```
import csv
import os
from datetime import datetime

# --- Global Data Storage ---
# Stores all expense dictionaries
EXPENSES = []
# Stores the monthly budget, loaded/saved separately or set by user
MONTHLY_BUDGET = 0.0
# Filename for persistence
FILE_NAME = 'expenses.csv'
BUDGET_FILE = 'budget.txt'

# --- Utility Functions for File Handling (Data Persistence) ---

def load_budget():
    """Loads the monthly budget from budget.txt."""
    global MONTHLY_BUDGET
    try:
        with open(BUDGET_FILE, 'r') as f:
            # Read and convert to float, default to 0.0 if file is empty
            MONTHLY_BUDGET = float(f.read().strip() or 0.0)
    except FileNotFoundError:
        print(f"INFO: Budget file '{BUDGET_FILE}' not found. Defaulting budget to $0.00.")
        MONTHLY_BUDGET = 0.0
    except ValueError:
        print("WARNING: Could not read budget amount. Defaulting budget to $0.00.")
        MONTHLY_BUDGET = 0.0

def save_budget():
    """Saves the current monthly budget to budget.txt."""
    try:
        with open(BUDGET_FILE, 'w') as f:
            f.write(str(MONTHLY_BUDGET))
    except Exception as e:
        print(f"ERROR: Failed to save budget to file. {e}")

def load_expenses():
    """Reads expense data from the CSV file and populates the global list."""
    global EXPENSES
    EXPENSES.clear()

    # Check if the file exists
    if not os.path.exists(FILE_NAME):
        print(f"INFO: Expense file '{FILE_NAME}' not found. Starting with an empty list.")
        return

    try:
        with open(FILE_NAME, mode='r', newline='') as file:
            reader = csv.DictReader(file)
            for row in reader:
                # Basic data validation and type conversion
                try:
                    expense = {
                        'date': row.get('date'),
                        'category': row.get('category'),
                        # Convert amount to float for calculations
                        'amount': float(row.get('amount')),
                        'description': row.get('description', '')
                    }

                    # Validate all required fields are present and not empty
                    if all(expense.get(key) for key in ['date', 'category', 'amount']):
                        EXPENSES.append(expense)
                    else:
                        print(f"WARNING: Skipping incomplete entry: {row}")

                except ValueError:
                    print(f"WARNING: Skipping entry due to invalid amount format: {row}")

        print(f"\nSUCCESS: Loaded {len(EXPENSES)} expenses from '{FILE_NAME}'.")
    except Exception as e:
        print(f"ERROR: Failed to load expenses from file. {e}")

def save_expenses():
    """Saves all current expenses to the CSV file."""
    if not EXPENSES:
        print("INFO: No expenses to save. File may be cleared.")
        # If there are no expenses, still create an empty file with headers

    # Define the fieldnames (column headers)
```

```
    # Define the fieldnames (column headers)
    fieldnames = ['date', 'category', 'amount', 'description']

    try:
        # Use 'w' mode to overwrite the file with the latest data
        with open(FILE_NAME, mode='w', newline='') as file:
            writer = csv.DictWriter(file, fieldnames=fieldnames)

            # Write the header row
            writer.writeheader()

            # Write all expense records
            writer.writerows(EXPENSES)

        print(f"\nSUCCESS: All {len(EXPENSES)} expenses and budget saved successfully.")
        save_budget() # Also save the budget
    except Exception as e:
        print(f"ERROR: Failed to save expenses to file. {e}")

# --- Core Application Functions ---

def add_expense():
    """Prompts the user for expense details and adds it to the list."""
    print("\n--- ADD NEW EXPENSE ---")

    # 1. Date Input and Validation (YYYY-MM-DD)
    while True:
        date_str = input("Enter date (YYYY-MM-DD, e.g., 2024-12-05): ").strip()
        try:
            # Check if the date format is valid
            datetime.strptime(date_str, '%Y-%m-%d')
            break # Exit loop if valid
        except ValueError:
            print("Invalid date format. Please use YYYY-MM-DD.")

    # 2. Category Input
    category = input("Enter category (e.g., Food, Travel, Bills): ").strip()
    if not category:
        print("Category cannot be empty. Returning to menu.")
        return

    # 3. Amount Input and Validation (must be a positive number)
    while True:
        amount_str = input("Enter amount spent: ").strip()
        try:
            amount = float(amount_str)
            if amount <= 0:
                print("Amount must be a positive number.")
                continue
            break # Exit loop if valid
        except ValueError:
            print("Invalid amount. Please enter a number.")

