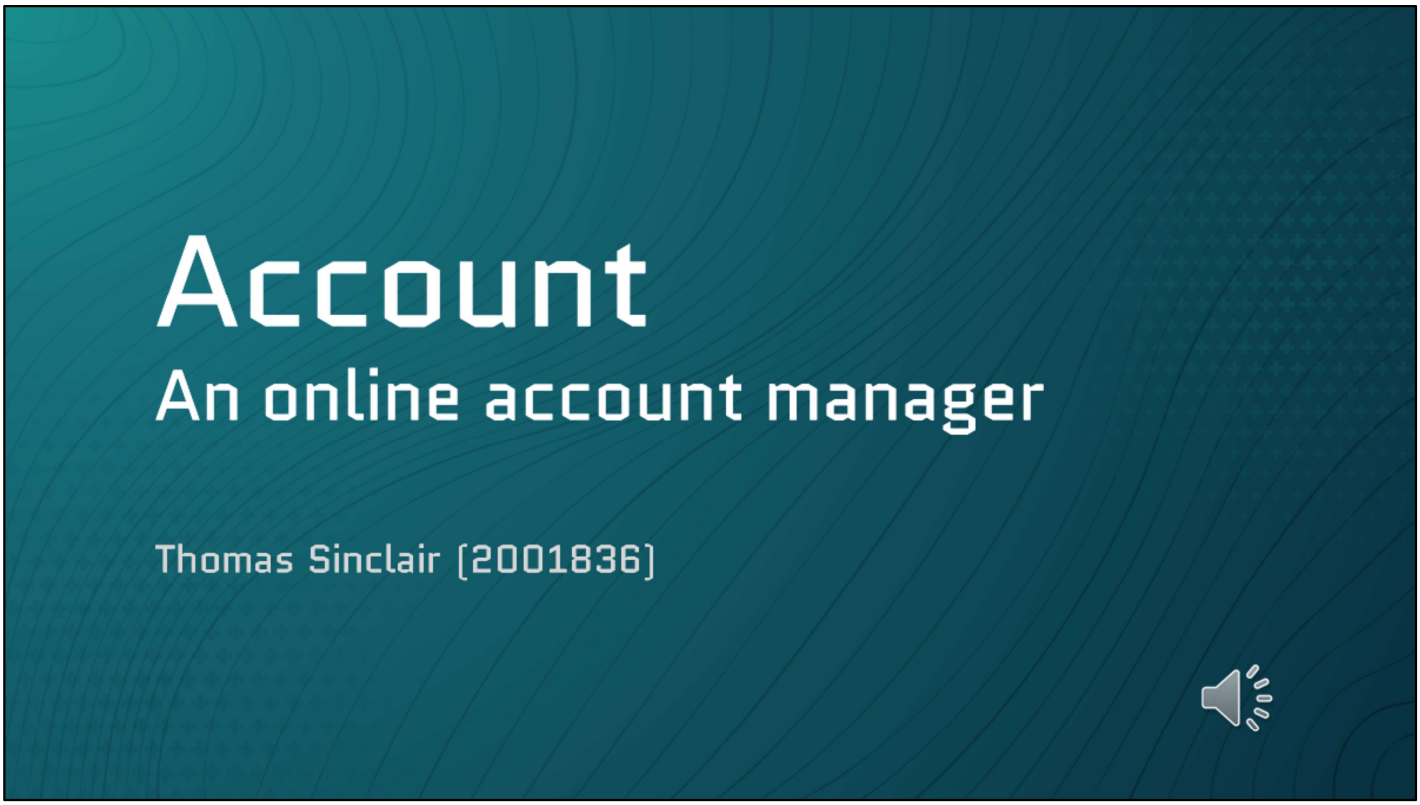
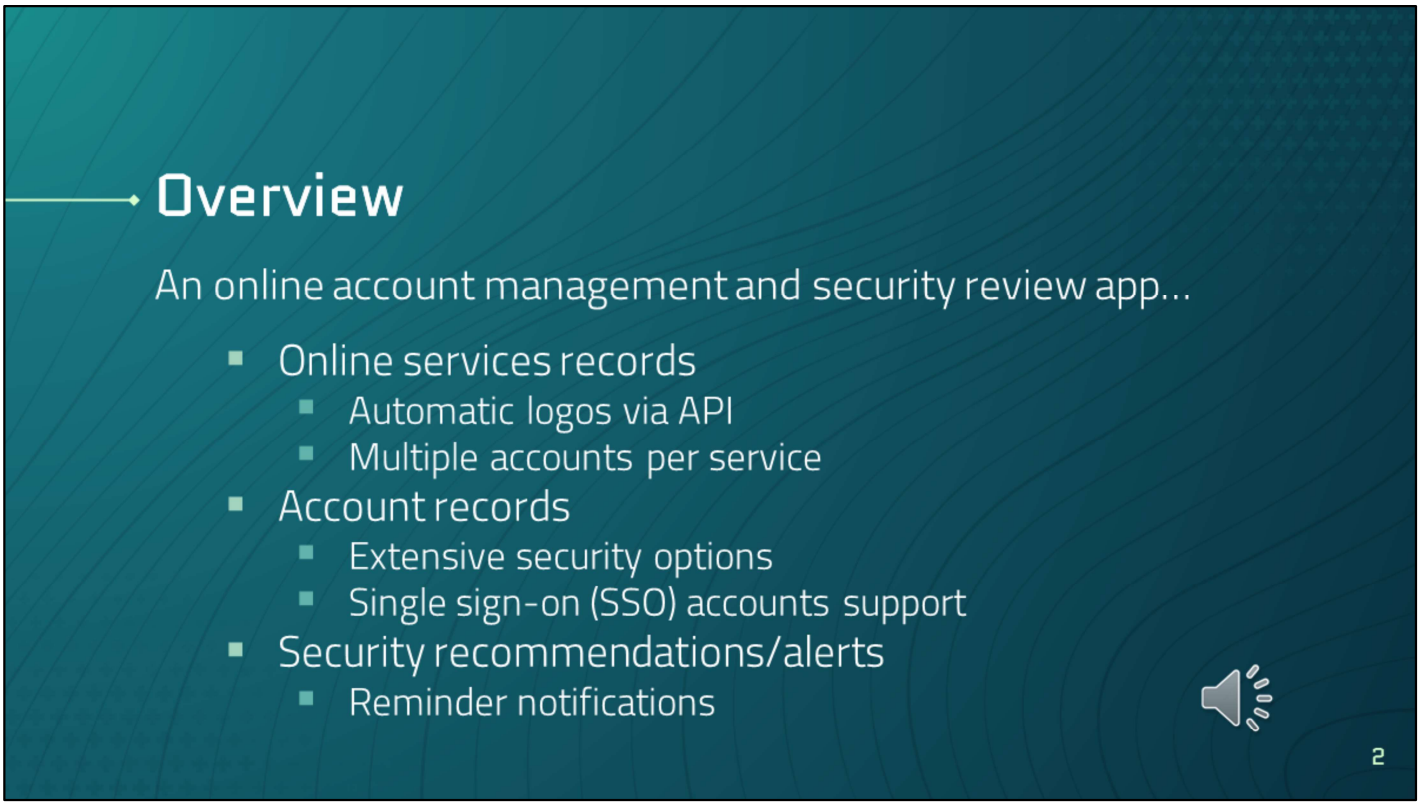


Hello and welcome to this presentation on my Android app project for module CMP309.

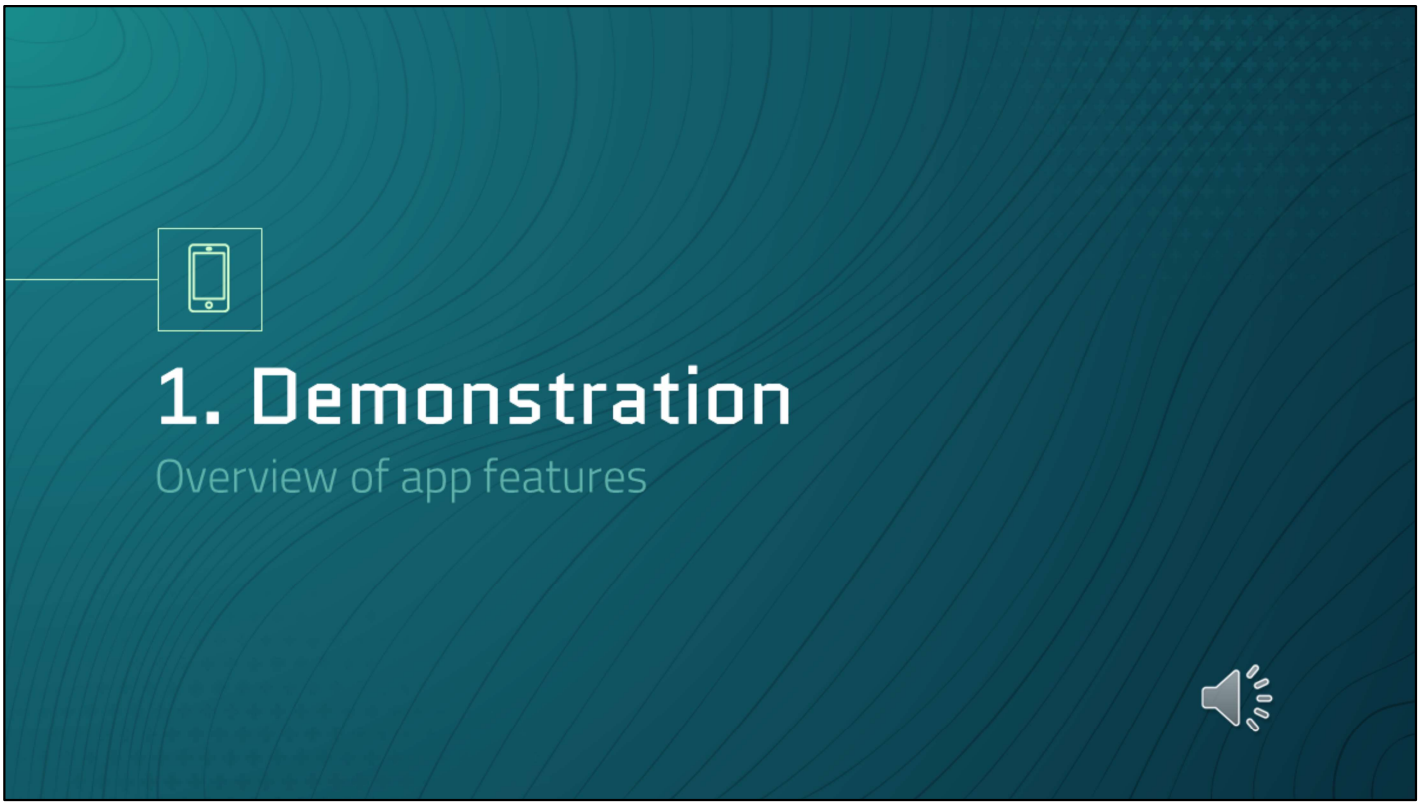
The app I have made is an online account manager called “Account”.



