Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Кафедра ИСиТ

Дисциплина: СПП

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ПС «АРМ Велосервиса»

БГУИР КП 1-40.01.01.306 ПЗ

Студент: гр. 981064 Ефименко П.В.

Руководитель: Бакунов А.М.

Минск 2021

СОДЕРЖАНИЕ

[Введение 5](#_Toc90849213)

[1 Анализ предметной области и формирования требований 7](#_Toc90849214)

[1.1 Анализ предметной области 7](#_Toc90849215)

[1.2 Обзор аналогов программных средств 11](#_Toc90849216)

[1.3 Постановка задачи 12](#_Toc90849217)

[1.4 Входные и выходные данные 13](#_Toc90849218)

[1.5 Обоснование выбора языка и среды программирования 14](#_Toc90849219)

[2 Моделирование предметной области и разработка функциональных требований 17](#_Toc90849220)

[2.1 Разработка функциональной модели 17](#_Toc90849221)

[2.2 Спецификация функциональных требований 17](#_Toc90849222)

[2.3 Инфологическая модель БД 18](#_Toc90849223)

[3 Проектирование программного средства 20](#_Toc90849224)

[3.1 Структура программного средства 20](#_Toc90849225)

[3.2 Физическая модель базы данных 21](#_Toc90849226)

[3.3 Разработка архитектуры программного средства 22](#_Toc90849227)

[3.4 Разработка алгоритмов программного средства 24](#_Toc90849228)

[3.5 Конструирование программного средства 25](#_Toc90849229)

[3.6 Разработка пользовательских интерфейсов 26](#_Toc90849230)

[4 Тестирование 28](#_Toc90849231)

[4.1 Описание тестового стенда 28](#_Toc90849232)

[4.2 Выбор и обоснование видов тестирования 28](#_Toc90849233)

[4.3 Выводы по тестированию 30](#_Toc90849234)

[5 Руководство пользователя 34](#_Toc90849235)

[ЗАКЛЮЧЕНИЕ 44](#_Toc90849236)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 45](#_Toc90849237)

[Приложение А 46](#_Toc90849238)

[Приложение Б 75](#_Toc90849239)

[Приложение В 76](#_Toc90849240)

[Приложение Г 77](#_Toc90849241)

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 и реализация документооборота предприятия.

Документооборот – это движение документов в организации с момента их создания или получения до завершения исполнения или отправки.

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 Велосервиса». 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 «Анализ предметной области и формирование требований» включает в себя анализ предметной области, обзор аналогов, постановка задачи, входные и выходные данные, а также выбор и обоснование средств разработки;
* второй раздел «Моделирование предметной области и разработка функциональных требований» включает в себя разработка функциональной модели, спецификация функциональных требований инфологическая модель БД;
* в разделе «Проектирование 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.

1. Анализ предметной области и формирования требований
   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 велосервиса. 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.
  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 java. На сегодняшний момент язык программирования java один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. Java – объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. Изначально новый язык программирования назывался Oak (James Gosling) и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения. Программы на Java могут быть транслированы в байт-код, выполняемый на виртуальной java-машине (JVM) – программе, обрабатывающей байт-код и передающей инструкции оборудованию, как интерпретатор, но с тем отличием, что байт-код, в отличие от текста, обрабатывается значительно быстрее. Язык Java зародился как часть проекта создания передового программного обеспечения для различных бытовых приборов. Реализация проекта была начата на языке C++, но вскоре возник ряд проблем, наилучшим средством борьбы с которыми было изменение самого инструмента — языка программирования. Стало очевидным, что необходим платформо-независимый язык программирования, позволяющий создавать программы, которые не приходилось бы компилировать отдельно для каждой архитектуры и можно было бы использовать на различных процессорах под различными операционными системами. Язык Java потребовался для создания интерактивных продуктов для сети Internet. Фактически, большинство архитектурных решений, принятых при создании Java, было продиктовано желанием предоставить синтаксис, сходный с C и C++. В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты C++.

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. Моделирование предметной области и разработка функциональных требований
   1. Разработка функциональной модели

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 Приложении Б.

* 1. Спецификация функциональных требований

Программное средство разработано на языке программирования java в обвязке с Фреймворком spring.

Spring Framework представляет собой просто контейнер внедрения зависимостей, с несколькими удобными слоями (например: доступ к базе данных, прокси, аспектно-ориентированное программирование, RPC, веб-инфраструктура MVC). Это все позволяет вам быстрее и удобнее создавать Java-приложения.

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 – 4, минимальный объем оперативной памяти – 8 Гигабайта, минимальная частота процессора на 1 ядро – 2 Гигагерц, минимальный объем дискового пространства – 5 Гигабайт.

PostgreSQL версии 13, Java версии 15, NodeJS версии 14.16.0, Yarn версии 1.16.0, Redis 5.0.5.

* 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 на рисунке 2.1.

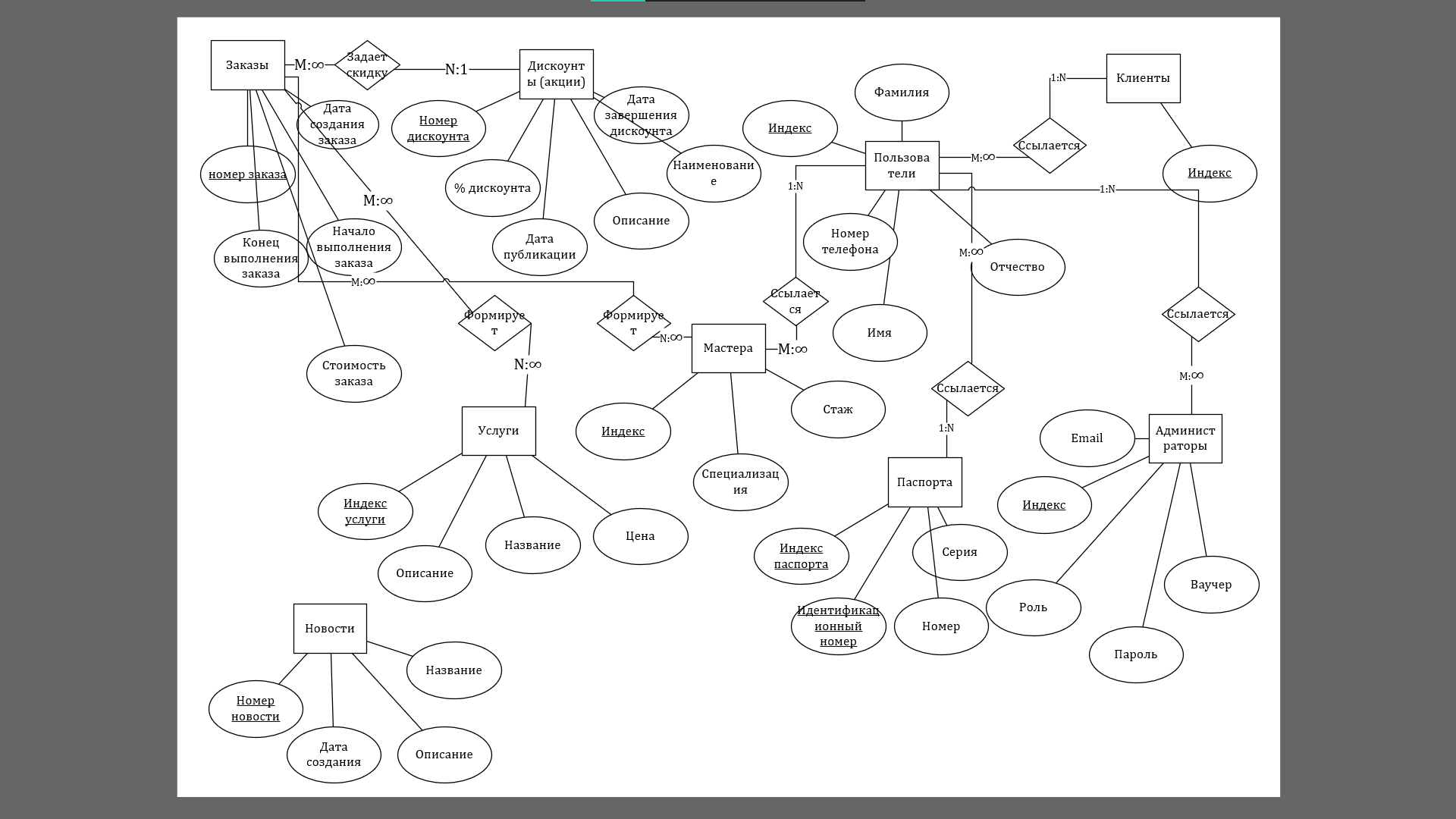


Рисунок 2.1 – Инфологическая модель БД

1. Проектирование программного средства
   1. Структура программного средства

Структура проекта представлена на рисунке 3.1.

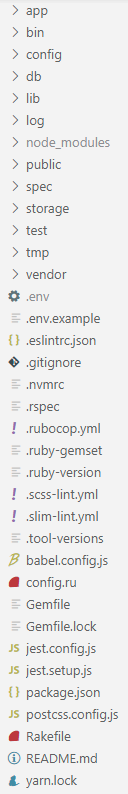
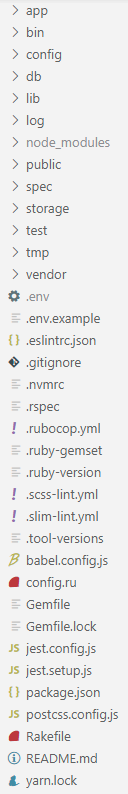
 

Рисунок 3.1 – Структура проекта

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.

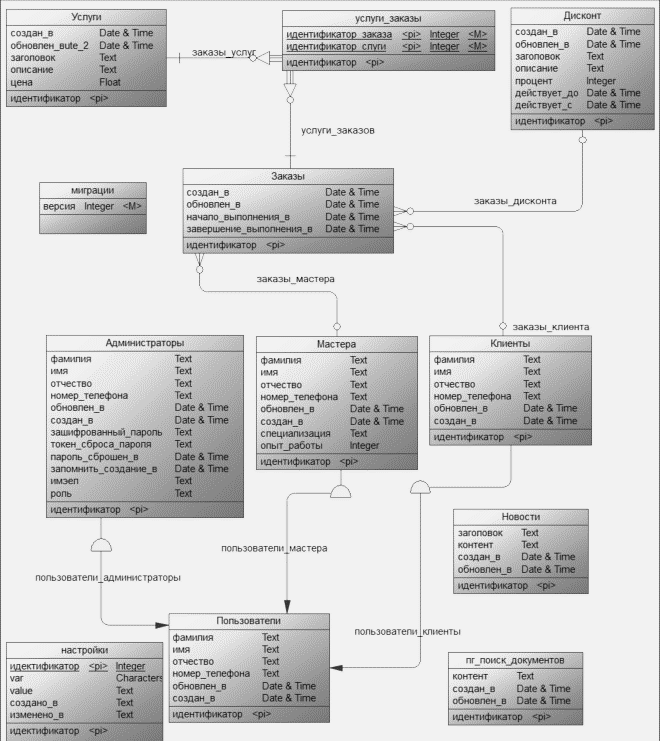


Рисунок 3.2 – Схема созданной базы данных

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. Разработка архитектуры программного средства

Приложение использует архитектурный стиль RESTful API. REST – это аббревиатура от Representational State Transfer («передача состояния представления»). Это согласованный набор архитектурных принципов для создания более масштабируемой и гибкой сети. REST API применяют везде, где пользователю сайта или веб-приложения нужно предоставить данные с сервера. Например, при нажатии иконки с видео на видеохостинге REST API проводит операции и запускает ролик с сервера в браузере. В настоящее время это самый распространенный способ организации API. Он вытеснил ранее популярные способы SOAP и WSDL.

У RESTful нет единого стандарта работы: его называют «архитектурным стилем» для операций по работе с серверов. Такой подход в 2000 году в своей диссертации ввел программист и исследователь Рой Филдинг, один из создателей протокола HTTP.

У RESTful есть семь принципов написания кода интерфейсов.

1. Отделения клиента от сервера (Client-Server). Клиент – это пользовательский интерфейс сайта или приложения, например, поисковая строка видеохостинга. В REST API код запросов остается на стороне клиента, а код для доступа к данным — на стороне сервера. Это упрощает организацию API, позволяет легко переносить пользовательский интерфейс на другую платформу и дает возможность лучше масштабировать серверное хранение данных.
2. Отсутствие записи состояния клиента (Stateless). Сервер не должен хранить информацию о состоянии (проведенных операций) клиента. Каждый запрос от клиента должен содержать только ту информацию, которая нужна для получения данных от сервера.
3. Кэшируемость (Casheable). В данных запроса должно быть указано, нужно ли кэшировать данные (сохранять в специальном буфере для частых запросов). Если такое указание есть, клиент получит право обращаться к этому буферу при необходимости.
4. Единство интерфейса (Uniform Interface). Все данные должны запрашиваться через один URL-адрес стандартными протоколами, например, HTTP. Это упрощает архитектуру сайта или приложения и делает взаимодействие с сервером понятнее.
5. Многоуровневость системы (Layered System). В RESTful сервера могут располагаться на разных уровнях, при этом каждый сервер взаимодействует только с ближайшими уровнями и не связан запросами с другими.
6. Предоставление кода по запросу (Code on Demand). Серверы могут отправлять клиенту код (например, скрипт для запуска видео). Так общий код приложения или сайта становится сложнее только при необходимости.
7. Начало от нуля (Starting with the Null Style). Клиент знает только одну точку входа на сервер. Дальнейшие возможности по взаимодействию обеспечиваются сервером.

REST API основывается на протоколе передачи гипертекста HTTP (Hypertext Transfer Protocol). Это стандартный протокол в интернете, созданный для передачи гипертекста. Сейчас с помощью HTTP отправляют любые другие типы данных.

Каждый объект на сервере в HTTP имеет свой уникальный URL-адрес в строгом последовательном формате.

В REST API есть четыре метода HTTP, которые используют для действий с объектами на серверах:

* GET (получение информации о данных или списка объектов)
* DELETE (удаление данных)
* POST (добавление или замена данных)
* PUT (регулярное обновление данных)

Такие запросы еще называют идентификаторами CRUD: create (создать), read (прочесть), update (обновить) delete (удалить). Это стандартный набор действий для работы с данными.

