CS262 Distributed Systems Final Project: Digitally-Facilitated Workshop Participation Application for Harvard College Office of BGLTQ Student Life

Live at http://tq-timeline.herokuapp.com/

Harshita Gupta

May 11, 2018

1 Introduction

In the Harvard College Office of BGLTQ Student Lifes (QuOffice's) Thinking Queerly workshops, the facilitators prompt participants to interrogate ways in which they've learned about normative gender and sexuality across their lifetimes. To this end, we lead two "timeline activities" which currently waste a lot of paper and are cumbersome to digitize and commit to institutional memory. For my final project, I would like to build a distributed system that facilitates the activity during the workshop and allows students to participate anonymously via their smartphones.

Traditionally, the activity is conducted by handing out post-its to the attendees and asking them to write down, on individual post-its, specific moments during which they learned about gender and sexuality across their lifetime. They then stick their 6-10 post-its on a whiteboard, plotting each post-it along a timeline of their life. Once participants are done adding their post-its to the timeline, the group gathers around the timeline and looks for trends/similarities across post-its.

This activity is arguably one of the most popular of the QuOffice's offerings. We've generalized the timeline activity format to offer another similar activity in which the board members of student groups may think about how gender and sexuality is represented in their group across the course of the academic year. The combination of the two provides an opportunity for specific, data-driven critical reflection

This format of a post-it-based timeline constructed on butcher paper, however, is currently very awkward and cumbersome since the entire group cant gather around the whiteboard and view the material in any efficient way.

1.1 Proposed System

The proposed system would be a mobile and desktop compatible web app. It would offer the following functions:

- **Workshop Administration** Allow QuOffice staff to create a session via an admin panel. The session would specify the question that participants must respond to via post-its, and specify the unit of measurement for the timelines timeseries: a numerical age or MM/YY date, to allow for different types of timeline questions like How has your group represented gender and sexuality over the course of the calendar year (a MM/YY timestamp) versus how have you learned about gender and sexuality across your life (a numerical age timestamp).
  - Allow the QuOffice to close the session via the admin panel so that it doesnt accept any more submissions.
  - Work on a limited budget: either use Azures services that are free for students or constrain the budget to a specific amount. Warn the QuOffice when the storage usage is nearing a budget constraint. Allow the QuOffice to export past session data as a CSV via the admin panel and delete it from the server to free up space for future sessions.
- **Participant Contribution** Allow participants to contribute to a specific session over a mobile-compatible interface, type the content for their post-its, and plot each post-it along a timeline.
- **Timeline Viewing** Have a well-designed view that visually represents the timeline and that can be projected on a screen at the front of the room. This view would live-update as post-its are submitted.

  - Have a mobile-compatible version of the same visual timeline so that users can examine the curated submissions more closely on their own phones and scroll through them.

1.2 Distributed Systems Problems Tackled

- Working on a limited financial budget. This will require: Finding an Azure/AWS/Google App Engine configuration that takes advantage of our @college email perks as much as possible. Constraining the app's resource usage to work within the free accounts constraints. (primarily storage, since workshops only have about 40 users at once)
- Concurrency: participants will be adding content to the timeline simultaneously during the course of the session. The system should handle this gracefully and update the live timeline gracefully as well.
- Immediate consistency: since this is a live workshop, the data will all be centralized and pushed to all users immediately.
- Live updates: we will need to implement an efficient delta updates scheme so that the live-updating of the timeline is quick and not a drain on the network, since workshops can often end up with $>$ 100 post-its. Sending all of the timeline to each participants phone on a refresh would not scale well.
- Support multiple simultaneous workshops: as the QuOffice hopes to expand its offerings, if we do multiple simultaneous workshops during Opening Days, the system should support traffic to multiple active sessions at once.
- Anonymity: due to the sensitive nature of the content being collected, we should ensure that no IP information/identifying information is retained about submissions.

2 The Application

2.1 Web Development Stack

Given that my intent was to develop an application for real-life use at the QuOffice, I prioritized deployment, ease of maintenance, and usability. This motivated my decision to deploy my web app at regular checkpoints during development, to minimize deployment integration time at a later point.

I've built web apps before with Python and Flask and found it maintainable, intuitive, and lightweight for simple apps that don't need to scale past 100-1000 users at a time. Flask is a clean web development stack in its separation of model-view-controller and usage of pythonic language across each of these layers. It uses the templating engine Jinja to dynamically generate HTML pages in response to GET and POST requests, and allows all server logic to be written in Python.

As an example, below is an extract from a jinja template that would be rendered with flask's render_template call and the additional parameters posts and sessions.

```
<section class="timeline">
  <ol>
    {% for post in posts %}
      <li>
      <div {% if post.on_sex %} class="onsex" {% endif %}>
        <time>
          {% if session.unit_is_year%}
            {% if post.mdy_timestamp %}
              {{ post.mdy_timestamp.strftime('%B %e') }}
            {% endif %}
          {% else %}
            Age {{post.year_timestamp}}
          {% endif %}
        </time>
          <p>{{post.body}}</p>
      </div>
    </li>
    {% endfor %}
    <li></li>
  </ol>
</section>
```

Through the use of curly-brace-enclosed templating language, HTML is turned into a jinja template.jinja can auto-generate HTML tags and fill them with the content of variables, without using any javascript.

