Ryan Mcgoff (4086944), Jerry Kumar (3821971), Jaydin Mcmullan (9702973)

What we learned from assignment 1

In assignment 1 our lack of a concise time management plan was Andrew's greatest concern for our project. It's easy to be overly ambitious by promising a list of features and then failing to deliver on them. But with our University workloads differing weekly, and each having different work schedules we needed a more flexible time management plan then simply a daily to-do list. This lead to us creating our own solution on Google docs.

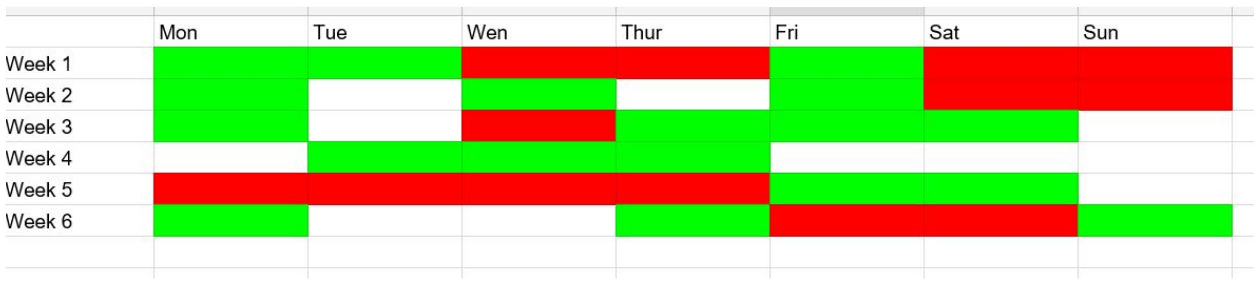
Planning

On the Friday of each week we would create a plan for the following week. We worked out we 6 weeks until our alpha product was due, so we split each task into categories and allocated an estimated difficulty/time to them. Each week we would strive to get at least 15 points. This way we would get everything completed by week 5 at the latest and have an extra week to clean up the code and produce the documentation.

Main Classes
FixedTask Editor (10)
FixedActivity (10)
FlexiTask Editor (10)
FlexiTask Activity (10)
Database (10)

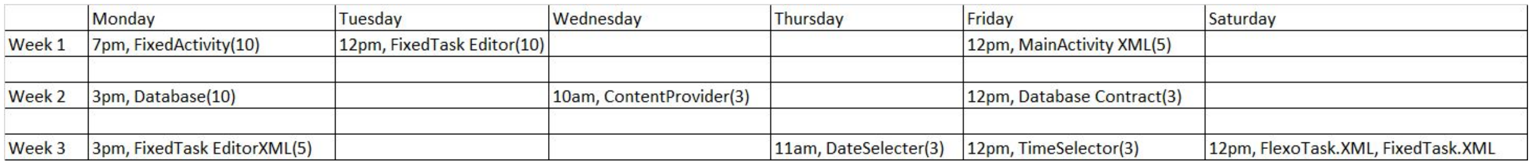
Design
FlexiTask XML (5)
FixedTask XML (5)
MainActivity XML (5)
FlexiTask Editor XML (5)
FixedTask Editor XML (5)

Other
Database Contract (3)
DateSelecter (3)
TimeSelector (3)
ContentProvider (3)
Commenting Code (3)



	Mon	Tue	Wen	Thur	Fri	Sat	Sun
Week 1	🟩	🟩	🟥	🟥	🟩	🟥	🟥
Week 2	🟩		🟩		🟩	🟥	🟥
Week 3	🟩		🟥	🟩	🟩	🟩	
Week 4		🟩	🟩	🟩			
Week 5	🟥	🟥	🟥	🟥	🟩	🟩	
Week 6	🟩			🟩	🟥	🟥	🟩

We had a shared GoogleSheets where we could indicate the days we were free, if a single person wasn't free on a particular day, we would highlight those days red and choose at least three days we were all free and willing to meet up during that week. After the days had been selected we planned which parts of the app we were going to work for that week. This was of course flexible, but we did find ourselves sticking to the schedule for the most part.



