



ANDROID APP ASSIGNMENT PART B

Mabel Shufflebotham [mas236]
mas236@aber.ac.uk

Contents

Introduction.....	2
Software Design.....	2
Datasource Package	2
Model Package.....	3
Packages and Their Purpose	5
generalScreens Package	5
Logic Package.....	5
Navigation Package.....	5
Use Case Diagram.....	5
Testing.....	7
REST API.....	18
Features of REST	18
Examples	18
Reflection	19
What went well and what didn't go well in my project.....	19
Improvements.....	20
What was learned.....	20
Major UI Design Changes	20
Flair	21
Final grade	21

Introduction

To develop and assess my skills in android app development, I was tasked to create an app independently that provided students with an easy, simple way to revise of their exams. Previously, I had designed what this app may look like, detailing the target audience and their impact of each design choice. Following this, I began to put this plan into action, attempting to plan and program this app using a client brief. This brief had function requirements that I successfully implemented into the app that uses a Room database, Kotlin as a programming language, and Material 3 design elements [1] - though not without challenges.

This document details the process of implementing my UI design, discussing my project structure, testing, and exploring the idea of using REST API as part of my app. Finally, I reflect on my own performance, discussing what I achieved and what could be improved in the app

Software Design

Throughout the project, I aimed to keep the software's structure as simple as possible to allow efficient error isolation and fixing. The project uses a database, a view model, screens, and components used by the screens, as Figure 1 shows. The three key sections of the project is the viewmodel, the database, and the screens.

Datasource Package

I implement a Room database in my code that uses SQLite. As Figure 1 illustrates, the app has one database with two tables: *questions* and *answers*. *Questions* stores the attributes 'question' (the question used in the app for the quiz) and the 'questionId' (a primary key that is an autogenerated integer starting from 1). *Answers* is structured similarly, having a primary key 'answerId', a string attribute 'answer' (answer that can belong to a question), 'correct' (an attribute that can either be CORRECT or INCORRECT, and it also has an attribute 'questionId'. This questionId is what I use to provide a connection to the question and answer. The questionId that the answer stored in its table has a one-to-many relationship – one question can have many answers, but an answer can only have one question. Each of their attributes is taken from the Question and Answer object.

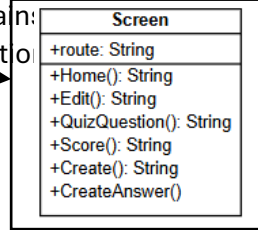
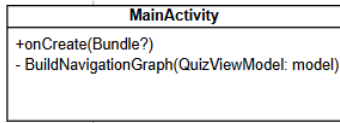
Both of these use their own DAO to store and use their own database queries, which I then call upon to populate the database at app launch (using `populateDatabase()`). Due to my lack of experience, rather than link questionId by a foreign key I just made several methods in my view model.

The database maintains some standard form throughout. It achieves first normal form (1NF) partially, though time and knowledge restrictions stopped me from trying to push this further. I ensure that each value is atomic and not empty before it gets inserted into the database, for example.

Overall, this package provides a connection between the view model and the database so that the database can be built within the view model at app launch, storing the questions and answers that are used in the app.

