**СОДЕРЖАНИЕ**

Введение5

1 Описание предметной области и определение требований к системе

с точки зрения предметной области6

1.1 Описание предметной области6

1.2 Описание основных процессов предметной области8

1.2 Определение требований к системе13

2 Постановка задачи и обзор методов ее решения14

3 Спецификация системы19

4 Модели представления системы и их описание22

5 Информационная модель системы и ее описание26

6 Проектирование программного средства автоматизации договорной деятельности торгового предприятия29

6.1 Архитектурные решения29

6.2 Проектирование пользовательского интерфейса33

7 Описание и разработка алгоритмов, реализующих бизнес-логику программного средства35

8 Руководство пользователя39

9 Результаты тестирования разработанной системы и оценка выполнения задач42

Заключение44

Список использованных источников45

Приложение А (обязательное) Листинг скрипта генерации базы данных46

Приложение Б (обязательное) Листинг программного кода50

**ВВЕДЕНИЕ**

Тема данной курсовой работы: «Проектирование и разработка программного средства автоматизации договорной деятельности торгового предприятия».

Во всех сферах хозяйствования на рыночных началах организация предпринимательской деятельности базируется на исключительных договорных взаимоотношениях.

В современных условиях при организации хозяйственных связей торговли с поставщиками товаров важное значение имеет четко выстроенная и отлаженная договорная работа. В ее основе лежит договор, в который, как правило, включены обязательства, принятые субъектами хозяйствования.

В договор могут включаться обязательства, урегулированные и неурегулированные законодательными и нормативными правовыми актами. Это позволяет партнерам по хозяйственным связям расширять возможности ведения свободной купли-продажи товаров и отражать в договоре свои интересы.

Договорные взаимоотношения это одно из неотъемлемых прав и основных составляющих во взаимоотношениях, хозяйственных связях в предпринимательской деятельности.

Вопросу организации договорной работы в организации посвящен ряд исследований отечественных и зарубежных авторов, однако, несмотря на это в отечественных торговых организациях для решения подобных задач при принятии управленческих решений полагаются, в основном, на опыт, не обращая внимания на теоретические наработки и современные технологии.

Особую актуальность в рамках данного вопроса приобретает установление договорных отношений с поставщиками, а также регулирование данной деятельности с ориентацией на усовершенствование, что представляет научный и практический интерес. В настоящее время данная тема весьма актуальна для любой организации вне зависимости от того, в какой сфере, с какими товарами и на каких рынках работает данная организация.

Исходя из этого, цель данной курсовой работы – разработать веб-приложение по автоматизации договорной деятельности торгового предприятия.

Для достижения цели данной курсовой работы, необходимо выполнение следующих задач:

– исследовать предметную область курсового проекта и определить требования к системе с точки зрения предметной области;

– выполнить функциональное и информационное моделирование разрабатываемой системы;

– написать веб-приложение, которое будет хранить информацию о договорах торгового предприятия;

– создать максимально легкий и понятный пользовательский интерфейс;

– организовать надежный обмен данных между клиентом и сервером;

– обеспечить хранение данных средствами производительной и надежной СУБД.

Проведем анализ поставленной цели: требуется разработать веб-приложение, которое будет автоматизировать договорную деятельность торгового предприятия. Данное приложение можно будет внедрять в различные торговые предприятия.

Применение базы данных в качестве хранилища информации позволит оптимально и эффективно хранить информацию, ее структурировать. Реализованный многопоточный сервер позволит производить распараллеливание работы сразу нескольких клиентов в разных потоках.

Теперь можно перейти к описанию предметной области и опеределению требований к к системе.

**1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ К СИСТЕМЕ**

* 1. **Описание предметной области**

В данном подразделе пояснительной записки будет описана предметная область текущего курсового проекта.

На первом этапе проектирования информационной системы необходимо выполнить анализ предметной области, то есть определить объекты предметной области и связи между объектами.

При выборе состава и структуры предметной области возможны два подхода: функциональный и предметный.

Функциональный подход реализует принцип движения «от задач» и применяется, когда определен комплекс задач, для обслуживания которых создается информационная система. В этом случае можно выделить минимальный необходимый набор объектов предметной области, которые должны быть описаны.

В предметном подходе объекты предметной области определяются с таким расчетом, чтобы их можно было использовать при решении множества разнообразных, заранее не определенных задач. Чаще всего используется комбинация этих двух подходов.

Предметная область данного курсового проекта: «Договорная деятельность торгового предприятия», то есть основным объектом предметной области является договорная деятельность. Данная область и будет подробно и детально рассмотрена далее в рамках данного подраздела.

В процессе продвижения товаров на рынок торговля выполняет связующие функции между производством и потреблением. Возникающие при этом отношения, базирующиеся на договорной основе, называются хозяйственными связями. Хозяйственные связи, их организация, регулирование и непосредственное осуществление выражают сущность договорной работы.

Осуществляющийся переход экономики страны от административно-командной экономики к рыночной существенно расширил возможности торговых организаций по установлению взаимоотношений с поставщиками товаров. В настоящее время в хозяйственные связи по поставкам товаров могут вступать оптовые и розничные торговые организации на условиях равноправия и свободного выбора партнеров.

При рыночных отношениях соответствующим образом соединено воедино приобретение нужного товара, требуемого качества, в установленные сроки поставки, по приемлемой цене у надежного поставщика. Определённое воздействие оказывает на организацию договорной работы переход от рынка продавца к рынку покупателя, сопровождающийся существенными изменениями в стратегии производства и реализации продукции.

Согласно п. 1 ст. 390 Гражданского кодекса Республики Беларусь договор представляет собой соглашение двух или нескольких лиц об установлении, изменении или прекращении гражданских прав и обязанностей.

Посредством договора субъекты хозяйствования юридически закрепляют право осуществления хозяйственных связей.

Понятие «хозяйственные связи» включает организационные, коммерческие, правовые, административные и другие отношения, складывающиеся между покупателями и продавцами в процессе поставок товаров.

Хозяйственные связи организаций торговли с поставщиками товаров представляют собой совокупность взаимоотношений по поставам товаров народного потребления.

Предметом хозяйственных связей является поставка (купля-продажа) товаров и услуг.

Субъектами же могут выступать юридические и физические лица любых организационно-правовых форм, принявших на себя обязательства по обеспечению тех или иных условий осуществления хозяйственных связей. Они могут выступать в качестве продавцов, покупателей и посредников.

В условиях рыночной экономики централизованное регулирование хозяйственных связей уступает место самостоятельному установлению договорных взаимоотношений между партнерами на базе правовых норм гражданского законодательства.

В договор могут включаться обязательства, урегулированные и неурегулированные законодательными и нормативными актами. Это позволяет партнерам по хозяйственным связям расширять возможности ведения свободной купли-продажи товаров и отражать в договоре свои интересы.

В то же время договорная деятельность существенно усложнилась и требует квалифицированного подхода коммерческих работников для ее эффективного осуществления.

Договорная деятельность включает в себя следующие этапы:

1 Преддоговорный этап, который включает в себя подготовку организаций торговли к заключению договоров с поставщиками товаров. На том этапе коммерческие и юридические специалисты организаций ведут предварительные переговоры с потенциальными поставщиками о предстоящей сделке, готовят проекты договоров и необходимые документы. Торговые переговоры требуют от принимающих в них участие специалистов глубоких знаний в области хозяйственного права, менеджмента, психологии, товароведения, экономики, ценообразования, владения необходимой коммерческой информацией и практическими навыками в переговорном процессе.

2 Заключение, изменение и расторжение договора. Наиболее часто в практике используются договоры поставки.

3 Учет и контроль исполнения договорных обязательств. После заключения договора поставки он приобретает юридическую силу и стороны обязаны соблюдать принятые на себя обязательства. За их нарушение сторона, допустившая данное нарушение несет имущественную ответственность в размерах, указанных в договоре либо в соответствии с положением о поставках товаров в Республике Беларусь.

В настоящее время основным нормативно-правовым документом для регулирования коммерческих взаимоотношений по поставкам товаров является Гражданский кодекс Республики Беларусь, в котором определены порядок заключения, изменения, исполнения и прекращения действия договоров поставки. В нем также определено право Совета Министров Республики Беларусь утверждать Положение о поставках товаров.

Закон Республики Беларусь «О поставках товаров для государственных нужд» наряду с Гражданским кодексом является одним из основных документов, регулирующих взаимоотношения предприятий и организаций по поставкам товаров.

Помимо данных документов правовое регулирование договорной деятельности и хозяйственных связей по поставкам товаров осуществляются следующими подзаконными нормативно-правовыми актами:

– положение о поставках товаров в Республике Беларусь (1996 г.);

– указ Президента Республики Беларусь «О некоторых мерах по упорядочению посреднической деятельности при продаже товаров» (от 07.03.2000 № 117);

– указ Президента Республики Беларусь «О некоторых вопросах заключения договоров и исполнения обязательств на территории Республики Беларусь» (от 15.08.2005 № 373 с изменениями и дополнениями) и другие.

Порядок заключения и исполнения внешнеторговых договоров определяется Венской конвенцией о международных договорах купли-продажи.

На этом описание предметной области завершено. Была успешно описана предметная область «Договорная деятельность торгового предприятия», с рассмотрением всех тонкостей и нюансов рынка.

**1.2** **Описание основных процессов предметной области**

В данном подразделе будут рассмотрены бизнес-процессы текущей прикладной области. При этом будет произведено описание процессов, как в текстовом формате, так и при помощи нотации IDEF0 в графическом виде.

Исходя из задания данного курсового проекта, целью моделирования данной системы является описание ее функциональности для дальнейшего использования созданной модели при разработке информационной модели.

IDEF0 используется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальные объекты, преобразуемые этими функциями.

Модель состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга. Диаграммы – главные компоненты модели, все функции и интерфейсы на них представлены как блоки и дуги. Место соединения дуги с блоком определяет тип интерфейса:

Тип интерфейса:

– управляющая информация входит в блок сверху;

– входная информация входит в блок слева;

– результаты выходят из блока справа;

– механизм (человек или автоматизированная система), который осуществляет операцию, входит в блок снизу.

Каждый компонент модели может быть декомпозирован (расшифрован более подробно) на другой диаграмме. Рекомендуется прекращать моделирование, когда уровень детализации модели удовлетворяет ее цель. Общее число уровней в модели не должно превышать 5-6.

Построение диаграмм начинается с представления всей системы в виде одного блока и дуг, изображающих интерфейсы с функциями вне системы. Затем блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами. Каждая детальная диаграмма является декомпозицией блока из диаграммы предыдущего уровня.

В стандарте IDEF0 предполагается наличие двух типов диаграмм: контекстной диаграммы и диаграммы декомпозиции.

Каждая модель должна иметь контекстную диаграмму верхнего уровня, на которой объект моделирования представлен единственным блоком с граничными стрелками. Эта диаграмма называется А-0 (А минус нуль). Стрелки на этой диаграмме отображают связи объекта моделирования с окружающей средой. Данная диаграмма представлена на рисунке 1.1.

На контекстной диаграмме, основной задачей которой является автоматизация договорной деятельности торгового предприятия, входными данными являются заявки на заключение договора. Результатом работы будет доставка товаров.

После описания системы в целом и ее взаимодействие с окружающим миром (контекстная диаграмма), проводится функциональная декомпозиция.

Диаграммы декомпозиции содержат родственные работы (функции, задачи), то есть дочерние работы, имеющие общую родительскую работу. Работы на диаграммах декомпозиции обычно располагаются по диагонали от левого верхнего угла к правому нижнему.



Рисунок 1.1 – Контекстная диаграмма автоматизации договорной деятельности торгового предприятия

Контекстная диаграмма первого уровня разбивается на пять процессов. Процессы, на которые разбита контекстная диаграмма:

– обработать заявку;

– оформить договор;

– подготовить товары;

– оформить документы;

– осуществить доставку.

Весь путь прохождения входных параметров проходит сверху вниз справа налево через эти пять блоков по порядку.

В рамках данного уровня поступает заявка на заключение договора, а на выходе мы имеем доставку товаров.

Управление представлено образцом заполнения, нормативными документами и законами, расчетом параметров перевозки товаров, ГОСТами оформления документов, правилами перевозки товаров, а механизмами выступают диспетчер, сотрудник по оформлению документов, начальник отдела, транспортное средство и водитель.

Второй уровень диаграммы представлен на рисунке 1.2.



Рисунок 1.2 – Диаграмма декомпозиции автоматизации договорной деятельности торгового предприятия

Третий уровень диаграммы – это разбиение процесса «Обработать заявку» на три процесса.

Данные процессы представлены далее:

– получить заявку;

– обработать заявку;

– зарегистрировать заявку.

В рамках данного процесса поступает заявка на заключение договора, на выходе имеем данные заявки.

Механизмом выступает диспетчер.

Управление представлено только образцом заполнения.

Третий уровень диаграммы представлен на рисунке 1.3.

Далее будет рассмотрен процесс оформления документов. Данное окно представлено на рисунке 1.4. Для этого необходимо декомпозировать соответствующий функциональный блок «Оформить документы».

Для достижения цели верхнего уровня процесс разбит на два функциональных блока:

– оформить и выдать путевой лист;

– оформить и выдать накладную.

Данные преобразований: оформленный путевой лист. На выходе система получает оформленные накладные.



Рисунок 1.3 – Диаграмма декомпозиции обработки заявки



Рисунок 1.4 – Диаграмма декомпозиции оформления документов

Управляющий механизм: сотрудник по оформлению документов. Механизм ограничения: ГОСТы оформления документов.

Далее будет подробно рассмотрено осуществление доставки. Данное окно представлено на рисунке 1.5. На данном этапе происходят следующие процессы:

– погрузить товары в транспортное средство;

– перевезти товары в назначенное место и время;

– выгрузить товары из транспортного средства.



Рисунок 1.5 – Диаграмма декомпозиции осуществления доставки

Входная информация: оформленные накладные. На выходе получаем «доставка груза».

Управляющие механизмы: водитель и транспортное средство. Механизм ограничения: правила перевозки товаров.

Таким образом, были рассмотрены и описаны основные процессы предметной области в нотации IDEF0. Из них видно, как оптимизируется и автоматизируется договорная деятельность торгового предприятия. При этом, и процессы самой предметной области были детально рассмотрены. На этом описание подраздела завершено.

* 1. **Определение требований к системе**

Перед разработкой программного средства предполагается составление требований к данному средству, дабы описать его средства и технологии разработки, функциональные требования.

Основным требованием к программному средству является разработка системы автоматизации договорной деятельности торгового предприятия с использованием объектно-ориентированных языков PHP и JavaScript. Используемые технологии должны быть HTML5, CSS3, Bootstrap 4. Архитектура приложения должна быть выполнена в архитектуре паттерна MVC. Интерфейс приложения должен быть оформлен с использованием каскадных таблиц стилей (CSS).

Помимо этого, структура программного средства должна иметь классическую трехслойную архитектуру, включающую презентационную логику, бизнес-логику, слой доступа к данным.

Взаимодействие между серверной и клиентской частями веб-приложения должно осуществляться с использованием протокола HTTPS.

Что касается требований к поставке программного средства, то выделяют следующие: исполняемые файлы должны работать в среде операционной системы Windows 7; используемой системой управления базой данных должна быть MySQL; язык интерфейса и элементов управления должен быть русский; программное средство должно запускаться без интегрированных средств разработки.

Перейдем к описанию инструментария, который понадобится для выполнения требований. Стоит отметить, что выбор инструментария для программирования огромный, как на стороне сервера, так и на стороне клиента, как бесплатных, так и платных.

Данные программные средства были выбраны для разработки программного продукта:

– интегрированная среда разработки Visual Studio Code для разработки на языках программирования PHP и JavaScript, языке разметки HTML и каскадных таблиц стилей CSS;

– инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию баз данных MySQL Workbench;

– система контроля версий Git для хранения нескольких версий программного продукта;

– браузер Google Chrome для проверки на кроссбраузерность и адаптивность программного продукта, а также проверки результатов работы программного продукта при разработке.

Технологии и инструменты, описанные выше, во-первых, идеально подходят для веб-разработки, во-вторых, являются современными и удобными в эксплуатации. Требования к программному средству определены, теперь можно переходить к постановке задачи курсового проекта.

**2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ**

Требования к программному средству определены, теперь можно переходить к постановке задачи данного курсового проекта.

Основной задачей данного курсового проекта является создание программного средства автоматизации договорной деятельности торгового предприятия.

Что касается подзадач курсового проекта, то выделяют следующие:

– создать веб-приложение, которое будет хранить информацию о договорах торгового предприятия;

– создать максимально легкий, понятный и удобный пользовательский интерфейс;

– реализовать авторизацию пользователей для предоставления им разных прав;

– организовать надежный обмен данных между клиентом и сервером;

– обеспечить хранение данных средствами производительной и надежной СУБД.

