1. **Основы Node.js**:
   * Что такое Node.js и его архитектура.

Node.js – это среда выполнения JavaScript, построенная на движке V8 Chrome. Она позволяет запускать JavaScript на сервере вместо браузера и предоставляет разработчикам возможность создавать серверные приложения и сетевые приложения. Архитектура Node.js основана на однопоточной событийно-ориентированной модели, которая позволяет эффективно обрабатывать большое количество одновременных подключений.

* + Установка Node.js и npm (Node Package Manager).

Чтобы установить Node.js и npm:

1. Перейдите на официальный сайт Node.js (https://nodejs.org) и загрузите установщик, соответствующий вашей операционной системе.
2. Запустите установщик и следуйте инструкциям.
3. После установки Node.js автоматически устанавливается npm, менеджер пакетов для Node.js.
4. Чтобы проверить установку, откройте терминал (командную строку) и введите команду `node -v` для проверки версии Node.js и `npm -v` для проверки версии npm.
   * Основы работы с REPL (Read-Eval-Print Loop).

REPL (Read-Eval-Print Loop) – это интерактивная среда, в которой можно выполнять JavaScript-код. Node.js предоставляет REPL для экспериментирования с кодом, отладки и быстрого прототипирования. Чтобы запустить REPL, введите команду `node` в терминале. После этого вы сможете вводить и выполнять JavaScript-код, а REPL будет выводить результаты немедленно.

1. **Модули и пакеты**:
   * Импорт и экспорт модулей в Node.js.

В Node.js модули используются для организации кода. Модули позволяют разбить программу на отдельные файлы, каждый из которых может экспортировать функции, объекты или переменные для использования в других модулях. Для импорта и экспорта модулей в Node.js используются ключевые слова `require` и `module.exports`.

* + Использование сторонних пакетов с помощью npm.

npm (Node Package Manager) – это инструмент, входящий в состав Node.js, который позволяет управлять сторонними пакетами и зависимостями в приложениях Node.js. Чтобы использовать сторонние пакеты, вы должны сначала установить их с помощью npm. Для установки пакетов необходимо выполнить команду `npm install <package-name>` в терминале. Зависимости устанавливаются в папку `node\_modules` в вашем проекте.

* + Понимание разницы между глобальными и локальными пакетами.

Глобальные пакеты устанавливаются в систему и доступны для использования в любом проекте. Они обычно используются для инструментов командной строки, таких как сборщики проектов или глобальные модули Node.js.

Локальные пакеты устанавливаются внутри конкретного проекта и являются зависимостями этого проекта. Они описываются в файле `package.json` и могут быть установлены с помощью команды `npm install` в папке проекта. Локальные пакеты не доступны для использования в других проектах, если их не установить специально.

1. **Система файлов и потоки (File System and Streams)**:
   * Работа с файлами и директориями в Node.js.

В Node.js есть модуль `fs`, который предоставляет API для работы с файлами и директориями. Он позволяет читать и записывать данные в файлы, создавать, перемещать и удалять файлы и директории, а также выполнять другие операции, связанные с файловой системой.

* + Использование потоков для обработки данных.

Потоки (streams) – это механизм в Node.js для обработки и передачи данных пакетами или частями. Потоки позволяют эффективно обрабатывать большие объемы данных и работать с ними по мере их получения. В Node.js существуют различные типы потоков, такие как потоки чтения, потоки записи, потоки трансформации и др. Потоки могут быть использованы для чтения данных из файла, отправки данных по сети, сжатия или распаковки данных и т.д.

1. **Асинхронное программирование**:
   * Обзор колбэков (callbacks) и промисов (Promises).

В Node.js асинхронное программирование осуществляется с помощью колбэков (callbacks) и промисов (Promises). Колбэк – это функция, которая передается в асинхронную операцию и вызывается по завершении операции. Промис – это объект, представляющий результат асинхронной операции, который может быть обработан путем привязки колбэков с помощью методов `then` и `catch`.

* + Применение async/await для управления асинхронным кодом.

`async/await` – это последовательный синтаксис для работы с асинхронным кодом в Node.js. `async` обозначает функцию, которая содержит асинхронный код, а `await` используется внутри таких функций для ожидания завершения асинхронных операций и получения их результатов. `async/await` делает асинхронный код более понятным и легким для написания и понимания.

1. **Express.js и веб-разработка**:
   * Введение в Express.js и создание веб-сервера.

Express.js – это минималистичный и гибкий веб-фреймворк для Node.js, который позволяет создавать веб-сервера и разрабатывать веб-приложения. Он предоставляет множество функций и утилит для обработки запросов, маршрутизации, работы с шаблонами, обработки ошибок и др.

* + Работа с маршрутами (routes) и обработка запросов.

