# **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.

# **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!

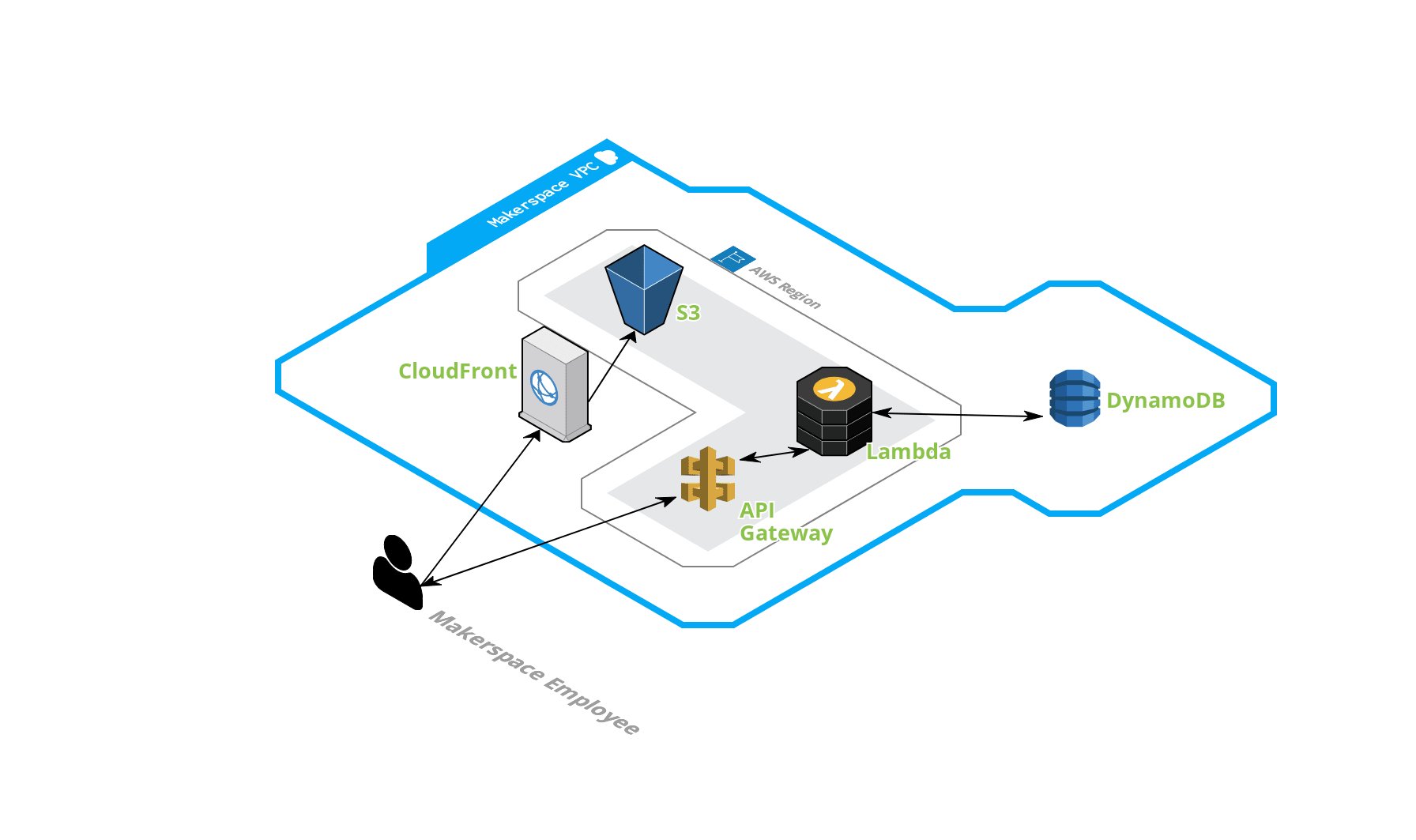
Scenario 2: It’s 7:03pm the following day. The Makerspace Maintenance Lead (Steph) gets a notification on her phone. It’s an email from the Makerspace Maintenance System saying that 3 tasks were not completed on time and are now *LATE*. It’s possible that one of her employees did these tasks but simply 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**

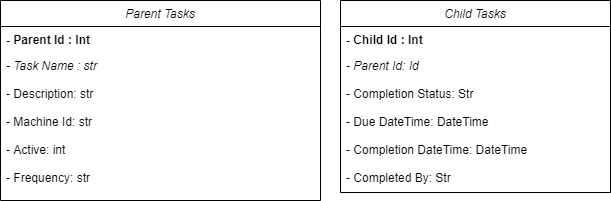
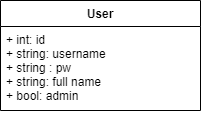
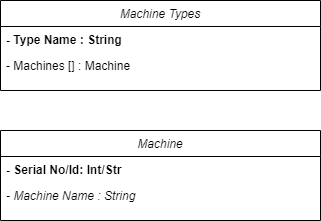
Internal/External API interfaces

