Lappeenrannan teknillinen yliopisto
School of Engineering Science

Sofware Development Skills

Miro Rahkonen, 0561542

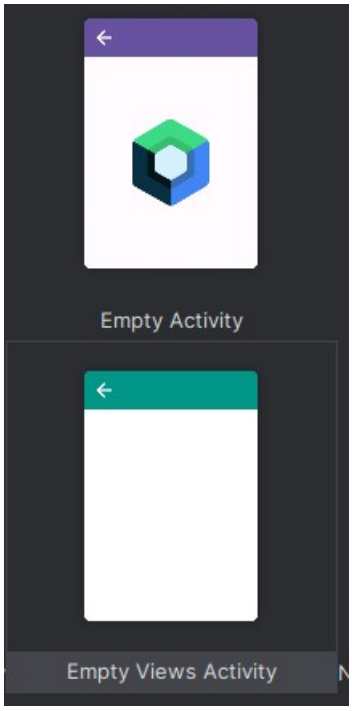
LEARNING DIARY, SOFTWARE DEVELOPMENT SKILLS: ANDROID

LEARNING DIARY

24.02.2024

Today I started the course and went over the topics that will be covered in the course. I had to first install the programs and packages that were needed for creating the android applications during the course. Installing and setting up Git, Android studio, and VSC were easy enough and I was able to start following the first tutorial videos listed on the moodle page.

Right at the start of the tutorial video I had issues with creating the project in Android Studio, as the activity type “Empty Activity” picked in the video would default to creating files using the newer Kotlin programming language instead of Java. The other files in the project were also different compared to what was shown in the tutorial. Kotlin was way different to use than Java so I tried converting the main file into a Java version, but it would always create some errors. This is one of the errors which got printed: *“You need to use a Theme.AppCompat theme (or descendant) with this activity“*, which I tried debugging through different stack overflow threads, but in the end I wasn’t able to fix it. I figured the issue was with initializing the project with setting Kotlin as the main programming language, so I ended up starting from scratch and created a new project using Java. I googled and found out that in the project creation step you need to pick “Empty Views Activity” instead of “Empty Activity”, which gave the option to use Java instead as the language. This fixed all my issues and the process to create the application was the same as was shown in the tutorial.



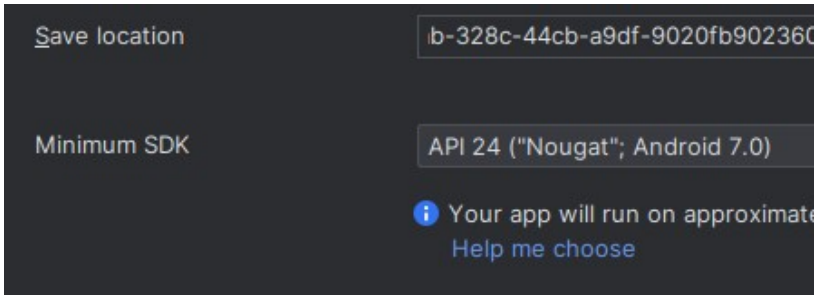
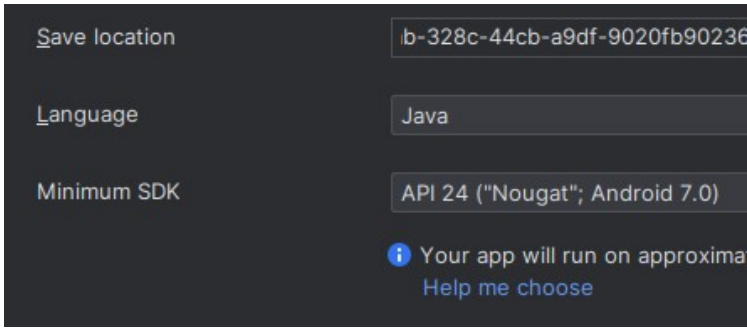


