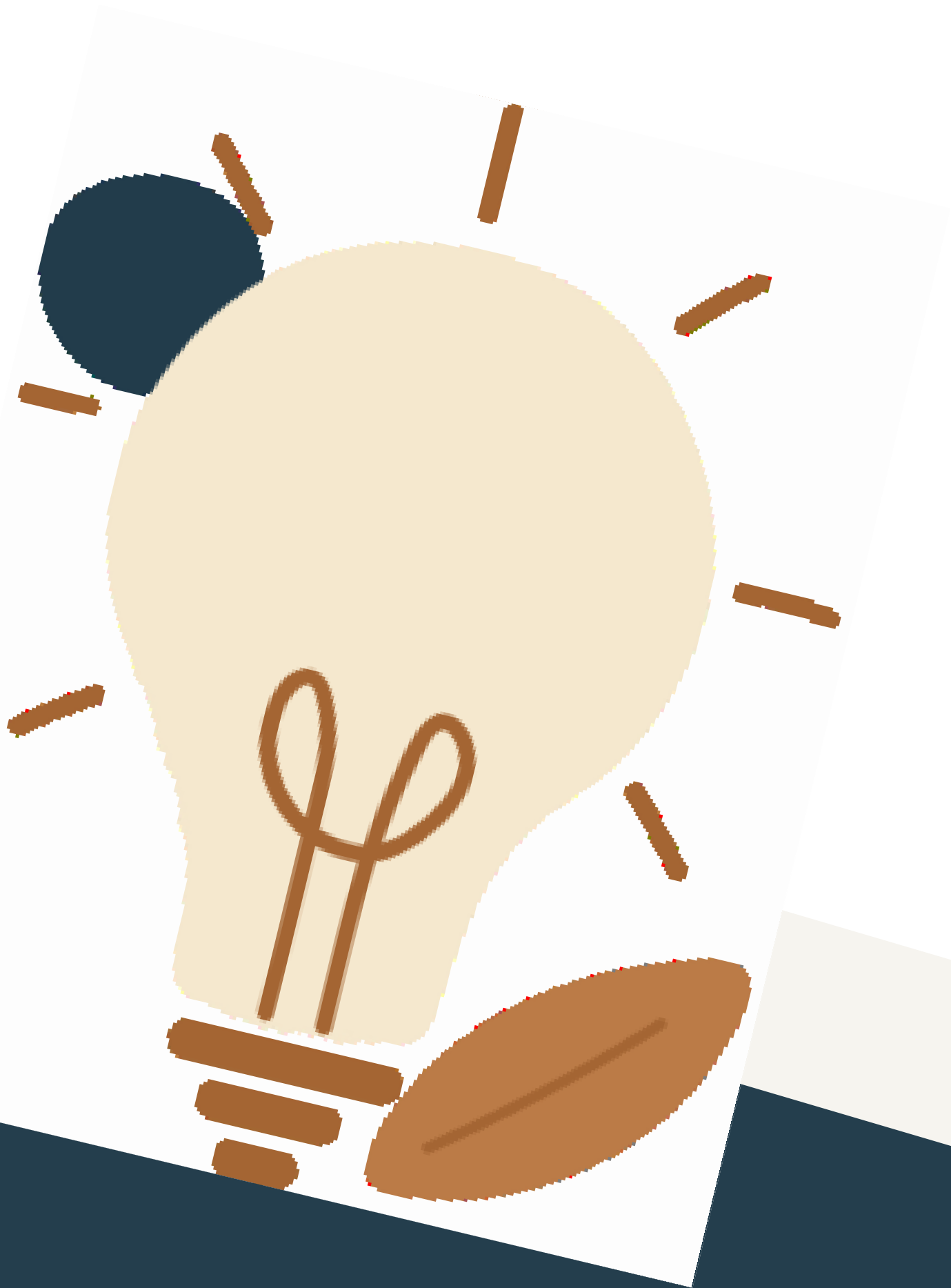




# webOS JS Service



1

# webOS의 JS Services

Node.js 기반의 자바스크립트 서비스

# JS Service

## External JS Service

VS

## Built-In Service

- 빌드환경 필요없음
- 플랫폼 외부에서 CLI로 패키징
- 앱과 함께 패키징 후 설치

- 빌드 환경에서 패키징
- 플랫폼 이미지에 추가하거나  
IPK 파일을 플랫폼 내에서 설치

## | JS Service 구조

파일명	설명
services.json	서비스가 구성되고 작동하는 방법을 설명하는 구성 파일입니다.
service_main.js	서비스 요청자의 요청을 처리하기 위한 메서드를 구현합니다.
package.json	NPM(Node Package Manager)의 구성 파일입니다.

## JS Service

## JS Service

### | services.json

- 서비스가 구성되고 작동하는 방법을 설명하는 구성 파일

```
{
  "id"           : string,
  "description"  : string,
  "engine"       : string,
  "executable"   : string,
  "services": [
    {
      "name"      : string,
      "description": string
    }
  ]
}
```

```
{
  "id": "com.sample.extnode.service",
  "description": "main service",
  "executable": "service_main.js",
  "services": [
    {
      "name": "com.sample.extnode.service",
      "description": "main service"
    }
  ]
}
```

## JS Service

### | package.json

- NPM(Node Package Manager)의 구성 파일
- 더 많은 작성 방식에 대해서는 [npm documentation](#)에서 확인할 수있음

```
{  
  "name": "com.domain.app.service",  
  "main": "helloworld_webos_service.js"  
}
```

```
{  
  "name": "com.sample.extnode.service",  
  "version": "1.0.0",  
  "description": "main service",  
  "main": "service_main.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit  
1"  
  },  
  "author": "",  
  "license": "BSD",  
  "dependencies": {  
    "upper-case": "^2.0.2"  
  }  
}
```

