2024

EchoText

Testing Plan
Version 1.01

Team: Ahmad Ghaddar, Kevin Xing, Harsh Bhagat, Andy Huang

Revision History

Date	Description	Author	Comments
11/06/24	Version 1.00	Andy Huang	First Revision
12/3/24	Version 1.01	Andy Huang	Minor changes in each tables on each section.

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date
	Kevin Xing	Team Lead Meeting Minutes	
	Ahmad Ghaddar	Presentation Lead Frontend Lead UI/UX Lead	
	Harsh Bhagat	GitHub Lead Backend and Data Management Lead	
	Andy Haung	Documentation Lead Testing/QA Lead	

Table of Contents

REVISION HISTORY ..... **I**

DOCUMENT APPROVAL ..... **I**

1. INTRODUCTION **1**

1.1 Purpose 1
1.2 Scope 1
1.3 Intended Audience 2
1.4 References 2

2. TEST DELIVERABLES **3**

3. TEST DATA **3**

3.1 Accessing Test Data 3
3.2 Data Usage Guidelines 3

4. TEST SCHEDULE **3**

5 Defect Management 5.1 Defect Tracking Process 4
5.2 Defect Classification 4
5.3 Defect Resolution Priorities 5
5.4 Defect Reporting Requirements 5

6. ROLES AND RESPONSIBILITIES **6**

7. UNIT TESTING **7**

7.1 Approach 7
7.2 Pass/Fail Criteria 7
7.3 Entry/Exit Criteria 7
7.4 Suspension/Resumption Criteria 7
7.5 Risks/Issues 7
7.6 Test Design/Execution 8
7.7 Items to be Tested 8
7.7.1 Database Class Unit Test Items *8*
7.7.2 TextParser Class Unit Test Items *10*
7.7.3 PasswordManager Class Unit Test Items *10*
7.7.4 CombineAudioFiles Function Unit Test Items *11*
7.7.5 TtsLogic Class Unit Test Items *11*
7.7.6 FileDirectory Class Unit Test Items *11*
7.7.7 VoiceProfileManager Class Unit Test Items *12*
7.7.8 Basic UI Unit Test Items *12*

8. INTEGRATION TESTING **13**

8.1 Approach 13
8.2 Pass/Fail Criteria 13
8.3 Entry/Exit Criteria 13
8.4 Suspension/Resumption Criteria 13
8.5 Risks/Issues 14
8.6 Test Design/Execution 14
8.7 Items to be Tested 14
8.7.1 Text Processing and TTS Integration Tests: *14*
8.7.2 Audio System Integration Tests: *15*
8.7.3 Settings and Profile Integration Tests: *15*
8.7.4 File Management Integration Tests: *15*
8.7.5 User Interface Integration Tests: *16*

9. SYSTEM TESTING 16

9.1 APPROACH 16
9.2 PASS/FAIL CRITERIA 17
9.3 ENTRY/EXIT CRITERIA 17
9.4 SUSPENSION/RESUMPTION CRITERIA 17
9.5 TEST DESIGN/EXECUTION 18
9.6 RISKS/ISSUES 19
9.7 ITEMS TO BE TESTED 19

10. USER ACCEPTANCE 20

1. Introduction

1.1 Purpose

This test plan specification document is to outline the scope, approach, and objectives for testing the system to ensure that it meets both the functional and none-functional requirements. It serves as a guideline to help define test cases. By identifying and validating the critical components and systems, this document will help ensure that the application's quality before release.

1.2 Scope

In Scope:
The integration testing will cover the interactions and data flow between major components of the EchoText application. This includes:
Core Functionalities:

- **Text-to-Speech (TTS):** Conversion of text to speech, including live playback and batch document processing.
- **File Management:** Importing documents and managing audio files, including renaming, deletion, and persistence with linking and database updates on changes.
- **User Interface (UI):** Verification of navigation, settings, and accessibility of all user-facing features.

Testing levels:

- **Unit Testing:** Validating individual components (e.g., generate button, clear text button, singular playback button).
- **Integration Testing:** Validating multi component features such as TTS inference, queue system, playback controls, and managing audio files.
- **System Testing:** Comprehensive validation of application performance, reliability, and user experience.