Figure 1: Settings for ’Empty Activity’ project



After figuring out how to create the project using Java, I managed to create a basic application according to the tutorial to get 2 numbers from text fields and adding them together when a button is pressed. Creating the layout for the app in the xml file's graphical user interface was quite easy. I haven't used Java in years, so I definitely need to re-learn a lot of the basic operations. Thankfully some of the operations are very similar to JavaScript, which I have a lot more experience with, so things like initializing objects by the object's ID, and getting data from the object were simple for me to grasp. At the end of the tutorial, I learned the basics of debugging applications using breakpoints and how to read the outputs line-by-line to determine issues in the app. This feature seems extremely useful, and I will focus on learning to use it better in the future. I'm not sure if this sort of line-by-line debugging is available in other programming languages or IDEs though it would be very helpful. Learning debugging using this tool will be very beneficial for future projects so I will focus on learning to use it.

Figure 2: Settings for 'Empty Views Activity' project

25.02.2024

Today I went over the second tutorial video and learned what activities and intents are. The tutorial video taught how to use intents to switch between different activities in the application when clicking a button, and putting data inside intents using the .putExtra() function to send data between different activities. The example application was very simple for now, but it gave me a good idea of what activities and intents are about and how to use them. The main operation of these functions was easy to understand but I am guessing it gets a lot more complicated when a lot of data needs to be exchanged between different activities. I didn't have any issues during this part of the tutorial as the information about intents and activities was still relevant and hasn't changed since the video came out.

In the tutorial I think it would have been informative to show how to create a button to go back to an activity as now in the app there is no button to do anything when going into the SecondActivity screen. To practice working with intents and activities myself I created the button on the second activity screen to go back to the main activity. Initially I did it by

initializing a new intent into the main screen as the new activity and starting the intent. However, when going back and forth between the main/second activities, it wouldn't close the previous activities and a lot of activities would be running at the same time which isn't ideal. Pressing the back button on the phone would go through all the previously opened activities and the user would have to press the back button a ton of times to eventually close the app. I then googled the correct way to "close" activities and found out that it was as simple as executing the finish() command, which would close the current activity and return to the previous one. Doing this feature gave me some more insight into working with intents and activities. I think these features are very important to learn for all android applications, so this lecture was very informative and helpful in learning about mobile development as a whole.

28.02.2024

Today I went over the third video in the tutorial and created an app that lists out a bunch of different items with information about each item. Additionally, clicking on an item in the list would open an activity which for now only displayed an image of the item but more things could be added to the activity as well.

In the tutorial a ListView element was used to list out the items, but it has since the video's release been made into a legacy element as a new and faster element called RecyclerView was created to replace it. ListViews were said to be very slow when dealing with a ton of listed items, so they weren't efficient. I tried using the RecyclerView element for the tutorial but creating it was a lot different from the tutorial, so I just ended up creating the list the same way as was shown in the video using ListViews. I figured that it would be more useful to learn the current methods of creating lists in modern apps as ListViews aren't widely used anymore, so I intend to learn to use RecyclerViews in the final project for this course so I can learn how lists are created in current applications.

In the tutorial I learned to create a list with a custom layout for each listed item to display important information about each item. The custom layout for the item listings were

created using a new layout file named listview_detail which would be placed into each separate item in the list. In the tutorial an adapter was created called ItemAdapter which was used to generate new items into the ListView. The adapter would use the custom layout which we created previously and would get data from the strings.xml file where we stored all the necessary data for each item. The strings.xml file was mentioned in previous tutorials but using it wasn't explained in them as it wasn't needed at the time. I was curious how the values were accessed, and it was finally explained in this video. By the end of the tutorial, we learned to use the adapter, custom layouts, and the strings.xml file together to dynamically generate a listing for items written into the strings.xml file and adding more data into the strings.xml file would generate more items into the application's ListView. Following along the tutorial was quite simple and I was able to understand most of the concepts that were needed to produce the list. I will definitely need to experiment more with ListView and RecyclerView to internalize creating them in the future. To get more practice with lists in android apps I will create one in the final project for this course.

