

## Содержание

Введение . . . . .	4
1 Описание предметной области . . . . .	5
1.1 Сервис Soundcloud и его особенности . . . . .	5
1.2 Медиа-сервер MPD . . . . .	6
1.3 Интернет-проигрыватель Mopidy . . . . .	7
1.4 Web-приложение Cmd.fm . . . . .	8
1.5 Текстовые интерфейсы . . . . .	9
2 Архитектура приложения . . . . .	11
2.1 Сетевое взаимодействие компонентов системы . . . . .	12
2.2 Структура команд сервера . . . . .	13
2.3 Контроллер . . . . .	14
2.4 Модель . . . . .	15
2.5 Представление . . . . .	16
3 Описание программного обеспечения . . . . .	19
3.1 Физическое описание системы . . . . .	19
3.2 Язык программирования JavaScript . . . . .	20
3.3 Платформа Node.js . . . . .	21
3.4 Принцип работы EventLoop . . . . .	22
3.5 Обратные вызовы и паттерн Promise . . . . .	22
3.6 Очередь сообщений ZMQ . . . . .	23
3.7 Физическая архитектура . . . . .	23
3.8 Технические требования . . . . .	24
Заключение . . . . .	25
Список использованных источников . . . . .	26
Приложение А Задание на бакалаврскую работу . . . . .	28
Приложение Б Руководство пользователя . . . . .	28
Приложение В Код программного продукта . . . . .	33

## Введение

В наши дни все больше и больше людей пользуются онлайн сервисами для прослушивания музыки. Это очень удобно, так как музыку можно слушать с любого устройства с доступом в сеть. Если раньше для размещения большой коллекции было необходимо выделять значительные ресурсы на локальных носителях и принимать определенные меры для их сохранности, то теперь в этом нет необходимости.

Одним из самых популярных сервисов онлайн музыки является Soundcloud [2]. Для удобства доступа к содержимому сайта существует множество клиентов. Однако, не существует такого клиента, который бы одновременно сочетал в себе:

- клиент-серверную архитектуру,
- наличие текстового интерфейса доступа,
- характерную для Soundcloud работу с комментариями к композиции.

В данной работе была спроектирована и разработана система, реализующая все указанные выше требования.

# 1 Описание предметной области

## 1.1 Сервис Soundcloud и его особенности

На сайте *Soundcloud* каждый исполнитель может выложить свои композиции для всеобщего прослушивания и скачивания. Он отличается от всех музыкальных сайтов тем, что позволяет привязывать комментарий не только к целой композиции, но и к отдельным ее частям.

На каждой web-странице композиции есть виджет музыкального для её воспроизведения.

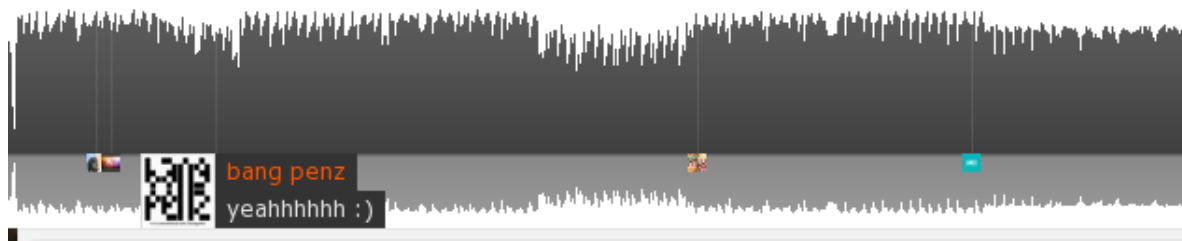


Рисунок 1.1 – Виджет плеера на Soundcloud

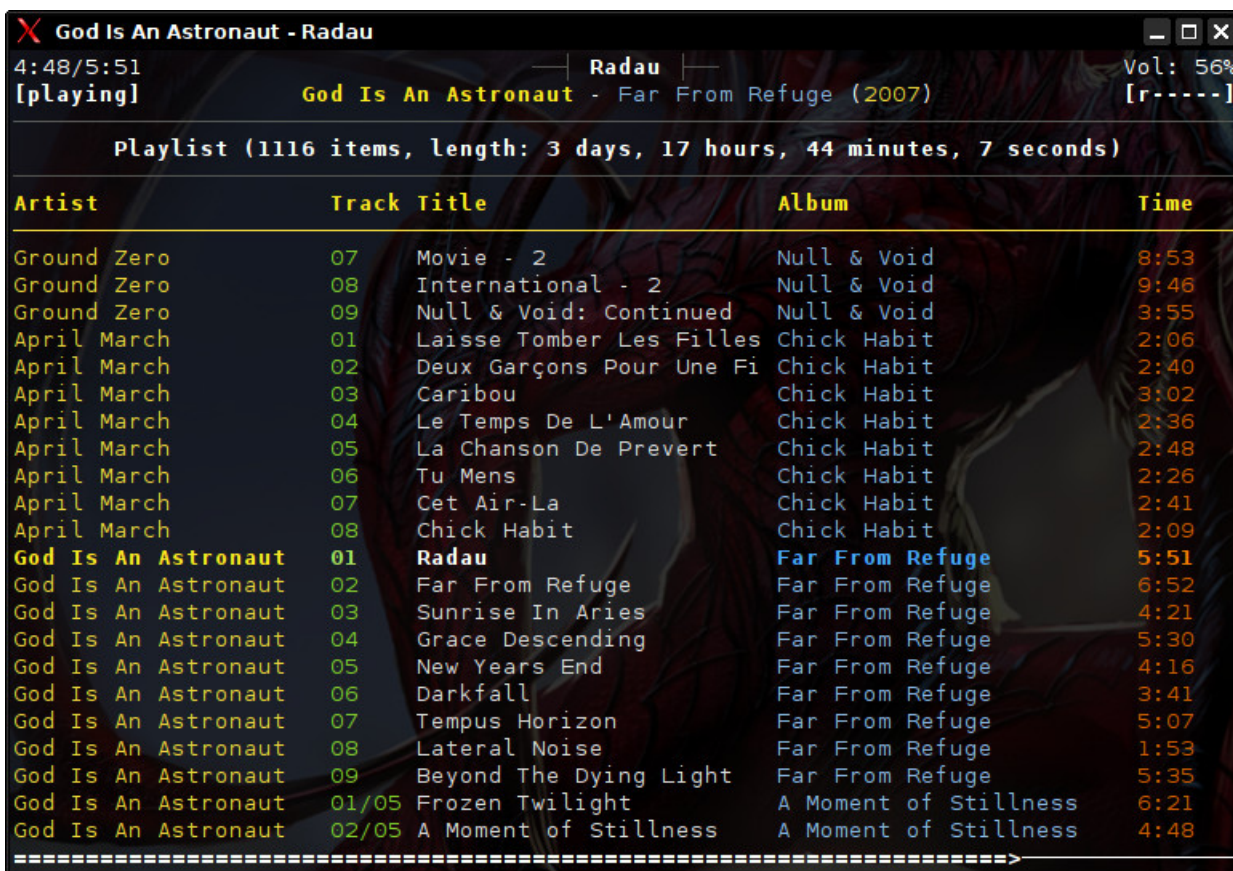
На виджете в местах, соответствующих временным отметкам расположены иконки пользователей, оставивших в этом месте свои комментарии. В момент когда проигрываемая композиция проходит отметку с иконкой пользователя над ней показывается соответствующий этой отметке комментарий. Таким образом, прослушивая трек можно узнать о мнениях пользователей не только насчет всей композиции, но и об отдельных её частях. Это не только очень актуально для музыкантов и продюсеров, но также интересно и для обычных слушателей.

Сервис предоставляет удобное API и SDK для доступа к данным сайта, что позволяет создавать приложения на любой платформе. И существует множество мобильных, настольных и web-приложений, которые пользуются данными сервиса и предоставляют доступ к управлению его содержимым.

Рассмотрим приложения, которые больше всего подходят под выдвинутые во введении требования.

## 1.2 Медиа-сервер MPD

*Music Player Daemon* [5] — музыкальный проигрыватель с клиент-серверной архитектурой. Предназначен для воспроизведения музыкальных файлов, расположенных на локальных носителях. Серверная часть воспроизводит музыку и отвечает на запросы от клиентов, а клиенты управляют воспроизведением и отображают текущий статус проигрывателя. В настоящий момент реализовано множество клиентов, реализующих различные парадигмы интерфейса под множество платформ. На сегодняшний день наиболее продвинутым текстовым клиентом является *ncstrcp* [6].



The screenshot shows a terminal window titled "God Is An Astronaut - Radau". The interface includes a progress bar at the top showing "4:48/5:51" and "[playing]". The current track is "God Is An Astronaut - Far From Refuge (2007)" with a volume of "56%". Below this, a playlist is displayed with 1116 items, totaling 3 days, 17 hours, 44 minutes, and 7 seconds. The playlist is a table with columns for Artist, Track, Title, Album, and Time.