I chose to work with Heroku because it pairs well with Flask deployments and supports a free tier of PostgreSQL.

2.1.1 Virtual Environment

To ensure that my local development would require minimal deployment overhead, I developed my application within a virtual environment and custom installs of all dependencies. Python's package manager, pip, handled dependency installation and generated a dependency requirements file that Heroku could read from directly.

2.2 Design Decisions

2.2.1 Database

My application is interested in storing the following types of data:

- A list of workshops, and data about each workshop: what is the title of the workshop, when was it created, what question does it ask participants to engage in for its timeline, what is the measuring unit for its timeline, is the workshop still acive, etc. The workshops should also have a O(1) link to the post-its associated with that workshop.
- Data about the post-its submitted to each workshop: their content, categorization, and position on the timeline.

- A list of administrators and the credentials necessary to log them in.

Given the highly defined structure of the data, I chose to use a SQL rather than NoSQL database to store information. Flask has a very convenient extension called Flask-SQLAlchemy that provides a flask wrapper to the popular Object Relational Mapper SQLAlchemy. The ORM allows Flask applications to manage a database using high-level entities such as classes, objects and methods instead of tables and SQL. SQLAlchemy is an excellent fit for our use case due to its additional property of supporting multiple database engines like MySQL, PostgreSQL and SQLite. This is extremely powerful since Heroku uses PostgreSQL while I will develop locally with SQLite (since SQLite does not require a server). Therefore I can use Heroku's more robust production tools when I deploy, but develop locally with SQLite, and not have to change my application.

I achieve this multi-platform support with a single line of Python code that searches for a Heroku-defined environmnt variable to a database, or sets up a link to an sqlite local databse file:

```
SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
    'sqlite:///' + os.path.join(basedir, 'app.db')
```

The relational database schema is defined as follows:

Admin
+ id : uint, primary key + username : string(64), unique + email : string(120), unique + password$_{h}ash$: $string(128)$ + workshops$_{c}reated$: $arrayofWorkshopActivity$
+ setPassword(p : password) : void + checkPassword(p: password) : boolean

WorkshopActivity
+ id : uint, primary key + unique$_{s}tr$: $string(20), unique, indexed$ + name : string(100) + question : string(140) + unit$_{i}s_{y}ear$: $boolean$ + admin$_{o}wner$: $uint, (primarykeyAdmin)$ + postits : array of PostIt + creator : backref to Admin

PostIt
+ id : uint, primary key + body : string(500) + year$_{t}imestamp$: $int, optional$ + mdy$_{t}imestamp$: $date, optional$ + on$_{s}ex$: $boolean$ + session$_{i}d$: $uint(primarykeyWorkshopActivity)$ + session : backref to WorkshopActivity

While most of this schema is self-explanatory, I would like to explain further the decision to have a unique string for each workshop that is distinct from the id of the workshop. This is a first step in making the app secure against outsider/aggressor attention: the QuOffice's online materials often get picked up by sites like Breitbart, and so I would not want the content of timelines to be easily locatable by such sources.

Additionally, I use the Flask extension Flask-Migrate to handle database creation and migration for Flask. Flask-Migrate's benefits are twofold: first, it adds database creation and management

services for native flask development. Secondly, it is a Flask wrapper for Alembic, a database migration framework for SQLAlchemy that allows making updates to the database schema over the app's lifecycle. Typically this is challenging since relational databases are centered around structured data; Flask-Migrate makes this easier.

2.2.2 Authentication

The application requires authentication for the following purposes:

- Exposing an admin panel to administrators only
- Allowing the creation, activation, and deletion of various workshop sessions.
- Allowing administrators to download CSVs of workshop data: we do not want to allow our data to be bulk-exported and circulable by the public

Therefore, the only access level that needs to be managed is that of the administrator. We will make timeline access public for all who know the unique string for a workshop.

Authentication and access management is handled by another very convenient flask extension: Flask-Login. Flask-Login manages information about user logged-in state, so that users can log in and navigate to different website pages while the app "remembers" that they're still logged in.

Flask-Login also protects specified view functions, or webpages, from users who aren't logged in. Through a simple login_required decorator, specific view functions like the admin panel's and the download/delete button's can be access-restricted.

2.2.3 Styling

Given my lack of interest or expertise in designing compelling front-end interfaces, I am always interested in having other, more artistically inclined people, do my work for me. I initially considered using Webflow to design the pages; I then would have exported the HTML/CSS/JS that Webflow generated and modified it to create templates that could have content dynamically inserted into them. I ended up not going this route, primarily because Webflow would have charged me $16 for attempting to download the source files of the designed websites. After this paywall hit me I also realized that using webflow would be overkill. I ended up using Flask-Bootstrap, another(!) flask extension instead.

