8tracks player for Windows

Contents

[Introduction 2](#_Toc332190784)

[Context 2](#_Toc332190785)

[8tracks as a service 2](#_Toc332190786)

[Scope 2](#_Toc332190787)

[Stakeholders 2](#_Toc332190788)

[Users 3](#_Toc332190789)

[Use cases 3](#_Toc332190790)

[Product backlog 3](#_Toc332190791)

[Implemented 3](#_Toc332190792)

[User stories removed from the product backlog 4](#_Toc332190793)

[Postponed user stories 4](#_Toc332190794)

[Logical view 4](#_Toc332190795)

[Process view 6](#_Toc332190796)

[Development view 8](#_Toc332190797)

[Technologies and external libraries 8](#_Toc332190798)

[Notable patterns 9](#_Toc332190799)

[Deployment view 10](#_Toc332190800)

[Quality assurance 11](#_Toc332190801)

[List of deliverables 11](#_Toc332190802)

# Introduction

This document describes the design of the 8tracks player for Windows. The player was done during the post-master program at TU Eindhoven and it represents a mini-project which aimed at exploring and getting (more) familiar with new technologies and trends in software industry. Even though that the intended time for making this project was one working week, due to its large scope, the project took more time to finish.

The structure of the document is in accordance with the 4+1 architecture model. Additionally, some quality aspects are addressed at the end of the document

# Context

## 8tracks as a service

8tracks is handcrafted internet radio. It offers a simple way for people to share and discover music through an online mix, a short playlist containing at least 8 tracks. On 8tracks, people can do two things: listen to a mix, or create a mix. Listeners can search for a mix by artist or genre, stream it in a legal, radio-style manner, and follow others who make compelling mixes. DJs upload MP3s or AACs to craft a playlist, add context with description and art, and publish their mix on 8tracks.com and a personal webpage.

## Scope

The 8tracks player is aimed to be a desktop-based music player that streams music from the 8tracks.com playlists and it does not provide an interface for creating mixes on the 8tracks. The main motivation for making the player was to provide a dedicated, lightweight application for listening music from 8tracks.com, which is the main usage scenario. It can be run in the background and thus not be as intrusive as having playing it via web-browser.

## Stakeholders

|  |  |  |
| --- | --- | --- |
| Name | Description | Interest |
| Listeners | People using the application on a daily basis | Listening to music from 8tracks |
| Maintainers | Persons maintaining the 8tracks player | Adding new features to the player and easily tracking down the sources of reported bugs |
| 8tracks team | The company that is developing and maintaining 8tracks service | Promoting their service, expanding to a new platform |
| Music industry officials | Authorities from the music industry | Keeping track of songs that are listened (due to music licensing and other legal concerns). |

Table Stakeholders

Due to the limited time, the main focus was on satisfying the needs of listeners and maintainers. The interests of the 8tracks team are still not fulfilled as the 8tracks player does not implement some constraints imposed by the terms and conditions of the API usage. As such, at this stage of its development, the player cannot be made publicly available and promote 8tracks service.

## Users

The users of the 8tracks player are people that enjoy listening music from 8tracks.com.

# Use cases

Following the agile software development methodologies, the use cases were structured in a form of user stories and together they formed a product backlog. Periodically, the user stories were reprioritized or indefinitely postponed.

## Product backlog

### Implemented

*As a listener, I want to play a mix, in order to enjoy the music of a certain mix.*

* *As a listener, I want the system to automatically go to the next song, in order to avoid manually switching to the next song and to enjoy music with the least amount of interaction.*
* *As a listener, I want to skip tracks, in order to avoid listening to the songs I don't like.*
* *As a listener, I want to FF, RW, pause/play a song, volume up/down, in order to listen to the parts of songs I like and so on.*

*As a listener, I want to browse through the newest / popular mixes, in order to get to know the latest hits.*

* *As a listener, I want to be shown only a portion of all mixes at the time, in order to avoid getting overwhelmed with mixes. (pagination)*

*As a listener, I want to search for mixes (per tags), in order to listen to / discover the ones I like the most.*

*As a listener, I want to mark a mix I like, in order to be able to find it later easily and replay it.*

