Fish Tank Monitoring System

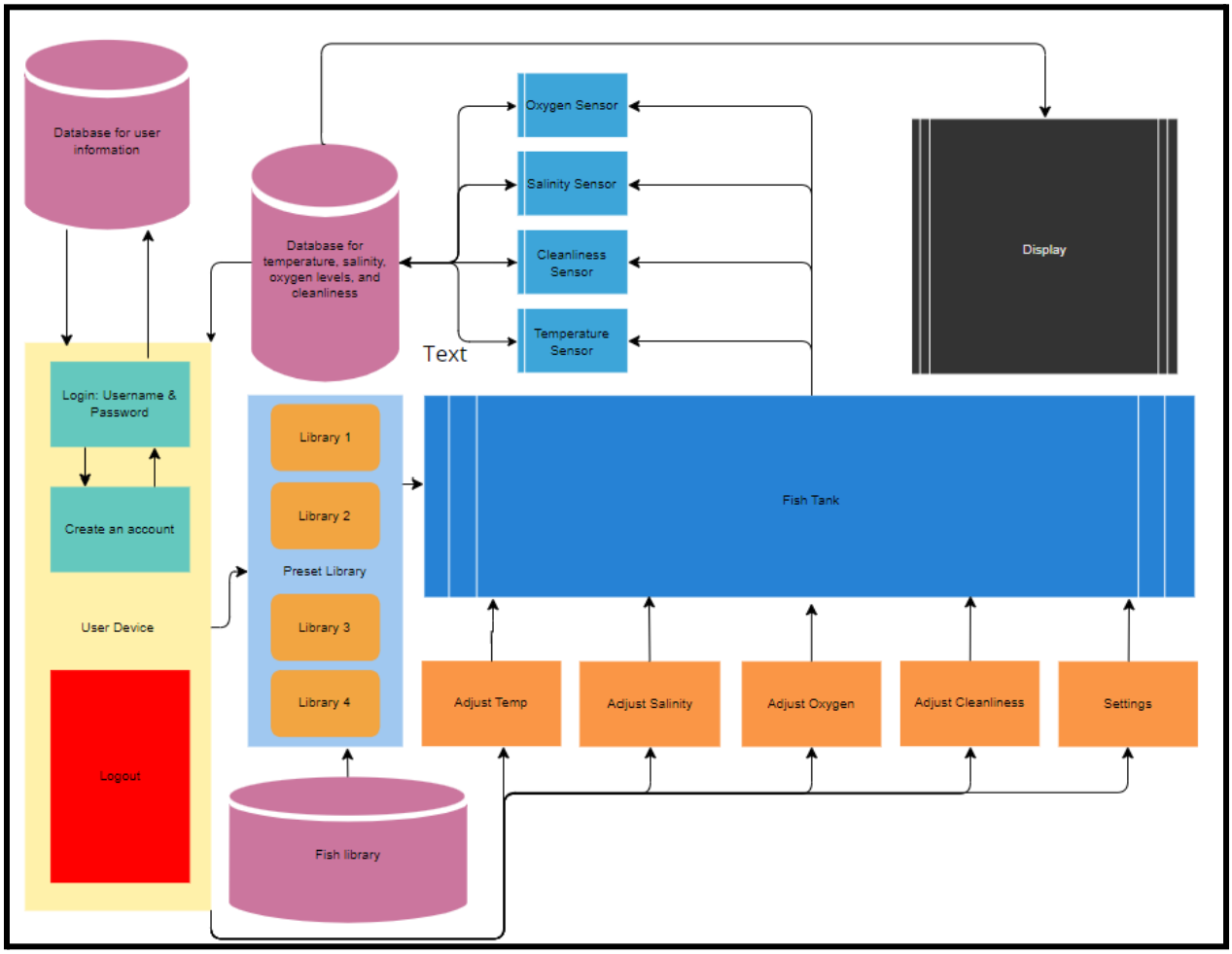
Software Design Specification Document

Development Team:	Role:	Contact:
Jose Garcia	Hardware Engineer	jgarciagomez7895@sdsu.edu
Alex Vo	Software Engineer	avo3681@sdsu.edu
Darren Lee	Security Engineer	dlee0083@sdsu.edu

System Overview

- The idea of the fishtank we are creating is to provide the user with an environment where they are able to access the necessary information to create a thriving aquarium. They will have access to information about a variety of fish as well as what fish can live in similar environments. The user will also be able to adjust certain living conditions for the fish such as temperature, oxygen levels, salinity, and cleanliness. The tank itself will have an automatic chemical balancing feature and will cater to the needs of the users and fish. The user will be able to interact with the fish tank system through the use of a mobile app that interfaces with the tank management system and hardware sensors. The app will also update users on the status of their tank via push notifications.

