

YOUR LOGO

# node js

Here is where your presentation  
begins

# Contents

1. Node.js 개요

2. 주요 기능 및 모듈

3. Node.js 설치 및 설정

4. Node.js의 고급 기능

5. Node.js 보안 및 성능 최적화

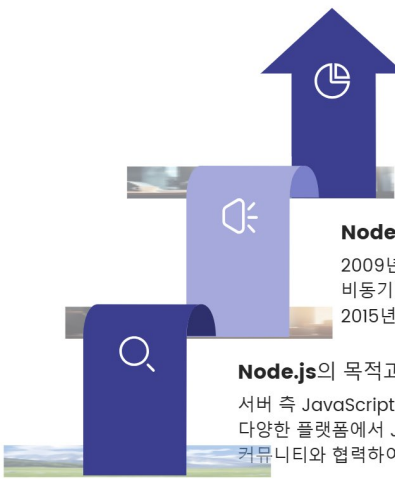
6. Node.js의 미래와 발전 방향

YOUR LOGO

01

## Node.js 개요

# Node.js의 정의



## Node.js란 무엇인가

Node.js는 서버 측에서 JavaScript를 실행할 수 있는 런타임 환경이다.  
V8 JavaScript 엔진을 사용하여 빠른 코드 실행을 제공한다.  
비동기 이벤트 주도 방식으로 높은 성능을 발휘한다.

## Node.js의 역사 및 발전

2009년 Ryan Dahl에 의해 처음 발표되었다.  
비동기 I/O 모델과 이벤트 루프를 채택하여 혁신적인 서버 개발을 가능하게 했다.  
2015년에는 io.js와의 통합으로 현재의 Node.js 재단이 설립되었다.

## Node.js의 목적과 비전

서버 측 JavaScript 환경을 통해 개발자의 생산성을 높이는 것.  
다양한 플랫폼에서 JavaScript로 애플리케이션을 개발하기 쉽도록 하는 것.  
커뮤니티와 협력하여 지속적인 발전과 성장을 도모하는 것.

# Node.js의 주요 특징



## 비동기 I/O와 이벤트 루프

비동기 방식으로 I/O 작업을 처리하여 블로킹 없이 동작한다.  
이벤트 루프를 통해 단일 스레드에서 여러 작업을 효율적으로 관리한다.  
높은 처리 성능과 확장성을 제공한다.



## 단일 스레드 아키텍처

단일 스레드 모델을 통해 동시성을 관리한다.  
멀티 스레드 환경에서 발생할 수 있는 복잡성을 피할 수 있다.  
비동기 이벤트 처리로 다수의 연결을 처리할 수 있다.



## 크로스 플랫폼 호환성

Windows, macOS, Linux 등의 다양한 운영체제에서 실행 가능하다.  
NPM(Node Package Manager)을 통해 다양한 모듈과 패키지를 지원한다.  
플랫폼 간의 일관된 개발 환경을 제공한다.



## 높은 성능 및 확장성

V8 엔진의 최적화 덕분에 빠른 코드 실행을 제공한다.  
이벤트 드리븐 구조로 대규모 트래픽 처리에 적합하다.  
클러스터 모듈을 사용하여 다중 코어 CPU를 활용할 수 있다.



# Node.js의 사용 사례



## 웹 서버 개발

Express.js와 같은 프레임워크를 사용하여 웹 애플리케이션을 구축한다.  
RESTful API 서버를 쉽게 생성할 수 있다.  
높은 동시성을 요구하는 웹 서비스를 구현하기에 적합하다.



## 실시간 애플리케이션

WebSocket을 통한 실시간 통신 기능을 제공한다.  
채팅 애플리케이션, 실시간 협업 도구 등에 사용된다.  
빠른 데이터 전송이 요구되는 애플리케이션에 유리하다.



## 마이크로서비스 아키텍처

독립적으로 배포 가능한 마이크로서비스를 개발한다.  
각 서비스는 작은 단위로 구조화되어 쉽게 관리 가능하다.  
서비스 간의 효율적인 통신을 위한 경량 프로토콜을 지원한다.



