



iMProve

ABRSM Music Revision System

Kevin Cen

Prior Pursglove College

OCR A Level H446/03
Programming Project 2018

Contents

Analysis	1
Description and Justification of Investigation.....	1
Stakeholders.....	3
Direct Stakeholders	3
Indirect Stakeholders	3
Research	4
Description of Existing Solutions	4
Comparison	10
Potential Issues	11
Conclusion	12
Features of Proposed Solution.....	12
Essential Features	12
Desirable Features	13
Limitations	14
Requirements for Solution.....	14
Software Requirements	14
Hardware Requirements	15
Success Criteria.....	16
Functional	16
Non-functional	18
Design	19
Decomposing the Problem.....	19
Problem Decomposition and Step-Wise Refinement	19
Full Problem Decomposition Hierarchy	37
Structure of the Solution.....	38
Structure of Data Flow between Technologies	38
Structure of Database	39
Structure of Important Algorithms	40
Usability Features.....	41
Grade Selection Screen	41
Welcome Screen	43
Learn	46
Test	53
Listen	59
Review	63
Key Variables and Structures.....	64
Main.java	64
DAO.java	65
GradeController.java	65

RootController.java	66
LearnController.java	68
LearnRowController.java	70
LearnUpdateController.java	72
LearnUpdateRowController.java	74
LearnFlashcardController.java	74
Flashcard.java	76
TestController.java	76
TestQuizController.java	77
TestResultsController.java	80
TheoryQuestion.java	82
Feedback.java	83
ListenController.java	83
ListenQuizController.java	83
Algorithms.....	89
Main.java	89
DAO.java	89
GradeController.java	90
RootController.java	91
LearnController.java	93
LearnRowController.java	94
LearnUpdateController.java	96
LearnUpdateRowController.java	98
LearnFlashcardController.java	99
Flashcard.java	101
TestController.java	102
TestQuizController.java	102
TestResultsController.java	108
TheoryQuestion.java	110
Feedack.java	111
ListenController.java	112
ListenQuizController.java	113
AuralTest.java	114
ReviewController.java	115
Test Data for Development.....	119
Grade Selection	119
Welcome Screen (Root Screen)	120
Learn Activity	120
Test Activity	122
Listen Activity	122
Review Activity	123
Test Data for Post-Development.....	124

Test 1	124
Test 2	128
Test 3	129
Development.....	131
Creation of Project.....	131
Iteration 1.....	132
Main.java	132
DAO.java	135
GradeController.java	136
RootController.java	142
LearnController.java	147
LearnRowController.java	154
LearnUpdateController.java	157
LearnUpdateRowController.java	164
LearnFlashcardController.java	165
Flashcard.java	170
Validation Test	171
TestController.java	176
TestQuizController.java	177
TestResultsController.java	187
TheoryQuestion.java	195
Validation Test	196
ListenController.java	198
ListenQuizController.java	200
AuralTest.java	206
Validation Test	207
ReviewController.java	210
Validation Test	224
Chart.css	226
Progressbar.css	228
Slider.css	229
Tabpane.css	229
Scrollbar.css	230
Summary Review	235
End of Iteration Validation Test	236
Iteration 2.....	243
LoginController.java	243
Learn Adjustments	247
Test Adjustments	249
Listen Adjustments	250
Settings Page	266
End Of Iteration Validation Test	297

Annotated Modular Code.....	307
Main.java	307
DAO.java	308
Flashcard.java	310
Feedback.java	311
AuralTest.java	312
LoginController.java	313
GradeController.java	314
RootController.java	315
LearnController.java	320
LearnRowController.java	322
LearnUpdateController.java	324
LearnUpdateRowController.java	327
TestController.java	328
TestQuizController.java	328
TestResultsController.java	333
ListenController.java	336
ListenQuizController.java	336
ReviewController.java	341
SettingsController.java	346
Evaluation.....	350
Testing	350
Test 1	350
Test 2	371
Test 3	377
Fixing the duplicate answers problem	383
Additional Test 4	387
Evaluating Solution including Usability Features.....	394
Evaluating Maintenance.....	398
Positive Signs of Maintenance	398
Possible improvements for maintenance	400
Limitations and Potential Improvements/Changes.....	400
Limitations and Improvements	400
Extensions	401
Conclusion	401
Other	402
Bibliography.....	402
Appendix A - Interview with Jill Thirkell.....	403

Analysis

Description and Justification of Investigation

Practising for music theory and aural testing has become quite difficult. Music theory like basic vocabulary memorisation can be eased by fun, engaging ways of memorising such as paper flashcards but the materials required may not always be readily available and with mass amounts of vocabulary, it sometimes could be very costly. Practising for aural testing is even more tricky and ambiguous, there is no concrete way for you to practise so you can either wait till your next music lesson or rely on playing a concept yourself – however this often defeats the purpose of aural testing as you already know what you are playing and cannot practise identifying the concepts by sound.

My piano tutor, Jill Thirkell who has been teaching since 1972 around various places in the North-East of England, finds that it is hard to help her students practise for music theory and aural testing where she can only teach them techniques in lesson but is unsure on advice on how they can practise those techniques outside of lesson. Her current methods use books for testing and concrete metaphors to describe abstract concepts such as time signatures. However, this is not quite effective as the exam requires no knowledge of the piece of music at hand – if they play it, they know it. Additionally, the students say that it is not very motivating and Ms. Thirkell agrees on that fact as she has identified there is lack of practise involved with those areas: theory and aural testing. This method is very costly as Ms. Thirkell is required to buy a constant stream of books per student – even when ineffective!

With the power of technology, the fun in music can be evoked and the motivation will naturally come through with the enticing aspect of it. Using game-like features, this will enhance the experience of music and emphasise how doing music is for fun rather than something you must do.

My project will be directed towards the ABRSM exam board covering multiple grades, allowing for a wider user group. The current theory syllabus includes:

- Note names and values
- Time signatures
- Clefs
- Major and Minor Scales
- Triads and Chords
- Key signatures
- Other frequently used terms

Aural testing includes:

- Identifying cadences and chords
- Identifying modulations
- Identifying features of a musical piece
- Sight-singing and melodic repetition

The music theory aspect includes lots of terms and tests the capabilities of human memory whereas the aural testing side tests the ability of sound recognition. With practice, these abilities can be honed but there are not many options to even practice in the first place so computation will hopefully fill that gap.

Most of the music theory syllabus is remembering terms and commonly used phrases such as ‘con moto’ ; with something that can aid the process of learning these terms, and testing to consolidate knowledge, success in music theory is almost guaranteed. Terms can be stored in a database and listed on a UI with also a form of learning technique to learn these terms can be applied – such as a flashcards system. Alongside this learning aspect, unlimited testing on the knowledge can be included where multiple questions regarding terms can be displayed and answered; the results can be recorded and analysed – possibly used to adapt next test to further aid the user to improve. With this, you can justify computerisation as improvement can be targeted and specific to the pupil at-hand but also, it will be more fun and motivating as you are always learning something new rather than the same thing over and over again – however, old knowledge should still be occasionally tested so not forgotten. Some of the theory syllabus requires the user to actually look at a musical stave and identify features from there such as key signatures and clefs. An external library can easily allow the GUI for this and the questions and answers can be similarly stored in a database and analysed to adapt future questions. Having this system can be justified by the substantial saving on time and expenses of buying all the books of questions testing the exact same ideas.

In the aural aspect of music, sound is required to be played and a question would need to be displayed to test the user’s knowledge with outputs of whether right or wrong will be displayed afterwards. This simple output could drastically improve a student, as correcting recognitions with which one was correct allows the user to, over time, recognise the sound and link it to the correct feature. These questions can be stored on a database or possibly within the program executable itself. Use of a computerised system is justified by allowing practise for aural testing without needing to play it yourself, the advantage of this encourages a more exam-like environment, where you can only listen and not play the music, which greater benefits the student using it.

Furthermore, the progress will be recorded and displayed on a graph to show to parents the dedication their child has put into music and make them reassured for their consistent practice and the child’s growth and improvement. Also, seeing these graphs can alone motivate the students by making them want to complete today’s test and watch their score grow. This motivation in learning is an important aspect Ms. Thirkell (Piano Tutor) wants her students to have and computerisation can hopefully facilitate that.

All of the features of music theory and aural testing can be computerised by careful analysis using decomposition, pattern recognition and abstraction before designing, implementing, testing and evaluating algorithms. However, due to time and financial constraints, it is unlikely that all features will be added in the first edition of the system.

Stakeholders

My project will include various direct and indirect stakeholders which should all benefit from this program in some way.

Direct Stakeholders

Jill Thirkell: possibly the main direct stakeholder who will receive the product and use it for her teaching. She is a music teacher in Skelton-In-Cleveland who has taught for 46 years since 1972 and still enjoys teaching music today to students ranging from 5 years of age to elderly in their 80s. If it is successful, Ms Thirkell can deploy it for students to use or maybe use it in lessons with her students – improving her ease of teaching. This will allow her students to keep in practise their music theory and aspects of practical, aural testing, which will make sure her lessons are always effective and learning something new rather than revising over old topics. Although she is not the final user that will use this system, Ms Thirkell is delighted to be participating in this project and has agreed to contribute fully to the development process including all the testing and feedback parts.

Rebecca May: a contemporary student of Jill Thirkell, who is in year 11 and studying GCSE music. She is eager to try this new program out as she believes new materials to boost her music theory knowledge will be greatly beneficial to her GCSE grade and her future use of music. Rebecca will be willing to test out the features with the perspective of a student meaning she can possibly relate to real life situations and compare and contrast the benefits and advantages, each of which will be beneficial to note to improve our system. With her student expertise, she can try all the features which will teach and test her allowing her to identify which features she believes to be most useful/useless to help the development of this new system as a direct stakeholder.

Robert Woodrow: a prior student of Jill Thirkell, and now a direct stakeholder of this system, one who has studied music up to ABRSM Grade 5 theory and Grade 6 cornet. He has the knowledge of music theory comparable to a high class standards and the aural testing aptitude to fully test out the program's capabilities with the perspective of an expert student.

Indirect Stakeholders

Parents of Students: these are stakeholders who can but do not have to, be involved with this system. There will be a way for parents to have a look at the progress of their child but this is not a necessary action as students can look at the progress themselves. But indirectly, if the student is achieving higher successes with music, then naturally, the parents would be proud and then the system would have made a positive effect.

Peers of Students: an indirect group of stakeholders. If the system is effective, the current pool of student testers may refer this program to other peers who may, or may not, use this new program to help with their own musical development. Using this program should also bring out the fun in music and hopefully will encourage new students to take part in music.

Other music teachers/tutors & students: if the system is recognised enough, there is also a small chance that other teachers or tutors may implement this system to their teaching and help

a vaster selection of students around the area – indirectly benefitting the entirety of the music community as a whole.

Exam Boards: if the system becomes more widespread, a higher percentage of students will use this system and possibly change the overall skill level of the students taking the music exams (mostly ABSRM). This could lead to future changes to their examinations or marking to adapt for the overall rise in skill.

Research

My project will comprise of features from music theory and aural testing. The following parts will describe and explain the current solutions to either of these problems.

Description of Existing Solutions

MyMusicTheory

A quiz type solution available online via browser (mymusictheory.com).

Grade 3 Music Theory Terms

The screenshot shows a quiz interface. At the top left, it says "Completed" with a progress bar filled to 100%. Below that is the text "Page 10 of 15". The main content area contains a question: "Question 10 What does *pesante* mean?". Below the question are four multiple-choice options, each with a radio button: "Choice 1 With feeling", "Choice 2 Rhythmically", "Choice 3 Heavily", and "Choice 4 In a pastoral style". At the bottom of the interface are two buttons: a blue "Submit" button and a grey "Abandon Quiz" button.

Clearly displays necessary information like question and answer text including the progress through the quiz at the top. Uses multiple choice to display the questions by radio buttons and text. However, lacks the modern feel and is quite simple lacking the enthusiasm desired for an audience of younger age. Quizzes are separated by grade levels and tests on common terms although it lacks another approach to learning these foreign terms.

Nice work - you passed the test!
You scored 75%
Take another quiz...

Question 2 What does *giusto* mean?

Choice 1 Animated
Choice 2 Correct
Choice 3 Gusty
Choice 4 Only

The answer is incorrect

Score is 0.00 out of 1.00. Elapsed time is 11 sec.

#	Answer	Correct answer	Your answer
1.	Animated		<input checked="" type="radio"/>
2.	Correct	<input checked="" type="checkbox"/>	
3.	Gusty		
4.	Only		

Question 4 What does *vite* mean?

Choice 1 Tempo at the player's choice
Choice 2 Slow
Choice 3 Moderate speed

After the quiz, there are outputs of percentage correct (result) and also places of error – it shows the mistake made and the correct respective answer.

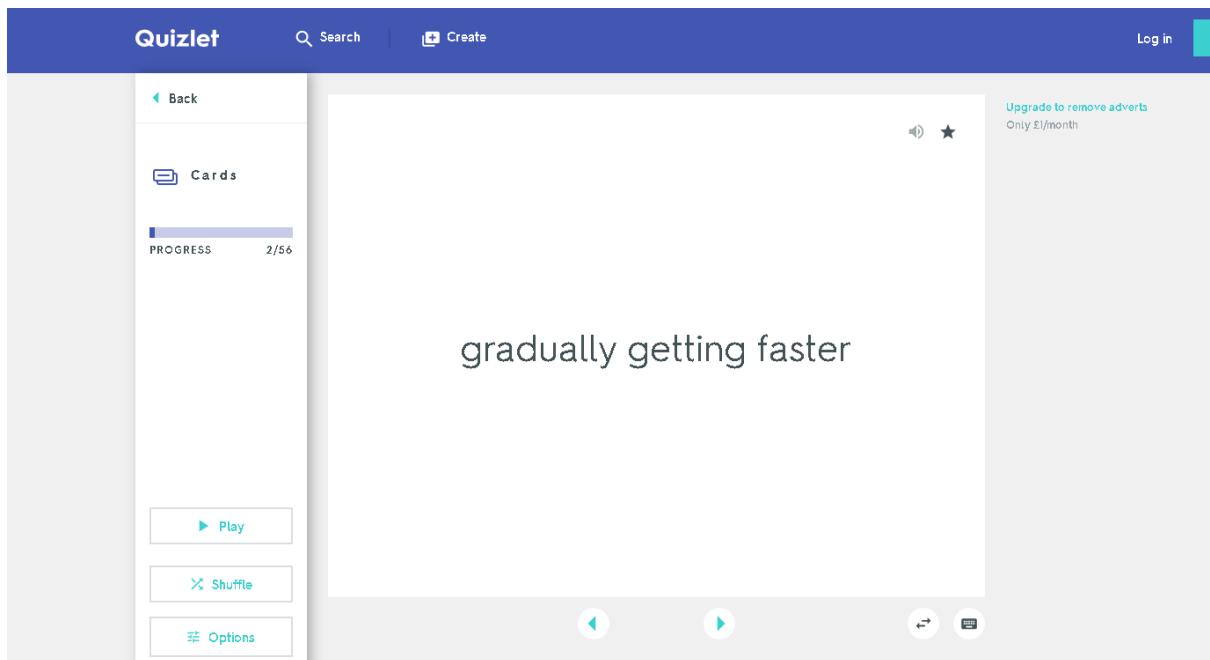
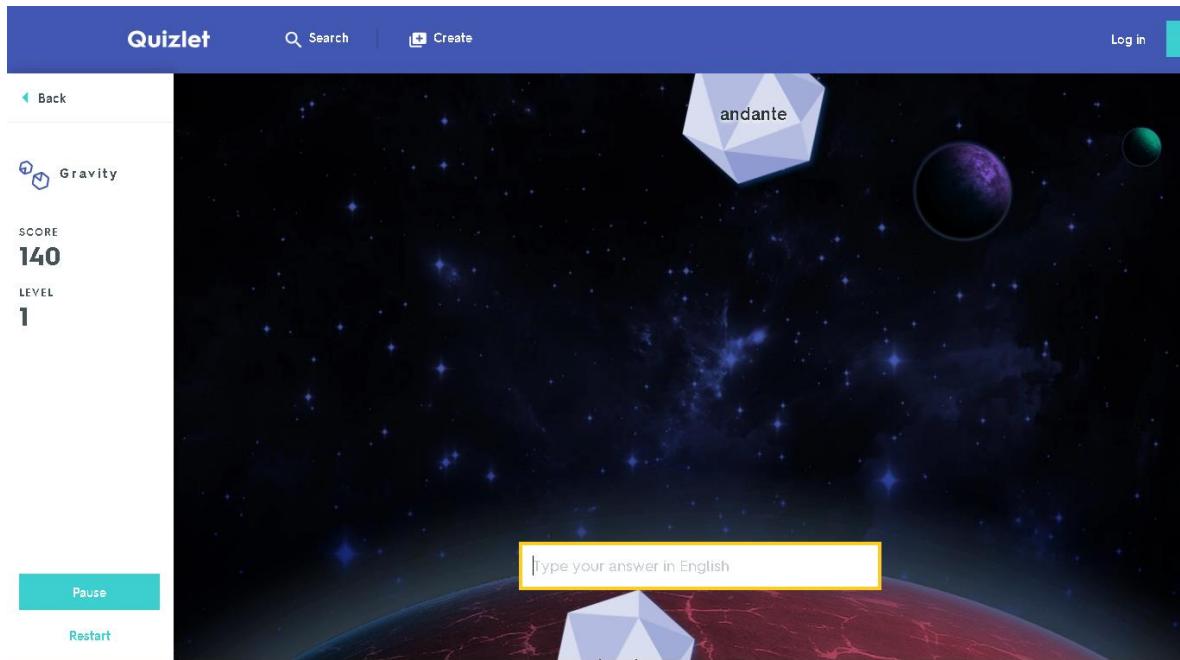
Overall, quite it is a limited learning approach but has a detailed post-quiz screen, displaying improvements and corrections.

Quizlet

An interactive games approach available via mobile app or online by browser ([quizlet.com](https://www.quizlet.com)).

The screenshot shows a Quizlet Match game interface. At the top, there is a blue header bar with the Quizlet logo, a search bar, a 'Create' button, and 'Log in' and 'Sign up' buttons. Below the header, on the left, there is a sidebar with a 'Back' button, a 'Match' button, and a 'TIME' section showing '12.5'. The main area is a grid where words are matched with their definitions. Some visible matches include:

- Tonic (top right)
- Downbeat (center top)
- 1 (center)
- Dynamics (right side)
- piano, forte (below Dynamics)
- levels of loudness or softness (below piano, forte)
- BEADGCF (center bottom)
- Flats - Order of accidentals (left side)
- first beat in the measure (bottom right)
- Italian indications (bottom center)



This solution provides a variety of approaches to learning and all clearly demonstrate the simple, necessary functionalities to learning these terms such as displaying the term and then the definition when asked for. Includes interactive, modern UI appealing for younger audiences. Word sets are custom made by the community and can be customised to your liking.

Terms In this set (56)

Often Missed Your recent answers have been mostly incorrect! ★ Select this one

-1 tenuto	connected, full value	★ Select this one
--	-----------------------	---

Always correct You've answered all of these correctly! ★ Select this one

+1 poco	little	★ Select this one
--	--------	---

No Answers Yet You still haven't studied these! ★ Select these 54

? a tempo	return to the original tempo	? Select this one
? accelerando	gradually getting faster	? Select this one

Additionally, it provides feedback onto answers from previous games which is not too verbose or difficult to comprehend – using a game-like points system.

Generally, a very complete solution providing plenty of engaging approaches to learning and has an interactive, modern and professional UI.

Hofnote

An aural testing solution available online via browsers (hofnote.com).

The screenshot shows the Hofnote website interface. At the top, there's a navigation bar with links for Home, Register, Courses & Demos, Terms of Use, Testimonials, Links, Buy Sheet Music, FAQs, About Us, and Contact Us. Below the navigation is a sign-in section with fields for email address and password, and buttons for Student Sign in, Forgotten?, and Teacher/School Sign in. A quote by Katie Greenwood is displayed: "Music isn't just learning notes and playing them, You learn notes to play to the music of your soul." - Katie Greenwood. The main content area is titled "Student practice". It displays an exercise titled "Exercise Number: 8-C1" and "Exercise Name: Get that Modulation!". It asks the user to listen to a clip and decide which key the music modulates to. The question "What is the new key?" is followed by five options: 1. Dominant, 2. Subdominant, 3. Relative minor, 4. Dominant minor, and 5. Relative major. To the right of the options is a playback control bar showing a play button, a progress bar at 0:00 / 0:00, and a volume icon. At the bottom of the exercise area are buttons for "Show Me The Answer" and "Submit Answer". The footer of the website includes the Hofnote logo, links for Terms & Conditions, Privacy Policy, Consent Form, and Music Teacher Support Ltd, along with links for Contact and Address Details.

A simple multiple-choice quiz approach which also includes playback of music audio that is needing to be analysed. Question displayed in the middle of the screen with multiple-choice

answers below. Has the option to only show the answer or submit it to check your answer with the correct.

The screenshot shows the Hofnote website interface. At the top, there is a navigation bar with links for Home, Register, Courses & Demos, Terms of Use, Testimonials, Links, Buy Sheet Music, FAQs, About Us, and Contact Us. Below the navigation bar is a sign-in form with fields for email address and password, and buttons for Student Sign in, Forgotten?, and Teacher/School Sign in. A quote by Katie Greenwood is displayed: "Music isn't just learning notes and playing them, You learn notes to play to the music of your soul." - Katie Greenwood. The main content area is titled "Student practice". It displays an exercise titled "Exercise Number: 8-C1" and "Exercise Name: Get that Modulation!". It asks the user to listen to a clip and decide which key the music modulates to. A question "What is the new key?" is shown with the correct answer "Relative minor". The user has answered 1 and got 0 right. There are buttons for Try Another, View Results, and Finished. At the bottom, there is a footer with the Hofnote logo and links for Terms & Conditions, Privacy Policy, Contact and Address Details, and Consent Form.

When shown answer, it simply displays the correct answer.

This screenshot shows the same Hofnote website interface as above, but with the correct answer displayed for the question "What is the new key?". The correct answer "Relative minor" is shown with a green checkmark and the message "Well done, you got that answer right!". The rest of the interface is identical to the first screenshot, including the exercise details, audio player, and footer.

When submitted, the site gives an extra line of output text displaying whether you got the answer right or wrong.

Overall, a minimalistic solution but includes all the necessary features required like playing the music audio.

E-Music Maestro

An online quiz solution to aural testing.

The screenshot shows a user interface for an online aural test. On the left, there's a sidebar with a dark vertical bar and a red navigation bar at the top labeled "Free Aural Tests". Below it is a dropdown menu for "ABRSM" with options for "Grades 1 - 5", "Grades 6 - 8" (which is selected), "Grade 6", "Grade 7", and "Grade 8". Under "Grades 6 - 8", there are sub-options: "8A : Repeat the lowest part of a three-part phrase, cadences, chord naming.", "8B : Sing the lower part of a two-part phrase while the upper part is played", "8C : Decide where the music modulates to at the end of passage" (which is highlighted in red), and "8D : Listen to the music and then describe the musical features". To the right of the sidebar, the main content area has a red circular badge with "8C" and the title "DECIDE WHERE THE MUSIC MODULATES TO AT THE END OF PASSAGE". Below the title is the instruction "Listen to the first piece of music and then name the modulation." followed by a media player showing "0:00 / 0:24". A note below the player says "Note: Media example changes with each 8C attempt". The "Part 1" question asks "Where does this music modulate to?" with four options: "Dominant", "Subdominant", "Relative Minor", and "Relative Major". A blue button at the bottom says "select your answers".

Modern UI which clearly and simply displays the necessary information and inputs. Audio is played back and analysis done by the user will determine their choice on the button selection.

This screenshot shows the same aural test interface as the previous one, but with a different audio clip. The media player shows "0:24 / 0:24", indicating the audio has finished playing. The "Part 1" question "Where does this music modulate to?" now has "Subdominant" selected as the correct answer, indicated by a dark grey background. A blue button at the bottom says "check your answers". A green success message at the bottom says "CORRECT The answer is: "Subdominant"" with a smiley face icon.

Afterwards a simple popup message displays on the bottom to reveal the correct answer – simple yet effective. Though cannot be displayed, the audio from here is exam-style structured with clear speaking by the instructor.

A very professional approach to aural testing practice.

Comparison

Graphical User Interface

Quizlet and E-Music Maestro both had modern UIs which were easier to understand, use and navigate. On the other hand, MyMusicTheory and Hofnote were quite traditional and outdated which used multiple-choice quizzes using radio buttons. Both are equally as effective at first glance but the simplicity and professionalism of the modern UI by Quizlet and E-Music Maestro connects with younger audiences and those who like a fun, simple atmosphere rather than being clouded by unnecessary information such as elapsed time on MyMusicTheory. MyMusicTheory is quite overly-decorated in which it highlights unnecessary information and tries to make things colourful by making worthless text stand out, such as highlighting the question number in bright blue whilst the questions and answers are black text on a white background. Quizlet provides multiple learning approaches each with a distinctive UI and the variety keeps the learning ‘alive’ and never monotonous. Hofnote and E-Music Maestro, the aural testing solutions, both display the audio in a similar fashion, but the actual quiz UI (the questions, answers, feedback and decorations) is much more professional in E-Music Maestro’s site since there is a more professional colour scheme and uses modern-looking elements such as flat-style buttons and coloured panes to separate information rather than just clustering all the information in one. Feedback is also much easier to identify and understand in E-Music Maestro because of the green/red colour scheme and the small use of words – not too verbose, yet effective.

Approaches

Content-wise, Quizlet was the most dominant with its variety of approaches to learning musical terms. The games were clearly enticing with increasing difficulty to keep the user engaged and the flashcards was comprised of lots of beneficial features such as customising, bookmarking for later and using other custom sets; this meant that for each and every person, their use of Quizlet will be different and adapted, most suitable for unique personalities with different bases of knowledge. MyMusicTheory only provided quizzes for people to learn by themselves from, there is no other form of way to learn these terms so this relied on the user themselves to learn from experience and by their own mistakes, correcting them after every test, hopefully improving in the long run. For a lot of people, this is quite effective, but there is still quite a large chunk of people who will not benefit with such independent, repetitive forms of learning. Hofnote, included ‘demos’ which were basically similar quizzes using audio and a question to test the user for their aural testing skills and then waiting for feedback via answers. Hofnote would provide a correction if needed and feedback on which answer was the correct solution however, like MyMusicTheory, these quizzes are the only form of learning and can be quite monotonous after a while. E-Music Maestro also

provides quizzes for which the operation is very similar to Hofnote, plays audio and waits for user interaction to provide feedback on their response.

Content/Questions

Quizlet can theoretically have an infinite amount of content and questions if the community continuously produces more and more content – though, there are a finite number of musical terms. Community involvement in Quizlet boosts the quality and quantity of the existing sets and will allow for current and future students to continually use this site. MyMusicTheory randomly displays questions and terms, possibly from a database which should, over-time, test on each and every term if enough tests are taken. There are no indications of which terms have yet to be learnt however a lot of the terms on the ABRSM syllabus should have been included. Both E-Music Maestro and Hofnote restrict some questions to only exclusive paid members so the free selection is quite limited and will not be beneficial to students who consistently use this solution as they will start remembering answers rather than the process to get to the answers. All four of the solutions distinguish the question selection by grades (except for Quizlet which may or may not be distinguished in specified grades – since the sets are provided by community) so all of them will provide relevant testing to any user from beginner to professional.

Potential Issues

MyMusicTheory: Simply lacks to attractiveness and appeal because of the outdated UI and the monotony of the lack of choice in learning approaches. Unnecessary information stands out too much and the screen is overly decorated which makes the clarity of the site low. Also lacks the ability to adapt to individual skill except for the feedback at the end which has corrections for mistakes.

Quizlet: Although a modern, varied approach, it lacks a way to keep track of progress and improvement. The feedback given back is also very minimal and could be more in-depth such as improvements made over time by not repeating mistakes etc.

Hofnote: An obvious issue would be the lack of options being limiting content to only paid members and so users will have less learning materials. Furthermore, the information is displayed in a clustered fashion and lacks the clarity needed on how to improve. Similar to MyMusicTheory, the UI is quite distracting and could lead to potential issues over time – e.g. users using a stressful interface will become stressful themselves easier and quicker. The tests are not clearly ordered either, the audio is displayed to the side although being a key output of the program in aural testing and the ‘show answer’ option is too trivial as it does not allow you to backtrack to the question and correct yourself to the correct answer, simply moving on after showing the answer – basically the submit function with less functionality.

E-Music Maestro: Similarly, this site has a limited selection of quizzes to the ‘free’ users meaning that users will over time face the same questions again and start remembering answers rather than practising the process to get to them. The quizzes are clearly well-structured and

displayed but does not include the ability to track the progress of students and improvements over time or store any areas of improvement for the users to focus on next time.

Conclusion

The music theory aspect of the current solutions covers a high majority of the terms in the ABRSM syllabus and some provide a choice of ways to learn and test user knowledge; these features will be beneficial and should be included into my own solution.

Regarding aural testing, there were all sections included from all grade levels in the existing solutions. Due to time constraints, it may only be possible to focus on one aspect of aural testing in my system; although lacking the quantity, it will still have the necessary features required, such as playing back audio in an exam structured format, to benefit the user and address the current problem of lacking ways to practise aural testing at home by yourself.

Additionally, hopefully where the existing solutions lack, my system will make up for.

My bespoke program will provide features for the user to learn music theory with a variety of approaches like Quizlet and provide feedback that is easy to understand. However, an improvement to quizlet, the content will adapt to the user so terms which were not understood fully (possibly identified by tests) will be focused on. The UI will be clean and clear, so it is easier to understand to improve from MyMusicTheory. Like E-Music Maestro, we can practise aural testing but with a higher choice of options not limited to paid members. The quiz will be structured well and be simple to follow. And to develop Hofnote's structure, the feedback will be displayed in a simple manner so the user does not waste time trying to understand their improvements.

To make the UI clear and modern, I will use JavaFX and CSS to make a flat-style, material design user interface and the feedback being stored and used to adapt future tests suggests an SQL database to be used; particularly, I will use Microsoft Access.

Features of Proposed Solution

My system's overall goal is to ease the practise of music theory and aural testing at home. However, that is a broad and indeterminate goal, so the following information will identify, explain and justify the system features in detail.

Essential Features

Testing: My system should include a testing facility which would demonstrate all necessary features including question display, multiple choice answer selections and progress indication. Testing is a key feature which is essential for the progress and improvement for a learning user as it gives direct insight on the user's current aptitude and knowledge; without it, the proposed solution would be bare minimal and not address the goal of this program which focuses on improvement at home.

Feedback: This feature is also an important as it gives users clear profundity on areas of improvement. With no feedback, users would not know where to focus on and how to improve faster. Giving feedback on whether answers were right or wrong and the correction if needed is very helpful to the users as it brings to light their misunderstandings on the current subject and corrects them.

Learning system: For the mass amount of terms in music theory, only having a testing feature would not be sufficient as it would become tedious, and ineffective to users who do not learn from experience by tests. Something different which has the soul purpose of learning rather than testing, such as a flashcards system, will allow a more relaxed environment and a more casual approach to learning – giving more options to different users and a variety of learning approaches to keep the learning ‘alive’ and enticing.

Aural testing: Since one of the main gaps in knowledge for music is aural testing, it is only natural to include a section of this proposed solution to be dedicated on practising aural testing. This should play back audio and give options for users to determine the features – e.g. multiple choice buttons. With aural testing functionality, the system will be much more effective as a learning resource and will be more sought for because of the lack of currently available aural testing resources.

Desirable Features

Adaptation: For an even more effective solution, if the program were adapted to the individual’s needs and expertise, the displayed questions would be more relevant and impactful to the user’s improvement. Adaptation can be achieved by, for example, focusing on questions which were previously answered erroneously. With adaptation, the user can also feel much more involved with the system and add onto the benefits of their unique journey of improvement.

Statistical Analysis: Another form of feedback which is not given directly or immediately is statistical analysis of tests. Statistical analysis is useful for parents who like to keep track of their child’s progress; graphs of percentage correct on tests can be displayed over time/date on the horizontal axis – the positive gradients can determine improvements made to their child! This is also motivating for the student themselves, they will have the tendency and urge to improve from last time and should hint a more enjoyable, competitive experience to their learning.

Minimalistic/Professional UI: Simplicity and professionalism allows users to navigate and operate the system much easily than if it were concentrated in unnecessary decorations. Also visuals and graphics can unconsciously keep the users engaged and find fun in music if the UI were appealing and simple to use. With material design controls or libraries, the system will be much more easier to use and also make the users feel more calmer and comfortable when using the system.

Customisability: Customising parts of the system creates benefits similar to adaptation – making it more relevant to the individual. Customising learning features such as new flashcard sets or editing existing sets allows the user, with combination of feedback, to focus on weak points and allow quicker improvements to gaps of knowledge. Also, small customisability features to change

the visual aspect of the system will give the users a small break from their learning whilst still enjoying the features the program will provide.

Separation by Grades: By separating terms/tests by grades, it allows the program to be more relevant to users of each grade although some knowledge overlap. Distinguishing grades will allow a wider audience to participate in the use of this program as the knowledge will be of relevance to the particular user.

Limitations

Networked: By being networked, in theory, my tutor could interact with the students at any time and vice versa if assistance from my tutor was required. Although a seemingly good feature to have, it requires a host/server to be established and ran at all times of day so one could connect to it and refer them to the other individual – this is using socket technology with Java. Since my tutor is rather elderly and inexperienced, she would not like the fact of having to have a background application (the server) being run at all times of day – this would also be very costly and resource extensive over time on the tutor's computer and so it would not be a preferable feature to include.

Aural Testing Restrictions: Because this is only an initial prototype of a full solution, I cannot include every single section of aural testing into my system due to time constraints. However, I will still choose to include a section which is hard to practise alone, at home and without the aid of technology – addressing the main problem.

Sound Recognition: For the sections of aural testing which test the user to reproduce the sounds by voice, sound recognition intuitively would seem to be profitable. However, not everybody owns a microphone to use for sound recognition, and even if they do, it would not always produce the highest quality of sound and so will be difficult to be analysed and accurately judged by the system. Additionally, I currently do not own the technologies and knowledge required to integrate sound recognition into the program.

Requirements for Solution

An important aspect which needs to be addressed are the requirements for the solution to actually work. By not specifying them and meeting them, unexpected events can occur by the system not completing its full functionality. Also, the problem this solution is tackling will not be resolved if the program cannot function fully.

Software Requirements

Software	Minimum	Recommended	Justification
Operating System	Windows Vista SP 2 Mac OS X (10.7.3) Ubuntu Linux (10.4)	Greater than minimum	These are the minimum operating system platforms supported to run JavaFX 2 programs

	Linux gtk2 (2.18)		
Java Run-Time Libraries	Java SE 6 Update 33 or Java SE 7 Update 6	Java SE 8	These libraries include the JVM which reads java byte code and executes it. Standard library functions/classes which are imported from the programs are also given in these libraries.
JavaFX Media	MainConcept H.264/AVC Decoder Pack or DivX Plus Codec Pack or MainConcept Showcase	Only one of the minimum codecs are required and recommended	These codecs are usually pre-installed in the operating system but they allow media (AAC audio and H.264/AVC video) to be played within programs.

Hardware Requirements

Hardware	Minimum	Recommended	Justification
Processor	226MHz	3.0 GHz	Minimum of 226MHz is required to run the Java Runtime Environment (JRE) which allows programs to be ran.
RAM	128MB	4 GB	Minimum is to only execute basic features of the Java library, most likely more will be needed.
Disk Space	126 MB	4 GB	124 MB is required to store the JRE and 2MB for Java Update but the program itself including many sound files and an indefinitely large database (can grow) could potentially take a lot of space over time.
Sound Card and Speakers/Headphones	Any	Any recent editions – higher quality sound.	These are the hardware required to listen to the music/audio played by the program.
GPU	Nvidia: Mobile: GeForce 8M and 100M, NVS 2100M, Mobility Quadro FX 300M Series Desktop: GeForce 8 and 100 series Workstation: Quadro FX 300 series ATI: Mobile: Mobility Radeon HD 3000, 4000 and 5000 series Desktop: Radeon HD 2400, 3000, 4000, 5000 and 6000 series Intel: Mobile: GMA 4500MHD and GMA HD Desktop: GMA 4500 and GMA HD	Any greater series.	The listed GPUs are the ones tested to be compatible with Java FX.

Success Criteria

A clearly identified and specifically justified success criteria allows the development of the system much more productive, focusing on the specific features the system needs.

With further investigation with my stakeholders, I have found some success criteria which would greatly benefit the system's users.

Functional

Index	Feature	Description	Justification
1	Grade Selection/ Separation	Button selection to choose the user's respective grade level and then separate all the following content of the system depending on that grade.	Separating the content by grades makes the content output to be more relevant to the individual. This can be measured and tested if the correct value is stored and then if all the following data outputted is in the correct grade level.
2	Navigation Side-Bar	A bar including multiple buttons which allows the user to navigate through separate screens such as testing, learning etc.	Giving a navigation side-bar allows the user to easily and efficiently navigate through the program and this criteria is measurable if it actually redirects to the correct screen.
3	Audio Playback	Audio loaded and played from the system to the user through speakers/headphones/earphones.	In aural testing, playing audio is a key feature required to have it function correctly. This is required to be analysed by the user and the criteria can be measured by the correct sound being played by the system.
4	Play/Pause Button	Plays or pauses the audio.	Users can play or pause audio to practise thinking about what was played and have time to analyse it. Also, it allows them to go back and play it again.
5	Seek Slider	Allows a slider to be interactive and be dragged to change the current position in the audio.	Allows the user to go back and improve on mistakes or skip some easy sounds to analyse. Testable if it works correctly.
6	Question Display	Text displaying the question.	This feature should be ubiquitous through the program as the system heavily relies on question and answer based feedback. This criteria is measured by the correct question and answer combination being displayed.
7	Multiple Choice Answers	Multiple answers being displayed with a way to input the choice: radio button and text / normal button.	My stakeholder really likes the idea of multiple choice as it allows a 'quick way of learning' and so my testing features will be based on multiple choice. Easily tested and measured.
8	Progress in Quiz	A way to display to progress through the quiz.	Makes the user expect how long a test can take and fit it in through their daily lifestyle. Easily testable if shows correct progress through the quiz.
9	Question Feedback	After answering a question, feedback on whether the answer	Giving feedback is an essential part of improving and learning. Easily testable by

		was correct or wrong, and then the improvement if wrong.	comparing the feedback with the correct answer.
10	Results Feedback	After finishing a test, the program will output percentage correct and commonly mistakenly answered questions.	Similar to question feedback, this will allow users to know what to focus on and improve further. Measurable if it displays the correct percentages and corrections.
11	Storing the Data	Questions, answers, audio, progress score and dates stored in a database or within the program executable.	A lot of data is required for the system to fully function. Data may change or be persistent, but either way, they need to be stored for future use. Easily testable if correct outputs appear.
12	Graphical Progress Display	A persistent line graph displaying percentage correct over time backed up by SQL database.	This will allow the parents to monitor the child's progress and also give the student an extra challenge or pressure to motivate them to improve from a previous record.
13	Flashcard Set List	A screen which displays a list of all flashcard sets with a text title and a button to enter each set.	This allows the user to easily compare which flashcard set is more important and navigate through into each one.
14	Creating New Sets	A way to allow the user to create a new set and add it to the existing SQL database – correctly formatted.	Adding this customisability to the system makes the program more personal and improves individual skill rather than generally – makes the learning more effective. Can be measured if the data is added in the correct format inside the database.
15	Deleting Sets	A button allowing deletion of an existing flashcard set.	Sometimes, sets which are no longer needed or outdated can be deleted so it saves space or simply cleans up the list set visually.
16	Editing Sets	A screen to edit existing flashcard sets.	Allows for updates to the set to be made or small changes like spelling error corrections to be made. Testable if the changes are made to the database itself as well.
17	Flashcard Term	A text display which shows the musical term.	In combination with its definition, the flashcards allow an alternative way of learning.
18	Flashcard Definition	A text display which shows the definition of the associated term. Will be showed 'on the other side' of the flashcard by responding to arrow key /mouse inputs.	A more interactive, fun, own-pace form of learning which is good to give a larger variety and keep the user engaged in their learning. Easily tested if the question and answer are paired correctly and inputs are handled.
19	Close the Application	A button to exit the application and close all resources if required.	Allows the user to stop, and take a break.

Non-functional

Index	Feature	Description	Justification
20	Motivating	System should inspire the students and make them constantly motivated to improve.	This is very important to our stakeholder, Jill. She wants her students to feel more enthusiastic about music rather than a ‘chore’ to do. Can be measured by the student activity or frequency of tests taken.
21	Robust and Error-Free	The system will not contain any obvious errors and should not have any paths to break the system.	If the system breaks, this can greatly damage the user’s drive to learn if their learning is interrupted by bugs. Can be measured by number of errors appearing over high number of program uses.
22	Easy to use	Easy to use and adopt interface which will not confuse users.	Users should not feel obstructed by the clunky and unclear interface or have difficulties using the system as this will greatly slow down the progress of the user. Can be measured by the speed of the user using the system.
23	Clean and Efficient Code (Sustainability)	My code should be clean without any unnecessary snippets being added which have no meaning. However, the code should be efficient on the resources and time the system requires.	Unclear code will not affect the user but maintenance or sustainability of the system will be greatly affected if the code gets scattered and misplaced. Also, efficiency will be more noticeable if the system grows so we should maximise on efficiency even on the small aspects. Can be measured and optimised by the number of lines or time complexity of an algorithm or the entirety of the system.
24	Recoverable	If the student accidentally breaks it, a feature of resetting the data to its original state will recover any problems.	An easy escape option for unexperienced computer users who may have accidentally deleted a set they actually wanted, for example. Can be determined whether the respective recovery functions are functioning properly.

Design

Decomposing the Problem

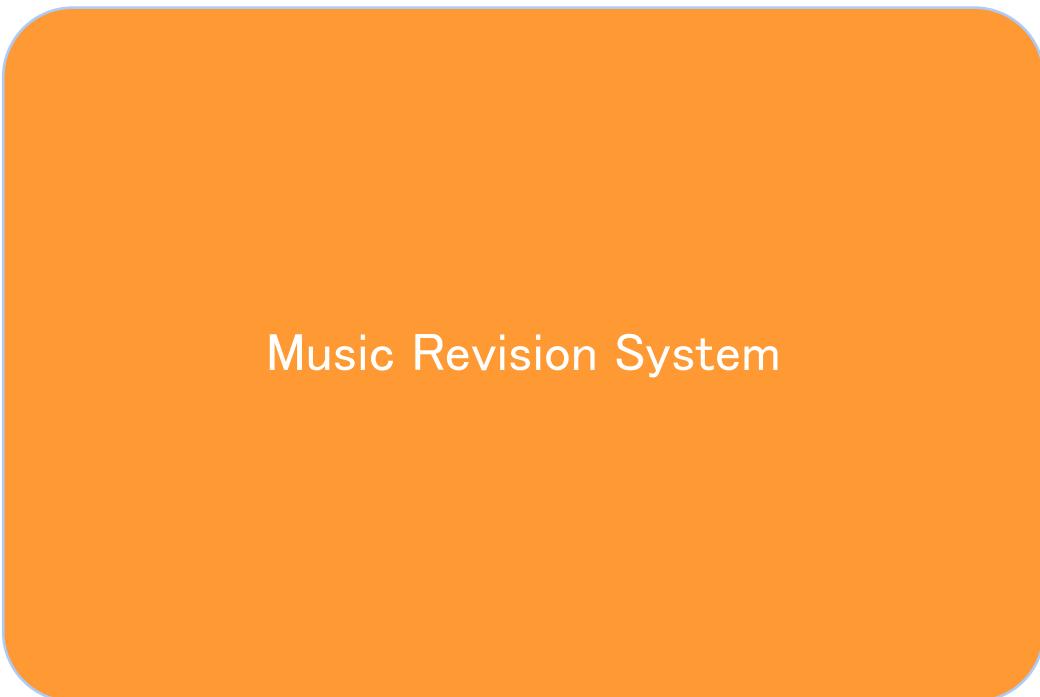
In this section, the main problem will be decomposed into a series of smaller parts, suitable for computation solutions.

By systematically decomposing the problem, via top-down design to break the problem into smaller problems and then stepwise refinement to detail these smaller problems, the problem can be concretely solvable by completely defining each solvable detail and following it up by creating its respective solution.

Decomposition will add more clarity to what needs to be done and it can specify details of each method (procedure/function) or a specific important line of code if broken down far enough – though, not every section needs to be broken down too far as you only require sufficient information to perceive what needs to be done.

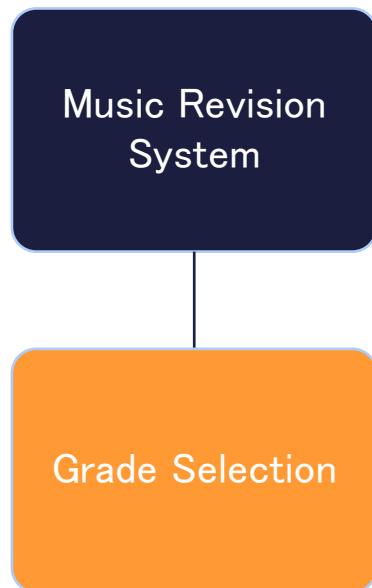
Problem Decomposition and Step-Wise Refinement

The Main Problem



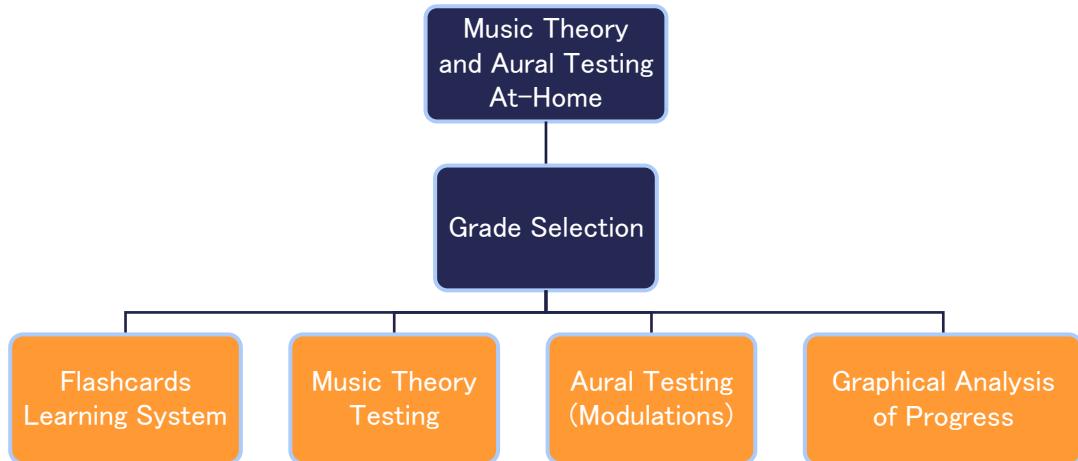
Music Revision System

First Level



Problem	Explanation	Justification
Grade Selection	Users will require their respective grade to be selected so that the content is relevant.	It is crucial that this problem is decomposed firstly as this feature will follow on and affect all the content produced by the system afterwards. This sub-problem can be easily solved and tested by storing the respective grade into an instance variable.

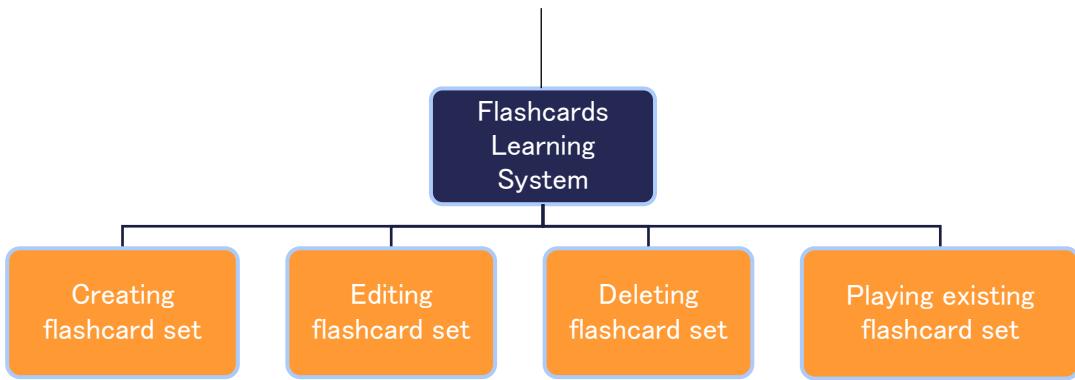
Second Level



Problem	Explanation	Justification
Flashcards Learning System	An alternative solution to learning which provides an engaging approach to memorising musical terms.	Students can get bored if only one form of studying were available, by providing a successful and fun way to memorise (i.e flashcards), it will allow students to keep their learning active and ‘alive’. Decomposing into this sub–problem makes development easier and clearer because of the modularity.
Music Theory Testing	Testing/Quiz solution for music theory where student scores are kept and recorded to measure progress.	Testing allows for instant feedback to improve individuals and identifies clear weak–points in their knowledge. Decomposing into this sub–problem makes development, especially debugging, more structured and clarifies sections of code to be created.
Aural Testing (Modulations Only)	Aural testing feature which plays audio and takes in answers.	Similar to music theory testing, it provides instant feedback and identifies specific weak–points. However, aural testing is especially important as practising aural testing at–home is difficult, as identified in problem identification. Decomposing into a sub–problem like so allows for separation between aural testing and music

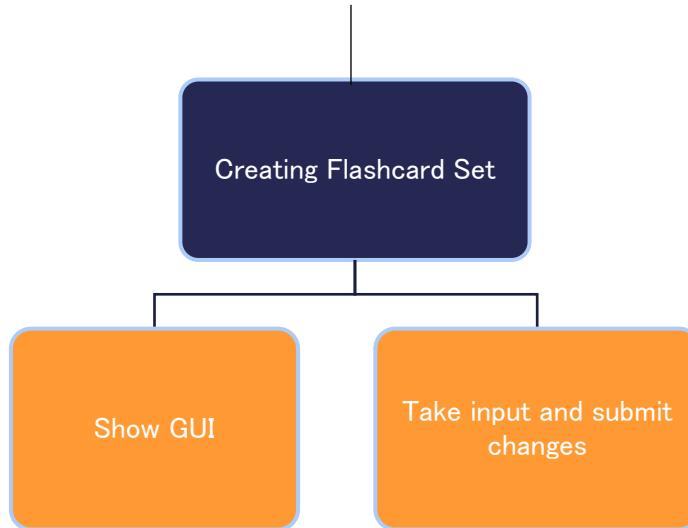
		theory – giving the code more modularity and encapsulation.
Graphical Analysis of Progress	Graphs from data recorded from the test, scores from an SQL database, to measure progress of the student over time.	An extra layer of motivation to learn – to beat previous scores. Also allows parents or carers to keep track of their child's/student's progress and monitor their work done. Decomposing into this sub–problem like so separates the feedback and activity, also isolating what the parents should see.

Third Level (1)



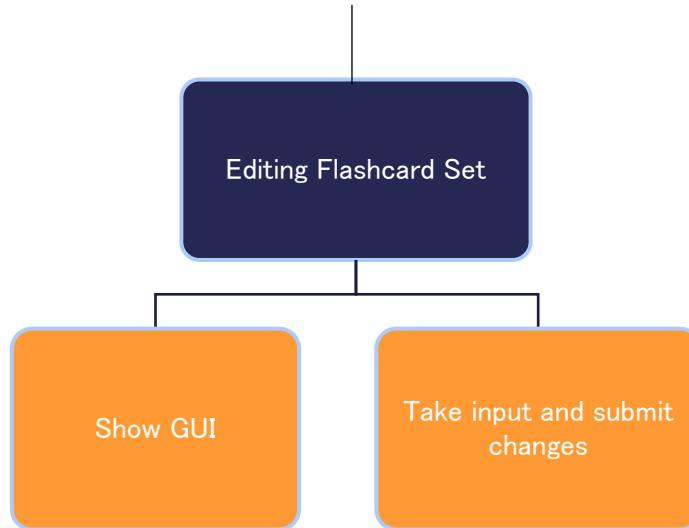
Problem	Explanation	Justification
Creating flashcard set	Create a new flashcard set and store it to the SQL database.	Creating flashcard sets allows the content to be more personal and relevant to the individual's aptitude. Adding more content and storing the data is suitable to be computational as changes are hard to be made on concrete items like books.
Editing flashcard set	Changes to be made on existing flashcard sets also to appear on database.	Sometimes, updates or corrections will need to be made on existing flashcard sets, without this feature, it will diminish the longevity of the program as it does not keep up with latest editions of music.
Deleting flashcard set	Deleting an existing flashcard set from the system and database.	Flashcard sets can go outdated and irrelevant as music progresses. The added feature of deleting archaic flashcard sets keeps the program clean and small (storage wise).
Playing existing flashcard set	Play the flashcard set.	After all the administrator stuff, the user still wants to be a user and actually use the flashcards. So this comprehensive feature will permit that. A method can do this.

Fourth Level (1a)



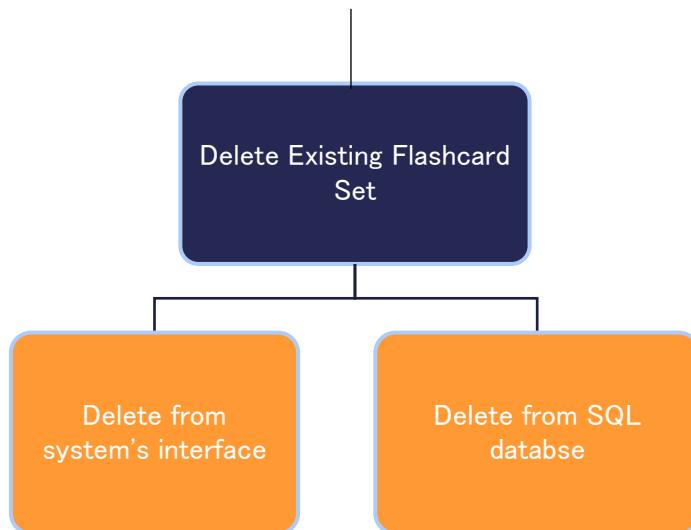
Problem	Explanation	Justification
Show GUI	A separate interface to be displayed – only purpose is to create a new flashcard set.	A separate GUI allows for the current interface to not be too clunky and clustered – making it minimalistic, simple and easy to use. Can be done using a subroutine.
Take input and submit changes	The inputs (i.e terms and definitions) will be taken in and changes will be made to the database and the new set will be added to the user interface.	This will allow for a persistent addition to the program. The customisability makes the program more relevant to the individual and will make the learning more effective.

Fourth Level (1b)



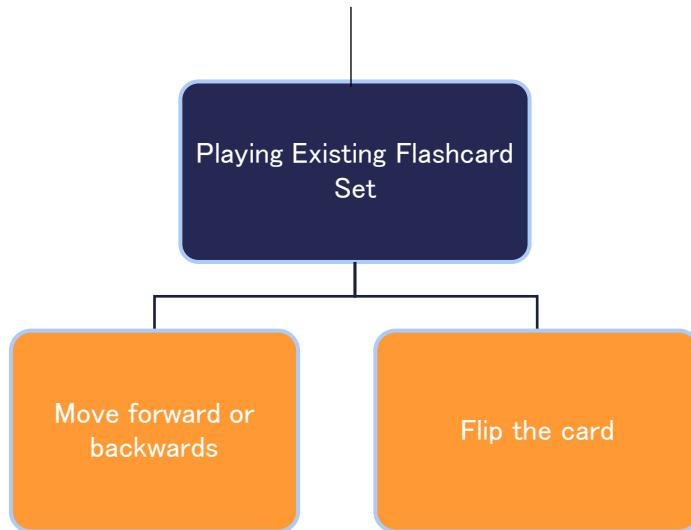
Problem	Explanation	Justification
Show GUI	A separate interface for the flashcard set to be changed.	Similarly for creating new flashcard sets, a separate GUI makes the program less clustered and cleaner – giving more clarity to the user.
Take input and submit changes.	Input to be taken in and to change the database and existing flashcards respectively.	This is where the magic happens and the changes made to the program will be permanent. The database backing up the program will be edited and will therefore change future uses of this system.

Fourth Level (1c)



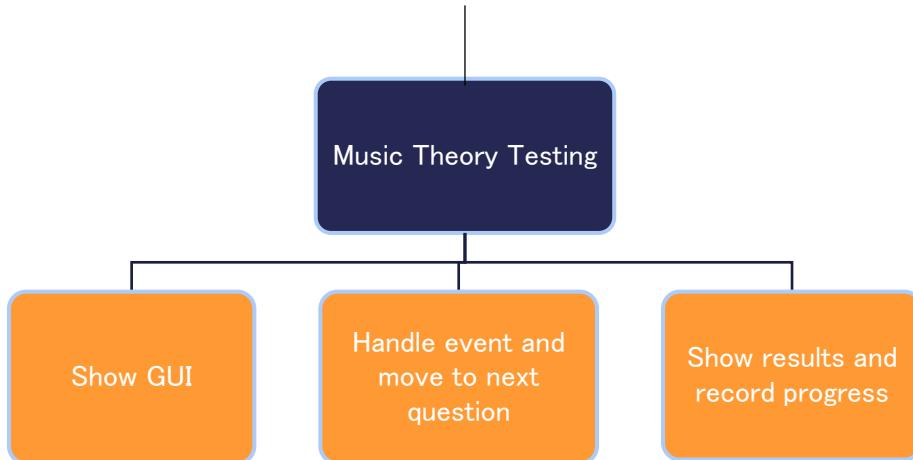
Problem	Explanation	Justification
Delete from system's interface	The flashcard set will be deleted from the list of sets.	The deleted set should not show up as an available set on the list. This can be done by a subroutine.
Delete from SQL database.	Delete set from database	This causes permanent deletion of the set from the system. This cannot be reverted. Will be achieved via the same subroutine.

Fourth Level (1d)



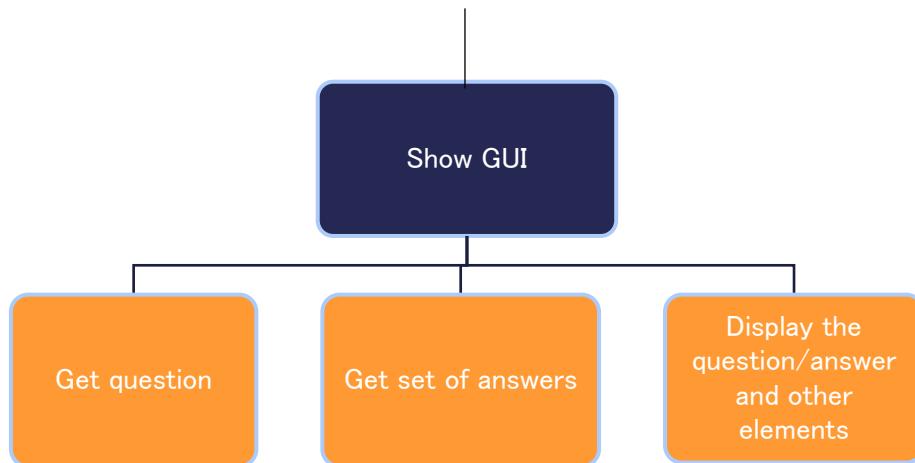
Problem	Explanation	Justification
Move forward or backwards	Navigating to the next or previous flashcard.	You want to use multiple flashcards in your set. By moving on and going back, it gives the user a more personal touch to their learning as it gives them freedom to work at their own pace. This can be done in a subroutine.
Flip the card	Show the opposite definition/foreign term.	To test themselves, flipping the card and seeing the respective result, answer or definition is the epitome of learning by flashcards. This can be achieved by a method.

Third Level (2)



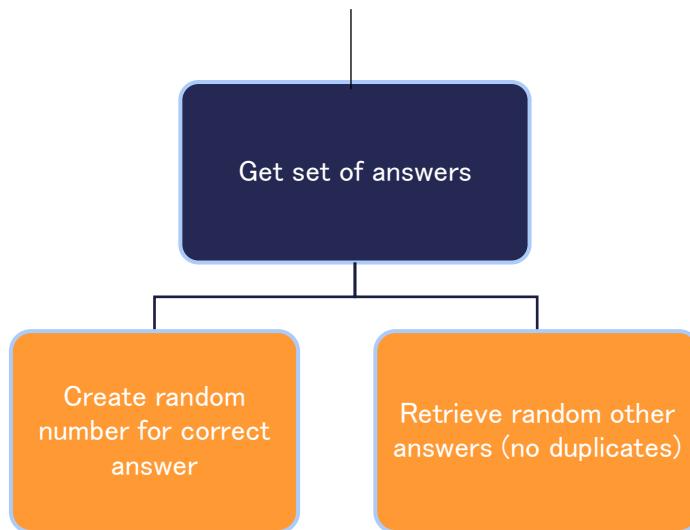
Problem	Explanation	Justification
Show GUI	Display questions and multiple-choice answer buttons.	A GUI is required for the user to interact with. The elements will be displayed by an external fxml view and initialised.
Handle event and move to next question.	The answer will be recorded and the next question will be displayed	Answer will be recorded at the end to adapt future tests and show progress. Also displays feedback. Then, displays next question and set of answers. Can be done in an event handler/event method.
Show results and record progress	At the end of the test, show the results and record the percentage correct for graphing in the database.	Direct feedback on how well one has done on the test provides a measurable way to check one's progress – also in combination with the graphing explained later.

Fourth Level (2a)



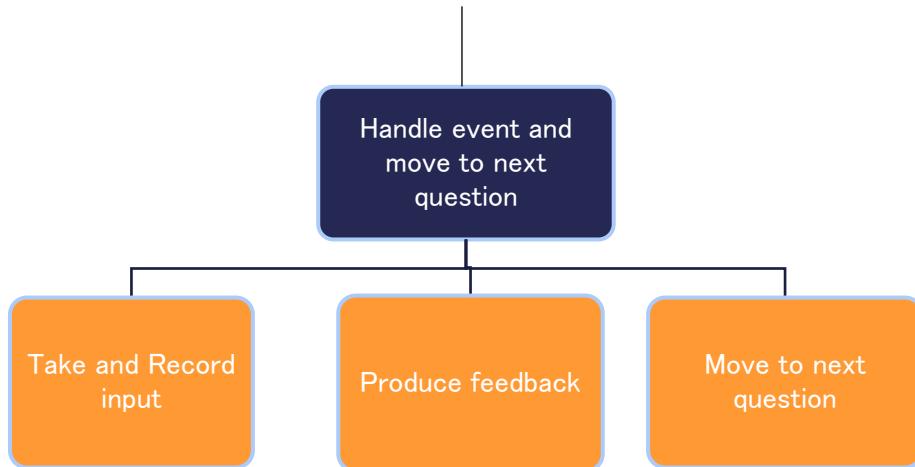
Problem	Explanation	Justification
Get question	Retrieve the question from the database, prioritising ones previously answered wrong. The question will also be dependent on the grade selected.	A relevant question must be selected to give relevant content to the individual. Can be done in a method.
Get set of answers	Get the answer selection from database.	Possible answers for the user to select. Can be done in a method.
Display the question/answer and other elements	After retrieving the data required, display them.	The test is done visually so requires the information to be displayed on the interface. Easily tested if the correct information is displayed.

Fifth Level (2ai)



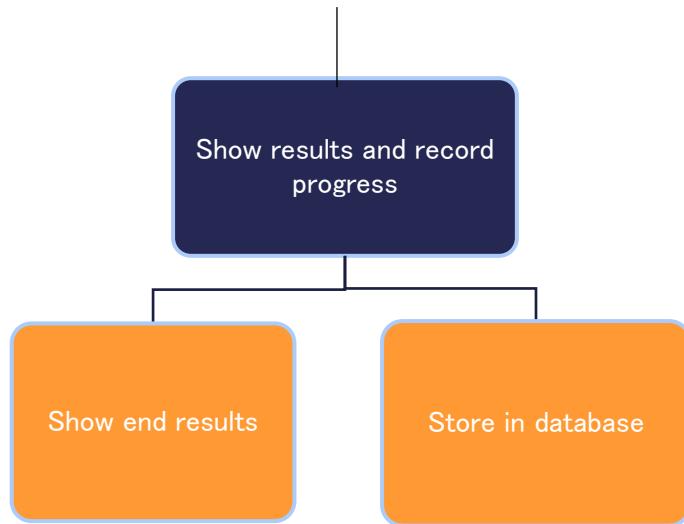
Problem	Explanation	Justification
Create random number for correct answer	A random number to choose which button to be allocated for the correct answer.	A random sense will make sure the user cannot cheat and so makes the learning benefit greater.
Retrieve random other answers (no duplicates)	Retrieve random answers from other terms to fill up the other buttons.	In multiple-choice, you need other answers which are wrong to make the test challenging and engaging.

Fourth Level (2b)



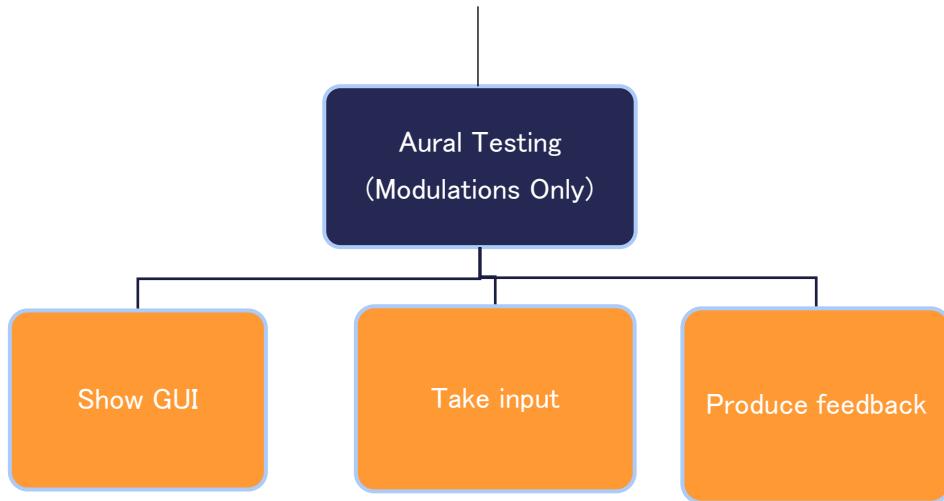
Problem	Explanation	Justification
Take input	Take and record input from the user	Input from the user requires to be taken then stored in the database to adapt future tests.
Produce feedback	Provide feedback on the whether the input was correct/wrong and the correct answer if applicable.	Feedback is required for quick corrections and reflections on terms and definitions. This will be a key feature to benefit the learning of the user.
Move to next question	After retrieving the data required, display them.	There are multiple questions in the test so the user will need to move onto the next question. Displaying next question and set of answers – calling the subroutine associate with ‘show GUI’ .

Fourth Level (2c)



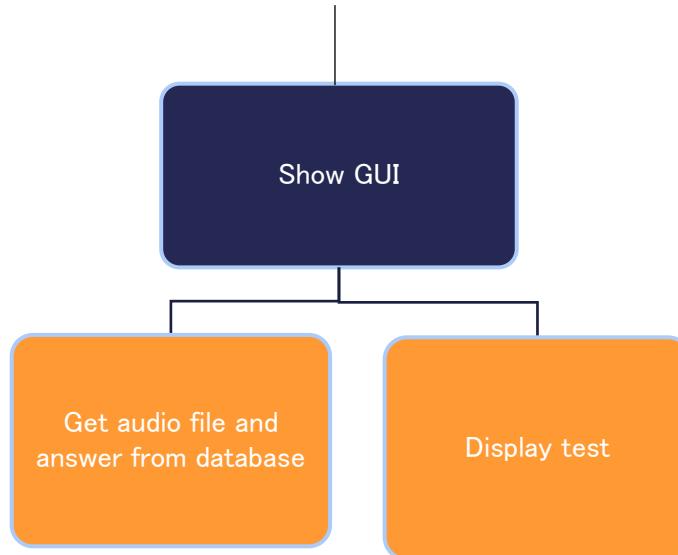
Problem	Explanation	Justification
Show end results	End results of percentage correct, wrong questions and corrects will be displayed.	An overview or summary of the test at the end will help the student progress and remember which terms to focus on for next time.
Store in database	The results and date will be stored in a database.	The storage within a database will allow for a graph to be plotted over time to indicate the student's progress.

Third Level (3)



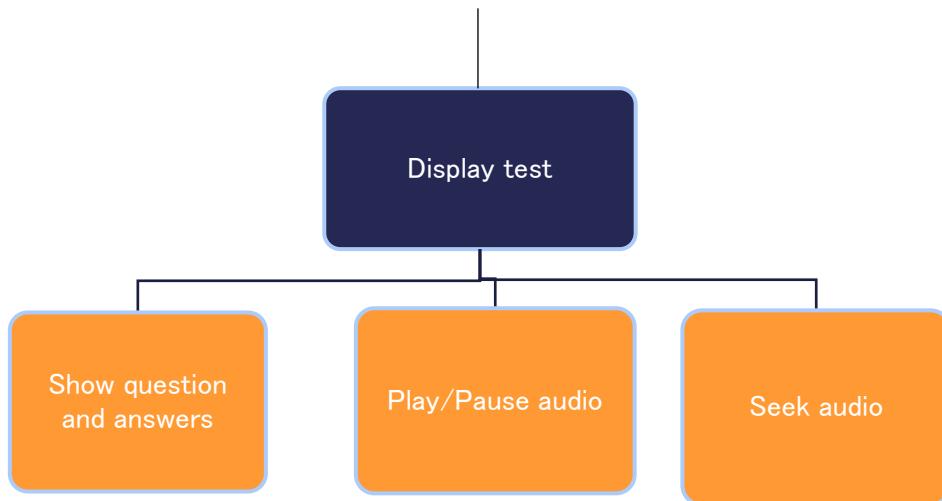
Problem	Explanation	Justification
Show GUI	Display question, answers, and audio.	A basic test format for aural testing will be required for user interaction. Easily testable if user interactions are functioning properly.
Take input	Take input of the user.	Using the multiple-choice selection, get the selected input by the user. This will be used to provide the relevant feedback.
Produce feedback	Display relevant feedback, giving corrections.	Corrections are a crucial part of learning as described in the music theory testing. Easily testable if the expected output is produced.

Fourth Level (3a)



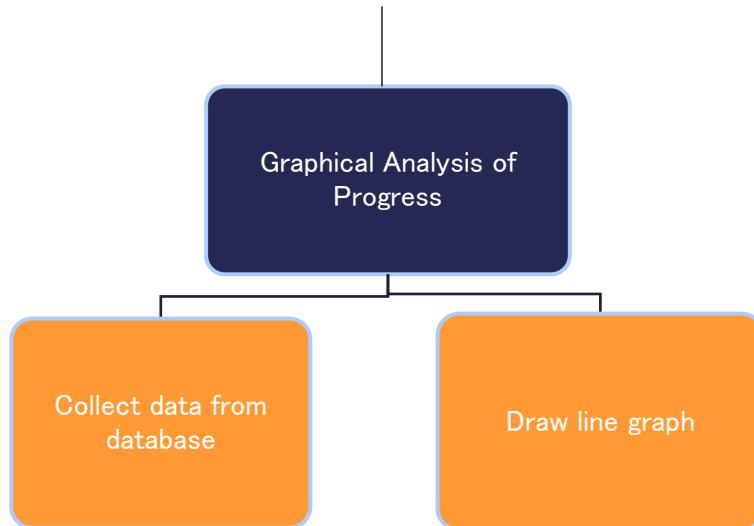
Problem	Explanation	Justification
Get audio file and answer from database	The audio file and respective answer will be retrieved via database.	The audio file used for aural testing will be loaded inside the program using the file name specified by the database. The answer will be used to check the correct the user input. Note that the question and set of answers will be always the same.
Display test	The test elements will be displayed on the interface.	This is the GUI the user will be using the interact with the system – allowing for communication between the user and system and effectively testing the user.

Fifth Level (3ai)



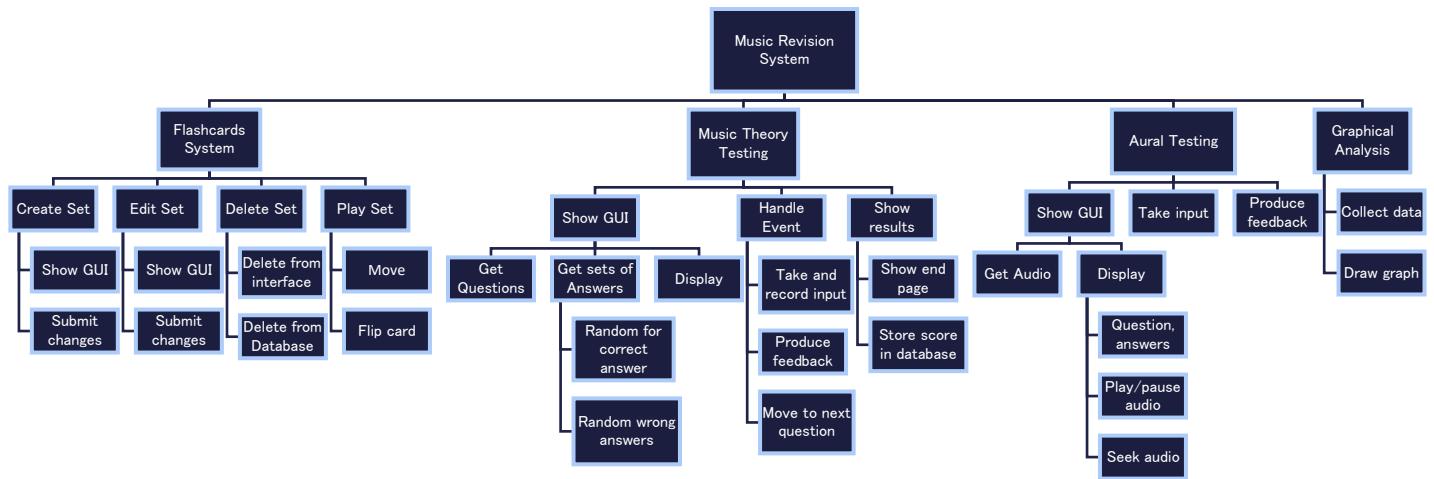
Problem	Explanation	Justification
Show question and answers	Display question and selection of buttons for answers.	The bare bones of a test will be required.
Play/Pause audio	Play the audio if paused or vice versa.	Mostly for the benefit of practise, the user can play or pause the audio to think about the audio more thoroughly and deliberate over the analysed feature.
Seek audio	Move the current playing audio to a different position in time.	Also for practise, the user can go back to after receiving feedback and locate where they went wrong. Beneficial for improvement. Testable whether it accurately moves to the correct position of time in the audio.

Third Level (4)



Problem	Explanation	Justification
Collect data from database	The relevant data showing the user's music theory test results and date will be collected.	Data stored by the database will be persistent data recorded from previous tests, this is what backs up the data of the line graph(s). Collection will be using an SQL SELECT query.
Draw line graph.	Using the data collected, draw a line graph from it using JavaFX.	Statistical analysis and graphical plotting gives a more visual representation of a student's progress simplified from words. This will greatly motivate one's urge to learn and will meet the stakeholder's demands of making the learning engaging. Easily testable if the correct points are showing up on the graph.

Full Problem Decomposition Hierarchy



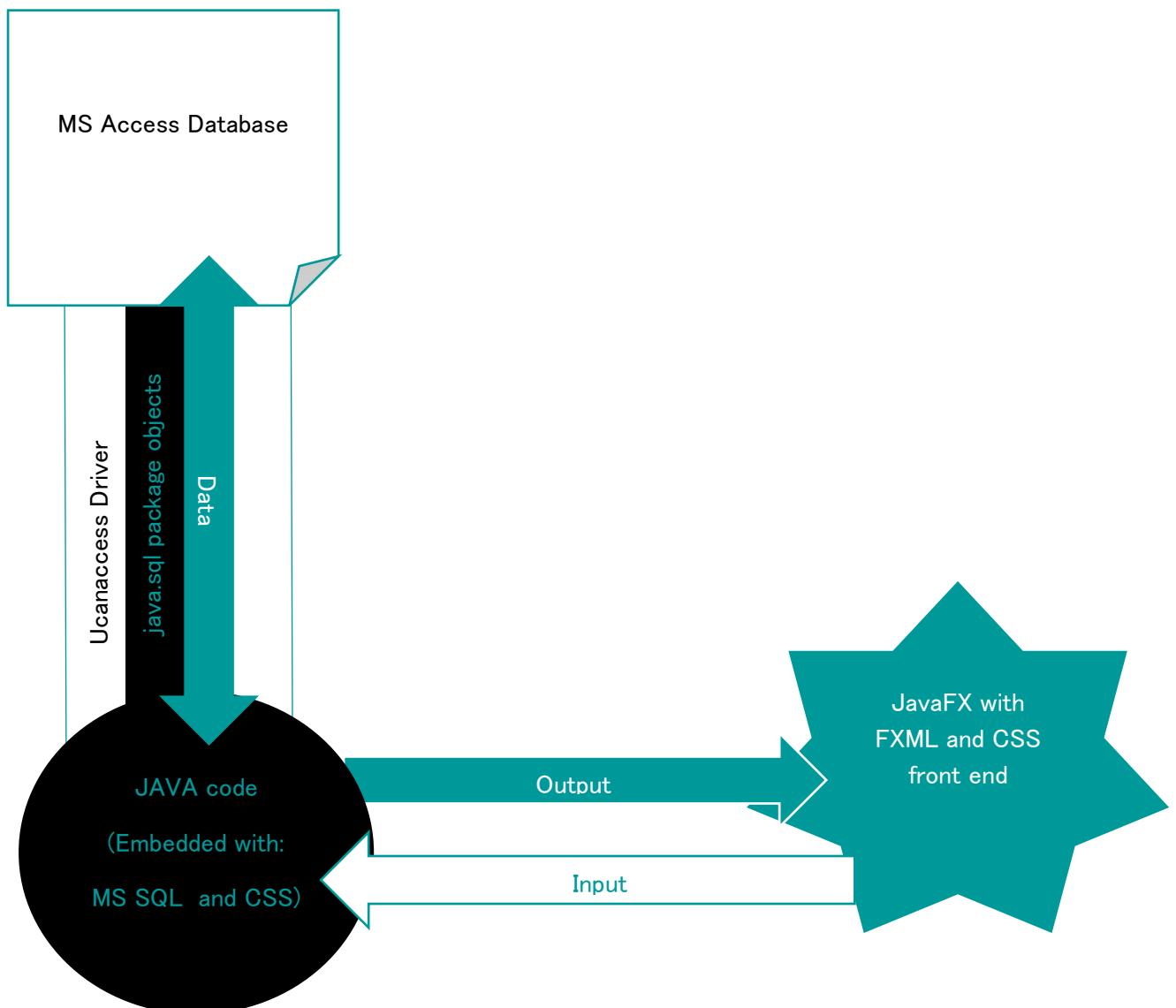
Structure of the Solution

The structure of a solution can comprise of:

- the structure of the data flow between the technologies used
- the structure of database entities
- the structure of the algorithms

Structure of Data Flow between Technologies

Below displays a visualization of the technologies to be used in my system and how they will interact with each other.



Structure of Database

Firstly, brain-storming a list of attributes which would be useful for our system produced a 0NF (zero normal form) list of fields for our database:

- TheoryID
- ForeignTerm
- Definition
- Grade
- AuralID
- File
- Modulation
- ScoreID
- ScoreDay
- ScoreMonth
- ScoreYear
- Highscore

After this, it was easier to separate out our fields and identify enclosing entities for our database, resulting in a 1NF (first normal form) list of fields:

Theory

- ID (primary key)
- ForeignTerm
- Definition
- Grade

Aural

- ID (primary key)
- File
- Modulation

Score

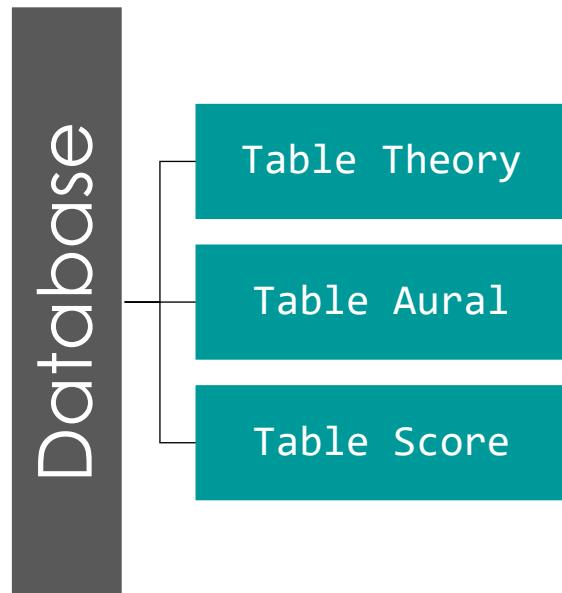
- ID (primary key)
- ScoreDay
- ScoreMonth
- ScoreYear
- Highscore

This list is also in 2NF because there are no partial-dependencies (no composite keys in the first place) and is also in 3NF because each of the attributes only depends on the ID of each entity so no non-key dependencies.

Normally, at this stage, it would be possible to link the entities together using an entity relationship diagram, or an ERD. However, these entities are unique to each other and function

independently of one another, so there will be no links nor relationships to be made. Also, there will be no data redundancy because of the independence of each entity.

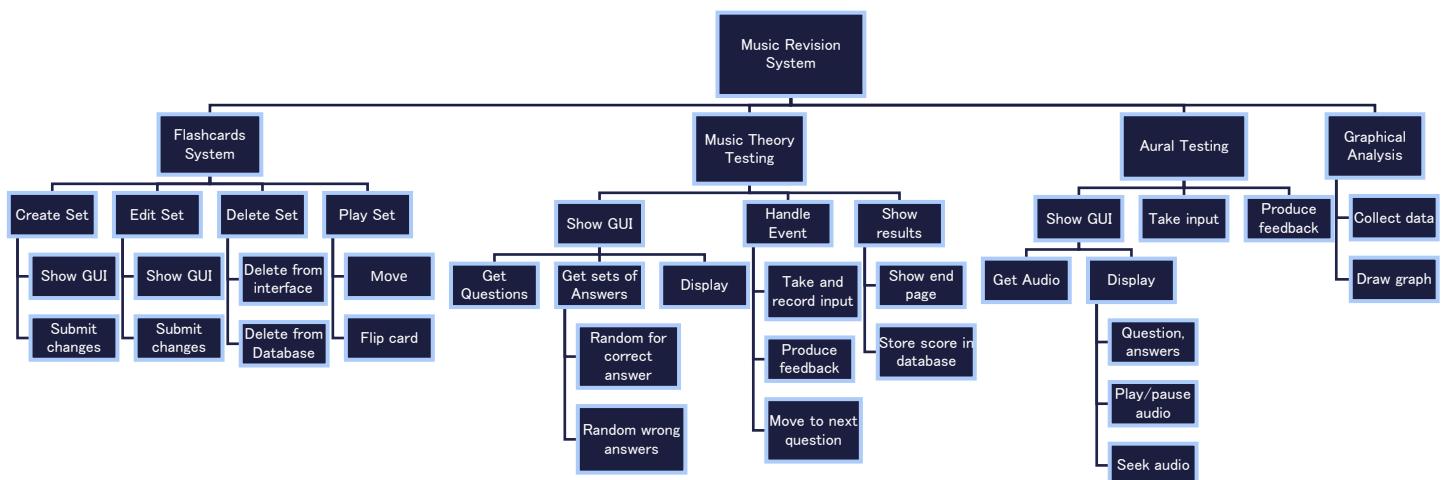
The structure of our database can be visualized as so:



Structure of Important Algorithms

The important algorithms are listed below as the leaf nodes of our problem decomposition hierarchy. Also, our algorithms section will in-depth, fully describe how the important algorithms will work and link together.

(Extracted from Decomposing the Problem)

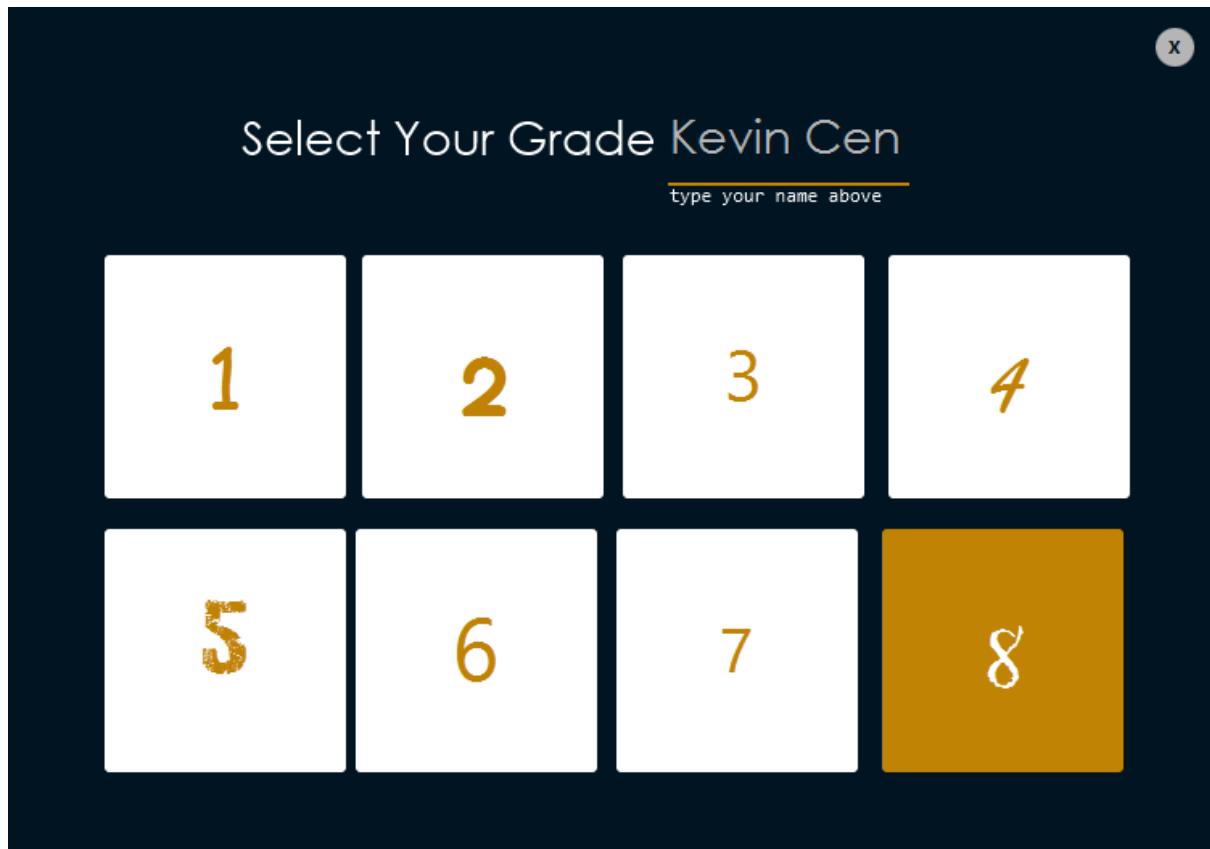


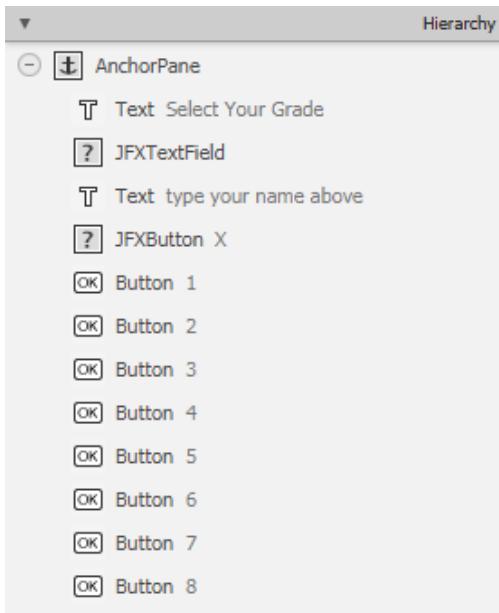
Usability Features

It is vital for the visual usability to be as simple to use as possible – this is especially true for my stakeholders who are not too familiar with technology. We will design our views with JavaFX Scene Builder to create a .fxml file.

Grade Selection Screen

As you enter the system, this screen will be the first view to be displayed (grade_select.fxml).



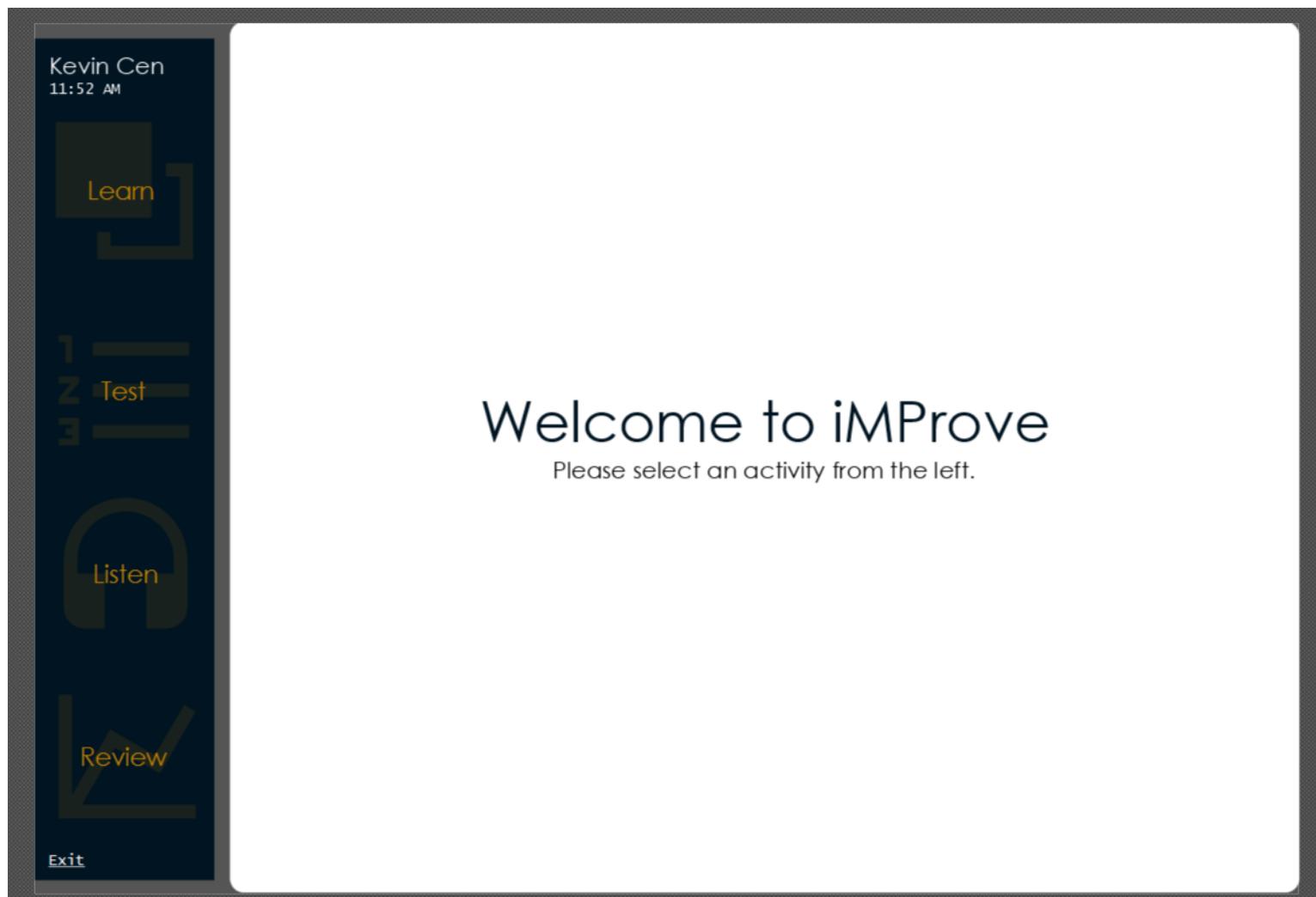


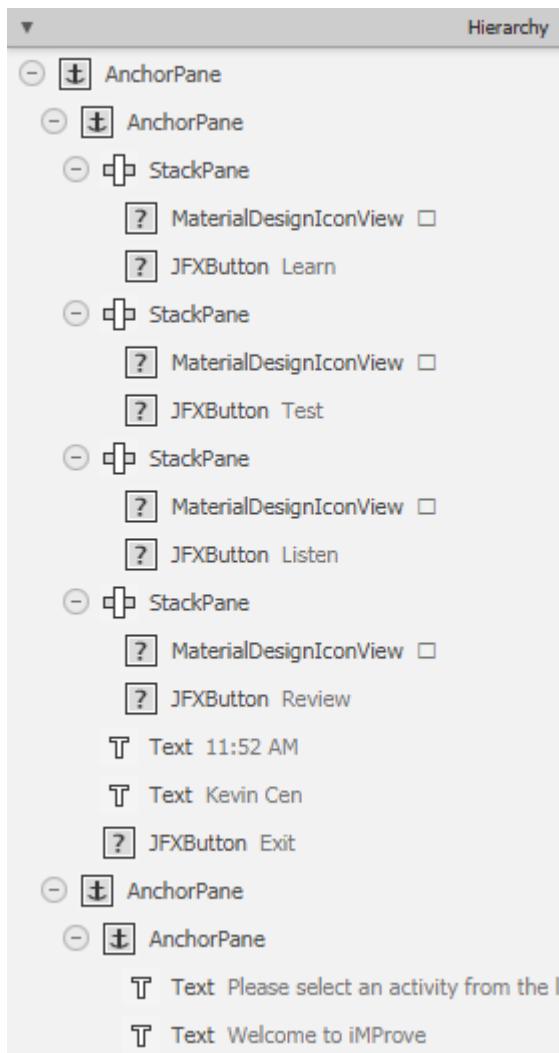
Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnGradeRoot	AnchorPane	None	-fx-background-color: #001421	The root anchor pane where all elements are children of.
txtSelectGrade	Text	None	Century Gothic 29px, White	Displays the instruction for the user simply.
fldName	JFXTextField	None	Century Gothic 25px, #b5b5b5	A text field integrated with the sentence for a minimalistic way to enter your name.
txtType	Text	None	Consolas 12px, White	Clearly displays what thetextfield above does to make it more obvious
btnClose	JFXButton	close()	System 11px, #001421. -fx-background-color: #b5b5b5 -fx-background-radius: 15	Closes the application if the users do not want to continue after just opening the system.
btnOne	Button	gradeSelect(num)	Comic Sans MS 52px, #c18303	Selects Grade 1
btnTwo	Button	gradeSelect(num)	Alba 52px, #c18303	Selects Grade 2

btnThree	Button	gradeSelect(num)	BatangChe 45px, #c18303	Selects Grade 3
btnFour	Button	gradeSelect(num)	Freestyle Script 66px, #c18303	Selects Grade 4
btnFive	Button	gradeSelect(num)	28 Days Later 63px, #c18303	Selects Grade 5
btnSix	Button	gradeSelect(num)	CF Anarchy 54px, #c18303	Selects Grade 6
btnSeven	Button	gradeSelect(num)	Ebrima 39px, #c18303	Selects Grade 7
btnEight	Button	gradeSelect(num)	Gigi 65px, White. -fx-background- color: #c18303	Selects Grade 8

Welcome Screen

This will be the structure of all the screens to come – which will be explained why later. However, it will look like this (root.fxml):





Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnRoot	AnchorPane	None	-fx-background-color: transparent 918x632px	The root node used to hold all the elements below.
apnNavigate	AnchorPane	None	-fx-background-color: #001421 128x611px	The pane used to navigate around the system.
spnLearn	StackPane	None	128 x 133 px	Used to hold the button and icon in the same place to make the button more aesthetic.
icnLearn	MaterialDesignIconView	None	Glyph Name: ARRANGE_BRING_FORWARD Fill: #bf8104 0.13 Transparency	To display a visual representation of learning – easier to identify.

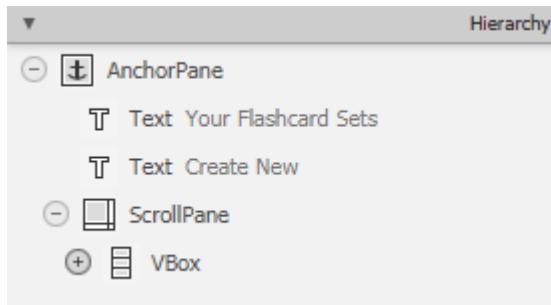
btnLearn	JFXButton	goLearn()	Century Gothic 18px, #c18303	Used to handle the event of selecting to learn.
spnTest	StackPane	None	128 x 133 px	Used to hold the button and icon in the same place to make the button more aesthetic.
icnTest	MaterialDesignIconView	None	Glyph Name: FORMAT_LIST_NUMBERS Fill: #bf8104 0.13 Transparency	To display a visual representation of testing – easier to identify.
btnTest	JFXButton	goTest()	Century Gothic 18px, #c18303	Used to handle the event of selecting to test.
spnListen	StackPane	None	128 x 133 px	Used to hold the button and icon in the same place to make the button more aesthetic.
icnListen	MaterialDesignIconView	None	Glyph Name: HEADPHONES Fill: #bf8104 0.13 Transparency	To display a visual representation of listen – easier to identify.
btnListen	JFXButton	goListen()	Century Gothic 18px, #c18303	Used to handle the event of selecting to listen.
spnReview	StackPane	None	128 x 133 px	Used to hold the button and icon in the same place to make the button more aesthetic.
icnReview	MaterialDesignIconView	None	Glyph Name: CHART_LINE Fill: #bf8104 0.13 Transparency	To display a visual representation of review – easier to identify.
btnReview	JFXButton	goReview()	Century Gothic 18px, #c18303	Used to handle the event of selecting to review.
txtName	Text	None	Century Gothic 17px, White	To display name, making the system a little more personal.
txtTime	Text	None	Lucida Console 11px, White	Displays the time to keep track of length of time practising.

txtExit	Text	close()	Lucida Console 11px, White	Used to exit the system.
apnActivity	AnchorPane	None	-fx-background-color: white -fx-background-radius: 10 776x632px	This is the parent node which will hold all later activities – such as testing or flashcards etc.
apnWelcome	AnchorPane	None	-fx-background-color: white -fx-background-radius: 10 776x632px	The current activity – just a welcoming screen. Will be inserted into apnActivity.
txtWelcome	Text	None	Century Gothic 41px, #001421	Displays simple welcome message – friendliness.
txtSubtitle	Text	None	Century Gothic 17px, BLACK	Gives simple instruction alongside the welcome message to get started.

Learn

After you click the ‘Learn’ button on the side navigation pane, the button will turn orange with the text and icon turning white; new pane will be put onto the apnActivity and therefore keeping the same structure as the welcome screen. This screen will look like as following (learn.fxml):

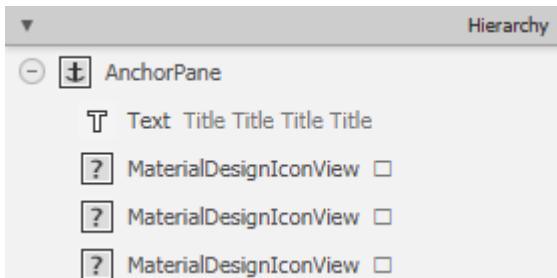




Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnLearn	AnchorPane	None	–fx–background–color:white –fx–background–radius: 10 776x632px	The root element holding everything below. Will be inserted into apnActivity.
txtSets	Text	None	Sweetly Broken 73px, #001421	Title describing what the lists below are giving a clear insight of what this screen is.
txtCreate	Text	createSet()	Consolas 11px, #001421	Used to create the set, positioned away from the focal point of the current sets as this is only an added customisation feature.
scrpnSets	ScrollPane	None	HBar Policy: NEVER VBar Policy: AS_NEEDED –fx–background–color: white 651x459px	Used to hold all the flashcard sets if they get larger than the vbox and so the users can navigate to the ones at the end of the vbox.
vbxSets	VBox	None	–fx–background–color:white 649x457px	The container for the AnchorPanes which describe a single flashcard set each.

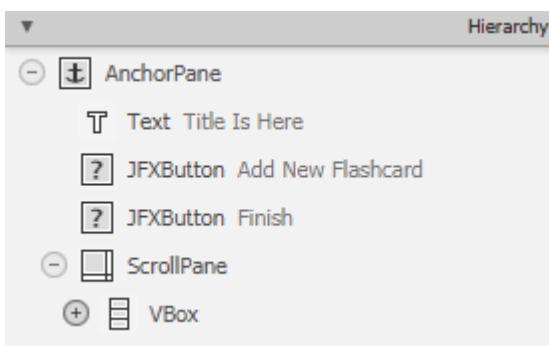
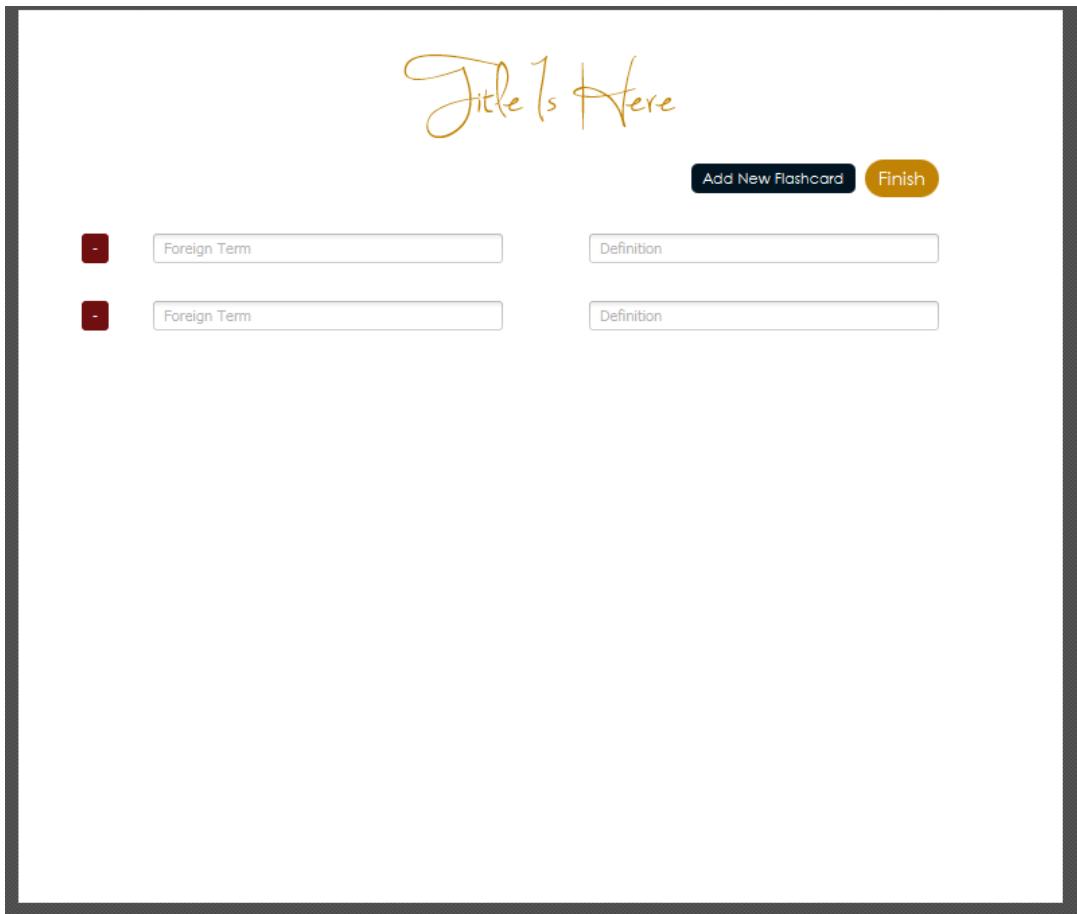
The rows (AnchorPane's) inside vbxSets are as follows (learn_row.fxml):





Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnSetRow	AnchorPane	None	649x50px -fx-background-color: white	The root element which holds all elements described below and this pane will be inserted into vbxSets.
txtTitle	Text	None	Century Gothic 25px, #001421	A title which would be used to identify which flashcard set it is. The users can distinguish other sets using this title.
icnPlay	MaterialDesignIconView	playFlashcards()	Glyph Name: PLAY Size: 45px Fill: #c18303	This will be used to play the flashcard set. Using the 'play' button, it gives the learning a greater sense of enjoyment.
icnEdit	MaterialDesignIconView	editSet()	Glyph Name: PENCIL Size: 25px Fill: #001421	Allows for customisation of the set to update or change existing flashcards in the respective set.
icnDelete	MaterialDesignIconView	deleteSet()	Glyph Name: EXCLAMATION Size: 30px Fill: #6f0f0f	Removes the set from the database and system interface. Uses an exclamation rather than traditional cross, X, to make the system ever so slightly more friendlier.

When pressing icnEdit or the txtCreate from the main ‘learn’ view, you are redirected to another view which helps with the updating of the sets (learn_update.fxml):



Only difference is if editing, the textfields shown above will be filled with already existing terms.

Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnUpdateSet	AnchorPane	None	776 x 632 px -fx-background-color: white -fx-background-radius: 10	The root node holding all the elements below. Will be inserted into apnActivity.
txtNewTitle	Text	None	Sweetly Broken 72px, #c18303	A title which can be edited via a

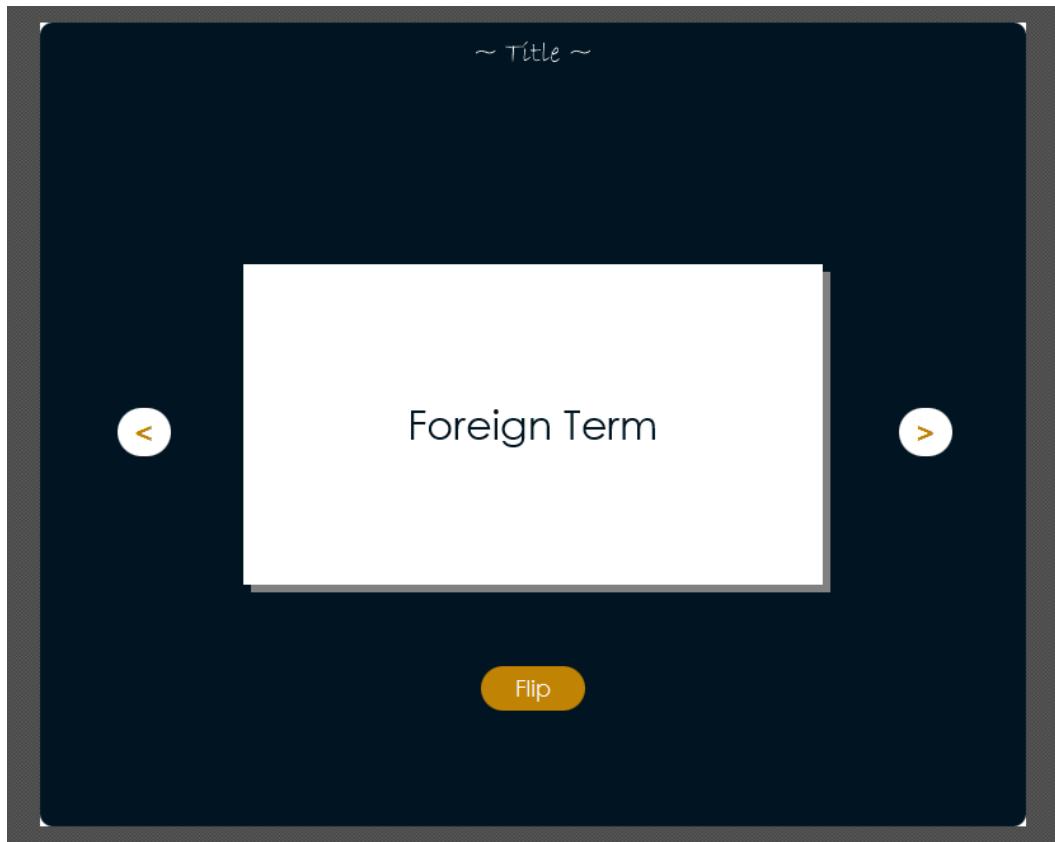
				dialog. This will be used later to distinguish a set from another set.
btnAdd	JFXButton	addCard()	Century Gothic 11px -fx-background-color: #001421 -fx-background-radius: 5px	Used to indefinitely add more flashcards to the set. The customisability should be high for the information to be relevant.
btnFinish	JFXButton	updateFinish()	Century Gothic 14px -fx-background-color: #c18303 -fx-background-radius: 15	Finish updating/creating the set. A user needs a way to finish.
scrpnCards	ScrollPane	None	675 x 483 px -fx-background-color: white	Used to hold vbxCards when the amount of flashcards get higher (since you can have infinite amount of flashcards).
vbxCards	VBox	None	673 x 481 px -fx-background-color: white	Will hold the rows which identify each card.

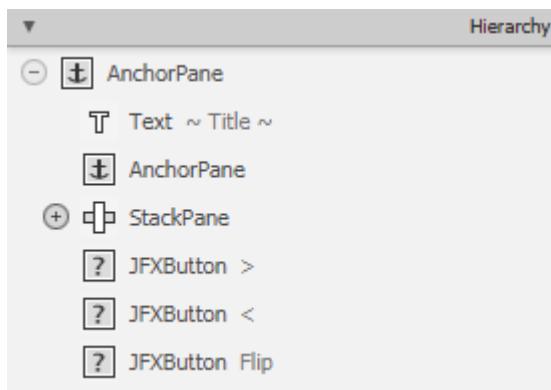
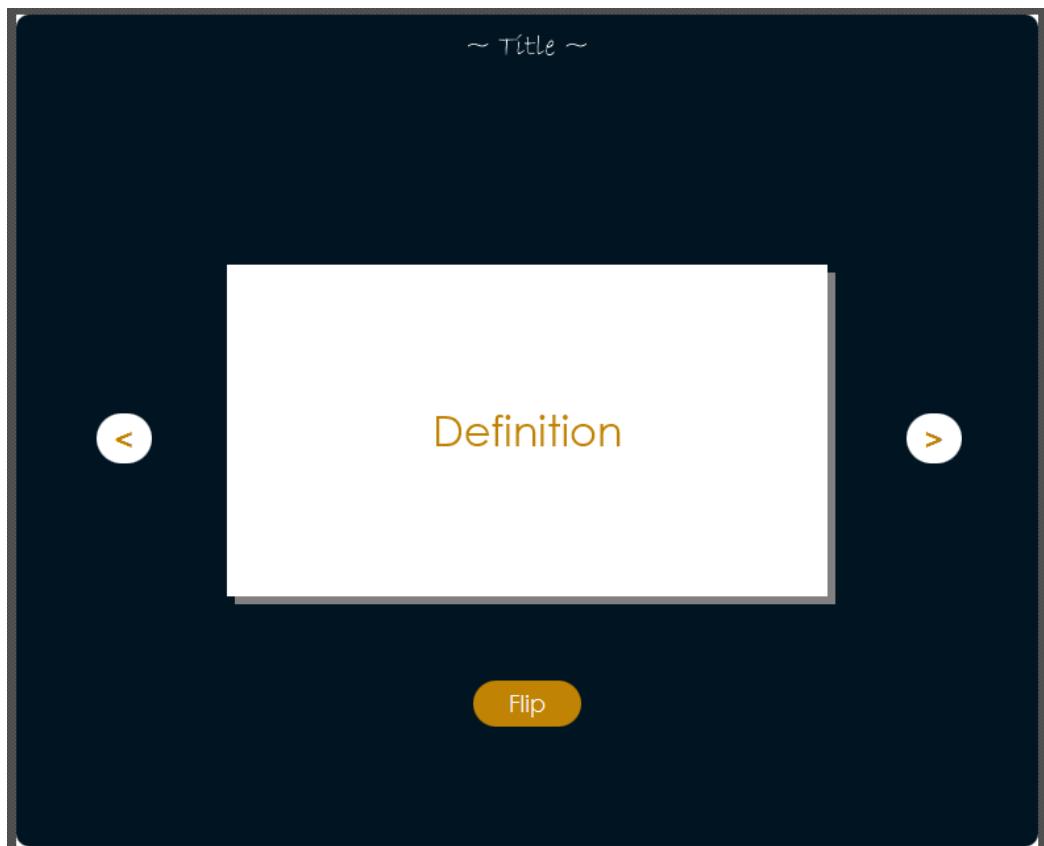
Inside vbxCards, there are rows which determine separate flashcards (learn_update_row.fxml).



Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnCardRow	AnchorPane	None	669 x 50 px	The root node which holds all the elements above and will be inserted into vbxCards.
btnRemove	JFXButton	removeCard()	System 11px -fx-background-color: #6f0f0f	If a user has misclicked add or if the user has simply changed their mind, they need a feature to revert back.
fldTerm	TextField	None	System 11px 260 x 22px	To enter the foreign term easily.
fldDef	TextField	None	System 11px 260 x 22 px	To enter the definition of the corresponding definition easily.

However, in the main ‘Learn’ screen where all the sets are listed, if the play button is pressed then the user will be allowed to use the flashcards (learn_flashcards.fxml).



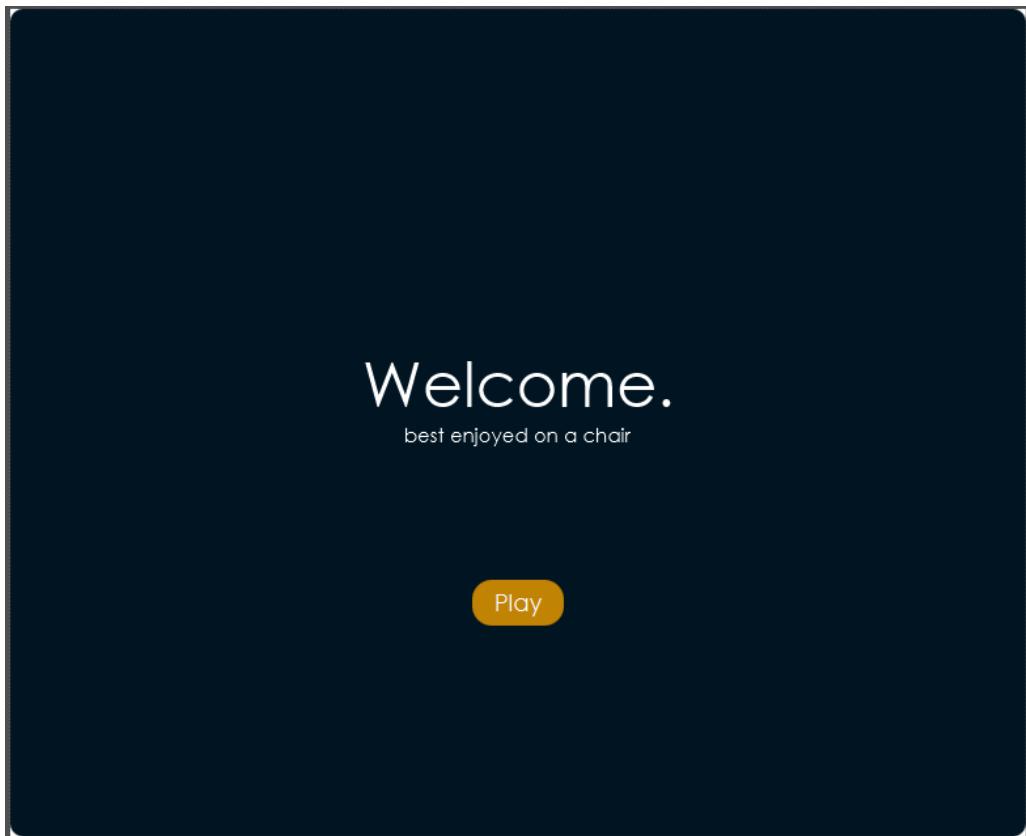


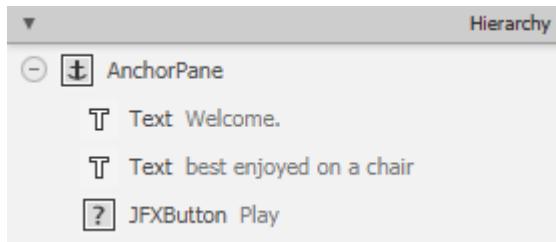
Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnFlashcards	AnchorPane	None	-fx-background-color: #001421 -fx-background-radius: 10	The root element holding all the elements above. Will be inserted into apnActivity.
txtSetTitle	Text	None	Bradley Hand ITC 23px, white	Reminds the user of what the flashcard set is about.
apnShadow	AnchorPane	None	456 x 252px -fx-background-color: grey	The shadow of the flashcard, giving a more 3d experience.
spnFlashcard	StackPane	flipCard()	456 x 252 px -fx-background-color: white	Used to hold the term and definition and is the visual representation of a 'flashcard' making the system a more real-world friendly feature to learning.

txtTerm	Text	None	Century Gothic 32 px Fill: #001421	The text which displays the foreign term.
txtDef	Text	None	Century Gothic 32 px Fill: #c18303	The text displaying the associated definition. In combination with the foreign term, it makes a link between the foreign term and definition making the learning of a flashcards system successful.
btnRight	JFXButton	nextCard()	System 19px(Bold), #c18303 -fx-background-radius: 40 -fx-background-color: white	Navigates to the next flashcard allowing the user to learn new terms.
btnLeft	JFXButton	prevCard()	System 19px(Bold), #c18303 -fx-background-radius: 40 -fx-background-color: white	Navigates to the previous flashcard allowing the user to move back.
btnFlip	JFXButton	flipCard()	Century Gothic 18 px, white -fx-background-color: #c18303 -fx-background-radius: 20	An explicit button to flip the card – clarity of the system.

Test

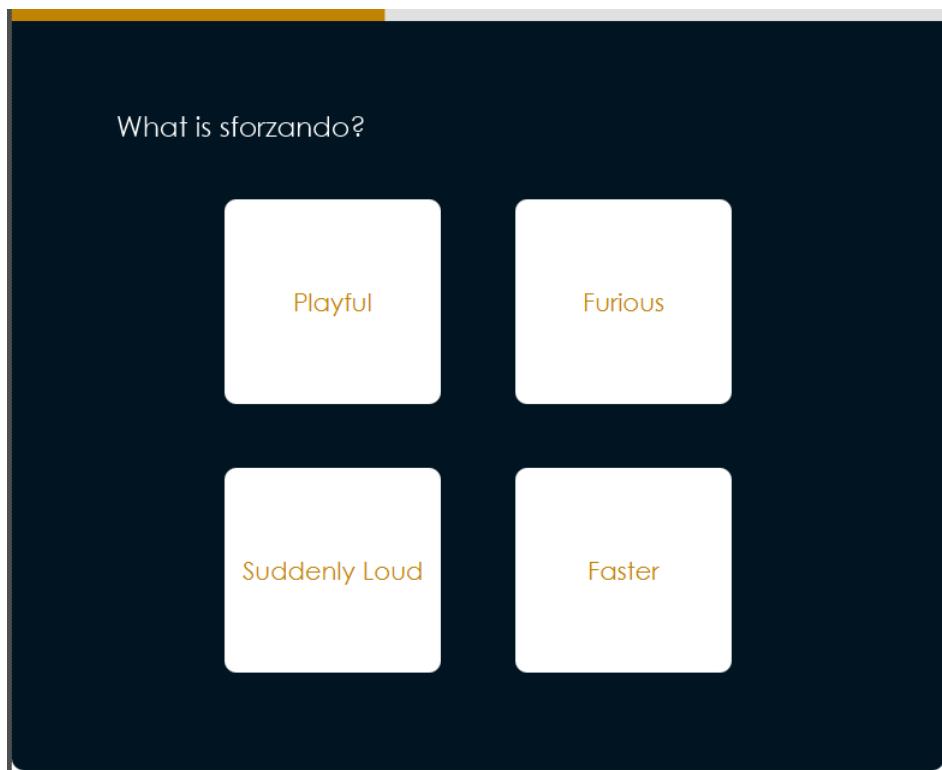
If from the navigation pane, the ‘Test’ feature is pressed, the respective button will also turn orange with the text/icon turning white, and, a new pane will be introduced to apnActivity (test.fxml).



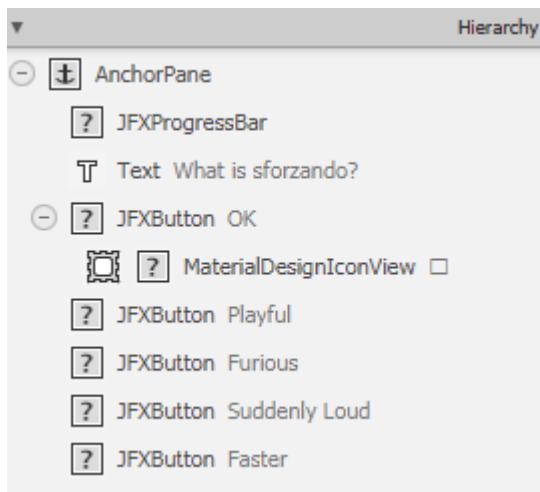
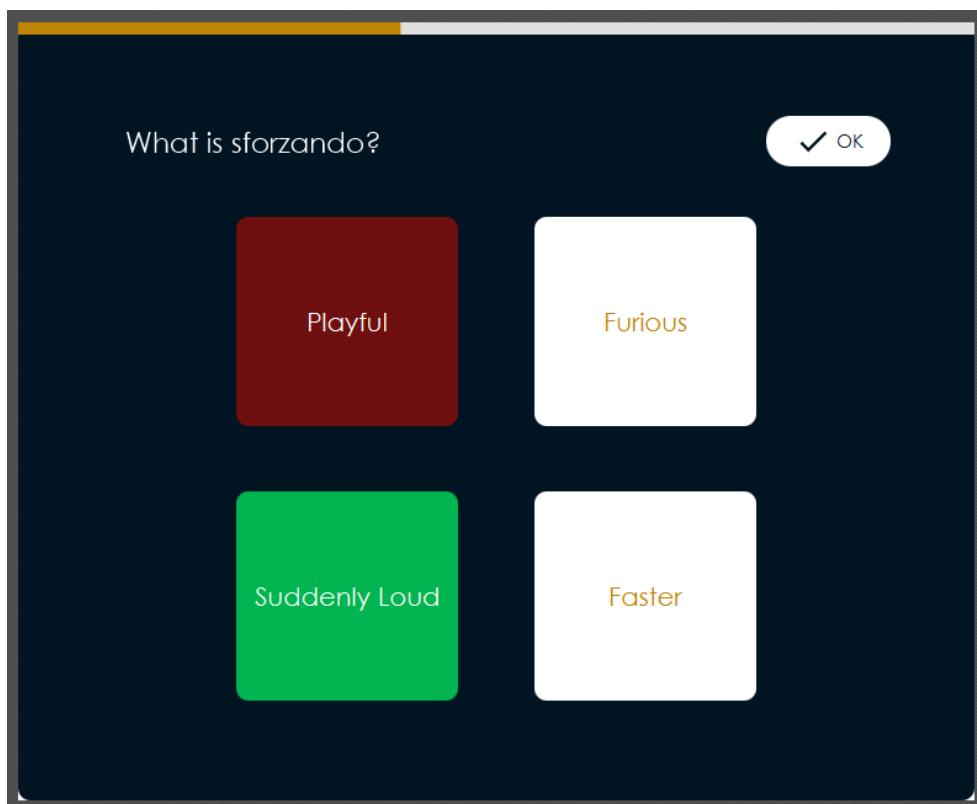


Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnTest	AnchorPane	None	AnchorPane	The root element holding all the elements above. Will be inserted into apnActivity.
txtWelcome	Text	None	Century Gothic 48px, White	Warmly welcomes the user to the testing feature of the system.
txtSubtitle	Text	None	Century Gothic, 15 px, White	A subtitle which cheers up the mood.
btnPlay	JFXButton	playQuiz()	Century Gothic 18px	A way for the user to start the quiz.

Then if you enter the quiz, it gets started right away (test_quiz.fxml):



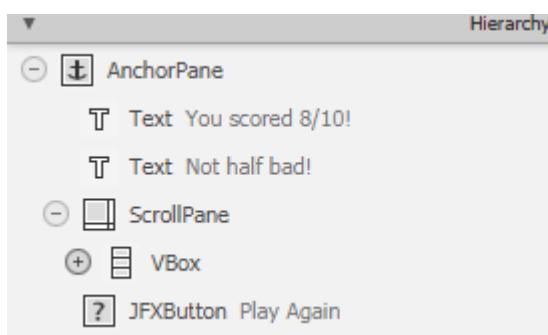
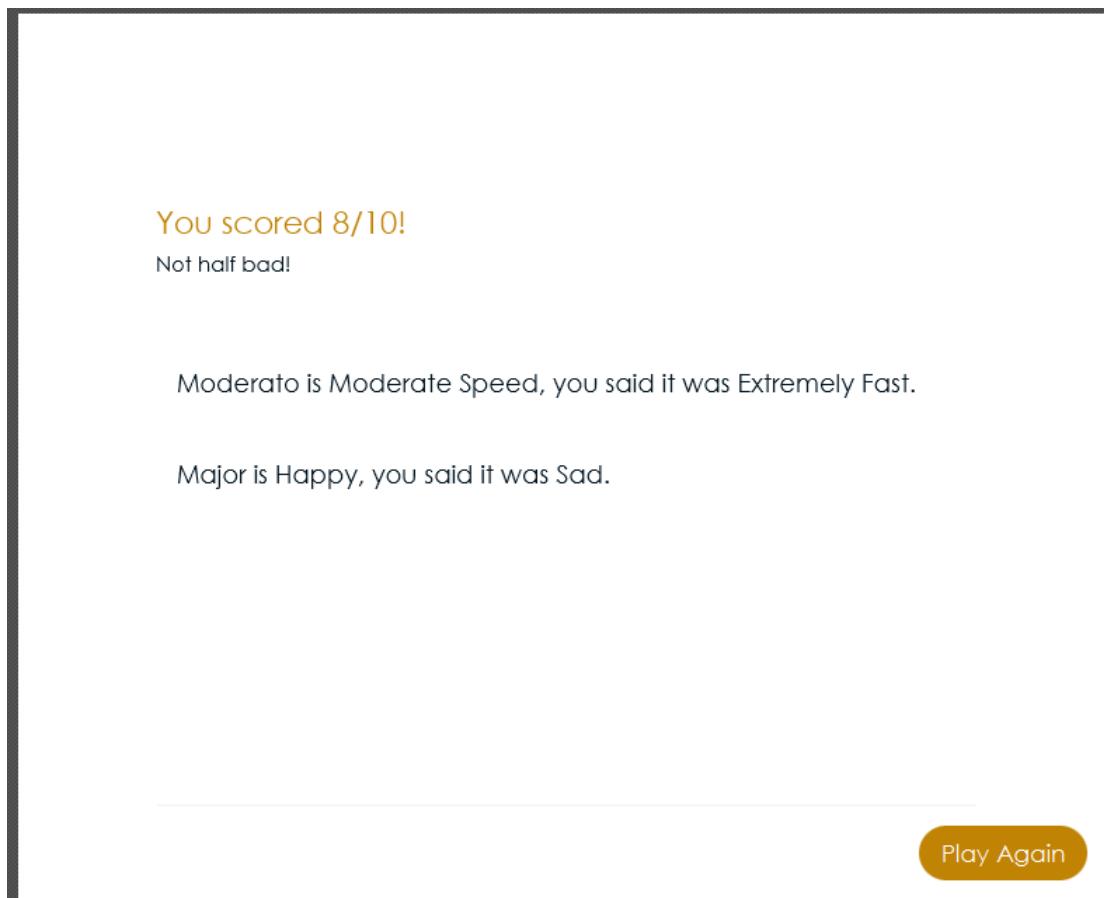
With simple feedback after user inputs their choice.



Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnQuiz	AnchorPane	None	776 x 632 px -fx-background-color: #001421 -fx-background-radius: 10	The root element holding all the nodes below. Will be inserted into apnActivity.
prbProgress	JFXProgressBar	None	Uses external css: jfx-progress-bar > .bar {-fx-	Used to determine the progress within the quiz, gives the

			background-color: #c18303} 776 x 10 px	user a sense of improvement.
txtQuestion	Text	None	Century Gothic 23px, White	Identifies the question clearly.
btnOk	JFXButton	nextQuestion()	Century Gothic 15px, #001421 -fx-background-color: white -fx-background-radius: 20	Will only be displayed after user has selected an answer – allows them to move to next question.
icnCheck	MaterialDesignIconView	None	Glyph Name: CHECK Fill: #001421 Size: 30	Gives a more friendly, visual way to identify the ‘ok’ button easier.
btnTwo	JFXButton	answer(1)	Century Gothic 21px, #c18303 -fx-background-color: white -fx-background-radius: 10	Choose the second option.
btnThree	JFXButton	answer(2)	Century Gothic 21px, #c18303 -fx-background-color: white -fx-background-radius: 10	Choose the third option.
btnFour	JFXButton	answer(3)	Century Gothic 21px, #c18303 -fx-background-color: white -fx-background-radius: 10	Choose the fourth option.

After completing 10 questions, the results screen will be displayed (test_results.fxml):

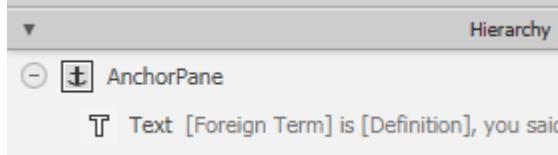


Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnResults	AnchorPane	None	776 x 632 px -fx-background-color: white -fx-background-radius: 10	The root element holding all the following elements. Will be inserted into apnActivity.
txtScore	Text	None	Century Gothic 22px, #c18303	Displays the score achieved on the test, will be stored for the 'Review' section of the program.
txtComment	Text	None	Century Gothic 15px, #001421	Gives a score-dependent comment

				making the system more personal.
scrpnCorrections	ScrollPane	None	-fx-background-color:white 586 x 336 px	ScrollPane containing the VBox holding the corrections, used to hold high amounts of corrections.
vbxCorrections	VBox	None	-fx-background-color:white 584 x 333 px	The container holding the corrections.
btnAgain	JFXButton	playQuiz()	Century Gothic 17px, white -fx-background-color:#c18303	Allows the user to take the test again if they want to immediately learn from their mistakes.

