Выбор баз данных.

Сначала разберёмся с блоком «каталог товаров».

Условия, относящиеся к созданию данного блока в бд:

1. Каталог товаров организован в древовидную структуру: группа -> категория -> подкатегория.
2. У каждого элемента структуры есть название, изображение.
3. Товар находится на уровне подкатегории. Причем, он может находиться только в одной подкатегории.
4. Подкатегория может принадлежать только одной категории. А категория, в свою очередь, только одной группе.
5. При создании товара обязательно указать его название, изображение, цену и количество данного товара на складах.
6. У товара могут быть дополнительные поля (вес, новый/б.у, длина, ширина и т.п.), причем, у разных товаров могут быть разные поля.
7. Данные о товарах обновляются не очень часто.
8. В каталоге товаров должен быть полнотекстовый поиск по товарам.
9. В разных городах цена на один и тот же товар может различаться.
10. Также есть жесткие требования ко времени отклика запросов к каталогу товаров.

Группа, категория и подкатегория - все они имеют предопределённые структурированные данные (название, изображение) и им нужна связанность, поэтому буду использовать реляционную бд, а в качестве СУБД возьму MySQL с высокой производительностью, надёжностью и безопасностью.

Названия изображений будут схожи с id категорий и их можно будет найти в директории (см. блок про структуру и хранение данных ниже), поэтому записывать в бд их не буду.

С товарами всё сложнее. У нас здесь есть несколько важных условий:

1) У разных товаров могут быть разные поля.
2) В каталоге товаров должен быть полнотекстовый поиск по товарам.
3) Данные о товарах обновляются не очень часто.
4) Также есть жесткие требования ко времени отклика запросов к каталогу товаров.

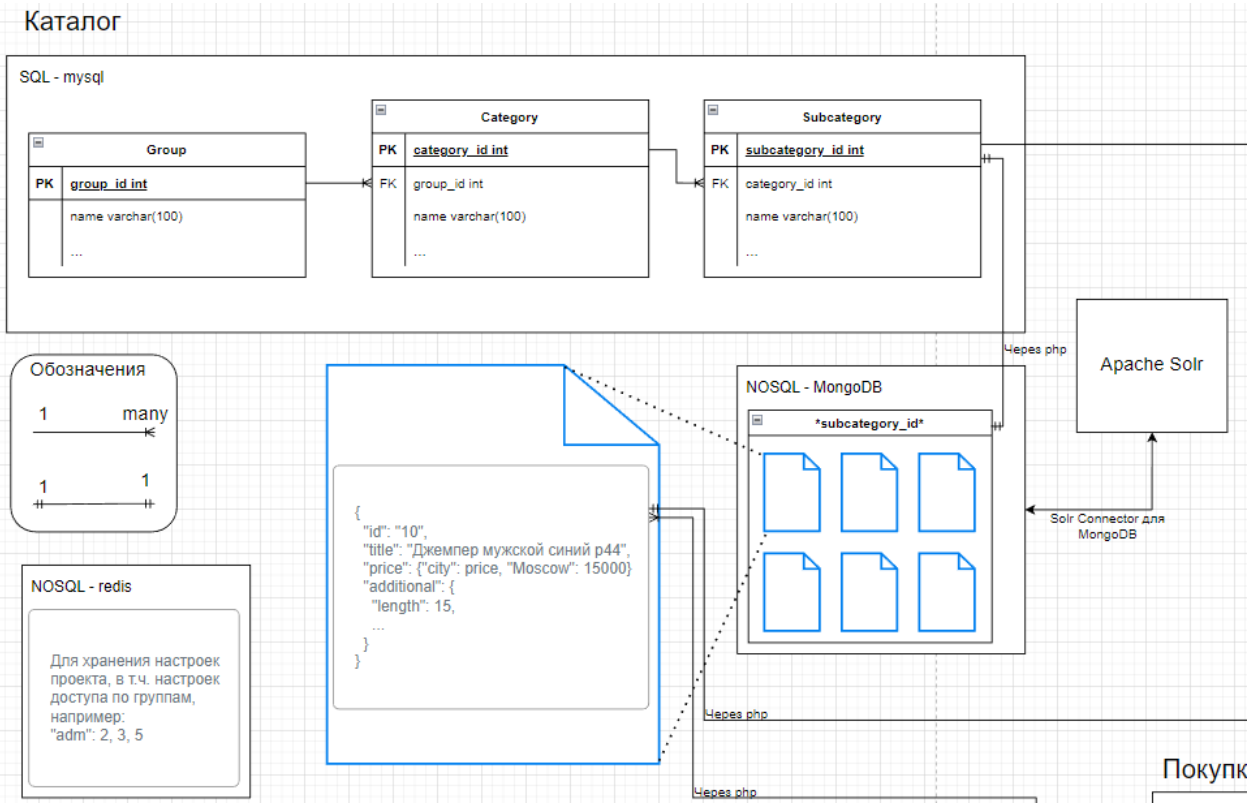
У нас есть 1 условие, которое говорит, что данные предугадать заранее невозможно, а значит использование реляционной бд не хорошая идея. Нужно использовать нереляционную бд, где мы сможем хранить неструктурированные данные – документоориентированная бд, например MongoDB. Так же этот вариант хорошо подходит под 3 условие. И так же MongoDB предоставляет высокую производительность и скорость чтения для выполнения пункта 4.