## 서버리스 컴퓨팅

AWS Lambda, Azure Functions 등과 연계하여 서버리스 애플리케이션을 개발한다.  
필요할 때만 실행되어 비용 효율적이다.  
확장성과 관리의 용이성을 제공한다.

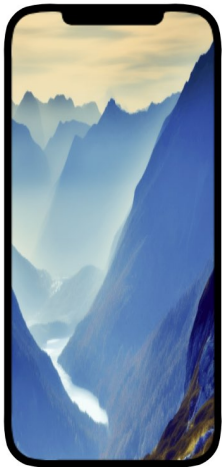
YOUR LOGO

02

주요 기능 및 모듈



## 파일 시스템 모듈



### 파일 읽기

파일을 비동기 또는 동기 방식으로 읽기  
다양한 인코딩 형식을 지원하는 파일 읽기  
특정 파일 위치에서 부분적으로 읽기



### 파일 쓰기

파일을 비동기 또는 동기 방식으로 쓰기  
새로운 파일 생성 및 기존 파일 덮어쓰기  
파일의 특정 위치에 데이터를 추가로 쓰기



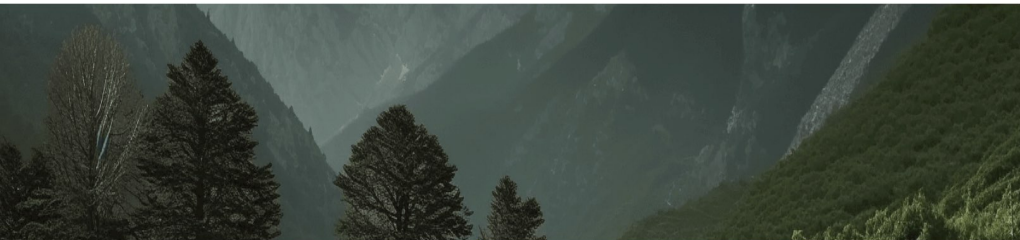
### 파일 삭제 및 변경

파일 제거 작업 수행  
파일의 이름 또는 위치 변경  
파일 접근 권한 수정





# 네트워크 모듈



## HTTP 서버 생성

간단한 HTTP 서버 설정 및 실행  
HTTP 요청 및 응답 처리  
라우팅 및 미들웨어 적용

## 웹 소켓 통신

실시간 양방향 통신 설정  
클라이언트-서버간 메시지 교환  
연결 상태 관리 및 끊기 핸들링

## RESTful API 구현

RESTful 엔드포인트 구성 및 처리  
HTTP 메서드(GET, POST, PUT, DELETE)를 통한 CRUD 작업  
JSON 데이터를 이용한 응답 처리



# 스트림 모듈

## 쓰기 스트림

스트림을 통해 파일 또는 데이터 소스 쓰기  
스트림 이벤트(`drain`, `finish`, `error`) 처리  
데이터를 조각별로 나누어 쓰기

## 파이핑과 스트리밍 데이터

읽기 스트림과 쓰기 스트림을 파이핑  
데이터 스트림을 효율적으로 처리  
스트림 간 데이터 전송 및 변환



## 읽기 스트림

스트림을 통해 파일 또는 데이터 소스 읽기  
스트림 이벤트(`data`, `end`, `error`) 처리  
버퍼를 사용하여 데이터 조각별로 읽기