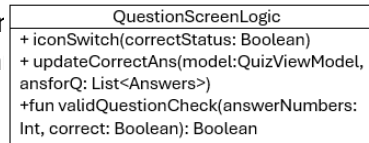
Model Package

BuildNavigationGraph
The model package contains:
Routes to screens

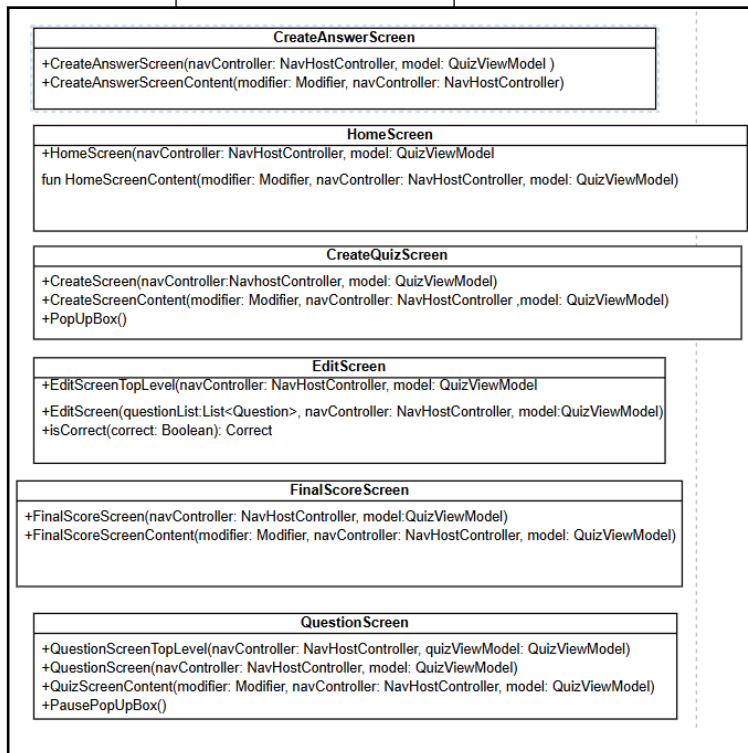


Gets logic
functions (which store
data in it belongs to) a
validation

Logic

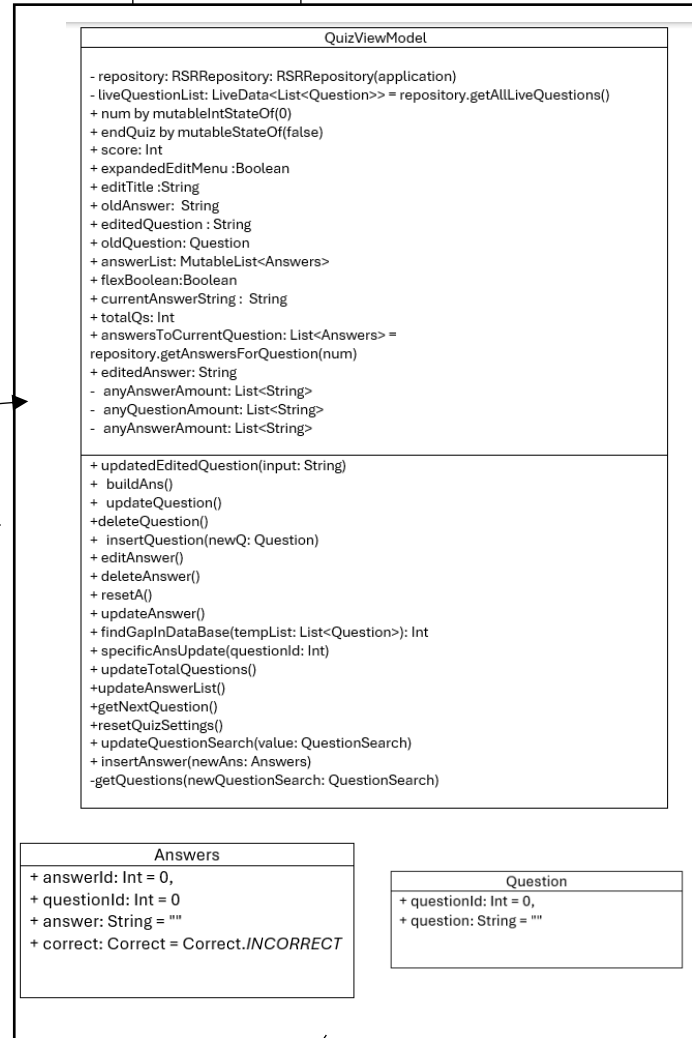


generalScreens

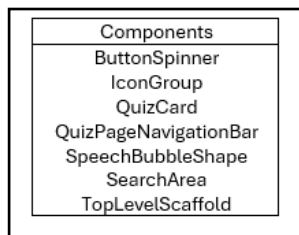


Loads
view
model
data

model



Screens use
components
for UI



Gets access to DAOS by using
the repository

Daos are used to get to
database model

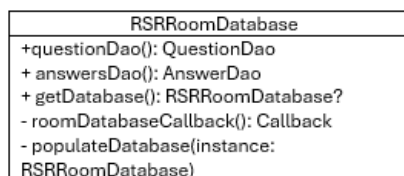
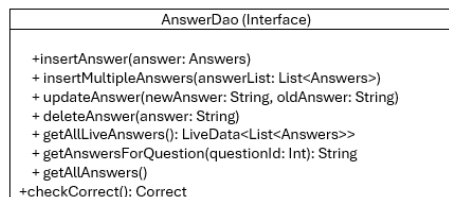
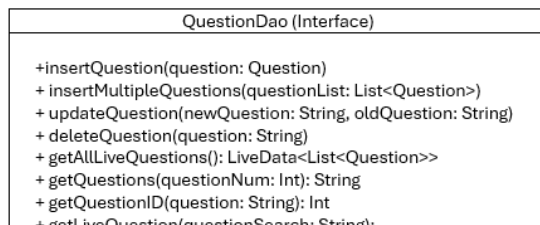
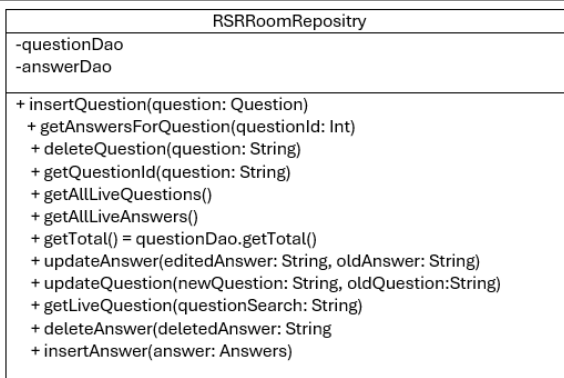


Figure 1: A UML Diagram of my app

the question and its id as a primary key). These are used to store the data for their namesakes, which then get inserted into their relevant tables. The model package also contains the view model for the entire app.

I initially considered doing separate view models for the quiz and edit/create/delete side of the app, but in the end, I stuck with one as this was simpler to implement and the fact that some code was having to be repeated as no clear line could be drawn between them. The view model stores all the variables and methods that would be used throughout the app or across several screens so that the code is less complex and more efficient.

Lastly, the model package contains an enum for correct, which the Answers object then uses in its database. I chose an enum as it is very simple to manage, and I already had a lot of familiarity using them.

