# AWS / Clemson Makerspace Capstone Project (Fall 2020) Final Report

December 9th, 2020

Brayden Fidler, Andy Hsu, Reagan Leonard, Evan Moxley, Josh Norman

## Table of Contents

[**AWS / Clemson Makerspace Capstone Project (Fall 2020) Final Report**](#_4y7q1jnakkn) **1**

[Table of Contents](#_qdwt71lvecl) 2

[**Overview/Objective**](#_iey5indlx6z) **4**

[**Background**](#_2ju6ag4f491l) **4**

[**Project Schedule**](#_vwikpac11b7r) **5**

[**Terminology**](#_mjlahvl4bq3n) **6**

[**Tenets**](#_8kchqjdkjqbo) **6**

[**Customer Experience & Working Backwards**](#_ijpvrp75a39t) **7**

[**Requirements**](#_vy3yqkbuytzb) **8**

[**Assumptions**](#_6oo6kvlulttr) **8**

[**Anti-requirements**](#_v1j6imip54oz) **8**

[**Out-of-scope**](#_y4hrogd90u7f) **9**

[**Usage scenarios**](#_onlfpzb1mi2) **10**

[**Diagram and high level design**](#_mg0d0epmnpi7) **11**

[**Interface**](#_zgxmp6wh431l) **12**

[**Details**](#_26a4lyz46dwe) **12**

[**Failure cases**](#_ktrm3ss6iq92) **15**

[**Tests and Audits**](#_zh1qpa596i8m) **15**

[**Scalability**](#_v8s8afp7giba) **16**

[**Availability**](#_a2tqs0hp7io7) **17**

[**Consistency**](#_7y4k3m4eq4ri) **17**

[**Performance**](#_5gajm5qrqse4) **17**

[**Security issues**](#_trtngdm4ikaz) **17**

[**Feature Toggles (FAC)**](#_v187yymdvtpf) **18**

[**Fulfillment/System Engineering/Ops Issues**](#_tyf339pcwzyc) **18**

[**Management Console Issues**](#_f2bu0x21fsif) **18**

[**Metrics**](#_archr7eqgwrw) **18**

[**Dependencies**](#_iosh9psa8oh2) **19**

[**Deployment plan**](#_49kfxbepsd0p) **21**

[**User Guide**](#_tz08na3iov22) **21**

[**Developer Guide**](#_sr58s4ici2jv) **23**

[**Alternate Designs**](#_fdjyczh4je7z) **31**

[**Open Questions**](#_o23khgxp53hi) **32**

[**External Documents (PRFAQ, SIM Issues, Trouble Tickets, etc.)**](#_c4kr0kjyd4h8) **33**

[**Think Big**](#_9d440nn6ll7v) **33**

[**Potential Patents**](#_8sk1efmzbu9p) **33**

[**GDPR alignment**](#_ke59s4oqpt1h) **33**

[**Example Workflows**](#_3feb5gd7yn92) **33**

[**Meeting Notes**](#_i391xlk2nr0u) **34**

# **Overview/Objective**

The goal of this platform is to provide the makerspace with a more efficient maintenance support system in order to help them save time, reduce mistakes, and improve their overall process. The primary functionality of this system will be a maintenance schedule that updates and shows recurring tasks and an alert system that notifies the maintenance lead that something needs to be done.

Members of the maintenance team will be able to view the various tasks for machines that need to be completed on a nightly, weekly and monthly basis. Team members will be able to mark tasks as complete in the system once they have physically completed the work.

If tasks are not completed by the specified time, the maintenance lead will be alerted of the incomplete task, so he/she can take action as needed. The maintenance lead will also be able to add, update, and delete tasks on an ‘as needed’ basis as maintenance needs change. This will allow the system to change over time as the Makerspace grows, adding new devices and complexity to the maintenance process. Additionally, the Maintenance team will be able to review their efficiency by reviewing the number of tasks completed and missed over various periods of time.

# **Background**

The Clemson Makerspace is an on campus lab that has a variety of laser cutters and 3d printers and is open for Clemson students/faculty to use. The goal of the Makerspace is to provide them with the resources to bring their product from concept to prototyping with machines that would otherwise be too expensive for any one individual to afford.

The Makerspace has grown over the course of the last several years, yet has encountered inevitable challenges with its growth. One of its challenges is performing regular maintenance on machines and equipment in a timely and efficient manner. Clemson employs a student-run maintenance team to upkeep and maintain these machines.

Due to the limited time and resources at the students disposal, sometimes maintenance tasks are left uncompleted. This maintenance system currently operates with a simple paper/spreadsheet system to track and schedule regular maintenance tasks. While this system worked in the early stages of the Makerspace’s growth, the maintenance is now looking to upgrade to a more complete solution to handle their workload. Thus, the Makerspace is partnering with Amazon Web Services and our Capstone team to deliver a platform to handle their maintenance load.

# Project Schedule

* **Sept. 18 - Beginning of first sprint; Beginning of project development**
  + Sprint 1: Frontend team starts on UI Wireframe; Backend team researches AWS Services/Creates Diagrams; Full team begins first draft of AWS design doc
* Sept. 25 - End of Sprint 1
  + Sprint 2: Frontend finish UI Wireframe; Backend start on development of UML Diagrams
* **Oct. 2 - End of Sprint 2: Finish Wireframe (Present to Makerspace and AWS)**
  + Sprint 3: Frontend research on AWS services for website hosting; A/B Test Wireframes
* **Oct. 9 - End of Sprint 3: Finish UML Diagram of Backend design; First draft of AWS design doc completed**
  + Sprint 4: Frontend starts development on main page of interface; Backend starts on Maintenance Task Development
* Oct. 16 - End of Sprint 4
  + Sprint 5: Backend starts on Maintenance Queue Development; Frontend continues developing main page
* Oct. 23 - End of Sprint 5
  + Sprint 6: Frontend continues developing main page; Backend finishes up maintenance queue
* **Oct. 30 - End of Sprint 6; Maintenance Queue Deployment; Main page of interface deployed for feedback; Final version of AWS design doc completed**
  + Sprint 7: Frontend begins development on child pages; Backend starts development on notification system
* Nov. 6 - End of Sprint 7
  + Sprint 8: Frontend continues development of child pages; Backend continues development of notification system
* Nov. 13 - End of Sprint 8
  + Sprint 9: Frontend continues development of child pages; Backend continues development of notification system
* **Nov. 20 - End of Sprint 9; Notification System Deployment; Entire UI deployed for user testing**
  + Sprint 10: Frontend team gathers feedback from Makerspace users on UI; Backend team gathers feedback from Makerspace users on system functionality; Both teams work on resolving issues
* **Nov. 27 - End of Sprint 10; Final testing begins**
  + Sprint 11: Begin testing final version of the system; Finalize project report; Write any remaining sections not yet completed
* **Dec. 4 - End of Sprint 11; Official end date of project; PRFAQ released**

