Savvy Path App Development Process

About:

The app leverages manual data entry to ensure users have full control over their financial tracking and planning.

Savvy is designed to keep users informed and motivated, ensuring they maintain a clear view of their financial trajectory while fostering better financial habits.

Core Functionality-

- Data Input and Organization
  - Users manually input financial data, including income, expenses, and savings, alongside a configurable bank balance. Each financial record includes essential details such as the amount and the date it is applied to the user's account.

- Month-by-Month Financial Insights
  - Current Month: Displays a detailed breakdown of financial activity, highlighting items already applied and those scheduled to apply within the current month.
  - Future Planning: Users can navigate forward to view upcoming months, enabling effective planning by reviewing expected income and expenses.

- Proactive Notifications
  - The app provides notifications to enhance user engagement and financial health:
    - Achievements: Alerts for improvements in income or reductions in expenses.
    - Warnings: Flags potential concerns, such as low bank balances or significant upcoming expenses.

Tech Stack Overview-

Client Stack:

- Typescript
- React & React Native
- TailwindCSS
- React Query
- Zod

Server Stack:

- Bun Runtime (Nodejs Alternative)
- expressjs
- mongoose (MongoDB)
- resend
- zod
- Typescript
- momentjs

DevOps & Tools:

- Docker
- AWS

Behind the scenes-

Once the user's financial data is saved in the **MongoDB** database, the backend server—hosted in a **Docker container on AWS** and running 24/7—takes over. Every day at **midnight**, a scheduled function is triggered.

This function iterates through each user in the database, checking their **income**, **expenses**, and **savings deposits**. It evaluates the **applied date** for each object and, when the date arrives, processes the object accordingly:

- **Income or Savings Deposit**: Adds the amount to the user's **bank balance**.
- **Expense**: Deducts the amount from the **bank balance**.

This daily calculation ensures that users always have an **updated and accurate view** of their financial data. The updates are logged as daily records, allowing users to review a detailed breakdown of their daily activities.

Forecasting and Notifications -

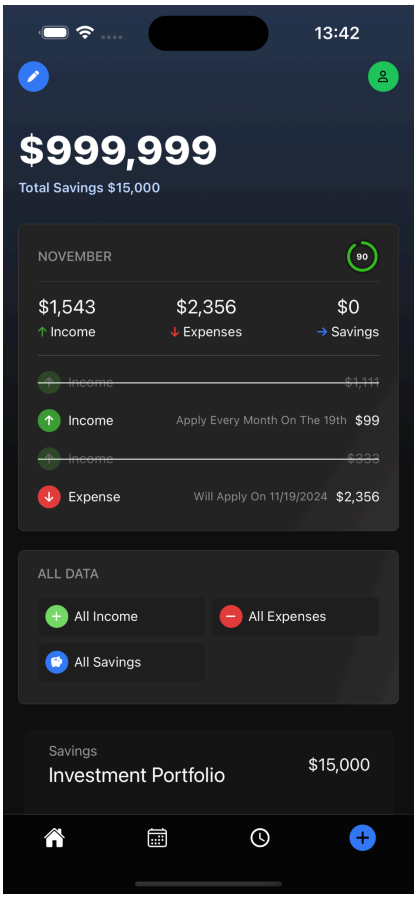
One of the app's standout features is its ability to provide users with a forward-looking financial projection. Users can select a future date—whether it's one month, six months, or even a year ahead—and instantly view their projected financial status for that date.
Once the user inputs their financial data, the backend performs advanced calculations. Given a specific date as input, the system processes all relevant financial objects (income, expenses, and savings) up to that date. The result provides a clear picture of the user's future financial situation, including:

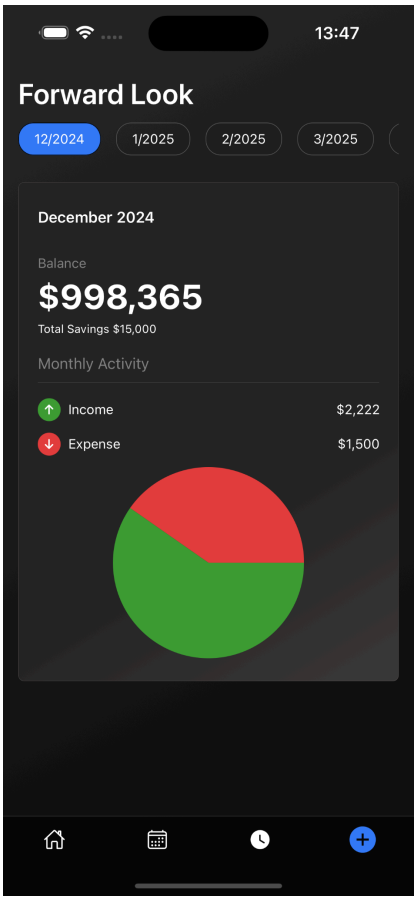
- Total **savings**
- **Income** received
- **Expenses** incurred
- Projected **bank balance**

