Project: YggFinance

Group members: Blake Hudson, Alan Holman, and Austin Kerr

Table of Contents

Requirements

https://docs.google.com/document/d/183tJPdbQjW1SCaJoEjVqWTh_mHdAAX0H52b6hTXzoHs/edit

1. **Yggdrasil Server**
   - 1.1. The Server will **host** 5 Services locally on a Docker Network
     - **1.1.1. Planning Service**
     - **1.1.2. Budgeting Service**
     - **1.1.3. Reconciling Service**
     - **1.1.4. NetWorth Service**
     - **1.1.5. WebServer Service**
       - 1.1.5.1. Serves the **YggFinance WebApp**
   - 1.2. The Server will support concurrent usage by **multiple users**.
     - 1.2.1. The Yggdrasil Server will not maintain a distinction between users of the system.
     - 1.2.2. The Yggdrasil Server will not maintain any state information on any user sessions between requests. (Stateless Server)
     - 1.2.3. The Yggdrasil Server will not maintain any user information between requests. (No Databases)
   - 1.3. The Yggdrasil Server will direct incoming requests to the appropriate Service within the Local Docker Network.
   - 1.4. The Yggdrasil Server will direct outgoing responses from the Services in the Local Docker Network to the corresponding request source.
2. **YggFinance Web App**

2.1. The User will be able to **connect to** the YggFinance Web App through a URL in a modern Web Browser.
  2.1.1. The YggFinance Web App will require the Web Browser to have JavaScript Enabled in order to function properly.
  2.1.2. The YggFinance Web App will require access to the Web Browser's Local Storage in order to function properly.
    2.1.2.1. The YggFinance Web App may also function without local storage though will not be designed to accommodate such a use case.
2.2. Upon connecting, the YggFinance Web App will open to the **Welcome Page**.
  2.2.1. The **Welcome Page** will contain a short description of each other page in the **Top Menu** with their respective uses and tools.
2.3. 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**.
  2.3.1. 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.
2.4. The YggFinance Web App will store User Input and Application State Data on the user's machine in Local Storage (HTML5).
2.5. The YggFinance Web App will be designed to accommodate a 16:9 aspect ratio display at at least a 720p resolution.
  2.5.1. Unsupported resolutions and aspect ratios may also function though the YggFinance Web App will not be designed to accommodate them.
2.6. The YggFinance Web App will use the Material UI **style library** for displaying all of its elements to the User.

3. Savings Planner Page

3.1. **Local Storage Data:**
  3.1.1. initialInvestment: number
  3.1.2. avgRateOfReturn: number
  3.1.3. monthlyContributions: number
  3.1.4. planningMode: boolean
  3.1.5. timeFrame: number
  3.1.6. savingsGoal: number
3.2. **Page Data:**
  3.2.1. savingsPlannerData: object (copy of Req. 3.1)
3.3. **Initial Investment TextField**
  3.3.1. Labeled: "Initial investment (USD)"
  3.3.2. Default: $0.00
  3.3.3. Validation: Non-Negative numbers
  3.3.4. Format: USD
3.4. **Average Rate of Return TextField**
  3.4.1. Labeled: "Average rate of return (%)"

3.4.2. Default: 0%
3.4.3. Validation: Non-Negative numbers
3.4.4. Format: Percentage

3.5. **Monthly Contributions TextField**
3.5.1. Labeled: “Monthly contributions (USD)”
3.5.2. Default: $0.00
3.5.3. Validation: Non-Negative numbers
3.5.4. Format: USD

3.6. **Planning Mode Switch**
3.6.1. Default: Unchecked (Left)
3.6.2. 3 Labels
3.6.2.1. Top Label: “Planning Mode”
3.6.2.2. Left Label: “Time Frame”
3.6.2.3. Right Label: “Savings Goal”
3.6.3. On Change:
3.6.3.1. Switch Planning Mode **(See Design Documentation 1.5)**

3.7. **Time-Frame Slider**
3.7.1. Labeled: “Years to grow”
3.7.2. Default: 10
3.7.3. Range: [1, 100]

3.8. **Savings Goal TextField**
3.8.1. Labeled: “Savings Goal (USD)”
3.8.2. Default: $0.00
3.8.3. Validation: Non-Negative numbers
3.8.4. Format: USD

3.9. **Calculate Time Frame Button**
3.9.1. Labeled: “Calculate Time Frame”
3.9.2. On Press:
3.9.2.1. Calculate Time Frame **(See Design Documentation 1.6)**

3.10. **Calculate Savings Goal Button**
3.10.1. Labeled: “Calculate Savings Goal”
3.10.2. On Press:
3.10.2.1. Calculate Savings Goal **(See Design Documentation 1.7)**

3.11. **Results Table**
3.11.1. Labeled: “Results”
3.11.2. Layout: 2 Columns, 4 Rows
3.11.2.1. Row 1:
3.11.2.1.1. Column 1: “End Balance:”
3.11.2.1.2. Column 2:
3.11.2.1.2.1. Default: “$0.00”
3.11.2.1.2.2. Format: USD
3.11.2.2. Row 2:
3.11.2.2.1. Column 1: “Time Frame:”
3.11.2.2.2. Column 2:

3.11.2.2.2.1. Default: “0 years”
3.11.2.2.2.2. Format: “# years”
3.11.2.3. Row 3:
3.11.2.3.1. Column 1: “Starting Amount:”
3.11.2.3.2. Column 2:
3.11.2.3.2.1. Default: “$0.00”
3.11.2.3.2.2. Format: USD
3.11.2.4. Row 4:
3.11.2.4.1. Column 1: “Total Contributions:”
3.11.2.4.2. Column 2:
3.11.2.4.2.1. Default: “$0.00”
3.11.2.4.2.2. Format: USD
3.11.2.5. Row 5:
3.11.2.5.1. Column 1: “Total Interest:”
3.11.2.5.2. Column 2:
3.11.2.5.2.1. Default: “$0.00”
3.11.2.5.2.2. Format: USD

4. Monthly Budget Page

4.1. **Local Storage Data:**
4.1.1. The “monthly-budget-data” entry from Local Storage
4.1.1.1. budgetedMonths: array of BudgetMonth
4.1.2. BudgetMonth: object
4.1.2.1. month: number
4.1.2.2. year: number
4.1.2.3. categories: array of Category
4.1.2.4. bankTransactions: array of Transaction
4.1.3. Category: object
4.1.3.1. name: string
4.1.3.2. budget: number
4.1.3.3. transactions: array of Transaction
4.1.4. Transaction: object
4.1.4.1. id: number
4.1.4.2. merchant: string
4.1.4.3. amount: number
4.1.4.4. date: string
4.1.4.5. isReconciled: boolean
4.2. **Page Data:**
4.2.1. monthlyBudgetData: object (copy of Req. 4.1.1)
4.3. **Month Select**
4.3.1. Labeled: “Select a Month”
4.3.2. Values
4.3.2.1. Format: “<month> <year>” (ex. “January 2021”)
4.3.2.2. Default Selected: None
4.3.3. On Value Change:

