Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСИС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

ПРОГРАММНОЕ СРЕДСТВДО

«ПРОИГРЫВАТЕЛЬ ПОДКАСТОВ»

БГУИР КП 1-40 01 01  001  ПЗ

Студент: гр. 651001 Азаров А. А.

Руководитель: Красковский П. Н.

Минск 2018

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ. 5

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6

1.1 Обзор функционала социальной сети Twitter 6

1.2 Анализ существующих аналогов 7

1.3 Постановка задачи 10

2 Разработка ПРОГРАММНОЙ АРХИТЕКТУРЫ 12

2.1 Разработка функциональных требований 12

2.2 Разработка алгоритма программного средства 13

2.3 Методы взаимодействия с Twitter API 15

2.4 Шаблон проектирования MVC 16

3 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ пс 17

3.1 Средства разработки ПС 17

3.2 Разработка программных модулей 18

4 Тестирование пс 22

5 Руководство пользователя 26

Заключение 32

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 33

Приложение А Текст программного модуля MainWindow.xaml.cs 34

Приложение Б Текст программного модуля LoginWindow.xaml.cs 36

Приложение В Текст программного модуля SaveWindow.xaml.cs 38

Приложение Г Текст программного модуля TweetWindow.xaml.cs 40

Приложение Д Текст программного модуля MainController.cs 41

Приложение Е Текст программного модуля LoginController.cs 45

Приложение Ж Текст программного модуля SaveController.cs 48

Приложение И Текст программного модуля TweetController.cs 50

Приложение К Текст программного модуля Token.cs 51

Приложение Л Текст программного модуля User.cs 52

Приложение М Текст программного модуля Tweet.cs 53

Приложение Н Текст программного модуля SavedToken.cs 54

Приложение П Текст программного модуля UserTimeLine.cs 55

Приложение Р Текст программного модуля HomeTimeLine.cs 56

**ВВЕДЕНИЕ**

Термин подкастинг появился в 2004 году и обозначает процесс создания и распространения звуковых файлов в Интернете. Как и радиостанции, подкасты имеют определённую тематику и периодичность издания. Человек, записывающий подкасты, называется подкастером. Обычно они делают это на любительской основе, хотя подкасты выпускаются и крупными медиа или новостными корпорациями. Распространяется подкаст обычно свободно, в виде .mp3/.aac файлов и публикуется на конкретном ресурсе в Интернете, с возможностью подписки. Этот ресурс, часто является сайтом, поддерживающий хостинг медиафайлов и в какой-то степени автоматизирующий помещение записей и подписку на обновления. Является типом социальных медиа и схож с технологией видеоблогов и интернет-радио. Может содержать запись речи в текстовом виде.

Для удобного прослушивания подкастов создано множество программных продуктов, таких как iTunes, Zune Software, Rhythmbox, gPodder, AmaroK. Они следят за обновлением подкаст-лент и автоматически загружают новые выпуски. Проверка обновлений обычно обеспечивается с помощью RSS. Подкасты могли изначально называться рсс-кастингами, но данное название оказалось слишком сложным в произношении. Настоящее название «подкаст» произошло от слов “iPod” и “casting”. iPod – торговая марка серии портативных медиапроигрывателей компании Apple, которая популяризировала прослушивание подкастов среди своих пользователей. “Casting” с английского переводится как «трансляция», что отсылает к радио и телепередачам.

На текущий момент именно iTunes, бесплатный музыкальный плеер компании Apple, является самой большой площадкой для хранения и каталогизации подкастов в Интернете, предоставляя каждому желающему записывать и регулярно выкладывать выпуски своих подкастов. И даже более, iTunes выработал стандарт де-факто для формата распространения подкастов через RSS.

Русскоязычный подкастинг развивается не так сильно, как на Западе, где каждое СМИ с DAU (уникальными пользователями в день) больше 100 тысяч почти наверняка имеет свой подкаст. Из русскоязычных СМИ подкасты регулярно выпускает разве что «Медуза», у которых аж 4 регулярных подкаста по состоянию на середину 2018 года. В остальном сегмент представлен любительскими подкастами, ведь требуется лишь микрофон, программа для записи и немного времени и, возможно, через неделю вашим голосом зазвучат колонки и наушники не только в России, но и во всём мире. Тем не менее, такие подкасты редко могут похвастаться хорошим качеством записи.

В результате личной заинтересованности автора в прослушивании подкастов в метро по дороге в университет, было решено создать приложение-клиент для компьютера под управлением операционной системы macOS, обладающее интуитивно понятным пользовательским интерфейсом и подходящее для ежедневной потребности автора в прослушивания подкастов.

Данное программное средство было написано на языке Swift 4.2 в среде разработки Xcode 10. Использовался графический фреймворк Cocoa и библиотека для работы с XML файлами «SWXMLHash» за авторством David Mohundro (@drmohundro).

В данной пояснительной записке отображены следующие этапы написания курсовой работы:

1. Анализ технологической базы подкастинга. Аналитический обзор существующих аналогов разрабатываемого программного продукта. Постановка задачи.
2. Разработка архитектуры программного средства и его функциональных возможностей. Анализ методов работы с сетью.
3. Обоснование технических приёмов программирования. Описание модулей программы.
4. Тестирование и проверка работоспособности программного средства. Анализ полученных результатов.
5. Руководство по использованию программного средства.

**1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

* 1. **Обзор технологической базы подкастинга**

«Подкаст» (англ. Podcast) – эпизодическая серия цифровых аудиофайлов, которые пользователь может загрузить и прослушать. Они часто доступен для подписки, так что новые эпизоды автоматически загружаются компьютер, мобильное приложение или портативный медиаплеер. Подкастинг очень похож на интернет-радио и не редко пересекаться с этим термином, тем не менее, подкасты часто имеют определённую тематику, незначительно меняющуюся от выпуска к выпуску, плюс подкасты можно скачать локально.

Создатель подкаста предоставляет доступ к списку файлов на сервере, доступ к которому можно получить через Интернет. Слушатель использует специальное клиентское ПО, который обращается к этому веб-серверу, проверяет наличие обновлений и позволяет автоматически загружать новые выпуски подкаста и даже воспроизводить их одновременно, не скачав полностью, как если бы подкасты транслировались, как интернет-радио. В реальной жизни подкасты могут включать в себя, помимо аудио файлов, видео, PDF, ePub файлы. Однако основным материалом является всё же аудиофайл. Создатели подкастов могут монетизировать свои подкасты, позволяя компаниям покупать рекламное время, или с помощью специальных сайтов, как Patreon, который позволяет специальный дополнительный контент для слушателей за определённую плату. Подкастинг является примером горизонтального медиа – производители являются потребителями, потребители могут стать производителями.

Подкаст распространяется в виде RSS-рассылки. RSS (Rich Site Summary – обогащённая сводка сайта) – семейство XML-форматов, предназначенных для описания лент новостей, анонсов статей, изменений в блогах и т.д. Информация в формате RSS обычно собирается, обрабатывается и представляется пользователю в удобном для него виде специальными программами-агрегаторами. Пример файла в формате RSS 2.0:

<?xml version="1.0"?>

**<rss** version="2.0"**>**

**<channel>**

**<title>**Liftoff News**</title>**

**<link>**http://liftoff.msfc.nasa.gov/**</link>**

**<description>**Liftoff to Space Exploration.**</description>**

**<pubDate>**Tue, 10 Jun 2003 04:00:00 GMT**</pubDate>**

**<managingEditor>**editor@example.com**</managingEditor>**

**</channel>**

**</rss>**

К сожалению, общего стандарта оформления RSS файла для подкастов не существует, однако компания Apple разработала стандарт де-факто в данной сфере. В XML-файле должны обязательно присутствовать такие теги, как <itunes:duration>, <itunes: summary>, <itunes:image> с атрибутом href и т.д.