*As a listener, I want to be able to log on to the system, in order to keep my settings and preferences protected and synchronized with the 8tracks service.*

* *As a listener, I want the system to be able to log me in automatically, in order to aboid doing it myself every time.*
* *As a listener, I want the create a new account, so that I can use all user-related features such as liked mixes.*

*As a listener, I want to be able to quickly access similar mixes, in order to keep playing the same type of music.*

*As a listener, I want the system to go to the next mix, in order to avoid having to do it manually myself.*

*As a listener, I want to save the songs or a whole playlist to external memory, in order to listen to it later offline.*

### User stories removed from the product backlog

*As a listener, I want to "fav" a song, in order to be able to find the particular songs I like quickly.*

*As a listener, I want to be able to read comments, in order to see what others thought about a particular mix.*

*As a listener, I want to be able to leave comments about a mix, in order to give feedback to the person that made the mix.*

*As a listener, I want to see information about other users, in order to contact them or to listen through the mixes they made.*

* *As a listener, I want to be able to follow other users, in order to be continuously updated on their newest mixes because I found their previous mixes entertaining.*

These user stories were initially put in, but after scoping the project they were put aside because they were not adding value to the main use case of listening the music and the effort needed to implement them greatly outweighed the impact. It was decided that the users could still easily go to the 8tracks website and do those things.

*As a listener, I want the system to save the songs in the MP3 format, in order for me to play it later with other, for me more convenient players or devices.*

This user story was removed because there are already a lot of good solutions for audio format conversion and the 8tracks player should not try to do too many things (the Swiss-army knife software (anti)pattern).

### Postponed user stories

*As a music industry, I want the system to report to me each songs that reaches 30s mark, in order to assure that the 8track service is legal.*

The implementation of this user story is postponed until the work on implementing constraints from the API terms and constraints starts. That will happen in the next development phase – making the application obey the legal constraints.

# Logical view

The architecture consists of four layers: infrastructure, domain, application, and the user interface.

Figure Layered architecture

On the implementation side, the package structure follows the 4-tier structure:



Figure Package structure

Apart from the main 4-tiers, all layers components have access to the utility, general-purpose functions such as accessing the settings or configuration file.

The infrastructure layer is responsible for the communication with the 8tracks server and the operating system audio playback routines.



Figure Communication package



Figure Media package

The domain layer encapsulates the main entities of the domain such as mixes, users and songs. Also, it is responsible for the authentication of the user, interpreting the user requests and server responses, browsing the media library and controlling the audio playback.



Figure Domain package

The application layer build on top of the domain layer and implements the constraints and rules of the application’s business logic. It mainly consists of view-models (from the prominent MVVM pattern) for different parts of the user interface.

The user interface is a presentation layer that provides all the necessary information to the user. Without much logic, this layer just consists of XAML-based definitions of user interface.

# Process view

The 8tracks player aims to recreate and enhance the experience of 8tracks listeners. In order to do that it is providing a user interface towards the listeners, where they can do more or less the same things as on the 8tracks website:

* search for mixes given various filtering criteria such as category, tags, and keywords
* listen to a particular mix
* control the playback (volume, skipping tracks and so on)
* liking mixes
* logging in or creating an account

All user actions are translated in the domain layer to lower-level commands using which the 8tracks player communicates with the 8tracks server (e.g. to get information about a mix, log in the user, toggle liking a mix) or to the underlying audio player (e.g. playing a song, stopping or paying the audio playback, skipping to another part of the song).



Figure Component diagram

The communication with the 8tracks server is done via 8tracks API that is a XML-based REST interface. The details of the API can be seen here: <http://8tracks.com/developers/api>

Developing components for audio playback was an unexpected challenge. As it turns out, the support for playing audio files in native .NET is very slim. Firstly, a component that communicates with directX drivers using WMI (Windows Messaging Interface) was developed. Although it satisfied the basic needs of playing an audio file, stopping and so forth, it turned out that the communication with directX in other direction was flaky. For example, asynchronous notifications that the end of stream has been reached were not reaching the 8tracks player. Fortunately, Microsoft provides ac native .NET library to communicate with windows media player which fully accommodates the needs of the 8tracks player. In runtime, one of the two audio player mechanisms (WMP or DirectX) will be used.