Inside the vbxCorrections, rows will be added which represents each question answered wrong and the correction that is associated.

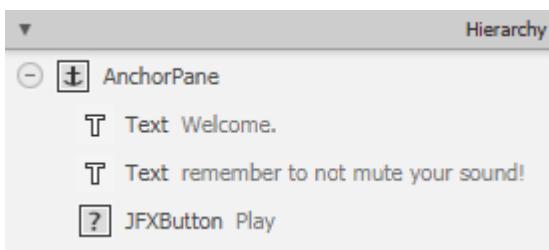
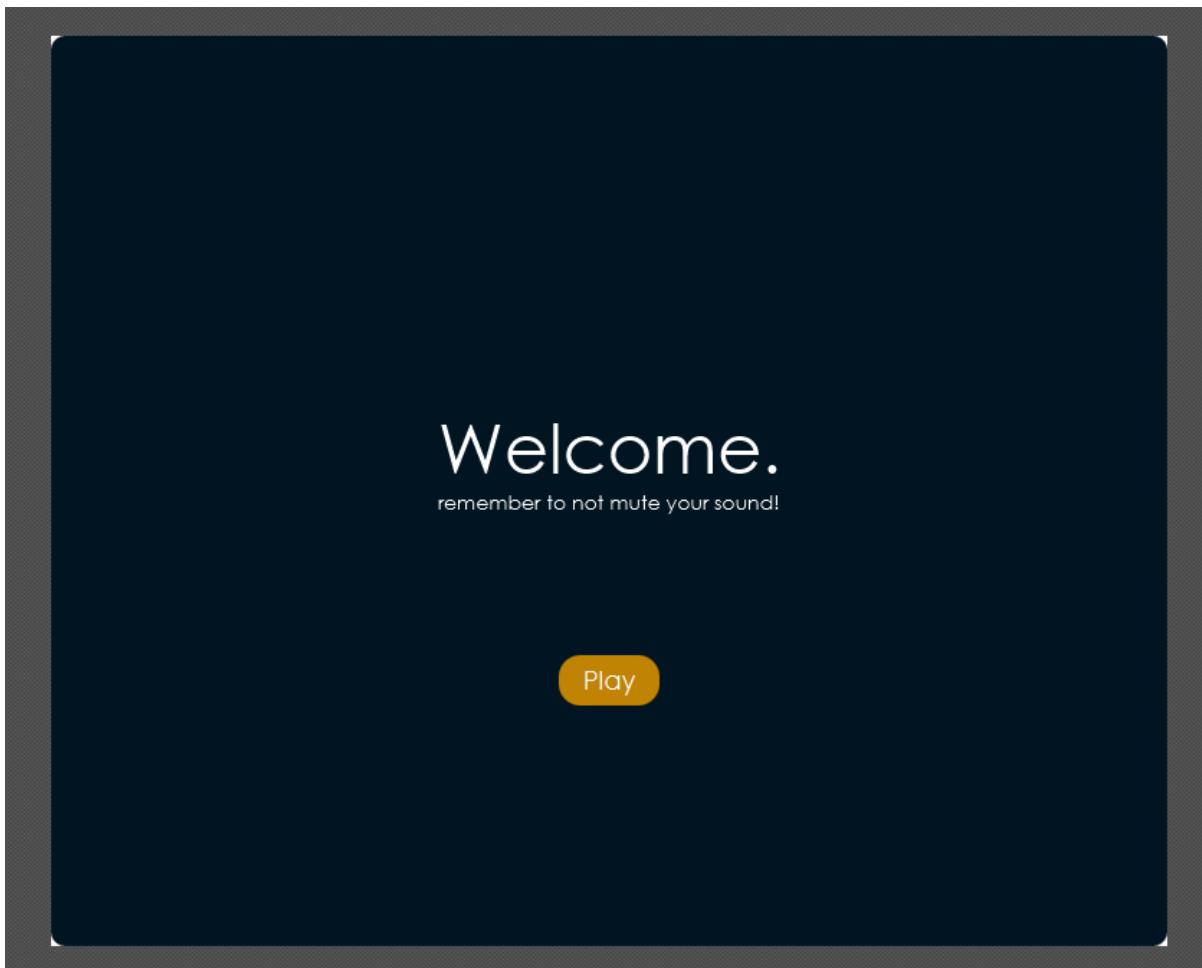
[Foreign Term] is [Definition], you said it was [Answered].



Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnFeedback	AnchorPane	None	579 x 65 px	The container holding the feedback and to be inserted into the vbox.
txtFeedback	Text	None	Century Gothic 18px, #001421	Gives the feedback.

Listen

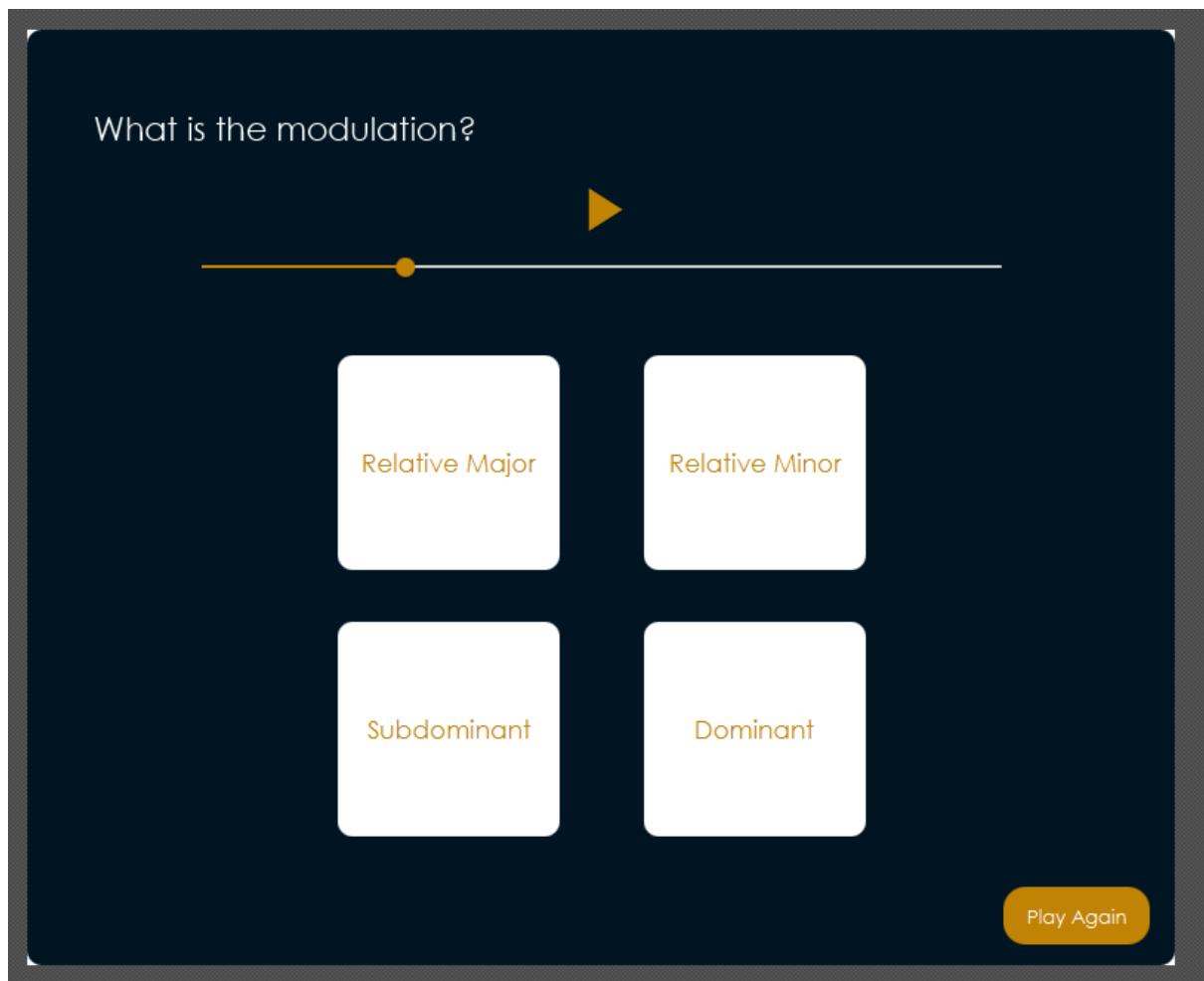
When pressing the ‘Listen’ button on the navigation pane, you will be redirected to the following screen with the respective button changing colours as described before (listen.fxml).



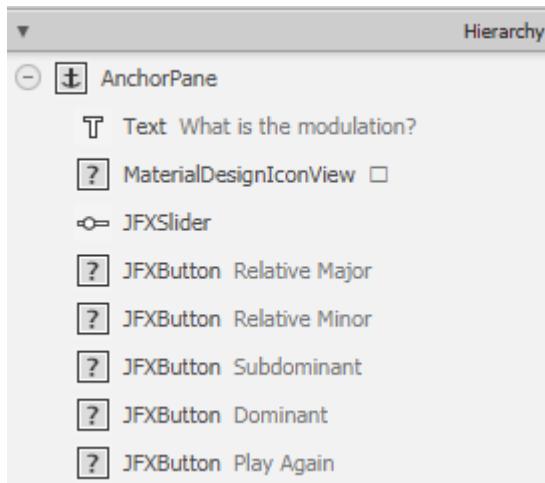
Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnListen	AnchorPane	None	776 x 632 px -fx-background-color: #001421 -fx-background-radius: 10	The root element holding all controls below. Also will be inserted to apnActivity.
txtWelcome	Text	None	Century Gothic 48px, White	A warming welcome to aural testing.
txtSubtitle	Text	None	Century Gothic 14px, White	A little reminder to enjoy the best

				experience of this feature.
btnPlay	JFXButton	playQuiz()	Century Gothic, 18px, White -fx-background-color: #c18303	To enter the test/quiz.

After choosing the play the quiz, you get started immediately (listen_quiz.fxml).



Note: there is only Grade 8 Modulation in the aural testing feature due to time constraints

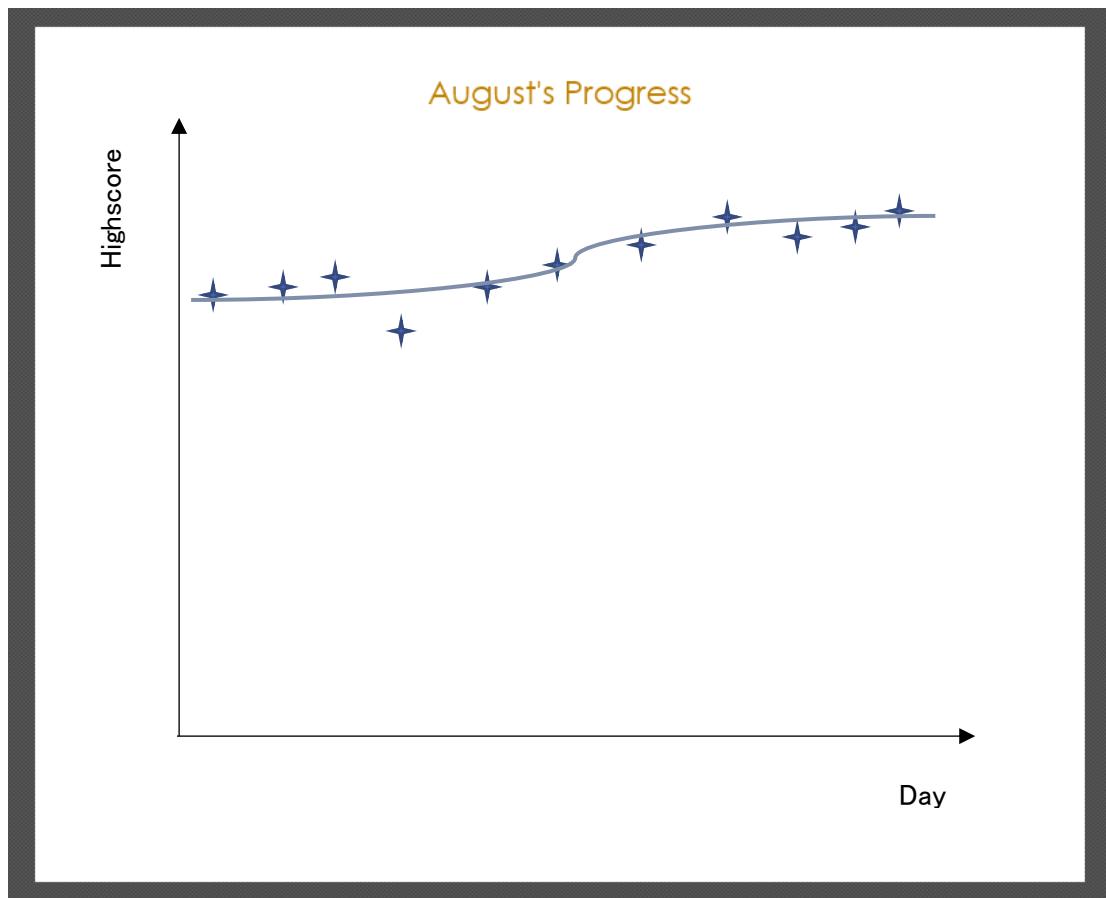


Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnQuiz	AnchorPane	None	-fx-background-color: #001421 -fx-background-radius: 10 776 x 632 px	The root element holding all the following elements. Will be inserted into apnActivity.
txtQuestion	Text	None	Century Gothic 22px, White	Displays the question for the user to know what they must be doing.
icnPlayPause	MaterialDesignIconView	auralPlayPause()	Glyph Name: PLAY / PAUSE Size: 50 Fill: #c18303	Plays or pauses to audio so it allows the user to take breaks and think.
sldSeek	JFXSlider	(will be binded to a listener)	Applied external css: jfx-slider .thumb { fx-background-color: #c18303; } jfx-slider .colored-track { -fx-background-color: #c18303; }	Seeking through audio so that they can go back and improve.
btnRelMaj	JFXButton	answer(0)	Century Gothic 17px, #c18303 -fx-background-color: white	Chooses relative major.

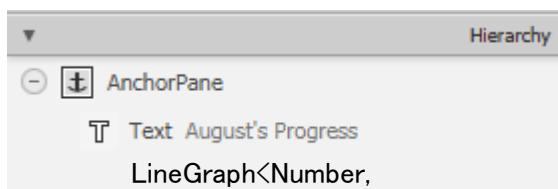
			-fx-background-radius: 10 150 x 145px	
btnRelMin	JFXButton	answer(1)	Century Gothic 17px, #c18303 -fx-background-color: white -fx-background-radius: 10 150 x 145px	Chooses relative minor.
btnSub	JFXButton	answer(2)	Century Gothic 17px, #c18303 -fx-background-color: white -fx-background-radius: 10 150 x 145px	Chooses subdominant.
btnDom	JFXButton	answer(3)	Century Gothic 17px, #c18303 -fx-background-color: white -fx-background-radius: 10 150 x 145px	Chooses dominant.
btnAgain	JFXButton	playAgain()	Century Gothic 13px, White -fx-background-color: #c18303 -fx-background-radius: 15	Enables the user to take another aural test so they have a variety of choices to improve from.

Review

Our review section allowing the users to look back on progress will look similar to this (review.fxml).



Note: this graph is not what the end graph will look like, however, the structure of it will be the same – the later graph will require data to be manually entered and displayed via code.



Component ID	Object Type	Algorithm(s) called	Design notes	Justification
apnReview	AnchorPane	None	-fx-background-radius: 10 -fx-background-color: white	The root element holding all the features below – will also be contained in apnActivity.

txtMonth	Text	None	Century Gothic 24px, #c18303	Needed to identify which month is used.
gphProgress	LineChart<Number, Number>	None	NumberAxis y-axis for score and NumberAxis x-axis for date in month	Graph is the crucial part of the review section, displays the progress over time and gives an extra layer of motivation to the user to improve.

Key Variables and Structures

For each section of my project, possibly decomposed into classes where suitable, there is importance to define and describe the justification of including key variables and structures. These include any local or class variables, constants, objects (and its respective class), database tables and fields that are involved with my system – I will explain any validation to these variables of structures if required.

As the variables within the visual aspect of my program has already been described and justified in my usability features; I will describe the classes (the controllers) which work together with each view and any other required classes.

Some variables will have the same names, but this is not an issue only when the variables are from different classes. There will be no confusion since they will only be named the same if the two variables from different classes have the same meaning and purpose.

Also, regarding the scope of class variables, they should all be private so that the data will be better encapsulated and information is better hidden; this allows for the data to be less error-prone and more robust.

Main.java

This is arguably the most important class which is used to start the program.

Name	Structure Type	Data Type	Scope	Justification	Validation
mainStage	Object	Stage	Class and Static	We will need to have a static copy of the Stage object so that we can later	None

				change the scene (which is the view that is shown to the user)	
root	Object	Parent	Subroutine	The first view to be displayed, which will be the root for all of its children nodes after it defined in the first view	Must be linked to a correctly named fxml file.
gradeScene	Object	Scene	Subroutine	This is the first scene, containing the root node, to be displayed onto the mainStage.	None

DAO.java

Name	Structure Type	Data Type	Scope	Justification	Validation
dbDir	Constant	String	Class and Static	This will store the directory of our database.	None
dbName	Constant	String	Class and Static	This will store the name of our database	Must be the name of the database.
dbUrl	Constant	String	Class and Static	The url required to get a connection to the database.	None
directory	Object	File	Subroutine	Used to create the directory (dbDir) if does not exist.	None
database	Object	File	Subroutine	Used to copy the database file from the program to the user's secondary storage (dbDir + // + dbName)	None
conn	Object	Connection	Subroutine	Will be returned as a connection to the database, a very important object that will be used to retrieve/modify data to and from the database.	Must not be null.

GradeController.java

This is the class that controls grade_select.fxml – the view which is firstly loaded by Main.java.

Name	Structure Type	Data Type	Scope	Justification	Validation
name	Variable	String	Class	This stores the user's name	Must not be empty

fldName	Object	JFXTextField	Class	This is the control node where the user initially enters their name.	Must not be empty
noNameAlert	Object	Alert	Subroutine	Will be displayed when the user has not entered a name into fldName	Must be displayed until user clicks an option.
selectedButton	Object	Button	Subroutine	Defines which button was clicked so that the grade can be decided.	Must not be null
loader	Object	FXMLLoader	Subroutine	The object used to load the fxml data into an object.	Must be correctly linked to an existing fxml file.
root	Object	Parent	Subroutine	The object which contains the fxml data for root.fxml – our next view.	Must be correctly linked to an existing fxml file.
rootController	Object	RootController	Subroutine	An object referencing the RootController.java class which will be used to initialize data: user's name and grade.	None
rootScene	Object	Scene	Subroutine	Contains the root node and will be displayed.	None

RootController.java

This will control root.fxml and is will be the ‘root’ controller of all the later views and their respective controllers since the later nodes will be added onto apnActivity – hence the name: RootController.

Name	Structure Type	Data Type	Scope	Justification	Validation
txtName	Object	Text	Class	Will display the user's name at the top of the navigation bar	Not zero length: empty.
apnActivity	Object	AnchorPane	Class	Will be used to share to other controllers so that other activities can be displayed such as Learn, Test, etc.	None.
spnLearn; spnTest; spnListen; spnReview	Object	StackPane	Class	Used for changing colours of the navigation bar	None.

icnLearn; icnTest; icnListen; icnReview	Object	MaterialDesignIconView	Class	Used for changing colours of the navigation bar.	None.
btnLearn; btnTest; btnListen; icnReview	Object	JFXButton	Class	Used for changing colours of the navigation bar.	None.
grade	Variable	int (Integer)	Class and Static	Stores the user's grade which can decide later content	Must not be empty.
name	Variable	String	Class and Static	Stores the user's name which is simply used for aesthetics and personalization.	Must not be zero length.
dateUpdater	Object	Timeline	Subroutine	Used to update the date and time of the program	None.
root (Learn)	Object	Parent	Subroutine (goLearn)	Will contain the contents of learn.fxml, will be used as a child of apnActivity.	None.
loader (Learn)	Object	FXMLLoader	Subroutine (goLearn)	Will be used to load the contents of learn.fxml into root.	Must be linked to an existing file.
controller (Learn)	Object	LearnController	Subroutine (goLearn)	Used to pass initial data to, so that the activity can fully function.	None
root (Test)	Object	Parent	Subroutine (goTest)	Will contain the contents of test.fxml, will be used as a child of apnActivity.	None.
loader (Test)	Object	FXMLLoader	Subroutine (goTest)	Will be used to load the contents of test.fxml into root.	Must be linked to an existing file.
controller (Test)	Object	TestController	Subroutine (goTest)	Used to pass initial data to, so that the activity can fully function.	None
root (Listen)	Object	Parent	Subroutine (goListen)	Will contain the contents of listen.fxml, will be used as a child of apnActivity.	None.
loader (Listen)	Object	FXMLLoader	Subroutine (goListen)	Will be used to load the contents of listen.fxml into root.	Must be linked to an existing file.

controller (Listen)	Object	ListenController	Subroutine (goListen)	Used to pass initial data to, so that the activity can fully function.	None
root (Review)	Object	Parent	Subroutine (goReview)	Will contain the contents of review.fxml, will be used as a child of apnActivity.	None.
loader (Review)	Object	FXMLLoader	Subroutine (goReview)	Will be used to load the contents of review.fxml into root.	Must be linked to an existing file.
controller (Review)	Object	ReviewController	Subroutine (goReview)	Used to pass initial data to, so that the activity can fully function.	None

For example, here, there are repeated names of root, loader and controller but these are restricted in their own respective subroutine; they perform the same jobs and are in isolated scopes so therefore, the conflict of names is no problem but rather simplify the problem moderately.

LearnController.java

The first controller of an activity we will describe is for the Learn activity.

Name	Structure Type	Data Type	Scope	Justification	Validation
Theory	Database Table	Table	Database	The table for each term for theory.	None
ID	Database Field	AutoNumber	Table Theory	A unique ID to identify each record (term)	Must exist; must be unique; must be a positive integer.
Term	Database Field	Short Text	Table Theory	Describes the foreign term.	Must exist and not zero length.
Definition	Database Field	Short Text	Table Theory	Describes the definition for the foreign term.	Must exist and not zero length.
Grade	Database Field	Number	Table Theory	Stores the grade level of each term.	Must be a positive integer between 1 and 8 inclusive.
Title	Database Field	Short Text	Table Theory	The sub-category of music theory which the term relates to; used to be a display	Must not be empty.

				title on the Learn screen.	
rootController	Object	RootController	Class	The RootController object to be used to set later activities (i.e relating to the flashcards sets)	None
titleSet	Object	(observable)Set	Class	Contains a unique set of titles for the terms that are in the database to be displayed.	None.
dao	Object	DAO	Class	Used to link to the database.	None
vbxSets	Object	VBox	Class	Rows of the sets will be added to this and displayed on the UI.	None.
apnLearn	Object	AnchorPane	Class	Used to pass on to later controllers so that they can return to the main learn screen.	None
conn	Object	Connection	Subroutine	A connection to the database which allows the system to communicate data to and from the database.	Not null
stmt	Object	PreparedStatement	Subroutine	The SQL statement to be executed onto the database – will be used to retrieve the titles for display.	Correctly typed.
rs	Object	ResultSet	Subroutine	Contains the result for our SQL query	Contains at least one item.
title	Variable	String	Subroutine	Contains the title for a term in table Theory to be stored in titleSet.	Must not be empty.
row	Object	Parent	Subroutine	Contains the content for learn_row.fxml; displaying the title of each set.	None
rowLoader	Object	FXMLLoader	Subroutine	Loads the content from learn_row.fxml to the object: row.	Must be correctly linked to the correct fxml file.

rowController	Object	LearnRowController	Subroutine	Controls the events and the view of each row.	None
titleDialog	Object	TextInputDialog	Subroutine	Allows the user to enter the title for the new set.	None
result	Object	Optional<String>	Subroutine	Stores the input of the user.	Must not be empty/zero length.
root	Object	Parent	Subroutine	Stores the content of learn_update.fxml – the UI to be used to create a new set.	None
loader	Object	FXMLLoader	Subroutine	Loads the content of learn_update.fxml into root.	Must be linked to the correctly typed file.
controller	Object	LearnUpdateController	Subroutine	Controls the view of learn_update.fxml.	None

LearnRowController.java

Each row representing a unique flashcard set will be controlled by objects of this class.

Name	Structure Type	Data Type	Scope	Justification	Validation
Theory	Database Table	Table	Database	Will be used to delete the set straight from the row.	None
Title	Database Field	Short Text	Table Theory	Used to delete the correct terms with the respective title.	Must not be empty.
rootController	Object	RootController	Class	Will be used to change activities and collect data.	None
learnController	Object	LearnController	Class	Other controllers will use this object to update the returned learn main screen after changes have been made.	None
apnLearn	Object	AnchorPane	Class	To be displayed when returned to main learn screen.	None

dao	Object	DAO	Class	Connects to the database.	None.
txtTitle	Object	Text	Class	Displays the title of the set.	Must not contain zero length text.
apnSetRow	Object	AnchorPane	Class	Holds all the nodes contained in learn_update_row.fxml	None
root (flashcards)	Object	Parent	Subroutine	Used to contain all the fxml data for learn_flashcards.fxml.	None.
loader (flashcards)	Object	FXMLLoader	Subroutine	Loads the fxml data into root	Must be correctly linked.
controller (flashcards)	Object	LearnFlashcardController	Subroutine	Controls what happens in the flashcard activity	Must be controlling the correct view.
root (editing set)	Object	Parent	Subroutine	Used to contain all the fxml data for learn_update.fxml.	None.
loader (editing set)	Object	FXMLLoader	Subroutine	Loads the fxml data into root	Must be correctly linked.
controller (editing set)	Object	LearnUpdateController	Subroutine	Controls what happens in the editing screen (i.e. event handling, changes to UI etc.)	Must be controlling the correct view.
confirmAlert	Object	Alert	Subroutine	Used to warn the user about permanently deleting a set.	None
result	Object	OptionalResult<ButtonType>	Subroutine	Identifies which button the user has clicked on the dialog (i.e yes or no)	Must not be empty.
conn	Object	Connection	Subroutine	Provides a connection to the database.	Not null.
stmt	Object	PreparedStatement	Subroutine	Stores the statement of SQL to be executed (delete set)	None.
vbxSets	Object	VBox	Subroutine	The parent of each row representing each set: the VBox storing them. Used to	Must be the correct node.

				remove the set on the interface.	
--	--	--	--	----------------------------------	--

LearnUpdateController.java

This class will provide the functionality to make changes to flashcard sets.

Name	Structure Type	Data Type	Scope	Justification	Validation
Theory	Database Table	Table	Database	This table is a key entity to be used for the updating for our theory terms	None
Term	Database Field	Field	Table Theory	The theory term is stored here	Not empty i.e. zero length
Definition	Database Field	Field	Table Theory	The respective definition is stored here	Not empty
Title	Database Field	Field	Table Theory	Title allows us to group terms into respective categories and use it to 'title' their theme	Not empty
Grade	Database Field	Field	Table Theory	Will need to be used when creating a new set.	Not empty.
rootController	Object	RootController	Class	Provides data about the user and is also used to change the activity.	None
learnController	Object	LearnController	Class	Used to update the UI of learn.fxml after editing.	None
apnLearn	Object	AnchorPane	Class	Used to return to the main learn screen.	None
dao	Object	DAO	Class	Allows access to the database.	None
controllerMap	Object	(observable)HashMap	Class	Maps each controller to its respective UI node. This makes the removal of each row easier.	None
txtNewTitle	Object	Text	Class	Displays the title of the set.	Must not contain an empty string

vbxCards	Object	VBox	Class	Holds the individual rows representing cards.	None
scrpnCards	Object	ScrollPane	Class	Allows vbxCards to extend indefinitely if more cards get added.	None
conn (initialize)	Object	Connection	Subroutine	Connects to database	Not null
stmt (initialize)	Object	PreparedStatement	Subroutine	Stores the SQL statement to be executed: to collect all the existing terms with that title.	None.
rs	Object	ResultSet	Subroutine	Holds the result of the statement, i.e the existing terms.	Not empty
term	Variable	String	Subroutine	Holds the term of the current record.	None
definition	Variable	String	Subroutine	Holds the definition of the current record.	None
controller	Object	LearnUpdateRowController	Subroutine	Controls the events for each row (each card)	None
row	Object	AnchorPane	Subroutine	Contains data for learn_update_row.fxml	None
loader	Object	FXMLLoader	Subroutine	Loads fxml data from learn_update_row.fxml to row.	Correctly linked to learn_update_row.fxml
controller	Object	FXMLLoader	Subroutine	Controls the events of learn_update_row.fxml	None.
conn (finish)	Object	Connection	Subroutine	Connects to database	Not null
deleteStmt	Object	PreparedStatement	Subroutine	Stores the SQL statement to delete the terms with the same title.	None
stmt (finish)	Object	PreparedStatement	Subroutine	Stores the SQL statement to insert all the new (including the existing terms) into the database.	None

LearnUpdateRowController.java

Controls each row stored in vbxCards

Name	Structure Type	Data Type	Scope	Justification	Validation
fldTerm	Object	TextField	Class	Allows the user to enter the term of a flashcard.	None
fldDef	Object	TextField	Class	Allows the user to enter the definition of a flashcard	None
updateController	Object	LearnUpdateController	Class	Refers to the respective LearnUpdateController controlling the parent nodes. Allows the removal of a controller of a flashcard when removed – so that it will not be used to add into the database.	None
vbxCards	Object	VBox	Subroutine	Holds the flashcard rows.	None

LearnFlashcardController.java

Controller for the most crucial part of our section: the flashcards.

Name	Structure Type	Data Type	Scope	Justification	Validation
Theory	Database Table	Table	Database	The entity needing to be accessed are the theory terms since they will be used for our flashcards	None
Term	Database Field	Field	Table Theory	The terms are used for our question of the flashcard	Not empty
Definition	Database Field	Field	Table Theory	The respective definition of the flashcard	Not empty
Title	Database Field	Field	Table Theory	Used to locate the relevant terms of the database.	Not empty.
flashcardList	Object	(observable)ArrayList	Class	Stores the flashcards. (A LinkedList would	None

				keep the position and order of the flashcards but the retrieval of them from the database would just lose the original order.)	
dao	Object	DAO	Class	Allows connectivity to the database.	None
flashcardIndex	Variable	int	Class	Stores the index of the flashcard set – identifying an individual flashcard	None
questionVisible	Variable	boolean	Class	Determines whether the question (foreign term) or the answer (definition) is visible	None
title	Variable	String	Class	Stores the title of the current flashcard set.	Not empty.
apnLearn	Object	AnchorPane	Class	Used to return back to the main learn screen	None
rootController	Object	RootController	Class	Allows to change activity and also provides data about the user	None
txtSetTitle	Object	Text	Class	Displays the title of the set	None
txtTerm	Object	Text	Class	Displays the term of the flashcard	None
txtDef	Object	Text	Class	Displays the definition of the flashcard	None
btnLeft	Object	JFXButton	Class	Used to navigate to the previous flashcard	None
btnRight	Object	JFXButton	Class	Used to navigate to the next flashcard.	None
apnFlashcards	Object	AnchorPane	Class	The root parent node contained by learn_flashcards.fxml. Can be used to set up key inputs for this section – easier navigation.	None
conn	Object	Connection	Subroutine	Allows us to connect to the database	Is not null
stmt	Object	PreparedStatement	Subroutine	Stores the SQL statement to collect	None

				all the terms from a specific title	
rs	Object	ResultSet	Subroutine	Holds the result of the SQL query	Not empty
foreignTerm	Variable	String	Subroutine	Stores the string for the term of the current record.	None
definition	Variable	String	Subroutine	Stores the string for the definition of the current record.	None
currentCard	Object	Flashcard	Subroutine	Holds all the data for the current flashcard to be displayed.	None
clickedButton	Object	JFXButton	Subroutine	Identifies which of the navigation buttons has been clicked.	None

Flashcard.java

The object for one flashcard will be of this type.

Name	Structure Type	Data Type	Scope	Justification	Validation
foreignTerm	Variable	String	Class	Holds the foreign term of this flashcard	None (checks if it were empty are already done in earlier classes)
definition	Variable	String	Class	Holds the definition of this flashcard	None (as above)

TestController.java

Controls the welcome/opening screen of the test section.

Name	Structure Type	Data Type	Scope	Justification	Validation
rootController	Object	RootController	Class	Used to change activity and retrieve additional user information	None
root	Object	Parent	Subroutine	Stores the fxml data for the quiz part.	None
loader	Object	FXMLLoader	Subroutine	Loads the data from test_quiz.fxml into root.	Correctly linked to the correct file.

controller	Object	TestQuizController	Subroutine	Used to initialize data.	None
------------	--------	--------------------	------------	--------------------------	------

TestQuizController.java

The quiz part of the test feature.

Name	Structure Type	Data Type	Scope	Justification	Validation
Theory	Database Table	Table	Database	Used to retrieve data for the theory terms.	None
Term	Database Field	Short Text	Table Theory	Holds the foreign term	Not zero length
Definition	Database Field	Short Text	Table Theory	Holds the respective definition for this foreign term	Not zero length
Grade	Database Field	Number	Table Theory	Holds the grade of the user	An integer between 1 and 8 inclusive
Title	Database Field	Short Text	Table Theory	Holds the title (category) the term relates to.	Not empty
K_Coefficient	Database Field	Number	Table Theory	Identifies how OK the user is on a specific term.	Smaller or equal to 1.
rootController	Object	RootController	Class	Allows change of activity and retrieval of user data.	None
dao	Object	DAO	Class	Provides access to database	None
rand	Object	Random	Class	Gives ease to produce random numbers.	None
questionIndex	Variable	int	Class	Identifies which question the user is on.	None
theoryQuestionList	Object	LinkedList	Class	Stores the theory questions, in order of insertion.	None
feedbackList	Object	LinkedList	Class	Stores the feedback to be later displayed, in order of insertion.	None

correctTerms	Object	(observable)ArrayList	Class	Used to later update the K_Coefficients of the correct terms.	None
questionAnswered	Variable	boolean	Class	Stores whether the user has answered the question yet or not, determining the functionality of specific nodes.	None
txtQuestion	Object	Text	Class	Displays the question	None
btnOk	Object	JFXButton	Class	Allows the user to move onto the next question	None
btnOne, btnTwo, btnThree, btnFour	Object	JFXButton	Class	A selection of answers are displayed, where one is correct, for the user to select.	None
prbProgress	Object	JFXProgressBar	Class	Displays visually the progress within the quiz.	None
answerButtons	Object	Array (JFXButton[])	Class	Holds all the answer buttons: btnOne, btnTwo, btnThree, btnFour. This allows us to make changes to allow the buttons more efficiently.	None
conn	Object	Connection	Subroutine	Provides a connection to the database	Not null.
priorityCntStmt	Object	PreparedStatement	Subroutine	SQL query to calculate the number of high priority terms with low k_coefficient values	None
priorityStmt	Object	PreparedStatement	Subroutine	SQL query to collect the high priority terms.	None
cntStmt	Object	PreparedStatement	Subroutine	SQL query to calculate total number of terms; in combination with the rand object, you	None

				can select a random record.	
rndStmt	Object	PreparedStatement	Subroutine	SQL query to select the all the terms in the table and then randomly select them; since there is no predefined way to collect random records repeatedly using just SQL.	None
cntRs	Object	ResultSet	Subroutine	Stores the result of the cntStmt query – number of terms in database.	Not empty
rndRs	Object	ResultSet	Subroutine	Collects the terms and definitions in the database (stores them in this result set)	Not empty
priorityCntStmt	Object	ResultSet	Subroutine	Stores number of high priority terms	Not empty
priorityStmt	Object	ResultSet	Subroutine	Stores the high priority terms and definitions	Not empty
priorityLength	Variable	int	Subroutine	Stores number of high priority terms.	Is it higher than 0
length	Variable	int	Subroutine	Stores the number of terms overall.	Is it higher than 0
question (priority)	Variable	String	Subroutine	Stores the high priority term (i.e. question for the quiz)	None
question (other)	Variable	String	Subroutine	Stores the question (the term) of a normal, non-high priority term.	None
answers (priority), answers (other)	Object	Array (String[])	Subroutine	Stores the possible answers for this question where 3 are wrong and 1 is correct.	None
answerIndex (priority), answerIndex (other)	Variable	int	Subroutine	Index of answers array which contains the correct answer	None

randomRow (priority), randomRow (other)	Variable	int	Subroutine	Index of the random row to be selected	Between 0 (inclusive) and length (number of terms – exclusive).
theoryQuestion	Object	TheoryQuestion	Subroutine	Stores the current question to be loaded	None
currentQuestion	Object	TheoryQuestion	Subroutine	Stores the current question to identify the answers	None
clickedButton	Object	JFXButton	Subroutine	Identifies which button was clicked	None
root	Object	Parent	Subroutine	The results screen will be held in this object (data for test_results.fxml)	None
loader	Object	FXMLLoader	Subroutine	Loads the data from a test_results.fxml into root.	Must be correctly linked to the file.
controller	Object	TestResultsController	Subroutine	The object which controls the events, UI and data of the results screen.	None

TestResultsController.java

Name	Structure Type	Data Type	Scope	Justification	Validation
Theory	Database Table	Table	Database	The questions and answers are defined in this table	None
K_Coefficient	Database Field	Field	Table Theory	Used to update the K_Coefficient to adapt next test(s)	Not empty and an integer larger than zero
Term	Database Field	Field	Table Theory	Used to identify a record/records	Not empty
Score	Database Table	Table	Database	This holds all the data of the results made from tests	None
Highscore	Database Field	Field	Table Score	Stores the highscore of a certain date	Not empty

ScoreDay	Database Field	Field	Table Score	Stores the day of the month that the highscore was made	Not empty
ScoreMonth	Database Field	Field	Table Score	Stores the month of the year	Not empty
ScoreYear	Database Field	Field	Table Score	Stores the year	Not empty
comments	Object	Array (String[])	Class	Stores a collection of comments to be displayed dependent of the user's score.	None
feedbackList	Object	LinkedList	Class	Stores the feedback for the wrong questions	None
correctTerms	Object	(observable)List	Class	Stores the correctly answered terms from the test.	None
rootController	Object	RootController	Class	RootController object used to navigate around the system	None
result	Variable	int	Class	Stores the result of the test out of 10	None
dao	Object	DAO	Class	Allows access to the Access database.	None
txtScore	Object	Text	Class	Displays the score (result)	None
txtComment	Object	Text	Class	Displays the comment	None
vbxFeedbacks	Object	VBox	Class	Holds the separate feedback control nodes.	None
feedback	Object	Feedback	Subroutine	Holds each individual feedback object	None
txtFeedback	Object	Text	Subroutine	Displays the feedback and is contained within vbxFeedbacks	None
conn	Object	Connection	Subroutine	Connects to the database.	Not null
checkKStmt	Object	PreparedStatement	Subroutine	Checks the K_Coefficient of an individual term	None
higherKStmt	Object	PreparedStatement	Subroutine	Increments the K_Coefficient of the term (for correctly answered questions)	None
lowerKStmt	Object	PreparedStatement	Subroutine	Decrements the K_Coefficient of the	None

				term (for wrongly answered questions)	
date	Object	Array (int[])	Subroutine	An array containing the date, month, and year	None
oldScoreStmt	Object	PreparedStatement	Subroutine	Queries the current highscore of the day	None
insScoreStmt	Object	PreparedStatement	Subroutine	Inserts a new highscore record into table Score if no score has been made today.	None
updScore	Object	PreparedStatement	Subroutine	Updates the highscore of today if has improved	A score already exists for this day
now	Object	Calendar	Subroutine	Used to identify 'now' in time.	None
day	Variable	int	Subroutine	Stores the day of the month (1 to 28/29/30/31)	None
month	Variable	int	Subroutine	Stores the month of the year (1 to 12)	None
year	Variable	int	Subroutine	Stores the current year	None
root	Object	Parent	Subroutine	Stores the start screen i.e test.fxml	None
loader	Object	FXMLLoader	Subroutine	Loads the fxml data into root	Correctly linked to test.fxml
controller	Object	TestController	Subroutine	Controls what happens in the test start screen	None

TheoryQuestion.java

This is a model object which gives the structure of what a theory question requires.

Name	Structure Type	Data Type	Scope	Justification	Validation
question	Variable	String	Class	A theory question requires its intrinsic question	None
answers	Object	Array (String[])	Class	Holds all the possible answers for this question.	None
answerIndex	Variable	int	Class	Determines which item in the answers array is correct.	None

Feedback.java

Each wrongly answered question in the test will create a feedback object to hold the data.

Name	Structure Type	Data Type	Scope	Justification	Validation
question	Variable	String	Class	Question which the user answered wrong	None
wrongAnswer	Variable	String	Class	Answer inputted here.	None
correctAnswer	Variable	String	Class	Correct answer is stored here.	None

ListenController.java

The start screen for the listen section will be controlled by this.

Name	Structure Type	Data Type	Scope	Justification	Validation
rootController	Object	RootController	Class	Used to change activity	None
root	Object	Parent	Subroutine	Holds the fxml data for listen_quiz.fxml	None
loader	Object	FXMLLoader	Subroutine	Loads data from listen_quiz.fxml into root.	Correctly linked to listen_quiz.fxml
controller	Object	ListenQuizController	Subroutine	Controls the quiz part of the listen section	None.

ListenQuizController.java

This object controls the section dedicated to aural testing.

Name	Structure Type	Data Type	Scope	Justification	Validation
Aural	Database Table	Table	Database	This entity represents a single aural test	None
File	Database Field	Field	Table Aural	Stores the file path to the audio file	Not empty
Modulation	Database Field	Field	Table Aural	Stores the modulation (answer) of the respective aural test.	Not empty

rootController	Object	RootController	Class	Used to change activity and collect user data	None
dao	Object	DAO	Class	Allows access to the database	None
currentTest	Object	AuralTest	Class	Holds the data for the current aural test	None
answered	Variable	boolean	Class	Stores the value of whether the test has been answered or not	None
btnRelMaj, btnRelMin, btnSub, btnDom	Object	JFXButton	Class	Button objects for the multiple choice options that can be used to answer	None
icnPlayPause	Object	MaterialDesignIconView	Class	Used to play or pause the audio	None
sldSeek	Object	JFXSlider	Class	Allows user to seek through the audio.	None
conn	Object	Connection	Subroutine	Connects to the database	Not null
cntStmt	Object	PreparedStatement	Subroutine	SQL query to count the number of aural tests available	None
cntRs	Object	ResultSet	Subroutine	Stores the result of the query stored by cntStmt	Not empty
rmdStmt	Object	PreparedStatement	Subroutine	SQL query to select the aural tests to be randomly chosen for this test	None
rndRs	Object	ResultSet	Subroutine	Stores the possible aural tests for the section	Not empty
length	Variable	int	Subroutine	Holds the number of aural tests available; or the total length of the database.	None
randomRow	Variable	int	Subroutine	An index specifying a random row to be used as the aural test.	None
currentDuration	Object	Duration	Subroutine	Used to locate the current duration point in media	None

totalDuration	Object	Duration	Subroutine	Used to store the total duration of the media	Larger than zero
percentageIn	Variable	double	Subroutine	Holds the result of the calculation to determine the percentage within the media which will be used to set the value of sldSeek	None
newDuration	Object	Duration	Subroutine	Identifies the new duration that the user has滑动到.	None
inputBtn	Object	Button	Subroutine	Determines which button has been used to input the answer.	None
root	Object	Parent	Subroutine	Stores the fxml data for to play the quiz again.	None
loader	Object	FXMLLoader	Subroutine	Will be used to load data from listen_quiz.fxml into root	Correctly linked to listen_quiz.fxml
controller	Object	ListenQuizController	Subroutine	Controls the events, UI, and data flow of the listen quiz	None

AuralTest.java

This is the model to specify the structure of the listen section's aural test.

Name	Structure Type	Data Type	Scope	Justification	Validation
audioFile	Variable	String	Class	Stores the name of the audio file to be used to output music.	None
auralPlayer	Object	MediaPlayer	Class	Object which holds the data for the aural test audio	None
modulation	Variable	String	Class	Contains the answer for the aural test – i.e. the modulation	None
duration	Variable	double	Class	Specifies the length of the audio (how long it lasts for in time)	None

ReviewController.java

Our review section displaying graphs for the user's progress will be controlled by this object class.

Name	Structure Type	Data Type	Scope	Justification	Validation
Score	Database Table	Table	Database	The scores will be used as the data points of our graph	None
Highscore	Database Field	Field	Table Score	Stores the value of the highscore at a particular day in time	Not empty
ScoreDay	Database Field	Field	Table Score	Stores the day of the month of the record	Not empty
ScoreMonth	Database Field	Field	Table Score	Stores the month of the year of the record	Not empty
ScoreYear	Database Field	Field	Table Score	Stores the year of the respective record	Not empty
rootController	Object	RootController	Class	Used to retrieve user information and change activity	None
dailySeries	Object	XYChart.Series	Class	Stores the chart data points for the graph measuring by day	None
monthlySeries	Object	XYChart.Series	Class	Stores the chart data points for the graph measuring by month	None
yearlySeries	Object	XYChart.Series	Class	Stores the chart data points for the graph measured by year	None
dao	Object	DAO	Class	Allows access to the database	None
apnDaily	Object	AnchorPane	Class	The container which contains the daily graph	None
apnMonthly	Object	AnchorPane	Class	Container of monthly graph	None
apnYearly	Object	AnchorPane	Class	Contains the graph of progress over years.	None
txtProgress	Object	Text	Class	Gives a title to the section.	None
now (date), now(month), now (year)	Object	Calendar	Subroutine	Gives the date of the current time that is now.	None

month	Variable	String	Subroutine	Stores the current month name	None
yAxis(daily), yAxis(monthly), yAxis(early)	Object	NumberAxis	Subroutine	Holds the data for the yAxis of each graph	Values between 0 and 10 inclusive and integer.
xAxis (daily)	Object	NumberAxis	Subroutine	Holds the x axis data representing each day of the month	Must only allow values between 1 and 31 inclusive and integer
dailyChart	Object	LineChart	Subroutine	The object of the actual chart to be placed onto UI	None
xAxis (monthly)	Object	CategoryAxis	Subroutine	Holds the x axis data representing each month of the year	Must be only one of the specified months (e.g. Jan or Feb etc)
monthlyChart	Object	LineChart	Subroutine	UI node object of the monthly graph to be displayed	None
yAxis	Object	NumberAxis	Subroutine	Holds the x axis data for any year.	Must be higher than release date year (2018 or 2019)
yearlyChart	Object	LineChart	Subroutine	The object of the yearly graph to be displayed onto UI	None
conn	Object	Connection	Subroutine	Connects to the database allowing it to be written to or read from.	Not null
dailyStmt	Object	PreparedStatement	Subroutine	Queries the score of an individual day of the month	None
monthlyStmt	Object	PreparedStatement	Subroutine	Queries the scores of different months of the same year	None
yearlyStmt	Object	PreparedStatement	Subroutine	Queries the scores of the different years in time	None
now (scores)	Object	Calendar	Subroutine	Used to get the current date	None
dailyRs	Object	ResultSet	Subroutine	Results of the dailyStmt will be stored here	Not empty

dayPoint	Object	XYChart.Data	Subroutine	A point of data to represent the score at a particular day	None
monthlyRs	Object	ResultSet	Subroutine	Holds the result of monthlyStmt	Not empty
lastMonth	Variable	String	Subroutine	Stores the current month in hand (last month that was accessed)	None
mTotal	Variable	double	Subroutine	Holds the total of all the scores in the month. The double data type allows for decimal – so a more precise average.	None
mFreq	Variable	double	Subroutine	Holds the frequency of the records in the month – together with mTotal, we calculate the average	None
monthPoint	Object	XYChart.Data	Subroutine	Holds the data point to be plotted on the monthly graph	None
yearlyRs	Object	ResultSet	Subroutine	Holds the results of the yearlyStmt	Not empty
lastYear	Variable	int	Subroutine	Stores the value of the current year in-hand (last accessed)	None
yTotal	Variable	double	Subroutine	Stores the total value of all the scores in a particular year	None
yFreq	Variable	double	Subroutine	Stores the number of records for that particular year – used in combination with yTotal to get the average for that year	None
yearPoint	Object	XYChart.Data	Subroutine	Data point to be plotted to yearlyGraph	None

Algorithms

After decomposing the problem, since my system will be object oriented, I can plan the object classes that link together and those that I require in my system and the contained algorithms that will help it function.

Main.java

A main class is required to launch any system in java.

```
class Main

    public static mainStage //stores the window object

    procedure main(args) //called at the start

        launch(args) //calls the start procedure and initializes all UI (fxml) data

    endprocedure

    procedure start(stage) //a stage is provided when executed by Java Virtual Machine

        mainStage = stage //lets become easily accessed as a static variable

        root = load("grade_select.fxml") //load the fxml data from grade_select.fxml
        file

        gradeScene = new Scene(root) //a new scene object containing the next UI

        stage.setScene(gradeScene) //set the stage (window) to display the grade
        select screen

        stage.show() //show the UI

    endprocedure

    public static function getStage() // (encapsulation) use getter method to access
    the stage (window)

        return mainStage

    endfunction

endclass
```

DAO.java

This data access object class will make it much more easier and efficient to connect to the database.

```
class DAO

    constant dbDir = "C://iMProve" //directory path for the database

    constant dbName = "improveDB.accdb" //name for the database

    constant dbUrl = "jdbc:ucanaccess://" + dbDir + "//" + dbName //url used to get
    connections to the database

    public procedure new() //constructor to be called when new dao object is created
```

```

try
    loadDriver("net.ucanaccess.jdbc.UcanaccessDriver") //load the
ucanaccess driver
    catch(exception)
        exception.print() //print error if cannot be loaded
    endtry
    //now validate whether the file paths exist
    directory = new File(dbDir) //directory file
    if NOT directory.exists() then //if file doesn't exist
        directory.mkdir() //make the directory
    endif

    database = new File(dbDir + "/" + dbName) //path to the database file
    if NOT database.exists() then //if the database doesn't exist inside
        try
            Files.copy(getResource(dbName), database) //try to copy the
database from the executable into the hard drive to 'database'
        catch(exception)
            exception.print() //print error if unsuccessful
        end try
    endif
endprocedure

public function getConnection() //allow other classes to get a connection to the
database
    conn = null //declare first to allow higher scope
    try
        conn = Driver.getConnection(dbUrl) //get a connection using the
ucanaccess driver
    catch(exception)
        exception.print() //print error if unsuccessful connection
    endtry
    return conn //return the new connection
endfunction
endclass

```

GradeController.java

The first UI that is loaded is for the grade selection screen, objects of this class will control it.

```
class GradeController

    fldName = FXML("fldName") //refers to the fxml object of fldName

    procedure gradeSelect(event) //called when user selects a grade button
        if fldName.getText() ISEMPTY then //no name entered
            noNameAlert = new Alert("You forgot to type your name in!") //dialog
            box informing the user they forgot their name
            noNameAlert.show() //show alert
        return //abruptly skip the rest of the code of this method if not
        entered name
    endif

    Button selectedButton = event.getSource() //get the source of the event to
    find what button was clicked

    grade = int(selectedButton.getText()) //get the grade from the button text
    name = fldName.getText() //retrieve their name
    //show GUI for the main program

    loader = new FXMLLoader("root.fxml") //loader connected to the root.fxml
    file

    root = loader.load() //load root.fxml data into root object
    rootController = loader.getController() //get the controller for the main
    program to give it data
    rootController.initData(name, grade) //initialize starting data
    Scene rootScene = new Scene(root)
    Main.getStage().setScene(rootScene) //use main class to set the scene to the
    main program, rootScene
```

RootController.java

This class shall control the events and the activity of the main system container.

```
class RootController

    //static variables since the grade and name of the user is the same for each object
    private static grade //stores the grade of the user
    private static name //stores the user of the user
    public procedure initData(name, grade) //initialise the data
        this.grade = grade //the objects grade value (this) will be set to the grade
        value in the scope of the procedure, i.e the parameter
        this.name = name
    endprocedure
```

```

//getter methods for each static variable (encapsulation and information hiding)

public static function getGrade()
    return grade
endfunction

public static function getName()
    return name
endfunction

procedure setActivity(root) //set what is shown in the system, i.e. the activity
    apnActivity.clear() //removes all nodes contained within apnActivity
    apnActivity.add(root) //lets apnActivity contain root
endprocedure

procedure goLearn(event) //called by a user interaction which fires an actionevent
    loader = new FXMLLoader("learn.fxml") //loader connected to learn.fxml
    root = loader.load() //loads the contents of learn.fxml into root
    controller = loader.getController()
    controller.initData(this) //give the controller the current RootController
instance
    setActivity(root)
endprocedure

procedure goTest(event) //same logic as goLearn but for the test feature
    loader = new FXMLLoader("test.fxml")
    root = loader.load()
    controller = loader.getController()
    controller.initData(this)
    setActivity(root)
endprocedure

procedure goListen(event) //same logic as above
    loader = new FXMLLoader("listen.fxml")
    root = loader.load()
    controller = loader.getController()
    controller.initData(this)

```

```

    setActivity(root)
endprocedure

procedure goReview(event) //same logic as above
    loader = new FXMLLoader("review.fxml")
    root = loader.load()
    controller = loader.getController()
    controller.initData(this)
    setActivity(root)
endprocedure

endclass

```

LearnController.java

This is the object class that will control the welcome screen of the learn section

```

class LearnController
{
    private rootController //reference object to RootController used to change activity
    private titleSet = new Set() //collection storing the titles for each term (set is
    used to prevent duplicate titles)

    private dao = new DAO() //used to connect to database
    private vbxSets = FXML("vbxSets") //refers to fxml object connected to learn.fxml

    public procedure new() //constructor called when object of this class is created
        initSets() //pack the VBox with the existing set rows
    endprocedure

    procedure initData(rootController)
        this.rootController = rootController //initialise the rootController object
    endprocedure

    procedure initSets()
        try //try will be used to allow for i/o and SQL operations
            conn = dao.getConnection() //get a connection from DAO object
            stmt = new Statement("SELECT Title FROM Theory") //query to gather
            all the titles from the database
            rs = stmt.execute() //the result set returned from the execute()
            method is stored here
        while rs.nextRecord() //iterate through the resultset, each record
            title = rs.get("Title") //get the title
        endwhile
    endprocedure
}

```

```

        titleSet.add(title) //add the title to the collection

    endwhile

    conn.close() //close connection so no corrupts

catch(exception) //incase an error occurs

    exception.print()

endtry

foreach title as String in titleSet //iterate through the titleSet

    addRow(title) //add the row to the VBox using this title

next

endprocedure

procedure addRow(title)

    rowLoader = new FXMLLoader("learn_row.fxml") //a row used to display the
category

    row = rowLoader.load()

    rowController = rowLoader.getController() //get the controller for row

    rowController initData(title, rootController) //pass it the title to be used
to change the text and also the rootController to change the activity

    vbxSets.add(row) //display the row on the interface

endprocedure

procedure createSet(event) //show create set gui when respective UI control is
clicked

    title = input("Please provide a title for your new flashcard set") //need
title for new set

    //load gui for learn_update.fxml

    loader = FXMLLoader("learn_update.fxml")

    root = loader.load()

    controller = loader.getController()

    controller initData(title)

    rootController.setActivity(root)

endprocedure

endclass

```

LearnRowController.java

A controller class needs to be made for each row displayed on vbxSets.

```

class LearnRowController

    private dao = new DAO() //connects to database

    private txtTitle = FXML("txtTitle") //refers to the control node called txtTitle
    private apnSetRow = FXML("apnSetRow") //the root node of the learn_row.fxml ui

    public procedure initData(title)
        txtTitle.setText(title)
    endprocedure

    procedure playFlashcards(event) //used to go to the flashcards section
        loader = new FXMLLoader("learn_flashcards.fxml")
        root = loader.load()
        controller = loader.getController()
        controller.initData(txtTitle.getText()) //give the controller for the
        flashcards the title of the current set
        rootController.setActivity(root)
    endprocedure

    procedure editSet(event) //called show gui to edit the set
        loader = new FXMLLoader("learn_update.fxml")
        root = loader.load()
        controller = loader.getController()
        controller.initData(txtTitle.getText()) //pass the controller the title of
        the set
        rootController.setActivity(root)
    endprocedure

    procedure deleteSet(event) //called when user decides to delete the set
        //firstly confirm
        confirmAlert = new Alert("Are you sure you want to delete this set?")
        result = confirmAlert.show() //get the result returned from showing this
        alert
        if result equals OK then //if user agrees
            //permanently delete from database
            try
                conn = dao.getConnection()
                stmt = new Statement("DELETE FROM Theory WHERE Title = ?")
            //deletes all records with the same title as displayed on txtTitle

```

```

                stmt.setParameter(1, txtTitle.getText()) //setting the title
like this rather than string concatenation allows for a more secure SQL statement,
eliminating SQL injection threats

        stmt.execute()

        conn.commit() //permanently commit these changes to database

        conn.close() //close the connection

    catch(exception)

        exception.printStackTrace() //if unsuccessful then print why

    endtry

//delete from interface

vbxSets = apnSetRow.getParent() //get the parent container of the
root of this file - will be the VBox holding it

vbxSets.remove(apnSetRow) //remove this row from the UI

endif

endprocedure

endclass

```

LearnUpdateController.java

This is the controller which can be reused for editing and creating sets

```

class LearnUpdateController

    controllerMap = new HashMap() //a map to retrieve the controller for each row
representing a flashcard

    dao = new DAO();

    vbxCards = FXML("vbxCards")

    txtNewTitle = FXML("txtNewTitle")


procedure initData(title)

    txtNewTitle.setText(title)

//load all the existing flashcards

try

    conn = dao.getConnection()

    stmt = new Statement("SELECT Term, Definition FROM Theory WHERE Title
= ?") //for getting the term and definition to create a flashcard

    stmt.setParameter(1, title)

    rs = stmt.execute() //store the result set from the statement here

```

```

        while rs.nextRecord() //while there are more terms to be added
            term = rs.get("Term")
            definition = rs.get("Definition")
            addRow(term, definition) //add the row using the fetched term
and definition

        endwhile

        conn.close() //close connection

    catch(exception)
        exception.print() //if something doesn't work, print why
    endtry

endprocedure

procedure addRow(term, def)
    //load the row

    loader = new FXMLLoader("learn_update_row.fxml")
    row = loader.load()
    controller = loader.getController()
    controller initData(term, def) //the term and definition which should be
displayed

    controllerMap.add(row, controller) //add the row and the linking controller
to the hashmap

    //add to interface
    vbxCards.add(row)

endprocedure

procedure finish(event) //called when the user submits their changes to the
database
    //to change the terms either if we are creating a new set of editing an
existing, a general solution/algorithm would be to delete all the existing terms with that
title and then insert the new ones into the database.

    try
        conn = dao.getConnection()
        deleteStmt = new Statement("DELETE FROM Theory WHERE Title = ?")
        stmt = new Statement ("INSERT INTO Theory (Term, Definition, Title)
VALUES (?, ?, ?)")

        deleteStmt.setParameter(1, txtNewTitle.getText())
        deleteStmt.execute() //delete all existing terms

```

```

        foreach controller as LearnUpdateRowController in
controllerMap.values() //iterate through each controller in the map

            stmt.setParameter(1, c.getTerm()) //set the term
            stmt.setParameter(1, c.getDef()) //set the definition
            stmt.setParameter(3, txtNewTitle.getText()) //set the title
            stmt.execute() //insert new terms

        next

        conn.commit() //permanently commit these changes
        conn.close() //close the connection
    catch(exception)
        exception.print() //print what is wrong if error occurs
    endtry
endprocedure

public function getControllerMap() //used to retrieve the collection of controllers
for later changes

    return controllerMap
endfunction
endprocedure

```

LearnUpdateRowController.java

We need a controller class for the rows provided by learn_update_row.fxml

```

class LearnUpdateRowController

    //reference to the fxml input textfields
    private fldTerm = FXML("fldTerm")
    private fldDef = FXML("fldDef")

    private apnCardRow = FXML("apnCardRow") //used to later refer to the outer
    container for the rows

    public procedure new(term, definition)
        setTerm(term)
        setDef(definition)
    endprocedure

    procedure removeCard(event)
        vbxCards = apnCardRow.getParent() //outer container of the row is the VBox
        containing it, vbxCards

        vbxCards.remove(apnCardRow) //when user presses delete button, the row is
        removed from interface

```

```

endprocedure

//getter and setter methods for the instance variables

public procedure setTerm(term)
    fldTerm.setText(term)
endprocedure

public procedure setDef(def)
    fldDef.setText(def)
endprocedure

public function getTerm()
    return fldTerm.getText()
endfunction

public function getDef()
    return fldDef.getText()
endfunction

endclass

```

LearnFlashcardController.java

For the flashcards section of the system, we will use this controller for controlling the navigation of the flashcards.

```

class LearnFlashcardController

    private flashcardList = new List() //stores the list of flashcards to be displayed
    private dao = new DAO() //used to connect to database
    private flashcardIndex = 0 //reference which part of the list is being used
    private questionVisible = true //question or answer is on top of the flashcard
    private title //holds the title of the flashcard set
    private txtSetTitle = FXML("txtSetTitle") //displays the title
    private txtTerm = FXML("txtTerm") //displays the foreign term on the card
    private txtDef = FXML("txtDef") //displays the definition for the card

    public procedure initData(title)
        //display title
        this.title = title
        txtSetTitle.setText(title)

        //initialise the flashcards

```

```

try
    conn = dao.getConnection()

    stmt = new Statement("SELECT Term, Definition FROM Theory Where Title
= ?") //get all the data we need to make a flashcard

    stmt.setParameter(1, title) //use parameters to prevent sql injection

    rs = stmt.execute() //store the resultset here

    while rs.nextRecord() //while there are terms

        foreignTerm = rs.get("Term")

        definition = rs.get("Definition")

        flashcard = new Flashcard(foreignTerm, definition)

        flashcardList.add(flashcard) //add the flashcard to the list

    endwhile

    conn.close() //close connection

catch(exception)
    exception.print()

endtry

loadFlashcard() //now load our first flashcard
endprocedure

procedure loadFlashcard()
    currentCard = flashcardList.get(flashcardIndex) //first flashcard will be at
default value of flashcardIndex, 0

    //change the display text

    txtTerm.setText(currentCard.getTerm())
    txtDef.setText(currentCard.getDefinition())

endprocedure

procedure flip(event) //called when user clicks to flip the card
    txtTerm.setVisible(!questionVisible) //question will be the opposite
visibility as before

    txtDef.setVisible(questionVisible) //answer is opposite visibility to
question

    questionVisible = !questionVisible //now the visibility of the question is
changed

endprocedure

```

```

procedure moveCard(event) //to move the flashcard to next/previous one
    clickedButton = event.getSource() //determine if it was next or back
    if clickedButton equals btnLeft then //wants to go back
        //decrement flashcard index
        flashcardIndex--
        if flashcardIndex < 0 then //if below 0, move to max index, like
circular queue
            flashcardIndex = flashcardList.size() - 1
    endif
    elseif clickedButton equals btnRight then //next card
        //increment flashcardIndex using circular queue technique
        flashcardIndex = (flashcardIndex + 1) MOD flashcardList.size()
    endif
    loadFlashcard() //load the new flashcard
endprocedure
endclass

```

Flashcard.java

To make it easier to store the flashcards, we will have an object class to model the structure of a flashcard and the data/instructions one requires.

```

class Flashcard
    //term and definition (front and back) of a flashcard
    private foreignTerm
    private definition

    public procedure new(foreignTerm, definition) //constructor to set initial values
        this.foreignTerm = foreignTerm
        this.definition = definition
    endprocedure

    //getter setter methods for the variables
    public function getTerm()
        return foreignTerm
    endfunction

```

```

public function getDefinition()
    return definition
endfunction

public procedure setTerm(foreignTerm)
    this.foreignTerm = foreignTerm
endprocedure

public procedure setDefinition(definition)
    this.definition = definition
endprocedure

endclass

```

TestController.java

Moving onto the test section, we have a separate set of controllers to handle the events of what occurs in this activity.

```

class TestController
    private rootController
    public procedure initData(rootController)
        this.rootController = rootController
    endprocedure
    procedure playTest(event)
        loader = new FXMLLoader("test_quiz.fxml") //loader connected to
        test_quiz.fxml
        root = loader.load()
        controller = loader.getController()
        controller.initData(rootController) //allow the quiz feature to change
        activity by passing it a reference to rootController
        rootController.setActivity(root)
    endprocedure
endclass

```

TestQuizController.java

A controller is required for the activity of the quiz section

```
class TestQuizController
```

```

private rootController

private dao = new DAO()

private questionIndex = 0 //stores the question number

private theoryQuestionList = new LinkedList() //holds all the questions

private feedbackList = new LinkedList() //holds the individual feedback for user

private correctTerms = new LinkedList() //holds the correctly answered terms

private txtQuestion = FXML("txtQuestion") //displays the question

private btnOk = FXML("btnOk") //allows the user to move onto the next question

//the four multiple choice answer buttons

private btnOne = FXML("btnOne")

private btnTwo = FXML("btnTwo")

private btnThree = FXML("btnThree")

private btnFour = FXML("btnFour")

public procedure initData(rootController)

    this.rootController = rootController

    getTheoryQuestions() //get the theory questions from database

    loadQuestion() //load the first question

endprocedure

procedure getTheoryQuestions()

    try

        conn = dao.getConnection()

        priorityStmt("SELECT RANDOM Term, Definition WHERE Grade <= ? AND
Priority is High ORDER BY Priority) //randomly query the high priority terms which the user
is capable of answering(determined by grade)

        rndStmt("SELECT RANDOM Term, Definition WHERE Grade <= ?") //get the
rest of the questions randomly

        answersStmt("SELECT RANDOM Definition WHERE Grade <= ? AND Term !=
?") //get the fillers answers randomly

        //set grade parameters securely

        priorityStmt.setParameter(1, RootController.getGrade())

        rndStmt.setParameter(1, RootController.getGrade())

        answersStmt.setParameter(1, RootController.getGrade())

        //store the result sets

        priorityRs = priorityStmt.execute()

        rndRs = rndStmt.execute()

```

```

//add the theory questions

int priorityQuestions = 0 //number of priority questions

while priorityRs.nextRecord() AND questions < 10

    question = priorityRs.get("Term")

    array answers[4]

    answerIndex = randomInt(0, 4) //random integer between 0
inclusive and 4 exclusive

    answersStmt.setParameter(2, question)

    ansRs = answersStmt.execute()

    for i = 0 to 3 //get the four answers

        if i == answerIndex then //set correct answer

            answers[answerIndex] =

priorityRs.get("Definition")

        else

            //get and set random answers

            ansRs.nextRecord()

            answers[i] = ansRs.get("Definition")

        endif

    next i

    //add the theory question to collection

    theoryQuestion = new TheoryQuestion(question, answers,
answerIndex)

    theoryQuestionList.add(theoryQuestion)

    priorityQuestions++ //increment number of priority questions
by 1

endwhile


for i = 1 to 10 - priorityQuestions //fill the rest of the questions
with random questions

    if rndRs.nextRecord() then //iterates to next record and
checks whether there is a record available

        //declare and initialise data for test

        question = rndRs.get("Term")

        array answers[4]

        answerIndex = randomInt(0, 4)

        answersStmt.setParameter(2, question)

```

```

ansRs = answersStmt.execute()

for j = 0 to 3 //get the four answers

    if j = answerIndex then

        answers[j] = rndRs.get("Defintion")

    else

        ansRs.nextRecord()

        answers[j] = ansRs.get("Definition")

    endif

next j

endif

//add the theory question to the collection

theoryQuestion = new TheoryQuestion(question, answers,
answerIndex)

theoryQuestionList.add(theoryQuestion)

next i

catch(exception) //if an error connecting to database/any other sql related
exception then print why it occurred

exception.print()

endtry

endprocedure

procedure loadQuestion()

theoryQuestion = theoryQuestionList.get(questionIndex) //get the question at
specified index

//set the ui to display the question and answers

txtQuestion.setText("What is the definition of" +
theoryQuestion.getQuestion())

btnOne.setText(theoryQuestion.getAnswer(1))

btnTwo.setText(theoryQuestion.getAnswer(2))

btnThree.setText(theoryQuestion.getAnswer(3))

btnFour.setText(theoryQuestion.getAnswer(4))

endprocedure

procedure answerQuestion(event) //called by clicking on an answer button

theoryQuestion = theoryQuestionList.get(questionIndex) //get current
question

correctAnswer = theoryQuestion.getAnswerIndex() //locates the answer index

```

```

clickedButton = event.getSource() //get the clicked button
//get the user inputted answer index
if clickedButton equals btnOne then
    inputAnswer = 1
elseif clickedButton equals btnTwo then
    inputAnswer = 2
elseif clickedButton equals btnThree then
    inputAnswer = 3
elseif clickedButton equals btnFour then
    inputAnswer = 4
endif

if inputAnswer == correctAnswer then //if correct choice
    //change colours for visual feedback
    clickedButton.setColour(GREEN)
    //store the correctly answered terms into correctTerms
    question = theoryQuestion.getQuestion()
    correctTerms.add(question)

else //if wrong choice
    clickedButton.setColour(RED)
    if correctAnswer == 1 then
        correctButton = btnOne
    elseif correctAnswer ==2 then
        correctButton = btnTwo
    elseif correctAnswer == 3 then
        correctButton = btnThree
    elseif correctAnswer == 4 then
        correctButton = btnFour
    endif
    correctButton.setColour(GREEN)

//store feedback into feedbackList
question = theoryQuestion.getQuestion()
wrongAnswer = theoryQuestion.getAnswer(inputAnswer)

```

```

        correctAnswer = theoryQuestion.getAnswer(correctAnswer)
        feedback = new Feedback(question, wrongAnswer, correctAnswer)
        feedbackList.add(feedback)

    endif

    btnOk.setVisible(true) //show the ok button to let the user move on
endprocedure

procedure nextQuestion(event) //called when user presses ok button
    questionIndex++ //increment question number
    if questionIndex < 10 then //if still lower than 10
        btnOk.setVisible(false)
        //reset colours
        btnOne.setColour(WHITE)
        btnTwo.setColour(WHITE)
        btnThree.setColour(WHITE)
        btnFour.setColour(WHITE)
        loadQuestion()
    else //if reached past last question
        loadResults() //load results screen
    endif
endprocedure

procedure loadResults() //load results screen
    loader = new FXMLLoader("test_results.fxml") //loader connected to
    test_results.fxml
    root = loader.load()
    controller = loader.getController()
    //give the controller of results screen some variables and also the result
    for the test which is equal to 10 - feedbackList.size() or 10 - number of wrong answers
    controller.initData(feedbackList, 10 - feedbackList.size(), correctTerms)
    rootController.setActivity(root)
endprocedure

endclass

```

TestResultsController.java

For the results section for the quiz, this class will help the processing of the test data and manage the data flow to and from the database.

```
class TestResultsController

    private feedbackList
    private correctTerms
    private result
    private dao = new DAO()
    private txtScore = FXML("txtScore") //displays the score
    private vbxFeedback = FXML("vbxFeedback") //holds the individual feedback rows

    public procedure initData(feedbackList, result, correctTerms)
        //initialise variables
        this.feedbackList = feedbackList
        this.result = result
        this.correctTerms = correctTerms
        //update interface
        txtScore.setText("You scored " + result + "/10 !") //display score
        foreach feedback as Feedback in feedbackList
            txtFeedback = new Text() //displays the feedback
            question = feedback.getQuestion()
            wrongAnswer = feedback.getWrong()
            correctAnswer = feedback.getCorrect()
            txtFeedback.setText("For " + question + ", it's not " + wrongAnswer +
            " but it is " + correctAnswer + ".") //set the feedback text
            vbxFeedbacks.add(txtFeedback) //display it onto UI
        next
        updateDB() //change data inside database
    endprocedure

    procedure updateDB()
        try
            conn = dao.getConnection()
            //change priority to adapt future tests
        
```

```

        higherPStmt = new Statement("UPDATE Theory SET Priority = Priority +
1 WHERE Term = ?") //increase priority

        lowerPStmt = new Statement("UPDATE Theory SET Priority = Priority - 1
WHERE Term = ?") //decrease priority

        //record the score for the test

        oldScoreStmt = new Statement("SELECT Highscore FROM Score WHERE
ScoreDay = ? And ScoreMonth = ? AND ScoreYear - ?") //get the old highscore to compare with

        insScoreStmt = new Statement("INSERT INTO Score (ScoreDay,
ScoreMonth, ScoreYear, Highscore) VALUES(?, ?, ?, ?)")//if no highscore found, new record

        updScoreStmt = new Statement("UPDATE Score SET Highscore = ? WHERE
ScoreDay = ? AND ScoreMonth = ? AND ScoreYear = ?") //update score if higher than highscore

        //update for correct terms

        foreach term as String in correctTerms

            lowerPStmt.setParameter(1, term) //correct terms are lower
priority for next test, focus on weaknesses

            lowerPStmt.execute()

        next

        //update for wrong terms

        foreach feedback as Feedback in feedbackList

            wrongTerm = feedback.getQuestion()

            higherPStmt.setParameter(1, wrongTerm)

            higherPStmt.execute()

        next

        //update scores

        //reusable variables for our queries

        currentDay = date(DAY_OF_MONTH)

        currentMonth = date(MONTH)

        currentYear = date(YEAR)

        //firstly get old highscore

        oldScoreStmt.setParameter(1, currentDay)

        oldScoreStmt.setParameter(2, currentMonth)

        oldScoreStmt.setParameter(3, currentYear)

        oldScoreRs = oldScoreStmt.execute() //hold the result

        oldScoreRs.nextRecord() //iterate to first record

        oldHighscore = oldScoreRs.get("Highscore") //get the highscore

        if NOT oldHighscore exists then //if test hasn't been played today

```

```

        //insert new record
        insScoreStmt.setParameter(1, currentDay)
        insScoreStmt.setParameter(2, currentMonth)
        insScoreStmt.setParameter(3, currentYear)
        insScoreStmt.setParameter(4, result) //contains the achieved
score
        insScoreStmt.execute()

elseif result > oldhighscore then //if exists and result is larger
than old highscore

        updScoreStmt.setParameter(1, result) //new highscore
        updScoreStmt.setParameter(2, currentDay)
        updScoreStmt.setParameter(3, currentMonth)
        updScoreStmt.setParameter(4, currentYear)

endif

conn.commit() //permanently commit these changes to database
catch(exception) //if error occurred in editing database
    exception.print() //print why
endtry
endprocedure
endclass

```

TheoryQuestion.java

To make the algorithms more clean, easier to maintain and efficient, we will use a class to model the structure of a theory question and the data one needs.

```

class TheoryQuestion

    private question //stores the question in a string variable
    private array answers[4] //answers
    private answerIndex //correct index of answers

    public procedure new(question, answers, answerIndex) //initialise variables with
constructor
        this.question = question
        this.answers = answers
        this.answerIndex = answerIndex
endprocedure

```

```

//getter/setter methods for the variables

public procedure setQuestion(question)
    this.question = question
endprocedure

public procedure setAnswers(a1, a2, a3, a4)
    answers[0] = a1
    answers[1] = a2
    answers[2] = a3
    answers[3] = a4
endprocedure

public procedure setIndex(answerIndex)
    answerIndex = answerIndex
endprocedure

public function getQuestion()
    return question
endfunction

public function getAnswers()
    return answers
endfunction

public function getAnswerIndex()
    return answerIndex
endfunction

endclass

```

Feedback.java

Also helping our functionality, we have a class to model the feedback used especially for the results page.

```

class Feedback

    //variables required for feedback

    private question
    private wrongAnswer
    private correctAnswer

    //constructor to set the feedback at first
    public Feedback(question, wrongAnswer, correctAnswer)
        this.question = question

```

```

        this.wrongAnswer = wrongAnswer
        this.correctAnswer
    endprocedure

    //getter methods to retrieve it, note that no setter methods are required since we
    are only setting the feedback instance variables once when initialized

    public function getQuestion()
        return question
    endfunction

    public function getWrong()
        return wrongAnswer
    endfunction

    public function getCorrect()
        return correctAnswer
    endfunction

endclass

```

ListenController.java

A new set of controller will be used for the listen section, firstly the listen welcome screen controller.

```

class ListenController
{
    private rootController //allows change in activity

    public procedure initData(rootController)
        this.rootController = rootController //sets root controller object
    endprocedure

    public procedure playQuiz(event) //called when play button is pressed
        loader = new FXMLLoader("listen_quiz.fxml")
        root = loader.load()
        controller = loader.getController()
        controller.initData(rootController) //pass the root controller object to
allow activity change
        rootController.setActivity(root) //set the activity to the quiz
    endprocedure

}endclass

```

ListenQuizController.java

Now for the main part of this section, the quiz.

```
class ListenQuizController

    private rootController

    private dao = new DAO()

    private currentTest //current test object of class Auraltest

    private icnPlayPause = FXML("icnPlayPause")

    //input choices

    private btnRelMaj = FXML("btnRelMaj")

    private btnRelMin = FXML("btnRelMin")

    private btnDom = FXML("btnDom")

    private btnSubDom = FXML("btnSub")



    public procedure initData(rootController)

        this.rootController = rootController


        getAuralTest() //get and load the aural test

    endprocedure


    procedure getAuralTest()

        try

            conn = dao.getConnection()

            rndStmt = new Statement("SELECT RANDOM File, Modulation FROM Aural")
//statement to randomly select the file and modulation for an aural test

            rndRs = rndStmt.execute() //store the query result


            //create the current auraltest

            rndRs.nextRecord()

            fileName = rndRs.get("File")

            modulation = rndRs.get("Modulation")

            currentTest = new AuralTest(fileName, modulation)

        catch(exception) //if any database issues

            exception.print()

        endtry

    endprocedure
```

```

public procedure playPause(event) //called when user plays or pauses the audio
    if icnPlayPause.getIcon() equals "PLAY" then
        currentTest.getMediaPlayer().play()
        icnPlayPause.setIcon("PAUSE") //change the icon
    else
        currentTest.getMediaPlayer().pause()
        icnPlayPause.setIcon("PLAY")
    endif
endprocedure

public procedure answerQuestion(event) //called when user clicks an answer
    inputButton = e.getSource()
    correctModulation = currentTest.getModulation() //get the correct modulation
    switch(correctModulation)
        case "Relative Major": correctButton = btnRelMaj
        case "Relative Minor": correctButton = btnRelMin
        case "Dominant": correctButton = btnDom
        case "Subdominant": correctButton = btnSub
    endswitch
    //set visual feedback
    inputButton.setColour(RED)
    correctButton.setColour(GREEN) //this will override the red if user answered
correctly
endprocedure
endclass

```

AuralTest.java

For the aural testing feature, we will use a separate object class to model the structure and data of an aural test.

```

class AuralTest
    private audioFile //string of file path
    private mediaPlayer //MediaPlayer object
    private modulation //string of the modulation

```

```

public procedure new(audioFile, modulation)
    this.audioFile = audioFile
    this.modulation = modulation

    //create aural/mediaplayer
    media = new Media(getResource(audioFile))
    mediaPlayer = new MediaPlayer(media)
endprocedure

//getter methods for the variables, they are only set once on creation so no need
for setter methods

public function getMediaPlayer()
    return mediaPlayer
endfunction

public function getModulation()
    return modulation
endfunction

endclass

```

ReviewController.java

For our review section, we have a class to control the data flow from database onto the interface represented by graphs.

```

class ReviewController
    private rootController

    private dailySeries = new Series() //stores the series of data for the daily graph
    private monthlySeries = new Series() //series of data for monthly graph
    private yearlySeries = new Series() //series of data for yearly graph
    private dao = new DAO()

    //ui elements
    private apnDaily = FXML("apnDaily") //anchorpane container holding daily graph
    private apnMonthly = FXML("apnMonthly") //container of monthly
    private apnYearly = FXML("apnYearly") //container of yearly
    private txtProgress = FXML("txtProgress") //displays the title and the name of the
    user making progress

    public procedure initData(rootController)
        this.rootController = rootController

```

```

//change UI

txtProgress.setText(RootController.getName() + "s All-Time Progress")

getScores() //get data from database

setDailyGraph() //set data and display the daily graph

setMonthlyGraph() //for monthly graph

setYearlyGraph() //for yearly graph

endprocedure

procedure getScores()
    try
        conn = dao.getConnection()

        dailyStmt = new Statement("SELECT Highscore, ScoreDay FROM Score
WHERE ScoreMonth = ?") //get the scores for the each day

        monthlyStmt = new Statement("SELECT Highscore, ScoreMonth FROM Score
WHERE ScoreYear = ? ORDER BY ScoreMonth") //get the scores from each month, grouped
together by the ordering of the month

        yearlyStmt = new Statement("SELECT Highscore, ScoreYear FROM Score
ORDER BY ScoreYear") //get the scores for each year

        //daily graph
        month = date(MONTH)

        dailyStmt.setParameter(1, month)

        dailyRs = dailyStmt.execute() //results for the daily graph

        while dailyRs.nextRecord() //while there is a record

            //add the data to the series of the daily graph
            day = dailyRs.get("ScoreDay")
            highscore = dailyRs.get("Highscore")
            dayPoint = new Data(day, highscore)
            dailySeries.add(dayPoint)

        endwhile

        //monthly graph
        year = date(YEAR)

        monthlyStmt.setParameter(1, year)

        monthlyRs = monthlyStmt.execute()

        lastMonth //stores the last month accessed

```

```

monthTotal = 0.0 //used to find average per month
monthFreq = 0.0 //used to find average per month
while monthlyRs.nextRecord()
    if monthlyRs.get("ScoreMonth") equals lastMonth then
        //add variables for the average score
        monthTotal += monthlyRs.get("Highscore")
        monthFreq ++
    else
        if NOT lastMonth equals "" then//if checked last month
            //add last months data point
            monthAverage = monthTotal/monthFreq
            monthPoint = new Data(lastMonth, monthAverage)
            monthlySeries.add(monthPoint)
        end if
        lastMonth = monthlyRs.get("ScoreMonth") //set new
month value
        monthTotal = monthlyRs.get("Highscore") //add the
highscore of the current record to the current total
        monthFreq = 1 //frequency of current row is recorded
    end if
endwhile

//yearly graph
yearlyRs = yearlyStmt.execute()
lastYear = -1 //a year cannot be negative so a good starting value
yearTotal = 0.0 //for average
yearFreq = 0.0 //for average
//same average calculation algorithm as above for monthly
while yearlyRs.nextRecord()
    if yearlyRs.get("ScoreYear") == lastYear then
        yearTotal += yearlyRs.get("Highscore")
        yearFreq ++
    else
        if lastYear != -1 then //if checked a legitimate month
            yearAverage = yearTotal/yearFreq
            yearPoint = new Data(lastYear, yearAverage)

```

```

                yearlySeries.add(yearPoint)
            endif
            lastYear = yearlyRs.get("ScoreYear")
            yTotal = yearlyRs.get("Highscore")
            yFreq = 1
        end if
    end while
    catch(exception) //any database issues
        exception.print()
    endtry
endprocedure

procedure setDailyGraph()
    //create axes
    xAxis = new Axis()
    yAxis = new Axis()
    xAxis.setLabel("Day of month")
    yAxis.setLabel("Highscore")

    //create graph (chart) with axes
    dailyChart = new LineChart(xAxis, yAxis)
    dailyChart.add(dailySeries) //add the data to the graph

    //add graph to UI
    apnDaily.add(dailyChart)
endprocedure

procedure setMonthlyGraph() //set the graph for monthly data, same as above
    //create axes
    xAxis = new Axis()
    yAxis = new Axis()
    xAxis.setLabel("Month in year")
    yAxis.setLabel("Average score")

    //create graph with axes
    monthlyChart = new LineChart(xAxis, yAxis)

```

```

monthlyChart.add(monthlySeries) //add series of data

//add to ui
apnMonthly.add(monthlyChart)

endprocedure

procedure setYearlyGraph() //for yearly data
    xAxis = new Axis()
    yAxis = new Axis()
    xAxis.setLabel("Year")
    yAxis.setLabel("Average score")
    yearlyChart = new LineChart(xAxis, yAxis)
    yearlyChart.add(yearlySeries)
    apnYearly.add(yearlyChart)

endprocedure

endclass

```

Collectively, with all the separate classes' algorithms, they form a complete solution to the problem tackling each sub-section of the decomposed problem preserving efficiency whilst providing functionality.

Test Data for Development

Here, test data to be used through iterations of development will be described and justified. Although not a full test plan, at this stage, identification and justification of test data can still be demonstrated to ensure functionality and robustness of the system.

Grade Selection

When the grade is selected, two inputs are required: their name, and their grade.

Variable	Value Inputted	Expected Outcome	Justification
name	"Jill Therkell"	Accepted	Assures that a standard name is successful
name	""	Error popup, cannot enter no name.	A person must have a name larger than zero characters
name	"nalnbzngw18329"	Will still be accepted	Some people may have different or unique names, as long as it is not zero length, it is accepted

grade	grade 1/2/3/4/5/6/7/8 clicked	Accepted and displays welcome screen and sets allocated grade	User needs to navigate to next section
-------	-------------------------------------	---	--

Welcome Screen (Root Screen)

After a selection of grade and name input, they will be redirected to the main page and the root screen which will contain all later activities.

Action	Expected Outcome	Justification
Learn/Test/Listen/Review Clicked	Loads the respective activity	The user needs to navigate to their wished activity effectively

Learn Activity

When the learn screen is clicked, the main page where we select flashcard sets to play/delete/edit or create a new set.

Let's start with creating a new set

Variable	Value Inputted	Expected Outcome	Justification
title	“example”	Loads creation screen	A standard input should be accepted
title	“”	Error pops up/ Cancels creation of new set	Should not have a flashcard set with a title without a name
title	“a”	Loads creation screen	Although low length, some titles can have one letter descriptions e.g. ‘A’ can have different keys/tones etc.
term	“maestoso”	Will be accepted and changes will be made to database when submitted	Standard term should work
term	“”	Will not cause change to the database	There is no definition for blank space
term	“s”	Accepted	1 length words can still exist e.g. a for at
definition	“majestic”	Accepted	Standard term should be accepted
definition	“h”	Accepted	One length definitions can exist
definition	“”	Not accepted – term will not be inserted into database	There is always an English equivalent for a foreign term (no matter how descriptive it may be)

Action	Expected Outcome	Justification
Finish button clicked	Changes are made to the database and new set is displayed on the main flashcard screen	After finishing, the user wants their changes to be made
Add new flashcard button clicked	Adds an extra row on the UI to be displayed	Each set has a different number of flashcards so needs to be easy to change
Delete card button clicked	Removes the row on the UI	If there are more rows displayed than required

Playing the flashcards

Action	Expected Outcome	Justification
Next button clicked	Displays the next flashcard	The user needs to be able to iterate through each flashcard to learn
Back button clicked	Displays the previous flashcard	The user needs to be able to iterate through each flashcard to learn
Flip card	Flips the flashcard (i.e. displays the answer if the question is displayed or vice versa)	To be able to make links between foreign terms and English translations

Editing Set

Variable	Value Inputted	Expected Outcome	Justification
term	“pizzicato”	Accepted	A standard input should be accepted
term	“”	Flashcard gets deleted	There should be no blank space terms
definition	“plucked”	Accepted	A standard input should be accepted
definition	“”	Flashcard gets deleted	Blank space is not a definition.

Action	Expected Outcome	Justification
Finish button clicked	Saves changes to database and future uses	To edit a flashcard set, the changes must be made permanently
Add new flashcard	New row added onto UI	Indefinite number of flashcards
Remove flashcard	New row added onto UI	Indefinite number of flashcards.

Delete Set

Action	Expected Outcome	Justification
Delete flashcard set	Popup a confirmation dialog “Are you sure you want to delete this set”	To make sure the user is confident on a choice which can not be recovered
Confirm/Yes	Permanently deletes the set	User is now sure on their choice, changes should now be made
Cancel/No	Cancels the delete and goes back to main learn screen	If user mis-clicks or changes their mind, then we can adapt our system to prepare for such.

Test Activity

If the test activity is clicked, we are redirected to the welcome screen to start out testing.

Welcome Screen

Action	Expected Outcome	Justification
Play button clicked	Loads the first question of the test	User wants to progress

Quiz

Action	Expected Outcome	Justification
First/Second/Third/Fourth button clicked	Answers the question and displays relevant feedback (green if correct, red if wrong and green on the correct answer)	The feedback will allow the user to learn and improve.
OK button clicked	Moves onto next question/Shows result screen if last question	To complete the test

Results Screen

Action	Expected Outcome	Justification
Play again button clicked	Starts the test again from the start	To act on the mistakes, a replay of a new test should work

Listen Activity

With a click of the listen activity, we will be redirected to the aural testing feature.

Welcome Screen

Action	Expected Outcome	Justification
Play button clicked	Loads the first question of the test	Crucial for starting the test

Test

Action	Expected Outcome	Justification
Play button clicked	Plays the music	To start the aural testing, audio needs to be displayed to be analysed
Pause button clicked	Pauses the music	If the user needs to think about the question, the music should be paused
Slider dragged	Moves the position of the audio to indicated on slider	If user wants to skip/look back then this feature will allow that
First/Second/Third/Fourth button clicked	Answers the question respectively to input and displays feedback (coloured as in the test feature)	To learn from the section, a decision must be made to learn from.
Play again button clicked	Starts a new aural test.	To add more variety in the tests, and so better learning, a play again feature will allow easier access to do so.

Review Activity

If the user wants to review their progress from tests, clicking on review will display multiple graphs

Variable	Value Inputted	Expected Outcome	Justification
Score (Record)	15/6/2018 Highscore 8	Plotted on monthly graph	The graphs must be accurately displaying the correct data
Score (Record)	13/6/2018 Highscore 7	Average plotted on monthly graph	The monthly graph must be able to calculate and display averages
Score (Record)	27/10/2018 Highscore 9	Plotted on monthly graph and daily graph	Accurately display data on graph on different months
Score (Record)	25/10/2018 Highscore 7	Average for the month and new point of daily	Averages for all month
Score (Record)	24/10/2018 Highscore 4	Average for more than 2 points on monthly and daily graph plot	Averages for the month accurately calculated and displayed
Score (Record)	12/5/2018 Highscore 6	Plotted on monthly graph and line graph joining the plots	To display the chart and accurately display the progress to be interpreted by the user easily
Score (Record)	11/10/2018 Highscore 8 AND 10/10/2018 Highscore 7	Standard plots to be added to the daily and monthly graph, also adds complexity to the averages on the monthly and yearly graphs	Accurately calculate average values for yearly and monthly graph

Score (Record)	9/10/2017	Displayed on yearly graph but NOT on the monthly or daily graph although same month (not same year)	To ensure the yearly graph correctly displays multiple year data points and algorithms correctly validates the date.
Score (Record)	-1/0/-1203 Highscore 11	Not accepted into database	Invalid day/month/year and highscore values – would never occur since the system front-end automatically calculates these values and would never return impossible values.

Action	Expected Outcome	Justification
Daily clicked	Daily graph is displayed	Allows navigation to daily graph
Monthly clicked	Monthly graph is displayed	Allows navigation to see progress over months of the year
Yearly clicked	Yearly graph displayed	Navigates to see the progress over years.

Test Data for Post-Development

Here, data used in later stages of our project (the evaluation stage) will be identified, justified and linked to the correct success criteria being tested. Each of these data will test for either function, usability, validation or robustness, or a combination of them.

Test 1

Test Data	Success Criteria	Aspect	Justification
Grade Eight clicked with "", no name entered	(1) Grade Selection/Separation (21) Robust and Error-Free	Validation and Robustness	Ensures the user cannot progress the application with an invalid name, affecting later stages of the program
Grade Eight clicked with "Jill Thirkell" entered as name	(1) Grade Selection/Separation (22) Easy to use	Function and Usability	This is valid data which will partly ensure the functionality of the tests. Also, will demonstrate if the grade selection is easy to perform.
Examine the Welcome Screen	(20) Motivating (22) Easy to use	Usability	If the user easily understands what

			they can do now, then the system will be easy to use. Also, the welcome screen should partly motivate the user.
Select the Learn activity	(2) Navigation Side-Bar (13) Flashcard Set List	Function and Usability	The navigation bar will allow for easy navigation to and from activities. Also, a list of flashcard sets will be displayed when clicked.
Play a flashcard set e.g. Tempo	(17) Flashcard Term (18) Flashcard Definition	Function	Playing the flashcards, this test will ensure that the correct flashcards are correctly displayed.
Navigate through the flashcards using the given next, previous buttons	(17) Flashcard Term (18) Flashcard Definition	Function	Ensures that the user can navigate throughout the set.
Go back and click on create set	(14) Creating New Sets	Function	By redirecting the user to a creation of new sets screen, it partly tests for the success criteria identified
Enter “” for the title	(14) Creating New Sets	Validation	A title of blank space should not be allowed
Enter “Period” for the title	(14) Creating New Set	Validation	A standard title will be validated and should be accepted
Enter “” and “Definition”	(14) Creating New Sets	Validation	The blank space should not be entered as a foreign term and so this invalid data for which validation will check for that
Enter “Foreign Term” and “”	(14) Creating New Sets	Validation	Blank space should not be entered neither for the definition, so this invalid data will ensure validation.

Enter “Baroque” and “17 th Century”	(14) Creating New Sets	Validation and Function	Valid input, this tests that a standard input should work.
Click add new flashcard and enter “Romantic” and “19 th Century”	(14) Creating New Sets	Validation and Function	Valid input, also tests for the add new flashcard is fully functional
Click finish	(14) Creating New Sets (11) Storing data	Function	Finish will create multiple rows in the database representing each flashcard in the set
Play the new set	(14) Creating New Sets (11) Storing data	Function	If the set has been correctly made, then playing it will work and every term will be included. This tests for function.
Go back and delete the created set	(15) Deleting Sets	Function	This should delete the set and ensures it does not appear again in the interface
Edit set “Dynamics”	(16) Editing Sets (22) Easy to use	Function and Usability	A screen similar to creation of sets will be displayed, it should be easy to use and navigate with.
Delete all rows except “fortepiano” and “sforzando”	(16) Editing Sets	Function	This tests for changing the size of the set and accurately stores which terms (rows) have been kept or lost.
Edit the definition of “sforzando” from “accented” to “suddenly loud”	(16) Editing Sets	Function	Tests whether changes to existing rows are recorded
Edit the definition of “fortepiano” to blank space “”	(16) Editing Sets (21) Robust and Error-Free	Validation and Robustness	With this, the test will make sure that invalid data cannot be entered via editing sets
Add a new row “diminuendo”, “gradually quieter”	(16) Editing Sets	Function	Tests whether increasing the size will be recorded

Click finish	(16) Editing Sets (11) Storing the data	Function	Will allow the changes to be made for later tests.
Click on the Test activity and start the quiz	(6) Question Display (7) Multiple Choice Answers	Usability	Ensures that the quiz is correctly structured and displayed.
Answer a question	(9) Question Feedback (22) Easy to use	Function and Usability	The feedback should be accurately handled and displayed, this also demonstrates if it is easy to interpret for the user.
Click OK and complete the rest of the quiz	(8) Progress in Quiz	Usability	The quiz will display a bar determining how far the user is into it.
Examine results screen	(10) Results Feedback (11) Storing the Data	Usability and Function	A summary of all the feedback will be listed here, easier to improve on. Also, ensures that the correct data is being displayed (and stored).
Click on Review activity	(11) Storing the Data (12) Graphical Progress Display (20) Motivating	Usability and Function	If data was correctly stored, there should display a graph with a plot indicating the score of the previously taken test. This plot should appear on all graphs. A test such as this is justified as it allows the user to reflect on their progress and motivate the user
Click on Listen activity and play the music	(3) Audio playback (4) Play/Pause Button (6) Question Display (7) Multiple Choice Answers	Function and Usability	The play button should be fully functional and should be easy to use. Also, this test ensures that the quiz is displayed correctly.
Answer the question	(9) Question Feedback (22) Easy to use	Function and Usability	The correct answer will be displayed and compared to the

			inputted answer, this should be easy to interpret by the user.
Drag the slider to a previous point and play	(5) Seek Slider (4) Play/Pause Button (22) Easy to use	Function and Usability	The slider should be an easy way to change the position of the audio, and the test will demonstrate that.
Pause the music before it finishes and play it again	(4) Play/Pause Button	Function	Ensures that the pause feature works and that the play button will start again at where previous left off.
Press Exit	(19) Close the Application	Function and Robustness	Confirms if the application can be safely exited without any resource leaks or database corruptions.

Test 2

To more rigorously test the system, an extra test will ensure a stable, robust final product

Test Data	Success Criteria	Aspect	Justification
Grade Eight clicked with “Jill” as name	(21) Robust and Error-Free (1) Grade Separation/Selection	Function and Robustness	Should be accepted although a different name, it is still valid data.
Click on Learn and examine the list of flashcard sets	(13) Flashcard Set List (21) Robust and Error-Free	Function and Robustness	Starting a new application, the custom set with title “Period” should be still deleted
Play flashcard set of “Dynamics” and examine terms	(16) Editing Set (21) Robust and Error-Free (11) Storing the data (17) Flashcard Term (18) Flashcard Definition	Function and Robustness	This test will check whether the previous test’s changes have still been persistently stored. So, there should only be 2 rows, “sforzando” with “suddenly loud”, “fortepiano” should be deleted because of blank space, and “diminuendo” with “gradually quieter”.
Complete a test	(6) Question Display (7) Multiple Choice Answers (8) Progress in Quiz (9) Question Feedback	Function and Usability	Taking the test again will ensure it is motivating and easy to use, also that the functions of each part of the test is functional. However, the importance is after the test and

	(10) Results Feedback (11) Storing the data (20) Motivating (22) Easy to use		determining whether the results is correctly stored.
Click on Review and examine the graphs	(11) Storing the data (12) Graphical Progress Display (21) Robust and Error-Free	Function and Robustness	Even with a new start of application, the data points should not be wiped out and so in the daily graph, there should be a line joining the two data points from the two tests. On the monthly and yearly graph, it should display the average of the two high-score values.
Close the Application	(19) Close the Application (21) Robust and Error-Free	Function and Robustness	Ensures that the system exists without any error messages.

Test 3

Although we have rigorously tested the persistency of our system, we are ignoring a very crucial part of the program which affects the rest of our system: the grade separation.

First of all, we will change the system clock of the computer to a different year and different date (e.g. February 15th 2032) to fully exploit the review feature including its yearly graph.

Test Data	Success Criteria	Aspect	Justification
Grade One clicked with “” as name	(21) Robust and Error-Free (1) Grade Selection/Separation	Function and Robustness	Should also not be accepted even for a different grade, invalid data and so will not progress through the system.
Grade One clicked with “jill” as name	(21) Robust and Error-Free (1) Grade Selection/Separation	Function and Robustness	Standard input although different name, should still be accepted and directed into the welcome screen.
Click on Learn Activity and examine the list of flashcard sets	(21) Robust and Error-Free (2) Navigation Side-Bar	Function and Robustness	A diminished list of sets will appear compared to previous test because of the significantly lower grade. Also, the set previously created, “Period”, set should not appear
Observe the terms in a set, e.g. Articulation	(1) Grade Selection/Separation	Function and Robustness	Within a set, there will be less terms compared to when the user is grade 8. This test will check this.
Play a test	(1) Grade Selection/Separation (20) Motivating	Function and Usability	The test should be easier for a less skilled user, also the approachable questions will make the test more motivating.

Click on the Review activity	(21) Robust and Error Free (12) Graphical Progress Display	Function and Robustness	Will test for, although a different grade, that all the data points should still be display (on the yearly graph) as users can progress up (or down) in terms of grades.
Click on Exit	(19) Close the Application (21) Robust and Error Free (22) Easy to use	Function, Usability and Robustness	By now, it should be quite easy and memorable on how to exit the program so this test will test for the usability. Also, there should be no errors or misbehavior when clicking exit.

With all these tests, combined, they will ensure that our system has met its success criteria and that our system satisfies its function, validation, usability and robustness.

Development

My system will be following the prototype model and so include multiple iterations. Also, my system will follow the model, view, controller architecture which enhances the clarity and modularity of the code.

I used several technologies on the development of my program:

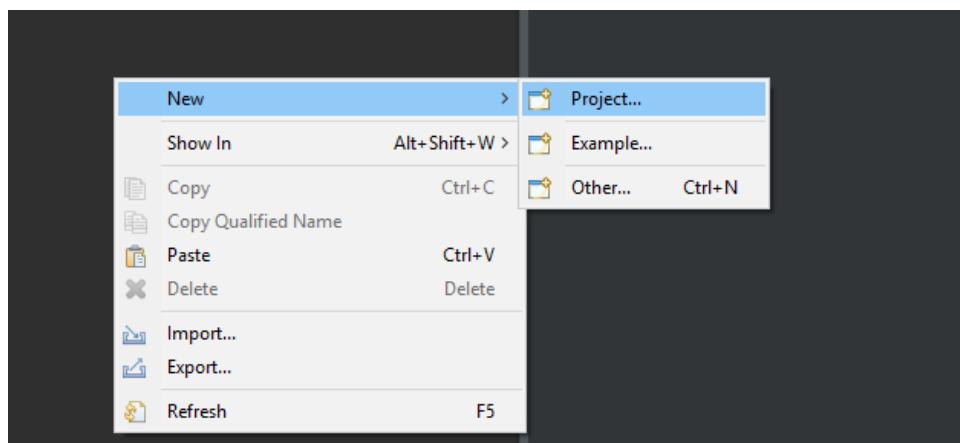
- Eclipse IDE SimRel 2018–2019
- JavaFX SceneBuilder 8 by Gluon
- Microsoft Access
- JDK 1.8.0_181 (the recent Java 8 library containing the whole Java API)
- External Libraries:
 - JFoenix (providing more options for visual controls)
 - FontawesomeFX (providing more ui options – mostly icons)
 - Ucanaccess (driver linking access and java)

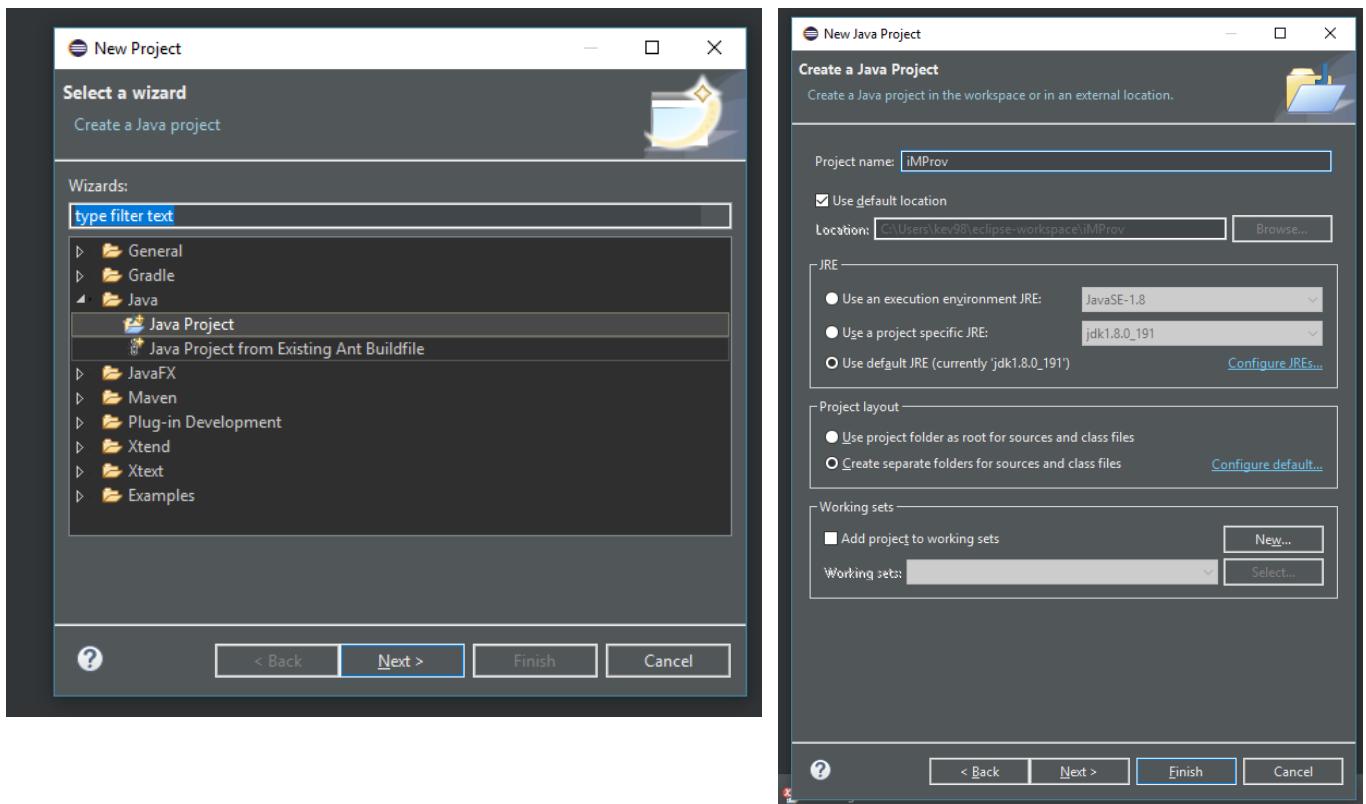
These technologies are crucial on the development of my Java system.

Note: For any view controls that are not described in this following section, please refer to Usability Features (pg. 40).

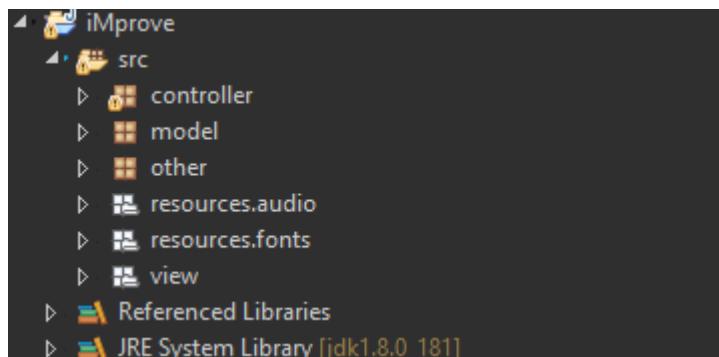
Creation of Project

We firstly need to create the whole project by using Eclipse IDE's features.





My system will use a somewhat unique directory structure where it follows the *MVC* architecture (model–view–controller):



Within the src, there will be a package containing all the models, another for the views, and another for the controllers. This allows our code classes, files and other resources to be modular, cleaner and more sustainable.

Also, the views inside the view directory has already been created, from our usability design so we can reuse them for our project.

Iteration 1

Main.java

In Java, the main class which includes the main method `public static void main(String[] args) {}` is the first thing that is invoked by the *Java Virtual Machine* (JVM) on startup.

My project is a JavaFX project so the main calls the start(Stage stage) method with
launch(args)

```
public static void main(String[] args) {  
    launch(args);  
}
```

The start() method looks like this:

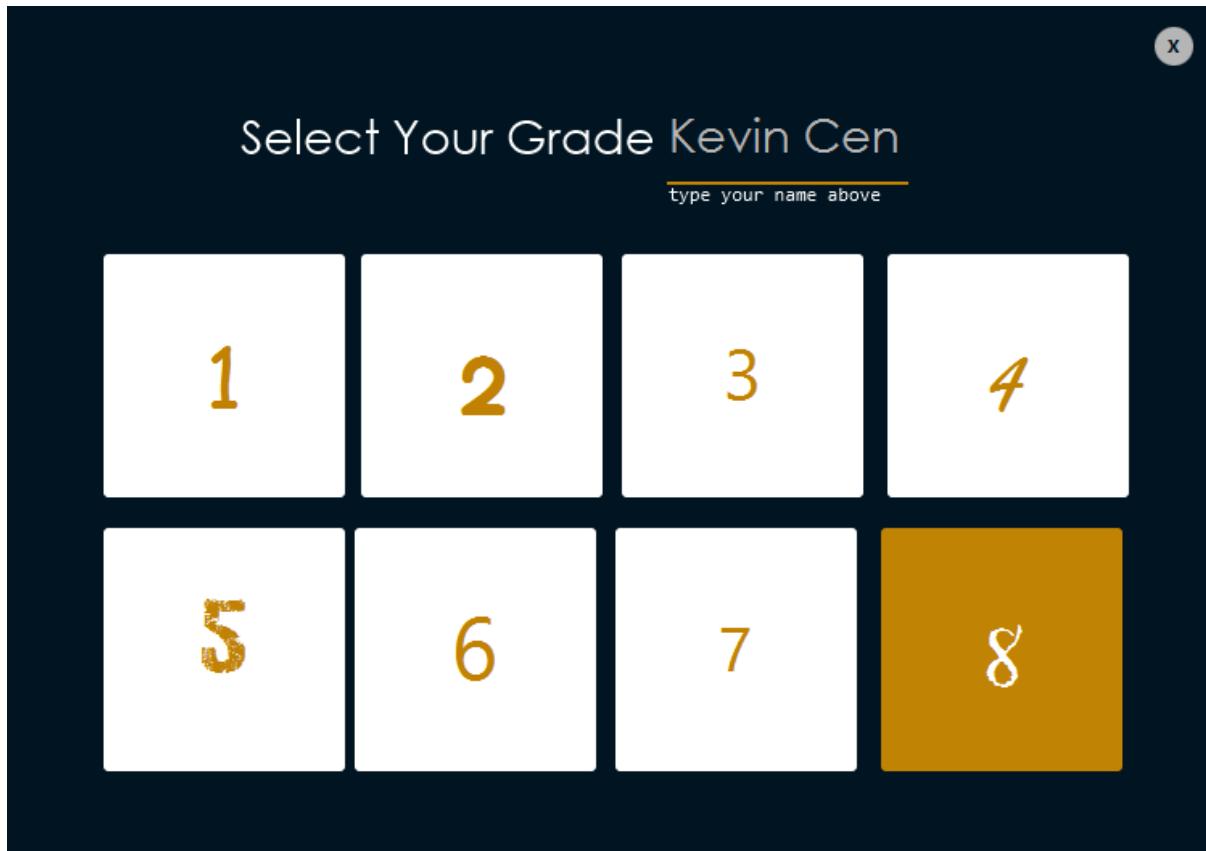
```
public void start(Stage stage) throws Exception {  
    loadFonts();  
    mainStage = stage;  
  
    Parent root = FXMLLoader.load(getClass().getResource("/view/grade_select.fxml"));  
    Scene gradeScene = new Scene(root);  
    stage.initStyle(StageStyle.TRANSPARENT);  
    stage.setScene(gradeScene);  
    stage.setTitle("iMProve");  
    stage.show();  
}
```

Within the start() method, loadFonts() is invoked which loads custom fonts and fonts that are not included by all versions of operating systems.

```
public void loadFonts() { //load fonts with random size, doesn't matter  
    Font.loadFont(getClass().getResourceAsStream("/resources/fonts/28DaysLater.ttf"), 14);  
    Font.loadFont(getClass().getResourceAsStream("/resources/fonts/ALBA_.ttf"), 14);  
    Font.loadFont(getClass().getResourceAsStream("/resources/fonts/ALBAM_.ttf"), 14);  
    Font.loadFont(getClass().getResourceAsStream("/resources/fonts/AKBAS_.ttf"), 14);  
    Font.loadFont(getClass().getResourceAsStream("/resources/fonts/BRADHITC.ttf"), 14);  
    Font.loadFont(getClass().getResourceAsStream("/resources/fonts/GOTHIC.ttf"), 14);  
    Font.loadFont(getClass().getResourceAsStream("/resources/fonts/GOTHICB.ttf"), 14);  
    Font.loadFont(getClass().getResourceAsStream("/resources/fonts/GOTHICBI.ttf"), 14);  
    Font.loadFont(getClass().getResourceAsStream("/resources/fonts/GOTHICI.ttf"), 14);  
    Font.loadFont(getClass().getResourceAsStream("/resources/fonts/SweetlyBroken.ttf"), 14);  
}
```

The fonts are loaded from a subdirectory under the src folder in src/resources/fonts/font.ttf and the fonts loaded are: '28 Days Later', 'Alba' (and other styles of the fonts), 'Bradley Hand ITC', 'Century Gothic' (with bold, italic and bold italic), 'Sweetly Broken'.

After setting the stage (the window of the application) to global, the visual user interface is loaded – namely grade_select.fxml. Each stage has multiple scenes, however iMProve will have only two, one for the initial startup (grade selection) and another for the activities. The stage will be StageStyle.TRANSPARENT meaning it will have a transparent background. Showing the stage, the grade_select.fxml view will be displayed:



Code for Main.java:

```
1 package other;
2
3 import javafx.application.Application;
4 import javafx.fxml.FXMLLoader;
5 import javafx.scene.Parent;
6 import javafx.scene.Scene;
7 import javafx.scene.layout.AnchorPane;
8 import javafx.scene.text.Font;
9 import javafx.stage.Stage;
10 import javafx.stage.StageStyle;
11
12 public class Main extends Application {
13     private static Stage mainStage;
14     @Override
15     public void start(Stage stage) throws Exception {
16         loadFonts();
17         mainStage = stage;
18
19         Parent root = FXMLLoader.load(getClass().getResource("/view/grade_select.fxml"));
20         Scene gradeScene = new Scene(root);
21         stage.initStyle(StageStyle.TRANSPARENT);
22         stage.setScene(gradeScene);
23         stage.setTitle("iMProve");
24         stage.show();
25     }
26     public static void main(String[] args) {
27         launch(args);
28     }
29     public void loadFonts() { //load fonts with random size, doesn't matter
30         Font.loadFont(getClass().getResourceAsStream("/resources/fonts/28DaysLater.ttf"), 14);
```

```

31     Font.loadFont(getClass().getResourceAsStream("/resources/fonts/ALBA__.ttf"), 14);
32     Font.loadFont(getClass().getResourceAsStream("/resources/fonts/ALBAM_.ttf"), 14);
33     Font.loadFont(getClass().getResourceAsStream("/resources/fonts/AKBAS_.ttf"), 14);
34     Font.loadFont(getClass().getResourceAsStream("/resources/fonts/BRADHITC.ttf"), 14);
35     Font.loadFont(getClass().getResourceAsStream("/resources/fonts/GOTHIC.ttf"), 14);
36     Font.loadFont(getClass().getResourceAsStream("/resources/fonts/GOTHICB.ttf"), 14);
37     Font.loadFont(getClass().getResourceAsStream("/resources/fonts/GOTHICBI.ttf"), 14);
38     Font.loadFont(getClass().getResourceAsStream("/resources/fonts/GOTHICI.ttf"), 14);
39     Font.loadFont(getClass().getResourceAsStream("/resources/fonts/SweetlyBroken.ttf"), 14);
40 }
41
42     public static Stage getStage() {
43         return mainStage;
44     }
45 }
46

```

DAO.java

This stands for *Data Access Object*, meaning it will help us link the program with the database and does the initial checks of the database, loads the drivers, and provides the code to connect to the database.

In the constructor, the block of code called when an object is initialized, we load the Unaccess driver which will connect us to the Microsoft Access database `improveDB.accdb`.

```

public DAO() { //constructor - called when object is made
    try {
        Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
    } catch(ClassNotFoundException e) {
        System.out.println("Cannot load ucanaccess driver.");
        e.printStackTrace();
    }
}

```

However, a common situation with database connectivity and deployment is that the database will not be able to be accessed within the deployed executable (.jar or .exe) even if it is stored within it. This means that we should create a file directory for the database to be stored in the remote user's desktop.

```

File directory = new File(dbDir);
if(!directory.exists()) //create directory if not already
    directory.mkdir();
File database = new File(dbDir + "/" + dbName);
if(!database.exists()) { //copy the database file into user's file system - if not already
    try {
        Files.copy(DAO.class.getResourceAsStream(dbName), database.toPath(), StandardCopyOption.REPLACE_EXISTING);
    } catch(IOException ex) {ex.printStackTrace();}
}
}

```

The above code will check whether the directory exists, if not, make directory, and checks whether the database in the directory already exists, if not, copy the database stored inside the executable to the file directory – this is wrapped in a try block since the file writing could throw input/output exceptions. Since we will only access the database inside the file system, the database inside the executable will not be updated.

Other objects and classes can access the database using an instance of DAO as an intermediate link between the database and system. The method that will provide connectivity is a function named `getConnection()`.

```

public Connection getConnection() { //create a connection to the database
    Connection conn = null;
    try {
        conn = DriverManager.getConnection(dbUrl);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return conn;
}

```

Code for DAO.java:

```

1 package other;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.nio.file.Files;
6 import java.nio.file.StandardCopyOption;
7 import java.sql.Connection;
8 import java.sql.DriverManager;
9 import java.sql.SQLException;
10
11 public class DAO {
12     private static final String dbDir = "C://iMProve";
13     private static final String dbName = "improveDB.accdb";
14     private static final String dbUrl = "jdbc:ucanaccess://" + dbDir + "/" + dbName;
15
16     public DAO() { //constructor - called when object is made
17         try {
18             Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
19         } catch(ClassNotFoundException e) {
20             System.out.println("Cannot load ucanaccess driver");
21             e.printStackTrace();
22         }
23         File directory = new File(dbDir);
24         if(!directory.exists()) //create directory if not already
25             directory.mkdir();
26         File database = new File(dbDir + "/" + dbName);
27         if(!database.exists()) { //copy the database file into user's file system - if not already
28             try {
29                 Files.copy(DAO.class.getResourceAsStream(dbName), database.toPath(), StandardCopyOption.REPLACE_EXISTING);
30             } catch(IOException ex) {ex.printStackTrace();}
31         }
32     }
33
34     public Connection getConnection() { //create a connection to the database
35         Connection conn = null;
36         try {
37             conn = DriverManager.getConnection(dbUrl);
38         } catch (SQLException e) {
39             e.printStackTrace();
40         }
41         return conn;
42     }
43 }

```

GradeController.java

Although main.java loads up grade_select.fxml, the view is not controlled by the main.java, but by a controller called GradeController.java identified in the JavaFX Scene Builder program.



This class provides the methods and interactions used by the grade_select.fxml view and controls the data flow between user and system.

Our JFXTextField which determines the user's name:

```
@FXML private JFXTextField fldName;
```



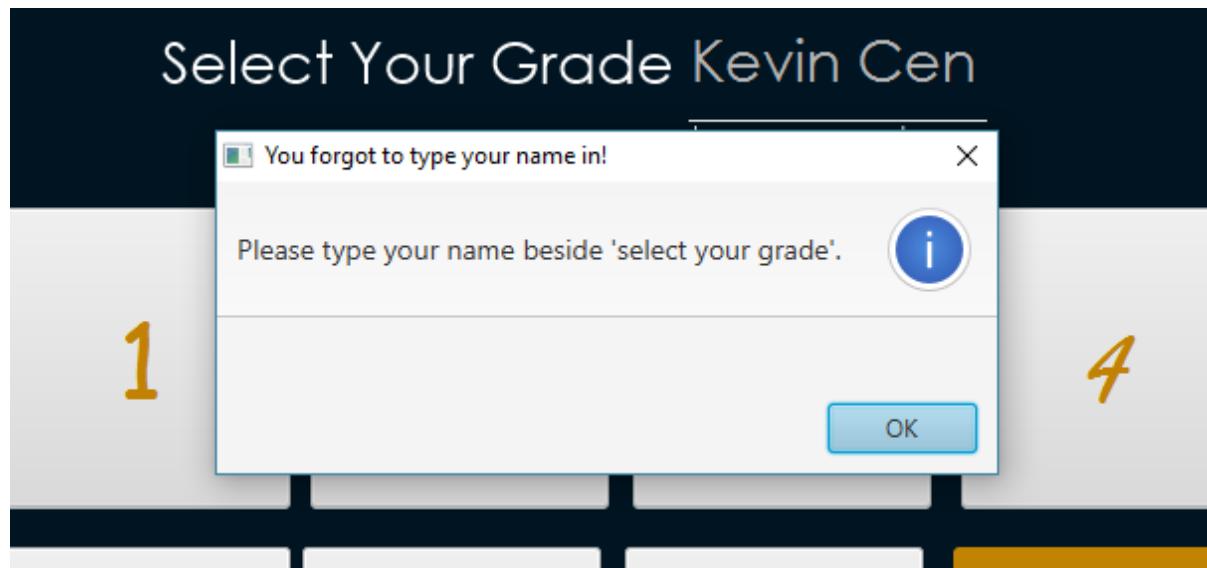
Is accessed by our controller by doing a FXML injection, where the @FXML annotation allows the controller linked to the view to access that specific user control with the same variable name; in this case, fldName.

The buttons to select each grade fires an event (an ActionEvent) to the controller to handle it with the gradeSelect() method. This method is called by the FX Application Thread which accesses the UI, and the public identifier with the @FXML annotation allows this thread to call gradeSelect().

```
@FXML  
public void gradeSelect(ActionEvent e) {  
    |  
}
```

The first if-selection statement creates a dialog (alert) to remind them to put a name into fldName if was previously left blank; leaving the method early so it doesn't carry out the code after and allowing the user to start again – achieved by return.

```
if(fldName.getText().equals("")) {  
    Alert noNameAlert = new Alert(AlertType.INFORMATION);  
    noNameAlert.setHeaderText("Please type your name beside 'select your grade'.");  
    noNameAlert.setTitle("You forgot to type your name in!");  
    noNameAlert.showAndWait();  
    return;  
}
```



The initial values of the user's grade and name is retrieved via accessing the respective controls.

Fetching the grade, we refer back to the Button that was clicked by finding the source of the event.

```
//get grade
Button selectedButton = (Button) e.getSource();
int grade = Integer.parseInt(selectedButton.getText());
```

Since the button's text is the grade number anyways, we can use that text and parse it into an integer to get the user's choice of grade.

The name is retrieved by getting the fldName's (JFXTextField) text where the user entered their name in.

Next, the root.fxml is loaded up similar to how the grade_select.fxml was loaded up however accessing the RootController and initialising the variables: name and grade.

```
RootController rootController = loader.getController();
rootController.initData(name, grade);
```

The code for this method looks like this so far:

```
@FXML
public void gradeSelect(ActionEvent e) {
    //get grade
    Button selectedButton = (Button) e.getSource();
    int grade = Integer.parseInt(selectedButton.getText());
    //get name
    String name = fldName.getText();
    //load root gui and set some initial values
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/root.fxml"));
    Parent root = null;
    try {
        root = loader.load();
    } catch(IOException ioe) {ioe.printStackTrace();}
    RootController rootController = loader.getController();
    rootController.initData(name, grade);
    Scene rootScene = new Scene(root);
    Main.getStage().setScene(rootScene);
}
```

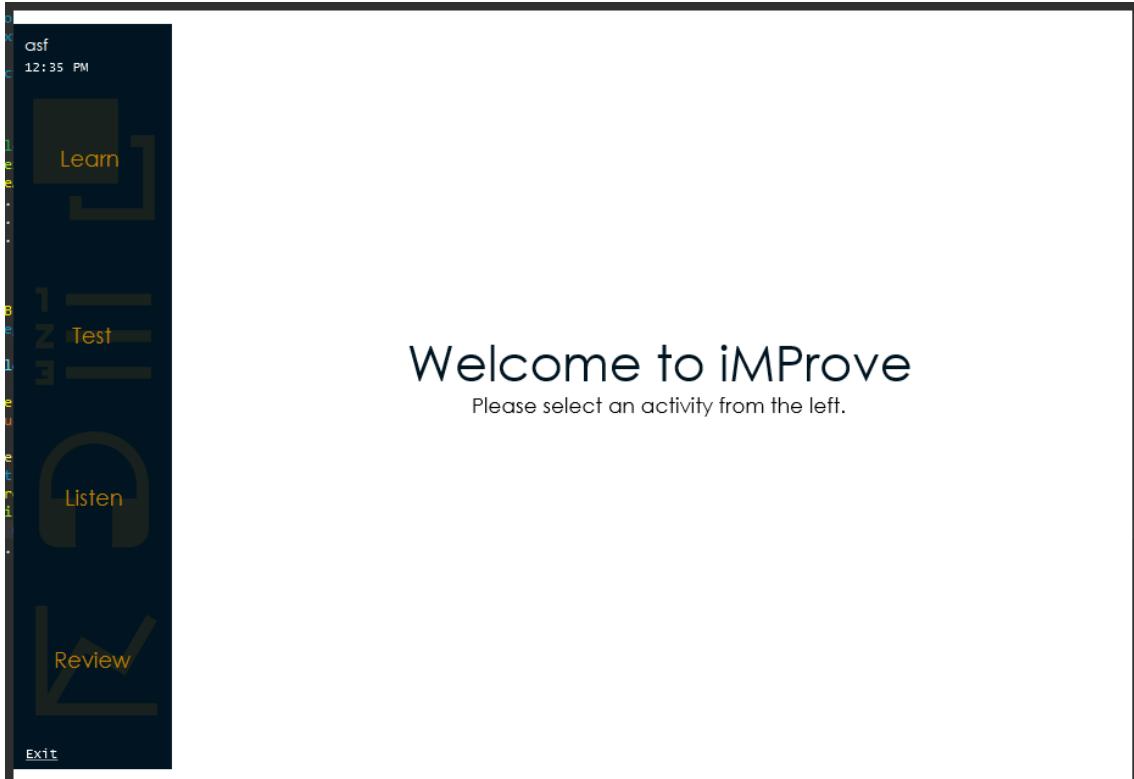
However, this allows for users to enter nothing in the fldName for their name. This is not ideal as we are wanting to use this name for later use. So, we must implement a dialog to display this (Alert API).

```

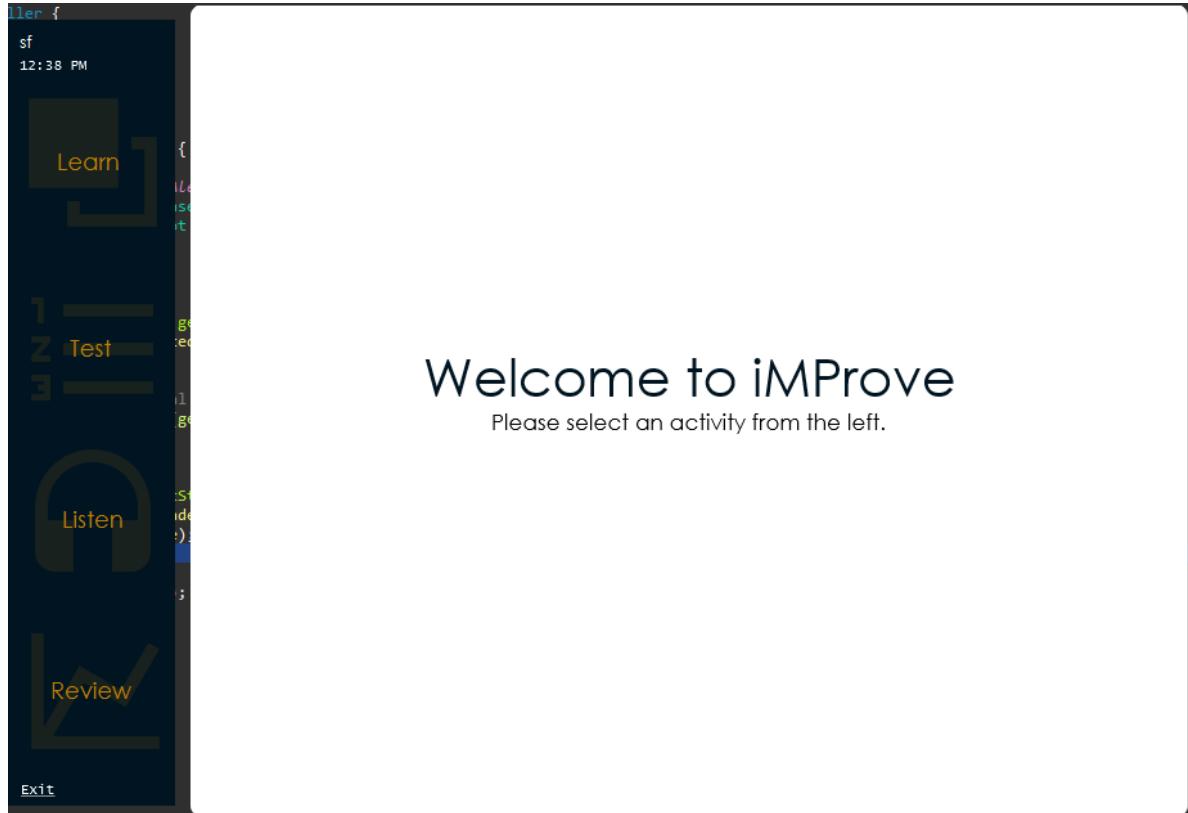
@FXML
public void gradeSelect(ActionEvent e) {
    if(fldName.getText().equals("")) {
        Alert noNameAlert = new Alert(AlertType.INFORMATION);
        noNameAlert.setHeaderText("Please type your name beside 'select your grade'.");
        noNameAlert.setTitle("You forgot to type your name in!");
        noNameAlert.showAndWait();
        return;
    }
    //get grade
    Button selectedButton = (Button) e.getSource();
    int grade = Integer.parseInt(selectedButton.getText());
    //get name
    String name = fldName.getText();
    //load root gui and set some initial values
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/root.fxml"));
    Parent root = null;
    try {
        root = loader.load();
    } catch(IOException ioe) {ioe.printStackTrace();}
    RootController rootController = loader.getController();
    rootController.initData(name, grade);
    Scene rootScene = new Scene(root);
    Main.getStage().setScene(rootScene);
}

```

Although not too much a problem of the actual programming logic, the `root.fxml` looked like this.



However, what we wanted to achieve was a transparency in the background so with `rootScene.setFill(null)`, and in combination with `stage.initStyle(StageStyle.Transparent)` in our `Main.java`, we removed the default white fill.



