Group name: PASSIO (Adams, Yuki, Kaycee, Jared, Bryce)
Lab time: Friday 8-10am

Milestone #5: Testing Update Report (Due April 12, 11:59pm)

For your final submission, you will need to produce a video walk-through and overview of your project as well as complete a report detailing the following items. Please make it clear in your responses which item you are addressing.

Video Walkthrough (10%)

https://youtu.be/6iN0cN7LkSM

Project deliverable (90%):

(2%) Update the requirement document, including:

- A brief description of the software you are building:
    - Our software is a web application that allows users to search for, find, and purchase tickets to music events in their area.
    - Our goal with this is to allow users to have streamlined, visually appealing access to the events they are interested in or exploring.
    - A brief overview of the website framework is: A verified host can create and request their event gets posted on the website. This event is reviewed by a verified website administrator, who can approve or decline the event. If the event is declined, the host sees this event under their declined events and will need to re-submit their event after making the necessary improvements. Once an event is approved by a verified administrator, it is posted to the website. This is where attendees can log into the website, search for/ view these events, and purchase tickets to their preferred event.

- A list of user groups for your software, along with an example scenario for each user group of how they will use the software.
    - Event Host:
        - These are the individuals who are creating and posting the events for individuals to attend.
        - Scenario:
            - The host logs into the system, and the system verifies that they are a verified host.

            - The host is then able to navigate to the event entry page, where they are prompted to enter their event details.
            - Upon submitting this form, the event is sent to a PASSIO administrator for approval.
            - If the event is approved by the administrator, users will be able to view and purchase tickets for this event. The host will also be able to edit this event as well.
        - Extensions (for scenario):
            - If the log in details are not a valid host.
                - System issues an error message and asks the user to enter valid credentials. If the credentials are for an attendee, the individual will be logged in as an attendee.
            - If the administrator declines the event:
                - The event will not be posted to the website and the host will see their proposed event under declined events under the event approval tab. Host will then need to re-submit the event.
    - <u>Event Attendee:</u>
        - These are the users who are using the website to purchase tickets for events that are approved to be on the website.
        - Scenario:
            - The attendee logs into the system and the system verifies that their credentials are valid.
            - The attendee can then search for the event they want to attend. The system returns a list of relevant events based on the search information entered.
            - The attendee clicks on the event they want to attend to see more details. They then click Add Ticket to Cart which adds a ticket for that event to their cart and redirects the user to the checkout page where they can review their purchase and submit their payment information to complete the event purchase. The system will confirm they entered payment information and connect a ticket to the attendee's profile upon purchase.
            - Tickets database and Attendee database information will reflect this purchase.
        - Extensions (for scenario):
            - Invalid login credentials entered

            - The system issues an error and will prompt the user to re-enter their login credentials.
        - Search invalid for search information entered
            - The system will issue an error indicating no search results were found and ask the user to re-enter their search information.
        - Event sold out
            - If the attendee clicks on an event that has no tickets left, the system will issue an error and indicate that there are no tickets available for the event, should the attendee attempt to purchase tickets. The attendee will not be able to purchase a ticket for this event and will remain on the “events” page.

- <u>PASSIO Administrator:</u>
    - This user is someone who handles the reviewing of submitted events, and decides if the event is appropriate to be posted on the website. Their approval is required for an event to be posted.
    - Scenario:
        - The administrator logs into the website. The system verifies that the login credentials are those of a validated administrator.
        - The administrator can navigate to the event approval page from the navbar containing a list of events that need to be approved.
        - After reviewing that the event information is appropriate for the website, the administrator approves the event. The system changes the status of this event to verified, which will make the event viewable to attendees.
        - If an administrator decides that the event isn’t up to standards they can decline the event. Once an event is declined it will show up under declined events for the host account that sent it to get approved.
    - Extensions (for scenario):
        - Invalid login information:
            - The system will throw an error and indicate that the login credentials are not a valid login.
        - All events have been approved.

        - If there are no events for the administrator to review, then they will be greeted with a blank page that indicates that there are no events to review.

- The final list of requirements of the software that was built. Incorporate feedback from M2.

