# <<Team Bioluminescence >>

# Purpose

The purpose of this document is to provide evidence for the different roles of the team members.

# Introduction

This project we worked on was an inventory management system for a made-up corporation. It involved hosting both the front-end code and backend lambda functions with the relational database on the AWS platform. Our work was split into two sub-teams, which consisted of Miles & Jonathan on the back-end, and Justin & Humberto on the front end. The backend work involved developing a MySQL relational database on the AWS RDS service. We based most of our tables on the UML analysis we created for our project and linked the necessary primary and foreign keys. Our back-end also included developing lambda functions that would carry out the functionality of the use cases and handle incoming requests from the front end. It was a middleman between the front-end requests and the data stored in the database. For the lambdas, we had to set up the API-Gateways, which were the endpoints where the front end would send their requests and the Gateway would then route it to the correct lambda and return the necessary response.

For the front end, it mostly involved developing react webpages in VSCode using node.js and a React component library called Material UI, which helps fast-track the front-end GUI development by providing a comprehensive collection of prebuilt components that are ready for use in production. On the front end, some of the work also involved using built-in functions like fetch() which handled the requests out to the back-end, and using the built-in Window.localStorage() function to handle storing the corporate/manager log-in credentials.

# Team organization, members, and responsibilities

Team Bioluminescence met 2-3 days a week over the course of the semester to plan, review and carry out each project’s iteration. Frequent meetings allowed the team to constantly be updated on each group’s progress and what needed to be done. The frequency of our meetings also allowed the team to coordinate and clear up any misunderstandings or issues with the required tools for the class, like AWS or React. Most of the teams' meetings were carried out over Zoom, with a few in-person meetings carried out on campus when all the members were available. The team was divided into two groups, front-end and back-end to carry out the project iteration goals with more speed. This decision was also made to allow all members of the team to dedicate more time to understanding their focus areas. By keeping two developers to front-end/back-end, each team was able to dedicate more time to understanding the depth of React/AWS, thus resulting in a more polished product.

Responsibilities:

* Justin Cabral – Front-End Lead. Webpage UI Designer. API connection developer. Use case tester.
* Jonathan Duran – Back-End Developer. AWS Lambda functions backend. Test case writer. Database manager. Meeting organizer. Trello Organizer
* Miles Gregg – Back-End Developer. AWS Lambda functions backend. Test case writer
* Humberto Martinez – Front-End Developer. Use case tester.

# Process

Going through each iteration of the group project our group initially met to produce a plan on Trello on which tasks each group member should complete. Trello was essential to see the current progress of the group project to see where we are falling behind at. Each member was assigned a certain number of tasks at each meeting to complete by the next meeting. The best technique our team used was swarm programming, used to ensure that the functionality of certain tasks work correctly before being deployed.

All code for the front-end website was thoroughly tested for each use case before being pushed to its own separate branch on GitHub. We had the main branch protected and were only allowed to submit our code to separate branches to avoid conflicting mergers with the main branch. Only pull requests could be made to the main branch. This allowed us to control the quality and the working aspect of our code base.

Before finishing each lambda function on the backend each one was tested thoroughly with different inputs and edge cases before being activated on API Gateway and then getting used on the front-end. We tried several ways to make our lambda functions fail so we could fix it and make it not fail in the future.

The main communication tools that our team used were Zoom, meeting in person, and Snapchat to communicate ideas, thoughts, and problems to other team members. This allowed each team member to always be in the loop with any problems that we are currently facing.