В Express.js маршруты (routes) определяют, как приложение должно обрабатывать запросы клиента на различные URL-адреса. Маршруты определяются с использованием методов маршрутизатора Express (`get`, `post`, `put`, `delete` и т.д.), которые принимают путь и обработчик для выполняемой операции при совпадении пути с запросом. Обработчик может быть функцией или цепочкой функций, называемой промежуточным ПО (middleware), которые выполняются перед или после обработки запросов.

* + Промежуточное ПО (middleware) в Express.

Промежуточное ПО (middleware) в Express – это функции, которые выполняются между началом обработки запроса и отправкой ответа. Промежуточное ПО может выполнять различные задачи, такие как аутентификация, авторизация, обработка ошибок, обработка сеансов и др. Каждое промежуточное ПО принимает три аргумента: `req` (объект запроса), `res` (объект ответа) и `next` (функция, вызывающая следующее промежуточное ПО или обработчик маршрута).

1. **WebSocket и реальное время**:
   * Введение в WebSocket и библиотеку Socket.io.

WebSocket – это протокол, обеспечивающий двустороннюю связь между клиентом и сервером в режиме реального времени. Он позволяет установить постоянное соединение между клиентом и сервером, через которое могут передаваться сообщения в обоих направлениях без необходимости отправлять повторные запросы от клиента.

* + Создание чат-приложения с WebSocket.

Socket.io – это библиотека для Node.js, которая обеспечивает поддержку WebSocket и других транспортных протоколов в режиме реального времени. Она упрощает создание приложений, основанных на событиях, и обеспечивает обмен данными между клиентом и сервером.

1. **Базы данных**:
   * Подключение и работа с базами данных, такими как MongoDB или MySQL.

В Node.js можно подключаться и работать с различными базами данных, такими как MongoDB или MySQL, используя соответствующие драйверы или ORM (Object-Relational Mapping) библиотеки. Для подключения к базе данных обычно требуется установить и настроить соответствующий драйвер, после чего можно выполнять операции CRUD (Create, Read, Update, Delete) для работы с данными.

* + Создание моделей данных и выполнение запросов.

Для работы с базой данных можно создавать модели данных, которые представляют сущности или коллекции данных в базе данных. Модели описывают структуру данных, операции чтения/записи/обновления/удаления и другую логику, связанную с данными. Запросы к базе данных могут выполняться используя специфичный язык запросов (например, SQL для реляционной базы данных) или объектно-ориентированные методы и API.

1. **Аутентификация и безопасность**:
   * Реализация аутентификации и авторизации в Node.js приложении.

Аутентификация – это процесс проверки подлинности пользователей в приложении. В Node.js приложении аутентификацию можно реализовать, например, с использованием пакетов `passport` или `jsonwebtoken`. Они позволяют проверить логин и пароль пользователя, выдать токен аутентификации или создать сеанс, который позволяет идентифицировать пользователя в последующих запросах.

* + Защита приложения от атак, таких как инъекции и CSRF (межсайтовая подделка запроса).

Безопасность – это широкий аспект, который охватывает защиту Node.js приложения от различных уязвимостей и атак. Некоторые из мер безопасности включают в себя предотвращение инъекций (например, SQL-инъекций), обработку и валидацию пользовательского ввода, защиту от межсайтовой подделки запроса (CSRF), использование параметризованных запросов к базе данных, обработку и логирование ошибок и т.д.

1. **Тестирование**:
   * Использование фреймворков для тестирования, таких как Mocha и Chai.

В Node.js для тестирования приложений могут использоваться различные фреймворки и инструменты, такие как Mocha, Chai, Jest и другие. Эти инструменты позволяют писать и запускать различные типы тестов, такие как модульные тесты (unit tests), интеграционные тесты, функциональные тесты и т.д. Тесты помогают обнаруживать проблемы в коде, предотвращать регрессии и обеспечивать корректность и надежность работы приложения.

* + Написание unit-тестов и интеграционных тестов.

При написании тестов рекомендуется использовать сценарии, в которых проверяются различные аспекты приложения: правильность возвращаемых результатов, обработка ошибок, взаимодействие с базой данных и другими сервисами, обработка входных данных и т.д. Тесты должны быть автоматизированы и регулярно запускаться для обнаружения проблем и предотвращения внесения ошибок при внесении изменений.

1. **Деплоймент и управление серверами**:
   * Деплоймент Node.js приложений на хостинг-платформах.

При деплое Node.js приложений на хостинг-платформах обычно используются инструменты и сервисы, которые позволяют загружать и запускать приложения на удаленных серверах. Некоторые популярные хостинг-платформы для Node.js включают Heroku, AWS Elastic Beanstalk, Microsoft Azure, Google Cloud Platform и DigitalOcean. Чтобы развернуть приложение на таких платформах, обычно требуется создать учетную запись, настроить окружение, загрузить код приложения и настроить параметры запуска. Документация каждой конкретной платформы предоставляет подробные инструкции по деплою Node.js приложений.

