483 Project Proposal - onWater

Ethan Skelton
Caitlynn Koback

Introduction

OnWater is a fly fishing app that allows users to store and access information regarding local fishing spots, and it also allows users to document their catches through photos. Many fishermen enjoy knowing the size of their fish. They take pride in the size of their catch and can compete amongst each other. OnWater wishes to provide a feature that allows the user to measure the fish without the use of outside tools. The OnWater team has attempted to create this feature in the past, using Augmented Reality (AR). Measuring through AR relies on multiple variables, but two main requirements are a 2D plane and consistent features. As such, this makes it difficult for AR-based measuring to work properly in a moving environment, such as a river, and on a wriggling subject -the fish. These are the exact complications the OnWater team was attempting to resolve. Research shows that on average, an eye grows much slower in comparison to a fish's body. As such, there is less variation between eye sizes as opposed to the body. This implies that it is possible to use the relative eye:body ratio to calculate the total length of the fish. To take advantage of this, we will be implementing a feature that allows the user to place points on a picture of the fish in order to calculate the pixel lengths of the eye and body, in which an algorithm will calculate the best estimate of the fish's length. Implementing a feature for this will be beneficial to all users of the app, as it removes the guesswork of describing the size of a fish.

Qualifications

Resumes

[See appendix A]

Project Background

Measuring Technique (Head:Eye Ratio)

Through a bit of research we found that in order to accurately measure a fish in a picture we would need some reference point that we know the length of. This point will allow us to calculate the length of a pixel in inches or centimeters, so that we can determine the length of the fish in the picture.

We came across the journal by Richardson that talked about using the Relative eye size of a fish in order to estimate the length of the fish. The article went into how through different species “the head height:eye diameter ratio was correlated more strongly with body length than was eye diameter alone”(1). This made us start researching to see if this was the same on freshwater fish since the article was done on saltwater fish.

We are going to talk with a biologist who specializes in fish who will hopefully help us come up with a plan for measuring the fish.

We took pictures of fish that we already knew the length of and then tried to see if we could estimate the length of the fish. In order to measure the fish we used sketch and other similar programs to measure the amount of pixels each of the measurements we wanted to observe are [See appendix B.3]. Using these programs allowed us to see if our methodology was going to work or not. We started by just measuring the diameter of the eye and this gave us some interesting results since we were assuming the diameter of the eye to be a specific value. Then we used the ratio of head height to eye diameter to estimate the length of the fish. We are still not sure if this ratio will be the solution to measuring the fish so if we are unable to come up with a solution with the ratio we will go down more of a machine learning path. Eventually lines will be placed on the picture to calculate the length, either by the user or through machine learning [See appendix B.2 to see the lines in picture 2].

Types of Fish

The types of fish that we are dealing with will also determine how we are able to measure the fish's length. Different species grow at different rates and in the article about the head:eye ratio of fish they say, “5 of the 6 fish species we examined the head height: eye diameter ratio was correlated more strongly with body length than was eye diameter alone“(1). This means that we will also have to determine the type of fish that is in the picture in order to accurately determine the length of the fish. We have two ways of doing this: either the user classifies the fish, or we use machine learning to classify the fish.

Machine Learning - Regression

Regression is the task of assigning a real value to an object based on certain attributes that it displays. Regression is defined as a “method for understanding the relationship between independent variables or features and a dependent variable or outcome”(2). This allows us to find the relationship between variables that we already have access to and make predictions on the outcome from the patterns that we have previously experienced.

Since we are trying to measure the length of the fish we believe that it will be best to use regression to determine the length that the fish will be. We will take in values from the picture and use those values to predict the length that the fish is.

K-Nearest Neighbors

K-nearest neighbors is a supervised machine learning model that takes the k most similar points and produces a prediction or classification for the point of interest (5). The number of similar points that the algorithm will be looking at (k) will be varied to

see which gives us the best result. From our varied k's we find which value gives us the best outcome and that will be the model that we will use to calculate the length of the fish.

A common phrase in machine learning is "no free lunch" which basically means that you have to fit the model to the data that you have. As we don't have the data we are going to start with k-nearest neighbors (k-NN) to see if we are able to get some good result for our measurements. k-NN is a great model for both classification and regression so we believe that if we implement it correctly we can use the algorithm for both fish measurement and fish classification.

Depending on how the research goes we are not sure exactly where we are going to use machine learning, but we know that we will end up using machine learning no matter what. If we are able to use the ratio to accurately measure the fish, then we will not use machine learning to measure the fish. Instead we would likely use machine learning to predict the species of fish. This will also help us get a more accurate measurement.

Work Schedule

We have split our project, and the app, into four phases. These phases are Optimizing Measurement Formula, Code Design, UI and Implementation, and Integration.

Throughout the entire project we used an agile lifecycle approach, meeting once or sometimes twice a week. This schedule gave us ample time to work independently, and the frequent meetings allowed us to keep them short and efficient. This means we were both able to keep everyone up to date and discuss solutions to current and future problems.

Phase 1

Phase 1 is mostly dedicated to research and data collection, as in order to find an accurate and efficient formula we will need a lot of different resources. Actions taken during this phase includes:

- Meeting with fish and fish growth pattern experts
- Collecting photos of caught fish with known measurements
    - Preferably held by fisherman so as to provide positioning variety
- Manually measuring the fish in the photos so as to test the proposed formulas

The more accurate our formula, the more precise our measurements are. It is important that the formula is accurate across all species and sizes, but a broader usage of the formula means we may need to choose between less accuracy and more variables.

Phase 2

Phase 2 focuses on learning how to use the tools that we will be using, getting a grasp on how we use them. In order to do this the phase focuses on the following aspects:

- Understanding React Native and getting it to work on our machines
- Understanding the basics of javascript
- Getting a strong understanding about how we want to implement features
- Making sure that react and javascript can implement those features
- Continuing to make sure the formula is correct

Phase 3

Phase 3 focuses on the implementation of the UI that will allow the user and us to compute the calculations. In order to do this, the phase will focus on the following aspects:

- Uploading and storing the photo
- Allowing the user to interact with the photo
- Allowing the program to gather the necessary data from the photo
- Allowing the program to display the process on the photo
- Allowing the program to store the calculations with the photo

This will be one of the larger phases, as it will set the basis for the building of the app feature.

Phase 4

Phase 4 is where we make sure that all of the previous phases are working and make sure that the app that we have created is behaving the way that we had hoped. We will do testing to make sure that the app works on different types of fish and to make sure the UI is working properly. This is the phase where we are testing to make sure that our app will work for the live demo and that we have a working product in the end.