1. System requirements
    1.1. Functional Requirements
        1.1.1. General (for the entire system)
            1.1.1.1. The system will validate a user's login credentials to allow users access to proper regions of the system and apply constraints to avoid them accessing incorrect regions. Example: an attendee should not be able to/ have to worry about navigating the event submission page that is made for hosts.
        1.1.2. Host
            1.1.2.1. The system provides a page for the host to enter all relevant information to submit their event(s).
            1.1.2.2. The system allows the host to update/edit/cancel events as required.
            1.1.2.3. When an event is created, its unique identifier is its auto - generated ObjectId that Mongo DB provides
            1.1.2.4. If an event is declined, the host will need to re-submit the event.
            1.1.2.5. If an event is approved it will be viewable for attendees to purchase tickets for. It will also create the predetermined number of tickets available for the event.
            1.1.2.6. If an event is edited it will not be available for users to view/purchase tickets on until it is approved again by an administrator.
        1.1.3. Attendee
            1.1.3.1. Attendees are able to view all available/approved events within the system.
            1.1.3.2. Attendees are able to search up their preferred event by typing in the event name/artist/venue/genre into the search bar or by selecting a date.
            1.1.3.3. Attendees can easily find their desired event by entering the event name, artist, venue, or genre into the search bar, or by selecting a date.

1.1.3.4. If an event is sold out, attendees are not able to purchase a ticket to that event.

1.1.3.5. Each ticket is assigned a specific ID which it can be found in the database with.

1.1.3.6. When an attendee purchases a ticket, it is linked to their account.

1.1.4. Administrator

1.1.4.1. The administrator is the only one who is able to view all submitted/unapproved events.

1.1.4.2. They are able to review any of these events individually to determine if it is suitable to be posted on the website.

1.1.4.3. Once the administrator has reviewed an event, they simply select if the event is “approved” or “declined”. If it is approved, the event will be viewable in the events homepage. If it is declined, the host will see it under declined events and will need to re-submit the event.

1.2. Non-Functional Requirements

1.2.1. Product Requirements

1.2.1.1. If someone knows what event they want to go to and they have their payment information ready, purchasing a ticket should be doable in under 90 seconds.

1.2.1.1.1. Open website > search for the event > click purchase ticket on an event > checkout.

1.2.2. Organisational Requirements

1.2.2.1. All ticketing will be done virtually to limit paper use/ waste.

1.2.3. External Requirements:

1.2.3.1. If a user is not willing to share their name and email address when purchasing a ticket, they cannot create an account.

1.2.3.2. Our system simply accepts payment and does not have access to users’ personal banking accounts.

1.3. Domain Requirements

1.3.1. The webpage is compatible with standard html web page design which is viewable in a browser.

2. User requirements (These were elaborated on above. This is a brief synopsis of each feature. See functional requirements above (Section 1.1) for more information).
    2.1. All Users:
        2.1.1. Able to create an account and log in.
        2.1.2. Access the unique features to their log in (discussed in Host, Attendee, and Administrator below).
    2.2. Host
        2.2.1. Create a specific event and enter it to be added to the website.
        2.2.2. Receives feedback if their event is not approved.
        2.2.3. Able to edit an existing event.
    2.3. Attendee
        2.3.1. Able to search for specific events.
        2.3.2. Able to purchase tickets to existing/ approved events.
        2.3.3. Is not able to purchase tickets to sold out events.
    2.4. Administrator
        2.4.1. Can accept or decline events that are submitted by verified hosts.

(70%) Status of the software implementation

- How many of your initial requirements that your team set out to deliver did you actually deliver (a checklist/table would help to summarize)?
    - For the requirements completed, please see the list provided in the previous section of this document → "The final list of requirements of the software that was built. Incorporate feedback from M2." This includes a layout of the requirements from milestone 2 that we completed and implemented in the system. The items that we were not able to implement will be listed in a section below.
- Were you able to deliver everything or are there things missing?
    - We were not able to deliver everything we aimed to deliver with our application. I think this is largely due to our group being very driven and striving to build the "perfect product" with limited knowledge and experiences, and it just simply wasn't feasible in the time frame of the semester for us to build everything we wanted to build.
- Did your initial requirements sufficiently capture the details needed for the project?
    - We do feel that our initial requirements (outlined in milestone 2) were sufficient to complete this project. We did a very good job of outlining all aspects of the project as an overview. For example, all of the requirements we indicated that each user should have were well laid out early on. These

did not change very much between milestone 2 and this milestone, which we are proud of. However, what we do feel we did miss was splitting up some of these items. There were a few tasks where we recognized afterwards that we inaccurately guessed how long that they would take to implement and how many smaller tasks it needed to be broken into. This will be discussed further in our "Reflections" below.