* + Использование управления процессами, таким как PM2.

PM2 (Process Manager 2) является инструментом управления процессами Node.js, который помогает в управлении приложениями на сервере. Он предоставляет множество функций, таких как мониторинг, перезапуск при падении, масштабирование и логирование. PM2 также позволяет делать мультипроцессорное развертывание, что помогает использовать все ресурсы сервера эффективно. Чтобы использовать PM2, необходимо его установить на сервере с помощью пакетного менеджера Node.js (например, npm) и запустить приложение с использованием команды "pm2 start". PM2 имеет обширную документацию, описывающую его возможности и способы конфигурации.

1. **Микросервисная архитектура**:
   * Введение в микросервисы и их реализация с использованием Node.js.

Микросервисная архитектура — это структура приложения, в которой функциональность разбивается на небольшие, независимые сервисы, которые могут быть развернуты и масштабированы независимо друг от друга. Node.js является популярной технологией для реализации микросервисов благодаря своей масштабируемости, высокой производительности и возможности асинхронного программирования. При реализации микросервисной архитектуры с использованием Node.js каждый сервис может быть реализован как отдельное Node.js приложение, которое взаимодействует с другими сервисами через API.

* + Взаимодействие между микросервисами.

Микросервисы могут взаимодействовать друг с другом через сетевые протоколы, такие как HTTP или сообщения, используя протоколы обмена сообщениями (например, AMQP или MQTT). Один из распространенных подходов к взаимодействию между микросервисами в Node.js – использование API-шлюзов или шаблонов, таких как API Gateway или Service Mesh, которые помогают управлять и маршрутизировать запросы между различными сервисами.

1. **Разработка API и RESTful сервисы**:
   * Проектирование и разработка API с использованием Express.js.

Express.js – это популярный фреймворк для разработки веб-приложений на Node.js, который предоставляет удобные методы для создания и настройки API. При проектировании и разработке API с использованием Express.js необходимо определить маршруты (routes) и обрабатывать запросы, опираясь на методы HTTP (GET, POST, PUT, DELETE и т.д.). Можно использовать дополнительные модули Express.js, такие как body-parser, для обработки тел запросов, и middleware для реализации аутентификации, авторизации и других функций.

* + Введение в RESTful архитектуру и реализация RESTful сервисов.

RESTful архитектура является стандартным подходом к разработке веб-сервисов, где ресурсы предоставляются посредством унифицированных интерфейсов и действий (GET, POST, PUT, DELETE и т.д.). При реализации RESTful сервисов с использованием Node.js и Express.js, рекомендуется придерживаться принципов RESTful дизайна, таких как использование правильных URL-адресов ресурсов, использование статусных кодов HTTP, обеспечение независимости запросов и т.д.

1. **Работа с событиями**:
   * Использование событий и EventEmitter в Node.js.

Node.js предоставляет модуль EventEmitter, который позволяет создавать и управлять событиями. События в Node.js могут быть определены в пользовательском коде или использоваться из встроенных модулей. EventEmitter предоставляет методы для создания событий, подписки на события, генерации событий и управления слушателями событий. Это мощный механизм, который позволяет разрабатывать асинхронные и реактивные приложения.

1. **Дополнительные инструменты и библиотеки**:
   * Изучение других библиотек и инструментов, таких как GraphQL, Redis и др.

В Node.js существует множество дополнительных инструментов и библиотек, которые могут значительно улучшить разработку и функциональность приложений. Например, GraphQL является альтернативой традиционным RESTful API и предоставляет гибкую систему запросов и мутаций данных. Redis – это высокопроизводительная база данных-ключ-значение, которая может использоваться для кеширования данных или реализации системы очередей. Ознакомление с такими инструментами и библиотеками может помочь в оптимизации и разработке расширенных приложений.

1. **Оптимизация и масштабирование**:
   * Оптимизация производительности Node.js приложений.

Для оптимизации производительности Node.js приложений можно использовать различные методы, такие как оптимизация базы данных и запросов, использование кэширования, улучшение алгоритмов обработки данных, оптимизация использования памяти и процессора, использование сжатия данных и оптимизация сетевых запросов. Использование инструментов профилирования и мониторинга также может помочь в выявлении и устранении узких мест в приложении.

* + Масштабирование приложений при помощи кластеризации и балансировки нагрузки.

Для масштабирования Node.js приложений можно использовать кластеризацию, когда несколько экземпляров приложения запускаются на разных процессах или серверах. Вместе с этим, нужно настроить балансировку нагрузки, которая позволит распределить запросы между экземплярами приложения для равномерного распределения нагрузки. Существуют разные методы балансировки нагрузки, включая DNS Round Robin, обратные прокси серверы (например, Nginx), а также облачные сервисы балансировки нагрузки, предоставляемые платформами, такими как AWS и Google Cloud.