SOFTWARE PROJECT FINAL REPORT

Brianne Kelley

Ke Ren

Date: 12/5/2024

Table of Contents

4.1. Architectural style(s) used

4.2. Architectural model (includes components and their interactions)

4.3. Technology, software, and hardware used

4.4. Rationale for your architectural style and model

5. Design

5.1. User Interface design

5.2. Components design (static and dynamic models of each component)

5.3. Database design

5.4. Rationale for your detailed design models

5.5. Traceability from requirements to detailed design models

6. Test Management

6.1. A complete list of system test cases

6.2. Traceability of test cases to use cases

6.3. Techniques used for test case generation

6.4. Test results and assessments (how good are your test cases? How good is your software?)

6.5. Defects reports

7. Conclusions

7.1. Outcomes of the project (are all goals achieved?)

7.2. Lessons learned

7.3. Future development

8. Appendices

8.1 Development environment

8.2 Third party libraries used

References

List of Figures

1. Introduction

1.1 Purpose and Scope

The purpose of this project is to create a user-friendly music player application that allows users to play audio files from a library, create and manage playlists, and customize playback modes (e.g., shuffle, loop, and order). This software is intended for personal use to enhance the audio listening experience.

- Supported platforms: Windows, macOS, Linux

- Minimum system requirements:

  * 2 GB RAM

  * 1 GHz processor

  * 100 MB available storage

1.2 Product Overview

Capabilities:

- Play songs in sequential, shuffled, or looped order.
- Allow users to create, rename, and manage playlists.
- Support for volume adjustment.
- Display song information, including title, artist, and duration.

Scenarios for Using the Product:

- A user can load songs into the library and play them in various playback modes.
- A user can create a custom playlist and rename it for better organization.
- A user can use shuffle mode for a randomized listening experience.

1.3 Structure of the Document

This document is structured into sections that detail the project management plan, requirement specifications, architecture, design, testing, and conclusions.

1.4 Terms, Acronyms, and Abbreviations

- UI: User Interface
- MP3: MPEG Audio Layer III

2. Project Management Plan

2.1 Project Organization

The project was divided into three main phases:

1. Development of core music player functionalities.
2. Implementation of playlist management.
3. Integration of playback modes.

2.2 Lifecycle Model Used

The project followed an iterative development model. Features were added and tested incrementally.

2.3 Risk Analysis

- Risk: File path issues when loading songs. Mitigation: Thorough error handling.
- Risk: User confusion over playback modes. Mitigation: Clear UI labels and instructions.

2.4 Hardware and Software Resource Requirements

- Hardware: A computer with Java Runtime Environment installed.
- Software: NetBeans IDE, JavaFX library, Java SDK.

2.5 Deliverables and Schedule

- Deliverables: Fully functional music player, user's manual, developer's guide.
- Schedule: The project was completed between 11/9/24 - 12/4/24.

3. Requirement Specifications

3.1 Stakeholders

- Primary users: Individuals looking for a lightweight, customizable music player.

3.2 Use Cases

3.2.1 Graphic Use Case Model

The use case diagram illustrates interactions between the user and the music player system.

