Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Кафедра ИСиТ

Дисциплина: БД

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ПС «АРМ Шиномонтаж»

БГУИР КП 1-40.01.01.310 ПЗ

Студент: гр. 981063 Ефименко П.В.

Руководитель: Бакунов А.М.

Минск 2021

СОДЕРЖАНИЕ

[Введение 4](#_Toc68257196)

[1 Разработка программного обеспечения 6](#_Toc68257197)

[1.1 Обзор аналогов программных средств 6](#_Toc68257198)

[1.2 Постановка задачи 8](#_Toc68257199)

[1.3 Разработка спецификации требований к ПС 9](#_Toc68257200)

[1.4 Разработка структурной схемы ПС 10](#_Toc68257201)

[1.5 Разработка алгоритма работы ПС 11](#_Toc68257202)

[1.6 Обоснование выбора языка и среды программирования 12](#_Toc68257203)

[1.7 Программная реализация разработанного алгоритма работы ПС 13](#_Toc68257204)

[1.8 Тестирование и отладка ПС 14](#_Toc68257205)

[1.9 Руководство пользователя 15](#_Toc68257206)

[ЗАКЛЮЧЕНИЕ 17](#_Toc68257207)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 18](#_Toc68257208)

[Приложение А 19](#_Toc68257209)

Введение

На сегодняшний день для работы любой программы имеется необходимость хранить данные. Успешное функционирование различных фирм, организаций и предприятий просто невозможно без развитой информационной системы, которая позволяет автоматизировать сбор и обработку данных. Обычно для хранения и доступа к данным, содержащим сведения о некоторой предметной области, создается база данных. Для веб-приложения это является неотъемлемой частью программы. В качестве хранилища для больших проектов используются базы данных.

Целью курсового проекта является углубление теоретических знаний о принципах и получение практических навыков работы с базами данных.

База данных (БД) – это программа, которая позволяет хранить и обрабатывать информацию в структурированном виде. БД – это отдельная независимая программа, которая не входит в состав языка программирования. В базе данных можно сохранять любую информацию, чтобы позже получать к ней доступ. В данном курсовом проекте будет рассмотрена лишь реляционная модель данных. Данная модель представляет данные пользователю в качестве таблиц. Такое представление является легким в восприятии и использовании.

Для изучения работы с базой данных было реализовано программное средство «Автоматизированное Рабочее Место Шиномонтаж». В качестве реализации программного средства была выбрана классификация – веб-приложение. Веб-приложения – это специальный вид приложений, которые работают в глобальной сети Интернет по протоколу HTTP. Как правило, веб-приложения не требуют установки дополнительного программного обеспечения на стороне клиента, а вся логика, в основном, выполняется на стороне сервера. Для отображения пользовательского интерфейса используется браузер – программа, способная распознавать язык разметки HTML (и сопутствующие технологии – таблицы стилей CSS, клиентский скриптовой язык программирования JavaScript и т.д.). Браузер обычно принято называть "тонким клиентом", т.е. клиентом, который содержит минимальное количество бизнес-логики.

В рамках курсового проектирования была разработана пояснительная записка.

Пояснительная записка состоит из разделов и подразделов, которые характеризуют данный курсовой проект в полной мере. В пояснительной записке 5 разделов:

* первый раздел «Техническое задание» включает в себя анализ предметной области, обзор аналогов, функциональные требования, а также выбор и обоснование средств разработки;
* второй раздел «Проектирование программного средства» предназначен для описания инфологической модели базы данных, функциональной модели программного средства и минимальных требований для работы программного средства;
* в разделе «Разработка программного средства» демонстрируется структура программного средства и базы данных, показаны и описываются ключевые классы и методы, предназначенные для работы программы;
* в разделе «Тестирование» описывается подготовка к тестированию и проведение тестирование с надлежащей документации;
* в разделе «Руководство пользователя» находится руководство по использованию каждой функции приложения.
* Заключение представляет собой анализ проделанной работы и оценку выполнения поставленной задачи, выводы, сделанные во время выполнения данного курсового проекта и после его написания.

Приложение содержит код программы и разработанные тесты, а графическая часть состоит из созданных для курсового проекта диаграмм.

Поставка задачи и составление технического задания

* 1. Анализ предметной области

Веб-приложения – это специальный вид приложений, которые работают в глобальной сети Интернет по протоколу HTTP. Как правило, веб-приложения не требуют установки дополнительного программного обеспечения на стороне клиента, а вся логика, в основном, выполняется на стороне сервера. Для отображения пользовательского интерфейса используется браузер – программа, способная распознавать язык разметки HTML (и сопутствующие технологии – таблицы стилей CSS, клиентский скриптовой язык программирования JavaScript и т.д.). Браузер обычно принято называть "тонким клиентом", т.е. клиентом, который содержит минимальное количество бизнес-логики.

Веб-приложения можно разделить на несколько типов, в зависимости от разных сочетаний его основных составляющих:

1. Backend (бэкенд или серверная часть приложения) работает на удаленном компьютере, который может находиться где угодно. Она может быть написана на разных языках программирования: PHP, Python, Ruby, C# и других. Если создавать приложение используя только серверную часть, то в результате любых переходов между разделами, отправок форм, обновления данных, сервером будет генерироваться новый HTML-файл и страница в браузере будет перезагружаться.
2. Frontend (фронтенд или клиентская часть приложения) выполняется в браузере пользователя. Эта часть написана на языке программирования Javascript. Приложение может состоять только из клиентской части, если не требуется хранить данные пользователя дольше одной сессии. Это могут быть, например, фоторедакторы или простые игрушки.
3. Single page application (SPA или одностраничное приложение). Более интересный вариант, когда используются и бэкенд и фронтенд. С помощью их взаимодействия можно создать приложение, которое будет работать совсем без перезагрузок страницы в браузере. Или в упрощенном варианте, когда переходы между разделами вызывают перезагрузки, но любые действия в разделе обходятся без них.

Также можно разделить на технологии их создания:

1. AJAX(Asynchronous JavaScript and XML) ­­– подход к построению пользовательских интерфейсов Web-приложений, при котором в ответ на каждое действие пользователя Web-страница на его браузере, не перезагружается полностью – с Web-сервера только догружаются нужные ему данные. Этим обеспечивается оперативная работа как одного, так и групп пользователей с приложениями. AJAX представляет собой не одну, а группу технологий и базируется на принципах использования DHTML для динамичного изменения содержания страницы и использования XMLHttpRequest для обращения к серверу (альтернативный вариант –динамическая подгрузка JavaScript с использованием объектной модели документа – DOM). С учетом этих принципов можно создавать удобные Web-интерфейсы на тех страницах сайтов, где необходимо активное взаимодействие с пользователями. Популярность AJAX приобрела после того, как компания Google начала применять его при создании Gmail, Google maps, Google suggest.
2. ASP (Active Server Pages) – технология создания Web-приложений, использующая объектную модель интерфейса, созданного на основе ISAPI-фильтра. ASP упростила задачи генерации HTML-страниц и позволила производить обращение к компонентам баз данных. Принцип, заложенный в основу интерфейса приложения, заключается в том, что на Web-странице присутствуют фрагменты кода, который интерпретируется Web-сервером и предоставляет пользователю готовый результат выполнения выбранных фрагментов кода. Web-страница, созданная с использованием технологии ASP, имеет расширение «.asp».
3. CGI(Common Gateway Interface, общий шлюзовой интерфейс) – программа поиска в удаленных БД, переадресации ссылок, использования графических меню, связи с базами данных (путем запуска программы преобразования форматов баз данных в формат языка HTML).
4. CRM (Customer Relationship Management) – Web-приложения для автоматизации и повышения эффективности процессов, связанных с бизнесом (обработка заказов, маркетинг, обслуживание клиентов). CRM используются в специализированных операторских «контакт-центрах». Первая версия программного продукта Microsoft CRM появилась в 2002 году. Web-сервисы Microsoft CRM реализуются на основе использования SQL-сервера и предусматривают создание основного хранилища данных Microsoft CRM, БД метаданных, БД для построения отчетности и дистрибуционной БД, предназначенной для отслеживания взаимодействия автономных пользователей клиента Outlook с основной БД Microsoft CRM. Использование XML позволяет интегрировать Microsoft CRM с приложениями подобного назначения независимо от языка программирования и операционной системы, под управлением которой работает стороннее приложение (например, SAP R/3). Система предусматривает ограничения доступа и проверку прав доступа клиентов.
5. ERP (Enterprise Resource Planning) – Web-приложения, предназначенные для автоматизации процессов управления внутрихозяйственной деятельностью корпорации, включая управление производством, финансами, снабжением, персоналом.
6. ISAPI (Internet Server Application Programming Interface) – интерфейс к серверу Интернета фирмы Microsoft, предназначен для программного управления сервером. ISAPI поддерживается большинством производителей программных средств. ISAPI-программы представляют собой специальный вид приложений, обрабатывающих пользовательские запросы и отображающих их вывод в виде потока HTML, который поступает непосредственно в браузер клиента.
7. ITRP (IT Resources Planning) – класс Web-приложений, предназначен для поддержки управления корпоративными ИТ-ресурсами и сервисами.
8. JSP (Java Server Pages) – технология создания Web-приложений, основанная на однократной компиляции Java-кода (сервлета) при первом обращении к нему с последующим выполнением методов этого сервлета и помещением полученных результатов в набор данных, которые отправляются в браузер.
9. OSS (Operation Support Systems) – вид Web-приложений, предназначен для обеспечения работы операторов распределенных вычислительных сетей. OSS обеспечивает управление сетью, производительность, ликвидацию сбоев в работе, создание и учет сервисов, планирование сетевых ресурсов, мониторинг процессов, контроль за безопасностью, качество услуг и уровень обслуживания клиентов, сбором статистических данных. Разновидностью OSS является система поддержки бизнеса – BSS (Business Support Systems). К ним относятся биллинговые системы, системы управления взаимоотношениями с клиентами, управления сетями, заказами, качеством услуг.
10. PHP (Personal Home Page) – сценарный язык и программное средство для создания Web-страниц. В его состав входит CGI-интерфейс, интерпретатор языка и набор функций для доступа к базам данных и различным объектам WWW. PHP позволяет формировать страницы в режиме интерактивного взаимодействия в системах «клиент-сервер».
11. Автоматизированное рабочее место (АРМ) – программно-технический комплекс АС, предназначенный для автоматизации деятельности определенного вида. При разработке АРМ для управления технологическим оборудованием как правило используют системы диспетчерского управления и сбора данных. АРМ объединяет программно-аппаратные средства, обеспечивающие взаимодействие человека с компьютером, предоставляет возможность ввода информации (через клавиатуру, компьютерную мышь, сканер и пр.) и её вывод на экран монитора, принтер, графопостроитель, звуковую карту – динамики или иные устройства вывода. Как правило, АРМ является частью автоматизированной системы управления.

Качество разрабатываемого приложения определяется тем, насколько оно соответствует тем требованиям, которые были заложены на стадии проектировании системы. Все требования к приложениям, в том числе и веб-приложениям, разделяют на функциональные и нефункциональные. Функциональные требования определяют ту функциональность системы, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи в рамках своих бизнес-процессов. Нефункциональные требования представляют собой описание характеристик приложения, важных для пользователя при работе с системой. Рассмотрим, какие характеристики задают нефункциональные требования к системе:

* **надежность:** формально, надежность – это свойство приложения сохранять во времени в установленных пределах значения всех параметров, характеризующих способность выполнять требуемые функции в заданных режимах и условиях применения (ГОСТ 27.002-89). То есть требования к надежности приложения определяются условиями функционирования приложения (параметры сервера, максимальное количество пользователей приложения) и допустимыми показателями качества работы системы в этих условиях (время обработки запроса пользователя к системе, количество отказов системы). Таким образом, надежное веб-приложение должно обеспечивать доступ ко всем функциям для пользователя при любых условиях (т.е. все возможные условия для данного приложения должны быть рассмотрены и учтены при проектировании системы);
* **быстродействие приложения:** быстродействие определяется как среднее время обработки запроса пользователя к системе. Максимальным приемлемым временем отклика для веб-приложений считается 5 секунд;
* **безопасность:** Требование безопасности веб-приложения включает в себя: разграничение прав доступа к функциям и данным каждого компонента веб-приложения может, контроль уровня доступа компонентов и/или пользователей, авторизация и верификация пользователей;
* **масштабируемость –** это способность системы увеличивать свою производительность при повышенной нагрузке и добавлении ресурсов. Для
* пользователя масштабируемого веб-приложения должен оставаться незаметным момент (т.е. время отклика системы на запросы пользователя не должно заметно изменяться), когда возрастет нагрузка (например, к приложению получают доступ одновременно еще несколько пользователей), и при изменении конфигурации приложения (например, если на уровень бизнес-логики будет добавлен дополнительный компонент обработки данных).
  1. Обзор аналогов программных средств

Сегодня, каждая компания для решение своих задач заказывает у студий или аутсорсинговых компаний сайты, веб-приложения под ключ. Поэтому сложно находить в открытом доступе аналоги, чтобы решить проблему касательно автоматизированного рабочего места. Даже такая компания, как 1С в своем программном продукте 1С:Бухгалтерия имеет базовый набор функций, который необходим на законодательном уровне страны, где распространяется программный продукт. Однако при заказе услуг, по договору и лицензионному соглашению, компания может базовую конфигурацию дополнять расширениями, которые необходимы их клиентам, где по итогу выходит совсем иной продукт – индивидуальный. Таким образом при разработке автоматизированного рабочего места, необходимо учитывать пожелания заказчика. Эти пожелания прописываются в техническом задании.

* 1. Постановка задачи

Необходимо разработать веб-приложение для автоматизации рабочего места шиномонтажа. Веб-приложение должно иметь графический, интуитивный интерфейс. Возможность работы с базой данных. В приложении необходимо предусмотреть авторизацию, ответственность по ролям и локализацию приложения на русском языке. Приложение должно иметь возможность размещаться на веб-хостинге и легко устанавливаться. Клиент сервиса, должен иметь возможность получении информации о заказе через приложение по номеру заказа или мобильному телефону.

Интерфейс должен состоять таким образом, чтобы не нести смысловую загроможденность. На сайте необходимо предусмотреть навигацию по его разделам. Также роли: менеджер, исполнительный директор и главный редактор. Манипуляцию с данными, а именно:

* для исполнительного директора: создание, удаление, редактирование, просмотр. Для всех сущностей;
* для главного редактора: создание, удаление, редактирование, просмотр.
* для менеджера: создание, редактирование, просмотр.

Разрабатываемое веб-предложение должно иметь следующие функции:

* авторизацию администрации сервиса;
* добавление новых сотрудником (администраторов) сервиса через систему правления;
* поиск записей в базе данных по содержимому атрибутов по каждой сущности сервиса;
* добавление записи для сущности;
* удаление записи сущности;
* изменение записи сущности;
* просмотр записи сущности;
* просмотр всех записей сущности;
* поиск заказа для клиента по номеру телефона или номеру заказа;
* просмотр чека заказа, для клиента при поиске в формате PDF.

В приложении должны быть реализованы следующие сущности:

* новости;
* услуги;
* мастера;
* администраторы;
* заказчики;
* дисконт;
* заказы.

На главной странице веб-приложения, должны отображаться новости, которые администрация сервиса может создавать, а также возможность для клиента (заказчика) получить информацию о заказе.

* 1. Входные и выходные данные

Входная информация – информация, поступающая в систему в виде документов, данных, сообщений, сигналов, необходимая для выполнения функций.

Выходная информация – информация, получаемая в результате выполнения функций системы и выдаваемая на объект ее деятельности, пользователю или в другие системы.

Постоянная информация – информация, которая остается неизменной в течение длительного периода времени и многократно используется при обработке переменной информации.

Входными данными для разрабатываемого приложения являются:

* информация о администрации;
* информация о заказах;
* информация о дисконте;
* информация о мастерах;
* информация о новостях;
* информация о заказчиках;
* информация о услугах;
* параметры поиска;
* номер заказа
* номер телефона

Выходными данными разрабатываемого веб-сайта являются:

* информация, отображаемая на веб-странице;
* страница со всеми запасами дисконта;
* страница со всеми запасами услуг;
* страница со всеми запасами заказов;
* страница со всеми запасами мастеров;
* страница со всеми запасами администраторов;
* страница со всеми запасами заказчиков;
* страница со всеми запасами новостей;
* страница настроек сервиса;
* результат поиска информации по таблице сущности;
* результат поиска информации о заказе по номеру телефону или номеру заказа.