# **Terminology**

* **Maintenance Task** - Tasks that are regularly performed by the employees of Makerspace to keep the space in working order
* **AWS CDK** - Amazon Web Services Cloud Development Kit (environment of the development of the program)
* **Maintenance Lead** - the head employee of the Makerspace who is ultimately responsible for the upkeep of the Makerspace lab and machines
* **Maintenance Team** - the team responsible for the day-to-day maintenance of the machines
* **Maintenance Queue** - the list of tasks with corresponding priority of its necessity of completion
* **Alerts** - notifications sent to the users to complete a scheduled maintenance task
* **Clemson’s SSO -** Single Sign On login implemented and managed by Clemson that requires a Clemson email and password, as well as a Dual-Factor authentication

# **Tenets**

**Put the Customer First**

* The main goal is to develop something that will benefit the Clemson Makerspace, so this should be the most important tenet we follow in the design.
* This tenet is one of Amazon’s key leadership principles and as our Capstone partners, it seems appropriate to follow in their footsteps.

**Allow Room for Improvement**

* Our design should never be set in stone. There may always be new information or opportunity to make the design better for the customer, opportunities that we should always be ready to take

**Consider Alternatives**

* Don’t just pick the best plan and execute it, always consider plan B, C, D, etc. so that each option is thoroughly thought out. In the future it might turn out that one of the alternative plans previously not considered becomes the new Plan A.
* There is not always one right way to do something

**Ask for Help**

* Although we are a new team with not a whole lot of real-world experience, we are surrounded with countless professional resources that are specifically meant to help us. We should be proactive in using these resources as best we can to help us achieve our goals for this design.
* This also goes for asking one another for help as a team because the ultimate outcome and responsibility for this project and design is shared

# **Customer Experience & Working Backwards**

The experience of the application should be mutually beneficial between the maintenance team and the maintenance lead while also benefiting the makerspace user with better maintenance outcomes.

The maintenance lead will benefit from having reminders when maintenance tasks are left incomplete allowing him or her to assign the work to other team members. The lead will also be able to confirm successful completion of tasks and the team member responsible for completing said tasks. Having this information could facilitate communication between the maintenance team, improving the overall quality of maintenance.

On the flip side, the maintenance team will be able to see maintenance tasks that need to be completed rather than relying on assumptions and a simple spreadsheet. Having the ability to see what has already been completed, what needs to be completed, and who has completed previous tasks in real time should help improve their process. Ideally, communication between team members will improve simply by having this information readily available. It will also minimize tasks that fall through the cracks.

One conflict we do want to avoid is creating a scenario where the maintenance lead is micromanaging team members. We realize the power of this application will increase ownership for maintenance tasks, which could lead to increased pressure on the student run maintenance team. We aim to design the application to avoid this kind of interaction while improving the system that is in place now.

Finally, the most critical experience is that of the end user of the makerspace - the students. The goal of this application is to improve the maintenance team’s process by providing real time maintenance tracking. Ideally, this will reduce the downtime of machines, enhance their efficiency, and increase their longevity. Hopefully, the improved quality of maintenance will lead to cost savings, which could in turn by funnelled back into the makerspace - improving the quality and availability of equipment for students in the future.

# **Requirements**

The System will be accessible by web browsers on personal devices or in the makerspace.

The System will have a list of maintenance tasks to be completed on a weekly, nightly, and monthly basis for the maintenance team.

The Maintenance Lead will have the ability to edit these tasks and add or remove tasks.

The Maintenance Lead will have the ability to change the frequency of tasks.

The Maintenance Lead will have the option to assign tasks to one or more employees.

The Maintenance Lead will have the ability to specify the Machines on which Maintenance Tasks are performed.

The Maintenance Team will have the ability to mark tasks as completed once they have been physically completed.

The System will sort Maintenance Tasks by machine as well as by categories specified by the Maintenance Lead.

The System will notify the Maintenance Lead via Email/Text when Maintenance Tasks are not completed by the specified time (typically the end of shifts).

The System will display various efficiency metrics for Maintenance Tasks and Machines.

The System will not be maintained by this team after Dec. 4th, 2020 and there is currently no operations team to maintain it going forward.

# **Assumptions**

Scheduling structure will follow a queue

We will not need to differentiate roles in terms of who can edit/delete tasks

The Makerspace will be responsible for funding any ongoing expenses after project completion, so we should take this into account when we choose services.

We will use AWS Lambda and Dynamo DB, which will not cost any more than the AWS free tier usage

Assumption that everyone in the makerspace is qualified to do every task

# **Anti-requirements**

The application will not create any employee specific reports on maintenance task completion due to the makerspace’s team environment. This metric utility would not properly show the team dynamics and external factors.

The application will not continually notify the maintenance team for each task completion or incompletion via SMS or Email in order to respect the employee’s time.

The application will not require a login using a card reader so that users can access the system from their phone without extra hardware

# **Out-of-scope**

* We are currently not covering any non-regularly scheduled tasks. That is, any maintenance tasks that may arise unexpectedly or that need to be completed but on an irregular basis are not currently going to be included in the maintenance queue.
  + This is considered out-of-scope according to the PRFAQ and from speaking with the customer.
* We are currently not planning to poll the Makerspace machines to monitor downtime/uptime.
  + This is considered out-of-scope for this project at the moment because of the complexity of this feature. It could be implemented in a future capstone project.
* We are currently not planning to implement any user login/SSO login into the system. As it stands now, our plan is to treat each user of the site the same as far as editing permissions.
  + We realize that this is very suboptimal and hope for this to actually be in-scope before the end of the semester/project. However, we are unsure that we will have enough time to implement this, so for now it is out-of-scope.

# **Usage scenarios**

Scenario 1: A Makerspace employee (Joe) goes into the lab to work his 4-7pm shift. He clocks in and then goes straight to one of the lab’s desktop computers to check the maintenance schedule. After logging in, he immediately sees the upcoming tasks at the top of the page that need to be completed before he leaves for the night. Joe communicates with his coworkers to decide who will do which tasks. Later, he begins completing the tasks he agreed to do. As he walks around the lab performing the maintenance tasks, he has his phone in his hand with the maintenance website pulled up so that he can mark the tasks complete on-the-go. Once he has completed all his tasks, Joe looks at his watch. It’s 7:00pm...time to go home!

Scenario 2: It’s 7:03pm the following day. The Makerspace Maintenance Lead (Steph) gets a notification on her phone. It’s an email from the Makerspace Maintenance System saying that 3 tasks were not completed on time and are now *LATE*. It’s possible that one of her employees did these tasks but simply forgot to mark them complete in the system. But just to be sure, she texts the employee who was in charge of closing up that night to ask what happened. The employee explains that he is sure that he and one of his coworkers completed their tasks that night, but isn’t sure about the third employee. Steph texts the third employee who tells her that she forgot to do her 3 tasks. With her knowledge of the urgency of the tasks, Steph has the choice to either tell the third employee to go back into the lab tonight to complete the maintenance or leave them until the morning. These tasks aren’t absolutely crucial and will not cause lasting damage to the machines if they are left overnight. So Steph tells the employee to get the tasks done as soon as she can the next morning. The employee apologizes and assures Steph that she will remember to check the system for her upcoming tasks next time.

