Project Requirements

Due 2/8/2020, 9:00 AM

1. **Project Name:** YggFinance
2. **Group Members:** Alan Holman, Blake Hudson, 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 ─ 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. **Tools & Technologies:**

Our project consists of a **Linux-based Server** side with **Containerized Microservices** running in **Docker**, and a Client side **WebApp** 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 our current plan is to set 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)** and **JavaScript(ES6)**. In order to properly deploy our back-end code, we will also be using **Docker(v.20.10)** along with **Hyper-text Transfer Protocol (HTTP) Representational state transfer (REST) Application Programming Interface (API)** calls. Additionally, the HTTP message type we have decided on is **JavaScript Object Notation (JSON).** The REST API calls will be facilitated through the use of **Spring Boot(v.2.4)** for Java and **Node.js(v.14.15)** for JavaScript. In order to manage the data our users enter into the application We will be using **Client-Side Cookies** As this allows us to maintain a **Stateless Server**, greatly simplifying our server design and removing the need for a user authentication system, databases, Event Stores, and an Event Bus. See Figure 1 in the diagrams section for a **System Topology Diagram** (created using **app.diagrams.net**) detailing how our planned system will be laid out.

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 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.

5. **Requirements list:**
    1. **YggFinance Web App**
        - 1.1. The User will be able to **connect to** the YggFinance Web App through a URL in any modern Web Browser.
        - 1.2. The YggFinance Web App will use the Material UI **style library** for displaying all of its elements to the User.
        - 1.3. The YggFinance Web App will contain 4 Pages accessible through tabs located in a **Top Menu**: The **Welcome Page**, The **Savings Planner Page**, The **Monthly Budget Page**, and The **Net Worth Tracking Page**.
            - 1.3.1. The **Top Menu** should be a default Material UI TabPanel that will be available to the User at the top of the Web App at all times.

1.4. Upon connecting, the YggFinance Web App will open to the **Welcome Page**.
    1.4.1. The **Welcome Page** will contain a short description of each other page in the **Top Menu** with their respective uses and tools.
1.5. The YggFinance Web App will **store User Collected Input Data** on the user's machine as a cookie.
    1.5.1. No User Information will be stored on the Yggdrasil Server between sessions. (Server is Stateless)

2. Savings Planner Page

2.1. The Savings Planner Page will have an **initial investment field** for the user to input data into.
    2.1.1. The Initial investment field will be **labeled**: "Initial investment (USD)".
    2.1.2. The Initial investment field will **validate user input** to be only non-negative numbers in US currency form.
        2.1.2.1. If user input does not match the required format or the input is empty, the Web App will display a ***Invalid Input Dialog*** (See Req. 5.1) to the user explaining such.
2.2. The Savings Planner Page will have an **average rate of return field** for the user to input data into.
    2.2.1. The average rate of return field will be **labeled**: "Average rate of return (%)".
    2.2.2. The average rate of return field will **validate user input** to be only non-negative decimals in percentage form. (e.g. The user enters 7.02% as a percentage, this translates to the decimal form 0.0702 for the calculations).
        2.2.2.1. If user input does not match the required format or the input is empty, the Web App will display a ***Invalid Input Dialog*** (See Req. 5.1) to the user explaining such.
2.3. The Savings Planner Page will have a **monthly contributions field** for the user to input data into.
    2.3.1. The monthly contributions field will be **labeled**: "Monthly contributions (USD)".
    2.3.2. The monthly contributions field will contain a **default value** of 0.
    2.3.3. The monthly contributions field will **validate user input** to be only non-negative numbers in US currency form.
        2.3.3.1. If user input does not match the required format or the input is empty, the Web App will display a ***Invalid Input Dialog*** (See Req. 5.1) to the user explaining such.
2.4. The Savings Planner Page will have a **Planning Mode Switch** for the user to toggle.
    2.4.1. The Planning Mode Switch will **default** to the left position.
    2.4.2. The Planning Mode Switch should have **3 Labels**.

