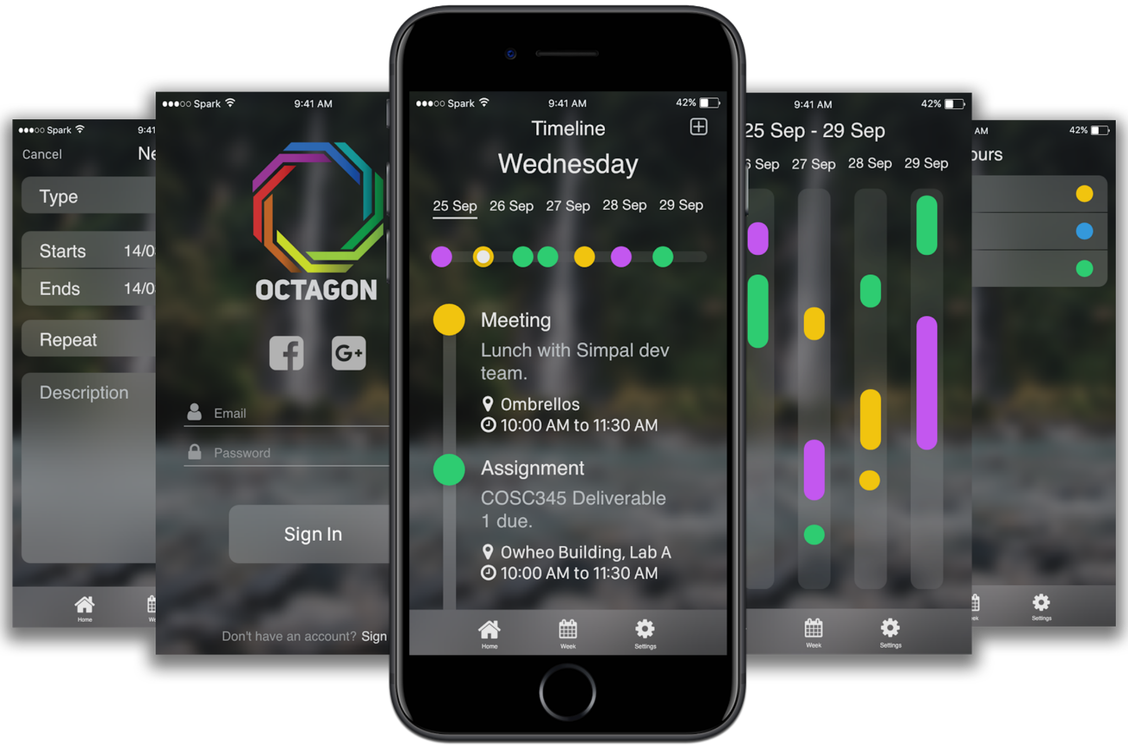
OCTAGON

BE ORGANISED



WHAT

Life is hard. People are busy. You've got to be here before there, but can't forget to grab that on the way. Things are complicated. Meet Octagon; simple and minimalistic. Octagon tells you where you need to be and when. It's a scheduling app whose goal is simple: display quick, relevant information at a glance. Overview your day. Octagon, shows you what you've got on and when. Tell it as little as or as much as you like. You're in control.

Currently, app stores are inundated with products that offer to organise your day. These apps offer huge customization but detract from the core concept; simplify. Octagon is different, it makes life easier with lightweight, intuitive design, seamless navigation and easy to manage events. It provides all the information you need and no more. We asked ourselves, what can we take away from these other apps and still have a scheduling app? Octagon is that answer. But it's more. It's a promise. Spend less time organising your organisation app, and spend more time organised.

Our primary focus is to display relevant information to users in a format that lets them absorb it at a glance. Day view should tell them what's on for the day, when and where. Tap an event and find out more information. Week view should tell them what's on for each day during a week. Tap a day and go to day view.

We will know we've succeeded when a user who has never used our app can pick it up and instantly know what they can do and how. Navigation should be easy and consistent. UX should be familiar. Recognition over recall. Creating an event should be quick and painless, and the user should decide how much information they want to provide. We should have as few mandatory fields as possible. Notifications should be intelligent. The system should remind users about events. An intimate relationship between the system and the user should be developed.