# JS Service

## | JS Service 구조

< service\_main.js >

```
JS
const Service = require('webos-service');
const pkgInfo = require('./package.json');

const service = new Service(pkgInfo.name); // pkgInfo.name = com.sample.exnode.service
service.register("hello", function(message) {
    message.respond({
        greeting: "Hello, This is webOS OSE!"
    });
});
```

- service.register 메서드로 서비스의 메서드를 luna service에 등록
- Client(앱)로부터 "hello" 요청을 받으면 요청 처리 후 "message" 객체의 respond 메서드를 사용해 응답메시지를 Client로 전달

### 결과 호출

```
root@raspberrypi4-64:/var/rootdirs/home/root# luna-send -n 1 -f luna://com.sample.hello.service/hello '{}'  
{  
  "greeting": "Hello, This is webOS OSE!",  
  "returnValue": true  
}
```

# JS Service

## | JS Service의 패키징

cmd

```
$ ares-package hello hello_service  
Create com.sample.hello_1.0.0_all.ipk to D:\Javascript Service  
Success
```

- CLI의 package 명령으로 패키징
- 매개변수로 앱 디렉터리, 서비스 디렉터리 순서로 명시하고, 서비스는 여러 개가 올 수 있음
  - ares-package app app\_service1 app\_service2

## | JS Service의 특징

- Client(앱)으로 부터 서비스 요청을 받을 때 활성화 되고, 5초간 추가적인 요청이 없으면 비활성화 됨 (Dynamic Service : 리소스 점유 방지)
- 지속적으로 유지되는 서비스를 위해, 자기 자신을 5초 이내에 지속적으로 호출하도록 하거나 Client에서 Subscribe를 요청하여 처리하도록 구현



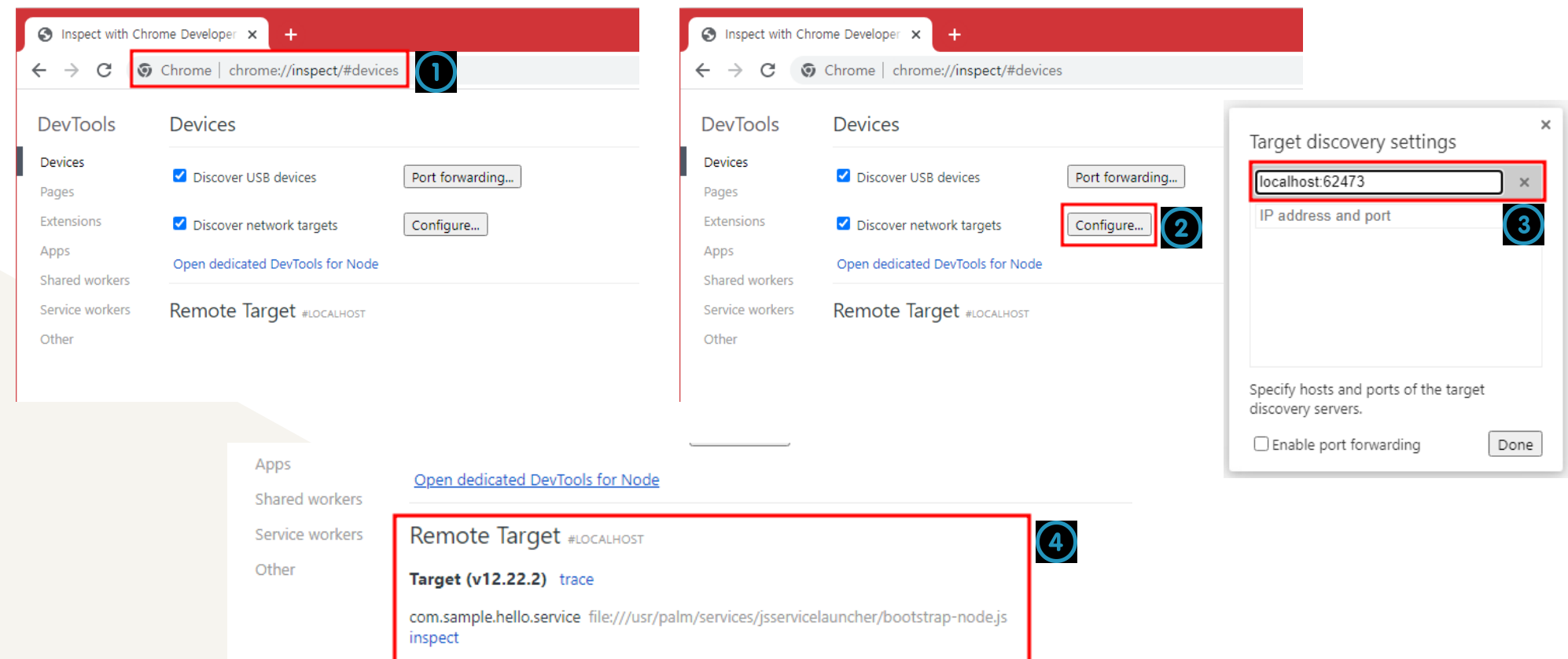
## | JS Service 디버깅

cmd

\$ ares-inspect -d ose -s com.sample.hello.service

- CLI의 inspect 명령으로 서비스 디버깅을 위한 서버 실행
- 노드의 inspector client에서 생성된 서버로 연결


## JS Service



# JS Service - Basic Service

## | 실습 - JS Service 개발: Basic

- 시나리오
  - 기본 웹앱/JS Service 를 생성
    - App id : com.sample.<이름 이니셜> | Service Id : com.sample.<이름 이니셜>.service
    - 예) com.sample.ojd | com.sample.ojd.service
  - Js 파일과 html 파일 수정
  - 앱과 서비스를 하나로 패키징
  - 패키지 설치 후 실행
  - Node Inspector로 로그 확인
- 결과



Hello, webOS!