2.4.2.1. The **Top Label** of the Planning Mode Switch should reflect the purpose of the switch.

2.4.2.1.1. The **Top Label** will **display**: “Planning Mode”.

2.4.2.2. The **Side Labels** of the Planning Mode Switch should reflect the Type of Output that the Savings Mode Planner will give upon pressing the Calculate Button.

2.4.2.2.1. The **Left Label** will **display**: “Time Frame”.

2.4.2.2.2. The **Right Label** will **display**: “Savings Goal”.

2.5. The Savings Planner Page will have a **Time-Frame <u>Slider</u>** for the user to adjust.

2.5.1. The Time-Frame Slider will be **labeled**: “Years to grow”.

2.5.2. The Time-Frame Slider will **default** to a value of 10 years.

2.5.3. The Time-Frame Slider will have a **valid value range** beginning at 1 year and ending at 80 years.

2.5.4. The Time-Frame Slider will **be enabled and visible** when the *Planning Mode Switch* is in the **Right** Position.

2.5.5. The Time-Frame Slider will **be disabled and hidden** when the *Planning Mode Switch* is in the **Left** Position.

2.6. The Savings Planner Page will have a **Savings Goal Text Field** for the user to input data into.

2.6.1. The Savings Goal Text Field will be **labeled**: “Savings Goal (USD)”.

2.6.2. The Savings Goal Text Field will **validate user input** to be only non-negative numbers in US currency form.

2.6.2.1. If user input does not match the required format or the input is empty, the Web App will display a ***Invalid Input Dialog*** (See Req. 5.1) to the user explaining such.

2.6.3. The Savings Goal Text Field will **be enabled and visible** when the *Planning Mode Switch* is in the **Left** Position.

2.6.4. The Savings Goal Text Field will **be disabled and hidden** when the *Planning Mode Switch* is in the **Right** Position.

2.7. The Savings Planner Page will have a **Calculate Button** for the user to press.

2.7.1. The Calculate Button will be **labeled**: “Calculate”.

2.7.2. The Calculate Button will **be disabled** by **default**.

2.7.3. The Calculate Button will **be enabled** when the following Text Fields contain **valid data**:

2.7.3.1. The Initial Investment Text Field

2.7.3.2. The Average Rate of Return Text Field

2.7.3.3. (If Enabled and Visible): The Savings Goal Text Field

2.7.4. When Pressed, The Calculate Button will **send a request** to the *Savings Planner Service* *(See Req. XXX)*.

2.7.4.1. The **request** should contain all of the data that the user has input into the Savings Planner Page.

2.7.4.2. When YggFinance **receives a response** from the *Savings Planner Service* *(See Req. XXX)*, YggFinance will update the *Results Table* *(See Req. 2.8)* with the data contained within the response.

2.8. The Savings Planner Page will have a **Results Table** to display results to the User.

2.8.1. The Results Table will be **labeled**: “Results”.

2.8.2. The Results Table **Layout** will have No Header, 2 Columns, and 4 Rows.

2.8.2.1. **Column 1, Row 1** will be **labeled**: “End Balance:”.

2.8.2.2. **Column 1, Row 2** will be **labeled**: “Starting Amount:”.

2.8.2.3. **Column 1, Row 3** will be **labeled**: “Total Contributions:”.

2.8.2.4. **Column 1, Row 4** will be **labeled**: “Total Interest:”.

2.8.2.5. **Column 2, Rows 1-4** will be **formatted** in US Currency form.

2.8.2.6. **Each Row in Column 2** will **display data** associated with the label on the same Row in Column 1. (See Req. 2.7.4.2)

