Arab Academy for Science & Technology & Maritime Transport

College of Engineering and Technology

Smart Village Campus

Department of Computer Engineering

Supervisor: Dr.Amira Gaber Mahmoud

TA: Eng. Farah Labib

Subject: Object-Oriented Programming

Calendar App Report

Mahmoud Sharaby , ID:231017889

Omar Ahamd Mohamad, ID:221007593

Abdelrahman Mohamed Badr, ID:222007596

Introduction:

In today's world, scheduling and planning our day is a vital part of our life. Thus, many people look for an easy and fast-going application to remind them about their scheduling and planning for the day, such as meetings and events. This project is developing a simple calendar application using C# as the base language for building and developing the app. In addition, using Windows Forms to create the Graphical user Interface(GUI). C# is an object-oriented language(oop), hence, the use of classes, encapsulation, inheritance, polymorphism and abstraction is depicted throughout the project illustrating the main concepts of oop.

Problem definition:

Scheduling and planning have become a headache for the person to remember every detail and every meeting s/he has set up. As a result, they use an application for that. This leads to why this project team decided to develop a reminder/calendar application.

Methodology:

This project explores and depicts the use of the four main concepts of OOP encapsulation, inheritance, polymorphism and abstraction. Encapsulation is the wrapping up blocks of codes in a single unit and limiting the access available for the unit as the developer wishes. Inheritance is the allowance of creating a base class that other classes can inherit, creating a reusability of the code. Polymorphism is the presence of two or more variant forms of a specific function from the base. On the other hand, abstraction is the process of identifying common patterns that have systematic variations in code. The application uses those concepts in developing a relation and a connection between different classes and Windows forms.

Structure of the Application:

User :

Overview

The provided code defines a class structure for managing user accounts in a Windows Forms application. It includes an abstract base class User_account and a derived class User_Preference that handles user data and interactions with a database.

Class: User_account (Abstract)

Purpose:

Serves as a base class for different types of user accounts, providing common properties and methods.

Properties:

- User_Name (protected): The name of the user.
- Email (public): The email address of the user.
- Password (protected): The password for the user account.
- Birthdate (protected): The birthdate of the user.

Methods:

DisplayUser() (virtual):

- Purpose: Displays the user's information to the console.
- Parameters: None.
- Return: None.
- Implementation: Prints User_Name, Email, and Birthdate to the console.

database_load(string s) (abstract):

- Purpose: Loads user data from the database.
- Parameters: s (string) - likely a user identifier.
- Return: A string indicating success or failure.

Login(string password) (abstract):

- Purpose: Authenticates the user with the provided password.
- Parameters: password (string) - the password entered by the user.
- Return: A boolean indicating whether the login was successful.

Class: User_Preference (Inherits from User_account)

Purpose:

Manages user preferences and provides implementations for database interactions.

Methods:

database_load(string s) (override):

- Purpose: Loads user data from the database using the database_handeler.
- Parameters: s (string) - likely a user identifier.
- Return: "1" if successful, null otherwise.
- Implementation: Retrieves user data from the database and assigns it to the properties.

Create_User(string name, string email, string date, string pass):

- Purpose: Creates a new user account in the database.
- Parameters:
    - name (string): The user's name.
    - email (string): The user's email.
    - date (string): The user's birthdate.
    - pass (string): The user's password.
- Return: Boolean indicating whether the user was created successfully.
- Implementation: Adds the user to the database and returns a boolean based on the operation's success.

Display_User(User_account user) (static):

- Purpose: Displays the user's information using the DisplayUser() method.
- Parameters: user (User_account) - the user instance to display.
- Return: None.

