Архитектура системы приёма электронных денег

Максим Бабич, 2009 г.

Вступление. Куй железо, не отходя от кассы.

Для многих проектов наступает момент, когда хочется, чтобы сайт был не только визиткой в интернете, но и приносил прибыль. И не только в виде оплаты рекламных баннеров или контекстной рекламы, но и в виде денег от своих посетителей.

Неважно, что им предложить: крутой спецэффект на фотографию-аватар, футболку с символикой проекта или доступ к приватным сообщениям первой красавицы сайта. Важно, как получить за это деньги. Причём, желательно сразу, пока человек не передумал потратить свои кровные.

Такое ограничение сразу приводит к вычёркиванию из списка методов оплаты заполнение квитанции в Сбербанке или поход к платёжному терминалу. Да, это тоже достойные методы, но методы небыстрые. Особенно, если на дворе поздний вечер, потенциальный клиент расслабился за бутылкой пива или рюмкой чая. Какой Сбербанк, тёпленьким его нужно брать, тёпленьким! А то может и передумать, пока сходит с квитанцией в банк, мужественно выстоит очередь среди старушек-пенсионерок или найдёт платёжный терминал с минимальной комиссией.

Какие же способы организации приёма моментальных платежей приходят на ум в первую очередь? Их, как минимум, несколько:

1. смс
2. банковские карты
3. интернет-банкинг
4. электронные деньги

У каждого метода свои достоинства и свои недостатки:

Платежи через смс — большая комиссия, но зато мобильник почти у каждого посетителя сайта.

Банковские карты — комиссия минимальная, но за пределами крупных городов ими мало кто пользуется для совершения покупок в интернете.

Интернет-банкинг ещё менее распространён среди населения, чем банковские карты, хотя для кого-то даже удобнее.

Электронные деньги — множество разных систем, технические и юридические трудности подключения, зато и аудитория большая! А уж сколько способов пополнения своего счёта!

Электронные деньги всё плотнее входят в жизнь пользователя интернета. Понимают это и крупные компании как российской, так и зарубежной интернет-индустрии. Так, в конце 2009 года компания PayPal открыла портал для разработчиков (https://www.x.com/), предоставив им инструменты для интеграции с этой крупной платёжной системой. В Рунете в 2009 году компания

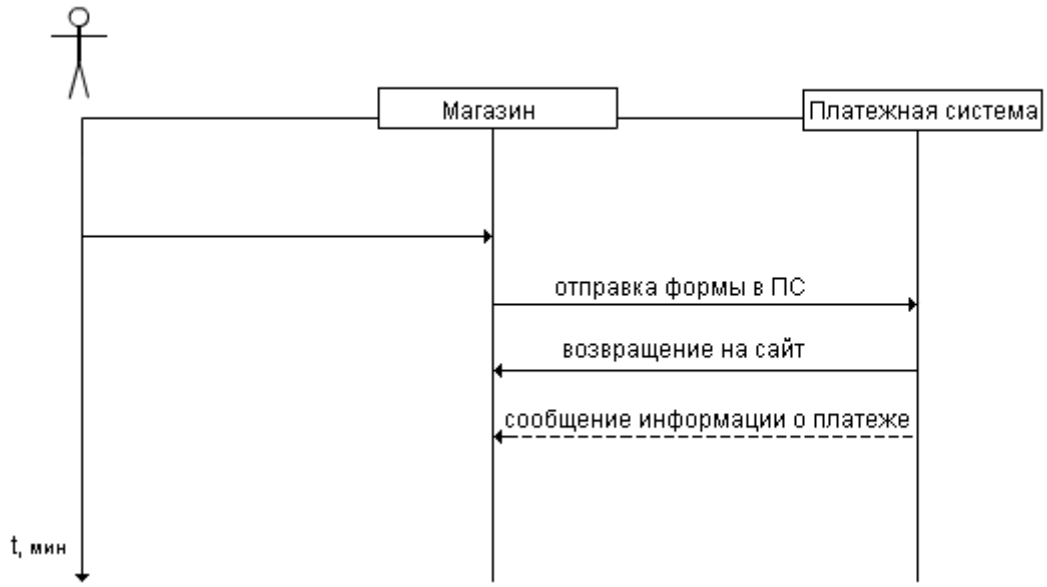
«Мэйл.Ру» — один из ключевых игроков отрасли — запустила собственную платёжную систему «Деньги@Mail.Ru» (https://money.mail.ru/), интегрировав её со своими ключевыми проектами.