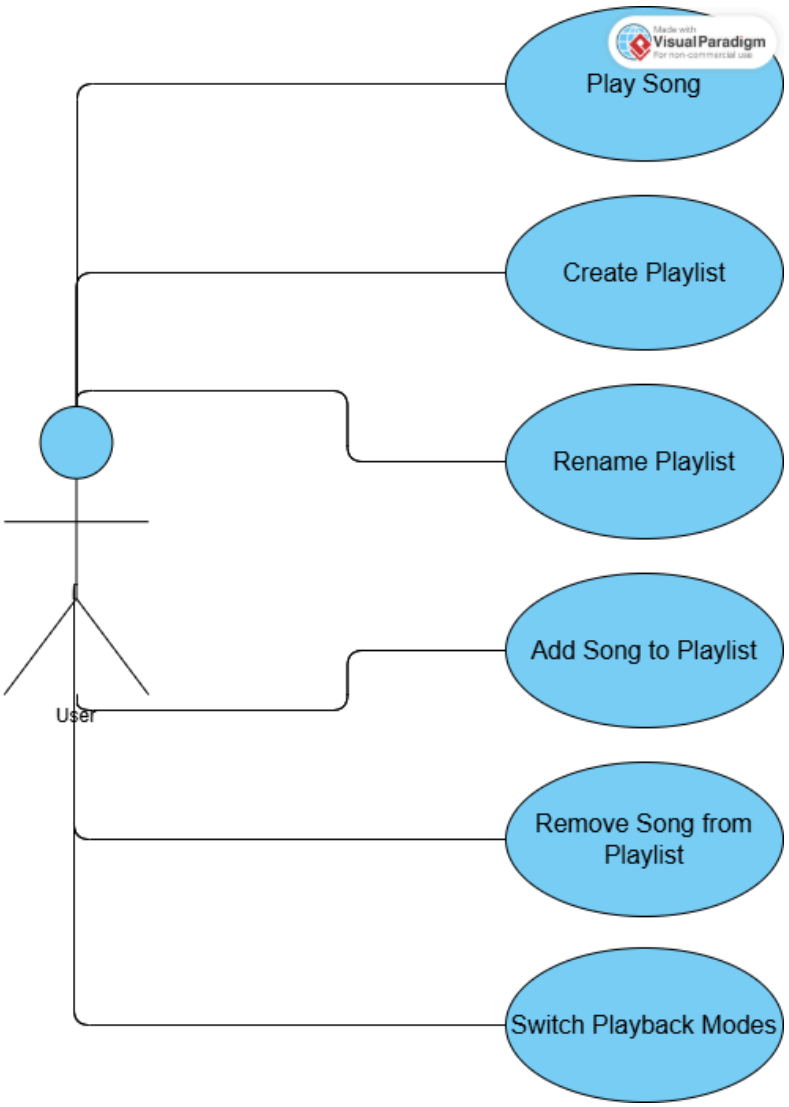

Fig 1.1

Use Case Diagram:

- Actors: User.
- Use Cases:
  - Play Song.
  - Create Playlist.
  - Rename Playlist.
  - Add Song to Playlist.
  - Remove Song from Playlist.
  - Switch Playback Modes (Order, Shuffle, Loop).

3.2.2 Textual Description

Use cases:

- Play Song
    1. Actors: User.
    2. Steps:
        - Select a song from the library or playlist.
        - Click the Play button.
        - Song starts playing, and details are displayed.

- Switch Playback Modes
    1. Actors: User.
    2. Steps:
        - Select a playback mode from the dropdown menu (Order, Shuffle, Loop).
        - Songs play according to the selected mode.
    3. Extensions:
        - Show an error message if there are no songs in the library.

- Create and Name a Playlist
    1. Actors: User.
    2. Steps:
        - Enter a playlist name in the text field.
        - Click Save Name.
        - Playlist is updated with the entered name.
    3. Extensions:
        - Prevent saving if the name field is empty.

- Play Song from Playlist
    1. Actors: User.
    2. Steps:
        - Click a song in the user playlist.
        - Song starts playing, and details are displayed.

    3. Extensions:

        - Display an error if the song file is missing.

- Add Song to Playlist
    1. Actors: User
    2. Steps:
        - Click "Add" button.
        - Selected songs are added to the chosen playlist.
    3. Extensions:
        - Prevent adding duplicate songs to the same playlist.
        - Display error message if no playlist is selected.
        - Allow multiple song selection for batch adding.

- Remove Song from Playlist
    1. Actors: User
    2. Steps:
        - Click "Remove" or "Delete" button.
        - Selected songs are removed from the playlist.

3.3 Rationale

The use case model was designed to cover typical user interactions with the system, ensuring functionality meets user needs.

The use case model was designed to:

- Capture user expectations comprehensively

- Ensure flexibility in user interactions

- Provide a clear roadmap for feature implementation

- Identify potential edge cases and system behaviors

3.4 Non-functional Requirements

- The system must support MP3 files.
- The UI should load within 2 seconds.
- The application should load songs and playlists within 2 seconds.
- Ensure the UI/UX design is simple and accessible for all users.

4. Architecture

4.1 Architectural Style

The project uses an event-driven architecture, leveraging JavaFX for UI and MediaPlayer for audio playback.

4.2 Architectural Model

The system comprises:

- UI Components: Buttons, ListViews, ComboBoxes.
- Media Player Core: Handles playback and controls.

- The system architecture includes:
    1. UI Layer: Contains JavaFX components like Buttons, ListView, and ComboBox.
    2. Controller Layer: Handles user actions (playSong, skipClicked, etc.).
    3. Playback Layer: Uses JavaFX MediaPlayer to manage audio playback.

    Logical view is employed to present the architecture of this music player app. There are several key components associated with the app. First is the GUI component. This is the front-facing layer that interacts directly with the user. It is responsible for handling user inputs, displaying data to the user, and providing a seamless and interactive experience. Second is the playback control component. This component processes the user inputs as received by the GUI component and proceeds with the desired outputs. Third is the data management/storage component. Data includes songs and their corresponding data, playlists and their corresponding data. In addition, this component also handles any action to modify these data as initiated by the user. Logical view is employed to present the architecture of this music player app. There are several key components associated with the app. First is the GUI component. This is the front-facing layer that interacts directly with the user. It is responsible for handling user inputs, displaying data to the user, and providing a seamless and interactive experience. Second is the playback control component. This component processes the user inputs as received by the GUI component and proceeds with the desired outputs. Third is the data management/storage component. Data includes songs and their corresponding data, playlists and their corresponding data. In addition, this component also handles any action to modify these data as initiated by the user.

