МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

Информационная система сервиса по подбору электрогитары

Студент _____ Синцев Никита Алексеевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Прошкина Е.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Макарычев П.П.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№_____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2021

Федеральное государственное бюджетное образовательное учреждение
высшего образования
Пензенский государственный университет
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
П.П. Макарычев
«__»_____20__ г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Синцев Никита Алексеевич гр. 17ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>
2.Тема работы Информационная система сервиса по подбору электрогитары

Тема утверждена приказом ПГУ №_____от "___" _____20 _ г.

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Прошкина Е.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Разработать информационную систему сервиса по подбору электрогитары в виде web-приложения

2) Среда выполнения:

Языковые средства разработки: Java, HTML, CSS

Инструментальные средства разработки: IntelliJ Idea Community Edition, фреймворк Spring, MySQL Workbench

Технологии разработки: ООП

3) Основные функции информационной системы:

а) регистрация и авторизация пользователей;

б) предоставление возможности подбора электрогитары на основе тестирования пользователя;

в) предоставить возможность просмотра каталога инструментов в базе данных сервиса;

г) предоставить возможность просмотра результата последнего пройденного тестирования;

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	*Анализ предметной области и постановка задачи*	10%	14.04.2021 – 20.04.2021	
2	*Проектирование*	25%	20.04.2021 – 30.04.2021	
3	*Реализация*	30%	30.04.2021 – 10.05.2021	
4	*Контроль качества программного обеспечения*	5%	19.05.2021 – 22.05.2021	
5	*Оформление пояснительной записки*	10%	22.05.2021 – 14.06.2021	

Дата выдачи задания " 28" декабря 2020 г

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 79 листов, 44 рисунка, 11 таблиц, 16 листингов, 2 приложения, 20 использованных источников.

ИНФОРМАЦИОННАЯ СИСТЕМА, СЕРВИС, ЭЛЕКТРОГИТАРА, БАЗА ДАННЫХ, ФРЕЙМВОРК Spring, Java, UML 2.0, Tomcat 9.0.

Объектом разработки являются программные средства автоматизации процесса выбора электрогитары.

Цель работы – разработка информационной системы сервиса по подбору электрогитары.

В результате проведенной работы создана информационная система сервиса по подбору электрогитары в виде web-приложения, реализующее подбор электрогитары по тестированию пользователя.

Разработка проводилась на языке программирования Java, в качестве веб-сервиса использовался Tomcat 9.0, в качестве сред разработки использовались среда программирования IntelliJ Idea Community Edition и фреймворк Spring.

Содержание

Введение 7

1 Анализ предметной области и требований 9

1.1 Анализ предметной области и постановка задач 9

1.2 Обзор систем, позволяющих осуществить подбор электрогитары 11

1.3 Анализ требований 17

1.3.1 Функциональные требования 17

1.3.2 Нефункциональные требования 18

2 Проектирование информационной системы 19

2.1 Выбор технологий проектирования 19

2.2 Проектирование логической модели базы данных 19

2.3 Проектирование программных средств 20

2.3.1 Диаграмма прецедентов 20

2.3.2 Диаграмма классов 22

2.3.3 Диаграмма состояний 25

2.3.4 Диаграмма деятельности 26

2.3.5 Диаграмма последовательности 28

2.3.6 Диаграмма кооперации 29

2.4 Проектирование интерфейса 30

2.5 Планирование и оценка бюджета 36

3 Реализация информационной системы 42

3.1 Физическая модель базы данных 42

3.2 Реализация программных средств 44

3.2.1 Реализация классов 44

3.2.2 Диаграмма компонентов 46

3.2.3 Диаграмма развертывания 47
3.3 Разработка пользовательского интерфейса 48
4 Контроль качества программного обеспечения 54
4.1 Функциональное тестирование 54
4.2 Метрики кода 56
Заключение 60
Список использованных источников 61
Приложение А. Листинг программы 63
Приложение Б. Результаты тестирования 75

Введение

Идея разработки сервиса для подбора электрогитары обусловлена, по большей части, полным или частичным непониманием принципов построения звука электрогитары начинающими музыкантами. Большинство из них так же испытывают проблемы, когда слышат специфичную терминологию, характерную для области гитаростроения. Скудное количество литературы о строении электрогитар и нежелание покупателей вникать в технические характеристики плохо сказываются на рынке музыкальных инструментов. Ситуацию усугубляет разрозненность мнений пользователей интернет-форумов, посвященных данной тематике. Такой ситуацией активно пользуются маркетологи крупных сетей магазинов и перекупщики подержанных инструментов, что приводит к ситуациям, в которых начинающие гитаристы получают негативный опыт владения неподходящим им инструментом и остаются незаинтересованными в дальнейшем обучении, а более опытные музыканты тратят деньги на гитары ненадлежащего качества.

Часть проблем, описанных выше, являются следствием неспособности неопытного музыканта проводить параллели между своими субъективными предпочтениями и техническими характеристиками инструментов, и могут быть решены лишь в индивидуальном порядке. Такое решение может предоставить информационная система, в основе которой лежит тестирование, выявляющее предпочтения пользователя в музыке и связывающее их с особенностями конкретных электрогитар.

Такая система могла бы как решить проблему выбора инструмента новичком, так и сформировать реестр гитар надлежащего качества во всех ценовых категориях, который могли бы использовать более опытные музыканты. Такой подход может быть реализован путем глубокого анализа предметной области с использованием опыта эксперта.

Целесообразно учитывать и современные тенденции разработки подобных систем. Самым практичным решением является веб-приложение, позволяющее пользователям легко и быстро получать доступ к системе.

Главным преимуществом такого подхода является возможность использования URL ссылок на страницы магазинов, либо поисковые страницы сайтов, ориентированных на продажу б/у товаров. Это позволит существенно снизить нагрузку на сервер приложения и упростить процедуру покупки инструмента для пользователя.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задач

В рамках анализа предметной области для понимания критериев подбора следует выделить основные части, из которых состоит любая электрогитара.

На электрогитару устанавливаются стальные струны, колебания которых звукосниматели преобразовывают в электрический ток. Электрогитара нуждается в специальном гитарном усилителе и кабинете, чтобы ее звук был слышим [1].

Тип звукоснимателя формирует основной характер звучания инструмента. В большинстве случаев используется 2 типа звукоснимателей.

Звукосниматель типа «сингл» – это звукосниматель для электрогитары с одной электромагнитной катушкой. Для этого типа звукоснимателей характерен слабый выходной сигнал. Чаще всего их используют в таких жанрах как джаз или блюз.

В то же время данные датчики обладают одним существенным недостатком. На датчик могут воздействовать различные электромагнитные поля, что может приводить к возникновению постороннего шума.

Звукосниматель типа «хамбакер» в отличие от звукоснимателя типа «сингл» в своей конструкции имеет две электромагнитные катушки, которые расположены рядом или одна над другой и всегда подключены таким образом, чтобы нейтрализовать шумы, создаваемые внешними электромагнитными полями, за счет суммарного сложения сигналов от двух электромагнитных катушек. В связи с этим выходной сигнал таких звукоснимателей является достаточно большим. Из-за большого выходного сигнала звукосниматели типа «хамбакер» пользуются популярностью в более агрессивных жанрах музыки [2].

В рамках ассоциативного подбора электрогитары следует так же рассмотреть возможные формы инструментов, популяризированные определёнными музыкальными исполнителями.

Самой популярной формой электрогитары является Fender Stratocaster, который в тоже время является самым копируемым в мире инструментом. Классический Stratocaster укомплектован тремя звукоснимателями типа «сингл», а более современные аналоги имеют конфигурацию, включающую в себя один звукосниматель типа «хамбакер» и два звукоснимателя типа «сингл». Популяризаторами данной формы являются Eric Clapton (The Yardbirds), Yngwie Malmsteen, Ritchie Blackmore (Deep Purple) и др.

Гитара формы Stratocaster изображена на рисунке 1.

Рисунок 1 – Fender Stratocaster

Вторая по популярности в мире гитара, однозначно, Gibson Les Paul. Материалом для корпуса этой гитары служит махагони (красное дерево). Гриф также изготавливают из красного дерева и вклеивают в корпус. Чаще всего на Gibson Les Paul установлены два звукоснимателя типа «хамбакер», а в некоторых модификациях их может быть даже три. Эта форма гитары популяризировалась благодаря таким музыкантам, как Jimmy Page (Led Zeppelin), Gary Moore, Шура (Би-2) и др.

Гитара формы Les Paul изображена на рисунке 2.

Рисунок 2 – Gibson Les Paul

Электрогитара Fender Telecaster является первой в мире цельнокорпусной электрогитарой. Классический гриф Telecaster выполнен из цельного куска клена и не имеет отдельной накладки. На гитаре установлены два звукоснимателя типа «сингл». Струны, как правило, имеют крепление сквозь корпус. На таких гитарах играли Kurt Cobain (Nirvana), George Harrison (The Beatles), Александр Васильев (Сплин) и другие известные музыканты [3].

Гитара формы Telecaster изображена на рисунке 3.

Рисунок 3 – Fender Telecaster

Рассмотренные варианты не отображают всего перечня существующих форм, но отражают принципиальную разницу между ними. В разрабатываемом приложении этот критерий является одним из определяющих факторов для выбора инструмента.

1.2 Обзор систем, позволяющих осуществить подбор электрогитары

Выделить аналоги разрабатываемой системы среди информационных систем, позволяющих провести подбор электрогитары на основе музыкальных предпочтений, не представляется возможным ввиду узкой направленности данной предметной области. Таким образом, ближайшими аналогами разрабатываемой рассматриваемой системы по цели и функциональности являются системы фильтрации и поиска сайтов крупных магазинов музыкальных инструментов, являющиеся основной платформой для выбора электрогитары.

Аналогами разрабатываемой системы были выбраны следующие магазины музыкальных инструментов:

– muztorg.ru;
– thomann.de;
– skifmusic.ru.

muztorg.ru – это крупнейший российский магазин музыкальных инструментов. На сайте магазина реализована система поиска и фильтрации, основными достоинствами которой являются:

– возможность фильтрации по ценовой категории;
– возможность фильтрации по бренду;
– возможность фильтрации по акциям магазина;
– возможность фильтрации по характеристикам инструментов;
– возможность сортировки по нескольким критериям;
– наличие поиска по популярным запросам других пользователей.

В результате фильтрации, пользователь имеет возможность просмотреть станицы инструментов, подходящих под его требования (рисунок 4).

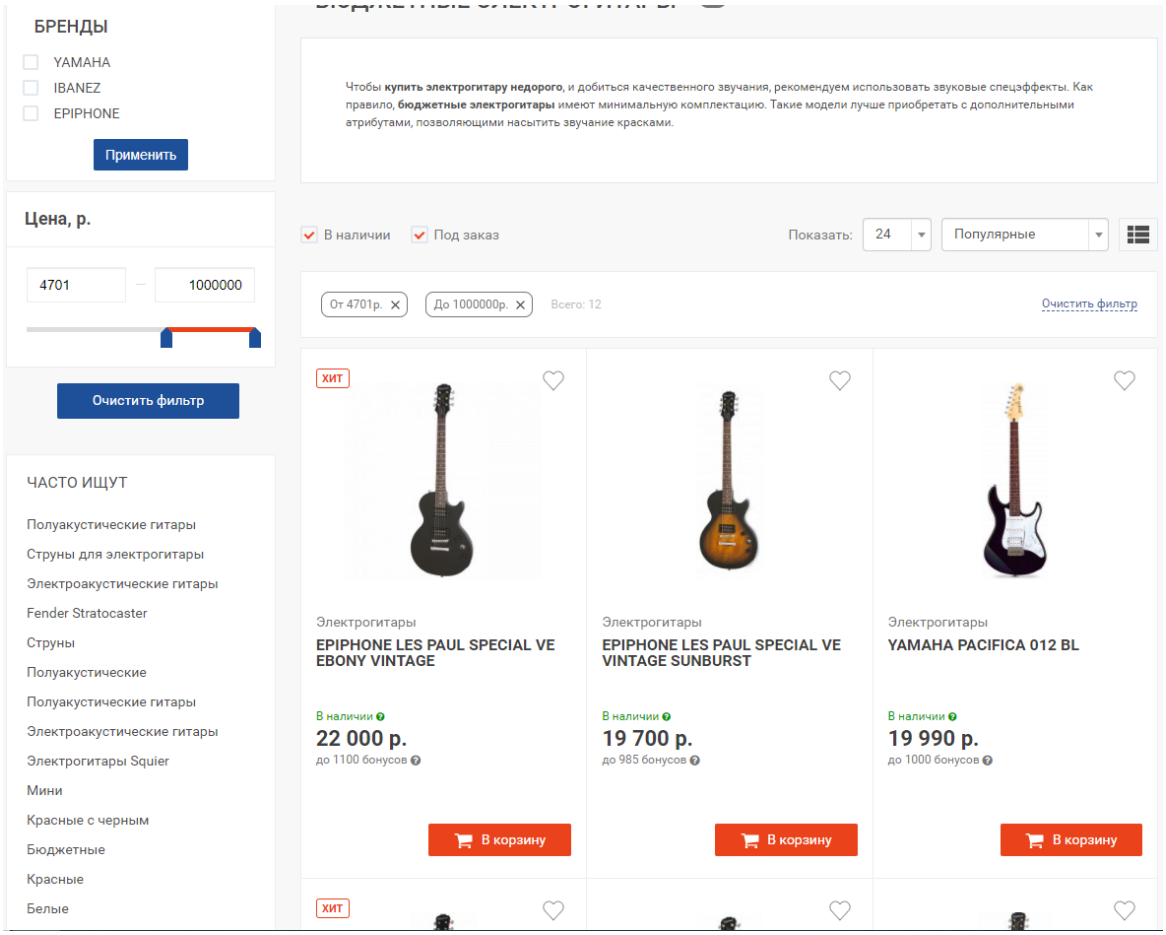


Рисунок 4 – Результат фильтрации на сайте магазина muztorg.ru

Среди недостатков системы можно выделить следующие:

– недостаточная оптимизация системы фильтрации и поиска: страница загружается каждый раз при изменении параметров фильтрации;

– описания многих инструментов не несут полезной информации;

– уцененные товары размещены на отдельном сайте;

– указанные характеристики не всегда совпадают с действительными.

thomann.de – это крупнейший европейский интернет-магазин музыкальных инструментов и оборудования. Среди положительных качеств системы фильтрации и поиска сайта магазина можно выделить следующие:

– дружелюбный интерфейс;

– рейтинговую систему страниц инструментов;

– возможность фильтрации по ценовой категории;

– возможность фильтрации по бренду;

– возможность фильтрации по характеристикам инструментов;

– возможность сортировки по нескольким критериям;

– большой ассортимент товаров, по сравнению с отечественными аналогами.

Среди недостатков системы можно выделить следующие:

– недостаточная оптимизация системы фильтрации и поиска: страница загружается каждый раз при изменении параметров фильтрации;

– разное форматирование описаний товаров;

– указанные характеристики не всегда совпадают с действительными.

В результате фильтрации, пользователь имеет возможность просмотреть станицы инструментов, подходящих под его требования (рисунок 5).

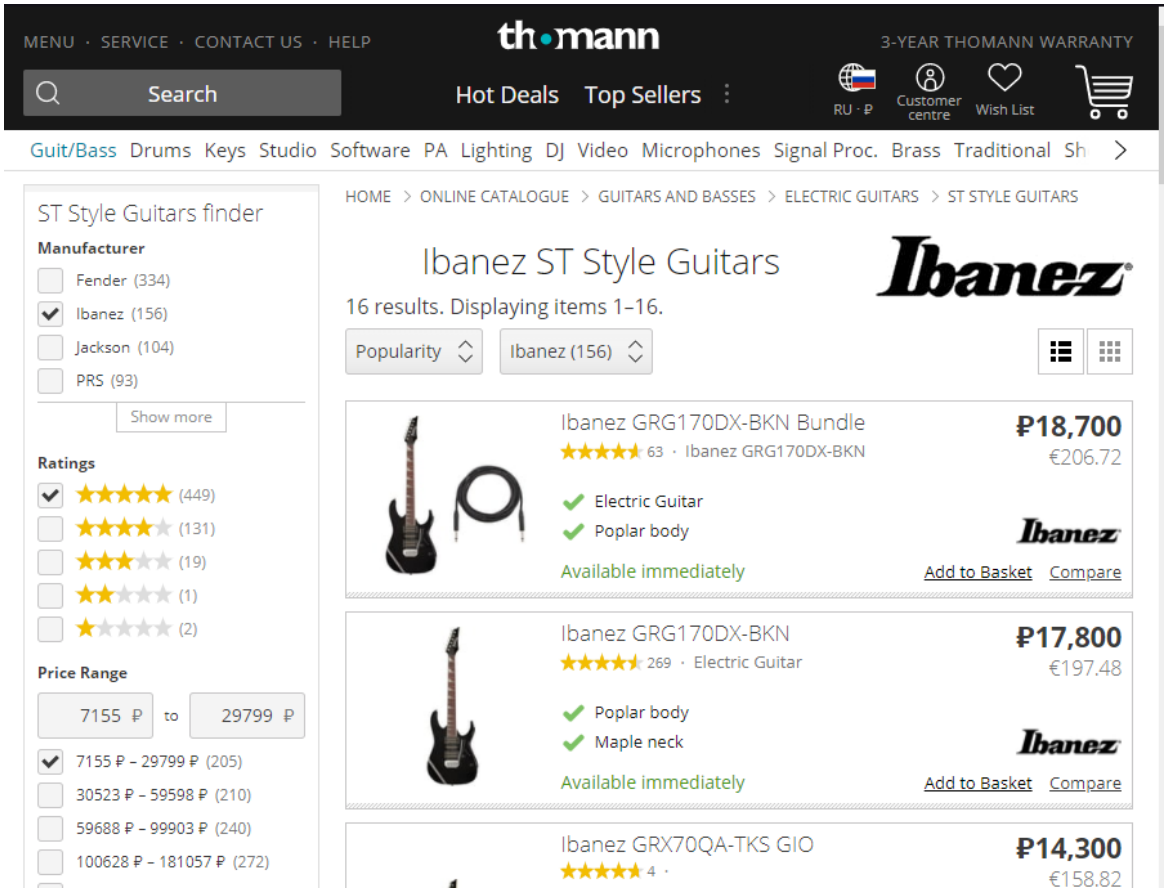


Рисунок 5 – Результат фильтрации на сайте магазина thomann.de

skifmusic.ru – это российский интернет магазин музыкальных инструментов. К достоинствам системы фильтрации и поиска относятся:

– возможность фильтрации по ценовой категории;
– большое количество инструментов в наличии;
– возможность фильтрации по бренду;
– возможность фильтрации по акциям магазина;
– возможность фильтрации по характеристикам инструментов;
– наличие б/у инструментов;
– возможность сортировки по нескольким критериям.

В результате фильтрации, пользователь имеет возможность просмотреть станицы инструментов, подходящих под его требования (рисунок 6).