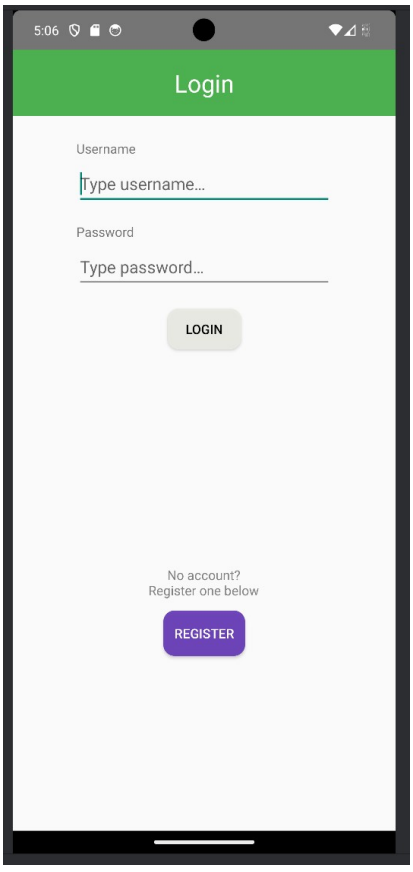
To make the list interactive, the tutorial showed how to add click listeners for each item in the list and switching to a different activity called the DetailActivity, which would display details about the selected item. For now the only thing that was shown on the screen was a picture of the item but more information could be easily added to the page if needed. Getting the image from the project files and setting it into an image element was quite easy but scaling the image down was quite complicated. I understand the overall concept of what is done in the image scaling function, but I wouldn't be able to remember the commands from memory. I would be able to add it onto other projects if I look at what is done in this tutorial application. The methods of making this functionality haven't changed since the release of this video so all the operations shown in the video still worked the same way now as they did when the video came out. Instead of ListViews, however I will be learning how to do the same functionality with RecyclerView elements instead as they are more relevant nowadays in modern apps.

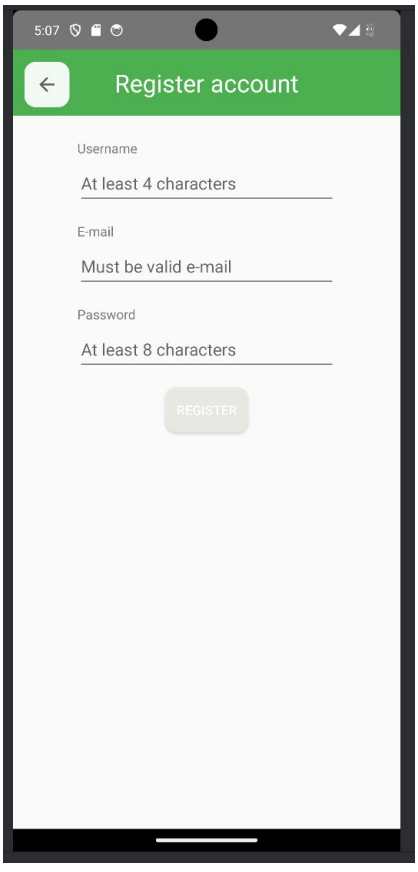
This tutorial gave me a good idea of using lists in android apps and I will be learning to use them more in the project for this course. Today's studying gave me a good started on what needs to be done to implement a interactive list into an app which I think is quite useful for

a ton of different situations, like a shopping app showing a list of different products, or a texting app listing the user's contacts which you can press and open a chat window. Lists can be implemented in a ton of different ways and the video taught me how to change the appearance and function of the ListView element. Next time I will start working on the project for this course and create some sort of app that will implement all the topics which were shown in the tutorial videos.

29.02.2024

Today I started working on the project and decided to do a note taking app with a login and register system. For now, I worked on the layout of the login and register screens and learned how to create an appbar for each screen and change its layout based on the current screen. This stackoverflow thread helped me a lot in understanding how to implement a single element, like an appbar, on multiple different screens without having to create it from scratch on all the screens. Using the thread I managed to create an appbar at the top of the app and I added a TextView and a button which would be useful in navigating the app. The button would return you back to the previous screen, and the TextView would always show the title of the current screen, like the login page or the register page. Below are screenshots how the screens look like currently and I will continue working on them over time.





