Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных систем и сетей
Кафедра Информатики

К защите допустить:

Заведующая кафедрой информатики

_____ Н. А. Волорова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему:

СЕРВИС, ПРЕДОСТАВЛЯЮЩИЙ ВОЗМОЖНОСТЬ ПОЛУЧЕНИЯ НОВИНОК МУЗЫКАЛЬНОЙ ИНДУСТРИИ

БГУИР ДП 1-40 04 01 00 045 ПЗ

Студент	Т. С. Лавник
Руководитель	В. В. Шиманский
Консультанты:	
от кафедры информатики	В. В. Шиманский
по экономической части	К. Р. Литвинович
Нормоконтролёр	Н. Н. Бабенко

Рецензент

Минск 2017

РЕФЕРАТ

Ключевые слова: музыкальная публикация; артист; тип публикации; поиск артистов; отложенные задачи.

Дипломный проект выполнен на 6 листах формата А1 с пояснительной запиской на 35 страницах, без приложений справочного или информационного характера. Пояснительная записка включает 4 глав, 2 рисунков, 4 таблиц, 44 формул и 3 литературный источник.

Целью дипломного проекта является создание инструмента, позволяющего решать задачи поиска новых музыкальных публикаций удобным способом.

Для достижения цели дипломного проекта было разработано веб-приложение на фреймворке Ruby on Rails. Веб-приложение может быть использовано для получения новых публикаций интересующих пользователя артистов. В приложении используются алгоритмы, позволяющие периодически проверять различные сервисы на наличие обновлений, касающихся интересующих релизов артистов. Также в приложении присутствует email-рассылка, отправляемая пользователям в случае наличия обновлений.

В разделе технико-экономического обоснования был произведён расчёт затрат на создание ПО, а также экономии средств при приобретении программного продукта, получаемой пользователем. Проведённые расчёты показали экономическую целесообразность проекта.

АННОТАЦИЯ

на дипломный проект «Сервис, предоставляющий возможность получения новинок музыкальной индустрии» студента УО «Белорусский государственный университет информатики и радиоэлектроники» Лавник Т. С.

Ключевые слова: музыкальная публикация; артист; тип публикации; поиск артистов; отложенные задачи.

Дипломный проект выполнен на 6 листах формата А1 с пояснительной запиской на 35 страницах, без приложений справочного или информационного характера. Пояснительная записка включает 4 глав, 2 рисунков, 4 таблиц, 44 формул и 3 литературный источник.

Целью дипломного проекта является создание инструмента, позволяющего решать задачи поиска новых музыкальных публикаций удобным способом.

Для достижения цели дипломного проекта было разработано веб-приложение на фреймворке Ruby on Rails. Веб-приложение может быть использовано для получения новых публикаций интересующих пользователя артистов. В приложении используются алгоритмы, позволяющие периодически проверять различные сервисы на наличие обновлений, касающихся интересующих релизов артистов. Также в приложении присутствует email-рассылка, отправляемая пользователям в случае наличия обновлений.

Во введении производится ознакомление с проблемой, решаемой в дипломном проекте.

В первой главе производится обзор предметной области проблемы решаемой в данном дипломном проекте. Приводятся необходимые теоретические сведения, а также производится обзор существующих разработок.

Во второй главе производится краткий обзор технологий, использованных для реализации ПО в рамках дипломного проекта.

В третьей главе производится обзор реализованного ПО. Описываются его составные части и особенности. Приводятся результаты практических испытаний и производится сравнение с существующим ПО.

В четвертой главе производится технико-экономическое обоснование разработки.

В заключении подводятся итоги и делаются выводы по дипломному проекту, а также описывается дальнейший план развития проекта.

СОДЕРЖАНИЕ

Введение 6
1 Обзор предметной области 7
1.1 Постановка задачи 7
1.2 Обзор существующих аналогов 8
1.3 Музыкальный альбом (публикация) 11
1.4 API 12
2 Обзор используемых технологий 14
2.1 Ruby on Rails 14
2.2 AngularJS 16
3 Проектирование и реализация 17
3.1 Трёхуровневая архитекутра приложения 17
4 Технико-экономическое обоснование эффективности сервиса, предоставляющего возможность получения новинок музыкальной индустрии 21
4.1 Введение и исходные данные 21
4.2 Расчёт сметы затрат и цены программного продукта 21
4.3 Расчёт экономической эффективности у разработчика 30
Заключение 34
Список использованных источников 35

ВВЕДЕНИЕ

Музыка сопровождала человечество на протяжении всей истории. Вначале люди использовали только свой голос, потом появились музыкальные инструменты. До сих пор прогресс не стоит на месте. С появлением компьютерной техники и интернета появилось ещё больше возможностей для самовыражения: программное обеспечение для звукозаписи, веб-сайты для публикации своих произведений, сервисы для анализа предпочтений и тому подобное. Учитывая существование изобилия музыкальных ресурсов, которые предоставлют пользователям разнообразные возможности, становится трудно следить за новыми публикациями любимых исполнителей, и, хотя подобные сервисы уже присутствуют на мировом рынке, не все они являются удобными и логически законченными.

В связи с вышесказанным, целью данного диплома является создание сервиса, который будет предоставлять возможность удобного получения уведомлений о новинках музыкальной индустрии. Сервис должен быть оснащён понятным и дружелюбным интерфейсом, позволять пользователю удобно регистрироваться и наполнять данными свой профиль, указывая свои предпочтения в виде списка интересующих артистов.

Для того, чтобы сервис был полезен пользователю, он должен быть удобен, привлекателен и реализовывать некоторую потребность. В этом состоит основа данного дипломного проекта: собирать информацию из разных сервисов в одном месте, тем самым упрощая этот для конечного пользователя. В некотором роде - это агрегат, который собирает интересы и возвращает ответ в виде удовлетворения этих интересов путём указания ссылок на публикации любимых артистов. Удобство должно быть выражено в дизайне интерфейса, который должен быть интуитивно ясен пользователю и помогать ему производить некоторые операции внутри сервиса. Также проект должен быть хорошо масштабируемым, чтобы можно было легко добавлять разные источники публикаций в свой арсенал.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Постановка задачи

Целью данного дипломного проекта является создание веб-сервиса, состоящего из серверной и клиентской частей с использованиям Ruby on Rails, AngularJS, Sidekiq, для получения уведомлений о новинках в музыкальной индустрии.

Для реализации данного проекта можно выделить следующие минимальные требования:

– Система должна быть компонентной, то есть разделённой на отдельные модули. Это облегчает разработку, внесение изменений и тестирование.

– Система должна поддерживать мультиязычность для охвата большей аудитории с возможностью добавления новых языков и редактирования старых.

– Система должна соответствовать современным требованиям к интерфейсам быть удобной для пользователя.

– Система должна быть выполнена в виде SPA-приложения с использованием асинхронных запросов.

– Система должна быть легко масштабируема для добавления новых источников публикаций.

– Система должна быть имплементирована с наличием авторизации, аутентификации, разных ролей, в том числе администраторской.

Для реализации вышеописанных требований необходимо решить следующие задачи:

– Изучить аналоги, присутствующие на рынке, для того, чтобы учесть ошибки и недостатки изучаемых сервисов, а также выделить сильные стороны и использовать их в разработке.

– Изучить предметную область, портрет целевой аудитории, чтобы лучше понимать нужды и правильно реализовать их в системе.

– Выбрать и изучить подходящие технологии для разработки сервиса.

– Создать архитектуру, которая будет удовлетворять требованиям разработки и реализовать её с использованием выбранных технологий.

