COSC 345 – Project Overview

Introduction

In an increasingly fast paced society, we need an easy, reliable, and dynamic way of managing our time. For most people, the use of a calendar application is a good way of efficiently managing their day-to-day (or even week-to-week) schedule. There are many calendar applications on the market, so the process of testing each one - to find the best that suits our needs - can be tedious and long-drawn-out. Often the applications will lack some feature that we want, or have problems that developers have overlooked. There are also issues with calendar applications that are native to our mobile devices, often they aren't dynamic enough or easy to use.

For this reason, we have decided to build a calendar application of our own. With the vision of a simple calendar - that is both easy to use and fits *our* needs - we hope to improve on many existing calendar applications. In addition to this, adding some functionality that is specific to students (particularly the Evision timetable synchronisation). Our aim is to build a calendar application that is crafted around providing service to university students.

What are we building?

The application our team has decided to build is a simple calendar application. The complete applications should have the following functionality:

- View days in a week, on a month level
- View each hour in a day, on a week level
- Set markers on a day, representative of an event or a reminder
- Set markers on hour(s) of a day, also representative of an event or reminder
- Alarms/alerts for high priority events/reminders
- Synchronisation with device time/date to ensure correctness
- Synchronisation with external calendars including Evision, Google calendar, Outlook/Hotmail calendar, and Facebook calendar.

The project is broken down into five modules. Backend, frontend, synchronisation, documentation, and testing. Backend refers to the underlying data structures, methods, and classes that allow the application to function. This is the most important part of the program. Frontend refers to the GUI that will display the data from the backend in a logical and well-structure way for the user to interact with (and to send data to the backend for processing). Synchronisation is about communicating with external API's (Google, Hotmail...etc.) so that we can pull calendar data from them. Documenting is to facilitate the use of the application by developers and users. Testing is about ensuring that all the application functions are working correctly and reliably.

Figure 5 (see Appendix) has a breakdown of these modules and the time set aside for each. The backend of the code should take our group three weeks to get to a working state. The GUI should take a further two weeks for correct interaction between the frontend and backend. Figure 1-4 (see Appendix) show a mock-up of how we expect the application to look**.** The application should take a total of six weeks to complete, which will leave us with 1-2 weeks 'breathing space' for any unforeseen events. Any additional features we decide to implement will not be added to the stable version of the application until adequate testing has been done.

Organisation - People and Roles

Our team consists of four members that all have a combined background in computer science, engineering, business, and mathematics. Table 1 lists our strengths and weakness, and with them we tried to organise members into roles that would benefit the team.

Table 1: Shows members strengths and weaknesses

Person	**Strengths**	**Weaknesses**
Kurt Weston	GUI design, understanding the system requirements, understanding interactions between components whether it's small sub-processes, or the system as a whole, planning and assistance in development.	Programming with memory managed languages. Personal time management. Testing.
Will Sanson	Testing, time management, software design models, communicating ideas.	Limited knowledge in developing mobile applications, as well as programming in c/c++.
Paul Souter	Backend programming with memory managed languages, understanding and building algorithms and efficiency of programs.	Writing reports, visualizing component interactions on a big picture level.
Ryan Swanepoel	Strong visualization of final products, understanding how things should work together, high 'last minute' efficiency.	Planning, impulsive decision making

Allocation of project work:

- Kurt: Management of tasks, algorithm development, assistance in front and backend
- Will: Largely testing and assistance in frontend GUI development and backend
- Paul: Implementation of the backend systems and interfacing with frontend
- Ryan: General assistance in front and backend with focus on data structures and optimisation. Further focus on interactions between frontend and backend.

Because we are all at a similar skill level, and have the same commitments outside and inside university, we can ensure that each person is making a somewhat similar contribution to the application in terms of time. Weekly meetings will ensure that task deadlines are met. This is a general guideline of the work, and in practice we should see large amounts of cooperative development or assistance rather than independent work.

Resource Requirements

The greatest resource requirement is time. This involves being available to continuously work on the application, whether it be attending meetings or implementing the application. Because we are not spending the entirety of our time on this one project, time management will be key to the success of it. Many of the resources required for building this project are either free or already owned. Some resources such as power and internet add no significant cost, as we are working at such small scale. The following table lists the main resources needed to complete this project.

Table 2: Shows the resources needed for the project

Item	**Description**
Computers (Hardware)	Personal and laboratory computers are already owned and/or freely available
Visual Studio	IDE we will use, free for students
Github	VCS that is free for open source projects
Travis-ci.org	For testing our project. Open source (com is not open source/free)
SmartSheet	Application for collaboration and work management - 30 day free trial
CMake	For building the software (Open source)
Lumzy	Mockup and Prototyping tool
Power	No/very little impact on costs (We do not make a large contribution to power bills of our own or that of the University)
Internet	Unlimited data (We add to the University internet bill, not our own)

Risk Analysis

Table 3: Shows the risks associated with our project development.

