1. Project name:

   YggFinance

2. Team member names:

   Blake Hudson, Alan Holman, and Austin Kerr

3. Abstract:

Personal Finance is an important topic for most individuals in modern times. The common person may find it difficult to know where to begin with finance tracking. Our group intends to produce an easy-to-use Web Application (Web App) called YggFinance to help alleviate this problem. YggFinance will help our end users gain a better grasp on three major areas of their personal finance ー Monthly Budgeting, Savings Planning, and Net Worth Tracking. We intend to develop tools for each of these categories that will make it easier for our users to track personal spending trends, to compare strategies for personal savings goals, and to gain insight into their current net worth. If time allows, we would also like to develop an additional tool that will allow users to track their various investments, though this is mostly outside of our current project scope at the time of this proposal.

4. Description:

The main goal of the YggFinance Web App is to serve as a template for users to track and plan their own personal finance journey. To accomplish this, we will design YggFinance with simplicity in mind for the user. We intend to have each finance category make clear indications for the user of what information is needed to plan specific aspects of their financial journey. We will implement a system to allow for user-reported spending and saving information. To rectify any possible discrepancies in user-inputted data, we will also implement a system where users can input data from their financial institutions. Users should be able to attain a solid understanding of their current financial state while also having the necessary tools at their disposal to plan for future financial goals.

We plan on supporting our front-end Web App on the back-end server by utilizing three main Microservices that parallel the tools we will be offering on our Web App. These Microservices are the Budgeting Service, the Planning Service, and the Net Worth Service. Each of these services will exclusively serve their own tool on the WebApp to allow for each tool to operate independently of each other. This will allow us to easily accommodate for variance in how much our users utilize different tools while it will simultaneously give our developers a high level understanding of how our system will work. For a more technical specification on how our entire system will be laid out, please see our System Topology Diagram depicted in Figure 1 and Figure 2 located in the Technology section.

The monthly budgeting page in YggFinance will have four main features ー categories, individual transaction tracking, monthly transaction reconciliation, and transaction history. The categories feature allows for the user to create various categories for categorizing transactions. The individual transaction tracking feature will allow the user to track each individual transaction they make throughout the month. The monthly transaction reconciliation feature on the other hand will allow the user to match their tracked transactions with the actual transactions from their bank statement. Finally, the transaction history feature will allow the user to view various aspects of their monthly budget through a given time period.

The monthly budgeting page will primarily show information on where the current month's spending in various categories stands in relation to the amount the user has allotted for that category. The user will have the option to define their own categories as well as determining how many they would like to see on the monthly budgeting page. When defining categories, the user can define the category to be either an income (money coming in) or an expense (money going out). Additionally, while defining the category the user will be able to set a budget goal for how much they intend to spend in that category for the month. Each time the user enters a transaction, YggFinance will display to the user their remaining budget in that category so that the user will always be aware of how much budget they have left.

For the individual transaction tracking feature, the user will be required to enter their individual transactions into the application manually throughout the month. This serves a dual purpose of maintaining user accountability for individual transactions as well as providing a basis of comparison for the monthly transaction reconciliation feature. For simplicity, these transactions will simply consist of the name of the merchant/institution the transaction was made with, the amount spent/received, and the category it falls under.

For the monthly transaction reconciliation feature, the user will be able to upload a .csv file from their financial institution(s) for the month to reconcile. Using the .csv file, YggFinance will be able to match the transactions from the financial institution to the individually tracked transactions entered by the user. YggFinance will initially attempt to match transactions based on the amount spent/received though, if there are multiple transactions of the same amount, it will also attempt to match by merchant/institution. YggFinance will expect all transactions to be accounted for on both the user tracked and financial institution side. If there are any discrepancies between the two, YggFinance will report these to the user and expect the user to reconcile the discrepancy. This process ensures both that YggFinance maintains an accurate history of transactions, as well as making the user aware of any possible tracking mistakes or fraudulent transactions on their account.