1.2 Обзор существующих аналогов

В данном разделе будут рассмотрены приложения, которые по логике своей работы пересекаются с тематикой данной дипломной работы, а именно предоставляют пользователям возможность узнавать о новинках музыкальной индустрии. Некоторые из них является веб-сервисами, направленными на видеоконтент, другие - на музыкальный. И те, и другие выполняют свою задачу и направлены на свою целевую аудиторию. Рассмотрим несколько из вышеописанных сервис.

1.2.1 YouTube - Just Released Music Videos

YouTube - видеохостинговый сервис, предоставляющий пользователям загружать свои видео, делать их некоторую обработку и получать статистику просмотров. Также на YouTube есть комментарии, кнопки “мне нравится” и “мне не нравится”. Таким образом данный сервис имеет большую вовлечённость пользователей.

Однако в данной дипломной работе рассматривается отдельный сервис, который пересекается с её тематикой - это Just Released Music Videos. Эта площадка показывает людям последние публикации артистов. Другими словами, Just Released Music Videos демонстрирует пользователям список из только что издававшихся видео-роликов певцов, музыкантов, музыкальных коллективов. Данный сервис интегрирован в YouTube и является его частью.

Список видеороликов, представленный в Just Released Music Videos отсортирован в порядке убывания по дате публикации, то есть, чем новее видео, тем выше в этом списке оно поднимается, что логично для пользователя, который ожидает увидеть именно новинки музыкальной индустрии. Также напротив видео будет указана длительность композиции и небольшая картинка, которая является превью к видео.

У данного сервиса можно выделить следующие преимущества:

– позволяет пользователю не только посмотреть музыку, но и увидеть видеоклип;

– позволяет пользователю видеть количество просмотров, таким образом оценивая его популярность;

– есть возможность включения субтитров, что является встроенной функциональностью YouTube, таким образом позволяя пользователю сразу видеть текст композиции при условии его присутствия;

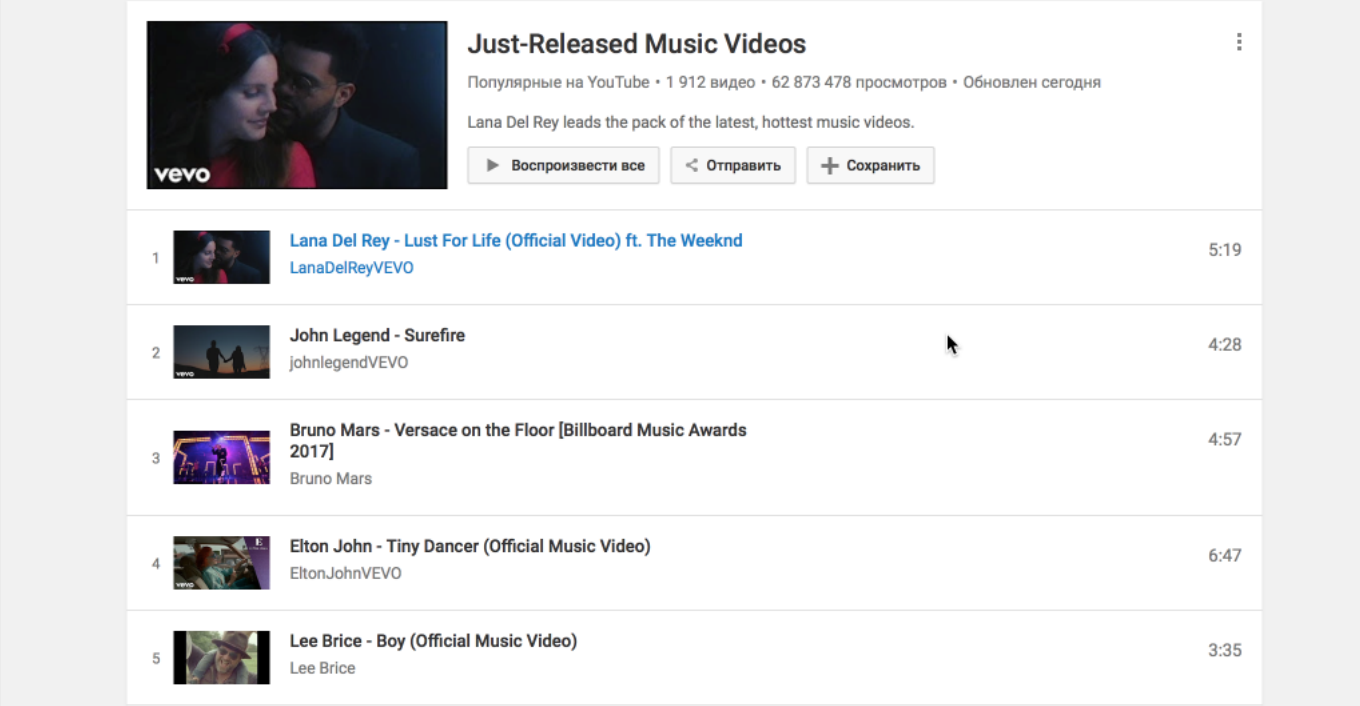

Рисунок 1.1 – Изображение сервиса YouTube - Just Released Music Videos

Однако кроме преимуществ можно выделить следующие недостатки:

– артисты, представленные в списке данного сервиса, никак не связаны с тем, что интересно пользователю, тем самым создавая лишнюю информационные данные;

– сервис не является самостоятельным, а интегрированным в YouTube;

– нет возможности получения прямых уведомлений на почту или в нотификациях переносимого гаджета;

1.2.2 AllMusicAllMusic представляет собой американский сервис, который включает в себя крупную онлайновую музыкальную базу данных. В AllMusic представлена информация о жанрах музыки, музыкантах и коллективах, в том числе новые публикации и профессиональные рецензии. Данный сервис является обладателем огромного музыкального архива, который насчитывает около шести миллионов композиций. Также на сайте имеется возможность оценки публикаций пользователями. AllMusic предоставляет пользователю довольно широкий спектр возможностей. Например, пользователь может подписаться на обновления, чтобы получать письма с уведомлением о новой публикации. Также есть возможность получать индивидуальные музыкальные рекомендации. Для этого необходимо заполнить артистов, которые нравятся пользователю, в поля для ввода и далее будут предложены альбомы артистов по каким-либо параметрам схо-

жих с введёнными ранее. Я считаю, что это весьма полезная функциональность для того, чтобы найти новую музыку. AllMusic обладает стильным, выдержанным, но, на мой взгляд, слегка загроможденным дизайном. Сервис предоставляет широкую функциональность, и по этой причине пользователю, заинтересованному лишь в получении музыкальных новинок, может мешать наличие множества разнообразных разделов. Сервис перестаёт быть сконцентрированным на одной вещи. Это, в зависимости от интересов конечного пользователя, может как оттолкнуть, так и быть интересным.