Major Milestones

Proposal Portfolio	November 25, 2022
Phase 1 - Formula and Research	January 22, 2023
Phase 2 - Code Planning and Design	February 20, 2023
Phase 3 - UI and Implementation	April 23 , 2023
Phase 4 - Integration	April 28, 2023

Responsibilities

Shared

- Measurement Research
- Fish anatomy research

Ethan Skelton

- Backend Calculations

Caitlynn Koback

- User Interface

Proposal Statement

Functional Requirements

- Ability to measure a fish accurately
- Ability to integrate into the onWater app
- Ability for user to interact with the app
- Place overlay on fish picture

Non-Functional Requirements

- Easy to use
- Well documented (code)
- Readable code
- Ability to log data
- Secure data transfer to database

Performance Requirements

Response Time:

- The program to measure a fish's length should be able to calculate the length in approximately 0.5 seconds from when the user presses the button to when the user receives the measurements
- The lines that the user moves around should feel responsive and not lag behind on their touch..

Interface Requirements

- Button for the user to press to calculate the length of the fish [See appendix B.2]
    - After the press UI updates to display length and picture with measurements that were taken on the fish to calculate the length
- User can open up picture to see where the machine found the eye:head ratio
- Allow the user to update the ratio if they believe its wrong

---

Development Standards, Tools, etc.

- React Native - development for onWater app
- Javascript - backend for onWater app
- Slack - for communication
- Github - for code storage and collaberation
- Metro, Android Studio, and XCode - making an emulator run our code

---

Methodology

Design Patterns Used

We have chosen to use the Abstract Factory for how we will be calculating the data (4). We chose the abstract factory pattern because we will be calculating the fish through a similar algorithm for all species of fish, but like we stated earlier the type of fish will affect the measurements that we use. So we decided that if we could abstract the data into different types of fish that it will make all of the measurement easier. This will allow us to sort measurements by the type of fish that we are dealing with.

[See appendix B.1]

Relevant UML Diagrams

[See appendix B.4, B.5, B.6, B.7, B.8, B.9, B.10]

Software Development Lifecycle

We have decided on the Agile Lifecycle (3) . We chose this life cycle approach for several reasons.

First, onWater is already using a very agile life cycle so we would like to stay as close to their work schedule. This is a major factor because we would like to work with onWater's schedule and not against it. Agile consists of scrums in order to stay up to date on all the progress that is being made in a project and sprints to get programs and processes pushed out as fast as possible (3). As onWater is a mobile app they have weekly sprints and scrums to keep progressing forward as fast as possible. We will be meeting with onWater every week on Wednesday in order to go over the work that we have done and the work that we plan to do going into the next week. We believe this life cycle will work best because it will allow us to fail fast and allow onWater to stay up to date with what we are doing.

Design Tradeoffs

One of the design tradeoffs that we faced while making the fish measuring app was trying to decide whether to create enough data to see if machine learning would be a viable path to finding the length of a fish, or whether we create an app that would be able to place that code into later and have an actual product to show. Since we knew that we could get a close measurement just

using a linear formula that we created we decided to create an app that would eventually host the backbone of a machine learning model. The reasons we decided not to use machine learning for the time being is because we would have first had to gather data by hand from a bunch of fish pictures with known length and for machine learning to work properly we would have had to gather hundreds of pictures.

Another design tradeoff we decided on was to use human input for measurements and not to try and use image recognition to find certain features on the fish. We could have used machine learning in order to determine certain features on a fish in order to calculate its length, but we figured that it would likely take to long to get both a UI to use plus train machine learning to find features on a fish, so we determined that we would reconsider if the UI took us less time than intended.

Expected Results

Given the fact that we are still unsure on whether or not the ratio will be of use and also the fact that machine learning algorithms can be very dependent on the data set. We will not know whether our measurements are good until we have run all of the tests and models. Hopefully with enough research we will be able to calculate the length of a fish in a picture accurately and should be able to display that in a way that is pleasant to the user.

Future Goals

In the future, we would like to gather a larger and more diverse database that would allow us to calculate formulas more accurate to individual species. Fish species can be extremely diverse, and while are formula works well for trout and fish similar to Rainbow Trout specifically, it lacks accuracy with smaller or young fish still growing.

Implementation of machine learning would allow us to identify individual species of fish, better implement individual and more accurate formulas for the species' lengths, and eliminate a large deal of user error when using the feature.

Using machine learning to identify the fish within a photo will allow the app to identify the necessary features of the fish, i.e the eye and head, in order to gain an accurate ratio between the two of them. This will allow the app to perform the necessary calculations without need for user input - eliminating the possibility of user error. Once the tool is capable of identifying the fish and its features, we can take it a step further in order to identify the species of fish and allow the app to assign species specific - and maybe even growth stage specific - formulas. If machine learning is to be fully implemented in this way, we would be able to do the entirety of the calculation behind the scenes, increasing efficiency and accuracy of the app.

References

[1] Richardson, JR, et al. "Using Relative Eye Size to Estimate the Length of Fish From a Single Camera Image." *Marine Ecology Progress Series*, vol. 538, Inter-Research Science Center, Oct. 2015, pp. 213–19. https://doi.org/10.3354/meps11476.

[2] Seldon. "Machine Learning Regression Explained." *Seldon*, 29 Oct. 2021, https://www.seldon.io/machine-learning-regression-explained.

[3]"The Agile Software Development Life Cycle: Wrike Agile Guide." *Wrike*, https://www.wrike.com/agile-guide/agile-development-life-cycle/.

[4]“Abstract Factory Pattern.” *GeeksforGeeks*, 13 Oct. 2021, https://www.geeksforgeeks.org/abstract-factory-pattern/.

[5]“What Is the K-Nearest Neighbors Algorithm?” *IBM*, https://www.ibm.com/topics/knn.

Appendix A (Qualifications)

{Resumes}

Ethan Skelton

Bozeman, Montana | (208) 651 – 1810 | ethan.skelton@student.montana.edu

PROFILE

I am a determined, hardworking, creative computer scientist with a love for solving challenging problems looking for opportunities to expand academic achievement into real world experience.

EDUCATION

Montana State University (2020 – Present graduate 2023 Fall)
Major Computer Science, Minor in Data Science

- Relevant Coursework: Statistics, AI, Graphical Analysis, and Machine Learning
- Achievements: Dean's List

SKILLS

Programming Languages: Java, Python, C#, C++, C, R, SQL
Other Experience: Embedded systems, Visual Studio, Git, Embedded, Design Thinking, Creating Specifications

EXPERIENCE

10Real (April '22 - August '22) – Developer
Responsibility for user experience design and game development for initial iOS augmented reality game, including low fidelity prototyping and initial game code to test design elements and user play experience. Developed and tested several hypotheses and for augmented reality interface.

Hudson's Hamburgers (May '22 - August '22) – Waiter
Working in a restaurant means focus on attention to detail and learning how to handle difficult customers.

Creekside Construction (April '21 - July '21) – Summer Worker
Responsible for various tasks at commercial job sites. Success was measured by willingness to take on challenging tasks and effectively support construction crew. Physically challenging work in a tough environment. Was highly regarded as go to worker by management.

School Projects:
Fish Measurement Project: Leading a group working alongside a company to design and create a fish measuring app that will be able to take a user's fish picture and perform some simple calculations to determine the length.

Facial Recognition Fridge: Combined both software with hardware to create a locking fridge that would scan a person's face to allow them access to the fridge.