Scenario 3: At the end of the week, Steph decides to check the history report for the week. She sees 2 tasks from Monday that were completed by Billy but were late. Billy happens to be working today so she walks over to him and asks why his tasks were late (she must have swiped away that email notification and forgot about it). Billy explains that he did complete the tasks on time that night, but forgot to mark them complete in the system until he got home about 30 minutes later. Steph lets him know that this is completely understandable and is no problem at all. She thanks him for his honesty.

# **Diagram and high level design**

The Makerspace Maintenance application is designed to be a simple serverless web application that utilizes a number of basic AWS services. The users should be able to handle any type of maintenance task, allow for future growth, and review completed data. This will be done using the basic AWS architecture shown below.

[Live UI Wireframe Designs](https://xd.adobe.com/view/6a430a3d-83f8-47d4-a0e8-db609f53244c-3f10/?fullscreen&hints=off)



# 

# **Interface**

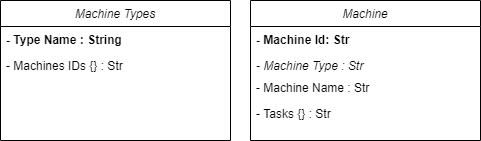
Internal/External API interfaces

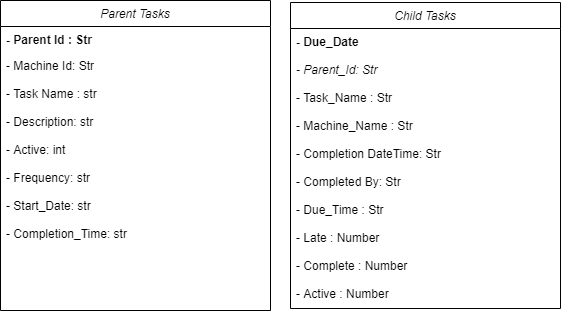
Intra-package Component interfaces

Domain models

# **Details**

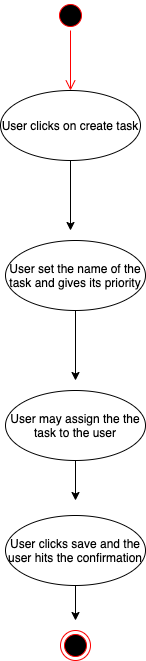
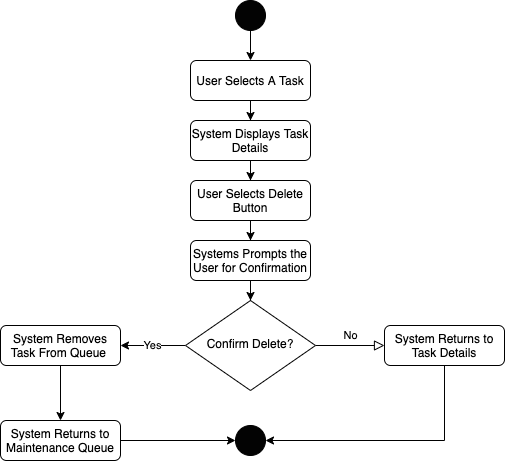
Class Diagrams:



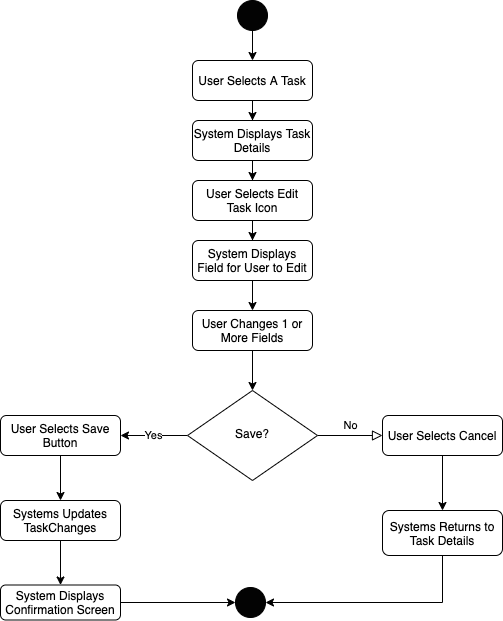
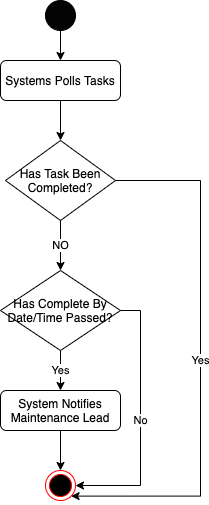


Flow Diagrams:

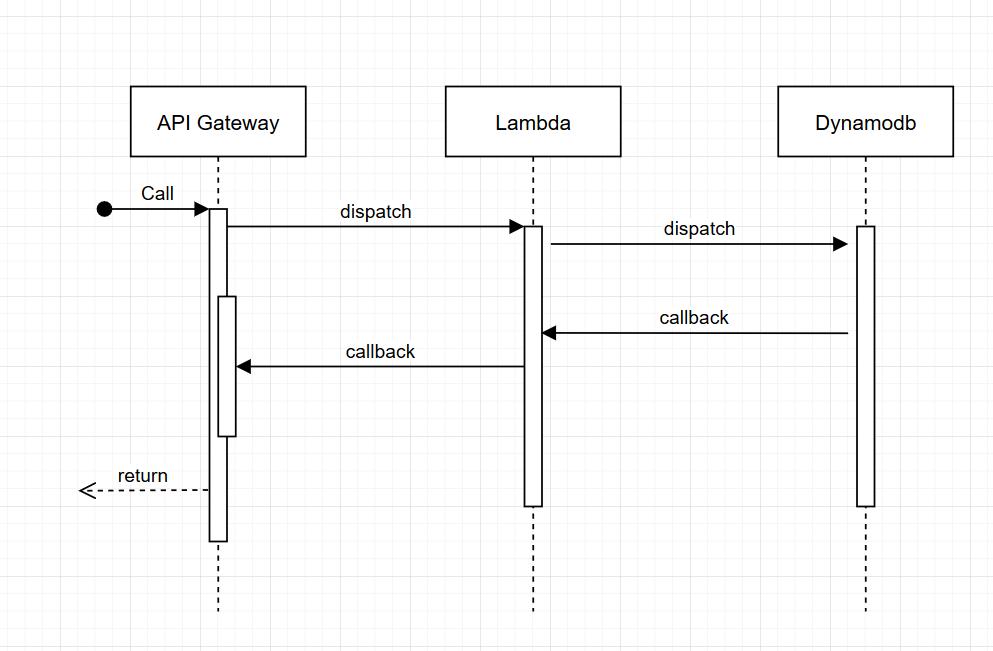
**Create Task**  Delete Task

Edit Task Notify Maintenance Lead

Sequence Diagram:



