

# Web Application for Aqualab Sensor Monitoring and Analysis - Milestone 6

Ruth Garcia, Haley Hamilton, Greg Thompson

# Milestone 6 Overview:



- Implement, test, and demo **final UI additions/styling**
  - Included adding final tweaks to the frontend including clearer ranges, the appearance of charts, CSV file downloading format, and the calculated data relationships.
- Implement, test, and demo **user roles and permissions**
  - Implementing JWT tokens and flask decorator functions to associate a client with a user and restrict/grant access to features based on their user role.
- **Final system integration** and error handling
  - Implemented stop run button, change range and change frequency features, updates after testing with live sensor and testing different “program recovery after shutdown” scenarios



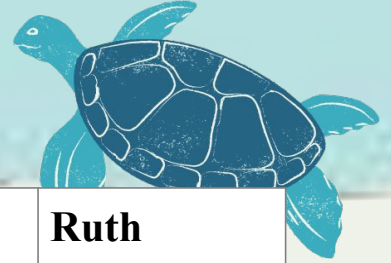
# Milestone 6 Overview:



- Implement, test, and demo of **the entire system**
  - Tested system with lab sensor multiple times, everything is functional.
- **Conduct evaluation and analyze results**
  - Tested 10 different features and UI pages cases with 7 lab volunteers and three different roles
- **Create user/developer manual**
- **Create demo video**
- **Make app more accessible remotely and Create a User Logging Feature**
  - Did not have enough time to complete this, future plans to help lab team/next project iteration are in the works!



# Milestone 6 Progress Matrix:



Task	Greg	Haley	Ruth
Implement, test, and demo <b>final UI additions/styling</b>	0%	80%	20%
Implement, test, and demo <b>user roles and permissions</b>	0%	70%	30%
<b>Final system integration and error handling</b>	60%	40%	0%
Implement, test, and demo <b>of the entire system</b>	30%	50%	20%
Conduct evaluation and analyze results	33%	33%	33%
Create user/developer manual	80%	20%	20%
Create demo video	0%	80%	20%
Make app more accessible remotely	-	-	-
Create a User Logging Feature	-	-	-

# User Manual

- Lengthy document to explain the system from both user side and developer side
  - The User component explains all action needed to install, configure, and run the application.
  - The Developer component contains a description of the purpose of each file, important variables, and the expected execution flow.
- Continuing Project
  - Dr. T intends to continue this project in the future with more Computer Science students.
  - The objective of this manual is to provide a future team the ability to understand and effectively modify our software.

Table of Contents	
<b>User Manual</b>	<b>1</b>
Setup	1
Downloading Files	1
Installing MongoDB	1
Installing Python and related libraries	1
Beginning Experiment	2
Connecting COM Ports	2
Starting MongoDB	2
Running From the Command Line	2
Accessing While Running	2
Expected Console Messages	2
Remote Access IP	2
Admin Privileges	2
Configuring Charts	2
Accessing Data	3
Shut Down	3
How to Shut Down from Remote Access	3
How to Continue After Shutdown	3
Known Issues and Workarounds	3
Reset Errors	3
<b>Developer Manual</b>	<b>4</b>
System Formatting	4
Overview	4
Back End	4
main.py	4
app.py	5
mail_server.py	5
sys_state.py	5
random_test_sensor.py	6
db_config.py	6
w_sensor.py	6
a_sensor.py	6
Front End	6
App.jsx and App.css	6
Navbar.jsx and Navbar.css	6
Analysis.jsx and AnalysisTool.css	7
ChangeRangeForm.jsx	7
ChangeSettingsForm.jsx	7
ConfigSensorsForm.jsx and ConfigSensorsForm.css	7
CreateUserForm.jsx	7
EditUserForm.jsx	7

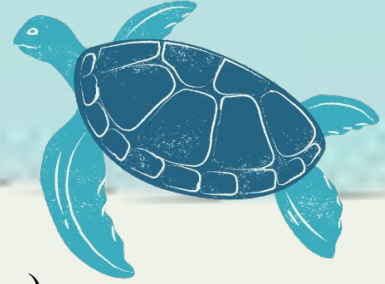
# Crash handling



- The threads have each been given an escape if a crash is detected
  - This prevents zombie threads from persisting after a fatal error in the main system
- Fatal errors in any thread besides the webapp and sensor handler are recoverable during runtime
  - The threads are reactivated with the same parameters and the error is logged
- Crashes in the main thread do not lose data
  - As the database persists separately to the system, it continues operation after the main thread crashes
  - Any sensor thread actively reading or writing a value will be allowed to finish before their escape triggers
  - This protects the system from data corruption as a result of a software crash
- Operating System shutdown is a risk
  - Events such as power loss or BSOD could shut the program down suddenly, creating corrupted data points
  - These will be visible and removable by manual inspections of the latest data upon restart



# Final System Integration



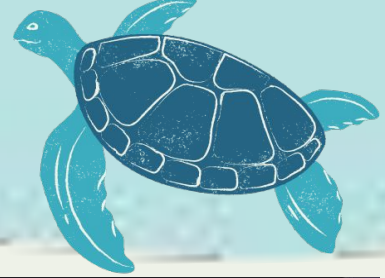
## Includes:

