



---

# **WWT & Rocky Mountain Labrador Rescue**

---

**CSCI 370 Advanced Software Engineering**  
December 8th, 2020

Dr. Wendy Fischer

Elise Renwick | Lucy Mahorner | Meg Zimmerman  
Jennifer Kuchta | Lindsey Kartvedt

## Introduction

Our client for this project was the Rocky Mountain Lab Rescue (RMLR). They are a non-profit rescue group that is run entirely by a small group of dedicated volunteers. In order to help RMLR get the full solution they need, we worked with a team from World Wide Technology (WWT). WWT is a large consulting firm that produces custom software for other companies. They support database solutions, web APIs, and a variety of other technologies that they provide as a contractor. We are working with a team of developers that include coders, UI designers, QAs, and a scrum leader. Throughout the course of this project, we have worked hand-in-hand with the WWT team in order to understand how they start a project from scratch. The only part of the project that was provided to us was the idea; the planning and execution were left up to us and the WWT team to decide at the beginning of the semester. We were able to attend client meetings, write stories throughout the whole process, design a solution, and then begin implementing said design. The solution that we designed for RMLR is a web-to-database solution to streamline RMLR's existing management system for adoption applications, volunteer applications, dog profiles, and foster applications. This solution utilizes database technology to migrate new and archived records from a spreadsheet to a database storage model. Our priority has been to implement this database system for adoption applications, although the product we have produced could be easily expanded to accommodate for these other categories as well.

## Requirements

### Functional Requirements:

1. A real database - NOT similar to Google Spreadsheets, as this is their current disliked method (we used Digital Ocean's DB platform)
2. Data is entered into the base automatically from applicant forms
3. Database allows only privileged individuals to view certain information, as some info is private
4. Database must be searchable and dynamic

### Stretch Goals:

1. Email bot that takes into account an applicants information from the database and personalizes the email
2. Email is edited and requires human approval before going out

### Non-functional Requirements:

1. Solution must fit into the already existing processes without hindrance
2. Solution must be maintainable for RMLR
3. Solution must be written in the given tech stack from WWT (listed in the System Architecture portion)
4. Database must be secure to protect sensitive data from applicants for both adoption and fostering

### Risks:

1. Security issues associated with setting up a database for RMLR, which could potentially leak client information (addresses, phone numbers, etc.)
  - a. To mitigate this risk, we decided to not put real information in the database until later, which allowed us to work with mock data, safely.
  - b. Permission tokens will also be addressed and properly used as the WWT team fully takes over the project at the end of the semester
2. A student breaching the NDA in a presentation, or even conversation with faculty discussing the scope of the project

- a. We had our faculty, Dr. Fisher, sign an NDA to avoid the risk of sharing proprietary information with non-privileged parties