Login(string pass) (override):

- Purpose: Authenticates the user by comparing the provided password with the stored password.
- Parameters: pass (string) - the password entered by the user.
- Return: Boolean indicating whether the password matches.

change_Password(string current_password, string new_password):

- Purpose: Allows the user to change their password if the current password is correct.
- Parameters:
    - current_password (string): The user's current password.
    - new_password (string): The new password to set.
- Return: Boolean indicating whether the password was changed successfully.
- Implementation: Updates the password in the database if the current password is correct.

forget_Password(string email, string birth):

- Purpose: Retrieves the user's password if the provided email and birthdate match.
- Parameters:
    - email (string): The user's email.
    - birth (string): The user's birthdate.
- Return: The user's password if the details match, otherwise null.

delete_User():

- Purpose: Deletes the user's account from the database.
- Implementation: Removes the user from the database.
- Takes no parameters

Database Interaction :

The database_handeler class, located in the WindowsFormsApp1.database namespace, is responsible for interacting with the database. It provides the following methods:

- loaduser(string s): Loads user data from the database.
- add(string name): Adds a new user to the database.
- adduser(string name, string password, string birthdate, string email): Adds a new user with detailed information.
- edit_pass(string name, string password, string email, string birthdate): Updates the user's password in the database.
- removeuser(string name): Removes the user from the database.

Class : Checklist

Overview

The provided code implements a checklist feature in a Windows Forms application, managing tasks and their completion status. It interacts with a database to save and load tasks, and displays the current date using a custom Calendar class

Checklist

- Purpose: Container class for managing checklist data.
- Nested Class: Checklist
    - Data Members:
        - DataTable todo: Stores tasks and their completion status.
- Methods:
    - create_table(): Initializes DataTable with "Tasks" and "Completed" columns.
    - add_rows(string task, string x): Adds a task row, converting x to boolean for "Completed".

Check_do Form

- Purpose: User interface for interacting with the checklist.
- Components:
    - DataGridView dataGridView1: Displays tasks.
    - TextBox textBox1: Input field for tasks.
    - Buttons (button1, button2, button3, button4): Handle add, edit, delete, and toggle completion actions.
    - Label label2: Displays today's date.
    - PictureBox pictureBox1: Navigation button to another form.

Checklist Methods:

- create_table(): Sets up the DataTable with necessary columns and returns it.
- add_rows(string task, string x): Adds a task row, ensuring x is a valid boolean string.

Check_do Form Methods:

- Check_do(): Constructor initializes UI components and loads tasks from the database.
- button1_Click: Adds a new task with "false" for completion status.
- button2_Click: Enables editing mode by setting is_editing to true.
- button3_Click: Removes a selected task from the DataTable and database.
- button4_Click: Toggles completion status or updates an edited task.
- pictureBox1_Click: Navigates to another form (Form2).

Usage Examples

- Adding a Task:
    - Enter task in textBox1.
    - Click button1 to add the task to the checklist and database.

- Editing a Task:
    - Select the task in dataGridView1.
    - Click button2 to enable editing.
    - Modify the task in textBox1.
    - Click button4 to update the task in the checklist and database.

- Deleting a Task:
    - Select the task in dataGridView1.
    - Click button3 to remove the task from the checklist and database.

Potential Improvements

- Validation:
    - Add validation for empty task entries.
    - Ensure x in add_rows is a valid boolean string.
- User Experience:
    - Implement confirmation dialogs for deletions.
    - Provide visual cues for editing mode.
- Exception Handling:
    - Handle potential exceptions in database operations.
    - Manage invalid boolean conversions in add_rows.

Notification :

Purpose

The provided code defines a notification system for sending automated emails to users. It is part of the project and uses the `System.Net.Mail` library to handle email communication. The code encapsulates its functionality within the `notification` class.

Key Features

1. **Forgot Password Email**:
    - Method: `sent_email_forget_pass(string pass, string email, string name)`
    - Purpose: Sends an email to a user containing their forgotten password.
    - Email Format:
        - Subject: "password"

        - Body: Includes the user's name and password.
2. **Account Deletion Email**:
    - Method: `sent_email_delete(string email, string name)`
    - Purpose: Notifies the user that their account has been deleted.
    - Email Format:
        - Subject: "Delete"
        - Body: Includes the user's name, a message about the account deletion, and a farewell note.
3. **Signup Welcome Email**:
    - Method: `sent_email_signup(string email, string name)`
    - Purpose: Welcomes a new user after signing up for a service.
    - Email Format:
        - Subject: "Signup"
        - Body: Includes the user's name, a welcome message, and a positive closing note.

