College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
IN721: Design and Development of Applications for Mobile Devices
Level 7, Credits 15

Assessment 01: Language Translator

Assessment Overview

For this assessment, you will use Kotlin with Android Studio to build a language translator. As well as implementing the core functionality, you will be required to research & implement seven components. In addition, marks will also be given for code elegance, functionality, robustness & git usage.

Assessment Table

Assessment Activity	Weighting	Learning Outcomes	Assessment Grading Scheme	Completion Requirements
Practicals	25%	1, 3, 4	CRA	Cumulative
Language Translator	20%	1, 3, 4	CRA	Cumulative
Wishlist	25%	1, 3, 4	CRA	Cumulative
Exam	30%	2, 3, 4	CRA	Cumulative

Conditions of Assessment

You will complete this assessment outside timetabled class time, however, there will be availability during the teaching sessions to discuss the requirements and progress of this assessment. This assessment will need to be completed by Friday, 01 May 2020 at 5pm.

Pass Criteria

This assessment is criterion-referenced with a cumulative pass mark of 50%.

Submission Details

You must submit your program files via **GitHub Classroom**. Here is the link to the repository you will be using for your submission – https://classroom.github.com/a/44o8YJ7J. For ease of marking, please submit the marking sheet with your name & student id number via **Microsoft Teams** under the **Assignments** tab.

Authenticity

All parts of your submitted assessment must be completely your work and any references must be cited appropriately including, externally-sourced graphic elements. All media must be royalty free (or legally purchased) for educational use. Failure to do this will result in a mark of zero.

Policy on Submissions, Extensions, Resubmissions & Resits

The school's process concerning **Submissions, Extensions, Resubmissions and Resits** complies with Otago Polytechnic policies. Students can view policies on the Otago Polytechnic website located at https://www.op.ac.nz/about-us/governance-and-management/policies.

Extensions

Please familiarise yourself with the assessment due dates. If you need an extension, please contact your lecturer before the due date. If you require more than a week's extension, a medical certificate or support letter from your manager may be needed.

Resubmissions

Students may be requested to resubmit an assessment following a rework of part/s of the original assessment. Resubmissions are completed within a short time frame (usually no more than 5 working days) and usually must be completed within the timing of the course to which the assessment relates. Resubmissions will be available to students who have made a genuine attempt at the first assessment opportunity. The maximum grade awarded for resubmission will be C-.

Learning Outcomes

At the successful completion of this course, students will be able to:

1. Implement complete, non-trivial, industry-standard mobile applications following sound architectural and code-quality standards.
2. Explain relevant principles of human perception and cognition and their importance to software design.
3. Identify relevant use cases for a mobile computing scenario and incorporate them into an effective user experience design.
4. Follow industry standard software engineering practice in the design of mobile applications.

Instructions

Application Requirements - Learning Outcomes 1, 3, 4

The language translator application **must** have the following functional requirements:

- System:
  - Open without modification in Android Studio. This includes the removal of the gradle & idea hidden directories.
  - Run without modification on multiple mobile devices.
- Features:
  - Text translation which includes support for at least four languages, excluding English.
    - Display a progress dialog while in translation.
    - Handle incorrect formatted input values.
  - Localization which includes support for at least four languages.
  - Exit the application via an alert dialog.
  - **Research:** Text-to-speech using the TextToSpeech Android class or a third-party library.
  - **Research:** Multiple mobile screen/device support. Include support for at least four screens/devices, excluding tablets & smartwatches devices.
  - **Research:** For each language, create an interactive quiz.
    - Each quiz must have at least five questions. For example, questions can be on colours, animals, numbers, etc.
    - Each question must have an image & at least four answers.
    - Provide appropriate feedback for correct & incorrect answers.
    - At the end of each quiz, the score is saved to shared preferences.
    - For each quiz, display the highest score.
- User-Interface/Experience:
  - Visually attractive user-interface with a coherent graphical theme and style. This can be a custom theme/style or Material Design.
  - **Research:** Toggle between light & dark mode.
  - **Research:** Splash screen with a transition animation.
  - **Research:** Adaptive launcher icon.
  - Bottom navigation which navigates the user to the appropriate activities.
- Publishing:
  - Application published to the Google Play Store.
  - Download application from Google Play Store on mobile device.