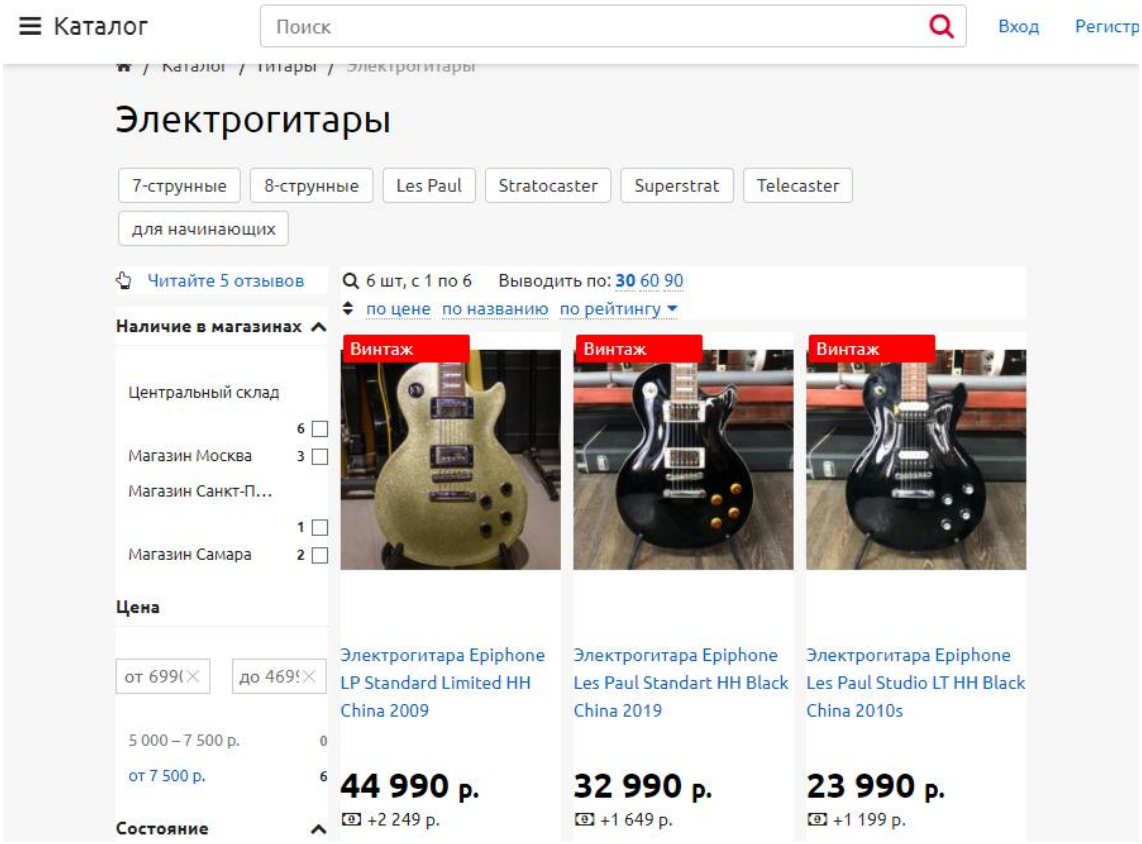


Рисунок 6 – Результат фильтрации на сайте skifmusic.ru

Среди недостатков системы можно выделить следующие:

– разное форматирование описаний товаров;

– указанные характеристики не всегда совпадают с действительными;

– устаревший интерфейс.

В результате сравнительного анализа был выявлен ряд недостатков и преимуществ каждого из продуктов-аналогов. Сайты магазинов музыкальных инструментов по большей части обладают продвинутым инструментарием для поиска и фильтрации товаров, позволяющим осуществлять подбор инструмента по параметрам, что является их главным преимуществом. Функция помощи по подбору инструмента отсутствует во всех рассмотренных системах, что является основным фактором, обуславливающим необходимость разработки новой системы.

В ходе исследования так же была выявлена необходимость сохранения результатов поиска или фильтрации, чего не позволяет функционал рассмотренных систем.

Главной необходимостью при подборе инструмента является условная объективность системы. Магазины музыкальных инструментов ориентированы на розничную продажу новых инструментов в наличии, что сводит их объективность по отношению к пользователю к минимуму.

Результаты сравнения продуктов-аналогов приведены в таблице 1.

Таблица 1 – Таблица сравнения аналогов

Критерии для сравнения	thomann.de	muztorg.ru	skifmusic.ru	Разрабатываемая система
Ориентированность	На инструменты, продаваемые на сайте	На инструменты, продаваемые на сайте	На инструменты, продаваемые на сайте	На модели электрогитар с высоким пользовательским рейтингом
Сохранение результатов фильтрации (подбора)	Нет	Нет	Нет	Есть
Наличие б/у инструментов	Только брак с уценкой	Только брак с уценкой	Есть	Есть
Наличие подбора, основанного на ассоциациях	Нет	Нет	Нет	Есть

Таким образом, актуальной является задача разработки системы, позволяющей подобрать музыкальный инструмент как в результате фильтрации и поиска, присущей рассматриваемым аналогам, с учетом выявленных преимуществ и недостатков, так и в результате подбора, основанного на ассоциациях, использующего элементы экспертных систем.

1.3 Анализ требований

1.3.1 Функциональные требования

При разработке информационной системы сервиса по подбору электрогитары возникает необходимость выделить функциональные требования. Такие требования напрямую влияют на качество системы, акцентируя внимание разработчика на критические с точки зрения работоспособности аспекты информационной системы [4].

В результате анализа предметной области и продуктов-аналогов были выделены следующие основные функциональные требования к системе:

– формирование набора предложений, основанных на музыкальных предпочтениях пользователя, выявленных в результате тестирования;

– просмотр пользователем каталога всех занесенных в систему предложений;

– регистрация пользователей в системе;

– авторизация пользователей в системе;

– хранение результатов последнего тестирования пользователя.

Для более наглядного отображения функциональных требований была построена диаграмма прецедентов, представленная на рисунке 7.

Рисунок 7 – Диаграмма прецедентов

Представленная диаграмма прецедентов в может быть детализирована в процессе проектирования информационной системы сервиса по подбору электрогитары. Детализация позволит выделить неочевидные задачи, влияющие на процесс разработки.

Рассмотренные функциональные требования позволят разработать информационную систему надлежащего качества.

1.3.2 Нефункциональные требования

Рекомендуемыми так же являются нефункциональные требования, оказывающие влияние на положительный опыт пользователя при взаимодействии с разрабатываемой информационной системой и общее быстродействие программных решений.

В соответствии с современными тенденциями разработки веб-приложений выделены следующие нефункциональные требования к системе:

– наличие современного и дружелюбного к пользователю интерфейса, разработанного с использованием языков HTML и CSS;

– адаптивность интерфейса к экранам мобильных устройств;

– использование современного фреймворка для разработки веб-приложений Spring.

Соблюдение рассмотренных нефункциональных требований позволит повысить качество взаимодействия пользователя с разрабатываемой системой, быстродействие программных решений и уменьшить сроки разработки.

2 Проектирование информационной системы

2.1 Выбор технологий проектирования

Для проектирования разрабатываемой информационной системы были выбраны диаграммы нотации UML, а также встроенные средства проектирования реляционных баз данных MySQL Workbench.

Для проектирования, а также для разработки базы данных была использована СУБД MySQL Workbench. MySQL Workbench – это СУБД, позволяющая создавать модели базы данных с помощью встроенных средств визуального проектирования, близким к нотации IDEF1X, генерировать скрипты для создания сущностей и атрибутов из существующей модели, а также эксплуатировать созданные базы данных, объединяя вышеупомянутый функционал в бесшовной оболочке [5].

Для разработки диаграмм нотации UML использовались средства сервиса diagrams.net.

2.2 Проектирование логической модели базы данных

Проектирование базы данных осуществлялось с использованием встроенных средств проектирования MySQL Workbench.

Существуют два уровня представления модели данных – логический и физический.

Логическая модель представляет собой абстрактный взгляд на данные и может иметь аналоги в реальном мире. Объектами логической модели являются сущности, атрибуты и отношения между сущностями [6]. Данная модель является универсальной и подходит для описания структуры базы данных любой СУБД.

Спроектированная логическая модель базы данных представлена на рисунке 8.

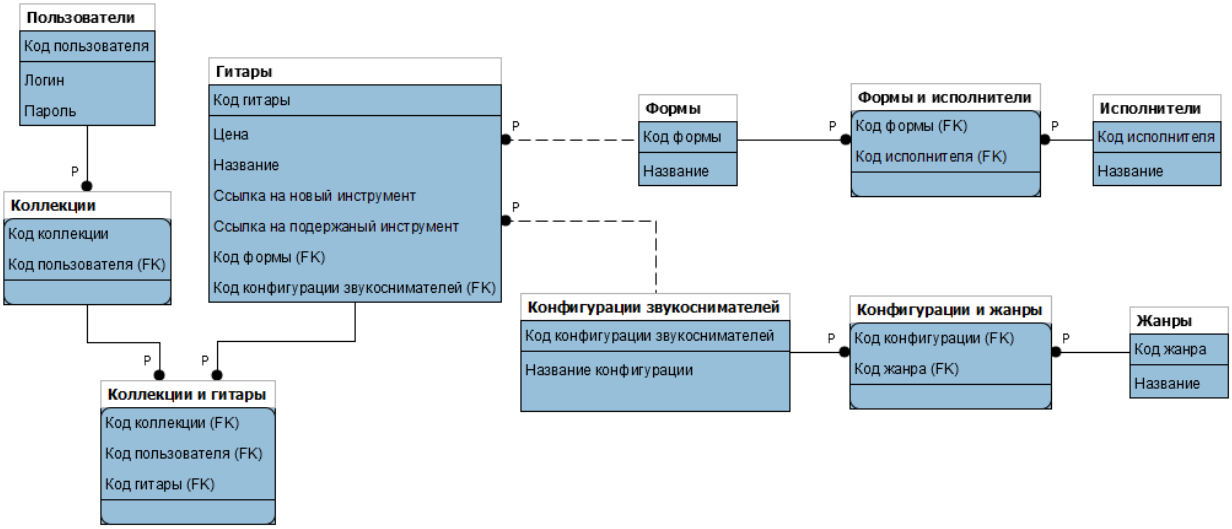


Рисунок 8 – Логическая модель данных

Таким образом была спроектирована логическая модель базы данных, описывающая структуру хранения данных в информационной системе сервиса по подбору электрогитары.

2.3 Проектирование программных средств

2.3.1 Диаграмма прецедентов

Для описания функциональных требований к системе была выбрана диаграмма прецедентов.

Общая функциональность системы в UML представляется с помощью диаграммы прецедентов. Она позволяет показать пользователей системы и ее функции [7]. Для цтого чтобы выделить неочевидные на первый взгляд варианты использования рассматриваемой информационной системы и упростить процесс дальнейшего проектирования и разработки расширим имеющуюся диаграмму прецедентов (смотри рисунок 3), добавив в нее включения и расширения имеющихся прецедентов.

Расширенная диаграмма прецедентов представлена на рисунке 9.

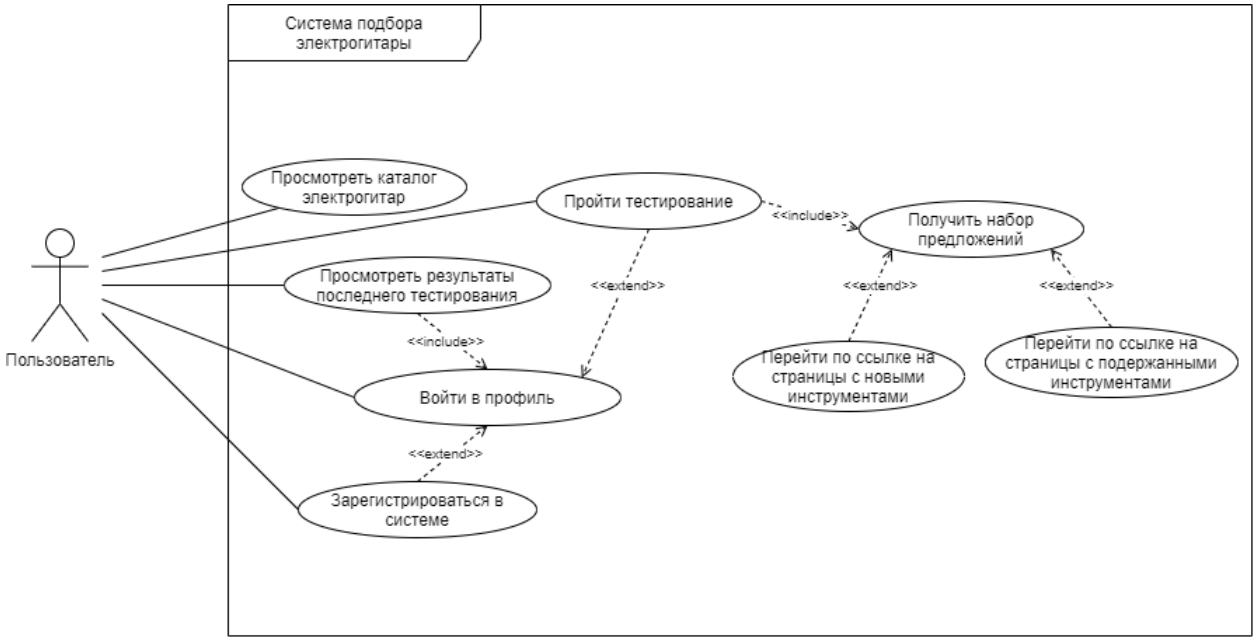


Рисунок 9 – Расширенная диаграмма прецедентов

Данная диаграмма отображает функционал приложения, доступный пользователю. Все следующие диаграммы, рассматриваемые в рамках разработки информационной системы, отражают те или иные аспекты прецедентов, указанных в диаграмме прецедентов.

Описание действующих лиц представлено в таблице 2.

Таблица 2 – Список действующих лиц диаграммы прецедентов

Название	Профиль, подготовка и навыки
1	2
Пользователь	Человек, не имеющий ни знаний в области устройства музыкальных инструментов, ни опыта игры на них. Желает приобрести инструмент, предположительно популярный в жанрах музыки, которые предпочитает прослушивать. В приоритете инструменты бюджетного ценового сегмента, достаточно качественные.

Краткое описание вариантов использования представлено в таблице 3.

Таблица 3 – Описание вариантов использования

Действующее лицо	Цель	Краткое описание
Пользователь	Зарегистрироваться в системе	Пользователь регистрируется в системе для получения возможности сохранять результаты опроса и подписываться на его обновления.
	Пройти тестирование	Для того, чтобы получить выборку предложений, пользователь должен ответить на ряд вопросов, в зависимости от выбранного режима. Формат вопросов – тест с вариантами ответа.
	Получить набор предложений	Пользователь получает выборку предложений, подходящих под параметры, полученные в ходе тестирования.
	Просмотреть каталог электрогитар в системе	Пользователь имеет возможность получить выборку всех возможных предложений, предусмотренных в системе.
	Просмотреть результаты последнего тестирования	Пользователь получает результаты последнего пройденного тестирования.
	Войти в профиль	Пользователь может войти в свой профиль в системе, если он был зарегистрирован в системе ранее.
	Перейти по ссылке на страницы с новыми инструментами	Пользовать может пройти по ссылке на страницу в интернет-магазине любой из представленных в выборке электрогитар.
	Перейти по ссылке на страницы с подержанными инструментами	Пользовать может пройти по сформированной ссылке на поисковую страницу на сайте avito.ru с указанными в URL-параметрах критериями поиска.

Таким образом была разработана диаграмма прецедентов, выступающая основой для дальнейшего проектирования и разработки.

2.3.2 Диаграмма классов

Диаграммы классов отображают концепцию и принципы работы объектно-ориентированных приложений. В них представляются все значимые классы приложения и связи между ними [8]. Диаграмма классов может содержать поля и методы для каждого класса.

В соответствии с моделью предметной области были спроектированы классы информационной системы, представленные на рисунке 10.

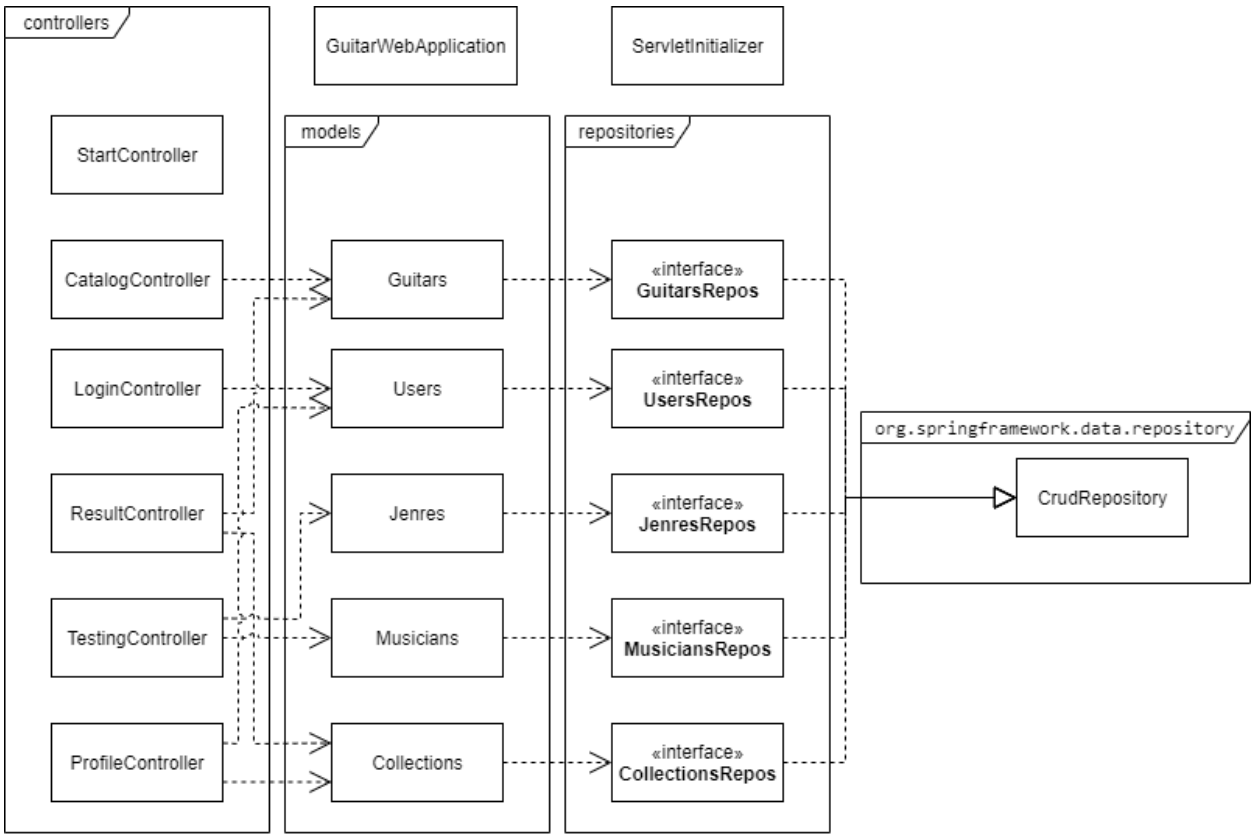


Рисунок 10 – Диаграмма классов

Изображенные на диаграмме (смотри рисунок 10) классы объединены в следующие пакеты:

– пакет «controllers», содержащий классы, обеспечивающие работу с HTML-страницами сервиса;

– пакет «models», содержащий классы, отображающие сущности базы данных в контексте веб-приложения;

– пакет «repositories», содержащий классы, хранящие методы взаимодействия с базой данных;

– пакет «org.springframework.data.repository», являющийся импортируемым пакетом библиотеки фреймворка Spring.

Разделение обосновано принципиальными функциональными различиями и используется для удобства разработки и дальнейшего обращения к методам и полям классов.

Краткое описание диаграммы классов представлено в таблице 4.

Таблица 4 – Краткое описание диаграммы классов

Название класса	Описание
GuitarWebApplication	Класс для инициализации приложения из WAR-архива
ServletInitalizer	Класс, принимающий все входящие запросы
CrudRepository	Стандартный класс фреймворка Spring, содержащий методы работы с базой данных.
Guitars	Класс, отображающий сущность базы данных «Гитары»
Users	Класс, отображающий сущность базы данных «Пользователи»
Jenres	Класс, отображающий сущность базы данных «Жанры»
Musicians	Класс, отображающий сущность базы данных «Исполнители»
Collections	Класс, отображающий сущность базы данных «Коллекции»
GuitarsRepos	Интерфейс, реализующий методы работы с сущностью «Гитары»
UsersRepos	Интерфейс, реализующий методы работы с сущностью «Пользователи»
JenresRepos	Интерфейс, реализующий методы работы с сущностью «Жанры»
MusiciansRepos	Интерфейс, реализующий методы работы с сущностью «Исполнители»
CollectionsRepos	Интерфейс, реализующий методы работы с сущностью «Коллекции»
CatalogController	Класс-контроллер, необходимый для взаимодействия со страницей «Каталог»
LoginController	Класс-контроллер, необходимый для взаимодействия со страницей «Авторизация и регистрация»
ResultController	Класс-контроллер, необходимый для взаимодействия со страницей «Результат»
StartController	Класс-контроллер, необходимый для взаимодействия со стартовой страницы
TestingController	Класс-контроллер, необходимый для взаимодействия со страницами тестирования
ProfileController	Класс-контроллер, необходимый для взаимодействия со страницей «Профиль пользователя»

Представленные классы отражают принципы работы фреймворка Spring и позволяют разрабатывать гибкие и современные веб-приложения, обеспечивающие оперативное взаимодействие с базой данных.

