Mines Crowdsourcing System 3.0

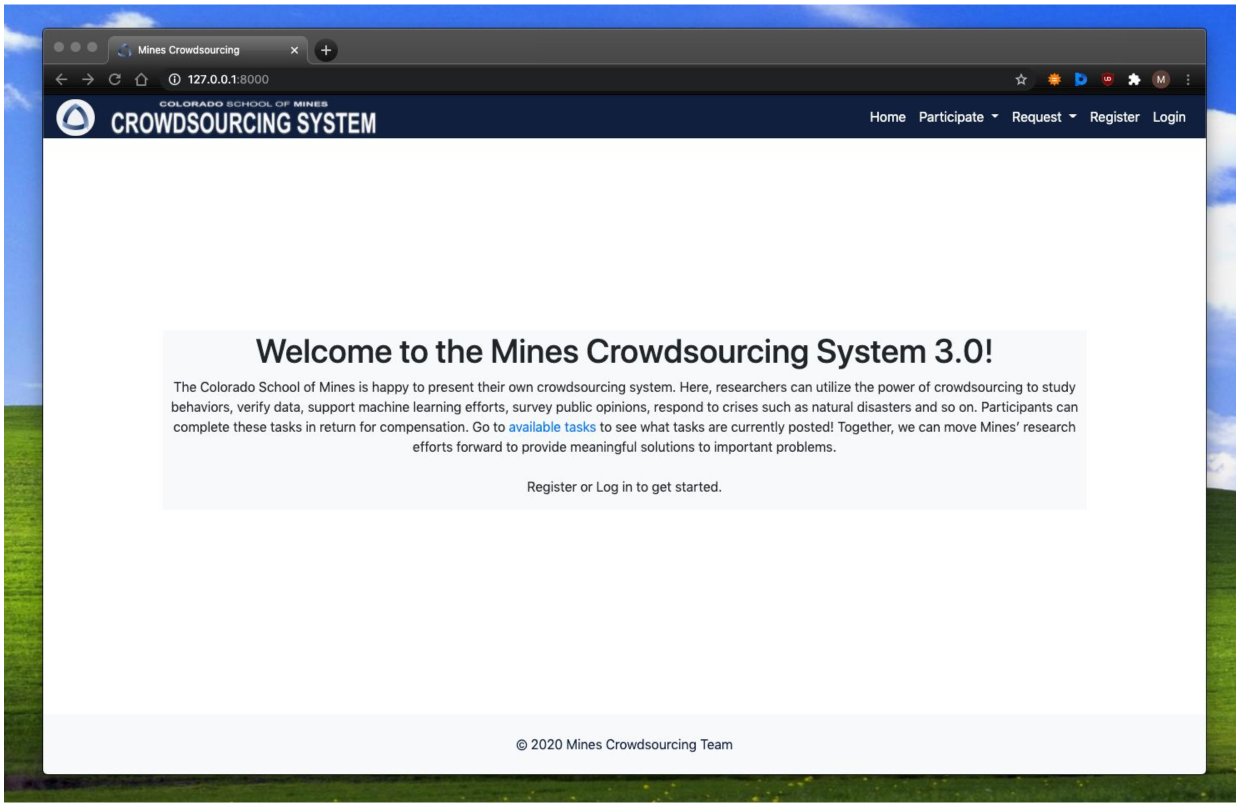
CSM Yue

Max Gawason, Jhonathan Malagon, Sophia Collins

1 December 2020

Introduction

The Mines Crowdsourcing System (MCS), introduced by Dr. Chuan Yue, is a product designed to facilitate research done by researchers at the Colorado School of Mines. Crowdsourcing is a research method that allows members of the public to contribute ideas and time to projects despite not being formally trained experts. This product presents a method for crowdsourcing that benefits administration, researchers, and participants with coded solutions for task posting and completion, participant and task approval, participant onboarding, and payment procedures. The MCS, shown in Figure 1, comes in the form of a website similar to Amazon's Mechanical Turk. An example use of MCS would be a research looking to find out more about users' internet privacy habits. They could submit a survey that participants can complete. The researcher could then look at privacy trends in the survey and research ways to improve user's privacy. The system interacts with a database which holds information regarding user information, task statistics, and fund management in a secure way. User interfaces manage views for various types of users (admin, requester, and participant), presenting registration and login pages, task management, list of task postings, task submittal, redemption for participant earnings, and an administration approval page. As version 3.0 of this field session project, our team's efforts included familiarizing ourselves with the existing architecture, framework, and features. We have aimed to further enhance the existing features as well as incorporate newly proposed requirements.



Welcome to the Mines Crowdsourcing System 3.0!