Packages and Their Purpose

generalScreens Package

The general screens package stores all the screens that are used in the app. These screens access the components package for complex UI and Material 3 features. These screens are all navigated through by the MainActivity using a navigation graph to keep it simple and concise. Each class ends with the screen suffix to conform to Jetpack Compose standards.

Almost all of the screens use state hoisting to flow from lower-level functions (building the ui) to top level functions (database functions, LiveDataList declarations. I attempted to keep this standard throughout, though some lesser screens may not use this due to their simplicity.

Logic Package

The logic package intended to be much bigger than it turned out to be as I had made it as I had larger plans for my app that were never executed. It holds the class QuestionScreenLogic, which has a few validation methods that are called by the quiz screens when necessary.

Navigation Package

Like the logic package, navigation package holds one class. This is Screen, which stores each screen as a data object so that they can be referenced throughout the program. I implemented this method to allow easy access to screens that conforms to jetpack compose standards typically.

Use Case Diagram

To ensure that my software plan was not overcomplicated, I created a use case diagram that assess how a singular person would interact with the app (Figure 2). The user should be able to do one of three things in this app: edit the questions (answers included) in the database, create new questions in the database that are then used in the app quiz mode, and play a quiz round

A key factor in the assignment brief was to ensure that navigation is simple, using as little clicks as possible. Therefore, I made the conscious effort to reduce the amount of clicks the user has to make by making the UI in the edit/create quiz screens alter their appearance based on user inputs. The case diagram reflects this; they have only three buttons to pick from at the start, keeping it simple. I chose to make the delete functions/edit answer functions extend from the

'edit quiz' task, for example, because I associate them with each other – having them all as functions (edit answer, remove answer, edit question, remove question) stops the user from

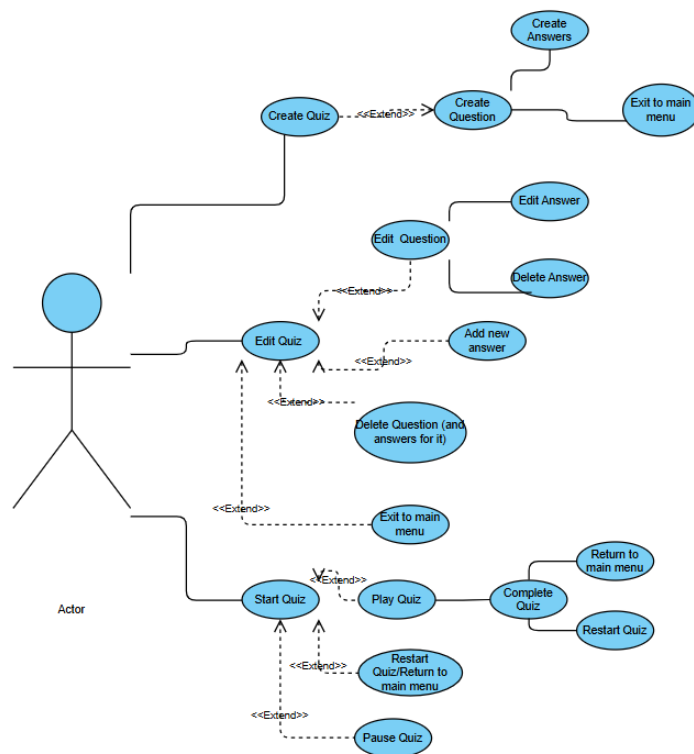
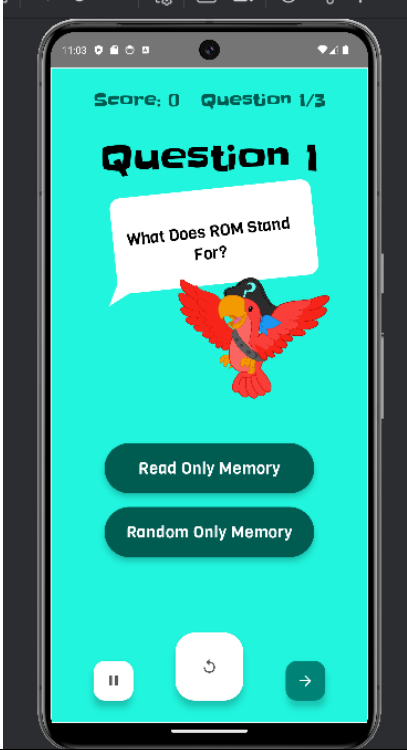


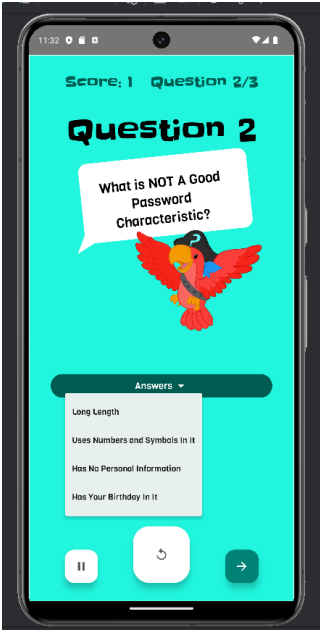
Figure 2: A use case diagram showing how the user would interact with my app