2.3.3 Диаграмма состояний

Каждый объект системы может характеризоваться состояниями, в которых может находится, и переходами между этими состояниями. Состояние объекта – это ситуация в жизненном цикле объекта, в которой объект соответствует определенным условиям, ожидает некоторого события или выполняет определенную деятельность [9].

Диаграмма состояний позволяет наглядно продемонстрировать все возможные состояния объекта во время жизненного цикла приложения. Такой подход к проектированию существенно снижает риски возникновения ситуаций, в которых могут нерационально использоваться ресурсы вычислительной среды информационной системы.

Диаграмма состояний объекта класса TestingController представлена на рисунке 11.

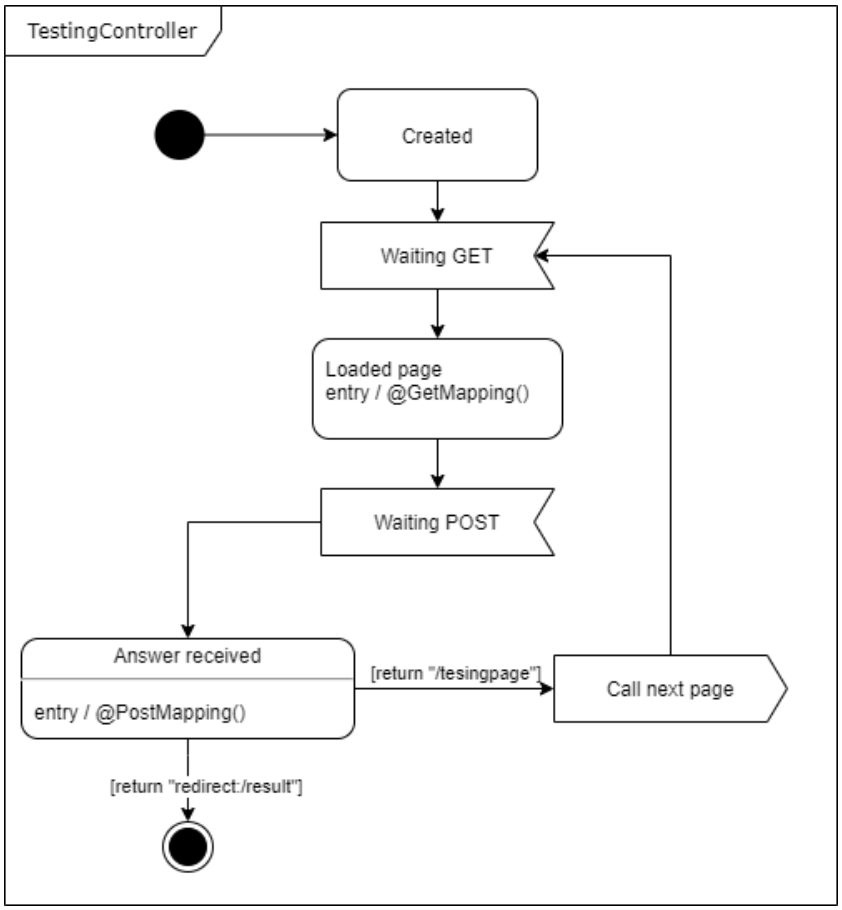


Рисунок 11 – Диаграмма состояний объекта класса TestingController

Краткое описание состояний объекта класс TestingController приведено в таблице 5.

Таблица 5 – Краткое описание состояний объекта класса TestingController

Состояние	Описание
Created	Состояние, при котором объект создается конструктором при запуске приложения.
Waiting GET	Состояние, в котором объект ожидает запроса GET, чтобы отобразить необходимую HTML-страницу тестирования.
Loaded page	Состояние, в котором объект загружает необходимую HTML-страницу тестирования. При переходе в это состояние выполняется функция с аннотацией @GetMapping.
Waiting POST	Состояние, в котором объект ожидает запроса POST, чтобы получить параметры ответа пользователя на вопрос тестирования.
Answer received	Состояние, в котором объект получает параметры ответа пользователя. При этом, при переходе в это состояние выполняется функция с аннотацией @PostMapping.
Call next page	Состояние, в котором объект вызывает следующую страницу тестирования.

Таким образом были выделены и смоделированы все возможные состояния объекта класса TestingController.

2.3.4 Диаграмма деятельности

Диаграмма деятельности – UML-диаграмма, на которой показано разбиение некоторой деятельности на ее составные части. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчиненных элементов – вложенных видов деятельности и отдельных действий, соединенных между собой потоками, которые идут от выходов одного узла к входам другого. Диаграммы деятельности используются при моделировании бизнес-процессов, технологических процессов, последовательных и параллельных вычислений [10].

В контексте рассматриваемой системы диаграмма подробно отображает вариант использования «Пройти тестирование».

Диаграмма деятельности в нотации UML представлена на рисунке 12.

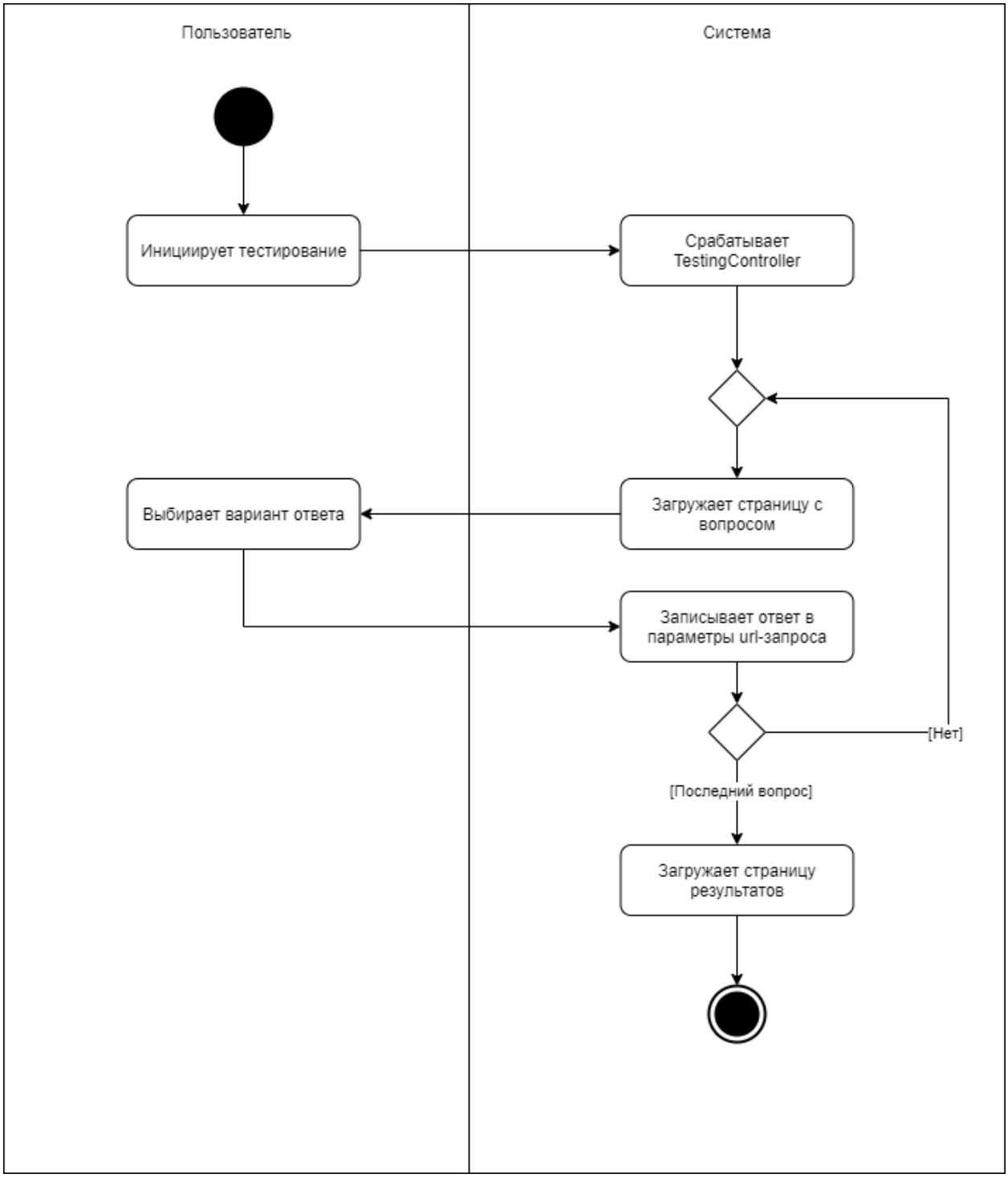


Рисунок 12 – Диаграмма деятельности прецедента «Пройти тестирование»

В представленной диаграмме деятельности описано взаимодействие пользователя и информационной системы сервиса по подбору электрогитары.

Пользователь инициирует тестирование в клиентской части информационной системы. Затем серверная часть приложения загружает вопросы последовательно в формате HTML-страниц.

2.3.5 Диаграмма последовательности

Для четкого понимания поведения объектов системы прецедента необходимо разработать диаграмму последовательности.

Диаграммы последовательности позволяют моделировать взаимодействие между объектами в едином сценарии использования. На диаграммах последовательности изображаются объекты, временные линии, планки активности и стрелки сообщений [11]. Такой подход к проектированию позволяет достигнуть максимально точной модели процесса.

Диаграмма последовательности прецедента «Зарегистрироваться в системе» представлена на рисунке 13.

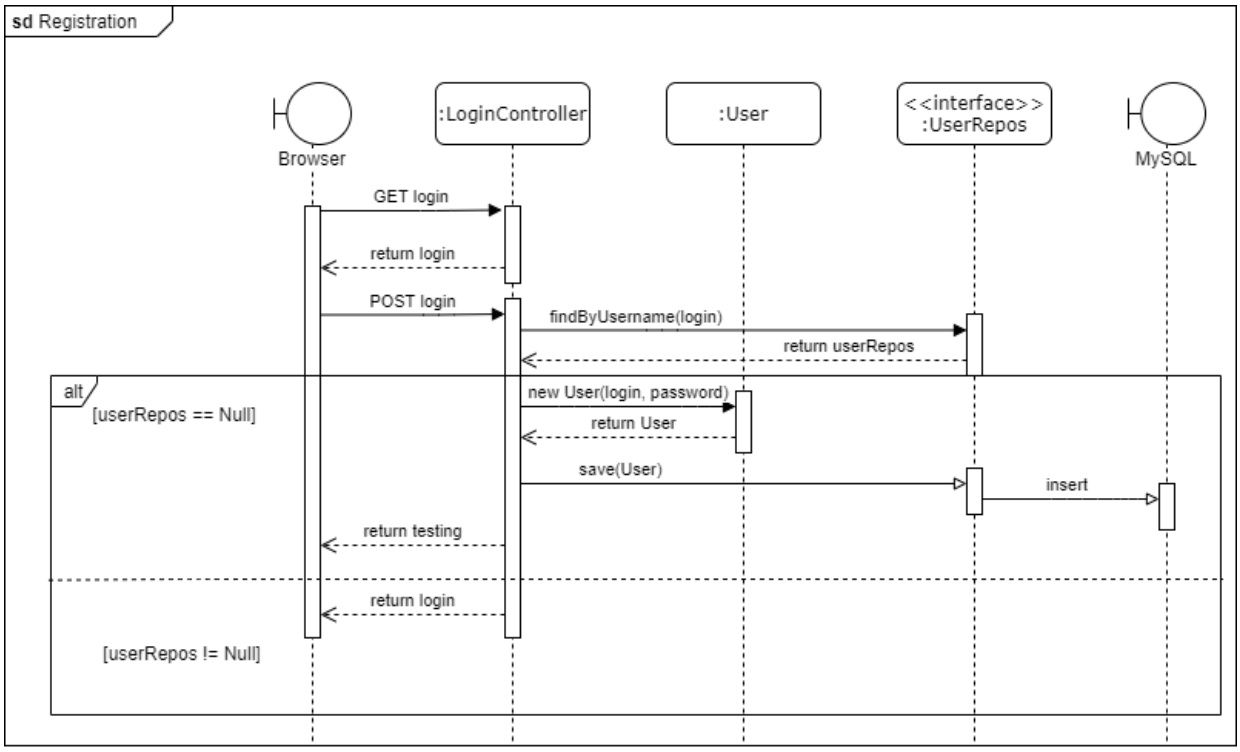


Рисунок 13 – Диаграмма последовательности прецедента «Зарегистрироваться в системе»

Построенная диаграмма отображает процесс регистрации пользователя в информационной системе сервиса по подбору электрогитары. Это

обусловлено тем, что процесс регистрации связан с данными пользователей и нуждается в детальном рассмотрении.

2.3.6 Диаграмма кооперации

Для наглядности процессов, содержащих в себе сложные взаимодействия большого количества объектов системы, используются диаграммы кооперации.

Диаграммы кооперации позволяют кратко и емко отобразить то, в какой последовательности объекты системы обмениваются сообщениями, содержащими данные или являющимися вызовом функций [12].

Диаграмма кооперации прецедента «Получить набор предложений» представлена на рисунке 14.

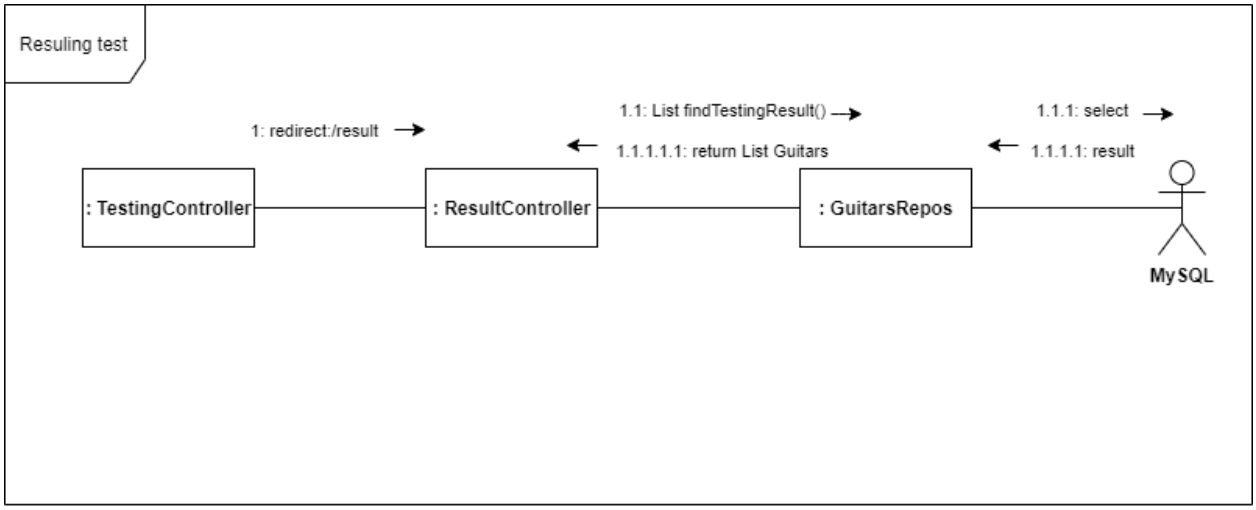


Рисунок 14 – Диаграмма кооперации прецедента «Получить набор предложений»

Представленная диаграмма кооперации (смотри рисунок 14) моделирует процесс формирования результирующей выборки электрогитар и вывод ее на страницу «Результаты». Краткое описание объектов прецедента «Получить набор предложений» представлено в таблице 6.

Таблица 6 – Краткое описание объектов процесса

Название объекта	Описание	Входные потоки	Выходные потоки
: TestingController	Объект класса TestingController,	Нет	1: redirect:/result

	осуществляющий взаимодействие с HTML-страницами тестирования		
: ResultController	Объект класса ResultController, осуществляющий взаимодействие с HTML-страницей результатов тестирования	1: redirect:/result 1.1.1.1.1: return List Guitars	1.1: List findTestingResult()
: GuitarsRepos	Интерфейс, реализующий методы взаимодействия объектов класса Guitars и соответствующей сущности базы данных	1.1: List findTestingResult() 1.1.1.1: result	1.1.1: select
MySQL	База данных MySQL	1.1.1: select	1.1.1.1: result

Таким образом была разработана диаграмма кооперации прецедента «Получить набор предложений».

2.4 Проектирование интерфейса

Важным этапом проектирования информационной системы является проектирование пользовательского интерфейса. Чтобы реализовать качественную графическую оболочку разрабатываемой системы необходимо смоделировать основные страницы будущего веб-сайта [13]. Основными страницами информационной системы сервиса по подбору электрогитары являются страницы, отображающие основные функциональные требования к разрабатываемой системе.

Знакомство пользователя с системой начинается со стартовой страницы веб-приложения.

Макет стартовой страницы веб-приложения представлен на рисунке 15.

Рисунок 15 – Макет стартовой страницы веб-приложения

Стартовая страница содержит заголовок с приветствием пользователя и кнопки «Авторизация» и «Пропустить», являющиеся переходами на страницы «Авторизация и регистрация» и «Тестирование» соответственно.

Макет страницы «Авторизация и регистрация» приведена на рисунке 16.

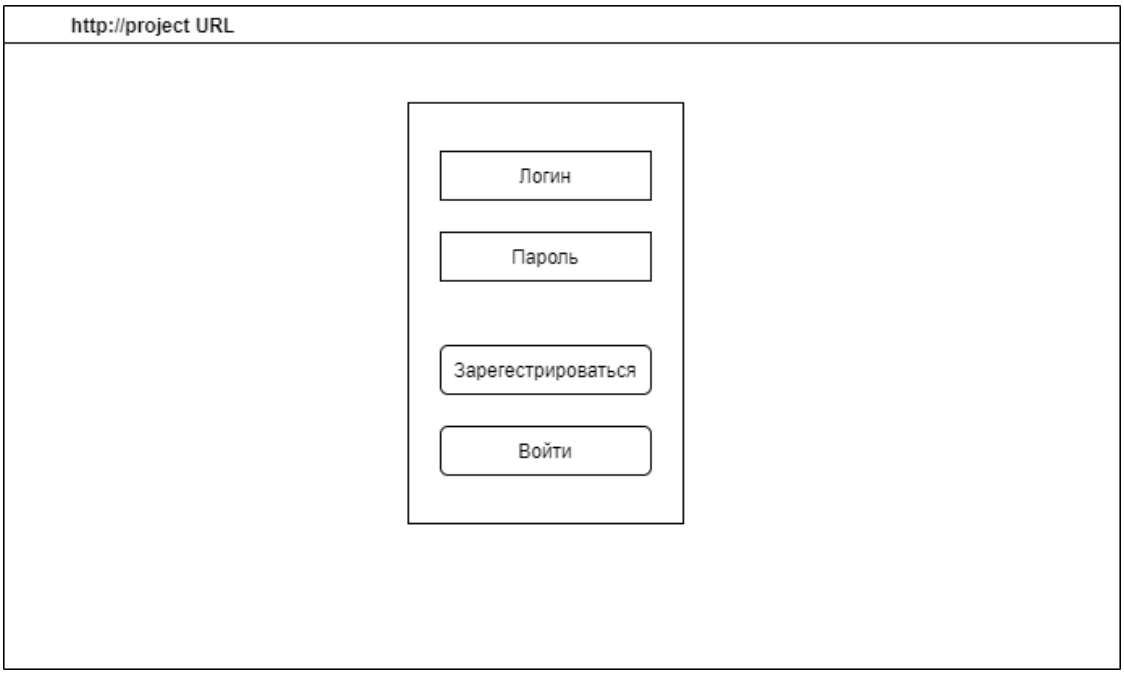


Рисунок 16 – Макет страницы «Авторизация и регистрация»

Страница «Авторизация и регистрация» содержит форму для заполнения логина и пароля пользователя, а также кнопки «Зарегистрироваться» и «Войти». При успешной регистрации или авторизации пользователь будет перенаправлен на страницу «Профиль».

Макет страницы «Профиль» представлена на рисунке 17.