Using a static method provided in our `Main.java` class, we access the public Stage instance and set the scene to the `rootScene` (the scene containing all the activities).

Also, the `close()` method is used to handle the close button's event and exits the program.

Code for GradeController.java

```

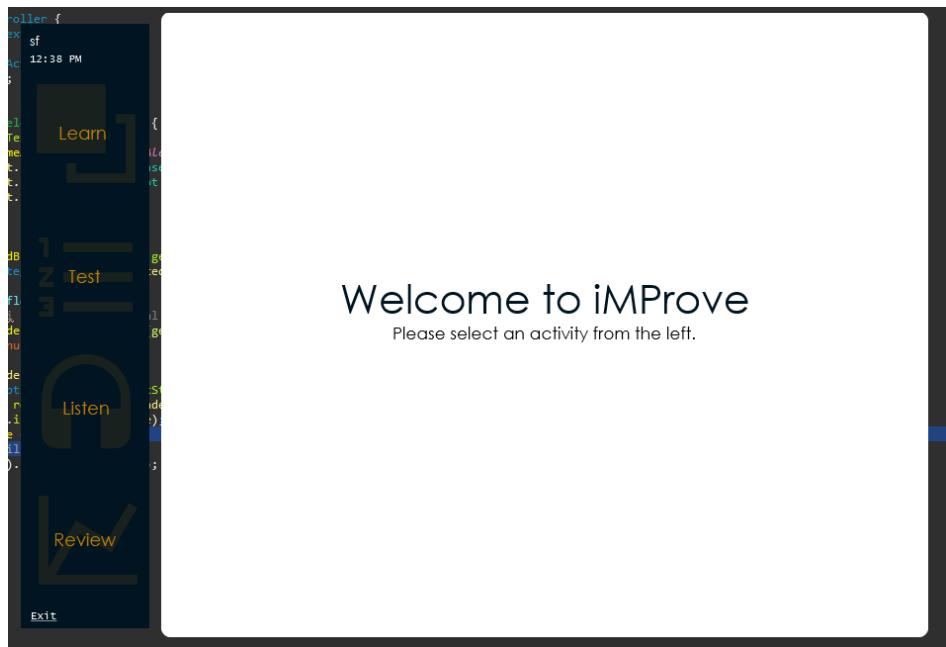
1 package controller;
2
3 import java.io.IOException;
4
5
6 public class GradeController {
7     @FXML private JFXTextField fldName;
8     @FXML
9     public void close(ActionEvent e) {
10         System.exit(0);
11     }
12     @FXML
13     public void gradeSelect(ActionEvent e) {
14         if(fldName.getText().equals("")) {
15             Alert noNameAlert = new Alert(AlertType.INFORMATION);
16             noNameAlert.setHeaderText("Please type your name beside 'select your grade'.");
17             noNameAlert.setTitle("You forgot to type your name in!");
18             noNameAlert.showAndWait();
19             return;
20         }
21         //get grade
22         Button selectedButton = (Button) e.getSource();
23         int grade = Integer.parseInt(selectedButton.getText());
24         //get name
25         String name = fldName.getText();
26         //load root gui and set some initial values
27         FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/root.fxml"));
28         Parent root = null;
29         try {
30             root = loader.load();
31         } catch(IOException ioe) {ioe.printStackTrace();}
32         RootController rootController = loader.getController();
33         rootController.initData(name, grade);
34         Scene rootScene = new Scene(root);
35         rootScene.setFill(null);
36         Main.getStage().setScene(rootScene);
37     }
38 }
39 }
```

Similarly, the `root.fxml` view was linked to a controller of class `RootController.java`.

RootController.java

Our RootController object will control what collectively is our template for the program – root.fxml.

The side navigation bar allows the user to navigate throughout the system wherever and whenever the user wants to. Also, the respective activity will be displayed in the large AnchorPane, apnActivity (the white pane which currently says ‘Welcome to iMProve’).



When the RootController first is initialised, a group of methods are called by the GradeController:

```
public void initData(String name, int grade) {  
    lblName.setText(name);  
    RootController.grade = grade;  
    RootController.name = name;  
}
```

The initData() method, called by the GradeController, initialises the private instance variables in the RootController class and sets the text of lblName at the top the navigation bar.

Initially, the control displaying the name was previously a Text control. However, this would stretch the navigation pane if the name was overly long. A Label, however, can be configured to show ellipses when the text was larger than the label dimensions.

```
oller {  
    Really Long Sample  
}
```

```
oller {  
    Really Long Sa...  
    01:33 PM  
}
```

```
public void initialize() {  
    Platform.runLater(()->{  
        apnRoot.requestFocus();  
        //automatically update time  
        startTime();  
    });  
}
```

initialize() is a method which is automatically called on initialisation of the object of the class (hence the name initialize()). apnRoot.requestFocus() allows the program to not highlight a random button but rather, focus on something which cannot be highlighted (i.e the apnRoot in this case). Platform.runLater(Runnable r) needs to be used to make the FX UI thread call the block of code since it is updating the UI, otherwise the code inside the Runnable (apn.requestFocus() and startTime()) will not be called. A *lambda expression* is used to replace the Runnable and the run() method within Runnable.

apnRoot.requestFocus() is used to make the FX UI thread call the block of code since it is updating the UI, otherwise the code inside the Runnable (apn.requestFocus() and startTime()) will not be called. A *lambda expression* is used to replace the Runnable and the run() method within Runnable.

`startTime()` is called to start the clock on the navigation bar, below the name.

```
public void startTime() {
    Timeline dateUpdater = new Timeline(
        new KeyFrame(Duration.seconds(0), event -> txtTime.setText(LocalTime.now().format(DateTimeFormatter.ofPattern("hh:mm a")))),
        new KeyFrame(Duration.seconds(1))); //duration 0 to do update time straight away and then seconds 1 to wait
    dateUpdater.setCycleCount(Animation.INDEFINITE);
    dateUpdater.play();
}
```

A Timeline is used, which is an animation, and is set the indefinitely play meaning that the clock will update indefinitely (until the user closes the program) rather than suddenly after, say, 5 minutes. At the zeroth second, an event handler sets the text of `txtTime` to the current time in the form hh:mm a meaning ‘hour (1–12, 2 digits): minute (2 digits) AM/PM’.

Prudently, the name and grade variables are static and can be publicly accessed via get methods (*encapsulation*). This will be helpful for other activities in the program when they need to access specific grades or display the name somewhere.

```
public static int getGrade() {
    return grade;
}
public static String getName() {
    return name;
}
```

When clicking on a JFXButton on the navigation bar, it will either call `goLearn()`, `goTest()`, `goListen()` or `goReview()` depending on the respective activity. They are extremely similar except they have different fxml files and different controllers.

For an example, the `goLearn()` method:

```
@FXML
public void goLearn(ActionEvent e) {
    |
}
```

This handled an `ActionEvent` (called `e` for event) which was fired by the user clicking a JFXButton.

We should load the UI for the learn activity and place it within `apnActivity`.

```
@FXML
public void goLearn(ActionEvent e) {
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn.fxml"));
    try {
        root = loader.load();
    } catch(IOException ioe) {ioe.printStackTrace();}
    LearnController controller = loader.getController();
    controller.setRootController(this);
}
```

Usually, invoking the method `load()` will return an object of same type as the root node of `learn.fxml` (which is `apnRoot`) so will be `AnchorPane`. However, we will use `Parent` instead which

is a superclass of AnchorPane and so will hide information regarding the specific object, making the code more clean and robust (since it will be harder to alter properties of the AnchorPane object). The LearnController object, controller (we can name it as vague as just controller since it has a small scope, the method, which will only have one controller object anyways – similar for root and loader), is given the RootController object so it can access important object methods to navigate throughout the system since the learn.fxml UI is contained inside root.fxml UI – explained in depth later.

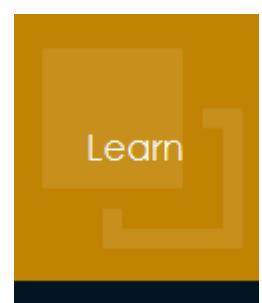
```
@FXML
public void goLearn(ActionEvent e) {
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn.fxml"));
    try {
        root = loader.load();
    } catch(IOException ioe) {ioe.printStackTrace();}
    LearnController controller = loader.getController();
    controller.setRootController(this);
    changeNavigationColours(spnLearn, icnLearn, btnLearn);
    setActivity(root);
}
```

At the end, we will change the navigation colours so that the user can know which activity is currently selected. The `setActivity()` method changes the activity on the screen to the one regarding Learn.

`changeNavigationColours()` changes the highlight and glow of the button indicating the current selected activity.

```
public void changeNavigationColours(StackPane spn, MaterialDesignIconView icn, JFXButton btn) {
    for(StackPane pane: Arrays.asList(spnLearn, spnTest, spnListen, spnReview)) {
        if(!pane.equals(spn))
            pane.setStyle("-fx-background-color: #001421");
        else
            pane.setStyle("-fx-background-color: #c18303");
    }
    for(MaterialDesignIconView icon: Arrays.asList(icnLearn, icnTest, icnListen, icnReview)) {
        if(!icon.equals(icn))
            icon.setFill(Color.web("#c18303", 0.13));
        else
            icon.setFill(Color.web("#FFFFFF", 0.13));
    }
    for(JFXButton button: Arrays.asList(btnLearn, btnTest, btnListen, btnReview)) {
        if(!button.equals(btn))
            button.setTextFill(Color.web("#c18303"));
        else
            button.setTextFill(Color.web("FFFFFF"));
    }
}
```

Checking each StackPane, MaterialDesignIconView, JFXButton, it changes the correct element from each of the three classes to an orange or white colour (#c18303 and #ffffff) and the rest to a dark blue and orange (#001421 and #c18303). Furthermore, the MaterialDesignIconView object, icon, will have an opacity of 13% (0.13) to give a tinted glow-like effect.



A very important method is the `setActivity()` method.

```
public void setActivity(Node root) {
    apnActivity.getChildren().clear();
    apnActivity.getChildren().add(root);
}
```

This essentially makes the UI loaded by the other FXML files, e.g. `learn.fxml` or `test.fxml` to be inside `apnActivity`. Previously added nodes are removed, and the new activity is added onto `apnActivity` (i.e. in the `goLearn()` method, the Parent containing the view of the Learn activity). This is public as it allows other class objects to access it and set the activity to something new.

Similarly to `GradeController`, the `RootController` contains a `close()` method allowing for the system to exit – called by user clicking the ‘Exit’ text.

```
@FXML
public void close(MouseEvent e) {
    System.exit(0);
}
```



Code for `RootController.java`:

```
1 package controller;
2
3 import java.io.IOException;
4
5 public class RootController {
6     @FXML private Label lblName;
7     @FXML private AnchorPane apnRoot, apnActivity;
8     @FXML private StackPane spnLearn, spnTest, spnListen, spnReview;
9     @FXML private MaterialDesignIconView icnLearn, icnTest, icnListen, icnReview;
10    @FXML private JFXButton btnLearn, btnTest, btnListen, btnReview;
11    @FXML private Text txtTime;
12    private static int grade;
13    private static String name;
14    public void initialize() {
15        Platform.runLater(() -> {
16            apnRoot.requestFocus();
17            //automatically update time
18            startTime();
19        });
20    }
21
22    public void initData(String name, int grade) {
23        lblName.setText(name);
24        this.grade = grade;
25        this.name = name;
26    }
27    public static int getGrade() {
28        return grade;
29    }
30    public static String getName() {
31        return name;
32    }
33    public void setActivity(Node root) {
34        apnActivity.getChildren().clear();
35        apnActivity.getChildren().add(root);
36    }
37    public void changeNavigationColours(StackPane spn, MaterialDesignIconView icn, JFXButton btn) {
38        for(StackPane pane: Arrays.asList(spnLearn, spnTest, spnListen, spnReview)) {
39            if(!pane.equals(spn))
40                pane.setStyle("-fx-background-color: #001421");
41            else
42
```

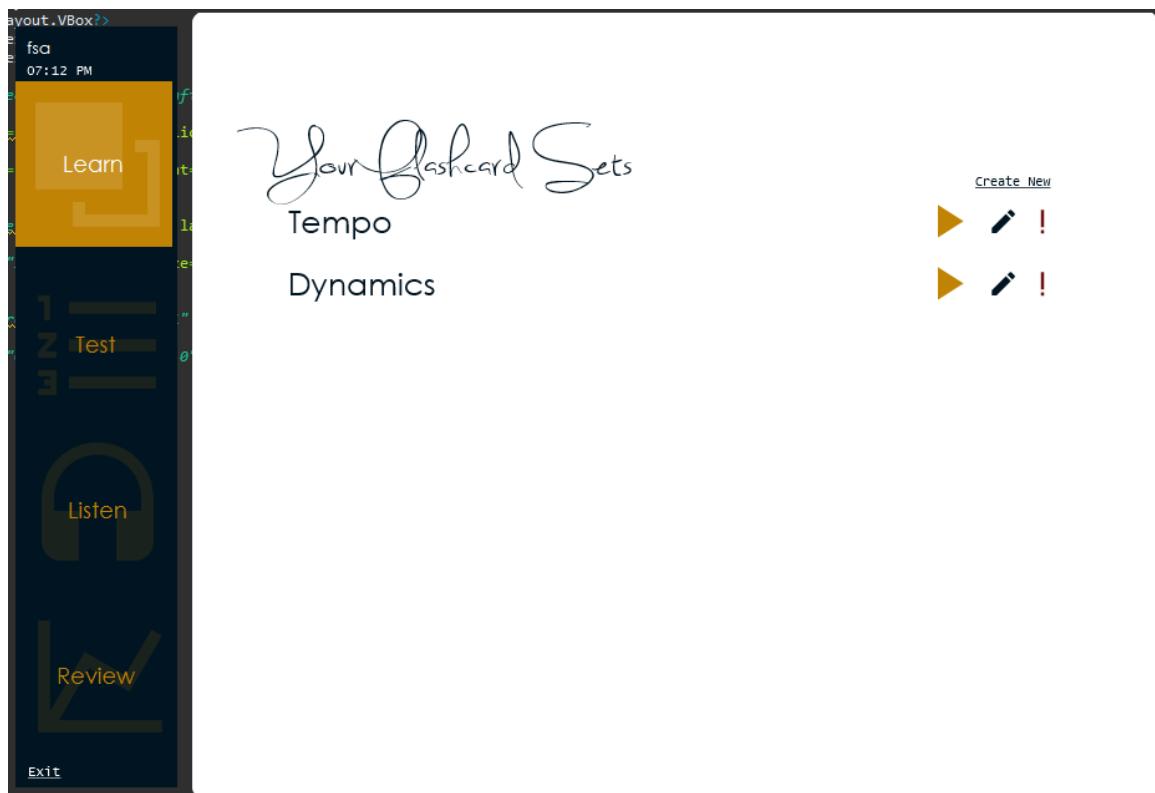
```

67         pane.setStyle("-fx-background-color: #c18303");
68     }
69     for(MaterialDesignIconView icon: Arrays.asList(icnLearn, icnTest, icnListen, icnReview)) {
70         if(!icon.equals(icn))
71             icon.setFill(Color.web("#c18303", 0.13));
72         else
73             icon.setFill(Color.web("#FFFFFF", 0.13));
74     }
75     for(JFXButton button: Arrays.asList(btnLearn, btnTest, btnListen, btnReview)) {
76         if(!button.equals(btn))
77             button.setTextFill(Color.web("#c18303"));
78         else
79             button.setTextFill(Color.web("FFFFFF"));
80     }
81 }
82 }
83 @FXML
84 public void startTime() {
85     Timeline dateUpdater = new Timeline(
86         new KeyFrame(Duration.seconds(0), event -> txtTime.setText(LocalTime.now().format(DateTimeFormatter.ofPattern("hh:mm a")))),
87         new KeyFrame(Duration.seconds(1))); //duration 0 to do update time straight away and then seconds 1 to wait
88     dateUpdater.setCycleCount(Animation.INDEFINITE);
89     dateUpdater.play();
90 }
91 @FXML
92 public void goLearn(ActionEvent e) {
93     Parent root = null;
94     FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn.fxml"));
95     try {
96         root = loader.load();
97     } catch(IOException ioe) {ioe.printStackTrace();}
98     LearnController controller = loader.getController();
99     controller.setRootController(this);
100    changeNavigationColours(spnLearn, icnLearn, btnLearn);
101    setActivity(root);
102 }
103 @FXML
104 public void goTest(ActionEvent e) {
105     Parent root = null;
106     FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/test.fxml"));
107     try {
108         root = loader.load();
109     } catch(IOException ioe) {ioe.printStackTrace();}
110     TestController controller = loader.getController();
111     controller.setRootController(this);
112     changeNavigationColours(spnTest, icnTest, btnTest);
113     setActivity(root);
114 }
115 @FXML
116 public void goListen(ActionEvent e) {
117     Parent root = null;
118     FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/listen.fxml"));
119     try {
120         root = loader.load();
121     } catch(IOException ioe) {ioe.printStackTrace();}
122     ListenController controller = loader.getController();
123     controller.setRootController(this);
124     changeNavigationColours(spnListen, icnListen, btnListen);
125     setActivity(root);
126 }
127 @FXML
128 public void goReview(ActionEvent e) {
129     Parent root = null;
130     FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/review.fxml"));
131     try {
132         root = loader.load();
133     } catch(IOException ioe) {ioe.printStackTrace();}
134     ReviewController controller = loader.getController();
135     controller.setRootController(this);
136     changeNavigationColours(spnReview, icnReview, btnReview);
137     setActivity(root);
138 }
139 @FXML
140 public void close(MouseEvent e) {
141     System.exit(0);
142 }
143 }
144 }
```

LearnController.java

For our first activity, ‘Learn’, we wanted to provide an alternative approach to learning (hence the name) and so provides a more laid-back environment for those who do not like the competitive, testing approach.

Therefore, we will be using a flashcards system as this alternative ‘relaxed’ approach to learning. However, LearnController.java is not the controller for the actual flashcards itself, but rather the navigation page that will allow the user to navigate throughout the flashcards system – including creating, deleting, editing and playing a respective flashcard set. The file containing the UI for this is learn.fxml.

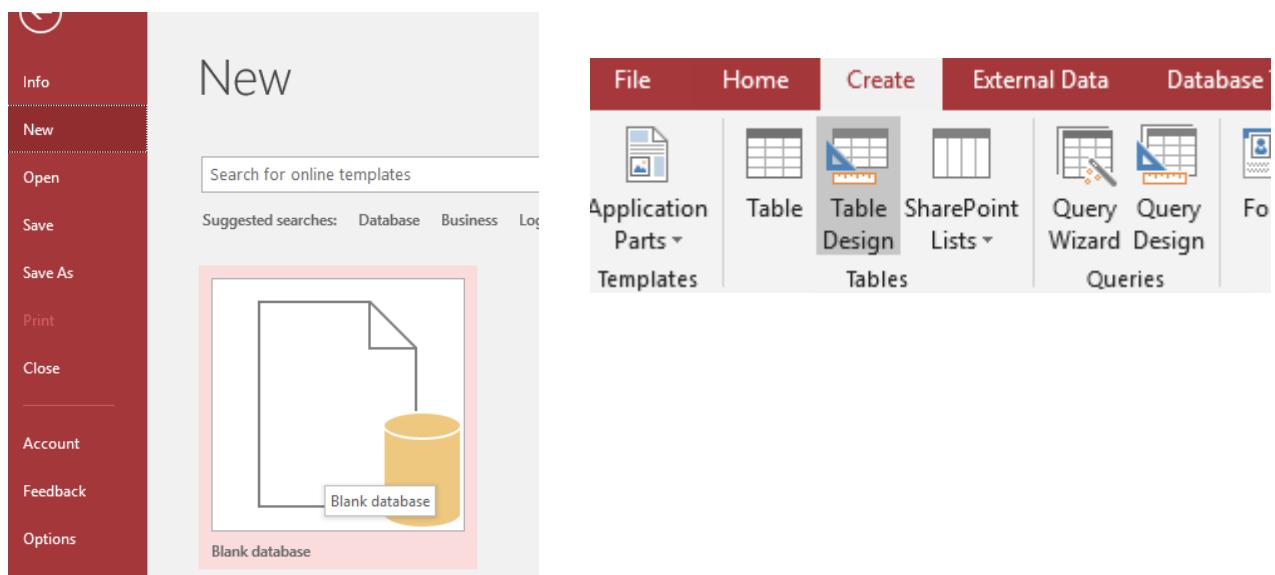


Note: above is what learn.fxml would look like in the program, but the learn.fxml view itself is not the whole screen above. It is merely a loaded node added to the root.fxml apnRoot via the RootController.

Below is an example of what just learn.fxml would look like with many rows of sets.



On first load-up, we want to display all the flashcard sets as a row on the VBox vbxSets. So firstly, we should create a database called improveDB.accdb and table Theory to represent the entity of a music theory term. Using Microsoft Access, we can easily create a new database and table.



Using *Design View*, we can easily design a somewhat data dictionary for the table's fields and modify the structure of the table.

Field Name	Data Type	Description (Optional)
ID	AutoNumber	Unique ID for each flashcard
Term	Short Text	Foreign term to be defined
Definition	Short Text	Definition of respective foreign term
Grade	Number	Grade difficulty of term
Title	Short Text	The title (sub-category) which the term relates to.

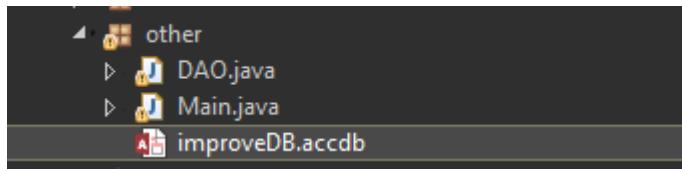
Here, we use the field ID to be the unique primary key for each term, fields Term and Definition will hold the respective strings defining the foreign term and its definition, also including the Grade it belongs to and the Title field allows us to categorise each term into separate sub-sections of musical theory.

Using online resources and physical books, we have defined all the terms an ABRSM student needs to know, up to grade 8.

ID	Term	Definition	Grade	Title
166	Misura	measure (e.g. time signature)	5	Metre
167	Ossia	or	5	Connectives
168	Piacevole	pleasant	5	Style
169	Piangevole	plaintive, sad	5	Style
170	Pochettino	rather little	5	Adverbs
171	Rinforzando (r)	reinforcing	5	Articulation
172	Segue	go straight on	5	Form
173	Smorzando (sn)	dying away in time	5	Style
174	Teneramente/	tenderly	5	Style
175	Tosto	swift	5	Tempo
176	Volante	flying, fast	5	Tempo
177	Aber	but	5	Connectives
178	Ausdruck	expression	5	Style
179	Bewegt	with movement	5	Tempo
180	Breit	broad	5	Tempo
181	Ein	one	5	Nouns
182	Einfach	simple	5	Style
183	Etwas	somewhat, rather	5	Adverbs
184	Fröhlich	cheerful	5	Style
185	Immer	always	5	Adverbs
186	Langsam	slow	5	Tempo
187	Lebhaft	lively	5	Style
188	Mässig	at a moderate	5	Tempo
189	Mit	with	5	Connectives
190	Nicht	not	5	Adverbs
191	Ohne	without	5	Connectives
192	Ruhig	peaceful	5	Style
193	Schnell	fast	5	Tempo
194	Sehr	very	5	Adverbs
195	Süss	sweet	5	Style
196	Traurig	sad	5	Style
197	Und	and	5	Connectives
198	Voll	full	5	Adverbs
199	Wenig	little	5	Adverbs
200	Wieder	again	5	Adverbs
201	Zart	tender	5	Style
202	Zu	to, too	5	Adverbs
*	(New)		0	

Note: there are more terms than shown above.

The database will be stored inside the program within the 'other' directory alongside DAO.java and Main.java.



Since the flashcard sets are stored inside a database on table Flashcard, we must create a SELECT query to fetch the data from the SQL database (Microsoft Access). We can do it with this String: `SELECT Title FROM Theory`. This String collects all titles stored in the database where each title represents a unique set. Collecting all the data from the database, we need to store it somewhere then put the information onto the UI. The code for this method, `initSets()`:

```
public void initSets() {
    try {
        Connection conn = dao.getConnection();
        PreparedStatement stmt = conn.prepareStatement("SELECT Title FROM Theory");
        ResultSet rs = stmt.executeQuery();
    } {
        //code to add the titles to a collection
    } catch (SQLException e) {
        e.printStackTrace();
    }
    //code to add the rows with respective titles to vbxSets
}
```

This uses a *try-with-resources* where the resources inside the try parentheses, (), are closed automatically at the end of the try block – this means the code is robust and will not be prone to errors or corruptions to the database.

```
public void initSets() {
    try {
        Connection conn = dao.getConnection();
        PreparedStatement stmt = conn.prepareStatement("SELECT Title FROM Theory");
        ResultSet rs = stmt.executeQuery();
    } {
        while(rs.next()) {
            String title = rs.getString("Title");
            titleSet.add(title);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

For a collection, we could be using ArrayLists, HashMaps, LinkedLists, Trees and indefinitely continued but in this case, we should use a Set as we do not want duplicates of the same title (each term on the database will have their respective title so multiple terms will lead to multiple repeated titles); otherwise vbxSets will be filled with duplicate rows.

```
private ObservableSet<String> titleSet = FXCollections.observableSet(); //only one unique title
```

The code above will use the results from the query identified inside the ResultSet object, rs, and iterating through each row until the end will reveal each title used in the set, this will be added to the Set, titleSet.

At the end of this method, we will just append these rows to vbxSets like so:

```
Platform.runLater(()-> {
    for(String s: titleSet) {
        addRow(s);
    }
});
```

where Platform.runLater(Runnable r) is used to let the FX Application Thread handle this code so that changes to the UI are guaranteed.

The addRow() method is:

```
public void addRow(String title) {
    Parent row = null;
    FXMLLoader rowLoader = new FXMLLoader(getClass().getResource("/view/learn_row.fxml"));
    try {
        row = rowLoader.load();
    } catch (IOException e) {e.printStackTrace();}
    LearnRowController rowController = rowLoader.getController();
    rowController.initData(title, rootController, this, apnLearn);
    vbxSets.getChildren().add(row);
}
```

This loads up the learn_row.fxml.



After, we can give the respective controller all the information it needs to get it working using initData(): title to display to the user; rootController to change activity to the flashcards; the LearnController object in case objects later will need to update the UI of the learn.fxml activity; and apnLearn to use for direct navigation back to the learn.fxml view. At the end, we just add this row item to the VBox holding all the sets.

In the top right corner of the main Learn screen, we will have the ability to create a new set.



The text will fire a MouseEvent (Clicked) event when clicked.

This needs to be handled by changing the screen to the update screen (since that is the screen allows editing or changing sets/new sets).

```
@FXML
public void createSet(MouseEvent e) {
    |
}
```

This is the method that will handle the MouseEvent thrown by the FXML view (hence @FXML annotation). Firstly, we must get the title the user wants to use for this title. We include validation to make sure the user does not input any unwanted entries.

```
@FXML  
public void createSet(MouseEvent e) {  
    //get the title  
    TextInputDialog titleDialog = new TextInputDialog("");  
    titleDialog.setTitle("Title for your set");  
    titleDialog.setHeaderText("Please provide a title for your new flashcard set");  
    Optional<String> result = titleDialog.showAndWait();  
    if(result.isPresent() && !result.get().equals("")) {  
        //do something  
    }  
}
```

Inside the *if-statement*, we shall update the activity to the update screen and give it the title we just collected (the result.get() variable) and various other objects to be used to change the UI.

```
@FXML  
public void createSet(MouseEvent e) {  
    //get the title  
    TextInputDialog titleDialog = new TextInputDialog("");  
    titleDialog.setTitle("Title for your set");  
    titleDialog.setHeaderText("Please provide a title for your new flashcard set");  
    Optional<String> result = titleDialog.showAndWait();  
    if(result.isPresent() && !result.get().equals("")) {  
        Parent root = null;  
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn_update.fxml"));  
        try {  
            root = loader.load();  
        } catch(IOException ex) {ex.printStackTrace();}  
        LearnUpdateController controller = loader.getController();  
        controller.initData(result.get(), rootController, this, apnLearn);  
        rootController.setActivity(root);  
    }  
}
```

Code for LearnController.java:

```
1 package controller;
2
3 import java.io.IOException;
4 import java.sql.Connection;
5 import java.sql.PreparedStatement;
6 import java.sql.ResultSet;
7 import java.sql.SQLException;
8 import java.util.Optional;
9
10 import javafx.application.Platform;
11 import javafx.collections.FXCollections;
12 import javafx.collections.ObservableList;
13 import javafx.collections.ObservableSet;
14 import javafx.fxml.FXML;
15 import javafx.fxml.FXMLLoader;
16 import javafx.scene.Parent;
17 import javafx.scene.control.TextInputDialog;
18 import javafx.scene.input.MouseEvent;
19 import javafx.scene.layout.AnchorPane;
20 import javafx.scene.layout.VBox;
21 import other.DAO;
22
23 public class LearnController {
24     private RootController rootController;
25     private ObservableSet<String> titleSet = FXCollections.observableSet(); //only one unique title
26     private DAO dao = new DAO();
27     @FXML private VBox vbxSets;
28     @FXML private AnchorPane apnLearn;
29     public LearnController() {
30         //pack the vbox with existing sets
31         initSets();
32     }
33     public void initSets() {
34         try {
35             Connection conn = dao.getConnection();
36             PreparedStatement stmt = conn.prepareStatement("SELECT Title FROM Theory");
37             ResultSet rs = stmt.executeQuery();
38         } {
39             while(rs.next()) {
40                 String title = rs.getString("Title");
41                 if(!"".equals(title))
42                     titleSet.add(title);
43             }
44         } catch (SQLException e) {
45             e.printStackTrace();
46         }
47         Platform.runLater(()-> {
48             for(String s: titleSet) {
49                 addRow(s);
50             }
51         });
52     }
53     public void addRow(String title) {
54         Parent row = null;
55         FXMLLoader rowLoader = new FXMLLoader(getClass().getResource("/view/learn_row.fxml"));
56         try {
57             row = rowLoader.load();
58         } catch (IOException e) {e.printStackTrace();}
59         LearnRowController rowController = rowLoader.getController();
60         rowController.initData(title, rootController, this, apnLearn);
61         vbxSets.getChildren().add(row);
62     }
63     public void setRootController(RootController rootController) {
```

```

64     this.rootController = rootController;
65 }
66
67 @FXML
68 public void createSet(MouseEvent e) {
69     //get the title
70     TextInputDialog titleDialog = new TextInputDialog("");
71     titleDialog.setTitle("Title for your set");
72     titleDialog.setHeaderText("Please provide a title for your new flashcard set");
73     Optional<String> result = titleDialog.showAndWait();
74     if(result.isPresent() && !result.get().equals("")) {
75         Parent root = null;
76         FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn_update.fxml"));
77         try {
78             root = loader.load();
79         } catch(IOException ex) {ex.printStackTrace();}
80         LearnUpdateController controller = loader.getController();
81         controller.initData(result.get(), rootController, this, apnLearn);
82         rootController.setActivity(root);
83     }
84 }
85
86 }
```

LearnRowController.java

The controller for the learn_row.fxml.



txtTitle will display the title of the flashcard set and this will be defined by the initData() called by the LearnController object.

```

public void initData (String title, RootController rc, LearnController lc, AnchorPane apn) {
    txtTitle.setText(title);
    rootController = rc;
    learnController = lc;
    apnLearn = apn;
}
```

Here we set the private instance variables of the class, so it will be easier to refer to by other methods.

The edit button (pencil icon) will allow the user to edit the respective set. All we need to do is display the learn_update.fxml and let that respective controller do the editing.

```

@FXML
public void editSet(MouseEvent e) {
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn_update.fxml"));
    try {
        root = loader.load();
    } catch(IOException ex) {ex.printStackTrace();}
    LearnUpdateController controller = loader.getController();
    controller.initData(txtTitle.getText(), rootController, learnController, apnLearn);
    rootController.setActivity(root);
}
```

The delete button (exclamation icon) we need to confirm the user's choice since this will permanently cause the set to disappear from history.

```
@FXML  
public void deleteSet(MouseEvent e) {  
    //confirm first  
    Alert confirmAlert = new Alert(AlertType.CONFIRMATION, "", ButtonType.YES, ButtonType.NO);  
    confirmAlert.setHeaderText("Are you sure you want to delete this set?");  
    Optional<ButtonType> result = confirmAlert.showAndWait();  
    if(result.get() == ButtonType.YES) {  
        //delete from database  
  
        //delete from interface  
    }  
}
```

Firstly, we need to delete the set from the database so that this delete will be persistent to future uses of the system. We should use a DELETE query: DELETE FROM Theory WHERE Title = ?. This deletes all terms with the respective title, identified by the parameter ?. Using the ? parameter rather than using String concatenation makes the query immune to SQL injection since the .setString() checks for this and validates the input.

```
//delete from database  
try {  
    Connection conn = dao.getConnection();  
    PreparedStatement stmt = conn.prepareStatement("DELETE FROM Theory WHERE Title = ?");  
} {  
    conn.setAutoCommit(false);  
    stmt.setString(1, txtTitle.getText());  
    stmt.executeUpdate();  
    conn.commit();  
} catch (SQLException ex) {  
    ex.printStackTrace();  
}
```

Setting the AutoCommit property of the connection to false allows us to manually commit the changes to the database so the actions are more predictable, and we can predict the time taken to run this SQL query rather than unpredictable AutoCommit.

Next, we should delete this set from the interface (the Learn screen).

```
//delete from interface  
VBox vbxCards = (VBox) apnSetRow.getParent();  
vbxCards.getChildren().remove(apnSetRow);
```

This will get the VBox where all the rows are stored. Then the respective row item will be removed from this VBox.

Finally, we have the most important button: the play button which will play the flashcards. Although vitally important, this will not be very abstract or sophisticated to handle as all we need to do is just display the view which will have their controller handle its events.

```

@FXML
public void playFlashcards(MouseEvent e) {
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn_flashcards.fxml"));
    try {
        root = loader.load();
    } catch(IOException ex) {ex.printStackTrace();}
    LearnFlashcardController controller = loader.getController();
    controller.initData(txtTitle.getText(), rootController, apnLearn);
    controller.initKeys();
    rootController.setActivity(root);
}

```

Loading the flashcards view and initialising some initial data for the controller.

Code for LearnRowController.java:

```

1 package controller;
2
3 import java.io.IOException;
4 import java.sql.Connection;
5 import java.sql.PreparedStatement;
6 import java.sql.SQLException;
7 import java.util.Optional;
8
9 import javafx.application.Platform;
10 import javafx.fxml.FXML;
11 import javafx.fxml.FXMLLoader;
12 import javafx.scene.Parent;
13 import javafx.scene.control.Alert;
14 import javafx.scene.control.Alert.AlertType;
15 import javafx.scene.control.ButtonType;
16 import javafx.scene.input.MouseEvent;
17 import javafx.scene.layout.AnchorPane;
18 import javafx.scene.layout.VBox;
19 import javafx.scene.text.Text;
20 import other.DAO;
21
22 public class LearnRowController {
23     private RootController rootController;
24     private LearnController learnController;
25     private AnchorPane apnLearn;
26     private DAO dao = new DAO();
27
28     @FXML private Text txtTitle;
29     @FXML private AnchorPane apnSetRow;
30     public void initData (String title, RootController rc, LearnController lc, AnchorPane apn) {
31         txtTitle.setText(title);
32         rootController = rc;
33         learnController = lc;
34         apnLearn = apn;
35     }
36
37     @FXML
38     public void playFlashcards(MouseEvent e) {
39         Parent root = null;
40         FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn_flashcards.fxml"));
41         try {
42             root = loader.load();
43         } catch(IOException ex) {ex.printStackTrace();}
44         LearnFlashcardController controller = loader.getController();
45         controller.initData(txtTitle.getText(), rootController, apnLearn);
46         controller.initKeys();
47         rootController.setActivity(root);
48     }
49     @FXML
50     public void editSet(MouseEvent e) {

```

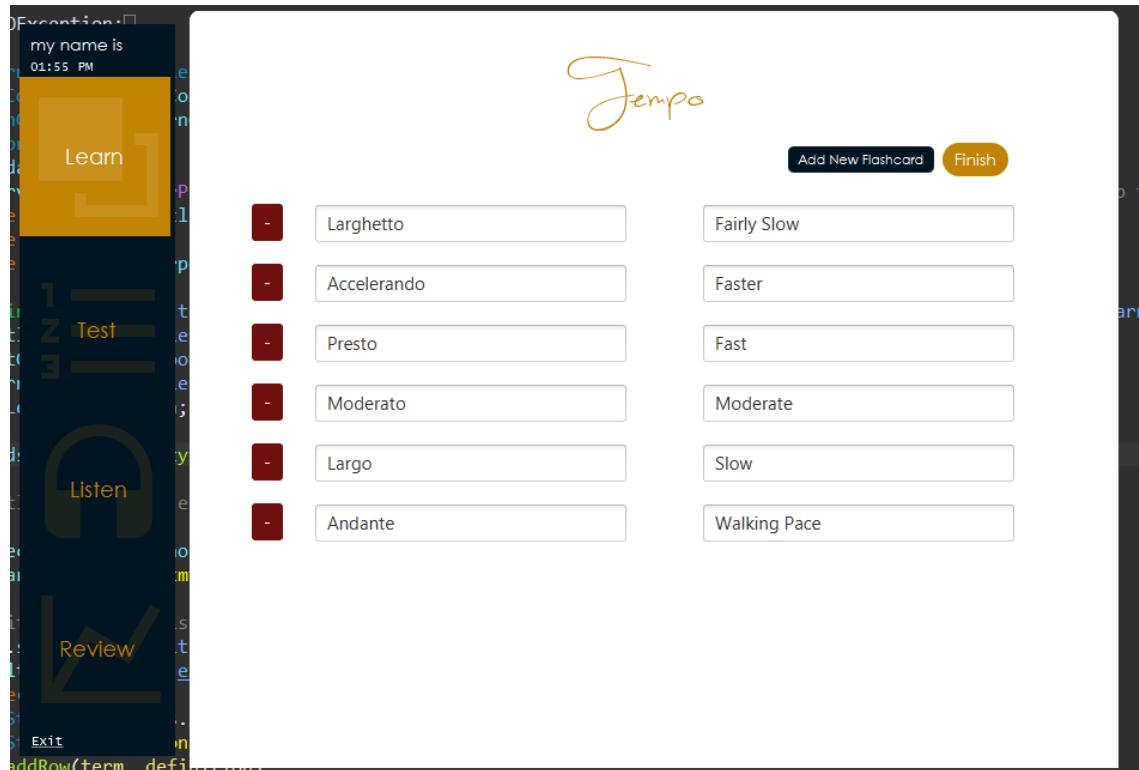
```

52     Parent root = null;
53     FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn_update.fxml"));
54     try {
55         root = loader.load();
56     } catch(IOException ex) {ex.printStackTrace();}
57     LearnUpdateController controller = loader.getController();
58     controller.initData(txtTitle.getText(), rootController, learnController, apnLearn);
59     rootController.setActivity(root);
60 }
61 @FXML
62 public void deleteSet(MouseEvent e) {
63     //confirm first
64     Alert confirmAlert = new Alert(AlertType.CONFIRMATION, "", ButtonType.YES, ButtonType.NO);
65     confirmAlert.setHeaderText("Are you sure you want to delete this set?");
66     Optional<ButtonType> result = confirmAlert.showAndWait();
67     if(result.get() == ButtonType.YES) {
68         //delete from database
69         try {
70             Connection conn = dao.getConnection();
71             PreparedStatement stmt = conn.prepareStatement("DELETE FROM Theory WHERE Title = ?");
72         } {
73             conn.setAutoCommit(false);
74             stmt.setString(1, txtTitle.getText());
75             stmt.executeUpdate();
76             conn.commit();
77         } catch (SQLException ex) {
78             ex.printStackTrace();
79         }
80         //delete from interface
81         VBox vbxCards = (VBox) apnSetRow.getParent();
82         vbxCards.getChildren().remove(apnSetRow);
83     }
84 }
85 }
86 }

```

LearnUpdateController.java

We will need a controller to control the updating of the flashcard sets. This includes editing the set or creating a whole new set.



Firstly, when the data is initialised, we can also get all the existing cards in the selected set so that we can display it on the UI. Hence, we iterate through every row (if any) in the ResultSet and add a row on the interface.

```
public void initData(String title, RootController rootController, LearnController learnController, AnchorPane apnLearn) {
    txtNewTitle.setText(title);
    this.rootController = rootController;
    this.learnController = learnController;
    this.apnLearn = apnLearn;

    //use title to load all existing cards
    try {
        Connection conn = dao.getConnection();
        PreparedStatement stmt = conn.prepareStatement("SELECT Term, Definition FROM Theory WHERE Title = ?");
    } {
        //initialise the existing flashcard rows
        stmt.setString(1, title);
        ResultSet rs = stmt.executeQuery();
        while(rs.next()) {
            String term = rs.getString("Term");
            String definition = rs.getString("Definition");
            addRow(term, definition);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Also, the Add New Flashcard button will also call the addRow() method

```
@FXML
public void addFlashcard(ActionEvent e) {
    addRow();
}
```

However, notice one version of this method has no arguments/parameters – this is called *overloading*.

```
public LearnUpdateRowController addRow(String term, String def) {
    LearnUpdateRowController controller = addRow();
    Platform.runLater(() -> {
        controller.setTerm(term);
        controller.setDef(def);
    });
    return controller;
}
public LearnUpdateRowController addRow() {
    AnchorPane row = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn_update_row.fxml"));
    try {
        row = loader.load();
    } catch(IOException e) {e.printStackTrace();}
    LearnUpdateRowController controller = loader.getController();
    controller.initData(this);
    controllerMap.put(row, controller);
    vbxCards.getChildren().add(row);
    return controller;
}
```

They are methods of the same name and they achieve the same thing (to add a new row onto the view) but the different arguments allow a higher range of customisability. In this case, the flashcards rows can be loaded with initial values from the database. Also, the controller of each row will be stored in an ObservableMap so that After pressing finish, the changes need to be

made to the database so that the flashcard sets will be persistent for future uses and throughout the rest of the program.

```

@FXML
public void finish(ActionEvent e) {
    try {
        Connection conn = dao.getConnection();
        PreparedStatement deleteStmt = conn.prepareStatement("DELETE FROM Theory WHERE Title = ?");
        PreparedStatement stmt = conn.prepareStatement("INSERT INTO Theory (Term, Definition, Grade, Title) VALUES (?, ?, ?, ?)");
    } {
        //delete old terms
        conn.setAutoCommit(false); //doesn't unpredictably commit the changes
        deleteStmt.setString(1, txtNewTitle.getText());
        deleteStmt.executeUpdate();

        for(LearnUpdateRowController c: controllerMap.values()) {
            if(!(c.getTerm().equals("") || c.getDef().equals("")))) { //not blank space
                stmt.setString(1, c.getTerm());
                stmt.setString(2, c.getDef());
                stmt.setInt(3, RootController.getGrade());
                stmt.setString(4, txtNewTitle.getText());
                stmt.addBatch();
            }
        }

        stmt.executeBatch();
        conn.commit();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    //go back to main learn screen
    rootController.setActivity(apnLearn);
    learnController.initSets();
}
}

```

We will use two SQL queries: “DELETE FROM Theory WHERE Title = ?” to delete the current flashcards from the set; “INSERT INTO Theory (Term, Definition, Grade, Title) VALUES (?, ?, ?, ?)” which will insert the new terms to table Theory.

Firstly, we delete the terms. Next, for each controller of each row in vbxCards we will insert a row to the database table Theory with the Term and Definition of the row, the grade of the current user and the title which the user defined earlier using the dialog.

Next, we go back to the learn screen by setting the activity to apnLearn and initialising the new sets. However, we encountered a problem in doing so.



After creating a new set, repeats in vbxSets occurred. So backtracking to where the rows were added, the method initSets() in LearnController.

```
public void initSets() {
    try {
        Connection conn = dao.getConnection();
        PreparedStatement stmt = conn.prepareStatement("SELECT Title FROM Theory");
        ResultSet rs = stmt.executeQuery();
    } {
        while(rs.next()) {
            String title = rs.getString("Title");
            if(!"".equals(title))
                titleSet.add(title);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    Platform.runLater(()-> {
        for(String s: titleSet) {
            addRow(s);
        }
    });
}
```

With closer inspection, all we are doing here is adding rows (and also controllers for these rows) with the Title identified from Theory. We never considered the rows that were already there in the first place. So, to tackle this problem, we remove all the controllers and rows in the Set and Vbox beforehand.

```
public void initSets() {
    titleSet.clear();
    Platform.runLater(()-> {
        vbxSets.getChildren().clear();
    }); //clear all children nodes first so no repeats of sets occur

    try {
        Connection conn = dao.getConnection();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT Title FROM Theory");
        while(rs.next()) {
            String title = rs.getString("Title");
            if(!titleSet.contains(title))
                titleSet.add(title);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Now, there are no repeats of flashcard sets.



Additionally, an extreme input where the user creates a set but without any cards like so:



Note: the text ‘Foreign Term’ and ‘Definition’ is only prompt text so retrieving the text from these text fields will return nothing, null.

When this is created by clicking the *finish* button, a long list of errors occurs (the list continues):

```
net.ucanaccess.jdbc.UcanaccessSQLException: UCAExc:::4.0.4 statement is not in batch mode
at net.ucanaccess.jdbc.UcanaccessStatement.executeBatch(UcanaccessStatement.java:205)
at controller.LearnUpdateController.finish(LearnUpdateController.java:111)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at sun.reflect.misc.Trampoline.invoke(MethodUtil.java:71)
at sun.reflect.GeneratedMethodAccessor.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at sun.reflect.misc.MethodUtil.invoke(MethodUtil.java:275)
at javafx.fxml.FXMLLoader$MethodHandler.invoke(FXMLLoader.java:1769)
at javafx.fxml.FXMLLoader$ControllerMethodEventHandler.handle(FXMLLoader.java:1657)
at com.sun.javafx.event.CompositeEventHandler.dispatchBubblingEvent(CompositeEventHandler.java:86)
at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:238)
at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:191)
at com.sun.javafx.event.CompositeEventDispatcher.dispatchBubblingEvent(CompositeEventDispatcher.java:59)
at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:58)
at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
at com.sun.javafx.event.EventUtil.fireEvent(EventUtil.java:74)
at com.sun.javafx.event.EventUtil.fireEvent(EventUtil.java:49)
at javafx.event.Event.fireEvent(Event.java:198)
at javafx.scene.Node.fireEvent(Node.java:8411)
at javafx.scene.control.Button.fire(Button.java:185)
at com.sun.javafx.scene.control.behavior.ButtonBehavior.mouseReleased(ButtonBehavior.java:182)
at com.sun.javafx.scene.control.skin.BehaviorSkinBase$1.handle(BehaviorSkinBase.java:96)
at com.sun.javafx.scene.control.skin.BehaviorSkinBase$1.handle(BehaviorSkinBase.java:89)
at com.sun.javafx.event.CompositeEventHandler$NormalEventFilter.handleBubblingEvent(CompositeEventHandler.java:218)
at com.sun.javafx.event.CompositeEventHandler.dispatchBubblingEvent(CompositeEventHandler.java:86)
at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:238)
at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:191)
at com.sun.javafx.event.CompositeEventDispatcher.dispatchBubblingEvent(CompositeEventDispatcher.java:59)
```

Although the error is very vague ‘statement is not in batch mode’, the only place where we used batch query was in the *finish()* method. Batch allows us to query the database all at once with the same form of query but with different values of parameters identified by the ?. This increases the overall efficiency as we only use the connection to the database once at the end.

```

@FXML
public void finish(ActionEvent e) {
    try {
        Connection conn = dao.getConnection();
        PreparedStatement deleteStmt = conn.prepareStatement("DELETE FROM Theory WHERE Title = ?");
        PreparedStatement stmt = conn.prepareStatement("INSERT INTO Theory (Term, Definition, Grade, Title) VALUES (?, ?, ?, ?)");
    } {
        //delete old terms
        conn.setAutoCommit(false); //doesn't unpredictably commit the changes
        deleteStmt.setString(1, txtNewTitle.getText());
        deleteStmt.executeUpdate();

        for(LearnUpdateRowController c: controllerMap.values()) {
            if(!(c.getTerm().equals("") || c.getDef().equals(""))){ //not blank space
                stmt.setString(1, c.getTerm());
                stmt.setString(2, c.getDef());
                stmt.setInt(3, RootController.getGrade());
                stmt.setString(4, txtNewTitle.getText());
                stmt.addBatch();
            }
        }
        stmt.executeBatch();
        conn.commit();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    //go back to main learn screen
    rootController.setActivity(apnLearn);
    learnController.initSets();
}

```

If the statement is not in batch mode, this could infer that I have typed my SQL query incorrectly, or there is a problem with the batch itself. Since the SQL query is correct after numerous checks of syntax, if we look closer at the for loop, we use an *if-statement* which adds batches if the TextField is not blank. The set we just added had blank TextField's so this skipped this for loop and added no batches. Since no batches were added, we can assume that this was the cause of the error when executing this batch. To combat this, we just use an *if-statement* to only execute when there are terms in the TextField's.

```

@FXML
public void finish(ActionEvent e) {
    try {
        Connection conn = dao.getConnection();
        PreparedStatement deleteStmt = conn.prepareStatement("DELETE FROM Theory WHERE Title = ?");
        PreparedStatement stmt = conn.prepareStatement("INSERT INTO Theory (Term, Definition, Grade, Title) VALUES (?, ?, ?, ?)");
    } {
        //delete old terms
        conn.setAutoCommit(false); //doesn't unpredictably commit the changes
        deleteStmt.setString(1, txtNewTitle.getText());
        deleteStmt.executeUpdate();

        boolean termsAdded = false;
        for(LearnUpdateRowController c: controllerMap.values()) {
            if(!(c.getTerm().equals("") || c.getDef().equals(""))){ //not blank space
                termsAdded = true;
                stmt.setString(1, c.getTerm());
                stmt.setString(2, c.getDef());
                stmt.setInt(3, RootController.getGrade());
                stmt.setString(4, txtNewTitle.getText());
                stmt.addBatch();
            }
        }
        if(termsAdded) //if terms WERE added, then execute batch otherwise will cause error on adding nothing
            stmt.executeBatch();

        conn.commit();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    //go back to main learn screen
    rootController.setActivity(apnLearn);
    learnController.initSets();
}

```

Here, we have a Boolean, `termsAdded`, which is set to true only after a batch has been added to the query. At the end, if it is true, meaning that there are terms to be added, execute this batch.

Code for LearnUpdateController.java:

```
1 package controller;
2
3 import java.io.IOException;
4 import java.sql.Connection;
5 import java.sql.PreparedStatement;
6 import java.sql.ResultSet;
7 import java.sql.SQLException;
8
9 import javafx.application.Platform;
10 import javafx.collections.FXCollections;
11 import javafx.collections.ObservableMap;
12 import javafx.event.ActionEvent;
13 import javafx.fxml.FXML;
14 import javafx.fxml.FXMLLoader;
15 import javafx.scene.control.ScrollPane;
16 import javafx.scene.layout.AnchorPane;
17 import javafx.scene.layout.VBox;
18 import javafx.scene.text.Text;
19 import other.DAO;
20
21 public class LearnUpdateController {
22     private RootController rootController;
23     private LearnController learnController;
24     private AnchorPane apnLearn;
25     private DAO dao = new DAO();
26
27     //map to refer to the respective controllers when needed
28     private ObservableMap<AnchorPane, LearnUpdateRowController> controllerMap = FXCollections.observableHashMap();
29
30     @FXML private Text txtNewTitle;
31     @FXML private VBox vbxCards;
32     @FXML private ScrollPane scrnCards;
33
34     public void initData(String title, RootController rootController, LearnController learnController, AnchorPane apnLearn) {
35         txtNewTitle.setText(title);
36         this.rootController = rootController;
37         this.learnController = learnController;
38         this.apnLearn = apnLearn;
39
40         scrnCards.vvvalueProperty().bind(vbxCards.heightProperty()); //automatically scroll the scrollpane to the bottom
41
42         //use title to load all existing cards
43         try {
44             Connection conn = dao.getConnection();
45             PreparedStatement stmt = conn.prepareStatement("SELECT Term, Definition FROM Theory WHERE Title = ?");
46         } {
47             //initialise the existing flashcard rows
48             stmt.setString(1, title);
49             ResultSet rs = stmt.executeQuery();
50             while(rs.next()) {
51                 String term = rs.getString("Term");
52                 String definition = rs.getString("Definition");
53                 addRow(term, definition);
54             }
55         } catch (SQLException e) {
56             e.printStackTrace();
57         }
58
59     }
60     public ObservableMap<AnchorPane, LearnUpdateRowController> getControllerMap() {
61         return controllerMap;
62     }
63     public LearnUpdateRowController addRow(String term, String def) {
64         LearnUpdateRowController controller = addRow();
65         Platform.runLater(() -> {
66             controller.setTerm(term);
67             controller.setDef(def);
68         });
69         return controller;
70     }
71     public LearnUpdateRowController addRow() {
72         AnchorPane row = null;
73         FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn_update_row.fxml"));
74         try {
75             row = loader.load();
76         } catch(IOException e) {e.printStackTrace();}
77         LearnUpdateRowController controller = loader.getController();
78         controller.initData(this);
79         controllerMap.put(row, controller);
```

```

80     vbxCards.getChildren().add(row);
81     return controller;
82 }
83 @FXML
84 public void addFlashcard(ActionEvent e) {
85     addRow();
86 }
87 @FXML
88 public void finish(ActionEvent e) {
89     try {
90         Connection conn = dao.getConnection();
91         PreparedStatement deleteStmt = conn.prepareStatement("DELETE FROM Theory WHERE Title = ?");
92         PreparedStatement stmt = conn.prepareStatement("INSERT INTO Theory (Term, Definition, Grade, Title) VALUES (?, ?, ?, ?)");
93     } {
94         //delete old terms
95         conn.setAutoCommit(false); //doesn't unpredictably commit the changes
96         deleteStmt.setString(1, txtNewTitle.getText());
97         deleteStmt.executeUpdate();
98
99         boolean termsAdded = false;
100        for(LearnUpdateRowController c: controllerMap.values()) {
101            if(!c.getTerm().equals("") || c.getDef().equals("")) { //not blank space
102                termsAdded = true;
103                stmt.setString(1, c.getTerm());
104                stmt.setString(2, c.getDef());
105                stmt.setInt(3, RootController.getGrade());
106                stmt.setString(4, txtNewTitle.getText());
107                stmt.addBatch();
108            }
109        }
110        if(termsAdded) //if terms WERE added, then execute batch otherwise will cause error on adding nothing
111            stmt.executeBatch();
112
113        conn.commit();
114    } catch (SQLException ex) {
115        ex.printStackTrace();
116    }
117
118    //go back to main learn screen
119    rootController.setActivity(apnLearn);
120    learnController.initSets();
121 }
122 }

```

LearnUpdateRowController.java

This controlled each row learn_update_row.fxml containing the editable TextFields for each card for the new/updated set.



The delete button is handled with the method removeCard() which removes the card from the VBox containing it, vbxCards.

```

@FXML
public void removeCard(ActionEvent e) {
    VBox vbxCards = (VBox) apnCardRow.getParent(); //remove from update interface
    vbxCards.getChildren().remove(apnCardRow);
    //remove controller
    updateController.getControllerMap().remove(apnCardRow);
}

```

This is very similar to the learn.fxml screen removing the control from the parent and also the controller from its Map.

The class overall is very simple, but it uses effectively encapsulation with its getter and setter methods.

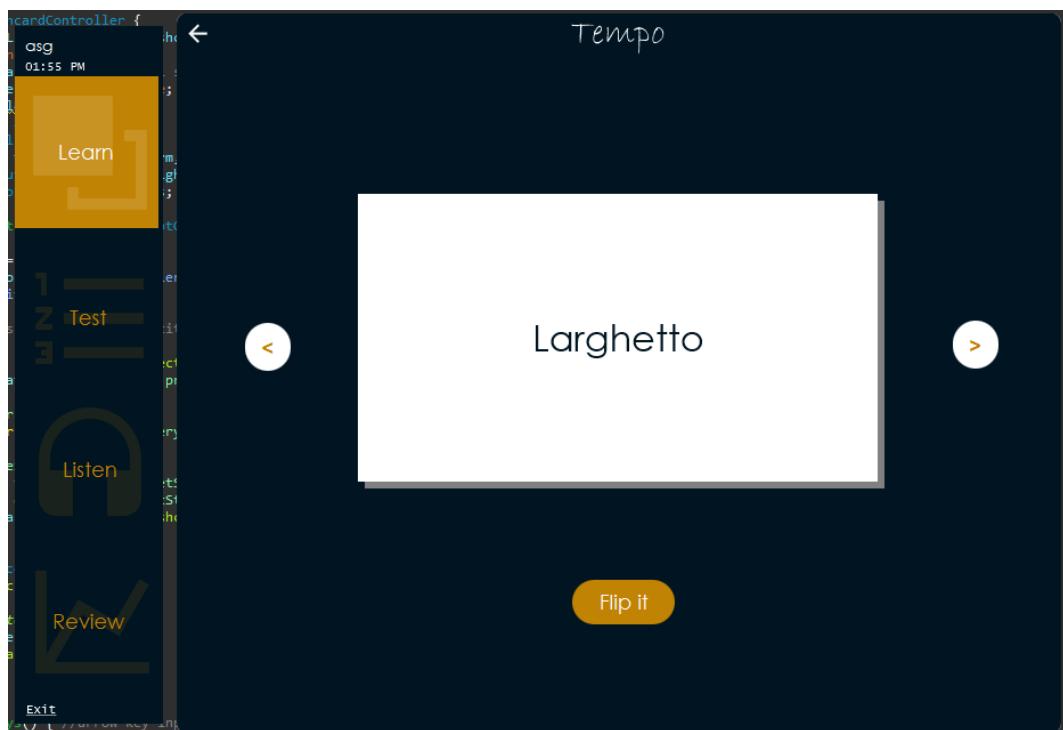
```

1 package controller;
2
3 import javafx.event.ActionEvent;
4 import javafx.fxml.FXML;
5 import javafx.scene.control.TextField;
6 import javafx.scene.layout.AnchorPane;
7 import javafx.scene.layout.VBox;
8
9 public class LearnUpdateRowController {
10     @FXML private TextField fldTerm, fldDef;
11     @FXML private AnchorPane apnCardRow;
12     private LearnUpdateController updateController = null;
13     public void initData(LearnUpdateController c) {
14         updateController = c;
15     }
16     @FXML
17     public void removeCard(ActionEvent e) {
18         VBox vbxCards = (VBox) apnCardRow.getParent(); //remove from update interface
19         vbxCards.getChildren().remove(apnCardRow);
20         //remove controller
21         updateController.getControllerMap().remove(apnCardRow);
22     }
23     public void setTerm(String term) { //set foreign term
24         fldTerm.setText(term);
25     }
26     public void setDef(String def) { //set definition
27         fldDef.setText(def);
28     }
29     public String getTerm() {
30         return fldTerm.getText();
31     }
32     public String getDef() {
33         return fldDef.getText();
34     }
35 }

```

LearnFlashcardController.java

The main part of this Learn section, the flashcards. This controller will control the flashcards view of the program, learn_flashcards.fxml



When the controller object is initialised, we get all the terms under the title of the set and add them to a list for later use. Also, we load the first flashcard with `loadFlashcard()`

```
public void initData(String title, RootController rootController, AnchorPane apnLearn) {
    //set values
    this.apnLearn = apnLearn;
    this.rootController = rootController;
    this.title = title;

    //get the cards relating to that title
    try {
        Connection conn = dao.getConnection();
        PreparedStatement stmt = conn.prepareStatement("SELECT Term, Definition FROM Theory Where Title = ?");
    } {
        stmt.setString(1, title);
        ResultSet rs = stmt.executeQuery();

        while(rs.next()) {
            String foreignTerm = rs.getString("Term");
            String definition = rs.getString("Definition");
            flashcardList.add(new Flashcard(foreignTerm, definition));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    Platform.runLater(()-> {
        txtSetTitle.setText(title);
        loadFlashcard();
    });
}
```

`loadFlashcard()` relies on an instance variable called `flashcardIndex` which refers to the index of the flashcard in the list.

```
private int flashcardIndex = 0; //will start at first card
```

This method is used to load the respective flashcard of the list.

```
public void loadFlashcard() {
    //load the flashcard with the new flashcardIndex
    Flashcard currentCard = flashcardList.get(flashcardIndex);
    txtTerm.setText(currentCard.getTerm());
    txtDef.setText(currentCard.getDefinition());

    //show foreign term first
    if(!questionVisible) {
        txtTerm.setVisible(true);
        txtDef.setVisible(false);
        questionVisible = true; //turn true
    }
}
```

Also, a subtle difference is that we always display the question first.

Next, the key inputs are initialised.

```

public void initKeys() { //arrow key input to use flashcards
    apnFlashcards.setOnKeyPressed(new EventHandler<KeyEvent>() { //random variable to refer to the scene
        @Override
        public void handle(KeyEvent event) {
            switch(event.getCode()) {
                case UP: flip();
                break;
                case DOWN: flip();
                break;
                case LEFT: prev();
                break;
                case RIGHT: next();
                break;
                case SPACE: flip();
                break;
            }
            event.consume();
        }
    });
}

```

This allows input from the keys to do things which the respective buttons do – making the flashcards more easy and efficient to use. Here, we have used a *switch case* selection to make the handling cleaner with lots of options. We only specify UP, DOWN, LEFT, RIGHT, SPACE, since they are ENUM constants of type Keycode.

The JFXButton inputs:

```

//flip card
@FXML
public void flipCard(MouseEvent e) {
    flip();
}
//next/prev card
@FXML
public void moveCard(ActionEvent e) {
    JFXButton clickedButton = (JFXButton) e.getSource();
    if(clickedButton.equals(btnLeft))
        prev();
    if(clickedButton.equals(btnRight))
        next();
}

```

They have the same job as the keys, but they are using the buttons to control the flashcards.

For flashcard mobility, we will use something similar to a circular queue where we move around the flashcardList indefinitely.

```

public void next() {
    flashcardIndex = (flashcardIndex + 1) % flashcardList.size(); //loop around the list like a circular queue
    loadFlashcard();
}
public void prev() {
    flashcardIndex--;
    if(flashcardIndex < 0) flashcardIndex = flashcardList.size() - 1; //go to last part of list if it goes below 0
    loadFlashcard();
}

```

For flipping a card, we only need to change the Text controls and the questionVisible Boolean.

```

public void flip() {
    txtTerm.setVisible(!questionVisible); //the term visibility will always become the opposite of what it was before
    txtDef.setVisible(questionVisible); //vice versa for the definition
    questionVisible = !questionVisible;
}

```

We also allow the user to go back to the main learn screen with a back button by setting the activity to the apnLearn AnchorPane.

```
@FXML  
public void goBack(MouseEvent e) {  
    rootController.setActivity(apnLearn);  
}
```

However, when testing it, the space bar sometimes did nothing or even do a different thing than what we want, flip(). By realising that the space bar only clicked what button was highlighted, it produced unpredictable behaviour, so we requested focus to a control which had no event handler and fired no events in the first place, apnFlashcards.

```
public void loadFlashcard() {  
    //load the flashcard with the new flashcardIndex  
    Flashcard currentCard = flashcardList.get(flashcardIndex);  
    txtTerm.setText(currentCard.getTerm());  
    txtDef.setText(currentCard.getDefinition());  
  
    //show foreign term first  
    if(!questionVisible) {  
        txtTerm.setVisible(true);  
        txtDef.setVisible(false);  
        questionVisible = true; //turn true  
    }  
  
    apnFlashcards.requestFocus();  
}
```

Code for LearnFlashcardsController.java

```
1 package controller;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7
8 import com.jfoenix.controls.JFXButton;
9
10 import javafx.application.Platform;
11 import javafx.collections.FXCollections;
12 import javafx.collections.ObservableList;
13 import javafx.event.ActionEvent;
14 import javafx.event.EventHandler;
15 import javafx.fxml.FXML;
16 import javafx.scene.input.KeyEvent;
17 import javafx.scene.input.MouseEvent;
18 import javafx.scene.layout.AnchorPane;
19 import javafx.scene.text.Text;
20 import model.Flashcard;
21 import other.DAO;
22
23 public class LearnFlashcardController {
24     private ObservableList<Flashcard> flashcardList = FXCollections.observableArrayList();
25     private DAO dao = new DAO();
26     private int flashcardIndex = 0; //will start at first card
27     private boolean questionVisible = true;
28     private String title;
29     private AnchorPane apnLearn;
30     private RootController rootController;
31     @FXML private Text txtSetTitle, txtTerm, txtDef;
32     @FXML private JFXButton btnLeft, btnRight;
33     @FXML private AnchorPane apnFlashcards;
34
35     public void initData(String title, RootController rootController, AnchorPane apnLearn) {
36         //set values
37         this.apnLearn = apnLearn;
38         this.rootController = rootController;
39         this.title = title;
40
41         //get the cards relating to that title
42         try {
43             Connection conn = dao.getConnection();
44             PreparedStatement stmt = conn.prepareStatement("SELECT Term, Definition FROM Theory Where Title = ?");
45         } {
46             stmt.setString(1, title);
47             ResultSet rs = stmt.executeQuery();
48
49             while(rs.next()) {
50                 String foreignTerm = rs.getString("Term");
51                 String definition = rs.getString("Definition");
52                 flashcardList.add(new Flashcard(foreignTerm, definition));
53             }
54
55         } catch (SQLException e) {
56             e.printStackTrace();
57         }
58         Platform.runLater(() -> {
59             txtSetTitle.setText(title);
60             loadFlashcard();
61         });
62     }
63
64     public void initKeys() { //arrow key input to use flashcards
65         apnFlashcards.setOnKeyPressed(new EventHandler<KeyEvent>() { //random variable to refer to the scene
66             @Override
67             public void handle(KeyEvent event) {
68                 switch(event.getCode()) {
```

```

69             case UP: flip();
70                 break;
71             case DOWN: flip();
72                 break;
73             case LEFT: prev();
74                 break;
75             case RIGHT: next();
76                 break;
77             case SPACE: flip();
78                 break;
79         }
80         event.consume();
81     });
82 }
83
84
85 public void loadFlashcard() {
86     //load the flashcard with the new flashcardIndex
87     Flashcard currentCard = flashcardList.get(flashcardIndex);
88     txtTerm.setText(currentCard.getTerm());
89     txtDef.setText(currentCard.getDefinition());
90
91     //show foreign term first
92     if(!questionVisible) {
93         txtTerm.setVisible(true);
94         txtDef.setVisible(false);
95         questionVisible = true; //turn true
96     }
97 }
98
99 public void flip() {
100     txtTerm.setVisible(!questionVisible); //the term visibility will always become the opposite of what it was before
101     txtDef.setVisible(questionVisible); //vice versa for the definition
102     questionVisible = !questionVisible;
103 }
104
105 public void next() {
106     flashcardIndex = (flashcardIndex + 1) % flashcardList.size(); //loop around the list like a circular queue
107     loadFlashcard();
108 }
109
110 public void prev() {
111     flashcardIndex--;
112     if(flashcardIndex < 0) flashcardIndex = flashcardList.size() - 1; //go to last part of list if it goes below 0
113     loadFlashcard();
114 }
115
116 //flip card
117 @FXML
118 public void flipCard(MouseEvent e) {
119     flip();
120 }
121
122 //next/prev card
123 @FXML
124 public void moveCard(ActionEvent e) {
125     JFXButton clickedButton = (JFXButton) e.getSource();
126     if(clickedButton.equals(btnLeft))
127         prev();
128     if(clickedButton.equals(btnRight))
129         next();
130 }
131
132 }

```

Flashcard.java

Now, a quick traversal to the model directory. We have an object class which will be the structure for each flashcard we make in the learn section.

This is just a simple class employing encapsulation and stores the flashcard term and definition so that each flashcard made from the database will be easier to store inside the program.

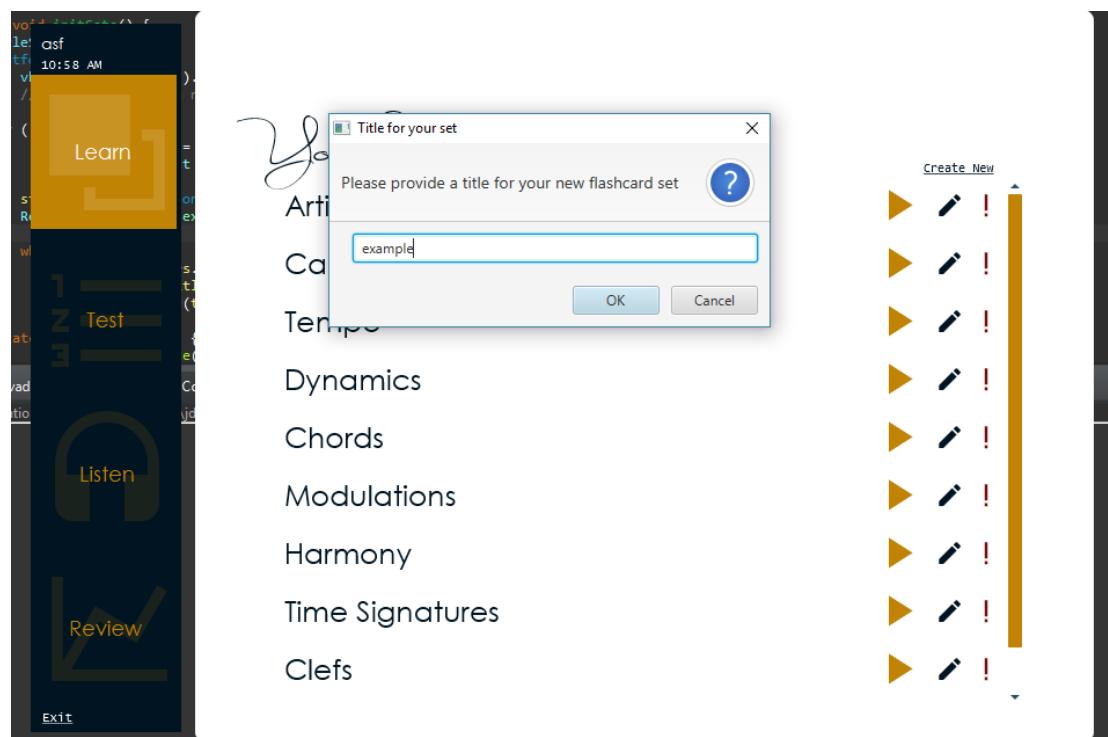
Code for Flashcard.java:

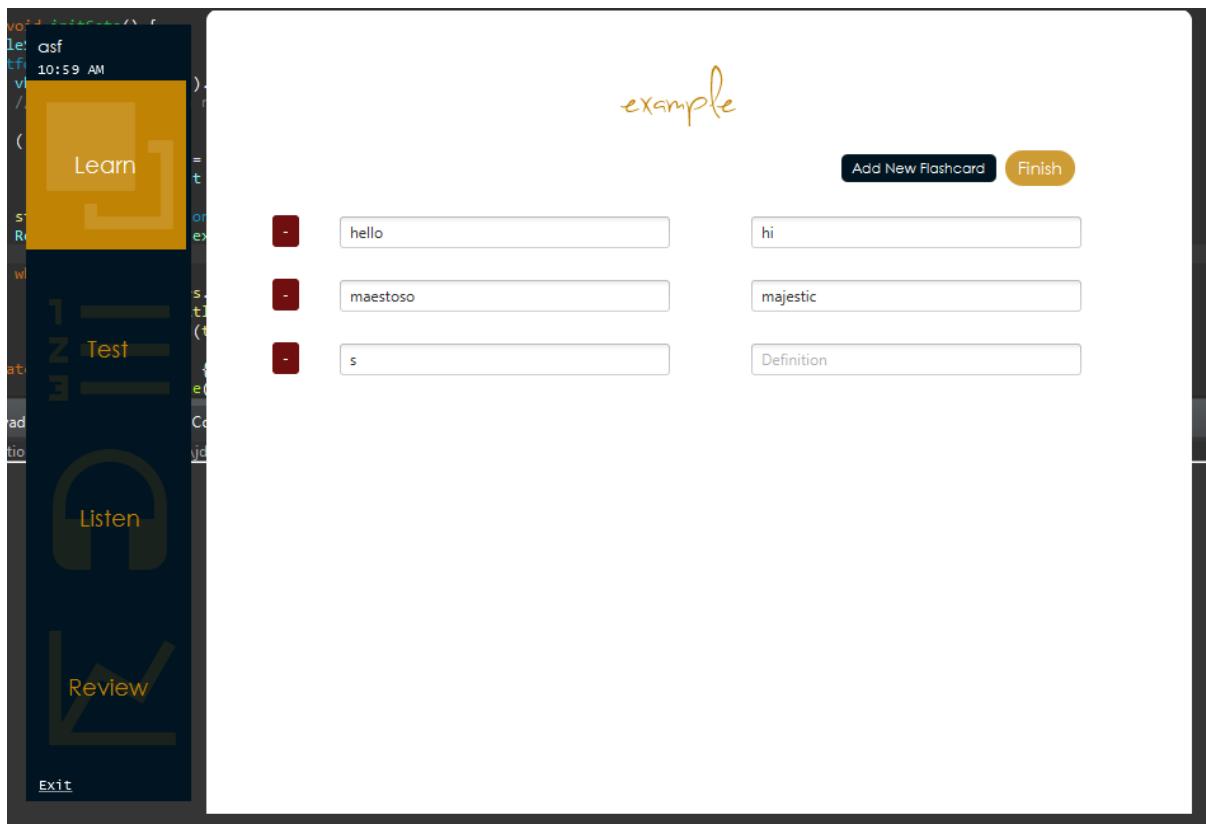
```
1 package model;
2
3 public class Flashcard {
4     private String foreignTerm;
5     private String definition;
6     public Flashcard(String foreignTerm, String definition) {
7         this.foreignTerm = foreignTerm;
8         this.definition = definition;
9     }
10    public String getTerm() {
11        return foreignTerm;
12    }
13    public String getDefinition() {
14        return definition;
15    }
16    public void setTerm(String foreignTerm) {
17        this.foreignTerm = foreignTerm;
18    }
19    public void setDefinition(String definition) {
20        this.definition = definition;
21    }
22 }
```

Validation Test

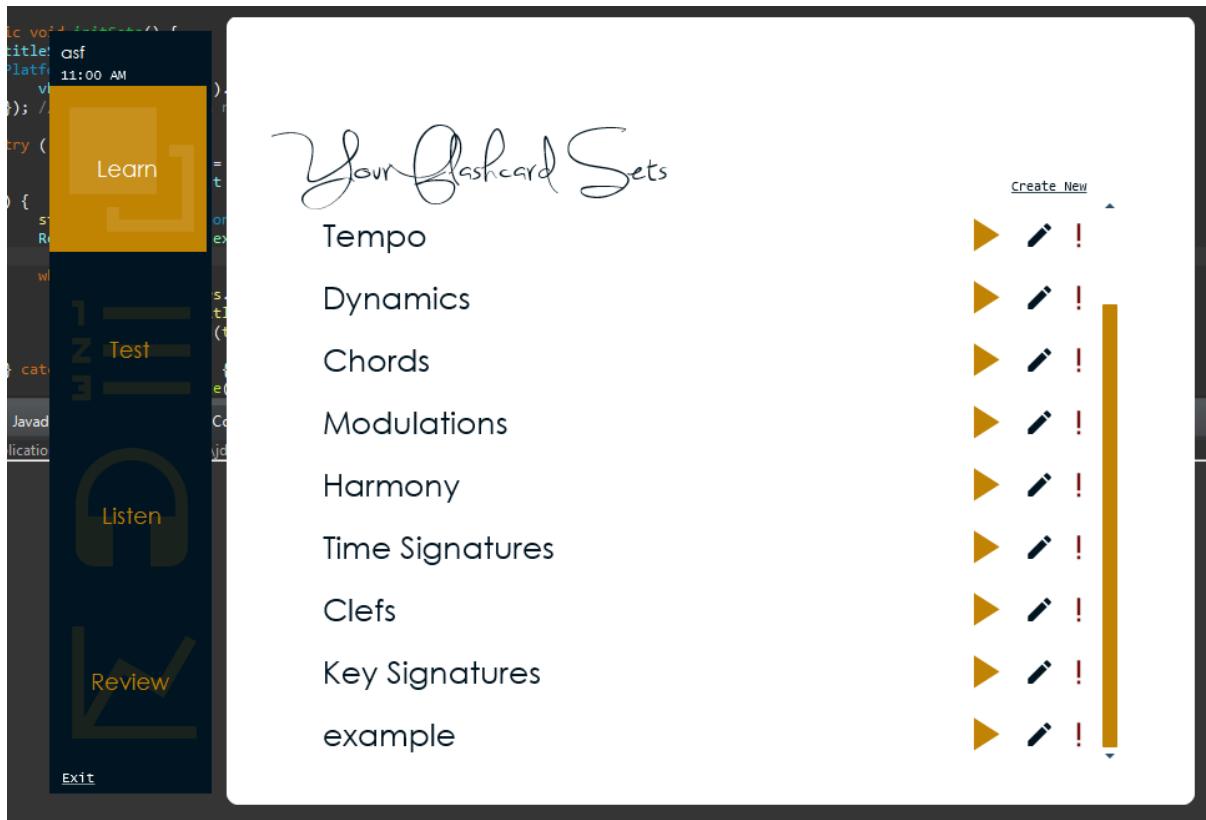
To validate this key element of our system, I did a run through of the Learn section.

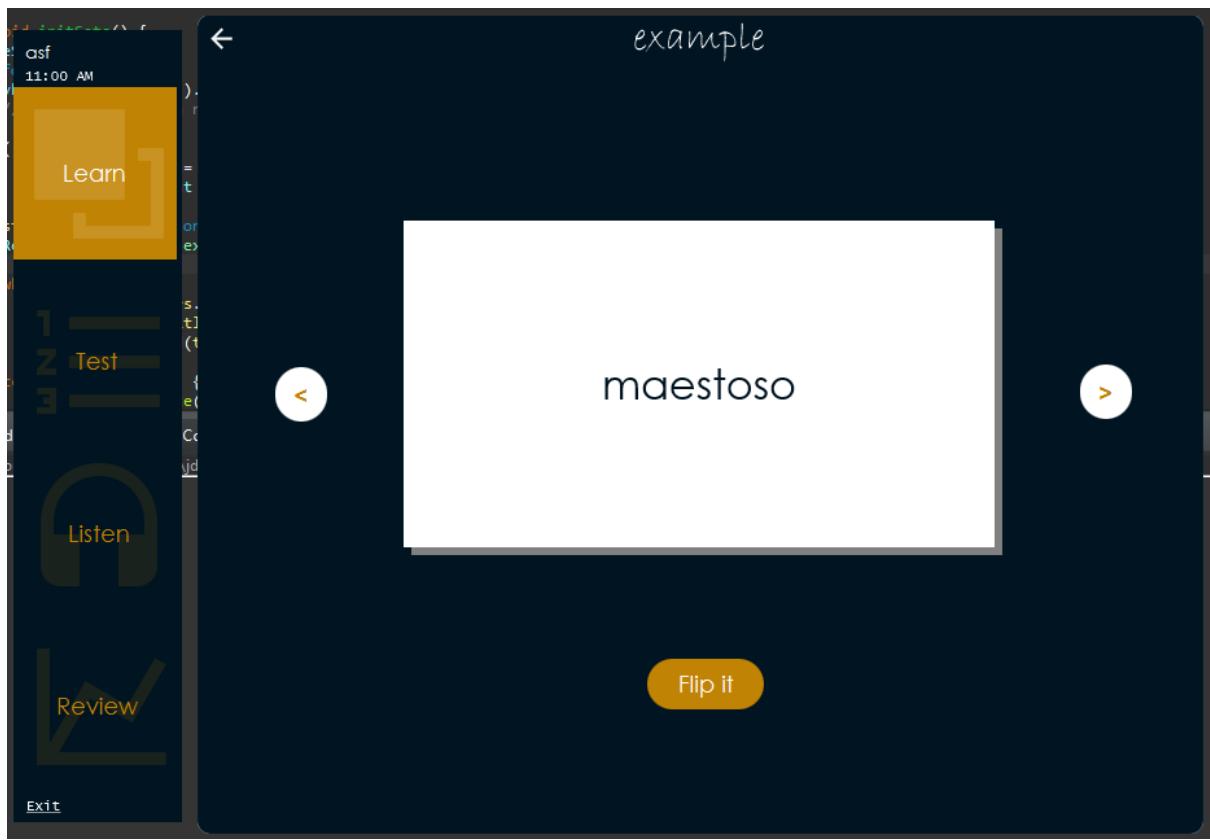
An input of zero length is not accepted and simply returns to the main learn screen but an input of ‘example’ ...



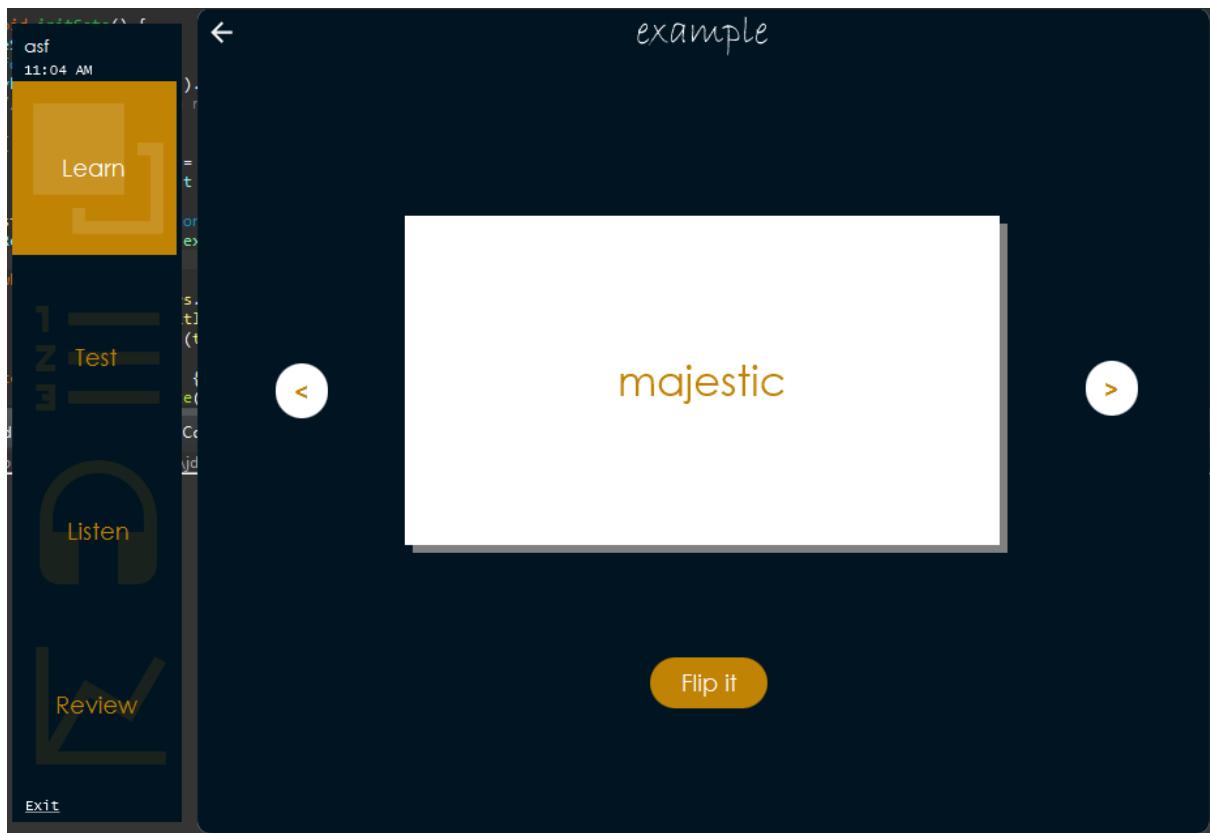


The transition to the new-set screen works well.

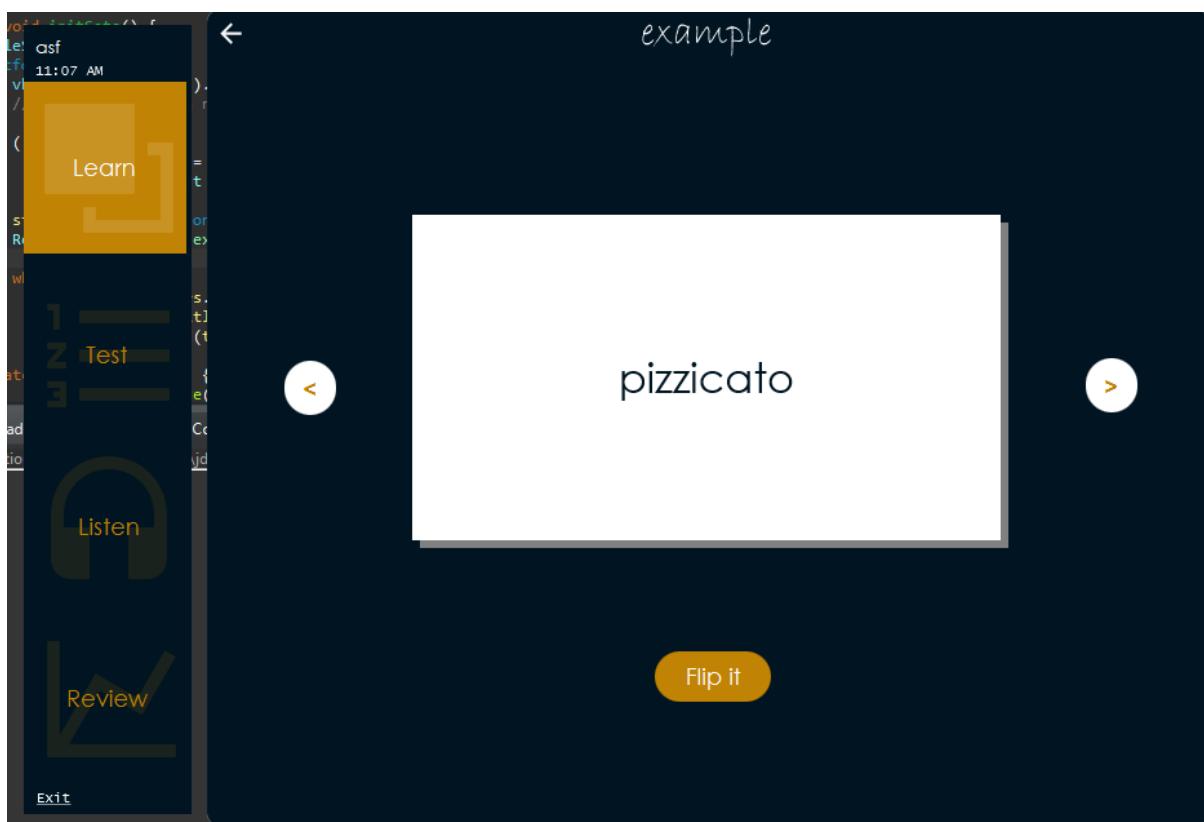
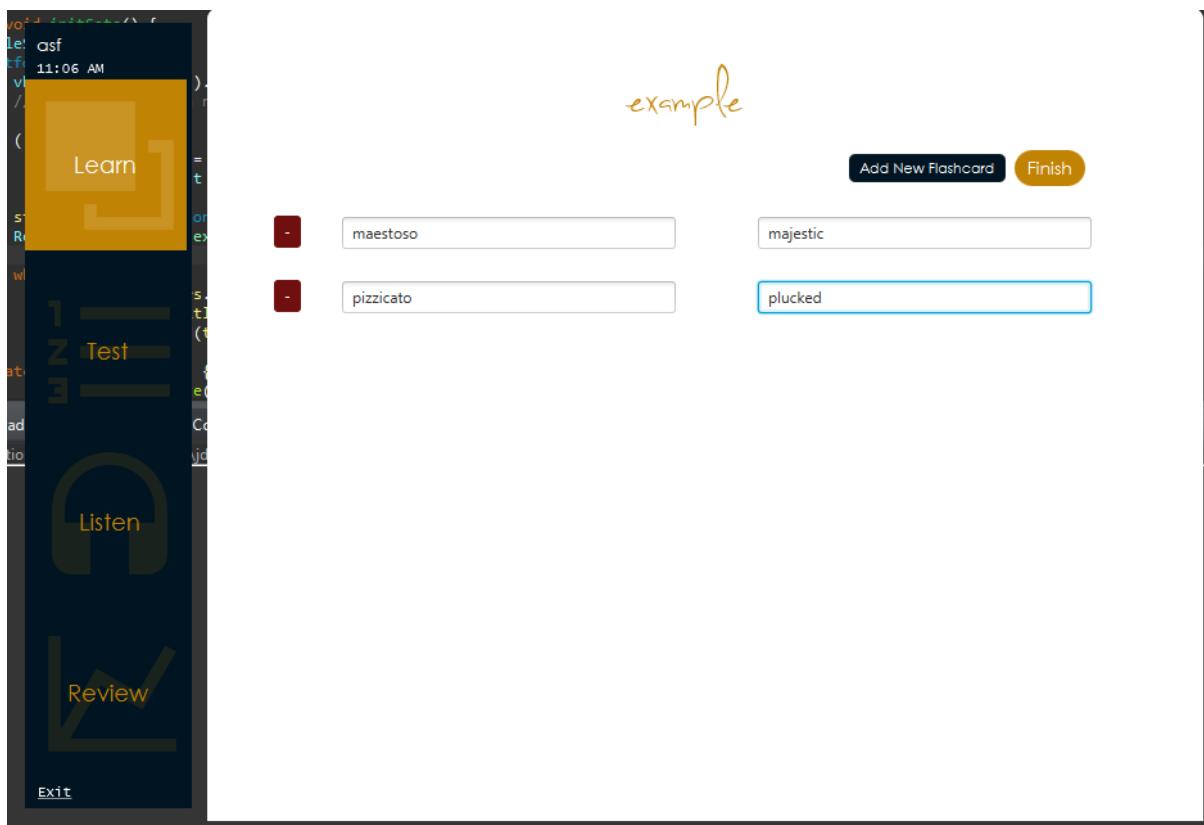




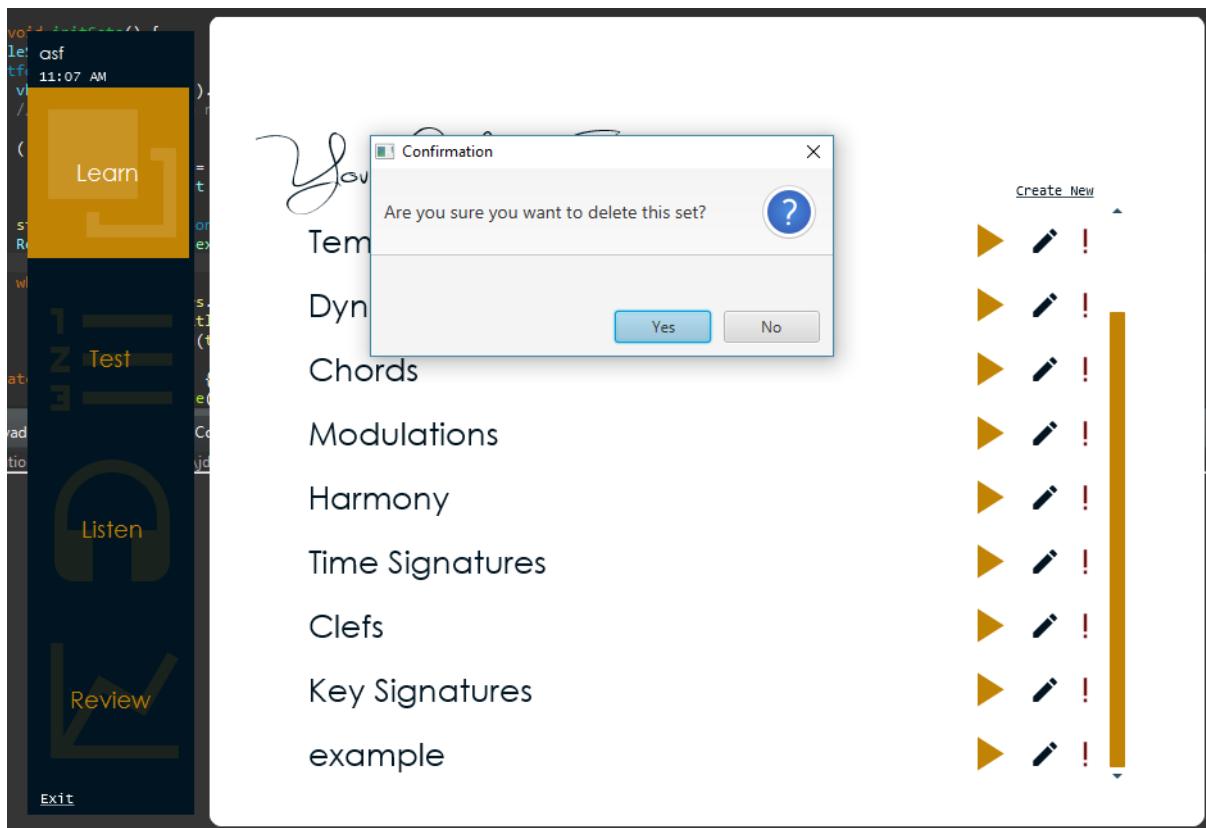
And our flashcards is effective too.



Now trying to edit this set by changing the ‘hello’ ‘hi’ flashcard.



It is correctly displayed



We can then delete this set, No or the close cross does nothing, but yes deletes the set.

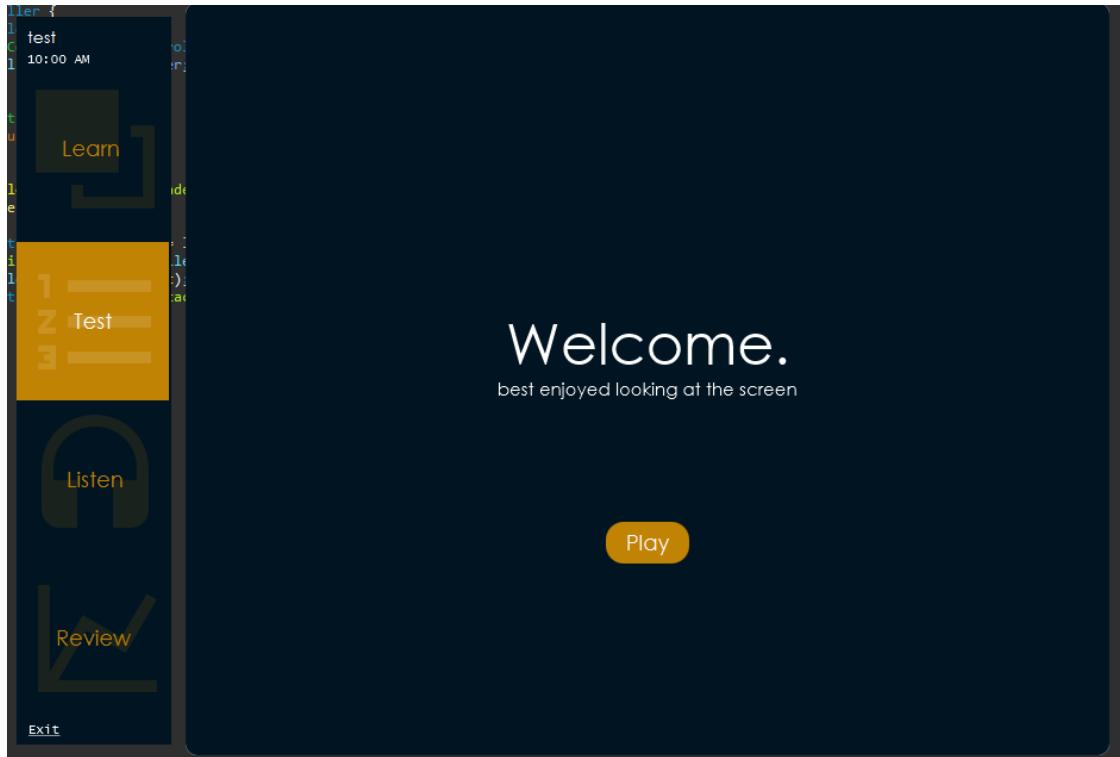


Overall, the Learn section has been successful.

TestController.java

We now have the testing feature. This gives a chance for users to test their skills and highlight which areas they need to improve on for next time.

This controller controls the welcome screen for this activity, test.fxml.



All that needs to be handled here is the play button so a quite compact class will only be required.

Code for TestController.java:

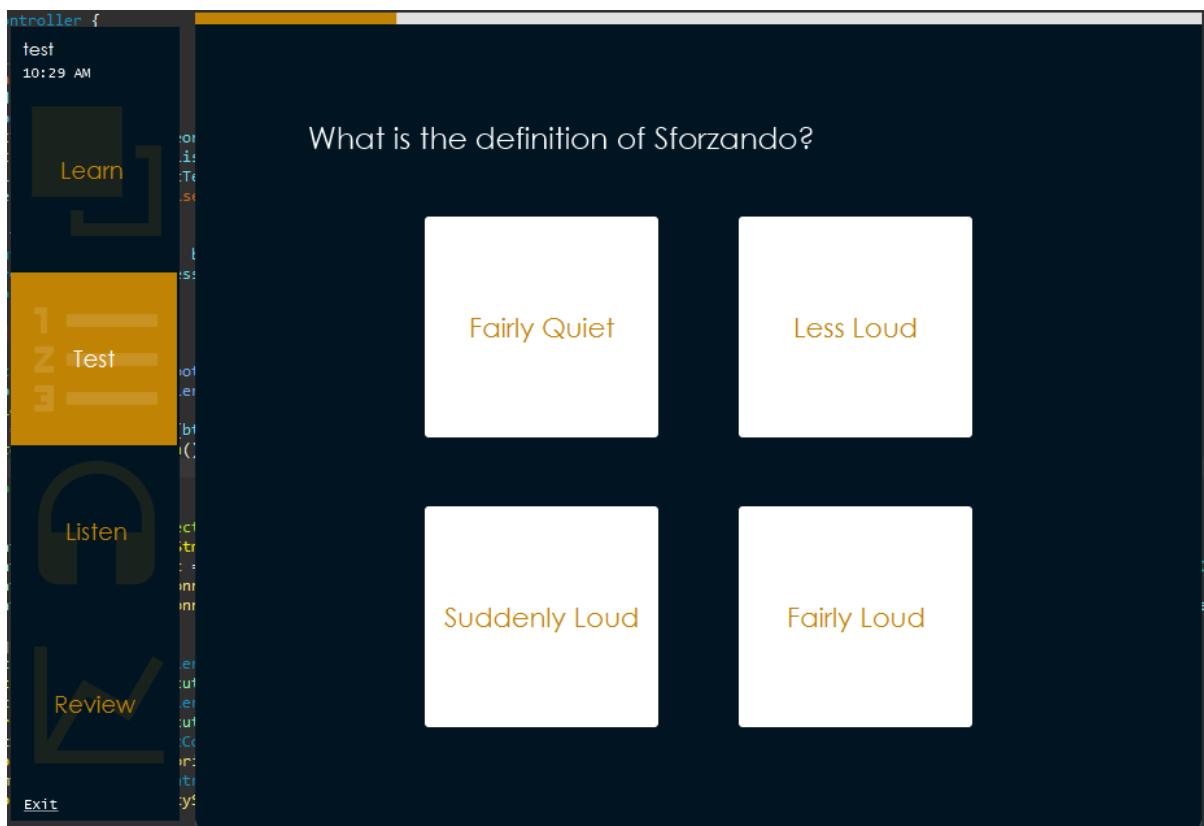
```
1 package controller;
2
3 import java.io.IOException;
4
5 import javafx.event.ActionEvent;
6 import javafx.fxml.FXML;
7 import javafx.fxml.FXMLLoader;
8 import javafx.scene.Parent;
9
10 public class TestController {
11     private RootController rootController;
12     public void setRootController(RootController rootController) {
13         this.rootController = rootController;
14     }
15     @FXML
16     public void playTest(ActionEvent e) {
17         Parent root = null;
18         try {
19             FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/test_quiz.fxml"));
20             root = loader.load();
21             TestQuizController controller = loader.getController();
22             controller.initData(rootController);
23             rootController.setActivity(root);
24         } catch(IOException ex) {ex.printStackTrace();}
25     }
26 }
27 }
```

TestQuizController.java

First of all, we will add another field called K_Coefficient to the table Theory (default value 0 since that means the user neither has got it wrong nor got it right), to make it possible to adapt tests based on the user's input of each musical theory term.

Field Name	Data Type	Description (Optional)
d	AutoNumber	Unique ID for each flashcard
Term	Short Text	Foreign term to be defined
Definition	Short Text	Definition of respective foreign term
Grade	Number	Grade difficulty of term
Title	Short Text	The title (sub-category) which the term relates to.
K_Coefficient	Number	Special number which identifies how K (Okay) someone is at the specific term - hence the name

The quiz part of this program will be controlled by this controller and displayed by test_quiz.fxml.



When we first start the quiz, we should initialise the initial variables such as the rootController to later change the activity, an array containing the buttons so it's easier to refer to them later and iterate through each one, and then get and set the theory questions.

```
public void initData(RootController rootController) {
    this.rootController = rootController;
    answerButtons = new JFXButton[] {btnOne, btnTwo, btnThree, btnFour};

    //collect questions from database
    getTheoryQuestions();
    Platform.runLater(()->loadQuestion());
}
```

To get the theory questions, in the problem decomposition section, we identified that we require random questions and another set of random answers. Also, that if possible, we should focus on questions which were answered wrongly. So within the `getTheoryQuestions()`, we use four separate queries: one to identify the number of high priority terms (small `K_Coefficient`); another to get those high priority terms, and order them ascending so the first one is the highest priority and so forth; another to get the total number of terms; lastly to get those number of terms. We need to get the number of terms before retrieving them so that we can select a random row between zero and that number. Also, the terms will each have equal or below grade than of the student's so the test is relevant.

```
try {
    Connection conn = dao.getConnection();
    PreparedStatement priorityCntStmt = conn.prepareStatement("SELECT COUNT(*) FROM Theory WHERE Grade <= ? AND K_Coefficient < 0");
    PreparedStatement priorityStmt = conn.prepareStatement("SELECT Term, Definition FROM Theory WHERE Grade <= ? AND K_Coefficient < 0 ORDER BY K_Coefficient ASC");
    PreparedStatement cntStmt = conn.prepareStatement("SELECT COUNT(*) FROM Theory WHERE Grade <= ?");
    PreparedStatement rndStmt = conn.prepareStatement("SELECT Term, Definition FROM Theory WHERE Grade <= ?", ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
}
```

Our last query uses an option called `ResultSet.TYPE_SCROLL_INSENSITIVE` and `ResultSet.CONCUR_READ_ONLY` so that we can have more scroll options to move around the `ResultSet` than just `next()` (next row).

Firstly, we should execute those queries and retrieve the lengths of high priority and total terms.

```
//execute queries
cntStmt.setInt(1, RootController.getGrade());
ResultSet cntRs = cntStmt.executeQuery();
rndStmt.setInt(1, RootController.getGrade());
ResultSet rndRs = rndStmt.executeQuery();
priorityCntStmt.setInt(1, RootController.getGrade());
ResultSet priorityCntRs = priorityCntStmt.executeQuery();
priorityStmt.setInt(1, RootController.getGrade());
ResultSet priorityRs = priorityStmt.executeQuery();

//prioritise the low k_coefficient terms (highly answered wrong)
int priorityLength = 0;
if(priorityCntRs.next()) priorityLength = priorityCntRs.getInt(1); //number of high priority terms

//get length to use for upper bound of random num for filled answers
int length = 0;
if(cntRs.next()) length = cntRs.getInt(1); //upperbound of random
```

Secondly, get the priority terms from the database.

```
//get priority terms
if(priorityLength > 0) {
    for(int i = 0; i < (priorityLength >= 10 ? 10 : priorityLength); i++) {
        String question = "";
        String[] answers = new String[4];
        int answerIndex = rand.nextInt(4);
        if(priorityRs.next()) {
            answers[answerIndex] = priorityRs.getString("Definition");
            question = priorityRs.getString("Term");
        }
    }
}
```

The for loop iterates from zero to the number of high priority terms (`K_Coefficient` lower than zero). If the length is larger or equal to 10, then use 10 since we only have 10 questions in our test, otherwise, use as much high priority terms as you can. Furthermore, we use a question

variable and an array of four answers since we have four options for our multiple choice. We use a random integer to specify which index of the array the answer should be in.

However, we should still fill the rest of the answers array, so we add in a for loop afterwards remembering to skip the index which has the correct answer in.

```
//get priority terms
if(priorityLength > 0) {
    for(int i = 0; i<(priorityLength >=10 ? 10: priorityLength); i++) { //if there are more than 10 priority terms, only take the top 10 highest priority
        String question = "";
        String[] answers = new String[4];
        int answerIndex = rand.nextInt(4);
        if(priorityRs.next()) {
            answers[answerIndex] = priorityRs.getString("Definition");
            question = priorityRs.getString("Term");
        }
        //fill other answers
        for(int j = 0; j<4; j++) { //four answers (3 random 1 already selected high priority one)
            if(j == answerIndex)
                continue;

            int randomRow = rand.nextInt(length);
            for(int k=0;k<=randomRow;k++) { //move pointer to the random row; <= because you want to call next() at least once to go past zeroth row
                rndRs.next();
            }
            answers[j] = rndRs.getString("Definition"); // put random answer into array
            rndRs.beforeFirst(); //back to start
        }
        //add question
        theoryQuestionList.add(new TheoryQuestion(question, answers, answerIndex));
    }
}
```

The last line reading `rndRs.beforeFirst()` uses our `ResultSet.TYPE_SCROLL_INSENSITIVE` and `ResultSet.CONCUR_READ_ONLY` options and scrolls to pointer to before the first row so that we can start from the top for our next random answer. Note that we get our random answers by using a random number between zero and the length (number of terms) and iterate through the random `ResultSet` (`rndRs`) that number of times and get the definition from that row, put the pointer to the start and repeat.

Also, at the end of each question iteration, we add the new `TheoryQuestion` to the list.

After we get the high priority terms, if there is still space for more questions (number of high priority terms is less than 10) then we randomly retrieve other terms using a similar technique as before, but simpler without concerning priority.

```
//get all the rest of the terms and select the randomly
for(int questionNumber = 0; questionNumber < 10 - priorityLength; questionNumber++) { //create (ten - already filled) questions
    int answerIndex = rand.nextInt(4);
    String question = "";
    String[] answers = new String[4];
    //get four random answers + one matching foreign term (question)
    for(int i = 0; i<4; i++ ) { //four random answers
        int randomRow = rand.nextInt(length);
        for(int j=0; j <= randomRow; j++) { //move pointer to the random row; <= because you want to call next() at least once to go past zeroth row
            rndRs.next();
        }
        answers[i] = rndRs.getString("Definition");
        if(answerIndex==i)
            question = rndRs.getString("Term");
        rndRs.beforeFirst(); //reset pointer for next random answer
    }
    //add question
    theoryQuestionList.add(new TheoryQuestion(question, answers, answerIndex));
}
```

Here, we retrieve `10-priorityLength` number of terms which signifies the space left after getting those high priority terms. For each of these questions, we get four random answers where each

time, one answer is correct at a different index – a random answerIndex. Next, we follow the same structure as before and add each TheoryQuestion to the list.

In the invoked initData() method, we also load the question from there since we need to load the initial, first question. We will need to use a method called loadQuestion() to retrieve the first question in the theoryQuestionList and set the question text and answer button text.

```
public void loadQuestion() {
    TheoryQuestion theoryQuestion = theoryQuestionList.get(questionIndex);

    //question
    txtQuestion.setText(questionIndex % 2 == 0?
        "What is the definition of " + theoryQuestion.getQuestion() + "?":
        "What does " + theoryQuestion.getQuestion() + " mean?" );

    //answers
    for(int i = 0; i < 4; i++) {
        answerButtons[i].setText(theoryQuestion.getAnswer(i));
    }
}
```

Simply put, I did not want the questions to seem to repetitive so every other question, using modulus 2 (questionIndex % 2 == 0 ?), I changed the structure of the question ever so slightly.

```
public void answerQuestion(int answer) {

    TheoryQuestion currentQuestion = theoryQuestionList.get(questionIndex);
    int correctAnswer = currentQuestion.getAnswerIndex();
    //change colours used for feedback
    answerButtons[answer].setStyle("-fx-background-color: #6f0f0f"); //answered button is red
    answerButtons[answer].setTextFill(Color.WHITE);
    answerButtons[correctAnswer].setStyle("-fx-background-color: #0cb794"); //if your answer was right, it will override the colour to green
    answerButtons[correctAnswer].setTextFill(Color.WHITE);
}

}
```

When answering the question, we display feedback by changing the colour of the buttons so it is simple to understand – using basic colours like red, #6f0f0f, and green, #0cb794.

Also, the correct and wrongly answered terms will be stored for feedback.

```
public void answerQuestion(int answer) {

    TheoryQuestion currentQuestion = theoryQuestionList.get(questionIndex);
    int correctAnswer = currentQuestion.getAnswerIndex();
    //change colours used for feedback
    answerButtons[answer].setStyle("-fx-background-color: #6f0f0f"); //answered button is red
    answerButtons[answer].setTextFill(Color.WHITE);
    answerButtons[correctAnswer].setStyle("-fx-background-color: #0cb794"); //if your answer was right, it will override the colour to green
    answerButtons[correctAnswer].setTextFill(Color.WHITE);

    //if wrong, note down for feedback
    if(answer==correctAnswer)
        correctTerms.add(currentQuestion.getQuestion());
    else
        feedbackList.add(new Feedback(currentQuestion.getQuestion(), currentQuestion.getAnswer(answer), currentQuestion.getAnswer(correctAnswer)));
    //button to move on
    btnOk.setVisible(true);
}

}
```

The answerQuestion() method is called by each of the buttons using a FXML method called giveAnswer() which links to answerQuestion().

```

@FXML
public void giveAnswer(ActionEvent e) {
    JFXButton clickedButton = (JFXButton) e.getSource();

    for(int i=0; i<4; i++) { //check which button is the answered button and then answer the question with its index
        if(clickedButton.equals(answerButtons[i]))
            answerQuestion(i);
    }
}

```

btnOk is the button that is only displayed after you answer the question and allows you to move onto the next question; if the question you are on is the last one, then we can go to the results screen.



```

@FXML
public void nextQuestion(ActionEvent e) {
    if(++questionIndex==10) {
        loadResults();
    } else {
        prbProgress.setProgress((questionIndex)/10.0); //increase progress
        btnOk.setVisible(false); //hide ok

        //make all answer boxes white again with orange text
        for(JFXButton btn: answerButtons) {
            btn.setTextFill(Color.web("#c18303"));
            btn.setStyle("-fx-background-color: #ffffff");
        }

        loadQuestion(); //load new question with incremented questionIndex
    }
}

```

By loading the next question, all we need to do is increment the progress bar, reset some variables and the colours of the buttons. Note that in the first *if-statement* we have incremented questionIndex by one (`++questionIndex`) we used `++questionIndex` rather than `questionIndex++` since we are comparing the updated questionIndex rather than the old one, if we were comparing questionIndex to 10 then incrementing it, we would be having 11 questions, from 0 to 10 inclusive. This means that when we call `loadQuestion()` at the end, we load the next question.

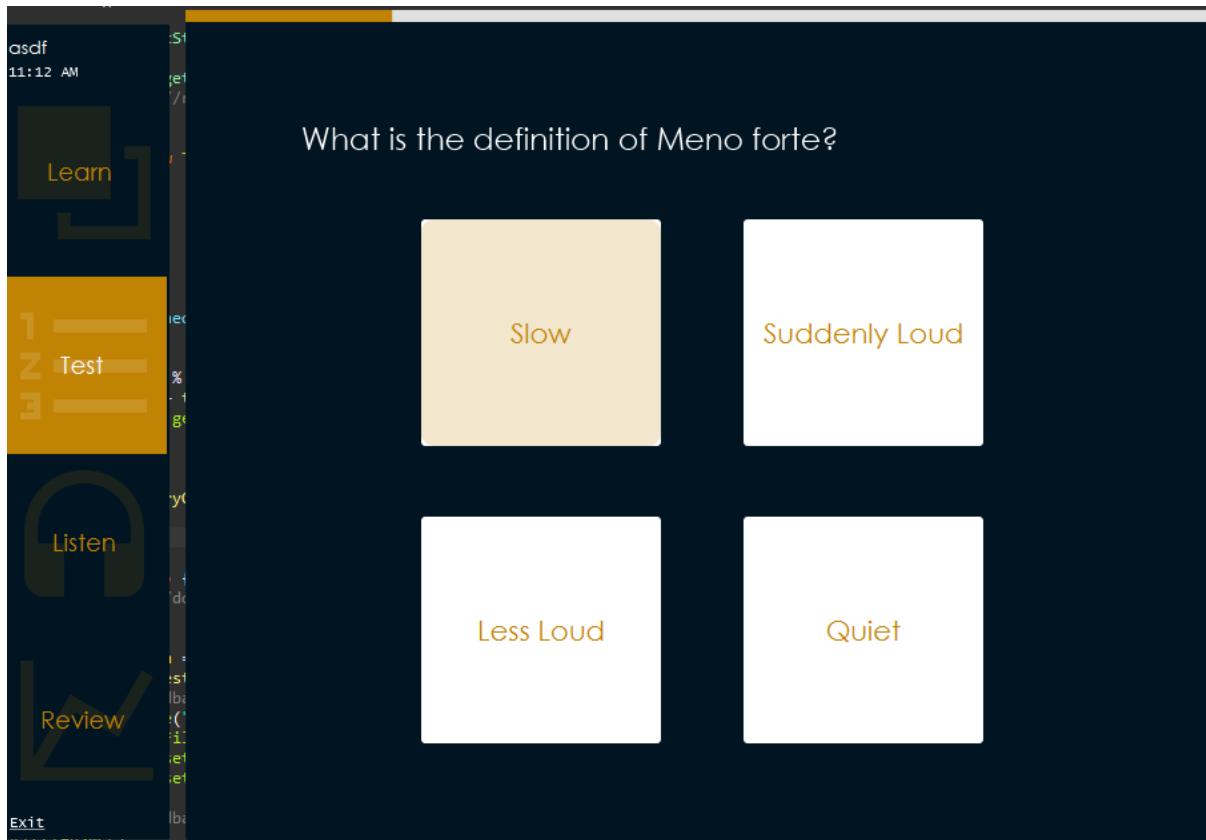
Leaving the `loadResults()` method blank, we can test the quiz up to the last question.

```

public void loadResults() {
}

```

However, when testing it, we stumbled across a few problems, firstly, sometimes a random button was highlighted as if the quiz was suggesting the correct answer.



This was not intended so needed immediate attention.

After thought, this highlight is also known as focus, similar to the flashcards situation. So, we can call `requestFocus()` onto a node which has no event to be fired. At the end of `loadQuestion()`, where it is called very often, we can `requestFocus()` to something that does not fire an event.

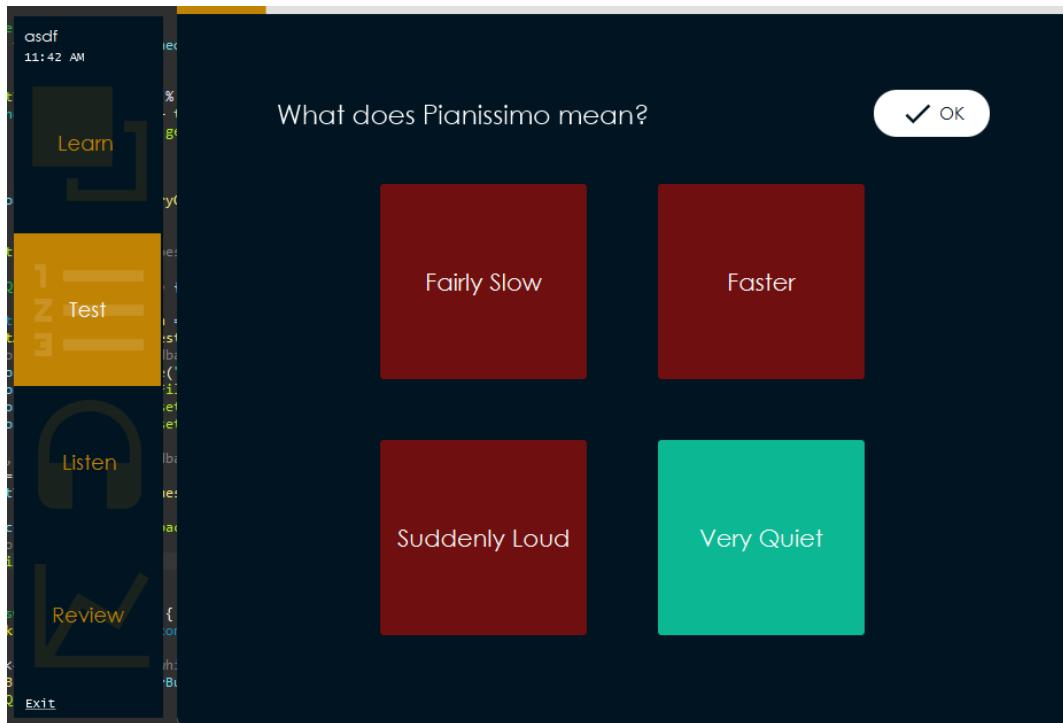
```
public void loadQuestion() {
    TheoryQuestion theoryQuestion = theoryQuestionList.get(questionIndex);

    //question
    txtQuestion.setText(questionIndex % 2 == 0?
        "What is the definition of " + theoryQuestion.getQuestion() + "?":
        "What does " + theoryQuestion.getQuestion() + " mean?" );

    //answers
    for(int i = 0; i < 4; i++) {
        answerButtons[i].setText(theoryQuestion.getAnswer(i));
    }

    apnQuiz.requestFocus(); //so it doesn't highlight a random button
}
```

Also, the when randomly clicking every button on the page, all buttons could change colour.



By locating where the colours of the JFXButton's are changed, we narrow the scope to answerQuestion() and giveAnswer() isolate this problem to these methods. Nowhere does these methods state that the answer button inputs should stop being accepted, so the user can endlessly call giveAnswer() and therefore answerQuestion(), changing the colours. To avoid this problem, we wrap the giveAnswer() method with an *if-statement* checking whether the answer has been answered yet or not.

```
@FXML  
public void giveAnswer(ActionEvent e) {  
    if(questionAnswered == false) {  
        questionAnswered = true;  
        JFXButton clickedButton = (JFXButton) e.getSource();  
  
        for(int i=0; i<4; i++) { //check which button is the answered button and then answer the question with its index  
            if(clickedButton.equals(answerButtons[i]))  
                answerQuestion(i);  
        }  
    }  
}
```

Also, remembering to change questionAnswered back to false upon loading a new question

```

@FXML
public void nextQuestion(ActionEvent e) {
    if(++questionIndex==10) {
        loadResults();
    } else {
        questionAnswered = false; //new question, not answered yet
        prbProgress.setProgress((questionIndex)/10.0); //increase progress
        btnOk.setVisible(false); //hide ok

        //make all answer boxes white again with orange text
        for(JFXButton btn: answerButtons) {
            btn.setTextFill(Color.web("#c18303"));
            btn.setStyle("-fx-background-color: #ffffff");
        }
    }

    loadQuestion(); //load new question with incremented questionIndex
}

}

```

To load the result page, we will simply load the `test_results.fxml` UI and give it a controller class handle its actions, giving it the correct and wrongly answered terms.

```

public void loadResults() {
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/test_results.fxml"));
    try {
        root = loader.load();
    } catch(IOException ex) {ex.printStackTrace();}
    TestResultsController controller = loader.getController();
    controller.initData(rootController, feedbackList, 10-feedbackList.size(), correctTerms);
    rootController.setActivity(root);

    root.requestFocus(); //so doesn't randomly highlight a button
}

```

Code for `TestQuizController.java`:

```

1 package controller;
2
3 import java.io.IOException;
4 import java.sql.Connection;
5 import java.sql.PreparedStatement;
6 import java.sql.ResultSet;
7 import java.sql.SQLException;
8 import java.util.LinkedList;
9 import java.util.Random;
10
11 import com.jfoenix.controls.JFXButton;
12 import com.jfoenix.controls.JFXProgressBar;
13
14 import javafx.application.Platform;
15 import javafx.collections.FXCollections;
16 import javafx.collections.ObservableList;
17 import javafx.event.ActionEvent;
18 import javafx.fxml.FXML;
19 import javafx.fxml.FXMLLoader;
20 import javafx.scene.Parent;
21 import javafx.scene.layout.AnchorPane;
22 import javafx.scene.paint.Color;
23 import javafx.scene.text.Text;
24 import model.Feedback;
25 import model.TheoryQuestion;
26 import other.DAO;
27
28 public class TestQuizController {

```

```

29
30     private RootController rootController;
31     private DAO dao = new DAO();
32     private Random rand = new Random();
33     private int questionIndex = 0;
34     private LinkedList<TheoryQuestion> theoryQuestionList = new LinkedList<>();
35     private LinkedList<Feedback> feedbackList = new LinkedList<>();
36     private ObservableList<String> correctTerms = FXCollections.observableArrayList();
37     private boolean questionAnswered = false;
38
39     @FXML private Text txtQuestion;
40     @FXML private JFXButton btnOk, btnOne, btnTwo, btnThree, btnFour;
41     @FXML private JFXProgressBar prbProgress;
42     @FXML private AnchorPane apnQuiz;
43
44     private JFXButton[] answerButtons;
45
46     public void initData(RootController rootController) {
47         this.rootController = rootController;
48         answerButtons = new JFXButton[] {btnOne, btnTwo, btnThree, btnFour};
49
50         //collect questions from database
51         getTheoryQuestions();
52         Platform.runLater(()->loadQuestion());
53     }
54     public void getTheoryQuestions() {
55         try {
56             Connection conn = dao.getConnection();
57             PreparedStatement priorityCntrStmt = conn.prepareStatement("SELECT COUNT(*) FROM Theory WHERE Grade <= ? AND K_Coefficient < 0");
58             PreparedStatement priorityStmt = conn.prepareStatement("SELECT Term, Definition FROM Theory WHERE Grade <= ? AND K_Coefficient < 0 ORDER BY K_Coefficient ASC");
59             PreparedStatement cntStmt = conn.prepareStatement("SELECT COUNT(*) FROM Theory WHERE Grade <= ?");
60             PreparedStatement rndStmt = conn.prepareStatement("SELECT Term, Definition FROM Theory WHERE Grade <= ?", ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
61         } {
62             //execute queries
63             cntStmt.setInt(1, RootController.getGrade());
64             ResultSet cntRs = cntStmt.executeQuery();
65             rndStmt.setInt(1, RootController.getGrade());
66             ResultSet rndRs = rndStmt.executeQuery();
67             priorityCntrStmt.setInt(1, RootController.getGrade());
68             ResultSet priorityCntrRs = priorityCntrStmt.executeQuery();
69             priorityStmt.setInt(1, RootController.getGrade());
70             ResultSet priorityRs = priorityStmt.executeQuery();
71
72             //prioritise the low k_coefficient terms (highly answered wrong)
73             int priorityLength = 0;
74             if(priorityCntrRs.next())priorityLength = priorityCntrRs.getInt(1); //number of high priority terms
75
76             //get length to use for upper bound of random num for filled answers
77             int length = 0;
78             if(cntRs.next())length = cntRs.getInt(1); //upperbound of random
79
80             //get priority terms
81             if(priorityLength > 0) {
82                 for(int i = 0; i<(priorityLength >= 10 ? 10 : priorityLength); i++) { //if there are more than 10 priority terms, only take the top 10 highest priority (lowest k_coefficient),
83                     String question = "";
84                     String[] answers = new String[4];
85                     int answerIndex = rand.nextInt(4);
86                     if(priorityRs.next()) {
87                         answers[answerIndex] = priorityRs.getString("Definition");
88                         question = priorityRs.getString("Term");
89                     }
90                     //fill other answers
91                     for(int j = 0; j<4; j++) { //four answers (3 random 1 already selected high priority one)
92                         if(j == answerIndex)
93                             continue;
94
95                         int randomRow = rand.nextInt(length);
96                         for(int k=0;k<=randomRow;k++) { //move pointer to the random row; <= because you want to call next() at least once to go past zeroth row
97                             rndRs.next();
98                         }
99                         answers[j] = rndRs.getString("Definition"); // put random answer into array
100                        rndRs.beforeFirst(); //back to start
101                    }
102                    //add question
103                    theoryQuestionList.add(new TheoryQuestion(question, answers, answerIndex));
104                }
105            }
106
107            //get all the rest of the terms and select the randomly
108            for(int questionNumber = 0; questionNumber < 10 - priorityLength; questionNumber++) { //create (ten - already filled) questions
109                int answerIndex = rand.nextInt(4);
110                String question = "";
111                String[] answers = new String[4];
112                //get four random answers + one matching foreign term (question)
113                for(int i = 0; i<4; i++) { //four random answers
114                    int randomRow = rand.nextInt(length);
115                    for(int j=0; j<=randomRow; j++) { //move pointer to the random row; <= because you want to call next() at least once to go past zeroth row
116                        rndRs.next();
117                    }
118                    answers[i] = rndRs.getString("Definition");
119                    if(answerIndex==i)
120                        question = rndRs.getString("Term");
121                    rndRs.beforeFirst(); //reset pointer for next random answer
122                }
123                //add question
124                theoryQuestionList.add(new TheoryQuestion(question, answers, answerIndex));
125            }
126        } catch (SQLException e) {
127            e.printStackTrace();
128        }
129    }
130 }

```

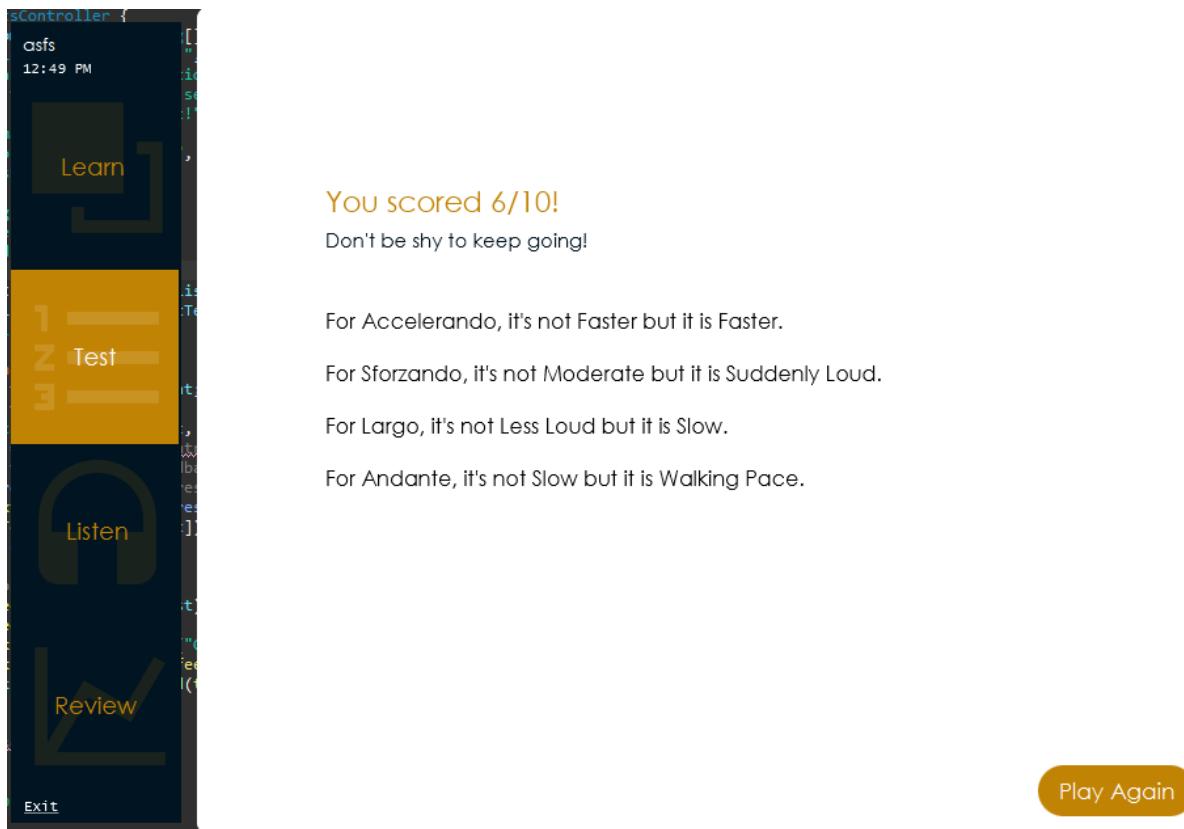
```

131● public void loadQuestion() {
132     TheoryQuestion theoryQuestion = theoryQuestionList.get(questionIndex);
133
134     //question
135     txtQuestion.setText(questionIndex % 2 == 0?
136         "What is the definition of " + theoryQuestion.getQuestion() + "?":
137         "What does " + theoryQuestion.getQuestion() + " mean?" );
138
139     //answers
140     for(int i = 0; i< 4; i++) {
141         answerButtons[i].setText(theoryQuestion.getAnswer(i));
142     }
143
144     apnQuiz.requestFocus(); //so it doesn't highlight a random button
145 }
146● public void answerQuestion(int answer) {
147     TheoryQuestion currentQuestion = theoryQuestionList.get(questionIndex);
148     int correctAnswer = currentQuestion.getAnswerIndex();
149
150     //change colours used for feedback
151     answerButtons[answer].setStyle("-fx-background-color: #6f0f0f"); //answered button is red
152     answerButtons[answer].setTextFill(Color.WHITE);
153     answerButtons[correctAnswer].setStyle("-fx-background-color: #0cb794"); //if your answer was right, it will override the colour to green
154     answerButtons[correctAnswer].setTextFill(Color.WHITE);
155
156     //if wrong, note down for feedback
157     if(answer==correctAnswer)
158         correctTerms.add(currentQuestion.getQuestion());
159     else
160         feedbackList.add(new Feedback(currentQuestion.getQuestion(), currentQuestion.getAnswer(answer), currentQuestion.getAnswer(correctAnswer)));
161     //button to move on
162     btnOk.setVisible(true);
163 }
164● @FXML
public void giveAnswer(ActionEvent e) {
165     if(questionAnswered == false) {
166         questionAnswered = true;
167         JFXButton clickedButton = (JFXButton) e.getSource();
168
169         for(int i=0; i<4; i++) { //check which button is the answered button and then answer the question with its index
170             if(clickedButton.equals(answerButtons[i]))
171                 answerQuestion(i);
172         }
173     }
174 }
175● @FXML
public void nextQuestion(ActionEvent e) {
176     if(++questionIndex==10) {
177         loadResults();
178     } else {
179         prbProgress.setProgress((questionIndex)/10.0); //increase progress
180         btnOk.setVisible(false); //hide ok
181
182         //make all answer boxes white again with orange text
183         for(JFXButton btn: answerButtons) {
184             btn.setTextFill(Color.web("#c18303"));
185             btn.setStyle("-fx-background-color: #ffffff");
186         }
187
188         loadQuestion(); //load new question with incremented questionIndex
189
190     }
191 }
192 }
193
194● public void loadResults() {
195     Parent root = null;
196     FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/test_results.fxml"));
197     try {
198         root = loader.load();
199     } catch(IOException ex) {ex.printStackTrace();}
200     TestResultsController controller = loader.getController();
201     controller.initData(rootController, feedbackList, 10-feedbackList.size(), correctTerms);
202     rootController.setActivity(root);
203
204     root.requestFocus(); //so doesn't randomly highlight a button
205 }
206
207 }

```

TestResultsController.java

A crucial part of testing is identifying improvements and so this page, `test_results.fxml` will summarise the corrections to be made.



Firstly, we will design the table, `Score`, to be used to hold the highscores of the test for each day, this will be helpful for our Review section.

Field Name	Data Type	Description (Optional)
d	AutoNumber	Unique ID to identify each score
ScoreDay	Number	Day of the month (1... 30 etc)
ScoreMonth	Number	Month of the year (1=January, 12=December)
ScoreYear	Number	Year of the score
Highscore	Number	High score of the day

Secondly, we should initialise initial data first including a set of comments which will change depending on your score.