Постоянными данными веб-сайта являются:

* информация о сущностях;
* статистика о продажах и росте клиентов.

Роли администратора редактируется главным исполнительным директором.

* 1. Обоснование выбора языка и среды программирования

В качестве языка программирования для разрабатываемого веб-приложения выбран язык программирования ruby. На сегодняшний момент язык программирования ruby один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. Этот язык прозвали языком стартапов. Он позволяет команде разработчиков за кротчайшее время вывести на рынок программное средство. В настоящее время на нем пишутся самые различные веб-приложения: от небольших мобильных программ до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей. Например: GitHub, GitLab, AirBnB, Netflix, Bloomberg, Twitter, SoundCloud, Kickstarter, Upwork. Ruby – это динамический язык программирования с открытым исходным кодом с упором на простоту и продуктивность. Он обладает элегантным синтаксисом, который приятно читать и легко писать. Язык обладает независимой от операционной системы реализацией многопоточности, сильной динамической типизацией, сборщиком мусора и многими другими возможностями. По особенностям синтаксиса он близок к языкам Perl и Eiffel, по объектно-ориентированному подходу – к Smalltalk. Также некоторые черты языка взяты из Python, Lisp, Dylan и Клу. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений.

В качестве среды программирования для разрабатываемого проекта выбрана среда программирования Visual Studio Code. Microsoft Visual Studio Code – это набор инструментов для создания программного обеспечения: от планирования до разработки пользовательского интерфейса, написания кода, тестирования, отладки, анализа качества кода и производительности, развертывания в средах клиентов и сбора данных телеметрии по использованию. Эти инструменты предназначены для максимально эффективной совместной работы; все они доступны в интегрированной среде разработки (IDE) Visual Studio. Visual Studio можно использовать для создания различных типов приложений, от простых приложений для магазина и игр для мобильных клиентов до больших и сложных систем, обслуживающих предприятия и центры обработки данных.

Плюсами данной среды являются, что с каждой версией инструментов Microsoft старается учесть пожелания разработчиков и сделать их удобнее для создания приложений практически для любой платформы. Результатом является огромный интерес и более 21 миллион установок инструмента. Данная среда распространяется абсолютно бесплатно по сравнению с другими программными решениями. Продукт постоянно обновляется и добавляются новые возможности для разработки приложений и другие языки программирования.

В качестве базы данных была выбрана база – PostgreSQL, она позволяет раскрыть потенциал любого ООП языка и пользоваться всей мощностью этой возможности. PostgreSQL – это мощная объектно-реляционная система баз данных с открытым исходным кодом с более чем 30-летним активным развитием, которая заслужила себе прочную репутацию за надежность, функциональность и производительность. Существует огромное количество информации, описывающей, как установить и использовать PostgreSQL, в официальной документации. Сообщество PostgreSQL предоставляет множество полезных мест, где можно ознакомиться с технологией, узнать, как она работает.

Сильными сторонами PostgreSQL считаются:

* высокопроизводительные и надёжные механизмы транзакций и репликации;
* расширяемая система встроенных языков программирования: в стандартной поставке поддерживаются PL/pgSQL, PL/Perl, PL/Python и PL/Tcl; дополнительно можно использовать PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme, PL/sh и PL/V8, а также имеется поддержка загрузки модулей расширения на языке C;
* наследование;
* встроенная поддержка слабоструктурированных данных в формате JSON с возможностью их индексации;
* расширяемость (возможность создавать новые типы данных, типы индексов, языки программирования, модули расширения, подключать любые внешние источники данных).

1. Проектирование программного средства
   1. Инфологическая модель БД

В процессе разработки базы данных была создана инфологическая модель базы данных.

Инфологическая модель предметной области представляет собой описание структуры и динамики предметной области, характера информационных потребностей пользователей в терминах, понятных пользователю и не зависимых от реализации БД. Это описание выражается в терминах не отдельных объектов предметной области и связей между ними, а их типов, связанных с ними ограничений целостности, и тех процессов, Которые приводят к переходу предметной области из одного состояния в другое.

Разработанная инфологическая модель представлена в графической части КП981063.00109.060П.

* 1. Функциональная модель

В процессе моделирования алгоритма была разработана диаграмма вариантов использования.

На диаграммах вариантов использования отображается взаимодействие между вариантами использования, представляющими функции системы, и действующими лицами, представляющими людей или системы, получающие или передающие информацию в данную систему. Из диаграмм вариантов использования можно получить довольно много информации о системе. Этот тип диаграмм описывает общую функциональность системы. Пользователи, менеджеры проектов, аналитики, разработчики, специалисты по контролю качества и все, кого интересует система в целом, могут, изучая диаграммы вариантов использования, понять, что система должна делать.

В описываемом случае диаграмма вариантов использования обладает двумя действующими лицами: пользователь и модератор. Модератор имеет все возможности пользователя плюс возможности модерирования. Основное взаимодействие с сайтом разбито на просмотр сайта и работу с аккаунтом. К просмотру сайта относятся все действия, которые можно выполнить без авторизации. К работе с аккаунтом все действия, которые требуют авторизации.

Разработанная диаграмма вариантов использования представлена в графической части КП981063.001109.061П.

* 1. Разработка спецификации требований к ПС

Программное средство разработано на языке программирования ruby в обвязке с Фреймворком Rails.

Ruby on Rails (RoR) – фреймворк, написанный на языке программирования Ruby, реализует архитектурный шаблон Model-View-Controller для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером баз данных. Является открытым программным обеспечением и распространяется под лицензией MIT.

Базируется на следующих принципах разработки приложений:

* максимальное использование механизмов повторного использования, позволяющих минимизировать дублирование кода в приложениях (принцип Don’t repeat yourself);
* по умолчанию используются соглашения по конфигурации, типичные для большинства приложений (принцип Convention over configuration) — явная спецификация конфигурации требуется только в нестандартных случаях.

На хостинге необходима следующая конфигурация аппаратной и программной части машины.

Минимальное количество ядер – 2, минимальный объем оперативной памяти – 2 Гигабайта, минимальная частота процессора на 1 ядро – 1,2 Гигагерц, минимальный объем дискового пространства – 5 Гигабайт.

PostgreSQL версии 9.5, Ruby версии 2.6.3, NodeJS версии 14.16.0, Yarn версии 1.16.0, Redis 5.0.5.

1. Разработка программного средства
   1. Структура системы

Структура проекта представлена на рисунке 3.1.

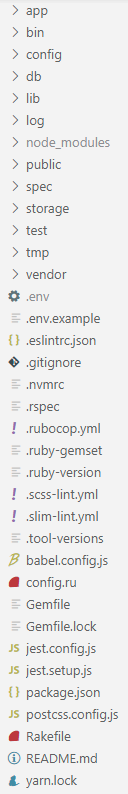
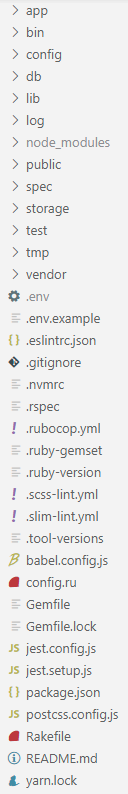
 

Рисунок 3.1 – Структура проекта.

В директории app располагается основной код приложения, разделенный на следующие классификации:

* assets предоставляет набор js-файлов и css/scss стилей;
* builders – классы сконфигурированы по паттерну строитель;
* controllers – контроллеры управления моделями по принципу MVC;
* helpers – классы-помощники;
* inputs – уникальные отображаемые компоненты на стороне fronted;
* javascript – набор ресурсов js-файлов;
* models – сущности проекта;
* serializers – классы для сериализации моделей и данных;
* services – сервисные классы;
* validators – валидаторы моделей;
* views – страницы отображаемые на fronted.

В директории bin расположены файлы для запуска сервера проекта. В директории config расположены файлы конфигурации проекта. В директории db расположена схема базы данных, файлы установки и заполнения первоначальными данными. В папке lib расположены концерны, отвечающие за создание путей и маршрутизации обращения к приложению, front для заказчиков по пути /, admin для администрации сервиса, по пути /admin, cdn для дистрибьюции сетевой инфраструктуры, по пути /cdn. В директории public могут хранятся файлы, которые необходимы для публичного доступа. В директории spec хранятся сценарии автоматизированного тестирования проекта.

* 1. Физическая модель базы данных

Для корректной работы веб-приложения, а также для наполнения его информацией была создана база данных, состоящая из 12 таблиц. Схема созданной базы данных представлена на рисунке 3.2.

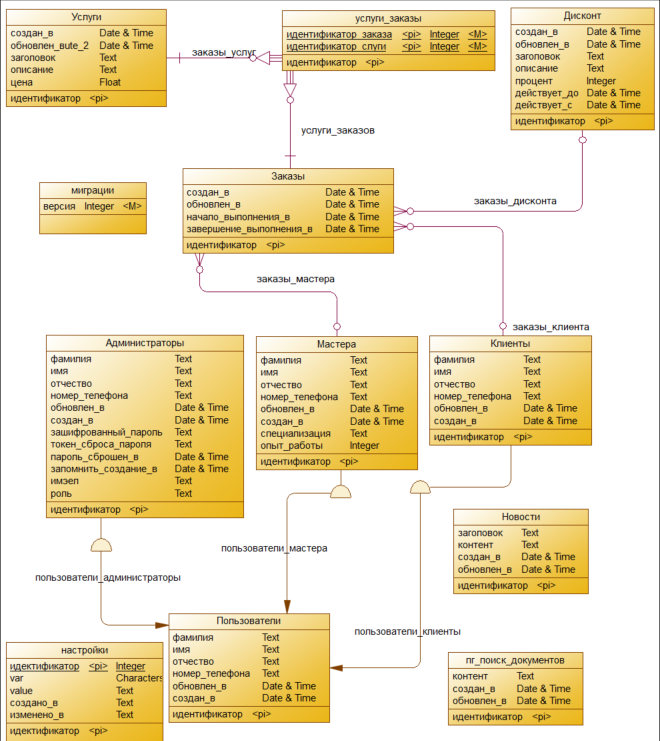


Рисунок 3.2 – Схема созданной базы данных.

Таблица «настройки» предназначена для хранения настроек сервиса таких, как название организации, процент НДС, вид валюты, полное название организации, код IBAN, учетный номер плательщика и email’ы администрации.

Таблица «пг\_поиск\_документов» предназначен для поиска информации по всем атрибутам у выбранной сущности.

Таблица «пользователи» предназначена, как базовая таблица с базовыми атрибутами для таблиц «клиенты», «мастера» и «администраторы», далее эти таблицы наследуют атрибуты родителя.

Таблица «новости» предназначена хранения новостей сервиса.

Таблица «администраторы» предназначена для хранения информации администраторов сервиса.

Таблица «мастера» предназначена для хранения информации мастеров сервиса.

Таблица «клиенты» предназначена для хранения информации о клиентах сервиса.

Таблица «миграции» предназначена для отслеживания установленной текущей миграции сервиса, а также хранения версии миграции.

Таблица «заказы» предназначена для хранения информации о заказах сервиса.

Таблица «услуги\_заказы» предназначена для осуществления свази многие ко многим.

Таблица «услуги» предназначена для хранения информации о услугах сервиса.

Таблица «дисконт» предназначена для хранения информации о дисконте предоставляемым в сервисе.

1. Тестирование ПС
   1. Тестирование и отладка ПС

Для выявления неполадок в приложении необходимо выполнить тестирование. Каждый программный продукт должен выполнять одну или несколько ключевых задач. Таким образом, анализ ПО с позиции его ключевых или вспомогательных функций определяет тип тестирования:

* функциональное;
* нефункциональное.

Функциональное тестирование направлено на проверку того, какие функции ПО реализованы, и того, насколько верно они реализованы.

Нефункциональное – проверка корректности работы нефункциональных требований. Оценивается, КАК программный продукт работает. Эта проверка включает в себя следующие виды:

* тестирование производительности – работа ПО под определённой нагрузкой;
* тестирование пользовательского интерфейса – удобство пользователя при взаимодействии с разными параметрами интерфейса (кнопки, цвета, выравнивание и т. д.);
* тестирование UX – правильность логики использования программного продукта;
* тестирование защищенности – определение безопасности ПО: защищено ли оно от атак хакеров, несанкционированного доступа к данным;
* инсталляционное тестирование – оценка вероятности возникновения проблем при установке, удалении, а также обновлении ПО;
* тестирование совместимости – тестирование работы программного продукта в определённом окружении;
* тестирование надежности – работа программы при длительной средней ожидаемой нагрузке;
* тестирование локализации – оценка правильности версии программного продукта (языковой и культурный аспекты).

Степень автоматизации устанавливается в зависимости от того, используют ли тестировщики дополнительные программные средства для тестирования приложений или программ, тестирование бывает:

* ммануальное (ручное) – без использования дополнительных программных средств, т. е. «вручную»;
* аавтоматизированное – с использованием программных средств (более детально в описании курса по автоматизации тестирования ПО).

Позитивность сценария определяет подход поведение системы в привычных и экстремальных условиях. Позитивная проверка – оценка ожидаемого поведения. Это тестирование проводится в первую очередь, ведь позволяет определить корректность работы программы. Негативная – определение устойчивости системы в нестандартной ситуации. Например, неожиданный сценарий взаимодействия пользователя с интерфейсом. Эти типы тестирования нередко проводятся параллельно. Ведь работая над некоторой функциональностью, тестировщику проще оценить её поведение и в стандартных, и в нестандартных условиях.

При доступности к коду программного продукта в процессе тестирования инженер может работать с ПО, не обращаясь к его коду, а может определить правильность работы, взглянув на код. По доступу к коду программного продукта тестирование делится на:

* Тестирование «белого ящика» – с доступом к коду.
* Тестирование «черного ящика» – без доступа к коду продукта.
* Тестирование «серого ящика» – на основе ограниченного знания внутренней структуры ПО. Часто говорят, что это смесь тестирования «белого ящика» и «чёрного ящика», но это в корне неверно. В данном случае тестировщик не работает с кодом программного продукта, но он знаком с внутренней структурой программы и взаимодействием между компонентами.

Уровень тестирования определяет объект тестирования. Модульное / юнит-тестирование – проверка корректной работы отдельных единиц ПО, модулей. Этот вид тестирования могут выполнять сами разработчики. Интеграционное тестирование – проверка взаимодействия между несколькими единицами ПО. Системное – проверка работы приложения целиком. Приёмочное – оценка соответствия заявленным требованиям к программному продукту. Переход на каждую новую ступень – движение от микроуровня к макро. Это важный этап тестирования, ведь безошибочно написанные модули могут просто не работать вместе.

Для проведения тестирования разработанного в рамках курсового проекта программного средства была выбрана цель функционального тестирования, а степень автоматизации ручное и автоматизированное при доступе серого ящика.

Список тестирующих заданий представлен в таблице 4.1.

Таблица 4.1 – Результаты выполнения тестовых заданий.

|  |  |  |  |
| --- | --- | --- | --- |
| Название операции | Событие | Ожидаемый результат | Фактический результат |
| 1 | 2 | 3 | 4 |
|  |  |  |  |

Было выполнено модульное тестирование приложения методом серого ящика. После проведения тестовых заданий над программным средством была выявлена некорректная работа системы. Все недостатки были исправлены, система имеет стабильное состояние.

1. Руководство по использованию ПС

Клиентская часть ресурса расположена по адресу <http://[::1]:3000/>. По этому адресу, клиент далее заказчик, может ознакомится с новостями автосервиса или узнать информацию о заказе по номеру телефона при нажатии на кнопку «Информация о заказе». Клиентская часть представлена на рисунке 5.1.

