

08

뷰 템플릿 적용하기

웹 서버에서 사용자 요청에 대한 결과를 응답으로 보낼 때 지금까지는 응답 객체의 `write()` 메소드 또는 `send()` 메소드 안에 직접 웹 문서 내용을 입력했습니다. 그런데 웹 서비스를 시작한 이후 유지관리를 할 때는 자바스크립트 코드 안에 응답 문자열을 입력하는 것 보다 응답 웹 문서를 별도 파일로 만들어 사용하는 것이 훨씬 좋습니다. 응답 웹 문서의 모양을 미리 만들어 둔 것을 템플릿(Template)이라고 합니다. 익스프레스에서는 따로 만들어 둔 템플릿 파일과 결과 값이 들어 있는 변수를 사용해 처리 결과에 따른 응답 웹 문서를 만들 수 있습니다. 이 장에서는 익스프레스에서 응답할 때 사용하는 뷰 템플릿에 대해 알아보겠습니다.

질문으로
시작하기

질문1. 응답 문서를 어떻게 미리 만들어 두었다가 사용하나요?

→ 08-1. ejs 뷰 템플릿 사용하기

질문2. jade라는 템플릿은 어떻게 사용하나요?

→ 08-2. jade 뷰 템플릿 사용하기

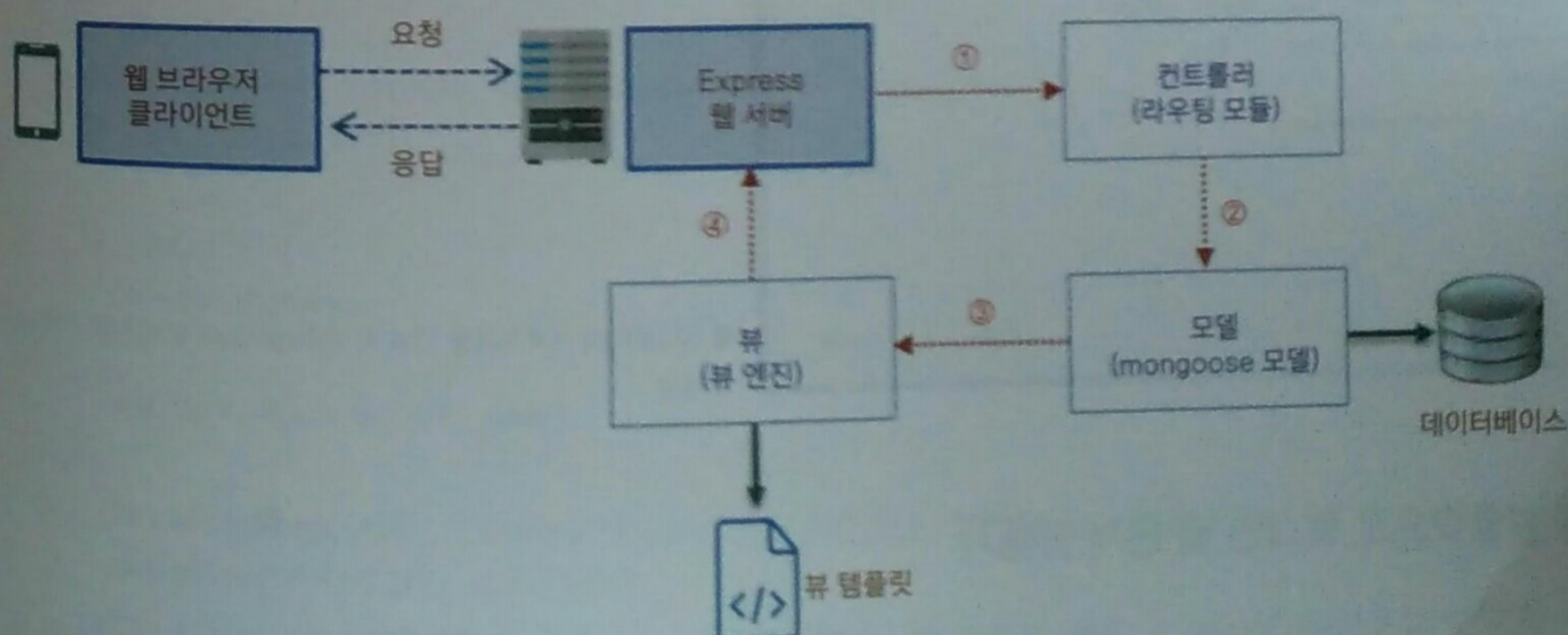
08-1 | ejs 뷰 템플릿 사용하기

최근에 만들어지는 새로운 언어(Language)들은 대부분 MVC 패턴(Model-View-Controller 패턴)을 사용합니다. 즉, 눈에 보이는 부분은 뷰, 뷰로 표현되는 데이터를 제공하는 것은 모델, 처리되는 과정을 담당하는 것은 컨트롤러로 구분하여 구성하면 구조를 더 쉽게 이해할 수 있습니다.

노드와 익스프레스도 예외는 아니어서 지금까지 만든 각각의 기능을 뷰, 모델, 컨트롤러로 나눌 수 있습니다. 앞 장에서 익스프레스로 만든 프로젝트를 살펴보면 라우팅 함수가 사용자 요청을 받은 후 데이터베이스에 데이터를 요청하고 그 결과를 사용자에게 보여 줍니다. 이 과정에서 사용자 요청을 처리하는 라우팅 함수를 컨트롤러(Controller), 데이터베이스에 데이터를 저장하거나 조회하는 함수를 모델(Model), 사용자에게 결과를 보여 주기 위해 만든 파일을 뷰(View)라고 합니다.

그중에서 뷰에 해당하는 부분을 살펴보면, 지금까지는 사용자에게 결과를 응답으로 보낼 때 자바스크립트 코드를 직접 입력하는 방법을 사용했습니다. 그런데 이 방식은 각각의 요청을 처리하는 함수마다 응답 코드를 문자열로 넣어 주어야 하므로 웹 문서를 코드 안에 입력하는 과정에서 오탈자가 생기기 쉽습니다. 따라서 웹 문서의 기본적인 형태를 별도의 파일로 미리 만들어 두고 사용하는 것이 좋습니다.

이제부터는 응답 웹 문서의 기본 형태를 뷰 템플릿 파일에 만들어 두고 사용합니다. 뷰 템플릿을 사용하면 웹 문서의 기본 형태는 뷰 템플릿으로 만들고 데이터베이스에서 조회한 데이터를 이 템플릿 안의 적당한 위치에 넣어 웹 문서를 만들게 됩니다. 이렇게 뷰 템플릿을 사용해 결과 웹 문서를 자동으로 생성한 후 응답을 보내는 역할을 하는 것이 뷰 엔진(View Engine)입니다.



▲ 익스프레스에서 뷰 엔진의 역할

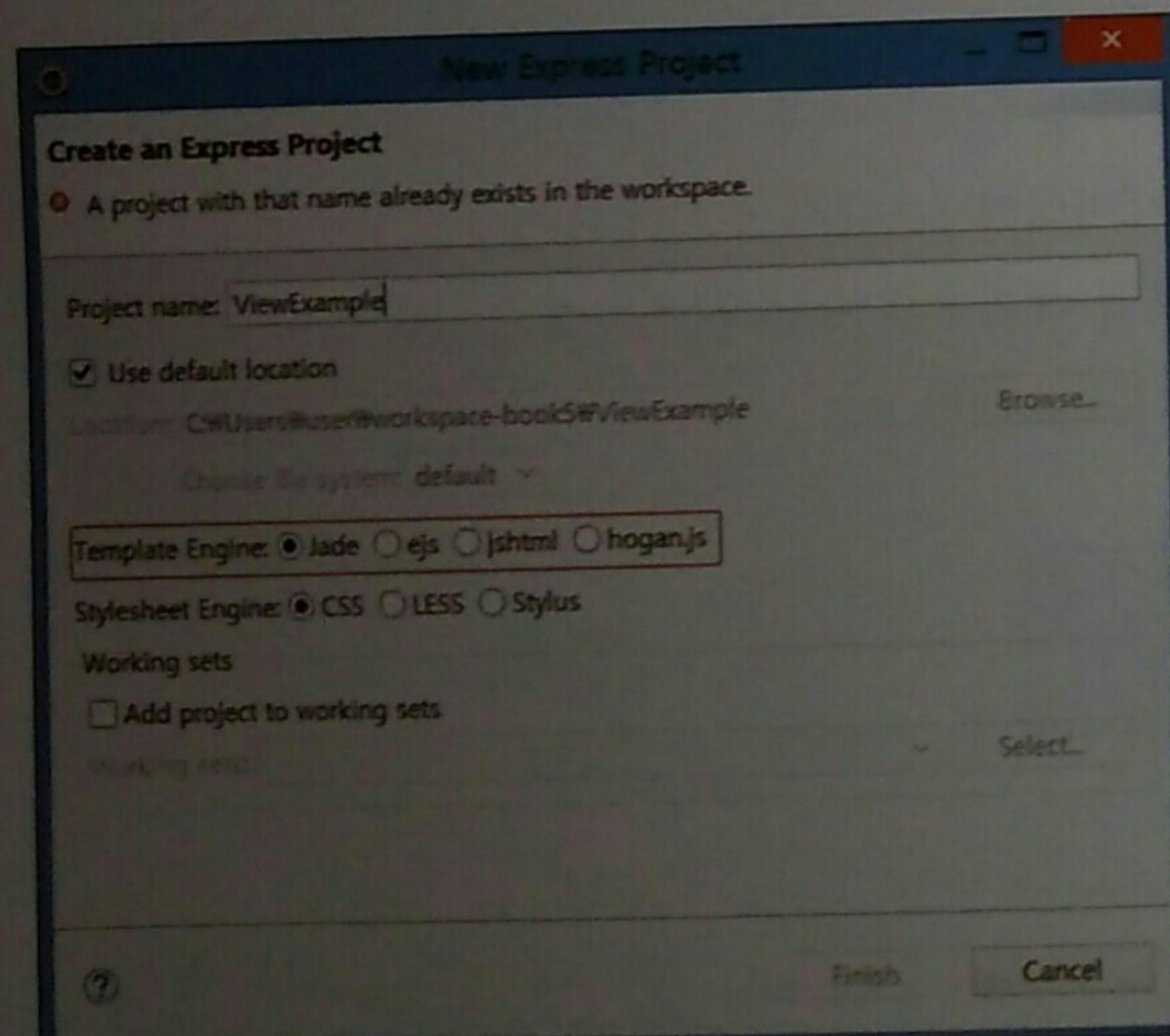
웹 브라우저에서 보내온 요청은 웹 서버인 익스프레스에서 컨트롤러로 보냅니다. 익스프레스에서는 특정 패스로 들어온 요청을 라우팅 함수에서 처리하므로 라우팅 함수를 컨트롤러라고 합니다. 이 컨트롤

러 안에서는 사용자 요청을 처리하기 위해 mongoose 스키마와 모델 객체를 이용해 데이터베이스를 조회하거나 데이터베이스에 저장합니다. 이런 역할을 하는 것이 모델이며, 모델에서 처리한 결과는 뷰 엔진으로 전달됩니다.

뷰 엔진은 뷰 템플릿 파일에서 웹 문서의 기본 형태를 읽어 들여 사용자가 보게 될 최종 웹 문서를 만든 후 클라이언트에 응답으로 보냅니다. 뷰 템플릿을 만드는 형식에는 여러 가지가 있는데 뷰 엔진이 각각의 뷰 템플릿을 처리합니다. 뷰 템플릿의 종류에 따라 뷰 엔진도 달라집니다. 익스프레스에서는 ejs, jade 등 여러 가지 뷰 엔진을 지원합니다. 그중에서 먼저 ejs 뷰 템플릿을 사용하는 방법에 대해 알아보겠습니다.

ejs를 사용하면 웹 문서에서 사용하는 태그를 그대로 템플릿 파일 안에 넣을 수 있으며 필요한 부분에만 변수를 삽입하거나 중간중간 자바스크립트 코드를 넣을 수도 있습니다. 따라서 웹 페이지에 익숙한 웹 개발자에게는 아주 쉬운 형식입니다.

익스프레스에서는 프로젝트를 새로 만들 때 보여 주는 대화상자에서 뷰 엔진을 선택할 수 있으며, 뷰 엔진을 ejs로 선택하면 ejs 형식의 뷰 템플릿 문서를 만들어 사용할 수 있습니다. 프로젝트를 처음 만들 때 다른 뷰 엔진을 선택했더라도 메인 파일인 app.js 파일의 코드를 수정해서 직접 뷰 엔진을 지정할 수도 있습니다.



◀ 익스프레스 프로젝트를 만들 때 대화상자에서 뷰 엔진을 선택하는 경우

뷰 템플릿으로 로그인 웹 문서 만들기

뷰 템플릿 사용법을 알아보기 위해 먼저 DefaultExample 프로젝트를 복사하여 ViewExample 프로젝트를 만듭니다. DefaultExample 프로젝트에는 익스프레스로 만든 웹 서버의 가장 기본적인 구조가 만들어져 있으므로 이 프로젝트를 복사하여 사용하면 편리합니다. 이 프로젝트 안에는 사용자 정보를 처리하는 함수들이 포함되어 있습니다. 그중에서 로그인 기능을 처리한 후 응답하는 과정에 뷰 템플릿을 적용해 보겠습니다.

먼저 프로젝트 안에 있는 app.js 파일을 열고 뷰 엔진을 ejs로 지정합니다. 그러면 이제부터는 ejs로 만든 뷰 템플릿을 사용해 응답을 보낼 수 있습니다.

참조 파일 | ViewExample>app.js

```
....  
app.set('views', __dirname + '/views');  
app.set('view engine', 'ejs');
```

app 객체의 set() 메소드는 속성을 설정하는 역할을 하는데, 속성 이름이 view engine으로 들어 있으면 이 부분이 바로 뷰 엔진을 설정하는 부분입니다. 두 번째 파라미터로 ejs를 전달하면 ejs 포맷의 뷰 템플릿을 처리할 수 있는 뷰 엔진이 설정됩니다.

그다음에는 [routes] 폴더 안에 있는 user.js 파일을 열어 봅니다. 그 안에는 로그인 기능을 처리하는 login 함수가 들어 있습니다. 이 함수의 코드 중에서 정상 처리되었다는 응답을 보내는 코드를 먼저 살펴봅니다.