Artist	Track	Title	Album	Time
Ground Zero	07	Movie - 2	Null & Void	8:53
Ground Zero	08	International - 2	Null & Void	9:46
Ground Zero	09	Null & Void: Continued	Null & Void	3:55
April March	01	Laisse Tomber Les Filles	Chick Habit	2:06
April March	02	Deux Garçons Pour Une Fi	Chick Habit	2:40
April March	03	Caribou	Chick Habit	3:02
April March	04	Le Temps De L'Amour	Chick Habit	2:36
April March	05	La Chanson De Prevert	Chick Habit	2:48
April March	06	Tu Mens	Chick Habit	2:26
April March	07	Cet Air-La	Chick Habit	2:41
April March	08	Chick Habit	Chick Habit	2:09
God Is An Astronaut	01	Radau	Far From Refuge	5:51
God Is An Astronaut	02	Far From Refuge	Far From Refuge	6:52
God Is An Astronaut	03	Sunrise In Aries	Far From Refuge	4:21
God Is An Astronaut	04	Grace Descending	Far From Refuge	5:30
God Is An Astronaut	05	New Years End	Far From Refuge	4:16
God Is An Astronaut	06	Darkfall	Far From Refuge	3:41
God Is An Astronaut	07	Tempus Horizon	Far From Refuge	5:07
God Is An Astronaut	08	Lateral Noise	Far From Refuge	1:53
God Is An Astronaut	09	Beyond The Dying Light	Far From Refuge	5:35
God Is An Astronaut	01/05	Frozen Twilight	A Moment of Stillness	6:21
God Is An Astronaut	02/05	A Moment of Stillness	A Moment of Stillness	4:48

Рисунок 1.2 – Плейлист ncstrcp

*Ncstrcp* поддерживает большинство возможностей *MPD*, имеет логичное устройство внутренних разделов и интуитивно понятный интерфейс. По умолчанию поддерживаются сочетания клавиш *Vim* и *Emacs*, являющиеся традиционными для текстовых интерфейсов.

Поддержка *Soundcloud* для *MPD* реализована в виде неофициального патча от одного из пользователей [7]. Данный патч добавляет возможность проигрывать треки путем отправки на сервер команды следующего вида:

```
mpc load soundcloud://track/≤track-id>
```

К сожалению, других возможностей *Soundcloud*, включая возможность просмотра комментариев к композициям, этот патч не предусматривает. Более того, для реализации этих функций необходимы не только изменения кода в серверной компоненте *MPD*, но также и реализация отдельного клиента с поддержкой этих функций.

### 1.3 Интернет-проигрыватель Moridy

*Mopidy* [4] — это музыкальный сервер с гибкими возможностями расширения. *Mopidy* может проигрывать музыку из множества различных источников, в том числе и с *Soundcloud*. *Mopidy* реализует протокол клиент-серверного взаимодействия проигрывателя *MPD*, что позволяет управлять воспроизведением с помощью любого из множества клиентов, совместимого с протоколом проигрывателя *MPD*.

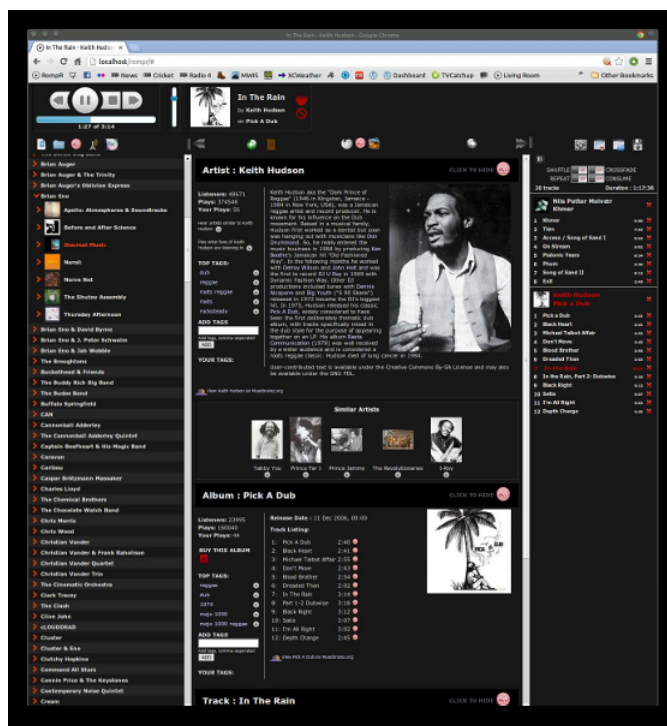
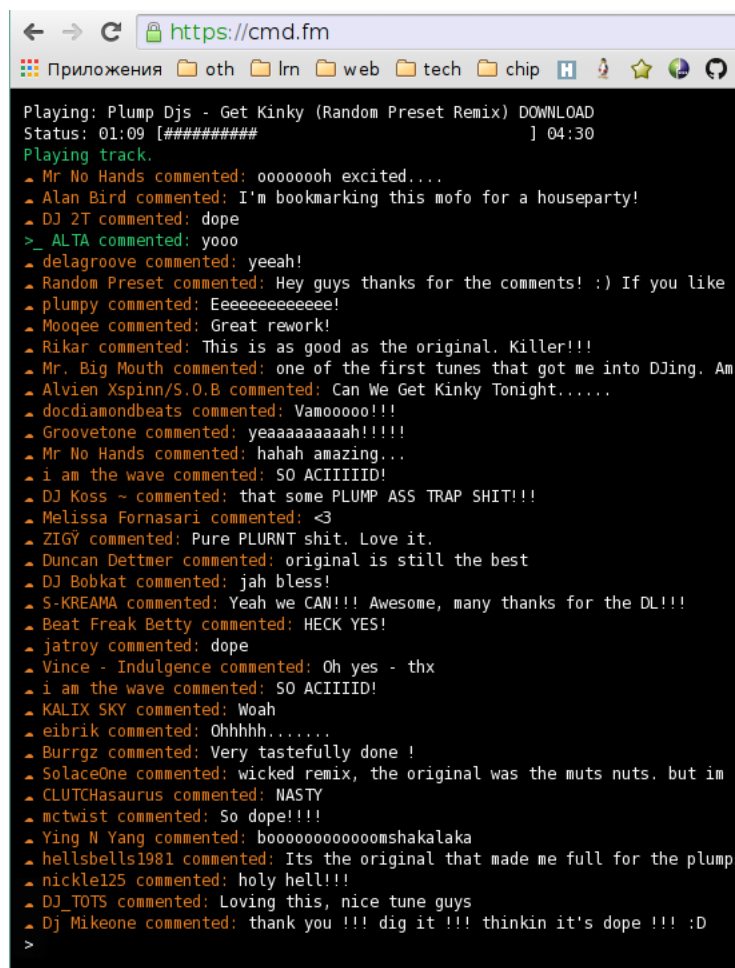


Рисунок 1.3 – Web-интерфейс Moridy

В силу этой особенности *Mopidy* не может отображать комментарии к композициям во время воспроизведения, так как протокол проигрывателя *MPD* не предусматривает просмотр и вывод комментариев в соответствии с текущим временем.

## 1.4 Web-приложение Cmd.fm

Главным и практически единственным мощным аналогом является сайт *cmd.fm* [3]. Этот сайт реализует радио на основе доступных на *Soundcloud* композиций. Интерфейс реализован в виде web-приложения и представляет собой имитацию текстового терминала.



```
← → ↻ https://cmd.fm
Приложения oth lrn web tech chip H ☆ ↻

Playing: Plump Djs - Get Kinky (Random Preset Remix) DOWNLOAD
Status: 01:09 [#####] 04:30
Playing track.
▶ Mr No Hands commented: ooooooh excited...
▶ Alan Bird commented: I'm bookmarking this mofo for a houseparty!
▶ DJ 2T commented: dope
> _ALTA commented: yooo
▶ delagroove commented: yeeah!
▶ Random Preset commented: Hey guys thanks for the comments! :) If you like
▶ plumpy commented: Eeeeeeeeeeeee!
▶ Mooqee commented: Great rework!
▶ Rikar commented: This is as good as the original. Killer!!!
▶ Mr. Big Mouth commented: one of the first tunes that got me into DJing. Am
▶ Alvien Xspinn/S.O.B commented: Can We Get Kinky Tonight.....
▶ docdiamondbeats commented: Vamooooo!!!
▶ Groovetone commented: yeaaaaaaaah!!!!
▶ Mr No Hands commented: hahah amazing...
▶ i am the wave commented: SO ACIIIIID!
▶ DJ Koss ~ commented: that some PLUMP ASS TRAP SHIT!!!
▶ Melissa Fornasari commented: <3
▶ ZIGY commented: Pure PLURNT shit. Love it.
▶ Duncan Dettmer commented: original is still the best
▶ DJ Bobkat commented: jah bless!
▶ S-KREAMA commented: Yeah we CAN!!! Awesome, many thanks for the DL!!!
▶ Beat Freak Betty commented: HECK YES!
▶ jatroy commented: dope
▶ Vince - Indulgence commented: Oh yes - thx
▶ i am the wave commented: SO ACIIIIID!
▶ KALIX SKY commented: Woah
▶ eibrik commented: Ohhhh.....
▶ Burrgez commented: Very tastefully done !
▶ SolaceOne commented: wicked remix, the original was the muts nuts. but im
▶ CLUTCHasaurus commented: NASTY
▶ mctwist commented: So dope!!!!
▶ Ying N Yang commented: boooooooooooooomshakalaka
▶ hellsbells1981 commented: Its the original that made me full for the plump
▶ nickle125 commented: holy hell!!!
▶ DJ_TOTS commented: Loving this, nice tune guys
▶ Dj Mikeone commented: thank you !!! dig it !!! thinkin it's dope !!! :D
>
```

Рисунок 1.4 – Страница комментариев cmd.fm

Команда *play* запускает радио, выбрав в плейлист композиции, соответствующие указанному жанру. Также доступна интерактивная справка по командам и другие

элементы, характерные для каноничных приложений с текстовым интерфейсом.

Многим пользователям этого сайта очень понравилась [17] реализация основной особенности *Soundcloud* — комментариев к композиции: по мере проигрывания в консоль плеера выводятся комментарии, оставленные пользователями в этот момент.

