Home Base
for UTM CSCI 352 Fall 2018

Adam Chisolm
Lukas Saul

Abstract

Our intention with this project is to create a simple-to-use calendar/reminder program to run on a desktop/laptop computer. It will keep track of events, reminders, and the weather while also being more customizable than other similar programs.

1. Introduction

Our goal for this program is to create a calendar system that will integrate locally stored events and reminders with Google Calendar, through the provided Google Calendar API. We expect people who struggle keeping track of their busy schedules to run the program on startup so that they can be reminded of their schedule, weather forecasts, reminders, etc. so they may be kept on track and prepare for the day. We hope that people will find that their lives are made a little easier having this program reminding them of what they need to do.

The ultimate goals for this semester regarding this program is to create an independent reminder and event system that will help someone keep track of their busy lives. We also want to integrate a weather forecast so that one may be better prepared once they leave the house.

1.1. Background

This program will be fairly similar to Google Calendar or Window's calendar even but much more customizable in that you will be able to change the background and layout to a couple different presets. We have witnessed that a great many users have trouble staying on track with their busy schedules, which is why we'll be creating this app to help people stay organized. We plan to use the Google Calendar API to allow for people to keep this calendar

1.2. Challenges

We found that WPF applications have quite a few limitations that impeded on our development of this application with the given time and our experience with them. We found that the calendar tool provided was very restrictive in terms of visual customization so we opted to build our own. This was fairly difficult and ate up a lot of dev time. We also struggled with getting the visual customization running using our calendar. In the end, we didn't get a chance to implement much of the customization aspects or the Google Calendar synchronization due to time constraints.

2. Scope

This project will be finished once the user can store an event into the calendar with the event showing up on the right day. Another factor in the project's completion is the ability to add and remove/check-off tasks from their TODO List. Once these two events have been added the core of the program will be done.

2.1. Requirements

We developed these requirements from what we have seen as good ideas in calendar programs and what we feel like most users would like to see in a calendar program.

2.1.1. Functional.

- The program will need to store events locally and keep track of them.
- The user should be able to alter the aesthetic of the program via background pictures, colors, sizes, layout.
- The user should be able to add a "reminder" or short term event, and be notified of its happening on the desktop.
- The user should be able to add a task to their TODO List so they may keep track of the completion of said task.

Use Case ID	Use Case Name	Primary Actor	Complexity	Priority
1	Add event to calendar	User	Med	1
2	Delete event from calendar	User	Med	1
3	Add task to TODO List	User	Easy	1
4	Remove task from TODO List	User	Med	1
5	Check off a task from the TODO List	User	Easy	2

TABLE 1. USE CASE TABLE

2.1.2. Non-Functional.

- Reliability – user created events must be easily accessed again and again without the loss of an event.
- Backup – user created events must be able to be stored as a whole to local memory in case of failure.

2.2. Use Cases

The User should be able to add a task/reminder to their calander. They should then be able to ”check off” the task if they completed it, or delete the task if they are no longer planning on completing it. See Table 1.

Use Case Number: 1
Use Case Name: Add event to calendar.
Description: A user of the calendar has figured out an event they need to be reminded of. They will click on the Add Event button which will begin the process for adding an event to the calendar.

1) User navigates to the ”Add Event” Button.
2) User left-clicks on ”Add Event” button.
3) The User is then directed to a pop-up window where they will input the information for the event. Figure 2

Termination Outcome: The User has now added an event to their calendar which will be reflected on the calendar.

Alternative: That event already exists.

1) User navigates to the ”Add Event” Button.
2) User left-clicks on ”Add Event” button.
3) The User is then directed to a pop-up window where they will input the information for the event.

Termination Outcome: The user will be shown an error message alerting them to the fact that the event already exists. User will be directed back to the calendar.

Use Case Number: 2
Use Case Name: Delete event from calendar.
Description: A user has either made an error in adding an event or an event has passed and they need to remove it. They will click on the Remove Event button which will begin the process for removing an event from the calendar. Figure 3

1) User navigates to the ”Remove Event” Button.
2) User left-clicks on ”Remove Event” button.
3) The User is then directed to a pop-up window where they will input the information for the event.

Termination Outcome: The User has now removed an event to their calendar which will be reflected on the calendar.

Alternative: That event doesn’t exist.

1) User navigates to the ”Remove Event” Button.
2) User left-clicks on ”Remove Event” button.
3) The User is then directed to a pop-up window where they will input the information for the event.