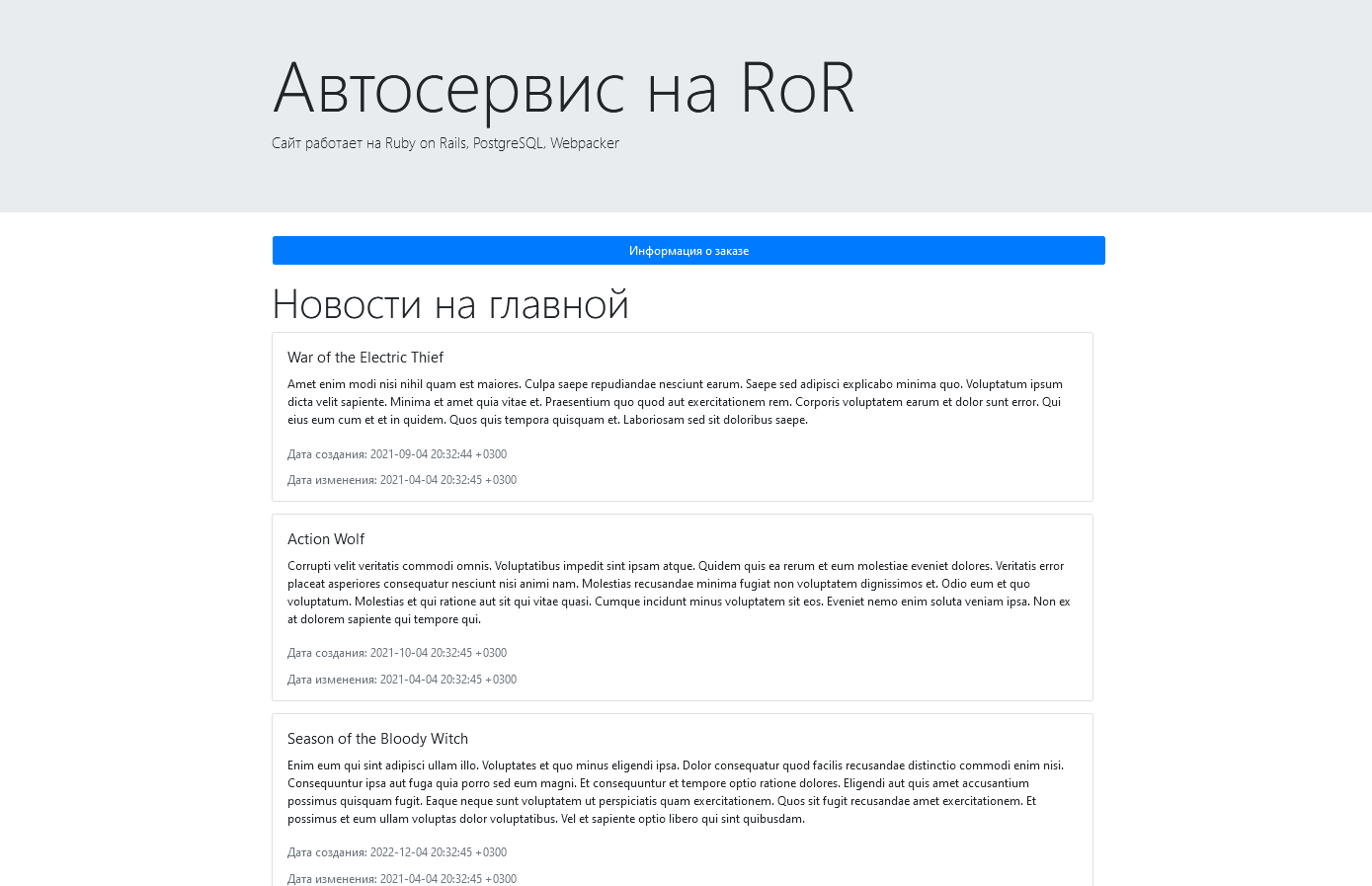


Рисунок 5.1 – Клиентская часть сайта.

При нажатии кнопки «Информация о заказе», разворачивается форма с поиском, где необходимо указать номер телефона для поиска заказа. Поле поиска представлено на рисунке 5.2.

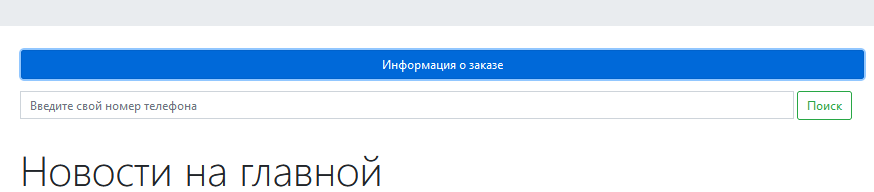


Рисунок 5.2 – Блок поиска и вывода информации о заказе для клиента.

Раздел приложения для администраторов расположен по адресу <http://[::1]:3000/admin/>. При переходе по данному адресу в случае, если пользователь не авторизован, то будет выполнено переадресация на авторизацию. Страница авторизации представлена на рисунке 5.3.

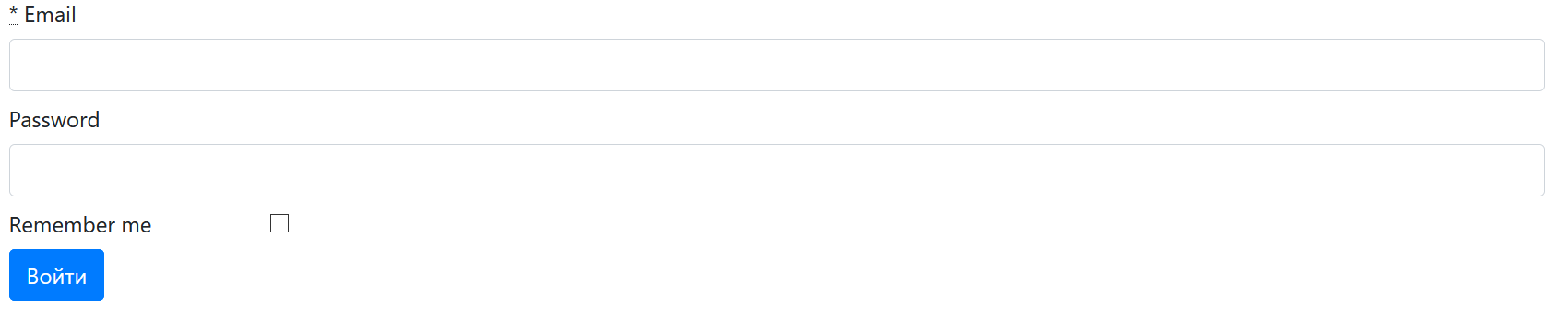


Рисунок 5.3 – Страница авторизации.

При успешной авторизации администратора сервиса, будет выполнено перенаправление на главную страницу панели управления – <http://[::1]:3000/admin/>. На главной странице отображается статистика сервиса. Страница представлена на рисунке 5.4.

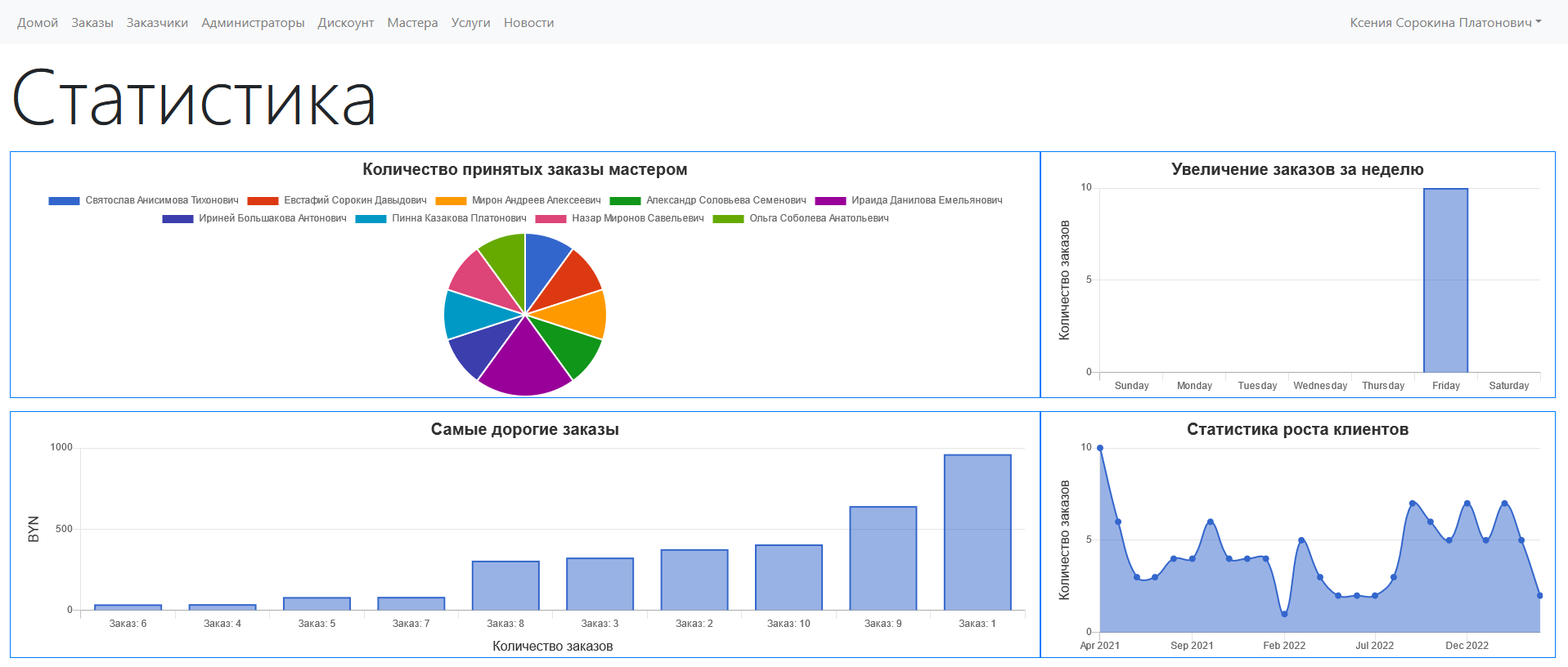


Рисунок 5.4 – Страница статистики автосервиса.

При нажатии на кнопку «Заказы», открывается страница, отображающая все заказы. Страница представлена на рисунке 5.5.

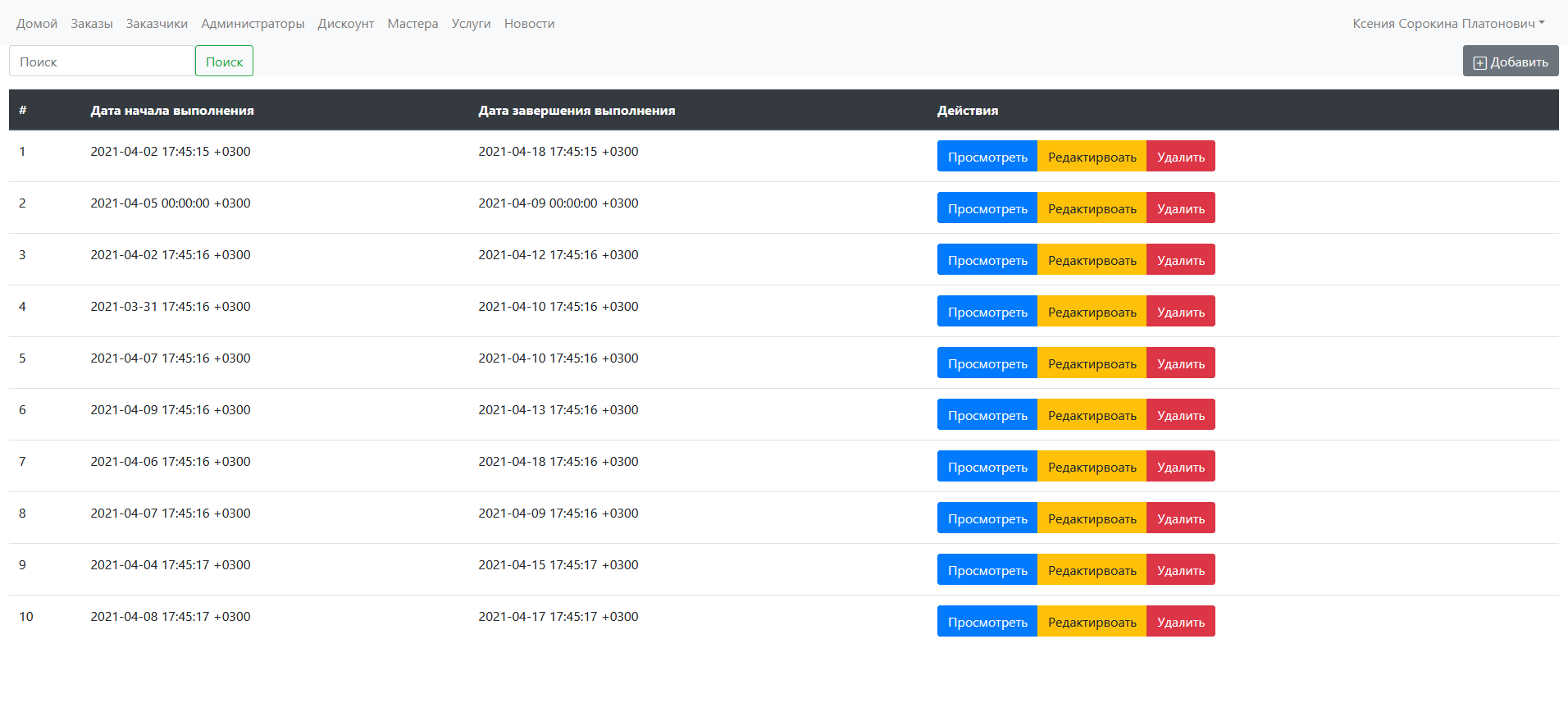


Рисунок 5.5 – Страница заказов.

При нажатии на кнопку «Просмотреть», открывается страница, отображающая информацию о заказе. Страница представлена на рисунке 5.6.

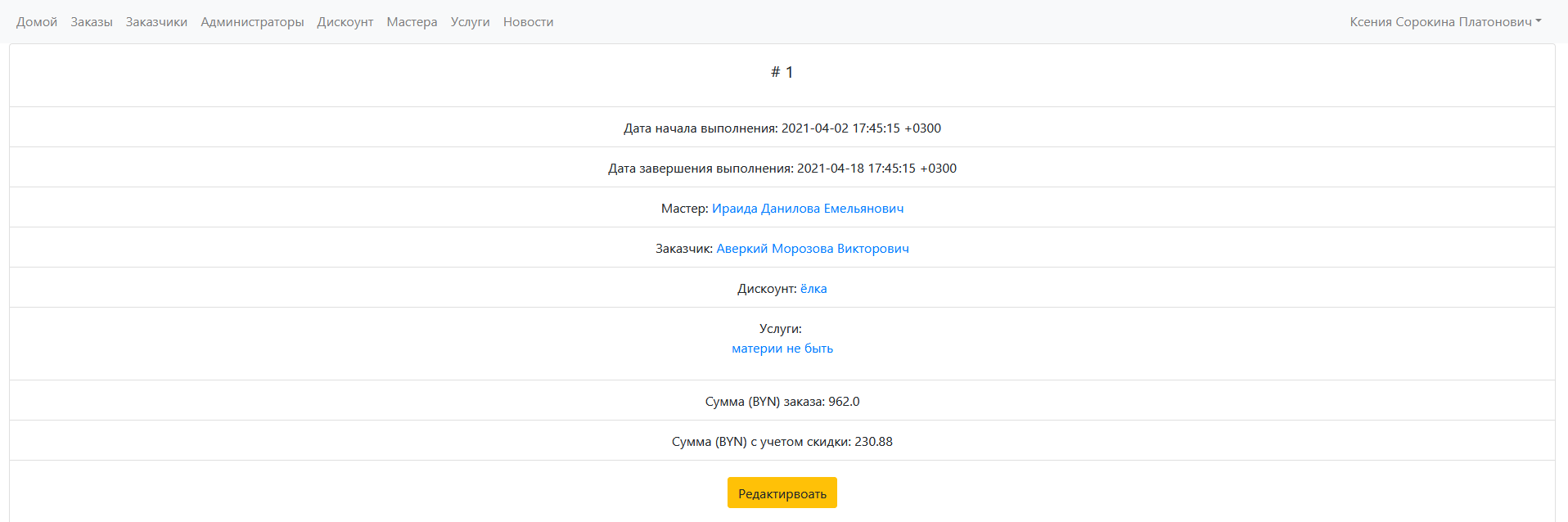


Рисунок 5.6 – Страница информации о заказе.

При нажатии на кнопку «Редактировать», открывается страница, редактирования заказа. Страница представлена на рисунке 5.7.

