2023-
2024

RedTrade Design Document

J. KLEIST, B. UNDERWOOD

Contents

Abstract

RedTrade is a stock simulation application targeted at individuals who are interested in learning more about the intricacies of financial trading. Its core features are interactive data visualization of stocks, clear explanations of financial terms, and a tool to allow users to invest and hold "play money" on the market. Redtrade will use AlphaVantages's capabilities to pull stock ticker information. By design, there will be a strong security component that helps ensure the integrity of personal information. RedTrade will use common software technologies (i.e., React Native for front end and Supabase for backend components), and will adopt native security infrastructure to create base line security, as well as the standard AES256 Encryption for session data. Software component Snyk will be used to validate the source code for common weaknesses and other vulnerabilities during the design phase.

Introduction

As of 2020, approximately 51% if families in the United States (U.S.) have holdings on the stock market. Breaking this down by age, only 41% of those with holdings on the stock market are aged 35 or below. (Parker, 2020) Investing is important because in the last 50 years, the buying power of the U.S. dollar has gone down by 675%. With the average savings account having a .23% Annual Percent Yield (APY), and inflation rates averaging around 2% by keeping money strictly in savings, that money will be losing value every year. The stock market has a return rate of around 10% based upon the S&P 500. From these statistics it should be clear that the only way to really grow money is by putting it on the stock market.

We at RedTrade believe that one of the key reasons behind the low amount of U.S. families with holdings on the stock market is a lack of financial education and exposure for young people, whether the education comes from parents or schools. When you consider how important investing is for achieving things such as home ownership and retirement, and that investing when you are young is the best way to grow your money, this number needs to change.

In the world of investing and trading, there is a stark absence of tools directed toward novice or new traders that approach the issue from an educational perspective. Websites that try to incorporate educational aspects to their stock market simulation, typically approach it from a point of gamification, giving people hundreds of thousands of dollars to put on the market. This creates a message that trading is just a game, as well as implies that you need to have a ton of money to invest in the market. While this approach may be fine for people who already know the ins and outs of the stock market, for someone new to trading this message is inappropriate because it makes investing seem out of reach, as well as makes investing seem directly adjacent to gambling. Especially for a young impressionable audience, the stock market needs to be approached from a point of realism.

That's where RedTrade comes in. With RedTrade, we aim to create a web application for people new to trading to experiment and become more confident and knowledgeable about what it looks like to trade securities, risk-free. RedTrade will have a strong focus on security measures to ensure all user information, from profiles to transaction histories, is safeguarded against unauthorized access. It will allow users to create a profile, choose how much money they want to begin with, and start buying and selling securities on a virtual stock market. It will have high quality charts the user can interact with, as well as a modern user interface implemented using React Native, a multiplatform web framework created by Meta. To make testing and debugging simpler, we will be using the Expo platform to test on both IOS and Android devices. User profiles, as well as the associated information such as held securities and settlement funds will be held in Supabase, an open-source cloud database. This database has integrated security protocols and allows for massive customization of user permissions.

Background

To make sure we are taking a unique approach to stock market education, we investigated the companies who provide comparable services to RedTrade. What we found is there are a few solutions currently available, but these services are lacking in several key areas. We used Investopedia, Wall Street Survivor, and Market Watch as examples of how RedTrade's service offerings are filling a unique niche in the field.

Investopedia's platform has a dated user interface, providing users with options to engage in virtual trading and maintain a history of their transactions. As a starting point for virtual trading, Investopedia allocates $100,000 to users, which is considerably more money than what most real-world traders can afford. This leads to issues with gamifying the stock market because it creates a perception that people just throw around huge amounts of money on the stock market, along with them having a "Game Leaderboard". Regarding data visualization, the platform is mediocre at best. Many of their visuals are pulled directly from stock ticker APIs with no transformation done. One of the largest drawbacks with Investopedia's data visualization is the lack of in-depth stock information on the individual stock pages, which leaves users wanting more details. Overall, the visualization features offered are quite simplistic and could use some modernization.(Gordon, 2023)

Wall Street Survivor's website also suffers from several design and usability issues. Their website is cluttered with an overwhelming amount of content, features, and advertisements, compromising the user experience. When it comes to a tool that is supposed to help people instill confidence in their ability to trade, being bombarded with advertisements and social media feeds detracts from the intended experience. In fact, the lack of advertisement regulation leads to frequent suspicious advertisements, diminishing the website's credibility. Besides the over-the-top advertisements, the tools that are a part of the simulation are not clearly differentiated from the rest of the site's content, leading to confusion when attempting to navigate the page. Additionally, the design of the website appears outdated, giving the impression that it hasn't been refreshed to keep up with current web standards. We believe that this website cares more about its revenue and clicks, as opposed to creating a stock market simulation that will be useful and educational to users. ("*Wall Street Survivors*")

Finally, MarketWatch's gaming section is directed specifically at groups of people all trading together. Each user can create or join "games" with either people they know or strangers. The good things that MarketWatch does compared with competitor websites is allow high levels of customization when setting up these games, which allow for a tailored experience for each new user. Unfortunately for someone who is new to trading this might be a bit on the complex end for them to understand. In addition to this, since this website is primarily a financial news and information website, it has a huge amount of feature bloat, and it's easy to navigate away from the

simulation into something that is more complicated than a new user needs. On top of that, if you accidentally navigate away from the simulation, it is near impossible to quickly get back to it. The platform is a part of the Dow Jones Network, a significant name in the financial information industry, giving MarketWatch more credibility among users. Despite this backing, its ambition to cover a wide array of functionalities compromises the quality of the individual features and the usability of the site. ("*Virtual Stock Exchange*")

From the competitors we researched, we found that the two major trends of stock market simulators are that most of them are created as an afterthought to the rest of their content, and they focus on showing the stock market as a game. However, as college students we recognize the potential danger in trivializing financial concepts down to a game. RedTrade is designed to be more than a game. It will provide a modern user interface with the specific focus on educating those new to the world of investing. By making our priority the end-user and offering features like customizable starting capital, simple explanations, and simulated recurring deposits RedTrade aims to guide users toward financial independence.

Qualifications

Jack E. Kleist

Bozeman, MT 59718 | 920.740.8918 | jackkleist@gmail.com | linkedin.com/in/jackkleist

Professional Profile

Highly motivated computer science student with a strong understanding of CS fundamentals and desire to be work in the IT field.

Education

Bachelor of Science, Computer Science (GPA 3.79) — Exp. Grad. Date: May 2024
Montana State University (MSU)

Relevant Coursework

Software Engineering, Data Structures and Algorithms, Discrete Structures, Cyber Security, Intermediate Tech Writing, Programming with C I, Systems Administration, User Interface Design, Digital Forensics

Professional Affiliations

MSU Chapter of Association for Computing Machinery — September 2022 – Present

Skills

Languages: Java, C, C++, Python, UML, SQL

Tools: Volatility 2 & 3, CyberChef, VirusTotal, GitHub, Git, IntelliJ, vim, Linux, MATLAB

Soft: communication, team player, enthusiastic learner, multi-tasking, detail oriented

Work Experience

User Support Associate

Montana State University, Bozeman, MT — October 2022 – September 2023

- Solve user problems using knowledge of Windows and Mac OS
- Work closely in a team within an enterprise level IT environment
- Create written documentation on found solutions to issues

Software Engineer Intern

Fast Enterprises, WA — May 2023 – August 2023

- Learned how to work within a corporate development environment
- Pushed project to production
- Actively worked and communicated with our client to determine project requirements

Volunteering

Bear Canyon Clean Up — September 2022

Brady Underwood

Maple Valley, WA 98038 | 425-358-1910

Email: underwood.brady@gmail.com | Website: https://brady.games/projects

Professional Summary

Junior in college at Montana State University studying computer science. Quick thinker and creative problem solver with a keen interest in software development, always striving to learn new things.

Technical Skills

- Programming languages: Java, Python, C, C++, C#, JavaScript (Node.js, React.js), HTML/CSS
- Query Languages: SQL, MongoDB
- Adobe Suite: Photoshop, Illustrator, Premiere

Education

Montana State University (2020-2024), *Bozeman MT*

Bachelor's in Computer Science, *Minor in Business Administration*

Green River College (2018-2020), *Auburn WA*

Associate in Business Administration

Work Experience

Frontend Development Intern (June 2023 – August 2023)

Propagation Labs, *Salt Lake City UT*

- Created and styled components using Flutter
- Developed basic logic in Dart programming language

Software Developer Intern (June 2022 – October 2022)

Envestnet, *Philadelphia PA*

- Contributed to designing a front-facing developer portal using React.js and Redux
- Created mobile app using React Native and Expo
- Participated in Agile methodology and daily standups

Freelance Web Designer (2019 - Current)

N/A, *Maple Valley WA*

- Assisted in designing and developing professional websites for a range of clients using HTML and React.js
- Utilized SSR frameworks such as SvelteKit and Next.js
- Implemented full stack routing with Express.js and Socket.io

Honors & Interests

- **Big Idea Challenge Finalist (2021)**

Received $1000 award from Blackstone Launchpad for placing first in the consumer products and services category.

- **Backend Development** *(Node.js, Python, MongoDB)*

Produced a variety of web applications using Node.js and Socket.io, from real time games to website hosting.

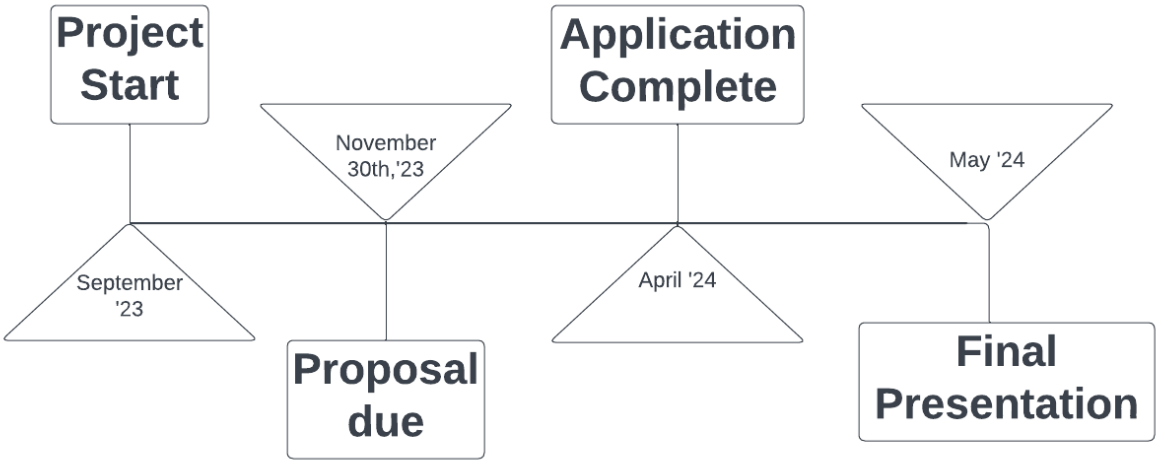
Work Schedule

Fall Semester

- ☑ 8-24: Started looking for Partners.
- ☑ 8-31: Posted ideas about project sponsors.
- ☑ 9-07: Made decision to create RedTrade
- ☑ 9-14: Reviewed project ideas with class
- ☑ 9-21: Began work on project abstract
- ☑ 9-28: Turned in RedTrade abstract
- ☑ ~~10-6~~: Discussed the format of the portfolio
- ☑ 10-12: Began work on the design portfolio
- ☑ 10-19: Created slides to describe RedTrade
- ☑ 10-26: Continued work on the design portfolio
- ☑ 11-02: Continued work on the design portfolio
- ☑ 11-09: Continued work on the design portfolio
- ☑ 11-16: Sent Clem design portfolio draft
- ☑ 11-30: Turn in design portfolio

Spring Semester

- ☑ 1-20: Meet to review final design specifications (Reduce specifications to stories)
- ☑ 1-27: Set up version control and testing pipeline
- ☑ 2-3: Begin agile development cycle toward prototype
- ☑ 2-10: Sprint meeting, work on code
- ☑ 2-17: Sprint meeting, work on code
- ☑ 2-24: Sprint meeting, work on code
- ☑ 3-2: Sprint meeting, work on code
- ☑ 3-9: Codebase review, deploy first prototype online
- ☑ 3-16: Sprint meeting, work on code
- ☑ 3-23: Sprint meeting, work on code
- ☑ 3-30: Sprint meeting, work on code
- ☑ 4-6: Sprint meeting, work on code
- ☑ 4-13: Codebase review, deploy second prototype online
- ☑ 4-20: Stress testing and bug fixing
- ☑ 4-24: Deploy final application
- ☑ 5-4: Work on final presentation
- ☑ 5-11: Work on final presentation
- ☑ 5-18: Work on final presentation
- ☑ 5-24: Give final presentation



Functional Requirements

User Story	Functional Requirement	Priority
As a user, I want to have the application be highly responsive and functional.	Written with a language that includes robust front end programming	High
As a user, I need to be able to keep track of my information after I log out of my account.	Must save user credentials	High
As an educator, I want to be able to set up a class group to look at students together.	Data structure for organizing Groups	Low
As a RedTrade user, I must be able to set up periodic deposits to my account to simulate actual trading practices.	Periodic investing on backend	Medium
As a user, I want to be able to see historical data about stocks.	Must have API to pull historical stock data	High
As a RedTrade user, I must be able to choose the amount of money I want to start trading with.	Be able to choose how much money you invest with	Medium
As an investor, I want to be able to learn about active investing through lessons that engage me.	Have a lesson section	Low

User Story	Functional Requirement	Priority
As a user, I want to be able to see how my money would grow through different strategies based on real-world data.	Need to have a way to hold information after a user goes offline	Medium

Non-Functional Requirements

User Story	Non-functional Requirement	Priority
As a user, I want to have clear, understandable user interfaces to learn and invest.	Needs to have clean visuals	High
As a user, I want a way to educate myself on basic terms relating to the stock market.	Needs to have simple explanations	High
As an investor, I need to be able to keep my financial data secure.	Needs to store user information securely	High
As a user, I want to use a website that is highly responsive and functional.	Website needs to be dynamic	Moderate