3. Monthly Budget Page

3.1. **Currently Displayed Month Select**

3.1.1. Defaults to current month.

3.1.2. Changes which month’s data to be displayed in *Categories Collapsible Table* (See Req. 3.2).

3.2. **Category Collapsible** EditTable

3.2.1. 4 Columns

3.2.1.1. Column 1: No Header

3.2.1.1.1. Each Row: Collapse Component

3.2.1.1.1.1. Default State: Not Hidden & Collapsed

3.2.1.1.1.2. Inside Collapse Component: Transaction Table (See Req. 3.3)

3.2.1.2. Column 2: “Category” Header

3.2.1.2.1. Each Row: Name of Category (TextField)

3.2.1.2.1.1. Default: “Unnamed”

3.2.1.2.1.2. Validation: Cannot be Empty

3.2.1.3. Column 3: “Spent” Header

3.2.1.3.1. Each Row: Amount Spent in USD (TextField)

3.2.1.3.1.1. Value: Adjusted Sum of all Entries in Column 4 of the Transaction EditTable (See Req. 3.3.1.5).

3.2.1.3.1.1.1. Adjustment: For each Row, Column 4’s number becomes negative if the entry in Column 3 is Checked.

3.2.1.3.1.2. Validation: Cannot be Edited

3.2.1.3.1.3. Format: USD Form (1.2345 -> $1.23)

3.2.1.4. Column 4: “Planned” Header

3.2.1.4.1. Each Row: Amount Planned in USD (TextField)
3.2.1.4.1.1. Default: $0.00
3.2.1.4.1.2. Validation: Must be a Non-negative Number
3.2.1.4.1.3. Format: USD Form (1.2345 -> $1.23)

3.3. Transaction EditTable

3.3.1. 5 Data Columns
3.3.1.1. Column 1: “Name” Header
3.3.1.1.1. Each Row: Name of the Transaction (TextField)
3.3.1.1.1.1. Default: “Unnamed”
3.3.1.1.1.2. Validation: Cannot be Empty
3.3.1.2. Column 2: “Mechant” Header
3.3.1.2.1. Each Row: Name of the Merchant the transaction was paid to the order of. (TextField)
3.3.1.2.1.1. Default: “Unknown”
3.3.1.2.1.2. Validation: Cannot be Empty
3.3.1.3. Column 3: “Income?” Header
3.3.1.3.1. Each Row: Whether the Transaction is an income or not (Toggle)
3.3.1.3.1.1. Default: Unchecked (an expense)
3.3.1.4. Column 4: “Spent” Header
3.3.1.4.1. Each Row: Amount spent on the transaction in USD (TextField)
3.3.1.4.1.1. Default: $0.00
3.3.1.4.1.2. Validation: Must be a Non-negative Number
3.3.1.4.1.3. Format: USD Form (1.2345 -> $1.23)
3.3.1.5. Column 5: “Date” Header
3.3.1.5.1. Each Row: Date the Transaction Occurred (DatePicker)
3.3.1.5.1.1. Default: Current Date
3.3.1.5.1.2. Validation: Cannot be Empty

3.4. **Reconcile Button**

3.4.1. Labeled: “Reconcile This Month’s Data”
3.4.2. Action: Opens Reconcile Dialog

3.5. **Reconcile Dialog**

3.5.1. **Title**: “Reconcile Transactions for <Selected Month>”
3.5.2. **Content**: 1 Header, 1 Switch
3.5.2.1. Header Labeled: “How would you like to reconcile your transactions for <Selected Month>?”
3.5.2.2. Upload Switch
3.5.2.2.1. Left Label: “Manually”
3.5.2.2.2. Right Label: “Upload a CSV”
3.5.2.2.3. Default: Unchecked (Left)
3.5.3. **Actions**: 2 Buttons
3.5.3.1. Cancel Button

3.5.3.1.1. Labeled: “Cancel”
3.5.3.1.2. Action: Closes Reconcile Dialog
3.5.3.2. Continue Button
3.5.3.2.1. Labeled: “Continue”
3.5.3.2.2. If Upload Switch is Unchecked (Manual Mode):
3.5.3.2.2.1. Opens Manual Reconciliation Page
3.5.3.2.2.2. Closes Reconcile Dialog
3.5.3.2.3. If Upload Switch is Checked (Upload Mode):
3.5.3.2.3.1. Opens CSV Upload Dialog
3.5.3.2.3.2. Closes Reconcile Dialog

