EC544 Final Report: Remote Locker

Professor Babak Kia Montazam

Team Members:
Muhammed Abdalla: muhabda@bu.edu
Ahmed Benmensour: ahbenmen@bu.edu
Haidar Lafta: hlafta@bu.edu

Clarity & organization:

We used a variety of programming languages and technologies to assemble our remote locker project. These included Javascript, C, Docker, Python, AWS EC2, and the ESP32 microcontroller.

Javascript was used to build the front-end user interface, which allows users to interact with the system and reserve lockers remotely. We used C to program the ESP32 microcontroller, which controls the lockers themselves and reservations. Docker was used to package and deploy our application to AWS EC2 instances. This allowed us to easily deploy and manage our application across multiple instances, ensuring that it was scalable and reliable. Python was used for server-side programming and data analysis. We used it to develop our back-end reservation system, which includes a FIFO queue for processing reservations in the order they're received, as well as a system for detecting and managing overlapping reservations. Finally, AWS EC2 provided us with the compute capacity we needed to run our application in the cloud. We were able to deploy our application to multiple EC2 instances, ensuring that it was always available to our users.

Furthermore, we learned about different API points that were crucial in helping us build an application that can communicate with other systems or services. Through this process, we gained an understanding of the different types of APIs, such as RESTful and SOAP, and how to use them to exchange data between systems. We also learned how to work with API documentation to understand the various endpoints and methods available.

In addition to this, we were able to sketch out the architecture of a centralized server model using AWS. This involved understanding the different components of the AWS infrastructure, such as EC2 instances, S3 buckets, and IAM roles, and how to configure them to create a scalable and secure application. By leveraging AWS, we were able to build a robust and reliable system that could handle a large number of concurrent users and requests.

Discussion of failure:

As part of our remote locker project, we've built a UI that looks great and functions as expected. However, we've run into a problem with the back end of our application. Specifically, the user is not able to reserve a locker because the server is not registered with the user's input. This is a significant issue that severely impacts the usability of our application. Without the ability to reserve lockers, users will be unable to make use of the primary function of our application, which is to provide remote access to lockers. This leads to frustration and disappointment among users, potentially causing them to abandon our application altogether.

Another issue with our remote locker project is that the device is not properly connected to Amazon Web Services (AWS), and commands from the Docker client are not executing on the device as intended. This failure has significantly hindered the functionality of our project, as the ability to remotely control and manage the lockers is a critical aspect of the project's success. Without proper communication and command execution between the device and AWS, users would not be able to remotely access the lockers, and the system would be rendered useless.

If the UI is incomplete, users may encounter issues with their login credentials or may not be able to properly secure their lockers. Similarly, if the device is not properly connected to AWS, it may be vulnerable to hacking attempts or other security breaches. Ensuring the security of our application and users' data is critical, and we must take every precaution to minimize the risk of security breaches.

Technical report of success:

Our remote locker project has made significant progress in recent weeks. One of the most notable successes we've achieved is the successful communication between the client and AWS server.

Specifically, we've been able to send input from the client to the AWS server, and establish a connection between the two devices. This success demonstrates that the basic communication infrastructure is functioning as intended, and sets the stage for more advanced functionality in the future.

Overall, we believe that this success is a significant milestone in the development of our remote locker project, and provides us with a solid foundation upon which to build more advanced functionality. With continued hard work and dedication, we're confident that we can overcome any remaining challenges and deliver a highly functional and user-friendly remote locker system.

One of our most significant successes has been our ability to create reservations through a FIFO queue and detect collisions and overlapping between reservations. This achievement has been critical in establishing the core functionality of our project. By implementing a FIFO queue system to manage reservations, we've been able to ensure that requests are processed in the order they're received. This has helped us to avoid any potential conflicts or collisions between multiple users trying to reserve the same locker at the same time.

Additionally, we've developed a system for detecting and managing overlapping reservations. This ensures that users aren't accidentally assigned to lockers that have already been reserved by someone else during their selected time slot.