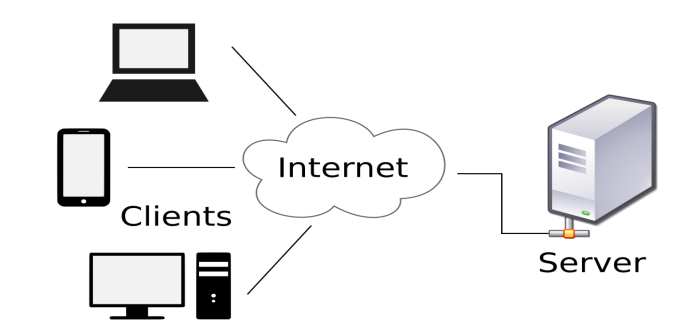
**ВВЕДЕНИЕ В NODE.JS**

**Клиент-серверная архитектура**

Клиент-серверная архитектура определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты, потребители этих функций. Сервером может быть приложение, расположенное на каком-либо узле в сети или в облаке, которое способно обслуживать входящие запросы.

Клиентом может быть приложение на десктопной системе или мобильной платформе, которое обращается за информацией к серверу. Модель взаимодействия на основе подобной архитектуры позволяет разделять функционал и вычислительную нагрузку между клиентскими приложениями (заказчиками услуг) и серверными приложениями (поставщиками услуг).  
  
Обычно сервер может обслуживать нескольких клиентов одновременно. В этом случае говорят о многопользовательском режиме. В таком режиме сервер обрабатывает запросы в нескольких потоках. Например, сервер Apache имеет Multi-Processing Modules (MPMs), которые отвечают за распределение и выполнение запросов в разных потоках.

**Ограничения многопоточной модели**

**Многопоточность** — свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени.

Когда за короткий промежуток времени приходит большое количество запросов, то запросы устанавливаются в очередь. В данном случае очередь — это список невыполненных клиентских запросов.

Иногда запросы могут иметь приоритеты. Приоритет — это уровень "важности" выполнения запроса. Запросы с более высокими приоритетами должны выполняться раньше. Обычно приоритет для запросов определяется разработчиками системы базируясь на информации в запросе и на источнике запроса.

При планируемой нагрузке т.е. ожидаемом количестве запросов за определенный промежуток времени многопоточность дает прирост в производительности.  
Но при пиковых нагрузках производительность системы резко падает, что выражается во времени ожидания. "Время ожидания" — это время, через которое пользователь, послав запрос серверу, получит от него ответ.

Все дело в том, программирование и поддержка многопоточного приложения довольно сложное дело. В ходе обработки запросов пользователей потоки могут взаимодействовать между собой, конкурировать при использовании одних и тех же ресурсов. В ходе взаимодействия и конкуренции возникают конфликты доступа к ресурсам и взаимные блокировки.

Взаимная блокировка это одна из самых неприятных проблем при программировании многопоточности. Чаще всего она случается, когда поток уже заблокировал ресурс А, после чего пытается провести блокировку Б, другой же поток заблокировал ресурс Б, после чего пытается заблокировать ресурс А. Самое печальное в этом типе багов то, что, как правило, ситуация происходит очень редко, и поймать его при тестировании практически невозможно.

**Асинхронное программирование**

Для предотвращения подобных конфликтов в многопоточном программировании используется много разных механизмов. Но так или иначе очень часто возникают ситуации, когда поток просто простаивает, ожидая доступа к ресурсу или результата выполнения другого потока. И именно поэтому многопоточная модель недостаточно эффективна для многопользовательских приложений с большим количеством операций ввода вывода.

Чтобы более эффективно использовать потоки было предложено асинхронное программирование. В традиционной практике программирования большинство операций ввода-вывода происходит синхронно, то есть последовательно в одном потоке. Этот поток будет заблокирован пока подобные операции не будут завершены. В асинхронном программировании выполнение операций ввода вывода позволяет продолжить обработку других задач, не ожидая завершения операций ввода вывода.

Именно поэтому сейчас набирают популярность технологии, которые поддерживают асинхронную модель программирования.

**Что такое MEAN-стек**

В мире ИТ стало модным говорить о стеке, то есть о наборе технологий, которые работают вместе. Не являются исключением и веб технологии. Одним из самых ранних стеков веб-технологий с открытым исходным кодом является стек LAMP: Linux® для операционной системы, Apache для веб-сервера, MySQL для базы данных и Perl (или Python или PHP) в качестве используемого языка программирования для создания веб-страниц на основе HTML.