API/Lambda Function Details

* Machine Configuration
  + Add Machine
  + Add Machine Type
  + Delete Machine
  + Delete Machine Type
  + Edit Machine Name
  + View Machine
  + View Machine By Types
  + View Machine Types
  + View Machine Upcoming Tasks
* Task Configuration
  + Create Task
  + Complete Task
  + Delete Task
  + Edit Task
  + Main Task
  + View Task
  + View Upcoming Tasks
* Reporting
  + Export History
  + View History

# **Failure cases**

In consideration of our design and implementation, we have foreseen several scenarios where the system may fail or need revision. The first concerns ease of access. In this scenario, we worry that the simple act of checking the application and marking tasks as completed may be seen as an annoying extra step. As a result, members of the maintenance team may omit or ignore the action of marking a task as complete, and the utility of the system is lost. Now, the maintenance lead would be effectively unaware of which tasks have and have not been completed.

Along those same lines, we worry if gaining familiarity with the application each semester may also cause users to avoid using the system. Given the system will primarily be utilized by college students, new team members and maintenance leads will be joining each and every semester. This means the new users must register with the system while becoming familiar with its ins and outs. If the process is too time consuming and difficult, we can foresee a scenario where a group of students elects to simplify the process and return to paper tracking or other measures.

Finally, our last failure scenario concerns the maintenance lead. Given the responsibilities of the maintenance lead and the administrative privileges of this person, the system could create added responsibilities for her. If the system is not designed efficiently, the lead could find herself spending additional time maintaining and reviewing the application despite its goal of simplifying the challenges of her role. Furthermore, we could foresee a scenario where an overly ambitious maintenance lead utilizes the application to micromanges the maintenance team, watching the task completion metrics with hawk-like scrutiny.

# **Tests and Audits**

1. Valid Employee Login
2. Invalid Employee Login
3. Creating a new account
4. Email verification completion
5. Deleting a maintenance task
6. Deleting a machine
7. Valid Admin Privileges Displayed
8. No Maintenance Tasks Available for Machine
9. No Maintenance Tasks Available for the Night
10. Large list of maintenance task (50) due in a night
11. Large list of maintenance tasks (120) on a machine
12. Recurring instances of monthly tasks
13. Recurring instances of nightly task
14. Recurring instances of weekly tasks
15. Instance of single tasks
16. Missed Task email notification
17. Change maintenance lead valid email input
18. Change maintenance lead invalid email input
19. Creating a task with the same name as another task
20. Completing a missed task
21. Download complete history of makerspace in CSV format
22. Download complete history of a machine in CSV format
23. Download complete history of a single week in CSV format
24. Download a complete history of single day in CSV format
25. Download an empty case of history in CSV format
26. Failure to fill in a data field when creating a new machine
27. Failure to fill in a data field when creating a new task
28. Failure to fill in a data field when creating a new account
29. Once a task is completed it shows the change within the main page
30. Once a task is completed it is removed from the upcoming tasks in the machine details sections
31. Filter the history section by date
32. Filter the history section by machine
33. Filter the history section by completion status
34. Sort the history section by date
35. Sort the history section by completion status
36. Go back to home from all pages
37. Cannot redirect without saving or discarding changes
38. Go back to the previous page using the X button in the corner
39. Go back to the previous page using the browser's native back button

# **Scalability**

Does the feature use DynamoDB tables? Do they have auto-scaling enable? Is the ratio of used to allocated capacity adequate? Do the tables have adequate minimums on a per region basis?

Does your design consider scalability concerns?

Does your design change our scalability posture?

Auto-scaling is enabled for this project due to the expandability of the servers provided by AWS. In addition, the values input into the database are in text format - allowing for minimum storage consumption. The database itself is also the free tier version of the dynamodb. All the handling of stability and scalability are handled by AWS. Besides the server choices, the separation of designs done in the program allows for administrators to add and remove users or tasks without trouble. Therefore, if extra space is required, the administrators are able to delete old data without trouble.

# **Availability**

Our design features a serverless web design utilizing AWS managed service. This design reduces downtime and increases availability. The AWS services chosen are available within all North American AWS regions. DynamoDB ( our non-relational AWS Database) operates with five-nines of reliability. AWS Lambda ( our compute AWS solution ) is running with 99.95% uptime. Our product is designed to be used by small teams of workers throughout the day. Therefore, we expect small amounts of site traffic, even with projected growth.

# **Consistency**

The performance of the program is still unknown at the moment. [Refer back after more programming is completed]

# **Performance**

We are electing to prioritize base performance rather than a load tested performance plan at this time. There are two reasons for this decision. The first and most obvious is that we are on a fixed schedule, and developing scripts and test scenarios for high volume workloads takes time. The other is that for the most part the volume of active users as well as data points will be fairly low. Our task history will grow overtime, but the customer/users do not plan to access the entire history of task completions on a regular basis.

This leads us to an important question that will affect performance. How will we access the tasks in the DynamoDb efficiently? For cost reasons we don’t want to scan the entire table on a regular basis, but we do need to query on multiple fields (ParentId, Completion Status, and Due Date). Right now we plan to utilize local secondary indexes, select our primary and sort keys with caution, and review the amount of records that are accessed by simple database calls.

As for our other services, we are not particularly concerned about performance for Lambda Functions, API Gateway Calls, or CloudFront Storage. The number of functions, calls, and web page hits will be relatively consistent in the immediate future, so we don’t anticipate having performance issues caused by these services.

# **Security issues**

System Security

* The system login has two different planned implementations. Ideally, it would be protected by Clemson’s SSO login system and limit access to those with a Clemson login, but this is dependent on whether our team gets permission from Clemson. Otherwise, it will require simple email verification to login, so potential security issues involve email authenticity.
* There will also be a separate role for Maintenance Lead to limit the number of users that can add/edit tasks in the system.
* The limited access to changing the database through the system will help protect the infrastructure and running up costs as well.

Database Security

* Each developer will use the AWS CDK along with their own AWS accounts and Github to stand up the development environment so we do not have to deal with IAM roles.
* In production, we will transfer this over so only a Makerspace-owned AWS account will have access.
* Security issues for this fall under regular AWS account security and making sure that only the people we want have read/write permissions.
* One potential issue is that since almost anyone can create an account and view the system after completing email authentication, the database could be subject to receiving a large number of new sign-ups, so we might need to implement functionality to remove members from the database after a maximum of 4 years, to avoid having unnecessary users left in it.