Performance requirement

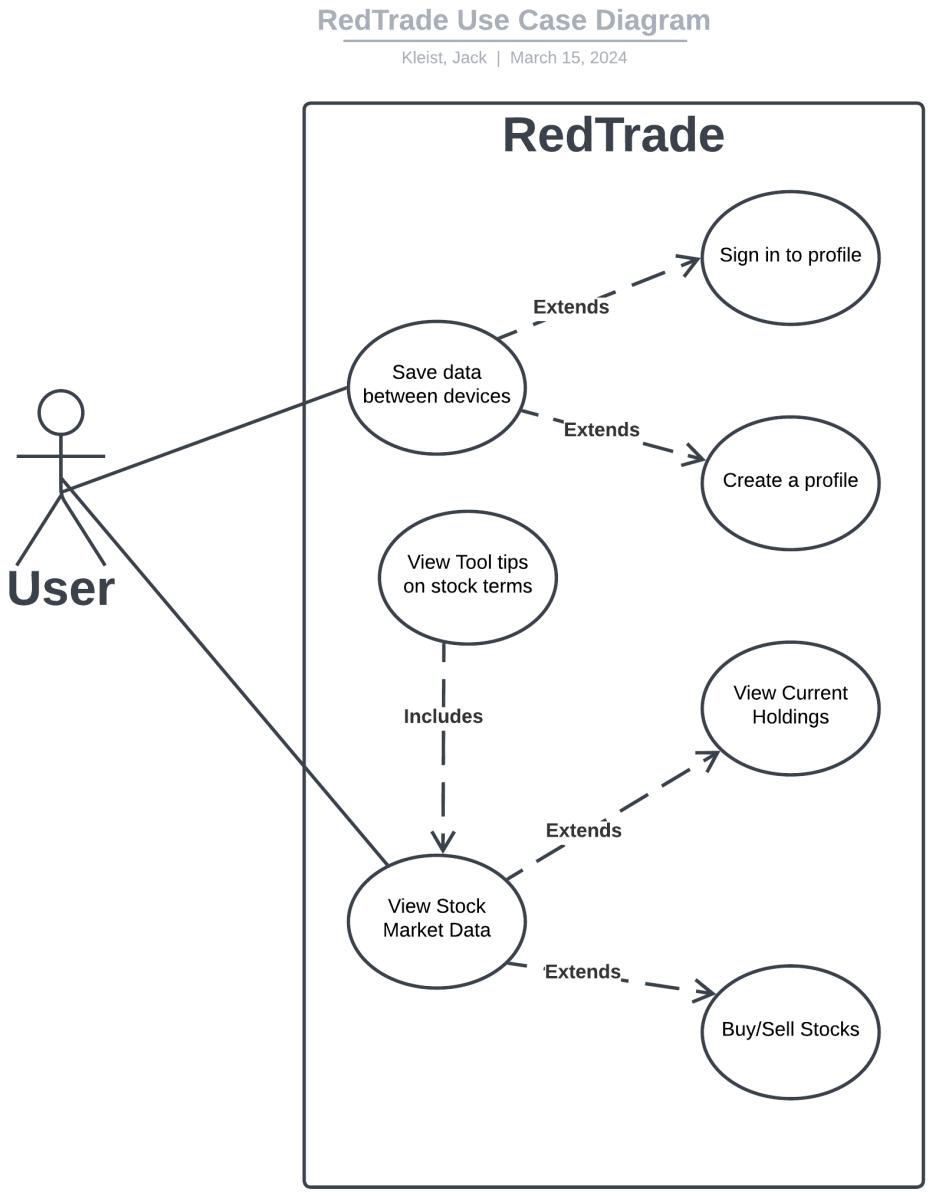
Since RedTrade will be a web-hosted application, the performance requirements are mainly tied to network capabilities. According to Google developers, the speed at which your website loads is key to being promoted by Google's algorithm. (*"Speed Index"*) With Google being one of the most used search engines, being able to compete with bigger companies through improved metrics is key. We also want to ensure a timely First Contentful Paint (FCP) to ensure users have a visual reference letting them know the page is loading. (Walton, 2019). We will also be referencing the research published in 2013 by Manhas, J. in the Journal of Computer Science and Engineering about factors involved with fast content loading when developing our application to ensure that we are taking a scientific approach to our design.

Interface Requirements

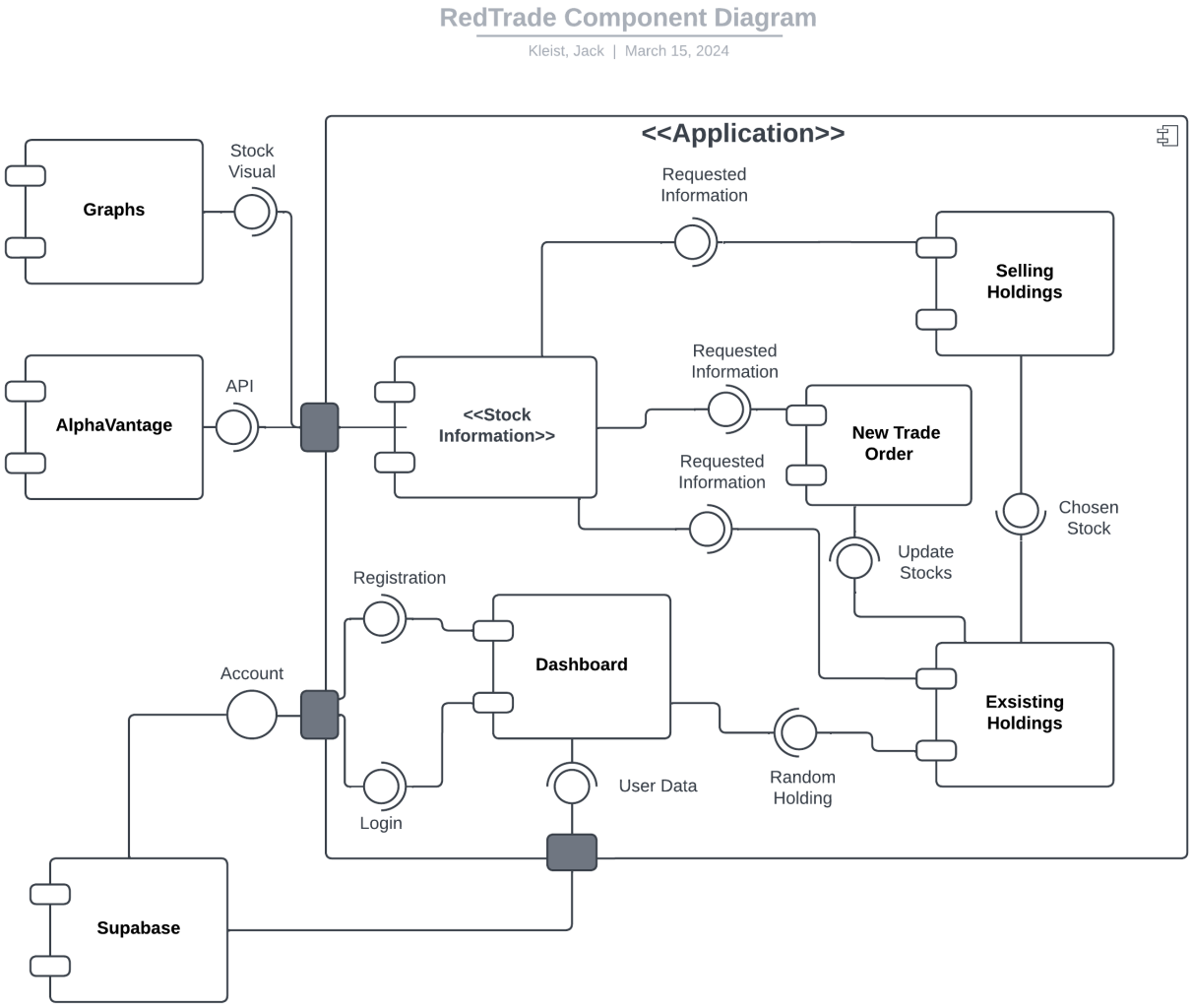
When designing RedTrade's interface, our top priority is ensuring its accessible and user-friendly, even to those who may not be tech-savvy. This means RedTrade's design will be intuitive, uncluttered, and forgiving for users who are making their first entrance to trading. The platform will use clear language throughout and present complex information a digestible manner. This will be done with visuals and step-by-step guides. The layout will be structured so navigation feels natural, with core features like stock searching, buying, and selling being front and center, and any potentially confusing features accompanied by helpful tips and explanations. Additionally, the interface will be built responsively, adapting seamlessly to various devices with the help of React Native. Interactive, understandable charts will be a cornerstone of the RedTrade interface, designed to provide users with detailed market insights while being simple enough to understand immediately. Lastly, we will use a color palette to favor readability and implement interactive elements, like tooltips, to reinforce learning.

Architectural Requirements

Use Case Diagram



Component Diagram



Development Standards

ISO/IEC/IEEE 223026:2023

The tools we will use in our technology stack are BitBucket for version control, React Native with javascript as the front-end framework, and Supabase for the backend. Security is a strong focus so we will comply with both ISO and NIST CSaG encryption guidelines. We will also be utilizing a tool called Synk to do automated code inspections within the development cycle to assist in finding security vulnerabilities.

Testing is another standard we will measure with unit-tests and end-to-end testing. Unit tests will be treated separately between the front-end and backend, while end-to-end tests will incorporate the full stack and make sure the code compiles and builds as expected.

Methodology

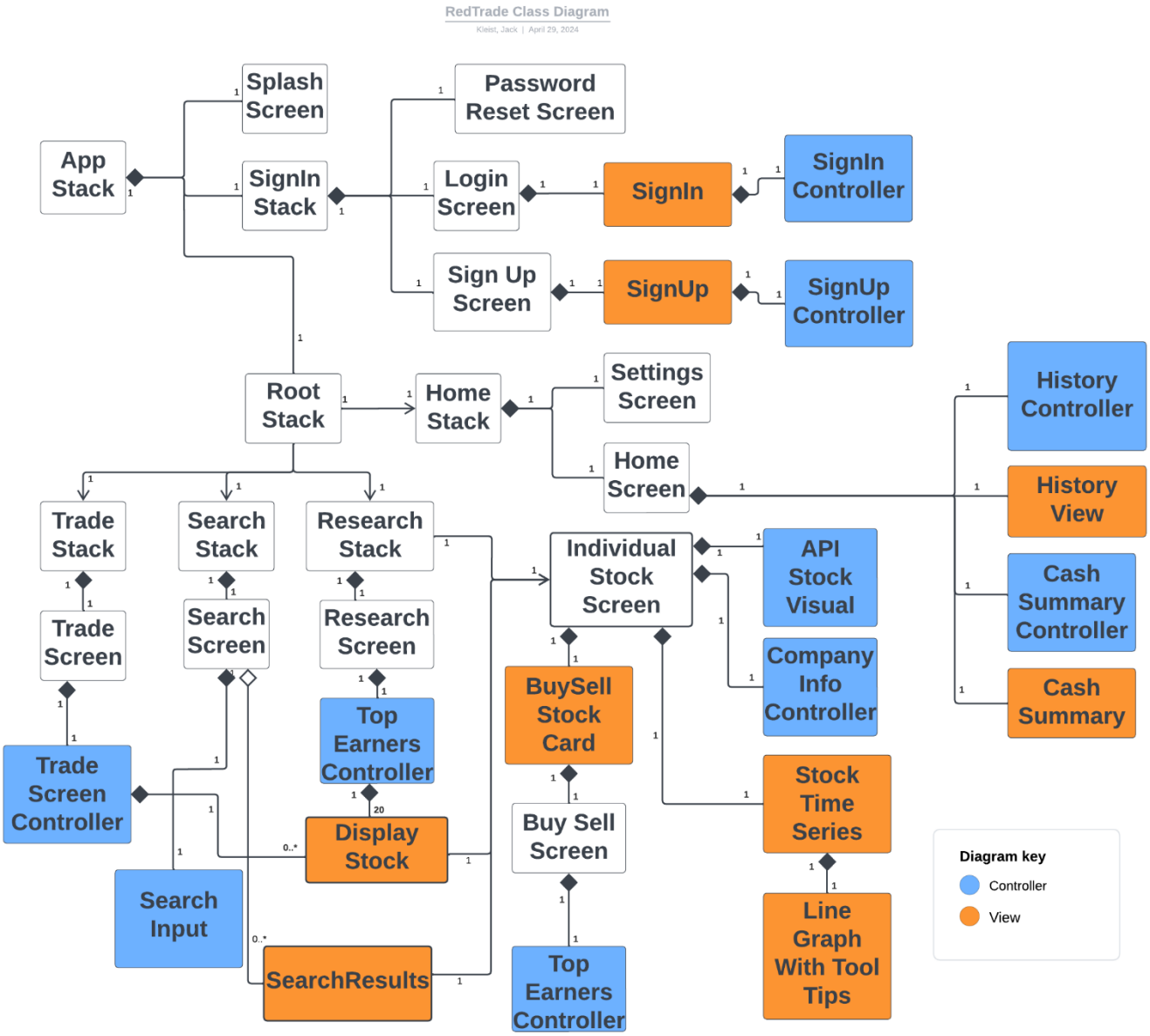
Opting for an Agile methodology in the development of RedTrade is a strategic decision aimed at maximizing flexibility and adaptability throughout the application's lifecycle. This approach allows our team to efficiently address emerging issues and enables us to make on-the-fly adjustments to the software, eliminating the need to start over. The incorporation of regular stand-up meetings, scheduled on a weekly or daily basis, establishes a core communication structure, enhancing our ability to collaborate and ensuring everyone is on the same page. Moreover, the Agile frameworks are instrumental in maintaining a balanced workload, preventing burnout by allowing the team to navigate project responsibilities alongside other commitments. The agile approach positions us to navigate the dynamic landscape of development with ease and confidence.