Схема деятельности алгоритма представлена в Приложении В.

* 1. Разработка алгоритмов программного средства

Интерфейс программы является «связью с внешним миром». Он ожидает в цикле прихода какой-либо команды, периодически проверяя её наличие. Такое поведение является стандартным для WEB-приложений. Когда приходит команда, происходит её разбор, а именно определяется тип и, соответственно, действие, которое нужно выполнить или определить какому модулю передать управление.

Перед запуском поиска оптимального сочетания критериев обязательно осуществляется проверка корректности модели. Данная проверка заключается в определении циклов в зависимостях компонентов одной подсистемы, в проверке правильности выражений. Одновременно вычленяются неиспользуемые компоненты и подсистемы, и их ветви «отрезаются» от общего дерева.

Специфика применяемых методов проектирования алгоритмов и используемых при этом инструментальных средств разработки программ может повлиять на форму представления и содержание алгоритма обработки данных. Алгоритм работы программного средства можно представить в виде блок-схемы.

Существует несколько типов алгоритмов:

− линейный алгоритм – набор команд (указаний), выполняемых последовательно во времени друг за другом;

− разветвляющийся алгоритм – алгоритм, содержащий хотя бы одно условие, в результате проверки которого может осуществляться разделение на несколько альтернативных ветвей алгоритма;

− циклический алгоритм – алгоритм, предусматривающий многократное повторение одного и того же действия (одних и тех же операций) над новыми исходными данными. К циклическим алгоритмам сводится большинство методов вычислений, перебора вариантов.

Алгоритм работы ПС:

− открыть приложение;

− пройти авторизацию (ввод логина и пароля, их проверка);

− выбрать операцию;

− ввести данные для проведения операции;

− провести операцию;

− сохранить изменения;

− выйти из приложения, либо продолжить работу.

Схема алгоритма представлена в Приложении Г.

Для данного программного средства выбран алгоритм смешанного типа, который включает в себя разветвляющийся и циклический алгоритмы.

* 1. Конструирование программного средства

В программном средстве будут использоваться языки для реализации поставленной задачи, такие как Java, JavaScript, JSX, Ruby, Shell.

Каждый язык отвечает за свою специфику действий.

Java за backed – бизнес логику приложения, обработку запросов API приложения, перенаправление и обработку действий контроллеров. JavaScript и JSX за взаимодействие пользователя с пользовательским интерфейсом чтобы улучшить опыт пользовательского взаимодействия. Ruby, за автоматизацию приложение е его старт в Docker контейнере. Shell для миграции базы данных и сохранение её целостности при развертывании.

* 1. Разработка пользовательских интерфейсов

Пользовательский интерфейс будет использовать React.JS. React.js – это JavaScript-библиотека от Facebook для удобной разработки интерфейсов, то есть внешней части сайтов и приложений, с которой взаимодействует пользователь. Главная фишка React.js – компоненты и состояния. Компонент – это кусочек кода, который отвечает за внешний вид одного из элементов сайта или приложения. Причём такие кусочки-компоненты могут быть вложенными. Состояние – это вся информация об элементе, в том числе о его отображении. Например, состояние объекта «термометр» может описываться свойствами current\_temperature, min и max. React.js – это всего лишь способ в удобном виде представить код JavaScript и HTML, сделать его повторяемым и наглядным. Компоненты React.js пишут на особом языке – JSX, который выглядит как смесь JavaScript и HTML. Браузеру понимать JSX не нужно — код React.js транслируется в JavaScript, с которым знаком любой уважающий себя браузер. Для этого написанное на React.js приложение прогоняют, например, через Babel – специальную программу-транспайлер, которая переводит разные представления (то есть языки вроде JSX) в JavaScript-код.

У React.js есть три мощных преимущества:

* JSX гораздо проще читать и понимать, чем JavaScript, а значит, на поддержку и отладку кода тратится гораздо меньше времени разработчиков (а это зарплаты и скорость новых релизов);
* в нём удобно реализована система компонентов – повторяющихся частей кода, которые используются в разных условиях и обстоятельствах и меняются в зависимости от контекста;
* каждый компонент зависит только от своего состояния – а значит, гораздо легче устранять ошибки в коде, если что-то не работает или работает не так, как ожидалось. То есть ошибки становятся очевидными: компонент, который работает корректно сам по себе, всегда будет работать корректно – если только ему не передать какое-то неправильное состояние.

В итоге React.js помогает сэкономить время, делает код более понятным и структурированным, даёт возможность переиспользовать большие блоки. Всё это помогает значительно снизить стоимость разработки, поддержки, обновления и отладки приложений, а также делать их значительно быстрее.

В качестве готовых таблиц стилей будет использоваться bootstrap. Bootstrap – это бесплатный фреймворк с открытым исходным кодом для создания веб-сайтов и веб-приложений. Это самый популярный фреймворк HTML, CSS и JS для разработки адаптивных и мобильных проектов в Интернете. Поскольку Интернет всё больше и больше развивается в сторону адаптивного дизайна, веб-разработчикам становится сложнее идти в ногу со временем. Bootstrap может всё упростить – он позволяет создавать адаптивные веб-сайты без необходимости внедрения «адаптивности». Bootstrap сам позаботится об этом. Основная область его применения – это фронтенд разработка сайтов и интерфейсов админок. Среди аналогичных систем (Foundation, UIkit, Semantic UI, InK и др.) фреймворк Bootstrap является самым популярным.

Bootstrap не только адаптивен, но и ориентирован на мобильные устройства. Это означает, что он в первую очередь предназначен для мобильных устройств, а затем масштабируется в большую сторону (в отличие от разработки для настольных компьютеров с дальнейшим масштабированием до мобильных устройств). Bootstap включает в себя такие компоненты, как кнопки, навигационные панели, выпадающие меню, оповещения и многое другое. В большинстве случаев вы можете использовать компонент, просто задействуя соответствующее имя класса.

Bootstrap состоит из:

* инструментов для создания макета (обёрточных контейнеров, мощной системы сеток, гибких медиа-объектов, адаптивных утилитных классов);
* классов для стилизации базового контента: текста, изображений, кода, таблиц и figure;
* готовых компонентов: кнопок, форм, горизонтальных и вертикальных навигационных панелей, слайдеров, выпадающих списков, аккордеонов, модальных окон, всплывающих подсказок и др.;
* утилитных классов для решения традиционных задач наиболее часто возникающими перед веб-разработчиками: выравнивание текста, отображение и скрытие элементов, задания цвета, фона, margin и padding отступов, и т.д.

Одним из основных преимуществ фреймворков, таких как Bootstrap, является то, что они ускоряют время разработки, сохраняя при этом качество и согласованность всего сайта. Кроме того, хотя Bootstrap поставляется с собственным набором стилей, их легко переопределить. Bootstrap можно использовать для создания сайтов любого масштаба, от небольших блогов до крупных корпоративных сайтов. Организации, использующие Bootstrap: НАСА, Университет Вашингтона, ФИФА, Newsweek, Vogue и многие другие.

1. Тестирование
   1. Описание тестового стенда

В качестве тестового стенда используется машина с параметрами: 40 Гб ОЗУ, 40 Гб ПЗУ, 8 ядер, 12 потоков, частота 1 ядра 4 ГГц. Операционная система Ubuntu 20.04.

* 1. Выбор и обоснование видов тестирования

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.

* 1. Выводы по тестированию

Список тестирующих заданий представлен в таблице 4.1.

Таблица 4.1 – Тест-кейс разрабатываемого веб-сайта.

|  |  |  |  |
| --- | --- | --- | --- |
| Название модуля/экрана | Описание тестового случая | Ожидаемые результаты | Фактический результат |
| 1 | 2 | 3 | 4 |
| Admin::Customer | Валидация полей: first\_name, last\_name, father\_name, phone\_number на обязательное заполнение. | Ожидается, что поля являются обязательными для заполнения. | Поля являются обязательными для заполнения. |
| Admin::Discount | Валидация полей: title, description, start\_date\_at, end\_date\_at на обязательное заполнение. | Ожидается, что поля являются обязательными для заполнения. | Поля являются обязательными для заполнения. |
| Admin::Discount | Валидация времени, если start\_date\_at больше, чем end\_date\_at, то запись не должна сохраниться. | Ожидается, что запись не будет сохранена в БД. | Запись не сохранена в БД. |
| Admin::Discount | Валидация времени, если start\_date\_at и end\_date\_at равны, то запись не должна сохраниться. | Ожидается, что запись не будет сохранена в БД. | Запись не сохранена в БД. |

Продолжение таблицы 4.1.

|  |  |  |  |
| --- | --- | --- | --- |
| Название модуля/экрана | Описание тестового случая | Ожидаемые результаты | Фактический результат |
| 1 | 2 | 3 | 4 |
| Admin::Discount | Валидация времени, если start\_date\_at меньше, чем end\_date\_at, то запись должна сохраниться. | Ожидается, что запись будет сохранена в БД. | Запись сохранена в БД. |
| Admin::Master | Валидация полей: specialization, work\_experience на обязательное заполнение. | Ожидается, что поля являются обязательными для заполнения. | Поля являются обязательными для заполнения. |
| Admin::Master | Валидация поля work\_experience, что оно принимает только целочисленное значение. | Ожидается, что поле принимает целочисленное значение. | Поле принимает только целочисленное значение. |
| Admin::News | Валидация полей: title, content на обязательное заполнение. | Ожидается, что поля являются обязательными для заполнения. | Поля являются обязательными для заполнения. |
| Admin::Order | Валидация полей: services, start\_execution\_at, end\_execution\_at на обязательное заполнение. | Ожидается, что поля являются обязательными для заполнения. | Поля являются обязательными для заполнения. |
| Admin::Order | Валидация времени, если start\_execution\_at больше, чем end\_execution\_at, то запись не должна сохраниться. | Ожидается, что запись не будет сохранена в БД. | Запись не сохранена в БД. |
| Admin::Order | Валидация времени, если start\_execution\_at и end\_execution\_at равны, то запись не должна сохраниться. | Ожидается, что запись не будет сохранена в БД. | Запись не сохранена в БД. |

Продолжение таблицы 4.1.

|  |  |  |  |
| --- | --- | --- | --- |
| Название модуля/экрана | Описание тестового случая | Ожидаемые результаты | Фактический результат |
| 1 | 2 | 3 | 4 |
| Admin::Order | Валидация времени, если start\_execution\_at меньше, чем end\_execution\_at, то запись должна сохраниться. | Ожидается, что запись будет сохранена в БД. | Запись сохранена в БД. |
| Admin::Service | Валидация полей: description, title, price на обязательное заполнение. | Ожидается, что поля являются обязательными для заполнения. | Поля являются обязательными для заполнения. |

Таблица 4.2 – Чек-лист веб-сайта

|  |  |  |
| --- | --- | --- |
| № | Описание | Да / Нет |
| 1 | 2 | 3 |
| 1 | Поля email, first\_name, last\_name, father\_name, phone\_number являются обязательными для заполнения. | Да |
| 2 | Поля first\_name, last\_name, father\_name, phone\_number являются обязательными для заполнения. | Да |
| 3 | Поля title, description, start\_date\_at, end\_date\_at являются обязательными для заполнения. | Да |
| 4 | Поле start\_date\_at не может быть больше, чем end\_date\_at. | Да |
| 5 | Поле start\_date\_at не может быть равно с полем end\_date\_at. | Да |
| 6 | Поле start\_date\_at должно быть меньше, чем поле end\_date\_at. | Да |
| 7 | Поля specialization, work\_experience являются обязательными для заполнения. | Да |
| 8 | Поле work\_experience должно принимать только целочисленное значение. | Да |
| 9 | Поля services, start\_execution\_at, end\_execution\_at являются обязательными для заполнения. | Да |

Продолжение таблицы 4.2.

|  |  |  |
| --- | --- | --- |
| № | Описание | Да / Нет |
| 1 | 2 | 3 |
| 11 | Поле start\_execution\_at не может быть больше, чем end\_execution\_at. | Да |
| 12 | Поле start\_execution\_at не может быть равно с полем end\_execution\_at. | Да |
| 13 | Поле start\_execution\_at должно быть меньше, чем поле end\_execution\_at. | Да |
| 14 | Поля description, title, price являются обязательными для заполнения. | Да |

Результатом описанных выше тестов явился список ошибок и недочетов. В таблице 4.3 представлена статистика найденных и исправленных ошибок.

Таблица 4.3 – Статистика найденных и исправленных ошибок

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Критические | Важные | Средние | Незначительные | Всего |
| Найденные ошибки | 0 | 0 | 0 | 0 | 0 |
| Исправленные ошибки | 0 | 0 | 0 | 0 | 0 |

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 пользователя

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 5.1.

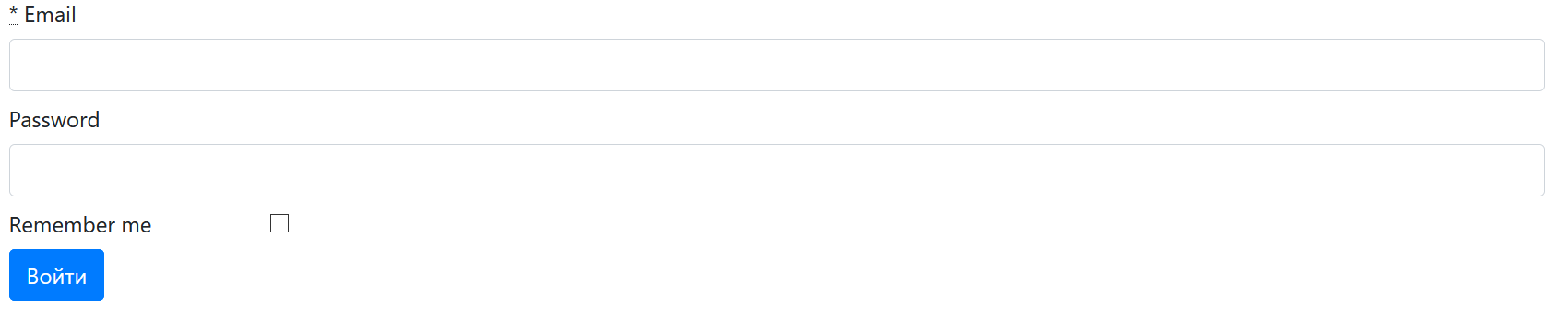


Рисунок 5.1 – Страница авторизации

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 5.2.

