

Final Project

Worth 33% of your final grade.

**Introduction:**

1. We are going to be building a blockchain ERP system.
2. This is a creative project, as long as your project can do the basics, I can run it on my computer, and I can look at your code you can score well
3. You can use any language you deem appropriate, you may even build on top of existing systems, like solidity.
4. Download the Project Specification sheet, you will use the same sheet for the submittal.

**Submittal Guidelines:**

1. Submit your Project Specification Sheet, along with your codebase zipped up.
2. Don't assume I already know how to run your code base, please provide a readme.txt within your zipped directory explaining how to run your code.

**Tips:**

1. Recommended Activities:
   1. Complete Quizzes 1,2,3
   2. Review all the material available in the CANVAS, and material you can find online to help you.
2. Feel free to use google, wikipedia, and other resources to enrich your understanding.
3. Write the reflection answers in your own words, if you score more than 40% in turn-it-in plagiarism checker, I will deduct 4 points.

Project Specifications

Andrew Millward, axm6090@psu.edu

# Intro

In this project you will be creating a Blockchain ERP for a bank. In this prompt you will need to:

1. Highlight what features need to be implemented and why.
2. Write technical specifications of how you are going to implement it. (Methods, features, classes, Inputs, outputs and processes)
3. Document the Technology Stack that you will be using, and create diagrams if possible, about what you plan to build.
4. Document Backend, Front-end, what is client-facing.
5. Document how the client (A worker in the Bank) will use the technology.

# Features of an Enterprise Resource Planner

[In this section please complete the feature summery, use online resource to fill in the gaps, I have started the feature overview. You need to expand and finish it. Add 6 total top-level features, and then describe what the feature is meant to do.]

1. **Financial Information Tracker:**This feature should track the bank’s total payroll, revenue generated, generate expense reports for various departments. It will contain two blockchains to track both expenses and revenue and then can access each of those chains to calculate different necessary values such as the net revenue once expenses are factored out. All blocks will also be added to the master blockchain whenever added to these local chains. As a whole, this feature should be implemented since it will help to track all financial transactions underwent by the bank with large amounts of control in terms of storage and retrieval of information.
2. **Standardizing HR information**This feature should track how many employees are in the bank. You need a method to add new employees along with their [salary, department, supervisor, employee-id], it should also be able to track corporate expenses per employee. This will track basic HR information using two blockchains, namely one for employees and one for complaints received to human resources from one employee to another. Complaints will allow the resource planner to hold basic human resource information about quarrels as a bookkeeping measure for the Bank’s entire blockchain system. All blocks will also be added to the master blockchain whenever added to these local chains. As a whole, this feature should be implemented since it standardizes the entire human resources department down to a few basics relating to managing employees and complaints effectively in one centralized system.
3. **Standardizing Bank’s Clients Information**This feature should track a bank’s client, [bank number, number of accounts, type of accounts]. Also you need to implement a calculation to generate revenue. Make a method that uses information from a client and calculates revenue. This portion will also include one blockchain to store client information as mentioned earlier that can be added, removed, and retrieved in the form of revenue or the client in general. All blocks will also be added to the master blockchain whenever added to these local chains. This feature should be implemented since it will allow the bank to keep track of its clients, their balances, and anything else of interest while making access to this information very straightforward.
4. **Supply Management Information**

This feature should manage resources owned by the bank around the business in order to track internal supplies for anything that may need resupply soon. For example, this would track paper usage so that it is easier to check supplies if more needs to be ordered. To accomplish this task, we will be using one blockchain to store a cumulative list of all supplies and usage for the bank in order to improve access to this information. All blocks will also be added to the master blockchain whenever added to these local chains. This feature is important for the business because critical supplies, although not often thought about, are important to keep in supply in order to run a functioning business.

1. **Project and investment management**

Manages various projects and investments made by the bank to better control and follow project supervisors, workforce allocation, and budgets for use also in financial and payroll information. The purpose of this for the bank is to track potential investments managed by supervisors and also tracking any project or expansion the bank may be undergoing which will help to keep this centralized so that any project query can be done from one location. Projects will be added to a local blockchain to maintain a closeby list of everything that is being done. All blocks will also be added to the master blockchain whenever added to these local chains. As a whole, this feature should be added to the project since it helps to effectively keep the bank on top of any investments while keeping those supervising them accountable for their performance in the task.

