Практическая работа №5

Введение в многослойные клиент-серверные архитектуры

Цель работы:

Ознакомить с многослойными клиент-серверными архитектурами, посмотреть разницу между ними, выявить плюсы и минусы.

Теоретические сведения:

В предыдущих практических работах мы неявно познакомились с одной из архитектур распределенных систем: клиент-серверная архитектура.

Тонкие клиенты

Тонкий клиент спроектирован так, чтобы основная часть обработки данных происходила на сервере. Тонкий клиент как правило без жесткого диска: действуют как простой терминал к серверу и требует постоянной связи с сервером.

Толстые клиенты

Толстый клиент выполняет основную часть обработки. У толстых клиентов нет необходимости в непрерывной связи с сервером, поскольку они в основном передают информацию на сервер.

Когда какой клиент использовать

Тонкие клиенты обеспечивают работу рабочего стола в средах, где конечный пользователь имеет четко определенное и регулярное количество задач, для которых используется система. Тонких клиентов можно найти в медицинских офисах, авиабилетах, школах, правительствах, производственных предприятиях и даже колл-центрах. Наряду с простотой установки, тонкие клиенты также предлагают более низкую общую стоимость владения по сравнению с толстыми клиентами.

Если вашим приложениям требуются мультимедийные компоненты или которые интенсивно используют пропускную способность, стоит присмотреться к разработке толстых клиентов. Одно из самых больших преимуществ толстых клиентов – некоторые операционные системы и программное обеспечение не могут работать на тонких клиентах. Толстые клиенты могут справиться с ними, поскольку у них есть свои собственные ресурсы.

Разница между «уровнем» и «слоем» в контексте распределённых систем

«Слой» и «уровень» относятся к функциональной части программного обеспечения, но «уровень» относится к ПО, которое работает в инфраструктуре отдельных частей приложения (на разных физических системах). Например, приложение "Контакты" на телефоне

является трехслойным приложением, но одноуровневым приложением, потому что все три слоя работают на одном физическом устройстве – на телефоне. Так, «слои» не могут предложить тех преимуществ, которыми обладают «уровни».

Трёхуровневая архитектура распределённых систем

Трёхуровневая архитектура (рисунок 5.1) организует приложения в три логических и физических вычислительных уровня: уровень представления (пользовательский интерфейс), уровень приложений (обработка данных происходит здесь) и уровень данных (хранение и управление данными). Поскольку каждый уровень работает в своей собственной инфраструктуре, он может разрабатываться одновременно отдельной командой разработчиков и может обновляться или масштабироваться по мере необходимости, не влияя на другие уровни, не говоря уже о реплицировании.

В трехуровневом приложении вся связь проходит через уровень приложения. Уровень представления и уровень данных не могут напрямую взаимодействовать друг с другом.