Долгое время пользователи *cmd.fm* ждали появления аналогичного приложения в формате текстового приложения для эмулятора терминала. В данный момент проект находится на стадии  $\beta$ -тестирования и доступно только для платформы MacOS X. Код приложения не доступен в открытом доступе, поэтому изменить архитектуру приложения на клиент-серверную не представляется возможным.

## 1.5 Текстовые интерфейсы

Консольные приложения — одна из важнейших вех в истории развития компьютеров. До появления графических интерфейсов консоль была единственным средством общения с ЭВМ и даже сейчас, в эпоху победившего GUI, или графического интерфейса, консоль никуда не исчезла. Конечно, консольные приложения, как правило, не обладают таким красивым и приятным для глаз интерфейсом. Не всегда сразу понятно как ими пользоваться, особенно для неискушенных пользователей, — приложения с GUI более очевидны для восприятия, с GUI легко начать работать. Однако пользователи, освоившие текстовый интерфейс, выполняют задачи гораздо быстрее, так как не тратят время на поиск пунктов меню, не сосредотачиваются на «прицеливании» в мелкие элементы интерфейса, их внимание не рассеивается на графические декорации [9]. Операция в текстовом интерфейсе представляет из себя нажатие одной или нескольких клавиш, интерфейс минималистичен и содержит в себе лишь необходимые элементы, позволяя сосредоточиться на поставленной задаче и выполнить её максимально быстро.

Одной из мощнейших возможностей программ с текстовым интерфейсом является возможность объединять несколько программ в конвейер. Конвейер [8] в терминологии UNIX — некоторое множество процессов, для которых выполнено следующее перенаправление ввода-вывода: то, что выводит на поток стандартного вывода предыдущий процесс, попадает в поток стандартного ввода следующего процесса.

Механизм конвейеров добавляет гибкость процессу взаимодействия программ. Все программы оперируют текстовыми данными, т.е. получают на вход текст и выводят его же, причем, обработка данных внутри программы идет «в потоке». Данные как-бы «протекают» через цепочку программ, разделенных символом конвейера в командной строке, при этом обрабатываемый текст не копируется каждый раз от операции к операции — данные модифицируются «на месте». Таким образом, имея некоторый набор простых утилит для обработки текста в потоке, пользователь может «собрать» себе новую программу под свои задачи, просто объединив конвейером несколько программ с помощью символа |

```
tail -f /var/log/Xorg.0.log | grep EE
```

Кроме того, программы с текстовым интерфейсом очень быстро работают по сети, то есть в случае, когда приложение запускается на удаленном сервере и изменения интерфейса передаются по сети. Для работы с интерфейсом GUI по каналам связи необходимо передавать растровую графику. Объем информации для передачи уменьшится на несколько порядков, если воспользоваться текстовым интерфейсом. Кроме того, текст гораздо лучше поддается сжатию.

Таким образом, CLI-интерфейс, или интерфейс командной строки, является одним из наиболее гибких средств для создания прикладных, и, в особенности, служебных приложений, а также предоставляет удобнейшее средство для применения модульного подхода к построению архитектуры программных систем.

Такой интерфейс приложений особо востребован среди пользователей \*nix систем. Поэтому создание консольного проигрывателя для *Soundcloud* для ОС Linux является актуальной задачей.



## 2 Архитектура приложения

Разработанный плеер имеет клиент-серверную архитектуру, реализующую паттерн MVC: серверная служба будет выступать в роли контроллера, модель – компонент для связи с API, а клиенты выступают в роли представлений.

Model-view-controller (MVC, «модель-представление-поведение») — схема использования нескольких шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. [11]

Такая архитектура выбрана из соображений расширения и масштабируемости, так как позволяет создавать новые клиенты на разных платформах абстрагируясь от реализации доступа к данным сервиса. Именно это является ключевым отличием от немногочисленных аналогов, имеющих монолитную архитектуру и привязанных к одной платформе.

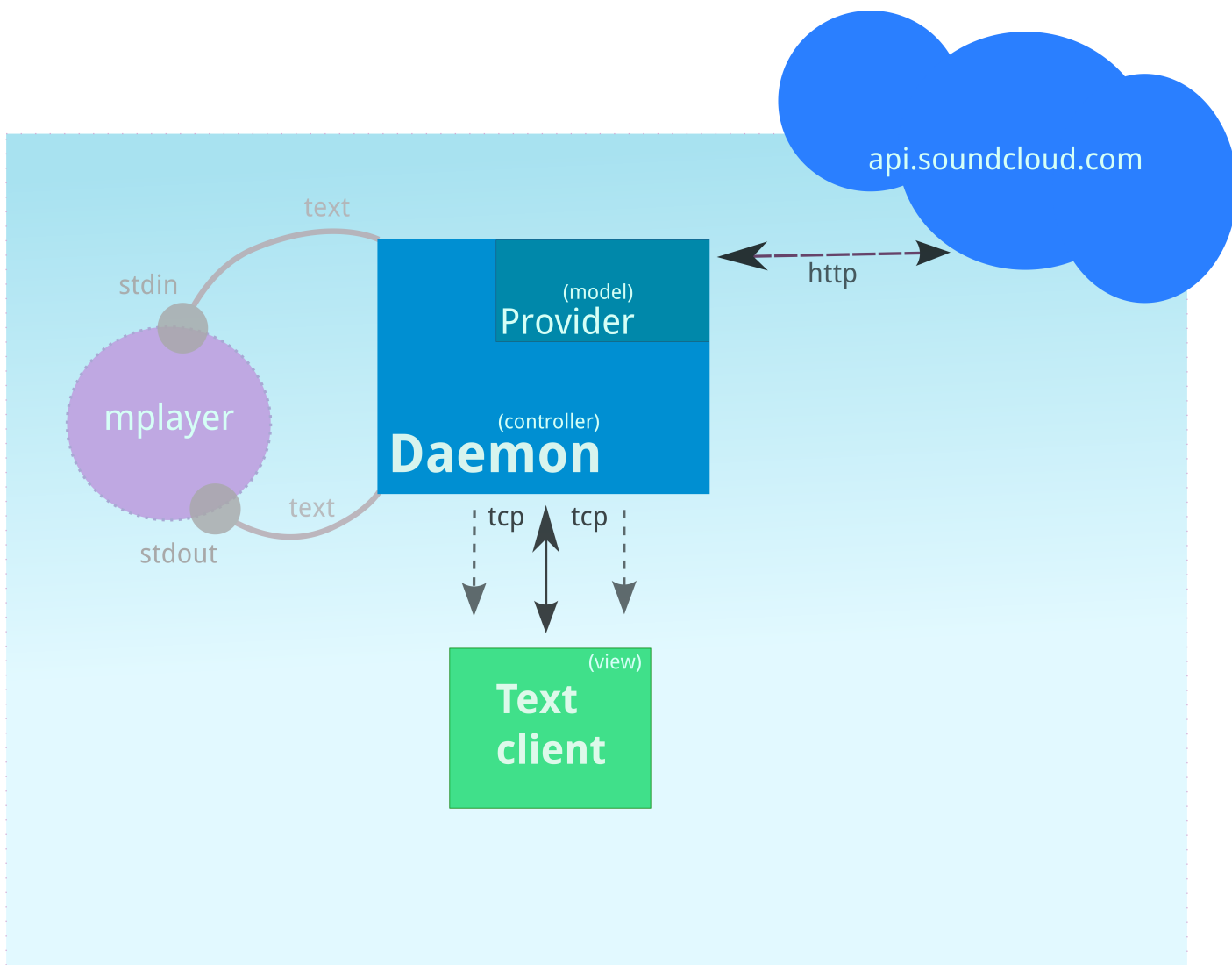


Рисунок 2.1 – Общая архитектура системы

На уровне архитектуры не предусмотрено механизмов разграничения прав доступа. Это сделано для упрощения модели: предполагается, что класс приложений, построенных на этой архитектуре, не нуждается в правах доступа к ресурсам. В случае необходимости их можно реализовать на уровне модели.

## 2.1 Сетевое взаимодействие компонентов системы

Сервер и клиенты общаются между собой по сети. Когда в сети появляется новый клиент, он подписывается на интересующие его события проигрывателя, такие как:

- началось воспроизведение новой композиции
- сменился список воспроизведения
- сменилась временная позиция

Таким образом, взаимодействие между клиентами и сервером происходит по принципу *publish/subscribe*.

Когда клиент только появляется в сети или запрашивает данные, взаимодействие между клиентом и сервером осуществляется по принципу *socket* — двусторонняя связь между новым клиентом и сервером.

По команде *getCommandList* или при запросе несуществующей команды сервер возвращает код ошибки и список доступных команд. Такое архитектурное решение было выбрано с целью обеспечить возможность легкой расширяемости системы команд сервера. С помощью списка команд клиент узнаёт о возможностях сервера и может решить, какие функции будут доступны в интерфейсе, а какие нужно убрать. Например, если сервер не отправляет информации о текущей временной позиции, то зависящие от неё элементы можно исключить или выдать сообщение, что запуск клиента не возможен. В то же время простым клиентам, которые могут только переключить композицию на следующий этап функция вообще не известна, они работают и без неё — им нужно лишь информация о доступности команды *next* на сервере.