	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Week 1	7pm, FixedActivity(10)	12pm, FixedTask Editor(10)			12pm, MainActivity XML(5)	
Week 2	3pm, Database(10)		10am, ContentProvider(3)		12pm, Database Contract(3)	
Week 3	3pm, FixedTask EditorXML(5)			11am, DateSelecter(3)	12pm, TimeSelector(3)	12pm, FlexoTask.XML, FixedTask.XML

We had finished everything two weeks early than expected (Week 3), so we used the additional time to fix bugs, and implement parts of our beta like fragments and a tab switcher. We also got our sorting algorithm correctly working for our flexi task and added general visual polish to each XML file. This ended up being harder than expected as we had written everything as its own separate activity and not every method was supported by the Fragment java class. This meant that Switching to fragments broke everything, and so we had to rethink a lot of our code (see problems we faced for more details). We also had no experience with SQL, so spent a few days learning the basics syntax for the language. This showed us the importance of allocating extra time to fix unforeseen problems.

Team Communication

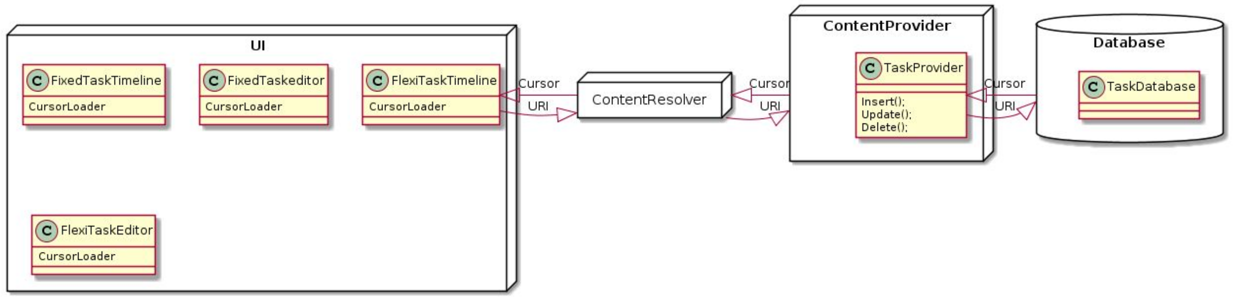
This was paramount, at first we tried to work on the project individually but quickly found this to be incredibly difficult seeing as we each had drastically different skill levels when it came to android development. We opted from a team-based solution, where we met each week at least three times, and spent a few hours working on the project together. This solution worked great, as we each had a say in the creation process, and the more experienced team member could explain some of the trickier android development coding and design concepts.

Version control

Towards the end of the project, as we became more comfortable using Android Studio, we began using GitHub's version control as a form of cloud storage for our work. But because Android provides an excellent local version control and much of work we did was done offline, we didn't use GitHub often. It did, however come in handy when we lost the charger for the main laptop we were using and needed to access our files on a temporary device. We also tried a variety of different repository sites including BitBucket and SourceForge.

On the night before the assignment was due we couldn't push our work to GitHub - we believe this was due our lack of understanding with how branching worked as well as having two Github accounts linked to Git. After hours of frustrated googling we were forced to delete our Git folder via the command line on GitBash and initialize a new one. This issue almost definitely could have been solved with a better understanding of Git. So, before we begin the beta stages of our project we will be completing basic GIT tutorials so this won't happen again.

How the app works



Our mainactivity (timeLineFragmentsContainer) is a viewpager containing the fragments for Fixed Timeline and Flexi Timeline, this allows the user to swipe between each fragment. When these fragment classes are first created (onCreate() method is called) each class initializes its own loadermanager. Once this loadermanager is up and running, it calls the OnCreateLoader() method , which initializes a new CursorLoader. This cursor loader is given a projection (a string array of the columns the activity/fragment want), the URI (like a URL but for database tables) and a sortOrder (Flexitask wants the data returned to be sorted by its priority calculation, whereas fixed task it wants the data ordered by due date) . This cursorloader in turn queries the contentresolver.
The contentresolver "resolves" the URI to our custom content Provider (Task Provider). This provider acts as an interface to the database, it's here that determines whether we are inserting, updating deleting or just querying data - and whether we want a specific row or the entire table. It performs the desired task and returns a Cursor with the data requested, back to the contentResolver, which returns back to the LoaderManager which passes it on to the cursorOnFinish() method. In this method we call the cursorAdapter swapCursor() method and pass it this newly returned cursor to process.