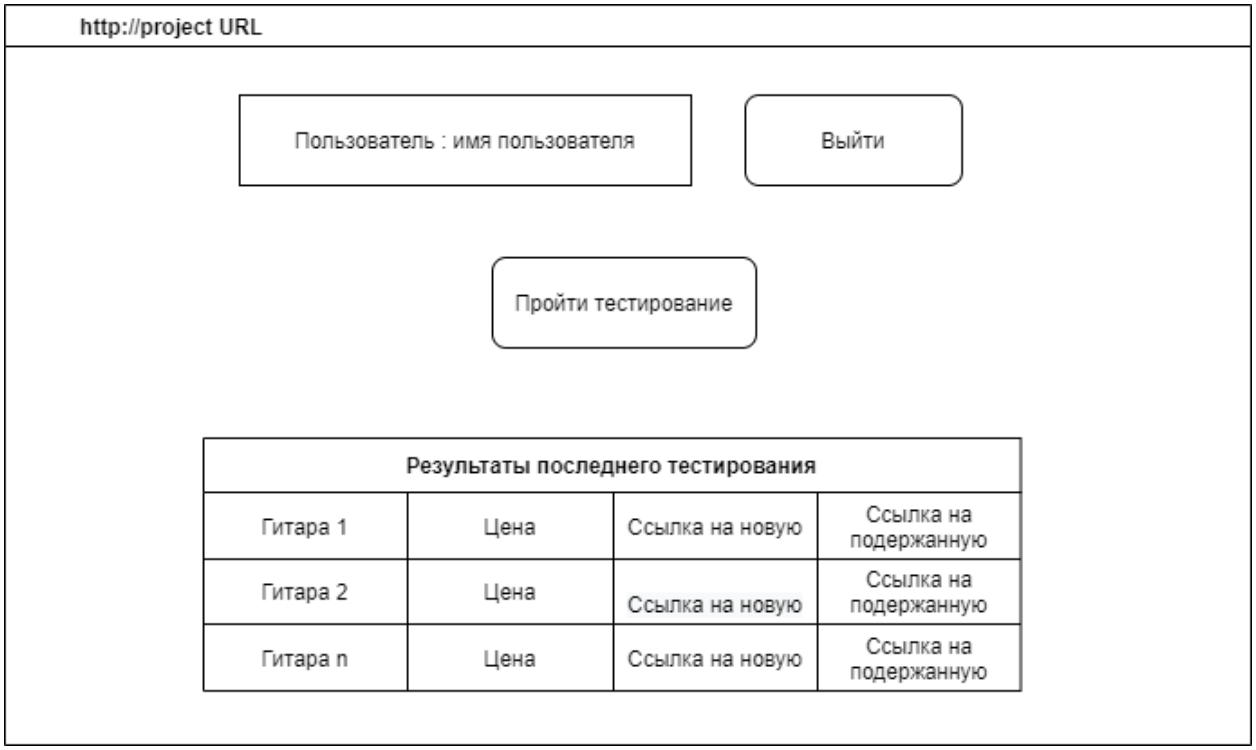


Рисунок 17 – Макет страницы «Профиль»

Страница «Профиль» содержит следующие элементы:

– Форму для отображения логина пользователя;

– кнопку «Выйти»;

– кнопку «Пройти тестирование»;

– таблицу «Результаты последнего тестирования» если тестирование было осуществлено.

Кнопка «Выйти» позволяет прекратить использование текущего профиля пользователя в текущем сеансе. Кнопка пройти тестирование перенаправляет пользователя на первую страницу тестирования. Таблица «Результаты последнего тестирования» содержат выборку гитар, полученных в результате тестирования, если такое было осуществлено.

Макет страницы «Тестирование» представлен на рисунке 18.

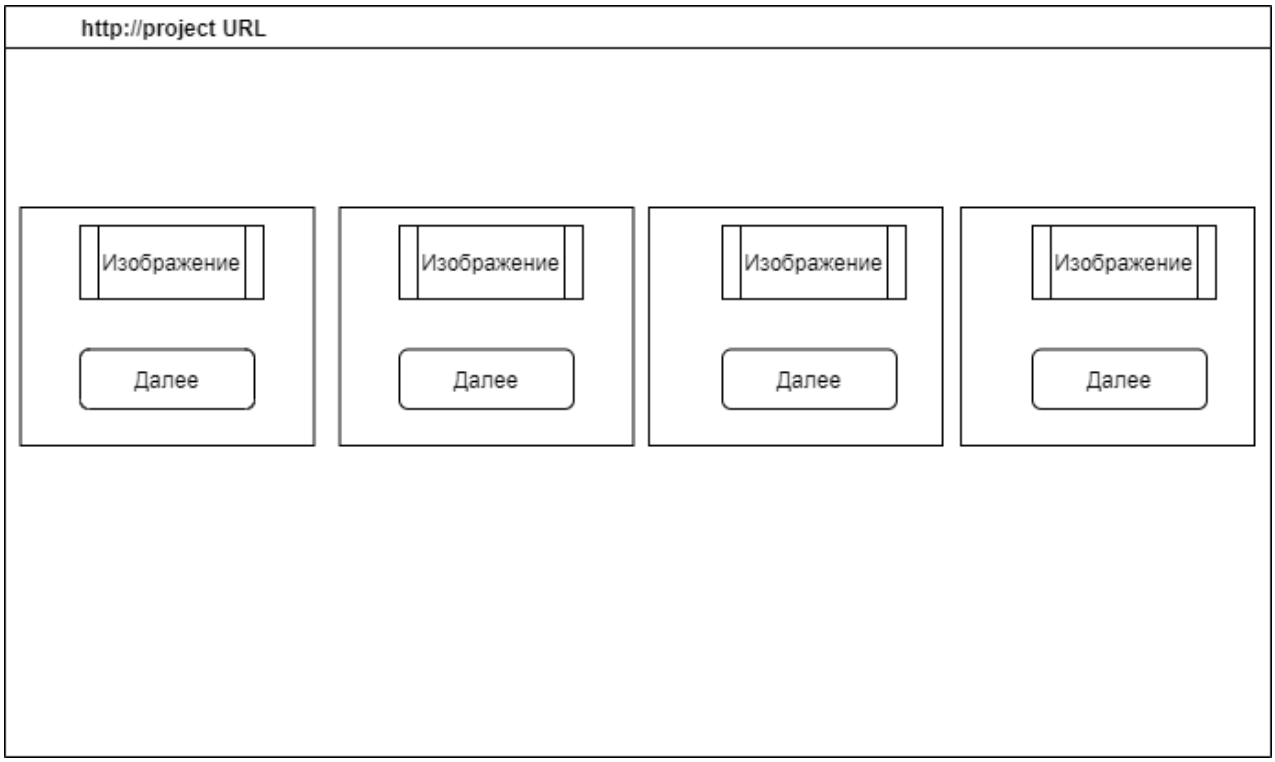


Рисунок 18 – Макет страницы «Тестирование»

Макет страницы «Тестирование» отражает принцип построения любой страницы, использующихся в ходе тестирования пользователя. Типовая страница тестирования содержит несколько блоков с вопросами и иллюстрациями, а также кнопками «Далее», продвигающих пользователя по страницам тестирования. Каждый ответ передается в URL-параметрах вплоть до конца тестирования. Когда пользователь ответит на последний вопрос тестирования, он будет перенаправлен на страницу результатов, где будет сформирована выборка электрогитар, оформленных в таблицу.

Страница «Результаты» содержит заголовок, сообщающий о конце тестирования и таблицу, содержащую выборку электрогитар, соответствующих выявленным в результате тестирования пользователя критериям.

Макет страницы «Результаты» представлен на рисунке 19.

Рисунок 19 – Макет страницы «Результаты»

Для формирования выборки электрогитар, система собирает все ответы пользователя в SQL-запрос SELECT, используя методы работы с базой данных, определенные в интерфейсе Guitars. После формирования выборки, гитары выводятся поочередно в таблицу «Результаты» со следующими атрибутами:

– название;
– цена;
– ссылка на новую гитару;
– ссылка на подержанную гитару.

Так же пользователь системы имеет возможность просмотреть каталог всех гитар, занесенных в систему.

Страница «Каталог» содержит таблицу, формируемую SQL-запросом SELECT, в которую выводится каждая гитара, содержащаяся в базе данных.

Макет страницы «Каталог» представлен на рисунке 20.

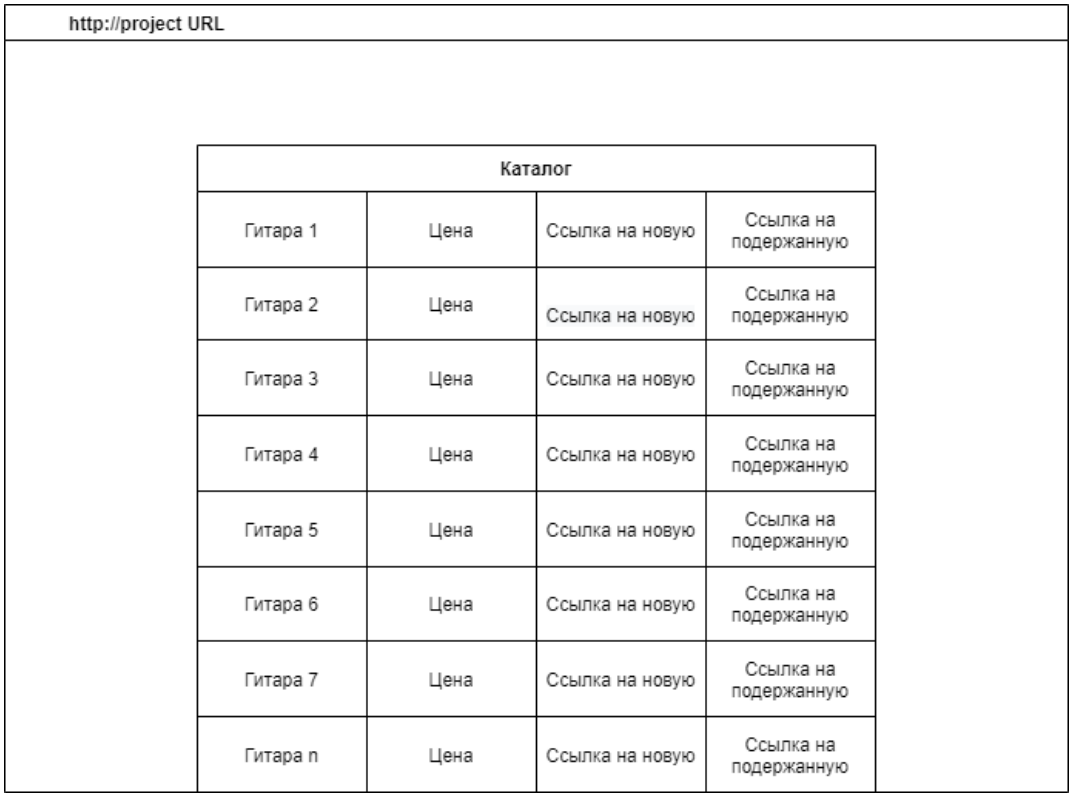


Рисунок 20 – Макет страницы «Каталог»

Карта основных страниц сайта представлена на рисунке 21.

Рисунок 21 – Карта основных страниц сайта

Таким образом были спроектированы основные функциональный аспекты пользовательского интерфейса путем макетирования HTML-страниц.

2.5 Планирование и оценка бюджета

В реализации проекта немаловажна оценка времени и стоимости реализации. Необходимо разделить проект на перечень задач, позволяющих структурировать и распределить временную и финансовую нагрузку на команду предполагаемых разработчиков информационной системы.

Перечень задач проекта представлен в таблице 7.

Таблица 7 – Перечень задач

№	Название задачи	Длительность	Предшественники
1	Начало проекта	52 дня	
2	Анализ требований для проекта разработки информационной системы сервиса по подбору электрогитары	7 дней	
3	Обзор систем-аналогов	4 дня	
4	Составление списка функциональных возможностей приложения	4 дня	
5	Выбор программного средства для реализации приложения	3 дня	3;4
6	Проектирование для проекта разработки информационной системы сервиса	7 дней	2
7	Проектирование интерфейса приложения	5 дней	
8	Проектирование структур данных	5 дней	
9	Выбор архитектуры программного обеспечения	2 дня	8
10	Реализация	23 дня	6
11	Программирование классов приложения	19 дней	
12	Создание базы данных	6 дней	
13	Создание интерфейса приложения	4 дней	11;12
14	Отладка	15 дней	10
15	Тестирование работы приложения	8 дней	
16	Исправление ошибок	7 дней	15

Из перечня задач (смотри таблицу 7) видно, что для реализации проекта необходимо осуществить шестнадцать задач. Чтобы оценить стоимость

разработки необходимо выделить предполагаемые ресурсы, затрачиваемые в ходе выполнения проекта.

Список предполагаемых затрачиваемых ресурсов проекта информационной системы сервиса по подбору электрогитары представлен в таблице 8.

Таблица 8 – Ресурсы проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист 1	Трудовой	110 Р\ч	200 Р\ч
Программист 2	Трудовой	95 Р\ч	190 Р\ч
Интернет	Затраты	2300 Р	
Компьютер	Материальные	60 000 Р	
Сервер	Затраты	1000 Р	

Среди предполагаемых затрат на разработку были выделены следующие типы ресурсов:

– трудовые;
– материальные;
– затраты.

Прежде всего, для разработки информационной системы нужны программисты и компьютеры. В качестве стандартной и сверхурочной ставок были взяты усредненные значения. Также немаловажной статьей затрат будут являться абонентская плата за услуги интернет и аренда сервера для размещения ресурсов веб-приложения.

Следующим шагом необходимо распределить ресурсы между задачами проекта.

Распределение ресурсов между задачами проекта информационной системы сервиса по подбору электрогитары продемонстрировано в таблице 9.

Таблица 9 – Распределение ресурсов проекта

№	Название задачи	Ресурс	Единицы
1	Начало проекта	Интернет Компьютер Сервер	2700 2 шт 2000
2	Анализ требований для проекта разработки информационной системы сервиса по подбору электрогитары		
3	Обзор систем-аналогов	Программист1	100 %
4	Составление списка функциональных возможностей приложения	Программист2	100 %
5	Выбор программного средства для реализации приложения	Программист1 Программист2	100 % 100 %
6	Проектирование для проекта разработки информационной системы сервиса		
7	Проектирование интерфейса приложения	Программист2	100 %
8	Проектирование структур данных	Программист1	100 %
9	Выбор архитектуры программного обеспечения	Программист2 Программист1	100 % 100 %
10	Реализация		
11 12	Программирование классов приложения Создание базы данных	Программист1 Программист1	100 % 100 %
13	Создание интерфейса приложения	Программист1 Программист2	100 % 100 %
14	Отладка		
15	Тестирование работы приложения	Программист1 Программист2	100 % 100 %
16	Исправление ошибок	Программист1 Программист2	100 % 100 %

Следующим шагом необходимо внести выделенные задачи и этапы в проект Microsoft Project. Это позволит провести автоматическое распределение задач и подсчет ресурсов.

Планирование задач в Microsoft Project представлено на рисунке 22.

		Режим задачи	Название задачи	Длительнс	Начало	Окончани	Предшественн	Названия ресурсов
1	✓		**Начало проекта**	**52 дней**	**Чт 01.04.21**	**Чт 10.06.21**		**Интернет[2 700,00 ₽];Компьютер[2];Сервер[2 000,00 ₽]**
2	✓		**Анализ требований для проекта разработки информационной системы сервиса по подбору электрогитары**	**7 дней**	**Чт 01.04.21**	**Чт 08.04.21**		
3	✓		Обзор систем-аналог	4 дней	Чт 01.04.21	Пн 05.04.21		Программист1
4	✓		Составление сг	4 дней	Чт 01.04.21	Пн 05.04.21		Программист2
5	✓		Выбор програм	3 дней	Вт 06.04.21	Чт 08.04.21	3;4	Программист2;Программист1
6	✓		**Проектирование для проекта разработки информационной системы сервиса**	**7 дней**	**Пт 09.04.21**	**Пн 19.04.21**	**2**	
7	✓		Проектировани	5 дней	Пт 09.04.21	Чт 15.04.21		Программист1
8	✓		Проектировани структур	5 дней	Пт 09.04.21	Чт 15.04.21		Программист2
9	✓		Выбор архитек	2 дней	Пт 16.04.21	Пн 19.04.21	8	Программист2;Программист1
10	✓		**Реализация**	**23 дней**	**Вт 20.04.21**	**Чт 20.05.21**	**6**	
11	✓		Программиров	19 дней	Вт 20.04.21	Пт 14.05.21		Программист1
12	✓		Создание базы	6 дней	Вт 20.04.21	Вт 27.04.21		Программист2
13	✓		Создание инте	4 дней	Пн 17.05.21	Чт 20.05.21	11;12	Программист1;Программист2
14	✓		**Отладка**	**15 дней**	**Пт 21.05.21**	**Чт 10.06.21**	**10**	
15	✓		Тестирование работы приложения	8 дней	Пт 21.05.21	Вт 01.06.21		Программист1;Программист2
16	✓		Исправление ошибок	7 дней	Ср 02.06.21	Чт 10.06.21	15	Программист1;Программист2

Рисунок 22 – Планирование задач в Microsoft Project

Во время планирования задач Microsoft Project отмечает все неточности, допущенные проектировщиком, и предлагает пути их решения, сокращая таким образом сроки разработки проекта на этапе проектирования, а также исключая ошибки в составлении плана работ и распределении ресурсов.

Microsoft Project по задачам, введенным в системе автоматически выстроил диаграмму Ганта, представленную на рисунке 23.

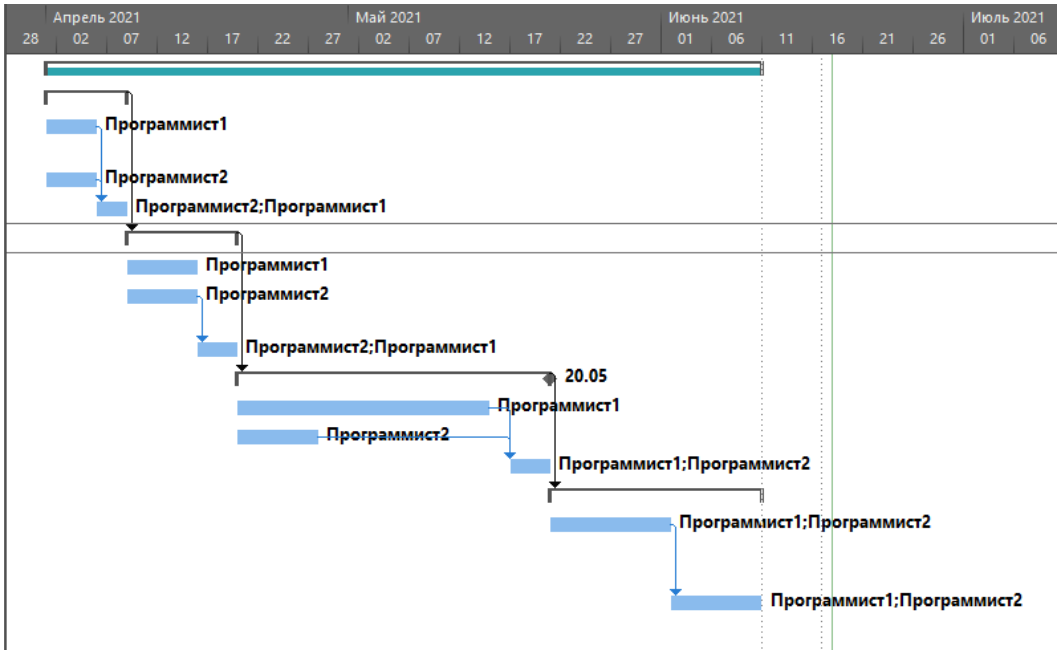


Рисунок 23 – Диаграмма Ганта для реализуемого проекта

Также функционал Microsoft Project позволяет отметить критический путь проекта, выделив критические задачи.

Диаграмма Ганта с отмеченным критическим путем представлена на рисунке 24.