Security and Customization:

- **User Settings:** Testing customizable voice profiles, TTS parameters, and application themes.
- **Security Features:** Ensuring password protection and data integrity.
- **Performance and Stability:**
- Stress-testing with large files and resource constraints to ensure responsiveness, reliability, and adherence to performance benchmarks.

Out of Scope:
The following items are not included in integration testing:

- **Cross-Platform Testing:** Compatibility testing on platforms outside the defined iOS scope (e.g., Android).
- **Hardware Variability:** Device-specific testing outside supported iOS versions or hardware configurations. iPhones using iOS version 16.0 and below will not be tested.

- **Unplanned Feature Development:** Features not outlined in the project requirements or traceability matrix.

1.3 Intended Audience

This document is intended for the following stakeholders involved in the development, testing, and validation of the EchoText application:

Development Team:

- Understand the scope, testing approach, and deliverables to ensure alignment with project requirements.
- Use test results to identify and resolve issues in functionality, integration, and performance.

Quality Assurance (QA) Team:

- Implement the testing plan by executing test cases, documenting outcomes, and reporting defects.
- Validate that the application meets defined functional and non-functional requirements.

Project Managers:

- Monitor testing progress and ensure timely resolution of defects.
- Use this document to track adherence to project timelines and quality standards.

Graduate Teaching Assistant (GTA):

- Evaluate the application's readiness and provide final approval during User Acceptance Testing.
- Offer feedback on critical features, usability, and overall application quality.

Stakeholders/End Users:

- Indirectly benefit from the testing process ensuring the application is reliable, user-friendly, and meets expectations.

This section ensures that all relevant parties understand their roles in the testing process and how the document supports successful project delivery.

1.4 References

The test cases referenced in this Testing Plan document maintain direct traceability to the use cases defined in the EchoText Design Document. Each test case (both Unit Tests UT1-UT64 and Integration Tests IT1-IT10) corresponds to specific functional requirements outlined in the EchoText Software Requirements Specification (SRS). The relationships between test cases, use cases, and requirements are documented in detail within the traceability matrix to ensure comprehensive test coverage of all system functionalities.

Additional reference documents:

- EchoText Software Requirements Specification
- EchoText Design Document
- EchoText Test Case Specification

2. Test Deliverables

The following deliverables will be produced and maintained throughout the testing process:

Test Plan Document:

- Details the testing strategy, scope, objectives, and approach for all testing phases.

Test Case Specification:

- Provides detailed descriptions of test cases, including inputs, expected results, and conditions for unit, integration, and system testing.

Traceability Matrix:

- Maps functional and non-functional requirements to corresponding test cases to ensure complete coverage.

Test Data:

- Includes predefined input data sets used for testing various scenarios (e.g., valid, invalid, edge cases).

Application:

- The build of the EchoText application is to be tested at each phase.

Test Scripts:

- Automated or manual scripts are used to execute test cases efficiently during each testing phase.

3. Test data

3.1 Accessing Test Data

All data needed for testing will be provided by the development team:

- Basic test data such as simple passcodes and text inputs will be included directly within the test cases or test scripts.
- Media test data, include audio files and pdf files, will be available under the “Test Data” section of our GitHub repository. They can be accessed via the following link:
    - https://github.com/Harsh-59/AI-Voices/tree/main/Test%20Data

  Media test data can also be provided via a USB stick upon request with the testing lead.

3.2 Data Usage Guidelines

- Confidentiality: All test data should be treated as confidential and used solely for testing purposes within the scope of this project.
- Integrity: Do not alter the test data files. If modifications are necessary for a specific test case, document the changes and inform the testing team.

4. Test Schedule

The testing schedule outlines when each phase of testing will begin and its dependencies:

Unit Testing:

- **Start Date**: 11/26/2024, after the completion of individual module development.
- **Dependency**: Requires all module code to be implemented and builds to pass static analysis.

Integration Testing:

- **Start Date**: 11/29/2024, immediately after unit testing for all modules is completed.
- **Dependency**: Requires all unit tests to pass successfully, and integrated modules must be stable.

System Testing:

- **Start Date**: 12/01/2024, after integration testing concludes.
- **Dependency**: Requires the full EchoText application to be built and stable after integration.