Для выполняется пункта 2 будем использовать Apache Solr, так как он быстрый, и имеет широкий набор функций для обработки и поиска данных.

В документах будут хранится такие данные как id товара, название, цена в зависимости от города (эти данные должны строго контролироваться на уровне php) и дополнительные опции. Название изображения будет такое же, как и id для экономии памяти, поэтому название изображения я не записываю.

Все документы будут разбиты на коллекции по подкатегориям (по их id). Для каждого документа (товара) уникален его id, вне зависимости от категории. Для этого можно создать генерацию id, например, по времени + id пользователя + что-то ещё. Сделано это для предотвращения возможной путаницы на складе или в корзине.

Концептуальная схема данной части:



Далее разберёмся с такими данными, как склады, города и пользователи.

Условия, относящиеся к созданию данного блока в бд:

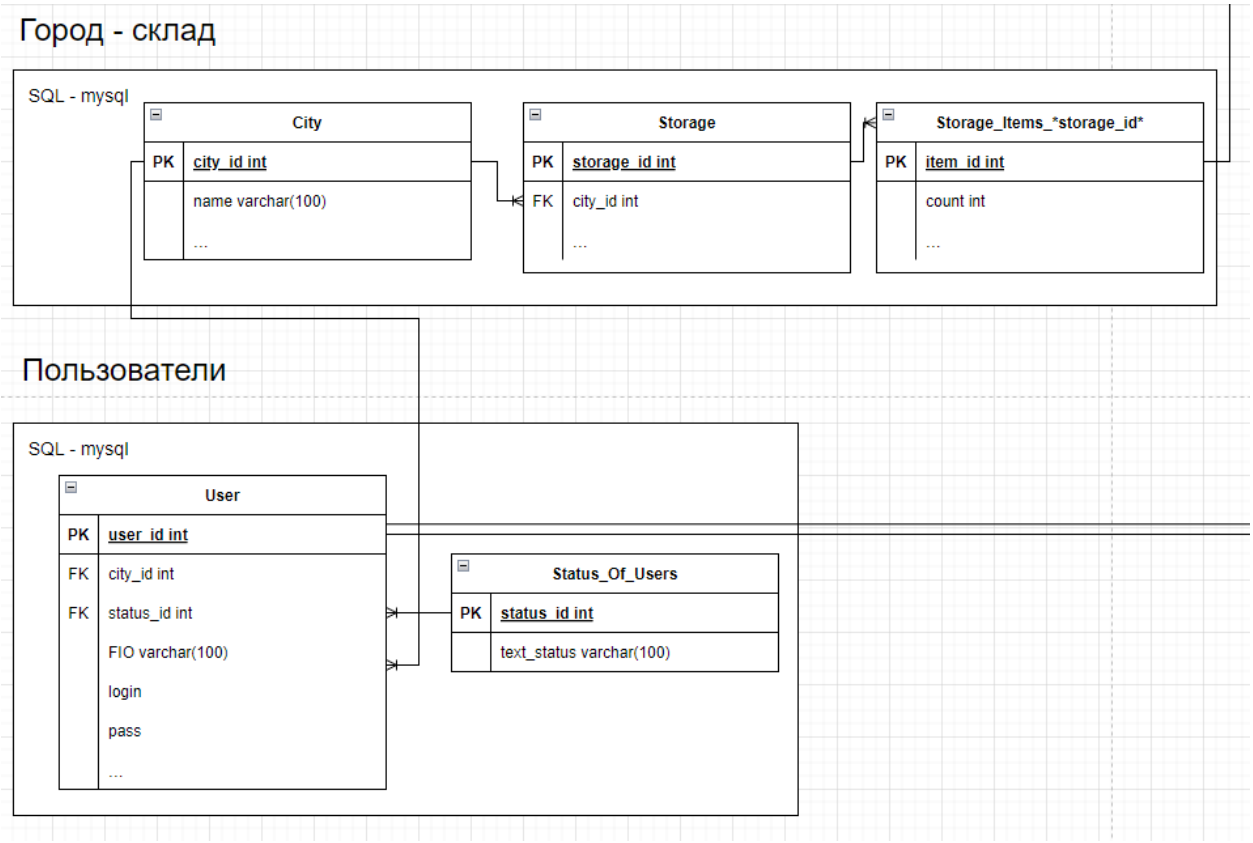
1. При входе в интернет-магазин, пользователя обязательно попросят выбрать город.
2. В каждом городе 1 и более складов. Один и тот же товар может быть в большом количестве на одном складе и отсутствовать на другом.