Для реализации поставленных задач нужно определиться с какими инструментами будет производиться их выполнение. Наиболее эффективную работу с централизованной БД обеспечивает архитектура клиент/сервер. В отличие от системы удаленной обработки, в которой имеется только один компьютер, клиент/серверная система состоит из множества компьютеров, объединенных в сеть. Компьютеры, называемые клиентами, занимаются обработкой прикладных программ. Компьютеры, называемые серверами, занимаются обработкой БД.

Тип компьютеров, используемых в качестве клиентов может быть разным, это могут быть большие ЭВМ или микрокомпьютеры. Однако, как правило, функции клиентов выполняют почти всегда ПК. В роли сервера может выступать компьютер любого типа, но по экономическим причинам функции сервера чаще всего также выполняют ПК, но имеющие более высокую производительность.

На сервере сети размещается БД и устанавливается мощная серверная СУБД – сервер баз данных. Сервер БД – это программный компонент, обеспечивающий хранение больших объемов информации, ее обработку и представление ее пользователям в сетевом режиме.

На компьютере-клиенте приложение-клиент формирует запрос к БД. Серверная СУБД обеспечивает интерпретацию запроса, его выполнение, формирование результата запроса и пересылку его по сети на клиентский компьютер. Клиентское приложение интерпретирует его необходимым образом и представляет пользователю. Клиентское приложение может также посылать запрос на обновление БД и серверная СУБД вносит необходимые изменения в БД. Схема архитектуры клиент/сервер приведена на рисунке 2.1.

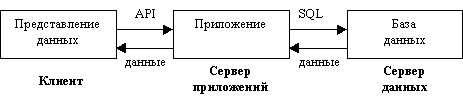


Рисунок 2.1 – Архитектура клиент/сервер

В архитектуре клиент/сервер функции клиентского приложения и серверной СУБД разделены.

Функции клиентского приложения разбиваются на следующие группы:

1 Ввод-вывод данных (презентационная логика) – это часть кода клиентского приложения, которая определяет, что пользователь видит на экране, когда работает с приложением.

2 Бизнес-логика – это часть кода клиентского приложения, которая определяет алгоритм решения конкретных задач приложения.

3 Обработка данных внутри приложения (логика базы данных) – это часть кода клиентского приложения, которая связывает данные сервера с приложением. Для этой связи используется процедурный язык запросов SQL, с помощью которого осуществляется выборка и модификация данных в серверных СУБД.

Сервер баз данных в общем случае осуществляет целый комплекс действий по управлению данными. Основными среди них являются следующие:

– выполнение пользовательских запросов на выбор и модификацию данных и метаданных, получаемых от клиентских приложений, функционирующих на ПК локальной сети;

– хранение и резервное копирование данных;

– поддержка ссылочной целостности данных согласно определенным в БД правилам;

– обеспечение авторизованного доступа к данным на основе проверки прав и привилегий пользователя.

Все серверные СУБД имеют клиентскую часть, которая обращается к БД посредством СУБД. Между клиентским приложением и СУБД не существует прямой связи и дополнительно встраиваются программные модули, позволяющие клиентскому приложению получать доступ к БД, создаваемым с помощью разных СУБД. Такие модули называются механизмами доступа к данным.

Следующей стадией разработки программного средства будет проектирование. На этом этапе для упрощения визуализации процесса проектирования используются так называемые нотации – схематическое выражение характеристик разрабатываемого программного средства.

После того, как программное средство будет спроектировано, произойдет переход к следующей стадии жизненного цикла – непосредственно разработке. На этом этапе будет разрабатываться программное средство в соответствии с ранее определенными требованиями.

После того, как этап разработки программного средства будет завершен, необходимо будет протестировать полученное программное средство. На этом этапе необходимо будет найти дефекты в программном средстве и сравнить описанное в требованиях поведение системы с реальным.

При обнаружении дефекта, необходимо будет его исправить, после чего повторить тестирование и на этот раз убедиться, что проблема исправлена и само исправление не стало причиной появления новых дефектов в системе.

После того, как тестирование программного средства пройдет успешно, необходимо будет пройти этап документации. Он будет включать в себя составление руководства пользователя, а также презентацию программного средства.

Выполнение всех описанных выше стадий жизненного цикла программного средства является обязательным при проектировании, разработке, тестировании и запуске программного средства.

На этом описание данного раздела завершено.

**3 СПЕЦИФИКАЦИЯ СИСТЕМЫ**

Текущий третий раздел описывает спецификацию вариантов использования.

Описание будет произведено двумя способами:

– текстовый вариант, описывающий в текстовом виде возможности системы;

– графический вариант при помощи диаграммы UML 2.0 – диаграммы вариантов использования.

Каждый вариант использования – это некоторое действие, которое доступно некоторому человеку, или актёру. То есть, это некоторый логически завершённый вариант работы системы.

Варианты использования описывают, с точки зрения действующего лица, группу действий в системе, которые приводят к конкретному результату.

Варианты использования являются описаниями типичных взаимодействий между пользователями системы и самой системой. Они отображают внешний интерфейс системы и указывают форму того, что система должна сделать (именно что, а не как).

При работе с вариантами использования важно помнить несколько простых правил:

– каждый вариант использования относится как минимум к одному действующему лицу;

– каждый вариант использования имеет инициатора;

– каждый вариант использования приводит к соответствующему результату (результату с «бизнес-значением»).

Варианты использования также могут взаимодействовать с другими вариантами использования.

Диаграмма вариантов использования состоит из актеров, для которых система производит действие и собственно действия Use Case, которое описывает то, что актер хочет получить от системы. Актер обозначается значком человечка, а Use Case – овалом. Дополнительно в диаграммы могут быть добавлены комментарии. Между актерами и вариантами использования могут быть различные виды взаимодействия. Каждый вариант использования показывает, как конкретный актер использует систему и в дальнейшем расширяется диаграммами состояний и последовательности действий.

Прецедент – это возможность моделируемой системы, благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат.

Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

На рисунке 3.1 изображена диаграмма вариантов использования курсового проекта. Роль актеров принадлежит администратору и пользователю.

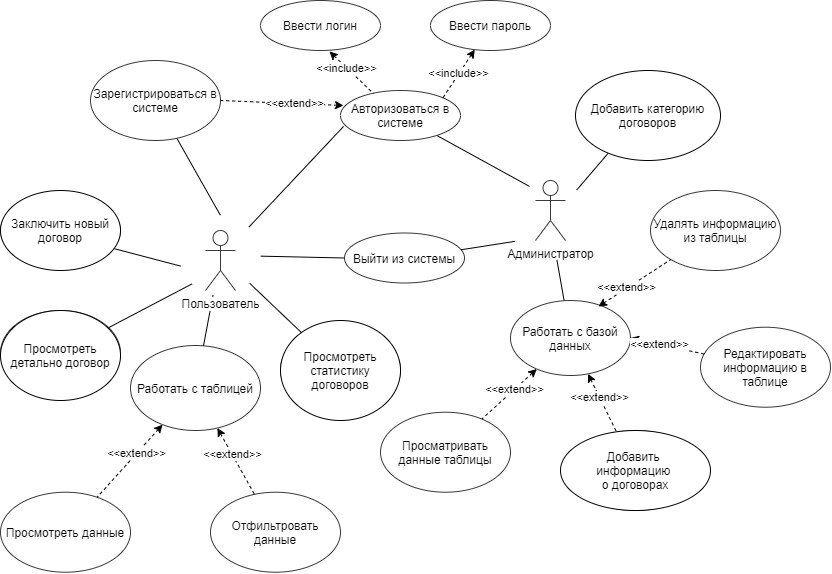


Рисунок 3.1 – Диаграмма вариантов использования

Данная диаграмма демонстрирует все варианты использования системы всеми основными актёрами.

При этом применяются три основных типа отношений:

– отношение ассоциации, которое показывает, что элементы имеют некоторую взаимосвязь;

– отношения расширения, которое применяется для моделирования выбираемого поведения системы;

– отношение включения, которое показывает зависимость между базовым вариантом использования и его специальным случаем.

Система предоставляет администратору следующие действия:

1 Авторизоваться в системе: ввести логин и ввести пароль.

2 Выйти из системы.

3 Добавить категорию договоров.

4 Работать с базой данных. Администратор может удалять информацию из таблицы, редактировать информацию в таблице, добавлять информацию о договорах, просматривать данные таблицы.

Система предоставляет пользователю следующие действия:

1 Авторизоваться в системе: ввести логин и ввести пароль.

2 Выйти из системы.

3 Просмотреть статистику договоров.

4 Работать с таблицей. Пользователь может просмотреть данные и отфильтровать данные.

5 Просмотреть детально договор.

6 Заключить новый договор.

7 Зарегистрироваться в системе.

Таким образом, описание всех вариантов использования системы и актёров завершено. Были описаны все варианты использования системы для каждой из ролей. На этом описание раздела завершено.

**4 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ И ИХ ОПИСАНИЕ**

В данном разделе будет продемонстрировано моделирование информационной системы с помощью стандарта UML, который использует графические обозначения для создания абстрактной модели системы и предназначен для определения, визуализации, проектирования и документирования в основном программных систем. UML позволяет описать систему практически со всех возможных точек зрения и разные аспекты поведения системы.

Для данного курсового проекта уже была построена такая диаграмма, как диаграмма вариантов использования. В данном разделе будут построены такие диаграммы как диаграмма состояний и диаграмма последовательности.

Диаграмма состояний предназначена для отображения состояний объектов системы, имеющих сложную модель поведения. Она показывает пространство состояний системы или ее элементов, события, которые влекут переход из одного состояния в другое, действия, которые происходят при изменении состояния. Объекты меняют своё состояние в ответ на происходящие события и стечением времени. Диаграмма состояний представляет состояния объекта и переходы между ними, а также начальное и конечное состояние объекта.

Диаграмма состояний данной системы приведена на рисунке 4.1.

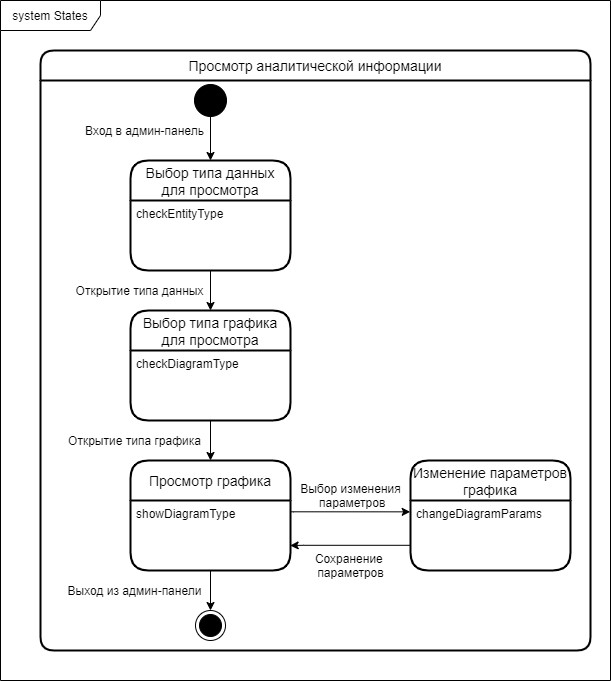


Рисунок 4.1 – Диаграмма состояний

Диаграмма состояний описывает один конкретный набор требований к состояниям системе. В данном случае описывает требования к состояниям в рамках просмотра графической аналитической информации о сущностях системы.

Алгоритм изменения состояний системы, проходящих в рамках основного состояния «Просмотр аналитической информации»:

1 «Выбор типа данных для просмотра». Начинается с сразу после начального состояния через переход «Вход в админ-панель». Означает, что в данный момент пользователь открыл панель управления сайта и выбирает, для какой сущности (договора, заказы и так далее) смотреть аналитическую информацию.

2 «Выбор типа графика для просмотра». Начинается после предыдущего состояния через переход «Открытие типа данных». Означает, что пользователь выбрал и открыл нужную сущность для аналитики и выбирает какой тип графика смотреть.

3 «Просмотр информации». Начинается после предыдущего события с перехода «Открытие типа графика». Выполняется выбор и открытие нужного графика пользователем системы, пользователь просматривает выбранный график. После данного состояния, если пользователь выбирает закрытие системы, то через переход «Выход из админ-панели» происходит переход к конечному состоянию.

4 «Изменение параметров графика». Если пользователь в предыдущем состоянии выбрал изменение параметров графика (переход «Выбор изменения параметров»), происходит переход к данному состоянию. В рамках состояния пользователь выбирает различные параметры графика. Переходит обратно к предыдущему состоянию через переход «Сохранение параметров».

Через диаграмму состояния отчётливо видны требования к системе с точки зрения пользователя системы для просмотра графической информации. Как итог, реализация диаграмма состояний в системе описывает требования.

Диаграмма последовательности – это диаграмма, где для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта.

В рамках курсового проекта диаграмма последовательности отражает алгоритм просмотра графика с аналитической информацией администратором в системе.

Диаграмма последовательности представлена на рисунке 4.2.

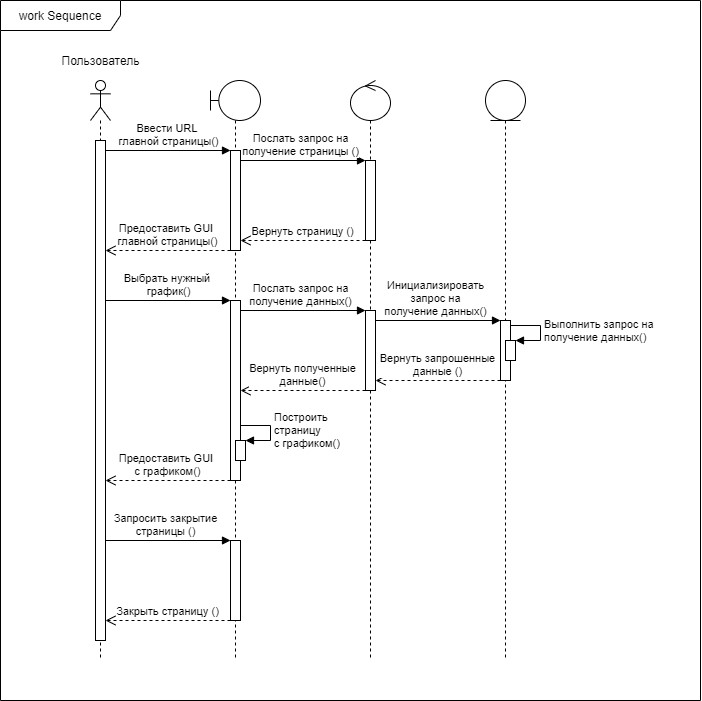


Рисунок 4.2 – Диаграмма последовательности

Алгоритм просмотра аналитической информации в системе имеет следующий вид:

1. пользователь вводит адрес главной страницы;
2. клиентское приложение по адресу посылает запрос на сервер;
3. сервер возвращает страницу клиентскому приложению, адрес которой был введён;
4. клиентское приложение строит страницу и отображает её;
5. пользователь выбирает нужный график;
6. клиентское приложение посылает запрос о получении данных для графика;
7. сервер запрашивает данные из базы данных из сервера MySQL;
8. сервер MySQL выбирает данные;
9. данные возвращаются серверу приложения;
10. сервер приложения возвращает данные клиентскому приложению;
11. клиентское приложение строит график по полученным данным;
12. построенный график отображается для пользователя системы;
13. пользователь системы после просмотра графика нажимает кнопку закрытия страницы;
14. клиентское приложение закрывает страницу.

В этом разделе было произведено полное описание моделей представления системы посредством двух UML-диаграмм. Раздел завершён.

**5 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ**

В данном разделе представлено описание модели данных и ее структура, которая обеспечивает работу системы.

Информационная модель – модель объекта, представленная в виде информации, описывающей существенные для данного рассмотрения параметры и переменные величины объекта, связи между ними, входы и выходы объекта и позволяющая путём подачи на модель информации об изменениях входных величин моделировать возможные состояния объекта. Информационные модели нельзя потрогать или увидеть, они не имеют материального воплощения, потому что строятся только на информации.

Информационные модели отражают различные типы систем объектов, в которых реализуются различные структуры взаимодействия и взаимосвязи между элементами системы. Для отражения систем с различными структурами используются различные типы информационных моделей: табличные, иерархические и сетевые.

Описание представлено в текстовом варианте и графическом в виде диаграммы в нотации IDEF1X.

IDEF1X является методом для разработки реляционных баз данных и использует условный синтаксис, специально разработанный для удобного построения концептуальной схемы. Концептуальной схемой называется универсальное представление структуры данных в рамках коммерческого предприятия, независимое от конечной реализации базы данных и аппаратной платформы.