# **Feature Toggles (FAC)**

N/A

# **Fulfillment/System Engineering/Ops Issues**

N/A

# **Management Console Issues**

N/A

# **Metrics**

How will we measure the success of this project/feature?

The success of this project will be determined by its ability to increase the efficiency and accountability of the Clemson Makerspace’s performance of their regularly scheduled maintenance tasks. It should do this in a cost-effective manner, being virtually (if not actually) free for the Makerspace to use. The system should help not only the maintenance lead, but also the Makerspace employees who may need reminders to do their maintenance tasks.

How will we monitor usage patterns with respect to this design?

Currently, there is no way to monitor the actual usage/uptime of the machines. However, we are able to monitor the usage of our web application. AWS Lambda monitors how many lambda function calls are made in the system and stores this information in the AWS cloud. DynamoDB keeps track of how many reads/writes are made to the database. Our system also has a built-in monitoring section that keeps track of the history of every task that passes through the system. This task history report can be found under the “History” tab in the “Other” section of the home page.

How will we be alerted of potential failures?

Potential failures will most likely only be noted by the Makerspace staff and not the system, itself. However, it is highly likely that any potential failures would be caught relatively quickly (at the very least, on the same-day) as there will always be nightly maintenance tasks that are due. So if for some reason there were not tasks due on a particular day, the Makerspace staff would know that something was wrong. Although, it should be noted that we currently do not have an “operations team” for this system, so it is unclear who they would report those errors to (potentially Dr. Herzog).

# **Dependencies**

* AWS CDK
  + Availability: Available in US East 1 Region
  + Limitations: Limited to Python, Java, Typescript, and .Net implementations.
  + Minimum Integration:*npm install -g aws-cdk*
* AWS CLI
  + Availability: Available in US East 1 Region
  + Limitations: Only a Command Line Interface, but used in CDK.
  + Minimum Integration: See the [AWS Command Line Interface installation](https://docs.aws.amazon.com/cli/latest/userguide/installing.html) page for more details.
* Node Js
  + Availability: Available in US East 1 Region
  + Limitations: Only used in CDK.
  + Minimum Integration: To install Node.js visit the [node.js](https://nodejs.org/) website.
* AWS Lambda
  + Availability: Available in US East 1 Region
  + Limitations: <https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html>
  + Minimum Integration: *pip install aws-cdk.aws-lambda*
* AWS API Gateway
  + Availability: Available in US East 1 Region
  + Limitations: 10,000 requests per second (RPS) with an additional burst capacity provided by the token bucket algorithm, using a maximum bucket capacity of 5,000 requests. \*
  + Minimum Integration**:** IAM Roles established and Lambda functions implemented.
* AWS DynamoDB
  + Availability: Available in US East 1 Region
  + Limitations: Cannot query by non-primary keys. Also pricing limitations for read/writes found here: <https://aws.amazon.com/dynamodb/pricing/provisioned/>
  + Minimum Integration: Simply having credentials for the account that will store the DB tables. Also creating said tables using scripts.
* AWS S3
  + Availability: Available in US East 1 Region
  + Limitations: <https://docs.aws.amazon.com/AmazonS3/latest/dev/BucketRestrictions.html>
  + Minimum Integration: TBD
* Python
  + Availability: Available in US East 1 Region
  + Limitations: None for our scope.
  + Minimum Integration: Additionally you will need to have the Python package installer (pip) installed. See installation instructions [here](https://pypi.org/project/pip/).
* Python Sdk for AWS
  + Availability: Available in US East 1 Region
  + Limitations: None for our scope.
  + Minimum Integration:*pip install boto3*
* React
  + Availability: Available in US East 1 Region
  + Limitations: TBD
  + Minimum Integration: You can [add React to an HTML page in one minute](https://reactjs.org/docs/add-react-to-a-website.html).
* AWS Cloudfront
  + Availability: Available in US East 1 Region
  + Limitations: <https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-limits.html>
  + Minimum Integration: Content uploaded to S3 and permissions created.

# **Deployment plan**

The current plan is to deploy the final version of our system on December 4th, 2020. This deployment will involve handing over the site’s URL to the Clemson Makerspace. We are building the system to be fully customizable by the system admins. The initial admins will be the current Maintenance Lead (Meg Nutall) and the Makerspace faculty member. These admin members will be able to edit tasks, machines, and add/remove other admins. Therefore, there will be no need for an operations team for this system.

Before handing over the working URL, we will deploy the project under the Clemson Makerspace AWS account so that they will have ultimate ownership of a new production environment. The setup will go as follows:

(This setup can be done on any machine through a terminal window as long as the correct AWS credentials are used)

1. Setup AWS CDK
   1. Follow instructions at <https://cdkworkshop.com/> under “Prerequisites” to set up the right environment
2. Once the cdk is setup and the correct aws credentials are entered the code can be cloned from github by running “git clone <url of repository>”
3. Now, under the project home directory, the correct packages must be installed by running “pip install -r requirements.txt” and “npm install”
4. Next run “cdk synth” “cdk bootstrap” and then “cdk deploy” to successfully build and deploy the AWS services under the current account

# User Guide

This user guide is meant to provide a sense of what we, the development team, had in mind for what the typical user, as well as an admin user, would use this maintenance system for. We also answer a list of what could be FAQs that both types of users might have when first learning to navigate and use the system.

**Typical User**

The typical user of this system would be a normal Clemson Makerspace employee. As we understand it, every Makerspace employee is required to have completed some degree of training on how to perform the regularly recurring maintenance tasks that our system is built to be used for. Each Makerspace employee is also required to perform certain maintenance tasks each working shift. Therefore, the typical use cases in which these users would interact with the system will be: 1) Checking what upcoming maintenance tasks need to be done during or at the end of their current shift, 2) Marking these tasks complete in the system, and 3) Potentially checking the History page to see if any tasks have been missed that need to be made up.

**FAQs**

* How do I know which tasks are due today?
  + The “Upcoming Tasks” section of the home page should show every task that is due that day. You can also view upcoming tasks for each machine by going to the machine’s page and clicking on the “Upcoming Tasks” button.
* How do I complete a task?
  + Simply click on the task and on the following page, click the “Mark Complete” button. There should be a confirmation page that pops up after you click this button to confirm that you marked the task complete.
* How do I view a particular machine’s page?
  + Click on the machine type that you are looking for in the “My Machines” section of the home page. This will produce a dropdown menu of all the individual machines that are of that type. Then click on the particular machine that you want to view. This will take you to that machine’s page.
* How can I view upcoming tasks beyond just the ones due today?
  + Navigate to the machine page of the machine that you wish to view more Upcoming tasks for. This will show a list of all future tasks associated with that machine.
* How can I view task history?
  + Task history can be viewed by clicking on the “History” button in the “Other” section of the home page. There is also a “History” button found on each machine page.

