Team **10-98** Members: Dailin Shen(ds3420)
Xiao Zhang(xz2627)

Summer COMS 4156 Final Report (June 30, 2017)

Part I Post-Mortem of New-Born ImHere Application

- **Brief Review of Revised Proposal**

The new-born ImHere application is equipped with *Timestamp Checking Mechanism* and *Coordinate Checking Mechanism*. The idea of two features comes from real life, in which late attendance is usually unexpected and distant student's sign-in is unacceptable.

New-born ImHere records the timestamp and the coordinate of each active course attendance window at first. Once a student submit a magic code, the application will not only get the code, but also track the timestamp and the coordinate. Student's provided magic code, timestamp and coordinate are compared respectively with the real magic code, course start-timestamp and course start-coordinate. Any sign-in with a timestamp 15 minutes later than the affiliated course start-timestamp will be marked as a late attendance; any sign-in with more than a 25-meter-difference of coordinate compared to course start-coordinate will pop out a message saying "Sign in area out of range. Try again! "

We hope the integration of *Timestamp Checking Mechanism* and *Coordinate Check Mechanism* would help securing a fair validation of course attendance and further help on students' performance.

- **Fulfillments**

1. During the implementation, we found some bugs existing in the original code which had to be fixed, such as:
   a. Error happens when opening attendance window with two or more students
   b. Correct secret code cannot be recognized
   c. Error happens when a teacher opens more than one attendance window
   d. Magic code is not updated on teacher's end when a teacher tries to generate the magic code for a class multiple times
2. We finished every mechanism we've planned in the revised proposal: *Timestamp Checking Mechanism* and *Coordinate Check Mechanism*.
3. Added an interface for teacher to view the statistics of student's late attendance records.
4. Crystallized UI design. Modifications are ranging from submit button to class layout to different notification messages.

*5. Only allowed integers as an input for magic code under student account.

*6. Limited the number of active attendance window under either student account or teacher account. In other words, a teacher has to close the existing open attendance window (if there is any) on his/her dashboard before he/she opens another one.

*7. Eliminated chances of classes with same class names. When a teacher attempts to add a class, if he/she types a course name that is already under his/her dashboard, the system would pop out a message saying "The course is already on your account. Please try again."

All the fulfillments above have been successfully tested with unit tests and system-level tests. Link to unit tests. Link to system level tests.

NOTE: * before a number index means the feature we newly added and is not proposed in revised proposal.

- **Challenges**

1. Getting coordinate of student's sign-in is hard.

The hidden idea of getting coordinate is converting IP address into coordinates. Before we made a conclusion on this, multiple ways were attempted including calling various IP API getters and using Flaskr built-in module, request.

It turned out that request class under flask module worked the best. Before we concluded on this, we only used user's IP address and convert it to coordinates. The problem of this was obvious. It would only work on local machine but not for Google Cloud at all. When we ran on gcloud platform, out system kept getting the same coordinates on teacher's end, which was actually where the google server was located in California. That is unexpected. We would like to have user's coordinates.

So then we switched to flask request class. By calling request.remote_addr, we got the IP address. Then we passed the IP address to a coordination conversion API url we found: http://ip-api.com/json. A json file with longitude and latitude would be generated, in which we could grab information from. This is the final solution to get location information in our *Coordinate Checking Mechanism*.

However, there is a drawback of the chosen method of getting coordinate. If the ImHere application is running on a local machine, calling request.remote_addr and further converting the result into a json would not generate any coordinate information at all. In order to avoid seeing coordinate errors and enable our system running locally, we assigned the coordinate to be [0, 0] if the application would run on a local machine. Hence, this is an execution difference between local environment versus Google Cloud Platform.

2. Getting a timestamp corresponding to coordinates or area zone is challenging.

When tracking the timestamp for an active course's attendance window or a student's sign-in, we would like to record the timestamp corresponding to the coordinates. First we recorded the datetime of each user with datetime.now(). But then we realized that the time was recorded in

UTC time instead of local time once the application would be running on the cloud. Since each timestamp we grabbed by using datetime would be saved in UTC time, the system would still be able to compare the student timestamp with course start-timestamp.