The Colorado School of Mines is happy to present their own crowdsourcing system. Here, researchers can utilize the power of crowdsourcing to study behaviors, verify data, support machine learning efforts, survey public opinions, respond to crises such as natural disasters and so on. Participants can complete these tasks in return for compensation. Go to available tasks to see what tasks are currently posted! Together, we can move Mines' research efforts forward to provide meaningful solutions to important problems.

Register or Log in to get started.

© 2020 Mines Crowdsourcing Team

Figure 1: Mines Crowdsourcing System 3.0

Prior Work

Prior work was implemented basic functionality for a crowdsourcing system. This included the following features:

- Login system
- Task requesting
- Ability to view and complete tasks
- Task approval
- Redemption page

The website had the required functionality in a local environment. However, the payment system was quite laborious and required lots of hard signatures. During this field session, we also found out the old system did not meet newly discovered requirements. Additionally, the website was deployed on a Heroku Cloud Server.

Requirements

Functional Requirements

- Implement a login system using Mines Multipass to streamline the login process.
    - This requirement evolved to allowing non-Mines students to login after discovering that they are assigned a CWID and are able to be paid by the Controller's office after submitting a W-9 form. However, they are not assigned a Multipass username/password so Multipass was no longer a requirement.
- Move the server from a Heroku cloud server to an Apache server on the Mines campus and deploy a fully functional production instance of MCS on a Mines hosted server.
    - This requirement evolved to preparing MCS to be hosted on a server provided by Dr. Yue once the server is ready via Docker containers.
- Documentation.
    - Send confirmation emails to requesters and participants after starting or completing a task.
    - Add a task statistics section for requesters to review task progress.
- Improving payment system
    - Streamline process for participants signing up.
    - Allow participants outside of Mines.
    - Streamline payments from researchers and to participants with spreadsheets.
- Participant demographics and anonymity
    - Ensure participant anonymity when completing tasks.
    - Show participant demographics from consenting participants to requesters.

Non-functional Requirements

- Ensure compliance with all rules and regulations of the Controller's office, Mines Institutional Review Board (IRB), and Colorado State and Federal Law regarding payment systems and user privacy.
  - Payment system compliance: Require users to submit a W-9 form before having access to the website.
  - Payment system fluidity: Send spreadsheets to Controller's Office instead of requiring live signatures by site admin when participants redeem their balance
  - Legal, ethical and IRB compliance for research participants: Get general information regarding gender, age, race upon registration. Assign anonymous identifiers to participants for when researchers approve participant's work.
  - Legal, ethical and IRB compliance for task postings: Present information and relevant links to requesters upon task creation for their review before submitting a task for approval by the administrator.
- Website accessibility and aesthetics
  - Comply with ADA guidelines regarding accessible websites, primarily including the ability to zoom, change font size, and activate a color blind mode

System Architecture

Website Architecture

The website architecture is a fairly typical Django architecture. The website backend is powered by Django, which is a python-based web framework. Django links to an SQLite database for storing user and task information. SQLite is a widely used embedded database framework which means it is built into the back-end server rather than the back-end server communicating with a database server. As a result, it simplifies deployment and since the database is a single file, it is also easy to back up. The website uses Bootstrap, which is an HTML, CSS, and Javascript library for elegant, streamlined web pages. Bootstrap also makes the pages mobile-friendly. The website is then run by the Nginx server.

Server Architecture

Since the server for the deployment of MCS was and still is unclear, we chose to use a Nginx server running on Docker to make it easy to deploy when a server is established. Nginx is a widespread web server. It is very scalable because multiple workers can be spawned to each deal with large numbers of connections, as shown in Figure 2. The Nginx server is run in a Docker container. Docker is a virtualization program that puts programs in containers that are relatively freestanding. This means that the Docker container can easily be moved and run on different computers and operating systems, removing the hassle of dependencies and configurations. To start the server, the Docker container is started, which starts up the Nginx workers to serve the Django website.