By using the loader thread and abstracting this data retrieval process to loaders and content providers, we allow the user to continue using the UI with few interruptions. We also use a similar querying technique in our editor activities, in addition to a simple if() statement that allows us to reuse the editor activities for both creation of a new task and updating an existing task.

We had to account for when the user “updates” the task from either the flexi or fixed timeline fragment in the form of the “done” button. When selecting “done” (the tick), for a fixed task, the app will check if that task is recurring or not. If it is it will update the tasks due date to (the current due date + recurring period * 86400000 (milliseconds in a day)) to get the new due date. It then restarts the cursor loader to go and retrieve these new changes.

When a flexi task is first created the date-last-completed field for that row/task is set to the current date (in milliseconds) and the next due date(in milliseconds) is derived from this by adding the recurring period * milliseconds in a day. When the user indicates they have finished an activity (by pressing the done button/ tick), the date-last-completed field is set to the current time and the next due date is recalculated.Our priority rating is calculated at two stages, one in when we query the data in the Flexitask fragment loader and want the return data sorted by priority and the other in the cursoradapter where we assign a XML color element to indicate the priority to the user.

This is the algorithm for the priority:

$$\frac{(\text{Todays Date} - \text{Date Created})/86{,}400{,}000 + 1}{\text{Recurring Frequency}}$$

This produces a number which determine how overdue a task is. We then organise the flexi tasks based on which tasks have the larger number, as well as assign priority colors.

The floating action button you see in our code was taken from https://github.com/Clans/FloatingActionButton . We plan on as implementing our own for the beta as we found letting the user create either a flexi or fixed task in other fragment didn’t make much sense. Clicking the floating action button should instead take you straight to the editor activity that corresponds to the fragment/page you are on - this is a change we will make in the beta.

Testing

Because our app uses timestamps to create and organize each task, we opted for a manual solution for most of the testing. Throughout our code you may see remnants of logstatments (androids equivalent of print statement), these can be accessed when running the app through android studio’s emulator. By using this method, we were able to catch and address bugs that may not have been immediately visible to the user, for instance our priority algorithm was chopping off the decimal place value because we weren't casting our long as a double - causing incorrect sorting.

We also tested (although not as extensively as we would have liked) how our UI would respond to accessibility changes, including text size changes. One of our group members suffers from minor visual impairment and was able to test this part of the app. They said they were happy with how the text adjusted to their phones settings and said it didn't obscure any major components of our app (aside from the title text covering the due date, but this is something we plan on addressing in the future).

Current issues in alpha:

- "Task selected" message not disappearing when user exits from an editor (FIXED)
- Due date for FixedTask does not include the time, so tasks occuring on the same day won't be ordered (ie: task 11/02/18 @11am vs 11/02/18 @12pm)
- Custom "recurring days" selector layout not formatted correctly
- Title of task overlaps with date due when long enough
- When the user indicates they have done/finished the task, we don't use the contentResolver to update the task and instead access our database directly via a raw query. This is temporary solution as we were having issues interfacing with our database via our content provider and haven't figured out why yet

Problems we found and fixed

One of the biggest challenges we encountered was combining fixed and flexi into one timeline like we planned. This proved to be impossible as both were measuring time and sorting in different ways, fixed was sorted by due date whereas flexi was sorted by how overdue it was. It made more sense to split these tasks into two editors and two timelines. We decided that to display two pieces of information we would use a ViewPager(lets the user swipe between two or more tabs of information), this meant we had to convert our existing activities extending appcompat to Fragments extending the fragment class. This broke a lot of our functionality as Appcompat supported different system methods than Fragments, but also forced us to learn the ins and outs of system methods we had been using to see what would still work and what needed a new solution.