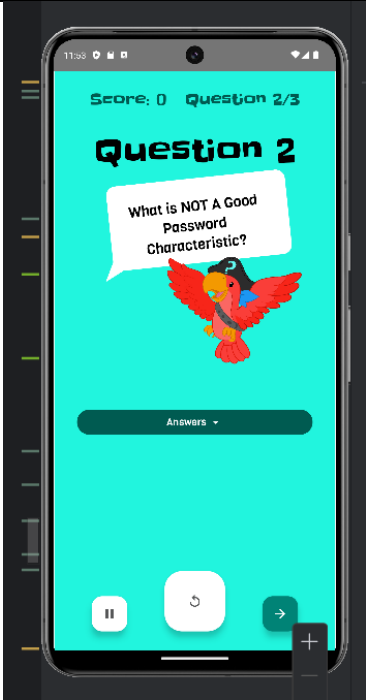
scrolling through too many screens and becoming lost/overwhelmed in the app. Therefore, I am of the opinion that my design choice is effective achieving the desired characteristics from the assignment brief.

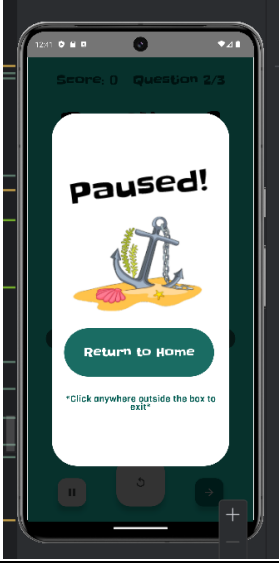
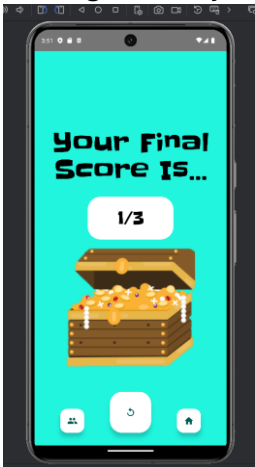
Testing



Though Android Studio provides automated testing, I could not get it to work due to my hardware being unsuited for the task, and a shortening time limit meant I could not spend much time investigating solutions. Therefore, I used a manual testing table to test my app's functionality and whether it can handle error cases such as not being able to submit an empty/blank question. The tests work through the app as sequentially as possible, first playing a quiz round, then editing some of the questions and answers, then repeating a quiz round to test if these edits are reflected in the quiz round.

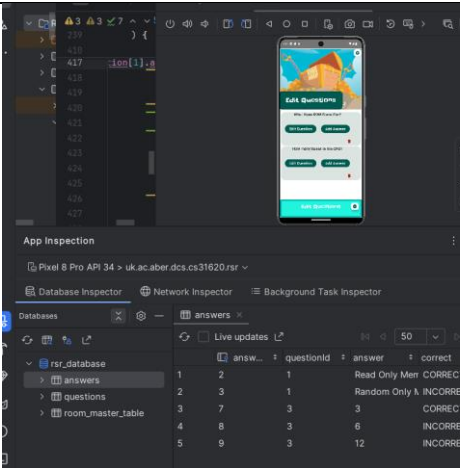
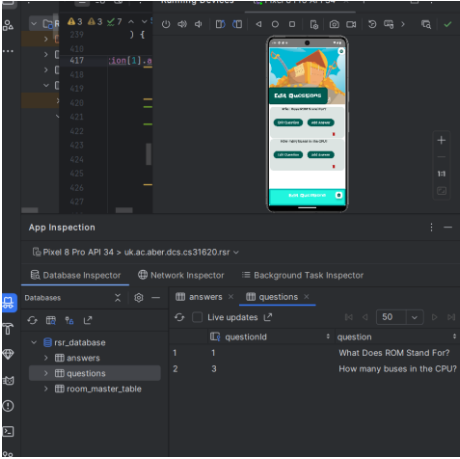
Test case	Action	Expected Outcome	Actual Outcome	Pass /Fail	Notes
1.1: clean launch	App launch/start app	Upon launching, the app should not crash and the home screen	The home screen is built successfully	Pass	
1.2.a: Start quiz mode	Select button "Let's get Revising"	UI should load successfully, dependent on how many answers are in the current question. If there are only two items in a list, the answers should be displayed on two buttons. Otherwise, they should be displayed as a drop down menu.	<p>The ui loads in correctly with all variables at their default values. The answers to the first question was displayed in two buttons with no issues regarding their text. The screenshot I use below confirms this:</p> 	Pass	
1.2.b: Loading question and answers	Select button "Let's get Revising"	Database is generated and filled with three default questions and answers.	Through the database inspector, I can see that the database has been generated correctly, as shown in the screenshots below. Answers:	Pass	

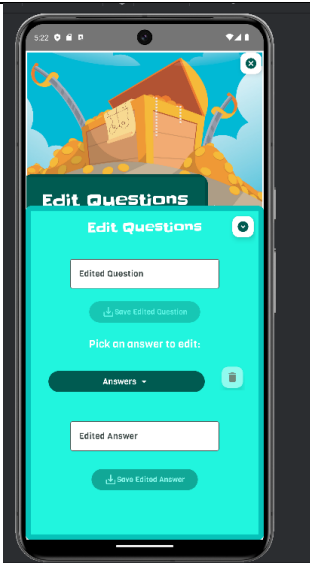
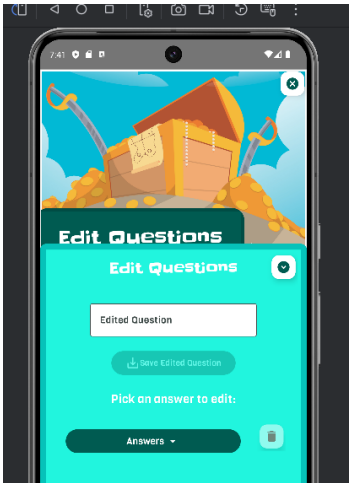