Software Architecture Overview



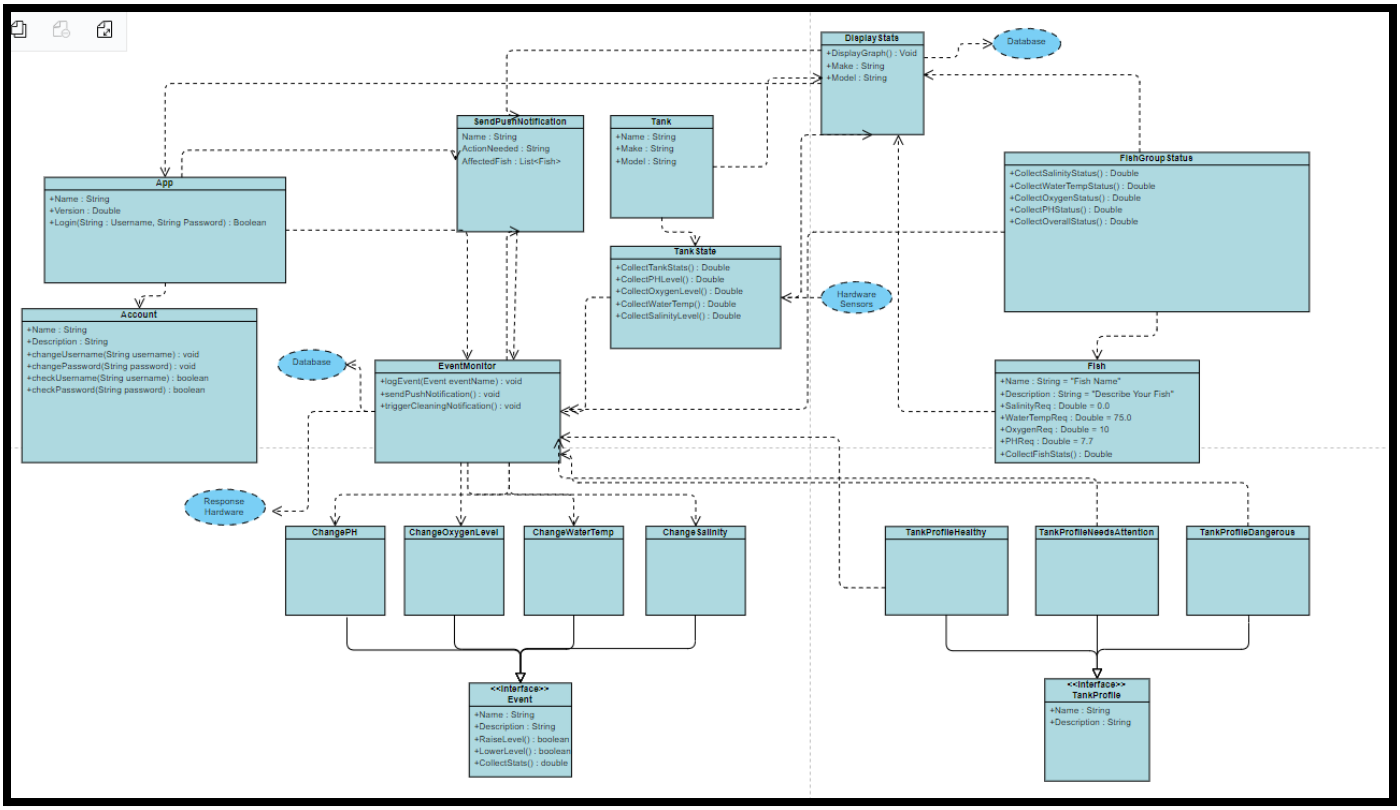
Pictured: Architectural Overview of the Fish Tank Monitoring System - Updated: Removed database for algorithms

Description of the software architecture diagram

Our fish tank is equipped with sensors that gather information about its salinity, temperature, oxygen levels, and cleanliness. This information is then stored in a database, allowing you to access historical data on these conditions throughout the day. Moreover, the current conditions are displayed on the exterior of the fish tank as well as in the accompanying app. The app has a login feature that stores user information, such as their username, email, phone number, and password, in a separate database. Once logged in, users can access fish libraries to learn more about the optimal living conditions for different fish species, including which ones can coexist. This account also stores the data and history of the user's fish tank. The fish library database contains information on the living conditions of various fish species. The app allows users to manually adjust the temperature, salinity, oxygen, and cleanliness levels of the

fish tank through their phone. These commands are stored in a separate database, and users can adjust the intensity and quantities of each element in the settings.