# JS Service 수정

## | JS Service의 JS 파일 수정

- Helloclient.js 파일 수정
  - 생성한 서비스 이름으로 수정

JS

...

// Subscribe & cancel subscription to helloService's heartbeat method

const Service = require('webos-service');

const service = new Service("com.sample.ojd.service"); // Register com.example.helloworld

console.log("simple call");// Change @SERVICE-NAME@ to real service name

...

webOS

# Web App 수정

## | 웹 앱의 HTML 파일 수정

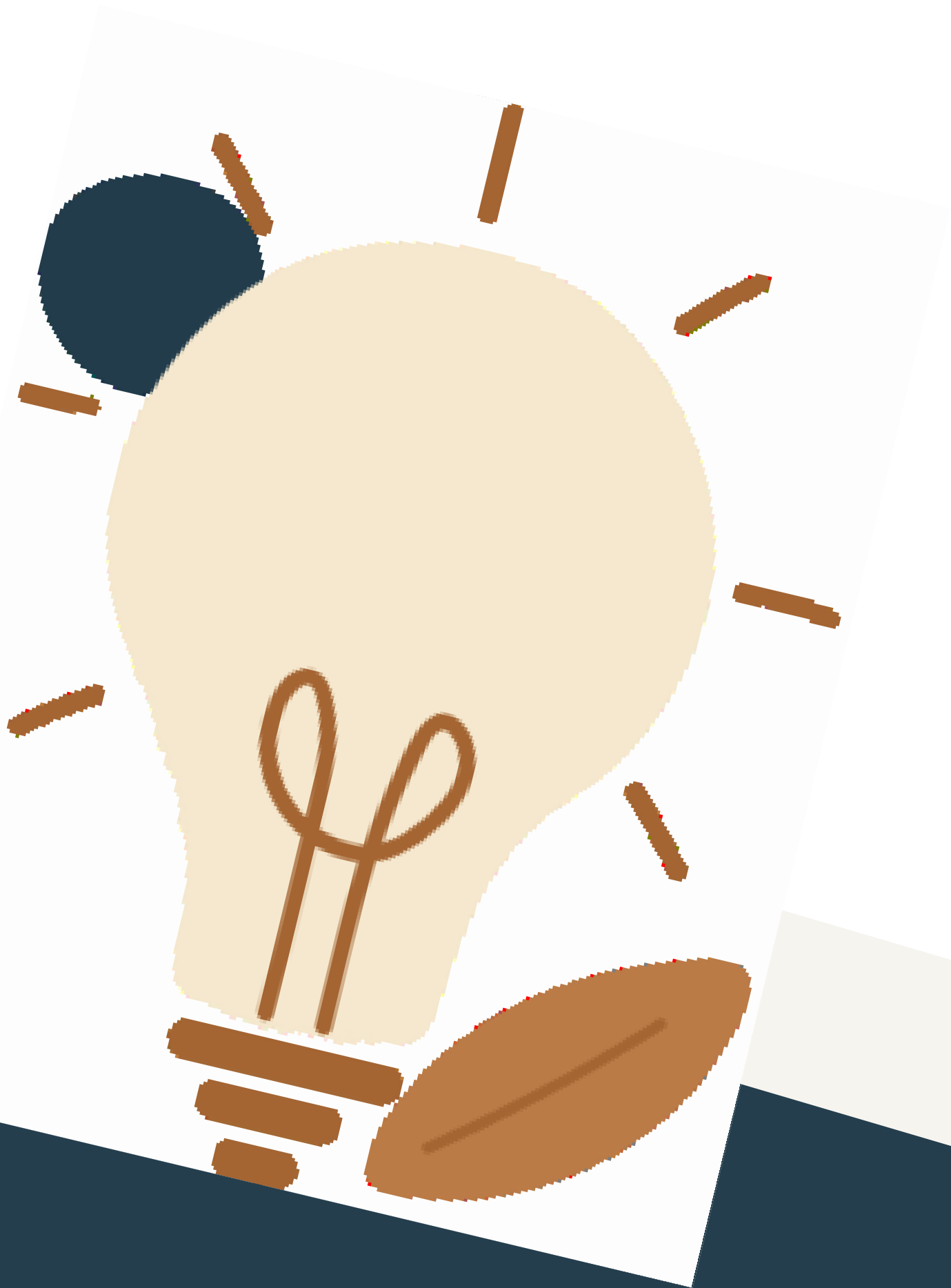
- Index.html 파일 수정
  - 생성한 서비스 이름으로 수정

JS

...

```
var helloApi = 'luna://com.sample.ldh.service/hello';  
var helloParams = '{"name":"webOS"}';
```

...



2

## JS Service에서 LS2 API 호출

JS Service에서는 어떻게 LS2 API를 호출하고 응답을 처리할까?