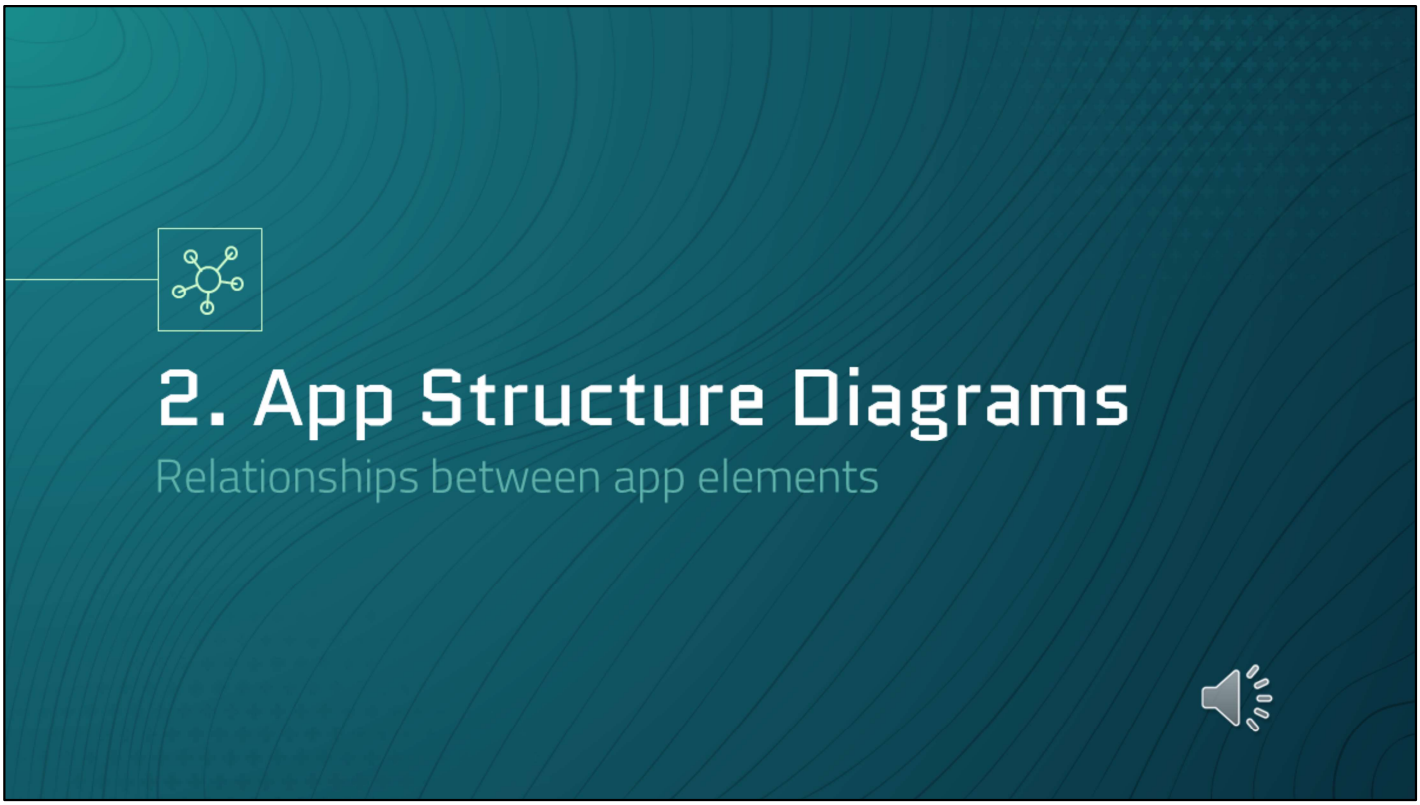
The app acts as an online account management and security review tool. Users input basic information about the online services they use (name and domain), then add the accounts they use with those.

This does not store passwords, only the login method (their email, phone, or single sign-on linked account), and whether the account has particular security features enabled.

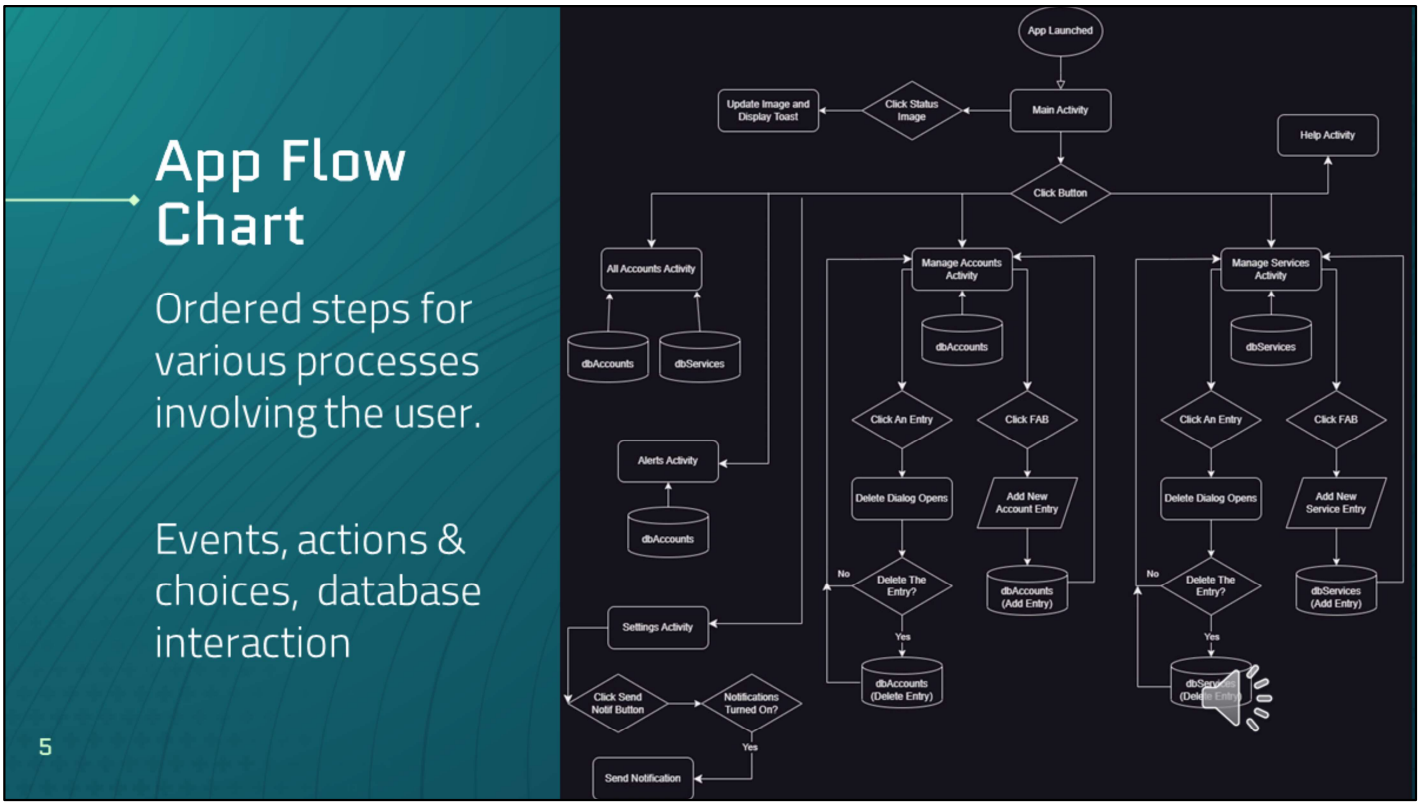
Based on what security methods the person uses with their accounts, relevant security recommendations (alerts) are displayed, with weekly reminders to review any outstanding alerts if necessary.



The following video is a recorded demonstration of the app being used, showcasing and describing its main features.



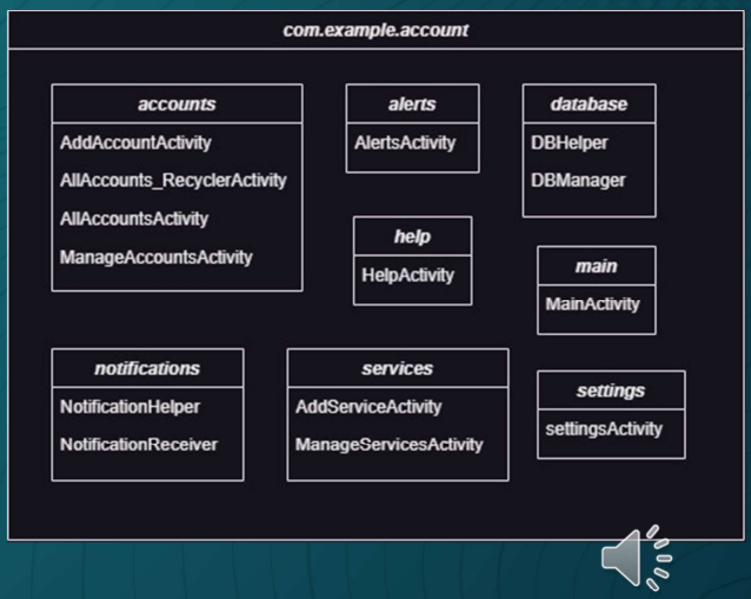
Now, a look at the app structure diagrams…



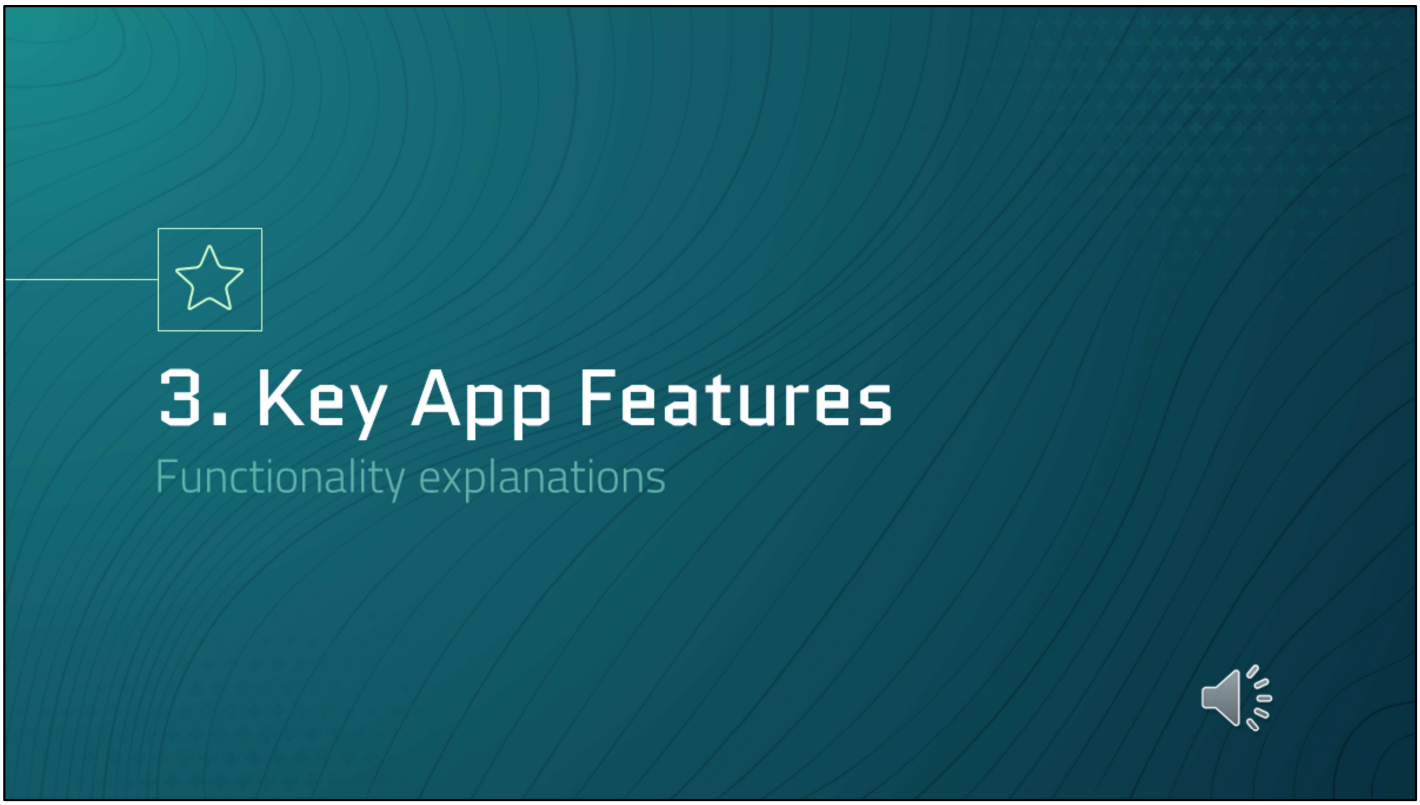
This is a flowchart of the user experience of the app, showing the steps in various processes – for example the path for how to add a new account entry to that database. Events are shown in rectangles, actions and choices shown in diamonds, and database interactions represented by the cylinders.

