Stock Prediction App

Evaluating Requirements

CS361 HW 3

Open	52.28	Div yield	-
High	52.81	Prev close	51.71
Low	49.47	52-wk high	52.81
Mkt cap	56.07B	52-wk low	19.05
P/E ratio	266.77		

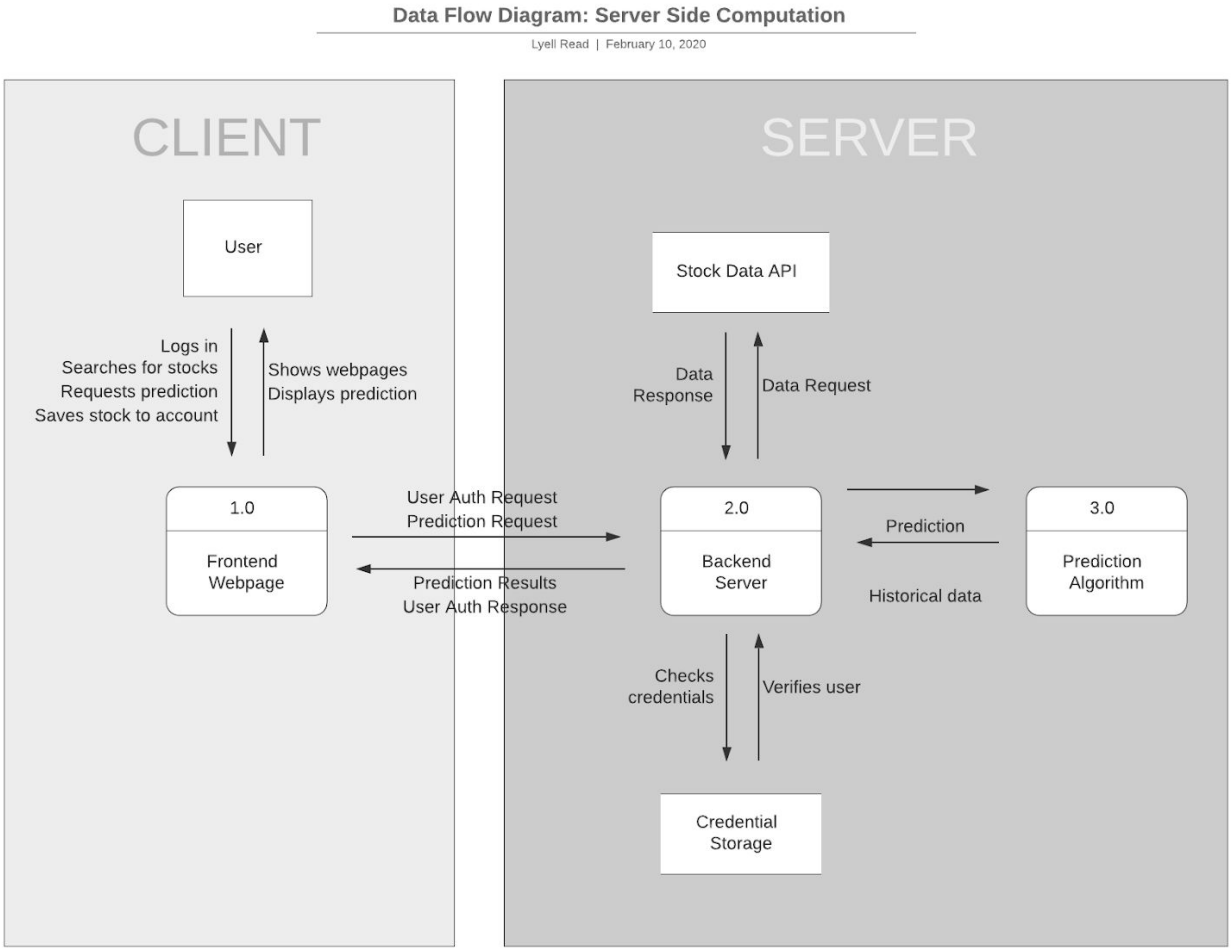
Team 16:

