The Assignment and Budget Tracking Program

The Assignment and Budget Tracking Program was designed to allow a small business to keep track of tasks that they have assigned to their employees and or volunteers. It contains several parts that work together to allow for the efficient management of business resources both human and financial.

The Task Module

The Task Module is at the core of the application. It allows users to see their tasks and report on each tasks progress. It contains features to allow users to report on monies spent and upload receipts for those expenditures and images of their progress. The Module also gives managers and administrators the ability to view the progress of all tasks either on a team basis or an individual user basis and supervisors to view the status of all tasks for users assigned to teams they are part of.

The Team Module

The Team Module allows for the organization of users into work groups that can have work assigned to the group instead of the individual. Team members can see the contact information for all members of the teams they are apart of as well as tasks assigned to the team. In addition, supervisors can see all the tasks assigned to all the members of a team so that they can judge the workload of the team members.

The Accounts Module

The Accounts Module allows Managers to view funds assigned to tasks in the application as well as what has been spent and allocated for different purposes. The accounts module has safeties in place that will not allow funds to be allocated or deallocated from different accounts unless they are available to be moved or released. In addition to accounts there are also sub accounts that provide a way to sub divide the resources of an account and reserve them for a project even if tasks associated with those funds have not been generated yet.

The Users Module

The Users Module allows managers to create and view all users in the system. The Users Module gives access to all users in a place that managers can update profiles, assign work, as well as manage teams that users are assigned to.

The Filter Module

The Filter Module is a top-level filter that effects what is viewable in every other module in the application. The Filter Module allows users to filter tasks by their status, Teams by name, and Users by name.

The Profile Module

The Profile Module gives a convenient spot for supervisors, managers, and administrators to see only their own profile and teams they are a part of as well as the tasks assigned to them and their teams.

The Business Need

The Assignment and Budget Tracking Program was designed because although there are other applications that can do everything that the application does all tested applications fell short of the users’ needs as outlined below.

* Existing task management programs such as Trello and Monday.com give a way to track assignments but they are too modularized for individuals and teams.
* Existing accounting software such as QuickBooks and Quicken provide a complete accounting solution but lack integration with task management software.
* The cost associated with a bigger enterprise solution was out of the question.

With these factors in mind a request was made for a lightweight piece of software that can keep track of tasks, receipts, and basic accounting as well as organize workers into different functional teams.

The Proposed Solution

The proposed solution is a Micro-services web application written in Python and JavaScript consisting of four services. That are containerized with Docker and run with Docker Swarm

The Front End

The front-end service leverages the ReactJS library created by Facebook to render a single page user interface from components that will be simple to maintain and extend as the business needs change and grow over time. Presentation is formatted with react-bootstrap and CSS to make the application user interface reactive to different screen sizes.

The Back End

The back end api manager is written in Flask-Restx. Flask-Restx is a wrapper for the Flask micro-framework that in addition to serving all the information the front-end needs create swagger documentation for the api quickly and simply so that the application routes can be easily viewed, maintained, and extended without duplicating work. The back-end leverages jwt tokens to provide security and protect the application from unauthorized access. Data management is done with Flask-SQLAlchemy to merge the handling of objects and database management. The python unitttest library will be used for unit testing the application and coverage.py will be used to test the coverage of the unit tests. The back end will be the only container that will be given access to the database. This will give an extra level of security to the application since the flask application will have control of all the requests that make it to the database in addition the flask application will only respond to requests that come from the front-end application’s domain. Since all traffic will be encrypted this should prevent unauthorized users from being able to successfully make api requests.

The Database

Data application data including Users, Accounts, Teams, Tasks, and any association tables will be stored in a PostgreSQL database. PostgreSQL is a free open-source database that is still suitable for enterprise relational data. This will allow the users to have a solid data storage and management solution without having to pay storage licensing fees for their storage solution. In addition, if the data grows to the point where they outgrow the docker container that the data will be stored in the database will be easy to migrate to a could provisioned version of PostgreSQL.

The Reverse Proxy

All of the other services will be hidden behind a NGINX proxy server that will handle all web traffic. This will give a much greater level of security to the application because none of the services will be directly accessible outside of the reverse proxy service. NGINX will handle upgrading all http connections to https connections to prevent leakage of sensitive information.

The Deployment

Application Hosting

The users’ need a way to access the application from multiple work sites in a secure way. They don’t have the local infrastructure to host the application on their own servers without creating a security concern on their network, so they have opted to have the application deployed to a cloud hosted virtual machine. An Amazon Web Services EC2 instance was selected as the appropriate provider and service for the task because it will give them the most local data center to their current operations as well as having an easy to manage and understand management console.

Deployment Details

The application is going to be deployed on a single EC2 instance with docker swarm. This allows for all environment variables and configurations to be saved in a single yaml file. The built images will be stored in a docker hub repository and docker will be able to download the images as well as any updates to the application. The application data will be stored in docker volumes that will protect the data they store by not allowing the data to be accessed outside the containers.

The Security Plan

The security plan is mostly illustrated elsewhere in this document but summarized here for a complete view of the steps that were taken to protect the application in a complete summarized manor.

Docker

All the services are stored in docker containers, and their associated data is stored in docker volumes that are not accessible from outside the containers.