Rover For EGEN 310: Project lead for multi-discipline engineering team to design a rover. Managing people, project scope, dependencies, and schedule.

References available on request

Caitlynn Koback

Caitlynn_koback@yahoo.com | 951-455-8393 | 128 Hood Drive, Dayton NV, 89403

An enthusiastic learner who is cooperative and excited to work with others. Friendly, patient, and supportive.

Experience

Tutor | Dayton Intermediate School - Dayton, NV | 11/2017 - 6/2018

- Helped students with subject knowledge and general abilities in learning, studying, and retaining information.
- Assisted students with taking advantage of other available resources.

Floater | Montana State University Dining Halls - Bozeman, MT | 09/2020 - 11/2020

- Support other colleagues and kitchen staff by proactively including side work duties
- Maintain cleanliness of tables and work stations in accordance to customer satisfaction and safety
- Ensured positive overall customer service

Education

Dayton High School | Dayton, NV
High School Diploma 2016-2020

Western Nevada College | Carson City, NV
Associate of Science 2018-2020

Montana State University | Bozeman, MT
Bachelors of Science, Computer Science 2020 - 2023
Minor in Computer Engineering

Skills

Languages: Java, Git, C, C++, Python, JavaScript
Operating Systems: Windows, Linux

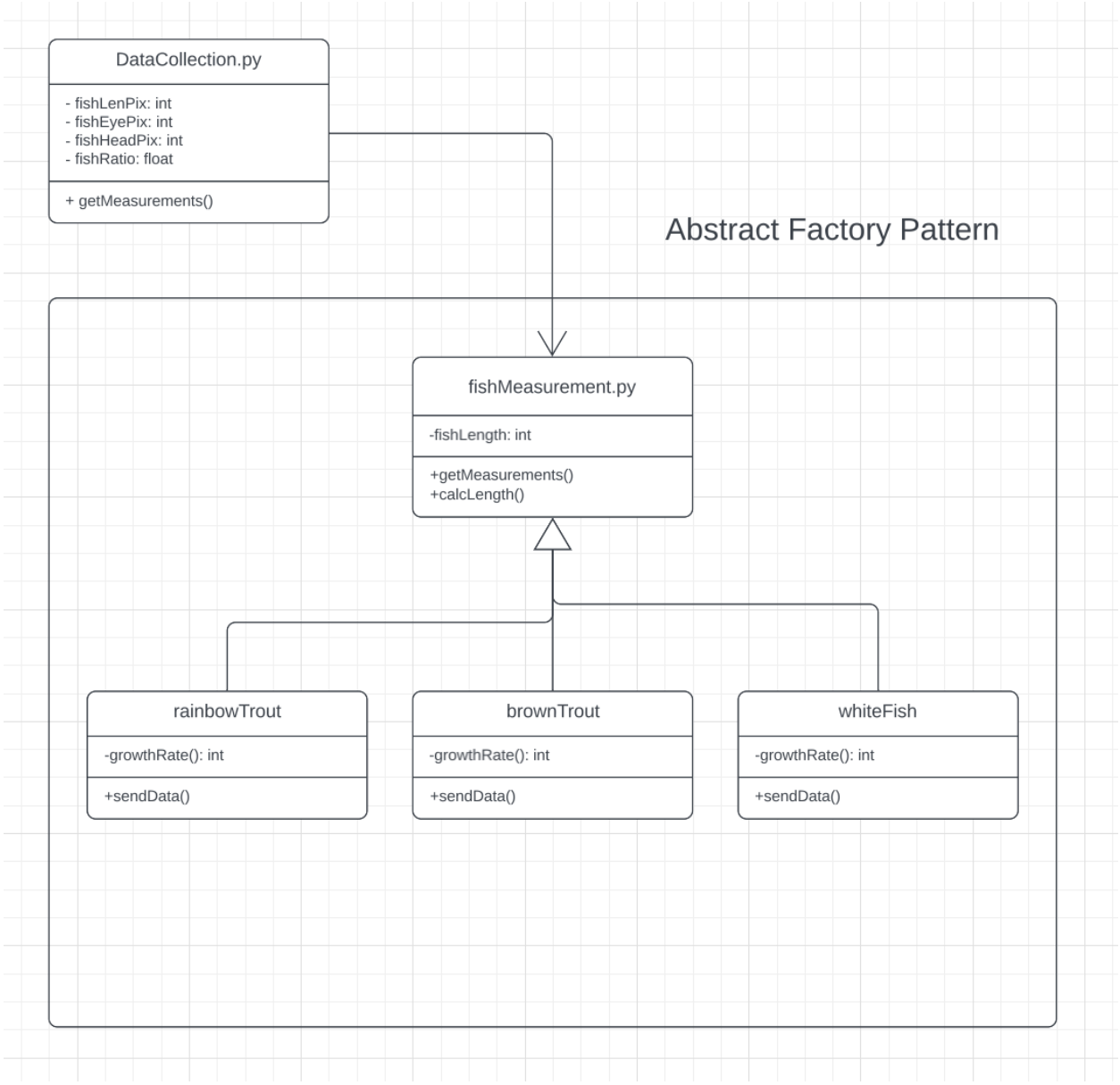
Projects

UX App Design -
Designed an alarm app Fall 2022

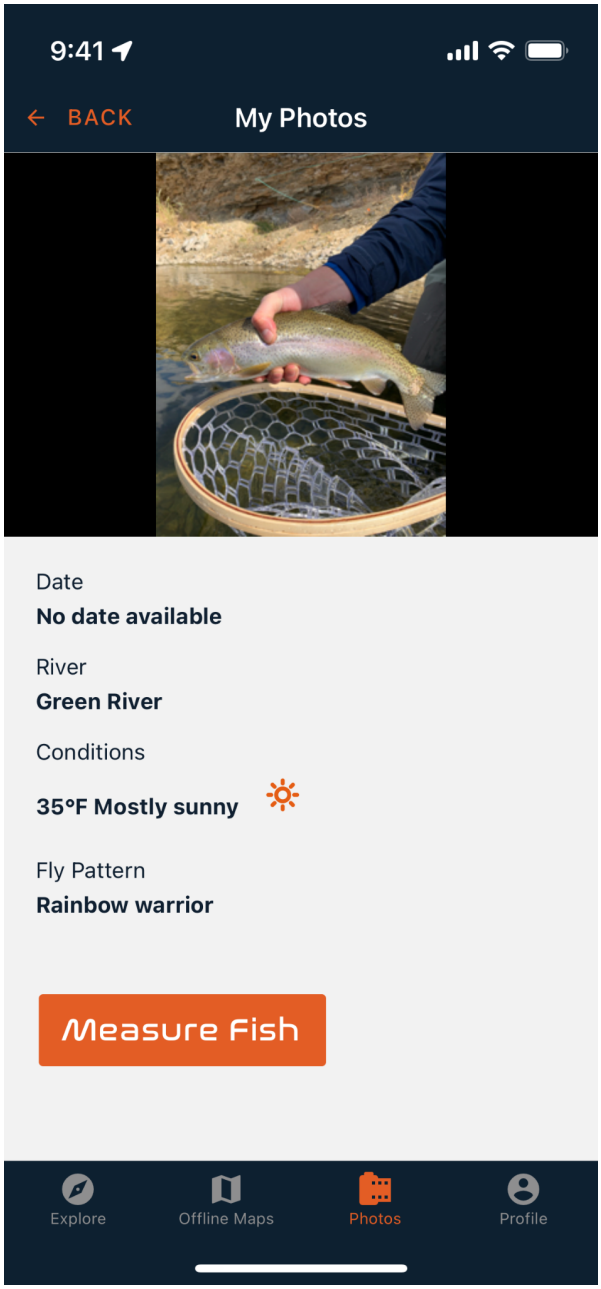
Rover -
Wired and programmed a rover to communicate over bluetooth through an xbox controller using a UWP developed for this project