# 이벤트 모듈



## EventEmitter 클래스

EventEmitter 인스턴스 생성 및 사용  
이벤트 등록 및 발생 관리  
EventEmitter 상속을 통해 사용자 정의 클래스 구현



## 이벤트 리스너 및 발생기

이벤트 리스너 등록 및 제거  
여러 이벤트 리스너 관리  
이벤트 한번 발생 및 지속 발생 처리



## 사용자 정의 이벤트 처리

사용자 정의 이벤트 생성 및 트리거  
복수 사용자 정의 이벤트 핸들링  
이벤트 우선 순위 및 이벤트 흐름 제어

YOUR LOGO

03

## Node.js 설치 및 설정

# Node.js 설치 방법



## 운영 체제별 설치

Windows에서 Node.js 설치 방법  
macOS에서 Node.js 설치 방법  
Linux 배포판별 Node.js 설치 방법



## 버전 관리 도구 사용 (nvm)

nvm 설치 및 설정  
nvm을 이용한 Node.js 버전 설치와 변경  
프로젝트별 Node.js 버전 관리



## npm을 이용한 패키지 설치

npm 설치 방법  
npm 명령어를 이용한 패키지 설치 및 삭제  
글로벌 패키지와 로컬 패키지의 차이점



# 개발 환경 설정



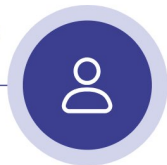
## IDE 및 텍스트 편집기 설정

Visual Studio Code 설치 및 확장팩  
설정  
WebStorm 설치 및 기본 설정  
Sublime Text와 Atom 설치 및 설정



## 디버깅 도구 설정

Node.js 디버깅 기초  
VS Code에서의 디버깅 설정  
크롬 개발자 도구를 이용한 디버깅



## 프로젝트 구조 및 구성

CommonJS와 ES Module 규약  
기본적인 프로젝트 디렉토리 구성 예  
시  
환경 설정 파일 (.env) 이용 방법



# 모듈 관리

## 패키지.json 파일 설정

package.json 파일 생성 및 초기 설정  
의존성 추가 및 스크립트 설정  
npm 스크립트 작성과 실행



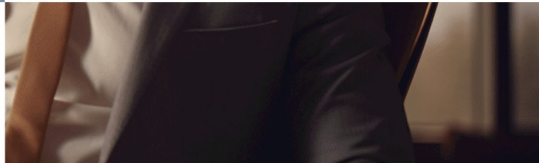
## 로컬 모듈과 글로벌 모듈

로컬 모듈 설치와 활용  
글로벌 모듈의 설치 및 관리  
글로벌 모듈로 설치해야 하는 패키지들



## npm 및 yarn 사용법

npm과 yarn의 차이점  
npm 명령어와 yarn 명령어 비교  
프로젝트에서 npm과 yarn의 혼용 예방 방법



YOUR LOGO

04

**Node.js**의 고급 기능





# 비동기 프로그래밍



## 콜백 함수

정의 및 역할  
중첩된 콜백 문제 (콜백 헬)  
콜백 함수의 사용 예시



## 프로미스 및 **async/await**

프로미스의 기초 개념  
프로미스 체이닝  
async/await의 사용 방법  
프로미스와 async/await 비교



## 이벤트 루프와 타이머

Node.js 이벤트 루프의 역할  
이벤트 루프의 단계  
타이머 함수 (setTimeout, setInterval)의 동작 원리

# 상용 라이브러리 및 프레임워크



## **Express.js** 프레임워크

Express.js의 주요 특징  
간단한 라우팅 구현 예제  
Express 미들웨어



## **Koa.js** 및 **Hapi.js**

Koa.js의 설계 철학 및 특징  
Hapi.js의 주요 기능 및 사용 예시  
Koa.js와 Hapi.js 비교



## **Socket.io** 라이브러리

실시간 통신의 필요성  
Socket.io의 주요 기능  
실시간 채팅 애플리케이션 예제



# 데이터베이스 통합

01

## MongoDB와 Mongoose

MongoDB의 기본 개념  
Mongoose를 이용한 스키마 정의  
MongoDB와 Mongoose를 이용한  
CRUD 예제

02

## MySQL 및 Sequelize

MySQL의 특징 및 장점  
Sequelize ORM의 기초  
Sequelize를 이용한 모델 정의 및 쿼리  
예제

03

## Redis와 캐싱

Redis의 주요 특징  
캐싱의 필요성과 이점  
Node.js 애플리케이션에서 Redis를 이  
용한 캐싱 구현

YOUR LOGO

05

## Node.js 보안 및 성능 최적화



# 보안 모범 사례

01



## 민감 데이터 보호

SSL/TLS를 사용하여 데이터 암호화  
환경 변수 또는 비밀 관리 서비스를 사용하여 비밀번호 및 API 키 보호  
데이터베이스와의 통신 시 암호화 프로토콜 사용

02



## XSS 및 CSRF 방어

사용자 입력 검증 및 필터링 실시  
콘텐츠 보안 정책(CSP)을 설정하여 악성 스크립트 실행 방지  
CSRF 토큰을 사용하여 요청의 유효성 검사