Достоинства:

1. Очень большое разнообразие контента на любую тематику.
2. Подкасты независимы от конкретной компании и будут существовать ещё очень долго.
3. Скорость – при публикации нового эпизода пользователи сразу же видят его в своей ленте.
4. Отсутствие цензуры.
5. Низкий порог вхождения, каждый может создать свой подкаст и поделиться со всеми в Интернете.

Недостатки:

1. Подкасты отвлекают от сфокусированной работы, нужно свободное время для прослушивания
2. Отсутствие формата представления подкастов в виде RSS-файла

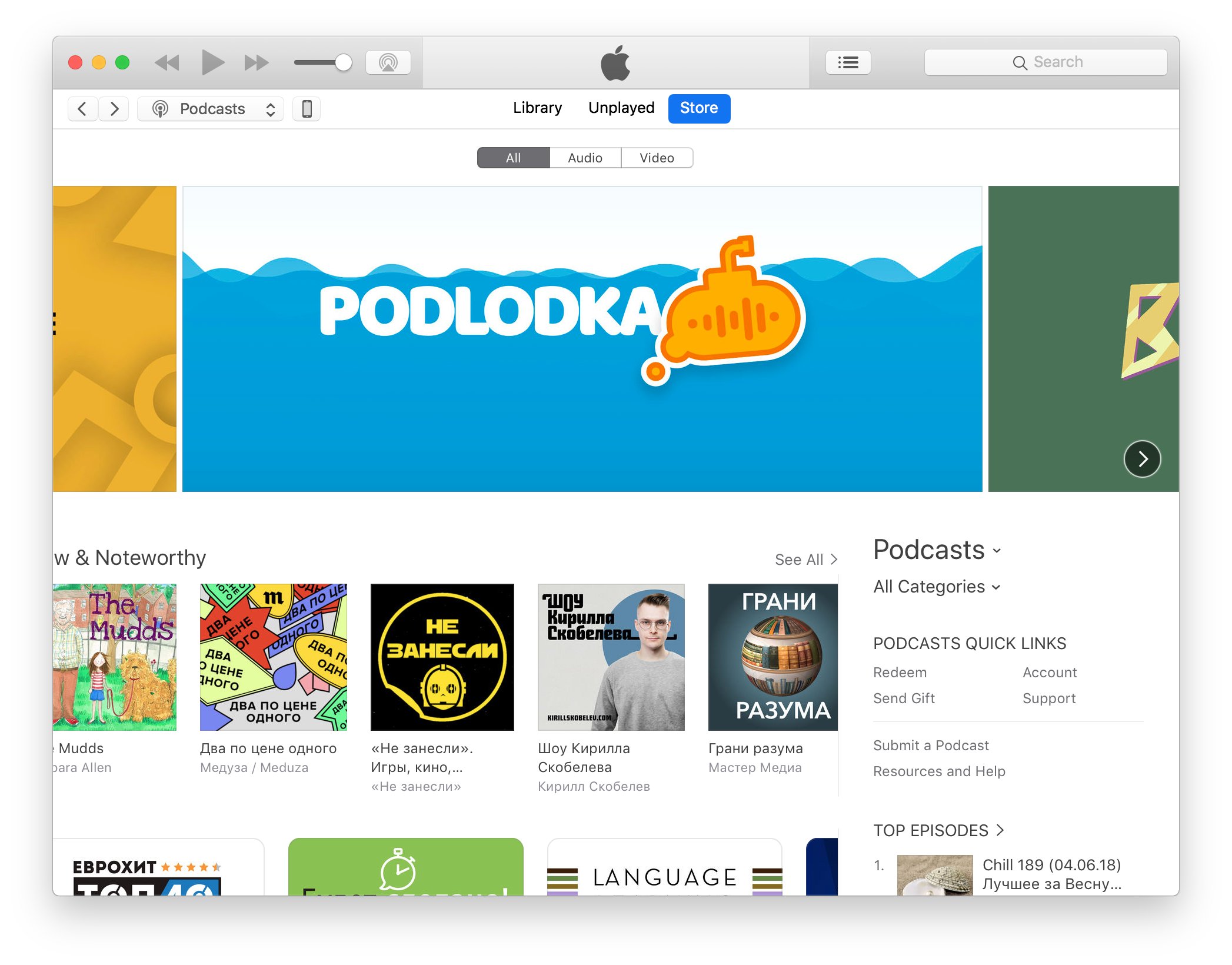
# 1.2 Анализ существующих аналогов

Можно выделить следующий общий функционал всех подкаст-клиентов:

* добавление подкаста;
* возможность просмотра списка добавленных подкастов;
* возможность просмотра эпизодов каждого подкаста;
* прослушивание отдельного эпизода;
* управление воспроизведением, громкостью.

Рассмотрим наиболее популярные на данный момент клиенты для операционных систем macOS и iOS:

**iTunes [1]** – медиаплеер для организации и воспроизведения музыки и фильмов, разработанный компанией Apple и распространяемый бесплатно. Его возможности далеко выходят за пределы обычного подкаст-клиента, тем не менее, это самое популярное приложение для прослушивания подкастов.

**

*Приложение iTunes для macOS 10.14*

Достоинства:

* интеграция в экосистему Apple;
* регулярные обновления;
* приложение является бесплатным;
* основная площадка для размещения подкастов.

Недостатки:

* слишком большое приложение с огромным количеством функций;
* прослушивание подкастов – это не основная функция;

**Castro [2]** – простое приложение для прослушивания подкастов. Предоставляет весь необходимый функционал. Доступен в Apple Store.



*Приложение Castro для iOS*

Достоинства:

* офлайн сохранение эпизодов;
* встроенный поиск подкастов в интернете;
* сохранение места прослушивания эпизода;
* нет русскоязычной версии.

Недостатки:

* только платная версия;
* устаревший дизайн.

**1.3 Постановка задачи**

Исходя из проведенного анализа можно сформировать требования к проектируемому программному средству.

Назначение приложения заключается в предоставлении пользователю возможности прослушивать конкретные эпизоды выбранных подкасты, сохранять список добавленных подкастов, удалять подкасты.

Пользователю доступны следующие возможности:

* добавление подкаста в список избранных;
* сохранение избранных подкастов между запусками приложения;
* просмотр списка избранных подкастов;
* удаление подкастов из списка избранных;
* вывод информации о каждом эпизоде: название, описание, дата выхода;
* прослушивание конкретного эпизода;
* управление перемоткой и возможность поставить воспроизведение на паузу;
* Управление громкостью при воспроизведении.

Приложение разработано в среде программирования Xcode 9, язык Swift 4 с использованием библиотеки SWXMLHash для работы с XML файлами, в качестве графической оболочки был использован фреймворк Cocoa.

Окончательный результат работы - оконное приложение, разработанное для операционной системы macOS. Программное средство должно потреблять минимальное количество ресурсов компьютера, но при этом предоставлять пользователю весь необходимый функционал.

Программа предоставляет интуитивно понятный пользовательский интерфейс на английском языке. Для корректной работы требуется интернет-подключение, достаточно быстрое для стриминга .mp3 фалов.

**2 РАЗРАБОТКА ПРОГРАММНОЙ АРХИТЕКТУРЫ**

**2.1** **Разработка функциональных требований**

Исходя из поставленной задачи и сформулированных требований, разрабатываемый программный продукт должен выполнять следующие действия:

1. осуществлять добавление подкаста в таблицу избранных подкастов;
2. предоставлять возможность удалять подкаст из таблицы избранных подкастов;
3. автоматически сохранять список избранных подкастов в базу данных и загружать заново при каждом запуске приложения
4. отображать таблицу всех эпизодов для каждого подкаста;
5. отображать название, дату выхода, описание для каждого эпизода в таблице;
6. воспроизводить выбранный пользователем эпизод;
7. управлять процессом воспроизведения эпизода:

* отображать текущую позицию в проигрывании эпизода
* отображать общую продолжительность эпизода;
* предоставлять возможность воспроизвести эпизод в любом месте;
* предоставлять возможность приостановить воспроизведение эпизода;
* управление громкостью воспроизведения. предоставлять возможность добавления отметки “нравится” к сообщению;

1. выводить основную информацию о создателе приложения, свою версию.