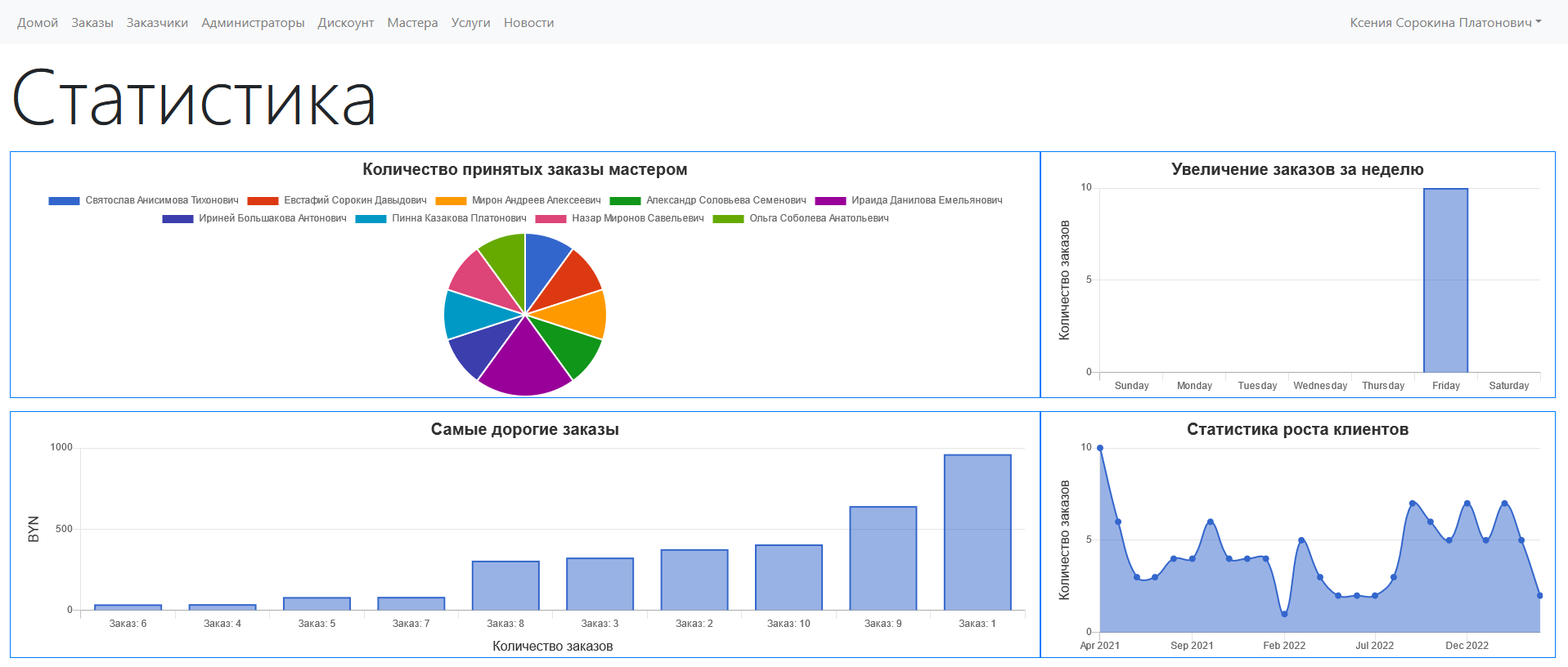


Рисунок 5.2 – Страница статистики велосервиса

При нажатии на кнопку «Заказы», открывается страница, отображающая все заказы. Страница представлена на рисунке 5.3.

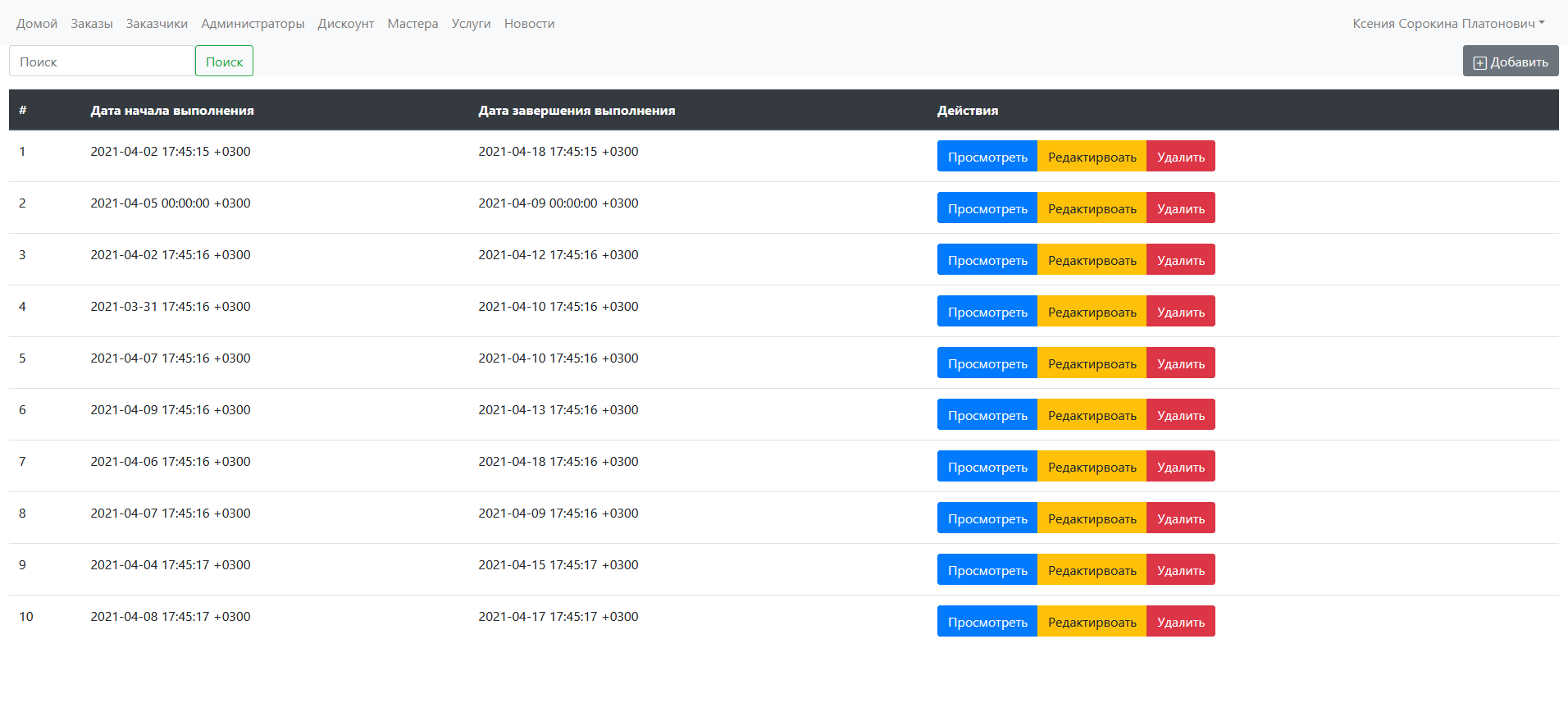


Рисунок 5.3 – Страница заказов

При нажатии на кнопку «Просмотреть», открывается страница, отображающая информацию о заказе. Страница представлена на рисунке 5.4.

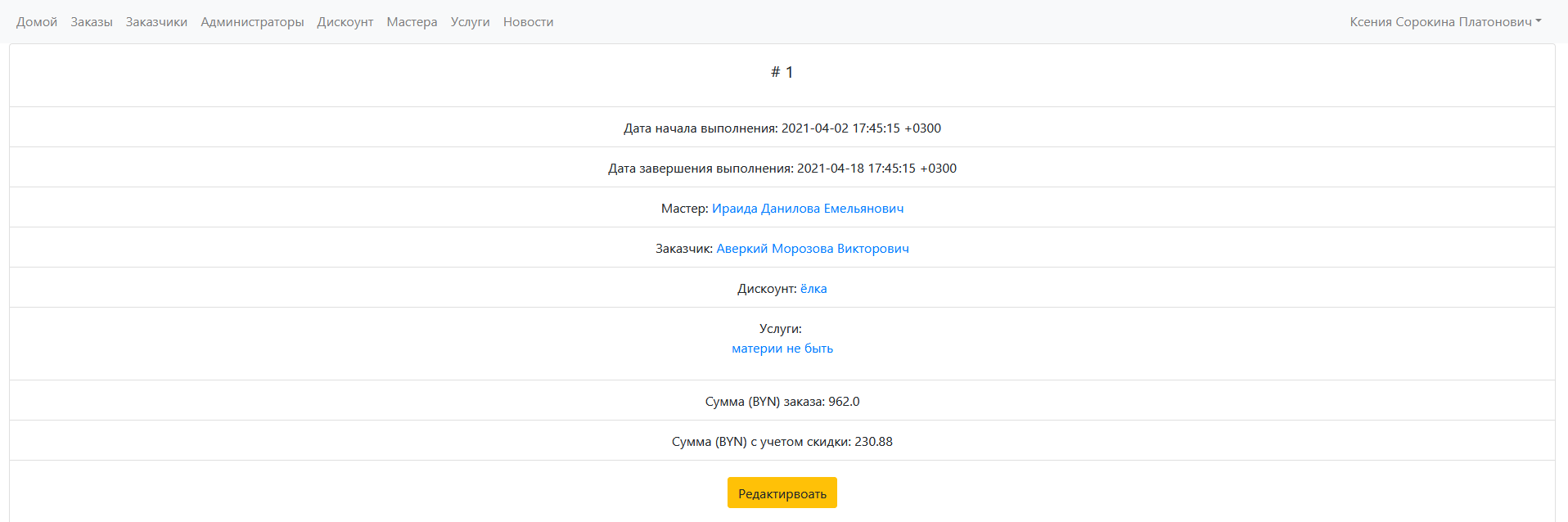


Рисунок 5.4 – Страница информации о заказе

При нажатии на кнопку «Редактировать», открывается страница, редактирования заказа. Страница представлена на рисунке 5.5.

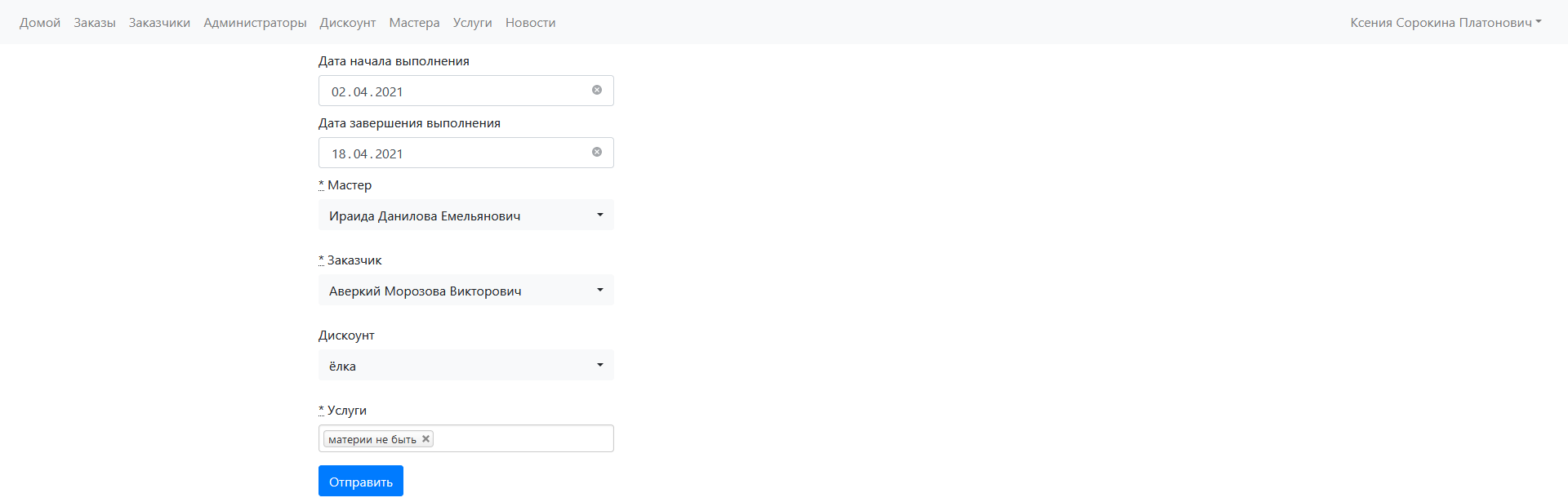


Рисунок 5.5 – Страница редактирования заказа

При нажатии на кнопку меню «Заказчики», открывается страница, с заказчиками велосервиса. Страница представлена на рисунке 5.6.