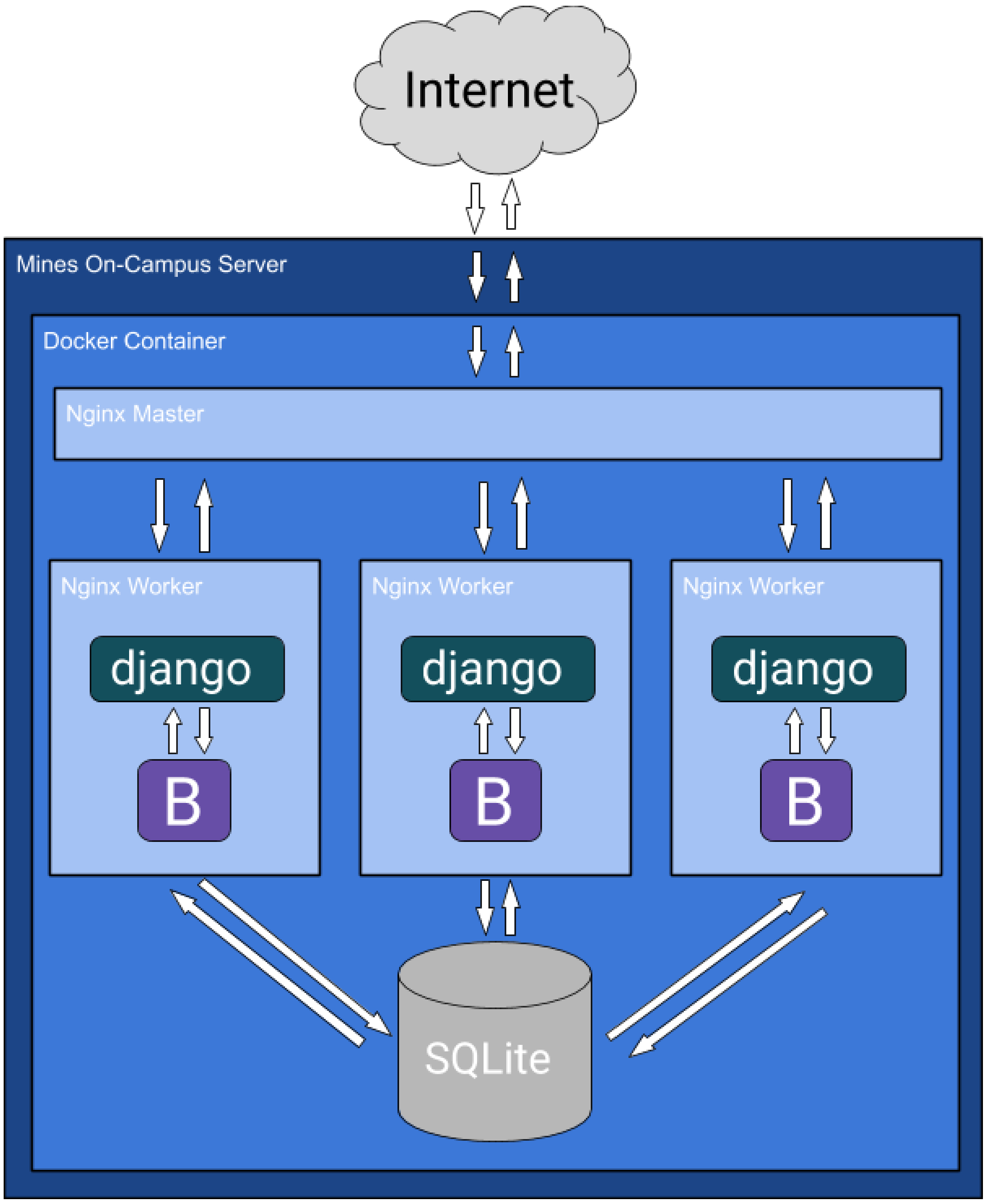


Figure 2: Server Architecture for MCS

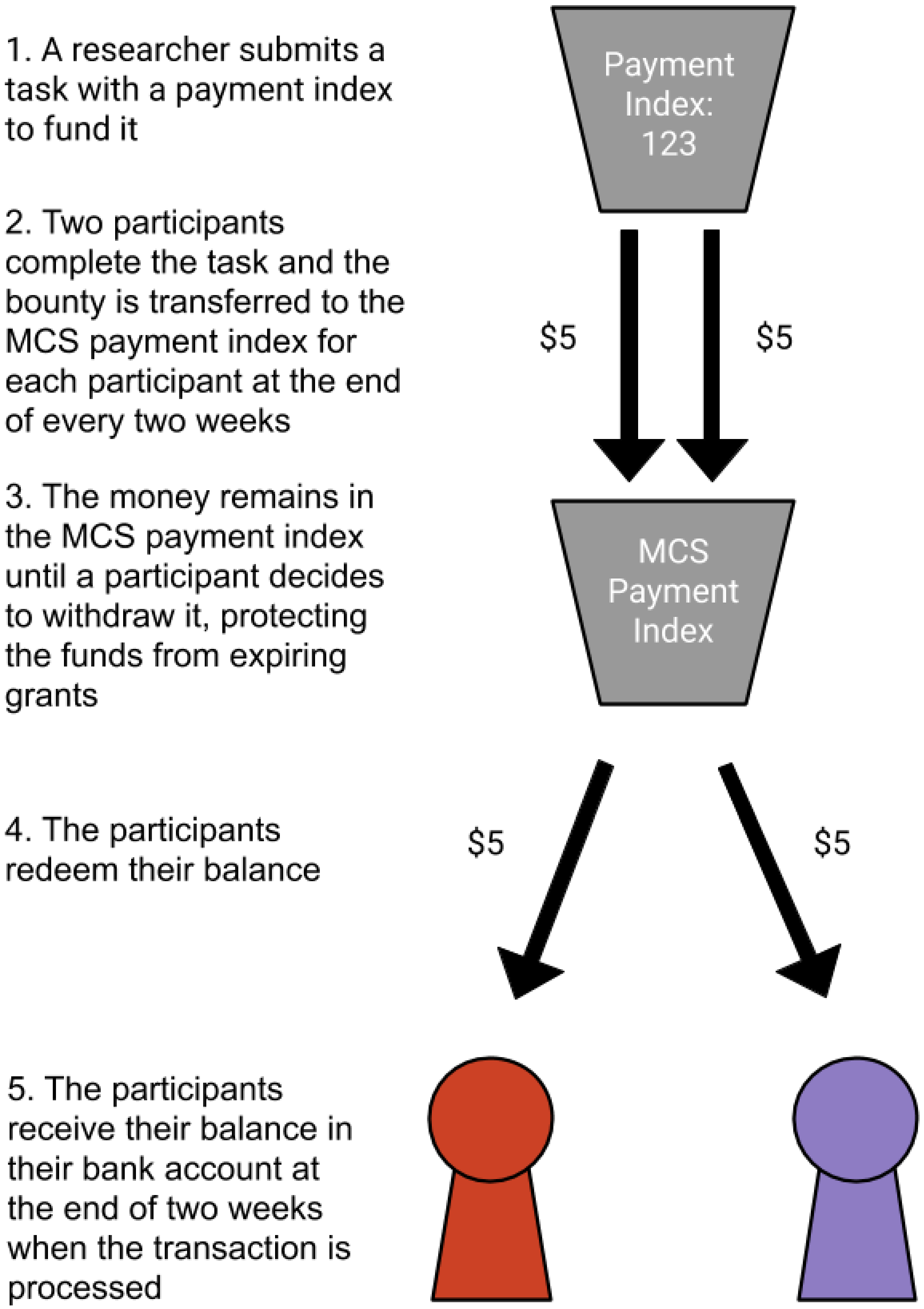
Payment System

One of the primary challenges with creating MCS was the payment system. Because the website is for researchers, most of the bounties are financed by research grants which have to be funneled through the Controller's office to pay students. The Controller’s office is responsible for dealing with payments to and from Mines. Additionally, there are a variety of constraints to pay participants because they each need a CWID, a W-9 form on file with the Controller’s office for tax purposes, and a linked bank account for direct payment. Due to some new constraints and opportunities from the Controller’s office, the payment system has been significantly changed from the previous semester’s work but still revolves around an MCS fund.

Instead of bank accounts, researchers have payment indexes with the Controller's office. This is a number where the Controller’s office holds the researcher’s funds from research grants and other funding. We have created an MCS payment index where funds can be held indefinitely. This removes the problem of expiring research grants and shifting funds that could result in nonexistent funds when a participant redeems their balance.

The process of a payment from start to finish is shown in Figure 3. When a requester submits a task, they submit a payment index that funds will be drawn from and certify that the index is valid through the end date of the task and contains enough funds to pay the maximum number of participants the bounty. When a participant completes a task and is approved, the bounty is marked as completed but not paid. Every two weeks, a scheduled job runs which compiles all these completed bounties into a spreadsheet that is sent to the Controller's office. The Controller’s office then moves each completed bounty from the researcher’s payment index into the MCS payment index. The previous payment system required researchers to submit a bulk payment when the task was posted. However, this created the problem of excess funds if a task did not hit the maximum number of participants. Also, the Controller’s office began to require that payment can only be made for a completed task, not an anticipated one.

After a participant completes a task and it is marked as approved, the bounty is added to their account. This balance can then be redeemed to their bank account after they reach a minimum balance of $5.00. When a participant chooses to redeem, their balance is marked to be paid out. Then when the payment job runs every two weeks, all the participants redeeming their balance are compiled into a separate spreadsheet. The Controller’s office will then transfer each participant's balance from the MCS payment index to their bank account. Since the bounty payment is processed first, there is no worry of overdrawing the MCS fund. This completes the payment cycle.