Appendix B (Diagrams)

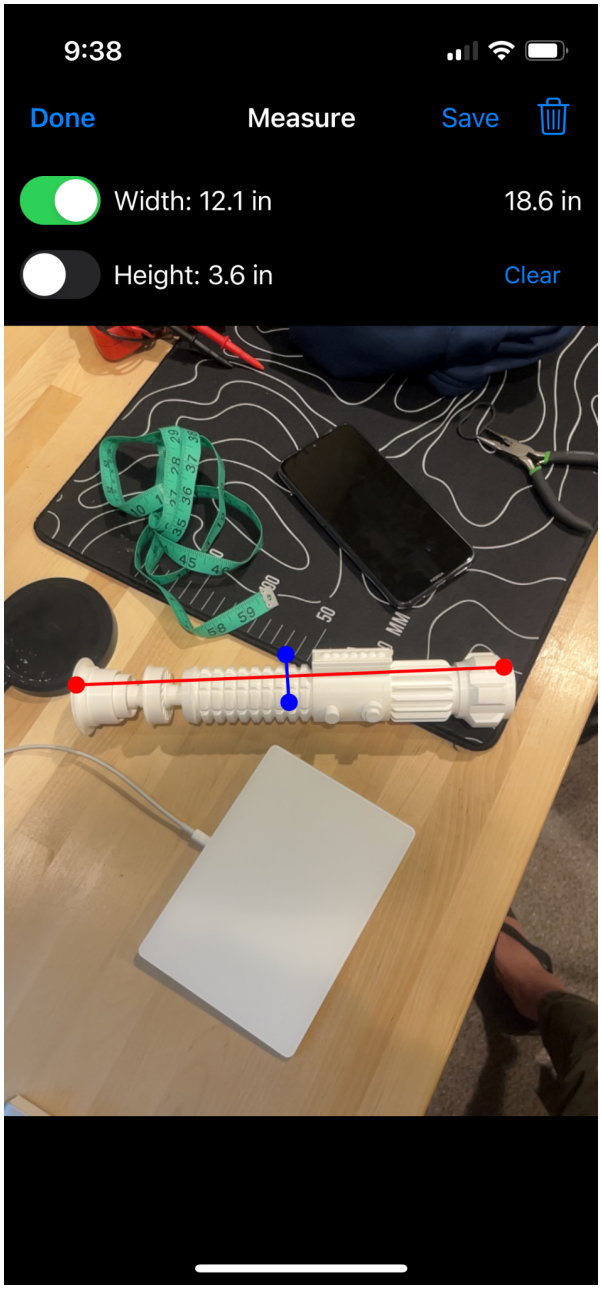
B.1 (Abstract Factory Pattern)



B.2 (UI Design)



