CSCI 370 Final Report

Walker Water 1

Tyler Emken
Phi Stanton
Erik Swanson
Wallace Schageman

Revised June 13, 2023

CSCI 370 Summer, 2023

Amelia Read

Table 1: Revision history

Revision	Date	Comments
New	May 21, 2023	Completed Sections: I. Introduction II. Functional Requirements III. Non-Functional Requirements IV. Risks V. Definition of Done XII. Team Profile
Rev – 2	May 28, 2023	Completed Sections: VI. System Architecture
Rev – 3	June 4, 2023	Updated Sections: II. Functional Requirements V. Definition of Done VI. System Architecture Completed Sections: VII. Software Test and Quality VIII. Project Ethical Considerations
Rev - 4	June 11, 2023	Updated Sections: I. Introduction II. Functional Requirements III. Non-Functional Requirements IV. Risks V. Definition of Done XIV. Team Profile Completed Sections: IX. Results
Rev – 5	June 13, 2023	Updated Sections: I. Introduction II. Functional Requirements III. Non-Functional Requirements IV. Risks V. Definition of Done VI. System Architecture Completed Sections: X. Project Completion Status XI. Future Work XII. Lessons Learned XIII. Acknowledgements

Table Of Contents

I. Introduction

a. Key Terms

Table 2. Key Terms

Microsoft Access	Local database management interface software.
Lotus 1-2-3	Database management software that handles data in the form of spreadsheets. Released in January of 1983, Microsoft Access usage support was ended in 2010, product discontinued in 2013.
PlanetScale	Database hosting site created by the same group who developed next.js
Prisma	Typescript Create Read Update Delete (CRUD) language with Object Relational Mapping (ORM) that allows creating queries for databases in typescript.
React	Open source javascript front end website development language.
Materials UI (MUI)	Node.js package with many pre-built react components.
Vercel	Front end cloud platform service for website deployment maintained by the Next.js group

b. Client & Current State of their software

Our client is Walker Water, a water management and irrigation company that manages supplying water from reservoirs to the users who have ownership/stock in them. They manage over 70 reservoirs that supply water to over 130 transport ditches, which around 2000 users receive their water from.

All their data is stored in old applications and requires users to either come into the office or call over the phone to place orders. They use a Microsoft Access app to access, add, and update any data stored in a Lotus 1-2-3 app. The current system does not have any interfaces to make handling the data easy or safe. For example, to create a new user, the current admins must manually add a new line in the user database and create a unique key for each user. The current system also has several bugs that inhibit usability, making it difficult to do any updates to the data. Some such bugs result in the software crashing when trying to update data.

c. Project Description

For this field session, we were tasked with overhauling and modernizing their current system to a web application, allowing them to manage the data online, as well as allow users to place water orders online.

The three main aspects of this project are: Overhauling and migrating their current database; building an "Admin portal" which has functionality to manage, update, and add data; and creating a "User portal" which has much more limited functionality for placing/updating orders, as well as viewing their own order history.

d. Data Source

i. Existing Database

Currently, much of their database exists in Lotus 1-2-3, and is accessed by a Microsoft Access App. The tables that exist on this app can be stored or exported as Microsoft Excel spreadsheets, or as comma separated files (CSV). Other pieces of their data are left in pdfs that they refer to by finding information on the pages manually.

ii. Existing App

The Microsoft Access app provides a very basic interface with labeled buttons that pull up the corresponding spreadsheet. These buttons lack any contrast with the background and have redundant data, for example, one such pair of buttons displays the same data but sorted by a different column.

e. Stakeholders

i. Water distribution:

  i. Walker Water

Walker Water is our primary stakeholder, as the goal of the project is giving them an application that has better functionality than their current setup. They will interact with our app the most and use it to give the other stakeholders help in the water ordering system.

  ii. Water commissioners

The water commissioners are the workers who go to the reservoir headgates and distribute water according to their reports and tickets. Our app is going to be used to automatically generate these tickets and reports rather than having them manually created by the admins at Walker Water. Because of this, they are another major stakeholder to consider with our project, as they directly use outputs from our software.

ii. Water consumers:

The other group of stakeholders to consider are the water consumers, as in the farmers and companies who use Walker Water to order their share of water from the reservoir systems. They rely on them to manage their appropriations, and to allot them credits based on the ownership they have in individual reservoirs. We must consider how these users will be interacting with our system and ensure that they are able to place and view their orders, as well as view their current balances as effectively and cleanly as possible.

f. Maintenance

i. Walker Water

  i. The current system they use is being supported by Walker Water
  ii. The system that we set up for them will need to be supported by them as well, and was made with maintenance in mind

The current system being used by Walker Water has extremely poor maintenance and a severe lack of documentation. They have frequent I.T. problems that need resolving, as the state of their app is degrading. We were tasked with creating a system that is much easier to maintain and upkeep so that these problems do not pop up as often.

To do this, we created thorough documentation in our software that thoroughly commented and documented what individual components and functions do, as it is likely that this project will be passed on to other teams to complete. Ensuring that we have documentation on how to set the project up, the dependencies that need to be installed, as well as the steps to run/download the databases were key.

II. Functional Requirements

All requirements listed were presented as part of the first project pitch. Highlighted requirements are part of the minimum requirements for the app. These requirements were then split up into ones that were decided to be beyond the scope of what our team would be able to do within the five-week period we had to work.

Requirements:

a. Database holding all tables/relations for user data, admin data, ditch data, reservoir data, and shareholder data.
b. Admin tool/Admin login
    i. Allow user profile setup.
    ii. Personalize user profile accounts with information such as what ditches the user has access to.
    iii. How much water the user owns or has leased.
c. Security
    i. Personal customer information is secure.
    ii. Administrator information is secure.
    iii. Customer does not have access to administrator tools.
d. Implement Rules for orders to be approved.
    i. Orders to not exceed amount of water available.
    ii. Minimum cumulative run amounts of water in the ditch
    iii. Red/green light to show rules have been approved and order went through.
    iv. Ditch specific constraints
e. Water order tickets for ditch riders
f. Keep track of usage through credits and debits to the remaining balance
g. Order history or look back.
h. Order and usage reports
i. Administration reports
j. Summary views of days remaining in the irrigation season and days of water left/net amount remaining.
k. A lease portal with agreement forms for both parties
l. Track run days or duration of the order from in the user's order history
m. Admin and user should be able to edit their current orders and future orders