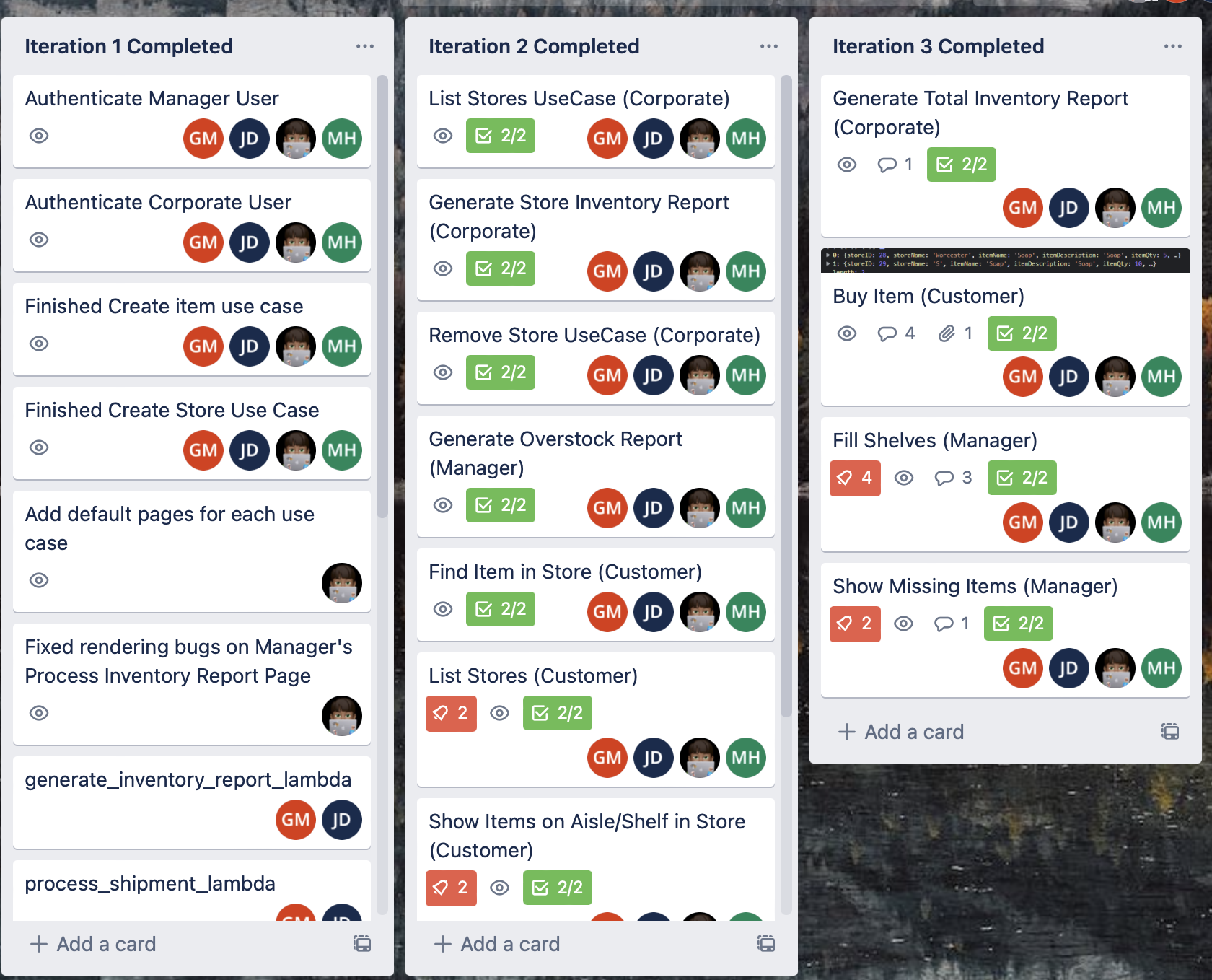
# Tools

For the front-end development, the tools used were GitHub, Git, VSCode, Discord, MaterialUI, Zoom, chromium-based web browsers, and Mozilla Firefox. GitHub and Git were used to keep source control of the project and a working repository. VSCode was the main IDE used to program, and Chrome was used as the development and testing browser due to its popularity and wide support. MaterialUI was chosen to add premade UI elements to aid in front-end development, as well as for its clean and functional appearance. Zoom was the main collaboration tool used, as it was essential for us to create video/voice calls. All the tools used in the front-end worked very well, and there wasn’t something that we tried to use that didn’t end up working for us.