Docker Swarm

The containers are managed by Docker Swarm. This allows for the configuration of Docker networks where that prevent the containers from communicating with other containers unless they share a common network. The only container that has any connection ports exposed to traffic outside of the docker networks is the NGINX proxy service, this prevents unauthorized connections to any of the containers. The applications networks are configured as follows

* Frontend – the front-end network is accessible to the NGINX container, back-end container, and front-end container
* Backend – the back-end network is accessible to the NGINX container and back-end container.
* Database – the database network is accessible to the back-end container and the database container. This stops anyone from accessing the database without first going through the protections built into the flask application

NGINX reverse proxy

The NGINX reverse proxy encrypts all web traffic as well as handles forwarding all requests to the front-end and back-end containers. This prevents any unencrypted communication with the api service.

JWT Tokens

All the routes in the flask API except the login route require JWT tokens of the appropriate access level in order to access the information served by the particular api route. To get a valid JWT Token the user must provide their username and password through an encrypted https connection from the front-end application. The JWT Token encrypts user information and stores it as a cookie so that it will be sent to the api with each request. The private key is not provided to the client so the data cannot be decrypted outside of the flask application.

Password Storage

Passwords are not stored in the database in clear text. They are hashed with a sha 256 hashing algorithm so that even if there is a breach of the application users’ passwords are not exposed.

Application Secrets

Any sensitive information that the application needs to operate is stored in the docker-compose.yml file. This way the information is not stored in the code or repository but is instead shared with the application as environmental variables. This also allows the secrets to be different in development, testing, and production environments.

Protected Ports

The only ports that will be exposed from the Amazon Web Services Virtual Private Network are 80, 443, and 22. Ports 80 and 443 are http and https and will be allowed to be accessed by any valid ipv4 or ipv6 address. They are the two ports that will provide access to the application. Port 22 is ssh, it is used to manage the EC2 instance and will only be accessible from the web administrators ip address and even then it will only be accessible with a pem key file.

The Development Plan

The project development will follow a modified agile development methodology that will allow the developer to get feedback from the users throughout the development of the project. This will allow the developer to react to the needs of the users and get feedback on actual design and usability as development progresses. Development will be divided into three major phases they will include.

* Design and proposal. During this phase sample drawings of the features will be provided to the clients for feedback. In addition, the data model and api will be designed.
* Development. During the development phase features will be developed one at a time as much as possible and user feedback will be requested with each completed feature. If modifications are needed they will be completed then resubmitted for user feedback. Some system tests may be requested at the conclusion of major feature groups so the users can get a feel of how the entire system works together. Unit tests will be created during alongside of the features they accompany so that every change can be validated not to break the rest of the application
* Testing and delivery. During the final development phase, the users will be trained on the application. They will then be asked to extensively test and use the application and give feedback on its performance. Providing the application meets user expectations the development and testing environments will be shut down and the production environment will be activated.

The Testing Plan

Testing of the application will be done in four ways. First unit tests will be developed alongside the flask api code to verify the code performs the function it was designed for. Second, the developer will perform integration tests on each feature as it is completed. Third both the developer and the project owner will carry out end-to-end tests of sub systems as they are completed to verify major portions of the application meet user needs. Finally, a team of users will carry out end-to-end user acceptance testing where they will user the entire application for a period of several days to ensure the application performs as intended.

The Development Timeline

Project development was only projected to take a few weeks originally to complete. However, with the limitations of volunteer labor for feedback and unforeseen delays to development including a drastic under estimation for coding time the development took several months to fully develop. The phases of development and milestones were as follows.

Design

Design of the system started the beginning of June 2021 and concluded the third week of June. The milestone for completion was the acceptance of the design sketches that would be the template for the project.

Development

Development was sub divided into five portions: environment and authentication, users function, Accounts function, teams function, and tasks function. Originally it was planned for this entire portion of the project to take under two weeks, so they were not planned individually, the process took just over 3 months. Because of the intertwined nature of the portions, they were developed alongside each other as features from a partner portion was needed. Development began the third week of June. Environment and authentication concluded the first week of July and was marked by the containers all functioning in the development and testing environments and the ability for a user to sign into the application. The users, and teams portions of the application were completed the second week of September and marked by the ability of a user to see and modify all the team and user information they were supposed to have access to. The accounts and tasks portions of the projects were the final portions to be completed because they relied the most heavily on the other modules. They were completed the final week of September and the associated milestone was users being able to fully track and update all the tasks they were supposed to have access for as well as full functionality of managers to see account performance and transactions. After completion of the tasks and accounts portions the environment had to be revisited to move the application to it’s final cloud environment. Development concluded the last week of September. The development portion of the project concluded with a fully functioning application.

Testing and delivery

Since the users were involved in the entire process final testing and delivery were completed quickly during the last week of September. The milestone for completion was an accepted project in the live environment.

The Environments

Three separate environments will be used for this project. They include the dev environment where the actual production will take place, the testing environment where the developer can see the progress in an environment that is verry similar to the live environment, and the live environment where the application is accessible to the end users. The details of the environment as well as the costs associated with the environments are detailed below

The development environment

The development environment will reside on the developer’s personal computer. It will consist of visual studio code, docker desktop, and the Windows subsystem for Linux. There is no cost associated with this environment because the developer already owns the environment and there are no licenses that have any cost needed for this environment.

The testing environment

The testing environment will reside on a virtual machine on the developer’s home server. It will consist of docker, docker-swarm, and the operating system will be ubuntu server. There is no cost associated with this environment because the developer already owns the environment and there are no licenses that have any cost needed for this environment.

The live environment

The live environment will reside on a virtual machine in the AWS EC2 service. The environment will be an AWS Linux image with docker, and docker-compose installed. The service charges per hour of use and will work out to about $10 per month if the environment is active all the time. In addition the users want a dedicated Domain to be associated with the environment this will be an extra cost of around $0.10 per month for the route 53 request service and around $30 per year for registration fees.