Города, склады - строго структурированные и очень связанные между собой данные, поэтому делаем их в SQL. Мы знаем, что на складах может быть много всего, поэтому стоит разделить эту таблицу на части и для каждого склада создавать свою таблицу с хранящимися у них товарами. Там же записывается кол-во этого товара на данном складе.

Расшифровку статуса (пользователь, администратор, контент-менеджер, …) буду хранить в sql, так как этот параметр не так часто запрашивается и нет необходимости в скорости его чтения, а вот в связанности есть, поэтому не выбрал для хранения redis.

Немного про выбор города. Этого нет ни на одной схеме, но описать работу нужно. При регистрации пользователь указывает город, который сохраняется в бд. Город так же хранится в куки для быстрого доступа. Каждый новый вход система ненавязчиво запрашивает у пользователя подтверждение местонахождения. Если пользователь проигнорировал окно, то город не меняется. Изменяется город через модуль “пользователь”.

Концептуальная схема данной части:



И разберёмся с корзиной и самим заказом.

Одним из условий было:

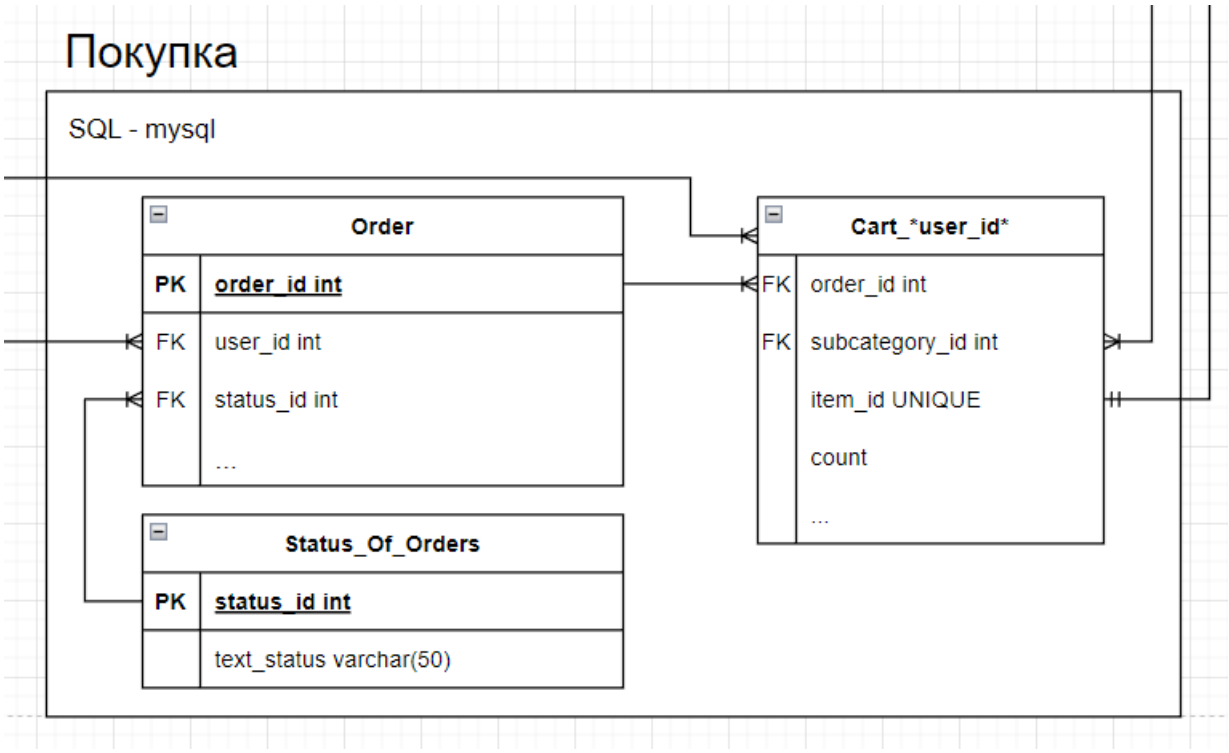
1. Товары можно положить в "Корзину" и потом оформить доставку.