4.3.3.1. Refresh Page **(See Design Documentation 2.2)**

4.4. **New Month Button**

4.4.1. Labeled: “Start a New Month”

4.4.2. On Press:

4.4.2.1. Add New Month **(See Design Documentation 2.4)**

4.5. **Save Changes Button**

4.5.1. Labeled: “Save Changes”

4.5.2. On Press:

4.5.2.1. Confirm Changes **(See Design Documentation 2.5)**

4.6. **Tracking Paper**

4.6.1. **Category Collapsible** EditTable

4.6.1.1. 4 Columns

4.6.1.1.1. Column 1: No Header

4.6.1.1.1.1. Each Row: Collapse Component

4.6.1.1.1.1.1. Default State: Visible & Collapsed

4.6.1.1.1.1.2. Inside Collapse Component: *Transaction EditTable* (4.8.2)

4.6.1.1.2. Column 2: “Category” Header

4.6.1.1.2.1. Each Row: Name of Category (TextField)

4.6.1.1.2.1.1. Default: “Unnamed”

4.6.1.1.2.1.2. Validation: Cannot be Empty

4.6.1.1.3. Column 3: “Spent” Header

4.6.1.1.3.1. Each Row: Amount Spent in USD (ReadOnly)

4.6.1.1.3.1.1. Value: Sum of all Entries in Column 3 of the Transaction EditTable (4.9.2).

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

4.6.1.1.4. Column 4: “Planned” Header

4.6.1.1.4.1. Each Row: Amount Planned in USD (TextField)

4.6.1.1.4.1.1. Default: $0.00

4.6.1.1.4.1.2. Validation: Must be a Non-negative Number

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

4.6.2. Transaction EditTable

4.6.2.1. 5 Data Columns

4.6.2.1.1. Column 1: “ID” Header

4.6.2.1.1.1. Each Row: ID of the Transaction (ReadOnly)

4.6.2.1.1.1.1. Value: the next incremental integer up from the largest current transaction id.

4.6.2.1.2. Column 2: “Mechant” Header

4.6.2.1.2.1. Each Row: Name of the Merchant the transaction was paid to the order of. (TextField)

4.6.2.1.2.1.1. Default: “Unknown”

4.6.2.1.2.1.2. Validation: Cannot be Empty

4.6.2.1.3. Column 3: “Spent” Header

4.6.2.1.3.1. Each Row: Amount spent on the transaction in USD (TextField)

4.6.2.1.3.1.1. Default: $0.00

4.6.2.1.3.1.2. Validation: Must be a valid number

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

4.6.2.1.4. Column 4: “Date” Header

4.6.2.1.4.1. Each Row: Date the Transaction Occurred (DatePicker)

4.6.2.1.4.1.1. Default: Current Date

4.6.2.1.4.1.2. Validation: Cannot be Empty

4.6.3. **Reconcile Button**

4.6.3.1. Label: “Reconcile This Month”

4.6.3.2. On Press:

4.6.3.2.1. Switch Visible Paper **(See Design Document 2.6)**

4.7. **Reconcile Paper**

4.7.1. Default: Hidden

4.7.2. **Self Tracked EditTable**

4.7.2.1. Same format as the *Transaction EditTable* (4.9.2) except:

4.7.2.1.1. All Existing Columns become ReadOnly

4.7.2.1.2. Additional Column

4.7.2.1.2.1. Column 5: “Reconciled?” Header

4.7.2.1.2.1.1. Each Row: Whether the Transaction has been reconciled or not (Toggle)

4.7.2.1.2.1.1.1. Default: Unchecked

4.7.3. **Bank Transactions EditTable**

4.7.3.1. Same format as the *Transaction EditTable* (4.9.2) except:

4.7.3.1.1. Additional Column

4.7.3.1.1.1. Column 5: “Reconciled?” Header

4.7.3.1.1.1.1. Each Row: Whether the Transaction has been reconciled or not (Toggle)

4.7.3.1.1.1.1.1. Default: Unchecked

4.7.4. **Upload CSV Button**

4.7.4.1. Labeled: “Upload a Bank Statement”

4.7.4.2. OnPress:

4.7.4.2.1. Opens the *Upload CSV Dialog* (4.10.5)

4.7.5. **Upload CSV Dialog**

4.7.5.1. **Title**: “Upload a Bank Statement CSV for <Selected Month>”

4.7.5.2. **Content**:

4.7.5.2.1. **File Upload Button**

4.7.5.2.1.1. Label: “Choose a File”

4.7.5.2.1.2. On Press:

4.7.5.2.1.2.1. Choose a File **(See Design Documentation 2.7)**

4.7.5.2.2. **File Upload TextField**

4.7.5.2.2.1. Label: “Chosen File”

4.7.5.2.2.2. Default: Disabled & Empty

4.7.5.2.3. **Headers Switch**

4.7.5.2.3.1. Label: “Does the CSV have a Header?”

4.7.5.2.3.2. Default: Unchecked

4.7.5.2.4. **Merchant Column TextField**

4.7.5.2.4.1. Label: “Column Containing Transaction Merchant”

4.7.5.2.4.2. Default: “A”

4.7.5.2.5. **Amount Column TextField**

4.7.5.2.5.1. Label: “Column Containing Transaction Amount”

4.7.5.2.5.2. Default: “B”

4.7.5.2.6. **Date Column TextField**

4.7.5.2.6.1. Label: “Column Containing Transaction Date”

4.7.5.2.6.2. Default: “C”

4.7.5.3. **Actions**: 2 Buttons

4.7.5.3.1. **Cancel Button**

4.7.5.3.1.1. Labeled: “Cancel”

4.7.5.3.1.2. On Press:

4.7.5.3.1.2.1. Close Dialog

4.7.5.3.2. **Confirm Button**

4.7.5.3.2.1. Labeled: “Confirm”

4.7.5.3.2.2. On Press:

4.7.5.3.2.2.1. Upload CSV **(See Design Documentation 2.8)**

4.8. **Reconciled Table**

4.8.1. Same format as the *Transaction EditTable* (4.9.2) except:

4.8.1.1. A Table not an EditTable.

4.8.1.2. All entries are ReadOnly (Labels).

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

5.1. **Local Storage Data:**
- 5.1.1. The “net-worth-data” entry from Local Storage
  - 5.1.1.1. assets: Assets object
  - 5.1.1.2. liabilities: Liabilities object
  - 5.1.1.3. netWorth: number
- 5.1.2. Assets: object
  - 5.1.2.1. realEstateValue: number
  - 5.1.2.2. checkingAccountsBalance: number
  - 5.1.2.3. savingsAccountsBalance: number
  - 5.1.2.4. retirementAccountsBalance: number
  - 5.1.2.5. automobilesValue: number
  - 5.1.2.6. other: number
- 5.1.3. Liabilities: object
  - 5.1.3.1. remainingMortgageBalance: number
  - 5.1.3.2. consumerDebt: number
  - 5.1.3.3. personalLoans: number
  - 5.1.3.4. autoLoans: number
  - 5.1.3.5. studentLoans: number
  - 5.1.3.6. other: number

5.2. **Page Data:**
- 5.2.1. netWorthData: object (copy of Req. 5.1.1)

5.3. **NetWorth Header**
- 5.3.1. Labeled: “Net Worth: <NetWorth>”
  - 5.3.1.1. <NetWorth>: the user’s net worth.
    - 5.3.1.1.1. Format: USD
    - 5.3.1.1.2. Default: $0.00

5.4. **Assets Header:** “Assets”
- 5.4.1. **Real Estate Value TextField**
  - 5.4.1.1. Labeled: “Real Estate Value”
  - 5.4.1.2. Format: USD
  - 5.4.1.3. Validation: Non-negative Numbers
- 5.4.2. **Checkings Account Balance TextField**
  - 5.4.2.1. Labeled: “Checkings Account Balance”
  - 5.4.2.2. Format: USD
  - 5.4.2.3. Validation: Non-negative Numbers
- 5.4.3. **Savings Account Balance TextField**
  - 5.4.3.1. Labeled: “Savings Account Balance”
  - 5.4.3.2. Format: USD
  - 5.4.3.3. Validation: Non-negative Numbers
- 5.4.4. **Retirement Account Balance TextField**
  - 5.4.4.1. Labeled: “Retirement Account Balance”
  - 5.4.4.2. Format: USD
  - 5.4.4.3. Validation: Non-negative Numbers
- 5.4.5. **Automobile Value TextField**

5.4.5.1. Labeled: “Automobile Value”
5.4.5.2. Format: USD
5.4.5.3. Validation: Non-negative Numbers
5.4.6. **Other Assets TextField**
5.4.6.1. Labeled: “Other Assets”
5.4.6.2. Format: USD
5.4.6.3. Validation: Non-negative Numbers
5.5. **Liabilities Header:** “Liabilities”
5.5.1. **Remaining Mortgage Balance TextField**
5.5.1.1. Labeled: “Remaining Mortgage Balance”
5.5.1.2. Format: USD
5.5.1.3. Validation: Non-negative Numbers
5.5.2. **Consumer Debt TextField**
5.5.2.1. Labeled: “Consumer Debt”
5.5.2.2. Format: USD
5.5.2.3. Validation: Non-negative Numbers
5.5.3. **Personal Loans TextField**
5.5.3.1. Labeled: “Personal Loans”
5.5.3.2. Format: USD
5.5.3.3. Validation: Non-negative Numbers
5.5.4. **Auto Loans TextField**
5.5.4.1. Labeled: “Auto Loans”
5.5.4.2. Format: USD
5.5.4.3. Validation: Non-negative Numbers
5.5.5. **Student Loans TextField**
5.5.5.1. Labeled: “Student Loans”
5.5.5.2. Format: USD
5.5.5.3. Validation: Non-negative Numbers
5.5.6. **Other Liabilities TextField**
5.5.6.1. Labeled: “Other Liabilities”
5.5.6.2. Format: USD
5.5.6.3. Validation: Non-negative Numbers
5.6. **Calculate Button**
5.6.1. Labeled: “Calculate NetWorth”
5.6.2. On Press:
5.6.2.1. Calculate Net Worth **(See Design Documentation 3.5)**

6. Other Components:

6.1. The **Invalid Input Dialog** will be displayed as a Dialog
6.1.1. The **Dialog** will contain a DialogTitle, Dialog Content, and DialogActions
6.1.1.1. The **DialogTitle** should display the text “Invalid Input”
6.1.1.2. The **DialogContent** should display the reason why the input is invalid.