Пользователь должен иметь возможность взаимодействовать с интерфейсом в каждый момент работы программы.

Необходимо реализовать корректное отображение изображений (обложек) подкастов. Программное средство должно корректно обрабатывать все возникающие ошибки и информировать пользователя о них. Для корректного использования приложения требуется Интернет-соединение.

**2.2** **Разработка алгоритма программного средства**

На стадии проектирования приложения в качестве основной концепции стоит отметить выполнение выбранных пользователем действий.

Первоначально осуществляется отображение таблицы избранных пользователем подкастов. Если в базе данных имеются записи о сохранённых в прошлый раз подкастах, то необходимо их вывести на экран. Необходимо предоставить пользователю текстовое поле для ввода URL-адреса нового подкаста. В случае ввода некорректного адреса или при невозможности определить используемый формат RSS представления подкаста следует вывести сообщение об ошибке.

Далее пользователь должен выбрать подкаст, чтобы приложение ассинхронно начало загрузку списка эпизодов, информации о них и обложки подкаста. При этом пользовательский интерфейс не блокируется и предоставляет возможность совершать другие действия.

При нажатии кнопки “Delete” текущий открытый подкаст удаляется. Вся информация о только что удалённом приложении должна быть удалена с экрана.

При нажатии кнопки выхода или комбинации клавиш cmd-Q работа приложения завершается.

Укрупнённая схема данной концепции представлена на рисунке 2.1.

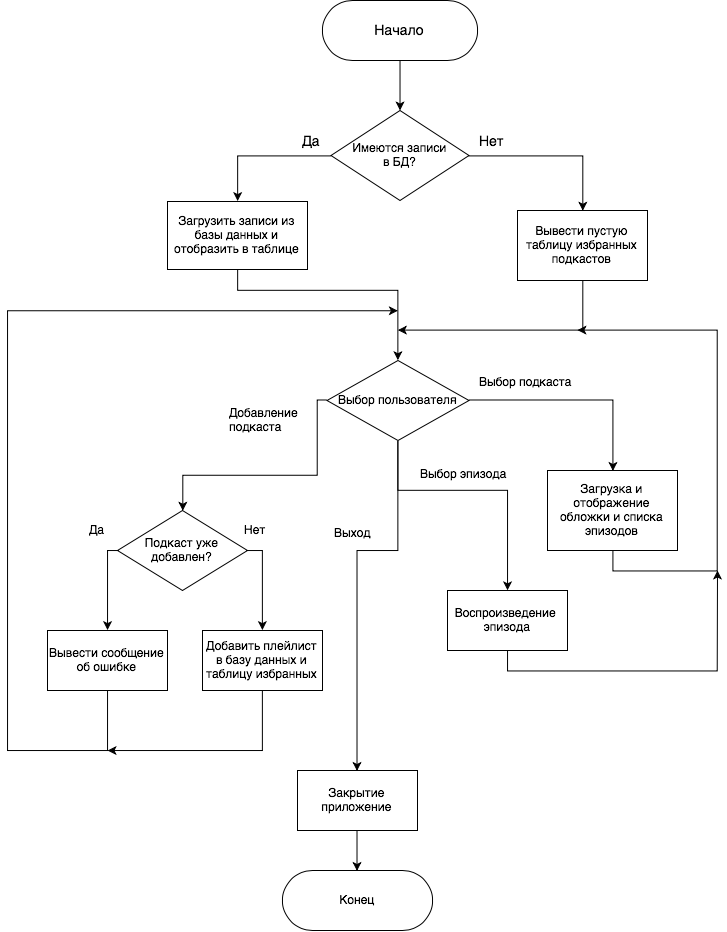


Рисунок 2.1 – Укрупнённая схема программы [3]

**2.3** Методы взаимодействия с RSS-файлом

Основного формата представления информации о подкасте в виде RSS-файла нет. Каждый хостинг придумывает свой формат и свои теги. Как как самый крупный хостинг на сегодняшний день – это iTunes, то большинство подкастов стремится соответствовать стандарту iTunes по оформлению подкастов. Тем не менее существует документ так называемых “best-practice”, который описывает, каким именно требованиям должен соответствовать RSS-файл с информацией о подкасте. [5]

Один RSS-файл описывает только один подкаст. Основная информация содержится в следующих тегах:

* Title (Заголовок)
* Description (Описание)
* Link (Ссылка)
* Update Frequency (Частота обновления)
* Author (Автор)
* Image (Обложка)

**Title.** Хранит название подкаста. RSS-файл обязан содержать этот тег и быть не пустым, чтобы считаться валидным. Следует избегать очевидных терминов «подкаст», «эпизоды», «выпуски» в этом теге. В RSS-файле путь к данному тегу обязан быть /rss/channel/title.

**Description.** Подкаст обязан иметь текстовое описание. Рекомендуется предоставлять длинную и короткую версию описания. Путь к данному тегу /rss/channel/itunes:subtitle.

**Link.** Не обязательный тег. Содержит ссылку на сайт подкаста. Эта ссылка должна открываться в браузере. Путь к данному тегу /rss/channel/link.

**Update Frequency.** Не обязательный тег. В нём рекомендуется указывать частоту обновления для проверки новых эпизодов. Если они больше не выходят, рекомендуется установить тег /rss/channel/itunes:complete.

**Author.** Этот тег содержит информацию об авторах подкаста. Используя стандартный тег iTunes рекомендуется указать только имя автора. Путь: /rss/channel/itunes:author.

**Image.** Обязательно следует предоставить ссылку на изображение, которое будет использоваться в качестве обложки подкаста. Путь к ссылке /rss/channel/image.

Далее следуют теги item, каждый из которых описывает структуру с информацией об эпизоде подкаста. Здесь основной интерес представляют теги:

* Title (Заголовок)
* Description (Описание)
* Publication Date (Дата публикации)
* Enclosure

**Title.** Хранит название эпизода. В RSS-файле путь к данному тегу обязан быть /rss/channel/item/title.

**Description.** Хранит описание эпизода. В RSS-файле путь к данному тегу обязан быть /rss/channel/item/description.

**Publication Date.** Хранит дату выхода эпизода. В RSS-файле путь к данному тегу обязан быть /rss/channel/item/pubDate.

**Ensclosure**. В атрибуте этого тега хранится путь к MP3 файлу с записью подкаста. Путь тегу: /rss/channel/item/enclosure.

Для проверки введённого пользователем URL делается запрос на данный адрес, если ничего не возвращается, то выводится сообщение об ошибке с предложением перепроверить введённый URL. Если был получен XML и в тегах **Title** и **Image** есть строковые значения, то адрес признаётся верным и новый подкаст добавляется в список избранных подкастов.

**2.3** Шаблон проектирования MVC [4]

Model-View-Controller – схема разделения данных приложения, пользова-тельского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо. Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления).

Модель (model) предоставляет данные и реагирует на команды контроллера, изменяя свое состояние. Модель не зависит от представления (не знает как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем) просто предоставляя доступ к данным и управлению ими.

Представление (view) отвечает за отображение данных модели пользователю, реагируя на изменение модели. Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

Контроллер (controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений. Контроллер обеспечивает «связи» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот.

Apple продвигает шаблон проектирования MVC как основной при разработке всех приложений на языке Swift. В разработанном приложении в качестве моделей рассматриваются файлы Parser.swift (обеспечивающий обработку пользователя), Episode.swift (представление одного эпизода). В качестве представлений – окно входа в аккаунт EpisodeCell (отображение ячейки эпизода в таблице) и файл Main.storyboard, в котором спроектирован основной интерфейс программы. В качестве контроллеров выступают классы PodcastsViewContorller (список избранных подкастов), PodcastDetailedViewController (выбранный пользователем подкаст), SplitViewController (разделение окна приложения на две части).