# JS Service

## | JS 서비스에서 LS2 API 사용

- Service를 등록하고 등록된 서비스 메소드를 호출하면 실행하는 함수로 구성

JS

```
const Service = require('webos-service');

service.register("ServiceName", function(msg){
  service.call("ServiceURI", { }, function(m2) {
    msg.respond({
      returnValue: true,
      Response: m2.payload
    });
  });
});
```

# JS Service

## | JS 서비스에서 LS2 API 사용

JS

```
const pkgInfo = require('./package.json');
const Service = require('webos-service');
const service = new Service(pkgInfo.name);

service.register("showToast", function(msg){
  console.log(logHeader, "SERVICE_METHOD_CALLED:/showToast");
  console.log(msg.payload.message);
  service.call("luna://com.webos.notification/createToast", {"message" : msg.payload.message },
function(m2) {
  console.log(logHeader, "SERVICE_METHOD_CALLED:com.webos.notification/createToast");
  console.log(m2.payload);
  msg.respond({
    returnValue: true,
    Response: m2.payload
  });
});
});
```

- service 객체의 call 메서드를 사용하여 LS2 API 호출
- LS2 API 호출 후 수신한 결과 값은 콜백으로 처리

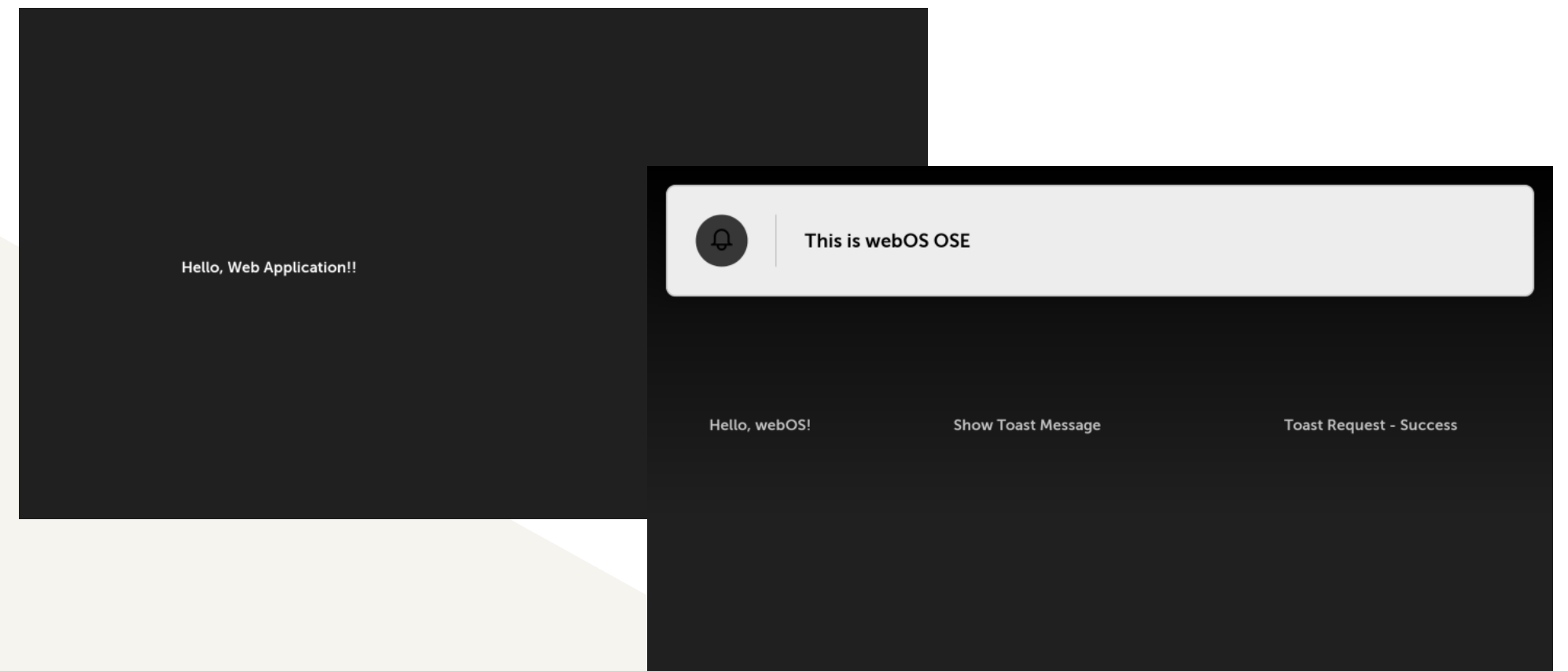
```
service.call(uri, arguments, function callback(message){...});
```

# JS Service LS2 API 호출

## | 실습 - JS Service 개발: API 호출(30분)

- 시나리오
  - 기본 웹앱/JS Service 에 실습용 html 과 js 파일을 붙여넣기
  - 웹 앱에서 “Show Toast Message” 를 클릭하면 JS Service로 Toast Notification 팝업 요청
  - 팝업 메시지는 “This is webOS OSE” 출력 (서비스로 전달하는 파라미터)
  - 호출에 성공하면 “Toast Request - Success” 출력
  - 에러 발생 시 서비스로부터 받은 에러 메시지 출력
  - 서비스가 notification을 실행하고 전달받은 response를 앱으로 전달

### • 결과



참조:

<https://gist.github.com/mibu82/478024ae2c6cedcff1bb77e189f25e48>



webOS

# Web App 수정

## | 웹 앱의 appinfo.json 파일 수정

- Notification - createToast의 ACG 추가

```
JS
...
"requiredPermissions": [
    "time.query",
    "activity.operation",
    "notification.operation"
]
...
```

webOS

# Web App 수정

## | 웹 앱의 HTML 파일 수정

- 화면에 디스플레이되는 문구 추가

```
JS
...
<body>
  <div>
    <h1 id="txt_msg">Hello, Web Application!!</h1>
    <h1 id="show_toast">Show Toast Message</h1>
    <h1 id="result"></h1>
  </div>
...
```

webOS

## | 웹 앱의 HTML 파일 수정

- Window.onload 함수에 추가

JS

...

var toastApi = 'luna://com.domain.app.service/showToast';

var toastParams = '{"message":"This is webOS OSE"}';

...