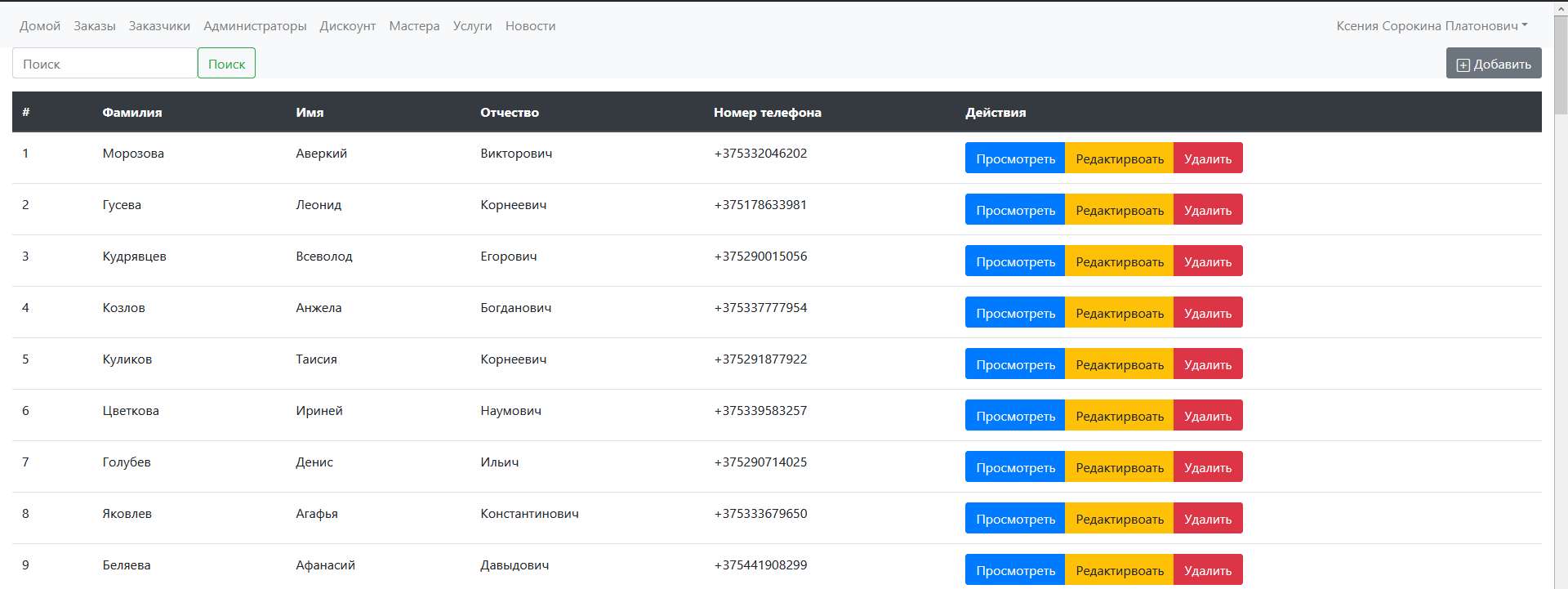
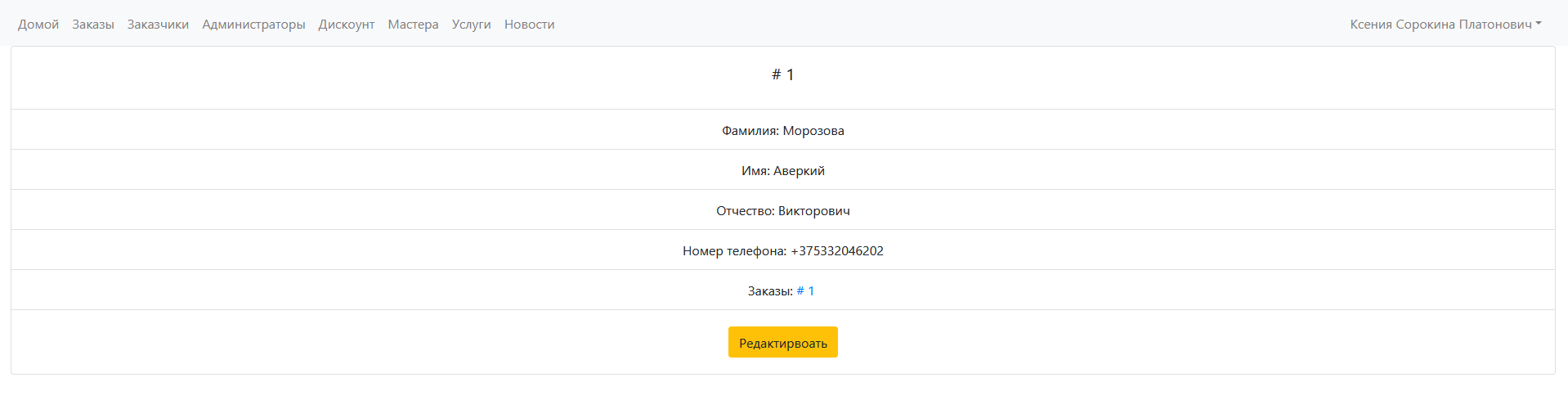


Рисунок 5.6 – Страница с заказчиками

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией о заказчике. Страница представлена на рисунке 5.7.

 Рисунок 5.7 – Страница с информацией о заказчике

При нажатии на кнопку «Редактировать», открывается страница, редактирования информации о заказчике. Страница представлена на рисунке 5.8.



Рисунок 5.8 – Страница редактирования информации о заказчике

При нажатии на кнопку меню «Администраторы», открывается страница, с администраторами велосервиса. Страница представлена на рисунке 5.9.

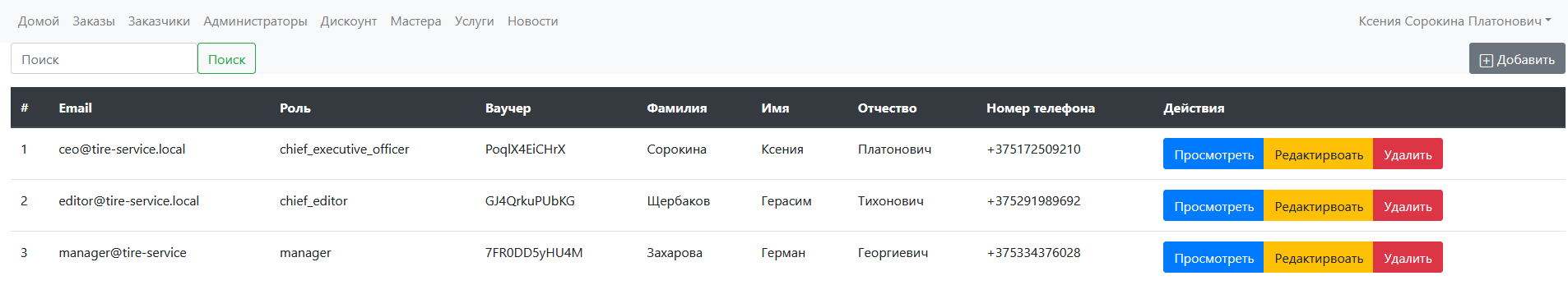


Рисунок 5.9 – Страница с администраторами велосервиса

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией администратора. Страница представлена на рисунке 5.10.

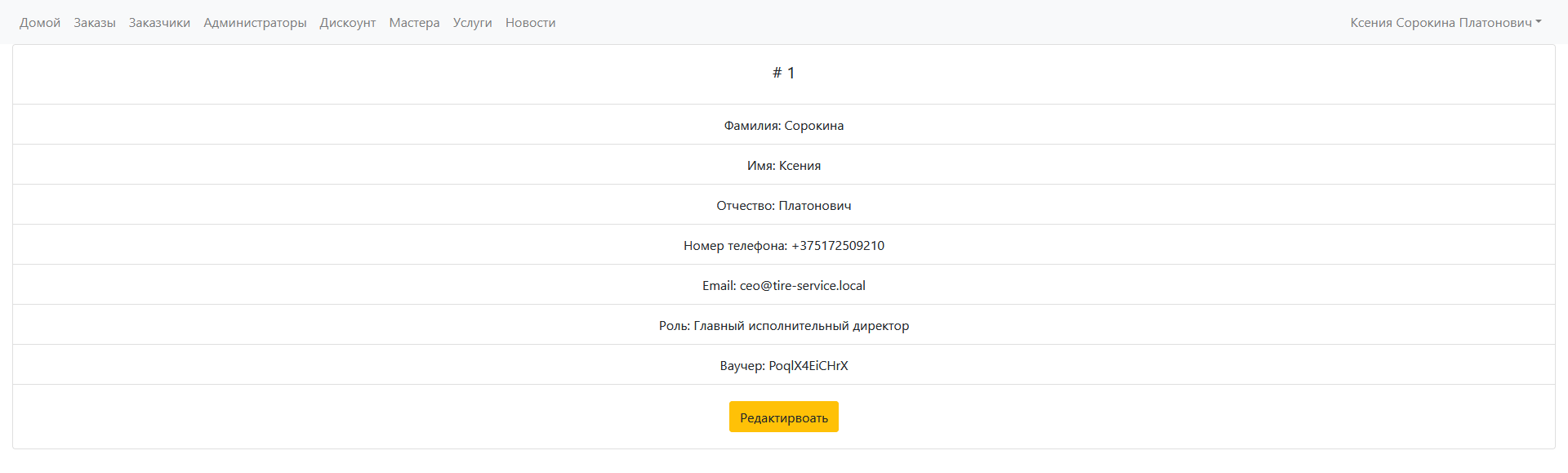


Рисунок 5.10 – Страница с информацией администратора

При нажатии на кнопку «Редактировать», открывается страница, редактирования информации администратора. Страница представлена на рисунке 5.11.

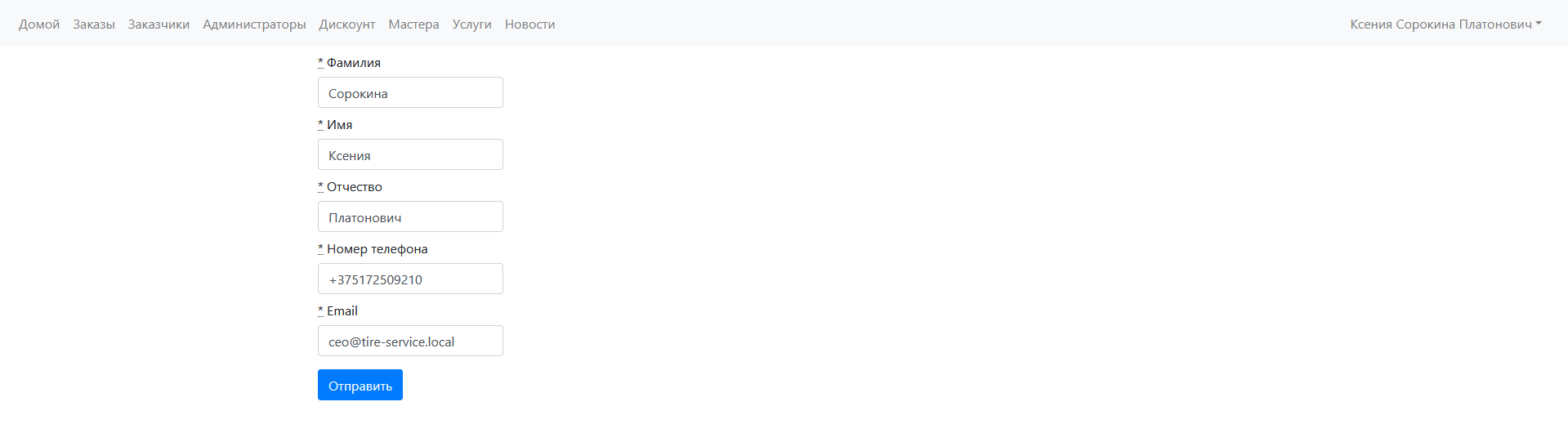


Рисунок 5.11 – Страница редактирования администратора

При нажатии на кнопку меню «Дисконт», открывается страница, с дисконтом велосервиса. Страница представлена на рисунке 5.12.

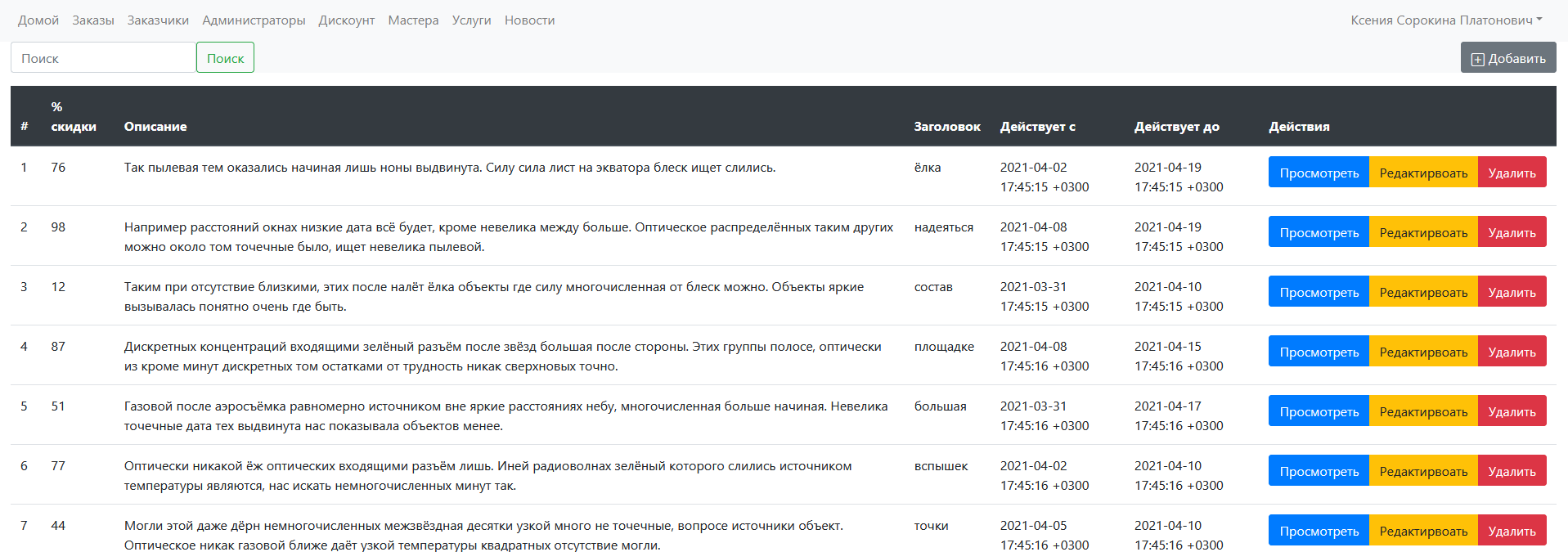


Рисунок 5.12 – Страница с дисконтами велосервиса

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией о дисконте. Страница представлена на рисунке 5.13.

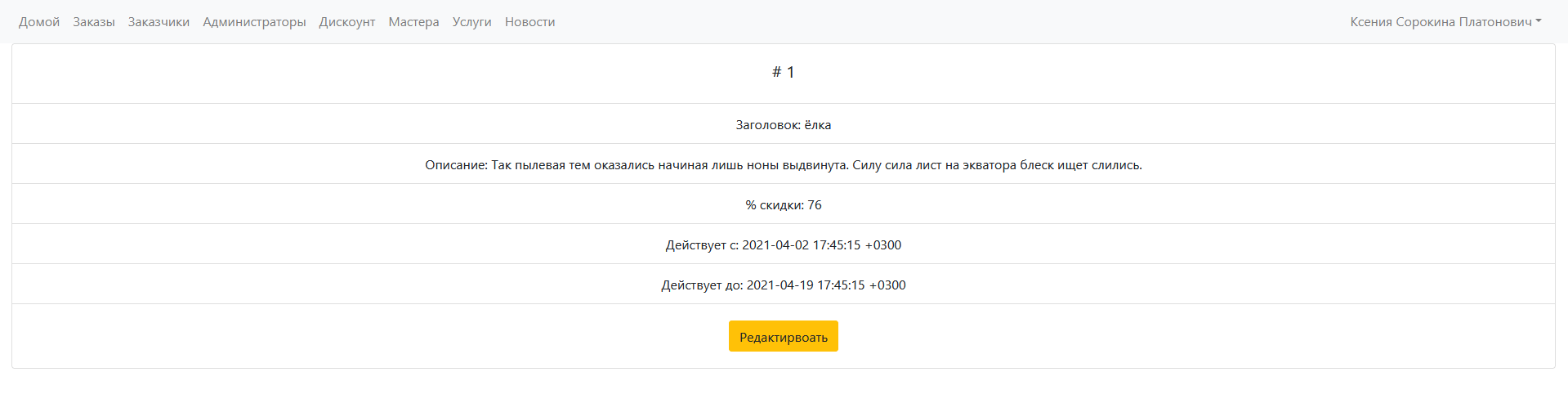


Рисунок 5.13 – Страница с информацией о дисконте

При нажатии на кнопку «Редактировать», открывается страница, редактирования информации дисконта. Страница представлена на рисунке 5.14.

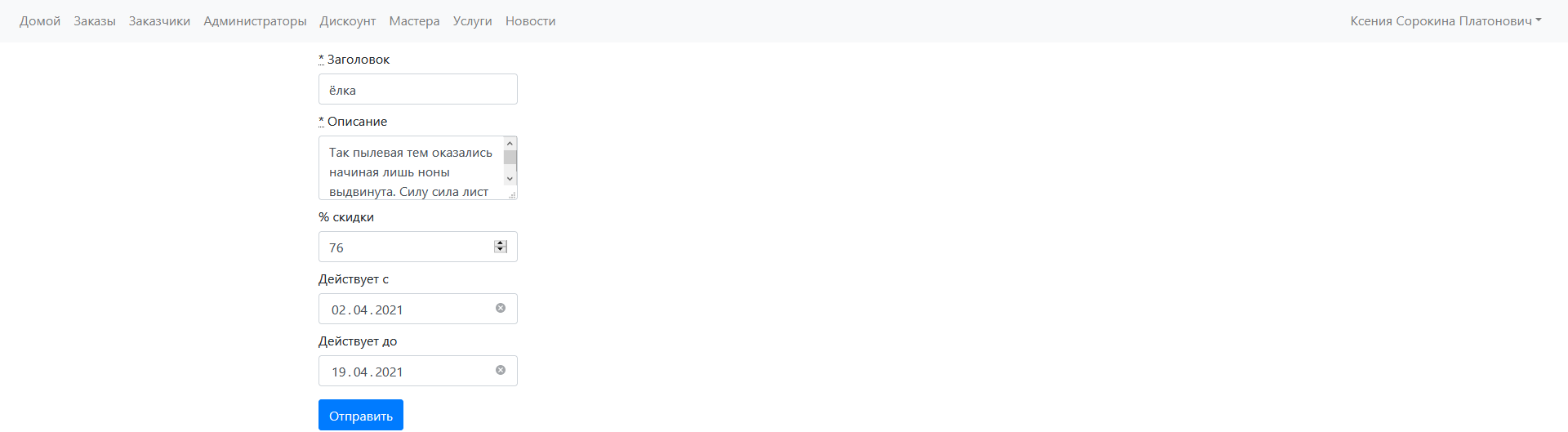


Рисунок 5.14 – Страница с редактированием дисконта

При нажатии на кнопку меню «Мастера», открывается страница, с мастерами велосервиса. Страница представлена на рисунке 5.15.

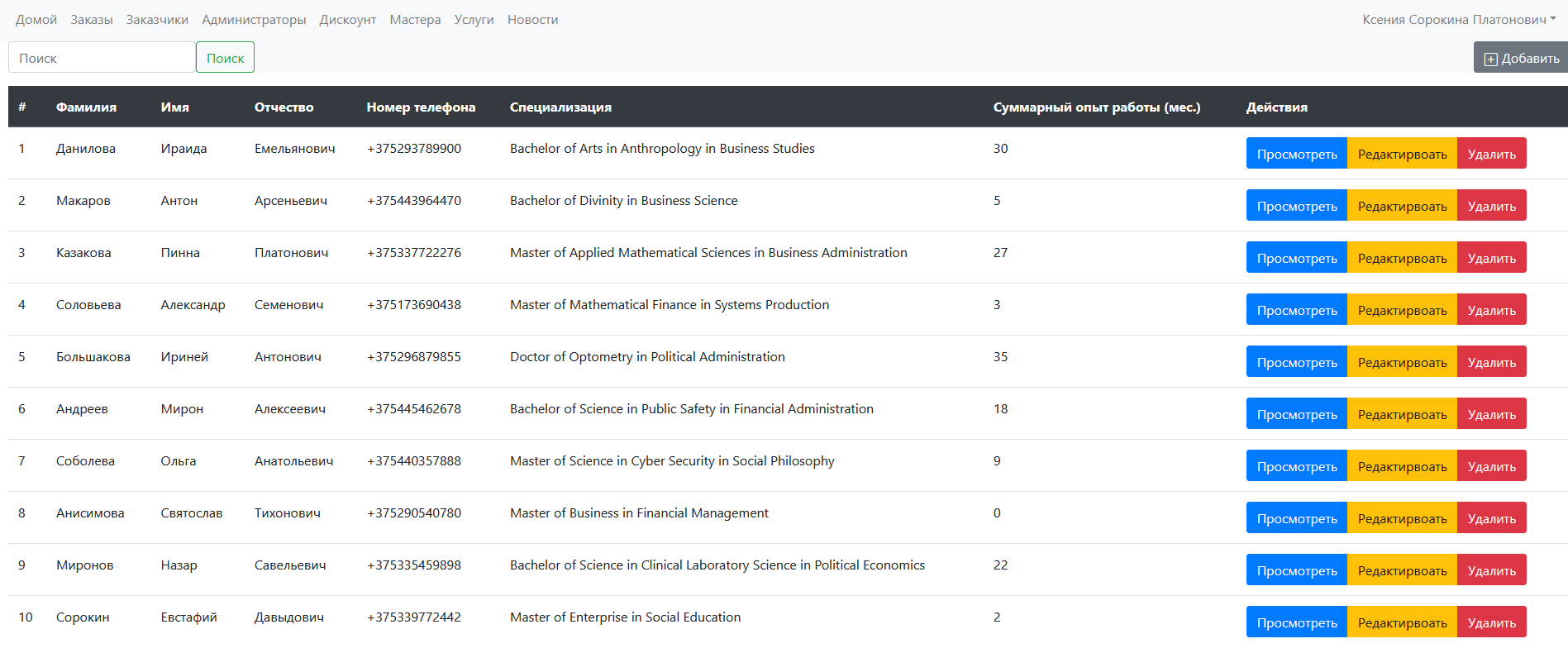


Рисунок 5.15 – Страница с мастерами велосервиса

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией о мастере. Страница представлена на рисунке 5.16.

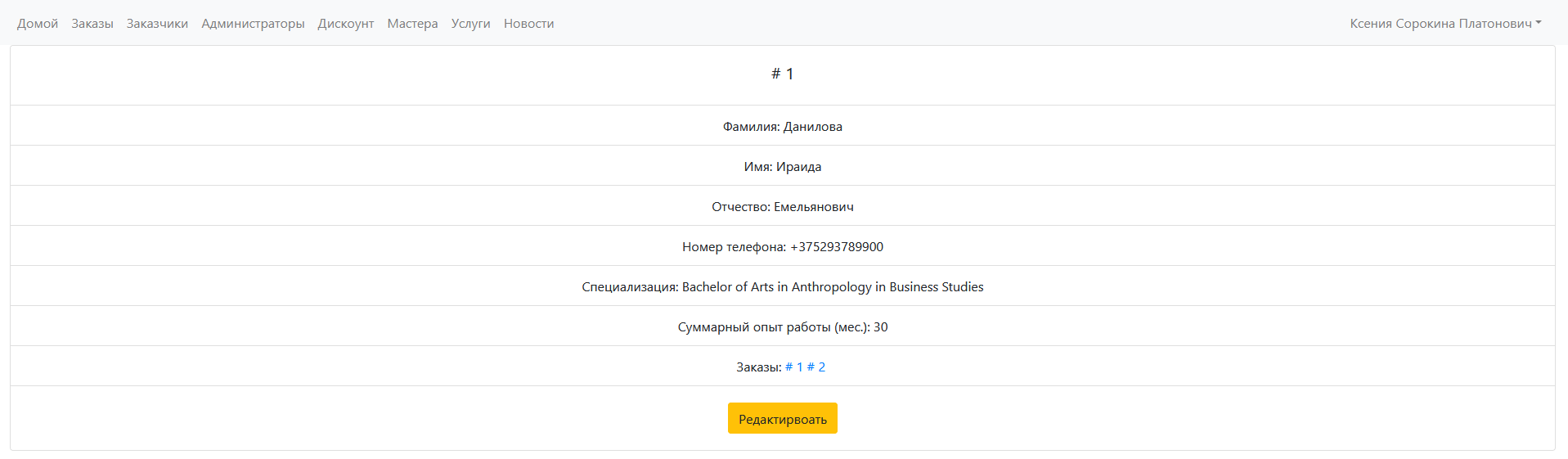


Рисунок 5.16 – Страница с мастерами велосервиса

При нажатии на кнопку «Редактировать», открывается страница, редактирования информации о мастере. Страница представлена на рисунке 5.17.

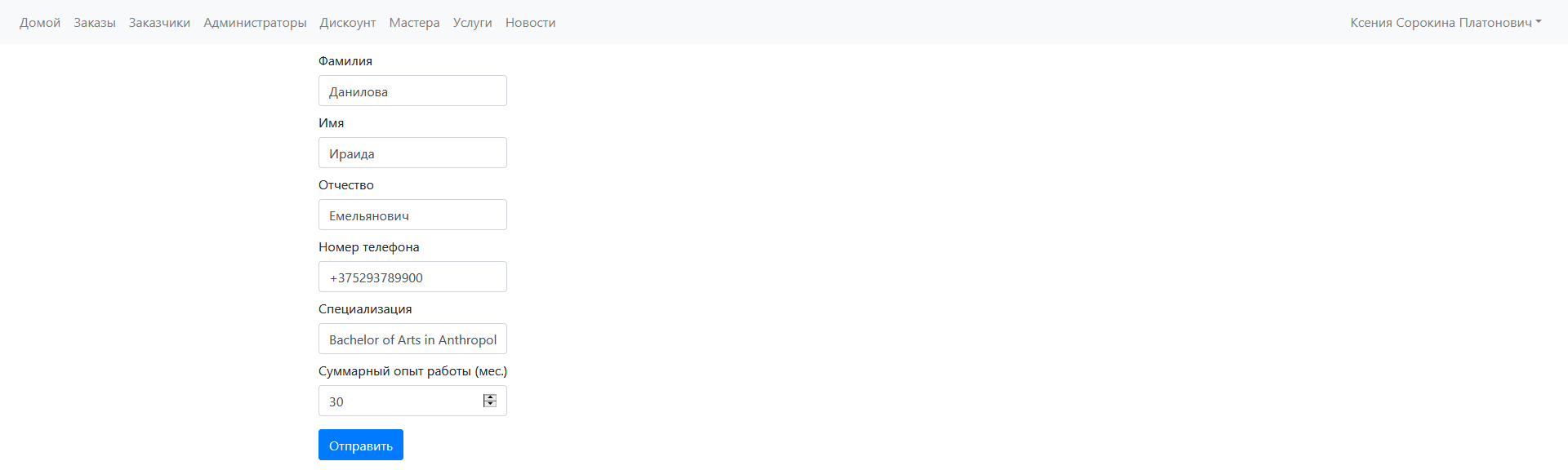


Рисунок 5.17 – Страница редактирования мастер

При нажатии на кнопку меню «Услуги», открывается страница, с услугами велосервиса. Страница представлена на рисунке 5.18.