**3 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ ПС**

**3.1 Средства разработки ПС**

*Технология Cocoa Framework [6]*

Cocoa Framework – API Операционной Системы macOS для построения графических интерфейсов. Использование Cocoa Framework было обусловлено в первую очередь тем, что это гарантирует получение стандартного стиля операционной системы. Например, когда в macOS 10.14 добавили поддержку тёмного интерфейса, программу не пришлось даже перекомпилировать, она уже поддерживала тёмный стиль отображения.

Главное окно приложения управляется SplitViewContorller, который делит окно на две части, сущестующих независимо друг от друга. Левая отвечает за отображение списка избранных подкастов с помощью компонента NSTableView и добавление новых с помощью NSTextField и NSButton, а правая часть окна отображает обложку выбранного подкаста с помощью компонента NSImageView, список эпизодов с помощью NSTableView и название выбранного подкаста с помощью NSTextField. Половины окна могут взаимодействовать между собой с помощью паттерна делегирования, так как на этапе разбиения экрана, левой половине даётся ссылка на объект контроллера правой половнны. Это позволяет явно указывать правой половине экрана, какой подкаст пользователь хочет увидеть.

*Библиотека SWXMLHash [7]*

Для упрощения работы с XML-файлами была задействована библиотека SWXMLHash 4.0.0, что обеспечило некоторый уровень абстракции и простоту парсинга RSS-файлов, которые суть есть XML. Данная библиотека было подключена с помощью менеджера зависимостей CocoaPods, который сам по себе тоже был установлен, как зависимость в проекте разрабатываемого приложения.

Для парсинга бинарных данных, которые приходит после запроса по URL подкаста, используется статический метод SWXMLHash.parse(data). Этот метод возвращает объект класса XMLIndexer, который представляет собой высокоуровневую абстракцию над XML и позволяет работать с ним как с ассоциированным массивом.

**3.2 Разработка программных модулей**

Структура модулей программного средства основывается на обьектно-ориентированной методологии программирования и реализует шаблон проектирования MVC.

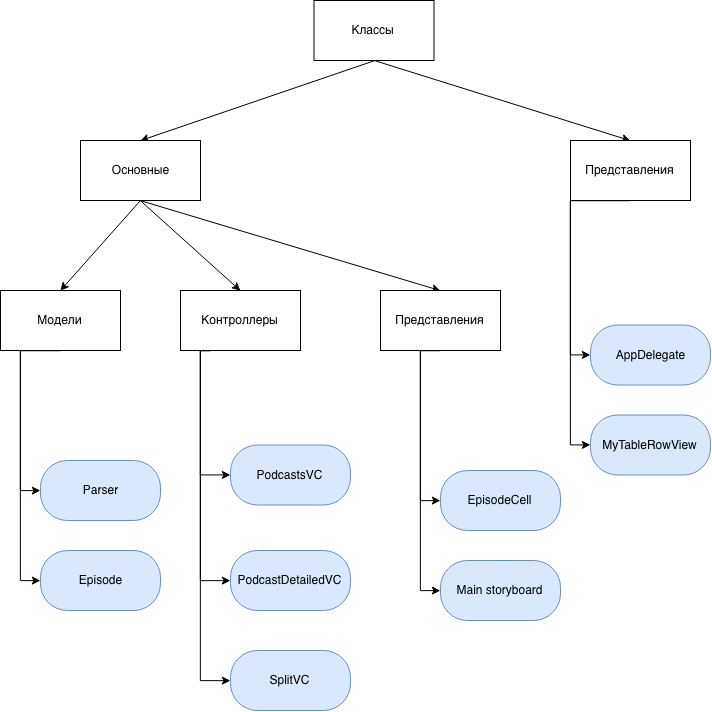


Рисунок 3.1 – Функциональные компоненты приложения

Класс Parser представляет собой класс, используемый для обработки и выделения информации о подкасте из XML- файла. Обьекты класса содержат метод getPodcastMetaData, принимающем бинарные данные, полученные после запроса на сервер пользователя и возвращающем информацию о подкасте, так же метод getEpisodes объекта возвращает массив объектов Episode.

Класс Episode представляет собой описание реального подкаста, а так же содержит статическое свойство formatter – объект класса DateFormatter, который используется для перевода строкового представления даты в RSS-файле в удобный для отображения формат.

Экземпляр класса PodcastsVC представляет View – объект ответчающий за обработку всех взаимодействий пользователя с представлением, в данном случае с левой частью экрана, где находится таблица избранных подкастов и NSTextField для добавления новых объектов. Для корректной работы таблицы требуется указать делегат объекта NSTableView и источник данных, то есть отображений рядов для таблицы. Этими делегатами в данном случае выступает объект класса PodcastVC, для этого класс соответствует интерфейсам NSTableViewDelegate и NSTableViewDataSource. Метод tableViewSelectionDidChange позволяет обработать клик пользователя на элемент таблицы.

Обьект класса PodcastDetailedVC ответчает за обработку всех взаимодействий пользователя с правой частью экрана, где находится таблица эпизодов подкаста и информация о текущем выбранном подкасте. Как и в PodcastsVC, требуется указать делегат объекта NSTableView и источник данных, то есть отображений рядов для таблицы, для этого класс соответствует интерфейсам NSTableViewDelegate и NSTableViewDataSource.

Экземпляр класса SplitVC создаётся раньше всех других контроллеров и создаёт остальные. Он вкладывает ссылки в PodcastsVC и PodcastDetailedVC друг на друга. Чтобы не произошла утечка памяти из-за подсчёта ссылок и замыкания объектов друг на друге, ссылка на объект PodcastsVC помечена как weak, это значит, что автоматический подсчёт ссылок не будет считать ссылку из PodcastDetailedVC в PodcastsVC.

Представление EpisodeCell отвечает за отображение ряда в таблице эпизодов. Класс NSTableView отвечает за отображение таблицы в окне программы и позволяет генерировать в коде кастомные вьюшки для рядов. Класс EpisodeCell как раз отвечает за генерацию одного ряда в таблице.

Файл Main.storyboard содержит самую основную информацию о внешнем виде программы, её окнах и порядке вызова контроллеров. Именно тут прописаны все отношения между компонентами фреймворка Cocoa и кодом в классах контроллера.

**4** **ТЕСТИРОВАНИЕ ПС**

Результаты тестирования программного средства представлены в таблице 4.1

Таблица 4.1 – Тестирование программного средства

|  |  |  |
| --- | --- | --- |
| Тестируемая функция, шаги | Ожидаемый результат | Полученный результат |
| 1.Запуск программы  2.Нажатие на закрытие окна или комбинации клавиш cmd-Q.  3.Попытка добавления подкаста с неверным форматом URL  4.Ввод верного URL, который не ведёт к корректному RSS-файлу  5.Добавление подкаста с помощью корректной ссылки на RSS-файл  6.Запуск программы без Интернет-соединения  7.Прогрузка главного ок-на приложения | 1.Открытие окна приложения  2. Закрытия окна приложения  3. Продолжение работы приложения, отсутствие новых подкастов  4.Вывод сообщения об ошибке и продолжение работы программы.  5.Добавление новой строчки с названием в таблице избранных подкастов  6.Вывод сообщения об ошибке соединения с сервером и завершение программы  7.Корректное отображение ленты новостей и профиля пользователя | 1.Открытие окна приложения  2. Закрытия окна приложения  3. Продолжение работы приложения, отсутствие новых подкастов  4. Вывод сообщения об ошибке и продолжение работы программы.  5. Добавление новой строчки с названием в таблице избранных подкастов  6. Вывод сообщения об ошибке соединения с сервером и завершение программы  7.Корректное отображение ленты новостей и профиля пользователя |