## 2.2 Структура команд сервера

Клиент формирует запросы серверу в виде команд. Структура команды запроса к серверу содержит в себе название команды и список параметров в формате JSON:

```
{  
  "name": "play",  
  "params": {  
    "tag": "rock"  
  }  
}
```

При создании сервера определяется набор команд, реализующий функции проигрывания. Серверная команда включает в себя поля, идентичные полям команды запроса, описания параметров и функцию к исполнению:

```
playIndex: {
  name: "playIndex",
  params: { index: "index of playlist element"},

  exec: function (params) {

    var index = params.index * 1 || 0;

    model._currentTrackIndex = index;
    daemon.play();

    return 0;

  }
},
```

Функция *exec* обязательно должна возвращать результат — это может быть либо результат запроса, либо статус выполнения команды. В случае, если запрашиваемой команды не существует, клиенту возвращается код ошибки и список доступных команд.

## 2.3 Контроллер

Контроллер — это «ядро» плеера, его серверная часть. Именно он проигрывает музыку и отвечает на запросы от клиентов. Контроллер предоставляет доступ к сервисам модели и интерфейсы управления для клиентов. При запуске контроллер создает точку подключения и ожидает запросы клиентов. Контроллер реализован как традиционный процесс-демон (Daemon)(*Daemon*). При запуске контроллер создает точку подключения и ожидает запросы клиентов.

К функциям контроллера относятся обработка запросов клиентов (проиграть трек, пауза) и подписка клиентов на события (сменился трек, трек приостановлен). Клиент подключается по протоколу *tcp*. Это может быть соединение на основе запроса/ответа или постоянное подключение.

Для иерархических связей в сервере используется каноничная для *JavaScript* прототипная модель. Прототипное программирование — стиль объектно-ориентированного программирования, при котором отсутствует понятие класса, а наследование производится путём клонирования существующего экземпляра объекта — прототипа [10].

Для создания сервера используется базовый прототип в котором реализованы функции для формирования сообщений, их обработки и отправки. При создании экземпляра сервера базовый прототип расширяется массивом команд, которые он может обрабатывать.

## 2.4 Модель

Модель — это компонент, осуществляющий запросы к API *Soundcloud*. В логике модели скрыты функции получения и обработки запрашиваемых данных.

В приложении используется вариант пассивной архитектуры MVC: модель не подписывает каких-либо клиентов на изменение данных, все взаимодействия происходят по методу запрос-ответ между моделью и контроллером. Каждый элемент модели это команда, аналогичная по структуре команде контроллера.

*Soundcloud* предоставляет разработчикам приложений так называемое REST API для доступа к контенту. Это традиционный метод доступа к ресурсам сети Интернет — запрос формируется в виде URL-адреса с параметрами поиска требуемых данных:

<http://api.soundcloud.com/tracks.json?&tag=rock&order=hotness>

В модели определена команда *getTaggedList*, принимающая в качестве параметра музыкальный жанр. С помощью этой команды контроллер получает список композиций для воспроизведения, их адрес воспроизведения, комментарии к композиции, её длительность и информацию об авторе.

## 2.5 Представление

Представление является расширяемой частью используемой архитектуры. В проекте +реализован консольный клиент к разработанному серверу.

Клиентское приложение может использовать весь функционал сервера, либо только часть доступных функций. Для интеграции различных расширений сервера и клиента используется следующий подход. Клиент запрашивает список команд, доступных на сервере(`getCommandList`). Получив список команд клиент оценивает набор функций, который сможет предоставить пользователю и принимает решение о запуске своих модулей, отображении элементов интерфейса и о запуске приложения в целом.

Клиент может подписаться на интересующие его события, а может лишь выполнить одну команду и завершить работу. Клиент может и не требовать никаких команд вовсе, лишь принять список команд и выполнить команду пользователя из командной строки. Это самый простой тип клиентов, представляющих из себя «адаптер» текстового интерфейса доступа к контроллеру.

В ходе работы был реализован клиент с текстовым интерфейсом. Программа клиента состоит из описания компонентов интерфейса и функций обработки ввода пользователя. При запуске или изменении размеров окна эмулятора терминала производится расчет размеров, компоновка компонентов интерфейса на экране и инициализация обработчиков событий пользовательского ввода. Обработчики событий могут отправить серверу команду, активировать или спрятать элементы интерфейса.

Также есть обработчики сообщений сервера и объект запроса. При инициализации обработчики подписываются на интересующие клиент оповещения от сервера: переключение композиции, смена статуса воспроизведения, изменение текущей временной отметки проигрываемой композиции и обновляют соответствующие элементы интерфейса по мере их поступления. Объект запроса посылает текстовые команды серверу и обрабатывает ответ. Для обработки и оповещений и запросов используются одни и те же функции, так как получаемые сообщения в обоих случаях также идентичны, как и логика их обработки.

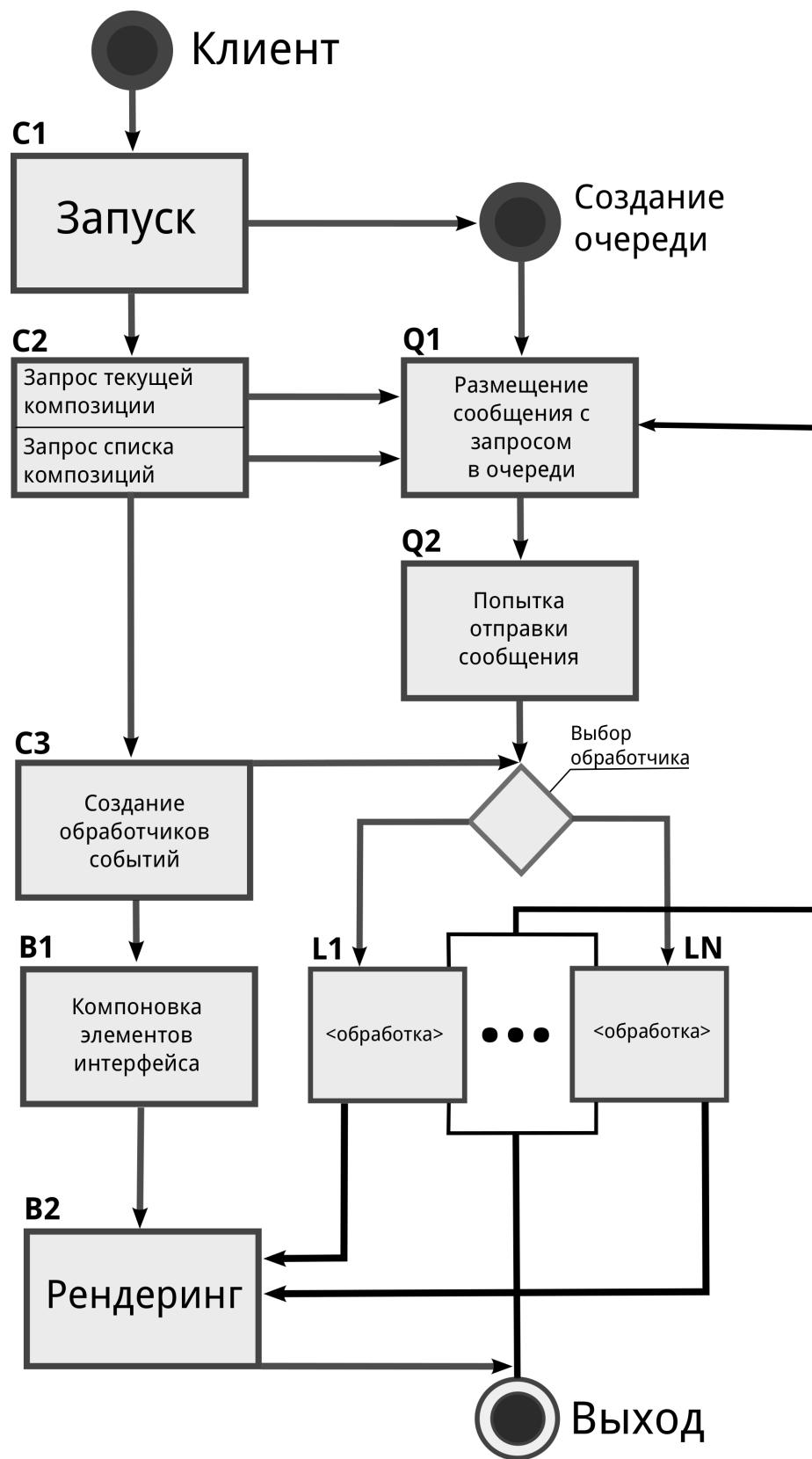


Рисунок 2.2 – Диаграмма деятельности клиента

Несколько слов о реализации вывода комментариев. Проигрыватель воспроизводит композиции из сети интернет. Соединение с интернетом не всегда стабильно, поэтому необходимо четко контролировать процесс вывода комментария в точности в тот момент, когда он был написан, — это является ключевой функцией приложения.

В клиент-серверном плеере MPD нет необходимости выводить комментарии и музыка проигрывается с локальных носителей. Поэтому в клиентах к MPD нет синхронизации времени с сервером, подсчет ведется внутри клиента и синхронизация происходит только если произойдет событие смены композиции, остановки или начала воспроизведения. В комментариях к исходным файлам клиента *Nstprcpp* эта функция называется «режим Idle». За счет сетевой передачи сигнала могут происходить небольшие смещения, но это не критично для пользователя. Композиции хранятся на локальном диске, поэтому вероятность задержек воспроизведения очень мала.

В случае веб-стриминга все наоборот. Экспериментальным путем выяснено, что частота синхронизации текущей позиции в треке на клиенте и сервере три раза в секунду достаточна для предотвращения «убегания» комментариев вперед или заметных задержек. Такой подход создает нагрузку на сеть, но она не существенна для локальных сетей, на работу в которых рассчитано приложение, в то время как алгоритм реализации остается простым и понятным, а пользовательский интерфейс комментариев ведет себя в соответствии с ожиданиями пользователя.