4.3 Technology, Software, and Hardware

- Technologies: JavaFX, Java Media API.
- Software: NetBeans IDE.

4.4 Rationale

This architecture ensures modularity and scalability for future enhancements.

5. Design

5.1 User Interface Design

- Simple and intuitive UI using JavaFX.
- Features a library view and user playlist view.

5.2 Components Design

- Static Model: UI components and their connections.
- Dynamic Model: Playback interactions and event handling.

Components design represents the key components of the system and their relationships:

1. UI Components: Buttons, ListView, and ComboBox for interacting with the user.
2. Controller: Handles logic such as playback and playlist management.
3. Media Layer: Manages audio files using JavaFX Media and MediaPlayer.

5.3 Database Design

No database was used. Data is stored in memory during runtime.

5.4 Rationale

The lightweight design ensures responsiveness and simplicity.

5.5 Traceability

All UI components are mapped to specific functionalities in the controller.

6. Test Management

6.1 Test Cases

- Test Case 1: From the homepage, click “play”
    - Expected Output: Song starts playing.
- Test Case 2: From the homepage, click “pause” and then click “play”
    - Expected Output: Song pauses playing, but can resume from where it pauses once “play” is clicked.
- Test Case 3: From the homepage, click “stop” and then click “play”
    - Expected Output: Song stops playing, but can start from the beginning once “play” is clicked.
- Test Case 4: From the homepage, move the volume slider to the right
    - Expected Output: The volume gets turned up.
- Test Case 5: From the homepage, move the volume slider to the left
    - Expected Output: The volume gets turned down.
- Test Case 6: From the homepage, click “skip”
    - Expected Output: The current song is skipped, and the next song starts playing from the beginning.
- Test Case 7: From the homepage, select “in order” from the dropdown box
    - Expected Output: The song is traversed(played) according to its current order.
- Test Case 8: From the homepage, select “shuffle” from the dropdown box
    - Expected Output: The song is traversed(played) in a randomized order.
- Test Case 9: From the homepage, select “loop” from the dropdown box
    - Expected Output: The current song is being played over and over again from the beginning to the end.
- Test Case 10: From the homepage, type “playlist1” into the text bar underneath the “playlist” label, click “save”, click “add” to start adding 3 songs into “playlist1”
    - Expected Output: The new playlist named “playlist1” is created with the 3 selected songs.
- Test Case 11: From the homepage, select “playlist1”, and then click “add” to start adding 1 more song into “playlist1”
    - Expected Output: The existing playlist named “playlist1” is added with the 1 selected song while having 3 songs originally existing.
- Test Case 12: From the homepage, select “playlist1”, and then click “remove” to start removing 2 songs from “playlist1”
    - Expected Output: The 2 selected songs are removed from the existing playlist named “playlist1”.
- Test Case 13: From the homepage, select “playlist1”, and then type “playlist2” into the text bar underneath the “playlist” label, click “rename”
    - Expected Output: The existing playlist named “playlist1” is renamed as “playlist2” while having the same content.

- Test Case 14: Time how long it needs to complete each of the Test Cases 1-6
    - Expected Output: Each of the Test Cases 1-6 should take less than 1 second to respond.
- Test Case 15: Select 10 different songs of MP3 formats to be played
    - Expected Output: Each of the 10 songs can play with no interruption.

6.2 Traceability

Each test case maps to a use case (e.g., Play Song, Create Playlist).

6.3 Techniques

Manual testing was used due to the simplicity of the application.

6.4 Test Results

All test cases passed successfully.

6.5 Defects Reports

No significant defects were reported.

7. Conclusions

7.1 Outcomes

All project goals were achieved, including playback modes and playlist management.

7.2 Lessons Learned

- Iterative development ensured smooth progress.
- Comprehensive testing helped identify and fix bugs early.
- Learned the use of Scene Builder to develop GUI

7.3 Future Development

- Add support for more audio formats
- Enable saving playlists to disk for reuse
- Add support for playlist merging
- Add more songs to the playlist
- Create another stage and interface for playlist management
- Dark/light mode UI themes
- Lyrics display feature
- Keyboard shortcut support

8. Appendices

8.1 Development Environment

- Java Version: Java 17

- JavaFX Version: 17.0.2

- Development OS: Windows 10/macOS Big Sur

8.2 Third-Party Libraries Used

- JavaFX Media API

- Apache Commons IO (for file handling)

References

- Sommerville, Ian. *Software Engineering*. 10th Edition, Pearson Education, 2015.