1. **Employee Scheduling Management**

This feature would be a cumulative schedule for the fiscal year, tracking what days each employee is scheduled to work. As a part of this section, employees will be able to add days, drop days, and swap days with other employees and retrieve information about their scheduled work time. It will be implemented using a blockchain to store all of the scheduled dates. All blocks will also be added to the master blockchain whenever added to these local chains. As a whole, this feature is a necessity for this project since it strongly complements some of the other HR features to created a dedicated access portal for employees to check their information, allowing for potential improvement in the future where only HR reps can access the HR portal and employees can access both.

# Technical Specifications

[In this section of the document you need to convert all the features you need to build into plan. You need to highlight all the classes, methods and flush out what your API would look like]

1. **Financial Information Tracker:**

|  |
| --- |
| Add\_Expense () method:  Inputs: Expense reports, employee, timestamp, reason, tags (0 by default)  Outputs: Boolean (0 if success, -1 if failure)  Processed: Adds a block to the chain. Adds expense to employee’s chain too. |
| Salaries\_tracker() method:  Inputs: employee salary  Outputs: (0 if success, -1 if failure)  Processed: Adds an expense specific to an employee (tag 1) |
| Add\_Misc\_Expenses() method:  Inputs: miscellaneous expense reports, employee, timestamp  Outputs: (0 if success, -1 if failure)  Processed: Generates a new block in an expense blockchain and adds it to the end of the expense chain with add\_expense(). (tag 0) |
| Add\_Revenue () method:  Inputs: report of revenue, source of revenue, timestamp,  Output: (0 if success, -1 if failure)  Processed: Generates a new block in an expense blockchain and adds it to the end of the revenue chain with add\_expense(). |
| Add\_loss() method:  Inputs: loss prevention losses, timestamp, employee id  Output: (0 if success, -1 if failure)  Processed: Adds loss prevention input to loss blockchain by generateing a new block for a blockchain and adds it to the end of the expense chain with add\_expense(). Uses tag 2. |
| **Class Expense {**  Employee: str,  Cost: float,  Time: time.time()  **}** |
| **Class Income {**  Revenue source: str,  Amount: float,  Time: time.time()  **}** |
| Revenue\_calc() method:  Inputs: N/A  Outputs: Total net revenue (total income – total expenses)  Processed: Sum up revenue streams minue sum of all expenses. |

1. **Standardizing HR information**

|  |
| --- |
| Add\_employee() method:  Inputs: Name, salary, corporate\_expenses = 0, overtime = 0  Outputs: Employee class object  Processed: Create a block for the employee class object and add it to chain. |
| Class Employee:  {  Name: str,  corporate-expenses: int.  Salary: int,  Overtime: int,  Department: str,  Supervisor: str,  Employee ID: int } |
| Remove\_employee() method:  Inputs: Name, fired with cause (bool)  Outputs: Severance pay  Processed: Create block to chain denoting employee was fired (id = 0). Calculate severance pay by multiplying a constant factor times time worked if fired with cause is false. |
| Get\_employee\_count() method:  Inputs: N/A  Outputs: Number of employees on workforce  Processed: Look through blockchain adding one for all valid employees and subtracting one for all invalid employees. Output the result. |
| add\_complaint() method:  Input: complaint description, employee 1 id, employee 2 id  Output: (0 if success, -1 if failure)  Processed: Add complaint blocks to a blockchain |
| Get\_complaint() method:  Input: employee id 2 id  Output: list of complaints  Processed: Find all complaints in blockchain targeted at one employee |
| Class Complaint:  {  Employee 1 id: int,  Employee 2 id,: int,  Complaint: str,  Time: time.time()  } |
| Retract\_complaint() method:  Inputs: complaint description, employee 1 id, employee 2 id  Output: (0 if success, -1 if failure)  Processed: Add duplicate complaint block to a blockchain, active = 0 |