Flask-Bootstrap provides a few very useful features. It:

- automatically applies basic bootstrap styling to the entire website when imported into my base template.
- plugs into jinja to provide automatic stylized templating of forms, which are much of the content of this app
- immediate mobile compatibility

Applying flask-bootstrap to my app only took a few lines of code. Additionally, I was able to replace the following form rendering template:

```
<form action="" method="post">
    {{ form.hidden_tag() }}
    <p>
        {{form.name.label}} <br>
        {{form.name}} <br>
        {{form.unique_str.label}} <br>
        {{form.unique_str.description}} <br>
        {{form.unique_str}} <br>
        {{form.date.label}} <br>
        {{form.date}} <br>
        {{form.question.label}} <br>
        {{form.question}} <br>
        {{form.unit_is_year.label}} <br>
        {{form.unit_is_year}} <br>
    </p>
    <p>{{ form.submit() }}</p>
</form>
```

with the below, and have it look beautiful styled! Win win.

```
{\% import 'bootstrap/wtf.html' as wtf \%}
<div class="row">
    <div class="col-md-4">
        {{wtf.quick_form(form)}}
    </div>
</div>
```

2.3 Website Structure and Features

Ultimately, the website ended up with the following pages and features:

- Homepage - few features, currently has links to other pages for your grading purposes, but will eventually be empty
- Workshop contribution pages: found at the pages at /session/ <session-identifier>
  Intended to be the landing page for workshop participants. This is the link that QuOffice staff will write on a board or put somewhere.
- Timeline viewing pages: found at /session/ <session-identifier> /view
  These are the pages that contain the crowdsourced post-its and display them along a scrollable, mobile-compatible timeline.
- Login, Register, and Logout Pages
- Admin Panel at /admin_panel
  - Create new workshops
  - View all existing workshops

  - Download content from existing workshops
  - Delete workshops

2.3.1 The Timeline

While I initially considered developing the visual components for the timeline myself, I discovered that there were multiple HTML/CSS/JS timeline templates available online. I selected Envato Tuts' timeline at https://codepen.io/tutsplus/pen/ZKpNwm, and modified it to work with jinja, because it had the best separation of content and style (minimal styling in the HTML tags), and so would support easy extensibility, and because it was colorful and most resemebled the post-it format we'd had earlier. It was additionally mobile compatible!

3 Results

3.1 Live Website

The website is currently live at http://tq-timeline.herokuapp.com/, and has achieved all my stated goals. Some limitations of the current version that I plan to fix once I polish it for the QuOffice (and once Professor Waldo has looked at it):

- currently allows anyone to register, will eventually only allow other admin to create new admin
- doesn't currently allow "turning off" sessions or hiding timelines from the public
- no live update or delta update, you have to refresh the timeline view to get its most recent version

I am quite happy with the final product and impressed at how much Flask and its extensions sped up development; I achieved far more goals than I expected to.

To test the website, I would take the following steps:

- Go to the landing page.
- Click Log in
- Click register
- Register an account
- Log in with this new account
- Check out the admin panel! I have populated it with two sessions of dummy data - follow the links to various timelines and their contribution forms, and try creating your own workshop yourself. Download some session data to test that functionality as well.
- On another browser, open up a timeline contribution link and submit some post-its. View the timeline results in a third browser window. After you submit a result, refresh the page and see the timeline update.

- Try any of the above steps, especially viewing the timeline, on a mobile phone (or simply by resizing your browser window). It's quite smooth, if I might say so myself.

4 In Summation

This project has been a very successful first step towards building a robust timeline application for the Harvard College Office of BLGTQ Student Life's Thinking Queerly Workshop Activities. We have successfully built a fast, cheap, lightweight full-stack web solution that can effectively support timeline activities on multiple devices. The code is open-sourced and ready to be extended.

The next area for focus in the development of the app is security-related features. A few simple features relating to this remain: allowing the QuOffice to "close" or deactivate a session, allowing them to keep the timeline's data in app but close the view off from the public.

I would ideally like to allow the QuOffice to optionally enable "monitoring", so that submitted post-its only appear on the visual timeline after theyre screened by another staffer in live-time. Since we sometimes conduct workshops with groups that aren't ready to maturely engage with our content, we would want to preserve the integrity of the space by not letting the digital mediums distance enable peoples irresponsible sides.

I would also like to look into DDOS protection for this app – this is not an area that I have any knowledge or expertise in, but would be something that I'm interested in learning about.

On the visual end, I would like to make the timeline view use screen real estate more wisely – I would like the timeline to be more densely packed and use the entire vertical screen area by stacking post its, etc.

The non-engineering next-steps would be to fine-tune the application with QuOffice staff, and then attempt to run a timeline activity with it. I am curious to see whether participants will feel a positive or negative change in their level of engagement with the activity, now that it's digitized and on their phones. Additionally, will the richness of the post-it contents change?

Ultimately, an excellent first step. I hope that you will get to see the application live when Thinking Queerly comes to your department meetings :)

5 Acknowledgements and References

Miguel Grinberg's Flask Mega-Tutorial, for teaching me how to use Flask two years ago, and for being a very handy reference manual as I returned to it!

Harvard College's Office of BGLTQ+ Student Life, for being one of the best student employment opportunities on campus.

Professor Waldo, for giving me the amazing opportunity to begin work on this project! Cheers.