CODE:-

```
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <stdexcept>
#include <memory>
#include <iomanip>
#include <limits>

using namespace std;

// Base Transaction Class
class Transaction {
protected:
    double amount;
    string category;
    string date;

public:
    Transaction(double amt, const string& cat, const string& dt)
    {
        amount=amt;
        category=cat;
        date=dt;

        }

    virtual void display() const = 0; // Pure virtual function
    double getAmount() const { return amount; }
    string getCategory() const { return category; }
};

// Derived ExpenseTransaction Class
class ExpenseTransaction : public Transaction {
public:
    ExpenseTransaction(double amt, const string& cat, const string& dt)
        : Transaction(amt, cat, dt) {}

    void display() const override {
        cout << "Expense - Category: " << category << ", Amount: Rs" << fixed << setprecision(2)
<< amount << ", Date: " << date << endl;
    }
};

// Derived IncomeTransaction Class
class IncomeTransaction : public Transaction {
private:
    string source;

public:
    IncomeTransaction(double amt, const string& src, const string& dt)
        : Transaction(amt, "Income", dt) {
            source=src;
        }

    void display() const override {
        cout << "Income - Source: " << source << ", Amount: Rs" << fixed << setprecision(2) <<
amount << ", Date: " << date << endl;
```

```
    }
};

// User Class
class User {
private:
    string name;
    double income;
    vector<shared_ptr<Transaction>> transactions;

public:
    User(const string& userName, double userIncome)
        : name(userName), income(userIncome) {}

    void addTransaction(shared_ptr<Transaction> transaction) {
        if (transaction->getAmount() < 0) {
            throw invalid_argument("Transaction amount cannot be negative.");
        }
        transactions.push_back(transaction);
    }

    void displayTransactions() const {
        cout << name << "'s Transactions:" << endl;
        for (const auto& transaction : transactions) {
            transaction->display();
        }
    }

    double getIncome() const { return income; }

    double calculateTotalExpenses() const {
        double total = 0.0;
        for (const auto& transaction : transactions) {
            if (transaction->getCategory() != "Income") {
                total += transaction->getAmount();
            }
        }
        return total;
    }

    void generateSummaryReport() const {
        cout << "\n--- Summary Report for " << name << " ---" << endl;
        cout << "Total Income: Rs" << fixed << setprecision(2) << income << endl;
        cout << "Total Expenses: Rs" << fixed << setprecision(2) << calculateTotalExpenses() <<
endl;
        cout << "Remaining Balance: Rs" << fixed << setprecision(2)
             << (income - calculateTotalExpenses()) << endl;
        cout << "----------------------------------------" << endl;
    }
};

// Budget Template Class
template<typename T>
class Budget {
private:
    map<string, T> categories;

public:
    void setBudget(const string& category, T amount) {
        categories[category] = amount;
```

```cpp
    }

    void displayBudgets() const {
        cout << "\nBudgets:" << endl;
        for (const auto& pair : categories) {
            cout << pair.first << ": Rs" << fixed << setprecision(2) << pair.second << endl;
        }
    }

    T getBudgetForCategory(const string& category) const {
        auto it = categories.find(category);
        if (it != categories.end()) {
            return it->second;
        } else {
            throw invalid_argument("Budget category not found.");
        }
    }
};

// Recommendation Class
class Recommendation {
public:
    static void suggestSavingsPlan(const User& user) {
        double totalExpenses = user.calculateTotalExpenses();

        // Suggest saving 20% of income
        double savingsGoal = user.getIncome() * 0.20;

        cout << "\n--- Savings Recommendation ---" << endl;
        cout << "Based on your income of Rs"
             << fixed << setprecision(2)
             << user.getIncome()
             << ", we recommend saving at least 20% of your income."
             << endl;

        if (totalExpenses > savingsGoal) {
            cout << "You are currently spending more than your savings goal."
                 << endl;
            cout << "Consider reducing your expenses."
                 << endl;
        } else {
            cout << "You are on track to meet your savings goal!"
                 << endl;
        }

        cout << "----------------------------------------"
             << endl;
    }
};

// Function to get valid input from the user
double getValidDoubleInput(const string& prompt) {
    double value;

    while (true) {
        cout << prompt;
        cin >> value;

        if (cin.fail()) { // Check for invalid input
            cin.clear(); // Clear the error flag
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
            cout << "Invalid input. Please enter a numeric value." << endl;
            continue; // Restart the loop
        }

        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear the buffer
        return value; // Return valid input
    }
}

// Function to get a valid string input from the user
string getValidStringInput(const string& prompt) {
    string input;

    while (true) {
        cout << prompt;
        getline(cin, input);

        if (input.empty()) { // Check for empty input
            cout << "Input cannot be empty. Please enter a valid input." << endl;
            continue; // Restart the loop
        }

        return input; // Return valid input
    }
}

int main() {
    try {
        string userName;

        // Get user name and income
        cout << "Enter your name: ";
        getline(cin, userName);

        double userIncome = getValidDoubleInput("Enter your monthly income: ");

        User user(userName, userIncome);

        char addMoreTransactions = 'y';

        // Adding transactions
        while (addMoreTransactions == 'y') {
            char transactionType;

            cout << "\nEnter transaction type (e for expense, i for income): ";
            cin >> transactionType;
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear the buffer

            if (transactionType == 'e') { // Expense transaction
                double expenseAmount = getValidDoubleInput("Enter expense amount: ");
                string expenseCategory = getValidStringInput("Enter expense category: ");
                string expenseDate = getValidStringInput("Enter expense date (YYYY-MM-DD): ");

                user.addTransaction(make_shared<ExpenseTransaction>(expenseAmount,
expenseCategory, expenseDate));

            } else if (transactionType == 'i') { // Income transaction
                double incomeAmount = getValidDoubleInput("Enter income amount: ");
                string incomeSource = getValidStringInput("Enter income source: ");
                string incomeDate = getValidStringInput("Enter income date (YYYY-MM-DD): ");

                user.addTransaction(make_shared<IncomeTransaction>(incomeAmount, incomeSource,
incomeDate));

            } else {
                cout << "Invalid transaction type. Please enter 'e' or 'i'." << endl;
                continue; // Restart loop for valid input
            }

            cout << "Do you want to add another transaction? (y/n): ";
            cin >> addMoreTransactions;
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear buffer after char input
        }

        // Display transactions and budget setup
        user.displayTransactions();

        Budget<double> budget;

        char setupBudget = 'y';
        while (setupBudget == 'y') {
            string budgetCategory = getValidStringInput("Enter budget category: ");
            double budgetAmount = getValidDoubleInput("Enter budget amount for this category: ");

            budget.setBudget(budgetCategory, budgetAmount);

            cout << "Do you want to add another budget? (y/n): ";
            cin >> setupBudget;
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear buffer after char input
        }

        // Display budgets and generate summary report
        budget.displayBudgets();
        user.generateSummaryReport();

        // Provide savings recommendation based on user's data
        Recommendation::suggestSavingsPlan(user);

    } catch (const exception& e) {
        cerr << "Error: " << e.what() << endl;
    }

    return 0;
}
```