Felix Brucker
Robert Detjens
Remi Kendig
Dominykas Zobakas
Lyell Read

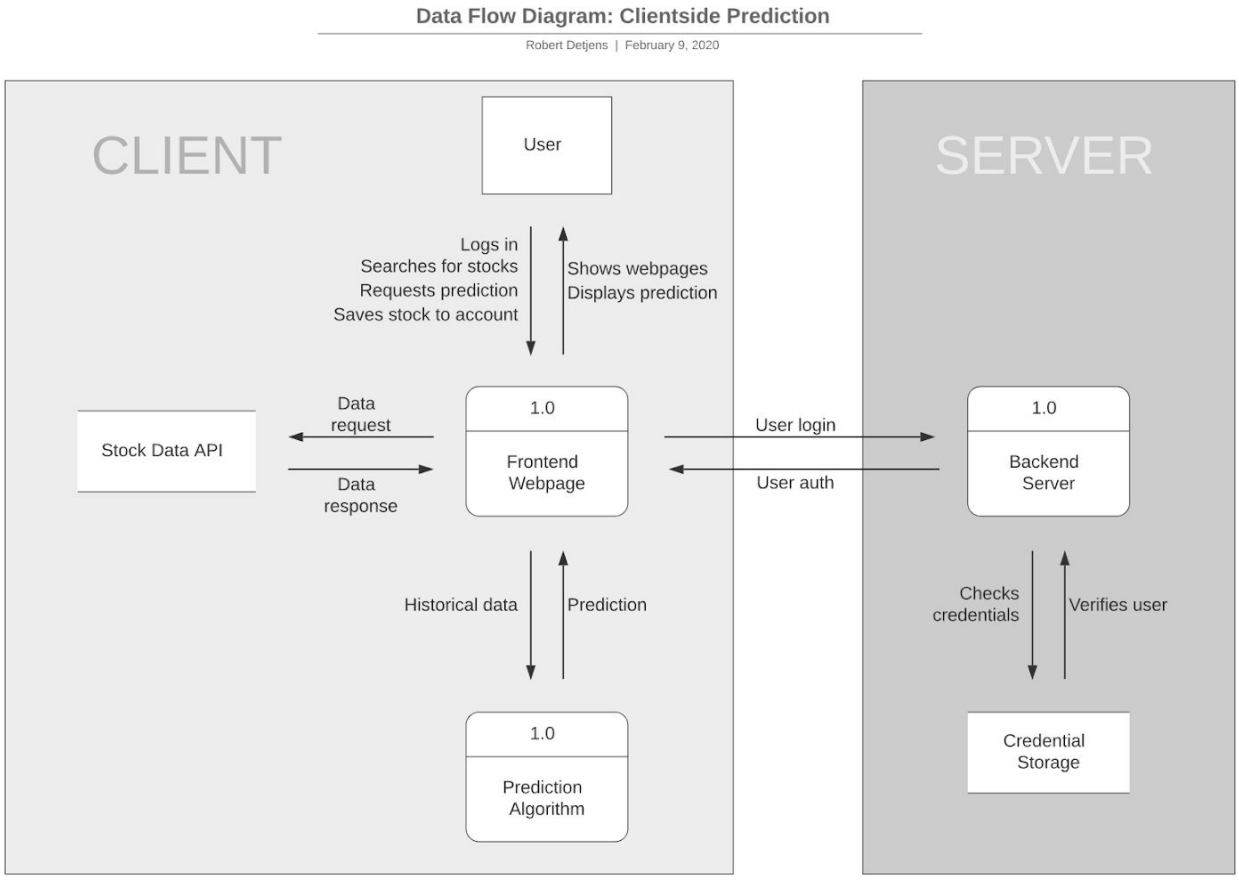
Table of Contents

Data Flow Diagrams

Architecture 1: Server Side Computation



Architecture 2: Client Side Computation Only



Quality Attributes

Reliability

- Architecture 1: The software will be as reliable as the server it runs on and the internet connection of the user. The software will also be subject to the server overloading; if there are too many users trying to access the website at once, it will run much slower.
- Architecture 2: With a client-only architecture, the only reliance on anything external from the web app will be for user login. This means that, even if there are many users using the software at once, it will still function as if only one were using it, aside from logging in.