			<div><div><div><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div><div>9</div></div><div><div>1</div><div>2</div><div>1</div><div>2</div><div>2</div><div>2</div><div>2</div><div>2</div><div>2</div></div><div><div>1</div><div>2</div><div>1</div><div>2</div><div>2</div><div>2</div><div>2</div><div>2</div><div>2</div></div><div><div>1</div><div>2</div><div>1</div><div>2</div><div>2</div><div>2</div><div>2</div><div>2</div><div>2</div></div></div><div>Questions:</div><div><div>1</div><div>2</div><div>3</div></div><div><div>1</div><div>2</div><div>3</div></div><div><div>1</div><div>2</div><div>3</div></div><div><div>1</div><div>2</div><div>3</div></div></div>		
1.3: moving to next question	Select answer and press the arrow button (next question)	The UI will update itself depending on the amount of answers for the next question. In this case, the two buttons should reload as a drop down menu with the answers of the next question.	<p>The UI updates, switching quickly to the answers of the next current question and updating the current question, as the screenshot below shows.</p> <div><p>The screenshot shows a mobile app interface. At the top, it says 'Score: 1 Question 2/3'. Below that, 'Question 2' is displayed in large text. A speech bubble contains the question: 'What is NOT A Good Password Characteristic?'. Below the question is a list of four options under the heading 'Answers': 'Long Length', 'Uses Numbers and Symbols in it', 'Has No Personal Information', and 'Has Your Birthday in it'. At the bottom, there are three buttons: a pause button, a refresh button, and a next question button.</p></div> <p>Compared to the screenshot in test 1.2.a, this has updated its question, score, and the way the app looks when listing the answers.</p>	Pass	It is worth noting that once an option is selected from the drop down menu it does not clear its value when the next question occurs, which could confuse the user until they select the menu.
1.4: score system	Select answer and press the arrow button (next question)	If the selected answer was correct the score increases. Otherwise it should not changed	As already shown in the first screenshot, the score increases if you select the correct answer. This did not increase when I rebuilt the app and purposefully selected the wrong answer:	Pass	

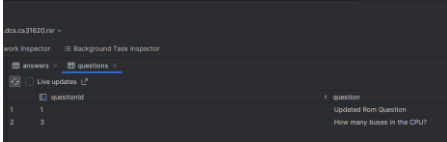
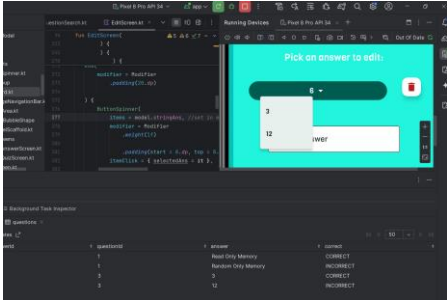
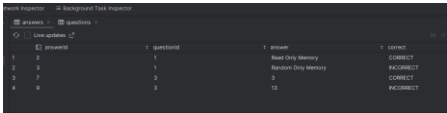
			 <p>(UI state after I clicked the incorrect answer – score remains as 0)</p>		
1.5: skip question	Select answer and press the arrow button (next question)	User should be able to move onto next question if an answer isn't selected, as they might want to skip it and study it later. If they do, the app should remain robust and continue as if they chose to skip the question.	The user can move onto the next question and the app does not crash/break.	Pass	
1.6: restart quiz	User selects the restart FAB button during the quiz round	The app navigates back to the home screen. When choosing to start a new game, the variables are reset so the user starts the quiz from the start again.	The quiz can be interrupted any time without consequence by selecting the reset button, navigating to home screen. Upon starting a new quiz session, it begins from the start again and plays without fault.	Pass	
1.7: pausing quiz	User selects the pause FAB button during the quiz round	A pause pop up box should appear that gives the option for the user to return to the home screen. Upon clicking outside the box the pause screen should exit (become disabled)	User presses the pause FAB and the overlay/menu for pausing appears, only exiting when either the user clicks the option to return to the home menu or clicks away from the menu. The screenshot below demonstrates this:	Pass	This does not actually 'pause' the screen behind it, but instead covers any interactable buttons. This was a workaround I did as I lack the knowledge of how I would


					achieve pausing a thread/state of a screen.
2.1: Final score display	All question have been answered in the quiz, and the next/arrow FAB is pressed again	Screen navigates to the FinalScore screen, which displays the final score the user achieved with an animation growing the box it is containing in.	<p>Once pressing the arrow FAB the final score of the quiz round is displayed with an animation running smoothly</p> 	Pass	Initially wanted the treasure chest to animate between two images, but I lacked the knowledge and time to do this, hence why it deviates from my UI design.
2.2: leaderboard navigation	The leaderboard FAB is pressed	App navigates to the leaderboard screen, which displayed a set of fake people with scores to beat.	The button is unresponsive to clicks/taps.	Fail	This is due to the unexpected shortness of time resulting in me not getting time to create this screen. The button, therefore, does not have a screen to navigate to.
2.3: end of quiz	The Home icon is pressed	App navigates to home screen, resetting all quiz variables until next round.	App navigates to home screen and, when launching another round, resets the variables so a new quiz begins.	Pass	There is also a restart button on the screen, but this has an identical function as