Попробуем остановиться на последнем методе приёма денег за товары или услуги сайта и поговорим подробнее о механизмах работы и архитектуре приложения, которое позволит принимать на сайте электронные деньги, такие как WebMoney, «Яндекс.Деньги» или вышеупомянутые «Деньги@Mail.Ru» и им подобные. Реализация работы с каждой конкретной платёжной системой нас не волнует — о тонкостях подключения можно почитать и в документации к этим платёжным системам, и в разных историях успеха или неудач.

Сценарии. Два сапога — пара.

Казалось бы, подключение каждой новой системы — действо нетривиальное, сопровождаемое плясками с бубном, и ни о какой универсальности речи быть не может. Однако, это не так — правильно спроектировав систему приёма платежей, подключить новую платёжную систему не составит никакого труда. Почему? Дело в том, что платёжные системы (по крайней мере, очень многие из них) делятся всего на два типа по методам оповещения сайта-продавца о принятии денег в его пользу.

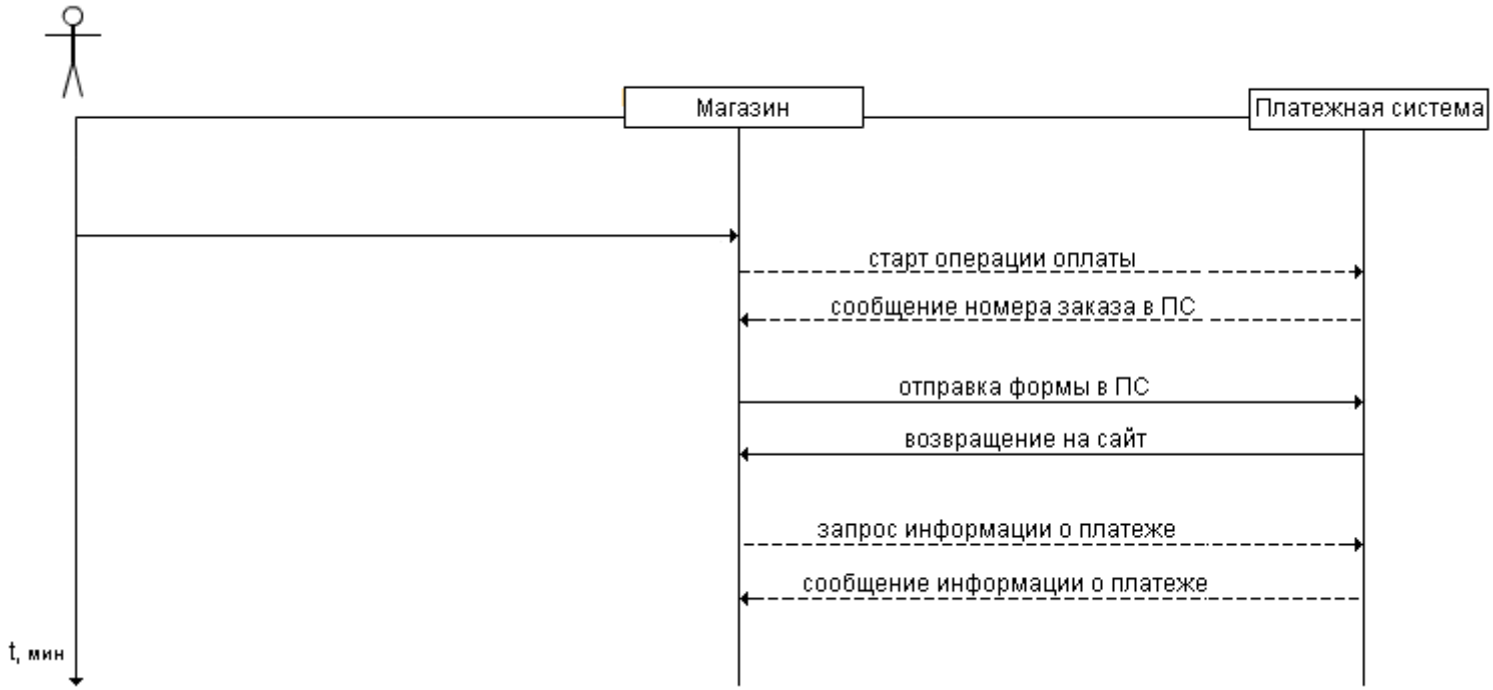
Первый тип платёжных систем самостоятельно сообщает продавцу о факте оплаты. Весь процесс оплаты выглядит следующим образом: посетитель сайта заполняет какую-то форму, где указывается помимо всего прочего и номер заказа в магазине. Форма вместе с покупателем отправляются на сайт платёжной системы, где пользователь оплачивает заказ (или отменяет его), после чего переадресовывается платёжной системой обратно на сайт продавца. После этого платёжная система в фоновом режиме отправляет информацию на заранее оговоренный адрес на сайте магазина, сообщая, что заказ с таким-то номером оплачен. Дополнительно система может поинтересоваться у сайта-продавца, готов ли он принять оплату по указанному номеру заказа. Эта фаза может называться Check, а фаза сообщения об оплате заказа — Pay.