Предполагаю, что это значит оформить заявку даже через несколько дней, или даже месяцев. Другими словами, корзина становится неким хранилищем выбранных товаров. Можно было бы реализовать это через куки, но нам, нужен доступ к одной и той же корзине с разных устройств. Можно было бы использовать redis для этого, ведь, в принципе, всё что нам нужно хранить в корзине пользователя – id добавленных товаров и их количество. Для этого подходят хеш-таблицы редиса, но у него есть минус: данные хранятся в оперативной памяти. Скорость, согласно заданию, здесь не так важна (нет условий про скорость в этой части), а вот данные в корзине могут хранится в относительно долгой перспективе, и если все пользователи добавят несколько товаров с свою корзину, то это может в худшем случае перегрузить оперативную память и вызвать сбои сервера, чего стоит избегать. Если же ограничить время хранения товаров в корзине разумным сроком, то можно использовать redis, но по данному (1) условию срок нельзя предугадать. Поэтому, для надёжности, буду использовать sql бд.

Записи о заказах появляются при добавлении первого товара в корзину. Товары в корзине хранятся в отдельной таблице. Так как одновременно в этой таблице может быть очень много записей, то следует разделить её и создавать для каждого пользователя свою таблицу в момент добавления первого товара и удалять через какое-то время после удаления последнего товара из неё, например через неделю (чтобы часто не пересоздавать таблицу корзины), но можно не удалять, а хранить в этой таблице историю заказов, так как к каждому продукту в корзине добавляется id заказа. Так же в неё записывается подкатегория товара, чтобы ускорить его поиск среди всех товаров, ограничившись коллекцией.

Расшифровку статуса буду хранить в sql, так как этот параметр не так часто запрашивается и нет необходимости в скорости его чтения, а вот в связанности есть, поэтому не выбрал для хранения redis.

Концептуальная схема данной части:



Для хранения настроек доступа и других настроек будем использовать redis, так как этих настроек не так много, представляются они в виде ключ – значение, а доступ к нм нужен быстый.

Финальный вид бд можно посмотреть в файле “BDandMore”

Где будет храниться медиаконтент.