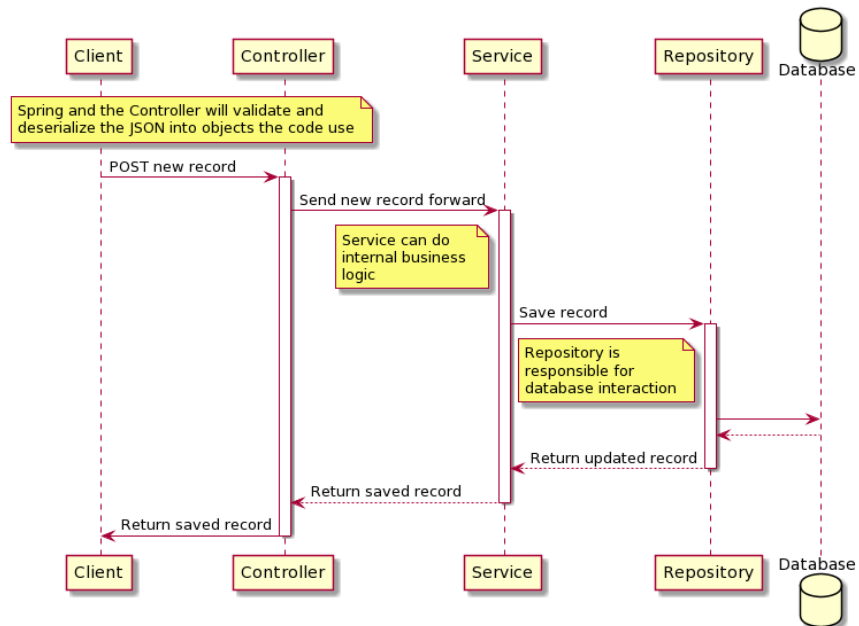
**Definition of Done:**

1. The portion of the solution in question has been fully tested.
2. Code reviews have been completed on the portion.
3. Client has seen the portion of the solution and has no further requests for change or improvement.
4. Project Handoff: WWT will continue working on this project after our class ends, so the final product will still be in development and the DB may not be fully deployed to RMLR
  - a. As such, there is some room for flexibility on point 3 of the above. Full-time developers will be able to continue the work where we left off once our semester ends

**System Architecture**

**Project Overview:**

The Rocky Mountain Labrador Rescue has repeatedly mentioned that the giant and strictly manual Google Spreadsheet they use to manage foster and adoption applications is their biggest pain point. With the current adoption application consisting of upwards of 120 questions, manual entry for each application is unreasonably lengthy and human error prone. Therefore, we made the decision to tackle this problem first. We are addressing it by migrating the spreadsheet data into a database, creating a program to easily enter new applications into the database, and co-developing an web interface with World Wide Technology which will allow the Lab Rescue board members to view and enter data. Automatic data entry from the application will allow the board members and volunteers more time to work with the dogs and other more important aspects of the Rescue. This also ensures the data is secure since people are required to input address, names, and other sensitive information to complete the adoption process. Ideally, this database and web interface will connect to RMLR’s current website to minimize the back and forth and make migration even easier, especially for matintence past WWT’s involvement. As seen in Figure 1 below, a brief outline of the overall project has been constructed to outline the connections between our tasks.



**Figure 1: Overview of Structure**

At the beginning of the project, we had hoped to expand on the current design by addressing other pain points, such as the overhead that goes in their manual response system. This was not a feature that we were able to develop or plan throughout the course of the semester due to time constraints-- the WWT team will continue to develop this and many other features before the project is concluded.

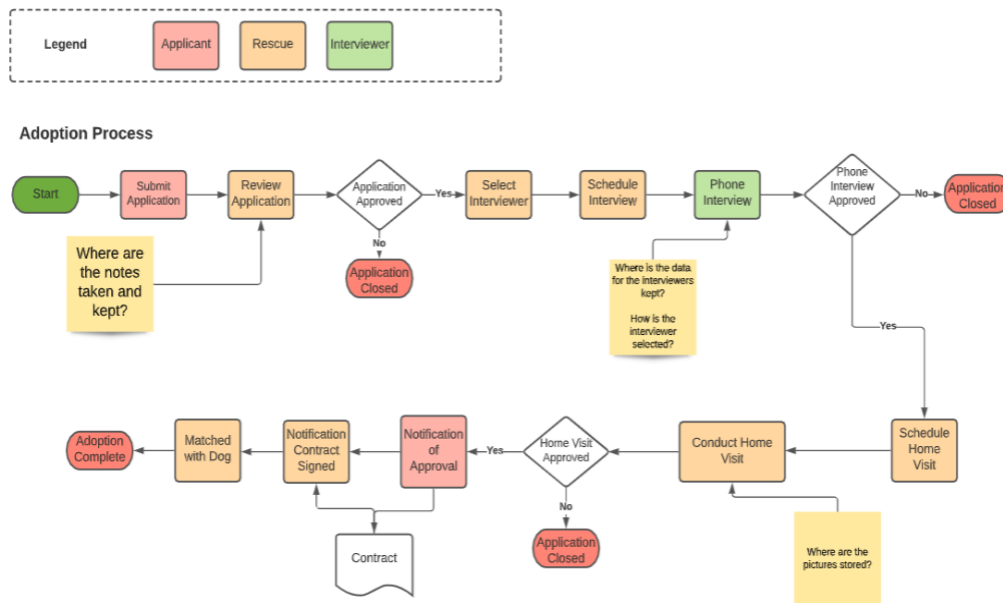
**Languages and Coding Environment:**

- Kotlin and Spring
- Javascript
- React
- VS Code
- Digital Ocean
- Git

WWT made the majority of the decisions about languages and coding environments given their experience in web development. The back end we are using is a Kotlin and Spring framework, and the front end is React which uses Javascript to talk to the back end and receive JSON responses. The team started on the local database service Docker, but is currently using Digital Ocean to store data. The specific coding environment is flexible and each person uses whatever they are comfortable in, but VS code and IntelliJ are the two main environments. Version control is done through Git.