How It Works

- **Email Initialization**: Each method constructs a `MailMessage` object with:
    - A "from" email address.
    - A "to" email address .
    - Subject and body content tailored to the specific purpose of the email.
- **SMTP Client Setup**: Each method creates a `SmtpClient` object configured to:
    - Use Gmail's SMTP server (`smtp.gmail.com`) on port 587.
    - Authenticate with hardcoded credentials.
    - Enable SSL for secure communication.
    - Send the email via the configured SMTP server.
- **Email Priority**: All emails are sent with high priority (`MailPriority.High`).

Observations

- **Functionality**:
    - The code implements basic email notification functionality, making it suitable for small applications requiring password recovery, account management, and user onboarding communications.
- **Customizability**:
    - The subject and body content can be customized to fit the branding or tone of the service.

Limitations

1. **Hardcoded Credentials**:
    - Gmail credentials are hardcoded, posing a significant security risk.
    - Recommendation: Store credentials in environment variables or a secure configuration file.
2. **Repetitive SMTP Configuration**:
    - SMTP setup is repeated in each method.
    - Recommendation: Extract the configuration into a reusable helper method.
3. **No Error Handling**:
    - The code does not handle exceptions, which could lead to unhandled errors if the email fails to send.
    - Recommendation: Add `try-catch` blocks to manage errors gracefully.
4. **Scalability**:
    - Currently limited to one email sender.
    - Recommendation: Allow dynamic configuration of the "from" email address for better scalability.
5. **Plain Text Passwords**:
    - User passwords are sent in plain text, which is insecure.

Database Handler:

Overview

The code implements a `database` handling system. It provides functionality for managing user information, files, and tasks. The system revolves around text file storage for data persistence, making it suitable for small-scale applications without requiring a full-fledged database.

Features

File Management

1. **File Initialization**:
    - A static constructor initializes the `filenames` list by reading from a `filenames.txt` file.
    - Removes empty entries from the `filenames` list during initialization.

2. **Adding Files**:
    - Method: `add(string name)`
    - Adds a filename to `filenames.txt` if it doesn't already exist.
3. **Removing Files**:
    - Method: `remove(string name)`
    - Deletes a filename from the `filenames` list and updates `filenames.txt`.
4. **Display Files**:
    - Method: `display()`
    - Outputs the contents of `filenames.txt` to the console.

User Management

1. **Add User**:
    - Method: `adduser(string name, string pass, string brith, string email)`
    - Creates a text file for a user and stores their password, birth date, and email.
2. **Remove User**:
    - Method: `removeuser(string name)`
    - Deletes the user's main file and a related secondary file (`name2.txt`).
3. **Load User Data**:
    - Method: `loaduser(string name)`
    - Reads a user's data (password, birth date, and email) from their file and returns it as an array.
4. **Edit User Password**:
    - Method: `edit_pass(string name, string pass, string email, string birth)`
    - Updates a user's password, birth date, and email in their file.

Calendar Management

1. **Load Calendar Events**:
    - Method: `loadclr(string name, string day, int year, int month)`
    - Loads events and notes from a user-specific calendar file.
2. **Add Calendar Event**:
    - Method: `addclr(string name, int day, int year, int month, string Event, string note, int num)`

- Adds an event and its corresponding note to the user's calendar file.

Task Management

1. **Load Tasks**:
    - Method: `load_check(string name)`
    - Reads tasks from a user's task file (`name2.txt`).
2. **Add Task**:
    - Method: `addtodo(string task, bool x, string name)`
    - Adds a task and its completion status to the user's task file.
3. **Update Task Status**:
    - Method: `check_bool(string x, string task, string name)`
    - Updates the completion status of a specific task in the user's task file.
4. **Remove Task**:
    - Method: `removetodo(string name, string task, string x)`
    - Removes a task and its status from the user's task file.