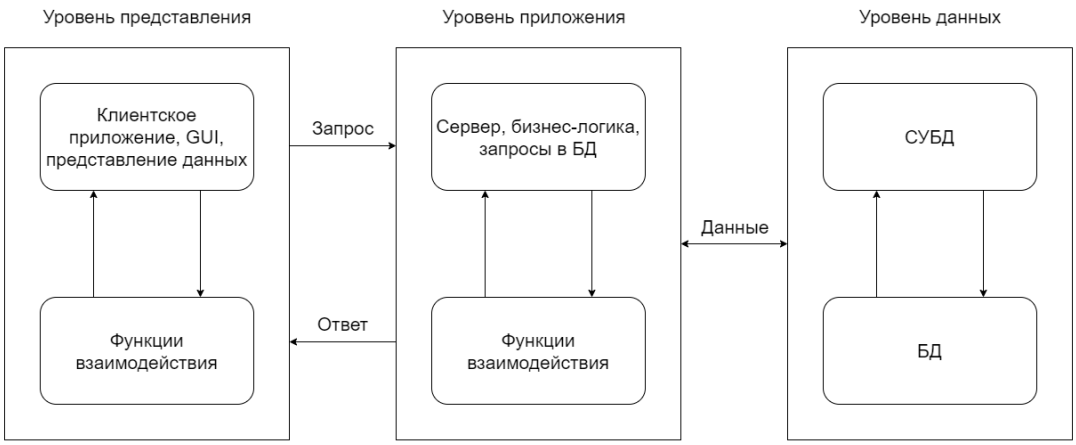

Рисунок 5.1 – Трёхуровневая архитектура

Уровень презентации

Это пользовательский интерфейс и слой связи приложения, на котором конечный пользователь взаимодействует с приложением. Его основная цель – отображать информацию для и собирать информацию от пользователя. Уровень презентации может работать в веб-браузере, в качестве настольного приложения или графического пользовательского интерфейса (GUI).

Уровень приложений

Уровень приложений, также известный как уровень логики или средний уровень, является сердцем приложения. На этом уровне информация, собранная на уровне представления, обрабатывается с использованием бизнес-логики. Уровень приложений также может добавлять, удалять или изменять данные на уровне данных.

Уровень данных

Уровень данных, иногда называемый уровнем базы данных, уровнем доступа к данным или серверной частью, – это место, где хранится и управляется информация, обрабатываемая приложением.

Зачем нужна трёхуровневая архитектура в распределённых системах

Как уже отмечалось ранее, главное преимущество трехуровневой архитектуры заключается в ее логическом и физическом разделении функций. Каждый уровень может работать на отдельной операционной системе и серверной платформе. Соответственно, не важно где будут расположены платформы: в облаке, в одном здании, на разных континентах. В любом случае части распределенной системы должны быть соединены надежными и защищенными линиями связи. Что касается скорости передачи данных, то она в значительной степени зависит от важности соединения между двумя частями системы с точки зрения обработки и передачи данных и в меньшей степени от их удаленности. Службы каждого уровня могут быть настроены и оптимизированы без влияния на другие уровни.

Другие преимущества:

- более быстрая разработка: отдельный уровень может разрабатываться одновременно разными командами, организация может быстрее выводить приложение на рынок, а программисты могут использовать новейшие и лучшие языки и инструменты в соответствии с каждым уровнем;
- улучшенная масштабируемость: отдельный уровень можно масштабировать независимо от других по мере необходимости;
- повышенная надежность: сбой на одном уровне с меньшей вероятностью повлияет на доступность или производительность других уровней;
- улучшенная безопасность: так как уровень представления и уровень данных не могут взаимодействовать напрямую, хорошо спроектированный уровень приложений может функционировать как своего рода внутренний брандмауэр, предотвращающий инъекции SQL и другие вредоносные эксплойты.

Пример: трёхуровневая архитектура в веб-разработке

В веб-разработке уровни имеют разные названия, но выполняют схожие функции:

- веб-сервер является уровнем представления и предоставляет пользовательский интерфейс. Обычно это веб-страница или веб-сайт, но никто не запрещает использование мобильных или декстопных приложений. Этот уровень по существу взаимодействует с клиентом, отображая данные различного рода. Лишь немногие из логики приложения выполняются на стороне клиента, имея ограниченные возможности;
- сервер приложений соответствует среднему уровню, в котором размещена бизнес-логика, используемая для обработки пользовательских

вводов. По сути, он играет роль объединения. При технической обработке различных входных данных и выборок, полученных клиентами, он играет роль взаимодействия с обширной базой данных, представленной на следующем уровне;

• сервер базы данных – это уровень данных или серверной части веб-приложения.

