# Capstone Purchase Order System (EDNS 1)

Team CPOS

**Team Members:**
Austin Reynolds
Marcus Loi
Toby Sellers
Qianshan Liu

**Client:**
Kimberly Walker

**Date:**
13th of June 2021

**Table of Contents**

## Introduction

### The Engineering Design and Society Capstone Program

Our client is an administrator in charge of managing the Engineering Design and Society (EDNS) Capstone Program which involves the collaboration between students and industry, government agencies, as well as community organizations through various projects. Capstone teams are assigned with different budgets which allows them to purchase materials for their respective projects. These teams, their budgets and purchase orders fall under the supervision of our client.

### Vision of our Product

The problem we were tasked to solve was a system to create and manage purchases for the EDNS Capstone Program. It also kept track of the budgets of every team and allowed for easy editing of those budgets by administration. The working solution that was used by our client involved a combination of emails and spreadsheets which were edited manually. This system had inefficiencies on a large scale due to the number of spreadsheets that were being used. The goal of our project was to provide a system that scales more effectively when new team and Project Advisor (PA) pairs were added.

Our final product was an interface and database that allowed the administrators to track and modify the budgets and purchase records of engineering capstone project teams. It also allowed team members to submit purchasing orders to then be approved by their PA. This software was aimed at helping our client as well as EDNS manage the logistics behind the purchases that will be made throughout the Capstone Program.

**Requirements**

The functional requirements of our project were divided into a purchase order system, a database, and an interface to allow us to better define the core specifications of our project. The non-functional requirements were determined based on hardware available to our client.

**Functional Requirements**

Purchase Order System Requirements

- Purchase request form for teams
- Allow the PA to approve team purchase orders
- Verify that the PA signed off on the purchase order
- Notify the team if they have insufficient budget for a purchase and block the order from going through
- Cancel purchases with negative/insufficient budget
- Allow Delivery Personnel to update the status of a Purchasing Order when it's been delivered/received

Database Requirements

- Query the database to find the names of team members and PAs as well as the name of their project
- Create new budget indices and team budgets as needed
- Keep track of team budgets as a whole as well as budget subtotals in indices
- Method to subtract from indices when a purchase order is filled
- Ability to print a budget summary/report
- Ensure the administrators s can access and edit the database
- Allow administrators to add new teams along with their PAs
- Method to generate unique team IDs for easier queries
- Team members should only be able to view their own individual budget

Interface Requirements

- Easy to understand interface for people unfamiliar with Computer Science
- Can be text-based or use command line as long as operation instructions are built in
- GUI preferred but not necessary

**Non-functional Requirements**

- Use minimal system memory and disk space
- Include failsafes in case of user error (e.g. incorrect info is entered) and a way to fix mistakes
- Make sure the program/database can be accessed by multiple users on the network simultaneously
- Must be able to run on Windows Operating System (OS)

## System Architecture

The overall system architecture of our project was visualized through a flowchart of the application, a GUI login flowchart, and an ERD design diagram. These visual entities helped provide an outlook on the flow of our program as a whole, as well as several important specifications that needed to be present in our final product.

## Application Flowchart

The application flow chart described the overall user experience of the application as an administrator. The administrator was the role intended for the client, and thus has the maximum number of operations and permissions, including viewing any team or purchase order, adding or deleting teams, filing the orders post purchase, and modifying team budgets.

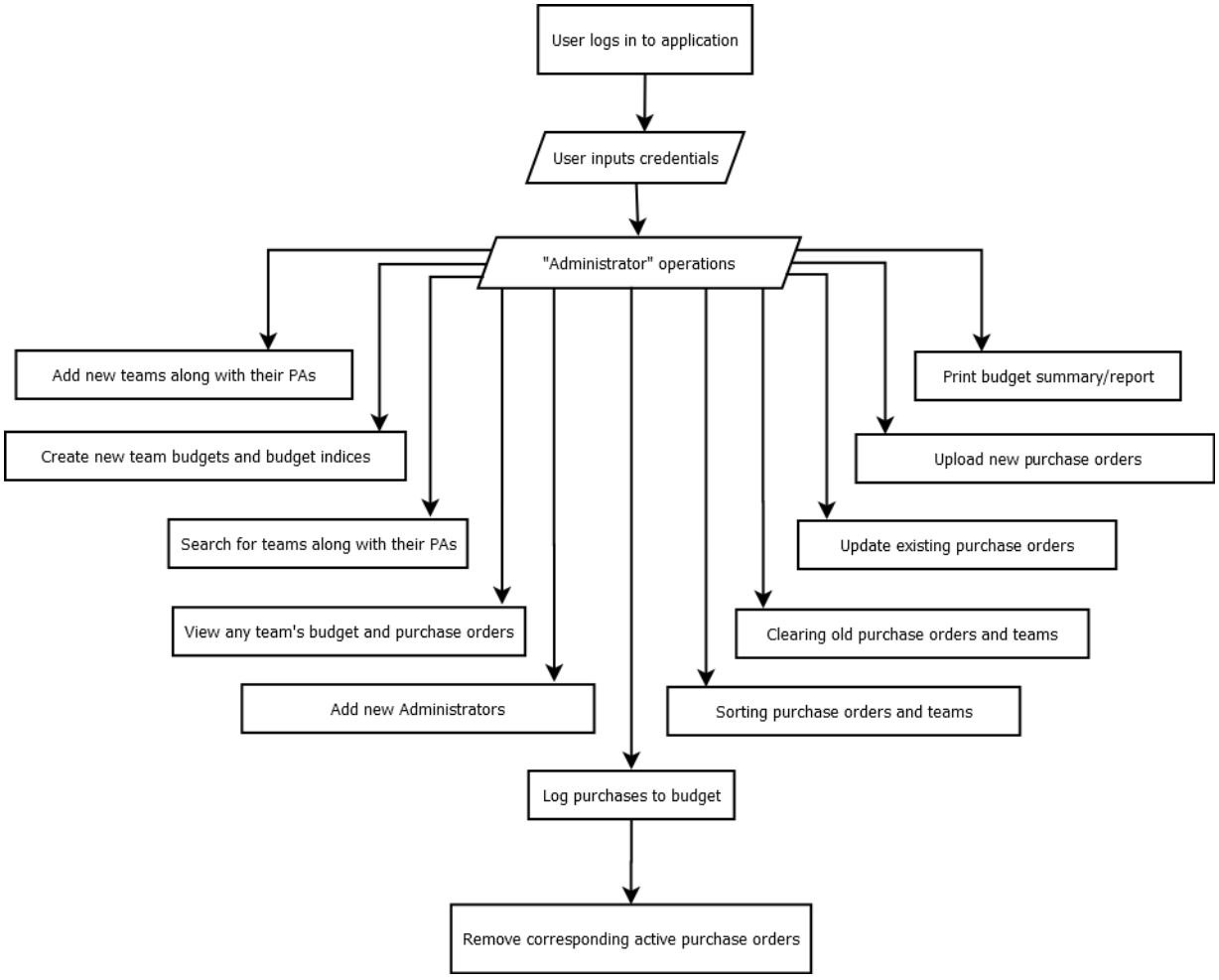


*Figure 1: Application Flowchart*

**GUI Login Flowchart**

One of the requirements of our project was to come up with a GUI that was able to handle the login process of the user before being able to access the database. The flowchart below showed how a user used their credentials to log into the system. Initially, the user would open the program and be prompted for their credentials. The program then checked the database for a user entry with matching credentials, and if there was a match it then checked that the user type selected also matched the one inside the database. If both checks were successful, the user was able to access the database and the functions allotted to their specific user type. Otherwise, they would be notified of a login failure and could then either attempt to login again, starting at the "Poll User Type" block, or exit the application entirely.