- If you have any requirements unimplemented, you will want to include an extra section showing those requirements have not been implemented.

Unimplemented requirements:

- This has been organised in the same manner as the completed requirements of the system was outlined above for ease of viewing.

1. System requirements
   - 1.1. Functional Requirements
     - 1.1.1. General (for the entire system)
       - 1.1.1.1. The system must keep confidential information confidential. This includes all user information on file, and any information that is needed to be shared will be approved by the user. (see next point).
       - 1.1.1.2. If a user's email, name etc is to be shared with an event for any reason, then the user must be asked to approve this before it is completed.
       - 1.1.1.3. Create a method to allow users to be able to be part of a rewards system that can give them deals on tickets OR give their hosted events a boost on the website for viewing.
       - 1.1.1.4. When an event is declined by the administrator it is to remain in the system so the host can go in and simply fix the issues, rather than have to resubmit the entire form.
       - 1.1.1.5. The system needs to validate payment details accurately, ensure security, comply with regulations, handle errors effectively, and confirm successful transactions.
     - 1.1.2. Host
       - 1.1.2.1. Allowing ability to upload music samples of artists and have artist profiles linked to events that are created.
       - 1.1.2.2. Allowing the host to determine the ticket price and having this reflected in the checkout window when someone purchases a ticket.
     - 1.1.3. Attendee

1.1.3.1. Any attendee information that is to be shared for ticket verification etc, is to be approved by the user when they create their account.

1.1.3.2. Ability to cancel a ticket. This is something that we didn't get to implement.

1.1.3.2.1. Apply cancellation policy and update DB to show this.

1.1.3.3. We planned on implementing the ability for attendees to rate an event or venue, but we didn't get to implementing this.

1.1.3.4. Ability for users to invite friends to join an event with them.

1.1.3.5. Applying the "surprise me" feature where users could get a random event selected for them by the website.

1.1.3.6. Taking into account attendees location for events, to show them events that are local to their area.

1.1.3.7. Providing an interactive helpline chat for users having concerns or questions during website use.

1.1.4. Administrator

1.1.4.1. Administrator being able to remove a user from the system if there are issues with that user.

1.2. Non-Functional Requirements

1.2.1. Product Requirements

1.2.2. Organisational Requirements

1.2.2.1. We didn't get to apply many security aspects to the system as it is something that we don't know a lot about at this time and it is an extra requirement in terms of the project.

1.2.2.2. Follow all guidelines for online security and protection of user privacy as outlined by:

1.2.2.2.1. Freedom of Information and Protection of Privacy Act (FIPPA):

1.2.2.2.1.1. https://www.bclaws.gov.bc.ca/civix/document/id/complete/statreg/96165_00

1.2.2.2.2. Personal Information Protection and Electronics Documents Act (PIPA):

1.2.2.2.2.1. https://www.bclaws.gov.bc.ca/civix/document/id/complete/statreg/03063_01

1.2.3. External Requirements:

1.2.3.1. Accessibility

1.2.3.1.1. Ideally we would have added all features to make our system as close to fully accessible as possible. We recognized early on that this would likely not be feasible in this project, but it is something we wanted to indicate that we did want to include. However, our website should be compatible with most modern screen readers.

1.2.3.1.2. Our goal would be for the system to optimise usability for all, which could be achieved by adding features like: talk to text, providing the website in many languages, voice recognition software etc. Ultimately we would want to apply everything outlined by the website content accessibility guidelines (WCAG).

1.2.3.1.2.1. https://www.w3.org/WAI/WCAG20/versions/guidelines/

1.2.3.1.3. We also wanted to have a form attached to the attendees profile where they could indicate any accessibility needs they may need at a venue. Additionally, we wanted to have a more specific venue page, where the venue could indicate what accessibility needs can be accommodated or any concerns attendees may need to know in this regard.

2. User requirements (see these explored in more detail in "functional requirements" above).
   2.1. All Users:
   2.2. Host
      2.2.1. Be able to indicate the accessibility details of the venue (wheelchair access etc) for attendees to be able to know if it is relevant to their concert experience.
   2.3. Attendee
      2.3.1. Be able to indicate any accessibility they will need at a concert and be able to discuss this with the event organiser.
   2.4. Administrator
      2.4.1. None.