**Admin User**

The admin user is designated as the Makerspace Maintenance Lead during any given semester. This user is the one who is ultimately responsible for making sure all maintenance tasks are completed. Therefore, this user will need a higher level of control over the Makerspace Maintenance system. The admin’s typical use cases when using the system will be all of the above cases of the typical user, as well as: 1) Configuration of any new machines, 2) Editing the frequency or details of any tasks necessary, 3) Creating any new tasks that are necessary, 4) Editing details of machines when necessary, 5) Managing admin settings if they are leaving the Maintenance Lead position and designating a new one, and 6) Receiving *all email alerts* from this maintenance system for late/missed tasks and using these alerts to make sure the tasks get completed in a timely manner.

**FAQs**

* When will I receive email alerts from the maintenance system?
  + Email alerts are sent out from the maintenance system whenever any task does not get completed by Makerspace closing time (7:00pm) on its due date.
* How do I create a new maintenance task?
  + To create a new task, navigate to the machine page of the machine that the new task will be for. Once on the correct machine page, click on the “Create New Task” button. Then fill out the form with all the correct information for the new task and click “Submit”. This will enter the new task into the database and it will repeat according to whatever frequency was set in the form (Daily, Weekly, or Monthly).
* How do I edit an existing task?
  + Editing tasks is much like creating a new task. Navigate to the appropriate machine page that the existing task belongs to and click the “Edit Tasks” button. This will take you to a page that lists each existing task (one instance each) associated with that particular machine. Click on the existing task you want to edit; this will take you to the edit form which should be filled out with the task’s current information. Change the details you would like to change and click “Submit”. This should bring up a confirmation screen.
* Is there a way to delete a task?
  + Yes. To delete an existing task, follow the above steps for editing a task, except on the edit page, simply click the “Delete” button. This will remove the task and all existing instances of it from the database.
* How do I edit a machine’s details?
  + Navigate to the machine page of the machine you would like to edit. Click “Edit Machine Details”, edit the appropriate details, and click “Submit”. This should bring up a confirmation screen.
* How do I configure a new machine?
  + Click on the “Configure New Machine” button in the “Other” section of the home page, enter the details of the new machine, and click “Submit”. This should bring up a confirmation screen.
* How do I manage the admin settings/change admin emails?
  + The only real “admin setting” that exists is the admin email. To change this, simply click on the “Manage Admin Settings” button in the “Other” section of the home page. On the next page, type in the new admin email address and click “Submit”. This should bring up a confirmation screen.

# Developer Guide

**API Gateway**

The following section provides the links, detailed descriptions, parameters, returned objects, and example input for each API call used in this project. They are listed by type: Machine calls, Reporting calls, and Task calls.

**Machine**

* Add Machine: <https://g9rkkyy2bi.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
    - Adds a new machine to the database
    - Inputs are new machine id, machine type, and new machine name
    - If the machine to be added is of a type that does not already exist, Add Machine Type must be called first to add this new type.
    - If the machine to be added already exists, it will return a 400
  + Params
    - machine\_id = <id>
    - machine\_type = <type>
    - machine\_name = <name>
  + Returns
    - A json object with a string indicating success
  + Example Input
    - machine\_id = L180
    - machine\_type = Laser
    - machine\_name = Laser-180
    - Returns: “Added machine: Laser L180 Laser-180”
* Add Machine Type: <https://wukxw7ggcl.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
    - Adds a new machine type to database with an empty set of ids
    - Input is new machine type
    - If the machine type already exists, it will return 400
  + Params
    - machine\_type = <type>
  + Returns
    - A json object with a string indicating success
  + Example Input
    - machine\_type = Laser
    - Returns: “Added machine type: Laser”
* Delete Machine: <https://twka1f7541.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
    - Deletes a machine from the database by removing it from the machine table and machine types table
    - It also marks all associated parent and child tasks as inactive
    - Upcoming child tasks are also deleted
    - Inputs machine id and machine type of the machine to be deleted
  + Params
    - machine\_id = <id>
    - machine\_type = <type>
  + Returns
    - A json object with a string indicating success
  + Example Input
    - machine\_id = L1
    - machine\_type = <Laser>
    - Returns: “Deleted machine: L1”
* Delete Machine Type: <https://w8kfq30ys9.execute-api.us-east-1.amazonaws.com/prod/> 
  + Description
    - Deletes a machine type from the database by removing it from the machine types table
    - Input is machine type to be deleted
  + Params
    - machine\_type = <type>
  + Returns
    - A json object with a string indicating success
  + Example Input
    - machine\_type = <Super Laser>
    - Returns: “Deleted machine type: Super Laser”
* Edit Machine Name: <https://zpox05lcz2.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
    - Edits a machine’s name in the database by updating the machine table
    - Inputs are machine id of the machine to be edited as well as a new name for this machine
  + Params
    - machine\_id = <id>
    - new\_name = <new name>
  + Returns
    - A json object with a string indicating success
  + Example Input
    - machine\_id = L1
    - new\_name = Laser-1
    - Returns: Edited machine: L1 with new name of Laser-1
* View Machine: <https://hy6p11mokb.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
    - Returns the details about a machine from the database
    - Input is id of requested machine
  + Params
    - machine\_id = <id>
  + Returns
    - A json object with each attribute of a machine in it as a key value pair
  + Examples
    - Ex 1
      * machine\_id = L3
      * Returns: {"Machine\_Id": "L3", "Type": "Laser", "Name": "Laser-3", "Tasks": ["645040b4-01be-4e15-b372-9e6f339c6695"]}
    - Ex 2
      * machine\_id = L4
      * {"Machine\_Id": "L4", "Type": "Laser", "Name": "LaZer-4"}
* View Machine by Types: <https://cyo1ay4lk2.execute-api.us-east-1.amazonaws.com/prod/> 
  + Description
    - Returns the details about all machines of a particular type
    - Input is the type of machine requested
  + Params
    - machine\_type = <type>
  + Returns
    - A json object with each machine of this type
    - Keys are machine ids and values are machine names
  + Example Input
    - Example 1
      * machine\_type = Laser
      * Returns: [{"Machine\_Id": "L1", "Type": "Laser", "Name": "Laser-1"}, {"Machine\_Id": "L4", "Type": "Laser", "Name": "LaZer-4"}, {"Machine\_Id": "L2", "Type": "Laser", "Name": "Laser-2"}, {"Machine\_Id": "L3", "Type": "Laser", "Name": "Laser-3", "Tasks": ["645040b4-01be-4e15-b372-9e6f339c6695"]}]
    - Example 2
      * machine\_type = Stapler
      * Returns: [ ]
* View Machine Types: <https://5av6zehh1f.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
    - Returns all the machine types in the database
  + Params
    - N/A
  + Returns
    - A json object with all the machine types
    - Keys are generic numbers and values are the different types
  + Example
    - Returns: [{"Machine\_Type": "3D Printers: Mini", "Machines": ["Mini1SKU"], "Machine\_Names": ["Mini 1"]}, {"Machine\_Type": "3D Printers: Taz XL", "Machines": ["NEWSKU", "TAZ223", "TAZ1SKU"], "Machine\_Names": ["TAZ2", "AnotherNewTaz", "Taz 1"]}, {"Machine\_Type": "Other", "Machines": ["O3", "O6", "O56"], "Machine\_Names": ["Other-3", "Other 1", "OTHER2"]}, {"Machine\_Type": "Laser Cutters", "Machines": ["LaserSKU"], "Machine\_Names": ["Laser 1"]}]
* View Parents By Machine: <https://49v5jw5aa4.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
    - TDB
  + Params
    - MachineId = <id>
  + Returns
    - A json object with all ParentTasks for the Machine
    - Keys are generic numbers and values are the different types
  + Example
    - MachineId = L3
    - Returns: [{"Start\_Date": "20201124", "Frequency": "Daily", "Machine\_Id": "L3", "Completion\_Time": "900", "Description": "Be Careful", "Parent\_Id": "645040b4-01be-4e15-b372-9e6f339c6695", "Name": "Clean Laser Parts"}]
* View Upcoming Tasks for Machine: <https://n4cezf1492.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
    - Returns all upcoming tasks (child tasks) for a specific machine
    - Only returns tasks that have not been completed and active
    - Input is the id of the machine and the days forward to search for upcoming class
    - It is configurable to return upcoming tasks on a specific day, or defaults to today’s date if not specified
  + Params
    - machine\_id = <id>
    - DaysForward = <days> (from 0 to +N)
  + Returns
    - A json object with all the upcoming tasks for that machine on the upcoming days
    - Keys are parent ids of tasks and values are objects with task details
  + Examples
    - Ex 1
      * machine\_id = L3
      * DaysForward = 0
      * Returns: [{"Completed\_DateTime": "", "Due\_Time": "1000", "Frequency": "Daily", "Task\_Name": "Clean Laser Parts", "Due\_Date": "20201125", "Completed\_By": "", "Machine\_Name": "Laser-3", "Parent\_Id": "645040b4-01be-4e15-b372-9e6f339c6695"}]
    - Ex 2
      * machine\_id = L3
      * DaysForward = 2
      * Returns: [{"Completed\_DateTime": "", "Due\_Time": "1000", "Frequency": "Daily", "Task\_Name": "Clean Laser Parts", "Due\_Date": "20201125", "Completed\_By": "", "Machine\_Name": "Laser-3", "Parent\_Id": "645040b4-01be-4e15-b372-9e6f339c6695"}, {"Completed\_DateTime": "", "Due\_Time": "1000", "Frequency": "Daily", "Task\_Name": "Clean Laser Parts", "Due\_Date": "20201126", "Completed\_By": "", "Machine\_Name": "Laser-3", "Parent\_Id": "645040b4-01be-4e15-b372-9e6f339c6695"}, {"Completed\_DateTime": "", "Due\_Time": "1000", "Frequency": "Daily", "Task\_Name": "Clean Laser Parts", "Due\_Date": "20201127", "Completed\_By": "", "Machine\_Name": "Laser-3", "Parent\_Id": "645040b4-01be-4e15-b372-9e6f339c6695"}]
    - Ex 3
      * machine\_id = L1
      * DaysForward = 7
      * Returns: []