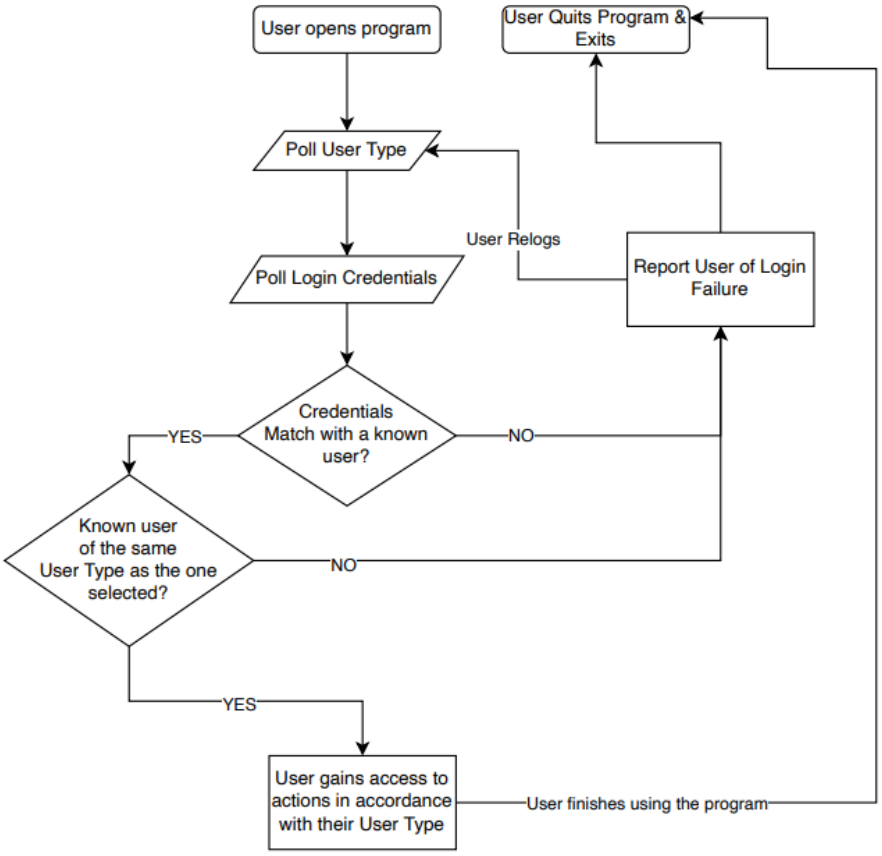


*Figure 2: GUI Login Flowchart*

**ERD Design**

The ERD design diagram shows the idea of how the tables within the database were handled. The model was centered around the "User" table, which contained the passwords, usernames and user types of all who use the database. Branching out from there, notice the two specific user types which were elaborated on as their own entities, "Team Member" and "Project Advisor". This was done because they directly relate to other entities like "Team" (each project team has members and a PA) and "Order Request" (order requests are made by team members and approved/voided by their PA). The "Team" entity contains multiple different "Budget Indices", a dollar amount identified by an index number. The "Order Request" table, specifically, contains the "Purchase Order", which contains "Item" entities as well as entities for "Company Info" and "Responsible Team Member Info".

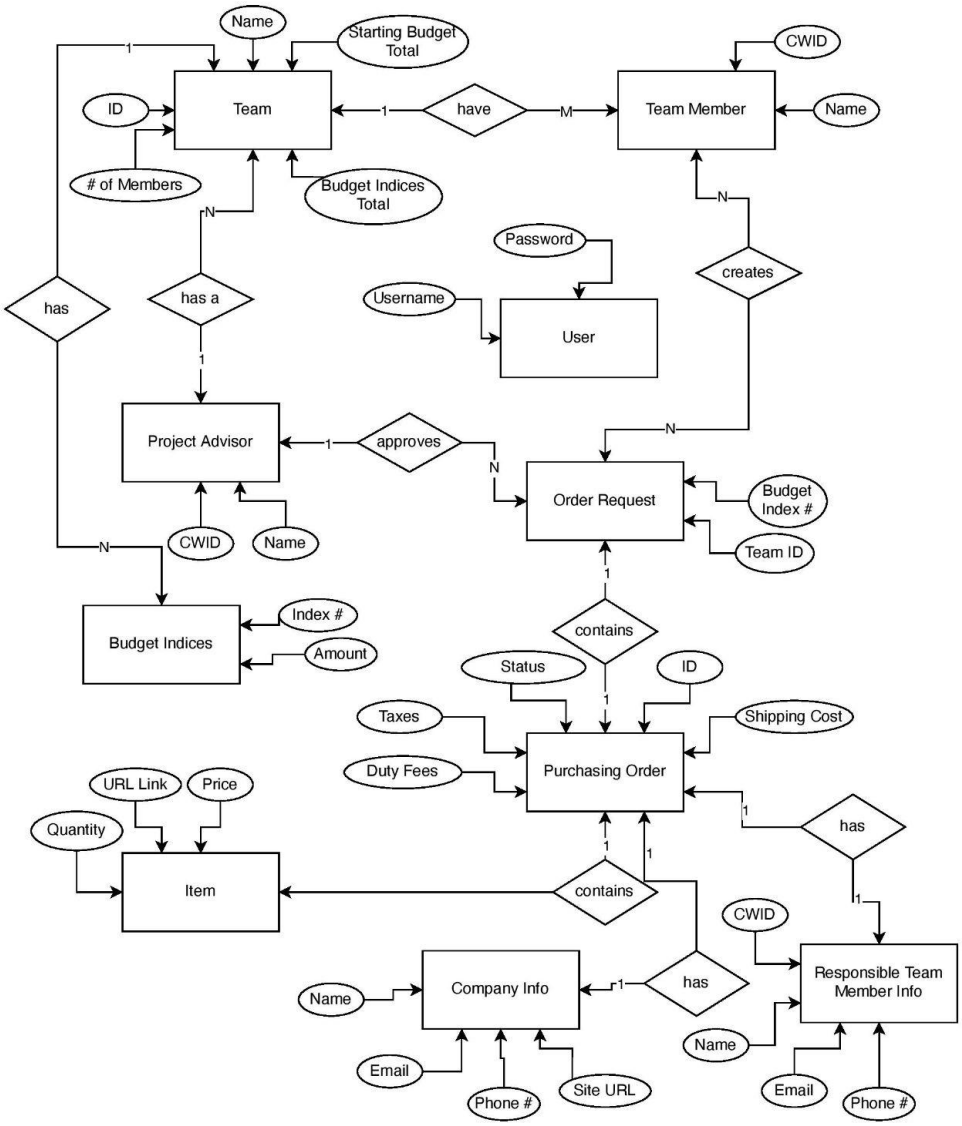


*Figure 3: ERD Design Diagram*

## Technical Design

### GUI Design

This UML diagram for our GUI shows the architecture of our GUI classes and how they relate to our Main class, the class that ties the backend and front end together. Each of the UI classes strictly obtained information from Main, without needing to interact with any other classes. This ensured that, as intended, Main fully facilitated all interactions between the backend and the frontend. There were four classes in the GUI, each corresponding to a different part of the user interface. Overall, our GUI was based on moving the user from a main overview to more detailed information to ensure the display was never cluttered and key high level info was presented first.