참조 파일 | ViewExample>/routes/user.js

```
var login = function(req, res) {  
    ....  
    authUser(database, paramInt, paramPassword, function(err, docs) {  
        if (err) {throw err;}  
  
        if (docs) {  
            console.dir(docs);  
  
            var username = docs[0].name;  
  
            res.writeHead('200', {'Content-Type' : 'text/html; charset = utf8'});  
            res.write('<h1>로그인 성공</h1>');  
            res.write('<div><p>사용자 아이디 : ' + paramInt + '</p></div>');  
            res.write('<div><p>사용자 이름 : ' + username + '</p></div>');  
            res.write("<br><br><a href = '/public/login.html'>다시 로그인하기</a>");  
            res.end();  
        }  
    }  
};
```

사용자 인증이 성공했을 때 클라이언트에 응답 웹 문서를 보내기 위해 여러 가지 태그를 입력한 것을 볼 수 있습니다. 사용자의 아이디와 이름이 변수에 들어 있으므로 + 기호로 다른 문자열과 함께 붙인 다음 응답 객체의 write() 메소드를 호출하여 응답을 보냅니다. 이렇게 클라이언트에 응답을 보내기 위해 입력한 코드 중에서 HTML 태그 부분만 새로운 뷰 템플릿 파일로 만듭니다. [views] 폴더에 login_success.ejs 파일을 만들고 다음 코드를 입력합니다.

참조 파일 | ViewExample>/views/login_success.ejs

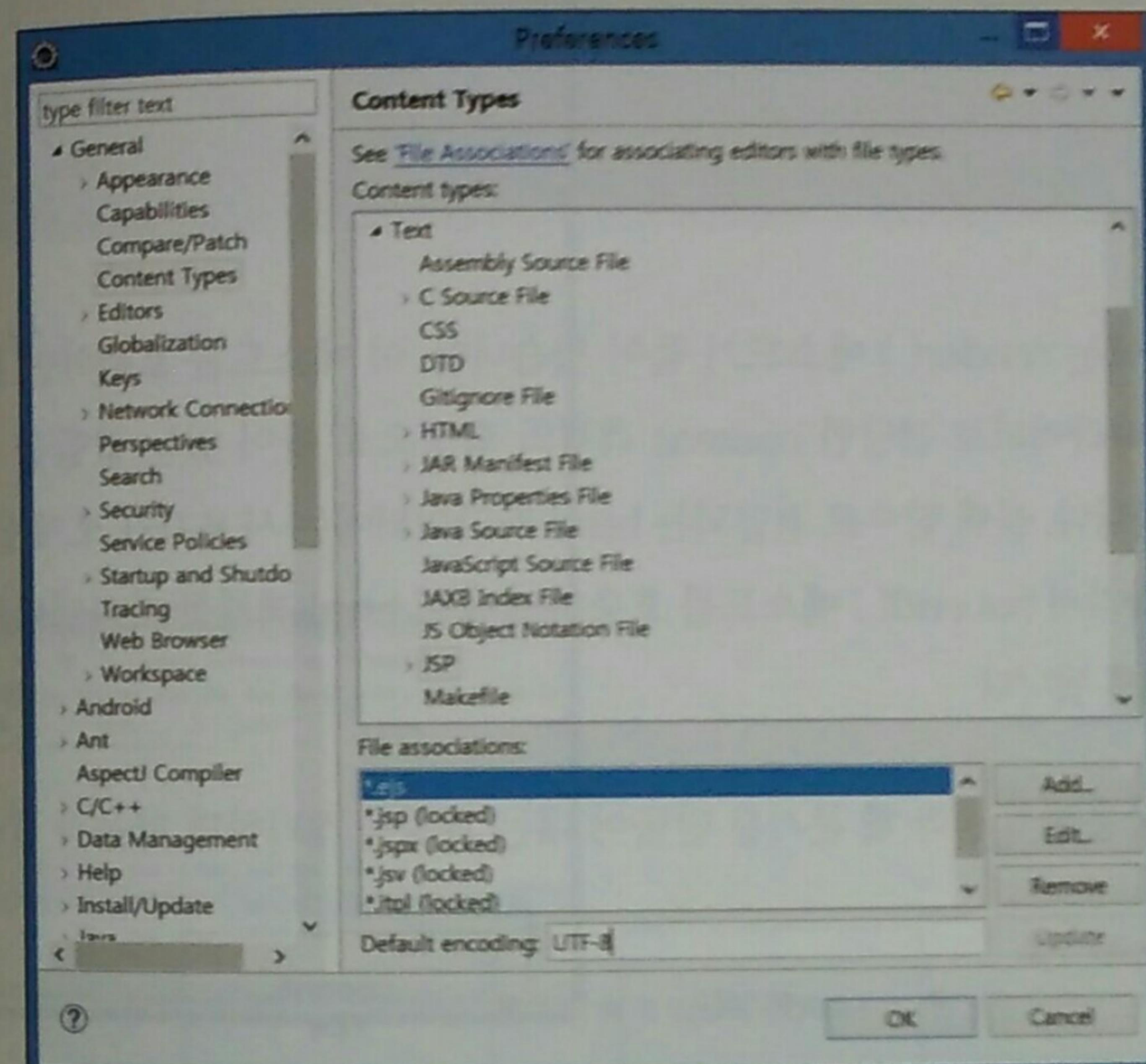
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8">
    <title>로그인 성공 페이지</title>
  </head>
  <body>
    <h1>로그인 성공</h1>
    <div><p>사용자 아이디 : <% = userid %> </p></div>
    <div><p>사용자 이름 : <% = username %> </p></div>
    <br><br><a href = '/public/login.html'>다시 로그인하기</a>
  </body>
</html>
```

user.js 파일 안에서 코드 중간에 문자열로 태그를 입력할 때는 <html> 태그나 <head> 태그가 필요하지 않았습니다. 하지만 별도의 파일을 하나 만들었으니 하나의 온전한 HTML 파일 형태를 만들기 위해 <html> 태그나 <head> 태그를 추가로 입력합니다. <body> 태그 안에는 로그인에 성공했을 때 메시지를 보여 주는 태그를 넣었습니다. 여기에서 중간에 <% 와 %> 기호가 들어 있는데 이 기호는 자바스크립트 코드를 넣어 주는 기호입니다. 즉, 웹 문서 내용 안에 변수 값을 넣어 줄 때 사용하는 기호입니다. 이 기호 중 앞에 있는 기호에 = 이 붙으면 바로 뒤에 변수를 넣을 수 있으며 그 변수의 값을 웹 문서에 출력할 수 있습니다.

ejs 파일을 열었을 때 태그에 색상이 표시되지 않나요?

ejs 확장자를 가진 파일을 열었을 때 태그에 색상이 표시되지 않는다면 이클립스가 ejs 파일을 제대로 인식하지 못하고 있다는 의미입니다. 이것은 ejs 확장자를 가진 파일을 열었을 때 이클립스에서 사용할 편집기가 등록되지 않아 생긴 문제입니다. 이런 경우 ejs 파일을 열어도 태그에 색상이 표시되지 않습니다. ejs 확장자를 가진 파일은 기본적으로 HTML 파일의 형태를 그대로 가지고 있으므로 HTML Editor나 JSP Editor를 설정하면 컬러링을 사용할 수 있습니다.

이클립스의 [Window → Preferences] 메뉴를 선택한 후 나타난 [Preferences] 대화 상자의 왼쪽 메뉴에서 [General → Content Types]를 선택합니다. 오른쪽에 있는 Content Types 항목 중에서 [Text → JSP]를 선택한 후 아래쪽 [File associations]에 *.ejs를 추가합니다. Default encoding 항목에는 UTF-8을 입력한 후 오른쪽의 [Update] 버튼을 누릅니다.



▲ ejs 확장자 파일의 편집기 설정 화면

아래쪽 [OK] 버튼을 누르고 나서 login_success.ejs 파일을 더블클릭하여 열면 컬러링이 적용됩니다.

뷰 엔진은 이 템플릿 파일을 읽어 들이고 userid와 username 변수의 값으로 해당 부분의 값을 대체한 후 그 결과를 만들어냅니다. 이제 이 템플릿 파일을 이용해 응답 웹 문서를 만든 후 클라이언트에게 응답을 보내도록 user.js 파일의 코드를 수정합니다.

```

.....
res.writeHead('200', {'Content-Type' : 'text/html; charset = utf8'});

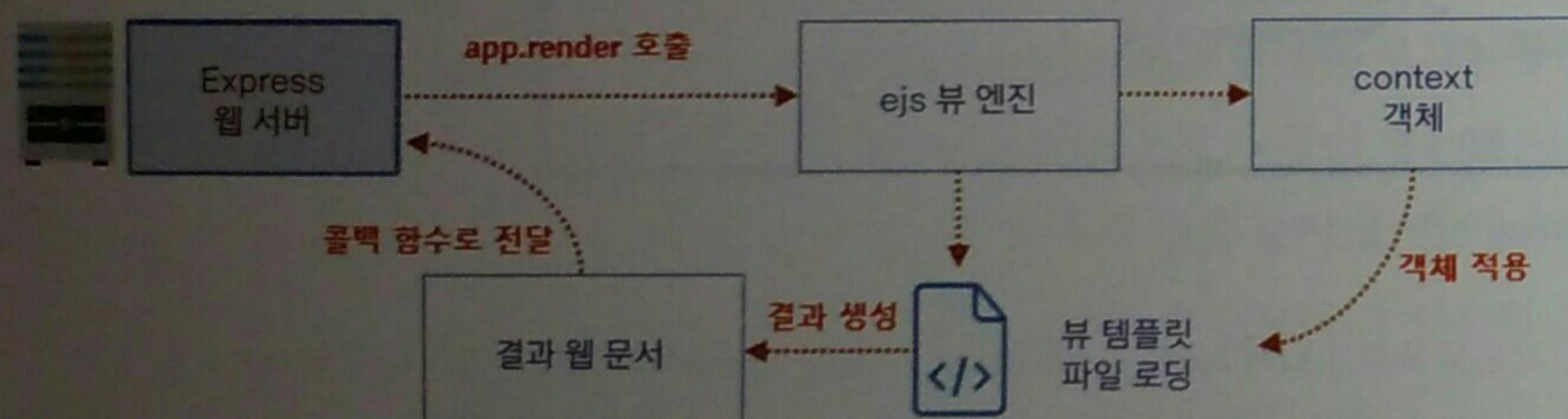
// 뷰 템플릿을 이용하여 렌더링한 후 전송
var context = {userid : paramId, username : username};
req.app.render('login_success', context, function(err, html) {
  if (err) {throw err;}
  console.log('rendered : ' + html);

  res.end(html);
});
.....

```

익스프레스 서버 객체인 app에는 render() 메소드가 들어 있습니다. 이 메소드를 호출하면 뷰 엔진이 템플릿 파일을 읽어 들인 후 파라미터로 전달한 context 객체의 속성으로 들어 있는 값들을 적용하고 그 결과를 콜백 함수로 돌려줍니다. 콜백 함수로 전달되는 html 파라미터에는 사용자가 보게 될 최종 웹 문서 코드가 들어 있습니다. 따라서 res.end() 메소드를 호출하면서 이 html 객체를 파라미터로 전달하면 클라이언트로 응답을 보내게 됩니다.

뷰 엔진이 뷰 템플릿 파일을 사용해서 결과 웹 문서를 만들어 내는 과정을 정리하면 다음과 같습니다.



▲ 뷰 엔진이 템플릿 파일로 결과 웹 문서를 만드는 과정

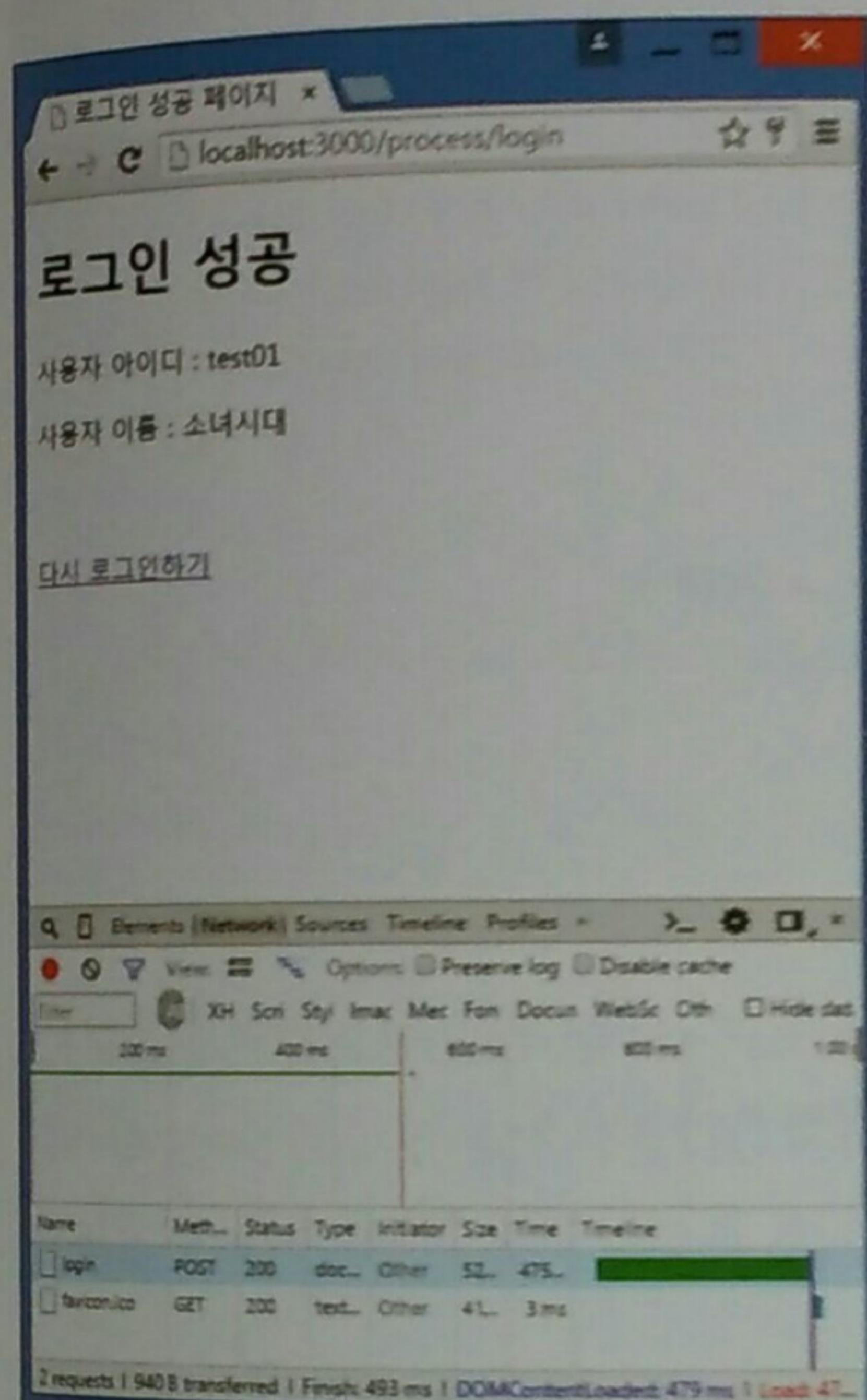
뷰 엔진은 뷰 템플릿 파일을 로딩한 후 context 객체의 속성들을 사용해 결과 웹 문서를 만들어 내므로 이 context 객체에 userid와 username 속성을 추가합니다. context 객체를 전달받는 템플릿 파일에서는 `<% = userid %>`와 `<% = username %>` 코드를 추가하여 변수에 들어 있는 문자열을 웹 문서에 출력합니다.

이제 뷰 엔진이 정상적으로 동작하는지 확인할 차례입니다. 먼저 명령 프롬프트를 열고 ejs 모듈을 설치합니다.

