Node.js Best Practice

- 코드 스타일
- 에러 처리 방법

코드 스타일

1. ESLint 사용

- 사용해야 하는 이유
  - 코드 스타일의 표준
  - 코드의 품질을 올려준다.
- 사용하지 않을 경우
  - 프로그래머가 간격이나 한줄이 길이(line-width) 등의 비본질적인 문제에 대해 집중해야 한다.
  - 프로젝트의 코드 스타일에 대해 과도하게 생각하느라 시간을 낭비하게 된다.

2. Node.js에 특화된 플러그인 사용

- 핵심 요약
  - Vanilla JS만 지원하는 ESLint의 표준 규칙 위에 `eslint-plugin-node`, `eslint-plugin-mocha`, `eslint-plugin-node-security` 와 같은 Node 특화 플러그인 사용
- 사용하지 않을 경우
  - 결함 있는 Node.js 코드 패턴들이 레이더에서 벗어날 수 있다.
  - 예를 들어, 변수로 된 파일 경로를 이용해 `require(파일경로변수)` 로 파일을 가져오는 코드
  - 공격자들이 어떤 JS script도 실행시킬 수 있게 하기 때문에(?), 미리 감지하고 알려준다.

3. 코드 블록의 중괄호를 같은 줄에서 시작

```
// Do
function someFunction() {
  // code
  return {
    // code
  }
}

// Avoid
function sonFunction()
{
  // code
  return
  {
    // code
  }
}
```

- `javascript automatic semicolon insertion` 때문에 위험한 코드이다.

4. 세미콜론 붙이기

- 사용해야 하는 이유
  - 코드의 명확성, 가독성 증가
- 사용하지 않을 경우
  - `javascript automatic semicolon insertion`로 인해 의도하지 않은 결과 발생

5. `===` 사용하기

- 핵심 요약
  - `==` 가 아닌 `===` 를 사용한다.
- 사용해야 하는 이유
  - `==` 는 두 변수를 공용 타입으로 변환 후에 비교하기 때문이다.
- 사용하지 않을 경우
  - 실제로 같지 않은 변수 비교에 `true` 를 반환한다.

에러 처리 방법

1. 내장된 Error 객체 사용

- 핵심 요약
  - 문자열이나 사용자가 임의로 정한 타입으로 에러를 던지는 것을 지양
  - 내장된 Error 객체를 사용
- 해야 하는 이유
  - 균일성을 향상하고 정보의 손실을 방지한다.
- 하지 않을 경우
  - 어떤 에러의 타입이 반환될지 불확실해져서 적절한 에러 처리가 어려워진다.
  - 에러 처리 로직과 모듈 사이의 상호운영성을 복잡하게 한다.
  - 스택 정보(stack trace)와 같은 중요한 에러 정보를 손실할 가능성이 있다.

2. 에러를 Express 미들웨어로 처리하지 말고 중앙집중적으로 처리

- 핵심 요약
  - 관리자에게 메일을 보내거나, 로깅을 하는 등의 에러 처리는 에러 전용 중앙집중 객체로 캡슐화한다.

```
module.exports.handler = new errorHandler();

function errorHandler() {
  this.handleError = async function(err) {
    await logger.logError(err);
    await sendMailToAdminIfCritical;
    await saveInOpsQueueIfCritical;
    await determineIfOperationalError;
  };
}
```

- 그렇지 않은 경우
  - 코드 중복과 부적절한 에러처리로 이어진다.
- IronMental 에서는?
  - 에러 코드 반환밖에 없기 때문에 미들웨어에서의 처리도 괜찮을 듯

3. 인자값 유효성 검사

- 핵심 요약
  - `Joi` 와 같은 유용한 헬퍼 라이브러리를 사용
- 사용하지 않을 경우
  - 깜빡하고 인자값을 제대로 넘기지 않았을 경우, 제대로 처리되지 않고 지나가는 지저분한 버그가 발생할 수 있다.