App Package Structure

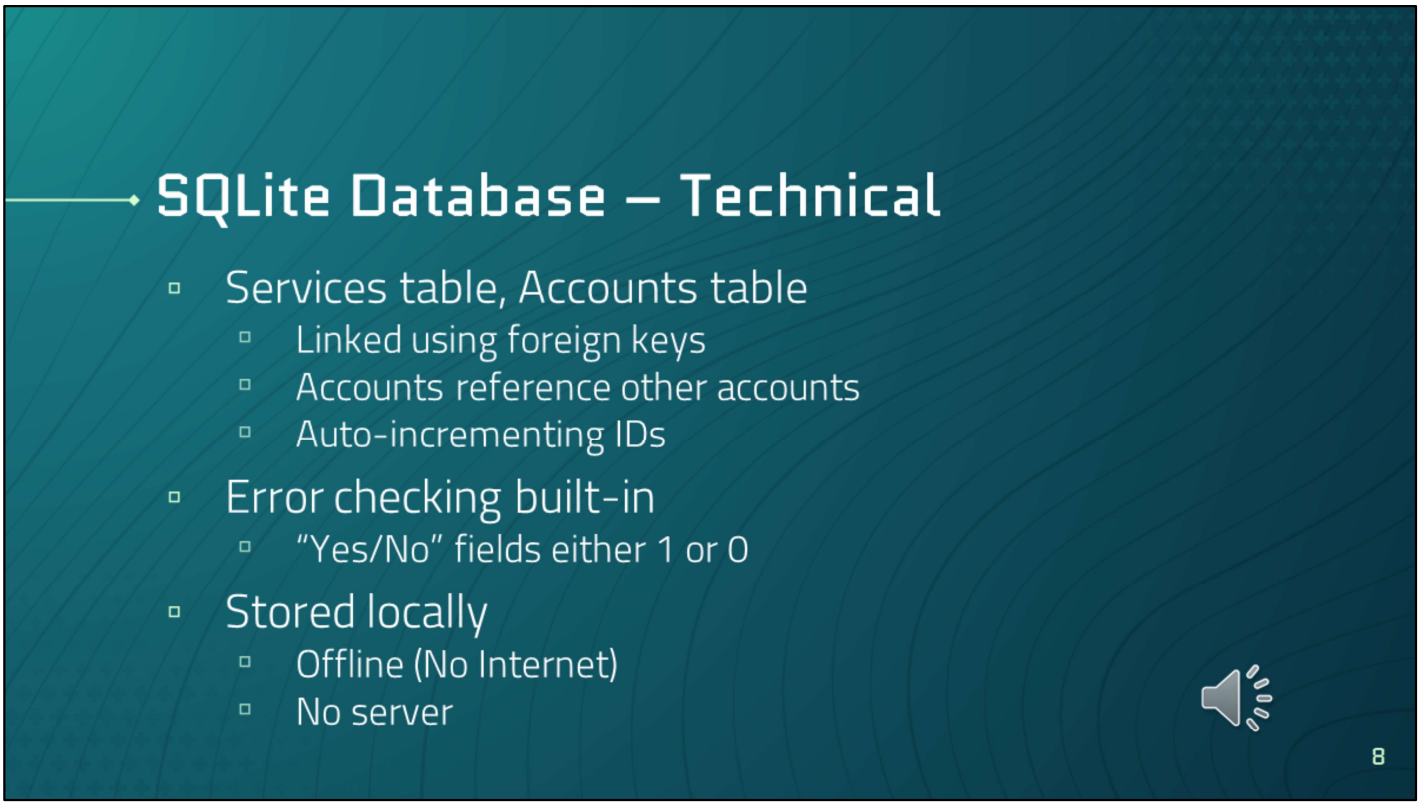
The directory structure of the application, with the Java classes they contain



This is the app package structure, which shows what filesystem directories the activities and other Java class files are stored in.



Next, a look at the key technical and functional aspects of the app



The app uses an SQL Lite database with 2 tables for the Services and Accounts respectively. These are linked on the service ID field using foreign keys, with the optional Single Sign-On field referencing other account IDs to provide support for that.

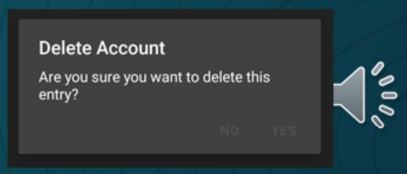
The database has been designed with error checking built-in, utilising “NOT NULL” to ensure certain fields are required, “REFERENCES” to ensure linked table fields are valid, and “CHECK” to implement a yes/no system allowing 1 or 0 only.

This is stored locally, requiring no Internet connection and not relying on an external server, which could cause issues otherwise.

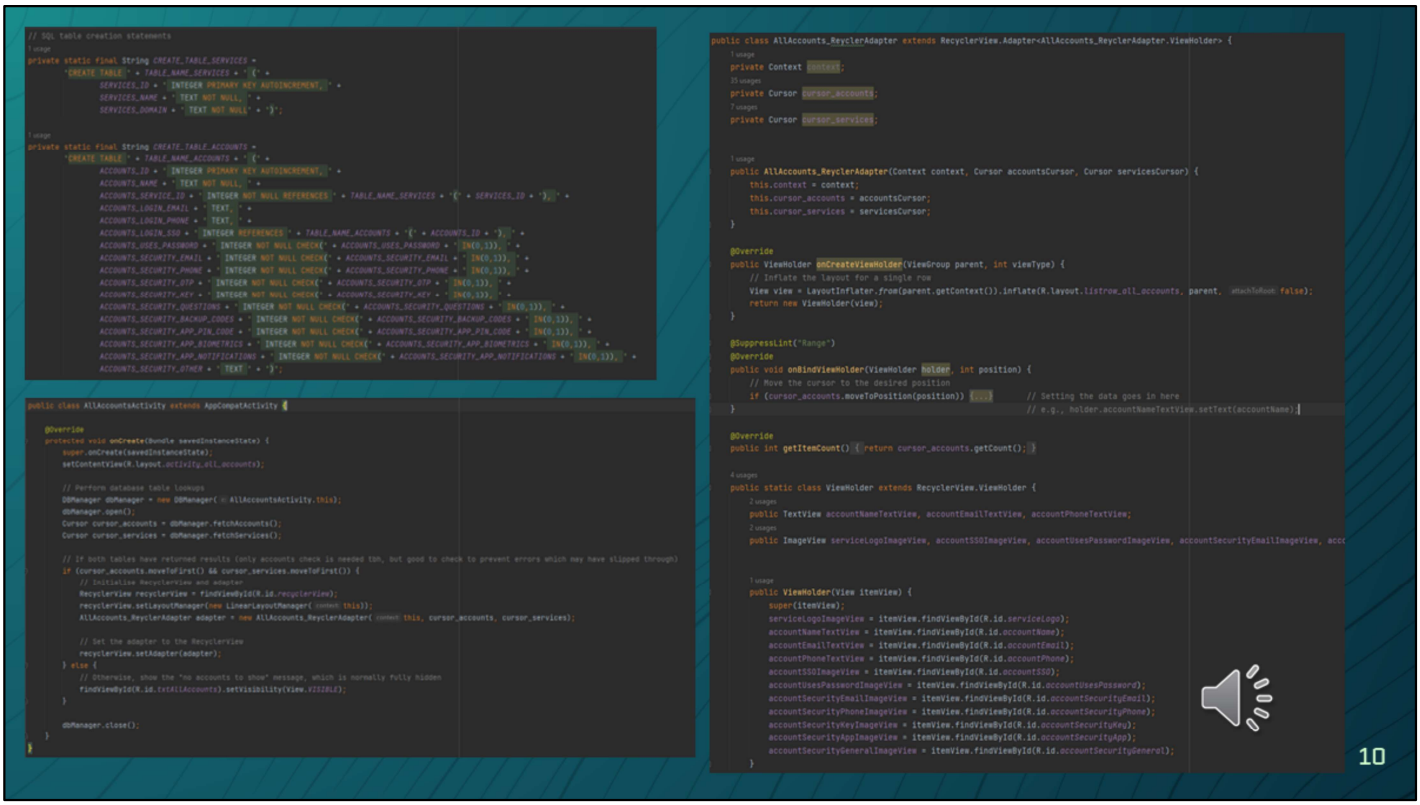
SQLite Database – User

- Recycler Views used to display in scrollable lists
- Pop-up Dialogs used to delete database records