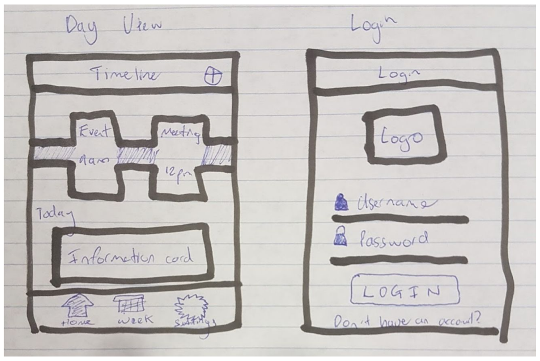
We believe that we can create the best scheduling app. No longer will double booking, missing a meeting or submitting an assignment late be an issue. Be organised.

DESIGN

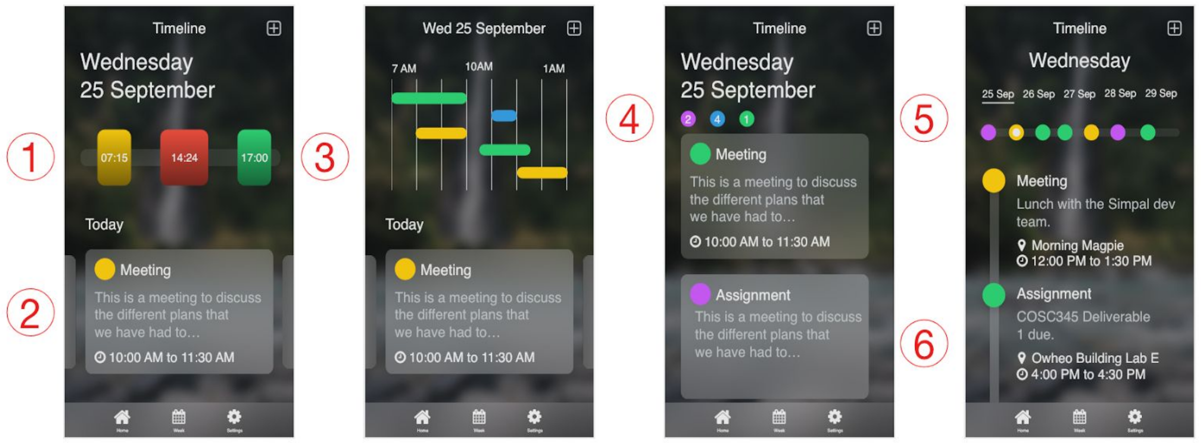
Design is an important part of any system. For any system to be useful, it needs to be usable. A large part of this is design. Our group took a task based design approach to designing our app. In other words, whenever we were thinking about adding a feature to our system, we would identify what task we were trying to complete and then base our design around that This approach ensured that we were always designing to meet a need and not designing for the sake of it. It allowed us to cut down on unnecessarily abstraction and stick to what is really important to our system.

We designed with the knowledge that the best design should always become invisible to the user; they should simply know how to use our system and should never have to think about what they are trying to achieve. Anytime we discussed adding a feature we asked ourselves a series of questions. Do we actually need it? Can we have a scheduling app without it? Is it intuitive? Does a user know what something will do before they do it? Does it reflect the real world? Will the user understand the feature without explanation? Is it simple? Can we say more with less? If the answer to these questions was "yes", we'd ask one final question: can we do it better? If the answer was "no", we knew that we had an important feature to add.

Like any good design, our concept took many iterations to get to where it is now. We began by creating a series of paper mockups, for each of the main tasks that we wanted our system to accomplish. Once we had finished creating our mockups we looked at some different digital design tools. We decided that Adobe Experience Design (XD) was the most suitable as it was free, worked on both macOS and Windows, and included a design prototyping feature that allowed us to make our prototypes interactive and test them on our phones. Using XD we were able to transfer our low fidelity, paper based mockups, to higher fidelity, digital mockups.



This allowed us to test our design and using the prototype feature on our own phones, we were able to get a feel for how our design really worked. With relatively low time cost involved, the ability to physically test the system helped find and address any issues that our design had. Ultimately, this allowed us to refine our design over several iterations. For example, the picture below shows 4 different iterations of Octagon's day view design.