Intra-package Component interfaces

Domain models

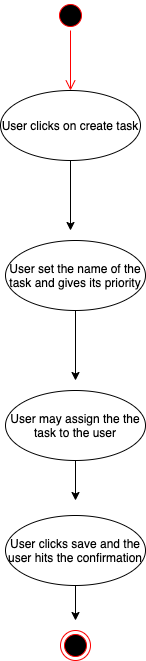
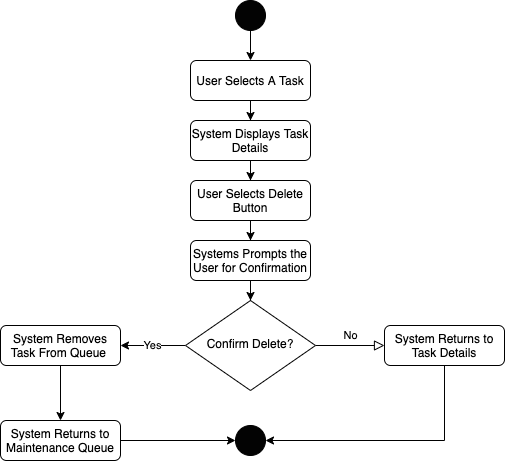
# **Details**

Class Diagrams:

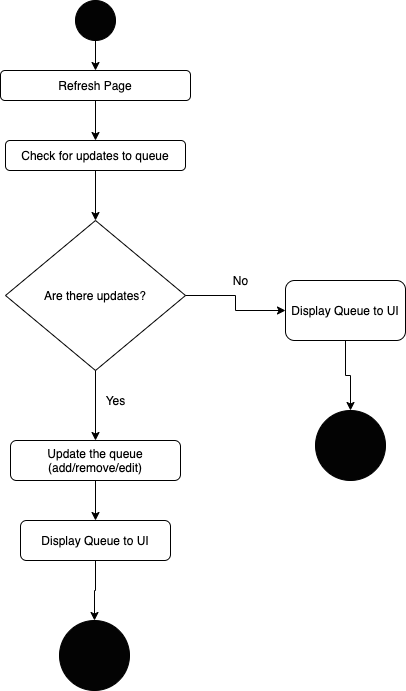
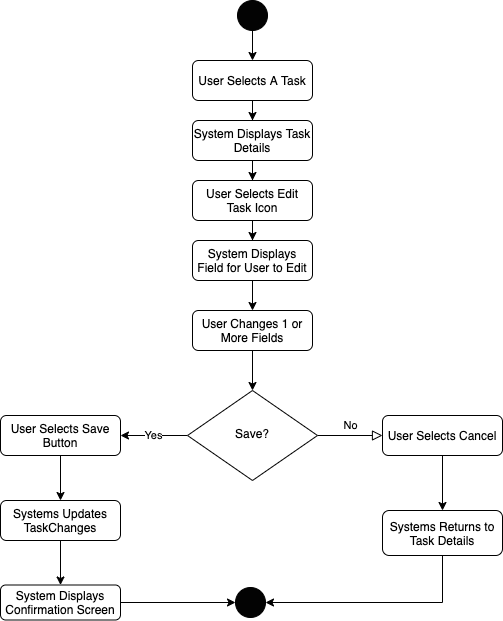


Flow Diagrams:

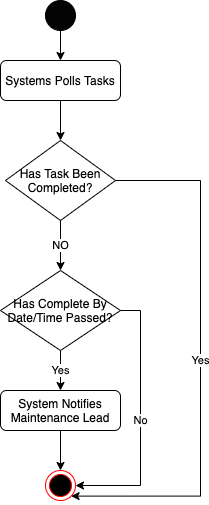
Create Task Delete Task

Edit Task Maintenance Queue



Notify Maintenance Lead



# **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?

The scalability of the project is due to expandable servers provided by AWS. Because the database is dynamic, it allows for numerous inputs into the database without worrying about storage - unless there are over thousands of gigabytes of text, which is highly unlikely. In addition, due to the database being dynamic along with separation of classes in our code, new tasks, users, administrators, and so forth could be added with ease. In addition, the administrators could add, remove, and delete tasks and people with their own will.

# **Availability**

Our design features a serverless web design utilizing AWS managed service which reduces downtime and increases availability. The AWS services chosen are available within if not all North America AWS regions. Our design is focused to be used for small teams of workers throughout the day so mass amounts of traffic is not expected. While our database lacks redundancy and read caches to speed up read/write calls; given the size of the Clemson Makerspace with projected growth database access shouldn’t be a problem

# **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**

Have you talked to our UX designer?