# Web App 수정

# Web App 수정

## | 웹 앱의 HTML 파일 수정

- Window.onload 함수에 추가

JS

```
function toast_callback(msg){
    console.log("toast_callback : " + msg);
    var arg = JSON.parse(msg);
    if (arg.returnValue) {
        document.getElementById("result").innerHTML = "Toast Request - Success";
        console.log("[APP_NAME: example web app] SHOWTOAST_SUCCESS returnValue : " +
arg.returnValue);
    }
    else {
        console.error("[APP_NAME: example web app] SHOWTOAST_FAILED errorText : " +
arg.Response.errorText);
    }
}
...
document.getElementById("show_toast").onclick = function() {
    bridge.onservicecallback = toast_callback;
    bridge.call(toastApi, toastParams);
}
```

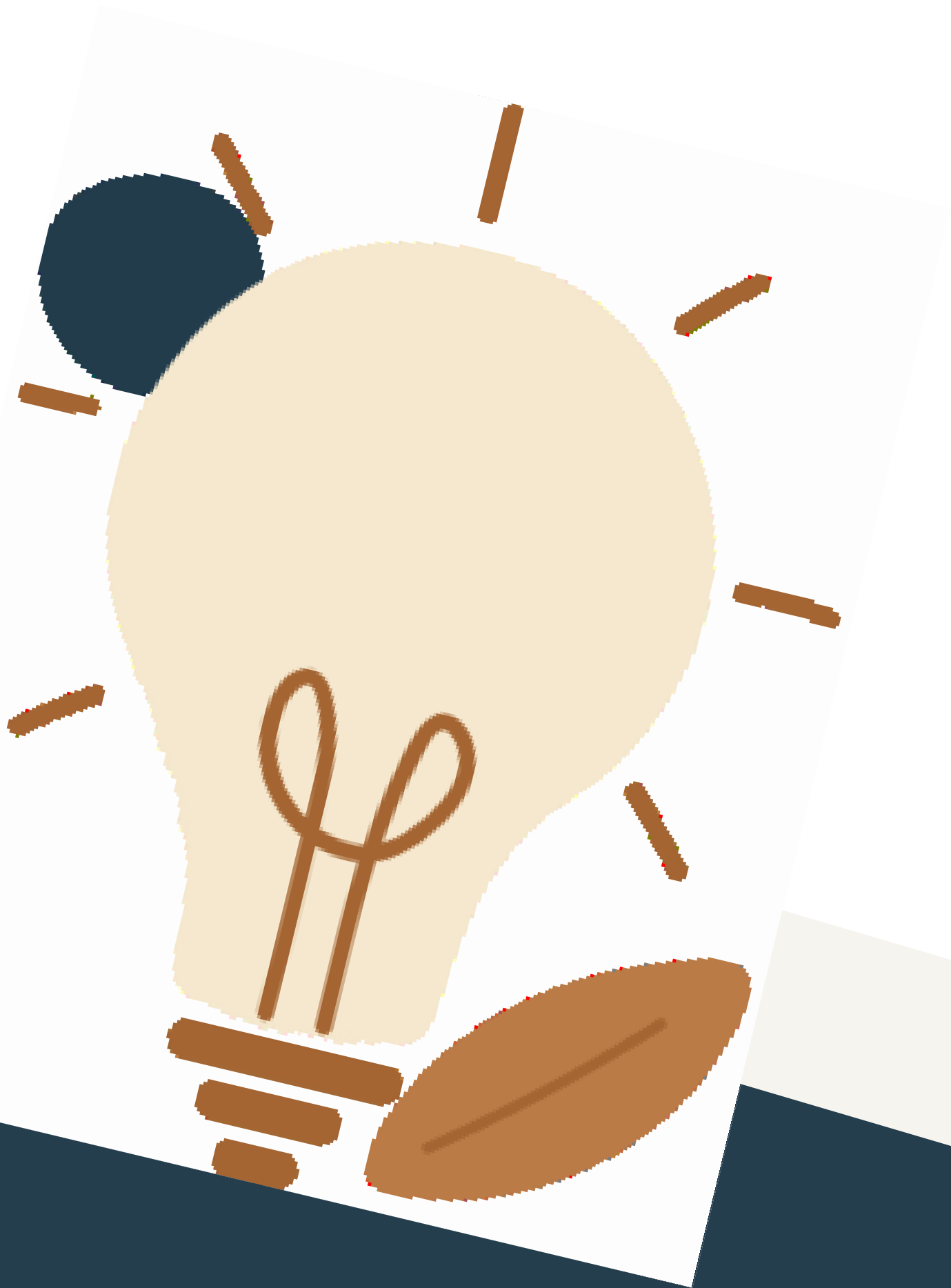
# JS Service 수정

## | JS Service의 JS 파일 수정

- 서비스 파일에 추가

JS

```
service.register("showToast", function(msg){
    console.log(logHeader,"SERVICE_METHOD_CALLED:/showToast");
    service.call("luna://com.webos.notification/createToast", {"message" : msg.payload.message },
function(m2) {
    console.log(logHeader, "SERVICE_METHOD_CALLED:com.webos.notification/createToast");
    msg.respond({
        returnValue: true,
        Response: m2.payload
    });
});
});
```



