Software Development Life Cycle

HI-AL-AN

TLD

Hiep Bui

Alper Ozdamar

Anurag Jha

# Document Identification

|  |  |
| --- | --- |
| Document Type | Top Level Design Document |
| Customer Name | SDLC Course |
| Document Location |  |

# Document Version Control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Created By | Change | Approved By |
| 1.0 | 01.05.2020 | Alper Ozdamar | First draft. Skeleton, Flows, E-R Diagram, etc. |  |
| 1.1 | 02.05.2020 | Anurag Jha | Edited for introduction, CI/CD,etc. |  |
| 1.2 | 03.05.2020 | Hiep Bui | Edited for CI/CD, React and Mobile part, etc. |  |
| 1.3 | 10.05.2020 | All Team | Latest updates before final delivery. |  |
|  |  |  |  |  |

# Table of Contents

[Document Identification 2](#_Toc39951054)

[Document Version Control 2](#_Toc39951055)

[Table of Contents 3](#_Toc39951056)

[Table of Figures 4](#_Toc39951057)

[1. References 5](#_Toc39951058)

[2. Glossary 5](#_Toc39951059)

[3. Abbreviations 5](#_Toc39951060)

[4. Introduction 6](#_Toc39951061)

[5. Architecture 7](#_Toc39951062)

[6. Interfaces 8](#_Toc39951063)

[6.1. Internal Interfaces 8](#_Toc39951064)

[6.2. External Interfaces 8](#_Toc39951065)

[7. Data Model (E-R Diagram) 9](#_Toc39951066)

[8. Tech Stack 10](#_Toc39951067)

[8.1. Back-End 10](#_Toc39951068)

[8.1.1. Java 10](#_Toc39951069)

[8.1.2. Micronaut 10](#_Toc39951070)

[8.1.3. RDMS 11](#_Toc39951071)

[8.1.4. Hibernate 11](#_Toc39951072)

[8.1.5. AWS 11](#_Toc39951073)

[8.1.6. AWS Lambda 12](#_Toc39951074)

[8.1.7. OAuth2 12](#_Toc39951075)

[8.1.8. Cloud Watch 13](#_Toc39951076)

[8.2. Front-End 13](#_Toc39951077)

[8.2.1. React 13](#_Toc39951078)

[8.3. Mobile 13](#_Toc39951079)

[8.3.1. Expo Framework 13](#_Toc39951080)

[8.3.2. Redux 13](#_Toc39951081)

[8.4. CI/CD 13](#_Toc39951082)

[9. Flows 15](#_Toc39951083)

[9.1.1. StockExtractor Flow 15](#_Toc39951084)

[9.1.2. Login/Oauth2 Flow 15](#_Toc39951085)

[9.1.3. Rebalancing Flow 16](#_Toc39951086)

[9.1.4. Buy/Sell Flow 16](#_Toc39951087)

[10. GUI Pages 17](#_Toc39951088)

[11. Logging 19](#_Toc39951089)

# Table of Figures

[Figure 1: Overall Architecture 7](#_Toc39950849)

[Figure 2: External Interfaces (IEX API) 8](#_Toc39950850)

[Figure 3: E-R Diagram 9](#_Toc39950851)

[Figure 4: Oauth2 Mechanism 13](#_Toc39950852)

[Figure 5: HiAlan CI/CD 14](#_Toc39950853)

[Figure 3: Home Screen 17](#_Toc39950854)

[**Figure 4: Invest Screen** 17](#_Toc39950855)

[**Figure 5: Stock Latest Price Screen** 18](#_Toc39950856)

# References

1. IEX API : <https://iexcloud.io/docs/api/>

# Glossary

|  |  |
| --- | --- |
|  |  |
|  |  |

# Abbreviations

|  |  |
| --- | --- |
| AWS | Amazon Web Services |
| IEX | Investors Exchange |
|  |  |

# Introduction

This document’s aim is to give a detail explanation and design details of HiAlan project. HiAlan is a mobile/web application, built and optimised for the giving financial advices to mobile users.

# Architecture



Figure 1: Overall Architecture

# Interfaces

# Internal Interfaces

Front-end and back-end communicates via REST APIs.