5 Defect Management

5.1 Defect Tracking Process

All defects discovered during testing will be documented using GitHub Issues. Upon discovery of a defect, testers must include steps to reproduce, expected results, actual results, and screenshots or logs where applicable.
A defect lifecycle will follow:

1. **New:** Defect is reported
2. **Open:** Defect is validated and assigned
3. **In Progress:** Developer is working on fix
4. **Fixed:** Developer has implemented fix
5. **Testing:** Fix is being validated
6. **Closed:** Fix is verified and approved
7. **Rejected:** Not considered a defect

5.2 Defect Classification

Defects will be classified according to the following severity levels:
Critical:

- Application crashes or becomes unusable
- Data loss or corruption
- Security vulnerabilities
- Core functionality failures (TTS conversion, audio playback)
- System deadlocks

High:

- Major feature malfunction
- Significant performance issues
- UI elements preventing feature usage
- Incorrect audio file generation
- Database synchronization issues

Medium:

- Minor feature limitations
- UI inconsistencies
- Non-critical performance issues
- Inconvenient but workable issues
- Non-critical functionality issues

Low:

- Cosmetic issues
- Minor UI improvements
- Documentation inconsistencies
- Enhancement requests
- Optional feature issues

5.3 Defect Resolution Priorities

Resolution priorities will be assigned based on severity and impact:

Priority 1 (Immediate):

- Must be fixed immediately
- Blocks testing progress
- Affects critical functionality
- All Critical severity defects

Priority 2 (High):

- Must be fixed before release
- High severity defects
- Affects major functionality
- Important user experience issues

Priority 3 (Medium):

- Fix required but not urgent
- Medium severity defects
- Minor functionality issues
- Can be worked around

Priority 4 (Low):

- Fix if time permits
- Low severity defects
- Cosmetic issues
- Enhancement requests

5.4 Defect Reporting Requirements

Each defect report must include:

- Clear and concise description
- Steps to reproduce
- Expected vs actual results
- Test environment details

- Screenshots or logs (if applicable)
- Severity and priority assessment
- Impact on functionality
- Related test case reference

6. Roles and Responsibilities

This section outlines the key roles and responsibilities for the EchoText testing team. Each team member has been assigned specific areas of focus to ensure comprehensive testing coverage across all aspects of the application. The responsibilities defined here establish accountability and provide a framework for effective communication and collaboration throughout the testing process.
Roles:

Test Lead (Andy Huang)

- Assign tests to each team member
- Review test results and documentation
- Track and manage defects
- Assign development tasks for failed tests
- Coordinate testing efforts among team members
- Maintain test documentation
- Ensure test deadlines are met
- Execute test cases as needed to support team workload
- Implement fixes for failed tests when necessary

Tester/Developer (Ahmad Ghaddar)

- Execute assigned test cases
- Document test results
- Report defects found during testing
- Implement fixes for failed tests as assigned
- Support UI/UX testing efforts

Tester/Developer (Kevin Xing)

- Execute assigned test cases
- Document test results
- Report defects found during testing
- Implement fixes for failed tests as assigned
- Support performance testing efforts

Tester/Developer (Harsh Bhagat)

- Execute assigned test cases
- Document test results
- Report defects found during testing
- Implement fixes for failed tests as assigned
- Support backend testing efforts

7. Unit Testing

7.1 Approach

Unit testing will be performed to verify the functionality and correctness of individual units of code, primarily functions and classes, in isolation from the other classes or the rest of the application. A white box testing approach will be employed, allowing testers to design test cases based on knowledge of the internal structures and implementation of the code. Tests will be written for critical and error-prone components to ensure that they are functioning correctly before proceeding to integration tests. Unit tests will be performed using XCTests with mock frameworks or dependencies if needed for isolation.

7.2 Pass/Fail Criteria

A unit test will pass if the function or class behaves as expected, producing the correct output from a given input. A unit test will fail if an error occurs or the test crashes during testing, if it produced a wrong result, or if the performance falls below a pre-determined threshold. Overall unit tests criteria are to achieve a 100% passing result. If any fails, testing will be terminated before the error is fixed and testing will be restarted from UT01.

7.3 Entry/Exit Criteria

Entry Criteria:

- Code implementation is complete for each module or function to be tested.
- Test cases are defined and reviewed.
- Test data is prepared, covering typical and edge cases.
- Testing environment and tools are set up and accessible.