Note any console changes related to this design

Has a UX consultation been undertaken?

# **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. So we can…..(in progress)

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)

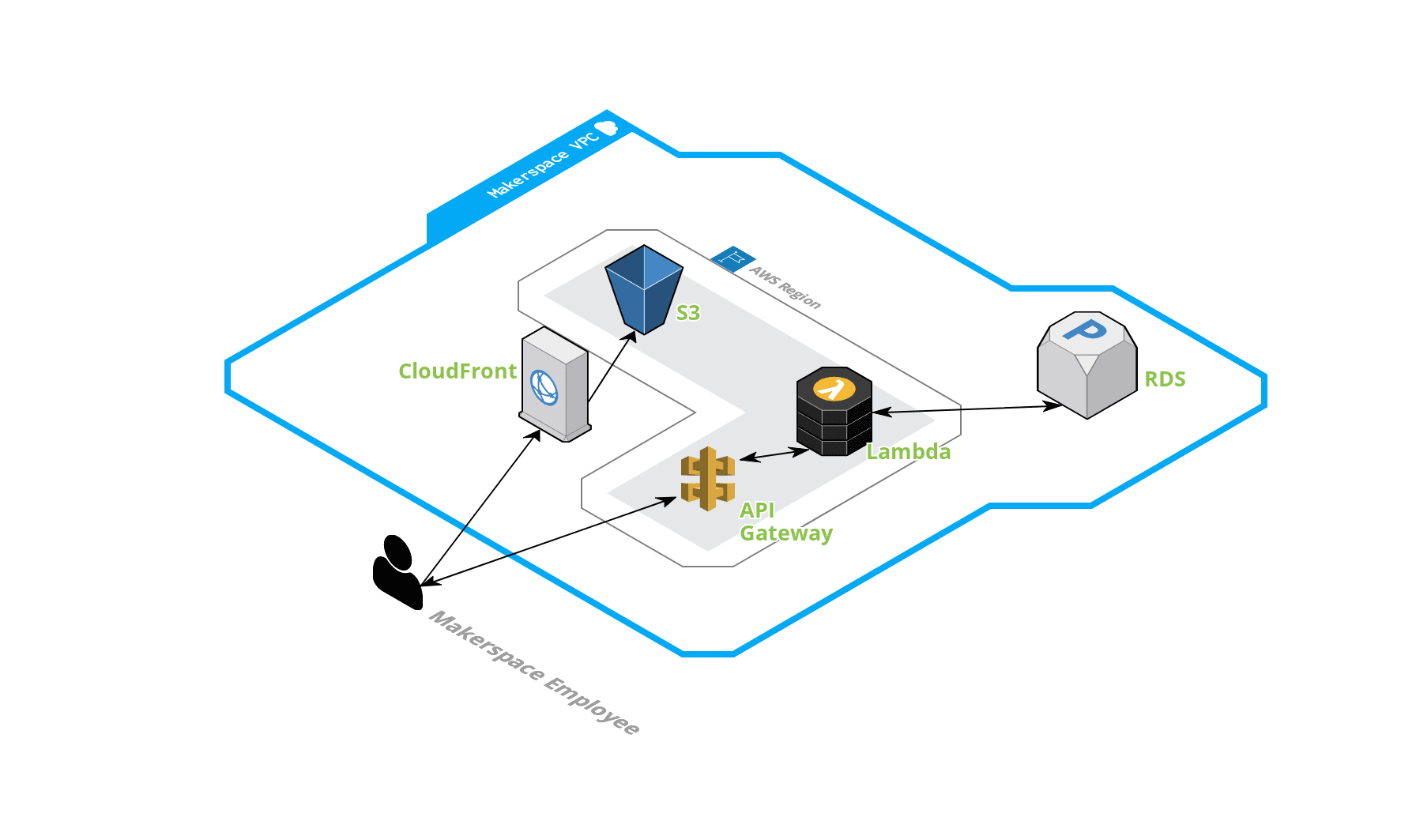
1. Setup AWS CDK
   1. Follow instructions at <https://cdkworkshop.com/> under “Prerequisites” to set up the right environment
2. Once the cdk is setup and the correct aws credentials are entered the code can be cloned from github by running “git clone <url of repository>”
3. Now, under the project home 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 application under that account.

# **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**

Don't wait to make a perfect document. Some open questions are reasonable and a design review can help fill them in.

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

Reference the original PRFAQ here so that consistency with the original intent can be validated or updated.

Include links where appropriate to give additional context to this design.

# **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**

Use this area to capture notes from prior meetings, including approvals, concerns and action items.

Capture notes from design reviews so that an audit trail is captured.l