Взаимодействия между уровнями в распределенных системах

Существует два основных способа обмена данных между системами: синхронный способ и асинхронный (блокирующее и неблокирующее). Блокирующее – если участвующие стороны должны дождаться окончания взаимодействия, прежде чем приступить к следующей работе. Параллельная работа фрагментов систем никак не связана с асинхронностью и неблокирующими взаимодействиями. Сервер может работать параллельно со своими другими клиентами, обрабатывая запрос в синхронном режиме. Под синхронностью понимается работа запрашивающего клиента и процесса на сервере, который обрабатывает этот запрос. Синхронное взаимодействие – если работа клиента приостановлена, пока сервер обрабатывает запрос. Асинхронное взаимодействие – если вместо блокировки клиент выполняет другие действия после выдачи запроса.

Синхронное взаимодействие

При синхронном взаимодействии (рисунок 5.2) уровень приложения уверен, что уровень представления не изменит своего состояния и будет ожидать ответа столько времени, сколько потребуется на обработку. Одновременно с этим, пока уровень представления ожидает ответа на запрос, он простаивает, теряет время и, следовательно, производительность. В отдельных ситуациях процесс на уровне представлений может быть выгружен из оперативной памяти, и при ответе придётся загружать процесс из самой медленной памяти. Именно из-за уверенности в текущем состоянии уровня представлений, в подавляющем большинстве используется синхронное взаимодействие.