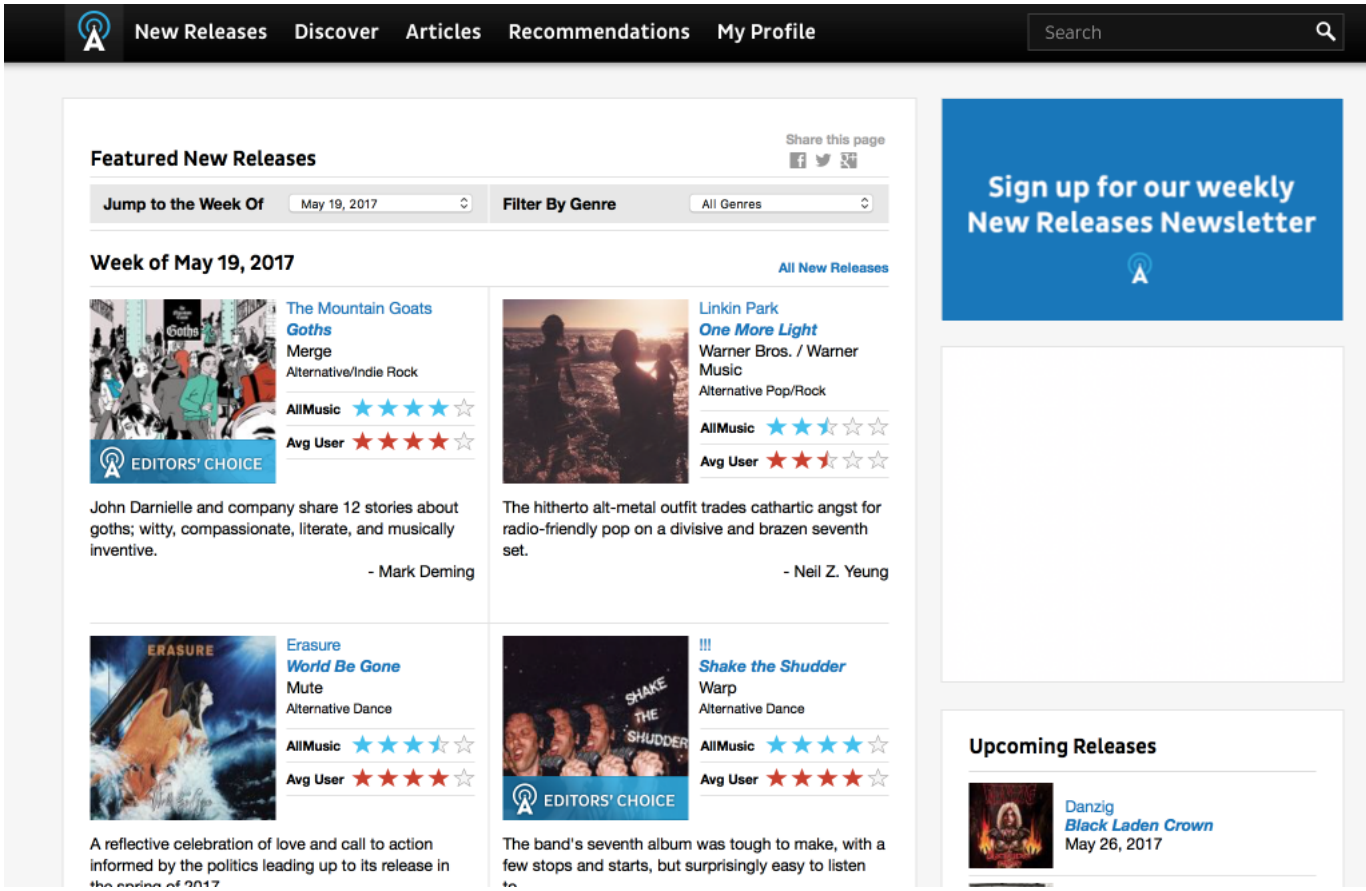

Рисунок 1.2 – Изображение сервиса AllMusic

Преимущества AllMusic:

– возможность регистрации через социальные сети;

– большая база данных публикаций;

– возможность оценивать альбомы, что позволяет пользователям судить о качестве;

– богатство жанров;

– возможность получать уведомления;

– наличие персональных рекомендаций;

– возможность прослушать отрывки композиций;

Также можно обозначить следующие недостатки:

– отсутствие концентрированности на публикациях артистов, и, как следствие, массивный интерфейс;

– отсутствие ссылок на сервисы для приобретения альбомов;

– отсутствие возможности импортировать артистов с устройств/сервисов;

– небольшая интерактивность;

1.2.3 ВыводУчитывая результаты ознакомления с уже присутствующими аналогами сервисов для получения информации о новинках музыкальной индустрии, можно сделать вывод, что большинство подобных сервисов либо слишком усложнены, либо делают акцент не на основную потребность конечного пользователя, а именно получение уведомлений, информации о новых публикациях любимых артистов.

1.3 Музыкальный альбом (публикация)

Музыкальный альбом - это некоторая совокупность музыкальных композиций, которые выпускаются вместе, в стандартном формате. Такие композиции должны быть доступны для произведения на популярных воспроизводящих устройствах. Существуют классификации альбомов по нескольким признакам: по объёмы, типу записи и так далее. Некоторые из них будут рассмотрены далее.

1.3.1 Классификация по объёму

– Сингл не относят к альбомам, так как состоят из одной-двух песен. Рассматривается в данной работе для большей ясности нижеприведённой информации.

– Стандартный альбом, или LP (от английского “Long Play”) обычно размещается на одной пластинке или же на одном компакт-диске. Время звучания насчитывает около 30-80 минут.

– Мини-альбом, или EP (от английского “Extended Play”) по своему размеру является промежуточным вариантом между стандартным и синглом.

– Двойной альбом выпускается на двух носителях информации, и время звучания такого альбома в среднем в два раза больше, чем стандартный альбом.

1.3.2 Классификация по типам записей и композиций

– Студийный альбом - альбом, записанный в звукозаписывающей студии.

– Концертный альбом - альбом, который записывается в то время, как идёт “живое” выступление перед публикой, то есть запись концерта.

– “Программный” альбом - альбом, в котором обычно содержаться только ранее не издававшиеся композиции, хотя бывают и исключения, когда в “программный” альбом включаются концертные записи.

1.3.3 Другие типы альбомов

– Демо представляет собой демонстрационную запись. В ней обычно содержится “сырой” материал, который используется для ознакомления других лиц с творчеством исполнителя. Зачастую это некоторые первые записи музыкантов, которые официально не издавались.

– Концептуальный альбом - это альбом, все композиции которого объединены некоторой идеей (концепцией). В таких альбомах можно усмотреть некоторый замысел в составе и порядке композиций.

– Промо - альбом, который выпускается небольшим тиражом специально для бесплатного распространения с целью рекламы исполнителя.

– Саундтрек - альбом, в составе которого используются композиции из одного или нескольких фильмов, компьютерных игр.

– Трибьют - альбом, в котором в дань уважения и в знак почитания творчества одни исполнители переигрывают песни другого определённого исполнителя. Такие композиции называются каверами.

1.4 API

API(application programming interface, программный интерфейс приложения) - это совокупность процедур, функций, и классов и т.д. , которую предоставляет сервис для использования в некоторых внешних приложениях (клиентах).

API используется для того, чтобы представить функциональность программы, при этом абстрагируясь от конкретной реализации.

WebAPI - это API, основу в котором составляют HTTP-запросы и HTTP-ответы, сконструированные по определённым принципам. API используется, как источник данных для клиентского приложения. Клиент использует HTTP-запросы с глаголами GET, POST, PATCH, DELETE и другие. При этом в теле запроса передаются необходимые параметры. Далее API принимает этот запрос, и, если необходимо, авторизирует клиент. Может использоваться ролевая авторизация, когда в системе присутству-

ют разные типы пользователей: администраторы, модераторы, конечные пользователи. После запрос обрабатывается, и возвращается ответ в виде XML или JSON. Ответ принимается клиентом, также обрабатывается и на основе результатов производятся необходимые действия.

