1. Unit Testing

Unit testing verifies individual components of the program to ensure they function as expected.

Test Cases for Unit Testing

Test Case ID	Component	Test Description	Input	Expected Output	Result
UT-01	`get_language_code()`	Check if the correct language code is returned for a given language name.	"English"	"en"	Pass/Fail
UT-02	`translator_function()`	Verify that the correct translation is returned.	Text: "Hello", `src='en'`, `dest='fr'`	"Bonjour"	Pass/Fail
UT-03	`text_to_voice()`	Check if the text-to-speech conversion produces an audio file and plays it.	Text: "Bonjour", Language: "fr"	Audio file is generated, played, and deleted successfully.	Pass/Fail
UT-04	Speech Recognition	Validate that the program correctly converts speech to text.	Spoken: "Hello"	"Hello" (transcribed text)	Pass/Fail
UT-05	Streamlit UI Buttons	Verify if "Start" and "Stop" buttons trigger the appropriate translation loop.	Button clicks	Translation starts/stops appropriately.	Pass/Fail
UT-06	Main Process Loop	Test that the translation pipeline executes in the correct sequence.	Input speech: "Hello"	Translated text: "Bonjour", output audio in French plays.	Pass/Fail

2. White-Box Testing: White-box testing focuses on testing the internal logic and structure of the code.

White Box Test Cases				
Test Case ID	**Functionality**	**Test Description**	**Expected Behavior**	**Result**
WB-01	`get_language_code()`	Test all conditional branches for language name mapping.	For valid language names: return language code; for invalid ones: return the input unchanged.	Pass/Fail
WB-02	`main_process()`	Verify the loop correctly transitions between states (listening, processing, etc.).	The loop cycles correctly through **listening**, **processing**, **translating**, and **text-to-speech** phases without breaking or infinite looping.	Pass/Fail
WB-03	Exception Handling	Test error handling for failed speech recognition (e.g., no microphone input).	Catch the error and display a meaningful error message to the user (e.g., "Microphone not detected").	Pass/Fail
WB-04	File Handling	Check that temporary audio files are created and deleted as expected.	Files (e.g., `cache_file.mp3`) are created during playback and removed after playback ends.	Pass/Fail
WB-05	Integration of Modules	Ensure the integration between speech recognition, translation, and TTS works.	Output from one module (e.g., recognized speech) is correctly passed to the next (e.g., translation, then TTS) without mismatched data or dropped processes.	Pass/Fail

3. Integration Testing

Integration testing ensures that individual modules work together as intended.

Integration Test Cases				
Test Case ID	**Modules Involved**	**Test Description**	**Expected Behavior**	**Result**
IT-01	Speech Recognition → Translation	Test the flow of data from speech recognition to translation.	Speech input is transcribed accurately and passed to the translation module.	Pass/Fail
IT-02	Translation → Text-to-Speech	Test the flow of translated text to text-to-speech conversion.	Translated text is accurately converted into audio and played.	Pass/Fail
IT-03	UI → Main Process Loop	Verify that user actions (Start/Stop) interact correctly with the main process loop.	Clicking "Start" begins translation, and clicking "Stop" ends the process gracefully.	Pass/Fail

---

4. Black-Box Testing

Black-box testing focuses on inputs and outputs without examining internal logic.

Black Box Test Cases				
Test Case ID	**Test Description**	**Input**	**Expected Output**	**Result**
BB-01	Test translation accuracy for common phrases.	Input speech: "Hello"	Audio output: "Bonjour" (French translation).	Pass/Fail
BB-02	Verify handling of unsupported languages.	Input language: "XYZ"	Error message displayed: "Unsupported language."	Pass/Fail
BB-03	Test response to no speech input.	Silence for 10 seconds	Error message: "No speech detected, please try again."	Pass/Fail
BB-04	Test system with a long phrase.	Input: "How are you today?"	Accurate translation and audio output within reasonable delay.	Pass/Fail

---

5. Performance Testing

Evaluate the performance of the system in terms of speed and resource utilization.				
Test Case ID	**Test Description**	**Metric**	**Expected Result**	**Result**
PT-01	Measure average translation time.	Time from input to audio output.	≤ 5 seconds for speech input of 10 seconds.	Pass/Fail
PT-02	Test memory usage during the process.	Memory consumed during playback.	≤ 50 MB additional memory usage during operation.	Pass/Fail

6. User Acceptance Testing (UAT)

Verify that the system meets user needs in a real world environment.			
Test Case ID	**Scenario**	**Expected Outcome**	**Result**
UAT-01	A user speaks in English to get a French translation.	The system outputs an accurate French translation, both as text and audio.	Pass/Fail
UAT-02	User selects unsupported languages.	An error message is displayed, and the system does not crash.	Pass/Fail
UAT-03	User clicks "Stop" during a translation.	The translation process halts immediately without crashing or leaving files behind.	Pass/Fail

---

7. Regression Testing

Ensure that new updates or changes to the code do not introduce new bugs or break existing functionality.

Test Plan:

- Re-run all unit, integration, and performance tests after any major code changes.
- Focus on areas where changes have been made (e.g., if speech recognition is updated, re-test UT-04, WB-02, and IT-01).

---

8. Error Handling Tests

Test Cases for Error Handling			
Test Case ID	**Error Scenario**	**Expected Behavior**	**Result**
EH-01	Microphone not detected.	Error message: "Microphone not detected. Please check your device."	Pass/Fail
EH-02	Internet connectivity lost.	Error message: "Internet connection is required for translation and TTS. Please reconnect."	Pass/Fail
EH-03	Unsupported language selected.	Error message: "Unsupported language."	Pass/Fail

---

9. Automation Testing

Develop scripts using tools like **PyTest** or **Unittest** to automate:

- Unit test execution.
- Regression testing after every major update.
- Performance testing for measuring response times.

---

10. Summary

This comprehensive testing strategy ensures:

- Each module (e.g., speech recognition, translation, TTS) works correctly.
- Integration between modules is seamless.
- Errors are handled gracefully.
- Performance and usability meet user expectations.