1. What the “calculate” button will look like

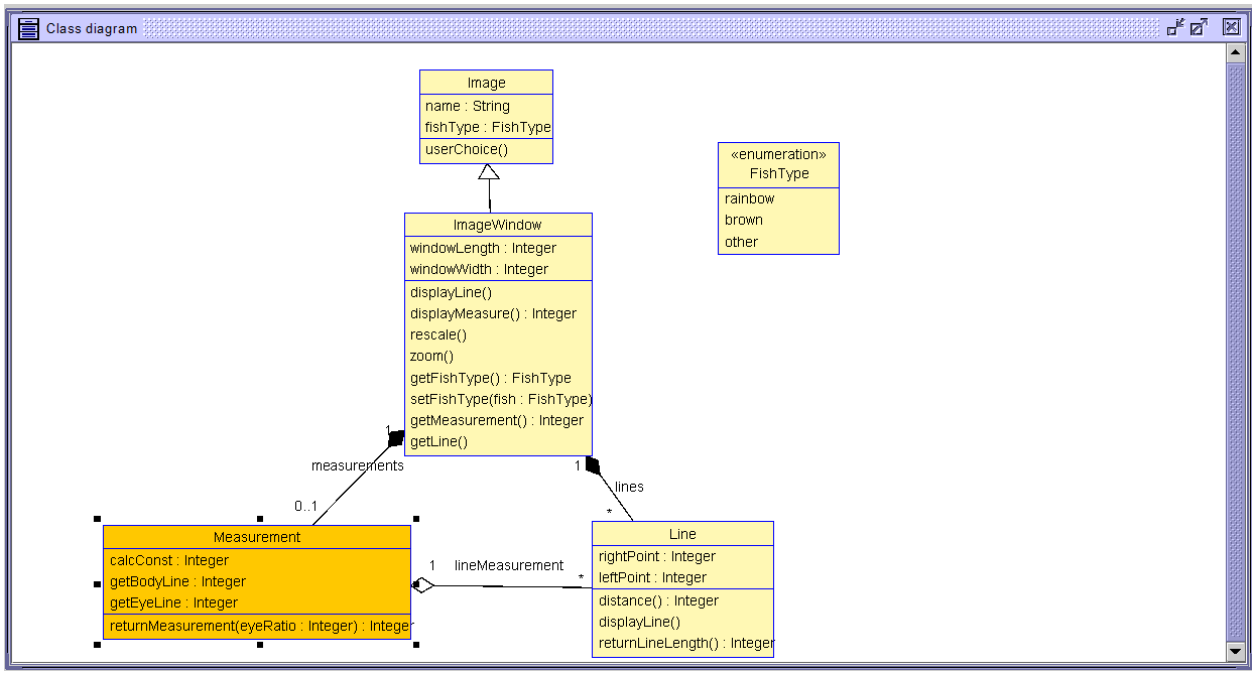


2. What the lines that will be on the fish will will look like

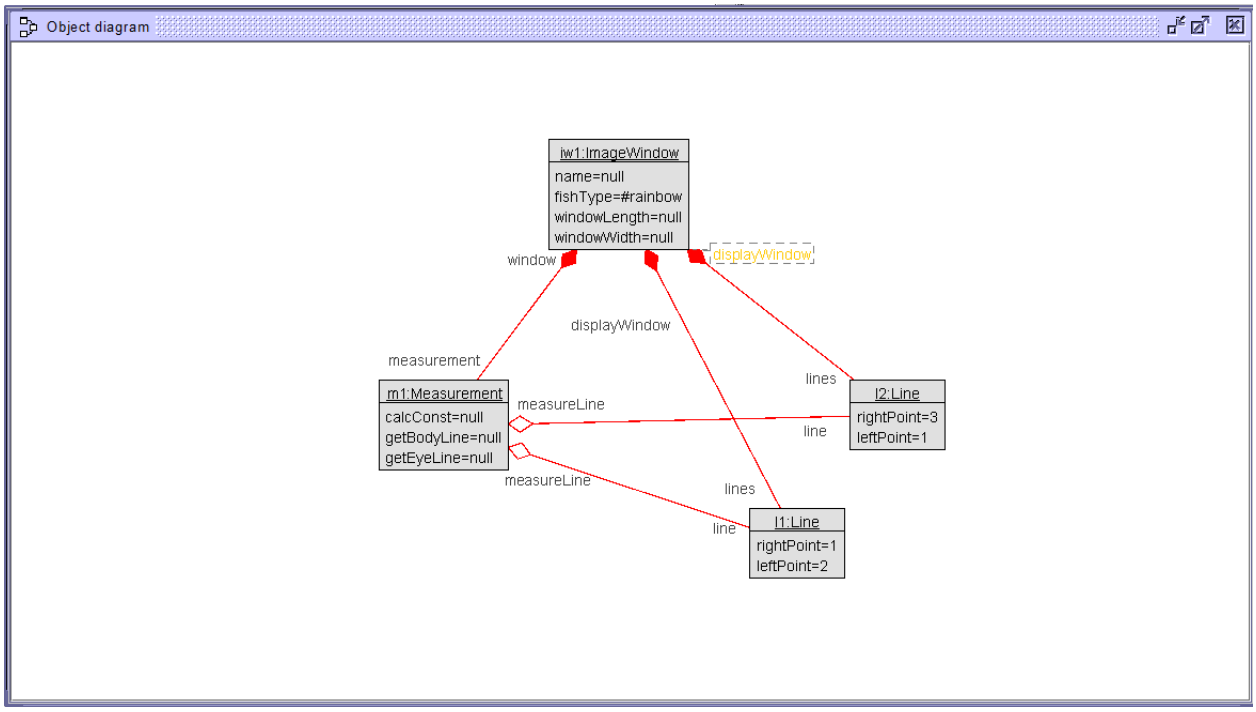
---

B.3 (Fish Measurements)

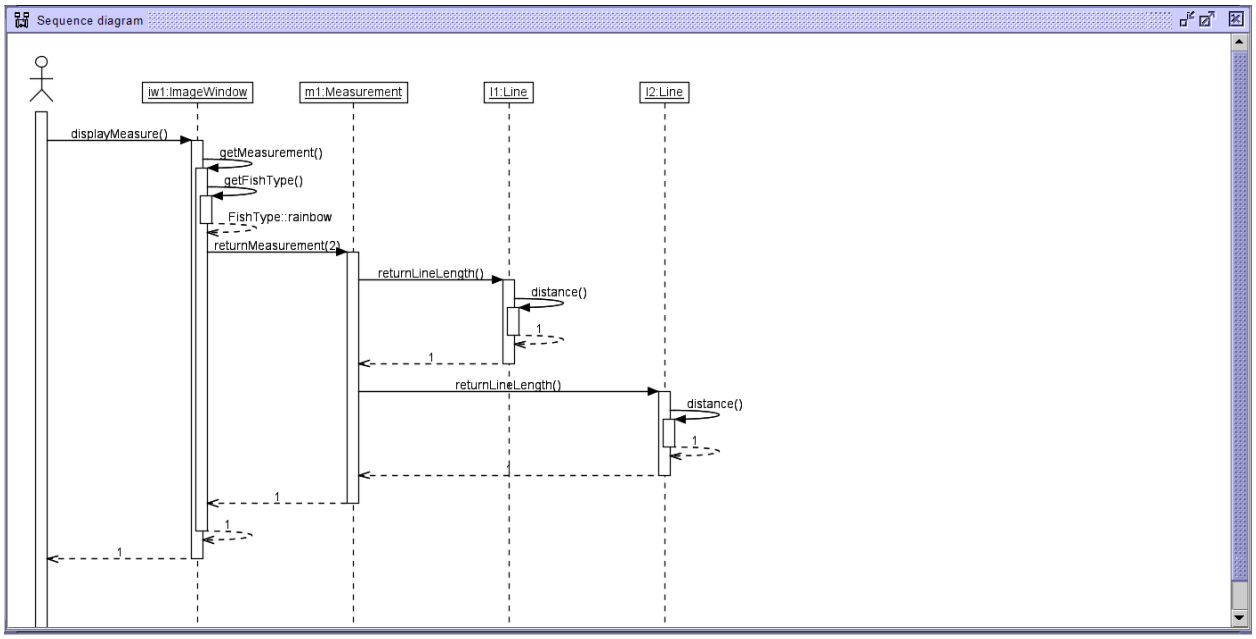
B.4 (Class Diagram)



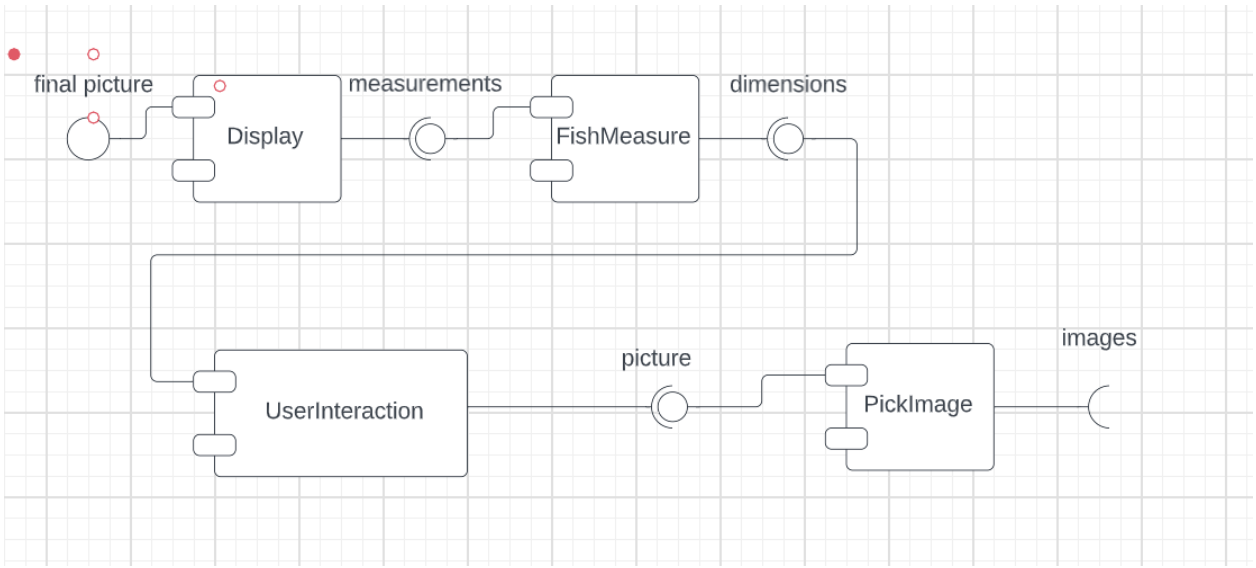
B.5 (Object Diagram)