**Reporting**

* ExportHistory: <https://ss2lndqvh1.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
    - This will compile an excel file with tasks completed in the last X number of days
    - Fields included are TaskName, MachineName, Status, CompletedBy, and CompletedOn
    - There is also a count of the number of Missed and Completed Tasks
    - The url returned by this function will only remain active for up to 15 minutes.
  + Params
    - DaysBack= <days> (from 1 to +N)
  + Returns
    - A json object containing the url that will download the generated excel file.
  + Example Input
    - DaysBack = 30
    - Returns: [https://export-history-bucket.s3.amazonaws.com/TaskHistory.xlsx?......](https://export-history-bucket.s3.amazonaws.com/TaskHistory.xlsx?...........it)
* ExportMachineHistory: <https://dsm43gm3h6.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - MachineId = <id>
    - DaysBack= <days> (from 1 to +N)
  + Returns
  + Example Input
* ViewMachineHistory: <https://nsffbn6fra.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - MachineId = <id>
    - DaysBack= <days> (from 1 to +N)
  + Returns
  + Example Input
* ViewHistory: <https://jg6x1qsr11.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - DaysBack= <days> (from 1 to +N)
  + Returns
  + Example Input
* ViewReportEmail: <https://ef4lftoa1m.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - Role = <role> (must be ‘sender’ or ‘recipient’)
  + Returns
    - Json string of the email for either the sender or recipient.
  + Example Input
    - Ex 1
      * Role = sender
      * Returns: ‘rpl@clemson.edu’
    - Ex 2
      * Role = recipient
      * Returns: ‘rpl@clemson.edu’
* UpdateReportEmail: <https://1z0erbr938.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - Email = <email>
    - Role = <role> (must be ‘sender’ or ‘recipient’)
  + Returns
    - Json string notifying the caller that the email has been submitted for verification.
  + Example Input