- If you have a requirement that is partially working but feel that there's enough you don't want to call the whole requirement as incomplete, then break it up into two requirements, with one shown as tested and working, while the other is shown as not working.
  - We do not feel we have any partially completed features. Please see the lists above for a layout of our implemented and unimplemented features.

- How many tasks are left in the backlog?

    - 1 - “Validate the user has entered a correct payment.” As stated, due to limited experience, our team faced challenges in allocating sufficient time to implement more sophisticated methods like this.

- What are the architecture/ key components of the system? This needs to be sufficiently detailed so that the reader will have an understanding of what was built and what components were used/created.
- For our web application the architecture is as follows:
- **Main architecture:**
    - <STORAGE> **MongoDB:** We used MongoDB as our primary database.It is a noSQL database as it was recommended online as a popular choice for creating web applications due to its flexibility, scalability, and performance.
    - <DEPLOYMENT> **Docker:** We utilized Docker to containerize our application for deployment. This packages our application and its dependencies as a whole so that the application can run effectively across different environments.
    - <BACK-END> **PyMongo, Python:** Since we did our backend in Python, we utilized Pymongo, which is the Python driver for MongoDB. This simply means PyMongo allows our python code (containing the Flask commands) to interact with our MongoDB database.
    - <BACK-END> **Flask:** We used Flask as our Python web application framework. We then used a wrapper included with Pymongo (flask_pymongo) to allow us to use Flask with Pymongo/ MongoDB. This way we could utilize our flask/ pymongo interactions to interact with the database.
        - Flask also uses Jinja2 templates to render HTML content, and uses Bootstrap for styling and layout (discussed more below).
        - These commands were often structured as “render_template(‘item.html’)”. Which allowed us to pull a Jinja html template using the Flask render command.
    - <FRONT-END> **BootStrap, Pingendo:** We used bootstrap for building our front end framework as it provides pre-designed HTML and CSS templates for commonly used items (buttons etc). We then leveraged this by utilizing Pingendo to aid with creating our website design. Pingendo is an online tool that utilizes the bootstrap framework to build the HTML/ CSS pages for our website. We then modified these templates to create a consistent theme for our pages and utilize the interactive items that gather/ display information for/ from our database.

    - <u><FRONT-END> **Jinja2**</u>**:** Jinja2 is integrated within Flask, and it is an HTML template engine. For our application, Jinja2 is used to render data retrieved from / being sent to MongoDB. So it allows for this data to be gathered and stored into MongoDB and also allows for the data to be retrieved from MongoDB and be displayed on the webpage. It also allowed us to embed Python code/ expressions into our html templates as needed.
- **<u>Other Relevant Items to mention:</u>**
    - <u><VERSION CONTROL/ PROJECT MANAGEMENT> **GitHub:**</u> We used GitHub to aid with project collaboration and version control, as it is a popular platform and was recommended by the professor. This allowed our group to collaborate on a Git repository, and track our issues, versions, code reviews etc., efficiently in one place. Many of us utilized GiitHub Dashboard to manage our GitHub repository/ project, and to aid with managing our commits, pull request, issues etc. Although some group members used the terminal in their chosen IDE.
    - <u><Integrated Development Environments (IDE)> **PyCharm, Visual Studio Code:**</u> The IDEs used by the group were either Pycharm or Visual Studio Code as these allowed our group to work in Python and worked well with our system overall.
- **<u>Additional items/ packages:</u>**
    - **<u>Redis:</u>** This is an additional item that is not necessary for the application to run at a basic level. However, this is used by Flask to cache frequently accessed data, among many other functions. It is used as it enhances performance of a system as a whole. It is also said to enhance scalability and function of the application, so it was added early on, as we wanted to try to implement items that would promote a product that can be effectively maintained and scaled over time.
    - **<u>blinker</u>:** This package provides a fast and simple object-to-object and broadcast signaling mechanism for Python.
    - **<u>bson:</u>** This package provides support for encoding and decoding BSON data (Binary JSON), which is used in MongoDB.
    - **<u>click</u>**<u>:</u> This package provides a framework for building command-line interfaces in Python.
    - **<u>markupsafe</u>**<u>:</u> This package provides support for safe HTML, XML, and XHTML markup.
    - **<u>pip</u>**<u>:</u> Pip is the package installer for Python, used for installing and managing Python packages.