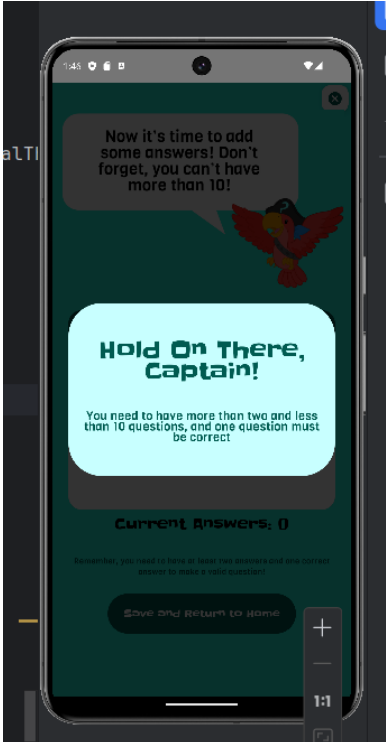
					what has already previously been tested
3.1 :UI build of the edit screen is successful	'Edit' button is pressed on launch of the app	App navigates to edit screen and loads UI successfully (the quiz cards in the scrolling list. All UI interacts with the user's actions correctly, such as scrolling behaviour.	<p>The app navigates to the correct screen with no errors, and the UI elements load correctly, as shown in the screenshot below:</p>  <p>The UI also interacts with the user correctly where desired – the quiz cards (each displaying a question and the options of how it can be edited) scroll:</p> 	Pass	The cyan box at the bottom is actually an expandable box that extends when the buttons are pressed to show their own menu due to my desire to reduce the complexity and amount of clicks my app needs to access all of its features. This successfully remained not expanded upon loading the screen.
3.2: delete question	Delete icon at the corner of each quiz card is pressed (for this example, the	The question is removed from the database, along with its associated answers. This change is instantly reflected in the UI, which removes this from the display	The question and all of its answers are deleted from the database correctly, removing the answers from the answers table:	Pass	

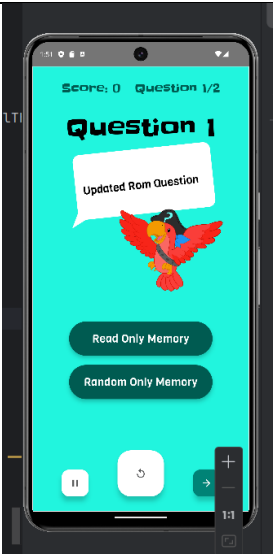
	<p>second question card (which of these is NOT a good password characteristic ?) has its delete/bin icon pressed</p>		<div><p>And the question from the questions table:</p><p>The UI reflects this change immediately, which can be seen in both screenshots.</p></div>		
3.3: edit question	<p>the 'edit question' button is selected</p>	<p>The cyan box at the bottom of the app should expand with a new menu which gives options on editing a questions and its current answers.</p>	<p>When 'edit question' button is pressed, the edit question menu is loaded and the box holding it expands to show the user this. This is excitable when the small fab in the top corner is pressed, as shown in the screenshot:</p>	Pass	<p>The contents of this menu will change if the 'add answer' button is pressed instead.</p>

					
3.3.a: database update for editing an answer	Edited question is submitted into the relevant text box and submits (clicks the button 'save edited answer')	If the textfield has any kind of string in it, the app will update the old version of the question so that the input is now the edited answer. This should be reflective in the questions table in the database. The edit question box menu shrinks.	<p>If the input in the textfield has letters in it/has value, the 'save edited answer' button is clickable. Otherwise it remains disabled:</p>  <p>If there is a value in the 'Edited Question' box then button is disabled:</p>  <p>Synchs up these changes in questions table:</p>	Pass	

					
3.3.b: deleting answer	<p>User deletes an answer to the current question being edited by selecting the desired answer and then pressing the small FAB delete button beside it.</p> <p>In this example, user tries to remove '3' from being an answer for the question 'how many buses in the CPU?'</p>	<p>The answer selected in the drop down menu is deleted from the database. If the user has not made a selection, the delete button should remain disabled and change when the state of the selected answer changes.</p>	<p>Answer selected (3) is removed from the database, and this changed is immediately synched to the rest of the program.</p> <p>The answers table after deletion:</p>  <p>The change in state of the delete button can be seen in previous screenshots in tests 3.</p>	Pass	
3.3.c: editing an answer	<p>User edits an answer ('12' from 'how many buses in the CPU') by typing new answer in the final text field once using the drop-down menu to select the answer they want to edit. They press 'save edited answer' button</p>	<p>The selected answer in the drop-down menu is edited, overwritten in the database with the new input as the answer option to the question. If there is no input in the text box then this will not occur, as the button becomes disabled</p>	<p>The answer '12' is edited to '13' and the database updates to reflect this change correctly, as shown:</p> 	Pass	
4.1: creating a quiz question	<p>From home screen, click 'Create' to create a new quiz question that is used in</p>	<p>If the user input has a value (if it wasn't left blank) then the page navigates to the create answers screen.</p>	<p>The create screen, when the 'create' button is pressed, navigates to the screen to the create answer screen</p>	Pass	<p>There is also a 'never mind' screen which navigates back to the home screen</p>