Этот метод достаточно прост — нужно всего лишь правильно сгенерировать форму и корректно обработать дальнейшие запросы от платёжной системы.

Платёжные системы второго типа не сообщают продавцу о факте оплаты, но предоставляют инструменты для самостоятельного получения сайтом-продавцом информации о статусе платежа. Эти инструменты обычно называются программным интерфейсом или API. Схема взаимодействия с платёжной системой в этом случае немного изменится.

Типичный сценарий работы с такими системами с точки зрения продавца выглядит следующим образом. Пользователь стартует процесс проведения платежа, магазин сохраняет заявку на оплату, после чего формирует заявку к платёжной системе на совершение платежа, подписывая её специальным секретным ключом или сертификатом, предварительно полученным от платёжной системы. Затем в фоновом режиме сайт продавца отправляет эти данные в платёжную систему, которая в ответ сообщает ему номер заказа в их системе учёта. Номер сохраняется в системе учёта продавца. После этого пользователь перенаправляется для дальнейшей оплаты в платёжную систему, а продавец с некоторой периодичностью её опрашивает, уточняя статус заказа. После получения информации, что статус заказа изменился на «оплачен», продавец изменяет статус заявки в своей системе и оказывает услуги пользователю.



Этот сценарий работы несколько сложнее, но зато более универсальный. Создав приложение, которое умеет работать по этой схеме, можно научить его работать и с платёжными системами первого типа, ведь, как видно из схем, такие системы — частый случай систем второго типа.

Как правило, большинству сайтов достаточно реализации первого сценария, благо, что практически все платёжные системы умеют работать на его основе. Он прост и в то же время позволяет полноценно принимать платежи на сайте продавца. Но, как показывает практика, не все функции платёжной системы доступны таким образом. Многие из возможностей систем доступны исключительно через программный интерфейс к ним. И вот тут не обойтись без реализации схемы работы с платёжной системой по второму сценарию. Например, так работают с платёжными системами многие обменные пункты, позволяющие менять валюту одной платёжной системы на валюту другой.

Архитектура. Палка, палка, огуречик — получился человечек.

Сложно построить дом без фундамента, а систему приёма электронных денег без биллинга — для учёта заказов и платежей покупателя без него не обойтись. Как грамотно спроектировать биллинговую систему — речь для целой группы статей, поэтому сейчас не будем подробно на этом останавливаться. Тем более что для наших целей не нужно знаний о том, каким должен быть биллинг: он может быть сколь угодно сложным или простым. Нам от него понадобятся всего две вещи — сумма заказа и его номер (думаю, не стоит специально рассуждать, почему этот номер должен быть уникальным, правда?). Все данные о товаре (например, какое-то текстовое описание товара или услуги) — опциональны.

Итак, начнем с общего описания схемы нашей системы. Как это ни странно, рецепт архитектуры системы приёма платежей по описанным сценариям очень и очень прост. Нам потребуется:

- очередь заявок — 1 шт.
- диспетчер обработки очереди — 1 шт.
- общая библиотека работы с заявками — 1 шт.
- общая библиотека веб-интерфейса — 1 шт.
- общая библиотека работы с платёжными системами — 1 шт.
- библиотеки работы с конкретными платёжными системами — N штук, по числу систем.