Exit Criteria:

- All planned unit tests have been executed with passing results.
- Defects identified during unit testing have been resolved or documented.
- Test results have been reviewed and approved by the testing lead.

7.4 Suspension/Resumption Criteria

Suspension Criteria:

- Unit testing will be suspended if critical dependencies (e.g., necessary libraries or testing frameworks) are unavailable or faulty.

Resumption Criteria:

- Unit testing will resume from the start once dependencies are restored or updated, and all blocking defects have been resolved.

7.5 Risks/Issues

Incomplete Coverage:

- Some less critical or trivial functions might not be tested, potentially leaving gaps or leading to errors in future tests.

Dependency Management:

- For functions or classes that inherently work with other components, the difficulty or time constraint in setting up mocks or dependency injection might hinder the ability to test units in isolation.

7.6 Test Design/Execution

Unit tests are designed using the white-box testing methodology, focusing on the internal logic of the code. The tests are categorized based on the components they cover:

- Database class (UT01 ~ UT27)
- TextParser class (UT28 ~ UT34)
- PasswordManager class (UT35 ~ UT41)
- combineAudioFiles function (UT42)
- TtsLogic class (UT43 ~ UT51)
- FileDirectory class (UT52 ~ UT56)
- VoiceProfileManager class (UT57 ~ UT 64)
- Basic UI navigation (UT65 ~ UT70)

Tests related directly the code, e.g. classes and functions (UT01 ~ UT64), will be executed by running the test script targeting the specific class or function. For those tests, the testers will:

1. Open the project in Xcode
2. Select the appropriate test target (e.g., DatabaseTests, TextParserTests, etc)
3. Run the test suite using “Command + U” or via the test navigator.
4. Observe the test results, ensuring all tests pass.
5. Document any failures or anomalies for further investigation.

Tests related to the UI (UT65 ~ UT70) will be performed manually to validate the functionality of the UI components.

7.7 Items to be Tested

7.7.1 Database Class Unit Test Items

ID	Title	Test Description
UT01	Database Initialization and Connection	Test that the database opens successfully on initialization.
UT02	Database Initialization and Connection	Test that creating tables works correctly (Audios, ModelProfiles, Documents).
UT03	Database Deletion	Test that the deleteDatabase method removes the database file as expected.

UT04	Database Deletion	Test for cases where deletion fails (e.g., no record present).
UT05	Insert Operations - Audios	Ensure that inserting an audio record saves all fields accurately.
UT06	Insert Operations - Documents	Validate that inserting a document record saves all fields correctly.
UT07	Insert Operations - ModelProfiles	Check that profile insertion captures all fields accurately.
UT08	Update Operations - Audios	Test updates to audio records by modifying each field independently.
UT09	Update Operations - Documents	Ensure document updates correctly modify each field.
UT10	Update Operations - ModelProfiles	Confirm that profile updates modify fields accurately.
UT11	Fetch Operations - Audios	Verify that fetching all audio records returns correct data in the expected format.
UT12	Fetch Operations - Documents	Test that fetching documents returns expected data.
UT13	Fetch Operations - Profiles	Check that all model profiles are returned accurately.
UT14	Delete Operations - Audios	Verify that an audio record can be deleted by ID.
UT15	Delete Operations - Documents	Ensure that deleting a document record works by ID.
UT16	Delete Operations - Profiles	Confirm that a profile is deleted by name.
UT17	Edge Cases - Non-existent Record Deletion	Attempt to delete a record by an ID or name that does not exist.
UT18	Edge Cases - Empty Inserts	Test inserting empty values for required fields to ensure constraints are enforced.
UT19	Edge Cases - Database Not Initialized	Simulate cases where database operations are called without initializing.
UT20	Linking Records - Audio to Document	Verify linking an audio record to a document updates the DocumentId field correctly.
UT21	Linking Records - Invalid Document ID	Test linking an audio record to a non-existent document ID.
UT22	Profile Management - Rename Profile	Test renaming a profile, including error cases like duplicate names.
UT23	Profile Management - Load Profile by Name	Check that loading a profile by name retrieves accurate data.
UT24	Profile Management - Profile Not Found	Test loading a profile with a name that doesn’t exist.
UT25	Audio Length Calculation	Verify that getAudioLength correctly calculates and returns the length of valid audio files.