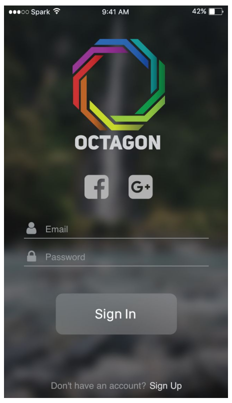


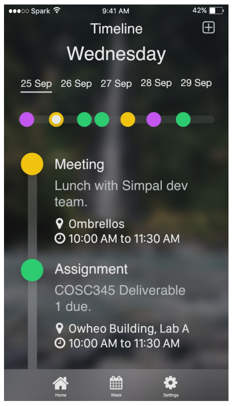
Our primary goal was to clearly and succinctly display all the information about which events a user has on in a day. Label 1 was our first attempt at designing this. Unfortunately, the size of the events on the timeline were too big and meant that having more than four events in a day caused the timeline to become overcrowded and unusable. Label 2 was our first attempt at displaying the "more information" view of an event. We started with a card based system in mind. As well as displaying the individual event information, the card system showed an edge of the next card so users knew whether they had another event that they could swipe horizontally to view it. One important thing that we learnt from the first iteration was that colour coding the events helped users identify the relationship between the timeline overview and the card based, more information view.

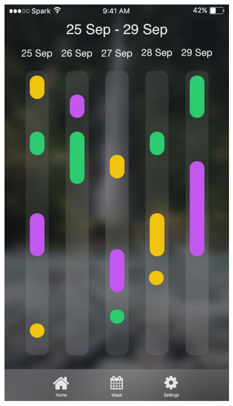
In our second iteration, at Label 3, we focused most of our improvements on the timeline overview. Since we knew that it was likely users would want to have more than four things on per day, we made the event bubbles on the timeline a lot smaller.

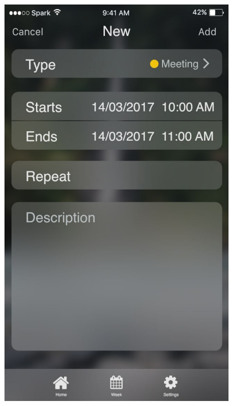
Although this was a good start, in iteration three, at Label 4, we realized that if we were happy to minimize the size of the event bubbles, then there was no reason why we couldn't just give a summary of the day's events. We also modified our card list to scroll vertically instead of horizontally, since we had claimed extra screen space. Even though this was true, we still really liked the idea of being able to look and get a sense of when events were happening in a day and what order they were scheduled in.

Consequently, in iteration four, at Label 5, we reverted back to a horizontal timeline system, but kept the idea of small event bubbles on the timeline. For the first time we also made a considerable change to our "more information view", moving away from the card based system. We felt that a linked list type view helped to give users a sense of time passing. We also added a horizontal list of the next five days, with a line underneath to indicate which date the user is looking at. These dates act as navigation and can be tapped to change the day displayed. We also made some more subtle design decisions including adding a hard limit on the amount of text allowed in the description and adding an indicator to the timeline overview to show which event a user is currently looking at.









WHO

- - - -

Charlie Rawstorn

Designer and Developer

Charlie's focus will be ensuring that the system is built to a high standard and documentation is clear, succinct and complete.

github.com/rawturtle
charlie.rawstorn@otago.ac.nz

Oliver Reid

Designer and Developer

Oliver's focus will be on ensuring that the system is usable and intuitive and that the system is built in a way that is aesthetically pleasing.

github.com/beardo01
oliver.reid@otago.ac.nz

Thomas Youngson

Designer and Developer

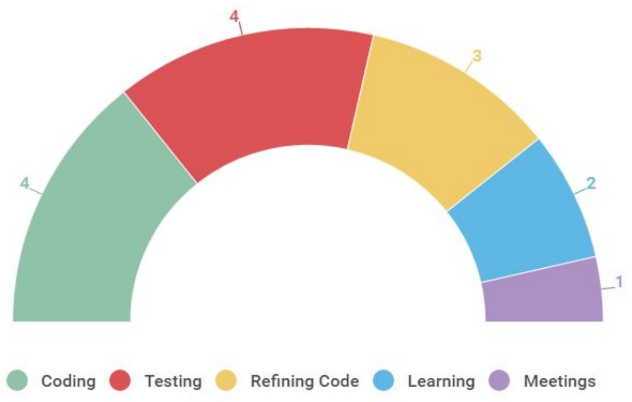
Thomas' focus will be on ensuring that both the system and documentation follow consistent design standards, as well as contributing to the code base.