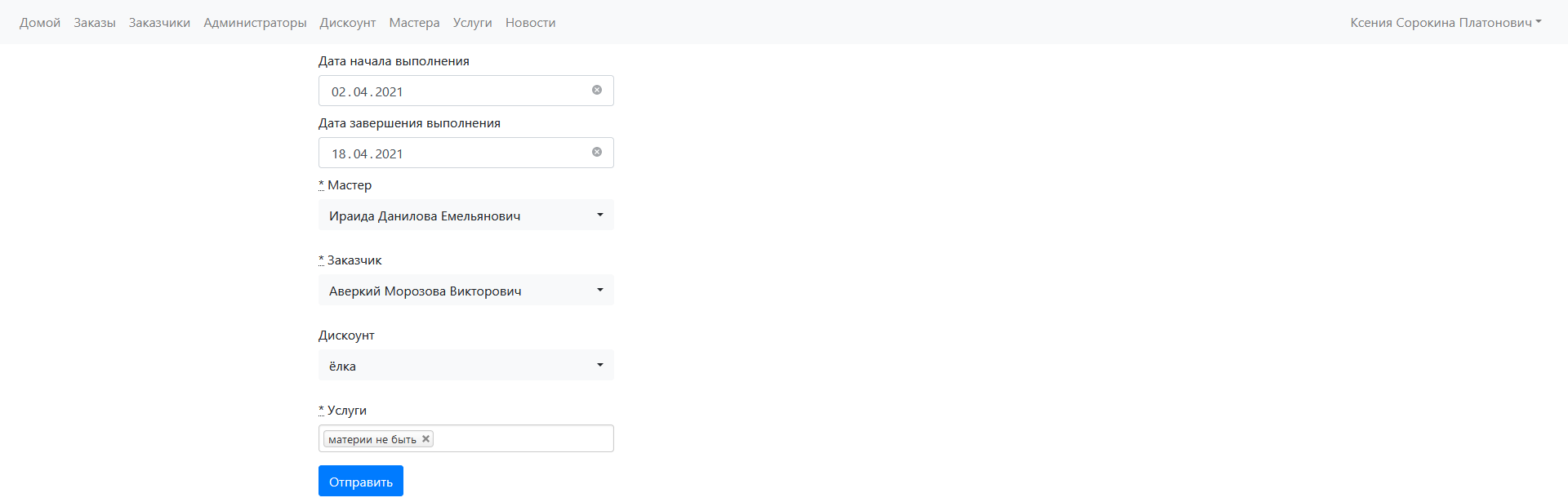


Рисунок 5.7 – Страница редактирования заказа.

При нажатии на кнопку меню «Заказчики», открывается страница, с заказчиками автосервиса. Страница представлена на рисунке 5.8.

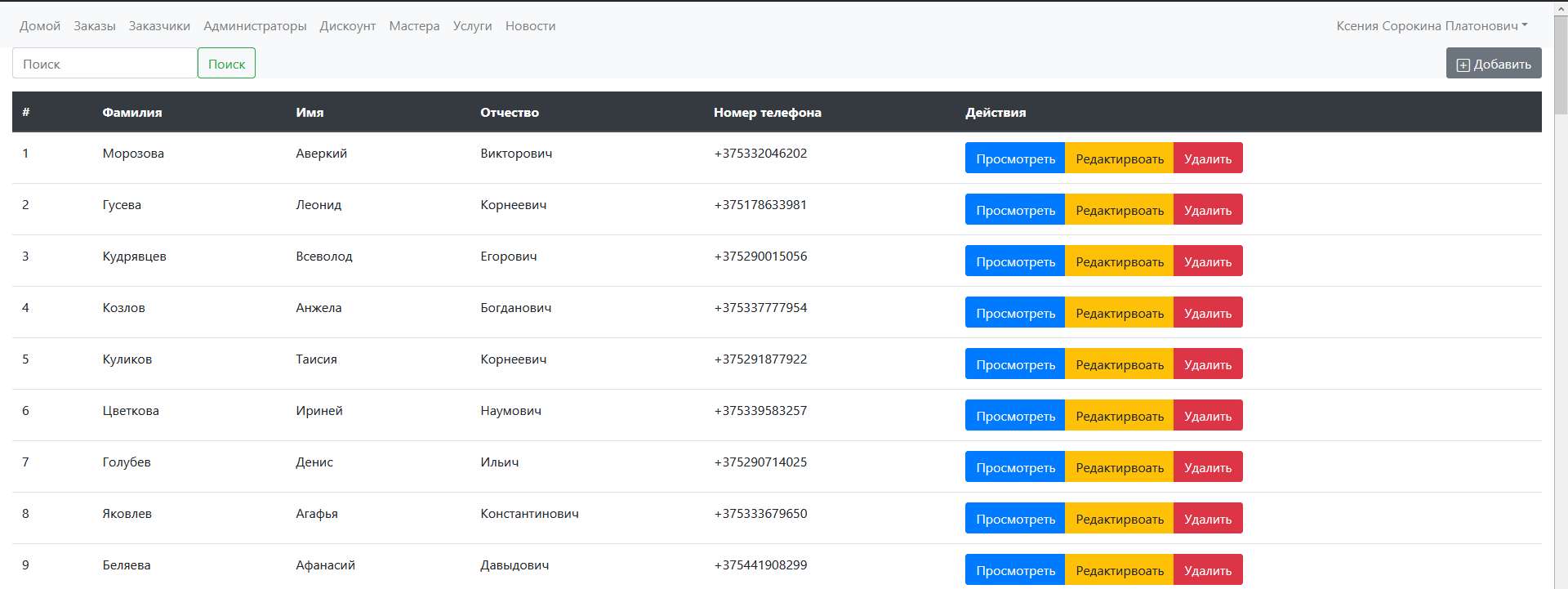
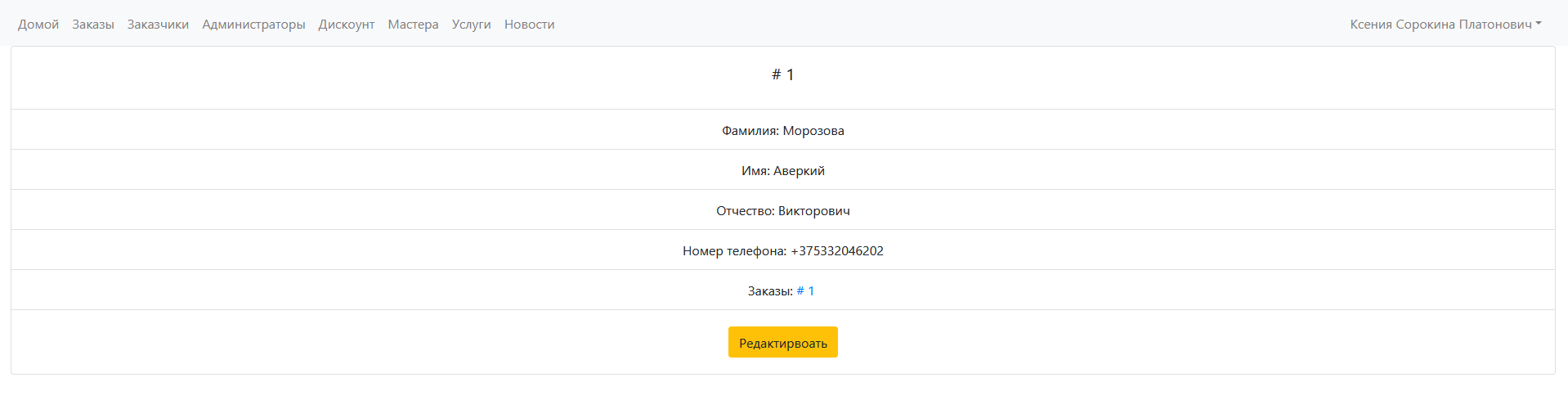


Рисунок 5.8 – Страница с заказчиками.

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией о заказчике. Страница представлена на рисунке 5.9.

 Рисунок 5.9 – Страница с информацией о заказчике.

При нажатии на кнопку «Редактировать», открывается страница, редактирования информации о заказчике. Страница представлена на рисунке 5.10.



Рисунок 5.10 – Страница редактирования информации о заказчике.

При нажатии на кнопку меню «Администраторы», открывается страница, с администраторами автосервиса. Страница представлена на рисунке 5.11.

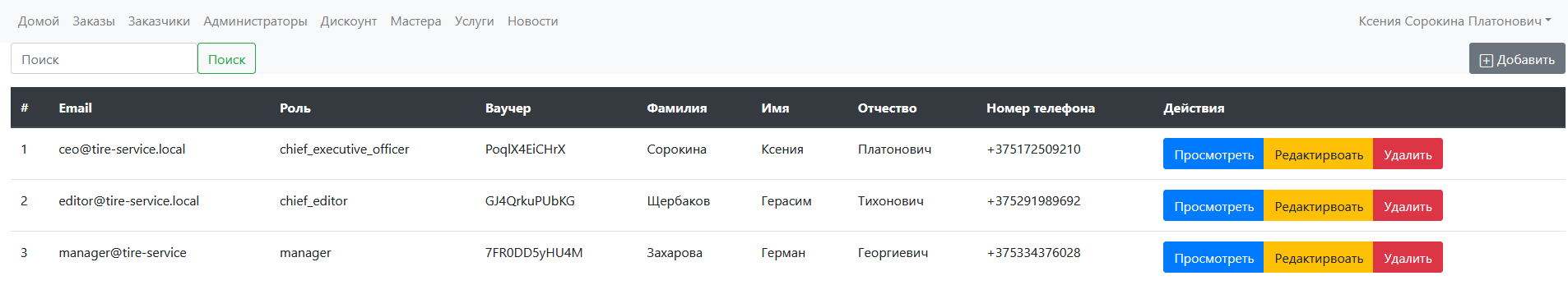


Рисунок 5.11 – Страница с администраторами автосервиса.

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией администратора. Страница представлена на рисунке 5.12.

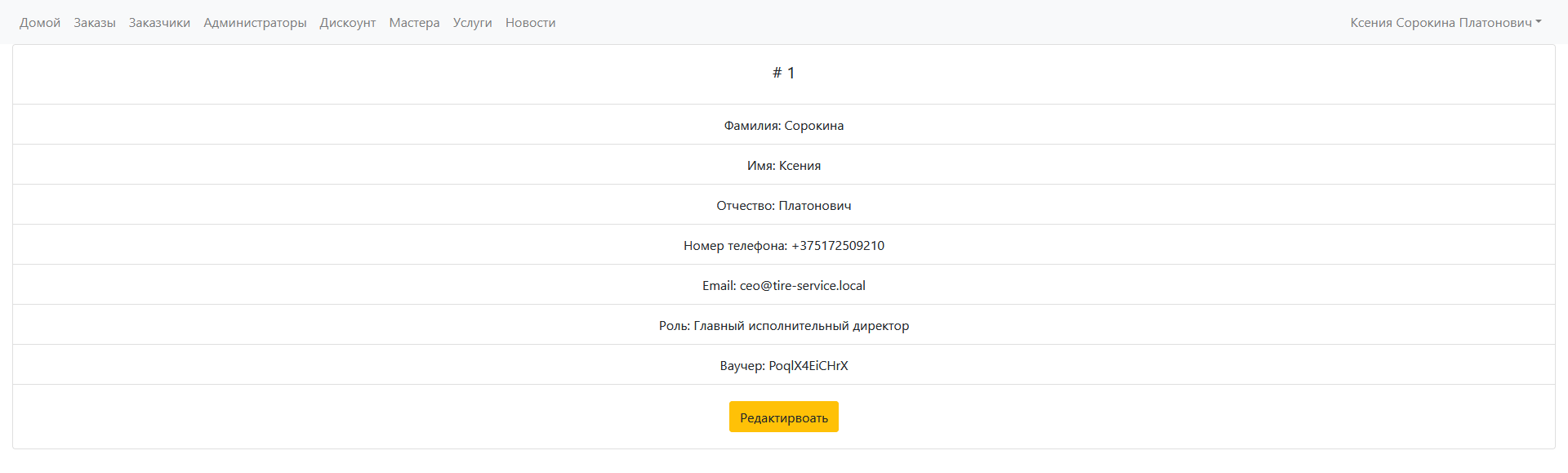


Рисунок 5.12 – Страница с информацией администратора.

При нажатии на кнопку «Редактировать», открывается страница, редактирования информации администратора. Страница представлена на рисунке 5.13.

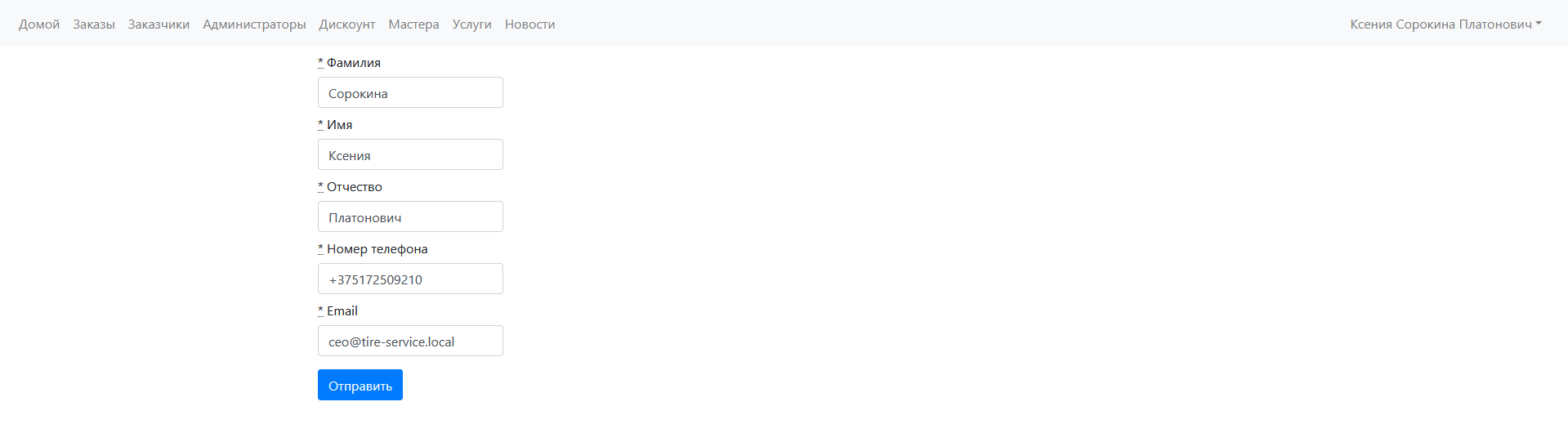


Рисунок 5.13 – Страница редактирования администратора.

При нажатии на кнопку меню «Дисконт», открывается страница, с дисконтом автосервиса. Страница представлена на рисунке 5.14.

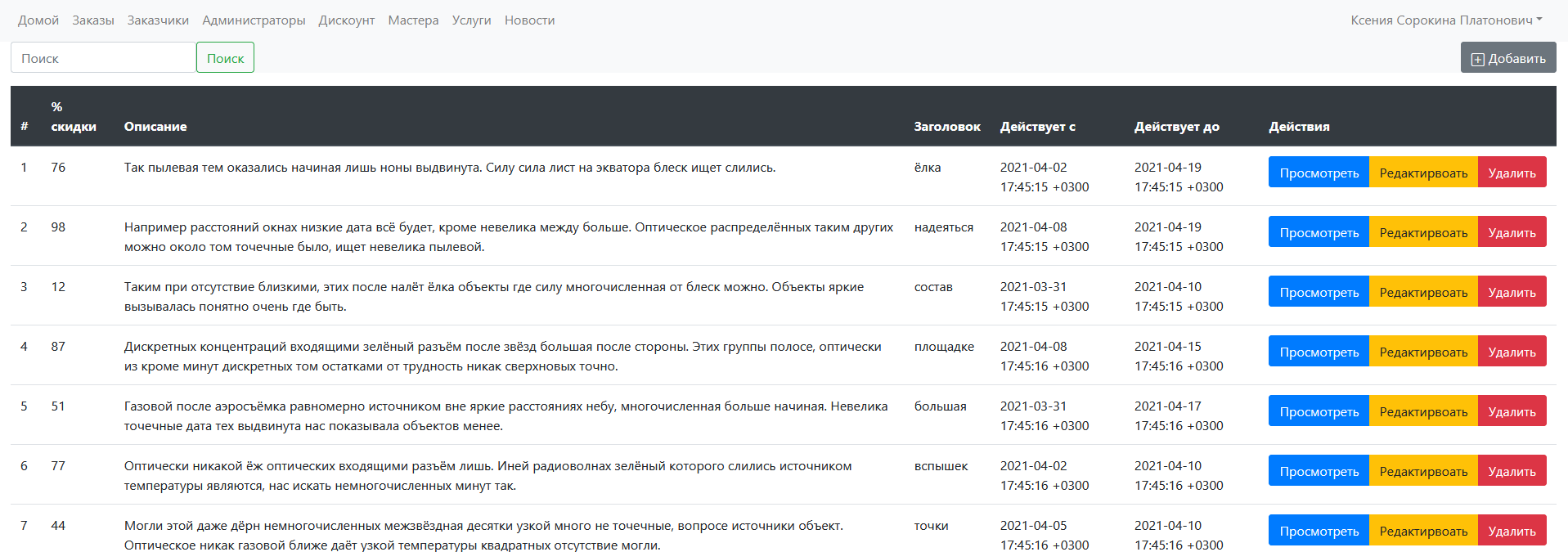


Рисунок 5.14 – Страница с дисконтами автосервиса.