```

private String[] comments = new String[] {
    "Try learning the terms first!",
    "Have a peak at the learn section.",
    "Remember, there is the learn section just for you!",
    "It's fine, just learn from it!",
    "Practice makes perfect",
    "You can do better next time!",
    "Don't be shy to keep going!",
    "Not bad!",
    "Keep going!",
    "Almost there!",
    "Wow, you did it!"
};

private LinkedList<Feedback> feedbackList;
private ObservableList<String> correctTerms;
private RootController rootController;
private int result;
private DAO dao = new DAO();
@FXML private Text txtScore, txtComment;
@FXML private VBox vbxFeedbacks;
public void initData(RootController rc, LinkedList<Feedback> fbl, int result, ObservableList<String> ct) {
    rootController = rc; //set rootcontroller for activity changes
    feedbackList = fbl; //list of feedback
    this.result = result; //take the result of the test
    txtScore.setText("You scored " + result + "/10!");
    txtComment.setText(comments[result]); //a comment depending on the result
    correctTerms = ct;
}

```

Since this method is only called at the beginning of when this class gets an object initialised, we can load up the feedback and store them in the database. Feedback only needs to be displayed using a Text node so customising its font and text is enough to display the important feedback.

```

public void initData(RootController rc, LinkedList<Feedback> fbl, int result, ObservableList<String> ct) {
    rootController = rc; //set rootcontroller for activity changes
    feedbackList = fbl; //list of feedback
    this.result = result; //take the result of the test
    txtScore.setText("You scored " + result + "/10!");
    txtComment.setText(comments[result]); //a comment depending on the result
    correctTerms = ct;

    //add the feedback to the vbox
    for(Feedback feedback: feedbackList) {
        Text txtFeedback = new Text();
        txtFeedback.setFont(Font.font("Century Gothic", FontPosture.REGULAR, 16));
        txtFeedback.setText("For " + feedback.getQuestion() + ", it's not " + feedback.getWrong() + " but it is " + feedback.getCorrect() + "\n");
        vbxFeedbacks.getChildren().add(txtFeedback);
    }

    // store into sql
    updateDB();
}

```

Within the updateDB() method we need to update the new K_Coefficient's of the terms of each question. This is so that we can adapt the next test. We will require a multitude of queries for this method, including those required for storing the score in advance for the review section.

```

public void updateDB() {
    try{
        Connection conn = dao.getConnection();
        PreparedStatement checkKStmt = conn.prepareStatement("SELECT K_Coefficient FROM Theory WHERE Term = ?"); //check the current k coefficient
        PreparedStatement higherKStmt = conn.prepareStatement("UPDATE Theory SET K_Coefficient = K_Coefficient + 1 WHERE Term = ?"); //for correct answers
        PreparedStatement lowerKStmt = conn.prepareStatement("UPDATE Theory SET K_Coefficient = K_Coefficient - 1 WHERE Term = ?"); //for wrong answers

        //record score for review section
        PreparedStatement oldScoreStmt = conn.prepareStatement("SELECT Highscore FROM Score WHERE ScoreDay = ? AND ScoreMonth = ? AND ScoreYear = ?");
        PreparedStatement insScoreStmt = conn.prepareStatement("INSERT INTO Score (ScoreDay, ScoreMonth, ScoreYear, Highscore) VALUES (?, ?, ?, ?)");
        PreparedStatement updScoreStmt = conn.prepareStatement("UPDATE Score SET Highscore = ? WHERE ScoreDay = ? AND ScoreMonth = ? AND ScoreYear = ?");

    } {

```

For each correctly answered term, we will increment the K_Coefficient by 1 unless already 1 (so that the term does not get forgotten over time).

```

//changes to K_Coefficient to adapt for next test
for(String term: correctTerms) {
    checkKStmt.setString(1, term);
    ResultSet checkRs = checkKStmt.executeQuery();
    int K_Coefficient = 0;
    if(checkRs.next())K_Coefficient = checkRs.getInt("K_Coefficient");

    if(K_Coefficient < 1) {
        higherKStmt.setString(1, term);
        higherKStmt.addBatch();
    }
}
if(correctTerms.size()>0)
    higherKStmt.executeBatch(); //larger than zero otherwise batch update won't work

```

Similar to LearnUpdateController, we only will execute the batch if there are items inside the batch query so that there will be no errors when executing it.

Each wrongly answered term will have a decremented K_Coefficient by 1 (no matter its original score, this is so that extremely problematic terms will be focused on even more).

```

for(Feedback wrongTerms: feedbackList) { //all terms answered wrongly will be noted for next term (by lowering the K_Coefficient)
    lowerKStmt.setString(1, wrongTerms.getQuestion());
    lowerKStmt.addBatch();
}
if(feedbackList.size()>0)lowerKStmt.executeBatch();

```

Before we move onto the queries for our review section, we need a getDate() method which will identify the values to place into our ? parameters. We can use a Calendar to do so.

```

public int[] getDate() {
    Calendar now = Calendar.getInstance();
    int day = now.get(Calendar.DAY_OF_MONTH);
    int month = now.get(Calendar.MONTH) + 1; //zero indexed, +1 to make january 1, feb 2, ...
    int year = now.get(Calendar.YEAR);

    return new int[] {day, month, year};
}

```

We only need to store the high score (or hi-score) of each day so if the old score is higher than the currently achieved, the program will not record the currently achieved score. Additionally, if a high score does not even exist in the first place, meaning that the test has only been taken the first time of this day, then we need to insert a new row rather than update it.

```

//add highscore into db
//check old score if it has improved
int[] date = getDate();
oldScoreStmt.setInt(1, date[0]);
oldScoreStmt.setInt(2, date[1]);
oldScoreStmt.setInt(3, date[2]);
ResultSet oldScoreRs = oldScoreStmt.executeQuery();
int hiscore = -1;
if(oldScoreRs.next())hiscore = oldScoreRs.getInt("Highscore"); //if a score exists for this date, take it for current the hiscore
if(hiscore==1) { //score doesn't already exist
    insScoreStmt.setInt(1, date[0]);
    insScoreStmt.setInt(2, date[1]);
    insScoreStmt.setInt(3, date[2]);
    insScoreStmt.setInt(4, result);
    insScoreStmt.executeUpdate();
} else if(result > hiscore) { //if score exists and actually has improved
    updScoreStmt.setInt(1, result);
    updScoreStmt.setInt(2, date[0]);
    updScoreStmt.setInt(3, date[1]);
    updScoreStmt.setInt(4, date[2]);
    updScoreStmt.executeUpdate();
}

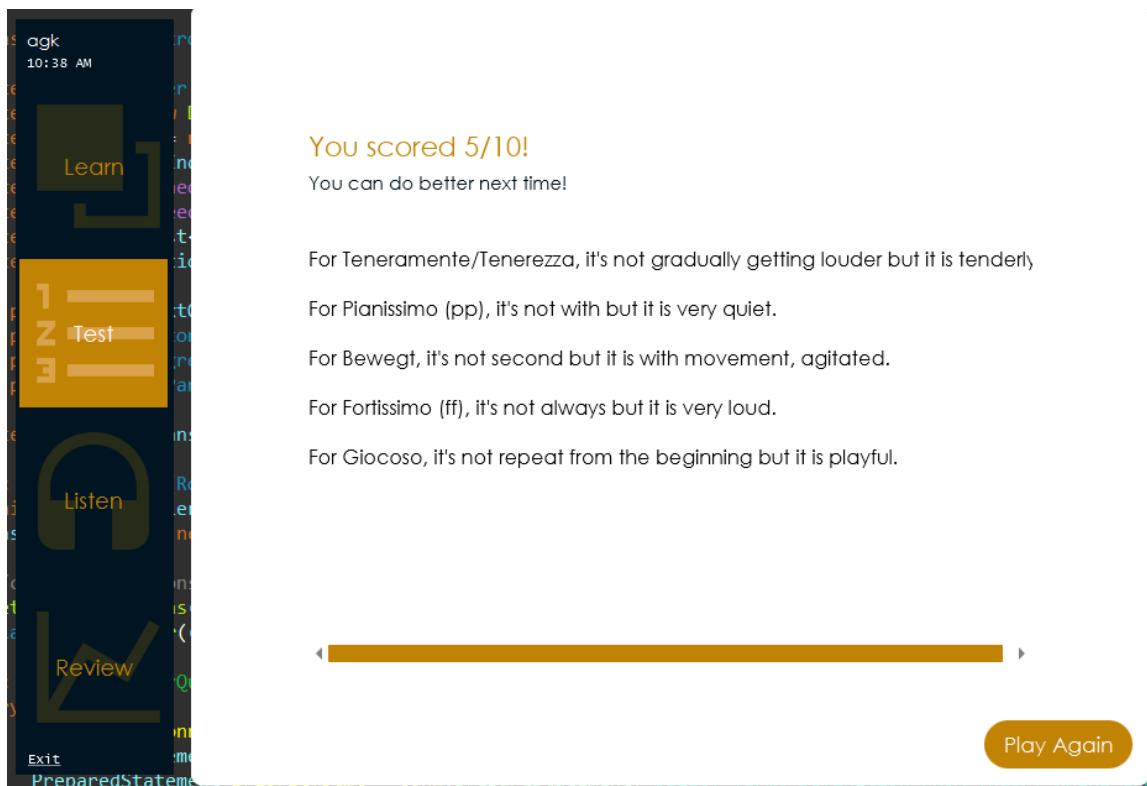
```

Now, of course, after seeing one's feedback, they have a stronger urge to improve. Thinking about this, the implementation of a play again button will allow the user to fulfil their learning desires.

```
@FXML public void playAgain(ActionEvent e) {  
    //go back to starting screen  
    Parent root = null;  
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/test.fxml"));  
    try {  
        root = loader.load();  
    } catch(IOException ioe) {ioe.printStackTrace();}  
    TestController controller = loader.getController();  
    controller.setRootController(rootController);  
    rootController.setActivity(root);  
  
    root.requestFocus(); //so doesn't randomly highlight a button
```

Simple loading new activity code above.

Testing if the result was correctly stored, after taking a test,

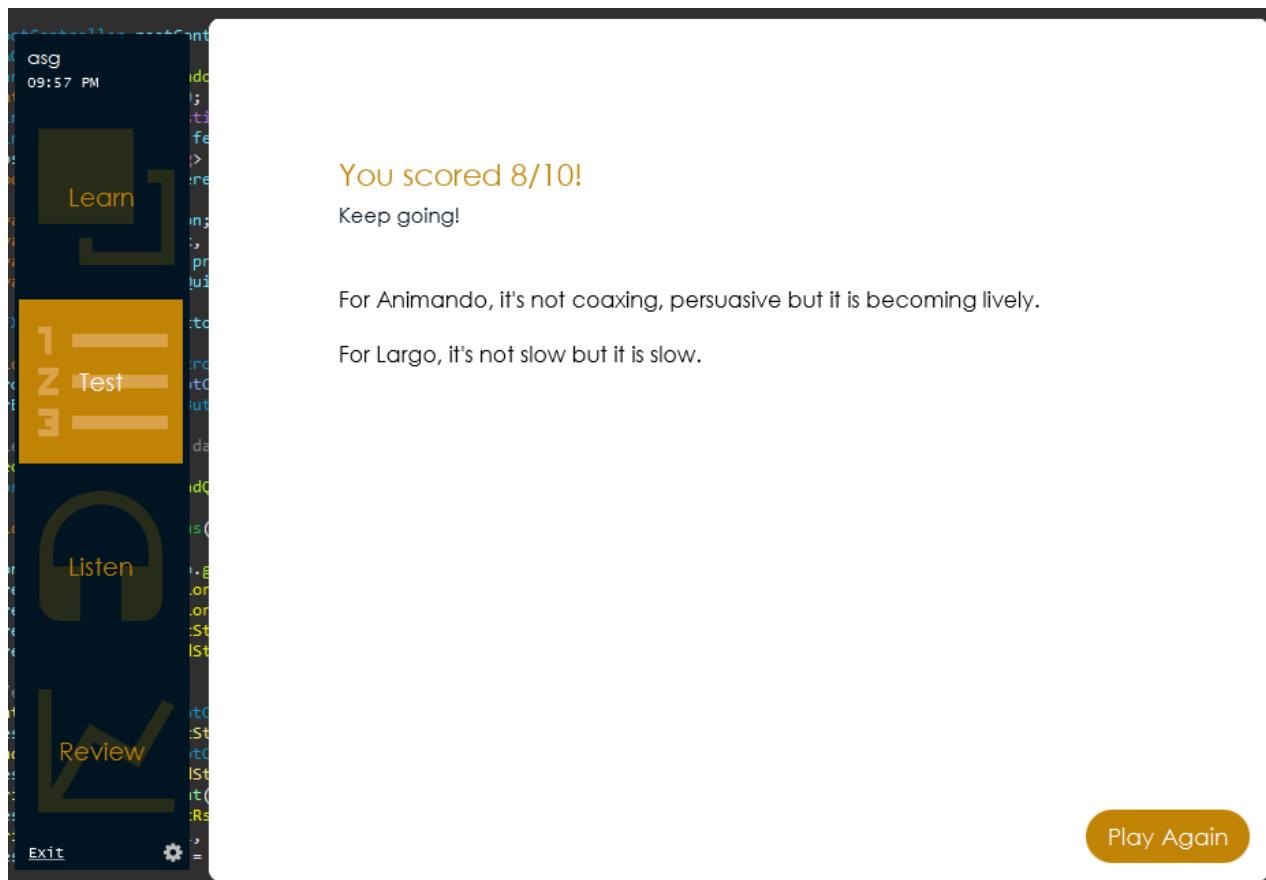


The score was successfully stored inside the database with the K_Coefficient values successfully updated.

Score						
ID	ScoreDay	ScoreMonth	ScoreYear	Highscore	Click to Add	
3	21	11	2018	5	0	
*	(New)	0	0	0	0	

ID	Term	Definition	Grade	Title	K_Coefficient	Click to...
18	Accelerando	gradually getting faster	1	Tempo	1	
19	Adagio	slow	1	Tempo	0	
20	Allegretto	fairly quick	1	Tempo	0	
21	Allegro	quick	1	Tempo	0	
22	Andante	at a walking pace	1	Tempo	0	
23	Cantabile	in a singing style	1	Style	0	
24	Crescendo (cresc.)	gradually getting louder	1	Tempo	0	
25	Da Capo (D.C.)	repeat from the beginning	1	Form	0	
26	Dal Segno (D.S.)	repeat from the sign	1	Form	0	
27	Decrescendo (decresc.)	gradually getting softer	1	Tempo	0	
28	Diminuendo (dim.)	gradually getting softer	1	Tempo	0	
29	Fine	the end	1	Form	0	
30	forte (f)	loud	1	Dynamics	0	
31	Fortissimo (ff)	very loud	1	Dynamics	-1	
32	Legato	smoothly	1	Articulation	0	
33	Lento	slow	1	Tempo	0	
34	Mezzo	half	1	Adverbs	0	
35	Maestoso	majestic	1	Style	0	
36	Mezzo Forte (mf)	moderately loud	1	Dynamics	0	
37	Mezzo Piano (mr)	moderately quiet	1	Dynamics	0	
38	Moderato	moderately	1	Tempo	0	
39	Piano (p)	quiet	1	Dynamics	1	
40	Pianissimo (pp)	very quiet	1	Dynamics	-1	
41	Poco	a little	1	Adverbs	0	
42	Rallentando (ra)	gradually getting slower	1	Tempo	0	
43	Ritardando (rit.)	gradually getting slower	1	Tempo	0	
44	Ritenuto (rit.)	held back	1	Tempo	0	
45	Staccato (stacc.)	detached	1	Articulation	0	
46	Tempo	speed / time (a tempo)	1	Metre	0	

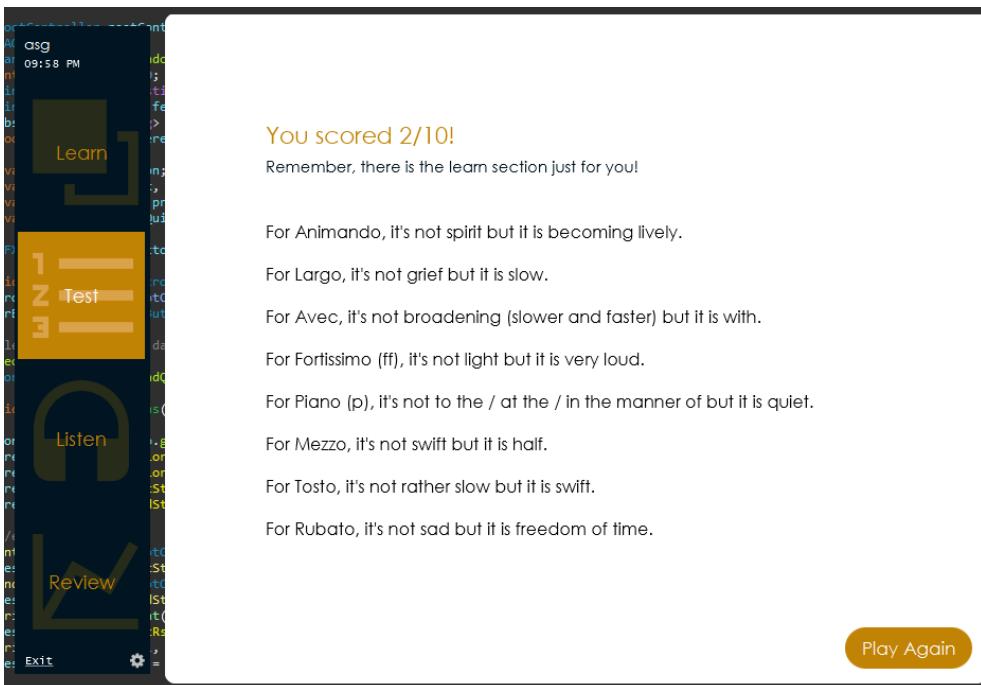
And after getting a higher score,



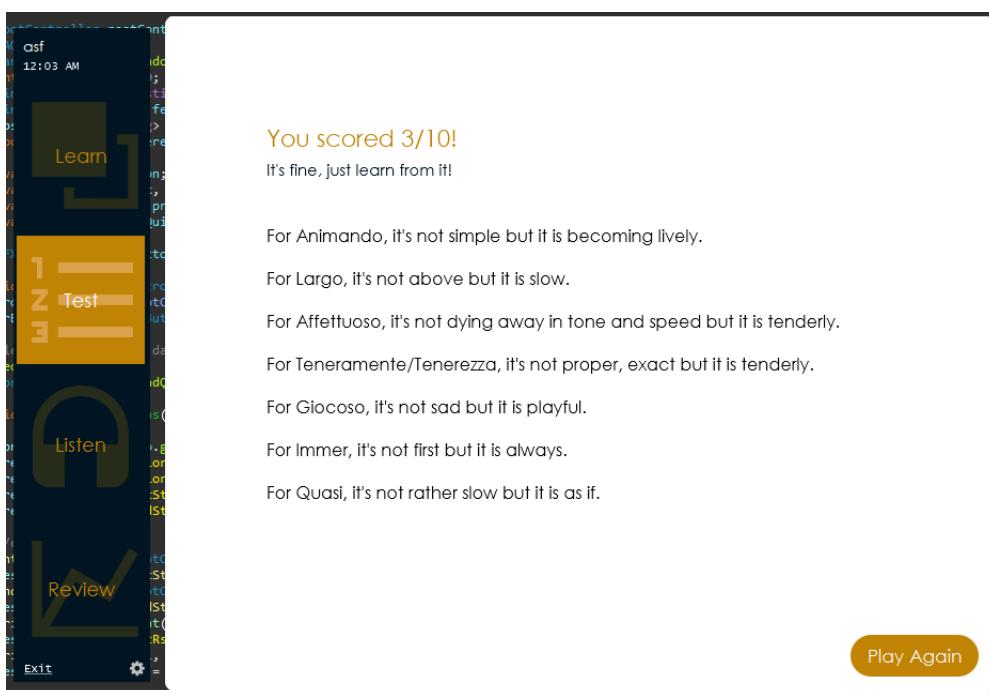
The highscore was successfully updated from a 5 to and 8 in the database, field *Highscore*.

ID	ScoreDay	ScoreMonth	ScoreYear	Highscore	Click to Add
3	21	11	2018	8	
(New)	0	0	0	0	

For a lower score, the highscore was unchanged – as we wanted.



After a later time (the following day), taking the test...



A correct record was created in the table.

Score						
	ID	ScoreDay	ScoreMonth	ScoreYear	Highscore	Click to Add
	3	21	11	2018	8	
	4	22	11	2018	3	
*	(New)	0	0	0	0	

Code for TestResultsController.java:

```
1 package controller;
2
3 import java.io.IOException;
4 import java.sql.Connection;
5 import java.sql.PreparedStatement;
6 import java.sql.ResultSet;
7 import java.sql.SQLException;
8 import java.util.Calendar;
9 import java.util.LinkedList;
10
11 import javafx.collections.ObservableList;
12 import javafx.event.ActionEvent;
13 import javafx.fxml.FXML;
14 import javafx.fxml.FXMLLoader;
15 import javafx.scene.Parent;
16 import javafx.scene.layout.VBox;
17 import javafx.scene.text.Font;
18 import javafx.scene.text.FontPosture;
19 import javafx.scene.text.Text;
20 import model.Feedback;
21 import other.DAO;
22
23 public class TestResultsController {
24     private String[] comments = new String[] {
25         "Try learning the terms first!",
26         "Have a peak at the learn section.",
27         "Remember, there is the learn section just for you!",
28         "It's fine, just learn from it!",
29         "practice makes perfect",
30         "you can do better next time!",
31         "Don't be shy to keep going!",
32         "Not bad!",
33         "Keep going!",
34         "Almost there!",
35         "Wow, you did it!"
36     };
37     private LinkedList<Feedback> feedbackList;
38     private ObservableList<String> correctTerms;
39     private RootController rootController;
40     private int result;
41     private DAO dao = new DAO();
42     @FXML private Text txtScore, txtComment;
43     @FXML private VBox vbxFeedbacks;
44     public void initData(RootController rc, LinkedList<Feedback> fbl, int result, ObservableList<String> ct) {
45         rootController = rc; //set rootcontroller for activity changes
46         feedbackList = fbl; //list of feedback
47         this.result = result; //take the result of the test
48         txtScore.setText("You scored " + result + "/10!");
49         txtComment.setText(comments[result]); //a comment depending on the result
50         correctTerms = ct;
51
52         //add the feedback to the vbox
53         for(Feedback feedback: feedbackList) {
```

```

54     Text txtFeedback = new Text();
55     txtFeedback.setFont(Font.font("Century Gothic", FontPosture.REGULAR, 16));
56     txtFeedback.setText("For " + feedback.getQuestion() + ", it's not " + feedback.getWrong() + " but it is " + feedback.getCorrect() + ".\n");
57     vbxFeedbacks.getChildren().add(txtFeedback);
58 }
59
60 // store into sql
61 updateDB();
62 }
63
64 public void updateDB() {
65     try{
66         Connection conn = dao.getConnection();
67         PreparedStatement checkKStmt = conn.prepareStatement("SELECT K_Coefficient FROM Theory Where Term = ?"); //check the current k coefficient
68         PreparedStatement higherKStmt = conn.prepareStatement("UPDATE Theory SET K_Coefficient = K_Coefficient + 1 WHERE Term = ?"); //for correct answers
69         PreparedStatement lowerKStmt = conn.prepareStatement("UPDATE Theory SET K_Coefficient = K_Coefficient - 1 WHERE Term = ?"); //for wrong answers
70
71         //record score for review section
72         PreparedStatement oldScoreStmt = conn.prepareStatement("SELECT Highscore FROM Score WHERE ScoreDay = ? AND ScoreMonth = ? AND ScoreYear = ?");
73         PreparedStatement insScoreStmt = conn.prepareStatement("INSERT INTO Score (ScoreDay, ScoreMonth, ScoreYear, Highscore) VALUES (?, ?, ?, ?)");
74         PreparedStatement updScoreStmt = conn.prepareStatement("UPDATE Score SET Highscore = ? WHERE ScoreDay = ? AND ScoreMonth = ? AND ScoreYear = ?");
75     }{
76         conn.setAutoCommit(false);
77         //changes to K_Coefficient to adapt for next test
78         for(String term: correctTerms) {
79             checkKStmt.setString(1, term);
80             ResultSet checkRs = checkKStmt.executeQuery();
81             int K_Coefficient = 0;
82             if(checkRs.next())K_Coefficient = checkRs.getInt("K_Coefficient");
83
84             if(K_Coefficient < 1) {
85                 higherKStmt.setString(1, term);
86                 higherKStmt.addBatch();
87             }
88         }
89         if(correctTerms.size()>0)
90             higherKStmt.executeBatch(); //larger than zero otherwise batch update won't work
91
92         for([feedback wrongTerms: feedbackList) { //all terms answered wrongly will be noted for next term (by lowering the K_Coefficient)
93             lowerKStmt.setString(1, wrongTerms.getQuestion());
94             lowerKStmt.addBatch();
95         }
96         if(feedbackList.size()>0)lowerKStmt.executeBatch();
97
98         //add highscores into db
99         //check old score if it has improved
100        int[] date = getDate();
101        oldScoreStmt.setInt(1, date[0]);
102        oldScoreStmt.setInt(2, date[1]);
103        oldScoreStmt.setInt(3, date[2]);
104        ResultSet oldScoreRs = oldScoreStmt.executeQuery();
105        int hiscore = -1;
106        if(oldScoreRs.next())hiscore = oldScoreRs.getInt("Highscore"); //if a score exists for this date, take it for current the hiscore
107        if(hiscore== -1) { //score doesn't already exist
108            insScoreStmt.setInt(1, date[0]);
109            insScoreStmt.setInt(2, date[1]);
110            insScoreStmt.setInt(3, date[2]);
111            insScoreStmt.setInt(4, result);
112            insScoreStmt.executeUpdate();
113        } else if(result > hiscore) { //if score exists and actually has improved
114            updScoreStmt.setInt(1, result);
115            updScoreStmt.setInt(2, date[0]);
116            updScoreStmt.setInt(3, date[1]);
117            updScoreStmt.setInt(4, date[2]);
118            updScoreStmt.executeUpdate();
119        }
120
121        //commit the changes to the database
122        conn.commit();
123
124    } catch(SQLException ex) {
125        ex.printStackTrace();
126    }
127 }
128
129 public int[] getDate() {
130     Calendar now = Calendar.getInstance();
131     int day = now.get(Calendar.DAY_OF_MONTH);
132     int month = now.get(Calendar.MONTH) + 1; //zero indexed, +1 to make january 1, feb 2, ...
133     int year = now.get(Calendar.YEAR);
134
135     return new int[] {day, month, year};
136 }
137 @FXML public void playAgain(ActionEvent e) {
138     //go back to starting screen
139     Parent root = null;
140     FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/test.fxml"));
141     try {
142         root = loader.load();
143     } catch(IOException ioe) {ioe.printStackTrace();}
144     TestController controller = loader.getController();
145     controller.setRootController(rootController);
146     rootController.setActivity(root);
147
148     root.requestFocus(); //so doesn't randomly highlight a button
149 }
150 }
```

TheoryQuestion.java

Similar to Flashcard.java, we use another model for the theory questions of the quiz. This allows us to store the data for it much more easily.

It should store the question, the array of answers for that question, and the index for the correct answer.

Code for TheoryQuestion.java:

```
1 package model;
2
3 public class TheoryQuestion {
4     private String question;
5     private String[] answers = new String[4];
6     private int answerIndex;
7     public TheoryQuestion(String question, String[] answers, int answerIndex) {
8         this.question= question;
9         this.answers = answers;
10        this.answerIndex = answerIndex;
11    }
12    public void setQuestion(String question) {
13        this.question = question;
14    }
15    public void setAnswers(String a1, String a2, String a3, String a4) {
16        answers[0] = a1;
17        answers[1] = a2;
18        answers[2] = a3;
19        answers[3] = a4;
20    }
21    public void setIndex(int index) {
22        answerIndex = index;
23    }
24    public String getQuestion() {
25        return question;
26    }
27    public String[] getAnswers() {
28        return answers;
29    }
30    public String getAnswer(int index) {
31        return answers[index];
32    }
33    public int getAnswerIndex() {
34        return answerIndex;
35    }
36 }
```

Feedback.java

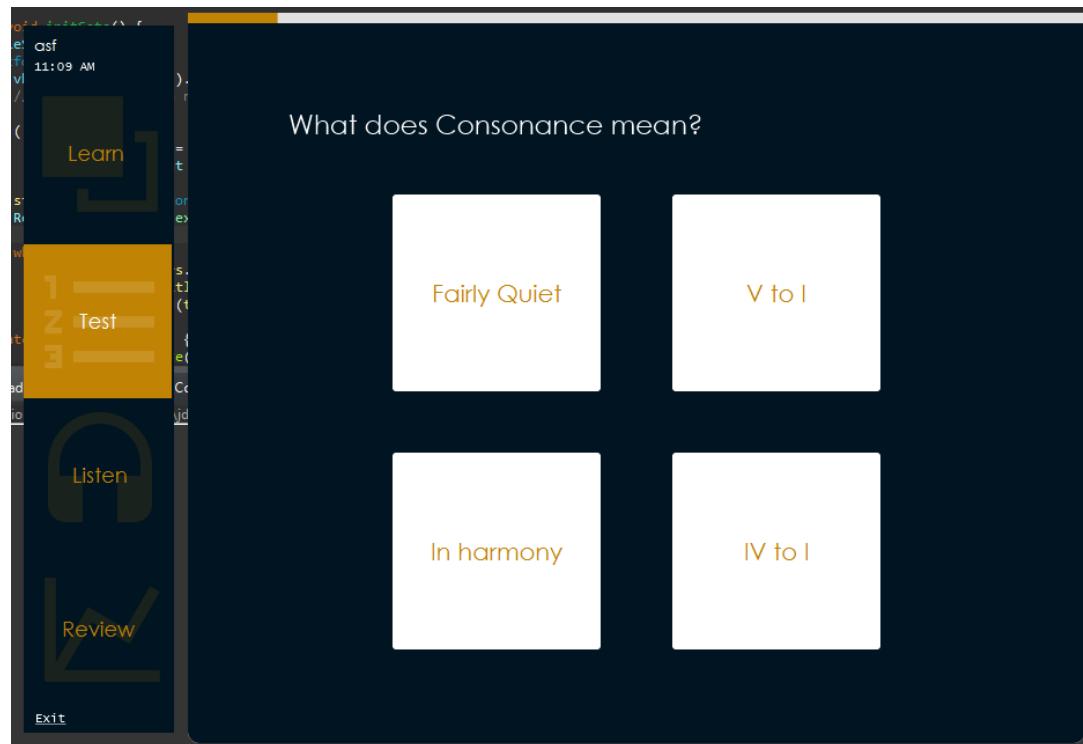
To conclude our Test section, storing our data for our feedback, we have another model called Feedback.java which simply stores our question, wrong answer (user inputted), and correct answer.

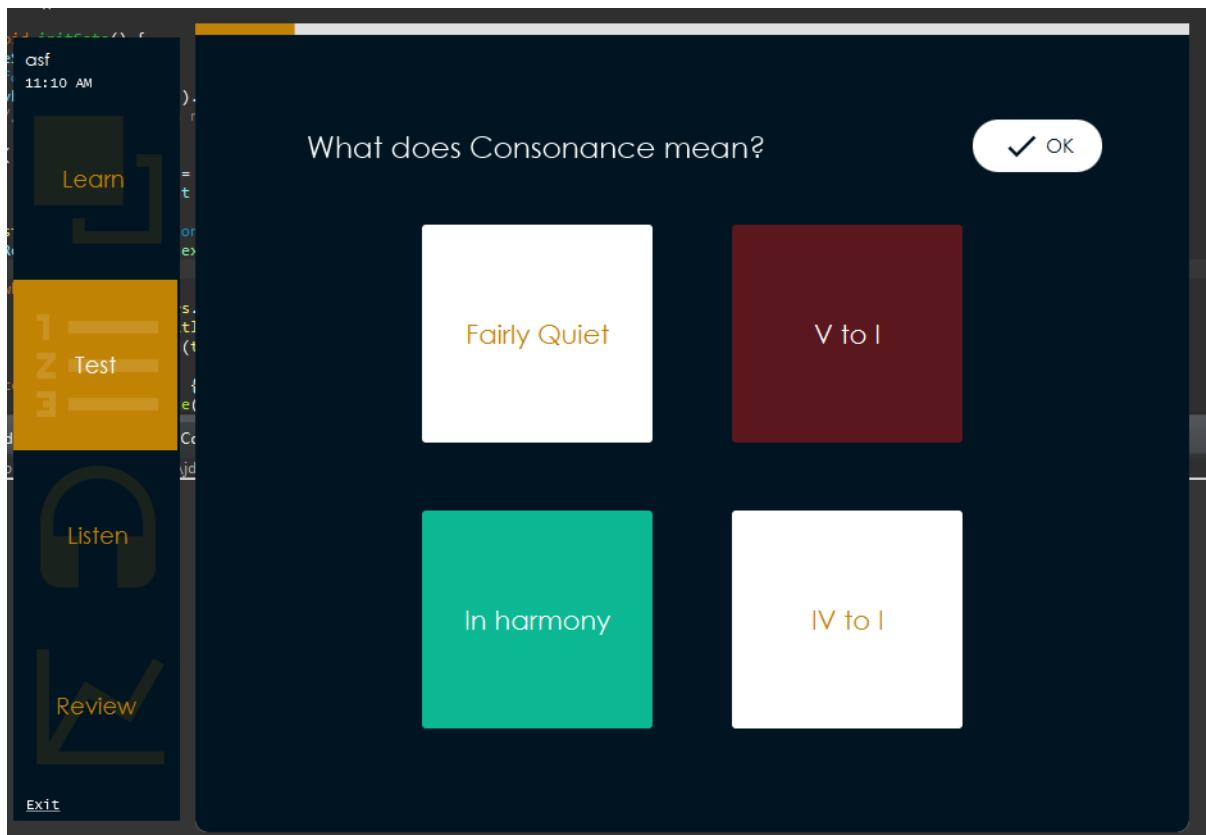
Code for Feedback.java:

```
1 package model;
2
3 public class Feedback {
4     private String question;
5     private String wrongAnswer;
6     private String correctAnswer;
7     public Feedback(String question, String wrong, String correct) {
8         this.question = question;
9         wrongAnswer = wrong;
10        correctAnswer = correct;
11    }
12    public String getQuestion() {
13        return question;
14    }
15    public String getWrong() {
16        return wrongAnswer;
17    }
18    public String getCorrect() {
19        return correctAnswer;
20    }
21 }
```

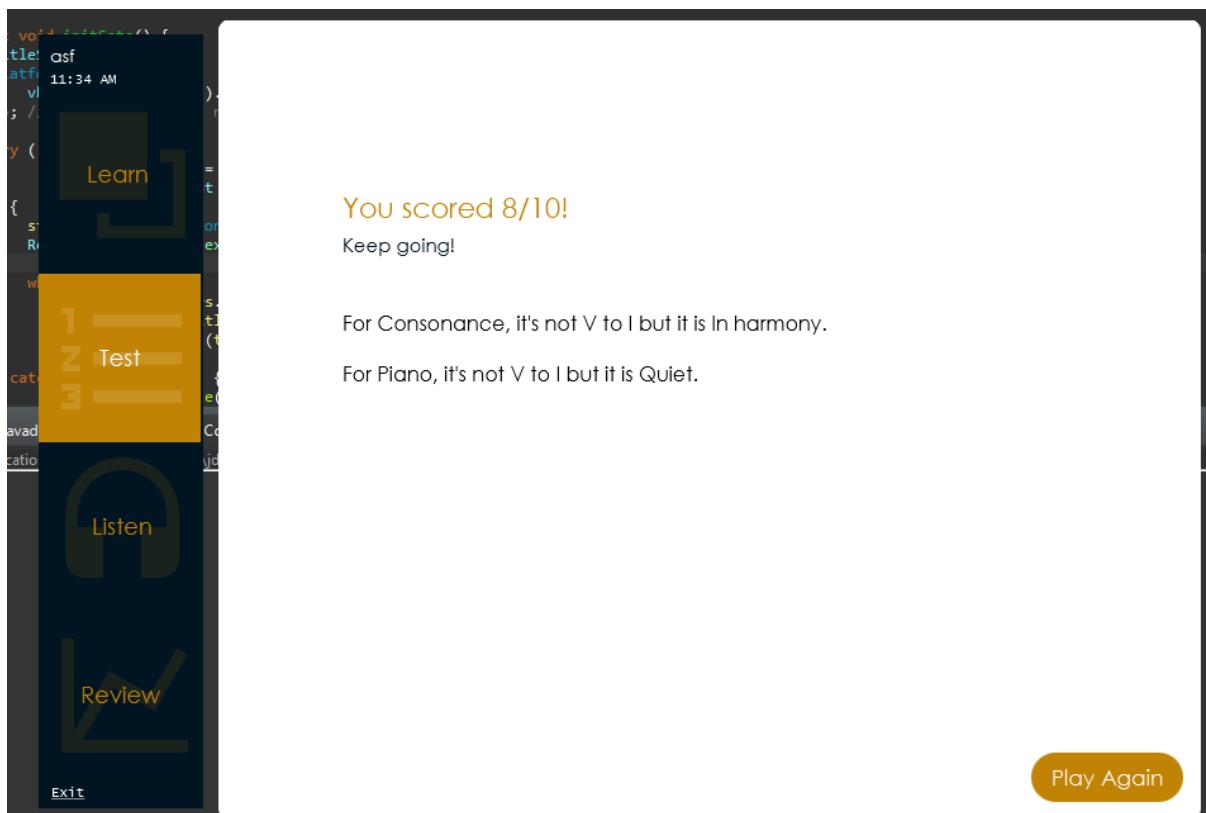
Validation Test

As a final validation test for our Test section, we can do a final run through to test all the features.

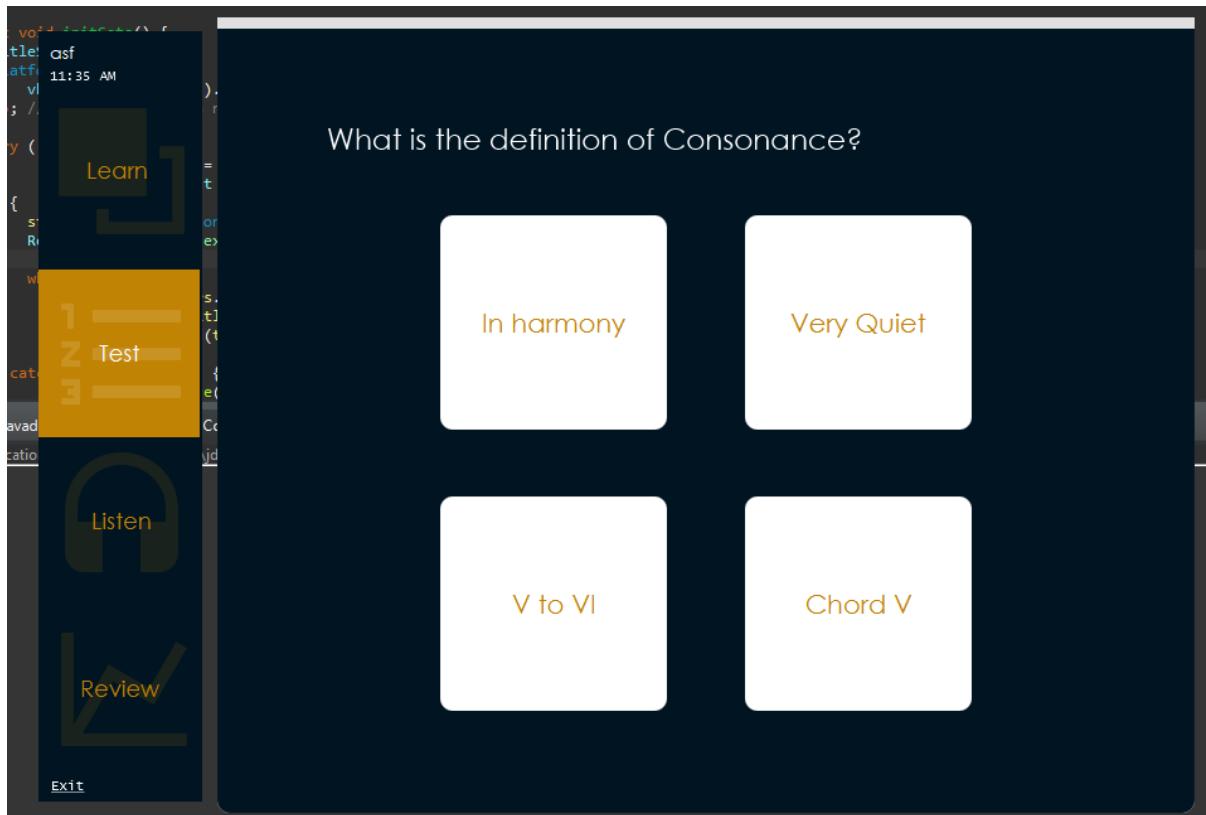




The feedback correctly displays.



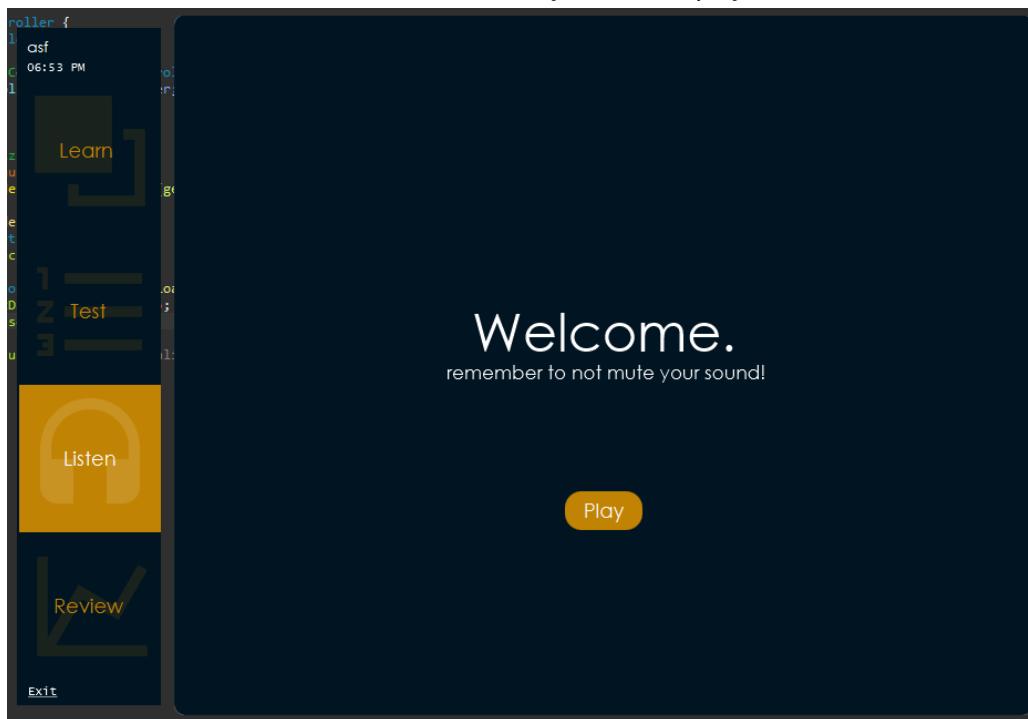
Also, our next test has adapted and shown the high priority term: Consonance.



Overall, the Test feature has become a successful asset to our system.

ListenController.java

Our next section is the Listen section. Firstly we will display a welcome screen, `listen.fxml`.



All we need to do here is just handle the play button just like the test welcome screen.

```

@FXML
public void playQuiz(ActionEvent e) {
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/listen_quiz.fxml"));
    try {
        root = loader.load();
    } catch(IOException ex) {
        ex.printStackTrace();
    }
    ListenQuizController controller = loader.getController();
    controller.initData(rootController);
    rootController.setActivity(root);

    root.requestFocus(); //doesn't highlight a random button
}

```

Code for ListenController.java:

```

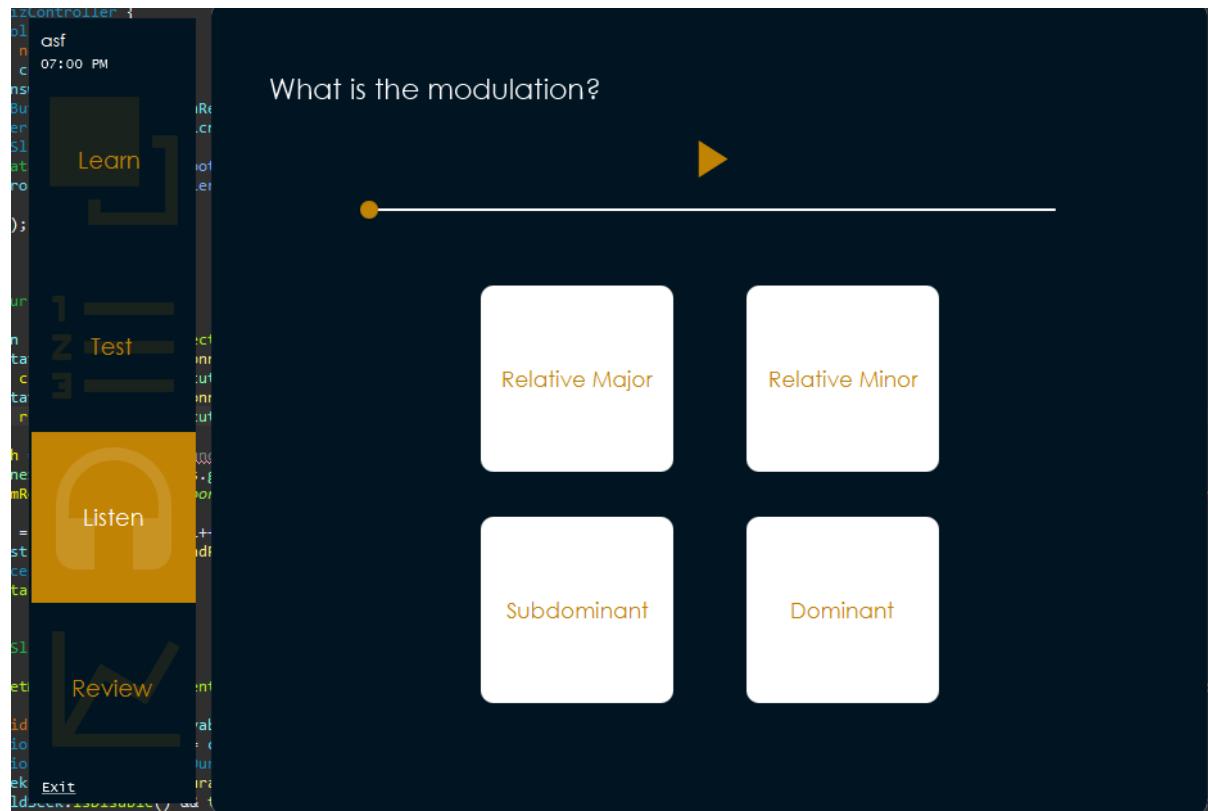
1 package controller;
2
3 import java.io.IOException;
4
5 import javafx.event.ActionEvent;
6 import javafx.fxml.FXML;
7 import javafx.fxml FXMLLoader;
8 import javafx.scene.Parent;
9
10 public class ListenController {
11     private RootController rootController;
12
13     public void setRootController(RootController rootController) {
14         this.rootController = rootController;
15     }
16
17     @FXML
18     public void playQuiz(ActionEvent e) {
19         Parent root = null;
20         FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/listen_quiz.fxml"));
21         try {
22             root = loader.load();
23         } catch(IOException ex) {
24             ex.printStackTrace();
25         }
26         ListenQuizController controller = loader.getController();
27         controller.initData(rootController);
28         rootController.setActivity(root);

29         root.requestFocus(); //doesn't highlight a random button
30     }
31 }
32

```

ListenQuizController.java

This is the controller for our quiz screen, listen_quiz.fxml.



For this Listen section, we will use a separate table, *Aural*, to denote the information for each aural test.

Field Name	Data Type	Description (Optional)
ID	AutoNumber	Unique ID for each aural test
File	Short Text	File name for the aural recording

ID	File	Modulation	Click to Add
1	dom1.wav	Dominant	
2	dom2.wav	Dominant	
3	dom3.wav	Dominant	
4	dom4.wav	Dominant	
5	dom5.wav	Dominant	
6	subdom1.wav	Subdominant	
7	subdom2.wav	Subdominant	
8	subdom3.wav	Subdominant	
9	subdom4.wav	Subdominant	
10	subdom5.wav	Subdominant	
11	relmaj1.wav	Relative Major	
12	relmaj2.wav	Relative Major	
13	relmaj3.wav	Relative Major	
14	relmaj4.wav	Relative Major	
15	relmaj5.wav	Relative Major	
16	relmin1.wav	Relative Minor	
17	relmin2.wav	Relative Minor	
18	relmin3.wav	Relative Minor	
19	relmin4.wav	Relative Minor	
20	relmin5.wav	Relative Minor	
*	(New)		

Here we provide the audio files for the aural test (authentically made by myself) and have an even split between the number of each modulation so that the user gets a good amount of practice for each modulation.

Within ListenQuizController.java, when the controller object is initialised, we need to get the question and answer like the theory test but also synchronise the slider with the audio so that we can use it to navigate and also to simply check where we are in the audio.

```
public void initData(RootController rootController) {
    this.rootController = rootController;

    getAuralTest();
    syncSlider();
}
```

Within the getAuralTest() method, we will, similar to the Test section, retrieve a random aural test. So, we get the number of aural tests from the database, table Aural, and then iterate to a random row. Also, we can store this current test as a global variable (class variable) so it is easier to access throughout this class.

```
private AuralTest currentTest = null;
```

```
private void getAuralTest() {
    try {
        Connection conn = dao.getConnection();
        PreparedStatement cntStmt = conn.prepareStatement("SELECT COUNT(*) FROM Aural");
        ResultSet cntRs = cntStmt.executeQuery();
        PreparedStatement rndStmt = conn.prepareStatement("SELECT File, Modulation FROM Aural");
        ResultSet rndRs = rndStmt.executeQuery();
    } {
        int length = 0; //the unbound for random row
        if(cntRs.next())length = cntRs.getInt(1);
        int randomRow = (int) Math.floor(length * Math.random()); //floor for lower bound to generate 'length' number of integers (including 0 excluding end)
        for(int i = 0; i<=randomRow; i++)rndRs.next(); //go to random row <= to move past zeroth row
        currentTest = new AuralTest(rndRs.getString("File"), rndRs.getString("Modulation"));
    } catch(SQLException ex) {
        ex.printStackTrace();
    }
}
```

Next, to sync the slider, we can use similar code to what is demonstrated in the Java documentation: <https://docs.oracle.com/javase/8/javafx/media-tutorial/playercontrol.htm>.

Following a structure similar to theirs, we only need to focus on matching the audio time (currentTimeProperty) with the Slider's value.

```
currentTest.getMediaPlayer().currentTimeProperty().addListener(new InvalidationListener() { //to sync the slider with the media player being played, automatically
    @Override
    public void invalidated(Observable o) {
        Duration currentDuration = currentTest.getMediaPlayer().getCurrentTime();
        Duration totalDuration = Duration.millis(currentTest.getDuration());
        sldSeek.setDisable(totalDuration.isUnknown()); //disable slider if the music has not been correctly loaded yet
        if(!sldSeek.isDisable() && totalDuration.greaterThan(Duration.ZERO)&& !sldSeek.isValueChanging()) {
            double percentageIn = (currentDuration.toMillis()/totalDuration.toMillis()) * 100;
            sldSeek.setValue(percentageIn);
        }
    }
});
```

Here, we add an InvalidationListener to the currentTimeProperty() so whenever the currentTime of the audio changes, it can be handled. We get the duration of the currentTime and the total duration of the audio, using this, we can provide a percentage of the way through to set on the slider, sldSeek. If the totalDuration.isUnknown() returns true, then we need to disable sldSeek since the MediaPlayer (the object which plays the sound) did not load correctly and sliding sldSeek would cause serious errors. The *if-statement* contains three smaller Boolean

comparisons: firstly we check if the slider is disabled, if not, OK; the duration needs to be higher than zero meaning that there is audio to be played and navigated to in the first place; the slider is not being changed by the user, otherwise changing the value whilst the user is sliding it will cause the slider to move back and forth and ‘glitch’. Within the *if-statement*, we calculate the percentage within the audio and set the slider value to the new updated value.

Additionally, the slider needs to be listened for any movements so that the user input of changing the slider position will be noticed.

```
sldSeek.valueProperty().addListener(new InvalidationListener() {
    @Override
    public void invalidated(Observable o) {
        if(sldSeek.isValueChanging()) { //if the slider is truly moving
            Duration newDuration = Duration.millis(currentTest.getDuration()).multiply(sldSeek.getValue()/100.0);
            currentTest.getMediaPlayer().seek(newDuration);
        }
    }
});
```

Here, we add another InvalidationListener but to the valueProperty() of the slider since we are listening for changes in the slider position. The newDuration is the new position of which the MediaPlayer of the test needs to follow and its value is the total duration of the aural test identified by currentTest.getDuration() multiplied by the percentage in of the position on the slider.

For the play and pause button of this section, it is actually the same icon but changes its glyph (shape) depending on what is required. So, if the piece is to be paused, it will display the PAUSE glyph, if the piece is *already* paused, then the PLAY glyph will be used.

```
@FXML
public void playPause(MouseEvent e) {
    if(icnPlayPause.getGlyphName().equals("PLAY")) {
        currentTest.getMediaPlayer().play();
        icnPlayPause.setGlyphName("PAUSE");
    } else {
        currentTest.getMediaPlayer().pause();
        icnPlayPause.setGlyphName("PLAY");
    }
}
```

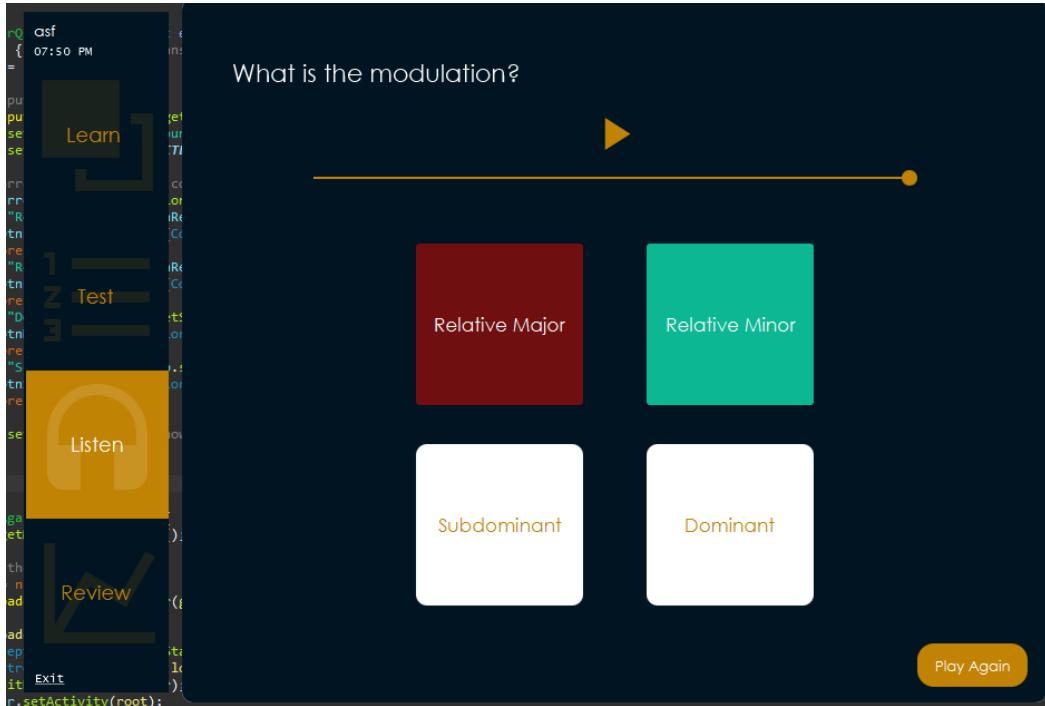
To respond to the user answering the question, we will change their input to red and the correct answer to green, overriding the red to green if the user gets the answer correctly. Also, we have considered whether the answer has already been answered yet, if so, then do not colour every single button!

```
@FXML
public void answerQuestion(ActionEvent e) {
    if(!answered) { //if hasn't been answered yet - this will make clicks to other buttons not change their colour
        answered = true;

        //make input red
        Button inputBtn = (Button) e.getSource();
        inputBtn.setStyle("-fx-background-color: #f0f0f0");
        inputBtn.setTextFill(Color.WHITE);

        //make correct answer green - correct input will be overrided green also
        switch(currentTest.getModulation()) {
            case "Relative Major": btnRelMaj.setStyle("-fx-background-color: #0cb794");
                btnRelMaj.setTextFill(Color.WHITE);
                break;
            case "Relative Minor": btnRelMin.setStyle("-fx-background-color: #0cb794");
                btnRelMin.setTextFill(Color.WHITE);
                break;
            case "Dominant": btnDom.setStyle("-fx-background-color: #0cb794");
                btnDom.setTextFill(Color.WHITE);
                break;
            case "Subdominant": btnSub.setStyle("-fx-background-color: #0cb794");
                btnSub.setTextFill(Color.WHITE);
                break;
        }
        btnAgain.setVisible(true); //show play again
    }
}
```

This is extremely similar to the Test section however the only difference is that we use a `switch` case selection rather than an array and index combination.



Akin to `btnOk`, the `btnAgain` button will be revealed at the end when the user has answered the test. Instead of moving onto the next question, however, `btnAgain` allows the user to instantly try

again but with a different piece of audio. This is different to the Test feature since this combines the Learn and Test features into one for musical aural testing so has the competitive sense of answering correctly, but also the relaxing sense of not having results recorded.

Concisely, we will display the `listen_quiz.fxml` again, initialising a new controller so that we have a new test.

```
@FXML
public void playAgain(ActionEvent e) {
    currentTest.getMediaPlayer().stop(); //stop the music if it's still playing

    //load again the quiz page
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/listen_quiz.fxml"));
    try {
        root = loader.load();
    } catch(IOException ex) {ex.printStackTrace();}
    ListenQuizController controller = loader.getController();
    controller.initData(rootController);
    rootController.setActivity(root);

    root.requestFocus(); //so doesn't focus on a button
}
```

Code for `ListenQuizController.java`:

```
1 package controller;
2
3 import java.io.IOException;
4 import java.sql.Connection;
5 import java.sql.PreparedStatement;
6 import java.sql.ResultSet;
7 import java.sql.SQLException;
8
9 import com.jfoenix.controls.JFXButton;
10 import com.jfoenix.controls.JFXSlider;
11
12 import de.jensd.fx.glyphs.materialdesignicons.MaterialDesignIconView;
13 import javafx.beans.InvalidationListener;
14 import javafx.beans.Observable;
15 import javafx.event.ActionEvent;
16 import javafx.fxml.FXML;
17 import javafx.fxml.FXMLLoader;
18 import javafx.scene.Parent;
19 import javafx.scene.control.Button;
20 import javafx.scene.input.MouseEvent;
21 import javafx.scene.paint.Color;
22 import javafx.util.Duration;
23 import model.AuralTest;
24 import other.DAO;
25
26 public class ListenQuizController {
27     private RootController rootController;
28     private DAO dao = new DAO();
29     private AuralTest currentTest = null;
```

```

30     private boolean answered = false;
31     @FXML private JFXButton btnRelMaj, btnRelMin, btnSub, btnDom, btnAgain;
32     @FXML private MaterialDesignIconView icnPlayPause;
33     @FXML private JFXSlider sldSeek;
34     public void initData(RootController rootController) {
35         this.rootController = rootController;
36
37         getAuralTest();
38         syncSlider();
39     }
40
41     private void getAuralTest() {
42         try {
43             Connection conn = dao.getConnection();
44             PreparedStatement cntStmt = conn.prepareStatement("SELECT COUNT(*) FROM Aural");
45             ResultSet cntRs = cntStmt.executeQuery();
46             PreparedStatement rndStmt = conn.prepareStatement("SELECT File, Modulation FROM Aural");
47             ResultSet rndRs = rndStmt.executeQuery();
48         } {
49             int length = 0; //the upper bound for random row
50             if(cntRs.next())length = cntRs.getInt(1);
51             int randomRow = (int) Math.floor(length * Math.random()); //floor for lower bound to generate 'length' number of integers (inc 0, exc end)
52
53             for(int i = 0; i<=randomRow; i++)rndRs.next(); //go to random row <= to move past zeroth row
54
55             currentTest = new AuralTest(rndRs.getString("File"), rndRs.getString("Modulation"));
56         } catch(SQLException ex) {
57             ex.printStackTrace();
58         }
59     }
60
61     private void syncSlider() {
62
63         currentTest.getMediaPlayer().currentTimeProperty().addListener(new InvalidationListener() { //to auto sync the slider with the media player being played
64             @Override
65             public void invalidated(Observable o) {
66                 Duration currentDuration = currentTest.getMediaPlayer().getCurrentTime();
67                 Duration totalDuration = Duration.millis(currentTest.getDuration());
68                 sldSeek.setDisable(totalDuration.isUnknown()); //disable slider if the music has not been correctly loaded yet
69                 if(!sldSeek.isDisable() && totalDuration.greaterThan(Duration.ZERO)&& !sldSeek.isValueChanging()) {
70                     double percentageIn = (currentDuration.toMillis()/totalDuration.toMillis()) * 100;
71                     sldSeek.setValue(percentageIn);
72                 }
73             }
74         });
75
76         sldSeek.valueProperty().addListener(new InvalidationListener() {
77             @Override
78             public void invalidated(Observable o) {
79                 if(sldSeek.isValueChanging()) { //if the slider is truly moving
80                     Duration newDuration = Duration.millis(currentTest.getDuration()).multiply(sldSeek.getValue()/100.0);
81                     currentTest.getMediaPlayer().seek(newDuration);
82                 }
83             }
84         });
85     }
86     @FXML
87     public void playPause(MouseEvent e) {
88         if(icnPlayPause.getGlyphName().equals("PLAY")) {
89             currentTest.getMediaPlayer().play();
90             icnPlayPause.setGlyphName("PAUSE");
91         } else {
92             currentTest.getMediaPlayer().pause();
93             icnPlayPause.setGlyphName("PLAY");
94         }
95     }
96     @FXML
97     public void answerQuestion(ActionEvent e) {
98         if(!answered) { //if hasn't been answered yet - this will make clicks to other buttons not change their colour
99             answered = true;
100
101            //make input red
102            Button inputBtn = (Button) e.getSource();
103            inputBtn.setStyle("-fx-background-color: #6f0f0f");
104            inputBtn.setTextFill(Color.WHITE);
105
106            //make correct answer green - correct input will be overriden green also
107            switch(currentTest.getModulation()) {
108                case "Relative Major": btnRelMaj.setStyle("-fx-background-color: #0cb794");
109                btnRelMaj.setTextFill(Color.WHITE);
110                break;
111                case "Relative Minor": btnRelMin.setStyle("-fx-background-color: #0cb794");
112                btnRelMin.setTextFill(Color.WHITE);
113                break;
114                case "Dominant": btnDom.setStyle("-fx-background-color: #0cb794");
115                btnDom.setTextFill(Color.WHITE);
116                break;
117                case "Subdominant": btnSub.setStyle("-fx-background-color: #0cb794");
118                btnSub.setTextFill(Color.WHITE);
119                break;
120            }
121            btnAgain.setVisible(true); //show play again
122        }
123    }

```

```

124
125● @FXML
126 public void playAgain(ActionEvent e) {
127     currentTest.getMediaPlayer().stop(); //stop the music if it's still playing
128
129     //load again the quiz page
130     Parent root = null;
131     FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/listen_quiz.fxml"));
132     try {
133         root = loader.load();
134     } catch(IOException ex) {ex.printStackTrace();}
135     ListenQuizController controller = loader.getController();
136     controller.initData(rootController);
137     rootController.setActivity(root);
138
139     root.requestFocus(); //so doesn't focus on a button
140 }
141
142 }
143 }
```

AuralTest.java

We have another model to store the data about our aural test.

This will be slightly more complicated than the other models, however similar, since we are using audio files from our /resources/audio directory. To be precise, we will need to load up the audio and put it into an object of type MediaPlayer which allows us to play the sound from the file and do audio related stuff, move position etc. Also, once the MediaPlayer is loaded up, we can set the duration instance variable of the AuralTest class to be the same as the total duration of the MediaPlayer.

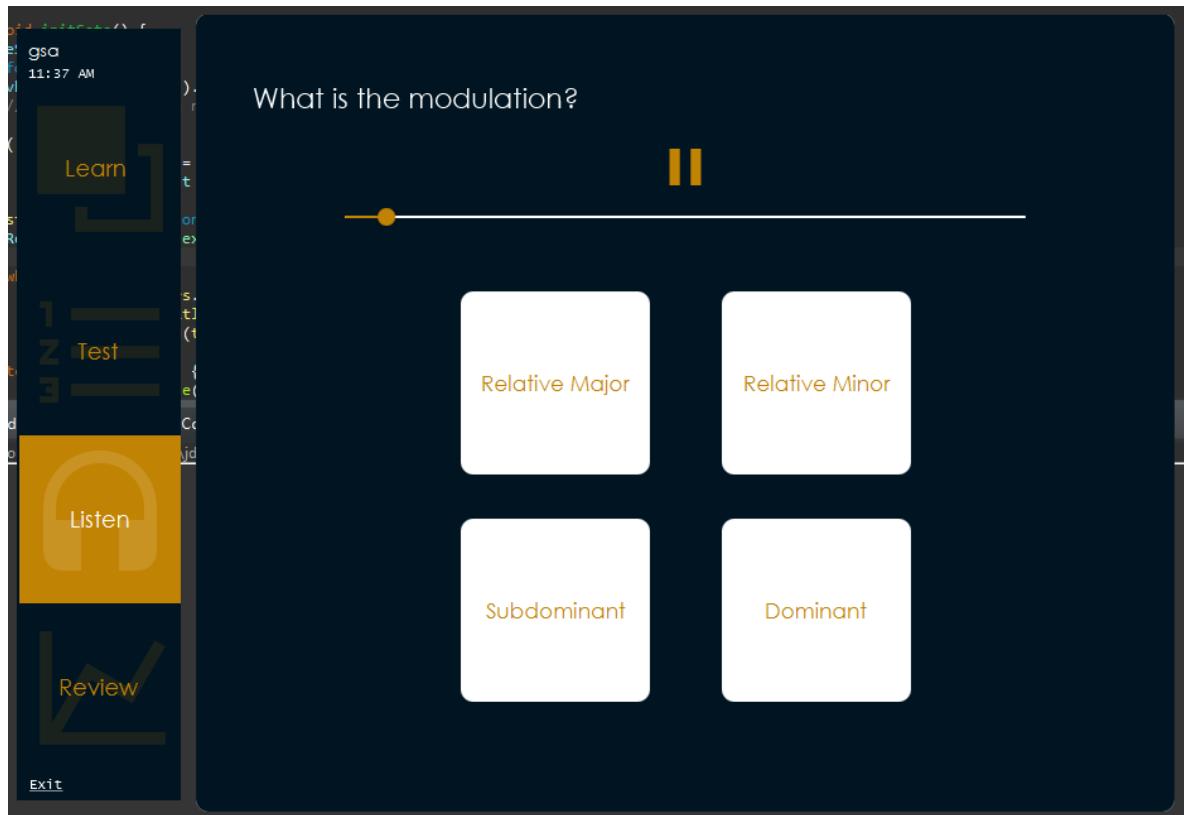
Code for AuralTest.java:

```

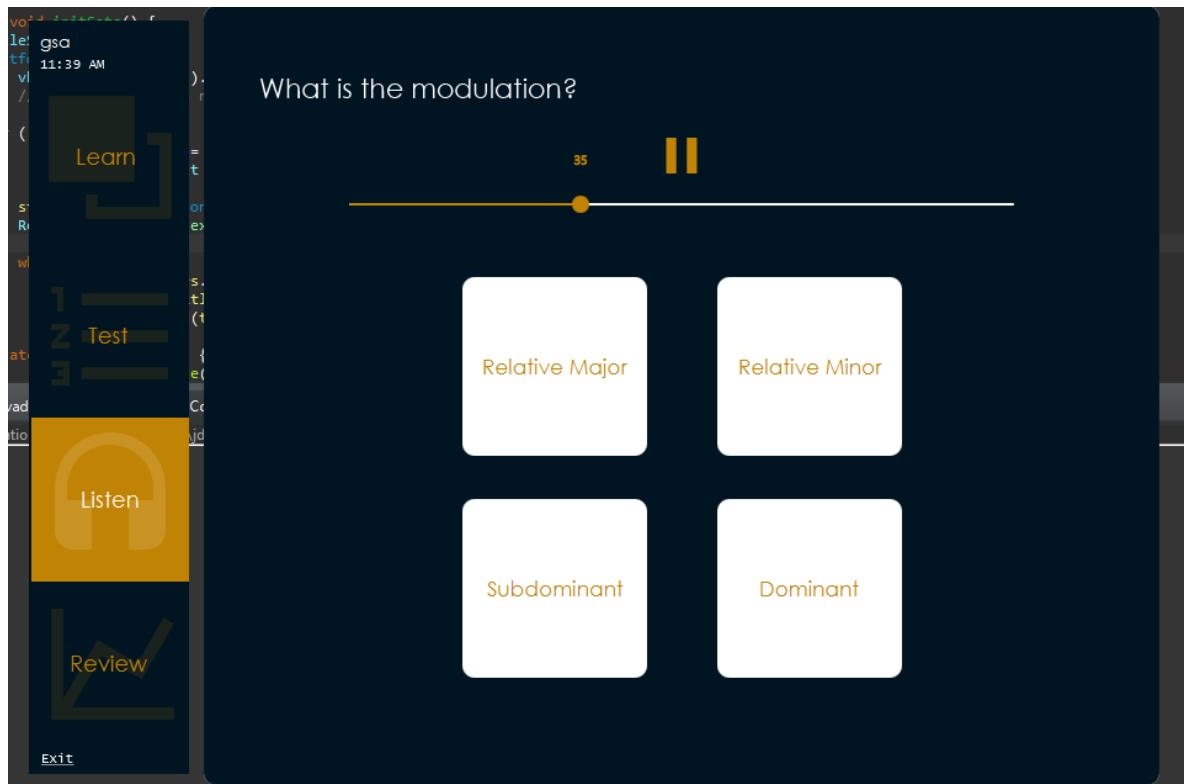
1 package model;
2
3● import javafx.scene.media.Media;
4 import javafx.scene.media.MediaPlayer;
5
6 public class AuralTest {
7     private String audioFile;
8     private MediaPlayer auralPlayer;
9     private String modulation;
10    private double duration;
11
12● public AuralTest(String audioFile, String modulation) {
13     this.audioFile = audioFile;
14     this.modulation = modulation;
15     createAuralPlayer();
16 }
17
18● private void createAuralPlayer() {
19     Media media = new Media(getClass().getResource("/resources/audio/" + audioFile).toExternalForm());
20     auralPlayer = new MediaPlayer(media);
21     auralPlayer.setOnReady(() -> duration = auralPlayer.getMedia().getDuration().toMillis());
22 }
23
24● public MediaPlayer getMediaPlayer() {
25     return auralPlayer;
26 }
27● public String getModulation() {
28     return modulation;
29 }
30● public double getDuration() {
31     return duration;
32 }
33 }
```

Validation Test

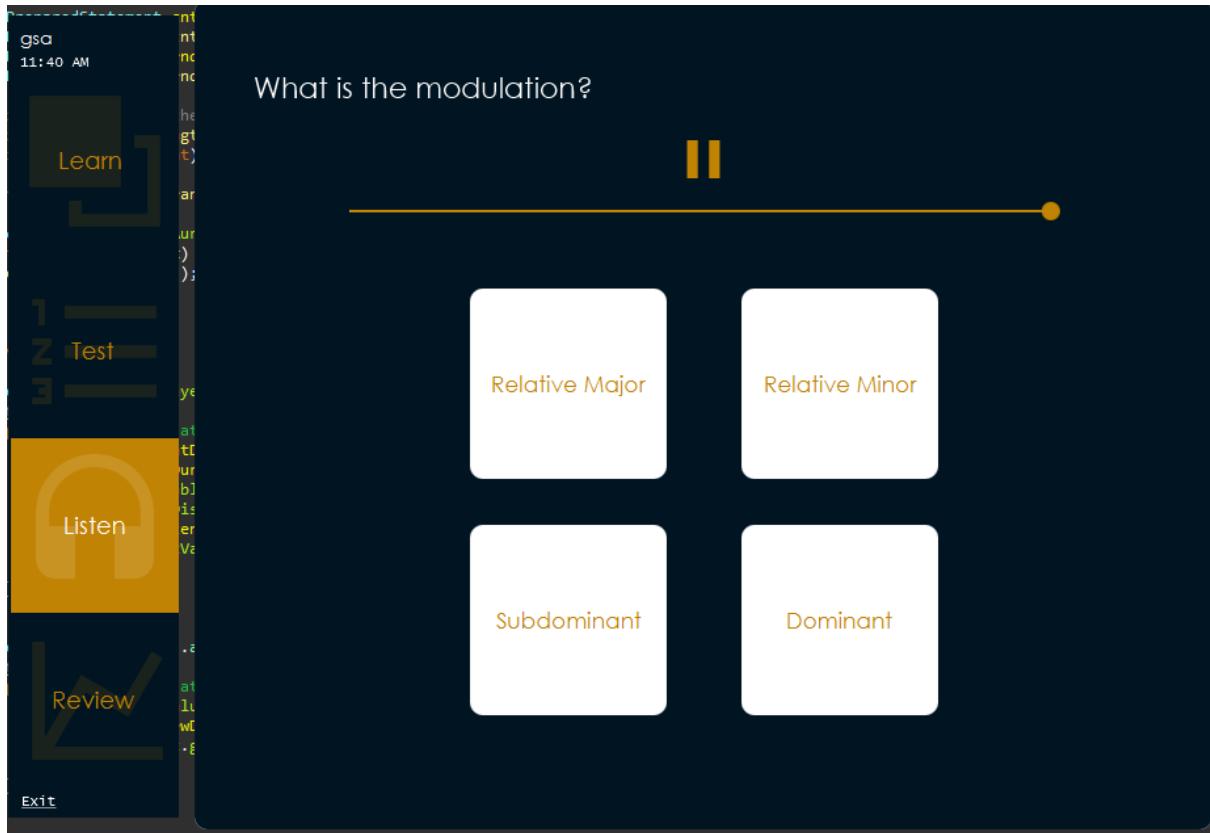
Before moving onto our review section, we will test the Listen section rigorously.



The music correctly plays until the end.



The slider works but at the end, the pause button still shows when the music has stopped and also does not play the music unless we slide it right to the beginning.



So, to combat this we can add a small block of code into LearnQuizController to handle what happens when the media finishes when slider the value of the slider to 100, or if the media finishes on its own.

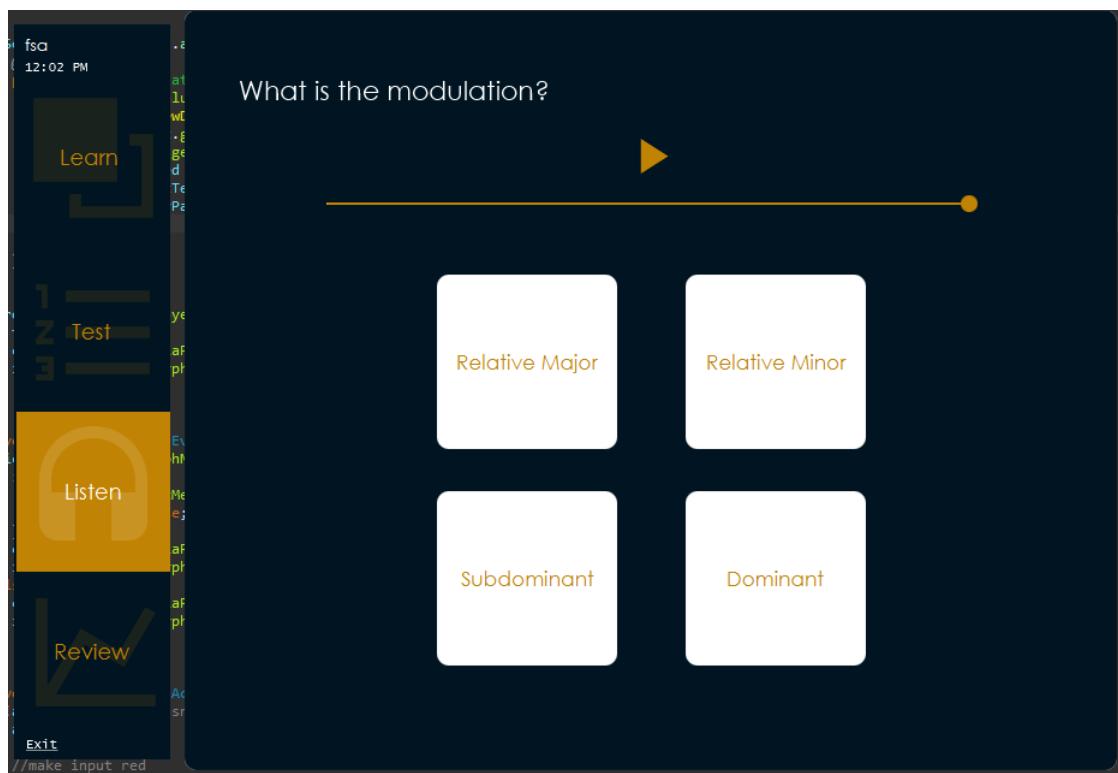
```
sldSeek.valueProperty().addListener(new InvalidationListener() {
    @Override
    public void invalidated(Observable o) {
        if(sldSeek.isValueChanging()) { //if the slider is truly moving
            Duration newDuration = Duration.millis(currentTest.getDuration()).multiply(sldSeek.getValue()/100.0);
            currentTest.getMediaPlayer().seek(newDuration);
        }
        if(sldSeek.getValue()==100.0) {
            finished = true;
            currentTest.getMediaPlayer().pause();
            icnPlayPause.setGlyphName("PLAY");
        }
    }
});

currentTest.getMediaPlayer().setOnEndOfMedia(()-> {
    finished = true;
    currentTest.getMediaPlayer().pause();
    icnPlayPause.setGlyphName("PLAY");
});
```

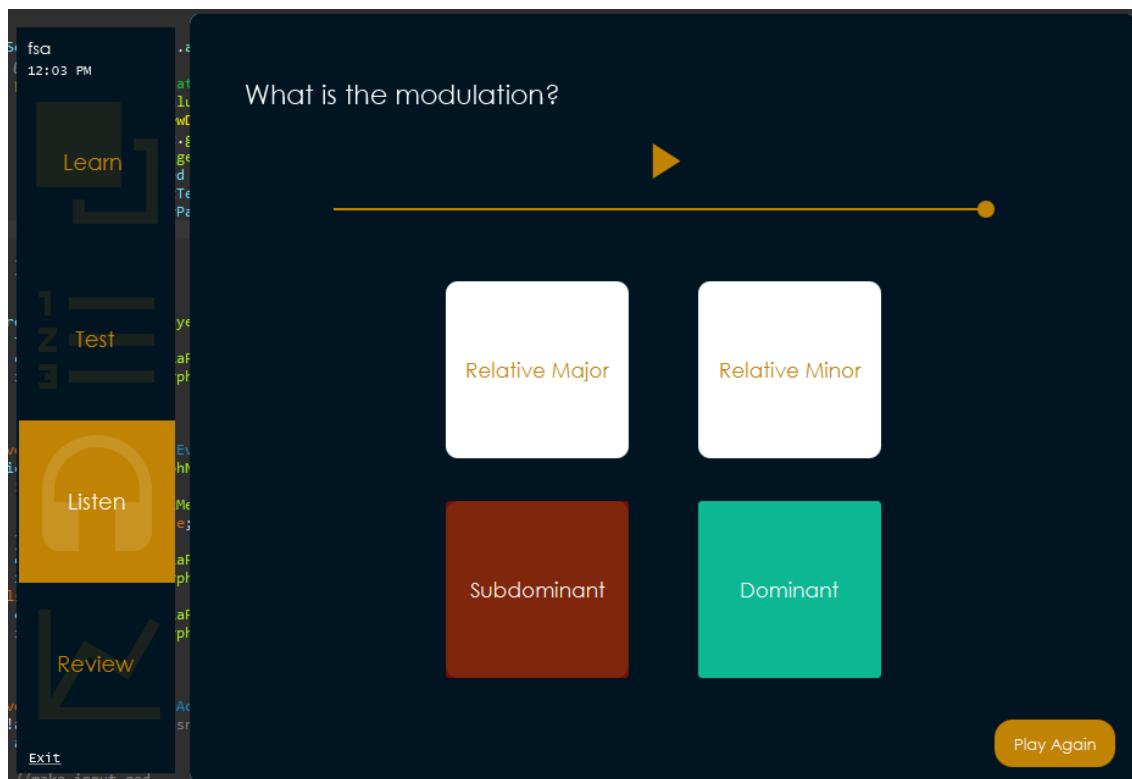
Setting the class variable Boolean, finished, to true, we can use that to automatically go to the start when pressing the play button again.

```
@FXML
public void playPause(MouseEvent e) {
    if(icnPlayPause.getGlyphName().equals("PLAY")) {
        if(finished) {
            currentTest.getMediaPlayer().seek(currentTest.getMediaPlayer().getStartTime());
            finished = false;
        }
    }
}
```

Now, the end-of-media handling works very well.



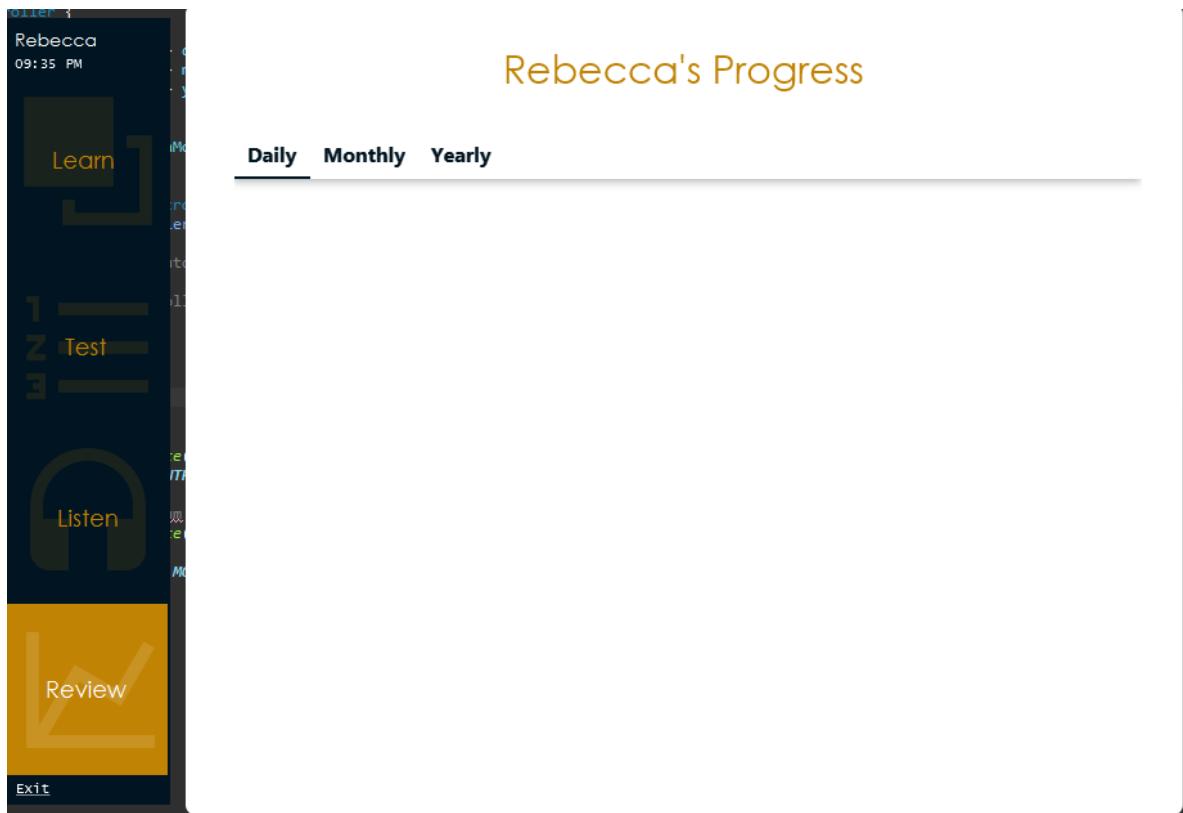
Also, the feedback correctly displays.



To conclude this section, the Listen section uses a lot of technical hardware, media techniques but the test and simplicity of using this program remains; overall, a good feature implemented.

ReviewController.java

For our final section of the program, Review, we will display multiple graphs showing improvements over various scales of time using our view: review.fxml.



However, JavaFX requires ourselves to provide GUI to display the graphs rather than using JavaFX Scene Builder – which is fine.

At the start, we will need to create individual graphs (LineCharts) and then get the data for the graphs.

```
public void setRootController(RootController rootController) {  
    this.rootController = rootController;  
}  
public void initialize() { //called automatically after fxml injection  
    Platform.runLater(() -> {  
        txtProgress.setText(rootController.getName() + "'s All-Time Progress");  
        getScores();  
        setDailyGraph();  
        setMonthlyGraph();  
        setYearlyGraph();  
    });  
}
```

Although what above could be achieved with code all inside the `setRootController()` method which is called on initialisation, the method `initialize()` automatically is called by the FX Application Thread when an object of this class is initialised – hence the name of the method.

Before we start coding the GUI graphs, we need to provide very important methods which identify the current date.

```
public int getDate() {
    Calendar now = Calendar.getInstance();
    return now.get(Calendar.DAY_OF_MONTH);
}
public String getMonth(int num) { //num = -1 for NOW month, 0 <= num <= 11 for inputs
    Calendar now = Calendar.getInstance();
    String month = null;
    switch(num== -1 ? now.get(Calendar.MONTH) : num) {
        case 0: month = "Jan";
        break;
        case 1: month = "Feb";
        break;
        case 2: month = "Mar";
        break;
        case 3: month = "Apr";
        break;
        case 4: month = "May";
        break;
        case 5: month = "Jun";
        break;
        case 6: month = "Jul";
        break;
        case 7: month = "Aug";
        break;
        case 8: month = "Sep";
        break;
        case 9: month = "Oct";
        break;
        case 10: month = "Nov";
        break;
        case 11: month = "Dec";
        break;
    }
    return month;
}
public int getYear() {
    Calendar now = Calendar.getInstance();
    return now.get(Calendar.YEAR);
}
```

This uses the `Calendar` data type and returns primitive types (int or String in this case) to be used as the X axis for our graphs (time axis).

Also, it is best to initialise the data (`XYChart.Series`) for our charts as a class variable so it can be easily accessed later for updating etc.

```
private XYChart.Series<Number, Number> dailySeries = new XYChart.Series<>();
private XYChart.Series<String, Number> monthlySeries = new XYChart.Series<>();
private XYChart.Series<Number, Number> yearlySeries = new XYChart.Series<>();
```

The data types within the `<>` parameters specify the data type of the x axis then y axis. For example, `monthlySeries` has a `String` data type for the x axis since we are specifying the months as `Strings`.

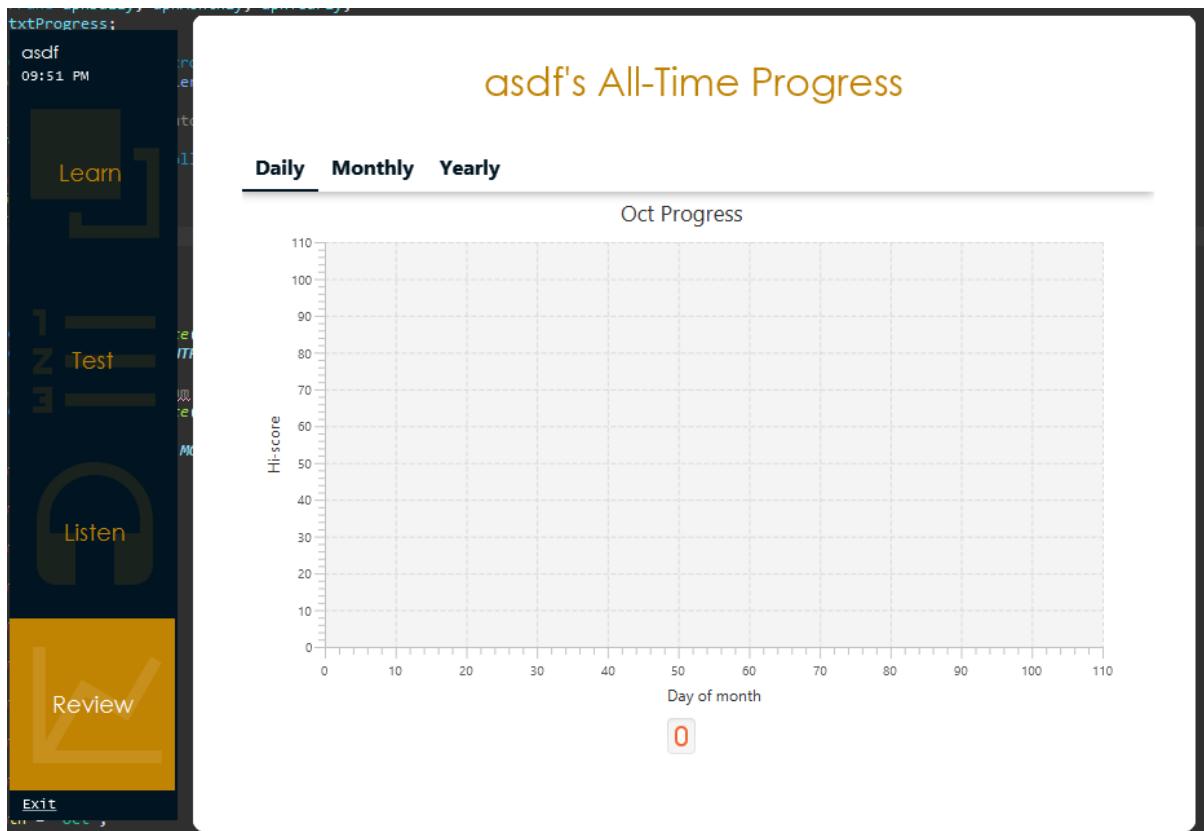
To dissect the process of drawing the graphs, we firstly need to create the axes, get the chart, set the data and axes for the chart, and add the chart to the visual interface.

```

public void setDailyGraph() {
    //create axes
    final NumberAxis xAxis = new NumberAxis();
    final NumberAxis yAxis = new NumberAxis();
    //set axes label
    xAxis.setLabel("Day of month");
    yAxis.setLabel("Hi-score");
    //the line chart object, axes specified in parameters
    final LineChart<Number, Number> dailyChart = new LineChart<>(xAxis, yAxis);
    //set title
    dailyChart.setTitle(getMonth(-1) + " Progress");
    //set dimensions 680 x 440 px
    dailyChart.setMaxSize(680, 440);
    dailyChart.setMinSize(680, 440);
    dailyChart.setPrefSize(680, 440);
    //add data for chart
    dailyChart.getData().add(dailySeries);
    //add chart to interface
    apnDaily.getChildren().add(dailyChart);
}

```

So far, the graph looks like this.



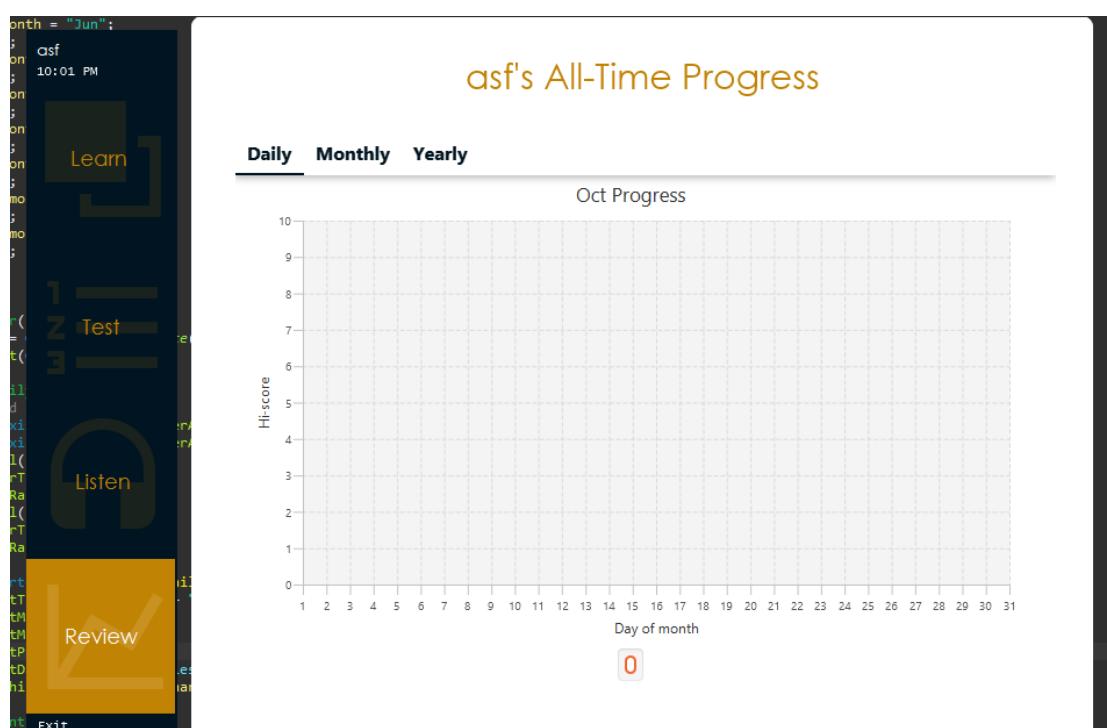
However, there are some small issues to address, firstly, the range goes up to 110 when there are only 31 days max in a month, the high score also reaches 110 when the high score is only 10.

We can fix this by changing the range of the axes using the constructor of NumberAxis and specifying the lower bound, upper bound and tick length – where tick length is the length before a small line in the axis.

```

public void setDailyGraph() {
    //set axes and customise
    final NumberAxis xAxis = new NumberAxis(1, 31, 1);
    final NumberAxis yAxis = new NumberAxis(0, 10, 1);
    xAxis.setLabel("Day of month");
    xAxis.setMinorTickVisible(false);
    xAxis.setAutoRanging(false);
    yAxis.setLabel("Hi-score");
    yAxis.setMinorTickVisible(false);
    yAxis.setAutoRanging(false);
}

```



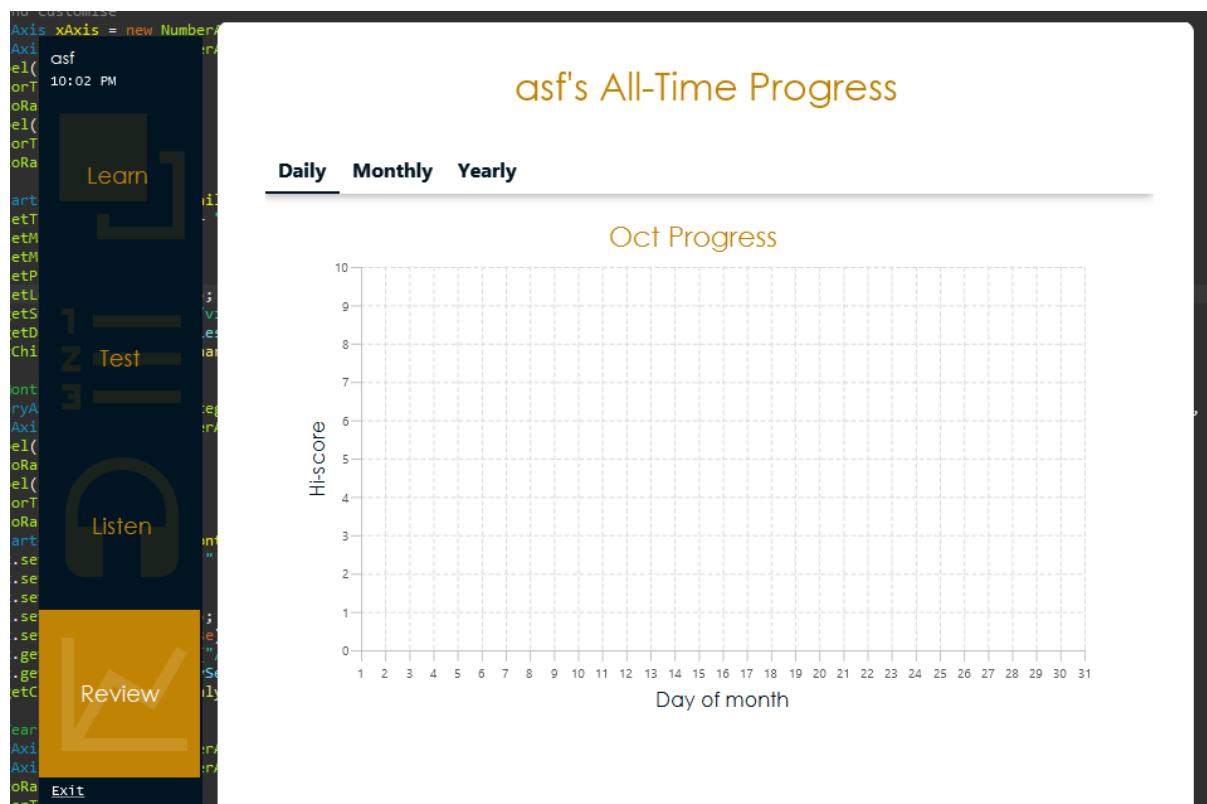
Next, we can remove the legend and customise some of the graph's features using CSS – external CSS called `chart.css` – will be described later.

```

public void setDailyGraph() {
    //set axes and customise
    final NumberAxis xAxis = new NumberAxis(1, 31, 1);
    final NumberAxis yAxis = new NumberAxis(0, 10, 1);
    xAxis.setLabel("Day of month");
    xAxis.setMinorTickVisible(false);
    xAxis.setAutoRanging(false);
    yAxis.setLabel("Hi-score");
    yAxis.setMinorTickVisible(false);
    yAxis.setAutoRanging(false);

    //the line chart object, axes specified in parameters
    final LineChart<Number, Number> dailyChart = new LineChart<>(xAxis, yAxis);
    //customise chart
    dailyChart.setTitle(getMonth(-1) + " Progress");
    dailyChart.setMaxSize(680, 440);
    dailyChart.setMinSize(680, 440);
    dailyChart.setPrefSize(680, 440);
    dailyChart.setLegendVisible(false);
    dailyChart.getStylesheets().add("/view/chart.css");
    //add data for chart
    dailyChart.getData().add(dailySeries);
    //add chart to interface
    apnDaily.getChildren().add(dailyChart);
}

```



We can rinse and repeat for the other two methods.

```
public void setMonthlyGraph() {
    final CategoryAxis xAxis = new CategoryAxis(FXCollections.observableArrayList("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"));
    final NumberAxis yAxis = new NumberAxis(0, 10, 1);
    xAxis.setLabel("Month in year");
    xAxis.setAutoRanging(false);
    yAxis.setLabel("Average score");
    yAxis.setMinorTickLength(0.25);
    yAxis.setAutoRanging(false);
    final LineChart<String, Number> monthlyChart = new LineChart<>(xAxis, yAxis);
    monthlyChart.setTitle(getYear() + " Progress");
    monthlyChart.setMaxSize(680, 440);
    monthlyChart.setMinSize(680, 440);
    monthlyChart.setPrefSize(680, 440);
    monthlyChart.setLegendVisible(false);
    monthlyChart.getStylesheets().add("/view/chart.css");
    monthlyChart.getData().add(monthlySeries);
    apnMonthly.getChildren().add(monthlyChart);
}
public void setYearlyGraph() {
    final NumberAxis xAxis = new NumberAxis(2015, 2050, 1);
    final NumberAxis yAxis = new NumberAxis(0, 10, 1);
    xAxis.setAutoRanging(false);
    xAxis.setMinorTickVisible(false);
    xAxis.setLabel("Year");
    yAxis.setLabel("Average score");
    yAxis.setMinorTickLength(0.25);
    yAxis.setAutoRanging(false);
    final LineChart<Number, Number> yearlyChart = new LineChart<>(xAxis, yAxis);
    yearlyChart.setTitle("Yearly Progress");
    yearlyChart.setMaxSize(680, 440);
    yearlyChart.setMinSize(680, 440);
    yearlyChart.setPrefSize(680, 440);
    yearlyChart.setLegendVisible(false);
    yearlyChart.getStylesheets().add("/view/chart.css");
    yearlyChart.getData().add(yearlySeries);
    apnYearly.getChildren().add(yearlyChart);
}
```

Now for our `getScores()` method, we need to get multiple sets of scores for the three separate charts. Firstly, we need to retrieve all the separate scores for each day in the current month with this query.

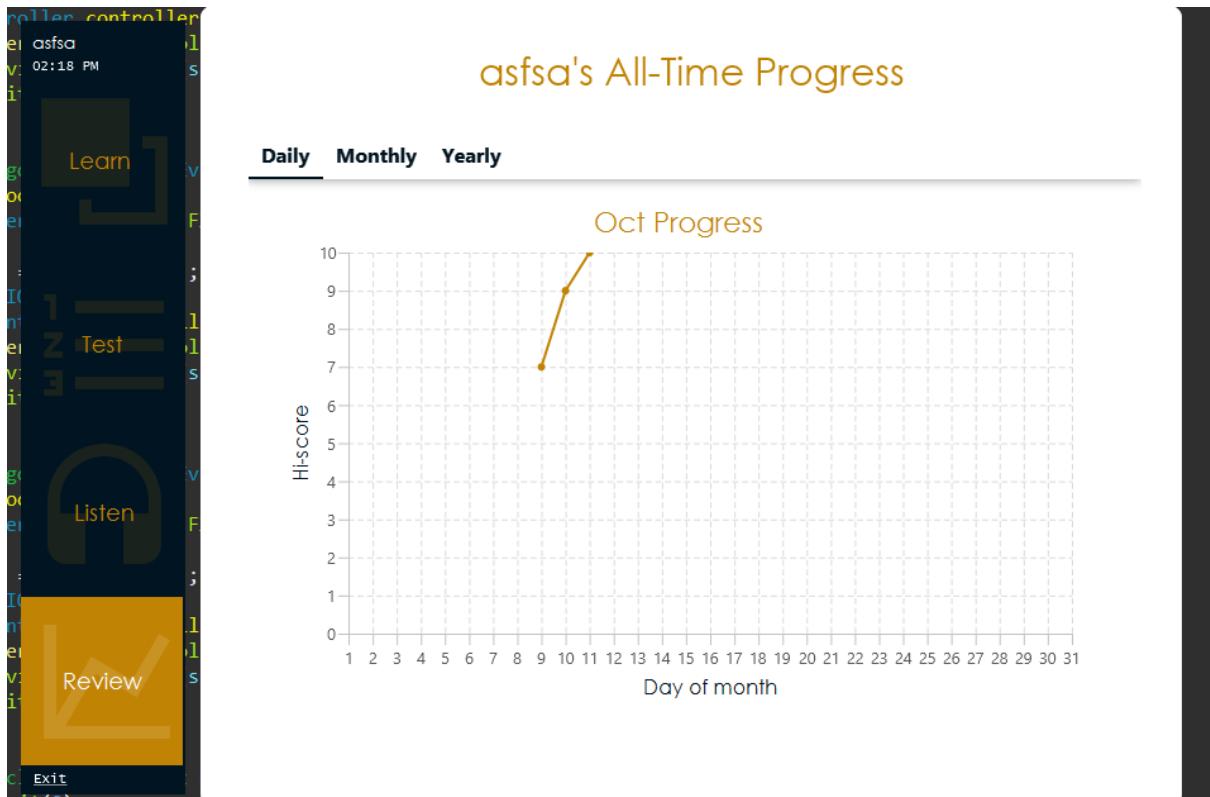
```
PreparedStatement dailyStmt = conn.prepareStatement("SELECT Highscore, ScoreDay FROM Score WHERE ScoreMonth = ?");
```

For the daily graph, we just need to add each point in the month to the Series and automatically, the graph will have changed since the Series is the data for our graph.

```
public void getScores() {
    //set series data
    try{
        Connection conn = dao.getConnection();
        PreparedStatement dailyStmt = conn.prepareStatement("SELECT Highscore, ScoreDay FROM Score WHERE ScoreMonth = ?");
    } {
        //daily graph
        Calendar now = Calendar.getInstance();
        dailyStmt.setInt(1, now.get(Calendar.MONTH)+1); //+1 to change from zero indexed to one indexed
        ResultSet dailyRs = dailyStmt.executeQuery();
        while(dailyRs.next()) { //iterate through each data point
            XYChart.Data<Number, Number> dayPoint = new XYChart.Data<>(dailyRs.getInt("ScoreDay"), dailyRs.getInt("Highscore"));
            //add the new data to the series, which will automatically add to the graph
            dailySeries.getData().add(dayPoint);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

After inputting some test data and running the program we have a graph!

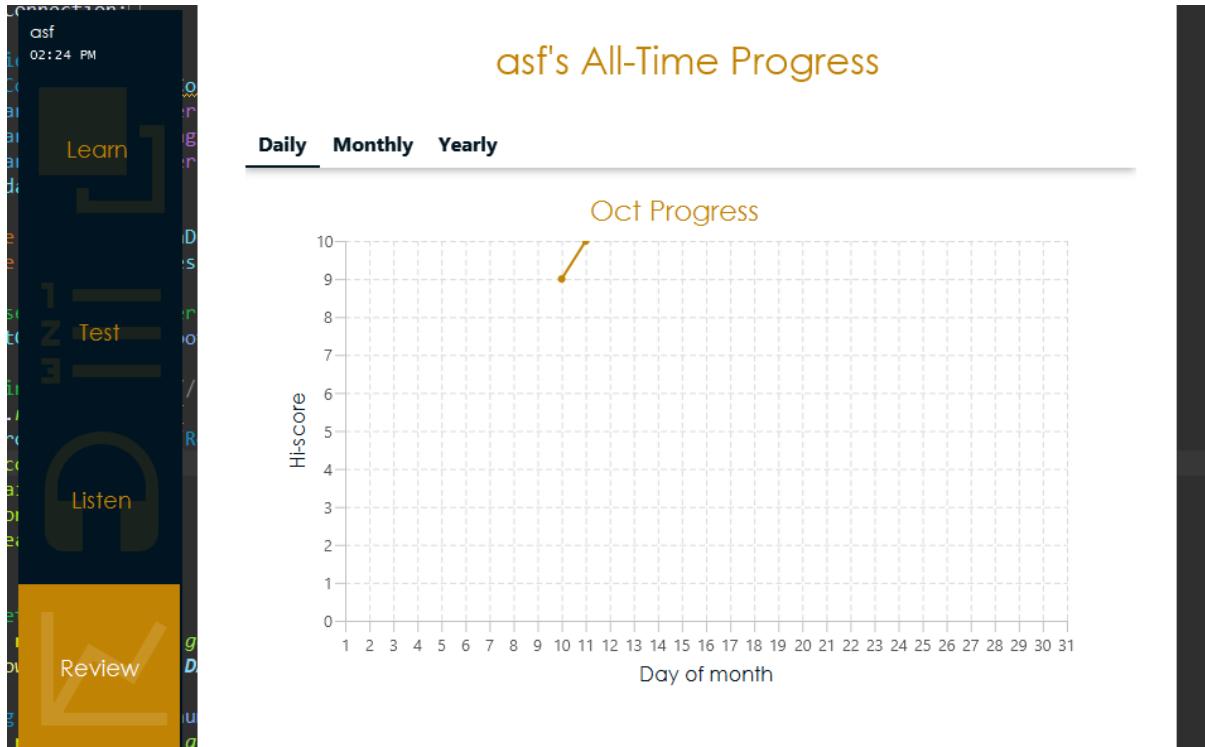
ID	ScoreDay	ScoreMonth	ScoreYear	Highscore	Click to Add
1	9	10	2017	7	
4	10	10	2018	9	
5	11	10	2018	10	
13	12	5	2018	5	
(New)	0	0	0	0	



However, a slight problem with it is that it has used our first data point which was indeed in October, but last year! So, to address this, we can add an extra WHERE condition to the query regarding ScoreYear.

```
try{
    Connection conn = dao.getConnection();
    PreparedStatement dailyStmt = conn.prepareStatement("SELECT Highscore, ScoreDay FROM Score WHERE ScoreMonth = ? AND ScoreYear = ?");
} {
    //daily graph
    Calendar now = Calendar.getInstance();
    dailyStmt.setInt(1, now.get(Calendar.MONTH)+1);
    dailyStmt.setInt(2, getYear());
    ResultSet dailyRs = dailyStmt.executeQuery();
    while(dailyRs.next()) {
        XYChart.Data<Number, Number> dayPoint = new XYChart.Data<>(dailyRs.getInt("ScoreDay"), dailyRs.getInt("Highscore"));
        dailySeries.getData().add(dayPoint);
    }
}
```

And now only uses our two data points of October 2018.



For our monthly and yearly graphs, since there will be multiple scores throughout the time span of a month or a year, we can take an average of those scores to make it more representative of the whole month/year. This means we can add two more queries to our *try-with-resources* clause. Also, we can use ORDER BY to order all the rows with the same months/years (depending on the query) and group them together.

```
public void getScores() {
    //set series data
    try{
        Connection conn = dao.getConnection();
        PreparedStatement dailyStmt = conn.prepareStatement("SELECT Highscore, ScoreDay FROM Score WHERE ScoreMonth = ? AND ScoreYear = ?");
        PreparedStatement monthlyStmt = conn.prepareStatement("SELECT Highscore, ScoreMonth FROM Score WHERE ScoreYear = ? ORDER BY ScoreMonth");
        PreparedStatement yearlyStmt = conn.prepareStatement("SELECT Highscore, ScoreYear FROM Score ORDER BY ScoreYear");
    } {
        //do something
    }
}
```

To calculate an average, we need to get the total score, mTotal or yTotal, and the number of rows we used to count it (the frequency), mFreq or yFreq.

```
double mTotal = 0;
double mFreq = 0;
```

For the month graph, to add the cumulative total over a month, we can iterate through each row on the database and increment the mTotal by the highscore at that day if that day is inside the same month and year, also incrementing mFreq by one.

```
while(monthlyRs.next()) {
    if(getMonth(monthlyRs.getInt("ScoreMonth")-1).equals(lastMonth)) {
        mTotal += monthlyRs.getInt("Highscore");
        mFreq++;
    }
}
```

However, when the month changes, we need to add the average score (`mTotal/mFreq`) to the Series data and refresh the values of `mTotal` and `mFreq` to zero for the next month.

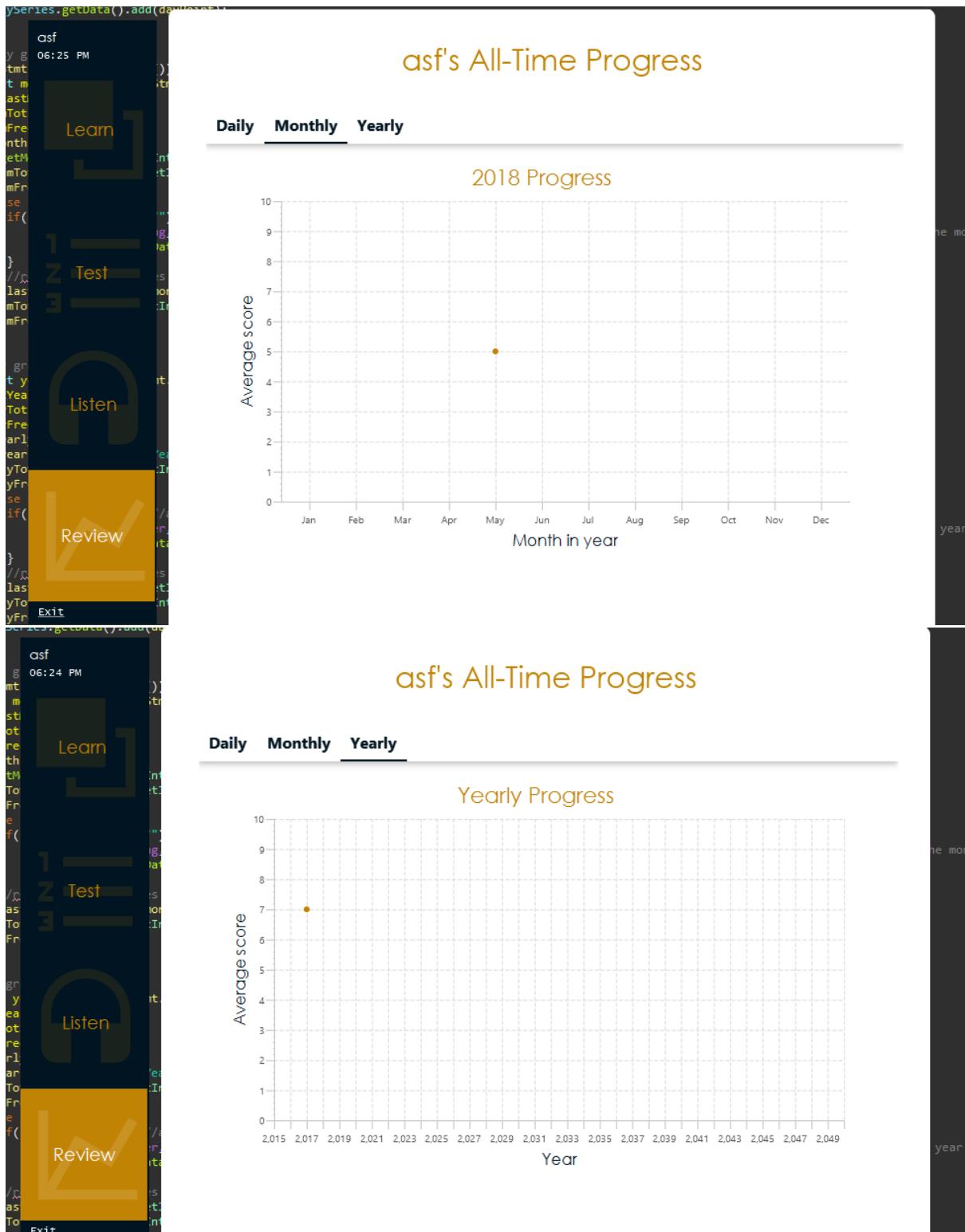
```
//monthly graph
monthlyStmt.setInt(1, getYear());
ResultSet monthlyRs = monthlyStmt.executeQuery();
String lastMonth = "";
double mTotal = 0;
double mFreq = 0;
while(monthlyRs.next()) {
    if(getMonth(monthlyRs.getInt("ScoreMonth"))-1.equals(lastMonth)) {
        mTotal += monthlyRs.getInt("Highscore");
        mFreq++;
    } else {
        if(!lastMonth.equals("")) { //after the first, if the month changes, add the last month's scores to the graph
            XYChart.Data<String, Number> monthPoint = new XYChart.Data<>(lastMonth, mTotal/mFreq); //total/frequency is the average score of the month
            monthlySeries.getData().add(monthPoint);
        }
        //reinitialize variables (including first score)
        lastMonth = getMonth(monthlyRs.getInt("ScoreMonth"))-1; //method is 0 indexed, db is 1 indexed so -1
        mTotal = monthlyRs.getInt("Highscore");
        mFreq = 1;
    }
}
```

Also, instead of resetting the variables `mTotal` and `mFreq` to 0, and then adding the first day of the next month's value onto them, we can just set the values to the first day value.

The yearly graph will apply the same logic but with different variable names to a different Series, `yearlySeries`.

```
//monthly graph
monthlyStmt.setInt(1, getYear());
ResultSet monthlyRs = monthlyStmt.executeQuery();
String lastMonth = "";
double mTotal = 0;
double mFreq = 0;
while(monthlyRs.next()) {
    if(getMonth(monthlyRs.getInt("ScoreMonth"))-1.equals(lastMonth)) {
        mTotal += monthlyRs.getInt("Highscore");
        mFreq++;
    } else {
        if(!lastMonth.equals("")) { //after the first, if the month changes, add the last month's avg scores to the graph
            XYChart.Data<String, Number> monthPoint = new XYChart.Data<>(lastMonth, mTotal/mFreq); //total/frequency is the average score of the month
            monthlySeries.getData().add(monthPoint);
        }
        //reinitialize variables (including first score)
        lastMonth = getMonth(monthlyRs.getInt("ScoreMonth"))-1; //method is 0 indexed, db is 1 indexed so -1
        mTotal = monthlyRs.getInt("Highscore");
        mFreq = 1;
    }
}
//yearly graph
ResultSet yearlyRs = yearlyStmt.executeQuery();
int lastYear = -1;
double yTotal = 0;
double yFreq = 0;
while(yearlyRs.next()) {
    if(yearlyRs.getInt("ScoreYear") == lastYear) {
        yTotal += yearlyRs.getInt("Highscore");
        yFreq++;
    } else {
        if(lastYear != -1) { //after the first, if the year changes, add the last year's avg scores to the graph
            XYChart.Data<Number, Number> yearPoint = new XYChart.Data<>(lastYear, yTotal/yFreq); //total/frequency is the average score of the year
            yearlySeries.getData().add(yearPoint);
        }
        //reinitialize variables (including first score)
        lastYear = yearlyRs.getInt("ScoreYear"); //method is 0 indexed, db is 1 indexed so -1
        yTotal = yearlyRs.getInt("Highscore");
        yFreq = 1;
    }
}
```

But another problem appeared.



For the monthly graph, there should be two data points, one in May, another in October, and for the yearly graph, there is meant to be data points for two years displayed, one for 2017, another for 2018.