```
% npm install ejs --save
```

app.js 파일을 실행한 후 웹 브라우저에서 [public] 폴더의 login.html 파일을 엽니다. 사용자 아이디는 test01, 비밀번호는 123456을 입력하고 [로그인] 버튼을 클릭하면 뷰 엔진이 만들어 낸 웹 문서가 나타납니다.

▶ http://localhost:3000/public/login.htm



◀ 뷰 엔진이 만들어 낸 로그인 성공 페이지

로그인 성공 페이지는 이전에 보았던 것과 같지만 만들어진 과정은 다릅니다. 즉, 이전에는 코드에 태그를 직접 입력한 웹 문서가 응답으로 보낸 것이었지만 지금은 login_success.ejs 파일에 입력한 태그들이 표시된 것입니다. 서버의 콘솔 창을 보면 render() 메소드를 호출했을 때 뷰 템플릿으로부터 만들어진 결과 웹 문서의 코드를 확인할 수 있습니다.

```
Console Debug
ViewExample-app.js [Node Application] Node.js Process
rendered : <!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>로그인 성공 페이지</title>
  </head>
  <body>
    <h1>로그인 성공</h1>
    <div><p>사용자 아이디 : test01 </p></div>
    <div><p>사용자 이름 : 소녀시대 </p></div>
    <br><br><a href='/public/login.html'>다시 로그인하기</a>
  </body>
</html>
```

▲ 뷰 템플릿으로부터 만들어진 결과의 문서의 모습입니다.

뷰 템플릿으로 사용자 리스트 웹 문서 만들기

이번에는 사용자 리스트 요청에 대한 응답으로 보여 줄 웹 문서를 뷰 템플릿으로 만들어 보겠습니다.
user.js 파일 안에 들어 있는 listuser 함수의 내용은 다음과 같습니다.

참조 파일 | ViewExample>/routes/user.js

```
....  
  
var listuser = function(req, res) {  
  
    ....  
  
    database.UserModel.findAll(function(err, results) {  
  
        ....  
  
        res.writeHead('200', {'Content-Type' : 'text/html; charset = utf8'});  
        res.write('<h2>사용자 리스트</h2>');  
        res.write('<div><ul>');  
  
        for (var i = 0; i < results.length; i++) {  
            var curId = results[i]._doc.id;  
            var curName = results[i]._doc.name;  
            res.write('      <li>' + i + ' : ' + curId + ', ' + curName + '</li>');  
        }  
  
        res.write('</ul></div>');  
        res.end();  
  
    ....
```

데이터베이스에서 조회한 사용자 리스트는 results라는 배열 객체에 들어 있습니다. 따라서 forEach 또는 for문을 사용해 배열 요소를 확인할 수 있는데 여기에서는 for문을 사용하고 있습니다. 이 코드 중에서 사용자가 보게 될 웹 문서를 만들어 내는 부분을 삭제하고, 그 대신 [views] 폴더에 listuser.ejs 파일을 만든 후 다음과 같이 입력합니다.

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "UTF-8">
<title>사용자 리스트 페이지</title>
</head>
<body>
<h2>사용자 리스트</h2>
<div>
<ul>
<% for (var i = 0; i < results.length; i++) { %
    var curId = results[i]._doc.id;
    var curName = results[i]._doc.name; %>
    <li>#<% = i %> - 아이디 : <% = curId %>, 이름 : <% = curName %></li>
<% } %>
</ul>
</div>
<br><br><a href = '/public/listuser.html'>다시 요청하기</a>
</body>
</html>
```

ejs 템플릿 파일에서는 <% 와 %> 기호 사이에 자바스크립트 코드를 넣을 수 있습니다. 따라서 user.js 파일의 listuser 함수 안에서 사용되던 for 문 전체를 태그 사이로 옮긴 후 단순히 태그가 보이는 부분과 사용자 아이디와 사용자 이름을 출력해야 할 부분을 구분하여 입력합니다.

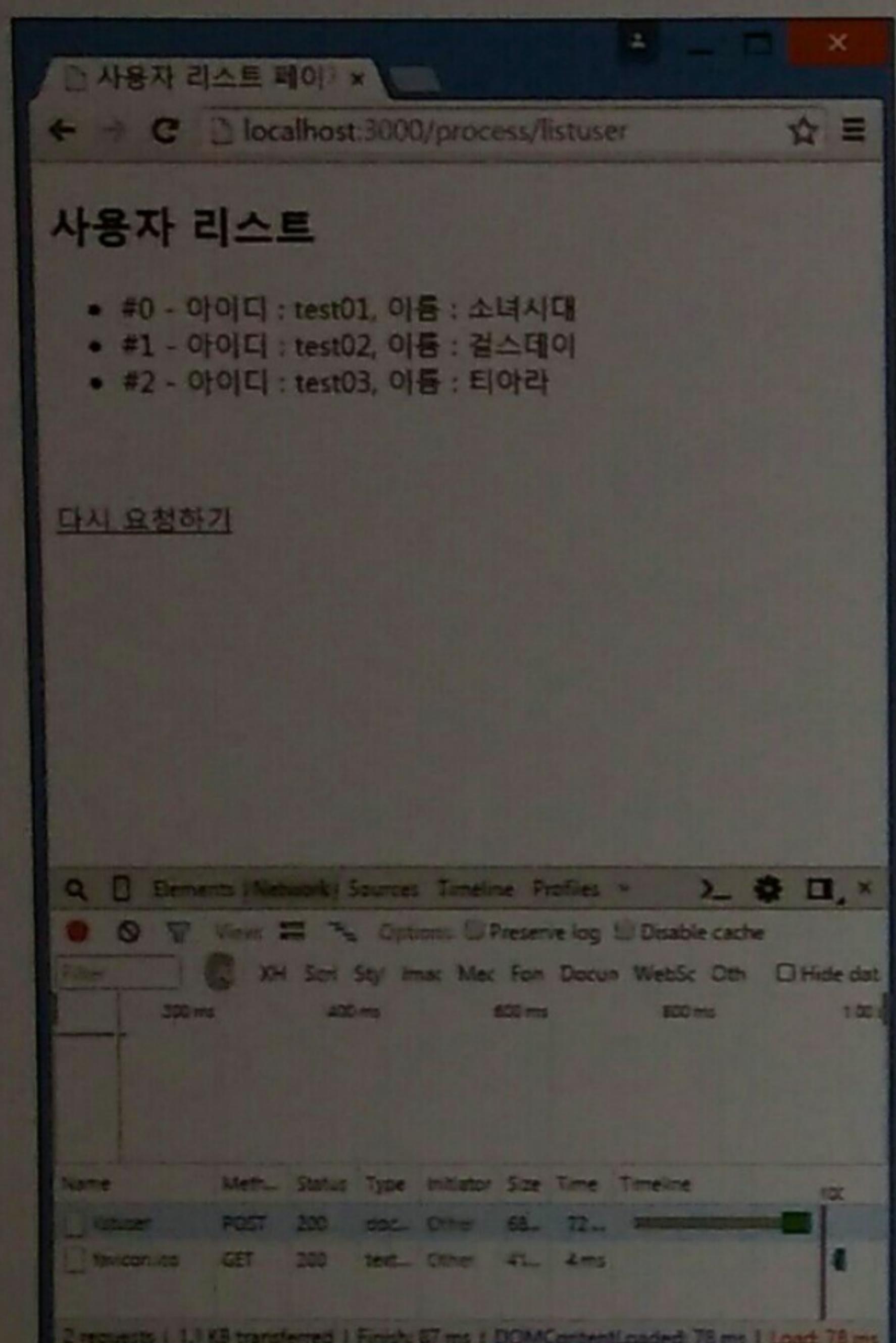
이렇게 만들어진 뷰 템플릿을 이용해 응답을 보내도록 user.js 파일을 수정합니다.

```
res.writeHead('200', {'Content-Type' : 'text/html; charset = utf8'});

// 뷰 템플릿을 이용하여 렌더링한 후 전송
var context = {results : results};
req.app.render('listuser', context, function(err, html) {
  if (err) {throw err;}
  res.end(html);
});
```

뷰 템플릿에 적용할 context 객체에는 사용자 리스트가 들어 있는 배열 객체를 results 속성 이름 그대로 넣어 줍니다. app.js 파일을 실행하고 listuser.html 파일을 연 후 [전송] 버튼을 누르면 응답 페이지가 나타납니다.

▶ http://localhost:3000/public/listuser.html



◀ 뷰 엔진이 만들어 낸 사용자 리스트 응답 페이지

응답 페이지는 listuser.ejs 파일을 뷰 템플릿으로 읽어 들인 후 필요한 값을 중간중간 넣어 만든 것이라고 이해하면 됩니다.

뷰 템플릿으로 사용자 추가 웹 문서 만들기

마지막으로 adduser 함수를 호출했을 때 사용자가 볼 수 있는 응답 페이지를 뷰 템플릿으로 만들어 보겠습니다. 이번에는 여러 개의 뷰 템플릿 파일에서 공통으로 사용되는 일부 내용을 또 다른 뷰 템플릿 파일로 만들었다가 삽입해서 사용하는 방법을 알아봅니다. 웹 문서에 들어가는 태그 중에서 <head> 태그는 대부분의 웹 문서에서 공통으로 사용되므로 별도의 ejs 파일로 만든 후 listuser.ejs 파일에서 읽어 들여 함께 보여 줍니다.

user.js 파일 안에 들어 있는 adduser 함수의 내용은 다음 코드와 같습니다.

```
.....
res.writeHead('200', {'Content-Type' : 'text/html; charset = utf8'});
res.write('<h2>사용자 추가 성공</h2>');
res.end();
....
```

응답 코드는 단 세 줄로 아주 단순하게 만들어져 있습니다. 이것을 adduser.ejs 템플릿 파일로 만들어 줍니다.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8">
    <title>헤드 부분 - ejs에서 include됨</title>
  </head>
  <body>
    <h2>% = title %</h2>

    <br><br><a href = '/public/login.html'>로그인으로 - ejs에서 include됨</a>
  </body>
</html>
```

<head> 태그와 </head> 태그 사이에 있는 헤더 내용은 다른 응답 페이지에서도 사용될 수 있는 공통 부분입니다. 이 경우에 공통 부분만 별도의 ejs 파일로 만들고 그 파일을 adduser.ejs 파일에 넣을 수 있습니다. <head> 태그 부분은 head.ejs 파일로 만듭니다. head.ejs 파일을 만들었으면 다음 코드를 입력합니다.

```
<head>
  <meta charset = "UTF-8">
  <title>헤드 부분 - ejs에서 include됨</title>
</head>
```

이 파일의 내용을 보면 <head> 태그 안에 들어 있던 태그만 따로 뺀 것임을 알 수 있습니다. <a> 태그 안에 들어 있는 내용도 여러 뷰 템플릿에서 사용될 수 있으므로 footer.ejs 파일을 만든 후 다음 코드를 입력합니다.

참조 파일 | ViewExample>/views/footer.ejs

```
<br><br><a href = '/public/login.html'>로그인으로 - ejs에서 include됨</a>
```

이 파일도 단순히 코드 일부분을 입력한 것입니다. adduser.ejs 파일을 다음과 같이 수정하여 분리한 파일을 삽입합니다.

참조 파일 | ViewExample>/views/adduser.ejs

```
<!DOCTYPE html>
<html>
  <% include ./head.ejs %>
  <body>
    <h2><% = title %></h2>

    <% include ./footer.ejs %>
  </body>
</html>
```

<% 기호와 %> 기호 사이에 include 키워드를 사용하면 별도로 분리되어 있는 ejs 파일을 삽입할 수 있습니다. 이 때 파일 이름은 상대 패스로 지정합니다.

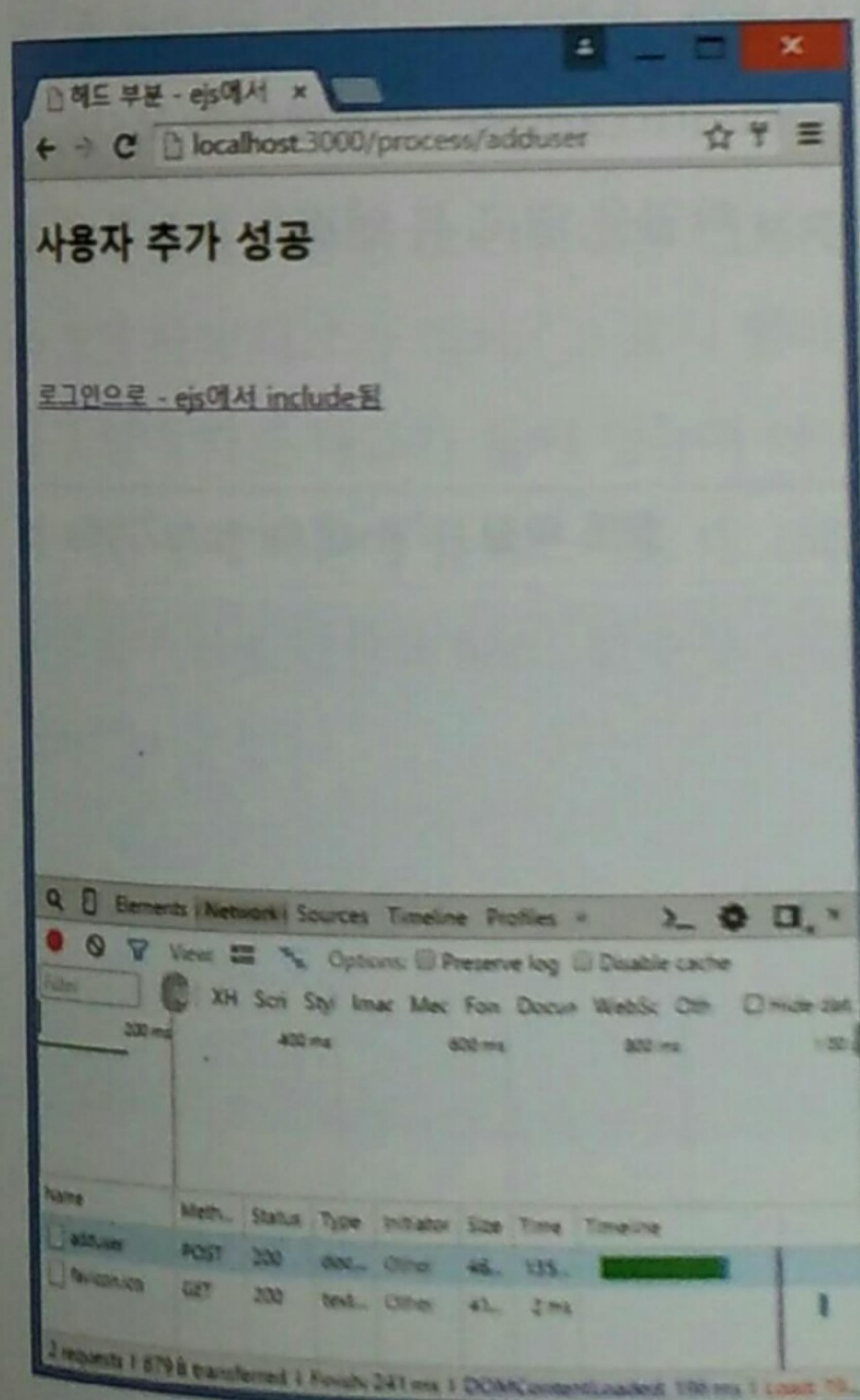
주의! include 키워드와 파일 이름 사이에 공백이 있다는 점에 주의하면서 입력합니다.

이제 뷰 템플릿으로 결과 웹 문서를 만들 수 있도록 user.js 파일을 열어 다음과 같이 수정합니다.