- Full implementation of final features (change range, change frequency)
- Implementing a stop run button
- Testing and making changes for “Program recovery after shutdown” scenarios
- Made changes to ensure proper connection to sensors
- Client explained that 1 water sensor would read both dissolved oxygen(DO) and carbon dioxide (CO<sub>2</sub>) and both we need to be monitored.
- The needed user interface, backend, and database changes were made to accommodate this





# User Roles/Permissions



- Implemented JWT token creation at successful login
- Token contains user id and role
- Basic functionality:
  - User tries to complete action
  - Client token sent to the backend with the action request
  - Role is verified can action completed / receives alert

```
# This route updates a high/low range values for a sensor in
@self.app.route("/change_range/<id>", methods=["PATCH"])
@require_role(["admin"])
def change_range(id):
    sensor_id = {"_id": ObjectId(id)} # Correctly format the
    existing_sensor = sensor_collection.find_one(sensor_id) #
    if not existing_sensor:
        return jsonify({"message": "Sensor not found"}), 404
```

The screenshot shows a web application interface. At the top, a dark notification box displays the message "localhost:5173 says" followed by "You do not have permission to access this resource." and an "OK" button. Below this, a modal window is open for configuring sensor ranges. The modal has a title bar with "Baud Rate" and "Range - CO2". Inside, there are two sections: "CO2 Range:" and "DO Range:". Each section has "Low:" and "High:" labels with corresponding input fields. The "CO2 Range:" section shows "Low:" with a value of 0 and "High:" with a value of 10. The "DO Range:" section shows "Low:" with a value of 0 and "High:" with a value of 10. At the bottom of the modal is a blue "Update" button. The modal also has a close button (X) in the top right corner.



# Test/Demo Entire System:

Tested the system hooked up with the sensor:

- Made changes to ensure proper sensor connection
- Let system run/played with sensor water to monitor data
- Tested program recovery scenarios (unplugged sensor, sensor reboot, etc...)



# Full System Demo:



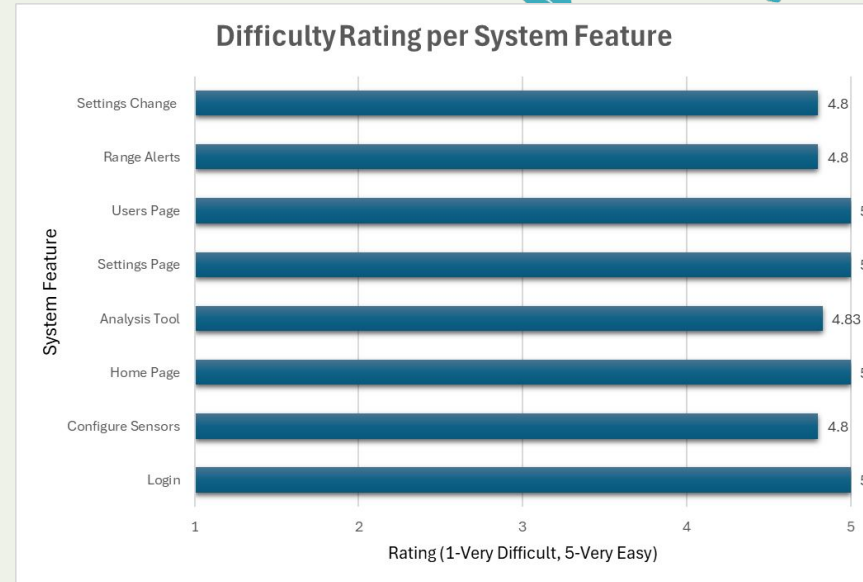
[https://www.youtube.com/watch?v=3nbFlJ7X27o&ab\\_channel=HaleyHamilton](https://www.youtube.com/watch?v=3nbFlJ7X27o&ab_channel=HaleyHamilton)



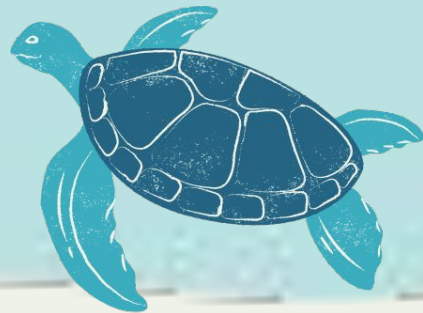
# UI and User Acceptance Testing/Evaluation:



- We did not get to user logging
  - Will implemented by a future team
- User Acceptance was completed
  - 2 admin
  - 4 observers
  - 10 different testing scenarios
- Analysis?
  - Overall: Client very satisfied!
  - Our Change Range button should be 'louder'
  - Easy to navigate and quick to understand UI
  - Alerting system on screen could be more obvious



# Lessons Learned?



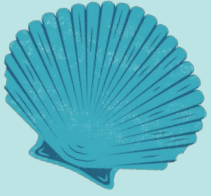
## **Importance of agile development process:**

- Difficult to align schedules
- Easy to lose sight of tasks that were/needed to be completed for the milestone
- Originally used JIRA, but it was difficult to keep updated and became extra work
- Important to remember how helpful weekly scrum meetings and daily stand ups can be.

## **Importance of planning:**

- difficult to plan a complex system with a lot of features upfront
- Would have benefited the project to spend more time in this area
- Ex: main program backend and architecture of the React frontend
- Moments we lost sight of some intended features/functionality and were not implemented in the easiest/scalable/best way





Questions?