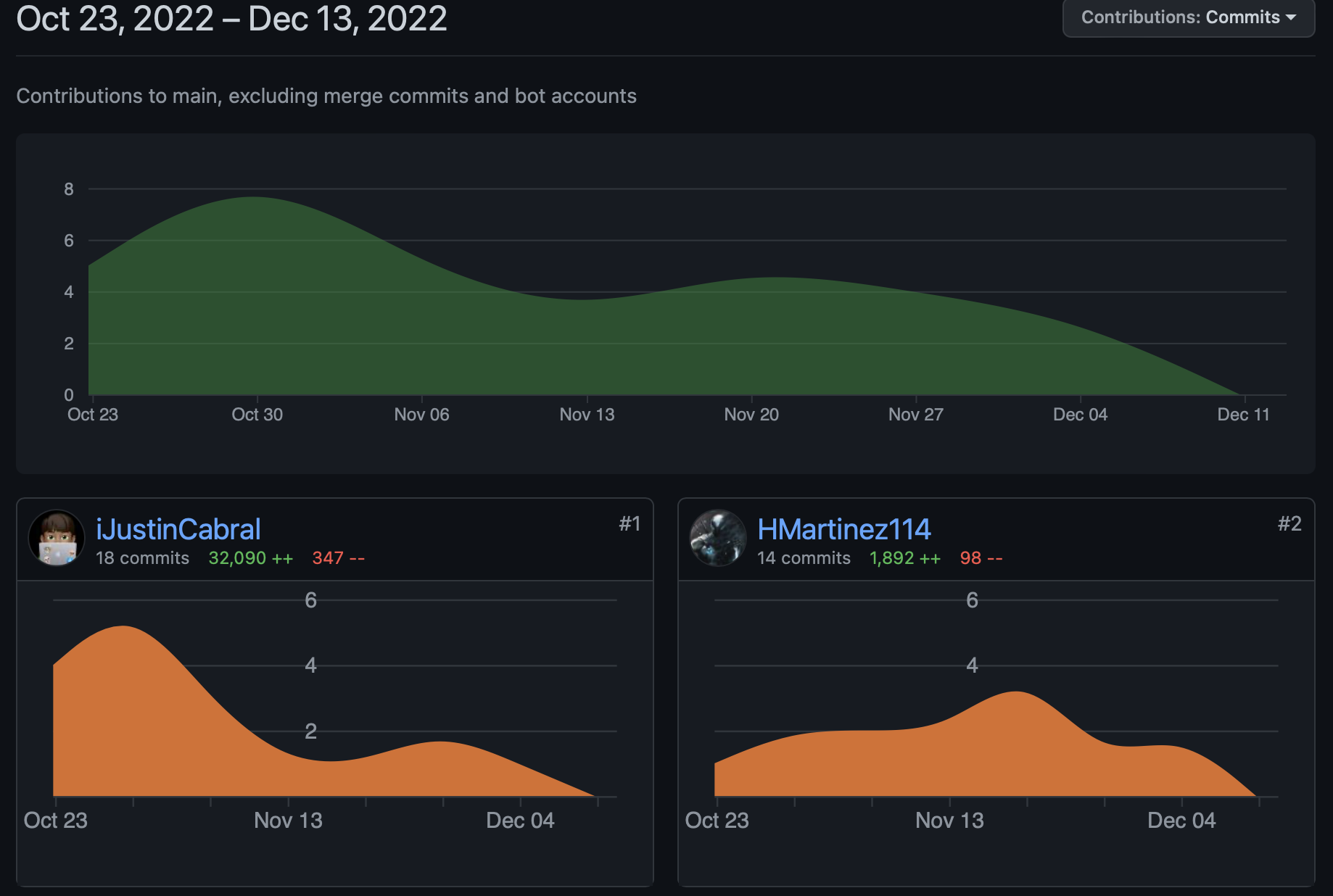
For the back-end development, the tools we utilized were Cloud9, Lambda, API Gateway, and S3. Cloud9 was utilized for storage of all lambda code for each iteration of the project, which was needed to develop and then deploy all lambda functions to AWS. The Lambda service inside of AWS was used to test the lambda functions before activating them on API Gateway within AWS. Then API Gateway was used to make our Lambda function link available on the internet, this created a URL which we can use the fetch function on the front-end to make requests. Finally, S3 was used in order to host all our front-end code on a URL within AWS, this was done by building the project with “npm run build" and then putting the build folder onto our S3 bucket.

# Accomplishments

Below is a screenshot of our Trello board, which we used to keep track of each of the iterations requirements, as well as providing a way to keep documentation of progress.



Below is a screenshot of our front-end GitHub with the commit insights along with a link to the URL where you can see our entire git commit history:



GitHub Link: <https://github.com/iJustinCabral/cloud_store/commits/main>

Our biggest accomplishment was learning how to use AWS without YAML scripts because it saved us so much time in the development process. We were able to focus more on writing the actual lambda code and testing it, instead of constantly worrying about the YAML scripts breaking everything. The YAML scripts were an unnecessary challenge that we were happy to move past by deploying the endpoints directly on AWS.

Another smaller accomplishment for us would be the proper use of version and branch control on GitHub which kept merge conflicts to zero. We used a structure of source control where our main branch was protected, and we were only allowed to develop on "development" branches which would then be merged into the main once testing was complete. This ensured that only the working code was ever merged into the main branch, removing the possibility of our main branch becoming contaminated with error-prone code.

For all the use cases we made very successful lambda functions in order to fulfill the requirements of each one. To make sure that all lambda functions worked correctly we made complex test cases to make sure we covered all requirements on each use case and that they worked flawlessly. Paired programming was done between Jonathan and Miles in order to complete the more complex lambda functions to make sure we didn’t break anything within the lambda functions. This was important in order to save time because having two brains working on one task allowed for fewer syntax and spelling mistakes. Also, this led to fewer errors overall.