Efficiency

- Architecture 1: Because the web app will have to communicate with its servers for every action, it may not be as efficient as if the data were stored client-side. However, because the algorithm for predicting stocks is not done on the users' devices, all devices will be able to run the website approximately equally.
- Architecture 2: The software should generally be very efficient, as it does not need to rely on anything external to itself, aside from user authentication. Not needing to communicate with an outside server means load times will be faster and performance will not be tied to what sort of load the server can handle. However, if the user's device or browser is unable to handle running the prediction algorithm, it will function less than optimally.

Integrity

- Architecture 1: The data is subject to the internet connection of the user and user experience is subject to the server's current load. If the server goes down, so does the website.
- Architecture 2: Unless a hacker/cracker can access the server containing user credential information, the client-heavy architecture should have high integrity. Less reliance on sending information over the internet means that it is harder to cause the app to fail.

Usability

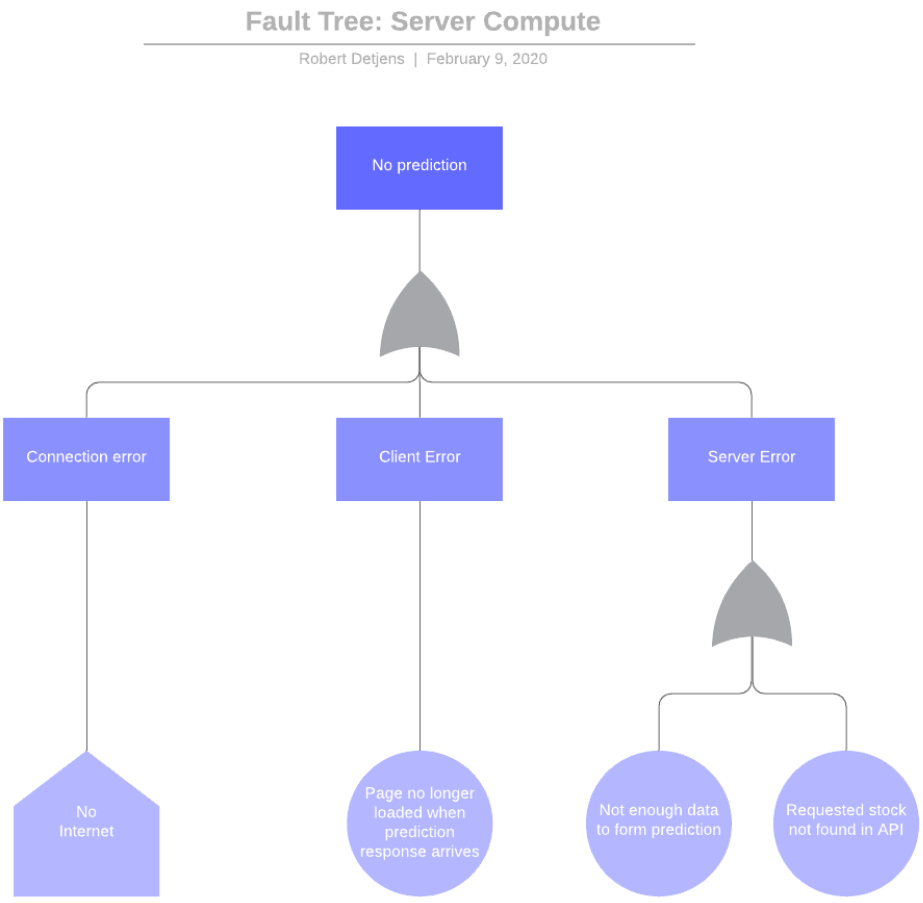
- Architecture 1: Given an effective UI, the software should have high usability. Users should be able to accomplish the tasks of finding stock predictions very easily, as well as saving stocks to their accounts for quick access. Stock information is always up-to-date.
- Architecture 2: Same as above.

