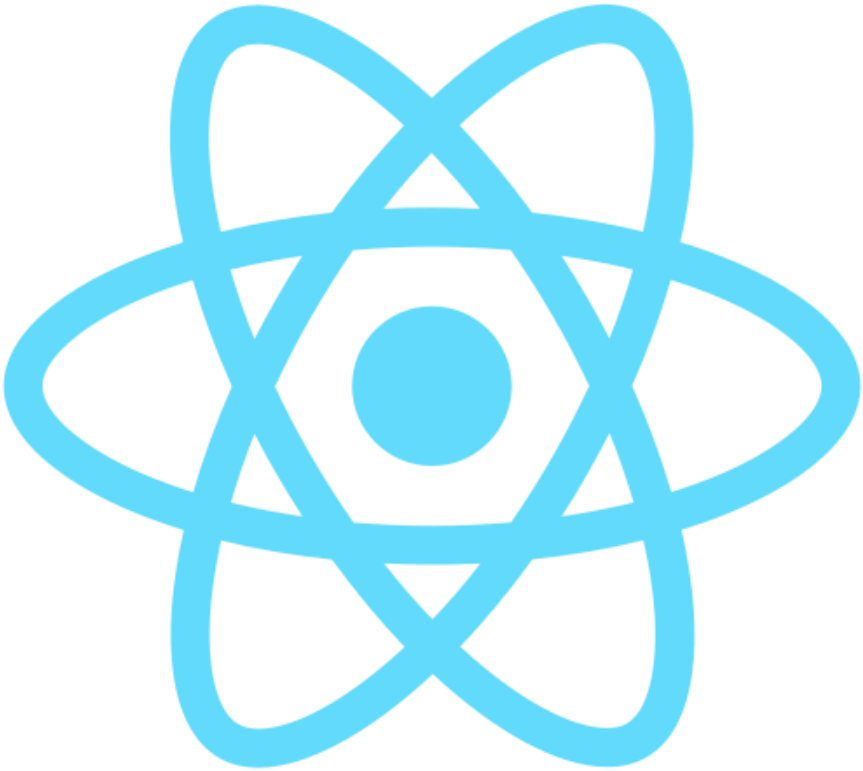
# Code Clapper

FINAL REPORT



**Contractor:** Dakota Moore - *Software Engineer*
**Cliet:** Craig Stansbury

DakotaMooreSE@gmail.com
https://github.com/DAMooreSE/codeClapper

Revised: October 6th, 2022

# Executive Summary

## Engineering Standards and Design Practices

- Agile Software Development
- Continuous Integration/Continuous Development

## Summary of Requirements

- Recording Service
- API
- User Interface

## Skills and Knowledge acquired

- Knowledge of Firebase and hosting applications using cloud services
- Knowledge of Developing applications on MacOS

# Table of Contents

# 1. Introduction

## 1.1. Acknowledgement

I would like to acknowledge Hendrik Swanepoel for developing the original application that this project is built off of and providing the source code for use in this project, as well as Craig Stansbury for being the client to this project.

## 1.2. Problem and Project Statement

### 1.2.1. Problem Statement

When creating pre-recorded lessons it's possible that a single clip requires multiple takes due to errors made when recording. This results in long post-production times due to having to remove the clips with errors and/or having to find the correct take of a particular clip.

### 1.2.2. Proposed Solution

The purpose of this project is to develop a recording application solution to solve the problem statement. The project is driven by the need described in the above problem statement, to reduce the amount of time required for post-production of a recorded lesson. To help reduce the amount of time spent editing, the recording application will include additional functionality that allows a user to restart a recording if an error is made, deleting what's been recorded and eliminating the need to edit out the mistake later in post-production.

## 1.3. Operational Environment

The operational environment for this project is a local device running MacOS that will store the project and all files produced by the application.

## 1.4. Functional Requirements

### 1.4.1. API

- The API shall expose the recording functions to requests from the User Interface.

### 1.4.2. User Interface

- The User Interface shall be rendered in a modern web browser.
- The User Interface shall remain functional when the API is down.

#### 1.4.3. Recording Service

- The recording service shall be able to detect any connected video and audio device and allow the user to select which ones to use.

- The recording service shall be able to record video captured on the device selected by the user.

- The recording service shall be able to record audio captured on the device selected by the user.

#### 1.4.4. Operational Environment

- The recording service shall run on MacOS-based machines.