I found out that having rounded corners for buttons wasn't a native property you can change for buttons, so I searched for ways to add them from different websites. This stackoverflow thread helped me a lot in understanding how layouts can be created for an element and how to add the layout to a button. To implement the rounded corners, I got some practice in modifying the XML file of the layout, which I preferred over the Design interface of Android Studio. In the tutorial the layout of the app was designed entirely using the design graphical user interface and I found it very slow. Modifying the code itself was much faster, so in the future I will mostly be learning to code the xml file instead of using the graphical user interface.

To make the app a bit more responsive I changed the behavior of the buttons so that the login/register buttons couldn't be pressed unless all the necessary data was put into the TextView. I found an interface called TextWatcher which would allow this sort of functionality. I checked the TextWatcher documentation on how to implement it and managed to make the login button clickable when the user input was valid, and unclickable when the user hasn't put in both the username and a password. Doing this feature taught me how to set variables of an element, like the background color of a button and the interactivity of the button, using the java code.

I was able to implement the general functionality of the login/register system, like switching to the register screen from the login screen and checking the input from the multiple TextViews. Next tile I will figure out how to create and store user accounts on some sort of database and login using the created account's credentials later. I have used mongoose databases previously for web applications to store login data locally, and I am interested in learning how a similar system can be done on mobile.

04.03.2024

Today I continued from last time and finished the user interfaces for the login and register screens and cleaned up the code for checking the login credentials by removing some redundant lines of code and creating a separate function for enabling and disabling the login button. I added the same checking function for the register page as well. I didn't have a lot of time today to work on the project so I mainly just cleaned up the project a bit and

created the git repository for it. I watched a tutorial on how version control is done on the Android Studio application, and it was quite easy and I was able to get the push the current version onto a git repository using the application. I will be pushing new versions onto the repo from now on. I have quite a lot of experience with git so using it isn't anything new, though I needed a refresher on how it is setup on Android Studio as I haven't used it in a couple years.

05.03.2024

Today I worked on the authentication system and integrated an SQLite database into the app with help from this video tutorial and this article on the topic. I googled different methods of implementing a database into the android app and the most popular methods used SQLite so I decided to use it as well. By following the tutorials I was able to create a SQLite database with methods to register a new account and login to the account using a username and password. For now, I only have the users stored on the database but later I will add the functionality to store notes for individual users, and then fetch the notes into a list to be displayed on the screen.

I had quite a lot of issues with implementing the database as the method used in the video would crash the app while inserting data into the database and I wasn't able to find a way to fix it. These issues, however, gave me some practice in using the debugging feature of Android Studio and I was able to figure out which line of code was causing the crash. Since I wasn't able to find a fix online I searched for another implementation and an article showed a different method which used an additional class to handle the queries. Eventually I was able to make the database work with my app while modifying it for my use case. I had to research some commands in detail as the tutorials didn't explain what each command did in detail. With the research I figured out the general functionality of the Cursor object, which is widely used in the SQLite database on Android. I learned how to query data from the database and store the data onto a cursor, which can then be returned elsewhere to display the data. Implementing the database into my project gave me a good understanding of how databases can be added to apps and it was quite enjoyable for me to

learn. I have used SQL databases before so I knew how it worked and the tutorials taught me how to use the SQL commands in the Java code.

06.03.2024

Today I continued on integrating the SQLite database into the app and added a notes table which stores all the users notes, and the notes for specific users can be fetched when a user logs in and is transferred onto the notes screen which displays the notes on a RecyclerView. I also implemented a RecyclerView which would display all the notes for a specific user. In the video tutorial for this course we used a ListView but for this project I wanted to learn to use RecyclerViews instead and I was able to implement it into my app while following another tutorial video on YT and by using some StackOverflow threads which resolved some issues I encountered.

Creating the RecyclerView was sort of similar to a ListView but required some more setup which wasn't too bad while following the tutorial. I was able understand what was done and how I could change it to fit for my use-case. For each listing I added 2 TextViews which would display the user's name and the text of the note. Each listing also has a delete button which I will implement next to delete notes from the database. I learned quite a lot with working with the SQLite database and how to use it for fetching and inserting data. I followed some threads on changing the appearance of some elements, like the buttons, and TextViews and the RecyclerView listings to make the app look a bit cleaner.