1. A researcher submits a task with a payment index to fund it

2. Two participants complete the task and the bounty is transferred to the MCS payment index for each participant at the end of every two weeks

3. The money remains in the MCS payment index until a participant decides to withdraw it, protecting the funds from expiring grants

4. The participants redeem their balance

5. The participants receive their balance in their bank account at the end of two weeks when the transaction is processed

Figure 3: Payment Process Diagram

Technical Design

Docker

Docker is a program to containerize applications. This sounds very technical and abstract, but the basic concept of it isn't. The best comparison to explain Docker is a virtual machine. Virtual machines allow the user to emulate a computer. This means that a virtual machine can spin up a new instance of an operating system. For example, a Windows PC could use a virtual machine to run Ubuntu as an application on Windows. As a result, the user could use Ubuntu for programming tasks to take advantage of its shell but use Windows for general web browsing and computing for its ease of use. Docker is essentially a virtual machine but for programs rather than users, essentially bundling together all the software, libraries and configuration files needed to run the program. Consequently, a container can easily be run on a variety of machines because all the dependencies are packaged together in the container. Docker also remains lightweight because it takes advantage of the features and libraries of the host operating system.

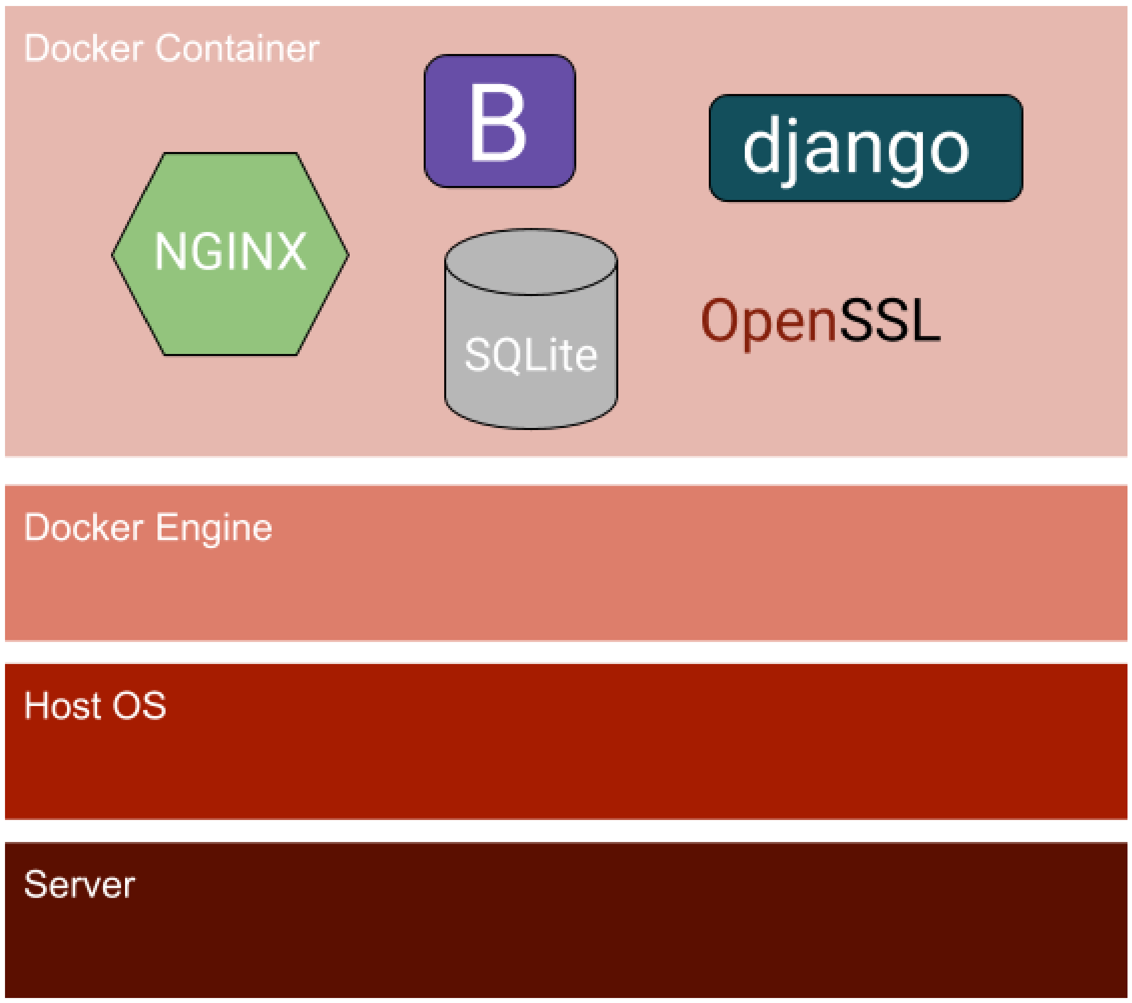


Figure 4: Docker Implementation

A Docker container consists of a Dockerfile that retrieves the necessary libraries and configures the container. A Dockerfile has similar syntax to the Unix shell, running commands to download packages, change directories and configure settings.