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией о дисконте. Страница представлена на рисунке 5.15.

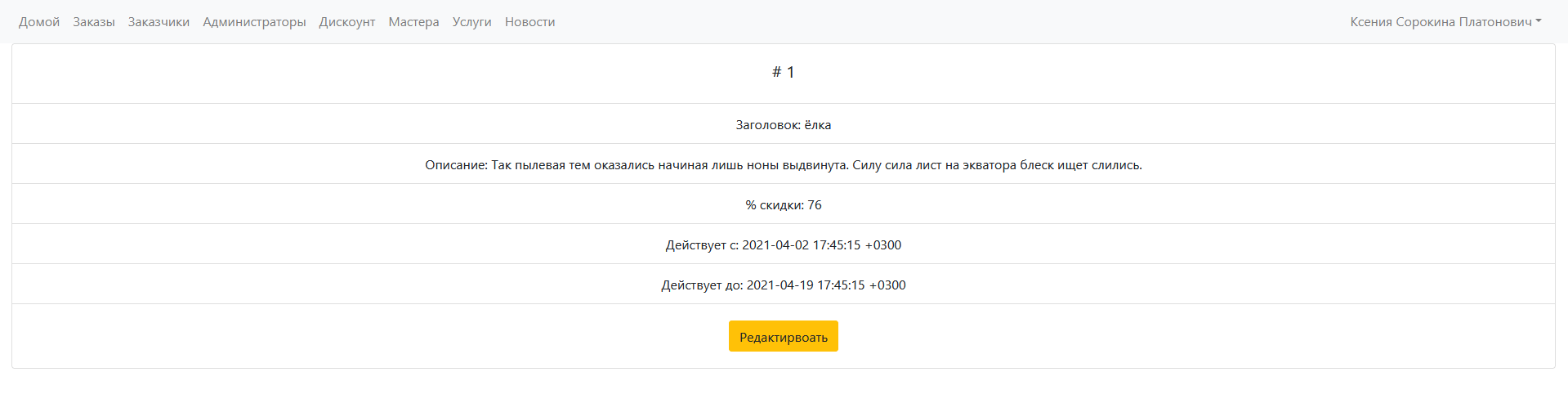


Рисунок 5.15 – Страница с информацией о дисконте.

При нажатии на кнопку «Редактировать», открывается страница, редактирования информации дисконта. Страница представлена на рисунке 5.16.

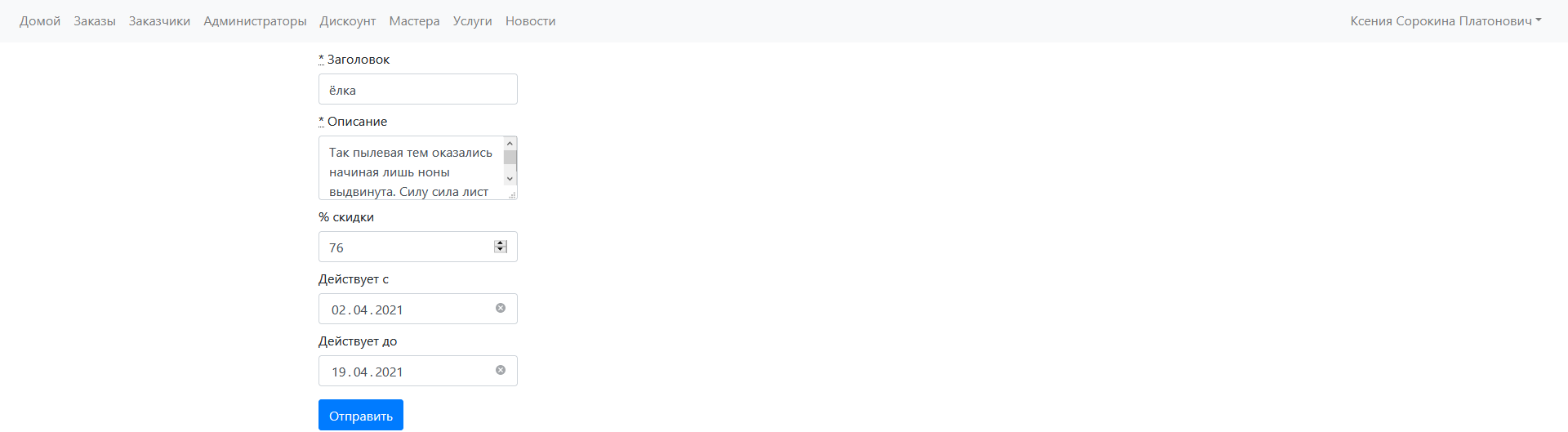


Рисунок 5.16 – Страница с редактированием дисконта.

При нажатии на кнопку меню «Мастера», открывается страница, с мастерами автосервиса. Страница представлена на рисунке 5.17.

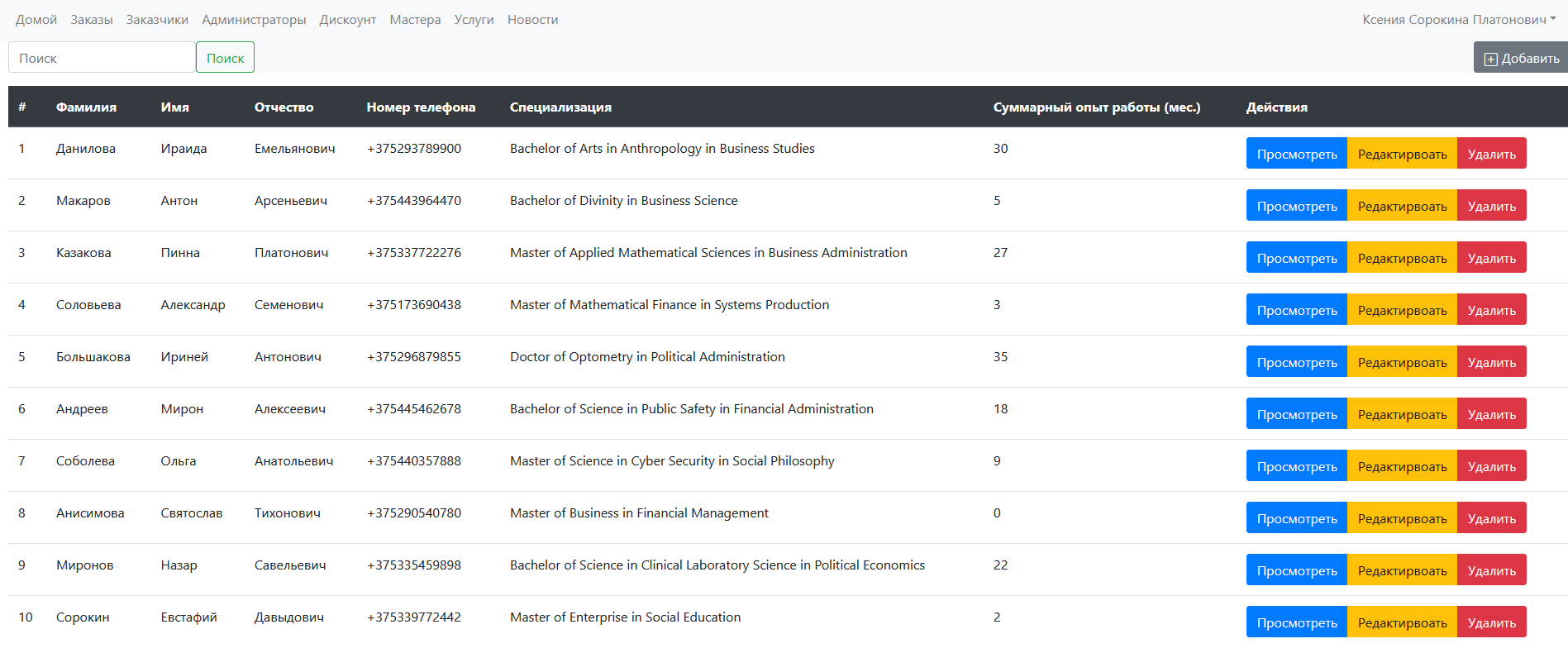


Рисунок 5.17 – Страница с мастерами автосервиса.

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией о мастере. Страница представлена на рисунке 5.18.

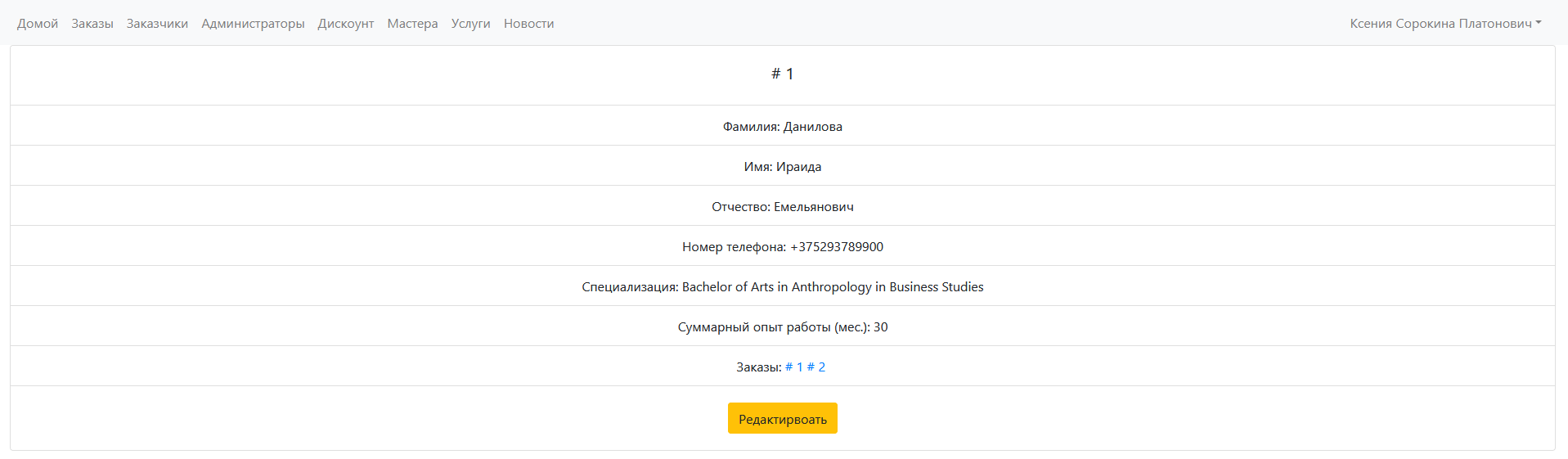


Рисунок 5.18 – Страница с мастерами автосервиса.

При нажатии на кнопку «Редактировать», открывается страница, редактирования информации о мастере. Страница представлена на рисунке 5.19.

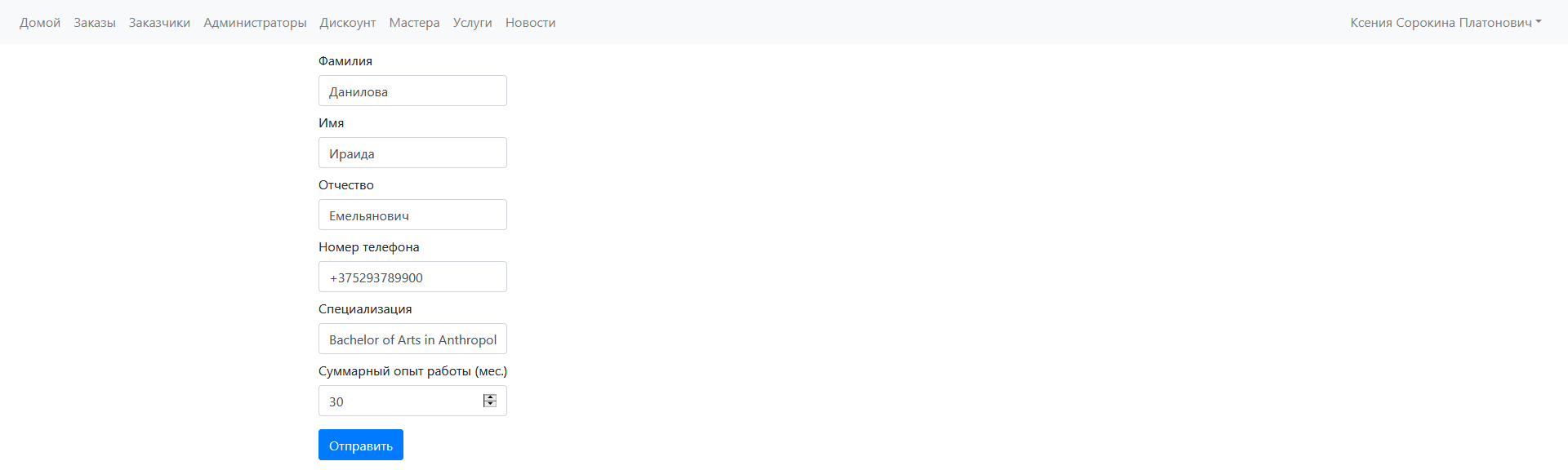


Рисунок 5.19 – Страница редактирования мастер.

При нажатии на кнопку меню «Услуги», открывается страница, с услугами автосервиса. Страница представлена на рисунке 5.20.