03



## 종단간 암호화

HTTPS 사용으로 클라이언트와 서버 간의 데이터 암호화  
JSON 웹 토큰 (JWT)을 사용하여 데이터 무결성 확보  
암호화 알고리즘을 사용하여 전송 데이터 보호



# 성능 최적화 기술



## 로드 밸런싱

여러 서버 간에 트래픽 분산으로 서버 부하 감소

Nginx, HAProxy 등을 사용하여 로드 밸런싱 구현  
로드 밸런서 설정 및 최적화 방법 학습



## 클러스터링

Node.js 클러스터 모듈 사용으로 멀티 코어 CPU 활용  
클러스터링 기법을 통해 단일 포인트 고장을 방지  
프로세스 관리 도구 (PM2 등)를 사용하여 클러스터 관리



## 메모리 누수 방지

효율적인 메모리 관리 기법 학습  
메모리 누수를 감지하는 도구 (Heap snapshots 등) 활용  
동적 메모리 할당 및 해제 파악



# 모니터링 및 로깅

## STEP. 01

### 성능 모니터링 도구

New Relic, Datadog과 같은 모니터링 도구 사용  
서버 자원 사용량 및 응답 시간 모니터링  
성능 병목 현상 발견 및 극복 방법 분석

## STEP. 02

### 중앙 집중 로깅 시스템

로그 수집 및 분석을 위하여  
ELK(Stack) 설정  
중앙 집권적 로깅 시스템 구축으로 데이터 통합 및 분석  
로그 설정 최적화 및 알림 구성 방법 학습

## STEP .03

### 실시간 에러 추적

Sentry, Loggly 등의 실시간 에러 추적 도구 도입  
애플리케이션 에러 발생 시 즉각적인 대응 체계 마련  
에러 로그 분석 및 패턴 수집을 통한 문제 해결



YOUR LOGO

06

## Node.js의 미래와 발전 방향



# Node.js의 트렌드



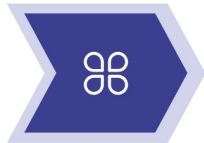
## 최신 기능 및 개선사항

ECMAScript 최신 버전 지원  
성능 최적화 및 속도 개선  
새로운 디버깅 툴 도입  
모듈 시스템 개선



## 커뮤니티와 생태계

활발한 오픈 소스 기여 증가  
다양한 서드파티 모듈과 라이브러리 제  
공  
글로벌 커뮤니티 이벤트와 밋업 개최  
온라인 포럼과 Q&A 활성화



## 경쟁 및 협업 프레임워크

Deno와의 기능 비교와 발전  
Rust와의 통합 시도  
클라우드 네이티브 환경과의 협업  
다양한 개발 프레임워크와의 호환성 향  
상

# 예상되는 발전



## 머신러닝 통합

TensorFlow.js와의 통합 확대  
데이터 분석 및 예측 기능 내장  
AI 기반 애플리케이션 개발 지원  
자동화된 머신러닝 모델 배포



## Edge Computing과의 결합

분산 컴퓨팅 환경 지원  
로컬 데이터 처리 성능 향상  
실시간 데이터 스트리밍 가능  
낮은 지연 시간의 IoT 솔루션 개발



## 강화된 보안 추가

협약서 자동 감지 및 패치 시스템  
강화된 인증 및 접근 제어  
데이터를 보호하는 암호화 기술  
로트 인증서 발급 메커니즘



# Node.js 커뮤니티 및 기여

## 01.

### 오픈 소스 기여 방법

GitHub를 통한 코드 기여  
버그 리포팅 및 수정 제안  
문서화 작업 참여 방법  
커뮤니티 토론에 적극 참여

## 02.

### 주요 컨퍼런스 및 이벤트

Node.js Interactive 컨퍼런스  
JSConf 글로벌 이벤트  
각국의 지역별 밋업  
온라인 웨비나 및 워크숍

## 03.

### 기업 및 사례 연구

대기업의 Node.js 도입 사례  
스타트업 성공 스토리  
다양한 산업 분야의 적용 사례  
오픈 소스 프로젝트를 통한 혁신

YOUR LOGO

# Thanks

Edited by 최유진