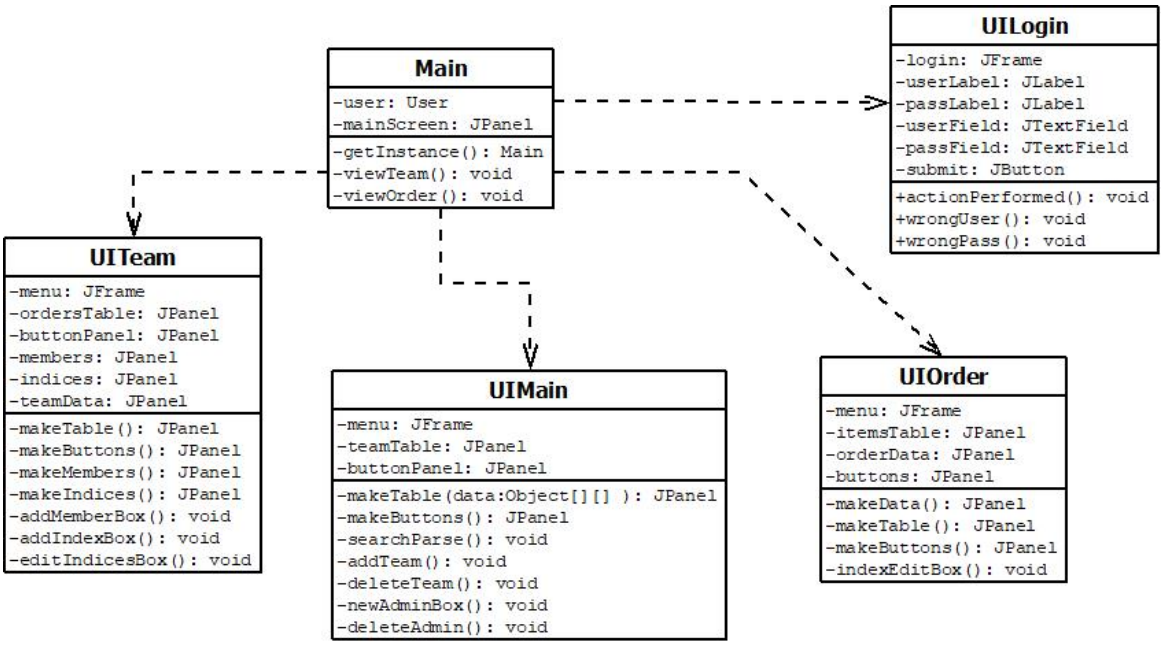


*Figure 4: UML of GUI*

First, the UILogin class (Figure 5) handled the interface where the user plugged in their credentials. The credentials were sent to Main, which checked their validity and either threw an error message or saved their credentials and opened up the main menu. This menu, contained in UIMain (Figure 6), presented an overview of all teams in the database and allowed the user to perform operations that broadly affected the system, like adding teams and users, deleting teams and users or searching for teams. The next class, UITeam (Figure 7), displayed an individual team's info when selected by the user. It allowed the user to edit more team specific info, such as team budget indices, team members, their project advisor, and the team's project name and ID. Additionally, it permitted the user to create a printable budget summary file and list all the team's purchase orders. Finally, UIOrder displayed all of the information contained in a purchase order, and allowed the user to update its status and modify the team's budget based on the purchases.