1. **Standardizing Bank’s Clients Information**

|  |
| --- |
| **Class client {**  Name: str,  Bank\_number: int,  Number\_of\_accounts: int,  Type\_of\_accounts: str[number\_of\_accounts]  Net\_balance: float  **}** |
| Add\_client() method:  Inputs: name, bank number, number of accounts, type of accounts, starting balance  Outputs: Client class object  Processed: Add client to blockchain |
| Remove\_client() method:  Inputs: name  Outputs: balance to transfer  Processed: Add transaction to blockchain to remove the client |
| Calculate\_revenue() method:  Inputs: client name  Outputs: yearly revenue generated by client  Processed: Use bank average rate of return on loans times the client’s balance to find the revenue generated by them. |
| Get\_client() method:  Inputs: client name  Outputs: Client block  Processed: Find client block in blockchain |

1. **Supply Management Information**

|  |
| --- |
| **Class Supply {**  Name: str,  Quantity: str  **}** |
| Add\_supply() method:  Inputs: name, quantity, date added, employee id  Outputs: (0 if success, -1 if failure)  Processed: Adds a supply to the blockchain |
| Use\_supply() method:  Inputs: name, quantity, date\_added, employee id  Outputs: (0 if success, -1 if failure)  Processed: Adds a negative supply to the blockchain |
| Get\_inventory() method:  Inputs: supply  Outputs: quantity  Processed: Finds cumulative sum of all adds and uses of supplies across the blockchain. |

1. **Project and Investment Management**

|  |
| --- |
| **Class Project {**  project supervisor: int (employee id),  workforce allocation: int (number of employees),  budget: float,  project name: str  **}** |
| Add\_project() method:  Inputs: project name, supervisor id, employees required, and budget/profit (budget > 0, profit <0)  Outputs: (0 if success, -1 if failure)  Processed: Add the project to the projects blockchain |
| Remove\_project() method:  Inputs: project name  Outputs: (0 if success, -1 if failure)  Processed: Add the project to the projects blockchain with 0 for supervisor (i.e. complete) |
| Get\_active\_projects() method:  Inputs: N/A  Outputs: Returns a list of every project that is currently active with an active supervisor overseeing its progression  Processed: Searches through the blockchain for all projects that have not yet been cancelled. |
| Get\_project() method:  Inputs: project name  Outputs: Returns project information if active, -1 if cancelled  Processed: Finds if project is active in the entire projects blockchain. |

1. **Employee Scheduling Management**

|  |
| --- |
| **Class Schedule {**  Employee id: int,  Day of year (0-365): int,  Cancelled = 0  **}** |
| Add\_Schedule() method:  Inputs: employee id, day  Outputs: (0 if success, -1 if failure)  Processed: adds employee to schedule blockchain |
| Remove\_Schedule() method:  Inputs: employee id, day  Outputs: (0 if success, -1 if failure)  Processed: adds employee to schedule blockchain with cancelled = 1. |
| Get\_Schedule() method:  Inputs: day  Outputs: all employees working on that day  Processed: search through schedule for a specific day and return all active employees who are scheduled on that specific day |
| Get\_Employee() method:  Inputs: employee  Outputs: List of every time that employee is scheduled to work across the entire schedule blockchain  Processed: Search through the blockchain and record every time an employee is scheduled and redact it if their schedule is cancelled at any point for that day. |
| Swap\_shifts() method:  Inputs: employee 1 id, employee 2 id, day 1, day 2  Outputs: (0 if success, -1 if failure)  Processed: Checks if employee 1 is still scheduled to work on day 1 and employee 2 is not and then checks if employee 1 is not scheduled to work on day 2 and employee 2 is and swaps them on the blockchain by running remove\_schedule() on both of them and then add\_schedule() on the opposite of them. |

# Technology Overview

[Here you talk about the programming language, libraries, and any technology that you are using to help create this program]

I plan to implement this project using Python as the base with a virtual blockchain done via file I/O rather than implementing it on a hosted server such as flask since I am more familiar with this system and, using this implementation, it will be easier to visualize the chain using the abstraction of blocks as files and the entire chain being the total number of files present. Additionally, storing with files initially will also mean that the chain is preserved when shutting the program off and on again, rather than having to reconstruct the chain server each time it is booted up. I plan to make this project heavily reliant on blockchains to accomplish many of the tasks of managing data and keeping logs of everything that happens across the entire enterprise resource planning application. As such, the total plan uses 9 blockchains, 8 of which are locally stored to different segments of the enterprise resource planner, and the last one which is a master blockchain to hold and store every element across the entire chain. A diagram of the proposed blockchain can be found on the next page in the form of a rough sketch of exactly how I plan on constructing the program and all of the internal components within. In order to access and use the program, for now, I plan to just let the user enter a series of commands built into the program at the end of the code. Alternatively, users will be able to enter information using google forms which will be retrieved using python’s bs4 module with its Beautiful soup function from an online database.