# External Interfaces

Our product will pull stock information 3rd Party iex API. Hialan calls IEX API through our lambda functions as you see in the below graph. For further information about IEX please visit: <https://iexcloud.io/docs/api/>

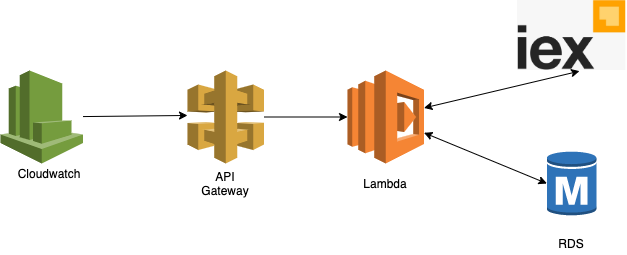


Figure 2: External Interfaces (IEX API)

# Data Model (E-R Diagram)

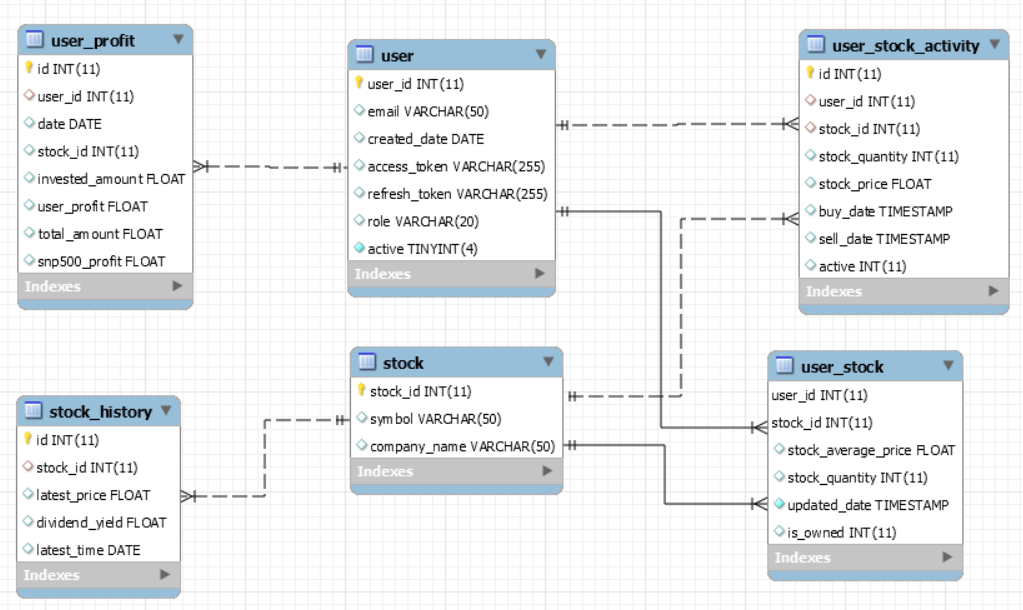


Figure 3: E-R Diagram

Our E-R diagram can be seen above. **User** table is storing user information like email, role, credentials. **Stock** table is for storing basic stock information. **User\_stock** information table stores user’s stock information, like stock quantity. **Stock\_history** table stores stock price, dividend\_yield in daily basis. **User\_stock\_activity** table stores every user activity related with stock like buy\_date, sell\_date, stock\_price,etc. And finally, **user\_profit** table daily stores user profit depending on the stocks that user have. This data is stored daily basis and also contains relative snp500 profit value.

# Tech Stack

# Back-End

# Java

Hialan back-end side is implemented by Java programming language. The main reason of choosing Java is the skill set of our team. We were be able to build our backend APIs very fast with Java. Many serious programmers tend to like statically typed code both for the simplicity and the safety. Other reasons Java developers have Eclipse, NetBeans, or IntelliJ, three top IDEs that are well-integrated with debuggers, decompilers, and servers.