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

6.2. The **Error Code Dialog** will be displayed as a Dialog

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

6.2.1.1. The **DialogTitle** should display the text “Error”

6.2.1.2. The **DialogContent** should display the error or error code

6.2.1.3. The **DialogActions** should contain a button displaying the word “Close”

6.3. The **Confirmation Dialog** will be displayed as a Dialog

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

6.3.1.1. The **DialogTitle** should display the text “Are you sure?”

6.3.1.2. The **DialogContent** should display “Are you sure you would like to <Action to be confirmed>?”

6.3.1.3. The **DialogActions** should contain a button displaying the word “No” and a button displaying the word “Yes”

Design Description

A 5-6 page written description of the system referring to the images and diagrams in the appendices as needed.

1. **Savings Planner Page (See Appendix 1.1 for UML Diagram; See Appendix 4.1 for UI Wireframe Diagram)**
    - 1.1. **On Page Load:**
        - 1.1.1. Retrieves the “savings-planner-data” entry from Local Storage **(See Appendix 3.1.1)** and stores it into a variable called savingsPlannerData **(See Requirements 3.2.1)**.
            - 1.1.1.1. Refresh Page **(1.2)**
    - 1.2. **Refresh Page:**
        - 1.2.1. Populates the Initial Investment TextField **(See Requirements 3.3)** with the “initialInvestment” field in the savingsPlannerData variable **(See Requirements 3.2.1)**.
        - 1.2.2. Populates the Average Rate of Return TextField **(See Requirements 3.4)** with the “avgRateOfReturn” field in the savingsPlannerData variable **(See Requirements 3.2.1)**.
        - 1.2.3. Populates the Monthly Contributions TextField **(See Requirements 3.5)** with the “monthlyContributions” field in the savingsPlannerData variable **(See Requirements 3.2.1)**.
        - 1.2.4. Sets the Planning Mode Switch **(See Requirements 3.6)** to the value of the “planningMode” field in the savingsPlannerData variable **(See Requirements 3.2.1)**.
        - 1.2.5. Sets the Time Frame Slider **(See Requirements 3.7)** to the value of the “timeFrame” field in the savingsPlannerData variable **(See Requirements 3.2.1)**.
        - 1.2.6. Populates the Savings Goal TextField **(See Requirements 3.8)** with the “savingsGoal” field in the savingsPlannerData variable **(See Requirements 3.2.1)**.
    - 1.3. **Save Changes:**
        - 1.3.1. Sets the “initialInvestment” field in the savingsPlannerData variable **(See Requirements 3.2.1)** to the value of the Initial Investment TextField **(See Requirements 3.3)**.
        - 1.3.2. Sets the “avgRateOfReturn” field in the savingsPlannerData variable **(See Requirements 3.2.1)** to the value of the Average Rate of Return TextField **(See Requirements 3.4)**.
        - 1.3.3. Sets the “monthlyContributions” field in the savingsPlannerData variable **(See Requirements 3.2.1)** to the value of the Monthly Contributions TextField **(See Requirements 3.5)**.
        - 1.3.4. Sets the “planningMode” field in the savingsPlannerData variable **(See Requirements 3.2.1)** to the value of the Planning Mode Switch **(See Requirements 3.6)**.

1.3.5. Sets the “timeFrame” field in the savingsPlannerData variable **(See Requirements 3.2.1)** to the value of the Time Frame Slider **(See Requirements 3.7)**.

1.3.6. Sets the “savingsGoal” field in the savingsPlannerData variable **(See Requirements 3.2.1)** to the value of the Savings Goal TextField **(See Requirements 3.8)**.

1.4. **Save To Local Storage:**

1.4.1. Overwrites the “savings-planner-data” entry in Local Storage **(See Appendix 3.1.1)** with the contents of the savingsPlannerData variable **(See Requirements 3.2.1)**.

1.5. **Switch Planning Mode:**

1.5.1. When Planning Mode Switch **(See Requirements 3.6)** is Unchecked (Left):

1.5.1.1. Hides the Time-Frame Slider **(See Requirements 3.7)**

1.5.1.2. Hides the Calculate Savings Goal Button **(See Requirements 3.10)**

1.5.1.3. Shows the Savings Goal TextField **(See Requirements 3.8)**

1.5.1.4. Shows the Calculate Time Frame Button **(See Requirements 3.9)**

1.5.2. When Planning Mode Switch **(See Requirements 3.6)** is Checked (Right):

1.5.2.1. Hides the Savings Goal TextField **(See Requirements 3.8)**

1.5.2.2. Hides the Calculate Time Frame Button **(See Requirements 3.9)**

1.5.2.3. Shows the Time-Frame Slider **(See Requirements 3.7)**

1.5.2.4. Shows the Calculate Savings Goal Button **(See Requirements 3.10)**

1.6. **Calculate Time Frame**

1.6.1. If the following Text Fields contain valid data:

1.6.1.1. Initial Investment TextField **(See Requirements 3.3)**

1.6.1.2. Average Rate of Return TextField **(See Requirements 3.4)**

1.6.1.3. Monthly Contributions TextField **(See Requirements 3.5)**

1.6.1.4. Savings Goal Text Field **(See Requirements 3.8)**

1.6.2. Then:

1.6.2.1. Save Changes **(1.3)**

1.6.2.2. Send a calc-plan-time-frame request **(See Appendix 2.1.2)** to the *Planning Service* **(4)** with the contents of the savingsPlannerData variable **(See Requirements 3.2.1)** as the calc-plan-time-frame-request-msg **(See Appendix 2.1.2.2)**.

1.6.2.3. If an error code is received, display the Error Code Dialog **(See Requirements 6.2)**

1.6.2.4. Otherwise, populate the *Results Table* **(See Requirements 3.11)** with the contents of calc-plan-time-frame-response-msg **(See Appendix 2.1.2.3)**
1.6.2.5. Save Changes **(1.3)**
1.6.2.6. Save To Local Storage **(1.4)**
1.6.2.7. Refresh Page **(1.2)**
1.6.3. Otherwise:
1.6.3.1. Display an *Invalid Input Dialog* **(See Requirements 6.1)**

1.7. **Calculate Savings Goal**
1.7.1. If the following Text Fields contain valid data:
1.7.1.1. Initial Investment TextField **(See Requirements 3.3)**
1.7.1.2. Average Rate of Return TextField **(See Requirements 3.4)**
1.7.1.3. Monthly Contributions TextField **(See Requirements 3.5)**
1.7.2. Then:
1.7.2.1. Save Changes **(1.3)**
1.7.2.2. Send a calc-plan-end-balance request **(See Appendix 2.1.1)** to the *Planning Service* **(4)** with the contents of the savingsPlannerData variable **(See Requirements 3.2.1)** as the calc-plan-end-balance-request-msg **(See Appendix 2.1.1.2)**.
1.7.2.3. If an error code is received, display the Error Code Dialog **(See Requirements 6.2)**
1.7.2.4. Otherwise, populate the *Results Table* **(See Requirements 3.11)** with the contents of calc-plan-end-balance-response-msg **(See Appendix 2.1.1.3)**
1.7.2.5. Save Changes **(1.3)**
1.7.2.6. Save To Local Storage **(1.4)**
1.7.2.7. Refresh Page **(1.2)**
1.7.3. Otherwise:
1.7.3.1. Display an *Invalid Input Dialog* **(See Requirements 6.1)**

2. Monthly Budget Page (See Appendix 1.2 for UML Diagram; See Appendix 4.2 for UI Wireframe Diagram)

2.1. **On Page Load:**
2.1.1. Retrieves the “monthly-budget-data” entry from Local Storage **(See Appendix 3.2.1)** and stores it into a variable called monthlyBudgetData **(See Requirements. 4.2.1)**
2.1.1.1. Populates the Month Select **(See Requirements. 4.3)** with the data for the “month” and “year” fields for each BudgetMonth in the “budgetedMonths” field.