2 ОБЗОР ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ

2.1 Ruby on Rails

Ruby on Rails - веб-фреймоворк, написанный на языке Ruby. Ruby on Rails известен тем, что позволяет очень быстро начать разрабатывать приложения. В данном веб-фреймворке есть возможность под названием “scaffold”, которая позволяет одной командой создавать представления, контроллер и модель для нужного объекта.

Философия Ruby on Rails складывается из двух основных пунктов:

– “Convention over configuration” - что переводится, как преобладание соглашений над конфигурацией. Это означает, что в экосистеме Ruby on Rails разработчикам предлагается пользоваться общепринятыми соглашениями для того, чтобы ускорить разработку. Например, названия таблиц в БД должны называться множественным числом модели, для которой создаётся таблица. Также в настройках самого фреймворка присутствует множество дефолтных значений. Это не означает, что разработчик не может сконфигурировать приложение по своему желанию. Такая возможность, безусловно, присутствует, однако предлагается пользоваться соглашениями. Это не только ускоряет разработку, но и увеличивает уровень взаимопонимая в команде.

– “DRY” - что расшифровывается, как “don’t repeat yourself”. Это означает избегать дублирования кода. Желательно писать приложение таким образом, чтобы оно легко масштабировалось, методы и компоненты могли быть переиспользованы в различных местах.

В Ruby on Rails используется ORM ActiveRecord. ORM представляет собой прослойку между логической моделью и её представлением в БД. ActiveRecord предоставляет богатый функционал для работы с моделями. ActiveRecord - это также шаблон проектирования. Принцип работы ActiveRecord состоит в том, что в приложении создаётся класс, который отображается на таблицу в БД, так что:

– любой объект класса, унаследованного от ActiveRecord::Base отображается на строку в таблице БД;

– чтобы создать новую запись в таблице, нужно создать новый валидный объект;

– в качестве свойств объекта выступают поля в соответствующей строке БД;

– строка в таблице БД изменяется или удаляется, если изменяется

или удаляется объект ActiveRecord::Base;

Также ActiveRecord предоставляет инструмент манипулирования таблицами посредством миграций - методов, в которых на языке Ruby описываются действия с БД. Для создания столбцов таблиц ActiveRecord предлагает следующие типы:

- binary
- boolean
- date
- datetime
- decimal
- float
- integer
- bigint
- primary_key
- references
- string
- text
- time
- timestamp

Веб-фреймворки в большинстве своём обладают HTML-процессорами, которые позволяют дополнять HTML-разметку различными вставками, и Ruby on Rails не является исключением. Стандартный HTML-процессор в Ruby on Rails называется ERB. Он позволяет помещать в HTML-разметку Ruby-код, который создаётся в контроллере, тем самым давая возможность разработчикам избежать статических HTML-страниц. Однако помимо ERB существуют и другие HTML-процессоры, которые предоставляют возможность не только дополнять HTML-разметку, но и вводить новый вид разметки веб-страниц со вставками Ruby-кода.

В Ruby on Rails присутствует инструмент для соответствия URL-адреса определённому методу в контроллере. Такой инструмент называется router или маршрутизатор. В проекте, созданном на базе Ruby on Rails, существует файл routes.rb, в который помещаются так называемые маршруты. В маршрутизаторе есть возможность не только создавать маршруты не только по одному, но и целыми группами с помощью методы resources, который предлагает создание всех необходимый операций для ресурса: CREATE, READ, UPDATE, DELETE - посредством HTTP запросов.

Выбор фреймворка Ruby on Rails для разработки обусловлен удоб-

ством его использования, расширяемостью его с помощью различных библиотек, созданных Ruby on Rails-сообществом, которое на протяжении не менее десяти лет активно развивается и предлагает богатый выбор инструментов для разработки.

2.2 AngularJS

AngularJS - javascript-фреймворк, созданный в компании Google, совершивший прорыв в веб-разработке. AngularJS позволяет создавать динамические SPA. AngularJS, как и Ruby on Rails, обладает MVC-архитектурой. На момент создания AngularJS эта архитектура была революционной по причине того, что javascript-код преимущественно представлял отдельные неорганизованные скрипты, которые очень сложно масштабировать. AngularJS в свою очередь предлагает разделить код на контроллеры, представления и модели(ресурсы), которые в свою очередь могут находиться в отдельных модулях.

3 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ

Разработанное программное обеспечение представляет из себя веб-приложение, написанное на фреймворке Ruby on Rails. Приложение предназначено для получения новинок музыкальной индустрии или, другими словами, музыкальных релизов выбранных пользователем музыкантов.

Существует множество подходов для построения архитектуры веб-приложений. Я выбрал в своей реализации трёхуровневую архитектуру.

3.1 Трёхуровневая архитекутра приложения

Архитектура приложения - это определённый подход в определении взаимоотношений компонентов приложения. Обычно архитектура приложений соответствует задачам разработки. В данной дипломной работе выбрана трёхуровневая архитектура. Эти уровни называются: уровень модели, уровень контроллера и уровень представления. Иначе говоря, MVC - Model, View, Controller.

Данная архитектура используется во множестве веб-фреймворков. Она удобна тем, что позволяет разделить зоны ответственности между компонентами, облегчить тестирование, придать гибкость разработке. Фреймворк Ruby on Rails и AngularJS также в своей основе используют данную архитектуру.

3.1.1 Уровень модели

Уровень модели - один из трёх уровней в MVC-архитектуре. Его обязанности можно разделить на две категории:

– бизнес-логика приложения;

– управление связью в базой данных;

Бизнес-логика включает в себя определённые правила, отношения и принципы, которые должны существовать в приложении. Эти правила могут быть выражены как в коде самого приложения, так и базе данных. Однако в данной дипломной работе бизнес-логика содержится именно в коде приложения, а именно в ruby-классах, наследующихся от класса ActiveRecord::Base, который является стандартом де-факто для моделей, использующихся во фреймворке Ruby on Rails.

В данной дипломной работе присутствует класс ReleaseIdentity, который используется для хранения и обработки объектов музыкальных публи-

каций. Пример реализации класса ReleaseIdentity представлен на листинге 3.1:

Листинг 3.1 – Базовая реализация класса ReleaseIdentity

```
class ReleaseIdentity < ActiveRecord::Base
end
```

Классы-наследники класса ActiveRecord::Base обладают богатым функционалом. Они содержат в себе возможности валидирования, организации связей между объектами, возможностями создания транзакций, сложных запросов в БД и многими другими. Подробнее об этом будет написано несколько позже, когда будет рассматриваться взаимодействие с БД.

Ярким примером бизнес-логики может служить проверка корректности конструируемых объектов. Например, название артиста не должно быть пустым и содержать только текстовые данные. Или же год начала карьеры артиста не может быть отрицательными или больше нынешнего года по значению. Всё это реализуется в Ruby on Rails с помощью механизма валидации.