Requirements Out of Scope:

a. Allow the user to tailor some aspects of their own account.
b. An interactive map showing their ditch and other orders in the ditch.
c. A calendar interface to track how many run days or duration of the order and other orders within the ditch.
d. Payment Method
e. Planning tool for ordering remaining water through the end of the season.
f. Water transit loss calculator
g. Flume/flow rate calculator
h. Usage monitoring tool/efficiency tools
    a. Type of crop being watered.
    b. Water per acre
i. 7-day weather forecast to see how the weather will affect water orders

III. Non-Functional Requirements

a. Research topics
   i. HTML
   ii. CSS
   iii. JavaScript
   iv. SQL
   v. Next.js
   vi. Prisma & Database hosting
   vii. User authentication

b. Easy to use user interface on both the admin and user side

c. Product delivered with quality, and is pleasant to work with

IV. Risks

a. Lack of experience in web design

Many of the team members on this project had little to no experience working on web applications, as well as little experience using the different languages required to work on sites, such as HTML, JavaScript, and TypeScript. This was a risk that continuously came into play throughout our project, with a moderate impact on our ability to complete requirements and deliver a working product. To combat this, we did extensive research and tutorial work so that each team member is more comfortable with web design.

b. Minimal experience accessing databases through different interfaces.

Our team has little experience accessing databases outside of systems such as PostgreSQL. This risk is relevant as we intended to migrate their data to a cloud-based server and need a way for our site to securely push and pull data from it. We took time to do extensive research to ensure our cloud server is secure so that it is protected from data breaches and that only the data needed is provided to the users.

c. Data release may be blocked by Colorado Mesa board of directors

Some data required for us to migrate may be blocked by the Colorado Mesa board of directors. This outcome was guaranteed and to manage this impediment a Non-Disclosure Agreement (NDA) was drafted to permit the team to view and manage user data securely. Regardless of the NDA granting access to the information, development was somewhat uninhibited as we could use knowledge of what information is necessary to begin creating the structure of our system.

d. Client has vague description of desired software appearance

Our client tasked us with creating the UI on our own, as UI design is outside of their ability. This was a guaranteed risk, as it was part of the project that they set up and explained during our first meeting. Because our product is to be used frequently by our client, as well as many users who may not be tech literate, we aimed to ensure that the site is as simple and easy to use as possible. We held weekly meetings with our client to prove the state of the site and its visuals to ensure it is going in a direction they are happy with.

e. Lack of web app security

This risk is the most dangerous and was likely to be seen due to lack of experience with web app security. The potential for leaks of personal user data, or regular users being able to use administrator problems is a risk that we needed to seriously address as it would have severe consequences if not properly handled. Our team performed extensive research into authenticating accounts, and restricting what data is accessible to which users.

V. Definition of Done

a. Minimum Useful feature set

- i. Updated Database
  - a. Only necessary data is preserved and formatted for best storage and use
  - b. Cloud/Server storage
  - c. Instructions for migrating data that we do not have full access to, or not in a format that is migratable
- ii. Admin Portal
  - a. View all clients, ditches, share distributions, and orders
  - b. Create, approve, and change user accounts
  - c. Place orders for users
  - d. Receive and view submitted user forms
  - e. Transfer shares between users
- iii. Customer Portal (semi-optional milestone)
  - a. Create an account that is sent for approval
  - b. View remaining balance, current order information, order history, and approved ditches
  - c. Place orders
  - d. Download forms and upload completed forms for review

b. Client Testing

- i. Revision of interface appearance and ease of use

c. Product Delivery

- i. Our product is delivered to them for further development. The site is not in a state to be deployed as there is admin functionality not yet implemented
- ii. Our goal was to have our final delivery for their project be the codebase and database that we have set up for them, as well as instructions for migrating it to their own accounts rather than be hosted on our own GitHub and PlanetScale accounts.

VI. System Architecture

a. Technical Issues

i. SQL Database Hosting

As a team, we needed to host a database in a way that all of us could access, instead of locally hosting our database on our individual setups. After some research, we decided on using PlanetScale to host our database, as the size of it would not be an issue, with the ability for our site to connect to it through Prisma.

ii. Framework for site development

Originally, we planned to use Bootstrap for development, because it was simple and automatically handled formatting for devices with different screen sizes. This had its own set of challenges, as the form that users would use to order water needed a set of constraints that would be difficult to implement on bootstrap. We ended up using Materials UI (MUI) to implement these constraints more easily, specifically with the restrictions on what days water could be ordered.

b. Architecture Description & Designs

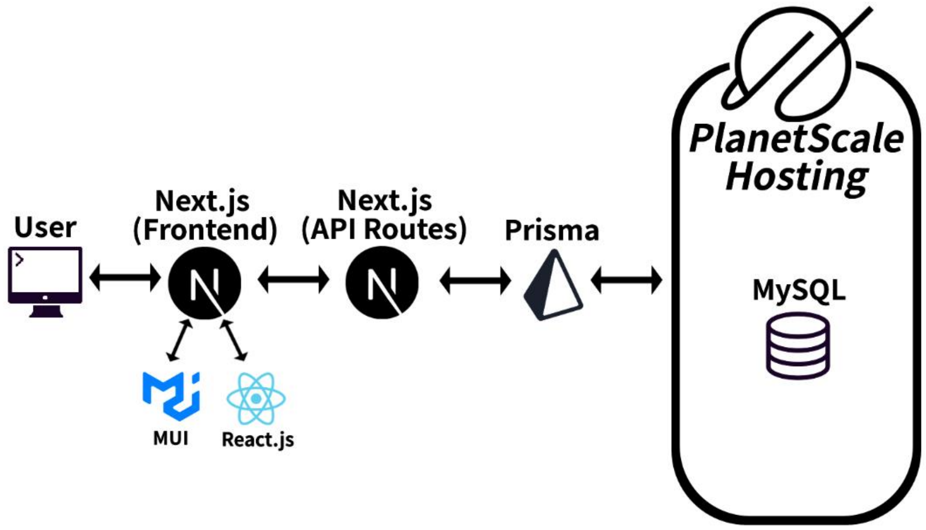
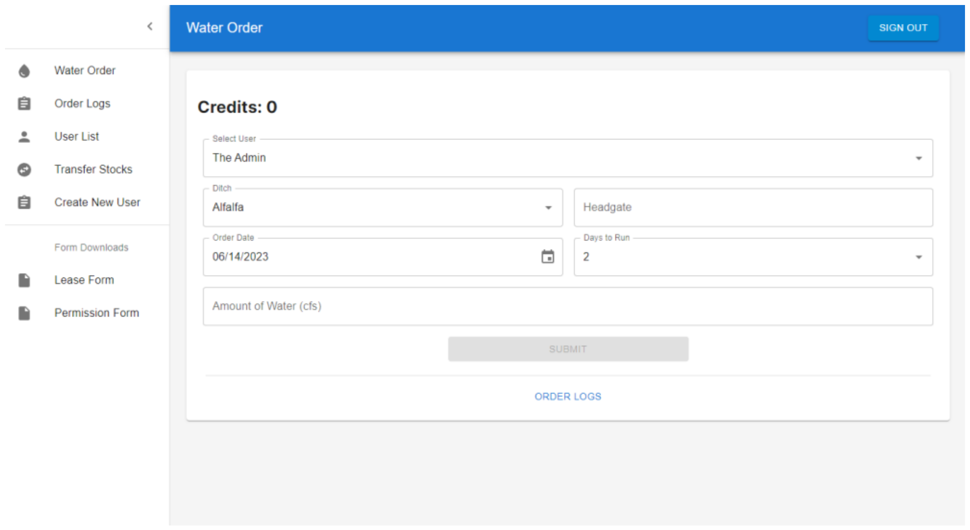