We did a load test with Jmeter in the past to compare the performance of Java Lambda and Node.js lambda functions. Even results was showing the clear winner is the node.js. Based on our products requirements, the average response time and memory usage difference between node.js and Java was negligible. You can see the load test details of both Java and node.js in this document: <https://drive.google.com/open?id=13iiMvNrv8X4oUA0Ue9PzcKCBLy7PaH2t>

We thought that cold start problem and memory usage problems are negligible and will not affect our product’s functionality and availability.

|  |  |  |
| --- | --- | --- |
| **50 Threads- 10 Minutes TEST** | **JAVA** | **NODE JS** |
| **MAX Mbyte** | 164 MByte | 126 MByte |
| **Average Response Time** | 143 ms | 69 ms |
| **FAILED REQUESTS** | 0 | 0 |
| **Total Requests** | 214.891 requests | 563.753 requests |

# Micronaut

As a development framework, our team were thinking to use Spring but then our professor advise us to search for Micronaut framework and our team found that Micronaut has enormous advantages over Spring. First advantage is Micronaut is giving us **Natively cloud-native app development’**. Most popular cloud features are supported by default without depending on any third party cloud services. The other big advantage is Micronaut’s **fast startup time**, **compile-time approach** and **low-memory footprint**. Micronaut features have the dedicated support for implementing and deploying functions to the AWS Lambda. And to be honest, we wanted to learn something new and something dedicated to cloud, so eventually we choosed Micronaut.

|  |  |
| --- | --- |
| https://lh5.googleusercontent.com/xgx0hoWu005k9dFHaWGyZATJ3DyYzXkqzJCyiHzIilT6hUjYDfcUTLLvic7OPfBudCRhITKc7QcqL3zZ_1EptKOeDAPB_hG3v5kFrkTgY7qHIvMexkrlzHVGPjeifl4oYpuZseZ8VNc | https://lh5.googleusercontent.com/BzOX0ILYH4v1PLD4rBOGY47FzDMQmcvHrWeClkBqQcCxmirTeiSRo9_JyUD9T_1GMH8cMDWYQByVnH79-AJYnd9U2Wg2uJ18VGEWRBtgzVXvvIiRYEG4vhggP3KzUe4RxV6eWR1ifrk |

# RDMS

Our team choosed to use SQL because our data was structured and in future we may need complex queires to run. As a RDMS engine, we decided to use Mysql which is is one of the world’s most popular databases. It is open source, reliable, compatible with all major hosting providers, cost-effective, and easy to manage.

# Hibernate

Our team decided to use ORM instead of plain JDBC and as ORM technology we choosed Hibernate. Because our team had positive experienced in Hibernate. Hibernate's primary feature is mapping from Java classes to database tables (and from Java data types to SQL data types). Hibernate also provides data query and retrieval facilities. Hibernate generates the SQL calls and attempts to relieve the developer from manual result set handling and object conversion and keep the application portable to all supported SQL databases with little performance overhead.

|  |  |  |
| --- | --- | --- |
|  | Hibernate | JDBC |
| DB Engine Dependence | Hibernate uses HQL which are independent of database. | Uses SQL queries for database interaction which vary with the type of database. |
| Cache | Provides two levels of caching. | Provides no caching support. |
| Connection Pooling | Can use by adding a dependency in your application build file&configuring it in hibernate configuration file | Itself does not provide any connection pooling facility. |

Hibernate provides a **caching mechanism**, which helps reduce the number of hits, as much as possible, that application makes to the database server. This will have a considerable effect regarding the performance of our application. There is no such caching mechanism available in JDBC. Hibernate **reduces the amount of** repeating lines of **code**, which you can often find with JDBC. For your understanding, I have left a simple scenario below.

# AWS

Our team choosed AWS as our cloud platform. Amazon Web Services is a cloud computing platform which provides services such as compute power, database storage, content delivery and many other functions. It is flexible, scalable, and reliable and due to this many companies are implementing it in their work.

AWS has an active developer community and we can be able to find what we are looking for, good documentation and tutorials. And the last but not the least our team had an AWS experience prior to this project. So we decided to choose AWS cloud platform.

