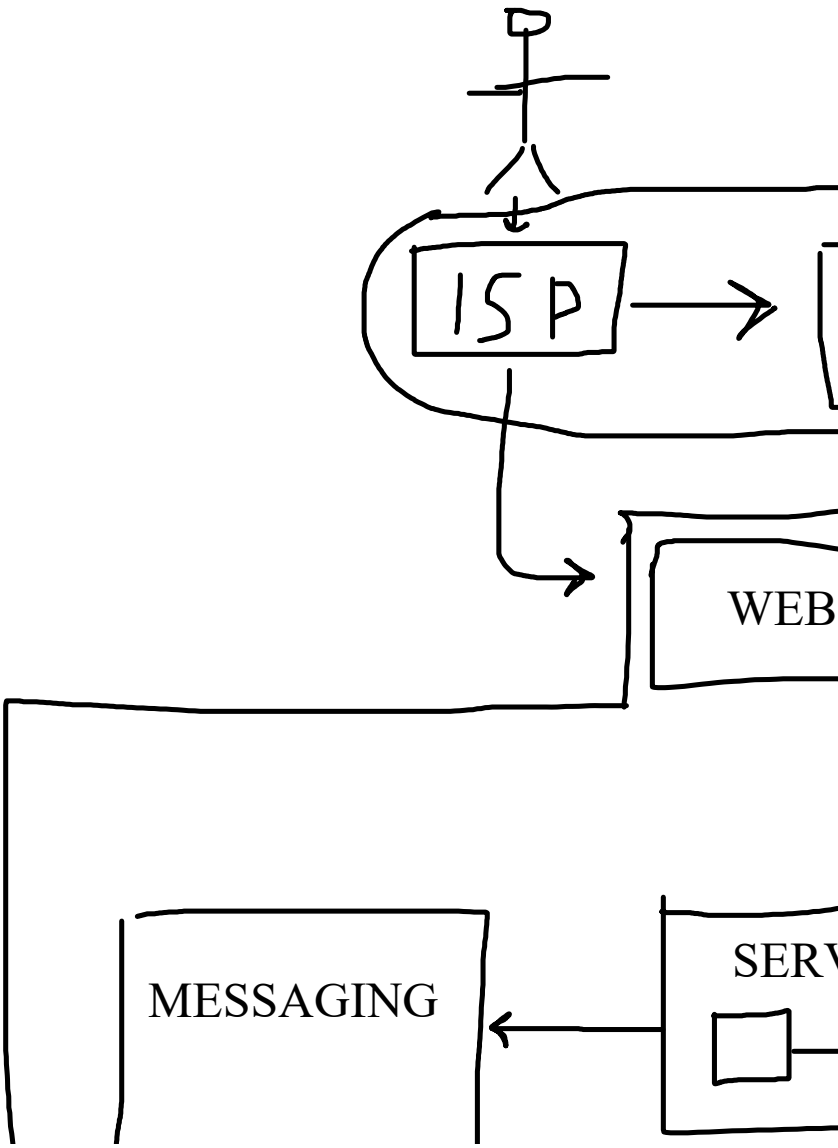
final project

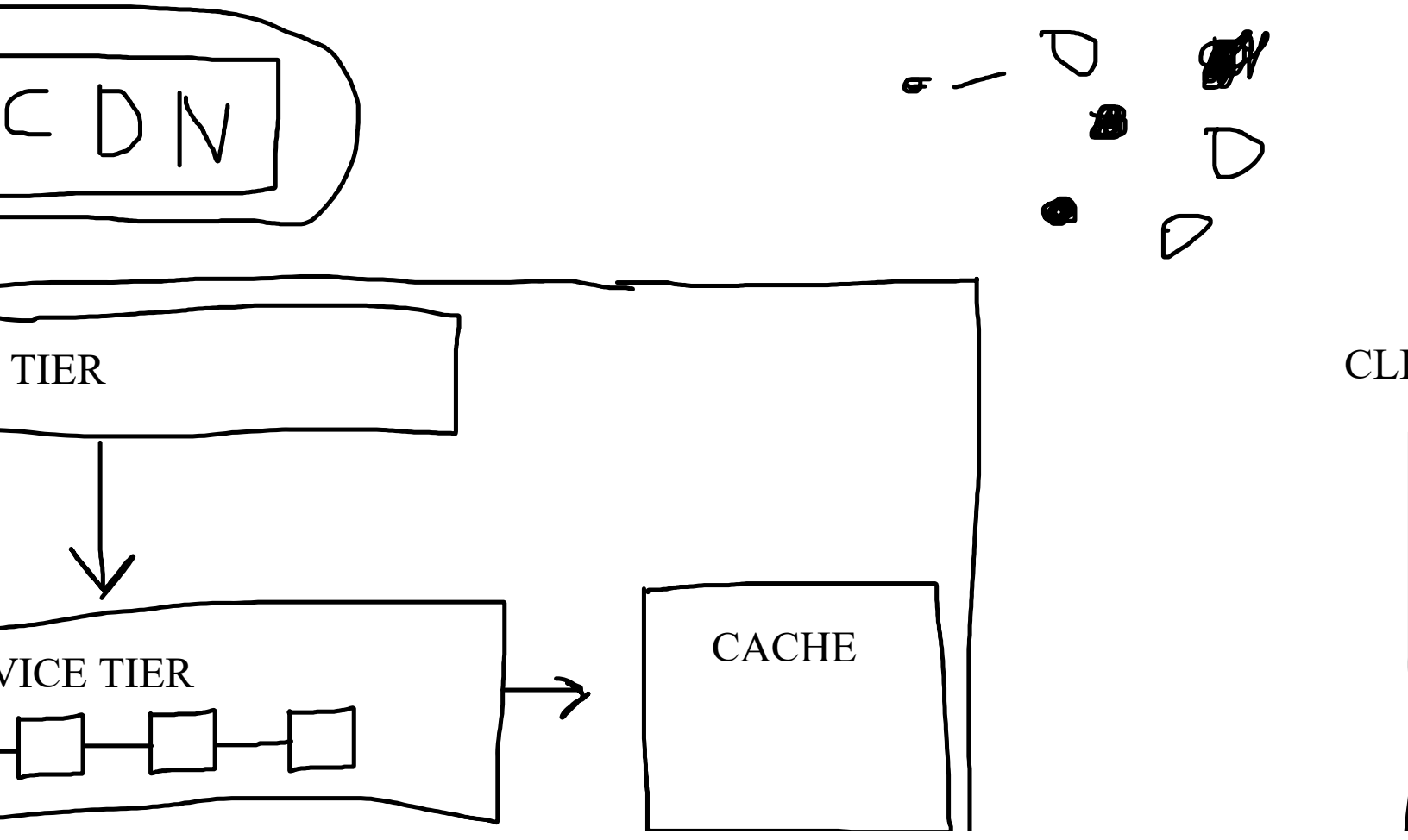
Friday, October 12, 2018 2:50 PM

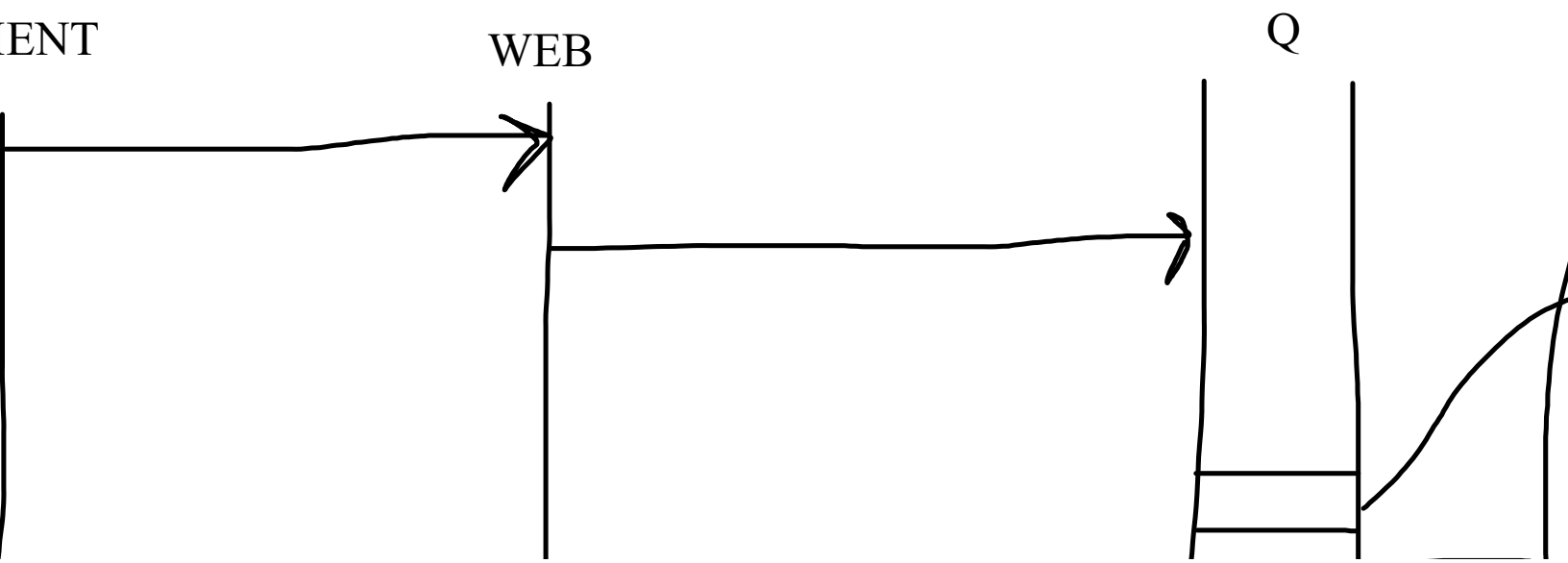