## 3 Описание программного обеспечения

### 3.1 Физическое описание системы

Файловая иерархия проекта содержит следующие компоненты:

- **server.js**

Содержит прототипы для создания демона и модели. Включает в себя два объекта-прототипа: *Model* и *Daemon* и методы:

- **exec(cmd)**  
выполняет переданную команду
- **getCommands()**  
возвращает список доступных команд
- **\_processMessage(request)**  
обработка команды, отправка результата клиенту
- **create(model)**  
инициализация демона и обработчика сообщений

- **index.js**

Файл запуска сервера проигрывателя. Здесь происходит создание сервера проигрывателя на основе прототипа, расширенного дополнительными функциями, моделью доступа к API и списком с объектами доступных команд.

- **spawnMplayer(filename)**  
запуск процесса mplayer
- **createTimePositionWatcher()**  
создание опрашивателя текущей временной позиции
- **requestTimePos()**  
запрос текущей временной позиции.

- **getComments(trackId)**

Получение списка комментариев

- **daemon.next()**

передать следующую композицию для воспроизведения в mplayer

- **daemon.play()**

начать воспроизведение

- **tui.js**

Файл запуска текстового клиента.

- **timestampToObject(timestamp)**

перевод временной метки в формат объекта времени

- **formatComment(comment)**

форматирование комментария перед выводом в интерфейс

- **showPage(page)**

показать страницу

- **fillTrackPage(track)**

заполнить страницу описания композиции

Для реализации проекта выбран язык программирования *JavaScript* на платформе *Node.js*

### 3.2 Язык программирования JavaScript

*JavaScript* [12] — прототипно-ориентированный сценарный язык программирования. Является диалектом языка *ECMAScript* [13].

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для реализации интерактивных веб-страниц, но на этом ареал его распространения не ограничивается. *JavaScript* все чаще и чаще применяется в

качестве языка систе много программирования настольных и серверных приложений, чему в наибольшей степени способствует развитие платформы *Node.js*.

### 3.3 Платформа *Node.js*

*Node.js* [14] — программная платформа, основанная на движке *V8* [16] (транслирующем *JavaScript* в машинный код) и превращающая *JavaScript* из узкоспециализированного языка в язык общего назначения. *Node.js* добавляет возможность *JavaScript* взаимодействовать с устройствами ввода/вывода через свой API (написанном на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из *JavaScript* кода. *Node.js* применяется преимущественно для реализации веб-серверов, но есть возможность разрабатывать на данной платформе и десктопные оконные приложения и даже программировать микроконтроллеры. В основе *Node.js* лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

Сервер проигрывателя реализован на платформе *Node.js*. Такой выбор был сделан по нескольким причинам.

Во-первых, это существенно упрощает обработку данных, полученных с API soundcloud, исключая из процесса обработки сереализацию, т.к. данные приходят в родном для *JavaScript* формате *JSON*. Также формат *JSON* хорош тем, что существенно снижает нагрузку на сеть, по сравнению с более объёмным *XML*.

Во-вторых, *Node.js* зарекомендовал себя как отличное средство для реализации серверного ПО.

За счет асинхронной обработки запросов время ожидания сервера на *Node.js* сводится к минимуму, а за счет широких возможностей потоковой обработки данных существенно снижаются затраты времени и памяти, так как исключается избыточное копирование данных. Парадигмы асинхронных событий и потоковой обработки данных лежат в основе платформы *Node.js*.

В большинстве реализаций *JavaScript* не существует возможностей для параллельных вычислений — все операции выполняются в одном потоке. Для реали-

зации асинхронных процедур и неблокирующих вычислений используется модель *EventLoop*. Node.js в данном случае не является исключением.

### 3.4 Принцип работы *EventLoop*

Все операции, по мере появления помещаются в очередь сообщений. Когда подходит очередь операции, она достаётся, выполняется, и виток цикла повторяется заново, со следующей командой. Поэтому все операции должны быть атомарными и неблокирующими. Если написать цикл с большим количеством итераций, то они поместятся в очередь *EventLoop* подряд. Если так сделать на веб-странице, то ни один элемент не будет доступен — когда пользователь попытается нажать на кнопку, функция-обработчик поместится в очередь *EventLoop*, занятую в данный момент выполнением команд итераций цикла. Чтобы избежать подобной ситуации необходимо каждую итерацию заносить в очередь *EventLoop* с помощью обратных асинхронных вызовов. В этом и заключается смысл понятия «неблокирующий».

### 3.5 Обратные вызовы и паттерн *Promise*

В следствии вышеописанного положения вещей, многие функции принимают в качестве параметра так называемый обратный вызов, или *callback* – специальная функция, которая выполняется по заверению операции. Так работают все асинхронные функции – либо принимают обратные вызовы, либо позволяют подписываться на события, о готовности данных или о чем угодно. При таком подходе образуется т.н. «callback hell», когда уровень вложенности и кросс-вызовов становится слишком сложным для понимания.

Для решения проблемы используется паттерн *Promises*. Функции, реализованные по этой схеме не принимают обратный вызов, а возвращают специальный объект, в котором помимо прочего содержится функция, выполняющая нужные операции. К нему при помощи метода *.then()* можно пристыковать еще функции, в порядке требуемой очередности выполнения. При вызове метода *.done()* все функции, переданные ранее, поместятся в очередь *EventLoop* асинхронно, давая возможность другим операциям занять место между ними. Программисту же важно, что код стал понятным, то есть ясно

прослеживается и контролируется последовательность операций внутри программы.

Такой подход позволяет реализовать асинхронную обработку запросов к серверу в одном потоке, что дает возможность избежать тяготы отладки многопоточных приложений.

### 3.6 Очередь сообщений ZMQ

Сетевое взаимодействие реализовано с помощью библиотеки ZeroMQ [15]. Она достаточно легковесна, имеет реализации связок для большинства популярных языков и платформ. MQ в названии расшифровывается как *message queue*, то есть библиотека реализует модель очереди сообщений. Даже если связь с сервером прервалась, сообщения не пропадут и будут доставлены как только это станет возможно. Также выбор данной библиотеки обоснован простотой использования и наличием большого сообщества пользователей.

### 3.7 Физическая архитектура

Музыкальные файлы проигрывается с помощью CLI-утилиты *mplayer*, используемой в качестве медиа-бэкэнда. При запуске сервер создает процесс *mplayer* в режиме *slave* и захватывает его потоки *stdin* и *stdout*. Так сервер управляет воспроизведением и перехватывает события времени и выхода.

Клиент реализован в текстовом интерфейсе и предназначен для запуска в эмуляторах терминала VT100 и подобных. При наличии соответствующей возможности на терминале выводятся цвета и символы UTF.

При запуске в командной строке можно указать адрес сервера. Для реализации элементов интерфейса использовалась библиотека *blessed* позволяющая легко добавлять базовые элементы интерфейса (такие как список, текстовое поле и строка) и следить за изменением размера окна терминала.

Интерфейс может работать в режиме совместимости со стандартным терминалом, т.е. в монохромном режиме, в размере 80x25 для полной совместимости с терминальными мультимплексорами, такими как *screen* и *tmux*.

### 3.8 Технические требования

Программа предназначена для запуска в операционной системе *Linux*. Для запуска программ в системе должна быть установлена платформа *Node.js* и проигрыватель *mplayer*. Для воспроизведения необходимо подключение к интернету, приемлемое для воспроизведения музыкальных файлов.

## Заключение

Таким образом, представленная система не только успешно выполняет свои задачи, но также выгодно отличается от аналогов клиент-серверной расширяемой архитектурой на основе паттерна MVC и возможностью трансляции комментариев к различным частям композиции.

На основе кода базовой серверной компоненты можно создавать различные медиа-сервера, меняя лишь модель и корректируя по необходимости функции обработки потоков данных.

Разработанная архитектура позволяет создавать композиты из серверов для интеграции плей-листов с нескольких онлайн-сервисов и локальных коллекций одновременно.

Связь между сервером и клиентами ведется посредством текстового обмена данными, что полностью соответствует принципам философии UNIX. Это позволяет создавать различные клиенты с очень простой архитектурой и очень большими возможностями для интеграции.

Работа обсуждена на конференции «Наука и молодежь-2014» и опубликована в материалах конференции. [18]

## Список использованных источников

1. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования [Текст]. / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес – СПб: Питер, 2001. – 381 с.
2. Soundcloud.com [Электронный ресурс]. // Soundcloud.com. – 2014. – Режим доступа: <http://soundcloud.com/> , свободный.
3. Cmd.fm [Электронный ресурс]. // Cmd.fm. – 2014. – Режим доступа: <http://cmd.fm/> , свободный.
4. Mopidy.com [Электронный ресурс]. // Mopidy.com. – 2014. – Режим доступа: <http://mopidy.com/> , свободный.
5. Music Player Daemon [Электронный ресурс]. // Wikipedia.org. – 2014. – Режим доступа: [http://ru.wikipedia.org/wiki/Music\\_player\\_Daemon](http://ru.wikipedia.org/wiki/Music_player_Daemon) , свободный.
6. NCurses Music Player Client (Plus Plus) [Электронный ресурс]. // NCurses Music Player Client (Plus Plus) – 2014. – Режим доступа: <http://ncmcpp.rybczak.net> , свободный.
7. Soundcloud playlist plugin patch [Электронный ресурс]. // Music Player Daemon (mpd) development list () – 2014. – Режим доступа: <http://comments.gmane.org/gmane.comp.audio.musicpd.devel/2218> , свободный.
8. Конвейер (UNIX) [Электронный ресурс]. // Wikipedia.org – 2014. – Режим доступа: [http://en.wikipedia.org/wiki/Pipeline\\_\(Unix\)](http://en.wikipedia.org/wiki/Pipeline_(Unix)) , свободный.
9. Take this GUI and shove it [Электронный ресурс]. // InfoWorld – 2014. – Режим доступа: <http://www.infoworld.com/d/networking/take-gui-and-shove-it-374> , свободный.
10. Прототипное программирование [Электронный ресурс]. // Wikipedia.org – 2014. –



Режим доступа: [http://ru.wikipedia.org/wiki/Прототипное\\_программирование](http://ru.wikipedia.org/wiki/Прототипное_программирование) , свободный.