В данной дипломной работе присутствует класс RymArtistsInfo, который используется для хранения результатов работы с сервисом Rate Your Music[1)]. В этом классе присутствуют следующие поля:

– id - уникальный идентификатор записи в таблице;

– formed year - год формирования музыкальной группы(исполнителя);

– disbanded yead - год окончания деятельности музыкальной группы(исполнителя);

– genres - жанры, которые присущи музыкальной группе(исполнителю);

– created at - дата создания записи в БД приложения;

– updated at - дата обновления записи в БД приложения;

В данном случае годы формирования и окончания деятельности музыкальной группы исполнителя должны придерживаться правил, описанных выше. Также строка, содержащая жанры, не может иметь длину, менее 2 символов. Для обеспечения этой части бизнес-логики я воспользовался инструментарием фреймворка Ruby on Rails. Используя метод validates, можно добиться проверки определённых правил перед сохранением записи, чтобы хранить в базе только корректные записи. Также существует метод validate, в который передаётся название метода, в котором можно описать более сложную логику. Таким образом можно проверить то, что год окончания деятельности музыкальной группы больше или равен го-

[1)] https://rateyourmusic.com/

ду начала карьеры. Пример использования продемонтрирован на листинге 3.2:

Листинг 3.2 – Реализация валидации в классе RymArtistsInfo

```
class RymArtistsInfo < ActiveRecord::Base
  validates :formed_year
  validates :disbanded_year, numericality: {only_integer: true, greated_than: 0,
      less_than: DateTime.now.year}
  validates :genres, length: {minimum: 2}
  validate disbanded_more_or_eql_formed

  private

  def disbanded_more_or_equal_formed
    disbanded_year >= formed_year
  end
end
```

Фреймворк Ruby on Rails предлагает ещё один полезный инструмент, который называется "функция обратного вызова". Такие функции существуют и на уровне базы данных, но в этом и состоит преимущество веб-фреймворка - позволять более компактно и удобно писать необходимый функционал. Функции обратного вызова работают таким образом, что в определённые моменты жизненного цикла объектов, такие, как перед сохранением, удалением или валидированием объекта, вызываются методы, в который можный поместить необходимый для этого момента код. Это ещё одно преимущество слоя модели, потому как позволяет избегать длинных цепочек вызовов методов, а воспользоваться удобным интерфейсом для обработки моментов жизненного цикла. В Ruby on Rails такие моменты обрабатываются путём добавления методов:

- before_save;
- before_create;
- before_validate;
- around_save;
- after_save;
- after_update;
- и др.;
- КОД КОЛБЭКА –

Стоит обратить внимание на тот факт, что с ростом приложений модели включают в себя всё больше логики и становятся очень большими. Минус этого состоит в том, что происходит нарушение одного из принципов объектно-ориентированного программирования. Это принцип единственной ответственности класса. Он нарушается в тот момент, когда класс

выполняет много различных функций: используется для подключения к БД, содержания бизнес-логики, создания других объектов и т.д. Однако существует способ улучшения данной архитектуры путём введения в слой модели дополнительного компонента - сервиса. Сервисы - это такие классы или модули, который содержат в себе код, который можно извлечь из модели, чтобы её не перегружать. Например, сервисы могут содержать в себе код обращения в стороннему приложению, либо расчёт цен и многое другое. Более того, код сервисов может использоваться в различных местах, тем самым обеспечивая модульность. Это помогает проще тестировать. Кроме этого, сервисы реализуют принцип инкапсуляции, то есть принцип сокрытия реализации. Имеется в виду, что коду, использующему сервис, необязательно знать, как именно реализован тот или иной метод. Если сервис отвечает за работу с поиском и передачей музыкальных публикаций, то коду, работающему с этим сервисом не важно, берёт сервис публикации с помощью API, делает запрос в БД или же эти публикации просто лежат в массиве в памяти приложения.

– НАПИСАТЬ ПРИМЕР КАКОГОНИБУДЬ СЕРВИСА –

Также слой модели ответственнен за работу с базой данных. Большим преимуществом данного слоя вообще и конкретно в реализации ActiveRecord является то, что данная ORM имеет один интерфейс для работы с различными базами данных, будь то PostgreSQL, MySQL, SQLite или другие. Используя классы, унаследованные от ActiveRecord::Base, можно обращаться к базе данных для того, чтобы получить список артистов, релизов или пользователей с помощью языка Ruby. Например, если необходимо сделать выборку артистов, год начала деятельности которых больше какого-то определённого, при этом не выбирая все поля, а только год прекращения деятельности и дату создания записи в БД, используется код, указанный в листинге 3.3:

Листинг 3.3 – Пример получения артистов по определённым параметрам

```
artists = RymArtistsInfo.select(:disbanded_year, :created_at).where('formed_year > ?',
    Date.new(2000, 4, 3))
```

Таким образом, классы, унаследованные от ActiveRecord::Base предоставляют удобный интерфейс, который помогает не писать чистый SQL-код. Возможности ActiveRecord::Base не ограничиваются простыми запросами. Существуют методы, которые позволяют делать сложные JOIN-запросы, избегать проблемы "N+1"и т.д.

4 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ СЕРВИСА, ПРЕДОСТАВЛЯЮЩЕГО ВОЗМОЖНОСТЬ ПОЛУЧЕНИЯ НОВИНОК МУЗЫКАЛЬНОЙ ИНДУСТРИИ

4.1 Введение и исходные данные

Целью данной дипломной работы является создание программного продукта, позволяющего пользователю получать новинки музыкальной индустрии в виде уведомлений о недавно вышедших публикациях выбранных им музыкальных исполнителей. Данный программный продукт позволит получать информацию из нескольких источников, тем самым помогая пользователю удобнее ориентироваться среди различных сервисов, получать только необходимую и интересную информацию. Это очень эффективно, поскольку сокращается время на поиски новых публикаций самостоятельно. В экономическом смысле - это сократит расходы на подписки на несколько сервисов, которые занимаются размещением на своей платформе новых музыкальных публикаций.

Для разработчика экономическая эффективность будет определяться как прибыль от реализации, а для пользователя - как экономия затрат. Для определения экономической эффективности рассчитаем смету затрат и цену программного продукта.

4.2 Расчёт сметы затрат и цены программного продукта

Для представления программного продукта на рынке он должен быть законченным, иметь презентабельный вид. Для реализации программного продукта необходимо пройти через два этапа: этап, связанный с разработкой ПО(выяснение требований, программирование, тестирование, отладка), и этап, связанный с непосредственной реализацией продукта на рынке (реализация, поддержка).

Программный комплекс относится ко 2-й группе сложности. Категория новизны – “В”. Расчеты выполнены на основе методического пособия [1].

Исходные данные для проекта указаны в таблице 4.1.

На основании сметы затрат и анализа рынка ПО определяется плановая отпускаемая цена. Для составления сметы затрат на создание ПО

Таблица 4.1 – Исходные данные

Наименование показателей	Буквенные обозначения	Единицы измерения	Количество
Коэффициент новизны	$К_н$	единиц	0,9
Группа сложности		единиц	2
Дополнительный коэффициент сложности	$К_{сл}$	единиц	1,08
Поправочный коэффициент, учитывающий использование типовых программ	$К_т$	единиц	0,7
Установленная плановая продолжительность разработки	$Т_р$	лет	0,33
Продолжительность рабочего дня	$Т_ч$	часов	8
Тарифная ставка 1-го разряда	$Т_{м1}$	руб.	250
Коэффициент премирования	$К_п$	единиц	1,5
Норматив дополнительной заработной платы исполнителей	$Н_д$	%	20
Отчисления в фонд социальной защиты населения	$З_{сз}$	%	34
Отчисления в Белгосстрах	$Н_{нс}$	%	0,6
Расходы на научные командировки	$Р_{нк}$	%	7
Прочие прямые расходы	$П_з$	%	10
Прочие накладные расходы	$П_{рн}$	%	8
Налог на прибыль	$П_{рн}$	%	18

необходима предварительная оценка трудоемкости ПО и его объёма. Расчет объёма программного продукта (количества строк исходного кода) предполагает определение типа программного обеспечения, всестороннее техническое обоснование функций ПО и определение объёма каждой функций. Согласно классификации типов программного обеспечения [1, с. 59, приложение 1], разрабатываемое ПО с наименьшей ошибкой можно классифицировать как ПО методо-ориентированных расчетов.