instagram clone

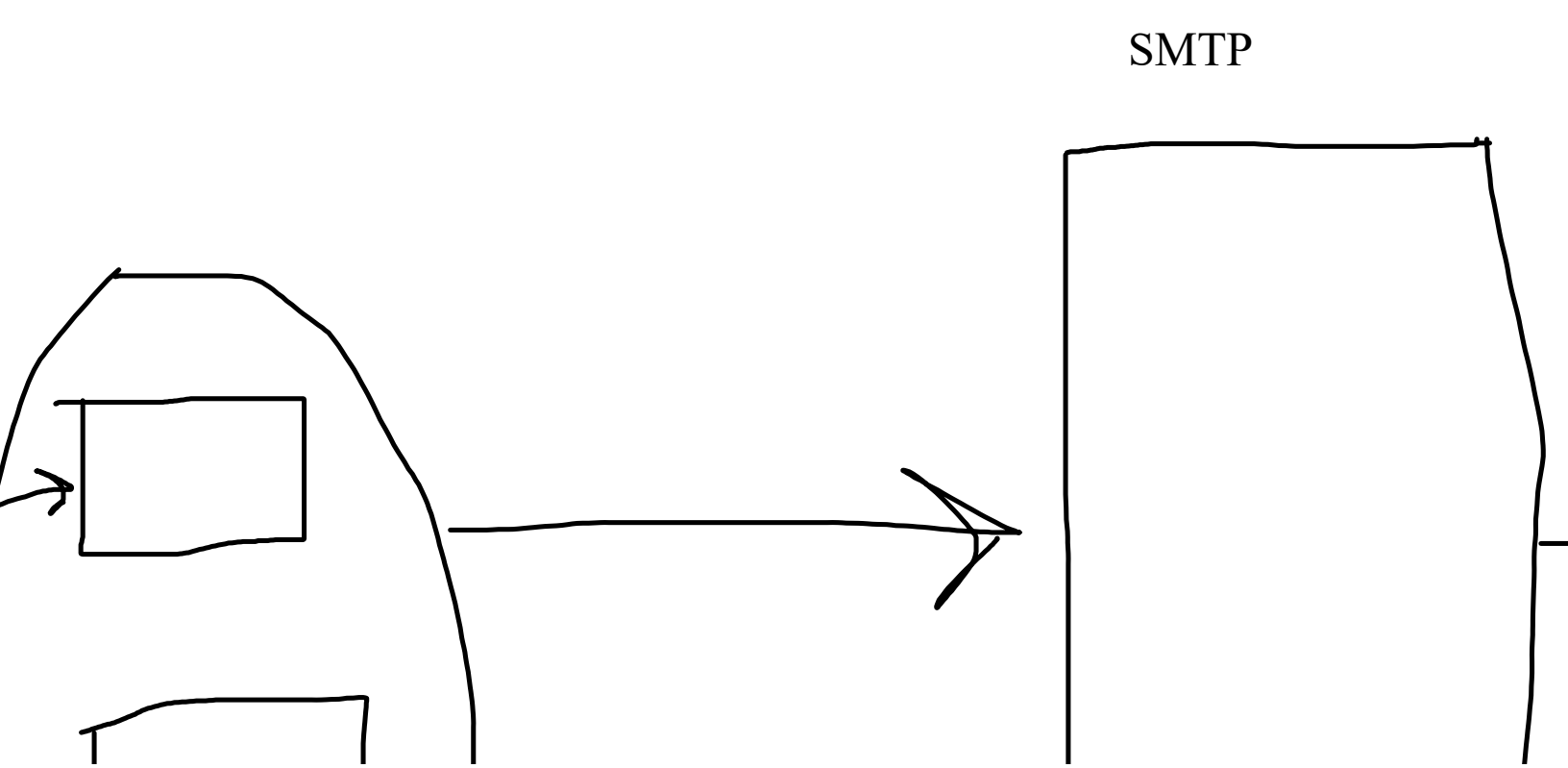
What we need:

- interface
- storage
  - images -> CDN
  - json files on disk
  - maybe database for comments only
  - image_name -> image_data: key, value for image storage
  - metadata about the images
- web servers
- auth services
- #dogs