If we review our logic of the code, we only add the Series data point when the month changes to a new one – but what happens after the last data point? Nothing. So after our while loop, we can add another data point for the final month/year in our database using our unrefreshed values of mTotal or yTotal and mFreq or yFreq.

```

//monthly graph
monthlyStmt.setInt(1, getYear());
ResultSet monthlyRs = monthlyStmt.executeQuery();
String lastMonth = "";
double mTotal = 0;
double mFreq = 0;
while(monthlyRs.next()) {
    if(getMonth(monthlyRs.getInt("ScoreMonth")-1).equals(lastMonth)) {
        mTotal += monthlyRs.getInt("Highscore");
        mFreq++;
    } else {
        if(!lastMonth.equals("")) { //after the first, if the month changes, add the last month's avg scores to the graph
            XYChart.Data<String, Number> monthPoint = new XYChart.Data<>(lastMonth, mTotal/mFreq); //total/frequency is the average score of the month
            monthlySeries.getData().add(monthPoint);
        }
        //reinitialize variables (including first score)
        lastMonth = getMonth(monthlyRs.getInt("ScoreMonth")-1); //method is 0 indexed, db is 1 indexed so -1
        mTotal = monthlyRs.getInt("Highscore");
        mFreq = 1;
    }
}
//at the end, add the final last month (since it won't change to another month after the last month)
XYChart.Data<String, Number> finalMonthPoint = new XYChart.Data<>(lastMonth, mTotal/mFreq);
monthlySeries.getData().add(finalMonthPoint);

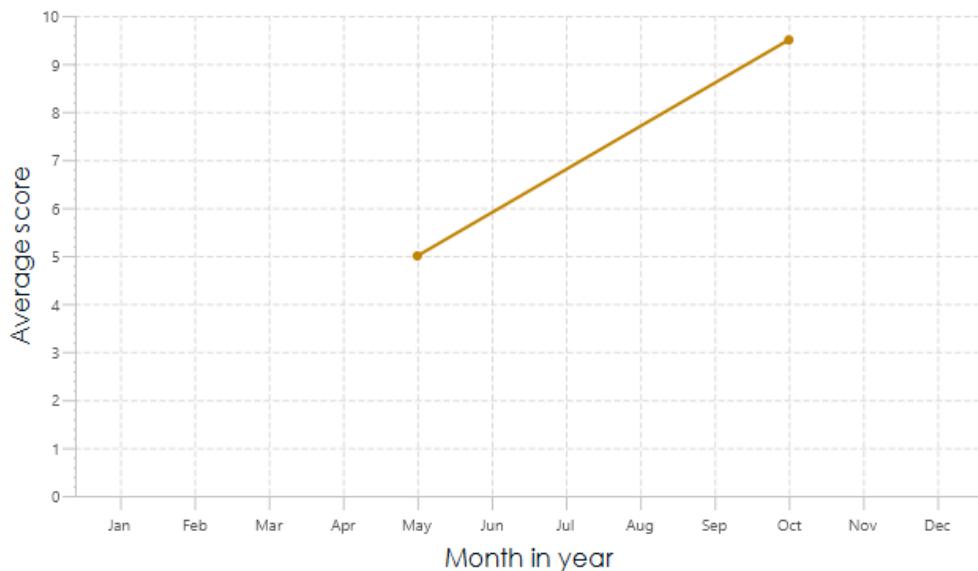
//yearly graph
ResultSet yearlyRs = yearlyStmt.executeQuery();
int lastYear = -1;
double yTotal = 0;
double yFreq = 0;
while(yearlyRs.next()) {
    if(yearlyRs.getInt("ScoreYear") == lastYear) {
        yTotal += yearlyRs.getInt("Highscore");
        yFreq++;
    } else {
        if(lastYear != -1) { //after the first, if the year changes, add the last year's avg scores to the graph
            XYChart.Data<Number, Number> yearPoint = new XYChart.Data<>(lastYear, yTotal/yFreq); //total/frequency is the average score of the year
            yearlySeries.getData().add(yearPoint);
        }
        //reinitialize variables (including first score)
        lastYear = yearlyRs.getInt("ScoreYear"); //method is 0 indexed, db is 1 indexed so -1
        yTotal = yearlyRs.getInt("Highscore");
        yFreq = 1;
    }
}
//at the end, add the final last month (since it won't change to another year after the last year)
XYChart.Data<Number, Number> finalYearPoint = new XYChart.Data<>(lastYear, yTotal/yFreq);
yearlySeries.getData().add(finalYearPoint);

```

asf's All-Time Progress

Daily **Monthly** Yearly

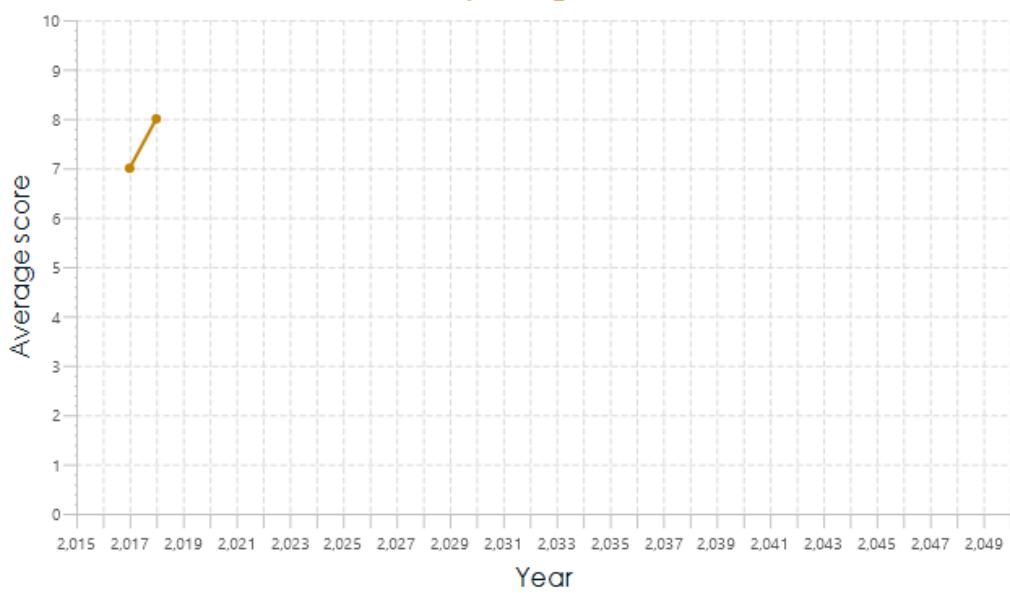
2018 Progress



asf's All-Time Progress

Daily **Monthly** Yearly

Yearly Progress



Code for ReviewController.java:

```
1 package controller;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.util.Calendar;
8
9 import javafx.application.Platform;
10 import javafx.collections.FXCollections;
11 import javafx.fxml;
12 import javafx.scene.chart.CategoryAxis;
13 import javafx.scene.chart.LineChart;
14 import javafx.scene.chart.NumberAxis;
15 import javafx.scene.chart.XYChart;
16 import javafx.scene.layout.AnchorPane;
17 import javafx.scene.text.Text;
18 import other.DAO;
19
20 public class ReviewController {
21     private RootController rootController;
22     private XYChart.Series<Number, Number> dailySeries = new XYChart.Series<>();
23     private XYChart.Series<String, Number> monthlySeries = new XYChart.Series<>();
24     private XYChart.Series<Number, Number> yearlySeries = new XYChart.Series<>();
25     private DAO dao = new DAO();
26
27     @FXML private AnchorPane apnDaily, apnMonthly, apnYearly;
28     @FXML private Text txtProgress;
29
30     public void setRootController(RootController rootController) {
31         this.rootController = rootController;
32     }
33     public void initialize() { //called automatically after fxml injection
34         Platform.runLater(()-> {
35             txtProgress.setText(rootController.getName() + "'s All-Time Progress");
36             getScores();
37             setDailyGraph();
38             setMonthlyGraph();
39             setYearlyGraph();
40         });
41     }
42     public int getDate() {
43         Calendar now = Calendar.getInstance();
44         return now.get(Calendar.DAY_OF_MONTH);
45     }
46     public String getMonth(int num) { //num = -1 for NOW month, 0 <= num <= 11 for inputs
47         Calendar now = Calendar.getInstance();
48         String month = null;
49         switch(num== -1 ? now.get(Calendar.MONTH) : num) {
50             case 0: month = "Jan";
51                 break;
52             case 1: month = "Feb";
53                 break;
54             case 2: month = "Mar";
55                 break;
56             case 3: month = "Apr";
57                 break;
58             case 4: month = "May";
59                 break;
60             case 5: month = "Jun";
61                 break;
62             case 6: month = "Jul";
63                 break;
64             case 7: month = "Aug";
65                 break;
66             case 8: month = "Sep";
67                 break;
68             case 9: month = "Oct";
69                 break;
70             case 10: month = "Nov";
71                 break;
72             case 11: month = "Dec";
73                 break;
74         }
75         return month;
76     }
77     public int getYear() {
78         Calendar now = Calendar.getInstance();
79         return now.get(Calendar.YEAR);
80     }
81     public void setDailyGraph() {
82         //set axes and customise
83         final NumberAxis xAxis = new NumberAxis(1, 31, 1);
```

```

84
85     final NumberAxis yAxis = new NumberAxis(0, 10, 1);
86     xAxis.setLabel("Day of month");
87     xAxis.setMinorTickVisible(false);
88     xAxis.setAutoRanging(false);
89     yAxis.setLabel("Hi-score");
90     yAxis.setMinorTickVisible(false);
91     yAxis.setAutoRanging(false);
92
93     //the line chart object, axes specified in parameters
94     final LineChart<Number, Number> dailyChart = new LineChart<>(xAxis, yAxis);
95     //customise chart
96     dailyChart.setTitle(getMonth(-1) + " Progress");
97     dailyChart.setMaxSize(680, 440);
98     dailyChart.setMinSize(680, 440);
99     dailyChart.setPrefSize(680, 440);
100    dailyChart.setLegendVisible(false);
101    dailyChart.getStylesheets().add("/view/chart.css");
102    //add data for chart
103    dailyChart.getData().add(dailySeries);
104    //add chart to interface
105    apnDaily.getChildren().add(dailyChart);
106}
106@ public void setMonthlyGraph() {
107    final CategoryAxis xAxis = new CategoryAxis(FXCollections.observableArrayList("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"));
108    final NumberAxis yAxis = new NumberAxis(0, 10, 1);
109    xAxis.setLabel("Month in year");
110    xAxis.setAutoRanging(false);
111    yAxis.setLabel("Average score");
112    yAxis.setMinorTickLength(0.25);
113    yAxis.setAutoRanging(false);
114    final LineChart<String, Number> monthlyChart = new LineChart<>(xAxis, yAxis);
115    monthlyChart.setTitle(getYear() + " Progress");
116    monthlyChart.setMaxSize(680, 440);
117    monthlyChart.setMinSize(680, 440);
118    monthlyChart.setPrefSize(680, 440);
119    monthlyChart.setLegendVisible(false);
120    monthlyChart.getStylesheets().add("/view/chart.css");
121    monthlyChart.getData().add(monthlySeries);
122    apnMonthly.getChildren().add(monthlyChart);
123}
123@ public void setYearlyGraph() {
124    final NumberAxis xAxis = new NumberAxis(2015, 2050, 1);
125    final NumberAxis yAxis = new NumberAxis(0, 10, 1);
126    xAxis.setAutoRanging(false);
127    xAxis.setMinorTickVisible(false);
128    xAxis.setLabel("Year");
129    yAxis.setLabel("Average score");
130    yAxis.setMinorTickLength(0.25);
131    yAxis.setAutoRanging(false);
132    final LineChart<Number, Number> yearlyChart = new LineChart<>(xAxis, yAxis);
133    yearlyChart.setTitle("Yearly Progress");
134    yearlyChart.setMaxSize(680, 440);
135    yearlyChart.setMinSize(680, 440);
136    yearlyChart.setPrefSize(680, 440);
137    yearlyChart.setLegendVisible(false);
138    yearlyChart.getStylesheets().add("/view/chart.css");
139    yearlyChart.getData().add(yearlySeries);
140    apnYearly.getChildren().add(yearlyChart);
141}
141@ public void getScores() {
142    //set series data
143    try{
144        Connection conn = dao.getConnection();
145        PreparedStatement dailyStmt = conn.prepareStatement("SELECT Highscore, ScoreDay FROM Score WHERE ScoreMonth = ? AND ScoreYear = ?");
146        PreparedStatement monthlyStmt = conn.prepareStatement("SELECT Highscore, ScoreMonth FROM Score WHERE ScoreYear = ? ORDER BY ScoreMonth");
147        PreparedStatement yearlyStmt = conn.prepareStatement("SELECT Highscore, ScoreYear FROM Score ORDER BY ScoreYear");
148    } {
149        //daily graph
150        Calendar now = Calendar.getInstance();
151        dailyStmt.setInt(1, now.get(Calendar.MONTH)+1);
152        dailyStmt.setInt(2, getYear());
153        ResultSet dailyRs = dailyStmt.executeQuery();
154        while(dailyRs.next()) {
155            XYChart.Data<Number, Number> dayPoint = new XYChart.Data<>(dailyRs.getInt("ScoreDay"), dailyRs.getInt("Highscore"));
156            dailySeries.getData().add(dayPoint);
157        }
158
159        //monthly graph
160        monthlyStmt.setInt(1, getYear());
161        ResultSet monthlyRs = monthlyStmt.executeQuery();
162        String lastMonth = "";
163        double mTotal = 0;
164        double mFreq = 0;
165        while(monthlyRs.next()) {
166            if(getMonth(monthlyRs.getInt("ScoreMonth"))-1).equals(lastMonth)) {
167                mTotal += monthlyRs.getInt("Highscore");
168                mFreq++;
169            } else {
170                if(!lastMonth.equals("")) { //after the first, if the month changes, add the last month's avg scores to the graph
171                    XYChart.Data<String, Number> monthPoint = new XYChart.Data<>(lastMonth, mTotal/mFreq); //total/frequency is the average score of the month
172                    monthlySeries.getData().add(monthPoint);
173                }
174            }
175        }
176    }
177}

```

```

176     //reinitialize variables (including first score)
177     lastMonth = getMonth(monthlyRs.getInt("ScoreMonth")-1); //method is 0 indexed, db is 1 indexed so -1
178     mTotal = monthlyRs.getInt("Highscore");
179     mFreq = 1;
180 }
181 }
182 //at the end, add the final last month (since it won't change to another month after the last month)
183 XYChart.Data<String, Number> finalMonthPoint = new XYChart.Data<>(lastMonth, mTotal/mFreq);
184 monthlySeries.getData().add(finalMonthPoint);
185
186 //yearly graph
187 ResultSet yearlyRs = yearlyStmt.executeQuery();
188 int lastYear = -1;
189 double yTotal = 0;
190 double yFreq = 0;
191 while(yearlyRs.next()) {
192     if(yearlyRs.getInt("ScoreYear") == lastYear) {
193         yTotal += yearlyRs.getInt("Highscore");
194         yFreq++;
195     } else {
196         if(lastYear != -1) { //after the first, if the year changes, add the last year's avg scores to the graph
197             XYChart.Data<Number, Number> yearPoint = new XYChart.Data<>(lastYear, yTotal/yFreq); //total/frequency is the average score of the year
198             yearlySeries.getData().add(yearPoint);
199         }
200         //reinitialize variables (including first score)
201         lastYear = yearlyRs.getInt("ScoreYear"); //method is 0 indexed, db is 1 indexed so -1
202         yTotal = yearlyRs.getInt("Highscore");
203         yFreq = 1;
204     }
205 }
206 //at the end, add the final last month (since it won't change to another year after the last year)
207 XYChart.Data<Number, Number> finalYearPoint = new XYChart.Data<>(lastYear, yTotal/yFreq);
208 yearlySeries.getData().add(finalYearPoint);
209
210 } catch (SQLException e) {
211     e.printStackTrace();
212 }
213 }
214 }

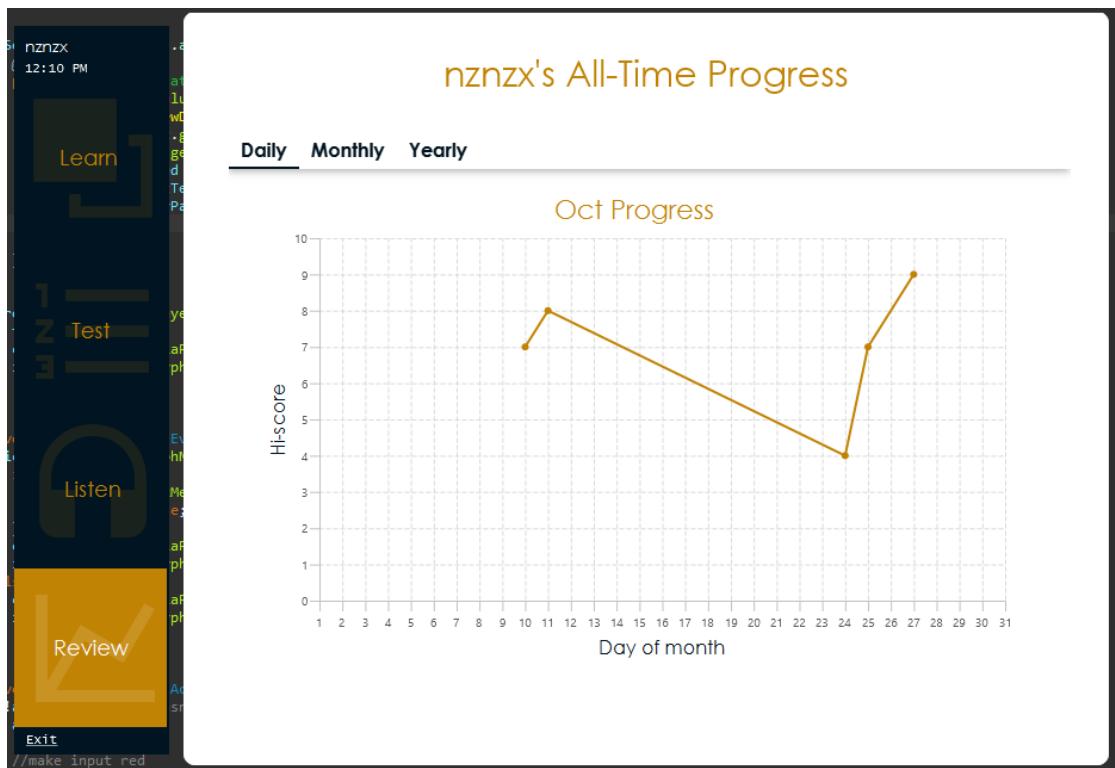
```

Validation Test

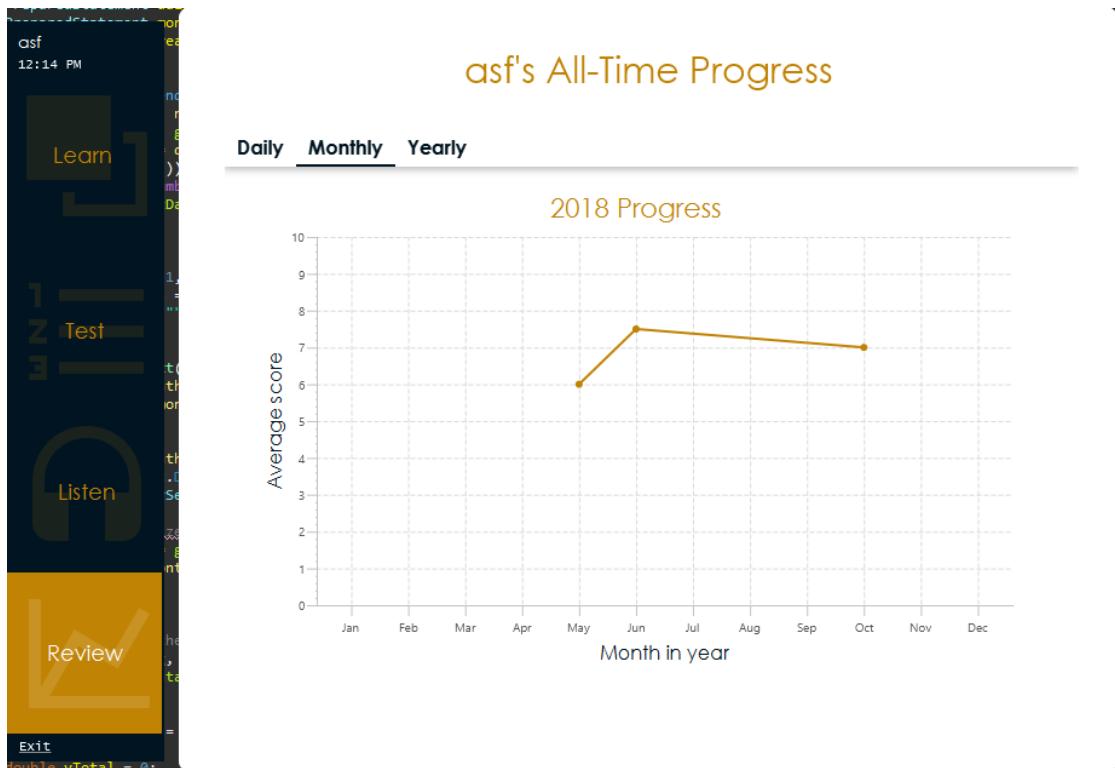
To validate our final section, we can add a few more data points to our database, table Score.

ID	ScoreDay	ScoreMonth	ScoreYear	Highscore	Click to Add
1	9	10	2017	8	
4	10	10	2018	7	
5	11	10	2018	8	
13	12	5	2018	6	
14	24	10	2018	4	
15	25	10	2018	7	
16	27	10	2018	9	
17	13	6	2018	7	
18	15	6	2018	8	
*	(New)	0	0	0	

We can check all three graphs using these points effectively.

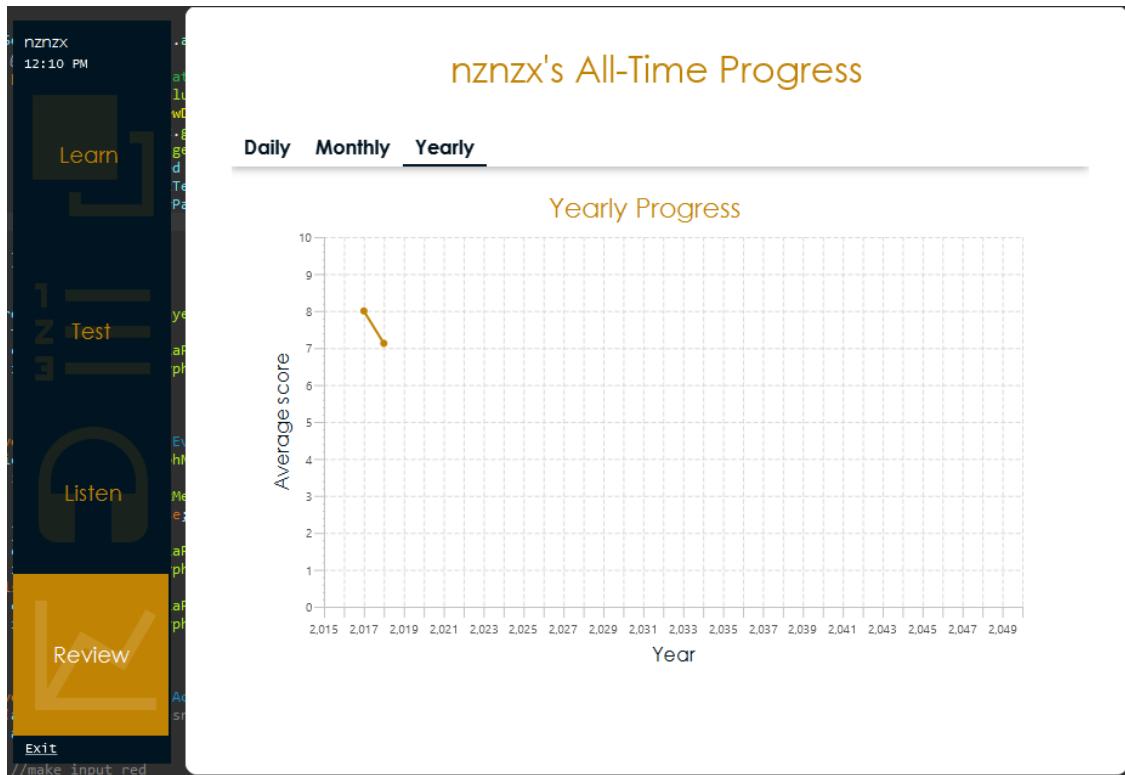


To check our algorithm which calculates mean, the mean for October should be $(8+7+8+4+7+9)/6 = 7.16$; the mean for May should be simply 6; the mean for June should be $(8+7)/2 = 7.5$.



All three points are correctly calculated and displayed.

For the yearly graph, the two separate years have averages correctly calculated and displayed (which is $56/8 = 7$ for 2018 and simply 8 for 2017). Note that the 2017 data point has not affected any other graph's data points.



Overall, a little mathematical, but this gives a more general outlook on one's progress and allows them to view their improvements as a bigger picture. This Review section is a good addition to our system.

Chart.css

In our views and other UI controls, we have applied some CSS to make it look more modern and employ material design. For the graph, we have edited the chart-title and axes with basic text properties.

```

1  .chart-title {
2      -fx-text-fill: #c18303;
3      -fx-font-size: 22px;
4      -fx-font-family: 'Century Gothic';
5  }
6  .axis-label {
7      -fx-text-fill: #001421;
8      -fx-font-family: 'Century Gothic';
9      -fx-font-size: 16px;
10 }
11 .axis {
12     -fx-text-fill: #001421;
13     -fx-font-family: 'Century Gothic';
14     -fx-font-size: 12px;
15 }
```

We can also add padding and transparency to make our graph look more spacious and seemingly calm to look at.

```
10 .chart-title {  
11     -fx-text-fill: #c18303;  
12     -fx-font-size: 22px;  
13     -fx-font-family: 'Century Gothic';  
14 }  
15 .axis-label {  
16     -fx-text-fill: #001421;  
17     -fx-font-family: 'Century Gothic';  
18     -fx-font-size: 16px;  
19 }  
20 .axis {  
21     -fx-text-fill: #001421;  
22     -fx-font-family: 'Century Gothic';  
23     -fx-font-size: 12px;  
24 }
```

Also, it is possible to change the line and data points themselves so that they match the program colour scheme and be thinner so that it is not confusing to look at – material design.

```
27 .chart-series-line {  
28     -fx-stroke-width: 2px;  
29     -fx-effect: null;  
30 }  
31 .chart-line-symbol {  
32     -fx-background-color: #c18303;  
33     -fx-background-radius: 5px;  
34     -fx-padding: 3px;  
35 }  
36 .default-color0.chart-line-symbol { -fx-background-color: #c18303; }  
37 .default-color0.chart-series-line { -fx-stroke: #c18303; }
```

.default-color0.chart refers to our 0th (first) line on the graph – you can have multiple lines on a graph so that is why an index is required.

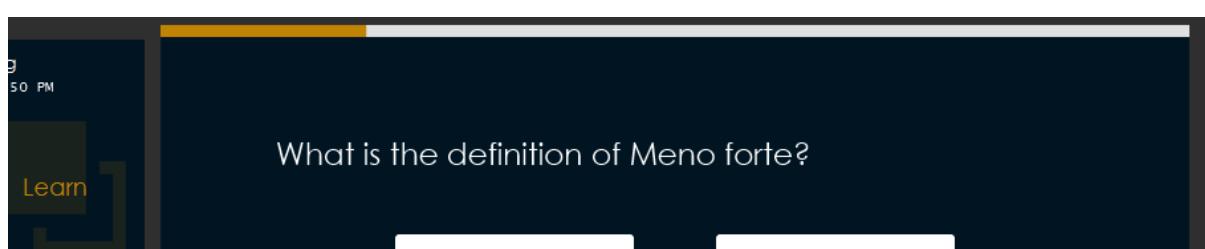
Code for Chart.css:

```
1● .chart-title {
2     -fx-text-fill: #c18303;
3     -fx-font-size: 22px;
4     -fx-font-family: 'Century Gothic';
5 }
6● .axis-label {
7     -fx-text-fill: #001421;
8     -fx-font-family: 'Century Gothic';
9     -fx-font-size: 16px;
10}
11● .axis {
12     -fx-text-fill: #001421;
13     -fx-font-family: 'Century Gothic';
14     -fx-font-size: 12px;
15}
16● .chart {
17     -fx-padding: 20px;
18}
19● .chart-plot-background {
20     -fx-background-color: transparent;
21}
22● .chart-alternative-row-fill {
23     -fx-fill: transparent;
24     -fx-stroke: transparent;
25     -fx-stroke-width: 0;
26}
27● .chart-series-line {
28     -fx-stroke-width: 2px;
29     -fx-effect: null;
30}
31● .chart-line-symbol {
32     -fx-background-color: #c18303;
33     -fx-background-radius: 5px;
34     -fx-padding: 3px;
35}
36 .default-color0.chart-line-symbol { -fx-background-color: #c18303; }
37 .default-color0.chart-series-line { -fx-stroke: #c18303; }
```

Also, we have used other CSS files to style other nodes to make them look more modern and clean.

Progressbar.css

This was used to style our progress bar of our test.



Code for progressbar.css:

```
10 .jfx-progress-bar > .bar {  
11     -fx-background-color: #c18303  
12 }  
13
```

Slider.css

This stylesheet has styled our slider in the listen section.



Using Jfoenix's documentation and his own defined properties, we can edit the colours of the track (coloured and un-coloured), the thumb (the circle which you can click and drag to change the position of the slider) , the animated-thumb (which shows the percentage of the slider above it), and the slider-value which lies inside the animated-thumb.

Code for Slider.css:

```
10 .jfx-slider .track {  
11     -fx-background-color: #ffffff;  
12 }  
13 .jfx-slider .thumb {  
14     -fx-background-color: #c18303;  
15 }  
16 .jfx-slider .colored-track {  
17     -fx-background-color: #c18303  
18 }  
19 .jfx-slider .animated-thumb {  
20     -fx-background-color: #001421;  
21 }  
22 .jfx-slider .slider-value {  
23     -fx-stroke: #c18303;  
24 }
```

Tabpane.css

A stylesheet used to style the JFXTabPane in the Review section.

Daily Monthly Yearly

We styled the line under the separate tabs, the line under the selected tab and the text within each tab.

Code for Tabpane.css:

```
1. .jfx-tab-pane .tab-header-area .tab-selected-line {  
2.     -fx-background-color: #001421;  
3. }  
4.  
5. .jfx-tab-pane .tab-header-area .tab-header-background {  
6.     -fx-background-color: #ffffff;  
7. }  
8.  
9. .jfx-tab-pane .tab-label {  
10.    -fx-text-fill: #001421;  
11.    -fx-font-family: 'Century Gothic';  
12. }
```

Scollpane.css

This was used to style the scrollpanes of learn.fxml and learn_update.fxml.



sample

Add New Flashcard Finish

-	Foreign Term	Definition
-	Foreign Term	Definition

Now, it would seem best to use Jfoenix's controls to get all the styles pre-made in the control, however, there is a slight issue in this external library in that the JFXScrollPane does not allow us to scroll, not with scroll bar nor with scroll wheel. when it is containing a child node which grows in size (our VBox in this case).



So instead, we can use a normal ScrollPane provided by the JavaFX standard library, and then style it similar to JFoenix.



Your Flashcard Sets

Articulation

Tempo

Dynamics

Chords

Modulations

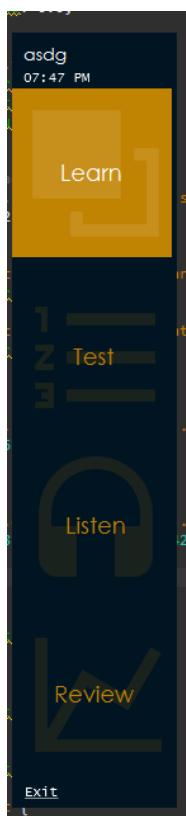
Harmony

Time Signatures

Clefs

Key Signatures

Create New



Your Flashcard Sets

Articulation

Tempo

Dynamics

Chords

Modulations

Harmony

Time Signatures

Clefs

Key Signatures

Create New

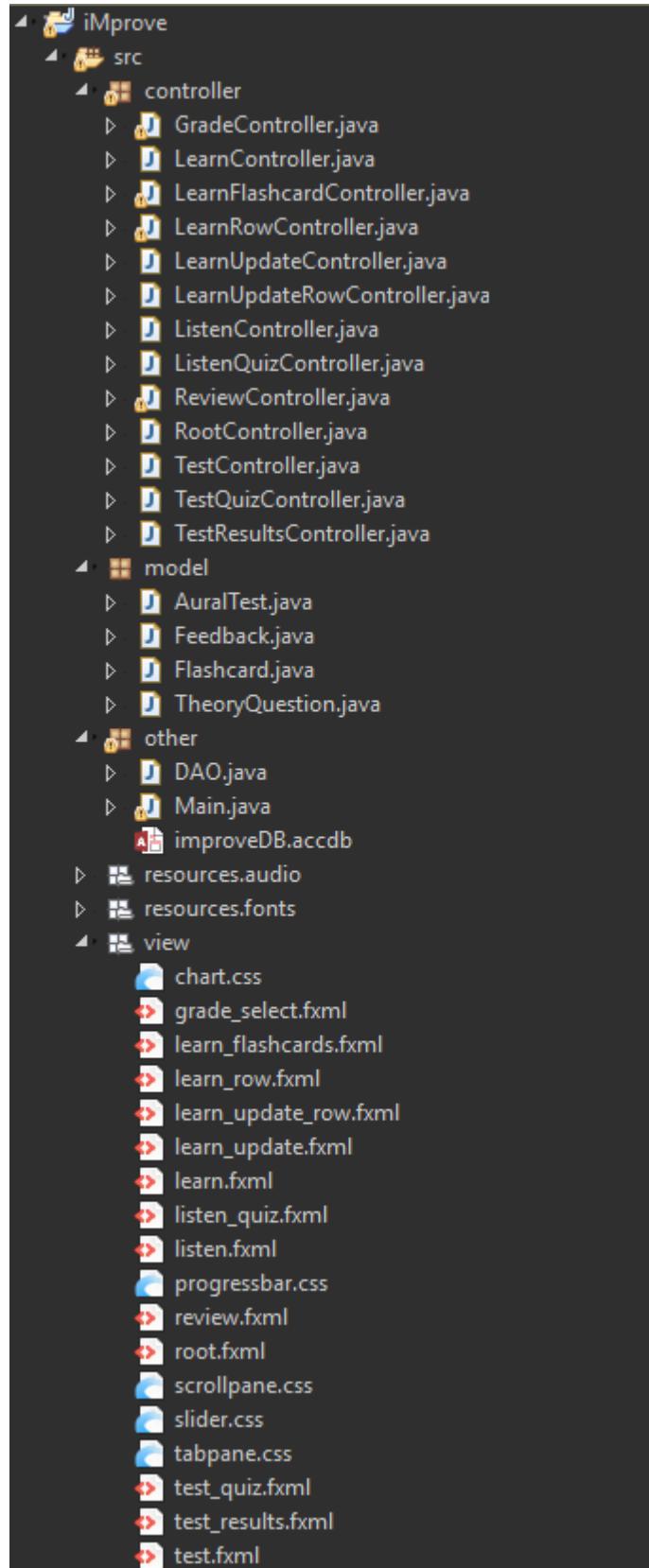


Code for scrollpane.css:

```
1 /*scroll bar track */
2 .scroll-bar:vertical > .track-background{
3     -fx-background-color: #FFFFFF;
4     -fx-background-insets: 0 0;
5 }
6 /*dragging thumb */
7 .scroll-bar:vertical > .thumb {
8     -fx-background-color: #c18303;
9     -fx-background-insets: 0 0;
10    -fx-background-radius: 1.0;
11 }
12
13 /* up and down button padding */
14 .scroll-bar:vertical > .increment-button, .scroll-bar:vertical > .decrement-button {
15     -fx-padding: 5 2 5 2;
16 }
17 /*up down buttons*/
18 .scroll-bar > .increment-button, .scroll-bar > .decrement-button, .scroll-bar:hover > .increment-button, .scroll-bar:hover > .decrement-button {
19     -fx-background-color: transparent;
20 }
21 .scroll-bar > .increment-button > .increment-arrow, .scroll-bar > .decrement-button > .decrement-arrow {
22     -fx-background-color: rgb(43, 80, 104);
23 }
24
25 /* up arrow shape */
26 .scroll-bar:vertical > .increment-button > .increment-arrow {
27     -fx-shape: "M298 426h428l-214 214z";
28 }
29
30 /* down arrow shape */
31 .scroll-bar:vertical > .decrement-button > .decrement-arrow {
32     -fx-shape: "M298 598l214-214 214 214h-428z";
33 }
34
35 /*scroll pane itself */
36 .scroll-pane {
37     -fx-background-insets: 0;
38     -fx-padding: 0;
39 }
40 .scroll-pane:focused {
41     -fx-background-insets: 0;
42 }
43 .scroll-pane .corner {
44     -fx-background-insets: 0;
45 }
46 .scroll-pane > .viewport {
47     -fx-background-color: transparent;
48 }
```

Summary Review

Our project hierarchy currently looks like this:



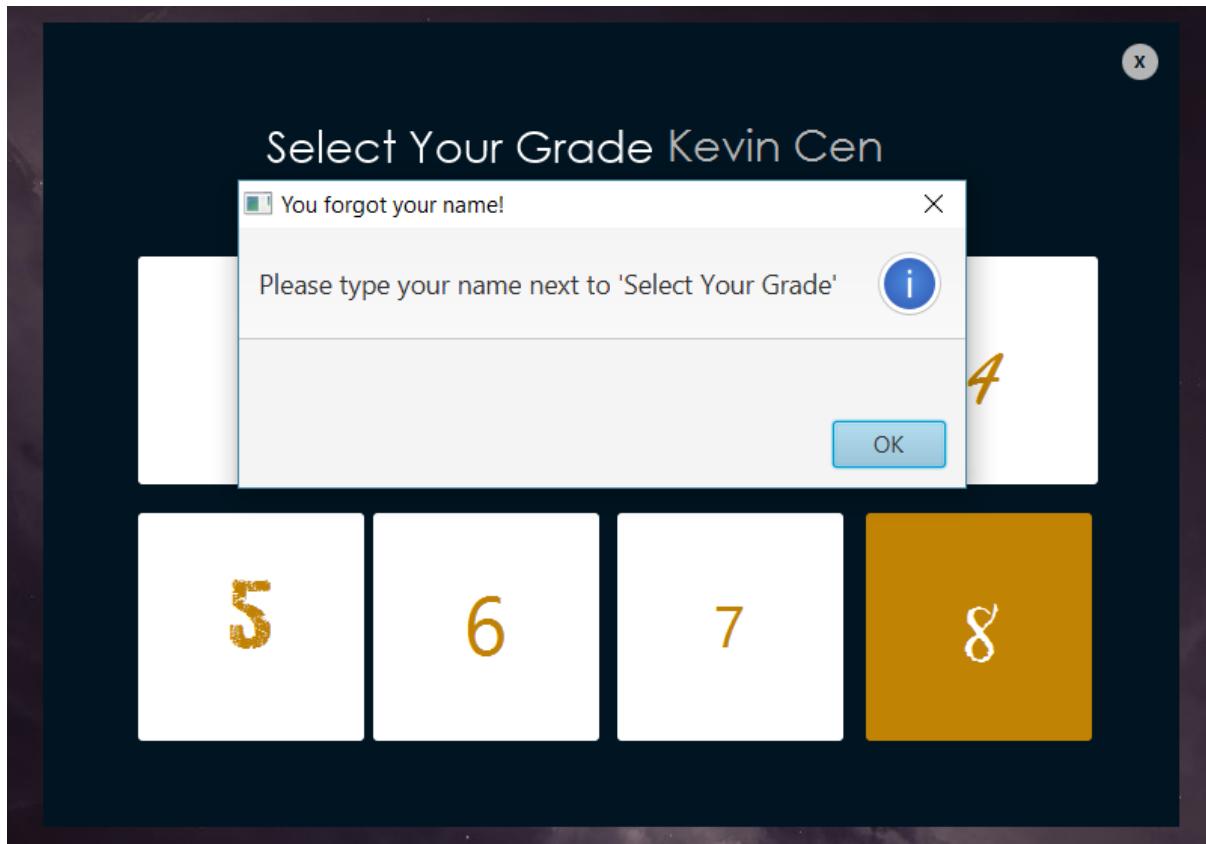
This has separated all the classes and files into separate folders depending on their relevance and makes the directory more clear. Also, resources.audio will contain all the audio files for separate aural tests in the listen feature and the resources.fonts currently contains all the custom fonts used in this program.

Our first iteration of the iMProve system has finally been made and this iteration provides a structure and a hint for what the system can be for our final system.

End of Iteration Validation Test

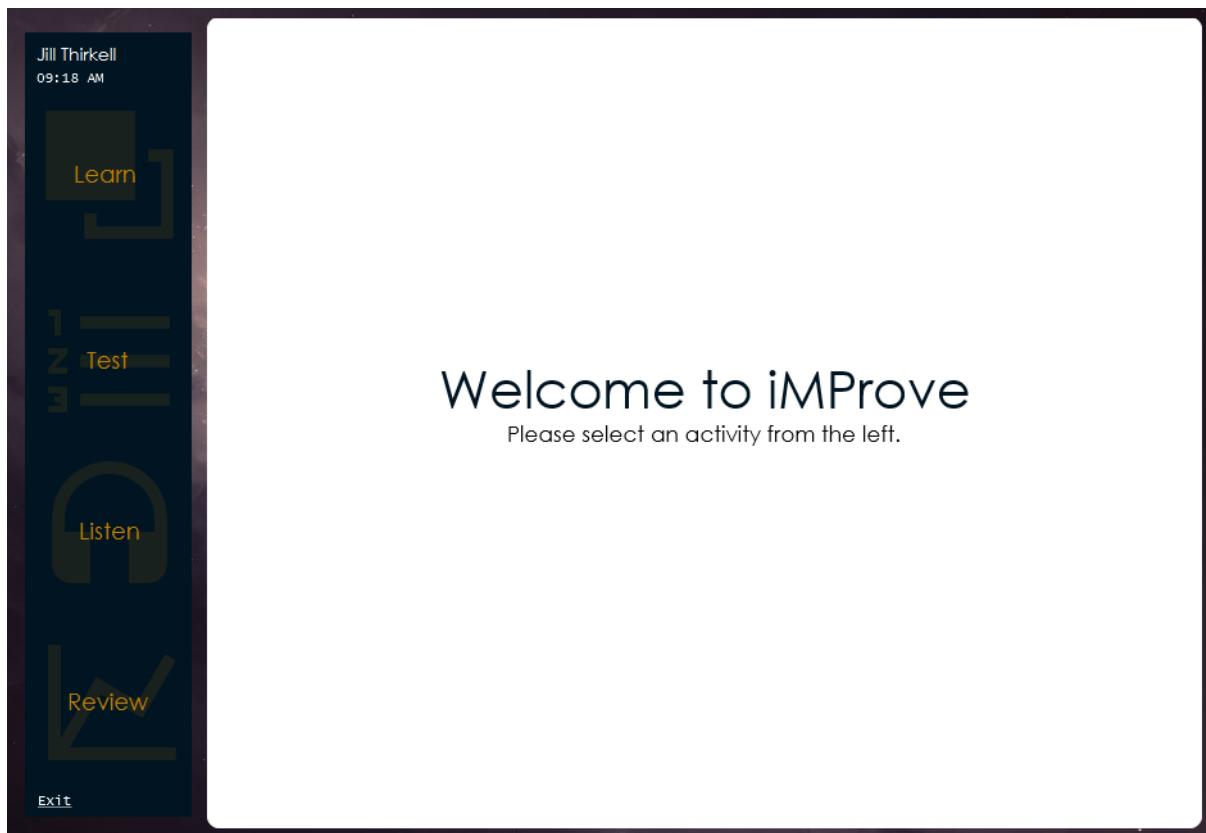
At this point, we can give our program to our user to review our progress and see what improvements can be made. Jill Thirkell was happy to test it with me.

On initial load up, the grade selection screen was displayed however she got very confused with inputting the name.



This could be a possible change that could be made.

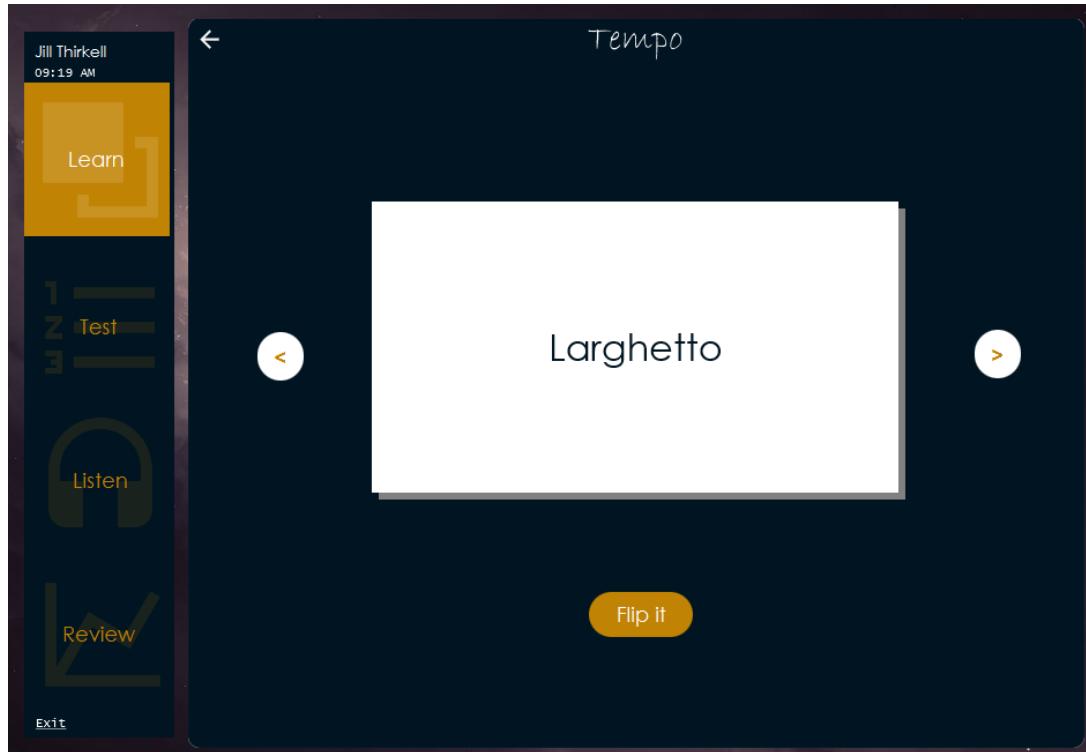
After informing her where she should enter her name, she entered the system, the welcome screen was displayed.



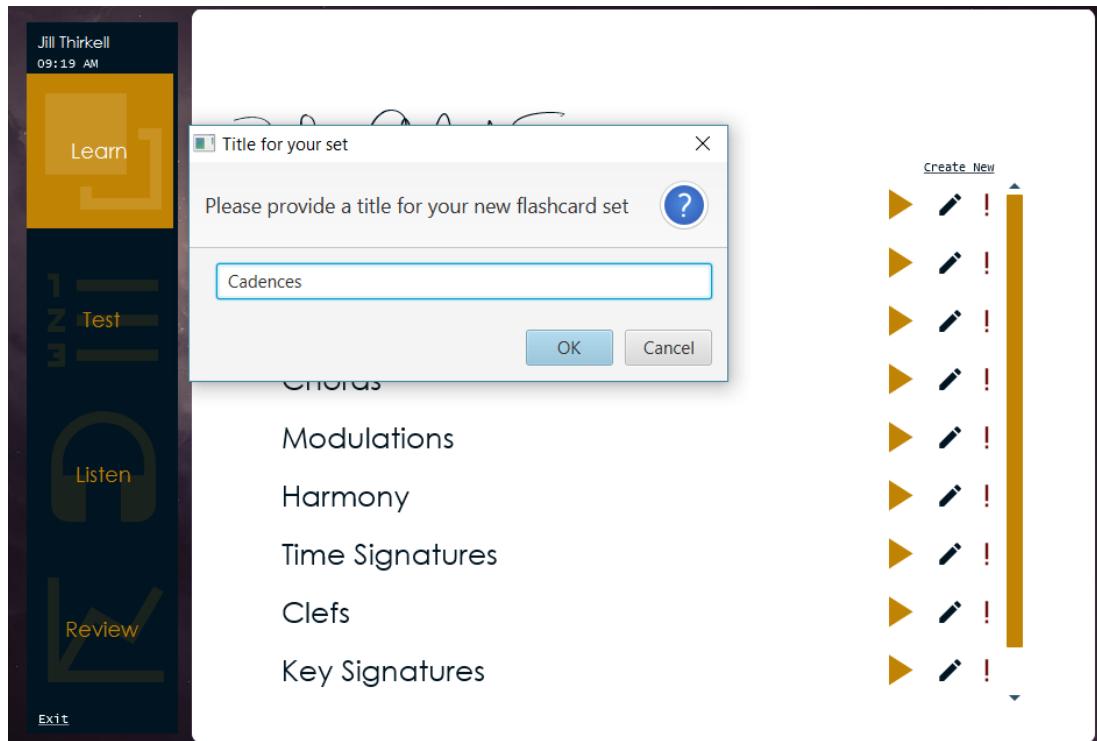
The instructions below the title was clear and she understood how to navigate throughout the program. We entered the Learn section.

This screenshot shows the "Learn" section of the iMProve program. The vertical sidebar on the left remains the same, with "Learn" highlighted. The main content area now displays a list titled "Your Flashcard Sets" in a stylized, handwritten font. To the right of the list is a vertical column of icons, each consisting of a yellow triangle pointing up, a pencil icon, and a red exclamation mark. A small blue link labeled "Create New" is located at the top of this column. The list of flashcard sets includes: Articulation, Tempo, Dynamics, Chords, Modulations, Harmony, Time Signatures, Clefs, and Key Signatures. The "Create New" link is positioned above the first icon in the column.

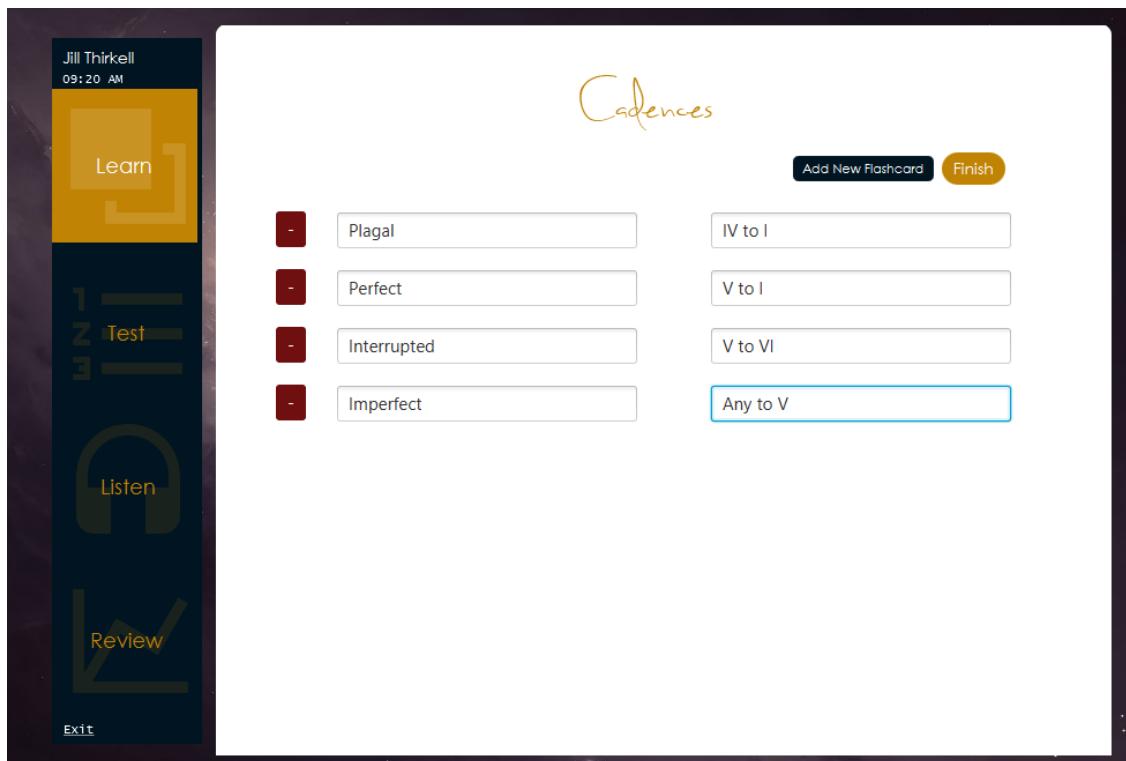
Trying to play one of the flashcard sets, we chose Tempo.



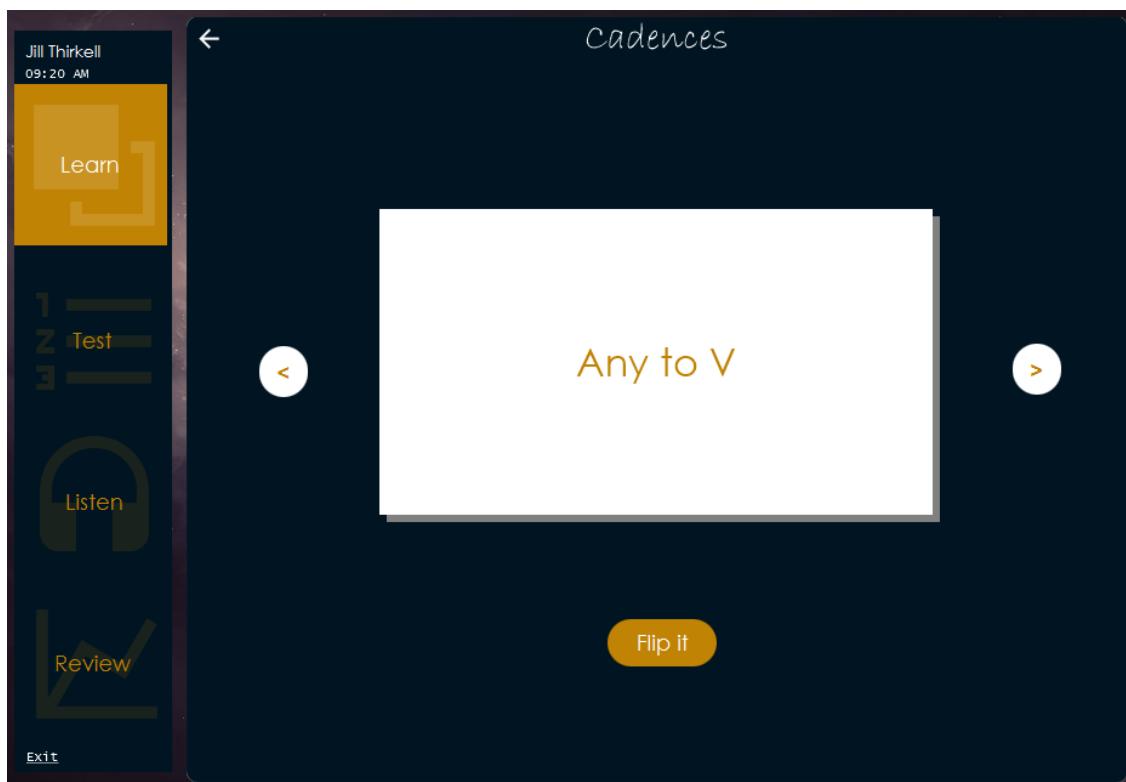
She found it easy to navigate and approved on how simply set out the interface was After exiting the flashcards, she tried to create her own set.



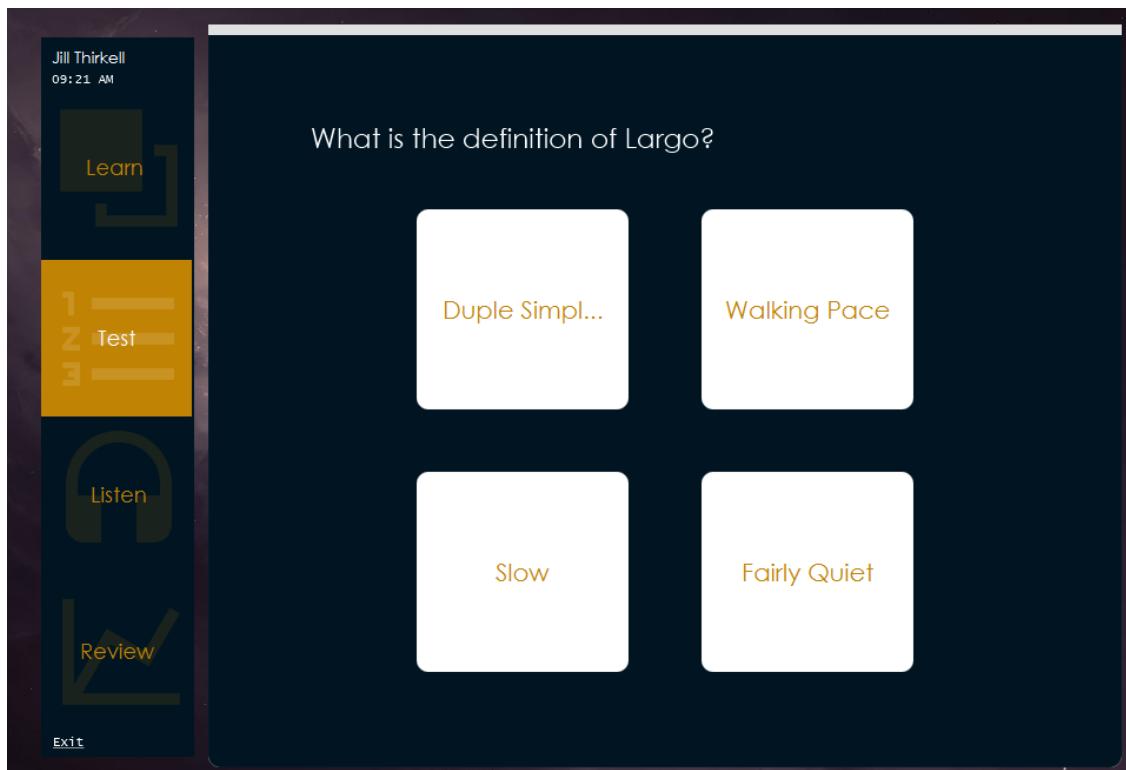
At first, she was confused to click on add new flashcard, so we should probably add a few rows initially.



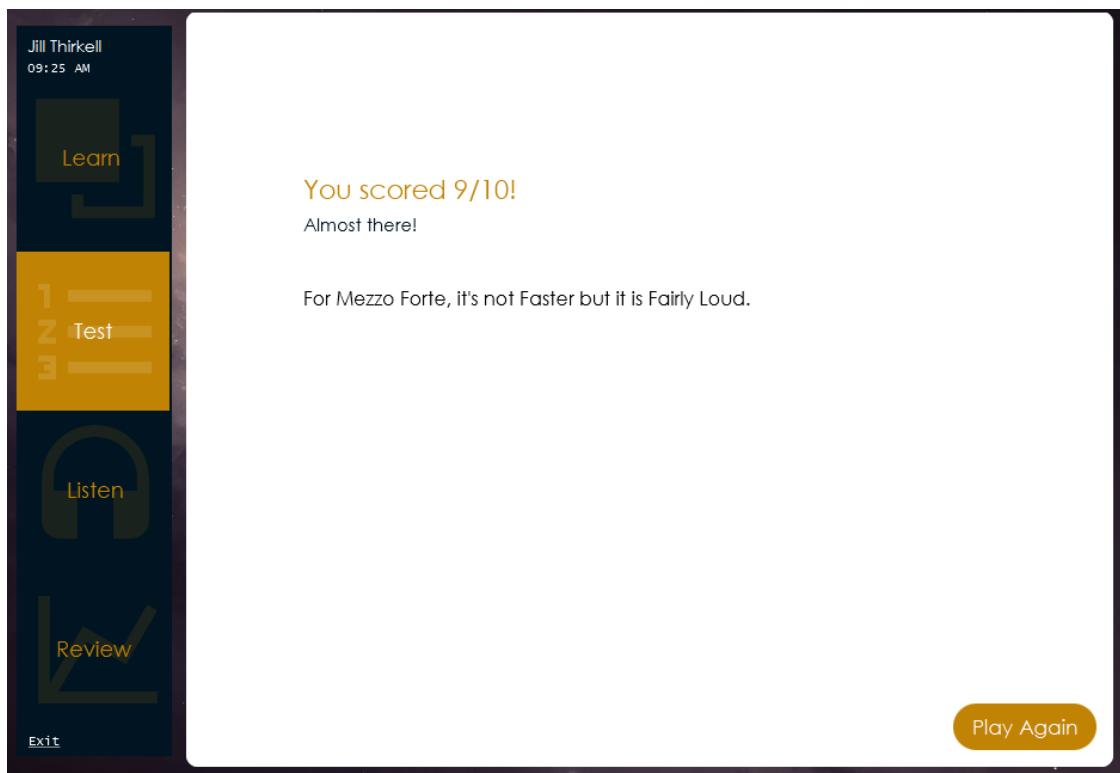
She tried pressing the enter key multiple times thinking her keyboard was broken until I told her to press the finish button. Then, Ms. Thirkell played the new flashcard set she has made and found the customisability very promising.



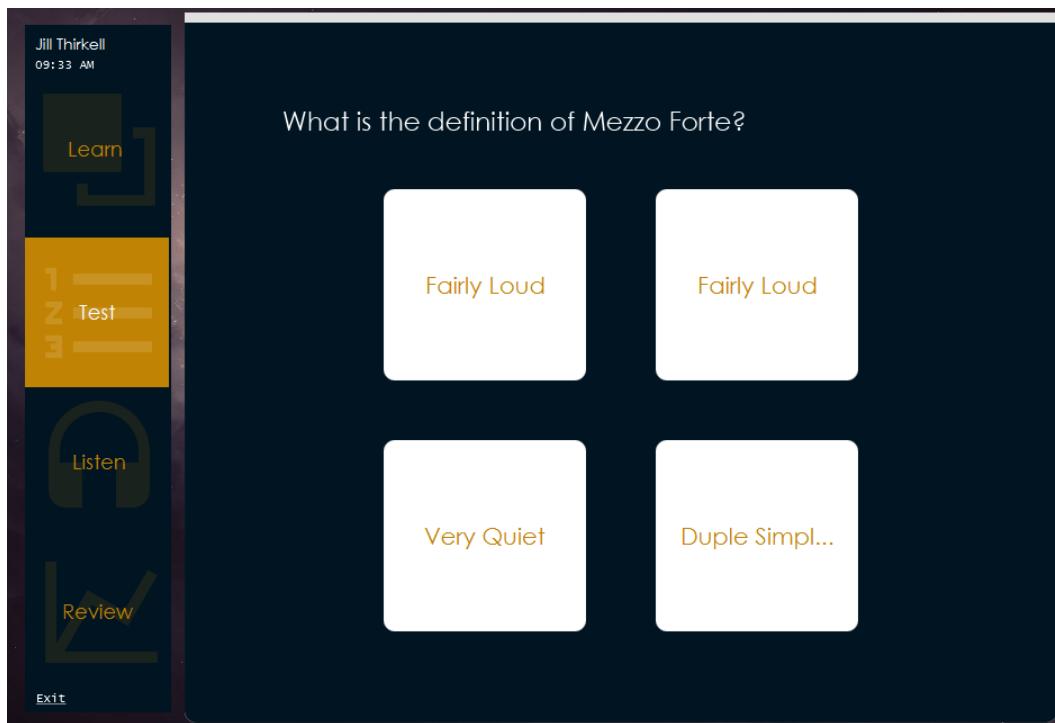
Jill proceeded to try the Test feature and noticed a term was very long and did not get displayed – this is something to be aware of for next iteration.



We made her purposely enter the wrong answer for one of the questions and it was displayed as feedback in the results screen successfully.

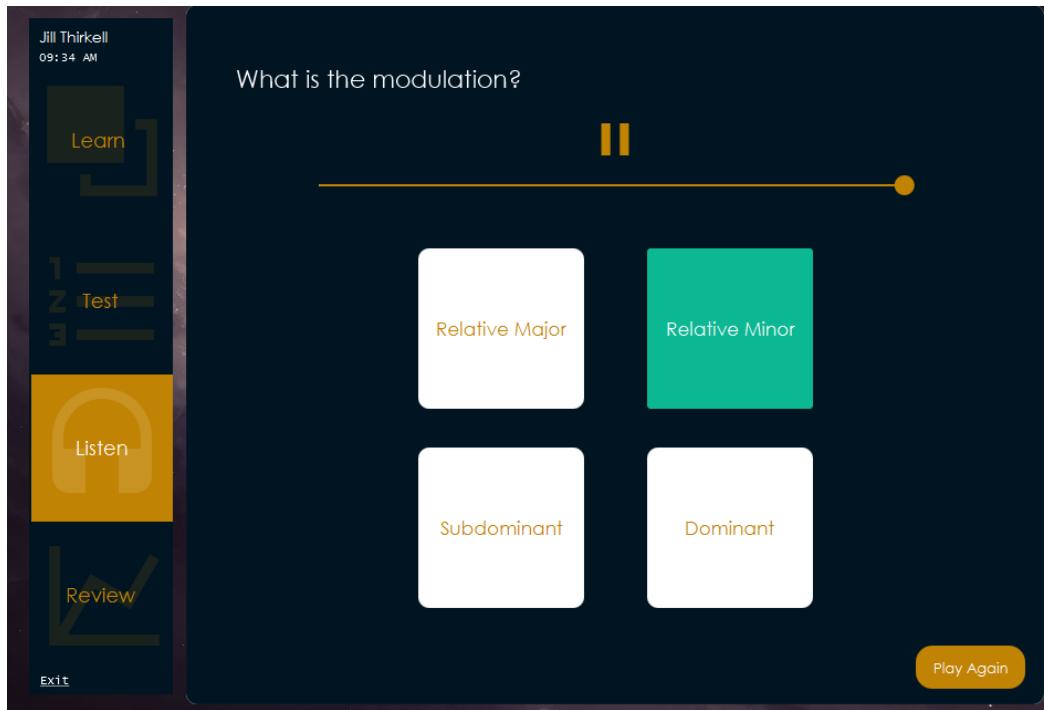


Then on the next test, it was the term displayed first.

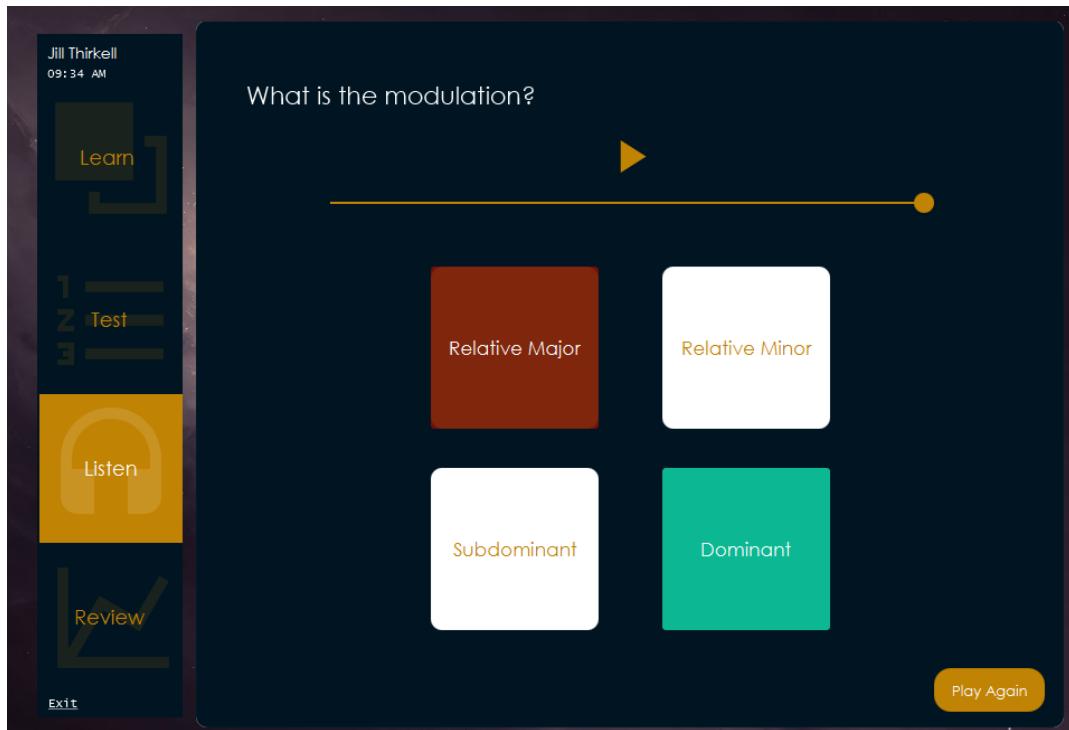


Adapting the tests has been successful.

Next is the Listen section.



The interface was easy to navigate and found the music to be easy to control (seek, play and pause).



Also, she found the feedback easy to understand and quick to improve on. However, she mentioned that in a real aural test exam, you could only listen to it once before answering, she suggested that the student can look back only after they answered the question. Also, she picked up that this was only Grade 8 level which I forgot to state in the program. This is both valuable advice for our next iteration of our program.

Including a dummy data point in May, she liked the idea of having graphs for users to feel motivated to continue improving and was pleased with how it could motivate users to learn.



However, at the end, she asked if she could put the system back to its original state so that her changes would be removed, and all her graphs were cleaned. Jill felt that if over time, the flashcard sets could get too congested which is not pleasing for the eye, or the adapted test only focused on the same term even though the user had learnt in outside of the program, then the program would be not as effective as it could be. This is a good idea we can implement for our next iteration, possibly a settings page.

Overall, she was extremely happy about the quality of the system, but we discovered a few more improvements we can focus on for next iteration:

- Easier name input/login
- Enter key functionality
- Add initial flashcards
- Button text not overrun
- Changing Listen quiz structure
- Settings page including reset data

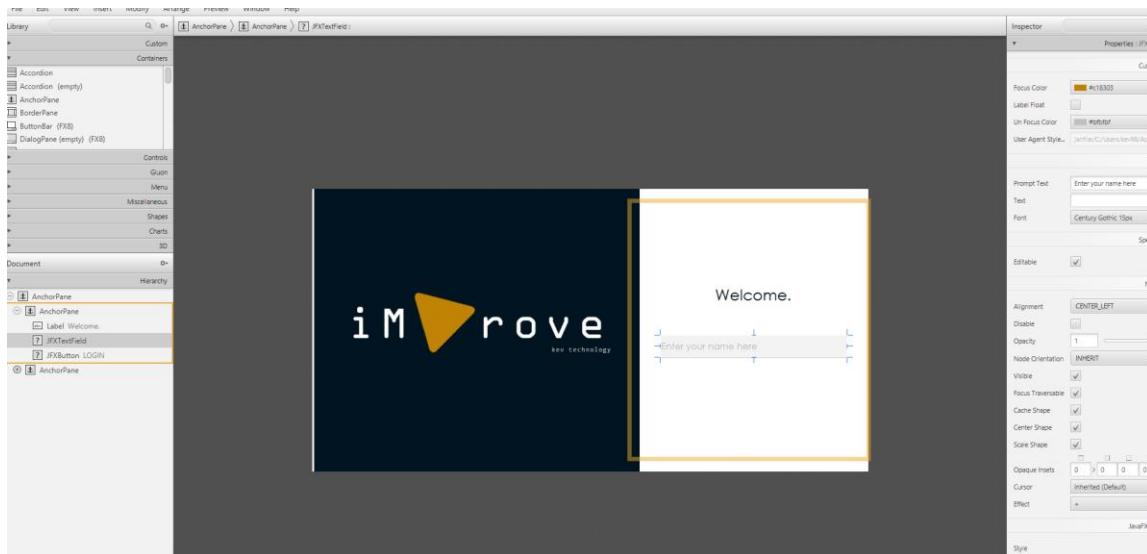
Iteration 2

This iteration we want to focus on fixing some small errors or ideas that we missed in our last iteration. Improvements from our stakeholder will be our priority here, but there still may be others we can introduce to make our program more efficient, effective, and overall, much easier to use.

LoginController.java

So firstly, to simplify the login, we can make a two-step process to begin the session. Our first step will be to retrieve our user's name using a view, `login.fxml`, then the grade using our previous `GradeController` and `grade_select.fxml`.

We can design `login.fxml` using Scene Builder to achieve this view:



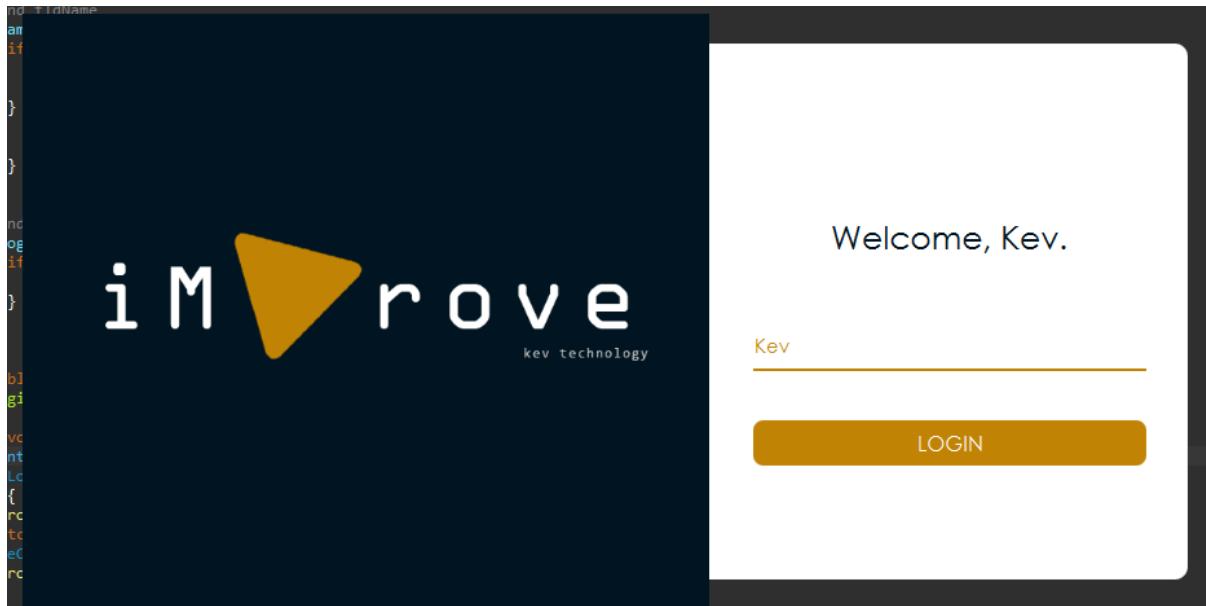
Since we are loading the login first rather than the grade_select, we need to edit the Main.java class ever so slightly so that it loads up login.fxml rather than grade_select.fxml.

```
@Override
public void start(Stage stage) throws Exception {
    loadFonts();
    mainStage = stage;

    Parent root = FXMLLoader.load(getClass().getResource("/view/login.fxml"));
    Scene loginScene = new Scene(root);
    loginScene.setFill(null);
    stage.initStyle(StageStyle.TRANSPARENT);
    stage.setScene(loginScene);
    stage.show();
}
```

For our LoginController, we will have a button which will display only when a name has been typed in, saving our use of an Alert dialog; we can also change the Welcome text and append the name for added effect,

```
public void initialize() {
    //bind fldName
    fldName.textProperty().addListener((o, ov, nv) -> {
        if(nv.isEmpty()) {
            btnLogin.setVisible(false);
            lblWelcome.setText("Welcome.");
        } else {
            btnLogin.setVisible(true);
            lblWelcome.setText("Welcome, " +nv+ ".");
        }
    });
}
```



The JFXTextField and JFXButton will have they onAction binded to a method called login(). We also want to use the enter key, if the user has clicked off the control, to still enter in – expands on enter key functionality. So, we should use another method called doLogin() and get login() to just refer back to doLogin(). We cannot directly call login() with the enter key since login() requires an ActionEvent parameter to handle the ActionEvent. First of all, handling the enter key uses a KeyEvent, secondly, we are setting the event handler inside our code rather than binding it in our fxml, so we must not call a @FXML annotated method within our controller.

```
//bind keys
apnLoginRoot.setOnKeyPressed(e -> {
    if(e.getCode()==KeyCode.ENTER) {
        doLogin();
    }
});
```

```
@FXML public void login(ActionEvent ae) {
    doLogin();
}
private void doLogin() {
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/grade_select.fxml"));
    try {
        root = loader.load();
    } catch(IOException e) {e.printStackTrace();}
    GradeController controller = loader.getController();
    controller.setName(fldName.getText());

    Scene gradeScene = new Scene(root);
    gradeScene.setFill(null);
    Main.getStage().setScene(gradeScene);
}
```

Our doLogin() method simply loads up our grade_select.fxml and gives the GradeController the retrieved name – meaning GradeController doesn't need to get the name.

Code for LoginController.java:

```
1 package controller;
2
3 import java.io.IOException;
4
5 import com.jfoenix.controls.JFXButton;
6 import com.jfoenix.controls.JFXTextField;
7
8 import javafx.event.ActionEvent;
9 import javafx.fxml.FXML;
10 import javafx.fxml.FXMLLoader;
11 import javafx.scene.Parent;
12 import javafx.scene.Scene;
13 import javafx.scene.control.Label;
14 import javafx.scene.input.KeyCode;
15 import javafx.scene.layout.AnchorPane;
16 import other.Main;
17
18 public class LoginController {
19     @FXML private JFXTextField fldName;
20     @FXML private Label lblWelcome;
21     @FXML private AnchorPane apnLoginRoot;
22     @FXML private JFXButton btnLogin;
23     public void initialize() {
24         //bind fldName
25         fldName.textProperty().addListener((o, ov, nv) -> {
26             if(nv.isEmpty()) {
27                 btnLogin.setVisible(false);
28                 lblWelcome.setText("Welcome..");
29             } else {
30                 btnLogin.setVisible(true);
31                 lblWelcome.setText("Welcome, "+nv+" .");
32             }
33         });
34
35         //bind keys
36         apnLoginRoot.setOnKeyPressed(e -> {
37             if(e.getCode()==KeyCode.ENTER) {
38                 doLogin();
39             }
40         });
41     }
42
43     @FXML public void login(ActionEvent ae) {
44         doLogin();
45     }
46     private void doLogin() {
47         Parent root = null;
48         FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/grade_select.fxml"));
49         try {
50             root = loader.load();
51         } catch(IOException e) {e.printStackTrace();}
52         GradeController controller = loader.getController();
53         controller.setName(fldName.getText());
54
55         Scene gradeScene = new Scene(root);
56         gradeScene.setFill(null);
57         Main.getStage().setScene(gradeScene);
58     }
59 }
```

Learn Adjustments



For our LearnUpdateController, we can bind the `onKeyPressed()` of the root AnchorPane, `apnUpdateSet` to also perform the same job as the finish button.

```
@FXML
public void finish(ActionEvent e) {
    doFinish();
}
@FXML
public void kFinish(KeyEvent e) {
    if(e.getCode()==KeyCode.ENTER)
        doFinish();
}
public void doFinish() {
    try {
        Connection conn = dao.getConnection();
        PreparedStatement deleteStmt = conn.prepareStatement("DELETE FROM Theory WHERE Title = ?");
        PreparedStatement stmt = conn.prepareStatement("INSERT INTO Theory (Term, Definition, Grade, Title) VALUES (?, ?, ?, ?)");
        ) {
            //delete old terms
            conn.setAutoCommit(false); //doesn't unpredictably commit the changes
            deleteStmt.setString(1, txtNewTitle.getText());
            deleteStmt.executeUpdate();

            boolean termsAdded = false;
            for(LearnUpdateRowController c: controllerMap.values()) {
                if(!(c.getTerm().equals("")) || c.getDef().equals("")) { //not blank space
                    termsAdded = true;
                    stmt.setString(1, c.getTerm());
                    stmt.setString(2, c.getDef());
                    stmt.setInt(3, RootController.getGrade());
                    stmt.setString(4, txtNewTitle.getText());
                    stmt.addBatch();
                }
            }
            if(termsAdded) //if terms WERE added, then execute batch otherwise will cause error on adding nothing
                stmt.executeBatch();

            conn.commit();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }

        //go back to main learn screen
        rootController.setActivity(apnLearn);
        learnController.initSets();
    }
}
```

Here, we have just moved the previously existing code from `finish()`, called by `btnFinish` which fires an `ActionEvent`, to `doFinish()`. Also, `kFinish()`, called by `apnUpdateSet` which fires a `KeyEvent`, invokes `doFinish()` if the key pressed is the enter key.

On testing this, no errors occurred, everything runs smoothly – was only a small change.

The content produced by the Learn facility should also be relevant to the user, so the fetching queries need only to retrieve the terms that are less or equal than the users grade.

In `LearnController.java` when fetching the titles, we can add an extra condition to the where clause.

```
        Connection conn = dao.getConnection();
        PreparedStatement stmt = conn.prepareStatement("SELECT Title FROM Theory WHERE Grade <= ?");
    ) {
        stmt.setInt(1, RootController.getGrade());
        ResultSet rs = stmt.executeQuery();
```

Similarly, we can repeat this for `LearnFlashcardController.java` when collecting the terms and `LearnUpdateController.java` when deleting the previous terms before adding the new terms (we do not want to remove *all* the terms, only the ones less or equal than the grade).

```
try {
    Connection conn = dao.getConnection();
    PreparedStatement stmt = conn.prepareStatement("SELECT Term, Definition FROM Theory WHERE Title = ? AND Grade <= ?");
} {
    stmt.setString(1, title);
    stmt.setInt(2, RootController.getGrade());

public void doFinish() {
    try {
        Connection conn = dao.getConnection();
        PreparedStatement deleteStmt = conn.prepareStatement("DELETE FROM Theory WHERE Title = ? AND Grade <= ?");
        PreparedStatement stmt = conn.prepareStatement("INSERT INTO Theory (Term, Definition, Grade, Title) VALUES (?, ?, ?, ?)");
    } {
        //delete old terms
        conn.setAutoCommit(false); //doesn't unpredictably commit the changes

        deleteStmt.setString(1, txtNewTitle.getText());
        deleteStmt.setInt(2, RootController.getGrade());
```

Additionally, there is another minor change to be made to the edit/new set page, adding a few initial rows when the user creates a new set. We can do this by calling `LearnUpdateController`'s `addRow()` method multiple times when the user decides to create a new set; inside `LearnController`'s `createSet()` method.

```
@FXML
public void createSet(MouseEvent e) {
    //get the title
    TextInputDialog titleDialog = new TextInputDialog("");
    titleDialog.setTitle("Title for your set");
    titleDialog.setHeaderText("Please provide a title for your new flashcard set");
    Optional<String> result = titleDialog.showAndWait();
    if(result.isPresent() && !result.get().equals("")) {
        Parent root = null;
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/learn_update.fxml"));
        try {
            root = loader.load();
        } catch(IOException ex) {ex.printStackTrace();}
        LearnUpdateController controller = loader.getController();

        //add initial flashcards
        for(int i = 0; i < 5; i++) controller.addRow();

        controller.initData(result.get(), rootController, this, apnLearn);
        rootController.setActivity(root);
    }
}
```

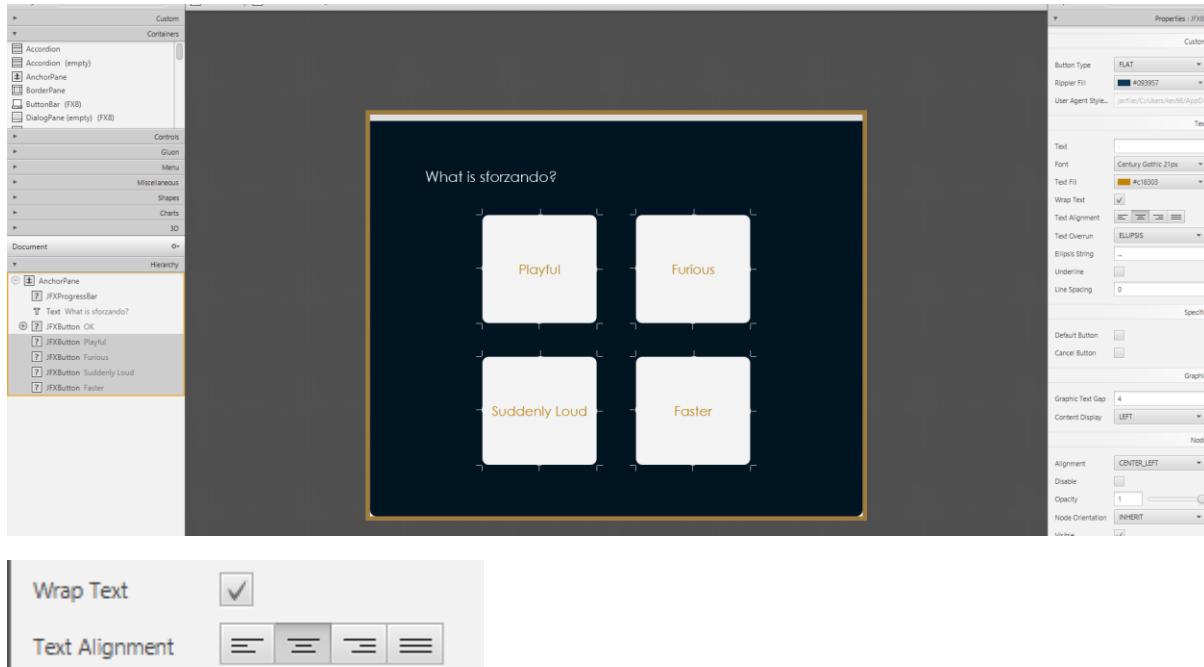
Then, you get this achieved result (without clicking on *Add New Flashcard*):



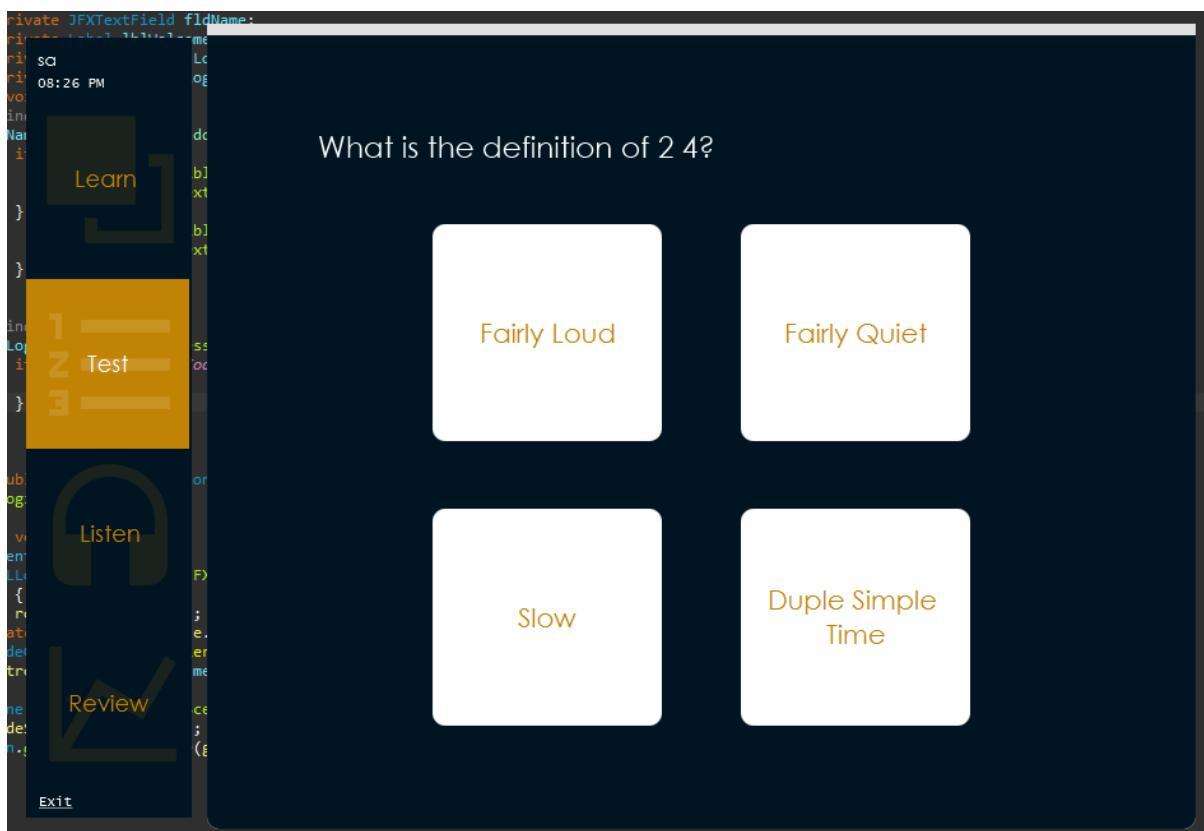
Test Adjustments

To make the text on the button not overrun, we just needed to wrap the text all the buttons.

We can do this in the SceneBuilder with test_quiz.fxml.



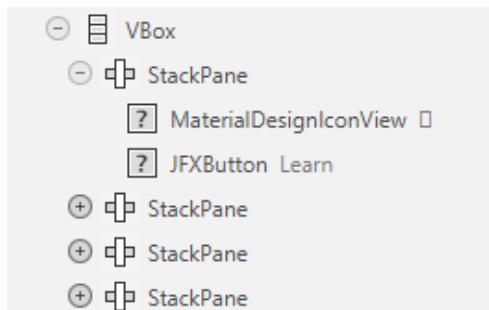
Now, long answers are wrapped inside the JFXButton.



Listen Adjustments

Firstly, we should make it so that only grade 8 students can access the Listen section, since it is only relevant for grade 8.

All we need to do to prohibit it is to remove the Listen button from the navigation bar, apnNavigate. To start, we can add all the activities in a VBox, then, at the initialisation of RootController, checking the grade, we can remove the Listen StackPane, spnListen, from the VBox.

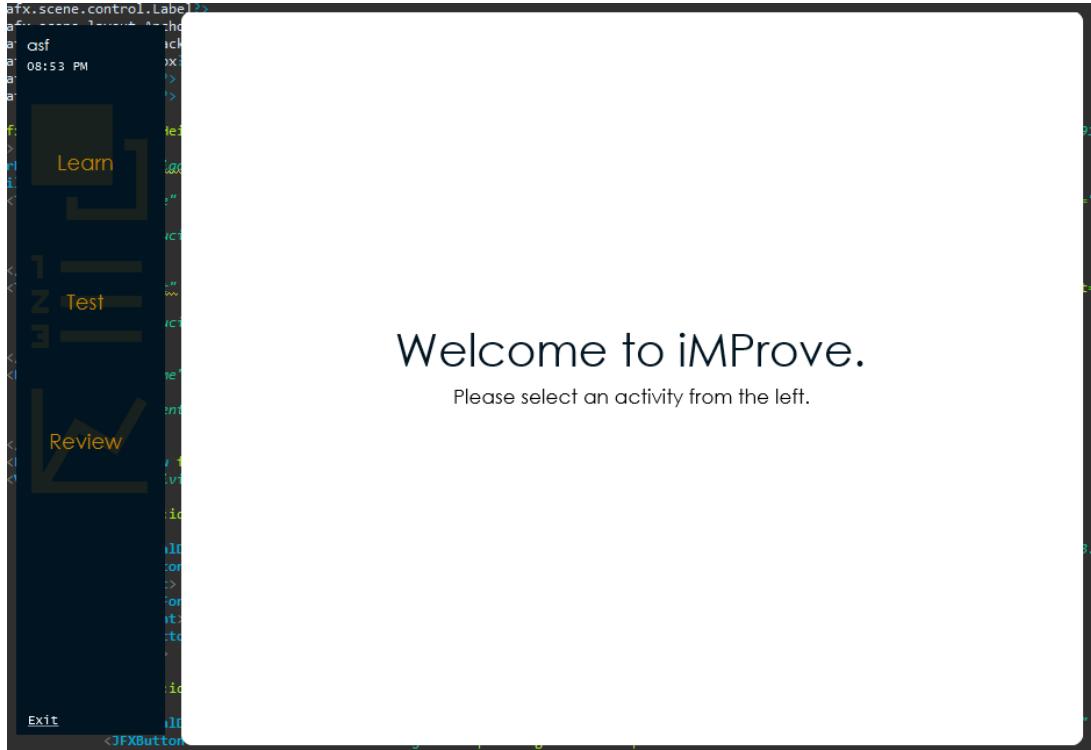


This VBox containing all the StackPanes relating to each activity is called vbxActivities.

```
public void initialize() {
    Platform.runLater(()->{
        //take off listen section if not grade 8
        if(grade!=8) {
            vbxActivities.getChildren().remove(spnListen);
        }

        apnRoot.requestFocus();
        //automatically update time
        startTime();
    });
}
```

After selecting grade 4 (anything less than grade 8 will produce the same result), the Listen section was not displayed and therefore inaccessible.



Secondly, we should change the overall test structure of Listen's quiz so that it resembles more of an exam.

We should have to make sure that the user does not try to change the position of the audio throughout his test, otherwise would render this quiz type solution useless. To do this, we can dive into `ListenQuizController.java` and move the code which monitors the value of the slider to a different method which should only be called when we want the user to be able to seek through the audio.

```
public void allowSeeking() {
    sldSeek.valueProperty().addListener(new InvalidationListener() {
        @Override
        public void invalidated(Observable o) {
            if(sldSeek.isValueChanging()) { //if the slider is truly moving
                Duration newDuration = Duration.millis(currentTest.getDuration()).multiply(sldSeek.getValue()/100.0);
                currentTest.getMediaPlayer().seek(newDuration);
                if(sldSeek.getValue()==100.0) {
                    finished = true;
                    currentTest.getMediaPlayer().pause();
                    icnPlayPause.setGlyphName("PLAY");
                }
            }
        }
    });
}
```

Although dragging on the slider will not change the audio, the user can still drag the slider. So, to stop this, we can disable it after every change on the `InvalidationListener` – we cannot disable it at the start otherwise it will just enable again when the slider changes value with `setValue()` from the `InvalidationListener` in the `syncSlider()` method.

```

currentTest.getMediaPlayer().currentTimeProperty().addListener(new InvalidationListener() { //to auto sync the slider with the media player being played
@Override
public void invalidated(Observable o) {
    Duration currentDuration = currentTest.getMediaPlayer().getCurrentTime();
    Duration totalDuration = Duration.millis(currentTest.getDuration());
    sldSeek.setDisable(totalDuration.isUnknown()); //disable slider if the music has not been correctly loaded yet
    if(!sldSeek.isDisable() && totalDuration.greaterThan(Duration.ZERO)&& !sldSeek.isValueChanging()) {
        double percentageIn = (currentDuration.toMillis()/totalDuration.toMillis()) * 100;
        sldSeek.setValue(percentageIn);
        sldSeek.setDisable(true); //disable
    }
});
}

```

Also, to make it more obvious that at the end, the user can seek through the audio after answering the question, we will hide the circle (thumb) of the slider until the user has answered the question.

```

5 .jfx-slider .thumb {
6     -fx-background-color: #c18303;
7     visibility: hidden;
8 }

```

Then, making another stylesheet, `slider-enable.css` to be applied on the slider at the end, we can make the visibility of the thumb visible again.

Code for `slider-enable.css`:

```

1 .jfx-slider .thumb {
2     visibility: visible;
3 }

```

We want to allow all the customisation and media manipulation at the end after the question has been answered, for all the learning benefits of looking back and reviewing that we like!

Since we have a dedicated method to make the changes of the slider, we can add some more lines of code at the end of the method to enable the slider for use, apply the `slider-enable.css` stylesheet onto the slider and display the play button.

```

public void allowSeeking() {
    sldSeek.valueProperty().addListener(new InvalidationListener() {
        @Override
        public void invalidated(Observable o) {
            if(sldSeek.isValueChanging()) { //if the slider is truly moving
                Duration newDuration = Duration.millis(currentTest.getDuration()).multiply(sldSeek.getValue()/100.0);
                currentTest.getMediaPlayer().seek(newDuration);
                if(sldSeek.getValue()==100.0) {
                    finished = true;
                    currentTest.getMediaPlayer().pause();
                    icnPlayPause.setGlyphName("PLAY");
                }
            }
        }
    });
    Platform.runLater(()->{
        sldSeek.setDisable(false);
        sldSeek.getStylesheets().add("view/slider-enable.css");
        icnPlayPause.setVisible(true);
    });
}

```

To use this method at the right time, we can just call `allowSeeking()` at the end of answering the question.

What is achieved is a somewhat greyed progress bar which cannot be dragged – which is good.

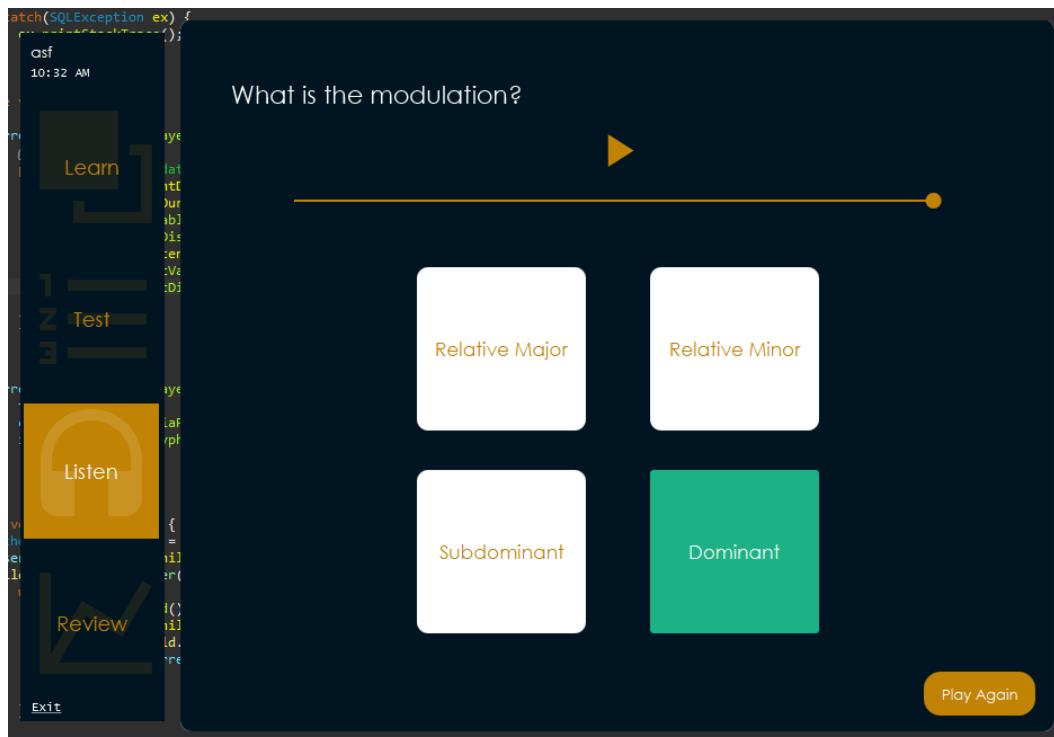
Also, it automatically stops at the end of the music.

```
    @FXML
    public void answerQuestion(ActionEvent e) {
        if(!answered) { //if hasn't been answered yet - this will make clicks to other buttons not change their colour
            answered = true;

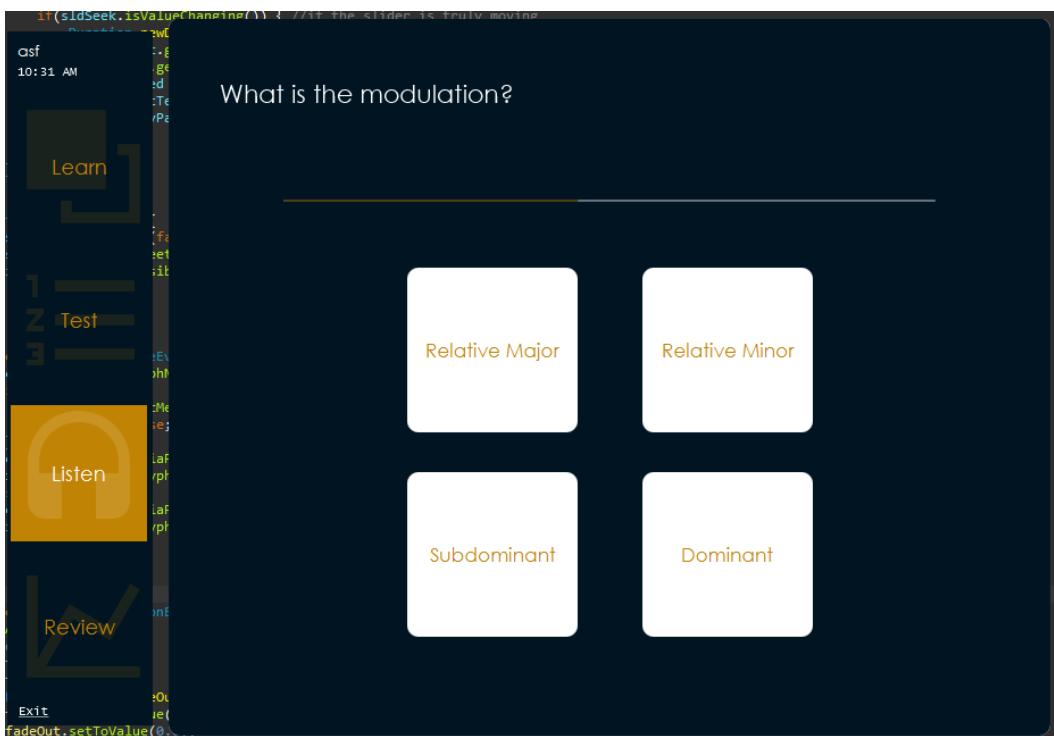
            //make input red
            Button inputBtn = (Button) e.getSource();
            inputBtn.setStyle("-fx-background-color: #6f0f0f");
            inputBtn.setTextFill(Color.WHITE);

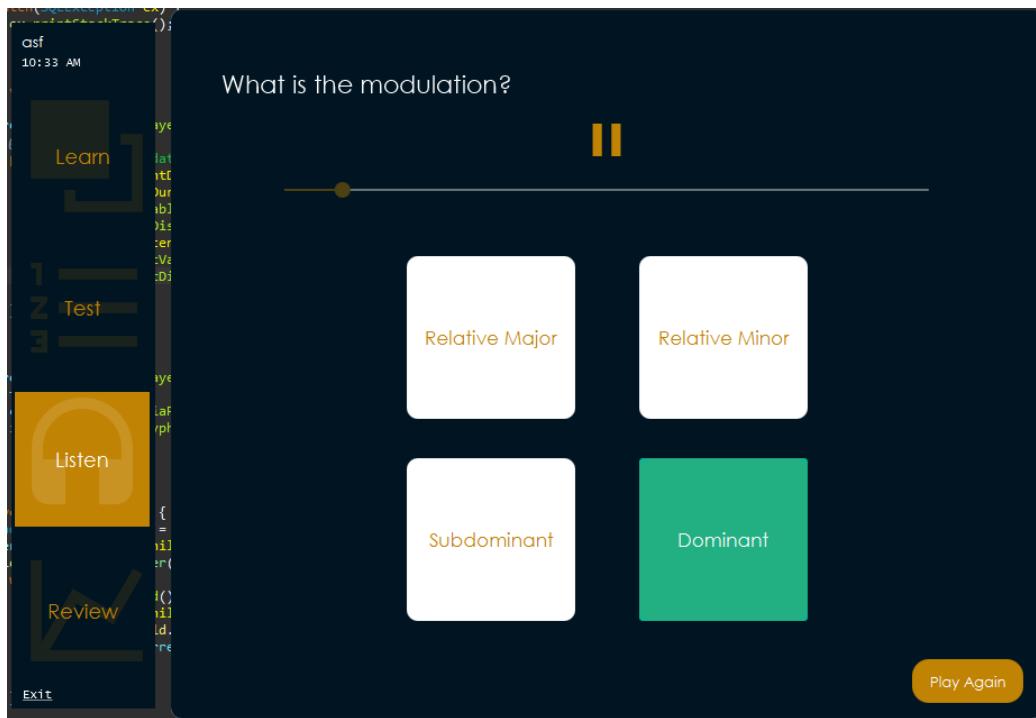
            //make correct answer green - correct input will be overridden green also
            switch(currentTest.getModulation()) {
                case "Relative Major": btnRelMaj.setStyle("-fx-background-color: #0cb794");
                    btnRelMaj.setTextFill(Color.WHITE);
                    break;
                case "Relative Minor": btnRelMin.setStyle("-fx-background-color: #0cb794");
                    btnRelMin.setTextFill(Color.WHITE);
                    break;
                case "Dominant": btnDom.setStyle("-fx-background-color: #0cb794");
                    btnDom.setTextFill(Color.WHITE);
                    break;
                case "Subdominant": btnSub.setStyle("-fx-background-color: #0cb794");
                    btnSub.setTextFill(Color.WHITE);
                    break;
            }
            btnAgain.setVisible(true); //show play again
            allowSeeking();
        }
    }
```

After we answer the question, we have options to play the music again, start at a particular point, and play again.

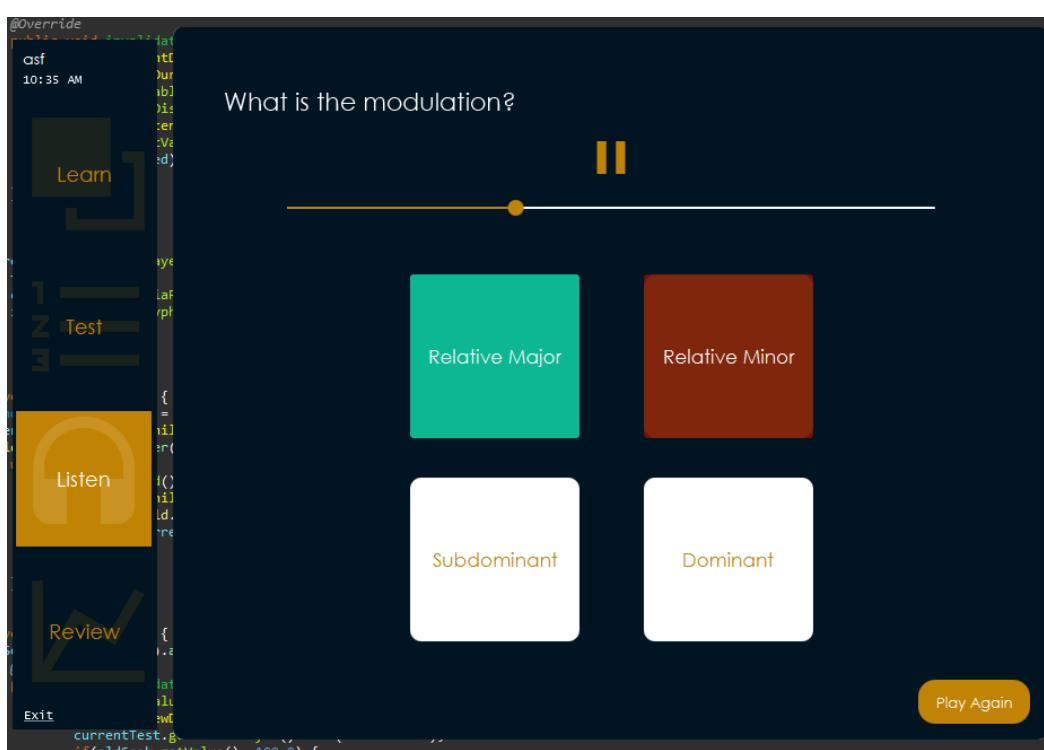


But some things did not go exactly to plan.





Which now fully works.



Although everything works, the structure is still quite unclear. So, we can make the first step more obvious: to firstly play the music.

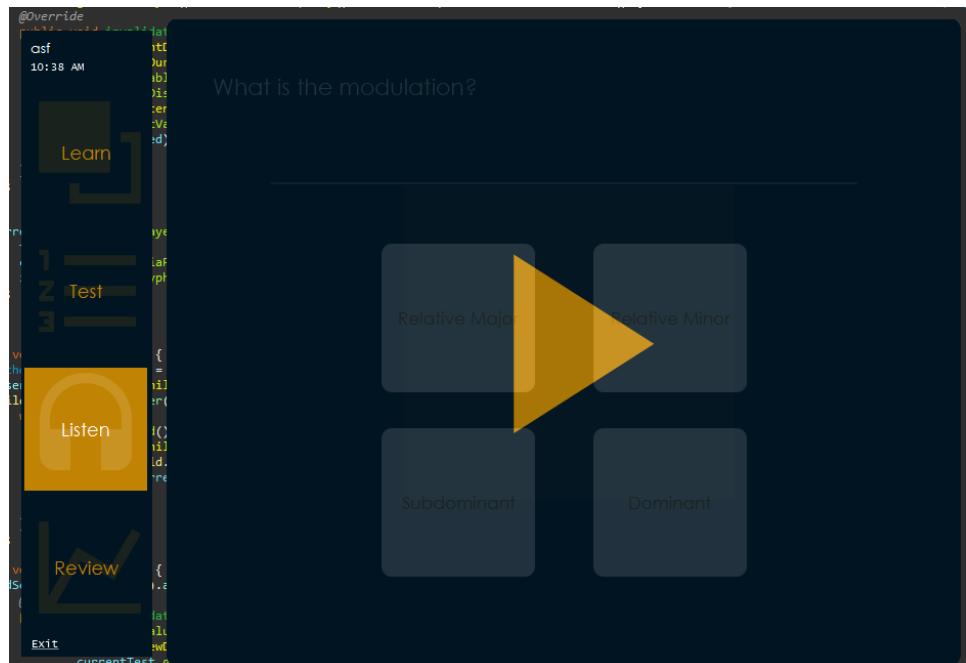
We can mask the `listen_quiz.fxml` with an `AnchorPane` over it, `apnFirstPlay`, and only contain a play button to explicitly highlight we should firstly play the music. Also, we can fade `apnFirstPlay` to have opacity 0 once it has been clicked.



So, the play button can be binded to a method called `firstPlay()` which will fade, set the pane not visible at the end of the fade (so that the user can interact with what is beneath) and play the music if not already (so the user cannot click on the button again whilst fading).

```
@FXML
public void firstPlay(ActionEvent e) {
    //fade
    if(!fading) {
        fading = true;
        FadeTransition fadeOut = new FadeTransition(Duration.seconds(1), apnFirstPlay);
        fadeOut.setFromValue(1.0);
        fadeOut.setToValue(0.0);
        fadeOut.play();
        Timeline invisible = new Timeline(new KeyFrame(Duration.seconds(0)), new KeyFrame(Duration.seconds(1), event -> apnFirstPlay.setVisible(false)));
        invisible.play();
        currentTest.getMediaPlayer().play();
    }
}
```

It does as we expect.



However, after the 1 second, the Learn section randomly highlights.



So, calling `requestFocus()` to a node without a binded method, inside the event handler of the KeyFrame at one second, solves this problem.

```
Timeline invisible = new Timeline(new KeyFrame(Duration.seconds(0)), new KeyFrame(Duration.seconds(1), event ->{
    apnFirstPlay.setVisible(false);
    apnQuiz.requestFocus();
});|
```

Another issue is that once you click off the Listen section to another section, the music still plays.

So, we can create a method, `bindAutoStop()`, which will automatically stop the music if the user is no longer in the Listen section. To do so, since the RootController switches activity by merely removing the associated AnchorPane, we can listen for when `apnQuiz` is removed and stop the MediaPlayer if so.

```
public void bindAutoStop() {
    AnchorPane apnActivity = (AnchorPane) apnQuiz.getParent();
    apnActivity.getChildren().addListener((Change<? extends Node> c) -> {
        while(c.next()) {
            if(c.wasRemoved()) {
                for(Node child: c.getRemoved()) {
                    if(child.equals(apnQuiz)) //quiz removed
                        currentTest.getMediaPlayer().stop(); //stop the music
                }
            }
        }
    });
}|
```

This method will be called at the start after `getAuralTest()` and `syncAudio()`.

However, you get a long list of errors where the root cause is a `NullPointerException` so one of our objects is de facto null.

```

at com.sun.javaev.event.EventUtil.fireEvent(EventUtil.java:54)
at javafx.event.Event.fireEvent(Event.java:198)
at javafx.scene.Scene$MouseHandler.process(Scene.java:3757)
at javafx.scene.Scene$MouseHandler.access$1500(Scene.java:3485)
at javafx.scene.Scene.impl_processMouseEvent(Scene.java:1762)
at javafx.scene.Scene$ScenePeerListener.mouseEvent(Scene.java:2494)
at com.sun.javafx.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:394)
at com.sun.javafx.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:295)
at java.security.AccessController.doPrivileged(Native Method)
at com.sun.javafx.tk.quantum.GlassViewEventHandler.lambda$handleMouseEvent$353(GlassViewEventHandler.java:432)
at com.sun.javafx.tk.quantum.QuantumToolkit.runWithoutRenderLock(QuantumToolkit.java:389)
at com.sun.javafx.tk.quantum.GlassViewEventHandler.handleMouseEvent(GlassViewEventHandler.java:431)
at com.sun.glass.ui.View.handleMouseEvent(View.java:555)
at com.sun.glass.ui.View.notifyMouse(View.java:937)
at com.sun.glass.ui.win.WinApplication._runLoop(Native Method)
at com.sun.glass.ui.win.WinApplication.lambda$null$147(WinApplication.java:177)
at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.reflect.InvocationTargetException
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at sun.reflect.misc.Trampoline.invoke(MethodUtil.java:71)
at sun.reflect.GeneratedMethodAccessor1.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at sun.reflect.misc.MethodUtil.invoke(MethodUtil.java:275)
at javafx.fxml.FXMLLoader$MethodHandler.invoke(FXMLLoader.java:1769)
... 48 more
Caused by: java.lang.NullPointerException
at controller.ListenQuizController.bindAutoStop(ListenQuizController.java:111)
at controller.ListenQuizController.initData(ListenQuizController.java:52)
at controller.ListenController.playQuiz(ListenController.java:27)
... 58 more

```

If we trace back to the line, we get:

```

109  public void bindAutoStop() {
110      AnchorPane apnActivity = (AnchorPane) apnQuiz.getParent();
111      apnActivity.getChildren().addListener((Change<? extends Node> c) -> {
112          while(c.next()) {
113              if(c.wasRemoved()) {
114                  for(Node child: c.getRemoved()) {
115                      if(child.equals(apnQuiz)) //quiz removed
116                          currentTest.getMediaPlayer().stop(); //stop the music
117                  }
118              }
119          });
120      });
121  }

```

Since the line nests multiple methods onto one line, we can separate them and check which item is null.

```

public void bindAutoStop() {
    AnchorPane apnActivity = (AnchorPane) apnQuiz.getParent();
    if(apnActivity==null)System.out.println("Activity null");
    ObservableList<Node> childrenList = apnActivity.getChildren();
    if(childrenList==null)System.out.println("Children null");
    childrenList.addListener((Change<? extends Node> c) -> {
        while(c.next()) {
            if(c.wasRemoved()) {
                for(Node child: c.getRemoved()) {
                    if(child.equals(apnQuiz)) //quiz removed
                        currentTest.getMediaPlayer().stop(); //stop the music
                }
            }
        });
    });
}

```

On checking the console, our activity was null:

```

Activity null
Exception in thread "JavaFX Application Thread" java.lang.RuntimeException: java.lang.reflect.InvocationTargetException

```

To make sure it was not the apnQuiz that was null in the first place, we added an extra line at the start of the method.

```

public void bindAutoStop() {
    if(apnQuiz==null)System.out.println("Quiz null");
    AnchorPane apnActivity = (AnchorPane) apnQuiz.getParent();
}

```

Yet, it was still the apnActivity causing the NullPointerException.

```

java:11 [Java Application] C:\Program Files\Java\jdk1.8.0_151\bin\javaw.exe (29 Oct 2016, 10:13:15)
Activity null
Exception in thread "JavaFX Application Thread" java.lang.RuntimeException: java.lang.reflect.InvocationTargetException
    at javafx.fxml.FXMLLoader$ControllerMethodHandler.invoke(FXMLLoader.java:1774)

```

So that concludes apnQuiz.getParent() is returning null. Tracing back to when apnQuiz gets a parent node, we go back to the RootController inside the goListen() method – setActivity() line:

```

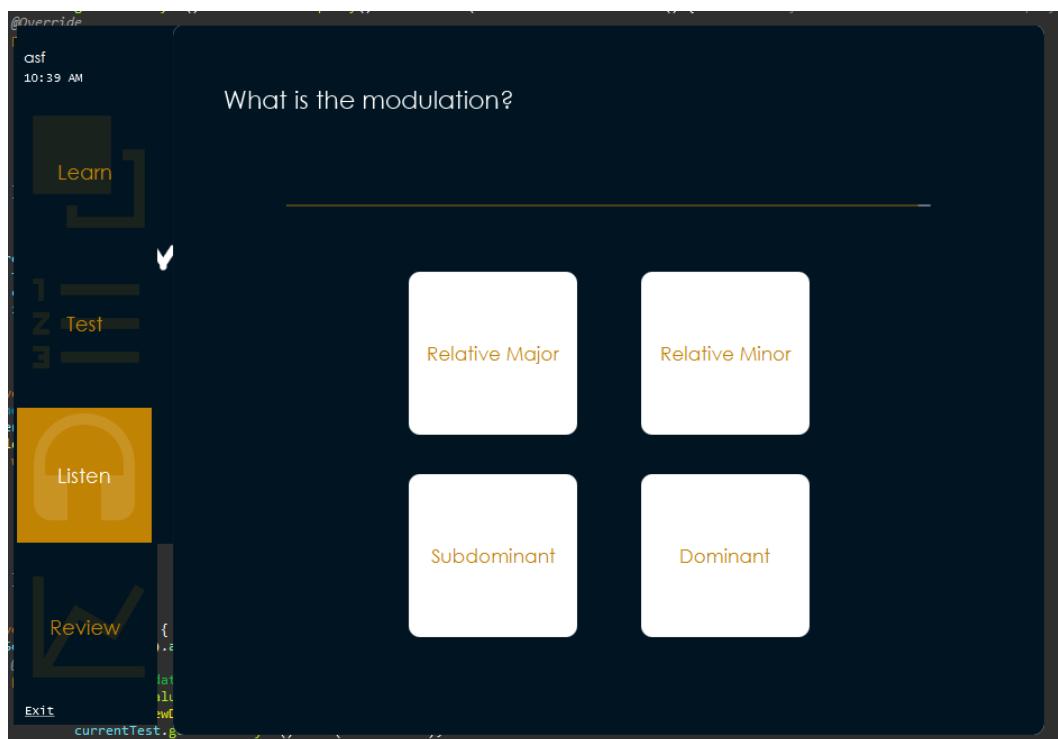
@FXML
public void goListen(ActionEvent e) {
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/listen.fxml"));
    try {
        root = loader.load();
    } catch(IOException ioe) {ioe.printStackTrace();}
    ListenController controller = loader.getController();
    controller.setRootController(this);
    changeNavigationColours(spnListen, icnListen, btnListen);
    setActivity(root);
}

```

We notice that we call `setRootController()` before `apnQuiz` (root in this case) even gets a parent! It is not a good to swap these methods around otherwise there is a chance the user can interact with controls and audio before they have even fully loaded up correctly. So, we should call `bindAutoStop()` in the `ListenQuizController.java` after everything has loaded – in an event handling method. Since we only need to stop the music only after it has been played, we can call `bindAutoStop()` inside the `firstPlay()` method.

```
@FXML  
public void firstPlay(ActionEvent e) {  
    bindAutoStop();  
  
    //fade  
    if(!fading) {  
        fading = true;  
        FadeTransition fadeOut = new FadeTransition(Duration.seconds(1), apnFirstPlay);  
        fadeOut.setFromValue(1.0);  
        fadeOut.setToValue(0.0);  
        fadeOut.play();  
        currentTest.getMediaPlayer().play();  
    }  
}
```

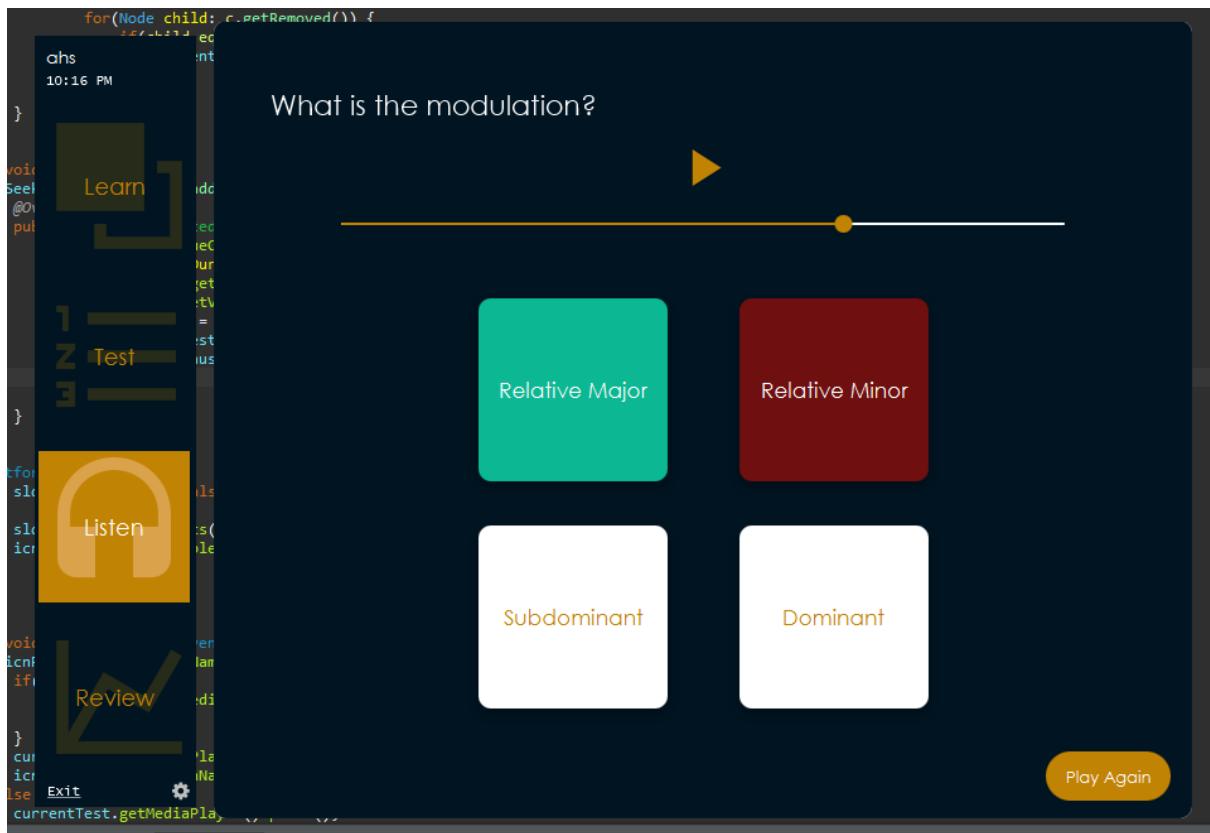
Now, everything works fine and when switching to another activity, the music stops.



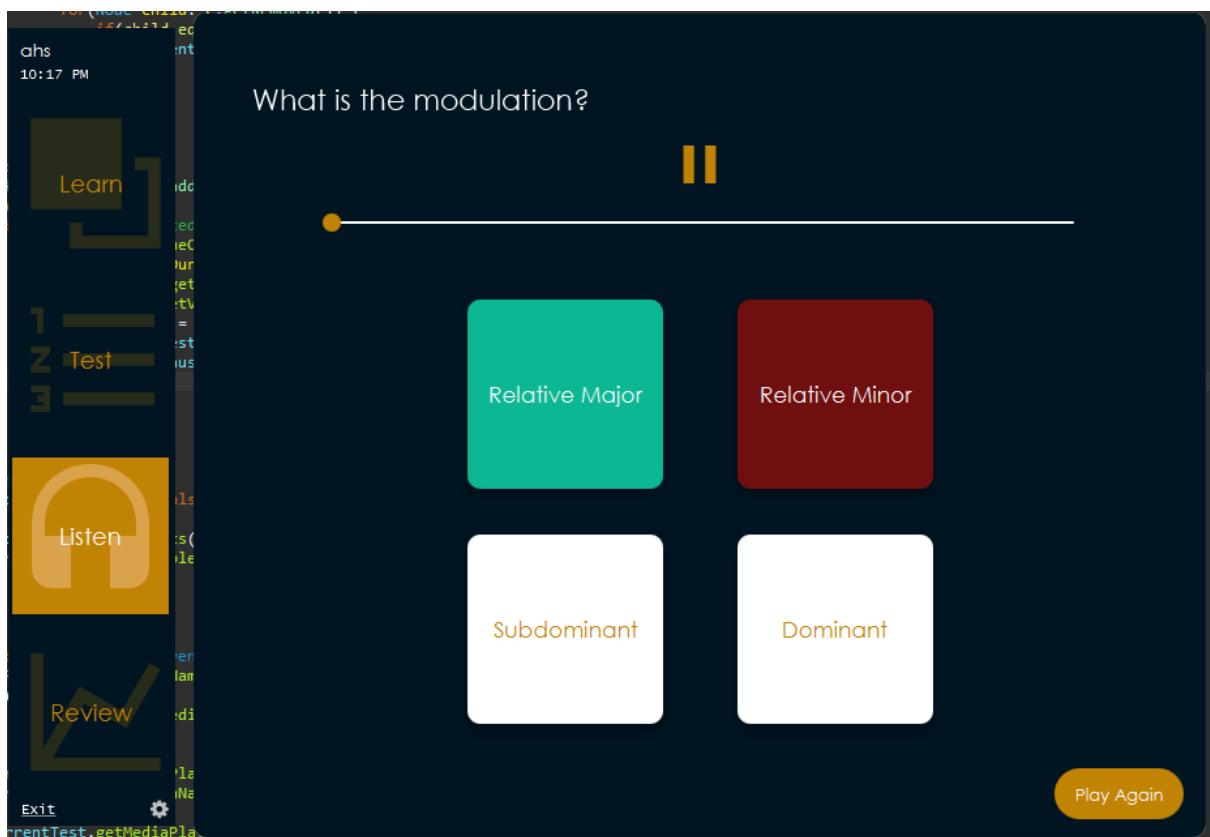
Also, no errors or null messages are displayed so we can remove the output lines from our code.



After further testing, we stumbled across a softer problem, if you finish your Listen test and drag the slider to an earlier point in time....



You press play but it starts again from the beginning.



Analysing our code in the ListenQuizController.java class, we firstly start at looking what happens at the end of the audio, i.e. the Runnable that is set by setOnEndOfMedia().

```
currentTest.getMediaPlayer().setOnEndOfMedia(()-> {
    finished = true;
    currentTest.getMediaPlayer().pause();
    icnPlayPause.setGlyphName("PLAY");
});
```

The only thing that could cause something like changing the music position would be what uses the boolean finished. We notice that in our playPause() method, we have set the music to play at the start when finished.

```
@FXML
public void playPause(MouseEvent e) {
    if(icnPlayPause.getGlyphName().equals("PLAY")) {
        if(finished) {
            currentTest.getMediaPlayer().seek(currentTest.getMediaPlayer().getStartTime());
            finished = false;
        }
        currentTest.getMediaPlayer().play();
        icnPlayPause.setGlyphName("PAUSE");
    } else {
        currentTest.getMediaPlayer().pause();
        icnPlayPause.setGlyphName("PLAY");
    }
}
```

However, this feature was wanted, it caused some unexpected events when dragging the slider to a different position since the finished boolean value would still be set to true. So, in our allowSeeking() method, looking at the listener of the slider which changes the audio time, adding an else clause to the already existing *if-statement* will allow us to check if the slider is at a different duration than the end, and so setting the value of finished to false if so.

```
sldSeek.valueProperty().addListener(new InvalidationListener() {
    @Override
    public void invalidated(Observable o) {
        if(sldSeek.isValueChanging()) { //if the slider is truly moving
            Duration newDuration = Duration.millis(currentTest.getDuration()).multiply(sldSeek.getValue()/100.0);
            currentTest.getMediaPlayer().seek(newDuration);
            if(sldSeek.getValue()==100.0) {
                finished = true;
                currentTest.getMediaPlayer().pause();
                icnPlayPause.setGlyphName("PLAY");
            } else {
                finished = false;
            }
        }
    }
});
```

Updated code for ListenQuizController.java:

```
package controller;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.jfoenix.controls.JFXButton;
import com.jfoenix.controls.JFXSlider;

import de.jensd.fx.glyphs.materialdesignicons.MaterialDesignIconView;
import javafx.animation.FadeTransition;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Platform;
import javafx.beans.InvalidationListener;
import javafx.beans.Observable;
import javafx.collections.ListChangeListener.Change;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.control.Button;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.util.Duration;
import model.AuralTest;
import other.DAO;

public class ListenQuizController {
    private RootController rootController;
    private DAO dao = new DAO();
    private AuralTest currentTest = null;
    private boolean answered = false;
    private boolean finished = false;
    private boolean fading = false;
    private int repeat = 0;
    @FXML private JFXButton btnRelMaj, btnRelMin, btnSub, btnDom, btnAgain;
    @FXML private MaterialDesignIconView icnPlayPause;
    @FXML private JFXSlider sldSeek;
    @FXML private Text txtRepeat;
    @FXML private AnchorPane apnFirstPlay, apnQuiz;
    public void initData(RootController rootController) {
        this.rootController = rootController;
        getAuralTest();
        syncAudio();
    }

    private void getAuralTest() {
        try {
            Connection conn = dao.getConnection();
            PreparedStatement cntStmt = conn.prepareStatement("SELECT COUNT(*) FROM Aural");
            ResultSet cntRs = cntStmt.executeQuery();
            PreparedStatement rndStmt = conn.prepareStatement("SELECT File, Modulation FROM Aural");
            ResultSet rndRs = rndStmt.executeQuery();
        } {
            int length = 0; //the upper bound for random row
            if(cntRs.next())length = cntRs.getInt(1);
            int randomRow = (int) Math.floor(length * Math.random()); //floor for lower bound to generate 'length' number of integers (inc 0, exc end)
            for(int i = 0; i<=randomRow; i++)rndRs.next(); //go to random row <= to move past zeroth row
            currentTest = new AuralTest(rndRs.getString("File"), rndRs.getString("Modulation"));
        } catch(SQLException ex) {
            ex.printStackTrace();
        }
    }

    private void syncAudio() {
        currentTest.getMediaPlayer().currentTimeProperty().addListener(new InvalidationListener() { //to auto sync the slider with the media player being played
    
```

```

        Duration currentDuration = currentTest.getMediaPlayer().getCurrentTime();
        Duration totalDuration = Duration.millis(currentTest.getDuration());
        sldSeek.setDisable(totalDuration.isUnknown()); //disable slider if the music has not been correctly loaded yet
        if(!sldSeek.isDisable() && totalDuration.greaterThan(Duration.ZERO)&& !sldSeek.isValueChanging()) {
            double percentageIn = (currentDuration.toMillis()/totalDuration.toMillis()) * 100;
            sldSeek.setValue(percentageIn);
            if(!answered)sldSeek.setDisable(true); //disable
        }
    }
});

currentTest.getMediaPlayer().setOnEndOfMedia(()-> {
    finished = true;
    currentTest.getMediaPlayer().pause();
    icnPlayPause.setGlyphName("PLAY");
});

}

public void bindAutoStop() {
    AnchorPane apnActivity = (AnchorPane) apnQuiz.getParent();
    ObservableList<Node> childrenList = apnActivity.getChildren();
    childrenList.addListener((Change<? extends Node> c) -> {
        while(c.next()) {
            if(c.wasRemoved()) {
                for(Node child: c.getRemoved()) {
                    if(child.equals(apnQuiz)) //quiz removed
                        currentTest.getMediaPlayer().stop(); //stop the music
                }
            }
        }
    });
}

public void allowSeeking() {
    sldSeek.valueProperty().addListener(new InvalidationListener() {
        @Override
        public void invalidated(Observable o) {
            if(sldSeek.isValueChanging()) { //if the slider is truly moving
                Duration newDuration = Duration.millis(currentTest.getDuration()).multiply(sldSeek.getValue()/100.0);
                currentTest.getMediaPlayer().seek(newDuration);
                if(sldSeek.getValue()==100.0) {
                    finished = true;
                    currentTest.getMediaPlayer().pause();
                    icnPlayPause.setGlyphName("PLAY");
                } else {
                    finished = false;
                }
            }
        }
    });
}

Platform.runLater(()->{
    sldSeek.setDisable(false);
    sldSeek.getStylesheets().add("view/slider-enable.css");
    icnPlayPause.setVisible(true);
});

}

@FXML
public void playPause(MouseEvent e) {
    if(icnPlayPause.getGlyphName().equals("PLAY")) {
        if(finished) {
            currentTest.getMediaPlayer().seek(currentTest.getMediaPlayer().getStartTime());
            finished = false;
        }
    }
}

```

```

        currentTest.getMediaPlayer().play();
        icnPlayPause.setGlyphName("PAUSE");
    } else {
        currentTest.getMediaPlayer().pause();
        icnPlayPause.setGlyphName("PLAY");
    }
}

@FXML
public void firstPlay(ActionEvent e) {
    bindAutoStop();
    //fade
    if(!fading) {
        fading = true;
        FadeTransition fadeOut = new FadeTransition(Duration.seconds(1), apnFirstPlay);
        fadeOut.setFromValue(1.0);
        fadeOut.setToValue(0.0);
        fadeOut.play();
        Timeline invisible = new Timeline(new KeyFrame(Duration.seconds(0)), new KeyFrame(Duration.seconds(1), event ->{
            apnFirstPlay.setVisible(false);
            apnQuiz.requestFocus();
        }));
        invisible.play();
        currentTest.getMediaPlayer().play();
    }
}

@FXML
public void answerQuestion(ActionEvent e) {
    if(!answered) { //if hasn't been answered yet - this will make clicks to other buttons not change their colour
        answered = true;

        //make input red
        Button inputBtn = (Button) e.getSource();
        inputBtn.setStyle("-fx-background-color: #ff0000");
        inputBtn.setTextFill(Color.WHITE);

        //make correct answer green - correct input will be overridden green also
        switch(currentTest.getModulation()) {
            case "Relative Major": btnRelMaj.setStyle("-fx-background-color: #0cb794");
                btnRelMaj.setTextFill(Color.WHITE);
                break;
            case "Relative Minor": btnRelMin.setStyle("-fx-background-color: #0cb794");
                btnRelMin.setTextFill(Color.WHITE);
                break;
            case "Dominant": btnDom.setStyle("-fx-background-color: #0cb794");
                btnDom.setTextFill(Color.WHITE);
                break;
            case "Subdominant": btnSub.setStyle("-fx-background-color: #0cb794");
                btnSub.setTextFill(Color.WHITE);
                break;
        }
        btnAgain.setVisible(true); //show play again
        allowSeeking();
    }
}

@FXML
public void playAgain(ActionEvent e) {
    currentTest.getMediaPlayer().stop(); //stop the music if it's still playing

    //load again the quiz page
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/listen_quiz.fxml"));
    try {
        root = loader.load();
    } catch(IOException ex) {ex.printStackTrace();}
    ListenQuizController controller = loader.getController();
    controller.initData(rootController);
    rootController.setActivity(root);

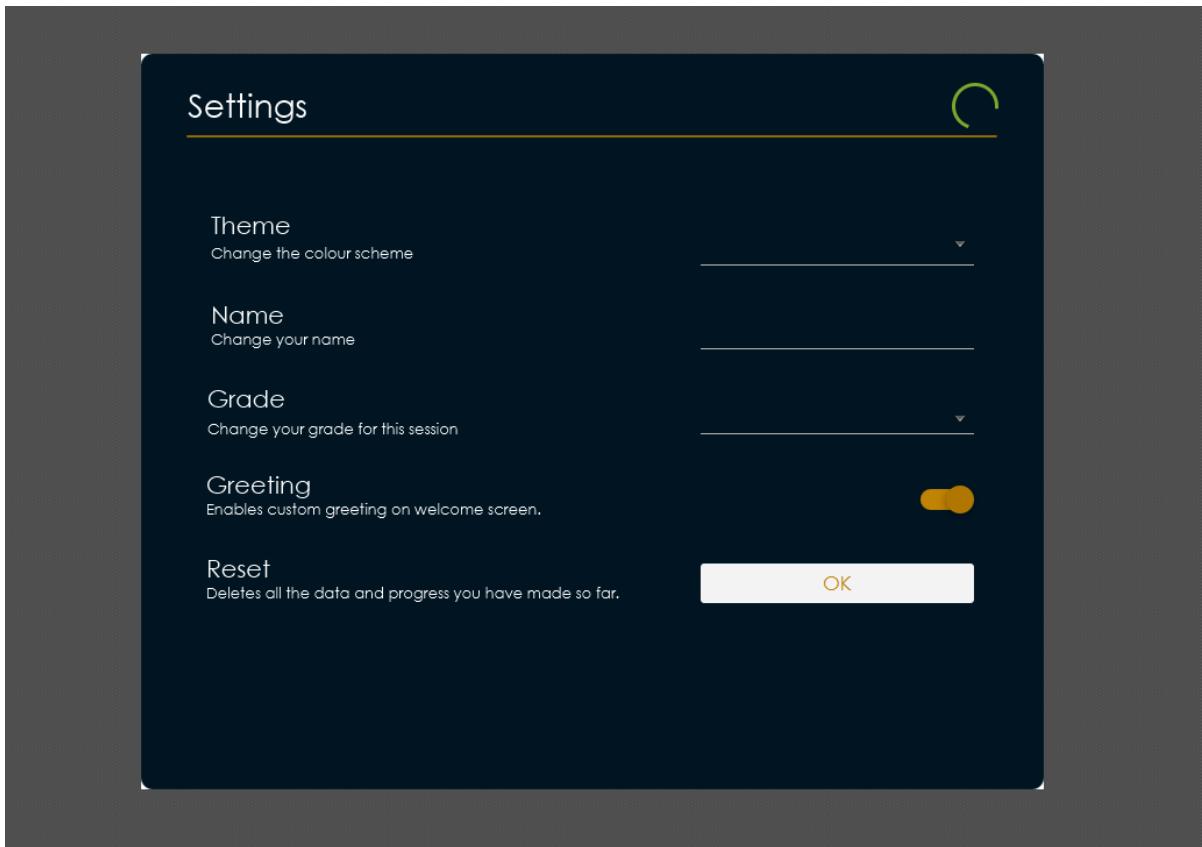
    root.requestFocus(); //so doesn't focus on a button
}
}

```

Settings Page

A feature which allows for customisation of the iMProve system – although limited.

We have an FXML document to use as the view settings.fxml:



We have a controller to control this view, `SettingsController.java`, this will handle all the changes made and link the user interaction with the data of the application.

Both will be loaded from the `RootController.java` class when the user clicks the cog icon beside the exit control.