2.2. **Refresh Page:**

2.2.1. Retrieves the selected BudgetMonth **(See Requirements. 4.3)** from the “budgetedMonths” field in the monthlyBudgetData variable **(See Requirements. 4.2.1)**

2.2.2. Retrieves the “categories” field from the selected BudgetMonth.

2.2.2.1. For Each Category object in the “categories” field:

2.2.2.1.1. Add it to the *Categories Collapsible EditTable* (4.9.1)

2.2.2.1.2. Retrieve the “transactions” field from the Category object.

2.2.2.1.3. For each Transaction object in the “transactions” field:

2.2.2.1.3.1. Add it to the Corresponding *Transaction EditTable* (4.9.2)

2.2.2.1.3.2. Add it to the *Self Tracked EditTable* (4.10.2) if the “isReconciled” field is false.

2.2.2.1.3.3. Add it to the *Reconciled Table* (4.11) if the “isReconciled” field is true.

2.2.2.1.4. Retrieves the “bankTransactions” field from the Category object.

2.2.2.1.4.1. For each Transaction object in the “bankTransactions” field:

2.2.2.1.4.1.1. Add it to the *Bank Transactions EditTable* (4.10.3).

2.3. **Save Changes:**

2.3.1. Retrieves the selected (4.6) BudgetMonth from the “budgetedMonths” field in the monthlyBudgetData object (4.2.1). (match on month & year pair)

2.3.1.1. Overwrites(as in (re)constructs) the “categories” field in the retrieved BudgetMonth.

2.3.1.1.1. For Each row in the *Categories Collapsible EditTable* (4.9.1):

2.3.1.1.1.1. Create a Category Object out of the data in the row.

2.3.1.1.1.1.1. For each row in the Collapse Component’s *Transaction EditTable* (4.9.2):

2.3.1.1.1.1.1.1. Create a Transaction object out of the data in the row.

2.3.1.1.1.1.1.2. Add the Transaction object to the “transactions” field in the Category object.

2.3.1.1.1.2. Add the Category Object to the “categories” field in the BudgetMonth.

2.3.1.1.2. For each row in the the *Self Tracked EditTable* (4.10.2):

2.3.1.1.2.1. Match the transaction data in the row with a Transaction object in the “transactions” field of some Category object in the “categories” field. (Match on ID)

2.3.1.1.2.2. Set the “isReconciled” field to the status of Column 5.

2.3.1.1.3. For each row in the *Reconciled Table* (4.11):

2.3.1.1.3.1. Match the transaction data in the row with a Transaction object in the “transactions” field of some Category object in the “categories” field. (Match on ID)

2.3.1.1.3.2. Set the “isReconciled” field to “true”

2.3.1.2. Overwrites(as in (re)constructs) the “bankTransactions” field in the retrieved BudgetMonth.

2.3.1.2.1. For Each row in the *Bank Transactions EditTable* (4.10.3):

2.3.1.2.1.1. Create a Transaction object out of the data in the row.

2.3.1.2.1.2. Add the Transaction object to the “bankTransactions” field.

2.3.2. Overwrites the selected (4.6) BudgetMonth from the “budgetedMonths” field in the monthlyBudgetData object (4.2.1) with the updated BudgetMonth.

2.3.3. Overwrites the “monthly-budget-data” entry in Local Storage (4.1.1) with the contents of the monthlyBudgetData variable (4.2.1).

2.3.4. If Reconcile Paper Visible:

2.3.4.1. Switch Visible Paper **(2.6)**

2.4. **Add New Month**

2.4.1. Adds another month to the Month Select.

2.4.2. Creates and adds a new BudgetMonth Object to the “budgetedMonths” field of the monthlyBudgetData object.

2.5. **Confirm Changes**

2.5.1. Open Confirmation Dialog **(See Requirements 6.3)**:

2.5.1.1. Action to be confirmed: “save your changes”

2.5.1.2. On Yes:

2.5.1.2.1. Save Changes **(2.3)**

2.5.1.3. On No:

2.5.1.3.1. Close Dialog

2.6. **Switch Visible Paper**

2.6.1. Refresh Page **(2.2)**

2.6.2. If Tracking Paper Visible:

2.6.2.1. Hide Tracking Paper **(See Requirements 4.6)**
2.6.2.2. Unhide Reconcile Paper **(See Requirements 4.7)**
2.6.3. Otherwise, If Reconcile Paper Visible:
2.6.3.1. Unhide Tracking Paper **(See Requirements 4.6)**
2.6.3.2. Hide Reconcile Paper **(See Requirements 4.7)**

2.7. **Choose a File**
2.7.1. opens browser dependent file picker requesting a csv file. (See This)
2.7.2. If CSV file chosen:
2.7.2.1. Update *File Upload TextField* (4.10.5.2.2) to display the name of the file chosen
2.7.3. Otherwise:
2.7.3.1. Reset *File Upload TextField* (4.10.5.2.2) to Default State.

2.8. **Upload CSV**
2.8.1. If no CSV file is provided, display an *Invalid Input Dialog* (See Req. 6.1)
2.8.2. Otherwise, send a request to the Monthly *Budget Service* (See Req. XXX) containing the CSV file and the column mappings.
2.8.3. If an error code is received, display the *Error Code Dialog* (See Req. 6.2)
2.8.4. Otherwise, parse the response and add the transactions to the *Bank Transactions EditTable* (Req. 4.10.3)
2.8.5. Close Dialog

3. Net Worth Page

3.1. **On Page Load:**
3.1.1. Retrieves the “net-worth-data” entry from Local Storage and stores it into a variable called netWorthData.
3.1.1.1. Refresh Page (5.4)

3.2. **Refresh Page:**
3.2.1. Retrieves the Assets object from the “assets” field in the netWorthData variable.
3.2.1.1. Populates the Real Estate Value TextField (5.7.1) with the “realEstateValue” field
3.2.1.2. Populates the Checkings Account Balance TextField (5.7.2) with the “checkingAccountsBalance” field
3.2.1.3. Populates the Savings Account Balance TextField (5.7.3) with the “savingsAccountsBalance” field
3.2.1.4. Populates the Retirement Account Balance TextField (5.7.4) with the “retirementAccountsBalance” field
3.2.1.5. Populates the Automobile Value TextField (5.7.5) with the “automobilesValue” field
3.2.1.6. Populates the Other Assets TextField (5.7.6) with the “other” field

3.2.2. Retrieves the Liabilities object from the “liabilities” field in the netWorthData variable.
3.2.2.1. Populates the Remaining Mortgage Balance TextField (5.8.1) with the “remainingMortgageBalance” field
3.2.2.2. Populates the Consumer Debt TextField (5.8.2) with the “consumerDebt” field
3.2.2.3. Populates the Personal Loans TextField (5.8.3) with the “personalLoans” field
3.2.2.4. Populates the Auto Loans TextField (5.8.4) with the “autoLoans” field
3.2.2.5. Populates the Student Loans TextField (5.8.5) with the “studentLoans” field
3.2.2.6. Populates the Other Liabilities TextField (5.8.6) with the “other” field
3.2.3. Retrieves the “netWorth” field from the netWorthData variable.
3.2.3.1. Populates the <NetWorth> part of the NetWorth Header (5.7) with the “netWorth” field.

3.3. **Save Changes:**
3.3.1. Retrieves the Assets object from the “assets” field in the netWorthData variable.
3.3.1.1. Populates the “realEstateValue” field with the Real Estate Value TextField (5.7.1)
3.3.1.2. Populates the “checkingAccountsBalance” field with the Checkings Account Balance TextField (5.7.2)
3.3.1.3. Populates the “savingsAccountsBalance” field with the Savings Account Balance TextField (5.7.3)
3.3.1.4. Populates the “retirementAccountsBalance” field with the Retirement Account Balance TextField (5.7.4)
3.3.1.5. Populates the “automobilesValue” field with the Automobile Value TextField (5.7.5)
3.3.1.6. Populates the “other” field with the Other Assets TextField (5.7.6)
3.3.2. Overwrites the “assets” field in the netWorthData variable with the updated Assets object.
3.3.3. Retrieves the Liabilities object from the “liabilities” field in the netWorthData variable.
3.3.3.1. Populates the “remainingMortgageBalance” field with the Remaining Mortgage Balance TextField (5.8.1)
3.3.3.2. Populates the “consumerDebt” field with the Consumer Debt TextField (5.8.2)
3.3.3.3. Populates the “personalLoans” field with the Personal Loans TextField (5.8.3)
3.3.3.4. Populates the “autoLoans” field with the Auto Loans TextField (5.8.4)

3.3.3.5. Populates the “studentLoans” field with the Student Loans TextField (5.8.5)
3.3.3.6. Populates the “other” field with the Other Liabilities TextField (5.8.6)
3.3.4. Overwrites the “liabilities” field in the netWorthData variable with the updated Liabilities object.
3.3.5. Parses the <NetWorth> part of the NetWorth Header (5.7).
3.3.6. Overwrites the “netWorth” field in the netWorthData variable with the parsed <NetWorth> number.