ISP
WEB
MESSAGING
SERV

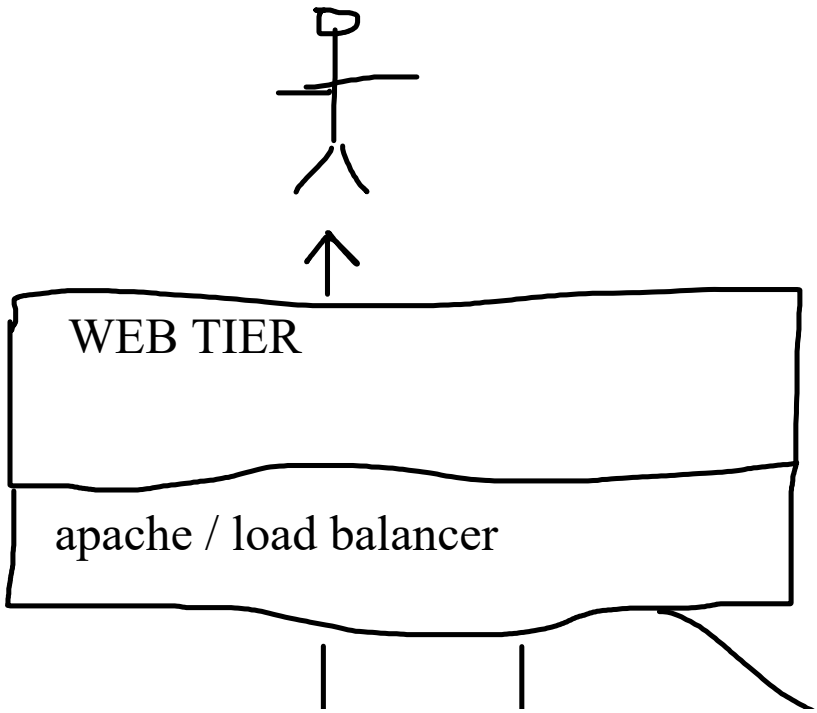

CDN
TIER
VICE TIER
CACHE
CL

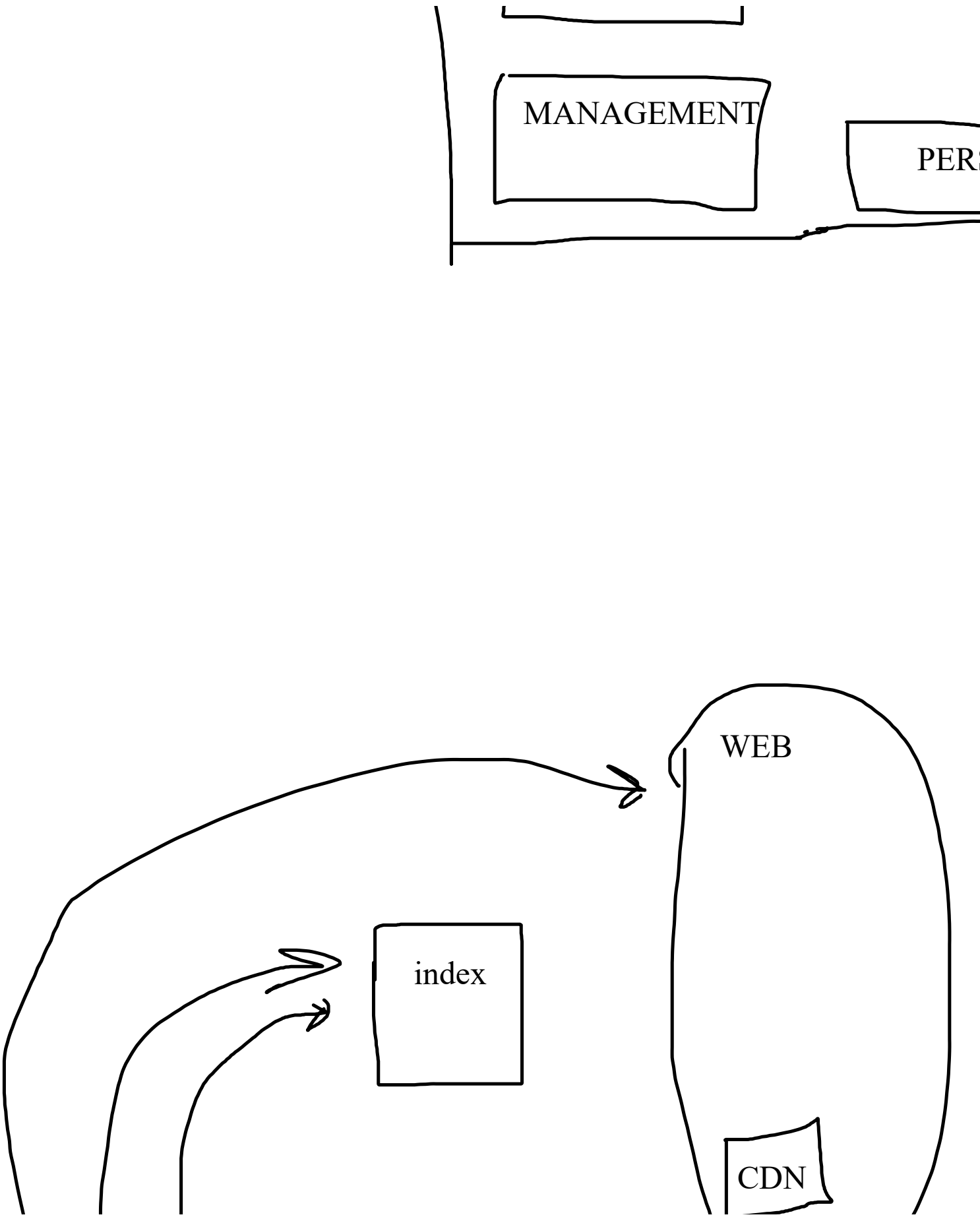

IENT
WEB
Q

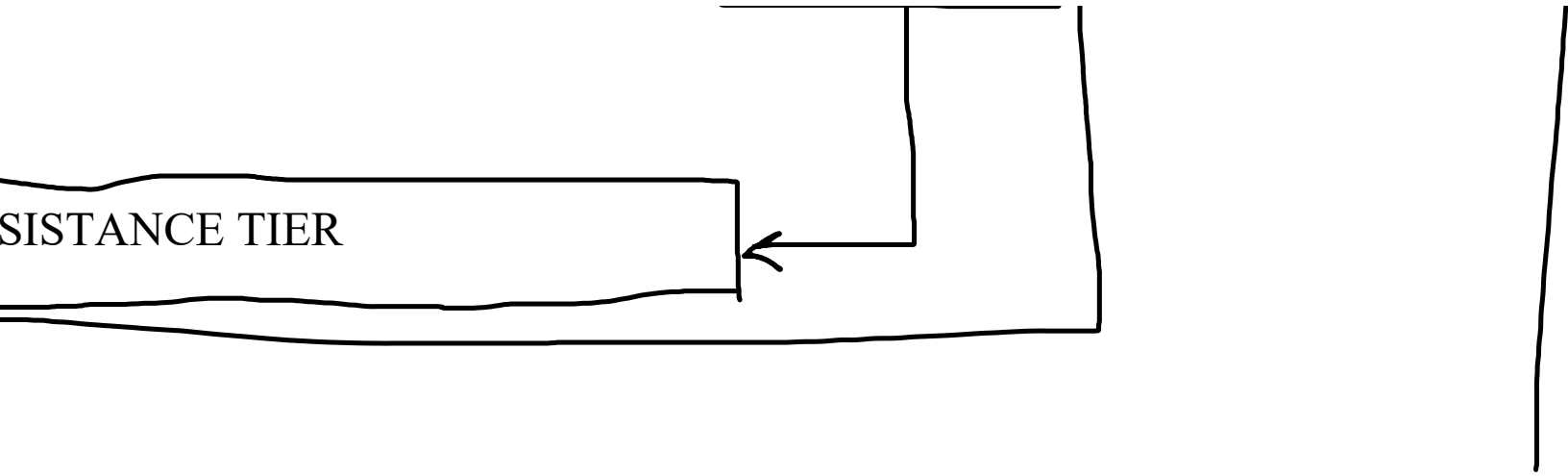

SMTP

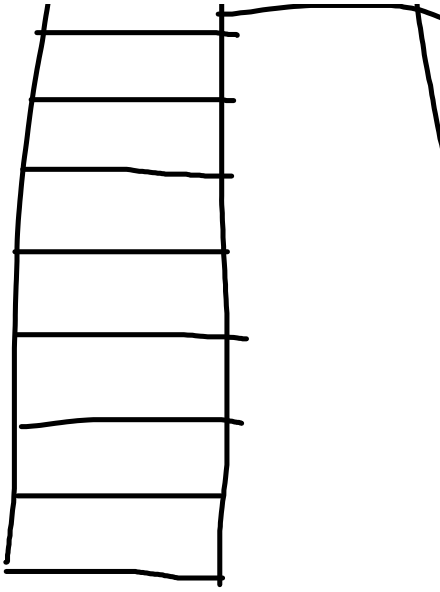

WEB

Just the idea

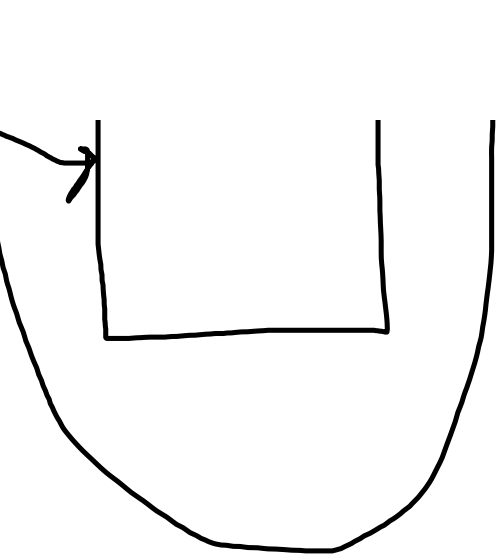


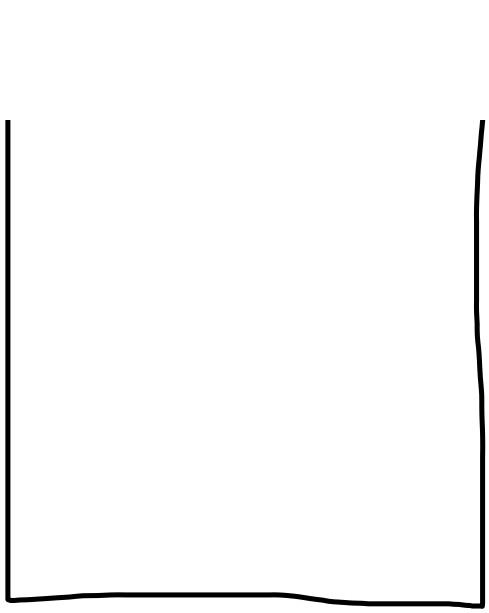

MANAGEMENT
PER
WEB
index
CDN


SISTANCE TIER

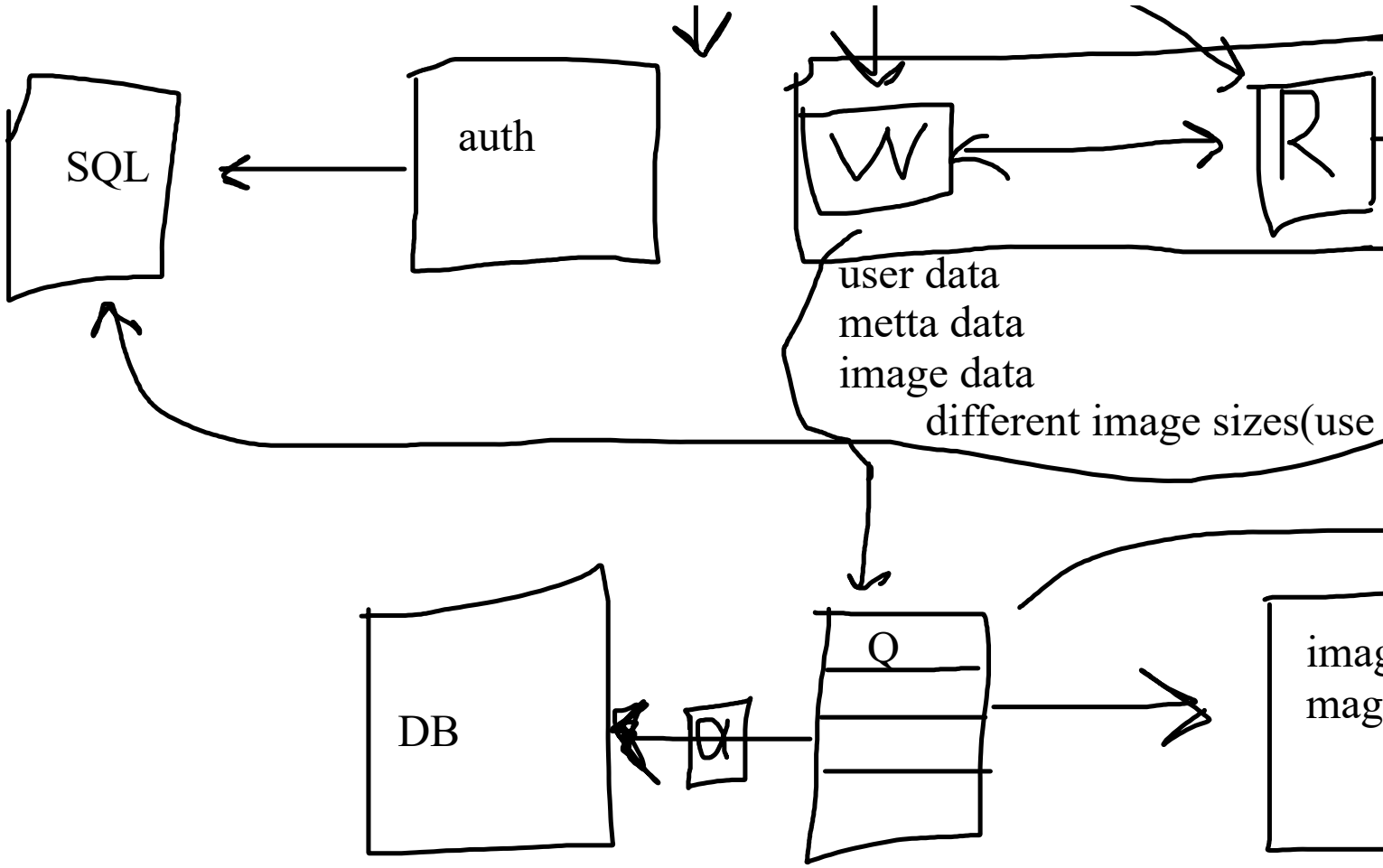


can spread out the work
asynchronous

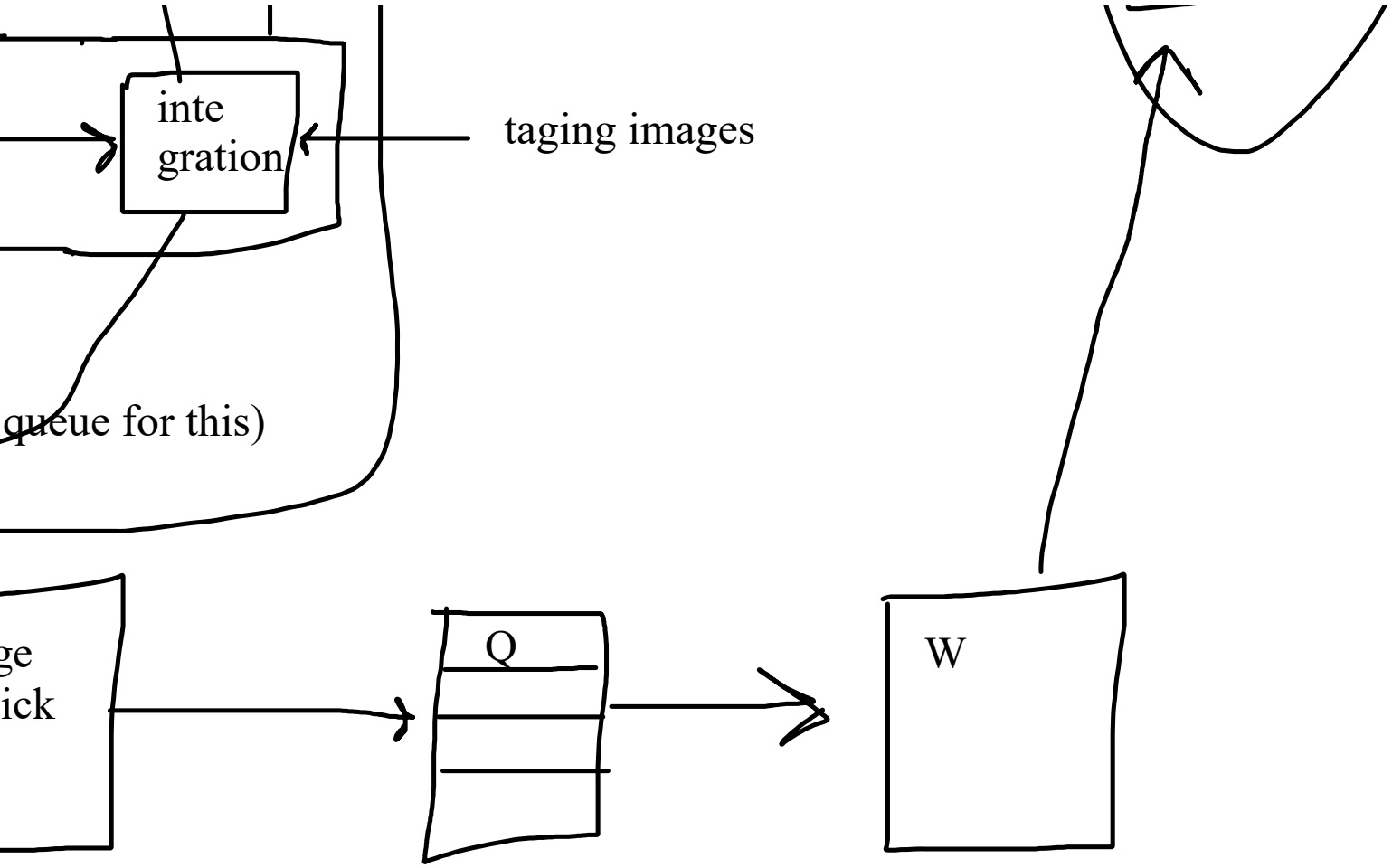




k over time



He will specify what we need
use components off the shelf as much as possible
- must be able to explain why you are using the component
- scalability
- latency
- breaking it appart
- frequency of requests


inte
gration
taging images
queue for this)
ge
ick
Q
W

testing
upload users and tokens to a cache
use them for testing

project

CS385 Cloud Systems Architecture Project - Fall 2018

Version: 1

October 19th, 2018

1 Overview