**Current Processes:**

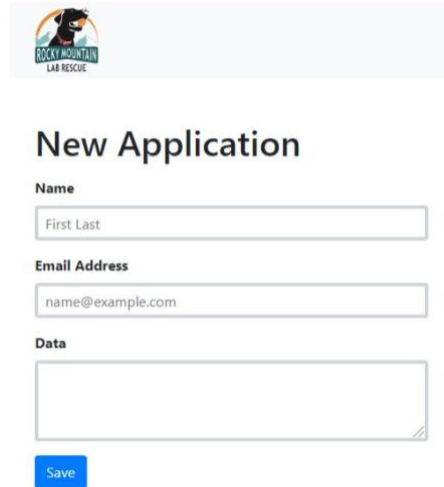
Figure 2 below outlines the process the lab rescue follows when an application is entered into their website. There are many complicated steps that require a lot of overhead work from the RMLR board members. We do not plan to automate the process entirely, but we want to address the portions that can be. For instance, currently once an application is submitted, it is emailed to the board members and they manually enter the data into the spreadsheet. This is time-consuming and error prone, but the board members want to retain the personal aspect of manual review. The solution will likely be automatic data population so that copy/paste and typing is eliminated from email application to database, but board members must manually review before the application data is officially added.



**Figure 2: Adoption Process Flowchart**

## Technical Design

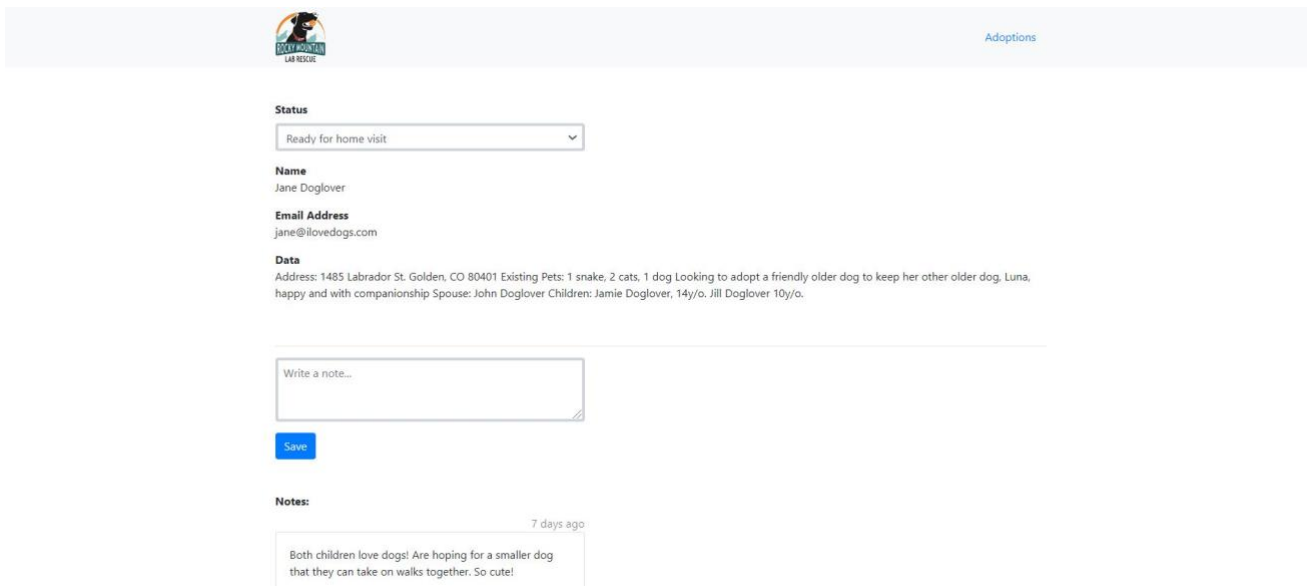
Project development will continue past this semester, so the “final project design” is limited to what the team accomplished up until this point. Figure 3 below depicts the current page where future dog adopters will file their applications.