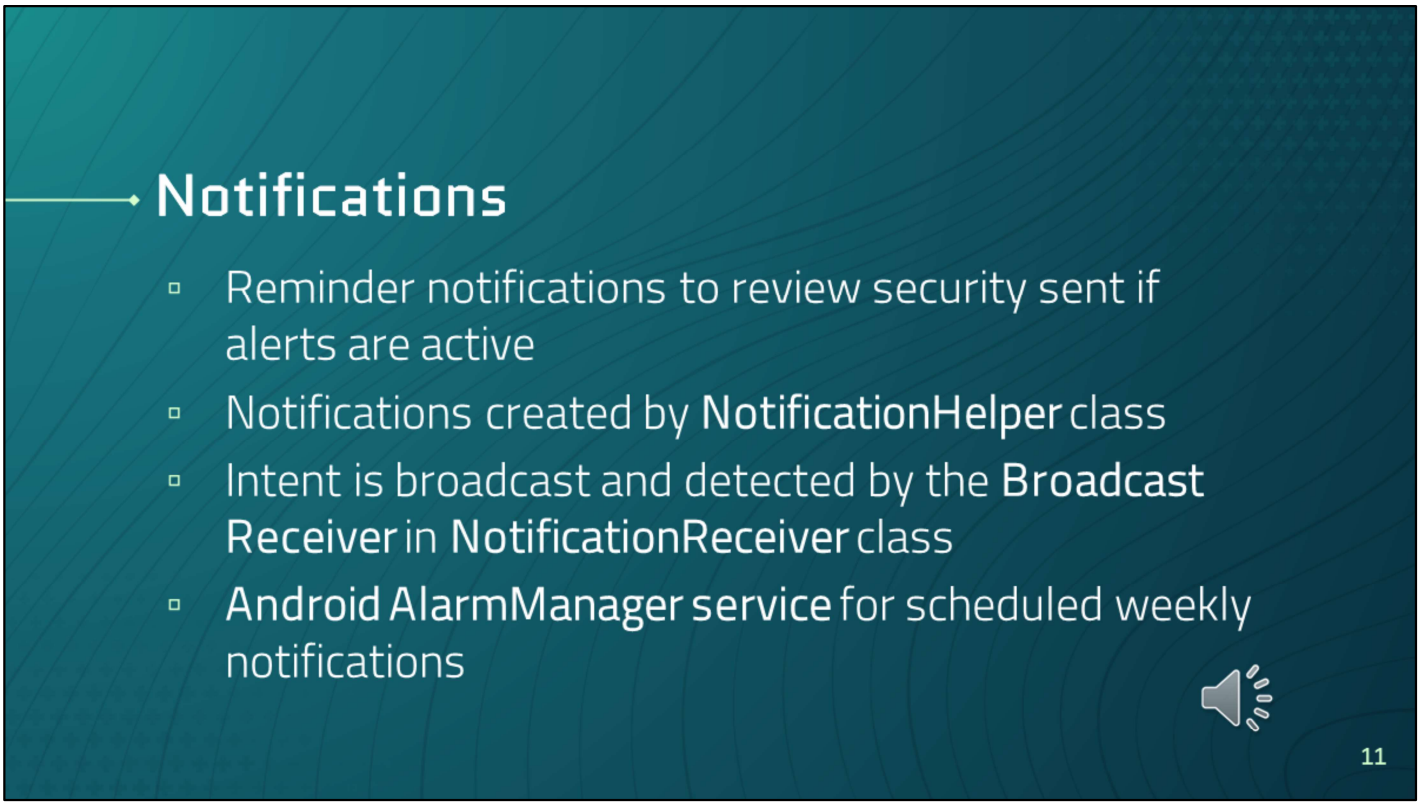
```
// Add OnClickListener to the entry row
entryLayout.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Create and show the AlertDialog
        AlertDialog.Builder builder = new AlertDialog.Builder( context: ManageServicesActivity.this);
        builder.setTitle("Delete Service");
        builder.setMessage("Are you sure you want to delete this entry?");

        // Set the positive button (Yes)
        builder.setPositiveButton( text: "Yes", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                // Call the deleteServicesEntry() method
                dbManager.deleteServiceEntry(serviceID,  deleteAllEntries: false);

                // Display confirmation toast message
                Toast.makeText(getApplicationContext(),  text: "Record deleted.", Toast.LENGTH_SHORT).show();

                // Recreate the activity, to refresh the UI
                recreate();
            }
        });

        // Set the negative button (No)
        builder.setNegativeButton( text: "No", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                // Dismiss the dialog and do nothing
                dialog.dismiss();
            }
        });

        // Create and show the AlertDialog
        AlertDialog dialog = builder.create();
        dialog.show();
    }
});
```



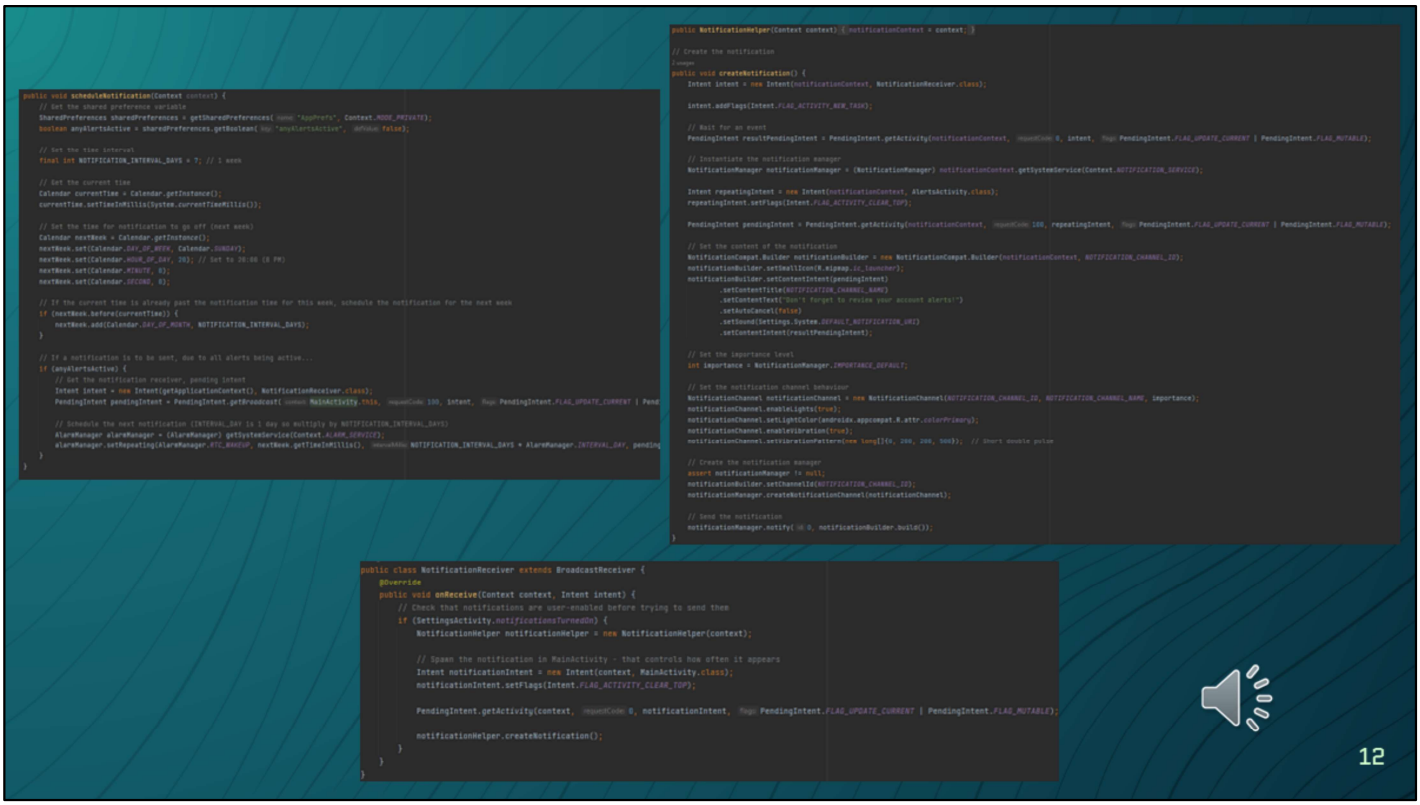
Recycler Views are used on the services and accounts listing pages (activities) to display the table records in scrollable rows. When clicked (tapped) on, these open a pop-up dialog to confirm if the user wants to delete the selected record.



The previous slide showed the code used to implement the pop-up dialogs, and what the result looks like. The screenshots here show the database structure as defined in DBHelper, and the Recycler View implementation for the “All Accounts” activity.



Notifications are used to send reminders to review security alerts. This is implemented using a NotificationHelper class that creates a notification once called by the NotificationReceiver class – itself implemented as a broadcast receiver and activated upon the particular broadcast. The reminder notifications are sent each week on Sunday at 8pm, if alerts are active – this is implemented using the built-in Android AlarmManager service.



These screenshots show the code used to schedule notifications used in the “Main” activity, as well as the notification receiver and notification helper classes,

Service Logo API

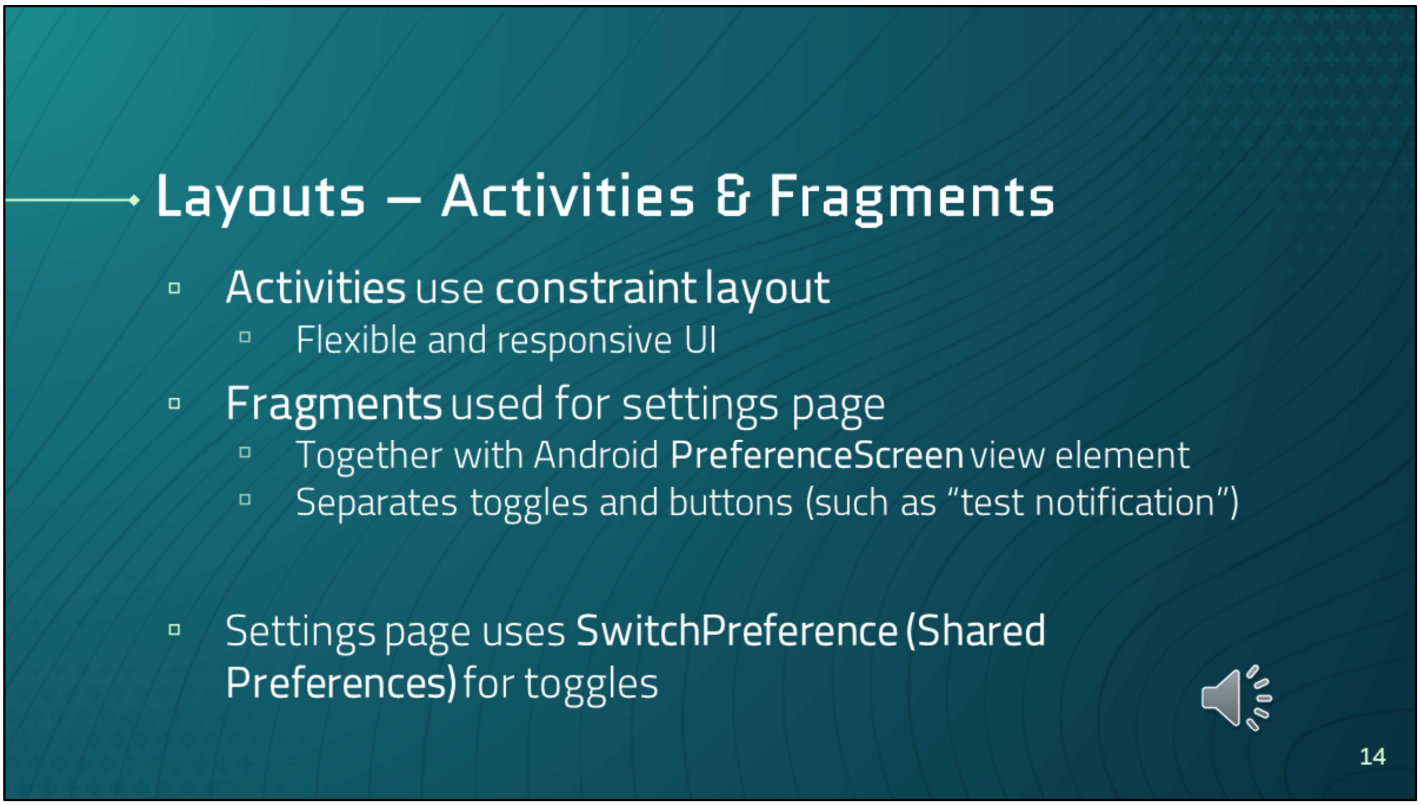
- 'Clearbit' Free Online Logo API
  - Gets service logo based on domain
  - Size, file format, and greyscale options
- Java URL class for Internet connection
- Data stream decoded using BitmapFactory
- Uses asynchronous background tasks

```
private class LoadImageTask extends AsyncTask<String, Void, Bitmap> {
    3 usages
    private ImageView imageView;

    1 usage
    public LoadImageTask(ImageView imageView) { this.imageView = imageView; }

    @Override
    protected Bitmap doInBackground(String... urls) {
        String imageUrl = urls[0];
        // This is "https://logo.clearbit.com/" + serviceDomain

        // Use API, get logo using domain if possible, and return it
        try {
            URL url = new URL(imageUrl);
            return BitmapFactory.decodeStream(url.openConnection().getInputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Return null otherwise
        return null;
    }

    @Override
    protected void onPostExecute(Bitmap result) {
        // If logo was found, use that
        if (result != null) {
            imageView.setImageBitmap(result);
            imageView.setAlpha(1f);
        } // Otherwise, will just use the default image (faded no image icon)
    }
}
```

13

The app communicates with a third-party API called “Clearbit” to get the logo of each service based on the entered website (domain). This also contains formatting options, however these are not currently used in the app. This is a free and simple to use API.