Disability Support

As we developed out app we made sure to keep mind the goals we had to support visual impairment. We ran into some issues with implementing some of our original ideas but have plans to address these all moving forward:

- Content descriptors: The content descriptor feature which allows screen readers to properly relay information is a big goal for us to get working in our app, but we have run into issues with getting our task titles into the content descriptor fields. This is because task titles change according to the user input. While this is a current problem in the alpha, we hope to have found a solution by the beta. We have however made progress with implementing these in the buttons and text fields.

- Focusable containers: Our plan to implement focusable containers has also hit some snags as finding proper ways to group items is a bit more difficult than we initially thought. Again, while this is a problem now in our alpha, we hope to have a working solution by the beta.

- Contrasting according to WWW standards: We found out that android has inbuilt contrasting features already implemented in the settings that allow modification to a lower contrast. This is for people with reading abilities such as dyslexia for whom high luminance colors are not readable. Since android can change this for a user, we have decided to choose high contrast colors for our apps to cater for people with a low contrast sensitivity (which you lose as you get older). This means that people with low contrast sensitivity (who need high contrast) can use our app easily and those who need low contrast can also use our app easily by utilising the low contrast option in the android settings menu.

- Large touch targets: Android also offer magnification settings to essentially allow larger touch targets on request. As such we have made the decision to keep our touch targets “normal” relative to non-visually impaired users, as those who need larger targets can easily change their settings to achieve this.

We noticed as we attempted to implement visual impairment support that android already does this very well, offering solutions such as contrast and magnification options. As such, we have also decided to implement support for another disability: memory loss.

Individuals with memory loss are the ones for whom our reminder app will be the most use, especially with the flexi-tasks allowing them to set up tasks that will better suit their lifestyle. On top of this, we are planning to implement:

- A history section that will allow users to view their past activities as well as keep track of the dates and frequency that they completed them. At the moment with our sql implementation we have the columns in place to support this, we simply need to make code that logs the dates that tasks are completed as stores them in the right column. This will also allow us to easily get the frequency of completion for each task. We will also need to create a new history activity for users to view this information and we hope the get this feature working by the beta.

- A notification system that will alert the user when they are not in the app about upcoming tasks that need to be completed. For users with memory loss, they may forget to check in with the app regularly and may miss important tasks, but with a notification system they will not need to worry about this, as important tasks will show up on their home notification screen. We will also allow users to set the frequency of notifications they want to receive as well as which tasks they want to be notified for (for example, only the high priority fixed tasks).

How to build our app

1) Download and install the latest version of Android Studio with the default settings (https://developer.android.com/studio/)
2) Download our project from Github
3) Unzip the file
4) Import our project into Android Studio (File->Import) or select the "open an existing Android Studio project" from the startup menu
5) Then browse to the Flex-task-project-master folder, make sure this folder shows an Android symbol
6) Once it's loaded, press the "RUN APP" button and select Pixel API < 22 as the virtual device (the device we used for testing) and click ok
7) If promoted, choose proceed without instant run

NOTE: You can test the priority algorithm program by looking at logcat while running the app on an emulator (the log message is created in the CursorAdaptor file). And change the phones date in the phone's system settings to simulate time progressions.

Where to from here

We met every single alpha milestone we had set out. This leaves us with plenty of time over the holidays and half of second semester to implement more features. Notifications were something we were thinking about implementing before we started the alpha, but due to time constraints it was dropped from the plan. Our accessibility options at the moment are rather poor, but this is largely due to most of features we wish to implement requiring a settings menu, a part of android development we are not yet familiar with. The aforementioned issues we ran into with version control /git hopefully be less likely to happen once we spend more time this mid semester break getting comfortable using it. Overall, we are all very happy with the progress that was made and feel confident we will be able to deliver on everything we outlined at the start of our project plus more.