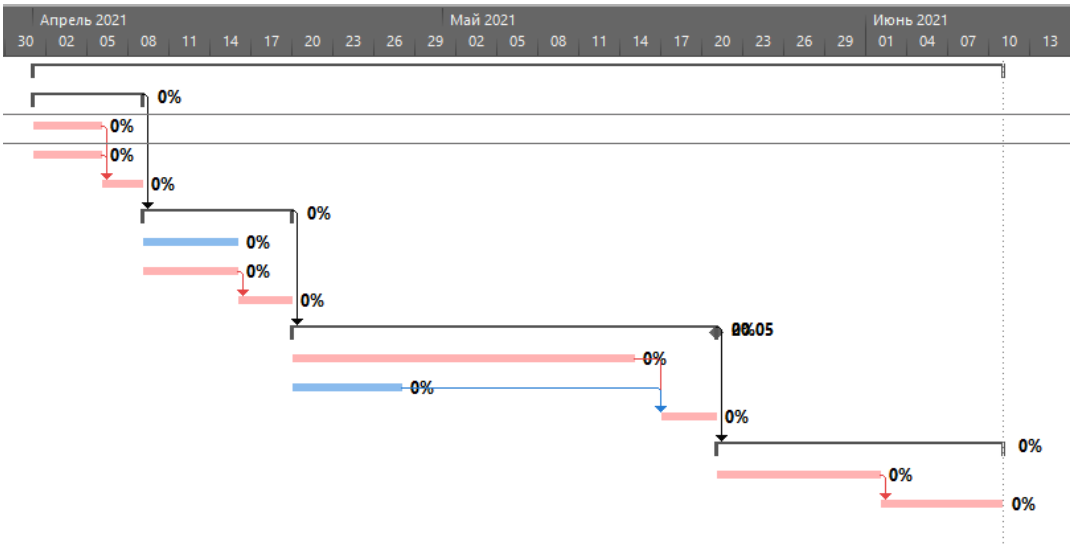


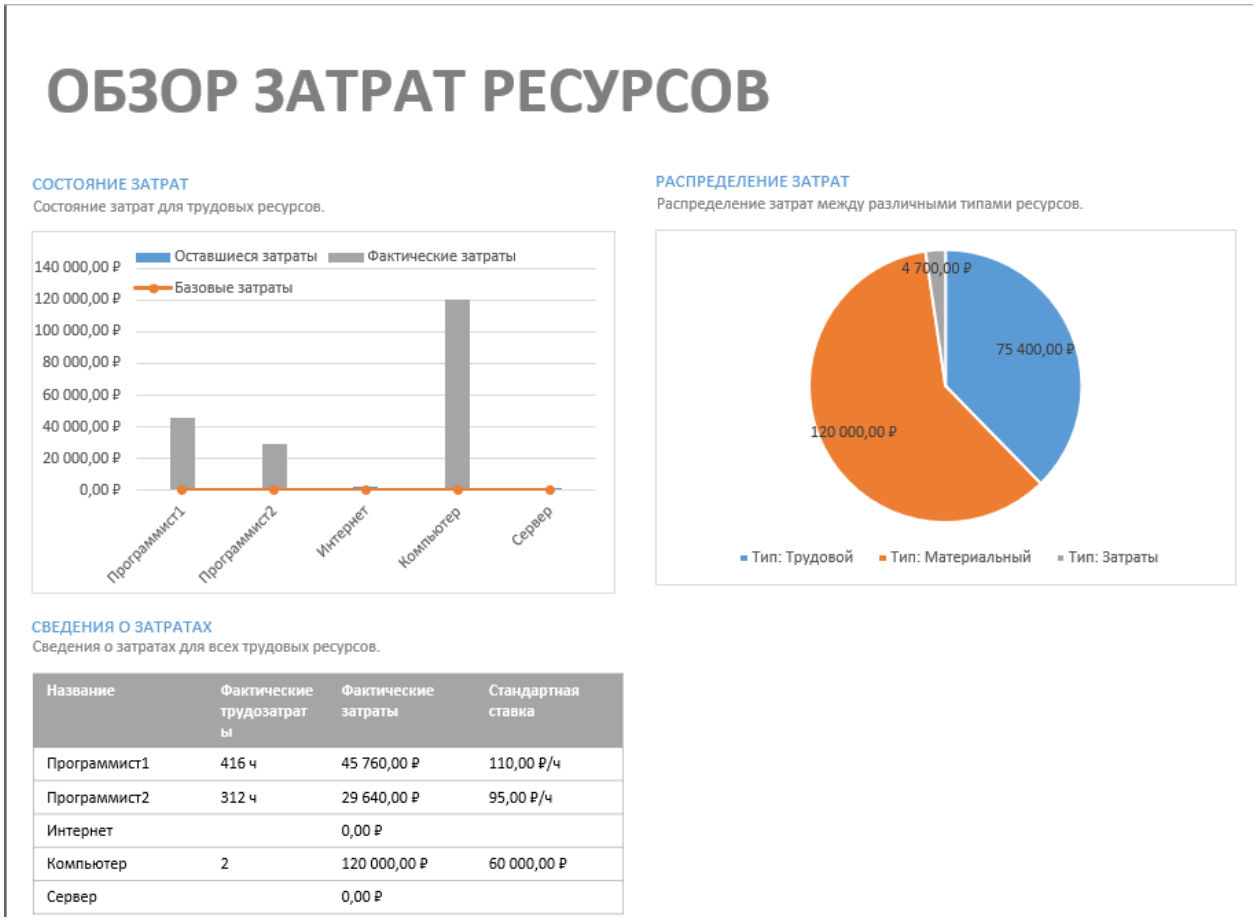
Рисунок 24 – Диаграмма Ганта проекта с отмеченным критическим путем

На рисунке (смотри рисунок 23) изображен сетевой график, построенный программой Microsoft Project Professional. Красным цветом на

графике выделен критический путь, содержащий критические работы номер 3, 4, 5, 6, 8, 9, 11, 13, 15 и 16.

Microsoft Project поддерживает формирование отчетов по различным аспектам проекта. Наиболее применимым к проекту разработки информационной системы сервиса по подбору электрогитары является отчет по ресурсам «Обзор затрат ресурсов», позволяющий наглядно и быстро проиллюстрировать затраченные на проект средства, сравнить распределение затрат между различными типами ресурсов и отобразить текущее состояние затрат в случае, если проект не завершен.

Для расчета затрат проекта был сформирован отчет «Обзор затрат ресурсов» в Microsoft Project, представленный на рисунке 25.



СВЕДЕНИЯ О ЗАТРАТАХ

Сведения о затратах для всех трудовых ресурсов.

Название	Фактические трудозатраты	Фактические затраты	Стандартная ставка
Программист1	416 ч	45 760,00 ₽	110,00 ₽/ч
Программист2	312 ч	29 640,00 ₽	95,00 ₽/ч
Интернет		0,00 ₽	
Компьютер	2	120 000,00 ₽	60 000,00 ₽
Сервер		0,00 ₽	

Рисунок 25 – Обзор затрат ресурсов

Таким образом срок разработки составляет 52 дня, общие затраты по ресурсам составляют 200100 рублей.

3 Реализация информационной системы

3.1 Физическая модель базы данных

Физическая модель строится на основе логической модели данных и учитывает особенности целевой СУБД [14]. В качестве СУБД был выбран продукт MySQL. В качестве средства проектирования была выбрана среда разработки реляционных баз данных MySQL Workbench, предоставляющая возможность спроектировать как логическую, так и физическую модель базы данных, а также экспортировать SQL-скрипт, позволяющий как создать таблицы с необходимыми свойствами на основе спроектированной модели, так и сгенерировать стартовое заполнение таблиц.

Спроектированная физическая модель базы данных представлена на рисунке 26.

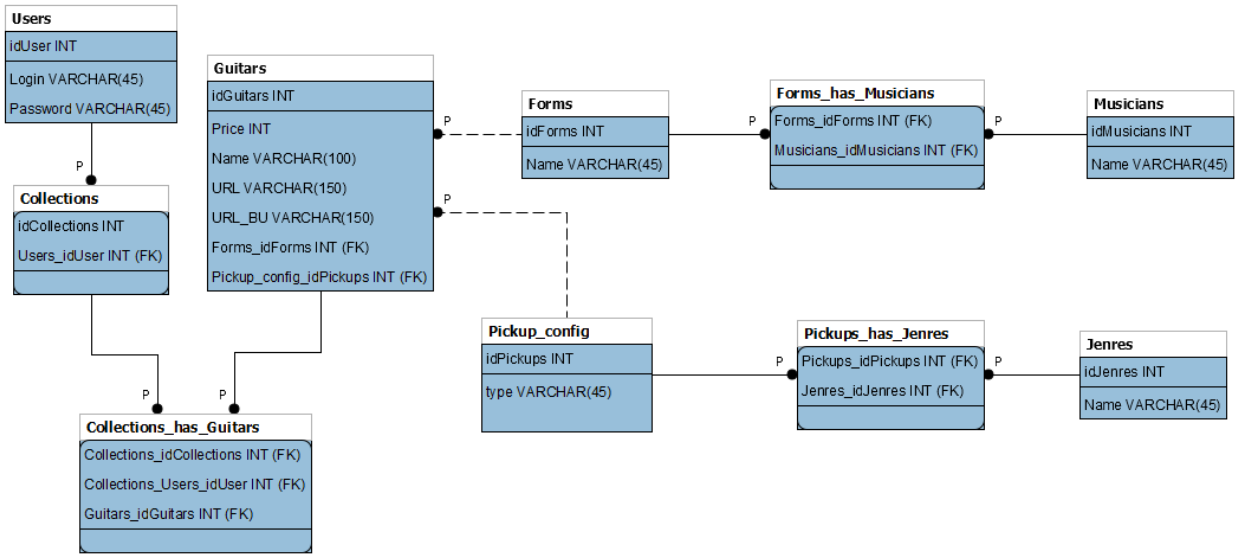


Рисунок 26 – Физическая модель базы данных

Данная модель базы данных отображает сущности, связи между ними, а также их атрибуты, типы атрибутов и внешние ключи.

Описание сущностей и их атрибутов физической модели базы данных информационной системы сервиса по подбору электрогитары представлены в таблице 10.

Таблица 10 – Описание сущностей и их атрибутов физической модели

Сущность	Атрибуты	Роль атрибута	Описание атрибута
Users	idUser	PK	Идентификатор пользователя в базе данных
	Login		Логин пользователя в системе
	Password		Пароль пользователя в системе
Collections	idCollections	PK	Идентификатор коллекции инструментов
	Users_idUser	PK FK	Идентификатор пользователя
Guitars_has_Collections	Guitars_idGuitars	PK FK	Идентификатор гитары
	Guitars_Forms_idForms	FK	Идентификатор формы гитары
	Guitars_Pickup_config_idPickups	FK	Идентификатор конфигурации звукоснимателей
	Collections_idCollections	PK FK	Идентификатор коллекции
	Collections_Users_idUser	FK	Идентификатор пользователя
Guitars	idGuitars	PK	Идентификатор гитары
	Price		Средняя стоимость гитары
	Name		Наименование
	URL		Ссылка на новый инструмент
	URL_BU		Ссылка на б/у инструмент
	Forms_idForms	FK	Идентификатор формы
	Pickup_config_idPickups	FK	Идентификатор конфигурации звукоснимателей
Forms	idForms	PK	Идентификатор формы
	Name		Название формы
Forms_has_Musicians	Forms_idForms	PK FK	Идентификатор формы
	Musicians_idMusicians	PK FK	Идентификатор исполнителя
Musicians	idMusicians	PK	Идентификатор исполнителя
	Name		Название исполнителя
Pickup_config	idPickups	PK	Идентификатор конфигурации звукоснимателей
	type		Аббревиатура конфигурации
Pickups_has_Jenres	Pickups_idPickups	PK	Идентификатор конфигурации звукоснимателей
	Jenres_idJenres	PK FK	Идентификатор жанра
Jenres	idJenres	PK	Идентификатор жанра
	Name		Название жанра

На основе построенной модели базы данных может быть сгенерирован скрипт создания базы данных. Кроме того, MySQL Workbench позволяет генерировать скрипты стартового заполнения созданных таблиц на этапе моделирования.

3.2 Реализация программных средств

3.2.1 Реализация классов

Программная реализация классов информационной системы сервиса по подбору электрогитары построена по принципам программирования в фреймворке Spring.

Классы делятся на следующие типы:

– классы типа «Контроллер», осуществляющие взаимодействие с HTML-страницами;

– классы типа «Модель», отражающие сущности базы данных;

– интерфейсы типа «Репозиторий», реализующие базовые методы работы с базами данных.

Примером реализации классов типа «Контроллер» выступает класс LoginController, приведенный в листинге 1.

Листинг 1 – Класс LoginController

```
@Controller
public class LoginController {
    @Autowired
    private UsersRepos userRepos;

    @GetMapping("/login.html")
    public String login(Model model) {
        Iterable<Users> users = userRepos.findAll();
        model.addAttribute("users", users);
        return "login";
    }

    @PostMapping("/login.html")
    public String loginData(@RequestParam String type,
@RequestParam String login, @RequestParam String password, Model
model) {
        Users user = new Users(login, password);
        if (type == "authorization") {
            if (userRepos.findByUsername(login) != null) {
                model.addAttribute("currentUser", user);
                return "redirect:/profile";
            } else if ((password == "") || (login == "")) {
                model.addAttribute("message", "Некоторые из
полей пусты!");
                return "login";
            } else {
```

```
                model.addAttribute("message", "Пользователь не
найден!");
                return "login";
            }
        } else {
            if (userRepos.findByUsername(login) != null) {
                userRepos.save(user);
                model.addAttribute("currentUser", user);
                return "redirect:/profile";
            } else if ((password == "") || (login == "")) {
                model.addAttribute("message", "Некоторые из
полей пусты!");
                return "login";
            } else {
                model.addAttribute("message", "Пользователь уже
зарегистрирован в системе!");
                return "login";
            }
        }
    }
}
```

Данный класс используется для реагирования системы на URL-запросы GET и POST по URL-адресу /login.html, а также для осуществления процесса авторизации и регистрации пользователей в информационной системе сервиса.

Примером реализации класса типа «Модель» может служить класс Guitars, приведенный в листинге 2.

Листинг 2 – Класс Guitars

```
@Entity
public class Guitars {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long idGuitars, idForm, idPickupConfiguration;
    private String name, url, urlBU;

    public Long getIdGuitars() {
        return idGuitars;
    }

    public Long getIdForm() {
        return idForm;
    }

    public Long getIdPickupConfiguration() {
        return idPickupConfiguration;
```

```
    }

    public String getName() {
        return name;
    }

    public String getUrl() {
        return url;
    }

    public String getUrlBU() {
        return urlBU;
    }
}
```

Данный класс используется для отображения сущности Guitars базы данных в приложении.

3.2.2 Диаграмма компонентов

Диаграмма компонентов используется для отображения взаимосвязи между частями системы [15].

Диаграмма компонентов информационной системы сервиса по подбору электрогитары представлена на рисунке 27.

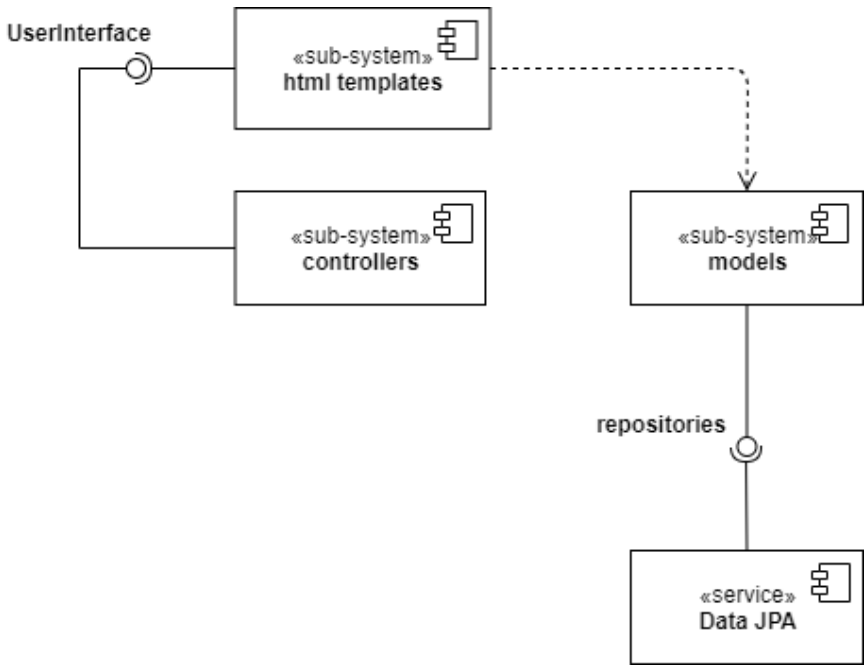


Рисунок 27 – Диаграмма компонентов

Представленная диаграмма компонентов описывает взаимодействие подсистем внутри разрабатываемой информационной системы.

3.2.3 Диаграмма развертывания

Диаграмма развертывания – это диаграмма, на которой представлены узлы выполнения программных компонентов реального времени, а также процессов и объектов.

Диаграмма развертывания применяется для представления общей конфигурации и топологии распределенной программной системы и содержит изображение размещения компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений – маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы [16]. Диаграмма развертывания представлена на рисунке 28.

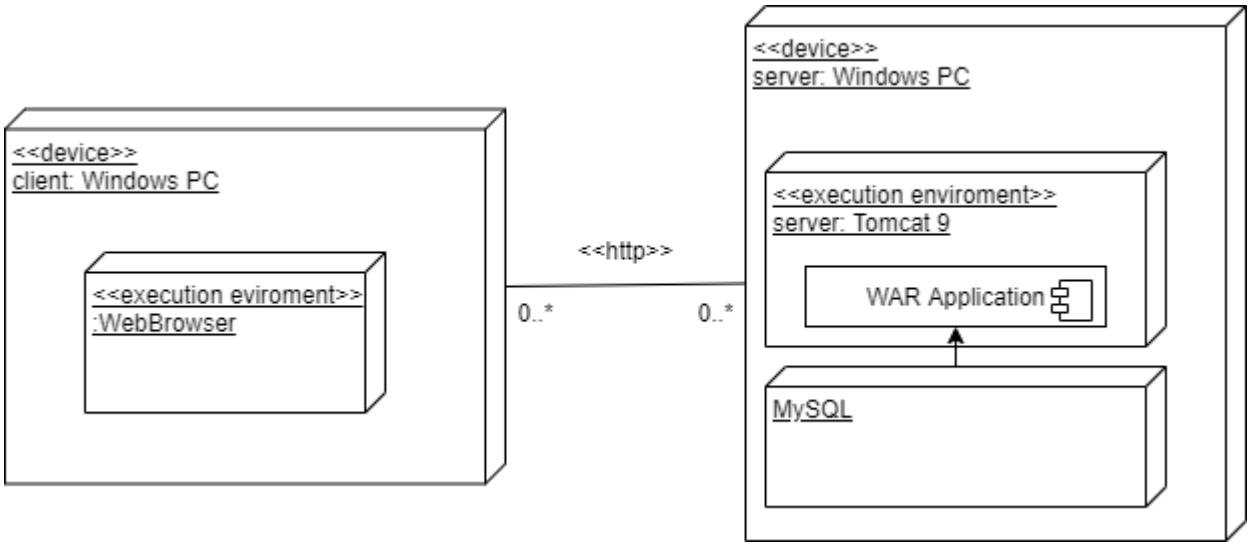


Рисунок 28 – Диаграмма развертывания

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих только на этапе ее исполнения.

Согласно разработанной диаграмме развертывания система будет состоять из WAR-архива с приложением, веб-страницами и их оформлением и базы данных MySQL. Развертывание приложения на вэб-ресурсе происходит при помощи веб-сервера Tomcat 9. Доступ пользователя к ресурсу будет осуществляться через браузер.

3.3 Разработка пользовательского интерфейса

На основе расширенной диаграммы вариантов использования и разработанного макета пользовательского интерфейса был реализован простой и интуитивно понятный графический пользовательский интерфейс на языке гипертекстовой разметки HTML с использованием таблиц стилей CSS. Такой подход к реализации позволяет разработать дружелюбный к пользователю интерфейс.

Каждая страница сайта содержит титульную верхнюю часть дизайн-макета, представленную на рисунке 29.