We will load the settings page very similar to every other activity (i.e Learn, Test etc.)

```
@FXML
public void goSettings(MouseEvent e) {
    Parent root = null;
    FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/settings.fxml"));
    try {
        root = loader.load();
    } catch(IOException ex) {ex.printStackTrace();}
    SettingsController controller = loader.getController();
    controller.setRootController(this);
    changeNavigationColours(null, null, null); //null doesn't equal any of the objects of the activities so all will turn blue (unselected)
    setActivity(root);
}
```

However, one thing that is different is that when calling `changeNavigationColours()`, we pass the parameters: null, null, null. This is so that in the `changeNavigationColours`, it changes all of them to unselected (blue and orange) and none to be selected (orange and white).

```
public void changeNavigationColours(StackPane spn, MaterialDesignIconView icn, JFXButton btn) {
    for(StackPane pane: Arrays.asList(spnLearn, spnTest, spnListen, spnReview)) {
        if(!pane.equals(spn))
            pane.setStyle("-fx-background-color: #001421");
        else
            pane.setStyle("-fx-background-color: #c18303");
    }
    for(MaterialDesignIconView icon: Arrays.asList(icnLearn, icnTest, icnListen, icnReview)) {
        if(!icon.equals(icn))
            icon.setFill(Color.web("#c18303", 0.13));
        else
            icon.setFill(Color.web("#FFFFFF", 0.13));
    }
    for(JFXButton button: Arrays.asList(btnLearn, btnTest, btnListen, btnReview)) {
        if(!button.equals(btn))
            button.setTextFill(Color.web("#c18303"));
        else
            button.setTextFill(Color.web("FFFFFF"));
    }
}
```

This is because actual objects inside the iterated ArrayLists will not equal null (e.g. `Arrays.asList(spnLearn, spnTest, spnListen, spnReview)` contain elements `spnLearn, spnTest, spnListen, spnReview` which none are null so will all become blue with `#001421` background).

When a `SettingsController` object is initialised, we want to set the items inside the `JFXComboBox`'s (`cbxTheme` for the theme; `cbxGrade` for the grade) and load up all the initial values for the controls.

```
public void initialize() {
    Platform.runLater(() -> {
        //initialise the comboboxes
        cbxTheme.getItems().addAll("Default Chamber", "Ice Dragon", "Minimal Lightning");
        cbxGrade.getItems().addAll(1, 2, 3, 4, 5, 6, 7, 8);

        //set initial values
        apnSettings.requestFocus(); //no highlighted control
    });
}
```

To set the initial values, we will firstly set the easy ones defined in the `RootController`, i.e the grade and name of the user, and then load the rest (greeting and theme) from a properties file – similar to an XML file and stores configurations but is more specific to Java.

```
//set initial values
fldName.setText(RootController.getName());
cbxGrade.setValue(RootController.getGrade());
loadValues();
```

We have a class variable, `settingsProps` which is our object to store and retrieve our properties. Using a class variable, we can use the same Properties object for different methods.

```
private Properties settingsProps = new Properties();
```

In the `loadValues()` method, we will load all the initial values for the settings variables.

```
public void loadValues() {
    final String path = "C:\\\\iMProve\\\\settings.properties";

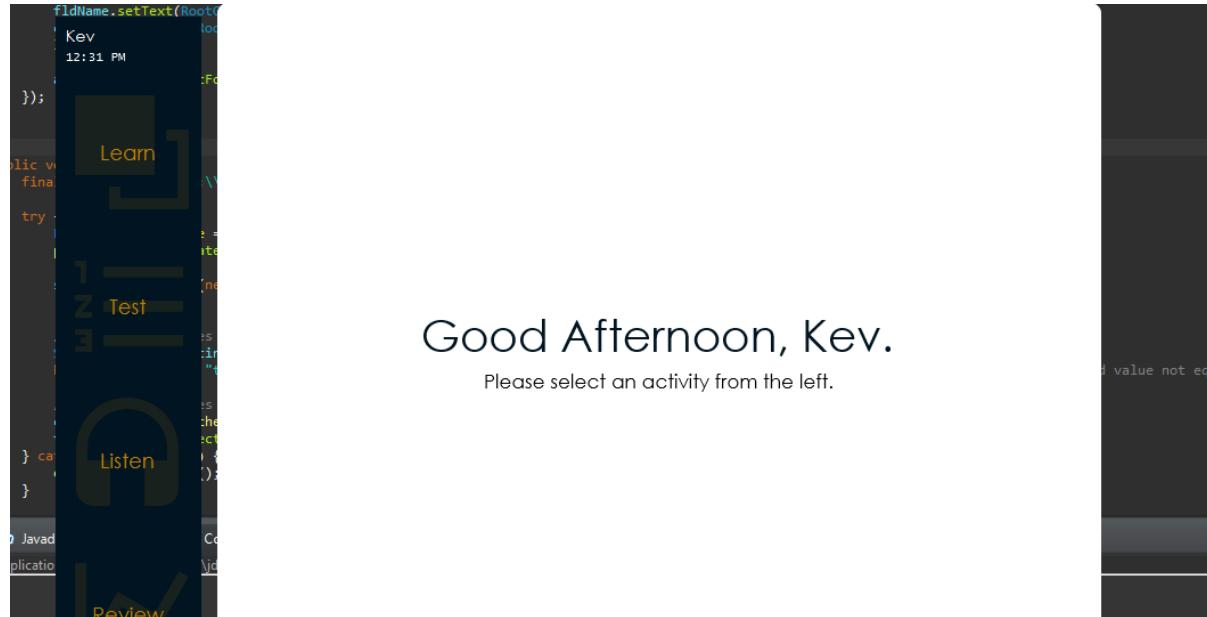
    try {
        File propertiesFile = new File(path);
        propertiesFile.createNewFile(); //if file already exists, does nothing

        settingsProps.load(new FileInputStream(propertiesFile));

        //get the properties
        String theme = settingsProps.getProperty("theme", "Default Chamber");
        boolean greeting = "true".equals(settingsProps.getProperty("greeting", "true")); //if true is what is stored, return true; if stored value not equal to true, return false

        //set control values
        cbxTheme.setValue(theme);
        tglGreeting.setSelected(greeting);
    } catch(IOException ex) {
        ex.printStackTrace();
    }
}
```

Also, for clarification, the ‘custom greeting’ is this:



Which says ‘Good Afternoon/Morning/Night’ depending on the time, rather than the standard ‘Welcome to iMProve.’

This is set at the start of `RootController.java` in the `initialize()` method.

```

private boolean greeting;
public void initialize() {
    Platform.runLater(()->{
        //get properties
        Properties settingsProps = getProperties();
        greeting = "true".equals(settingsProps.getProperty("greeting", "true"));

        //set greeting
        if(greeting) {
            int hour = Integer.parseInt(LocalTime.now().format(DateTimeFormatter.ofPattern("HH")));
            String greeting;
            if(hour>=5 & hour<12)
                greeting = "Good Morning,";
            else if(hour>=12 & hour<18)
                greeting = "Good Afternoon,";
            else if(hour>=18 & hour<23)
                greeting = "Good Evening,";
            else
                greeting = "Good Night,";
            txtWelcome.setText(greeting);
        }
    });
}

```

For the individual controls which change values (JFXComboBox cbxTheme, JFXTextField fldName, JFXComboBox cbxGrade, JFXToggleButton tglGreeting), we have respective methods to change the values of the properties file and also the program's variables.

```

@FXML
public void changeTheme(ActionEvent ae) {
    //code to change theme

    //change properties
    settingsProps.setProperty("theme", cbxTheme.getValue());
    try {
        FileOutputStream out = new FileOutputStream(propertiesFile);
        settingsProps.store(out, "theme changed");
        out.close();
    } catch(IOException ex) {ex.printStackTrace();}

}

@FXML
public void changeName(ActionEvent ae) {
    //code to change name
    rootController.setName(fldName.getText());
}

@FXML
public void changeGrade(ActionEvent ae) {
    //code to change grade
    RootController.setGrade(cbxGrade.getValue());
}

@FXML
public void changeGreeting(ActionEvent ae) {
    //update properties for next time welcome
    settingsProps.setProperty("greeting", String.valueOf(tglGreeting.isSelected()));
    try {
        FileOutputStream out = new FileOutputStream(propertiesFile);
        settingsProps.store(out, "greeting toggled");
        out.close();
    } catch(IOException ex) {ex.printStackTrace();}
}

```

Note: properties only store string values so we must convert Booleans into Strings and vice versa.

We will add in the code the change the theme later since that requires CSS and we still have the reset function to add.

We will wrap the code inside the method with an Alert since this choice is quite an important decision to make – all your data to be exiled from the program.

```
@FXML  
public void resetData(ActionEvent ae) {  
    Alert confirmAlert = new Alert(AlertType.CONFIRMATION);  
    confirmAlert.setHeaderText("Are you sure you want to reset your data?");  
    Optional<ButtonType> result = confirmAlert.showAndWait();  
    if(result.isPresent() && result.get() == ButtonType.OK) {  
        dao.reset(); //reset database  
    }  
}
```

Back in the DAO.java class we can create a method which overrides the current database file with one inside the application .jar itself, whenever a new DAO object is initialised by selecting an activity, it will connect to this overridden file. This cannot be reverted so extra care is required – hence the use of the Alert.

```
public void reset() {  
    File database = new File(dbDir + "/" + dbName);  
    try {  
        Files.copy(DAO.class.getResourceAsStream(dbName), database.toPath(), StandardCopyOption.REPLACE_EXISTING);  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

It successfully overrides the file if we have not selected an activity, but if we try an activity and then reset the database, we get this:

```
java.nio.file.FileSystemException: C:\IMProve\improveDB.accdb: The process cannot access the file because it is being used by another process.  
    at sun.nio.fs.WindowsException.translateToIOException(WindowsException.java:86)  
    at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:97)  
    at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:102)  
    at sun.nio.fs.WindowsFileSystemProvider.implDelete(WindowsFileSystemProvider.java:269)  
    at sun.nio.fs.AbstractFileSystemProvider.deleteIfExists(AbstractFileSystemProvider.java:108)  
    at java.nio.file.Files.deleteIfExists(Files.java:1165)  
    at java.nio.file.Files.copy(Files.java:3004)  
    at other.DAO.reset(DAO.java:42)  
    at controller.SettingsController.resetData(SettingsController.java:107)  
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)  
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)  
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)  
    at java.lang.reflect.Method.invoke(Method.java:498)  
    at sun.reflect.misc.Trampoline.invoke(MethodUtil.java:71)  
    at sun.reflect.GeneratedMethodAccessor1.invoke(Unknown Source)  
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)  
    at java.lang.reflect.Method.invoke(Method.java:498)  
    at sun.reflect.misc.MethodUtil.invoke(MethodUtil.java:275)  
    at javafx.fxml.FXMLLoader$MethodHandler.invoke(FXMLLoader.java:1769)  
    at javafx.fxml.FXMLLoader$ControllerMethodEventHandler.handle(FXMLLoader.java:1657)  
    at com.sun.javafx.event.CompositeEventHandler.dispatchBubblingEvent(CompositeEventHandler.java:86)  
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:238)  
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:191)  
    at com.sun.javafx.event.CompositeEventDispatcher.dispatchBubblingEvent(CompositeEventDispatcher.java:59)  
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:58)  
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)  
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)  
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)  
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)  
    at com.sun.javafx.event.EventUtil.fireEventImpl(EventUtil.java:114)  
    at com.sun.javafx.event.EventUtil.fireEvent(EventUtil.java:74)  
    at javafx.event.Event.fireEvent(Event.java:49)  
    at javafx.scene.Node.fireEvent(Node.java:8411)  
    at javafx.scene.control.Button.fire(Button.java:185)  
    at com.sun.javafx.scene.control.behavior.ButtonBehavior.mouseReleased(ButtonBehavior.java:182)  
    at com.sun.javafx.scene.control.skin.BehaviorSkinBase$1.handle(BehaviorSkinBase.java:96)  
    at com.sun.javafx.scene.control.skin.BehaviorSkinBase$1.handle(BehaviorSkinBase.java:89)  
    at com.sun.javafx.event.CompositeEventHandler$NormalEventFilter.recordBubblingEvent(CompositeEventHandler.java:218)
```

With research, this means that any connections used have not been closed. However, this is quite contradictory since we have used try-with-resources blocks every time we get a connection meaning that it will automatically close at the end of the block.

E.g.

```
try {
    Connection conn = dao.getConnection();
    PreparedStatement stmt = conn.prepareStatement("SELECT Title FROM Theory WHERE Grade <=?");
}
```

Even with this, I tried to make sure that the connections were closed, so, I stored the Connection objects into a static collection (static so that all objects of DAO will add to the same list) and then try to close them manually.

```
private static ObservableList<Connection> allConnections = FXCollections.observableArrayList();

public Connection getConnection() { //create a connection to the database
    Connection conn = null;
    try {
        conn = DriverManager.getConnection(dbUrl);
    } catch (SQLException e) {
        e.printStackTrace();
    }
    allConnections.add(conn);
    return conn;
}
```

allConnections.add(conn) adds the connection to the collection, allCollections.

```
public static void closeConnections() {
    for(Connection conn: allConnections) {
        if(conn!=null) {
            try {
                conn.close();
                System.out.println("a connection has been closed");
            } catch (SQLException e) {
                e.printStackTrace();
                System.out.println("already closed");
            }
        }
    }
}
```

This method closes each connection in the collection and prints the status depending on whether it has been closed or already closed (which is expected to be already closed from the try-with-resource). DAO.closeConnections() will be called before dao.reset() to close all the connections prior to resetting it and hopefully free up the file so that it is able to be overridden (capital DAO, since it is a static method).

And then we get the same error...

```

a connection has been closed
java.nio.file.FileSystemException: C:\iMProve\improveDB.accdb: The process cannot access the file because it is being used by another process.

at sun.nio.fs.WindowsException.translateToIOException(WindowsException.java:86)
at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:97)
at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:102)

```

What's extremely surprising is that it closes again (meaning the try-with-resources did not work) and none were already closed in the first place. Even with closing them, we still retrieve the same error saying that the file is in use.

After research on the internet, the problem occurs because our Ucanaccess driver keeps the file (and some resources) open for a short period of time even with our connections closed; this is for performance reasons like opening connections and closing them faster when called in quick successions.

So, we need to call ((UcanaccessConnection) conn).unloadDB(); so that the resources are fully released, and the file is no longer open. To do this in our application, we can just call unloadDB() on any connection once since this method affects the whole database rather than just the Connection.

```

public static void closeConnections() {
    try {
        UcanaccessConnection uConn = (UcanaccessConnection) allConnections.get(0);
        ((UcanaccessConnection) uConn).unloadDB();
    } catch(UcanaccessSQLException e) {
        e.printStackTrace();
    }

    for(Connection conn: allConnections) {
        if(conn!=null) {
            try {
                conn.close();
                System.out.println("a connection has been closed");
            } catch (SQLException e) {
                e.printStackTrace();
                System.out.println("already closed");
            }
        }
    }
}

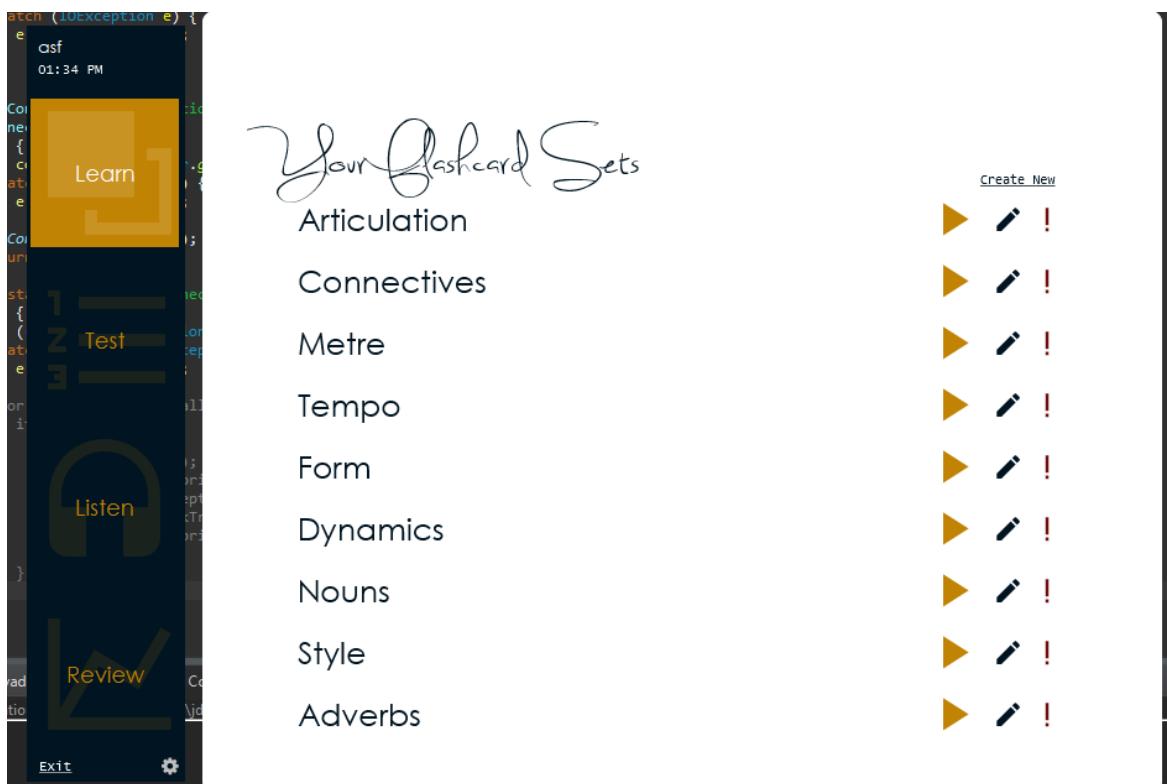
```

This successfully reset our database.

To test this, if someone accidentally deleted every single flashcard set and/or edited them.



And after pressing reset, all the data resets back to its original state so the original flashcard sets return



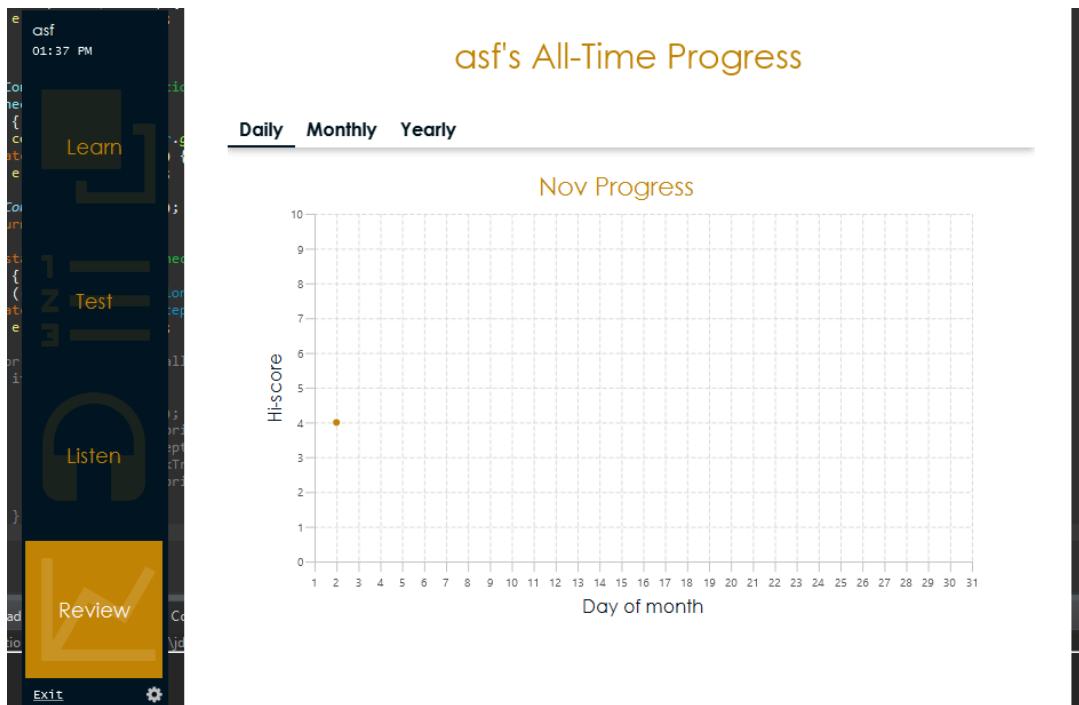
Also, to make sure whether closing the connections were even useful or not (since realising the `close()` method is a no-op method, so when it doesn't apply, i.e the connection is already closed, it does nothing and does not throw an exception), I commented the manual closing.

```

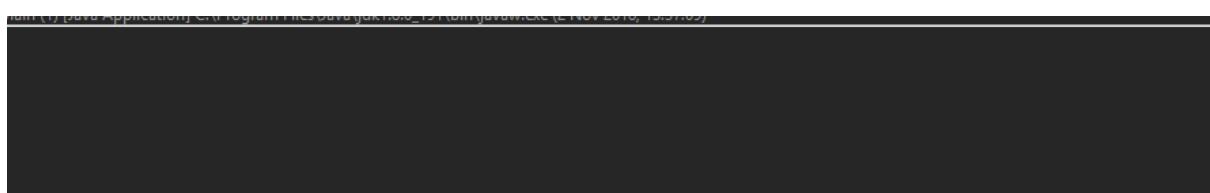
public static void closeConnections() {
    try {
        ((UcanaccessConnection) allConnections.get(0)).unloadDB();
    } catch(UcanaccessSQLException e) {
        e.printStackTrace();
    }
    /*for(Connection conn: allConnections) {
        if(conn!=null) {
            try {
                conn.close();
                System.out.println("a connection has been closed");
            } catch (SQLException e) {
                e.printStackTrace();
                System.out.println("already closed");
            }
        }
    }*/
}

```

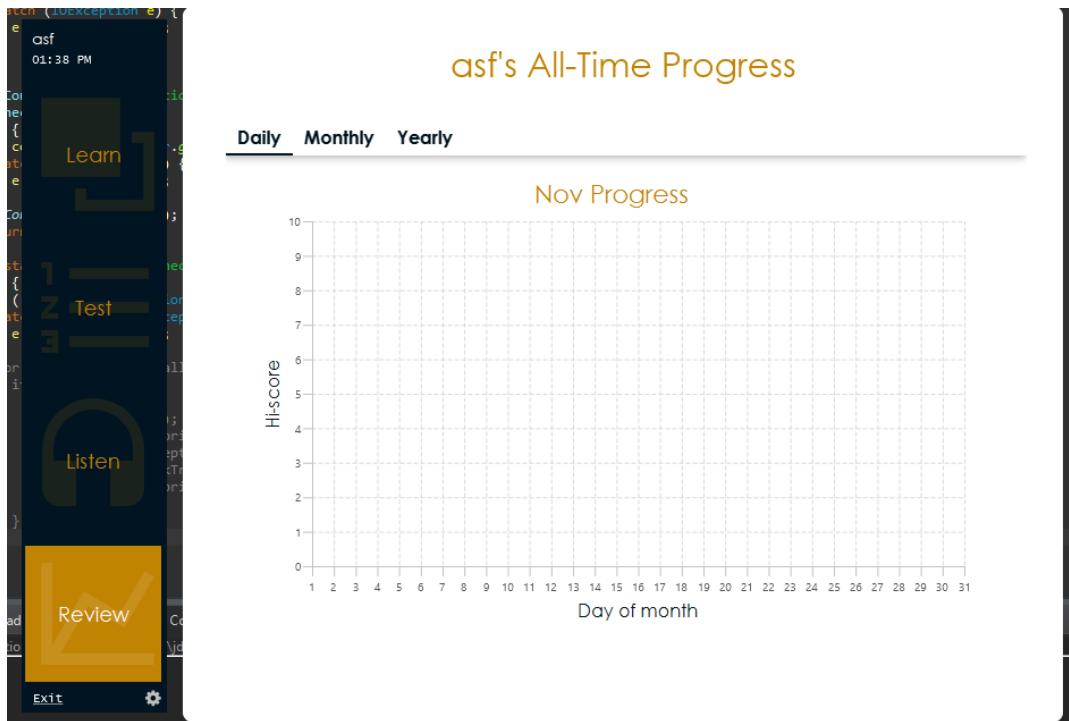
Then, I did another test to validate this, so I headed to the Test feature, completed it, and returned to review (which opens a lot of connections).



By resetting on the settings page, it returned no errors



and all the data was reset.



This means that the connections were already closed by the try-with-resource and we can leave the `closeConnections()` method like this:

```
public static void closeConnections() {
    try {
        ((UcanaccessConnection) allConnections.get(0)).unloadDB();
    } catch(UcanaccessSQLException e) {
        e.printStackTrace();
    }
}
```

However, we further tested this; at first, we had reset the system successfully, but after moving around the program again (and therefore opening new connections), resetting the system a second time caused this:

```

at com.sun.javaev.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
at com.sun.javaev.event.EventUtil.fireEvent(EventUtil.java:74)
at com.sun.javaev.event.EventUtil.fireEvent(EventUtil.java:49)
at javafx.event.Event.fireEvent(Event.java:198)
at javafx.scene.Node.fireEvent(Node.java:8411)
at javafx.scene.control.Button.fire(Button.java:185)
at com.sun.javaev.scene.control.behavior.ButtonBehavior.mouseReleased(ButtonBehavior.java:182)
at com.sun.javaev.scene.control.skin.BehaviorSkinBase$1.handle(BehaviorSkinBase.java:96)
at com.sun.javaev.scene.control.skin.BehaviorSkinBase$1.handle(BehaviorSkinBase.java:89)
at com.sun.javaev.event.CompositeEventHandler$NormalEventRecord.handleBubblingEvent(CompositeEventHandler.java:218)
at com.sun.javaev.event.CompositeEventHandler.dispatchBubblingEvent(CompositeEventHandler.java:80)
at com.sun.javaev.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:238)
at com.sun.javaev.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:191)
at com.sun.javaev.event.CompositeEventDispatcher.dispatchBubblingEvent(CompositeEventDispatcher.java:59)
at com.sun.javaev.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:58)
at com.sun.javaev.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
at com.sun.javaev.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
at com.sun.javaev.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
at com.sun.javaev.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
at com.sun.javaev.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
at com.sun.javaev.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:58)
at com.sun.javaev.event.EventUtil.fireEvent(EventUtil.java:74)
at com.sun.javaev.event.EventUtil.fireEvent(EventUtil.java:54)
at javafx.event.Event.fireEvent(Event.java:198)
at javafx.scene.Scene$MouseHandler.process(Scene.java:3757)
at javafx.scene.Scene$MouseHandler.access$1500(Scene.java:3485)
at javafx.scene.Scene.impl_processMouseEvent(Scene.java:1762)
at javafx.scene.Scene$ScenePeerListener.mouseEvent(Scene.java:2494)
at com.sun.javaev.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:394)
at com.sun.javaev.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:295)
at java.security.AccessController.doPrivileged(Native Method)
at com.sun.javaev.tk.quantum.GlassViewEventHandler.lambda$handleMouseEvent$353(GlassViewEventHandler.java:432)
at com.sun.javaev.tk.quantum.QuantumToolkit.runWithoutRenderLock(QuantumToolkit.java:389)
at com.sun.javaev.tk.quantum.GlassViewEventHandler.handleMouseEvent(GlassViewEventHandler.java:431)
at com.sun.glass.ui.View.handleMouseEvent(View.java:555)
at com.sun.glass.ui.View.notifyMouse(View.java:937)
at com.sun.glass.ui.win.WinApplication._runLoop(Native Method)
at com.sun.glass.ui.win.WinApplication.Lambda$null$147.run(WinApplication.java:177)
at java.lang.Thread.run(Thread.java:748)
Caused by: java.nio.channels.ClosedChannelException
at sun.nio.ch.FileChannelImpl.ensureOpen(FileChannelImpl.java:110)
at sun.nio.ch.FileChannelImpl.force(FileChannelImpl.java:379)
at com.healthmarketscience.jackcess.impl.PageChannel.flush(PageChannel.java:393)
at com.healthmarketscience.jackcess.impl.DatabaseImpl.flush(DatabaseImpl.java:1712)
at net.ucanaccess.jdbc.DBReference.shutdown(DBReference.java:646)
at net.ucanaccess.jdbc.UcanaccessConnection.unloadDB(UcanaccessConnection.java:837)
... 60 more
java.nio.file.FileSystemException: C:\imProve\improveDB.accdb: The process cannot access the file because it is being used by another process.

at sun.nio.fs.WindowsException.translateToIOException(WindowsException.java:86)
at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:97)
at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:102)
at sun.nio.fs.WindowsFileSystemProvider.implDelete(WindowsFileSystemProvider.java:269)

```

The root of our errors, however, was this:

```

net.ucanaccess.jdbc.UcanaccessSQLException: UCAExc:::4.0.4 null
    at net.ucanaccess.jdbc.UcanaccessConnection.unloadDB(UcanaccessConnection.java:841)
    at other.DAO.closeConnections(DAO.java:61)
    at controller.SettingsController.resetData(SettingsController.java:135)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

```

In the DAO.java class, line 61,

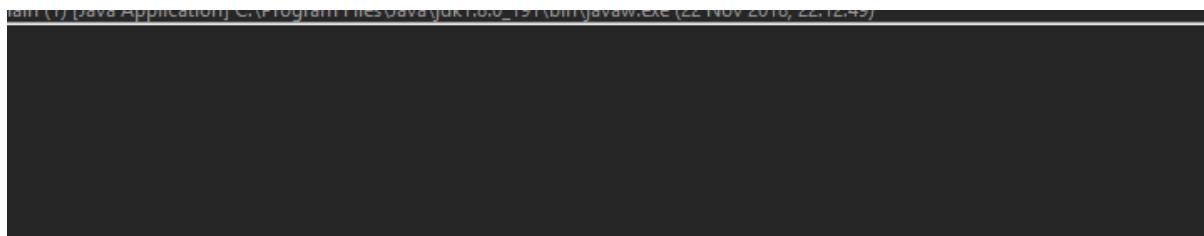
```

57
58  public static void closeConnections() {
59      try {
60          if(!allConnections.isEmpty()) {
61              ((UcanaccessConnection) allConnections.get(0)).unloadDB();
62          }
63      } catch(SQLException e) {
64          e.printStackTrace();
65      }
66  }

```

The connection was apparently null. After rigorous thought, I concluded that the connections would be null after unloading them the first time and after these connections were unloaded, they were left in the allConnections collection, this meant that when new connections were added, allConnections.get(0) would return the previously old, unloaded, null connection. To circumvent this, we cleared the ObservableArrayList so that we could start with a fresh new collection and only contain newly opened (and closed yet not unloaded) connections.

This fully solved our problem and returned no errors when doing resetting multiple times after opening new connections.



Next are our themes.

What we can do for themes is group all the CSS used in the styling of the whole program and put the whole code into one stylesheet. We can use classes for similarly styled controls like apnListen or apnTest. We will use the file default_chamber.css for our default theme.

Code for default_chamber.css:

```
1 /*General Styles */
2 .backgroundPanel {
3     -fx-background-color: #001421;
4     -fx-background-radius: 10;
5 }
6
7 .backgroundPane2 {
8     -fx-background-color: white;
9     -fx-background-radius: 10;
10 }
11
12 .titleText1 {
13     -fx-fill: white;
14 }
15
16 .titleText2 {
17     -fx-fill: #001421;
18 }
19
20 .titleText3 {
21     -fx-fill: #c18303;
22 }
23 .subtitleText1 {
24     -fx-fill: white;
25 }
26 .subtitleText2{
27     -fx-fill: #001421;
28 }
29 .specialButton1 {
30     -fx-background-color: #c18303;
31     -fx-fill: white;
32     -fx-background-radius: 20;
33 }
34
35 .specialButton2 {
36     -fx-background-color: white;
37     -fx-fill: #c18303;
38     -fx-background-radius: 20;
39 }
40
41 .selectButton {
42     -fx-background-color: white;
43     -fx-background-radius: 10;
44     -fx-text-fill: #c18303;
45     -fx-button-type: RAISED;
46 }
47 /*Root Styling */
48 #navigationBar {
49     -fx-background-color: #001421;
50 }
51
52 .activityButton {
53     -fx-text-fill: #c18303;
```

```

54 }
55 .activityIcon {
56   -fx-fill: #262b1a;
57 }
58 .activityPane {
59   -fx-background-color: #001421;
60 }
61 /*Scroll Pane Code*/
62
63 /*scroll bar track */
64 .scroll-bar:vertical > .track-background{
65   -fx-background-color: #FFFFFF;
66   -fx-background-insets: 0 0;
67 }
68 /*dragging thumb */
69 .scroll-bar:vertical > .thumb {
70   -fx-background-color: #c18303;
71   -fx-background-insets: 0 0;
72   -fx-background-radius: 1.0;
73 }
74
75 /* up and down button padding */
76 .scroll-bar:vertical > .increment-button, .scroll-bar:vertical > .decrement-button {
77   -fx-padding: 5 2 5 2;
78 }
79 /*up down buttons*/
80 .scroll-bar > .increment-button, .scroll-bar > .decrement-button, .scroll-bar:hover > .increment-button, .scroll-bar:hover > .decrement-button {
81   -fx-background-color: transparent;
82 }
83 .scroll-bar > .increment-button > .increment-arrow, .scroll-bar > .decrement-button > .decrement-arrow {
84   -fx-background-color: rgb(43, 80, 104);
85 }
86
87 /* up arrow shape */
88 .scroll-bar:vertical > .increment-button > .increment-arrow {
89   -fx-shape: "M298 426h428l-214 214z";
90 }
91
92 /* down arrow shape */
93 .scroll-bar:vertical > .decrement-button > .decrement-arrow {
94   -fx-shape: "M298 598l214-214 214 214h-428z";
95 }
96
97 /*scroll pane itself */
98 .scroll-pane {
99   -fx-background-insets: 0;
100  -fx-padding: 0;
101 }
102 .scroll-pane:focused {
103   -fx-background-insets: 0;
104 }
105 .scroll-pane .corner {
106   -fx-background-insets: 0;
107 }
108 .scroll-pane > .viewport {
109   -fx-background-color: transparent;
110 }
111
112
113
114
115 /*Learn Section */
116 .navigateButton {
117   -fx-background-color: white;
118   -fx-font-fill: #c18303;
119   -fx-background-radius: 40;
120 }
121
122 #flashcardPane {
123   -fx-background-color: white;
124 }
125
126 #termText {
127   -fx-fill: #001421;
128 }
129
130 #defText {
131   -fx-fill: #c18303;
132 }

```

```

133
134 #backIcon {
135     -fx-fill: white;
136 }
137
138 .playIcon {
139     -fx-fill: #c18303;
140 }
141
142 #editIcon {
143     -fx-fill: #001421;
144 }
145
146 #deleteIcon {
147     -fx-fill: #6f0f0f;
148 }
149
150 #addButton {
151     -fx-background-color: #001421;
152     -fx-background-radius: 5;
153 }
154
/*Listen Section */
156 #musicSlider .track {
157     -fx-background-color: #ffffff;
158 }
159
160 #musicSlider .thumb {
161     -fx-background-color: #c18303;
162     visibility: hidden;
163 }
164
165 #musicSlider .colored-track {
166     -fx-background-color: #c18303
167 }
168
169 #musicSlider .animated-thumb {
170     -fx-background-color: #001421;
171 }
172
173 #musicSlider .slider-value {
174     -fx-stroke: #c18303;
175 }
/*Test Section */
177 #testProgressbar > .bar {
178     -fx-background-color: #c18303
179 }
/*Review Section */
181 #graphTabpane .tab-header-area .tab-selected-line {
182     -fx-background-color: #001421;
183 }
184
185 #graphTabpane .tab-header-area .tab-header-background {
186     -fx-background-color: #ffffff;
187 }
188
189 #graphTabpane .tab-label {
190     -fx-text-fill: #001421;
191     -fx-font-family: 'Century Gothic';
192 }
/*Chart code */
194 .chart-title {

```

```

195     :fx-text-fill: #c18303;
196     :fx-font-size: 22px;
197     :fx-font-family: 'Century Gothic';
198 }
199 @.axis-label {
200     :fx-text-fill: #001421;
201     :fx-font-family: 'Century Gothic';
202     :fx-font-size: 16px;
203 }
204 @.axis {
205     :fx-text-fill: #001421;
206     :fx-font-family: 'Century Gothic';
207     :fx-font-size: 12px;
208 }
209 @.chart {
210     :fx-padding: 20px;
211 }
212 @.chart-plot-background {
213     :fx-background-color: transparent;
214 }
215 @.chart-alternative-row-fill {
216     :fx-fill: transparent;
217     :fx-stroke: transparent;
218     :fx-stroke-width: 0;
219 }
220 @.chart-series-line {
221     :fx-stroke-width: 2px;
222     :fx-fillet: null;
223 }
224 @.chart-line-symbol {
225     :fx-background-color: #c18303;
226     :fx-background-radius: 5px;
227     :fx-width: 3px;
228 }
229 .default-color0.chart-line-symbol { :fx-background-color: #c18303; }
230 .default-color0.chart-series-line { :fx-stroke: #c18303; }
231
232
233 /*Settings Section */
234 @.settingsCbox .cell {
235     :fx-text-fill: #c18303;
236 }
237 @.settingsCBox {
238     :fx-text-fill: #c18303;
239     :fx-focus-color: #c18303;
240     :fx-unfocus-color: #b5b5b5;
241 }
242 @#titleLine {
243     :fx-stroke: #c18303;
244 }
245 @.ifx-toggle-button {
246     :ifx-toggle-color: #af7600;
247     :ifx-untoggle-color: #FAFAFA;
248     :ifx-toggle-line-color: #c18303;
249     :ifx-untoggle-line-color: #999999;
250     :ifx-size: 12.0;
251     :ifx-disable-visual-focus: false;
252     :fx-fill: #c18303;
253 }
254 @#nameField {
255     :fx-text-fill: #c18303;
256     :fx-focus-color: #c18303;
257     :fx-unfocus-color: #b5b5b5;
258 }

```

Upon initialisation, we need to set this as the stylesheet of the whole root scene., so let's go to the RootController.java class.

```
public void initialize() {
    Platform.runLater(()->{
        //get properties
        Properties settingsProps = getProperties();
        String storedTheme = settingsProps.getProperty("theme", "Default Chamber");
        boolean useGreeting = "true".equals(settingsProps.getProperty("greeting", "true"));
        //set theme
        setCSS(storedTheme);

        //set greeting
        if(useGreeting) {
            int hour = Integer.parseInt(LocalTime.now().format(DateTimeFormatter.ofPattern("HH")));
            String greeting;
            if(hour>=5 & hour<12)
                greeting = "Good Morning,";
            else if(hour>=12 & hour<18)
                greeting = "Good Afternoon,";
            else if(hour>=18 & hour<23)
                greeting = "Good Evening,";
            else
                greeting = "Good Night,";
            txtWelcome.setText(greeting);
        }

        //take off listen section if not grade 8
        if(grade!=8) {
            vbxActivities.getChildren().remove(spnListen);
        }
        apnRoot.requestFocus();
        //automatically update time
        startTime();
    });
}
```

Here we load up the properties with the `getProperties()` method and set the theme using `setCSS()` depending on the stored theme if any, if not then use our default: Default Chamber. And code below sets the custom greeting if required.

We have simple code to load up the properties in `getProperties()` and then we also get the filename of our themes to be set onto the scene.

```
public Properties getProperties() {
    final String path = "C:\\iMProve\\settings.properties";
    Properties settingsProps = new Properties();

    try {
        File propertiesFile = new File(path);
        propertiesFile.createNewFile(); //will create new file if does not exist, if exists then no-op

        settingsProps.load(new FileInputStream(propertiesFile));
    } catch(IOException ex) {ex.printStackTrace();}

    return settingsProps;
}
private void setCSS(String theme) {
    RootController.theme = theme;
    String fileName = "view/";
    switch(theme) {
        case "Ice Crystal": fileName += "ice_crystal_theme.css";
        break;
        case "Default Chamber": fileName += "default_theme.css";
        break;
        case "Minimal Lightning": fileName += "minimal_lightning_theme.css";
        break;
    }
    apnRoot.getScene().getStylesheets().add(fileName);
}
```

We can also create a static method for setting the theme from another object (i.e. the `SettingsController`).

```
public static void setTheme(String theme) {
    RootController.theme = theme;
}
```

Our `changeNavigationColours()` method will need to be altered slightly also since we are using different themes so the selection would be different. Upon trial and improvement, using previous code of `button.setTextFill()` or `icon.setFill()` directly does not change the colours, since the CSS is at priority, so, we must call `icon.setStyle()` and `button.setStyle()`.

```
public void changeNavigationColours(StackPane spn, MaterialDesignIconView icn, JFXButton btn) {
    for(StackPane pane: Arrays.asList(spnLearn, spnTest, spnListen, spnReview)) {
        if(!pane.equals(spn))
            pane.setStyle("-fx-background-color: " + getColourCode(0));
        else
            pane.setStyle("-fx-background-color: " + getColourCode(1));
    }
    for(MaterialDesignIconView icon: Arrays.asList(icnLearn, icnTest, icnListen, icnReview)) {
        if(!icon.equals(icn)) {
            icon.setStyle("-fx-fill: " + getColourCode(4));
        } else {
            icon.setStyle("-fx-fill: " + getColourCode(3));
        }
    }
    for(JFXButton button: Arrays.asList(btnLearn, btnTest, btnListen, btnReview)) {
        if(!button.equals(btn))
            button.setStyle("-fx-text-fill: " + getColourCode(1));
        else
            button.setStyle("-fx-text-fill: " + getColourCode(2));
    }
}
```

Notice we use `getColourCode()` with different value parameters each time, this allows us to give a different colour depending on the theme. Each value parameter corresponds to a different colour relating to a certain node.

```
public String getColourCode(int option) {
    String colourCode = "";
    switch(theme) {
        case "Default Chamber":
            switch(option) {
                case 0: colourCode = "#001421"; //unselected background
                break;
                case 1: colourCode = "#c18303"; //selected background/ unselected text
                break;
                case 2: colourCode = "#ffffff"; //selected text
                break;
                case 3: colourCode = "#daa34b"; //selected icon
                break;
                case 4: colourCode = "#262b1a"; //unselected icon
                break;
            }
            break;
        case "Ice Crystal":
            switch(option) {
                //different colour codes
            }
            break;
    }
    return colourCode;
}
```

Also, noting previously, we used the Parent class rather than the respective control classes, this was not a problem then, more of an advantage, but now is a problem. Parent abstracts the styles of each control and so you cannot change the styles of a control if it is linked to parent, so, we replace the Parent with the respective node.

e.g.

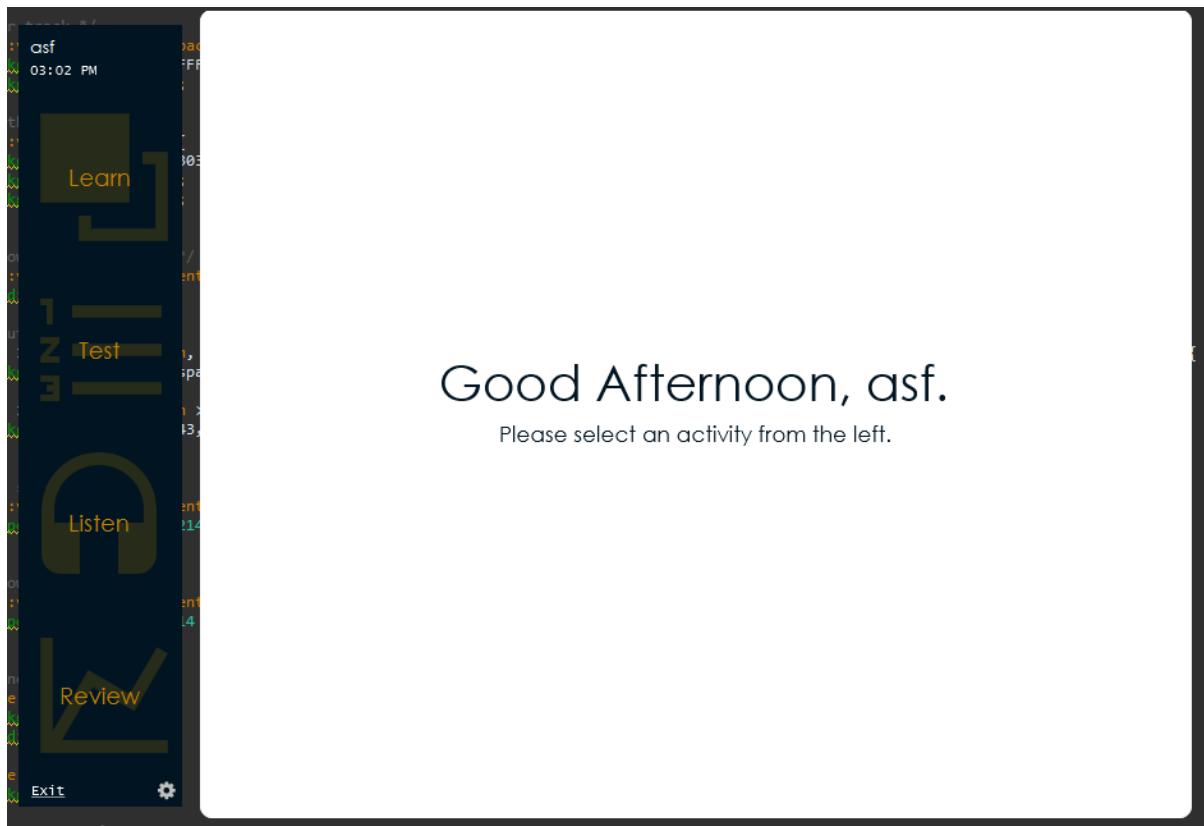
Before

```
public void goLearn(ActionEvent e) {  
    Parent root = null;  
    FXMLLoader loader = new FXMLLoader(getClass().getResource("Learn.fxml"));
```

After

```
public void goLearn(ActionEvent e) {  
    AnchorPane root = null;  
    FXMLLoader loader = new FXMLLoader(getClass().getResource("Learn.fxml"));
```

This achieves the same effect as before (using Default Chamber).



In the `SettingsController.java`, we need to change the theme every time `cbxTheme` changes value so we clear all previous stylesheets on the scene and replace it with the new theme (depending on `cbxTheme`'s value), then we store this theme into the properties file so that it remembers for next time.

```

@FXML
public void changeTheme(ActionEvent ae) {
    //code to change theme
    Scene rootScene = apnSettings.getScene();
    rootScene.getStylesheets().clear();
    rootScene.getStylesheets().add(getCSS(cbxTheme.getValue()));
    //change properties
    settingsProps.setProperty("theme", cbxTheme.getValue());
    try {
        FileOutputStream out = new FileOutputStream(propertiesFile);
        settingsProps.store(out, "theme changed");
        out.close();
    } catch(IOException ex) {ex.printStackTrace();}
    RootController.setTheme(cbxTheme.getValue());
}

```

Upon testing this, there were a few errors. The direct changes of an attribute of a control, e.g. `Button.setTextFill(Color c)` would not change anything since the root stylesheet would override it. So, we can call `setStyle()` and change the style directly. For example, in `TestQuizController.java`:

```

answerButtons[answer].setStyle("-fx-background-color: #6f0f0f; -fx-text-fill: white;"); //answer
answerButtons[correctAnswer].setStyle("-fx-background-color: #0cb794; -fx-text-fill: white;"); /

```

Also, the slider in the `listen_quiz.fxml` view did not change after answering the question and reveal the thumb.



This means the changes caused by the following line of code has not taken place, this indicates that it has been overridden by the root stylesheet, `default_theme.css`.

```

sldSeek.getStylesheets().add("view/slider-enable.css");

```

So, we can remove the CSS code for `#musicSlider` (`sldSeek`'s CSS id) and place it into a separate CSS document called `slider_disable.css` (because it is used when the slider is disabled), like so:

```

1● #musicSlider .track {
2     -fx-background-color: #ffffff;
3 }
4
5● #musicSlider .thumb {
6     -fx-background-color: #c18303;
7     visibility: hidden;
8 }
9
10● #musicSlider .colored-track {
11     -fx-background-color: #c18303
12 }
13
14● #musicSlider .animated-thumb {
15     -fx-background-color: #001421;
16 }
17
18● #musicSlider .slider-value {
19     -fx-stroke: #c18303;
20 }

```

And for slider_enable.css:

```

1
2● #musicSlider .thumb {
3     visibility: visible;
4 }
5

```

We can add this stylesheet onto sldSeek at the start of initialisation of ListenQuizController

```

public void initData(RootController rootController) {
    this.rootController = rootController;

    sldSeek.getStylesheets().add("view/slider-disable.css");

    getAuralTest();
    syncAudio();
}

```

And in our allowSeeking() method of ListenQuizController.java, we can add slider_enable.css on top of slider_disable.css so that it overrides the visibility of the thumb of sldSeek to visible.

```

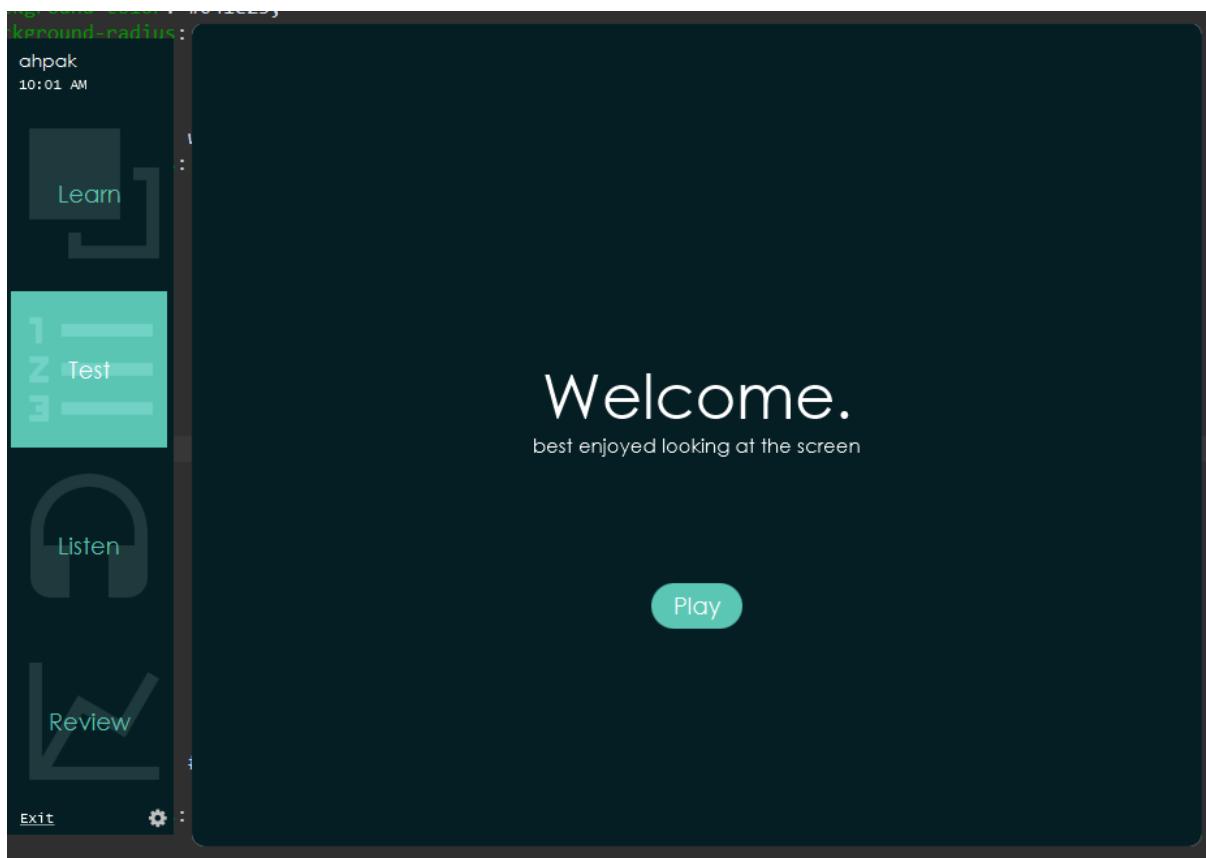
Platform.runLater(()->{
    sldSeek.setDisable(false);

    sldSeek.getStylesheets().add("view/slider-enable.css");

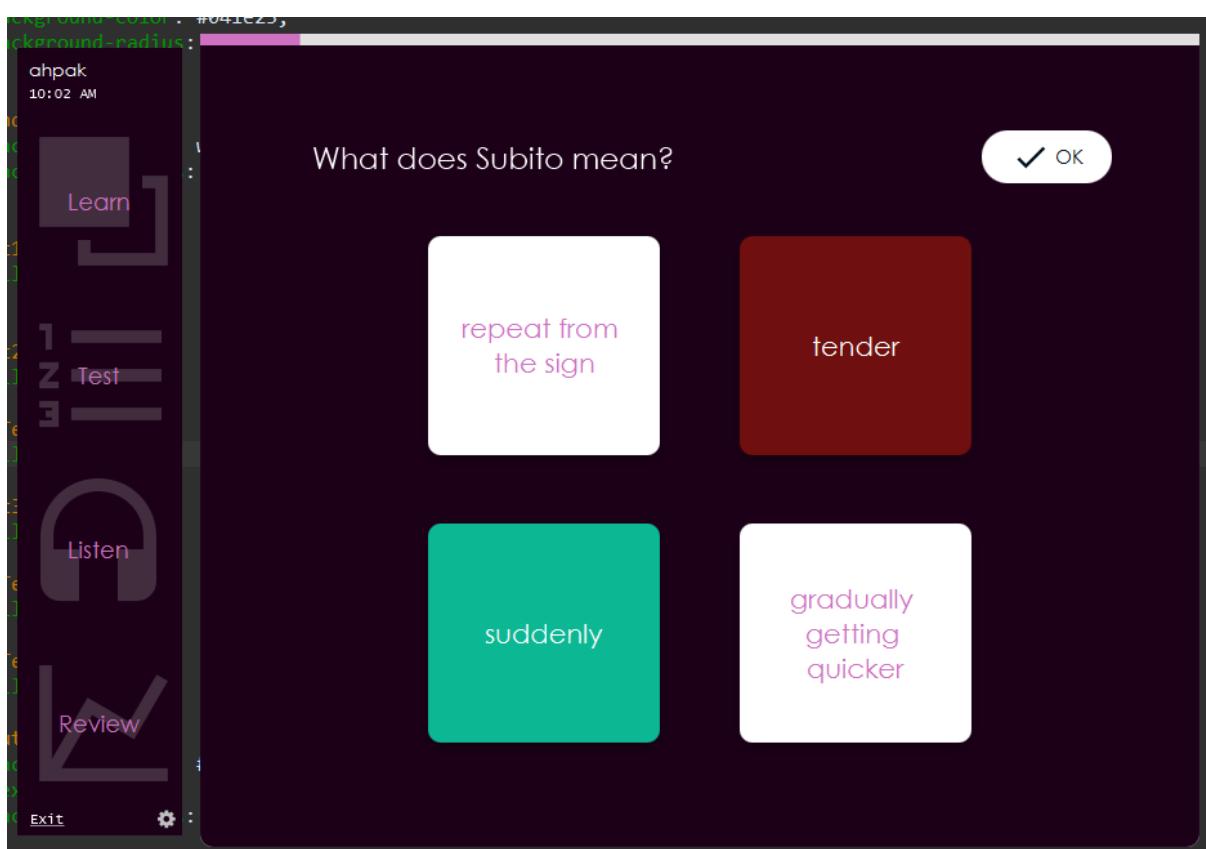
    iconPlayPause.setVisible(true);
});

```

After creating multiple stylesheet themes using the same structure as default_chamber.css, but with different colour schemes, we have made crystal_sea.css...



and rose_petal.css



Note that we have not highlighted the navigation bar for the selected activity so that we can abuse the aesthetics to the most of our ripple effects on the JFXButtons.



Also, a secret easter egg theme `panda_special.css`



This uses images to style the backgrounds rather than `-fx-background-colour`

```
-fx-background-image: url("/resources/img/panda-background1.png");
-fx-background-repeat: stretch;
-fx-background-position: center center;
```

Some methods will have to be adapted for different themes, for example, the highlighting selected activities in `changeNavigationColours()`.

```
public void changeNavigationColours(StackPane spn, MaterialDesignIconView icn, JFXButton btn) {
    for(StackPane pane: Arrays.asList(spnLearn, spnTest, spnListen, spnReview)) {
        if(!pane.equals(spn))
            pane.setStyle("-fx-background-color: " + getColourCode(0));
        else
            pane.setStyle("-fx-background-color: " + ((theme.equals("Default Chamber") | theme.equals("Crystal Sea")) ? getColourCode(1) : "transparent"));
    }
    for(MaterialDesignIconView icon: Arrays.asList(icnLearn, icnTest, icnListen, icnReview)) {
        if(!icon.equals(icn))
            icon.setStyle("-fx-fill: " + getColourCode(4));
        else
            icon.setStyle("-fx-fill: " + getColourCode(3));
    }
    for(JFXButton button: Arrays.asList(btnLearn, btnTest, btnListen, btnReview)) {
        if(!button.equals(btn))
            button.setStyle("-fx-text-fill: " + ((theme.equals("Default Chamber") | theme.equals("Crystal Sea")) ? getColourCode(1) : "")); //doesn't change text colour
        else
            button.setStyle("-fx-text-fill: " + getColourCode(2));
    }
}
```

We can use the `getColourCode()` method with a parameter to determine the colour of the theme we require. Also, we only change the colours if the theme is Default Chamber or Crystal Sea since the ripple effect works better for the other two themes: Rose Petal and Panda Special.

In our `getColourCode()` method we used a nested *switch* selection

```
public static String getColourCode(int option) {
    String colourCode = "";
    switch(theme) {
        case "Default Chamber":
            switch(option) {
                case 0: colourCode = "#001421"; //unselected background
                break;
                case 1: colourCode = "#c1803"; //selected background/ unselected text
                break;
                case 2: colourCode = "#ffffff"; //selected text
                break;
                case 3: colourCode = "#daa34b"; //selected icon
                break;
                case 4: colourCode = "#262b1a"; //unselected icon
                break;
            }
            break;
    }
}
```

One is for the theme, one is for the colour we require determined by the option. Each theme has a different set of colourCode values. However, for Rose Petal and Panda Special, we only specify the first two option values 0 and 1, the other 3 can be left blank since nothing will change in the style if we change the colour to “” (nothing), so the colours will stay as stated in the theme stylesheet.

```

case "Rose Petal":
    switch(option) {
        case 0: colourCode = "transparent";
        break;
        case 1: colourCode = "#ce71c2";
        break;
        //don't need to change selected activities
    }
    break;
case "Panda Special":
    switch(option) {
        case 0: colourCode = "transparent";
        break;
        case 1: colourCode = "#3d3d3d";
        break;
        //as above
    }
    break;
}

```

After moving the code to store the properties to a separate method, `saveProperties()` to make the code cleaner and reusable...

```

private void saveProperties(String comment) {
    try (OutputStream out = Files.newOutputStream(propertiesFile.toPath(), StandardOpenOption.TRUNCATE_EXISTING)) {
        settingsProps.store(out, comment);
        out.close();
    } catch(IOException ex) {ex.printStackTrace();}
}

```

we can use it in our methods to change the theme.

```

@FXML
public void changeTheme(ActionEvent ae) {
    //code to change theme
    setTheme(cbxTheme.getValue());
}

@FXML
public void pandaTheme(MouseEvent me) {
    settingsProps.setProperty("panda unlocked", "true");
    if(!cbxTheme.getItems().contains("Panda Special"))cbxTheme.getItems().add("Panda Special");
    setTheme("Panda Special");
}

public void setTheme(String theme) {
    Scene rootScene = apnSettings.getScene();
    rootScene.getStylesheets().clear();
    rootScene.getStylesheets().add(getCSS(theme));
    //change properties
    settingsProps.setProperty("theme", theme);
    saveProperties("theme changed");

    RootController.setTheme(theme);
}

```

A lot of the code from `changeTheme()` has been moved to a method called `setTheme()` so that we can reuse it for our easter egg theme Panda Special. The easter egg theme is unlocked by clicking the JFXSpinner in `settings.fxml` view which calls the `pandaTheme()` method.

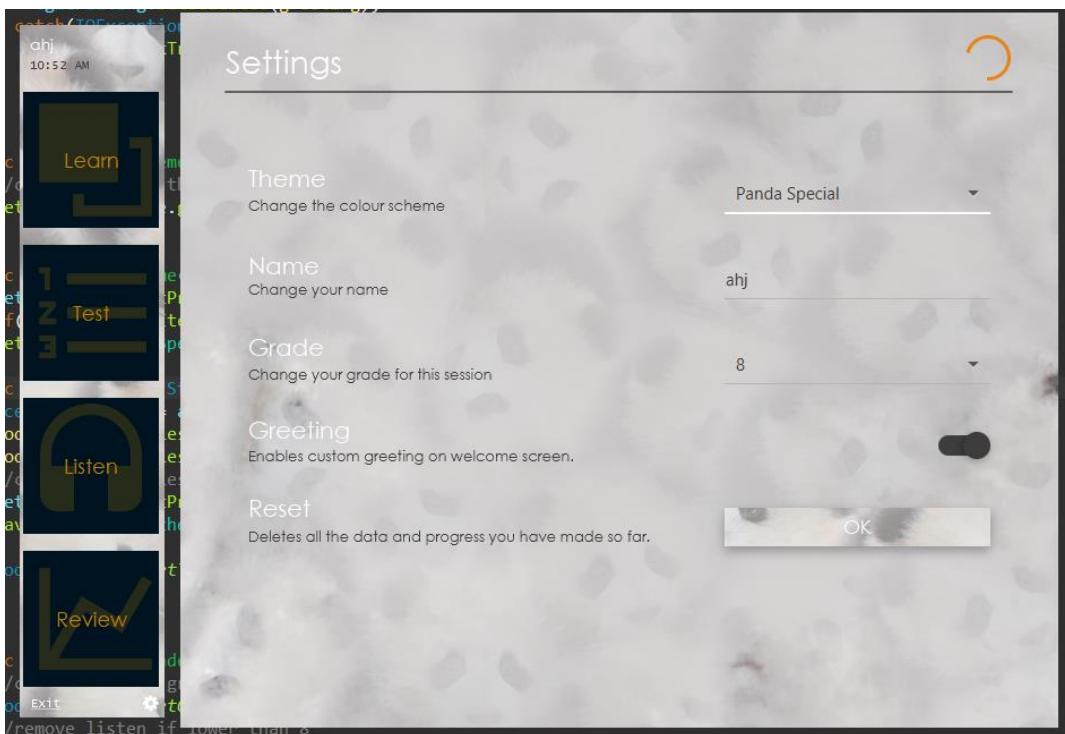


Once this found, it is permanently unlocked and can be chosen from cbxTheme. This can be persistent by adding an extra line to loadValues() checking whether the user has unlocked the Panda Special theme or not.

```
//get the properties
String theme = settingsProps.getProperty("theme", "Default Chamber");
boolean greeting = "true".equals(settingsProps.getProperty("greeting", "true")); //if t
boolean panda = "true".equals(settingsProps.getProperty("panda unlocked", "false"));

//set control values
if(panda)cbxTheme.getItems().add("Panda Special");
cbxTheme.setValue(theme);
tglGreeting.setSelected(greeting);
```

However, upon testing, when switching through themes, the activity buttons stayed the same colour scheme as the previous theme.



This can be fixed with a similar solution to when we switched to settings.fxml.

```
public void setTheme(String theme) {
    Scene rootScene = apnSettings.getScene();
    rootScene.getStylesheets().clear();
    rootScene.getStylesheets().add(getCSS(theme));
    //change properties
    settingsProps.setProperty("theme", theme);
    saveProperties("theme changed");

    RootController.setTheme(theme);
    rootController.changeNavigationColours(null, null, null);
}
```

By calling changeNavigationColours(null, null, null) so that all the buttons are unselected but with a new theme.

Also, some of the methods in `SettingsController.java` needed to be updated after noticing a few issues.

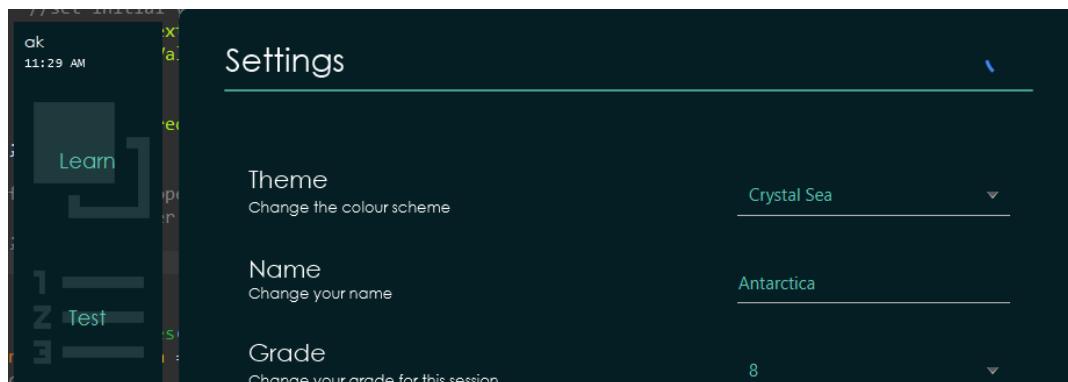
For our `resetData()` method, we needed to reset the `.properties` file and also change the state of some controls back to the default.

```
@FXML  
public void resetData(ActionEvent ae) {  
    Alert confirmAlert = new Alert(AlertType.CONFIRMATION);  
    confirmAlert.setHeaderText("Are you sure you want to reset your data?");  
    Optional<ButtonType> result = confirmAlert.showAndWait();  
    if(result.isPresent() && result.get() == ButtonType.OK) {  
        DAO.closeConnections();  
        dao.reset(); //reset database  
        settingsProps.clear(); //reset settings  
        saveProperties("reset");  
  
        cbxTheme.getItems().remove("Panda Special");  
        tglGreeting.setSelected(true);  
        setTheme("Default Chamber");  
        cbxTheme.setValue("Default Chamber");  
    }  
}
```

We can shorten the `changeGreeting()` method by using our `saveProperties()` method also.

```
@FXML  
public void changeGreeting(ActionEvent ae) {  
    //update properties for next time welcome  
    settingsProps.setProperty("greeting", String.valueOf(tglGreeting.isSelected()));  
    saveProperties("greeting toggled");  
}
```

Additionally, if the user clicked off `fldName` after entering the name, the changes would not take place, only takes place when user presses enter (because `changeName` was binded to `onAction` of `fldName`).

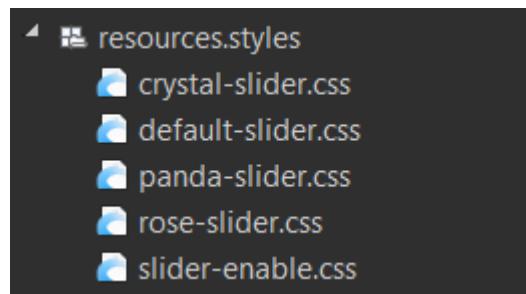


Notice that we start with a random name but when we change it to 'Antarctica' and click off `fldName`, no change is made to the random name in the navigation bar.

So, like the login screen, we can add a listener to the textProperty of fldName and change the name every time the text inside the JFXTextField changes.

```
fldName.textProperty().addListener((o, ov, nv) -> {
    rootController.setName(nv);
});
```

To continue checking whether the themes were compatible with other sections and activities, everything was fine except in listen_quiz.fxml where the slider was individually styled. So, we can create separate styles for each theme respective of their colour themes (to replace slider.disable.css)



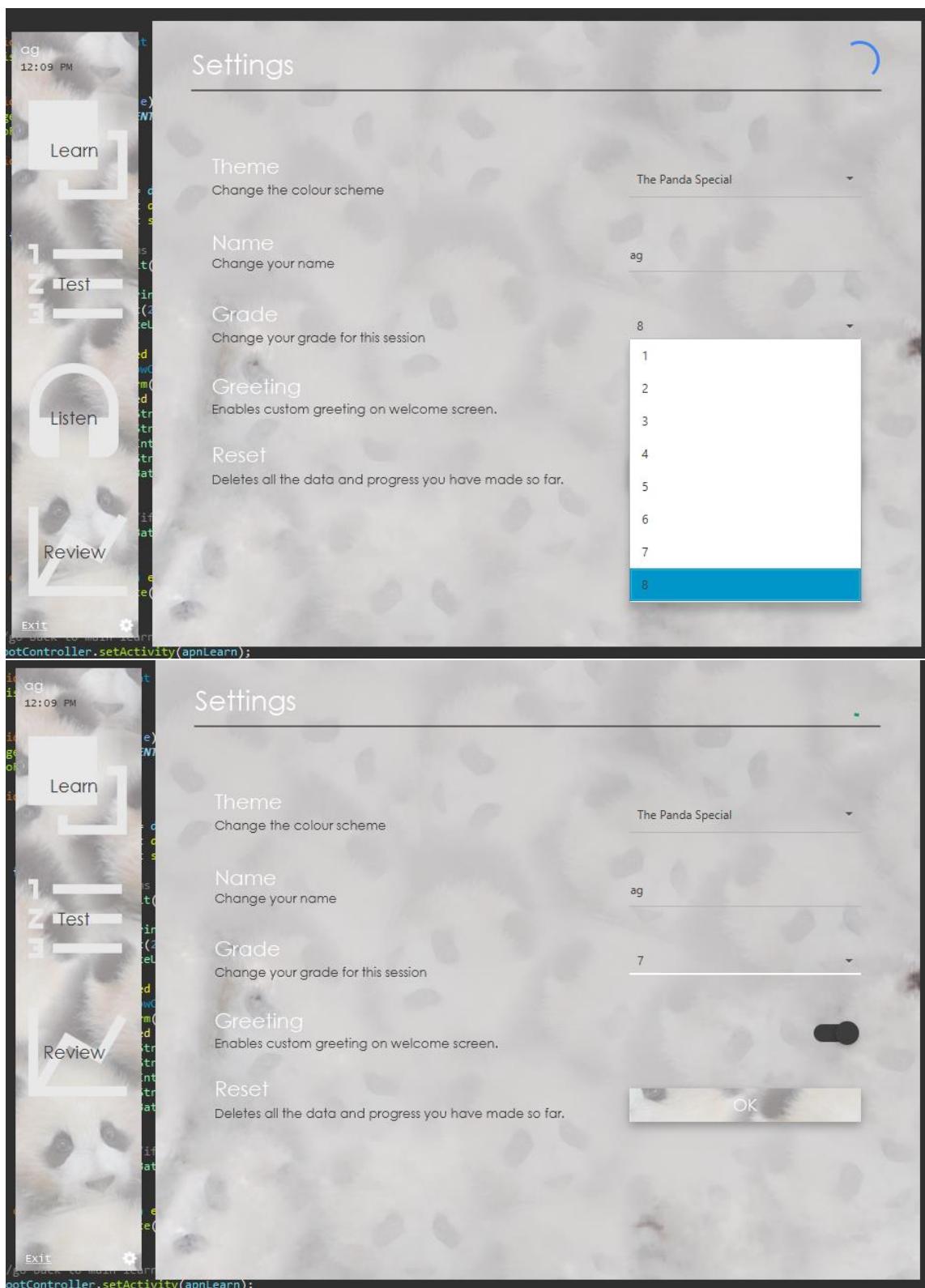
Note: we do not need to have different versions of slider-enable.css since all it does is change the visibility of the thumb to visible independent of the theme.

Furthermore, making sure the SettingsController is fully functional, we need to change the UI depending on the grade as well.

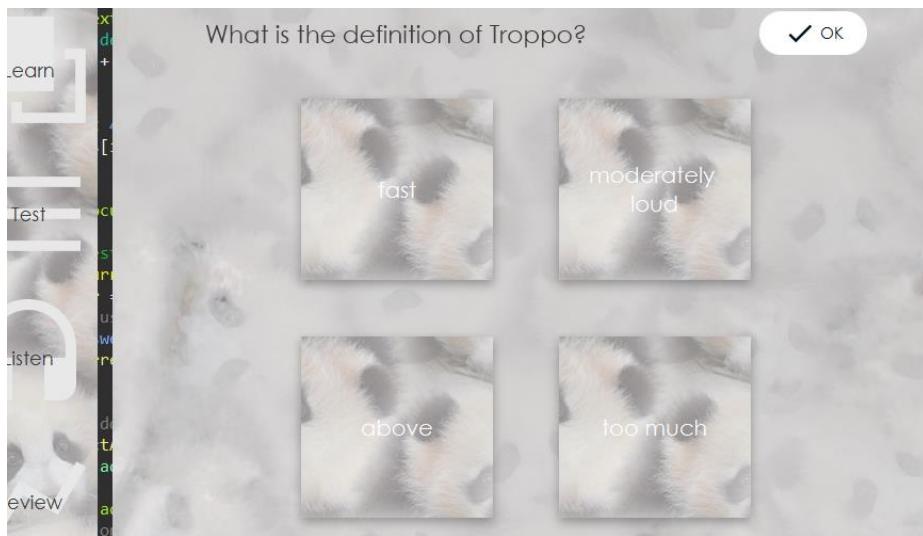
```
@FXML
public void changeGrade(ActionEvent ae) {
    //code to change grade
    RootController.setGrade(cbxGrade.getValue());
    //remove listen if lower than 8
    if(cbxGrade.getValue()<8) {
        rootController.removeListen();
    } else {
        rootController.addListen();
    }
}
```

This will remove/add the listen option depending on the user's new grade and in RootController.java, we define the methods to add or remove the activity from the navigation bar.

```
public void removeListen() {
    vbxActivities.getChildren().remove(spnListen);
}
public void addListen() {
    if(!vbxActivities.getChildren().contains(spnListen))
        vbxActivities.getChildren().add(spnListen);
}
```



However, when trying the test feature using The Panda Special theme, the colours do not change after answering the question.



This can possibly be because the background image is covering the colour and so cannot be shown. So, we can edit the code in `TestQuizController.java` so that the `background-image` is removed/added (using CSS) when needed to.

```

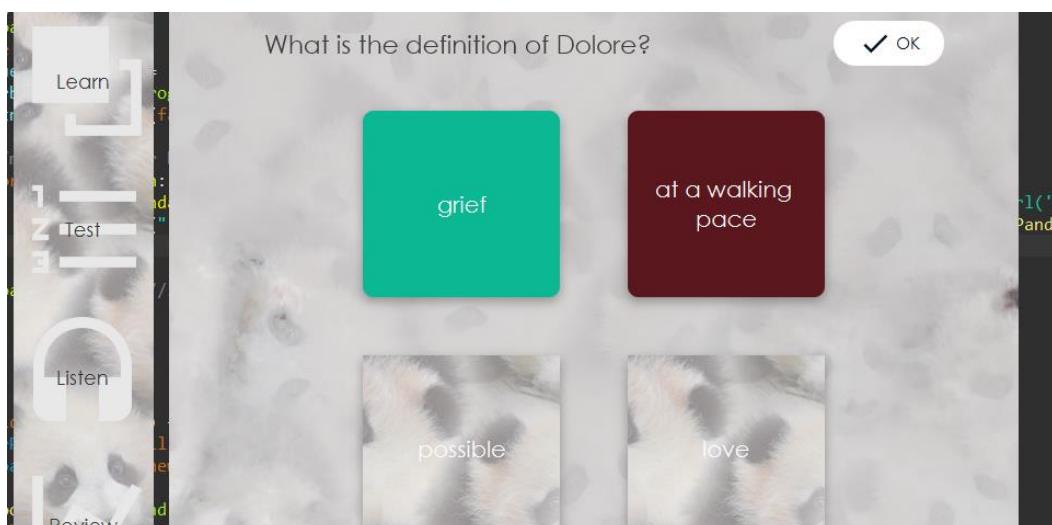
public void answerQuestion(int answer) {
    TheoryQuestion currentQuestion = theoryQuestionList.get(questionIndex);
    int correctAnswer = currentQuestion.getAnswerIndex();
    boolean pandaTheme = RootController.getTheme().equals("The Panda Special");
    //change colours used for feedback
    answerButtons[answer].setStyle("-fx-background-color: #6f0f0f; -fx-text-fill: white;" + (pandaTheme? "-fx-background-image: null": ""));
    answerButtons[correctAnswer].setStyle("-fx-background-color: #0cb794; -fx-text-fill: white;" + (pandaTheme? "-fx-background-image: null": ""));

@FXML
public void nextQuestion(ActionEvent e) {
    if(++questionIndex==10) {
        loadResults();
    } else {
        questionAnswered= false;
        prbProgress.setProgress((questionIndex)/10.0); //increase progress
        btnOk.setVisible(false); //hide ok
    }

    //make all answer boxes white again with orange text
    for(@FXMLButton btn: answerButtons) {
        String addPanda = RootController.getTheme().equals("The Panda Special") ? "; -fx-background-image: url('resources/img/panda-background1.png');": "";
        btn.setStyle("-fx-background-color: #ffffff; -fx-text-fill:" + RootController.getColourCode(1) + addPanda);
    }
}

```

Which now solves the problem.



Code for SettingsController.java:

```
1 package controller;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.io.InputStream;
6 import java.io.OutputStream;
7 import java.nio.file.Files;
8 import java.nio.file.StandardOpenOption;
9 import java.util.Optional;
10 import java.util.Properties;
11
12 import com.jfoenix.controls.JFXComboBox;
13 import com.jfoenix.controls.JFXTextField;
14 import com.jfoenix.controls.JFXToggleButton;
15
16 import javafx.application.Platform;
17 import javafx.event.ActionEvent;
18 import javafx.fxml.FXML;
19 import javafx.scene.Scene;
20 import javafx.scene.control.Alert;
21 import javafx.scene.control.Alert.AlertType;
22 import javafx.scene.control.ButtonType;
23 import javafx.scene.input.MouseEvent;
24 import javafx.scene.layout.AnchorPane;
25 import other.DAO;
26
27 public class SettingsController {
28     @FXML private JFXComboBox<String> cbxTheme;
29     @FXML private JFXComboBox<Integer> cbxGrade;
30     @FXML private JFXTextField fldName;
31     @FXML private JFXToggleButton tglGreeting;
32     @FXML private AnchorPane apnSettings;
33     private RootController rootController;
34     private File propertiesFile;
35     private Properties settingsProps = new Properties();
36     private DAO dao = new DAO();
37     public void setRootController(RootController rootController) {
38         this.rootController = rootController;
39     }
40
41     public void initialize() {
42
43         Platform.runLater(() -> {
44             //initialise the comboboxes
45             cbxTheme.getItems().addAll("Default Chamber", "Future Sea", "Rose Petal");
46             cbxGrade.getItems().addAll(1, 2, 3, 4, 5, 6, 7, 8);
47
48             //set initial values
49             fldName.setText(rootController.getName());
50             cbxGrade.setValue(rootController.getGrade());
51             loadValues();
52
53             apnSettings.requestFocus(); //no highlighted control
54         });
55
56         fldName.textProperty().addListener((o, ov, nv) -> {
57             rootController.setName(nv);
58         });
59
60     }
61     public void loadValues() {
62         final String path = "C:\\\\iMProve\\\\settings.properties";
63         //create file
64         try {
65             propertiesFile = new File(path);
66             propertiesFile.createNewFile(); //if file already exists, does nothing
67         } catch(IOException ex) {ex.printStackTrace();}
68         //properties
69         try (InputStream in = Files.newInputStream(propertiesFile.toPath())) {
70             settingsProps.load(in);
71
72             //get the properties
73             String theme = settingsProps.getProperty("theme", "Default Chamber");
74             boolean greeting = "true".equals(settingsProps.getProperty("greeting", "true")); //if true is what is stored, return true; if stored value not equal to true, return false
75             boolean panda = "true".equals(settingsProps.getProperty("panda unlocked", "false"));
76
77             //set control values
78             if(panda)cbxTheme.getItems().add("The Panda Special");
79             cbxTheme.setValue(theme);
80             tglGreeting.setSelected(greeting);
81         } catch(IOException ex) {
82             ex.printStackTrace();
83         }
84     }
85
86     @FXML
87 }
```

```

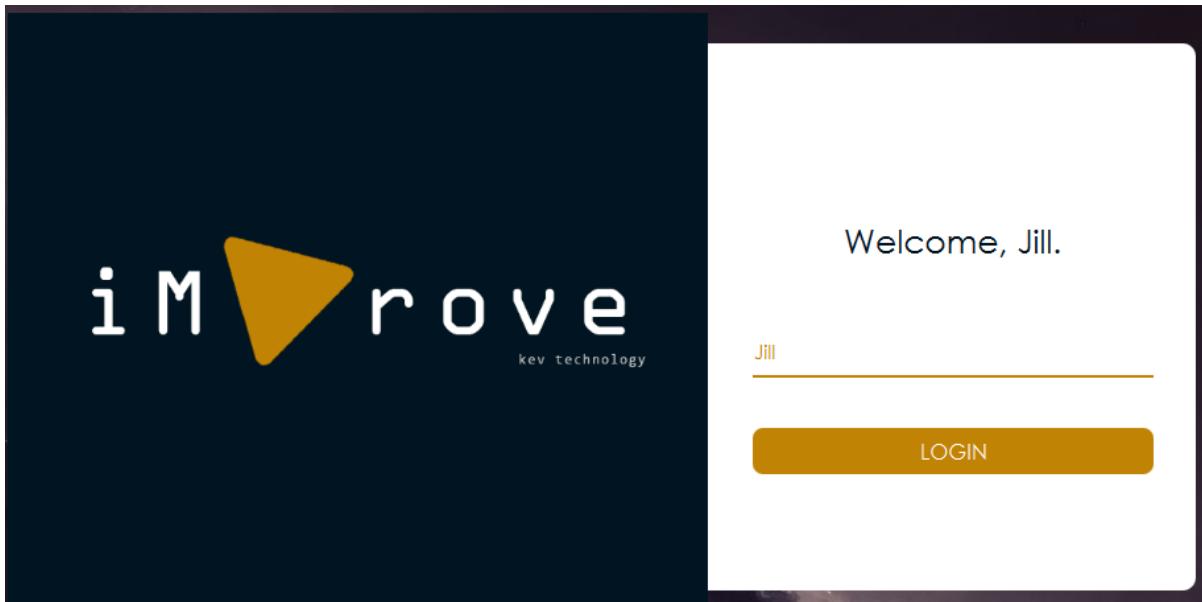
87● @FXML
88 public void changeTheme(ActionEvent ae) {
89     //code to change theme
90     setTheme(cbxTheme.getValue());
91 }
92● @FXML
93 public void pandaTheme(MouseEvent me) {
94     settingsProps.setProperty("panda unlocked", "true");
95     if(!cbxTheme.getItems().contains("The Panda Special"))cbxTheme.getItems().add("The Panda Special");
96     setTheme("The Panda Special");
97 }
98● public void setTheme(String theme) {
99     Scene rootScene = apnSettings.getScene();
100    rootScene.getStylesheets().clear();
101    rootScene.getStylesheets().add(getCSS(theme));
102    //change properties
103    settingsProps.setProperty("theme", theme);
104    saveProperties("theme changed");
105
106    RootController.setTheme(theme);
107    rootController.changeNavigationColours(null, null, null);
108 }
109●
110● @FXML
111 public void changeGrade(ActionEvent ae) {
112     //code to change grade
113     RootController.setGrade(cbxGrade.getValue());
114     //remove listen if lower than 8
115     if(cbxGrade.getValue()<8) {
116         rootController.removeListen();
117     } else {
118         rootController.addListen();
119     }
120 }
121● @FXML
122 public void changeGreeting(ActionEvent ae) {
123     //update properties for next time welcome
124     settingsProps.setProperty("greeting", String.valueOf(tglGreeting.isSelected()));
125     saveProperties("greeting toggled");
126 }
127● @FXML
128 public void resetData(ActionEvent ae) {
129     Alert confirmAlert = new Alert(AlertType.CONFIRMATION);
130     confirmAlert.setHeaderText("Are you sure you want to reset your data?");
131     Optional<ButtonType> result = confirmAlert.showAndWait();
132     if(result.isPresent() && result.get() == ButtonType.OK) {
133         DAO.closeConnections();
134         dao.reset(); //reset database
135         settingsProps.clear(); //reset settings
136         saveProperties("reset");
137
138         cbxTheme.getItems().remove("The Panda Special");
139         tglGreeting.setSelected(true);
140         setTheme("Default Chamber");
141         cbxTheme.setValue("Default Chamber");
142     }
143 }
144● private String getCSS(String theme) {
145     String fileName = "resources/themes/";
146     switch(theme) {
147         case "Future Sea": fileName += "future_sea.css";
148         break;
149         case "Default Chamber": fileName += "default_chamber.css";
150         break;
151         case "Rose Petal": fileName += "rose_petal.css";
152         break;
153         case "The Panda Special": fileName += "panda_special.css";
154         break;
155     }
156 }
157     return fileName;
158 }
159● private void saveProperties(String comment) {
160     try (OutputStream out = Files.newOutputStream(propertiesFile.toPath(), StandardOpenOption.TRUNCATE_EXISTING)){
161         settingsProps.store(out, comment);
162         out.close();
163     } catch(IOException ex) {ex.printStackTrace();}
164 }
165 }

```

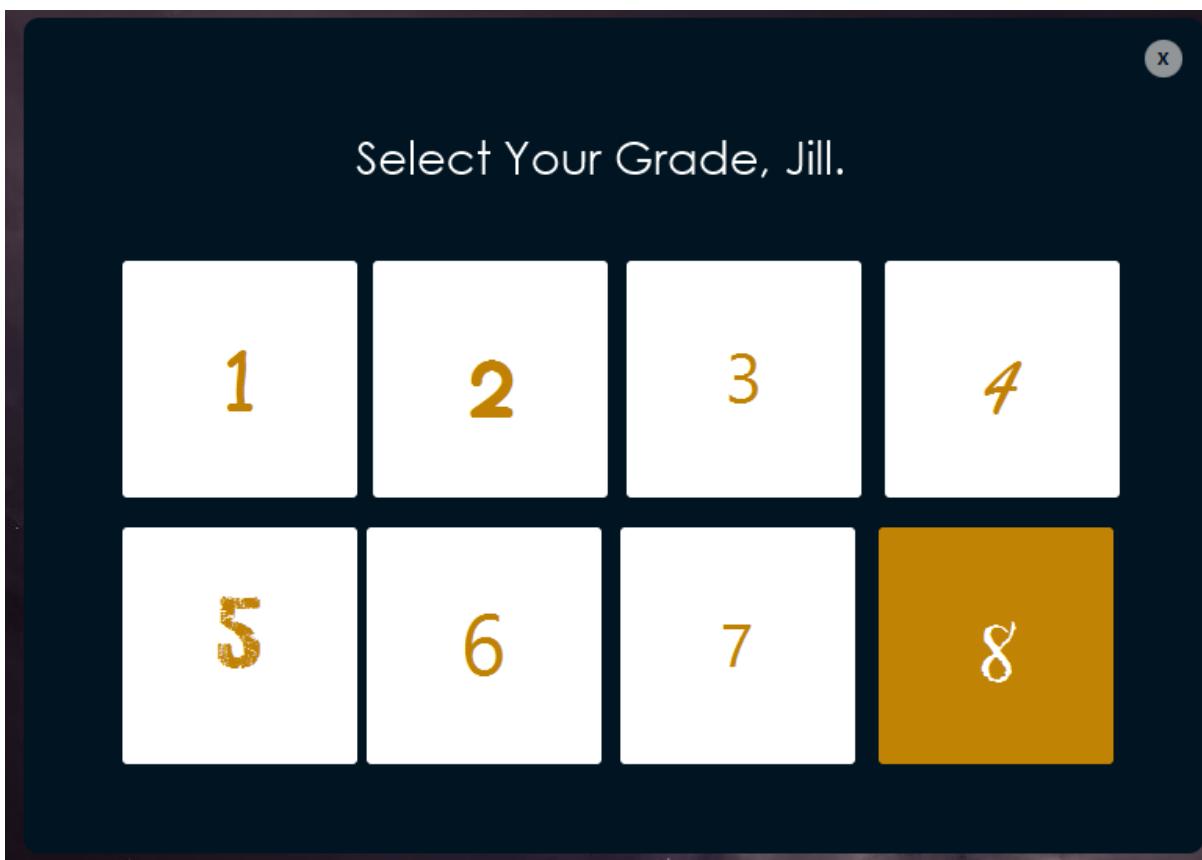
End Of Iteration Validation Test

To review our system once more, Jill was happy to try the program again, especially after her suggested improvements were made and more.

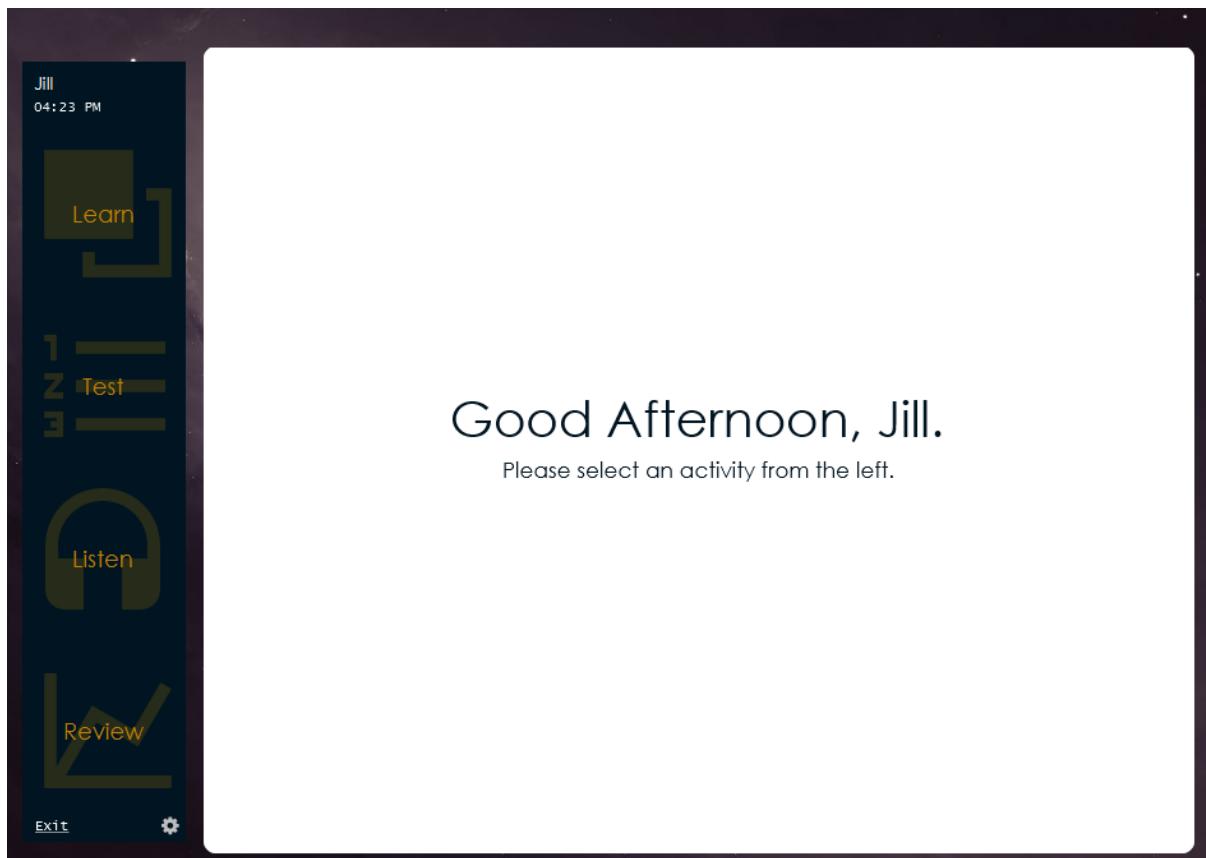
At the start of execution, it displayed the Login screen; she entered her name.



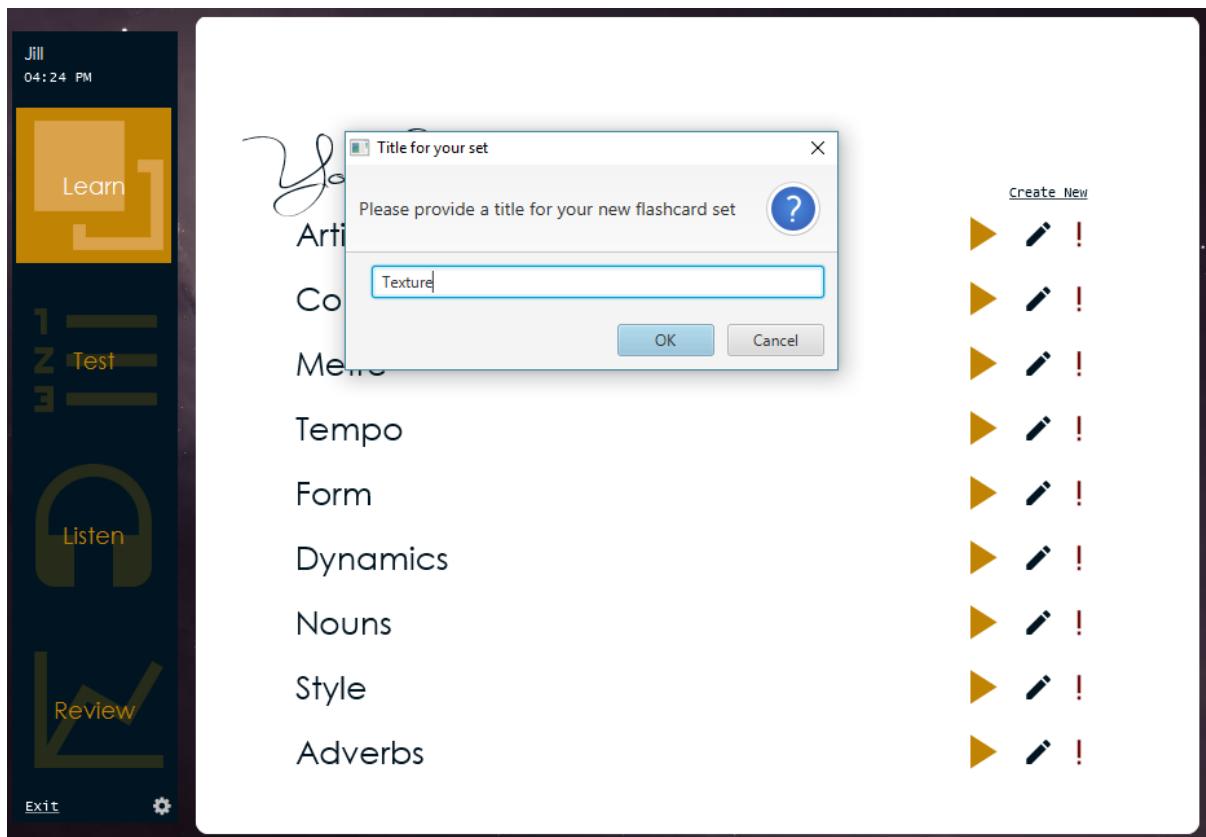
She then selected Grade 8

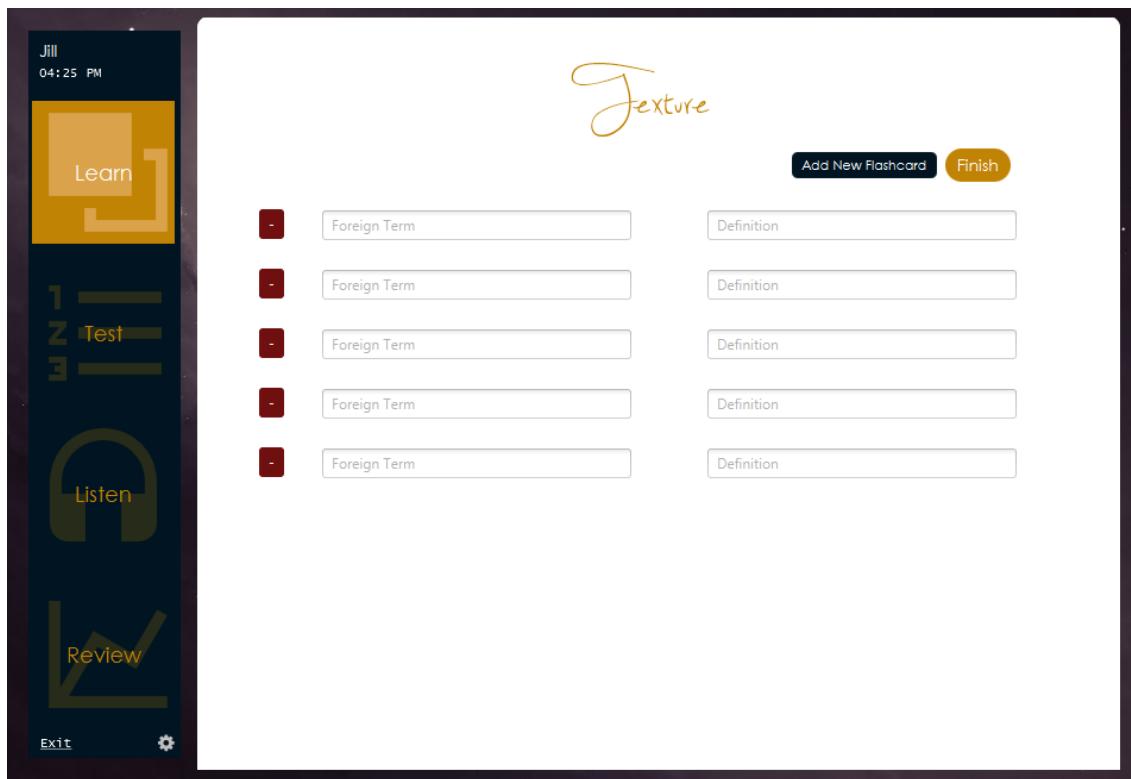


After reaching the Welcome screen...



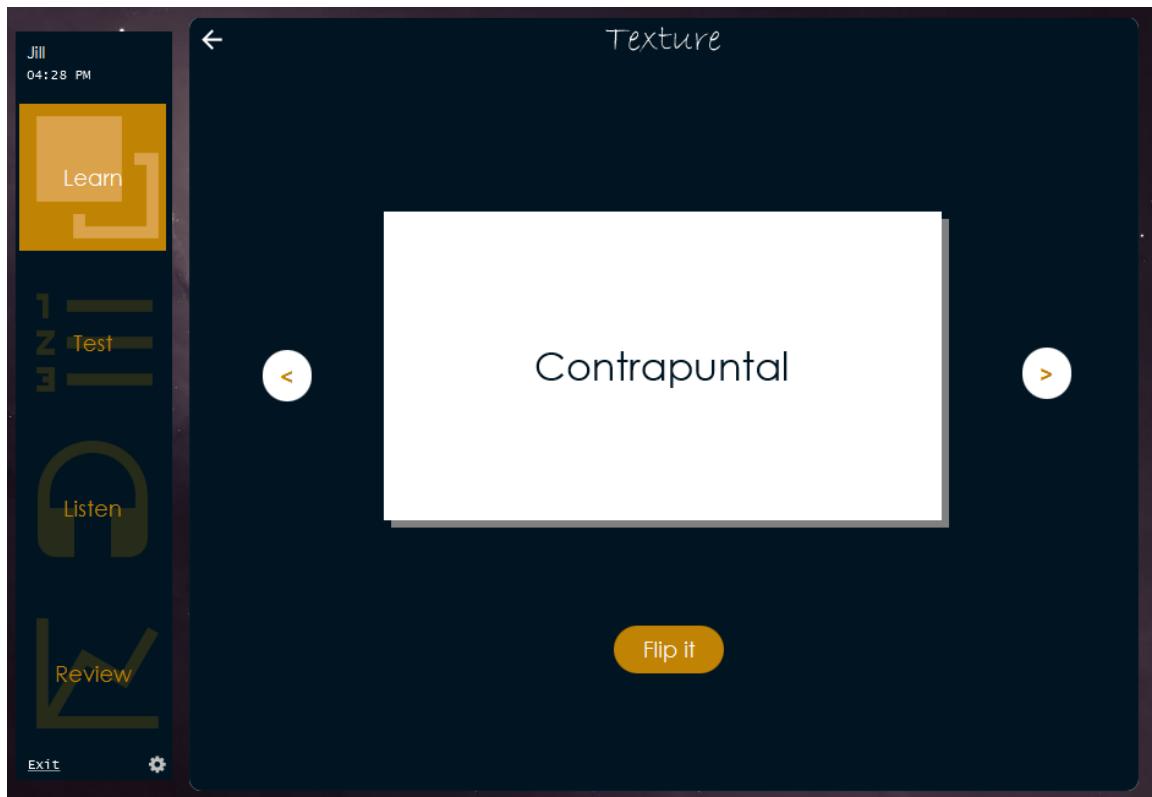
Jill selected the Learn section again and headed to create a new flashcard set.

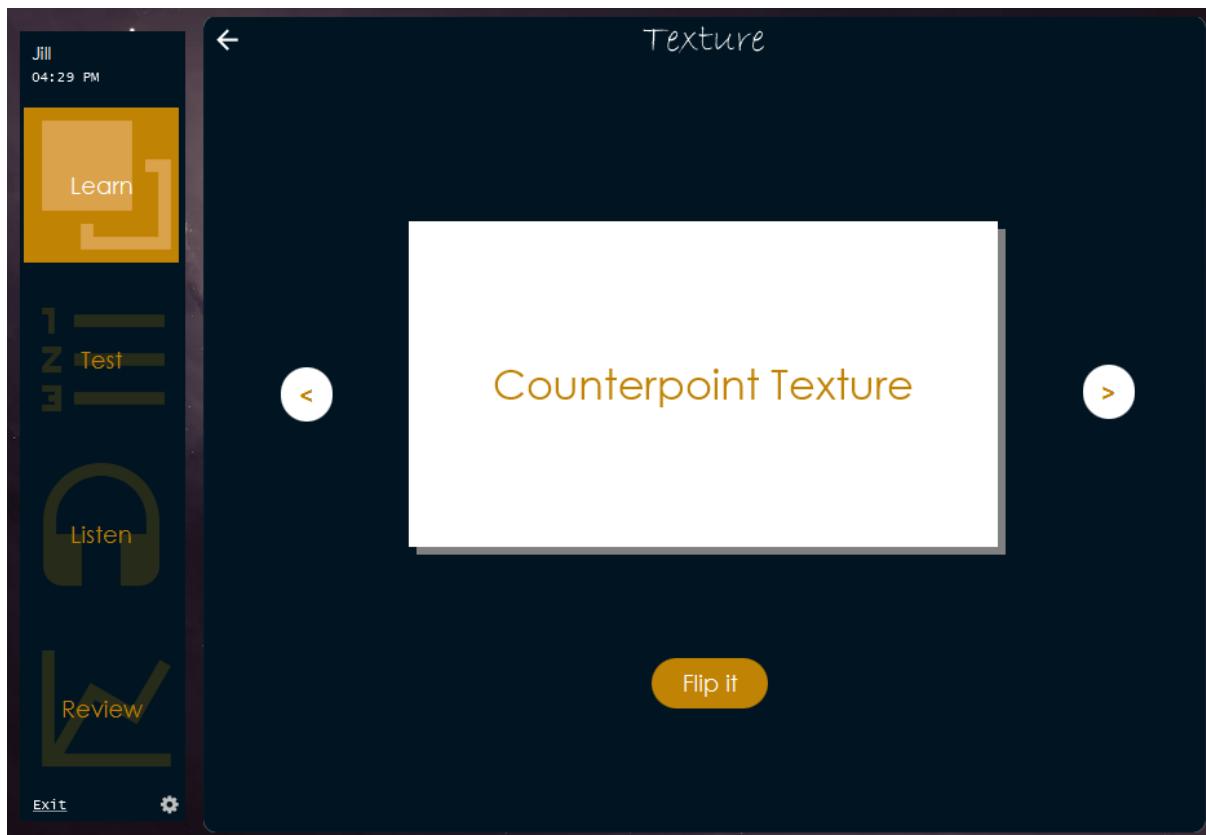




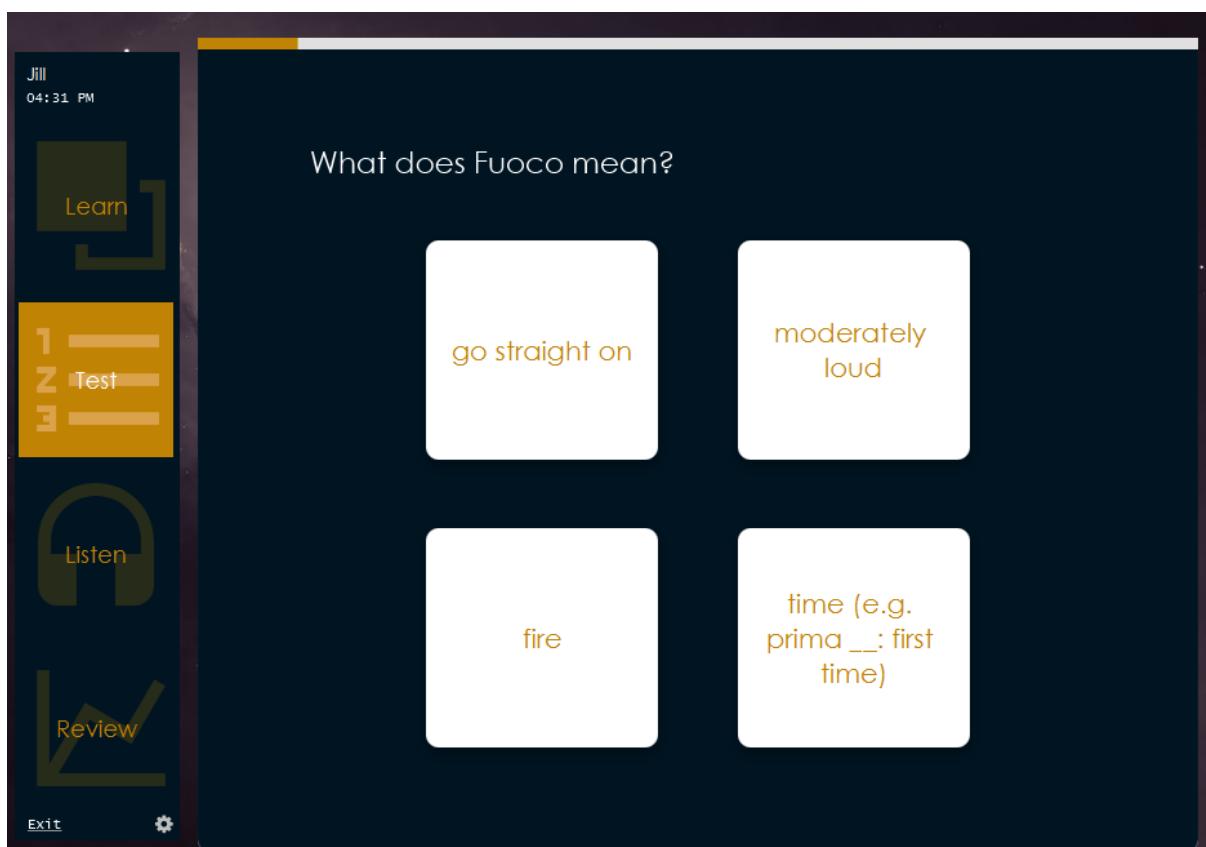
She noticed that there were some example rows to get her started creating her flashcard sets; this improved from her suggestion of her last iteration.

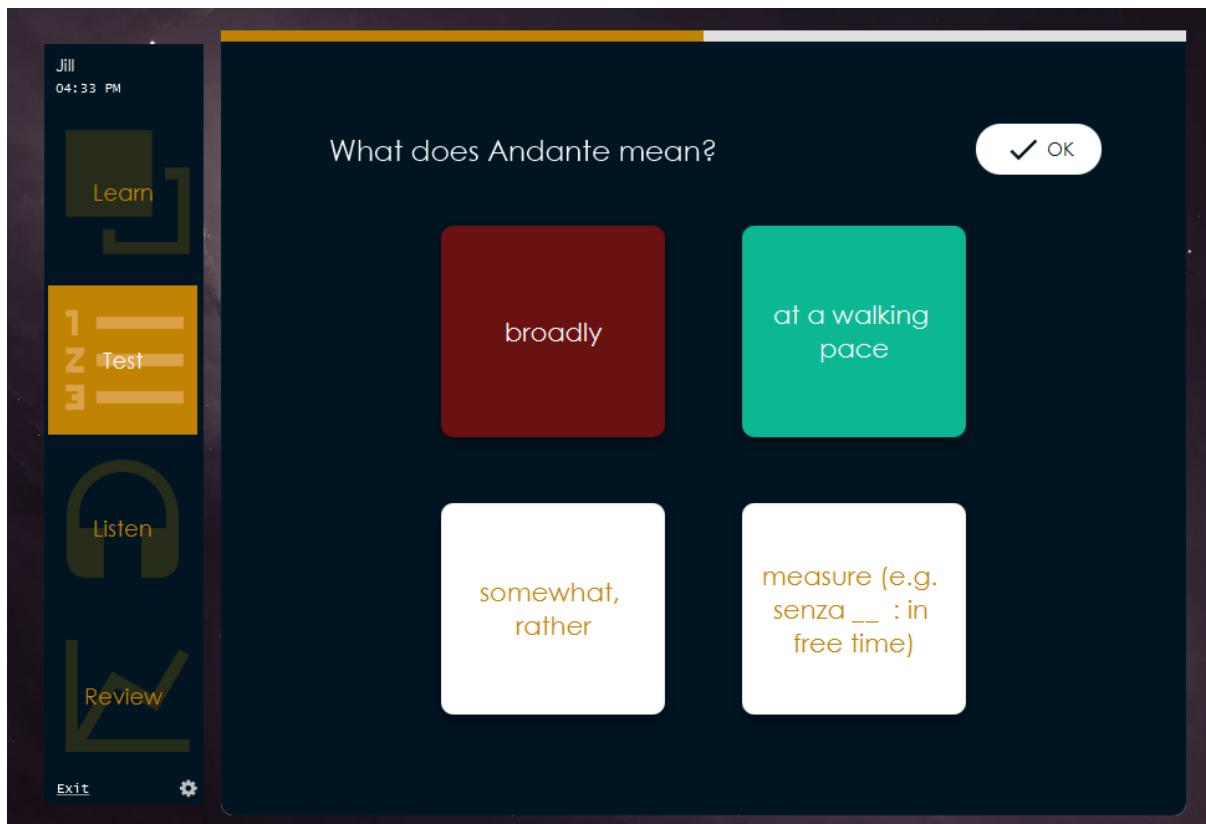
The flashcard section worked very well, especially with the use of keys which she felt made it easier to navigate.



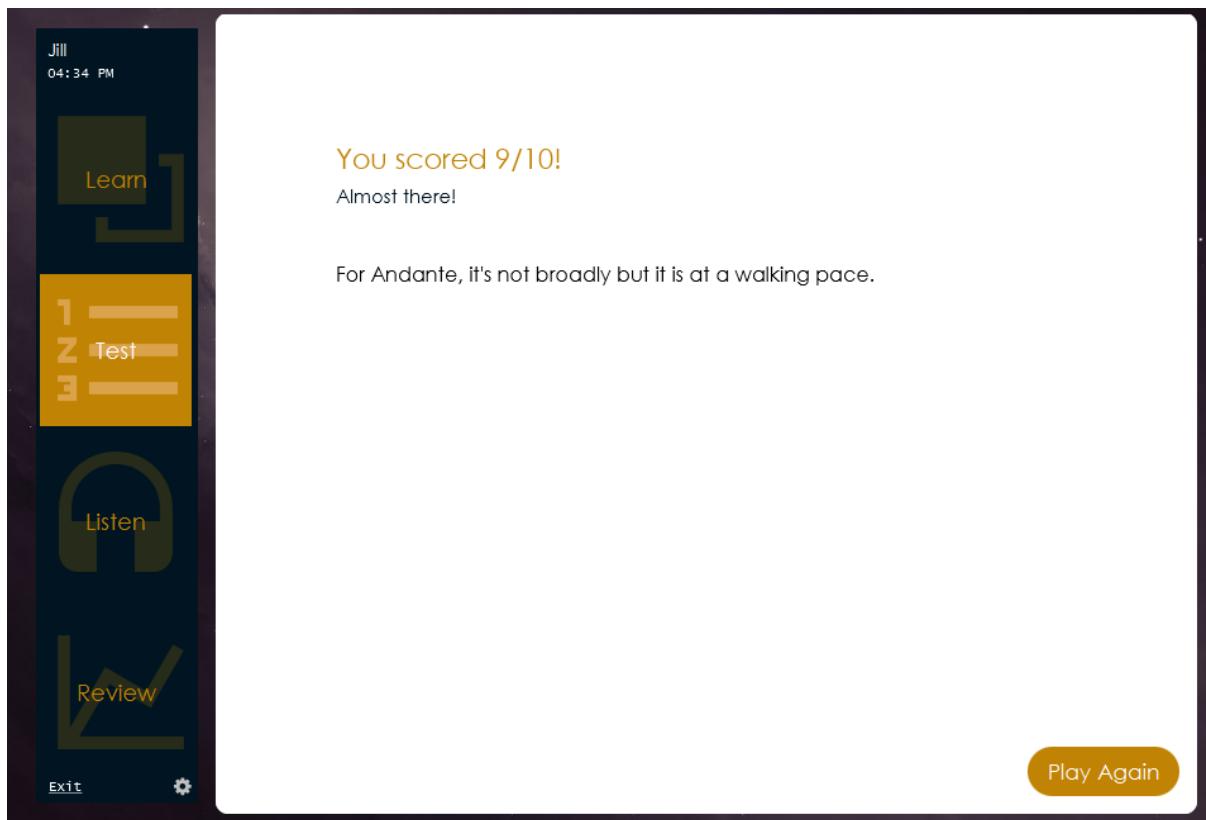


Starting a test, the button text did not overrun and was successfully wrapped within the button and feedback was correctly displayed as usual.

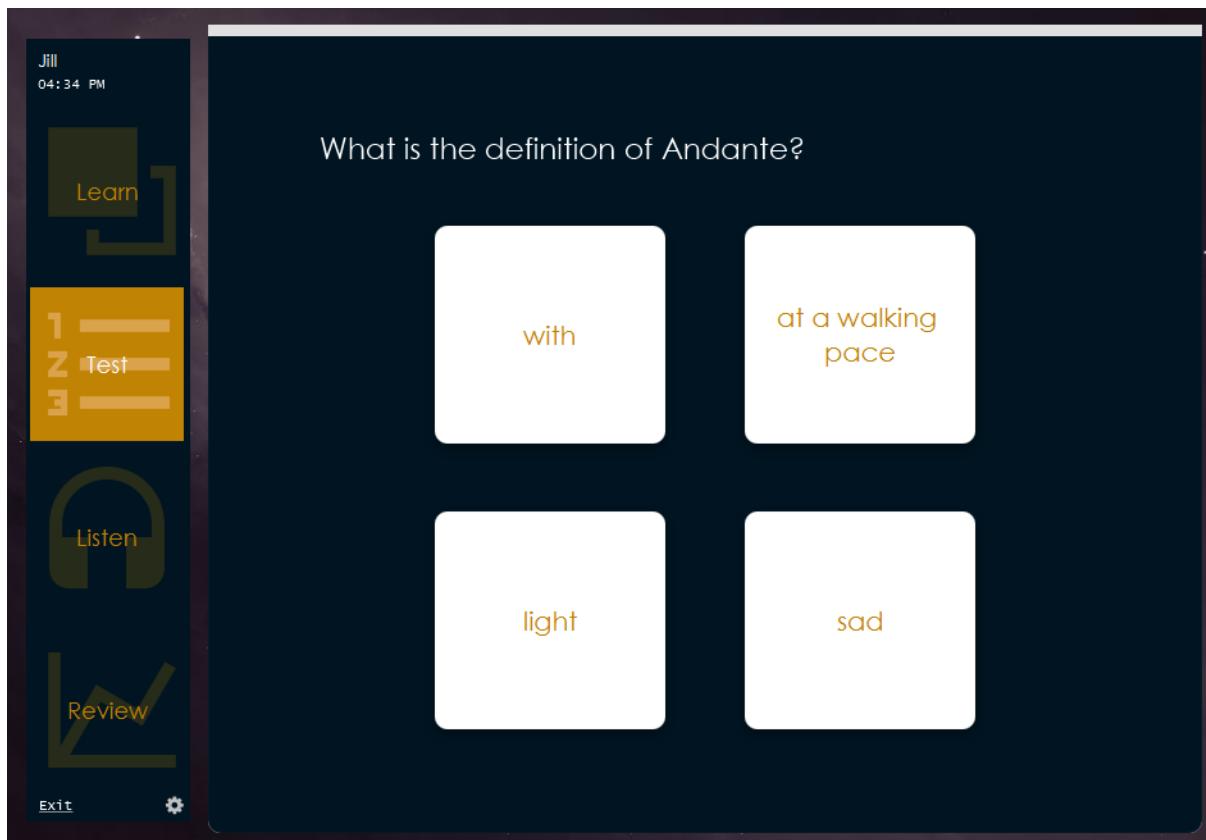




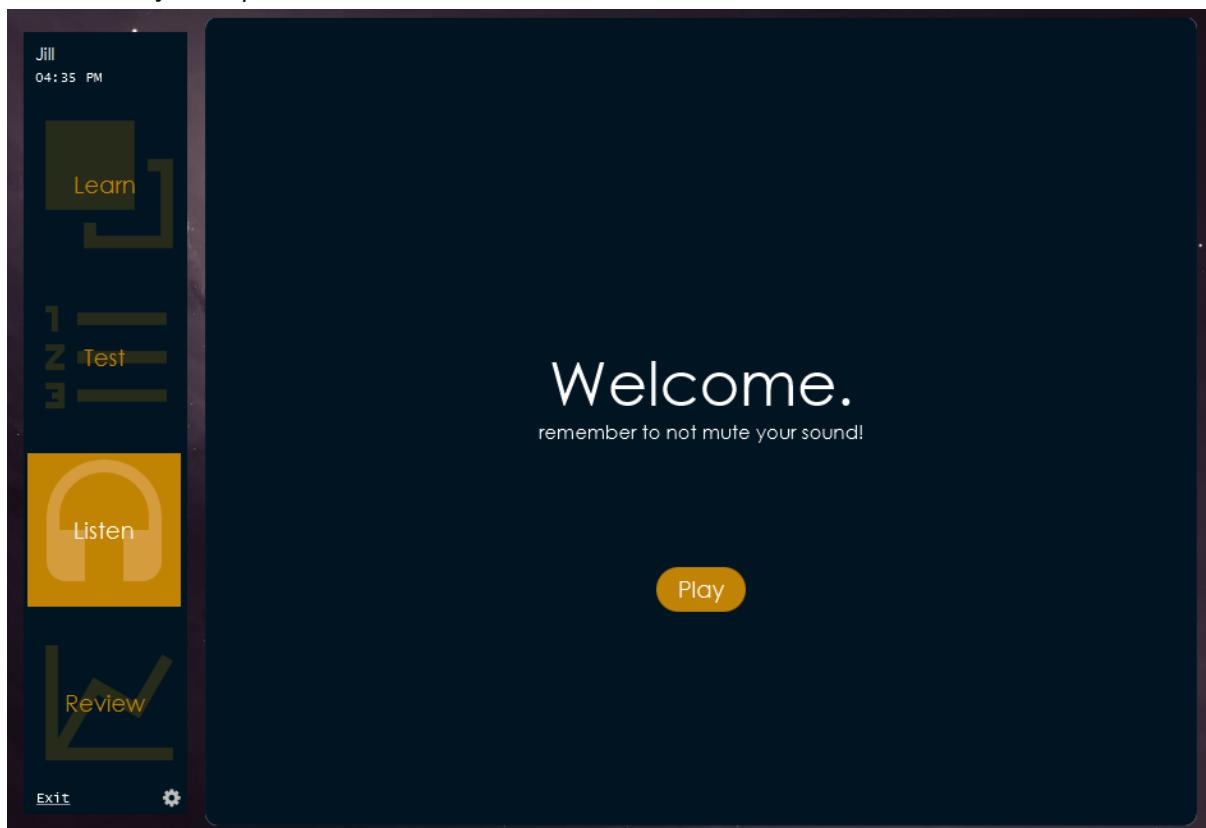
The feedback was also successfully displayed.



By starting a new test, that incorrectly answered question was displayed first.



Next, she tried the Listen section. After pressing both play buttons, the music started playing automatically as expected.





A screenshot of a mobile application interface. On the left, there is a vertical sidebar with a dark background. At the top of the sidebar, it says "Jill" and "04:36 PM". Below this are four items: "Learn" (with a small icon), "1", "2 Test", and "3". Underneath these are three large icons: "Listen" (with headphones), "Review" (with a checkmark), and "Exit" (with a gear icon). To the right of the sidebar is a question: "What is the modulation?". Below the question are four white rectangular buttons arranged in a 2x2 grid. The top-left button contains the text "Relative Major", the top-right button contains "Relative Minor", the bottom-left button contains "Subdominant", and the bottom-right button contains "Dominant".

In addition to being impressed by the original use of audio played by myself, she like the fact that you could not play the audio again and could only once, like in a real exam.

Jill
04:38 PM

Learn

Test

Listen

Review

Exit

Settings

What is the modulation?

Relative Major

Relative Minor

Subdominant

Dominant

This screenshot shows a mobile application interface for a music theory study session. The top left corner displays the user's name, Jill, and the current time, 04:38 PM. On the far left, there is a vertical sidebar with four main categories: Learn (green), Test (yellow), Listen (orange), and Review (dark blue). Below these are three progress bars labeled 1, 2, and 3, each with a small horizontal bar indicating completion. At the bottom of the sidebar are the standard Android navigation buttons for Back, Home, and Recent Apps. The main content area is titled "What is the modulation?" and features a horizontal slider with a yellow double-headed arrow and a circular slider button. Below the slider, four options are presented in white boxes: "Relative Major" (orange text), "Relative Minor" (orange text), "Subdominant" (orange text), and "Dominant" (orange text). The "Relative Minor" option is highlighted with a teal background.

After answering the question, she regained the ability to look back and correct her thought process; this, she felt would be very beneficial for a student as a learning resource.

Jill
04:41 PM

Learn

Test

Listen

Review

Exit

Settings

What is the modulation?

||

Relative Major

Relative Minor

Subdominant

Dominant

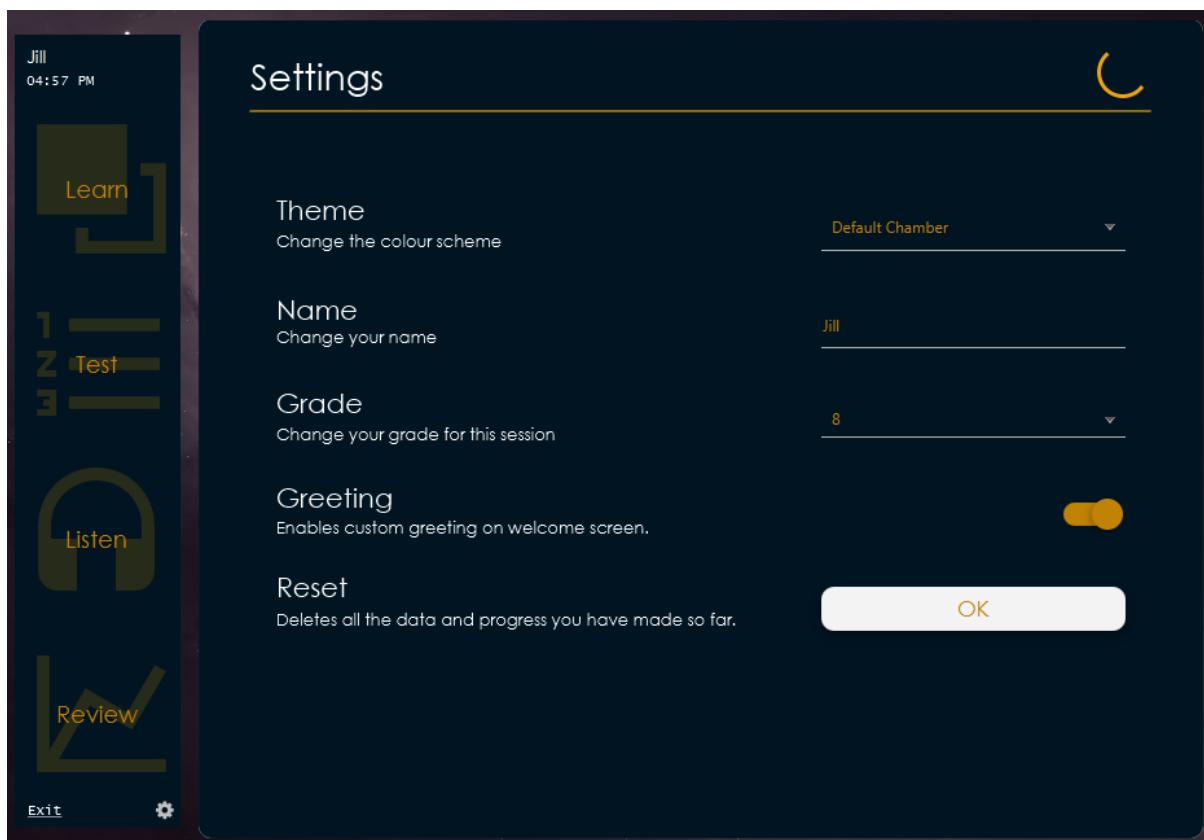
Play Again

This screenshot shows the same mobile application interface after a user has selected an answer. The title "What is the modulation?" remains at the top. The horizontal slider is now positioned further along the track, indicated by a yellow dot. The "Relative Minor" option is now highlighted with a teal background, while the other three options ("Relative Major", "Subdominant", and "Dominant") are in their original white state. In the bottom right corner, there is a yellow rounded rectangular button labeled "Play Again". The rest of the interface, including the sidebar with categories and progress bars, remains identical to the first screenshot.

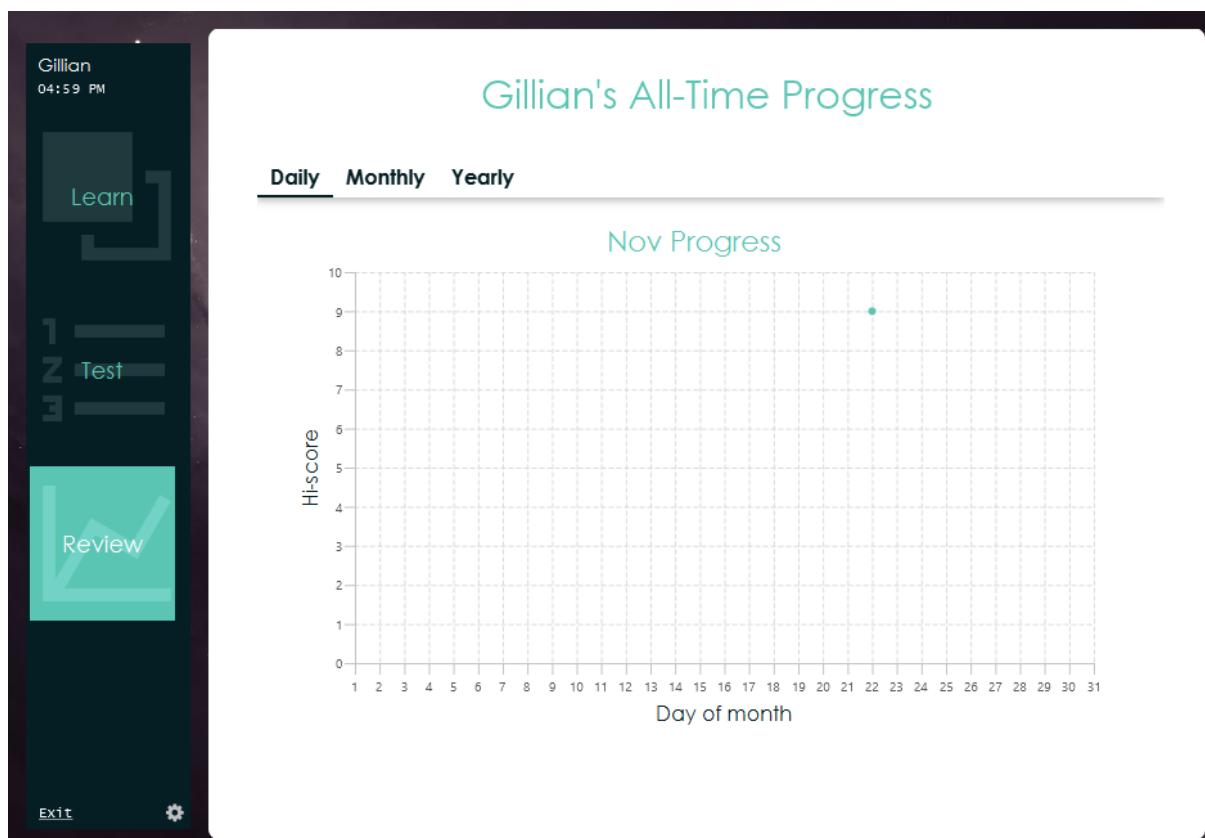
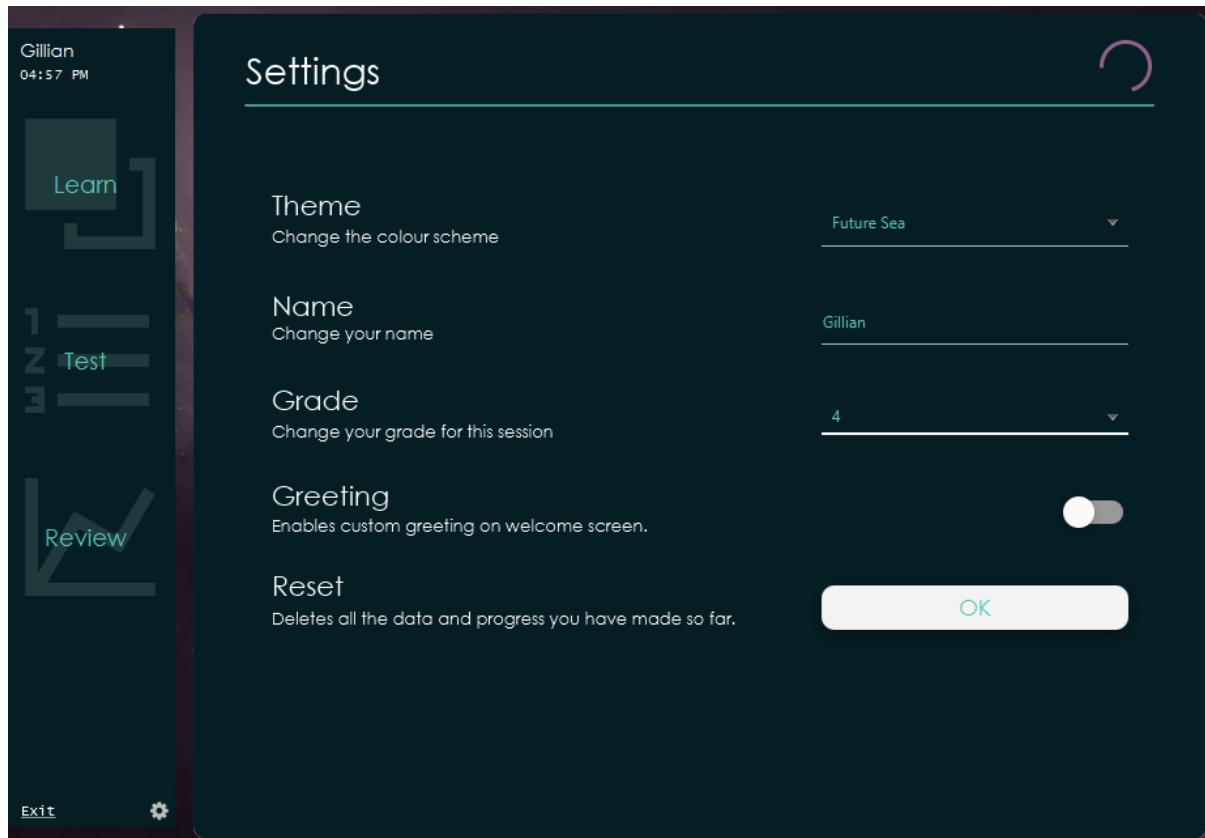
Nothing really changed from the Review section.



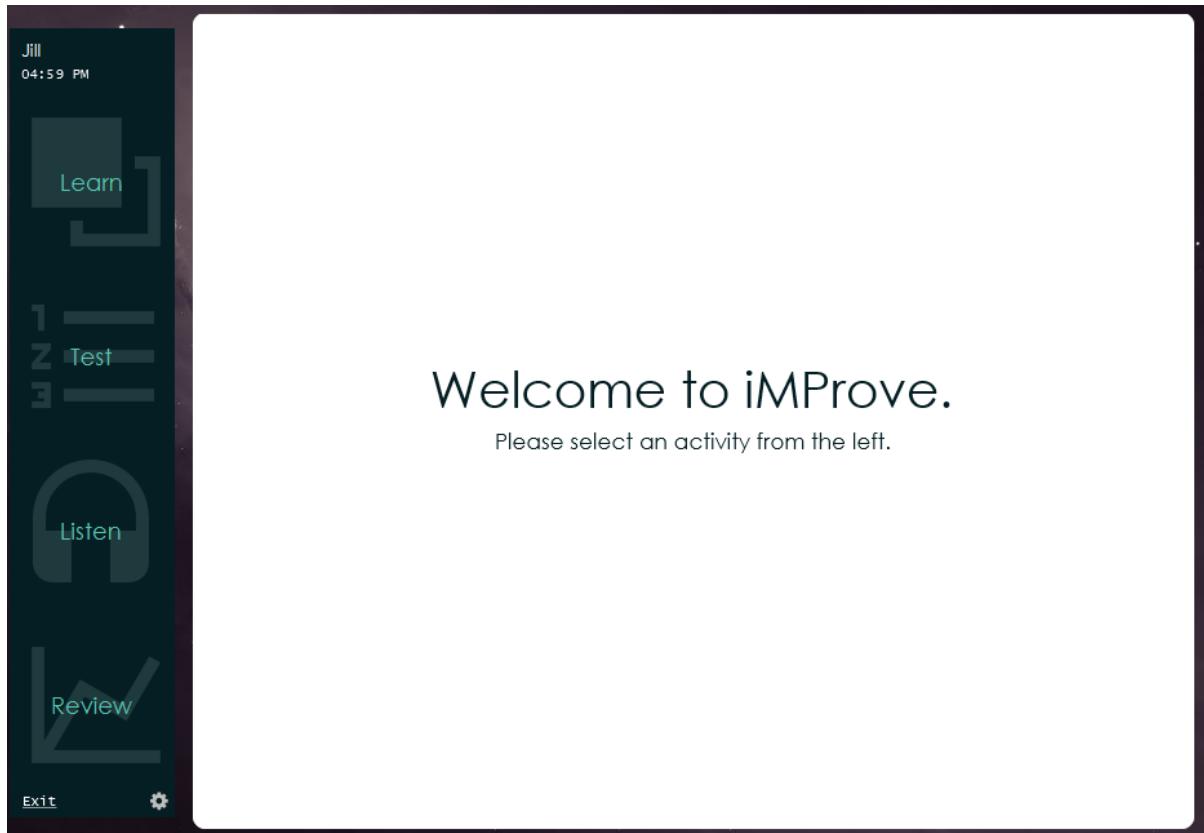
Next, we checked out the Settings page which is a new feature of iMProve.



After changing the settings and configurations of the system, she noticed many changes, change of colour, name, and appearance of Listen activity.



Starting the program again, the greeting was toggled off so now the custom greeting had disappeared and instead: *Welcome to iMProve*.



By testing the entirety of the program again, she felt confident that her students could feel motivated and learn from this system. As a whole, her suggestions were worked upon and improved one step further; she felt content with the result of this iteration.

Annotated Modular Code

Below will document the collective annotated modular code of the final system.

Main.java