Figure 1. System Architecture Diagram

The architecture for the water ordering system web application can be broken down into two distinct parts. The front end and the back end. The front end of the water ordering system is built primarily on React.js and MUI as a third party React component library. React.js is used as the foundation for our front end (styling, organization, etc.), while MUI serves to give the webpage interactable components. These components include date pickers, buttons, and interactable text boxes. Some of the pages admins interact with are shown below. For more admin pages, user pages, and sign in/out pages, see appendix A and B.



Credits: 0

Select User: The Admin

Ditch: Alfalfa

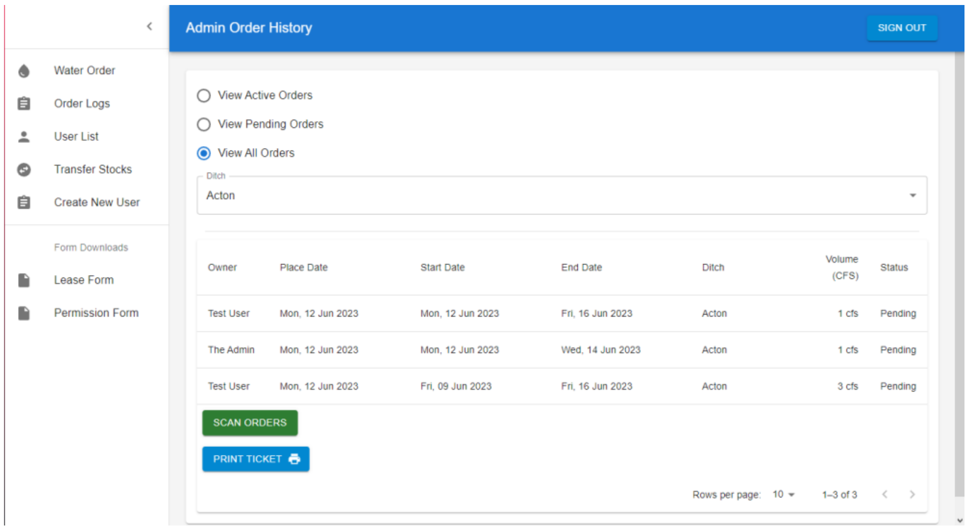
Headgate

Order Date: 06/14/2023

Days to Run: 2

Amount of Water (cfs)

Figure 2. Admin Water Order Page



View Active Orders
View Pending Orders
[✓ View All Orders]

Ditch: Acton

Owner	Place Date	Start Date	End Date	Ditch	Volume (CFS)	Status
Test User	Mon, 12 Jun 2023	Mon, 12 Jun 2023	Fri, 16 Jun 2023	Acton	1 cfs	Pending
The Admin	Mon, 12 Jun 2023	Mon, 12 Jun 2023	Wed, 14 Jun 2023	Acton	1 cfs	Pending
Test User	Mon, 12 Jun 2023	Fri, 09 Jun 2023	Fri, 16 Jun 2023	Acton	3 cfs	Pending

Rows per page: 10 1–3 of 3

Figure 3. Admin Water Order Logs

The backend consists of two main components: the Node.js server and the MySQL database. The Node.js server is currently hosted locally and uses Prisma to communicate with the database. The server is hosted on PlanetScale, which is connected to our local MySQL database and sends data to be stored in the database. Data also moves in the other direction (database, to server, to Next.js, to front end). On top of everything, we use the Next.js framework. We use Next.js for our server-side data collection and static website generation. Included below is the ERD design document for our MySQL database.