3.6. **CSV Upload Dialog**
3.6.1. **Title**: “Upload a Bank Statement CSV for <Selected Month>”
3.6.2. **Content**:
3.6.2.1. File Upload Button
3.6.2.1.1. Label: “Choose a File”
3.6.2.1.2. Actions:
3.6.2.1.2.1. opens browser dependent file picker requesting a csv file. (See This)
3.6.2.1.2.2. If CSV file chosen:
3.6.2.1.2.2.1. Update File Upload TextField to display the name of the file chosen
3.6.2.1.2.3. Otherwise:
3.6.2.1.2.3.1. Reset File Upload TextField to Default State.
3.6.2.2. File Upload TextField
3.6.2.2.1. Label: “Chosen File”
3.6.2.2.2. Default: Disabled & Empty
3.6.3. **Actions**: 2 Buttons
3.6.3.1. Cancel Button
3.6.3.1.1. Labeled: “Cancel”
3.6.3.1.2. Action: Closes CSV Upload Dialog
3.6.3.2. Continue Button
3.6.3.2.1. Labeled: “Continue”
3.6.3.2.2. If CSV file is chosen:
3.6.3.2.2.1. Uploads the CSV file to the server to process.
3.6.3.2.2.2. Receives a JSON table containing the data in the CSV file processed by the server in response.
3.6.3.2.2.3. Updates (TBD) Table with JSON Table contents
3.6.3.2.2.4. Opens Manual Reconciliation Page
3.6.3.2.2.5. Closes CSV Upload Dialog
3.6.3.2.3. Otherwise:

3.6.3.2.3.1. Opens Invalid Input Dialog
3.6.3.2.3.2. Note: CSV Upload Dialog remains open after Invalid Input Dialog Closes

3.7. **CSV Reconciliation Dialog**
3.7.1. **Title**: “Confirm CSV Format for <Selected Month>”
3.7.2. **Content**:
3.7.2.1.
3.7.3. **Actions**: 2 Buttons
3.7.3.1. Cancel Button
3.7.3.1.1. Labeled: “Cancel”
3.7.3.1.2. Action: Closes CSV Upload Dialog
3.7.3.2. Continue Button
3.7.3.2.1. Labeled: “Continue”
3.7.3.2.2.

3.8. **Reconcile** will be performed after the month the user chooses to reconcile is over and all transactions are entered.
3.8.1. Reconciling the budget finalizes the month that was reconciled and closes it for editing.
3.8.2. Reconciling means that all transactions are accurate per the user’s bank statement
3.8.3. The user will click the reconcile button in the bottom right of the monthly budget page.
3.8.3.1. The reconcile button will be labeled: “Reconcile”.
3.8.3.2. Once clicked the reconcile button will bring up a new screen titled “Reconcile”.
3.8.3.2.1. In the reconcile screen the user will have the option to select from prior months to reconcile via dropdown menu
3.8.3.2.2. The user will have the option to do a manual reconciliation or csv upload.
3.8.3.2.2.1. To select which option the user will have a clickable toggle switch next to each option.
3.8.3.2.2.2. The option that is selected will be filled in.
3.8.3.3. CSV reconciliation will ask the user a series of questions regarding their csv file.
3.8.3.3.1. The user will select if the csv file has a header.
3.8.3.3.2. If there is not a header the user will select which columns aligns with the merchant, date, amount and amount
3.8.3.3.3. If there is a header, YggFinance will read in the headers and attempt to match them with the appropriate fields.
3.8.3.3.3.1. The user will confirm if the headers YggFinance has selected are correct.

3.8.3.4. A manual reconciliation will require the user to “check off” each transaction entered.

3.8.3.4.1. YggFinance will display a list of all transactions entered for the selected month with an empty box next to each one

3.8.3.4.2. If that user can match the data in the transaction to their bank statement the box is checked.

3.8.3.4.3. Once all transactions are checked the reconcile button is clicked

3.8.3.4.3.1. Clicking the submit button brings up a pop up box with the message: “Are you sure you want to reconcile this month’s transactions?