UML Class Diagram



Pictured: Unified Modeling Language (UML) graph of the Fish Tank Monitoring System

New Additions
+checkUsername(String username) : boolean added string as input **+checkPassword(String password) : boolean added string as input** **+Login(String : Username, String Password) is now documented as returning a boolean**

Description of classes

EventMonitor	The event monitor responds to incoming events and creates events depending on the data from the: - TankState

	- TankProfile - FishGroupStatus
<<interface>> Event	Event is an interface which is implemented as new variables are added to the tank.
ChangeSalinity	ChangeSalinity implements the Event interface and changes the salt levels within the tank.
ChangeWaterTemp	ChangeWaterTemp implements the Event interface and changes the water temperature levels within the tank.
ChangeOxygenLevel	ChangeOxygenLevels implements the Event interface and changes the oxygen level inside the tank.
ChangePH	ChangePHLevels implements the Event interface and changes the acidity level inside the tank.
CleanTank	Sends notification to the user to clean the tank, cannot be done chemically.
TankState	Tank State represents the current state of the tank based off of the data from the hardware sensors.
FishGroupStatus	FishGroupStatus takes the total status of all the fish based on the biological needs of the fish.
Fish	This class represents a fish and represents the biological needs and description of a particular fish.
<<interface>> Tank Profile	Tank profile is an interface that is implemented by different types of tank profiles.
Tank Profile - Healthy	This class represents the conditions for a healthy tank.
Tank Profile - Needs Attention	This class represents the conditions for an unhealthy tank that requires intervention or assistance.
Tank Profile - Dangerous	This tank represents the conditions for an unhealthy tank in which the inhabitants inside are at extreme risk.
Tank	The tank class represents the physical attributes along with the make and model of the Tank.
Account	The account class represents a user and includes a username and password. Outside classes can use the checkUsername() and the checkPassword() functions for authentication. These functions return a boolean to validate if the user authenticated successfully.

DisplayStats	The display stats is used by the application to display the statistics of the fish, as well as different chemical levels in the tank.
App	The app class is the main class to authenticate the user, in order to access all the other programs.
SendPushNotification	The SendPushNotification class is used to send notifications to the user's phone. It also sends the fish along with it to let them know what fish are affected by the tank's current state.

Attributes and Operations

EventMonitor	+logEvent(Event eventName) - Logs an event attempt in the database +sendPushNotification() - Sends a push notification to the owner's phone +triggerCleaningNotification() - Creates a notification to clean the fish tank
<<interface>> Event	+Name +Description +RaiseLevel - Returns true if level is raised, false otherwise +LowerLevel - Returns true if level is raised, false otherwise
ChangeSalinity	Implements Event Interface
ChangeWaterTemp	Implements Event Interface
ChangeOxygenLevel	Implements Event Interface
ChangePH	Implements Event Interface
TankState	+CollectTankStats
FishGroupStatus	+CollectSalinityStatus - Check to see whether the salinity of the tank matches overall fish needs. +CollectWaterTempStatus - Check to see whether the water temperature of the tank matches overall fish needs. +CollectOxygenStatus - Check to see whether the oxygen levels of the tank matches overall fish needs. +CollectPHStatus - Check to see whether the PH level of the tank

	matches overall fish needs. +CollectOverallStatus - Collects all statuses and creates an average for the statuses of all the fish.
Fish	+Name +Description +SalinityReq +WaterTempReq +OxygenReq +PHReq +CollectFishStats()
<<interface>> Tank Profile	+Name +Description
Tank Profile - Healthy	+Implements TankProfile Interface
Tank Profile - Needs Attention	+Implements TankProfile Interface
Tank Profile - Dangerous	+Implements TankProfile Interface
Tank	+Name +Description
Account	+Name +Description +Username() +Password() +ChangeUsername() +ChangePassword()
DisplayStats	+DisplayGraph() : Returns void as it displays the graph of the tank data. +Make +Model
App	+Name : String +Version : Double +Login(String : Username, String Password)
SendPushNotification	Name : String ActionNeeded : String AffectedFish : List<Fish>