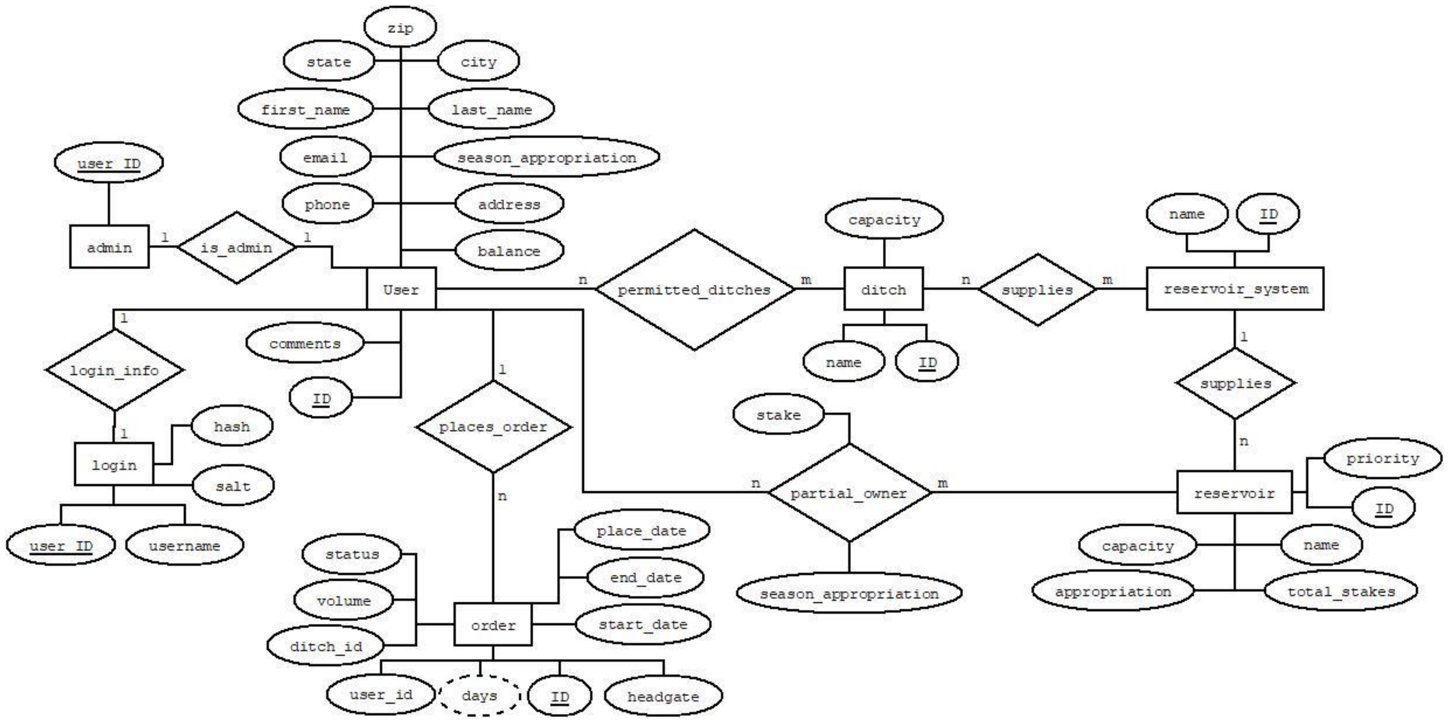


Figure 4. Database Entity Relational Diagram

VII. Software Test and Quality

<u>Tests:</u>

- **a. Database read and write testing**
  - i. This test is to test the read and write functionalities of our Prisma schema.
  - ii. This test uses a singleton mock copy of our schema to test read and write methods without writing to the actual Prisma schema.
  - iii. To unit test our Prisma schema within a typescript file we used Jest. Jest is a unit testing package for typescript.
  - iv. For this series of tests to pass, and for the database to meet the accessibility requirements, every table must be able to always read and write data. We created tests accordingly.
  - v. The only edge case for these tests is testing the password in the login table. The hash and the salt fields must match in the mock Prisma schema and the generated value for the test to pass.
- **b. Admin tool/Admin login testing**
  - i. This test is to test the admin tools and admin functionality of the web app.
  - ii. To thoroughly test the functionality of the admin panel, we must test: the login loading the admin portal, user account search, account approval, and form view and approval.
  - iii. No specific tools are used for these tests.
  - iv. For these tests to pass an account with admin privileges should be able to login and see panel options viewable only to admins; search and find any user out of the database; view pending accounts, approve/deny those accounts, and those accounts update accordingly; forms uploaded by users can be viewed, approved/denied, and update those forms' status accordingly.
  - v. There are no edge cases for these tests.
- **c. User account creation/User Login testing**
  - **i.** The purpose of this test is to test if user accounts are being created and whether users can login to the system or not.
  - ii. To test these functionalities, we must check the database for specific user accounts when created. We must also be able to login to test/dummy user accounts with the corresponding username and password.
  - iii. There are no specific tools used in these tests.
  - iv. For these tests to be acceptable, every user created must be able to be queried from the database, as well as every dummy user having the ability to login with their corresponding credentials.
  - v. There are no edge cases for these tests.
- **d. Security testing**
  - i. The purpose of this test is to test our security functionality so that personal user information is not compromised.
  - ii. To test security functionality, we must ensure that user created passwords are encrypted. As well as personal user information. To do this, we will use various penetration tests.
  - iii. There are not any specific tools that we are using in security testing. We will again use dummy user accounts (not real users) to test security.
  - iv. For these tests to meet the threshold of acceptability, user accounts must be secure and only accessible by said user.
  - v. There are no edge cases in these tests.
- **e. Order approval testing**
  - i. This test ensures that the user's orders are safely stored in the database and can then be approved by the site admins for an actual order.
  - ii. To test this order system, we will make sure that orders made by users are properly formatted and stored in the database when an order is made and can be properly updated by the user.
  - iii. There are not any specific tools that we are using in order approval testing.

VIII. Project Ethical Considerations

a. **Which ACM/IEEE Principles are particularly pertinent to the development of your product? Why?**
    i. ACM Principle 1.6: Respect Privacy
        i. The project we undertook was updating our client's current database setup, allowing it to be more easily used, changed, and accessed. This is less of an update and more of a complete overhaul of their current system. Because of this, we dealt with handling user data and information such as: accessing their reservoir ownerships, email & phone-number, and other sensitive information that they are currently storing.
    ii. ACM Principle 3.6: Use Care when Modifying or Retiring Systems
        i. The current database system will deprecate upon this project's completion with a future team. As such, all functionality that Walker Water currently has or may potentially have must be accounted for and included in the new system. Additionally, all stored data must be migrated such that the data needs are maintained.
    iii. IEEE Principle 2.01: Client and Employer
        i. Our work on this project was not only working as software engineers, but also as consultants for our client. We helped to form and guide the requirements related to the design and some of the functionality of the features, as our client does not have much experience in web applications and creating clean user interfaces and the current system was not entirely understood.

b. **Which ACM/IEEE Principles were most in danger of being violated? What were the negative impact if these Principles were indeed violated?**
    i. ACM Principle: 3.6: Use Care when Modifying or Retiring Systems
        i. The negative impacts of violating this principle range from mild to relatively severe. On the milder side, if overhauling their system goes poorly then the project requirements would have failed to be met, as updating their system is one of the main functional requirements. On the more severe side, if the new system does not properly handle and account for security risks, then user data and privacy would be violated, letting unauthorized users access other user information.
    ii. IEEE Principle 2.01: Client and Employer
        i. In terms of negative impacts relating to violations of this IEEE principle, many of them would mostly come down to client dissatisfaction and not meeting the requirements that they were expecting. Communication between us and the client is of utmost importance, especially because we are working with them to make recommendations for what would make the site easy to use.