3.4. **Save To Local Storage:**
3.4.1. Overwrites the “net-worth-data” entry in Local Storage (5.1.1) with the contents of the netWorthData variable (5.2.1).

3.5. **Calculate Net Worth**
3.5.1. If any field is invalid, display an *Invalid Input Dialog* (6.1)
3.5.2. Otherwise:
3.5.2.1. Save Changes (5.5)
3.5.2.2. Send a calc-net-worth request to the *NetWorth Service* (XXX) with the contents of the netWorthData variable as the calc-net-worth-request-msg.
3.5.2.3. If an error code is received, display the *Error Code Dialog* (6.2)
3.5.2.4. Otherwise, populate the <NetWorth> part of the *NetWorth Header* (5.6) with the calc-net-worth-response-msg’s “netWorth” field
3.5.2.5. Save Changes (5.5)
3.5.2.6. Save To Local Storage (5.6)
3.5.2.7. Refresh Page (5.4)
3.5.2.8. Close Dialog

4. Planning Service: (see Design 1.6 & 1.7; Module diagram 1)
4.1. When a calc-plan-time-frame request is received the calc-plan-time-frame-request-msg is passed as INPUT.
4.2. INPUT is in the format listed Appendix 2.1.b.ii
4.3. calcTimeFrame() function performs the calculation to determine the time frame required to acquire the desired investment value.
4.4. The formula calcTimeFrame() is using is t =ln(F/p)/(ln(1+r/n)n the formula for compound interest
4.4.1. t = timeFrame
4.4.2. F = endBalance
4.4.3. p = startingAmount
4.4.4. r = totalInterest
4.4.5. n = 1
4.5. The OUTPUT as shown in appendix 2.1.b.iii. updated contents of calc-plan-time-frame-response-msg

4.6. When a calc-plan-end-balance request is received the calc-plan-end-balance-request-msg is passed as INPUT.
4.7. INPUT is in the format listed Appendix 2.1.a.ii
4.8. calcEndBalance() function performs the calculation to determine the time ending amount of an investment after some time.
4.9. The formula calcEndBalance() is using is F = P(1+r/n)^(nt) the formula for compound interest
4.9.1. t = timeFrame
4.9.2. F = endBalance
4.9.3. p = startingAmount
4.9.4. r = totalInterest
4.9.5. n = 1
4.10. The OUTPUT as shown in appendix 2.1.a.iii. updated contents of calc-plan-end-balance-response-msg

5. Monthly Budget Service: (see Design 2.8)

5.1. When a csv-process request is received, csv-process-request-msg is passed as INPUT to parseRequestToCSV().
5.2. INPUT is in the format described by csv-process-request-msg **(See Appendix 2 2.1.2)**
5.3. parseRequestToCSV() is a function that takes a FormData object with the form described by csv-process-request-msg **(See Appendix 2 2.1.2)**
5.3.1. The appended file in the FileData object will contain the user's bank transactions for a given month in a csv file.
5.3.2. The three strings will contain the letters of the columns of the rows required for the reconcile feature to parse.
5.3.2.1. merchantColumn: string of column identifier containing the column where merchants are listed
5.3.2.2. amountColumn: string of column identifier containing the column where amounts of transactions are listed
5.3.2.3. dateColumn: string of column identifier containing the column where dates of transactions are listed
5.3.3. The hasHeaders boolean in the FileData object tells parseCSV() if there is a header row so that it can be skipped during parsing if necessary.
5.3.4. parseRequestToCSV() will then read in the specified columns and create a simplified csv
5.3.5. The newly created csv will then be passed to parseCSV()
5.3.5.1. parseCSV() will create an array of Transactions
5.3.5.2. A Transaction is specified in appendix 2.a.iii
5.4. The OUTPUT is formed by parseCSV() in the format described by csv-process-response-msg **(See Appendix 2 2.1.3)**.

6. Net Worth Service: (see Design 3.5)

6.1. When a calc-net-worth request is received, the calc-net-worth-msg is passed as INPUT to calcNetWorth().

6.2. INPUT is in the format listed Appendix 3.a.ii

6.3. calcNetWorth() is a function that calculates the user's net worth by adding all assets and subtracting all Liabilites

6.3.1. The first thing calcNetWorth does is sum the total of all Assets

6.3.2. Then calcNetworth sums the total of all Liabiliites

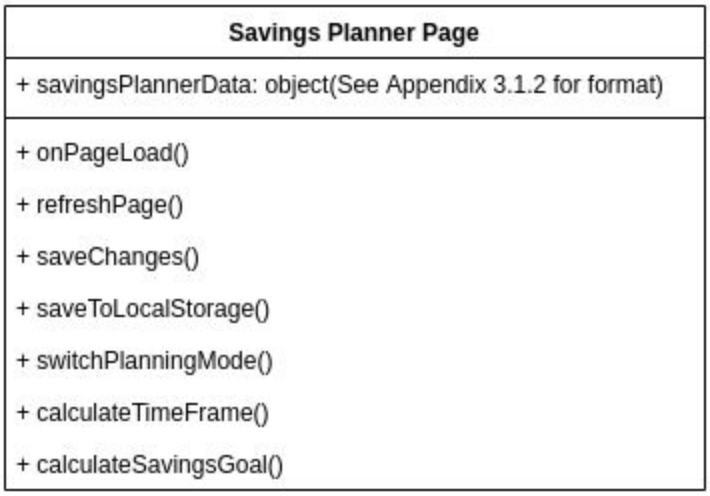
6.3.3. netWorth is the calculated by subtracting the sum of the Assets and the sum of Liabilities

6.3.4. netWorth is then updated on the calc-net-worth-response-msg and output by calcNetWorth().

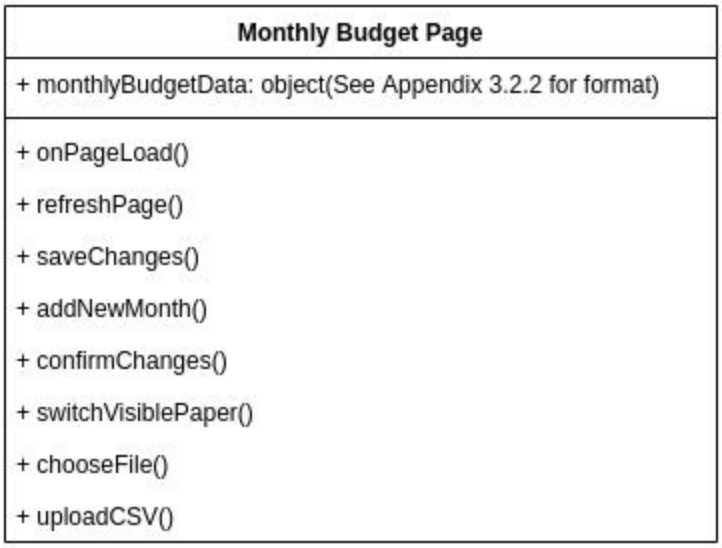
6.4. The OUTPUT as shown in Appendix 3.a.ii. updated contents of calc-net-worth-response-msg

