FINAL REPORT #SENSINGGANG

Video

https://drive.google.com/uc?id=1bNXSIaLGYqi31bLwPJq-mQGYQvtF3ix7&export=download

Testing sheet

https://docs.google.com/spreadsheets/d/1_0GSVH6_xUYvIhY4pgwSaa_UJ0NYOiRrHFRWhBNGK00/edit#gid=0

General Development:

1. **What did your team build? Is it feature complete and running?**
   Our IoT Sensor Monitoring system is an application that allows users to take advantage of the IoT and the MQTT network in order to monitor sensors in BC Parks without the need for additional set up beyond account creation and the subscription to their data-streams of choice. This application takes on the roles of the web-broker and the client in the MQTT protocol entities, thereby simplifying the data acquisition process for the user. Our system will be subscribed to the sensors set up in BC Parks, the data from which will be stored in our database. The user can choose to view data from specific sensors from within that set, and can make choices on how that data is represented and set thresholds on the data streams which, if exceeded, will send them a notification. Our system will be data agnostic, meaning it will be able to support most data formats, allowing users to select from a wide variety of sensors and to have that data represented in ways relevant to the user. That customizability is further expanded by the choices the user can make upon selecting which data to view. In other words, the data type and the parameters set by the user will determine the graphical representation and range of that data. For example, the user can wish to have their sensor data displayed in various chart forms such as bar, line, radar, and pie. Users will be able to create an account, set which data streams are of interest and how they want that data to be displayed in its “widget” on their home page. After the account is created, users will be able to modify the widgets of their home page, both in the way the data is represented and by removing or adding data streams.
   Our system is functional and running, however some concessions have been made in terms of feature implementation, as adding some of the functionality did not make sense for us considering the scope of the project. Nonetheless, we have designed our application to be scalable in such a way that those changes could easily be implemented. For example, we have omitted adding and querying from an external relational database such as MySQL, as the built-in Django database SQLite was sufficient in order to demonstrate the more important functionality of the project. Likewise, we have not implemented testing coverage due to our repository’s permissions, and our UI is also not quite production ready, although it is more than sufficiently developed for user testing and a sub-optimal but viable customer experience. There are also a few scalability limitations in terms of the amount of sensor streams available, however we did not feel this was necessary for our scope. It was not realistic for us to set up or gather real data from IoT devices in BC Parks. That being said, the essentials, such as the account creation and management, subscription to data streams, publishing data, and the graphical representations are perfectly functional.
2. **How many of your initial requirements that your team set out to deliver did you actually deliver (a checklist/table would help to summarize)? Were you able to deliver everything or are there things missing? Did your initial requirements sufficiently capture the details needed for the project?**

- *Data collection - collect data from IoT devices* ✅
- *Notification/Alerts - alert users when thresholds are exceeded or encountered* ❌ We did not implement this feature as the main features were prioritized and due to core constraints
- *Process data - provide meaningful results from collected data* ✅
    - *Display data (graphs, etc)* ✅
    - *Export data in relevant formats (csv, JSON, etc)* ✅
- *Store Data- store the collected data in a database* ✅ We are using the built in Django database (SQLite) for simplicity, however if we were to take this to production, we would consider using another database (MySQL, MongoDB) for scalability
- *Remote Monitoring - User can choose which sensors they wish to monitor* ✅
- *Requires account to view data* ✅
- *Secure Data - Ensure privacy/security of data collected from sensors* ✅
    - *Data is collected, handled, and stored in a secure and private manner*
- *Usability of System - Implement user interface that is intuitive and easy to use/navigate by the user* ✅
    - *user interface is similar to that of other common UIs*
- *Maintainability of the system - System should be easy to maintain and update over time to ensure that it remains secure against new threats and vulnerabilities. (Security)* ✅
    - *Modularity - Components of system can be easily upgraded or replaced without affecting the whole system; thus, allowing for easy updates and maintenance.*
    - Cross-Site Requestt Forgery (CSRF) attack prevention through Django's built in protection against CSRF attacks through generating tokens for each form submission and validating/checking them on the server side
    - Django's built in user authentication that handles user login, logout, and password management.
- *Performance/Efficiency - System should be able to withstand high traffic (data volume) and perform to its best capabilities* ✅
- *Reliable - Consistence performance according to its specifications with little to no issues and failures* ✅
- *The application should allow them to access data without needing an advanced understanding of the technology as would traditionally be required to access data from the IoT. Application allows users to take advantage of the IoT and the MQTT network in order to monitor sensors in BC Parks without the need for additional set up beyond account creation and the subscription to their data-streams of choice.* ✅