    # 4. Description Input
    description = input("Enter a brief description: ").strip()

    # Create the expense dictionary
    new_expense = {
        'date': date_str,
        'category': category,
        'amount': amount,
        'description': description
    }

    # Store the expense
    EXPENSES.append(new_expense)
    print("\n[✓] Expense added successfully!")

def view_expenses():
    """Retrieves and displays all stored expenses in a formatted table."""
    print("\n--- VIEW ALL EXPENSES ---")

    if not EXPENSES:
        print("No expenses recorded yet.")
        return

    # Calculate column widths for clean formatting
    col_widths = {
        'date': 12,
        'category': 15,
        'amount': 10,
        'description': 40
    }
```

```
    # Header
    header = (f"{'Date':<{col_widths['date']}} | "
              f"{'Category':<{col_widths['category']}} | "
              f"{'Amount':<{col_widths['amount']}} | "
              f"{'Description':<{col_widths['description']}}")

    separator = "-" * len(header)

    print(separator)
    print(header)
    print(separator)

    # Loop through and display each entry
    for expense in EXPENSES:
        # Data Validation check (as required by the problem statement)
        if not all(expense.get(key) for key in ['date', 'category', 'amount', 'description']):
            print(f"WARNING: Incomplete entry skipped: {expense}")
            continue

        # Format amount as currency string
        amount_str = f"${expense['amount']:.2f}"

        # Truncate description if too long
        description_fmt = expense['description'][:col_widths['description']].ljust(col_widths['description'])

        row = (f"{expense['date']:<{col_widths['date']}} | "
               f"{expense['category']:<{col_widths['category']}} | "
               f"{amount_str:>{col_widths['amount']}} | "
               f"{description_fmt}")

        print(row)

    print(separator)

def calculate_total_expenses():
    """Calculates the sum of all recorded expense amounts."""
    # Ensure amount is float before summing
    return sum(e['amount'] for e in EXPENSES if isinstance(e.get('amount'), (int, float)))

def track_budget():
    """Allows setting a monthly budget and tracks spending against it."""
    global MONTHLY_BUDGET
    print("\n--- TRACK MONTHLY BUDGET ---")

    # Prompt to set a new budget
    while True:
        try:
            current_budget = MONTHLY_BUDGET
            prompt = (f"Enter total monthly budget (Current: ${current_budget:.2f}). "
                      "Enter 0 to keep current budget, or a new amount: ").strip()

            new_budget_str = input(prompt)

            if new_budget_str == '0':
                # User wants to keep the current budget
                break

            new_budget = float(new_budget_str)
            if new_budget < 0:
                print("Budget cannot be negative. Please enter a non-negative amount.")
                continue

            MONTHLY_BUDGET = new_budget
            print(f"\n[✓] Monthly budget set to: ${MONTHLY_BUDGET:.2f}")
            break

        except ValueError:
            print("Invalid input. Please enter a number for the budget.")

    # Calculate and display the status
    total_expenses = calculate_total_expenses()
    print(f"\nTotal Expenses Recorded: ${total_expenses:.2f}")
    print(f"Monthly Budget Limit:    ${MONTHLY_BUDGET:.2f}")

    if MONTHLY_BUDGET > 0:
        remaining_balance = MONTHLY_BUDGET - total_expenses

        if remaining_balance < 0:
            print("\n!!! BUDGET WARNING !!!")
            print(f"You have **EXCEEDED** your budget by: ${abs(remaining_balance):.2f}")
        else:
            print("\n[✓] Budget Status")
```

```
            print(f"Remaining balance for the month: ${remaining_balance:.2f}")
    else:
        print("\nINFO: No monthly budget has been set.")


# --- Main Application Menu ---

def main_menu():
    """The main interactive, menu-driven interface."""

    # Load data on startup
    load_expenses()
    load_budget()

    while True:
        print("\n" + "="*40)
        print("PERSONAL EXPENSE TRACKER MENU")
        print("="*40)
        print("1. Add expense")
        print("2. View expenses")
        print("3. Track budget")
        print("4. Save expenses")
        print("5. Exit (Saves and quits)")
        print("="*40)

        choice = input("Enter your choice (1-5): ").strip()

        if choice == '1':
            add_expense()
        elif choice == '2':
            view_expenses()
        elif choice == '3':
            track_budget()
        elif choice == '4':
            save_expenses()
        elif choice == '5':
            print("\nSaving expenses and exiting the program. Goodbye!")
            save_expenses()
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 5.")

# --- Program Entry Point ---
if __name__ == "__main__":
    main_menu()
```