However, when it came to UI design, showing UTC time on the screen was too confusing. At the first several attempts, we just tried to change the datetime's 'tzinfo' attribute, which saved the timezone information - but still, when the timestamp was printed out, we could only saw UTC time on the screen, just followed with a different timezone information. It was not readable, so we switched the idea to change the timestamp before rendering an html file.

We used the timedelta() function to modify the recorded datetime object. We basically minus four hours from the recorded timestamp, which would be converted to US/Eastern time. When we tested it, we found the time that matches local time on the UI.

Since ImHere is an attendance checking system for Columbia University, and there is only 4 hour difference between UTC time and US/Eastern time, minu-four-hour solution was hard-coded. Although it turned out that the simple minus-four-hour solution fitted our condition the best, we're eager to find a better way storing area-aware timestamp instead of manually subtracting the time difference before rendering an html page.

3. More strict requirement for application deployment on Google Cloud Platform

In the unit test, we wanted to use html's POST to add class and remove class from teacher's account. The adding part can easily be done, since the add_class() only needed a name for the class. But when removing, the remove_class() needed a cid to operate and remove the corresponding record in the datastore, which we can't capture from the POST method. Thus in order to have the cid for newly added class, we used datastore.Client().

When we ran the tests locally on the virtual machine, we kept the datastore emulator open. It automatically told which project is using it, so datastore.Client() was enough to get the data and all tests passed. But when we tried to run the tests on google cloud shell, we need to specifically put the project ID in our code, to tell which project was requiring the data.

In addition, we also have to specify the ssl library we're using when deploying our application on Google Cloud Platform. To sum them together, deploying an application on Google Cloud Platform uses much more strict requirements compared to that in local environment.

Part II Feedbacks From Demo And Potential Enhancement Features

- **Feedbacks**

Empty secret code input made our application crash, resulting in an error message saying "Internal Server Error". The problem was brought up because of comparing a Nonetype input with an integer. Generally there are two solutions. One, convert what we grab from secret code input box into integer type first and then compare it to the actual secret code which is an integer type as well. Second, assign the input box of magic code on main_students.html as a required input box. This would prevent the circumstance of unmatched type comparison.

The concern was immediately fixed during the class. When a student clicks on “submit” button with an empty secret code, the application would pop out a notification saying “Invalid Secret Code. Please try again.”

- **Potential Enhancement Features**

1. Have a mechanism checking if the logged-in UNI is a student UNI or a teacher UNI. If it is a student, he/she cannot have access to register as a teacher.
2. Accept columbia email address only during registration. Have a mechanism automatically extracting the affiliated UNI from logged email address rather than today’s manually UNI input box. It is apparent that UNI extraction is far more inaccurate than a human input.
3. The ImHere application combines with course registration system. Suppose a student successfully registered a class, if ImHere application is tuned with course registration data, registered students will be automatically added to the affiliated class. Successfully registered classes would be automatically updated to students’ ImHere dashboard as well. The pre-condition of this new feature is that ImHere application has an access to share school’s registration database and further get course info.
4. Add a field for teacher when opening attendance window: the minutes that attendance window remains valid. This way teachers can have the freedom of setting how many minutes after attendance is open, students will be marked as late. So any signin beyond those time would be marked as late-attendance. Now we have set this time to be 15 minutes, hard-coded in the backend.
5. Check and record the timezone when opening attendance window. Right now the time we recorded has been forced to minus 4 hours from the UTC time in order to match the local time, but it can be vary if we record the timezone information and make it do the calculation with different hours from central time.

Part III Project Links

- **New-born ImHere Project Github Repo Link:**
  https://github.com/dailinshen/coms4156_jumpstart

- **New-born ImHere Project Google Cloud Platform Link:**
  https://coms-4156.appspot.com/

- **System Level Testing Results & Static Analyzer Link:**
  https://github.com/dailinshen/coms4156_jumpstart/issues/15