# AWS Lambda

In the backend, our team used AWS Lambda functions. AWS Lambda is Amazon’s serverless compute service. You can run your code on it without having to manage servers or even containers. It’ll automatically scale depending on how much work you feed into it. When you do run Lambdas in the cloud, you’ll only pay for what you use. In some cases, this can save significant sums of money compared to the cost of running VMs or [containers](https://scalyr.com/blog/containers-benefits-and-making-a-business-case/). We can easily configure our lambda functions through AWS. (Like DB credentials, oauth2 parameters, mobile client parameters etc.)

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Pricing | Performance |
| Reduced Maintenance | Cost alterations |
| Scalability |  |
| Faster Delivery |  |

# OAuth2

Our team needed some authentication mechanism for authenticating user identity and upcoming requests. So we choosed OAuth 2.0. OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones. So as an example when user clicks signup button our Google Oauth2 authentication mechanism is triggering. User enters his gmail account credentials. By doing this user gives permission to our application to access their necessary information to authenticate them to our system. If Google Authorization server accepts credentials, Google calls our callback function and at this point, we are creating the user in our database with user’s access token. And for the upcoming requests we validate user requests based on that access token.

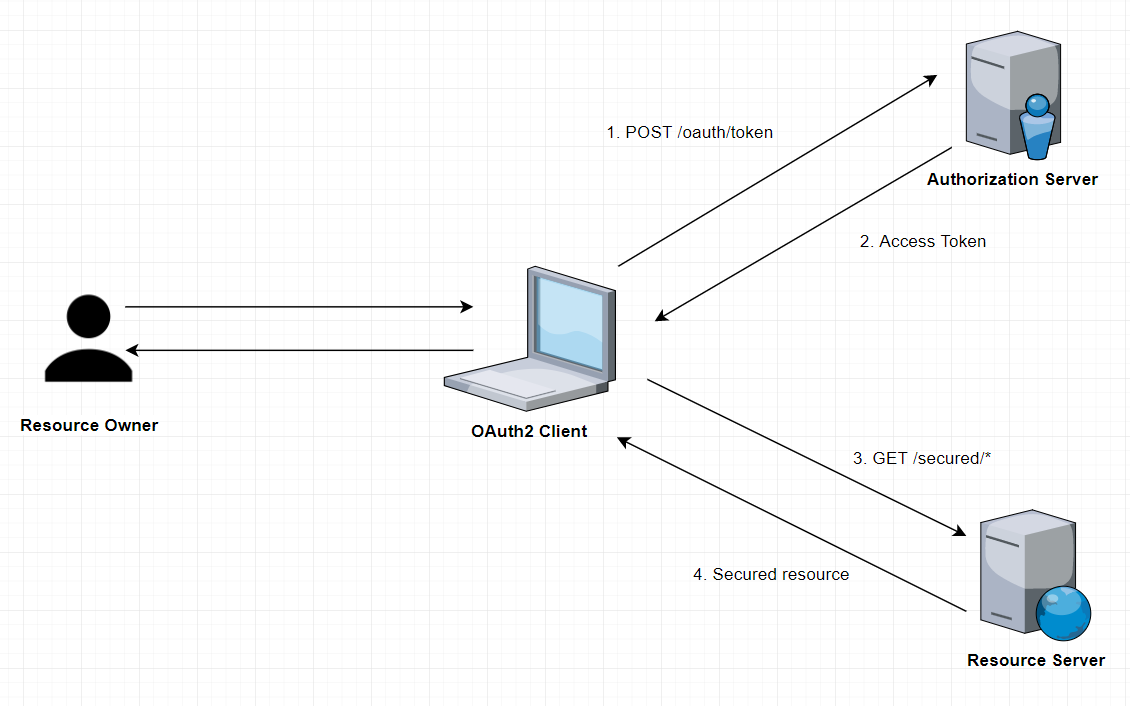


Figure 4: Oauth2 Mechanism

# Cloud Watch

Cloud Watch is a component of Amazon Web Services (AWS) that provides monitoring for AWS resources and the customer applications running on the Amazon infrastructure. On the other hand, our team used Cloud Watch for scheduling daily jobs. For example you can choose Fixed rate of and specify how often the task is to run, or you can choose Cron expression and specify a cron expression that defines when the task is to be triggered. We used scheduled jobs for pulling stock data and calculate user profits daily.