Development Plan & Timeline

Name	Role
Jose Garcia	Responsible for the hardware integration with the software system. Will develop APIs to connect with the other backend software used by Alex. Will be responsible for developing unit and integration tests in order to ensure each hardware module operates as intended.
Alex Vo	Responsible for software integration of the system and ensuring proper control logic for the tank monitoring system. Alex will be responsible for creating the overall end to end testing suite along with ensuring the proper alerts and notifications are sent between each class.
Darren Lee	Responsible for setting up the security and authentication of the system. His responsibilities will include verifying the software integrity of the system such as ensuring that hardware does not fail. In addition, he will ensure that all users are authenticated successfully. Because this system can be accessed remotely through an app we would like to prevent our clients' fish tank system from being accessed.

2023 Development Timeline (9 Months)

March - Jose Garcia

- Investigate the market to determine the best commercial off-the-shelf hardware options
- Experiment with different hardware to identify the best reliability.
- Begin mapping out code that will follow the UML diagram

May - Jose Garcia

- Develop software notification system, and start development on android and iOS platforms.
- Create methods to add/remove dirt and gunk in the tank, salinity, and oxygen as well as increase/decrease temperature.

- Create and test databases to retrieve and store fish data

July - Alex Vo

- Meet with the client to determine to demonstrate a working model of the tank
- Make any changes that the client might want (materials, settings, user interface)
- Create user information database
- Find a discover any possible bugs in the prototype
- Begin testing

August - Darren Lee

- Refine the product, start creating unit tests and identify any bugs
- Finalize materials for fish tank
- Ensure clients needs were met

October - Alex Vo

- Discuss with the client about any final adjustments
- Deliver final product to the client for public use

December - Darren Lee

- Follow up with the client, and resolve any issues/bugs post-release after a couple of months of being out.
- Create a maintenance system
- Test before providing any updates

Testing Strategy Overview:

In this document our team will outline two overall strategies to test critical features of our overall Fish Tank Monitoring system. This testing plan is not designed to be an exhaustive list of ways our team verifies the correctness of the system, but is designed to provide an introduction to testing a portion of some of the key components. In our document our team outlines three main features we verify through our testing program. We plan to test:

- The fish tank data collection system
- The fish tank environment response system
- App authentication

Overview of tests:

Test Type	Description
Unit Test	FishGroupStatus - CollectOxygenStatus(): Double
Functional Test	Fish - +CollectFishStats() : Double
System Test	DisplayStats - +DisplayGraph() : Void

Test Type	Description
Unit Test	ChangeOxygenLevel - RaiseLevel() : Boolean

Functional Test	App - Login(String : Username, String Password) -> +checkUsername() : boolean, +checkPassword() : boolean
System Test	EventMonitor - +logEvent(Event eventName): void

Test Set One: Fish tank data collection system

Unit Test One

- FishGroupStatus - CollectOxygenStatus(): Double

Context: In this unit test we are testing to ensure that the method CollectOxygenStatus returns a double which represents the average PPM (parts per million) oxygen levels requirements for fish inside the tank.

Class Name: FishGroupStatus
Method Name: Double - CollectOxygenStatus()
Returns: Double - OxygenStatus

Testing For Correctness

Condition: Oxygen Status has not been collected

//Precondition
//OxygenStatus = 0.0;

//Postcondition
//OxygenStatus = Should be any value other than 0.0

Condition: Oxygen Status is outdated and and needs to be updated

//Precondition
//OxygenStatus = 5.5; Example outdated input

//Postcondition
//OxygenStatus = Should be any value other than 5.5

Testing For Error
Condition: Function should not return a negative value
//Precondition
//OxygenStatus = 5.5; Example output
//CollectOxygenStatus()

//Postcondition
//OxygenStatus = Should be any value other than negative, oxygen levels cannot be negative

Condition: The lowest possible value returned should be zero
//Precondition
//OxygenStatus = 5.5; Example output
//CollectOxygenStatus()

Functional Test One

- Fish - +CollectFishStats() : Double -> Collects the status of all the fish and creates an average score for them

Context: In this functional test we are testing to ensure that the method CollectsFishStats returns a double which represents the average maintenance difficulty (how hard it is to take care of the fish) score of the fish in the tank based on the needs of the fish.