Теперь подробнее о каждой из частей.

1. **Что такое очередь заявок?**

Очередь — это таблица заявок на оплату заказов. Каждая заявка может быть только одного статуса (иногда говорят «состояния»). В самом простом случае этих статусов всего три: заявка «*оплачена*», «*не оплачена*», «*в обработке*». Чем более точную информацию хочется получить о состоянии заказа, тем больше статусов может понадобиться. Например, могут добавиться статусы «*обрабатывается диспетчером*», «*при обработке заявки произошла ошибка*», «*заявка выполнена, но услуга ещё не оказана*» и так далее. Для обозначения статуса в очереди достаточно будет пары символов: двадцать шесть английских букв и десять цифр дадут более тысячи комбинаций, которых хватит на все случаи жизни.

Читатели, уже узнавшие выражения «реляционная база данных» и «нормализация», возразят, что таблиц должно быть больше, но я утверждаю, что достаточно одной. Описания подключенных платёжных систем, какие-то настройки, текстовые описания статусов можно хранить, например, в конфигурационном файле или как константы в одной из библиотек. В конце концов, все эти данные изменяются крайне редко, возможно ни разу не изменятся за всё время жизни проекта. Заводить ради них дополнительные таблицы в базе данных не нужно.

Какие ещё поля могут понадобиться в этой таблице?

Чтобы связать нашу заявку с платёжной системой, потребуется хранить соответствие между нашим номером и номером заказа (платежа) в платёжной системе. Как правило, такие номера — числа, однако бывают и достаточно экзотичные варианты. Поэтому для этого поля лучший тип данных — строка. Такой же тип можно выбрать для идентификатора платёжной

системы, которую покупатель указал как способ оплаты, и для адреса, на который надо перенаправить пользователя для продолжения оплаты — некоторые платёжные системы передают магазину и подобную информацию.

Не обойтись и без учёта дат. Как минимум, это дата создания и дата оплаты заявки. В идеальном варианте нужен и «срок годности заявки» — крайняя дата, когда ещё нужно пытаться что-то разузнать о статусе этой заявки в платёжной системе. Если не учитывать этот срок, можно бесконечно долго опрашивать платёжную систему, надеясь получить от неё информацию, что заявка наконец-то оплачена. Хотя пользователь давным-давно мог отказаться от оплаты.

Ещё несколько важных полей — это ближайшая дата обработки заявки диспетчером и количество времени, на которое будет отложена обработка заявки в случае неудачи. Эти поля нужны, чтобы не пытаться постоянно обрабатывать заявку в случае, если её статус не изменился или произошла какая-то ошибка при попытке получить информацию о нём. Например, когда мы обратились за статусом заявки, сайт платёжной системы оказался недоступен. Вполне вероятно, что через 10-20 секунд сайт станет доступен, и нам стоит попытаться ещё раз узнать статус. Вот для этого мы изменим дату следующей обработки и увеличим промежуток между попытками. На какую величину увеличивать этот промежуток — дело ваше. Можно использовать сложные статистические расчёты, а можно просто увеличивать ровно в два раза, начиная с 30 секунд и заканчивая 2–3 часами.

2. Библиотека работы с заявками

Для работы с заявками из очереди потребуется всего несколько функций.

- создание заявки
- получение информации о заявке
- изменение статуса и других данных заявки
- получение заявки из очереди
- удаление заявки

Эта библиотека должна уметь работать с базой данных, поэтому одним из аргументов перечисленных функций будет дескриптор подключения к СУБД. В качестве результата выполнения функции может возвращаться структура данных, содержащая как информацию о заявке, так и данные об ошибках, которые произошли во время выполнения.

3. Веб-интерфейс

Для чего нужна эта часть системы, думаю, понятно без всяких объяснений. Основное её назначение — вывести покупателю информацию о заказе; сделать всё, чтобы покупатель смог нажать на кнопку «Оплатить»; передать информацию о действиях другим частям нашей системы.