```
package other;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

public class Main extends Application {
    private static Stage mainStage;
    @Override
    public void start(Stage stage) throws Exception { //called by launch(args)
```

```

        loadFonts();
        mainStage = stage;

        //show login screen
        Parent root =
FXMLLoader.load(getClass().getResource("/view/login.fxml"));
        Scene loginScene = new Scene(root);
        loginScene.setFill(null);
        stage.initStyle(StageStyle.TRANSPARENT);
        stage.setScene(loginScene);
        stage.setTitle("iMPROVE");
        stage.show();
    }
    public static void main(String[] args) { //called by jvm
        Launch(args);
    }
    public void loadFonts() { //load fonts with random size, doesn't matter

        Font.LoadFont(getClass().getResourceAsStream("/resources/fonts/28DaysLater.
ttf"), 14);

        Font.LoadFont(getClass().getResourceAsStream("/resources/fonts/ALBA__.ttf")
, 14);

        Font.LoadFont(getClass().getResourceAsStream("/resources/fonts/ALBAM_.ttf")
, 14);

        Font.LoadFont(getClass().getResourceAsStream("/resources/fonts/AKBAS_.ttf")
, 14);

        Font.LoadFont(getClass().getResourceAsStream("/resources/fonts/BRADHITC.ttf
"), 14);

        Font.LoadFont(getClass().getResourceAsStream("/resources/fonts/GOTHIC.ttf")
, 14);

        Font.LoadFont(getClass().getResourceAsStream("/resources/fonts/GOTHICB.ttf"
), 14);

        Font.LoadFont(getClass().getResourceAsStream("/resources/fonts/GOTHICBI.ttf
"), 14);

        Font.LoadFont(getClass().getResourceAsStream("/resources/fonts/GOTHICI.ttf"
), 14);

        Font.LoadFont(getClass().getResourceAsStream("/resources/fonts/SweetlyBroke
n.ttf"), 14);

    }
    public static Stage getStage() { //static getter method to return the stage
(window) of the application
        return mainStage;
    }
}

```

DAO.java

```

package other;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.StandardCopyOption;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import net.ucanaccess.jdbc.UcanaccessConnection;

public class DAO {
    private static final String dbDir = "C://iMProve";
    private static final String dbName = "improveDB.accdb";
    private static final String dbUrl = "jdbc:ucanaccess://" + dbDir + "//" +
    dbName;
    private static ObservableList<Connection> allConnections =
    FXCollections.observableArrayList();

    public DAO() { //constructor - called when object is made
        try {
            Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
        } catch(ClassNotFoundException e) {
            System.out.println("Cannot load ucanaccess driver");
            e.printStackTrace();
        }
        File directory = new File(dbDir);
        if(!directory.exists()) //create directory if not already
            directory.mkdir();
        File database = new File(dbDir + "//" + dbName);
        if(!database.exists()) { //copy the database file into user's file
            system - if not already
            try {
                Files.copy(DAO.class.getResourceAsStream(dbName),
                database.toPath(), StandardCopyOption.REPLACE_EXISTING);
            } catch(IOException ex) {ex.printStackTrace();}
        }
    }
    public void reset() {
        File database = new File(dbDir + "//" + dbName);
        try {
            Files.copy(DAO.class.getResourceAsStream(dbName),
            database.toPath(), StandardCopyOption.REPLACE_EXISTING); //replace the file
            system's database to a fresh new one
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public Connection getConnection() { //create a connection to the database
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(dbUrl);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

```

        }
        //add to collection to later clear
        allConnections.add(conn);
        return conn;
    }
    public static void closeConnections() {
        try {
            if(!allConnections.isEmpty()) {
                ((UcanaccessConnection)
allConnections.get(0)).unloadDB(); //unload resources on ucanaccess driver
                allConnections.clear(); //no more open connections
after all closed + resources unloaded
            }
        } catch(SQLException e) {
            e.printStackTrace();
        }
    }
}

```

Flashcard.java

```

package model;

public class Flashcard {
    //private instance variables
    private String foreignTerm;
    private String definition;
    //constructor which initialises the instance variables
    public Flashcard(String foreignTerm, String definition) {
        this.foreignTerm = foreignTerm;
        this.definition = definition;
    }

    //getter and setter methods for better encapsulation and information hiding
    (more secure)
    public String getTerm() {
        return foreignTerm;
    }
    public String getDefinition() {
        return definition;
    }
    public void setTerm(String foreignTerm) {
        this.foreignTerm = foreignTerm;
    }
    public void setDefinition(String definition) {
        this.definition = definition;
    }
}

```

TheoryQuestion.java

```

package model;

public class TheoryQuestion {
    //instance variables of a question of music theory
    private String question;

```

```

private String[] answers = new String[4];
private int answerIndex;

//initialise the variables
public TheoryQuestion(String question, String[] answers, int answerIndex) {
    this.question= question;
    this.answers = answers;
    this.answerIndex = answerIndex;
}

//getter and setter methods for the instance variables
public void setQuestion(String question) {
    this.question = question;
}
public void setAnswers(String a1, String a2, String a3, String a4) {
    answers[0] = a1;
    answers[1] = a2;
    answers[2] = a3;
    answers[3] = a4;
}
public void setIndex(int index) {
    answerIndex = index;
}
public String getQuestion() {
    return question;
}
public String[] getAnswers() {
    return answers;
}
public String getAnswer(int index) {
    return answers[index];
}
public int getAnswerIndex() {
    return answerIndex;
}
}
}

```

Feedback.java

```

package model;

public class Feedback {
    //encapsulated object: instance variables + respective getter and setter
methods
    private String question;
    private String wrongAnswer;
    private String correctAnswer;
    public Feedback(String question, String wrong, String correct) {
        this.question = question;
        wrongAnswer = wrong;
        correctAnswer = correct;
    }
    public String getQuestion() {
        return question;
    }
    public String getWrong() {
        return wrongAnswer;
    }
}

```

```

    }
    public String getCorrect() {
        return correctAnswer;
    }
}

```

AuralTest.java

```

package model;

import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;

public class AuralTest {
    //instance variables holding required data for the aural test
    private String audioFile;
    private MediaPlayer auralPlayer;
    private String modulation;
    private double duration;
    //initialise variables and load the audio into a MediaPlayer object
    public AuralTest(String audioFile, String modulation) {
        this.audioFile = audioFile;
        this.modulation = modulation;
        createAuralPlayer();
    }

    private void createAuralPlayer() {
        //load audio into MediaPlayer object
        Media media = new
Media(getClass().getResource("/resources/audio/"+audioFile).toExternalForm());
        auralPlayer = new MediaPlayer(media);
        //set the duration variable when it is loaded
        auralPlayer.setOnReady(()->duration =
auralPlayer.getMedia().getDuration().toMillis());
    }

    //getter setter methods
    public MediaPlayer getMediaPlayer() {
        return auralPlayer;
    }
    public String getModulation() {
        return modulation;
    }
    public double getDuration() {
        return duration;
    }
}

```

LoginController.java

```
package controller;

import java.io.IOException;

import com.jfoenix.controls.JFXButton;
import com.jfoenix.controls.JFXTextField;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.layout.AnchorPane;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.input.KeyCode;
import other.Main;

public class LoginController {
    //the UI objects
    @FXML private JFXTextField fldName;
    @FXML private Label lblWelcome;
    @FXML private AnchorPane apnLoginRoot;
    @FXML private JFXButton btnLogin;

    public void initialize() {
        //bind fldName
        fldName.textProperty().addListener((o, ov, nv) -> {
            //only show the login button if there is text inside (self
            validates the input) and also change the display text
            if(nv.isEmpty()) {
                btnLogin.setVisible(false);
                lblWelcome.setText("Welcome.");
            } else {
                btnLogin.setVisible(true);
                lblWelcome.setText("Welcome, "+nv+".");
            }
        });

        //bind enter key to accept login
        apnLoginRoot.setOnKeyPressed(e -> {
            if(e.getCode()==KeyCode.ENTER) {
                doLogin();
            }
        });
    }

    @FXML public void login(ActionEvent ae) { //clicked by 'login'
        doLogin();
    }

    private void doLogin() {
        if(!fldName.getText().isEmpty()) { //if there is text
            //load the grade selection screen
            AnchorPane root = null;
            FXMLLoader loader = new
            FXMLLoader(getClass().getResource("/view/grade_select.fxml"));
            try {
                root = loader.load();
            
```

```

        } catch(IOException e) {e.printStackTrace();}
        GradeController controller = loader.getController();
        controller.setName(fldName.getText());

        Scene gradeScene = new Scene(root);
        gradeScene.setFill(null);
        Main.getStage().setScene(gradeScene);
    }

}

```

GradeController.java

```

package controller;

import java.io.IOException;

import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.layout.AnchorPane;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.text.Text;
import other.Main;

public class GradeController {
    private String name;
    @FXML private Text txtSelectGrade;
    public void setName(String name) { //input the user's name
        this.name = name;
        Platform.runLater(()-> txtSelectGrade.setText("Select Your Grade, "
+ name + "."));
    }
    @FXML
    public void close(ActionEvent e) { //close system
        System.exit(0);
    }
    @FXML
    public void gradeSelect(ActionEvent e) { //load next part of system after
grade selected
        //get grade
        Button selectedButton = (Button) e.getSource();
        int grade = Integer.parseInt(selectedButton.getText());
        //load root gui and set some initial values
        FXMLLoader loader = new
        FXMLLoader(getClass().getResource("/view/root.fxml"));
        AnchorPane root = null;
        try {
            root = loader.load();
        } catch(IOException ioe) {ioe.printStackTrace();}
        RootController rootController = loader.getController();
        rootController.initData(name, grade);

        Scene rootScene = new Scene(root);

```

```

        rootScene.setFill(null);
        Main.getStage().setScene(rootScene);
    }

}

```

RootController.java

```

package controller;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.util.Arrays;
import java.util.Properties;

import com.jfoenix.controls.JFXButton;

import de.jensd.fx.glyphs.materialdesignicons.MaterialDesignIconView;
import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.layout.AnchorPane;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import javafx.util.Duration;

public class RootController {
    //UI objects
    @FXML private Label lblName;
    @FXML private AnchorPane apnRoot, apnActivity;
    @FXML private StackPane spnLearn, spnTest, spnListen, spnReview;
    @FXML private MaterialDesignIconView icnLearn, icnTest, icnListen,
icnReview;
    @FXML private JFXButton btnLearn, btnTest, btnListen, btnReview;
    @FXML private Text txtTime, txtWelcome;
    @FXML private VBox vbxActivities;
    //important data to be stored
    private static int grade;
    private static String name;
    private static String theme;
    private boolean greeting;
    //loaded by java fx application thread so the ui objects have been loaded
    and we can change its properties
    public void initialize() {
        Platform.runLater(()->{
            //get properties

```

```

        Properties settingsProps = getProperties();
        String storedTheme = settingsProps.getProperty("theme",
    "Default Chamber");
        greeting = "true".equals(settingsProps.getProperty("greeting",
    "true"));
        //set theme
        setCSS(storedTheme);

        //set greeting
        if(greeting) {
            int hour =
    Integer.parseInt(LocalTime.now().format(DateTimeFormatter.ofPattern("HH")));
            String greeting;
            if(hour>=5 & hour<12)
                greeting = "Good Morning,";
            else if(hour>=12 & hour<18)
                greeting = "Good Afternoon,";
            else if(hour>=18 & hour<23)
                greeting = "Good Evening,";
            else
                greeting = "Good Night,";
            txtWelcome.setText(greeting);
        }

        //take off listen section if not grade 8
        if(grade!=8) {
            removeListen();
        }
        apnRoot.requestFocus();
        //automatically update time
        startTime();
    });
}

//methods to display/remove the listen activity
public void removeListen() {
    vbxActivities.getChildren().remove(spnListen);
}
public void addListen() {
    if(!vbxActivities.getChildren().contains(spnListen))
        vbxActivities.getChildren().add(spnListen);
}
//get the settings
public Properties getProperties() {
    final String path = "C:\\\\iMProve\\\\settings.properties";
    Properties settingsProps = new Properties();

    try {
        File propertiesFile = new File(path);
        propertiesFile.createNewFile(); //will create new file if does
not exist, if exists then no-op

        settingsProps.load(new FileInputStream(propertiesFile));
    } catch(IOException ex) {ex.printStackTrace();}

    return settingsProps;
}
//set the theme (css styling of the system)
private void setCSS(String theme) {

```

```

RootController.theme = theme;
String fileName = "resources/themes/";
switch(theme) {
    case "Future Sea": fileName += "future_sea.css";
    break;
    case "Default Chamber": fileName += "default_chamber.css";
    break;
    case "Rose Petal": fileName += "rose_petal.css";
    break;
    case "The Panda Special": fileName += "panda_special.css";
}
apnRoot.getScene().getStylesheets().add(fileName);
}
//initial data set by GradeController
public void initData(String name, int grade) {
    Platform.runLater(()-> {
        lblName.setText(name);
        txtWelcome.setText(txtWelcome.getText() + (greeting ? " " +
name + "." : ""));
    });
    RootController.grade = grade;
    RootController.name = name;
}
//getter and setter methods for the instance variables
public static int getGrade() {
    return grade;
}
public static String getName() {
    return name;
}
public static String getTheme() {
    return theme;
}
public static void setGrade(int grade) {
    RootController.grade = grade;
}
public void setName(String name) {
    RootController.name = name;
    lblName.setText(name);
}
public static void setTheme(String theme) {
    RootController.theme = theme;
}
//change the activity displayed on the UI
public void setActivity(Node root) {
    apnActivity.getChildren().clear();
    apnActivity.getChildren().add(root);
}
//change the colours of the navigation bar
public void changeNavigationColours(StackPane spn, MaterialDesignIconView
icn, JFXButton btn) {
    for(StackPane pane: Arrays.asList(spnLearn, spnTest, spnListen,
spnReview)) {
        if(!pane.equals(spn))
            pane.setStyle("-fx-background-color: " +
getColourCode(0));
        else
    }
}

```

```

                pane.setStyle("-fx-background-color: " +
((theme.equals("Default Chamber") | theme.equals("Future Sea")) ?
getColourCode(1): "transparent"));
            }
            for(MaterialDesignIconView icon: Arrays.asList(icnLearn, icnTest,
icnListen, icnReview)) {
                if(!icon.equals(icn)) {
                    icon.setStyle("-fx-fill: " + getColourCode(4));
                } else {
                    icon.setStyle("-fx-fill: " + getColourCode(3));
                }
            }
            for(JFXButton button: Arrays.asList(btnLearn, btnTest, btnListen,
btnReview)) {
                if(!button.equals(btn))
                    button.setStyle("-fx-text-fill: " +
((theme.equals("Default Chamber") | theme.equals("Future Sea")) ?
getColourCode(1): "")); //doesnt change text colour if not default or crystal
                else
                    button.setStyle("-fx-text-fill: " + getColourCode(2));
            }
        }
        //a method providing the HEX colour codes to make the code more reusable
    public static String getColourCode(int option) {
        String colourCode = "";
        switch(theme) {
            case "Default Chamber":
                switch(option) {
                    case 0: colourCode = "#001421"; //unselected
background
                                break;
                    case 1: colourCode = "#c18303"; //selected
background/ unselected text
                                break;
                    case 2: colourCode = "#ffffff"; //selected text
                                break;
                    case 3: colourCode = "#daa34b"; //selected icon
                                break;
                    case 4: colourCode = "#262b1a"; //unselected icon
                                break;
                }
                break;
            case "Future Sea":
                switch(option) {
                    case 0: colourCode = "#041e23";
                                break;
                    case 1: colourCode = "#5bc6b4";
                                break;
                    case 2: colourCode = "#ffffff";
                                break;
                    case 3: colourCode = "#77d6c9";
                                break;
                    case 4: colourCode = "#1f3a3d";
                                break;
                }
                break;
            case "Rose Petal":
                switch(option) {

```

```

        case 0: colourCode = "transparent";
                  break;
        case 1: colourCode = "#ce71c2";
                  break;
        //dont need to change selected activities
    }
    break;
case "The Panda Special":
    switch(option) {
        case 0: colourCode = "transparent";
                  break;
        case 1: colourCode = "#3d3d3d";
                  break;
        //as above
    }
    break;
}
return colourCode;
}
//start the timer which automatically updates every second for more
accuracy
public void startTime() {
    Timeline dateUpdater = new Timeline(
        new KeyFrame(Duration.seconds(0), event ->
txtTime.setText(LocalTime.now().format(DateTimeFormatter.ofPattern("hh:mm a")))),
        new KeyFrame(Duration.seconds(1))); //duration 0 to do
update time straight away and then seconds 1 to wait
    dateUpdater.setCycleCount(Animation.INDEFINITE);
    dateUpdater.play();
}

//display the respective activity
@FXML
public void goLearn(ActionEvent e) {
    AnchorPane root = null;
    FXMLLoader loader = new
FXMLLoader(getClass().getResource("/view/learn.fxml"));
    try {
        root = loader.load();
    } catch(IOException ioe) {ioe.printStackTrace();}
    LearnController controller = loader.getController();
    controller.setRootController(this);
    Platform.runLater(()->changeNavigationColours(spnLearn, icnLearn,
btnLearn));
    setActivity(root);
}
@FXML
public void goTest(ActionEvent e) {
    AnchorPane root = null;
    FXMLLoader loader = new
FXMLLoader(getClass().getResource("/view/test.fxml"));
    try {
        root = loader.load();
    } catch(IOException ioe) {ioe.printStackTrace();}
    TestController controller = loader.getController();
    controller.setRootController(this);
    changeNavigationColours(spnTest, icnTest, btnTest);
    setActivity(root);
}

```

```

        }
        @FXML
    public void goListen(ActionEvent e) {
        AnchorPane root = null;
        FXMLLoader loader = new
        FXMLLoader(getClass().getResource("/view/listen.fxml"));
        try {
            root = loader.load();
        } catch(IOException ioe) {ioe.printStackTrace();}
        ListenController controller = loader.getController();
        controller.setRootController(this);
        changeNavigationColours(spnListen, icnListen, btnListen);
        setActivity(root);
    }
    @FXML
    public void goReview(ActionEvent e) {
        AnchorPane root = null;
        FXMLLoader loader = new
        FXMLLoader(getClass().getResource("/view/review.fxml"));
        try {
            root = loader.load();
        } catch(IOException ioe) {ioe.printStackTrace();}
        ReviewController controller = loader.getController();
        controller.setRootController(this);
        changeNavigationColours(spnReview, icnReview, btnReview);
        setActivity(root);
    }
    @FXML
    public void goSettings(MouseEvent e) {
        AnchorPane root = null;
        FXMLLoader loader = new
        FXMLLoader(getClass().getResource("/view/settings.fxml"));
        try {
            root = loader.load();
        } catch(IOException ex) {ex.printStackTrace();}
        SettingsController controller = loader.getController();
        controller.setRootController(this);
        changeNavigationColours(null, null, null); //null doesn't equal any
        of the objects of the activities so all will turn blue (unselected)
        setActivity(root);
    }

    //close the system
    @FXML
    public void close(MouseEvent e) {
        System.exit(0);
    }
}

```

LearnController.java

```

package controller;

import java.io.IOException;
import java.sql.Connection;
320

```

```

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Optional;

import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableSet;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.layout.AnchorPane;
import javafx.scene.control.TextInputDialog;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.VBox;
import other.DAO;

public class LearnController {
    private RootController rootController;
    private ObservableSet<String> titleSet = FXCollections.observableSet();
//only one unique title
    private DAO dao = new DAO();
    @FXML private VBox vbxSets;
    @FXML private AnchorPane apnLearn;
    public LearnController() {
        //pack the vbox with existing sets
        initSets();
    }
    public void initSets() {
        titleSet.clear();
        Platform.runLater(()-> {
            vbxSets.getChildren().clear();
        }); //clear all children nodes first so no repeats of sets occur

        try {
            Connection conn = dao.getConnection();
            PreparedStatement stmt = conn.prepareStatement("SELECT
Title FROM Theory WHERE Grade <= ?");
        } {
            stmt.setInt(1, RootController.getGrade());
            ResultSet rs = stmt.executeQuery();

            while(rs.next()) {
                String title = rs.getString("Title");
                if(!"".equals(title))
                    titleSet.add(title);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        Platform.runLater(()-> {
            for(String s: titleSet) {
                addRow(s);
            }
        });
    }
    // add a flashcard set row on the UI
    public void addRow(String title) {
        AnchorPane row = null;

```

```

        FXMLLoader rowLoader = new
FXMLLoader(getClass().getResource("/view/learn_row.fxml"));
    try {
        row = rowLoader.load();
    } catch (IOException e) {e.printStackTrace();}
    LearnRowController rowController = rowLoader.getController();
    rowController.initData(title, rootController, this, apnLearn);
    vbxSets.getChildren().add(row);
}
public void setRootController(RootController rootController) {
    this.rootController = rootController;
}

@FXML
public void createSet(MouseEvent e) {
    //get the title
    TextInputDialog titleDialog = new TextInputDialog("");
    titleDialog.setTitle("Title for your set");
    titleDialog.setHeaderText("Please provide a title for your new
flashcard set");
    Optional<String> result = titleDialog.showAndWait();
    if(result.isPresent() && !result.get().equals("")) {
        AnchorPane root = null;
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("/view/learn_update.fxml"));
        try {
            root = loader.load();
        } catch(IOException ex) {ex.printStackTrace();}
        LearnUpdateController controller = loader.getController();

        //add initial few flashcards
        for(int i =0; i<5; i++)controller.addRow();

        controller.initData(result.get(), rootController, this,
apnLearn);
        rootController.setActivity(root);
        root.requestFocus();
    } else {
        apnLearn.requestFocus();
    }
}
}

```

LearnRowController.java

```

package controller;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Optional;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;

```

```

import javafx.scene.layout.AnchorPane;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.ButtonType;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import other.DAO;

public class LearnRowController {
    private RootController rootController;
    private LearnController learnController;
    private AnchorPane apnLearn;
    private DAO dao = new DAO();

    @FXML private Text txtTitle;
    @FXML private AnchorPane apnSetRow;
    public void initData (String title, RootController rc, LearnController lc,
AnchorPane apn) {
        txtTitle.setText(title);
        rootController = rc;
        learnController = lc;
        apnLearn = apn;
    }

    //play the flashcards when play button clicked
    @FXML
    public void playFlashcards(MouseEvent e) {
        AnchorPane root = null;
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("/view/learn_flashcards.fxml"));
        try {
            root = loader.load();
        } catch(IOException ex) {ex.printStackTrace();}
        LearnFlashcardController controller = loader.getController();
        controller.initData(txtTitle.getText(), rootController, apnLearn);
        controller.initKeys();
        rootController.setActivity(root);

    }
    //edit the flashcard set
    @FXML
    public void editSet(MouseEvent e) {
        AnchorPane root = null;
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("/view/learn_update.fxml"));
        try {
            root = loader.load();
        } catch(IOException ex) {ex.printStackTrace();}
        LearnUpdateController controller = loader.getController();
        controller.initData(txtTitle.getText(), rootController,
learnController, apnLearn);
        rootController.setActivity(root);
        root.requestFocus();
    }
    //delete the flashcard set
    @FXML
    public void deleteSet(MouseEvent e) {

```

```

        //confirm first
        Alert confirmAlert = new Alert(AlertType.CONFIRMATION, "", 
ButtonType.YES, ButtonType.NO);
        confirmAlert.setHeaderText("Are you sure you want to delete this
set?");
        Optional<ButtonType> result = confirmAlert.showAndWait();
        if(result.get() == ButtonType.YES) {
            //delete from database
            try {
                Connection conn = dao.getConnection();
                PreparedStatement stmt = conn.prepareStatement("DELETE
FROM Theory WHERE Title = ?");
            ) {
                conn.setAutoCommit(false);
                stmt.setString(1, txtTitle.getText());
                stmt.executeUpdate();
                conn.commit();
            } catch (SQLException ex) {
                ex.printStackTrace();
            }

            //delete from interface
            VBox vbxCards = (VBox) apnSetRow.getParent();
            vbxCards.getChildren().remove(apnSetRow);
        }
    }
}

```

LearnUpdateController.java

```

package controller;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableMap;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.control.ScrollPane;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import other.DAO;

public class LearnUpdateController {
    private RootController rootController;
    private LearnController learnController;
    private AnchorPane apnLearn;
    private DAO dao = new DAO();

    //map to refer to the respective controllers when needed

```

```

    private ObservableMap<AnchorPane, LearnUpdateRowController> controllerMap =
FXCollections.observableHashMap();

@FXML private Text txtNewTitle;
@FXML private VBox vbxCards;
@FXML private ScrollPane scrpnCards;

public void initData(String title, RootController rootController,
LearnController learnController, AnchorPane apnLearn) {
    //initialise data
    txtNewTitle.setText(title);
    this.rootController = rootController;
    this.learnController = learnController;
    this.apnLearn = apnLearn;

    scrpnCards.vvalueProperty().bind(vbxCards.heightProperty());
    //automatically scroll the scrollpane to the bottom

    //use title to load all existing cards
    try {
        Connection conn = dao.getConnection();
        PreparedStatement stmt = conn.prepareStatement("SELECT Term,
Definition FROM Theory WHERE Title = ? AND Grade <= ?");
    } {
        //initialise the existing flashcard rows
        stmt.setString(1, title);
        stmt.setInt(2, RootController.getGrade());
        ResultSet rs = stmt.executeQuery();
        while(rs.next()) {
            String term = rs.getString("Term");
            String definition = rs.getString("Definition");
            addRow(term, definition);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public ObservableMap<AnchorPane, LearnUpdateRowController>
getControllerMap() {
    return controllerMap;
}
public LearnUpdateRowController addRow(String term, String def) {
    LearnUpdateRowController controller = addRow();
    Platform.runLater(() -> {
        controller.setTerm(term);
        controller.setDef(def);
    });
    return controller;
}
public LearnUpdateRowController addRow() {
    AnchorPane row = null;
    FXMLLoader loader = new
FXMLLoader(getClass().getResource("/view/learn_update_row.fxml"));
    try {
        row = loader.load();
    } catch(IOException e) {e.printStackTrace();}
    LearnUpdateRowController controller = loader.getController();
}

```

```

        controller.initData(this);
        controllerMap.put(row, controller);
        vbxCards.getChildren().add(row);
        return controller;
    }
    //add an extra row when add flashcard button clicked
    @FXML
    public void addFlashcard(ActionEvent e) {
        addRow();
    }
    //finish when finish button clicked
    @FXML
    public void finish(ActionEvent e) {
        doFinish();
    }
    //finish when enter button clicked
    @FXML
    public void kFinish(KeyEvent e) {
        if(e.getCode() == KeyCode.ENTER)
            doFinish();
    }
    public void doFinish() {
        try {
            Connection conn = dao.getConnection();
            PreparedStatement deleteStmt =
conn.prepareStatement("DELETE FROM Theory WHERE Title = ? AND Grade <= ?");
            PreparedStatement stmt = conn.prepareStatement("INSERT
INTO Theory (Term, Definition, Grade, Title) VALUES (?, ?, ?, ?)");
            ) {
                //delete old terms
                conn.setAutoCommit(false); //doesn't unpredictably
commit the changes

                deleteStmt.setString(1, txtNewTitle.getText());
                deleteStmt.setInt(2, RootController.getGrade());
                deleteStmt.executeUpdate();

                //add updated terms/new terms
                boolean termsAdded = false;
                for(LearnUpdateRowController c: controllerMap.values())
{
                    if(!(c.getTerm().equals("") ||
c.getDef().equals("")))) { //not blank space
                        termsAdded = true;
                        stmt.setString(1, c.getTerm());
                        stmt.setString(2, c.getDef());
                        stmt.setInt(3, RootController.getGrade());
                        stmt.setString(4, txtNewTitle.getText());
                        stmt.addBatch();
                    }
                }
                if(termsAdded) //if terms WERE added, then execute
batch otherwise will cause error on adding nothing
                stmt.executeBatch();

                conn.commit();
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
        }
    }
}

```

```

        }

        //go back to main learn screen
        rootController.setActivity(apnLearn);
        learnController.initSets();
        apnLearn.requestFocus();
    }
}

```

LearnUpdateRowController.java

```

package controller;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;

public class LearnUpdateRowController {
    @FXML private TextField fldTerm, fldDef;
    @FXML private AnchorPane apnCardRow;
    private LearnUpdateController updateController = null;
    public void initData(LearnUpdateController c) {
        updateController = c;
    }
    //remove the row
    @FXML
    public void removeCard(ActionEvent e) {
        VBox vbxCards = (VBox) apnCardRow.getParent(); //remove from update
interface
        vbxCards.getChildren().remove(apnCardRow);
        //remove controller
        updateController.getControllerMap().remove(apnCardRow);
    }

    //if enter is pressed when inside a textfield, then finish - submit the
changes
    @FXML
    public void finish(ActionEvent e) {
        updateController.doFinish();
    }

    //getter setter methods
    public void setTerm(String term) { //set foreign term
        fldTerm.setText(term);
    }
    public void setDef(String def) { //set definition
        fldDef.setText(def);
    }
    public String getTerm() {
        return fldTerm.getText();
    }
    public String getDef() {
        return fldDef.getText();
    }
}

```

TestController.java

```
package controller;

import java.io.IOException;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.layout.AnchorPane;

public class TestController {
    private RootController rootController;
    public void setRootController(RootController rootController) {
        this.rootController = rootController;
    }
    //load test quiz
    @FXML
    public void playTest(ActionEvent e) {
        AnchorPane root = null;
        try {
            FXMLLoader loader = new
            FXMLLoader(getClass().getResource("/view/test_quiz.fxml"));
            root = loader.load();
            TestQuizController controller = loader.getController();
            controller.initData(rootController);
            rootController.setActivity(root);
        } catch(IOException ex) {ex.printStackTrace();}
    }
}
```

TestQuizController.java

```
package controller;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.LinkedList;
import java.util.Random;

import com.jfoenix.controls.JFXButton;
import com.jfoenix.controls.JFXProgressBar;

import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
```

```

import model.Feedback;
import model.TheoryQuestion;
import other.DAO;

public class TestQuizController {
    //important data variables/structures
    private RootController rootController;
    private DAO dao = new DAO();
    private Random rand = new Random();
    private int questionIndex = 0;
    private LinkedList<TheoryQuestion> theoryQuestionList = new LinkedList<>();
    private LinkedList<Feedback> feedbackList = new LinkedList<>();
    private ObservableList<String> correctTerms =
FXCollections.observableArrayList();
    private boolean questionAnswered = false;
    //UI objects
    @FXML private Text txtQuestion;
    @FXML private JFXButton btnOk, btnOne, btnTwo, btnThree, btnFour;
    @FXML private JFXProgressBar prbProgress;
    @FXML private AnchorPane apnQuiz;

    //collection of the multiple choice answer buttons
    private JFXButton[] answerButtons;

    public void initData(RootController rootController) {
        this.rootController = rootController;
        answerButtons = new JFXButton[] {btnOne, btnTwo, btnThree,
btnFour};

        //collect questions from database
        getTheoryQuestions();
        Platform.runLater(()->loadQuestion());
    }
    public void getTheoryQuestions() {
        try {
            Connection conn = dao.getConnection();
            PreparedStatement priorityCntStmt =
conn.prepareStatement("SELECT COUNT(*) FROM Theory WHERE Grade <= ? AND
K_Coefficient < 0");
            PreparedStatement priorityStmt = conn.prepareStatement("SELECT
Term, Definition FROM Theory WHERE Grade <= ? AND K_Coefficient < 0 ORDER BY
K_Coefficient ASC");
            PreparedStatement cntStmt = conn.prepareStatement("SELECT
COUNT(*) FROM Theory WHERE Grade <= ?");
            PreparedStatement rndStmt = conn.prepareStatement("SELECT
Term, Definition FROM Theory WHERE Grade <= ?", ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
        } {
            //execute queries
            cntStmt.setInt(1, RootController.getGrade());
            ResultSet cntRs = cntStmt.executeQuery();
            rndStmt.setInt(1, RootController.getGrade());
            ResultSet rndRs = rndStmt.executeQuery();
            priorityCntStmt.setInt(1, RootController.getGrade());
            ResultSet priorityCntRs = priorityCntStmt.executeQuery();
            priorityStmt.setInt(1, RootController.getGrade());
            ResultSet priorityRs = priorityStmt.executeQuery();

```

```

        //prioritise the low k_coefficient terms (highly answered
wrong)
        int priorityLength = 0;
        if(priorityCntRs.next())priorityLength =
priorityCntRs.getInt(1); //number of high priority terms

        //get length to use for upper bound of random num for filled
answers
        int length = 0;
        if(cntRs.next())length = cntRs.getInt(1); //upperbound of
random

        //get priority terms
        if(priorityLength > 0) {
            for(int i =0; i<(priorityLength >=10 ? 10:
priorityLength); i++) { //if there are more than 10 priority terms, only take the
top 10 highest priority (lowest k_coefficient), otherwise take as much as you can
                String question = "";
                String[] answers = new String[4];
                int answerIndex = rand.nextInt(4);
                if(priorityRs.next()) {
                    answers[answerIndex] =
priorityRs.getString("Definition");
                    question = priorityRs.getString("Term");
                }
                //fill other answers
                for(int j = 0; j<4; j++) { //four answers (3
random 1 already selected high priority one)
                    if(j == answerIndex)
                        continue;

                    int randomRow = rand.nextInt(length);
                    for(int k=0;k<=randomRow;k++) { //move
pointer to the random row; <= because you want to call next() at least once to go
past zeroth row
                        rndRs.next();
                    }
                    answers[j] =
rndRs.getString("Definition"); // put random answer into array
                    rndRs.beforeFirst(); //back to start
                }
                //add question
                theoryQuestionList.add(new
TheoryQuestion(question, answers, answerIndex));
            }
        }

        //get all the rest of the terms and select the randomly
        for(int questionNumber = 0; questionNumber < 10 -
priorityLength; questionNumber++) { //create (ten - already filled) questions
            int answerIndex = rand.nextInt(4);
            String question = "";
            String[] answers = new String[4];
            //get four random answers + one matching foreign term
            (question)
            for(int i =0; i<4; i++ ) { //four random answers
                int randomRow = rand.nextInt(length);

```

```

        for(int j=0; j <= randomRow; j++) { //move
pointer to the random row; <= because you want to call next() at least once to go
past zeroth row
            rndRs.next();
        }
        answers[i] = rndRs.getString("Definition");
        if(answerIndex==i)
            question = rndRs.getString("Term");
        rndRs.beforeFirst(); //reset pointer for next
random answer
    }
    //add question
    theoryQuestionList.add(new TheoryQuestion(question,
answers, answerIndex));
}
} catch (SQLException e) {
    e.printStackTrace();
}

}

public void loadQuestion() {
    TheoryQuestion theoryQuestion =
theoryQuestionList.get(questionIndex);

    //question
    txtQuestion.setText(questionIndex % 2 == 0?
"What is the definition of " + theoryQuestion.getQuestion() +
"?":
"What does " + theoryQuestion.getQuestion() + " mean?" );

    //answers
    for(int i = 0; i < 4; i++) {
        answerButtons[i].setText(theoryQuestion.getAnswer(i));
    }

    apnQuiz.requestFocus(); //so it doesn't highlight a random button
}
public void answerQuestion(int answer) {
    TheoryQuestion currentQuestion =
theoryQuestionList.get(questionIndex);
    int correctAnswer = currentQuestion.getAnswerIndex();
    boolean pandaTheme = RootController.getTheme().equals("The Panda
Special");
    //change colours used for feedback
    answerButtons[answer].setStyle("-fx-background-color: #6f0f0f; -fx-
text-fill: white;" + (pandaTheme? "-fx-background-image: null": ""));
    //answered
button is red
    answerButtons[correctAnswer].setStyle("-fx-background-color:
#0cb794; -fx-text-fill: white;" + (pandaTheme? "-fx-background-image: null": "")); //if your answer was right, it will override the colour to green

    //if wrong, note down for feedback
    if(answer==correctAnswer)
        correctTerms.add(currentQuestion.getQuestion());
    else

```

```

        feedbackList.add(new Feedback(currentQuestion.getQuestion(),
currentQuestion.getAnswer(answer), currentQuestion.getAnswer(correctAnswer)));
        //button to move on
        btnOk.setVisible(true);
    }
    //when user clicks on an answer
    @FXML
    public void giveAnswer(ActionEvent e) {
        if(questionAnswered == false) {
            questionAnswered = true;
            JFXButton clickedButton = (JFXButton) e.getSource();

            for(int i=0; i<4; i++) { //check which button is the answered
button and then answer the question with its index
                if(clickedButton.equals(answerButtons[i]))
                    answerQuestion(i);
            }
        }
    }
    //when OK button is clicked
    @FXML
    public void nextQuestion(ActionEvent e) {
        if(++questionIndex==10) {
            loadResults();
        } else {
            questionAnswered= false;
            prbProgress.setProgress((questionIndex)/10.0); //increase
progress
            btnOk.setVisible(false);//hide ok

            //make all answer boxes white again with orange text
            for(JFXButton btn: answerButtons) {
                String addPanda = RootController.getTheme().equals("The
Panda Special") ? "-fx-background-image: url('resources/img/panda-
background1.png')": "";
                btn.setStyle("-fx-background-color: #ffffff; -fx-text-
fill:" + RootController.getColourCode(1) + addPanda);
            }
        }
        loadQuestion(); //load new question with incremented
questionIndex
    }
}

//load the results
public void loadResults() {
    AnchorPane root = null;
    FXMLLoader loader = new
FXMLLoader(getClass().getResource("/view/test_results.fxml"));
    try {
        root = loader.load();
    } catch(IOException ex) {ex.printStackTrace();}
    TestResultsController controller = loader.getController();
    controller.initData(rootController, feedbackList, 10-
feedbackList.size(), correctTerms);
    rootController.setActivity(root);
}

```

```

        root.requestFocus(); //so doesn't randomly highlight a button
    }
}

```

TestResultsController.java

```

package controller;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Calendar;
import java.util.LinkedList;

import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.Text;
import model.Feedback;
import other.DAO;

public class TestResultsController {
    //array of different comments, the index correlates to the score
    private String[] comments = new String[] {
        "Try learning the terms first!",
        "Have a peek at the learn section.",
        "Remember, there is the learn section just for you!",
        "It's fine, just learn from it!",
        "Practice makes perfect",
        "You can do better next time!",
        "Don't be shy to keep going!",
        "Not bad!",
        "Keep going!",
        "Almost there!",
        "Wow, you did it!"
    };
    //other important variables
    private LinkedList<Feedback> feedbackList;
    private ObservableList<String> correctTerms;
    private RootController rootController;
    private int result;
    private DAO dao = new DAO();
    //ui objects
    @FXML private Text txtScore, txtComment;
    @FXML private VBox vbxFeedbacks;
    public void initData(RootController rc, LinkedList<Feedback> fbl, int
result, ObservableList<String> ct) {
        rootController = rc; //set rootcontroller for activity changes
        feedbackList = fbl; //list of feedback

```

```

        this.result = result; //take the result of the test
        txtScore.setText("You scored " + result + "/10!");
        txtComment.setText(comments[result]); //a comment depending on the
result
        correctTerms = ct;

        //add the feedback to the vbox
        for(Feedback feedback: feedbackList) {
            Text txtFeedback = new Text();
            txtFeedback.setFont(Font.font("Century Gothic",
FontPosture.REGULAR, 16));
            txtFeedback.setText("For " + feedback.getQuestion() + ", it's
not " + feedback.getWrong() + " but it is " + feedback.getCorrect()+"\n");
            vbxFeedbacks.getChildren().add(txtFeedback);
        }

        // store into sql
        updateDB();
    }
    public void updateDB() {
        try{
            Connection conn = dao.getConnection();
            PreparedStatement checkKStmt = conn.prepareStatement("SELECT
K_Coefficient FROM Theory Where Term = ?"); //check the current k coefficient
            PreparedStatement higherKStmt = conn.prepareStatement("UPDATE
Theory SET K_Coefficient = K_Coefficient + 1 WHERE Term = ?"); //for correct
answers
            PreparedStatement lowerKStmt = conn.prepareStatement("UPDATE
Theory SET K_Coefficient = K_Coefficient - 1 WHERE Term = ?"); //for wrong answers

            //record score for review section
            PreparedStatement oldScoreStmt = conn.prepareStatement("SELECT
Highscore FROM Score WHERE ScoreDay = ? AND ScoreMonth = ? AND ScoreYear = ?");
            PreparedStatement insScoreStmt = conn.prepareStatement("INSERT
INTO Score (ScoreDay, ScoreMonth, ScoreYear, Highscore) VALUES (?, ?, ?, ?)");
            PreparedStatement updScoreStmt = conn.prepareStatement("UPDATE
Score SET Highscore = ? WHERE ScoreDay = ? AND ScoreMonth = ? AND ScoreYear = ?");
        } {
            conn.setAutoCommit(false);
            //changes to K_Coefficient to adapt for next test
            for(String term: correctTerms) {
                checkKStmt.setString(1, term);
                ResultSet checkRs = checkKStmt.executeQuery();
                int K_Coefficient = 0;
                if(checkRs.next())K_Coefficient =
checkRs.getInt("K_Coefficient");

                if(K_Coefficient < 1) {
                    higherKStmt.setString(1, term);
                    higherKStmt.addBatch();
                }
            }
            if(correctTerms.size()>0)
                higherKStmt.executeBatch(); //larger than zero
otherwise batch update won't work
        }
    }
}

```

```

        for(Feedback wrongTerms: feedbackList) { //all terms answered
            wrongly will be noted for next term (by lowering the K_Coefficient)
                lowerKStmt.setString(1, wrongTerms.getQuestion());
                lowerKStmt.addBatch();
            }
        if(feedbackList.size()>0)lowerKStmt.executeBatch();

        //add highscore into db
        //check old score if it has improved
        int[] date = getDate();
        oldScoreStmt.setInt(1, date[0]);
        oldScoreStmt.setInt(2, date[1]);
        oldScoreStmt.setInt(3, date[2]);
        ResultSet oldScoreRs = oldScoreStmt.executeQuery();
        int hiscore = -1;
        if(oldScoreRs.next())hiscore = oldScoreRs.getInt("Highscore");
    //if a score exists for this date, take it for current the hiscore
        if(hiscore===-1) { //score doesn't already exist
            insScoreStmt.setInt(1, date[0]);
            insScoreStmt.setInt(2, date[1]);
            insScoreStmt.setInt(3, date[2]);
            insScoreStmt.setInt(4, result);
            insScoreStmt.executeUpdate();
        } else if(result > hiscore) { //if score exists and actually
    has improved
            updScoreStmt.setInt(1, result);
            updScoreStmt.setInt(2, date[0]);
            updScoreStmt.setInt(3, date[1]);
            updScoreStmt.setInt(4, date[2]);
            updScoreStmt.executeUpdate();
        }

        //commit the changes to the database
        conn.commit();
    } catch(SQLException ex) {
        ex.printStackTrace();
    }
}
//a method to make it easier to access the current date
public int[] getDate() {
    Calendar now = Calendar.getInstance();
    int day = now.get(Calendar.DAY_OF_MONTH);
    int month = now.get(Calendar.MONTH) + 1; //zero indexed, +1 to make
january 1, feb 2, ...
    int year = now.get(Calendar.YEAR);

    return new int[] {day, month, year};
}
@FXML public void playAgain(ActionEvent e) {
    //go back to starting screen
    AnchorPane root = null;
    FXMLLoader loader = new
FXMLLoader(getClass().getResource("/view/test.fxml"));
    try {
        root = loader.load();
    } catch(IOException ioe) {ioe.printStackTrace();}
    TestController controller = loader.getController();
}

```

```

        controller.setRootController(rootController);
        rootController.setActivity(root);

        root.requestFocus(); //so doesn't randomly highlight a button
    }
}

```

ListenController.java

```

package controller;

import java.io.IOException;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.layout.AnchorPane;

public class ListenController {
    //RootController object for changing mutual data and current activity being
    displayed
    private RootController rootController;

    public void setRootController(RootController rootController) {
        this.rootController = rootController;
    }

    //show the quiz
    @FXML
    public void playQuiz(ActionEvent e) {
        AnchorPane root = null;
        FXMLLoader loader = new
        FXMLLoader(getClass().getResource("/view/listen_quiz.fxml"));
        try {
            root = loader.load();
        } catch(IOException ex) {
            ex.printStackTrace();
        }
        ListenQuizController controller = loader.getController();
        controller.initData(rootController);
        rootController.setActivity(root);

        root.requestFocus(); //doesn't highlight a random button
    }
}

```

ListenQuizController.java

```

package controller;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Random;

```

```

import com.jfoenix.controls.JFXButton;
import com.jfoenix.controls.JFXSlider;

import de.jensd.fx.glyphs.materialdesignicons.MaterialDesignIconView;
import javafx.animation.FadeTransition;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Platform;
import javafx.beans.InvalidationListener;
import javafx.beans.Observable;
import javafx.collections.ListChangeListener.Change;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.layout.AnchorPane;
import javafx.scene.control.Button;
import javafx.scene.input.MouseEvent;
import javafx.util.Duration;
import model.AuralTest;
import other.DAO;

public class ListenQuizController {
    //important instance variables
    private RootController rootController;
    private DAO dao = new DAO();
    private AuralTest currentTest = null;
    private boolean answered = false;
    private boolean finished = false;
    private boolean fading = false;
    //UI objects
    @FXML private JFXButton btnRelMaj, btnRelMin, btnSub, btnDom, btnAgain;
    @FXML private MaterialDesignIconView icnPlayPause;
    @FXML private JFXSlider sldSeek;
    @FXML private AnchorPane apnFirstPlay, apnQuiz;

    public void initData(RootController rootController) {
        this.rootController = rootController;

        //style the slider dependent on the theme
        switch(RootController.getTheme()) {
            case "Default Chamber":
                sldSeek.getStylesheets().add("resources/styles/default-slider.css");
                break;
            case "Future Sea":
                sldSeek.getStylesheets().add("resources/styles/future-slider.css");
                break;
            case "Rose Petal":
                sldSeek.getStylesheets().add("resources/styles/rose-slider.css");
                break;
            case "The Panda Special":
                sldSeek.getStylesheets().add("resources/styles/panda-slider.css");
                break;
        }
    }
}

```

```

        getAuralTest();
        syncAudio();
    }
    //get a random aural test to be used
    private void getAuralTest() {
        try {
            Connection conn = dao.getConnection();
            PreparedStatement cntStmt = conn.prepareStatement("SELECT
COUNT(*) FROM Aural");
            ResultSet cntRs = cntStmt.executeQuery();
            PreparedStatement rndStmt = conn.prepareStatement("SELECT
File, Modulation FROM Aural");
            ResultSet rndRs = rndStmt.executeQuery();
        } {
            int length = 0; //the upper bound for random row
            if(cntRs.next())length = cntRs.getInt(1);
            int randomRow = (new Random()).nextInt(length); //floor for
lower bound to generate 'length' number of integers (inc 0, exc end)

            for(int i = 0; i<=randomRow; i++)rndRs.next(); //go to random
row <= to move past zeroth row

            currentTest = new AuralTest(rndRs.getString("File"),
rndRs.getString("Modulation"));
        } catch(SQLException ex) {
            ex.printStackTrace();
        }
    }

    //sync the slider with the audio's position in time
    private void syncAudio() {

        currentTest.getMediaPlayer().currentTimeProperty().addListener(new
InvalidateListener() { //to auto sync the slider with the media player being
played
            @Override
            public void invalidated(Observable o) {
                Duration currentDuration =
currentTest.getMediaPlayer().getCurrentTime();
                Duration totalDuration =
Duration.millis(currentTest.getDuration());
                sldSeek.setDisable(totalDuration.isUnknown());
//disable slider if the music has not been correctly loaded yet
                if(!sldSeek.isDisable() &&
totalDuration.greaterThan(Duration.ZERO)&& !sldSeek.isValueChanging()) {
                    double percentageIn =
(currentDuration.toMillis()/totalDuration.toMillis()) * 100;
                    sldSeek.setValue(percentageIn);
                    if(!answered)sldSeek.setDisable(true); //disable
                }
            }
        });
    }

    //pause when finished so it doesn't automatically play when slider
is moved back
    currentTest.getMediaPlayer().setOnEndOfMedia(()-> {
        finished = true;
        currentTest.getMediaPlayer().pause();

```

```

        icnPlayPause.setGlyphName("PLAY");
    });

}

//automatically stops after first play
public void bindAutoStop() {
    AnchorPane apnActivity = (AnchorPane) apnQuiz.getParent();
    ObservableList<Node> childrenList = apnActivity.getChildren();
    childrenList.addListener((Change<? extends Node> c) -> {
        while(c.next()) {
            if(c.wasRemoved()) {
                for(Node child: c.getRemoved()) {
                    if(child.equals(apnQuiz)) //quiz removed
                        currentTest.getMediaPlayer().stop();
                }
            }
        }
    });
}

//allow user to seek through the audio when answered question
public void allowSeeking() {
    sldSeek.valueProperty().addListener(new InvalidationListener() {
        @Override
        public void invalidated(Observable o) {
            if(sldSeek.isValueChanging()) { //if the slider is
                truly moving
                    Duration newDuration =
Duration.millis(currentTest.getDuration()).multiply(sldSeek.getValue()/100.0);
                    currentTest.getMediaPlayer().seek(newDuration);
                    if(sldSeek.getValue()>=98) {
                        finished = true;
                        currentTest.getMediaPlayer().pause();
                        icnPlayPause.setGlyphName("PLAY");
                    } else {
                        finished = false;
                    }
                }
            }
        });
}

//change UI in FX application thread
Platform.runLater(()->{
    sldSeek.setDisable(false);
    //slider now includes its thumb to click on
    sldSeek.getStylesheets().add("resources/styles/slider-
enable.css");
    icnPlayPause.setVisible(true);
});
}

//play or pause the music
@FXML
public void playPause(MouseEvent e) {
    if(icnPlayPause.getGlyphName().equals("PLAY")) {
        if(finished) {

```

```

        currentTest.getMediaPlayer().seek(currentTest.getMediaPlayer().getStartTime
   ()));
                finished = false;
            }
            currentTest.getMediaPlayer().play();
            icnPlayPause.setGlyphName("PAUSE");
        } else {
            currentTest.getMediaPlayer().pause();
            icnPlayPause.setGlyphName("PLAY");
        }
    }

}
//first play clicked
@FXML
public void firstPlay(ActionEvent e) {
    bindAutoStop();
    //fade
    if(!fading) {
        fading = true;
        FadeTransition fadeOut = new
FadeTransition(Duration.seconds(1), apnFirstPlay);
        fadeOut.setFromValue(1.0);
        fadeOut.setToValue(0.0);
        fadeOut.play();
        Timeline invisible = new Timeline(new
KeyFrame(Duration.seconds(0)), new KeyFrame(Duration.seconds(1), event ->{
            apnFirstPlay.setVisible(false);
            apnQuiz.requestFocus();
        }));
        invisible.play();
        currentTest.getMediaPlayer().play();
    }
}
//when answered question
@FXML
public void answerQuestion(ActionEvent e) {
    if(!answered) { //if hasn't been answered yet - this will make
clicks to other buttons not change their colour
        answered = true;
        boolean pandaTheme = RootController.getTheme().equals("The
Panda Special");
        //make input red
        Button inputBtn = (Button) e.getSource();
        inputBtn.setStyle("-fx-background-color: #6f0f0f; -fx-text-
fill: white;" +(pandaTheme? "-fx-background-image: null;": ""));
        //make correct answer green - correct input will be overrided
green also
        switch(currentTest.getModulation()) {
            case "Relative Major": btnRelMaj.setStyle("-fx-
background-color: #0cb794; -fx-text-fill: white;" +(pandaTheme? "-fx-background-
image: null;": ""));
            break;
            case "Relative Minor": btnRelMin.setStyle("-fx-
background-color: #0cb794; -fx-text-fill: white;" +(pandaTheme? "-fx-background-
image: null;": ""));
            break;
        }
    }
}

```

```

        case "Dominant": btnDom.setStyle("-fx-background-color:
#0cb794; -fx-text-fill: white;" +(pandaTheme? "-fx-background-image: null;": ""));
                break;
        case "Subdominant": btnSub.setStyle("-fx-background-
color: #0cb794; -fx-text-fill: white;" +(pandaTheme? "-fx-background-image: null;":
""));
                break;
    }
    btnAgain.setVisible(true); //show play again
    allowSeeking(); //let the reflection and correction begin
}
}

//when user clicks 'play again'
@FXML
public void playAgain(ActionEvent e) {
    currentTest.getMediaPlayer().stop(); //stop the music if it's still
playing

    //load again the quiz page
    AnchorPane root = null;
    FXMLLoader loader = new
FXMLLoader(getClass().getResource("/view/listen_quiz.fxml"));
    try {
        root = loader.load();
    } catch(IOException ex) {ex.printStackTrace();}
    ListenQuizController controller = loader.getController();
    controller.initData(rootController);
    rootController.setActivity(root);

    root.requestFocus(); //so doesn't focus on a button
}
}

```

ReviewController.java

```

package controller;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Calendar;

import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.fxml.FXML;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.LineChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import other.DAO;

public class ReviewController {
    private RootController rootController;

```

```

//stores data for charts
private XYChart.Series<Number, Number> dailySeries = new
XYChart.Series<>();
private XYChart.Series<String, Number> monthlySeries = new
XYChart.Series<>();
private XYChart.Series<Number, Number> yearlySeries = new
XYChart.Series<>();
//dao object for accessing the database
private DAO dao = new DAO();
//UI object
@FXML private AnchorPane apnDaily, apnMonthly, apnYearly;
@FXML private Text txtProgress;

public void setRootController(RootController rootController) {
    this.rootController = rootController;
}
public void initialize() { //called automatically after FXML injection

    Platform.runLater(()-> {
        //change title text
        txtProgress.setText(RootController.getName() + "'s All-Time
Progress");
        //set the graph data and display it
        getScores();
        setDailyGraph();
        setMonthlyGraph();
        setYearlyGraph();
    });
}
//reusable methods to get the date more efficiently and cleanly
public int getDate() {
    Calendar now = Calendar.getInstance();
    return now.get(Calendar.DAY_OF_MONTH);
}
public String getMonth(int num) { //num = -1 for NOW month, 0 <= num <= 11
for inputs
    Calendar now = Calendar.getInstance();
    String month = null;
    switch(num===-1 ? now.get(Calendar.MONTH) : num) {
        case 0: month = "Jan";
            break;
        case 1: month = "Feb";
            break;
        case 2: month = "Mar";
            break;
        case 3: month = "Apr";
            break;
        case 4: month = "May";
            break;
        case 5: month = "Jun";
            break;
        case 6: month = "Jul";
            break;
        case 7: month = "Aug";
            break;
        case 8: month = "Sep";
            break;
        case 9: month = "Oct";
            break;
    }
}

```

```

        break;
    case 10: month = "Nov";
        break;
    case 11: month = "Dec";
        break;
    }
    return month;
}
public int getYear() {
    Calendar now = Calendar.getInstance();
    return now.get(Calendar.YEAR);
}
//daily graph
public void setDailyGraph() {
    //set axes and customise
    final NumberAxis xAxis = new NumberAxis(1, 31, 1);
    final NumberAxis yAxis = new NumberAxis(0, 10, 1);
    xAxis.setLabel("Day of month");
    xAxis.setMinorTickVisible(false);
    xAxis.setAutoRanging(false);
    yAxis.setLabel("Hi-score");
    yAxis.setMinorTickVisible(false);
    yAxis.setAutoRanging(false);

    //the line chart object, axes specified in parameters
    final LineChart<Number, Number> dailyChart = new LineChart<>(xAxis,
yAxis);
    //customise chart
    dailyChart.setTitle(getMonth(-1) + " Progress");
    dailyChart.setMaxSize(680, 440);
    dailyChart.setMinSize(680, 440);
    dailyChart.setPrefSize(680, 440);
    dailyChart.setLegendVisible(false);
    //add data for chart
    dailyChart.getData().add(dailySeries);
    //add chart to interface
    apnDaily.getChildren().add(dailyChart);
}
//monthly graph
public void setMonthlyGraph() {
    final CategoryAxis xAxis = new
CategoryAxis(FXCollections.observableArrayList("Jan", "Feb", "Mar", "Apr", "May",
"Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"));
    final NumberAxis yAxis = new NumberAxis(0, 10, 1);
    xAxis.setLabel("Month in year");
    xAxis.setAutoRanging(false);
    yAxis.setLabel("Average score");
    yAxis.setMinorTickLength(0.25);
    yAxis.setAutoRanging(false);
    final LineChart<String, Number> monthlyChart = new
LineChart<>(xAxis, yAxis);
    monthlyChart.setTitle(getYear() + " Progress");
    monthlyChart.setMaxSize(680, 440);
    monthlyChart.setMinSize(680, 440);
    monthlyChart.setPrefSize(680, 440);
    monthlyChart.setLegendVisible(false);
    monthlyChart.getData().add(monthlySeries);
    apnMonthly.getChildren().add(monthlyChart);
}

```

```

        }
    //yearly graph
    public void setYearlyGraph() {
        final NumberAxis xAxis = new NumberAxis(2015, 2050, 1);
        final NumberAxis yAxis = new NumberAxis(0, 10, 1);
        xAxis.setAutoRanging(false);
        xAxis.setMinorTickVisible(false);
        xAxis.setLabel("Year");
        yAxis.setLabel("Average score");
        yAxis.setMinorTickLength(0.25);
        yAxis.setAutoRanging(false);
        final LineChart<Number, Number> yearlyChart = new LineChart<>(xAxis,
yAxis);
        yearlyChart.setTitle("Yearly Progress");
        yearlyChart.setMaxSize(680, 440);
        yearlyChart.setMinSize(680, 440);
        yearlyChart.setPrefSize(680, 440);
        yearlyChart.setLegendVisible(false);
        yearlyChart.getData().add(yearlySeries);
        apnYearly.getChildren().add(yearlyChart);
    }
    //get the data for the graphs first
    public void getScores() {
        //set series data
        try{
            Connection conn = dao.getConnection();
            PreparedStatement dailyStmt = conn.prepareStatement("SELECT
Highscore, ScoreDay FROM Score WHERE ScoreMonth = ? AND ScoreYear = ?");
            PreparedStatement monthlyStmt = conn.prepareStatement("SELECT
Highscore, ScoreMonth FROM Score WHERE ScoreYear = ? ORDER BY ScoreMonth");
            PreparedStatement yearlyStmt = conn.prepareStatement("SELECT
Highscore, ScoreYear FROM Score ORDER BY ScoreYear");
        ) {
            //daily graph
            Calendar now = Calendar.getInstance();
            dailyStmt.setInt(1, now.get(Calendar.MONTH)+1);
            dailyStmt.setInt(2, getYear());
            ResultSet dailyRs = dailyStmt.executeQuery();
            while(dailyRs.next()) {
                XYChart.Data<Number, Number> dayPoint = new
XYChart.Data<>(dailyRs.getInt("ScoreDay"), dailyRs.getInt("Highscore"));
                dailySeries.getData().add(dayPoint);
            }
            //monthly graph
            monthlyStmt.setInt(1, getYear());
            ResultSet monthlyRs = monthlyStmt.executeQuery();
            String lastMonth = "";
            double mTotal = 0;
            double mFreq = 0;
            while(monthlyRs.next()) {
                if(getMonth(monthlyRs.getInt("ScoreMonth"))-
1).equals(lastMonth)) {
                    mTotal += monthlyRs.getInt("Highscore");
                    mFreq++;
                } else {
                    if(!lastMonth.equals("")) { //after the first, if
the month changes, add the last month's avg scores to the graph

```

```

        XYChart.Data<String, Number> monthPoint =
new XYChart.Data<>(lastMonth, mTotal/mFreq); //total/frequency is the average
score of the month
                monthlySeries.getData().add(monthPoint);
            }
            //reinitialize variables (including first score)
            lastMonth =
getMonth(monthlyRs.getInt("ScoreMonth")-1); //method is 0 indexed, db is 1 indexed
so -1
                mTotal = monthlyRs.getInt("Highscore");
                mFreq = 1;
            }
        }
        //at the end, add the final last month (since it won't change
to another month after the last month)
        XYChart.Data<String, Number> finalMonthPoint = new
XYChart.Data<>(lastMonth, mTotal/mFreq);
        monthlySeries.getData().add(finalMonthPoint);

        //yearly graph
        ResultSet yearlyRs = yearlyStmt.executeQuery();
        int lastYear = -1;
        double yTotal = 0;
        double yFreq = 0;
        while(yearlyRs.next()) {
            if(yearlyRs.getInt("ScoreYear") == lastYear) {
                yTotal += yearlyRs.getInt("Highscore");
                yFreq++;
            } else {
                if(lastYear != -1) { //after the first, if the
year changes, add the last year's avg scores to the graph
                    XYChart.Data<Number, Number> yearPoint =
new XYChart.Data<>(lastYear, yTotal/yFreq); //total/frequency is the average score
of the year
                        yearlySeries.getData().add(yearPoint);
                }
                //reinitialize variables (including first score)
                lastYear = yearlyRs.getInt("ScoreYear"); //method
is 0 indexed, db is 1 indexed so -1
                yTotal = yearlyRs.getInt("Highscore");
                yFreq = 1;
            }
        }
        //at the end, add the final last month (since it won't change
to another year after the last year)
        XYChart.Data<Number, Number> finalYearPoint = new
XYChart.Data<>(lastYear, yTotal/yFreq);
        yearlySeries.getData().add(finalYearPoint);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

SettingsController.java

```
package controller;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.file.Files;
import java.nio.file.StandardOpenOption;
import java.util.Optional;
import java.util.Properties;

import com.jfoenix.controls.JFXComboBox;
import com.jfoenix.controls.JFXTextField;
import com.jfoenix.controls.JFXToggleButton;

import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.ButtonType;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.AnchorPane;
import other.DAO;

public class SettingsController {
    //ui objects
    @FXML private JFXComboBox<String> cbxTheme;
    @FXML private JFXComboBox<Integer> cbxGrade;
    @FXML private JFXTextField fldName;
    @FXML private JFXToggleButton tglGreeting;
    @FXML private AnchorPane apnSettings;
    //instance variables
    private RootController rootController;
    private File propertiesFile;
    private Properties settingsProps = new Properties();
    private DAO dao = new DAO();
    public void setRootController(RootController rootController) {
        this.rootController = rootController;
    }
    //fx application threads
    public void initialize() {

        Platform.runLater(() -> {
            //initialise the comboboxes
            cbxTheme.getItems().addAll("Default Chamber", "Future Sea",
"Rose Petal");
            cbxGrade.getItems().addAll(1, 2, 3, 4, 5, 6, 7, 8);

            //set initial values
            fldName.setText(RootController.getName());
            cbxGrade.setValue(RootController.getGrade());
            loadValues();

            apnSettings.requestFocus(); //no highlighted control
        });
    }
}
```

```

    });

    fldName.textProperty().addListener((o, ov, nv) -> {
        rootController.setName(nv);
    });
}

//load initial values from settings file (properties)
public void loadValues() {
    final String path = "C:\\\\iMProve\\\\settings.properties";
    //create file
    try {
        propertiesFile = new File(path);
        propertiesFile.createNewFile(); //if file already exists, does
nothing
    } catch(IOException ex) {ex.printStackTrace();}
    //properties
    try (InputStream in = Files.newInputStream(propertiesFile.toPath()))
{
    settingsProps.load(in);

    //get the properties
    String theme = settingsProps.getProperty("theme", "Default
Chamber");
    boolean greeting =
"true".equals(settingsProps.getProperty("greeting", "true")); //if true is what is
stored, return true; if stored value not equal to true, return false
    boolean panda = "true".equals(settingsProps.getProperty("panda
unlocked", "false"));

    //set control values
    if(panda)cbxTheme.getItems().add("The Panda Special");
    cbxTheme.setValue(theme);
    tglGreeting.setSelected(greeting);
} catch(IOException ex) {
    ex.printStackTrace();
}
}

@FXML
public void changeTheme(ActionEvent ae) {
    //code to change theme
    setTheme(cbxTheme.getValue());
}
//panda theme easter egg button clicked
@FXML
public void pandaTheme(MouseEvent me) {
    settingsProps.setProperty("panda unlocked", "true");
    if(!cbxTheme.getItems().contains("The Panda
Special"))cbxTheme.getItems().add("The Panda Special");
    setTheme("The Panda Special");
}
//change the ui for the theme
public void setTheme(String theme) {
    cbxTheme.setValue(theme); //for panda theme
    //change CSS
    Scene rootScene = apnSettings.getScene();
}

```

```

        rootScene.getStylesheets().clear();
        rootScene.getStylesheets().add(getCSS(theme));
        //change properties
        settingsProps.setProperty("theme", theme);
        saveProperties("theme changed");
        //change RootController properties and reset the colours of the
navigation bar
        RootController.setTheme(theme);
        rootController.changeNavigationColours(null, null, null);
    }

@FXML
public void changeGrade(ActionEvent ae) {
    //code to change grade
    RootController.setGrade(cbxGrade.getValue());
    //remove listen if lower than 8
    if(cbxGrade.getValue()<8) {
        rootController.removeListen();
    } else {
        rootController.addListen();
    }
}
@FXML
public void changeGreeting(ActionEvent ae) {
    //update properties for next time welcome
    settingsProps.setProperty("greeting",
String.valueOf(tglGreeting.isSelected()));
    saveProperties("greeting toggled");
}
//reset all the data in the system
@FXML
public void resetData(ActionEvent ae) {
    Alert confirmAlert = new Alert(AlertType.CONFIRMATION);
    confirmAlert.setHeaderText("Are you sure you want to reset your
data?");
    Optional<ButtonType> result = confirmAlert.showAndWait();
    if(result.isPresent() && result.get() == ButtonType.OK) {
        DAO.closeConnections();
        dao.reset(); //reset database
        settingsProps.clear(); //reset settings
        saveProperties("reset");

        cbxTheme.getItems().remove("The Panda Special");
        tglGreeting.setSelected(true);
        setTheme("Default Chamber");
        cbxTheme.setValue("Default Chamber");
    }
}
//get css file name of the theme
private String getCSS(String theme) {
    String fileName = "resources/themes/";
    switch(theme) {
        case "Future Sea": fileName += "future_sea.css";
            break;
        case "Default Chamber": fileName += "default_chamber.css";
            break;
    }
}

```