Общий объём программного продукта определяется исходя из количества и объёма функций, реализованных в программе:

$$V_o = \sum_{i=1}^{n} V_i \,, \tag{4.1}$$

где V_i — объём отдельной функции ПО, LoC;
n — общее число функций.

На стадии технико-экономического обоснования проекта рассчитать точный объём функций невозможно. Вместо вычисления точного объёма функций применяются приблизительные оценки на основе данных по аналогичным проектам или по нормативам [1, с. 61, приложение 2], которые приняты в организации.

Таблица 4.2 – Перечень и объём функций программного модуля

№ функции	Наименование (содержание)	Объём функции, LoC	
		по каталогу (V_i)	уточненный ($V_i^{\text{у}}$)
101	Организация ввода информации	100	60
102	Контроль, предварительная обработка и ввод информации	520	520
111	Управление вводом/выводом	2700	700
304	Обслуживание файлов	520	580
305	Обработка файлов	750	750
309	Формирование файла	1100	1100
506	Обработка ошибочных и сбойных ситуаций	430	430
507	Обеспечение интерфейса между компонентами	730	730
605	Вспомогательные и сервисные программы	460	280
701	Математическая статистика и прогнозирование	8370	3500
Итог		8980	8400

Каталог аналогов программного обеспечения предназначен для предварительной оценки объёма ПО методом структурной аналогии. В разных организациях в зависимости от технических и организационных условий, в которых разрабатывается ПО, предварительные оценки могут корректироваться на основе экспертных оценок. Уточненный объём ПО рассчитывается по формуле:

$$V_{\text{у}} = \sum_{i=1}^{n} V_i^{\text{у}}\,, \tag{4.2}$$

где $V_i^{\text{у}}$ — уточненный объём отдельной функции ПО, LoC;
n — общее число функций.

Перечень и объём функций программного модуля перечислен в таблице 4.2. По приведенным данным уточненный объём некоторых функций изменился, и общий объём ПО составил $V_o = 8980\,\text{LoC}$, общий уточненный общем ПО — $V_{\text{у}} = 8400\,\text{LoC}$.

По уточненному объёму ПО и нормативам затрат труда в расчете на единицу объёма определяются нормативная и общая трудоемкость разработки ПО. Уточненный объём ПО — 8400 LoC. ПО относится ко второй категории сложности: предполагается его использование для сложных статистических расчетов и решения задач классификации, также необходимо обеспечить переносимость ПО [1, с. 66, приложение 4, таблица П.4.1]. По полученным данным определяется нормативная трудоемкость разработки ПО. Согласно укрупненным нормам времени на разработку ПО в зависимости от уточненного объёма ПО и группы сложности ПО [1, с. 64, приложение 3] нормативная трудоемкость разрабатываемого проекта составляет $\text{Т}_{\text{н}} = 224$ чел./дн.

Нормативная трудоемкость служит основой для оценки общей трудоемкости $\text{Т}_{\text{о}}$. Используем формулу (4.3) для оценки общей трудоемкости для небольших проектов:

$$\text{Т}_{\text{о}} = \text{Т}_{\text{н}} \cdot \text{К}_{\text{с}} \cdot \text{К}_{\text{т}} \cdot \text{К}_{\text{н}}\,, \tag{4.3}$$

где $\text{К}_{\text{с}}$ — коэффициент, учитывающий сложность ПО;
$\text{К}_{\text{т}}$ — поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей;
$\text{К}_{\text{н}}$ — коэффициент, учитывающий степень новизны ПО.

Дополнительные затраты труда на разработку ПО учитываются через коэффициент сложности, который вычисляется по формуле

$$\text{К}_{\text{с}} = 1 + \sum_{i=1}^{n} \text{К}_i\,, \tag{4.4}$$

где К_i — коэффициент, соответствующий степени повышения сложности ПО за счет конкретной характеристики;
n — количество учитываемых характеристик.

Наличие двух характеристик сложности позволяет [1, с. 66, приложение 4, таблица П.4.2] вычислить коэффициент сложности

$$К_с = 1 + 0{,}08 = 1{,}08\,. \tag{4.5}$$

Разрабатываемое ПО использует стандартные компоненты. Степень использования стандартных компонентов определяется коэффициентом использования стандартных модулей — $К_т$. Согласно справочным данным [1, с. 68, приложение 4, таблица П.4.5] указанный коэффициент для разрабатываемого приложения $К_т = 0{,}7$. Трудоемкость создания ПО также зависит от его новизны и наличия аналогов. Разрабатываемое ПО не является новым, существуют аналогичные более зрелые разработки у различных компаний и университетов по всему миру. Влияние степени новизны на трудоемкость создания ПО определяется коэффициентом новизны — $К_н$. Согласно справочным данным [1, с. 67, приложение 4, таблица П.4.4] для разрабатываемого ПО $К_н = 0{,}9$. Подставив приведенные выше коэффициенты для разрабатываемого ПО в формулу (4.3) получим общую трудоемкость разработки

$$Т_о = 224 \times 1{,}08 \times 0{,}7 \times 0{,}9 \approx 152 \text{ чел./дн.} \tag{4.6}$$

На основе общей трудоемкости и требуемых сроков реализации проекта вычисляется плановое количество исполнителей. Численность исполнителей проекта рассчитывается по формуле:

$$Ч_р = \frac{Т_о}{Т_р \cdot Ф_{эф}}\,, \tag{4.7}$$

где $Т_о$ — общая трудоемкость разработки проекта, чел./дн.;
$Ф_{эф}$ — эффективный фонд времени работы одного работника в течение года, дн.;
$Т_р$ — срок разработки проекта, лет.

Эффективный фонд времени работы одного разработчика вычисляется по формуле

$$Ф_{эф} = Д_г - Д_п - Д_в - Д_о\,, \tag{4.8}$$

где $Д_г$ — количество дней в году, дн.;
$Д_п$ — количество праздничных дней в году, не совпадающих с выходными днями, дн.;
$Д_в$ — количество выходных дней в году, дн.;
$Д_п$ — количество дней отпуска, дн.

Согласно данным, приведенным в производственном календаре для пятидневной рабочей недели в 2013 году для Беларуси [2], фонд рабочего времени составит

$$Ф_{эф} = 365 - 9 - 104 - 21 = 231 \text{ дн.} \tag{4.9}$$

Учитывая срок разработки проекта $Т_р = 4$ мес. $= 0{,}33$ года, общую трудоемкость и фонд эффективного времени одного работника, вычисленные ранее, можем рассчитать численность исполнителей проекта

$$Ч_р = \frac{152}{0{,}33 \times 231} \approx 2 \text{ рабочих.} \tag{4.10}$$

Вычисленные оценки показывают, что для выполнения запланированного проекта в указанные сроки необходимо два рабочих. Информация о работниках перечислена в таблице 4.3.

Таблица 4.3 – Работники, занятые в проекте

Исполнители	Разряд	Тарифный коэффициент	Чел./дн. занятости
Программист I-категории	13	3,04	76
Ведущий программист	15	3,48	76

Месячная тарифная ставка одного работника вычисляется по формуле

$$Т_ч = \frac{Т_{м_1} \cdot Т_к}{Ф_р}, \tag{4.11}$$

где $Т_{м_1}$ — месячная тарифная ставка 1-го разряда, Br;
$Т_к$ — тарифный коэффициент, соответствующий установленному тарифному разряду;
$Ф_р$ — среднемесячная норма рабочего времени, час.