c. **Apply two Tests articulated by Michael Davis to your product.**
    i. Publicity Test:
        i. If our product were launched and appeared on the front page of a newspaper, it would most likely be for positive purposes. Unless something goes wrong with implementation, the product and choices made are beneficial to the client and all their users.
    ii. Professional Test:
        i. The Software Engineering code of ethics states that the product should be designed with public and client interests in mind. Our product is built to fulfill client requirements, which in turn are created to make it easier for users to order their water, as well as make it easier for our clients to manage their user's orders and data to serve them more efficiently.

d. **What are the ethical considerations for your project if your software quality plan is not implemented properly (or is not comprehensive enough)?**
    i. If our software quality plan was not implemented properly around data security, our product may suffer from poor security surrounding sensitive user data. One of the issues could be that users are able to flag themselves as an administrator in the database and have access to all the admin tools that allow them to change ownerships and orders within the database, as well as have access to

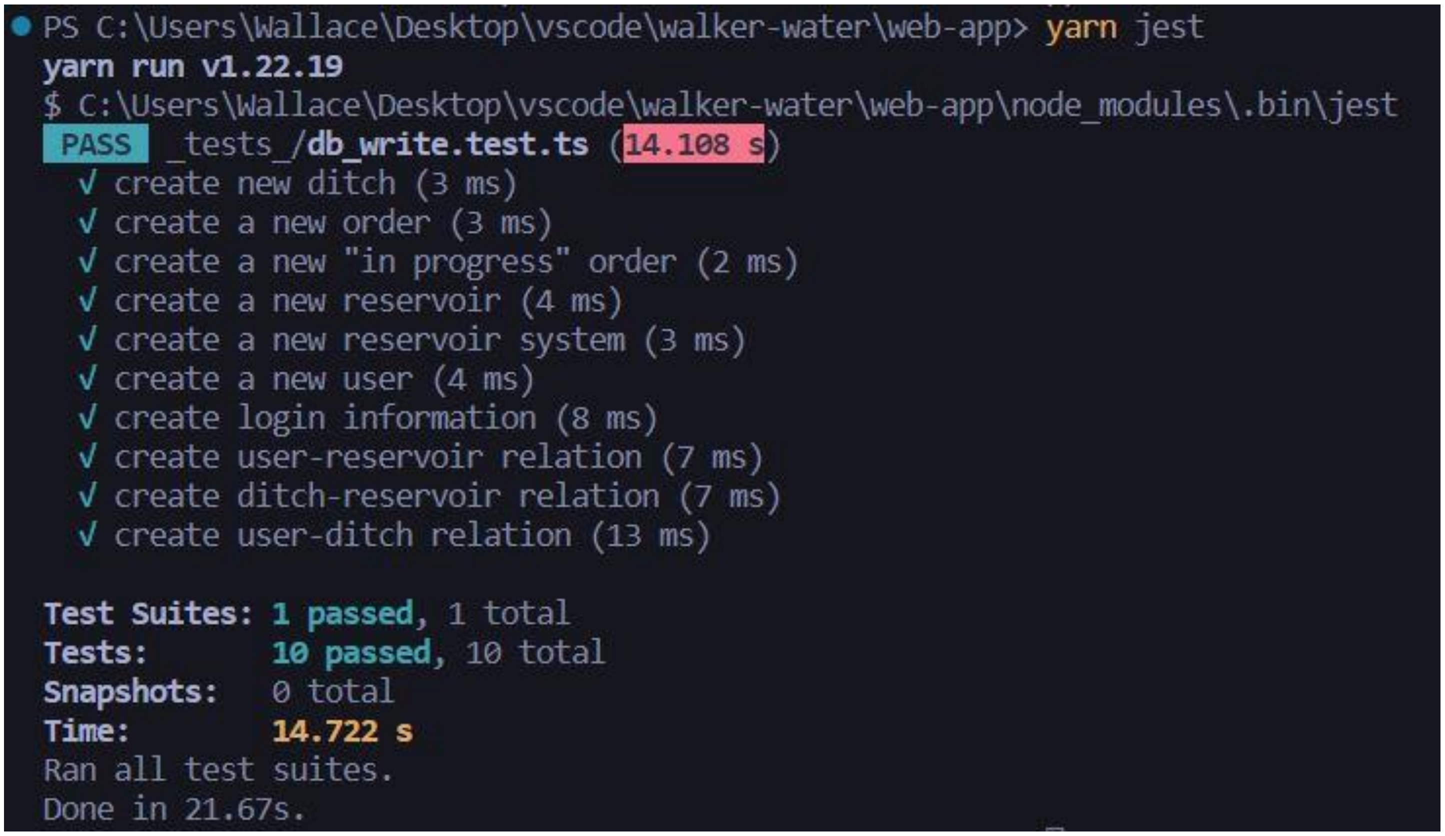
every other user. To ensure that this cannot happen, we ran extensive testing on security with edge case tests.

ii. Our software quality plan also needed to be implemented properly for our visual design as well. While not having as severe consequences as a data breach if not properly implemented, there are still ethical problems that arise. A difficult to use UI would potentially alienate many users, especially older ones who are not as tech literate, resulting in discrimination about who our product is able to benefit, as well as not meeting the requirements our client gave for this project.

iii. Maintenance was also considered heavily as a part of our software quality plan. This project is a first step in overhauling their current system and will need to be updated and given more tools in the future. Because of this, a failure in our software quality plan could result in the entire system being extremely hard to maintain for future teams working on it, causing major problems down the road if they need to rework or change anything on the project that we started.

IX. Results

The results of our write Jest unit testing for our Prisma database schema can be seen below:



```
● PS C:\Users\Wallace\Desktop\vscode\walker-water\web-app> yarn jest
  yarn run v1.22.19
  $ C:\Users\Wallace\Desktop\vscode\walker-water\web-app\node_modules\.bin\jest
  PASS  _tests_/db_write.test.ts (14.108 s)
    √ create new ditch (3 ms)
    √ create a new order (3 ms)
    √ create a new "in progress" order (2 ms)
    √ create a new reservoir (4 ms)
    √ create a new reservoir system (3 ms)
    √ create a new user (4 ms)
    √ create login information (8 ms)
    √ create user-reservoir relation (7 ms)
    √ create ditch-reservoir relation (7 ms)
    √ create user-ditch relation (13 ms)

  Test Suites: 1 passed, 1 total
  Tests:       10 passed, 10 total
  Snapshots:   0 total
  Time:        14.722 s
  Ran all test suites.
  Done in 21.67s.
```

This series of tests tells us whether the data written to the database and its respective tables match the values needed for each table. For example, the ditch table needs an ID (integer) and a name (string). If an ID and a name are not provided, or extra data is given, the "create new ditch" test will fail. If tests are failing, it would prompt us and future developers to know that there are issues with writing to the schema, and either the schema, or the query being written would need to be altered. As mentioned above, the tests use a different schema to read and write data that is a complete copy of the original schema. This is so we do not actually read and write data to the real schema when running our tests.

For our client feedback on visual UI, we met weekly to demo the site's aesthetic and function. Our clients liked the site's restrictions and constraints making it difficult for users to enter bad data. Their feedback mostly centered around different functionality that they wanted implemented, or changes around how certain functions were implemented. Overall, the visual UI was considered pleasing, and so we will be keeping the current style of the site.

X. Project Completion Status

Unfortunately, many of the features for the online water ordering system had to be scrapped, and many more were not able to be completed. The number of functional requirements ended up being far too great to be able to finish in a 5-week period. As a team, we did end up getting a decent head start on the project, opening it up for future development and additions. The database structure was complete, the vast majority of its interaction was created, and the webapp has some features implemented. Several of the unimplemented features were requirements that we discussed during our first meeting with our client and were marked as stretch goals in case the primary features were implemented first.

<u>Implemented features:</u>

a. Updated MySQL database using Prisma, hosted on PlanetScale
b. Working NodeJs backend
c. Most (not all) functional database queries
d. Admin and User logins
e. User password encryption
f. Placing orders for users
    a. Placed orders debit user accounts
g. Viewing user order history

<u>Unimplemented features/Partially Completed:</u>

a. Water order tickets for ditch riders
b. Ditch capacity management for placing orders
c. Keep track of usage through credits and debits to the remaining balance
d. Order and usage reports
e. Administration reports
f. Summary views of days remaining in the irrigation season and days of water left/net amount remaining
g. A lease portal with agreement forms for both parties
h. Track run days or duration of the order from the user's order history
i. Admin/User order editing
j. Most admin functionality

XI. Future Work

There is still plenty of work to be done on the system we set up for our clients to have a fully functioning product. Most future development will center around implementing a front end interface to make use of existing backend database query functions. Their company does not have a GitHub or PlanetScale setup currently, meaning that future teams may need to set these up for themselves, or Walker Water will need to set these up beforehand. In the future, the team will need to handle deployment of the site. Next.js recommends using Vercel to deploy, although future research may need to be done in the future on what is best for it.

Resources Required:

a. No hardware requirements
b. Software requirements:
    i. npm (Link: npm)
    ii. yarn (Link: Yarn)

Knowledge & Skills Required:

a. Typescript/Javascript
b. Next.js/Node.js
c. React & MUI component library
d. Prisma Querying
    i. MySQL knowledge
e. Site Deployment

[List of dependencies are found in ./web-app/yarn.lock]

XII. Lessons Learned

Although we may not have gotten as much done as we had hoped, we as a team learned a great deal from this experience. Throughout the 5 weeks, our client was mostly unsure about how to implement the water ordering system, and what exactly they wanted. This left the project very open-ended and daunting to most of us. Also, most of us had never tried our hand at web development in the past. Questions about what technologies to use and how to use them took up just about the entire first week of the course. Half of the time working on this project was spent reading documentation and trying repeatedly to implement software none of us had touched before. This period of trial and error was an invaluable learning experience for each one of us with zero handholding.

Another lesson we learned was project consulting. As our client obviously had little experience with modern software and web development, we had to guide the client on what functional requirements were possible and/or necessary and the work needed to develop such features. This made defining the scope of the project difficult. Small or seemingly easy functional requirements would turn into complex multi-part features that could take weeks. For example, the database, which initially seemed like a cookie cutter schema, ended up needing several additions as our client realized they needed different data to be stored. This caused managing the database to take up the bulk of our development time. Additionally, our client requested us to guide them on the design of the user interface to appeal to the modern generation.

As the list of functional requirements grew, we realized we couldn’t ship a complete working product. As such, we would need to make sure that the code base was easily manageable for future development. Clear documentation was crucial for the final steps of development. Having a detailed readme as well as an in-depth file descriptions file in the root directory was mandatory to explain every single page and file as well as how they fit into the final product at large. The documentation process provided the team with knowledge on how to write easy and manageable documentation for other developers.

XIII. Acknowledgements

We would like to thank our clients at Walker Water, John Walker and Denise Jackson, for their time, and our advisor Amelia Read for her guidance.

XIV. Team Profile

Tyler Emken:

Engineering Discipline/undergraduate level: Computer Science / Senior

Hometown: Aurora, CO

Work Experience: Research intern at GolfTEC, Aurora Sports Park

Sports/Activities/Interests: Music and computers

Wallace Schageman:

Engineering Discipline/undergraduate level: Computer Science / Junior

Hometown: Houston, TX

Work Experience: Worked as an intern for Milestone Environmental

Sports/Activities/Interests: Reading, Music, and Computers

Phi Stanton:

Engineering Discipline/undergraduate level: Computer Science / Senior

Hometown: St. Louis, MO

Work Experience: Tutor at Kumon, and TA (Teaching Assistant) for CSCI 306

Sports/Activities/Interests: Drawing, Computers, and Skiing

Erik Swanson:

Engineering Discipline/undergraduate level: Computer Science / Junior

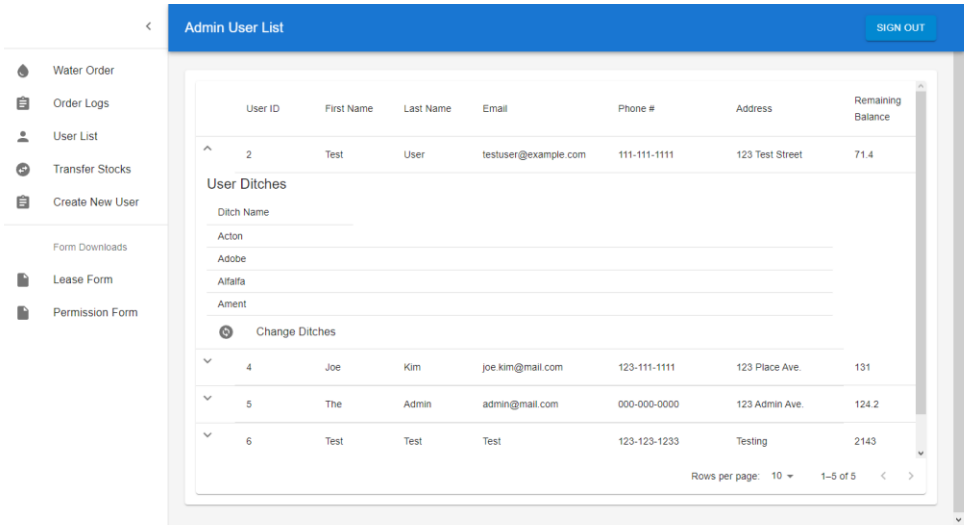
Hometown: Chicago, IL

Work Experience: Private Computer Science/Math tutor

Sports/Activities/Interests: Swimming, Video Games and Music

Appendix A – Admin Pages:

Figure 3a: Admin User List

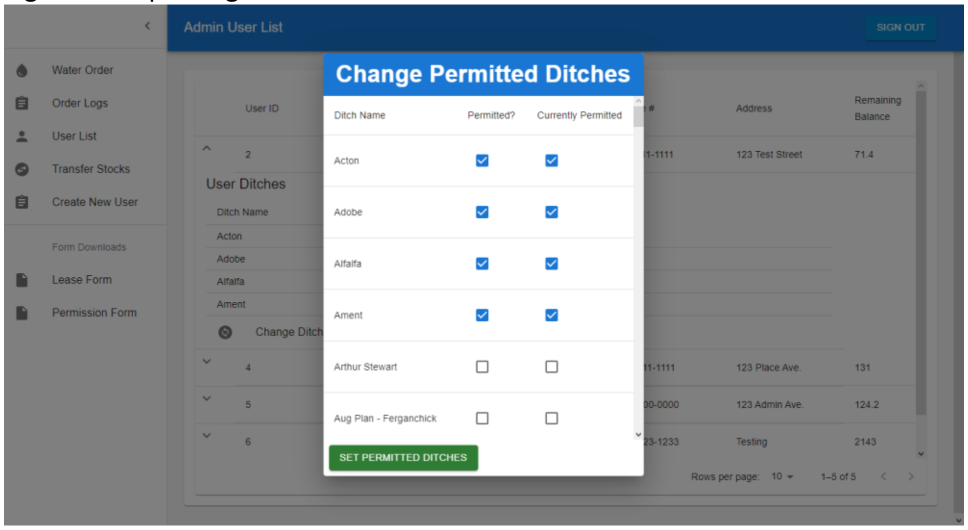


User ID	First Name	Last Name	Email	Phone #	Address	Remaining Balance
2	Test	User	testuser@example.com	111-111-1111	123 Test Street	71.4
4	Joe	Kim	joe.kim@mail.com	123-111-1111	123 Place Ave.	131
5	The	Admin	admin@mail.com	000-000-0000	123 Admin Ave.	124.2
6	Test	Test	Test	123-123-1233	Testing	2143

User Ditches

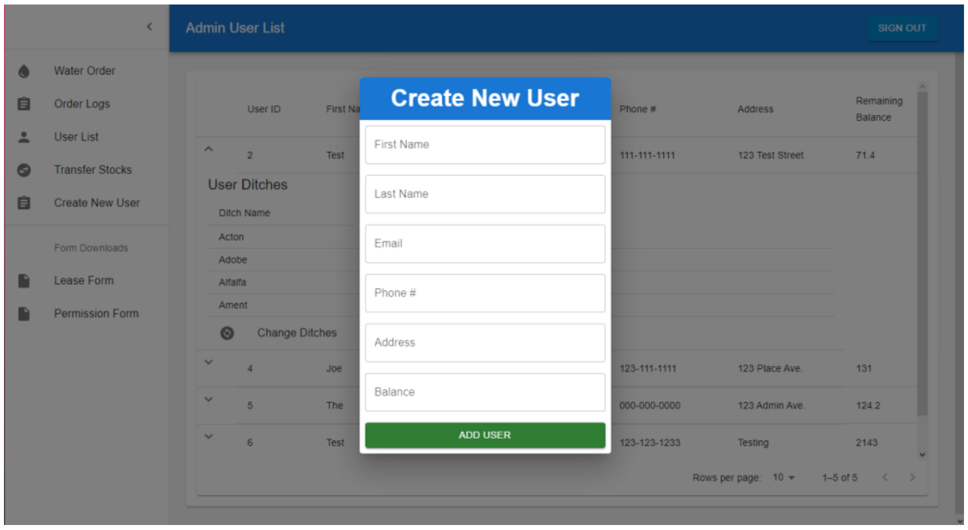
Ditch Name
Acton
Adobe
Alfalfa
Ament

Figure 3b: Updating User's Permitted Ditches



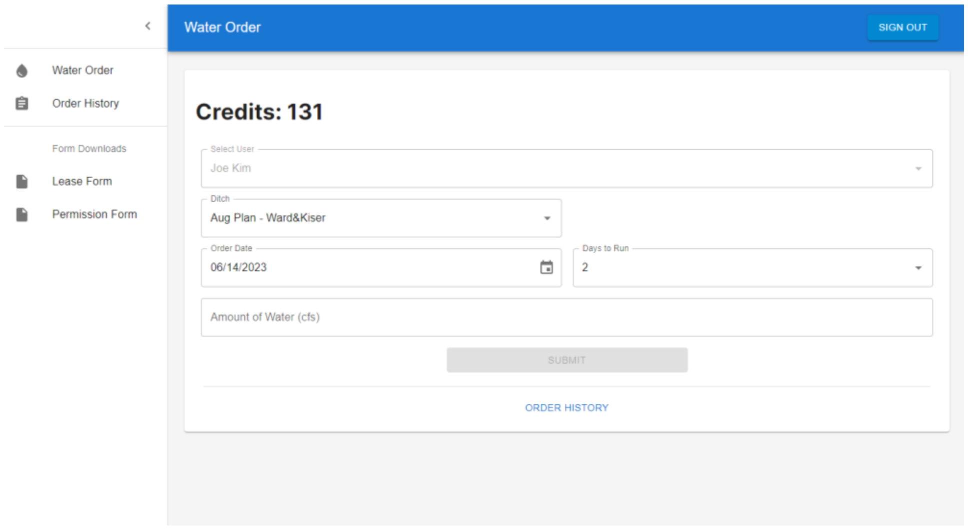
Ditch Name	Permitted?	Currently Permitted
Acton	✓	✓
Adobe	✓	✓
Alfalfa	✓	✓
Ament	✓	✓
Arthur Stewart	☐	☐
Aug Plan - Ferganchick	☐	☐