**Tasks**

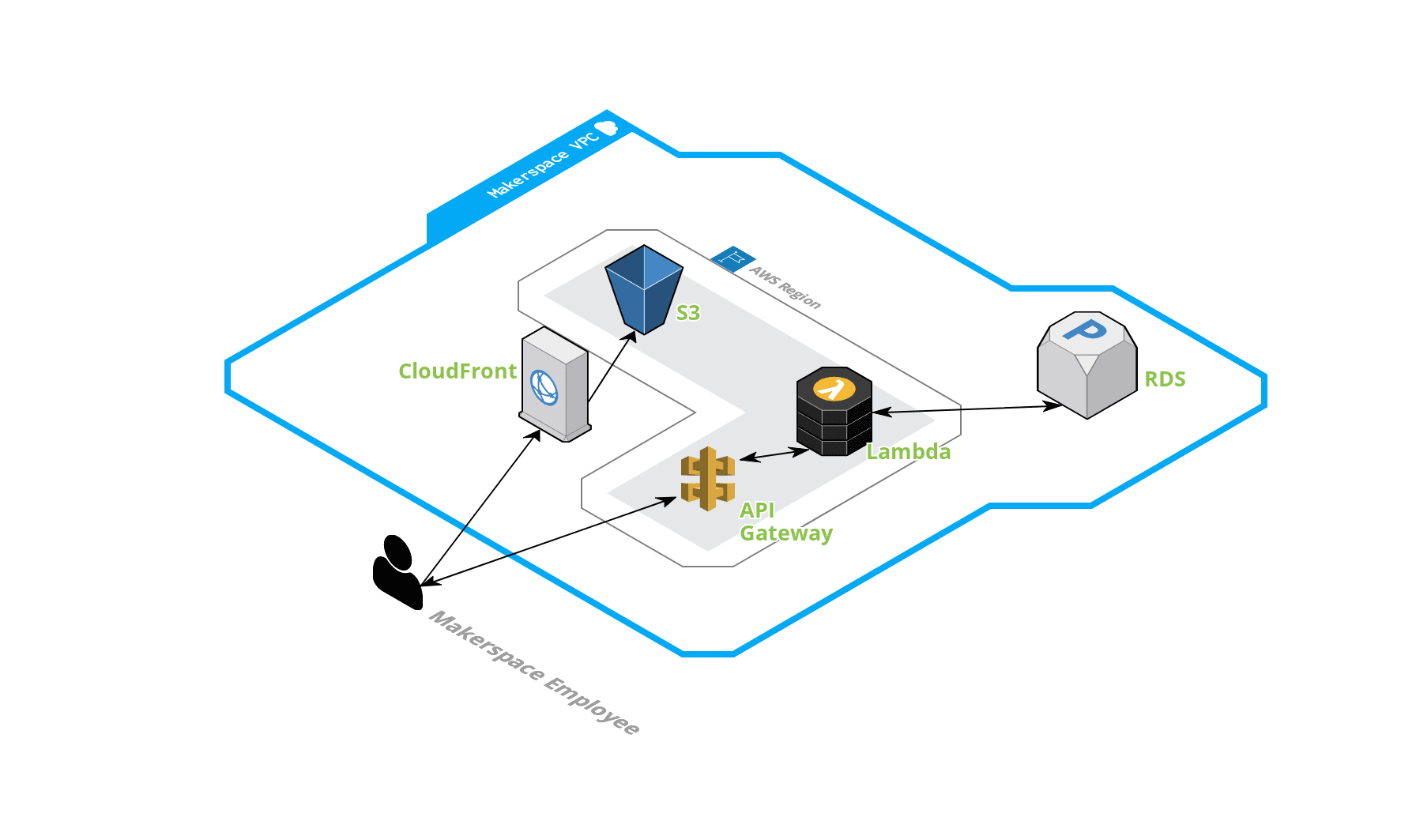
* CompleteTask: <https://5qgycj9pqj.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - ParentId = <id>
    - DueDate = <Date> (format: 20201231 = Dec. 31, 2020)
    - CompletedBy = <User>
  + Returns
  + Example Input
* CreateTask: <https://e3093oy8ih.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - TaskName = <name>
    - Frequency = <freq> (Daily, Monthly, Weekly)
    - StartDate = <date (format: 20201231 = Dec. 31, 2020) \*today or later
    - CompletionTime = <time> (0000 to 2359)
    - Description = <desc>
    - MachineName = <name>
    - MachineId = <id>
  + Returns
    - String - ParentId of Task
  + Example Input
* DeleteTask: <https://zymqwj6hed.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - ParentId = <id>
  + Returns
  + Example Input
* EditTask: <https://pckf0k3zab.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - ParentId = <id>
    - TaskName = <name>
    - Frequency = <freq> (Daily, Monthly, Weekly)
      * StartDate = <date (format: 20201231 = Dec. 31, 2020) \*today or later
      * StartDate required only when Frequency is changing.
    - CompletionTime = <time> (0000 to 2359)
    - Description = <desc>
    - MachineName = <name>
    - MachineId = <id>
  + Returns
  + Example Input
* ViewTask: <https://gdk4r3mlbj.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - ParentId = <id>
    - DueDate = <Date> (format: 20201231 = Dec. 31, 2020)
  + Returns
  + Example Input
* ViewUpcomingTasks: <https://9mgegu2fge.execute-api.us-east-1.amazonaws.com/prod/>
  + Description
  + Params
    - DaysForward = <days> (from 0 to +N)
  + Returns
  + Example Input

# **Alternate Designs**

Were alternatives considered?

Why were they discarded?

* DynamoDB over RDS



* Group by Machines > Task Categories
* Machine Types with DropDowns over All Machines listed on interface
* Give the user ability to add Machine Types
* Shifting to using Login system

DynamoDB was used over RDS due to pricing. Due to the small number of data points needed, it is unnecessary to manage a full scale RDS. With DynamoDB, it allows for automatic storage scaling without requiring users to maintain any server or keep up with any fault tolerance. Throughput is scaled with the table as well. In addition, most of our functions are for lookup queries rather than joining.

The benefits of an RDS system is that the developers of this application are familiar with SQL. It allows development to be much easier without having to learn a new tool. In addition, the power of RDS is much higher. It allows for multiple joins, multivalued indexes, and large storage.

# **Open Questions**

No open questions at the moment. (We will likely have some at the very end of the project.)

# **External Documents (PRFAQ, SIM Issues, Trouble Tickets, etc.)**

[Link to the original PRFAQ](https://drive.google.com/file/d/1kGvhQU4akRGQXekuhNhHlTS8F1fxtKIS/view) provided by our AWS mentors at the start of the project.

# **Think Big**

If there were no limits to this project, the design would have additional features that are currently considered out of scope. For example, the system would also be able to monitor machine activity, including uptime and usage which would allow for better analytics and management for the maintenance team. However, this would require the capability to connect our system to the machines. With additional time, it would allow for a trial period for the maintenance team to use the system for as long as they like and let us make as many iterations and changes as needed in order to perfectly tailor it to their uses, whereas now we only have access to limited feedback and limited time for changes once we finish the system. After a trial period more use cases might come to light that we could implement. Other design possibilities that we would add include an SSO login and the ability to schedule non-regular maintenance tasks.

Hopefully a future Capstone team will be able to build off of our framework and continue to improve the Makerspace based on the limitations of our project this semester. One potential idea is to have one all-encompassing system for all things Clemson Makerspace. It would include the current maintenance system we are working on, as well as machine details and analytics, maintenance team member details, maintenance guides for performing tasks, as well as anything else that could be used by the makerspace.

# **Potential Patents**

N/A

# **GDPR alignment**

N/A

# **Example Workflows**

Useful as an optional Appendix to the main document. An example workflow tells a small narrative for a happy path, or interesting use case that helps tie the concepts in the design document together. This is an especially useful tool for human-computer workflows of our customers, operators, product managers, or carrier delivery people.

* Signing in as an employee allows for all nightly tasks to be completed and storing that employee’s name as the one who completed
  + As an employee, I can log in, select a task to be complete from the upcoming task section then mark it as complete
* Signing in as an administrator allows for me to update task details and occurrences
  + As an administrator, I can log in, select a machine under the “Machines” section, then click edit tasks, which will display a list of current tasks on that machine, clicking on one of those task will allow me to edit its occurrence.
* Signing in as an administrator allows for me to download a CSV file of a filtered task history
  + As an administrator, I can log in, and enter the “History” section until “Others”, here I’ll be able to filter by date, completion, etc., then I can press the Export button to download the CSV file of history

# **Meeting Notes**

N/A