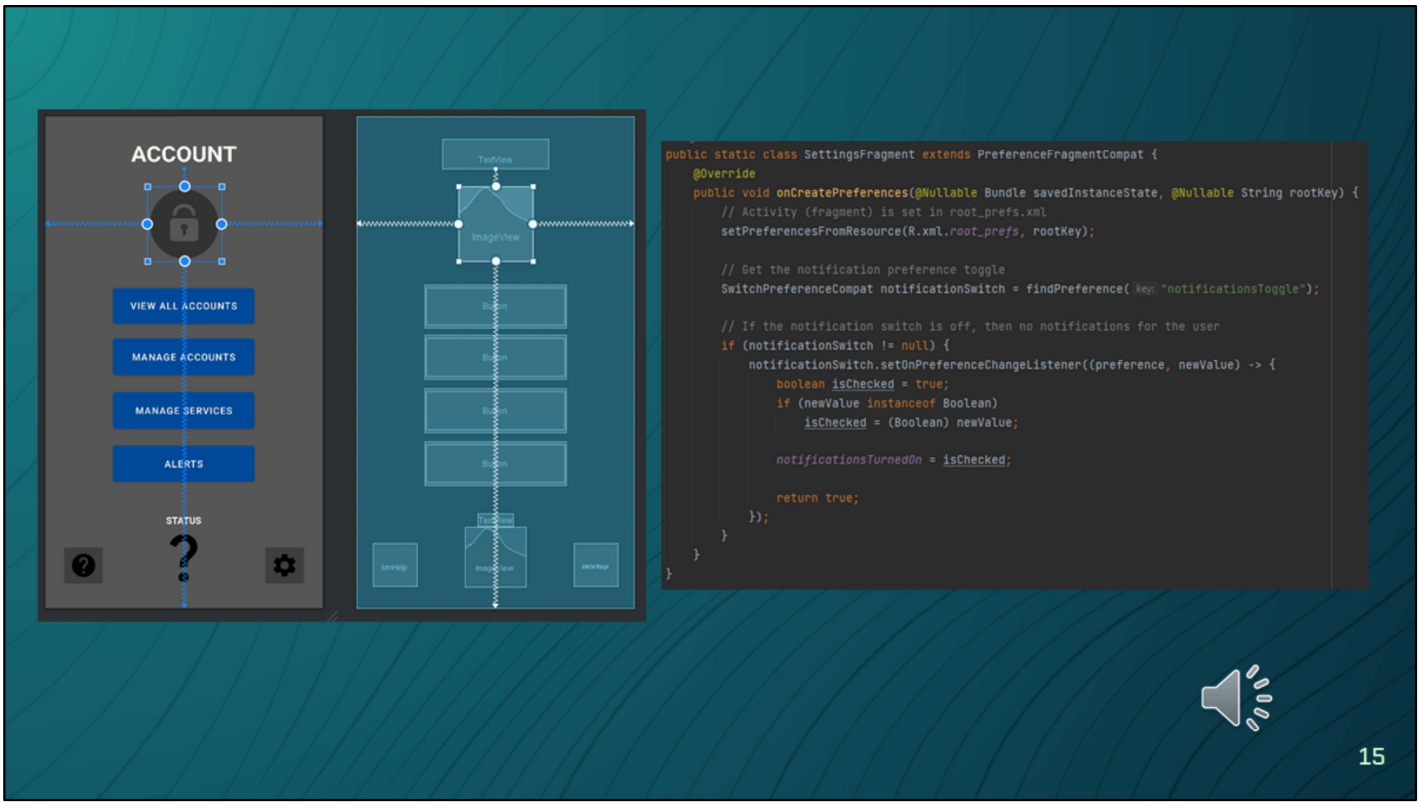
It is implemented using the built-in URL class from Java, which initiates an Internet connection to the API and makes the request. The returned data is decoded and displayed as an image using the BitmapFactory class of the Android Bitmap library. This is accomplished using asynchronous background tasks, in order to allow the main UI to display and update with the logos once retrieved, which prevents hanging or waiting behaviour.



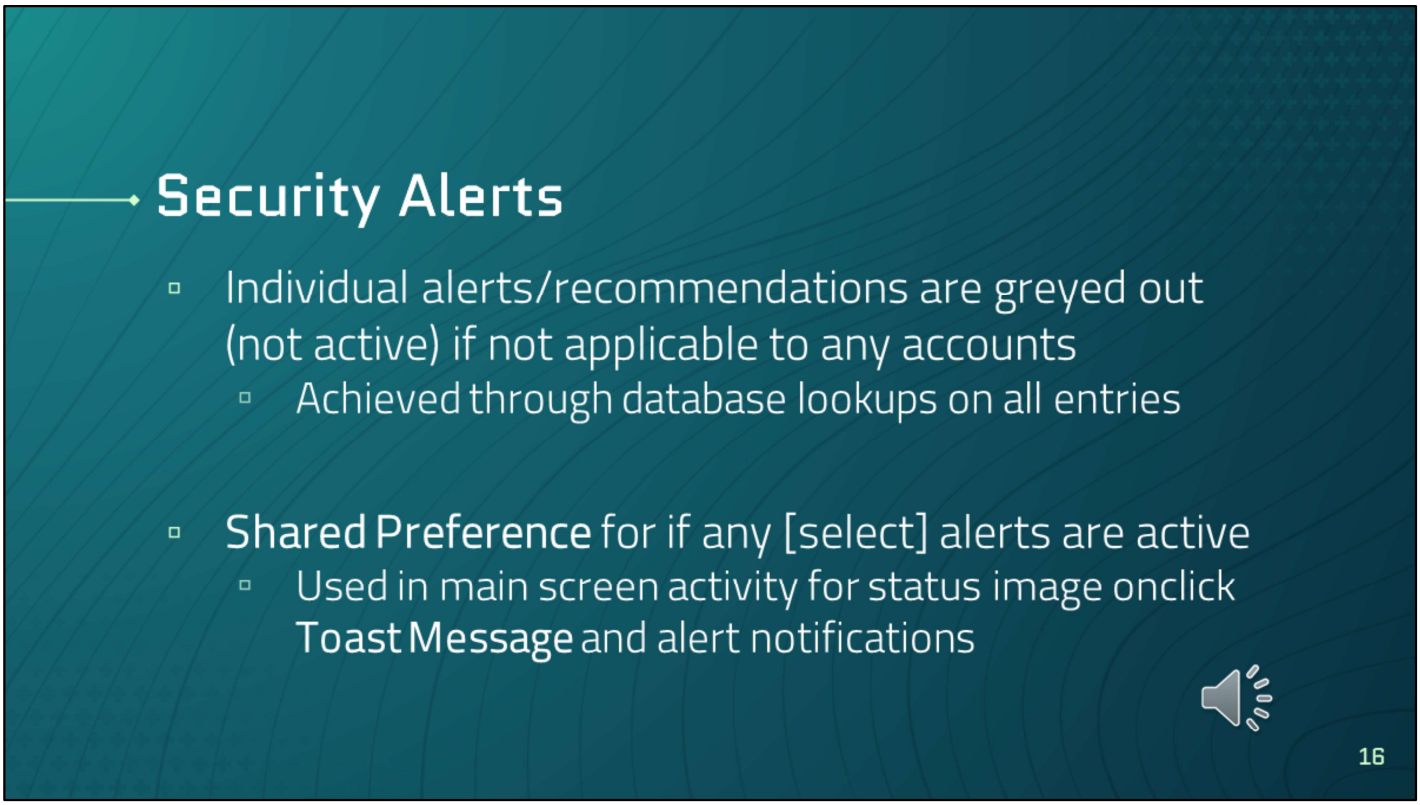
Aside from RecyclerView mentioned previously, other layouts are used in the app. Regular activities use the Android constraint layout for UI design, which allows them to be flexible and responsive.

Fragments are also used, notably in the settings page. Here, the Android PreferenceScreen view element is put into a fragment and then included in the activity page above any buttons. This helps the separate the design and logic (such as onClick functions) for the settings toggles, and any buttons used (such as the "send a test notification" button).

These toggles use a type of Android shared preference called "SwitchPreference" to save the state of the toggle (if it's on or off).

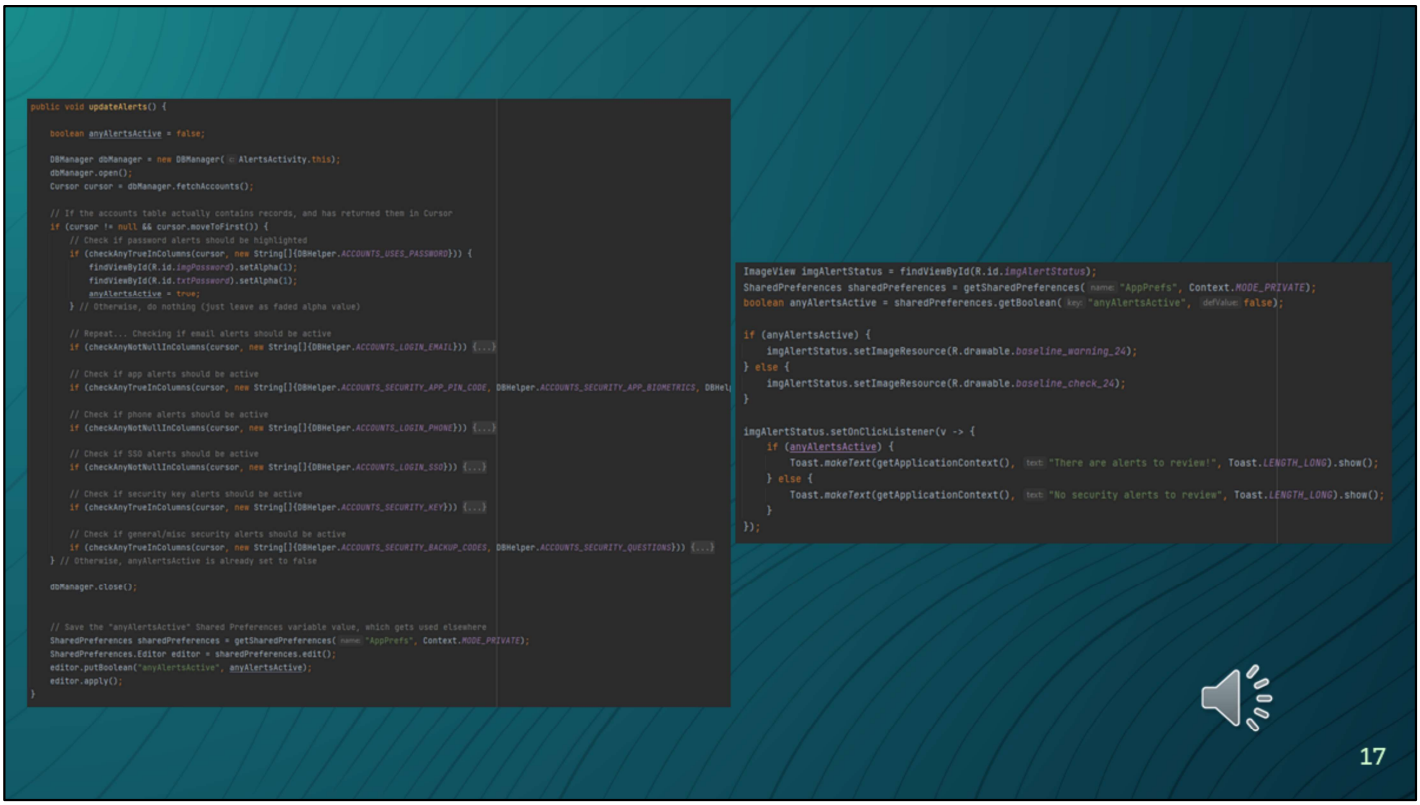


These screenshots show the use of constraint layout on the Main activity, and the code used to implement the switch preference fragment on the “settings” activity.