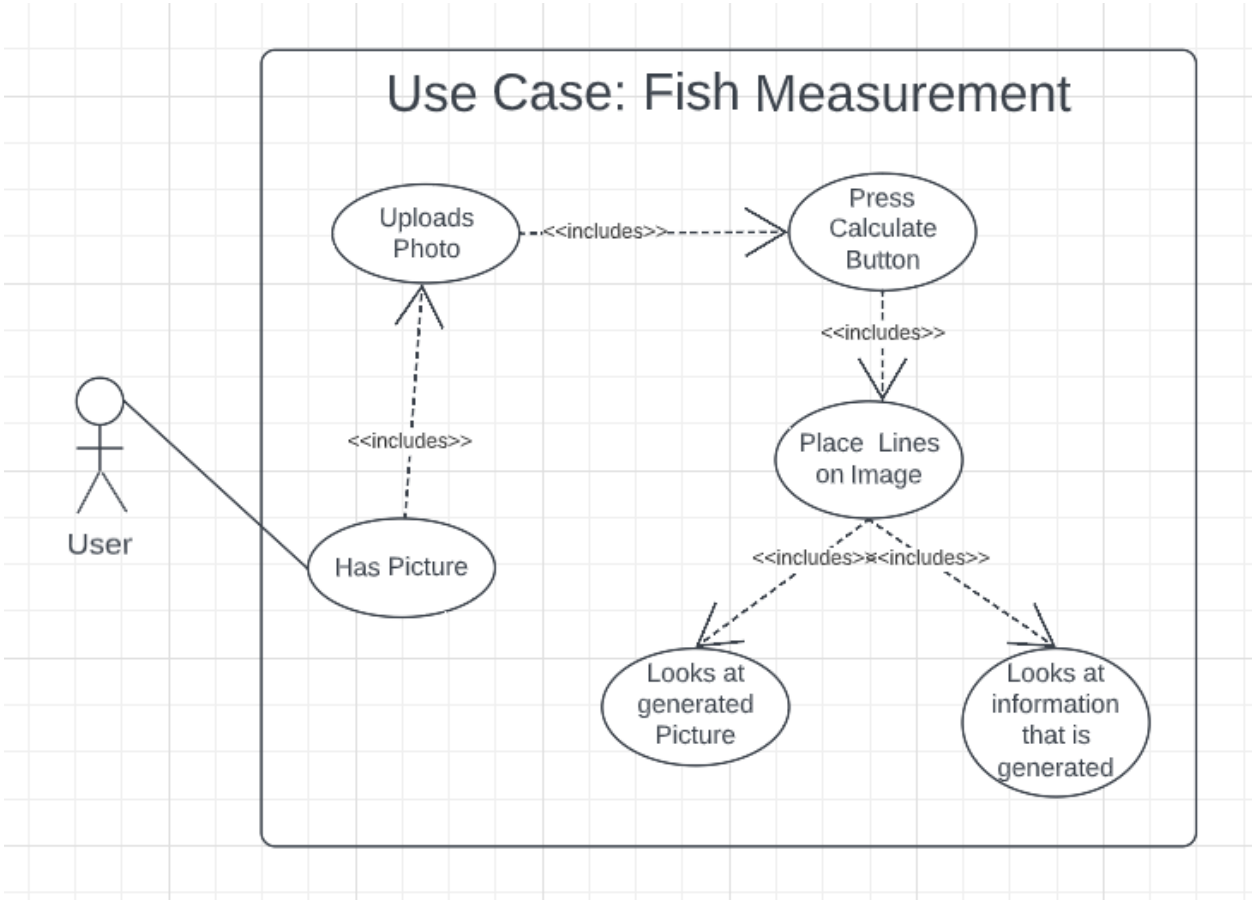
B.6 (Sequence Diagram)



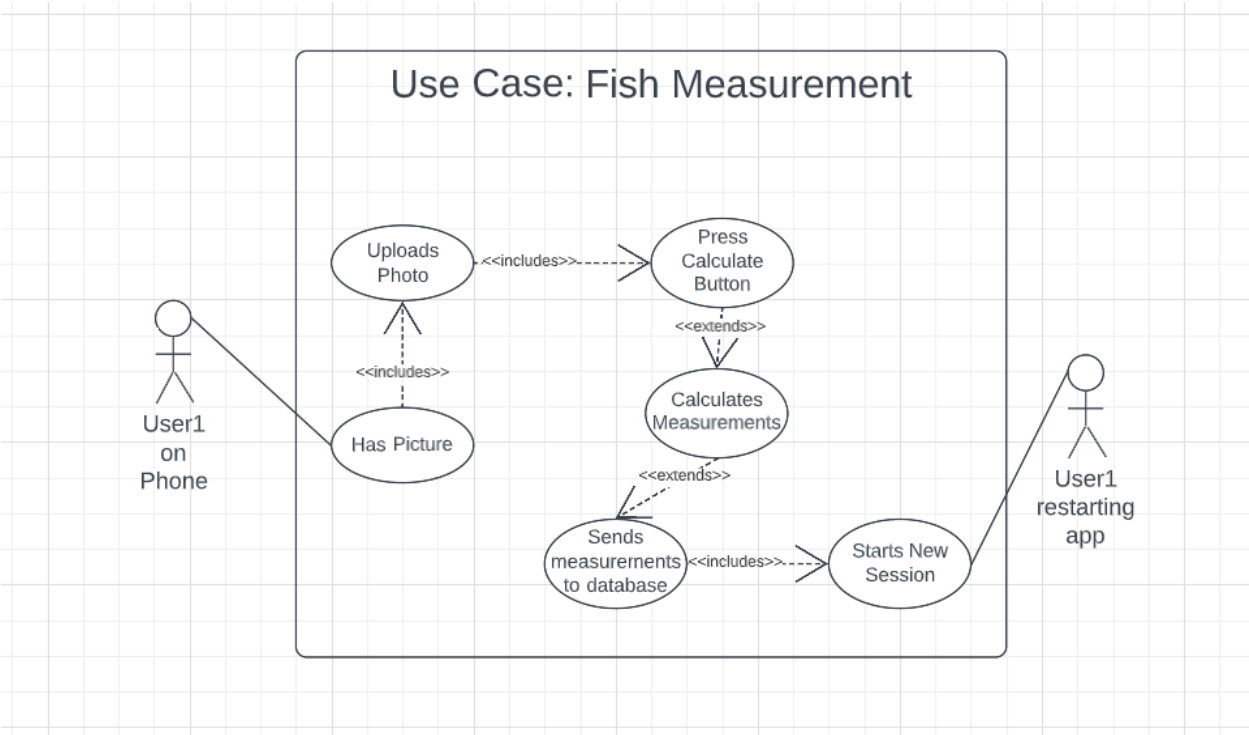
B.7 (Component Diagram)