- What design patterns did you use in the implementation? You will need to reflect on what you planned to build vs what you built.

For this section, we only looked at the design patterns we learned in class.

For the below design patterns, we included the following:

- APPLIED: We indicate which design patterns we felt we utilised in our application design.
- REFACTORING: We indicated here what we intended to build if it wasn't implemented. Where applicable, we also tried to think about other means by which we could have refactored our code to implement these design patterns further. We felt this would be a helpful step for us to learn from during our project reflection process.

Design Patterns Utilized:

Creational

1. Singleton
    a. APPLIED: Although not traditionally applied in the manner which we learned in class, we feel that given code only uses one instance of PyMongo that is shared across our entire application, we could consider this singleton application. This is seen in our app.py file in the form of "mongo = PyMongo(app)".
    b. REFACTOR: We could have applied the singleton pattern for our database connection. We could have created a class to represent our database connection and ensure only one instance of that class is created; in the manner we learned in class. We believe we could have implemented this in a similar manner as seen in the provided course code labelled as "Singleton" in the modules page of our COSC 310 course.
2. Factory/ Abstract Factory
    a. APPLIED: We do not feel we implemented this pattern.
    b. REFACTOR: We felt that if we were to grow our program to have a more vast selection of events, we could apply the factory pattern to that. By this, we mean that we could have an "events_factory" which creates the generic event class, and it will rely on subclasses to create the specific type of event. For example we could have subclasses that make the event for: a single night event, a weekend event, a week long festival etc. Then we could use this to create

events that are more tailored for their function but all have their base functionality created via the factory method.

Structural

1. Facade
    a. APPLIED: We feel that it could be said that us utilising our aforementioned project architecture, we are applying the facade pattern. This is because Flask handles all the HTTP queries in a simple manner, Flask/BootStrap serve as a facade for underlying CSS complexities, Flask/ Jinja act as a facade for HTML responses. Additionally, Flask/PyMongo provide a simplified interface for interacting with our database, and Docker simplifies the difficulties that can come with managing the application environment.
    b. REFACTOR: Not Applicable.
2. Adapter
    a. APPLIED: Not that we can see/ that we implemented.
    b. REFACTOR: We feel a good example of where this may be applicable would be if our application had to interact with different databases.
3. Model-View-Controller (MVC)
    a. APPLIED: In our code, we felt we were able to separate the model (our MongoDB operations), the view (ie. the HTML templates), and the controllers (the Flask routes). The MongoDB operations were those utilising the Pymongo methods (such as ".insert_one({})" or ".find()"), and usually started with 'mongo.db.'. The flask routes were those labelled as '@app.route()'.
    b. REFACTOR: We felt we did a good job with this pattern.

Behavioural

1. Iterator
    a. APPLIED: We do iterate through all events in the database in our "events_display" method, however it doesn't use a specified iterator to do this. So it technically may not be considered the iterator pattern.
    b. REFACTOR: We could have added an Events iterator for the above mentioned method, however, this feels unnecessary.
2. Mediator

    a. APPLIED: We feel some of our code loosely resembles the mediator pattern. These include the Flask application as it is acting as a mediator between different components of our web application. We appreciate this could be considered an "intermediate" and not a true "mediator".
    b. REFACTOR: Not applicable.
3. <u>Observer</u>
    a. APPLIED: We feel that the Flask routes we utilised are an example of the Observer pattern as they listen for HTTP requests and trigger the indicated responses/ update the appropriate objects upon that request being made.
    b. REFACTOR: We felt that we could have somehow applied the observer pattern within the administrator review process and with event edits. This is because when a host submits an event, they need to wait for administrative approval before the event is posted. We would then apply an observer to the event so that when the administrator makes their decision, the host who created the event is updated of this change. We planned to add in an observer for when events are edited. This is because the event will have a list of attendees that would need to be updated when the event is edited. Thus, we wanted to make this occur so that when a host edited an event, the attendees of that event were notified of the change.
4. <u>State</u>
    a. APPLIED: Not that we can see.
    b. REFACTOR: We planned to apply this pattern in 2 places:
        i. We wanted to utilise this for the events to indicate if the event is in the "editing", "review", or "verified" states. We did have this implemented in our skeleton Python code, but our program functionality was not at a level where it made sense for us to add this into our Flask routes,
        ii. For the user states of logged in, logged out and this would differ for the type of user (host, attendee, administrator). However, as mentioned, we were unable to apply this method in our current project iteration.