Documentation - Learning Outcomes 1, 3, 4

Write the following documentation requirements in the README.md file:

- Step-by-step user guide. Must include a screenshot of each activity/fragment provided with a description.
- **Research:** Code commented & documented using KDoc. Kotlin's documentation generation tool is called Dokka. Click here for usage instructions.

Git Usage - Learning Outcomes 1, 3, 4

The language translator repository must have the following git requirements:

- For each feature & user-interface/experience functional requirement, create a branch. **Note:** There should be at least nine branches excluding master.
- Commit messages reflect the context of each functional requirement change

Additional Resources

- Documenting Kotlin Code - https://kotlinlang.org/docs/reference/kotlin-doc.html
- Android Design - https://developer.android.com/design
- Adaptive Launcher Icon - https://developer.android.com/guide/practices/ui_guidelines/icon_design_adaptive

Git Usage - Learning Outcomes 1, 3, 4

The language translator repository must have the following git requirements:

- For each feature & user-interface/experience functional requirement, create a branch. **Note:** There should be at least nine branches excluding master.
- Commit messages reflect the context of each functional requirement change

Additional Resources

- Documenting Kotlin Code - https://kotlinlang.org/docs/reference/kotlin-doc.html
- Android Design - https://developer.android.com/design
- Adaptive Launcher Icon - https://developer.android.com/guide/practices/ui_guidelines/icon_design_adaptive

Assessment 01: Language Translator
Assessment Rubric

	10-9	8-7	6-5	4-0
Functionality & Robustness	Application thoroughly demonstrates functionality & robustness on the following: • Open without modification in Android Studio. This includes the removal of the gradle & idea hidden directories. • Run without modification on multiple mobile devices. • Text translation which includes support for at least four languages, excluding English. • Display a progress dialog while in translation. • Handle incorrect formatted input values. • Localization which includes support for at least four languages. • Exit the application via an alert dialog. • Text-to-speech using the TextToSpeech Android class or a third-party library. • Multiple mobile screen/device support. Include support for at least four screens/devices, excluding tablets & smartwatches devices. • Interactive quiz for each language. • Visually attractive user-interface with a coherent graphical theme and style. • Toggle between light & dark mode. • Splash screen with a transition animation. • Adaptive launcher icon. • Bottom navigation which navigates the user to the appropriate activities. • Application published to Google Play Store.	Programs mostly demonstrate functionality & robustness on the following: • Open without modification in Android Studio. This includes the removal of the gradle & idea hidden directories. • Run without modification on multiple mobile devices. • Text translation which includes support for at least four languages, excluding English. • Display a progress dialog while in translation. • Handle incorrect formatted input values. • Localization which includes support for at least four languages. • Exit the application via an alert dialog. • Text-to-speech using the TextToSpeech Android class or a third-party library. • Multiple mobile screen/device support. Include support for at least four screens/devices, excluding tablets & smartwatches devices. • Interactive quiz for each language. • Visually attractive user-interface with a coherent graphical theme and style.	Programs demonstrate functionality & robustness on some of the following: • Open without modification in Android Studio. This includes the removal of the gradle & idea hidden directories. • Run without modification on multiple mobile devices. • Text translation which includes support for at least four languages, excluding English. • Display a progress dialog while in translation. • Handle incorrect formatted input values. • Localization which includes support for at least four languages. • Exit the application via an alert dialog. • Text-to-speech using the TextToSpeech Android class or a third-party library. • Multiple mobile screen/device support. Include support for at least four screens/devices, excluding tablets & smartwatches devices. • Interactive quiz for each language. • Visually attractive user-interface with a coherent graphical theme and style.	Programs do not or do not fully demonstrate code elegance on any of the following: • Open without modification in Android Studio. This includes the removal of the gradle & idea hidden directories. • Run without modification on multiple mobile devices. • Text translation which includes support for at least four languages, excluding English. • Display a progress dialog while in translation. • Handle incorrect formatted input values. • Localization which includes support for at least four languages. • Exit the application via an alert dialog. • Text-to-speech using the TextToSpeech Android class or a third-party library. • Multiple mobile screen/device support. Include support for at least four screens/devices, excluding tablets & smartwatches devices. • Interactive quiz for each language. • Visually attractive user-interface with a coherent graphical theme and style.