The screenshot shows the 'New Application' form for Rocky Mountain Lab Rescue. At the top left is the organization's logo. The form title is 'New Application'. It contains three input fields: 'Name' with a placeholder 'First Last', 'Email Address' with a placeholder 'name@example.com', and a larger 'Data' field. A blue 'Save' button is located at the bottom of the form.

**Figure 3: Adoption Application Form**

Users will enter their name, email address for contact, and at this point any relevant data. In the future, this form will match the application that applicants currently fill out on the RMLR website where data is actually their address, pet owning history, signatures, lifestyle information, etc. Then users hit submit and their form is inserted into the database. RMLR board members and certified volunteers will be able to see the adoption by clicking on it from the list of all adoptions as shown in Figure 5, which takes them to the specific application information shown in Figure 4.



The screenshot shows the 'Adoptions' page for Rocky Mountain Lab Rescue. The page header includes the logo and a navigation link for 'Adoptions'. The main content area displays a single application with the following details:

- Status:** Ready for home visit (dropdown menu)
- Name:** Jane Doglover
- Email Address:** jane@ilovedogs.com
- Data:** Address: 1485 Labrador St. Golden, CO 80401 Existing Pets: 1 snake, 2 cats, 1 dog Looking to adopt a friendly older dog to keep her other older dog, Luna, happy and with companionship Spouse: John Doglover Children: Jamie Doglover, 14y/o. Jill Doglover 10y/o.

Below the application details is a text area for notes with a placeholder 'Write a note...' and a blue 'Save' button. A 'Notes:' section shows a note from 7 days ago: 'Both children love dogs! Are hoping for a smaller dog that they can take on walks together. So cute!'.

**Figure 4: Adoption Application - Filed**



# All Adoptions

#	Id	Name	Email Address	Data	Status
1	45c2bbd3-0ea1-4b75-aea6-5b1...	Neil Peterson	peterson@wwt.com	adfsdfsfgsf	NO STATUS
2	2997fd0e-b375-4e9c-9c9e-b414...	Neil Peterson	peterson@wwt.com	wdfgetgregsdf	NO STATUS
3	0586a872-afe3-4836-85c8-8570...	Neil Peterson	peterson@wwt.com	asdasdf	Home visit
4	74349f1a-c356-43af-a420-b269...	Neil Peterson	peterson@wwt.com	asdfsfsf	Application rejected
5	9340ea92-63b3-4f37-bee5-34dd...	Neil Peterson	peterson@wwt.com	sdfsfsf	Application complete

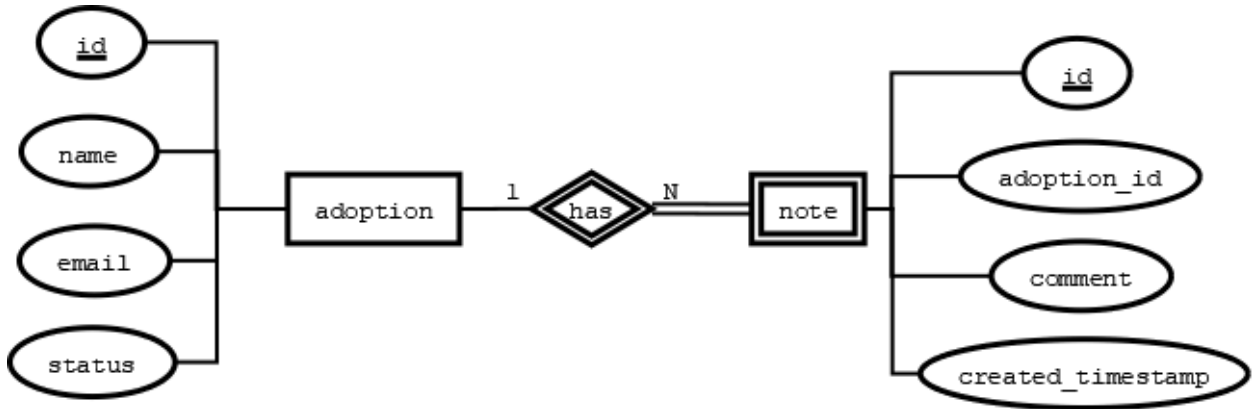
**Figure 5: Adoption Application List**

The key section “Status” shown at the top of Figure 4 is a drop down menu that allows board members to select the application's current status. They start as “New Application” but contain numerous other options including “Ready for Home Visit”, “Approved for Adoption”, and “Rejected” to name a few. Board members can also add unlimited notes to the application - a direct request from RMLR, because they often need to make reminders and small informational updates to the applications as they progress through the adoption process, and these notes need to be available for everyone to see and add to.