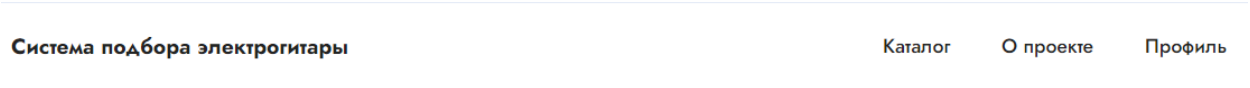


Рисунок 29 – Титульная часть дизайн-макета

Элемент содержит ссылки на следующие страницы:

– страница «Каталог» с каталогом всех имеющихся в системе предложений с электрогитарами;

– страница «О проекте» с информацией о проекте;

– страница «Профиль», содержащую информацию о текущем пользователе сервиса.

Кнопка «Профиль» содержит в себе ссылку на страницу с профилем авторизованного пользователя, в которой содержится форма просмотра результатов последнего опроса этого пользователя.

Страница «Профиль пользователя» содержит форму отображения пользователя, в которой выводится логин текущего пользователя, и кнопку «Выход» позволяющий выйти из текущей учетной записи в гостевую. Также в профиле пользователя есть возможность инициировать тестирование.

Страница «Профиль пользователя» представлена на рисунке 30.

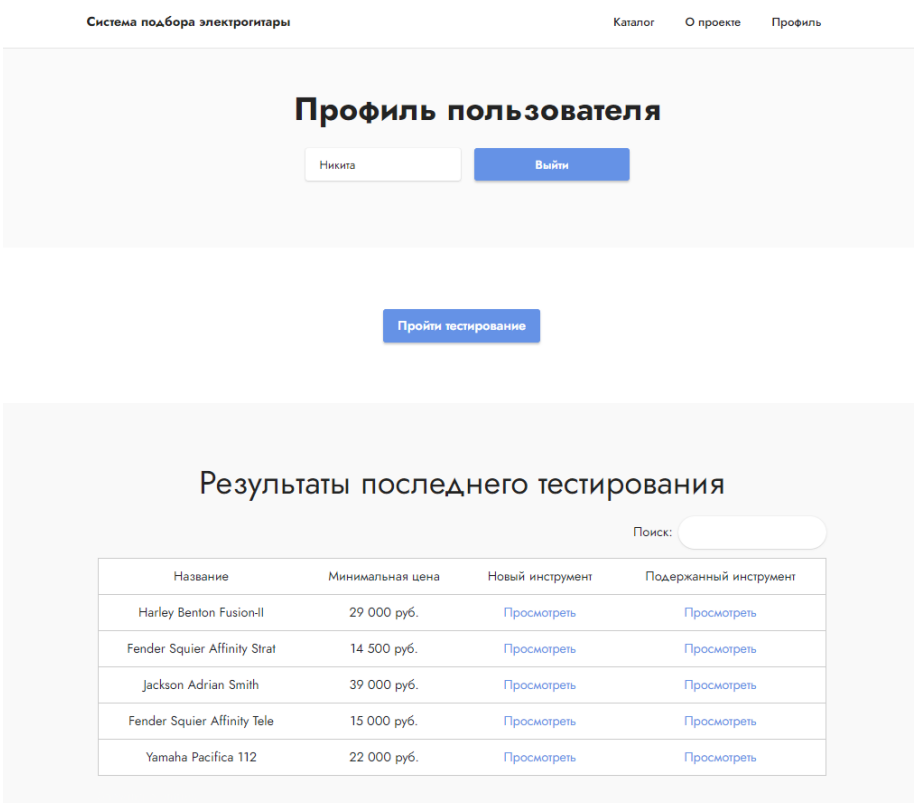


Рисунок 30 – Профиль пользователя

При посещении веб-ресурса сервиса по подбору электрогитары на экран выводится приветственная страница, представленная на рисунке 31.

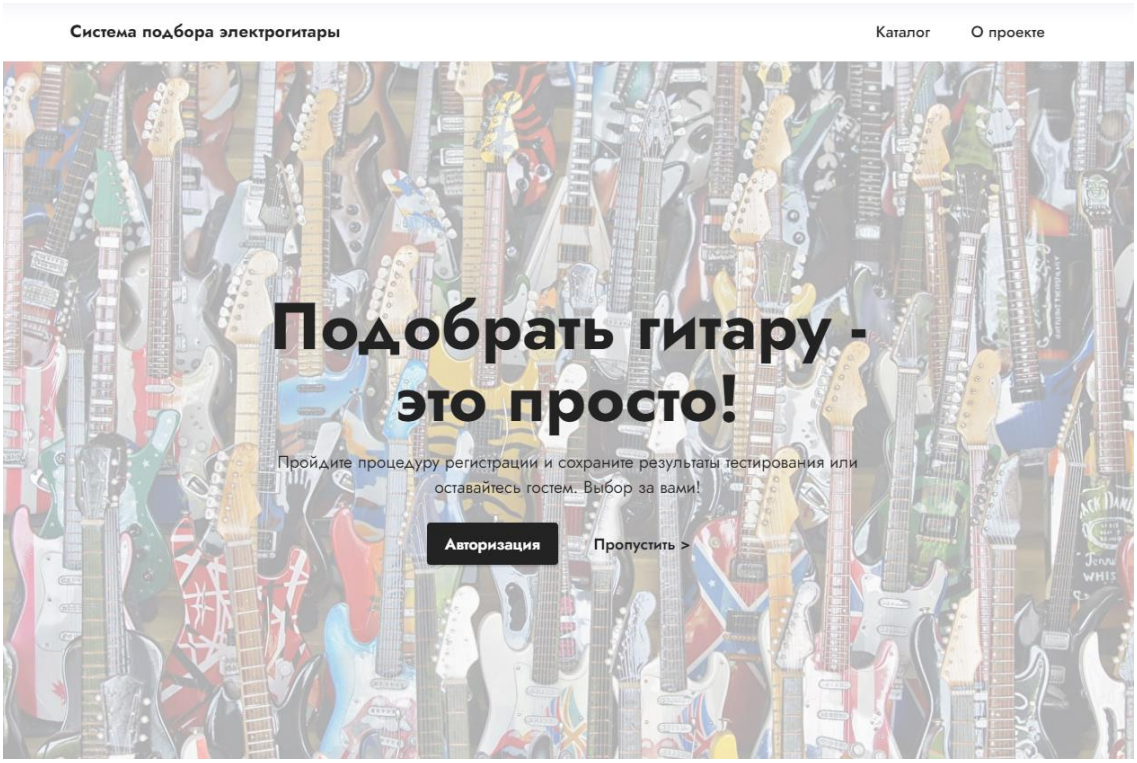


Рисунок 31 – Приветственная страница сервиса

Приветственная страница содержит следующие функциональные элементы:

– кнопка «Авторизация»;

– кнопка «Пропустить».

В случае нажатия кнопки «Пропустить» пользователь будет перенаправлен на страницу тестирования. При этом пользователь будет находится в гостевом профиле и не будет иметь возможности просматривать результаты последнего тестирования.

В случае нажатия кнопки «Авторизация» пользователь будет перенаправлен на страницу с формой авторизации и регистрации пользователя, представленной на рисунке 32.

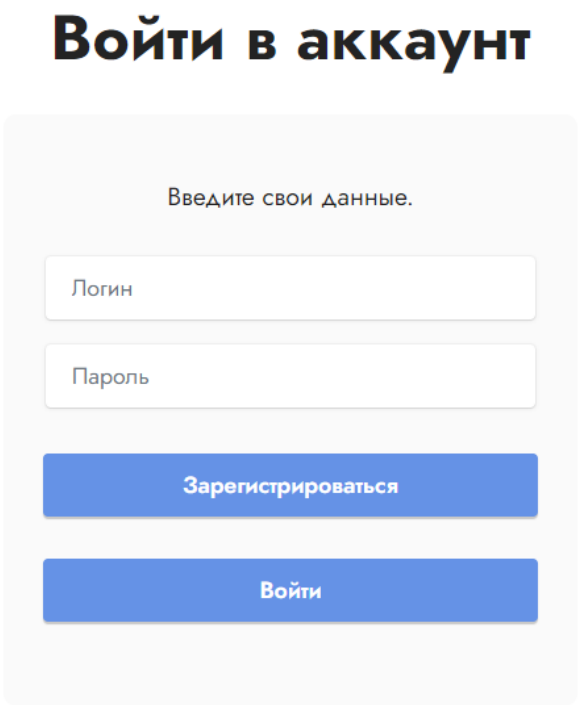


Рисунок 32 – Форма авторизации и регистрации

Форма авторизации и регистрации содержит следующие функциональные элементы:

– поле заполнения логина пользователя;

– поле заполнения пароля пользователя;

– кнопка «Зарегистрироваться», инициирующая процесс регистрации и последующей авторизации;

– кнопка «Войти», завершающая процесс авторизации.

После заполнения формы пользователь имеет возможность войти в систему под имеющейся учетной записью, нажав кнопку «Войти», либо создать новую учетную запись, нажав кнопку зарегистрироваться.

Обе упомянутые кнопки ведут на страницу содержащую форму тестирования, представленную на рисунке 33.

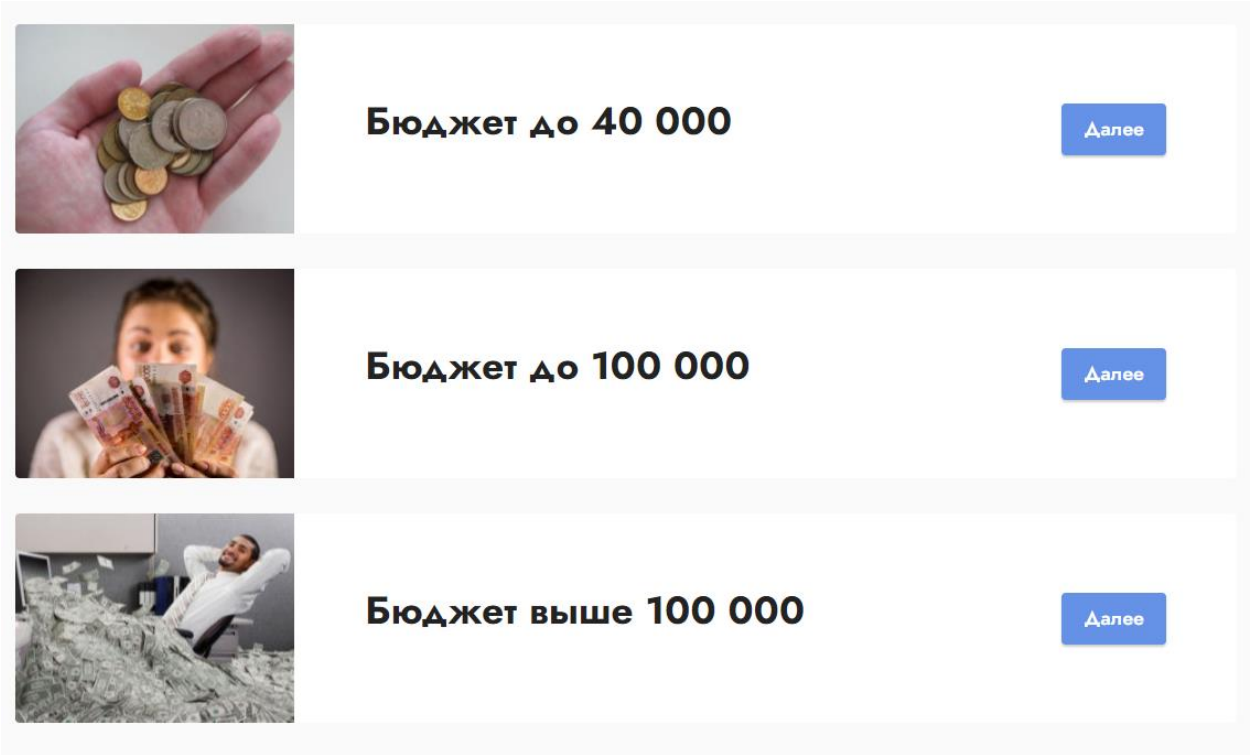


Рисунок 33 – Форма тестирования

Форма тестирования содержит элементы, представляющие собой иллюстрации и варианты ответа с кнопками «Далее», служащими переходами между страницами теста.

После прохождения тестирования пользователю будет представлена форма с результатами, изображенная на рисунке 34.

Название	Минимальная цена	Новый инструмент	Подержанный инструмент
Harley Benton Fusion-II	29 000 руб.	Просмотреть	Просмотреть
Fender Squier Affinity Strat	14 500 руб.	Просмотреть	Просмотреть
Jackson Adrian Smith	39 000 руб.	Просмотреть	Просмотреть
Fender Squier Affinity Tele	15 000 руб.	Просмотреть	Просмотреть
Yamaha Pacifica 112	22 000 руб.	Просмотреть	Просмотреть

Рисунок 34 – Форма с результатами тестирования

Форма представляет собой таблицу с отображением атрибутов сущности «Гитары». Столбцы «Новый инструмент» и «Подержанный инструмент» содержат ссылки на страницы интернет-магазина thomann.de и платформы avito.ru соответственно. Ссылка на страницу платформы содержит в себе параметры для поиска объявлений, содержащих название инструмента. Пример страницы, отображающейся после перехода по ссылке, ведущей на платформу avito.ru, представлен на рисунке 35.

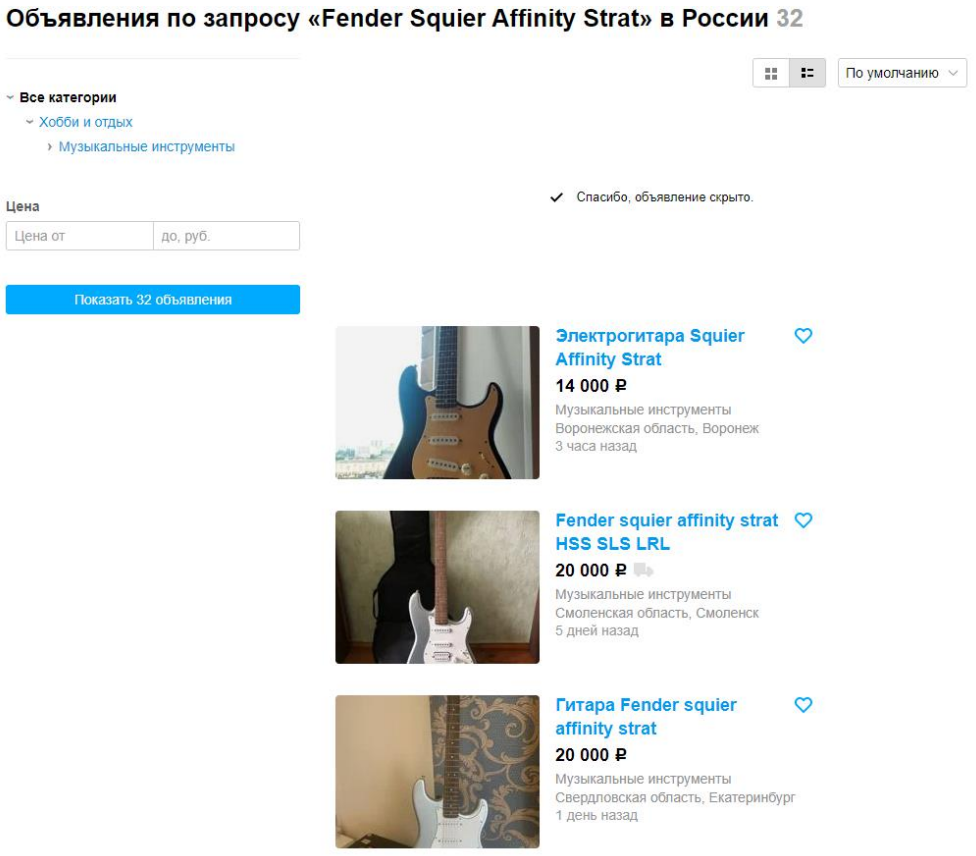


Рисунок 35 – Пример страницы, отображающейся после перехода по ссылке, ведущей на платформу avito.ru

В системе также предусмотрена возможность просмотра всех имеющихся в базе данных инструментов. Отображение происходит при помощи формы «Каталог», представленной на рисунке 36.

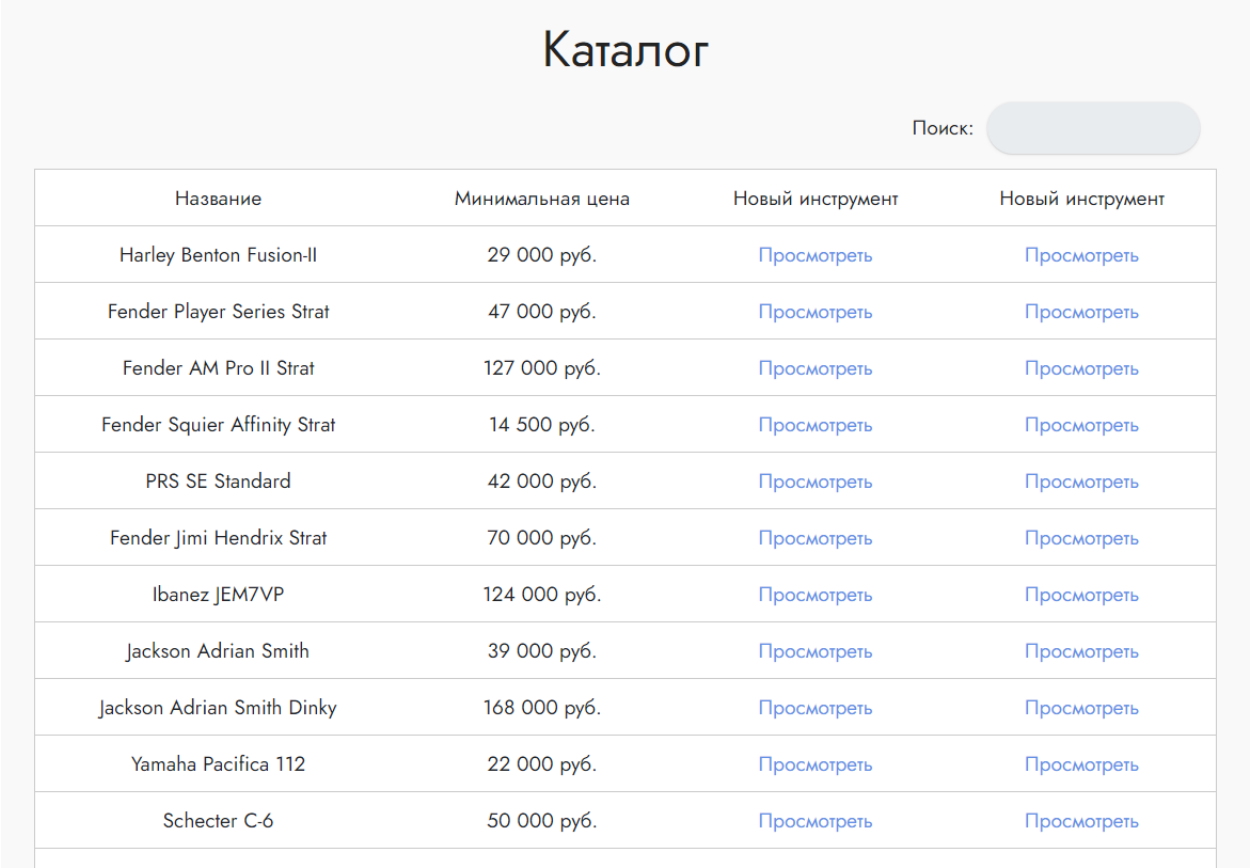


Рисунок 36 – Форма «Каталог»

Форма «Каталог» содержит инструмент для поиска, позволяющий отфильтровать записи по названию.

Таким образом был спроектирован интерфейс рассматриваемого сервиса по подбору электрогитары. Для работы с представленным интерфейсом не требуется иметь специфичных навыков.

4 Контроль качества программного обеспечения

Контроль качества программного обеспечения – неотъемлемый этап в разработке приложений. Одним из важнейших способов контроля качества является тестирование программного обеспечения.