UT26	Audio Length Calculation - Invalid Path	Test handling of invalid or nonexistent file paths.
UT27	Concurrency Handling	Simulate concurrent accesses to test data consistency across multiple operations.

7.7.2 TextParser Class Unit Test Items

ID	Title	Test Description
UT28	Initialization	Verify TextParser initializes correctly with an InputStream.
UT29	Start and Stop Parsing	Ensure startParsing begins parsing on a background thread.
UT30	Start and Stop Parsing	Test that stopParsing sets terminateFlag to true, stopping parsing.
UT31	Chunk Emission - First Chunk	Verify that the first chunk ends at the first comma, period, or 50 words.
UT32	Chunk Emission - Subsequent Chunks	Confirm that subsequent chunks are emitted according to the set word limits.
UT33	Delegate Call - Parsing Completion	Verify textParserDidFinish is called with the correct total chunk count when parsing completes.
UT34	Delegate Call - Parsing Termination	Confirm textParserDidTerminate is called with the total word count when parsing is terminated.

7.7.3 PasswordManager Class Unit Test Items

ID	Title	Test Description
UT35	Initialization	Verify PasswordManager initializes correctly, checking for an existing password and recovery code.
UT36	Set Password	Test that setPassword stores the password and generates a recovery code.
UT37	Remove Password	Confirm removePassword clears the stored password and recovery code.
UT38	Validate Password	Verify validatePassword correctly matches input against the stored password.
UT39	Recover Account	Ensure recoverAccount validates the recovery code and allows resetting or removing the password.
UT40	Generate Recovery Code	Test that generateRecoveryCode produces a valid six-digit code.
UT41	Finalize New Password	Verify finalizeNewPassword sets the new password if it matches the temporary password.

7.7.4 CombineAudioFiles Function Unit Test Items

ID	Title	Test Description
UT42	Stitch Audios	Verify combineAudioFiles works and that the result audio file is correct.

7.7.5 TtsLogic Class Unit Test Items

ID	Title	Test Description
UT43	Initialization	Verify TtsLogic initializes correctly with dependencies (Database, FileDirectory, VoiceProfileManager).
UT44	Generate Speech Streaming	Test that generateSpeechStreaming processes text into chunks and triggers TTS generation.
UT45	Stop Speech	Confirm stopSpeech halts text parsing, clears queues, and resets generation states.
UT46	Audio Chunk Handling	Ensure audio chunks are enqueued properly into AVQueuePlayer during streaming playback.
UT47	Playback Completion	Verify playerDidFinishPlaying handles the end of playback and clears the queue correctly.
UT48	Text Parsing Delegate	Confirm textParser(_:didProduceChunk:) processes chunks correctly for TTS generation.
UT49	Text Parsing Completion	Test textParserDidFinish(_:totalChunks:) triggers audio stitching and database updates.
UT50	Change Model	Ensure changeModel updates the active model and resets the TTS instance if not generating.
UT51	Reset	Test reset clears all states, reinitializes TTS, and resets progress.

7.7.6 FileDirectory Class Unit Test Items

ID	Title	Test Description
UT52	Initialization	Verify FileDirectory initializes correctly and its properties return valid directory URLs.
UT53	Directory Creation	Test that createApplicationSupportDirectories creates the database, documents, and audiofiles directories.
UT54	Directory Existence	Confirm that createApplicationSupportDirectories does not throw errors if directories already exist.
UT55	Directory Permissions	Ensure the created directories are writable and accessible.
UT56	Error Handling	Verify appropriate error handling when directory creation fails (e.g., insufficient permissions).

7.7.7 VoiceProfileManager Class Unit Test Items

ID	Title	Test Description
UT57	Initialization	Verify VoiceProfileManager initializes with a default profile when no saved profile exists.
UT58	Initialization	Ensure the correct profile is loaded if a saved profile name is found in UserDefaults.
UT59	Profile Switching	Test switchToProfile correctly updates the active profile and saves changes to the previous profile.
UT60	Add Profile	Verify addProfile creates a new profile and switches to it.
UT61	Rename Profile	Confirm renameProfile changes the name of an existing profile and updates UserDefaults if active.
UT62	Delete Profile	Test deleteProfile removes a profile and switches to the default profile if the active one is deleted.
UT63	Profile Updates	Ensure changes to pitch, speed, or model are saved to the database automatically via observers.
UT64	Prevent Default Profile Changes	Verify that the "Default" profile cannot be deleted or renamed.

