Name TBD

Deliverable 1 Report

Project Architecture

The code for Lightning Calendar is located within a folder titled “calendar” within the comm-central repository. It is a portion of the larger application Thunderbird, which is an email client. The Lightning Calendar is exported to be used by the larger Thunderbird application as a package.

Base

The majority of code is within a subfolder of “calendar” titled “base”. Within “base” are a number of folders as described in the following sections.

Backend

The backend folder contains wrappers for many kinds of data used by the calendar such as dates and recurring events. A factory design pattern is then used to package these together for exporting for use elsewhere in the calendar.

There also exists a subfolder within the backend folder called “libical” which contains a variety of C++ files which are analogous to the other JavaScript files.

Content

The Content Folder contains the main UI elements and behaviors. It defines things like event creation, displays, view management, etc.

It has a fairly standard layout with superclasses for components. Each actual component extends an appropriate superclass.

Modules

The Modules folder consists of multiple utility files that provide helper functions for other classes within the codebase, ranging from conversion between different formats to interactions with the DOM.

Public

The Public folder contains interface files. Classes that are used throughout the rest of the program implement these interfaces. Like in Java, the role of these files is to provide abstraction to the program.

Src

The src folder contains a large number of js files. These files almost all represent the data structure of the app. This includes the Calendar, items in the calendar, alarms, attachments,

attendees and anything else stored in a Calendar or Calendar entry. It also describes the Filter and search implementations as well as a few services useful to managing the data objects. These would be services such as the timezone comparison and week parsing and checking availability locks.

Themes

The themes folder provides all the styling for the application on various operating systems, mostly through CSS files.

Import-Export

The import-export folder of the Thunderbird calendar provides ways to convert calendars from various file types. It allows the user to convert to and from a CSV file and allows users to export a calendar as an HTML file. It also allows users to decide how to export the file as, either as a weekly printout or a monthly calendar grid.

Libical

For a bit of background, Lighting is compatible with iCalendar, which is a calendar specific MIME type. It sends data that look like json objects. Each data object contains the components, which contain properties, values, and parameters. Furthermore, Libical is an open source implementation of iCalendar. As such, the libical folder contains definitions of variables for interfacing with iCalendar.

Lightning

The lightning folder provides a window in the sidebar of the main Thunderbird window labelled “Events”, that allows a user to view any upcoming events at a glance. It mainly consists of XHTML and JS files that comprise the window, as well as React code for loading and state changes.

Itip

Provides email support for the calendar. Called when sending a message inviting users and for any other email uses.

Resources

The resources folder of the calendar app provides some UI elements and a few scripts. Specifically a date-time picker UI element, and a create Calendar component.

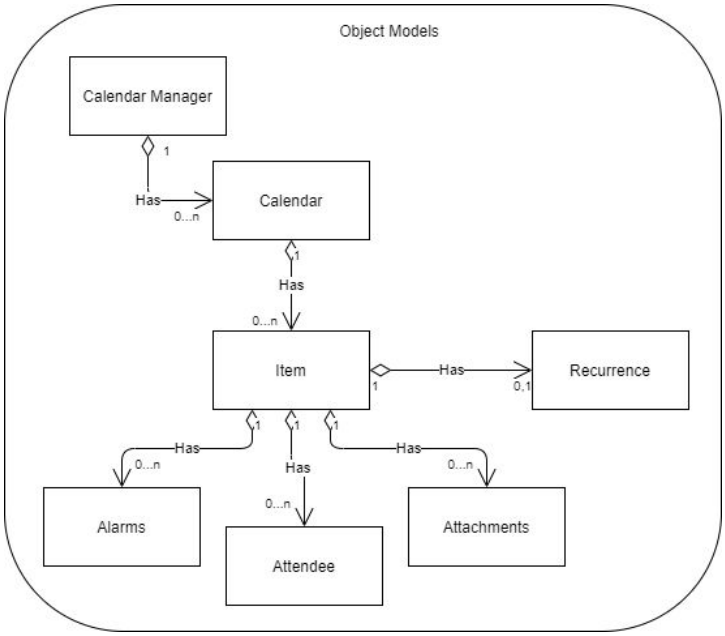
Commentary

Everything is compartmentalized into modules for use by other things. Then everything is bundled together like a package to be used by the rest of the Thunderbird client. This is a good use of encapsulation to allow pieces to be worked on separately and reused if needed.

The create calendar component is noticeably placed outside of the base folder. This may have been due to package loading conflicts.

Many of the lower level code involving prototypes is not very well commented. This has led to some difficulty understanding the purpose of some of the pieces of data. As well, many of the inputs and outputs don't have their types explicitly stated.

Because the calendar is primarily in JavaScript, we felt that UML wasn't a very applicable tool. JavaScript has prototypes which are similar to classes but are not exactly the same. Interfaces and subclasses are not used as extensively as they may be used in an OOP language like Java. This is reflected by our difficulties in finding a reverse engineering tool for JavaScript.



We have a diagram describing the data model of the calendar manager. The calendar manager contains many calendars, each containing any number of events, labelled as items. Each item may have a recurrence rule if it is a recurring item. Each item may have any number of attachments, attendees, or alarms. With each of these is the various data objects such as date and time and duration. It is a relatively simple and intuitive design.

Software Development Process

Chosen Process - Waterfall

For our project we decided to base our development process on the Waterfall methodology. The Waterfall methodology comprises a relatively large initial planning phase. This aims to streamline the development process and avoid surprises and hiccups that may cause unexpected delays or other issues.

We selected the Waterfall methodology because our tasks were not expected to change due to client demands (we have no real client) and the tasks are small enough (they're mostly bug fixes) so it should be easy to make a concrete plan that can be adhered to as part of the Waterfall paradigm.

Other Processes and Reasons for Non-Selection

Extreme Programming

Extreme Programming is an iterative process that provides a fast-paced method of software creation involving a consistent interaction between developers and users, with several versions of an app produced per day. During a working period, user stories are collected and prioritised through a "planning game" with the business, after which the code is created, refactored, and tested daily to ensure a working program at every stage of development.

We declined to choose Extreme Programming for this project as it assumes users are available to provide constant feedback and a business division exists that provides a specific direction for development. There is no business involvement with this project, and users may be slow to provide specific feedback due to the relatively minor importance of the application we are working on, and that it may take a while until a full release is completed that includes our code.

Kanban

The Kanban model calls for a board containing a number of small tasks that team members select for completion. The goal of this is to break down a large task into smaller tasks of different priority levels. This makes it easier to maximize the priority of completed tasks with respect to the amount of time spent working on tasks.

This was not chosen for our project because the scope of the tasks we anticipate completing are too small for us to reap any actual benefits of Kanban. The changes we aim to implement are primarily bug fixes with potentially some small features. In a normal Kanban board, a bug fix is a small enough task to be a single task. It would not be something that could be broken down further.

Spiral

The Spiral model is a planned model that requires a risk assessment of the requirements so that the team knows which tasks will require the most amount of work and the most benefit for the product owners. It is a cycle of stages that goes from determining the objectives of the current cycle, a risk analysis of the chosen tasks, development and planning the next phase of the spiral. In the Spiral model, tasks are planned such that the overall risk is minimized for the team.

We did not choose to use the Spiral model as we are not making new software that carries risk during development nor are we communicating with stakeholders such as product owners to plan the objectives of each spiral phase. The bug fixes we are implementing are merely a modification of existing code, which is not a given phase of the spiral model. We are also not communicating with the stakeholders of the company to gain approval for the finished fixes, nor are we requiring their permission to continue performing bug fixes.