От веб-интерфейса нужны тоже всего несколько функций:

- вызов функции создания заявки
- вызов функции получения данных о заявке
- вызов функции удаления заявки
- функция для приема запросов от платёжных систем и вызова нужных обработчиков (если работа с платёжной системой ведётся по первому сценарию, описанному выше)
- базовая функция обработки ошибок и вывода данных в нужном формате

Хорошо, если последняя указанная функция сможет отдавать одни и те же данные в нескольких форматах: как минимум HTML и JSON (или XML). JSON понадобится, чтобы использовать AJAX и не перезагружать странички. Но это на любителя, можно обойтись и без излишеств.

4. Библиотеки работы с конкретными платёжными системами

Специфика работы каждой платёжной системы своя, и бесполезно пытаться написать универсальную библиотеку для работы с ними. Скорее всего, это не получится, а если и получится, то она будет очень громоздкой, весьма сложной в разработке и поддержке. Поэтому про универсальность в данном случае лучше забыть и под каждую платёжную систему писать свою библиотеку. Впрочем, каждая библиотека должна выполнять похожие действия:

- формирование URL и данных для запроса на выполнение платежа в платёжной системе
- разбор ответа платёжной системы и возврат номера заказа (как в нумерации платёжной системы, так и в нашей собственной)
- формирование URL и данных для запроса статуса заявки
- разбор ответа платёжной системы и возврат статуса заказа

Данные из этих библиотек нужно возвращать в едином формате, чтобы их обработка, группировка и непосредственная передача в платёжную систему могли идти из единой точки.

5. Общая библиотека работы с платёжными системами

Этой единой точкой станет общая библиотека работы с платёжными системами. Именно она будет отвечать за работу с системами электронных платежей, в том числе понимать, какие конкретно функции вызывать, чтобы правильно сформировать запрос в платёжную систему или обработать ответ от неё. Если точнее, то функции этой библиотеки таковы:

- работа с общими настройками (например, информацией о подключенных платёжных системах)
- определение платёжной системы, с которой взаимодействуем в данный момент, и обращение к библиотеке, которая предназначена для работы с ней.
- получение от библиотеки работы с конкретной платёжной системой данных, необходимых для создания заявки.
- передача ответа платёжной системы в соответствующую библиотеку и получение от неё информации о статусе заявки

6. Диспетчер очереди заявок

Диспетчер — это некое приложение, работающее в фоновом режиме. Метод его запуска неважен. Диспетчер может запускаться по расписанию, а может работать как демон Unix или служба Windows. Количество копий диспетчера зависит от объёма заявок, обрабатываемых системой. Если их число измеряется десятками или сотнями в день, хватит и одного диспетчера. Если счёт идёт на тысячи и десятки тысяч — может понадобиться целая бригада роботов-обработчиков.

Задача диспетчера — обрабатывать очередь заявок, получать необходимые данные для взаимодействия с платёжными системами от общей библиотеки работы с ними, обмениваться информацией с платёжными системами и на основе обработки их ответов менять статусы заявок.

По сути, диспетчер работает практически со всеми частями нашей системы кроме веб-интерфейса, а также делает какие-то дополнительные вещи, например, пишет логи ответов от платёжных систем.

Логи нужно записывать как можно более полно, ведь они — единственный способ узнать, что происходило в процессе обмена данными с платёжной системой, какая информация была передана и получена, какие ошибки произошли во время обмена. Чем больше вы знаете, тем проще разбираться в случае каких-то непредвиденных обстоятельств.

Снова сценарии. Ты — мне, я — тебе.