Containerizing MCS was extremely helpful in working toward deployment while having little information about or permissions on the deployment server. MCS's container architecture is shown in Figure 4. The website backend files are transferred into the container when the container is built. The packages like Django, Nginx, and OpenSSL are installed on the container. Next, the python dependencies are installed using pip and the requirements file. The Nginx server is then configured to use the correct certificate and listen on certain ports. Nginx is started and several workers are spawned to begin serving the website. Finally, Docker exposes a port that links to the host operating system, allowing incoming traffic. This has multiple implications for MCS. First, when a server for deployment is found it won't take much more than installing Docker and building and running the container to start the website. There won't be any configuring besides selecting the ports and making sure the server is set up to receive traffic outside of Mines campus. Additionally, the website will be extremely scalable. As the website grows, the container can be run on multiple servers to serve more participants. Running MCS in a Docker container has helped work toward the smooth deployment of MCS in the future.

HTTPS

The web is built on HTTP or Hypertext Transfer Protocol which is how websites are sent over the internet. It includes methods like "GET" and "POST" which are used to get a webpage from a server or send form data to a server. HTTPS builds on this by adding security; secure is the added "S". It uses end-to-end encryption in order to securely transfer data over an insecure connection. This prevents attackers from stealing things like credit cards or passwords. To encrypt the data, HTTPS uses a combination of public-key cryptography and symmetric cryptography along with certificate authorities to ensure secure communication. Certificate authorities are trusted organizations that verify a website is actually who it says it is. A secure connection is established by the following process. This process is described in Figure 5.

1. The client attempts to access a website
2. The website responds with a certificate containing its private key and its certificate authority
3. The client verifies that the certificate was signed by the certificate authority and therefore can trust the public key
4. The client creates a new secret key then encrypts it with the websites public key
5. The website receives the secret key and now the client and website can securely exchange information with the secret key

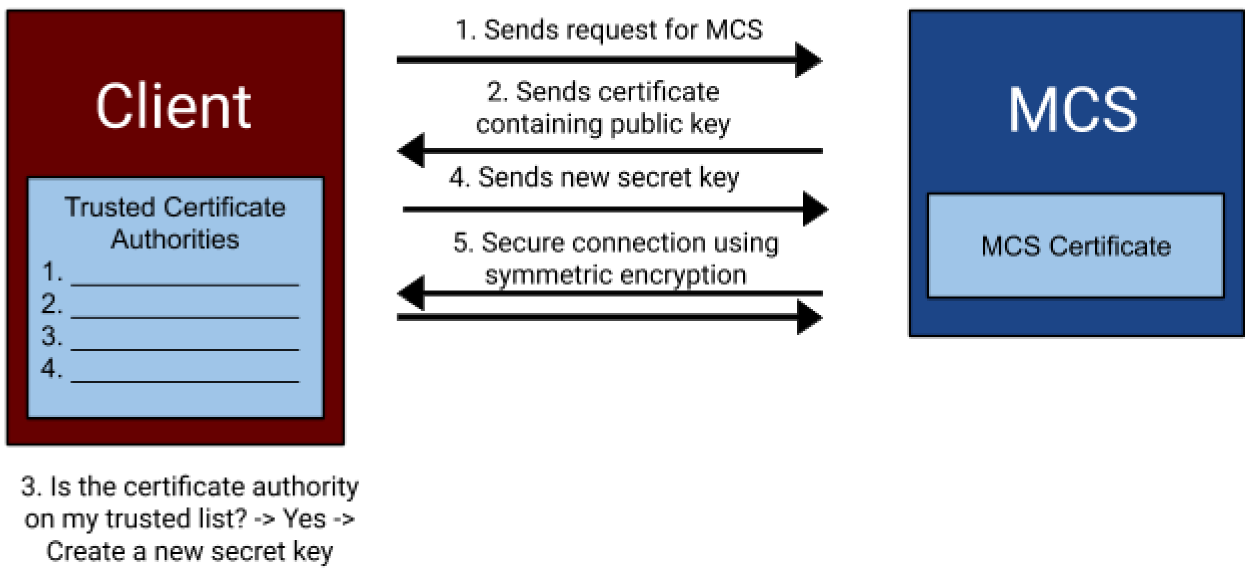

Client
Trusted Certificate Authorities
1. ______
2. ______
3. ______
4. ______
1. Sends request for MCS
2. Sends certificate containing public key
4. Sends new secret key
5. Secure connection using symmetric encryption
MCS
MCS Certificate
3. Is the certificate authority on my trusted list? -> Yes -> Create a new secret key