Подставив данные из таблицы 4.3 в формулу (4.11), приняв значение тарифной ставки 1-го разряда $\text{Т}_{\text{м}_1} = 250\,\text{Br}$ и среднемесячную норму рабочего времени $\Ф_{\text{р}} = 160$ часов получаем

$$\text{Т}_{\text{ч}}^{\text{прогр. I-разр.}} = \frac{250 \times 3{,}04}{160} = 5\,\text{Br/час}; \tag{4.12}$$

$$\text{Т}_{\text{ч}}^{\text{вед. прогр.}} = \frac{250 \times 3{,}48}{160} = 5\,\text{Br/час}. \tag{4.13}$$

Основная заработная плата исполнителей на конкретное ПО рассчитывается по формуле

$$\text{З}_{\text{о}} = \sum_{i=1}^{n} \text{Т}_{\text{ч}}^{i} \cdot \text{Т}_{\text{ч}} \cdot \Ф_{\text{п}} \cdot \text{К}\,, \tag{4.14}$$

где $\text{Т}_{\text{ч}}^{i}$ — часовая тарифная ставка i-го исполнителя, Br/час;
$\text{Т}_{\text{ч}}$ — количество часов работы в день, час;
$\Ф_{\text{п}}$ — плановый фонд рабочего времени i-го исполнителя, дн.;
К — коэффициент премирования.

Подставив ранее вычисленные значения и данные из таблицы 4.3 в формулу (4.14) и приняв коэффициент премирования $\text{К} = 1{,}5$ получим

$$\text{З}_{\text{о}} = (5 \times 76 + 5 \times 76) \times 8 \times 1{,}5 = 9120\,\text{Br}\,. \tag{4.15}$$

Дополнительная заработная плата включает выплаты предусмотренные законодательством от труде и определяется по нормативу в процентах от основной заработной платы

$$\text{З}_{\text{д}} = \frac{\text{З}_{\text{о}} \cdot \text{Н}_{\text{д}}}{100\%}\,, \tag{4.16}$$

где $\text{Н}_{\text{д}}$ — норматив дополнительной заработной платы, %.

Приняв норматив дополнительной заработной платы $\text{Н}_{\text{д}} = 20\%$ и подставив известные данные в формулу (4.16) получим

$$\text{З}_{\text{д}} = \frac{9120 \times 20\%}{100\%} \approx 1824\,\text{Br}\,. \tag{4.17}$$

Согласно действующему законодательству отчисления в фонд соци-

альной защиты населения составляют 34% , в фонд обязательного страхования — 0,6%, от фонда основной и дополнительной заработной платы исполнителей. Общие отчисления на социальную защиту рассчитываются по формуле

$$З_{сз} = \frac{(З_о + З_д) \cdot Н_{сз}}{100\%}. \tag{4.18}$$

Подставив вычисленные ранее значения в формулу (4.18) получаем

$$З_{сз} = \frac{(9120 + 1824) \times 34{,}6\%}{100\%} \approx 3787\,\text{Br}. \tag{4.19}$$

По статье «материалы» проходят расходы на носители информации, бумагу, краску для принтеров и другие материалы, используемые при разработке ПО. Норма расходов $Н_{мз}$ определяется либо в расчете на 100 строк исходного кода, либо в процентах к основной зарплате исполнителей 3% — 5%. Затраты на материалы вычисляются по формуле

$$М = \frac{З_о \cdot Н_{мз}}{100\%} = \frac{9120 \times 3\%}{100\%} \approx 273{,}6\,\text{Br}. \tag{4.20}$$

Расходы по статье «машинное время» включают оплату машинного времени, необходимого для разработки и отладки ПО, которое определяется по нормативам в машино-часах на 100 строк исходного кода в зависимости от характера решаемых задач и типа ПК, и вычисляются по формуле

$$Р_м = Ц_м \cdot \frac{V_о}{100} \cdot Н_{мв}, \tag{4.21}$$

где $Ц_м$ — цена одного часа машинного времени, Br;
$Н_{мв}$ — норматив расхода машинного времени на отладку 100 строк исходного кода, часов.

Согласно нормативу [1, с. 69, приложение 6] норматив расхода машинного времени на отладку 100 строк исходного кода составляет $Н_{мв} = 15$, применяя понижающий коэффициент 0,3 получаем $Н'_{мв} = 4{,}5$. Цена одного часа машинного времени составляет $Ц_м = 5000\,\text{Br}$. Подставляя известные данные в формулу (4.21) получаем

$$Р_м = 5000 \times \frac{8400}{100} \times 4{,}5 = 1\,890\,000\,\text{Br}. \tag{4.22}$$

Расходы по статье «научные командировки» вычисляются как про-

цент от основной заработной платы, либо определяются по нормативу. Вычисления производятся по формуле

$$Р_к = \frac{З_о \cdot Н_к}{100\%}, \tag{4.23}$$

где $Н_к$ — норматив командировочных расходов по отношению к основной заработной плате, %.

Подставляя ранее вычисленные значения в формулу (4.23) и приняв значение $Н_к = 7\%$ получаем

$$Р_к = \frac{9120 \times 7\%}{100\%} = 638\,\text{Br}. \tag{4.24}$$

Статья расходов «прочие затраты» включает в себя расходы на приобретение и подготовку специальной научно-технической информации и специальной литературы. Затраты определяются по нормативу принятому в организации в процентах от основной заработной платы и вычисляются по формуле

$$П_з = \frac{З_о \cdot Н_{пз}}{100\%}, \tag{4.25}$$

где $Н_{пз}$ — норматив прочих затрат в целом по организации, %.

Приняв значение норматива прочих затрат $Н_{пз} = 10\%$ и подставив вычисленные ранее значения в формулу (4.25) получаем

$$П_з = \frac{9120 \times 10\%}{100\%} = 912\,\text{Br}. \tag{4.26}$$

Статья «накладные расходы» учитывает расходы, необходимые для содержания аппарата управления, вспомогательных хозяйств и опытных производств, а также расходы на общехозяйственные нужны. Данная статья затрат рассчитывается по нормативу от основной заработной платы и вычисляется по формуле.

$$Р_н = \frac{З_о \cdot Н_{рн}}{100\%}, \tag{4.27}$$

где $Н_{рн}$ — норматив накладных расходов в организации, %.

Приняв норму накладных расходов $Н_{рн} = 8\%$ и подставив известные данные в формулу (4.27) получаем

$$Р_н = \frac{9120 \times 8\%}{100\%} = 730\,\text{Br}. \tag{4.28}$$

Общая сумма расходов по смете на ПО рассчитывается по формуле

$$С_р = З_о + З_д + З_{сз} + М + Р_м + Р_{нк} + П_з + Р_н. \tag{4.29}$$

Подставляя ранее вычисленные значения в формулу (4.29) получаем

$$С_р = 1\,907\,284{,}6\,\text{Br}. \tag{4.30}$$

Расходы на сопровождение и адаптацию, которые несет производитель ПО, вычисляются по нормативу от суммы расходов по смете и рассчитываются по формуле

$$Р_{са} = \frac{С_р \cdot Н_{рса}}{100\%}, \tag{4.31}$$

где $Н_{рса}$ — норматив расходов на сопровождение и адаптацию ПО, %.

