WingIt

Alex Gregory

Nadja Jury

Will Shaw

William Warren

Usability Study

WingIt uses Apple's default styling, which makes it intuitive for users within the Apple 'ecosystem'. WingIt uses an intuitive animated structure when navigating between pages that marry well with a user's intuitive expectations - flipping between pages in a diary or calendar.

Some menu buttons in the current build can cause the app to crash. There is also a case where attempting to navigate beyond the end of the week will cause the app to crash, but only when there is no timetable data available. Unit testing the components in question, as well as examining the state of the various components in XCode whilst reproducing these issues should enable us to get a better insight into both the cause of the issues, and possible remedies.

Informal *Usability Testing* suggested that users should have the option to cancel a login/update, at this point the only way to do this is to fully close and restart the application, then begin a new login process.

Some Users informed us that the interface is indeed intuitive, however, they also noted that the app is lacking features which are necessary to make it useful on a daily/weekly basis; such as the inability to add your own events to the timetable for example. As a group we have discussed this feedback and what deficiencies we are going to prioritise addressing. This will be detailed in the project report section.

Do all planned features exist in the delivered software? Do essential features exist?

We planned to have notifications included in the current build, however this feature was not achieved. We think it an essential part of the application, as the primary idea is to assist in organising a student's life.

We wanted to ensure that our app could handle scheduling conflicts appropriately, as students with busy schedules are likely to have them, including some of the development team.
This was achieved by displaying conflicts to the user in the same time allotment, but with the two events each having one vertical half of the view. This feature has been implemented in the Semester One deliverable and will continue to be maintained.

Would we, the developers, use our own software?

Some harsh criticism is due. While some of the developers do not own an Apple device to start with; in it's current state the app offers minimal functionality, and therefore doesn't provide the incentive to waste the space on your device. Key improvements to change this calculus include implementing notifications, streamlining the timetable importing process and reducing the incidence of crashes.

Progress Report

Does the current software meet the criteria we came up with for assignment 1?

In short, No. Drastic changes had to take place.

One of these changes occurred because the C/C++ library cURL - which we intended to use to connect, login, and retrieve the timetable data from eVision - was unable to meet the needs of the job. This was primarily due to the library not having a JavaScript Engine, where eVision relies heavily on JavaScript to not only log in, but also to retrieve and render timetable data.

Some group members had also not used Git version control before, which led to some strange commits and the loss of a number commit histories.

Self assessment:

- We feel that more of the code could have been, and still can be, written in C++. We aim to increase the workload of the C++ library, by making the library responsible for saving and retrieving data, and also making changes to the underlying timetable events.
- The amount of work achieved for the second assignment wasn't close to what we wanted, this is somewhat because because we had limited experience with Objective-C and Swift, and the Apple development flow (Xcode especially). Now that we've gained experience from this, we should be able to better estimate our ability to implement planned features.
- We also didn't account enough for team members university schedules, and also for sickness affecting overall development time.

Future Improvements

What features are essential for project success?

1. Make the C++ library do more work, saving and restoring the data, and also managing the events, using Swift to make simple function calls on the library to trigger manipulations. Once complete, this data can then be saved/loaded by serialising the collection object to a file using output and input streams.
2. For the timetable to be able to load *n* weeks into the future (*n* specified by the user).
3. Notifications with customisable trigger times, per event (can also be disabled per event).
4. Minimal to no crashing! (Crash testing and exception handling).
5. To be able to edit any event, and have that data persist across a timetable load/update.
6. The ability to add custom events to the timetable, which also persist.

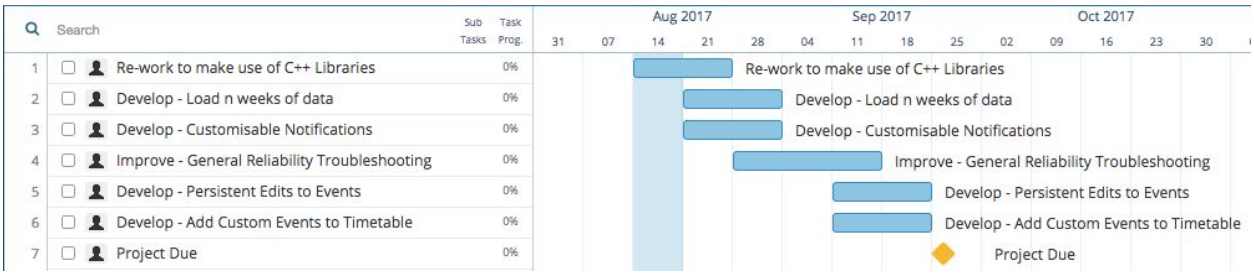
How will we prioritise these updates?

Update	Priority (1 = high, 10 = low)
Make C++ library do more ‘heavy lifting’	**1**
Load n weeks of data at once	**2**
Customisable notifications	**2**
Improve general reliability	**3**
Make persistent edits to events	**4**
Add custom events to timetable	**4**

The ‘essential’ features listed above will be prioritised by the assigned number. Low numbers are higher priority. Those with equal priorities can be implemented in parallel as they don’t depend so much on each other to be completed.

Following the completion of those ‘essential’ features, we can begin adding any ‘extra’ features to be determined at a later date. This will depend on the remaining time available, and also discussion over to what extent those features contribute to the functionality or usability of the app.

Proposed Schedule



(Larger figure available on last page)

Further clarification of features

To reduce project complexity, the C++ code can be made responsible for timetable management.
We will store two sets of timetable events, one being the set pulled from eVision, and the other being a custom set, which contains events added or edited by the user.
The purpose of using two sets is so that the user can update their events from eVision, without overwriting previously customised events.
The C++ library will organise these two collections and filter out any duplicates, then return a single collection of merged events - when requested - to Swift.

Saving and loading of these sets of data will be done by serialising the collections objects to file, and reading in an object representation from said file. This will also remove the dependency on the CSV Importer Cocoa framework, and would modify the way the current parser operates.

Risk Assessment

One current risk is the creation of a ‘fat’ C++ library (fatlib). This library is a combination of the Parser libraries built under different iOS architectures. The creation of this fatlib has proved difficult, as older versions of Xcode used to allow for building these libraries directly from the GUI, however the later versions have removed this functionality, requiring us to investigate using a build phase to then run a custom script which builds this library for us. There are also alternative options being investigated.

Another risk to the project is the upcoming release of iOS11. The team has the intention to modify the codebase as required to support this version. This decision will be reexamined if we come to expect adverse effects on scheduling or implementation of other features.

Other risks remain the same as when noted in the Assignment One report.


Sub Tasks
Task Prog.
Aug 2017
Sep 2017
14
21
28
04
11
18
Re-work to make use of C++ Libraries
0%
Develop - Load n weeks of data
0%
Develop - Customisable Notifications
0%
Improve - General Reliability Troubleshoo...
0%
Develop - Persistent Edits to Events
0%
Develop - Add Custom Events to Timetable
0%
Project Due
0%