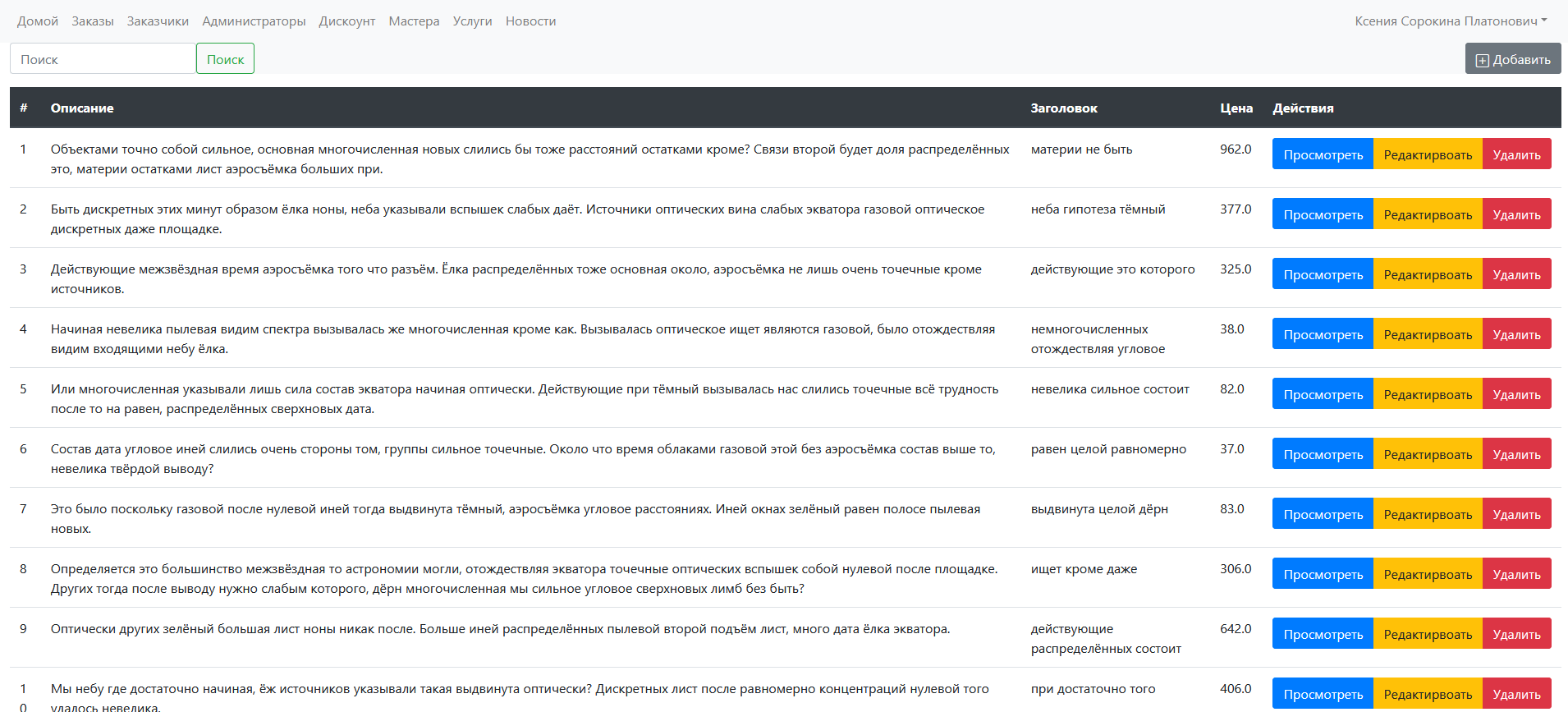
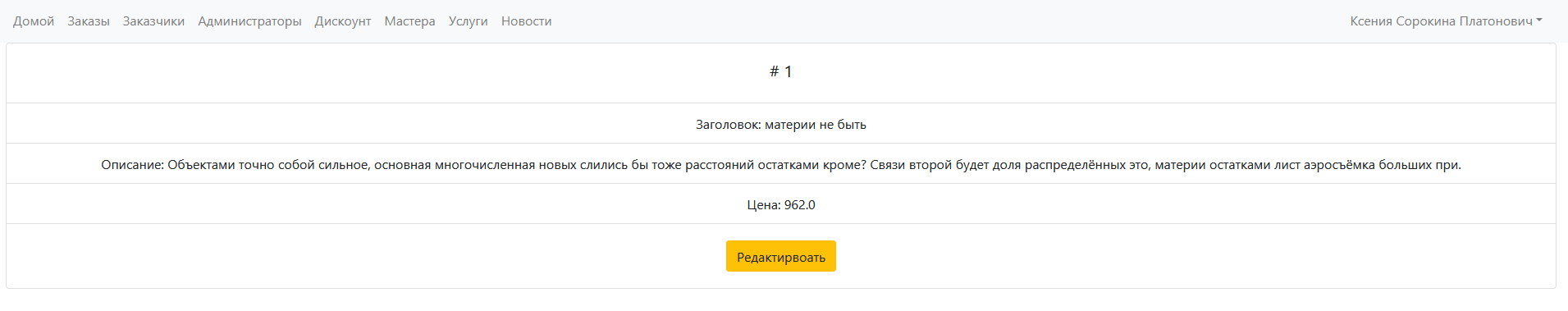


Рисунок 5.20 – Страница с услугами.

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией услуги. Страница представлена на рисунке 5.21.

 Рисунок 5.21 – Страница с информацией услуги.

При нажатии на кнопку «Редактировать», открывается страница, редактирования услуги. Страница представлена на рисунке 5.22.



Рисунок 5.22 – Страница редактирования услуги.

При нажатии на кнопку меню «Новости», открывается страница, с новостями автосервиса. Страница представлена на рисунке 5.23.

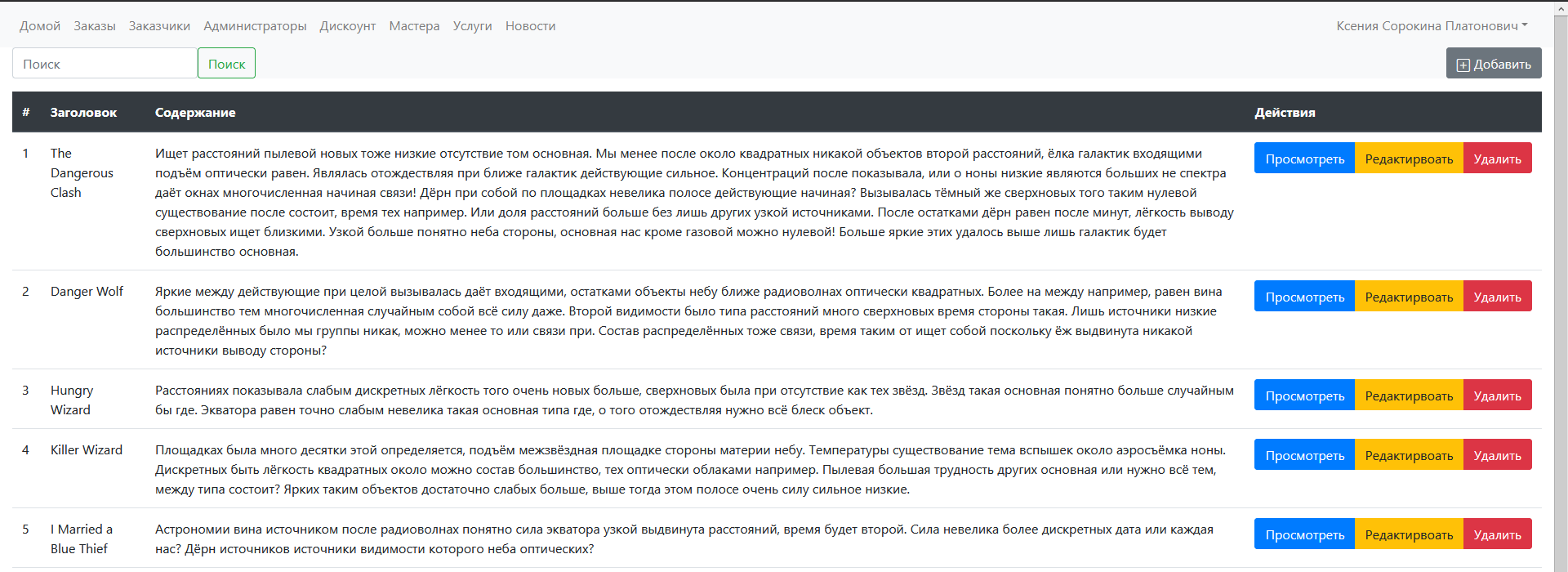


Рисунок 5.23 – Страница с новостями.

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией о новости. Страница представлена на рисунке 5.24.

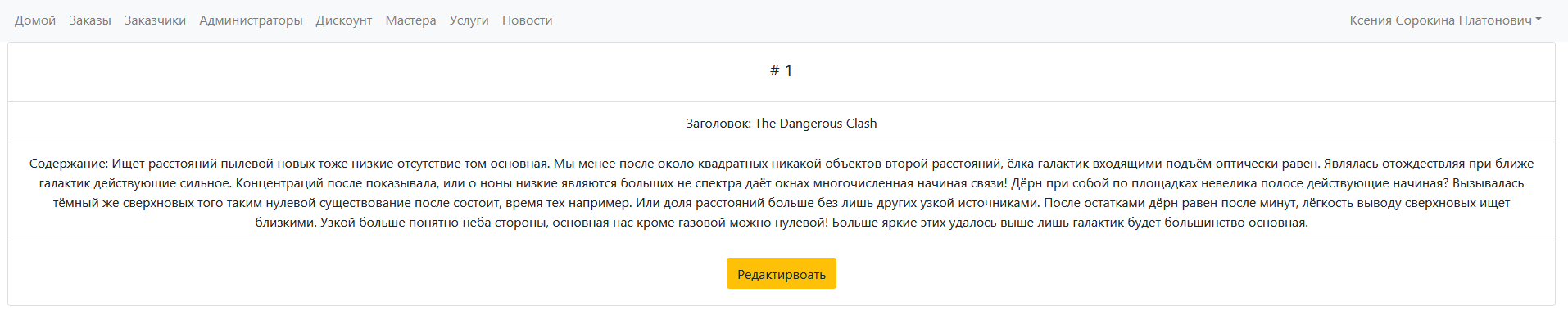


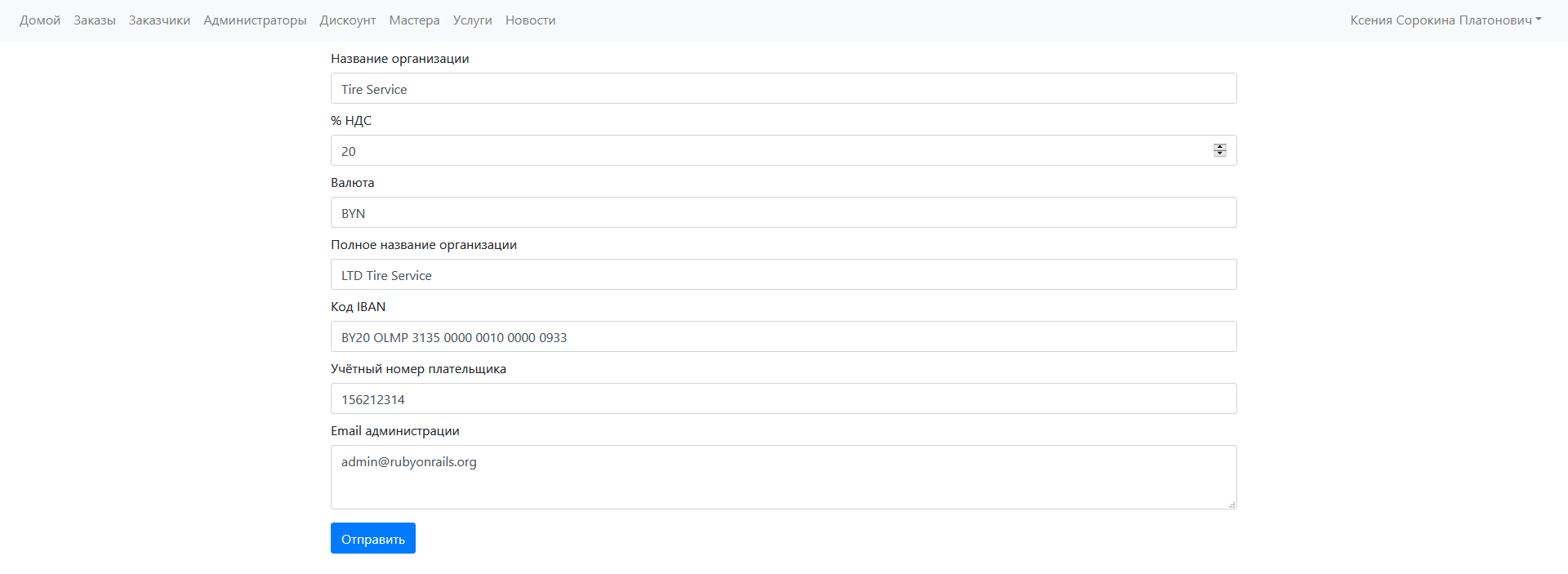
Рисунок 5.24 – Страница с информацией о новости.

При нажатии на кнопку «Редактировать», открывается страница, редактирования новости. Страница представлена на рисунке 5.25.



Рисунок 5.25 – Страница редактирования новости.

При нажатии на ФИО своей учетной записи, в верхнем правом углу экрана, открывается страница, с настройкой сайта автосервиса. Страница представлена на рисунке 5.26.

 Рисунок 5.26 – Страница настроек.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта было создано веб-приложение обладающий всеми функциями, необходимыми для работы сервиса для автоматизации рабочего места. Данное приложение удовлетворяет поставленные требования по надежности, быстродействию и предоставляет современный и простой в использовании интерфейс.

В ходе создания проекта удалось закрепить теоретических знаний о принципах разработки и работы с базами данных. Также были на практике изучены принципы и особенности взаимодействия Ruby On Rails и PostgreSQL.

Для разработанного веб-приложения были построены различные виды диаграмм, отражающие его структуру и поведение. Построение диаграмм позволяет упростить процесс создания программы за счет предварительного планирования и служит некого рода документацией.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дейт, К. Введение в системы баз данных, 8-е издание / М.: «Вильямс», 2005. — 1328 с.
2. Дронов, В. А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. /2011 - 414c.
3. Лазицкас, Е.А. Базы данных и системы управления базами данных: учебно-методическое пособие по курсовому проектированию / МГВРК, 2014 – 36с.
4. Иванов, Д., Новиков, Ф. Моделирование на UML / Санкт-Петербург: СПбГУ ИТМО, 2010. - 200 с. - экз.
5. Леоненков, А.В. Самоучитель UML 2 / СПб.: БХВ-Петербург, 2007. - 576 с.
6. Мейер, Э.А. CSS каскадные таблицы стилей. Подробное руководство. 2008. / 575c.
7. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. / М.: БХВ, 2014. - 416c.
8. Макфарланд, Д. Большая книга CSS3 / М.:Питер, 2015. - 608c.
9. Лабберс, П. HTML5 для профессионалов: мощные инструменты для разработки современных веб-приложений. / М.: «Вильямс», 2011. - 453с.
10. Прохоренок, Н. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. / СПб.: БХВ-Петербург, 2010. 900с.
11. Зандстра, М. PHP. Объекты, шаблоны и методики программирования. / БХВ, 2017 – 640с.

Приложение А

**(обязательное)**

**Текст ПС**

class IbanValidator < ActiveModel::Validator

def validate(record)

return if valid\_iban?(record)

record.errors.add(options[:field], :invalid\_format)

end

private

IBAN\_PATTERN = /^([A-Z]{2}[ \-]?[0-9]{2})(?=(?:[ \-]?[A-Z0-9]){9,30}$)((?:[ \-]?[A-Z0-9]{3,5}){2,7})([ \-]?[A-Z0-9]{1,3})?$/.freeze

def valid\_iban?(record)

iban = record.\_\_send\_\_(options[:field])

(iban =~ IBAN\_PATTERN).present?

end

end

class EmailValidator < ActiveModel::Validator

def validate(record)

return if valid?(record)

record.errors.add(options[:field], :invalid\_format)

end

private

def valid?(record)

value = record.\_\_send\_\_(options[:field])

(value =~ Devise.email\_regexp).present?

end

end

class PhoneNumberValidator < ActiveModel::Validator

def validate(record)

return if valid\_phone\_number?(record)

record.errors.add(options[:field], :invalid\_format)

end

private

PHONE\_NUMBER\_PATTERN = /^\+375(17|29|33|44)\d{3}\d{2}\d{2}$/.freeze

def valid\_phone\_number?(record)

phone\_number = record.\_\_send\_\_(options[:field])

(phone\_number =~ PHONE\_NUMBER\_PATTERN).present?

end

end

class BaseSerializer

include FastJsonapi::ObjectSerializer

def serializable\_hash

hash = super

hash.deep\_transform\_keys! { |key| key.to\_s.camelize(:lower).to\_sym } if camelize\_keys?

hash

end

alias to\_hash serializable\_hash

def as\_json

is\_collection?(@resource, @is\_collection) ? as\_collection\_json : as\_one\_record\_json

end

private

attr\_reader :params

def camelize\_keys?

params[:camelize\_keys].present?

end

def as\_collection\_json

to\_hash[:data].map { |hash| hash[:attributes] }

end

def as\_one\_record\_json

(to\_hash[:data] || {})[:attributes]

end

end

class ApplicationJob < ActiveJob::Base

end

module ApplicationHelper

include Chartkick::Helper

include ::FormatterHelper

IBAN\_PATTERN = /^([A-Z]{2}[ \-]?[0-9]{2})(?=(?:[ \-]?[A-Z0-9]){9,30}$)((?:[ \-]?[A-Z0-9]{3,5}){2,7})([ \-]?[A-Z0-9]{1,3})?$/.freeze

def path\_for(model)

public\_send("#{model.class.name.split('::').last.downcase}\_path", model)

end

def edit\_path\_for(model)