The transaction history feature will allow the user to view various aspects of their monthly budget through a given time period. After selecting the desired time period, the user will then select certain inputs to add or omit certain pieces of data about their monthly budget. These inputs will include inputs for viewing transactions greater than or less than a specified amount, comparing total spent in selected categories, as well as viewing transactions grouped by merchant/institution. The user will also be able to change the date range of their transaction history in case they would like to view their transaction history over a longer span of time.

Aside from the monthly budgeting page, YggFinance will also help the user reach their saving goals through the use of its savings planner on the Planning page. The user can enter their current investment balances, rate of return, expected contribution, and desired investment income into YggFinance's savings planner. Using this information, the savings planner will be able to give the user a time-frame on when their goal will be met. The savings planner can also suggest how much more the user needs to contribute in order to reach their goals if they are not on track to reach it in their desired timeline. Goals for the savings planner can be either a one time saving goal, a recurring savings goal, or an income based savings goal. Examples of these include saving a down payment for a house as a one time goal, buying a software subscription for a recurring goal, and reaching your Financial Independence Retire Early(FIRE) number for an income based goal.

Finally, The user will also have access to a net worth tracker available on the Net Worth page. This will give users a better understanding of their approximate worth by taking user input of both assets and liabilities. A net worth profile contains assets such as a home, vehicles, stocks and bonds, financial account savings, and tangible assets. The net worth profile will also take input for users' liabilities such as credit card debts, loans, mortgages, and more. Users will be able to view assets and liabilities as appropriate subcategories, allowing users to determine how much of both assets and liabilities are made up of categories such as cash and properties (assets) or loans and credit cards (liabilities). To account for changes in values of the net worth's elements, items will be able to be retroactively changed by the user in order to update the net worth to the correct value. The net worth tracker will be a long-term tracker for users, allowing them to see progress over the course of their lifetimes. This function also allows users to track overall liability value and asset value to make educated decisions about how to minimize risks while maximizing growth.

5. Feature list

Features to be complete by the end of the semester:

- Transaction tracking
- Monthly budget reconciliation
- View transaction history
- Custom categories for organizing the monthly budget
- Savings goal planner
- Net worth tracking

Feature to be completed if there is time:

- Connect to user accounts using Plaid API
- Debt payoff tracker

Feature that cannot be implemented, but would like to:

- Tailored investment advice for the user
- Portfolio rebalancing
- Email notifications for large purchases or payday