Основными компонентами информационной модели в методике IDEF1X являются сущности, отношения и атрибуты. Для этих компонентов приняты специальные средства графического изображения.

Сущность определяют, как множество объектов, обладающих общими свойствами.

Конкретные элементы этого множества называют экземплярами сущности. Если сущность A может быть определена только с помощью ссылки на свойства некоторой другой сущности B, то A называют дочерней сущностью, а B выступает в роли родительской сущности.

Сущности в IDEF1X-диаграммах изображают в виде прямоугольников, причем рекомендуется у зависимых сущностей углы прямоугольников изображать скругленными.

Отношения между сущностями в IDEF1X являются бинарными отношениями. Выделяют идентифицирующие отношения – связи типа родитель-потомок, в которых потомок однозначно определяется своей связью с родителем, и неидентифицирующие отношения, означающие, что у связанного этим отношением экземпляра одной сущности может быть, а может и не быть соответствующего экземпляра второй сущности.

То есть это нотация, которая отображает логическую модель некоторой модели данных.

Логическая модель данных является визуальным представлением структур данных, их атрибутов и бизнес-правил. Логическая модель представляет данные таким образом, чтобы они легко воспринимались бизнес-пользователями.

Проектирование логической модели должно быть свободно от требований платформы и языка реализации или способа дальнейшего использования данных.

Разработчик модели использует требования к данным и результаты анализа для формирования логической модели данных. Разработчик приводит логическую модель к третьей нормальной форме и проверяет ее на соответствие корпоративной модели данных, если она существует.

Модель данных в нотации IDEF1X представлена на рисунке 5.1.

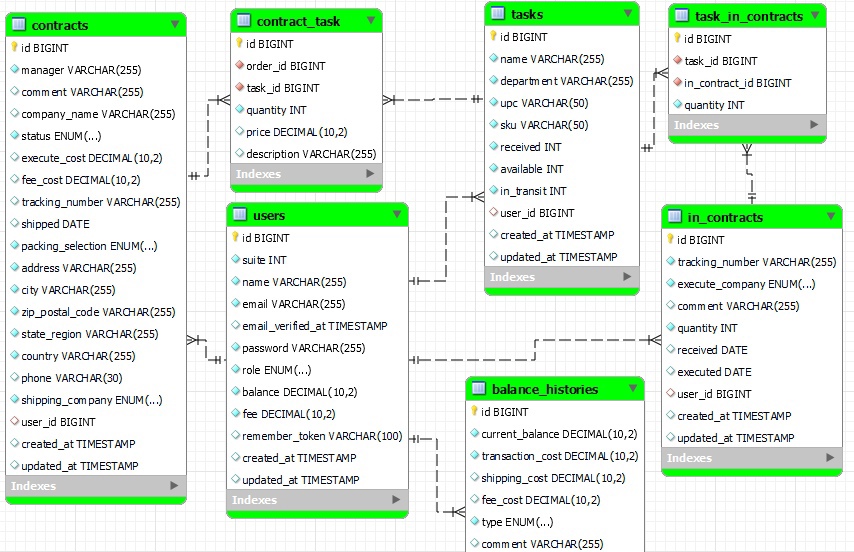


Рисунок 5.1 – Информационная модель базы данных текущего проекта

Разработка таблиц была выполнена с помощью программной среды MySQL Workbench.

Рассмотрим данные таблицы базы данных текущего проекта.

Таблица «истории баланса» содержит данные об истории баланса пользователей. Поля таблицы истории баланса:

1 id – первичный ключ таблицы идентифицирующий запись. Это поле заполняется автоматически значениями типа BIGINT.

2 current\_balance (текущий баланс) – в поле хранится информация о текущем балансе пользователя. Это поле заполняется значениями типа DECIMAL (10, 2).

3 transaction\_cost (сумма транзакции) – в поле хранится информация о сумме транзакции. Это поле заполняется значениями типа DECIMAL (10, 2).

4 shipping\_cost (стоимость доставки) – в поле хранится информация о стоимости доставки товаров. Это поле заполняется значениями типа DECIMAL (10, 2).

5 fee\_cost (стоимость комиссии) – в поле хранится информация о стоимости комиссии за выполнение операций. Это поле заполняется значениями типа DECIMAL (10, 2).

6 type (тип) – поле, указывающее тип баланса пользователя. Данное поле заполняется значениями типа ENUM (…).

7 comment (комментарий) – поле заполняется комментарием к операции над балансом. Данное поле заполняется значениями типа VARCHAR (255).

Таблица «пользователи» содержит данные об учетной записи пользователя системы. Поля таблицы пользователи:

1 id – первичный ключ таблицы идентифицирующий запись. Это поле заполняется автоматически значениями типа BIGINT.

2 suite (уникальный номер) – поле, указывающее уникальный номер пользователя. Данное поле заполняется значениями типа INT.

3 name (имя пользователя) – в поле занесены данные фамилии, имени и отчества пользователя. Это поле заполняется значениями типа VARCHAR (255).

4 email (электронная почта). В поле занесены данные электронной почты пользователя, которые служат для авторизации. Это поле заполняется значениями типа VARCHAR (255). Это поле уникальное, так как нельзя записать в таблицу пользователей с одинаковыми электронными почтами.

5 password (пароль) – хранится пароль пользователя. Поле заполняется значениями типа VARCHAR (255).

6 role (роль) – поле, указывающее роль пользователя (user или admin). Данное поле заполняется значениями типа ENUM (…).

7 balance (баланс) – в поле хранится информация о балансе данного пользователя. Это поле заполняется значениями типа DECIMAL (10, 2).

8 fee (ставка комиссии) – в поле хранится информация о ставке комиссии за операции. Это поле заполняется значениями типа DECIMAL (10, 2).

Таблица «в контракте» содержит данные о контрактах пользователя. Поля таблицы в контракте:

1 id – первичный ключ таблицы идентифицирующий запись. Это поле заполняется автоматически значениями типа BIGINT.

2 tracking\_number (код отслеживания) – в поле занесены данные о коде отслеживания. Это поле заполняется значениями типа VARCHAR (255).

3 execute\_company (компания-исполнитель) – поле, указывающее название компании исполнителя контракта. Данное поле заполняется значениями типа ENUM (…).

4 comment (комментарий) – поле заполняется комментарием к доставке товаров пользователю. Данное поле заполняется значениями типа VARCHAR (255).

5 quantity (количество) – поле, указывающее количество товара для доставки. Данное поле заполняется значениями типа INT.

6 received (полученный товар) – поле заполняется информацией о дате получения товара. Данное поле заполняется значениями типа DATE.

7 executed (исполненный контракт) – поле заполняется информацией о дате исполнения контракта. Данное поле заполняется значениями типа DATE.

Таблица задачи в контракте содержит информацию о том, какой контракт был заключен и на какое количество товаров. Данная таблица связывает две таблицы: в контракте и задачи. Поля таблицы задачи в контракте:

1 id – первичный ключ таблицы идентифицирующий запись. Это поле заполняется автоматически значениями типа BIGINT.

2 quantity (количество) – поле, указывающее количество товара в контракте. Данное поле заполняется значениями типа INT.

Таблица «задачи» содержит данные о задачах. Поля таблицы задачи:

1 id – первичный ключ таблицы идентифицирующий запись. Это поле заполняется автоматически значениями типа BIGINT.

2 name (название) – поле заполняется названием задачи, которая прописывается в контракте. Данное поле заполняется значениями типа VARCHAR (255).

3 department (отдел) – поле заполняется названием отдела, которому поручено выполнение поставленной задачи. Данное поле заполняется значениями типа VARCHAR (255).

4 upc (универсальный товарный код) – поле заполняется универсальным товарным кодом. Данное поле заполняется значениями типа VARCHAR (50).

5 sku (единица учета запасов) – поле заполняется единицей учета запасов. Данное поле заполняется значениями типа VARCHAR (50).

6 received (зарезервировано) – поле, указывающее на количество зарезервированного товара. Данное поле заполняется значениями типа INT.

7 available (доступно) – поле, указывающее на количество доступного товара. Данное поле заполняется значениями типа INT.

8 in\_transit (в пути) – поле, указывающее на количество товара, находящегося в пути. Данное поле заполняется значениями типа INT.

Таблица «задачи контракта» содержит информацию о том, к какому контракту относится какая задача. Данная таблица связывает две таблицы: контракты и задачи. Поля таблицы задачи контракта:

1 id – первичный ключ таблицы идентифицирующий запись. Это поле заполняется автоматически значениями типа BIGINT.

2 quantity (количество) – поле, указывающее на количество задач в контракте. Данное поле заполняется значениями типа INT.

3 price (цена) – в поле хранится информация о сумме, прописанной в контракте. Это поле заполняется значениями типа DECIMAL (10, 2).

4 description (описание) – поле заполняется описанием к каждой задаче контракта. Данное поле заполняется значениями типа VARCHAR (255).

Таблица «контракты» содержит данные о контрактах пользователей. Поля таблицы контракты:

1 id – первичный ключ таблицы идентифицирующий запись. Это поле заполняется автоматически значениями типа BIGINT.

2 manager (менеджер) – поле заполняется именем менеджера, который будет заключать контракт. Данное поле заполняется значениями типа VARCHAR (255).

3 comment (комментарий) – поле заполняется комментарием к контракту. Данное поле заполняется значениями типа VARCHAR (255).

4 company\_name (комания) – поле заполняется именем компании. Данное поле заполняется значениями типа VARCHAR (255).

5 status (статус) – поле, указывающее какой статус у контракта на данный момент. Данное поле заполняется значениями типа ENUM (…).

6 execute\_cost (стоимость исполнения) – в поле хранится информация о том, сколько будет стоить исполнение контракта. Это поле заполняется значениями типа DECIMAL (10, 2).

7 fee\_cost (стоимость комиссии) – в поле хранится информация о стоимости комиссии за выполнение операции над контрактом. Это поле заполняется значениями типа DECIMAL (10, 2).

8 tracking\_number (код отслеживания) – в поле занесены данные о коде отслеживания. Это поле заполняется значениями типа VARCHAR (255).

9 shipped (доставлено) – поле заполняется информацией о дате доставления контракта. Данное поле заполняется значениями типа DATE.

10 packing\_selection (тип упаковки) – поле, указывающее в какую упаковку завернуть контракт. Данное поле заполняется значениями типа ENUM (…).

11 address (адрес) – в поле занесены данные об адресе, где был заключен контракт. Это поле заполняется значениями типа VARCHAR (255).

12 city (город) – в поле занесены данные о городе, где был заключен контракт. Это поле заполняется значениями типа VARCHAR (255).

13 zip\_postal\_code (почтовый индекс) – в поле занесены данные о почтовом индексе. Это поле заполняется значениями типа VARCHAR (255).

14 state\_region (область) – в поле занесены данные об области, где был заключен контракт. Это поле заполняется значениями типа VARCHAR (255).

15 country (страна) – в поле занесены данные о стране, где был заключен контракт. Это поле заполняется значениями типа VARCHAR (255).

16 phone (номер телефона) – в поле занесены данные о номере телефона. Это поле заполняется значениями типа VARCHAR (30).

17 shipping\_company (транспортная компания) – поле, указывающее название транспортной компании, которая доставляет контракты. Данное поле заполняется значениями типа ENUM (…).

Таким образом, можно заметить, что описание основных моделей данных и способов их хранения подошло к логическому завершению. Соответственно, раздел завершён.

**6 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**АВТОМАТИЗАЦИИ ДОГОВОРНОЙ ДЕЯТЕЛЬНОСТИ ТОРГОВОГО ПРЕДПРИЯТИЯ**

**6.1 Архитектурные решения**

В данном курсовом проекте будет использоваться клиент-серверная архитектура. И для того, чтобы было проще при проектировании данного программного средства, можно воспользоваться уже готовыми решениями – фреймворками и шаблонами проектирования.

Фреймворк – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

В программной инженерии шаблон проектирования приложений – это многократно применяемое решение регулярно возникающей проблемы в рамках определённого контекста архитектуры приложения. Шаблон – это не законченное архитектурное решение, которое можно напрямую преобразовать в исходный или машинный код. Это описание подхода к решению проблемы, который можно применять в разных ситуациях.

Данное программное средство будет разрабатываться на фреймворке Laravel.

Laravel – бесплатный веб-фреймворк с открытым кодом, предназначенный для разработки с использованием архитектурной модели MVC (англ. Model View Controller – модель-представление-контроллер). Исходный код проекта размещается на GitHub.

Laravel – это удобный и популярный php-фреймворк, предназначенный для разработки сайтов. Обладает открытым исходным кодом, что позволяет его дорабатывать, адаптируя под любые проекты. Laravel-фреймворк основан на компонентах Symfony, поэтому в нем применяется кодовая база высокого качества.

Разработка сайтов на Laravel подходит для проектов любого уровня сложности, среди которых могут быть стартапы, корпоративные ресурсы, интернет-магазины, маркетплейсы, социальные сети. Фреймворк позволяет делать букинг-сервера, аукционы, мобильные приложения, сервисы на основе набора REST API. Здесь создаются ресурсы с логичной архитектурой и упорядоченной структурой, а также порталы с неограниченным набором функций.

Преимущества создания веб-приложений с использованием фреймворка Laravel:

1 Laravel-фреймворк обновляется примерно раз в полгода. Это служит гарантией безопасности его использования и увеличивает производительность движка. Laravel не нагружает сервер.

2 Данный фреймворк заслужил немалую популярность и собрал вокруг себя большое сообщество, что позволяет обмениваться навыками, а также решать вопросы любого уровня сложности. Совместное использование открыло доступ к бесплатному скачиванию базовых модулей, которые идентичны для большинства проектов.

3 Процесс установки и настройки Laravel очень прост. Фреймворк поддерживает разные версии ресурса. Разработчики могут тестировать обновления в дополнительной версии, а затем вносить изменения в основную. Встроенные во фреймворк инструменты защищают от XSS и SQL-инъекций.

Думаю, что на этом можно завершить обоснование выбора основного фреймворка для разработки данного программного средства.

Следующий фреймворк, который было решено использовать – это Bootstrap.

Bootstrap (также известен как Twitter Bootstrap) – свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

Bootstrap – это открытый и бесплатный HTML, CSS и JavaScript фреймворк, который используется веб-разработчиками для быстрой вёрстки адаптивных дизайнов сайтов и веб-приложений.

Фреймворк Bootstrap используется по всему миру не только независимыми разработчиками, но иногда и целыми компаниями.

Основная область его применения – это фронтенд разработка сайтов и интерфейсов админок. Среди аналогичных систем (Foundation, UIkit, Semantic UI, InK и других) фреймворк Bootstrap является самым популярным.

Фреймворк Bootstrap позволяет верстать сайты в несколько раз быстрее, чем это можно выполнить на «чистом» CSS и JavaScript. А в нашем мире время – это очень ценный ресурс. Кроме этого, его популярность ещё обусловлена доступностью. Она заключается в том, что на нём даже начинающий разработчик может верстать достаточно качественные макеты, которые трудно было бы выполнить без глубоких знаний веб-технологий и достаточной практики.

Преимущества Bootstrap при его использовании для frontend разработки:

– высокая скорость создания качественной адаптивной вёрстки даже начинающими веб-разработчиками (достигается это благодаря использованию готовых классов и компонентов, созданных профессионалами);

– кроссбраузерность и кроссплатформеннось (корректное отображение и работа сайта во всех поддерживаемых этим фреймворком браузерах и операционных системах);

– наличие большого количество готовых хорошо продуманных компонентов, протестированных огромным сообществом веб-разработчиков на различных устройствах;

– возможность настройки под свой проект, достигается это посредством изменения SCSS переменных и использования миксинов (можно изменить количество колонок, цвета, радиус скруглений, отступы между колонками и так далее);

– низкий порог вхождения; для работы с фреймворком не обязательно иметь «глубокие» знания по HTML, CSS, JavaScript и jQuery (достаточно знать только основы этих технологий);

– однородность дизайна и его согласованность между различными компонентами (в Bootstrap все компоненты выполнены в едином стиле);

– наличие огромных сообществ и обучающих материалов; при желании это поможет не только хорошо разобраться в фреймворке, но и найти ответы практически на любые вопросы.

Думаю, что на этом можно завершить обоснование выбора фреймворка Bootstrap для разработки данного программного средства.

И последнее, что необходимо описать в данном подразделе это шаблон проектирования MVC. Как было сказано выше, на данном паттерне основан фреймворк Laravel.

Концепция MVC (Model – View – Controller) – схема использования нескольких паттернов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. На рисунке 6.1 представлена структура паттерна MVC.