Design Patterns

In the exploration of design patterns for a web-based application, two prominent candidates emerged as particularly fitting: the Observer pattern and the Model View Component (MVC) pattern. The Observer pattern presents itself as a valuable choice, especially in the context of numerous graphs depicting stock ticker information. By implementing the Observer pattern, a single notifier class could efficiently liaise with our stock ticker API, disseminating results seamlessly to all observing components. This approach not only simplifies the system by consolidating API key management into a singular entity but also enhances maintainability and reduces overall complexity. (Freeman & Robson, 2014)

Alternatively, the MVC pattern is a standard architecture model for web application development. It allows us to separate the objects interacting with API and network calls and the object displaying information to the user. We modified the traditional model in favor of a model that fits well with React native, which consists only of the Controller and View. This allows us to

scale and add new pieces and components more easily, as every time we add a new feature, we need to add both the view component and the view component.



Design Trade Offs

When deciding what technologies to use for RedTrade there was a long list of options considered. Ultimately, we decided to harness the flexibility of React Native for the front end. This decision addressed our need for a modern, responsive user interface that could adapt to different devices effortlessly. React Native, one of the most popular platforms for cross-platform development, will allow for the creation of a dynamic, engaging experience for users new to trading, making the complex world of finance simple and interactive. Some other options that were investigated were Flutter and Svelte. Flutter would have been an alternative, flexible, cross-platform tool but was decided against due to its smaller community and requirement of the Dart programming language. Svelte is a simple and flexible framework for the web with built-in state management but can't be built to a mobile app, a necessary requirement to be considered an accessible app.

The backend is where security is most important. Consequently, we've chosen Supabase, an open source alternative to Firebase. Supabase provides a free, cloud-based database solution that is perfect for managing user profiles and transactional data with the capability for expansion as needed. A competitor that we could have gone with for database storage in a page based format is MongoDB's Atlas. We decided against this as we felt that the information that we needed to store made more sense being stored in a traditional database. Supabase's comprehensive toolset will help safeguard user data and allow for expansion and optimization of our application. Utilizing Supabase means RedTrade benefits from industry standard data encryption, network security, and identity access management. This will keep our user's personal information under strict protection. Altogether, RedTrade strikes a balance between a highly secure, scalable backend and smooth, accessible frontend experience.

References

Parker, Kim. "More than Half of U.S. Households Have Some Investment in the Stock Market." *Pew Research Center*, Pew Research Center, 25 Mar. 2020, www.pewresearch.org/short-reads/2020/03/25/more-than-half-of-u-s-households-have-some-investment-in-the-stock-market/.

Scott, Gordon. “How to Use the Investopedia Simulator.” *Investopedia*, Investopedia, 7 Mar. 2023, www.investopedia.com/how-to-use-the-investopedia-simulator-5221184.

“Stock Market Simulator: Practice Trading for Free.” *Wall Street Survivor*, 25 Sept. 2023, www.wallstreetsurvivor.com/.

“Virtual Stock Exchange.” *MarketWatch*, www.marketwatch.com/games. Accessed 28 Nov. 2023.

“Speed Index.” *Chrome for Developers*, Google, developer.chrome.com/en/docs/lighthouse/performance/speed-index/. Accessed 28 Nov. 2023.

Philip, Walton. “First Contentful Paint (FCP)” *Web.Dev*, 7 Nov. 2019, web.dev/articles/fcp.

Manhas, Dr. (2013). “A Study of Factors Affecting Websites Page Loading Speed for Efficient Web Performance.”, International Journal of Computer Sciences and Engineering. 30 Nov. 2013, Volume one, Issue 3

“ISO/IEC/IEEE 223026:2023.” *ISO*, July 2023, www.iso.org/obp/ui/en/#iso:std:iso-iec-ieee:23026:ed-2:v2:en.

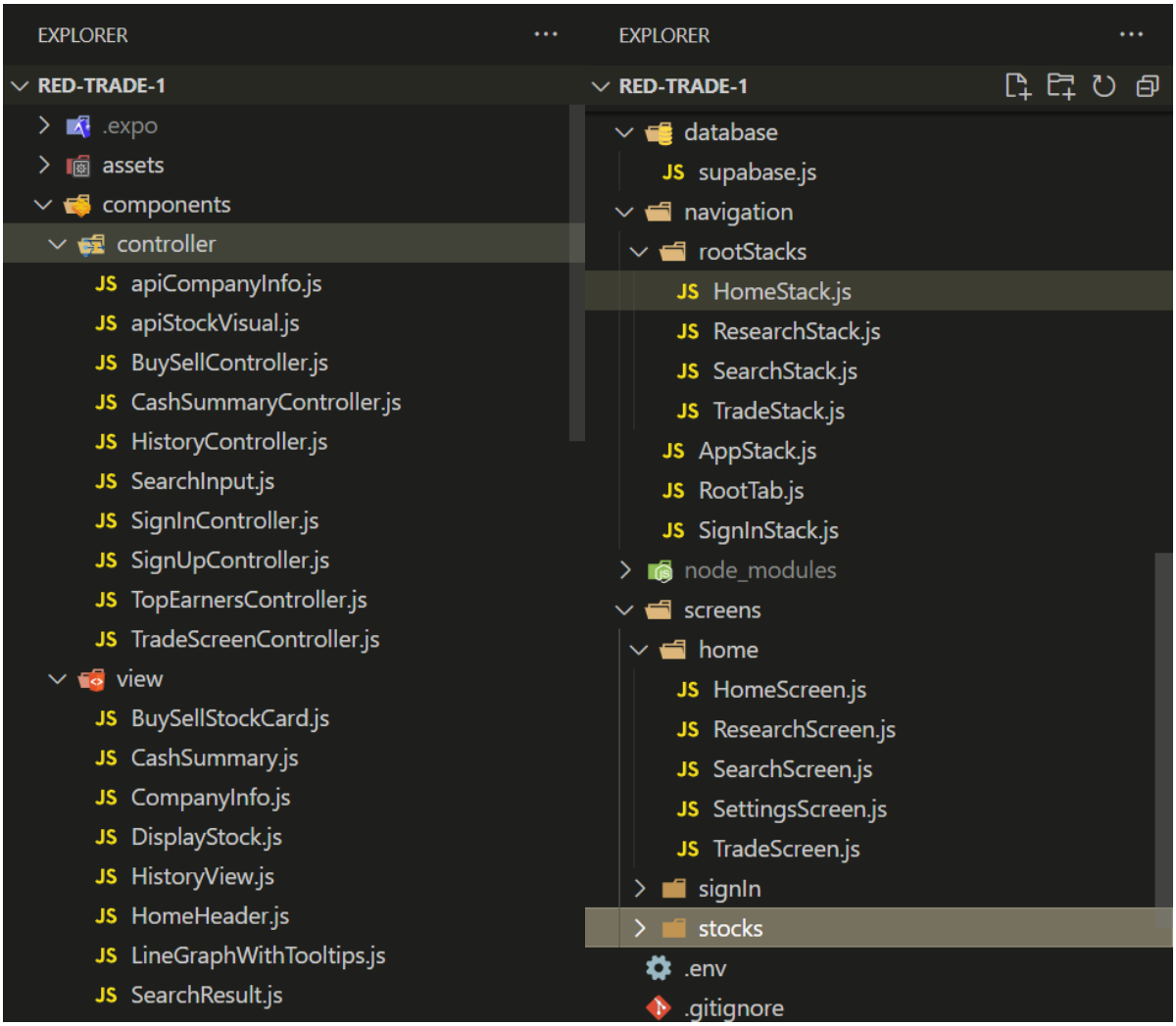
Computer Security Division, Information Technology Laboratory. “Cryptographic Standards and Guidelines: CSRC.” *CSRC*, US Government, csrc.nist.gov/projects/cryptographic-standards-and-guidelines. Accessed 28 Nov. 2023.

Freeman, Eric, and Elisabeth Robson. *Head First Design Patterns: A Brain-Friendly Guide*. O’Reilly, Edition: 10th Anniversary Ed., 2014.

Large Language Models were used in the refinement of this paper.

Source Code

Folder Structure



App.js

```
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';
import { DefaultTheme, DarkTheme, NavigationContainer } from '@react-
navigation/native';
import 'react-native-gesture-handler';
import AppStack from './navigation/AppStack';

export default function App() {
  return (
    <NavigationContainer theme={DarkTheme}>
      <AppStack />
    </NavigationContainer>
  );
```

```
}
```

Database

Supabase.js

```
import 'react-native-url-polyfill/auto'
import AsyncStorage from '@react-native-async-storage/async-storage'
import { createClient } from '@supabase/supabase-js'
import * as SecureStore from 'expo-secure-store';
import * as aesjs from 'aes-js';
import 'react-native-get-random-values';
import { AppState } from 'react-native'



// As Expo's SecureStore does not support values larger than 2048
// bytes, an AES-256 key is generated and stored in SecureStore, while
// it is used to encrypt/decrypt values stored in AsyncStorage.
class LargeSecureStore {
  async _encrypt(key, value) {
    const encryptionKey = crypto.getRandomValues(new Uint8Array(256 / 8));

    const cipher = new aesjs.ModeOfOperation.ctr(encryptionKey, new
aesjs.Counter(1));
    const encryptedBytes = cipher.encrypt(aesjs.utils.utf8.toBytes(value));

    await SecureStore.setItemAsync(key,
aesjs.utils.hex.fromBytes(encryptionKey));

    return aesjs.utils.hex.fromBytes(encryptedBytes);
  }

  async _decrypt(key, value) {
    const encryptionKeyHex = await SecureStore.getItemAsync(key);
    if (!encryptionKeyHex) {
      return encryptionKeyHex;
    }

    const cipher = new
aesjs.ModeOfOperation.ctr(aesjs.utils.hex.toBytes(encryptionKeyHex), new
aesjs.Counter(1));
    const decryptedBytes = cipher.decrypt(aesjs.utils.hex.toBytes(value));

    return aesjs.utils.utf8.fromBytes(decryptedBytes);
  }
```

```
  async getItem(key) {
    const encrypted = await AsyncStorage.getItem(key);
    if (!encrypted) { return encrypted; }

    return await this._decrypt(key, encrypted);
  }

  async removeItem(key) {
    await AsyncStorage.removeItem(key);
    await SecureStore.deleteItemAsync(key);
  }

  async setItem(key, value) {
    const encrypted = await this._encrypt(key, value);

    await AsyncStorage.setItem(key, encrypted);
  }
}
const supabaseUrl = 'https://rlvczrlioajeayefaphs.supabase.co'
const supabaseAnonKey =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InJsdmN6cmx
pb2FqZWF5ZWZhcGhzIiwicm9sZSI6ImFub24iLCJpYXQiOjE3MDc0MTUyMDYsImV4cCI6MjAyMjk5MTIw
Nn0.9rHqByotdO2NNI7I6EhjZt3IdwP3V6s8dJ5nrfG3Ltc'


export const supabase = createClient(supabaseUrl, supabaseAnonKey, {
  auth: {
    storage: new LargeSecureStore(),
    autoRefreshToken: true,
    persistSession: true,
    detectSessionInUrl: false,
  },
});

// Tells Supabase Auth to continuously refresh the session automatically if
// the app is in the foreground. When this is added, you will continue to receive
// `onAuthStateChange` events with the `TOKEN_REFRESHED` or `SIGNED_OUT` event
// if the user's session is terminated. This should only be registered once.
AppState.addEventListener('change', (state) => {
  if (state === 'active') {
    supabase.auth.startAutoRefresh()
  } else {
    supabase.auth.stopAutoRefresh()
  }
```

```
})
```

Components/Controllers

ApiCompanyInfo.js

```
import { useEffect, useState } from "react";
import { View,StyleSheet } from "react-native"
import CompanyInformation from "../view/CompanyInfo";




export default function CompanyInfoController({symbol}){

    const[result,setResult] = useState(null);
    // calls the API to retrieve information on the stock

    useEffect(() => {
        companyInfoAPI()
      },[symbol]);

      async function companyInfoAPI() {
        const alphaUrl =
          process.env.EXPO_PUBLIC_ALPHAVANTAGE_URL +
          "function=OVERVIEW&symbol=" +
          symbol +
          "&apikey=" +
          process.env.EXPO_PUBLIC_ALPHAVANTAGE_KEY;

        try {
          const response = await fetch(alphaUrl);
          const resultsJson = await response.json();
          setResult(resultsJson);
          console.log(resultsJson); // Moved console.log inside to log the
updated state
        } catch (error) {
          console.error(error);
        }
      }



    return(
        <View style={styles.container}>
            <CompanyInformation results={result}/>
```

```
        </View>
    )


}

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

ApiStockVisual.js

```
import React, { useEffect, useState } from "react";
import { View, Text, StyleSheet} from "react-native";
import StockTimeSeries from "../view/StockTimeSeries";
import { useNavigation } from "expo-router";

   export async function symbolApi(symbol, name, lastPrice, hasTimeSeriesInfo,
result) {
    try {
      const alphaUrl =
        process.env.EXPO_PUBLIC_ALPHAVANTAGE_URL +
        'function=TIME_SERIES_INTRADAY&symbol=' +
        symbol +
        '&interval=15min&apikey=' +
        process.env.EXPO_PUBLIC_ALPHAVANTAGE_KEY;

      const response = await fetch(alphaUrl);
      const resultsJson = await response.json();
      var timeSeriesData = resultsJson["Time Series (15min)"];
      if(typeof timeSeriesData === 'undefined'){
        hasTimeSeriesInfo(false)
      } else {
        hasTimeSeriesInfo(true);
        result(timeSeriesData);
        const lastEntryKey =
Object.keys(timeSeriesData)[Object.keys(timeSeriesData).length - 1];
        console.log(lastEntryKey)
       console.log(Object.keys(timeSeriesData).length)
        const lastEntry = +(timeSeriesData[lastEntryKey]['4. close']);
```