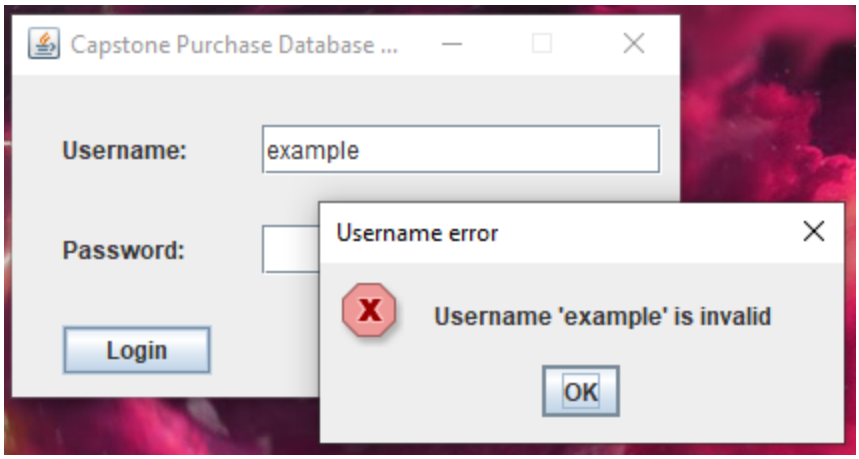


*Figure 5: Login window and invalid user warning*

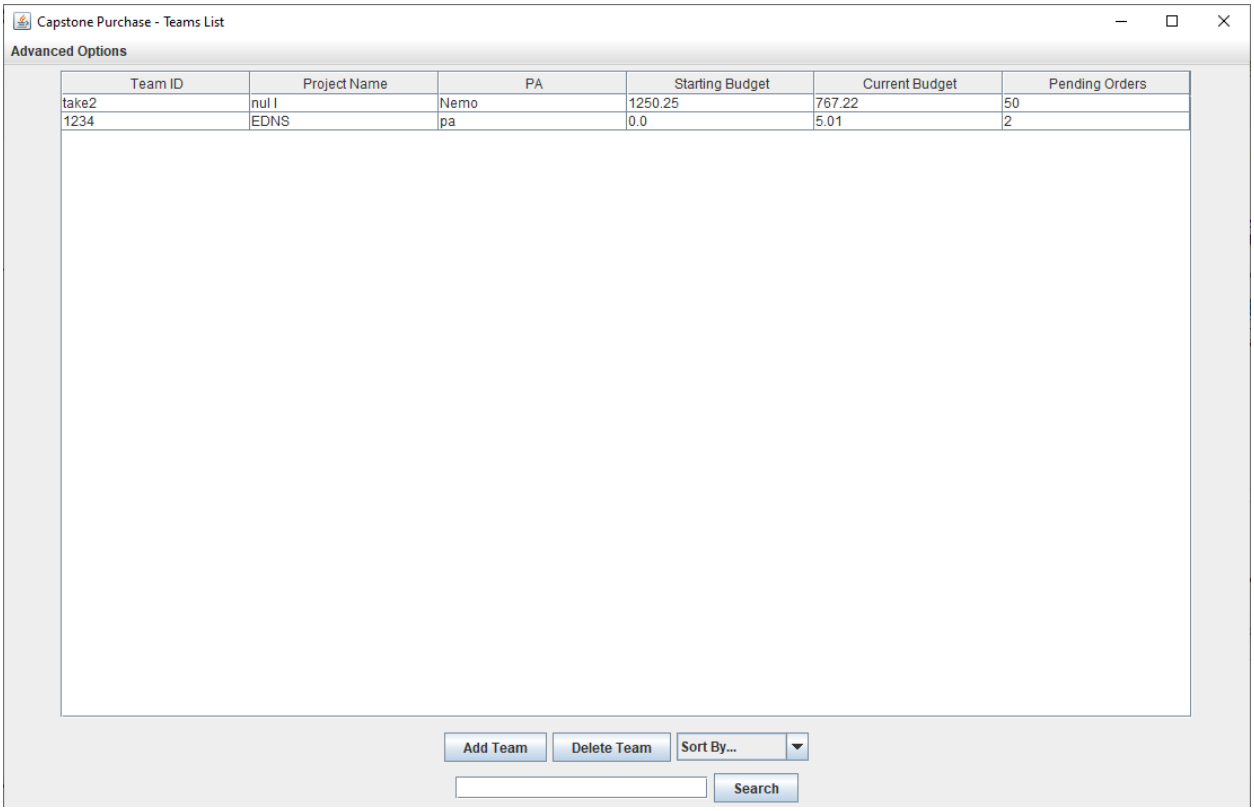


*Figure 6: Example Main Menu*

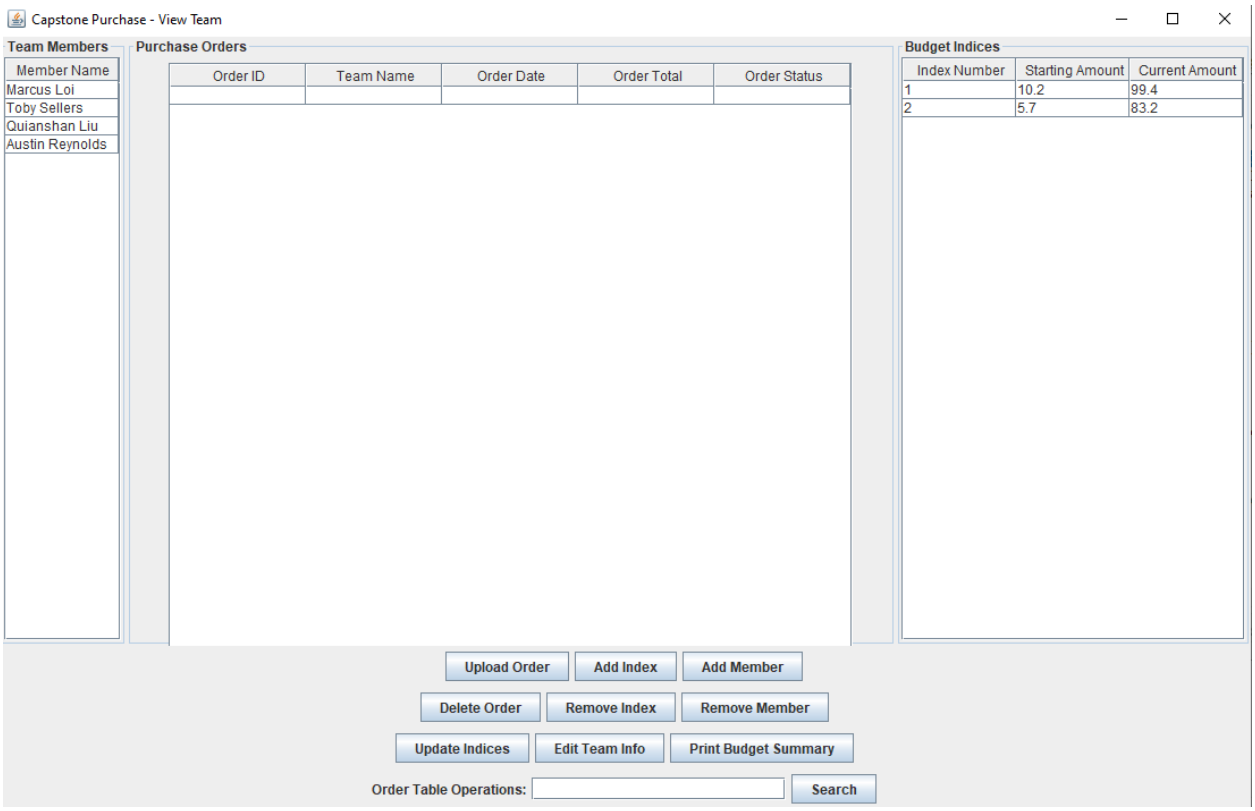


*Figure 7: Example Team View*

**Database Design and Queries**

Since one of the requirements of our project was to compile and process data related to teams and their purchase orders as well as budgets, we came up with a database design which can be queried using MySQL statements. Since we had to start our database from scratch, we began by creating tables with columns corresponding to the data we were working with. This was done through a series of MySQL statements in our database such as the ones shown below.

```
create table  TeamIndex(
    Tname varchar(255),
    ID int,
    Tindex varchar(255),
    startingBudget decimal(20,4),
    budget decimal(20,4)
);
```

```
create table  purchaseOrder(
    orderId varchar(255),
    memberID int,
    email varchar(255),
    memberPhone int,
    orderComplete bool,
    vendor varchar(255),
    international bool,
    method varchar(255),
    specialRequest varchar(255),
    total int,
    orderStatus varchar(255),
    deliveryDate date,
    approvalDate date,
    paName varchar(255),
    paComment varchar(255),
    teamId int
);
```

*Figure 8: Statements to create tables in our database*

Next, we needed to make sure that the data that will be processed is managed and distributed into tables correctly. We performed some tests by manually adding data into the tables and running multiple queries against the database to check if desired results were obtained. This would ensure that the queries made against our database in the future would continue to return accurate and expected results.

| Tname | ID | Tindex | startingBudget | budget |
|---|---|---|---|---|
| rex | 1 | nasa | 8000.0000 | 8000.0000 |
| tma | 2 | mines | 8000.0000 | 8000.0000 |
| cnm | 3 | cpos | 8000.0000 | 8000.0000 |
| soft | 4 | big | 8000.0000 | 8000.0000 |
| mega | 5 | ssss | 8000.0000 | 8000.0000 |
| mech | 6 | ssr | 8000.0000 | 8000.0000 |
| gas | 7 | slack | 8000.0000 | 8000.0000 |
| hooh | 8 | mechdepar | 8000.0000 | 8000.0000 |
| lkm | 9 | huanjia | 8000.0000 | 8000.0000 |
| student | 10 | ssss | 8000.0000 | 8000.0000 |
| loop | 11 | ssss | 8000.0000 | 8000.0000 |
| sas | 12 | ssss | 8000.0000 | 8000.0000 |
| rex | 1 | sal | 4000.0000 | 4000.0000 |

*Figure 9: Team budget table*

For instance, in our table designated for team budgets in our database, each team was assigned with a unique ID so that queries made against the team budgets table would return results containing all indexes with the same ID. We made sure that returned results were consistently accurate through a series of queries which served as a stress test for our database.