- The User Interface shall be run on a MacOS-based machine and accessible from devices that are not running on MacOS.
    - The User Interface should be interacted with using a modern web browser. Safari and Chrome will be the supported browsers.

### 1.5. Non-functional Requirements

#### 1.5.1. API

- The API shall execute the recorder functions in under 10 seconds.

#### 1.5.2. User Interface

- If the Recording Service is not available, the User Interface shall display loss of functionality to the user.

#### 1.5.3. Recording Service

- The Recording Service should be able to run in a MacOS-based system via local deployment using Firebase.

#### 1.5.4. Maintainability

- The documentation made available through the project's website is descriptive enough for the client or any future developers to understand how to modify the project properly when needed.

- When the Client wishes to upgrade their Operating System, the project should maintain all functionality.

- When the client wishes to modify the project, the project shall not be difficult to modify.

## 1.6. Intended Users and Uses

Code Clapper has one main type of user: a human user. The use case of this user consists of the following:

#### 1.6.1. Record a video that contains no audio.
#### 1.6.2. Record a video that contains one or more audio tracks.
#### 1.6.3. Export a file containing the recorded video (and audio).

These use cases may have overlapping interactions with the system.

## 1.7. Assumptions and Limitations

#### 1.7.1. Assumptions

- Hardware and operating system environments are made available by the client, and these resources are sufficient for the development.

- I am provided the source code for the original CodeClapper application that can be used in the project's development.

#### 1.7.2. Limitations

- This project has a budget provided by the client, thus, the resources and development time are constrained by a financial cap.

- If the source code provided for development was found to be insufficient, the project would be built from scratch possibly using

other software and resources, or the scope of the project would be modified to work with the existing resources.

## 1.8. Expected End Product and Deliverables

### 1.8.1. A Fully Functional Recording Service

There will be a recording service that's deployable on a computer running MacOS that's accessible on the local network. The recorder will be able to use any connected audio or video device to record from. The recorder will have the functionality to record video and audio separately and will allow a recording that's in progress to be restarted, saved, or canceled. It will record the video first and then allow the recording of one or more audio files that will be combined with the video when exported. When a video is exported, the recording service will also output a single video file containing everything that was recorded from the start of the video recording.

### 1.8.2. Web-based User Interface

The web-based user interface will contain web pages that will allow users to login, start a new recording session, record video and/or audio, and export saved recordings.

### 1.8.3. Project documentation

There will be some kind of "deployment manual" that lives within the project repository that describes how to build and deploy the recording service and Web-based User Interface. This manual will also include the documentation for creating a Firebase account and linking your build to your account, as well as documentation regarding the development environment setup and required dependencies. This is close to the same manual that can be seen in the appendix.

# 2. Specifications and Analysis

## 2.1. Proposed Design

The software solution consists of the following components: a User Interface, An API, and a recording service. The API receives requests from the User interface to view some data or issue a command to the recording service.