The main user story of playing a mix involves all components. Firstly, the user issues a request to play a certain mix e.g. by pressing the play button. The 8tracks player then creates a request for the 8tracks server to play a certain mix, and the server responds with the message containing the URI of the first track in the mix. After that, the 8tracks player starts the audio playback via windows media player. Once the song is over, the windows media player will notify the 8tracks played that the end of stream is reached, the 9tracks player will get the URI of the next song from the 8tracks player and play it.



Figure Sequence diagram for playing a mix

# Development view

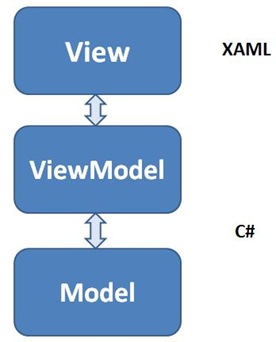
## Technologies and external libraries

The 8tracks player was developed using the .NET software stack. The basis for the development was .NET 4.0 - or more precisely, C# and WPF. The player utilizes the new features of .NET that were introduced in its version 4, such as Reactive extensions (Rx) and Language INtegrated Querying (LINQ). The combination of the two makes a very expressive and powerful tool for solving problems revolving around asynchronicity and collections. Both elements are present in the 8tracks player:

* Asynchronicity is there wherever there are is interaction with the user and time-consuming function calls (e.g. communication with the server) and,
* Operating over collections is in the nature of the problem – the user is presented with a number of mixes, each mix contains a number of tracks.

Since the target platform for the 8tracks player is Windows, C# chosen as the most modern language for Windows development. Because of its support for MVVM pattern which promotes loose coupling and provides testable design, WPF was chosen for the development of the user interface. In order to ease the integration of the business logic developed in C# and the user interface written in XAML, a ReactiveUI library was used.

The ReactiveUI library closes the gap between the Reactive Extensions in C# and WPF, giving the developers a very nice way of connecting reactive enumerable (so called observables) and WPF property notification mechanisms.

More information about Reactive Extensions and Reactive UI can be found here: <http://www.reactiveui.net/>

## Notable patterns

The pattern that shaped the df calculation, whichesign of the whole application is Model-View-ViewModel (MVVM). MVVM facilitates a clear separation of the development of the graphical user interface (in this case XAML) from the development of the business logic or back end logic known as the model. The view model of MVVM is a value converter meaning that the view model is responsible for exposing the data objects from the model in such a way that those objects are easily managed and consumed in the user interface. In fact, in XAML is not possible to do any kind of arithmetic calculations. Furthermore, MVVM was designed to make use of data binding functions in WPF to better facilitate the separation of view layer development from the rest of the pattern by removing virtually all GUI code (“code-behind”) from the view layer.



Figure MVVM pattern in the 8tracks player

In order to separate the application from the external processes and to unify the interface towards different audio players, the Adapter pattern was used. Also, in order to make audio players statefull and additional checks, Decorator pattern was used to enhance the functionality of any underlying audio player.



Figure Decorator and Adapter pattern

# Deployment view

The player is distributed to the listeners in a form of installer that contains all the necessary libraries and configuration files the player needs. The user needs to assure a stable internet connection in order to enable the 8tracks player to communicate and stream music from the 8tracsk server without any problems.



Figure Deployment diagram

The target platform for the player is Windows operating system. The application was built for x86 generation of processors, thus it will successfully run on both 32 and 64 bit versions of Windows operating system.

# Quality assurance

During the development of the player, Test-driver development was used following the well-known practice: write a test first, see it fail, make it pass and refactor towards better design. In addition to that, a behavioral driven development (BDD) approach was applied that allowed the user stories to be incorporated in test cases. The framework used for the BDD was StoryQ and more information about it can be found here: <http://storyq.codeplex.com/>

# List of deliverables

At the end of this mini-project the following items were delivered:

* Source code of the project and the test cases
* Mini project report
* Installer for the 8tracks player
* Testing report