The “alerts” activity displays all alert messages by default, but greys out (fades) the ones that aren’t relevant. It determines this through database lookups on all records in the account database, and checking the relevant fields. For example, if a security key is used with at least one account, then the security key alert is active.

A shared preference variable is used to determine whether any alerts (or any select alerts) are active. This is used to control whether alert notifications are sent, and also to configure the status image on the main screen activity and its related onclick Toast message.



The first screenshot shows the code used in the “Alerts” activity to update the status of the alerts and update the UI, and create the shared preference. The second screenshot shows how the “Main” activity uses that shared preference to change the status image, and produce an onclick toast message.

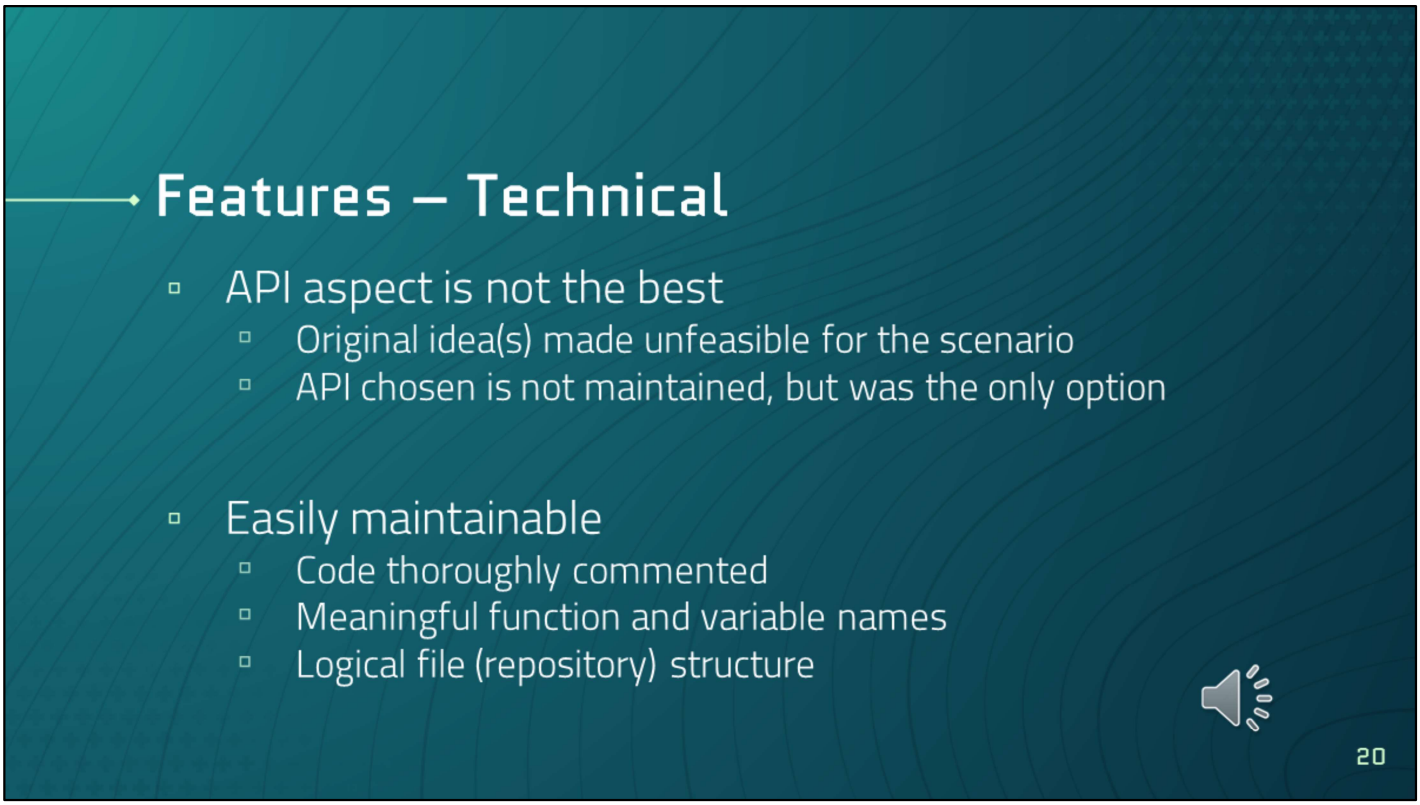
Finally, a critical analysis of various aspects of the application design and implementation.

Features

- Core functionality works as intended
- Varying Android layouts and features demonstrated
- All features serve a purpose

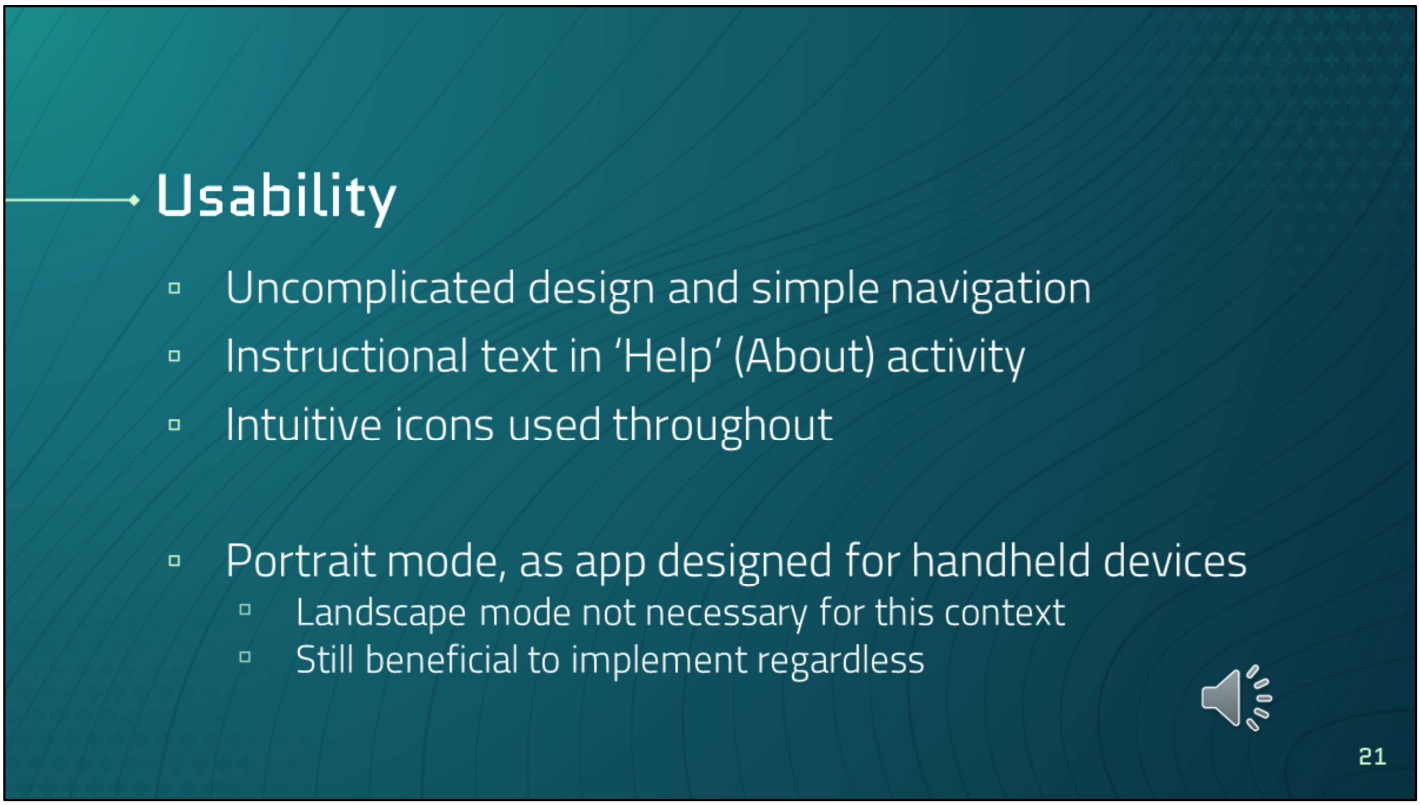
- Original ideas were ambitious and scaled back due to:
  - Time and personnel constraints
  - Android programming knowledge
- App serves the same purpose nonetheless

Overall, the core functionality of the app works as intended, even if scaled back from the original vision due to time and workload factors. The app demonstrates various Android layouts and features to provide this functionality. All features implemented serve a purpose, as they relate to the overall app experience.