11. Model-View-Controller [Электронный ресурс]. // Wikipedia.org – 2014. – Режим доступа: <http://ru.wikipedia.org/wiki/Model-View-Controller> , свободный.
12. JavaScript [Электронный ресурс]. // Wikipedia.org – 2014. – Режим доступа: <http://ru.wikipedia.org/wiki/JavaScript> , свободный.
13. ECMAScript [Электронный ресурс]. // Wikipedia.org – 2014. – Режим доступа: <http://ru.wikipedia.org/wiki/ECMAScript> , свободный.
14. Node.js [Электронный ресурс]. // Wikipedia.org – 2014. – Режим доступа: <http://ru.wikipedia.org/wiki/Node.js> , свободный.
15. ØMQ [Электронный ресурс]. // Wikipedia.org – 2014. – Режим доступа: <http://en.wikipedia.org/wiki/ØMQ> , свободный.
16. V8 (движок JavaScript) [Электронный ресурс]. // Wikipedia.org – 2014. – Режим доступа: [http://ru.wikipedia.org/wiki/V8\\_\(движок\\_JavaScript\)](http://ru.wikipedia.org/wiki/V8_(движок_JavaScript)) , свободный.
17. Stumblers Who Like This Page [Электронный ресурс]. // Stumbleupon.com – 2014. – Режим доступа: <http://www.stumbleupon.com/content/18isk3> , свободный.
18. Смолянинов А.Ю. КОНСОЛЬНЫЙ КЛИЕНТ К МУЗЫКАЛЬНОМУ САЙТУ // Горизонты образования. Приложение. Сборник трудов Научно-техн. конф. "Наука и молодежь - 2014". секц. "Информац. техн.", подсекц. "ПОВТ и АС", 24 апреля 2014., Барнаул: АлтГТУ, с.00 Режим доступа: <http://edu.secna.ru/publication/5/release/94/attachment/30/> , свободный.

# Приложение Б

## Руководство пользователя

Приложение предназначено для исполнения в среде текстовой консоли, которая обычно предоставляется эмулятором терминала. Для запуска необходимо знать сетевой адрес, на котором запущен сервер. Если это тот же компьютер, то достаточно выполнить команду:

```
ncs-tui localhost
```

В противном случае необходимо заменить *localhost* на адрес сервера воспроизведения.

Окно программы содержит в себе элементы отображения статуса проигрывателя и строку ввода команд. В окне программы можно увидеть (снизу вверх)

- текущую временную отметку
- длительность текущей композиции,
- её название,
- автора,
- область заголовка текущего раздела,
- область текущего раздела,
- индикатор прогресса композиции
- строку ввода команд



Рисунок Б.1 – Элементы интерфейса окна программы

Интерфейс имеет четыре логических раздела:

1. помощь
2. текущий плейлист
3. информация о композиции
4. комментарии

Переход по разделам осуществляется по нажатию соответствующим им номерным клавишам 1,2,3,4 на клавиатуре. Для удобства навигации при запуске программы область заголовка раздела отображает помощь по навигации:



Рисунок Б.2 – Запуск программы

В разделе помощи всегда можно найти краткую справку по управлению проигрывателем:



Рисунок Б.3 – Раздел помощи

Во вкладке «плейлист» отображается список воспроизведения текущих композиций, расположенных в порядке проигрывания. Здесь можно сменить проигрываемую композицию, выбрав нужную из списка и нажав клавишу Enter:



Рисунок Б.4 – Раздел списка воспроизведения

Вкладка информации о композиции содержит в себе краткий обзор свойств проигрываемой композиции:

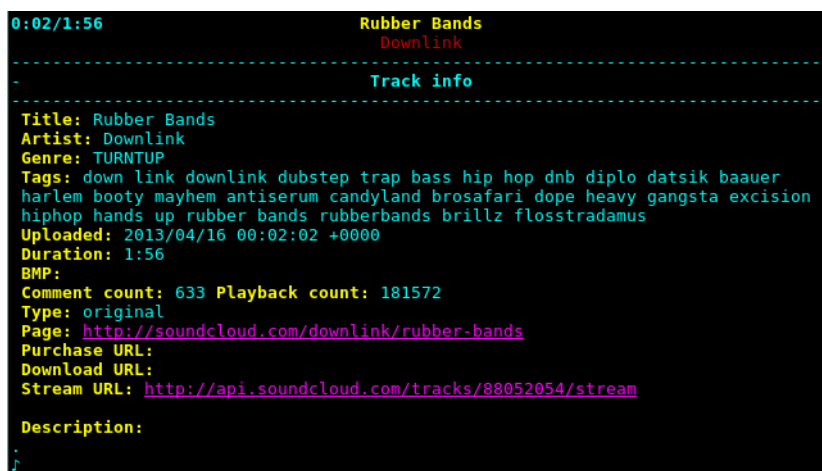


Рисунок Б.5 – Раздел информации о композиции

Раздел комментариев представляет собой список отзывов пользователей, появляющихся по мере проигрывания композиции:

```
0:18/1:26          Out Of My League Acoustic
                    chelsearongs
-----
                    Comments
-----
> Roland Concepcion: great :)
> mikkofaustino: ang galing kaht wala pa haha
> Aroll Jane: xD
> shibeeen: why isn't complete ?
> johnlester213: tite
> johnlester213: tite
> Bejay Gonzales: thanks for dedicating this one for ME! :)
> Lara Melissa Tare: Aww :""&gt; &lt;3
> Allan Pol: kumpleto sana <3
> Jaja Mandigma: For who?? =)))
> Amber Hazuki: For me? I'm flattered, really XD
> Kevin Gonda: For me?! Awww, thank you so much po!
> Jason Towne: Great music!
> Cherry Mae Collado: Aweee :">
> joan_galanta: aww. my favorite song <3 so cute!
> tehfilipinokid: Its for me!? Aww. Thanks. Just kidding. Whoever this
song is dedicated to is very lucky.
> JuliaYsabelle: I'M SOBBING IT'S BEAUTIFUL <3
> Jasmine Jeromay Laureano: For who? =)))
> Fllay.: I love you with all that I am.
> Iana Dimayuga: I raped the play button.
> Lina Plank: Beautiful voice!! <3<3<3
> Ian Ronquillo.: for me :)
> Tovy Luchavez Ü: kakainlove <3
> babyeeh: Love love love
> andhinir: A
> Myloe Beitia: PLAY PLAY PLAY PLAY PLAY AGAIN AND AGAIN ♥
> Jiyandreah: gondo gondo sana buuhin tong cover na to :'(
> Nina Mondido: thank you for granting my request :)
> Sherwin: Fucking oooooooooooooommmeeee.
> triciacabela: love you chelseaaa!! :">
> Ged Ignacio Atienza: 1 of my fav song <3
> adeleekee: this song is amazing! :)
> dafaqitsxcore: i raped the play button <3
> ♥KATE♥: idol
> Gemiimay: sweet.
> jappeeng: wow talaga!!
> Anna Faith Carandang: <3 IDOL
> Shalom Angelique Abracosa: Una palang ang ganda naaa
> brionalano: You made me feel alive again and rekindle the memories that
I've had when I was younger. :(
> Christen Diana Gallinera: My fave cover of this song!!!!
.....
♪
```

Рисунок Б.6 – Раздел комментариев

Для начала воспроизведения радио необходимо нажать сочетание клавиш Alt-x для отображения строки ввода команд. В ней необходимо выполнить команду *play* с желаемым жанром для воспроизведения.

```
Danny Brown - #ExpressYourSelf
The XX, Wu-Tang Clan, K0 - Gravel Pit Intro (Dubic Mashup) [FREE DL]
Migos ft. Chif Keef & Ballout - .9 on me
Drunk In Love (Trap Remix)
Lil Bibby - My Hood Feat. Lil Herb
...
r play rock
```

Рисунок Б.7 – Строка ввода команд

После этого проигрыватель загрузит список композиций плейлист и начнет воспроизведение. Также для выполнения доступны команды *pause* и *resume*, останавливающие и возобновляющие воспроизведение текущей композиции.