**Quality Assurance**

In order to ensure that the end-product of our project fulfilled the software quality requirements that were established, we deployed several quality control methods throughout our development process. Code reviews were held on a weekly basis to enable us to monitor our progress, ensuring that we were able to get sufficient work done in our given timeframe while adhering to our project design and requirements specifications. To maintain the consistency and readability of our code, general software engineering principles such as SOLID - the first five principles of object oriented design, and YAGNI - a principle of extreme programming, were followed and upheld during our code review sessions. For ease of maintenance, we included comments for all non-trivial methods to highlight their purpose and how they worked outside the method body. On top of that, we also implemented a variety of testing methods to ensure that the functionalities of our software met the required project specifications. These testing methods included unit tests which focused on the general operations of our code and integration tests which were responsible for the seamless unification between our program's interface and database.

Database Unit & Integration Testing:

- Tested team and budget creation, including edge cases (0 members, 0 indices, $0 budget) to ensure all info was properly recorded in the database and any attempts to create an invalid team threw an exception.
- Tested user authentication, including new database user authority. This test confirmed that the administrator role would not have any error within the authority process.
- Tested purchase order creation with valid and invalid purchase order types to make sure any successful order submission was valid. This would streamline the order creation and review process since our client would not have to worry about order validity.
- Tested database capacity to confirm that our database was able to contain the maximum amount data that needed to be stored, ensuring that the data that our client would add onto the database in the future would not overflow.

Interface Integration Testing:

- Tested navigation of user throughout system to ensure that they could reach all valid operations for their role
- Ensured the user was notified whenever an exception was thrown so they knew what went wrong with their input.
- Tested that email notifications were sent to proper users when purchase orders are submitted/approved/rejected/fulfilled.

**Results**

Test Results

Our tests ensured that all different parts of the project could interact properly. The SQL database talked to the backend, which talked to the Main class, which talked to the GUI. Overall, the tests made sure that data flowed properly between storage and output, and that user inputs properly affected the database. Additionally, usability tests ensured the client could properly operate the interface and was satisfied with its functionality, and that multiple users could operate the system simultaneously without causing issues.

Summary of Implemented Features:

- Server housed SQL database containing all team, budget, and purchase order info
- Database can be accessed by multiple users at once
- Full GUI frontend for database

Features We Did Not Have Time For:

- Team member and PA access
- Email notifications
- Login integration with Mines Multipass
- Complete synchronization for multiple users

Due to the lack of preexisting infrastructure and tight time constraints, we ended up with a lot of possible features going unimplemented. We originally wanted everyone related to capstone projects, including PAs and the actual capstone team members, to have access to the database. This would have fully automated the process, as all budget related functions by all parties involved could have been done through our system. However, we realized we didn't have time to implement such a large scale system and ensure it worked properly. The system would need to handle a very large number of users, and ensure all the features for different user types worked correctly.

Additionally, since all users would have required accounts for access, it would have been most practical to simply integrate our system with Mines multipass. Since our current system only has a handful of users, the amount of setup and additional security concerns related to implementing Mines multipass made it more practical for us to just create and use our own account database.

We also hoped to further automate the system with email notifications sent out to users when purchase orders were approved, purchased, or delivered, so they could receive updates without constantly checking our database. This also proved to be too complex for the scope of the project. In summary, our original idea made the system as automated and user friendly as possible, but would have taken too much extra time to finish in the few weeks we were allotted. However, what we have made is solid enough that it is possible to add those features on top of what we currently have in the future, if the client desires it.

Lessons Learned:

- SQL database connection with Java, given that our team had little experience with the java.sql library. We encountered several issues while trying to establish a stable connection to our SQL database. Although it took our team a while to install and adjust the server we used for testing, the experiences we gained in database porting would definitely come in handy for future database developments.
- Java is a powerful programming language, yet is relatively easy to use given our experience with it. Java has an extensive list of modules that are capable of supporting a lot of functionalities. For instance, existing modules in Java allowed us to turn our code into a web-based application as well as integrate SQL databases and queries with our interface.
- The scope of the project was large, given that it doesn't have any pre-existing infrastructure to build upon. As such, the functionality, both the Java code and the database server infrastructure, had to be implemented all from the ground up.
- Having to work with multiple outside parties slowed down development. Because the system had to be accessible by multiple users, we had to work with Information & Technology Solution (ITS) - the Colorado School of Mines IT department, to put the database on their network. This complicated the development process, as we had to communicate with them and work within their constraints as well as the client's. It also took them a long time to provide us with the server space needed to test the project under working conditions.

**Future Work**

Based on the final design of our project, we were able to identify some modifications that could be made to a few components of our software in the future. These modifications could be beneficial towards the sustainability and maintenance of our design in the long run.

Based on our current implementation of the login process and the storage of user credentials which were both tied into the database set up on the administrator's local device, this component could be replaced with Mines Multipass. Since Mines Multipass provided login services for all Mines' staff and students with an extensive and secure record of all its users' information, its adoption into our program would greatly improve the efficiency and user experience throughout the login process. The synchronization between the existing user database provided by Mines Multipass would remove our software's dependency on user information stored in a local database. The validity of each user's credentials would be managed by Mines Multipass which would significantly improve the overall security of our system. The replacement of our login component with Mines Multipass could be achieved relatively easily with some assistance from ITS and Shibboleth - the providers responsible for the Mines Multipass system.

Besides that, additional features such as the use of cloud-based storage for data related to team purchase orders and budgets as well as a form of encryption for this data could be added to our developed product. These features could prolong the lifespan of our system, given the flexibility offered by cloud storage and the added security provided by data encryption methods. By integrating our software with existing cloud storage services, the amount of data that could be stored and processed is exponentially increased while providing a historical record of past team purchases and budgets. This feature could be added considerably easily through the reservation of a set amount of storage on cloud and the migration of existing data to said storage. On the other hand, the addition of data encryption would improve the overall security of our product by deterring data breaches from unauthorized users. This could be done by encoding raw data with a randomly generated encryption key prior to being stored.