```
        console.log('Last entry log:'+lastEntry);
        lastPrice(lastEntry)
      }
    } catch (error) {
      console.error(error);
    }
  }


const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

BuySellController.js

```
import moment from "moment";
import { supabase } from "../../database/supabase";
import { Alert } from "react-native";
import { useState } from "react";
import { symbol } from "prop-types";
export async function getCurrentHoldings(holdings, ticker) {
    try {
        const { data: { user } } = await supabase.auth.getUser()
        const { data: holding, error: holdingerror } = await supabase
            .from('holdings')
            .select(
                'shares'
            ).eq('user_id', user.id)
            .ilike('symbol', ticker)
        console.log(holding)
        if (holding.length === 0) {
            console.log("holding is undefined")
            holdings(-1)
        } else {
            holdings(holding[0].shares)
        }
        if (holdingerror) {
            throw error
        }
    }
```

```
    catch (error) {
        if (error instanceof Error) {
            Alert.alert(error.message)
        }
    }
}


export async function PlaceBuyOrSell(action, ticker, shares, amount, executing,
price, current_holdings) {
    //Made a switch statement to allow for potential other options with stock in
the future
    executing(true);
    switch (action) {
        case 'buy':
            //TODO: Add supabase call to buy amount of stock with funds
            try {
                const { data: { user } } = await supabase.auth.getUser()
                //if user does not have the stock, add it to their holdings
                if (current_holdings < 0) {
                    const { data: insert, error: seconderror } = await supabase
                        .from('holdings')
                        .insert([{
                            shares: +shares,
                            user_id: user.id,
                            symbol: ticker,
                            last_price: price,
                            last_update: new moment()
                        },]
                        ).select()
                    console.log(insert)
                    if (seconderror) {
                        console.error("Error inserting data:",
seconderror.message);
                        throw error;
                    }
                    //if user has the stock, update their holdings and the last
purchased price
                } else {
                    var newShares = parseInt(current_holdings,10) +
parseInt(shares,10)
                    console.log(newShares)
                    const { data: update, error: updateError } = await supabase
                        .from('holdings')
                        .update([{
```

```
                            shares: newShares,
                            last_price: price,
                            last_update: new moment()
                        },]).eq('user_id', user.id)
                        .ilike('symbol', ticker)
                        .select()
                    console.log(update)
                    if (updateError) {
                        console.error("Error inserting data:",
updateError.message);
                        throw error;
                    }
                }

                const { data: money, error: moneyError } = await supabase
                    .from('profiles')
                    .select(
                        'current_funds'
                    ).eq('id', user.id)
                if (moneyError) {
                    console.error("Error inserting data:", moneyError.message);
                    throw error;
                }


                console.log("Current funds: " + money[0].current_funds)
                var newMoney = ((+money[0].current_funds - amount).toFixed(2))
                if(newMoney<0){
                    newMoney=0;
                }
                const { data: cashUpdate, error: cashUpdateError } = await
supabase
                    .from('profiles')
                    .update([{
                        current_funds: newMoney,
                        updated_at: new moment()
                    },]).eq('id', user.id)
                    .select()
                console.log(cashUpdate)
                if (cashUpdateError) {
                    console.error("Error inserting data:",
cashUpdateError.message);
                    throw error;
                }
```

```
                const { data: transaction, error: trasnactionError } = await
supabase
                        .from('transactions')
                        .insert([{
                            symbol: ticker,
                            buy: 1,
                            stock_amount: shares,
                            price: price,
                            user: user.id
                        },]
                        ).select()
                        console.log(transaction)
            } catch (error) {
                if (error instanceof Error) {
                    Alert.alert(error.message)
                }
            }
            console.log(`Bought ${shares} share(s) of ${ticker} for ${amount}`)

            break
        case 'sell':
            //TODO: update supabase call to sell amount of stock with funds
            try {
                // if (holding[0] === undefined) {
                //     const { data: insert, error: seconderror } = await
supabase
                //         .from('holdings')
                //         .insert([{
                //             shares: shares,
                //             user_id: user.id,
                //             symbol: ticker,
                //             last_price: price,
                //             last_update: new moment()
                //         },]
                //         ).select()
                //     console.log(insert)
                //     if (seconderror) {
                //         console.error("Error inserting data:",
seconderror.message);
                //         throw error;
                //     }
                //     //if user has the stock, reduce the ammount of stock that
they have by sold ammount
                // } else {
                const { data: { user } } = await supabase.auth.getUser()
```

```
                var newShares = parseInt(current_holdings,10) -
parseInt(shares,10)
                console.log(newShares)
                const { data: update, error: updateError } = await supabase
                    .from('holdings')
                    .update([{
                        shares: newShares,
                        last_price: price,
                        last_update: new moment()
                    },]).eq('user_id', user.id)
                    .ilike('symbol', ticker)
                    .select()
                console.log(update)
                if (updateError) {
                    console.error("Error inserting data:", updateError.message);
                    throw error;
                }

                const { data: money, error: moneyError } = await supabase
                    .from('profiles')
                    .select(
                        'current_funds'
                    ).eq('id', user.id)
                if (moneyError) {
                    console.error("Error getting money error data:",
moneyError.message);
                    throw error;
                }

                console.log("Current funds: " + money[0].current_funds)
                var newMoney = +money[0].current_funds + amount;
                const { data: cashUpdate, error: cashUpdateError } = await
supabase
                    .from('profiles')
                    .update([{
                        current_funds: newMoney,
                        updated_at: new moment()
                    },]).eq('id', user.id)
                    .select()
                console.log(cashUpdate)
                if (cashUpdateError) {
                    console.error("Error inserting data:",
cashUpdateError.message);
                    throw error;
                }
```

```
                const { data: transaction, error: trasnactionError } = await
supabase
                .from('transactions')
                .insert([{
                    symbol: ticker,
                    sell: 1,
                    symbol: ticker,
                    stock_amount: shares,
                    price: price,
                    user: user.id
                },]
                ).select()
            } catch (error) {
                if (error instanceof Error) {
                    Alert.alert(error.message)
                }
            }
            console.log(`Sold ${shares} share(s) of ${ticker} for ${amount}`)
            break
    }
    executing(false);
}
```

CashSummaryController.js

```
import React, { useState,useEffect } from 'react'
import { Alert, StyleSheet, View, AppState, Switch, Text} from 'react-native'
import { Button, Input } from 'react-native-elements'
import {supabase} from '../../database/supabase'
import {Picker} from '@react-native-picker/picker'
import CashSummary from '../../components/view/CashSummary'


  export async function getFunds(loading,funds,name) {
  try{
    const { data: { user } } = await supabase.auth.getUser()
    const { data,error } = await supabase
    .from('profiles')
    .select(
      'current_funds,first_name'
    ).eq('id',user.id)
        funds(+data[0].current_funds.toFixed(2))
        name(data[0].first_name)
        loading(false)
    if (error) {
```

```
      throw error
    }
  } catch (error) {
    if (error instanceof Error) {
      Alert.alert(error.message)
    }
  }
}
```

HistoryController.js

```
import { useState } from "react"
import { supabase } from "../../database/supabase";


export async function getHistory(historyInfo,loading) {
    console.log("getting history")
    try {
        const { data: { user } } = await supabase.auth.getUser()
        const { data: history, error: historyError } = await supabase
            .from('transactions')
            .select('*')
            .eq('user', user.id)
            .order('created_at', { ascending: false })
            .limit(10)
        historyInfo(history);
        console.log("made it the history")
        if(historyError){
            console.log(historyError)
        }
    } catch(historyError){
        console.error(historyError)
    }
}
```

SearchInput.js

```
import { useState, useEffect } from "react"
import { StyleSheet, Text, View, Button, Pressable } from 'react-native';
import { Input } from 'react-native-elements'
import Ionicons from '@expo/vector-icons/Ionicons';

import { TextInput } from 'react-native-gesture-handler';
```

```
function SearchInput({ onSearch }) {
    const [apikey, setApikey] =
useState(process.env.EXPO_PUBLIC_ALPHAVANTAGE_KEY)
    const [symbol, setSymbol] = useState('')
    const [loading, setLoading] = useState(false)

    async function searchAPI() {
        setLoading(true)
        alphaUrl = process.env.EXPO_PUBLIC_ALPHAVANTAGE_URL
        alphaUrl = alphaUrl + 'function=SYMBOL_SEARCH&keywords=' + symbol +
'&apikey=' + apikey
        await fetch(alphaUrl)
            .then((res) => res.json())
            .then((resultsJson) =>
                onSearch(resultsJson.bestMatches)
            )
            .catch((error) => console.error(error))
        setLoading(false)
    }

    return (
        <View style={styles.container}>
            <View style={styles.leftContainer}>
                <TextInput style={styles.input} placeholder="Search Ticker"
defaultColor='white'
                    keyboardType="default" onChangeText={(text) =>
setSymbol(text)} leftIcon={{ type: 'font-awesome', name: 'envelope', color:
'black' }}
                    activeOutlineColor="white"
                    placeholderTextColor='rgb(232,232,232)'
                ></TextInput>
            </View>

            <Pressable disabled={loading} onPress={() => searchAPI()}
style={[styles.button, styles.primaryButton]}>
                <Ionicons name="search" size={20} color='rgb(48,48,48)' />
            </Pressable>
        </View>
    );
}
export default SearchInput;

const styles = StyleSheet.create({
    container: {
```

```
        alignItems: 'center',
        justifyContent: "space-between",
        flexDirection: "row"
    },
    leftContainer:{
        flex:3,
        height:48,
    },
    input: {
        borderWidth: 1,
        height:48,
        padding: 10,
        backgroundColor: "rgb(26,26,26)",
        fontSize:15,
        color:'rgb(215,215,215)',
        borderRadius: 6,
        flex: 1,
        marginTop: 10,
        marginLeft: 5,
        marginRight:16,
    },
    button: {
        flex: 1,
        paddingVertical: 6,
        borderRadius: 6,
        marginTop: 10,
        textAlign: "center",
        alignItems: 'center',
        justifyContent: 'center'
    },
    buttonText: {
        fontSize: 14,
        fontWeight: '700',
        color: "rgb(35,35,35)",
    },
    primaryButton: {
        backgroundColor: "rgb(235,235,235)",
        borderColor: "rgb(235,235,235)",
        borderWidth: 3,
        marginRight: 5,
    },

});
```

SignInController.js

```
import { supabase } from "../../database/supabase"
import { Alert } from "react-native"

export async function signInWithEmail(email,password) {
  console.log(email)
    const { error } = await supabase.auth.signInWithPassword({
      email: email,
      password: password,
    })
    if (error) Alert.alert(error.message)
  }
```

SignUpController.js

```
import { Alert} from 'react-native'
import {supabase} from '../../database/supabase'



export async function
signUpWithEmail(email,password,firstName,lastName,currentFunds) {
    const {
        data: { session },
        error,
    } = await supabase.auth.signUp(
        {
            email: email,
            password: password
        }
    )
    if (error) Alert.alert(error.message)
    //else if (!session) Alert.alert('Please check your inbox for email
verification!')
    try {
        const updates = {
            id: session?.user.id,
            first_name: firstName,
            last_name: lastName,
            current_funds: currentFunds,
            completed_signup: 0,
            updated_at: new Date()
        }
        const { data, error: second } = await supabase.from('profiles').update(
```

```
            updates
        ).eq('id', session?.user.id).select()
        console.log(data)
        if (second) {
            throw second
        }
    } catch (second) {
        if (error instanceof Error) {
            Alert.alert(second.message)
        }
    } finally {
    }

}
```

TopEarnersController.js

```
import { useEffect, useState } from "react";
import { View, StyleSheet, ActivityIndicator } from "react-native"
import CompanyInformation from "../view/CompanyInfo";
import DisplayStock from "../view/DisplayStock";
import { supabase } from "../../database/supabase";
import moment from "moment";




export default function TopEarnersController({date}) {

  const [result, setResult] = useState(null);
  const [lastDate, setLastDate] = useState(null);
  // calls the API to retrieve information on the stock

  useEffect(() => {
    topEarnersAPI()
  }, [""]);
  //TODO when api call is working, update this code to make it save the
information to database
  async function topEarnersAPI() {
    var hasData = false
    //requests data from 'top_gainers" from the remote database
    let { data: firstRequest, error } = await supabase
      .from('top_gainers')
      .select()
      .order('created_at',{ascending:false})
```

```
      .limit(1)
    if (error){ console.error(error)}
    if(firstRequest[0]!== undefined){
      hasData = true
      console.log("thinks has data is: "+hasData)
      created_at_moment = new moment(firstRequest[0].created_at)
    }
    current_date = new moment()
    if(hasData&&current_date.isSame(created_at_moment, 'day')) {
        console.log("first request: " + firstRequest[0].created_at)
      //if the latest date from the database is the same as the current day, use
that data for display
        date(firstRequest[0].created_at)
        setResult(firstRequest[0].top_volume)

    }
    //else, query the API to get the current information
    else {
      const alphaUrl =
        process.env.EXPO_PUBLIC_ALPHAVANTAGE_URL +
        "function=TOP_GAINERS_LOSERS&apikey=" +
        process.env.EXPO_PUBLIC_ALPHAVANTAGE_KEY;

      try {
        //send request to alphavantage api
        const response = await fetch(alphaUrl);
        const resultsJson = await response.json();
        //if the json length is not 0, i.e. there are responses
        if (Object.keys(resultsJson).length !== 0) {
          setResult(resultsJson["most_actively_traded"])
          console.log("Recognized that results were not 0")
          const created_at = new moment().format("YYYY/MM/DD")
          let { data, error } = await supabase
            .from('top_gainers')
            .upsert({
              created_at: created_at,
              top_gainers: resultsJson["top_gainers"],
              top_losers: resultsJson["top_losers"],
              top_volume: resultsJson["most_actively_traded"]
           })
            .select()
          if (error) console.error(error)
          else {
            console.log(data)
            date(created_at)
```

```
          }
        //if there is no response, get the last
        } else {
          let { data: secondRequest, error } = await supabase
          .from('top_gainers')
          .select()
          .order('created_at')
          .limit(1)
          if (error){ console.error(error)}
          else{
            date(secondRequest[0].created_at)
            setResult(secondRequest[0].json_data)
          }
        }
      } catch (error) {
        console.error(error);
      }
    }

  }




  return (
    <View style={styles.container}>{
      !result?<View>
        <ActivityIndicator size='large'/>
      </View> :

      result.map((result) =>
        <View style={styles.resultContainer} key={result["ticker"]}>
          <DisplayStock symbol={result["ticker"]} price={+(result["price"])}
origin={'TopEarners'} />
        </View>
      )
    }
    </View>
  )


}

const styles = StyleSheet.create({
```

```
  container: {
    flex: 1,
    paddingVertical:16,
  },
  searchContainer: {
    marginBottom: 16,
  },
  resultContainer: {
    marginVertical: 6,
  },
});
```