08.03.2024

Today I implemented the functionality to delete entries from the database and automatically update the RecyclerView on the screen when the delete button is pressed for a specific note. I mainly followed this video which gave me a good idea of how to produce the functionality I needed for my app though it wasn't wholly applicable to my app since I needed to fetch and send data through a SQLite database which made it a bit more difficult. Initially I had issues understanding how to get the RecyclerView to update the list when new data was entered and deleted from it. I eventually figured out that updating the RecyclerView was done by an adapter and the adapter could be notified that data in the list

was changed. After I understood how the RecyclerView can be programmatically updated the rest of the implementation was quite straight forward. Creating the functionality to delete and add new notes on the app in a RecyclerView gave me a good understanding of how to work with lists and updating them programmatically without having to recreate them from scratch whenever data was added or removed from them.

13.3.2024

I've worked on the RecyclerView over a couple days and learned about how to add decorations into the items in the list. I also updated the deleting functionality as it wasn't working properly previously. I found out that the item position in a RecyclerView doesn't track for the whole list and only for the items that are currently on the screen. For example, when scrolling to the bottom a list far away from the top, the top most item visible on the screen is at the position 0, and not at the very top of the list which I thought it would. This caused the deletion to delete an item at the very top of the list and not the item that was selected from the visible part of the list. I tried to find a simple solution to get the correct position of the selected item but I didn't find a solution that was easy to understand. Due to this I resorted to initializing the list again by fetching all the notes again from the database and telling the RecyclerView to update the data. This is slower than my desired functionality of just deleting the specific note but it does the same thing.

Trying to figure out problems while working with the RecyclerView gave me a lot of insight into how the RecyclerViews work in general and to manipulate the items in the list. Additionally I learned some more about changing the layout of the app and how to make the app elements look better by editing the XML files to change their appearance.

17.3.2024

Today I thought of a better way to delete items from the list and implemented the change into the app. I didn't think of it before but now it seems pretty obvious that a much better way to delete an item from the list of Note objects is to search for the deleted note based on its unique ID and then delete it, after which you can notify the RecyclerView to check for changes. Searching for a particular object from an ArrayList was a bit more complicated than I thought, as the command "indexOf()" command requires you to match every

parameter of the object for it to find the exact object. The command to search the notes-list would then look like this: $notes.indexOf(new\ Note(noteid, message, username));$
However, this command wouldn't find the object even if all the variables were the same without adding an override function which would make this command work. I found the solution to this problem from this stackoverflow answer, where you needed to add an equals-function into the Note java-class file, which would check if every parameter matched the one which was searched for. The function I created is below.

```
@Override
public boolean equals(Object object) {
    if (object == this) return true;
    if (object == null) return false;
    if (getClass() != object.getClass()) return false;

    Note other = (Note) object;
    if(id != other.id) return false;
    if(!message.equals(other.message)) return false;
    if(!username.equals(other.username)) return false;
    return true;
}
```

In the documentation on indexOf() this function wasn't mentioned clearly and I don't quite understand why adding this overriding function fixes the issue, and I don't know what to search to understand the mechanics of @override. This solution made the app function how I intended it to which I am happy about. I didn't do much more this time and the project is almost finished. I might add the ability to edit notes if I'm able to find an elegant way to edit the text by clicking on the text in the RecyclerView.

22.3.2024

Today I finished up the project and did some final changes to make the app look a bit better like adding padding to to some elements to make the texts not stick to the very corner of the element, and then I added clearing the EditText fields when submitting the data to make it more usable. Additionally I learned how to make the RecyclerView scroll to the bottom of the list when adding a new item to the list which could be easily done with a single command. I then recorded the demonstration video and then finalized the GitHub repo to add the necessary information to finish this course.

I enjoyed working on the app and I ended up adding more features than I anticipated since I liked working with the editor and learning about mobile development. This course gave me a lot of good information about how development on mobile is done and I got a refresher on now to code in Java. I intend to continue learning about mobile development as I enjoyed creating apps in this course.