In addition to these projections, the app ensures users remain well-informed with a notification system. Both current and future data are analyzed to generate actionable alerts:

- **Positive Alerts:** Celebrate milestones such as increased income or strong savings growth.
- **Negative Alerts:** Warn users about potential issues like overdraft risks, unexpected spikes in expenses, or a downward trend in their financial health.
- **Preventative Warnings:** Highlight areas where spending has significantly increased, ensuring users are prepared and aware of upcoming challenges.

This comprehensive forecasting and alert system empowers users to make informed financial decisions, stay proactive, and remain confident in their financial planning.


13:42
$999,999
Total Savings $15,000
NOVEMBER
90
$1,543
Income
$2,356
Expenses
$0
Savings
Income
$1,111
Income
Apply Every Month On The 19th
$99
Income
$333
Expense
Will Apply On 11/19/2024
$2,356
ALL DATA
All Income
All Expenses
All Savings
Savings
Investment Portfolio
$15,000


13:47
Forward Look
12/2024
1/2025
2/2025
3/2025
December 2024
Balance
$998,365
Total Savings $15,000
Monthly Activity
Income
$2,222
Expense
$1,500

Handling Future Month calculation:

```
let bankAmount = userData.value!.bank_amount as number;



   for (let key in incomeData.value) {

     let currentYear = new Date().getFullYear();

     let currentMonth = new Date().getMonth() + 1;

     //generating the dates between the current time and the wanted date.

     const diff = getMonthsAndYearsBetween(

       currentMonth,

       currentYear,

       parsedMonth,

       parsedYear

     );



     while (

       currentYear < parsedYear ||

       (currentYear === parsedYear && currentMonth <= parsedMonth)

     ) {

       currentMonth++;

       if (currentMonth > 12) {

         currentMonth = 1;

         currentYear++;

       }

     }
```

```
      const o = incomeData.value[key];


      if (o.consistent) {
        // Loop through the months and years between the start and end time
        for (const { month, year } of diff) {
          // Check if the month and year is after the end_time
          if (
            (o.end_time?.year && year > o.end_time.year) ||
            (o.end_time?.year === year && month > o.end_time.month)
          ) {
            // Skip this month if the end_time has been reached (the user provided when is the last date of availability)
            continue;
          }
          // Check if the month and year are in the ignore_months array (means the user ignored the month)
          const isIgnored = o.ignore_months?.some(
            (ignore) => ignore.month === month && ignore.year === year
          );
          if (isIgnored) {
            continue;
          }
          // Apply the income value to the bank amount for this month
          bankAmount += o.amount;
```

```
      }
    } else {
      if (o?.active_status === "applied") {
      } else {
        bankAmount += o.amount;
      }
    }
  }
```

Handling the Notification for the future changes of the bank balance:

```
const [savingsActivity, totalExpenses, totalIncome] = await Promise.all([
      SavingModel.aggregate([
        { $match: { userId: req.userId } },
        { $unwind: "$one_time_deposit" },
        {
          $match: {
            "one_time_deposit.month": currentMonth,
            "one_time_deposit.year": currentYear,
            "one_time_deposit.active": true,
          },
        },
        {
          $group: {
            _id: null,
            totalDeposit: { $sum: "$one_time_deposit.amount" },
          },
        },
      ]),
      ExpenseModel.aggregate([
        {
          $match: get_all_monthly_yearly(
            req.userId!,
            currentMonth,
            currentYear
          ),
        },
        {
          $group: {
            _id: null,
            totalExpenses: { $sum: "$amount" },
          },
        },
      ]),
      IncomeModel.aggregate([
        {
          $match: get_all_monthly_yearly(
            req.userId!,
            currentMonth,
            currentYear
          ),
```

```
        },
        {
          $group: {
            _id: null,
            totalIncome: { $sum: "$amount" },
          },
        },
      ]),
    ]);

    const total_deposit = savingsActivity[0]?.totalDeposit || 0;
    const total_expenses = totalExpenses[0]?.totalExpenses || 0;
    const total_income = totalIncome[0]?.totalIncome || 0;
    // Calculating the end month result for the bank balance
    if (user.bank_amount + total_income - total_deposit - total_expenses < 0)
{
      return res.status(200).json({
        success: true,
        positive: false,
        data: `In the end of the month your bank amount will be negative`,
      });
    }
```