Termination Outcome: The User will be shown an error message alerting them to the fact that the event doesn’t exist and will be directed by to their calendar.

Use Case Number: 3
Use Case Name: Add task to TODO List.
Description: A user has decided there is a task they would like to keep their completion of tracked. They will choose a List to add it to and begin adding it to their chosen list.

1) User navigates to the List of TODO lists.
2) User left-clicks on the TODO lists button.
3) The User is then types whatever it is they have to do in the provided space.

Termination Outcome: The User has now added a task to their specified list in the application.

Use Case Number: 4

Use Case Name: Remove task fromTODO List.
Description: A user has decided there is a task they would like to remove from their TODO list. This will start the process for deleting a task from the TODO List.

1) The User navigates to the TODO List.
2) The User left-clicks on the check box next to the the item in the todo list they wish to delete.
3) The User then clicks the ”Delete Task” Button to remove the task from the list

Termination Outcome: The User has now removed the task list in the application.

Use Case Number: 5
Use Case Name: Check off task fromTODO List.
Description: A user has determined that a task on their TODO List has been completed. This will start the process for deleting a task from the TODO List.

1) The User navigates to the TODO List.
2) The User left-clicks on the check box next to the the item in the todo list they wish to delete.
3) The User then clicks the ”Task Completed” Button to cross out the task from the TODO List

Termination Outcome: The User has now checked the task off on the list in the application.

2.3. Interface Mockups

Main Interface 1

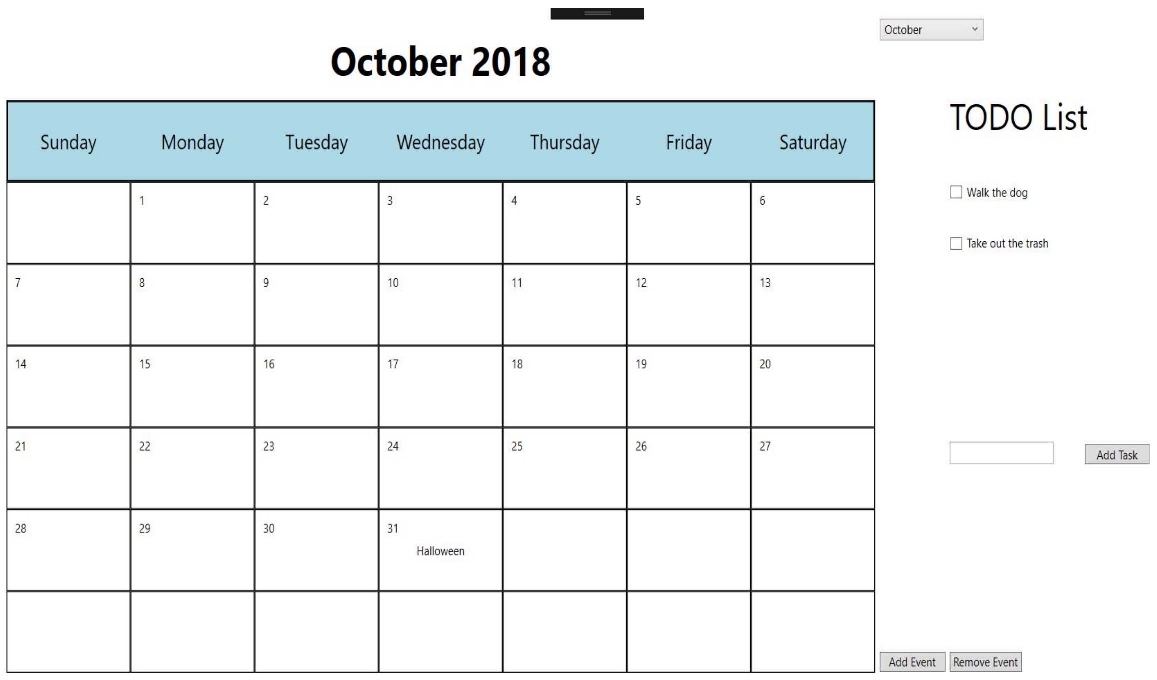

Figure 1. Picture of the interface

Here the user is greeted with the main screen. Calendar on the left and TODO list on the right. The month can be changed using the dropbox to the upper right of the calendar and events can be added or removed using the buttons to the bottom right of the calendar. Tasks can be added to the TODO list via the button and text box below the TODO list area, tasks are removed when the checkbox has been checked. 1

Event Addition 2

Here a user may add an event to the calendar by inputting the information into the textboxes and clicking Add Event. 2

Event Removal 3

Here a user may remove an event fromt the calendar by putting the event’s information into the textboxes and clicking Remove Event. 3

3. Project Timeline

Here is our project timeline.

