Shazamify Pi
Technical Project Report

Nathanael Lee
Benjamin Ikanovic
Omar Pleitez

December 2023

Abstract

This document outlines the software architecture and development of Shazamify Pi, a real-time music recognition and analysis device powered by a Raspberry Pi. The report details the migration from a monolithic prototype to a robust Model-View-Controller (MVC) architecture, which forms the core of the application. Key features discussed include the asynchronous audio processing pipeline, integration with ACRCloud for song recognition, and use of the Spotify API for metadata retrieval. The report also covers the technical challenges encountered and the solutions implemented, concluding with a roadmap for future work.

GitHub Repository: `https://github.com/zealous-zebra/EE-2140FinalProject`

Contents

1 System Architecture and Development Summary

The Shazamify Pi project's software underwent a significant architectural maturation, transitioning from a functional prototype to a scalable application by implementing a Model-View-Controller (MVC) framework. This core objective was successfully achieved, establishing a structure that decouples the user interface from the backend logic to promote modularity and easier maintenance.

Key features were developed within this new architecture: the user interface was built using `PyQt6`, the audio processing layer for recording and analysis was modularized, and service clients for ACRCloud and Spotify were created. A crucial refinement was the use of `PyQt6`'s `QThread` to enable non-blocking audio recording, ensuring the UI remains responsive. Testing focused on verifying signal and slot connections between the UI and controller, and confirming successful data exchange with external APIs.

2 Architectural Refactoring and Core Feature Implementation

A primary focus of this development cycle was to refactor the project's foundation. The initial proof-of-concept, while functional, was built as a monolithic script that tightly coupled the UI with backend logic. This design hindered scalability and parallel development. To address these limitations, a deliberate migration to a Model-View-Controller (MVC) architecture was performed.

2.1 The Model-View-Controller (MVC) Framework

Purpose: To separate application logic from the user interface, improving code organization, testability, and scalability.

Implementation: The application was restructured into a dedicated Python package (`shazamify/`). The "View" (`shazamify/ui/`) was built with `PyQt6` and is responsible only for displaying data and emitting user interaction signals. The "Model" (`shazamify/services/` and `shazamify/audio/`) handles all data processing and business logic, including API communication and audio analysis. The central `Controller` (`controller.py`) acts as the intermediary, listening for signals from the View and orchestrating the Model's functions to fulfill user requests.

Validation: The architecture was validated by confirming that UI events correctly triggered controller methods via `pyqtSignal` connections. This refactoring yielded immediate benefits, including improved modularity and the ability for team members to develop components in parallel.

2.2 Asynchronous Audio Processing and Recognition

Purpose: To capture and identify audio without freezing the application, a critical requirement for a smooth user experience.

Implementation: Within the new architecture, a `Recorder` class inheriting from `QObject` was designed to run in a separate `QThread`. This module handles audio capture using the `sounddevice` library and emits signals upon progress and completion. For song identification, a `RecognitionClient` interfaces with the ACRCloud API, sending a saved `.wav` file for analysis.

Validation: Functionality was confirmed by initiating recordings from the UI. The application remained fully responsive, progress updates were correctly displayed, and the recorded audio file was successfully sent to the ACRCloud API upon completion.

2.3 API Integration for Metadata Retrieval

Purpose: To fetch detailed song information after a successful identification.

Implementation: A modular `SpotifyClient` class was created using the `spotipy` library as part of the "Model" layer. This module handles API authentication and provides a clean interface for searching a track by its title. It returns structured data including song name, artist, album, and the album art URL.

Validation: The client was tested by providing it with song titles identified by ACRCloud. It consistently returned an accurate dictionary of song details, which were then correctly passed by the controller to the UI for display.

3 Challenges and Resolutions

Key technical challenges in this iteration involved ensuring application stability and security, both of which were addressed by the architectural improvements.

- **Challenge 1:** UI responsiveness during I/O operations. Initial attempts to record audio directly from a UI button callback caused the application to freeze.
  Resolution: The `PyQt6` threading model (`QThread` with a worker `QObject`) was implemented. Audio recording tasks were offloaded to a background thread, which communicated back to the main UI thread via signals, ensuring a smooth user experience.

- **Challenge 2:** Secure management of API credentials. Storing API keys directly in the source code posed a security risk.
  Resolution: A `.env` file was used to store sensitive keys for Spotify and ACRCloud. The `python-dotenv` library was integrated to load these keys as environment variables at runtime. This file was added to `.gitignore` to prevent it from being committed to version control.

4 Future Work and Next Steps

With a stable software foundation now in place, future development will focus on transitioning the software prototype into a fully realized hardware product. Key areas for the next development cycle include:

- Final integration of the existing software with the Raspberry Pi hardware platform.
- Performance testing and optimization of the audio recognition pipeline on the Raspberry Pi to evaluate latency and reliability.
- Expansion of project documentation to include hardware setup instructions and environment configuration steps.
- Refinement of the user interface for improved usability on the target touchscreen display.

These tasks will be prioritized in the upcoming iteration to ensure a fully functional end-to-end system.