Maintainability

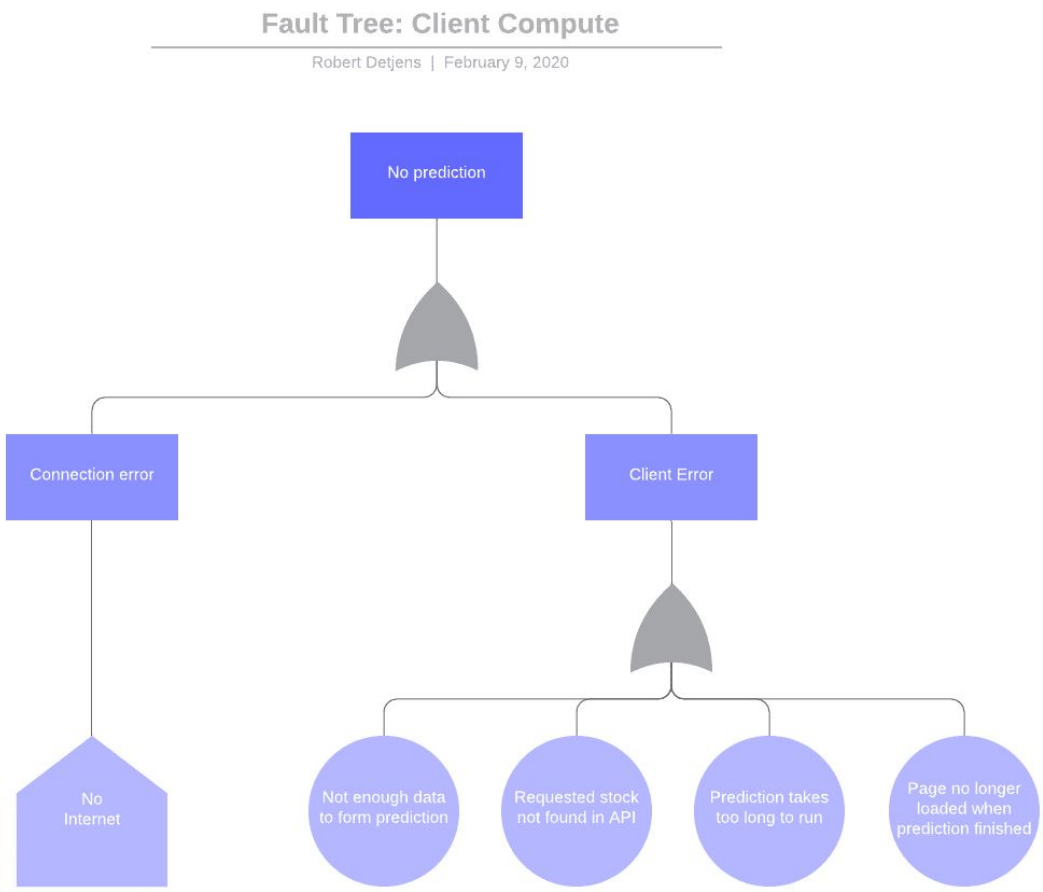
- Architecture 1: The software should be easy to maintain, as communication with the servers each time means only the servers have to be updated with any algorithm changes. Users will not have to wait to sync any new website updates, as it is all handled server-side.
- Architecture 2: The web app should generally be as easy to maintain as any other web app, as pushing updates will simply be the same as updating any other sort of website.