## Appendix

Figure 1 - Application Flowchart

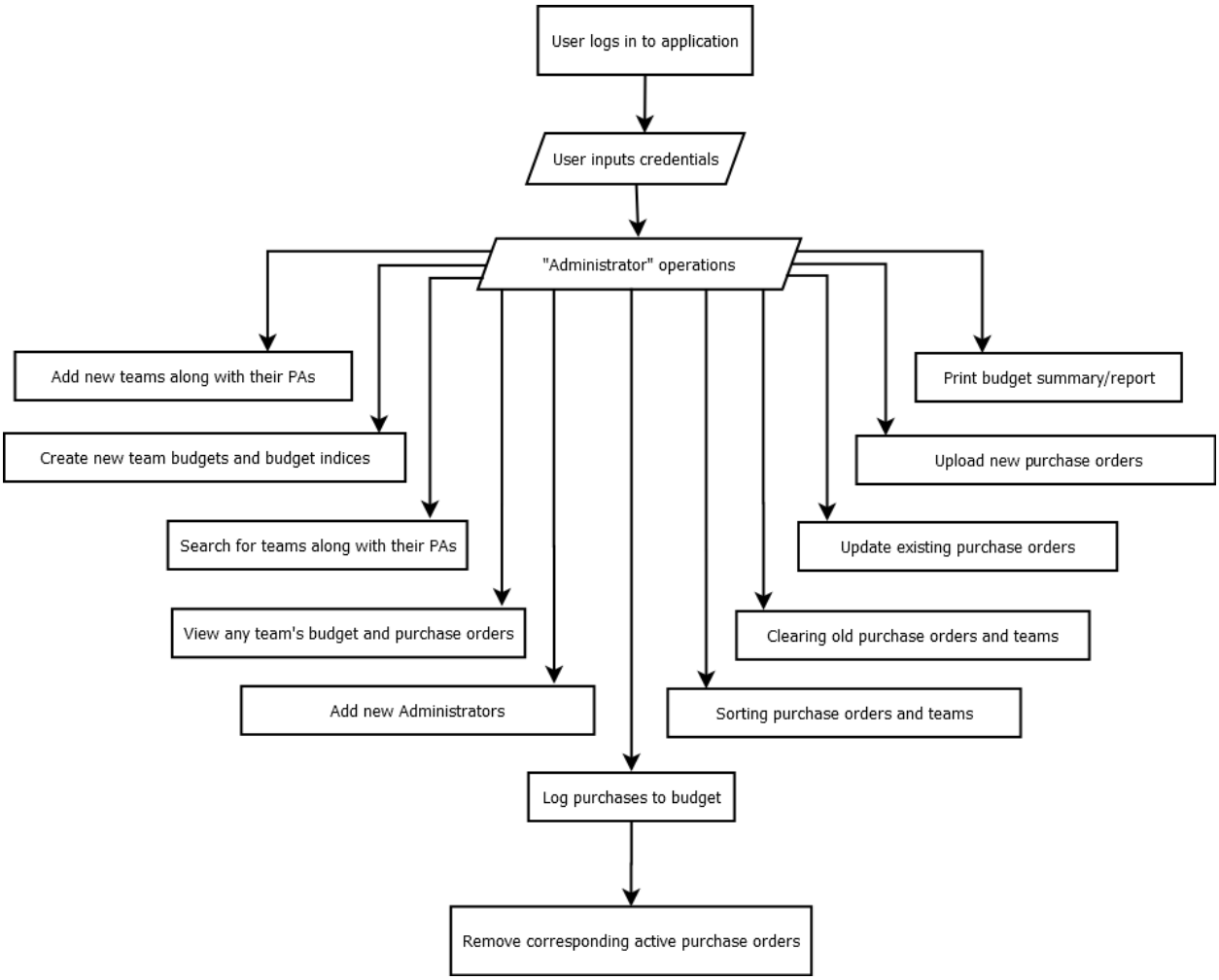

Figure 2 - GUI Login Flowchart

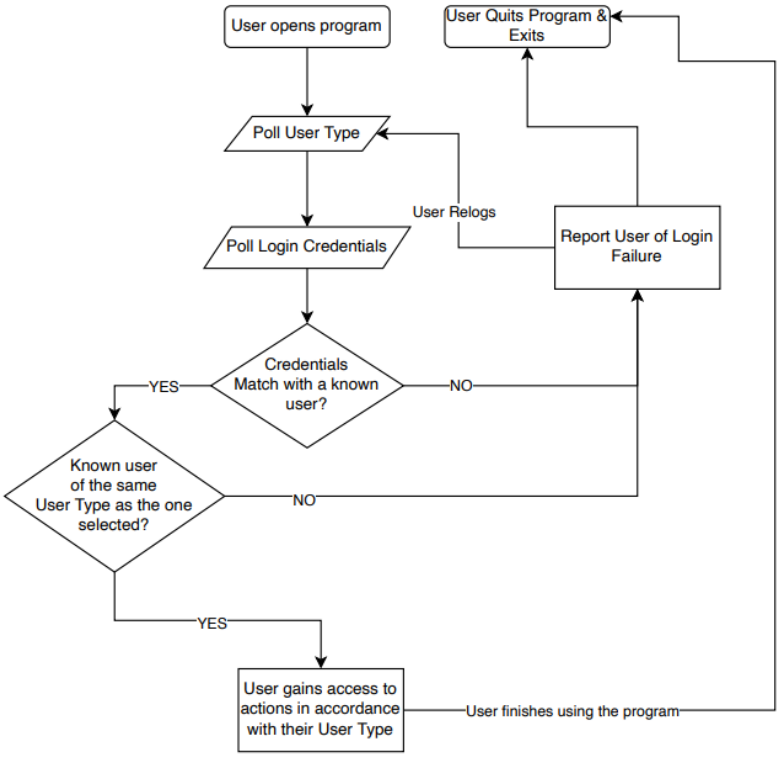

Figure 3 - ERD Design Diagram

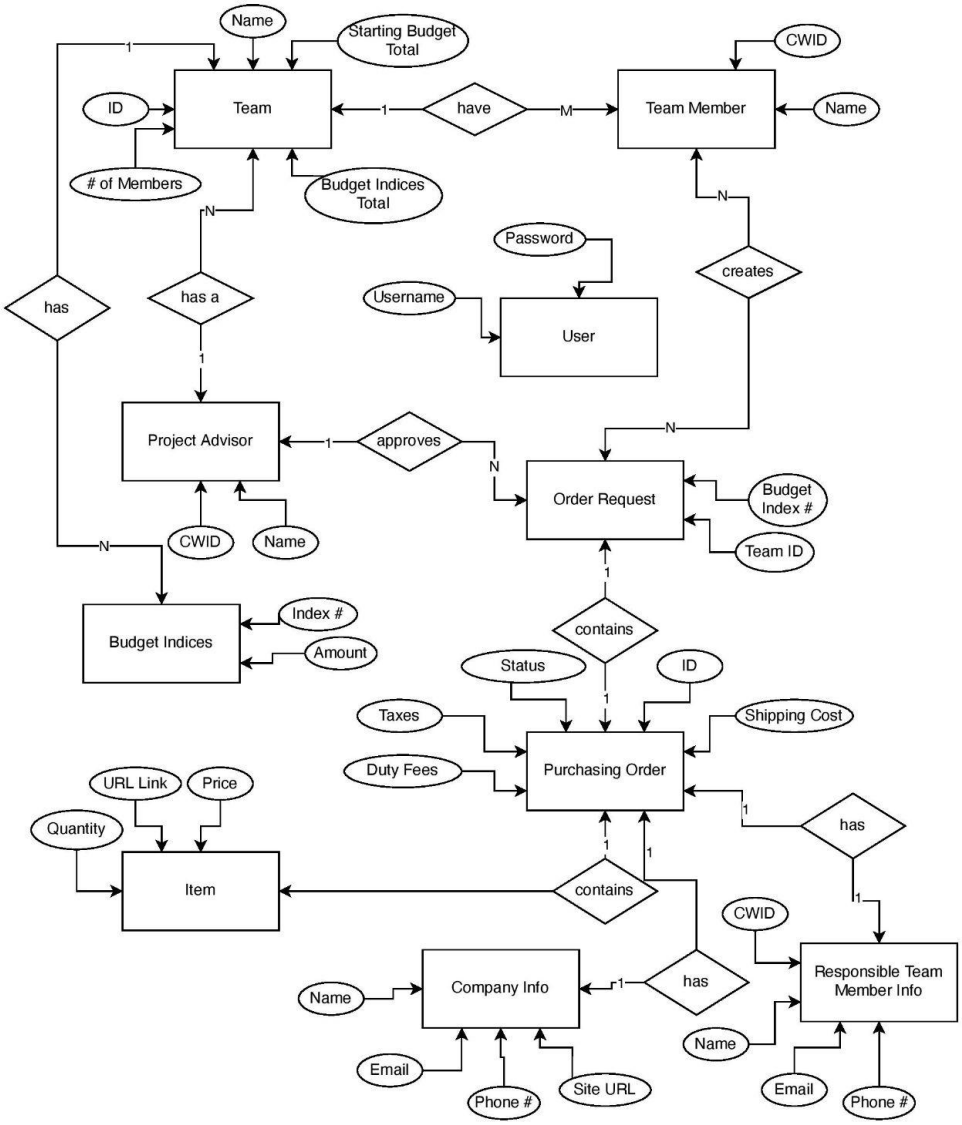

Figure 4 - UML of GUI

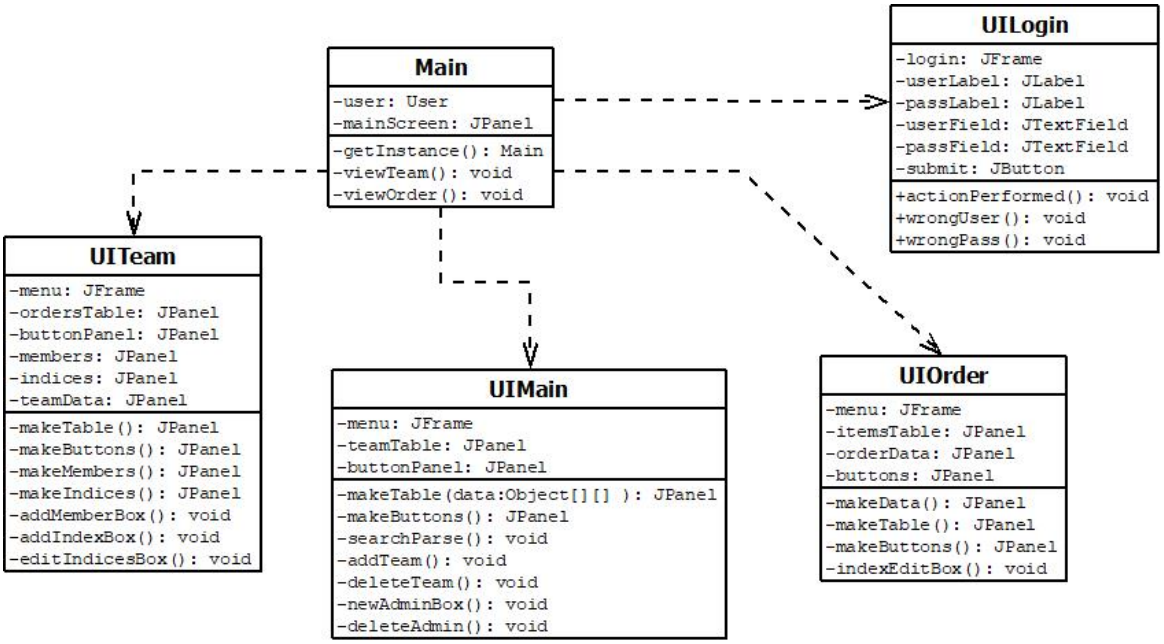


Figure 5 - Login Window and Invalid User Warning

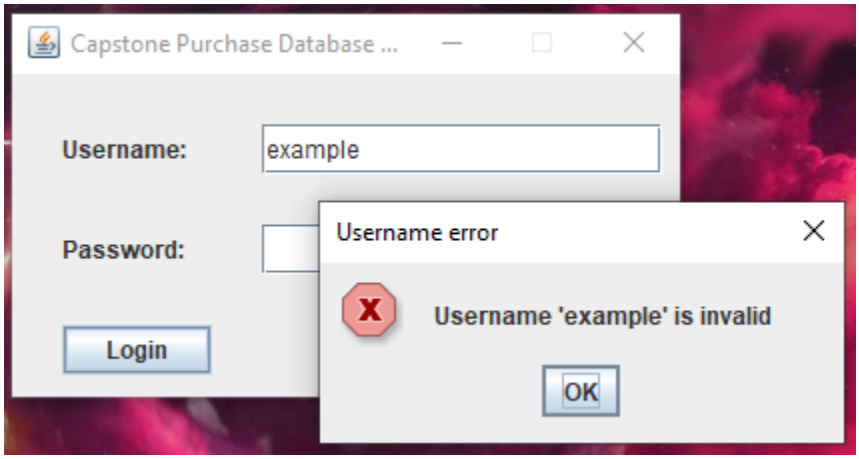

Figure 6 - Example Main Menu

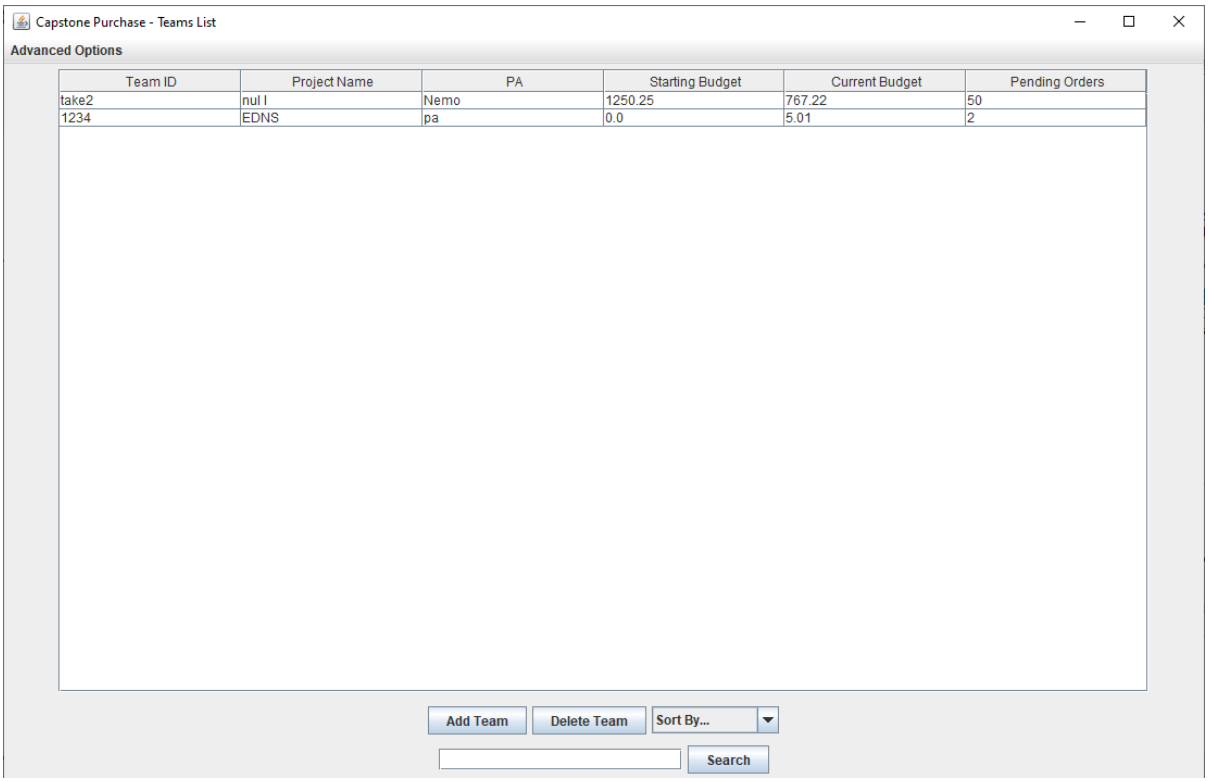


Figure 7 - Example Team View

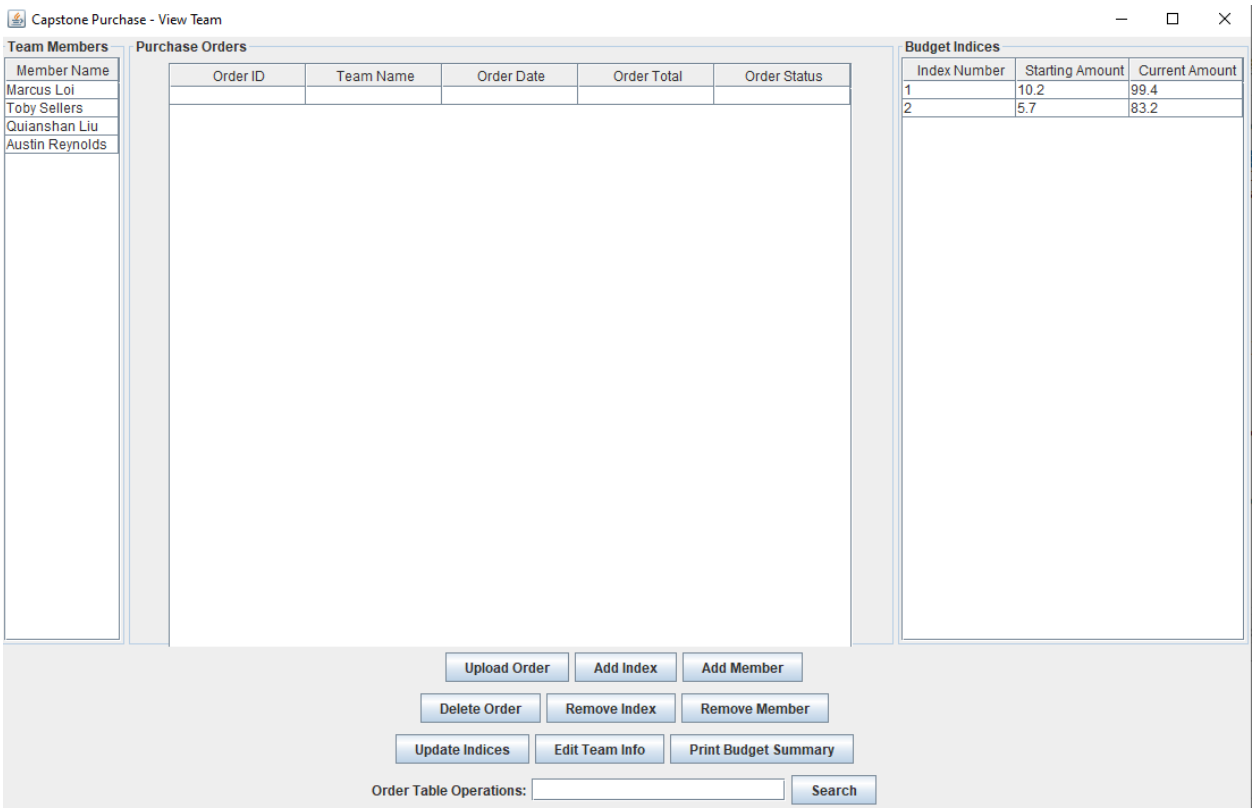

Figure 8 - Statements to create tables in our database

```
create table  TeamIndex(