3

## JS Service에서 Node 기본 모듈 사용

Node.js가 기본으로 제공하는 모듈을 사용해 보자

## JS Service

### | JS Service에서 Node.js 내장 모듈 사용

```
const pkgInfo = require('./package.json');
const Service = require('webos-service');
const service = new Service(pkgInfo.name);
const fs = require('fs');

service.register("writeFile", function(message) {
  const inputText = message.payload.text;

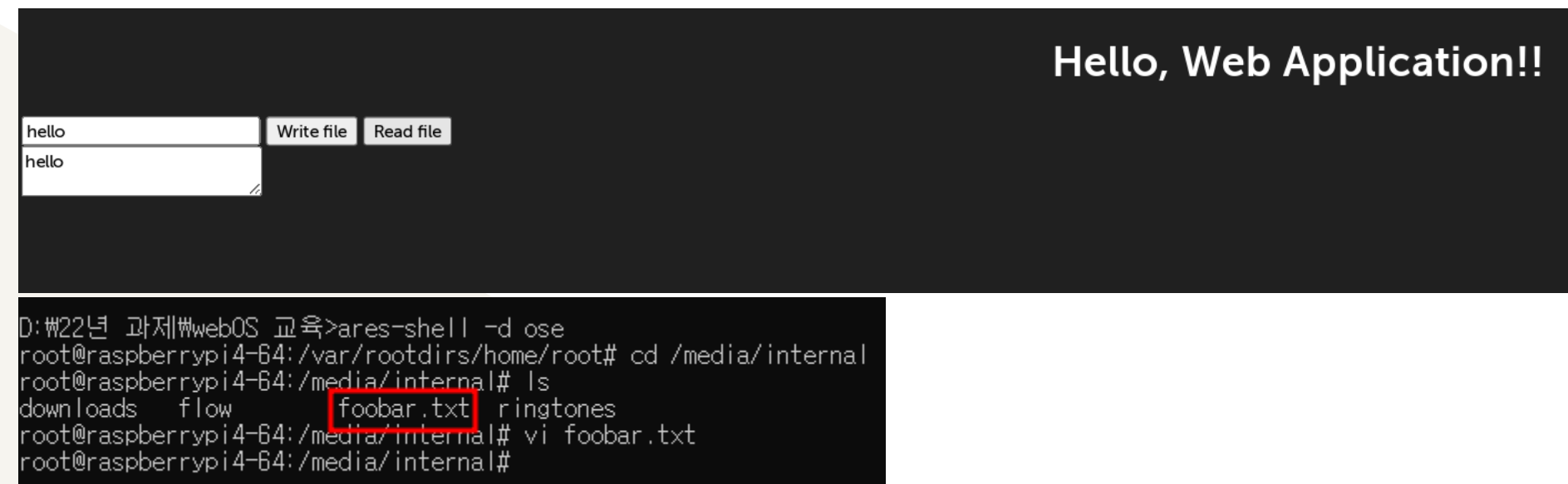
  try{
    fs.writeFileSync('/media/internal/foobar.txt', inputText, "utf8");
    message.respond({
      returnValue: true,
      reply: "write complete",
      errorMsg : ""
    });
  } catch (err) {
    message.respond({
      returnValue: false,
      reply: "write failed",
      errorMsg : err
    });
  }
});
```

- Require 메서드로 사용하고자 하는 내장 모듈 로드
- 서비스 요청을 처리하는 코드 내에서 사용

# JS Service Node Module

## | 실습 - JS Service 개발: Node Module (30분)

- 시나리오
  - 템플릿으로 기본 웹앱/JS Service 생성 후 실습용 html 과 js 파일을 붙여넣기
    - 생성 시 앱 ID : com.sample.fileio | 서비스 ID : com.sample.fileio.service
  - fs 패키지 추가 후 fs 패키지의 readFileSync 와 writeFileSync 함수를 사용해 파일 읽기 쓰기 구현
  - Text input 폼에 글을 입력하고 write 버튼을 클릭하면 입력된 내용이 파일로 저장 (경로 : /media/internal/foobar.txt)
  - 읽기에 성공하면 "write complete" 출력 , 쓰기에 성공하면 "read complete" 출력 (서비스가 앱으로 reply 의 값으로 전달)
  - ares-shell 명령으로 라즈베리파이에 접속하여 foobar.txt 유무와 내용 확인
- 결과



참조:

<https://gist.github.com/mibu82/6e84a0eb04d2bfabea8fa59eabd5013f>



# Web App 수정

## | 웹 앱의 HTML 파일 수정

- 화면에 버튼 및 Input box, textarea 추가

```
html
...
<body>
  <div>
    <h1 id="txt_msg">Hello, Web Application!!</h1>
    <form>
      <input type="text" id="inputText"> <input type="button" id="writeButton" value="Write file">
      <input type="button" id="readButton" value="Read file">
      <br/>
    </form>
  </div>
  <div>
    <textarea id="readData">

    </textarea>
  </div>
...
```

# Web App 수정

## | 웹 앱의 HTML 파일 수정

- Window.onload 함수에 추가 – callback 함수

JS

```
function fsWrite_callback(msg){
    var arg = JSON.parse(msg);
    console.log("arg : " + arg);
    if (arg.returnValue) {
        console.log("[APP_NAME: example web app] CALLWRITE_SUCCESS response : " + arg.reply);
    }
    else {
        console.error("[APP_NAME: example web app] CALLWRITE_FAILED error : " + arg.errorMsg);
    }
}

function fsRead_callback(msg){
    var arg = JSON.parse(msg);
    if (arg.returnValue) {
        document.getElementById("readData").innerHTML = arg.text;
        console.log("[APP_NAME: example web app] CALLREAD_SUCCESS response : " + arg.reply);
    }
    else {
        console.error("[APP_NAME: example web app] CALLREAD_FAILED errorText" + arg.errorMsg);
    }
}
```

# Web App 수정

## | 웹 앱의 HTML 파일 수정

- Window.onload 함수에 추가 - 클릭 이벤트 처리 함수

JS

```
document.getElementById("writeButton").onclick = function() {  
    var writeFileApi = 'luna://com.sample.fileio.service/writeFile';  
    var writeText = document.getElementById("inputText").value;  
    var writeParams = '{"text": "' + writeText + '"}';  
  
    console.log("writeparam : " + writeParams);  
    bridge.onservicecallback = fsWrite_callback;  
    bridge.call(writeFileApi, writeParams);  
}  
  
document.getElementById("readButton").onclick = function() {  
    var readFileApi = 'luna://com.sample.fileio.service/readFile';  
    var readParams = '{}';  
  
    bridge.onservicecallback = fsRead_callback;  
    bridge.call(readFileApi, readParams);  
}
```