github.com/thomasyoungson
thomas_youngson@hotmail.com

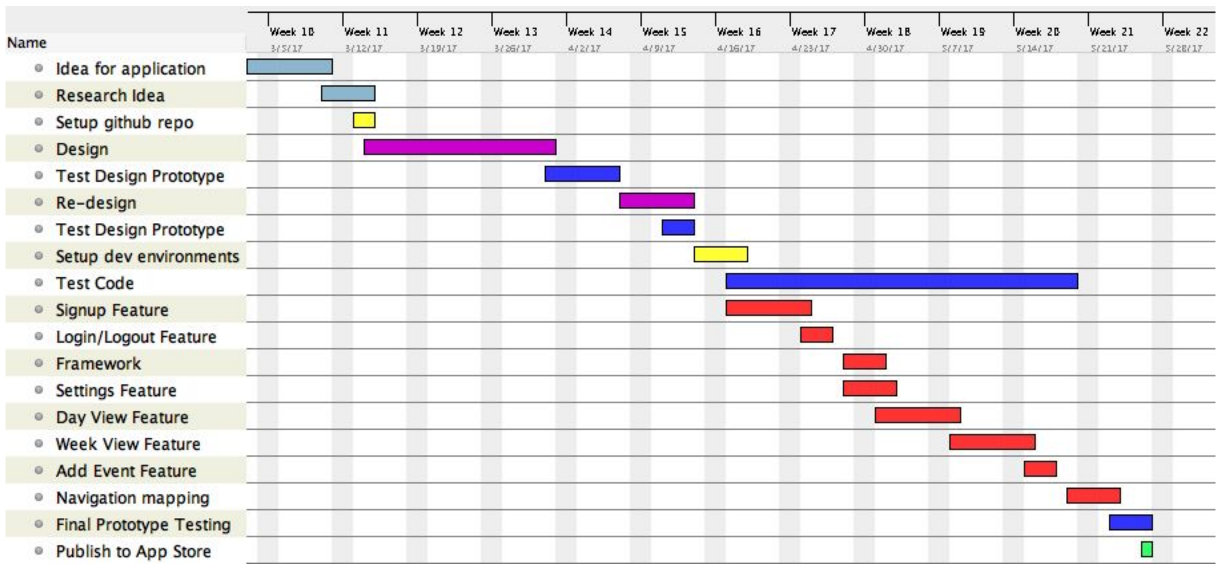
TIMELINE

- - - -

Our goal is to have a working prototype within 13 weeks. We have set aside 6 weeks for planning and design and 7 weeks for coding and testing. Planning and design is important as it means that once we reach the development stage, we have a clear goal of what we are creating. On the other hand, coding and testing is important because it covers the actual implementation of our system.



We have designated 14 hours, each, per week to work on our system during the coding and testing phase. We plan to spend around 4 hours coding, 4 hours testing, 3 hours refining code and an additional three hours learning and meeting. 14 hours per person, per week over 7 weeks equates to 294 hours total, which we think is a reasonable estimate of how long our app will take to develop. Since we know that people are generally pretty bad at planning how long things are going to take, we intend on having a meeting at the start of each week, to see where everyone is at and get a general feel for how the project is progressing. This is an agile approach to development.



HOW

- - - -

Our project is broken up into three main parts: frontend (the part that the user sees and interacts with), backend (the part that manages computation and talks to the database; data storage and retrieval) and storage (where and how the data is stored).

In terms of frontend, we elected to use a frontend framework called Ionic. We chose this because Ionic is completely open source and allows the frontend of mobile apps to be written in CSS, HTML5 and Sass. This means that our team doesn't have to learn any new frontend languages like Swift, which helps to reduce development time and cost. Ionic is built on top of AngularJS and Apache Cordova. Cordova works by using CSS and HTML for the rendering and Javascript for the logic. It can also be extended through Javascript function calls, to call native system plugins or plugins that we can write ourselves. Specifically, it can call and talk to either Swift or Objective-C. Since we want to focus on C++ for the backend part of our assignment we will write a Cordova plugin which will call an Objective-C function which will wrap and pass the call onto our backend C++.