    Tname varchar(255),
    ID int,
    Tindex varchar(255),
    startingBudget decimal(20,4),
    budget decimal(20,4)
);
```

```
create table  purchaseOrder(
    orderId varchar(255),
    memberID int,
    email varchar(255),
    memberPhone int,
    orderComplete bool,
    vendor varchar(255),
    international bool,
    method varchar(255),
    specialRequest varchar(255),
    total int,
    orderStatus varchar(255),
    deliveryDate date,
    approvalDate date,
    paName varchar(255),
    paComment varchar(255),
    teamId int
);
```

Figure 9 - Team budget table

| Tname | ID | Tindex | startingBudget | budget |
|---|---|---|---|---|
| rex | 1 | nasa | 8000.0000 | 8000.0000 |
| tma | 2 | mines | 8000.0000 | 8000.0000 |
| cnm | 3 | cpos | 8000.0000 | 8000.0000 |
| soft | 4 | big | 8000.0000 | 8000.0000 |
| mega | 5 | ssss | 8000.0000 | 8000.0000 |
| mech | 6 | ssr | 8000.0000 | 8000.0000 |
| gas | 7 | slack | 8000.0000 | 8000.0000 |
| hooh | 8 | mechdepar | 8000.0000 | 8000.0000 |
| lkm | 9 | huanjia | 8000.0000 | 8000.0000 |
| student | 10 | ssss | 8000.0000 | 8000.0000 |
| loop | 11 | ssss | 8000.0000 | 8000.0000 |
| sas | 12 | ssss | 8000.0000 | 8000.0000 |
| rex | 1 | sal | 4000.0000 | 4000.0000 |