Class Name: Fish
Method Name: Double: CollectFishStats()

Calls Method: Double: +SalinityReq()
Calls Method: Double: WaterTempReq()
Calls Method: Double: OxygenLevel()
Calls Method: Double: PHReq()
Returns: Double - Score

Testing For Correctness

Condition: In case it need to take all the data but actually calculated with missing data

//Precondition

SalinityReq = 0.5

WaterTempReq = 67

OxygenLevel = 8.3

PHReq = 7.0

//Postcondition

//Score = 1; Since these values are common values needed for fish welfare a low score is needed to show that it is easy to take care of this fish.

Testing For Error

Condition: It needs to take data to calculate for information but could not calculate properly

Condition: In case it need to take all the data but actually calculated with missing data

//Precondition

SalinityReq = 0.5

WaterTempReq = 67

OxygenLevel = 8.3

PHReq = 7.0

//Postcondition

//Score = Anything except an error or null. Any type of exception triggered may cause further errors within the system.

The CollectFishStats function will call results from SalinityReq, WaterTempReq, OxygenLevel, PHReq in the same class of fish, which are collected from class FishGroupStatus and call all the functions respectively. Then, it will calculate an average number from all the data to send it to class DisplayStats which is also sent to the database.

System Test One

- DisplayStats - +DisplayGraph() : Void

Context: In this system test we are testing to ensure that the method DisplayGraph properly displays the graph of all the fish related data.

DisplayStats
DisplayGraph(): void

Used Classes:

- Tank State
- Tank
- Fish
- FishGroupStatus
- EventMonitor

Testing For Correctness

Condition: Running and logging every event on a graph in the app

//Precondition

// Graph is not displayed

//Postcondition

// Graph is displayed showing logged events for fish

Testing For Error

Condition: Ensuring every single event logged in the graph is accurate

//Precondition

// Graph is not displayed

//Postcondition

// Numbers should be displayed and NULL should not be displayed on the graph.

Class : Function - Name	Testing For:
DisplayGraph() : void	Graph is up to date and aligns with current stats
TankState: +CollectTankStats : Double	Current stats are accurate and align with what is occurring inside the tank
App : void	Accurate graph is displayed in the app

Further Description:

To conduct a thorough system test for this function, our first step is to verify whether the app can successfully display the graph in our app. We can accomplish this by testing the DisplayGraph() function within the DisplayStats class, which retrieves data from the hardware sensor database. If the graph fails to appear in the app after it is called from the App class, we will need to revise our approach and retest until it functions correctly.

If the graph appears as expected, our next priority is to verify its accuracy and alignment with the actual tank state. We can compare the graph's values against external resources, such as a trusted thermostat, to confirm their validity. By comparing these values, we can establish whether the graph is reliable and make any necessary adjustments.

Assuming the graph is accurate, we can periodically verify the app's ability to update properly. If the values remain consistent, we can confidently conclude that the system is functioning correctly. Otherwise, further revision is required.

Test Set Two: Fish tank environment response system & App Authentication

Unit Test Two

- ChangeOxygenLevel - RaiseLevel() : Boolean

Context: In this unit test we are testing to ensure that the method ChangeOxygenLevel successfully raises the oxygen levels inside the tank.

Event <<Interface>>
ChangeOxygenLevel
Boolean: RaiseLevel()

Testing For Correctness

Condition: If the Event monitors the Oxygen level is too high then it should change it lower

//Precondition

// Oxygen level is higher than requirements

//Postcondition

// Oxygen levels meet requirements, function returns true

Condition: If the Event monitors the Oxygen level is too low then it should change it higher

//Precondition

// Oxygen level is lower than requirements

//Postcondition

// Oxygen levels meet requirements, function returns true

Testing For Error

Condition: In situation the Monitor should change the Oxygen level but do not

//Precondition

// Oxygen level is lower than requirements

//Postcondition

// Oxygen levels remain the same, false is returned;

Condition: In situation the Monitor should not change the Oxygen level but do anyway

//Precondition

// Oxygen level is at normal levels;

//Postcondition

// Oxygen levels are higher or lower than normal, false is returned.

The Event should monitor constantly the Oxygen level of the tank whether it meets the requirement of presets. If any changes happen, it will change the level accordingly to either lower or higher. After changes, the event will keep monitoring the post-change whether it did correctly or not. All will be compared to the database and anything changes happen will report to EventMonitor then it could send Push notification to the app for the user to monitor.