```
.....
res.writeHead('200', {'Content-Type' : 'text/html; charset = utf8'});
// 뷰 템플릿으로 렌더링한 후 전송
var context = {title : '사용자 추가 성공'};
req.app.render('adduser', context, function(err, html) {
  if (err) {throw err;}
  res.end(html);
});
.....
```

app.js 파일을 실행하고 adduser.html 파일을 연 후 사용자 정보를 추가하면 응답 페이지가 나타납니다.

▶ <http://localhost:3000/public/adduser.html>



◀ 뷰 엔진이 만들어 낸 사용자 추가 성공 응답 페이지

제목 부분과 아래쪽 링크는 별도로 분리되어 있는 ejss 파일에서 읽어와 추가했음을 아셨겠죠? 지금까지 ejss를 사용해 뷰 템플릿 파일을 만들고 ejss 뷰 엔진으로 변환한 후 응답으로 보내는 방법에 대해 알아보았습니다.

08-2 | jade 뷰 템플릿 사용하기

jade 포맷은 웹 문서의 태그를 그대로 사용하지 않고 최대한 간단한 형태로 입력하기 때문에 공백과 들여쓰기를 기준으로 태그의 구조가 결정됩니다. 따라서 jade 포맷을 사용하면 HTML 태그를 사용하는 것보다 훨씬 적은 내용을 입력해도 웹 문서를 만들 수 있습니다. jade를 사용할 때는 ejs와 마찬가지로 뷰 템플릿 파일을 먼저 만들고 jade 뷰 엔진에서 응답 웹 문서를 만들 때 사용합니다.

정박사의
한마디

들여쓰기를 잘못하면 jade가 제대로 처리되지 않아요

[Tab] 키를 사용해 구분하는 jade의 코드는 들여쓰기를 잘못하면 웹 문서가 제대로 만들어지지 않습니다. 때문에 들여쓰기 구조를 잘 생각해서 입력해야 합니다.

jade에 대한 자세한 내용은 다음 사이트를 참조하기 바랍니다.

▶ <http://jade-lang.com>

먼저 jade로 뷰 템플릿 문서를 만들고 뷰 엔진이 이 템플릿을 사용해 자동으로 만들어 낸 결과 웹 문서가 어떤 모양이 되는지를 확인해 보겠습니다. 예제 파일의 DefaultExample 프로젝트를 복사하여 ViewExample2 프로젝트를 만듭니다. 프로젝트 안에 있는 app.js 파일을 열고 뷰 엔진을 jade로 지정하면 jade로 만든 템플릿으로 응답을 보낼 수 있습니다.

참조 파일 | ViewExample2>app.js

```
app.set('views', __dirname + '/views');
app.set('view engine', 'jade');
```

jade로 HTML 문서 만들기

app 객체의 set() 메소드를 호출할 때 속성으로 이름으로 view engine을 전달하는 부분을 찾아 두 번째 파라미터의 값을 jade로 바꿔 줍니다. 그러면 jade 뷰 엔진이 설정됩니다. 이제 jade로 HTML 문서를 어떻게 만드는지 간단한 예제를 만들어 봅니다. [views] 폴더에 test1_success.jade 파일을 만들고 다음과 같이 입력합니다.

```
doctype html
html
  head
    title "성공"
  body
    block content
      #container
        p "조회에 성공했습니다."
```

jade는 공백(space)을 두 개 사용하여 들여쓰기를 하기 때문에 HTML 태그를 사용하는 웹 문서와 형태가 많이 다릅니다.

주의! Tab을 사용하여 들여쓰기를 할 때에는 Tab 하나당 공백 두 개만큼 들여쓰도록 설정하세요.

정박사의
한마디

공백과 Tab을 같이 사용하면 오류가 발생할 수 있어요

jade로 만드는 뷰 템플릿에서 공백과 Tab을 함께 사용하면 오류가 발생할 수 있습니다. 공백과 Tab 중 하나만 사용하기 바랍니다.

첫 번째 줄에 사용된 doctype은 <!DOCTYPE> 태그를 나타냅니다. 그 아래부터 시작되는 태그의 이름을 보면 시작 태그가 꺽쇠(<>) 표시 없이 입력되어 있습니다. 그런데 끝을 표시하는 태그가 없기 때문에 태그 안에 다른 태그가 들어 있는지 아니면 하나의 태그 다음에 태그가 있는 것인지 구별할 수 없습니다. 이런 문제 때문에 들여쓰기로 태그와 태그 사이의 관계를 구별합니다. 이제 [routes] 폴더 안에 test.js 모듈 파일을 만들고 test1 함수를 만듭니다. 이 함수는 클라이언트에서 테스트를 위해 보내온 요청을 처리합니다.

```

var test1 = function(req, res) {
    console.log('test 모듈 안에 있는 test1 호출됨.');

    res.writeHead(200, {'Content-Type' : 'text/html;charset = utf8'});

    // 뷰 템플릿을 이용하여 렌더링한 후 전송
    var context = {};
    req.app.render('test1_success', context, function(err, html) {
        if (err) {throw err;}
        console.log('rendered : ' + html);

        res.end(html);
    });
};

module.exports.test1 = test1;

```

지금 만든 test1 함수는 제일 간단한 형태의 코드를 가진 테스트용 함수입니다. 응답을 보내기 위해 test1_success.jade 파일을 템플릿으로 읽은 후 context 객체의 속성들을 적용하여 웹 문서를 만듭니다. 지금 읽어 들인 뷰 템플릿 파일에서는 변수를 사용하지 않으므로 context에는 아무런 속성도 넣지 않은 채 파라미터로 전달해도 됩니다. module.exports에 test1이라는 속성을 추가하고 test1 함수를 할당했으므로 이제 이 모듈을 불러들이는 쪽에서 test1 함수를 사용할 수 있습니다. config.js 파일에 새로 추가한 라우팅 정보를 추가합니다.

```

.....
route_info : [
    .....
    ,{file : './test', path : '/process/test1', method : 'test1', type : 'post'}
]
}

```

라우팅 정보는 route_info 속성에 배열로 추가되어 있습니다. 따라서 그 속성 값의 마지막에 새로운 한 줄을 추가합니다. 파일의 이름은 ./test로 하고 사용자가 요청할 패스는 /process/test1로 합니다. test.

js 모듈 파일을 불러왔을 때 사용할 수 있는 함수의 이름은 test1이므로 실행될 method 속성의 값을 test1로 합니다. 그리고 클라이언트가 요청할 때 사용할 방식은 post로 등록합니다.

마지막으로 사용자가 요청을 위해 사용할 수 있는 웹 문서를 [public] 폴더 안에 test1.html 파일로 만듭니다. 이 웹 문서는 버튼 하나가 들어 있는 가장 단순한 형태의 웹 문서입니다.

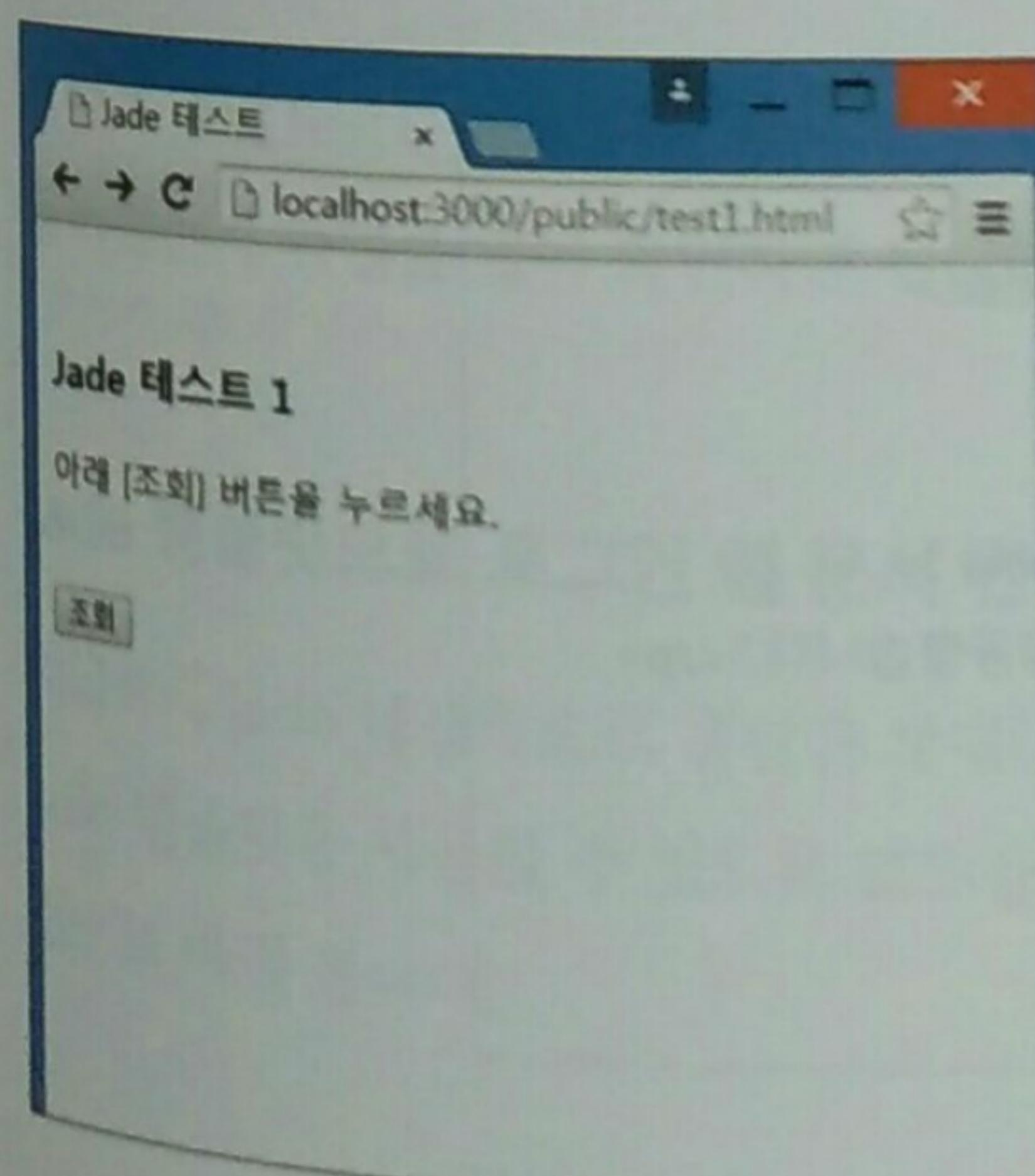
참조 파일 | ViewExample2>public>test1.html