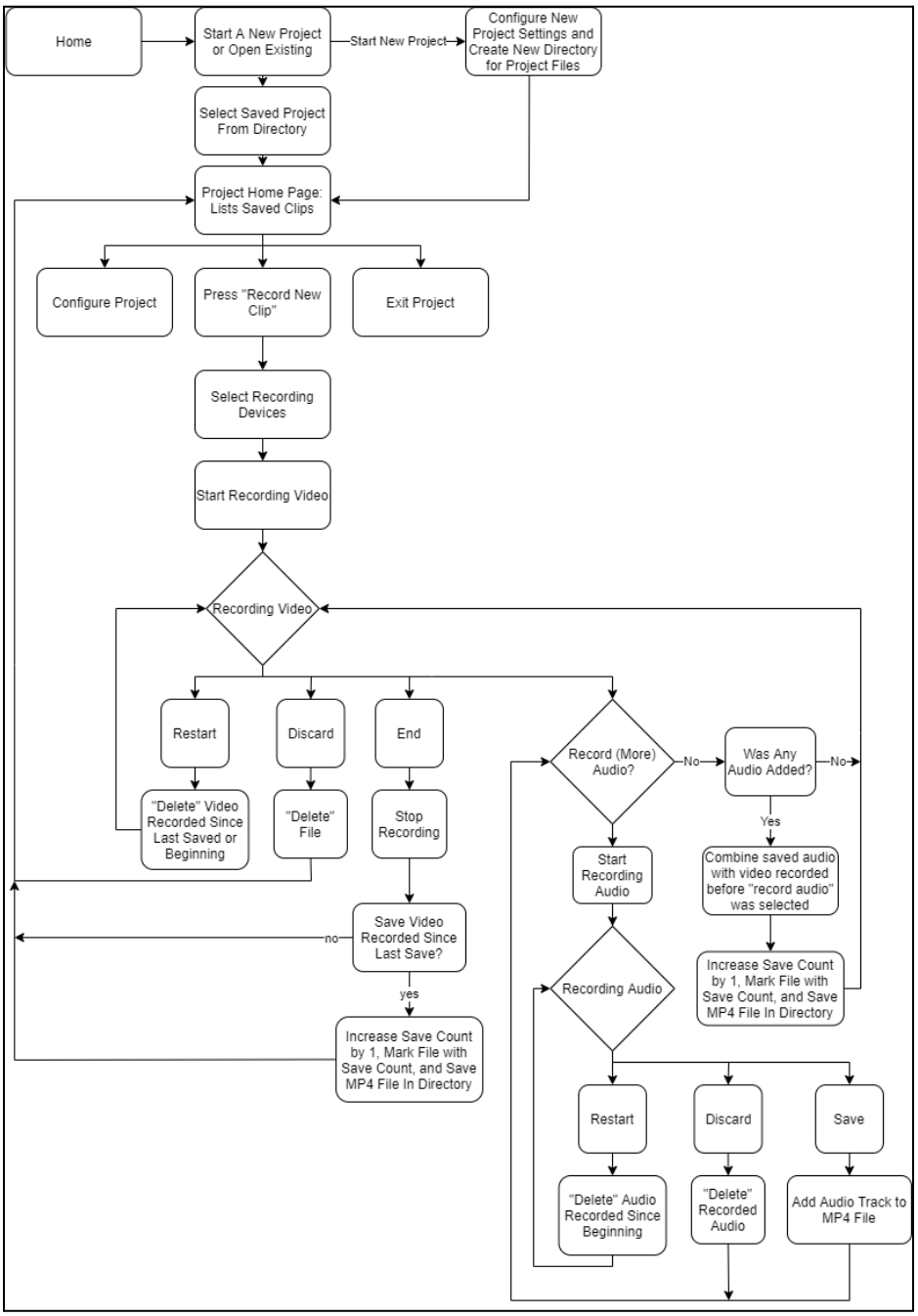



Figure 1: Flow chart diagram

#### 2.1.1. Recording Service

- The recording service will record video from a selected device and allow the user to restart the recording, cancel, or save it with the option to record audio afterwards before exporting.
- The recording service will record audio from a selected device when a user selects to record audio for a saved video clip and allow the user to restart the recording, cancel, save and record additional clips, or save and export a file that combines the video recording with the audio recording(s).
- The recording service will save all of the files created during a session in such a way that a user is able to easily navigate and manage all of the files for a project.

#### 2.1.2. User Interface

- The user interface will be accessed by logging into an account that's created using an email and password.
- The user interface will be accessible by any device that's able to use a web browser to access web pages and that's connected to the same local network as the machine that's hosting the application.
- The user interface will allow the user to access the functionality of the recording service.

## 2.2. Development Process

This project used an agile approach to development that consisted of three to four week sprints, where the client and I review, tests, and/or discusses changes and progress made during that sprint as well as discuss what problems and/or tasks should be prioritized during the next sprint. For all merge requests, there were tests that accompanied all new work introduced into the stable branch. All merge requests were required to pass all component and system-level tests, and be reviewed and tested by the client as well.

## 2.3. Design Plan

# 3. Statement of Work

## 3.1. Technology Considerations

The original source code provided as a framework for the project used many deprecated or outdated versions of many softwares. Because of this, a decent amount of time was dedicated to determine if it would be viable to build the application using the source code by updating or implementing different software, or if it would require less time and/or work to design and build the software from scratch using other various technologies. It was determined that a lot of time and work could potentially be saved by trying to revise the old source code.

## 3.2. Project Tracking Procedures

I broke the project up into smaller sets of tasks and then used Trello to track the progress throughout the project. Using Trello, the client would be able to track the progress of the project and see what I was currently working on and what was planned to be worked on next and take that into consideration during sprint reviews.