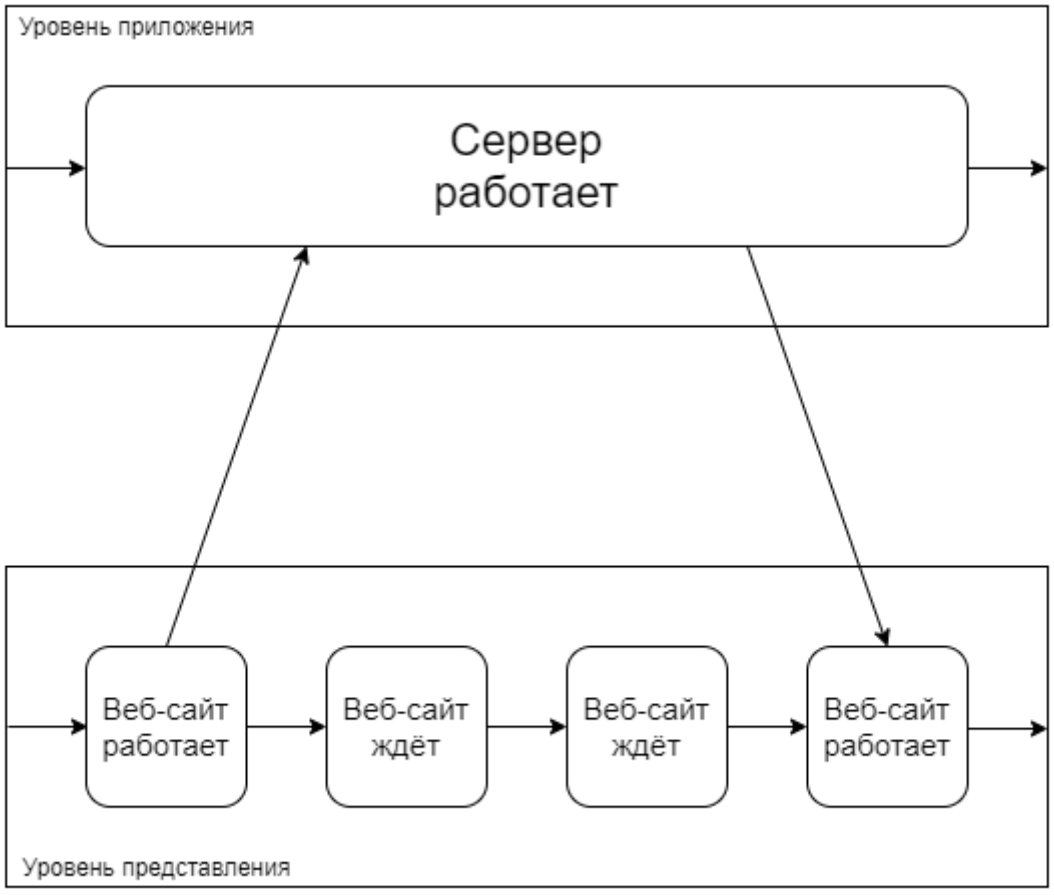

Рисунок 5.2 – Пример синхронного взаимодействия на основе веб-сайта и сервера

Асинхронное взаимодействие

При асинхронном способе обмена данных уровень представлений не ожидает запрос от уровня приложений, он отправляет некое сообщение и выполняет свою работу дальше. Через промежуток времени уровень представлений обращается к уровню приложений (или по определенному триггеру), чтобы узнать состояние своего сообщение. Если уровень приложений уже обработал сообщение, то при повторном запросе он отдаст ответ. Так асинхронное взаимодействие расширяет существующий функционал – появляется ещё два новых типа взаимодействия: сохранный и несохранный. При сохранной обработке сообщений появляется некая сущность между уровнем представления и уровнем приложения, так называемая очередь (рисунок 5.3, 5.4). Уровень представлений складывает свои сообщения в очередь. Уровень приложений проверяет эту очередь, достает сообщения, обрабатывает их и складывает во вторую очередь. Уровень представлений проверяет вторую очередь, если там есть ответ на определенный запрос, достает сообщение и обрабатывает его. При таком подходе уровни представления и приложения независимы друг от друга.

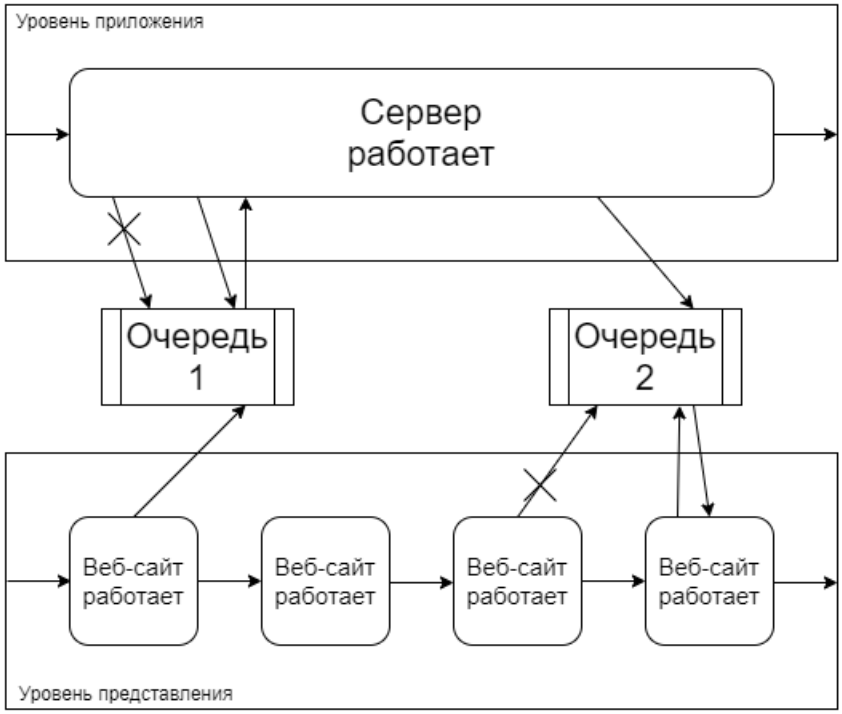

Рисунок 5.3 – Первый пример сохранного асинхронного обмена данными на основе веб-сайта и сервера