TradeScreenController.js

```
import React, { useState, useEffect } from 'react'
import { Alert, StyleSheet, View, AppState, Switch, Text, ActivityIndicator }
from 'react-native'
import { Button, Input } from 'react-native-elements'
import { supabase } from '../../database/supabase'
import CashSummary from '../../components/view/CashSummary'
import DisplayStock from '../view/DisplayStock'
import { useNavigation } from '@react-navigation/native'
import { getHistory } from './HistoryController'

export default function TradeScreenController() {
  const [loading, setLoading] = useState(false)
  const [stocks, setStocks] = useState(null)
  nav = useNavigation();
  useEffect(()=> {
    const unsubscribe = nav.addListener('focus', () => {
        // Do something when the screen is focused
        // This could be fetching data, updating state, etc.
        getStocks()
      });

      // Return cleanup function to unsubscribe from event listener
      return unsubscribe;
},[nav])



  async function getStocks() {
    try {

      const { data: { user } } = await supabase.auth.getUser()
```

```
      const { data, error } = await supabase
        .from('holdings')
        .select(
          'symbol, shares, last_price'
        ).eq('user_id', user.id)
      setStocks(data)
      if (error) {
        throw error
      }
    } catch (error) {
      if (error instanceof Error) {
        Alert.alert(error.message)
      }
    }
  }
  if (stocks === null) {
    return (
      <View style={styles.container}>
        <ActivityIndicator />
      </View>
    )
  }
  else if(Object.keys(stocks).length === 0) {
    return(
    <View style={styles.container}>
      <Text style={styles.resultContainer}>No stocks held</Text>
    </View>)
  }
  else {
  return (
    <View style={styles.container}>
      {stocks.filter(result => result.shares >=1).map((result) =>
        <View style={styles.resultContainer} key={result["symbol"]}>
          <DisplayStock shares={result.shares} symbol={result.symbol}
price={result.last_price}/>
        </View>
      )}
    </View>
  )
}
}
const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
```

```
  },
  searchContainer: {
    marginBottom: 16,
  },
  resultContainer: {
    marginVertical: 6,
    color: 'white'
  },
});
```

Components/Views

BuySellStockCard.js

```
import React from 'react';
import { View, Text, Dimensions, StyleSheet, Alert } from 'react-native';
import { useState, useEffect } from 'react';
import LineChartWithTooltips from './LineGraphWithTooltips';
import { useNavigation } from '@react-navigation/native';

const screenWidth = Dimensions.get('window').width;

function StockTimeSeries({ title, chartData }) {
  const nav = useNavigation()
  const [formattedData, setFormattedData] = useState([]);

  function prepareChartData(data) {
    const chartData = [];

    for (const timestamp in data) {
      if (data.hasOwnProperty(timestamp)) {
        const closePrice = parseFloat(data[timestamp]['4. close']).toFixed(2);
        chartData.push({ time: timestamp, value: closePrice });
      }
    }

    // Sort data by timestamp in ascending order
    chartData.sort((a, b) => new Date(a.time) - new Date(b.time));
    console.log("This is inside the prepare function, this is the data:",
chartData)
    return chartData;
  }

  useEffect(() => {
    const fetchData = async () => {
      if (chartData) {
```

```
      const preparedData = prepareChartData(chartData);
      setFormattedData(preparedData);
    }
  };

  fetchData();
}, [chartData]);

if (formattedData?.length === 0) {
  return (
    <View style={styles.container}>
      <Text>Loading...</Text>
    </View>
  );
} else {
  return (
    <View style={styles.container}>
      <Text style={styles.headerText}>{title}</Text>
      <LineChartWithTooltips
        data={{
          labels: formattedData.map((dataPoint) =>
            dataPoint.time
          ),
          datasets: [
            {
              data: formattedData.map((dataPoint) => {
                // Convert the value to a float
                const floatValue = parseFloat(dataPoint.value);

                // Check if the value is a valid float
                if (!isNaN(floatValue)) {
                  return floatValue; // Return the float value
                } else {
                  return null; // Or handle invalid value
                }
              }).filter(value => value !== null)
            },
          ],
        }}
        width={screenWidth}
        height={200}
        yAxisPrefix="$"
        verticalLabelRotation={45}
        chartConfig={{
          backgroundColor: 'black',
```

```
            backgroundGradientFrom: 'rgb(24,24,24)',
            backgroundGradientTo: 'rgb(24,24,24)',
            decimalPlaces: 2,
            color: (opacity = 1) => `rgba(0, 122, 255, ${opacity})`,
            labelColor: (opacity = 1) => `rgba(255, 255, 255, ${opacity})`,
            style: {
              borderRadius: 14,
            },
            propsForDots: {
              r: '0',
              strokeWidth: '2',
              stroke: '#ffa726',
            },
            // propsForLabels: {
            //   angle: -45
            // },
          }}
          bezier
          withVerticalLabels={false}
          withInnerLines={false}
          yAxisLabel="$"
          withOuterLines={false}
          withHorizontalLines={false}
          withVerticalLines={false}
          xLabelsOffset={-15}
        />
      </View>
    );
  }
}


const styles = StyleSheet.create({
  headerText:{
    fontSize:20,
    fontWeight:'500',
    color:'white',
    paddingVertical:12,
    backgroundColor:'rgb(24,24,24)',
    textAlign:'center'
  },
  container: {
  },
});
```

```
export default StockTimeSeries;
```

SignUp.js

```
import React, { useState } from 'react'
import { Alert, StyleSheet, View, AppState, Switch, Text } from 'react-native'
import { Button, Input } from 'react-native-elements'
import { supabase } from '../../database/supabase'
import { Picker } from '@react-native-picker/picker'
import { signUpWithEmail } from '../controller/SignUpController'

export default function SignUp() {
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')
    const [loading, setLoading] = useState(false)
    const [emailError, setEmailError] = useState(false)
    const [firstName, setFirstName] = useState('')
    const [lastName, setLastName] = useState('')
    const [currentFunds, setCurrentFunds] = useState(0.00)

    async function checkEmail() {
        const re = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
        re.test(email) ? setEmailError(false) :
            setEmailError(true)
    }
    return (
        <View style={styles.container}>
            <View style={[styles.verticallySpaced, styles.mt20]}>
                <Input
                    label="Email"
                    leftIcon={{ type: 'font-awesome', name:
'envelope',color:'white' }}
                    onChangeText={(text) => {
                        setEmail(text)
                    }}
                    value={email}
                    placeholder="email@address.com"
                    autoCapitalize={'none'}
                    onBlur={(text) => checkEmail(text)}
                    style={styles.input}
                />
                {emailError && <Text style={[{
                    color: 'red'
                    , fontSize: 12, fontStyle: 'italic', paddingLeft: 12
                }]}>Email is not valid</Text>}
            </View>
```

```
            <View style={styles.verticallySpaced}>
                <Input
                    label="Password"
                    leftIcon={{ type: 'font-awesome', name: 'lock', color:'white'
}}
                    onChangeText={(text) => setPassword(text)}
                    value={password}
                    secureTextEntry={true}
                    placeholder="Password"
                    autoCapitalize={'none'}
                    style={styles.input}
                />
            </View>
            <View style={styles.verticallySpaced}>
                <Input label="First Name" value={firstName || ''}
onChangeText={(text) => setFirstName(text)} style={styles.input}
                />
            </View>
            <View style={styles.verticallySpaced}>
                <Input label="Last Name" value={lastName || ''}
onChangeText={(text) => setLastName(text)} style={styles.input}
                />
            </View>
            <Text style={styles.text}>Starting funds</Text>
            <Picker
                selectedValue={currentFunds}
                onValueChange={(value, index) => setCurrentFunds(value)}
                mode="dropdown" // Android only
                style={styles.picker}
            >
                <Picker.Item label="$100" value={100.00} />
                <Picker.Item label="$500" value={500.00} />
                <Picker.Item label="$1000" value={1000.00} />
                <Picker.Item label="$10000" value={10000.00} />
            </Picker>
            <View style={styles.verticallySpaced}>
                <Button title="Sign up" disabled={loading || emailError}
onPress={() => {
                    setLoading(true)
                    signUpWithEmail(email, password, firstName, lastName,
currentFunds)
                    setLoading(false)
                }} />
            </View>
        </View>
```

```
    )
}


const styles = StyleSheet.create({
    picker:{
        backgroundColor:'rgb(215,215,215)',
        color:'black',
        padding:0,
        marginBottom:16,
    },
    container: {
        marginTop: 40,
        padding: 12,
    },
    verticallySpaced: {
        paddingTop: 4,
        paddingBottom: 4,
        alignSelf: 'stretch'
    },
    mt20: {
        marginTop: 20,
    },
    text:{
        color:'grey'
    },
    input:{
        color:'rgb(215,215,215)',
        fontSize:15,
        paddingLeft:8,
    },
})
```

SignIn.js

```
import React, { useState } from 'react'
import { Alert, StyleSheet, View, AppState, Switch, Text } from 'react-native'
import { Button, Input } from 'react-native-elements'
import { supabase } from '../../database/supabase'
import { Picker } from '@react-native-picker/picker'
import { signInWithEmail } from '../controller/SignInController'

export default function SignIn({ option }) {
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')
    const [loading, setLoading] = useState(false)
```

```
    const [firstName, setFirstName] = useState('')
    const [lastName, setLastName] = useState('')
    const [currentFunds, setCurrentFunds] = useState(0.00)
    return (
        <View style={styles.container}>
            <View style={[styles.verticallySpaced, styles.mt20]}>
                <Input
                    label="Email"
                    leftIcon={{ type: 'font-awesome', name: 'envelope',
color:'white' }}
                    onChangeText={(text) => {
                        setEmail(text)
                    }}
                    value={email}
                    placeholder="email@address.com"
                    autoCapitalize={'none'}
                    style={styles.input}
                />
            </View>
            <View style={styles.verticallySpaced}>
                <Input
                    label="Password"
                    leftIcon={{ type: 'font-awesome', name: 'lock', color:'white'
}}
                    onChangeText={(text) => setPassword(text)}
                    value={password}
                    secureTextEntry={true}
                    placeholder="Password"
                    autoCapitalize={'none'}
                    style={styles.input}
                />
            </View>
            <View style={[styles.verticallySpaced, styles.mt20]}>
                <Button title="Sign in" disabled={loading} onPress={() =>
signInWithEmail(email,password)} />
            </View>
        </View>
    )
}

const styles = StyleSheet.create({
    container: {
        marginTop: 40,
        padding: 12,
    },
```

```
    verticallySpaced: {
        paddingTop: 4,
        paddingBottom: 4,
        alignSelf: 'stretch'
    },
    mt20: {
        marginTop: 20,
    },
    input:{
        color:'rgb(215,215,215)',
        fontSize:15,
        paddingLeft:8,
    },
})
```

SearchResults.js

```
import { StyleSheet, Text, View, Button, Pressable } from 'react-native';
import { useNavigation } from '@react-navigation/native';

function SearchResult({ticker, name, flip}) {
    const nav = useNavigation();


    return (
        <View style={[styles.container, flip && styles.test]}
onTouchEnd={()=>nav.navigate('Stock',{symbol:ticker,name:name})}>
            <Text style={styles.tickerText}>{ticker}</Text>
            <Text style={styles.labelText} numberOfLines={1}>{name}</Text>
        </View>
    );
}
export default SearchResult;

const styles = StyleSheet.create({
    container: {
        alignItems: 'center',
        position:'relative',
        justifyContent: "space-between",
        flexDirection: "row",
        paddingVertical:12,
        paddingHorizontal:24,
    },
    test:{
        backgroundColor: "rgb(26,26,26)",
    },
    tickerText:{
```

```
        fontWeight:'700',
        fontSize:20,
        color:"white",
        paddingRight:24,
    },
    labelText:{
        color:"rgb(218,218,218)",
        overflow:'hidden',
    },
});
```

LineGraphWithToolTips.js

```
import PropTypes from 'prop-types';
import React, { useState } from 'react';
import { Circle, G, Rect, Text } from 'react-native-svg';
import { Dimensions } from 'react-native';
import { LineChart } from 'react-native-chart-kit';

const screenWidth = Dimensions.get('window').width;

const Tooltip = ({ x, y, textX, textY, stroke, pointStroke, position }) => {
    let tipW = 136,
        tipH = 36,
        tipX = 5,
        tipY = -9,
        tipTxtX = 12,
        tipTxtY = 6;
    const posY = y;
    const posX = x;

    if (posX > screenWidth - tipW) {
        tipX = -(tipX + tipW);
        tipTxtX = tipTxtX - tipW - 6;
    }

    const boxPosX = position === 'left' ? posX - tipW - 10 : posX;

    return (
        <G>
            <Circle
                cx={posX}
                cy={posY}
                r={4}
                stroke={pointStroke}
                strokeWidth={2}
                fill={'blue'}
```

```
            />
            <G x={boxPosX < 40 ? 40 : boxPosX} y={posY}>
                <Rect
                    x={tipX + 1}
                    y={tipY - 1}
                    width={tipW - 2}
                    height={tipH - 2}
                    fill={'rgba(255, 255, 255, 0.9)'}
                    rx={2}
                    ry={2}
                />
                <Rect
                    x={tipX}
                    y={tipY}
                    width={tipW}
                    height={tipH}
                    rx={2}
                    ry={2}
                    fill={'transparent'}
                    stroke={stroke}
                />
                <Text x={tipTxtX} y={tipTxtY} fontSize="10" textAnchor="start">
                    {textX}
                </Text>

                <Text
                    x={tipTxtX}
                    y={tipTxtY + 14}
                    fontSize="12"
                    textAnchor="start">
                    {textY}
                </Text>
            </G>
        </G>
    );
};

Tooltip.defaultProps = {
    position: 'right',
};

const tooltipDecorators = (state, data, valueFormatter) => () => {
    if (state === null) {
        return null;
    }
```

```
    const { index, value, x, y } = state;
    const textX = data.labels[index];
    const position = data.labels.length === index + 1 ? 'left' : 'right';

    return (
        <Tooltip
            textX={String(textX)}
            textY={valueFormatter(value)}
            x={x}
            y={y}
            stroke={'#00ccff'}
            pointStroke={'#00ccff'}
            position={position}
        />
    );
};

const LineChartWithTooltips = ({ valueFormatter, ...props }) => {
    const [state, setState] = useState(null);
    return (
        <LineChart
            {...props}
            decorator={tooltipDecorators(state, props.data, valueFormatter)}
            onDataPointClick={setState}
        />
    );
};

LineChartWithTooltips.propTypes = {
    valueFormatter: PropTypes.func,
};

LineChartWithTooltips.defaultProps = {
    valueFormatter: value => String(value),
};

export default LineChartWithTooltips;
```