Strengths

1. **Simplicity**:
    - Uses plain text files for data storage, making it easy to understand and implement.
2. **Separation of Concerns**:
    - Divides responsibilities across multiple methods for user, calendar, and task management.
3. **Customizability**:
    - Can be adapted for additional functionality or storage formats.

Weaknesses and Recommendations

1. **No Error Handling**:
    - The code lacks exception handling, which could lead to crashes if files are missing or inaccessible.
    - **Recommendation**: Add `try-catch` blocks around file operations.
2. **Hardcoded File Paths**:
    - File paths and names are hardcoded, limiting flexibility.
    - **Recommendation**: Use a configuration file or constants for file paths.

3. **Data Security**:
    - Storing passwords in plain text is insecure.
    - **Recommendation**: Use hashed passwords for storage.
4. **Inefficient Updates**:
    - Removing and rewriting files for updates is inefficient.
    - **Recommendation**: Use a database or structured storage format (e.g., JSON or XML).
5. **Scalability**:
    - The reliance on text files makes the system unsuitable for large datasets or concurrent access.
    - **Recommendation**: Transition to a relational database system like SQLite or MySQL.
6. **Repetitive Code**:
    - File operations and similar logic are duplicated across methods.
    - **Recommendation**: Refactor common logic into helper methods.

The Calendar

The calendar is the heart of this project; it can shuffle through dates and take in the date with a single click, saving multiple events and any notes on that date.

First, there is the calendar class itself which contains all the methods and constructors.

Constructors:

There are quite a few constructors, most of them are to get to day day via the “date.now” predefined method.

Methods:

1. **Pre:**

    The previous month and year are obtained by decreasing the month until it reaches the first month of the year, then setting the month to the last month and decreasing the year.

2. **Next:**

   In order to obtain the next month and year, it increases the current month until it reaches the last month of the year, then resets the month to the first month and increases the current year.

3. **First_day:**

   Gets the first day of the week using the “dayofweek” predefined method.

4. **Convert:**

   This method is overloaded to either convert month, year, and day to datetime format or convert datetime format to month, year, and day.

5. **Month_length**

   Gets the number of days in that month using the GregorianCalendar class and the GetDaysInMonth predefined method.

6. **Month_Name**

   aquifers the month name from the .tostring(“MMM”) method.

Calendar Forms

Calendar:

The calendar application consists of two forms, each containing a dedicated user control. The primary form, referred to as the Calendar Form, serves as the initial interface. It features four buttons:

1. **Refresh Button**: Quickly refreshes the form by closing and reopening it.
2. **Previous Button**: Implements the pre method to navigate to the previous view.
3. **Next Button**: Implements the next method to move to the next view.
4. **Home Button**: Redirects to the Home Form.

A user control named day_tab is responsible for displaying the calendar. It dynamically generates placeholder user controls until reaching the first day of the week, determined by the First_day method. Subsequently, it increments the displayed date by one for each day until the end of the month, which is identified using the Month_length method.

Events:

Adding events to the calendar is straightforward. Clicking on a specific day's user control opens an events form. Within this form, pressing the Add Button creates a new user control for entering event details, including:

- Event name
- Notes about the event
- A **Save Button** to store the event details.

Saved events are written to a file to ensure data persistence, even when the application is closed. Additionally, this data is shared with the Calendar Form, enabling the `day_tab` control to display up to two saved events per day. The application supports creating up to nine events for a single day.

Conclusion:

In conclusion, this calendar and reminder application successfully demonstrates the principles of object-oriented programming while providing essential functionalities such as event scheduling, notifications, and data management through a user-friendly Windows Forms interface. While the project excels in modularity and simplicity, addressing critical areas like data security, scalability, and code efficiency will further enhance its robustness and adaptability. By implementing recommended improvements, this application has the potential to transition from a basic tool to a secure and scalable solution for effective scheduling and planning.