	the quiz mode. Enter a new quiz question and press submit				
4.2: submitting an empty new question	User attempts to create a new question but does not enter a question in the text field	If the input is empty then the create screen will not navigate to the next screen, displaying a pop up to prompt the user to enter a valid question.	<p>When the 'Create' button is pressed a pop up loads, as shown below. The screen does not navigate to the create answers screen until a valid question is entered.</p> 	Pass	
5.1: creating a new incorrect answer	Navigate to create answer page. Types a new answer for the new question in the designated textfield and saves the answer with the button 'save answer'	A new answer is created and stored temporarily which is 'incorrect' The textfield should clear, allowing more submissions, and the condition of the current answers being added is updated.	New answer submitted and the text field is reset. The count of the current answers increases by one.	Pass	
5.2: creating a new correct answer	The same process as explained in the previous text. Before submitting, the green 'mark answer as correct	The 'mark answer as correct' button becomes disabled to prevent any more correct answers being made (validation).	Same as previous test	Pass	

5.3: saving the new question to the database	Makes a new question with answers. Presses the 'save and return to home' button.	The question and the answers are saved to the database. App navigates to home screen (only if validation is passed)	The question and the answers are saved to the database and navigates to home screen (only if validation is passed)		
5.4: validation for submitting a new question	Submits a question that doesn't have a correct answer, or doesn't have over two answers, or has more than 10 answers.	The invalid data will not be submitted to the database and a popup box loads to inform the user of the issue	<p>The invalid data was not submitted to the database and a popup box loaded to inform the user of the issue</p> 	pass	
5.5: submitting a blank answer	Create new answers for new question. Submits the blank answer	Should not be able to submit the answer at all as the button is disabled when there is no string in the textfield	Could not submit question until it has value	Pass	
6.1: ensuring changes made in edits/create/delete are applied to the quiz mode	Starts a new quiz session after editing the database	After editing/creating/deleting elements from the database, these changes should be reflected in the UI and quiz round	Each change made in the previous tests in this table are in subsequent sessions of the quiz mode, as the screenshot shows below:	Pass	

			<div data-bbox="775 190 1048 741">A screenshot of a game interface on a smartphone. The screen has a teal background. At the top, it says 'Score: 0' and 'Question 1/2'. Below that, 'Question 1' is written in large, bold, black letters. A white speech bubble contains the text 'Updated Rom Question'. In the center is a colorful parrot. Below the parrot are two dark green buttons with white text: 'Read Only Memory' and 'Random Only Memory'. At the bottom, there is a control bar with a pause button, a power button, a right arrow button, and a volume slider. The status bar at the very bottom shows '1:1' and a small icon.</div>		
			<div data-bbox="775 741 1209 889"><p>Notably, the question number has changed after the removed question, and the edited ROM question is on display</p></div>		

REST API

A REST system could be introduced as part of my project if I wanted to create a separate web server that, instead of local storage of data, gives persistence outside of the app itself with flexibility.

Features of REST

I would consider using REST as part of future development/a different approach in my project due to the six constraints they have (client server architecture, stateless, cacheable, uniform interface, layer system and code on demand). Most notably, the stateless characteristic they offer means that they would not have to store old client requests in the server, which would make the performance of the app increase with the smaller amounts of data transferred between the client and server in comparison to persistence.

Not only this, but their flexibility/portability (as they use standard HTTP protocols that many things on the internet use) paves the way to the revision app becoming easier to access on multiple platforms. As this app is marketed at every kind of student, this feature could increase its reach to many more devices.

Lastly, the implementation of REST provides an online persistent database, allowing a database to not have to be stored locally on the device. Throughout my project, I struggled with the performance of the emulator on my hardware, so this is a feature I would be keen to use in this hypothetical scenario

Examples

Some of the key functions of my app, such as removing and reviewing data, have been given as an example in REST below. I have attempted to make these as close to what they would be for my app.

This GET method aims to receive a JSON file of a answer that all have a question id of 1, like how I start each quiz session in my app. It returns a collection resource.

GET /RSR.com/ answers/2

Sending headers: Accept: application/json; q=1.0,

application/xml; q=0.8, text/;q=0.1*

Received headers: Content-Type: application/json

Content received: JSON or XML or plaintext representation of the resource

Status codes: 200 OK, 404 Not Found If exists, receives JSON file representing workout with an id of 2

In addition, I use the 'delete' option a lot in my code to allow the user to edit data or removing questions from the database. An example of this in REST could be the below snippet of code – of exists, delete the question that has a questionId.

DELETE /RSR.com/question/3

Sending headers: Accept: application/json; q=1.0, application/xml; q=0.8, text/; q=0.1*

Received headers: Content-Type: application/json

Content received: JSON or XML or plaintext representation of the resource

Status codes: 200 OK, 202 Accepted, 204 No Content

Reflection

What went well and what didn't go well in my project

Throughout the project, I experienced many challenges, some of which I managed to overcome. As a whole, I believe I succeeded in programming the core functionality of my app despite initial struggles that I mention in the next subsection. Due to my experience in other programming languages, I was able to pick up Kotlin quite easily in regard to its syntax and most of its logic, so the front end development came with little challenge. However, I did encounter my first major struggle the use databases and their queries using Room – more specifically, the use of LiveData objects over regular list objects.