HomeHeader.js

```
import Ionicons from '@expo/vector-icons/Ionicons';
import { Pressable, StyleSheet, Text, View } from 'react-native';
import { useNavigation } from "@react-navigation/native";

function HomeHeader() {
    const nav = useNavigation();
```

```
    return (
        <Pressable
            style={styles.button}
            onPress={() => nav.navigate('Settings')}
            hitSlop={{
                bottom: 25,
                left: 25,
                right: 25,
                top: 25,
            }}
            pressRetentionOffset={{
                bottom: 50,
                left: 50,
                right: 50,
                top: 50,
            }}
        >
            <Ionicons name="cog" size={24} color={'white'} />
            {/* <Ionicons name="log-out" size={24} color={'white'}
style={styles.spaceLeft} /> */}
        </Pressable>
    );
}
export default HomeHeader;

const styles = StyleSheet.create({
    button: {
       paddingRight:16,
       flexDirection:'row'
    },
    spaceLeft:{
        paddingLeft:18,
    }
});
```

HistoryView.js

```
import { useNavigation } from "@react-navigation/native";
import { useEffect, useState } from "react";
import { getHistory } from "../controller/HistoryController";
import { ActivityIndicator, StyleSheet, Text, View } from "react-native";
import DisplayStock from "./DisplayStock";

export default function HistoryView({history}) {
    if(history === null){
        return(
            <View style={styles.container}>
```

```
                <ActivityIndicator/>
            </View>
        )
    }

    return(
    <View style={styles.container}>
        {history.map((history) =>
        <View style={styles.resultContainer} key={history["created_at"]}>
          <Text style={{color:'white'}}>{history['buy']?"Bought
"+history['stock_amount']+" shares of "+ history['symbol']
          + " at $"+history["price"].toFixed(2)
          :"Sold "+history['stock_amount']+" shares of "+ history['symbol']
          + " at $"+history["price"].toFixed(2)}
          </Text>
        </View>
      )}
    </View>)

}
const styles = StyleSheet.create({
    container: {
      flex: 1,
      paddingVertical:16,
    },
    searchContainer: {
      marginBottom: 16,
    },
    resultContainer: {
      marginVertical: 6,
    },
  });
```

DisplayStock.js

```
import { StyleSheet, Text, View, Button, Pressable } from 'react-native';
import { useNavigation } from '@react-navigation/native';
import Ionicons from '@expo/vector-icons/Ionicons';

function DisplayStock({ symbol, name, price, shares }) {
    const nav = useNavigation();
    return (
        <View style={styles.cardContainer} >
            <View style={styles.cardLeft} onTouchEnd={() => nav.navigate('Stock',
{ symbol: symbol, name: name })}>
                <View style={styles.directionIndicator}>
                    <Ionicons name="arrow-up-outline" size={16} color='black' />
```

```
                </View>
                <View>
                    <Text style={styles.tickerText}>{symbol}</Text>
                </View>
            </View>
            <View style={styles.cardRight}>
                <Text style={styles.priceText}>
                    {price === null ? "Loading..." : "$" + price.toFixed(2)}
                </Text>
            </View>
        </View>
    );
}
export default DisplayStock;

const styles = StyleSheet.create({
    directionIndicator: {
        width: 28,
        height: 28,
        borderRadius: 99,
        backgroundColor: "#3AC070",
        marginRight: 16,
        alignItems:'center',
        justifyContent:'center'
    },
    tickerText: {
        fontWeight: "700",
        fontSize: 18,
        color: "rgb(215,215,215)",
    },
    priceText: {
        fontWeight: "700",
        fontSize: 18,
        color: "rgb(195,195,195)",
    },
    labelText: {
        color: "rgb(195,195,195)",
    },

    cardContainer: {
        alignItems: 'center',
        justifyContent: "space-between",
        flexDirection: "row",
        width: "100%",
        height:40,
```

```
    },
    cardLeft: {
        alignItems: 'center',
        flexDirection: "row"
    },
    cardRight: {
        alignItems: 'center',
    },

});
```

CompanyInfo.js

```
import React, { useState } from "react";
import { View, Text, StyleSheet, Pressable } from "react-native";
import { ScrollView } from "react-native-gesture-handler";
import Ionicons from '@expo/vector-icons/Ionicons';

const CompanyInformation = ({ results }) => {
  let [showAdditionalInfo, setShowAdditionalInfo] = useState(false);
  let [showTerms, setShowTerms] = useState(false);


  if (!results) {
    // If results are empty or not available yet, you can return a loading
indicator or handle accordingly
    return (
      <View>
        <Text style={styles.subtext}>Loading...</Text>
      </View>
    );
  } else if (Object.keys(results).length === 0) {
    return (
      <View >
        <Text style={styles.subtext}>Sorry, no detailed information found.</Text>
      </View>
    );
  }

  // Now you can safely access properties of the results object
  const companyName = results.Name;
  const industry = results.Industry;
  const description = results.Description;


  return (
    <View style={styles.container}>
```

```
      <ScrollView style={styles.scrollContainer}>
        <Pressable onPress={() => setShowAdditionalInfo(!showAdditionalInfo)}
style={styles.button}>
          <Ionicons name='information-circle-outline' size={22}
color={'rgb(25,25,25)'} />
          <Text style={styles.buttonText}>{showAdditionalInfo ? 'Hide addition
company information' : 'Learn More About Company'}</Text>
          <Ionicons name={showAdditionalInfo ? 'arrow-up' : 'arrow-down'}
size={22} color={'rgb(25,25,25)'} />
        </Pressable>
        {showAdditionalInfo ? <>
          <View style={styles.additionalInfoContainer}>
            <Text style={styles.titleText}>Company Name:</Text>
            <Text style={styles.infoText}>{companyName}</Text>
          </View>
          <View style={styles.additionalInfoContainer}>
            <Text style={styles.titleText}>Description:</Text>
            <Text style={styles.infoText}>{description}</Text>
          </View>
          <View style={styles.additionalInfoContainer}>
            <Text style={styles.titleText}>Industry:</Text>
            <Text style={styles.infoText}>{industry}</Text>
          </View>
        </>
          : <></>}

      </ScrollView>

      <ScrollView style={styles.scrollContainer}>
        <Pressable onPress={() => setShowTerms(!showTerms)}
style={styles.button2}>
          <Ionicons name='school-outline' size={22} color={'white'} />
          <Text style={styles.buttonText2}>{showAdditionalInfo ? 'Hide addition
company information' : 'Learn some Terms'}</Text>
          <Ionicons name={showTerms ? 'arrow-up' : 'arrow-down'} size={22}
color={'white'} />
        </Pressable>
        {showTerms ? <>
          <View style={styles.additionalInfoContainer2}>
            <Text style={styles.titleText}>Market Capitalization:</Text>
            <Text style={styles.infoText}>Represents the total value of all a
company's shares of stock; critical for assessing company size and investment
risk.</Text>
          </View>
          <View style={styles.additionalInfoContainer2}>
```

```
            <Text style={styles.titleText}>Price-to-Earnings Ratio (P/E):</Text>
            <Text style={styles.infoText}>A metric used to determine if a stock
is overvalued or undervalued by comparing the current stock price to the
company's earnings per share.</Text>
          </View>
          <View style={styles.additionalInfoContainer2}>
            <Text style={styles.titleText}>Dividend Yield:</Text>
            <Text style={styles.infoText}>A financial ratio that shows how much a
company pays out in dividends each year relative to its stock price; important
for assessing income return on an investment.</Text>
          </View>
        </>
          : <></>}

      </ScrollView>
    </View>
  );
};

export default CompanyInformation;

const styles = StyleSheet.create({
  container: {
    alignItems: 'center',
    width: '100%'
  },
  scrollContainer: {
    marginBottom: 16,
  },
  subtext: {
    fontSize: 14,
    fontStyle: 'italic'
  },
  button: {
    flex: 1,
    flexDirection: 'row',
    paddingVertical: 8,
    borderRadius: 6,
    textAlign: "center",
    alignItems: 'center',
    justifyContent: 'center',
    backgroundColor: 'rgb(218, 218, 218)',
    width: '100%',
  },
  button2: {
```

```
    flex: 1,
    flexDirection: 'row',
    paddingVertical: 8,
    borderRadius: 6,
    textAlign: "center",
    alignItems: 'center',
    justifyContent: 'center',
    width: '100%',
    backgroundColor: '#4992FF',
  },
  buttonText: {
    fontSize: 14,
    fontWeight: '700',
    color: "rgb(25,25,25)",
    marginRight: 16,
    marginLeft: 12
  },
  buttonText2: {
    fontSize: 14,
    fontWeight: '700',
    color: "white",
    marginRight: 16,
    marginLeft: 12
  },
  additionalInfoContainer: {
    backgroundColor: 'rgb(195,195,195)',
    padding: 8,
  },
  additionalInfoContainer2: {
    backgroundColor: '#bfd9ff',
    padding: 8,
  },
  titleText: {
    fontWeight: '600'
  },
  infoText: {
    marginLeft: 4,
  },
  test: {
    backgroundColor: 'green',
  }
});
```

CashSummary.js

```
import { StyleSheet, Text, View, Button, Pressable } from 'react-native';
import {useEffect} from 'react';
```

```
function CashSummary({funds}) {
    if(funds === undefined){
        funds = 0;
    }
    return (
        <View style={styles.cardContainer}>
            <View style={styles.cardHeader}>
                <View>
                    <Text style={styles.cardSubtitle}>Cash Value</Text>
                    <Text style={styles.cardTitle}>{funds===null? 'Loading...' :
'$'+funds.toFixed(2)}</Text>
                </View>
            </View>
        </View>
    );
}
export default CashSummary;

const styles = StyleSheet.create({
    buttonLG: {
        flex: 1,
        paddingVertical: 12,
        borderRadius: 6,
        textAlign: "center",
        alignItems: 'center',
        justifyContent: 'center',
    },
    buttonText: {
        fontSize: 16,
        fontWeight: '700',
        color: "rgb(28,28,28)",
    },

    primaryButton: {
        backgroundColor: "rgb(218, 218, 218)",
        borderColor: "rgb(218, 218, 218)",
        borderWidth: 3,
        marginRight: 16,
    },
    secondaryButton: {
        borderColor: "rgb(218, 218, 218)",
        borderWidth: 3,
    },
    buttonSecondaryText:{
```

```
        color:'rgb(218, 218, 218)',
    },
    cardContainer: {
        alignItems: 'flex-start',
        justifyContent: 'flex-start',
        backgroundColor: "rgb(38,38,38)",
    },
    cardSubtitle: {
        fontSize: 18,
        fontWeight: '500',
        color:'rgb(215,215,215)'
    },
    cardTitle: {
        fontSize: 42,
        fontWeight: '800',
        color:'white'
    },
    cardGraph: {
        flex: 1,
        marginLeft: 32,
        marginRight: 16,
        height: 3,
        backgroundColor: "black"
    },
    cardHeader: {
        alignItems: 'start',
        justifyContent: 'start',
        flexDirection: "row",
        width: "100%",
        padding: 16,
    },
    cardFooter: {
        alignItems: 'center',
        justifyContent: 'space-between',
        flexDirection: "row",
        width: "100%",
        padding: 16,
        backgroundColor: "rgb(26,26,26)",
    },
});
```

BuySellStockCard.js

```
import { StyleSheet, Text, View, Button, Pressable } from 'react-native';
import DisplayStock from "./DisplayStock"
import { useNavigation } from '@react-navigation/native';
```

```
function BuySellStockCard({symbol,name,price}) {
    let nav = useNavigation();
    console.log("the name of the stock: "+name)
    return (
        <View style={styles.container}>
            <View style={styles.cardHeader}>
                <DisplayStock symbol={symbol} name={name} price={price}
origin={"BuySellCard"}/>
            </View>
            <View style={styles.cardFooter}>
                <Pressable style={[styles.button, styles.primaryButton]}
onPress={()=>nav.navigate('Buy or Sell',{symbol:symbol,name:name, price:price,
action:"buy"})}>
                    <Text style={styles.buttonText}>Buy</Text>
                </Pressable>
                <Pressable style={[styles.button, styles.secondaryButton]}
onPress={()=>nav.navigate('Buy or Sell',{symbol:symbol,name:name, price:price,
action:"sell"})}>
                    <Text style={[styles.buttonText,
styles.buttonSecondaryText]}>Sell</Text>
                </Pressable>
            </View>
        </View>
    );
}
export default BuySellStockCard;

const styles = StyleSheet.create({
    container: {
        alignItems: 'flex-start',
        justifyContent: 'flex-start',
        backgroundColor: "rgb(38,38,38)",
        marginTop:-16,
    },
    cardHeader: {
        padding: 16,
    },
    cardFooter: {
        alignItems: 'center',
        justifyContent: 'space-between',
        flexDirection: "row",
        width: "100%",
        padding: 16,
        backgroundColor: "rgb(26,26,26)",
    },
```

```
    button: {
        flex: 1,
        paddingVertical: 12,
        borderRadius: 6,
        textAlign: "center",
        alignItems: 'center',
        justifyContent: 'center',
    },
    buttonText: {
        fontSize: 16,
        fontWeight: '700',
        color: "rgb(28,28,28)",
    },

    buttonSecondaryText: {
        color:'rgb(218, 218, 218)',
    },

    primaryButton: {
        backgroundColor: "rgb(218, 218, 218)",
        borderColor: "rgb(218, 218, 218)",
        borderWidth: 3,
        marginRight: 16,
    },
    secondaryButton: {
        borderColor: "rgb(218, 218, 218)",
        borderWidth: 3,
    },

});
```