# JS Service 수정

## | JS Service의 JS 파일 수정

- 서비스 파일에 추가 - writeFile

```
JS
...
service.register("writeFile", function(message) {
  console.log(logHeader, "SERVICE_METHOD_CALLED:/writeFile");
  console.log(message.payload.text);
  const inputText = message.payload.text;
  console.log("inputed : " + inputText);

  try{
    fs.writeFileSync('/media/internal/foobar.txt', inputText, "utf8");
    message.respond({
      returnValue: true,
      reply: "write complete",
      errorMsg : ""
    });
  } catch (err) {
    message.respond({
      returnValue: false,
      reply: "write complete",
      errorMsg : err
    });
  }
});
...

```

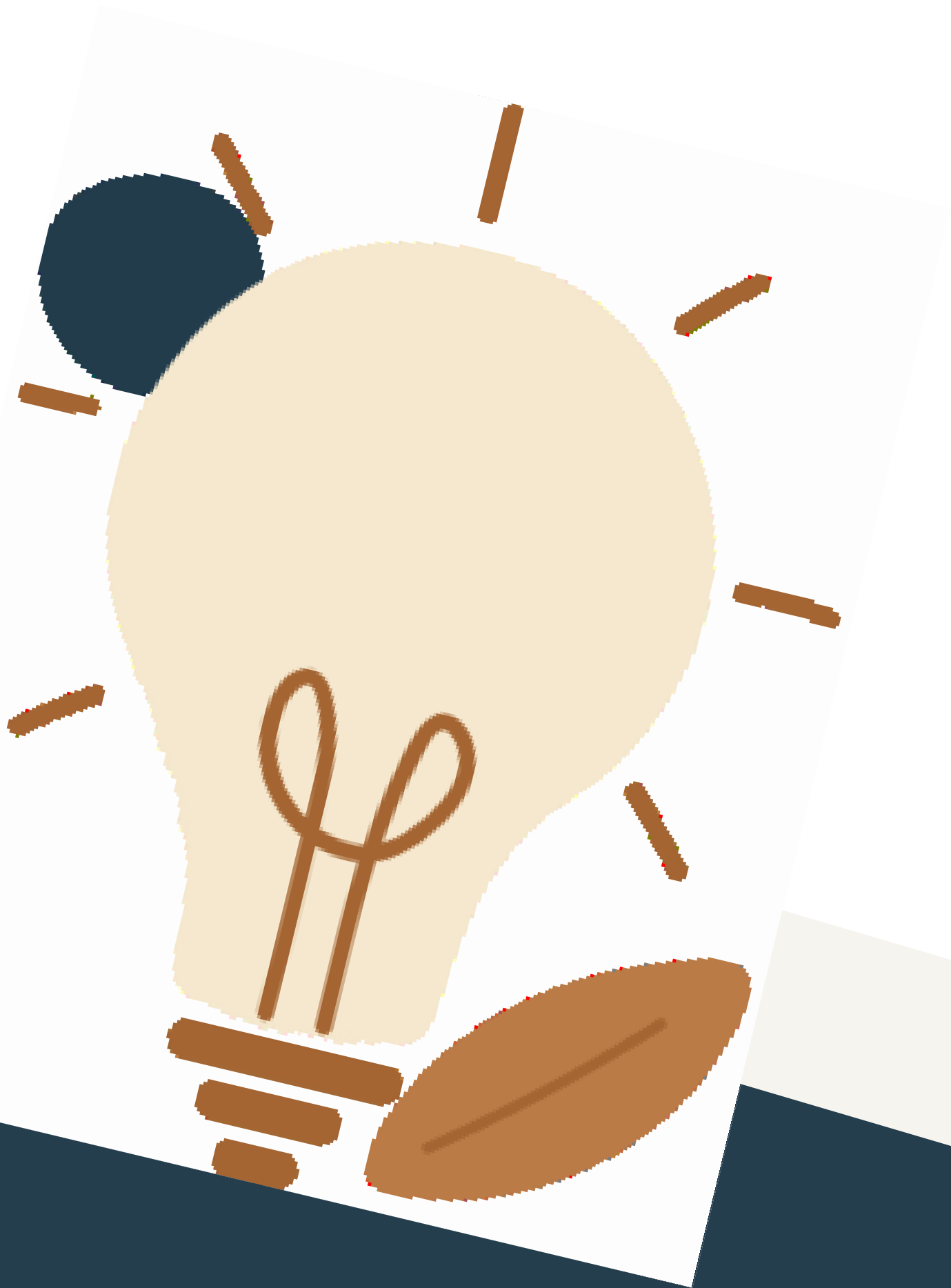
# JS Service 수정

## | JS Service의 JS 파일 수정

- 서비스 파일에 추가 - readFile

```
JS
...
service.register("readFile", function(message) {
  let readData;

  try{
    readData = fs.readFileSync('/media/internal/foobar.txt','utf8');
    message.respond({
      returnValue: true,
      reply: "Read complete",
      text : readData,
      errorMsg : ""
    });
  } catch (err) {
    message.respond({
      returnValue: true,
      reply: "Read fail",
      text : readData,
      errorMsg : err
    });
  }
});
...
```



4


## JS Service에서 Node 외장 모듈 사용

Node.js를 위한 다양한 외부 사용해 보자

webOS

| JS Service에서 외부 모듈 사용


JS Service

  Search Sign Up Sign In


This package has been deprecated


*Author message:*


Use ``str.toLocaleUpperCase`` or ``str.toUpperCase``


**upper-case** 


3.0.0 • Public • Published 3 days ago

 Readme

 Code Beta

 0 Dependencies

 362 Dependents

 13 Versions

Transforms the string to upper case.