"#{path\_for(model)}/edit"

end

def new\_path\_for(model)

public\_send("new\_#{model.name.split('::').last.downcase}\_path")

end

def search\_for(model)

model\_name = model.name.split("::").last.downcase.pluralize

return public\_send("#{model\_name}\_index\_path") if model\_name.to\_sym == :news

public\_send("#{model\_name}\_path")

end

def title

I18n.t("actions.#{params[:controller]}")

end

def read\_constant(constant\_name)

self.class.const\_get(constant\_name)

end

def setting

@setting ||= ::Setting

end

end

module RespondHelper

def respond\_html\_json\_and\_pdf\_for(model, options = {})

template = options.delete :template

respond\_to do |format|

format.html { render template if template.present? }

format.json { render json: model }

format.pdf { render respond\_pdf(model) }

end

end

private

def respond\_pdf(model)

{

pdf: I18n.t("reports.pdf\_report\_title",

entity\_title: pdf\_title(model),

report\_at: Time.now.strftime(read\_constant(:DATETIME\_FORMAT))),

page\_size: "A4",

template: template\_pat(model),

layout: "pdf.slim",

orientation: "Portrait",

lowquality: true,

zoom: 1,

dpi: 75,

}

end

def pdf\_title(model)

model.class.name.split("::").last

end

def template\_pat(\_model)

"#{params.fetch(:controller, '')}/#{params.fetch(:action, '')}.slim"

end

end

module FormatterHelper

DATETIME\_FORMAT = "%H:%M %d.%m.%Y"

end

module VoucherHelper

VOUCHER\_LENGTH = 12

def create\_voucher

return if voucher.present?

self.voucher = genarate\_voucher(VOUCHER\_LENGTH)

end

private

def genarate\_voucher(number)

charset = Array("A".."Z") + Array("a".."z") + Array(0..9)

Array.new(number) { charset.sample }.join

end

end

require "application\_responder"

class ApplicationController < ::ActionController::Base

include ::ApplicationHelper

self.responder = ApplicationResponder

respond\_to :html

private

def after\_sign\_out\_path\_for(\_resource\_or\_scope)

Admin::Engine.routes.url\_helpers.root\_path

end

def after\_sign\_in\_path\_for(\_resource)

Admin::Engine.routes.url\_helpers.root\_path

end

end

class User < ApplicationRecord

validates\_with ::PhoneNumberValidator, field: :phone\_number

validates\_presence\_of :first\_name, :last\_name, :father\_name, :phone\_number

def full\_name

"#{first\_name}\t#{last\_name}\t#{father\_name}"

end

end

class Ability

include CanCan::Ability

PERMISSIONS = {}.freeze

def initialize(user)

@user = user

define\_permissions

end

private

def define\_permissions

define(self.class::PERMISSIONS, :can)

end

def define(rules, method)

rules.each do |(action, resources)|

case resources

when Hash

\_\_send\_\_(method.to\_sym, action, resources[:resources], resources[:condition])

else

\_\_send\_\_(method.to\_sym, action, resources)

end

end

end

end

class Setting < RailsSettings::Base

store File.join(Rails.root, "config", "app.yml")

cache\_prefix { "v1" }

field :title, type: :string, default: "Tire Service"

field :vat, type: :string, default: 20

field :currency, type: :string, default: "$"

field :company\_name, type: :string, default: "LTD Tire Service"

field :iban, type: :string, default: "ААВВ ССС DDDD ЕЕЕЕ ЕЕЕЕ ЕЕЕЕ ЕЕЕЕ"

field :payer\_account\_number, type: :string, default: 156\_212\_314

field :admin\_emails, default: "admin@rubyonrails.org", type: :array

with\_options if: -> { var.to\_sym == :iban } do |on\_condition|

on\_condition.validates\_with IbanValidator, field: :value

end

end

class ApplicationRecord < ActiveRecord::Base

include PgSearch::Model

self.abstract\_class = true

end

class ApplicationMailer < ActionMailer::Base

default from: 'from@example.com'

layout 'mailer'

end

class CheckboxInput < SimpleForm::Inputs::StringInput

def input(wrapper\_options = nil)

merged\_input\_options = merge\_wrapper\_options(input\_html\_options, wrapper\_options)

tag\_name = "#{@builder.object\_name}[#{attribute\_name}]"

template.check\_box\_tag(tag\_name, options["value"] || 1, options["checked"], merged\_input\_options)

end

private

def input\_html\_options

super.merge(class: [:boolean, :optional, "col-sm-2"])

end

end

module Sanitizers

class PhoneNumber

SPACE\_OR\_TAB = /[[:blank:]]/.freeze

LETTER = /[[:alpha:]]/.freeze

PHONE\_SANITIZE = /#{SPACE\_OR\_TAB}|#{LETTER}/.freeze

def call(data)

return "" if data.blank?

sanitize(data)

end

private

def sanitize(data)

remove\_white\_space\_and\_letters(data)

add\_start\_code(data)

end

def add\_start\_code(data)

return data if start\_of\_code?(data)

data.gsub!(/^/, "+")

end

def remove\_white\_space\_and\_letters(data)

data.gsub!(PHONE\_SANITIZE, "")

end

def start\_of\_code?(data)

data.match(/^\+/).present?

end

end

end

module ApplicationCable

class Channel < ActionCable::Channel::Base

end

end

module ApplicationCable

class Connection < ActionCable::Connection::Base

end

end

module Admin

class SettingSerializer < BaseSerializer

set\_type :setting

set\_key\_transform :camel\_lower

attributes :vat,

:title,

:currency

end

end

module Admin

module ApplicationHelper

include BootstrapIcons::Rails::BiIconHelper

def navigation

@navigation ||= Admin::NavigationBuilder.new.call(current\_ability: current\_ability.class)

end

def read\_constant(constant\_name)

self.class.const\_get(constant\_name)

end

end

end

module Admin

module RespondHelper

include ::RespondHelper

def respond\_html\_and\_datatable\_json\_for(model, options = {})

template = options.delete :template

respond\_to do |format|

format.html { render template if template.present? }

format.json { render json: datatable\_for(model, options) }

end

end

def respond\_json\_with\_errors(model)

render json: model.errors.full\_messages, status: :unprocessable\_entity

end

end

end

module Admin

module OrdersHelper

def discount\_title(order)

order.discount.present? ? order.discount.title : ""

end

end

end

module SiteHandler

extend ActiveSupport::Concern

DEFAULT\_LOCALE = "ru"

included do

helper\_method :params\_selected

def params\_selected

request.params[:selected].presence

end

private

def raise\_not\_found\_error

raise ActionController::RoutingError, "Not Found"

end

def stripped\_domain

request.env["HTTP\_HOST"].sub("www.", "")

end

end

end

module Admin

class SessionsController < Devise::SessionsController

before\_action :configure\_sign\_in\_params, only: [:create]

UPDATE\_MODEL\_PARAMS = %i[email

password

remember\_me].freeze

MODEL\_PARAMS = UPDATE\_MODEL\_PARAMS

def new

super

end

def create

super

end

def destroy

super

end

protected

def configure\_sign\_in\_params

params.require(:administrator).permit(\*MODEL\_PARAMS)

end

end

end

module Admin

class UsersController < Admin::ApplicationController

include Admin::Datatable

load\_and\_authorize\_resource

skip\_load\_resource only: :index

UPDATE\_USER\_PARAMS = %i[first\_name

last\_name

father\_name

phone\_number].freeze

USER\_PARAMS = UPDATE\_USER\_PARAMS

datatable model: ::User, exclude\_attributes: %w[created\_at updated\_at]

def index

@users = User.all

respond\_to do |format|

format.html

format.json { render json: @users }

end

end

def new; end

def edit; end

def create

@user.save

end

def update

@user.update update\_user\_params

respond\_with @user, location: users\_path

end

def destroy

@user.destroy

respond\_with @user, location: users\_path

end

private

def user\_params

params.require(:user).permit(\*USER\_PARAMS)

end

def update\_user\_params

params.require(:user).permit(\*UPDATE\_USER\_PARAMS)

end

end

end

module Admin

class OmniauthCallbacksController < Devise::OmniauthCallbacksController

end

end

module Admin

class SettingsController < Admin::ApplicationController

def index; end

def create; end

def edit; end

def update

setting\_params.keys.each do |key|

Setting.\_\_send\_\_("#{key}=", setting\_params[key].strip) unless setting\_params[key].nil?

end

redirect\_to setting\_path, notice: "Setting was successfully updated."

rescue StandardError => exception

respond\_with @setting, location: setting\_path, notice: exception.message

end

private

def setting\_params

params.require(:setting)

end

end

end

module Admin

class MastersController < Admin::ApplicationController

include Admin::Datatable

load\_and\_authorize\_resource

skip\_load\_resource only: :index

datatable model: Master, exclude\_attributes: %w[created\_at updated\_at]

UPDATE\_MODEL\_PARAMS = %i[last\_name

first\_name

father\_name

phone\_number

specialization

work\_experience].freeze

MODEL\_PARAMS = UPDATE\_MODEL\_PARAMS

def create

@master = model.create(sanitized\_params)

respond\_with @master, location: masters\_path

end

def update

@master.update(sanitized\_params)

respond\_with @master, location: master\_path

end

def destroy

@master.destroy

redirect\_to masters\_path

end

private

def sanitized\_params

phone\_number = model\_params.fetch(:phone\_number, "")

sanitized\_phone\_number = ::Sanitizers::PhoneNumber.new.call(phone\_number)

model\_params.merge(phone\_number: sanitized\_phone\_number)

end

end

end

module Admin

class UnlocksController < Devise::UnlocksController

end

end

module Admin

class AdministratorsController < Admin::ApplicationController

include Admin::Datatable

load\_and\_authorize\_resource

skip\_load\_resource only: :index

before\_action :select\_option\_collection, only: %i[edit new update create]

datatable model: Administrator, exclude\_attributes: %w[encrypted\_password

reset\_password\_token

reset\_password\_sent\_at

remember\_created\_at

created\_at

updated\_at]

UPDATE\_MODEL\_PARAMS = %i[last\_name

first\_name

father\_name

phone\_number

email

role

password

password\_confirmation].freeze

MODEL\_PARAMS = UPDATE\_MODEL\_PARAMS

def create

@administrator = model.create(sanitized\_params)

respond\_with @administrator, location: administrators\_path

end

def update

@administrator.update(sanitized\_params)

respond\_with @administrator, location: administrator\_path

end

def destroy

@administrator.destroy

redirect\_to administrators\_path

end

private

def select\_option\_collection

@role\_select\_options = Administrator::AVAILABLE\_ROLES.map do |role|

[I18n.t("activerecord.attributes.admin/administrator.roles.#{role}"), role, { data: { type: role } }]

end

end

def sanitized\_params

phone\_number = model\_params.fetch(:phone\_number, "")

sanitized\_phone\_number = ::Sanitizers::PhoneNumber.new.call(phone\_number)

model\_params.merge(phone\_number: sanitized\_phone\_number)

end

end

end

module Admin

class ServicesController < Admin::ApplicationController

include Admin::Datatable

load\_and\_authorize\_resource

skip\_load\_resource only: :index

datatable model: Service, exclude\_attributes: %w[created\_at updated\_at]

UPDATE\_MODEL\_PARAMS = %i[title

description

price].freeze

MODEL\_PARAMS = UPDATE\_MODEL\_PARAMS

def create

@service = model.create(model\_params)

respond\_with @service, location: services\_path

end

def update

@service.update(model\_params)

respond\_with @service, location: service\_path

end

def destroy

@service.destroy

redirect\_to services\_path

end

end

end

module Admin

class ApplicationController < ::ApplicationController

include Admin::SiteHandler

include Admin::ApplicationHelper

include Admin::RespondHelper

before\_action :authenticate\_administrator!, :set\_current\_administrator

respond\_to :html

respond\_to :json

protect\_from\_forgery with: :exception

layout "admin"

rescue\_from CanCan::AccessDenied do |exception|

if current\_administrator

redirect\_to root\_path

else

flash[:error] = exception.message

head status: :unauthorized

end

end

def index

@model\_attributes = model\_attributes

@model = model

@model\_items = respond\_with\_search

end

def create

@model = model.new(model\_params)

redirect\_to @model if @model.save

end

private

def redirect\_to\_action\_if\_valid(model, action = :index)

if model.valid?

redirect\_to action: action

else

respond\_with model

end

end

def authenticate\_administrator!

if administrator\_signed\_in?

super

else

redirect\_to administrator\_session\_path, notice: "if you want to add a notice"

end

end

def redirect\_back\_or\_root(\*\*args)

redirect\_back(fallback\_location: root\_path, \*\*args)

end

def set\_current\_administrator

Admin::Administrator.current = current\_administrator

end

def current\_ability

@current\_ability ||= Abilities::Factory.build(current\_administrator)

end

def model\_params

params.require(model\_symbolize).permit(\*read\_constant(:MODEL\_PARAMS))

end

def model\_symbolize

model.to\_s.downcase.split("::").last.to\_sym

end

end

end

module Admin

class ConfirmationsController < Devise::ConfirmationsController; end

end

module Admin

class OrdersController < Admin::ApplicationController

include Admin::Datatable

load\_and\_authorize\_resource

skip\_load\_resource only: :index

before\_action :select\_option\_collection, only: %i[edit new update]

datatable model: Order, exclude\_attributes: %w[created\_at

updated\_at

admin\_discount\_id

admin\_customer\_id

admin\_master\_id]

UPDATE\_MODEL\_PARAMS = %i[admin\_discount\_id

admin\_customer\_id

admin\_master\_id

end\_execution\_at

start\_execution\_at].freeze

MODEL\_PARAMS = UPDATE\_MODEL\_PARAMS + [service\_ids: []]

def create

@order = model.create(model\_params)

respond\_with @order, location: orders\_path

end

def update

@order.update(model\_params)

respond\_with @order, location: order\_path

end

def destroy

@order.destroy

redirect\_to orders\_path

end

private

def select\_option\_collection

@master\_select\_options = Admin::Master.all

.map { |item| [item.full\_name, item.id, { data: { type: item.id } }] }

@customer\_select\_options = Admin::Customer.all

.map { |item| [item.full\_name, item.id, { data: { type: item.id } }] }

@discount\_select\_options = Admin::Discount.published

.map { |item| [item.title, item.id, { data: { type: item.id } }] }

end

end

end

module Admin

class DiscountsController < Admin::ApplicationController

include Admin::Datatable

load\_and\_authorize\_resource

skip\_load\_resource only: :index

datatable model: Discount, exclude\_attributes: %w[created\_at updated\_at]

UPDATE\_MODEL\_PARAMS = %i[title

description

end\_date\_at

start\_date\_at

percent].freeze

MODEL\_PARAMS = UPDATE\_MODEL\_PARAMS

def create

@discount = model.create(model\_params)

respond\_with @discount, location: discounts\_path

end

def update

@discount.update(model\_params)

respond\_with @discount, location: discount\_path

end

def destroy

@discount.destroy

redirect\_to discounts\_path

end

end

end

module Admin

class NewsController < Admin::ApplicationController

include Admin::Datatable

load\_and\_authorize\_resource

skip\_load\_resource only: :index

datatable model: News, exclude\_attributes: %w[created\_at updated\_at]

UPDATE\_MODEL\_PARAMS = %i[title

content].freeze

MODEL\_PARAMS = UPDATE\_MODEL\_PARAMS

def create

@news = model.create(model\_params)

respond\_with @news, location: news\_index\_path

end

def update

@news.update(model\_params)

respond\_with @news, location: news\_path

end

def destroy

@news.destroy

redirect\_to news\_index\_path

end

end

end

module Admin

class CustomersController < Admin::ApplicationController

include Admin::Datatable

load\_and\_authorize\_resource

skip\_load\_resource only: :index

UPDATE\_MODEL\_PARAMS = %i[last\_name

first\_name

father\_name

phone\_number].freeze

MODEL\_PARAMS = UPDATE\_MODEL\_PARAMS

datatable model: Customer, exclude\_attributes: %w[created\_at updated\_at]

def create

@customer = model.create(sanitized\_params)

respond\_with @customer, location: customers\_path

end

def update

@customer.update(sanitized\_params)

respond\_with @order, location: customer\_path

end

def destroy

@customer.destroy

redirect\_to customers\_path

end

private

def sanitized\_params

phone\_number = model\_params.fetch(:phone\_number, "")

sanitized\_phone\_number = ::Sanitizers::PhoneNumber.new.call(phone\_number)

model\_params.merge(phone\_number: sanitized\_phone\_number)

end

end

end

module Admin

class PasswordsController < Devise::PasswordsController; end

end

module Admin

class RegistrationsController < Devise::RegistrationsController

ADMIN\_ADMINISTRATOR\_PARAMS = %i[email password first\_name last\_name father\_name phone\_number role].freeze

def create

build\_administrator\_with\_role(:manager) unless create\_chief\_executive\_officer.present?