```
.....  
<h3>Jade 테스트 1</h3>  
<p>아래 [조회] 버튼을 누르세요.</p>  
<br>  
<form id = "form1" method = "post" action = "/process/test1">  
  <input type = "submit" value = "조회" name = "">  
</form>  
<br>  
.....
```

클라이언트에서 /process/test1이라는 패스로 요청할 때 요청 파라미터를 넣을 필요가 없으므로 웹 브라우저를 열고 주소창에 요청 패스를 그대로 입력하는 것도 가능합니다. 하지만 여기에서는 일반적인 경우를 생각해서 클라이언트가 test1.html 웹 문서를 열고 버튼을 눌렀을 때 웹 서버에 요청하도록 만들었습니다.

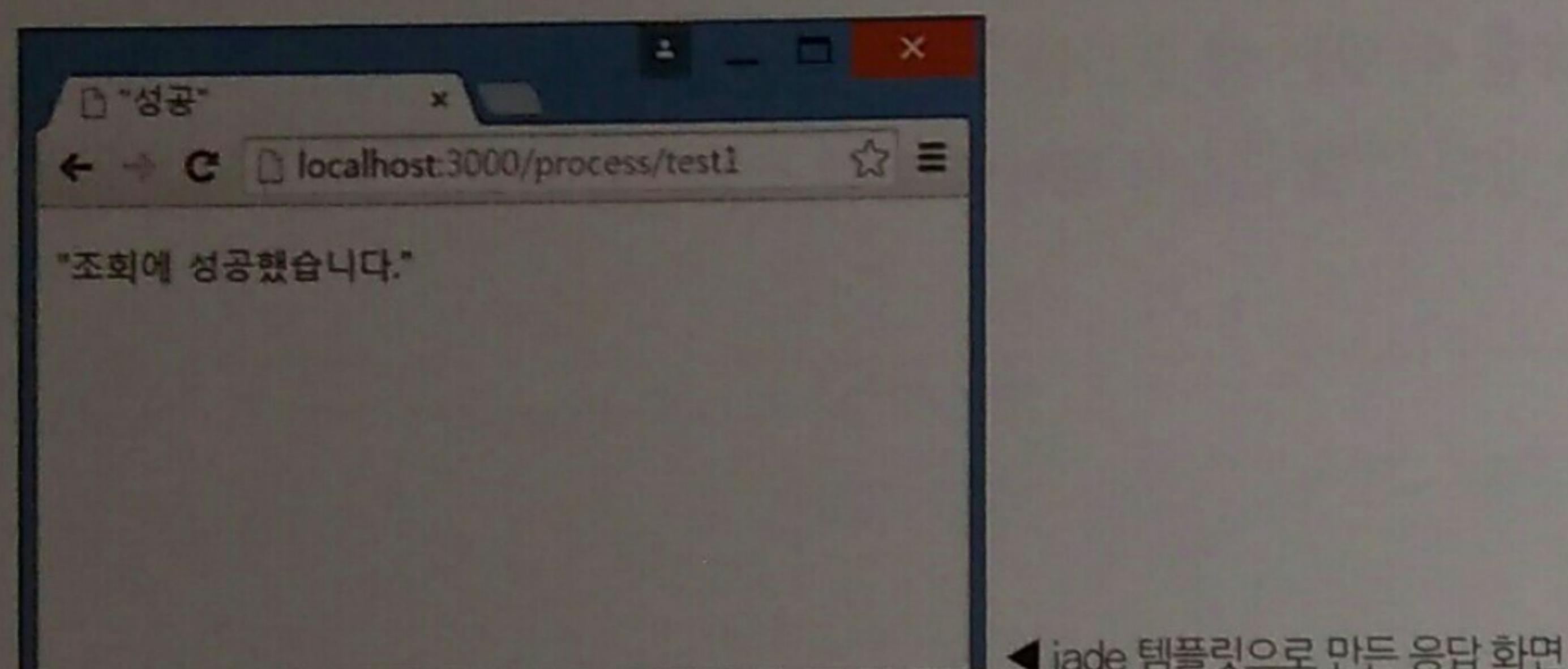
이제 app.js 파일을 실행한 후 웹 브라우저에서 test1.html 파일을 열어 봅니다.

▶ <http://localhost:3000/public/test1.html>

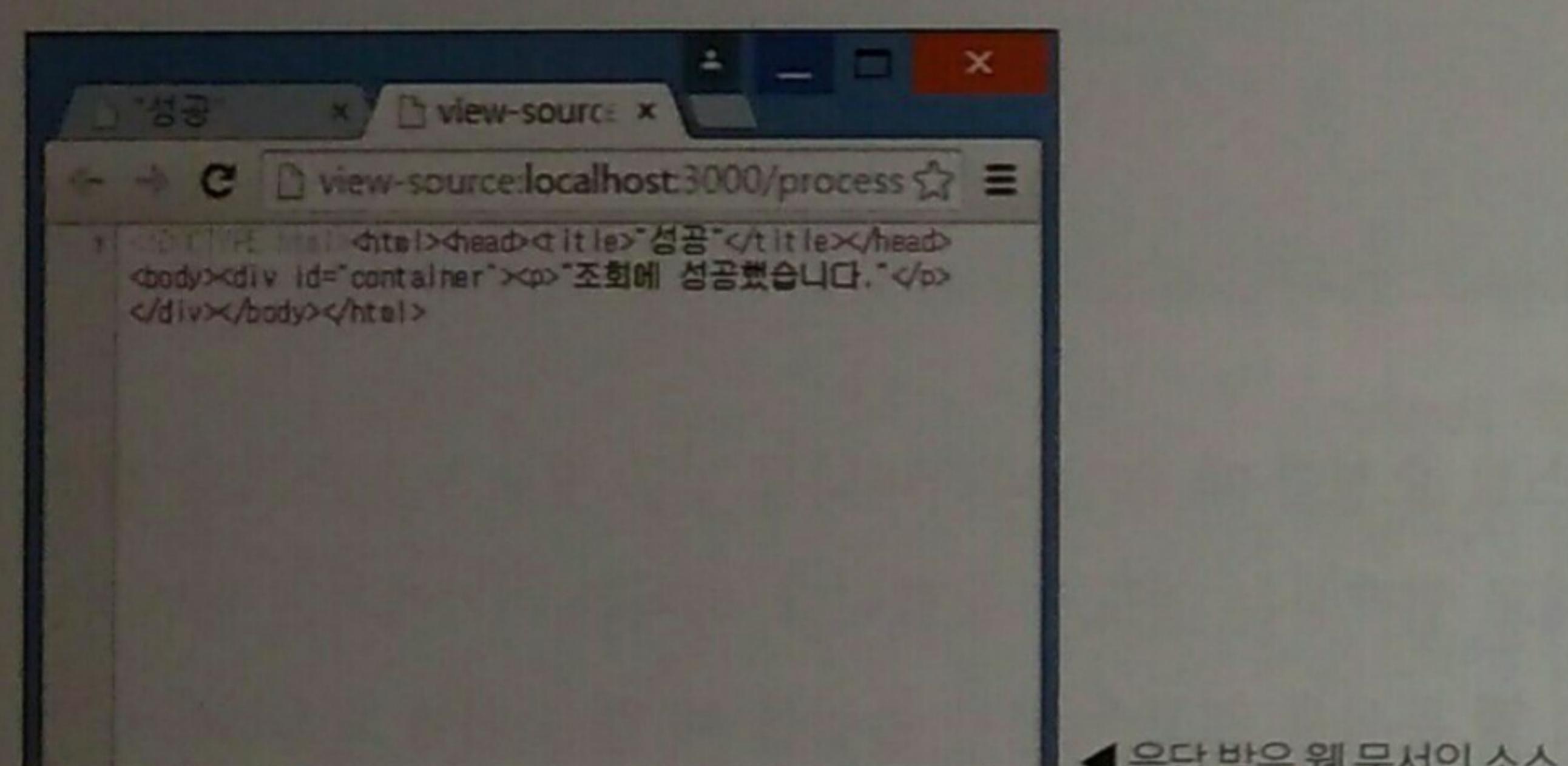


◀ 클라이언트가 요청할 때 사용할 웹 문서

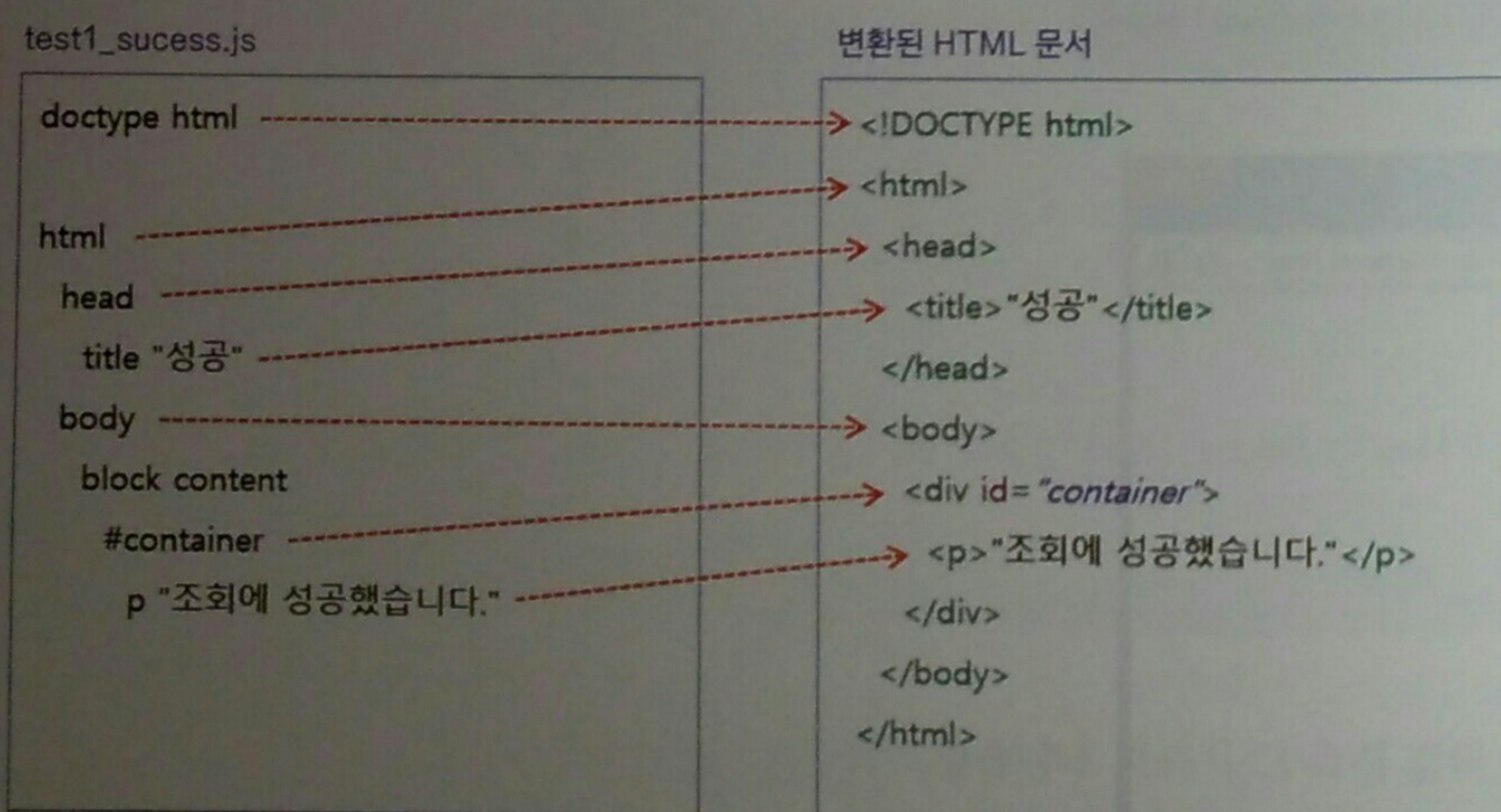
test1.html 파일을 띄웠을 때 보이는 웹 문서에서 [조회] 버튼을 누르면 test1_success.jade 파일로부터 만들어진 응답을 받을 수 있습니다.



정상적으로 응답이 온 것을 확인했으니 응답 웹 문서의 소스를 확인하여 웹 서버의 뷰 엔진에서 어떤 결과를 만들어 냈는지도 확인해 보겠습니다. 웹 문서 위에서 마우스 오른쪽 버튼을 누르면 [페이지 소스 보기] 메뉴가 보입니다. 이 메뉴를 누르면 응답으로 받은 웹 문서의 소스가 조회됩니다.



이제 jade 템플릿에 넣어둔 각각의 태그 이름이 HTML 문서의 태그로 어떻게 변환되는지에 대해 알 수 있을 것입니다.



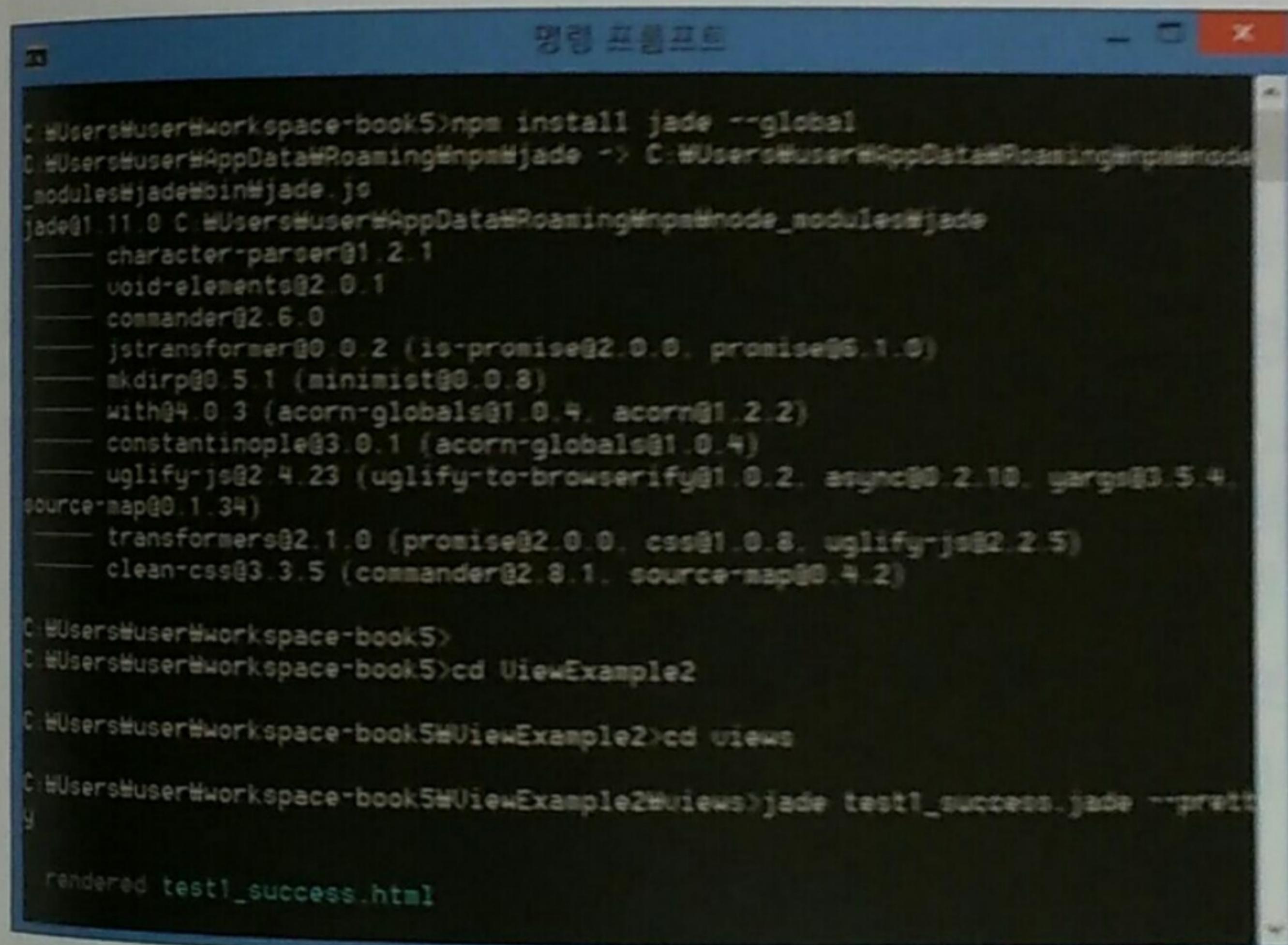
▲ 응답으로 받은 웹 문서의 소스

jade 템플릿과 응답으로 받은 웹 문서의 태그를 비교해 보면 jade 템플릿이 훨씬 간단해 보입니다. 특히 끝을 표시하는 태그가 없기 때문에 문서의 모양은 간단해 졌지만 앞에서 언급한 것처럼 공백이나 Tab을 잘못 입력하면 오류가 발생할 수 있으니 주의하세요. 만약 웹 서버에 적용하기 전에 여러분이 입력한 템플릿이 잘 만들어졌는지 확인하고 싶다면 명령 프롬프트에서 jade 명령을 사용해 결과 HTML 문서를 미리 변환해 볼 수도 있습니다. jade 명령을 사용하려면 먼저 jade 패키지가 --global 옵션으로 설치되어야 합니다. 명령 프롬프트를 열고 다음 명령을 입력합니다.

```
% npm install jade --global
```

설치가 완료되면 ViewExample2 프로젝트의 [views] 폴더로 이동한 후 다음과 같이 jade 명령을 입력합니다.