Продолжение Таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Тестируемая функция, шаги | Ожидаемый результат | Полученный результат |
| 8. Выбор подкаста в списке избранных подкастов  9. Удаление выбранного подкаста  10.Прокрутка списка эпизодов  11.Нажатие на эпизод  12.Нажатие на кнопку паузы  13.Попытка перемотки назад  14.Попытка перемотки вперёд, в уже загруженное место | 8. Отображение обложки подкаста, списка эпизодов  9. Исчезновение подкаста из списка избранных, очистка экрана  10. Корректное отображения названия выпуска, даты выхода, подробного описания эпизода.  11.Начало воспроизведения эпизода, появление компонентов управления проигрыванием  12.Приостановка проигрывания эпизода  13.Корректное воспроизведение эпизода с нового места  14.Корректное воспроизведение эпизода с нового места | 8. Отображение обложки подкаста, списка эпизодов  9. Исчезновение подкаста из списка избранных, очистка экрана  10. Корректное отображения названия выпуска, даты выхода, подробного описания эпизода.  11. Начало воспроизведения эпизода, появление компонентов управления проигрыванием  12.Приостановка проигрывания эпизода  13.Корректное воспроизведение эпизода с нового места  14.Корректное воспроизведение эпизода с нового места |

Продолжение Таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Тестируемая функция, шаги | Ожидаемый результат | Полученный результат |
| 15. Попытка перемотки вперёд, в ещё не загруженное место  16.Нажатие кнопки управления громкостью  17.Передвижение слайдера громкости | 15.Корректное воспроизведение эпизода с нового места  16.Появление панели управления громкостью (слайдер)  17.Изменение громкости воспроизведения эпизода | 15.Корректное воспроизведение эпизода с нового места  16. Появление панели управления громкостью (слайдер)  17. Изменение громкости воспроизведения эпизода |

**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Для запуска приложением достаточно запустить файл azPodcasts.app или собрать из исходников в Xcode. Готовый проект доступен по ссылке <https://github.com/azarovalex/azPodcasts>. При первом запуске пользователь увидит пустой экран и пустую таблицу слева. Под этой таблицей есть поле ввода для добавления нового подкаста

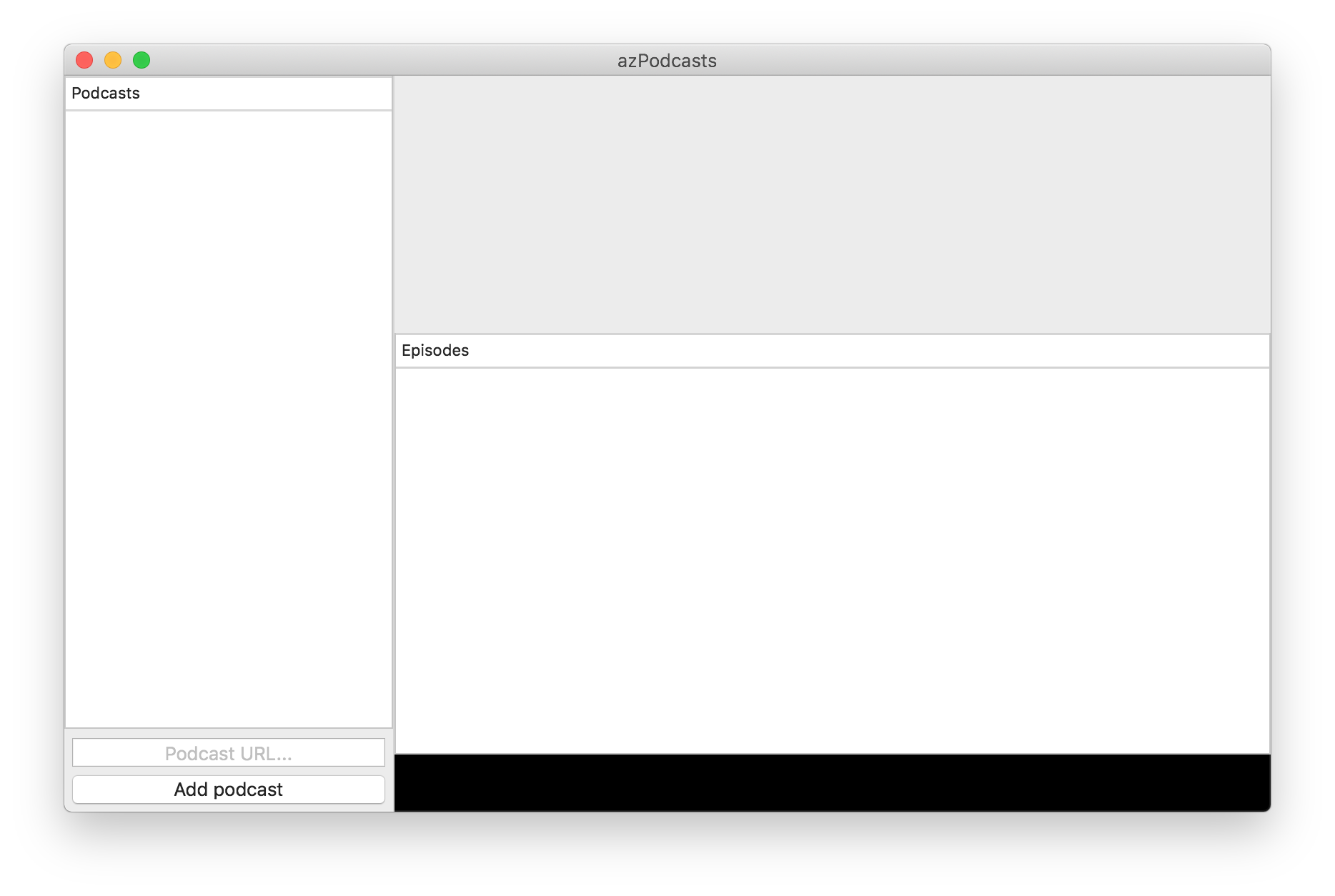


Рисунок 5.1 – Пустое окно при запуске программы

При нажатии кнопки “Add podcast” приложение попытается скачать содержимое по данной ссылке и представить как RSS-файл подкаста, для его последующего добавления в таблицу избранных. В случае некорректного формата RSS файла выведется окно с ошибкой. (Рисунок 5.2)

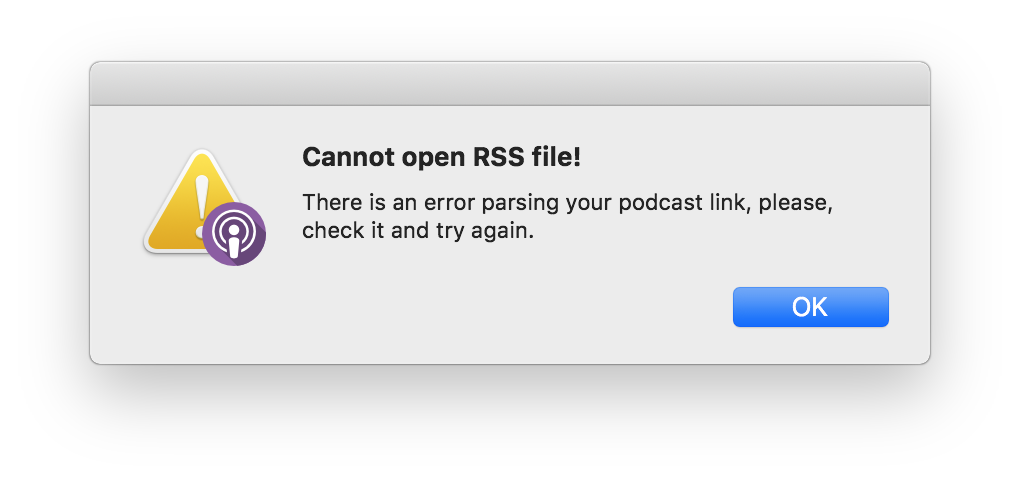


Рисунок 5.2 – Сообщение об ошибке открытия RSS-файла

Если пользователь ввёл корректный адрес подкаста, его название отобразится в левой таблице (рисунок 5.3).

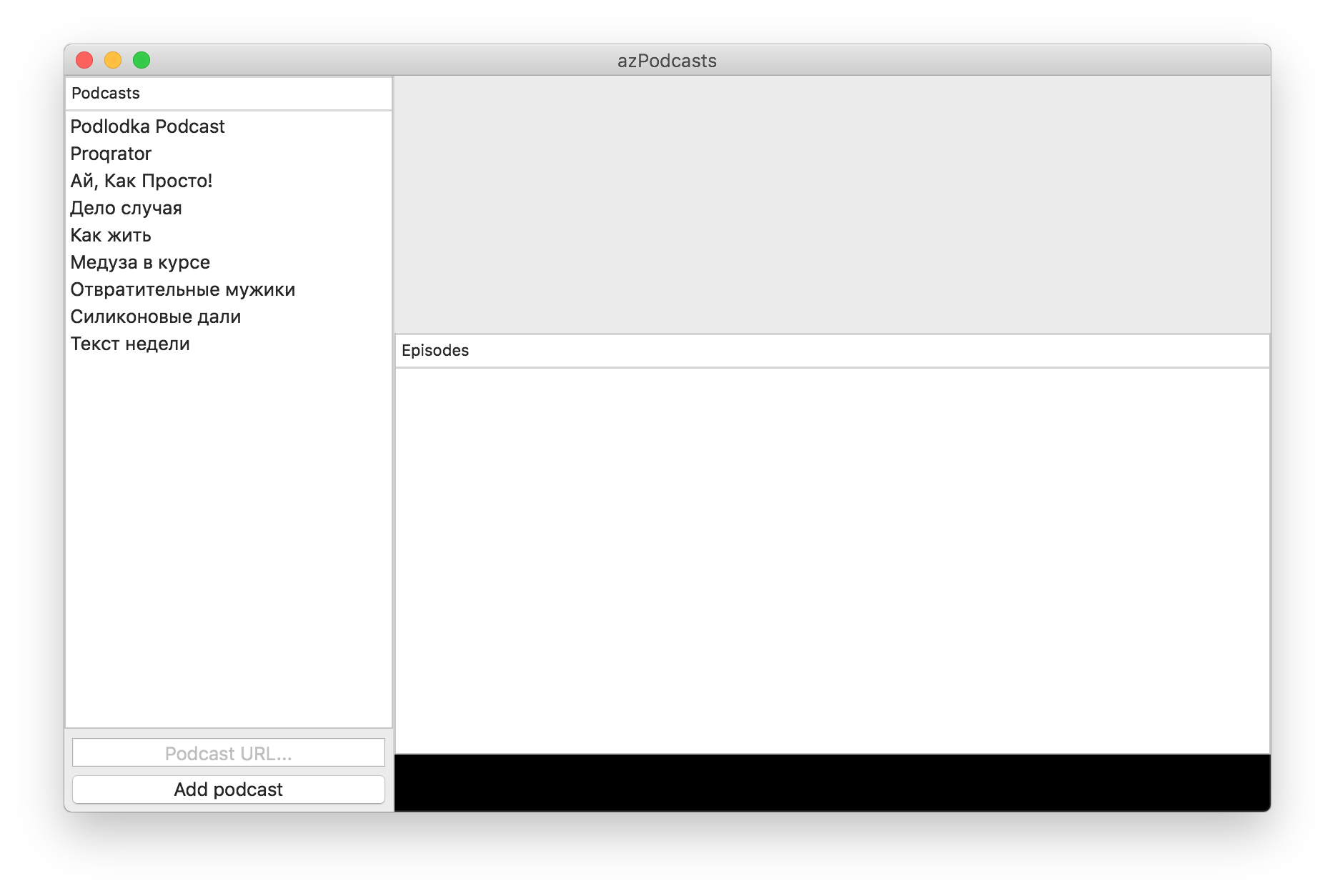


Рисунок 5.3 – Внешний вид приложения с добавленными подкастами

При выборе подкаста в таблице избранных, пользователю отображается окно с подробной информацией о выбранном подкасте: название, обложка и список эпизодов, при чём каждый эпизод отображается вместе со своим названием, датой выпуска и подробным описанием. (Рисунок 5.4)

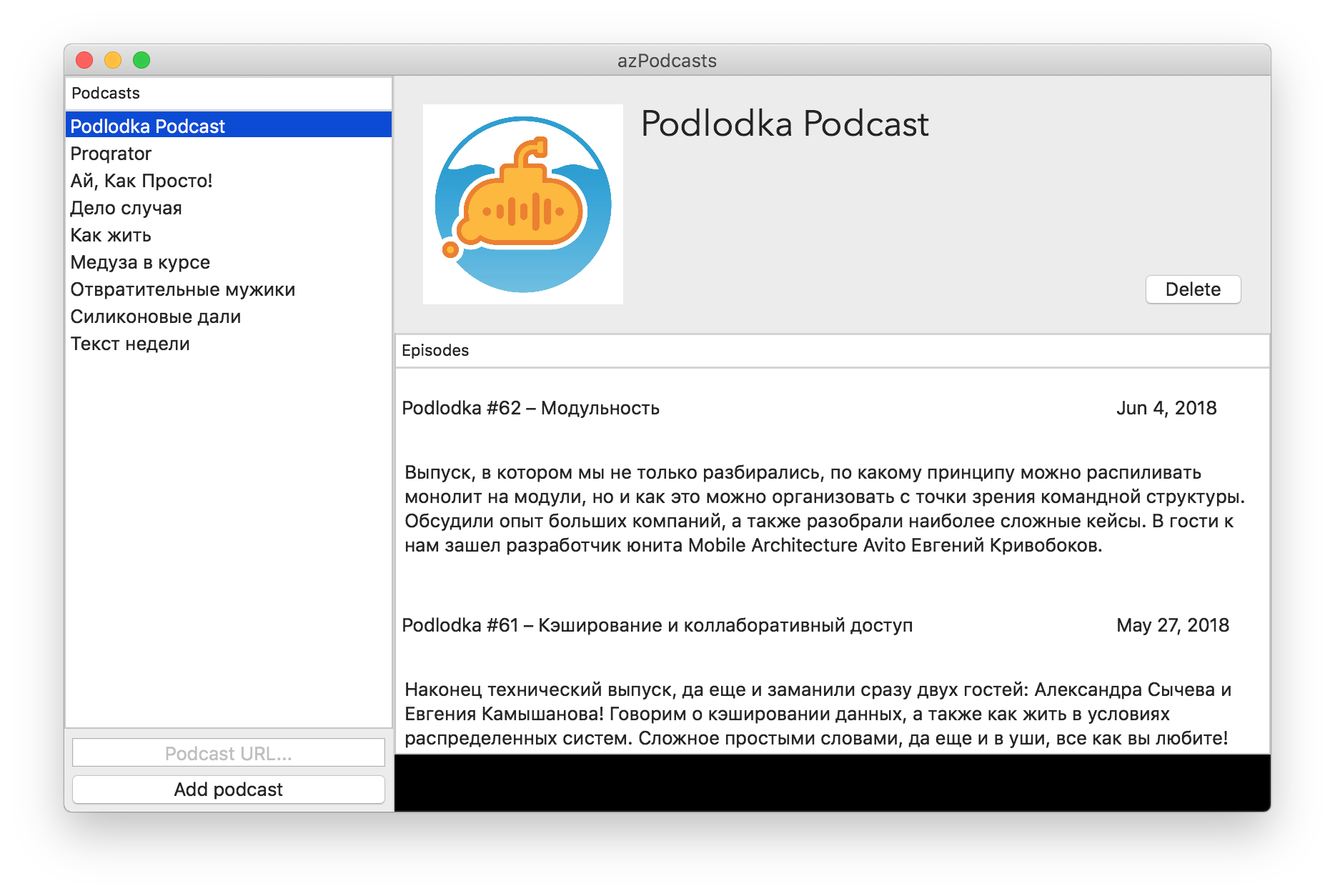


Рисунок 5.4 – Подробная информация о подкасте

Пользователь может удалить подкаст нажатием на кнопку «Delete», что уберёт его из списка избранных подкастов и очистит экран. При нажатии на выпуск подкаста начинается его загрузка и воспроизведение, появляются компоненты управления процессом воспроизведением. (Рисунок 5.5) Пользователь может приостанавливать воспроизведение, перематывать, менять громкость.

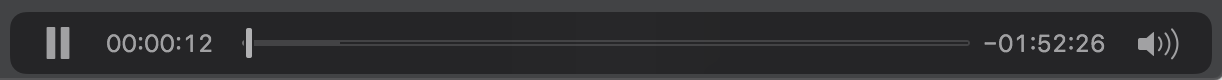


Рисунок 5.5 – Главное окно приложения**ЗАКЛЮЧЕНИЕ**

В результате работы над курсовым проектом было разработано клиентское приложение для прослушивания подкастов. Данное приложение в полной мере реализует все требования, сформулированные в условии курсового проекта, обладает интуитивно понятным пользовательским интерфейсом и предоставляет необходимый функционал.

Для создания программного комплекса были исследованы язык разметки XML, формат описания лент новостей, статей и подкастов RSS, возможности языка Swift по работе с сетью и мультимедиа.

В ходе разработки были изучены возможности визуальной среды разработки Xcode 9, получены навыки в создании качественных оконных приложений для операционной системы macOS. При создании архитектуры программного средства использовался шаблон проектирования MVC, а также изучены возможности по работе с XML, предоставляемые библиотекой SWXMLHash.

Работа была разделена на этапы, такие как анализ прототипов, литературных источников и постановка требований к проектируемому программному средству, разработка алгоритма и его схемы, конструирование программного средства, отладка и тестирование. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее программное средство.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] **Официальное приложение iTunes для ОС macOS 10.14** – Режим доступа:

https://itunes.apple.com- Дата доступа: 18.05.18

[2] **Приложение Castro для iOS 11** – Режим доступа:

http://supertop.co/castro/- Дата доступа: 18.05.18

[3] ГОСТ 19.701–90 (ИСО 5807–85) [Текст]. – Единая система программной документации: Сб. ГОСТов. - М.: Стандартинформ, 2005 с.

[4] Model–view–controller | Wikipedia – Режим доступа: https://en.wikipedia.org/wiki/Model–view–controller Дата доступа: 21.05.18.

[5] **Podcast Feed Best Practice**– Режим доступа: https://github.com/gpodder/podcast-feed-best-practice/blob/master/podcast-feed-best-practice.md- Дата доступа: 21.05.18.

[6] **Cocoa Application Layer | Apple Documentation** – Режим доступа: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX\_Technology\_Overview/CocoaApplicationLayer/CocoaApplicationLayer.html- Дата доступа: 25.05.18.

[7] **Документация по работе с SWXMLHash**– Режим доступа: https://github.com/drmohundro/SWXMLHash- Дата доступа: 21.05.18.

**ПРИЛОЖЕНИЕ А**

**Текст программного модуля** AppDelegate.swift

import Cocoa

@NSApplicationMain

class AppDelegate: NSObject, NSApplicationDelegate {

// MARK: - Core Data stack

lazy var persistentContainer: NSPersistentContainer = {

let container = NSPersistentContainer(name: "azPodcasts")

container.loadPersistentStores(completionHandler: { (storeDescription, error) in

if let error = error {

fatalError("Unresolved error \(error)")

}

})

return container

}()

// MARK: - Core Data Saving and Undo support

@IBAction func saveAction(\_ sender: AnyObject?) {

let context = persistentContainer.viewContext

if !context.commitEditing() {

NSLog("\(NSStringFromClass(type(of: self))) unable to commit editing before saving")

}

if context.hasChanges {

do {

try context.save()

} catch {

let nserror = error as NSError

NSApplication.shared.presentError(nserror)

}

}

}

func windowWillReturnUndoManager(window: NSWindow) -> UndoManager? {

return persistentContainer.viewContext.undoManager

}

func applicationShouldTerminate(\_ sender: NSApplication) -> NSApplication.TerminateReply {

let context = persistentContainer.viewContext

if !context.commitEditing() {

NSLog("\(NSStringFromClass(type(of: self))) unable to commit editing to terminate")

return .terminateCancel

}

if !context.hasChanges {

return .terminateNow

}

do {

try context.save()

} catch {

let nserror = error as NSError

let result = sender.presentError(nserror)

if (result) {

return .terminateCancel

}

let question = NSLocalizedString("Could not save changes while quitting. Quit anyway?", comment: "Quit without saves error question message")

let info = NSLocalizedString("Quitting now will lose any changes you have made since the last successful save", comment: "Quit without saves error question info");

let quitButton = NSLocalizedString("Quit anyway", comment: "Quit anyway button title")

let cancelButton = NSLocalizedString("Cancel", comment: "Cancel button title")

let alert = NSAlert()

alert.messageText = question

alert.informativeText = info

alert.addButton(withTitle: quitButton)

alert.addButton(withTitle: cancelButton)

let answer = alert.runModal()

if answer == .alertSecondButtonReturn {

return .terminateCancel

}

}

// If we got here, it is time to quit.

return .terminateNow

}

}

**ПРИЛОЖЕНИЕ Б**

**Текст программного модуля** SplitVC.swift

import Cocoa

class SplitVC: NSSplitViewController {

@IBOutlet weak var podcastsItem: NSSplitViewItem!

@IBOutlet weak var detailedViewItem: NSSplitViewItem!

override func viewDidLoad() {

super.viewDidLoad()

guard let podcastsVC = self.podcastsItem.viewController as? PodcastsVC,

let detailedVC = self.detailedViewItem.viewController as? PodcastDetailedVC else { return }

podcastsVC.detailedVC = detailedVC

detailedVC.podcasts = podcastsVC

}

}

**ПРИЛОЖЕНИЕ В**

**Текст программного модуля** Episode.swift

import Foundation

class Episode {

var title = ""

var pubdate = Date()

var description = ""

var audioURL = ""

static let formatter: DateFormatter = {

let formatter = DateFormatter()

formatter.dateFormat = "EEE, dd MMM yyyy HH:mm:ss zzz"

return formatter

}()

}

**ПРИЛОЖЕНИЕ Г**

**Текст программного модуля**  Parser.swift

import Foundation

import SWXMLHash

class Parser {

func getPodcastMetaData(data: Data) -> (title: String?, imageURL: String?) {

let xml = SWXMLHash.parse(data)

return (xml["rss"]["channel"]["title"].element?.text, xml["rss"]["channel"]["itunes:image"].element?.attribute(by: "href")?.text)

}

func getEpisodes(data: Data) -> [Episode] {

var episodes = [Episode]()

let xml = SWXMLHash.parse(data)

for item in xml["rss"]["channel"]["item"].all {

let episode = Episode()

if let title = item["title"].element?.text {

episode.title = title

}

if let description = item["description"].element?.text {

episode.description = description

}

if let description = item["description"].element?.text {

episode.description = description

}

if let audioURL = item["enclosure"].element?.attribute(by: "url")?.text {

episode.audioURL = audioURL

}

if let pubdateString = item["pubDate"].element?.text {

episode.pubdate = Episode.formatter.date(from: pubdateString) ?? Date()

}

episodes.append(episode)

print(episode.pubdate)

}

return episodes

}

}

**ПРИЛОЖЕНИЕ Д**

**Текст программного модуля** Podcasts.swift

import Cocoa

class PodcastsVC: NSViewController {

@IBOutlet weak var podcastURLTextField: NSTextField!