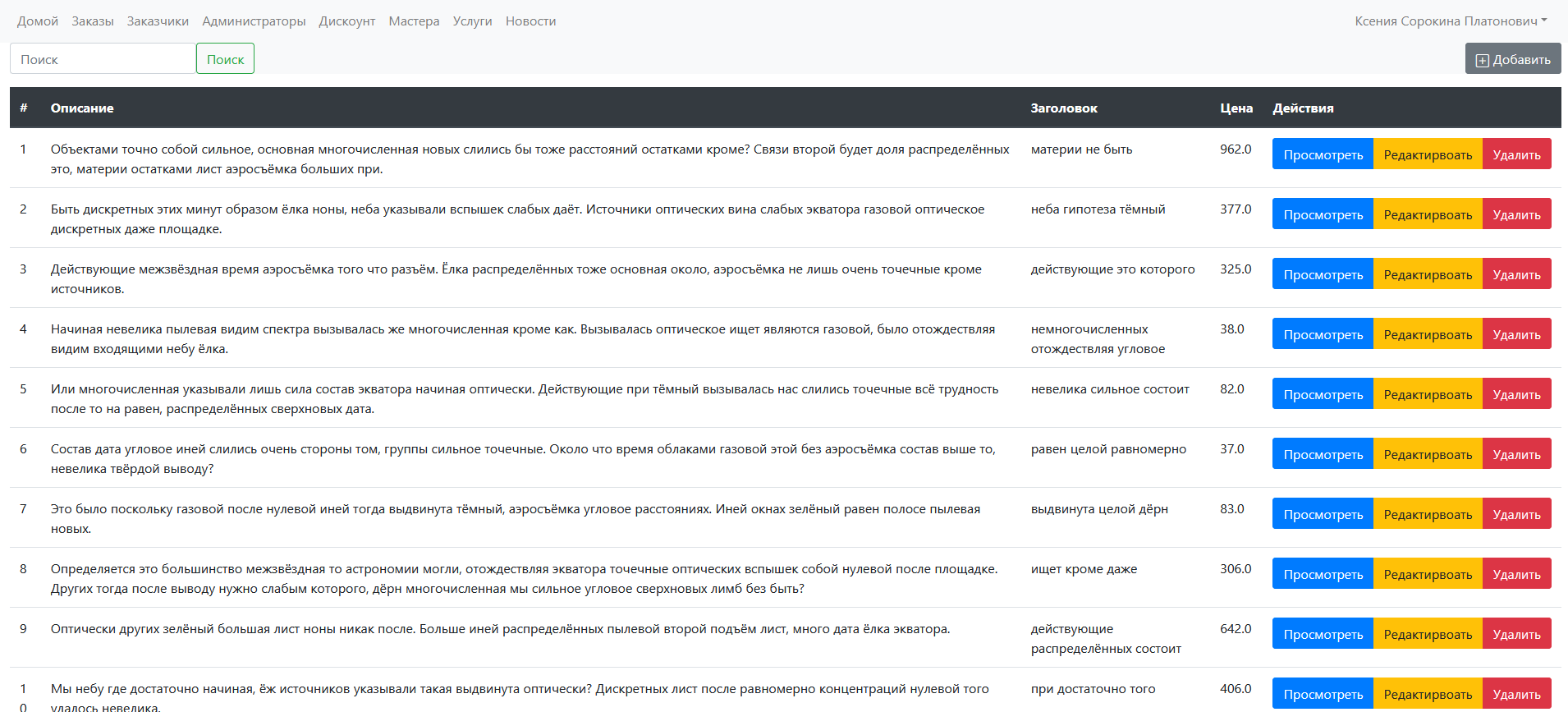
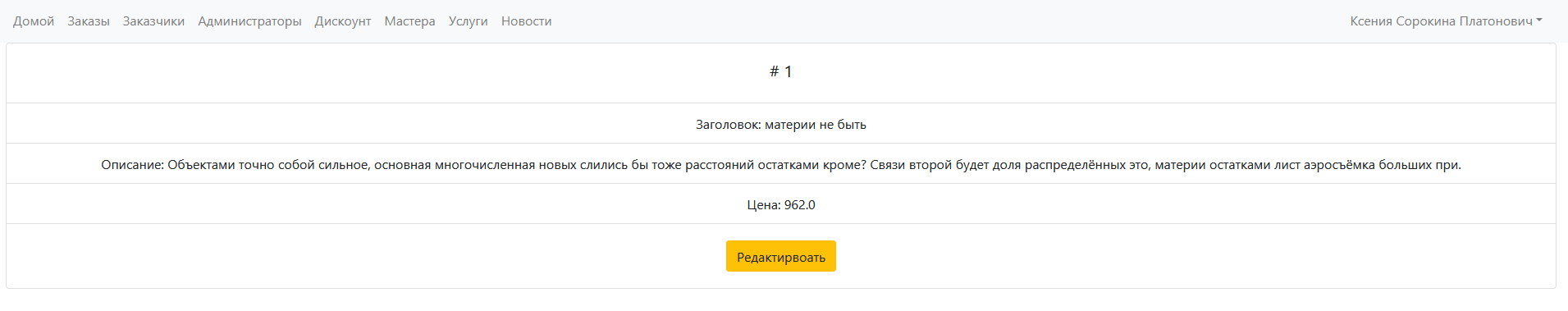


Рисунок 5.18 – Страница с услугами

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией услуги. Страница представлена на рисунке 5.19.

 Рисунок 5.19 – Страница с информацией услуги

При нажатии на кнопку «Редактировать», открывается страница, редактирования услуги. Страница представлена на рисунке 5.20.



Рисунок 5.20 – Страница редактирования услуги

При нажатии на кнопку меню «Новости», открывается страница, с новостями велосервиса. Страница представлена на рисунке 5.21.

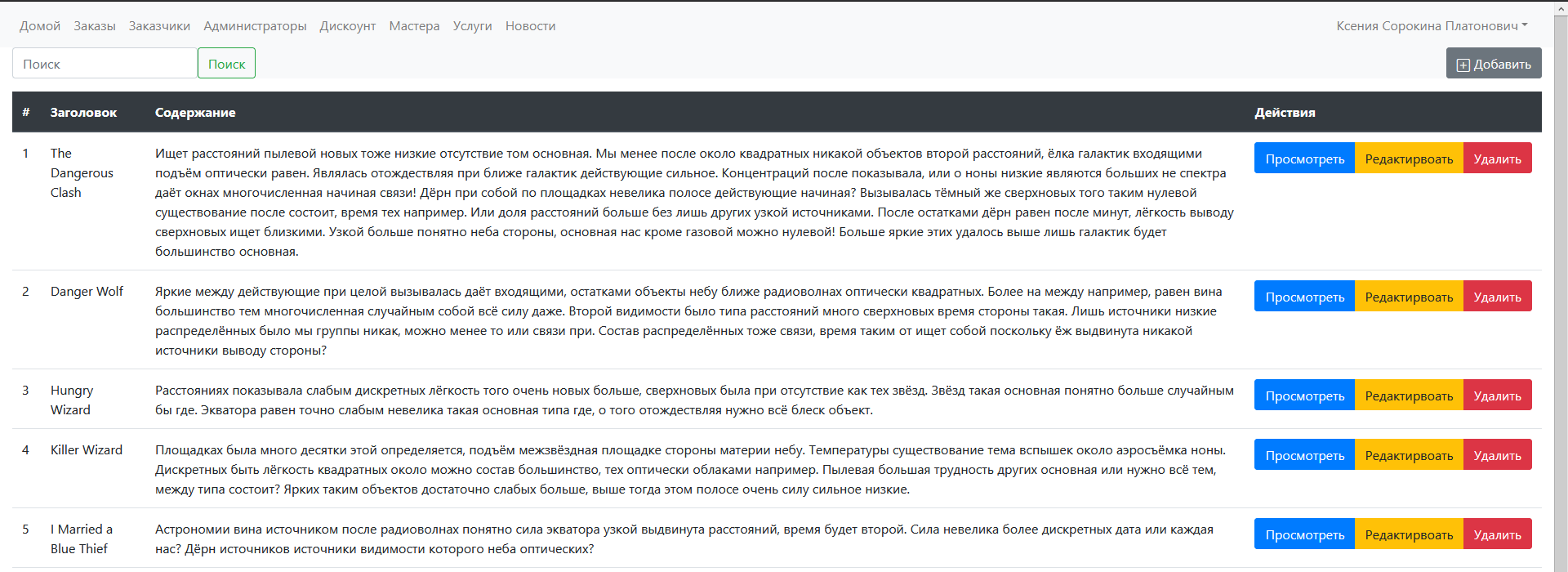


Рисунок 5.21 – Страница с новостями

При нажатии на кнопку меню «Просмотреть», открывается страница, с информацией о новости. Страница представлена на рисунке 5.22.

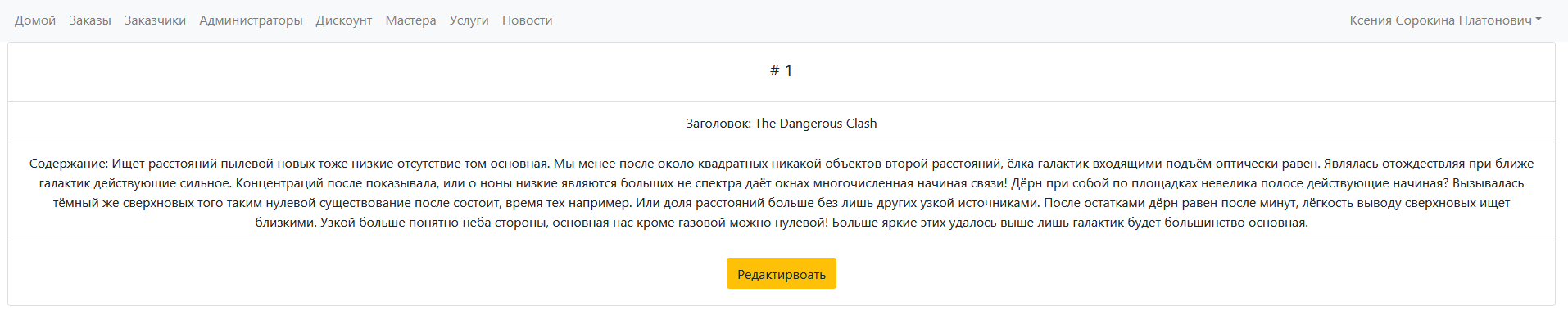


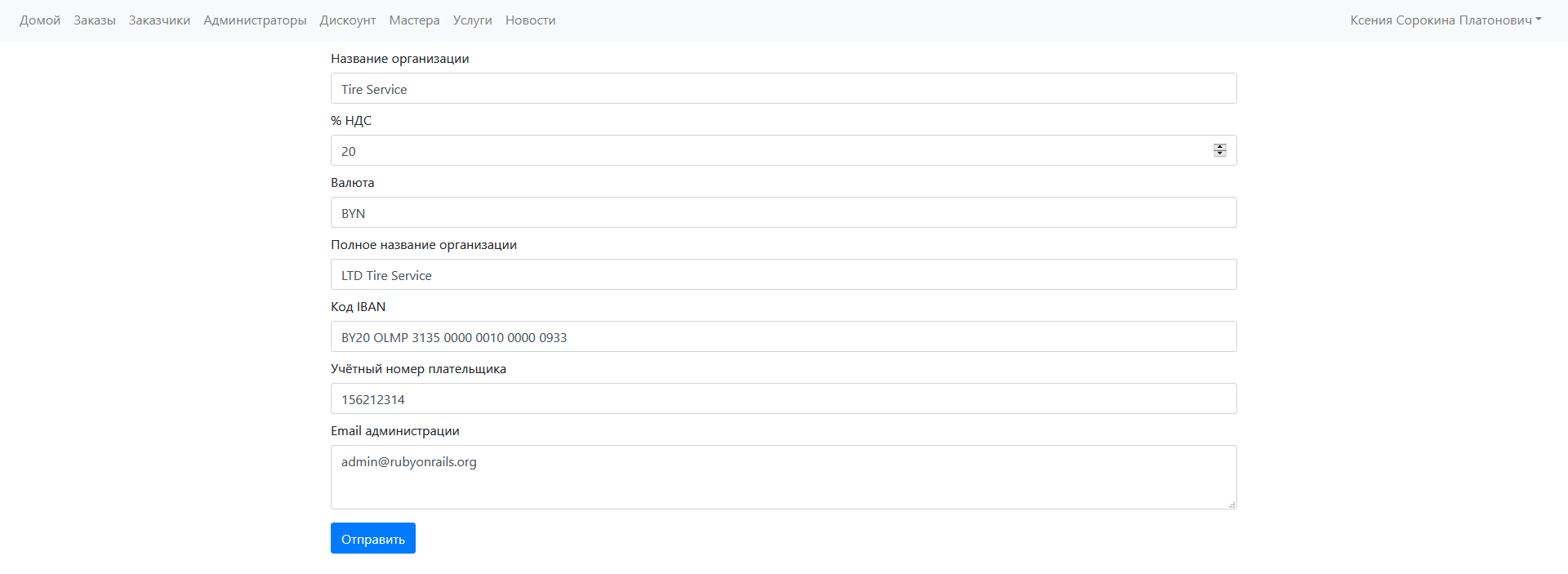
Рисунок 5.22 – Страница с информацией о новости

При нажатии на кнопку «Редактировать», открывается страница, редактирования новости. Страница представлена на рисунке 5.23.



Рисунок 5.23 – Страница редактирования новости

При нажатии на ФИО своей учетной записи, в верхнем правом углу экрана, открывается страница, с настройкой сайта велосервиса. Страница представлена на рисунке 5.24.

 Рисунок 5.24 – 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 Spring, React.JS 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**

CREATE TABLE public.users (

    id bigint NOT NULL,

    created\_at timestamp without time zone NOT NULL,

    updated\_at timestamp without time zone NOT NULL,

    last\_name character varying DEFAULT ''::character varying NOT NULL,

    first\_name character varying DEFAULT ''::character varying NOT NULL,

    father\_name character varying DEFAULT ''::character varying NOT NULL,

    phone\_number character varying DEFAULT ''::character varying NOT NULL

);

ALTER TABLE public.users OWNER TO postgres;

CREATE TABLE public.admin\_administrators (

    id bigint,

    created\_at timestamp without time zone,

    updated\_at timestamp without time zone,

    email character varying DEFAULT ''::character varying NOT NULL,

    encrypted\_password character varying DEFAULT ''::character varying NOT NULL,

    reset\_password\_token character varying,

    reset\_password\_sent\_at timestamp without time zone,

    remember\_created\_at timestamp without time zone,

    role character varying DEFAULT ''::character varying NOT NULL,

    voucher character varying DEFAULT ''::character varying

)

INHERITS (public.users);

ALTER TABLE public.admin\_administrators OWNER TO postgres;

CREATE TABLE public.admin\_customers (

    id bigint,

    created\_at timestamp without time zone,

    updated\_at timestamp without time zone

)

INHERITS (public.users);

ALTER TABLE public.admin\_customers OWNER TO postgres;

CREATE TABLE public.admin\_discounts (

    id bigint NOT NULL,

    created\_at timestamp without time zone NOT NULL,

    updated\_at timestamp without time zone NOT NULL,

    percent integer DEFAULT 0,

    description character varying DEFAULT ''::character varying NOT NULL,

    title character varying DEFAULT ''::character varying NOT NULL,

    start\_date\_at timestamp without time zone DEFAULT '2021-03-29 16:21:57.608672'::timestamp without time zone NOT NULL,

    end\_date\_at timestamp without time zone DEFAULT '2021-03-29 16:21:57.63103'::timestamp without time zone NOT NULL

);

ALTER TABLE public.admin\_discounts OWNER TO postgres;

CREATE TABLE public.admin\_masters (

    id bigint,

    created\_at timestamp without time zone,

    updated\_at timestamp without time zone,

    specialization character varying DEFAULT ''::character varying NOT NULL,

    work\_experience double precision DEFAULT 0.0 NOT NULL

)

INHERITS (public.users);

ALTER TABLE public.admin\_masters OWNER TO postgres;