7.7.8 Basic UI Unit Test Items

ID	Title	Test Description
UT65	Generated Tab Navigation Link	Test navigation to the Generated screen via the bottom tab bar.
UT66	Documents Tab Navigation Link	Test navigation to the Documents screen via the bottom tab bar.
UT67	Text History Tab Navigation Link	Test navigation to the Text History screen via the bottom tab bar.
UT68	Settings Tab Navigation Link	Test navigation to the Settings screen via the bottom tab bar.
UT69	Main Tab Navigation Link	Test navigation to the Main screen via the bottom tab bar from the Settings tab.
UT70	Support Navigation Link in Settings	Test navigation to the Support screen via the Settings page.

8. Integration Testing

8.1 Approach

The integration testing approach for EchoText will focus on verifying the interactions between major components of the system, ensuring they work together seamlessly. Combining multiple functions into one test case, this testing phase specifically targets the interfaces and interactions between components as issues often arise when data is passed between different parts of the system. The tester will need to deploy an IOS device simulator and manually run each test to ensure that each test passes without issues. We will employ white box testing which will allow us to examine the internal logic and data flow between integrated components, including the inspection of code paths, data structures, and component interfaces.

8.2 Pass/Fail Criteria

A test will be considered successful only when all outcomes precisely match the predefined expectations outlined in the test case documentation. Any deviation from the expected results, whether through omission of expected behavior or failure to meet specified conditions, will result in a test failure. If any unexpected or undefined behaviors occur during testing (even if they do not directly contradict expected results), it will require an evaluation. This evaluation process will determine whether the test case requirements need to be revised to accommodate valid but previously undocumented behaviors, or if the unexpected behavior constitutes a test failure requiring system modification. The final determination of pass or fail status will be made only after this review process is complete and all behaviors are properly documented and validated.

8.3 Entry/Exit Criteria

Entry Criteria:

- Before integration testing can begin, all individual components must have successfully passed their unit tests. The test environment must be properly configured with all necessary tools available, and test data must be prepared and reviewed.

Exit Criteria:

- Integration testing will be considered complete only when all planned test cases have been executed and no critical or high-severity defects remain open. All component interfaces must function according to specifications, with performance requirements met during integration testing.

8.4 Suspension/Resumption Criteria

Suspension Criteria:

- Testing will be suspended if a critical issue that block further testing are discovered. Some examples are if the iPhone simulator issues prevent reliable testing, test data corruption, or physical hardware/device failures which can impact testing capacity.

Resumption Criteria:

- Testing may resume once all critical issues related to the suspension have been resolved and verified.

8.5 Risks/Issues

Incomplete Coverage:

- Some less critical or trivial functions might not be tested, potentially leaving gaps or leading to errors in future tests.

Dependency Management:

- For functions or classes that inherently work with other components, the difficulty or time constraint in setting up mocks or dependency injection might hinder the ability to test units in isolation.

8.6 Test Design/Execution

Integration tests are designed using the white box testing methodology, focusing on the interaction between components. The tests are categorized based on the functional flows they validate:

- Text Processing and TTS Integration Tests (IT01~IT02)
- Audio System Integration Tests (IT03~IT04)
- Settings and Profile Integration Tests (IT05~IT07)
- File Management Integration Tests (IT08~IT09)
- User Interface Integration Tests (IT10)

Tests will be performed manually as they require observation of component interactions and real-time behaviors. For each test, the testers will:

1. Launch EchoText on a test device (iPhone XR or newer)
2. Navigate to the appropriate section of the app
3. Execute the test steps as documented in the test case
4. Verify all expected results occur
5. Document any deviations from expected behavior
6. Record test results including screenshots where relevant

Some integration tests may require specific setup conditions (e.g., multiple audio files for queue testing, PDFs for batch processing). Testers must ensure all prerequisites are met before beginning each test sequence.