3.8.3.4.3.2. The two options are buttons, one labeled “yes” the other box “cancel”.

3.8.3.4.3.3. Clicking cancel returns the user to the reconcile screen

3.8.3.4.3.4. Clicking yes saves the month that the reconciliation was performed.

4. Net Worth Page will guide the user and receive input to calculate their net worth.

4.1. Once the user’s net worth data has been initialized, it will be displayed.

4.1.1. Net worth will be displayed in U.S. dollar form at the top of the net worth tab

4.1.2. Net worth will be displayed as a pie chart

4.1.2.1. The pie chart will be divided by a percentage of each factor that goes into net worth from a the user’s total net worth

4.2. The fields required to calculate net worth will be located below the displayed data.

4.3. The user will be asked enter the following data via textbox in U.S. dollar form, each separate textbox will be labeled as follows:

4.3.1.1. Fields will be in two columns aligned next to each other

4.3.1.2. Fields 4.2.2 - 4.2.7 will be in the left column

4.3.1.3. Fields 4.2.8-4.2.13 will be in the right column

4.3.2. “Real estate value”

4.3.3. “Checkings accounts value”

4.3.4. “Savings account values”

4.3.5. “Retirement account values”

4.3.6. “Automobile values”

4.3.7. “Other Assets”

4.3.8. “Remaining mortgages balance”

4.3.9. “Consumer debt”

4.3.10. “Personal loans”

4.3.11. “Auto loans”

4.3.12. “Student loans”

4.3.13. “Other liabilities”

4.4. Once the data is entered the submit button will be clicked

4.4.1. The submit button will be located at the bottom right of the list of fields.

4.5. Clicking submit will update the displayed information

5. Other Components:

5.1. The **Invalid Input Message** will be displayed as a Dialog

5.1.1. The **Dialog** will contain a DialogTitle, Dialog Content, and DialogActions

5.1.1.1. The **DialogTitle** should display the text “Invalid Input”

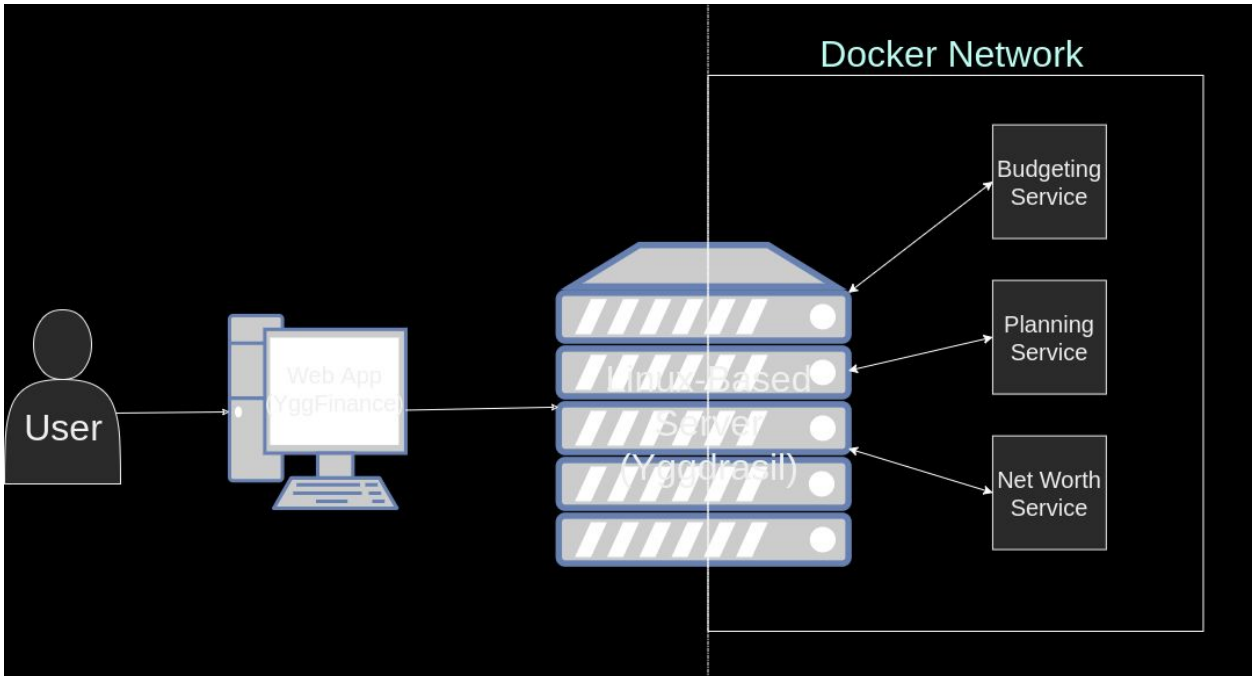
5.1.1.2. The **DialogContent** should display the reason why the input is invalid.