	• Download application from Google Play Store on mobile device.	• Toggle between light & dark mode. • Splash screen with a transition animation. • Adaptive launcher icon. • Bottom navigation which navigates the user to the appropriate activities. • Application published to Google Play Store. • Download application from Google Play Store on mobile device.	• Toggle between light & dark mode. • Splash screen with a transition animation. • Adaptive launcher icon. • Bottom navigation which navigates the user to the appropriate activities. • Application published to Google Play Store. • Download application from Google Play Store on mobile device.	• Toggle between light & dark mode. • Splash screen with a transition animation. • Adaptive launcher icon. • Bottom navigation which navigates the user to the appropriate activities. • Application published to Google Play Store. • Download application from Google Play Store on mobile device.
Documentation	Step-by-step user guide thoroughly describes each activity/fragment screenshot. Application code is thoroughly commented & documented with KDoc/Dokka.	Step-by-step user guide thoroughly describes each activity/fragment screenshot. Most application code is commented & documented with KDoc/Dokka.	Step-by-step user guide describes each activity/fragment screenshot. Some application code is commented & documented with KDoc/Dokka.	Step-by-step user guide does not or does not fully describe each activity/fragment screenshot. Application code is not or is not fully commented & documented with KDoc/Dokka.
Code Elegance	All Kotlin files contain no magic numbers/strings. All XML files contain no magic numbers/strings. Note: use dimen.xml (resource directory) to store magic numbers. Application thoroughly demonstrates code elegance on the following: • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to various OO design principles.	Most Kotlin files contain no magic numbers/strings. Most XML files contain no magic numbers/strings. Note: use dimen.xml (resource directory) to store magic numbers. Application mostly demonstrates code elegance on the following: • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to various OO design principles.	Some Kotlin files contain no magic numbers/string. Some XML files contain no magic numbers/strings. Note: use dimen.xml (resource directory) to store magic numbers. Application demonstrates some code elegance on the following: • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to various OO design principles.	Some Kotlin files contain no magic numbers/strings. Some XML files contain no magic numbers/strings. Note: use dimen.xml (resource directory) to store magic numbers. Application does not or does not fully demonstrate code elegance on any of the following: • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to various OO

	• Adhere to a complex OO architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Efficient algorithmic approach.	• Adhere to a complex OO architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Efficient algorithmic approach.	• Adhere to a complex OO architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Efficient algorithmic approach.	design principles. • Adhere to a complex OO architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Efficient algorithmic approach.
Git Usage	Git commit messages thoroughly reflect the functional requirement changes. Git branches thoroughly named & describe the context of the functional requirements.	Git commit messages mostly reflect the functional requirement changes. Git branches mostly named & describe the context of the functional requirements.	Git commit messages reflect some of the functional requirement changes. Git branches named & describe some of the context of the functional requirements.	Git commit messages do not or do not fully reflect the context of each solution. Git branches incorrectly named & do not or do not fully describe the context of each solution.

Marking Cover Sheet

OTAGO POLYTECHNIC
Te Kura Matatini ki Otago

Assessment 01: Language Translator

IN721: Design & Development of Application for Mobile Devices
Level 7, Credits 15
Bachelor of Information Technology

Name: _____ Date: _____

Learner ID: _____

Assessor's Name: _____

Assessor's Signature: _____

Criteria	Out Of	Weighting	Final Result
Functionality & Robustness	**10**	**35**	
Documentation	**10**	**25**	
Code Elegance	**10**	**30**	
Git Usage	**10**	**10**	
		Final Result	**/100**
This assessment is worth 25% of the final mark for the Design & Development of Application for Mobile Devices course.			