- What degree and level of re-use was the team able to achieve and why?

- From our application, we don't have extensive re-use, through methods such as inheritance and abstraction, at this time. We were focussed on first getting the application working and understanding the process, and then we were planning to refactor appropriately if time permitted. This refactoring would have added items that would have supplemented more re-use into our code.
- However, we do have some degree of what we feel could be considered reuse within our code:
  - Our html templates are rendered within many of our routes to allow for consistency within our page designs. We utilised the same html pages where necessary to limit redundancy, duplicate code and conflicting html pages.
    - This also allowed us to have consistent structuring within our routes as well. This made creating and modifying routes easy.
  - Our database operations are utilised in a consistent pattern across multiple routes which reuses a similar pattern. This makes the code easy to follow and maintain and limits duplication of paths.
  - There is conditional logic in several of the routes (for example the login() and register() routes) which allow for the reuse of code paths, instead of creating duplicate code.

- Indicate any known bugs in the software. Explain how these bugs can be fixed.
- BUG 1: The home, profile, my events, event approval and my ticket pages don't have the main image in the background. This was attempted to be implemented but broke the html following it.
  - BUG 1 FIX: Re-create the structure of the pages in the html to properly have the main image in the background.

(15%) A step-by-step guide for handing over the project that includes:

- Installation details needed to deploy the project/ Dependencies needed to deploy the project.
- For project hand over, we will walk through the aforementioned system architecture to indicate what the next team will need to run the project:
  a. GitHub & Git
    - Download latest version of Git
    - GitHub Desktop OR use terminal inside your IDE can be used to handle git commands (push, pull, commits etc).

        - **GitHub repo link:** https://github.com/COSC310-Team-Passio/PassIO-COSC310-EventTicketProject.git

   b. Docker
      - Install Docker from the Docker website.
      - Build the Docker container through running the docker-compose file

   c. Python IDE
      - Our group used either PyCharm (via JetBrains) or Visual Studio code (Visual Studio website download) as our Python IDEs.
      - Once you have one of these downloaded, clone the PassIO Project repository on your local machine.
      - Using the terminal go to where you cloned the project and enter 'docker-compose up –build'. This will build the project in docker with everything necessary to run it.
      - When doing this, you will need to make sure docker is running, which you can see in the Docker application.

   d. Running the application:
      - We have the application running on:
         - localhost:5001/
         - This port is defined in the docker-compose.yml file.
         - There is also a link that will appear in your terminal that will take you to the 5001 port.

- Any maintenance issues required to run the project (e.g., account information)

   a. Account Information
      - We have generic admin and host accounts set up that you can use to test the features:
      - Email: host@host.com

        Password: 123

        Email: admin@admin.com

        Password: 123

        Email: user@user.com

        Password: 123

(3%) Reflections (Comment on the following items as a team):

1. How did your project management work for the team? What was the hardest thing, and what would you do the same/differently the next time you plan to complete a project like this?

- We found that our project management worked well because we all fell into our own roles. We tried to implement the "people not process" mentality of task delegation. We tried to play to the strengths of our group, not in a Golden Hammer sort of way, but more as an optimization of time method. We would pair group members together on a task if time allowed to allow for students who knew more about a topic to be able to help other group members who weren't as fluent in that portion of the project.
- The hardest thing was simply how much work was asked of us in a week, with full course loads and work, we sometimes had a hard time to get completed what was asked of us. Next time we do a project like this we would probably build the architecture together at the start, as we found it a bit tricky to get everyone on the same page in the beginning. We would also likely be able to create better timelines for ourselves for time management. Given we haven't done a lot of these tasks before, we feel we sometimes would overestimate how long some tasks would take and then unexpected hiccups would have us underestimating how long another portion would take.
- We would probably keep a similar workflow if we did it again though. We feel that we stayed on track well overall and we were able to communicate quite well throughout the entire process with how often our meetings occurred and via our online communications (Discord and GitHub).
- Following issues made on github and the structure we laid out helped move us in the correct direction. Next time it will be important to have more a more rigorous follow up protocol that allows the other members to know exactly what needs to be done if a member isn't able to complete a portion of the project in the projected time frame.