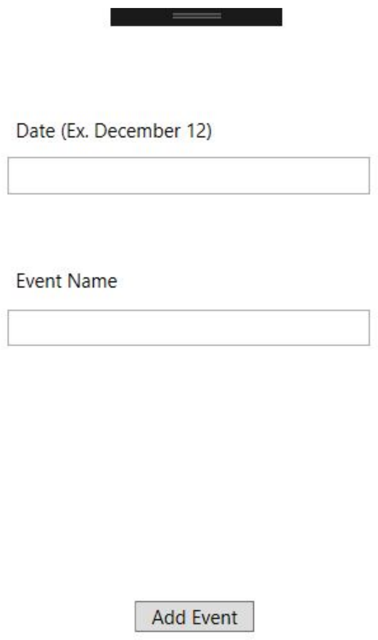

Figure 2. Picture of the event addition interface

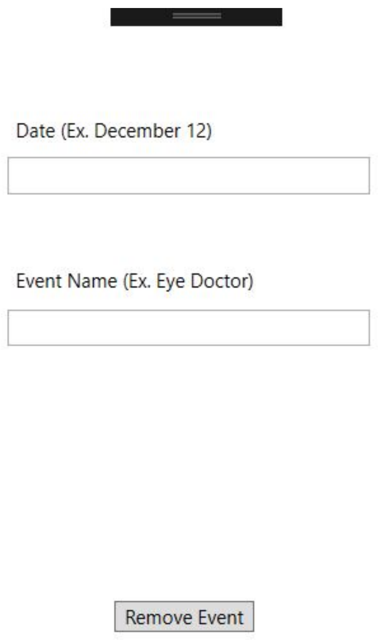

Figure 3. Picture of the event removal interface

- 28 August 2018: Project Start
- 4 September 2018 - 20 September 2018: Gather requirements
- 20 September 2018: Proposal draft due

- 27 September 2018 - 9 October 2018: Group meetings to refine project ideas
- 9 October 2018: Updated proposal draft due
- 11 October 2018: Present project proposal

- 12 November 2018: Project update
- 29 November 2018: Project demo
- 5 December 2018: Final presentation

4. Project Structure

The first design decision was on how to build the calendar. We went with a calendar build on a grid view because this would allow for a much more customizable calendar, which was one of our big goals.

4.1. UML Outline

Figure 4.