All fields include accessibility accommodation for users who may have impaired vision. Users can hit tab to navigate through the application form where embedded into the HTML will be descriptions of each field for auditory aid as well. When a user tabs or hovers over a field or application in the list it is highlighted and circled with a blue line to ensure that the correct field is selected, as RMLR does deal with sensitive information.

The website has a long way to go until it can begin being utilized by RMLR, but this is a great start with all of the major components of a database accounted for. The main database design is very simple with only two tables within it to store the data. RMLR being able to maintain our solution after the project ends is an essential part of our design and keeping the database paired down is a key part of that. There is the adoption table that stores all adoption instances created with all of the required information: a generated id, name, email address, and a general field for other relevant data. In addition to this table, there is a note table with a generated id, the id of the adoption instance that the note is for, the comment (i.e. the notes that RMLR board members add to each adoption), and the created timestamp for when the note is added. It is important to note that the relationship between the adoption instances and the notes is one to many as each adoption can have multiple saved and separate notes throughout the review process. The created\_timestamp column in the note table exists because of the need for multiple notes to be added at different times; this column is how the notes are sorted and displayed. The note’s adoption\_id column as pictured below is a foreign key from the id column in the adoption table. This also factors into the fact that the note table is a weak entity as without the existence of an adoption instance, there cannot be any notes. As the project progresses after the semester is done, the WWT team will likely expand upon the database and its tables in order to allow for a more robust storage of information throughout the adoption

process; however, the ERD and tables in figures 6 and 7 depict where the project is at the end of our involvement.



**Figure 6: Database ERD**

adoption

NAME	TYPE	OPTIONS
id	UUID	not null, unique
name	VARCHAR (255)	not null
email	VARCHAR (255)	not null
data	TEXT	
status	VARCHAR (255)	not null

**Primary key (id)**

note

NAME	TYPE	OPTIONS
id	UUID	not null, unique
adoption_id	UUID	not null
comment	TEXT	not null
created_timestamp	TIMESTAMP	with timezone, not null

**Primary key (id)**

**Foreign key (adoption\_id) references adoption (id)**

**Figure 7: Database Tables**

## Quality Assurance

### WWT's Unique QA Process:

Much like their approach to agile, World Wide Technology has a unique method of assuring quality products for their customers. They rely on Quality Advocates rather than quality assurance. The difference here is that a Quality Advocate has a much more active role on the team. Generally, they are involved in nearly every part of a project. They work with customers to write stories. They help determine the team's definition of ready or done. They work hand-in-hand with developers to ensure that test driven development (TDD) is being done thoroughly and at every step of the project. They always speak up and ask questions when they feel that it is essential for the quality of the product. These are just a few of the key roles that a Quality Advocate takes on within a project team in order to develop and deliver a high quality solution for their customer.

### Our Team's Quality Advocate:

Our team's Quality Advocate was Kayla Koger. Kayla follows all of the roles detailed in the last paragraph. She is present at nearly all of our meetings in order to ensure that our standard of quality is being met. This quality assurance is especially important for our project because the lab rescue is not technically savvy and they don't have the time or resources to fix any problems in our product. Now obviously Kayla's roles look different on this project than they do on her typical projects since the development team is mostly composed of us students. She has a couple of other non-standard tasks that she helps with so that we can work well with the WWT team. Kayla helps us learn different testing processes as educating team members on agile and TDD is one of a Quality Advocate's non-standard jobs on a team. So an example of this is that she showed us very simple load testing on the preliminary front end of our solution and taught us how to tell if the website broke under stress and where the break came from. Kayla also actively ensures that there is a very collaborative and open working environment on the team in order to foster a higher quality of work. She is always very vocal about making sure that we understand everything being done within the project so that we can deliver quality work.