A picture containing text, map

Description automatically generated

Figure 1: Diagram of Planned Enterprise Resource Planner with blockchains denoted in bottom left.

# Software end-points

[Here you cover what the backend will be doing, what the front-end will be doing, what user interface the client (the Bank employee) will be using – and the features of these various end-points.

The backend of the software will manage all of the data structures and methods mentioned in the first section of this report. This will include all of the functions and management of data behind the scenes for the program. As for the front end, this area will be focused on making the service accessible from the user. For the main case, I will be using a google form as the input for the data. This allows users to simply click options on the form that they desire to accomplish, enter the parameters, and then hit submit, allowing them to easily enter more commands later without having to actually do any of the behind the scenes programming. For users with more experience, they will also be able to access the main.py function to run the script from behind the scenes as a terminal-style output.

# Documentation

[A manual for how the client will use the technology that you are making]

To use this software, use the google form at <https://forms.gle/2rq5oFxLcyPPZo6EA>. This form will allow you to enter information necessary to perform commands. Once entered, wait around 2-10 minutes for Google to process and update the database, and then run the main.py script. Unfortunately, I was not able to get this method working in every case as Google tends to add random HTML tags into some of the cells which would not be present on a normal, locally hosted server or database, so if the program fails to execute the command you entered, that is why. In order to execute it manually, the following commands are available to be typed at the bottom of main.py to execute:

**FINANCIAL INFORMATION TRACKER:**

expense\_chain.add\_expense(amount, employee\_id, reason)

expense\_chain.add\_misc\_expense(amount, employee\_id, reason)

expense\_chain.add\_salary(amount, employee\_id, reason)

expense\_chain.add\_loss(amount, employee\_id, reason)

revenue\_chain.add\_revenue(amount, employee\_id, reason)

revenue\_chain.remove\_revenue(amount, employee\_id, reason)

print(expense\_chain.expense\_calc())

print(revenue\_chain.calculate\_revenue())

print(revenue\_chain)

print(expense\_chain)

**HR INFORMATION TRACKER:**

employee\_chain.add\_employee(name, salary, department, supervisor\_id, employee\_id)

employee\_chain.remove\_employee(name, salary, department, supervisor\_id, employee\_id)

print(employee\_chain.get\_employee\_count())

print(employee\_chain)

complaint\_chain.add\_complaint(description, employee1\_id, employee2\_id)

complaint\_chain.retract\_complaint(description, employee1\_id, employee2\_id)

complaint\_chain.get\_complaint(employee1\_id)

print(complaint\_chain)

**STANDARDIZING CLIENT INFORMATION**

client\_chain.add\_client(name, bank\_num, account\_num, type, bal)

client\_chain.remove\_client(name, bank\_num, account\_num, type, bal)

client\_chain.calculate\_revenue(name)

client\_chain.get\_client(name)

print(client\_chain)

**SUPPLY MANAGEMENT INFORMATION**

supply\_chain.add\_supply(name, quantity)

supply\_chain.use\_supply(name, quantity)

supply\_chain.get\_inventory(name)

print(supply\_chain)

**PROJECT/INVESTMENT MANAGEMENT INFORMATION**

project\_chain.add\_project(supervisor\_id, workers, budget, name)

project\_chain.remove\_project(supervisor\_id, workers, name)

print(project\_chain.get\_active\_projects())

print(project\_chain.get\_project(name)

print(project\_chain)

**EMPLOYEE SCHEDULING INFORMATION**

schedule\_chain.add\_schedule(employee\_id, day)

schedule\_chain.remove\_schedule(employee\_id, day)

print(schedule\_chain.get\_schedule(day)

print(schedule\_chain.get\_employee(employee\_id)

schedule\_chain.swap\_shifts(emp\_id1, emp\_id2, day1, day2)

print(schedule\_chain)