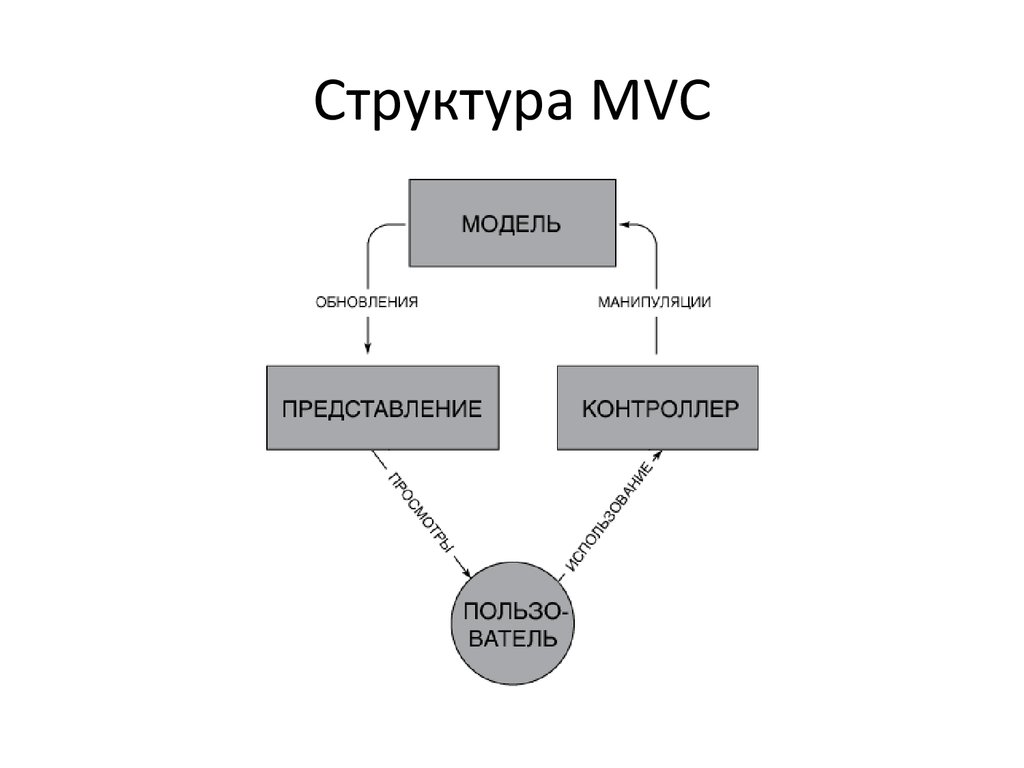


Рисунок 6.1 – Структура паттерна MVC

Концепция MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

1 Модель отвечает за управление данными, она сохраняет и извлекает сущности, используемые приложением, как правило, из базы данных и содержит логику, реализованную в приложении.

2 Представление несет ответственность за отображение данных, которые даёт контроллер. С представлением тесно связано понятие шаблона, который позволяет менять внешний вид показываемой информации. В веб-приложении представление часто реализуется в виде HTML-страницы.

3 Контроллер связывает модель и представление. Он получает запрос от клиента, анализирует его параметры и обращается к модели для выполнения операций над данными запроса. От модели поступают уже скомпонованные объекты. Затем они перенаправляются в представление, которое передаёт сформированную страницу контроллеру, а он, в свою очередь, отправляет её клиенту.

Шаблоны проектирования, такие как MVC, определяются, прежде всего, чтобы облегчить общение разработчиков между собой. Вместо того чтобы говорить коллеге: «Давай введем абстракцию доступа к данным, потом организуем отдельный уровень обработки вывода данных, а между ними вставим посредника», вы можете обратиться к общепринятому лексикону и сказать: «А давай-ка применим здесь шаблон проектирования MVC».

Основная цель применения этой концепции состоит в отделении бизнес-логики от её представления. За счет такого разделения повышается возможность повторного использования. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. При этом, выполняются следующие задачи:

1 К одной модели можно присоединить несколько видов, при этом не затрагивая реализацию модели. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы, гистограммы и круговой диаграммы.

2 Не затрагивая реализацию видов, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных), для этого достаточно использовать другой контроллер.

3 Ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический интерфейс, либо разрабатывают бизнес-логику. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики (модели), вообще не будут осведомлены о том, какое представление будет использоваться.

Важно отметить, что как представление, так и контроллер зависят от модели. Однако модель не зависит ни от представления, ни от контроллера. Тем самым достигается назначение такого разделения: оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

Также архитектуру данного программного средства можно рассмотреть на примере диаграммы компонентов и диаграммы классов.

Описание компонентов системы предполагает создание диаграммы компонентов. Данная диаграмма представлена на рисунке 6.2.

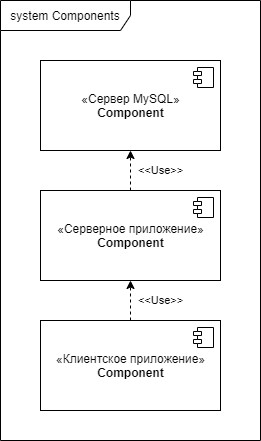


Рисунок 6.2 – Диаграмма компонентов

Программное средство имеет трехуровневую архитектуру, то есть состоит из трёх основных компонентов:

1. «Клиентское приложение» – это компонент, который выполняет построение графического пользовательского интерфейса и управление им. Использует внешние интерфейсы компонента «Серверное приложение».
2. «Серверное приложение» – это компонент, который осуществляет обработку данных, их валидацию, содержит бизнес-логику. Использует внешние интерфейсы компонента «Сервер MySQL».
3. «Сервер MySQL» – это компонент, который осуществляет работу с данными, отвечает за их хранение, позволяет их получать.

При этом, каждый из компонентов предполагается размещать на разных персональных компьютерах (или телефонах). Естественно, разместить можно и на одном девайсе.

Для описания размещения системы была создана диаграмма развёртывания. Диаграмма представлена на рисунке 6.3.

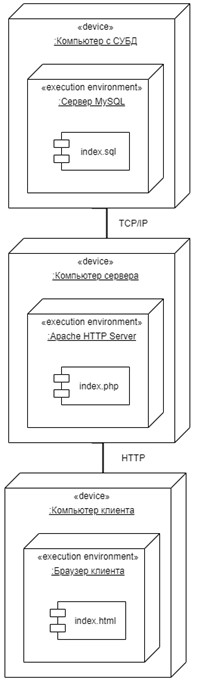


Рисунок 6.3 – Диаграмма развёртывания

На диаграмме представлены три узла:

1. «Компьютер клиент» – это устройство пользователя системы, на котором выполняется такой компонент, как «Клиентское приложения». При этом, внутри данного узла размещается узел среды выполнения «Браузер клиента». Это может быть любой современный браузер, через который клиент работает. В среде выполнения выполняется файл «index.html», который отвечает за разметку страницы и подгружает исполняемые скрипты и стили. Обменивается с узлом «Компьютер сервера» данными посредствам протокола HTTP (в перспективе – HTTPS).
2. «Компьютер сервера» – это устройство на сервере, на котором выполняется такой компонент, как «Серверное приложения». При этом, внутри данного узла размещается узел среды выполнения «Apache HTTP Server». В среде выполнения выполняется файл «index.php», который является точной входа в проект. Обменивается с узлом «Компьютер с СУБД» данными посредствам протокола TCP/IP.
3. «Компьютер с СУБД» – это устройство, на котором выполняется такой компонент, как «Сервер MySQL». При этом, внутри данного узла размещается узел среды выполнения «Сервер MySQL». В среде выполнения выполняется файл «index.sql», который является базой данных приложения.

Описанные выше компоненты и узлы позволяют организовать работу современного веб-приложения с разграничением функционала и распределением нагрузки.

Далее следует рассмотреть диаграмму классов.

Диаграмма классов – структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

Диаграмма классов данного проекта представлена на рисунке 6.4.

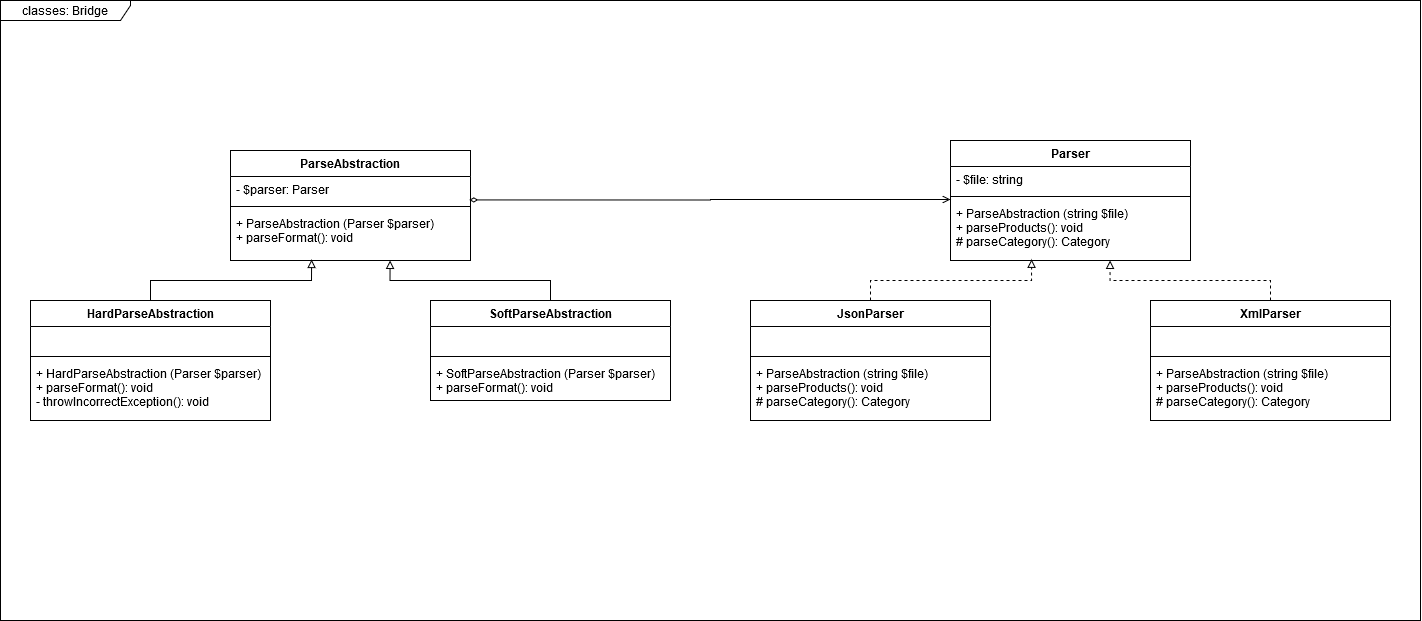


Рисунок 6.4 – Диаграмма классов

Диаграмма классов является ключевым элементом в объектно-ориентированном моделировании. На диаграмме классов представлены в рамках три компонента:

1 В верхней части написано имя класса. Имя класса выравнивается по центру и пишется полужирным шрифтом. Имена классов начинаются с заглавной буквы. Если класс абстрактный – то его имя пишется полужирным курсивом.

2 Посередине располагаются поля (атрибуты) класса. Они выровнены по левому краю и начинаются с маленькой буквы.

3 Нижняя часть содержит методы класса. Они также выровнены по левому краю.

На этом описание данного подраздела завершено. Были описаны архитектурные решения, которые будут внедрены в программное средство и представлена диаграмма компонентов, диаграмма развертывания и диаграмма классов.

**6.2 Проектирование пользовательского интерфейса**

Пользовательский интерфейс – это система средств для взаимодействия пользователя с компьютером, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и тому подобных). При этом в отличии от интерфейса командной строки, пользователь имеет произвольный доступ (с помощью клавиатуры или указательного устройства ввода) ко всем видимым экранным объектам, а на экране реализуется модель и осуществляется прямое манипулирование.

Одним из требований к хорошему графическому интерфейсу программной системы является концепция «предсказуемости», чтобы система работала предсказуемо, чтобы пользователь заранее интуитивно понимал, какое действие выполнит программа после получения его команды.

Существует три основных критерия качества пользовательского интерфейса:

– скорость работы пользователей;

– количество человеческих ошибок;

– скорость обучения.

Рассмотрим каждый из них. Первый – скорость работы пользователя. Согласно Дональду Норману, взаимодействие пользователя с системой (не только компьютерной) состоит из семи шагов:

– формирование цели действий;

– определение общей направленности действий;

– определение конкретных действий;

– выполнение действий;

– восприятие нового состояния системы;

– интерпретация состояния системы;

– оценка результата.

Таким образом, процесс размышления занимает почти все время, в течение которого пользователь работает с компьютером, т. к. шесть из семи этапов полностью заняты умственной деятельностью. Соответственно, повышение скорости этих размышлений приводит к существенному улучшению скорости работы.

Существенно повысить скорость собственно мышления пользователей невозможно, но качественный пользовательский интерфейс должен уменьшить влияние факторов, усложняющих (и, соответственно, замедляющих) процесс мышления.

Второй – количество человеческих ошибок. Пользовательский интерфейс должен содержать элементы, которые позволят уменьшить количество допускаемых ошибок. К этим элементам относятся:

– плавное обучение пользователей в процессе работы;

– снижение требований к бдительности;

– повышение разборчивости и заметности индикаторов.

Кроме того, пользовательский интерфейс должен содержать средства, позволяющие снизить чувствительность системы к ошибкам. К ним относятся:

– блокировка потенциально опасных действий пользователя до получения подтверждения правильности действия;

– проверка системой всех действий пользователя перед их принятием;

– самостоятельный выбор системой необходимых команд или параметров, когда от пользователя требуется только проверка.

И последний – это скорость обучения. Пользовательский интерфейс должен содержать средства, позволяющие пользователю в максимально короткие сроки научиться работать с программой или системой. К таким средствам относятся различного вида справочные системы, подсказки, информационные сообщения.

Все эти три критерия качества пользовательского интерфейса были учтены при проектировании программного средства автоматизации договорной деятельности торгового предприятия.

Теперь представим пользовательский интерфейс. На рисунке 6.5 представлен интерфейс для программного средства, построенный в программе Figma.

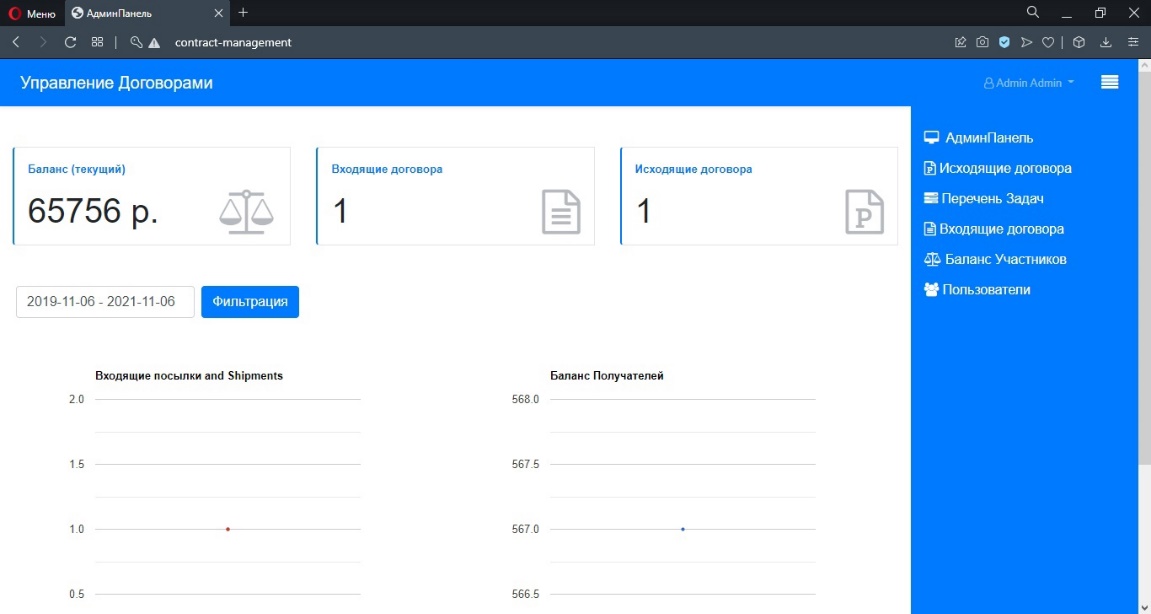


Рисунок 6.5 – Пользовательский интерфейс программного средства

На рисунке 6.6 представлен интерфейс для программного средства (мобильная версия).

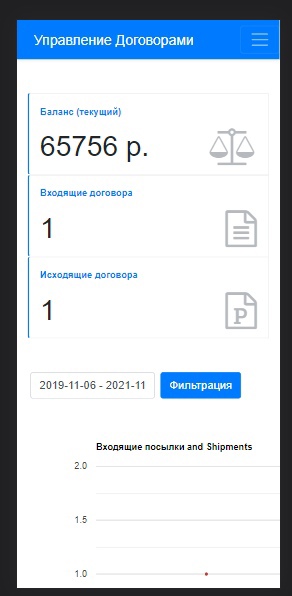


Рисунок 6.6 – Интерфейс программного средства (мобильная версия)

На этом данный подраздел завершен. В данном разделе было спроектировано программное средство автоматизации договорной деятельности торгового предприятия. Теперь можно переходить непосредственно к разработке программного средства.