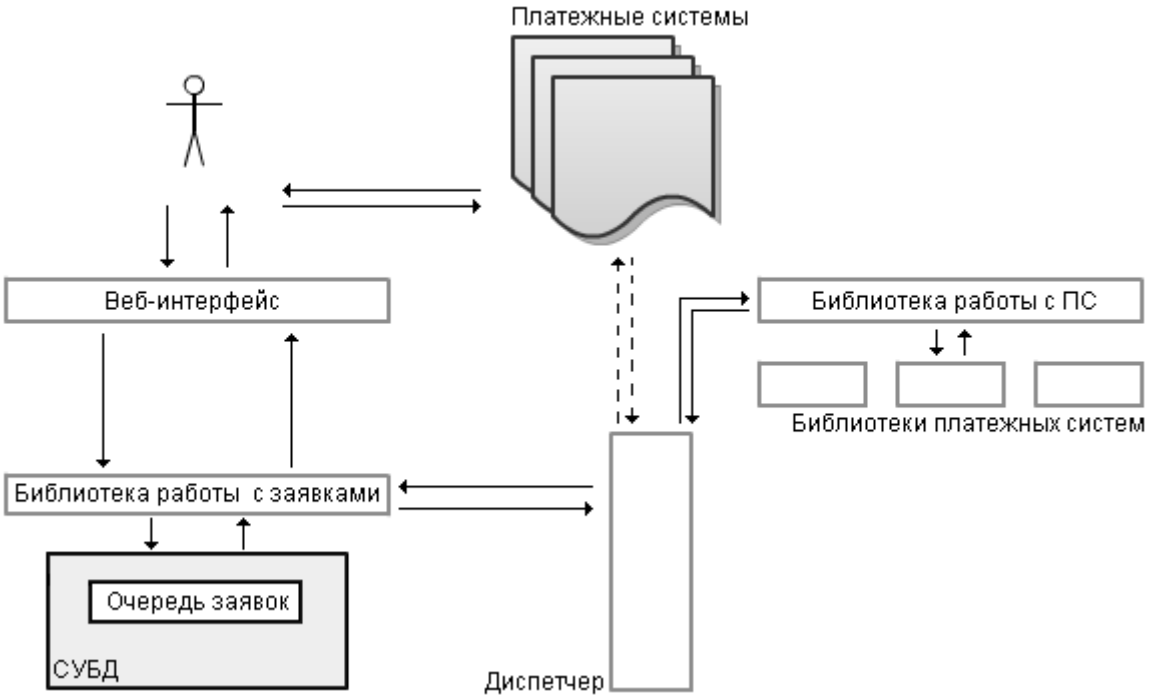
После рассказа о том, из каких частей состоит система, снова вернёмся к описанным сценариям работы с платёжными системами. Итак, пользователь зашёл на сайт, создал заказ и перешел к выбору платёжной системы, через которую готов платить.

Веб-интерфейс обращается к общей библиотеке платёжных систем, получает и выводит список доступных валют, а также информацию о платеже. Пользователь жмет «Оплатить», создается заявка со статусом «новая», информация о ней попадает в очередь. Пользователю в это время показываем сообщение «Подождите, через несколько секунд Вы будете перенаправлены на сайт платёжной системы». Вот тут-то и понадобятся данные о заявке в JSON-формате: пока пользователь на экране видит всякие часики и прочие похожие штучки, мы с помощью JavaScript опрашиваем очередь, узнавая статус заявки из JSON-данных ответа.

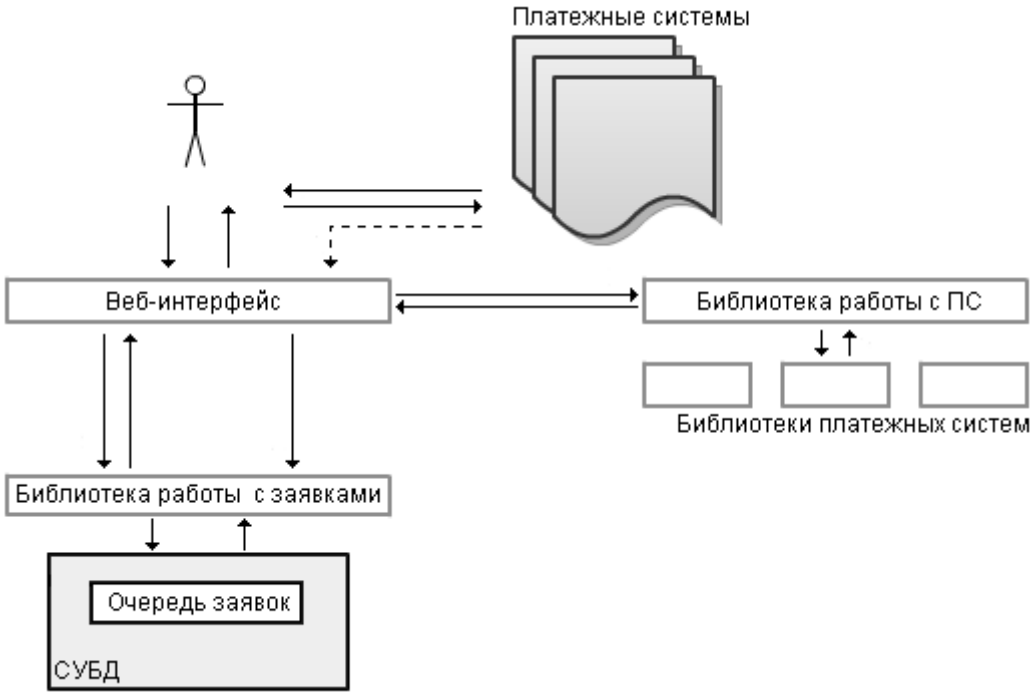
В это время диспетчер получает из очереди новую заявку, меняет ей статус на «заявка обрабатывается», формирует запрос в платёжную систему, подписывает его секретным ключом или сертификатом и отправляет в платёжную систему. В ответ получает номер заказа в платёжной системе и сохраняет номер в нашей очереди, присваивая заявке статус «заявка готова к оплате».