Fault Trees

Architecture 1: Server Compute

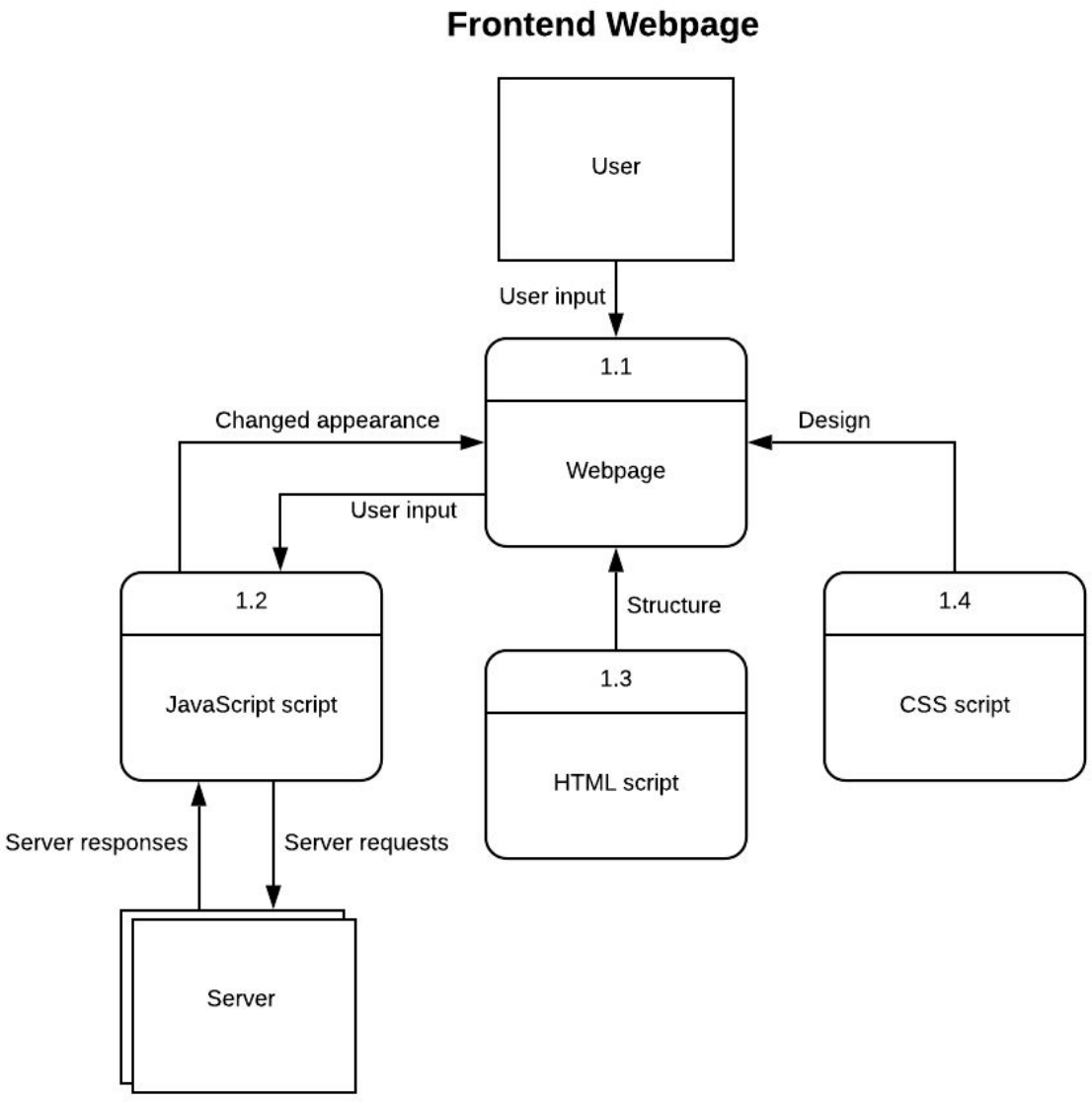


Architecture 2: Client Compute

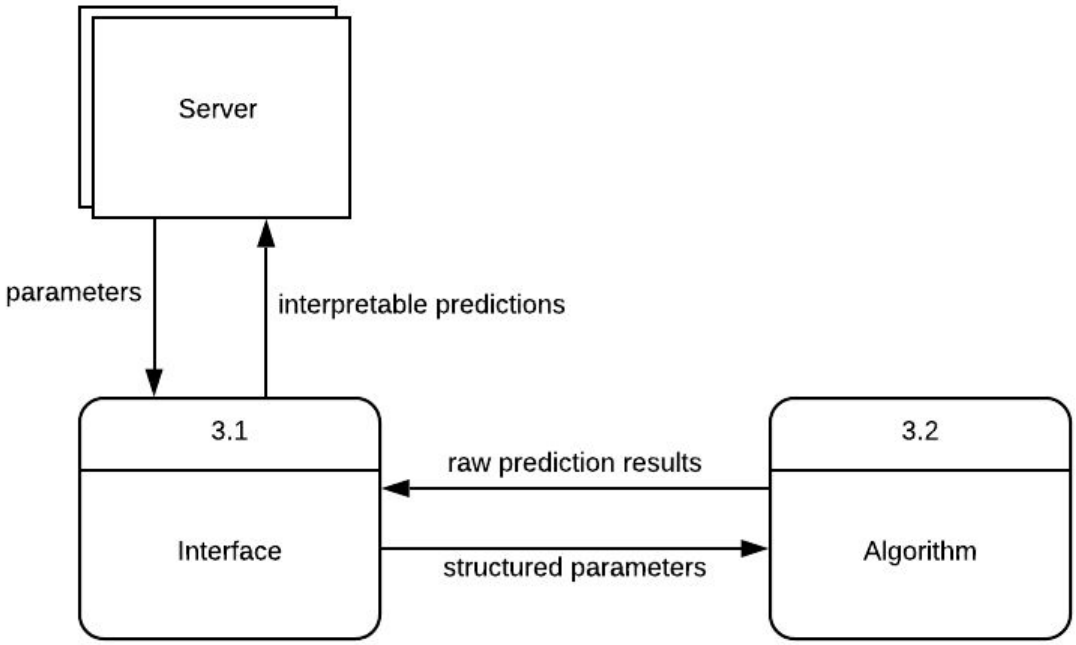


Decomposition

We think that architecture 1 is the right choice for our app. This decision is mostly based on the efficiency quality attribute. Server based computation allows usage from different types of devices e.g. different Operating Systems or mobile devices with less computational power, while providing a consistent user experience. Another feature of the first architecture is that the Stock data is always up-to-date which is very important in the quickly changing environment that the stock market is.
Two important elements of the first architecture are the Frontend Webpage and the Prediction Algorithm.



Prediction Algorithm



Validation

Use Case 1: Saving a Stock to a User's Profile

The User will already be logged in and connected to the web app. They will then search for a stock, either by name, ticker symbol, or category. As they do this, the software will contact the server with a request for that information. Once it finds matching stocks, it will access the prediction algorithm and create a prediction for each stock found. The stocks and their predictions will then be sent to the client and displayed on the web page. The User will then select that they wish to save one of the found stocks, causing the software to send a request to save the stock to its servers. The server will then update the user's account information and send back a success message to the web page. The User will see a success message, and be able to view the stock and its prediction on their dashboard.

Use Case 2: Generating a Prediction for a Stock

The User will already be logged in and connected. They will search for a stock. The web page will send a request for that information to the server, which will send back stock names while the website displays blank graphs in the meantime. Server-side, the server will request stock information from the Stock Data API. When it gets that information, it will send it to the prediction algorithm. The algorithm will process the data and return a prediction for the future of each stock. When the server has all the stock and prediction data, it will send it to the website, and display it, replacing the blank graph it displayed before.

Use Case 3: Creating an Account

The User will open the website in their browser of choice. Because the User will not be logged in, the website will display a button to login or register. The User will click on that button. In a section for registering, the User will enter their desired username, email address, and password. When the user clicks the "register" button, the website will send their account credentials to the server. Because that User's account does not yet exist in the Credential Storage, the server will create a new entry with their username, email, and password. When this is done, a success message will be sent to the website. The User will then be able to login to the website with their username/email and password.