Appendix 1 - Module/Service Diagram

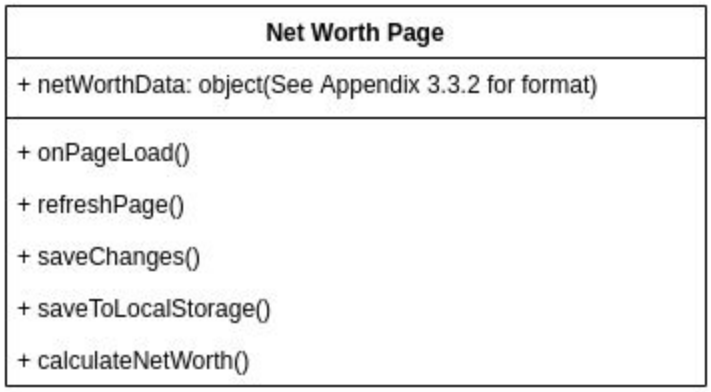
1. Savings Planner Page



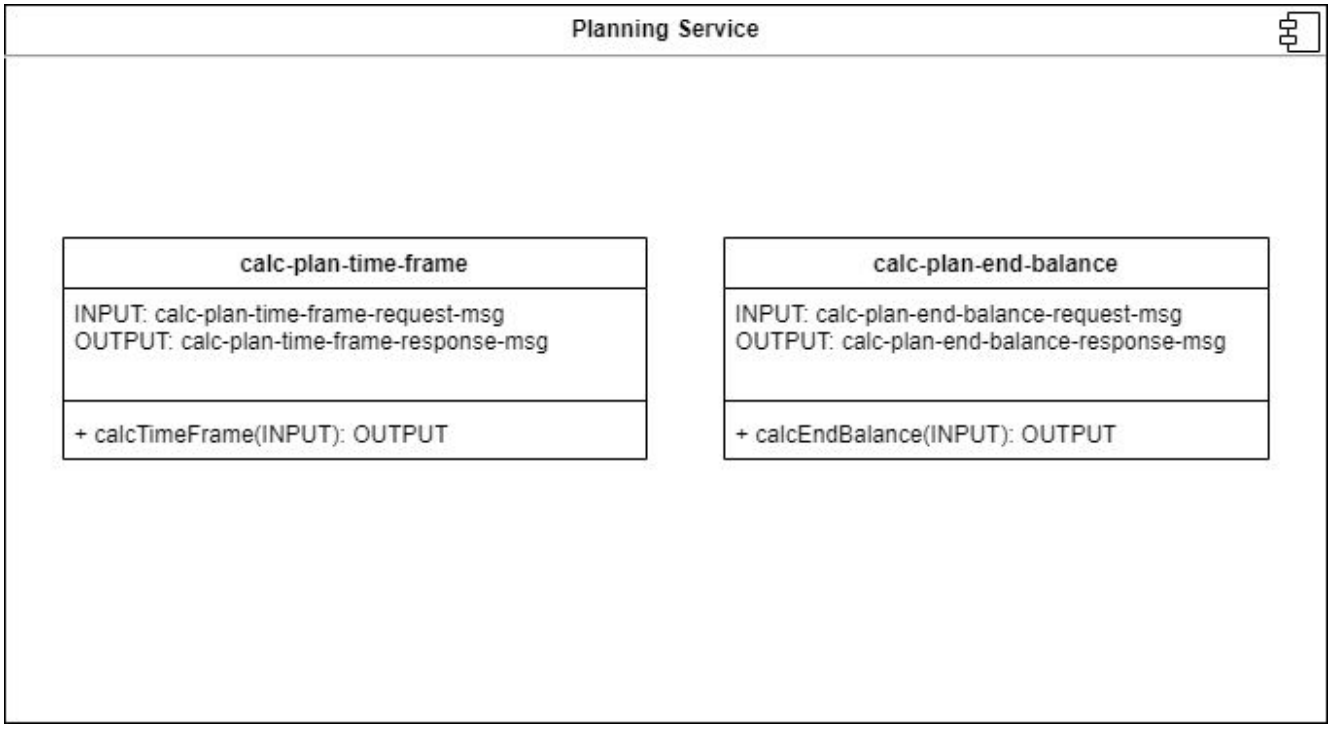
2. Monthly Budget Page



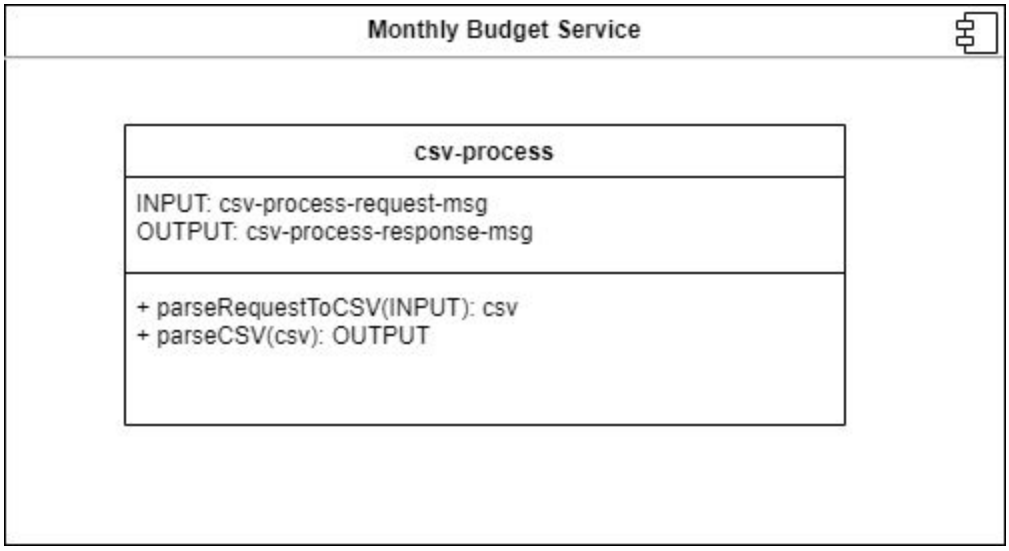
3. Net Worth Page



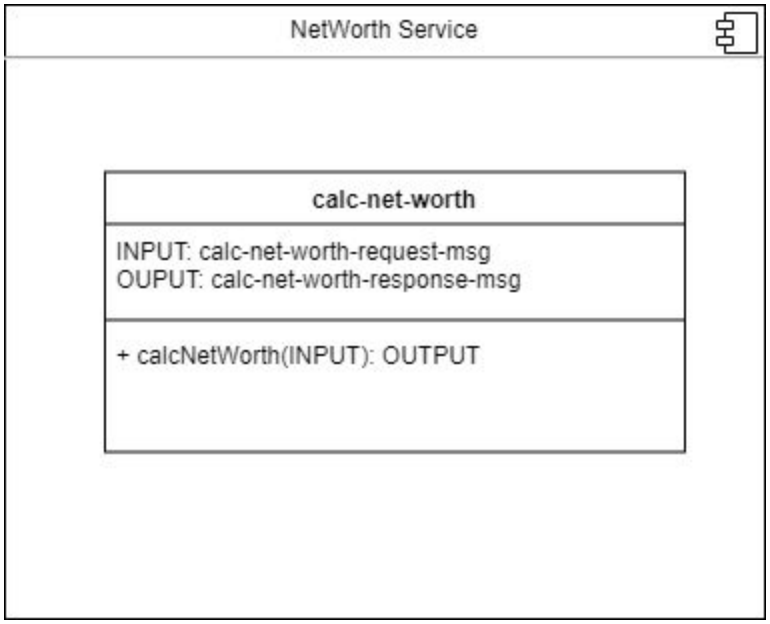
4. **Planning Service**



5. **Monthly Budget Service**



6. **Net Worth Service**



Appendix 2 – Message Documentation

1. Planning Service API
    1.1. calc-plan-end-balance
        1.1.1. HTTP POST "/YggFinance/savings-planner-page/calc-plan-end-balance"
        1.1.2. calc-plan-end-balance-request-msg
            1.1.2.1. Request Content-Type: "application/json"
            1.1.2.2. Request JSON Contract:
                1.1.2.2.1. *"savings-planner-data"* entry (See Appendix 3.1)
        1.1.3. calc-plan-end-balance-response-msg
            1.1.3.1. Request Content-Type: "application/json"
            1.1.3.2. Request JSON Contract:

```
{
    endBalance: number,
    timeFrame: number,
    startingAmount: number,
    totalContributions: number,
    totalInterest: number
}
```

    1.2. calc-plan-time-frame
        1.2.1. HTTP POST "/YggFinance/savings-planner-page/calc-plan-time-frame"
        1.2.2. calc-plan-time-frame-request-msg
            1.2.2.1. Request Content-Type: "application/json"
            1.2.2.2. Request JSON Contract:
                1.2.2.2.1. *"savings-planner-data"* entry (See Appendix 3.1)
        1.2.3. calc-plan-time-frame-response-msg
            1.2.3.1. Response Content-Type: "application/json"
            1.2.3.2. Response JSON Contract:

```
{
    endBalance: number,
    timeFrame: number,
    startingAmount: number,
    totalContributions: number,
    totalInterest: number
}
```

2. Monthly Budget Service API
    2.1. csv-process
        2.1.1. HTTP POST "/YggFinance/monthly-budget-page/csv-process"
        2.1.2. csv-process-request-msg
            2.1.2.1. Request Content-Type: "multipart/form-data"
            2.1.2.2. Request Form Contract:

```
// set fields
    hasHeaders: boolean
    merchantColumn: string
```

```
        amountColumn: string
        dateColumn: string
    // appended files
        file: file
```

