Hello. I'm Matt Jackson. In this presentation, I'll be conducting a code review of the artifacts to be used for the capstone project of my SNHU computer science software engineering bachelor's degree. I actually already hold a master's degree in computer science, but I felt that was much more theoretical than practical. I wanted to get more hands-on-keyboard learning and experience and I wanted to use up my GI bill from my time as Army intelligence officer for USCYBERCOM. I also have a few years' experience as a Navy Civilian, developing and conducting cyber security tests for avionic equipment. I am currently seeking work, and am most interested in a role as a penetration tester or SOC analyst, but am also open to software development and reverse engineering.

In this code review, we'll start with an overview of the structure, flow and functionality of the application. Next, we'll look at the documentation. We'll examine the meaningfulness of the variables and look for redundancies. There aren't really any arithmetic operations other than converting date times from human readable to epoch time, but we'll examine that process. We'll look at any loops and branches to ensure they function and end correctly, and finally we'll talk about security measures.

Artifact 1 is essentially the entire project as the enhancement goal is to translate it from Java to Kotlin. We will now discuss the functionality and features and analyze the existing code. We will review and analyze the code next. The enhancement will demonstrate proficiency in the Kotlin programming language, and sound mobile development practices such as user interface and user experience, and data management. This is in line with course outcome 4 to use well-founded and innovative techniques, skills, and to implement computer solutions by developing a searchable application that provides an intuitive, pleasing user interface to solve the problem of searching through and sorting a large number of events

STRUCTURE

In the main activity we check that the database has been created (if not we create it), then checks if the user is logged in. If so, it takes us to the EventList activity, otherwise it takes us to the login activity.

Upon creation of the Login activity, first permissions are checked to ensure the user wants to allow the app to send SMS messages for alerts when an event is due.

After permissions are selected, the user has the option to login or register. Several text listeners are set up and upon filling out the fields, registration and login buttons are enabled.

The login and register buttons check the user's credentials against the database, and depending on the results of the query, send the user back to the main activity page or return to the Login page. An infinite loop that breaks when the user successfully logs in might be a better structure than starting a new Login activity each time. Same with the Main intent.

Once logged in, the user id is passed via shared preferences back to the Main activity where is checked against the default value. If the user id is anything but the default value, the activity continues to the Event List Activity.

Here the event list is displayed. Upon creation of the activity, the database is queried for the user name and events list with the logged in user's id for all their saved events, which then populate the events list. There are also button listeners for editing events and logging out, and, when populated, individual events from the list view.

The edit event page populates the event fields, and has a date & time pickers Each field is editable. There are buttons for deleting and saving each event, as well as setting a text message alarm at the time of the event. The time picker isn't always properly formatted and doesn't default to the current time.

FUNCTIONALITY

We mentioned permission checking earlier. This check may be better moved from the Login activity to either the main activity or the registration function. Moving it to the main activity makes sense as it's a pre-requisite, while moving it to the registration function would allow each user to choose their own permissions. In practice, for a single user phone, allowing each user to select their own permissions could be sub-optimal as one would have to reset the permission to deny each logout, otherwise when one user allowed it, all would be allowed to send SMS messages. However, if the permission is denied on logout, the user would have to make the choice each login. This is also less than desirable, so it should be moved to the main activity, and only the first user will make the choice.

Alarm Receiver:

It appears that a txt message can't be blank; my app kept crashing if it was. So in the alarm receiver there is a comparison that sets the event title if it's not already set. The alarm time is set when an event is saved with an alarm in the Edit event activity.

DBHelper

Handles database functions such as creating the database and tables, inserting events and users, updating and deleting events. It also handles user verification.

Event and User are class objects holding the two types of information. Because there are only two simple objects, they could be consolidated into one file

Here is where the majority of enhancement 3 will be focused by building more API calls to use advanced database search functionality such as matching partial words or date ranges. The DBHelper is not very secure. We will discuss this further below in the security section, but a number of the helper functions use direct user input rather than sanitizing and parameterizing the queries. This makes them ripe for sql injection. Correcting these flaws will be in line with course outcome 5: developing a security mindset that anticipates adversarial exploits.

EventViewAdapter

This sets the given event date and title fields. For enhancement 2, this class will be used to sort the list in different orders depending on user selection of date or event title. Here is where the binary search tree data structure will be built to then feed the Events list. This aligns with course outcome 3, designing and evaluating computing solutions that solve a given problem using algorithmic principles. Building and using a local variable instead of querying the database should make the app more responsive and pleasing to the user.

Receiver

The Receiver class is superfluous, and its only functionality is to create a Log message.

A “Magic number” is used in the alarm receiver class for the phone number variable used to send the text message alert to. This number is the phone number of the device the application runs on and should be accessed programmatically. At the very least, it should be relegated to a constant at the beginning of the class where it can be easily found and changed.

DOCUMENTATION:

Documentation in this application is more or less non existent. The vast majority of comments are test code that has not been removed. This needs to be remedied in the capstone project.

VARIABLES:

Variables are somewhat hit or miss when as to when they clearly describe what they're used for and when they are vague. Most of the time they are defined and used closely enough together to be found, but this does not lend itself to modularity or expansion.

For example, in DBHelper, the sql String variable on line 301 in getEvents could better be labled as sqlQuery. The cursor varibable, c on line 302 is not clear when used even a few lines later.

Functions are better named.

Because the capstone project will be written in Kotlin rather than Java, the android developer Kotlin style-guide should be referenced.

https://developer.android.com/kotlin/style-guide

LOOPS AND BRANCHES

There may be some opportunities to use more loops, rather than calling activities. Additionally, more thought should be put into what stage of the activities should be called when (such as on Start, Stop, Destroy, Pause and Resume. Right now All intents are called to on create, and sometimes loop back to themselves. When to finish the activites should also be explored.

The only for loop used is in the EventList activity when the database list is iterated through to create an EventsView for each event in the list.

DEFENSIVE PROGRAMMING:

There are a number of unused imports in just about every class, view and activity. (show examples) These should be removed to avoid introducing security flaws (even if unused, if arbitrary code can be run due to injection or a buffer overflow, flawed imports could be accessed). Likewise with test code and even superfluous logging. (Show examples) Both increase the attack surface by providing information leaks.

In the DBHelper, the code may be vulnerable to SQL injection due to user input as the insert events, add user, and remove event functions, don't sanitize or parameterize input. This is especially dangerous in the insert events and add user functions as they openly rely on user input. The remove function uses existing event ids to query the database, but in the event the application is expanded, this API should also sanitize and parameterize its input.

A major flaw is that the database stores passwords in plain text. This is especially egregious because the Username field at the top of the Event List page erroneously displays the password instead of the username. This is due to using the wrong column index in line 131 of the DBHelper. The correct column index is 1, but this should be

improved even further by not retrieving the password at all: SELECT user from rather than SELECT *.