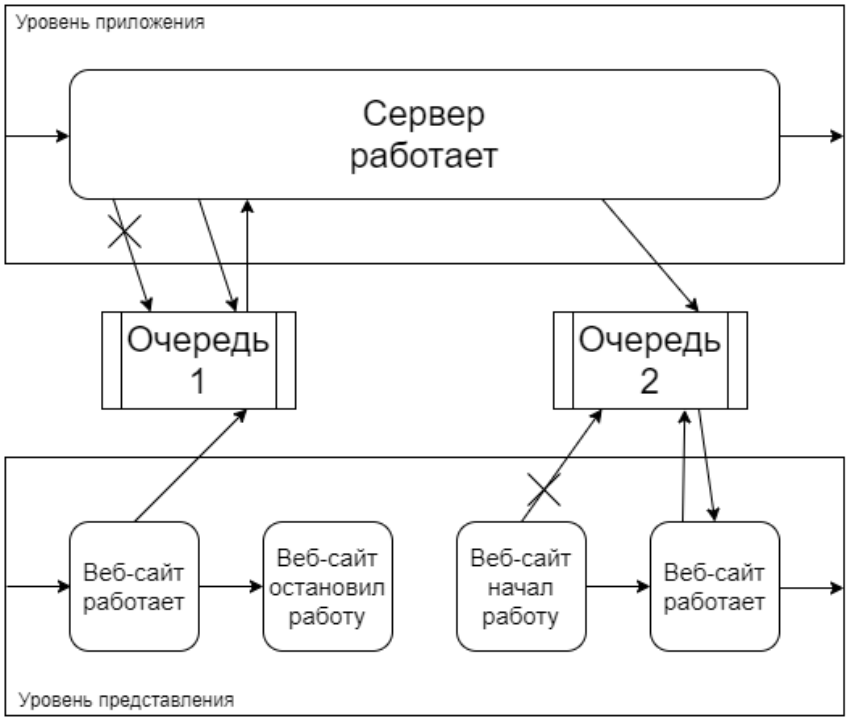

Рисунок 5.4 – Второй пример сохранного асинхронного обмена данными на основе веб-сайта и сервера

При несохранной обработке сообщений никаких очередей или иных промежуточных обработчиков не добавляется (рисунок 5.5). Получается,

что если уровень представления вдруг прекратит свою работу и начнёт её заново, то получить ответы на отправленные ранее запросы будет невозможно (рисунок 5.6). Данные хранятся на уровне приложения только до тех пор, пока обрабатываются. Как только обработка завершится, уровень приложения попытается отправить ответ уровню представления и, вне зависимости от факта получения, удалит данные у себя.