3. **What is the architecture of the system? What are the key components? What design patterns did you use in the implementation? This needs to be sufficiently detailed so that the reader will have an understanding of what was built and what components were used/created. You will need to reflect on what you planned to build vs what you built.**
   The architecture of our system was largely dictated by the Django framework and our use of the MQTT protocol. Our system follows the MVC (Model View Controller) design pattern, partially due to the

architecture required by the Django framework. The user interface, the control logic and the data are all distinct components, ensuring clarity during development. Our main components are the UI, which was developed using custom Django Templates and stylesheets, the MQTT functionality, which resides in the views python files, the graphical representations, and the account creation and login. Per Django convention, each of these features are contained within their respective “apps” (ergo, modules) that strike a balance between the single responsibility principle (SRP) and code reuse, so while the module is responsible for a single feature, there is, for example, inheritance on every page from the master template and stylesheet. Our system functions with event-based control, so the user dictates the system’s behaviour, and when appropriate, each component reacts in some way to user input. An exception to this is the data stream from IoT devices, which in a production version would be continuous, although in the system’s current form, data acquisition is also performed upon the user’s request. Our system currently runs off of Docker, thereby making it a client-server architecture, however, a production version would employ a three tier architecture for data storage scalability. A production version would also employ the Adapter pattern for the potential of increasing the IoT device diversity, allowing our system to take in a larger variety of data.

4. **What degree and level of re-use was the team able to achieve and why?**
   We have employed a high degree of reuse, especially considering the ultra-specific nature of our project’s individual features. For example, each of our applications (pages) extend a master template with a navbar and a master stylesheet. We are also using multiple APIs, such as Paho MQTT, ChartJS, Django Authorization Library, Django Test Library, as well as the Django Framework. There is also some internal code reuse.

5. **How many tasks are left in the backlog?**
   Most of the tasks left in our backlog are a result of design decisions concerning scalability that we made in order to fulfill the project requirements. We prioritized simplicity and essential functionality with the understanding that were this product to be taken to production, changes could easily be made, thereby clearing the backlog of most of these features and ensuring the easy delivery of a production ready version. Among these are database choice, method of populating the homePage grid and the subscription page functionality. There were also features we didn’t have time to get to and were moved to the backlog, such as the notifications/alerts.

CI/CD:

1. **What testing strategies did you implement? Comment on their degree of automation and the tools used. Would you (as a team) deal with testing differently in the future? Make sure to ensure that your testing report is updated to reflect what's actually been done.**

   We used a Test Driven Development strategy where, wherever possible, we wrote tests first to ensure the subsequent code would fit the intended requirements. This came in the form of unit tests for each component, however, many of these created mock Client objects, tested page flow/redirection, as well as ensuring API functionality and connectivity. These were all put in place to cover the necessary integration tests for the system. As per our CI/CD pipeline, we utilized GitHub actions to automatically run all of our tests when pushing to our development branch, as well as running all tests when we completed a final merge to the production branch. We would’ve like to implement coverage testing, however popular tools such as codecov.io required special permissions from the COSC310 repository that we were not able to give. This was unfortunate, however it forced us to be rigorous in checking that every component was tested precisely. Our full test plan can be found on our excel spreadsheet as well as in our previous milestone documents.

2. **How did your branching workflow work for the team? Were you successful in properly reviewing the code before merging as a team?**

   The branching workflow was very effective for us. The process of creating our own branches allowed us to work on separate features more efficiently and often provided a more seamless integration process. There were initially some struggles with correctly setting up our .gitignore files as Django and python virtual environments would track a lot of unwanted dependencies. Once we ignored unwanted dependencies and each set up our own local virtual environments, the branching workflow process became a lot smoother. The initial integration of our system components involved merging the login/create account branch with the home page. Then, separately, the graph and MQTT API branches were merged and integrated. Finally, that component was merged with the new homepage. We tried to avoid stove pipe anti pattern during this process, and although we improved this was one of the issues we ran into a few times. Incrementally these branches would be merged to the CI branch, which is our active development branch. This branch has our CI setup through GitHub actions to ensure our unit and integration tests ran. Every significant commit made by us was followed by a created pull request that would require a comment and an inspection/approval by a team member. We found it helpful to do this during our meetings as having a few eyes to look over the code/ resolve merge conflicts was helpful. These pull requests were linked with our project board issues and updated accordingly by one of the team members.

3. **How would your project be deployed? Is it docker ready and tested? Provide a brief description of the level of dockerization you have implemented and what would be required to deploy.**

   We integrated local level of Dockerization for our application. It is deployable on localhost and contains all dependencies when built. This is part of our CI/CD pipeline where any pushes to the development branch runs automatic tests using GitHub actions. If these tests all pass, the branch is updated and is deployed using a docker container. To do this on ones local machine, they build a docker image and then run the container. This can be accessed by navigating to http://localhost:8000. Further down the line, this could be improved to use an Apache or similar server for deployment, however we felt this was out of scope for the project.

