1. Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).
    - We have not made any big changes since the proposal in stage 1. The only thing we ended up removing was class restrictions on items. We only placed College restrictions on items. For the most part our final project is still the same: a way for students to reserve resources.
    - We intended to implement a deployment on a Kubernetes cluster through Sky Pilot, although as the deadline drew near, it was more cost and time efficient to set up a public-facing application load balancer linked to our domain (using Route 53) in front of our containerized application hosted on an EC2 instance with a static IP.
2. Discuss what you think your application achieved or failed to achieve regarding its usefulness.
    a. Our application's primary goal was to provide utility for users to check-in and check-out items, as well as view available items, and from the admin side, perform operations in case something needs to be manually overridden. We implemented every single one of these features, as has achieved its desired goal. In terms of its usefulness, or better put, utility, is exemplified through the features of our tables which all provide search, multi-element filtering, and sorting functionality to make the desired goal for our users seamless. We even implemented a geolocation feature which allows the user to view how distanced they are from a certain building, to further ease convenience and thus utility. As such, we have succeeded in providing a clear usefulness to our application in the features that seek to aid our engaged users.
3. Discuss if you change the schema or source of the data for your application
    - Our schema did not change, and we still ended up generating our data although we decreased the volume.
4. Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?
    - We implemented our database as we had modeled it in our UML diagram.
5. Discuss what functionalities you added or removed. Why?
    - The only functionality we removed was creating restrictions based off of class enrollment. We felt that this use case would have been very niche and would have just added more complexity to our database.
6. Explain how you think your advanced database programs complement your application.
    - Indexes: Indexes did not complement our application well, as most of the attributes used for JOINS and other clauses (WHERE, GROUP BY etc) were all either primary keys or foreign keys.
    - Triggers: Triggers complemented our application well, especially when we developed CRUD functionalities. In the case of inserting, deleting, updating a record in either the Inventory relation or the Computer relation, we had to make complementary updates to the other table. In doing so, triggers allowed our application to handle such scenarios aptly.

    - Stored Procedures: We employed numerous stored procedures in our application's backend. Instead of having query statements in our code, using stored procedures helped us improve the readability of our backend code. Also, we were able to dynamically pass parameters to our queries without having to directly edit query strings. Furthermore, being able to create temporary tables within stored procedures allowed our complex queries to be more organized and efficient, without having multiple subqueries within one single query.
    - Transactions: The transactions we came up with to meet project requirements (containing at least one advanced query) were not necessarily useful for the application.
7. Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.
    - Cristian: One challenge I encountered was learning how to use react and fastapi. Web dev was new to me so having to also learn how to make the frontend work with the backend was a bit confusing. I would say that it is good to learn but if a future team wanted to make development a bit easier, they could make a single page application.
    - Leo: I also had difficulties learning React and Fastapi. With no experience in web development as well, I had some trouble learning how the frontend code worked.
    - Elliot: I had little to no experience with frontend development, so understanding and applying the concepts in react (like Hooks, asynchronous functions, creating and tying java script modules and components) was difficult. It was a style of programming that I was not used to.
    - Luke: I suffered difficulties with deployment. For example, there were several issues requiring manual node package installment within the containerized application, issues with increasing compute that required me to upgrade the EC2 instance several times. In addition there were unclear instructions through NameCheap (where the domain was purchased) regarding the SSL certificate, including self-generating a PEM key + a CRT, that was not as detailed as I would have liked. Lastly, rerouting through Route 53 to allow the user to not have to specify port 3000 to access the frontend was more trouble than I had anticipated due to issues setting up a reverse proxy and not fully understanding the different types of records.
8. Are there other things that changed comparing the final application with the original proposal?
    a. Our final application had no changes compared to the original proposal other than removing the class based restrictions.
9. Describe future work that you think, other than the interface, that the application can improve on
    - The application has several implementations that could help benefit the user experience. The first thing is the ability to create future reservations within a designated time slot, rather than just a simple check-in feature that began immediately. In terms of constraints, we could add class restrictions and certain

policies (e.g. no more than X reservations, can't make back to back future reservations, etc.). Lastly, when creating a reservation, we could benefit from providing more item information to the corresponding reservation. In terms of security, the hashing of passwords in our database as well as upgrading from our current practice from long-lived JWT bearer tokens. In terms of deployment, we could benefit from a more optimized build including using a multi-layered image and optimized node builds, as we had to upgrade to a T3 large instance, as our app grew in node package bloat.

10. Describe the final division of labor and how well you managed teamwork.

We met weekly in-person for a few hours which helped ensure we maintained steady progress proportioned evenly amongst all members. We communicated through Discord, clearly communicating any problems that we were encountering. The final division of labor was distributed quite evenly.

- Cristian was primarily responsible for the backend routes and integrating them with the stored procedures, transactions, triggers he created. He was also responsible for the generation of a lot of our user data through notebooks and the creation of many of the tables. He was the primary creator and reviewer of our complex queries
- Elliot was primarily responsible for driving the meetings, often ensuring we met project requirements and communicated issues to our respective TA. In terms of development, Elliot was known for developing the creative component on the frontend, revision of backend routes, development of some frontend components, and creation of many of the tables, stored procedures, and triggers.
- Leo was responsible for many of the backend routes and a lot of the data generation process. He created many stored procedures and queries, notably for Reservations, which performed all four of the CRUD operations, allowing us to complete that portion of the project.
- Luke was primarily responsible for system design, frontend development, and deployment. He was the primary creator for frontend functionality and integrating this with the backend. He set up role-based auth, the asynchronous driver with several queries, creating routes for the stored procedures, and made sure the app was containerized and deployable.