**There was nothing for the project that we did not complete.**

## Deliverables

Project Deliverables:

* **Group Analysis**: Lamdba Calculator Link, Storyboards & JSON Payloads, UML Diagram
* **Group Iteration 1**: Database Schemas, Corporate & Manager Log-in page, Authenticate Manager, Authenticate Corporate, Process Shipment, Generate Inventory Report, Assign Item, Create Item, and Create Store.
* **Group Iteration 2**: List Stores, Remove Store, Generate Inventory Report (corporate), Generate Overstock Report, List Stores (customer), List Items on Aisle/Shelf, and Find Item in Store.
* **Group Iteration 3**: Generate Total Inventory Report (manager), Fill Shelves, Show Missing Items, and Buy Item.

# Reflection

## What worked, what didn’t work

The most common problem we ran into was CORS when accessing the API endpoints but once we figured out how to fix that, there really weren’t any other major problems that slowed down our development process. The CORS problem was addressed by looking up how to set up CORS on the API-Gateway and relying on the professor for some advice, which helped us locate where our issues were. Our issues with the API-Gateway CORS errors were mainly not having the Options portion of the Gateway setup and changing the allowed URLs to reflect the front-end URL or making it open and adding in the necessary 200 & 400 responses.

## Our biggest mistake

Our team made no major mistakes. Throughout the project, minor mistakes were made, like forgetting to mark some tasks marked as finished, slightly delaying the development of another lambda function or page. Another example of a minor mistake would be forgetting to change a test case in AWS for lambda functions, sometimes confusing the front-end developers on what to expect in the response of a lambda function. Overall, no mistake cost the team more than half an hour of time. Quick and consistent communication allowed for any confusion to be resolved and development to resume in a timely manner.

## Changes we would make

We, fortunately, did not have any problems during our development so there isn't anything we would change about our project or our project structure.

The biggest change I would suggest for future classes is developing more videos on how to effectively use AWS and not focusing on using YAML scripts to deploy and develop the lambda functions. Our team did not want to spend unnecessary time learning to write YAML scripts, thus we watched LinkedIn learning and Udemy tutorials that showed us how to develop and deploy lambda functions both directly on cloud9 by using the GUI instead of the command line and also testing and editing the code directly on the AWS lambda service. This helped us focus more on writing the lambda code and testing it versus spending more time checking and fixing issues with the YAML script, which we were not very familiar with. The CLI deployment and creation of the lambdas with the YAML scripts provided more challenges than necessary and led to more errors, issues, and misconfigurations of both the lambda functions, code, and API-Gateways. Once, we were familiar with AWS in general and we did not have to rely on the YAML scripts our process for creating, editing, and deploying lambda functions and their respective API-Gateways became more streamlined and involved much less time fixing simple errors.

# Lessons learned

Perhaps the biggest lesson learned here was the coordination between back-end API developers and the front-end web developers. At first, we were going to split into sub-teams to do our programming but once we decided to do swarm programming sessions, we were able to work out any problems when it came to sending and receiving payloads from the API endpoints. We recommend for any team going forward to implement swarm programming so the project can move forward at a steady pace, and no one feels left out or behind. It is also a great way to share knowledge with one another because everyone can bring their own expertise to the table when programming together.

## Things we learned

The most important lesson we learned as a team was how to overcome learning curves from using a new platform that we had never encountered. We had a very small window between finishing the individual assignments and starting the group project, thus the time to overcome some of the learning curves with learning how to use and be familiar with AWS was very little. This really made us think about how to most efficiently and effectively learn what we needed to about AWS to get the project rolling. We learned how to fast-track our familiarization with AWS by using high-quality courses and materials online and passing on the knowledge to one another. This was really important because not all of us had as much spare time to research and learn AWS and therefore required some of us to take a lead, which helped us spend less time figuring out how to use AWS during the iteration phases and led to a more sophisticated application.

An overarching lesson learned from both teams is the importance of communication and clarity. For the front-end website creation, in order to ensure that the communication to the backend worked as intended, both teams needed to agree on what the payload body consisted of, as well as what the response payload would contain. We quickly learned at the beginning of the project that understanding what each side was expecting in terms of payload names and order of payload was vital to getting proper responses to be able to use them in the front end. We also learned how to integrate premade libraries into React projects and the importance of reading their documentation to ensure complete functionality. MaterialUI was an extremely helpful tool for adding clean and functional UI elements to our front end, but due to the library's vast number of preinstalled elements, any developer that doesn’t understand how each element can interact with another or what kind of sx parameters are necessary will not be able to get it working.

## Advice to future teams

Since our team was able to meet all deadlines in a timely manner, our advice to future teams should be to decide early on which people are working on the front-end, and which people are working on the back-end. However, that does not mean you work separately. There was never a time when the front-end developers and the back-end developers were not programming at the same time. We applied a new industry standard that is swarm programming. This helped reduce all conflict when issues arose as to why certain API endpoints weren’t working. We also used Trello to keep track of tasks to be completed, which helped the team stay very organized.

Also, some advice for future teams would be to get to know how AWS works in general and focus on the services you will be mostly working on