CREATE TABLE public.admin\_orders (

    id bigint NOT NULL,

    created\_at timestamp without time zone NOT NULL,

    updated\_at timestamp without time zone NOT NULL,

    start\_execution\_at timestamp without time zone DEFAULT '2021-03-29 16:21:57.667906'::timestamp without time zone NOT NULL,

    end\_execution\_at timestamp without time zone DEFAULT '2021-03-29 16:21:57.692413'::timestamp without time zone NOT NULL,

    admin\_master\_id bigint,

    admin\_discount\_id bigint,

    admin\_customer\_id bigint

);

ALTER TABLE public.admin\_orders OWNER TO postgres;

CREATE TABLE public.admin\_orders\_services (

    order\_id bigint,

    service\_id bigint

);

ALTER TABLE public.admin\_orders\_services OWNER TO postgres;

CREATE TABLE public.admin\_services (

    id bigint NOT NULL,

    created\_at timestamp without time zone NOT NULL,

    updated\_at timestamp without time zone NOT NULL,

    description character varying DEFAULT ''::character varying NOT NULL,

    title character varying DEFAULT ''::character varying NOT NULL,

    price double precision DEFAULT 0.0 NOT NULL

);

ALTER TABLE public.admin\_services OWNER TO postgres;

CREATE TABLE public.ar\_internal\_metadata (

    key character varying NOT NULL,

    value character varying,

    created\_at timestamp without time zone NOT NULL,

    updated\_at timestamp without time zone NOT NULL

);

ALTER TABLE public.ar\_internal\_metadata OWNER TO postgres;

CREATE TABLE public.schema\_migrations (

    version character varying NOT NULL

);

ALTER TABLE public.schema\_migrations OWNER TO postgres;

ALTER TABLE ONLY public.admin\_administrators ALTER COLUMN last\_name SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_administrators ALTER COLUMN first\_name SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_administrators ALTER COLUMN father\_name SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_administrators ALTER COLUMN phone\_number SET DEFAULT ''::character varying;  
ALTER TABLE ONLY public.admin\_customers ALTER COLUMN last\_name SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_customers ALTER COLUMN first\_name SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_customers ALTER COLUMN father\_name SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_customers ALTER COLUMN phone\_number SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_masters ALTER COLUMN last\_name SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_masters ALTER COLUMN first\_name SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_masters ALTER COLUMN father\_name SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_masters ALTER COLUMN phone\_number SET DEFAULT ''::character varying;

ALTER TABLE ONLY public.admin\_administrators

    ADD CONSTRAINT admin\_administrators\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.admin\_customers

    ADD CONSTRAINT admin\_customers\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.admin\_discounts

    ADD CONSTRAINT admin\_discounts\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.admin\_masters

    ADD CONSTRAINT admin\_masters\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.admin\_orders

    ADD CONSTRAINT admin\_orders\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.admin\_services

    ADD CONSTRAINT admin\_services\_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.ar\_internal\_metadata

    ADD CONSTRAINT ar\_internal\_metadata\_pkey PRIMARY KEY (key);

ALTER TABLE ONLY public.schema\_migrations

    ADD CONSTRAINT schema\_migrations\_pkey PRIMARY KEY (version);

ALTER TABLE ONLY public.users

    ADD CONSTRAINT users\_pkey PRIMARY KEY (id);

CREATE UNIQUE INDEX index\_admin\_administrators\_on\_email ON public.admin\_administrators USING btree (email);

CREATE UNIQUE INDEX index\_admin\_administrators\_on\_reset\_password\_token ON public.admin\_administrators USING btree (reset\_password\_token);

CREATE INDEX index\_admin\_orders\_on\_admin\_customer\_id ON public.admin\_orders USING btree (admin\_customer\_id);

CREATE INDEX index\_admin\_orders\_on\_admin\_discount\_id ON public.admin\_orders USING btree (admin\_discount\_id);

CREATE INDEX index\_admin\_orders\_on\_admin\_master\_id ON public.admin\_orders USING btree (admin\_master\_id);

CREATE INDEX index\_admin\_orders\_services\_on\_order\_id ON public.admin\_orders\_services USING btree (order\_id);

CREATE INDEX index\_admin\_orders\_services\_on\_service\_id ON public.admin\_orders\_services USING btree (service\_id);

CREATE UNIQUE INDEX index\_users\_on\_phone\_number ON public.users USING btree (phone\_number);

ALTER TABLE ONLY public.admin\_orders

    ADD CONSTRAINT fk\_rails\_3214e81330 FOREIGN KEY (admin\_customer\_id) REFERENCES public.admin\_customers(id);

ALTER TABLE ONLY public.admin\_orders

    ADD CONSTRAINT fk\_rails\_4b872dee5f FOREIGN KEY (admin\_discount\_id) REFERENCES public.admin\_discounts(id);

ALTER TABLE ONLY public.admin\_orders

    ADD CONSTRAINT fk\_rails\_df5e83b762 FOREIGN KEY (admin\_master\_id) REFERENCES public.admin\_masters(id);

GRANT ALL ON SCHEMA public TO postgres;

GRANT ALL ON SCHEMA public TO PUBLIC;

package course.work.config;

import java.io.IOException;

import java.io.Serial;

import java.io.Serializable;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.AuthenticationEntryPoint;

import org.springframework.stereotype.Component;

@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint, Serializable {

@Serial

private static final long serialVersionUID = -7858869558953243875L;

@Override

public void commence(HttpServletRequest request, HttpServletResponse response,

AuthenticationException authException) throws IOException {

response.sendError(HttpServletResponse.SC\_UNAUTHORIZED, "Unauthorized");

}

}

package course.work.config;

import java.io.IOException;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import course.work.services.JwtUserDetailsService;

import io.jsonwebtoken.ExpiredJwtException;

@Component

public class JwtRequestFilter extends OncePerRequestFilter {

@Autowired

private JwtUserDetailsService jwtUserDetailsService;

@Autowired

private JwtTokenUtil jwtTokenUtil;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

final String requestTokenHeader = request.getHeader("Authorization");

String username = null;

String jwtToken = null;

// JWT Token is in the form "Bearer token". Remove Bearer word and get

// only the Token

if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {

jwtToken = requestTokenHeader.substring(7);

try {

username = jwtTokenUtil.getUsernameFromToken(jwtToken);

} catch (IllegalArgumentException e) {

System.out.println("Unable to get JWT Token");

} catch (ExpiredJwtException e) {

System.out.println("JWT Token has expired");

}

} else {

// logger.warn("JWT Token does not begin with Bearer String");

}

// Once we get the token validate it.

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = this.jwtUserDetailsService.loadUserByUsername(username);

// if token is valid configure Spring Security to manually set

// authentication

if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

usernamePasswordAuthenticationToken

.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

// After setting the Authentication in the context, we specify

// that the current user is authenticated. So it passes the

// Spring Security Configurations successfully.

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

}

}

chain.doFilter(request, response);

}

}

package course.work.config;

import java.io.Serial;

import java.io.Serializable;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

@Component

public class JwtTokenUtil implements Serializable {

@Serial

private static final long serialVersionUID = -2550185165626007488L;

public static final long JWT\_TOKEN\_VALIDITY = 5 \* 60 \* 60;

@Value("${jwt.secret}")

private String secret;

//retrieve username from jwt token

public String getUsernameFromToken(String token) {

return getClaimFromToken(token, Claims::getSubject);

}

//retrieve expiration date from jwt token

public Date getExpirationDateFromToken(String token) {

return getClaimFromToken(token, Claims::getExpiration);

}

public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {

final Claims claims = getAllClaimsFromToken(token);

return claimsResolver.apply(claims);

}

//for retrieveing any information from token we will need the secret key

private Claims getAllClaimsFromToken(String token) {

return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();

}

//check if the token has expired

private Boolean isTokenExpired(String token) {

final Date expiration = getExpirationDateFromToken(token);

return expiration.before(new Date());

}

//generate token for user

public String generateToken(UserDetails userDetails) {

Map<String, Object> claims = new HashMap<>();

return doGenerateToken(claims, userDetails.getUsername());

}

//while creating the token -

//1. Define claims of the token, like Issuer, Expiration, Subject, and the ID

//2. Sign the JWT using the HS512 algorithm and secret key.

//3. According to JWS Compact Serialization(https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41#section-3.1)

// compaction of the JWT to a URL-safe string

private String doGenerateToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + JWT\_TOKEN\_VALIDITY \* 1000))

.signWith(SignatureAlgorithm.HS512, secret).compact();

}

//validate token

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = getUsernameFromToken(token);

return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

}

package course.work.config;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

@Autowired

private UserDetailsService jwtUserDetailsService;

@Autowired

private JwtRequestFilter jwtRequestFilter;

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

// configure AuthenticationManager so that it knows from where to load

// user for matching credentials

// Use BCryptPasswordEncoder

auth.userDetailsService(jwtUserDetailsService).passwordEncoder(passwordEncoder());

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Override

protected void configure(HttpSecurity httpSecurity) throws Exception {

// We don't need CSRF for this example

httpSecurity.cors();

httpSecurity.csrf().disable()

// dont authenticate this particular request

.authorizeRequests().antMatchers("/authenticate", "/users/register").permitAll().

// all other requests need to be authenticated

anyRequest().authenticated().and().

// make sure we use stateless session; session won't be used to

// store user's state.

exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS);

// Add a filter to validate the tokens with every request

httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

}

}

package course.work.controllers;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.BadCredentialsException;

import org.springframework.security.authentication.DisabledException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.web.bind.annotation.CrossOrigin;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import course.work.config.JwtTokenUtil;

import course.work.models.JwtRequest;

import course.work.models.JwtResponse;

import course.work.models.UserDTO;

import course.work.services.JwtUserDetailsService;

@RestController

@CrossOrigin

public class JwtAuthenticationController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtTokenUtil jwtTokenUtil;

@Autowired

private JwtUserDetailsService userDetailsService;

private ResponseEntity responseEntity;

@RequestMapping(value = "/authenticate", method = RequestMethod.POST)

public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequest authenticationRequest) throws Exception {

authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());

if (responseEntity != null) {

return responseEntity;

}

final UserDetails userDetails = userDetailsService.loadUserByUsername(authenticationRequest.getUsername());

final String token = jwtTokenUtil.generateToken(userDetails);

return ResponseEntity.ok(new JwtResponse(token));

}

@RequestMapping(value = "/users/register", method = RequestMethod.POST)

public ResponseEntity<?> saveUser(@RequestBody UserDTO user) throws Exception {

return ResponseEntity.ok(userDetailsService.save(user));

}

private void authenticate(String username, String password) throws Exception {

try {

authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));

} catch (DisabledException e) {

throw new Exception("USER\_DISABLED", e);

} catch (BadCredentialsException e) {

responseEntity = (ResponseEntity) ResponseEntity.badRequest();

}

}

}

package course.work.controllers;

import com.lowagie.text.DocumentException;

import course.work.models.\*;

import course.work.repositories.ProjectRepository;

import course.work.repositories.RoleRepository;

import course.work.repositories.TaskRepository;

import course.work.repositories.UserRepository;

import course.work.requests.ProjectCreate;

import course.work.services.PdfExporter;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.text.DateFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.time.LocalDateTime;

import java.time.ZoneId;

import java.time.format.DateTimeFormatter;

import java.util.Date;

import java.util.List;

import java.util.Locale;

@RestController

@CrossOrigin

@RequestMapping("/projects")

public class ProjectController {

@Autowired

private ProjectRepository repository;

@Autowired

private UserRepository userRepository;

@Autowired

private TaskRepository taskRepository;

@Autowired

private RoleRepository roleRepository;

@GetMapping(value = "/")

List<Project> findAll() {

return repository.findAll();

}

@PostMapping(value = "/create")

Project create(@RequestBody ProjectCreate project) {

Project newProject = new Project();

newProject.setName(project.getName());

newProject.setDescription(project.getDescription());

newProject.setStatus(ProjectStatus.Open);

newProject.setStartAt(new Date());

// Add user to the project

User user = userRepository.findById(project.user\_id).orElse(null);

assert user != null;

Role role = new Role();

role.setRole(RoleType.ProjectManager);

user.getRoles().add(role);

// Role role = new Role();

// role.setRole(RoleType.ProjectManager);

// role.getUsers().add(user);

// user.getRoles().add(role);

newProject.getUsers().add(user);

return repository.save(newProject);

}

@DeleteMapping(value = "/{id}")

Integer destroy(@PathVariable int id) {

repository.deleteById(id);

return id;

}

@PutMapping(value = "/add\_task/{id}/{user\_id}")

Project addTask(@PathVariable int id, @PathVariable int user\_id, @RequestBody Task task) {

Project project = repository.findById(id).orElse(null);

Task newTask = new Task();

User user = userRepository.findById(user\_id).orElse(null);

newTask.setUser(user);

newTask.setDescription(task.getDescription());

newTask.setTitle(task.getTitle());

newTask.setEstimation(task.getEstimation());

newTask.setEstimation(task.getEstimation());

newTask.setPriority(task.getPriority());

newTask.setStatus(TaskStatus.NotStarted);

taskRepository.save(newTask);

assert project != null;

project.getTasks().add(newTask);

return repository.save(project);

}

@GetMapping("/export/pdf/{before}/{after}")

public void exportToPDF(@PathVariable String before, @PathVariable String after, HttpServletResponse response) throws DocumentException, IOException, ParseException {

response.setContentType("application/pdf");

DateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-dd\_HH:mm:ss");

String currentDateTime = dateFormatter.format(new Date());

String headerKey = "Content-Disposition";

String headerValue = "attachment; filename=users\_" + currentDateTime + ".pdf";

response.setHeader(headerKey, headerValue);

SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd", Locale.ENGLISH);

Date beforeDate = formatter.parse(before);

Date afterDate = formatter.parse(after);

afterDate.setHours(23);

List<Project> listProjects = repository.findAllByCreatedAtBetween(beforeDate, afterDate);

PdfExporter exporter = new PdfExporter(listProjects, before, after);

exporter.export(response);

}

public static Date convertDate(String date) throws ParseException {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd", Locale.ENGLISH);

return java.util.Date

.from(LocalDateTime.parse(date,formatter).atZone(ZoneId.systemDefault()).toInstant());

}

}

package course.work.controllers;

import course.work.models.Note;

import course.work.models.Task;

import course.work.repositories.TaskRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@CrossOrigin

@RequestMapping("/tasks")

public class TaskController {

@Autowired

private TaskRepository repository;