### Installation

Install

```
> npm i upper-case
```

Repository

# JS Service

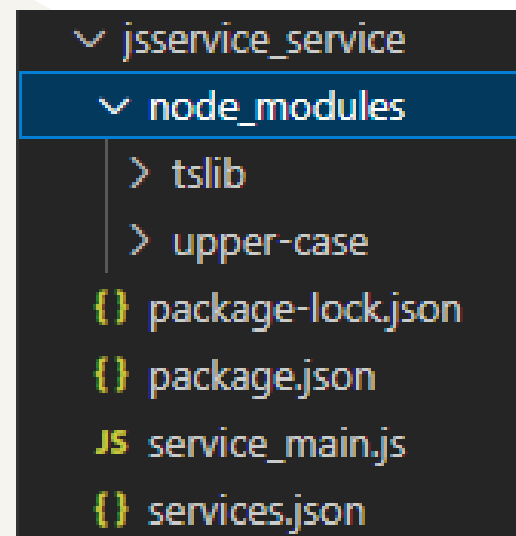
## | JS Service에서 외부 모듈 사용

### 1. Node Package 설치

```
D:\Javascript Service\jsservice_service>npm install upper-case --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN com.sample.jsservice.service@1.0.0 No repository field.
npm WARN com.sample.jsservice.service@1.0.0 license should be a valid SPDX license expression

+ upper-case@2.0.2
added 2 packages from 2 contributors and audited 2 packages in 0.844s
found 0 vulnerabilities
```

- 프로젝트 디렉터리에 노드 패키지를 설치
- 프로젝트 디렉터리에 node\_modules가 생성되고 package.json 파일에 dependencies 에 패키지 이름과 버전이 추가됨





# JS Service

## | JS Service에서 외부 모듈 사용

### 2. 자바스크립트 서비스에서 설치한 패키지 사용

```
const pkgInfo = require('./package.json');
const Service = require('webos-service');
const service = new Service(pkgInfo.name);
const uc = require('upper-case');

service.register("changeCase", function(message){
  let text = message.payload.text;
  if (message.payload.case == "uc"){
    text = uc.upperCase(text);

    message.respond({
      returnValue: true,
      Response: text
    });
  } else {
    message.respond({
      returnValue: false,
      errorText: "can't change the case"
    });
  }
});
```

- Require 메서드로 사용하고자 하는 외부 모듈 로드
- 서비스 요청을 처리하는 코드 내에서 사용

# JS Service External Node Module

## | 실습 – JS Service 개발: External Node Module (15분)

- 시나리오
  - 기본 웹앱 / 기본 JS 서비스 템플릿 생성
  - helloApi 의 서비스 URI 수정
  - JS 서비스 디렉토리에 Uppercase 노드 모듈 설치
  - Hello, webOS를 출력하는 부분에 uppercase를 적용하여 출력
  - 앱 ID : com.sample.extnode | 서비스 ID : com.sample.extnode.service
- 결과

HELLO, WEBOS!

webOS

## | 웹 앱의 HTML 파일 수정

- 서비스 URI 수정

JS

```
var helloApi = 'luna://com.sample.extnode.service/hello';
```

# Web App 수정

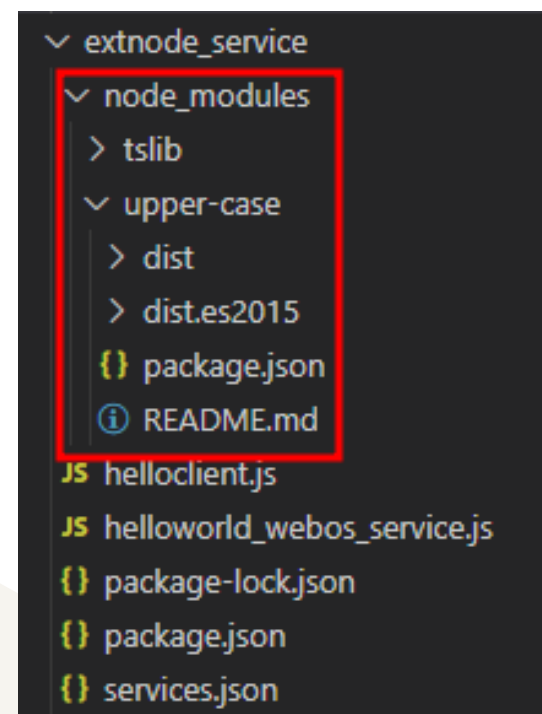
# JS Service 수정

## | JS Service의 JS 파일 수정

- Upper-case node module 설치 (Host PC)

Service 디렉토리에서 아래 명령어 실행

```
Extnode_service > npm install upper-case --save
```



# JS Service 수정

## | JS Service의 JS 파일 수정

- Upper-case 패키지 추가

JS

```
const uc = require('upper-case');
```

- upperCase 함수 사용

JS

```
const responseMsg = uc.upperCase("Hello, " + name + "!");

message.respond({
  returnValue: true,
  Response: responseMsg
});
```

# JS Service External Node Module

## | 실습 - JS Service 개발: External Node Module

- 시나리오
  - Upper-case 프로젝트 복사 (app / service)
  - Md5 module 설치
  - 앱 ID : com.sample.md5.{이름 이니셜} | 서비스 ID : com.sample.md5.{이름 이니셜}.service
  - Appinfo.json 및 service 구성 파일 수정
  - Name 파라미터에 md5 적용
- 결과

HELLO, 2F9252A3D9283C2D676756A931E01443!



# Question