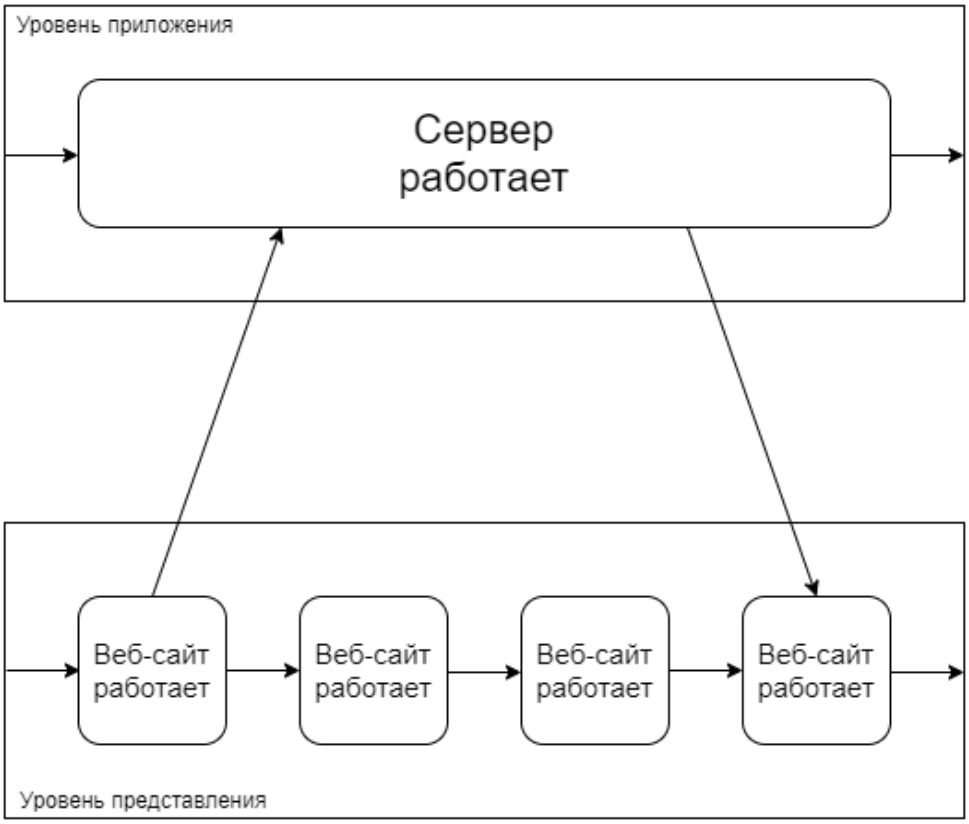

Рисунок 5.5 – Первый пример несохранной асинхронной обработки данных на основе веб-сайта и сервера

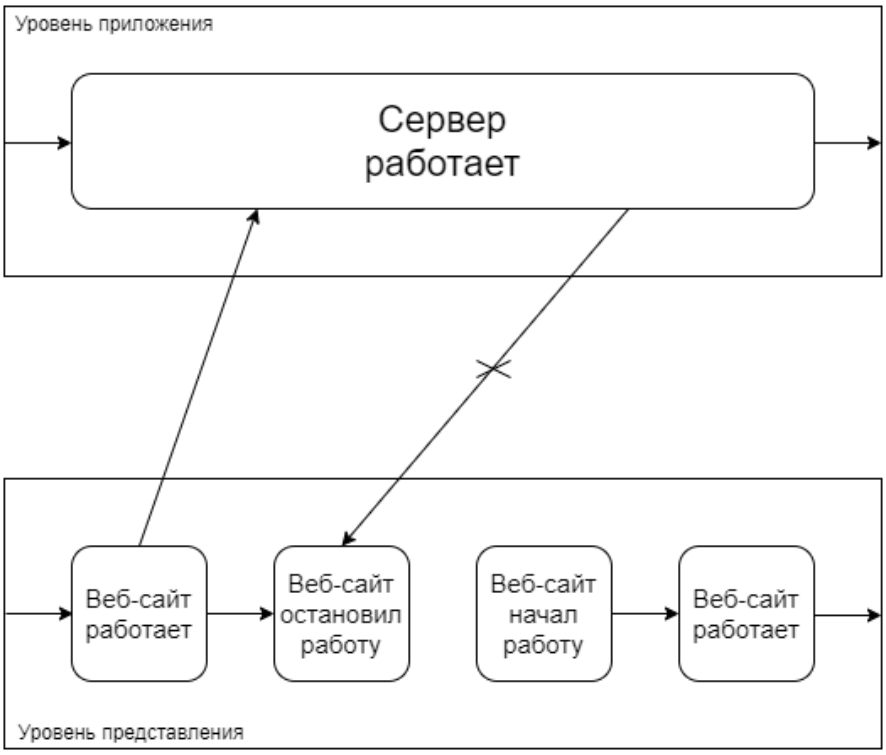

Рисунок 5.6 – Второй пример несохранной асинхронной обработки данных на основе веб-сайта и сервера

Задание на практическую работу:

Поскольку для трёхуровневой архитектуры необходимо физическое разделение подсистем, то предлагается разработать трёхслойное приложение: БД, сервер, приложение. В качестве приложения можно использовать: запросы в postman/insomnia/testmace, простой сайт, десктопное приложение, мобильное приложение. В качестве БД можно использовать: SQLite, PostgreSQL.

Нечетные варианты реализуют синхронное взаимодействие через API. Чётные варианты реализуют несохранное асинхронное взаимодействие через WebSockets.