# Front-End

# React

TODO : Hiep&Anurag : Could you add some words here. May be copy/paste presentation things should be enough.

# Mobile

TODO : Hiep&Anurag : Could you add some words here. May be copy/paste presentation things should be enough.

# Expo Framework

TODO : Hiep&Anurag : Could you add some words here. May be copy/paste presentation things should be enough.

# Redux

TODO : Hiep&Anurag : Could you add some words here. May be copy/paste presentation things should be enough.

# CI/CD

TODO : Hiep&Anurag : Could you add some words here. May be copy/paste presentation things should be enough.

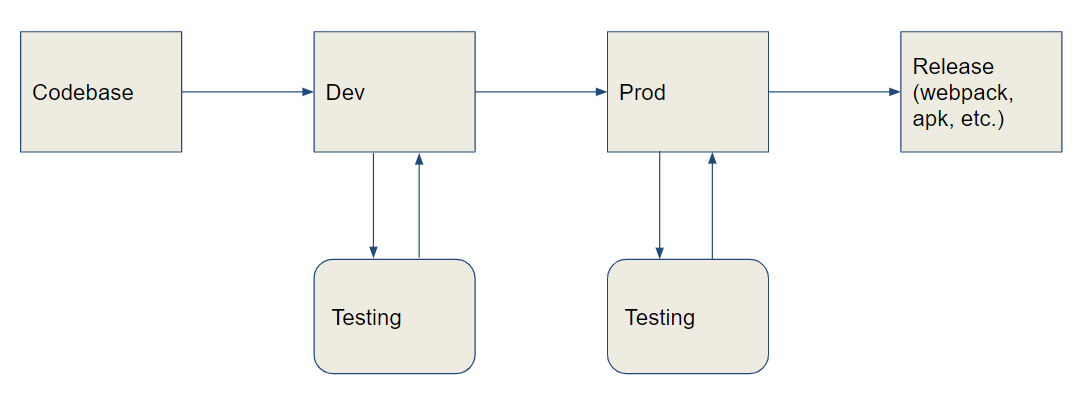
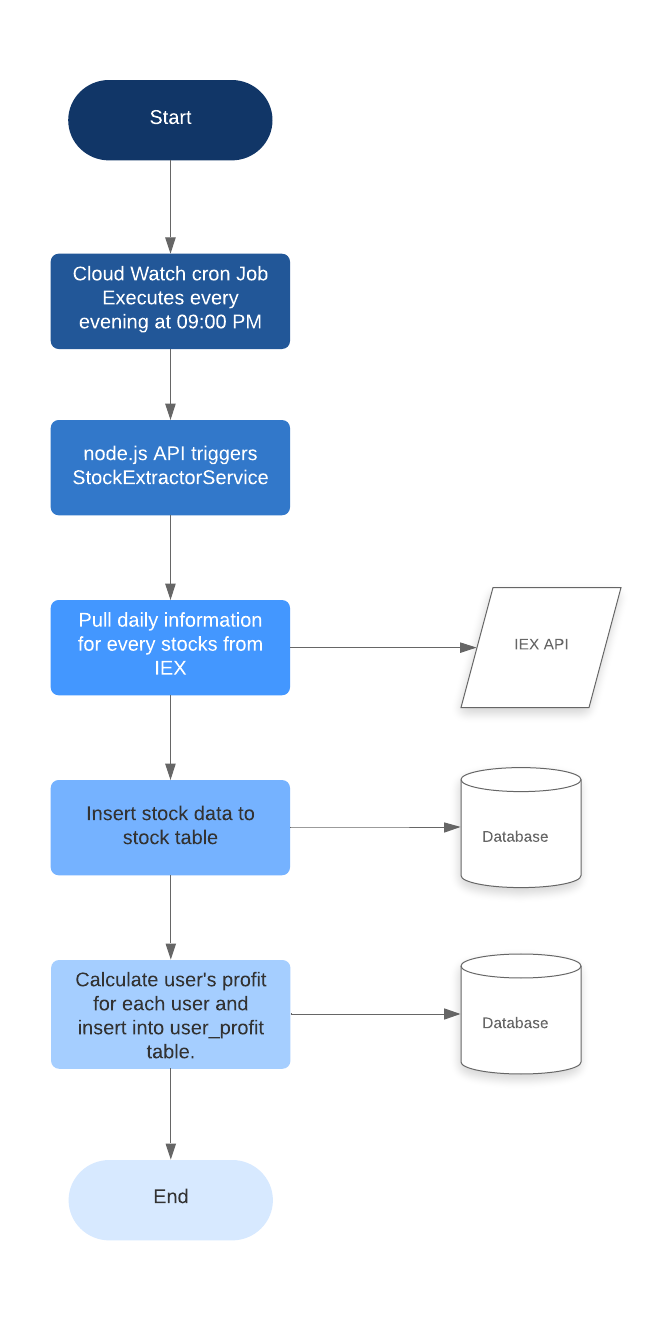