Function Test Two

- App - Login(String : Username, String Password) -> +checkUsername() : boolean, +checkPassword() : boolean

Context: In this unit test we are testing to ensure that the method Login successfully raises the oxygen levels inside the tank.

App - Login(String : Username, String Password)
+checkUsername() : boolean

+checkPassword() : boolean
Returns boolean based off login status.

Testing For Correctness

Condition: The App class should allow users to sign into their account

//Precondition

// The user is not signed into their account

//Postcondition

// The user is signed into their account

Condition: Personalized settings and information is preserved from last sign in

//Precondition

// Settings are not accessible before login.

//Postcondition

// Settings are now accessible after login.

Testing For Error

Condition: Signing in should direct the user to a home page

Condition: Info matches last sign in

Condition: Logging out should direct user to the login screen

In this system test, our objective is to verify the login feature's functionality within the fish tank's app. The first step is to confirm that a user can successfully create an account. This entails storing the user's information in a database that we can access from the app when the user logs in.

Next, the App class accesses the Account class to find if the user's input matches with the stored usernames and passwords. If a match is found, the user should be granted access to the app's features. If no match is found, then an error message should appear. In the event a match

is found, we will proceed to adjust various settings within the app and verify that they are maintained after the user logs out and logs back in.

To accomplish this, we must first confirm that users can log out of their account in the first place and then log back in. Upon logging out, the user should be redirected to the login screen, where the login process can be repeated. Once the user logs in, their previous settings should be preserved.

System Test Two

- EventMonitor - +logEvent(Event eventName): void

Context: In this system test we are testing to ensure that the logEvent function identifies, executes, and logs an event.

EventMonitor
Double: OxygenStatus
void logEvent(Event eventName)

Testing For Correctness

Condition: Running and logging an event that results in the successful change of the water temperature levels.

//Precondition
// CollectWaterTempStatus 70
// Collect WaterStatus 60

//Postcondition
// ChangeWaterTemp is executed successfully and is logged

Condition: Running and logging an event that results in the unsuccessful change of PH levels due to missing supplies.

//Precondition
// CollectPHStatus 7.7
// CollectPHReq 6.0

//Postcondition
// ChangePH is successfully executed and logged.

Class : Function - Name	Testing For:
EventMonitor : logEvent(Event eventName)	Event is identified, created, run, and logged
ChangeWaterTemp: +RaiseLevel() : boolean	WaterTemperature is raised successfully;

Class : Function - Name	Testing For:
TankState : CollectWaterTemp() : Double	WaterTemperature from tank is collected
FishGroupStats: CollectWaterTempStatus() : Double	Water Temperature requirements from fish is collected
TankProfileNeedsAttention : getDescription	The profile for what a healthy tank looks like is collected

Class : Function - Name	Testing For:
+CollectPHStatus() : Double, CollectPHStatus	Collects the PH status needs of the grip.
<<Interface>> Event : ChangePH	Changes the PH level inside the tank.

Database Management Strategy

Data Management Strategy:

For our fish tank management system we plan to use SQL as our main database system. Our system will store two types of data, information and telemetry statistics relating to the current fish tank status and potentially sensitive user information. If our user data were to be compromised and released into the public we could be the target of a lawsuit of loss of public trust, as a result our team plans to isolate the user and non-user data into separate databases. In our system we decided to use three different types of databases separated by the type of information stored.

We have a database for:

- User Information
- Tank State
- Fish Library

We isolated these components because each component is uniquely different, in addition if we were to centralize every database creating rigid permission levels would be difficult. We would expect that the user information database would be limited in its exposure to the internet and would assign it different security permissions than the tank state or fish library. Fish Library and Tank State hold non sensitive data, as a result it would not cause significant impact if this information were to be exposed publicly.

Trade Off Discussion:

Our team has also investigated other alternative options such as NoSQL and have concluded that SQL takes precedence NoSQL due to its need for structure and hierarchy. More specifically implementing a structure and hierarchy of tanks and fishes could be done significantly easier using SQL allowing for easier implementation and code review. NoSql is a newer technology and could be considered in the future but our team decided to pick SQL due to its standing use. We anticipate a long service life for each of our tanks and as a result have decided to pick SQL due to its longer development as a result we expect to discover less bugs with it. In addition because

SQL handles complex queries significantly better this will allow our developers to easily request tank environment variables to produce the relevant results. For our software of choice we have decided to select Microsoft's MySQL server due to their dedicated development team. For our table contents we designed our classes to work well with each database so that data stored in each database relates directly to the UML diagram.