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?**

   Project management started with using GitHub issues as well as our collaborative Notion document. We wrote down our sprint cycles and each assigned certain issues to each of those cycles. We tracked/updated our process in our documents. About a third of the way through our project, we discovered GitHub Projects. This was really easy to setup with our issues already being on the repo. We have found this to be an excellent tool for project management. We assign each issue to a person, set priority levels, size of tasks, and link our pull requests to those specific issues. This has been beneficial as we have felt a lot more organized and on track with our sprint cycles. It is also great as tasks that cannot be finished immediately can be moved to the backlog section instead of just being forgotten about. This allows us to reflect on our shortcomings and go back to aspects that have been missed if we have extra time. Before this, it was a little tricky to stay organized and hold accountability for what we each completed. If we were to do this project again, we would 100% immediately start with using GitHub projects. To be able to track the size and priority

of tasks as well as having a tool to visualize and modify a Kanban type board would have improved our initial workflow.

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

   Not particularly. We thought we had covered most corners, however upon receiving poor feedback for our milestone 2, we chose to do some significant revamps for that milestone. This included taking a closer look at our system architecture, the publish subscribe model, as well as how MQTT works in the real world. We also found it difficult to understand the initial requirements of this project in general, so I think this also contributed to our confusion. We never received feedback or a regrade on our milestone 2 rework, however we felt after that we had a better understanding of the requirements of our system as a whole.

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

   I feel that we could have planned our Kanban board better by using GitHub projects from the start. The combination of taking notes on what we had done along with GitHub issues worked fine but to have everything all in one place would have been an improvement. Django was new to us all, and I feel more work could have been put in as a complete group at the start to get to know the framework better as well as decide on the ways we want to set up environments, .gitignore files, as well as templates and redirects. Of course this is hard to do as we were all inexperienced using this framework, however this could have saved us some time in the long run as we fell into the stove pipe anti-pattern at times.

4. **What process did you use (ie Scrum, Kanban..), how was it managed, and what was observed?**

   We followed a Kanban Agile methodology using GitHub projects as our main tool for this. We also set out sprint cycles which correlated to the issues we created for our project board. We all worked on incrementally adding to the project in small chunks, where we also modified the priorities of certain tasks as their importance fluctuated. Managing our process involved actively updating our Projects page as well as having 2-3 meetings per week to check progress, make new deadlines, as well as reflect on what could be improved in the next phases of development. These meetings also required us to revisit requirements for the project regularly so we would keep on track with our scope. We observed improved collaboration and communication, as well as accountability for getting work done on time. We all have struggled with groups in the past with poor communication, and we all feel we were able to communicate very effectively which allowed us to achieve most of our goals.

5. **As a team, did you encounter issues with different team members developing with different IDEs? In the future, would the team change anything in regard to the uniformity of development environments?**

   This was not much of an issue for us as we all use VSCode. Initially, we attempted to use GitHub Codespaces, and although it seems like a nice tool, it caused quite a few headaches for us. We were all brand new to Django, and trying to configure this at the same time as getting Codespaces configured was quite difficult. We thought it would be best to just use VSCode locally to remove that layer of complexity. We would not rush to change this in the future as our IDE choices did not cause any issues for us.

6. **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).**

Overall, the project timeline followed our expectations. However, this is considering the fact that we implemented large buffers during the planning phase in anticipation of the challenge of using a new framework and the MQTT protocol for the first time. The biggest hurdle for us was definitely the adjustment to Django. We figured because we all had some Python experience this would be a logical and “fun” choice, however the learning curve shocked us all. For most of us the first few long days of this project involved many hours of setup, configuration, figuring out urls, etc. This was a pain but we feel we all learned a lot and now are much more proficient with this very powerful framework. Another thing that we didn’t realize that took as much effort as it did was setting up a good quality project board and being diligent with updating it. The effort put into this payed off, and so after our project board was configured the improved workflow left us with a better workflow. Using it from the beginning would’ve been hugely advantageous. Another aspect that took a lot more time than we imagined was dealing with the integration of certain parts of our project. Merge conflicts were common due to stove pipe anti-patterns as well as incompatible redirects and pages. This definitely slowed down the delivery process, and a more diligent approach to the project partitioning would streamline things in the future.

7. **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).**

   Initially, we had planned to do the discord clone. While this sounded fun, a clone is inherently unoriginal and the safest choice. We switched to the IoT device application, which, paired with our framework of choice, Django, meant we were developing a system with a tool that was new to us in a domain of computer science we hadn’t even heard about. These were big risks, but we felt the rewards and opportunity for personal growth were greater, and the fact we stuck it out and were able to create a functioning system with the functionality we had envisioned has made us proud. Using Django, ChartJS, as well as the paho-mqtt API proved challenging at times however the new knowledge gained for all of us was invaluable.