Figure 4: Create New User Popout



Appendix B – User Pages & Sign in/Sign out:

Figure 1: User Water Order



Credits: 131

Select User: Joe Kim

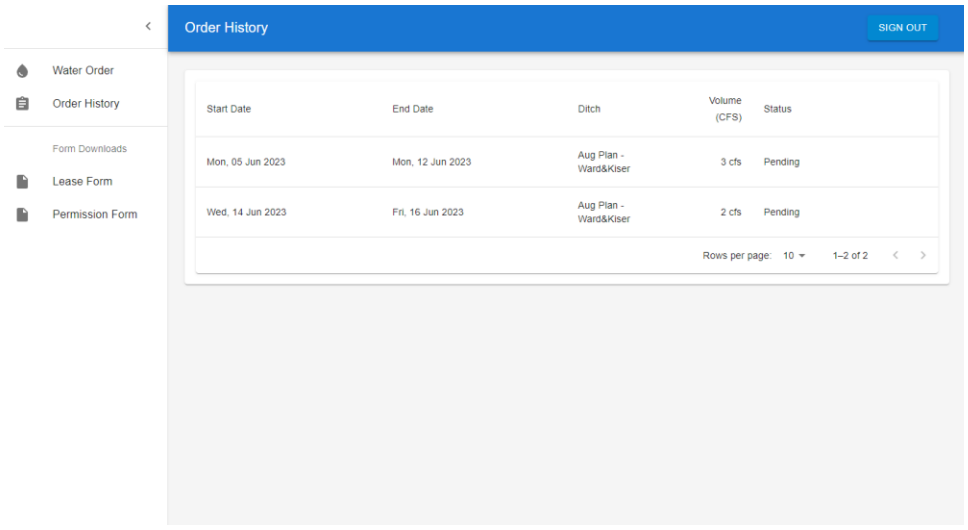
Ditch: Aug Plan - Ward&Kiser

Order Date: 06/14/2023

Days to Run: 2

Amount of Water (cfs)

Figure 2: User Order History



Start Date	End Date	Ditch	Volume (CFS)	Status
Mon, 05 Jun 2023	Mon, 12 Jun 2023	Aug Plan - Ward&Kiser	3 cfs	Pending
Wed, 14 Jun 2023	Fri, 16 Jun 2023	Aug Plan - Ward&Kiser	2 cfs	Pending

Rows per page: 10 1–2 of 2

Figure 3: Sign Out

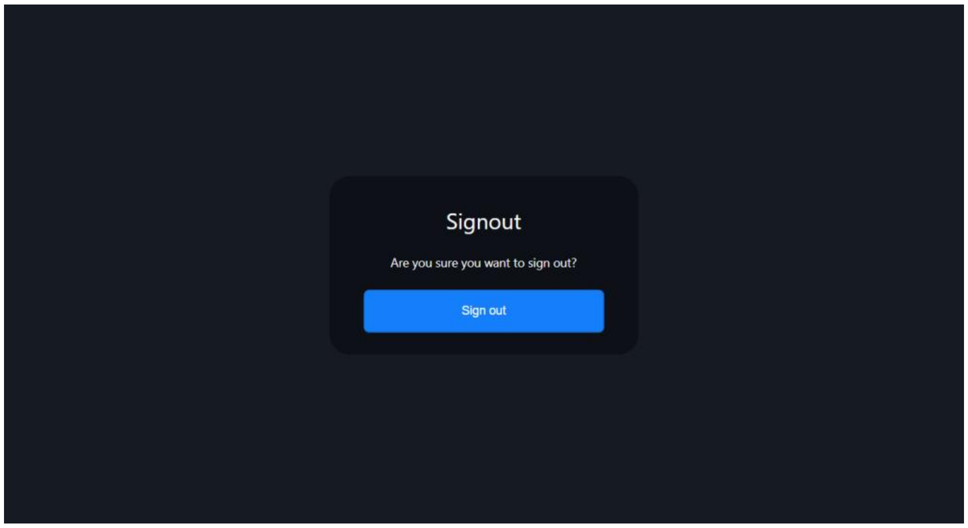
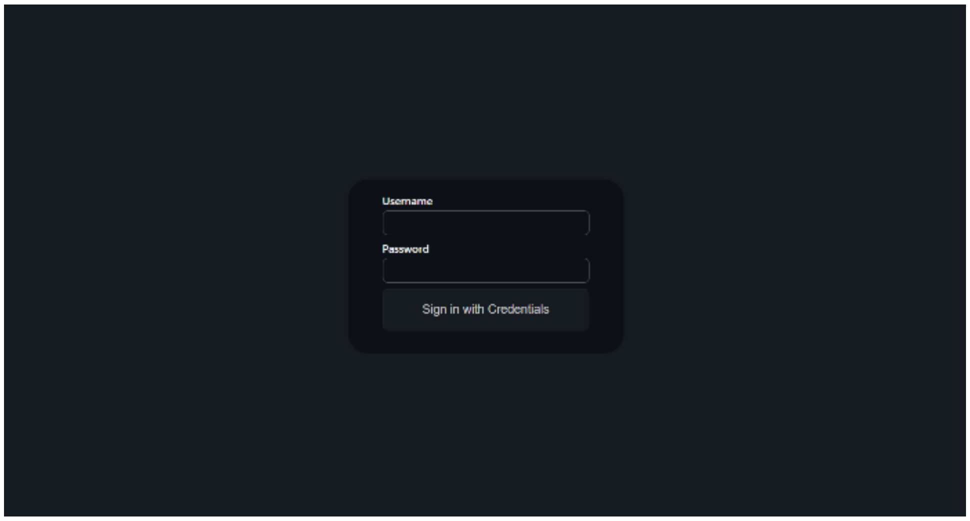


Figure 4: Sign In


Username
Password
Sign in with Credentials