**7 Описание и разработка алгоритмов,**

**реализующих бизнес-логику программного**

**средства**

Рассмотрим алгоритм фильтрации данных в таблице по введенному параметру. Данный алгоритм представлен на рисунке 7.1.



Рисунок 7.1 – Алгоритм фильтрации данных в таблице

Для фильтрации данных необходимо ввести параметр фильтрации, если параметр не будет введен система запросит его ввести. Следующий этап – это считывание параметра с поля и отправка запроса на сервер. После выполнения данной операции система будет ждать ответ. Когда ответ будет получен, данные в таблице отфильтруются.

В этом разделе было произведено описание алгоритма посредством блок-схемы. Раздел завершён.

**8 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

В данном разделе будет рассмотрен интерфейс программы, который был спроектирован в ходе выполнения курсового проекта.

Для начала пользователю необходимо будет запустить сервер на удаленном устройстве.

Для того, чтобы программное средство корректно работало, необходимо определить следующее требование, которое предъявляется к хостингу, – это поддержка MySQL версии 8.х для корректной работы с соответствующей СУБД (хранение и получение данных).

Требование определено, теперь можно приступать к развертыванию программного средства на стороннем устройстве.

Алгоритм развертывания базы данных следующий:

1) приобрести место на хостинге для развертывания базы данных;

2) открыть предоставляемый интерфейс пользователя;

3) ввести имя пользователя и пароль;

4) нажать кнопку импорта;

5) нажать «Выбрать файл», в проводнике выбрать из корневой директории проекта файл contract\_management.sql, нажать кнопку «Вперед», дождаться завершения операции.

База данных развернута.

После того, как был запущен сервер можно открывать окно авторизации. Окно авторизации необходимо для определения прав доступа, так как администратор и пользователь имеют разные права доступа и данным ролям полагается различный функционал. На рисунке 8.1 изображено окно авторизации.

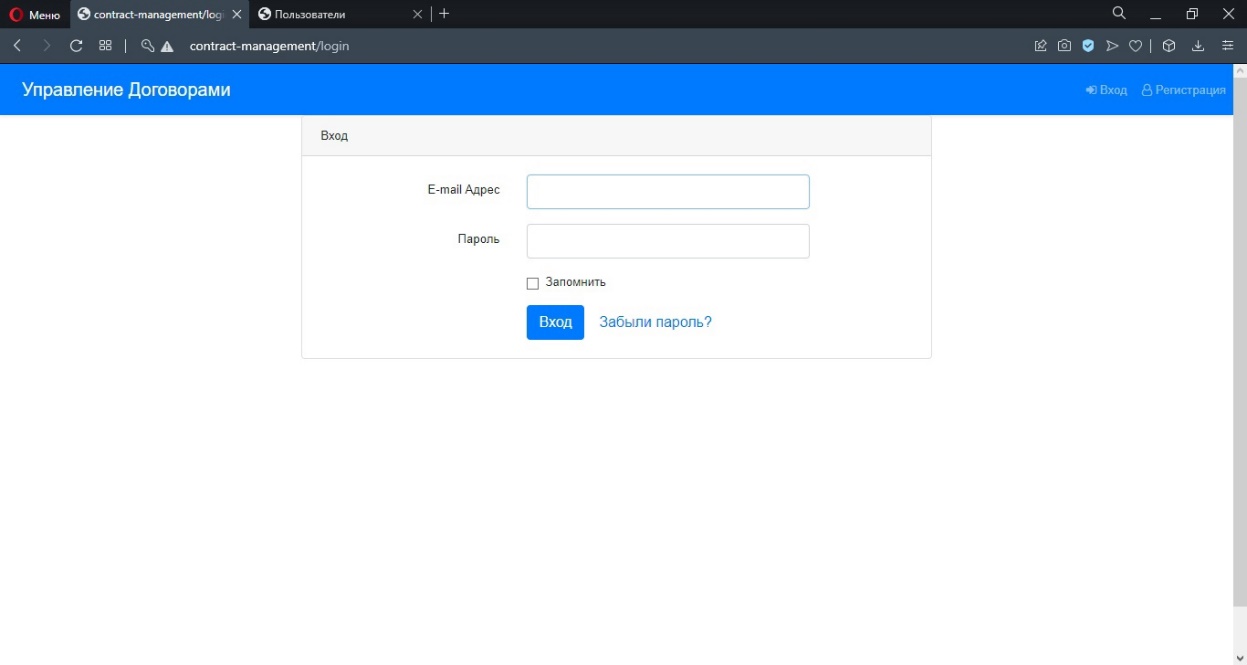


Рисунок 8.1 – Окно авторизации

Для входа в систему необходимо ввести электронную почту и пароль. По этим данным система сама определит роль для входа в систему.

После введения определенной электронной почты и пароля система считает данные с базы данных и сверит их с введенными. Если пользователь входит в систему как администратор, то система ему откроет окно администрирования. Данное окно представлено на рисунке 8.2.

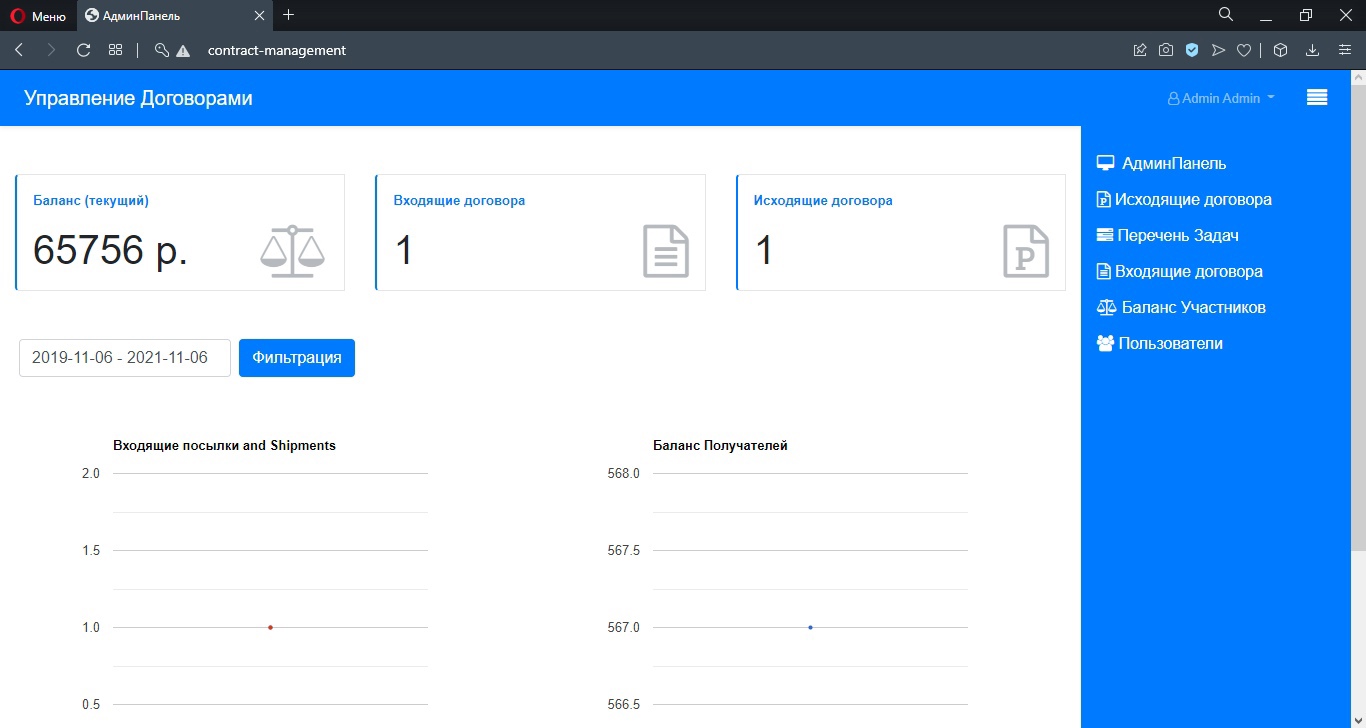


Рисунок 8.2 – Окно администрирования

В данном окне расположена информация о текущем балансе, о количестве входящих договоров, о количестве исходящих договоров. Помимо этого, здесь расположена графическая информация о входящих посылках и исходящих. А на другом графике можно посмотреть динамику баланса получателей.

Если нажать на меню справа, то откроется панель администратора. Здесь можно просмотреть информацию об исходящих договорах, перечне задач, входящих договорах, балансе участников и информацию о пользователях системы.

На рисунке 8.3 представлено окно с информацией об исходящих договорах. Здесь можно настроить сколько последних записей показывать, а также предусмотрен поиск по исходящим договорам и экспорт договоров. Помимо этого, предусмотрена сортировка по каждому из полей таблицы.

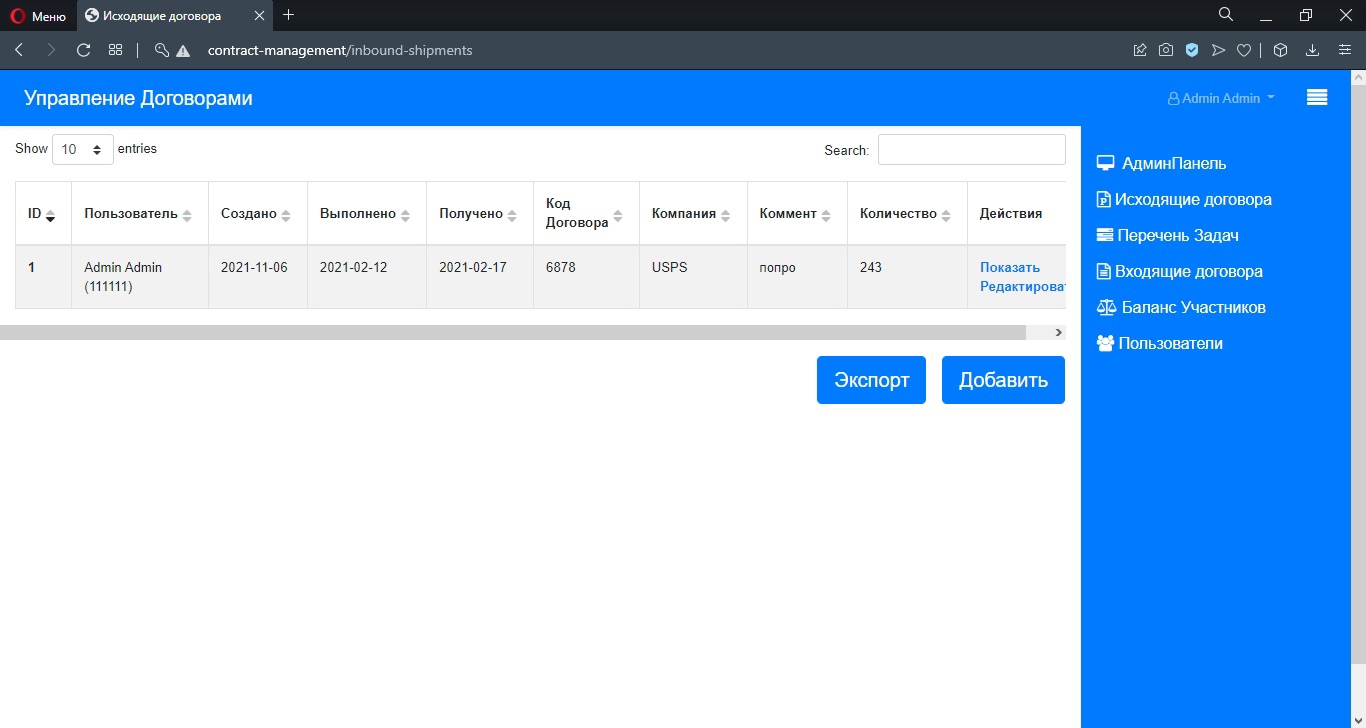


Рисунок 8.3 – Информация об исходящих договорах

На рисунке 8.4 представлено окно с информацией о перечне задач. Здесь можно настроить сколько последних записей показывать, а также предусмотрен поиск по перечню задач и экспорт задач. Помимо этого, предусмотрена сортировка по каждому из полей таблицы.

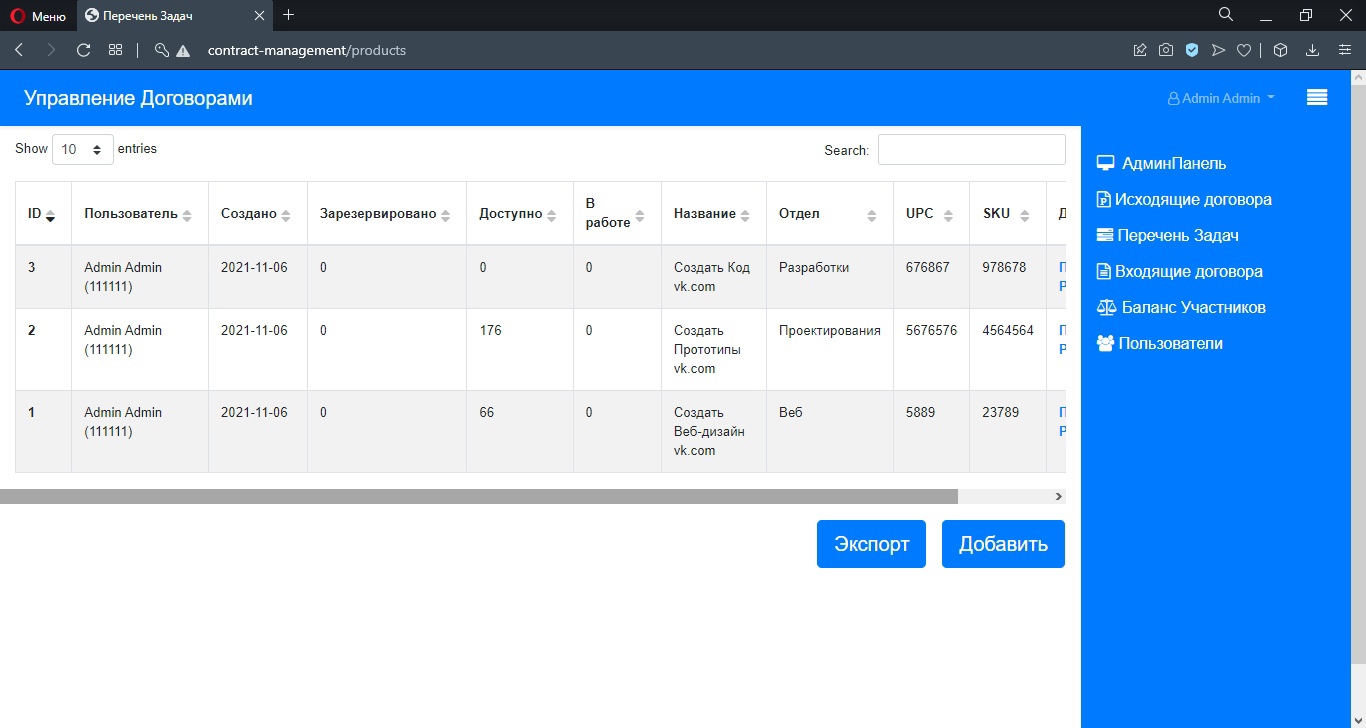


Рисунок 8.4 – Информация о перечне задач

Так как вся информация не помещается в таблицу, напротив каждого договора есть кнопка «Показать». Нажав на нее, откроется всплывающее окно с полной информацией о договоре. Идентификационный номер договора, статус, информация о пользователе, кому выполнен договор, по какому адресу, код договора, стоимость проведения, какая организация выступает в качестве перевозчика, тип упаковки, комментарий, когда создан договор и когда выполнен.

Помимо этого, есть возможность просмотреть какие вещи находятся к данном договоре, в каком количестве и за какую цену.

Данное всплывающее окно представлено на рисунке 8.5.