Possible Risk	**Solution**
We can't say with certainty how long this project will take. Due to our limited experience with software development, our estimates on project scheduling has a large margin of error. With little experience working in a team, poor task allocation can diminish efficiency of software development.	Our time allocation will be aided by the use of a gantt chart, in which we must allow for unforeseen ‘speed bumps’ in the development of the application.
Lack of experience in C/C++ will disrupt workflow, as we are learning as we code. We do not know how to complete tasks until we have finished them. Learning as you go diminishes work efficiency.	We will reduce the amount of learning required in this project by using libraries. This will aid us in transferring our knowledge of C into C++, without ‘re-inventing the wheel’.
If code is written by one person and the rest of the group has no understanding of how it works. If something were to happen to that person - then the group would have to waste time trying to work out how that piece of code works.	We will be sharing all the knowledge as we submit code to a GitHub repository. Any code that is not intuitive must be rewritten or if there is no other option then comment the difficult code.
Change in requirements. This may arise from ambitious ideas or change of management.	To handle these problems we may have to scale-down our milestones, such that the overall project does not differ from the original goal of having an easy-to-use planner.
Failure of third-party tools	To minimize the chance of third-party tool failures we will only use stable builds, and libraries/frameworks that are from respected developers.

How to ensure quality

A calendar application's sole purpose is planning and time management. As such, this should be a very streamlined process. One important attribute to ensure quality is usability. The application itself should not be more complicated than it needs to be. Furthermore, the application will require some user interface intuition in its use. To ensure quality of this application, we will need a fair amount of testing amongst ourselves. We will also be placing the application on the Google Play store, in the hopes of receiving feedback and ideas to maximise the usability and user interface intuition. As a general guideline to maximise usability and intuitive design is to follow Jakob Nielsen's 10 usability heuristics (see Appendix, Table 1).

The backend will require the most testing. This will be testing without the use of a GUI - for speed and efficiency - until we are confident that the backend is reliable and working as it should. Then we will test the GUI frontend and backend interaction, and identify any new bugs. Any additional test cases should be first tested without GUI, and then with the GUI once it is working as intended. To help with testing we will use free Unit Testing Frameworks that are included with Visual Studio, as well as external testing frameworks.

Our application will need documentation, both for the source code, and for the instructions on how the users can interact with our product. Source code documentation should help anyone reading the code to understand anything that is not clear (though all code should be clear to anyone who understands basic programming). Instructions should be to help users interact with the application incase they are unfamiliar with similar applications, and to help users make the most of the features.

Finally, we will make use of GitHub's distributed version control so that we can all work on our different tasks, and use GitHub as a central repository. We will use Travis CI to check our code when it is pushed to the repository to make sure any new additions will not cause the project's test cases to fail.