2.1.3. csv-process-response-msg

2.1.3.1. Response Content-Type: “application/json”

2.1.3.2. Response JSON Contract:

```
{
    transactions: array[Transaction]
}

const Transaction = {
    merchant: string,
    amount: number,
    date: string, // (date.toJSON)
    isReconciled: boolean
}
```

3. NetWorth Service API

3.1. calc-net-worth

3.1.1. HTTP POST “/YggFinance/net-worth-page/calc-net-worth”

3.1.2. calc-net-worth-request-msg

3.1.2.1. Request Content-Type: “application/json”

3.1.2.2. Request JSON Contract:

3.1.2.2.1. *“net-worth-data”* entry (See Appendix 3.3)

3.1.3. calc-net-worth-response-msg

3.1.3.1. Response Content-Type: “application/json”

3.1.3.2. Response JSON Contract:

```
{
    netWorth: number
}
```

4. WebServer Service API

4.1. get-web-app

4.1.1. HTTP GET “/YggFinance”

4.1.2. get-web-app-request-message

4.1.2.1. N/A

4.1.3. get-web-app-response-message

4.1.3.1. Response Content-Type: “text/html”

4.1.3.2. Response Content: The YggFinance WebApp

Appendix 3 – Storage Documentation

React Local Storage for data in:

1. **Savings Planner Page**
   - 1.1. **Name: “savings-planner-data”**
   - 1.2. **Content: {Contract}**

```
{
   initialInvestment: number,
   avgRateOfReturn: number,
   monthlyContributions: number,
   planningMode: boolean,
   timeFrame: number,
   savingsGoal: number
}
```

2. **Monthly Budget Page**
   - 2.1. **Name: “monthly-budget-data”**
   - 2.2. **Content {Contract}:**

```
{
   budgetedMonths : array[BudgetMonth]
}

const BudgetMonth = {
   month: number,
   year: number,
   categories: array[Category],
   bankTransactions: array[Transaction]
}

const Category = {
   name: string,
   budget: number,
   transactions: array[Transaction]
}

const Transaction = {
   merchant: string,
   amount: number,
   date: string, // (date.toJSON)
   isReconciled: boolean
```

```
}
```

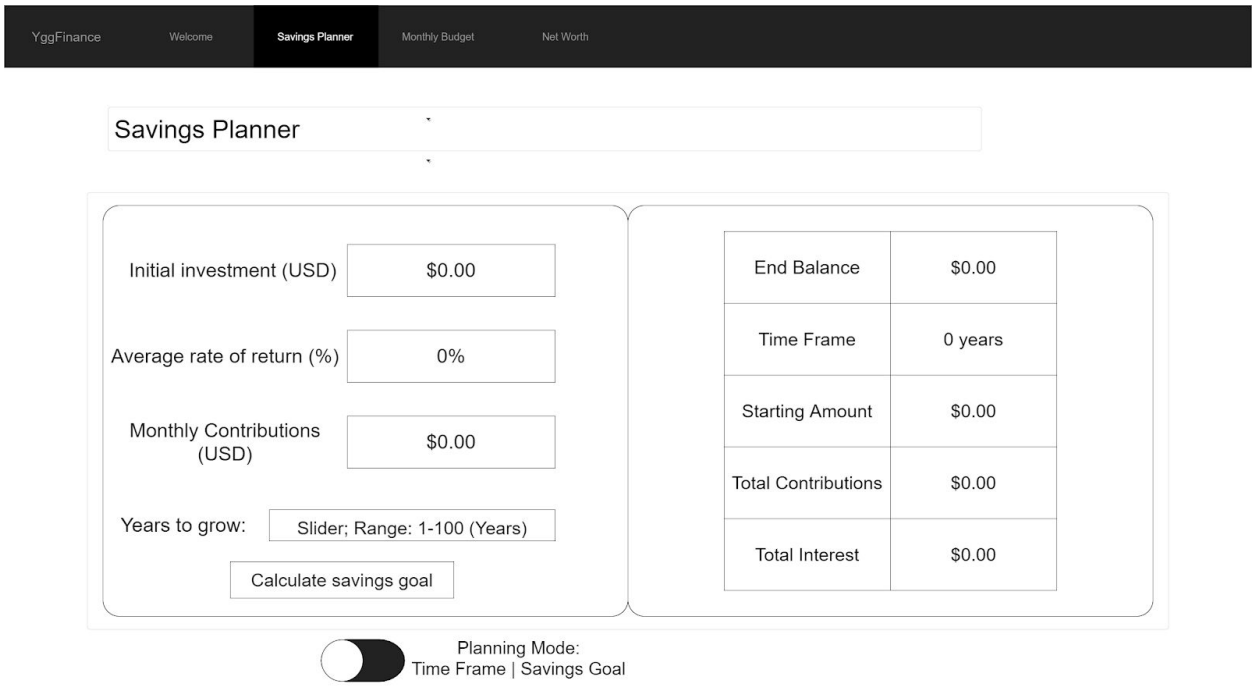
3. **Net Worth Tracking Page**
   - 3.1. **Name: “net-worth-data”**
   - 3.2. **Content {Contract}:**

```
{
    assets: Assets,
    liabilities: Liabilities,
    netWorth: number
}

const Assets = {
    realEstateValue: number,
    checkingAccountsBalance: number,
    savingsAccountsBalance: number,
    retirementAccountsBalance: number,
    automobilesValue: number,
    other: number
}

const Liabilities = {
    remainingMortgageBalance: number,
    consumerDebt: number,
    personalLoans: number,
    autoLoans: number,
    studentLoans: number,
    other: number
}
```