@IBOutlet weak var tableView: NSTableView!

var podcasts = [Podcast]()

var detailedVC: PodcastDetailedVC? = nil

override func viewDidLoad() {

super.viewDidLoad()

tableView.delegate = self

tableView.dataSource = self

getPodcasts()

}

@IBAction func addPodcastClicked(\_ sender: Any) {

guard let url = URL(string: podcastURLTextField.stringValue) else { return }

guard isPodcastExsists(withURL: url.absoluteString) == false else {

showAlert(withAlertStyle: .critical, header: "Podcast aleready exists!", body: "There is already a podcast with the same url.")

return

}

URLSession.shared.dataTask(with: url) { (data, response, error) in

guard error == nil, data != nil else { return }

let parser = Parser()

let info = parser.getPodcastMetaData(data: data!)

guard info.title != nil else {

self.showAlert(withAlertStyle: .critical, header: "Cannot open RSS file!", body: "There is an error parsing your podcast link, please, check it and try again.")

return

}

guard let appDelegate = NSApplication.shared.delegate as? AppDelegate else { return }

let managedContext = appDelegate.persistentContainer.viewContext

let podcast = Podcast(context: managedContext)

podcast.url = url.absoluteString

podcast.imageURL = info.imageURL

podcast.title = info.title

appDelegate.saveAction(nil)

self.getPodcasts()

}.resume()

podcastURLTextField.stringValue = ""

}

func getPodcasts() {

guard let appDelegate = NSApplication.shared.delegate as? AppDelegate else { return }

let managedContext = appDelegate.persistentContainer.viewContext

let fetchRequest: NSFetchRequest<Podcast> = Podcast.fetchRequest()

fetchRequest.sortDescriptors = [NSSortDescriptor(key: "title", ascending: true)]

do {

podcasts = try managedContext.fetch(fetchRequest)

print(podcasts)

} catch {

print(error)

}

DispatchQueue.main.async(execute: {

self.tableView.reloadData()

})

}

func isPodcastExsists(withURL url: String) -> Bool {

guard let appDelegate = NSApplication.shared.delegate as? AppDelegate else { return false }

let managedContext = appDelegate.persistentContainer.viewContext

let fetchRequest: NSFetchRequest<Podcast> = Podcast.fetchRequest()

fetchRequest.predicate = NSPredicate(format: "url == %@", url)

do {

let matchingPodcasts = try managedContext.fetch(fetchRequest)

if matchingPodcasts.count >= 1 {

return true

} else {

return false

}

} catch {

print(error)

}

return false

}

func showAlert(withAlertStyle alertStyle: NSAlert.Style, header: String, body: String) {

let alert = NSAlert.init()

alert.alertStyle = alertStyle

alert.messageText = header

alert.informativeText = body

alert.addButton(withTitle: "OK")

DispatchQueue.main.async {

alert.runModal()

}

}

}

extension PodcastsVC: NSTableViewDelegate, NSTableViewDataSource {

func numberOfRows(in tableView: NSTableView) -> Int {

return podcasts.count

}

func tableView(\_ tableView: NSTableView, viewFor tableColumn: NSTableColumn?, row: Int) -> NSView? {

let cell = tableView.makeView(withIdentifier: NSUserInterfaceItemIdentifier.init("podcastCell"), owner: self) as? NSTableCellView

let currentPodcast = podcasts[row]

if currentPodcast.title != nil {

cell?.textField?.stringValue = currentPodcast.title!

} else {

cell?.textField?.stringValue = "Unkown Title"

}

return cell

}

func tableViewSelectionDidChange(\_ notification: Notification) {

guard tableView.selectedRow >= 0 else { return }

let currentPodcast = podcasts[tableView.selectedRow]

detailedVC?.podcast = currentPodcast

detailedVC?.updateView()

}

}

**ПРИЛОЖЕНИЕ Е**

**Текст программного модуля** PodcastDetailedVC.swift

import Cocoa

import AVKit

class PodcastDetailedVC: NSViewController {

@IBOutlet weak var avPlayerView: AVPlayerView!

@IBOutlet weak var titleLabel: NSTextField!

@IBOutlet weak var imageView: NSImageView!

@IBOutlet weak var tableView: NSTableView!

@IBOutlet weak var deleteButton: NSButton!

var podcast: Podcast?

weak var podcasts: PodcastsVC?

var episodes = [Episode]()

var player: AVPlayer? = nil

override func viewDidLoad() {

super.viewDidLoad()

tableView.delegate = self

tableView.dataSource = self

updateView()

}

func updateView() {

if podcast != nil {

if podcast?.title != nil {

titleLabel.stringValue = podcast!.title!

} else {

titleLabel.stringValue = ""

}

if podcast?.imageURL != nil {

imageView.image = NSImage(byReferencing: URL(string: podcast!.imageURL!)!)

} else {

imageView.image = nil

}

tableView.isHidden = false

deleteButton.isHidden = false

titleLabel.isHidden = false

imageView.isHidden = false

} else {

imageView.isHidden = true

tableView.isHidden = true

deleteButton.isHidden = true

titleLabel.isHidden = true

}

getEpisodes()

}

func getEpisodes() {

if podcast?.url != nil {

if let url = URL(string: podcast!.url!) {

URLSession.shared.dataTask(with: url) { (data, response, error) in

if error != nil {

print(error!.localizedDescription)

} else {

if data != nil {

let parser = Parser()

self.episodes = parser.getEpisodes(data: data!)

DispatchQueue.main.async {

self.tableView.reloadData()

}

}

}

}.resume()

}

}

}

@IBAction func deleteButtonClicked(\_ sender: Any) {

guard let appDelegate = NSApplication.shared.delegate as? AppDelegate else { return }

let managedContext = appDelegate.persistentContainer.viewContext

if podcast != nil {

managedContext.delete(podcast!)

appDelegate.saveAction(self)

podcasts?.getPodcasts()

}

podcast = nil

updateView()

}

}

extension PodcastDetailedVC: NSTableViewDataSource, NSTableViewDelegate {

func numberOfRows(in tableView: NSTableView) -> Int {

return episodes.count

}

func tableView(\_ tableView: NSTableView, heightOfRow row: Int) -> CGFloat {

return 150

}

func tableView(\_ tableView: NSTableView, viewFor tableColumn: NSTableColumn?, row: Int) -> NSView? {

if let cell = tableView.makeView(withIdentifier: NSUserInterfaceItemIdentifier(rawValue: "episodeCell"), owner: self) as? EpisodeCell {

let episode = episodes[row]

cell.titleLabel.stringValue = episode.title

cell.descriptionLabel.stringValue = episode.description

let dateFormatter = DateFormatter()

dateFormatter.dateFormat = "MMM d, yyyy"

cell.dateLabel.stringValue = dateFormatter.string(from: episode.pubdate)

return cell

}

return NSTableCellView()

}

func tableView(\_ tableView: NSTableView, rowViewForRow row: Int) -> NSTableRowView? {

return MyNSTableRowView()

}

func tableViewSelectionDidChange(\_ notification: Notification) {

guard tableView.selectedRow >= 0 else { return }

let episode = episodes[tableView.selectedRow]

player?.pause()

player = nil

if let audioURL = URL(string: episode.audioURL) {

player = AVPlayer(url: audioURL)

avPlayerView.player = player

player?.play()

}

}

}

class MyNSTableRowView: NSTableRowView {

override func draw(\_ dirtyRect: NSRect) {

super.draw(dirtyRect)

self.isEmphasized = false

}

}

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КП 1–40 01 01 001 ПЗ | | | | Пояснительная записка | | | | 38 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР 651001 001 ПД | | | | Схема программы | | | | **Формат А1** | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | **БГУИР КП 1-40 01 01 001 Д1** | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Проигрыватель подкастов  Ведомость курсового  проекта |  | | | | Лист | Листов |
| Разраб. | | Азаров А. А. |  |  | Т |  | |  | 38 | 38 |
| Пров. | | Красковский П.Н. |  |  | Кафедра ПОИТ  гр. 651001 | | | | | |
|  | |  |  |  |