Screens

HomeScreen.js

```
import { StyleSheet, Text, View, ActivityIndicator } from 'react-native';
import CashSummary from '../../components/view/CashSummary';
import { getFunds } from '../../components/controller/CashSummaryController';
import { useEffect, useState } from 'react';
import { useNavigation } from '@react-navigation/native';
import HistoryController from '../../components/controller/HistoryController';
import HistoryView from '../../components/view/HistoryView';
import { getHistory } from '../../components/controller/HistoryController';
import { normalizeUnits } from 'moment';
```

```
function HomeScreen() {
    const [funds,setFunds] = useState(null);
    const [loading,setLoading] = useState(true);
    const [name,setName]= useState('')
    const [history,setHistory] = useState(null)
    nav = useNavigation();
    useEffect(()=> {
        const unsubscribe = nav.addListener('focus', () => {
            // Do something when the screen is focused
            // This could be fetching data, updating state, etc.
            getHistory((history)=>{
                console.log(history)
                setHistory(history)
            })
            getFunds((loaded)=>{
                setLoading(loaded)
            },
            (fund)=>{
                console.log('Current funds are:',fund)
                setFunds(fund)},
                (name)=>{
                    setName(name)
                }
            )

          });


          // Return cleanup function to unsubscribe from event listener
          return unsubscribe;
    },[nav]
    )
    if(loading){
        return(<View style={[styles.container,{justifyContent:'center'}]}>
                <ActivityIndicator size='large'
style={{justifyContent:'center'}}/>
            </View>)
    }


    return (
        <View style={styles.container}>
            <Text style={styles.header}>Howdy {name}</Text>
            <CashSummary funds={funds}/>
```

```
            <Text style={styles.header}>History</Text>
            <HistoryView history={history}/>
        </View>
    );
}
export default HomeScreen;

const styles = StyleSheet.create({
    header: {
        fontWeight: '700',
        fontSize: 20,
        marginTop: 24,
        marginBottom:16,
        color:'white',
        paddingLeft:12,
    },
    placeholderText:{
        color:'rgb(215,215,215)',
        fontSize:12,
        paddingLeft:12,
    },
    container: {
        flex: 1,
    },
});
```

ResearchScreen.js

```
import { StyleSheet, Text, View } from 'react-native';
import { ScrollView } from 'react-native-gesture-handler';
import TopEarnersController from
'../../components/controller/TopEarnersController';
import { useState } from 'react';

function ResearchScreen() {
    const[lastUpdate,setLastUpdate] = useState("");
    return (
        <View style={styles.container}>
            <Text style={styles.header}>Yesterday's Top Volume Traded</Text>
            <View></View>
            <Text style={styles.date}>{"Last Update: "+lastUpdate}</Text>
            <ScrollView>
            <TopEarnersController date={(date)=> {
                console.log("Last date was: "+date)
                setLastUpdate(date)}
            }/>
            </ScrollView>
```

```
        </View>
    );
}
export default ResearchScreen;

const styles = StyleSheet.create({
    container: {
        flex: 1,
        padding: 20,
    },
    stockContainer: {
        marginVertical: 6,
    },
    header: {
        fontWeight: '700',
        fontSize: 20,
        marginBottom: 3,
        color:'rgb(235,235,235)'
    },
    date:{
        fontSize:12,
        fontStyle:'italic',
        color:'rgb(215,215,215)'
    }
});
```

SearchScreen.js

```
import { useState, useEffect } from "react"
import {
    StyleSheet, Text, View, SafeAreaView,
    ScrollView
} from 'react-native';
import SearchInput from '../../components/controller/SearchInput'
import SearchResult from '../../components/view/SearchResult';

function SearchScreen() {
    const [searchResults, setSearchResults] = useState([])
    const [hasSearched,setHasSearched] = useState(false)

    return (
        <SafeAreaView style={styles.container}>
            <View style={styles.searchContainer}>
                <SearchInput onSearch={(results) => {
                    console.log("Search results:", results)
```

```
                    setSearchResults(results)
                    setHasSearched(true)
                }
                } />
            </View>{
                //display no results if the search field is empty or if there are
no results
            searchResults== null|| (searchResults.length==0 && hasSearched) ?
             <View style={styles.resultContainer} >
                <Text style={{textAlign:'center'}}>No Results</Text>
            </View>
            :
            <ScrollView style={styles.scrollView}>
                {searchResults.map((result,i) =>
                    <View style={styles.resultContainer} key={result["1.
symbol"]}>
                        <SearchResult ticker={result["1. symbol"]}
name={result["2. name"]} flip={!(i%2)} />
                    </View>
                )}
            </ScrollView >}
        </SafeAreaView>
    );
}
export default SearchScreen;

const styles = StyleSheet.create({
    container: {
        flex: 1,
    },
    searchContainer: {
        marginBottom: 24,
    },
    resultContainer: {
    },
});
```

SettingsScreen.js

```
import { StyleSheet, Switch, Text, View,Button} from 'react-native';
import { supabase } from '../../database/supabase';


function SettingsScreen() {
  return (
    <View style={styles.settingsContainer}>
      <Text style={styles.headerText}>
```

```
        Account Settings
      </Text>
      <Text style={styles.headerText}>
        Security Settings
      </Text>
      <Text style={styles.headerText}>
        Accessibility Settings
      </Text>
      <Text style={styles.headerText}>
        Appearance Settings
      </Text>
      <Button title="Sign Out" onPress={() => supabase.auth.signOut()} />
    </View>
  );
}
export default SettingsScreen;

const styles = StyleSheet.create({
  headerText: {
    fontSize: 16,
    marginBottom: 16,
  },
  settingsContainer: {
    padding: 18
  }
});
```

TradeScreen.js

```
import { StyleSheet, Text, View } from 'react-native';
import TradeScreenController from
'../../components/controller/TradeScreenController';

function TradeScreen() {
    return (
        <View style={styles.container}>
            <Text style={styles.header}>My Stocks</Text>
            <TradeScreenController/>
        </View>
    );
}
export default TradeScreen;

const styles = StyleSheet.create({
    container: {
        flex: 1,
        padding: 20,
```

```
    },
    header: {
        fontWeight: '700',
        fontSize: 20,
        marginBottom: 16,
        color:'white'
    },
    cardContainer: {
        marginVertical: 6,
    }
});
```

LoginScreen.js

```
import { Button, StyleSheet, Text, View } from 'react-native';
import { useNavigation } from "@react-navigation/native";
import { supabase } from '../../database/supabase';
import 'react-native-url-polyfill/auto'
import { useState,useEffect } from 'react';
import { Session } from '@supabase/supabase-js';
import { signInWithEmail } from '../../components/controller/SignInController';
import SignIn from '../../components/view/SignIn';
import { Image } from 'expo-image';
function LoginScreen() {
    const [session, setSession] = useState(null)

    useEffect(() => {
      supabase.auth.getSession().then(({ data: { session } }) => {
        setSession(session)
      })

      supabase.auth.onAuthStateChange((_event, session) => {
        setSession(session)
      })
    })
    const nav = useNavigation();
    const signUp = <Text onPress={()=>nav.navigate('Sign Up')}
style={{color:'red'}}>Sign up</Text>
    const resetPassword = <Text style={{color:'red'}}
onPress={()=>nav.navigate('Reset Password')}>Reset password</Text>

    return (
        //no style sheet on this because component has built in style
        <View style={styles.container}>
            <Text style={styles.headerText}>Welcome to</Text>
            <Image source={require('../../assets/RedTradeFullLogo.png')}
              contentFit='contain'
```

```
              style={styles.image}
            />
            <Text style={styles.subheaderText}>Sign In or Sign Up</Text>
              <SignIn/>
            <Text style={styles.text}>Don't have an account? {signUp}</Text>
            <Text style={styles.text}>Forgot your password?
{resetPassword}</Text>
        </View>
    );
}
export default LoginScreen;

const styles = StyleSheet.create({
    headerText:{
      marginTop:20,
      color:'white',
      fontSize:28,
      fontWeight:'500',
      fontStyle:'italic'
    },
    subheaderText:{
      color:'rgb(215,215,215)',
      fontSize:22,
      fontWeight:'400'
    },
    container: {
        flex: 1,
        justifyContent: 'center',
        padding:12
    },
    text:{
      color:'rgb(220,220,220)',
      textAlign:'center'
    },
    specialText:{
      fontWeight:'600',
      textDecorationLine:'underline',
    },
    image:{
      contentFit:"cover",
      height:'10%'
    }
});
```

PasswordResetScreen.js

```
import {  StyleSheet, Text, View, Alert} from 'react-native';
```

```
import { Button,Input } from 'react-native-elements';
import { supabase } from '../../database/supabase';
import 'react-native-url-polyfill/auto'
import { useState,useEffect } from 'react';


function PasswordResetScreen(){

/**
 * Step 1: Send the user an email to get a password reset token.
 * This email contains a link which sends the user back to your application.
 */

    const [email,setEmail] = useState('')
    const [loading,setLoading] = useState(false)

    async function resetPassword(){
        setLoading(true)
        const { data, error } = await supabase.auth
        .resetPasswordForEmail(email)
        if(error) Alert.alert(error.message)
        else Alert.alert("Check your inbox for reset link.")
        setLoading(false)
    }

    return(
        <View styles={styles.container}>
            <View style={[styles.verticallySpaced,styles.mt20]}>
            <Input label="Email" Value={email || ''} onChangeText={(text) =>
setEmail(text)} />
            </View>
            <View style={styles.verticallySpaced}>
            <Button  disabled={loading} onPress={()=>resetPassword()}
title='Reset Password'/>
            </View>
        </View>
    );
}
export default PasswordResetScreen;

const styles = StyleSheet.create({
    container: {
        flex:1,
        marginTop: 40,
        padding: 12,
```

```
        justifyContent:'center'
      },
      verticallySpaced: {
        paddingTop: 4,
        paddingBottom: 4,
        alignSelf: 'stretch',
      },
      mt20: {
        marginTop: 20,
      },
});
```

SignUpScreen.js

```
import React, { useState } from 'react'
import { Alert, StyleSheet, View, AppState, Input, Text } from 'react-native'
import { Button } from 'react-native-elements'
import { supabase } from '../../database/supabase'
import SignUp from '../../components/view/SignUp';
import { useNavigation } from "@react-navigation/native";

function SignUpScreen() {

    const nav = useNavigation();
    const signIn = <Text onPress={() => nav.navigate('Login')}
style={styles.specialText}>Sign In Here</Text>

    return (
        <View style={styles.container}>
            {/* <Text style={styles.headerText}>Sign Up</Text> */}
            <SignUp/>
            <Text style={styles.text}>Already have an account? {signIn}</Text>
        </View>
    );
}
export default SignUpScreen;

const styles = StyleSheet.create({
    headerText: {
        marginTop: 40,
        color: 'white',
        fontSize: 28,
        fontWeight: '500',
        marginBottom:-40
    },
    subheaderText: {
        color: 'rgb(215,215,215)',
```

```
        fontSize: 22,
        fontWeight: '400'
    },
    container: {
        flex: 1,
        justifyContent: 'center',
        padding: 10,
    },
    text: {
        color: 'rgb(220,220,220)'
    },
    specialText: {
        fontWeight: '600',
        textDecorationLine: 'underline',
    }
});
```

SplashScreen.js

```
import { Button, StyleSheet, Text, View,Image} from 'react-native';
import { useNavigation } from "@react-navigation/native";
function SplashScreen() {
    return (
        <View style={styles.container}>
            <Image Source={require('../../assets/RedTradeSplash.png')}/>
        </View>
    );
}
export default SplashScreen;

const styles = StyleSheet.create({
    container: {
        flex: 1,
        alignItems: 'center',
        justifyContent: 'center',
    },
});
```