```
% jade test1_success.jade --pretty
```



```
C:\Users\user\workspace-book5>npm install jade --global
C:\Users\user\AppData\Roaming\npm\jade -> C:\Users\user\AppData\Roaming\npm\node_modules\jade
jade@1.11.0 C:\Users\user\AppData\Roaming\npm\node_modules\jade
+-- character-parser@1.2.1
+-- void-elements@2.0.1
+-- commander@2.6.0
+-- jstransformer@0.0.2 (is-promise@2.0.0, promise@6.1.0)
+-- mkdirp@0.5.1 (minimist@0.0.8)
+-- with@4.0.3 (acorn-globals@1.0.4, acorn@1.2.2)
+-- constantinople@3.0.1 (acorn-globals@1.0.4)
+-- uglify-js@2.4.23 (uglify-to-browserify@1.0.2, esnext@0.2.10, yargs@3.5.4,
source-map@0.1.34)
+-- transformers@2.1.0 (promise@2.0.0, css@1.0.3, uglify-js@2.2.5)
+-- clean-css@3.3.5 (commander@2.8.1, source-map@0.4.2)

C:\Users\user\workspace-book5>
C:\Users\user\workspace-book5>cd ViewExample2

C:\Users\user\workspace-book5\ViewExample2>cd views

C:\Users\user\workspace-book5\ViewExample2\views>jade test1_success.jade --pretty
rendered test1_success.html
```

▲ 명령 프롬프트에서 jade 명령을 사용하는 경우

이렇게 하면 test1_success.html 파일이 만들어집니다. 그 파일 안에 있는 태그들이 기대한 모양대로 만들어졌는지 파일을 열어 확인하면 됩니다.

jade 템플릿으로 로그인 웹 문서 만들기

지금까지 jade 템플릿으로 응답을 보내는 방법을 알아보았습니다. 이제 로그인에 대한 응답을 보낼 때도 jade 템플릿을 사용할 수 있도록 코드를 수정해 보겠습니다. 먼저 로그인 화면에 대한 응답을 jade 템플릿으로 바꿔 봅니다.

[views] 폴더에 login_success.jade 파일을 만들고 다음 코드를 입력합니다.

참조 파일 | ViewExample2>/views/login_success.jade

```
doctype html
html
  head
    meta(charset = 'utf8')
    title 로그인 성공 페이지
  body
    h1 로그인 성공
    div
      p 사용자 아이디 : #{userid}
    div
      p 사용자 이름 : #{username}
    br
    br
    a(href = '/public/login.html') 다시 로그인하기
```

위의 코드는 앞에서 만든 로그인 성공 응답 웹 문서를 jade 형식으로 바꾼 것인데, 사용자 아이디와 사용자 이름은 변수로 받아 넣어 줍니다. 변수를 넣고 싶다면 # 기호 뒤에 중괄호를 붙이고 그 안에 변수 이름을 입력합니다. 명령 프롬프트에서 jade 명령으로 login_success.jade 파일을 login_success.html 파일로 변환하여 오류가 있는지 미리 확인해 봅니다. 특히 공백 대신 탭을 넣었을 때 발생하는 오류는 미리 확인하여 수정하는 것이 좋습니다.

```
% jade login_success.jade --pretty
```

이제 이 템플릿 파일을 사용해 클라이언트에 응답을 보내도록 user.js 파일의 코드를 수정합니다. 이 코드는 앞에서 ejs로 템플릿을 만들었을 때 사용한 코드와 같지만, 뷰 엔진이 jade로 설정되어 있으므로 실제로 실행될 때는 jade 템플릿을 읽어 들입니다.

참조 파일 | ViewExample>/routes/user.js

```
.....
res.writeHead('200', {'Content-Type' : 'text/html; charset = utf8'});
// 뷰 템플릿으로 렌더링한 후 전송
var context = {userid : paramId, username : username};
req.app.render('login_success', context, function(err, html) {
```

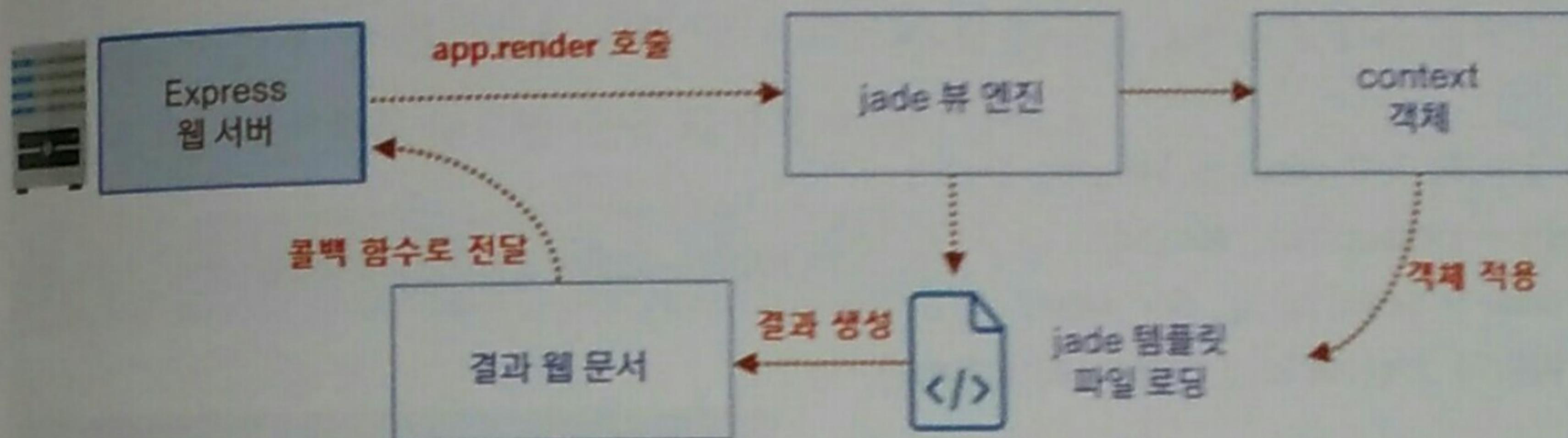
```

if (err) {throw err;}
console.log('rendered : ' + html);

res.end(html);
});
.....

```

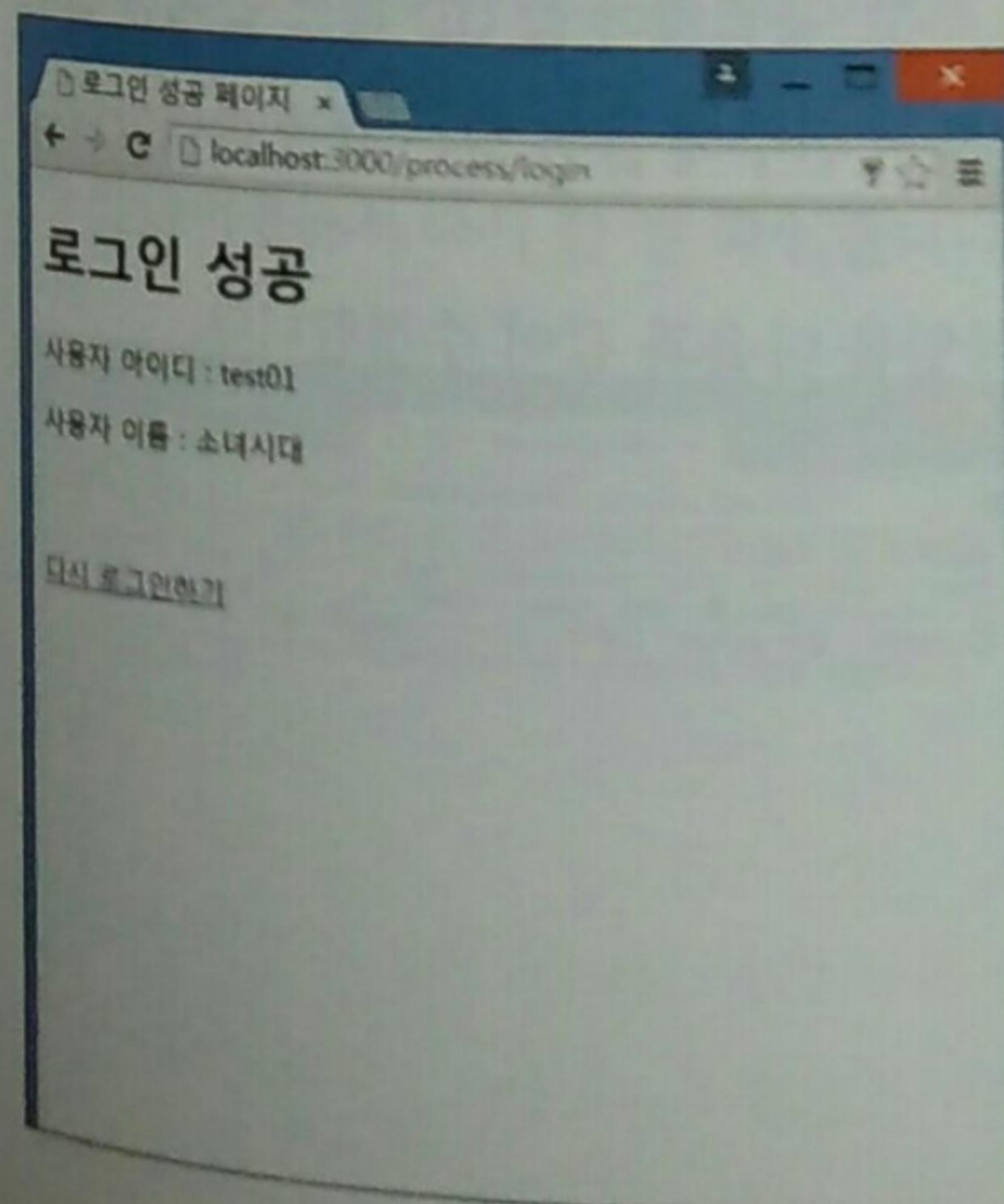
위의 코드는 앞에서 사용한 것과 같지만 render() 메소드는 login_success.jade 파일을 변환하여 결과 HTML 파일을 만들어 준다는 것을 이해해야 합니다. jade 템플릿으로 뷰 엔진이 결과 웹 문서를 만들어 내는 과정을 정리하면 다음과 같습니다.



▲ jade 템플릿 파일로 결과 웹 문서를 만드는 과정

jade 뷰 엔진은 jade 템플릿 파일을 로딩한 후 context 객체에 들어 있는 속성들로 결과 웹 문서를 만들어냅니다. 이제 사용자가 요청했을 때 정상적으로 동작하는지 확인할 차례입니다. app.js 파일을 실행한 후 웹 브라우저에서 login.html 파일을 엽니다. 사용자 아이디는 test01, 비밀번호는 123456을 입력하고 [전송] 버튼을 클릭하면 jade 템플릿으로 만들어진 웹 문서가 나타납니다.

▶ <http://localhost:3000/public/login.html>



◀ jade 뷰 엔진이 만들어 낸 로그인 성공 페이지

주의! 로그인 성공 페이지는 앞에서 본 것과 같지만 jade 템플릿 파일에 입력한 태그들이 변환되어 표시된다는 점에 주의하세요.

jade 템플릿으로 사용자 리스트 웹 문서 만들기

이번에는 사용자 리스트를 보여 줄 웹 문서를 jade 템플릿으로 만들어 보겠습니다. [views] 폴더에 listuser.jade 파일을 만들고 다음과 같이 입력합니다.

참조 파일 | ViewExample2>/views/listuser.jade

```
doctype html
html
  head
    meta(charset = 'utf8')
    title 사용자 리스트 페이지
  body
    h1 사용자 리스트
    div
      ul
        - for (var i = 0; i < results.length; i++) {
          - var curId = results[i]._doc.id;
          - var curName = results[i]._doc.name;
          li #{i} - 아이디 : #{curId}, 이름 : #{curName}
        -
      br
      br
      a(href = '/public/listuser.html') 다시 요청하기
```

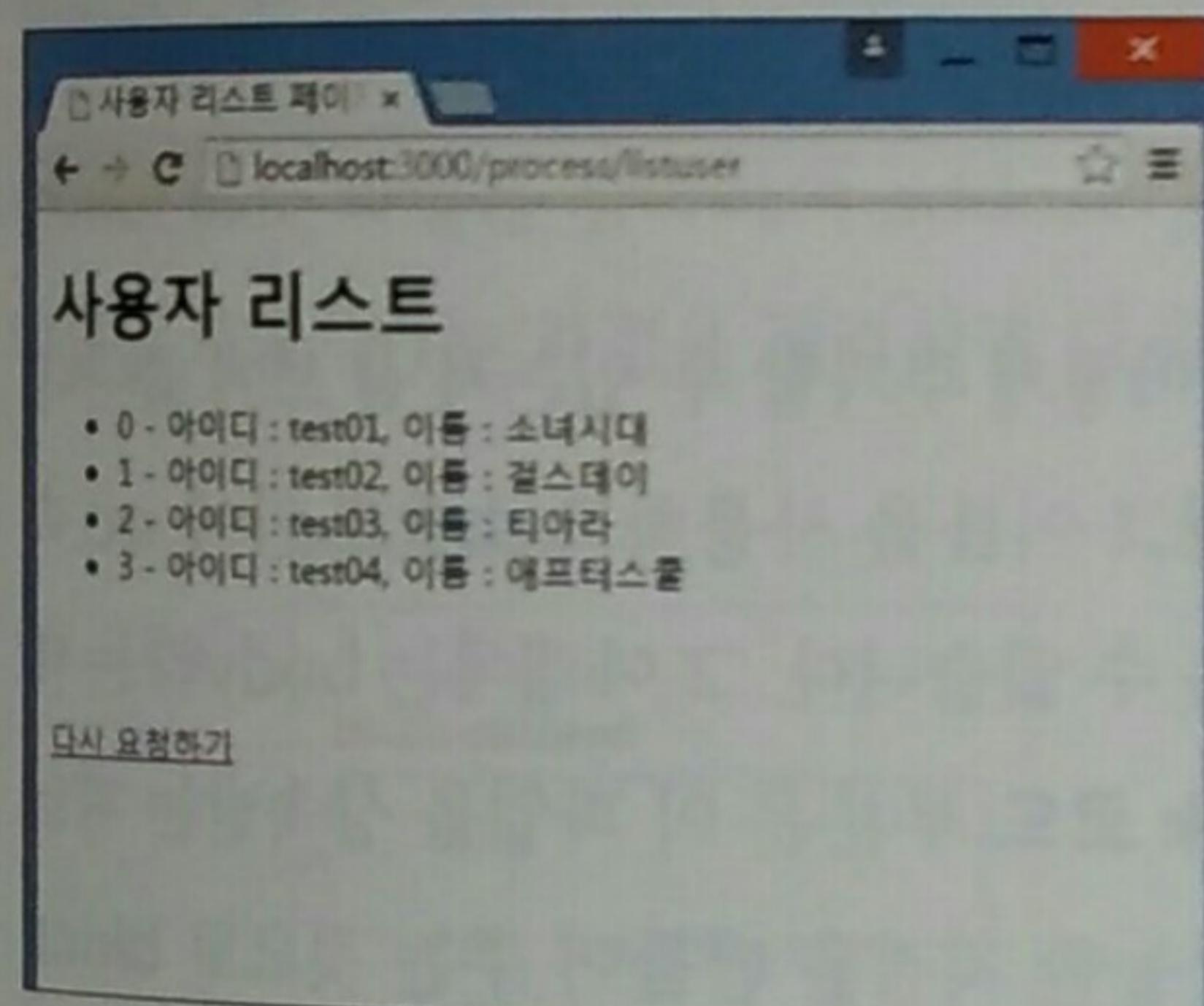
jade 템플릿 파일 안에서 자바스크립트 코드를 사용하려면 코드 줄의 가장 앞에 - 기호를 붙여 줍니다. 이 기호는 일종의 접두어로 그 뒤에 나오는 코드들은 HTML 파일에 들어가는 태그나 데이터로 변환하지 않습니다. 다시 말해, - 기호가 붙어 있는 부분은 웹 문서가 아닌 자바스크립트 코드로 인식되므로 이 코드의 실행 결과와 다른 내용이 합쳐져 최종 웹 문서가 됩니다. - 기호 뒤에 자바스크립트 코드를 넣어 줄 수 있으므로 for 문에 해당하는 코드 앞에는 모두 - 기호를 붙여 주고, 출력해야 할 태그 앞에만 붙이지 않습니다. 태그를 출력할 코드 뒤에는 #과 중괄호를 사용해서 변수에 들어 있는 값을 출력합니다.

이렇게 만들어진 뷰 템플릿을 이용해 응답을 보내도록 user.js 파일을 다음과 같이 수정합니다.