@GetMapping(value = "/")

List<Task> findAll() {

return repository.findAll();

}

@GetMapping(value = "/{id}")

Task findOne(@PathVariable int id) {

return repository.findById(id).orElse(null);

}

@PutMapping(value = "/{id}")

Task update(@PathVariable int id, @RequestBody Task task) {

Task oldTask = repository.findById(id).orElse(null);

assert oldTask != null;

oldTask.setTitle(task.getTitle());

oldTask.setDescription(task.getDescription());

oldTask.setEstimation(task.getEstimation());

oldTask.setWorkTime(task.getWorkTime());

oldTask.setPriority(task.getPriority());

Note note = new Note();

if (task.getNoteText() != null) {

note.setText(task.getNoteText());

oldTask.getNotes().add(note);

}

oldTask.setStatus(task.getStatus());

return repository.save(oldTask);

}

@DeleteMapping(value = "/{id}")

Integer destroy(@PathVariable int id) {

repository.deleteById(id);

return id;

}

}

package course.work.controllers;

import course.work.models.User;

import course.work.repositories.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@CrossOrigin

@RequestMapping("/users")

public class UserController {

@Autowired

private UserRepository repository;

@GetMapping(value = "/")

List<User> findAll() {

return repository.findAll();

}

@GetMapping(value = "/{id}")

User findOne(@PathVariable int id) {

return repository.findById(id).orElse(null);

}

@PostMapping(value = "/get\_by\_username")

User getByUsername(@RequestBody User user) {

return repository.findByUsername(user.getUsername());

}

@PutMapping(value = "/{id}")

User update(@PathVariable int id, @RequestBody User user) {

User oldUser = repository.findById(id).orElse(null);

assert oldUser != null;

oldUser.setName(user.getName());

oldUser.setUsername(user.getUsername());

oldUser.setPassword(user.getPassword());

oldUser.setEmail(user.getEmail());

return repository.save(oldUser);

}

@DeleteMapping(value = "/{id}")

Integer destroy(@PathVariable int id) {

repository.deleteById(id);

return id;

}

}

package course.work.models;

import com.fasterxml.jackson.annotation.JsonIgnore;

import javax.persistence.\*;

@Entity

@Table(name = "user")

public class DAOUser {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

@Column

private String username;

@Column

@JsonIgnore

private String password;

@Column

public String name;

@Column

public String email;

public int getId() {

return id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

package course.work.models;

import java.io.Serial;

import java.io.Serializable;

public class JwtRequest implements Serializable {

@Serial

private static final long serialVersionUID = 5926468583005150707L;

private String username;

private String password;

//need default constructor for JSON Parsing

public JwtRequest()

{

}

public JwtRequest(String username, String password) {

this.setUsername(username);

this.setPassword(password);

}

public String getUsername() {

return this.username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return this.password;

}

public void setPassword(String password) {

this.password = password;

}

}

package course.work.models;

import java.io.Serial;

import java.io.Serializable;

public class JwtResponse implements Serializable {

@Serial

private static final long serialVersionUID = -8091879091924046844L;

private final String jwttoken;

public JwtResponse(String jwttoken) {

this.jwttoken = jwttoken;

}

public String getToken() {

return this.jwttoken;

}

}

package course.work.models;

import lombok.Getter;

import lombok.Setter;

import org.hibernate.annotations.CreationTimestamp;

import org.hibernate.annotations.UpdateTimestamp;

import javax.persistence.\*;

import javax.validation.constraints.Min;

import java.util.Date;

@Entity

@Table(name = "note")

public class Note {

@Getter

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

public int id;

@Getter

@Setter

@Column(columnDefinition="TEXT")

private String text;

@Getter

@CreationTimestamp

private Date createdAt;

@Getter

@UpdateTimestamp

private Date updatedAt;

}

package course.work.models;

import lombok.Getter;

import lombok.Setter;

import org.hibernate.annotations.CreationTimestamp;

import org.hibernate.annotations.UpdateTimestamp;

import javax.persistence.\*;

import java.util.Date;

import java.util.HashSet;

import java.util.Set;

@Entity

@Table(name = "project")

public class Project {

@Getter

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

public int id;

@Getter

@Setter

@Column(unique=true, nullable = false)

private String name;

@Getter

@Setter

private String description;

@Getter

@Setter

@Column(columnDefinition = "ENUM('Open', 'Closed')")

@Enumerated(EnumType.STRING)

private ProjectStatus status;

@Getter

@Setter

@OneToMany(fetch = FetchType.EAGER)

private Set<Task> tasks = new HashSet<>();

@Getter

@Setter

@ManyToMany(fetch = FetchType.EAGER,

cascade = {

CascadeType.PERSIST,

CascadeType.MERGE

})

private Set<User> users = new HashSet<>();

@Getter

@Setter

@Column(nullable = false)

private Date startAt;

@Getter

@Setter

private Date endAt;

@Getter

@CreationTimestamp

private Date createdAt;

@Getter

@UpdateTimestamp

private Date updatedAt;

}

package course.work.models;

public enum ProjectStatus {

Open,

Closed

}

package course.work.models;

import lombok.Getter;

import lombok.Setter;

import javax.persistence.\*;

import java.util.HashSet;

import java.util.Set;

@Entity

@Table(name = "role")

public class Role {

@Getter

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int id;

@Getter

@Setter

@Column(columnDefinition = "ENUM('User', 'ProjectManager', 'Admin')")

@Enumerated(EnumType.STRING)

private RoleType role;

@Getter

@Setter

@ManyToMany(fetch = FetchType.EAGER,

cascade = {

CascadeType.PERSIST,

CascadeType.MERGE

})

private Set<User> users = new HashSet<>();

}

package course.work.models;

public enum RoleType {

User,

ProjectManager,

Admin

}

package course.work.models;

import lombok.Builder;

import lombok.Getter;

import org.hibernate.annotations.CreationTimestamp;

import org.hibernate.annotations.UpdateTimestamp;

import lombok.Setter;

import org.w3c.dom.Text;

import javax.persistence.\*;

import javax.validation.constraints.Min;

import java.util.Date;

import java.util.HashSet;

import java.util.Set;

@Entity

@Table(name = "task")

public class Task {

@Getter

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

public int id;

@Getter

@Setter

@Min(0)

@Column(columnDefinition = "float default 0.0")

private Float estimation;

@Getter

@Setter

private String title;

@Getter

@Setter

@Column(columnDefinition = "ENUM('NotStarted', 'InProgress', 'Submitted', 'Closed')")

@Enumerated(EnumType.STRING)

private TaskStatus status;

@Getter

@Setter

@Column(columnDefinition="TEXT")

private String description;

@Getter

@Setter

private String priority;

@Getter

@Setter

private float workTime;

@Getter

@Setter

@ManyToOne(fetch = FetchType.EAGER, optional = true)

private User user;

@Getter

@Setter

@OneToMany(fetch = FetchType.EAGER, cascade = {CascadeType.ALL})

private Set<Note> notes = new HashSet<>();

@Getter

@Transient

private String noteText;

@Getter

@Setter

@Column(nullable = true)

private Date startAt;

@Getter

@Setter

private Date dueAt;

@Getter

@CreationTimestamp

private Date createdAt;

@Getter

@UpdateTimestamp

private Date updatedAt;

}

package course.work.models;

public enum TaskStatus {

NotStarted,

InProgress,

Submitted,

Closed

} package course.work.models;

import course.work.models.Project;

import course.work.models.Role;

import course.work.models.Task;

import lombok.Data;

import lombok.Getter;

import lombok.Setter;

import javax.persistence.\*;

import java.util.HashSet;

import java.util.Set;

@Entity

@Data

@Table(name = "user")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

public int id;

public String name;

public String username;

public String email;

public String password;

@Getter

@Setter

@ManyToMany(fetch = FetchType.LAZY,

cascade = {

CascadeType.PERSIST,

CascadeType.MERGE

})

private Set<Role> roles = new HashSet<>();

}

package course.work.models;

public class UserDTO {

private String username;

private String password;

private String email;

private String name;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

package course.work.repositories;

import course.work.models.Project;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import java.util.Date;

import java.util.List;

@Repository

public interface ProjectRepository extends JpaRepository<Project, Integer> {

List<Project> findAllByCreatedAtBetween(

Date createdAtBefore,

Date createdAtAfter

);

}

package course.work.repositories;

import course.work.models.Role;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface RoleRepository extends JpaRepository<Role, Integer> {

}

package course.work.repositories;

import course.work.models.Project;

import course.work.models.Task;

import course.work.models.User;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

package course.work.repositories;

import org.springframework.data.repository.CrudRepository;

import course.work.models.DAOUser;

public interface UserDao extends CrudRepository<DAOUser, Integer> {

DAOUser findByUsername(String username);

}

import java.util.Set;

@Repository

public interface TaskRepository extends JpaRepository<Task, Integer> {

}

package course.work.repositories;

import course.work.models.User;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

@Repository

public interface UserRepository extends JpaRepository<User, Integer> {

@Query("SELECT u FROM User u WHERE u.username=:username")

public User findByUsername(@Param("username") String username);

}

package course.work.requests;

import course.work.models.Project;

import lombok.Getter;

import lombok.Setter;

public class ProjectCreate extends Project {

@Getter

@Setter

public int user\_id;

}

package course.work.services;

import java.util.ArrayList;

import course.work.models.\*;

import course.work.repositories.UserDao;

import course.work.repositories.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;

@Service

public class JwtUserDetailsService implements UserDetailsService {

@Autowired

private UserDao userDao;

@Autowired

private PasswordEncoder bcryptEncoder;

@Autowired

private UserRepository userRepository;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

DAOUser user = userDao.findByUsername(username);

if (user == null) {

throw new UsernameNotFoundException("User not found with username: " + username);

}

return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(),

new ArrayList<>());

}

public User save(UserDTO user){

User newUser = new User();

newUser.setUsername(user.getUsername());

newUser.setName(user.getName());

newUser.setEmail(user.getEmail());

newUser.setPassword(bcryptEncoder.encode(user.getPassword()));

// Add role to the user

Role role = new Role();

role.setRole(RoleType.User);

newUser.getRoles().add(role);

User userr = userRepository.save(newUser);

return userr;

}

}

package course.work.services;

import course.work.models.Project;

import java.awt.Color;

import java.io.File;

import java.io.IOException;

import java.util.List;

import java.util.concurrent.atomic.AtomicReference;

import javax.servlet.http.HttpServletResponse;

import com.lowagie.text.\*;

import com.lowagie.text.pdf.\*;

import course.work.models.TaskStatus;

public class PdfExporter {

private List<Project> projectList;

private String before;

private String after;

public PdfExporter(List<Project> projectList, String before, String after) {

this.projectList = projectList;

this.before = before;

this.after = after;

}

private void writeTableHeader(PdfPTable table) {

PdfPCell cell = new PdfPCell();

cell.setBackgroundColor(Color.BLUE);

cell.setPadding(5);

Font font = FontFactory.getFont(FontFactory.HELVETICA);

font.setColor(Color.WHITE);

cell.setPhrase(new Phrase("Project Name", font));

table.addCell(cell);

cell.setPhrase(new Phrase("Description", font));

table.addCell(cell);

cell.setPhrase(new Phrase("Tasks count", font));

table.addCell(cell);

cell.setPhrase(new Phrase("Not Started", font));

table.addCell(cell);

cell.setPhrase(new Phrase("In Progress", font));

table.addCell(cell);

cell.setPhrase(new Phrase("Submitted", font));

table.addCell(cell);

cell.setPhrase(new Phrase("Closed", font));

table.addCell(cell);

cell.setPhrase(new Phrase("Hours spent", font));

table.addCell(cell);

}

private void writeTableData(PdfPTable table) throws IOException {

for (Project project : projectList) {

String path = "src/main";

File file = new File(path);

String absolutePath = file.getAbsolutePath();

BaseFont base = BaseFont.createFont(absolutePath + "/arial.ttf", BaseFont.IDENTITY\_H, BaseFont.EMBEDDED);

Font font = new Font(base, 14);

Phrase name = new Phrase(project.getName(), font);

Phrase desc = new Phrase(project.getDescription(), font);

table.addCell(name);

table.addCell(desc);

table.addCell(String.valueOf(project.getTasks().size()));

table.addCell(String.valueOf(project.getTasks().stream().filter(

task -> task.getStatus() == TaskStatus.NotStarted

).count()));

table.addCell(String.valueOf(project.getTasks().stream().filter(

task -> task.getStatus() == TaskStatus.InProgress

).count()));

table.addCell(String.valueOf(project.getTasks().stream().filter(

task -> task.getStatus() == TaskStatus.Submitted

).count()));

table.addCell(String.valueOf(project.getTasks().stream().filter(

task -> task.getStatus() == TaskStatus.Closed

).count()));

final float[] worktime = {0};

project.getTasks().stream().forEach((a) ->

{

worktime[0] += a.getWorkTime();

}

);

table.addCell(String.valueOf(worktime[0]));

}

}

public void export(HttpServletResponse response) throws DocumentException, IOException {

Document document = new Document(PageSize.A4);

PdfWriter.getInstance(document, response.getOutputStream());

document.open();

String path = "src/main";

File file = new File(path);

String absolutePath = file.getAbsolutePath();

BaseFont base = BaseFont.createFont(absolutePath + "/arial.ttf", BaseFont.IDENTITY\_H, BaseFont.EMBEDDED);

Font font = new Font(base, 14);

font.setSize(18);

font.setColor(Color.BLUE);

Paragraph p = new Paragraph("Projects report [" + this.before + " - " + this.after + "]", font);

p.setAlignment(Paragraph.ALIGN\_CENTER);

document.add(p);

PdfPTable table = new PdfPTable(8);

table.setWidthPercentage(100f);

table.setWidths(new float[]{3f, 3.5f, 2f, 2f, 2f, 2f, 2f, 2f});

table.setSpacingBefore(10);

writeTableHeader(table);

writeTableData(table);

document.add(table);

document.close();

}

}

package course.work;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication

@ComponentScan(basePackages = {"course.work"})

public class WorkApplication {

public static void main(String[] args) {

SpringApplication.run(WorkApplication.class, args);

}

}

Приложение Б

**(обязательное)**

Приложение В

**(обязательное)**

Приложение Г

**(обязательное)**