BuySellScreen.js

```
import { ActivityIndicator, Pressable, StyleSheet, Text, View, Alert } from
'react-native';
import { useNavigation } from '@react-navigation/native';
import { useState, useEffect } from 'react';
import { Input, Button } from 'react-native-elements';
import { Slider } from '@miblanchard/react-native-slider';
import { getFunds } from '../../components/controller/CashSummaryController';
```

```
import { PlaceBuyOrSell, getCurrentHoldings } from
'../../components/controller/BuySellController';


function BuySellScreen({ route }) {
    const nav = useNavigation();
    const [action, setAction] = useState(route.params.action)
    const [symbol, setSymbol] = useState(route.params.symbol.toUpperCase())
    const [name, setName] = useState(route.params.name)
    const [price, setPrice] = useState(route.params.price)
    const [shares, setShares] = useState(1);
    const [loading, setLoading] = useState(true);
    const [executing, setExecuting] = useState(false);
    const [currentHoldings, setCurrentHoldings] = useState(null)
    //TODO: Make this actually pull in amount of owned shares for sell page
    const [ownedShares, setOwnedShares] = useState('');
    //TODO: Make this actually pull in total amount spendable (Using 1000 as a
fill in)
    const [totalFunds, setTotalFunds] = useState(null)
    useEffect(() => {
        getFunds((loaded) => {
            setLoading(loaded)
        },
            (fund) => {
                console.log('Current funds are:', fund)
                setTotalFunds(fund)
            },
            (name)=>{
            }
        )
        getCurrentHoldings((holding) => {
            console.log('current holdings are:', holding)
            setCurrentHoldings(holding)
        }, symbol
        )
    }, ['']
    )
    const handlePress = async () => {
        try {
            // Call your asynchronous function here
            await PlaceBuyOrSell(action, symbol, shares, Math.round(price *
shares * 100) / 100,
                (executing) => {
                    setExecuting(executing)
                }, price, currentHoldings);
```

```
            // After the async function completes, navigate back to the previous
screen
            nav.goBack();
        } catch (error) {
            console.error('Error:', error);
            // Handle errors if needed
        }
    };

    if (loading === true || currentHoldings === null) {
        return (<View style={styles.container}>
            <ActivityIndicator />
        </View>)
    }
    if (totalFunds < price && action === "buy") {
        Alert.alert("Insuffecient Funds", "Sorry, you don't have enough funds to
buy this stock", [{
            text: 'Ok',
            onPress: () => nav.goBack()
        }]);
    }
    if ((currentHoldings === 0 ||currentHoldings === -1) && action === "sell") {
        Alert.alert("No Owned Stock", "Sorry, you don't own any of this stock",
[{
            text: 'Ok',
            onPress: () => nav.goBack()
        }]);
    }

    return (
        <View style={styles.container}>
            <Text style={styles.action}>{action}</Text>
            {
                action === 'sell' ? (
                    <View style={styles.fundContainer}>
                        <Text style={styles.fundText}>Total Value</Text>
                        <Text style={styles.fundText}>{"$" + Math.round(price *
currentHoldings * 100) / 100}</Text>
                    </View>
                ) : (
                    <View style={styles.fundContainer}>
                        <Text style={styles.fundText}>Total Funds</Text>
                        <Text style={styles.fundText}>{"$" + totalFunds}</Text>
                    </View>
```

```
                )
            }

            <View style={styles.priceContainer}>
                <Text style={styles.dolla}>$</Text><Text
style={styles.price}>{(price * shares).toFixed(2)}</Text>
            </View>
            <Text style={styles.shareAmount}>{shares} {shares === 1 ? 'Share' :
'Shares'}</Text>
            {action === 'sell' && currentHoldings >=1  ?
            <View style={styles.sliderContainer}>
                <Slider
                    value={shares}
                    onValueChange={shares => setShares(shares)}
                    minimumValue={1}
                    maximumValue={Math.floor(currentHoldings>=1?currentHoldings:1
)}
                    style={styles.slider}
                    step={1}
                />
            </View>
            :
            <View style={styles.sliderContainer}>
                <Slider
                    value={shares}
                    onValueChange={shares => setShares(shares)}
                    minimumValue={1}
                    maximumValue={Math.floor(totalFunds /
price)<1?10:Math.floor(totalFunds / price)}
                    style={styles.slider}
                    step={1}
                />
            </View>
            }
            {/* TODO: Replace console.log with supabase calls to */}
            <Pressable onPress={handlePress} disabled={executing}>
                <Text style={[styles.pressable, (action == 'sell' ?
styles.pressableSell : styles.pressableBuy)]} >{action} {shares} {shares === 1 ?
'Share' : 'Shares'} of {symbol.toUpperCase()}</Text>
            </Pressable>
        </View>
    );
}
export default BuySellScreen;
```

```
const styles = StyleSheet.create({
    container: {
        flex: 1,
        alignItems: 'center',
        justifyContent: 'center',
    },
    fundContainer: {
        position: 'absolute',
        top: 32,
        right: 32,
    },
    fundText: {
        fontWeight: '600',
        fontSize: 16,
        color: 'rgb(215,215,215)'
    },
    action: {
        position: 'absolute',
        left: 24,
        top: 24,
        fontSize: 48,
        textTransform: 'uppercase',
        fontWeight: '800',
        letterSpacing: 8,
        opacity: 0.3,
        color:'white'
    },
    priceContainer: {
        display: 'flex',
        flexDirection: 'row',
        alignItems: 'center',
    },
    dolla: {
        fontSize: 24,
        fontWeight: '800',
        marginRight: 8,
        color:'rgb(215,215,215)'
    },
    price: {
        fontSize: 56,
        fontWeight: '600',
        color:'rgb(235,235,235)'
    },
    shareAmount: {
        fontSize: 18,
```

```
        fontWeight: '500',
        color:'rgb(165,165,165)'
    },
    pressable: {
        textTransform: 'capitalize',
        padding: 16,
        width: 'auto',
        color:'white',
        borderRadius: 8,
        fontWeight:'600',
    },
    pressableBuy:{
        backgroundColor: "#4992FF",

    },
    pressableSell:{
        backgroundColor: "#FF4949",
    },
    sliderContainer: {
        paddingLeft: 64,
        paddingRight: 64,
        alignItems: 'stretch',
        justifyContent: 'center',
        width: '100%',
        marginTop: 12,
        marginBottom: 12,
    },
    slider: {
    }
});
```

IndividualStockScreen.js

```
import { useState, useEffect } from 'react';
import { StyleSheet, Text, View, Alert, ActivityIndicator } from 'react-native';
import { symbolApi } from '../../components/controller/apiStockVisual';
import CompanyInfoController from '../../components/controller/apiCompanyInfo';
import BuySellStockCard from '../../components/view/BuySellStockCard';
import StockTimeSeries from '../../components/view/StockTimeSeries';
import { useNavigation } from '@react-navigation/native';

export default function IndividualStockScreen({ ticker, route }) {


    const [loading, setLoading] = useState(false)
    const [symbol, setSymbol] = useState(route.params.symbol)
    const [name, setName] = useState(route.params.name)
```

```
    console.log(symbol)
    const [lastPrice, setLastPrice] = useState(null)
    const [hasTimeSeriesInfo, setHasTimeSeriesInfo] = useState(null)
    const [timeResult, setTimeResult] = useState(null)
    const nav = useNavigation()
    useEffect(() => {
        symbolApi(symbol, name, (lastPrice) => {
            console.log("Closing Price:", lastPrice)
            setLastPrice(lastPrice)
        },
            (timeSeries) => {

                //bool that says whether or not symbol has a time series
                setHasTimeSeriesInfo(timeSeries)

            },
            (result) => {
                console.log(result)
                setTimeResult(result)
            }
        )
    }
        , [symbol]
    )

    if (hasTimeSeriesInfo === null) {
        return (<View style={styles.container}>
            <ActivityIndicator style />
        </View>
        )
    } else if (hasTimeSeriesInfo === false) {
        Alert.alert("No Data Found", "No stock information for this symbol,
sorry", [{
            text: 'Ok',
            onPress: () => nav.goBack()
        }]);
    } else {
        console.log(name)

        return (
            <View style={styles.container}>
                {timeResult ? (
                    // Render content when result is available
                    <View>
                        <StockTimeSeries chartData={timeResult} title={name} />
```

```
                        {/* Add more components to display other information */}
                    </View>
                ) : (
                    // Render a loading indicator while waiting for data
                    <View>
                        <Text>Loading...</Text>
                    </View>
                )}
                <BuySellStockCard symbol={symbol} name={name} price={lastPrice}
/>
                <CompanyInfoController symbol={symbol} />
            </View>
        );
    }
}

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

Navigation

HomeStack.js

```
import { createStackNavigator } from "@react-navigation/stack";

import HomeScreen from "../../screens/home/HomeScreen";
import SettingsScreen from "../../screens/home/SettingsScreen";
import HomeHeader from "../../components/view/HomeHeader";

const Stack = createStackNavigator();

const HomeStack = () => (
    <Stack.Navigator screenOptions={{headerRight:() => <HomeHeader/>,
    }}>
        <Stack.Screen name="Home" component={HomeScreen} options={
            {headerShown:true}
        }/>
        <Stack.Screen name="Settings" component={SettingsScreen}
options={{headerShown:true}}/>
    </Stack.Navigator>
);
```

```
export default HomeStack;
```

ResearchStack.js

```
import { createStackNavigator } from "@react-navigation/stack";
import ResearchScreen from "../../screens/home/ResearchScreen";
import IndividualStockScreen from "../../screens/stocks/IndividualStockScreen";
import BuySellScreen from "../../screens/stocks/BuySellScreen";

const Stack = createStackNavigator();
//TODO add a header info icon to pop up an alert to let people know that the data
displayed on this page is from the previous date
const ResearchStack = () => (
    <Stack.Navigator screenOptions={{
    }}>
        <Stack.Screen name="Research" component={ResearchScreen} options={
            {headerShown:true}
        }/>
        <Stack.Screen name="Stock" component={IndividualStockScreen}/>
        <Stack.Screen name="Buy or Sell" component={BuySellScreen}/>
    </Stack.Navigator>
);

export default ResearchStack;
```

SearchStack.js

```
import { createStackNavigator } from "@react-navigation/stack";
import SearchScreen from "../../screens/home/SearchScreen";
import IndividualStockScreen from "../../screens/stocks/IndividualStockScreen";
import BuySellScreen from "../../screens/stocks/BuySellScreen";

const Stack = createStackNavigator();

const SearchStack = () => (
    <Stack.Navigator screenOptions={{
    }}>
        <Stack.Screen name="Search" component={SearchScreen} options={
            {headerShown:true}
        }/>
        <Stack.Screen name="Stock" component={IndividualStockScreen}/>
        <Stack.Screen name="Buy or Sell" component={BuySellScreen}/>
    </Stack.Navigator>
);

export default SearchStack;
```

TradeStack.js

```
import { createStackNavigator } from "@react-navigation/stack";
import IndividualStockScreen from "../../screens/stocks/IndividualStockScreen";
import TradeScreen from "../../screens/home/TradeScreen";
import BuySellScreen from "../../screens/stocks/BuySellScreen";

const Stack = createStackNavigator();

const TradeStack = () => (
    <Stack.Navigator screenOptions={{
    }}>
        <Stack.Screen name="Trade" component={TradeScreen} options={
            {headerShown:true}
        }/>
        <Stack.Screen name="Stock" component={IndividualStockScreen}/>
        <Stack.Screen name="Buy or Sell" component={BuySellScreen}/>
    </Stack.Navigator>
);

export default TradeStack;
```

AppStack.js

```
import { createStackNavigator } from "@react-navigation/stack";
import SplashScreen from "../screens/signIn/SplashScreen";
import RootTab from "./RootTab";
import { Session } from "@supabase/supabase-js";
import { useState,useEffect, createContext } from "react";
import { supabase } from "../database/supabase";
import SignInStack from "./SignInStack";

const Stack = createStackNavigator();
const SessionContext = createContext(null);

function AppStack(){
    const [session, setSession] = useState(null)
    const [loading,setLoading] = useState(true)

    useEffect(()=>{
        const subscription = supabase.auth.onAuthStateChange(
            async (event, session) => {
              if (event === 'SIGNED_OUT') {
                setSession(null)
                setLoading(false)
              }
              else if (event == "PASSWORD_RECOVERY") {
```

```
                const newPassword = prompt("What would you like your new password
to be?");
                const { data, error } = await supabase.auth
                  .updateUser({ password: newPassword })

                if (data) alert("Password updated successfully!")
                if (error) alert("There was an error updating your password.")
              }
              else{
                setLoading(false)
                setSession(session)
              }
    })

    return () => {
        subscription.unsubscribe()
      }
    }, [])

    return(
    <Stack.Navigator screenOptions={{headerShown:false}}>
      {/* While checking loading status, show splash screen */}
        {loading ?
        (<Stack.Screen name="SplashScreen" component={SplashScreen}/>) :
        //if the token is null, show the sign in or sign up page
        session == null ?
        (<Stack.Screen name="PasswordStack" component={SignInStack}/>)
        //else sign in to the main stack.
        //TODO once app screens are completed, create tutorial
        :
        (<Stack.Screen name="App" component={RootTab}/>)
    }
    </Stack.Navigator>
    )
}

export default AppStack;
```

RootTab.js

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import Ionicons from '@expo/vector-icons/Ionicons';

import HomeStack from './rootStacks/HomeStack';
import ResearchStack from './rootStacks/ResearchStack';
import SearchStack from './rootStacks/SearchStack';
```

```
import TradeScreen from '../screens/home/TradeScreen';
import HomeHeader from '../components/view/HomeHeader'
import TradeStack from './rootStacks/TradeStack';

const Tab = createBottomTabNavigator();

function RootTab() {
  return (
    <Tab.Navigator screenOptions={{
      tabBarStyle: {borderTopWidth:0, height:70},
      tabBarShowLabel:false,

    }}>
      <Tab.Screen name="HomeStack" component={HomeStack} options={{
        headerShown:false,
      tabBarIcon: ({color}) => <Ionicons name="home" size={28} color={color} />,
      title:"Home"
    }} />
      <Tab.Screen name="ResearchStack" component={ResearchStack} options={{
        headerShown:false,
      tabBarIcon: ({color}) => <Ionicons name="library" size={28} color={color}
/>,
    }} />
      <Tab.Screen name="SearchStack" component={SearchStack} options={{
        headerShown:false,
      tabBarIcon: ({color}) => <Ionicons name="compass" size={28} color={color}
/>,
    }} />
      <Tab.Screen name="TradeStack" component={TradeStack} options={{
        headerShown:false,
      tabBarIcon: ({color}) => <Ionicons name="cash" size={28} color={color} />,
    }} />
    </Tab.Navigator>
  );
}
export default RootTab;
```

SignInStack.js

```
import { createStackNavigator } from "@react-navigation/stack";

import LoginScreen from "../screens/signIn/LoginScreen";
import PasswordResetScreen from "../screens/signIn/PasswordResetScreen";
import SignUpScreen from "../screens/signIn/SignUpScreen";

const Stack = createStackNavigator();
```

```
const SignInStack = () => (
    <Stack.Navigator screenOptions={{
    }}>
        <Stack.Screen name="Login" component={LoginScreen}
options={{headerShown:false}}/>
        <Stack.Screen name="Reset Password" component={PasswordResetScreen}/>
        <Stack.Screen name="Sign Up" component={SignUpScreen}
options={{headerShown:true}} />
    </Stack.Navigator>
);

export default SignInStack;
```