8.7 Items to be Tested

8.7.1 Text Processing and TTS Integration Tests:

ID	Title	Test Description
IT01	Text-to-Speech Conversion	Test the process of converting input text to speech, verifying that text input is processed correctly, TTS engine

		generates audio, live playback works during conversion.
IT02	Document Conversion	Tests the conversion of a document to speech. Document parsing extracts text, text processor handles the content, TTS engine generates audio, live playback works during conversion.

8.7.2 Audio System Integration Tests:

ID	Title	Test Description
IT03	Media Control	Verify the integration between the audio playback system and the user interface controls. Test includes play, pause, seek, and volume control functionality.

8.7.3 Settings and Profile Integration Tests:

ID	Title	Test Description
IT04	Voice Profile Creation	Verify that new voice profiles can be created with custom settings and are properly stored in the system.
IT05	Passcode Creation	Verify the passcode creation process, including validation of matching entries and recovery code generation.
IT06	Passcode Validation	Test the passcode validation system when accessing the app after minimize or close
IT07	Recovery Code Functionality	Validate the recovery code system for regaining access when passcode is forgotten.
IT08	Passcode Removal	Test the secure removal of passcode protection through settings.

8.7.4 File Management Integration Tests:

ID	Title	Test Description
IT09	Audio Search Functionality	Tests search feature in generated view. UI captures search input, search processor handles query, database executes search, and results display in UI.
IT10	Document Search Functionality	Tests search feature in Document view. UI captures search input, search processor handles query, database executes search, and results display in UI.
IT11	Document Import Process	Test the document import functionality by importing multiple PDFs and verifying they

		are correctly processed and stored with accurate metadata.
IT12	File Rename and Delete	Validate individual file operations including renaming files and deleting single files, ensuring database consistency and proper file system updates.
IT13	Batch File Deletion	Test the multi-select functionality and batch deletion capability, ensuring multiple files can be selected and deleted simultaneously.
IT14	File Sharing Functionality	Verify the file sharing capabilities through iOS share sheet, ensuring files can be shared through different methods while maintaining file integrity and playback functionality.

8.7.5 User Interface Integration Tests:

ID	Title	Test Description
IT15	Progress Indication System	Verify that progress indicators accurately reflect the status of background operations such as TTS processing and batch processing of documents.

9. System Testing

9.1 Approach

System testing for EchoText will utilize a white box testing methodology, focusing on the internal logic, structural integrity, and data flow between components. This approach ensures comprehensive validation of the application's functionality by directly examining the code and its execution paths.
White-box testing will focus on:

- Verifying all code paths, including edge cases and exception handling.
- Inspecting the flow of data between modules such as TTS, file management, and user settings.
- Ensuring algorithmic correctness for queue management, audio generation, and other core features.
- Testing internal interactions between integrated components to identify bottlenecks and logical errors.

9.2 Pass/Fail Criteria

Each test case will have clear rules to decide if its "Passes" or "Fails" based on how the app performs.

- A test will Pass if the app works as expected and matches the description in the test case.
- A test will Fail if there is any issue or error that stops a feature from working as it should.

Criteria for Pass:

- The feature works exactly as described in the test case.
- The app runs smoothly without crashing or showing serious errors.
- Buttons, menus, and other UI elements work properly and respond correctly.
- The audio playback is clear, and the TTS feature generates understandable speech.

Criteria for Fail:

- The feature does not work as intended, or the UI elements are broken or incorrect.
- The app crashes, freezes, or shows major errors during use.
- The TTS output is distorted, unclear, or incorrect.
- This ensures that the app works as it should and delivers a good user experience.

9.3 Entry/Exit Criteria

Entry Criteria

We will start system testing when these conditions are met:

- The app has successfully passed earlier tests (unit and integration testing).
- All main features (TTS generation, playback, UI controls, file management) are built and ready for testing.
- The app is stable enough to run without major problems that stop testing.
- The testing environment (devices and settings) is prepared and ready to use.

Exit Criteria

We will finish system testing when these conditions are met:

- All important test cases have passed, especially the ones marked critical or medium priority.
- There are no serious bugs or issues left unfixed.
- Any small problems have been noted and scheduled for future fixes.
- The app is stable and performs as expected.

9.4 Suspension/Resumption Criteria

Suspension Criteria

Testing might be paused if any of these problems happen:

- **Critical Failure:** A major bug makes the app unusable or stops testing completely (e.g., the app crashes on startup).
- **Environment Issue:** Problems with the testing setup, like broken devices or unavailable resources.