For a large majority of time in my code, I had issues with getting the app to run due to the view model having too much work in the main thread. It was difficult for me to solve this issue, though I eventually found out that this error had occurred due to my app running database queries without using LiveData, which in turn was too much to run in the main thread and caused crashing. Being the largest problem during development, I consider it a success that I managed to fix this issue and provide full functionality of the app (at the sacrifice of production time for the rest of the app).

Aside from this issue, the coding was quite easy once I became more experienced with the language. I was able to complete all the functional requirements from the assignment brief, storing data in a database and providing a functional quiz app for students to use. The use of Material 3 greatly aided me and my performance, as I was able to make components of UI elements, set a house style, and use Material 3 libraries to build my app. In particular, I am proud of how I manage to change the answer layout depending on the amount of answers to the current quiz question in the round, dynamically changing this from two buttons (if there are only two answers in the question) and using a drop down menu; all of which use Material 3 components.

It is worth noting, in contrast to this success, that the lack of time I had caused me to not complete full production of the app, leaving out screens from my prototype and leaving out some validation on some areas such as stopping the user from trying to answer a question without selecting an answer. Also, the unexpected time crunch caused me to abandon fixing minor UI glitches. For example, the drop-down menu does not clear its displayed answer in the quiz section of the app when the question moves on until you reselect the drop down, which could confuse users. However, I remain strong in my belief that I should provide app functionality over refinement, so I do not regret taking more time to fix the major error I have previously explained.

Unexpectedly, I was surprised by the ease of using database queries in my code. It made processing smoother by using DAOs and a repository (declared in the view model) to retrieve

specific groups of data from my database, having programmed all as @Query to specify what data I wanted at key points in the app.

Overall, though I am disappointed with myself for not completing the app entirely and missing screens that I had originally planned, I believe I had many successes in production of the UI elements. The main areas I felt challenged in, and therefore I know can be improved, were the back-end development and time/ Despite these challenges, the app is functional and provides all of its user requirements, which is a priority to me and so I was pleased I achieved all client requirements.

Improvements

To improve the production of this project, I would first emphasise the importance of time management. I have spent many hours creating this code, fixating on errors in the program, and that has caused me to not fully complete the prototype to my personal standards. Also, the rush due to this time pressure caused less neatness and efficiency in my code at the later stages. I can guess that there are more efficient ways that my code could be written and structured.

Most obviously, the main way I would like to see this project theoretically develop going forward is the inclusion of validation of user inputs and the implementation of the screens in my UI prototype going forward. In many areas there are actions that require more validation – for example, not allowing the user to delete answers to a question if it only has two left, as this is the bare minimum to build a question – but I simply ran out of time. Future improvements would involve getting others to test my app to try and break it, using this data to program more validation cases to keep the app robust. Right now, I acknowledge that the app is not as robust as I would like, and so further development would theoretically focus on fixing this.

Overall, it is clear to me that the main improvement is to better manage time/have more time in the project, as this would trigger all the other improvements I have discussed. I had many ideas in my prototype that I could have included but I had to cut back the scale of my project due to the time fact.

What was learned

The key learning I took from this experience was the use of database models, hoisting, and using threads in Kotlin. I spent significant time researching these concepts as they were critical to efficient code. This studying allowed me to use a view model to run database queries as well as store items as a global state hoisted from the main activity – knowledge that I will surely use later in other projects such as my final year project to create a better app.

Major UI Design Changes

As a whole, I felt that I have stuck close to the UI designs prior. This is largely due to Material 3, which allowed me to recreate my material 3 theme and provided me a vast library of methods that I could build the content of each screen on. I also used the same image assets as I had done in my prototype. However, a major difference between the two projects is the cut down of screens and animations in my final product compared to my design document. These were both caused by the same factor: time. I prioritised meeting the functional requirements of the assignment brief, sacrificing marks I may have gotten in flair to dedicate all my time to the core functions of my app.

In addition, I found that I had a few ideas that were a bit too ambitious for my skill set, such as advanced animations on shapes and between screens. Though I implemented a few animations in my product, such as a fade in at launch into the home screen, lots of them remain unused as they were not necessary to the app's purpose and I had to prioritise other tasks first therefore.

Flair

Unfortunately, due to my issues with time management, I do not think I managed to produce anything that would classify as flair. I initially planned to create a false leaderboard that would generate false people to content with on the app. In my project, I do implement more trivial design aspects such as animations between screens and on some boxes, such as in my Final score screen.

Final grade

Overall, I would give myself a grade within the range of 50 – 55% in this part of the assignment. This is largely due to the incompleteness of my app and the rather messy structure of my code. I am aware there are better solutions than what I have provided, but the lack of knowledge and time has resulted in me unable to explore these options and prioritise programming an answer which is 'good enough'. Despite the setback, I have worked hard to find alternate ideas, some of these working well in exchange. I also complete all functional requirements in the assignment brief and create a database to store data.

There is also the issue in the fact I am not confident in what REST can do for my application. This section of the document is lesser in quality and content therefore, and so I have to take into account the drop in grade I may achieve as a result.

When taking these factors into account, and assessing the quality of this report, I can conclude that the grade I estimated for part B of this assignment is accurate with what I have produced.