Эти технологии не были изначально написаны для совместной работы, но в силу обстоятельств применялись вместе. С тех пор мы стали свидетелями взрыва веб-стеков. Кажется, что у каждого современного языка программирования есть соответствующий веб-фреймворк (или два) включающий в себя пестрый набор технологий, чтобы сделать быстрой и легкой разработку нового веб-сайта.

Поэтому такое большое внимание привлекает появившийся MEAN стек на основе технологий: MongoDB (базаданных), Express (бекенд фреймворк), AngularJS (фронтенд фреймфорк), Node.js (платформа для выполнения JavaScript).

Этот стек представляет собой полностью современный подход к веб-разработке в котором один язык программирования (JavaScript) работает на каждом уровне вашего приложения, от клиента до сервера и базы данных.

MEAN стек — это больше, чем просто аббревиатура и набор технологий.  
Попытка перенести базовую платформу из ОС (Linux) в среду выполнения JavaScript (Node.js) приводит к независимости от ОС: Node.js работает также и в Windows® и OS X, как в Linux. Node.js также заменяет веб-сервер Apache в стеке LAMP. Но Node.js — это гораздо больше, чем простой веб-сервер. Фактически, вы не разворачиваете готовое приложение на автономном веб-сервере вместо этого веб-сервер включен в ваше приложение и автоматически устанавливается в стеке MEAN. В результате процесс развертывания становится значительно проще, потому что требуемая версия веб-сервера явно определена вместе с остальными зависимостями времени выполнения.

Переход от традиционной реляционной базы данных, такой как MySQL к NoSQL, schemaless, document-oriented хранилищу данных (все эти понятия будут рассмотрены позже более подробно), например, MongoDB, представляет собой фундаментальное изменение в стратегии хранения данных. Вы потратите меньше времени на создание SQL и больше времени на написание map/reduce функций в JavaScript.  
Вы также избегаете большого количества логики для преобразования данных, поскольку MongoDB поддерживает объектную нотацию JavaScript (JSON). Следовательно, писать RESTful веб-сервисы проще, чем когда-либо.

**Эволюция веб к одностраничным приложениям**

Но самое большое изменение в MEAN по сравнению с LAMP — это переход от традиционной генерации страниц на стороне сервера к одностраничному приложению (SPA) на стороне клиента. Используя Express вы все еще можете обрабатывать создание страниц и маршрутизацию на стороне сервера, но теперь акцент делается на представлениях на стороне клиента с помощью Angular. Эти изменения связаны не только с переносом ядра реализации Model-View-Controller (MVC) со стороны сервера на сторону клиента (притом входящий в состав стека фреймворк Express.js обеспечивает и традиционную маршрутизацию, и генерацию страниц на стороне сервера).

Но вам придётся перейти от синхронного менталитета работы с данными к асинхронному. Также, и это самое главное, вы перейдете от странично-ориентированного представления вашего приложения к компонентно-ориентированному.

Таким образом основное преимущество MEAN стек это простота и общая структура.

* — **MongoDB** предлагает более гибкий, удобный уровень для хранения данных.
* — **Node.js** обеспечивает среду выполненя и по сути является вашим сервером, а Express помагает стандартизировать процесс создания веб-сайтов.
* — Со стороны клиента **Angular** обеспечивает прозрачный способ добавления интерактивных функций и компонентов на основе AJAX.

Объединяя их вместе, вы можете создать простой, прозрачный и согласованный механизм для перемещения данных от пользователя к хранилищу данных и обратно.

*Основные преимущества MEAN стек*:

* — **MongoDB** приспособлено для использования в облаке, что позволяет легко создать, администрировать и масштабировать кластер MongoDB
* — **Node.js** обладает высокой производительностью и упрощает уровень сервера
* — **JavaScript** используется на всех уровнях, что значительно упрощает программирование
* — **JSON** используется как формат данных на всех уровнях
* — **Angular** полностью отделяет уровень представления от других уровней в приложении.

[ПРЕДЫДУЩАЯ ЧАСТЬ](https://www.jsexpert.net/course/fundamental-nodejs/?course_type=content&course_page=1&lecture=1)[СЛЕДУЮЩАЯ ЧАСТЬ](https://www.jsexpert.net/course/fundamental-nodejs/?course_type=section-quiz&course_page=1&lecture=2&section-quiz=1)