- **Dependency Failure:** Something the app relies on, like a cloud service, is not working or unavailable.

Resumption Criteria

Testing will start again when these issues are fixed:

- The problem that caused the pause has been resolved, and the app is stable.
- The testing setup is working properly again.
- Any required external services or dependencies are back online and functioning.

9.5 Test Design/Execution

System tests are categorized based on the functionalities they validate:

1. **Core Application Functionalities (ST01):**
    - Validate the TTS generation process, ensuring text input (manual, copy-paste, document import) is processed correctly and audio playback functions as expected.
    - Inspect internal data handling during batch document processing and verify playback controls operate without errors.
2. **File and Queue Management (ST02):**
    - Verify file operations like importing, renaming, deleting, and exporting while ensuring database consistency.
    - Test queue management logic for operations such as adding, deleting, and reordering files.
3. **User Settings and Customization (ST03):**
    - Test the implementation of user preferences such as voice profiles, TTS parameters, and themes.
    - Inspect the persistence mechanism for user settings to ensure preferences are retained across sessions.
4. **Performance and Stability (ST04):**
    - Monitor the application's behavior under resource constraints, such as low memory or processing large files.
    - Validate the app’s performance under prolonged usage and ensure stability during concurrent operations.

Execution Steps

For system tests focusing on internal code flow and interactions (ST01 ~ ST04), testers will:

Setup:

- Deploy the latest EchoText build on a test device or simulator (e.g., iPhone XR or newer).
- Prepare necessary dependencies, such as mock databases, APIs, and test data.

Execution:

- Use Xcode debugging tools to step through code paths during test execution.
- Validate logical conditions, loops, and data transformations in the code.
- Simulate edge cases like large file sizes, rapid user interactions, and invalid inputs.

Documentation:

- Record the execution path for each test case, noting any unexpected behaviors or failed conditions.
- Log any bugs with detailed reproduction steps and screenshots/logs where applicable.

9.6 Risks/Issues

Risks

- **Incomplete Features:** Some features might not be fully developed, causing delays in testing.
- **Device Compatibility:** The app might not work well on all devices or operating systems, leading to problems like playback issues or broken UI.
- **Performance Problems:** The app might slow down or struggle with large or complex text during TTS generation or playback.
- **Dependency Issues:** Services the app relies on, like third-party TTS engines, might fail or be unavailable, affecting the app's functionality.

Issues

- **Hard-to-Find Bugs:** Some bugs might only happen in specific situations, making them tricky to find and fix.
- **Test Data Limits:** Not having enough test data might make it harder to check edge cases or unique user scenarios.
- **Regression Problems:** Features that used to work might break after updates or changes to the code.
- **User Experience Problems:** Some parts of the app might be confusing or hard for users to understand, causing usability issues during testing.

9.7 Items to be Tested

ID	Title	Test Description
ST01	Core Application Functionalities	Validate the core functionalities of the EchoText application, focusing on text-to-speech (TTS) generation. Ensure users can input text (manual, copy-paste, and document import), generate audio files, and listen to playback. Verify that generated audio is saved in the "Generated Audio" list for future access. Validate the ability to process batch document conversions. Ensure that playback controls (play, pause, rewind, fast-forward, and speed adjustment) operate correctly without errors.

ST02	File and Queue Management	Ensure the proper functionality of file management operations, including importing files, renaming, deleting, and exporting generated audio files. Verify queue management operations such as adding, deleting, and reordering files. Validate file persistence and bookmarking functionality after the application restarts. Test file operations for large files (up to 100 MB).
ST03	User Settings and Customization	Validate the functionality of user settings, including customizing voice profiles, adjusting TTS parameters like pitch and speed, and toggling between light and dark themes. Verify that settings changes persist across sessions and can be reset to defaults. Ensure security features like password protection work as expected. Confirm all settings sections and menus are accessible and navigable.

10. User Acceptance

The User Acceptance Testing for EchoText will be conducted through a formal meeting with the Graduate Teaching Assistant (GTA). This meeting serves as the final validation of the application's functionality and features. The objective of the meeting is to demonstrate all key features of EchoText, validate that the application meets GTA requirements, obtain formal approval from the GTA, document any feedback or suggestions, and address any concerns or questions.