All of our programs classes will be written in C++ and C++ will be responsible for the programs computation (like finding an available time to add an event) and talking to the database to retrieve and store data. To do so, C++ will use ODB which is an open source, cross platform and cross database object-relational mapping system. Object Relation Mapping systems or ORM's allow our C++ code to persist C++ objects to a relational database without having to deal with tables, columns or SQL and without having to manually write any mapping code. This helps to save a lot of development time because we aren't needing to worry about database specifics.

In terms of our database, we have elected to use PostgreSQL hosted on Google Cloud Platform. We chose PostgreSQL as it offers the advantages of being open source and having a strong community backing. Google Cloud Platform was an obvious choice as it offers reliable, scalable storage that let's us focus less on what is going on behind the scenes and more on the code itself. The pricing is also dynamic and works on a pay as you go basis. Although this means that there is a little bit of uncertainty about what the cost of the storage server will be each month, Google lets us set up price alerts and tailor our storage instance to meet pricing budgets. Through negotiation, Google also offered us $100 credit to get us started which helps us reduce our cost. Using Google's "Google Cloud Platform Pricing Calculator", we were able to estimate our monthly cost for storage at about $10.

Other than the code itself, we also contracted a logo designer from Fiverr (https://fiverr.com) to design our app logo. This cost us $10. We also use Google's "Google Maps API" to provide suggestions when a user is entering a location for an event. Google offers a free 25,000 API calls per day with a cost of 50 cents per 1000 calls over 25,000. Initially, we don't expect to exceed 25,000 Google Map API calls, so don't expect to pay anything for Google's service. To be eligible to develop and release iOS apps, Apple charges developers an annual 99 USD registration fee. We will open one developer account with Apple so expect to pay $99.

Indirect costs that the team have considered include the cost of hardware, like laptops, that the team requires to develop our app and our time that we will put into developing the app. Although these aren't direct costs that will show up on a bill, they are still important to consider.

QUALITY ASSURANCE

- - - -

To ensure that our program works as intended, testing is important. We will write unit tests for each part of our backend C++ code. Unit testing allows us to test each component of our system and isolate it from the rest of our system. When an error occurs, this helps to determine what exactly is responsible for creating the error. We will also write unit tests for the Ionic (frontend) part of our system. Jasmine is a popular testing framework which allows developers to write tests for Angular apps running Ionic. Jasmine will allow us to check that functions are being called and doing what we expect. Jasmine also allows us to mock function dependencies so we know where the issue really lies.

In addition to testing, we will also use code coverage tools like CoverageMeter. Coverage tools allow us to check which parts of our program are being executed when a particular test or set of components is ran. Again, this will allow us to identify, track down and fix any issues in our system. Finally, to ensure that our unit tests are run constantly we will use continuous integration tools like Travis CI. Travis CI runs our system's tests every time we commit to GitHub. This means that we will quickly discover if our latest commit broke anything, and give us the opportunity to fix it before it becomes a problem.

Ionic and XCode also allow us to emulate our app on an iOS device or through the web browser. This will mean that we can constantly build and run our app to ensure that it is behaving as expected. We will be able to guarantee the quality of our app by constantly carrying out this process ourselves, running the unit tests and getting feedback from friends who will be able to test the app. Getting help from friends, in the form of testing will be beneficial in two regards. Firstly, they might find issues that we wouldn't since we wrote the system. Being the developers and testers of an app often results in functional fixedness. Secondly, it will be beneficial because they will be able to give feedback on other aspects of our system that things like unit tests cannot. Other than functionality, they might consider issues like usability, design and learnability.

WHY

- - - -

Life without Octagon is hard. You either have several different apps or use part of a much larger, complicated system. Organizing your life becomes a chore. You have to organize organizing. Octagon changes that. People will use Octagon because it's easy. It does one thing and it does it well. Want to know when an assignment is due? Instead of going to a website, logging in, navigating to the page you want, then scrolling through a PDF, add it once to Octagon. Want to know what you've got on this week? Instead of loading up a slow calendar application that requires way too much information and is way too complex, open Octagon. Want to find a time slot next week to schedule a meeting? Open Octagon. It's all there. Quick information; at a glance.

Life's simple with Octagon. Why wouldn't you use it?