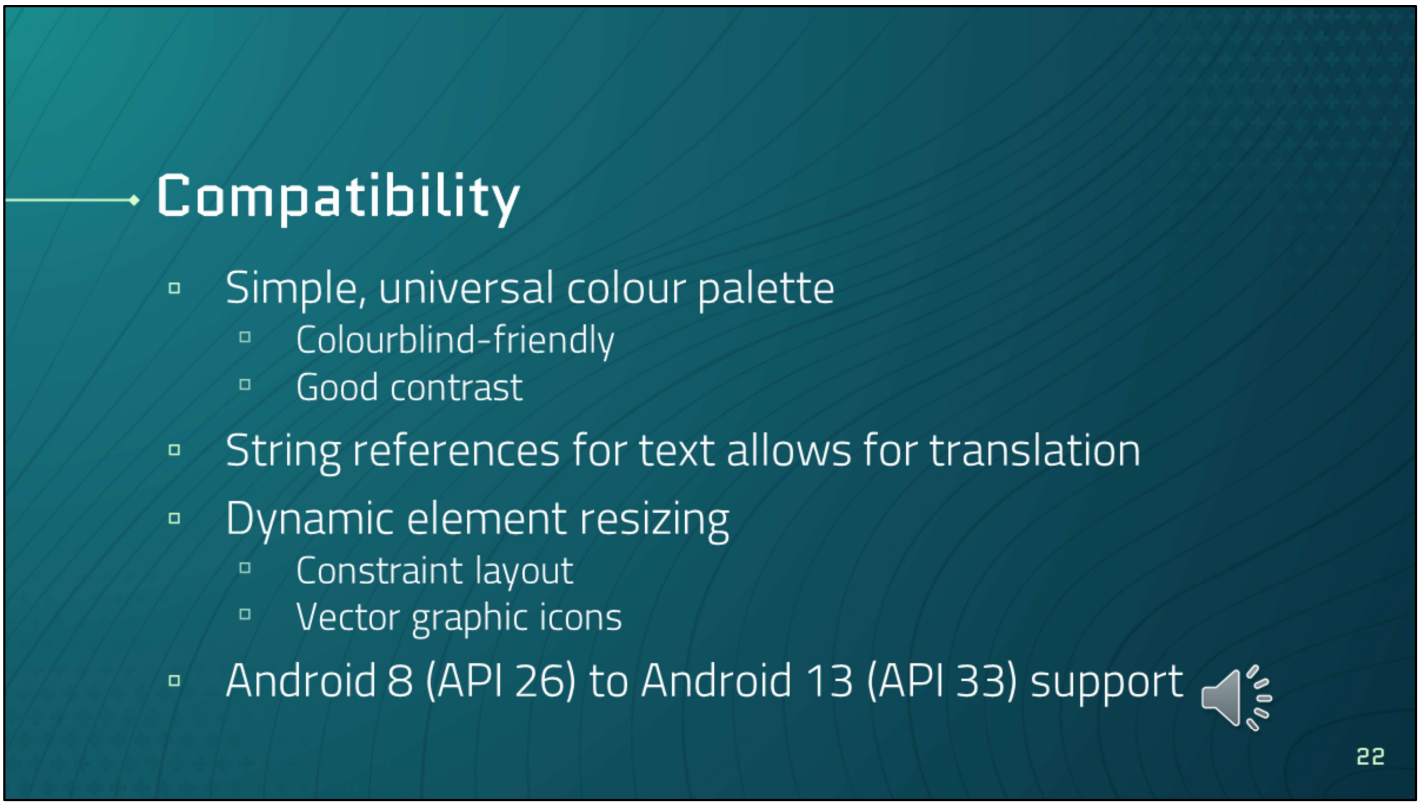
The API usage in this app is not the best or most advanced, primarily due to the issues encountered with the original idea. Without going into too much detail, the original vision was to use the website “Have I Been Pwned”’s API to check if a user’s email had been compromised in a breach – this was premium and did not offer student keys. Likewise, using their Twitter account to alert about new breaches wouldn’t work due to Twitter’s recent API changes making it premium as well.
The API used for getting company logos is technically now legacy, and doesn’t get maintained, however all the other options are, again, premium (paid), so unfortunately there was not much choice.

The code itself is easily maintainable thanks to code comments, meaningful entity (functions and variables) names, and a logical filesystem (repository) structure.



In terms of usability, the app features an overall uncomplicated design with simple navigation. The "Help" activity provides guidance on how to use the app, and intuitive icons are used throughout.

The app is designed and intended for handheld, portrait-oriented devices. That is specifically what Android is made for, and smartphones and tablets both natively have the ability to display this. Whilst landscape mode would be required for devices such as laptops (ChromeOS on Chromebooks is not Android but does support Android apps, for example), this is not a primary concern for Android app development normally, and was not a core requirement for this project anyway. Due to time limits on development, redesigning the UI to *appropriately* support landscape mode was not feasible, and would have wasted resources given the context. Nonetheless, it would still be beneficial to implement in future.

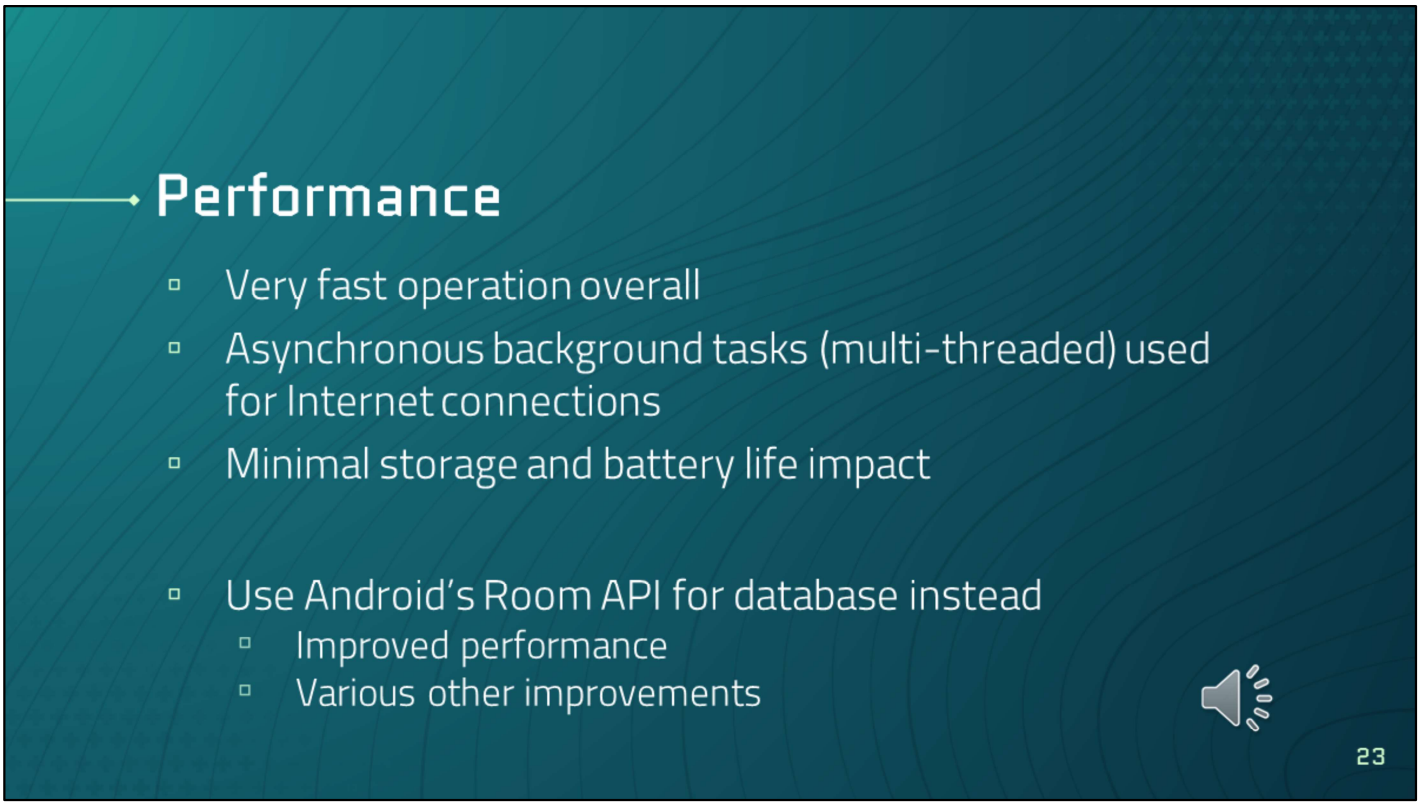


In terms of compatibility, the app uses a simple colour palette with a good contrast and colourblind-friendly colours. This uses a grey background appropriate for viewing in darker environments, so a separate dark mode theme is not necessarily needed. A light mode may still be useful for accessibility purposes however, which should be explored in future.

All UI text used throughout the app uses string references, which allows for easy translations.

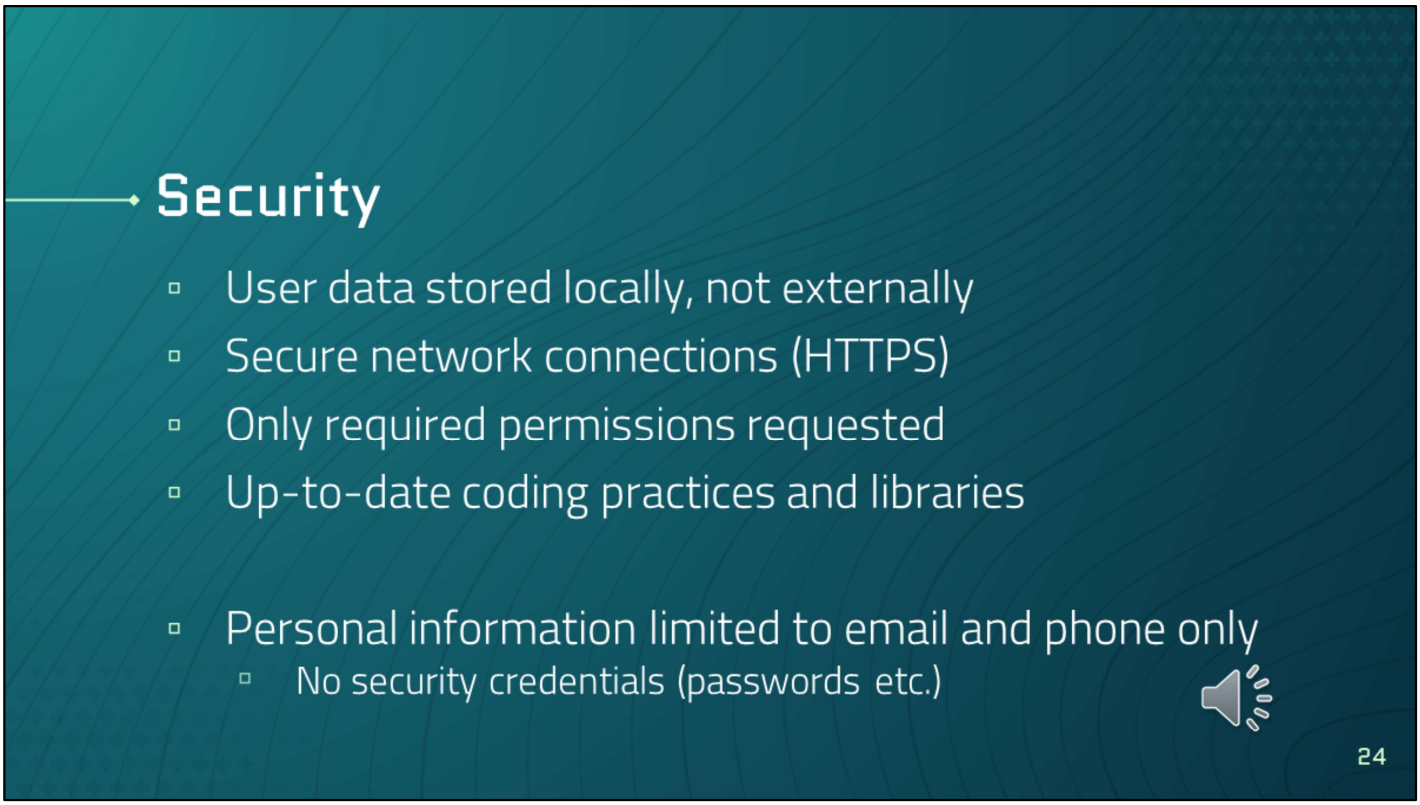
The app also supports various dynamically resized elements for compatibility with different screen sizes and accessibility features. Constraint layout in activities, and icons using vector graphics achieve this.