2. Do you feel that your initial requirements were sufficiently detailed for this project? Which requirements did you miss or overlook?

- We are actually quite happy with our initial requirement details. We feel that given we spent a lot of time laying out these requirements, we did grasp the breadth of the project requirements needed.
- What we did really notice is at the start of the project we overestimated how much we would be able to apply to our project. We tried to focus on creating a good framework to optimise scalability, maintainability, and adaptability, and we were thinking very big picture. However, in the end we were able to get as far as meeting the minimum requirements with a bit of added functionality, and we were unfortunately unable to add many of our higher level features.

- A key thing to mention, which was alluded to above, is that we should have broken down some of our requirements more effectively when it came to task/ user story/ use case creation. This simply came with the territory given we are new to much of this framework and processes. However, this is what often lead to certain tasks taking longer than they probably should have.

3. What did you miss in your initial planning for the project (beyond just the requirements)?

- As previously mentioned, knowing what we know now, we should have broken our tasks down further. We also got more fluent with utilising GitHub as the project went on, so this would have been very helpful with this at the start of the project.
- We really set out to create the best application, but we could have aimed to create a functional app, as we tried to implement some quite "fancy" features at the beginning that we had to simplify later on.
- We should have met as a group and gone over the basics of implementation and the system architecture as we realised halfway through the project that we were not as much on the same page as we thought we were. This was largely due to us being quite unfamiliar with a lot of the architecture.

4. Would you (as a team) deal with testing differently in the future?

- Probably. As expected, since we were spending so much time simply getting the application working, we didn't get to do structured testing as much as we wanted to until the end of the project. This is simply because we didn't want to spend hours figuring out how to test a portion of the project until we knew that portion was going to work. Again, simply another item that comes with the territory of trialling a new venture.
- What we would do differently is utilise test driven design since we now have a better understanding of how the architecture works together. This way we could be more fluent with our testing and have a battery of tests that can be implemented autonomously.

5. If you were to estimate the efforts required for this project again, what would you consider? (Really I am asking the team to reflect on the difference between what you thought it would take to complete the project vs what it actually took to deliver it).

- As is often the case, we felt it was more work than expected. We knew it was going to be a big task to complete the entirety of the system as we were all unfamiliar with part (or, in some cases, all) of the system. However, there was a steeper learning curve to some aspects of the system than others. For many of us, this was mainly understanding how much of the work Flask can do for us, how this integrates with html code/ the database, and then how and what to test once we did get it working. We think we did a pretty good job of getting the tasks we set out to do done, but we all agreed that creating good tests took much more time than we thought it would.

- We suppose this can be summarised to: everything seemingly took a bit (or quite a bit) longer than anticipated. Especially when something didn't go according to plan and you had to backtrack to correct it. This really ate up a lot of time for us at different timestamps along the semester. Next time we do a project, ideally we would make less of these mistakes, but we would likely also estimate task timelines to be slightly longer.

6. What did your team do that you feel is unique or something that the team is especially proud of (was there a big learning moment that the team had in terms of gaining knowledge of a new concept/process that was implemented).

- We felt that utilising a noSQL database was somewhat unique as this is something none of us had tackled before or had been introduced to in other classes. As most other coursework focussed on SQL databases.
- We are proud of how much we were able to teach ourselves. Most of our project architecture is foreign territory for us, and much of it we have never been taught in a course. Many of us did not know how to work with html code before and now we feel we have a functional knowledge of this. Utilising Flask was a beneficial process as it is amazing how it streamlines the application creation. This could be said for many of our architecture components. Items such as Docker, PyMongo, Bootstrap, Pingengo are all items we had either lightly heard of or never heard of and it is a good feeling to look back and feel that we can see how they are applied. We feel this is beneficial as it also allows us to begin seeing the tools that are available to us as we think about what we want to create in our other projects, careers etc.
- It was interesting to learn and think about how the software engineering framework is applied. The design principles and patterns were interesting to see and think about how these are applied. It is easy to see how these can be beneficial and also to see what mistakes we have been making in our code. The software processes and models were interesting as they give you terminology and frameworks to follow. Sometimes these frameworks are in areas where it seems like you don't need to worry about it, but as you work further on a project you can see how this can result in technical debt and code smells, amongst other items that could negatively impact a team.