B.8 (Use Case Fish Measurement)



B.9 (Persistance)



Appendix C (Code)

App.tsx

```
//start imports
import * as React from 'react';
import { Button, View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
//end imports

//start file imports
import LengthScreen from "./LengthScreen";
import PictureScreen from "./PictureScreen";
import EyeLengthScreen from "./EyeLengthScreen";
import DisplayScreen from './DisplayScreen';
//end file imports

//creates a stack navigator so that we can change in between screens
const Stack = createNativeStackNavigator();

//App function serves as navigation so that we can get from one page to another
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={PictureScreen} />
        <Stack.Screen name="Length" component={LengthScreen} />
        <Stack.Screen name="EyeLength" component={EyeLengthScreen} />
        <Stack.Screen name="Display" component={DisplayScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

DisplayScreen.tsx

```
//start imports
import { StyleSheet, Text, View, TouchableOpacity, Image } from 'react-native';
//end imports

//start file imports
import CalcLength from './CalcLength';
//end file imports

//Controls the screen where all the information about the screen is diplayed
const DisplayScreen = ({ navigation, route }) => {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        {/* Displays the image since Line isnt being called */}
        <Image
        source={{ uri: route.params.uri }}
        style={{ width: '100%', height: '80%' }}
        />
        {/* Calculates the length and displays it below the picture */}
        <CalcLength head={route.params.head} eye={route.params.eye}/>
        {/* Sends the user back the start so that they can try another picture */}
        <TouchableOpacity onPress={() => navigation.navigate('Home')}
style={styles.button}>
            <Text style={styles.buttonText}>Restart</Text>
        </TouchableOpacity>
      </View>
    );
  }

  export default DisplayScreen;

  const styles = StyleSheet.create({

    button: {

      width: 250,

      height: 60,

      backgroundColor: '#ff6500',

      alignItems: 'center',

      justifyContent: 'center',

      borderRadius: 4,
```

```
    marginBottom:12
  },
  buttonText: {
    textAlign: 'center',
    fontSize: 15,
    color: '#ffffff'
  }
});
```

PictureScreen.tsx

// App.js

```
//start imports
import React, { Component, useState } from 'react';
import { StyleSheet, Text, View, TouchableOpacity, Button, Image, ImageBackground }
from 'react-native';
import ImagePicker from 'react-native-image-crop-picker';
//end imports

//Picture screen allows user to pick and edit their picture to send through the
calculations
export default class PictureScreen extends React.Component<{}, {resourcePath: string
| undefined}> {

  constructor(props: any) {

    super(props);

    this.state = {
      resourcePath: '',
    };
  }

//select file selects a file to send to calculate and edits the phote to users liking
  selectFile = () => {
    var options = {
      mediaType: 'photo',
      includeBase64: true,
      title: 'Select Image',

      customButtons: [

        {

          name: 'customOptionKey',

          title: 'Choose file from Custom Option'

        },

      ],

      storageOptions: {

        skipBackup: true,
```

```
        path: 'images',

      },

    };

    ImagePicker.openPicker({

      //Check user response, if the library is successfully opened, jump to else
      cropping: true,

    }).then(image => {
      this.setState({
        resourcePath: image.path
      });
    });

  };

  render() {

    return (

      <View style={styles.container}>
         {/* displays the image that the user will be editing */}
          <Image

            source={{ uri: this.state.resourcePath }}

            style={{ width: '100%', height: '80%' }}

          />

          {/* A button to select a photo that the user wants to calculate */}
        <TouchableOpacity onPress={this.selectFile} style={styles.button}>

            <Text style={styles.buttonText}>Select File</Text>

        </TouchableOpacity>

          {/* A button to start the calculation process. Sends the photo to the next
screen */}
        <TouchableOpacity onPress={() => this.props.navigation.navigate('Length',
{uri: this.state.resourcePath})} style={styles.button}>

                <Text style={styles.buttonText}>Calculate</Text>

        </TouchableOpacity>
```

```
      </View>
    );
  }

}


//styles for react native
const styles = StyleSheet.create({

  container: {

    flex: 1,

    padding: 30,

    alignItems: 'center',

    justifyContent: 'center',

    backgroundColor: '#fff'

  },

  button: {

    width: 250,

    height: 60,

    backgroundColor: '#ff6500',

    alignItems: 'center',

    justifyContent: 'center',

    borderRadius: 4,

    marginBottom:12

  },

  buttonText: {

    textAlign: 'center',

    fontSize: 15,
```

```
    color: '#ffffff'
  }
});
```

Line.tsx

```
//start imports
import Svg, {Line} from 'react-native-svg';
import React, {useState, useRef, useEffect} from 'react';
import {Animated, View, StyleSheet, PanResponder, Text, Image} from 'react-native';
//end imports

// draws the line onto the screen using a svg
const drawLine = (
  x_1:string,
  y_1:string,
  x_2:string,
  y_2:string
  ) => {
  return (
    <Svg>
      <Line x1={x_1} y1={y_1} x2={x_2} y2={y_2} stroke="#ff6500" strokeWidth="1" />
    </Svg>
  );
};

//turns a value into an integer value that I can manipulate
const valToNum = (
  val: Animated.Value
) => {
  return (
    Number(JSON.stringify(val))
  )
}

// calculates the distance of the line
const calcDistance = (
  x_1:string,
  y_1:string,
  x_2:string,
  y_2:string
  ) => {
  return (
    Math.sqrt(Math.pow((Number(x_1) - Number(x_2)), 2) + Math.pow((Number(y_1) -
Number(y_2)), 2)).toFixed(2)
  );
};



// the LineMovement class
const LineMovement = ({uri, onChange}) => {
```

```
  //local variables start
  const cirDim = 10;
  const cirColor = "#ff6500"
  const endpoint1 = useState(new Animated.ValueXY())[0];
  const endpoint2 = useState(new Animated.ValueXY())[0];

  const [dist, setDist] = useState("0")

  const [x1, setX1] = useState("0");
  const [y1, setY1] = useState("0");
  const [x2, setX2] = useState("0");
  const [y2, setY2] = useState("0");

  //local variables end

  //When one of the points change then set the distance
  useEffect(()=>{
    setDist(calcDistance(x2,y2,x1,y1));
  },[x1,y1,x2,y2])

  //when the distance changes then send it out to the onChange variable to be used
outside
  useEffect(()=>{
    onChange(dist);
  },[dist])

  //first dot code
  const panResponder1 = useState(
    PanResponder.create({
      onMoveShouldSetPanResponder: () => true,
      onPanResponderMove: (_, gesture) => {
        endpoint1.x.setValue(gesture.dx)
        endpoint1.y.setValue(gesture.dy)