## Приложение В

### Код программного продукта

#### server.js

```
var zmq = require('zmq'),
    util = require('util'),
    responder = zmq.socket('rep');

var Model = {
  _commands: null
};

Model.getCommands = function () {
  return this._commands;
};

Model.exec = function (cmd) {

  var answer;

  if (this._commands[cmd.name] === undefined) {
    console.log(this._commands);
    console.error("No such command: ", cmd.name);
  } else {
    answer = this._commands[cmd.name].exec(cmd.params);
  }

  return answer;
};

var Daemon = {

  _model: null,
  _responder: zmq.socket('rep'),
  _publisher: zmq.socket('pub')
```

```

};

Daemon.__proto__ = Model;

Daemon.getCommands = function () {
    return this._commands;
};

Daemon.exec = function (cmd) {

    var answer;

    if (this._commands[cmd.name] !== undefined) {
        answer = this._commands[cmd.name].exec(cmd.params);
    }

    return answer;
};

Daemon._processMessage = function (request) {

    var cmd, answer = {};

    try {
        cmd = JSON.parse(request.toString());
    } catch (e) {

        this._responder.send('{ "error" : "bad command format"}');
        return;
    }

    console.log(cmd);
    answer = this.exec(cmd);

    if (answer === undefined) {

        answer = this.getCommands();

        this._responder.send(

```



```

        '{ "error" : "no such command", "params": ' +
        '{ "commands" : ' + JSON.stringify(answer) + ' } }')';
    } else {

        this._responder.send('0');

        var response;

        answer === 0 ?
            response = 0 :
            response = {
                'command': cmd.name,
                'data': answer
            };

        this._publisher.send('EVENT ' + JSON.stringify(response));
    }
};

```

```

Daemon.create = function (model) {

    var daemon = Object.create(this);

    daemon._model = model;
    daemon._responder.bind('tcp://*:5555', function(err) {
        if (err) {
            console.log(err);
        } else {
            console.log("Listening on 5555...");
        }
    });

    daemon._publisher.bind('tcp://*:5556', function(err) {
        if (err) {
            console.log(err);
        } else {
            console.log("Publishing on 5556...");
        }
    });
};

```

```

    }
  });

  daemon._responder.on('message', function (req) {
    daemon._processMessage(req);
  });

};

module.exports = {

  daemon: Daemon,
  model: Model
};

```

## index.js

```

var daemon = require('./server').daemon,
    model = {},
    EventEmitter = new (require('events').EventEmitter)(),
    rest = require('rest'),
    querystring = require('querystring');

console.log( EventEmitter );

var mplayer,
    timePosTimeout,
    clientID = 'f5dcaf5f7c97d2996bb30bb40d23ee57',
    isPlaying = false;

function spawnMplayer(filename) {

  if (mplayer !== undefined && mplayer.stdin.writable === true) {

    clearTimeout(timePosTimeout);

    mplayer.removeAllListeners('exit');
    mplayer.stdin.write('quit\n');
  }
}

```

```

    mplayer = require('child_process').
        spawn('mplayer', ['-slave', '-quiet',
                           '-cache', '100',
                           '-ao', 'alsa',
                           '-cache-min', '10',
                           filename]);

    createTimePositionWatcher();
}

function createTimePositionWatcher() {

    var timeRegex = /^ANS_TIME_POSITION=(\d+)\.(\d+)/;

    mplayer.stdin.on('error', function(e) {
        console.log(e);
    });

    mplayer.stdout.on('data', function(chunk) {

        var matches = chunk.toString().trim().match(timeRegex);

        if (matches) {

            var timestamp = matches[1] * 1000 + matches[2] * 100;
            EventEmitter.emit('mplayer_time_pos', timestamp);
        }
    });

    requestTimePos();
}

EventEmitter.on('mplayer_time_pos', function(timePos) {
    daemon._publisher.send('TIMEPOS ' + timePos);
});

EventEmitter.on('begin_play', function() {
    daemon._processMessage(JSON.stringify({name: 'getCurrentTrack', params: {}}));
});

```

```

function requestTimePos() {

    mplayer.stdin.write('get_time_pos\n');
    timePosTimeout = setTimeout(requestTimePos, 100);
}

model.__proto__ = require('./server').model;
model._commands = {

    list: {
        name: "list",
        params: {},
        exec: function(params) {
            return fs.readdirSync('sounds');
        }
    },
    getTaggedTracks: {
        name: "getTaggedTracks",
        params: { "tag": "tag to search"},
        exec: function(params) {

            var path = 'http://api.soundcloud.com/tracks.json?',
                apiParams = querystring.stringify({
                    consumer_key: clientID,
                    tags: params.tag,
                    filter: 'streamable',
                    //order: 'created_at'
                    order: 'hotness'
                });

            return rest(path + apiParams)
                .then(function (request) { return request.entity; })
                .then(JSON.parse);
        }
    }
};

var d = daemon.create(model);

```

```

console.log('daemon creted.');
```

```

model._playlist = null;
model._currentTrackIndex = 0;
```

```

model.getComments = function (trackId) {

    var path = 'http://api.soundcloud.com/tracks/' + trackId + '/comments.json?',
        apiParams = querystring.stringify({ consumer_key: clientID });

    return rest(path + apiParams)
        .then(function (request) { return request.entity; })
        .then(JSON.parse);

};
```

```

daemon.next = function next() {

    var playlistLength = model._playlist.length || 0;

    if (playlistLength == 0) return 0;

    model._currentTrackIndex++;

    if (model._currentTrackIndex >= playlistLength) {
        model._currentTrackIndex = 0;
    }

    daemon.play();

    return 0;

};
```

```

daemon.play = function play() {

    var currentTrack = model._playlist[model._currentTrackIndex];

    console.log('Playing: ' + currentTrack.title + ' [' + currentTrack.duration + ']');
    console.log('URL: ' + currentTrack.stream_url);

```

```

var commentsPromise = model.getComments(currentTrack.id);

commentsPromise.done(function(comments) {

    var groupedByTime = {};

    /* Group comments by cutted timestamp:
    *
    *
    * e.g. if
    * comment.timestamp == 115798
    * then -->
    * groupedByTime['1157'].push( 'comment' )
    *
    */

    comments.forEach(function(comment) {

        var stringTime = comment.timestamp ?
            comment.timestamp.toString() : '0',
            timeIndex = stringTime.substr(0, stringTime.length - 2);

        if (groupedByTime[timeIndex] === undefined) {
            groupedByTime[timeIndex] = [];
        }

        groupedByTime[timeIndex].push(comment);

    });

    currentTrack.comments = groupedByTime;
    EventEmitter.emit('begin_play');
});

spawnMplayer(currentTrack.stream_url + "?consumer_key=" + clientID);

mplayer.on('exit', function () {

    isPlaying = false;

```

```

        clearTimeout(timePosTimeout);
        daemon.next();
    });

    isPlaying= true;

};

daemon._commands = {

    pause: {
        name: "pause",

        exec: function(params) {

            if (isPlaying) {
                mplayer.stdin.write('pause\n');
                isPlaying = false;
            }

            return 0;
        }
    },
    resume: {
        name: "resume",

        exec: function(params) {

            if (!isPlaying) {
                mplayer.stdin.write('pause\n');
                isPlaying = true;
            }

            return 0;
        }
    },
    toggle: {
        name: "toggle",

```

```

    exec: function(params) {

        mplayer.stdin.write('pause\n');
        isPlaying = !isPlaying;

        return 0;
    }

},
n: {
    name: "n",

    exec: daemon.next
},
playIndex: {
    name: "playIndex",
    params: { index: "index of playlist element"},

    exec: function (params) {

        var index = params.index * 1 || 0;

        model._currentTrackIndex = index;
        daemon.play();

        return 0;

    }
},
p: {
    name: "p",
    params: {
        tag: "tag to play"
    },

    exec: function p(params) {

        function createPlaylist (allTracks) {

            model._playlist = allTracks.filter(function hasStreamUrl(track) {

```



```

        return track.stream_url !== undefined;
    }).sort(function (a,b) {
        return a.duration*1 - b.duration*1;
    });

    model._currentTrackIndex = 0;

    daemon._processMessage(JSON.stringify({name: 'getPlaylist', params: {}}));

    return allTracks;
}

var allTracks = model.exec({

    name: "getTaggedTracks",
    params: {
        tag: params.tag
    }
});

allTracks.then(createPlaylist);
allTracks.then(daemon.play);
allTracks.done();

return 0;
}
},
getCurrentTrack: {
    name: "getCurrentTrack",
    exec: function() {

        var currentTrack;

        try {
            currentTrack = model._playlist[model._currentTrackIndex];
        } catch(e) {
            return 0;
        }
    }
}

```

```

        return currentTrack === undefined ? 0 : currentTrack;
    }

    },
    getPlaylist: {
        name: "getPlaylist",
        exec: function() {

            return model._playlist || 0;
        }

    }

};

```

## tui.js

```

var format = require('util').format,
    zmq = require('zmq'),
    rq = zmq.socket('req'),
    timeSub = zmq.socket('sub');
    trackSub = zmq.socket('sub');

rq.connect(format('tcp://%s:5555', process.argv[2]));
timeSub.connect(format('tcp://%s:5556', process.argv[2]));
timeSub.subscribe('TIMEPOS');

trackSub.connect(format('tcp://%s:5556', process.argv[2]));
trackSub.subscribe('EVENT');

var currentTrack, playlist;

rq.on('message', function(reply){

});

trackSub.on("message", function(reply) {

```

```

var replyMatches = /^[A-Z]+ ({.*})/.exec(reply.toString());

if (!replyMatches || !replyMatches[1]) return;

var jsonReply = JSON.parse(replyMatches[1] || '{}');

switch(jsonReply.command) {

case 'getCurrentTrack': {

    currentTrack = jsonReply.data;

    if (!currentTrack.comments) {
        currentTrack.comments = [];
    }

    titleBox.setContent( currentTrack.title );
    artistBox.setContent( currentTrack.user.username );
    fillTrackPage(currentTrack);

    commentBox.setContent('');

    if (currentTrack.comments[''] !== undefined) {

        currentTrack.comments[''].forEach(function(comment){
            commentBox.setContent(commentBox.content + '\n' +
                formatComment(comment));

        });
        commentBox.setScrollPerc(100);

    }

    screen.render();

}; break;
case 'getPlaylist': {