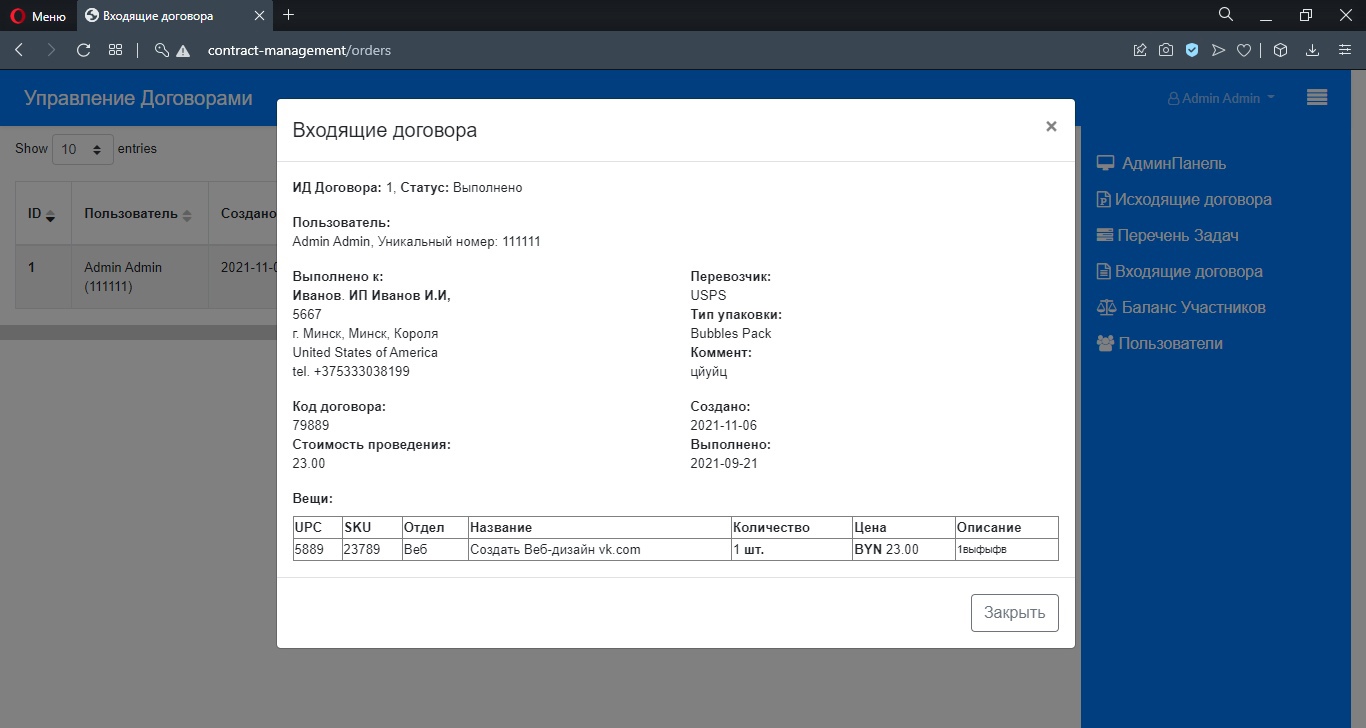


Рисунок 8.5 – Всплывающее окно с полной информацией о входящем договоре

Следующая таблица, которая предусмотрена в данной системе, – это таблица пользователи. Здесь можно просмотреть информацию по каждому пользователю системы. А также при необходимости редактировать информацию о нем, удалить данного пользователя и просмотреть его баланс, а также просмотреть полную информацию о нем. Как и в других таблицах, здесь предусмотрен показ определенного количества пользователей, поиск и сортировка.

Данная таблица представлена на рисунке 8.6.

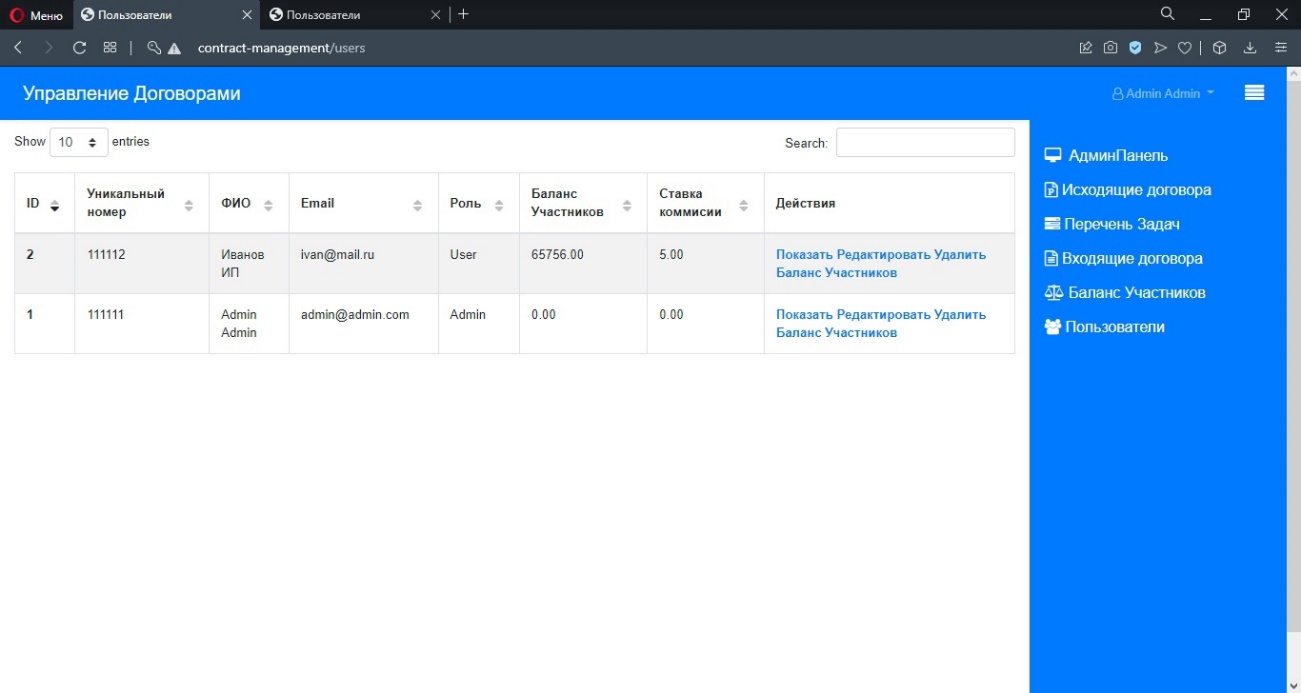


Рисунок 8.6 – Информация о пользователях системы

Чтобы изменить личную информацию, необходимо справа нажать на имя пользователя и в открывающемся списке выбрать редактирование личной информации. Откроется всплывающее окно, где можно поменять имя пользователя, электронную почту и нажать кнопку «Изменить». Для смены пароля необходимо ввести старый пароль, новый пароль и подтвердить новый пароль. Дальше нажать кнопку «Изменить пароль». Если старый пароль будет указан верно, то произойдет смена пароля. Если нет, то появится уведомление об ошибке.

Данное окно представлено на рисунке 8.7.

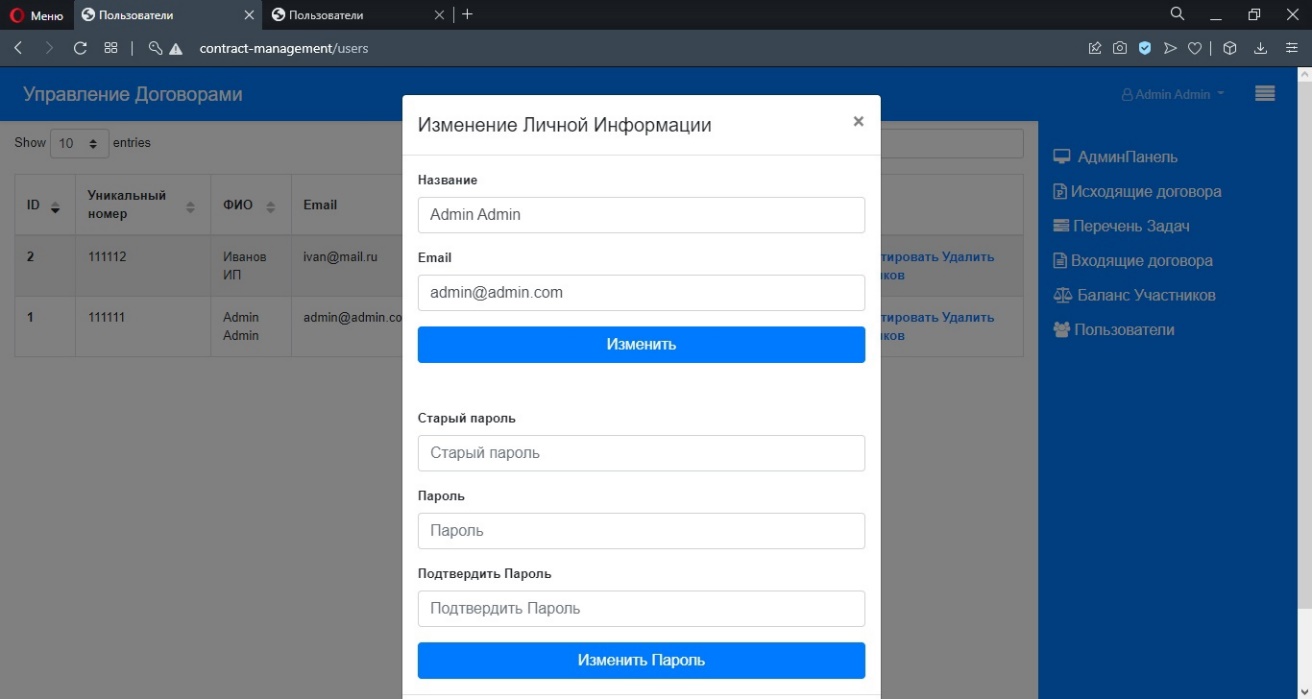


Рисунок 8.7 – Всплывающее окно для изменения личной информации

И последняя таблица, которую можно просмотреть в системе, – это баланс участников. Здесь можно настроить сколько последних записей показывать, а также предусмотрен поиск по балансу участников. Помимо этого, предусмотрена сортировка по каждому из полей таблицы.

Данная таблица представлена на рисунке 8.8.

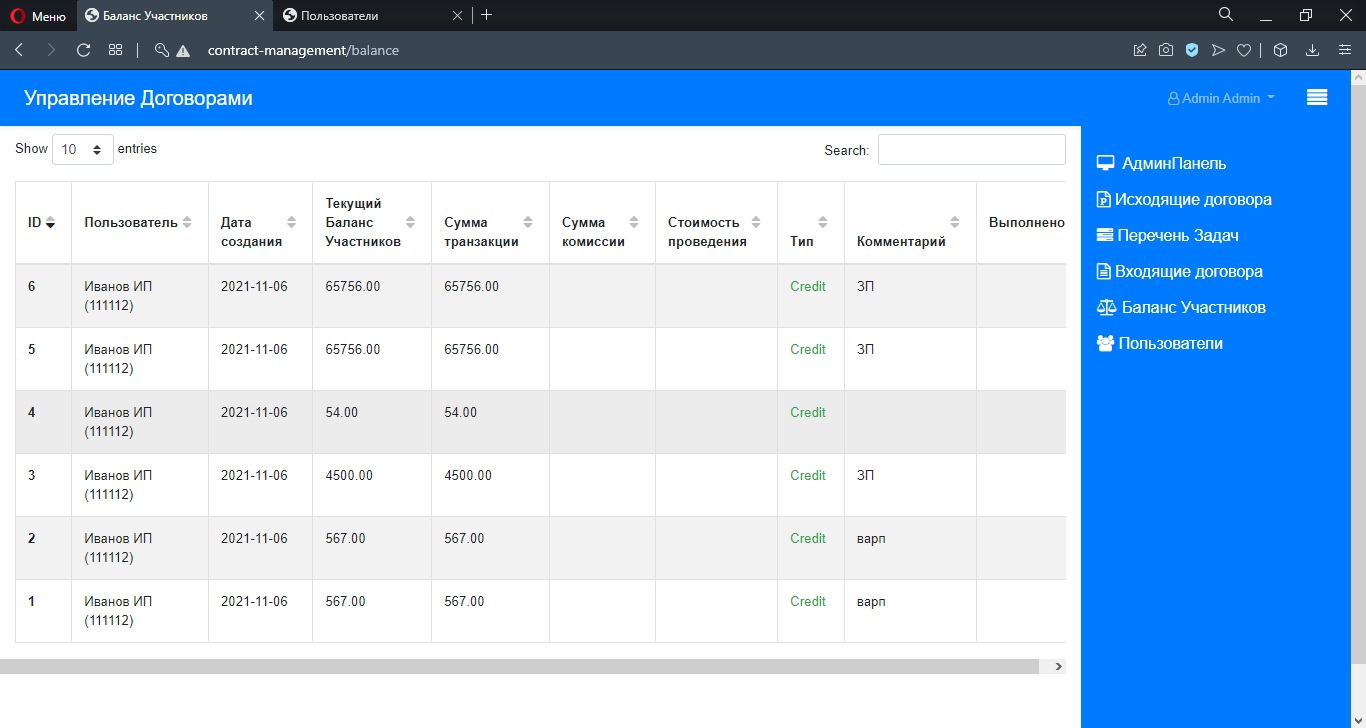


Рисунок 8.8 – Информация о балансе участников

В заключение этого раздела можно отметить, что интерфейс программы очень прост и удобен для использования любому пользователю, как администратору, так и пользователю. На этом описание данного раздела завершено.

**9 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ И ОЦЕНКА ВЫПОЛНЕНИЯ ЗАДАЧ**

Текущий завершающий раздел посвящён тестированию разработанной системы.

На текущем этапе система полностью разработана и будет проводится работа по выявлению некоторых ошибок, багов и неточностей, которые могли быть не замечены при разработке.

Стоит отметить, что тестирование зачастую проводилось именно в процессе разработки программного продукта: разработанные функции тестировались с разной нагрузкой и различными данными.

Тестирование программного обеспечения – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом.

Виды тестирования по объекту тестирования:

– функциональное тестирование;

– тестирование производительности;

– нагрузочное тестирование;

– стресс-тестирование;

– тестирование стабильности;

– конфигурационное тестирование;

– юзабилити-тестирование;

– тестирование безопасности;

– тестирование локализации;

– тестирование совместимости.

В рамках тестирования продукта будут реализованы два вида тестирования (помимо обширного модульного тестирования в процессе разработки).

Первый вид тестирования – стресс-тестирование.

Стресс-тестирование – это один из видов тестирования программного обеспечения, которое оценивает надёжность и устойчивость системы в условиях превышения пределов нормального функционирования.

Стресс-тестирование особенно необходимо для «критически важного» ПО, однако также используется и для остального ПО. Обычно стресс-тестирование лучше обнаруживает устойчивость, доступность и обработку исключений системой под большой нагрузкой, чем то, что считается корректным поведением в нормальных условиях.

То есть это тестирование в пиковый момент, под большой нагрузкой. В рамках данной задачи было проведено тестирование многих вкладок разных браузеров данной системы, проверять при работе множества клиентов.

При этом, будет отмечаться, присутствуют ли некорректные подвисания или баги системы.

Результат стресс-тестирования: в разработанной системе дополнительных багов и ошибок выявлено не было. При этом, система работала стабильно и без подвисаний и остановок работы. Следовательно, тестирование прошло успешно.

Следующий вид тестирования – это тестирование локализации.

Локализация программного обеспечения – это процесс адаптации программного обеспечения к культуре какой-либо страны. Как частность – перевод пользовательского интерфейса, документации и сопутствующих файлов программного обеспечения с одного языка на другой.

Следовательно, это тестирование системы на разных языках данной системы.

То есть мы проверяем работу системы на разных языках.

В результате проверки системы на двух языках: английском и русском, багов и некорректностей перевода выявлено не было. Следовательно, тестирование локализации пройдено успешно.

Все виды тестирования успешно пройдены, следовательно, эта система полностью исправна и готова к выпуску. Соответственно, данный раздел успешно завершён.

**ЗАКЛЮЧЕНИЕ**

На этом основное описание данного проекта подходит к концу. В результате проведённой работы удалось выполнить все цели и задачи, поставленные перед началом работы.

При выполнении курсового проекта было спроектировано и разработано программное средство автоматизации договорной деятельности торгового предприятия.

Первый этап работы заключался в постановке конкретных задач для реализации системы. Следующий этап – это этап проектирования бизнес-логики системы. Была составлена база данных с помощью sql скрипта.

Для написания сервера и пользовательской части использовалась среда разработки Visual Studio Code и высокоуровневые языки программирования PHP и JavaScript. В работе использовался паттерн проектирования MVC.

Данный проект имеет удобный и доступный пользовательский интерфейс (внешнее оформление), который делает веб-приложение простым для использования. Помимо этого, реализовано множество некрупных дополнений и улучшений, упрощающих работу, что сделало пользовательский интерфейс гораздо более удобным и понятным.

Разработанная система была протестирована, в результате чего не было выявлено серьёзных проблем, что гарантирует надёжность данной системы.

Результатом работы является рабочее программное средство автоматизации договорной деятельности торгового предприятия, предназначенное для применения на различных торговых предприятиях.

В дальнейшем планируется усовершенствование системы и добавление нового функционала.

Разработка проекта, вдобавок ко всему, помогла освоить современные средства разработки веб-приложений, оснастить компьютер современным инструментарием, приобрести бесценный опыт в освоении программирования в целом.

Таким образом, проект успешно реализован, осуществлено его подробное описание, получено много новых знаний. На этом описание проекта завершено.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Введение в методологию IDEF0 [Электронный ресурс] – Режим доступа: http://www.info-system.ru/.

[2] Диаграмма последовательности [Электронный ресурс] – Режим доступа: http://it-gost.ru/.

[3] Использование MySQL [Электронный ресурс] – Режим доступа: http://www.dev.mysql.com/.

[4] Шаблон MVC [Электронный ресурс] – Режим доступа: http://ru.wikipedia.com/.

[5] Фримен Э. О. Паттерны проектирования / Э. О. Фримен, К. А. Сьерра, Б. Н. Бейтс. – СПб : Питер, 2001. – 824с.

[6] Описание методологии IDEF0 [Электронный ресурс] – Режим доступа: http://www.logists.by/.

[7] Вводная информация об MVC [Электронный ресурс] – Режим доступа: http://webdevelop.info/.

[8] Что такое Use Case? [Электронный ресурс] – Режим доступа: http://school.system-analysis.ru/.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг скрипта генерации базы данных**

**contract\_management.sql**

-- phpMyAdmin SQL Dump

-- version 5.0.2

-- https://www.phpmyadmin.net/

--

-- Хост: 127.0.0.1:33066

-- Время создания: Ноя 06 2021 г., 19:09

-- Версия сервера: 8.0.19

-- Версия PHP: 7.4.5

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

START TRANSACTION;

SET time\_zone = "+00:00";

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101 SET NAMES utf8mb4 \*/;

--

-- База данных: `contract\_management`

--

CREATE DATABASE IF NOT EXISTS `contract\_management` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci;

USE `contract\_management`;

-- --------------------------------------------------------

--

-- Структура таблицы `balance\_histories`

--

CREATE TABLE `balance\_histories` (

`id` bigint UNSIGNED NOT NULL,

`current\_balance` decimal(10,2) NOT NULL,

`transaction\_cost` decimal(10,2) NOT NULL,

`shipping\_cost` decimal(10,2) DEFAULT NULL,

`fee\_cost` decimal(10,2) DEFAULT NULL,

`type` enum('Debit','Credit') COLLATE utf8mb4\_unicode\_ci NOT NULL,

`comment` varchar(255) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,

`user\_id` bigint UNSIGNED DEFAULT NULL,

`created\_at` timestamp NULL DEFAULT NULL,

`updated\_at` timestamp NULL DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

--

-- Дамп данных таблицы `balance\_histories`

--

INSERT INTO `balance\_histories` (`id`, `current\_balance`, `transaction\_cost`, `shipping\_cost`, `fee\_cost`, `type`, `comment`, `user\_id`, `created\_at`, `updated\_at`) VALUES

(1, '567.00', '567.00', NULL, NULL, 'Credit', 'варп', 2, '2021-11-06 12:36:16', '2021-11-06 12:36:16'),

(2, '567.00', '567.00', NULL, NULL, 'Credit', 'варп', 2, '2021-11-06 12:36:20', '2021-11-06 12:36:20'),

(3, '4500.00', '4500.00', NULL, NULL, 'Credit', 'ЗП', 2, '2021-11-06 12:38:13', '2021-11-06 12:38:13'),

(4, '54.00', '54.00', NULL, NULL, 'Credit', NULL, 2, '2021-11-06 12:38:59', '2021-11-06 12:38:59'),

(5, '65756.00', '65756.00', NULL, NULL, 'Credit', 'ЗП', 2, '2021-11-06 12:40:33', '2021-11-06 12:40:33'),

(6, '65756.00', '65756.00', NULL, NULL, 'Credit', 'ЗП', 2, '2021-11-06 12:42:42', '2021-11-06 12:42:42');

-- --------------------------------------------------------

--

-- Структура таблицы `failed\_jobs`

--

CREATE TABLE `failed\_jobs` (

`id` bigint UNSIGNED NOT NULL,

`connection` text COLLATE utf8mb4\_unicode\_ci NOT NULL,

`queue` text COLLATE utf8mb4\_unicode\_ci NOT NULL,

`payload` longtext COLLATE utf8mb4\_unicode\_ci NOT NULL,

`exception` longtext COLLATE utf8mb4\_unicode\_ci NOT NULL,

`failed\_at` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

-- --------------------------------------------------------

--

-- Структура таблицы `migrations`

--

CREATE TABLE `migrations` (

`id` int UNSIGNED NOT NULL,

`migration` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`batch` int NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

--

-- Дамп данных таблицы `migrations`

--

INSERT INTO `migrations` (`id`, `migration`, `batch`) VALUES

(1, '2014\_10\_12\_000000\_create\_users\_table', 1),

(2, '2014\_10\_12\_100000\_create\_password\_resets\_table', 1),

(3, '2019\_08\_19\_000000\_create\_failed\_jobs\_table', 1),

(4, '2020\_07\_24\_180536\_create\_shipments\_table', 1),

(5, '2020\_07\_24\_181829\_create\_products\_table', 1),

(6, '2020\_07\_24\_182803\_create\_orders\_table', 1),

(7, '2020\_07\_24\_183733\_create\_product\_shipment\_table', 1),

(8, '2020\_07\_24\_183929\_create\_order\_product\_table', 1),

(9, '2020\_07\_25\_165748\_create\_balance\_histories\_table', 1);

-- --------------------------------------------------------

--

-- Структура таблицы `orders`

--

CREATE TABLE `orders` (

`id` bigint UNSIGNED NOT NULL,

`customer` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`comment` varchar(255) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,

`company\_name` varchar(255) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,

`status` enum('Создано','Выполнено') CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci NOT NULL DEFAULT 'Создано',

`shipping\_cost` decimal(10,2) DEFAULT NULL,

`fee\_cost` decimal(10,2) DEFAULT NULL,

`tracking\_number` varchar(255) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,

`shipped` date DEFAULT NULL,

`packing\_selection` enum('Bubbles Pack','Carton') COLLATE utf8mb4\_unicode\_ci NOT NULL,

`address` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`city` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`zip\_postal\_code` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`state\_region` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`country` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`phone` varchar(30) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,

`shipping\_company` enum('USPS','FedEx','DHL','UPS','APC') COLLATE utf8mb4\_unicode\_ci NOT NULL,

`user\_id` bigint UNSIGNED DEFAULT NULL,

`created\_at` timestamp NULL DEFAULT NULL,

`updated\_at` timestamp NULL DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

--

-- Дамп данных таблицы `orders`

--

INSERT INTO `orders` (`id`, `customer`, `comment`, `company\_name`, `status`, `shipping\_cost`, `fee\_cost`, `tracking\_number`, `shipped`, `packing\_selection`, `address`, `city`, `zip\_postal\_code`, `state\_region`, `country`, `phone`, `shipping\_company`, `user\_id`, `created\_at`, `updated\_at`) VALUES

(1, 'Иванов', 'цйуйц', 'ИП Иванов И.И,', 'Выполнено', '23.00', NULL, '79889', '2021-09-21', 'Bubbles Pack', 'Короля', 'г. Минск', '5667', 'Минск', 'United States of America', '+375333038199', 'USPS', 1, '2021-11-06 13:03:58', '2021-11-06 13:03:58');

-- --------------------------------------------------------

--

-- Структура таблицы `order\_product`

--

CREATE TABLE `order\_product` (

`id` bigint UNSIGNED NOT NULL,

`order\_id` bigint UNSIGNED NOT NULL,

`product\_id` bigint UNSIGNED NOT NULL,

`quantity` int UNSIGNED NOT NULL,

`price` decimal(10,2) DEFAULT NULL,

`description` varchar(255) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

--

-- Дамп данных таблицы `order\_product`

--

INSERT INTO `order\_product` (`id`, `order\_id`, `product\_id`, `quantity`, `price`, `description`) VALUES

(1, 1, 1, 1, '23.00', '1выфыфв');

-- --------------------------------------------------------

--

-- Структура таблицы `password\_resets`

--

CREATE TABLE `password\_resets` (

`email` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`token` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`created\_at` timestamp NULL DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

-- --------------------------------------------------------

--

-- Структура таблицы `products`

--

CREATE TABLE `products` (

`id` bigint UNSIGNED NOT NULL,

`name` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`brand` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`upc` varchar(50) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`sku` varchar(50) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`received` int NOT NULL DEFAULT '0',

`available` int NOT NULL DEFAULT '0',

`in\_transit` int NOT NULL DEFAULT '0',

`user\_id` bigint UNSIGNED DEFAULT NULL,

`created\_at` timestamp NULL DEFAULT NULL,

`updated\_at` timestamp NULL DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

--

-- Дамп данных таблицы `products`

--

INSERT INTO `products` (`id`, `name`, `brand`, `upc`, `sku`, `received`, `available`, `in\_transit`, `user\_id`, `created\_at`, `updated\_at`) VALUES

(1, 'Создать Веб-дизайн vk.com', 'Веб', '5889', '23789', 0, 66, 0, 1, '2021-11-06 12:52:34', '2021-11-06 13:03:58'),

(2, 'Создать Прототипы vk.com', 'Проектирования', '5676576', '4564564', 0, 176, 0, 1, '2021-11-06 12:54:33', '2021-11-06 12:58:12'),

(3, 'Создать Код vk.com', 'Разработки', '676867', '978678', 0, 0, 0, 1, '2021-11-06 12:55:20', '2021-11-06 12:55:20');

-- --------------------------------------------------------

--

-- Структура таблицы `product\_shipment`

--

CREATE TABLE `product\_shipment` (

`id` bigint UNSIGNED NOT NULL,

`product\_id` bigint UNSIGNED NOT NULL,

`shipment\_id` bigint UNSIGNED NOT NULL,

`quantity` int UNSIGNED NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

--

-- Дамп данных таблицы `product\_shipment`

--

INSERT INTO `product\_shipment` (`id`, `product\_id`, `shipment\_id`, `quantity`) VALUES

(1, 1, 1, 67),

(2, 2, 1, 176);

-- --------------------------------------------------------

--

-- Структура таблицы `shipments`

--

CREATE TABLE `shipments` (

`id` bigint UNSIGNED NOT NULL,

`tracking\_number` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`shipping\_company` enum('USPS','FedEx','DHL','UPS','APC') COLLATE utf8mb4\_unicode\_ci NOT NULL,

`comment` varchar(255) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,

`quantity` int UNSIGNED NOT NULL,

`received` date DEFAULT NULL,

`shipped` date DEFAULT NULL,

`user\_id` bigint UNSIGNED DEFAULT NULL,

`created\_at` timestamp NULL DEFAULT NULL,

`updated\_at` timestamp NULL DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

--

-- Дамп данных таблицы `shipments`

--

INSERT INTO `shipments` (`id`, `tracking\_number`, `shipping\_company`, `comment`, `quantity`, `received`, `shipped`, `user\_id`, `created\_at`, `updated\_at`) VALUES

(1, '6878', 'USPS', 'попро', 243, '2021-02-17', '2021-02-12', 1, '2021-11-06 12:58:12', '2021-11-06 12:58:12');

-- --------------------------------------------------------

--

-- Структура таблицы `users`

--

CREATE TABLE `users` (

`id` bigint UNSIGNED NOT NULL,

`suite` int NOT NULL,

`name` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`email` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`email\_verified\_at` timestamp NULL DEFAULT NULL,

`password` varchar(255) COLLATE utf8mb4\_unicode\_ci NOT NULL,

`role` enum('User','Admin') COLLATE utf8mb4\_unicode\_ci NOT NULL DEFAULT 'User',

`balance` decimal(10,2) NOT NULL DEFAULT '0.00',

`fee` decimal(10,2) NOT NULL DEFAULT '0.00',

`remember\_token` varchar(100) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,

`created\_at` timestamp NULL DEFAULT NULL,

`updated\_at` timestamp NULL DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;

--

-- Дамп данных таблицы `users`

--

INSERT INTO `users` (`id`, `suite`, `name`, `email`, `email\_verified\_at`, `password`, `role`, `balance`, `fee`, `remember\_token`, `created\_at`, `updated\_at`) VALUES

(1, 111111, 'Admin Admin', 'admin@admin.com', NULL, '$2y$10$PYQMJNVkQYS.r3Z1tWZnbeu3EXpgEw4ym9fUCk0jTSr1SlhpasC1S', 'Admin', '0.00', '0.00', NULL, '2021-11-06 11:19:45', '2021-11-06 11:19:45'),

(2, 111112, 'Иванов ИП', 'ivan@mail.ru', NULL, '$2y$10$SGxXonX6gqT/FgnF2/BoJ.xwpHCl104KMiekb97bppbwer7oohVHS', 'User', '65756.00', '5.00', NULL, '2021-11-06 12:33:01', '2021-11-06 12:42:42');

--

-- Индексы сохранённых таблиц

--

--

-- Индексы таблицы `balance\_histories`

--

ALTER TABLE `balance\_histories`

ADD PRIMARY KEY (`id`),

ADD KEY `balance\_histories\_user\_id\_foreign` (`user\_id`);

--

-- Индексы таблицы `failed\_jobs`

--

ALTER TABLE `failed\_jobs`

ADD PRIMARY KEY (`id`);

--

-- Индексы таблицы `migrations`

--

ALTER TABLE `migrations`

ADD PRIMARY KEY (`id`);

--

-- Индексы таблицы `orders`

--

ALTER TABLE `orders`

ADD PRIMARY KEY (`id`),

ADD KEY `orders\_user\_id\_foreign` (`user\_id`);

--

-- Индексы таблицы `order\_product`

--

ALTER TABLE `order\_product`

ADD PRIMARY KEY (`id`),

ADD KEY `order\_product\_order\_id\_foreign` (`order\_id`),

ADD KEY `order\_product\_product\_id\_foreign` (`product\_id`);

--

-- Индексы таблицы `password\_resets`

--

ALTER TABLE `password\_resets`

ADD KEY `password\_resets\_email\_index` (`email`);

--

-- Индексы таблицы `products`

--

ALTER TABLE `products`

ADD PRIMARY KEY (`id`),

ADD KEY `products\_user\_id\_foreign` (`user\_id`);

--

-- Индексы таблицы `product\_shipment`

--

ALTER TABLE `product\_shipment`

ADD PRIMARY KEY (`id`),

ADD KEY `product\_shipment\_product\_id\_foreign` (`product\_id`),

ADD KEY `product\_shipment\_shipment\_id\_foreign` (`shipment\_id`);

--

-- Индексы таблицы `shipments`

--

ALTER TABLE `shipments`

ADD PRIMARY KEY (`id`),

ADD KEY `shipments\_user\_id\_foreign` (`user\_id`);

--

-- Индексы таблицы `users`

--

ALTER TABLE `users`

ADD PRIMARY KEY (`id`),

ADD UNIQUE KEY `users\_email\_unique` (`email`);

--

-- AUTO\_INCREMENT для сохранённых таблиц

--

--

-- AUTO\_INCREMENT для таблицы `balance\_histories`

--

ALTER TABLE `balance\_histories`

MODIFY `id` bigint UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=7;

--

-- AUTO\_INCREMENT для таблицы `failed\_jobs`

--

ALTER TABLE `failed\_jobs`

MODIFY `id` bigint UNSIGNED NOT NULL AUTO\_INCREMENT;

--

-- AUTO\_INCREMENT для таблицы `migrations`

--

ALTER TABLE `migrations`

MODIFY `id` int UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=10;

--

-- AUTO\_INCREMENT для таблицы `orders`

--

ALTER TABLE `orders`

MODIFY `id` bigint UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=2;

--

-- AUTO\_INCREMENT для таблицы `order\_product`

--

ALTER TABLE `order\_product`

MODIFY `id` bigint UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=2;

--

-- AUTO\_INCREMENT для таблицы `products`

--

ALTER TABLE `products`

MODIFY `id` bigint UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=4;

--

-- AUTO\_INCREMENT для таблицы `product\_shipment`

--

ALTER TABLE `product\_shipment`

MODIFY `id` bigint UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=3;

--

-- AUTO\_INCREMENT для таблицы `shipments`

--

ALTER TABLE `shipments`

MODIFY `id` bigint UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=2;

--

-- AUTO\_INCREMENT для таблицы `users`

--

ALTER TABLE `users`

MODIFY `id` bigint UNSIGNED NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=3;

--

-- Ограничения внешнего ключа сохраненных таблиц

--

--

-- Ограничения внешнего ключа таблицы `balance\_histories`

--

ALTER TABLE `balance\_histories`

ADD CONSTRAINT `balance\_histories\_user\_id\_foreign` FOREIGN KEY (`user\_id`) REFERENCES `users` (`id`) ON DELETE SET NULL ON UPDATE SET NULL;

--

-- Ограничения внешнего ключа таблицы `orders`

--

ALTER TABLE `orders`

ADD CONSTRAINT `orders\_user\_id\_foreign` FOREIGN KEY (`user\_id`) REFERENCES `users` (`id`) ON DELETE SET NULL ON UPDATE SET NULL;

--

-- Ограничения внешнего ключа таблицы `order\_product`

--

ALTER TABLE `order\_product`

ADD CONSTRAINT `order\_product\_order\_id\_foreign` FOREIGN KEY (`order\_id`) REFERENCES `orders` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

ADD CONSTRAINT `order\_product\_product\_id\_foreign` FOREIGN KEY (`product\_id`) REFERENCES `products` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

--

-- Ограничения внешнего ключа таблицы `products`

--

ALTER TABLE `products`

ADD CONSTRAINT `products\_user\_id\_foreign` FOREIGN KEY (`user\_id`) REFERENCES `users` (`id`) ON DELETE SET NULL ON UPDATE SET NULL;

--

-- Ограничения внешнего ключа таблицы `product\_shipment`

--

ALTER TABLE `product\_shipment`

ADD CONSTRAINT `product\_shipment\_product\_id\_foreign` FOREIGN KEY (`product\_id`) REFERENCES `products` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

ADD CONSTRAINT `product\_shipment\_shipment\_id\_foreign` FOREIGN KEY (`shipment\_id`) REFERENCES `shipments` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

--

-- Ограничения внешнего ключа таблицы `shipments`

--

ALTER TABLE `shipments`

ADD CONSTRAINT `shipments\_user\_id\_foreign` FOREIGN KEY (`user\_id`) REFERENCES `users` (`id`) ON DELETE SET NULL ON UPDATE SET NULL;

COMMIT;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Листинг программного кода**

@extends('layouts.app')

@section('title')

Исходящие договора

@endsection

@section('content')

<div class="main-container">

@if (session('success'))

<div class="alert alert-success" role="alert">

{{ session('success') }}

</div>

@endif

@if (session('error'))

<div class="alert alert-danger" role="alert">

{{ session('error') }}

</div>

@endif

<div class="table-container">

<table class="table table-bordered table-striped table-hover" id="dtEntityTable">

<thead class="thead-primary">

<tr>

<th scope="col" class="th-sm">ID</th>

@if (Auth::user()->role == 'Admin')

<th scope="col" class="th-sm">Пользователь</th>

@endif

<th scope="col" class="th-sm">Создано</th>

<th scope="col" class="th-sm">Выполнено</th>

<th scope="col" class="th-sm">Получено</th>

<th scope="col" class="th-sm">Код Договора</th>

<th scope="col" class="th-sm">Компания</th>

<th scope="col" class="th-sm">Коммент</th>

<th scope="col" class="th-sm">Количество</th>

<th scope="col" class="th-sm">Действия</th>

</tr>

</thead>

<tbody>

@foreach($shipments as $shipment)

<tr>

<th scope="row">{{ $shipment->id }}</th>

@if (Auth::user()->role == 'Admin')

<td>{{ $shipment->user->name }} ({{ $shipment->user->suite }})</td>

@endif

<td>{{ $shipment->created\_at->format('Y-m-d') }}</td>

<td>{{ $shipment->shipped }}</td>

<td>{{ $shipment->received }}</td>

<td>{{ $shipment->tracking\_number }}</td>

<td>{{ $shipment->shipping\_company }}</td>

<td>{{ $shipment->comment }}</td>

<td>{{ $shipment->quantity }}</td>

<td>

<a href="#" class="show-shipment text-primary font-weight-bold show-entity-button" data-value-id="{{ $shipment->id }}">Показать</a>

@if(Auth::user()->role == 'Admin' || $shipment->received == null)

<a href="#" class="edit-shipment text-primary font-weight-bold edit-entity-button" data-value-id="{{ $shipment->id }}">Редактировать</a>

@endif

@if($shipment->received == null && Auth::user()->role != 'Admin')

<a href="#" class="show-shipment text-primary font-weight-bold delete-entity-button"

data-value-id="{{ $shipment->id }}">Удалить</a>

@endif

</td>

</tr>

@endforeach

</tbody>

</table>

</div>

<button type="button" class="btn btn-primary btn-lg float-right my-3 mr-3 create-shipment" data-toggle="modal"

data-target="#modalAdd">Добавить

</button>

<form action="{{ route('parse') }}" method="post" enctype="multipart/form-data">

@csrf

<input type="hidden" name="type" value="shipments" class="display-none">

<input id="import-input" type="file" name="file" class="display-none" accept=".csv, .xlsx, .xls">

<input id="import-submit" type="submit" value="Submit" class="display-none">

{{-- <button id="import-open" type="button" class="btn btn-primary btn-lg float-right my-3 mr-3">Import</button>--}}

</form>

<a href="{{route('exportShipments')}}" class="btn btn-primary btn-lg float-right my-3 mr-3">Экспорт</a>

</div>

<div class="container-fluid mt-5">

<div class="row">

<div class="col-12 text-center-mobile">

<a href="{{ route('inbound-shipments') }}" class="badge badge-primary text-full-size"><i class="fa fa-file-powerpoint-o" aria-hidden="true"></i> Исходящие договора</a>

<a href="{{ route('products') }}" class="badge badge-primary text-full-size"><i class="fa fa-tasks" aria-hidden="true"></i> Перечень Задач</a>

<a href="{{ route('orders') }}" class="badge badge-primary text-full-size"><i class="fa fa-file-text-o" aria-hidden="true"></i> Входящие договора</a>

<a href="{{ route('balance') }}" class="badge badge-primary text-full-size"><i class="fa fa-balance-scale" aria-hidden="true"></i> Баланс Участников</a>

@if(\Illuminate\Support\Facades\Auth::user())

@if(\Illuminate\Support\Facades\Auth::user()->role == 'Admin')

<a href="{{ route('users') }}" class="badge badge-primary text-full-size"><i class="fa fa-users" aria-hidden="true"></i> Пользователи</a>

@endif

@endif

</div>

</div>

</div>

@endsection

@section('modal')

<div class="modal fade" data-backdrop="static" id="modalAdd" tabindex="-1" role="dialog"

aria-labelledby="modalAddLabel"

aria-hidden="true">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="modalAddLabel">Добавить Входящий договор</h5>

<button type="button" class="close close-modal-button" data-dismiss="modal" aria-label="Закрыть">

<span aria-hidden="true">&times;</span>

</button>

</div>

<form class="form-submit">

<div class="modal-body">

@csrf

<div class="form-group">

<label for="tracking\_number-number">Код Договора</label>

<input type="text" class="form-control" required maxlength="255" id="tracking\_number"

aria-describedby="ariaDescribedbyHelp" placeholder="Код Договора">

<small id="ariaDescribedbyHelp" class="form-text text-danger"></small>

</div>

<div class="form-group">

<label for="shipping\_company">Компания</label>

<select class="form-control" id="shipping\_company">

<option>USPS</option>

<option>FedEx</option>

<option>DHL</option>

<option>UPS</option>

<option>APC</option>

</select>

</div>

<div class="form-group">

<label for="comment">Коммент</label>

<textarea rows="5" class="form-control" maxlength="255" id="comment"

aria-describedby="commentHelp" placeholder="comment"></textarea>

<small id="commentHelp" class="form-text text-danger"></small>

</div>

<div class="form-group">

<label for="shipped">Дата исполнения</label>

<input type="date" class="form-control" id="shipped" name="date" placeholder="date"

aria-describedby="dateHelp" required>

<small id="dateHelp" class="form-text text-danger"></small>

</div>

@if(\Illuminate\Support\Facades\Auth::user()->role == 'Admin')

<div class="form-group">

<label for="received">Дата получения</label>

<input type="date" class="form-control" id="received" name="date" placeholder="date"

aria-describedby="dateHelp">

<small id="dateHelp" class="form-text text-danger"></small>

</div>

@endif

<div class="form-group">

<label for="productFormControlSelect1">Выбор Продукта</label>

<div class="products-container">

{{-- <div class="product-container">--}}

{{-- <select class="form-control product-select product-shipment-select"></select>--}}

{{-- <input type="number" class="form-control quantity" placeholder="quantity"--}}

{{-- required min="1" max="10000">--}}

{{-- <a href="#" class="remove-product-select">--}}

{{-- <i class="fa fa-times fa-2x text-primary" aria-hidden="true"></i>--}}

{{-- </a>--}}

{{-- </div>--}}

</div>

<div class="button-plus mt-2 mr-2">

<a href="#" class="add-product-select">

<i class="fa fa-2x fa-plus text-primary" aria-hidden="true"></i>

</a>

</div>

</div>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-outline-secondary close-modal-button" data-dismiss="modal">Закрыть

</button>

<button type="submit" class="btn btn-primary save-changes">Создать</button>

</div>

</form>

</div>

</div>

</div>

<div class="modal fade" id="showModal" tabindex="-1" role="dialog"

aria-labelledby="showModalLabel"

aria-hidden="true">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="showModalLabel">Исходящие договора</h5>

<button type="button" class="close" data-dismiss="modal" aria-label="Закрыть">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<form>

<div class="form-group">

<label class="font-weight-bold" for="showId">ID</label>

<span class="form-control form-control-height " id="showId"></span>

</div>

<div class="form-group">

<label class="font-weight-bold" for="showUser">Пользователь</label>

<span class="form-control form-control-height " id="showUser"></span>

</div>

<div class="form-group">

<label class="font-weight-bold" for="showShipped">Выполнено</label>

<span class="form-control form-control-height " id="showShipped"></span>

</div>

<div class="form-group">

<label class="font-weight-bold" for="showReceived">Получено</label>

<span class="form-control form-control-height " id="showReceived"></span>

</div>

<div class="form-group">

<label class="font-weight-bold" for="showCompany">Компания</label>

<span class="form-control form-control-height " id="showCompany"></span>

</div>

<div class="form-group">

<label class="font-weight-bold" for="showTrackingNumber">Код Договора</label>

<span class="form-control form-control-height " id="showTrackingNumber"></span>

</div>

<div class="form-group">

<label class="font-weight-bold" for="showComment">Коммент</label>

<span class="form-control form-control-height " id="showComment"></span>

</div>

<div class="form-group">

<label class="font-weight-bold" for="showQuantity">Количество</label>

<span class="form-control form-control-height " id="showQuantity"></span>

</div>

<div class="form-group">

<label class="font-weight-bold" for="showCreated">Создано</label>

<span class="form-control form-control-height " id="showCreated"></span>

</div>

<div class="form-group">

<label class="font-weight-bold">Перечень Задач</label>

<div class="show-products-container">

<div class="show-product-container">

<span class="form-control form-control-height show-product">Идентификатор</span>

<span class="form-control form-control-height show-quantity quantity-width"><span class="show-quantity"></span><span class="font-weight-bold"> шт.</span></span>

</div>

</div>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-outline-secondary" data-dismiss="modal">Закрыть</button>

</div>

</div>

</div>

</div>

@endsection

@section('scripts')

<script src="{{ asset('js/shipment.js') }}" defer></script>

<script src="{{ asset('js/select2.min.js') }}"></script>

@endsection

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\User;

use App\Http\Requests\UserRequest;

use App\BalanceHistory;

use Illuminate\Support\Facades\Session;

use Illuminate\Support\Facades\Auth;

use Illuminate\Support\Facades\Hash;

use App\Http\Requests\UserSelfUpdateRequest;

use App\Http\Requests\ChangePasswordRequest;

use Illuminate\Http\Exceptions\HttpResponseException;

use App\Http\Requests\AddBalanceRequest;

class UserController extends Controller

{

public function index()

{

return response()->json(User::all(), 200);

}

public function show($id)

{

return response()->json(User::with(['balanceHistories' => function($query) {

$query->orderBy('created\_at', 'DESC');

}])->find($id), 200);

}

public function update(UserRequest $request, $id)

{

$user = User::find($id);

// $this->createBalanceHistory($user, $request->balance);

$user->role = $request->role;

$user->fee = $request->fee;

$user->save();

Session::flash('success', 'Пользователь обновлен!');

return response()->json($user, 200);

}

public function addBalance(AddBalanceRequest $request, $id)

{

$user = User::find($id);

$this->createBalanceHistory($user, $request->balance + $user->balance, $request->comment);

$user->balance += $request->balance;

$user->save();

Session::flash('success', 'Пользователь обновлен!');

return response()->json($user, 200);

}

public function updateSelf(UserSelfUpdateRequest $request, $id)

{

$user = User::find($id);

if(Auth::user()->id = $user->id) {

$user->name = $request->name;

$user->email = $request->email;

$user->save();

Auth::attempt(['email' => $user->email, 'password' => $user->password]);

Session::flash('success', 'Личная информация обновлена!');

return response()->json($user, 200);

} else {

Session::flash('error', 'Это не ваш аккаунт!');

return response()->json(null, 200);

}

}

public function changePassword(ChangePasswordRequest $request, $id)

{

$user = User::find($id);

if(Auth::user()->id = $user->id) {

if(Hash::check($request->last\_password, $user->password)){

$user->password = Hash::make($request->password);

$user->save();

Auth::attempt(['email' => $user->email, 'password' => $user->password]);

Session::flash('success', 'Пароль Изменён!');

return response()->json($user, 200);

} else {

throw new HttpResponseException(response()->json(['last\_password' => ['It isn\'t your last password']], 403));

}

} else {

Session::flash('error', 'Это не ваш аккаунт!');

return response()->json(null, 200);

}

}

public function destroy($id)

{

Session::flash('success', 'Пользователь удален!');

return response()->json(User::destroy($id), 200);

}

private function createBalanceHistory(User $user, $newBalance, $comment = '') {

if($user->balance != $newBalance) {

$balanceHistory = new BalanceHistory();

$balanceHistory->user\_id = $user->id;

$balanceHistory->current\_balance = $newBalance;

$balanceHistory->transaction\_cost = abs($user->balance - $newBalance);

$balanceHistory->type = $user->balance - $newBalance > 0 ? 'Debit' : 'Credit';

$balanceHistory->comment = $comment;

$balanceHistory->save();

}

}

}

<?php

namespace App\Http\Controllers;

use App\Order;

use App\Product;

use App\Shipment;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

class ExportController extends Controller

{

public function exportProducts(){

if(Auth::user()->role == 'Admin') {

$entities = Product::orderBy('id', 'desc')->get();

} else {

$entities = Product::where('user\_id', Auth::id())->orderBy('id', 'desc')->get();

}

header('Content-type: application/txt');

header('Content-Disposition: attachment; filename="products.csv"');

foreach ($entities as $entity){

echo iconv("UTF-8", "Windows-1251","{$entity->name}; {$entity->brand};{$entity->upc};{$entity->sku};\r\n");

}

}

public function exportOrders(){

if(Auth::user()->role == 'Admin') {

$entities = Order::orderBy('id', 'desc')->get();

} else {

$entities = Order::where('user\_id', Auth::id())->orderBy('id', 'desc')->get();

}

header('Content-type: application/txt;');

header('Content-Disposition: attachment; filename="orders.csv"');

foreach ($entities as $entity){

$products = '';

foreach ($entity->products as $product){

$products .= "{$product->name}:{$product->pivot->quantity}:";

if($product->pivot->price) {

$products .= "{$product->pivot->price}:";

}

if($product->pivot->description) {

$products .= "{$product->pivot->description}:";

}

$products = substr($products,0,-1) . '|';

}

if ($products) {

$products = substr($products,0,-1);

}

echo iconv("UTF-8", "Windows-1251",

"{$entity->customer}; {$entity->comment};{$entity->company\_name};{$entity->shipping\_company};"

."{$entity->tracking\_number}; {$entity->packing\_selection};{$entity->address};{$entity->city};"

."{$entity->zip\_postal\_code}; {$entity->state\_region};{$entity->country};{$entity->phone};$products;\r\n"

);

}

}

public function exportShipments(){

if(Auth::user()->role == 'Admin') {

$entities = Shipment::orderBy('id', 'desc')->get();

} else {

$entities = Shipment::where('user\_id', Auth::id())->orderBy('id', 'desc')->get();

}

header('Content-type: application/txt;');

header('Content-Disposition: attachment; filename="shipments.csv"');

foreach ($entities as $entity){

$products = '';

foreach ($entity->products as $product){

$products .= "{$product->name}:{$product->pivot->quantity}|";

}

if ($products) {

$products = substr($products,0,-1);

}

echo iconv("UTF-8", "Windows-1251","{$entity->tracking\_number}; {$entity->shipping\_company};{$entity->comment};{$entity->quantity};$products;\r\n");

}

}

}