При очередном опросе очереди веб-интерфейс узнает, что можно продолжить оплату, и отправит пользователя в платёжную систему по тому адресу, который она нам сообщила или который мы сформировали на основе настроек библиотеки работы с этой платёжной системой. Что сделает пользователь дальше — его дело. Но на случай, если поймёт, что этой электронной валюты у него нет, и вернется назад, на странице со статусом заказа дадим ему ссылку на удаление заявки и создание новой (с обязательной возможностью выбрать другую электронную валюту).

Периодически диспетчер просматривает очередь заявок в статусе «готова к оплате» и проверяет статус в соответствующей платёжной системе. Если статус не изменился, откладывает заявку в сторону до лучших времен. Как только статус поменялся, диспетчер помечает заявку как «оплаченная» или «отклонённая». Больше она к диспетчеру не попадёт. Её обработкой займутся скрипты биллинга.



Это описание соответствует второму сценарию работы с платёжными системами — когда мы сами должны контролировать статус заявки в платёжной системе. В случае первого сценария всю работу за диспетчера сделает сама платёжная система: после того, как наш покупатель оплатит заказ, она обратится к нашему веб-интерфейсу и сообщит, что получена оплата по такой-то заявке. Веб-интерфейс передаст данные в общую библиотеку работы с платёжными системами, она с помощью других библиотек проверит корректность формата сообщаемой информации и через библиотеку работы с заявками поменяет статус той, данные о которой поступили от платёжной системы. После чего передаст веб-интерфейсу данные для ответа платёжной системе, что информация принята. Всё.



Итоги. Чем хороша описанная система?

1. **Система быстрая**

Описанная схема позволяет все операции с пользователем выполнять крайне быстро, ведь сохранение заявки в очередь или получение данных о заявке по уникальному номеру — очень быстрые операции. Вся медленная работа выполняется диспетчером в фоне.

2. **Хорошо масштабируется**

Мы можем организовать десятки очередей, используя различные методы деления на части (партиционирование) — от разделения по валюте до разделения по фазе луны в момент создания заказа. Для обработки каждой очереди можем запустить десятки и сотни демонов (конечно, если позволят ресурсы сервера).

3. **Проста**

Подключение новой платёжной системы — всего лишь добавление новой библиотеки, которая должна уметь выполнять всего несколько действий, описанных выше. Вся схема достаточно проста для реализации программистом практически любого уровня, а значит поддерживать её несложно.

Нигде в статье я не упомянул конкретных инструментов для реализации схемы. Просто потому, что их выбор непринципиален. Вы можете выбрать Perl или PHP, или Python для веб-интерфейса и библиотек, для работы с базой данных. Вы можете выбрать MySQL, Oracle, PostgreSQL или даже SQLite для организации очереди. Вы можете использовать LWP::UserAgent, CURL или даже php-шную функцию file_get_contents() для работы с платёжными системами через HTTP или HTTPS.

Выбор инструментов за вами. Главное, чтобы они помогли покупателю остаться довольным вашим сайтом и вашей системой приема электронных денег.