```

```

var tracks = [];

playlist = jsonReply.data;

playlist.forEach(function (track) {
    tracks.push('{yellow-fg} ' + track.title + ' {/yellow-fg}');
});

playlistBox.setItems(tracks);
screen.render();

};break;

}

});

function timestampToTimeObject(timestamp) {

    var stringTimestamp = timestamp + '',
        timestampAsSeconds =
            stringTimestamp.substr(0, stringTimestamp.length - 3),
        minutes = timestampAsSeconds / 60 | 0,
        seconds = timestampAsSeconds % 60;

    return {
        m: minutes,
        s: seconds > 9 ? seconds : '0' + seconds
    };
}

function formatComment(comment) {

    var commentString =

        '{yellow-fg}' +
        ' _ ' +
        comment.user.username +
        ': {/yellow-fg}' +

```

```

        comment.body;

    return commentString;
}

timeSub.on('message', function(timeString) {

    if (currentTrack === undefined) return;

    var timestampString = timeString.toString().split(' ')[1],
        timeIndex = timestampString.substr(0, timestampString.length - 2),
        timeObject = timestampToTimeObject(timestampString),
        durationObject = timestampToTimeObject(currentTrack.duration);

    timeBox.setContent(timeObject.m + ':' + timeObject.s +
                        '/' +
                        durationObject.m + ':' + durationObject.s);

    var progressPerc = (timestampString | 0) / (currentTrack.duration / 100);
    trackProgressBar.setProgress(progressPerc);

    if ( timeIndex && currentTrack.comments[timeIndex] !== undefined) {
        currentTrack.comments[timeIndex].forEach(function(comment){

            commentBox.setContent(commentBox.content + '\n' +
                                   formatComment(comment));

        });

        commentBox.setScrollPerc(100);
    }

    screen.render();

});

var blessed = require('blessed');

// Create a screen object.
var screen = blessed.screen({
    fastCSR: true,

```

```

        useBCE: true
    });

    var form = blessed.form({
        parent: screen,
        keys: true,
        width: '100%',
        height: '100%'

    });

    var pages = [];
    form.on('resize', function (data) {

        pages.forEach(function (page) {
            page.height = form.height - 7;
            page.setScrollPerc(100);
        });

        screen.render();
    });

    var promptBox = blessed.box({
        parent: form,
        shrink: true,
        keys: true,

        bottom: 0,
        left: 0,
        content: 'J',
        width: 1,
        height: 1,

        name: 'prompt'
    });
    var commandTextbox = blessed.textbox({
        parent: form,
        shrink: true,
        keys: true,

```

```

    left: 2,
    bottom: 0,

    width: '100%',
    height: 1,

    name: 'command'
});

var titleBox = blessed.box({
  parent: form,
  top: 0,
  align: 'center',
  height: 1,
  width: "100%",

  style: {
    fg: 'yellow',
    bold: true
  }

});

var artistBox = blessed.box({
  parent: form,
  top: 1,
  width: "100%",
  height: 1,
  align: 'center',

  content: 'Artist here.',
  style: {
    fg: 'red'
  }

});

var timeBox = blessed.box({
  parent: form,
  top: 0,

```

```

    height: 1,
    left: 0,
    shrink: true,

    style: {
        bold: true
    }

});
var pageTitleBox = blessed.box({
    parent: form,

    width: '100%',
    bold: true,

    align: 'center',
    content: '1: Help \t 2: Playlist \t 3: Track Info \t 4: Comments',
    border: {
        ch: '-'
    },
    top: 2,
    height: 3

});
var pageOptions = {
    parent: form,
    tags: true,
    scrollable: true,

    width: '100%',
    content: "page",
    padding: {
        left: 1,
        right: 1
    },
    top: 5,
    height: form.height - 7
};

var playlistOptions = {

```



```

    parent: form,
    tags: true,
    scrollable: true,

    width: '100%',
    padding: {
        left: 1,
        right: 1
    },

    top: 5,
    height: form.height - 7,

    selectedBg: 'yellow',
    selectedFg: 'black',
    selectedBold: true

};
var commentBox = blessed.box(pageOptions),
    trackBox = blessed.box(pageOptions),
    playlistBox = blessed.list(playlistOptions),
    helpBox = blessed.box(pageOptions);

commentBox.title = 'Comments';
helpBox.title = 'Help';
trackBox.title = 'Track info';
playlistBox.title = 'Playlist';

helpBox.setContent("this is help page.");
trackBox.setContent("this is track page.");
playlistBox.focus();

pages.push(helpBox, playlistBox, trackBox, commentBox);

function fillTrackPage(track) {

    var durationObject = timestampToTimeObject(track.duration),
        trackInfo =
            '{yellow-fg}{bold}Title:{/bold}{/yellow-fg} '

```

```

+ track.title + '\n' +
'{yellow-fg}{bold}Artist:{/bold}{/yellow-fg} '
+ track.user.username + '\n' +
'{yellow-fg}{bold}Genre:{/bold}{/yellow-fg} ' +
+ track.genre + '\n' +
'{yellow-fg}{bold}Tags:{/bold}{/yellow-fg} ' +
+ track.tag_list + '\n' +
'{yellow-fg}{bold}Uploaded:{/bold}{/yellow-fg} ' +
+ track.created_at + '\n' +
'{yellow-fg}{bold}Duration:{/bold}{/yellow-fg} ' +
+ durationObject.m + ':' + durationObject.s + '\n' +
'{yellow-fg}{bold}BMP:{/bold}{/yellow-fg} ' +
+ (track.bmp || '') + '\n' +
'{yellow-fg}{bold}Comment count:{/bold}{/yellow-fg} ' +
+ track.comment_count +
'{yellow-fg}{bold} Playback count:{/bold}{/yellow-fg} ' +
+ track.playback_count + '\n' +
'{yellow-fg}{bold}Type:{/bold}{/yellow-fg} ' +
+ (track.track_type || '') + '\n' +
'{yellow-fg}{bold}Page:{/bold}{/yellow-fg} ' +
+ track.permalink_url + '\n' +
'{yellow-fg}{bold}Purchase URL:{/bold}{/yellow-fg} ' +
+ (track.purchase_url || '') + '\n' +
'{yellow-fg}{bold}Download URL:{/bold}{/yellow-fg} ' +
+ (track.download_url || '') + '\n' +
'{yellow-fg}{bold}Stream URL:{/bold}{/yellow-fg} ' +
+ track.stream_url + '\n' + '\n' +
'{yellow-fg}{bold}Description:{/bold}{/yellow-fg} ' + '\n'
+ track.description + '\n' ;

```

```

trackBox.setContent(trackInfo);

```

```

}

```

```

function fillHelpPage() {

```

```

    helpBox.setContent(
        '\t M-p :{yellow-fg} Play / pause {/yellow-fg}\n' +
        '\t M-x :{yellow-fg} Call command prompt {/yellow-fg}\n' +
        '\n' +
        '\t {bold}{yellow-fg}Available commands {/yellow-fg}{/bold} \n' +

```

```

        '\n' +
        '\t p tag:<tag> :{yellow-fg} ' +
        'Play radio with genre <tag> {/yellow-fg}\n' +
        '\t pause :{yellow-fg} Pause playback {/yellow-fg}\n' +
        '\t resume :{yellow-fg} Resume playback {/yellow-fg}\n' +
        '\t n :{yellow-fg} Next track in playlist {/yellow-fg}\n'

    );
    screen.render();
}

```

```

var trackProgressBar = blessed.progressbar({
  parent: form,
  tags: true,

  barBg: 'black',

  fg: 'black',
  padding: {
    left: 1,
    right: 1
  },
  bottom: 1,
  height: 1,
  filled: 0,
  ch: '.'
});

trackProgressBar.setProgress(100);

screen.key(['escape', 'q', 'C-c'], function(ch, key) {
  return process.exit(0);
});
screen.key(['M-x'], function(ch, key) {
  commandTextbox.readInput();
});
screen.key(['M-x'], function(ch, key) {
  commandTextbox.readInput();
});
screen.key(['M-p'], function(ch, key) {

```

```

    var toggleRequest = JSON.stringify({ name: 'toggle', params: {} });
    rq.send(toggleRequest);

});

function showPage(pageToShow) {

    pages.forEach(function (page) {
        page.hidden = true;
    });

    pageToShow.hidden = false;
    pageToShow.focus();
    pageTitleBox.setContent(pageToShow.title);
    screen.render();
}

screen.key(['1'], function () { showPage(helpBox); });
screen.key(['2'], function () { showPage(playlistBox); });
screen.key(['3'], function () { showPage(trackBox); });
screen.key(['4'], function () { showPage(commentBox); });

playlistBox.key(['up'], function () {

    playlistBox.up();
    playlistBox.focus();
    screen.render();
});

playlistBox.key(['down'], function () {

    playlistBox.down();
    playlistBox.focus();
    screen.render();
});

playlistBox.key(['enter'], function () {

    var playFromList =
        JSON.stringify({

```

```

        name: 'playIndex',
        params: {
            index: playlistBox.selected
        }
    });

    rq.send(playFromList);
    playlistBox.focus();

    screen.render();
});

commandTextbox.key('enter', function() {

    var inputArray = commandTextbox.value.split(' '),
        command = inputArray[0], params = {};

    if (inputArray[1] !== undefined) {

        var param = inputArray[1].split(':');
        params[param[0]] = param[1];
    }

    var request = JSON.stringify({ name: command, params: params });

    rq.send(request);
    commandTextbox.clearValue();

    screen.render();
});

var getCurrentTrackRequest =
    JSON.stringify({
        name: 'getCurrentTrack',
        params: {}
    }),
getPlaylist =
    JSON.stringify({
        name: 'getPlaylist',

```

```
        params: {}  
    });  
  
    rq.send(getCurrentTrackRequest);  
    rq.send(getPlaylist);  
  
    fillHelpPage();  
    showPage(helpBox);  
  
    screen.render();
```