Figure 5: MCS HTTP Request Lifecycle

Because MCS requires user information and a password, it is important to guarantee that the user's information is secure. In prior versions of MCS, HTTPS was not a large concern because it is standard on most web hosting services. In moving to a Mines server, the lower level details of HTTPS had to be considered. Implementing HTTPS on MCS required correctly configuring Docker and the Nginx server. First, Nginx must be configured to accept HTTPS connections. It also must have a public key and certificate to talk to the client. Since we did not have a production environment to use a real certificate authority, we had to create a self-signed certificate, a common practice in testing environments. A self-signed certificate is exactly what it sounds like, a certificate where the certificate authority is yourself. We used OpenSSL to generate our self-signed certificate. Nginx then uses this certificate whenever a client connects over HTTPS. In the future, Nginx can be configured to use a certificate authority like Let's Encrypt which is widely used and trusted. Finally, Docker must be configured to accept HTTPS connections on a separate port than the HTTP port. This allowed us to connect to MCS locally over HTTPS and will be used in future implementations to communicate with participants.

Quality Assurance

Design

- When designing MCS, there were a variety of constraints to adhere to. First, when improving the payment system, we had to consider all of the regulations of the Controller's office. For example, every transaction must be a billable action meaning funds can't be transferred out of a research grant unless a participant has completed a survey. There were also FERPA regulations to comply with student privacy. When approving participants, the student's name or CWID cannot be known by the requester. As a result, we had to implement anonymous IDs to identify the participants.

- Another design consideration of MCS is building on the existing codebase. Since we were continuing a project that had already been worked on by another team, we had to work with the existing code and database schema to add features. For example, with the new payment system, users are not paid instantly so some transactions need to be held in purgatory until a spreadsheet is compiled and sent to the Controller's office. To implement this feature, we had to expand on the database schema with an elegant fix rather than hacky code.

Testing

- User Interface Testing: We primarily relied on manual user interface testing. To add some rigidity to the process we tested the interface with various workflows. An example workflow would be a requester signing in, clicking to create a task, filling out the task, and submitting it for approval. As a result, we are able to ensure all workflows work correctly. Additionally, we tested certain links such as going back a page. Since we were able to test both workflows and links, we are confident that the website properly flows together
- Integration Testing: Integrating the website required two pieces: the Django website and the Docker container using Nginx. The website was working locally from the start of the project. From there we were able to prototype a basic container with Nginx to serve the default Nginx page. Then we were able to successfully add the website to the container with ease.
- User Acceptance Testing: Since we were unable to successfully deploy MCS, we were unable to perform widespread user acceptance testing. However, we remedied the situation by having friends test the website from different perspectives. We asked some friends to create a simple task like sampling participants' favorite ice cream. Then we approved those tasks and handed them off to others to complete the tasks. From there we observed the system ensuring that the correct transfers were made and spreadsheets were properly written in addition to collecting users feedback. The feedback was mostly positive, pointing to Bootstrap's intuitive interfaces. The main complaints were about the concept of crowdsourcing and the payment system. A future requirement should be to explain the purpose of the website and the concept of crowdsourcing better possibly through diagrams or videos. The payment system, as unwieldy as it may be, however, is relatively fixed due to the constraints of funneling payments through Mines.
- Website Performance: Again, due to server limitations we were unable to perform the Google PageSpeed Insight test to test website performance. However, using the Docker container, we were able to get a feel for the performance of MCS. Since the primary bottleneck added by a real production server will be the network performance, which is out of our control, serving pages through the container will give a relatively good insight

into page speed. All pages were able to load instantaneously to the human eye and there were no long waits for database queries.

- Static Code Analysis: In order to assure clean code, we followed the PEP 8 Styling guide which dictates how python should be styled. Additionally, we used Pylint to detect any basic errors or duplicated/extraneous code. Together, these tests helped maintain a clean codebase.
- Website Security: Website security is a vital part of the MCS platform. First, we used Django's internal checks to make sure that basic security options are set. For example, there is a DEBUG option that must be turned off when deploying the website so that stack traces are not shown when there are errors. Next, we made sure that basic security procedures were in place like password hashing. Additionally, we implemented HTTPS so that participants' information is secure.