### QA for Our Project:

Quality Assurance for our project extended beyond Kayla's job as well, QA has been built into our development process from the very start. For the website's backend we practiced Test Driven Development, or TDD, adhering to the Red Green Refactor method. This means we wrote the tests first and they are non-passing, which is why they are "red". Then we built in the code, allowing the tests to pass or become "green". Finally, we refactored the code as we continued to implement new stories. This method ensured that as we added new pieces to the website and database, no past functionality would be lost and we would maintain the same standard of quality. The frontend development is a little different than the backend because the website UI was not a fully formed idea when we began implementing front end stories. The process is usually functionality first and aesthetics second, so the overall design can vastly change the proposed functionality, especially within different browsers. Our overall goal was ending with test coverage at 80% or greater.

WWT also made frequent use of "mob-sessions" to ensure quality code was written. These mob-sessions occurred twice a week and were a mix of group and pair programming. We, the students, were matched up with different WWT developers who acted as mentors in the beginning of each session to help us program to their standard, which allowed us to really dig into the new programs and coding styles we were learning without fear of damaging the website. This also enabled us to work on more stories



ourselves outside of mob-sessions, and code was then checked before being pushed to the master branch, and all tests were run each time to make sure that nothing has broken.

Once we had a physical website and database working, we also could start to have regular demos with Rocky Mountain Lab Rescue so they stayed updated with our process, could provide feedback on the site, and could add new story concepts as we progressed through the implemented ones. Finally, our project's "Definition of Done" is a bit unique since the project does not end with us at the end of the semester. WWT will continue developing with RMLR through next year. For Field Session specifically, our definition of done was having a website where the adoption application could be filled out, added to the database, and then accessed by board members through the website. This functionality addressed RMLR's main pain points, so it was an excellent stopping point for us to assess and assure that our work with WWT was on par with their quality standards.

### **Ethics:**

If we were to pick one IEEE Principle that is in most danger of being violated, it would be the Client and Employer principle. This is because while we are working for WWT, we are creating this project for RMLR, and with the two different parties it is easy to have miscommunication. This could ultimately result in a demo for instance where the client is not happy with a certain attribute of the project. For instance we created a notes option for the volunteers to add notes to different applications, but in a demo the rescue may or may not like the feature that every member of the rescue can see and edit the existing notes. This is ultimately why we have these demos, so that we can hear from both the client and the employer and make little changes if there was miscommunication and prevent a situation where we may unknowingly not act with the client's best interest, while following our employer's instructions. Lucky for us, we do have full time employees from WWT to look out for holes in our plan and better prevent IEEE principles from being violated. However, if our software quality plan is not implemented properly, that's when we'd start to see some gaps in our product. For instance if we did not fully test a name field on an application to account for letters with accents, and then a customer is given some type of error for their name field. Another example would be an accessibility issue where we only designed adoption applications to be a website form, but some customers may only be able to fill out physical copies of the application, whether that be a medical issue regarding screen time or if someone simply doesn't have access to the internet.

### **Davis Questions:**

As mentioned in the section above, we have identified two ethical implications our project might have:

1. The name field does not account for accents.
2. Accessibility shortcomings.

Given these two implications we looked at two of the Michael Davis Questions:

1. Common Practice Test: What if everyone behaved this way?
2. Professional test: What does the Software Engineering code of ethics say about this problem?

Applying the common practice test to the first implication we identified, we are asking, "what would happen if every software developer that created a solution that required name input, did not account for accents?" This would lead to a big discrimination problem. Suddenly, every person with an accent in their name would find it difficult to use any online form. For some people we risk making them feel as though their identity is an inconvenience to society, which is an emotional toll we would never want our project to have. For accessibility we run into a similar problem with discrimination. Instead of discriminating against someone's name and identity, if every software developer did not make accessibility accommodations we would discriminate against people of different abilities and those

without access to a device. This would have a greater toll than just emotionally because these people would be cut off entirely.

With both of the identified potential implications, the professional test points out clear issues. For one, we would not be acting consistently with the public interest. We would be acting for part of the public, but we would not be including the interests of everyone our solution may serve. Secondly, our product would not be meeting the highest professional standards. With the name accent and the accessibility implications, our software would have failed at testing and not met our highest standard of quality.