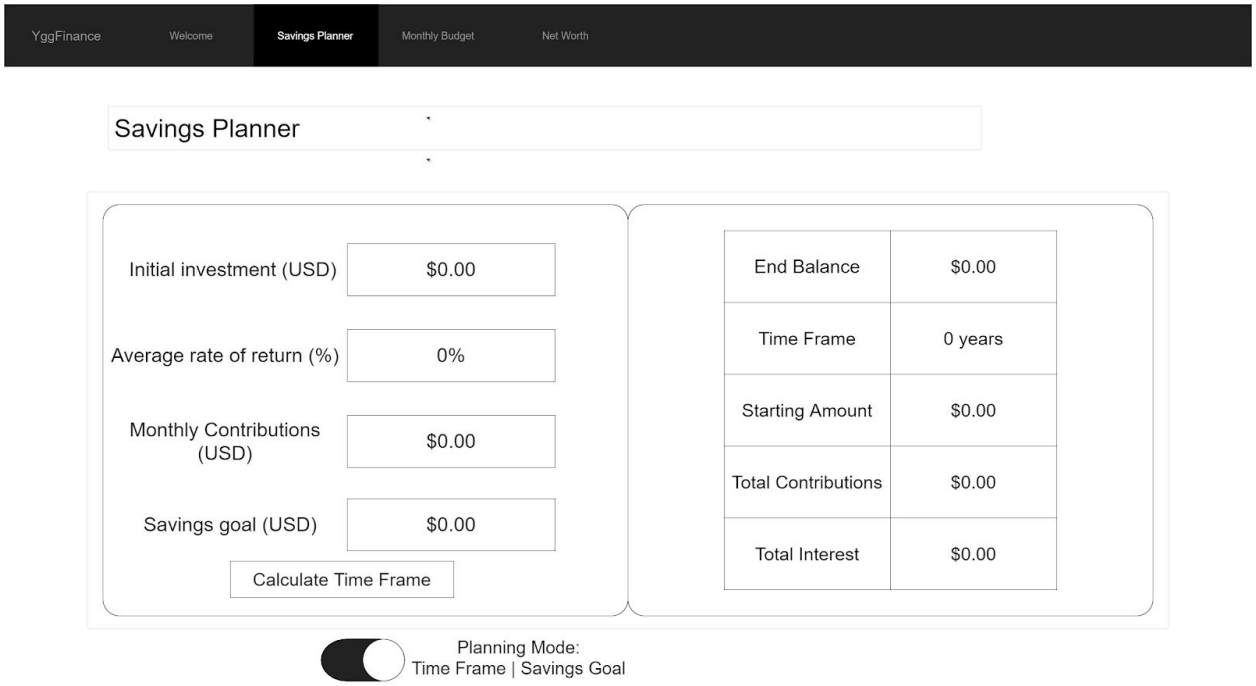
Appendix 4 – UI Wireframe

1. Savings Planner Page

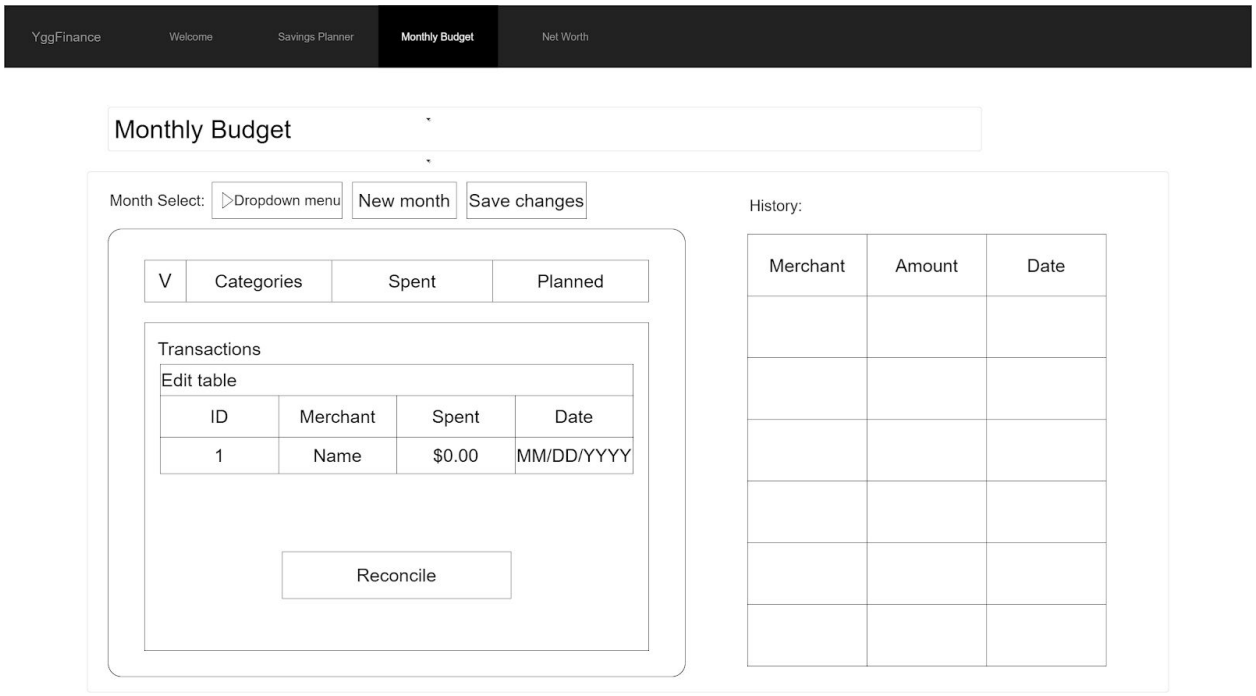
Time Frame Mode:



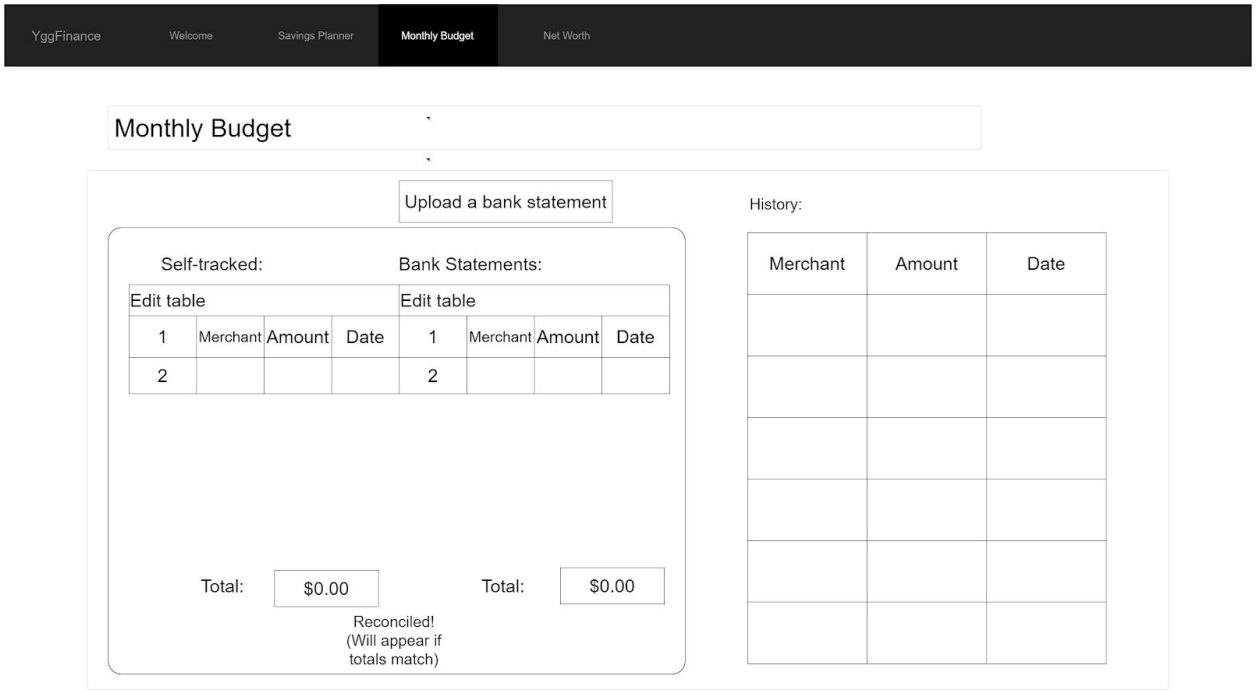
Savings Goal Mode:



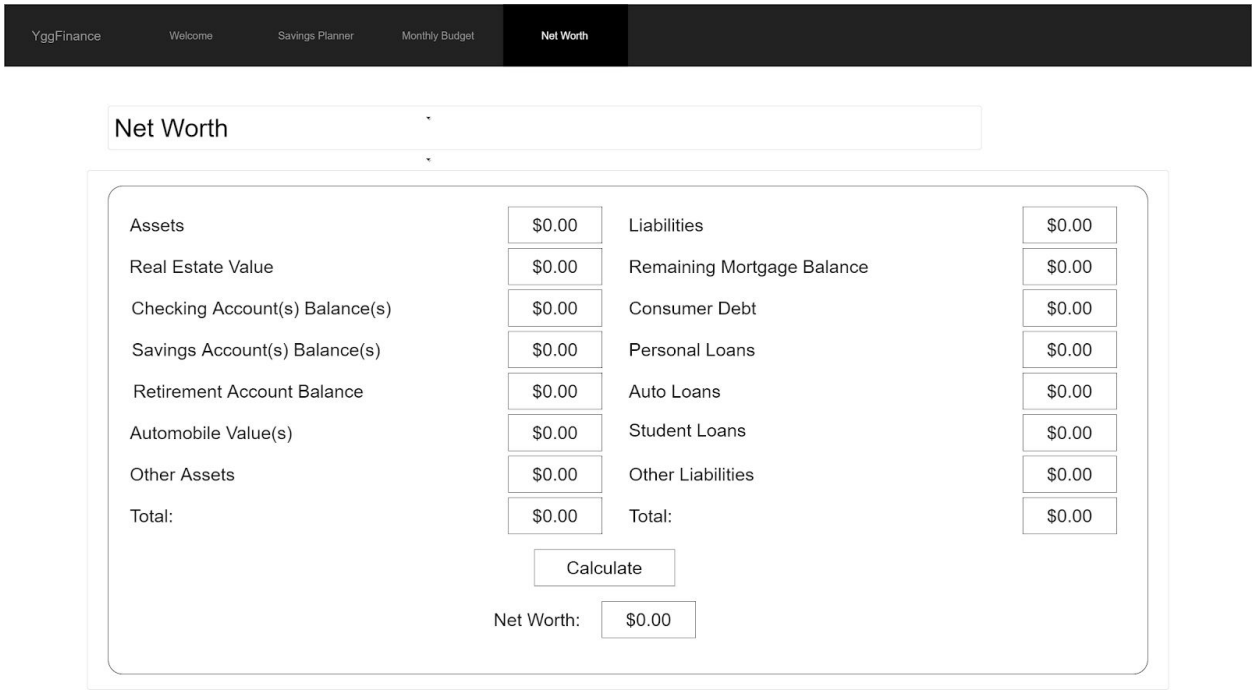
2. Monthly Budget Page



Reconciliation Mode:



3. Net Worth Tracking Page



4. Welcome Page