      },
      onPanResponderRelease: (_, gesture) => {
        endpoint1.extractOffset();
        //Sets the x1 and y1 values to use to create the line
        setX1((valToNum(endpoint1.x)-30 + 236).toString());
        setY1((valToNum(endpoint1.y)-85 + 572 - cirDim/2).toString());

      },
    }),
  )[0];

  //second dot code
  const panResponder2 = useState(
    PanResponder.create({
```

```
      onMoveShouldSetPanResponder: () => true,
      onPanResponderMove: (_, gesture) => {
        endpoint2.x.setValue(gesture.dx)
        endpoint2.y.setValue(gesture.dy)
      },
      onPanResponderRelease: (_, gesture) => {
        endpoint2.extractOffset();
        //Sets the x2 and y2 values to use to create the line
        setX2((valToNum(endpoint2.x)-30 + 236).toString());
        setY2((valToNum(endpoint2.y)-85 + 572 + cirDim/2).toString());
      },
    }),
  )[0];

//what will be rendered from the class
  return (
  <View style={styles.container}>
    <View style={styles.center}>
      {/* Displays the distance of the line */}
      <Text>
        {dist}
      </Text>
      {/* Displays the images */}
      <Image

      source={{ uri }}

      style={{ width: '100%', height: '80%' }}

      />
      {/* Draws the line onto the image */}
      <View style={styles.behind}>
        {drawLine(x2,y2,x1,y1)}
      </View>
      {/* Draws the first dot onto the screen */}
      <Animated.View
      style={[
        {
          width: cirDim,
          height: cirDim,
          transform: [
            {
              translateX:endpoint1.x
            },
            {
              translateY:endpoint1.y
            }
          ],
```

```
          borderRadius: cirDim/2,
          backgroundColor: cirColor
        }
      ]}
      {...panResponder1.panHandlers}
      />

      {/* Draws the second dot onto the screen */}
      <Animated.View
      style={[
        {
          width: cirDim,
          height: cirDim,
          transform: [
            {
              translateX:endpoint2.x
            },
            {
              translateY:endpoint2.y
            }
          ],
          borderRadius: cirDim/2,
          backgroundColor: cirColor,
          justifyContent:'center'
        }
      ]}
      {...panResponder2.panHandlers}
      />
      </View>

    </View>
  );
};


const styles = StyleSheet.create({
container: {
  flex: 1,
  alignItems: 'center',
  height: '100%',
  width: '100%',
  justifyContent: 'center',
},
center: {
  width: '100%',
  height: '100%',
  alignItems: 'center',
  justifyContent: 'center',
```

```
},
behind: {
  alignItems: 'center',
  justifyContent: 'center',
  position: 'absolute',
  width: '100%',
  height: '100%'
}
});

export default LineMovement;
```

LengthScreen.tsx

```
/start imports
import { StyleSheet, Text, View, TouchableOpacity } from 'react-native';
//end imports
//start file imports
import LineMovement from './Line';
//end file imports

//Controls the Screen where the user will calculate the head length
const LengthScreen = ({ navigation, route }) => {
  //start local variables
  let headHeight = '0';
  //end local variables

  return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        {/* Displays the lines and picture on the screen */}
        <LineMovement uri={route.params.uri} onChange={(distance)=>{
          headHeight = distance
        }}/>
        {/* Button to move onto the next screen and send the pixel distance with it
*/}
        <TouchableOpacity onPress={() => navigation.navigate('EyeLength', {uri:
route.params.uri, head:headHeight})} style={styles.button}>
            <Text style={styles.buttonText}>Calculate</Text>
        </TouchableOpacity>
      </View>
    );
  }

  export default LengthScreen;



  const styles = StyleSheet.create({

    button: {

      width: 250,

      height: 60,

      backgroundColor: '#ff6500',

      alignItems: 'center',

      justifyContent: 'center',
```

```
    borderRadius: 4,
    marginBottom:12
  },
  buttonText: {
    textAlign: 'center',
    fontSize: 15,
    color: '#ffffff'
  }
});
```

EyeLengthScreen.tsx

```
//start imports
import { StyleSheet, Text, View, TouchableOpacity } from 'react-native';
//end imports
//start file imports
import LineMovement from './Line';
//end file imports

//Controls the Screen where the user will calculate the eye length
const EyeLengthScreen = ({ navigation, route }) => {
    //start local variable
    let eyeHeight = '0';
    //end local variables

    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        {/* Displays the line and image on the screen */}
        <LineMovement uri={route.params.uri} onChange={(distance)=>{
          eyeHeight=distance
        }}/>
        {/* Button to move on to the next screen and send values with it */}
        <TouchableOpacity onPress={() => navigation.navigate('Display', {uri:
route.params.uri, head:route.params.head, eye:eyeHeight})} style={styles.button}>
            <Text style={styles.buttonText}>Calculate</Text>
        </TouchableOpacity>
      </View>
    );
  }

  export default EyeLengthScreen;



  const styles = StyleSheet.create({

    button: {

      width: 250,

      height: 60,

      backgroundColor: '#ff6500',

      alignItems: 'center',

      justifyContent: 'center',
```

```
    borderRadius: 4,
    marginBottom:12
  },
  buttonText: {
    textAlign: 'center',
    fontSize: 15,
    color: '#ffffff'
  }
});
```

CalcLength.tsx

```
//start imports
import React, {useState, useEffect} from 'react';
import {Animated, View, StyleSheet, PanResponder, Text, Image} from 'react-native';
//end imports

//CalcLength will calculate the fishes length based on certain parameters
//head - head length in pixels
//eye - eye length in pixels
const CalcLength = ({head, eye}) =>{

    //start local variables
    const offset = 2;
    const slope = 53;
    const [fishType, setFishType] = useState("Rainbow");
    const [headHeight, setHeadHeight] = useState("-1");
    const [eyeHeight, setEyeHeight] = useState("-1");
    const [totalLength, setTotalLength] = useState("Not Calculated Yet");
    const [ratio, setRatio] = useState(1);
    //end local variables

    //sets values that were sent into file
    const setValues = (
        head:string,
        eye:string

    ) => {
        setEyeHeight(eye)
        setHeadHeight(head)
    }

    //calculates the length based off of variables that were found
    const calcLength = () => {
        return slope*ratio + offset;
    }

    //calculates the eye:head ratio that is used to find the length
    const getEyeHeadRatio = () => {
        return Number(eyeHeight)/Number(headHeight)
    }

    //When the headHeight changes then it sets the ratio so that the ratio is correct
for render
    useEffect(() =>{
        setRatio(getEyeHeadRatio)
    }, [headHeight]);
```

```
    //When the ratio changes then it sets the total length so that the total length
is correct at render
    useEffect(() =>{
        setTotalLength(calcLength().toString())
    }, [ratio]);

    //when the file is called on startup this function is called
    useEffect(() => {
        setValues(head,eye)
    }, []);

    //returns the calculated length when the file is called
    return (
        <View>
            <Text>Calculate Length</Text>
            <Text>{totalLength}</Text>
        </View>
    )
}

export default CalcLength;
```