```
....,
res.writeHead('200', { 'Content-Type' : 'text/html; charset = utf8'});
// 뷰 템플릿으로 랜더링한 후 전송
var context = {results : results};
req.app.render('listuser', context, function(err, html) {
  res.end(html);
});
....,
```

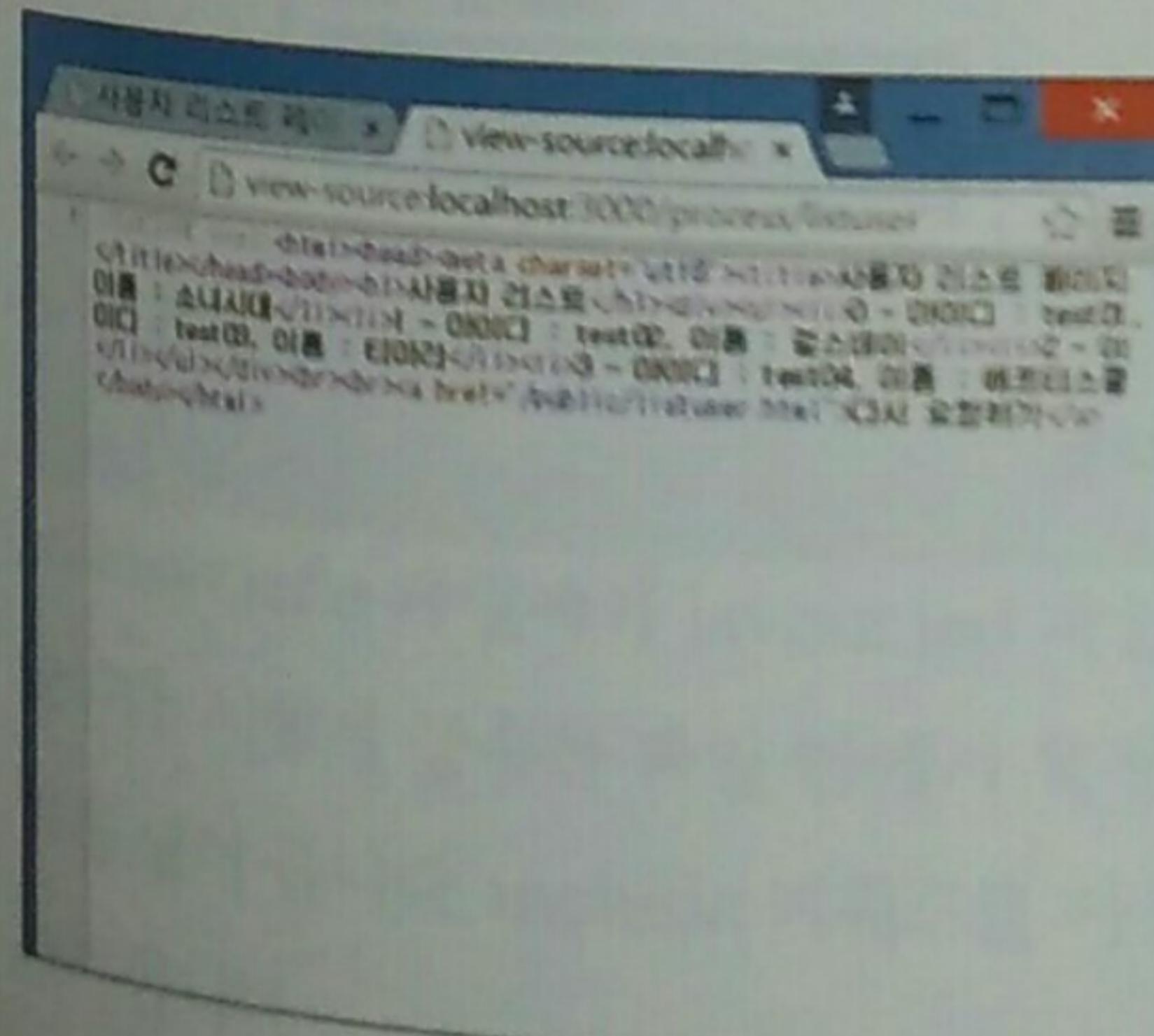
이 부분도 이전에 ejs 템플릿을 위해 만든 코드와 같습니다. app.js 파일을 실행하고 listuser.html 파일을 연 후 [전송] 버튼을 누르면 응답 페이지가 나타납니다.

▶ http://localhost:3000/public/listuser.html



◀ jade 뷰 엔진이 만들어 낸 사용자 리스트 응답 페이지

응답 페이지는 listuser.jade 뷰 템플릿 파일에서 만들어진 것입니다. 이 웹 문서에서 마우스 오른쪽 버튼을 누른 후 [페이지 소스 보기] 메뉴를 누르면 웹 문서의 태그를 확인할 수 있습니다. 원하는 태그 형태가 만들어졌는지 다시 한 번 확인합니다.



◀ 응답받은 웹 문서의 소스

jade 템플릿으로 사용자 추가 웹 문서 만들기

마지막으로 adduser 함수를 호출했을 때 사용자가 볼 수 있는 응답 페이지를 jade 템플릿으로 만들어 보겠습니다. 이번에는 대부분의 응답 문서에서 공통으로 사용하는 부분을 layout.jade라는 별도 파일로 만든 후 adduser.jade 파일에서 상속하도록 만들 것입니다. ‘상속한다’는 것은 layout.jade 파일의 내용을 바탕으로 adduser.jade 파일에서 일부 내용을 수정한다는 뜻입니다.

먼저 layout.jade 파일을 만들고 다음 코드를 입력합니다.

참조 파일 | ViewExample2>/views/layout.jade

```
doctype html
html
  head
    meta(charset = 'utf8')
    title extends로 상속함
    script(src = '/public/jquery-2.1.4.min.js')
  body
    block content
    include ./footer.jade
```

이 파일은 jade 템플릿에서 <head> 태그 안에 <script> 태그를 어떻게 추가할 수 있는지 잘 보여 줍니다. 외부 자바스크립트 파일을 링크하여 사용할 때는 script라는 태그 이름을 사용하고 소괄호 안에 src 속성을 추가합니다. 이렇게 하면 외부 자바스크립트 파일을 불러올 수 있습니다. 그 아래에는 body라는 태그 이름이 있고 하위 태그로 block이 추가되어 있습니다. block 코드 부분은 이 파일을 상속받는 파일에서 이 부분을 대체할 수 있다는 것을 알려 줍니다. 즉, 대체 가능한 지점을 만들어 주는 것으로 block 키워드 뒤에 나오는 것이 블록의 이름입니다. 그 아래에 보이는 include 키워드는 다른 jade 템플릿 파일을 읽어 와서 코드를 붙여 줍니다.

footer.jade 파일을 만들고 다음 코드를 입력합니다.

참조 파일 | ViewExample2>/views/footer.jade

```
div#footer
  a(href = '/public/login.html') 로그인으로 - jade에서 include됨
```

footer.jade 파일 안에서는 <div> 태그를 하나 추가하고 그 태그의 id 속성 값으로 footer라고 넣었습니다. 그 아래에는 <a> 태그를 사용해서 /public/login.html 페이지로 이동하였습니다. 이렇게 만든 footer.jade 파일의 내용은 layout.jade 파일에서 include 키워드를 사용해서 포함시킵니다.

이제 adduser.jade 파일을 만들고 다음 코드를 입력합니다.

참조 파일 | ViewExample2>/views/adduser.jade

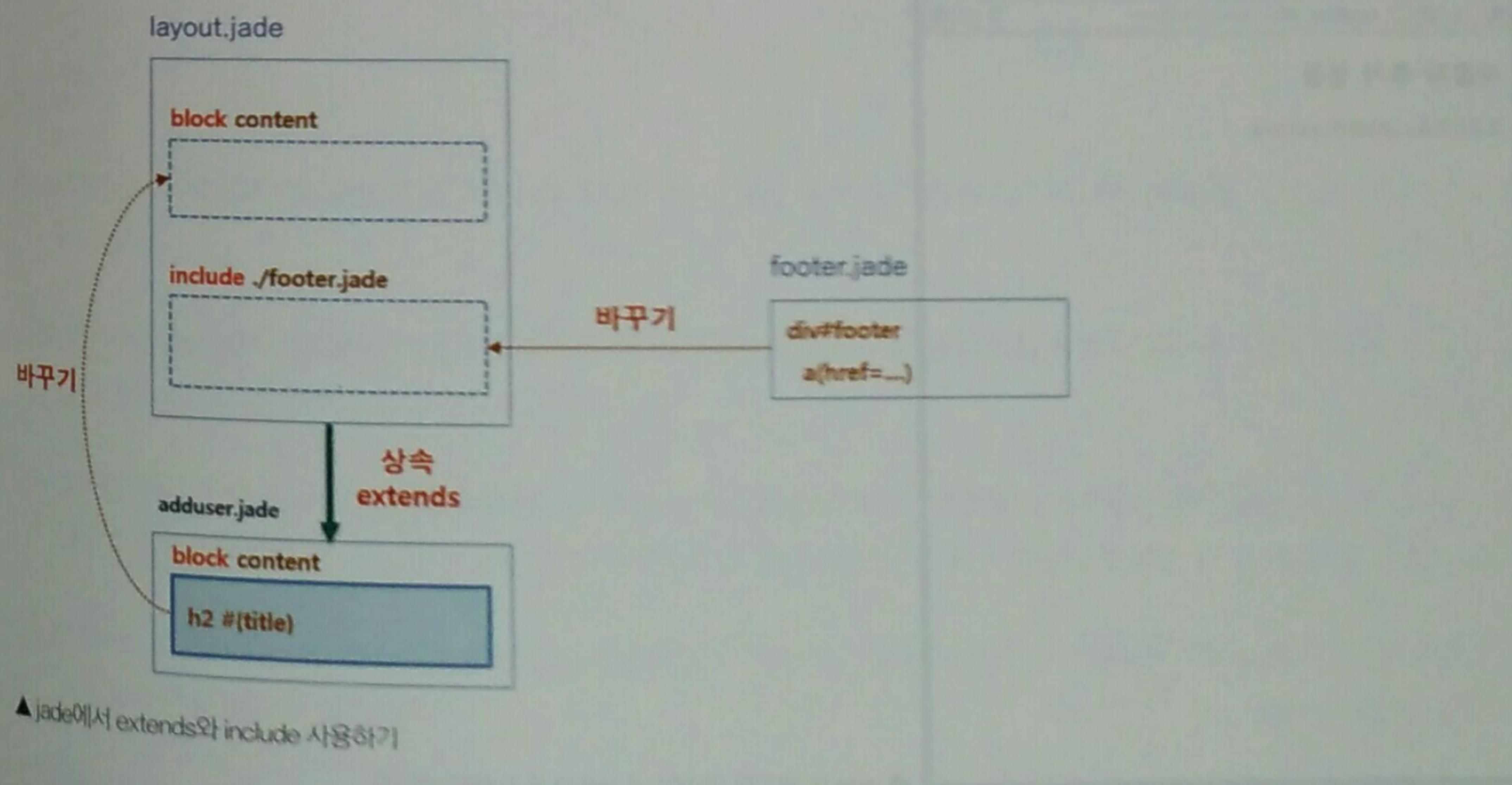
```
extends layout
```