Тестирование – способ обеспечения качества программного продукта, которое представляет собой ряд мероприятий, связанных с различного рода испытаниями тестируемого объекта для установления соответствия или несоответствия техническим требованиям и направленных на выявление дефектов [17].

Под дефектами понимаются ошибки в программном обеспечении, которые ведут к нарушению работы приложения. Необнаруженные на стадии разработки дефекты способны принести заказчику материальные убытки, а также исправление таких ошибок при эксплуатации программного обеспечения может быть затруднено и экономически невыгодно. Поэтому тестирование является неотъемлемой частью разработки приложений.

Для выпускной квалификационной работы было выбрано тестирование методом черного ящика, которое подразумевает по собой осуществление функционального тестирования.

4.1 Функциональное тестирование

Функциональное тестирование – способ проверки программного обеспечения путем составления плана тестирования, который строится на основе прецедентов, описанных в диаграмме вариантов использования. Следовательно, план тестирования включает в себя не только описание самих тестов, но и их предполагаемые результаты выполнения.

Функциональное тестирование является одним из ключевых видов тестирования, задача которого установить соответствие разработанного программного обеспечения исходным функциональным требованиям заказчика [18].

Результаты функционального тестирования приведены в приложении В. План функционального тестирования приведен в таблице 11.

Таблица 11 – План функционального тестирования

№	Вариант использования	Тест	Результаты
1	Авторизация	Авторизация с неверными данными	Тест выполнен. Система не нашла пользователя в базе и выдала ошибку (рисунок Г.1).
2		Авторизация с верными данными	Тест выполнен. Система перенаправила пользователя на страницу прохождения тестирования (рисунок Г.2).
3	Регистрация	Регистрация с имеющимся логином.	Тест выполнен. Система вывела сообщение об ошибке (рисунок Г.3).
4		Регистрация с пустыми полями.	Система вывела сообщение об ошибке (рисунок Г.4).
5		Регистрация с корректными данными.	Тест выполнен. Система перенаправила пользователя на страницу прохождения тестирования (рисунок Г.5).
6	Прохождение тестирования	Прохождение тестирования и получение выборки предложений.	Тест выполнен. Коллекция гитар добавлена в базу данных и выведена в форму результатов. (рисунок Г.6).
7	Просмотр каталога электрогитар	Перейти по ссылке «Каталог» на титульной форме страницы.	Тест выполнен. Пользователю выведены все записи в базе данных (рисунок Г.7).
8	Просмотреть результаты последнего тестирования	Перейти по ссылке «Профиль» на титульной форме страницы.	Тест выполнен. Выборка загружена из базы данных и выведена в форму (рисунок Г.8).

Таким образом, проанализировав результаты функционального тестирования можно сделать вывод о том, что разработанная информационная система сервиса по подбору электрогитары работает корректно и соответствует разработанным функциональным требованиям.

4.2 Метрики кода

Метрики кода представляют собой набор оценок программного обеспечения, которые дают разработчикам более глубокое представление о разрабатываемом коде и служат рекомендательными характеристиками. Они позволяют руководителям проектов и предприятий изучить сложность разработанного или разрабатываемого проекта, оценить объем работ, стилистику разрабатываемой программы и усилия, потраченные каждым разработчиком для реализации решения [19].

Примером часто используемой метрики является комплексный показатель качества кода. Рассчитывается метрика по формуле:

$$MI = \max\left(0, (171 - 5.2 * ln(HV) - 0.23 * CC - 16.2 * ln(LoC)) * \frac{100}{171}\right),$$

где HV – Halstead Volume, вычислительная сложность: чем больше операторов, тем больше значение этой метрики;

CC – Cyclomatic Complexity, показатель структурной сложности кода, т.е. количество различных ветвей в коде, чем больше этот показатель, тем больше тестов должно быть написано, для полного покрытия кода;

LoC – количество строк кода.

Комплексный показатель качества кода может принимать значения от 0 до 100 и показывает относительную сложность поддержки кода. Чем больше значение этой метрики, тем легче поддерживать код.

Для определения вычислительной сложности программы сначала следует рассчитать словарь (1) и длину программы (2), а затем вычислить на их основе вычислительную сложность (3).

Словарь программы:

$$\eta = \eta_1 + \eta_2, \quad (1)$$

где η_1 – число простых (отдельных) операторов и операций в программе;

η_2 – число простых (отдельных) операндов (констант и переменных) в программе.

Длина программы:

$$N = N_1 + N_2, \quad (2)$$

где N_1 – общее число простых (отдельных) операторов и операций в программе;

N_2 – общее число простых (отдельных) операндов в программе.

Вычислительная сложность:

$$HV = Nlog_2\eta. \quad (3)$$

Структурную сложность программы можно рассчитать по формуле:

$$CC = КОР + КОЦ + 1,$$

где КОР –количество операторов разветвления;

КОЦ – количество операторов цикла.

Если полученное значение СС больше 10, то это свидетельствует о структурной сложности программы [20].

Для анализа метрик кода выбрана функция получения и обработки данных пользователя со страницы login.html, представленная в листинге 3.

Листинг 3 – Функция получения и обработки данных пользователя с HTML-страницы login.html

```
public String loginData(@RequestParam String type, @RequestParam
String login, @RequestParam String password, Model model) {
    Users user = new Users(login, password);
    if (type == "authorization") {
        if (userRepos.findByUsername(login) != null) {
            model.addAttribute("currentUser", user);
            return "redirect:/profile";
        } else if ((password == "") || (login == "")) {
            model.addAttribute("message", "Некоторые из полей
пусты!");
            return "login";
        } else {
            model.addAttribute("message", "Пользователь не
```

```
найден!");
            return "login";
        }
    } else {
        if (userRepos.findByUsername(login) != null) {
            userRepos.save(user);
            model.addAttribute("currentUser", user);
            return "redirect:/profile";
        } else if ((password == "") || (login == "")) {
            model.addAttribute("message", "Некоторые из полей
пусты!");
            return "login";
        } else {
            model.addAttribute("message", "Пользователь уже
зарегистрирован в системе!");
            return "login";
        }
    }
}
```

Произведем расчет комплексного показателя качества кода (MI) для функции (смотри листинг 3).

Данные, необходимые для расчета формул 1 и 2 приведены в таблице 12.

Таблица 12 – Данные для расчета словаря и длины программы

Операторы			Операнды		
№	Оператор	Количество	№	Операнд	Количество
1	if	3	1	type	1
2	else if	2	2	login	5
3	return	6	3	password	3
4	==	4	4	user	3
5	!=	2			
6	else	3			
Итог	η_1=6	N_1=20		η_2=4	N_2=12

Словарь программы:

$$\eta = \eta_1 + \eta_2 = 6+4 = 10.$$

Длина программы:

$$N = N_1 + N_2 = 20 + 12 = 32.$$

Вычислительная сложность:

$$HV = 50 \times 3{,}807 = 106.3.$$

Количество строк кода:

$$LoC = 29.$$

Структурная сложность:

$$CC=8+0+1=9.$$

Расчет комплексного показателя качества:

$$MI = \max\left(0, (171 - 5.2 * \ln(106.3) - 0.23 * 9 - 16.2 * \ln(29)) * \frac{100}{171}\right)=112,5.$$

Комплексный показатель качества приблизительно равен 112,5.

На основе результатов расчёта можно сделать вывод о том, что структурная сложность данной функции невысока, следовательно, программное обеспечение не является структурно сложным. Комплексный показатель качества кода находится чуть выше среднего уровня, что говорит о том, что исходный код программного обеспечения сравнительно просто поддерживать.

Таким образом, программное обеспечение информационной системы сервиса по подбору электрогитары имеет средний уровень качества.

Заключение

В ходе выполнения выпускной квалификационной работы была разработана информационная система сервиса по подбору электрогитары.

Был проведен анализ предметной области и обзор систем, позволяющих подобрать гитару. На этом этапе разработки были определены функциональные и нефункциональные требования к разрабатываемой системе.

В процессе проектирования были выбраны технологии проектирования, выполнено планирование проекта и его бюджета, а также разработана логическая модель данных, макет веб-сайта и диаграммы UML.

На основе моделей и диаграмм, полученных в результате проектирования, разработаны физическая модель базы данных, классы приложения, диаграмма компонентов, диаграмма развертывания и пользовательский интерфейс. Разработанное приложение написано на языке объектно-ориентированного программирования Java и использует фреймворк Spring, который является одним из самых перспективных фреймворков для разработки веб-приложений.

Разработанные программные средства были протестированы методами функционального тестирования и оценены методом анализа метрик кода. В результате тестирования система показала средние результаты, что является достаточным для комфортной эксплуатации пользователями.

Список использованных источников

1. Мартин Кох Изготовление электрогитары / Мартин Кох. - 2001. - 5 с.

2. Устройство звукоснимателя электрогитары // Tvoya-gitara [Электронный ресурс]. – Режим доступа: http://tvoya-gitara.ru/ustroystvo-gitary/ustroystvo-zvukosnimatelya-electrogitary (дата обращения: 10.06.2021).

3. Типы корпусов электрогитар // Tvoya-gitara [Электронный ресурс]. – URL: http://tvoya-gitara.ru/ustroystvo-gitary/tipyi-korpusov-elektrogitar (дата обращения: 10.06.2021).

4. Анализ требований к программному обеспечению // Lektsii [Электронный ресурс]. – URL: https://lektsii.org/7-10861.html (дата обращения: 10.06.2021).

5. Руководство по MySQL // Metanit [Электронный ресурс]. – URL: https://metanit.com/sql/mysql/ (дата обращения: 10.06.2021).

6. Дейк К., Введение в системы баз данных/ К. Дейк. – М. Вильямс, 2001. – 1072 с.

7. Лекция 7: Диаграммы прецедентов: крупным планом // Intuit [Электронный ресурс]. – URL: https://intuit.ru/studies/courses/1007/229/lecture/5962 (дата обращения: 10.06.2021).

8. UML-диаграммы классов [Электронный ресурс]. – URL: https://prog-cpp.ru/uml-classes/ (дата обращения: 10.06.2021).

9. Лекция 3: Виды диаграмм UML// Intuit [Электронный ресурс]. – URL: https://intuit.ru/studies/courses/1007/229/lecture/5954?page=4 (дата обращения: 10.06.2021).

10. Прошкина, Е. Н. Методы и средства проектирования информационных систем и технологий. Методические указания к выполнению лабораторных работ / Е. Н. Прошкина. – Пенза: ПГУ, 2014. – 92с.

11. State & Transition Diagram — что это и как применять // Хабр Электронный ресурс]. – URL: https://habr.com/ru/post/548192/ (дата обращения: 10.06.2021).

12. Иванов, Д. Моделирование на UML. Учебно-методическое пособие / Д. Иванов, Ф. Новиков – 2017. – 200 с.

13. Интерфейс пользователя // Национальная библиотека им. Н. Э. Баумана [Электронный ресурс]. URL: https://ru.bmstu.wiki/GUI (дата обращения: (10.06.2021).

14. Мокрозуб, В. Г. Реляционные базы данных в автоматизированных интеллектуальных информационных системах / В. Г. Мокрозуб. – Монография: Издательский дом «Спектр», 2011. - 108 с.

15. Иванов, Д. Моделирование на UML. Учебно-методическое пособие / Д. Иванов, Ф. Новиков – 2017. – 200 с.

16. Простое руководство к диаграммам развертывания UML // Creatly [Электронный ресурс]. – URL: https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-развёрт/ (дата обращения: 10.06.2021).

17. Фундаментальная теория тестирования — что это и как применять // Хабр [Электронный ресурс]. – URL: https://habr.com/ru/post/549054/ (дата обращения: 10.06.2021).

18. Функциональное тестирование программного обеспечения // daglab [Электронный ресурс]. – URL: https://daglab.ru/funkcionalnoe-testirovanie-programmnogo-obespechenija/ (дата обращения: 10.06.2021).

19. Программный код и его метрики // Хабр [Электронный ресурс]. – URL: https://habr.com/ru/company/intel/blog/106082/ (дата обращения: 10.06.2021).

20. Метрики качества программного обеспечения // Coderlessons [Электронный ресурс]. – URL: https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/izuchite-upravlenie-kachestvom-programmnogo-obespecheniia/metriki-kachestva-programmnogo-obespecheniia (дата обращения: 10.06.2021).

Приложение А

ЛИСТИНГ ПРОГРАММЫ

(рекомендуемое)

Листинг А.1 – Класс GuitarWebAppApplication

```
@AutoConfigurationPackage
@SpringBootApplication
@ComponentScan(basePackages= {
      "com.example.demo.controllers",
      "com.example.demo"
       })
@EntityScan("com.example.demo.models")
@EnableJpaRepositories("com.example.demo.repos")
public class GuitarWebAppApplication {

   public static void main(String[] args) {
      SpringApplication.run(GuitarWebAppApplication.class,
args);
   }
}
```

Листинг А.2 – Класс CatalogController

```
@Controller
public class CatalogController {

    @Autowired
    private GuitarsRepos guitarRepos;

    @GetMapping("/catalog.html")
    public String catalog(Model model) {
        Iterable<Guitars> guitars = guitarRepos.findAll();
        model.addAttribute("guitars", guitars);
        return "catalog";
    }
}
```

Листинг А.3 – Класс LoginController

```
@Controller
public class LoginController {
    @Autowired
    private UsersRepos userRepos;

    @GetMapping("/login.html")
    public String login(Model model) {
        Iterable<Users> users = userRepos.findAll();
        model.addAttribute("users", users);
        return "login";
    }

    @PostMapping("/login.html")
    public String loginData(@RequestParam String type,
@RequestParam String login, @RequestParam String password, Model
model) {
        Users user = new Users(login, password);
```

```
        if (type == "authorization") {
            if (userRepos.findByUsername(login) != null) {
                ActiveUser.activeUser = user;
                model.addAttribute("currentUser", login);
                return "redirect:/profile";
            } else if ((password == "") || (login == "")) {
                model.addAttribute("message", "Некоторые из
полей пусты!");
                return "login";
            } else {
                model.addAttribute("message", "Пользователь не
найден!");
                return "login";
            }
        } else {
            if (userRepos.findByUsername(login) != null) {
                userRepos.save(user);
                ActiveUser.activeUser = user;
                model.addAttribute("currentUser", user);
                return "redirect:/profile";
            } else if ((password == "") || (login == "")) {
                model.addAttribute("message", "Некоторые из
полей пусты!");
                return "login";
            } else {
                model.addAttribute("message", "Пользователь уже
зарегистрирован в системе!");
                return "login";
            }
        }
    }
}
```

Листинг А.4 – Класс ProfileController

```
@Controller
public class ProfileController {

    @Autowired
    private UsersRepos userRepos;
    @Autowired
    private CollectionsRepos collectionsRepos;

    @GetMapping("/profile.html")
    public String profile(Model model) {
        model.addAttribute("users", ActiveUser.activeUser);
        Collections collections =
collectionsRepos.findByUserId(ActiveUser.activeUser.getIdUser())
;
        return "profile";
    }

    @PostMapping("/profile.html")
    public String profilePost(@RequestParam String post, Model
```

```
model){
        if (post == "exit"){
            ActiveUser.activeUser =
userRepos.findByUsername("guest");
            return "start";
        }
        else if (post == "test") return "redirect:/testing1";
        else return "profile";
    }
}
```

Листинг А.5 – Класс StartController

```
@Controller
public class StartController {
    @GetMapping("/")
    public String start(Model model) {
        return "start";
    }
}
```

Листинг А.6 – Класс TestingController

```
@Controller
public class TestingController {
    private Long[] argumentArray;
    @Autowired
    private JenresRepos jenresRepos;
    @Autowired
    private MusiciansRepos musiciansRepos;
    @Autowired
    private GuitarsRepos guitarsRepos;
    @Autowired
    private UsersRepos usersRepos;
    @Autowired
    private CollectionsRepos collectionsRepos;

    @GetMapping("/testing1.html")
    public String testing1(Model model) {
        Iterable<Jenres> jenres = jenresRepos.findAll();
        Iterable<Musicians> musicians =
musiciansRepos.findAll();
        model.addAttribute("jenres", jenres);
        model.addAttribute("musicians", musicians);
        return "testing1";
    }

    @PostMapping("/testing1.html")
    public String getTesting1(@RequestParam Long answer, Model
model) {
        argumentArray[0] = answer;
        return "testing2";
    }
```

```
    @GetMapping("/testing2.html")
    public String testing2(Model model) {
        Iterable<Jenres> jenres = jenresRepos.findAll();
        Iterable<Musicians> musicians =
musiciansRepos.findAll();
        model.addAttribute("jenres", jenres);
        model.addAttribute("musicians", musicians);
        return "testing2";
    }

    @PostMapping("/testing2.html")
    public String getTesting2(@RequestParam Long answer, Model
model) {
        argumentArray[1] = answer;
        return "testing3";
    }

    @GetMapping("/testing3.html")
    public String testing3(Model model) {
        Iterable<Jenres> jenres = jenresRepos.findAll();
        Iterable<Musicians> musicians =
musiciansRepos.findAll();
        model.addAttribute("jenres", jenres);
        model.addAttribute("musicians", musicians);
        return "testing3";
    }

    @PostMapping("/testing3.html")
    public String getTesting3(@RequestParam Long answer, Model
model) {
        argumentArray[2] = answer;
        return "testing4";
    }

    @GetMapping("/testing4.html")
    public String testing4(Model model) {
        Iterable<Jenres> jenres = jenresRepos.findAll();
        Iterable<Musicians> musicians =
musiciansRepos.findAll();
        model.addAttribute("jenres", jenres);
        model.addAttribute("musicians", musicians);
        return "testing4";
    }

    @PostMapping("/testing4.html")
    public String getTesting4(@RequestParam Long answer, Model
model) {
        argumentArray[3] = answer;
        return "testing5";
    }

    @GetMapping("/testing5.html")
    public String testing5(Model model) {
```

```
        Iterable<Jenres> jenres = jenresRepos.findAll();
        Iterable<Musicians> musicians =
musiciansRepos.findAll();
        model.addAttribute("jenres", jenres);
        model.addAttribute("musicians", musicians);
        return "testing5";
    }

    @PostMapping("/testing5.html")
    public String getTesting5(@RequestParam Long answer, Model
model) {
        argumentArray[4] = answer;
        return "testing6";
    }

    @GetMapping("/testing6.html")
    public String testing6(Model model) {
        Iterable<Jenres> jenres = jenresRepos.findAll();
        Iterable<Musicians> musicians =
musiciansRepos.findAll();
        model.addAttribute("jenres", jenres);
        model.addAttribute("musicians", musicians);
        return "testing6";
    }

    @PostMapping("/testing6.html")
    public String getTesting6(@RequestParam Long answer, Model
model) {
        argumentArray[5] = answer;
        return "testing7";
    }

    @GetMapping("/testing7.html")
    public String testing7(Model model) {
        Iterable<Jenres> jenres = jenresRepos.findAll();
        Iterable<Musicians> musicians =
musiciansRepos.findAll();
        model.addAttribute("jenres", jenres);
        model.addAttribute("musicians", musicians);
        return "testing7";
    }

    @PostMapping("/testing7.html")
    public String getTesting7(@RequestParam Long answer, Model
model) {
        argumentArray[6] = answer;
        return "results";
    }

    @GetMapping("/results.html")
    public String results(Model model) {
        Iterable<Guitars> guitars =
guitarsRepos.testResult(argumentArray[0],argumentArray[1],argume
ntArray[2], argumentArray[3],argumentArray[4],argumentArray[5],
```

```
argumentArray[6]);
        model.addAttribute("guitars", guitars);
        Collections collections = new Collections();
        while (guitars != null){
           Guitars singleGuitar =guitars.iterator().next();
        collections =
collectionsRepos.addCollection(singleGuitar.getIdGuitars(),
ActiveUser.activeUser.getIdUser(),
collections.getIdCollections());
        }
        return "results";
    }
}
```

Листинг А.7 – Класс-сущность Collections

```
@Entity
@Table(name = "collections_has_guitars")
public class Collections {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "collections_id_collections")
    private Long idCollections;
    @Column(name = "collections_users_id_user")
    private Long idUser;
    @Column(name = "guitars_id_guitars")
    private Long idGuitar;

    public Long getIdCollections() {
        return idCollections;
    }

    public void setIdCollections(Long idCollections) {
        this.idCollections = idCollections;
    }

    public Long getIdUser() {
        return idUser;
    }

    public void setIdUser(Long idUser) {
        this.idUser = idUser;
    }

    public Long getIdGuitar() {
        return idGuitar;
    }

    public void setIdGuitar(Long idGuitar) {
        this.idGuitar = idGuitar;
    }
}
```