The app is built for API level 29 (Android 10), however offers support for devices from Android 8 (API 26) to Android 13 (API 33) – where the demonstration video was recorded. The former is now almost 5 years old, which is the typical maximum time that a modern smartphone will receive security updates – after 5 years, support is dropped.



As for performance, the app performs very well overall, with incredibly fast operation – however it is not complex so that is not particularly surprising. As an example of good performance techniques, the multi-threaded asynchronous background tasks used to retrieve service logos via that API, help to improve performance and prevent waiting/hanging behaviour.
The app is very small – at least in comparison to many commercial apps nowadays – requiring megabytes in the teens numeric range, and has a negligible impact on battery life.

One area that can be improved here is to switch to using Android's "Room" library for database operations. This acts as an abstraction layer over SQL Lite, and helps to improve performance and efficiency, alongside providing various other improvements such as simplified database management, seamless integration and data handling, and testing and maintenance support.



Regarding security, user data is stored locally on the device, rather than externally on a server, which could otherwise be a potential concern. When communicating over a network (such as for the API logo lookups), only secure connections are used (through HTTPS).
The app only requests the permissions it needs to function – the only one here being notifications, which can be turned off anyway – and up-to-date coding practices and libraries are used throughout.

The app does collect personal information, however this is limited to a user's email address and/or phone number, and does not collect any security credentials (such as passwords or tokens) or other information about the security methods – only whether they are in use or not. The information collected itself is not *extremely* sensitive, in the same way financial details are for example, but *is* sensitive nonetheless, and therefore should really be managed with increased security. This is discussed more in the following slides on improvements.

Security Scan

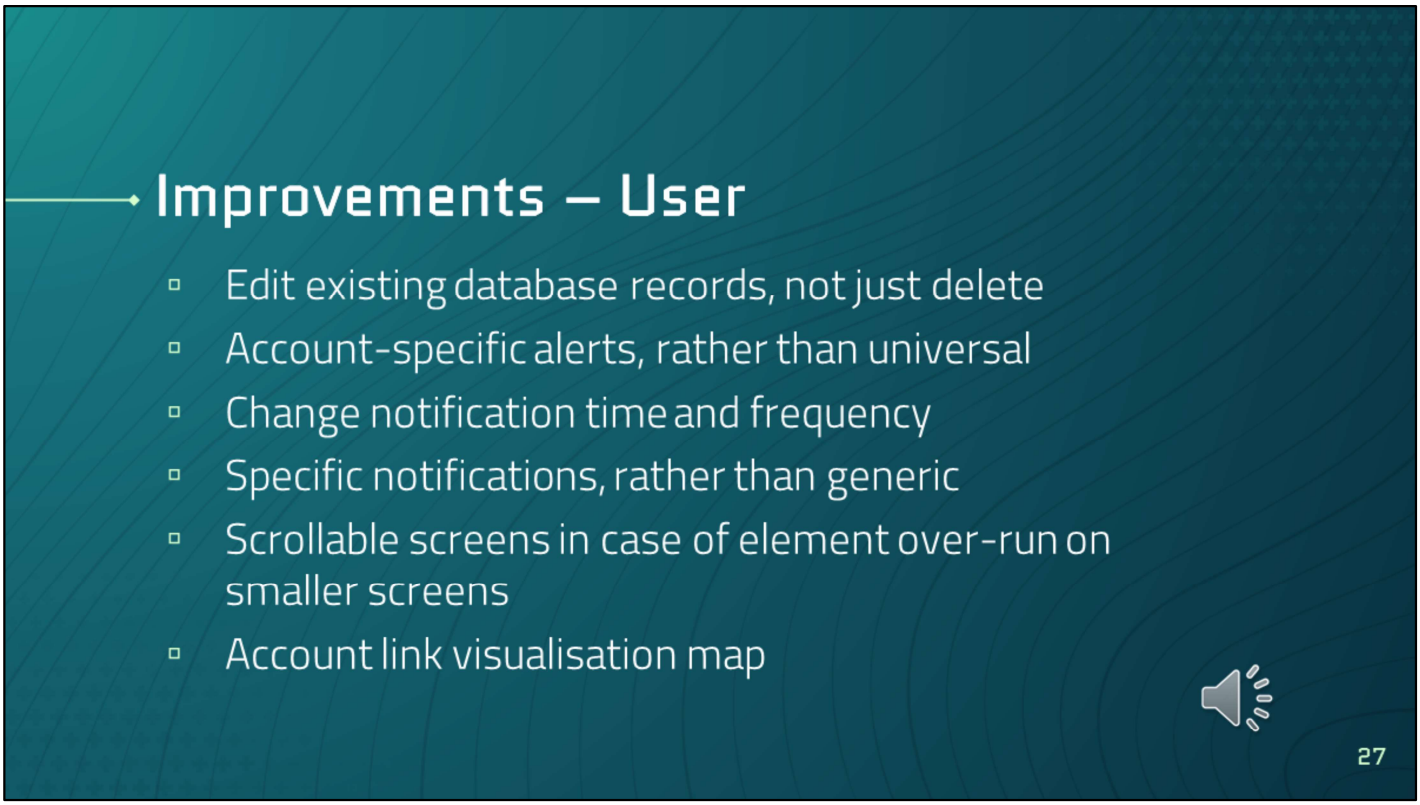
- REX Scanner used to review the app's security
  - 3 Low and 8 Medium severity issues
- Object deserialisation
- Unencrypted Socket
- Potential Path Traversal
- Predictable pseudorandom number generator

The online android security scanner tool “Rex Scanner” was used against the app. This identified 3 Low-severity and 8 Medium-severity issues. The main issues were related to object deserialsation, use of unencrypted sockets, potential path traversal, and predictable pseudorandom number generation. These are not major issues though, and can be investigated and rectified further in the future.

Aside from those mentioned previously – such as better database operation, the developer has identified several areas for improvement.

A major technical improvement would be that of implementing robust security in the app, due to the sensitive information it collects. This includes things like secure programming practices and data storage, but also protecting access to the app (or just particular areas) with a security system such as a pin-code or built-in phone biometrics.

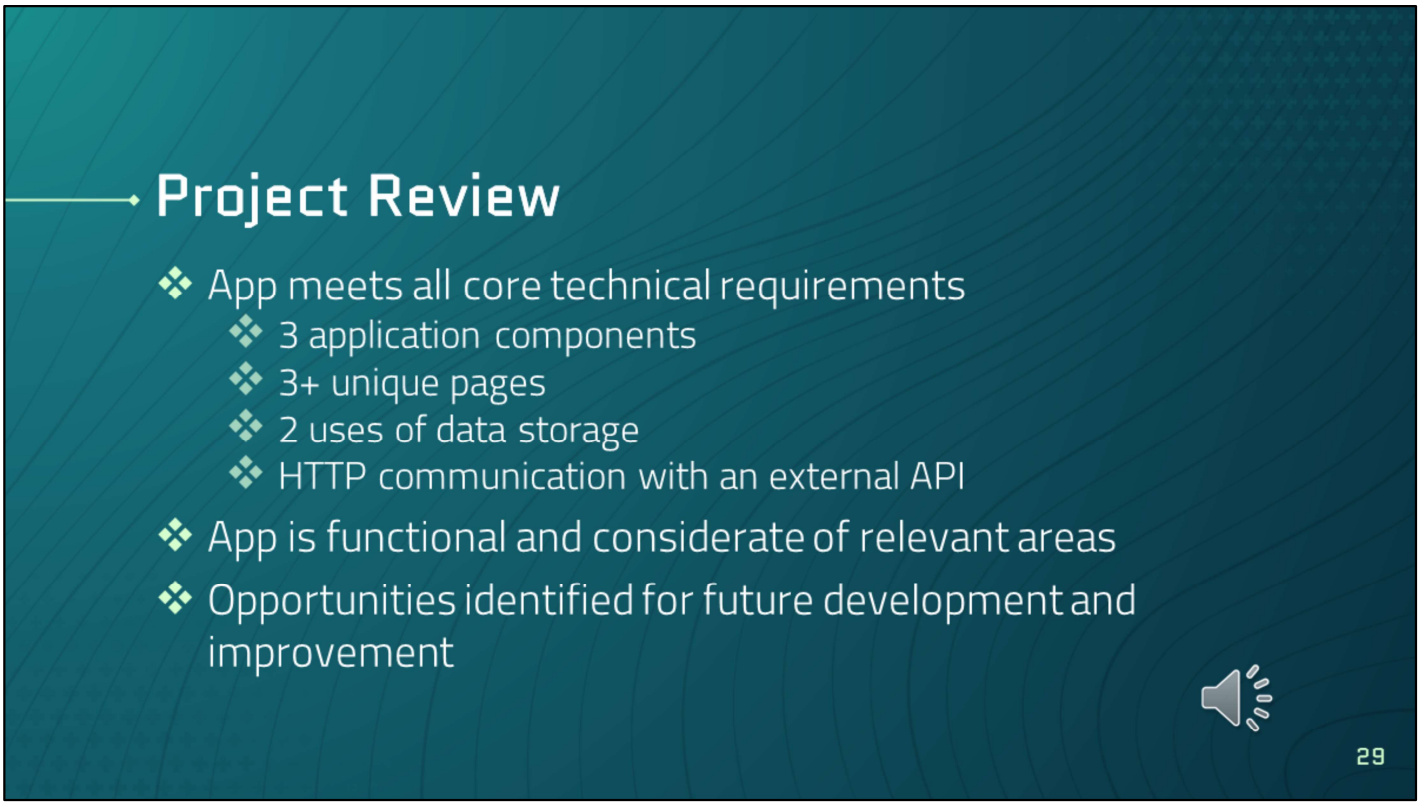
Currently, the app doesn't do much in the way of input validation, which could cause issues if a user enters incorrect information. Whilst there are checks built-in to areas such as the database, this only goes so far and doesn't check for things such as incorrect domains or blank data entries, at the point of input.



Aspects affecting the user or concerning the app design can be improved, such as the aforementioned expanded orientation and theme options.

- The ability to edit existing database records for the accounts and services tables would be greatly beneficial, as currently they can only be added and deleted.
- Implementing account-specific alerts, rather than just universal alerts would also be very beneficial for the user
- The settings activity should have the ability to change the time that the reminder notifications are sent, and at what frequency
- Likewise, adding specific notifications rather than just generic notifications would be very good
- In cases where the screen is small enough, text or other visual elements may be hidden beyond the page, so having scrollable screens would add protection from this
- Finally, implementing an account link visualisation map would be useful for the app, and possibly serve as its unique feature going forward.
    - In truth, this idea was the original catalyst for this whole app, and was meant to go in the “All Accounts” activity, however overall workload unfortunately meant this could not be realised.

Some closing statements…



Overall, the app meets all technical requirements of the project and functions as intended. Its design is considerate of relevant areas such as compatibility and security, and several opportunities have been identified for improvement and future development. All of this has been covered in the presentation. I would consider this to be a success overall.

Thank you for listening.

References

Documentation : Android Developers (2023) Android Developers. Available at: https://developer.android.com/docs (Accessed: 21 May 2023).

Logo API (2021) Clearbit API Documentation For Developers. Available at: https://dashboard.clearbit.com/docs#logo-api (Accessed: 21 May 2023).

REX | Home (2023) Rexscan.com. Available at: https://www.rexscan.com/ (Accessed: 21 May 2023).

Presentation template by SlidesCarnival