```
block content
```

```
h2 #{title}
```

다른 jade 파일을 상속하여 새로운 jade 파일을 만들 때는 extends 키워드를 사용합니다. 이렇게 다른 파일의 내용을 상속받으면 block으로 지정했던 부분을 대체할 수 있습니다. layout.jade 파일에서 이름이 content인 block을 지정했으므로 여기에서도 block content 코드를 입력한 후 그 아래에 대체할 코드를 입력합니다. 대체할 코드는 <h2> 태그와 함께 title 변수 값을 출력합니다.

layout.jade 파일을 상속하여 adduser.jade 파일을 만드는 과정과 layout.jade 파일에서 footer.jade 파일을 포함시키는 과정을 정리하면 다음과 같습니다.



adduser.jade 파일에서 layout.jad 파일의 내용을 그대로 사용하려면 extends 키워드를 사용해 상속 합니다. 이렇게 상속한 파일 안에서 특정 영역에 있는 코드를 바꾸려면 block 키워드를 찾아 새로운 코드를 넣어 줍니다. include 키워드를 사용하여 다른 jade 파일을 지정하면 그 파일의 내용을 읽어 온 후 추가합니다.

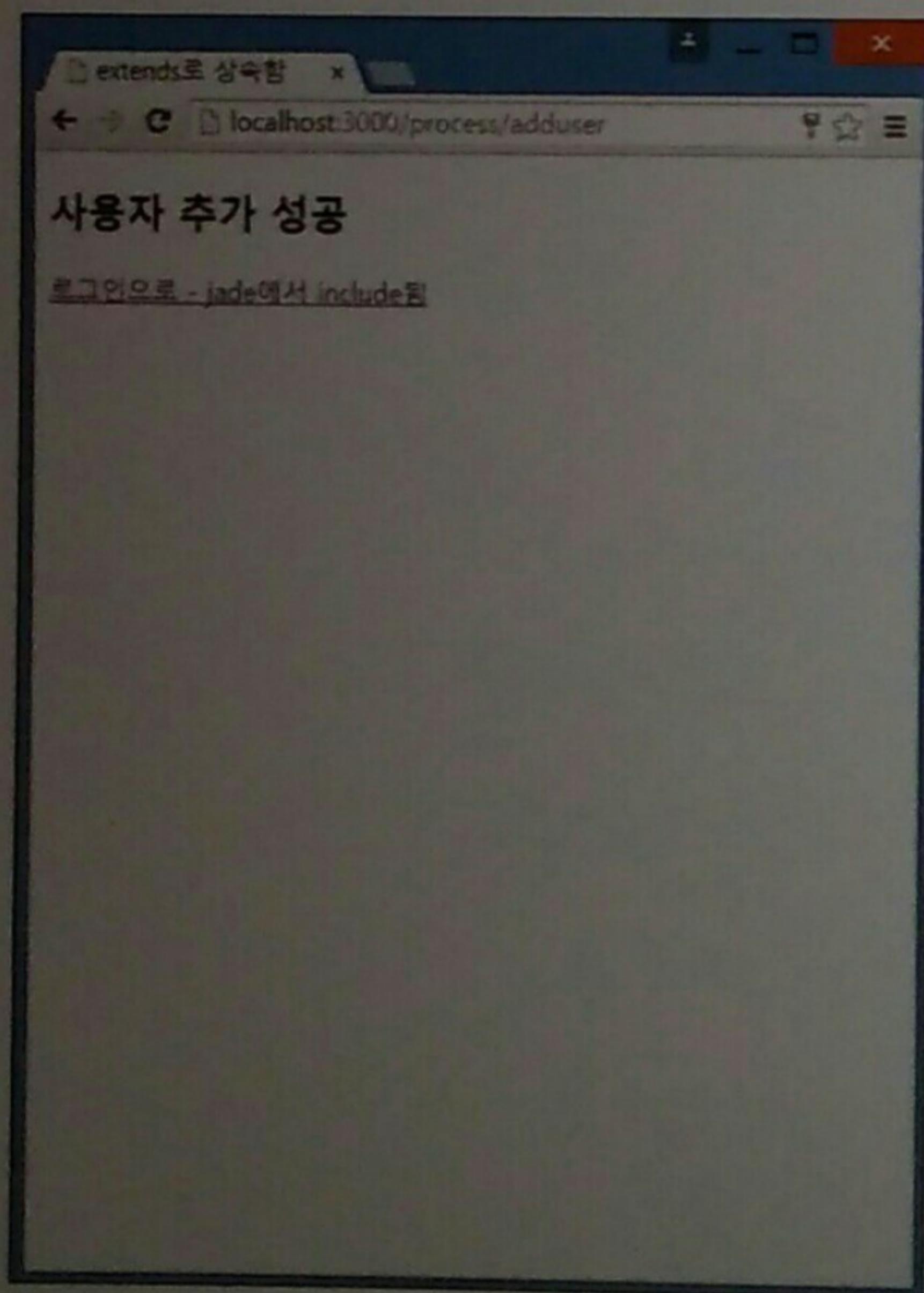
이제 마지막으로 adduser.jade 뷰 템플릿을 사용하도록 user.js 파일을 다음과 같이 수정합니다.

참조 파일 | ViewExample2>/routes/user.js

```
....  
res.writeHead('200', {'Content-Type' : 'text/html; charset = utf8'});  
  
// 뷰 템플릿으로 렌더링한 후 전송  
var context = {title : '사용자 추가 성공'};  
req.app.render('adduser', context, function(err, html) {  
    if(err) {throw err;}  
  
    res.end(html);  
});  
....
```

app.js 파일을 실행하고 adduser.html 파일을 연 후 사용자를 추가하면 응답 페이지가 나타납니다.

▶ <http://localhost:3000/public/adduser.html>



◀ jade 뷰 엔진이 만들어 낸 사용자 추가 응답 페이지

지금까지 jade 템플릿 파일을 만들고 jade 뷰 엔진을 사용해서 결과 웹 문서를 자동으로 만들어 낸 후 응답을 보내는 방법까지 알아보았습니다. 이렇게 ejs나 jade 뷰 템플릿을 사용하면 클라이언트에게 응답 보낼 웹 문서를 손쉽게 관리할 수 있고, 훨씬 간단하게 수정할 수 있습니다. 이제 여러분은 컨트롤러에 해당하는 라우팅 함수를 만들 수 있게 되었고 데이터베이스를 다루는 모델, 그리고 결과 웹 문서를 만들어 내는 뷰 엔진까지 다룰 수 있게 되었습니다. 이 세 가지를 반복 학습해서 충분히 이해하면 웹 서버에 기능을 추가하는 것이 아주 쉬워집니다.

다음 장부터는 웹 서버에 인증 기능을 추가하는 방법이나 웹 서버 외의 다양한 서버를 만드는 방법에 대해 본격적으로 다뤄 보겠습니다.

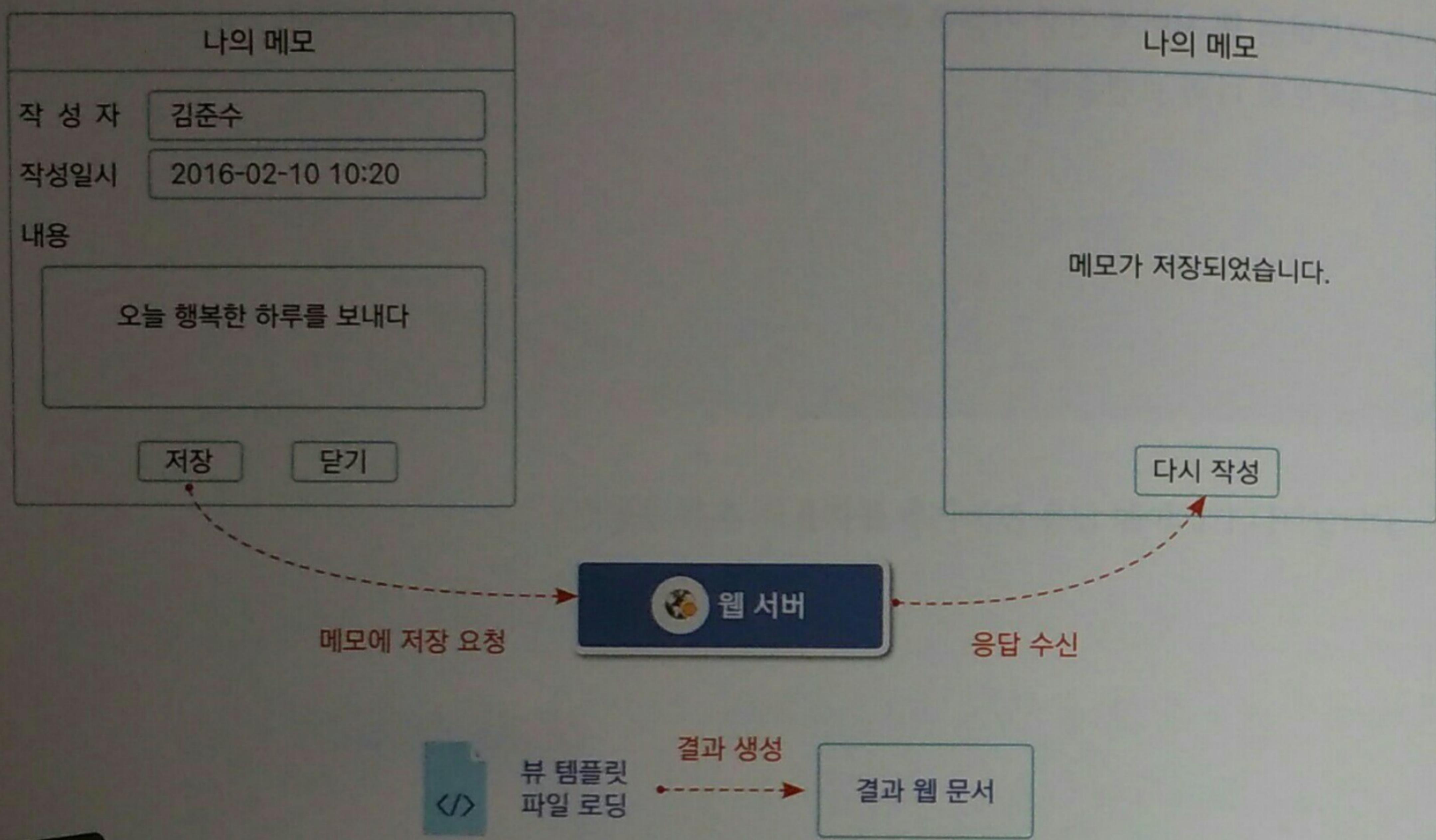
01. 07장에서 만든 Mission-07의 응답 화면을 ejs 뷰 템플릿에서 자동 생성하는 방법으로 바꾸어 보세요.

프로젝트 소스

Mission09

난이도

초급 ★★★★☆

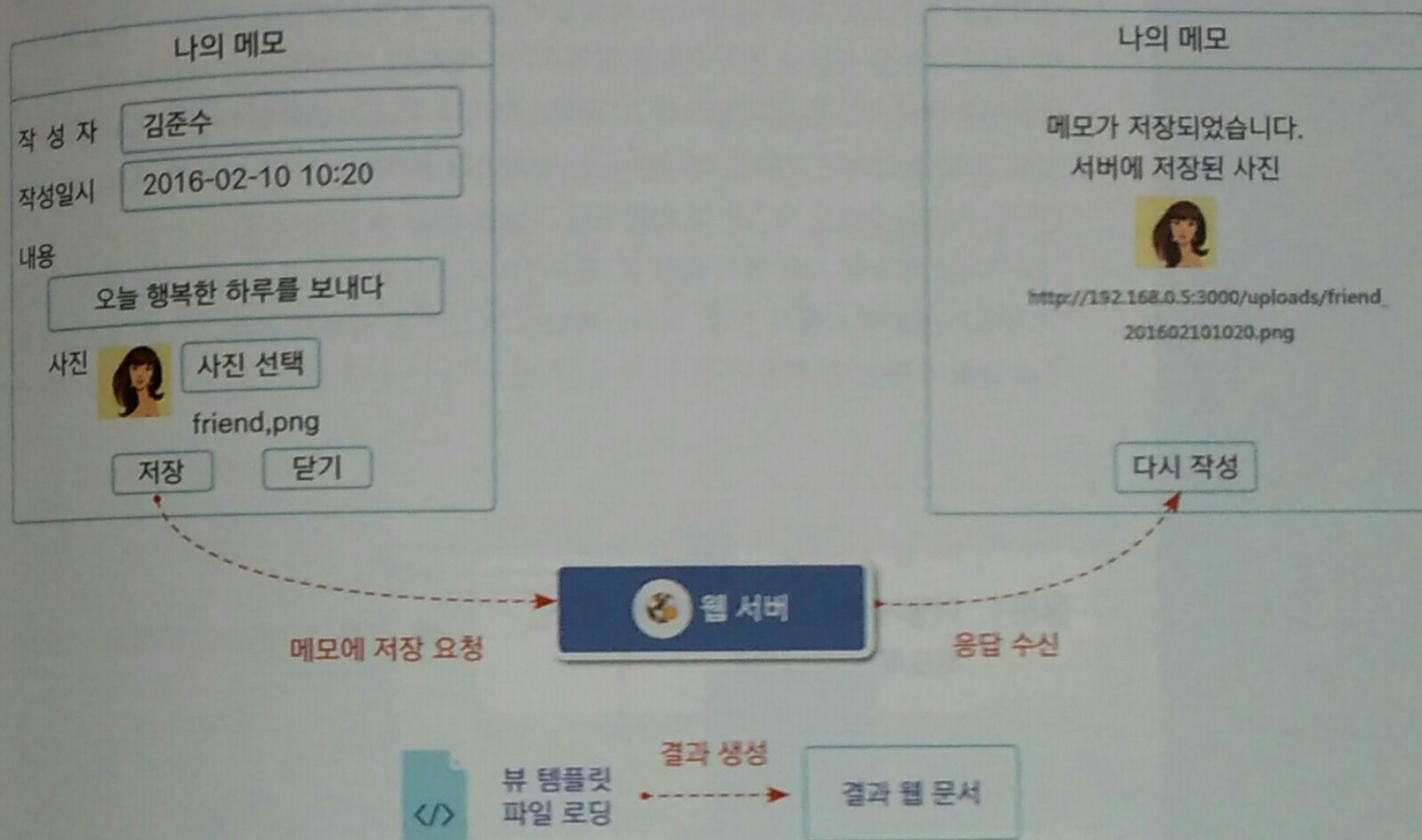


- Hint!**
- 작성 화면의 구성은 Mission-07과 같으므로 해당 소스 파일을 복사하여 새로운 파일을 만듭니다. 코드는 수정할 필요가 없습니다.
 - 작성 화면에서 [저장] 버튼을 누르면 웹 서버로 메모 내용을 보냅니다. 웹 서버에서 메모 내용을 저장하는 방식은 이전과 같습니다.
 - 데이터베이스에 저장한 후 정상적으로 저장되었다는 응답 메시지를 클라이언트로 보낼 때 ejs 뷰 템플릿을 적용합니다. 즉, 응답 화면을 위해 ejs 뷰 템플릿 파일을 만들어 적용합니다.
 - 클라이언트에서 응답 메시지를 받으면 응답 화면에 메시지를 보여 줍니다. 응답 화면의 구성도 Mission-07과 같으므로 수정할 필요가 없습니다.

MEMO

02. 07장에서 만든 Mission-08의 작성 화면과 응답 화면을 ejs 뷰 템플릿에서 자동 생성하는 방법으로 바꿔 보세요.

프로젝트 소스 Mission10 난이도 중급 ★★★☆☆



- Hint!**
- 작성 화면의 구성은 Mission-08과 같으므로 해당 소스 파일을 복사하여 새로운 파일을 만듭니다. 코드는 수정할 필요가 없습니다.
 - 작성 화면에서 [저장] 버튼을 누르면 웹 서버로 메모 내용을 보내고 사진 파일도 업로드합니다. 웹 서버에서 메모 내용과 사진 정보를 저장하는 방식은 이전과 같습니다.
 - 데이터베이스에 저장한 후 정상적으로 저장되었다는 응답 메시지를 클라이언트로 보낼 때 ejs 뷰 템플릿을 적용합니다. 즉, 응답 화면을 위해 ejs 뷰 템플릿 파일을 만들어 적용합니다.
 - 클라이언트에서 응답 메시지를 받으면 응답 화면에 메시지와 사진을 보여 줍니다. 응답 화면의 구성도 Mission-08과 같으므로 수정할 필요가 없습니다.

MEMO