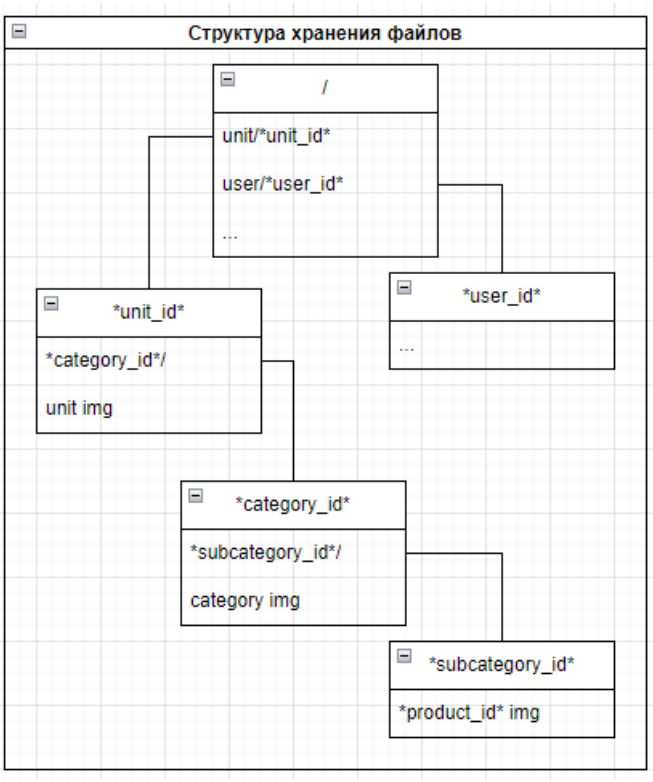
Медиафайлы можно хранить на том же сервере, где находится сам проект. У этого есть свой ряд +:

- Простота управления и обслуживания.
- Снижение нагрузки на сеть.

Но минусы слишком велики: ограниченный объем хранилища и ухудшение производительности сервера. Этот способ хранения подходит для маленьких проектов, но данный ожидается большим.

Поэтому медиафайлы будем хранить на отдельном специализированном для хранения файлов FTP сервере. У этого есть свои минусы: дополнительные расходы, требуется обсуживать больше серверов, но плюсов больше: производительность, безопасность.

Файловая структура будет древовидной, для увеличения скорости поиска нужного файла на сервере и удобства использования. Концептуальная схема данной части:



Финальный вид структуры можно посмотреть в файле “BDandMore”

Схема, со всеми компонентами системы и их взаимодействием.

Система будет иметь модульную структуру - для простоты разработки и поддержания системы выберем вариант реализации системы на основе паттерна MVP (Model-View-Presenter, на схеме контроллер выполняет функции “Presenter”).

PHP код будет исполнятся на apache сервере, так как он надёжен, гибок, обеспечивает полную поддержку php, позволяя запускать php скрипты непосредственно на сервере, что повышает производительность приложений. Но Apache тяжёлый и может не справится с большой нагрузкой. Для предотвращения частых падений сервера из-за нагрузки, будем использовать несколько apache серверов. Предавать трафик на них мы будем через nginx (так как он имеет высокую производительность), являющийся в этот схеме обратным прокси и выполняющим задачу балансировщика нагрузки. Так же на nginx сервере будем хранить статический контент для быстрой выдачи пользователю, например, css файлы и т.п.

Единой точкой входа в приложение будет роутер. Его задача – определить в какой модуль отправить запрос и проверить доступ к самому модулю. Например, в админскую часть на уровне роутера не пустят пользователя без нужного статуса и, соответственно, не пропустят никакие админские действия. Если нужен запрет в самом модуле, то есть файл “access” с подключённым бд редис, в которой можно запретить определённые действия для групп пользователей или вывод определённой информации, например запретить оформлять заказ незарегистрированному пользователю и не выводить саму кнопку заказа (при возможности просмотра товара) или разрешить менять товары только контент-менеджерам через админ панель.

После пользователь попадает в контроллер, который, на основе запроса из адресной строки решает, что нужно сделать. Выбрав действие, он начинает взаимодействие с моделью, после которого возвращает результат браузеру.