6-7. Technology and Server Information

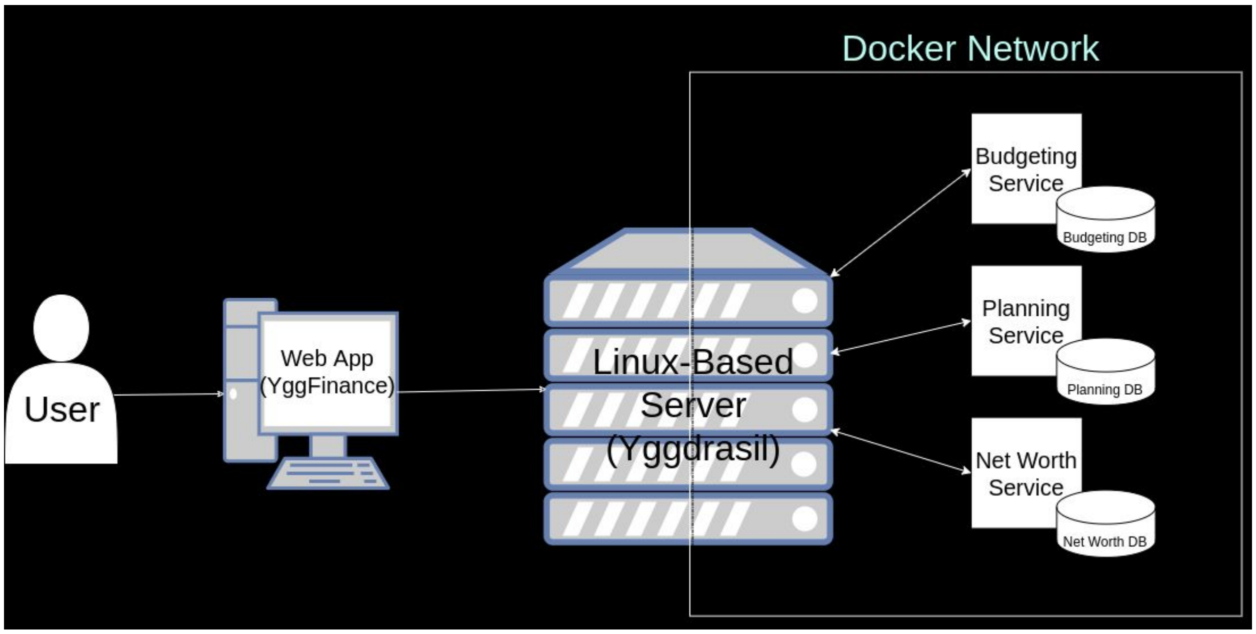
Our project consists of a Linux-based Server side with Containerized Microservices running in Docker, and a Client side WebApp using React.js accessible from a Web Browser. We plan to deploy the microservices onto a Docker Network running on a Linux-based Server to serve as the main backend of our frontend Web App. Regarding the frontend Web App, we intend to use the React.js(v.17.0) JavaScript library due to its high industry use as well as its effectiveness in developing Web Apps. We intend to develop the frontend as a Web App as this allows us to produce an application that can be accessed across many different user system configurations therefore increasing our potential user base.

For the backend, we have chosen to develop using Containerized Microservices due to the high industry use of the architecture as well as the flexibility it gives us in developing with multiple different Programming Languages concurrently. For our implementation of the Containerized Microservices, we are opting to use Docker, as it provides us with flexibility in server setup and application deployment in addition to being one of the most commonly used technologies we've researched. Docker Networks provide a layer of abstraction over the server logic meaning that the execution of the program does not directly depend on the server configuration it is running on. This allows us to be flexible with our choice of server hosting platforms without being overly tied to one platform and is the primary reason for our choice in using this technology. We may consider utilizing a cloud computing provider such as Amazon Web Services(AWS) for Deploying our Docker Network, though currently the member in charge of the server is looking into setting up a personal linux server so this may be unnecessary.

In order to limit the number of new technologies our team members must acquire to succeed, we have decided to work in the programming languages that are the most familiar to us