The goal of the project is to put in practice the concepts and techniques learned in class to build and deploy

an application in the cloud.

For this project you will be building an Instagram-like service (http://www.instagram.com). You will need to do a fair amount of programming in order to build the core components that are required to execute the business logic of the application. You will also need to set up and run other off-the-shelf components that might be necessary to support your application, for which you will also need to do some considerable amount of research.

You will need to work in groups of three/four students. Groups of different sizes are allowed subject to approval by the instructor.

2 Definitions

The following definitions apply to the next sections of this guide:

- **content**: Short for *digital content*. In the context of this project, it only refers to image media (i.e. no video, audio, text files, etc). There are many image/graphic media formats but for this project you only need to support JPEG (https://jpeg.org/jpeg/).
- **rendition**: A rendition is a scaled version of an image. There are many uses cases for them. For example, due to differences in screen resolution, it is more efficient to render images using scaled versions of an original image. Another common example is when images are presented in galleries, previews and thumbnails. Although you are not required to build a UI for this project, we want to simulate the process of creating these scaled versions. For this project, when an image is uploaded to your application, it will need to produce the following renditions:

Table 1: Image rendition sizes

Rendition	**Width (pixels)**	**Height (pixels)**
Thumbnail	161	161
Medium	612	612
Default	1080	1080
Original	–	–

- **tags**: Tags are keywords that are associated with an uploaded image. They are useful for users that want to promote their content, since tagged images will be included in the feeds of other users that subscribe to the tags that match the image's. For example, if a user uploads an image and tags it with the word "flower", this image will be presented to users that subscribe to the tag "flower".

- **feeds**: A user's feed is a personalized list of content that is presented to a user, based on the user's subscriptions and organized in chronological order. This list is constantly updated when other users add content and assign tags.
- **subscriptions**: Users determine their interests by subscribing to other users and also to tags. When

a user adds new content, all his subscribers will have the new content included in their feeds, as well as all subscribers to the tags associated with the new content.

3 Guidelines

You will need to **maintain your project's source code on a public repository** such as (but not limited to) github.com, bitbucket.com or gitlab.com.

As mentioned earlier, you will need to develop one or more components that will constitute the core business logic. This is clearly an effort that will require you to write source code. You can use any language that you feel comfortable with.