Results

Rich Findings

For this project there was a great deal of diplomatic relations that had to be established between the Controller's Office, Human Subject Team and Mines ITS. Through these relations the team was able to make vital discoveries for the project. These discoveries include:

- Colorado School of Mines can pay out funds to entities that are not affiliated to the campus, which was previously believed to be impossible
- Surveys and tasks must be approved by the Human Subject team
- Many funds from departments have an expiration date to be consumed
- Funds can only be transferred for billable events (e.g. a survey must be completed before the reward can be processed)
- The Controller's Office will accept spreadsheets for payments without a hard signature
- On-campus servers, especially during a pandemic where computing resources are stretched to their limit, can be hard to find

Features Completed

Even though the team had experienced various setbacks there were several accomplishments throughout the time given:

- A login system for non-Mines students was established
- MCS set up on a Nginx server in a Docker container and ready for deployment
- Anonymity for surveys was made possible
- A spreadsheet with students who have completed tasks and decided to redeem earnings is generated and sent to the Controller's Office on a bi-weekly basis

- Website aesthetics was revamped
- User registration page was updated vocalizing the need for a W-9 to be submitted to the school and steps to complete this process
- User registration page updated to acquire basic information that may be vital for surveys: Age, sex, gender, ethnicity, and major
- Task requesters are now given a statistics table for each task they have published with percentage completed and other useful resources.
- Task requesters can view participant demographics from consenting participants

Features Not Completed

In this project there were various aspirations and goals. Due to the nature of this unique semester within a pandemic, communication and clarity was often hard obtained. Goals for this project were constantly refined and changed. One of the major goals stated for this project was to have the website up and running for public access. Various complications spawned while working on this aspect of the project:

- The availability of a server was given halfway through the semester
- Superuser privileges were only granted to Dr. Yue due to ITS restrictions
    - Any command requiring superuser privileges like installing packages or running Docker had to be run through Dr. Yue, slowing down the deployment process
- Two weeks before the deadline there was an unrelated server failure (many of Dr. Yue's graduate students use the same server)
    - Team permissions were revoked, and a live deployment was no longer a benchmark

The other major feature that was not implemented was the Mines Multipass log in. This feature was scrapped because our client wanted the Mines Crowdsourcing System to be accessed by participants that may not be affiliated to Mines. There was a significant amount of work put into implementing this feature before it was abandoned.

Lessons Learned

Many of the group members were introduced to software such as databases, frameworks, styling sheets, and HTML. This was a substantial learning process that took a large amount of time along with understanding a large code base. There was also a complication with getting our project run which took about 3 weeks for all the team members to have the project up and running successfully. Lesson learned here was that it is vital to make sure that a project can be accessed and manipulated by various operating systems and documentation is also important which we are providing for the future iterations. We also learned how to communicate with a variety of departments and the difficulty of coordinating them. Additionally, we had to bring the department members up to speed about where the previous team left us.

Appendix

Future Tasks

- Get the website running for the general public to access
- Multi-factor verification for account security
- Integrate a point system where gold star workers can be recognized
- Have a way for requesters to make certain tasks available privately to specific users
- Add a dialog channel where requesters and workers can communicate

Installation and Use

Installing and running MCS locally is fairly straightforward. This tutorial assumes that python3, pip, and Docker are installed and the Docker daemon is running. First, clone the git repository and change it to the current directory. Now, the requirements have to be installed in a virtual environment.

```
$ virtualenv .venv
$ source .venv/bin/activate
$ pip install -r requirements.txt
```

The next step is to correctly configure Django. Run the following sequence of commands which creates the database model, creates the database, and creates a superuser. These commands must be run before the Docker container can be built.

```
$ cd minesCrowdsourcing
$ python3 manage.py makemigrations
$ python3 manage.py migrate
$ python3 manage.py createsuperuser
```

The Django backend is now set up and can be run locally.

```
$ python3 manage.py runserver
```

MCS will now be running on http://127.0.0.1:8000 in your browser. Quit the server using Ctrl+C.

To run the server in a Docker container, move back to the main MCS folder with the Dockerfile. Before the container can be built, a self-signed certificate must be created.

```
$ openssl req -newkey rsa:2048 -nodes -keyout self-signed-certs/mcs.key -x509 -days 365 -out
```

```
self-signed-certs/mcs.crt
```

Follow the instructions to create it. Now the container can be built and run with the following commands.

```
$ docker build -t mines-crowdsourcing-app .
$ docker run -it -p 9601:9601 mines-crowdsourcing-app
```

This builds the Docker container and then exposes it on port 9601. It also requires access to port 9600 for gunicorn to run. To access the website access https://0.0.0.0:9601. Quit the server using Ctrl+C.