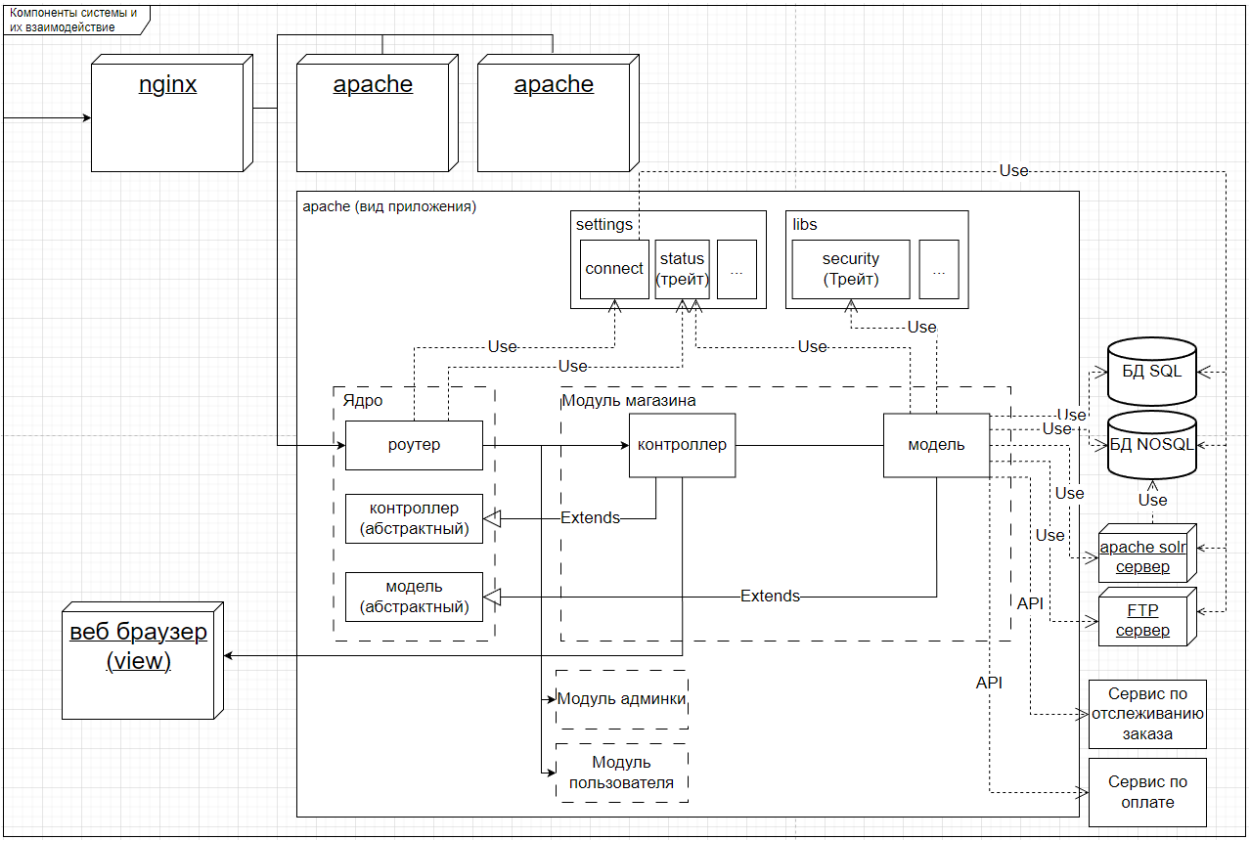
Модель же отвечает за всю бизнес-логику и за все внешние связи. Именно она занимается обработкой данных.

Представление отдам на сторону клиента. Можно это сделать при помощи React.

Взаимодействие между фронтендом и бекендом буду осуществлять при помощи Ajax или параметров в URL. Соответственно сервер будет только обрабатывать запрос и высылать ответ в виде JSON.

Немного про выбор города. Изменяется город через модуль “пользователь”.

Финальный вид системы можно посмотреть в файле “BDandMore”



Реализовать прототип API по оформлению заказа.

Под оформлением заказа я понимаю процесс от выбора продукта, до отправки данных в логистическую компанию. Поэтому реализован только модуль магазина с необходимыми полями и методами: показ юнитов, типов, подтипов, работа с корзиной (удаление, добавление товара), работа с заказом (создание, изменение статуса, проверка, оплата, связь с логистом).

Такая функциональность, как авторизация, регистрация, и прочие методы, связанные с функционированием учётной записи пользователя находятся в модуле user и здесь не реализованы.

Такая функциональность, как изменение товаров, и прочие методы, связанные с функционированием администрации находятся в модуле adm и здесь не реализованы.

Проверка доступа осуществляется из редиса на уровне роутера (доступ к модулю) и на уровне модели (доступ к действиям и данным в модуле).

Реализацию API по оформлению заказа можно посмотреть в директории “\task-vk1\ example” (точка входа: index.php)