Appendix

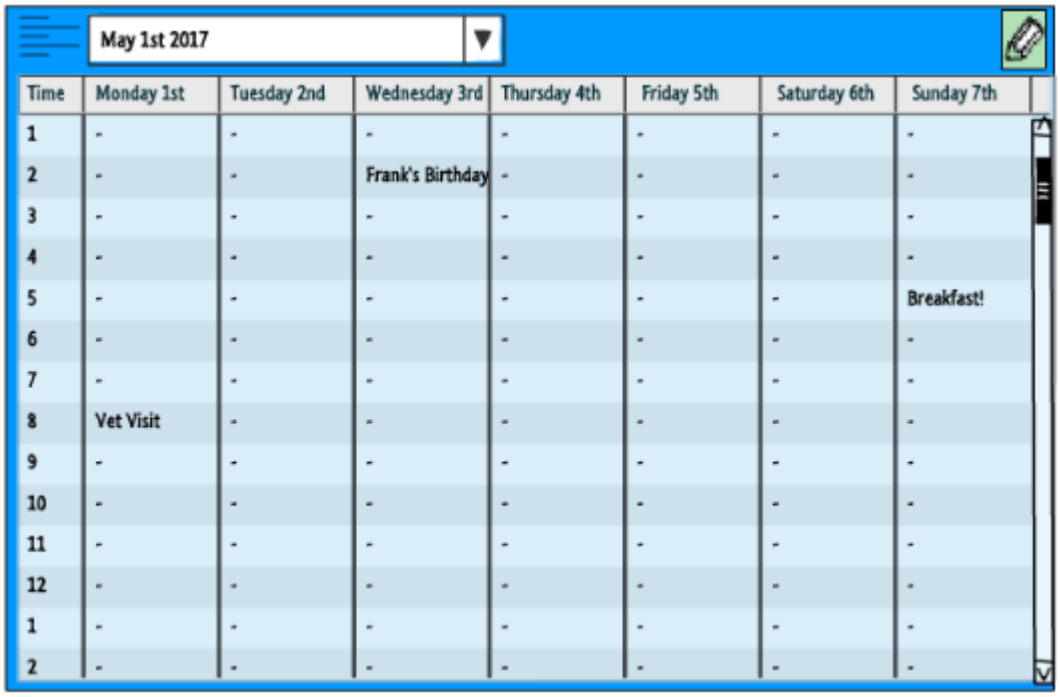

Figure 1 : Week View

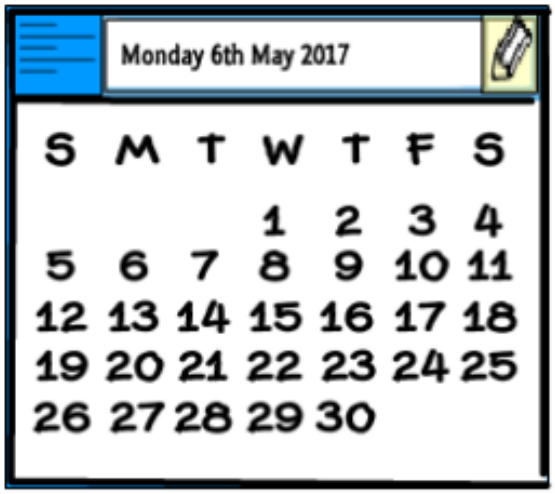

Figure 2 : Week View

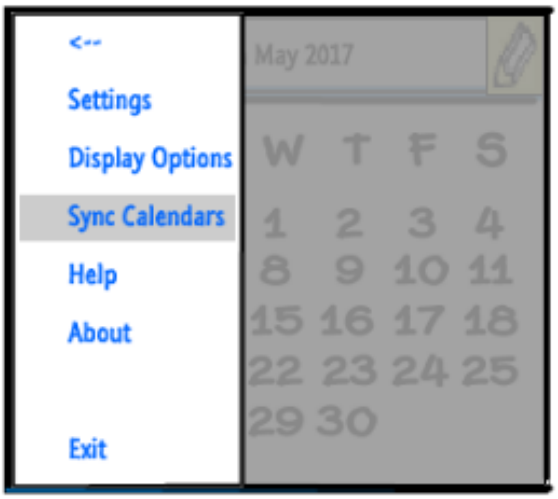

Figure 3 : Week View

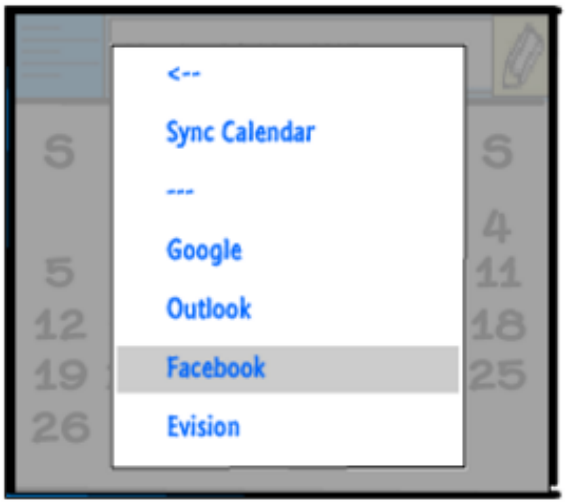

Figure 4 : Week View

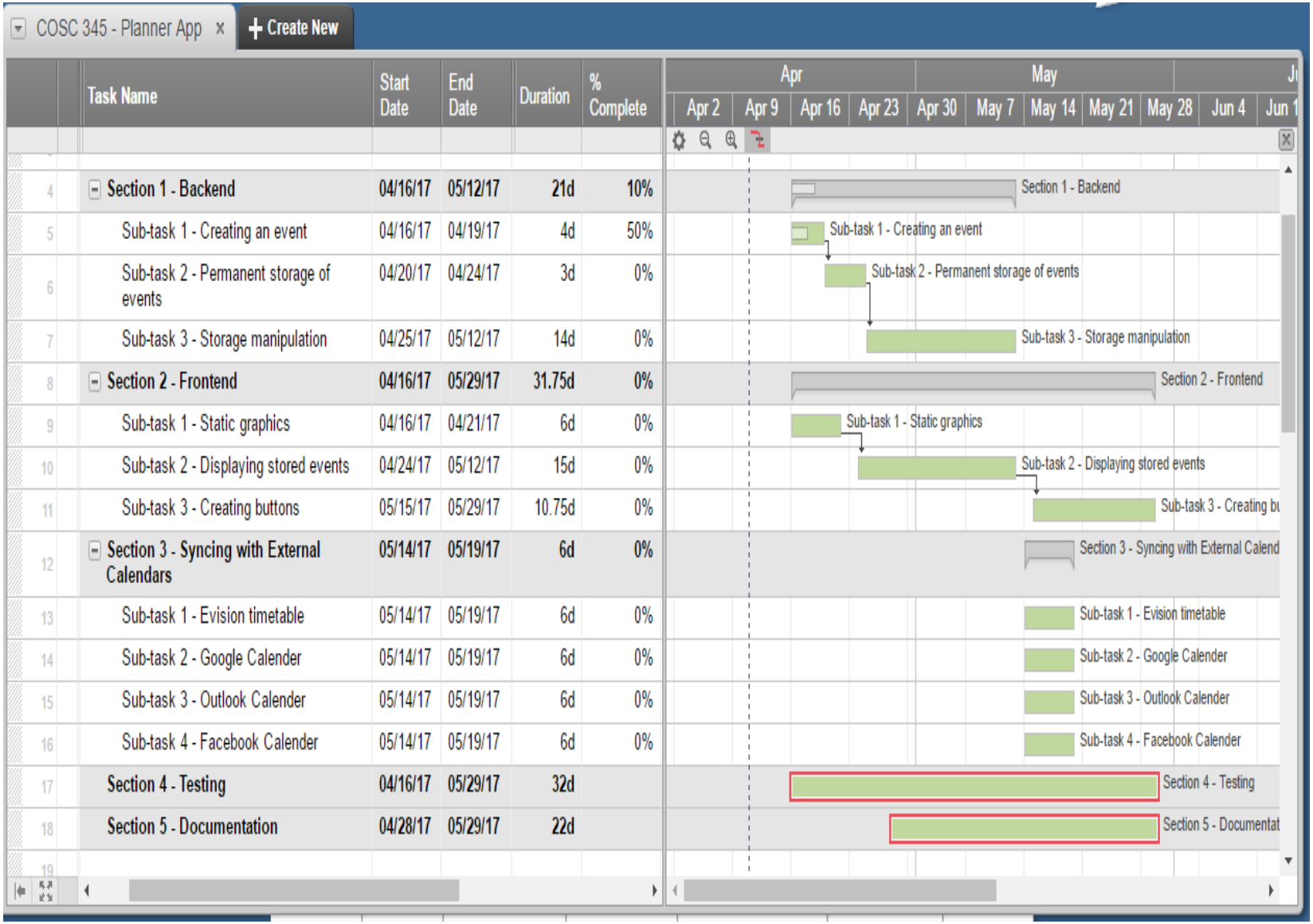

Figure 5: Gantt chart of project schedule

Table 1: Jakob Nielsen's 10 Usability Heuristics

Heuristic	**Description**
Visibility of system status	The system should keep the user updated on the current status of actions, or processes occurring in application. (eg: pulling from google calendar should display a icon and message indicating that it is working)
Match between system and the real world	The application should be intuitive in that buttons and markers should represent a real life equivalent. For example, edit button is a pencil.
User control and freedom	Make sure the user is aware of actions that have extensive results (and don't unintentionally administer it). For this application it may not apply as much.
Consistency and standards	Should follow platform conventions. Different menu's should have consistent layouts.
Error prevention	Careful design ensuring errors don't occur, or states that are error-prone are carefully handled.
Recognition rather than recall	Minimize user's memory load by making objects or actions recognisable. (for example, edit button should be recognised as edit, or "+" should be recognised as an action to add an event).
Flexibility and efficiency of use	Frequent actions are able to be made as fast as possible (efficiency). Should cater to experienced and inexperienced users (flexible).
Aesthetic and minimalist design	The application should implement a minimalistic design to reduce clutter, additionally should look nice.
Help users recognize, diagnose, and recover from errors	If users encounter a problem in the application, give readable (non-technical) feedback on the problem. Give solutions if possible.
Help and documentation	As a last resort, the user should be able to seek help. Ideally from inside the application. Mainly clarification on how to use it.
Reference: Nielsen, J. (1994b). Enhancing the explanatory power of usability heuristics. Proc. ACM CHI'94 Conf. (Boston, MA, April 24-28), 152-158.	