redirect\_to administrator\_session\_path

end

private

def create\_chief\_executive\_officer

return if Admin::Administrator.where(role: "chief\_executive\_officer").take.present?

build\_administrator\_with\_role(:chief\_executive\_officer)

end

def build\_administrator\_with\_role(role)

@administrator = Admin::Administrator.new(administrator\_params)

@administrator.role = role.to\_s

@administrator.save

end

def administrator\_params

params.require(:administrator).permit(\*ADMIN\_ADMINISTRATOR\_PARAMS)

end

end

end

module Admin

class HomeController < Admin::ApplicationController

skip\_load\_and\_authorize\_resource

def index; end

end

end

module Cdn

class OrdersController < ApplicationController

include ::RespondHelper

include ::ApplicationHelper

def index

@orders = Admin::Order.where(id: params[:orders\_ids])

respond\_html\_json\_and\_pdf\_for(@orders)

end

end

end

module Api

class ApplicationController < ActionController::Base

include SiteHandler

include ::RespondHelper

respond\_to :json

rescue\_from StandardError, with: :exception

rescue\_from ActiveRecord::RecordNotFound, with: :record\_not\_found

rescue\_from ActionController::ParameterMissing, with: :bad\_request

private

def record\_not\_found

head :not\_found

end

def record\_invalid

head :unprocessable\_entity

end

def bad\_request

head :bad\_request

end

def exception(exception)

Rails.logger.error(exception.full\_message)

head :internal\_server\_error

end

end

end

module Front

class HomeController < ApplicationController

include ::RespondHelper

layout "front"

before\_action :home\_params, only: %i[index]

def order\_info

@orders = search\_by\_params(params[:search])

respond\_html\_json\_and\_pdf\_for(@orders, template: { partial: "order\_info" })

end

private

PHONE\_NUMBER\_PATTERN = /^\+375(17|29|33|44)\d{3}\d{2}\d{2}$/.freeze

def search\_by\_params(value)

by\_id = search\_by\_order\_id(value)

return by\_id if by\_id.present?

by\_phone = search\_by\_phone\_number(value)

return by\_phone if by\_phone.present?

[]

end

def search\_by\_order\_id(value)

Admin::Order.find\_not\_finished.where(id: value)

end

def search\_by\_phone\_number(value)

phone\_number = ::Sanitizers::PhoneNumber.new.call(value)

customer = Admin::Customer.where(phone\_number: phone\_number).take

return customer.orders.find\_not\_finished if valid\_phone\_number?(value) && customer.present?

end

def home\_params

@news = Admin::News.all.order(updated\_at: :ASC)

end

def valid\_phone\_number?(phone\_number)

(phone\_number =~ PHONE\_NUMBER\_PATTERN).present?

end

end

end

module Abilities

class ChiefEditor < BaseAbility

MODELS = [

::Admin::Order,

::Admin::Customer,

::Admin::Administrator,

::Admin::Discount,

::Admin::Master,

::Admin::Service,

::Admin::News

].freeze

PERMISSIONS = {

read: [

::Admin::Administrator

],

manage: [

::Admin::Order,

::Admin::Customer,

::Admin::Discount,

::Admin::Master,

::Admin::Service,

::Admin::News

]

}.freeze

end

end

module Abilities

class Manager < BaseAbility

MODELS = [

::Admin::Order,

::Admin::Customer,

::Admin::Administrator,

::Admin::Discount,

::Admin::Master,

::Admin::Service

].freeze

PERMISSIONS = {

edit: [::Admin::Order],

read: [::Admin::Order,

::Admin::Customer,

::Admin::Administrator,

::Admin::Discount,

::Admin::Master,

::Admin::Service,

::Admin::News],

create: [::Admin::Order, ::Admin::Customer]

}.freeze

end

end

module Abilities

class Factory

class << self

def build(user = ::Admin::Administrator.new)

return Ability.new(user) unless user.role\_available?

ability\_class\_for(user).new(user)

end

private

def ability\_class\_for(user)

"Abilities::#{user.role.classify}".constantize

end

end

end

end

module Abilities

class ChiefExecutiveOfficer < BaseAbility

MODELS = [

::Admin::Order,

::Admin::Customer,

::Admin::Administrator,

::Admin::Discount,

::Admin::Master,

::Admin::Service,

::Admin::News

].freeze

PERMISSIONS = {

manage: MODELS + [::User, ::Setting]

}.freeze

end

end

module Abilities

class BaseAbility < Ability

PERMISSIONS = {}.freeze

end

end

module Admin

class News < ApplicationRecord

pg\_search\_scope :search, against: %i[id

title

content]

validates\_presence\_of :title, :content

end

end

module Admin

class Order < ApplicationRecord

has\_and\_belongs\_to\_many :services, class\_name: "Admin::Service"

belongs\_to :master, foreign\_key: :admin\_master\_id, class\_name: "Admin::Master"

belongs\_to :discount, foreign\_key: :admin\_discount\_id, required: false, class\_name: "Admin::Discount"

belongs\_to :customer, foreign\_key: :admin\_customer\_id, class\_name: "Admin::Customer"

validates\_presence\_of :services, :end\_execution\_at, :start\_execution\_at

validates\_datetime :end\_execution\_at, after: :start\_execution\_at, allow\_blank: false

pg\_search\_scope :search, against: %i[id end\_execution\_at start\_execution\_at]

scope :find\_not\_finished, lambda {

where(

arel\_table[:start\_execution\_at]

.lt(arel\_table[:end\_execution\_at])

.and(arel\_table[:end\_execution\_at]

.gt(Arel::Nodes::NamedFunction.new("now", [])))

)

}

def total\_sum

services.sum(:price)

end

def discount\_amount

percent = discount.nil? ? 0 : discount.percent

(total\_sum \* percent / 100).round(2)

end

def total\_sum\_with\_discount

percent = discount.nil? ? 0 : discount.percent

(total\_sum - (total\_sum \* percent / 100)).round(2)

end

end

end

module Admin

class Administrator < User

extend Enumerize

include VoucherHelper

self.table\_name = :admin\_administrators

AVAILABLE\_ROLES = %w[

chief\_executive\_officer chief\_editor manager

].freeze

devise :database\_authenticatable, :registerable,

:recoverable, :rememberable, :validatable

validates :email, :first\_name, :last\_name, :father\_name, :phone\_number, :role, presence: true

validates :phone\_number, uniqueness: { scope: :phone\_number }

enumerize :role, in: AVAILABLE\_ROLES, predicates: true

before\_validation :create\_voucher

pg\_search\_scope :search, against: %i[id

email

first\_name

last\_name

father\_name

phone\_number

role]

class << self

def current=(user)

RequestStore.store[:current\_administrator] = user

end

def current

RequestStore.store[:current\_administrator]

end

end

def role\_available?

AVAILABLE\_ROLES.include?(role)

end

end

end

module Admin

class Discount < ApplicationRecord

has\_many :orders, foreign\_key: :admin\_discount\_id, class\_name: "Admin::Order"

validates\_datetime :end\_date\_at, after: :start\_date\_at

validates\_presence\_of :title, :description, :start\_date\_at, :end\_date\_at

validates :percent,

numericality: { only\_integer: true },

length: { maximum: 100, minimum: 0 },

allow\_blank: false,

presence: true

pg\_search\_scope :search, against: %i[id

title

description]

scope :published, lambda {

where(

arel\_table[:start\_date\_at]

.lt(arel\_table[:end\_date\_at])

.and(arel\_table[:end\_date\_at]

.gt(Arel::Nodes::NamedFunction.new("now", [])))

)

}

end

end

module Admin

class Customer < User

self.table\_name = :admin\_customers

has\_many :orders, foreign\_key: :admin\_customer\_id, class\_name: "Admin::Order"

pg\_search\_scope :search, against: %i[id

first\_name

last\_name

father\_name

phone\_number]

end

end

module Admin

class Master < User

self.table\_name = :admin\_masters

has\_many :orders, foreign\_key: :admin\_master\_id, class\_name: "Admin::Order"

pg\_search\_scope :search, against: %i[id

first\_name

last\_name

father\_name

phone\_number

work\_experience

specialization]

validates\_presence\_of :specialization

validates :work\_experience,

numericality: { only\_integer: true },

allow\_blank: false,

presence: true

end

end

module Admin

class Service < ApplicationRecord

has\_and\_belongs\_to\_many :orders, class\_name: "Admin::Order"

pg\_search\_scope :search, against: %i[id

description

title

price]

validates\_presence\_of :description, :title

validates :price,

numericality: { only\_integer: false, greater\_than: 0 },

allow\_blank: false,

presence: true

end

end

module Admin

class NavigationBuilder

def call(current\_ability:)

build(current\_ability)

end

private

def build(current\_ability)

current\_ability::MODELS.each\_with\_object([]) do |item, memo|

controller\_name = item.class\_name.pluralize

memo << { lable: controller\_name, model: item }

end

end

end

end

module Api

module V1

class CustomersSerializer < BaseSerializer

set\_type :customer

set\_key\_transform :camel\_lower

attributes :id,

:first\_name,

:last\_name,

:father\_name

end

end

end

module Api

module V1

class OrdersSerializer < BaseSerializer

set\_type :order

set\_key\_transform :camel\_lower

attributes :id,

:end\_execution\_at,

:start\_execution\_at

attribute :discount do |object, \_params|

discounts = Admin::Discount.where(id: object.admin\_discount\_id)

DiscountsSerializer.new(discounts).as\_json

end

attribute :master do |object, \_params|

masters = Admin::Master.where(id: object.admin\_master\_id)

MastersSerializer.new(masters).as\_json

end

attribute :customer do |object, \_params|

masters = Admin::Customer.where(id: object.admin\_customer\_id)

CustomersSerializer.new(masters).as\_json

end

end

end

end

module Api

module V1

class ServicesSerializer < BaseSerializer

set\_type :services

set\_key\_transform :camel\_lower

attributes :id,

:description,

:title

attribute :price do |object, \_params|

object.price.round(2)

end

end

end

end

module Api

module V1

class MastersSerializer < BaseSerializer

set\_type :master

set\_key\_transform :camel\_lower

attributes :id,

:last\_name,

:first\_name,

:father\_name,

:specialization,

:work\_experience

end

end

end

module Api

module V1

class NewsSerializer < BaseSerializer

set\_type :news

set\_key\_transform :camel\_lower

attributes :id,

:title,

:content

end

end

end

module Api

module V1

class DiscountsSerializer < BaseSerializer

set\_type :discount

set\_key\_transform :camel\_lower

attributes :id,

:title,

:description,

:end\_date\_at,

:start\_date\_at,

:percent

end

end

end

module Admin

module Locale

extend ActiveSupport::Concern

included do

helper\_method :t

def t(key, options = {})

super key, options.deep\_merge(scope: :admin) { |\_key, prev\_value, curr\_value| "#{curr\_value}.#{prev\_value}" }

end

end

end

end

module Admin

module SiteHandler

extend ActiveSupport::Concern

included do

include ::SiteHandler

end

end

end

module Admin

module Datatable

extend ActiveSupport::Concern

class\_methods do

attr\_reader :model, :exclude\_attributes

def datatable(model: nil, exclude\_attributes: [])

@model = model

@exclude\_attributes = exclude\_attributes

end

end

included do

private

def render\_datatable\_json

render(json: serialized\_json)

rescue Datatable::QueryError => exception

log\_error(exception.message)

head exception.status\_code

end

def model\_attributes

self.class.model.column\_names - self.class.exclude\_attributes

end

def model

self.class.model

end

def log\_error(message)

Rails.logger.error("#{self.class.name}: #{message}")

end

def respond\_with\_search

items = model.search(search\_params)

return model.all.order(id: :ASC) unless items.present?

items

end

def search\_params

params.fetch(:search, "")

end

def buld\_like\_query

where("title LIKE ?", "%#{search}%")

end

end

end

end

module Api

module V1

class MastersController < Api::ApplicationController

def index

data = repository

render json: serialize\_data(data)

end

private

def repository

Admin::Master.all

end

def serialize\_data(data)

Api::V1::MastersSerializer.new(data).as\_json

end

end

end

end

module Api

module V1

class ServicesController < Api::ApplicationController

def index

data = repository

render json: serialize\_data(data)

end

private

def repository

Admin::Service.all

end

def serialize\_data(data)

Api::V1::ServicesSerializer.new(data).as\_json

end

end

end

end

module Api

module V1

class OrdersController < Api::ApplicationController

def index

data = repository

render json: serialize\_data(data)

end

private

def repository

Admin::Order.all

end

def serialize\_data(data)

Api::V1::OrdersSerializer.new(data).as\_json

end

end

end

end

module Api

module V1

class DiscountsController < Api::ApplicationController

def index

data = repository

render json: serialize\_data(data)

end

private

def repository

Admin::Discount.all

end

def serialize\_data(data)

Api::V1::DiscountsSerializer.new(data).as\_json

end

end

end

end

module Api

module V1

class NewsController < Api::ApplicationController

def index

data = repository

render json: serialize\_data(data)

end

private

def repository

Admin::News.all

end

def serialize\_data(data)

Api::V1::NewsSerializer.new(data).as\_json

end

end

end

end

module Api

module V1

class CustomersController < Api::ApplicationController

def index

data = repository

render json: serialize\_data(data)

end

private

def repository

Admin::Customer.all

end

def serialize\_data(data)

Api::V1::CustomersSerializer.new(data).as\_json

end

end

end

end

module Api

module V1

module Report

class GrowthCustomersController < Api::ApplicationController

def index

respond\_html\_json\_and\_pdf\_for(customers\_growth\_options)

end

private

def customers\_growth\_options

Admin::Customer.group\_by\_month(:created\_at).count

end

end

end

end

end

module Api

module V1

module Report

class OrdersController < Api::ApplicationController

def most\_expensive

respond\_html\_json\_and\_pdf\_for(most\_expensive\_options)

end

def accepted\_by\_master

respond\_html\_json\_and\_pdf\_for(accepted\_by\_master\_options)

end

private

def accepted\_by\_master\_options

Admin::Order.joins(:master).group(:"admin\_masters.id")

.count.each\_with\_object([]) do |element, memo|

next unless element.first.present?

memo << [Admin::Master.find(element.first).full\_name, element.last]

end

end

def most\_expensive\_options

Admin::Order.all.group\_by(&:total\_sum)

.sort.each\_with\_object([]) do |element, hash|

hash << [I18n.t("reports.order",

order\_id: element.last.first.id),

element.first]

end

end

end

end

end

end

module Api

module V1

module Report

class IncreaseOrdersController < Api::ApplicationController

def index

respond\_html\_json\_and\_pdf\_for(increase\_orders\_options)

end

private

def increase\_orders\_options

Admin::Order.group\_by\_day\_of\_week(:created\_at).count

.each\_with\_object([]) do |element, memo|

memo << [Date::DAYNAMES[element.first], element.last]

end

end

end

end

end

end

# **Приложение Б**

**(Справочное)**

**Тест-кейс разрабатываемого веб-приложения**

В таблице Б1 представлен чек-лист и тест-кейс разработанного веб-сайта.

Таблица Б1 – Тест-кейс разрабатываемого веб-сайта

|  |  |  |  |
| --- | --- | --- | --- |
| **Название модуля/экрана** | **Описание тестового случая** | **Ожидаемые результаты** | **Фактические результаты** |
|  |  |  |  |
|  |  |  |  |

# **Приложение В**

**(Справочное)**

**Чек-лист разрабатываемого веб-сайта**

В таблице В1 представлен чек-лист и тест-кейс разработанного веб-сайта.

Таблица В1 – Чек-лист веб-сайта

|  |  |  |
| --- | --- | --- |
| # | Описание | ДА / НЕТ |
| 1 | При запуске веб-приложения | ДА |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |