ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ УЧЕТА ИКОНТРОЛЯ ПРОИЗВОДСТВЕННОЙ ДЕЯТЕЛЬНОСТИ ОПТОВОЙ БАЗЫ ИП БРОВКОВ Г.Е.

Выпускная квалификационная работа

обучающейся по направлению подготовки
02.03.02 Фундаментальная информатика и информационные технологии
очной формы обучения, 4 курса группы 07001301
Курбатовой Ларисы Сергеевны

Научный руководитель
к.т.н., доцент
В.М. Михелев

БЕЛГОРОД 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Двадцать первый век – время информационных технологий, которые в своем уровне развития дошли до того, что компьютеры есть практически у каждого. Благодаря быстрому распространению сети интернет перед людьми открываются все новые и новые возможности. А благодаря новейшим разработкам, услуги провайдеров становятся все более качественными, обеспечивают более высокую скорость работы пользователей.

На сегодняшний день уже нельзя представить без интернета ни одну сферу нашей жизни. Мы можем оплачивать услуги, покупать все необходимое, не выходя из дома. Все это стало толчком к развитию такой ветви информационных технологий как создание web-сайтов.

Люди проводят часы в поисках нужной информации об услугах или чем-либо еще. Часто на интернет-просторах встречается реклама, которая тоже может быть интересна, а рекламодатель, в свою очередь, может привлечь новых клиентов, не затрачивая при этом много ресурсов.

Благодаря этому все современные компании, организации, учреждения имеют в сети интернет собственные сайты, на которых располагается информация об их деятельности, контактах. Такие ресурсы выдают только ту информацию, которая будет интересна пользователю. Любой человек может быстро найти то, что ему нужно. Таким образом, любая организация может обеспечить себе рекламу и привлечь новых людей в свое дело.

Грамотно созданный сайт будет способствовать развитию и продвижению компании, однако сайты используются не только для рекламы, но и для обеспечения возможности ведения бизнеса.

Имея свой специальный электронный ресурс, служащие тратят меньше времени на поиск бумажных документов. Благодаря центральной базе данных, регулярно создаются резервные копии файлов, благодаря чему

исключается возможность того, что документ будет безвозвратно потерян, если его забудут в самолёте, случайно или преднамеренно уничтожат, или же просто сгинет в офисном беспорядке. Совершенно исключается потеря времени на поиски файлов и документов, которых, по какой-то причине, не оказалось на своём месте.

Такое web-приложение позволит сэкономить массу времени по внесению данных, их обработке. Хранение информации в файлах менее затратное по сравнению с электронным хранилищем. Базы данных позволяют производить множество операций с информацией и использовать для любых своих целей. Использование компьютера позволяет сэкономить большое количество средств и времени для получения нужных данных, а также упрощает ведение и доступ к ним. В этом состоит актуальность и значимость данного дипломного проекта.

Выпускная квалификационная работа состоит из следующих частей:

- Первая глава посвящена выбору программного обеспечения и системному анализу предметной области, проектированию web-приложения;
- Во второй главе описана разработка информационного обеспечения (создание БД и реализация бизнес-логики);
- В третьей главе описана разработка программного обеспечения (создание web-приложения);
- В четвертой главе показан пользовательский интерфейс при работе с web-приложением.

ВКР состоит из 66 страниц, содержит 34 рисунка, 1 таблицу, заключение, список используемой литературы и приложение.

ГЛАВА 1. ПРОЕКТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ

Всемирные исследования и опыт российских аналитиков показывают, что деятельность продуктивных рыночных отношений между импортерами и клиентами не представляется возможной без причастности оптовых элементов, обозначающихся в роли предприимчивого коммерческого посредника.

Оптовая коммерция осуществляет посредническую роль между изготовлением товара и его продажей. Процесс оборота продукции часто суммируется из двух стадий оптового и розничного сбыта. Оптовое товарообращение по своей экономической сути демонстрирует собой процесс продажи продукции общепроизводственными или торговыми фирмами другим торговым фирмам или предприятиям для дальнейшей перепродажи потребителям. На оптовых фирмах лежит ответственная задача воплощения в жизнь главной экономической функции торговли — доставки товаров от производителя до потребителя, для того, чтобы удовлетворить спроси получить максимум выгоды.

Оптовая торговля являет собой важнейшую долю системы распределения. Она регламентирует как отраслевые, так и профессиональные торги вследствие скапливания и передвижения товаров. В этом и заключается основная коммерческая деятельность.

Оптовая торговля должна регулировать стоимость, помогать обеспечивать повышение качества продукции для потребителей. Главной деятельностью данного бизнеса является совершенствование способов доставок и поставок товаров, а также освоение сложившейся обстановки рынка, расположение изготовления продукции в нужном разнообразии и количестве для покупателей, организация складов и т.д.

В настоящее время у этого бизнеса есть ряд своих недостатков. Среди них можно отметить неисполнение сроков поставок, нарушение обязательств по договорам, поставкам, хранению, качеству товаров.

Ценовая политика на данном рынке имеет прямое влияние на эффективность работы хозяйственного комплекса государства в целом. От этого же и зависит равновесие внутри рынка. Опт содействует эффективной работе коммерческих связей. Этот бизнес является более выгодным по сравнению с розничным, благодаря большому обороту денежных ресурсов, различных связей, четким знаниям в этой области.

Настоящая разновидность бизнеса играет важную роль в системе экономических контактов между регионами государства, видами производства, изготовителями продукции. Широкие поставки динамично воздействуют на количество и разновидность выпускаемых товаров, запрашивают смену или приостановление производства товаров, не пользующихся спросом, а на товар, удовлетворяющий потребности клиентов добиваются улучшения качества. По причине этого есть возможность возврата продукции неудовлетворительного качества, таким образом, крупнопромышленные организации повышают качество и улучшают продукцию, производимую ими в дальнейшем.

Фирмы, продвигающие товары в оптовых масштабах, динамично участвуют в обеспечении сбыта продукции покупателям и корректируют разнообразие ассортимента в различных торговых точках, таким образом, они влияют на неизменное присутствие на прилавках тех или иных товаров, имеющихся в наличии. Эти предприятия участвуют в различных рекламных акциях и процедурах, а еще налаживают поставки избыточных товаров со складов магазинов в другие части страны.

1.1 Анализ деятельности предприятия

Торговая фирма ИП Бровков Г.Е. занимается оптовой продажей различных кондитерских изделий.

Специфика торговли продуктами питания состоит в том, что товары с течением времени ухудшают свои потребительские качества и могут представлять потенциальную опасность для человека. В связи с этим к организациям, торгующим продуктами питания, государство предъявляет особые требования, которые не могут не сказываться, в том числе, и на документообороте таких фирм.

Организации, торгующие продуктами питания, должны соблюдать еще массу определенных требований, предъявляемых государством.

Все эти требования, предъявляемые к организациям - продавцам продуктов питания, составляют особенности торговли продовольствием, что в свою очередь сказывается и на документообороте таких фирм, которые традиционно входят в разряд организаций, обладающих значительным объемом документации.

В общепринятом значении под документооборотом понимается движение первичных учетных документов от момента их получения или создания самой организацией до исполнения и передачи в архив на хранение. Заметим, что правила своего документооборота торговая фирма определяет самостоятельно, руководствуясь при этом размерами самой фирмы, ее внутренней структурой, а также интенсивностью хозяйственных операций.

1.2 Выбор инструментальных средств для создания программного обеспечения

Существует множество средств для создания web-приложений, рассмотрим некоторые из них.

HTML–Язык разметки гипертекста (Hypertext Markup Language), или, как его чаще называют, HTML, — это компьютерный язык, лежащий в основе World Wide Web (Всемирной Паутины). Благодаря языку HTML любой текст можно разметить, преобразовав его в гипертекст с последующей публикацией в Web.

Язык HTML имеет собственный набор символов, с помощью которых Web-браузеры отображают страницу. Эти символы, называемые дескрипторами, включают в себя элементы, необходимые для создания гиперссылок.

Одной из отличительных особенностей HTML-документов является то, что сам документ содержит только текст, а все остальные объекты встраиваются в документ в момент его отображения Браузером с помощью специальных тэгов и хранятся отдельно. При сохранении HTML-файла в месте размещения документа создается папка, в которую помещаются сопутствующие ему графические элементы оформления[2].

PHP– В первую очередь PHP используется для создания скриптов, работающих на стороне сервера, для этого его, собственно, и придумали. PHP способен решать те же задачи, что и любые другие CGI-скрипты, в том числе обрабатывать данные html-форм, динамически генерировать html страницы и тому подобное. Но есть и другие области, где может использоваться PHP.

Вторая область – это создание скриптов, выполняющихся в командной строке. То есть с помощью PHP можно создавать такие скрипты, которые будут исполняться, вне зависимости от web-сервера и браузера, на конкретной машине.

Ajax – расшифровывается как Asynchronous Javascript And XML (Асинхронные Javascript и XML) и технологией в строгом смысле слова не является. Если в стандартном web-приложении обработкой всей информации занимается сервер, тогда как браузер отвечает только за взаимодействие с пользователем, передачу запросов и вывод поступившего HTML, то в Ajax-

приложении между пользователем и сервером появляется еще один посредник - движок Ajax. Он определяет, какие запросы можно обработать "на месте", а за какими необходимо обращаться на сервер.

Поведение сервера тоже изменилось. Если раньше на каждый запрос сервер выдавал новую страницу, то теперь он отсылает лишь те данные, которые нужны клиенту, а HTML из них прямо в браузере формирует движок Ajax.

Асинхронность проявляется в том, что далеко не каждый клик пользователя доходит до сервера, причем обратное тоже справедливо - далеко не каждая реакция сервера обусловлена запросом пользователя. Большую часть запросов формирует движок Ajax, причем его можно написать так, что он будет загружать информацию, предугадывая действия пользователя.

Где стоит использовать Ajax:

- Формы. Они очень медленны. Если асинхронно передавать данные, страница не перезагружается;

- Навигация в виде "дерева". Простая топология намного удобнее, но если уж до этого дошло, лучше использовать Ajax;

- Голосования. Пользователю будет приятней оставить свой голос за несколько секунд, чем за 30-40;

- Фильтры. Часто на сайтах делают сортировку по дате, по имени. Ajax это будет значительно удобнее.

JavaScript – язык программирования для создания интерактивных HTML-документов. Это объектно-ориентированный язык разработки встраиваемых приложений, выполняющихся как на стороне клиента, так и на стороне сервера. Синтаксис языка очень похож на синтаксис Java – поэтому его называют – Java-подобным.

Основные области применения JavaScript делятся на следующие категории:

- динамическое создание документа с помощью сценария;

- оперативная проверка достоверности заполняемых пользователем полей форм HTML до передачи их на сервер;

- создание динамических HTML-страниц совместно с каскадными таблицами стилей и объектной моделью документа;

- взаимодействие с пользователем при решении "локальных" задач, решаемых приложением JavaScript, встроенном в HTML-страницу.

Среди всех рассмотренных инструментальных средств, PHP является наиболее подходящим и удобным для создания web-приложения. Рассмотрим его более подробно.

PHP (Hypertext Pre Processor) – один из самых популярных инструментов веб-программирования на стороне сервера. Работа PHP в самом простом варианте сводится к обработке http запроса клиента. Обработка запроса, в свою очередь, заключается в программном формировании гипертекста в соответствии с параметрами запроса, после чего полученная разметка возвращается клиенту. Когда клиент (интернет браузер) запрашивает обычную статическую интернет страницу (чаще всего с расширением html), сервер в качестве ответа возвращает ему содержимое этой страницу без изменений “как есть”. Если запрашивается php страница, то в процессе обработки запроса содержимое указанной страницы сначала обрабатывается интерпретатором PHP, и только потом результат этой обработки отправляется клиенту.

Другими словами, PHP – это препроцессор гипертекста, что и отражено в его названии. Препроцессор потому что окончательной обработке гипертекст подвергается уже на стороне клиента, результат которой мы видим в окне браузера (процессором гипертекста является уже сам браузер). Можно сказать, что PHP – это генератор гипертекста, поскольку в большинстве случаев его работа – это программная генерация HTML разметки по содержимому базы данных или по любой другой структурированной информации, размещенной на сервере. Аббревиатура выглядит, как PHP, а не как, к примеру, HPP или иначе, поскольку

первоначально расшифровывалась как Personal Home Page Tools – инструментарий для создания персональных интернет страниц. Таким образом, первый вариант расшифровки PHP отражал его назначение, а нынешний – принцип работы.

PHP – это язык программирования, который поддерживает практически все основные конструкции процедурного программирования: переменные, условные операторы, циклы, функции и т.д. PHP – это объектно-ориентированный язык программирования – он поддерживает классы и объекты, а также привычное наследование на уровне классов. PHP – это язык веб-программирования, поскольку он в первую очередь создан для разработки динамических интернет сайтов и поэтому содержит большое количество готовых решений, применяемых в этой сфере, таких как:

- обработка и извлечение параметров http запросов GET и POST;
- формирование и отправка http заголовков;
- инфраструктура для хранения данных сеанса;
- программные сервисы для работы с cookies;

1.3 Выбор СУБД

Способы организации хранения информации связаны с ее поиском-операцией, предполагающей извлечение хранимой информации.

Хранение и поиск информации являются не только операциями над ней, но и предполагают использование методов осуществления этих операций. Информация запоминается так, чтобы ее можно было отыскать для дальнейшего использования. Возможность поиска закладывается во время организации процесса запоминания. Для этого используют методы маркирования запоминаемой информации, обеспечивающие поиск и последующий доступ к ней. Эти методы применяются для работы с файлами, базами данных и т.д.

Программное обеспечение, осуществляющее операции над базами данных, получило название СУБД– система управления базами данных.

С понятием базы данных тесно связано понятие системы управления базой данных. Это комплекс программных средств, предназначенных для создания структуры новой базы, наполнение ее содержимым, редактирование содержимого и визуализации информации. Под визуализацией информации базы понимается отбор отображаемых данных в соответствии с заданным критерием, их упорядочение, оформление и последующая выдача на устройства вывода или передачи по каналам связи.

В мире существует множество систем управления базами данных. Несмотря на то, что они могут по-разному работать с разными объектами и предоставляют пользователю различные функции и средства, большинство СУБД опираются на единый устоявшийся комплекс основных понятий.

Важнейшее требование к современным СУБД - межоперабельность (или интероперабельность). Это качество можно трактовать как открытость системы, позволяющая встраивать ее как компонент в сложную разнородную распределенную среду. Оно достигается как за счет использования интерфейсов, соответствующих международным, национальным и промышленным стандартам, так и за счет специальных решений.

Базы данных могут содержать различные объекты. Основными объектами любой базы данных являются ее таблицы. Простейшая база данных имеет хотя бы одну таблицу. Соответственно, структура простейшей базы данных тождественно равна структуре ее таблицы.

Структуру двумерной таблицы образуют столбцы и строки. Их аналогами в простейшей базе данных являются поля и записи. Если записей в таблице пока нет, значит, ее структура образована только набором полей. Изменив состав полей базовой таблицы (или их свойства), мы изменяем структуру базы данных и, соответственно, получаем новую базу данных.

СУБД должна обеспечивать реляционную модель работы с данными. Сама модель подразумевает определенный тип связи между сущностями из

разных таблиц. Чтобы хранить и работать с данными, такой тип СУБД должен иметь определенную структуру (таблицы). В таблицах каждый столбец может содержать данные разного типа. Каждая запись состоит из множества атрибутов (столбцов) и имеет уникальный ключ, хранящейся в той же таблице - все эти данные взаимосвязаны между собой, как описано в реляционной модели.

Отношения в базах данных можно рассматривать как математическое множество, содержащее в себе число атрибутов, которые суммарно представляют собой базу данных и информацию, хранящуюся в ней. При создании структуры таблицы каждое поле записи должно иметь заранее описанный тип (например: строка, целочисленное значение и т.д.). Все СУБД имеют в своем составе различные типы данных, которые не всегда взаимозаменяемы. При работе с СУБД всегда приходится сталкиваться с подобными ограничениями.

PostgreSQL является самой профессиональной из всех СУБД. Она свободно распространяемая и максимально соответствует стандартам SQL.

От других СУБД, таких как MySQL и SQLite, PostgreSQL отличается поддержкой востребованного объектно-ориентированного и/или реляционного подхода к базам данных. Например, полная поддержка надежных транзакций, т.е. атомарность, последовательность, изоляционность, прочность. Благодаря мощным технологиям Postgre очень производительна. Параллельность достигнута не за счет блокировки операций чтения, а благодаря реализации управления многовариантным параллелизмом (MVCC), что также обеспечивает соответствие ACID. PostgreSQL очень легко расширять своими процедурами, которые называются хранимые процедуры. Эти функции упрощают использование постоянно повторяемых операций.

Хотя PostgreSQL и не может похвастаться большой популярностью в отличие от MySQL, существует довольно большое число приложений, облегчающих работу с PostgreSQL, несмотря на всю мощность функционала.

Сейчас довольно легко установить эту СУБД, используя стандартные менеджеры пакетов операционных систем.

Вопросы обеспечения надежности особенно важны в приложениях уровня предприятия при работе с критически важными данными. СУБД PostgreSQL дает возможность настраивать горячее резервирование и восстановление на заданный момент времени в прошлом, а также поддерживает различные виды репликации (синхронную, асинхронную и каскадную). Все это позволяет строить отказоустойчивые системы с «теплым» или «горячим» резервированием, а также создавать надежные кластерные решения[1].

Особое внимание в PostgreSQL уделено обеспечению безопасности — СУБД предоставляет различные методы аутентификации: по паролю в открытом или зашифрованном виде; по внешней аутентификации. При управлении пользователями и доступом к объектам базы данных имеется возможность выделять отдельных пользователей и роли, которые могут быть вложенными; доступ к объектам базы может осуществляться как напрямую пользователями, так и косвенно через роли.

СУБД PostgreSQL обеспечивает полную поддержку свойств ACID и гарантирует изоляцию транзакций благодаря механизму многоверсионного управления одновременным доступом — транзакции на чтение никогда не блокируют транзакции на запись, и наоборот. Это обеспечивает полную изоляцию транзакций, гарантирующую, что результат работы одновременных транзакций будет такой же, как и при их последовательном исполнении.

Расширяемость — одно из фундаментальных свойств системы, лежащее в основе ее архитектуры. Пользователи могут самостоятельно добавлять функции, типы данных, операторы для работы с новыми типами, использовать индексные методы доступа. Подключение к внешним источникам (Foreign Data Wrappers) осуществляется через интерфейсы практически ко всем СУБД, а загружаемые расширения позволяют,

например, поддерживать геоинформационные данные PostGIS, осуществлять нечеткий поиск с помощью триграмм, работу с массивами и др.

В СУБД PostgreSQL учитываются особенности архитектуры многоядерных процессоров, поэтому производительность растет почти линейно с увеличением количества ядер, но тем не менее в системе имеются некоторые пределы.

В PostgreSQL используется планировщик запросов, позволяющий оптимизировать сложные запросы. Способность планировщика исключать просмотр дочерних таблиц на основе анализа условия запроса и имеющихся ограничений целостности (constraintexclusion) позволяет реализовать в PostgreSQL секционирование (partitioning), что особенно актуально для крупных хранилищ данных.

1.4 Выбор фреймворка

В процессе разработки сайта может измениться многое, если не всё – от дизайна до бизнес-логики. Масштабные перемены могут ожидать проект и в будущем. Возможно, после запуска потребуется добавить на сайт различные модули (например, новый раздел с материалами, личный кабинет пользователя или почтовую рассылку). Если в коде с самого начала наблюдается сильная связанность (зависимость одних функций от других), то время на разработку увеличивается, а число ошибок возрастает. Конечно, можно решать возникающие проблемы быстрыми заплатами.

Описанное относится в основном к крупным, сложным проектам. Но беда может случиться и при работе над небольшим сайтом. Поэтому крайне желательно, чтобы любой проект с самого начала разработки имел гибкую и легко расширяемую структуру.

Чтобы уберечь себя от возможных проблем, можно изначально создавать программу на жёстком каркасе, который позволяет подключать к себе дополнительные модули. По сути, мы будем писать свой код только в

тех местах, где это разрешено создателями каркаса (в так называемых «точках расширения»). Так мы не поломаем базовые части системы и сможем сконцентрироваться на своих текущих задачах.

В мире программирования описанный каркас называют фреймворком. Фреймворк – не обычная программная библиотека. Если библиотека – это просто набор функций, которые не влияют на архитектуру программы, то фреймворк сам, по сути, является архитектурой. Каркас гарантирует стандартную структуру программ и их поведение по умолчанию.

Рассмотрим несколько популярных фреймворков и выберем один из них.

ZendFramework 2 – фреймворк с открытым исходным кодом, который используется для разработки веб-приложений и сервисов на PHP 5.3. Он использует на 100% объектно-ориентированный код и большинство новых функций PHP 5.3: пространства имен, позднее статическое связывание, замыкание. Zend– надежное решение с множеством вариантов конфигурации. Обычно его не рекомендуют использовать для небольших приложений, а вот для крупных проектов это самое то.

CodeIgniter – PHP-фреймворк с десятилетним стажем и очень простым процессом установки, требующим минимальной конфигурации, так что начать будет просто. Хороший выбор, если есть риск конфликта разных PHP-версий: он идеально работает практически на всех платформах виртуального и выделенного хостинга.

Yii2 – это современный и широконаправленный PHP фреймворк, для разработки веб и консольных приложений. Он призывает к чистому написанию кода, без лишних связанностей – принципу DRY (don’t repeat yourself), направленному на снижение повторения кода, событийно-ориентированное программирование, когда выполнение подпрограммы определяется исходя из события, запущенного пользователем, соглашение по конфигурации, по которому если класс соответствует соглашению наименованию, тогда он не нуждается в дополнительной конфигурации. Для

фреймворка уже разработано много библиотек, виджетов и расширений как основными, так и сторонними разработчиками. Это все позволяет быстро разрабатывать качественные приложения.

Yii — это фреймворк для веб-программирования общего назначения, который может быть использован для разработки практически любых веб-приложений. Благодаря своей легковесности и наличию продвинутых средств кэширования, Yii особенно подходит для разработки приложений с большим потоком трафика, таких как порталы, форумы, системы управления контентом (CMS), системы электронной коммерции [3].

Превосходство Yii над другими фреймворками заключается в эффективности, широких возможностях и качественной документации. Yii изначально спроектирован очень тщательно для соответствия всем требованиям при разработке серьёзных веб-приложений. Yii не является ни побочным продуктом какого-либо проекта, ни сборкой сторонних решений. Он является результатом большого опыта авторов в разработке веб-приложений, а также их исследований наиболее популярных веб-фреймворков и приложений.

Таким образом, среди всех рассмотренных фреймворков, Yii больше всего подходит для создания web-приложения. Рассмотрим его более подробно.

Фреймворк базируется на паттерне проектирования MVC.

Model-view-controller — схема использования нескольких шаблонов 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. Данная схема проектирования часто используется для построения архитектурного каркаса, когда переходят от теории к реализации в конкретной предметной области.

Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счёт

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. В частности, 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.

Концепция MVC позволяет разделить данные (модель), представление и обработку действий (производимую контроллером) пользователя на три отдельных компонента:

- Модель
  - Предоставляет знания: данные и методы работы с этими данными;
  - Реагирует на запросы, изменяя своё состояние;
  - Не содержит информации, как эти знания можно визуализировать;
- Представление, вид – отвечает за отображение информации (визуализацию). Часто в качестве представления выступает форма (окно) с графическими элементами;

- Контроллер – обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Важно отметить, что как представление, так и контроллер зависят от модели; однако модель (активная) 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.

Для реализации схемы «Model-View-Controller» используется достаточно большое число шаблонов проектирования (в зависимости от сложности архитектурного решения), основные из которых — «наблюдатель», «стратегия», «компоновщик»:

- Наиболее типичная реализация — в которой вид отделён от модели путём установления между ними протокола взаимодействия, использующего «аппарат событий» (обозначение «событиями» определённых ситуаций, возникающих в ходе выполнения программы, — и рассылка уведомлений о них всем тем, кто подписался на получение): при каждом особом изменении внутренних данных в модели (обозначенном как «событие»), она оповещает о нём те зависящие от неё представления, которые подписаны на получение такого оповещения — и представление обновляется. Так используется шаблон «наблюдатель»;
- При обработке реакции пользователя — представление выбирает, в зависимости от реакции, нужный контроллер, который обеспечит ту или иную связь с моделью. Для этого используется шаблон «стратегия», или вместо этого может быть модификация с использованием шаблона «команда»;
- Для возможности однотипного обращения с подобъектами сложносоставного иерархического вида — может использоваться шаблон «компоновщик». Кроме того, могут использоваться и другие шаблоны

проектирования — например, «фабричный метод», который позволит задать по умолчанию тип контроллера для соответствующего вида.

1.5 Постановка задачи

Повышение уровня коммерческой работы требует постоянного совершенствования ее технологии, особенно использования новой техники управления.

Весьма актуальна задача компьютеризации процессов управления коммерческой работы по оптовым закупкам и оптовой продаже товаров.

Постоянный учет и контроль оптовых закупок товаров, характеризующихся большим количеством поставщиков, десятками наименований товаров ассортимента, возможен лишь с помощью компьютеров. Ручная, карточная форма учета поставок, осуществляемая товароведами, трудоемка и не обеспечивает быстрого и точного их учета по всей совокупности разновидностей ассортимента от большого количества поставщиков и по частным срокам поступления. Такая система учета выполнения договоров в групповом ассортименте, как правило, не обеспечивает принятия оперативных мер воздействия на поставщиков, допускающих нарушения обязательств по поставкам товаров в развернутом ассортименте, приводит к срывам поставок и перебоям в поступлении товаров. Для этих целей необходима организация оперативной обработки коммерческой информации и управления коммерческими процессами. Это обеспечивает автоматизацию учета поставки и реализации товаров по внутригрупповому ассортименту, высвобождает время для реальной коммерческой работы с поставщиками и покупателями, повышает производительность труда коммерческого аппарата.

Таким образом, в дипломной работе необходимо разработать web-приложение об организационной структуре, в котором должны быть описаны все организационные звенья с указанием направления их деятельности, функции, которые они выполняют, показано распределение видов продуктов,

затрат и прибыли, закупки и продажи товаров. Также должна быть обеспечена возможность ведения всей учетной информации, загрузки и хранения документов предприятия.

ГЛАВА 2. РАЗРАБОТКА ИНФОРМАЦИОННОГО ОБЕСПЕЧЕНИЯ WEB-ПРИЛОЖЕНИЯ

При создании любого web-приложения критическим аспектом является его архитектура, где она представляет собой концептуальное видение структуры будущих функциональных процессов и технологий на системном уровне и во взаимосвязи. Обычно Web-приложения организаций проектируются как композиция компонентов, взаимодействующих на высоком уровне, которые сами могут быть системами. Архитектура Web-приложения организации делает понимание системы легче, определяя ее функциональность и структуру способом, который раскрывает проектировочные решения и позволяет обозревателю задавать вопросы об удовлетворении проектных требований, распределении функциональности и реализации компонентов.

При проектировании современных Web-приложений организаций их архитектура должна разрабатываться с учетом многих заинтересованных сторон, оно должно быть понятным пользователям, давать возможность разработчикам сделать план и графики системы, позволять определять ключевые интерфейсы, функции и технологии, а также позволять оценить график и бюджет исполнения проекта. При этом от архитекторов современных Web-приложений требуется ответственность за создание удовлетворительной и осуществимой концепции приложения на самом раннем этапе его разработки, поддержки цельности этой концепции на протяжении разработки и определения пригодности результирующей системы для использования клиентом. Разработка архитектуры Web-приложения – это процесс описания архитектур информационных систем в достаточной деталировке, чтобы сделать их более полезными для разработки.

Программные архитектуры, используемые в настоящее время:

- файл-серверная;
- клиент-серверная;
- многоуровневая.

Файл-сервер. Эта архитектура централизованных баз данных с сетевым доступом предполагает назначение одного из компьютеров сети в качестве выделенного сервера, на котором будут храниться файлы централизованной базы данных. В соответствии с запросами пользователей файлы с файл-сервера передаются на рабочие станции пользователей, где и осуществляется основная часть обработки данных. Центральный сервер выполняет в основном только роль хранилища файлов, не участвуя в обработке самих данных. После завершения работы пользователи копируют файлы с обработанными данными обратно на сервер, откуда их могут взять и обработать другие пользователи. Такая организация ведения данных обладает рядом недостатков, например, при одновременном обращении множества пользователей к одним и тем же данным производительность работы резко падает, так как необходимо дождаться, пока пользователь, работающий с данными, завершит работу. В противном случае возможно затирание исправлений, сделанных одними пользователями, изменениями, внесенными другими пользователями.

Клиент-сервер. В основе этой концепции лежит идея о том, что помимо хранения файлов базы данных, центральный сервер должен выполнять основную часть обработки данных. Пользователи обращаются к центральному серверу с помощью специального языка структурированных запросов (SQL, Structured Query Language), на котором описывается список задач, выполняемых сервером. Запросы пользователей принимаются сервером и порождают в нем процессы обработки данных. В ответ пользователь получает уже обработанный набор данных. Между клиентом и сервером передается не весь набор данных, как это происходит в технологии файл-сервер, а только данные, которые необходимы клиенту. Запрос пользователя длиной всего в несколько строк способен породить процесс

обработки данных, затрагивающий множество таблиц и миллионы строк. В ответ клиент может получить лишь несколько чисел.

Технология клиент-сервер позволяет избежать передачи по сети огромных объемов информации, переложив всю обработку данных на центральный сервер. Кроме того, рассматриваемый подход позволяет избежать конфликтов изменений одних и тех же данных множеством пользователей, которые характерны для технологии файл-сервер. Технология клиент-сервер реализует согласованное изменение данных множеством клиентов, обеспечивая автоматическое соблюдение целостности данных. Эти и некоторые другие преимущества сделали технологию клиент-сервер очень популярной. К недостаткам этой технологии можно отнести высокие требования к производительности центрального сервера. Чем больше клиентов обращается к серверу, и чем больше объем обрабатываемых данных, тем более мощным должен быть центральный сервер.

2.1 Логическое проектирование БД

Существует три основных типа логических моделей данных на основе записей: иерархическая, сетевая и реляционная модели данных. В сетевой модели данные представлены в виде коллекции записей, а связи - в виде наборов. В отличие от реляционной модели, связи здесь явным образом моделируются наборами, которые реализуются с помощью указателей. Сетевую модель можно представить как граф с записями в виде узлов графа и наборами в виде его ребер.

Иерархическая модель является ограниченным подтипом сетевой модели. В ней данные также представлены как коллекции записей, а связи - как наборы. Однако в иерархической модели узел может иметь только одного родителя. Иерархическая модель может быть представлена как древовидный граф с записями в виде узлов (которые также называются сегментами) и множествами в виде ребер.

Репрезентативные (или реализационные) модели содержат подробности представления данных в файлах.

Основными понятиями ER-диаграммы **(**Entity-Relationship**,** диаграммы сущность-связь**)** являются сущность, связь и атрибут.

Сущность — это реальный или виртуальный объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению. Если не вдаваться в подробности, то можно считать, что сущности соответствуют таблицам реляционной модели.

Под информационным объектом понимается некоторая сущность фрагмента действительности, например, организация, документ, сотрудник, место, событие и т. д.

Связи выступают в модели в качестве средства, с помощью которого представляются отношения между сущностями, имеющими место в предметной области.

В предметной области выделяются различные типы объектов, представляемые в информационной системе в каждый момент времени конечным набором экземпляров данного типа. Каждый тип объекта включает (идентифицируется) присущий ему набор атрибутов (свойств, характерных признаков, параметров).

Связь — это ассоциация между сущностями, включают по одной сущности из каждого участвующего в связи типа сущности.

Атрибут представляет логически неделимый элемент структуры информации, характеризующийся множеством атомарных значений.

Один или некоторая группа атрибутов объекта данного типа могут исполнять роль ключевого атрибута, по которому различаются конкретные экземпляры объектов (объект «Лицо» - ключ совокупность атрибутов «Фамилия», «Имя», «Отчество» или один атрибут «Номер паспорта»).

Различные типы объектов и различные экземпляры одного типа объекта могут быть охвачены определенными отношениями, которые в рамках ER-модели выражаются связями.

Таким образом, логическая модель – графическое представление структуры базы данных с учетом принимаемой модели данных (иерархической, сетевой, реляционной и т.д.), независимое от конечной реализации базы данных и аппаратной платформы.

Иными словами, она показывает, что именно хранится в базе данных: объекты предметной области, их атрибуты и связи между ними.

Этапы логического проектирования:

1. Преобразование локальной концептуальной модели данных в локальную логическую модель.

2. Определение набора отношений исходя из структуры локальной логической модели данных.

3. Проверка модели с помощью правил нормализации.

4. Проверка модели в отношении транзакций пользователей.

5. Создание диаграммы сущность-связь.

6. Определение требований поддержки целостности данных. (Обязательные данные, ограничения для доменов атрибутов, целостность сущностей (ключ не может быть NULL), требования данного предприятия (бизнес-правила)).

7. Обсуждение разработанных локальных логических моделей данных с конечными пользователями.

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но без привязки к конкретной СУБД.

Таким образом, на основе системного анализа предметной области, была создана логическая модель базы данных, содержащая информацию обо всех звеньях предприятия, его деятельности в плане закупок и продаж,

прибыли и расходов, личных данных поставщиков и постоянных клиентов, а также сотрудников.

Логическая модель представлена на рис. 2.1:

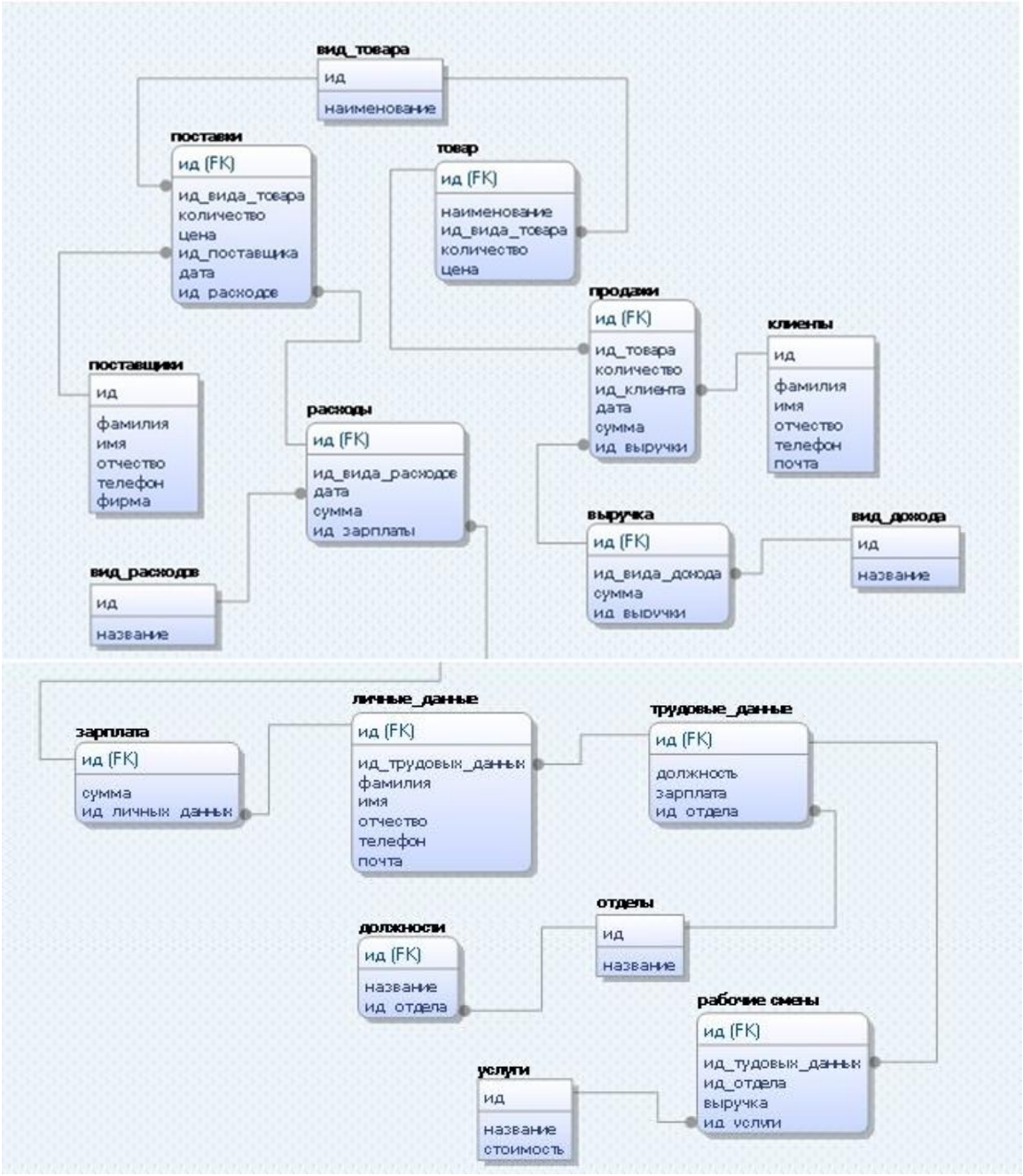


Рис.2.1. Логическая ER-модель БД

На основании логической модели строится физическая модель.

2.2 Физическое проектирование БД

Физическое проектирование базы данных – процесс подготовки описания реализации базы данных на вторичных запоминающих устройствах; на этом этапе рассматриваются основные отношения, организация файлов и индексов, предназначенных для обеспечения эффективного доступа к данным, а также все связанные с этим ограничения целостности и средства защиты.

Физическое проектирование является последним этапом создания проекта базы данных, при выполнении которого проектировщик принимает решения о способах реализации разрабатываемой базы данных. Во время предыдущего этапа проектирования была определена логическая структура базы данных. Однако физическое проектирование неразрывно связано с конкретной СУБД. Между логическим и физическим проектированием существует постоянная обратная связь, так как решения, принимаемые на этапе физического проектирования с целью повышения производительности системы, способны повлиять на структуру логической модели данных.

Как правило, основной целью физического проектирования базы данных является описание способа физической реализации логического проекта базы данных.

Проектирование базы данных — это итерационный процесс, который имеет свое начало, но не имеет конца и состоит из бесконечного ряда уточнений. Как только проектировщик приходит к пониманию работы предприятия и смысла обрабатываемых данных, а также выражает это понимание средствами выбранной модели данных, приобретенные знания могут показать, что требуется уточнение и в других частях проекта.

Этапы физического проектирования баз данных:

1. Перенос глобальной логической модели данных в среду СУБД.
2. Проектирование основных отношений.
3. Разработка способов получения производных данных.

4. Реализация ограничений предметной области.
5. Проектирование физического представления базы данных.
6. Анализ транзакций.

Физическая модель базы данных содержит все детали, необходимые конкретной СУБД для создания базы: наименования таблиц и столбцов, определения первичных и внешних ключей (см. рис. 2.2).

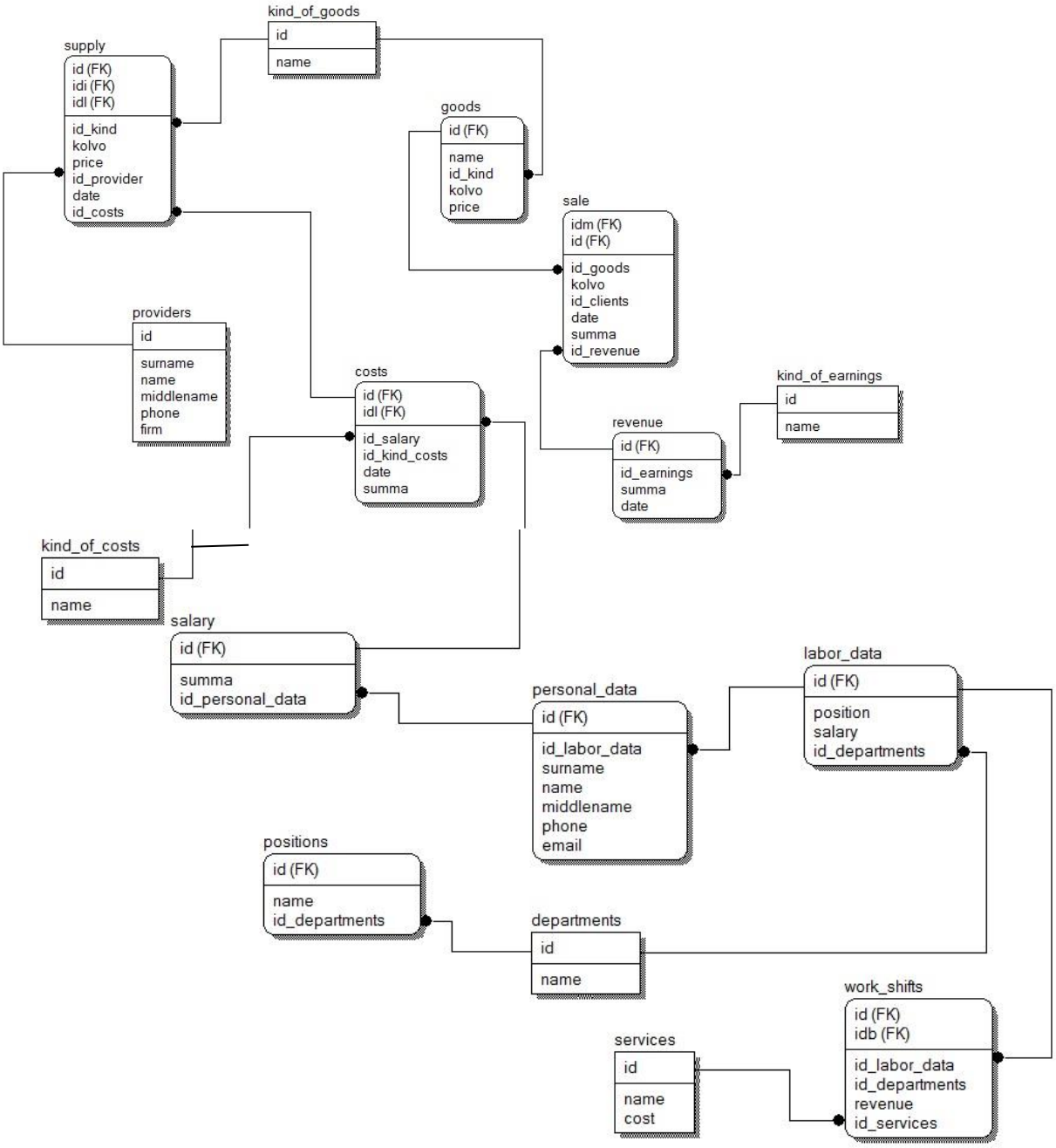


Рис. 2.2. Физическая ER-модель БД

2.3 Создание БД

Для разработки физической модели базы данных было использовано программное средство pgAdmin, предоставляющее дружественный интерфейс для управления базами данных СУБД PostgreSQL.

Для создания новой таблицы необходимо выполнить следующую последовательность действий:

1. Выбрать базу данных.
2. Нажать кнопку «Создать таблицу».
3. После загрузки формы создания таблицы, необходимо ввести имя для создаваемой таблицы.
4. Заполнить все необходимые поля, задать имена полей таблицы, указать типы данных.
5. Создать первичные и внешние ключи.

Создание таблицы представлено на рис. 2.3.

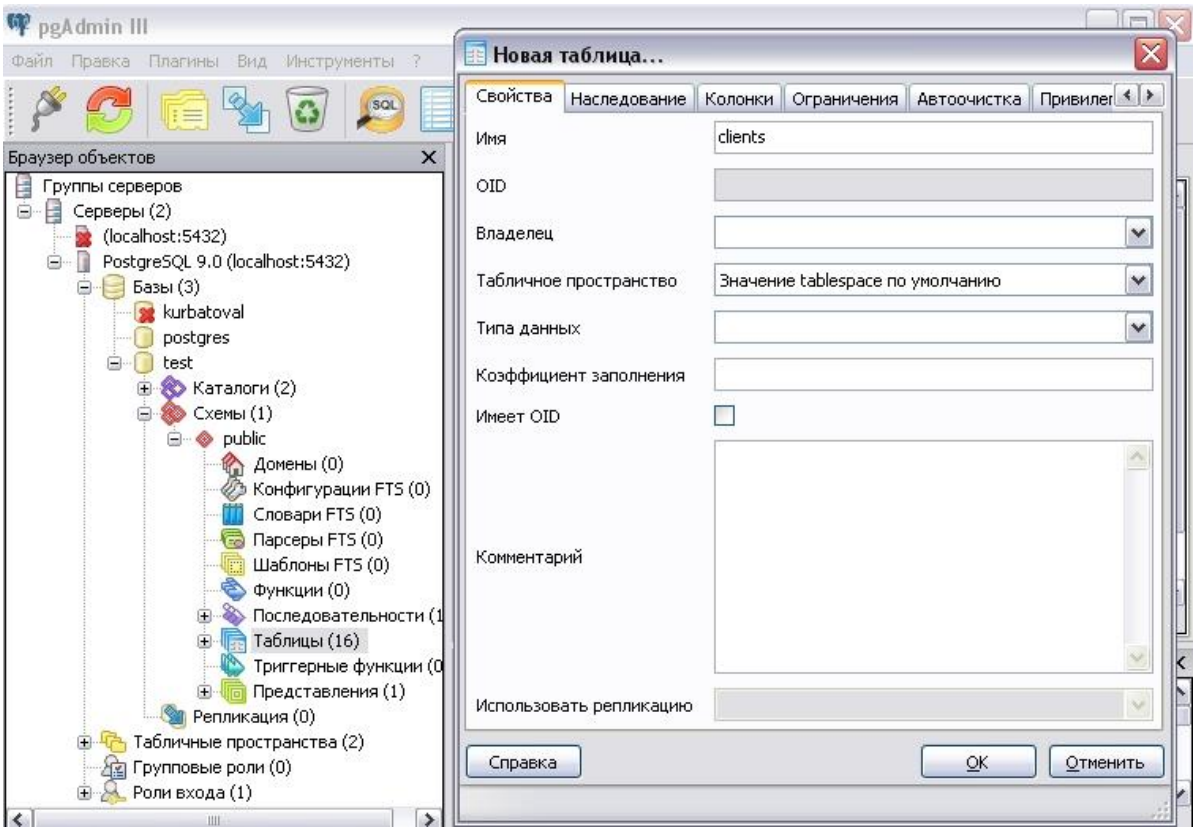


Рис. 2.3. Создание таблицы «Клиенты»

Создание колонок таблицы представлено на рис. 2.4.

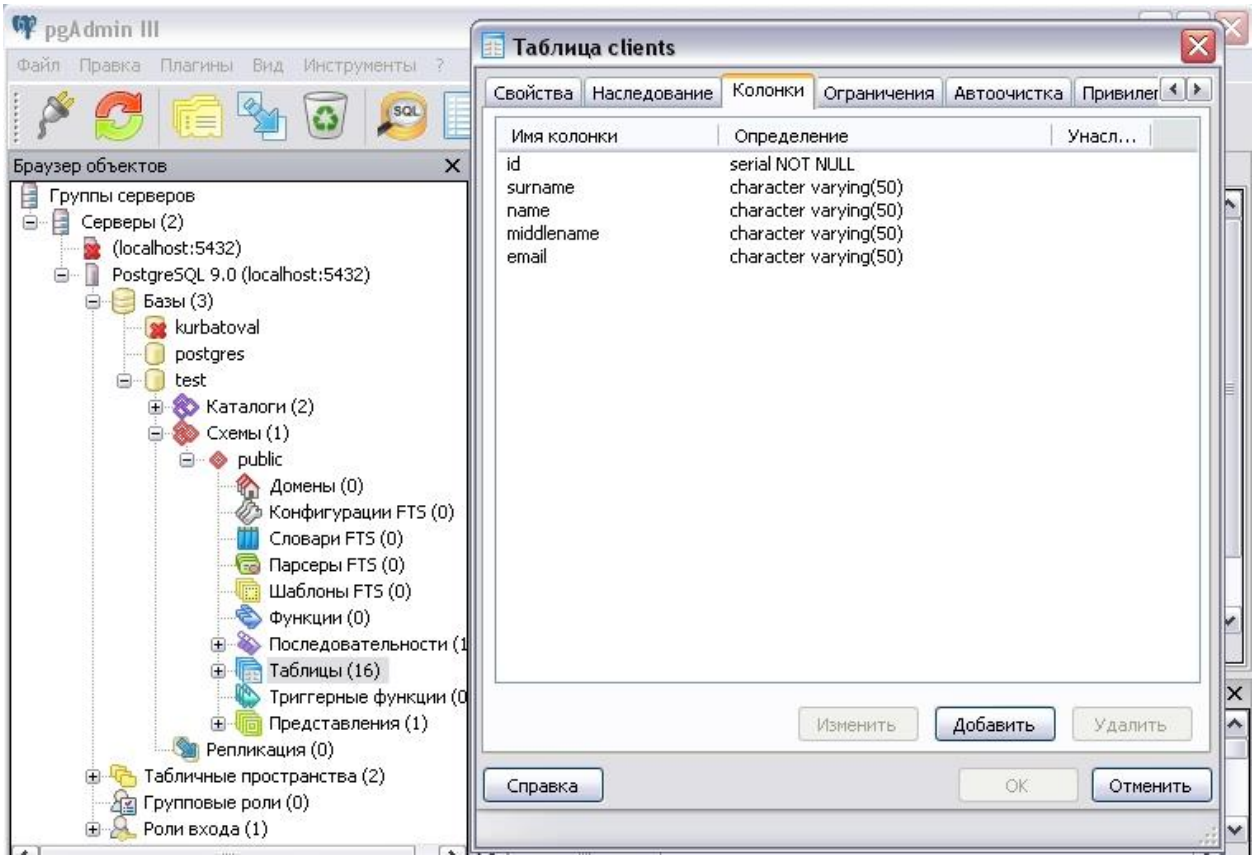


Рис.2.4. Колонки таблицы «Клиенты»

Первичный ключ – это атрибут или группа атрибутов, которые единственным образом идентифицируют каждую строку в таблице.

Для первичного ключа используем тип поля serial. Он является автоинкрементным, также, автоматом выполняет действия по созданию последовательности и привязки ее к полю. Тип serial может быть использован только при создании таблицы. Если при создании таблицы некоторые поля имеют тип serial, то для каждого из этих полей выполняются следующие действия:

1. создается последовательность с именем tblname_colname_seq;
2. тип поля меняется с serial на integer;
3. в модификаторы поля добавляется конструкция notnulldefaultnextval('tblname_colname_seq'::regclass).

Создание первичного ключа представлено на рис. 2.5.

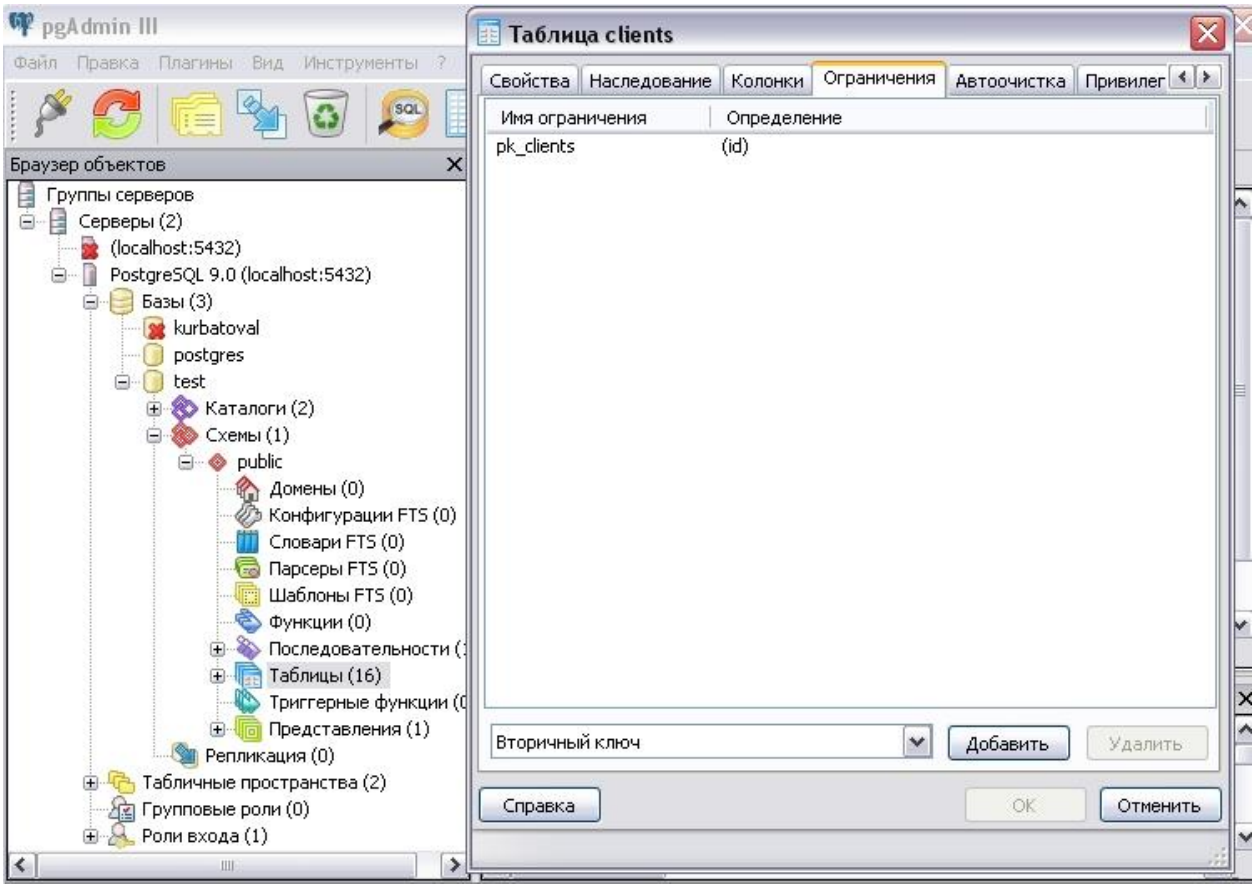


Рис. 2.5. Первичный ключ таблицы «Клиенты»

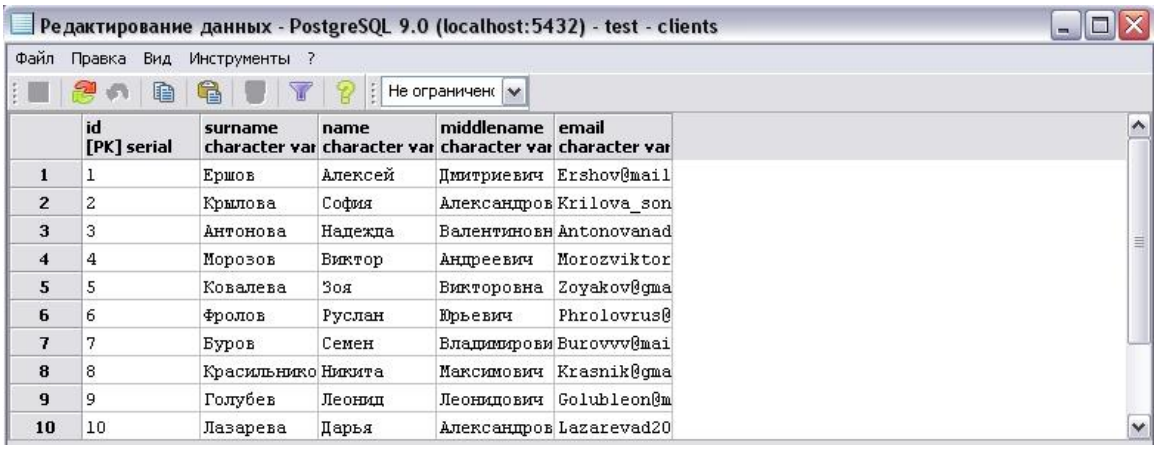
Мы создали таблицу средствами pgAdmin, а так выглядит SQL-код создания нашей таблицы, если бы была необходимость создавать ее вручную через консоль (см. рис. 2.6).

Панель SQL

```
CREATE TABLE clients
(
  id serial NOT NULL,
  surname character varying(50),
  "name" character varying(50),
  middlename character varying(50),
  email character varying(50),
  CONSTRAINT pk_clients PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE clients OWNER TO postgres;
```

Рис.2.6. Код создания таблицы

Далее нажимаем «Просмотр всех строк», после чего открывается форма, где можно не только просмотреть, но и заполнить данные (см. рис. 2.7):



	id [PK] serial	surname character var	name character var	middlename character var	email character var
1	1	Ершов	Алексей	Дмитриевич	Ershov@mail
2	2	Крылова	София	Александров	Krilova_son
3	3	Антонова	Надежда	Валентиновн	Antonovanad
4	4	Морозов	Виктор	Андреевич	Morozviktor
5	5	Ковалева	Зоя	Викторовна	Zoyakov@gma
6	6	Фролов	Руслан	Юрьевич	Phrolovrus@
7	7	Буров	Семен	Владимирови	Burovvv@mai
8	8	Красильнико	Никита	Максимович	Krasnik@gma
9	9	Голубев	Леонид	Леонидович	Golubleon@m
10	10	Лазарева	Дарья	Александров	Lazarevad20

Рис.2.7. Данные таблицы

Таким же образом создадим остальные таблицы.

Между двумя или более таблицами базы данных могут существовать отношения подчиненности. Отношения подчиненности определяют, что для каждой записи главной таблицы (master, называемой еще родительской) может существовать одна или несколько записей в подчиненной таблице (detail, называемой еще дочерней)[7].

Существует три разновидности связей между таблицами базы данных:

- «один-ко-многим»,

-«один-к-одному»,

-«многие-ко-многим».

Отношение «один-ко-многим» имеет место, когда одной записи родительской таблицы может соответствовать несколько записей в дочерней таблице.

Связь «один-ко-многим» является самой распространенной для реляционных баз данных.

В широко распространенной нотации структуры баз данных IDEF1X отношение «один-ко-многим» изображается путем соединения таблиц

линией, которая на стороне дочерней таблицы оканчивается кружком или иным символом.

Создаем связи между таблицами с помощью вторичных ключей, как, например, в таблице «Продажи» на рис. 2.8:

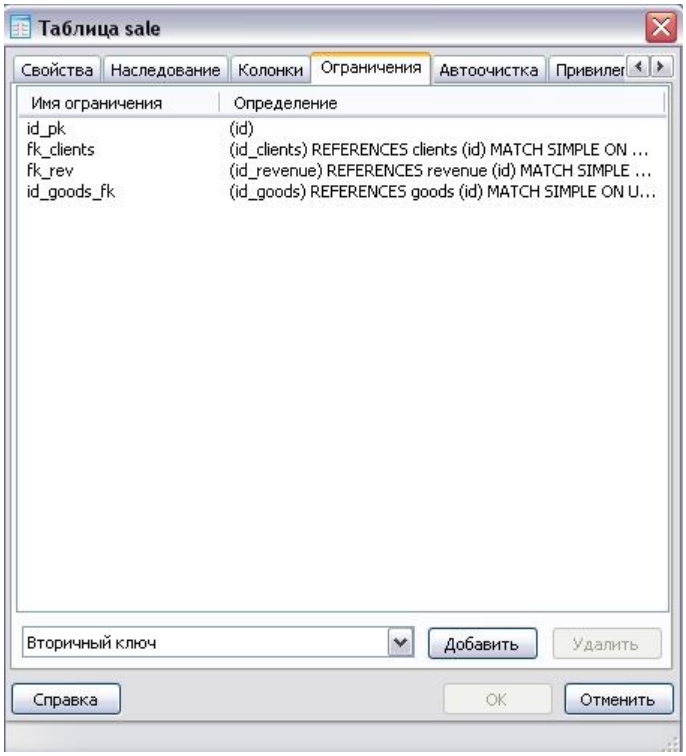


Рис.2.8. Связи между таблицами

На рис. 2.9 показан SQL-код при создании связей.

Панель SQL

```
CREATE TABLE sale
(
  id serial NOT NULL,
  id_goods integer,
  kolvo integer,
  id_clients integer,
  summa double precision,
  id_revenue integer,
  date timestamp without time zone,
  CONSTRAINT id_pk PRIMARY KEY (id),
  CONSTRAINT fk_clients FOREIGN KEY (id_clients)
      REFERENCES clients (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_rev FOREIGN KEY (id_revenue)
      REFERENCES revenue (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT id_goods_fk FOREIGN KEY (id_goods)
      REFERENCES goods (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
```

Рис. 2.9. Код создания вторичных ключей

В итоге мы имеем полноценную базу данных (см. рис. 2.10).

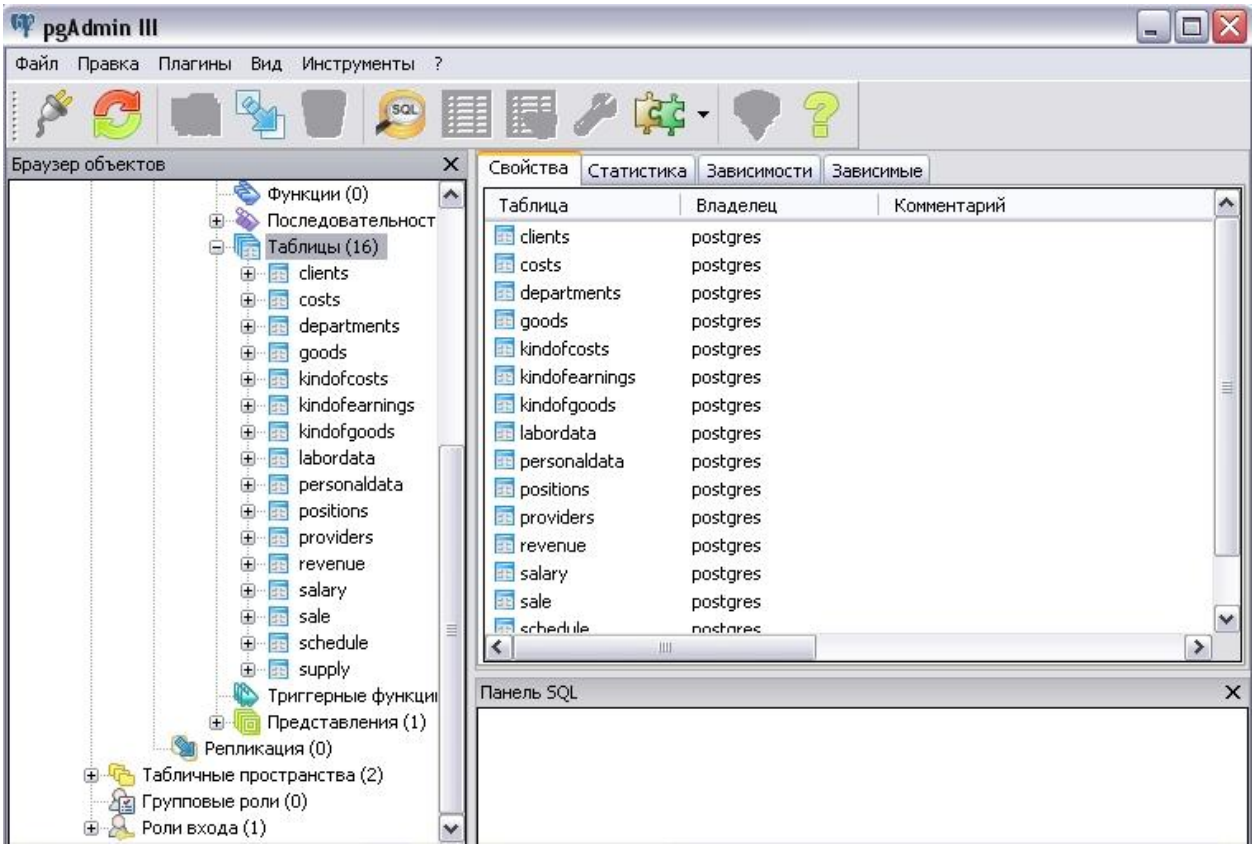


Рис. 2.10. База данных

Кроме таблиц также создадим домены.

В реляционной модели данных с понятием тип данных тесно связано понятие домена, которое можно считать уточнением типа данных.

Домен – это семантическое понятие. Домен можно рассматривать как подмножество значений некоторого типа данных имеющих определенный смысл. Домен характеризуется следующими свойствами:

- Домен имеет уникальное имя (в пределах базы данных);
- Домен определен на некотором простом типе данных или на другом домене;
- Домен может иметь некоторое логическое условие, позволяющее описать подмножество данных;
- Домен несет определенную смысловую нагрузку.

Если тип данных можно считать множеством всех возможных значений данного типа, то домен напоминает подмножество в этом множестве.

Отличие домена от понятия подмножества состоит именно в том, что домен отражает семантику, определенную предметной областью. Может быть несколько доменов, совпадающих как подмножества, но несущие различный смысл.

Основное значение доменов состоит в том, что домены ограничивают сравнения. Некорректно, с логической точки зрения, сравнивать значения из различных доменов, даже если они имеют одинаковый тип.

На рис. 2.11 показан синтаксис создания доменов в PostgreSQL.

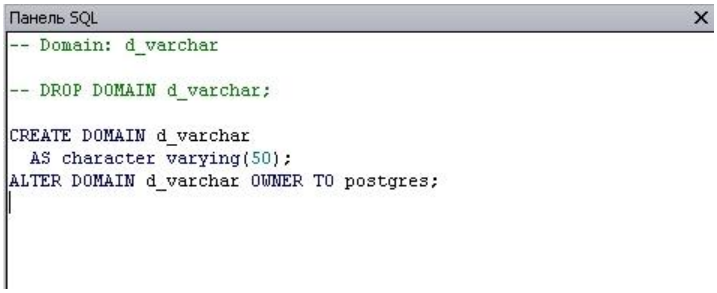


Рис. 2.11.SQL-код создания домена

Создание домена представлено на рис. 2.12.

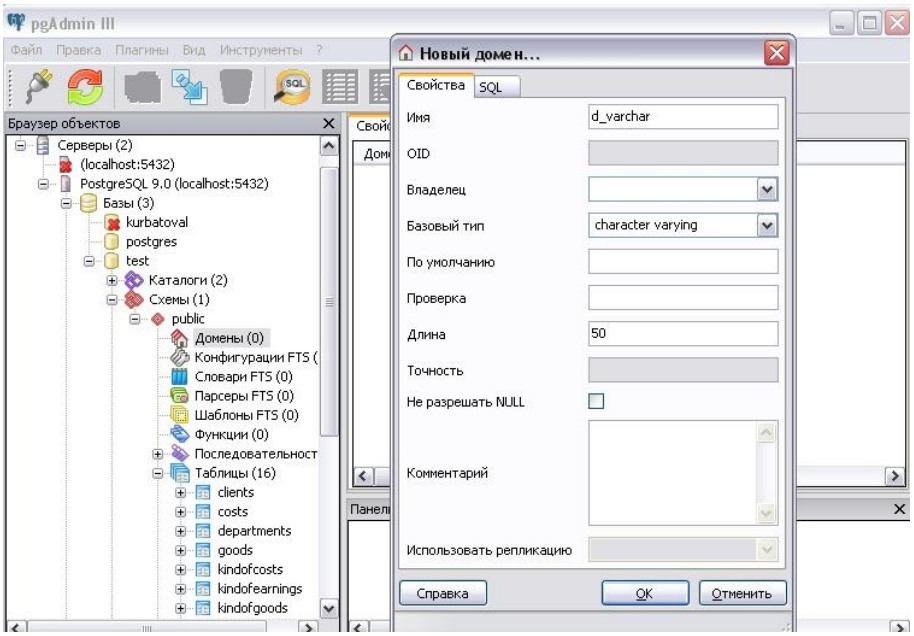


Рис. 2.12. Создание домена

Далее создаем домены для всех нужных типов данных.

Таблица 2.1

Домены БД

Имя домена	Тип	Длина	Ограничения	По умолчанию
D_integer	integer		>0	
D_smallint	smallint		>0	
D_float	float		>0	
D_varchar	varchar	50		
D_timestamp	timestamp			Current_timestamp

2.4 Программирование на стороне SQL-сервера

На стороне SQL-сервера разработаем бизнес-логику.

Бизнес-логика – в разработке информационных систем – совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает). Иначе можно сказать, что бизнес-логика — это реализация правил и ограничений автоматизируемых операций. Является синонимом термина «логика предметной области».

Проще говоря, бизнес-логика — это реализация предметной области в информационной системе. К ней относятся, например, формулы расчёта ежемесячных выплат по ссудам (в финансовой индустрии), автоматизированная отправка сообщений электронной почты руководителю проекта по окончании выполнения частей задания всеми подчиненными (в системах управления проектами), отказ от отеля при отмене рейса авиакомпанией (в туристическом бизнесе) и т. д.

Самым удобным способом реализации бизнес-логики для данного проекта в СУБД PostgreSQL являются представления.

Представления (view) – это одно из мощных средств языка SQL, предназначенное для осуществления работы механизма подсхем пользователей базы данных. Представления позволяют скрыть от пользователей схему базы данных. Они представляют собой хранимые в базе данных запросы, выраженные операторами SELECT. На базе одних представлений могут быть созданы новые представления, которые наследуют все свойства базовых представлений. Формировать представления могут пользователи с привилегиями SELECT для используемых в представлениях таблиц (базовых таблиц)[9].

Основой всех синтаксических конструкций, начинающихся с ключевого слова SELECT, является синтаксическая конструкция “табличное выражение”.

В спецификации запроса задается список выборки (список арифметических выражений над значениями столбцов результата табличного выражения и констант). В результате применения списка выборки к результату табличного выражения производится построение новой таблицы, содержащей то же число строк и результаты вычисления соответствующих арифметических выражений из списка выборки.

На рис. 2.13 представлено представление, которое считает сумму товаров, которые есть на складе.

Редактирование данных - PostgreSQL 9.0 (localhost:5432) - test - chet

Файл Правка Вид Инструменты ?

Не ограничен

	kolvo integer	id_kinds integer	id integer	price double precis	id_kind integer	summa double precis
1	50	1	1	374	1	18700
2	50	1	1	600	1	30000
3	50	1	1	1140	1	57000
4	50	1	1	515	1	25750
5	50	1	1	400	1	20000
6	30	1	1	374	1	11220
7	30	1	1	600	1	18000
8	30	1	1	1140	1	34200
9	30	1	1	515	1	15450
10	30	1	1	400	1	12000

Рис. 2.13. Представление подсчета суммы

В этом представлении берутся данные из таблиц «Поставки», «Вид товара» и «Товары», а сумма выводится уже в новом поле.

SQL-код данного представления показан на рис. 2.14.

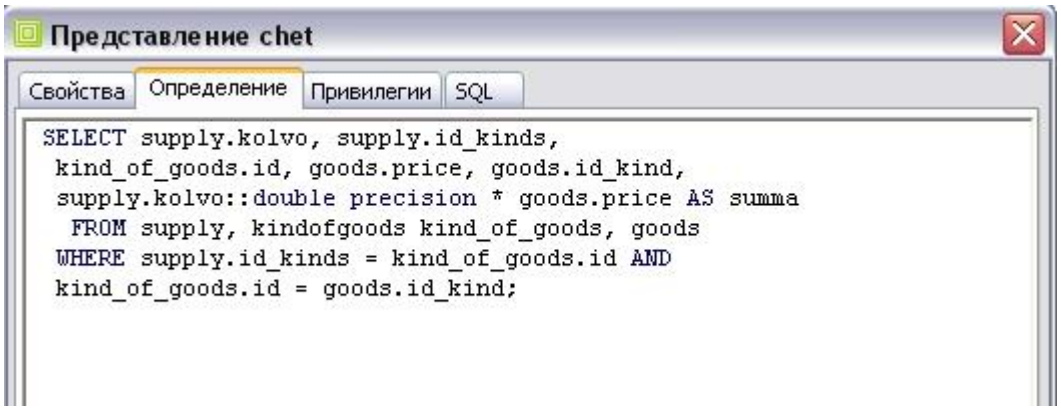


Рис. 2.14 Представление «Счет»

На этом создание базы данных можно считать завершенной.

2.5 Структура приложения Yii Framework

Прежде чем приступать к разработке своего приложения, рассмотрим ресурсы, которые предоставляет для этого Yii Framework.

После установки Yii у нас есть работающее приложение Yii, к которому можно получить доступ, введя в браузере следующие адреса: http:/hostname/basic/web/index.php или http://hostname/index.php, в зависимости от конфигурации.

Мы имеем базовый шаблон, в котором можно добавлять или удалять код и вносить в него изменения по мере необходимости.

Кроме веб-приложения, имеется консольный скрипт, называемый yii, который находится в базовом каталоге приложений. Этот сценарий может использоваться для запуска фоновых задач и задач обслуживания для приложения.

В общем случае файлы в приложении можно разделить на два типа: базовые/веб и файлы в других каталогах. К первому можно получить прямой доступ через браузер, в то время как последний не может и не должен быть.

Yii реализует архитектурный шаблон model-view-controller (MVC), который отражается в вышеупомянутой организации каталогов. Каталог моделей содержит все классы моделей, каталог views содержит все сценарии представления, а каталог контроллеров содержит все классы контроллеров.

Каждое приложение имеет сценарий входа web/index.php, который является единственным доступным в Интернете скриптом PHP в приложении. Скрипт ввода принимает входящий запрос и создает экземпляр приложения для его обработки. Приложение решает запрос с помощью своих компонентов и отправляет запрос элементам MVC. Виджеты используются в представлениях для создания сложных и динамических элементов пользовательского интерфейса.

На следующей диаграмме показана статическая структура приложения.

На рис. 2.15 наглядно показана реализация модели MVC.

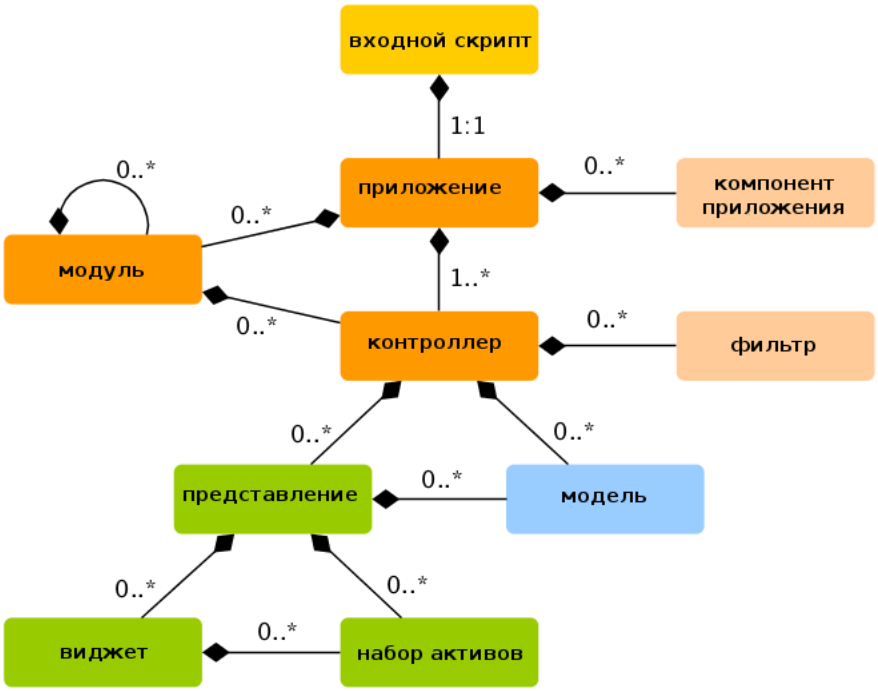


Рис. 2.15.Структура приложения Yii

Каким образом это осуществляется? Разберемся в реализации работы данной структуры.

Рассмотрим жизненный цикл работы приложения, представленный на рис. 2.16.

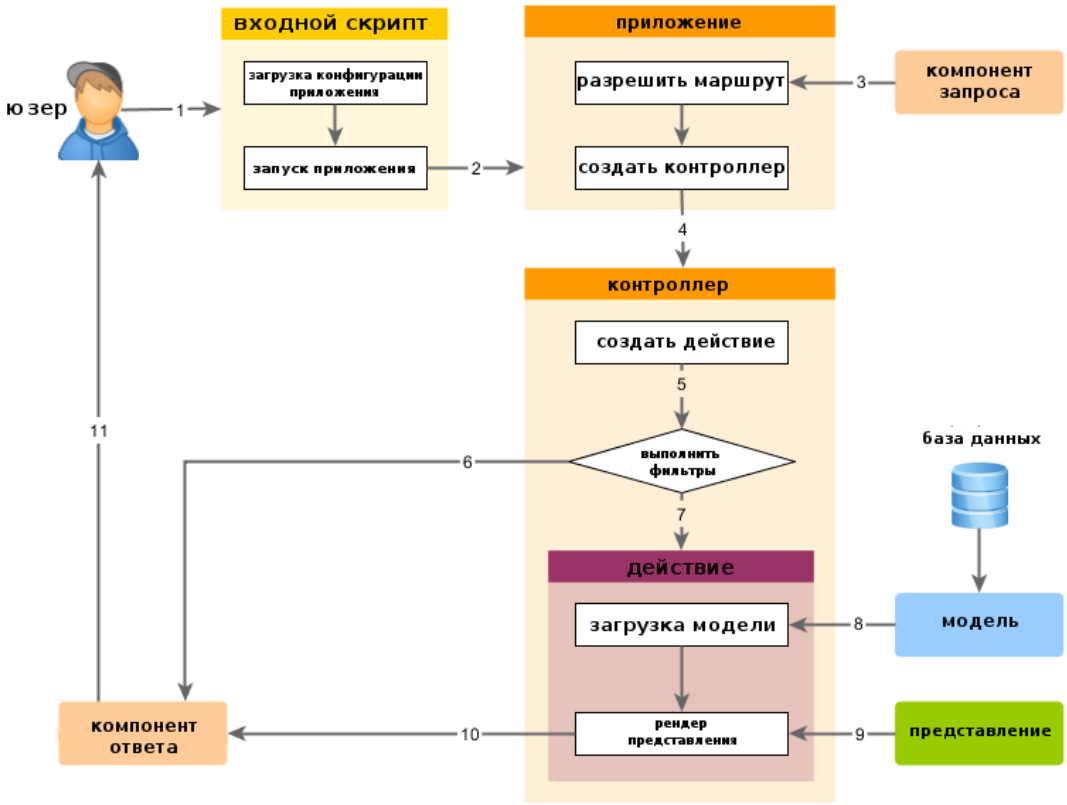


Рис. 2.16. Схема обработки запроса

Каждому шагу соответствует определенное действие.

1. Пользователь делает запрос к сценарию входа web / index.php.

2. Сценарий входа загружает конфигурацию приложения и создает экземпляр приложения для обработки запроса.

3. Приложение разрешает запрошенный маршрут с помощью компонента приложения-запроса.

4. Приложение создает экземпляр контроллера для обработки запроса.

5. Контроллер создает экземпляр действия и выполняет фильтры для действия.

6. Если какой-либо фильтр не работает, действие отменяется.

7. Если все фильтры проходят, действие выполняется.

8. Действие загружает модель данных, возможно из базы данных.

9. Действие отображает представление, предоставляя ему модель данных.

10. Отрендеренный результат возвращается компоненту приложения ответа.

11. Компонент ответа отправляет обработанный результат в браузер пользователя.

Таким образом, мы рассмотрели структуру базового приложения Yii Framework и теперь, на его основе, можем приступить к созданию своего спроектированного ранее приложения.

ГЛАВА 3. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Программное обеспечение представляет собой совокупность программ, правил и процедур, сплетенных в единый искусственный разум, с целью быстрой точной обработки информации и предоставления конечного результата.

Спецификация – это законченное поведение программы и требования к ней, которую требуется разработать. Web-приложение должно иметь множество отдельных страниц, обеспечивающих ведение информации. Должна быть обеспечена возможность управления базой данных, а именно: добавление информации в таблицу, удаление информации из таблицы, сохранение или обновление данных. Также должна быть создана система администрирования с отдельными правами администратора и других пользователей, также реализована возможность загрузки и хранения файлов и документов на сайте.

3.1 Подключение к БД

Для работы приложения в первую очередь требуется установить соединение с созданной ранее базой данных.

В Yii для этого используются объекты доступа к данным (DAO).

Объекты доступа к данным (DAO) предоставляют общий API для доступа к данным, хранящимся в различных СУБД. Это позволяет без проблем поменять используемую СУБД на любую другую без необходимости изменения кода, использующего DAO для доступа к данным.

Yii DAO является надстройкой над PHP DataObjects (PDO).

PHP DataObjects — расширение для PHP, предоставляющее разработчику простой и универсальный интерфейс для доступа к различным базам данных.

PDO предлагает единые методы для работы с различными базами данных, хотя текст запросов может немного отличаться. Так как многие СУБД реализуют свой диалект SQL, который в той или иной мере поддерживает стандарты ANSI и ISO, то при использовании простых запросов можно добиться совместимости между различными языками. На практике это означает, что можно достаточно легко перейти на другую СУБД, при этом, не меняя или частично изменяя код программы.

Скорость работы и масштабируемость: PDO не использует абстрактных слоёв для подключения к БД, наподобие ODBC, а использует для разных БД их «родные» драйверы, что позволяет добиться высокой производительности. В настоящее время для PDO существуют драйверы практически ко всем общеизвестным СУБД и интерфейсам. Так же позволяет работать сразу с несколькими базами данных одновременно.

Yii DAO состоит из четырёх основных классов:

- CDbConnection: представляет подключение к базе данных;
- CDbCommand: представляет запрос к базе данных, который необходимо выполнить;
- CDbDataReader: представляет однонаправленный поток строк данных, возвращаемых в ответ на запрос;
- CDbTransaction: представляет транзакцию базы данных.

Для установления соединения с базой необходимо создать экземпляр класса CDbConnection и активировать его (см. листинг 3.1). Дополнительная информация, необходимая для подключения к БД (хост, порт, имя пользователя, пароль и т.д.), указывается в DSN (DataSourceName).

Листинг 3.1. Подключение к БД

```
<?php
return [
'class' => 'yii\db\Connection',
```

Продолжение. Листинг 3.1. Подключение к БД

```
'dsn' => 'pgsql:host=localhost;port=5432;dbname=test',
'username' => 'postgres',
'password' => '123',
'charset' => 'utf8',];
```

Когда соединение с БД установлено, мы можем выполнять SQL-запросы. Но прежде чем создать код запросов, нам необходимо определить, какие данные мы будем получать от пользователей и каким правилам они должны соответствовать. Для фиксации этой информации можно использовать класс модели данных.

3.2 Создание моделей данных

Модель используется для хранения данных и применимых к ним бизнес-правил.

Модель представляет собой отдельный объект данных. Это может быть запись таблицы базы данных или HTML-форма с полями для ввода данных. Каждое поле объекта данных представляется атрибутом модели. Каждый атрибут имеет текстовую метку и может быть проверен на корректность, используя набор правил.

Yii предоставляет два типа моделей: модель формы и ActiveRecord. Оба типа являются наследниками базового класса CModel.

Модель формы — это экземпляр класса CFormModel. Она используется для хранения данных, введённых пользователем. Как правило, мы получаем эти данные, обрабатываем, а затем избавляемся от них. Например, на странице авторизации модель такого типа может быть использована для представления информации об имени пользователя и пароле. Подробное описание работы с формами приведено в разделе Работа с формами.

ActiveRecord (AR) — это шаблон проектирования, используемый для абстрагирования доступа к базе данных в объектно-ориентированной форме.

Каждый объект AR является экземпляром класса CActiveRecord или класса, унаследованного от него, и представляет отдельную строку в таблице базы данных.

В зависимости от того, каким образом используются введённые данные, мы можем использовать два типа моделей. Если мы получаем данные, обрабатываем их, а затем удаляем, то используем модель формы; если же после получения и обработки данных мы сохраняем их в базе данных, то используем ActiveRecord. Оба типа моделей данных используют один и тот же базовый класс CModel, который определяет общий интерфейс, используемый формами.

Часто при работе с формами для каждого поля требуется отображать его метку. Она подсказывает пользователю, какие данные ему требуется ввести в поле. Мы, конечно, можем задать метки полей в представлении, но, если указать их непосредственно в модели данных, мы выиграем в удобстве и гибкости.

По умолчанию CModel в качестве меток возвращает названия атрибутов. Изменить их можно, переопределив метод attributeLabels().

Рассмотрим листинг 3.2 создания модели на основе таблицы «Клиенты».

Листинг 3.2. Модель таблицы «Клиенты»

```
<?php
namespaceapp\models;
useYii;
class Clients extends \yii\db\ActiveRecord
{
public static function tableName()
   {
return 'clients'; }
public function rules()
   {
return [
          [['surname', 'name', 'middlename', 'email'], 'string', 'max' => 50],
       ]}
public function attributeLabels()
   {
return [
          'id' => 'ID',
```

Продолжение. Листинг 3.2. Модель таблицы «Клиенты»

```
'surname' => 'Surname',
'name' => 'Name',
          'middlename' => 'Middlename',
          'email' => 'Email',];
}
public function getSales()
   {
return $this->hasMany(Sale::className(), ['id_clients' => 'id']);  }
}
```

В каждой модели обязательно прописываем возвращаемые параметры таблиц.

Теперь, когда готова модель, можно приступать к написанию кода для работы с ней. Всю логику обработки мы помещаем в действие контроллера.

3.3 Создание контроллеров

Контроллер (controller) — это экземпляр класса CController или унаследованного от него класса. Он создается объектом приложения в случае, когда пользователь его запрашивает. При запуске контроллер выполняет соответствующее действие, что обычно подразумевает создание соответствующих моделей и отображение необходимых представлений. В самом простом случае действие — это метод класса контроллера, название которого начинается на action[8].

У контроллера есть действие по умолчанию, которое выполняется в случае, когда пользователь не указывает действие при запросе. По умолчанию это действие называется index. Изменить его можно путём установки значения CController::defaultAction.

Следующий листинг 3.3 определяет контроллер site с действиями index (действие по умолчанию) и contact.

Листинг 3.3. Контроллер «SiteController»

```
classSiteController extends CController
{
public function actionIndex()
   {
      // ...
   }
public function actionContact()
{
      // ...
   }}
```

Экземпляр контроллера создаётся, когда CWebApplication обрабатывает входящий запрос. Получив идентификатор контроллера, приложение использует следующие правила для определения класса контроллера и его местоположения:

- если установлено свойство CWebApplication::catchAllRequest, контроллер будет создан на основе этого свойства, а контроллер, запрошенный пользователем, будет проигнорирован. Как правило, это используется для установки приложения в режим технического обслуживания и отображения статической страницы с соответствующим сообщением;
- если идентификатор контроллера обнаружен в CWebApplication::controllerMap, то для создания экземпляра контроллера будет использована соответствующая конфигурация контроллера;

Как было упомянуто выше, действие — это метод, имя которого начинается на action. Более продвинутый способ — создать класс действия и указать контроллеру создавать экземпляр этого класса при необходимости. Такой подход позволяет использовать действия повторно.

Для создания класса действия необходимо выполнить следующее (см. листинг 3.4).

Листинг 3.4. Создание класса

```
classUpdateActionextendsCAction
{publicfunctionrun(){
// некоторая логика действия}}
```

Чтобы контроллер знал об этом действии, необходимо переопределить метод actions() в классе контроллера (см. листинг 3.5).

Листинг 3.5.Переопределение метода

```
classPostControllerextendsCController
{
publicfunctionactions()
{
returnarray(
'edit'=>'application.controllers.post.UpdateAction',
);}}
```

Контроллеры создаем для каждой модели.

3.4 Создание видов

Представления (виды) – это часть MVC архитектуры, это код, который отвечает за представление данных конечным пользователям. В веб приложениях виды создаются обычно в виде видов - шаблонов, которые суть PHP скрипты, в основном содержащие HTML код и код PHP, отвечающий за представление и внешний вид [10]. Виды управляются компонентом приложения view, который содержит часто используемые методы для упорядочивания видов.

Как было написано ранее, вид – это просто PHP скрипт, состоящий из PHP и HTML кода. В примере ниже – вид, который представляет форму таблицы «Клиенты». PHP код в листинге 3.6 генерирует динамический контент, как, например, заголовок страницы и саму форму, тогда как HTML организует полученные данные в готовую html страницу:

Листинг3.6.Код вида таблицы «Клиенты»

```
<?php
useyii\helpers\Html;
useyii\widgets\DetailView;
$this->title = $model->name;
$this->params['breadcrumbs'][] = ['label' => 'Clients', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="clients-view">
<h1><?= Html::encode($this->title) ?></h1>
```

Продолжение. Листинг3.6. Код вида таблицы «Клиенты»

```
<p>
<?= Html::a('Update', ['update', 'id' => $model->id], ['class' => 'btnbtn-primary']) ?>
<?= Html::a('Delete', ['delete', 'id' => $model->id], [
            'class' => 'btnbtn-danger',
            'data' => [
                'confirm' => 'Are you sure you want to delete this item?',
'method' => 'post',
            ],
        ]) ?>
</p>
</div>
```

Также вставим виджеты.

Виджеты представляют собой многоразовые строительные блоки, используемые в представлениях для создания сложных и настраиваемых элементов пользовательского интерфейса в рамках объектно-ориентированного подхода. Для того, чтобы использовать виджет в представлении, достаточно вызвать метод yii\base\Widget::widget(). Метод принимает массив настроек для инициализации виджета и возвращает результат его рендеринга (см. листинг 3.7):

Листинг3.7. Виджет в представлении «Клиенты»

```
<?=DetailView::widget([
        'model' => $model,
        'attributes' => [
            'id',
            'surname',
'name',
            'middlename',
            'email:email',
],
    ]) ?>
```

При создании виджетов, следует придерживаться основных принципов концепции MVC. В общем случае, основную логику следует располагать в классе виджета, разделяя при этом код, отвечающий за разметку в представления.

Разрабатываемые виджеты должны быть самодостаточными. Это означает, что для их использования должно быть достаточно всего лишь добавить виджет в представление.

3.5 Разработка запросов к базе данных

Запросы, сделанные к приложению, представлены в терминах yii\web\Request объектов, которые предоставляют информацию о параметрах запроса, HTTP заголовках, cookies и т.д. Для получения доступа к текущему запросу вы должны обратиться к объекту request application component, который по умолчанию является экземпляром yii\web\Request.

Чтобы получить параметры запроса, вы должны вызвать методы get() и post() компонента request. Они возвращают значения переменных $_GET и $_POST соответственно[4].

Построенный поверх DAO, построитель запросов позволяет конструировать SQL выражения в программируемом и независимом от СУБД виде. В сравнении с написанием чистого SQL выражения, использование построителя помогает писать более читаемый связанный с SQL код и генерировать более безопасные SQL выражения.

Использование построителя запросов, как правило, включает два этапа:

1. Создание объекта yii\db\Query представляющего различные части (такие как SELECT, FROM) SQL выражения SELECT.
2. Выполнить запрос методом yii\db\Query (таким как all()) для извлечения данных из базы данных.

Листинг 3.8 показывает обычное использование построителя запросов.

Листинг 3.8. Построитель запросов

```
$rows = (new \yii\db\Query())
->select(['id', 'email'])
   ->from('user')
   ->where(['last_name' =>'Марков'])
   ->limit(10)
->all();
```

Приведённый выше код создаёт и выполняет следующее SQL выражение(см. листинг 3.9), где параметр:last_name привязывается к строке 'Марков'.

Листинг 3.9. Привязка строки

```
SELECT`id`, `email`
FROM`user`
WHERE`last_name`= :last_name
LIMIT10
```

Создав объект yii\db\Query, можно вызвать различные методы для создания различных частей SQL выражения. Имена методов напоминают ключевые слова SQL, используемые в соответствующих частях SQL запроса. Например, чтобы указать FROM часть запроса, вызовем метод from(). Все методы построителя запросов возвращают свой объект, который позволяет объединять несколько вызовов в цепочку.

Метод select() определяет фрагмент SELECT SQL запроса. Можно указать столбцы, которые должны быть выбраны, они должны быть указаны в виде массива или строки[5]. Имена столбцов автоматически экранируются при создании SQL-запроса при его генерации из объекта yii\db\Query (см. листинг 3.10):

Листинг 3.10.Запрос

```
$query->select(['id', 'email'])
```

Имена столбцов выбираем вместе с префиксами таблиц и/или алиасами столбцов, также как при записи обычного SQL выражения. Например, в листинге 3.11:

Листинг 3.11. Выбор имен столбцов

```
$query->select(['user.idASuser_id', 'email'])
```

Если мы не используем метод select() при создании запроса, будет использована*, что означает выбрать все столбцы.

Кроме имён столбцов, возможно также использовать SQL выражения (см. листинг 3.12). Для этого нужно использовать формат массива для использования выражений, которые содержат запятые для предотвращения некорректного автоматического экранирования:

Листинг 3.12.SQL-Выражение

```
$query->select(["CONCAT(first_name, ' ', last_name) AS full_name", 'email'])
```

Когда имеешь дело со сложными данными, иногда может потребоваться использовать несколько разных моделей для обработки данных, введенных пользователем. Для примера, информация о зарплате сотрудника хранится в таблице salary, а данные профиля хранятся в таблице personal_data, и нам нужно обрабатывать входные данные о пользователе через модели Salary и Personal_data. Учитывая поддержку Yii моделей и форм, можно решить данную задачу способом, не сильно отличающимся от обработки одинарной модели.

Создадим форму, которая позволила бы вам собирать данные для обеих моделей Salary и Personal_data.

Действие контроллера для обработки данных пользователя и данных профиля может быть написано следующим образом (см. листинг 3.13).

Листинг 3.13. Обработка данных из двух моделей

```
publicfunctionactionUpdate($id){
$salary = Salary::findOne($id);
$personal_data = Personal_data::findOne($id);
if (!isset($salary, $personal_data)) {
thrownewNotFoundHttpException("The user was not found.");}
$salary->scenario = 'update';
$personal_data->scenario = 'update';
if ($salary->load(Yii::$app->request->post()) &&$personal_data->load(Yii::$app-
>request->post())) {
$isValid = $ salary->validate();
$isValid = $personal_data->validate() &&$isValid;
if ($isValid) {
$salary->save(false);
$personal_data->save(false);
return$this->redirect([' salary/view', 'id' =>$id]);}    }
return$this->render('update', [
' salary' =>$ salary,
'personal_data' =>$personal_data,]);    }}
```

В действии update, мы сначала загружаем из базы модели $salary и $personal_data. Затем мы вызываем метод yii\base\Model::load() для заполнения этих двух моделей данными, введенными пользователем. В случае успеха мы проверяем модели и сохраняем их[6]. В противном случае мы рендерим представление update, которое содержит контент, представленный в листинге 3.14.

Листинг 3.14. Код действияupdate

```
<?php
useyii\helpers\Html;
useyii\widgets\ActiveForm;
$form = ActiveForm::begin([
'id' =>' salary-update-form',
'options' => ['class' =>'form-horizontal'],
]) ?>
<?=$form->field($salary, 'id') ?>
<?=$form->field($personal_data, 'name') ?>
<?= Html::submitButton('Update', ['class' =>'btnbtn-primary']) ?>
<?phpActiveForm::end()?>
```

Как видно в примере, в представлении update рендерятся поля ввода для двух моделей $salary и $personal_data.

На этом разработка закончена, а само приложение можно рассмотреть как модульную схему всех его компонентов, представленную на рис. 3.1.

Модульная схема дает полное представление о структуре разработанного приложения, количестве и составе входящих в него файлов.

В модуле «Models» описаны модели структур всех таблиц (их полей и связей), также для каждой модели создан отдельный контроллер(модуль «Controllers»), выполняющий функции управления базой данных (удаление, добавление, просмотр, обновление). В модуле viewsописаны все представления для каждой из таблиц. В центральном модуле представлены главные файлы приложения, отвечающие за подключение к базе данных, а также отображение главных страниц.

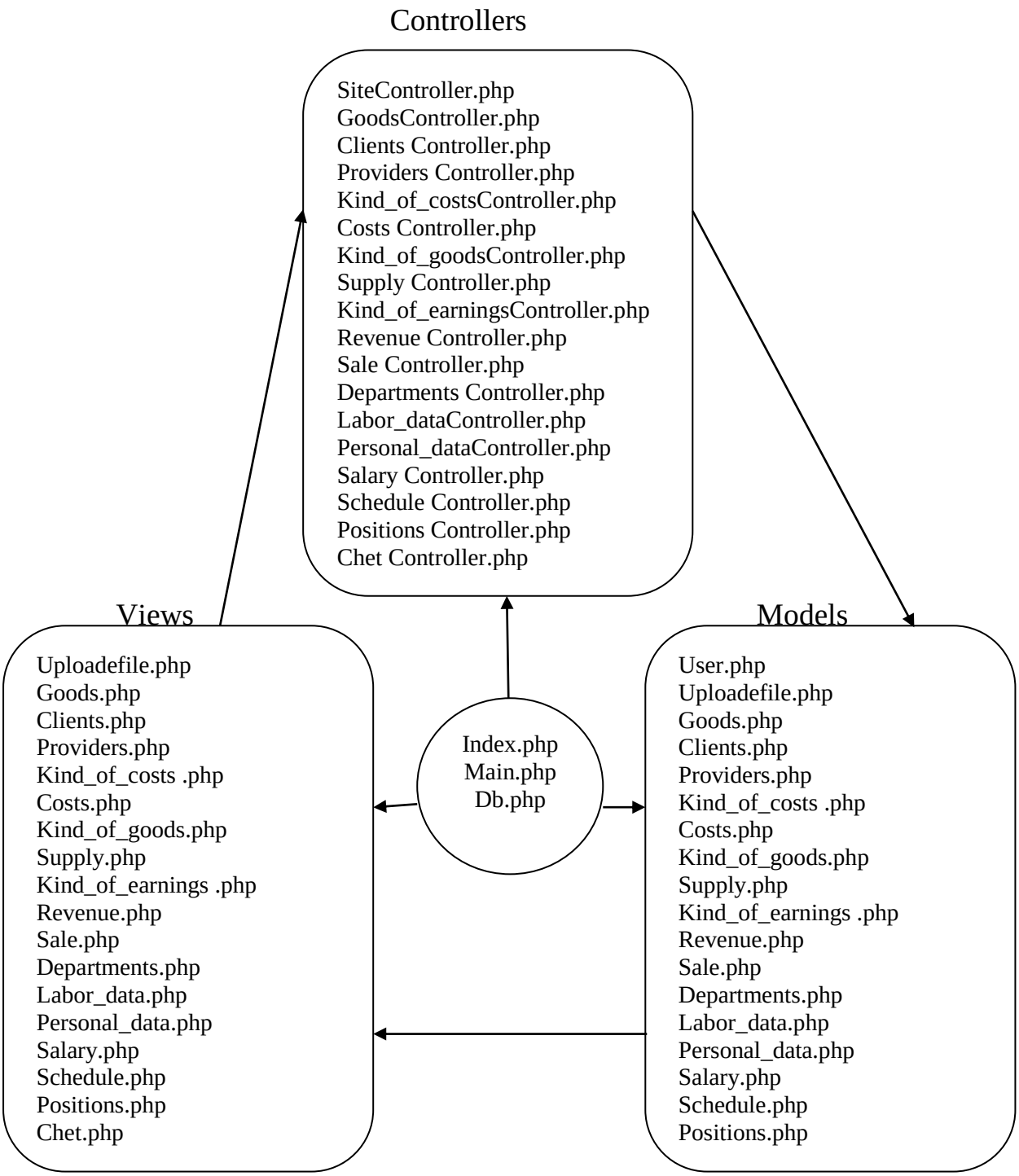


Рис. 3.1. Модульная схема приложения

После того, как приложение было создано, необходимо провести тестирование его работоспособности.

ГЛАВА 4. ТЕСТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ

В процессе создания информационных систем нередки ошибки и дефекты – это вполне ожидаемое и нормальное явление, а в условиях ограниченных временных ресурсов неизбежно возникает необходимость в организации эффективного контроля и управления всем процессом тестирования.

Тестирование программного обеспечения – это проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

В более широком смысле, тестирование – это одна из техник контроля качества, включающая в себя действия по планированию работ, проектированию тестов, выполнению тестирования и анализу полученных результатов.

4.1 Функциональное тестирование

Функциональное тестирование является одним из ключевых видов тестирования, задача которого – установить соответствие разработанного программного обеспечения (ПО) исходным функциональным требованиям заказчика. То есть проведение функционального тестирования позволяет проверить способность информационной системы в определенных условиях решать задачи, нужные пользователям.

Обычно, функциональное тестирование проводится на двух уровнях:

- Компонентное (модульное) тестирование. Тестирование отдельных компонентов программного продукта, сфокусированное на их специфике, назначении и функциональных особенностях;

- Интеграционное тестирование. Данный вид тестирования проводится после компонентного тестирования и направлен на выявление дефектов взаимодействия различных подсистем на уровне потоков управления и обмена данными.

Так как мы имеем дело с небольшим приложением, при нахождении ошибок можем сразу же исправить их.

На рис. 4.1 показана главная страница, где сверху расположена панель со ссылками на различную информацию.

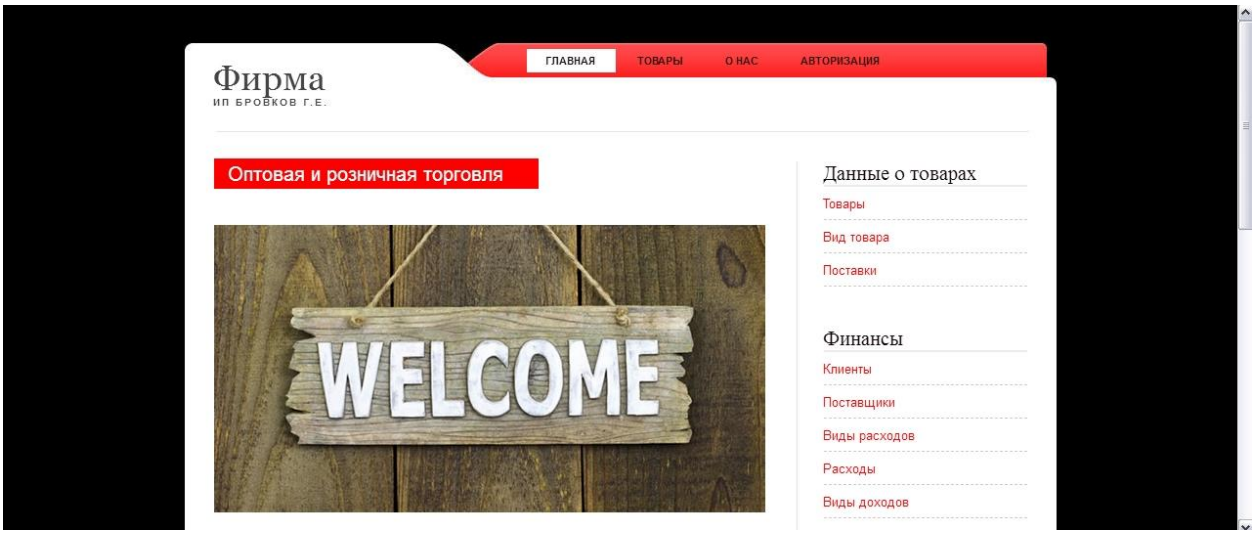


Рис. 4.1. Главная страница

Также на рис. 4.2 представлены ссылки на данные таблиц базы данных.

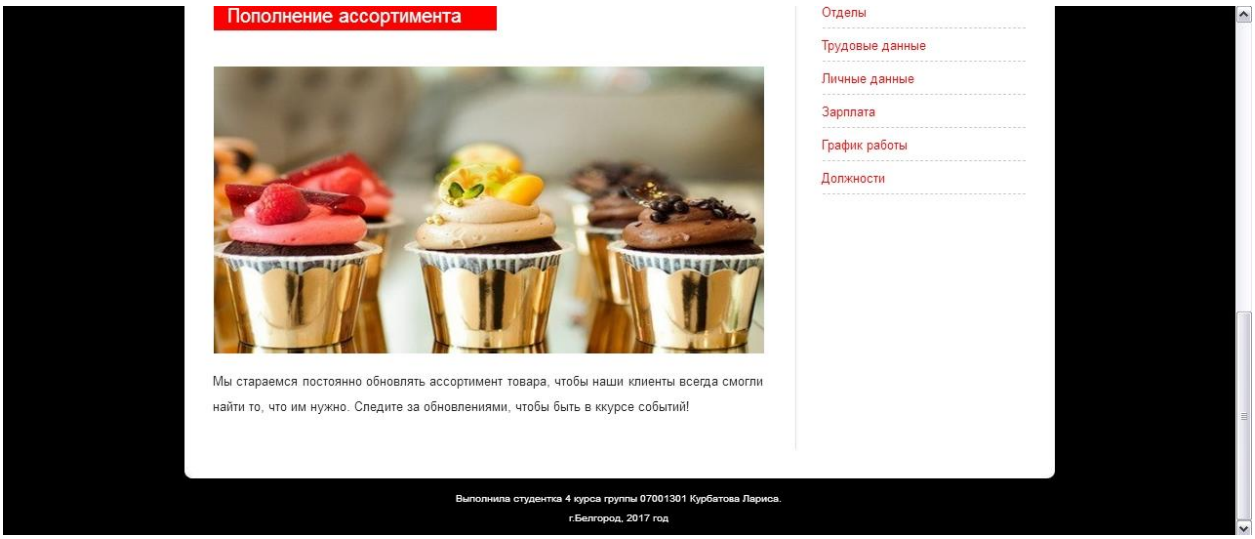


Рис. 4.2. Главная страница

Во вкладке товары на рис. 4.3 наглядная информация о товарах, которые входят в ассортимент фирмы.

Рис. 4.3. Товары

Так как товаров достаточно много, для удобства просмотра реализованы кнопки для быстрого перехода на следующую страницу (см.рис. 4.4).

Рис. 4.4. Товары

Далее рассмотрим работу функционала web-приложения, который предоставляет разные возможности доступа для разных пользователей.

4.2. Тестирование подсистемы администратора

На рис. 4.5 представлена авторизация на сайте.

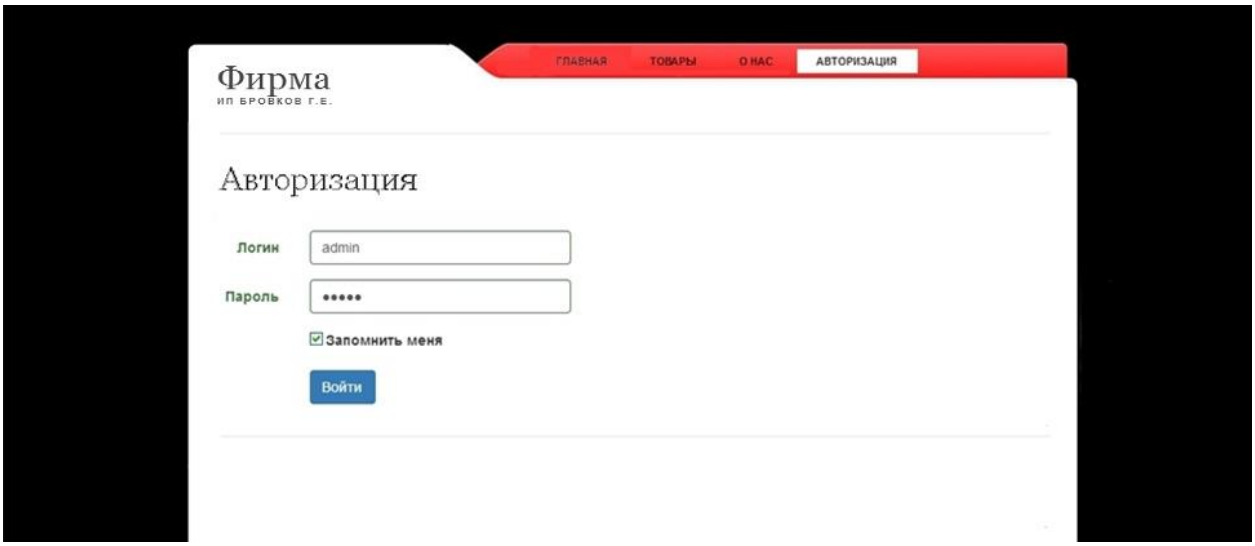


Рис. 4.5. Авторизация

После авторизации администратору становится доступна кнопка «Управление», скрытая от других пользователей (см. рис. 4.6).

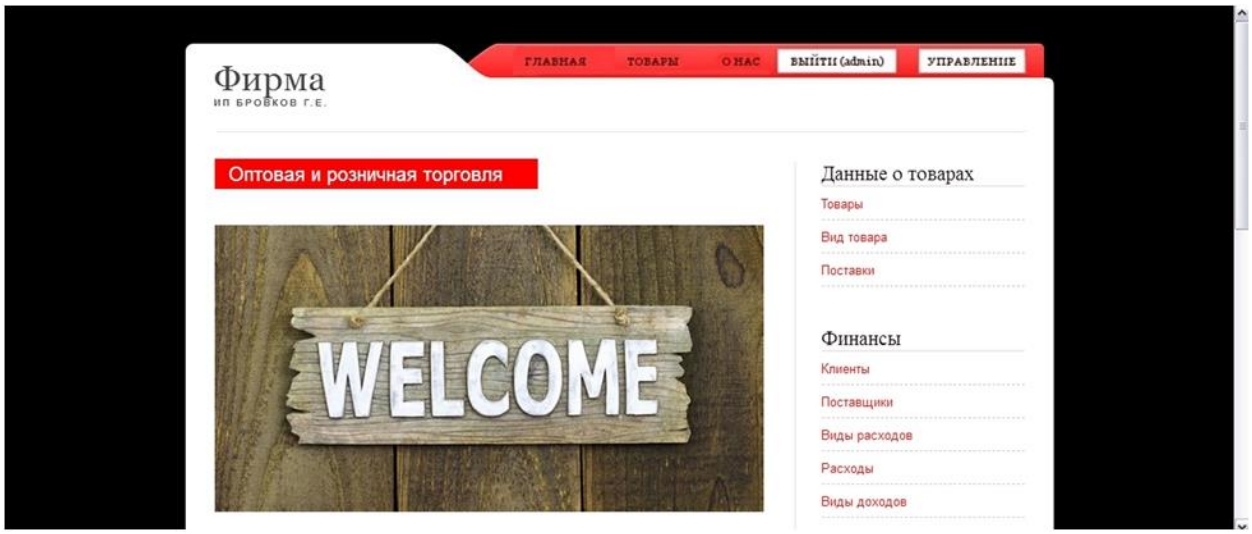


Рис. 4.6. Авторизация

Главная задача администратора – управление пользователями, реализованная с помощью RBAC.

RBAC (Role Based Access Control) – управление доступом на основе ролей. Разграничение доступа на основе ролей позволяет реализовать гибкие и динамические правила доступа для функционирования web-приложения.

При нажатии на кнопку «Управление» на рис. 4.7 открывается форма управления сайтом.

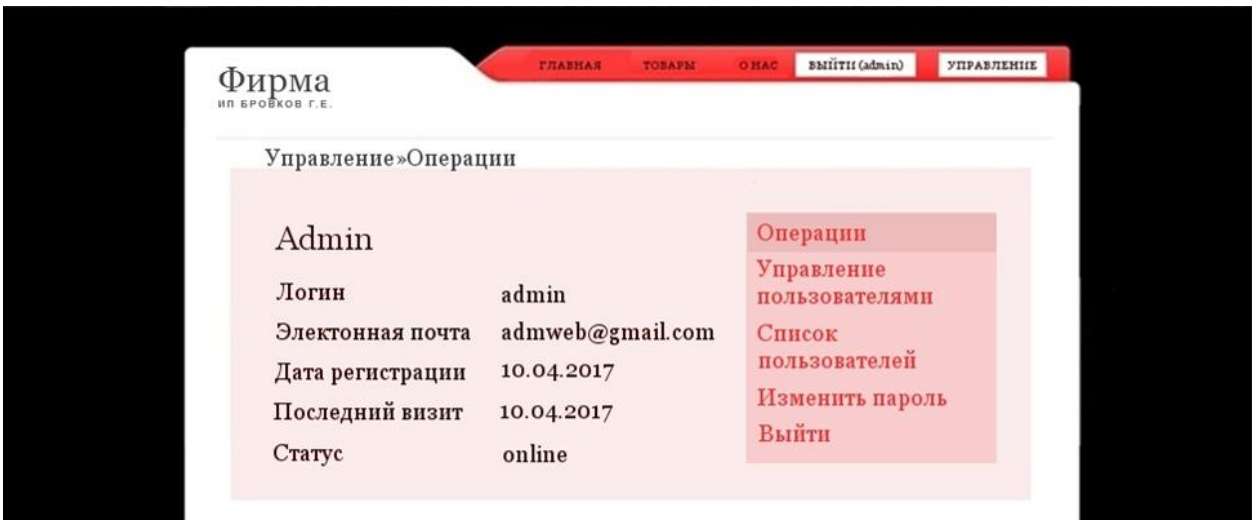


Рис. 4.7.Форма управления

Таблица данных или GridView - это один из сверхмощных Yii виджетов. Он может быть полезен, если необходимо быстро создать административный раздел системы. GridView использует данные, как провайдер данных и отображает каждую строку, используя columns для предоставления данных в таблице (см. рис. 4.8).

Рис. 4.8. Настройка ролей

При нажатии на кнопку «Список пользователей» открывается форма со всеми зарегистрированными пользователями и данными их активности. Также здесь находится ссылка на создание новых пользователей (см. рис. 4.9). Как мы видим, создает аккаунты только администратор, после чего раздает данные сотрудникам.

Рис. 4.9.Просмотр пользователей

В любое время администратор может изменить свой пароль, что представлено на рис. 4.10:

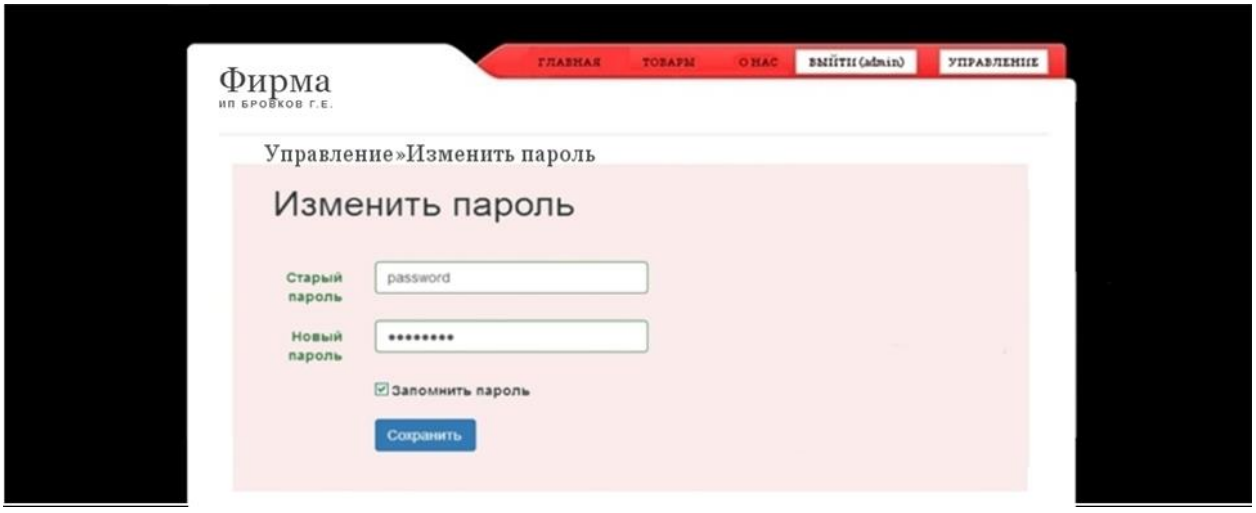


Рис. 4.10. Изменение пароля

Далее рассмотрим работу пользователей, то есть сотрудников нашей организации, которым будет предоставлен доступ с особыми правами.

4.3 Тестирование подсистемы пользователей

Доступ к данным из базы есть у всех пользователей, но возможность полноценной работы, а именно манипулирования данными: добавления, удаления, обновления (см. рис. 4.11), есть только у пользователей с правами товароведа.

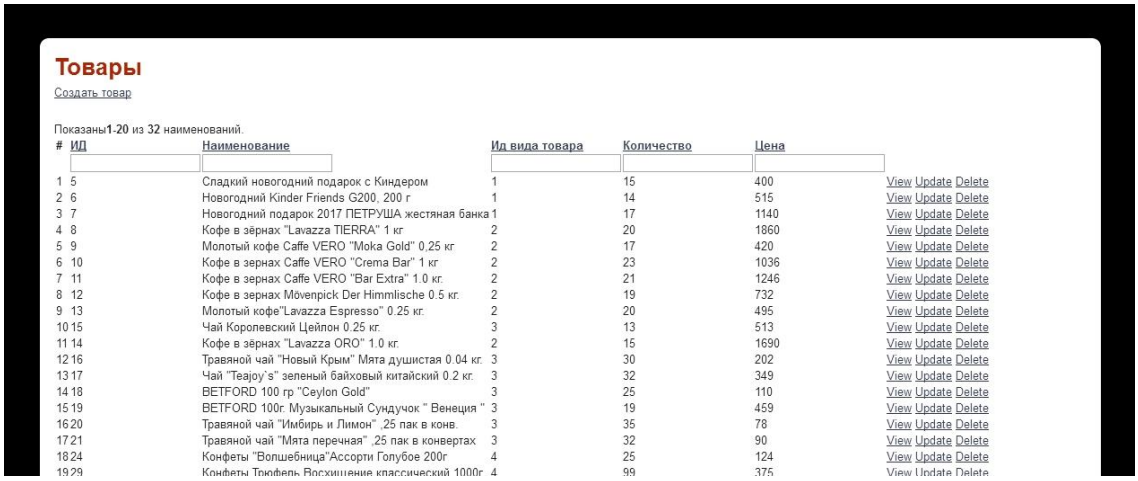
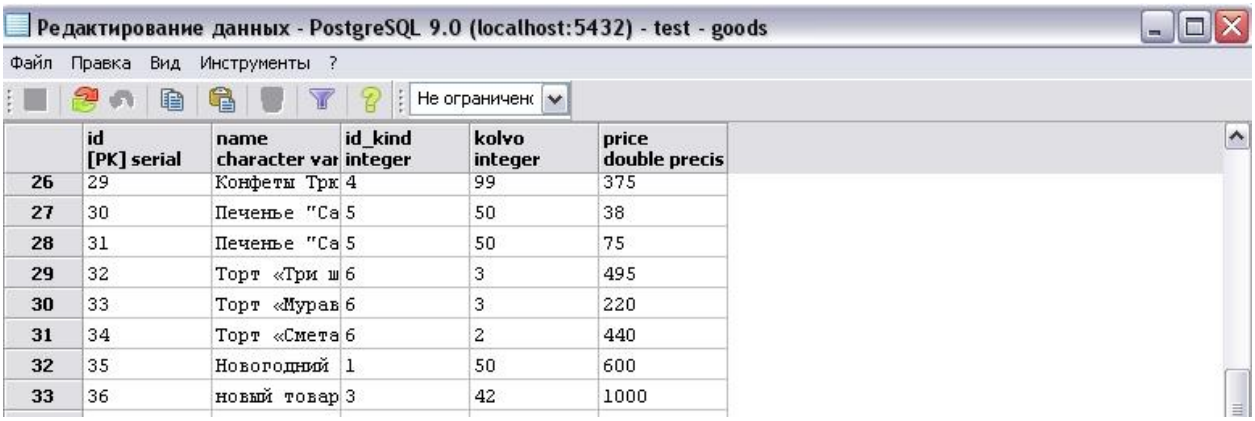


Рис. 4.11. Данные из таблицы «Товары»

При нажатии на кнопку «Создать» открывается форма создания новой записи, представленная на рис. 4.12.

Рис. 4.12. Создание новой записи

Проверим, действительно ли создалась новая запись. Для этого откроем pgadmin, зайдем в таблицу товары и пролистаем до конца. Действительно, введенный товар появился, это видно на рис. 4.13.



	id [PK] serial	name character var	id_kind integer	kolvo integer	price double precis
26	29	Конфеты Трк	4	99	375
27	30	Печенье "Са	5	50	38
28	31	Печенье "Са	5	50	75
29	32	Торт «Три ш	6	3	495
30	33	Торт «Мурав	6	3	220
31	34	Торт «Смета	6	2	440
32	35	Новогодний	1	50	600
33	36	новый товар	3	42	1000

Рис. 4.13. Проверка записи в таблице

Если что-то было введено некорректно или же информация изменилась, всегда можно изменить ее, нажав кнопку «Обновить» (см. рис. 4.14).

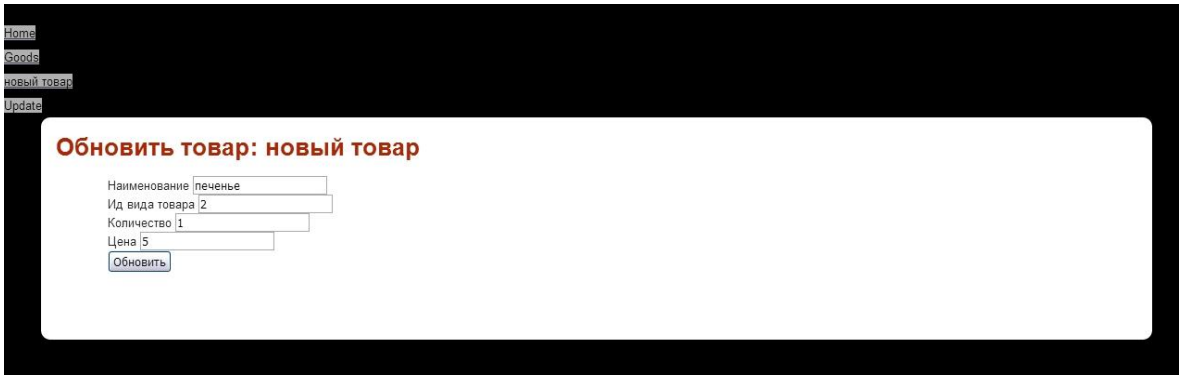


Рис. 4.14. Обновление товара

Появляется форма (см. рис. 4.15) чтобы пользователь имел возможность сразу проверить правильность данных.

Рис. 4.15. Обновленная запись

Во вкладке документы находятся все загруженные на сайт документы.

Загрузка файлов выполняется при помощи класса yii\web\UploadedFile, который представляет каждый загруженный файл в виде объекта UploadedFile (см. рис.4.16).

Рис. 4.16. Размещенные документы

Нажав кнопку «Загрузить» (см. рис. 4.17) можно скачать любой из представленных файлов на свой компьютер.

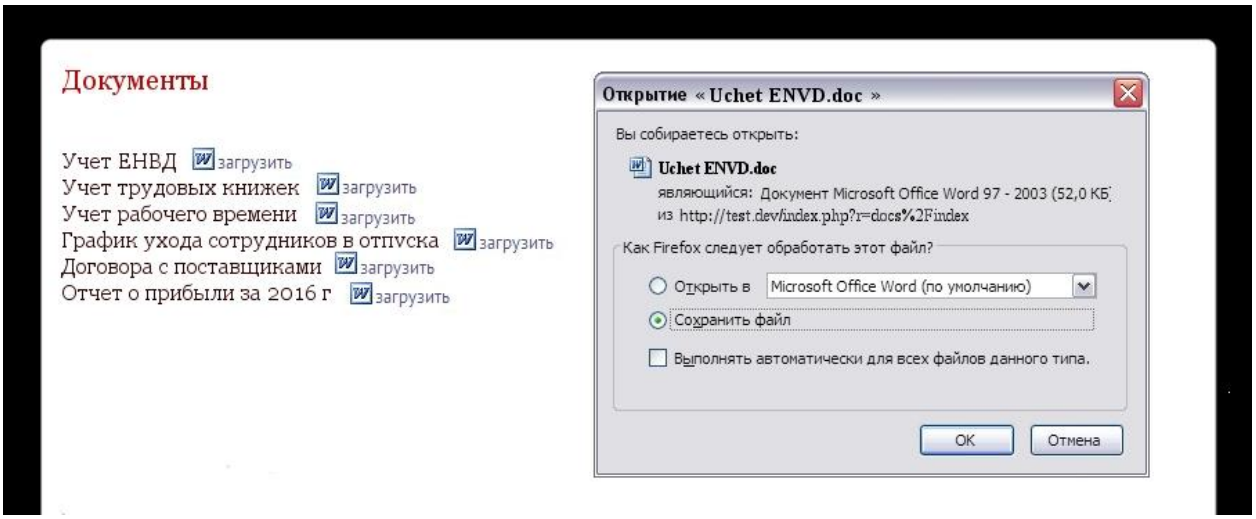


Рис. 4.17. Загрузка документа

Таким образом, мы протестировали готовое web-приложение и убедились, что все работает правильно, дефектов нет.

ЗАКЛЮЧЕНИЕ

В рамках работы над выпускной квалификационной работой было разработано web-приложение для осуществления учетной автоматизированной деятельности на предприятии индивидуального предпринимателя. В разработанном приложении предусмотрено разделение пользователей на группы с разным уровнем доступа к элементам системы. Реализованные в разработанной среде возможности позволяют администратору: регистрировать пользователей в системе, назначать уровень доступа к конкретным материалам, пользователям с правами товароведа вести учетную базу данных, а также решены отдельные вопросы, касающиеся документной базы предприятия.

Программирование на стороне сервера было осуществлено в СУБД PostgreSQL с помощью утилиты pgAdmin, а клиентская часть приложения с помощью фреймворка Yii 2.0 общего назначения.

Были созданы модели, контроллеры, представления, соответствующие паттерну MVC.

В ходе выполнения выпускной квалификационной работы были приобретены практические и теоретические знания и навыки в области создания удаленных баз данных,web-программировании, а также создания web-приложений.

Предлагаемое web-приложение отличается гибкостью в плане замены любых составляющих частей приложения. Все технологии являются бесплатными и свободно распространяемыми.

Разработанная система позволяет эффективно управлять учетной деятельностью предприятия без использования лишних ресурсов.

Таким образом, задачи, поставленные в начале выпускной квалификационной работы, были выполнены, а цель – достигнута.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Карпова Т.С. Базы данных: модели, разработка, реализация / Т.С. Карпова. – СПб.: Питер, 2001. – 304 с.

2. Печникова В.Н. Создание Web-страниц и Web-сайтов. Самоучитель: учебное пособие/ В. Н. Печникова. – Москва: Триумф, 2006.— 464 с.

3. Сафронов М.Н. Разработка приложений в Yii 2/М.Н. Сафронов.– Москва: ДМК Пресс , 2015.–115с.

4. Макконелл С. Совершенный код, 2-е издание/С. Макконелл.–Спб.: Питер, 2012.–68 с.

5. Веллинг Т. Разработка Web-приложений на PHP и MySQL / Т. Веллинг.–СПб.: ДиаСофтЮП, 2003. –672 с.

6. Байенс Дж. Примочки программирования в Web/ Дж. Байенс.– СПб.: Питер, 2001.– 440с.

7. Ригс С. Администрирование PostgreSQL 9. Книга рецептов/С. Ригс, Х. Кросинг.–Москва:ДМК Пресс, 2013.–246с.

8. Электронный ресурс документация по Yiiфреймворкhttp://www.yiiframework.com/doc/guide/1.1/ru.

9. Электронный ресурс документация PostgreSQLhttps://postgrespro.ru/blog/news/56838.

10. Электронный ресурсhttp://www.php.net/manual/ru/.

ПРИЛОЖЕНИЕ

Приложение 1.

Index.php

```
<?php
$this->title = 'Диплом';
?>
<div id="wrapper">
<div id="header">
<div id="logo">
<h1><a href="#">Фирма</a></h1>
<h2><span>ИП Бровков Г.Е.</span></h2>
</div>
<div id="menu">
<ul>
        <li class="active"><a href="#">Главная</a></li>
        <li><a href="http://test.dev/goods.php %2Findex ">Товары</a></li>
        <li><a href="http://test.dev/about.php? %2Findex ">О нас</a></li>
        <li><a href="http://test.dev/login.php? %2Findex >Авторизация</a></li>

</ul>
</div>
</div>
<div id="page">
<div id="page-bgtop">
<div id="content">
<div class="post">
<h2 class="title"><a>Оптоваяирозничнаяторговля</a></h2>
<div class="entry">
<p><imgsrc="images/img001.jpg" alt="" width="564" height="294" /></p>
<p> Друзья! Добро пожаловать на наш новый ресурс! С его помощью работать станет проще. Теперь мы можем отказаться от лишних бумаг, и вести учет нашей деятельности непосредственно здесь.База данных расположена на нашем сервере, и никто, кроме работников фирмы не будет иметь к ней доступ.</p>
</div>
</div>
<div class="post">
<h2 class="title"><a href="#">Пополнение ассортимента</a></h2>
<divclass="entry">
<p><imgsrc="images/2.jpg" alt="" width="564" height="294" /></p><p>Мы стараемся постоянно обновлять ассортимент товара.Вы всегда можете найти то , что нужно.Следите заобновлениями, чтобы быть в курсе событий!</p>
</div>
</div>
</div>
<divid="sidebar">
```

```
        <ul>
                <li>
                <h2>Данные о товарах</h2>
        <ul>
                <li><a href="http://test.dev/index.php?r=goods%2Findex">Товары</a></li>
                <li><a href="http://test.dev/index.php?r=kindofgoods%2Findex">Вид
товара</a></li>
                <li><a href="http://test.dev/index.php?r=supply%2Findex">Поставки</a></li>

        </ul>
        </li>
                <li>
                <h2>Финансы</h2>
        <ul>
                <li><a href="http://test.dev/index.php?r=clients%2Findex">Клиенты</a></li>
                <li><ahref="http://test.dev/index.php?r=providers%2Findex">Поставщики</a>
        </li>
                <li><a href="http://test.dev/index.php?r=kindofcosts%2Findex">Виды
расходов</a></li>
                <li><a href="http://test.dev/index.php?r=costs%2Findex">Расходы</a></li>
                <li><a href="http://test.dev/index.php?r=kindofearnings%2Findex">Виды
доходов</a></li>
                <li><a href="http://test.dev/index.php?r=revenue%2Findex">Выручка</a></li>
                <li><a href="http://test.dev/index.php?r=sale%2Findex">Продажи</a></li>
        </ul>
                </li>
                <li>
                <h2>Данные сотрудников </h2>
        <ul>
                <li><a
href="http://test.dev/index.php?r=departments%2Findex">Отделы</a></li>
                <li><a href="http://test.dev/index.php?r=labordata%2Findex">Трудовые
данные</a></li>
                <li><a href="http://test.dev/index.php?r=personaldata%2Findex">Личные
данные</a></li>
                <li><a href="http://test.dev/index.php?r=salary%2Findex">Зарплата</a></li>
                <li><a href="http://test.dev/index.php?r=schedule%2Findex">График
работы</a></li>
                <li><a
href="http://test.dev/index.php?r=positions%2Findex">Должности</a></li>
                </ul>
        </li>
        </div>
        <div style="clear: both; height: 1px"></div>
        </div>
        </div>
<divid="footer">
        <pid="legal">Выполниластудентка 4 курсагруппы 07001301
КурбатоваЛариса</a>.</p>
        <p id="links">г.Белгород, 2017 год</a></p>
        </div>
        </div>
```

Приложение 2.

Main.php

```
<?php
useyii\helpers\Html;
useyii\bootstrap\Nav;
useyii\bootstrap\NavBar;
useyii\widgets\Breadcrumbs;
use app\assets\AppAsset;
AppAsset::register($this);
?>
<?php $this->beginPage() ?>
<!DOCTYPE html>
<html lang="<?=Yii::$app->language ?>">
<head>
<meta charset="<?=Yii::$app->charset ?>">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="keywords" content="" />
<meta name="description" content="" />
<?= Html::csrfMetaTags() ?>
<title><?= Html::encode($this->title) ?></title>
          <link href="/default.css" rel="stylesheet" type="text/css" />
<?php $this->head() ?>
</head>
<body>
<?php $this->beginBody() ?>
<div class="wrap">
<?php
NavBar::begin([
'brandLabel' => 'ФирмаИПБровковГ.Е.',
       'brandUrl' =>Yii::$app->homeUrl,
       'options' => [
          'class' => 'navbar-inverse navbar-fixed-top',
       ],
   ]);
echoNav::widget([
       'options' => ['class' => 'navbar-navnavbar-right'],
       'items' => [
['label' => 'Главная', 'url' => ['/site/index']],
['label' => 'Товары', 'url' => ['/site/about']],
['label' => 'Онас', 'url' => ['/site/contact']],
Yii::$app->user->isGuest ? (
['label' => 'Авторизация', 'url' => ['/site/login']]
          ) : (
             '<li>'
             . Html::beginForm(['/site/logout'], 'post')
             . Html::submitButton(
                'Logout (' .Yii::$app->user->identity->username . ')',
['class' => 'btnbtn-link logout']
             )
```

```
                    . Html::endForm()
                    . '</li>'
                )
            ],
        ]);
    NavBar::end();
?>
    <div class="container">
    <?= Breadcrumbs::widget([
            'links' =>isset($this->params['breadcrumbs']) ? $this->params['breadcrumbs'] : [],
        ]) ?>
    <?= $content ?>
    </div>
    </div>
    </body>
    </html>
    <?php $this->endPage() ?>
```

Приложение 3.

Login.php

```
    <?php
    useyii\helpers\Html;
    useyii\bootstrap\ActiveForm;

    $this->title = 'Login';
    $this->params['breadcrumbs'][] = $this->title;
    ?>
    <div class="site-login">
    <h1><?= Html::encode($this->title) ?></h1>
    <?php $form = ActiveForm::begin([
        'id' => 'login-form',
        'layout' => 'horizontal',
        'fieldConfig' => [
            'template' => "{label}\n<div class=\"col-lg-3\">{input}</div>\n<div class=\"col-
lg-8\">{error}</div>",
            'labelOptions' => ['class' => 'col-lg-1 control-label'],
        ],
    ]); ?>
    <?= $form->field($model, 'username')->textInput(['autofocus' => true]) ?>
    <?= $form->field($model, 'password')->passwordInput() ?>
    <?= $form->field($model, 'rememberMe')->checkbox([
            'template' => "<div class=\"col-lg-offset-1 col-lg-3\">{input}
{label}</div>\n<div class=\"col-lg-8\">{error}</div>",
        ]) ?>
    <div class="form-group">
    <div class="col-lg-offset-1 col-lg-11">
    <?= Html::submitButton('Login', ['class' => 'btnbtn-primary', 'name' => 'login-button'])
?>
```

```
</div>
</div>

<?phpActiveForm::end(); ?>
<div class="col-lg-offset-1" style="color:#999;">
</div>
</div>
```

Приложение 4.

Db.php

```
<?php
return [
'class' => 'yii\db\Connection',
        'dsn' => 'pgsql:host=localhost;port=5432;dbname=test',
    'username' => 'postgres',
    'password' => '123',
    'charset' => 'utf8',
];
```

Приложение 5.

SiteController.php

```
<?php
namespace app\controllers;
useYii;
useyii\filters\AccessControl;
useyii\web\Controller;
useyii\filters\VerbFilter;
use app\models\LoginForm;
use app\models\ContactForm;
use app\models\Chet;
classSiteController extends Controller
{
public function behaviors()
    {
return [
        'access' => [
            'class' =>AccessControl::className(),
            'only' => ['logout'],
            'rules' => [
                [
                    'actions' => ['logout'],
                    'allow' => true,
                    'roles' => ['@'],
                ],
            ],
```

```
                ],
                'verbs' => [
                    'class' =>VerbFilter::className(),
                    'actions' => [
                        'logout' => ['post'],], ], ]; }
        public function actions()
            {
        return [
                'error' => [
                    'class' => 'yii\web\ErrorAction',
                ],
                'captcha' => [
                    'class' => 'yii\captcha\CaptchaAction',
                    'fixedVerifyCode' => YII_ENV_TEST ? 'testme' : null,
                ],
            ];
            }
public function actionIndex()
            {
        return $this->render('index');
            }
public function actionLogin()
            {
        if (!Yii::$app->user->isGuest) {
        return $this->goHome();
                }
                $model = new LoginForm();
        if ($model->load(Yii::$app->request->post()) && $model->login()) {
        return $this->goBack();
                }
        return $this->render('login', [
                    'model' => $model,
                ]);
            }
public function actionLogout()
            {
        Yii::$app->user->logout();

        return $this->goHome();
            }
public function actionContact()
            {
                $model = new ContactForm();
        if ($model->load(Yii::$app->request->post()) && $model->contact(Yii::$app-
>params['adminEmail'])) {
        Yii::$app->session->setFlash('contactFormSubmitted');

        return $this->refresh();
                }
        return $this->render('contact', [
                    'model' => $model,
                ]);
```

```
    }

public function actionAbout()
    {
return $this->render('about');
    }
}
public function actionChet()
{
    $model = new app\models\Chet();

if ($model->load(Yii::$app->request->post())) {
if ($model->validate()) {
            // form inputs are valid, do something here
return;
        }
    }

return $this->render('chet', [
        'model' => $model,
    ]);
}
```

Приложение 6.

Clients.php

```
<?php
namespace app\models;
useYii;
useyii\base\Model;
useyii\data\ActiveDataProvider;
use app\models\Clients;
classClientsSearch extends Clients
{
public function rules()
    {
return [
            [['id'], 'integer'],
            [['surname', 'name', 'middlename', 'email'], 'safe'],
        ];
    }
function scenarios()
    {
        // bypass scenarios() implementation in the parent class
return Model::scenarios();
    }
public function search($params)
    {
        $query = Clients::find();
```

```
        // add conditions that should always apply here
        $dataProvider = new ActiveDataProvider([
            'query' => $query,
        ]);
        $this->load($params);
    if (!$this->validate()) {
            // $query->where('0=1');
    return $dataProvider;
        }
        // grid filtering conditions
        $query->andFilterWhere([
            'id' => $this->id,
        ]);
        $query->andFilterWhere(['like', 'surname', $this->surname])
            ->andFilterWhere(['like', 'name', $this->name])
            ->andFilterWhere(['like', 'middlename', $this->middlename])
            ->andFilterWhere(['like', 'email', $this->email]);
    return $dataProvider;
      }
    }
```

Приложение7.

Clients/index.php

```
    <?php
    useyii\helpers\Html;
    useyii\grid\GridView;
    $this->title = 'Clients';
    //$this->params['breadcrumbs'][] = $this->title;
    ?>
    <div class="clients-index">
    <h1><?= Html::encode($this->title) ?></h1>
    <?php // echo $this->render('_search', ['model' => $searchModel]); ?>
    <p>
    <?= Html::a('Create Clients', ['create'], ['class' => 'btnbtn-success']) ?>
    </p>
    <?=GridView::widget([
        'dataProvider' => $dataProvider,
        'filterModel' => $searchModel,
        'columns' => [
    ['class' => 'yii\grid\SerialColumn'],
            'id',
            'surname',
            'name',
            'middlename',
            'email:email',

        [ 'class' =>yii\grid\ActionColumn::className(),
'contentOptions' => ['class' => 'action-column'],
```

```
                'buttons' => [
                'view' => function ($url, $model, $key) {
                return Html::a('View', $url);
                },
                'update' => function ($url, $model, $key) {
                return Html::a('Update', $url);},
                'delete' => function ($url, $model, $key) {
                return Html::a('Delete', $url);
                },],
                ],
        ]]); ?>
    </div>
```

Приложение 8.

UploadedFile.Php

```
        namespace app\models;
            use yii\base\Model;
            use yii\web\UploadedFile
           class UploadForm extends Model
    {
     public $imageFile
            public function rules()
             {
            return [
            [['imageFile'], 'file', 'skipOnEmpty' => false, 'extensions' => 'png, jpg'],
          ];
       }
       public function upload()
       {
          if ($this->validate()) {
             $this->imageFile->saveAs('uploads/' . $this->imageFile->baseName . '.' . $this-
    >imageFile->extension);
             return true;
          } else {
             return false;
          }
       }
class SiteController extends Controller
    {
       public function actionUpload()
       {
          $model = new UploadForm();

          if (Yii::$app->request->isPost) {
             $model->imageFile = UploadedFile::getInstance($model, 'imageFile');
             if ($model->upload()) {
                // file is uploaded successfully
```

```
            return;
        }
    return $this->render('upload', ['model' => $model]);
  }
}
```

Приложение9.

UserRoleRule.php

```
<?php
namespaceconsole\controllers;
useYii;
useyii\console\Controller;
usecommon\components\rbac\UserRoleRule;
classRbacControllerextendsController
{
publicfunctionactionInit()
{
$auth = Yii::$app->authManager;
$auth->removeAll();
$dashboard = $auth->createPermission('dashboard');
$dashboard->description = Управление';
$auth->add($dashboard);
$rule = newUserRoleRule();
$auth->add($rule);
$user = $auth->createRole('user');
$user->description = 'Пользователь';
$user->ruleName = $rule->name;
$auth->add($user);
$moder = $auth->createRole('tovaroved');
$moder->description = 'Товаровед';
$moder->ruleName = $rule->name;
$auth->add($moder);
$auth->addChild($user,$tovaroved);
$admin = $auth->createRole('admin');
$admin->description = 'Администратор';
$admin->ruleName = $rule->name;
$auth->add($admin);
$auth->addChild($admin, $user);
   }
}
publicfunctionbehaviors()
{
return [
'access' => [
'class' =>AccessControl::className(),
'rules' => [
              [
'allow' =>true,
```

```
'actions'=>['login','error'],
'roles' => ['?'],
                ],
                [
'allow' =>true,
'roles' => ['moder'],
                ],
            ],
publicfunctiongetUser()
{
if ($this->_user === false) {
$this->_user = User::find()
->andWhere(['or', ['username' =>$this->username],
->one();
if (!Yii::$app->user->can('dashboard', ['user' =>$this->_user])) {
$this->_user = null;
        }
    }
return$this->_user;
}
```