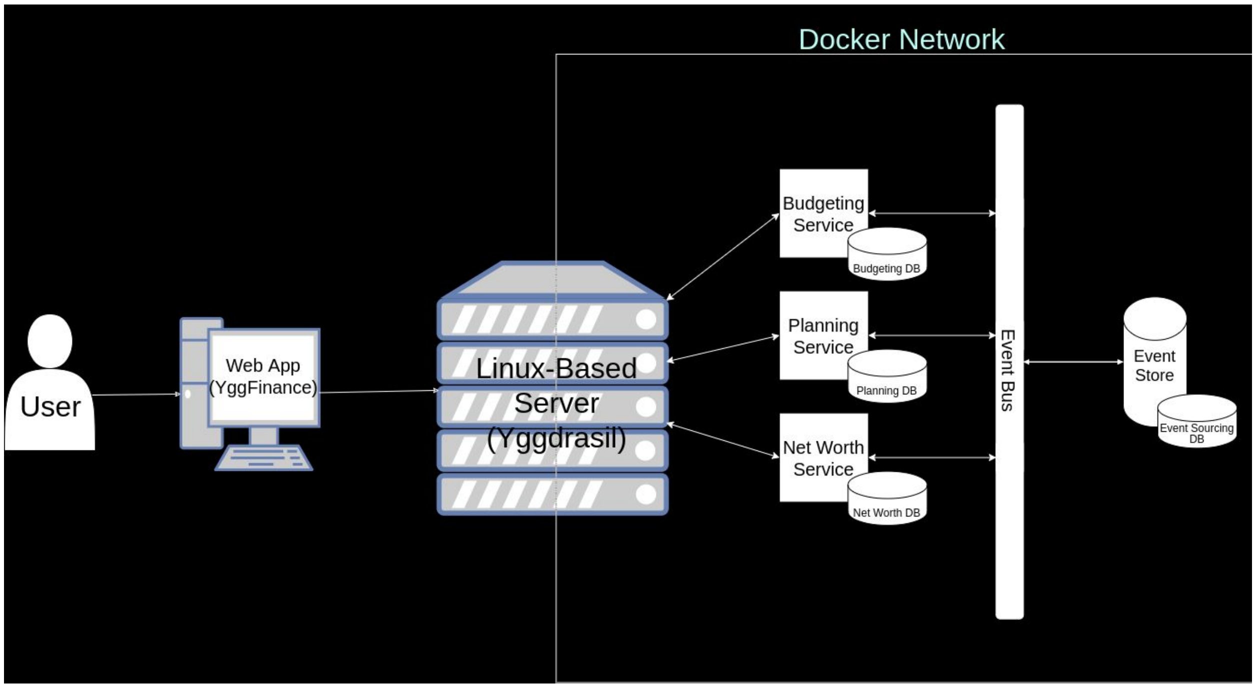
— Java(v.14.0), Python(v.3.9), and JavaScript(ES6). In order to properly deploy our back-end code, we will also be using Docker(v.20.10) along with Representational state transfer (REST) Application Programming Interface (API) calls. These REST API calls will be facilitated through the use of Spring Boot(v.2.4) for Java, Nginx(v.1.19) for Python, and Node.js(v.14.15) for JavaScript. In order to manage the data our users enter into the application, we will be using a Database per Service Pattern with each database being written in SQLite(v.3.34) for simplicity. See Figure 1 below for a System Topology Diagram (created using app.diagrams.net) detailing how our planned system will be laid out.

Figure 1. A Simple full System Topology Diagram for the YggFinance System



Depending on our need, we may also need to incorporate additional concepts and technologies such as an Event Bus using RabbitMQ(v.3.8) and an Event Store using MongoDB(v.4.4). An Event bus is an architectural pattern that allows services to interact with one another without depending on each other for correct operation. Rather than Services calling each other’s APIs and forming a dependency, Services can subscribe to and publish certain events that might occur in the application to the Event Bus. The job of the Event Bus is to send incoming published events to the services that are subscribed to those events. As a result, services need not know how to access each other or even if the other services are even running at the same time at all in order to function correctly. The Event Store on the other hand is another architectural pattern that is essentially a database for storing previous events. This allows for an application to keep track of all previous actions for auditing purposes as well as potentially roll back the application to an earlier state if something goes wrong. See Figure 2 below for a full System Topology Diagram should we need to utilize these additional technologies.

Figure 2. An Advanced full System Topology Diagram for the YggFinance System



For similar reasons to our programming language choice, we are opting to each use Integrated Development Environments (IDEs) that we are familiar with or that are best optimized for our programming language and technologies. These are Netbeans(v.12.0) for Java, and VSCode(v.1.52) for Python and JavaScript. For version control, we will be using git(v.2.30) with some members opting to use TortoiseGit(v.2.11) for a windows shell interface. The platform our team will be using to host our git repository is GitHub as this was one of the Requirements that we were given for developing this project. Our development team is primarily using Windows 10 systems for development, though the member in charge of server configuration will be using a Linux system for development instead. For team communication, our team has agreed to utilize Discord as our primary communication platform and will utilize other technologies such as Email and Short Message Service (SMS) messaging as sparingly as possible.