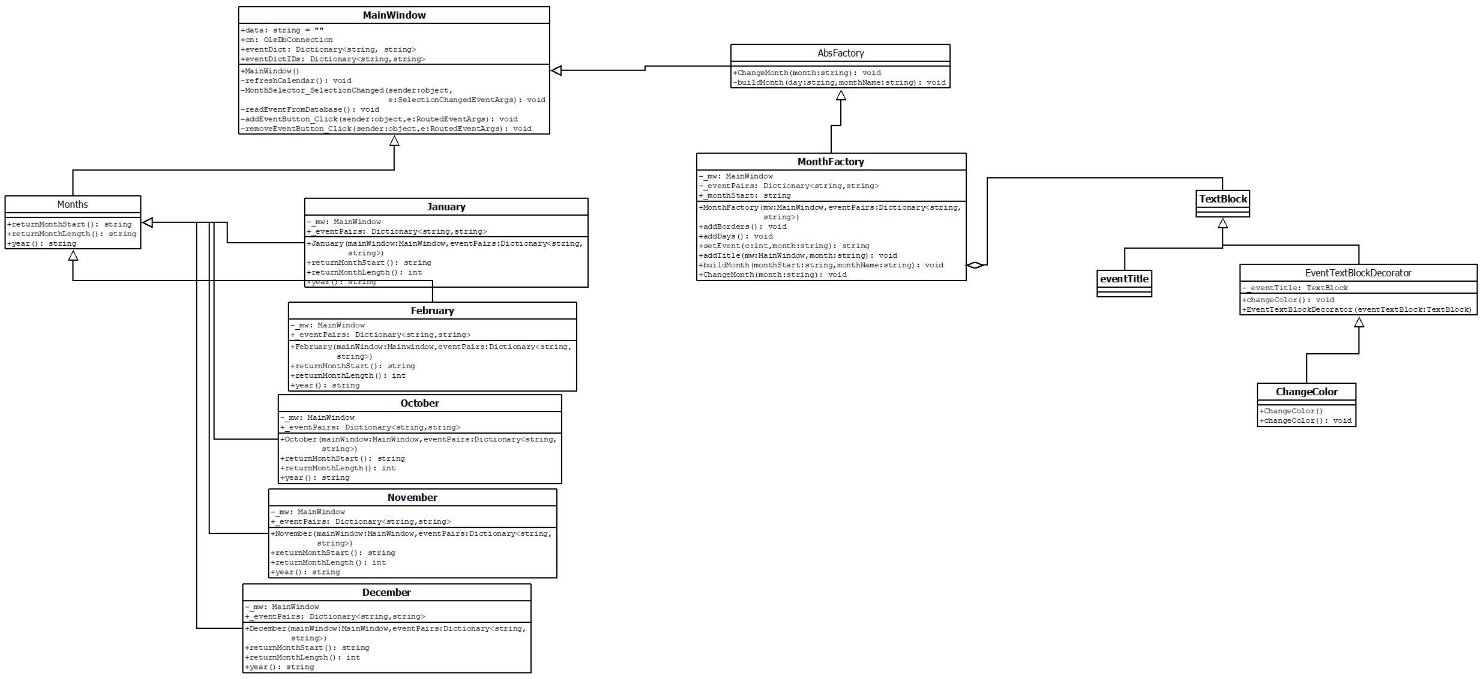
MainWindow
+data: string = ""
+cn: OleDbConnection
+eventDict: Dictionary<string, string>
+eventDictIDs: Dictionary<string,string>
+MainWindow()
-refreshCalendar(): void
-MonthSelector_SelectionChanged(sender:object, e:SelectionChangedEventArgs): void
-readEventFromDatabase(): void
-addEventButton_Click(sender:object,e:RoutedEventArgs): void
-removeEventButton_Click(sender:object,e:RoutedEventArgs): void
AbsFactory
+ChangeMonth(month:string): void
-buildMonth(day:string,monthName:string): void
MonthFactory
-_mw: MainWindow
-_eventPairs: Dictionary<string,string>
+_monthStart: string
+MonthFactory(mw:MainWindow,eventPairs:Dictionary<string, string>)
+addBorders(): void
+addDays(): void
+setEvent(c:int,month:string): string
+addTitle(mw:MainWindow,month:string): void
+buildMonth(monthStart:string,monthName:string): void
+ChangeMonth(month:string): void
TextBlock
eventTitle
EventTextBlockDecorator
-_eventTitle: TextBlock
+changeColor(): void
+EventTextBlockDecorator(eventTextBlock:TextBlock)
ChangeColor
+ChangeColor()
+changeColor(): void
Months
+returnMonthStart(): string
+returnMonthLength(): string
+year(): string
January
-_mw: MainWindow
+_eventPairs: Dictionary<string,string>
+January(mainWindow:MainWindow,eventPairs:Dictionary<string, string>)
+returnMonthStart(): string
+returnMonthLength(): int
+year(): string
February
-_mw: MainWindow
+_eventPairs: Dictionary<string,string>
+February(mainWindow:Mainwindow,eventPairs:Dictionary<string, string>)
+returnMonthStart(): string
+returnMonthLength(): int
+year(): string
October
-_mw: MainWindow
+_eventPairs: Dictionary<string,string>
+October(mainWindow:MainWindow,eventPairs:Dictionary<string, string>)
+returnMonthStart(): string
+returnMonthLength(): int
+year(): string
November
-_mw: MainWindow
+_eventPairs: Dictionary<string,string>
+November(mainWindow:MainWindow,eventPairs:Dictionary<string, string>)
+returnMonthStart(): string
+returnMonthLength(): int
+year(): string
December
-_mw: MainWindow
+_eventPairs: Dictionary<string,string>
+December(mainWindow:MainWindow,eventPairs:Dictionary<string, string>)
+returnMonthStart(): string
+returnMonthLength(): int
+year(): string

Figure 4. UML of the project as it is structured so far

4.2. Design Patterns Used

We used the Factory pattern to make each month in the calendar. We also used the Decorator Pattern to decorate events in the calendar.

5. Results

We were able to get the basic functionality of the calendar working. We were not able to get Windows notifications to work, nor were we able to get the calendar to sync using a Google account. We are still working on the to-do list, which should be done before the final presentation.

5.1. Future Work

To make this app more useful, we would like to let the user login with their Google account to keep their calendars synced across multiple devices. We would also like to add some customization aspects to the calendar such as color scheme, layout, etc.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.