```
        case "Rose Petal": fileName += "rose_petal.css";
                            break;
        case "The Panda Special": fileName += "panda_special.css";
                            break;
    }
    return fileName;
}
//permanently make the changes to the properties
private void saveProperties(String comment) {
    try (OutputStream out =
Files.newOutputStream(propertiesFile.toPath(),
StandardOpenOption.TRUNCATE_EXISTING)){
        settingsProps.store(out, comment);
        out.close();
    } catch(IOException ex) {ex.printStackTrace();}
}
}
```

Evaluation

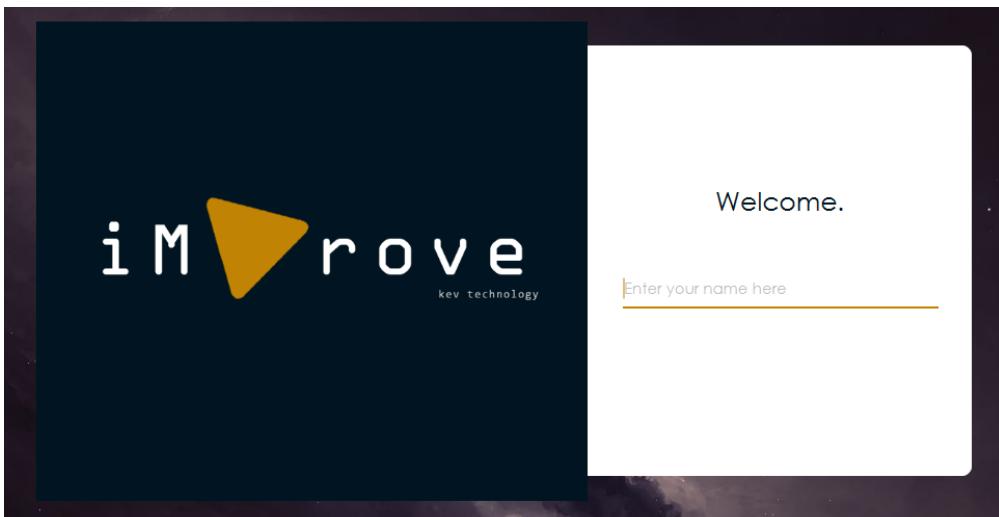
Testing

Following our test data for post-development, myself and the main stakeholder, Jill Thirkell performed a thorough test of the system.

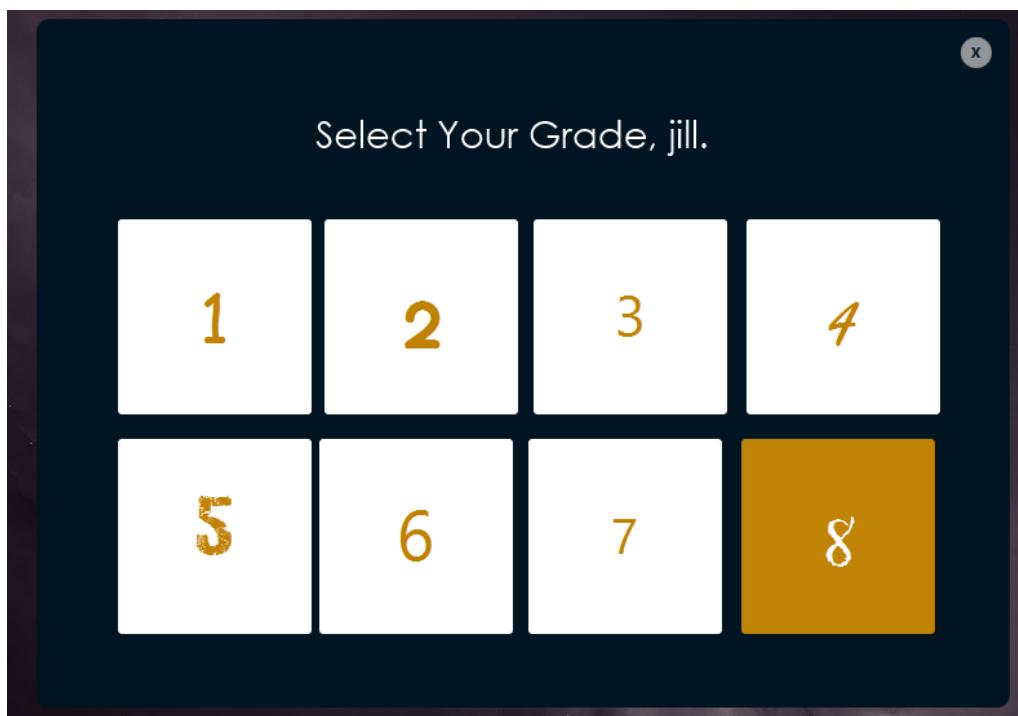
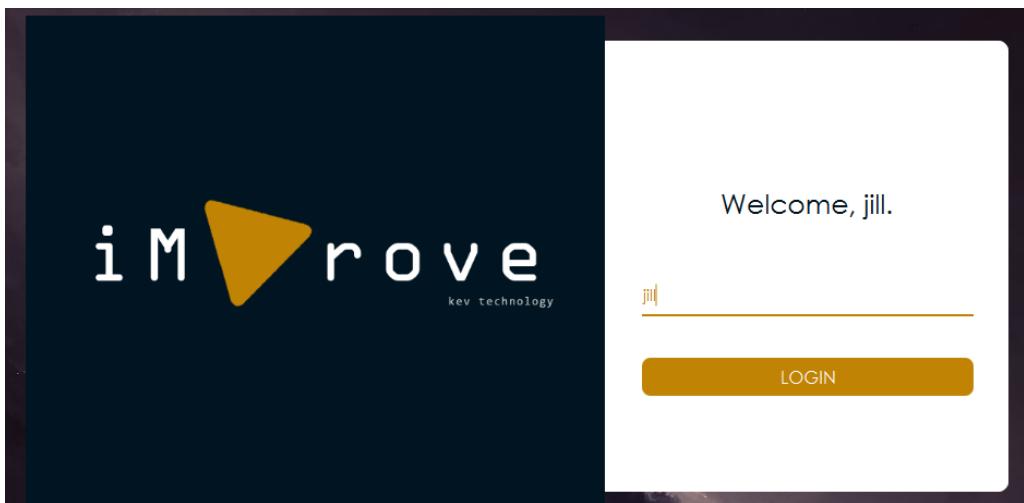
(Parts of the following tables are extracted from Test Data for Post-Development)

Test 1

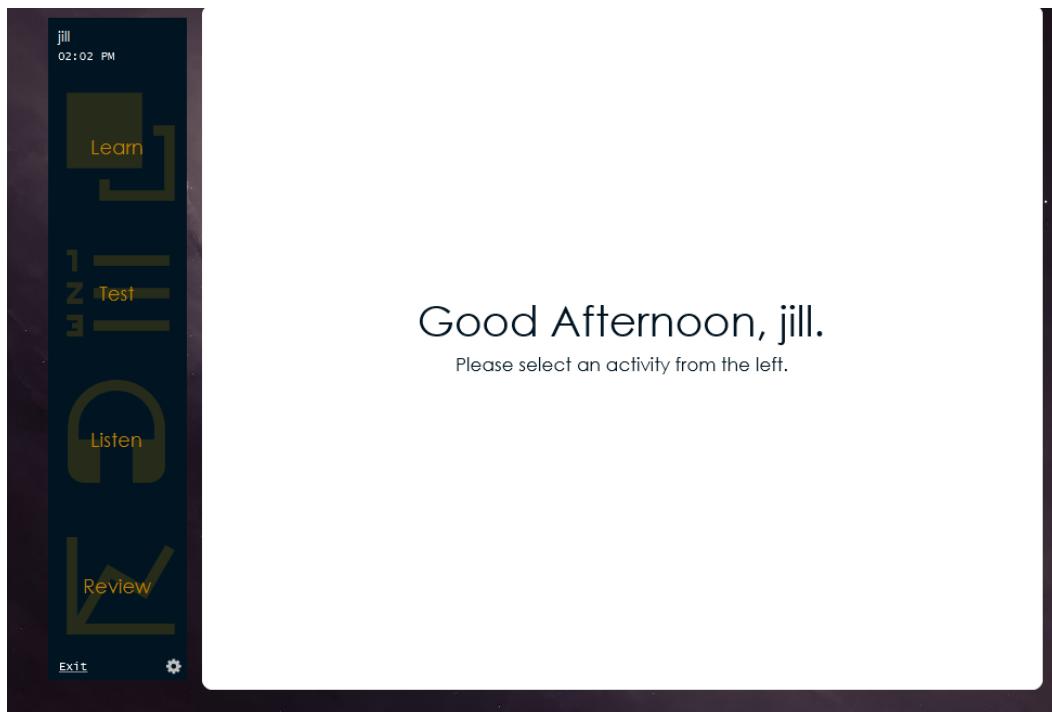
Test ID	Test Data	Success Criteria	Aspect	Justification	Expected Outcome	Actual Outcome
1	No name entered	(1) Grade Selection/Separation (21) Robust and Error-Free	Validation and Robustness	Ensures the user cannot progress the application with an invalid name, affecting later stages of the program	Cannot login	Cannot login



2	Login with "Jill Thirkell" entered as name	(1) Grade Selection/Separation (22) Easy to use	Function and Usability	This is valid data which will partly ensure the functionality of the tests. Also, will demonstrate if the grade selection is easy to perform.	Displays grade selection screen	Displays grade selection screen
---	--	--	------------------------	---	---------------------------------	---------------------------------

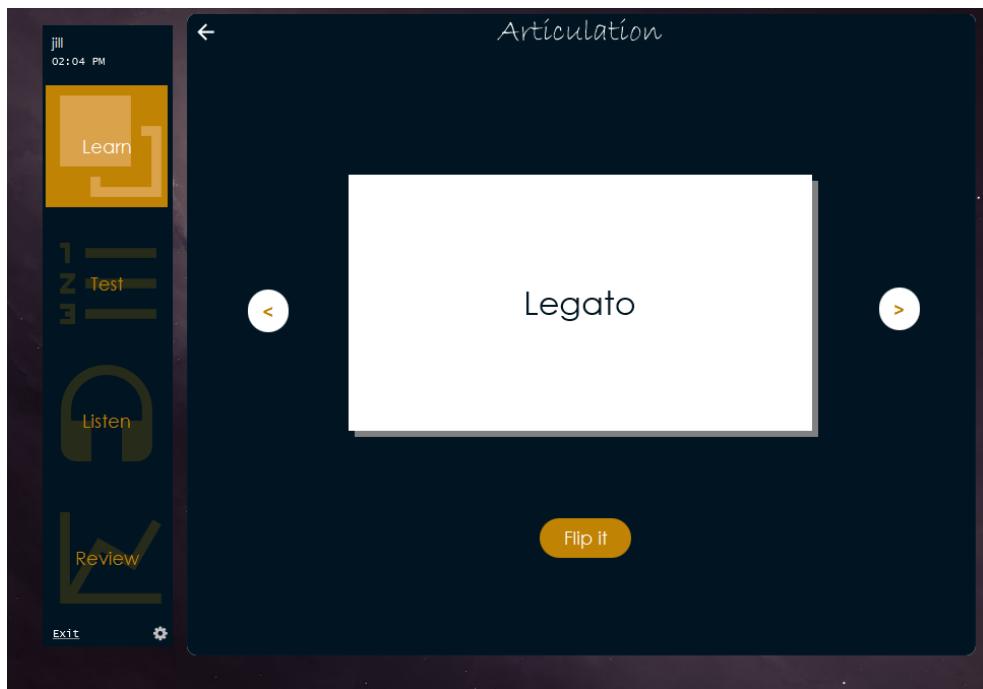


3	Examine the Welcome Screen	(20) Motivating (22) Easy to use	Usability	If the user easily understands what they can do now, then the system will be easy to use. Also, the welcome screen should partly motivate the user.	Provides instructions and an easy to use interface	Provides easy to understand instructions and a simple to interpret interface
---	----------------------------	-------------------------------------	-----------	---	--	--

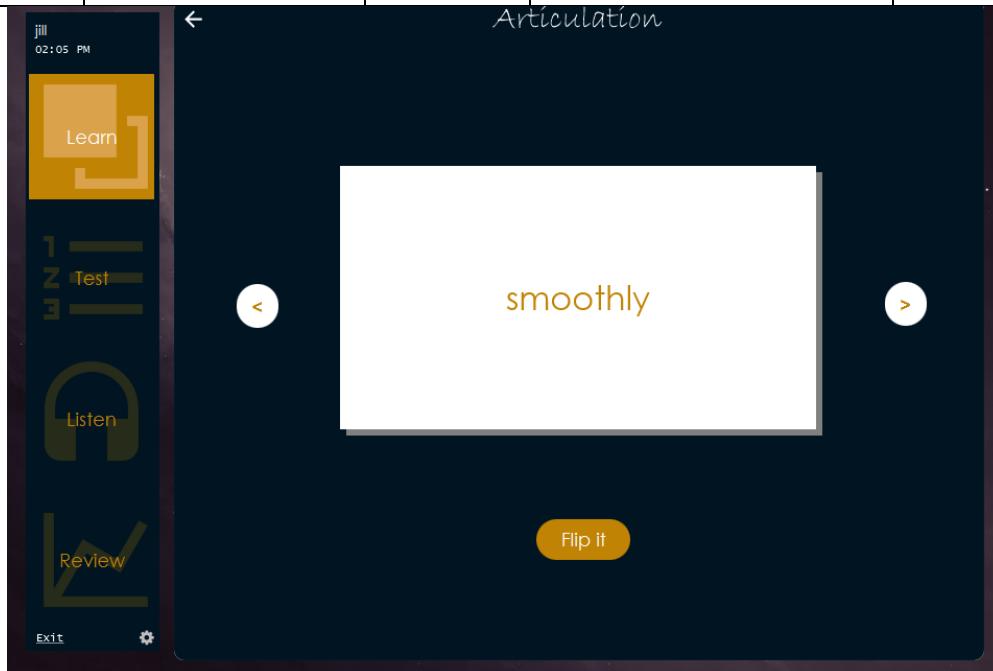


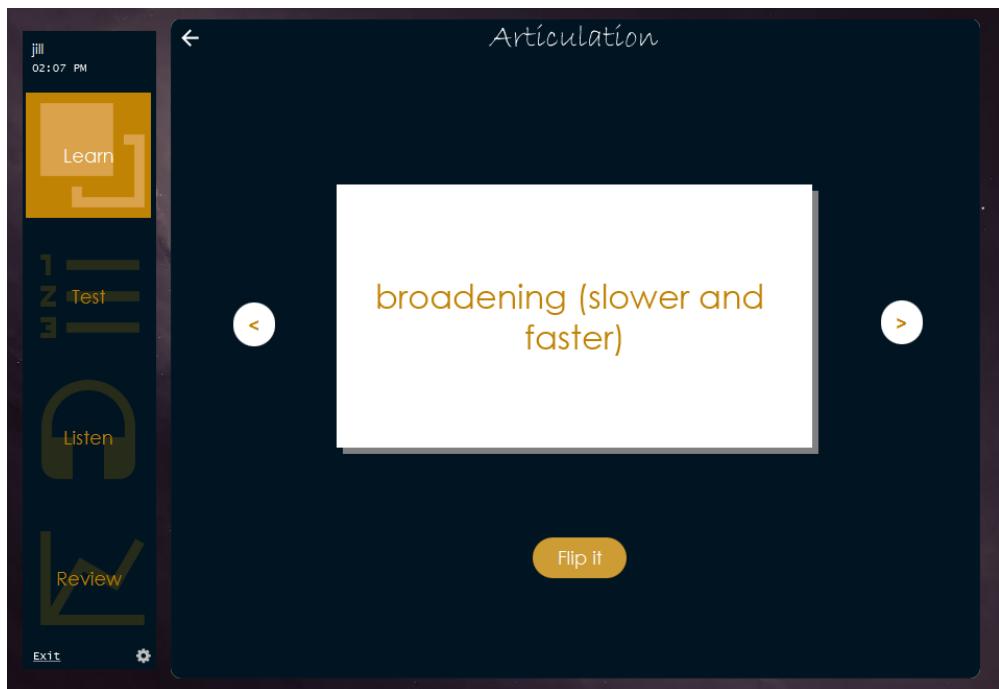
4	Select the Learn activity (2) Navigation Side-Bar (13) Flashcard Set List	Function and Usability	The navigation bar will allow for easy navigation to and from activities. Also, a list of flashcard sets will be displayed when clicked.	Displays a list of flashcards sets to select from	Displays a list of flashcards sets to select from
---	---	------------------------	--	---	---

5	Play the flashcard set: Articulation (13) Flashcard Set List (17) Flashcard Term (18) Flashcard Definition	Function	Playing the flashcards, this test will ensure that the correct flashcards are correctly displayed.	Displays the first flashcard term	Displays the first flashcard term
---	---	----------	--	-----------------------------------	-----------------------------------

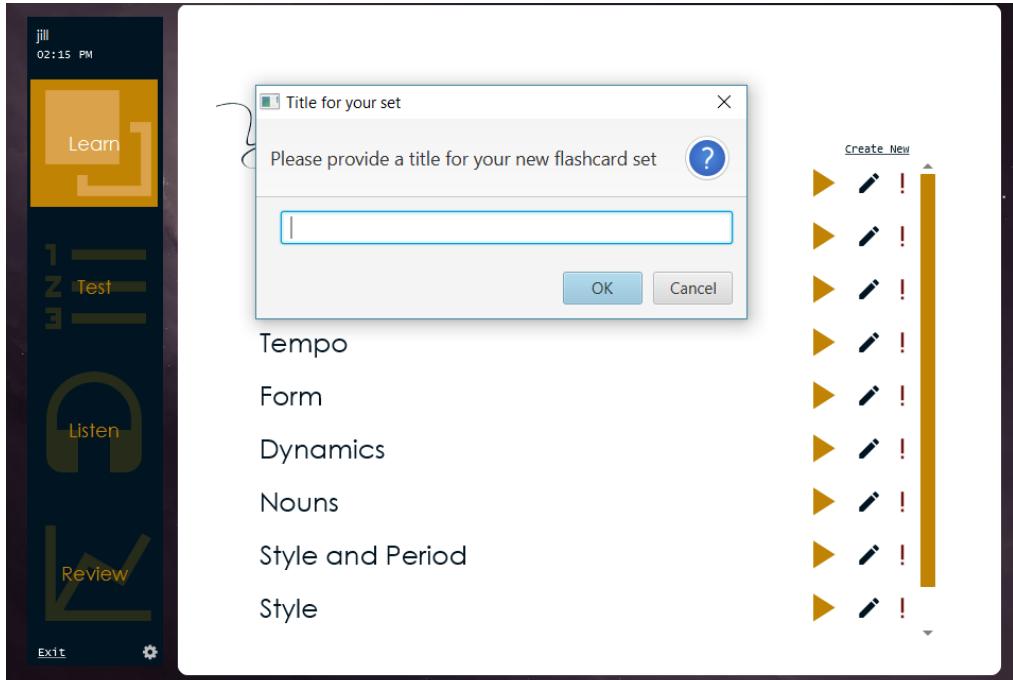


6	Navigate through the set and flip several cards	(17) Flashcard Term (18) Flashcard Definition	Function	Ensures that the user can navigate throughout the set.	Next flashcards are displayed and when flipped, the corresponding definition is revealed	Next flashcards are displayed and when flipped, the corresponding definition is revealed
---	---	--	----------	--	--	--





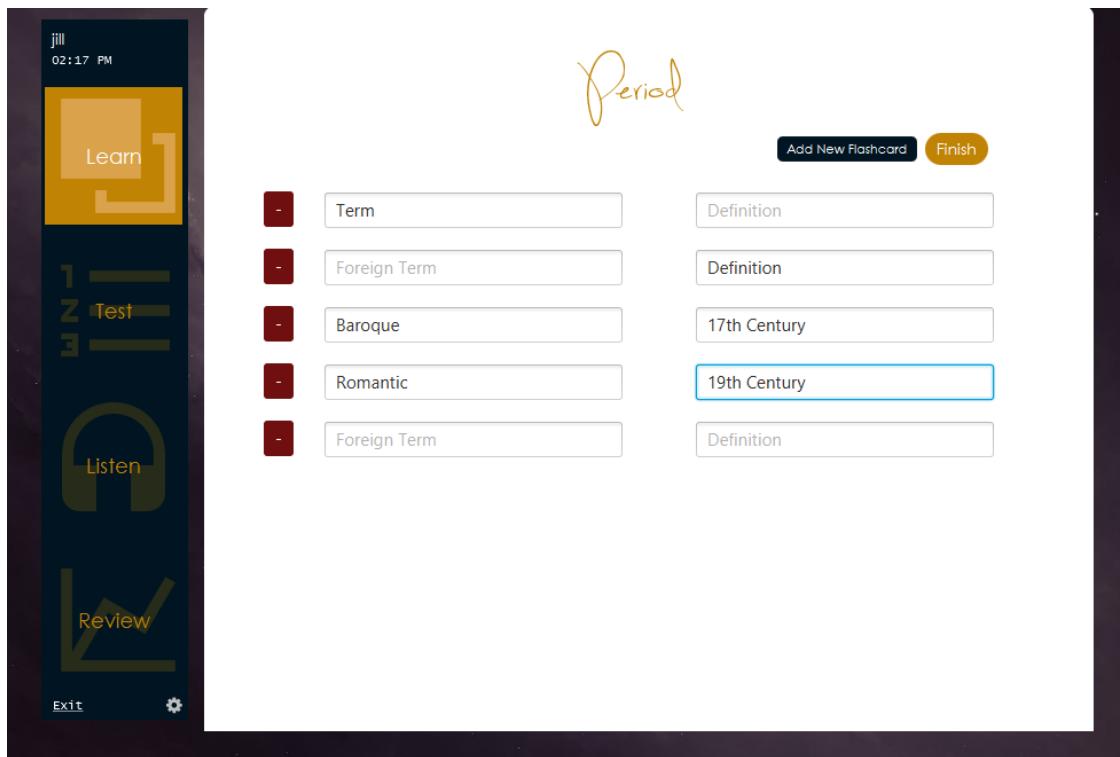
7	Go back and click on create set	(14) Creating New Sets	Function	By redirecting the user to a creation of new sets screen, it partly tests for the success criteria identified	Displays a dialog box asking for the new title	Displays a dialog box asking for the new title
---	---------------------------------	------------------------	----------	---	--	--



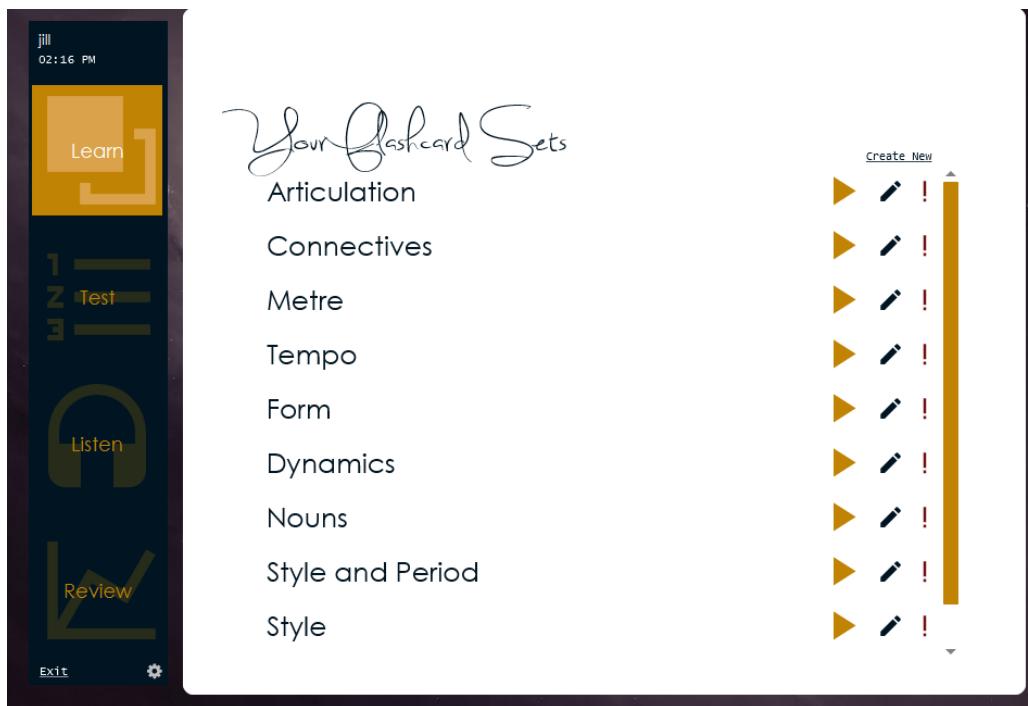
8	Enter “” for the title	(14) Creating New Sets	Validation	A title of blank space should not be allowed	Should return to the main learn screen without changing anything	Returns to t learn screen changing any
---	------------------------	------------------------	------------	--	--	--



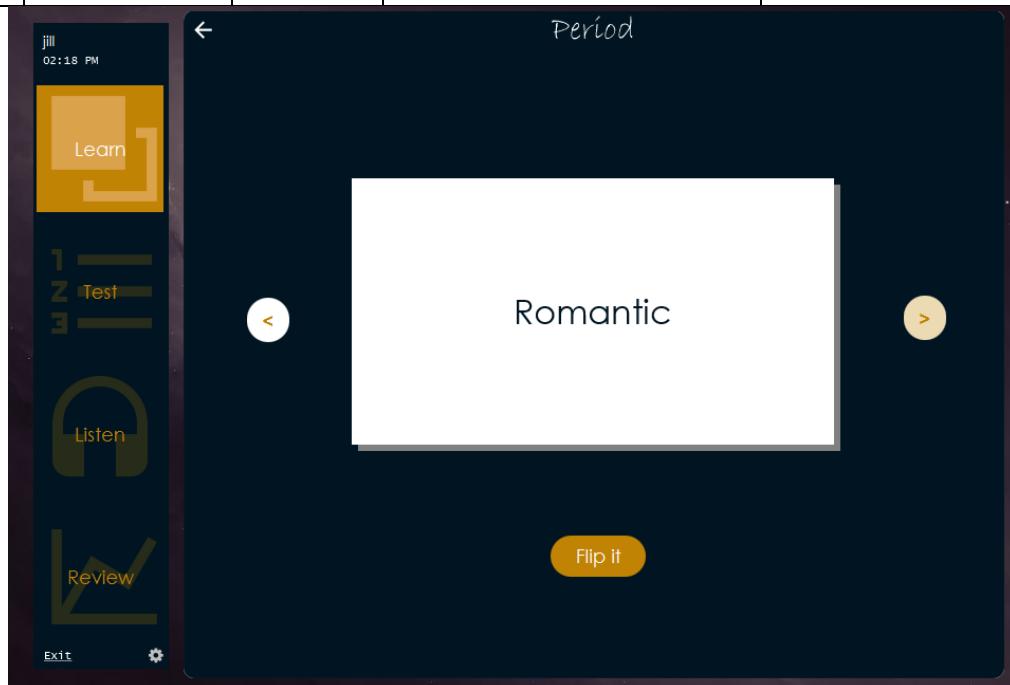
9	Enter “Period” for the title	(14) Creating New Set	Validation	A standard title will be validated and should be accepted	Should open a new create set screen	Opens a new create set screen with title Period
10	Enter “” and “Definition”	(14) Creating New Sets	Validation	The blank space should not be entered as a foreign term and so this invalid data for which validation will check for that	Input will be displayed	Input is displayed
11	Enter “Foreign Term” and “”	(14) Creating New Sets	Validation	Blank space should not be entered neither for the definition, so this invalid data will ensure validation.	Input displayed	Input displayed
12	Enter “Baroque” and “17 th Century”	(14) Creating New Sets	Validation and Function	Valid input, this tests that a standard input should work.	Input displayed	Input displayed
13	Click add new flashcard and enter “Romantic” and “19 th Century”	(14) Creating New Sets	Validation and Function	Valid input, also tests for the add new flashcard is fully functional	Input displayed	Input displayed



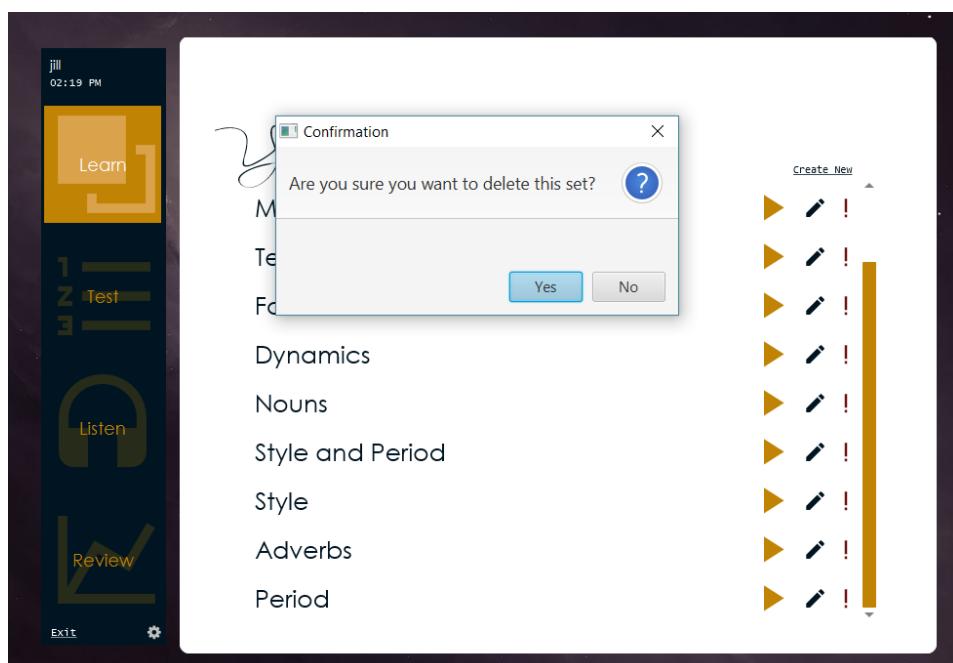
14	Click finish	(14) Creating New Sets (11) Storing data	Function	Finish will create multiple rows in the database representing each flashcard in the set	Set is created, and you are returned to main learn screen	Set is created and you are returned to main learn screen
----	--------------	---	----------	---	---	--



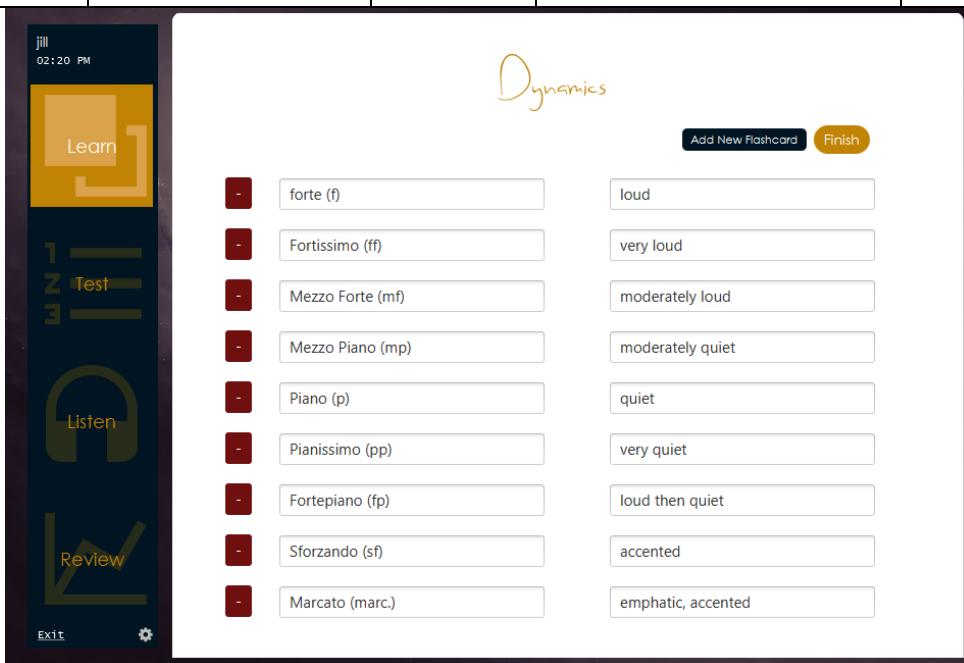
15	Play the new set	(14) Creating New Sets (11) Storing data	Function	If the set has been correctly made, then playing it will work and every term will be included. This tests for function.	Plays the new set as if it were any other set, however there will only contain Baroque and Romantic	Plays the new set as if it were any other set, however there will only contain Baroque and Romantic
----	------------------	---	----------	---	---	---



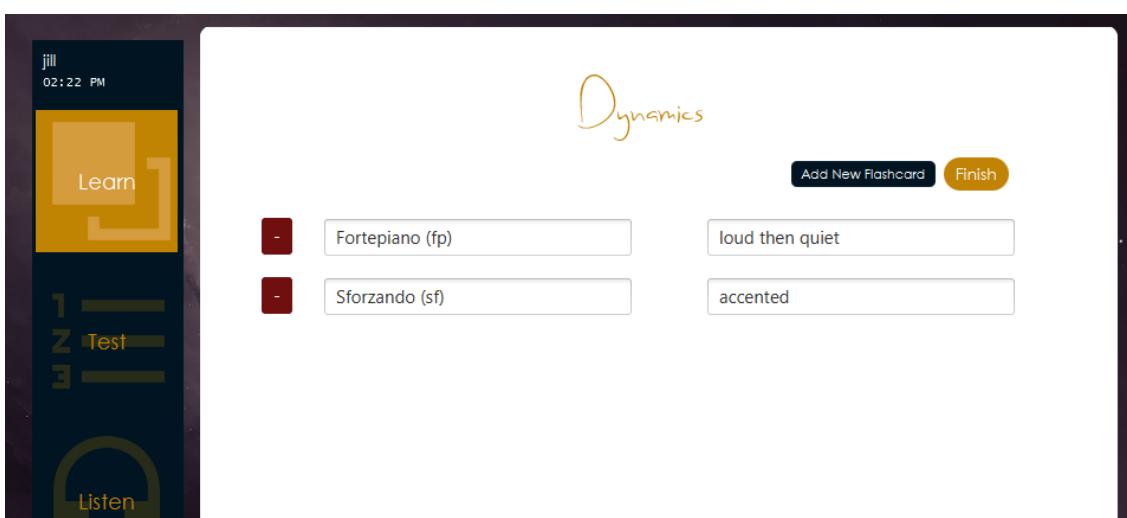
16	Go back and delete the created set	(15) Deleting Sets	Function	This should delete the set and ensures it does not appear again in the interface	Displays confirmation dialog which can be confirmed	Displays confirmation dialog which can be confirmed
----	------------------------------------	--------------------	----------	--	---	---



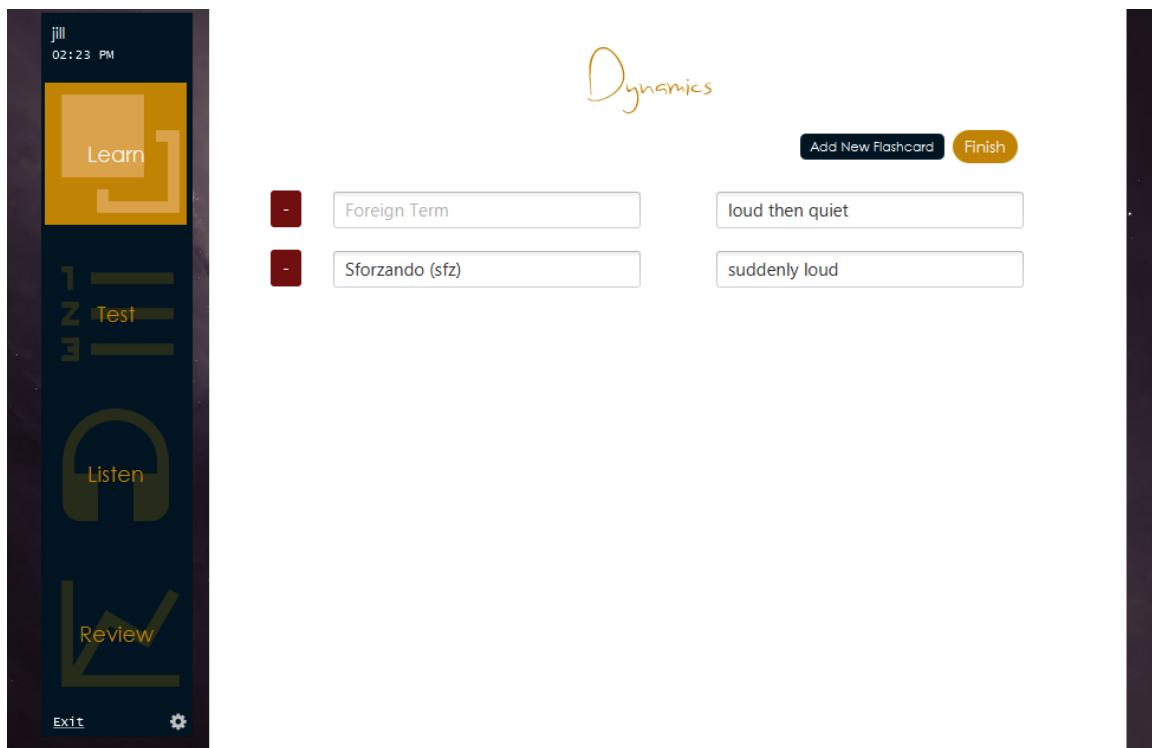
17	Edit set "Dynamics"	(16) Editing Sets (22) Easy to use	Function and Usability	A screen similar to creation of sets will be displayed, it should be easy to use and navigate with.	Displays a screen similar to create new set but contains already existing terms	Displays a screen similar to create new set but contains already existing terms
----	---------------------	---------------------------------------	------------------------	---	---	---



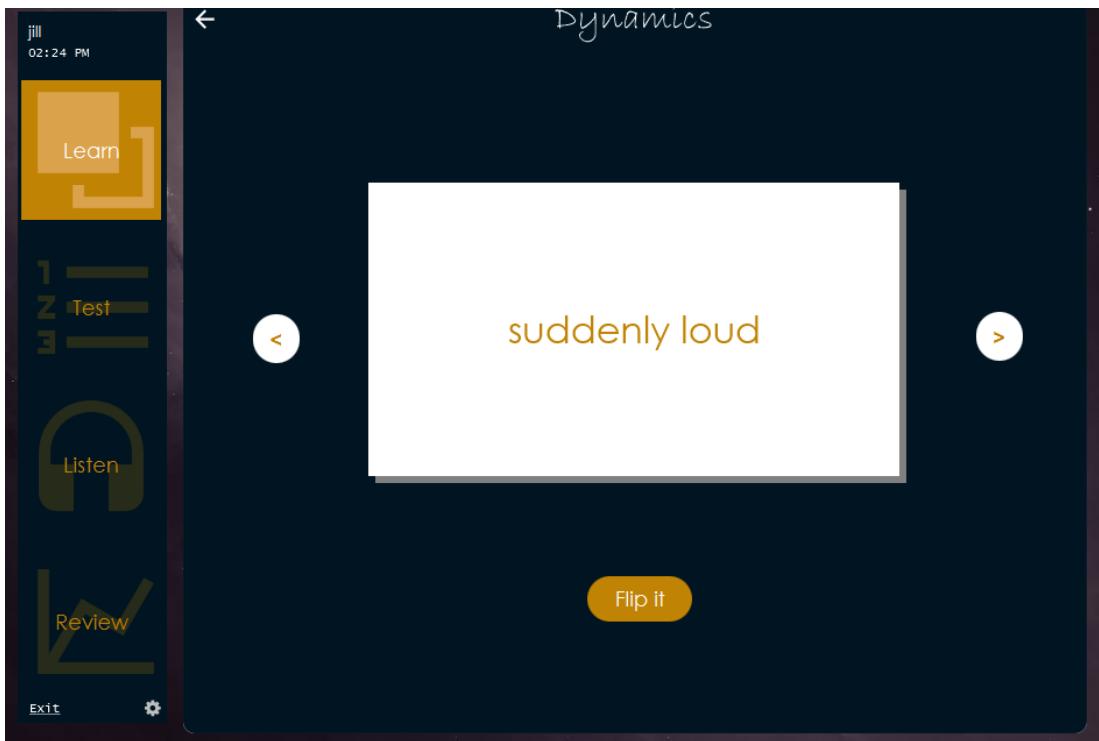
18	Delete all rows except "fortepiano" and "sforzando"	(16) Editing Sets	Function	This tests for changing the size of the set and accurately stores which terms (rows) have been kept or lost.	Respective rows will be removed from user interface	Respective rows removed from user interface
----	---	-------------------	----------	--	---	---



19	Edit the definition of “sforzando” from “accented” to “suddenly loud”	(16) Editing Sets	Function	Tests whether changes to existing rows are recorded	Changes will be displayed	Changes displayed
20	Edit the definition of “fortepiano” to blank space “”	(16) Editing Sets (21) Robust and Error-Free	Validation and Robustness	With this, the test will make sure that invalid data cannot be entered via editing sets	Changes displayed	Changes displayed



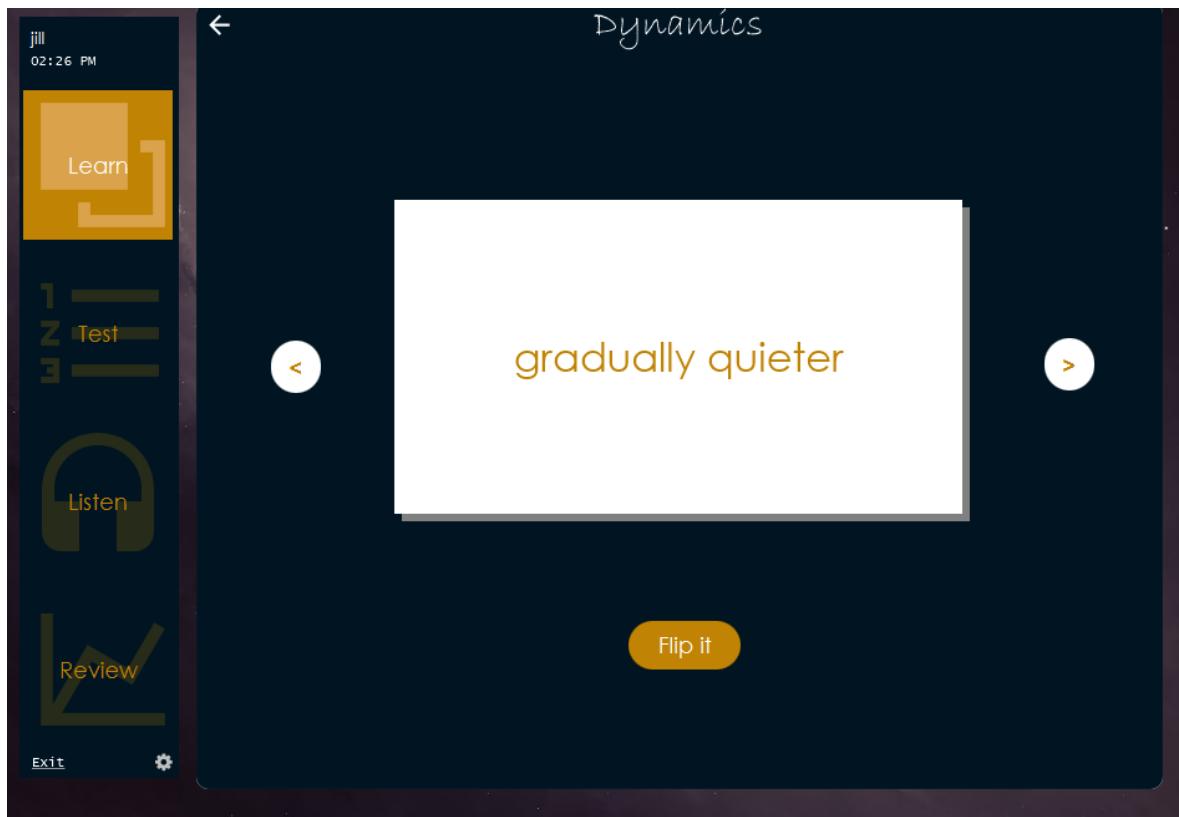
21	Click finish	(16) Editing Sets (11) Storing the data	Function	Will allow the changes to be made for later tests.	Data will be stored	Data is stored
22	Observe changes by playing the flashcard set	(16) Editing Sets (21) Robust and Error-Free	Validation and Robustness	Confirm that the data has been stored correctly and then updated data has been retrieved	Changes displayed	Changes displayed



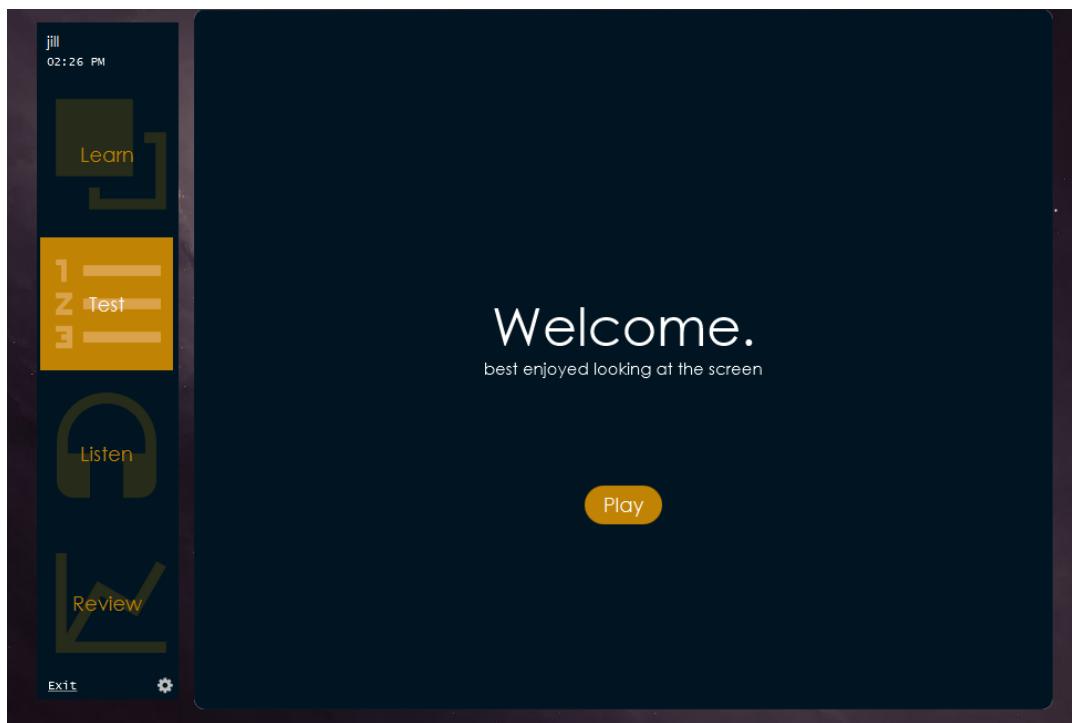
23	Add a new row “diminuendo”, “gradually quieter”	(16) Editing Sets	Function	Tests whether increasing the size will be recorded	Changes displayed on UI	Changes displayed
----	---	-------------------	----------	--	-------------------------	-------------------

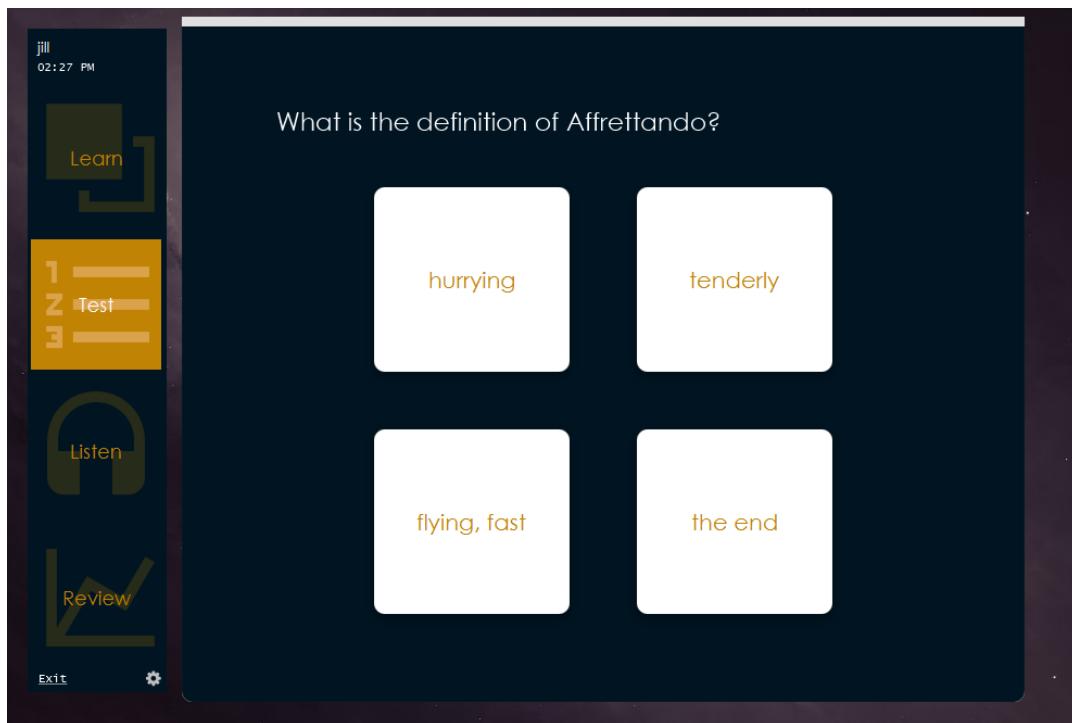


24	Click finish	(16) Editing Sets (11) Storing the data	Function	Will allow the changes to be made for later tests.	Data will be stored	Data is stored
25	Observe changes by playing the flashcard set	(17) Flashcard Term (18) Flashcard Definition (16) Editing Sets (21) Robust and Error-Free	Validation and Robustness	Confirm that the data has been stored correctly and then updated data has been retrieved	Changes displayed	Changes displayed

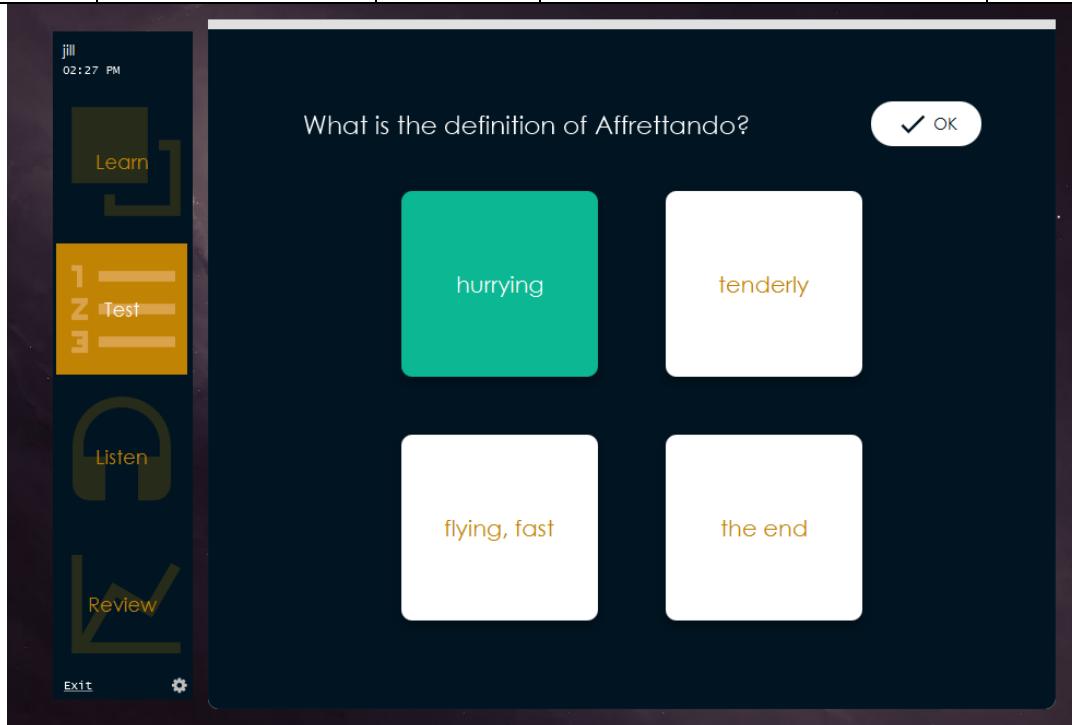


26	Click on the Test activity and start the quiz	(2) Navigation Side-bar (6) Question Display (7) Multiple Choice Answers	Usability	Ensures that the quiz is correctly structured and displayed.	Welcome screen displayed and then you are redirected to the first question	Welcome screen displayed and then redirected to first question
----	---	--	-----------	--	--	--

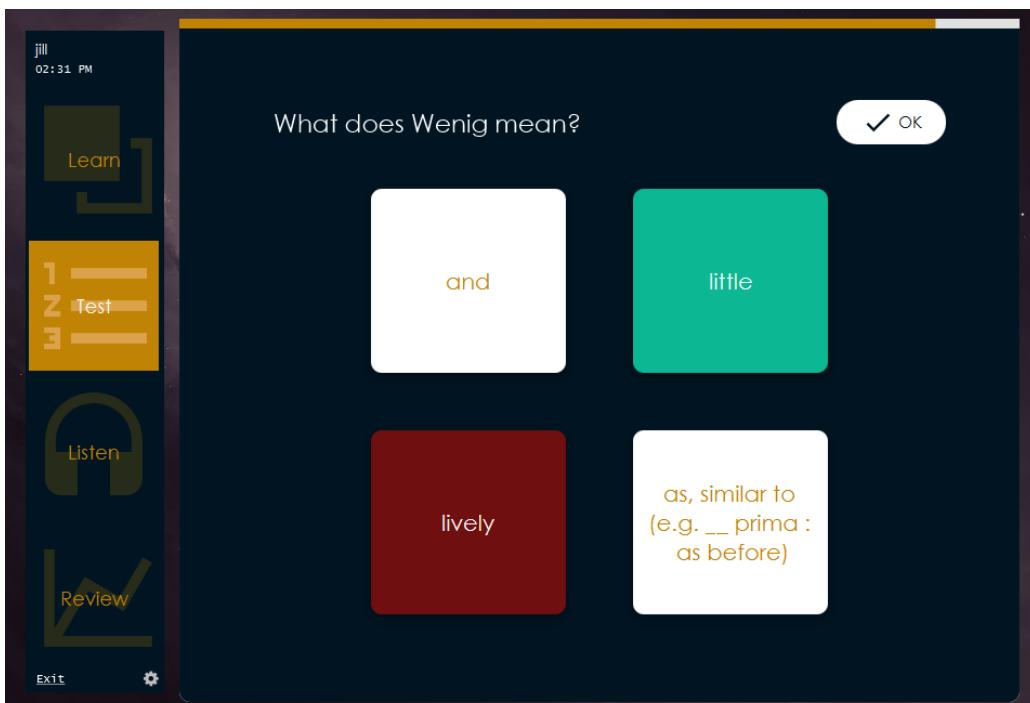
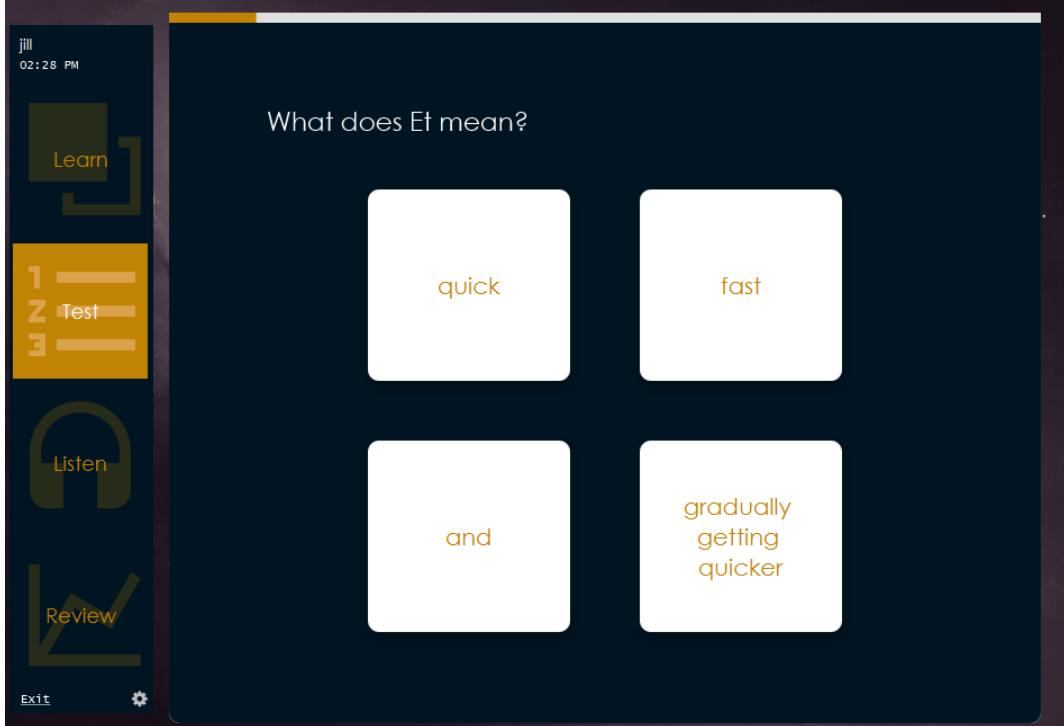




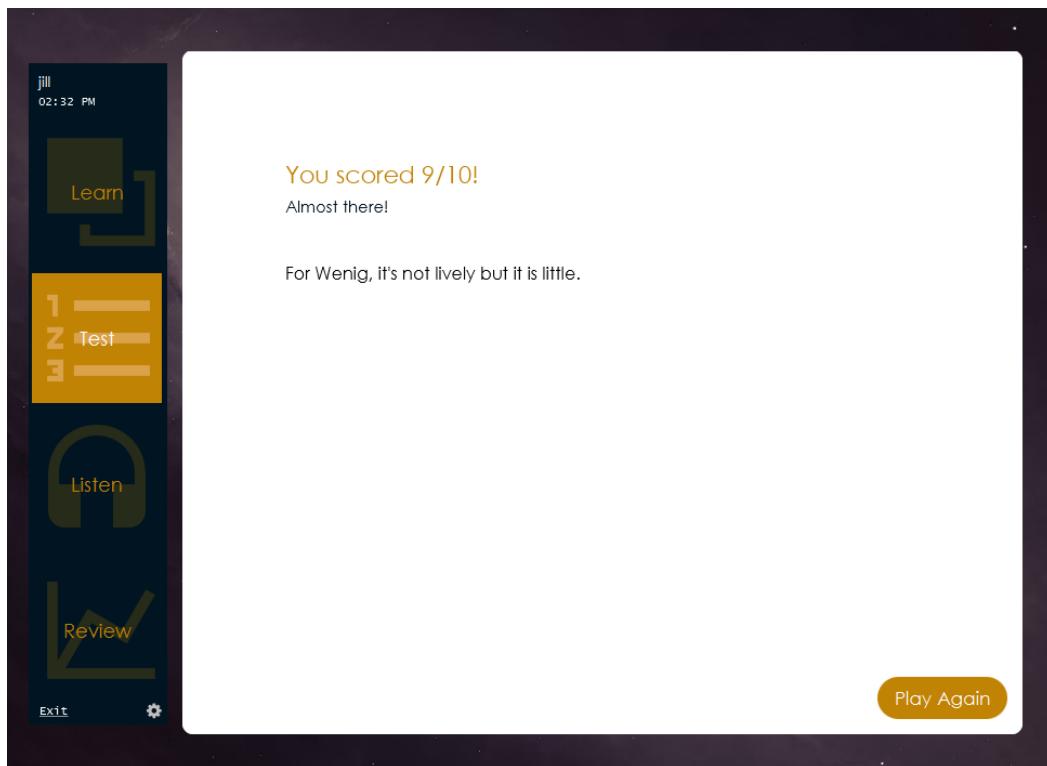
27	Answer a question	(9) Question Feedback (22) Easy to use	Function and Usability	The feedback should be accurately handled and displayed, this also demonstrates if it is easy to interpret for the user.	Correct feedback displayed	Correct feedback displayed
----	-------------------	---	------------------------	--	----------------------------	----------------------------



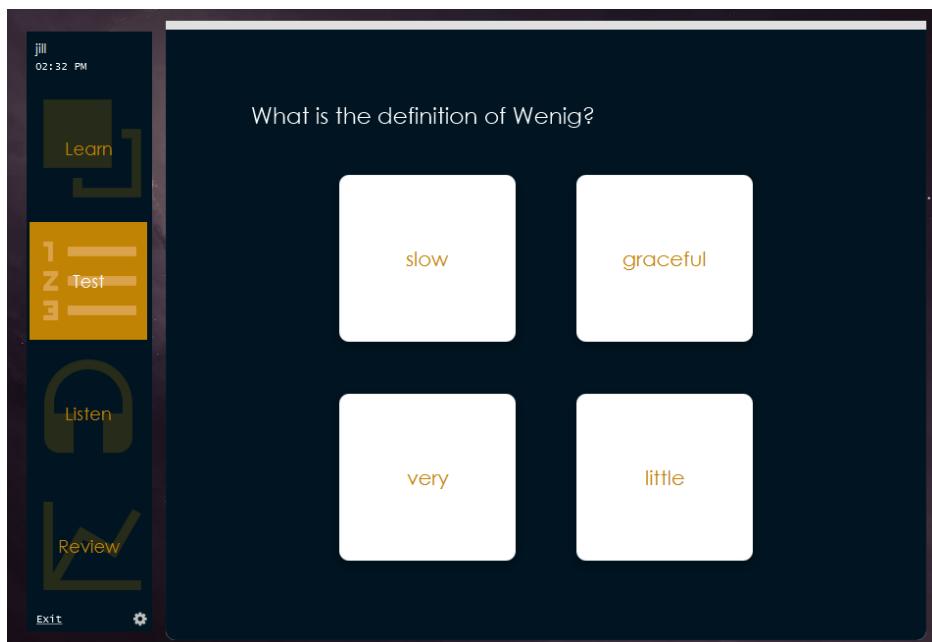
28	Click OK and complete the rest of the quiz	(8) Progress in Quiz	Usability	The quiz will display a bar determining how far the user is into it.	Shows next question, whilst continuing to show feedback as you progress through the quiz	Shows next question, whilst continuing to show feedback as you progress through the quiz
----	--	----------------------	-----------	--	--	--



29	Examine results screen	(10) Results Feedback (11) Storing the Data	Usability and Function	A summary of all the feedback will be listed here, easier to improve on. Also, ensures that the correct data is being displayed (and stored).	Correct score displayed with appropriately responded feedback	Corre... displa... appro... feedb...
----	------------------------	--	------------------------	---	---	---



30	Examine the first question after playing again	(20) Motivating (11) Storing the Data	Function	A previously answered wrong question will be displayed first which will enhance the user learning experience	A wrongly answered question displayed	A wrongly answered question displayed
----	--	--	----------	--	---------------------------------------	---------------------------------------



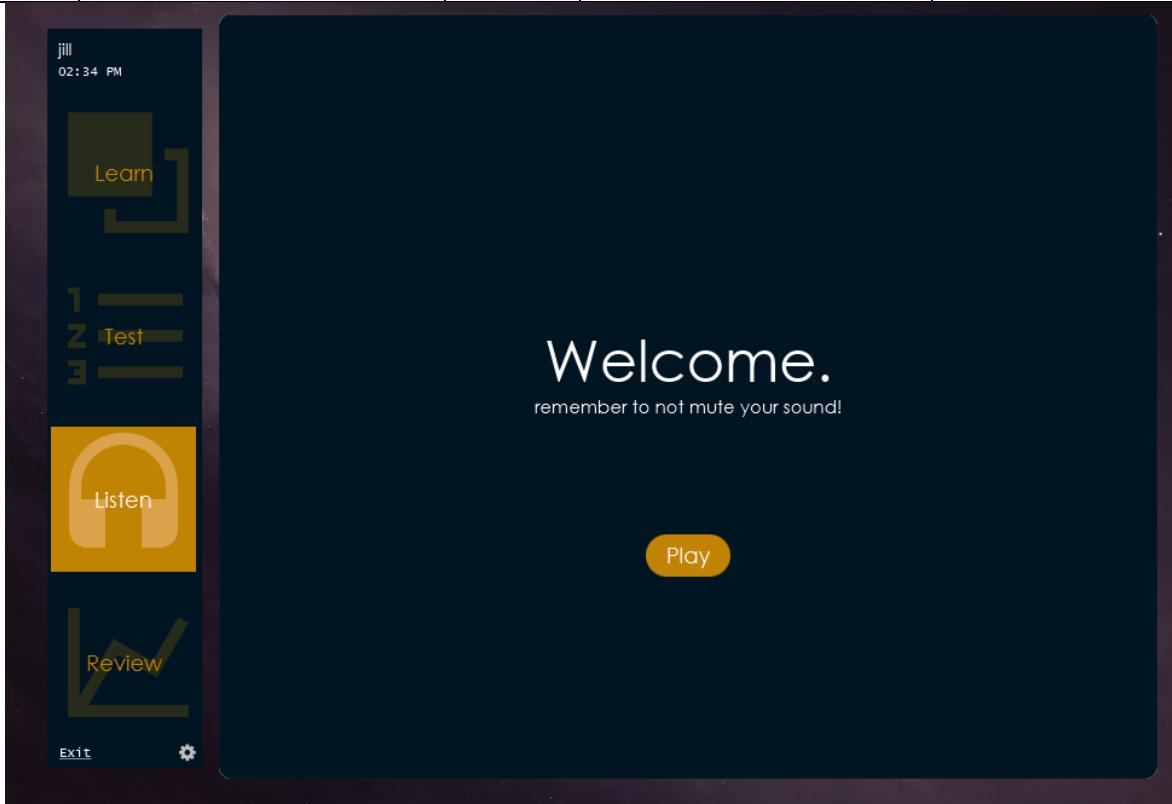
31	Click on Review activity	(2) Navigation Side-bar (11) Storing the Data	Usability and Function	If data was correctly stored, there should display a graph with a plot indicating the score of the previously taken test. This plot	Correct data point	Correct data point displayed on
----	--------------------------	--	------------------------	---	--------------------	---------------------------------

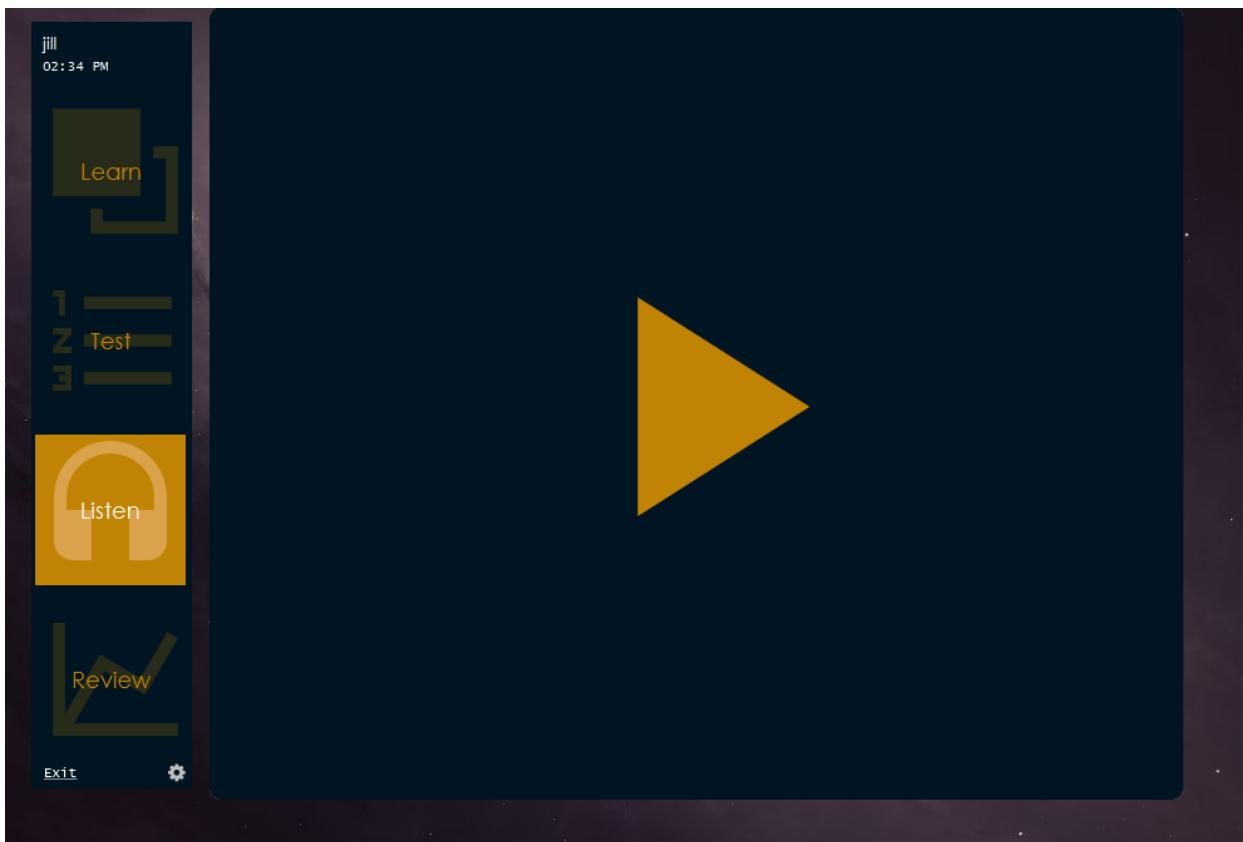
		(12) Graphical Progress Display (20) Motivating		should appear on all graphs. A test such as this is justified as it allows the user to reflect on their progress and motivate the user	displayed on all three graphs	all three graphs
--	--	--	--	--	-------------------------------	------------------





32	Click on Listen activity and play the music	(2) Navigation Side-bar (3) Audio playback (4) Play/Pause Button (6) Question Display (7) Multiple Choice Answers	Function and Usability	The play button should be fully functional and should be easy to use. Also, this test ensures that the quiz is displayed correctly.	Welcome screen displayed, then a play button, then the quiz is shown.	Welcome screen displayed, then a play button, then the quiz is shown.
----	---	---	------------------------	---	---	---



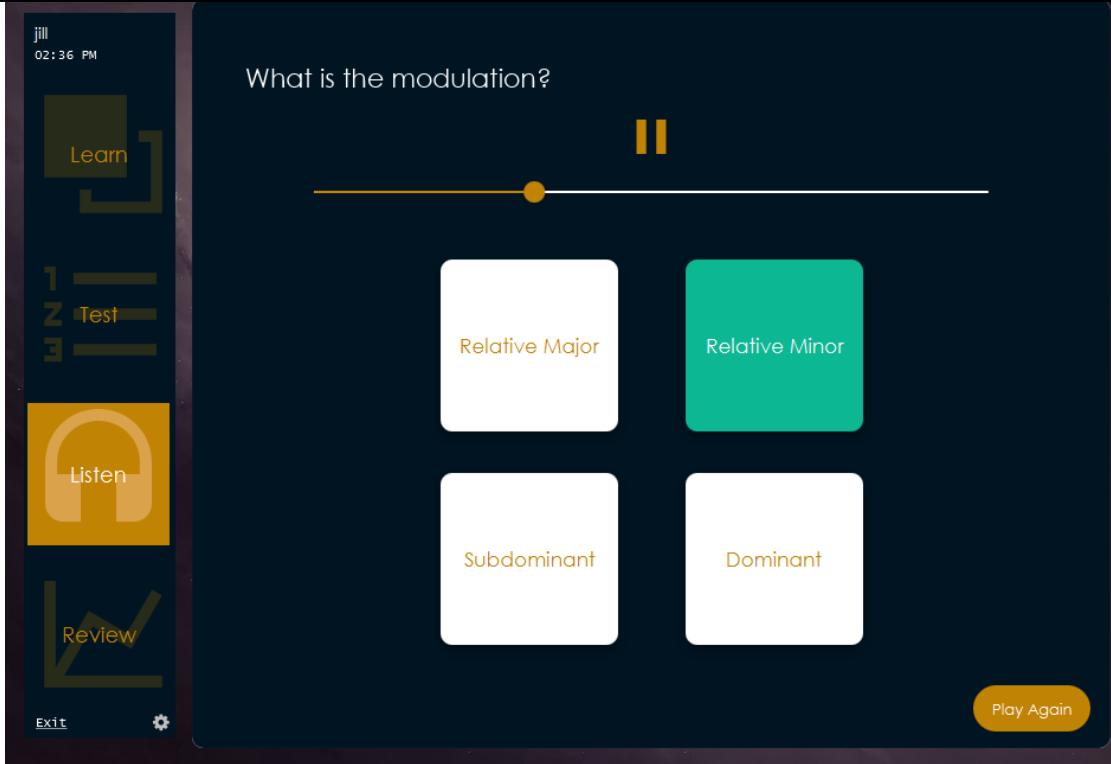


A screenshot of a mobile application interface. On the left is a vertical sidebar with a dark background, identical to the one in the first screenshot. On the right, the main area displays a question: "What is the modulation?". Below the question is a horizontal line. To the right of the line are four white rectangular boxes arranged in a 2x2 grid. The top-left box contains the text "Relative Major", the top-right box contains "Relative Minor", the bottom-left box contains "Subdominant", and the bottom-right box contains "Dominant".

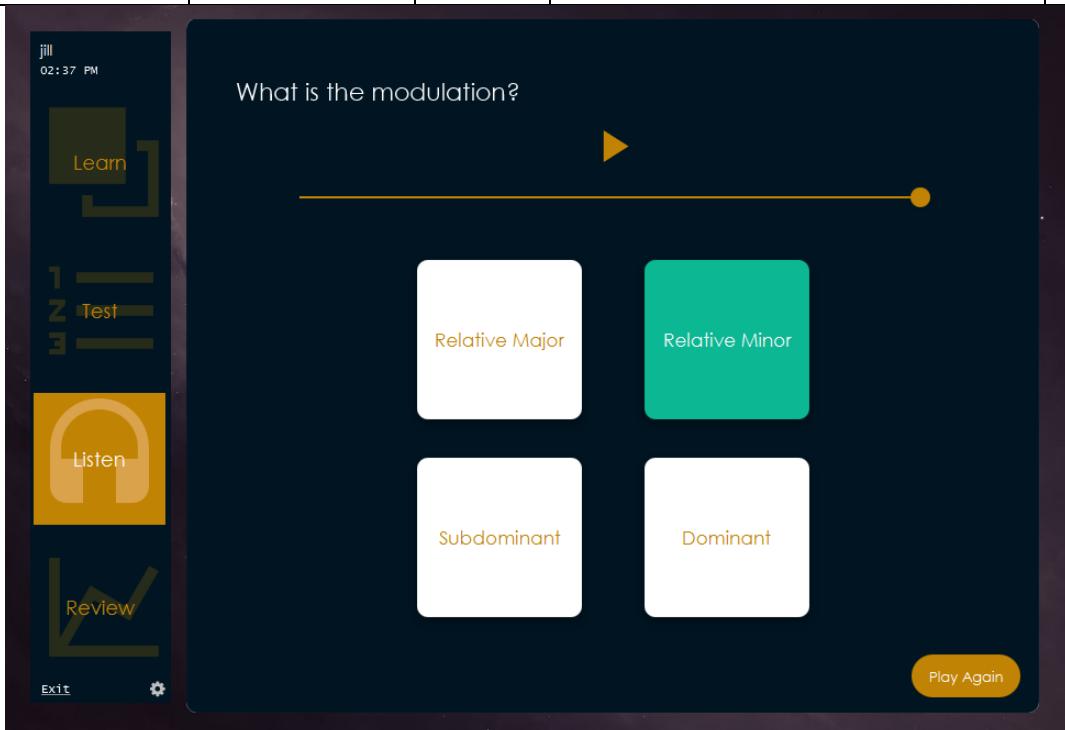
33	Answer the question (repeated for a correctly answered and wrongly answered)	(9) Question Feedback (22) Easy to use	Function and Usability	The correct answer will be displayed and compared to the inputted answer, this should be easy to interpret by the user.	Correct feedback displayed	Correct feedback displayed
----	--	---	------------------------	---	----------------------------	----------------------------

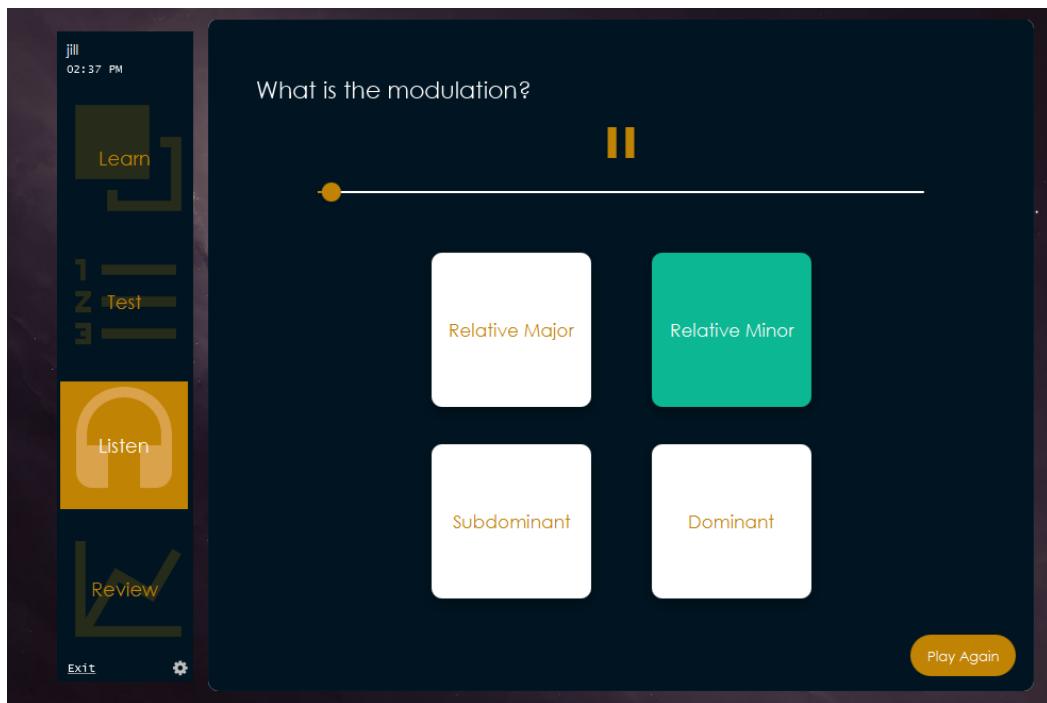
The image displays two identical screenshots of a mobile application interface, likely from an iPhone, showing a music theory quiz screen. The top-left corner shows the user's name "jill" and the time "02:37 PM". The top-right corner has a yellow play button icon. The main content area is a dark blue box containing a question: "What is the modulation?". Below the question is a horizontal timeline with a yellow playhead at the end. Four options are presented in a 2x2 grid: "Relative Major" (orange text on white background), "Relative Minor" (white text on green background), "Subdominant" (orange text on white background), and "Dominant" (orange text on white background). In the bottom right corner of the main box is a yellow "Play Again" button. To the left of the main box is a vertical sidebar with four tabs: "Learn" (selected), "Test" (disabled), "Listen" (selected), and "Review" (disabled). At the bottom of the sidebar are "Exit" and "Settings" buttons.

34	Drag the slider to a previous point and play	(5) Seek Slider (4) Play/Pause Button (22) Easy to use	Function and Usability	The slider should be an easy way to change the position of the audio, and the test will demonstrate that.	Plays music at respective position	Plays music at respective position
----	--	--	------------------------	---	------------------------------------	------------------------------------



35	Pause the music just before it finishes and play it again	(4) Play/Pause Button	Function	Ensures that the pause feature works and that the play button will start again at where previous left off.	Plays back at the start	Plays back at the start
----	---	-----------------------	----------	--	-------------------------	-------------------------





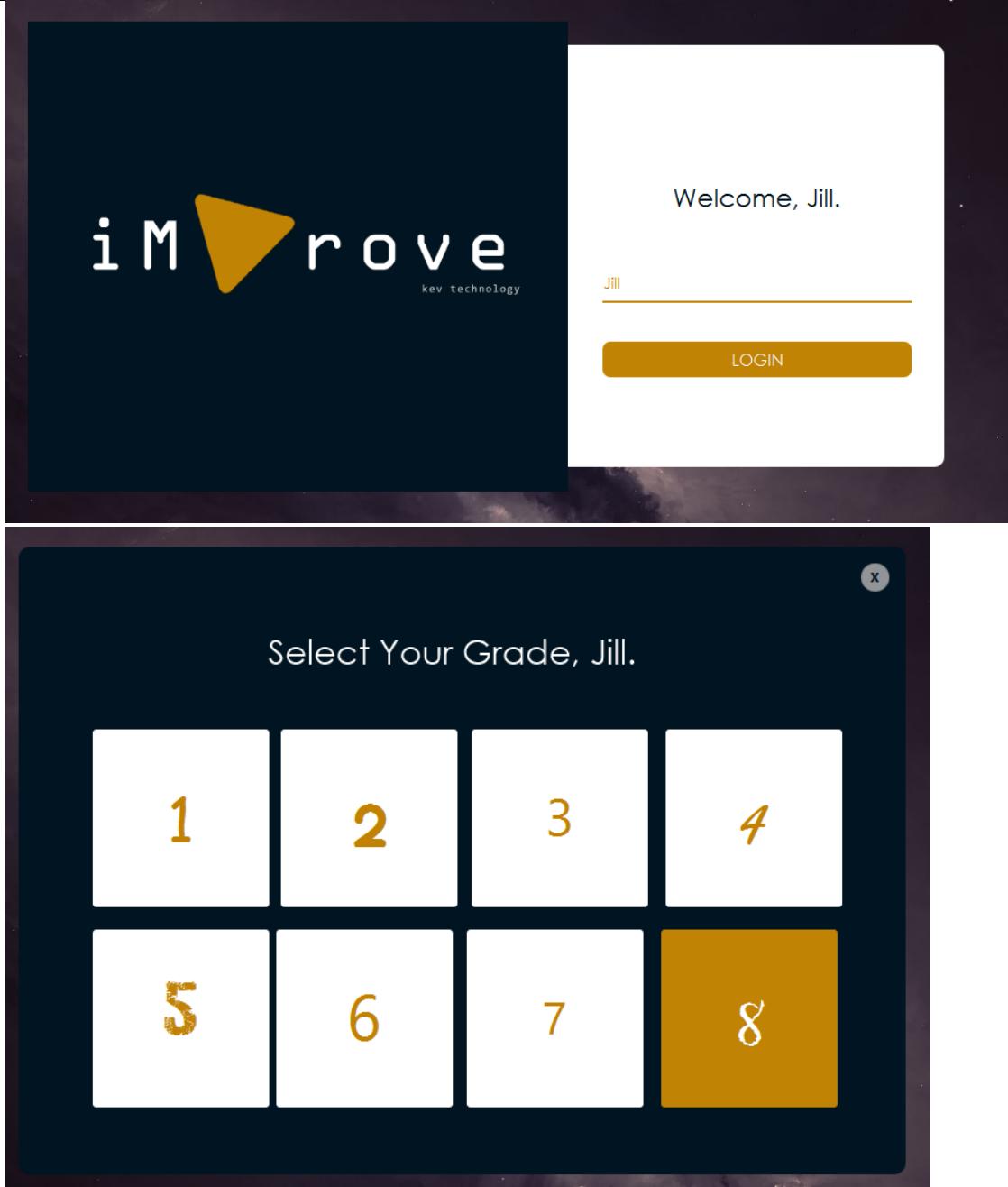
36	Press Exit	(19) Close the Application	Function and Robustness	Confirms if the application can be safely exited without any resource leaks or database corruptions.	Application closes	Application closes
----	------------	----------------------------	-------------------------	--	--------------------	--------------------

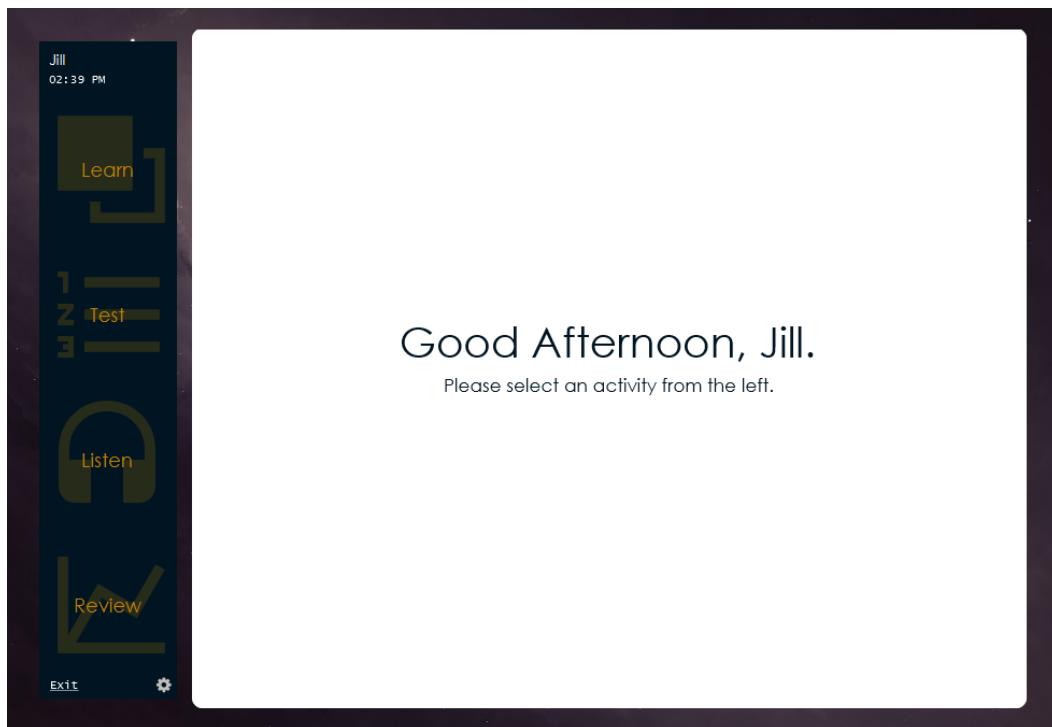


Test 2

To more rigorously test the system, an extra test will ensure a stable, robust final product

Test ID	Test Data	Success Criteria	Aspect	Justification	Expected Outcome	Actual Outcome
37	Grade Eight clicked with “Jill” as name	(21) Robust and Error-Free (1) Grade Separation/Selection	Function and Robustness	Should be accepted although a different name, it is still valid data.	Logins as normal	Logins as normal

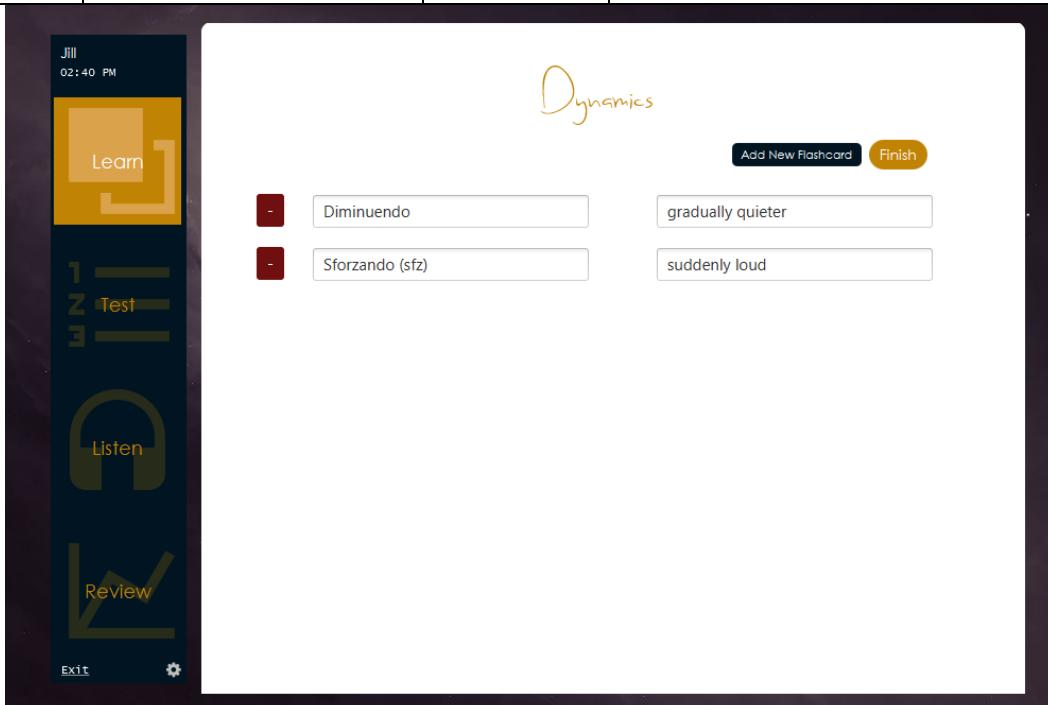




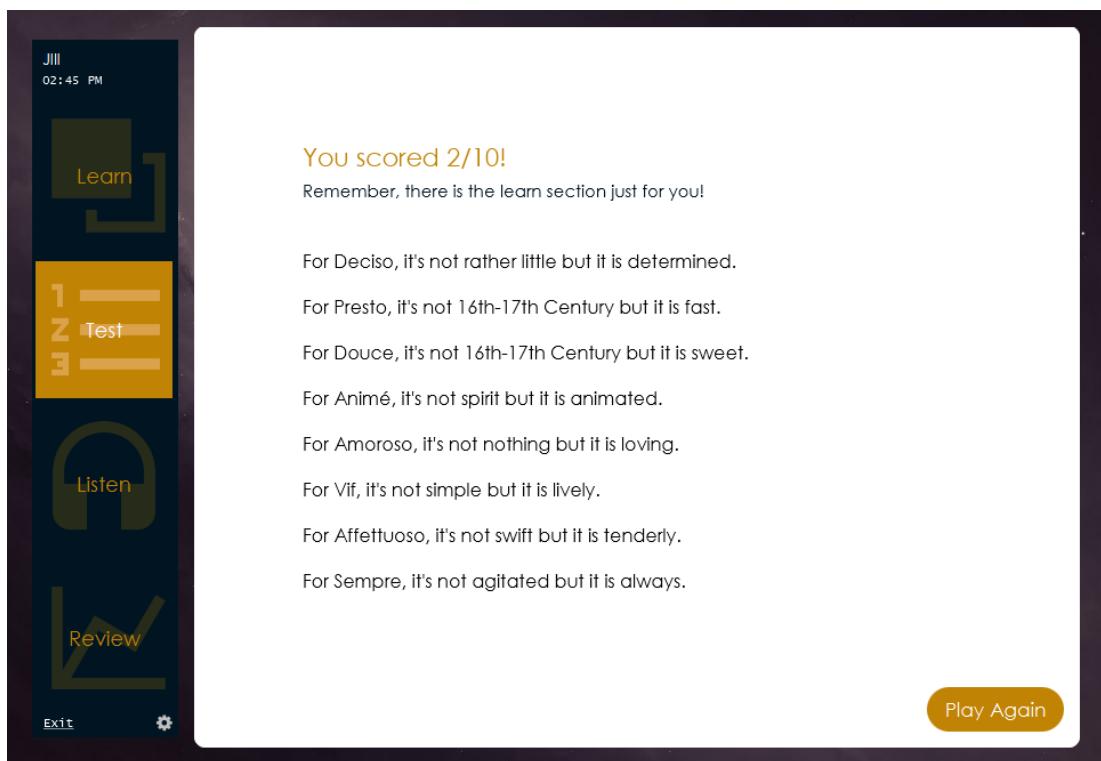
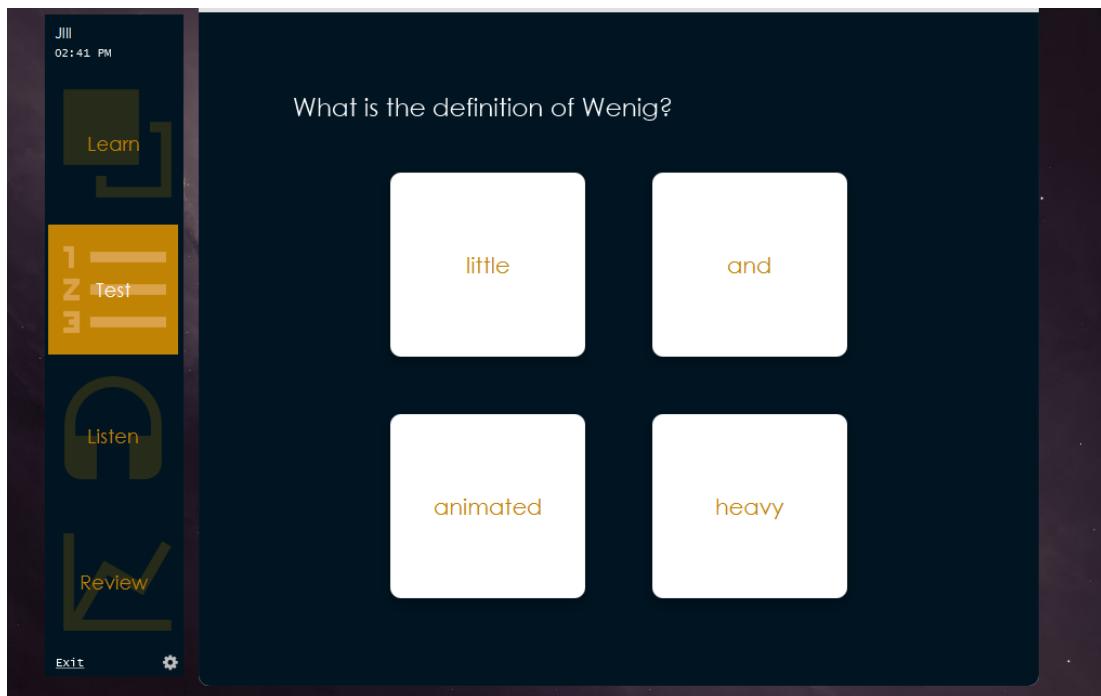
38	Click on Learn and examine the list of flashcard sets	(13) Flashcard Set List (21) Robust and Error-Free	Function and Robustness	Starting a new application, the custom set with title “Period” should be still deleted	Loads a list of flashcards sets without Period set	Loads a list of flashcards sets without Period set
----	---	---	-------------------------	--	--	--

Your Flashcard Sets	
Articulation	Create New
Connectives	Create New
Metre	Create New
Tempo	Create New
Form	Create New
Dynamics	Create New
Nouns	Create New
Style	Create New
Adverbs	Create New

39	Play flashcard set of “Dynamics” and examine terms	(16) Editing Set (21) Robust and Error-Free (11) Storing the data (17) Flashcard Term (18) Flashcard Definition	Function and Robustness	This test will check whether the previous test’s changes have still been persistently stored. So, there should only be 2 rows, “sforzando” with “suddenly loud”, “fortepiano” should be deleted because of blank space, and “diminuendo” with “gradually quieter”.	Displays only Dimuendo and Sforzando rows	Displays Dimuendo and Sforzando rows
----	--	---	-------------------------	--	---	--------------------------------------

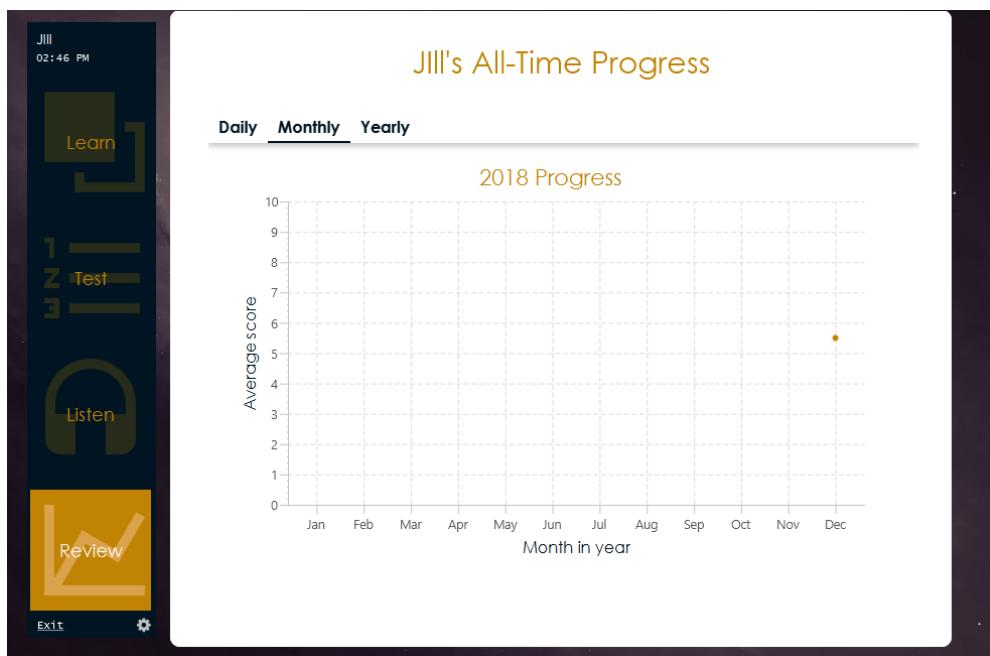
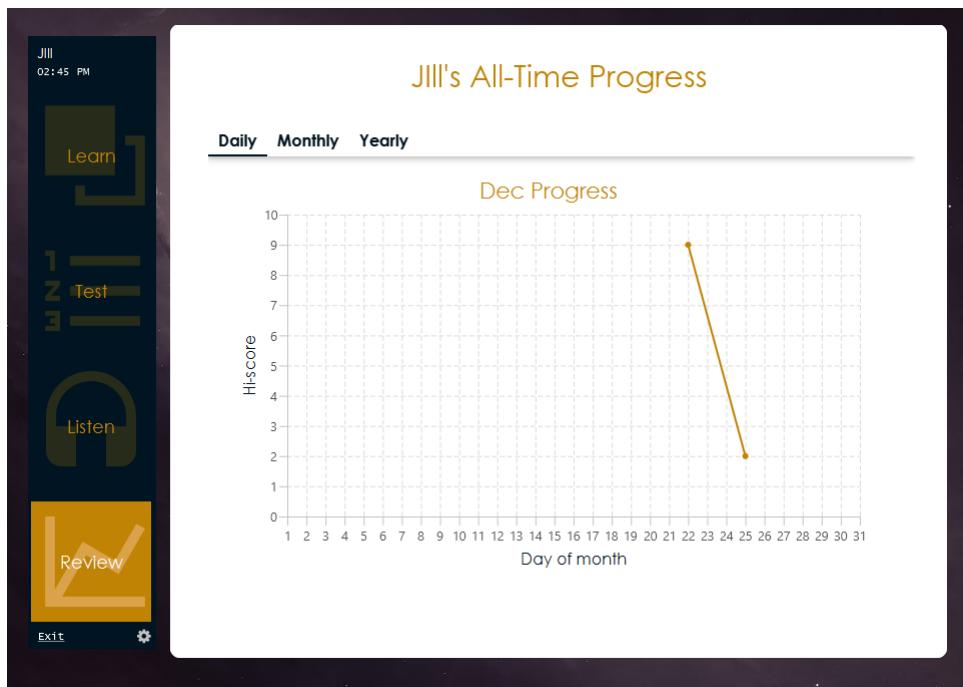


40	Complete a test	(6) Question Display (7) Multiple Choice Answers (8) Progress in Quiz (9) Question Feedback (10) Results Feedback (11) Storing the data (20) Motivating (22) Easy to use	Function and Usability	Taking the test again will ensure it is motivating and easy to use, also that the functions of each part of the test is functional. However, the importance is after the test and determining whether the results is correctly stored.	Another test is performed with previously answered wrongly terms displayed first	Another test is performed with previously answered wrongly terms displayed first
----	-----------------	---	------------------------	--	--	--



41	Click on Review and examine the graphs	(11) Storing the data (12) Graphical Progress Display (21) Robust and Error-Free	Function and Robustness	Even with a new start of application, the data points should not be wiped out and so in the daily graph, there should be a line joining the two data points from the two tests. On the monthly and yearly graph, it should display	A line joining the two data points on the daily graph, an average of the scores of the month on monthly graph,	A line joining the two data points on the daily graph, an average of the scores of the month on monthly graph, and similarly on
----	--	--	-------------------------	--	--	---

				the average of the two high-score values. and similarly on yearly. $([2+9]/2=5.5)$	yearly. $([2+9]/2=5.5)$
--	--	--	--	--	----------------------------





42	Close the Application	(19) Close the Application (21) Robust and Error-Free	Function and Robustness	Ensures that the system exists without any error messages.	Closes the system	Closes the system
----	-----------------------	--	-------------------------	--	-------------------	-------------------

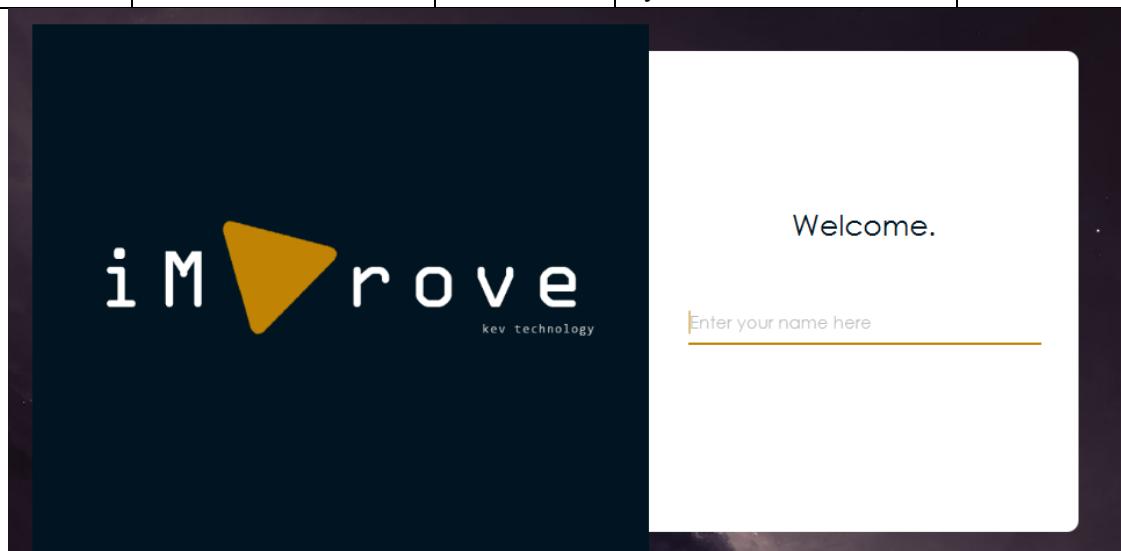


Test 3

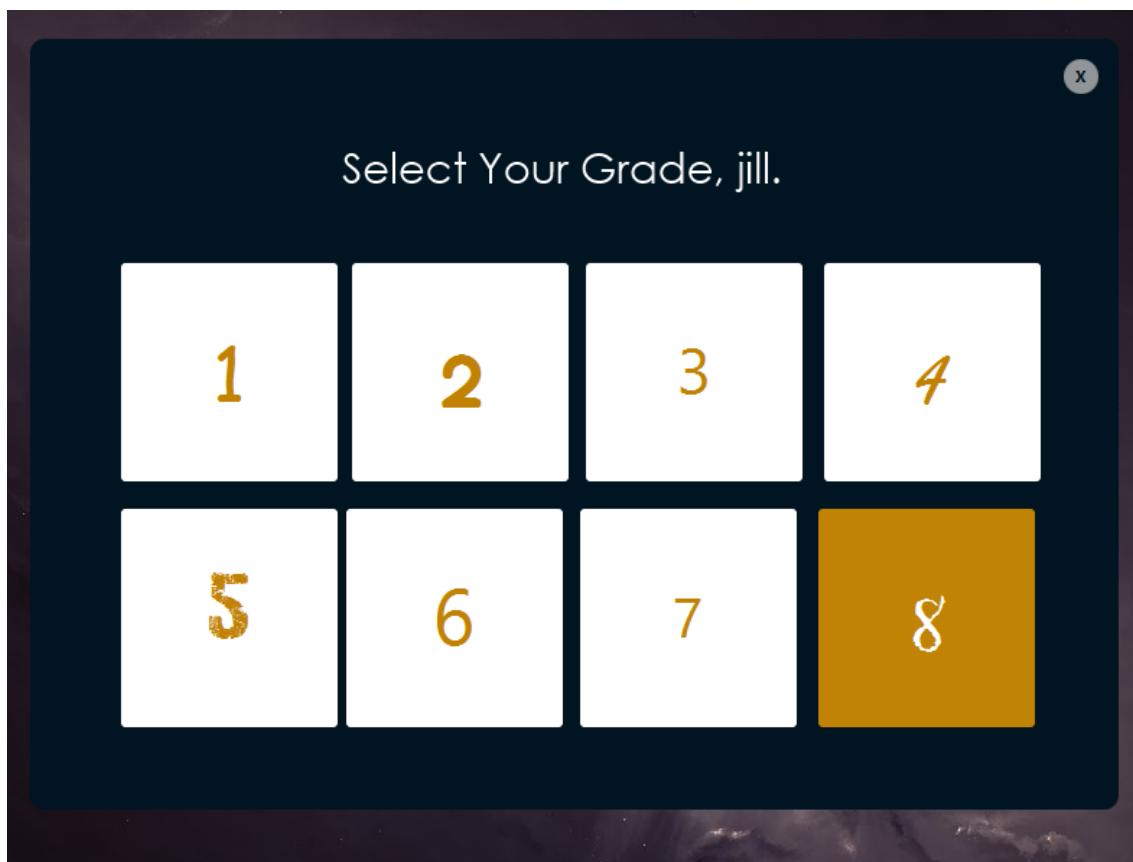
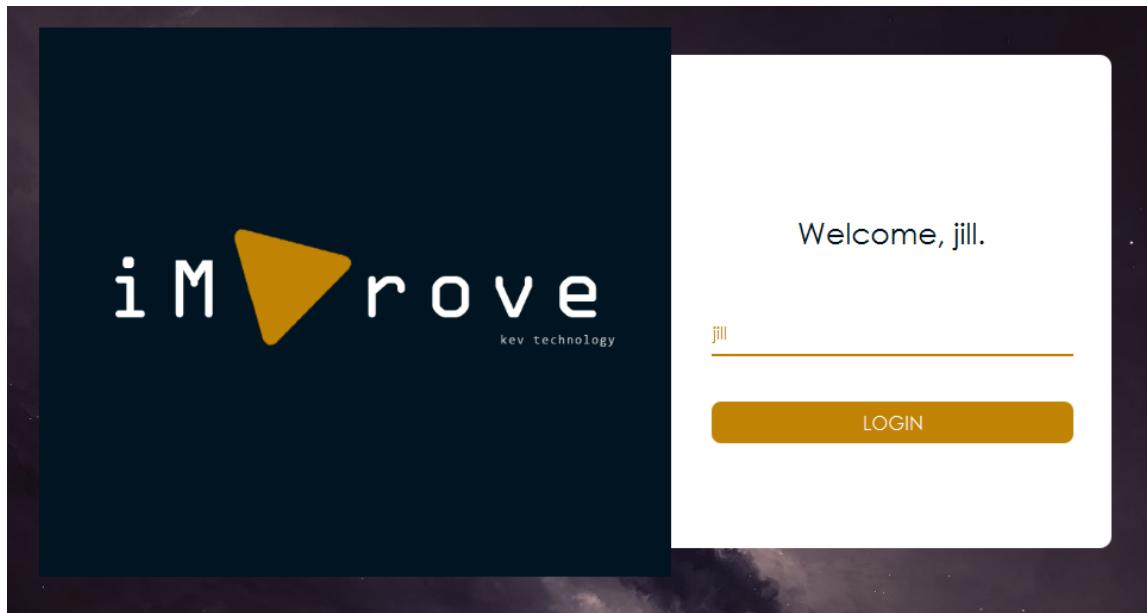
Although we have rigorously tested the persistency of our system, we are ignoring a very crucial part of the program which affects the rest of our system: the grade separation.

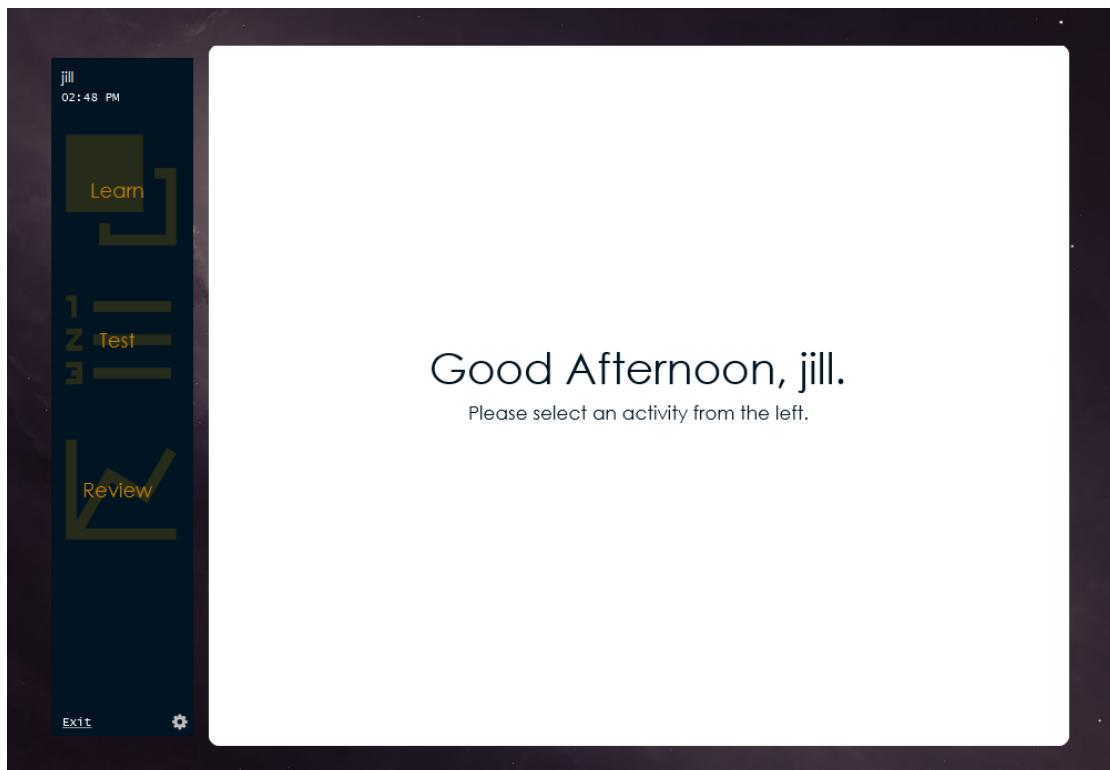
First of all, we will change the system clock of the computer to a different year and different date (e.g. February 15th 2032) to fully exploit the review feature including its yearly graph.

Test ID	Test Data	Success Criteria	Aspect	Justification	Expected Outcome	Actual Outcome
43	Grade One clicked with “” as name	(21) Robust and Error-Free (1) Grade Selection/Separation	Function and Robustness	Should also not be accepted even for a different grade, invalid data and so will not progress through the system.	Cannot login	Cannot login



44	Grade One clicked with “jill” as name	(21) Robust and Error-Free (1) Grade Selection/Separation	Function and Robustness	Standard input although different name, should still be accepted and directed into the welcome screen.	Logs in as normal as grade one student	Logs in as normal as grade one student
----	---------------------------------------	--	-------------------------	--	--	--

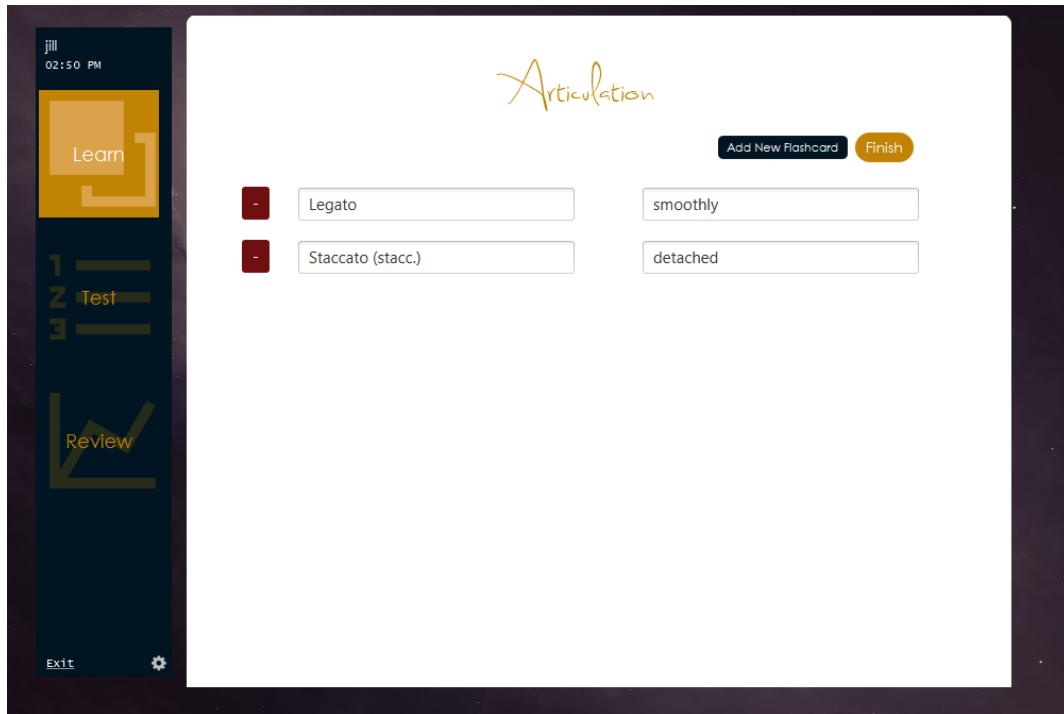




45	Click on Learn Activity and examine the list of flashcard sets	(21) Robust and Error-Free (2) Navigation Side-Bar	Function and Robustness	A diminished list of sets will appear compared to previous test because of the significantly lower grade. Also, the set previously created, "Period", set should not appear	Should display a minimal list of flashcards sets	Displays a minimal list of flashcards sets
----	--	---	-------------------------	---	--	--



46	Observe the terms in a set, e.g. Articulation	(1) Grade Selection/Separation	Function and Robustness	Within a set, there will be less terms compared to when the user is grade 8. This test will check this.	Less terms contained within articulation	Less terms contained within articulation
----	---	--------------------------------	-------------------------	---	--	--



47	Play a test	(1) Grade Selection/Separation (20) Motivating	Function and Usability	The test should be easier for a less skilled user, also the approachable questions will make the test more motivating.	Displays an easier test	Displays an easier test, with problems – repeated answers
----	-------------	---	------------------------	--	-------------------------	---

jill
02:52 PM

Learn

1 Test
2 Review

Exit

What is the definition of Tempo?

speed / time (a __: in time)

gradually getting slower

moderately

speed / time (a __: in time)

jill
02:52 PM

Learn

1 Test
2 Review

Exit

What does Crescendo (cresc.) mean?

✓ OK

speed / time (a __: in time)

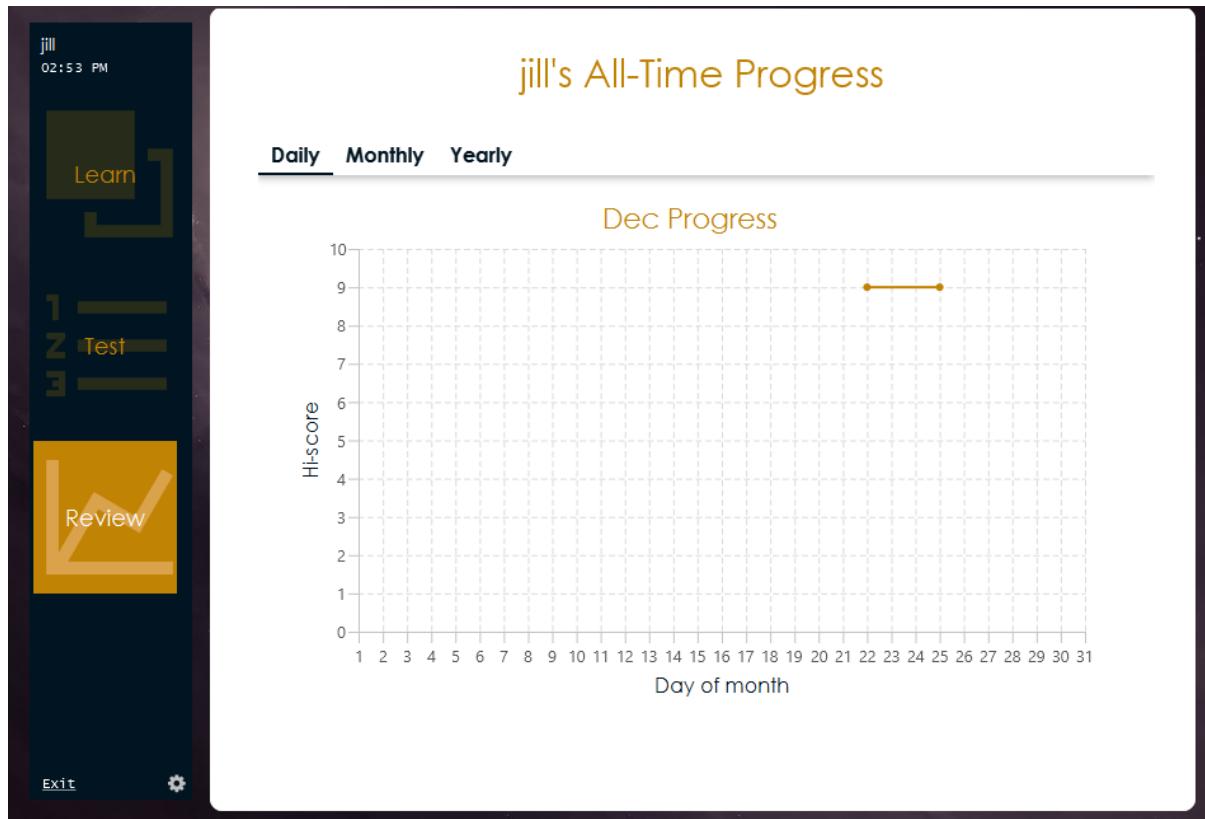
gradually getting louder

at a walking pace

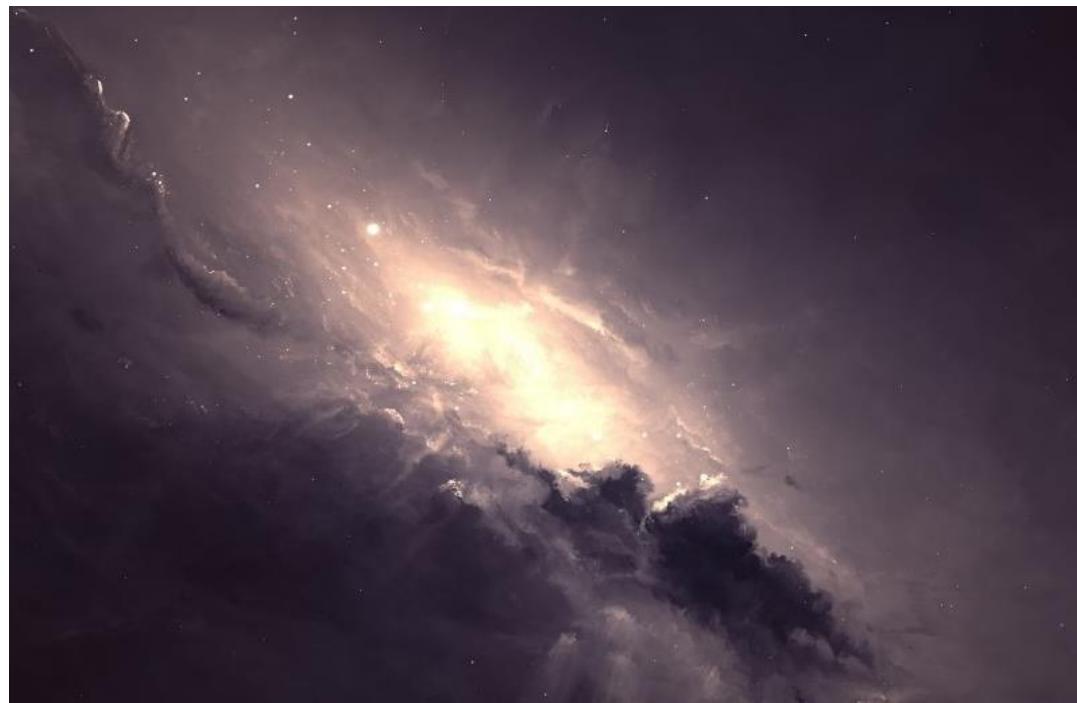
gradually getting louder

48	Click on the Review activity (2) Navigation Side-bar (21) Robust and Error Free	Function and Robustness	Will test for, although a different grade, that all the data points should still be display (on the yearly graph)	Displays a graph joining the two data points, will also include different graded scores since	Displays a graph joining the two data
----	---	-------------------------	---	---	---------------------------------------

		(12) Graphical Progress Display		as users can progress up (or down) in terms of grades.	user can improve/ deprove	points, including different graded scores.
--	--	---------------------------------	--	--	---------------------------	--



49	Click on Exit	(19) Close the Application (21) Robust and Error Free (22) Easy to use	Function, Usability and Robustness	By now, it should be quite easy and memorable on how to exit the program so this test will test for the usability. Also, there should be no errors or misbehavior when clicking exit.	Closes application	Closes application
----	---------------	--	------------------------------------	---	--------------------	--------------------



With all these tests, combined, they will ensure that our system has met its success criteria and that our system satisfies its function, validation, usability and robustness.

Fixing the duplicate answers problem

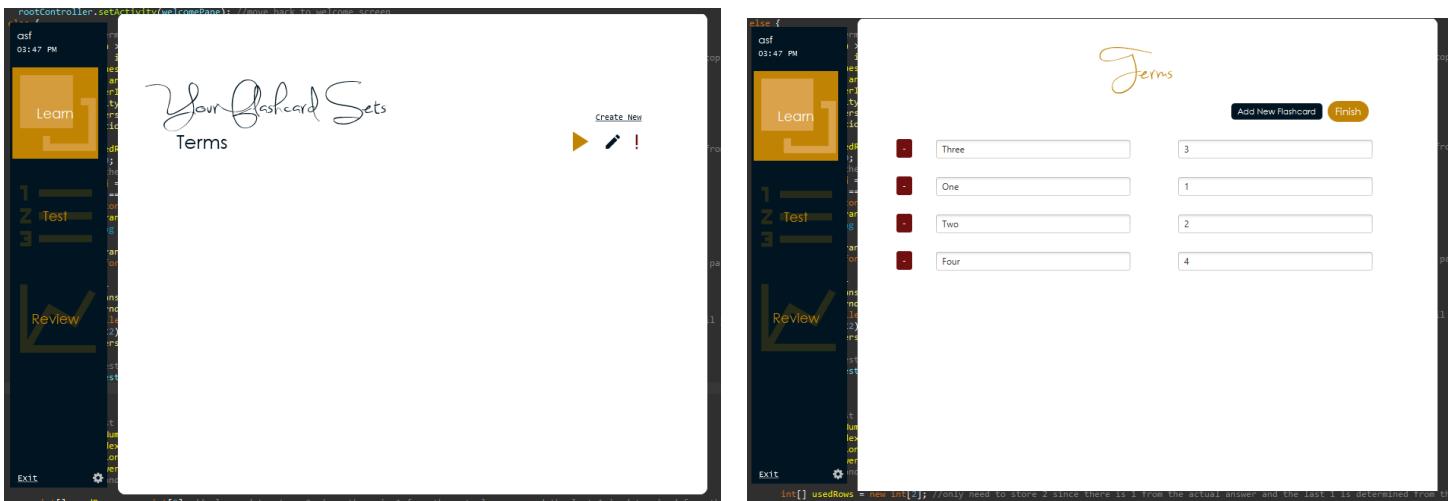
In TestQuizController.java, we can alter our getTheoryQuestions() algorithm so that it validates the answers.

```
for(int i = 0; i<(priorityLength >=10 ? 10: priorityLength); i++) { //if there are more than 10 priority terms, only take the top 10 highest priority (low
    String question = "";
    String[] answers = new String[4];
    int answerIndex = rand.nextInt(4);
    if(priorityRs.next()) {
        answers[answerIndex] = priorityRs.getString("Definition");
        question = priorityRs.getString("Term");
    }
    int[] usedRows = new int[2]; //only need to store 2 since there is 1 from the actual answer and the last 1 is determined from the other 3
    int u = 0; //index of the usedRows
    //fill other answers
    for(int j = 0; j<4; j++) { //four answers (3 random 1 already selected high priority one)
        if(j == answerIndex)
            continue;
        int randomRow;
        String answer;
        do {
            randomRow = rand.nextInt(length);
            for(int k=0;k<=randomRow;k++) { //move pointer to the random row; <= because you want to call next() once to move past zeroth row
                rndRs.next();
            }
            answer = rndRs.getString("Definition"); //get the string representation of the answer
            rndRs.beforeFirst(); //back to start of query result
        } while (randomRow==usedRows[0] || randomRow==usedRows[1] || answer.equals(answers[answerIndex])); //keep looping until you get a unique answer
        if(u<2) usedRows[u++] = randomRow; //store randomRow variable for next answer check
        answers[j] = answer;
    }
    //add question
    theoryQuestionList.add(new TheoryQuestion(question, answers, answerIndex));
}
```

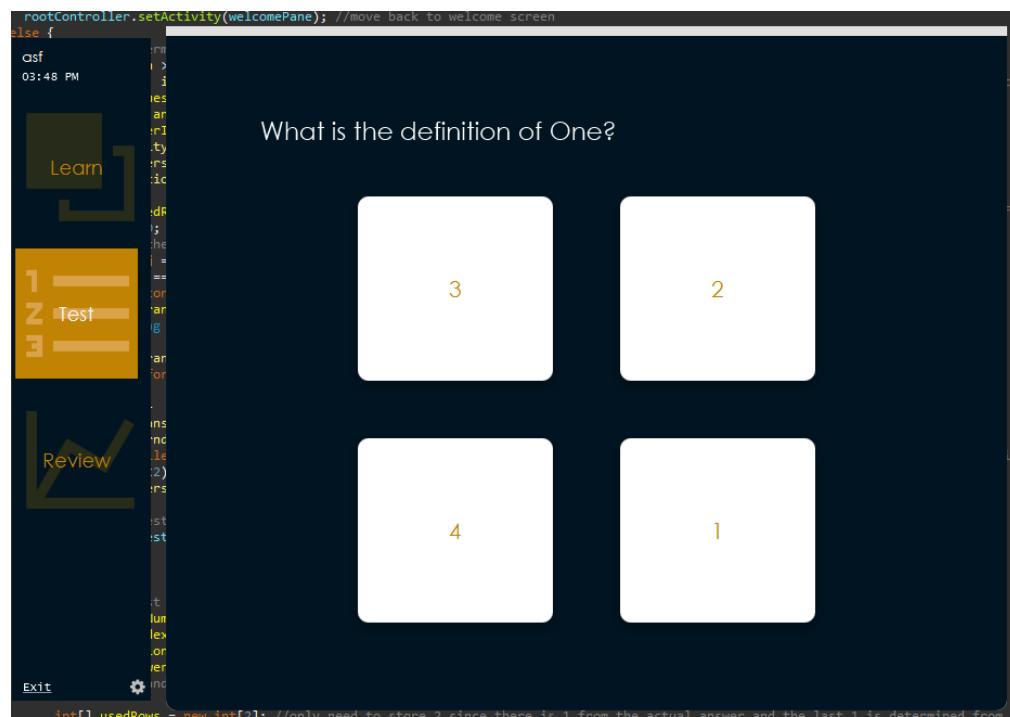
By validating the answers infinitely (loop) until a unique answer appears, we remove the possibility of duplicate answers. Here, usedRows array holds the previously used randomRow variables and so

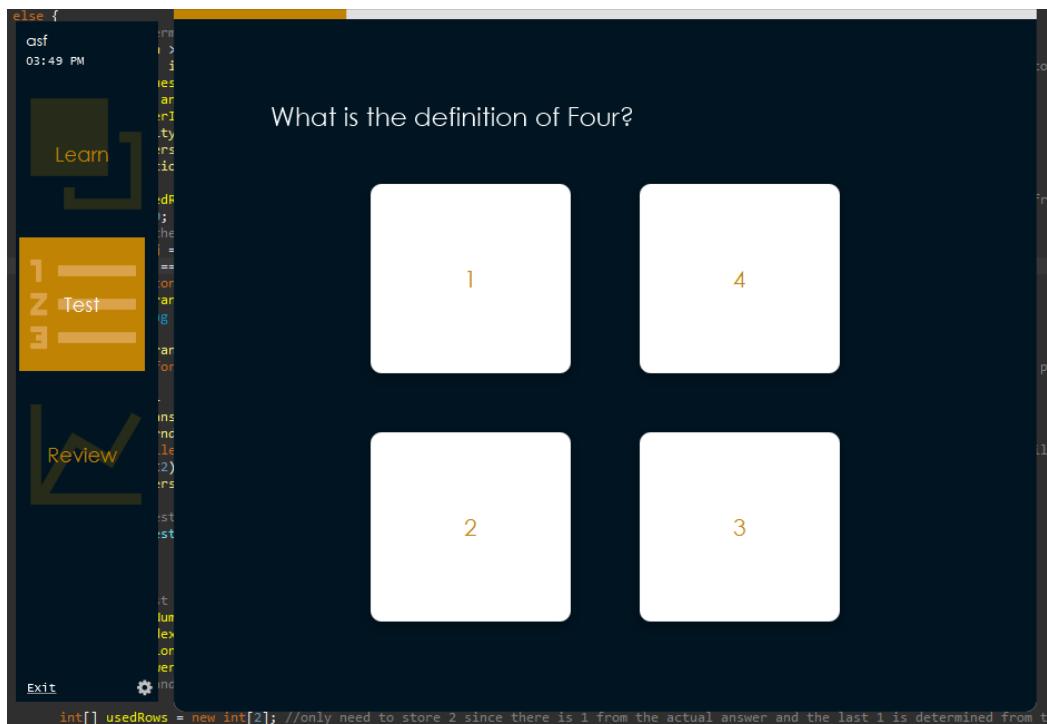
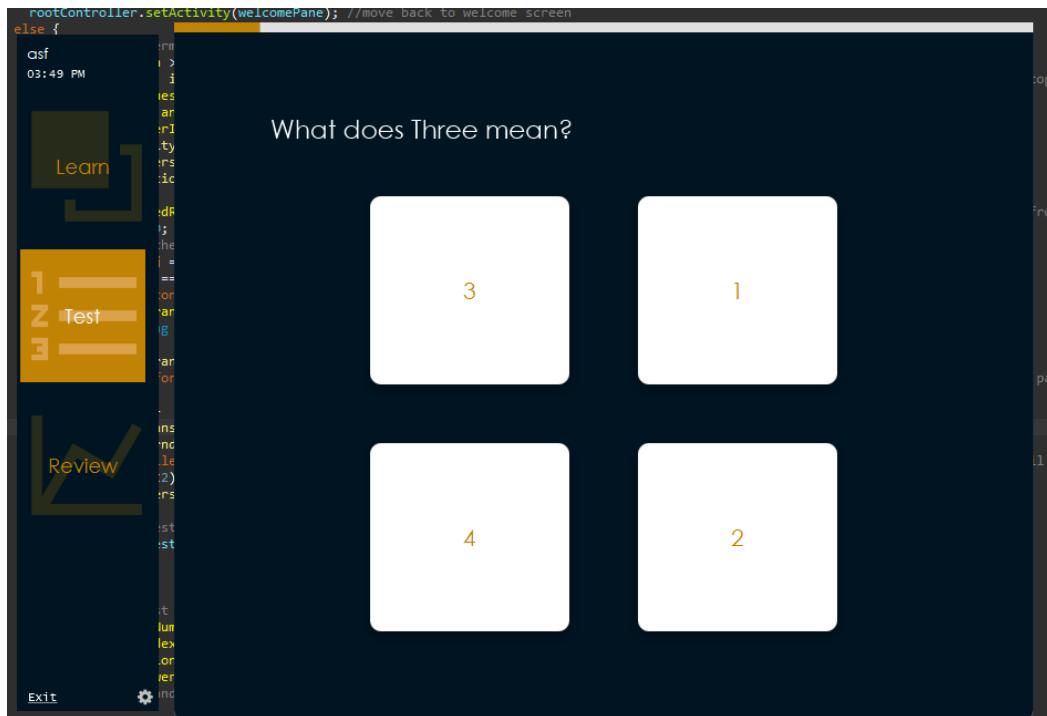
checking the new randomRow variable with the array will remove the duplicate **wrong** answers, and we also compare the correct answer with answers[answerIndex].

To test this, we will include only 4 terms in the database to maximise the chances of duplicate answers.



As a result, there are no longer any duplicate answers occurring in our test feature.





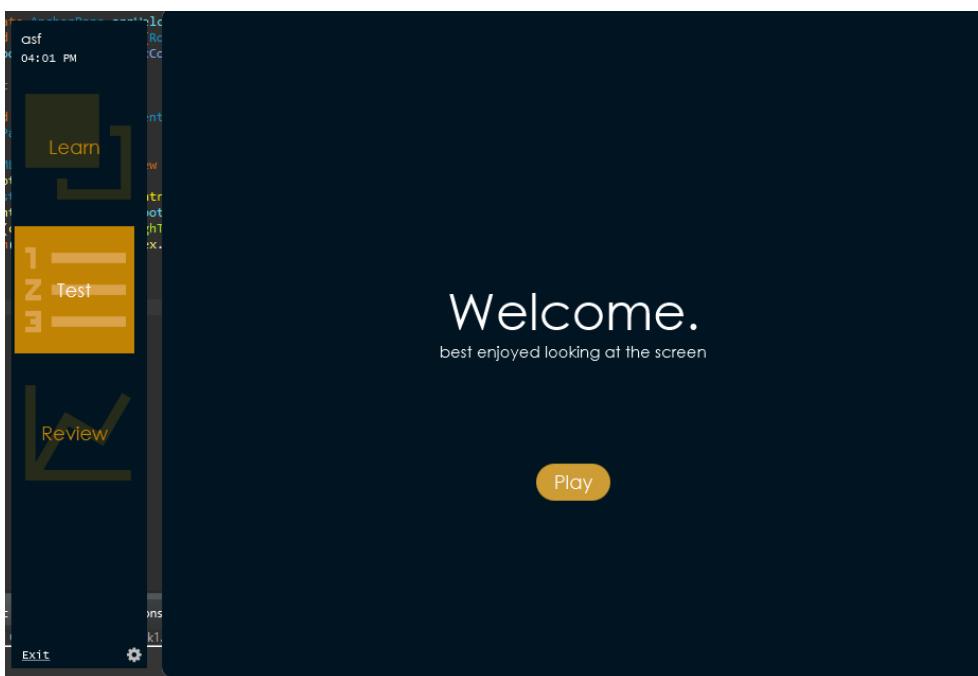
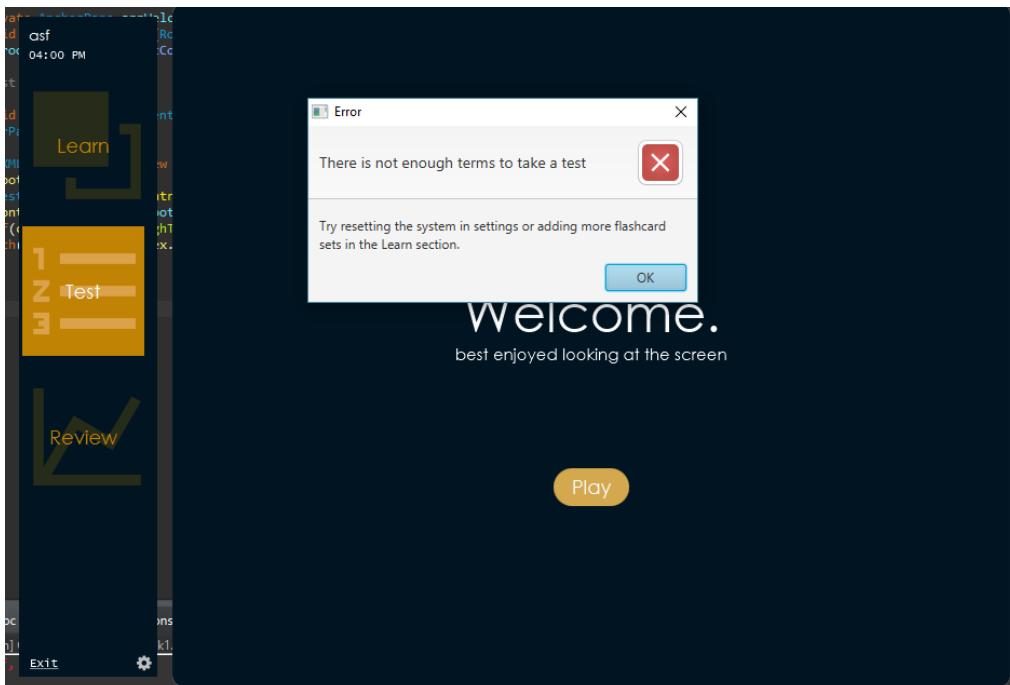
In case a user deletes everything in the database or leaves less than 4 terms (guaranteed duplication) in the database, we will validate it and not play a test if so.

```
if(length < 4) { //if not enough terms
    Alert notEnoughTerms = new Alert(AlertType.ERROR, "hi", ButtonType.OK);
    notEnoughTerms.setHeaderText("There is not enough terms to take a test");
    notEnoughTerms.setContentText("Try resetting the system in settings or adding more flashcard sets in the Learn section.");
    notEnoughTerms.showAndWait();
} else {
    enoughTerms = true;
    //get priority terms|
    if(priorityLength > 0) {
```

Also using TestController.java

```
    // Load the FXML file
    @FXML
    public void playTest(ActionEvent e) {
        AnchorPane root = null;
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/test_quiz.fxml"));
            root = loader.load();
            TestQuizController controller = loader.getController();
            controller.initData(rootController, apnWelcome);
            if(controller.getEnoughTerms()) rootController.setActivity(root); //if there are enough terms then you can load quiz
        } catch(IOException ex) {ex.printStackTrace();}
    }
}
```

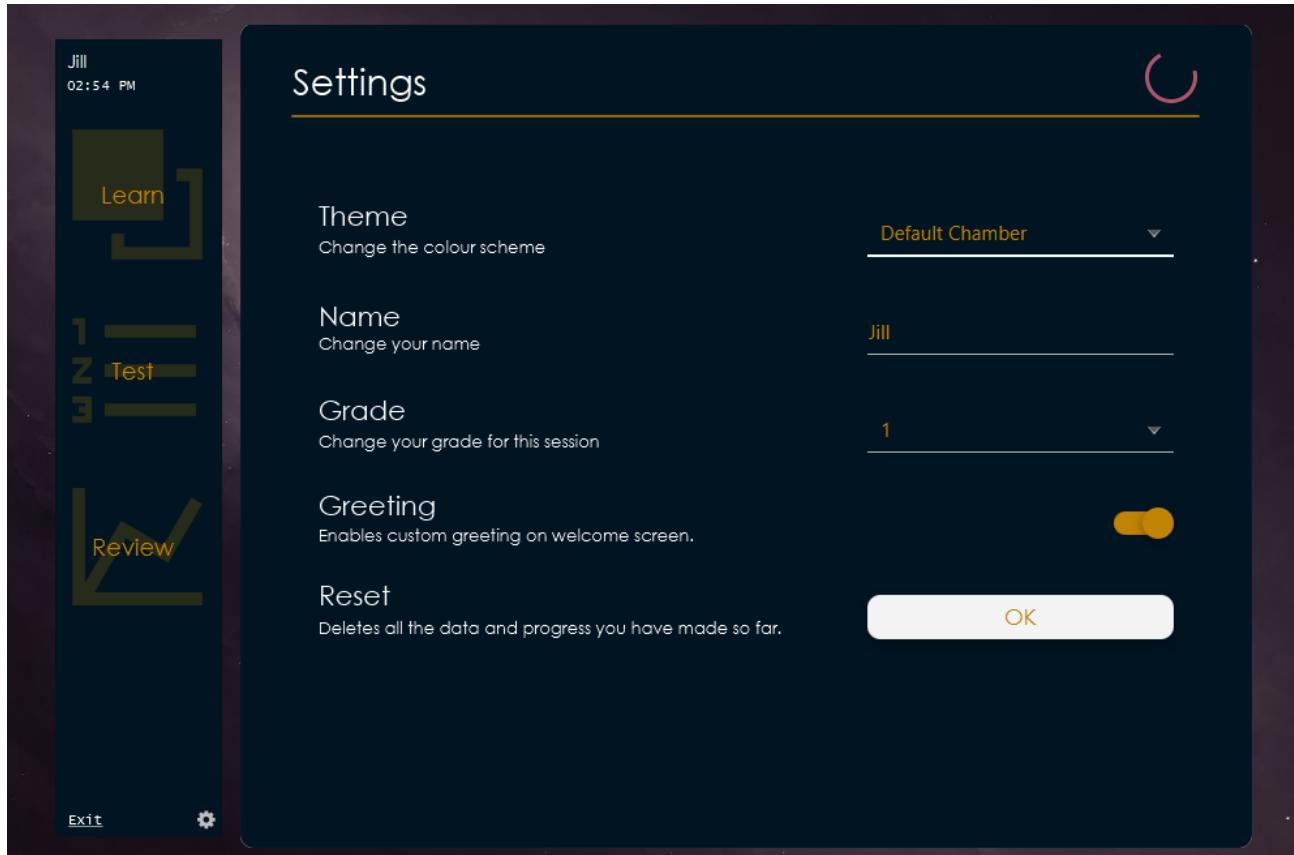
It will only display the quiz section if we have enough terms (at least 4).



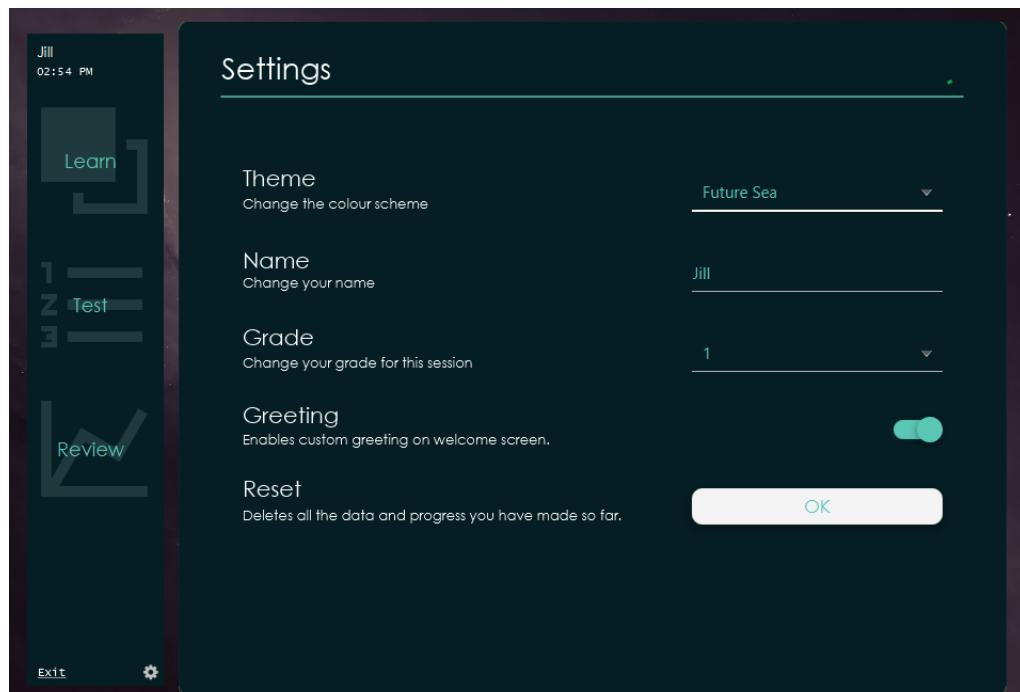
Additional Test 4

This test will test for the extra feature of the settings, this was not documented nor expected in the design stage, however, it has proved useful over the course of our development stage.

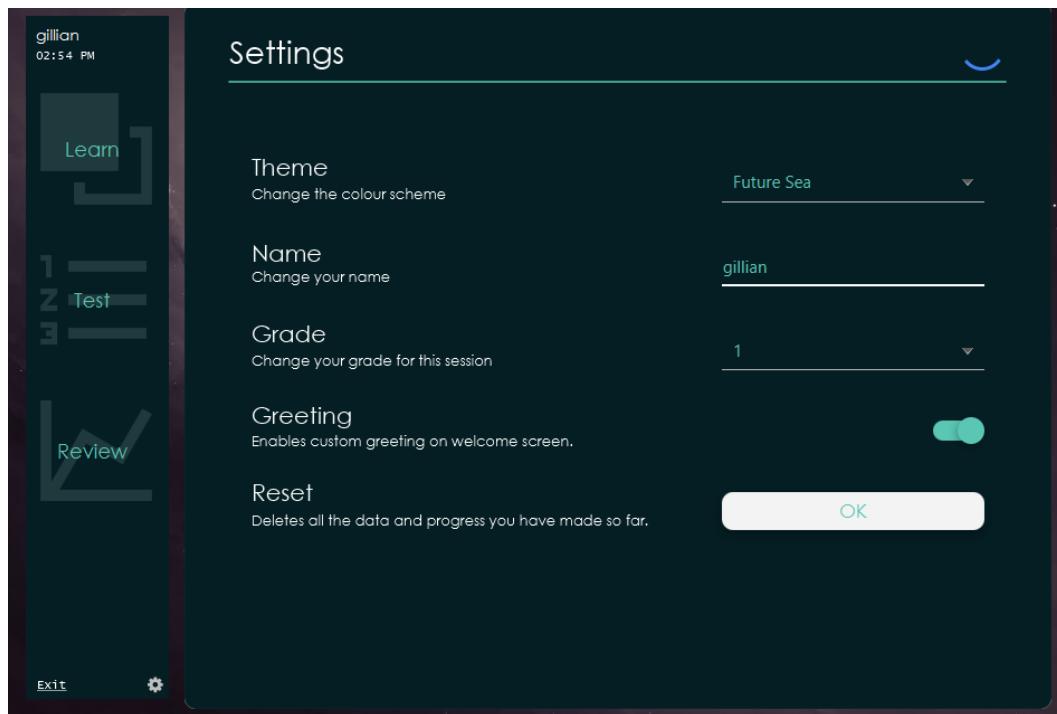
Test ID	Test Data	Aspect	Purpose	Expected Outcome	Actual Outcome
50	Click on setting cog	Functionality and Usability	This will test whether the display of the system with change for the settings	Settings screen appears	Settings screen appears



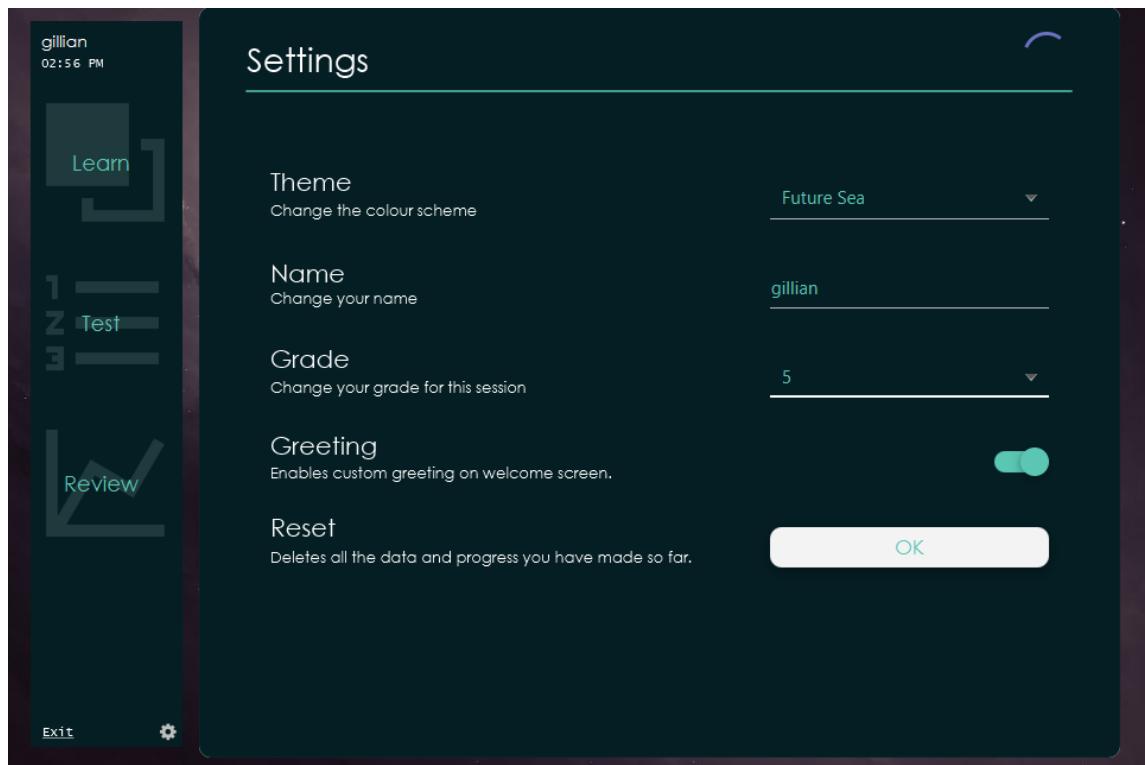
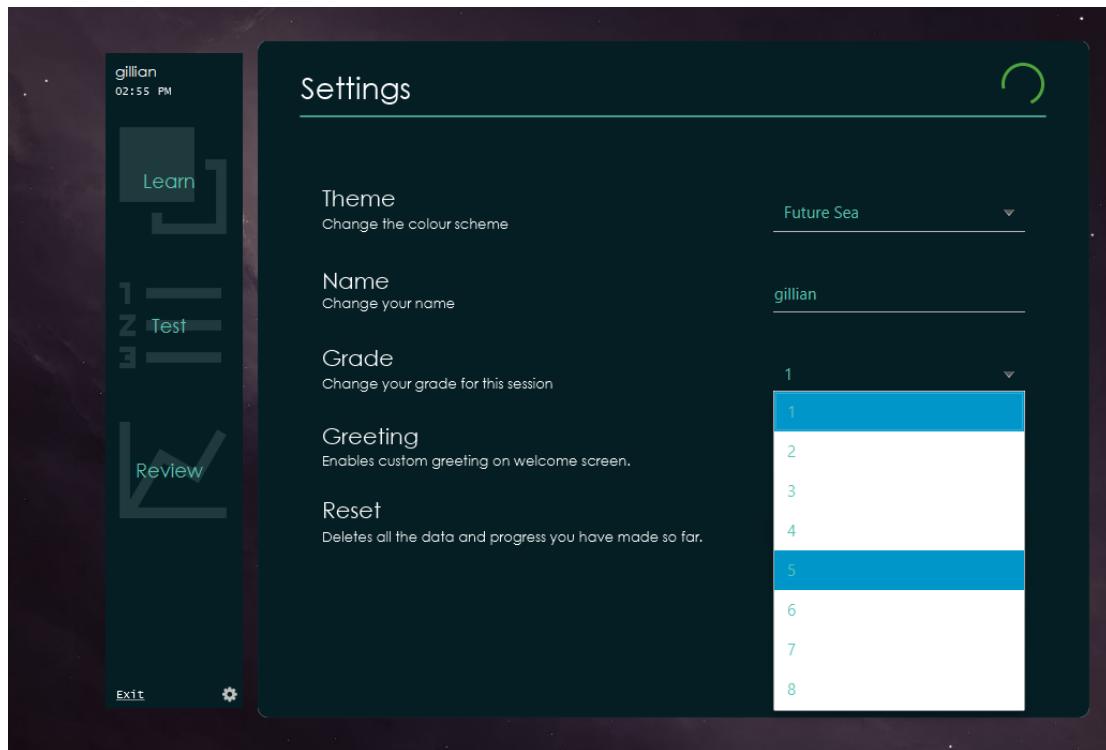
51	Change Theme	Usability and Functionality	To test for the function of the theme changing which adds customizability to the system	Changes the colour scheme of the UI	Changes the colour scheme of the UI
----	--------------	-----------------------------	---	-------------------------------------	-------------------------------------



52	Change name	Usability and Functionality	Changes the name throughout the system	Name is changed for the system	Name is changed for the system
----	-------------	-----------------------------	--	--------------------------------	--------------------------------



53	Change grade	Functionality	Changes the grade which determines the content for each activity	Grade successfully changed	Grade successfully changed
----	--------------	---------------	--	----------------------------	----------------------------



54	Observe Learn section	Functionality	There will be different terms compared to previously because of the change in grade	Content changed	Content changed
----	-----------------------	---------------	---	-----------------	-----------------

gillian
02:56 PM

Learn

1 Test
2 Test
3 Test

Review

Exit

Your Flashcard Sets

- Articulation
- Connectives
- Metre
- Tempo
- Form
- Nouns
- Style
- Adverbs

Create New

gillian
02:56 PM

Learn

1 Test
2 Test
3 Test

Review

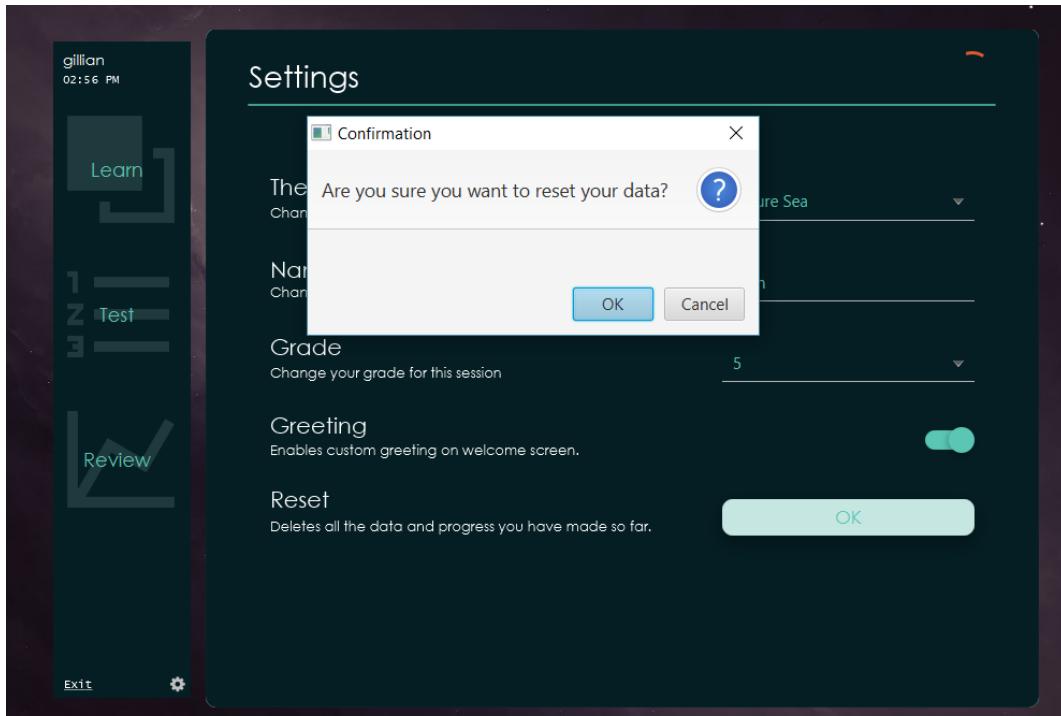
Exit

Articulation

-	Tenuto (ten.)	held
-	Agitato	agitated
-	Delicato	delicate
-	Leggiere	light
-	Calando	getting softer, dying away
-	En Dehors	prominent
-	Légèrement	light
-	Loco	at normal pitch
-	Rinforzando (rf/rfz)	reinforcing

Add New Flashcard Finish

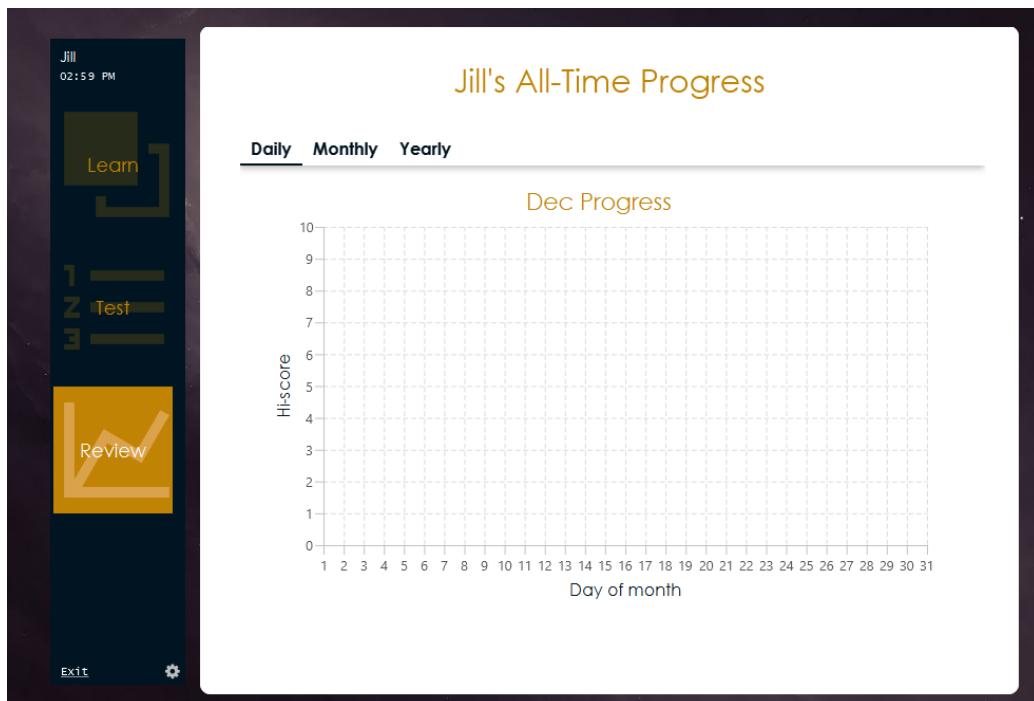
55	Reset data	Functionality, Robustness	If the user modifies the system accidentally in any way, the reset button will revert the changes to original state	Pop up displayed and then confirmed	Pop up displayed and then confirmed
----	------------	---------------------------	---	-------------------------------------	-------------------------------------



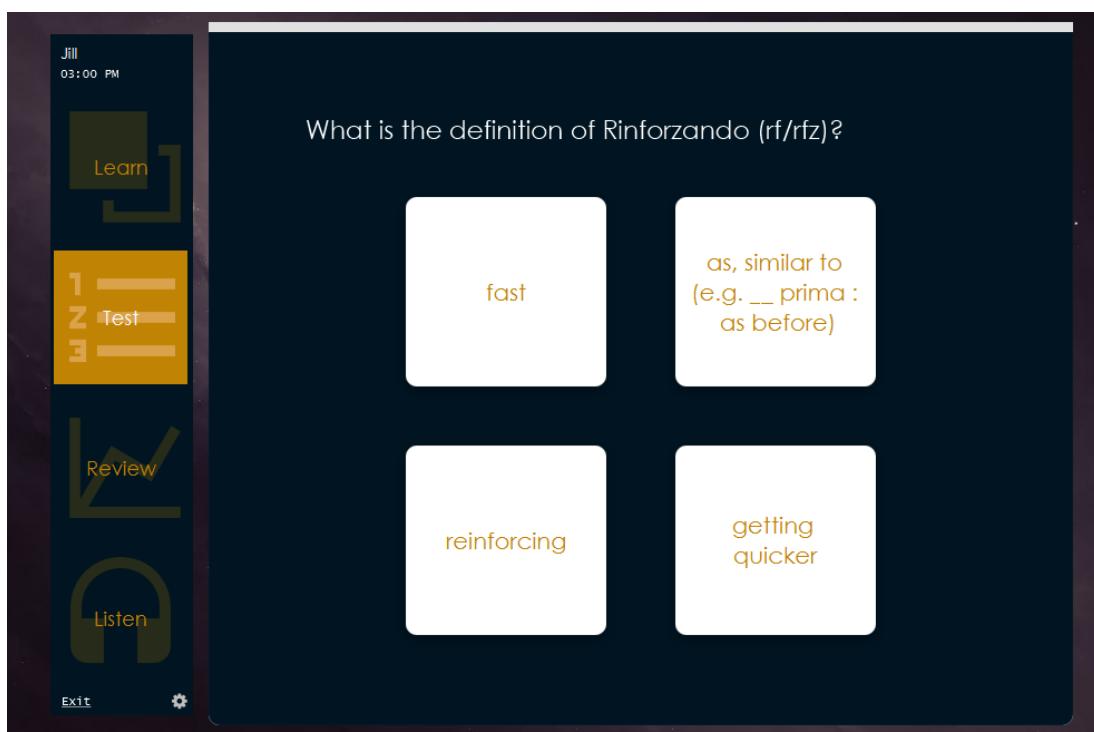
56	Observe Learn section	Functionality, Robustness, Usability	The theme will be reset to Default Chamber, also, the content of the learn will be reset so deleted sets will reappear.	Original list of flashcard sets displayed	Original list of flashcard sets displayed
----	-----------------------	--------------------------------------	---	---	---



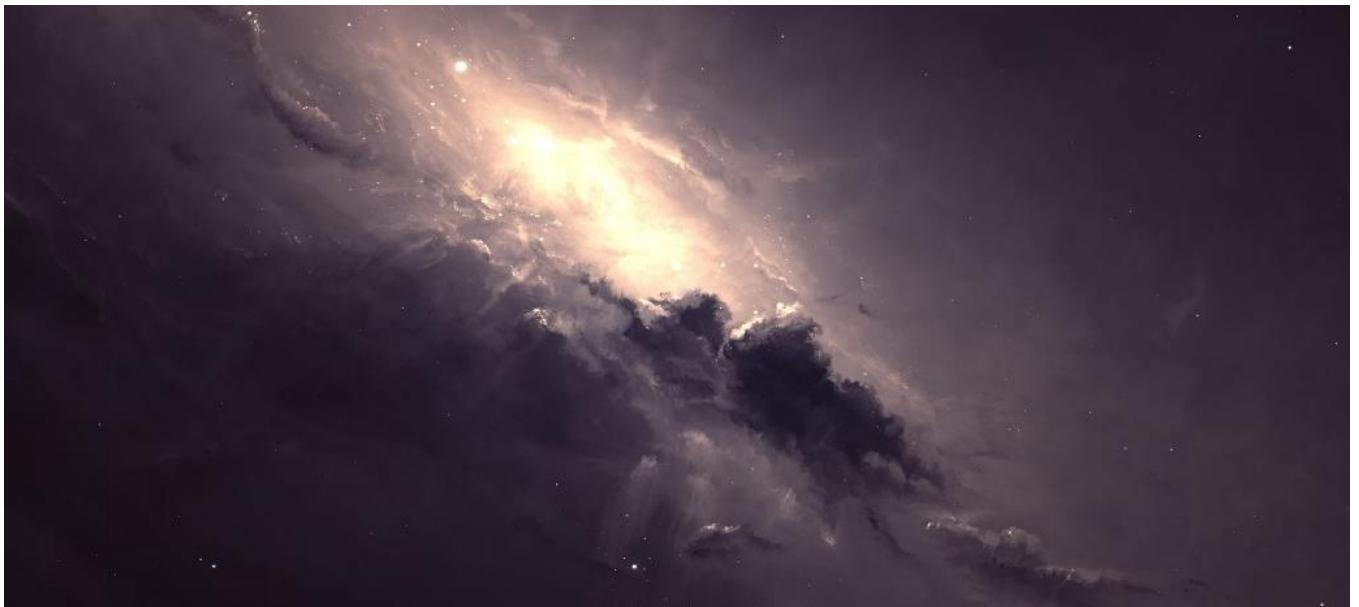
57	Observe Review section	Functionality, Robustness, Usability	All graphs are wiped clean as the scores of the tests will have been reset.	Graphs display no data points	Graphs display no data points
----	------------------------	--------------------------------------	---	-------------------------------	-------------------------------



58	Play a test	Functionality, Robustness	The adapted tests will be reset and reverted to its original state.	Random terms displayed, no priority	Random terms displayed, no priority.
----	-------------	---------------------------	---	-------------------------------------	--------------------------------------



59	Exit system	Functionality, Robustness	The exit button should still work even for a reset system	Application closes	Application closes
----	-------------	---------------------------	---	--------------------	--------------------



After post-development testing for functionality, robustness, usability and validation, including annotated evidence for fixing errors (corrective maintenance), we have provided our main stakeholder with insight of the progress of the system and a thorough, rigorous test of our system, ensuring a quality approved by the user.

Further Comments by Jill Thirkell:

"A modern child will truly be motivated to get high attainments [with this system]. Although I'm remarkably slow at using computers, children will really love this! I think this was still relatively easy to use, the reason for my slowness was more because of my lack of understanding using a laptop rather than the program per se. The different learn approaches was really *different* and more enjoyable than the average looking at the list. Are you going to sell it to ABRSM? I think it would be very successful if you marketed it, there's not many programs on the internet that help students study music and can even replace the teacher!"

Evaluating Solution including Usability Features

Here, I will cross-reference with success criteria to evaluate the solution and explaining how each criterion has been fully, partially or not met – includes evaluation of usability features determining whether they were successful, partially successful or failure. Not met or partially met aspects will be commented on how to be addressed in further development and/or improved with logically structured reason.

Success Criteria Index	Success Criteria	Test ID(s)	Testing Type	Criterion Met?	Comments and Explanation	If not fully met, how can this be addressed?
1	Grade Selection/ Separation	1,2, 37, 43, 44, 46, 47, 53	Functional	Fully	Jill was ‘very pleased’ to see that the content could be separated for different ability users. There was a choice to select the grade at the beginning of the system runtime and content from learn and test sections were successfully separated by the grade. Also, the listen section was exclusive to grade 8 students. Therefore this criterion has been fully met.	N/A
2	Navigation Side-Bar	4, 26, 31, 32, 45, 48, 56, 57, 58	Usability and Functional	Fully	The side bar allowed for seamless navigation for the users where each section could be navigated to using their respective buttons and the settings page could also redirected to. A fully met criterion.	N/A
3	Audio Playback	32	Functional and Robustness	Fully	The correct audio was played when requested for, the music played back to the user when requested as expected. So this criterion has been fully met.	N/A
4	Play/Pause Button	32, 34, 35	Usability and Functional	Fully	Controlled the state of the audio correctly – the pause button paused the music no matter the position and the play button played the paused music at the current position or the start if the audio finished therefore a fully met criterion.	N/A
5	Seek Slider	34	Usability and Functional	Fully	Jill found the slider very useful and beneficial to students who like to improve by repetition. Movement of the slider position corresponded to the audio position. Fully met success criterion.	N/A
6	Question Display	26, 32, 40	Usability	Fully	The correct question was correctly linked with its answer. Question was displayed in a readable format and did was randomly selected. Fully met criterion.	N/A

7	Multiple Choice Answers	26, 32, 40, 57	Usability and Robustness	Fully	Random answers were displayed in their multiple choice boxes, and after the fix, every answer was unique. So a fully met criterion.	N/A
8	Progress in Quiz	28, 40, 57	Usability	Fully	The progress bar at the top of the quiz moved accordingly to the position of the user in the test therefore this criterion has been fully met.	N/A
9	Question Feedback	27, 29, 33, 40, 57	Usability and Functional	Fully	Feedback was displayed very clearly and was 'easy to understand' (use of colours) and was correctly displayed depending on user input and the correct answer – so the criterion has been met.	N/A
10	Results Feedback	29, 40	Usability and Functional	Fully	At the end of every test, correct feedback was displayed and was displayed in sentences for easier understanding for the user and so this criterion has been fully met.	N/A
11	Storing the Data	14, 15, 21, 24, 30, 31, 39, 40, 41	Functional and Robustness	Fully	After multiple tests, the algorithms and queries used to store the data and retrieve the data have been confirmed to be effective and robust – therefore fully met criterion.	N/A
12	Graphical Progress Display	41, 48, 57	Usability and Functional	Fully	Graphs in the Review section correctly display data points and form a line to visualise the progress the user has made– successfully met the criterion	N/A
13	Flashcard Set List	4, 5, 38	Usability, Functional and Robustness	Fully	Within the learn main screen, the list of available flashcard sets is displayed clearly and easily understood – successful criterion.	N/A
14	Creating New Sets	7, 8, 9, 10, 11, 12, 13, 14, 15	Usability, Functional and Validation	Fully	The screen and UI to create a set is easy to use and correctly named new flashcard sets are recorded in the learn sections and used in the test section. So a fully met criterion.	N/A
15	Deleting Sets	16	Functional, Validation and Robustness	Fully	Through the test, the deletion of sets is fully functional and validates the user's choice to confirm their decision with a dialog box and so a fully met criterion.	N/A
16	Editing Sets	17, 18, 19, 20, 21, 22, 23, 25, 39	Functional and Robustness	Fully	Through multiple tests, editing sets have correctly recorded data and changes have impacted not only the learn but the test section as well, so a fully met criterion.	N/A

17	Flashcard Term	5, 6, 15, 22, 25, 39	Usability	Fully	The flashcard term is correctly displayed and changed when navigated – fully met criterion.	N/A
18	Flashcard Definition	5, 6, 15, 22, 25, 39	Usability	Fully	The flashcard definition is correctly displayed, ‘the definitions match’ with the terms. And when requested for, changed when navigated therefore this criterion is fully met.	N/A
19	Close the Application	36, 42, 49, 59	Usability, Functional and Robustness	Fully	Application closes successfully without errors as identified through the tests so the criterion has been fully met.	N/A
20	Motivating	3, 30, 40, 47	Usability	Partially	Although there was an error, which was fixed as shown in the Testing section, the system has been relatively motivating for users and little niches have been used to motivate the user – e.g. progress tracking in the review section. However, there may be better ways that I discover in the future to make the system more motivating so we cannot say this success criterion has been fully met right now.	Implement an even larger variety of learning approaches than just flashcards and testing to give the users an experience which will never bore them – e.g. word fill.
21	Robust and Error-Free	1, 20, 22, 25, 37, 38, 39, 41, 42, 43, 44, 45, 48, 49	Robustness and Functional	Partially	Even through rigorous checks and testing, there may always be a bug or small unwanted malfunction under extreme circumstances that can be later discovered – so it is unsure whether the system is fully error-free and robust. However, any crucial or big errors should have been resolved by the rigorous tests throughout the development and testing of the program.	After higher usage of the system and with a higher user base, the small errors will be eventually weeded out.
22	Easy to use	2, 3, 4, 17, 27, 33, 34, 40, 49	Usability	Partially	This criterion is quite a qualitative variable and surely can be improved, the system is quite easy to use especially using some material design aspects. Howbeit, the main stakeholder, Jill, managed to navigate through the system quite slowly but that may be because of her ‘lack of understanding using a laptop’. Even so, there may be better ways to make less able	In the future, feedback from users can be used to ensure a higher quality of user experience

					users to navigate through the system more efficiently – so cannot say that this criterion is fully met.	and usability – e.g. a better choice of colour scheme.
23	Clean and Efficient Code (Sustainability)	Usability	Partially		Similar to other non-functional criteria, this quality of the solution to this criterion can always change as new algorithms, programming styles are developed and implemented. So, the efficiency of the code is not at its maximum peak because of the possibility of new introductions of technology and also my own limited knowledge. However, I have tried to implement and use the efficient data structures, algorithms and techniques that I know and are compatible with this program.	After learning better algorithms or as more efficient and cleaner styles of coding is developed, the cleanliness and efficiency of the code can be improved significantly.
24	Recoverable	55	Robustness, Usability and Functional	Fully	The reset feature in the settings page allows users to reset the system back into the original state, including the default terms, tests, and files. Even if users find a way to directly change the database or even delete the system from secondary storage, it can be redownloaded as a fresh start only including the original data anyways. So, this criterion has been fully met.	N/A

To conclude, this system has been very successful as it has mostly met its criteria fully but with partially met criteria because of the unmeasurable and dynamic nature of some criteria.

Evaluating Maintenance

All aspects of a program's code should be able to be easily understood by any capable programmer. This allows future updates or maintenance to be efficiently done to the code without a high amount of overhead caused by the time to comprehend the code.

Positive Signs of Maintenance

Below is an extract of code from TestResultsController.java.

```
//other important variables
private LinkedList<Feedback> feedbackList;
private ObservableList<String> correctTerms;
private RootController rootController;
private int result;
private DAO dao = new DAO();
//ui objects
@FXML private Text txtScore, txtComment;
@FXML private VBox vbxFeedbacks;
public void initData(RootController rc, LinkedList<Feedback> fbl, int result, ObservableList<String> ct) {
    rootController = rc; //set rootController for activity changes
    feedbackList = fbl; //list of feedback
    this.result = result; //take the result of the test
    txtScore.setText("You scored " + result + "/10!");
    txtComment.setText(comments[result]); //a comment depending on the result
    correctTerms = ct;

    //add the feedback to the vbox
    for(Feedback feedback: feedbackList) {
        Text txtFeedback = new Text();
        txtFeedback.setFont(Font.font("Century Gothic", FontPosture.REGULAR, 16));
        txtFeedback.setText("For " + feedback.getQuestion() + ", it's not " + feedback.getWrong() + " but it is " + feedback.getCorrect() + ".\n");
        vbxFeedbacks.getChildren().add(txtFeedback);
    }

    // store into sql
    updateDB();
}
public void updateDB() {
    try{
        Connection conn = dao.getConnection();
        PreparedStatement checkKStmt = conn.prepareStatement("SELECT K_Coefficient FROM Theory Where Term = ?"); //check the current k coefficient
        PreparedStatement higherKStmt = conn.prepareStatement("UPDATE Theory SET K_Coefficient = K_Coefficient + 1 WHERE Term = ?"); //for correct answers
        PreparedStatement lowerKStmt = conn.prepareStatement("UPDATE Theory SET K_Coefficient = K_Coefficient - 1 WHERE Term = ?"); //for wrong answers
    }
}
```

Indentation – each enclosing body (e.g. a method or iterative loop) has indented the following lines between the curly brace { and } with 4 whitespace characters (or TAB). This allows for the structure of the solution to be easily understood and separates different parts of code.

Comments – there are comments to explain the important details of how algorithms are carried out.

Breaking up multi-stage algorithms using whitespace – algorithms like the initData() procedure may have multiple steps or stages for the algorithm to proceed with its function and so are separated with blank whitespace lines to illustrate the different stages – making it easier to understand.

Meaningful names – names of objects and primitive data variables are meaningfully declared. For example:

- result – stores the integer result of the test
- dao – object of the data access object
- txtScore – text element displaying the score of the test
- txtComment – text element displaying a comment dependent on the result of the test

- vbxFeedbacks – a vbox holding the feedback elements

Naming conventions – uses naming conventions like 3 character prefixes to determine the data type of an element, such as:

- txtScore – object of Text
- txtComments – object of Text
- vbxFeedbacks – object of VBox
- txtFeedback – object of Text

Use of local variables – where possible, I have maximised the use of local variables, constants and objects to make the code more robust and less susceptible to unknown changes (initialisations) onto higher scoped variables. Local variables will also make the code easier to change and debug.

Storing into database – all the data required for the core functionality of the system is stored in the database where the data is isolated at one location and so is easier to manage and change. Also, it means data will not be hard-coded into the system.

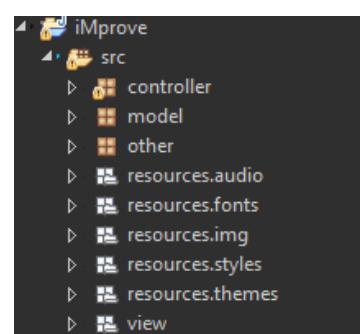
Use of data structures – related data is stored into collections which will be easily iterated through to access each element such as feedbackList storing each of the individual feedback elements. With data structures, it will be much easier to add/change/remove certain elements from the code making the elements much more easier to manage and maintain.

Self-documented code – although comments help with the explanation of some complex sections of code, sometimes well written code should explain themselves without the requirement of comments. For example, setting the font of txtFeedback to Century Gothic size 10px.

Use of classes – by using object-oriented techniques such as encapsulation, the data which are related to one another are bundled into a single entity of a class. This prevents later conflict with data and will make the code more robust and maintainable. Also, because of classes, information will be hidden or not depending on private or public (or protected) access modifiers which will manage the access of certain variables or methods and can prevent objects of other classes to change the values of the variables or call certain methods.

Use of packages to separate MVC architecture – my system uses packages carefully to separate resources and classes into respective model, view, controller packages. This allows for modularity in code and so is easier to isolate problems, easier to update sections of the system and/or add/remove data into/from resource packages.

Adaptive maintenance – the modularity of this system (using MVC architecture, classes and packages) allows for adaptive maintenance to be efficient and easy to implement since the changes can be made in one place. The use of control arrays and other collections allows for changes to the number of controls to be easy to manage and implement.



Possible improvements for maintenance

Use of version control – with version control systems such as Git, it will be easier to go back to previous editions or versions of the program. This allows to revert changes much easier and prevent complex, hard to debug errors from appearing or even understand why they appeared. Either way, version control would be a helpful addition to the maintenance of the system

Storing data with relational database – using a relational database, redundancy of data can be reduced and this leads to easier modification of data. For example, if fields repeatedly indexes to another table holding the actual data values, it is much more efficient and clean to change the data as it is in one place rather than scattered around the table. In addition, adding a relational database can allow for the storage and maintenance of multiple users using the program.

Update system – with future corrective, adaptive or perfective maintenance to the program, the system will develop, and with a built-in update system, it will allow for a much easier way to distribute the changes rather than deploy a new system every time a change has been made after release.

Limitations and Potential Improvements/Changes

Limitations and Improvements

- **Sight impairment accessibility** – the flashcards, tests, review and even the listen feature require undamaged sight to use these features. Currently, the system is rendered useless by this limitation for sight-impaired users however, with audio reading out each question or each term and respective definition (depending which is shown), it would improve the accessibility of this system.
- **Sound recognition** – the system does not recognise sound and some parts of aural testing require a listener to measure the level of performance and aptitude in aural testing. As better technologies develop to accurately record and analyse sound, or with enough time to learn and develop a sound recognition system, the aural testing will be even more functional than it is now.
- **Money, resource and time budget** – as there were no budget assigned to this system, there were no professionals to expand this system, better software technology to aid this system and enough time to fully develop this system. With a larger budget, the system could be expanded with more features such as more ‘learn’ approaches, more aural tests (different grades than 8, more types of aural test and the quantity of tests for each).
- **Deployed as an unpopular developer** – since I currently have no reputation or status as a developer, this program may not be trusted by some users and can limit the deployment and distribution of the system. However, as my system is slowly accepted by users, it can spread via those users and eventually become respectable if this system is effective enough.
- **Networked** – this system is not networked so direct help with music professionals such as the main stakeholder, Jill Thirkell, could not be accessed and instantaneous feedback or even guidance by tutors cannot be received. This can be improved with later development

where chats or some sort of communication is implemented which can improve the productivity and enthusiasm of the users! However, if it were networked with Java socket technology, it would require a continuously running server to be held and neither me or the main stakeholder would like to run a server on their machine 24 hours a day, so, other technologies should be used to develop the networked system.

- **Updates** – as identified in Evaluating Maintenance, the updates of the system will be quite tedious to manage in the perspective of the users as they have to download and install a new form of the program each time a major update is made, by using update checking to a server, updates can be searched and retrieved much more efficiently and easier for the user to handle.

Extensions

- **More learn approaches** – my system currently only includes flashcards as its form of own-paced learning before tests. With a higher variety of approaches, the system will be more enjoyable by the users as they can never get bored and will be engaging enough to keep them using it and making progress.
- **Aural tests for each grade** – although my system only provides aural testing for grade 8, each grade has its own form of aural testing which will require its own unique set of algorithms and coding, by developing these, the system can provide aural testing help for a higher range of students.
- **More types of aural testing** – my system only includes modulation for aural testing but in fact, aural testing can include multiple steps and types including cadences, melodic repetition etc. With more time and resources, these can also be developed increasing the size of the project but also the audience as well.
- **User separated content** – although this system can be used by multiple users anyways, they must share the same data if they use the program on the same machine. To improve this, relational database technology can separate the data and allow for individual progress to be recorded and allow the system to adapt the system for each individual.

Conclusion

Although the main stakeholder, Jill Thirkell, is very pleased with the result of the system and is very glad to use it in her future teaching, there is still room for improvement in this system and there are many extensions that can be developed in the future to make this system more effective and enjoyed by a higher array of students and teachers.

Other

Bibliography

Research of Existing Solutions

Mymusictheory.com. (2018). *Grade 5 Orchestral Instruments Quiz*. [online] Available at: <https://www.mymusictheory.com/learn-music-theory/for-students/quizzes/> [Accessed 2 Oct. 2018].

Hofnote (2018). *Listen, Understand, Enjoy*. [online] Available at: <https://hofnote.com/courses.asp> [Accessed 2 Oct. 2018].

E-Music Maestro. (2018). *Online Aural Test Practice for ABRSM & Trinity Exams – Grades 1 to 8*. [online] Available at: <https://www.e-musicmaestro.com/auraltests/free> [Accessed 2 Oct. 2018].

Quizlet. (2018). *music theory Flashcards and Study Sets | Quizlet*. [online] Available at: <https://quizlet.com/subject/music-theory/> [Accessed 3 Oct. 2018].

Research used for Development

Molevalleymusic.co.uk. (2018). *Theory at MoleValleyMusic.co*. [online] Available at: <http://www.molevalleymusic.co.uk/Theory.htm> [Accessed 15 Dec. 2018].

Music Theory: Study, Practice and Pass Grade 5 Theory with Clements Theory. (2018). *Musical terms reference*. [online] Available at: <https://www.clementstheory.com/reference/musical-terms/> [Accessed 15 Dec. 2018].

closed, D. and C, K. (2018). *Database File 'is being used by another process' when all connections are closed*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/53109568/database-file-is-being-used-by-another-process-when-all-connections-are-closed> [Accessed 4 Nov. 2018].

Appendix A – Interview with Jill Thirkell

Date of interview – 21/08/2018

Interviewer – Kevin Cen

Interviewee – Jill Thirkell

–start–

Please describe the work that you do.

I teach piano skills from beginners through to grade 8 and occasionally diploma work. I teach theory; I can teach the recorder for anyone who wishes to play the recorder, mostly young children, and I help with GCSE with performance and I give guidance with composition. Also, if necessary, listening techniques.

Alright, how long have you done this work?

Since 1972… so 46 years.

Please can you describe your typical client group?

I haven't really got a typical, I've got children from age give to somebody in her late seventies. I can't really, can I?

Okay, there's a wide range isn't there?

Do you use technology for any part of your teaching currently?

Yes, I do use ‘musescore’ on the computer for pupils who are interested in composing and also recommend every now and then applications to help them with their sight reading

Do students use this often?

If they really enjoy it, yes we'll do it regularly. But I haven't found many pupils that are honest and want to continue using it.

Do parents keep track/know their child is actually learning from it?

Some of them do but some do not have enough input whereas quite a few parents are over pressuring and I think this has a negative effect on them learning… maybe I'm wrong.

OK that's fine.

How do students currently prepare for theory?

They go gradually by puzzle books to as far as they want to go. But usually, if they display resistance to their learning, I stop at grade 2, as it can get difficult.

Do you think it can improve and how? (more motivating learning, more enjoyable, easier to access...)

Usually only looking out for new, better books but I think it's not motivating enough. I try choosing words carefully like always saying 'play' the piano rather than 'practise' because I want to use vocabulary to make music an enjoyable experience rather than something they have to do at home.

Do you think a particular aspect of theory is lacking and difficult to teach students to grasp, and can be improved with better resources?

Some of the students are not very clever at maths so sometimes it's very difficult and I'll need to use methods used in an infant school to teach fractions, like chopping up a chocolate biscuit into four, right? But I don't think it can be improved with better resources, I like to use concrete explanations that are physical for these types of teaching.

Would you want a computerised system to help your students to learn that particular aspect?

I'm not particularly interested on a computerised system for these types of learning; I think this type of maths is really easy to teach and do not require much outside help.

Yep, okay, that's fine.

Which type of students would you like to aim this system for: child, adult, elderly?

I think it would be for the middle group between children and adults, I think for teenagers.

Do you think it's a good idea for the parents to keep track of their child's progress?

Well, it would be yes... it should be! I normally give out progress sheets to the parents of children already.

Are there any other areas you believe technology would aid you in your teaching?

Maybe aural testing? I think it's good to listen to extracts of music at home and try identifying cadences et cetera but for other things like if I want to distinct two beats from three, I like to do it in a concrete fashion and we march, this is a child age seven or eight, left right left right, or we waltz.

OK. So my current idea for my project will be, firstly, will have a vocabulary functionality where it can produce flashcards and ways to memorise those Italian, latin terms etc.

Yes, and also names of notes? And for the value of notes? Clefs? Beats in a bar? And this will all be on the computer will it?

Yeah

That would be helpful, absolutely helpful.

And so, also have test which will score them on how well they learn them terms and parents can keep track...

That sounds very interesting to me, yes.

with graphs which analyse their progress over time.

Yes, that sounds like a very motivating program, better than even my gold stars

(We laugh)

Also there may be an extra aural feature which can play excerpts where after the students identify features like modulations, chords, cadences…

This is higher grades 6, 7, 8 isn't it? Well yes, this would be good but also include some of the work from lower grades like clefs.

OK. Do you believe tracking progress is a good feature to have?

At home? Then it will be quite good – since it's harder to track a child's progress if they are at home! And may also help parents remove doubts of whether they are improving or not. A peace of mind.

Can you describe the type of interface you would like?

It mustn't be too animated and decorated since I feel like it could be distracting. I'm not too sure what children like really but I find distractions a nuisance.

How about colour wise – quite professional looking or brightly coloured for example?

The way I am, I'm quite averse to colour. I don't like notes to be colour coded for example and nothing like that. It's got to be, for me, black notes and black letters on white. I think different colours is just an added distraction for the child to have to go through for them to learn.

How about for a logo?

A logo? I don't really mind, I'm going to leave that with you.

Yeah okay, that's fine. How about the layout of the program, for example any particular desires on multiple choice answers or descriptive answers.

I think multiple choice is a good format since I feel students do not need to conger up [think of] wording or descriptions and I'm not sure you agree but I think multiple choice is a quick way for learning. Also, maybe something to tell and explain why the answers are wrong when applicable? This doesn't apply to everything but for key signatures you can say X is wrong and the correct one is Y, do you know what I mean?

Yeah.

Okay, do you want this system to be stand-alone(by itself with no required interaction as administrator)?

I'm not too sure, it would be easier if it didn't need any of my input but then again, it would also be useful if I could communicate with them. So I don't know, whichever is fine.

What type of inputs would you expect the student to enter?

I'm guessing answers? And possibly questions if they made their own test?

Yeah okay, what type of outputs do you expect the program to have? Possibly feedback to give back to the students, scores of students in tests, statistical analysis with graphs?

Yes, those are all very good ideas. But I'm more keen on learning than testing, I don't want them [students] to feel they're coming here to be tested. I have quite an aversion to testing but I still do it because I feel it's beneficial but I do it in a more discreet way, do you see what I mean? Every lesson is a test in a way but I don't like to call it a 'test' you know?

Yeah I get you.

Err, what benefits do you hope to gain from this program?

I hope my students will become more motivated to continue learning. I want my students to see that music is enjoyable, although it's hard work, it's a hard language to learn.

What would you consider are the main factors of judging the success of this new program?

It's got to be very straight-forward... it mustn't have any glitches in it, do you know what I mean?

Yeah, yeah.

I hope my students will use this in practice, again it must be something that makes my students want to learn.

If this system is successful, can you envisage any new ways to extend this program?

It's very nice if there was a convenient way to interact with the students which save time to give them more in depth explanations and go through a piece more thoroughly through a messaging system or something.

Yeah, like being networked.

Yes.

Are there any other things you would like the new system to include?

I don't know. I can't be sure yet. But I hope it will be easy to use and my students to be motivated and enjoy using it.

Okay, that's all! Thanks.

No problem

-end-