Besides the core application components, you will need to run other supporting services (based on of-the-shell components). **All components should run using Docker/Kubernetes and should be managed using scripts or automated processes as much as possible**. You should be able to manage your application with just a few commands. Kubernetes and Google Cloud Platform provide many tools for automation, and you should use leverage them.

A very important feature for this project is **the ability to scale up or down in a fast and reliable way**. You need to identify which components in your application need to scale, and provide mechanisms to do so. Your application should not suffer downtime during scaling operations, and scaling should be automated as much as possible.

You will also need to **identify and incorporate metrics** that help you evaluate the application performance and react to demand (e.g. to be able determine when scaling is required.) Google Cloud offers several tools for implementing metrics and autoscaling. You are also encouraged to do your own research and determine if and how they fit in your project.

Technical documentation should be included in the source code repository. Your instructor (or anyone else) should be able to compile, set up and run the application by just reading the documentation (and your grade in the project will depend on the ability of the instructor to run and experiment with your application).

3.1 Presentation

In the last week of the semester you will present your project. The schedule of presentations will be set by agreement with the instructor in the weeks prior. During the presentation, you should at least include:

- An overview of your application
- Description of the components
- Design challenges and decisions
- A demonstration of the tool/service that you provisioned
- Metrics and scale demonstration

3.2 Grade

The project will account for 20% of your final grade, which will be distributed as follows:

- Source Code and Technical docs: 15%
- Final Presentation / Demo: 5%

4 Functional Requirements

The following list provides a high level view of the functional requirements that your application needs to implement. Detailed specifications about APIs that need to be implemented are provided in the next section.

- The application will allow users to create accounts. You only need to support account registration, log-in and log-out.
- Users will be able to upload images with an associated description and an optional list of tags.
- When an image is uploaded, renditions will be generated. The renditions produced for an uploaded image should be accessible through their own distinct url.
- Content can only be added by authenticated users. Once content has been added, it will be associated with the account that uploaded the content.
- All content is public, there is no need to implement private accounts. Anyone, even unauthenticated users, can fetch images if they know the URL.
- Users can modify the description and tags associated with an image that they have already updated.
- Users can subscribe to other users and tags.
- Users are presented with a feed of images uploaded by the users and tags they subscribe, sorted by chronological order.
- Users can search for content. Search can be done based on tags or words included in the image description. Tags take precedence over descriptions.
- Users can remove their content. When content is removed, all image renditions, description and tags are removed and no longer appear in searches or other users' feeds.

5 Required REST APIs

5.1 Account APIS

Used to manage accounts. You need to implement the following endpoints:

`POST /account/register`

Used to register an account.

`POST /account/token`

Use to login an establish a session

`DELETE /account/token`

Used to delete/deactivate a token.

5.2 Content API

This API is used to interact with the content (images).

`POST /content`

Allows a multipart file upload. This endpoint also allows including comments and tags. This endpoint requires an active authenticated user, which will become the owner of the content. Returns the id of the content that was added. Since the content needs to be processed, you might not be able to return the URL of the renditions on the response, case in which you should return the appropriate corresponding status code.

`GET /content/{content_id}`

Retrieves the URLs of the content, it's owner, description and tags. It does not require authentication. Note that this service does not return the images itself, but the URLs where the different renditions of the content can be fetched. If the renditions are not ready, then a 202 status code should be returned.

`PUT /content/{content_id}`

Allows replacing the description and tags of a specific piece of content. Only the owner of the content can perform this action

`DELETE /content/{content_id}`

Deletes a piece content and its associated comments and tags. Only the owner of the content can perform this action

5.3 Subscriptions API

Subscriptions allow users to browse content that is associated with other users or their interests.

`POST /subscriptions/`

Subscribes an authenticated user to either another user or tags

`GET /subscriptions/`

`GET /subscriptions/`

Returns the list of subscriptions for the current user.

`GET /subscriptions/subscribers/{tag or username}`

Returns the list of subscribers to the current user

5.4 Feed API

`GET /feed`

Returns the feed for the currently logged in user.

5.5 Search API

`POST /search`

Returns a list of content whose tags or description match a search query. The search query consists of a single string.

6 Metrics and Scalability

As part of your project, you need to implement monitoring that allows you to automatically scale. You only need to implement this for the components directly associated with `POST content` and `GET content`.

7 Load Testing

As part of your project, you need to provide utilities/scripts that allow measuring the performance of the following APIS:

- `POST content` this test should upload randomly generated images of different fixed sizes. Note that for this test you should use many different accounts, so you probably want to execute first a setup task where accounts and tokens are created as well as the image files. Once this task is completed, then the actual test can be executed using the images and tokens as input.
- `GET content` this test should use the content generated in the `POST content` test to verify the performance of fetching that content. During this test you will only need to measure fetching the URLs of the content renditions, you don't need to measure the actual step of downloading the images.

Note that these tests are complex and you will not be able to use Apache Bench for them. It is recommended that you use a tool such as JMeter, Gatling, Locust, etc. These tests should be run in two modes: with

autoscaling of the application turned on and autoscaling off, as a mechanims to compare the effectivity of your scaling policies.

You need to include as part of the project's documentation the result of these tests and a discussion of the obtained results.

8 Changelog

- **10/19/2018**: Original Version.