Figure 5: HiAlan CI/CD

# Flows

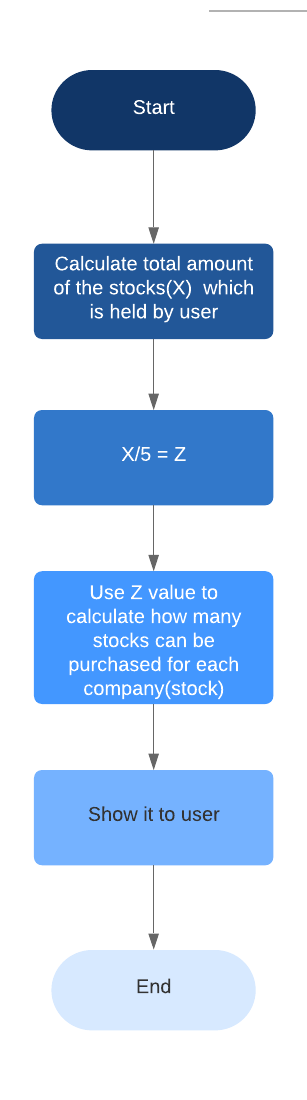
# StockExtractor Flow



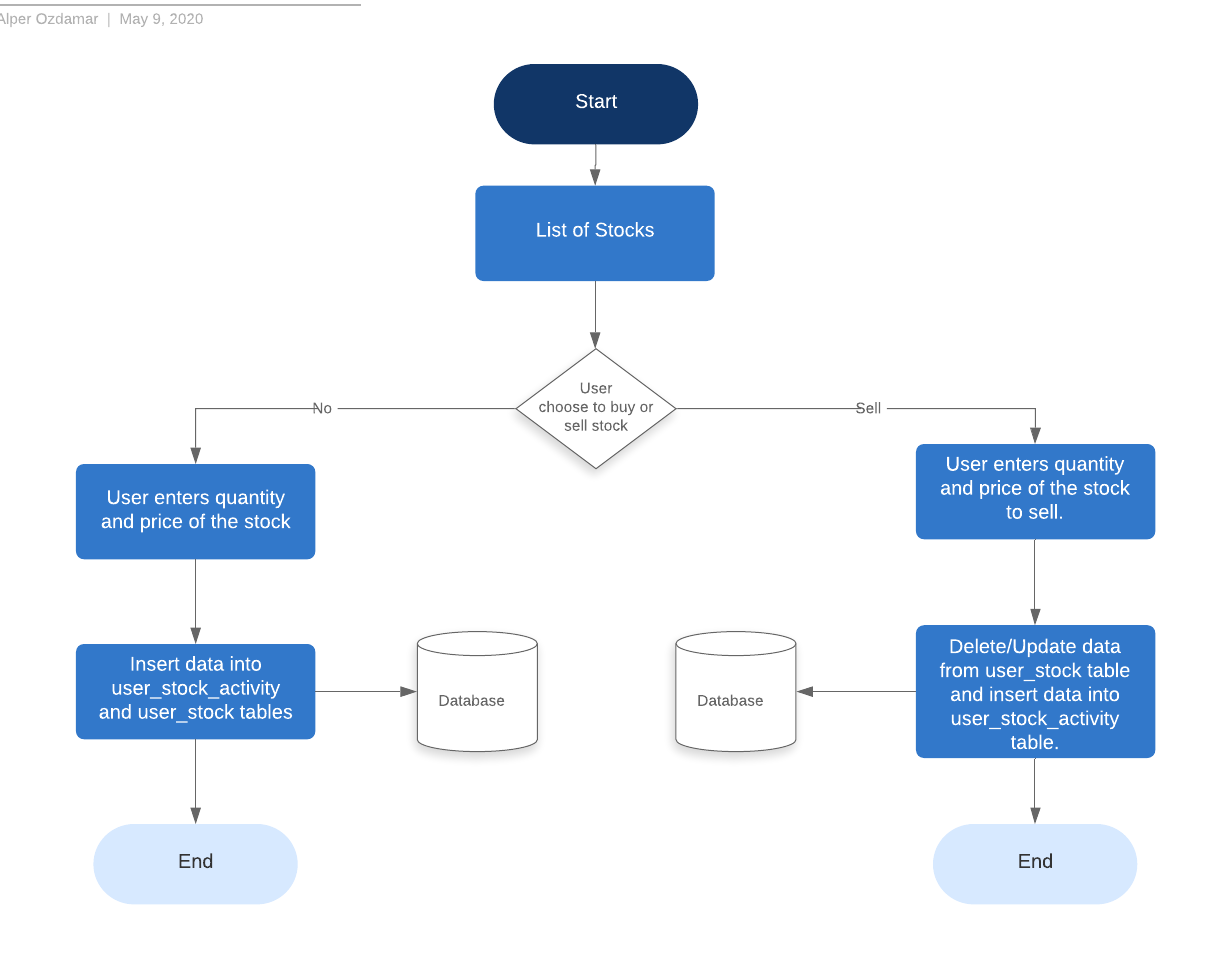
# Login/Oauth2 Flow

@TODO

# Rebalancing Flow



# Buy/Sell Flow



# GUI Pages

Hialan GUI pages given as seen below:

|  |  |
| --- | --- |
| https://lh4.googleusercontent.com/UOc5aHqt5kYu6hRlvHmzLgwZDtf6-C_f_vErvw0ck6Ze0N7RKxotvzdJenhupG6IvE45eUsArpn1kn6uOJEAC1NrLLKCH7oZjY8U-5f_3TqoKF6GHm_A8MYiqoB-X363ppj7iHSlRq8  Figure 3: Home Screen | https://lh3.googleusercontent.com/qsusyp2wx9XHhZ_us6olIpURJzjOPdpszXnffsWyks17yaU-ocyXLEpNeGxynujkNJ3G9VZv1NtswtDT8hRwKEo_2EJrRhzZkd4Sp5WvxeM5FA7ausP4L79ZtVfDTEi8EpLO-OOJBuU  **Figure 4: Invest Screen** |

|  |  |
| --- | --- |
| https://lh6.googleusercontent.com/cISSja0-LjQGz9zOpFA9HTWr7oWB80S3JDCESF4m4e7asKkWt1Fuqf71beLqLIMeY0C4gxcmeXpJLZ35r3EFE7sV9H93NSPKPoU304b2T5yLUv6WfIZWg5xFfktBlfFyE6ewiesIiSE    **Figure 5: Stock Latest Price Screen** |  |

# Logging

Hialan uses Amazon CloudWatch Logs to monitor, store, and access log files from Amazon Elastic Compute Cloud (Amazon EC2) instances, AWS CloudTrail, Route 53, and other sources.

CloudWatch Logs enables to centralize the logs from all of our systems, applications, and AWS services that we use, in a single, highly scalable service. We can then easily view them, search them for specific error codes or patterns, filter them based on specific fields, or archive them securely for future analysis. CloudWatch Logs enables us to see all of your logs, regardless of their source, as a single and consistent flow of events ordered by time, and we can query them and sort them based on other dimensions, group them by specific fields, create custom computations with a powerful query language, and visualize log data in dashboards.