8. Data Sources

The User will input nearly all of the data we will be using through the Web App of their own accord. For the monthly transaction tracking and net worth tracking services, the user will be submitting csv files that they can obtain from their financial institutions through bank statements. Assuming we have time, we may utilise Plaid API in order to gather user financial institution data as a means to collect the data with live updates. Additionally, if we find ourselves with extra project time, we also hope to use Alpha Vantage API to send live updates for stocks connected to the users' savings planner profile.

9. Team members' backgrounds

Austin Kerr will work mainly on the frontend, creating an interface for users to interact with the Web App. Austin has some experience in Visual Studio with C++ and C#. He also has some experience with Java using Netbeans, but for this project he will use React.js and Node.js through VSCode.

Blake Hudson has a background in C++ and C#/.net. As it pertains to this project he also has experience using Java and NetBeans, which will be used for his main contribution on this project. Blake also has limited experience with Python. Blake will be designing backend functionality to support the main features of YggFinance, like the monthly budgeting, net worth tracker, and savings planner.

Alan Holman is proficient with a large variety of programming languages including every programming language to be used for this project. In addition he has experience working with Microservices and Web Apps as well as with setting up and configuring Linux-Based Servers. Alan will primarily be focused on coordinating the various technologies used throughout the project and will aid the other team members in development across the Full Stack.

10. Dependencies, limitations, and risks

One main limitation for our group completing this project on time is Blake and Austin's experience level with Docker and the front end technologies. In particular React.js and Node.js are completely new technologies to these members and fairly new to the remaining member as well. One way we plan to overcome this limitation, should issues arise, is to utilize all of the resources we have available in order to overcome our shortcomings. If despite this effort we determine that we will not be able to meet our project goals by the deadline, we may consider decreasing our project scope or changing technologies in order to ensure we meet the deadline.

There is also a minor risk that some technologies and platforms we have chosen may not integrate as well as we are expecting. In order to mitigate this risk, we plan to remain flexible in our choice of technologies and platforms so that, should this happen, we will be able to quickly adapt our technology choice to meet our needs. Luckily, The Microservices Software Architecture affords us this flexibility to change technologies to meet our needs so this greatly mitigates the chance of this risk occurring.

11. Timeline

Week	Description
02/01/2021 - Week 01	Setting up development environments
02/08/2021 - Week 02	Familiarizing ourselves with the technologies we will be using
02/15/2021 - Week 03	Backend: Initial template Docker Images running in all dev. envs.; Frontend: Initial template WebApp running
02/22/2021 - Week 04	Backend: Setup SQLite Databases in all 3 Services Frontend: Create 3 separate pages for each tool in the WebApp
03/01/2021 - Week 05	Backend: Begin work on Budgeting Service Begin work on Planning Service Frontend: Able to Navigate between the 3 pages in a menu
03/08/2021 - Week 06	Backend: Continue work on Budgeting Service Continue work on Planning Services Frontend: Begin work on the Budgeting Tool Page
03/15/2021 - Week 07	Backend: Continue work on Budgeting Service Continue work on Planning Services Frontend: Continue work on Budgeting Tool Page
03/22/2021 - Week 08	Backend: Continue work on Budgeting Service Complete work on Planning Service Frontend: Complete work on Budgeting Tool Page
03/29/2021 - Week 09	Server: Set up a Linux-Server with Docker Backend: Complete work on Budgeting Service Frontend: Begin work on Planning Tool Page
04/05/2021 - Week 10	Server: Test Server config by deploying a template Docker network Backend: Begin work on Net Worth Service Frontend: Complete work on Planning Tool Page
04/12/2021 - Week 11	Server: Deploy Budgeting & Planning Services to Server Backend: Continue work on Net Worth Service Frontend: Begin work on Net Worth Tool Page
04/19/2021 - Week 12	Server: Deploy all Services to Server Backend: Complete work on Net Worth Service Frontend: Complete work on Net Worth Tool Page
04/26/2021 - Week 13	Implementation and testing
05/03/2021 - Week 14	Presentation Week!