# 4. Testing and Implementation

## 4.1. Process of Testing

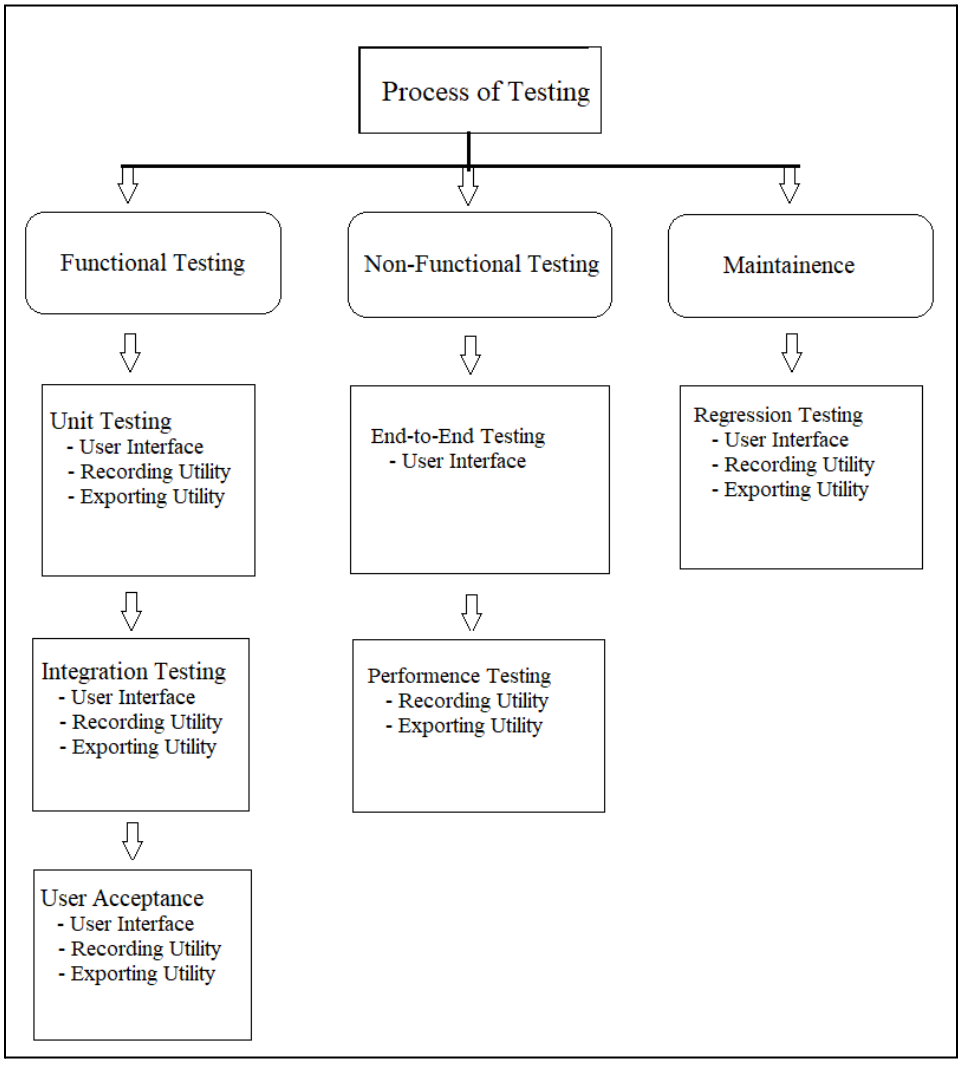



Figure 2: Process of testing

Figure 2 displays an overview of the process of testing. I followed each flow as I progressed forward in the development of the project. For each flow, I implemented the testing in the first block before moving on to the next and so on. This allowed me to break down testing into concise parts to know what needs to be accomplished for testing.

## 4.2. Hardware and Software

### 4.2.1. API

### 4.2.2. Recording Service

### 4.2.3. User Interface

## 4.3. Functional Testing

### 4.3.1. Unit Testing

### 4.3.2. Integration Testing

### 4.3.3. System (end-to-end) Testing

## 4.4. Non-functional Testing

### 4.4.1. API

### 4.4.2. Recording Service

### 4.4.3. User Interface

## 4.5. Results

# 5. Closing Material

## 5.1. Conclusion

Code Clapper has been continuously used by the client to aid in the creation of products of sufficient quality using the tools and functionality provided to them in the software application.