## **Results**

### **Summary of Progress:**

We have made a web-based solution for the members of the lab rescue which allows for board members to enter adoption application data, view all applications, look at individual applications, update the status on individual applications, and add persisting notes to individual applications. The data we currently have for an application is id, name, email, data, and status. There are multiple web pages that allow for data viewing in a few possible ways.

### **Summary of Testing:**

Our testing consisted of unit tests on both the front end and back end of the application. For the back end we relied on mock objects to test responses from the API. For the front end we took a non functional testing approach. We entered mock data into our testing website and checked for breaks using the inspection tools provided in the browser. With the help of our QA we did further testing on the application as a whole. Before each task was complete we would examine the behaviour and do stress testing by refreshing the page repeatedly. QA tests included identifying where something could go wrong and forcing the process to fail so that we could confirm our expected results and understanding of the product.

### **Testing Results:**

Following TDD practice, whenever our code has been pushed to the final solution, all of our unit tests have passed. Since this is how most of our testing is done, we do not have a lot on the results themselves. Our client has recently been given access to an environment where they can test out our application. Since our product is still in the early stages of development, they have not brought up any key issues with the features that have been implemented.

### **Feedback From Client:**

Over the course of the semester we've had two demos with the board members at Rocky Mountain Lab Rescue. In both demos the rescue was extremely excited about the team's progress, as the website and database will improve the efficiency of the rescue's adoption process immensely. Minimizing the time spent with adoption applications allows the rescue volunteers to spend more time focusing on the dogs. During the first demo, the rescue was introduced to the basic functionality of the website and database and was extremely pleased with the design. They frequently mentioned how the secure database would prevent the rescue from losing their data, the main concern they communicated at the beginning of the project. In the most recent demo, the rescue volunteers expressed their concerns with not being able to see the timestamp of the volunteers' notes. They were also concerned that the adoption application

(similar to a Google Form) was not able to be edited. These concerns are relatively easy fixes and brought out the true purpose of a demo: demonstrating the product to the client periodically so that they are completely satisfied with the final product.

### **Features implemented:**

1. Created an adoption application record with id, name, email, data, and status.
2. Connected the backend to a DigitalOcean database.
3. Multiple web page views for records (all records, individual)
4. Made the status of each adoption updateable
5. Made a notes section on individual adoptions for stakeholders to save important notes on each

### **Features not Implemented/Future Work:**

1. Authentication: Before WWT hands off the final product to the lab rescue they have expressed that having a form to authenticate and authorize users is essential. The information that the lab rescue has about applicants needs to be kept confidential. As a team, we might not play a part in this work since our semester is coming to an end.
2. Searching adoption record by name or email: This is another high priority task that the lab rescue has expressed a need for. This feature exists with google sheets so in order for our product to be an improvement for them, we need this feature. Once again, this might be a feature that WWT implements after we hand off what we have completed.

### **Lessons Learned:**

1. Code reviews are a very uplifting experience and each person should go into a session with the mindset that everyone completed their best work given the knowledge and experience they had at the time.
2. Quality Assurance should be practiced by all members and considered during all processes of the development of a product.
3. As a team we learned the general structure of web development, how the pieces fit together, and what programming languages can be used.
4. Kotlin was a new language to almost every team member, so we all have another language in our set of skills now.
5. Public speaking in a mask is difficult.
6. Unit tests almost never pass the first time after the code for them has been completed which highlights the importance of TDD.
7. Not all companies share the same agile, QA, or testing processes, so there may be a learning curve.
8. Having clearly written, testable stories keeps all members of the team on the same page so that the project can flow smoothly.
9. Pair programming can be very beneficial, especially when different members of the team have different strengths and weaknesses.
10. Constructing a database schema from scratch is a lot of logical thinking to prevent logical changes later on, but you can change the schema (relatively) easily since we only have mock data in development.

## Appendices

### Development Setup Instructions

After WWT finishes the project and fully hands it off to RMLR, they will likely want to continue the upkeep and expand/make changes to the website and database with their own volunteers and workers so it's important to ensure that this is easily done.

#### About

Client side of the pet adoption and foster application administration

This project was bootstrapped with Create React App (<https://github.com/facebook/create-react-app>).

