfactory: Infrequent use of Git, etc.
    - Unsatisfactory: Limited tool use
    - Failure: lack of tool use
- Design
    - Excellent: Adheres to standards, well designed
    - Good: Adheres to standards somewhat, some awkward parts
    - Satisfactory: Some good parts, some nasty parts
    - Unsatisfactory: Little effort went into documenting and designing the project
    - Failure: clear lack of design
- Adhering to Standards
    - Excellent: Excellent attempt at making a standards compliant website. Most things are compliant.
    - Good: An attempt at making a standards compliant website. Some not compliant.
    - Satisfactory: Inconsistent.
    - Unsatisfactory: poor attempt to meet standards.
    - Failure: failed to apply what was learned in class
- Addressing Feedback:
    - Excellent: TAs suggestions were implemented, TA approves of implementation set.
    - Good: The good TA suggestions were implemented ;)
    - Satisfactory: Feedback ignored mostly, but some followed.
    - Unsatisfactory: Majority of Feedback ignored.
    - Failure: Feedback ignored.
- Presentation:
    - Excellent: Presentation within time, shows teamwork, promotes the application, uses the live application to show functionality.
    - Good: Presentation nearly within time, some team works, reasonable presentation.
    - Satisfactory: Presentation exists but has problems.

    - Unsatisfactory: Missing or terrible presentation (lack of practice, lack of preparation, irrelevant) OR presentation includes the video demo.
    - Failure: no presentation
- Video Demo:
    - Excellent: Video is well presented and not boring, less than 2 minutes. Posted to the discussion forum thread to share it with everyone in the class.
    - Good: Video presents the functionality and is less than 2 minutes.
    - Satisfactory: Video is longer than 2 minutes, or doesn't accurately present the project.
    - Unsatisfactory: A video exists and it is a demo.
    - Failure: lack of video, failure to make a video.
- AJAX
    - Excellent: Uses AJAX appropriately and well (documented)
    - Good: Uses some AJAX (documented)
    - Satisfactory: AJAX not really used
    - Unsatisfactory: An attempt was made.
    - Failure: No AJAX

License

Published: Sat, 06 Jan 2024 at 00:00 MST

By Hazel Victoria Campbell

Category: general

Tags: project grading