Приняв значение норматива расходов на сопровождение и адаптацию $Н_{рса} = 30\%$ и подставив ранее вычисленные значения в формулу (4.31) получаем

$$Р_{са} = \frac{1\,907\,284{,}6 \times 30\%}{100\%} \approx 572\,185\,\text{Br}. \tag{4.32}$$

Полная себестоимость создания ПО включает сумму затрат на разработку, сопровождение и адаптацию и вычисляется по формуле

$$С_п = С_р + Р_{са}. \tag{4.33}$$

Подставляя известные значения в формулу (4.33) получаем

$$С_п = 1\,907\,284{,}6 + 572\,185 = 2\,479\,469{,}6\,\text{Br}. \tag{4.34}$$

4.3 Расчёт экономической эффективности у разработчика

Важная задача при выборе проекта для финансирования это расчет экономической эффективности проектов и выбор наиболее выгодного проекта. Разрабатываемое ПО является заказным, т. е. разрабатывается для

одного заказчика на заказ. На основании анализа рыночных условий и договоренности с заказчиком об отпускной цене прогнозируемая рентабельность проекта составит $У_{рп} = 35\%$. Прибыль рассчитывается по формуле

$$П_с = \frac{С_п \cdot У_{рп}}{100\%}, \tag{4.35}$$

где $П_с$ — прибыль от реализации ПО заказчику, Br;
$У_{рп}$ — уровень рентабельности ПО, %.

Подставив известные данные в формулу (4.35) получаем прогнозируемую прибыль от реализации ПО

$$П_с = \frac{2\,479\,469{,}6 \times 35\%}{100\%} \approx 867\,814\,\text{Br}. \tag{4.36}$$

Прогнозируемая цена ПО без учета налогов включаемых в цену вычисляется по формуле

$$Ц_п = С_п + П_с. \tag{4.37}$$

Подставив данные в формулу (4.37) получаем цену ПО без налогов

$$Ц_п = 2\,479\,469{,}6 + 867\,814 = 3\,347\,283{,}6\,\text{Br}. \tag{4.38}$$

Налог на добавленную стоимость рассчитывается по формуле

$$НДС = \frac{Ц_п \cdot Н_{дс}}{100\%}, \tag{4.39}$$

где $Н_{дс}$ — норматив НДС, %.

Норматив НДС составляет $Н_{дс} = 20\%$, подставляя известные значения в формулу (4.39) получаем

$$НДС = \frac{3\,347\,283{,}6 \times 20\%}{100\%} \approx 669\,457\,\text{Br}. \tag{4.40}$$

Расчет прогнозируемой отпускной цены осуществляется по формуле

$$Ц_о = Ц_п + НДС. \tag{4.41}$$

Подставляя известные данные в формулу (4.41) получаем прогнози-

руемую отпускную цену

$$Ц_{о} = 3\,347\,283{,}6 + 669\,457 \approx 4\,016\,740{,}6\,\text{Br}\,. \tag{4.42}$$

Чистую прибыль от реализации проекта можно рассчитать по формуле

$$П_{ч} = П_{с} \cdot \left(1 - \frac{Н_{п}}{100\%}\right), \tag{4.43}$$

где $Н_{п}$ — величина налога на прибыль, %.

Приняв значение налога на прибыль $Н_{н} = 18\%$ и подставив известные данные в формулу (4.43) получаем чистую прибыль

$$П_{ч} = 867\,814 \times \left(1 - \frac{18\%}{100\%}\right) = 711\,607\,\text{Br}\,. \tag{4.44}$$

Программное обеспечение разрабатывалось для одного заказчика в связи с этим экономическим эффектом разработчика будет являться чистая прибыль от реализации $П_{ч}$. Рассчитанные данные приведены в таблице 4.4. Таким образом было произведено технико-экономическое обоснование разрабатываемого проекта, составлена смета затрат и рассчитана прогнозируемая прибыль, и показана экономическая целесообразность разработки.

Таблица 4.4 – Рассчитанные данные

Наименование	Условное обозначение	Значение
Нормативная трудоемкость, чел./дн.	$Т_н$	224
Общая трудоемкость разработки, чел./дн.	$Т_о$	152
Численность исполнителей, чел.	$Ч_р$	2
Часовая тарифная ставка программиста I-разряда, Br/ч.	$Т_ч^{прогр.\ I\text{-}разр.}$	5
Часовая тарифная ставка ведущего программиста, Br/ч.	$Т_ч^{вед.\ прогр.}$	5
Основная заработная плата, Br	$З_о$	9120
Дополнительная заработная плата, Br	$З_д$	1824
Отчисления в фонд социальной защиты, Br	$З_{сз}$	3787
Затраты на материалы, Br	М	273,6
Расходы на машинное время, Br	$Р_м$	1 890 000
Расходы на командировки, Br	$Р_к$	638
Прочие затраты, Br	$П_з$	912
Накладные расходы, Br	$Р_н$	730
Общая сумма расходов по смете, Br	$С_р$	1 907 284,6
Расходы на сопровождение и адаптацию, Br	$Р_{са}$	572 185
Полная себестоимость, Br	$С_п$	2 479 469,6
Прогнозируемая прибыль, Br	$П_с$	867 814
НДС, Br	НДС	669 457
Прогнозируемая отпускная цена ПО, Br	$Ц_о$	4 016 740,6
Чистая прибыль, Br	$П_ч$	711 607

ЗАКЛЮЧЕНИЕ

В данном дипломном проекте был рассмотрен вопрос автоматического построения структуры вероятностной сети на основе экспериментальных данных. В рамках дипломного проекта была разработана библиотека кода для представления и автоматического построения структуры сети. В разработанной библиотеке использовались два различных подхода к оценке качества сети, на основе принципа МДО и оценке апостериорной вероятности структуры для имеющихся экспериментальных данных. Также для разных оценок использовались разные стратегии поиска оптимальной структуры сети в пространстве возможных решений.

В целом были получены удовлетворительные результаты на хорошо изученных и известных сетях Asia и ALARM. Результаты работы реализованных в библиотеке функций поиска в большинстве случаях превосходят по качеству функциональность уже существующего программного обеспечения. Также был предложен способ улучшения качества обучаемой сети на малом объеме данных, основанный на предварительной рандомизации экспериментальных данных. Данный способ удовлетворительно зарекомендовал себя в проведённых тестах. Помимо предложенной модификации были произведены небольшие улучшения в хорошо известных алгоритмах, направленные на повышение скорости их работы. Для повышения производительности применялась мемоизация и использовались прологарифмированные версии некоторых оценок.

В результате цель дипломного проекта была достигнута. Было создано программное обеспечение. Но за рамками рассматриваемой темы осталось еще много других алгоритмов вывода структуры и интересных вопросов, связанных, например, со статистическим выводом суждений в вероятностных сетях, нахождением параметров распределения и других вопросов, возникающих при работе с вероятностными сетями. Эти задачи также являются нетривиальными и требуют детального изучения и проработки — задача статистического вывода, например, является $\mathcal{NP}$-трудной [3] — и не рассматриваются в данном дипломном проекте из-за временных ограничений на его создание. В дальнейшем планируется развивать и довести существующее ПО до полноценной библиотеки, способной решать более широкий класс задач, возникающих в области применения вероятностных сетей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Палицын, В. А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В. А. Палицын. — Минск : БГУИР, 2006. — 76 с.

[2] Календарь праздников на 2013 год для Беларуси [Электронный ресурс]. — Электронные данные. — Режим доступа: `http://calendar.by/2013/#bkm`. — Дата доступа: 05.03.2013.

[3] Koller, D. Probabilistic Graphical Models: Principles and Techniques / D. Koller, N. Friedman. — MIT Press, 2009. — 1270 P.