- Dev URL: <https://rmlr-dev-i2qmm.ondigitalocean.app/>

- Prod URL: `_TBD_`

### How to Run the Application and Run the Tests

**Run the client side** - in a terminal at the correct location

```
yarn start
```

If the client side has started up correctly you should see the following

```
Compiled successfully!
```

```
You can now view client in the browser.
```

```
  Local:           http://localhost:3000
```

```
  On Your Network: http://192.168.1.172:3000
```

```
Note that the development build is not optimized.
```

```
To create a production build, use yarn build.
```

and a browser tab or window should automatically open to <http://localhost:3000>

\*Please note the server will run continuously until Ctrl + C is entered into the terminal\*

#### Run the tests on the client side

```
yarn test
```

### Development Setup

#### Tools & Applications

\* [Homebrew](<https://brew.sh/>) \*(Mac Only)\*

i. Check if [Homebrew](<https://brew.sh/>) is installed  
`which brew`

ii. Install [Homebrew](<https://brew.sh/>) if not already installed  
`/usr/bin/ruby -e "$(curl -fsSL`

```
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

iii. Update [Homebrew](<https://brew.sh/>).  
`brew update`

iv. Add the casks tap.  
`brew tap homebrew/cask-versions`

\* [Chocolatey](<https://chocolatey.org>) \*(Windows Only)\*

\* [Visual Studio Code](https://code.visualstudio.com)

\* On Mac

```
brew cask install visual-studio-code
```

\* On Windows \*(unconfirmed)\*

```
choco install vscode
```

## **Suggested Setup for Client Side Development (Mac OS X)**

i. Install [NVM](https://github.com/nvm-sh/nvm) (Node Version Manager)

```
brew update  
brew install nvm
```

ii. Add following line to your profile. (.profile or .zshrc or .zprofile or .bashrc)

```
export NVM_DIR=~/.nvm  
source $(brew --prefix nvm)/nvm.sh
```

iii. Source your profile

```
source ~/.profile
```

iv. Install [Node.js](http://nodejs.org/) (4.0.0 or later)

```
nvm install node
```

v. Clone this repository: <https://docs.github.com/en/free-pro-team@latest/github/authenticating-to-github/connecting-to-github-with-ssh>

```
git clone git@github.com:RMLR-CO/project.git
```

vi. Install dependencies

```
yarn install
```

vii. Serve the presentation and monitor source files for changes

```
yarn start
```

viii. Open <<http://localhost:3000>> to view the website

Available Scripts

In the project directory, you can run:

### **yarn start**

Runs the app in the development mode.

Open <http://localhost:3000> to view it in the browser.

The page will reload if you make edits.

You will also see any lint errors in the console.

### **yarn test**

Launches the test runner in the interactive watch mode.

### **yarn build**

Builds the app for production to the `build` folder. It correctly bundles React in production mode and optimizes the build for the best performance. The build is minified and the filenames include the hashes. Your app is ready to be deployed!

### **yarn eject**

\*\*Note: this is a one-way operation. Once you `eject`, you can't go back!\*\*

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This command will remove the single build dependency from your project. Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESLint, etc) right into your project so you have full control over them. All of the commands except `eject` will still work, but they will point to the copied scripts so you can tweak them. At this point you're on your own. You don't have to ever use `eject`. The curated feature set is suitable for small and middle deployments, and you shouldn't feel obligated to use this feature. However we understand that this tool wouldn't be useful if you couldn't customize it when you are ready for it.

### **Learn More**

You can learn more at <https://facebook.github.io/create-react-app/docs/getting-started>  
To learn React, check out the <https://reactjs.org/>

**Code Splitting** - <https://facebook.github.io/create-react-app/docs/code-splitting>

**Analyzing the Bundle Size** - <https://facebook.github.io/create-react-app/docs/analyzing-the-bundle-size>

**Making a Progressive Web App** - <https://facebook.github.io/create-react-app/docs/making-a-progressive-web>

**Advanced Configuration** - <https://facebook.github.io/create-react-app/docs/advanced-configuration>

**Deployment** - <https://facebook.github.io/create-react-app/docs/deployment>

**`yarn build` fails to minify** - <https://facebook.github.io/create-react-app/#npm-run-build-fails-to-minify>