5.1.1.3. The **DialogActions** should contain a button displaying the word “Okay”

5.2.

6. Diagrams

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



7. Updated Timeline

Week	Description

02/01/2021 - Week 01	All: Working on the Requirements Document
02/08/2021 - Week 02	All: Familiarizing ourselves with the technologies we will be using Blake: ● Setup development environment ● Create Test projects with current technologies ● Research third party libraries Alan: ● [Monday] Revised Requirements Document ● [Tuesday] Revised Requirements Document ● [Wednesday] Revise Requirements Document ● [Friday] Revise Requirements Document ● [Saturday] Finish Revising Requirements Document Austin: ● Set up test environment for React.js ● Work on skeleton for top menu and multiple pages
02/15/2021 - Week 03	Blake: ● Troubleshoot any tech issues ● Further learning on chosen 3rd party libraries Alan: ● Spring Break 1 ● [Tuesday] Begin Working on Design Document ● [Wednesday] Continue Working on Design Document ● [Thursday] Continue Working on Design Document ● [Friday] Continue Working on Design Document ● [Saturday] Finish Working on Design Document Austin: ● Have working top menu and finished welcome page
02/22/2021 - Week 04	Blake: ● Work on initial design and create flowchart for monthly budget feature Alan: ● [Tuesday] Revise Design Document ● [Wednesday] Revise Design Document ● [Thursday] Finish Revising Design Document Austin: ● Work on Net Worth Page(User Inputs section)
03/01/2021 - Week 05	Blake: ● Work on Monthly budgeting code Alan: ● [1 day] Get Development Environment Set up ● [1 day] Create Test Project in Node.js

	• [1 day] Create Test Project in Java Spring Boot • [1 day] Set up Docker Compose file in repo • [1 day] Test hitting test API endpoints on Docker Network Austin: • Finish Net Worth Page(Pie chart visualization)
03/08/2021 - Week 06	Blake: • Work on Monthly budgeting code Alan: • [3 days] Set up the Yggdrasil Server • Spring Break 2 Austin: • Work on Savings Planner Page(Investment Calculator)
03/15/2021 - Week 07	Blake: • Work on Monthly budgeting code Alan: • [1 day] Begin work on Savings Planner Service • [2 days] Continue work on Savings Planner Service • [1 day] Finish work on Savings Planner Service • [1 day] Deploy Savings Planner Service Austin: • Savings planner page(Monthly contributions/time slider)
03/22/2021 - Week 08	Blake: • Work on Monthly budgeting code Alan: • [1 day] Begin work on Net Worth Service • [2 days] Continue work on Net Worth Service • [1 day] Finish work on Net Worth Service • [1 day] Deploy Net Worth Service Austin: • Monthly Budget Page(Planner selections)
03/29/2021 - Week 09	Blake: • Work on Monthly budgeting code Alan: • [Busy, No Work] • Good Friday Austin: • Monthly budget(transaction entry)
04/05/2021 - Week 10	Blake: • Integration and testing Alan: • [4-5 days] Perform Integration Testing on Deployed Services

	Austin: ● Monthly budget(reconciliation interface)
04/12/2021 - Week 11	Blake: Alan: ● [4 days] Perform System Testing on Final System Austin: ● UI cleanup
04/19/2021 - Week 12	Project Buffer Time
04/26/2021 - Week 13	Project Buffer Time
05/03/2021 - Week 14	Presentation Week!