Implications

Considering validation and verification, Architecture 1 is the best choice for the needed capabilities, especially if the central server is built with care and precision, and a wide audience is to be reached.

Use Cases

Saving Stock

Regardless of architecture type, this is data that gets saved to central servers, and is significantly affected by how well central servers are designed. Variety in user device capabilities may still create varying barriers to access this capability, since the command to save a stock has to go through a client-side user interface.

Generating a Prediction

This use case varies the most given architecture type. Using architecture 1, predictions are more up to date and more consistent across devices. The former is especially important, given the changing nature of stocks.

Creating an Account

Like saving stock, this use case is similar in both architectures. However using Architecture 1 implies the same protocols used to protect the security of users login credentials can also be used more readily for other data that passes through the central server in this model.

Quality Attributes

Reliability

Using Architecture 2 transfers responsibility from the central server to individual devices. While this makes central server failures less severe, it puts undue strain on user devices and dis-unifies the user experience. This is why we will go on to use Architecture 1. Building Architecture 1 with a stable central server increases reliability regardless of devices used to access.

Efficiency

Architecture 1 would have generally lower efficiency due to back-and-forth, but that efficiency cost is worth consistency. Any improvements to the central server would also improve the efficiency of the entire system.

Integrity

There are more security concerns over transferring data in architecture 1, however if the central server is built well to handle passwords, which it should regardless of architecture, it should have the same faculties to protect other data.

Usability

Both designs have high usability when executed right, but Architecture 1 has a more consistent usability that improves when the central server is improved.

Maintainability

In architecture 1 updates to the central server are more prominent. This consolidates the maintenance more than the other model. As a live service the possibility for quick change and improvement is ripe regardless.

Failure Potential

In Architecture 1, responsibility for faults is more distributed between server and client, as opposed to Architecture’s client-heavy model. This presents more potential vectors for failure, but puts less liability on the client side. This puts less strain on a partially uncontrolled environment that depends on the user’s device. When we make the central server more robust in this scenario, it also improves a greater portion of the error tree.

Contributions

Customer

HW1

- We met with our customer Ghaith Shan after class on 1/23/2020.
- We have maintained communication with our customer on Discord since 1/16/2020.

HW2

- We met with our customer Ghaith Shan after class on 1/30/2020 to present the paper mockups.
- We discussed requirement revisions and went over the mockups with our customer over Discord on 1/31/2020.

HW3

- We did not meet with our customer as we had already verified that we share a common understanding of the goals and requirements for this app so far.

Team members

HW1

- Felix Brucker: Document Creation, Data Flow Diagram, Proofreading
- Robert Detjens: Functional Specifications, Non-functional Specifications, ERD Revision, Use Case 2, Use Case 2 MSD
- Remi Kendig: Use Case 1, Data Flow Diagram
- Dominykas Zobakas: ERD Revision, Use Case 1 MSD
- Lyell Read: ERD First Draft, Document Formatting, Customer Contributions, Functional Definitions, Non-Functional Definitions, Use Case 3, Use Case 3 MSD

HW2

- Felix Brucker: Mockup proofing, Use Case 1 Revision
- Robert Detjens: Paper mockups, Mockup meeting w/ customer, Definition revision
- Remi Kendig: Changes made since HW1
- Dominykas Zobakas: MSD Revisions, ERD revision
- Lyell Read: Mockup proofing, Paper -> Digital mockup drawings, Appendix, Corrections to UC1 & UC3 MSDs.

HW3

- Felix Brucker: Implications
- Robert Detjens: Architecture 2 Data Flow Diagram, Fault Trees, Document Formatting
- Remi Kendig: Quality Attribute Assessment, Architecture Validation
- Dominykas Zobakas: Decomposition
- Lyell Read: Architecture 1 Data Flow Diagram, Team Member Contributions