## 5.2. Resources

React.JS: https://reactjs.org/docs/getting-started.html
React-DOM: https://reactjs.org/docs/react-dom.html
React-App-Rewired: https://www.npmjs.com/package/react-app-rewired
Babel: https://babeljs.io/docs/en/
Aperture: https://www.npmjs.com/package/aperture
Firebase: https://firebase.google.com/docs
jQuery: https://api.jquery.com/

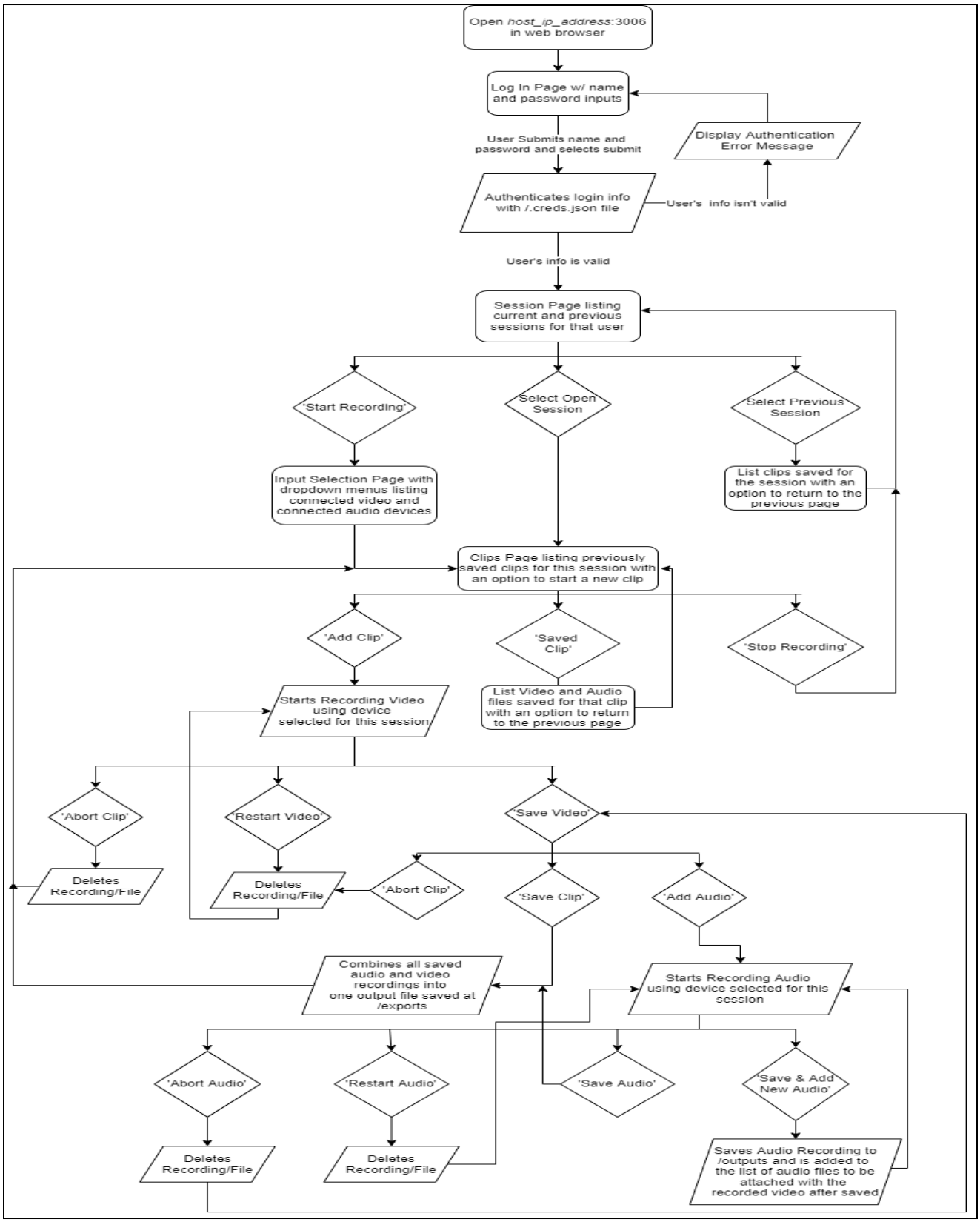
# 6. Appendices

## 6.1. Operational Manual

The current operation manual for Code Clapper can be found in the projects Google Drive Folder, specifically at this link:

Code Clapper Deployment Guide

## 6.2. Previous Design Versions



## 6.3. Project Code

All of the source code for the project can be found in the project's git repository found at this link here:

https://github.com/DAMooreSE/codeClapper

This repository contains all the source code provided by the client originally, all previous test and stable versions of Code Clapper, and all documentation regarding the early development stages of the project.