Листинг А.8 – Класс-сущность Guitars

```
@Entity
@Table(name = "guitars", schema = "guitardb")
public class Guitars {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long idGuitars;
    @Column(name = "forms_id_forms")
    private Long idForm;
    @Column(name = "pickup_config_id_pickups")
    private Long idPickupConfiguration;
    @Column(name = "name")
    private String name;
    @Column(name = "url")
    private String url;
    @Column(name = "url_bu")
    private String urlBU;

    public Guitars() {
    }

    public void setIdGuitars(Long idGuitars) {
        this.idGuitars = idGuitars;
    }

    public void setIdForm(Long idForm) {
        this.idForm = idForm;
    }

    public void setIdPickupConfiguration(Long
idPickupConfiguration) {
        this.idPickupConfiguration = idPickupConfiguration;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public void setUrlBU(String urlBU) {
        this.urlBU = urlBU;
    }

    public Long getIdGuitars() {
        return idGuitars;
    }

    public Long getIdForm() {
        return idForm;
```

```
    }

    public Long getIdPickupConfiguration() {
        return idPickupConfiguration;
    }

    public String getName() {
        return name;
    }

    public String getUrl() {
        return url;
    }

    public String getUrlBU() {
        return urlBU;
    }
}
```

Листинг А.9 – Класс-сущность Jenres

```
@Entity
@Table (name = "jenres")
public class Jenres {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long idJenre;
    @Column(name = "name")
    private String name;

    public void setIdJenre(Long idJenre) {
        this.idJenre = idJenre;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Long getIdJenre() {
        return idJenre;
    }

    public String getName() {
        return name;
    }
}
```

Листинг А.10 – Класс-сущность Musicians

```
@Entity
@Table(name = "musicians")
public class Musicians {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long idMusician;
    @Column(name = "name")
    private String name;

    public void setIdMusician(Long idMusician) {
        this.idMusician = idMusician;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Long getIdMusician() {
        return idMusician;
    }

    public String getName() {
        return name;
    }
}
```

Листинг А.11 – Класс-сущность Users

```
@Entity
@Table(name = "users")
public class Users {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long idUser;
    @Column(name = "login")
    private String login;
    @Column(name = "password")
    private String password;

    public Users(){
    }

    public void setIdUser(Long idUser) {
        this.idUser = idUser;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Users(String login, String password) {
```

```
    }

    public Long getIdUser() {
        return idUser;
    }

    public String getLogin() {
        return login;
    }

    public String getPassword() {
        return password;
    }

}
```

Листинг А.12 – Интерфейс CollectionsRepos

```
@Repository
public interface CollectionsRepos extends
CrudRepository<Collections, Long> {

    @Query(value = "INSERT INTO
guitardb.collections(id_collections, users_id_user)\n" +
            "SELECT * FROM (newId, IdUsersP) AS tmp\n" +
            "WHERE NOT EXISTS (\n" +
            "    SELECT name FROM guitardb.collections WHERE
users_id_user = idUsersP\n" +
            ") LIMIT 1;\n" +
            "insert into
guitardb.collections_has_guitars(collections_id_collections,
collections_users_id_user, guitars_id_guitars) values (newId,
idUsersP,idGuitarsP);",
            nativeQuery = true)
    Collections addCollection(@Param("idGuitarsP") Long
idGuitars, @Param("idUsersP") Long idUsers, @Param("newId") Long
newCollectionId);

    @Query(value = "select * from collections_has_guitars
where\n" +
            "collections_users_id_user = idUserP;",
            nativeQuery = true)
    Collections findByUserId(@Param("idUserP") Long idUsers);
}
```

Листинг А.13 – Интерфейс GuitarsRepos

```
@Repository
public interface GuitarsRepos extends CrudRepository<Guitars,
Long> {
    @Query(value = "SELECT * from guitars where\n" +
            "forms_id_forms in \n" +
            "\t(select id_forms from Forms where\n" +
```

```
            "\tid_forms in \n" +
            "\t\t(select forms_id_forms from forms_has_musicians
where\n" +
            "        musicians_id_musicians = (idMus1 or idMus2
or idMus3)))\n" +
            "and\n" +
            "pickup_config_id_pickup_config in \n" +
            "\t(select id_pickup_config from pickup_config
where\n" +
            "    id_pickup_config in (select pickups_id_pickups
from pickups_has_jenres where\n" +
            "\t\tjenres_id_jenres = (idJen1 or idJen2 or
idJen3)))" +
            "and " +
            "(price <= priceT);" +
            "",
            nativeQuery = true)
    Iterable<Guitars> testResult(@Param("priceT") Long price,
                         @Param("idMus1") Long idMus1,
@Param("idMus2") Long idMus2, @Param("idMus3") Long idMus3,
                         @Param("idJen1") Long idJen1,
@Param("idJen2") Long idJen2, @Param("idJen3") Long idJen3);
}
```

Листинг А.17 – Интерфейс UsersRepos

```
@Repository
public interface UsersRepos extends CrudRepository<Users, Long>
{
    @Query(value = "SELECT * from users where name = :username",
        nativeQuery = true)
    Users findByUsername(@Param("username") String username);
}
```

Приложение Б

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

(рекомендуемое)

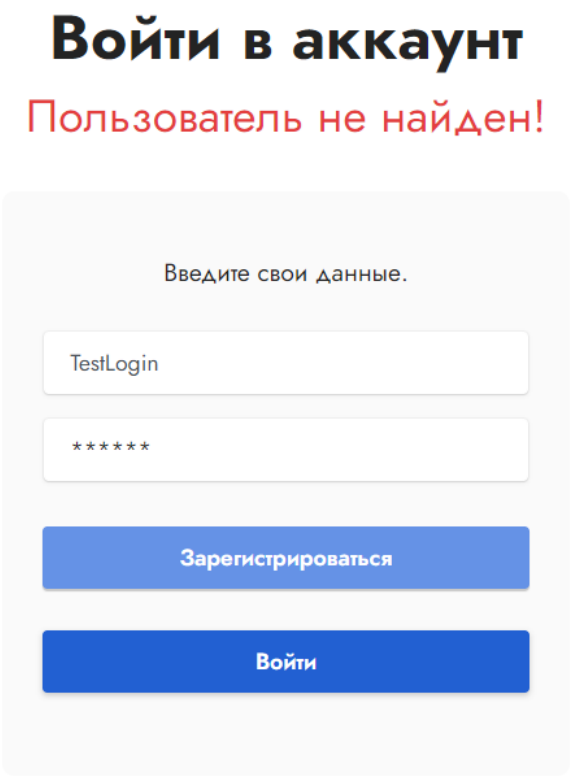


Рисунок Б.1 – Результат функционального теста №1

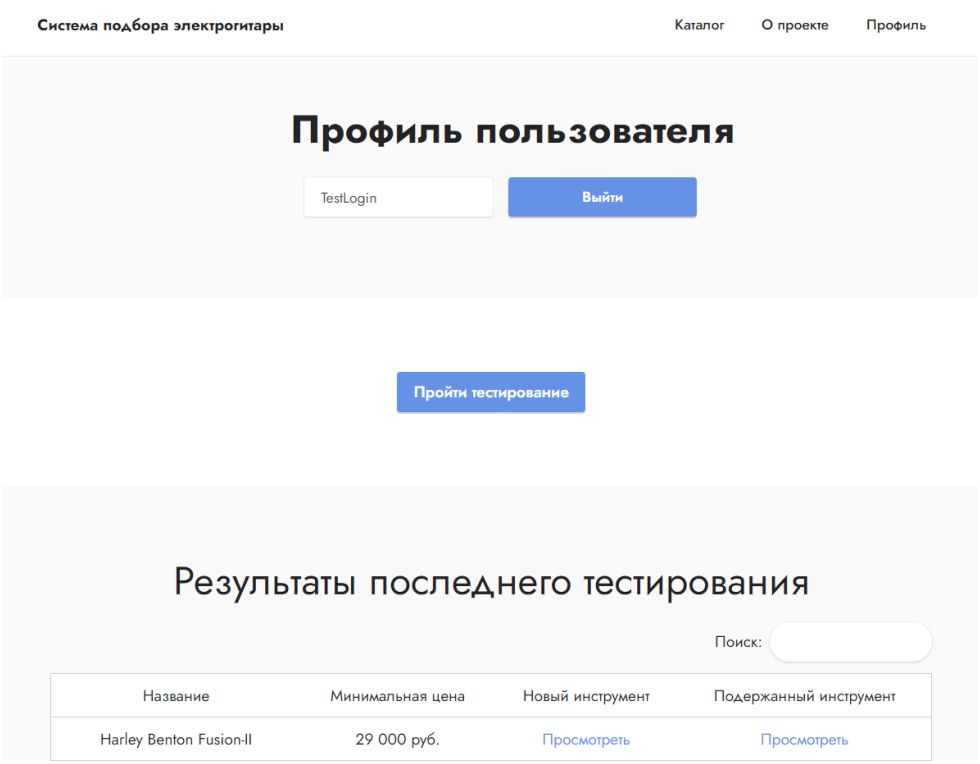


Рисунок Б.2 – Результат функционального теста №2

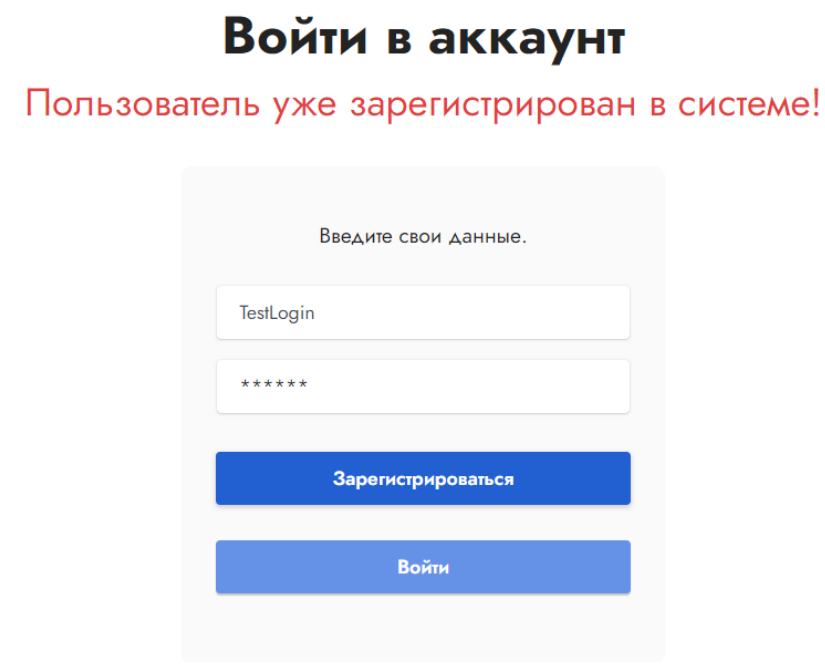


Рисунок Б.3 – Результат функционального теста №3

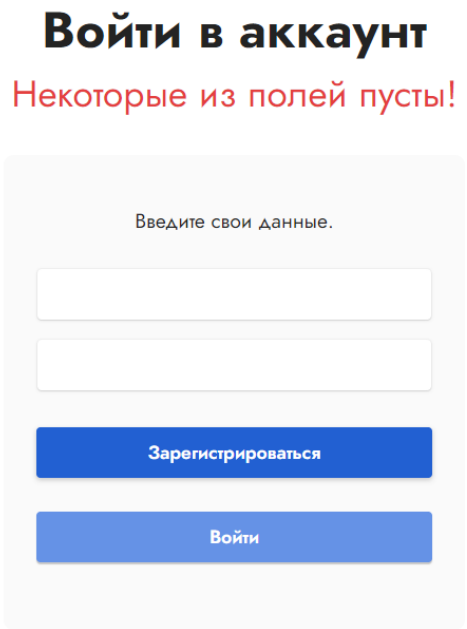


Рисунок Б.4 – Результат функционального теста №4

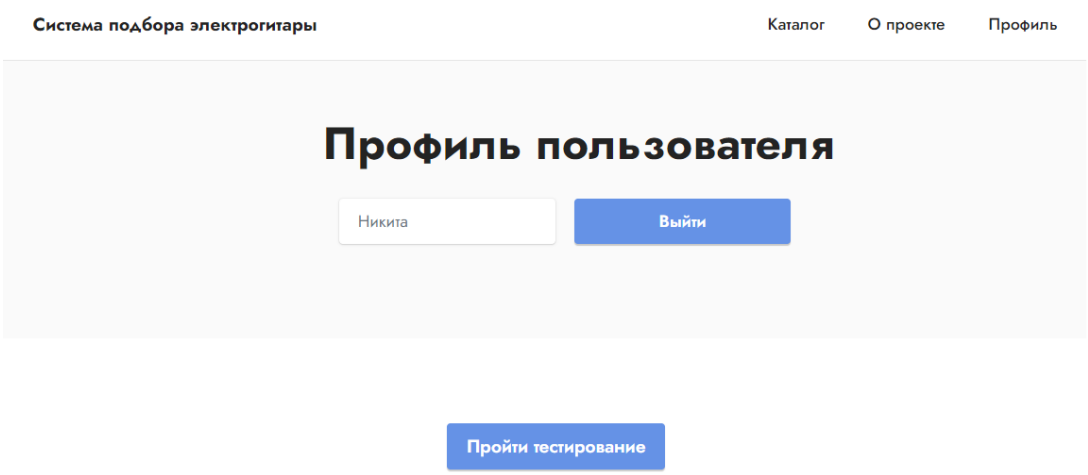


Рисунок Б.5 – Результат функционального теста №5

Название	Минимальная цена	Новый инструмент	Подержанный инструмент
Harley Benton Fusion-II	29 000 руб.	Просмотреть	Просмотреть
Fender Squier Affinity Strat	14 500 руб.	Просмотреть	Просмотреть
Jackson Adrian Smith	39 000 руб.	Просмотреть	Просмотреть
Fender Squier Affinity Tele	15 000 руб.	Просмотреть	Просмотреть
Yamaha Pacifica 112	22 000 руб.	Просмотреть	Просмотреть

Рисунок Б.6 – Результат функционального теста №6

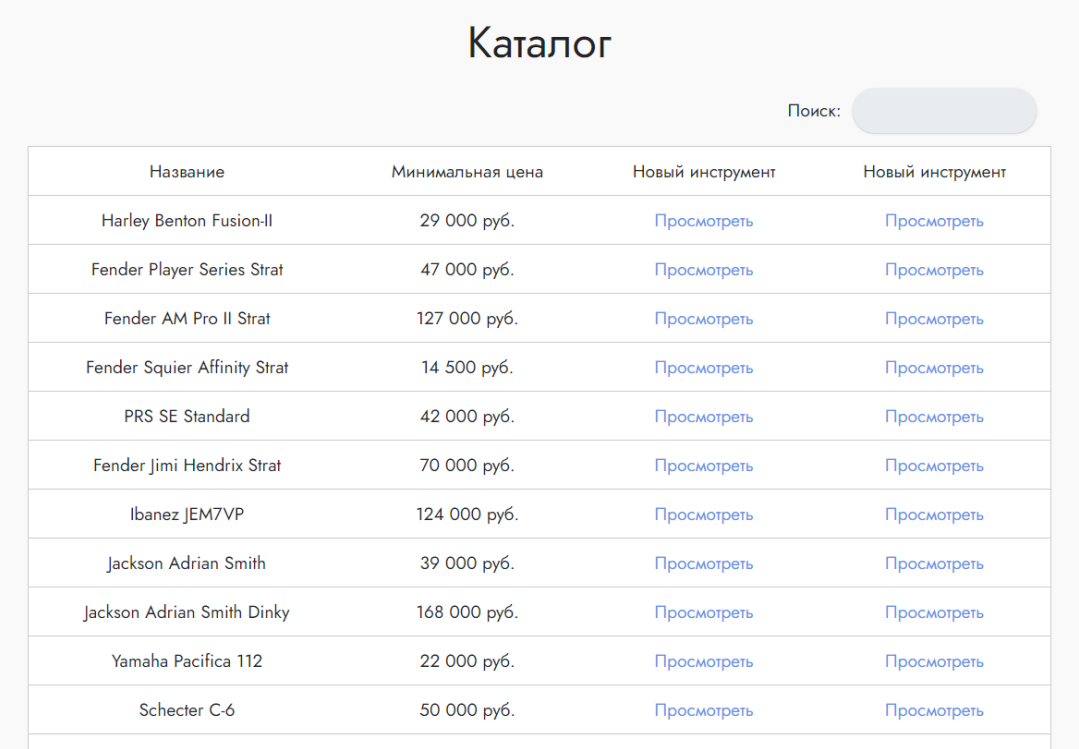


Рисунок Б.7 – Результат функционального теста №7

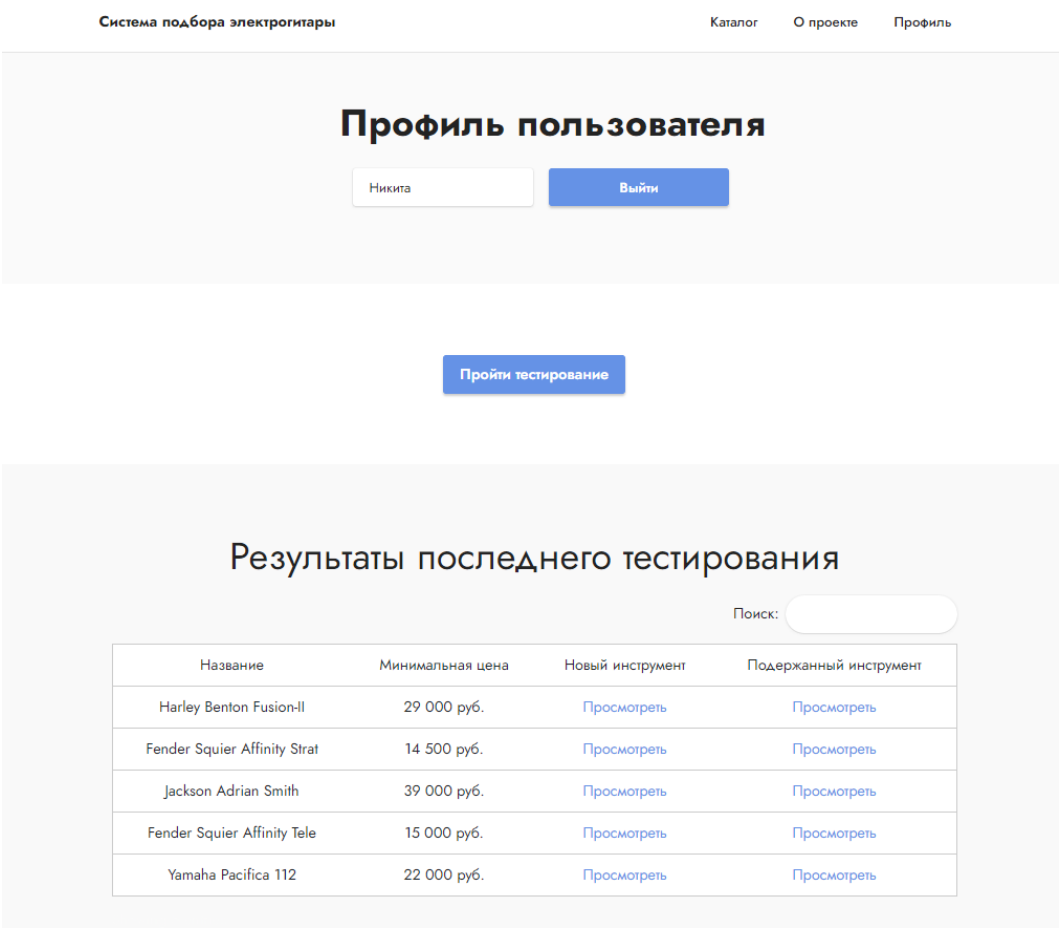


Рисунок Б.8 – Результат функционального теста №8